Sztuczna inteligencja
Podręcznik do wykładów i ćwiczeń

Piotr Fulmański, Marta Grzanek

Piotr Fulmański[1]
Marta Grzanek[2]

Wydział Matematyki i Informatyki,
Uniwersytet Łódzki
Banacha 22, 90-238, Łódź
Polska

e-mail 1: fulmanp@math.uni.lodz.pl, 2: marta@math.uni.lodz.pl

Data ostaniej modyfikacji: **28 maja 2010**

Spis treści

Wprowadzenie

Sztuczna inteligencja jako nauka zrodziła się z dwóch potrzeb. Po pierwsze ogarniająca nas z dnia na dzień fala informacji wymaga stosowania coraz wymyślniejszych i skuteczniejszych metod ich przetwarzania. Klasyczne rozwiązania wywodzące się z metod matematycznych okazują się często niemożliwe do zastosowania ze względu na

Rosnącą złożoność obliczeniową.

Problemy podczas obliczeń numerycznych.

Złożonością procesu przekraczającą nasze obecne możliwości jego opisu.

Drugi powód, to jak zwykle u ludzi, chęć poznania czegoś nowego. Zapewne gdyby informatycy nie potrzebowali metod sztucznej inteligencji to i tak ktoś by się tym zajmował. Tym czasem okazało się, że naśladowanie przyrody i przetwarzanie danych zgodnie z niektórymi wypracowanymi przez nią mechanizmami daje szansę na przezwyciężenie problemów napotykanych przy stosowaniu metod klasycznych.

Jako pierwszy poddano analizie jeden z najdoskonalszych tworów przyrody przetwarzający dane – mózg. Tak narodziły się *sztuczne sieci neuronowe*. *Algorytmy genetyczne* to z kolei zauważenie faktu, iż w przyrodzie przeżywa najlepiej przystosowany. Mówiąc matematycznie – przeżywa osobnik maksymalizujący funkcję przystosowania. Natomiast *systemy rozmyte* stanowią odpowiedź na potrzebę przetwarzania danych nieprecyzyjnych typu *jadę szybko* (Co to znaczy *szybko*? 50 km/h, 100 km/h a może 72154 km/s???) czy *jest zimno* (*Zimno* to znaczy ile mamy stopni? -5, -54 czy może +10???). Te trzy zagadnienia stanowią dzisiejszy rdzeń tak zwanej *sztucznej inteligencji* i w wielu sytuacjach wspierają się wzajemnie. Słowo *sztucznej* sugeruje, że mamy do czynienia z czymś nienaturalnym, wymyślonym, co jednak „samo z siebie" potrafi zachować się inteligentnie. Tak naprawde to tylko ludzie, w ciągłym dążeniu do stworzenia maszyn na swój wzór i podobieństwo, przypisują inteligentne zachowanie tworom, które jedynie je symulują. Nie zmienia to faktu, iż jest to niezwykle ciekawy świat. Świat, który pochłania coraz bardziej w miarę jak się go poznaje. Daje nam jednocześnie potężne narzędzie możliwe do wykorzystania w różnych dziedzinach życia jak i pozwala nam poznać i zrozumieć lepiej samych siebie, bo przecież nas w pewnym stopniu symuluje.

Mamy głęboką nadzieję, że uda nam się pokazać Państwu choć część tego wspaniałego świata i zachęcić do samodzielnego podróżowania po nim. Niniejszy dokument stanowi elektroniczą wersję notatek jakie sporządzaliśmy dla siebie przez lata przygotowując się do zajęć obejmujących tą tematykę. Materiał tutaj zawarty niejednokrotnie może okazać się obszerniejszy od realizowanego na zajęciach, ale chcieliśmy zebrać w jedną całość informacje jakie mamy porozrzucane w najprzeróżniejszych miejscach. Integralną część Podręcznika stanowią pliki z przykładowymi programami, które można ściągnąć ze strony

W tekście pojawiają się także ćwiczenia. Odpowiedzi na niektóre z nich podane są na stronach następujących po stronie z pytaniem. Inaczej mówiac: najpierw jest pytanie a dopiero dalej są odpowiedzi. Dlatego zanim nie przejrzy się całego dokumentu proszę się nie niepokoić. Często odpowiedź nie jest w postaci jawnej. Ukryta jest bowiem pod postacią innych ćwiczeń. Ich wykonanie i własne przemyślenia dają dopiero rozwiązanie. Dokument ten postał głownie dla Państwa i od Państwa zależy jaki będzie w przyszłości. Dlatego proszę o zgłaszanie wszelkich uwag i propozycji, nawet tych związanych z błędami ortograficznymi czy wielkością czcionki. Życzymy wszystkim udanej i fascynującej podróży przez krainy Sztucznej Ineligencji

Piotr Fulmański, Marta Grzanek
Łódź, 29 czerwca, 2002

Zawartość podręcznika i uwagi

Naszym zamiarem nie było napisanie kolejnej książki o sztucznej inteligencji. Zależało nam natomiast aby zebrać w jednym miejscu wszystko to co do zajęć z tego zakresu może się przydać; stąd też w nazwie słowo „*podręcznik*". Nie tyle bowiem jest trudno wskazac pozycje warte przeczytania, co bardzo trudno jest w oparciu o nie stworzyć spójny materiał pozwalający na zagospodarowanie określonej ilości jednostek lekcyjnych. W większości książek brakuje odpowiednich „uzasadnień". Bardzo często wzory podawane są niejako z rękawa bez żadnych uzasadnień, ewentualnie z podaniem numeru cytowanej pozycji. Niestety większość przypadków cytowane pozycje są bardzo trudna do zdobycia a nawet jeśli się to już uda, to bez dodatkowych komentarzy często trudno w oryginalnych pracach doszukać się związku ze wzorami podawanymi przez cytującego autora. Dodatkową niedogodnością są różne notacje i przyjęte umowy co dla osoby stykającej się z zagadnieniem pierwszy raz jest dużym utrudnieniem.

Naszym celem jest podanie materiału w sposób jasny i zrozumiały a nie mądry i naukowy. Zdajemy sobie bowiem sprawę, że zwroty typu „łatwo widać" albo „oczywistą konsekwencją" lub całe strony wzorów bez słowa komentarza a co gorsza prostego przykładu ilustrującego zagadnienie znacznie redukują grono osób potencjalnie zainteresowanych tematem. Zdecydowanie jesteśmy zwolennikami podejścia do tematu typu: „popatrzcie jakie to proste" niż „popatrzcie jaki jestem mądry".

Ponieważ naszym zamierzeniem nie było zastpienie istniejącej literatury a raczej wypełnienie pewnej luki na rynku edukacyjnym dlatego w przypadku wątpliwości odsyłamy do literatury, której obszerny spis podany jest na końcu. Co ważniejsze spis ten zawiera pozycje „możliwe do osiągnięcia".

Cały podręcznik powstał na bazie notatek z różnych naszych zajęć mniej lub bardziej związanych ze sztuczną inteligencją. Stąd też dobór takiego a nie innego materiału i to w takich właśnie proporcjach, podzielonego dodatkowo na materiał zarówno ściśle wykładowy jak i typowo ćwiczeniowy. Postanowiliśmy utrzymać taki logiczny podział na niejako dwie części, dzięki czemu osoba zainteresowana wykładem wcale nie musi zaglądać do ćwiczeń i na odwrót. Niezależność (oczywiście w pewnym stopniu) każdej z części pozwala także na indywidualne przeglądanie jego zawartości w dowolnym porządku. W proponowanych ćwiczeniach chcemy pokazać, że wiedza „wykładowa" może znaleźć zastosowanie praktyczne.

I tak „Podręcznik" zawiera rozdziały poświęcone następującym dziedzinom sztucznej inteligencji:

Rozważania natury teoretyczno-filozoficznej. Materiał wykładowy znajduje się w rozdziale 1.

Strategie przeszukiwań przestrzeni. Materiał znajduje się w rozdziale 5. Jeśli to tylko było możliwe każdy algorytm był przedstawiony ogólnie i na konkretnych

przykładach. Tak więc materiał ten nadaje się zarówno na ćwiczenia jak i wykład.

Sztuczne sieci neuronowe. Materiał wykładowy znajduje się w rozdziałach 15 oraz 16. Materiał ćwiczeniowy znajduje się w rozdziałach:

- 24 – problem podziału przestrzeni.
- 25 – sieć jednowarstwowa, rozpoznawanie liter.
- 26 – sieć dwuwarstwowa, problem XOR.
- 27 – gra w czołgi, czyli Scorch.
- 28 – kompresja obrazu przy pomocy sieci jednokierunkowej.
- 29 – sieć Hopfielda, odtwarzanie zapamiętanego obrazu.
- 32 – sieć samoorganizująca, kompresja obrazu.
- **??** – metoda Levenberga-Marquardta.

Poniżej zebrano wszystkie metody nauki sztucznych sieci neuronowych, jakie omówione zostały w podręczniku.

- Reguła perceptronu – punkt 16.5.1 strona 138.

Algorytmy genetyczne. Materiał wykładowy znajduje się w rozdziale 39. Materiał ćwiczeniowy znajduje się w rozdziałach:

- 40 – magiczne kwadraty.
- 41 – zadanie optymalizacyjne.

Problem wiedzy i jej reprezentacji. Materiał wykładowy znajduje się w rozdziale 3 (Wiedza a język). Materiał ćwiczeniowy znajduje się w rozdziałach:

- brak

Podejmowanie decyzji, wnioskowanie. Materiał wykładowy znajduje się w rozdziale 3 (Wiedza a język). Materiał ćwiczeniowy znajduje się w rozdziałach:

- **??**, 12 – automaty Moore'a.
- 13 – drzewo decyzyjne.

Całość materiału z punktu „*Sztuczne sieci neuronowe*" pozwala na przeprowadzenie semestralnych zajęć (30 godzin wykładu + 30 godzin ćwiczeń) z zakresu sztucznych sieci neuronowych. Wybierając materiał z pozostałych punktów można przeprowadzić semestralne zajęcia z przedmiotu „*Elementy sztucznej inteligencji*".

Po wielu próbach udało nam się osiągnąć pewien względny porządek w oznaczeniach, ale prosimy o czujność i jednocześnie elastyczność myślową. Sztywne trzymanie się pewnych ustaleń, np. sposobu indeksowania, owocowało by koniecznością używania wielu indesków w przypadkach gdy jest to całkiem zbyteczne. Zwykle z konteksktu wynika czym dany indeks jest. I tak w jednym miejscu zapis p_i może oznaczać i-ty wzorzec uczący (i-ty wektor), natomiast gdzie indziej x_i oznacza i-tą składową wektora x. Ponad to ze względu na to, że często wektory są jednoelementowe, nie stosujemy specjalnego kroju czcionki na wyróżnienie wektora. Z tych samych powodów nie używamy znaku transpozycji. Przyjmujemy domyślnie, że wszystkie wymiary są tak dobrane aby było możliwe przeprowadzenie wymaganej operacji.

Poniżej przedstawiamy spis używanych symboli. Oczywiście z wymienionych powyżej powodów nie należy trzymać się go bardzo ściśle

- t

Rozdział 1

Sztuczna inteligencja

1.1 Zarys materiału

Niniejszy wykład ma charakter wybitnie interdyscyplinarny. Obejmuje bowiem swoim zasięgiem tematy i problemy ogólnie znane pod nazwą Sztucznej Inteligencji. Jak się dosyć szybko przekonamy, dziedzina ta ma swoje powiązania z logika, informatyką, matematyką, psychologią, filozofią, biologią, lingwistką, robotyką...

Ze względu na tak szeroki zakres poruszanych tematów znaczną trudnością jest takie przedstawienie wiadomości aby ułożyły się one w jakąś sensowną całość.

Całość zagadnień podzielona została na dwie grupy, które wzajemnie będą się przeplatac.

- Grupę pierwszą stanowią praktyczne przykłady zaczerpnięte z niektórych dziedzin wchodzących w skład Sztucznej Inteligencji wraz z omówieniem podstawowych zagadnień (np. sieci neuronowe czy modelowanie zachowań). Wykład odpowiedzialny będzie w tym przypadku za wprowadzenie pojęć i ich teoretyczną prezentację; implementacje pozostawione zostaną na ćwiczenia.

- Drugą grupę stanowią wiadomości ogólne czyli w pewnym sensie teoria:

  - jak doszło do powstania dziedziny nauki nazywanej dziś Sztuczną Inteligencją – prezentacja badań, prac i osiągnięć związanych z konstruowaniem myślących maszyn;
  - rozważania natury filozoficznej związane z, ogólnie mówiąc, myśleniem.

Wydaje nam się, iż istotnym jest aby podczas całego cyklu zajęć poszukiwać odpowiedzi na pewne pytania. Raczej nie zostaną one zamieszczone w sposób jawny, gdyż każdy sam musi wyrobić sobie swoje własne zdanie i mieć intuicję związaną z tym tematem. Oto przykładowe pytania:

Czy maszyna może myśleć? Pytanie wydaje się być postawione bardzo prosto, ale rodzi serię kolejnych pytań:

- Co to jest maszyna?
- Co to znaczy myśleć?
- Kiedy możemy powiedzieć, że „coś" myśli?

Jaka możliwa maszyna może myśleć? Jeśli stwierdzimy, że maszyna może myśleć to czy każda? Tak, nie? Czym się powinna charakteryzować?

Czy myśląca maszyna będzie równoznaczna ze sztucznym człowiekiem? Jest to pytanie o istotę człowieczeństwa. Co to znaczy być człowiekiem czy być podobnym do niego? Podobieństwo to jest funkcjonalne, strukturalne czy może jeszcze jakieś inne?

Dlaczego budujemy maszyny myślące?

- Budujemy aby dzięki temu zrozumieć istotę myślenia, istotę człowieczeństwa. A zatem w tym kontekście AI jest środkiem a nie celem. Środkiem służącym poznaniu nas samych.
- Zastąpią nas wszędzie tam gdzie mogło by być zagrożone życie ludzkie.
- Jeśli maszyny będą potrafiły myśleć, to możliwe, że będą to robiły na tyle sprawnie, że rozwiążą pewne problemy, których ludzie do tej pory nie rozwiązali.

Czy maszyna myśląca będzie posiadać prawa istoty myślącej?

Czy maszyna myśląca zaliczana będzie do „bytów" żywych? Konsekwencją obu powyzszych pytań są kolejne, bardziej szczegółowe:

- Czy można wyrzucić wyeksploatowaną maszynę? To tak jak gdyby wyrzucić kogoś z pracy tylko dlatego, że przekroczył pewien wiek. Zauważmy, że wyrzucenie maszyny ma znacznie szerwsze konsekwencje, bo zwykle jest równoznaczne z jej uśmierceniem.
- Problem zniewolenia maszyn i ich przymuszania do pracy.
- Jeśli maszyna będzie posadała prawa przysługujące bytom żywym, to czy może odmówić pracy ze względu na poglądy?

Oczywiście pytań podobnych do tych jest wiele. Każdy może uzupełnić tę listę o własne i poszukiwać odpowiedzi.

1.2 Sztuczna Inteligencja - czym jest?

Jednym z podstawowych dążeń człowieka jest pragnienie wiedzy i poznawania. Umiejętność dziwienia się od początku odróżniała go od innych istot żyjących. To właśnie owo nieustanne szukanie sedna i istoty rzeczy rodziło przez wieki postęp. Człowiek zawsze pragnął poznawać coraz lepiej i dokładniej. Te dążenia odzwierciedlają powstanie dziedziny nauki określanej mianem **Sztucznej Inteligencji**. Początkową ideą, leżącą u podstaw sztucznej inteligencji, było stworzenie myślących maszyn. Czy jest to możliwe? Na to pytanie nadal nie znamy odpowiedzi.

Powszechnie wiadomo, że komputery są w stanie rozwiązywać szereg różnych zadań szybciej i efektywniej niż ludzie. Jednakże najczęściej dostarczamy komputerowi wiedzy potrzebnej do rozwiązania danego problemu w postaci konkretnego algorytmu. **Wszędzie tam, gdzie znamy dokładny algorytm działania, prowadzący do wykonania interesującego nas zadania, inteligencja nie jest konieczna.** Jest jednak

wiele problemów, których rozwiązanie nie da się ująć w ścisłe reguły. Pojęcie tzw. „inteligentnego komputera" zakłada, że komputer potrafi efektywnie rozwiązać problem, nawet jeśli nie zostały zdefiniowane wszystkie kroki procesu rozwiązania.

Ze zbudowaniem myślącej maszyny wiązano wielkie nadzieje. Jednak w dziedzinie sztucznej inteligencji trudno znaleźć proste i ładne rozwiązania,czy wręcz „prawa inteligencji" na wzór praw fizyki czy chemii. Być może takie prawa w ogóle nie istnieją? Sam temat budowania sztucznej inteligencji jest dosyć kontrowersyjny. Pojawiły się opinie, że (sztucznej) inteligencji nie da się skonstruować, a istniejące systemy nie mają nic wspólnego z prawdziwą inteligencją, ponieważ wykonują jedynie operacje na liczbach i symbolach. Przy okazji powstało wiele pytań natury filozoficznej, wielu naukowców zaczęło frapować pytanie: „Czy maszyna może myśleć?" lub „Co mamy na myśli, gdy mówimy że program zachowuje się inteligentnie?" Tak naprawdę jednak problem leży chyba na poziomie definicji terminów „inteligencja (naturalna)" i „sztuczna inteligencja".

Dobrym punktem wyjścia w rozważaniach nad sztuczną inteligencją jest definicja inteligencji jako takiej. Ponieważ jednak inteligencja traktowana jako pewna własność czy cecha umysłu ma wiele odcieni znaczeniowych, zatem mamy wiele różnych definicji. Oto niektóre z nich:

Definicja 1.1 (Tiepłow)**.** *Inteligencja to właściwość psychiczna, która przejawia się we względnie stałej, charakterystycznej dla jednostki, efektywności wykonywania zadań.*

Definicja 1.2 (Stern)**.** *Inteligencja to ogólna zdolność adaptacji do nowych warunków i wykonywania nowych zadań.*

Definicja 1.3 (Piaget)**.** *Inteligencja to zdolność rozwiązywania problemów.*

Definicja 1.4 (Spearman)**.** *Inteligencja to dostrzeganie zależności, relacji.*

Definicja 1.5 (Ferguson)**.** *Inteligencja to zdolność uczenia się.*

Definicja 1.6 (Boring)**.** *Inteligencja to to, co mierzą testy inteligencji.*

Definicja 1.7. *Inteligencja to zdolność do aktywnego przetwarzania informacji, przekształcania ich z jednej formy w inną poprzez operacje logiczne.*

Definicja 1.8. *Inteligencja to zdolność do przetwarzania informacji na poziomie abstrakcyjnych idei (np. umiejętność dokonywania obliczeń matematycznych lub gry w szachy).*

Definicja 1.9. *Inteligencja to zdolność do twórczego, a nie tylko mechanicznego przetwarzania informacji, czyli tworzenia zupełnie nowych pojęć i ich nieoczekiwanych połączeń.*

Definicja 1.10. *Inteligencja to zespół zdolności umysłowych umożliwiających jednostce sprawne korzystanie z nabytej wiedzy oraz skuteczne zachowanie się wobec nowych zadań i sytuacji.*

Z punktu widzenia dalej prowadzonych rozważań najbardziej adekwatną dla nas definicją jest:

Definicja 1.11. *Inteligencja jest to zdolność uczenia się i rozumienia zjawisk poprzez doświadczenie, zdolność zdobywania wiedzy i wykorzystywania jej w celu szybkiego i efektywnego reagowania na nowe sytuacje; zdolność rozumowania w celu efektywnego rozwiązywania problemów.*

A zatem inteligencję charakteryzuje

- uczenie się ([...]*zdolność uczenia się*[...]),
- zdobywanie wiedzy ([...]*zdolność zdobywania wiedzy*[...]),
- wniosowanie ([...]*wykorzystywania jej*[wiedzy][...]*zdolność rozumowania*[...]),
- uogólnianie ([...]*reagowania na nowe sytuacje*[...]).

Skoro taka jest natura (naturalnej) inteligencja, to sztuczna inteligencja, jakkolwiek rozumiana, np. jako

- inteligencja udawana (gdy, stwarzamy pozory inteligentnego zachowania dzięki stosowaniu odpowiednich algorytmów – np. gra w szachy),
- inteligencja symulowana (gdy symulujemy procesy biochemiczne mające swój udział „w inteligencji"),
- inteligencja inna od naturalnej

także powinna charakteryzować się zdolnością

- zdobywania wiedzy,
- uczenia się,
- wniosowania,
- uogólniania.

Spróbujmy teraz określić czym jest sztuczna inteligencja. Nie należy poniższych definicji traktować jako ścisłych i jedynie słusznych. Jest to pewna propozycja, zapewne tak samo dobra jak wiele innych. Przyznać należy jednak, że właśnie w takich formach występuje to pojęcie najczęściej

Definicja 1.12. *Sztuczna Inteligencja jest dziedziną wiedzy której celem i przedmiotem badań są maszyny potrafiące rozwiązywać zadania, przy rozwiązywaniu których człowiek korzysta ze swojej inteligencji.*

Definicja 1.13. *[24] Artificial intelligence (AI) is the study of how to make computers do things which, at the moment, people do better.*

Sami badacze Sztucznaj Inteligencji dzielą się na dwa obozy, wedle propozycji i charakterystyki Johna Searle'a:

Silna Sztuczna Inteligencja (*ang. Strong Artificial Inteligence*) Zwolennicy tego obozu uważają, iż zbiór problemów stawianych przed nauką (jako taką, Sztuczną Inteligencją w szczególności) i rozwiązywalnych metodami naukowymi pokrywa się z klasą problemów rozstrzygalnych algorytmicznie, czyli rozwiązywalnych przez urządzenia pracujące na zasadzie maszyny Turinga. Mówiąc bardziej obrazowo, dla każdego zachowania lub akcji jaka wydarzy się w naszym otoczeniu można napisać program (klasycznymi metodami na klasyczny komputer), który to zachowanie lub akcję wyjaśni. Stąd często określa się ich mianem algorytmistów. Mówiąc inaczej, odpowiednio zaprogramowany komputer jest w istotny sposób równoważny mózgowi, ma stany poznawcze, jest umysłem a pewne cechy umysłowe można przypisać logicznemu działaniu dowolnego urządzenia liczącego.

Słaba Sztuczna Inteligencja (*ang. Weak Artificial Inteligence*) Zwolennicy tego poglądu są zdania, iż możliwe jest pełne poznanie badanych zagadnień, ale poznanie to nie będzie opierało się na procedurach algorytmicznych. Mówiąc inaczej komputer pozwala jedynie formułować i sprawdzać hipotezy dotyczące mózgu.

Z kolei Roger Penrose proponuje podział oparty o następujące stanowiska

1. Myślenie zawsze polega na obliczeniach, a w szczególności świadome doznania postają wskutek realizacji odpowiedniego procesu obliczeniowego (stanowisko to pokrywa się z SAI).

2. Świadomość jest cechą fizyczną działającego mózgu. O ile wszystkie fizyczne procesy można symulować obliczeniowo, to jednak symulacjom tym nie towarzyszy świadomość.

3. Odpowiedni procesy fizyczne w mózgu powodują powstanie świadomości, ale tych procesów nie można nawet symulować obliczeniowo (stanowisko to pokrywa się z WAI).

4. Świadomości nie można wyjaśnić w żaden fizyczny, obliczeniowy czy inny naukowy sposób (mistycyzm).

W calym wykładzie będzimy trzymać się następującej notacji: przez **Sztuczną Inteligencję** określać będziemy dziedzinę badań zaś przez **sztuczną inteligencję** przedmiot badań.

Zauważmy, że jeśli Sztuczna Inteligencja bada myślący sztuczny system poznawczy to czyni to w oparciu o znajomość systemu naturalnego stając się tym samym bliską **naukom kognitywnym** (*ang. Cognitive Science*). Nauki kognitywne są studium inteligencji i systemów inteligentnych ze szczególnym odniesieniem się do zachowania inteligentnego jako procesu dającego się obliczyć. Jest to określenie na interdyscyplinarny zbór nauk związanych ze zdobywaniem i używaniem wiedzy. Swój wkład wnoszą tutaj: psychologia, lingwistyka, filozofia, atropologia, nauki o mózgu, pedagogika, logika, matematyka, informatyka... Istnieje pięć głównych pól badawczych w naukach kognitywnych: reprezentacja wiedzy, język, uczenie się, myślenie, percepcja.

Mimo bliskości nauk kognitywnych i Sztucznej Inteligencji będziemy je rozgraniczać: nauki kognitywne za przedmiot swych badań obrały naturalne systemy poznawcze ze szczególnym uwzględnieniem roli człowieka, natomiast Sztuczna Inteligencja jedynie takie systemy modeluje.

1.3 Cele i zadania szucznej inteligencji

Można wyodrębnić dwa główne cele rozwoju Sztucznej Inteligencji:

- Budowa inteligentnych systemów (komputerowych), tzn. systemów wykazujących cechy podobne do cech inteligentnego działania człowieka, do skutecznego rozwiązywania trudnych zagadnień.

- Opracowanie obliczeniowej (algorytmicznej) teorii inteligencji.

Głównymi zadaniami sztucznej inteligencji są:

reprezentacja wiedzy , aby móc przyjmować pojawiające się informacje o świecie, rozumieć je oraz konfrontować z już posiadaną wiedzą,

wnioskowanie , aby wyciągać wnioski z pojawiających się informacji, i podejmować decyzje o dalszych działaniach,

uczenie się dla dostosowania się do nowo pojawiających się okoliczności, nieprzewidzianych przez twórców systemu, pojmowania nowych zjawisk, itp.,

rozumienie języka naturalnego aby można było praktycznie sprawdzić zdolności systemu sztucznej inteligencji,

posługiwanie się wizją w celu samodzielnego pozyskiwania wiedzy o świecie,

robotyka czyli praktyczna konstrukcja systemu zdolnego poruszać się i wykonywać działania w świecie rzeczywistym.

Osiągnięcie postawionych celów jest jeszcze odległe i zależy w znacznej mierze od wyboru narzędzi i właściwego podejścia do zagadnienia. Co rozumiemy przez „właściwe podejście" niech wyjaśni analogia maszyn myślących (jakkolwiek pojęcie *maszyny* byśmy rozumieli) do maszyn latających.

Zauważmy, że latanie możemy rozumieć i rozpatrywać na kilka sposobów. Najbardziej naturalne podejście polega na podpatrywaniu otaczającego nas świata i jego naśladowaniu. W ten oto sposób, wcześniej czy później, będziemy musieli dojść do wniosku, że istotą latania jest machanie skrzydłami pokrytymi albo jakąś błoną albo „piórami". W ten oto sposób, możemy poświęcić mnóstwo czasu na konstrukcję odpowiednich skrzydeł, błon i sztucznych piór. Czy jednak weryfikujący naszą teorię Ikar XXI wieku skacząc z wysokiej wieży faktycznie poleci?

Istnieje też drugi sposób – zrozumieć elementarne zasady którym podlega lot, bez ograniczania samych siebie żadnymi wstępnymi założeniami. To nic, że ludzka wersja obiektu latającego ma inaczej zbudowane i co istotne całkowicie nieruchome skrzydła. Ważne, że cel: maszyna latająca udało się osiągnąć. To podejście jest o tyle użyteczne, że oprócz latającej maszyny daje nam też zarówno pewien zestaw narzędzi jak i znacznie lepsze zrozumienie problemu i wiedzę leżące u podstaw zjawiska nazywanego lataniem. Dzięki temu możliwe jest stworzenie latających maszyn zupełnie nie nie przypominających istot żywych, np. helikoptery.

A zatem myśląca maszyna wcale nie musi być zbudowana na wzór i podobieństwo człowieka, tak jak samolot raczej nie jest podobny do ptaka. Pamiętajmy o tym, gdy pytamy „Czy komputer na prawdę myśli?" bo „Czy samolot na prawdę lata?".

1.4 Wiedza i jej reprezentowanie

Przede wszystkim należy zadać sobie dwa pytania: „Czym jest wiedza" oraz „Jak ją reprezentować?" Zgromadzenie odpowiedniej wiedzy i jej wewnętrzna reprezentacja jest podstawową sprawą dla wszystkich systemów mających się inteligentnie zachowywać. „*W AI podchodzi się do tego pragmatycznie: reprezentacja wiedzy to kombinacja struktur danych i procedur interpretacyjnych tak dobranych, że właściwie użyte, prowadzić będą do inteligentnego zachowania. Same struktury danych nie są jeszcze wiedzą, jak sama encyklopedia nią nie jest, potrzebny jest jeszcze czytelnik, interpretator*" [4].

Wyróżnić można następujące rodzaje wiedzy [4]:

Wiedza o obiektach i przedmiotach.

Wiedza o zdarzeniach , należy znać również następstwa przyczynowe i sekwencję czasową zdarzeń.

Umiejętności. Jest to typ wiedzy praktycznej, którą zdobywa się metodą prób i błędów i której nie da się przekazać w sposób werbalny

Meta-wiedza , czyli wiedza o samej wiedzy (wiem, że nic nie wiem), wynikająca z niedoskonałości naszej percepcji i metod pomiarowych, a więc z niepełnych danych, z oceny wiarygodności tych danych, z ograniczeń ludzkiej pamięci i zdolności do rozumowania.

Przekonania prawdziwe czy fałszywe, z góry powzięte nastawienia, wpływają na rozumowanie w oczywisty sposób. Mają różne stopnie pewności i trzeba je traktować jako pewien rodzaj wiedzy.

W używaniu wiedzy ważne są trzy aspekty:

Gromadzenie nowej wiedzy. Nie jest tylko akumulacją, dodawaniem nowych faktów, lecz wymagany jest proces tworzenia związków z już istniejącą wiedzą. Konieczna jest więc klasyfikacja wiedzy oraz interacja z wiedzą już posiadaną.

Wydobywanie z bazy wiedzy faktów, związanych z danym problemem. Dla czaowieka proces kojarzenia faktów jest dosyć naturalny, lecz jak kojarzyć w komputerowych programach? Można stosować łączenie lub grupowanie pewnych struktur danych, lecz wyklucza to większość nieprzewidzianych skojarzeń.

Rozumowanie przy użyciu tych faktów w poszukiwaniu rozwiązania. Występuje wtedy, gdy system ma zrobić coś, co nie jest w jawny sposób zaprogramowane. Możemy wyróżnić:

- **Rozumowanie formalne** – występuje w logice matematycznej i opiera się na ustalonych regułach wnioskowania.
- **Rozumowanie proceduralne** – oznaczające postępowanie wedaug jakieś instrukcji, procedury, a więc składa się z serii kroków. Jego odmianą jest budowanie modelu i symulacja komputerowa danej sytuacji.
- **Rozumowanie przez analogię** – często stosowane przez ludzi, lecz bardzo trudne dla programów komputerowych.
- **Rozumowanie przez uogólnienie** – bardzo powszechne w życiu codziennym i w matematyce, trudno jest jednak je zaprogramować.
- **Meta-rozumowanie** – związane jest z wiedzą o tym, co już wiemy (np. czy ja mogę to skądś wiedzieć?).

1.5 Dziedziny sztucznej inteligencji

//uzup//opisac kazde

Rozwiązywanie problemów i strategie przeszukiwań

Teoria gier

Automatyczne dowodzenie twierdzeń

Przetwarzanie języka naturalnego (w tym przetwarzanie mowy)

Sieci neuronowe

Systemy eksperckie

Algorytmy genetyczne

Robotyka

Uczenie się maszyn

Wyszukiwanie informacji (inteligentne bazy danych)

Programowanie automatyczne

Logika rozmyta

1.6 Test Turinga

1.6.1 Idea tesu

Test turinga – gra w naśladownictwo.

1.6.2 Zarzuty

Sprzeciw teologiczny

Zarzut teologiczny wyraża się w stwierdzeniu:

Dusza jest dana od Boga i przysługuje tylko człowiekowi.

Jednak powoływanie Boga na świadka i gwaranta zawsze musi być bardzo ostrożne aby nie popełnić nadużycia i aby nie wypowiadać się w kwestiach o których nie mamy pojęcia (bo niby skąd możemy wiedzieć jaki On jest, co może a czego nie itd). Sam Turing zarzut ten komentuje następująco:

Wydaje mi się, że cytowany wyżej argument pociąga za sobą poważne ograniczenie wszechpotęgi Boga Wszechmogącego. Przyznano, że istnieją pewne rzeczy, których On nie może zrobić, takie jak uczynienie jedności równą dwóm, ale czyż nie powinniśmy wierzyć, że może On obdarzyć duszą słonia, jeśli będzie uważał, że słoń jest tego godny? [...]

Podobny argument można sformułować w przypadku maszyn. Może on wydawać się inny, gdyż jest trudniejszy do „przełknięcia". Ale naprawdę oznacza on jedynie nasze przekonanie o mniejszym prawdopodobieństwie uważania przez Niego tych warunków materialnych za odpowiednie do obdarzenia duszą.

Jednakże jest to tylko spekulacja. Teologiczne argumenty nie wywierają na mnie głębokiego wrażenia [...] Gdyż [...] takie argumenty często bywały niewystarczające w przeszłości: w czasach Galileusza argumentowano, że teksty: „I słońce stało jeszcze... i nie spieszyło się zejść prawie przez cały dzień" (Jozue X. 13) i „Dał ziemi podstawę, tak, że nigdy nie powinna się ona ruszyć" (Psalm CV. 5) w sposób wystarczający zbijają teorię

Kopernika. Przy naszej obecnej wiedzy taki argument wydaje się bezwartościowy. Gdy ta wiedza nie była dostępna, to wywierało to zupełnie inne wrażenie.

Zarzut „lepiej o tym nie myśleć"

Zarzut teologiczny wyraża się w stwierdzeniu:

Konsekwencje myślenia maszyn byłyby zbyt okropne.
Miejmy nadzieję i wierzmy, że one nie mogą myśleć.

Wyrazem reprezentowania takiego stanowiska są wypowiedzi podobne do: *jeśli poważnie założyć jej* [sztucznej inteligencji] *realność [. . .] to nie sposób nie zauważyć swego rodzaju puszki Pandory przeróżnych trudności i niebezpieczeństw: społecznych, ekonomicznych, psychologicznych i jeszcze innych. najśmielsze odkrycia naukowe i wynalazki techniczne nie są w stanie dać nam niczego filozoficznie interesującego, gdy chodzi o tak zwaną sztuczną, czyli faktycznie alternatywną inteligencję. Natomiast mogłyby spowodować katastrofę humanitarną w stylu Georga'a Orwella.*

Zarzut wyjątkowości człowieka

Zarzut wyjątkowości człowieka wyraża się w stwierdzeniu:

Miło jest wierzyć, że człowiek jest lepszy od wszystkich innych istot[1]**.**

Zarówno ten jak i poprzedni argument sam Turing skomentował następująco:
Nie myśle, że ten argument jest wystarczająco poważny, aby trzeba go było zbijać.

Sprzeciw matematyczny

Zarzut świadomości

Zarzut niemożności

Zgadzam się z tobą, że możesz zrobić maszyny wykonujące to wszystko, o czym wspomniałeś, ale nigdy nie będziesz w stanie zrobić maszyny, która by zrobiła X.

kasparow i szachy

o indukcji

Nie wydaje mi się, aby prace i zwyczaje rodzaju ludzkiego stanowiły odpowiedni materiał, do którego można by stosować naukową indukcję. Bardzo dużą część czasoprzestrzeni trzeba by zbadać, aby móc otrzymać wiarygodne wyniki. Inaczej możemy (tak jak większość angielksich dzieci) rozstrzygnąć, ze każdy mówi po angielsku i że [w związku z tym] głupio jest uczyć się francuskiego.

Zarzut nieformalności zachowań

Problem sędziego

nasze skłonności czy nastawienie, przypisywanie komuś i nieprzypisywanie komuś innemu faktu posiadania inteligencji

[1] Lepszy w sensie posiadania unikalnych cech dających jemu dominującą pozycję.

Na zakończenie zbierzmy wszystkie dotychczasowe rozważania związane z testem Turinga i pojęciem maszyny myślącej w sensie Turinga w jedną spójną formę:

Definicja 1.14 (Maszyna myśląca w sensie Turinga)**.** *O maszynie myślącej w sensie Turinga powiemy, gdy:*

1. *będzie będzie ona zdolna do porozumiewania się z (dowolnym) człowiekiem za pomocą języka pisanego;*
2. *język musi być rozumiany i akceptowany przez obie zainteresowane strony (człowieka i maszynę);*
3. *rozumienie przez maszynę oznacza generowanie „sensownych" zdań jako „odpowiedź" na poprzedzające je z zdania (zarówno własne jak i człowieka).*

Żadna inna funkcja przypisywana człowiekowi nie musi być spełnona.

1.7 Dwa różne pokoje

1.7.1 Chiński Pokój

Chiński pokój, argument chińskiego pokoju – eksperyment myślowy zaproponowany przez amerykańskiego filozofa i językoznawcę Johna Searle'a i przedstawiony w jego pracy z 1980 roku pt. Minds, Brains, and Programs, mający pokazać, że nawet efektywne symulacje komputerowe nie urzeczywistniają prawdziwego rozumu, odkąd wykonywanie różnorodnych zadań (np. obliczeniowych) nie musi opierać się na rozumieniu ich przez wykonawcę. Służy on przeciwnikom teorii tzw. mocnej sztucznej inteligencji jako kontrargument. U podstaw eksperymentu stoi niezgodność między syntaksą a semantyką.

Od opublikowania pracy Searle'a argument chińskiego pokoju był głównym punktem debaty nad możliwością mocnej sztucznej inteligencji. Zwolennicy teorii mocnej sztucznej inteligencji wierzą, że właściwie zaprogramowany komputer nie jest prostą symulacją lub modelem umysłu, lecz liczy w sposób właściwy umysłowi, tzn. rozumie, ma stany kognitywne i może myśleć. Argument Searle'a (precyzyjniej – eksperyment myślowy) zamierzony w celu podminowania tego stanowiska idzie w sposób następujący:

Załóżmy, że wiele lat temu skonstruowaliśmy komputer, który zachowuje się jakby rozumiał język chiński. Innymi słowy, komputer bierze chińskie znaki jako podstawę wejściową i śledzi zbiór reguł nimi rządzący (jak wszystkie komputery), koreluje je z innymi chińskimi znakami, które prezentuje jako informację wyjściową.

Załóżmy, że ten komputer wykonuje to zadanie w sposób tak przekonujący, że łatwo przechodzi test Turinga, tzn. przekonuje Chińczyka, że jest Chińczykiem. Na wszystkie pytania, które człowiek zadaje, udziela właściwych odpowiedzi w sposób tak naturalny, że Chińczyk jest przekonany, iż rozmawia z innym Chińczykiem. Zwolennicy mocnej sztucznej inteligencji wyciągają stąd wniosek, że komputer rozumie chiński, tak jak człowiek.

Teraz Searle proponuje, żeby założyć, iż to on sam siedzi wewnątrz komputera. Innymi słowy, on sam znajduje się w małym pokoju, w którym dostaje chińskie znaki, konsultuje książkę reguł, a następnie zwraca inne chińskie znaki, ułożone zgodnie z tymi regułami. Searle zauważa, że oczywiście nie rozumie ani słowa po chińsku, mimo iż wykonuje powierzone mu zadanie. Następnie argumentuje, że jego brak rozumienia dowodzi, że i komputery nie rozumieją chińskiego, znajdując się w takiej samej sytuacji jak on:

są bezumysłowymi manipulatorami symboli i nie rozumieją, co 'mówią', tak jak i on nie rozumie treści chińskich znaków, którymi operował.

Formalne argumenty

W roku 1984 Searle wyprodukował bardziej formalna wersje argumentu w której chiński pokój jest częścią. On podał listę czterech założeń:

1. Mózgi wytwarzają umysły
2. Syntaksa nie jest wystarczająca dla semantyki
3. Programy komputerowe są całkowicie zdefiniowane przez ich formalna albo syntaktyczną strukturę
4. Umysły mają treści mentalne w szczególności one maja zawartości semantyczne

Drugie założenie jest widocznie wspierane przez argumentu chińskiego pokoju ponieważ Sercem utrzymuje ze pokój przestrzega jedynie formalnych reguł składni i nie żozumiećhińskiego. Searle sugeruje ze to prowadzi bezpośrednio do czterech konkluzji

1. Żaden program komputerowy z siebie nie jest wystarczający aby dać systemowi umysł. Wkrótce programy nie są umysłami i nie są z siebie zdolne by mieć umysły.
2. Sposób w jaki funkcje mózgu wytwarzają umysły nie może być wyłącznie zasługą wykonywania programu komputerowego.
3. Cokolwiek jest w stanie wytworzyć umysł będzie miało władze przyczynowe co najmniej równoważne mózgowi
4. Procedury programu komputerowego nie będą same z siebie wystarczające aby wytworowi ludzkiemu nadać stany mentalne równoważne ludzkim. Wytwory ludzkie będą wymagały zdolności i władz mózgu

Ten argument oparty ściśle na scenariuszu chińskiego pokoju jest skierowany na stanowisko nazwane przez Searla "mocna AI". Mocna AI jest poglądem ze odpowiednio zaprogramowane komputery (albo programy same) mogą rozumieć naturalny język i aktualnie mieć inne zdolności mentalne podobne do ludzkich których możliwości one naśladują. Nawiązując do mocnej AI komputer może grac w szachy inteligentnie, robić bystre ruchy albo rozumieć język. Przez kontrast słaba AI ("weak AI") jest poglądem, że komputery są tylko użyteczne w psychologii, lingwistyce i innych obszarach w części ponieważ mogą symulować zdolności mentalne. Ale słaba AI nie twierdzi ze komputery aktualnie rozumieją albo są inteligentne. Argument chińskiego pokoju nie jest skierowany w słabą AI ani nie jest jego celem pokazanie ze maszyny nie mogą myśleć - Searle powiada, że mózgi są maszynami a mózgi myślą. On jest skierowany w pogląd, że **formalne wyliczenia na symbolach produkują myśl**.

1.7.2 odpowidz systemu

Chociaż osoba w chińskim pokoju nie rozumie chińskiego, przypuszczalnie osoba w pokoju wraz z książką reguł rozpatrywanych razem jako system rozumie ten język.

Odpowiedzią Searla na to jest, że ktoś może w zasadzie zapamiętać książkę reguł, a wtedy będzie mógł reagować jakby rozumiał chiński, ale nadal będzie tylko postępował według zbioru reguł, bez rozumienia znaczenia symboli, jakimi manipuluje. To prowadzi

do interesującego problemu osoby, która może płynnie rozmawiać po chińsku ńie znającćhińskiego. To jest otwarte, czy taka osoba rozumie chiński nawet jeśli chiński mówca twierdzi inaczej.

W Consionsciousness Explained Daniel C. Dennett podaje rozszerzenie odpowiedzi systemu, którego podstawa polega na tym, że przykład Searla jest zamierzony do wprowadzenia wyobrażającego sobie w błąd. Jesteśmy proszeni wyobrazić sobie maszynę, która przeszłaby test Turinga przez zwykle manipulowanie symbolami. Jest wysoce nieprawdopodobne ze taki zgrubny system przeszedłby test Turinga.

Jeżeli system byłby rozszerzony żeby włączyć rozmaite systemy detekcji by doprowadzić do spójnych odpowiedzi i byłby przepisany na wielka maszynę równoległą zamiast na szeregową maszynę Von Neumanna szybko byłoby bardziej mniej óczywisteźe nie ma świadomego postrzegania. Dla chińskiego pokoju przejść test Turinga albo operator musiałby być wspomagany przez wielka liczbę pomocników albo ilość czasu danego na wyprodukowanie odpowiedzi na nawet najbardziej podstawowe pytania byłaby absolutnie nadzwyczajna - wiele milionów lub przypuszczalnie miliardów lat.

Uwaga zrobiona przez Dennetta jest taka, że przez wyobrażenie "tak, to jest przekonywające używać tablicy wyszukiwania do wejścia i dać wyjście i przejść test Turinga-żaburzamy złożoność istotnie zaangażowaną w taki sposób ze to rzeczywiście wydaje się óczywisteźe ten system nie może być świadomy. Jednak taki system jest nieodpowiedni. Jakikolwiek rzeczywisty system zdolny do istotnego spełnienia koniecznych warunków byłby tak złożony ze wcale nie byłoby óczywisteźe brakuje mu zrozumienia chińskiego. On musiałby z pewnością porównywać koncepcje i formułować możliwe odpowiedzi usuwać opcje i tak dalej aż wypadłby albo jak powolna i całkowita analiza semantyki wejścia albo zachowywałby się jak każdy inny mówiący po chińsku. Dopóki nie musimy dowodzić ze miliard mówiących chińskim ludzi jest czymś więcej niż siecią równoległa symulująca maszynę Von Neumanna na wyjściu musimy zaakceptować, że chiński pokój jest tak samo mówiący po chińsku jak każdy inny Chińczyk.

1.7.3 Jasny Pokój

Analogia Churchlandów(9) jest bardzo sugestywną i przy tym najnowszą repliką wymierzoną przeciw trzeciemu aksjomatowi omawianego eksperymentu myślowego; określić ją można jako wariant odpowiedzi systemowej.

Filozofowie z San Diego odrzucają pewnik Searle'a: Składnia sama przez się ani nie kształtuje semantyki ani do niej nie wystarcza; uważają go za bardzo spłycony i sądzą, że cały Chiński Pokój został specjalnie skonstruowany po to, by obudować wyżej wymieniony aksjomat. Podają też - przejrzysty według nich - kontrprzykład zwany Jasnym Pokojem. "Wyobraźmy sobie ciemny pokój a w nim człowieka trzymającego duży magnes (...). Jeśli ten człowiek będzie wymachiwał magnesem do góry i na dół, to (...) powinno to wywołać koliste rozchodzenie się fal elektromagnetycznych, a zatem powinno zrobić się jasno. Jednak każdy z nas wie, że (...) jest nie do pomyślenia, byśmy mogli uzyskiwać światło poprostu przez poruszanie magnesem w kółko"(10).

Ktoś może twierdzić na podstawie powyższego przykładu, że światło nie ma związku z falami elektromagnetycznymi, gdyż potrząsanie magnesem nie daje światła w ciemnym pokoju. Odwołując się zaś do Chińskiego Pokoju można wykazywać, jak Searle, że potrząsanie, mieszanie chińskich symboli nie da "światłażozumienia języka chińskiego. Analogicznie do argumentu z 1980 roku Churchlandowie również podają aksjomaty związane z Jasnym Pokojem:

Aksjomat 1: Elektryczność i magnetyzm są formami energii.

Aksjomat 2: Zasadniczą cechą światła jest jasność.

Aksjomat 3: Energia nie jest ani konieczna, ani wystarczająca dla uzyskania jasności.

Wniosek 1: Elektryczność i magnetyzm nie są konieczne ani wystarczające dla wyjaśnienia istoty światła.

Gdyby taki tok myślenia przedstawiono po ogłoszeniu koncepcji Maxwella na temat elektromagnetycznej natury światła a przed uznaniem faktu tożsamości światła i fal elektromagnetycznych, to aksjomaty te mogłyby stanowić podstawę odrzucenia teorii Maxwella - konkludują Churchlandowie. A przecież jednak wskutek badań teoria ta została potwierdzona. Toteż Churchlandowie radzą czekać powołując się na precedensy w historii nauki, kiedy to poglądy odrzucane były następnie akceptowane. Według nich być może późniejsze badania wykażą, że można zbudować semantykę jedynie przy pomocy samej syntaktyki i wnioski wyciągnięte z Chińskiego Pokoju uważają za przedwczesne.

Na zakończenie tego rozdziału przypominamy dobrze znaną historię związaną z, mogącym być uważanym za pewien przejaw inteligencji, automatycznym tłumaczeniem. Zadanie polegało na przetestowaniu jakości działania programu tłumaczącego w ten sposób aby najpierw przetłumaczyć zadaną frazę z angielskiego na rosyjski a potem wynik tłumaczenia przetłumaczyć na angielski i porównać z oryginałem. Zdanie wyjściowe brzmiało:

The spirit is willing but the flash is weak (Chciałaby dusza lecz ciało nie może).

W wyniku użycia programu otrzymano następujące zdanie w języku angielskim:

The vodka is strong but the meat is rotten (Wódka jest mocna, ale mięso jest zepsute).

Rozdział 2

Reprezentacja i przeszukiwanie

Już na pierwszych latach studiów na kierunkach informatycznych dowiadujemy się o bardzo dużej roli algorytmów i struktur danych. W szczególności te drugie częstokroć determinują jakie algorytmy można użyć i na ile będą one efektywne. Zamiana jednej struktury danych na inną, przy pozostawieniu reszty algorytmu bez zmian, może pociągać za sobą zupełnie inne działanie (porównaj algorytmy przeszukiwania wszerz i w głąb opisane w rozdziale 5).

Mówiąc o sztucznej inteligencji w aspekcie czysto użytkowym, można powiedzieć inżyniersko-technicznym, dziedzinę tą określimy jako

Nauka o sposobach reprezentacji i przeszukiwania zgromadzonej wiedzy wykorzystująca inteligentne mechanizmy.

Oczywiście zdanie to jest bardzo ogólne, ale zawiera dwa bardzo istotne problemy, z których musimy zdawać sobie sprawę, gdy podejmujemy się rozwiązania praktycznego problemu

Problem reprezentacji. W jaki sposób przedstawiać obiekty świata rzeczywistego i związki między nimi, aby maksymalnie ułatwić operowanie zgromadzoną wiedzą.

Problem przeszukiwania. W jaki sposób przeszukiwać wiedzę jaką posiadamy w celu znalezienia odpowiedzi na postawione pytanie.

Pod kontem tych dwóch problemów spojrzymy teraz na sztuczną inteligencję.

2.1 Problem reprezentacji

Liczby rzeczywiste

Rolą każdej reprezentacji jest uchwycenie istotnych cech obiektów i ich udostępnienie algorytmowi rozwiązującemu postawiony problem. Złożoność świata, który staramy się opisać wymaga od nas zachowania odpowiedniego stopnia abstrakcji aby możliwe stało się osiągnięcie akceptowalnej efektywności.

Mając to na uwadze, przyjrzyjmy się liczbom rzeczywistym. Ogólnie mówiąc, liczby rzeczywiste wymagają nieskończonego ciągu cyfr aby móc je przedstawić. Jeśli nawet dla niektórych liczb ciąg ten ma skończoną długość to często i tak znacznie przekracza zakresy reprezentacji maszynowej. Niech rozważaną przez nas dalej liczbą o nieskończonym

rozwinięciu dziesiętnym będzie liczba π

$$\pi = 3.14159265358979323846264338327950288419716939937510\ldots$$

Oczywiście najlepiej jaśli możliwa jest jej reprezentacja w postaci symbolicznej, ale w praktyce wcześniej czy później i tak będziemy musieli odwołać się do jej reprezentacji liczbowej. Siłą rzeczy reprezentacja ta będzie skończona, tj. wykorzystywała jedynie pewną liczbę cyfr z rozwinięcia. Kwestią wyważenia złożoności reprezentacji i dokładności otrzymanych wyników jest wybór ilości cyfr. Załóżmy, że w rozważanym systemie możemy przeznaczyć na reprezentację 8 znaków. Wówczas liczbę π zapiszemy np. jako

```
76543210 - numer znaku
||||||||
+3.14159
```

Dla kontrastu liczbę Eulera zapiszemy jako

```
76543210 - numer znaku
||||||||
+2.71828
```

W ten oto sposób otrzymujemy trzecią reprezentację. Teoretycznie umożliwia ona reprezentację liczb rzeczywistych, które mieszczą się w przedziale $(-99999.9, +99999.9)$. Ma ona trzy zasadnicze wady

1. „Marnuje" jedno miejsce na zapisanie znaku rozdzielającego część całkowitą od ułamkowej.

2. Dokładność reprezentacji jest zminna: liczby małe reprezentowane są dokładniej (więcej miejsc po przecinku) niż duże

   ```
   +999.111 - dokładność 0.001
   +9.99111 - dokladność 0.00001
   ```

3. Jest kłopotliwa w praktyce (zautomatyzowane wykonywanie obliczeń), co spowodowane jest tym, że w każdej liczbie znak rozdzielający część całkowitą od ułamkowej może być w innym miejscu co wymaga dodatkowych zabiegów w celu jego „obsłużenia".

Wadę drugą i trzecią można wyeliminować ustalając ilość cyfr części całkowitej i ułamkowej (tzw. notacja stałoprzecinkowa, ang. *fixed-point*), np. całkowita 3, ułamkowa 4. Wówczas nie ma konieczności zapisywania znaku rozdzielającego obie części od siebie, ale za to pojawiają się inne problemy. Spójrzmy jak teraz zapiszemy liczbę π

```
76543210 - numer znaku
||||||||
+0031415
```

Widzimy, że dokładność reprezentacji liczby π zmalała. Oczywiście zawsze możemy ustalić, że część całkowita to 1 znaki a ułamkowa 6, ale wówczas największą liczbą dodatnią jest $+9.999999$ co zapewne nie wystarczy aby móc w projektowanym systemie

wykonywać działania arytmetyczne z odpowiednio szerowkiego zakresu. Pamiętać bowiem musimy, że szukana reprezentacja nie ma służyć do reprezentacji jednej liczby rzeczywistej. Pozbyliśmy się zatem trzech problemów, ale zyskaliśmy jeden nowy: ścisłe powiązanie zakresu i dokładności – zwiększając jedno, zmniejszamy drugie.

Najpowszechniej stosowanym obecnie sposobem pozwalającym przezwyciężyć (przynajmniej częściowo) problem związany z notacją stałoprzecinkową jest notacją zminnoprzecinkowa (ang. *floating point*). Wykorzystujemy w niej takie oto spostrzeżenie. Rozważmy nastnastępujące liczby rzeczywiste

$$+0.000000000123450$$

$$+1.234500000000000$$

$$+12345.00000000000$$

$$+123450000000000.0$$

Zauważmy, że każdą z nich możemy zapisać w pewnie znormalizowany a co ważniejsze wymagający mniej miejsca sposób

$$+1.2345 \cdot 10^{-10} = +0.000000000123450$$

$$+1.2345 \cdot 10^{0} = +1.234500000000000$$

$$+1.2345 \cdot 10^{+4} = +12345.00000000000$$

$$+1.2345 \cdot 10^{+14} = +123450000000000.0$$

Ogólny wzór ma postać

$$z_M M \cdot 10^{z_C C},$$

gdzie z_M – to znak mantysy (i w efekcie liczby), z_C – znak cechy, M – mantysa (liczba „rzeczywista" a więc posiadająca część całkowitą i ułamkową), C – cecha (liczba naturalna). Liczby 10 nie musimy pamiętać, bo przecież zakładamy, że operujemy liczbami systemu dziesiętnego. Przyjmując założenie, że na mantysę przeznaczamy pięć miejsc a na cechę jedno, liczbę π zapiszemy jako

```
+31415+0
```

Zapis ten wymagał przyjęcia jeszcze jednego ustalenia, podobnego do tego z systemu stałoprzecinkowego, a związanego z ilością znaków przypadających na część całkowitą i ułamkową w mantysie. Jest to konieczne, ze względu na brak jednoznaczności liczb rzeczywistych zapisywanych w takiej notacji

$$3.1415 \cdot 10^{0} = 314.15 \cdot 10^{-3} = 0.0031415 \cdot 10^{+3} = \ldots$$

W tym przypadku na część całkowitą przypada jeden znak, na część ułamkową cztery znaki.

Stałoprzecinkowy zapis liczy rzeczywistej jest przykładem poszukiwania złotego środka pomiędzy tym co chcemy wyrazić (liczby rzeczywiste), możliwościami jakie mamy (system dziesiętny), ograniczeniami jakie są nam narzucone (ograniczenie fizyczne do ośmiu znaków) i łatwością użytkowania.

Sortowanie przy pomocy drzewa

Rozpatrzmy teraz jeden z najbardziej elementarnych problemów algorytmiki – problem sortowania (liczb). Istnieje wiele algorytmów różniących się stopniem skomplikowania, szybkością działania, uniwersalnością, które pozwolą to zadanie rozwiązać. Okazuje się natomiast, że zadanie to bardzo łatwo można wykonać nie pisząc praktycznie żadnego algorytmu.

Aby pokazać, że jest to możliwe przyjmijmy drzewo binarne jako wykorzystywaną strukturę danych (patrz rozdział 4). Na drzewie określamy dwie operacje. Pierwsza z nich o nazwie `put(elem)` pozwala wprowadzić element do drzewa, druga – `print()` – pozwla wypisać elementy drzewa. Działanie `put(elem)` definiujemy następująco

1. Jeśli drzewo jest puste, wówczas `elem` staje się jego korzeniem.
2. Jeśli drzewo nie jest puste, wówczas `elem` musi trafić do takiego (nowo utworzonego) węzła, aby wszystkie elementy po lewej stronie (lewe dzieci) były mniejsze lub równe danemu elementowi, a wszystkie elementy po prawej stronie (prawe dzieci) były większe od wstawianego elementu.

Jeśli więc mamy drzewo puste to wstawiając do niego jakiś element automatycznie staje się one korzeniem tego drzewa

```
drzewo: puste

wstawiamy: 5

drzewo:

5
```

Wstawiając elementy do niepustego drzewa musimy już należć dla nich odpowiednią pozycję

```
drzewo:

          5
         / \
        3   10

wstawiamy: 14

drzewo:

          5
         / \
        3   10
             \
              14


wstawiamy: 7
```

`drzewo:`

```
    5
   / \
  3   10
     / \
    7   14
```

Działanie `print()` definiujemy następująco:

1. Wypisywanie elementów drzewa zaczynamy od korzenia.
2. Aby wypisać dany węzeł najpierw trzeba wypisać wszystkie jego lewe dzieci, potem dany węzeł, a na końcu wszystkie prawe dzieci.

A zatem wywołanie `print()` dla drzewa o węzłach jak poniżej

```
    c
   / \
  d   a
     / \
    b   e
```

spowoduje wypisanie

```
d, c, b, a, e
```

Mając określone zasady działania takiej struktury danych, wszystko co należy zrobić, aby posortować jakiś ciąg liczb, to

1. Używając metody `put(elem)` wstawić te elementy do drzewa (kolejność jest dowolna, np. w porządku występowania).
2. Wypisać elementy używając metody `print()`.

Przykład ten pokazuje, że dobór odpowiedniej reprezentacji czy struktury danych, na której przyjdzie nam działać może znacznie ułatwić przyszłe operacje. Przecież sortowane liczby, zamiast w opisanym drzewie, teoretycznie równie dobrze można by było przechowywać w tablicy czy na stosie. Tylko co byśmy wtedy zyskali? W jakim stopniu ułatwiło by to nasze działania zmierzajace do posortowania wszystkich liczb?

Reprezentacja opisowa

Często jednak problemy stawiane przed sztuczną inteligencją nie dają się tak łatwo opisać za pomocą „tradycyjnych" metod i narzędzi jak to np. miało miejsce w przypadku liczb rzeczywistych. Wynika to z innych zapotrzebowań stawianych przez sztuczną inteligencję

- sztuczna inteligencja kładzie raczej nacisk na „jakość" reprezentacji niż ilość opisanych obiektów;
- wykorzystuje raczej wnioskowanie czy przekształcenia symboliczne niż czyste obliczenia numeryczne;

- wymaga raczej stosowania szerokiego spektrum technik pozwalających na operowanie różnorodną wiedzą niż pojedynczego, dobrze zdefiniowanego algorytmu.

Wielokrotnie zależy nam na oddaniu powiązania pomiędzy obiektami niż na opisywaniu ich samych (w sensie wymieniania cech). Na przykład samo wymienienie części składowych samochodu nie jest jego wystarczającym opisem. W tym przypadku możliwe, że nawet istotniejsze okazuje się opisanie wzajemnych zależności pomiędzy elementami i sposobu współdziałania, niż nawet bardzo dokładne wymienianie cech (atrybutów) im przynależnych. Jednym ze sposobów opisowej reprezentacji jest **rachunek predykatów**. **Predykat** jest nazwą relacji (własności, cechy) jaka wiąże ze sobą argumenty, np.

- `jestZolty(slonce).` – predykat `jestZolty` opisujacy to, że obiekt `slonce` posiada kolor żółty;
- `lubi(zosia,jasia).` – predykat `lubi` opisujacy to, że obiekty `zosia` i `jas` związane są relacją `lubi` (zapis ten można czytać: *Zosia lubi Jasia* nie jest jednak powiedziane, że zachodzi związek *Jaś lubi Zosię*);
- `wiekszy(slon,mrowka).` – predykat `wiekszy` opisujacy to, że obiekt `slon` jest większy niż obiekt `mrowka`;
- `posiada(adam,samochod(fiat,uno)).` – predykat `posiada` opisujacy to, że obiekt `adma` posiada samochód, przy czym dodatkowo wiadomo, że jest to samochód marki Fiat.

Taka reprezentacja opisowa szczególnie dobrze nadaje się do zapisu „rzeczy" nie posiadających reprezentacji numerycznej, jak np. *lubi* czy *posiada* i oddawania szerszych związków i zależności pomiędzy obiektami. Bez niej kłopotliwe staje się już choćby opisanie sceny ze świata kilkunastomiesięcznego dziecka. Niech sceną będzie budowla z klocków o następujących kształtach: kostka (sześcian), kula, piramida (czworościan) (rys. **??**). Scenę tę przy pomocy predykatów możemy opisać w następujący sposób

```
kostka(a).
kostka(b).
kostka(c).
kula(d).
piramida(e).
jestNa(b,a).
jestNa(d,b).
jestNa(e,c).
```

Na tym zakończymy wymienianie różnych sposobó reprezentacji, bo nie taki jest cel tego rozdziału. Chcieliśmy w nim bowiem pokazać, że wybór reprezentacji nie jest wcale zadaniem łatwym. Często jest to kompromis pomiędzy tym co chcemy uzyskać a tym co możemy uzyskać (np. reprezentacja liczb rzeczywistych). W poszukiwaniu tego kompromisu cały czas powinniśmy mieć na uwadze użytkowość wybranej reprezentacji – czasem warto użyć tej trudniejszej jeśli potem daje ona łatwiejsze rozwiązanie problemu (np. sortowanie przy pomocy drzewa). W wielu przypadkach problemy stawiane przed sztuczną inteligencją nie dają się tak łatwo opisać za pomocą „tradycyjnych" metod i narzędzi co zmusza nas do poszukiwania nie tyle alternatywnych sposobów reprezentacji co całkowicie nowych (np. rachunek predykatów).

2.2 Problem przeszukiwania

Wybór właściwej reprezentacji to pierwszy krok w kierunku rozwiązania problemu. W dalszej części mówiąc **stan**, będziemy mieli na myśli konkretną „wartość" reprezentacji, np.

- w przypadku liczb rzeczywistych stan to reprezentacja konkretnej liczby rzeczywistej w wybranej notacji;
- w przypadku gry w szachy, stan to konkretne ustawienie figur na szachownicy;
- w przypadku diagnozowania choroby (stanu pacjenta), stan to pytania na które znamy odpowiedź wraz z samymi odpowiedziami;
- w przypadku przepisu kucharskiego, stan to etap na jakim jesteśmy przyrządzając potrawę.

Drugim, obok reprezentacji, istotnym elementem systemów inteligentnych jest przeszukiwanie (poszukiwanie, ang. *search*). Ludzie rozwiązując problem, świadomie lub nie, rozważają wiele alternatywnych strategii. Na przykład szachista „przeszukuje" możliwe do wykonania ruchy i wybiera najlepszy biorąc pod uwagę odpowiedź przeciwnika, stopień w jakim wykonane posunięcie wspiera obraną strategię gry czy nawet psychiczny stan oponenta (np. osoba wyraźnie rozkojarzona może nie zwrócić w porę uwagi na zbliżające się zagrożenie, przez co można pozowlić sobie na bardziej ryzykowne warianty gry). Rozważa także cele krótkoterminowe jak zbicie figury przeciwnika, czy poświęcenie własnej za cenę zyskania przewagi pozycyjnej. Rozważanie tych wszystkich aspektów i dobór odpowiednich do nich technik w sztucznej inteligencji nazywa się **przeszukiwaniem przestrzeni stanów** (ang. *state space search*).

W przypadku gry w szachy, poczynając od wejściowego ustawienia figur, zawsze mamy skończoną ilość ruchów jakie możemy wykonać. Każdy z nich pwoduje postanie na szachownicy innego układu figur (stanu). Na każdy z takich stanów, przeciwnik może odpowiedzieć wykonując skończoną ilość ruchów ponownie prowadzących do różnych stanów, itd. W tym przypadku, każdy stan możemy reprezentować jako węzeł w grafie (drzewie). Połączenia między węzłami opisują fakt, że można przejść z jednego stanu do drugiego. Otrzymaną w tym przypadku strukturę nazywamy **grafem stanów** (ang. *state space graph*). Teoretycznie graf ten reprezentuje (może reprezentować) wszystkie możliwe do wykonania ruchy a więc i wszystkie możliwe do przeprowadzenia gry. Zatem poszukiwanie efektywnej strategii gry wiąże się ze znalezieniem takiej ścieżki (lub ścieżek), która prowadzi z bieżącego stanu (węzła) do stanu najkorzystniejszego. Gra zaś polega na takiem wykonywaniu ruchów aby znajdować się możliwie blisko wybranej strategii.

W większości problemów, nawet jeśli dają się one (teoretycznie) opisać przy pomocy grafu stanów, to i tak opis ten, w swej pełnej formie, jest mało użyteczny. Dlaczego? Odpowiedź jest prosta – nie jesteśmy w stanie z niego skorzystać, ze względu na ogromną ilość stanów jakie musielibyśmy zbadać. Złożoność drzewa gry w tym przypadku została oszacowana ([28]) na przynajmniej 10^{120} (tzw. liczba Shannona, ang. *Shannon number*; obecnie szacuje się tą wielkość na 10^{123}). Jest to liczba większa niż liczba nanosekund jakie upłynęły od wielkiego wybuchu.

Ludzie zwykle odrzucają strategie siłowe (ang. *brute-force search* lub *exhaustive search*) polegające na rozważaniu **wszystkich** możliwych stanów (sytuacji). Częściej

raczej, posiłkując się posiadanym doświadczeniem, biorą pod uwagę tylko te ruchy, które mają szansę zbliżyć nas do rozwiązania, wybierają te najbardziej „obiecujące", pomijając mnóstwo tych, które zgodnie ze zdrowym rozsądkiem muszą prowadzić do porażki. Taka zdolność do selektywnego przeszukiwania przestrzeni, z unikaniem stanów ewidentnie nie prowadzących do rozwiązania, nazywana jest **heurystyką**.

2.3 Kółko i krzyżyk – przykład

W przykładzie tym spróbujemy pokazać wzajemną zależność reprezentacji i, będącej niejako jej konsekwencją, przyjmowanej strategii gry.

2.4 Heurystyka

Pojęcie *heurystyki* (gr. *heuresis* – odnaleźć, odkryć, *heureka* – znaleźć) spotykane jest w wielu, czasem odległych od siebie naukach jak np. filozofii, sztucznej inteligencji czy teorii informacji. W najogólniejszym ujęciu jest to nazwa dziedziny wiedzy, której cel stanowi poszukiwanie i badanie optymalnych metod oraz reguł odnajdywania odpowiedzi na stawiane zapytania lub problemy. Jest to także umiejętność wykrywania nowych faktów oraz związków pomiędzy nimi, prowadząca do odkrywania nowych prawd i stawiania hipotez.

W informatyce i teorii obliczeń heurystyka to metoda znajdowania rozwiązań, która kosztem znacznego skrócenia czasu działania nie gwarantuje znalezienia rozwiązania optymalnego, a często nawet dopuszczalnego (prawidłowego). Wbrew pozorom takie działanie ma sens, gdyż

- Rozwiązań takich używamy w sytuacji, gdy nie jest możliwe znalezienie poprawnego (w sensie: dającego dopuszczalne/optymalne rozwiązania) algorytmu (np. prognozowanie pogody) lub jego czas wykonania jest nieakceptowalny (np. analiza wszystkich możliwych posunięć w grze w szachy).

- Często rozwiązania przybliżone pozwalają na ich podstawie otrzymać rozwiązanie optymalne za pomocą poprawnego algorytmu, ale przy znacznie mniejszym nakładzie czasowym (np. szybkie sprowadzenie w przestrzeni poszukiwań w sąsiedztwo globalnego minimum a następnie dokładne wyznaczenie wartości tego minimum innym algorytmem, którego znaczna złożoność czasowa zostaje ograniczona przez wstępne zawężenie dziedziny).

- Tak na prawdę, rzadko kiedy potrzebujemy rozwiązania optymalnego. W wielu przypadkach w zupełności wystarcza jego przybliżenie. Można powiedzieć, że ludzie poszukują nie tyle rozwiązań optymalnych co satysfakcjonujących. Satysfakcjonjący w tym przypadku oznacza spełniający pewne kryteria uznane za fundamentalne i dopuszczający możliwość niespełnienia kryteriów za takie nie uznanych. Znalezienie pierwszego z takich rozwiązań, często kończy poszukiwania. Dobrym przykładem takiego postępowania jest poszukiwanie miejsca na plaży w zatłoczonym kurorcie. Fundamentalnym kryterium może być (i zwykle jest) bliskość wejścia na plażę a mniej istotnym dostępna przestrzeń czy też odległość do wody. Większość ludzi rozkłada się zwykle tuż przy wejściu na plażę całkiem nie biorąc pod

uwagę, że kilkadziesiąt metrów dalej od niego, mieli by znacznie więcej miejsca[1].

- Pomimo tego, że heurystyka może dawać niewystarczająco dobre rozwiązanie w najgorszym przypadku, to najgorszy przypadek relatywnie rzadko pojawia się w praktyce.

2.4.1 Heurystyka a algorytm

Zasadnicza różnica między postępowaniem algorytmicznym a heurystycznym polega na tym, że pierwsze podejście zawsze daje rozwiązanie (choć czas oczekiwania na rozwiązanie może być nawet nieskończenie długi), podczas gdy drugie dopuszcza możliwość zawodnego działani. Stąd też techniki algorytmiczne stosowane są najczęściej w tych przypadkach, dla których znana jest skuteczna, sprawdzona i dobrze opisana metodologia postępowania. Heurystyka natomiast wykorzystywana jest wszędzie tam, gdzie nie są znane ścisłe i dokładne procedury, gdzie podążanie „utartymi" sposobami myślenia nie prowadzi do rozwiązania, gdzie wymagana jest zdolność odnajdywania odpowiedzi na postawione pytania (problemy). Można powiedzieć, że stosując heurystykę i tak nic nie tracimy, bo algorytmicznie i tak problemu nie jesteśmy w stanie rozwiązać.

Zanim przystąpimy do wyboru odpowiedniej metody heurystycznej, musimy zastanowić się nad kilkoma kluczowymi kwestiami

1. Czy problem daje się rozłożyć na niezależne problemy, podobne do wyjściowego a jednak o znacznie mniejszej złożoności?
2. Czy w razie stwierdzenia nieoptymalnego działania istnieje możliwość cofnięcia się, czy też powtórzenia pewnych operacji?
3. Na ile przewidywalna jest przestrzeń stanów związana z problemem?
4. Na ile trudnym zadaniem jest ocena uzyskanego rozwiązania?
5. Czy interesuje nas tylko rozwiązanie problemu czy też ciąg operacji do niego prowadzący?
6. Jak duża wiedza konieczna jest do rozwiązania problemu?

2.4.2 Pytanie 1

Możliwość rozłożnenia zadanego problemu na podproblemy (najlepiej jeśli w pewnym stopniu niezależne) tej samej natury, ale o znacznie mniejszym stopniu złożoności jest bardzo cenną właściwością pozwalającą na

- uproszczenie metody rozwiązującej – gdyż podproblemy są prostsze niż problem zadany;
- przyspieszenie, gdyż niezależność podproblemów pozwala na ich równoległe wykonywanie.

Problemy, które posiadają taką własność nazwiemy **dekomponowalnymi** (?) (ang. *decomposable*), te zaś które jej nie posiadają **niedekomponowalnymi** (?) (ang. *non-decomposable*).

[1] To irracjonalne moim zdaniem zachowanie należy tłumaczyć chyba tylko i wyłącznie przeogromną leniwością.

Problem dekomponowalny

Jako przykład dekompozycji problemu rozpatrzmy algorytm sortowania przez scalanie (ang. *merge sort*). Algorytm ten jest dobrym przykładem algorytmów typu **dziel i zwyciężaj** (ang. *divide and conquer*). Ideą działania tego typu algorytmów jest właśnie podział problemu na „mniejsze" podproblemy, tzn. takie, których rozwiązanie jest łatwiejsze (szybsze itp).

W algorytmie można wyróżnić trzy podstawowe kroki:

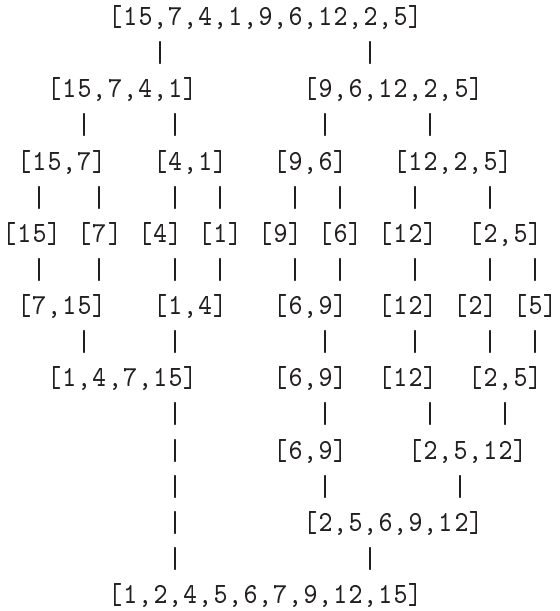
1. Podział danych na dwa podciągi równej długości (w przypadku nieparzystej liczby wyrazów jednen z nich będzie zawierał o jeden element więcej).

2. Zastosowanie sortowania przez scalanie dla każdej części oddzielnie (jeśli zawiera ona więcej niż jeden element).

3. Łączenie posortowanych ciągów w jeden posortowany ciąg.

Zauważmy, że występująca w kroku 3 operacja łączenia jest bardzo prosta i szybka do wykonania ze względu na to, że łączone ciągi są już posortowane.

Przykład działania:



dziel i zwyciężaj calka sortowanie
nondecomposable: uklad rownan

Pytanie 2

klasy problemu: ignorable, gdy uzyskane wyniki pośrednie mogą zostać zignorowane, gdyż nie mają żadnego wpływu na ostateczne rozwiązanie. np. dowodzenie twierdzen recoverable, gdy można cofnąć wykonane kroki do pewnego miejsca aby rozpocząć z niego poszukiwanie rozwiązania w innym kierunku np. gra w kwadraty irrecoverable, gdy wykonanie jakiegoś kroku jest niodwołalne (np. gra w szachy)

Pytanie 3

certain-outcome (np. gra w kwadraty)
uncertain-outcome (np. gra w szachy a jeszcze lepiej w pokera)

Pytanie 4

any-path problem (absolute) (np. czy istnieje połączenie pomiędzy węzłami grafu)
best-path problem (relative) (np. ścieżka o najmniejszym koszcie w grafie)
But is this the solution to the problem? The answer is that we cannot be sure unless we also try all other paths to make sure that none of them is shorter.

Pytanie 5

path problem ze sloikami: informacja, że rozwiązaniem jest stan (2,0) nie jest wystarczająca (ilość wody w sloiku 4l, il. H2O w sl. 3l). state problem z calkami, mini-max (wybor ruchu)

Pytanie 6

gra w szachy - mala wiedza (reguly) / duza wiedza gdy dochodzi baza otwarc
chat-boot - duza wiedza wykrywanie w tlumie podejrzanie zachowujacych sie ludzi

2.5 Co dalej?

tutaj info o tym jakie rozdzialy nalezy czytac, tj. ktore sa o reprezentacji a ktore o szukaniu

Rozdział 3

Wiedza a język

Problem wiedzy i jej miejsca w (sztucznej) inteligencji omawiany był w rozdziale 1. Jeśli, już przyjmiemy, że wiedza jest nam niezbędna, to do rozwiązania pozostaje wiele kwestii związanych z ogólnie pojętym „zarządzaniem" wiedzą. W szczególności interesować nas będą sposoby jej reprezentacji i, jako ich konsekwencja, użyteczność tych sposobów.

W rozdziale tym przyjrzymy się jednemu ze sposobów reprezentowania wiedzy, mianowicie pod postacią faktów i reguł języka logiki (ang. *language of logic*). Posługiwanie się logiką w celu reprezentacji wiedzy jest bardzo kuszące, gdyż udostępnia nam bardzo potężny aparat matematyczny pozwalający na wnioskowanie. Oto możeby bowiem powiedzieć, że pewne zdanie jest prawdziwe w oparciu o inne znane nam zdania o których wiemy, że są prawdziwe. Idea dowodu matematycznego, jako ciągu logicznie wynikających z siebie zdań prawdziwych może zostać potraktowana jako sposób na znalezienie odpowiedzi na postawione pytanie czy zadany problem.

3.1 Logika

Logika (gr. *logos*: słowo, przyczyna, zasada) jest nauką o zasadach i warunkach poprawnego rozumowania i wnioskowania. Logika bada i klasyfikuje struktury zdaniowe (złożone ze stwierdzeń i argumentów), rozpatrując zarówno formalne systemy wnioskowania jak i zdania języka naturalnego. W węższym znaczeniu jako tzw. logika dedukcji (rachunki logiczne), jest nauką o prawach wynikania. W znaczeniu szerszym traktowana jest jako zespół dyscyplin naukowych, dla których przedmiotem badań jest język i czynności badawcze (rozumowanie, definiowanie, klasyfikowanie, itp), których to analizę prowadzi się w celu podania takich reguł posługiwania się językiem i wykonywanie owych czynności, które czyniłyby tę działalność, możliwie najbardziej skuteczna.

Tak więc za pomocą logiki możemy zapisać pewne zdania, związki między nimi oraz sposoby wnioskowania zdań na podstawie zdań innych i sprawdzać w sposób formalny czy zdania te są prawdziwe czy nie.

3.2 Proste fakty

Przyjrzyjmy się następującym zdaniom:

- Pada.
- Jest słonecznie.

- Wieje.
- Jeśli pada to nie jest słonecznie.

Każde z nich jest prostym stwierdzeniem, wyraża prosty **fakt**. Stąd też nazwa rachunek zdań (ang. *propositional calculus*, *propositional logic*). Zdania te formalnie możemy zapisać w następujący sposób:

- `pada`
- `słonecznie`
- `wieje`
- `pada -> ~słonecznie`

Zauważmy w tym momencie, że za pomocą tych faktów stworzyliśmy pewną bazę wiedzy. Co ważniejsze, wiedzą tą możemy zarządzać, w szczególności wykorzystać ją, np. do stwierdzenia, że z faktu iż pada deszcz wynika, że nie jest słonecznie.

W ogólności, rachunek zdań to dział logiki matematycznej badający związki między zdaniami (zmiennymi zdaniowymi) lub funkcjami zdaniowymi utworzonymi za pomocą spójników zdaniowych ze zdań lub funkcji zdaniowych prostszych. Rachunek zdań określa sposoby stosowania spójników zdaniowych w poprawnym wnioskowaniu.

W klasycznym rachunku zdań przyjmuje się założenie, że każdemu zdaniu można przypisać jedną z dwu wartości logicznych – prawdę albo fałsz, które umownie przyjęto oznaczać 1 i 0. Klasyczny rachunek zdań jest więc dwuwartościowym rachunkiem zdań.

W rachunku zdań treść rozpatrywanych zdań nie ma znaczenia, istotna jest jedynie ich wartość logiczna. Wartość logiczną zdań złożonych powstałych przez zastosowanie spójników zdaniowych określa funkcja prawdy, związana z każdym spójnikiem zdaniowym. Wartość ta zależy wyłącznie od prawdziwości lub fałszywości zdań składowych, nie zależy natomiast od ich treści. Szczególną rolę w rachunku zdań odgrywają takie zdania złożone, dla których wartość logiczna jest równa 1, niezależnie od tego, jakie wartości logiczne mają zdania proste, z których się składają. Takie zdania nazywa się prawami rachunku zdań lub tautologiami.

Tak więc mamy sposób na reprezentację wiedzy i wykorzystanie tej reprezentacji. Co jest ważne, sposób ten posiada dosyć mocne i sprawdzone podstawy teoretyczne (logika, matematyka) dające nadzieję na poprawne funkcjonowanie systemu o niego opartego. Logika ta jednak posiada znaczne ograniczenia, ktore dosyć łatwo jest pokazać. Przyjrzyjmy się takim oto zdaniom:

- Zosia jest kobietą.
- Asia jest kobietą.

i ich odpowiednikom wyrażonym za pomocą rachunku zdań

- `zosiakobieta`
- `asiakobieta`

Jak więc widać fakty, które stworzyliśmy nie pozwalają na wyciągnięcie ŻADNYCH wniosków co do „podobieństw" między nimi. Nie możemy dokonywać rozbioru znaczeniowego podanych faktów (pomijamy kwestię tego, że przecież znaczenie „wyrazów" dla komputera nie jest znane). A przecież wiemy, że w języku naturalnym fakty te wyrażają stwierdzenie, że dwa różne „obiekty": `zosia` i `asia` mają cechę wspólną: są `kobietą`.

3.3 Rachunek predykatów

Rachunek predykatów idzie trochę dalej niż rachunek zdań. Moża powiedzieć, że rachunek predykatów bada „wewnętrzną" strukturę zdań. Każde zdanie języka naturalnego wstępnie rozkładane jest na dwie zasadnicze części:

- **obiekty** występujące w zdaniu,
- **predykaty** czyli wyrażenie opisujące pewne **własności** lub **relacje**.

Tak „rozłożone" zdanie występuje w rachunku predykatów pod postacią `predykat(obiekt)` lub `predykat(obiekt1,obiekt2,...,obiektN)`. Rozważmy następujący przykład. Niech dane będzie zdanie:

Żółty miś ma czerwony sweterek.

które możemy zapisać jako dwa zdania połączone operatorem logicznym `AND`:

Miś jest żółty.
`AND`
Miś ma czerwony sweterek.

Z pierwszego ze zdań możemy wyodrębnić obiekt `miś` oraz predykat `być żółtym`. Oznacza to, że obiekt `miś` przynależy do tej klasy (zbioru) obiektów, których cechą jest „żółtość". Z drugiego ze zdań możemy wyodrębnić obiekt `miś` oraz predykat `maCzerwonySweterek`. Oznacza to, że obiekt `miś` przynależy do tej klasy (zbioru) obiektów, których cechą jest posiadanie czerwonego sweterka. Oba te zdania można zapisać jako:

```
żółty(miś) AND maCzerwonySweterek(miś)
```

zamiast zapisu stosowanego w rachunku zdań:

```
żółtymiś AND maCzerwonySweterekmiś
```

Taki rodzaj uogólnienia tworzy pewnego rodzaju relacje: `żółty` oraz `maCzerwonySweterek` pomiędzy obiektem `miś` a zbiorem {`TRUR`, `FALSE`}. Oto możemy bowiem powiedzieć, że `miś` jest w relacji `żółty` z elementem `TRUE`

```
miś -(relacja:  żółty)-> TRUE
```

Inaczej, możemy powiedzieć, że

```
żółty(miś) -> TRUE
```

lub inaczej, że prawdziwy jest fakt

```
żółty(miś).
```

W konsekwencji możemy orzekać o prawdziwości zdań bardziej złożonych:

```
żółty(miś) AND maCzerwonySweterek(miś)
```

czy też, zapisanych w odpowiedniej postaci, zdań będących odpowiednikiem np. takiego oto zdania języka naturalnego

Wszystkie żółte misie z czerwonym sweterkiem znają Krzysia.

Zanim przejdziemy dalej, przyjrzyjmy się kilku faktom wyrażonym w języku naturalnym i ich odpowiednikom wyrażonym jako predykaty.

1. Obelix był mężczyzną.
2. Obelix był Galem.
3. Wszyscy Galowie nie lubią Cezara.
4. Galowie próbują tylko utrudniać życie Rzymianom, których nie lubią.

Obelix był mężczyzna.

`mężczyzna(obelix).`
Fakt ten wyraża podstawową wiedzę o Obelixie, mianowicie to, że zalicza się do grona obiektów o których mozna powiedzieć: mężczyzna. W tym momencie mamy też sposobność zaobserwowania bardzo istotnej rzeczy: wyrażając zdanie języka naturalnego w rachunku predykatów powinniśmy, jeśli tylko nie jest to konieczne z punktu widzenia rozwiązaywanego zadania, „zapomnieć" o formie gramatycznej, właściwym czasie itp. Z naszego punktu widzenia zupełnie nie jest istotne zachowanie formy gramatycznej. Zapis w stylu `mężczyznąBył(obelix).` będzie utrudnieniem. Jak bowiem stwierdzić, czy obiekty `x` oraz `y` mają coś wspólnego jeśli wiemy o nich tylko tyle, że:

```
mężczyznąBył(x).
mężczyzna(y).
```

Pamiętajmy, że nie możemy dokonywać rozbioru znaczeniowego podanych faktów. A mamy tutaj dwie RÓŻNE relacje: `mężczyznąBył` oraz `mężczyzna` .

Obelix był Galem.

`gal(obelix).`

Wszyscy Galowie nie lubią Cezara.

`all(x),gal(x) -> ~lubi(x,cezar).`

Galowie próbują tylko utrudniać życie Rzymianom, których nie lubią.

`all(x),all(y),gal(x),rzymianin(y) -> ~lubi(x,y),utrudniaŻycie(x,y).` W tym zdaniu szczególnie pojawia się problem niejednoznaczności języka mówionego. Czy bowiem oznacza ono, że Galowie utrudniają życie tylko tym z Rzymian, których nie lubią, czy może oznacza ono, że Galowie nie lubią Rzymian i utrudniają im życie?

Tak więc konkretna postać logiki jaką będziemy posługiwać się w dalszej części, to rachunek predykatów (pierwszego rzędu). Intuicyjnie rozumiany predykat jest wyrażeniem opisującym pewne **własności** lub **relacje**. Bardziej formalnie, predykatem (funkcją zdaniową) nazwiemy takie wyrażenie $W(x)$, w którym jeśli podstawimy za x jakąś wartość

z zakresu jej zmienności to będziemy mogli o nim powiedzieć, że jest prawdziwe lub fałszywe. Predykat składa się ze swojej nazwy oraz z jednego, lub wielu argumentów. Argumentami predykatu mogą być stałe, zmienne lub inne predykaty. Rachunek predykatów pierwszego rzędu (ang. *first order predicate calculus*) to system logiczny, w którym kwantyfikatory mogą mówić tylko o obiektach, nie zaś o ich zbiorach. Tak więc nie mogą występować kwantyfikatory typu „dla każdej funkcji X na Y ...", „istnieje własność p, taka że ..." czy „dla każdego podzbioru X zbioru Z ...". Rachunek ten nazywa się też krótko rachunkiem kwantyfikatorów.

Chcąc zapisać zdanie opisujące w jakimś sensie otaczający nas świat czy też nas samych, musimy mieć możliwość zapisania obiektów, których zdanie to dotyczy. W rachunku predykatów używamy do tego termów. Term występuje w jednaj z następujących postaci:

Symbol stałej. Oznacza pojedynczy byt lub pojęcie.

Symbol zmiennej. Oznacza w różnych chwilach czasowych różne byty. Zwykle używane w połączeniu z kwantyfikatorami określającymi zakres działania zmiennej.

Term złożony. Składa się z symbolu funkcyjnego oraz uporządkowanego zbioru termów stanowiących argumenty. Term złożony może oznaczać byt zależny od bytów opisanych jego argumentami. Symbol funkcyjny wskazuje, jak byt zależy od opisujących go argumentów. Przykładami mogą być `lubi(jas,malgosia)` czy `czlowiek(jas)`. Termy złożne nazywane są też strukturami.

Chcąc wyrazić pewne zdanie o obiekcie lub ich większej ilości i związkach między nimi, używać będziemy **symboli predykatów**. Symbol predykatu oraz ciąg termów stanowiących jego argument nazywamy **zdaniem atomowym**. Przykładami mogą być `lubi(jas,malgosia)` czy `czlowiek(jas)`. Powyższe przykłady zaczerpnięte zostały z języka Prolog, gdzie jak widać struktura może być traktowana jako cel, jako argument innej struktury lub jako jedno i drugie. Teoretycznie w rachunku predykatów istnieje zasadnicze rozróżnienie symboli funkcyjnych używanych do tworzenia argumentów i symboli predykatów pełniących funkcję funktorów tworzących zdania.

Zdania złożone tworzymy ze zdań atomowych za pomocą **złączeń logicznych** używając znanych operatorów *nie*, *i*, *lub*, *wynika*, *jest równoważne z*. Ze względu na właściwości odstraszające zapisu matematycznego stosować będziemy bardziej zinformatyzowany zapis. I tak zamiast

- $\neg$ oznaczającego *nie*, czyli *negację*, pisać będziemy `~`;
- $\wedge$ oznaczającego *i*, czyli *koniunkcję*, pisać będziemy `&`;
- $\vee$ oznaczającego *lub*, czyli *alternatywę*, pisać będziemy `|`;
- $\longrightarrow$ oznaczającego *wynika*, czyli *implikację*, pisać będziemy `->`;
- $\Longleftrightarrow$ oznaczającego *jest równoważne z*, czyli *równoważność*, pisać będziemy `<->`.

Nie powinno być dla nas także zaskoczeniem (bo mowa jest o tym na I roku studiów), że implikację i równoważność można zapisać przy pomocy koniunkcji, alternatywy i przeczenia, np. jak to przedstawiono poniżej:

- `a -> b` jest równoważne zapisowi `(~a) | b`;

- `a <-> b` jest równoważne zapisowi `(a & b) | (~a & ~b)`;
- `a <-> b` jest równoważne zapisowi `(a -> b) & (b -> a)`.

Znaczenie zmiennych w zdaniach określone jest wówczas, gdy są one wprowadzone za pomocą kwantyfikatorów. W rachunku predykatów używamy dwóch kwantyfikatorów:

- *kwantyfikatora ogólnego*: $\forall$, $\bigvee$, który zapisywać będziemy `all(x,Z)`;
- *kwantyfikatora szczegółowego*: $\exists$, $\bigwedge$, który zapisywać będziemy `exists(x,Z)`.

3.4 W kierunku języka – postać klauzulowa

Jak doskonale wiemy, wyrażenia rachunku predykatów można zapisywać na wiele różnych równoważnych sobie sposobów. Znakomicie utrudnia to ich formalne przekształcanie. Stąd potrzeba posługiwania się jedną, z wielu możliwych, postaci, a mianowicie **postacią klauzulową**. Zdanie rachunku predykatów zapisane w postaci klauzulowej jest bardzo podobne do zestawu reguł języka Prolog.

Przekształcenia takiego dokonać można w sześciu etapach.

Etap 1. Usunięcie implikacji

Usuwamy wszystkie wystąpienia implikacji i równoważności zgodnie z wcześniej podanymi tożsamościami.

Etap 2. Przesuwanie negacji do wewnątrz

W kroku tym należy wyrażenie przekształcić tak aby negacja miała zastosowanie tylko do wyrażenia atomowego, np.

```
~(a(b) & c(d)   przekształcamy na   ~a(b) | ~c(d)
~all(a,b(a))    przekształcamy na   exists(a,~b(a))
```

Kożystamy tutaj z następujących tożsamości

- `~(a & b)` jest równoważne zapisowi `~a | ~b`;
- `~exists(x,Z)` jest równoważne zapisowi `all(x,~Z)`;
- `~all(x,Z)` jest równoważne zapisowi `exists(x,~Z)`.

Etap 3. Skolemizacja

W etapie tym eliminujemy kwantyfikator szczegółowy. Dokonuje się to przez wprowadzenie nowych symboli stałych, **stałych Skolema**, w miejsce zmiennych wprowadzanych za pomocą tych kwantyfikatorów. Zamiast bowiem mówić, że istnieje obiekt o pewnych własnościach, można utworzyć taki obiekt i nadać jemu nazwę. Przykładowo

```
exists(X, a(X) & b(X))
```

zostaje przekształcone na

```
a(g103) & b(g103)
```

gdzie `g103` to pewna niepowtarzalna stała.

Etap 4. Przesunięcie kwantyfikatorów ogólnych na zewnątrz

W tym etapie kwantyfikatory ogólne przenosimy poza wyrażenie. Przykładowo

```
all( X, a(X) -> all( Y,b(Y) -> c(X,Y) ) )
```

zostaje przekształcone na

```
all( X,all( Y,a(X) -> ( b(Y) -> c(X,Y) ) ))
```

Dalej, ponieważ kwantyfikatory same w sobie nie niosą już żadnych dodatkowych informacji, to pamiętając, że wszystkie zmienne są przez nie wprowadzane, możemy je usunąć. Przykładowo

```
all(X,a(X) | b(X)) & all(Y,c(d,Y) | d(Y))
```

zostaje przekształcone na

```
(a(X) | b(X)) & (c(d,Y) | d(Y))
```

Etap 5. Wyprowadzenie koniunkcji poza alternatywę

Teraz dążymy do zapisu wyrażenia w **koniunkcyjnej postaci normalnej**. W tej postaci wyrażenie nasze jest zbiorem koniunkcji, w którym elementy łączone są albo literałami albo literałami połączonymi `|`. Korzystamy tutaj z następujących tożsamości

- `( A & B ) | C` jest równoważne zapisowi `( A | C ) & ( B | C )`;
- `( A & C ) | ( B & C )` jest równoważne zapisowi `( A | B ) & C`.

Przykładowo

```
a(X) | b(c,X) & ( d(c) | e(c) )
```

zostaje przekształcone na

```
( a(X) | b(c,X) ) & ( a(X) | ( d(c) | e(c) ) )
```

Etap 6. Przekształcanie w klauzule

Wyrażenie z jakim mamy do czynienia, jest zbiorem koniunkcji obejmującym albo literały albo literały połączone alternatywą. Traktując chwilowo literały połączone alternatywą jak jeden literał, otrzymujemy wyrażenie postaci np.

```
(A & B) & (C & (D & E))
```

W tym momencie zupełnie nie ma znaczenia jak rozmieszczone są nawiasy i jaka jest kolejność literałów. Tak więc możemy

- pozbyć się wszystkich nawiasów;
- poprzestawiać literały według naszego upodobania;
- wiedząc, że na tym poziomie rozważań są same koniunkcje możemy się ich pozbyć, zapisując wyrażenie jako zbiór `{A,B,C,D,E}`. Wyrażenia umieszczone w tym zbiorze nazywamy **klauzulami**. Zauważmy, że ponieważ jest to zbiór wiec automatycznie zupełnie nie ma znaczenia kolejność elementów.

Klauzule są albo literałami albo literałami połączonymi alternatywą. Tak więc ponownie jak to miało miejsce dla koniunkcji kolejność literałów nie ma znaczenia i możemy pozbyć się nawiasów. Tak więc całe wyrażenie możemy zapisać jako klauzulę będącą zbiorem literałów domyślnie połączonych alternatywą.

Tym samym otrzymujemy wyrażenie w postaci klauzulowej. Odpowiada jemu zestaw klauzul, z których każda jest zestawem literałów. Literał natomiast jest albo formułą atomową albo zanegowaną formułą atomową.

3.5 Zapis klauzul

Jak wiemy klauzula skład się z pewnej ilości literałów, z których część może być zanegowana. Przyjmujemy teraz następującą umowę dotyczącą zapisu klauzul.

- najpierw zapisujemy literały niezanegowane, potem zanegowane;
- grupę literałów niezanegowanych od literałów zanegowanych rozdzielamy symbolem `:-`;
- literały niezanegowane rozdzielamy znakiem średnika;
- literały zanegowane zapisujemy bez znaku negacji i rozdzielamy je przecinkami.

Zapis ten, choć może wydawać się dziwny ma następujące uzasadnienie.
Klauzula:

```
{a1,~b3,a2,a4,~b1,a3,~b2}
```

odpowiada zapisowi:

```
( a1 | ~b3 | a2 | a4 | ~b1 | a3 | ~b2 )
```

co jest równoważne z:

```
(a1 | a2 | a3 | a4) | (~b1 | ~b2 | ~b3)
```

co jest równoważne z:

```
(a1 | a2 | a3 | a4) | ~(b1 & b2 & b3)
```

co jest równoważne z:

```
(b1 & b2 & b3) -> (a1 | a2 | a3 | a4)
```

Przekształcamy w ten sposób pewne wyrażenie, np.:

```
{a1,~b3,a2,a4,~b1,a3,~b2}
```

do postaci występującej np. w Prologu:

```
(a1 ; a2 ; a3 ; a4) :- (b1 , b2 , b3)
```

3.6 Rezolucja

Doszliśmy do momentu, w którym wyrażenia rachunku predykatów, potrafimy zapisać w jeden zunifikowany sposób. Tym samym jesteśmy w stanie reprezentować w jednolity sposób większy zbiór przesłanek, lub faktów opisujących świat. Faktów stwierdzających, że tak a nie inaczej jest i nie podlega to dyskusji. Stąd nazywać będziemy je także aksjomatami. W oparciu o zbiór faktów można spróbować wyciągać wnioski (dowodzić twierdzenia) – i właśnie tym zajmiemy się w dalszej części.

Narzędziem wykorzystywanym przy automatycznym dowodzeniu twierdzeń jest zasada rezolucji. Rezolucja pozwala przekształcać wyrażenia zapisane w postaci klauzulowej. Mając dwie odpowiednio powiązane ze sobą klauzule, na ich podstawie generujemy nową będącą wnioskiem z dwóch poprzednich. Zasada jest taka, że jeśli po lewej stronie jednej klauzuli i po prawej stronie drugiej jest takie samo wyrażenie atomowe, to można wówczas jako wniosek wygenerować klauzulę powstałą przez sklejenie tych dwóch klauzul bez powtarzającego się wyrażenia. Na przykład

Z klauzul postaci:

```
a(x);b(x) :- c(y),d(y).
e(x) :- b(x),f(x).
```

wnioskujemy, że:

```
a(x);e(x) :- c(y),d(y),f(x).
```

Podany przykład wygląda prosto, ale zauważmy, że jest to najprostsza postać problemu, który możemy skomplikować

- używając zmiennych w klauzulach;
- dopasowaniu może podlegać wiele literałów a nie tylko jeden.

Spójrzmy teraz na przykład rezolucji ze zmiennymi.

```
(1) a(f(X));b(X) :- .
(2) b(Y) :- c(f(Y),Y).
(3) c(Z,m) :- a(Z).
```

Stosując rezolucję do klauzuli drugiej i trzeciej

```
b(Y) :- c(f(Y),Y).
        c(Z   ,m) :- a(Z).
```

mamy następujący ciąg dopasowań

```
Y przyjmie wartość m
f(Y) przyjmie wartość f(m)
Z przyjmie wartość f(m)

b(Y) przyjmie wartość b(m)
a(Z) przyjmie wartość a(f(m))
```

Zatem ostatecznie otrzymujemy

```
(4) b(m) :- a(f(m)).
```

Stosując rezolucję do klauzuli pierwszej i nowo otrzymanej czwartej otrzymamy klauzulę piątą

```
(1)          a(f(X));b(X) :- .
(4) b(m) :- a(f(m)).

(5) b(m);b(m) :- .
```

Przedstawiony proces dopasowywania czasem nazywa się unifikacją choć nie zawsze oba te terminy znaczą dokładnie to samo. Oczywiście rezolucję można powtarzać dalej z nadzieją, że w końcu natrafimy na poszukiwany przez nas wniosek. Niestety nie można zagwarantować, że się nam to uda, nawet jeśli szukane twierdzenie wynika z przesłanek.

Ważną własnością rezolucji jest to, że jest ona *całkowicie falsyfikowalna*. Oznacza to, że jeśli zbiór klauzul jest niespójny (sprzeczny), to metodą rezolucji można z niego wyprowadzić klauzulę pustą postaci `:-` . Zbiór klauzul jest sprzeczny, jeśli nie istnieje możliwość uzgodnienia wszystkich predykatów, symboli stałych i symboli funkcyjnych, aby uzyskać jedynie same prawdziwe twierdzenia (wnioski).

Czytelnikowi zapewne doskonale jest znana reguła wnioskowania *modus tollens* (łac. *modus tollendo tollens*, sposób zaprzeczający przy pomocy zaprzeczenia) – wnioskowanie logiczne, reguła logiki mówiąca, że jeśli zaakceptujemy że z X wynika Y, oraz że Y jest fałszywe, to musimy zaakceptować też fałszywość X.

$$[(X \to Y) \wedge \neg Y] \to \neg X.$$

Przekładając to na język reguł możemy powiedzieć, że jeśli teza $\{A_1, \ldots, A_n\}$ nie jest sprzeczna to formuła B jest jej wnioskiem wtedy i tylko wtedy, gdy teza $\{A_1, \ldots, A_n, \neg B\}$ jest sprzeczna.

Jeśli zatem nasza hipoteza jest niesprzeczna, wystarczy do zbioru faktów dodać zanegowane klauzule tworzące naszą hipotezę. Jeśli z posiadanych przesłanek wynika hipoteza, wówczas rezolucja daje regułę pustą. Ogólnie rzecz ujmując, zarówno fakty jak i hipoteza są najzwyklejszymi klauzulami. Zbiór tych klauzul możemy oznaczyć jako $\{C_1, \ldots, C_i, C_j, \ldots, C_m\}$, gdzie $\{C_1, \ldots, C_i\} = \{A_1, \ldots, A_n\}$ natomiast $\{C_j, \ldots, C_m\} = \{B_1, \ldots, B_k\}$. W tym momencie, z punktu widzenia działania rezolucji, klauzule C_l dla $l = 1, \ldots, m$ są nierozróżnialne w tym sensie, że nie wiadomo, które są faktami a które hipotezą mającą być dopiero udowodnioną. Nie wiadomo bowiem, czy

- $\neg C_1$ wynika z $\{C_2, C_3, C_4, \ldots, C_i\}$, czy też
- $\neg C_2$ wynika z $\{C_1, C_3, C_4, \ldots, C_i\}$, czy też
- $\neg C_3$ wynika z $\{C_1, C_2, C_4, \ldots, C_i\}$, czy też ...

Oczywiście kombinacji takich istnieje bardzo dużo. Dodatkowym problemem jest ustalenie odpowiednich etapów rezolucji tak aby w wyniku stosowania aksjomatów i otrzymanych na ich podstawie wniosków, otrzymać kolejne wnioski prowadzące nas bezpośrednio do rozwiązania.

3.7 Klauzule Horna

Klauzula Horna to klauzula, w której co najwyżej jeden element jest niezanegowany. Zatem jako klauzule Horna rozpatrywać będziemy klauzule, w których

1. tylko jeden literał jest niezanegowany (tzw. klauzula Horna z głową), np. `a(X) :- b(X),c(X).`

2. wszystkie literały są zanegowane (tzw. klauzula Horna bez głowy), np. `:- a(X).`

Wszystkie problemy posiadające rozwiązanie i dające się zapisać w formie klauzul Horna cechuje to, że tylko jedna klauzula jest bez głowy. Musi ona istnieć, gdyż inaczej nie będziemy mogli wyprowadzić klauzuli pustej. Klauzula pusta nie posiada bowiem głowy a w wyniku rezolucji dwóch klauzul Horna z głową zawsze otrzymamy klauzulę z głową. W zupełności natomiast wystarcza tylko jedna klauzula bez głowy jako, że każdy dowód przez rezolucję wykorzystujący więcej niż jeden aksjomat bez głowy można przekształcić tak, aby używać co najwyżej jednej reguły bez głowy.

Załóżmy, że mając następujący zbiór faktów i reguł

```
(1) a :- b,c.
(2) c :- d.
(3) b.
(4) d.
```

chcemy udowodnić `a`. W tym celu możemy postąpić w następujący sposób.

1. Zgodnie z tym co powyżej napisano, będziemy chcieli wykorzystać sprzeczność. Tak więc dodajemy do naszego zbioru faktów i reguł zaprzeczenie `a`, czyli `:- a.`

   ```
   (1) a :- b,c.
   (2) c :- d.
   (3) b.
   (4) d.
   (5) :- a.
   ```

2. Stosując rezolucję do 1 i 5 otrzymujemy klauzule 6

   ```
   (1) a :- b,c.
   (2) c :- d.
   (3) b.
   (4) d.
   (5) :- a.
   (6) :- b,c.
   ```

   Zauważmy, że nowo otrzymaną klauzulę dodajemy do wcześniejszego zbioru faktów i reguł. Tym samym zbiór ten się powiększa o nowe fakty i reguły.

3. Stosując rezolucję do 6 i 3 otrzymujemy klauzule 7

   ```
   (1) a :- b,c.
   (2) c :- d.
   (3) b.
   (4) d.
   (5) :- a.
   (6) :- b,c.
   (7) :- c.
   ```

4. Stosując rezolucję do 7 i 2 otrzymujemy klauzule 8

```
(1) a :- b,c.
(2) c :- d.
(3) b.
(4) d.
(5) :- a.
(6) :- b,c.
(7) :- c.
(8) :- d.
```

5. Stosując rezolucję do 8 i 4 otrzymujemy klauzule 9

```
(1) a :- b,c.
(2) c :- d.
(3) b.
(4) d.
(5) :- a.
(6) :- b,c.
(7) :- c.
(8) :- d.
(9) :-
```

Otrzymaliśmy więc klauzulę pustą, co oznacza, że po dodaniu zaprzeczenia `a` zbiór faktów i reguł stał się sprzeczny. Zatem wcześniej musiał być niesprzeczny, czyli `a` musiało być prawdą.

Przedstawionemu procesowi odpowiada następującej sekwencja poszukiwań języka Prolog.

1. Potraktujmy `a` jako cel.

2. Znajdź fakt lub regułę o `a`. Pierwszą na jaką natrafiamy jest `a :- b,c.`, która nakazuje udowodnić najpierw `b` a potem `c`.

3. Znajdujemy fakt dotyczący `b`.

4. Znajdź fakt lub regułę o `c`. Pierwszą na jaką natrafiamy jest `c :- d.`, która nakazuje udowodnić najpierw `d`.

5. Znajdujemy fakt dotyczący `d`.

Udowodnione zostało zatem wszystko, co potrzebne jest do wykazania prawdziwości `a` a więc `a` jest prawdziwe.

W ogólności stosując rezolucję możemy udowodnić wszystko to, co możemy pokazać stosując mechanizm nawracania Prologu. W drugą stronę nie jest to już jednak prawdą. Powodem tego jest chociażby wzrastająca, wraz z postępowanie procesu rezolucji, liczba dostępnych faktów i reguł.//tutu// więcej napisać!!!!

3.8 Co z tego wynika?

Staraliśmy się pokazać w tym rozdziale jak rachunek predykatów można wykorzystać jako nośnik wiedzy. Co ważniejsz, chcieliśmy pokazać, że nośnik taki nie tylko jest jakąś fanaberią, ale w istocie pozwala na operowanie przechowywaną wiedzą. Istotne jest także to, że daje się go sprowadzić do pewnych uniwersalnych i jednolitych w swej konstrukcji wyrażeń. Dzięki temu można zautomatyzować proces posługiwania się wiedzą w tej postaci, tworząc choćby odpowiedni język programowania. Jednym z takich języków jest Prolog.

//tutu// krotko opisac prolog

Zaliczany jest on do języków logicznych, a więc języków opisujących obiekty i związki między nimi a także potrafiących na tej podstawie wnioskować

Prologowi poświęcony został rozdział 46.

Na zakończenie zasygnalizujmy jedynie, że podany sposób reprezentacji wiedzy jest jednym z wielu możliwych i w wielu przypadkach nie jest on wystarczajacy:

- *Jest ciepło.* Co to znaczy: *ciepło*? Ile to jest stopni? (problem reprezentacji wartości ciągłych i nieprecyzyjnych w systemach ze swej natury dyskretnych)
- *Blondyni często mają niebieskie oczy.* Co to znaczy: *często*?
- *Skocz do sklepu po mleko.* Do którego sklepu? Jakie mleko? Ile tego mleka? Faktycznie mam skakać? (problem niekompletności bazy wiedzy)
- *Pracownicy IBM-a wierzą, że Deep Blue wygra, ale ja myśle, że przegra.* Jak reprezentować uczucia, wierzenia intuicję tak aby powiązać je w jeden spójny system.

Tych kilka przykładów sugeruje, że jest jeszcze wiele obszarów wiedzy, które musimy nauczyć się reprezentować w efektywny sposób. O niektórych z nich mówić będziemy w kolejnych rozdziałach //tutu//.

Rozdział 4

Podstawowe informacje

W rozdziale tym zebrano wszystkie pojęcia „pomocnicze", jakie pojawiają się na kartach tego podręcznika, a które wymagają wyjaśnienia lub komentarza.

4.1 Graf

4.2 Drzewo

W informatyce pod pojęciem drzewa rozumiemy spójny skierowany graf acykliczny. Wprowadza się następujące terminy:

- krawędzie grafu nazywane są **gałęziami**;
- wierzchołki, z których wychodzi co najmniej jedna krawędź to **węzły**, pozostałe zaś określane są mianem **liści**. Przyjmuje się, że wierzchołek ma co najwyżej jedną krawędź wchodzącą;
- węzeł lub liść do którego nie prowadzą żadne krawędzie to **korzeń**.

4.3 Podstawowe pojęcia z zakresu rachunku prawdopodobieństwa

Niech (Ω, P) będzie przestrzenią zdarzeń elementarnych z prawdopodobieństwem P.

Definicja 4.1. Zmienną losową *nazywamy funkcję X odwzorowująca zbiór Ω wyników pewnego doświadczenia losowego zbiór liczb rzeczywistych*

$$X : \Omega \to R.$$

Definicja 4.2. Zmienna losowa skokowa *(zmienna losowa dyskretna) to zmienna losowa X przyjmująca co najwyżej przeliczalną liczbę wartości $x_1, x_2, \ldots$ odpowiednio z dodatnimi prawdopodobieństwami $p_1, p_2, \ldots$ takimi, że $\sum_{i=1}^{\infty} p_i = 1$.*

Definicja 4.3. Rozkładem prawdopodobieństwa zmiennej losowej skokowej *nazywamy zbiór par (x_i, p_i), gdzie x_i, $i = 1, 2, \ldots$ są wartościami jakie przyjmuje zmienna losowa zaś p_i jest prawdopodobieństwem z jakim ta zmienna losowa przyjmuje wartość x_i.*

Definicja 4.4. Wartość oczekiwaną[1] **zmiennej losowej** X *nazywamy liczbę* EX *określoną dla skokowej zmiennej losowej o skończonym zbiorze wartości w następujący sposób:*

$$EX = \sum_{i=1}^{n} x_i p_i, \tag{4.1}$$

gdzie n *jest licznością zbioru wartości,* x_i *są wartościami a* p_i *oznaczają prawdopodobieństwo* $P(X = x_i) = p_i$.

Własność 4.1. *Jeżeli istnieje* EX*, to dla każdej liczby rzeczywistej* c *istnieje* $E(cX)$

$$E(cX) = cEX.$$

Przykład 4.1. Obliczanie wartości oczekiwanej

Niech X *będzie liczbą oczek wyrzuconych w rzucie kostką. Wówczas* $P(X = x_i) = \frac{1}{6}$ *dla* $i = 1, \ldots, 6$*. Zatem wartość oczekiwana zmiennej losowej* X *wynosi*

$$1 \cdot \frac{1}{6} + \cdots + 6 \cdot \frac{1}{6} = 3.5$$

Definicja 4.5. *Wartość oczekiwaną zmiennej losowej postaci* $Y = (X - EX)^2$ *oznaczamy przez* D^2X *i nazywamy* **wariancją zmiennej losowej** X.

Wariancja zmiennej losowej jest wartością oczekiwaną kwadratów odchyleń wartości tej zmiennej losowej od jej warości oczekiwanej. Wariancja zmiennej losowej X *jest parametrem charakteryzującym stopień rozproszenia wartości zmiennej losowej* X *na osi liczbowej względem* EX*. Dowodzi się, że*

$$D^2X = E(X^2) - (EX)^2.$$

Własność 4.2. *Jeżeli istnieje* D^2X*, to dla każdej liczby rzeczywistej* c *istnieje* $D^2(cX)$ *oraz zachodzi*

$$D^2(cX) = c^2 D^2X.$$

Definicja 4.6. *Pierwiastek kwadratowy z wariancji zmiennej losowej nazywa się* **odchyleniem standardowym** *zmiennej losowej* X *i oznacza się go przez* σ_X*:* $\sigma_X = \sqrt{D^2X}$.

Przykład 4.2. Obliczanie wariancji

Kontynuując poprzedni przykład (4.1) mamy, że zbiorem wartości zmiennej losowej X^2 *jest zbiór* $\{1^2, \ldots, 6^2\}$ *oraz* $P(X^2 = i^2) = \frac{1}{6}$*, dla* $i = 1, \ldots, 6$*. Zatem wariancja zmiennej losowej* X *równa się:*

$$D^2X = E(X^2) - (EX)^2 = \frac{1}{6}(1^2 + \cdots + 6^2) - (3.5)^2 = \frac{35}{12} \approx 2.92$$

Odchylenie standardowe zmiennej losowej X *jest równe* $\sigma_X = \sqrt{\frac{35}{12}} \approx 1.7$.

[1] A także: wartością średnią, wartością przeciętną, nadzieją matematyczna.

Twierdzenie 4.1 (Centralne twierdzenie graniczne)**.** *tutu*

Mówiąc bardziej obrazowo, centralne twierdzenie graniczne mówi, że jeśli X_i są niezależnymi zmiennymi losowymi o jednakowym rozkładzie, takiej samej wartości oczekiwanej μ i skończonej wariancji σ^2, to gdy n dąży do nieskończoności, wówczas zmienna losowa postaci

$$\frac{\sum_{i=1}^{n} X_i - n\mu}{\sigma\sqrt{n}}$$

zbiega według rozkładu do standardowego rozkładu normalnego.

Definicja 4.7. Kowariancję *dwóch zmiennych lososwych X i Y o wartościach oczekiwanych $E(X) = \bar{(}x)$, $E(Y) = \bar{(}y)$ definiujemy jako*

$$cov(X, Y) = E((X - \bar{x})(Y - \bar{y})).$$

Powyższe zapisać można także jako:

$$cov(X, Y) = E(X \cdot Y) - \bar{x}\bar{y}$$

lub

$$cov(X, Y) = E(X \cdot Y) - \bar{x}E(Y) - \bar{y}E(X) + \bar{x}\bar{y}.$$

Jeżeli między zmiennymi losowymi X i Y nie istnieje żadna zauważalna korelacja liniowa, a istnieją ich wartości oczekiwane, to kowariancja przyjmuje wartość 0. Wartości kowariancji zbliżone, czy nawet równe zero nie świadczą jednak o całkowitej niezależności zmiennych losowych. Zawsze istnieje bowiem możliwość, że są one zależne nieliniowo. Jako bardzo prosty przykład potwierdzający ostatnie zdanie, rozważmy przedział liczb rzeczywistych $t \in [0, 2\pi]$, a zmienne losowe określmy jako:

$$X = \sin(t), \quad Y = \cos(t).$$

W tym przypadku, pomimo ich oczywistej zależności (jedynka trygonometryczna) mamy $cov(X, Y) = 0$.

W teorii prawdopodobieństwa i statystyce, kowariancja stanowi miarę tego czy i jak dwie zmienne zmieniają się razem. Jeśli zmieniają się razem wtedy wariancja może być:

- *dodatnia, co oznacza, że obie wartości jednocześnie rosną;*
- *ujemna, co oznacza, że gdy jedna z wartości rośnie, to druga maleje.*

Definicja 4.8. *Funkcję F określoną na zbiorze liczb rzeczywistych wzorem*

$$F(t) = P(X \leq t)$$

nazywamy **dystrybuantą zmiennej losowej** *X.*

Własność 4.3. *Dystrybuanta jest funkcją niemalejącą, prawostronnie ciągłą, w $-\infty$ ma granicę równą 0, a w $+\infty$ granicę równą 1.*

Definicja 4.9. *Gęstością rozkładu zmiennej losowej X nazywamy nieujemną funkcję f, dla której*

$$\int_{-\infty}^{\infty} f(x)dx = 1.$$

Definicja 4.10. Rozkładem normalnym *nazywamy rozkład zmiennej losowej X o gęstości określonej wzorem:*

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-m)^2}{2\sigma^2}},$$

gdzie m, $\sigma > 0$ są liczbami rzeczywistymi będącymi parametrami rozkładu. Zapis $X : N(m, \sigma)$ czytamy: zmienna losowa X o rozkładzie normalnym z parametrami m i σ. Jeśli $X : N(m, \sigma)$ to wartość oczekiwana i wariancja zmiennej losowej X wynoszą odpowiednio: $EX = m$, $D^2X = \sigma^2$ (czyli σ jest odchyleniem standardowym zmiennej X). Funkcja gęstości zmiennej losowej o rozkładzie naturalnym osiąga maksimum globalne dla $x = m$ i wynosi ono $\frac{1}{\sigma\sqrt{2\pi}}$.

Dystrybuanta zmiennej losowej o rozkładzie normalnym i parametrach m i σ jest określona wzorem

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-(t-m)^2}{2\sigma^2}} dt.$$

Definicja 4.11. *Gęstością rozkładu $X : N(0, 1)$ (rozkład normalny o parametrach $m = 0$ i $\sigma = 1$) jest funkcja*

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}. \tag{4.2}$$

Dystrybuantą rozkładu $X : N(0, 1)$ nazywamy funkcję

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt. \tag{4.3}$$

Definicja 4.12. *Całka z funkcji gęstości rozkładu normalnego nosi nazwę funkcji błędu lub normalnej całki błędu, oznaczamy ją jako $erf(\cdot)$ i obliczamy zgodnie ze wzorem*

$$erf(x) = \frac{1}{\sqrt{2\pi}} \int_{-x}^{x} e^{-\frac{y^2}{2}} dy. \tag{4.4}$$

Własność 4.4. *Gdy $x \to \infty$, wtedy*

$$1 - \Phi(x) \sim \frac{1}{\sqrt{2\pi}x} e^{-\frac{x^2}{2}}. \tag{4.5}$$

Znak $\sim$ oznacza, że stosunek wyrażeń po obu stronach tego znaku dąży do jedności. Dowód znaleźć można np. w [5], str. 132.

Twierdzenie 4.2. *Dla funkcji błędu zachodzi*

$$erf(x) = 1 - 2\,(1 - \Phi(x)) \tag{4.6}$$

Dowód. Z definicji funkcji $erf(\cdot)$ mamy

$$\begin{aligned} erf(x) &= \frac{1}{\sqrt{2\pi}} \int_{-x}^{x} e^{-\frac{y^2}{2}} dy = \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{y^2}{2}} dy - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-x} e^{-\frac{y^2}{2}} dy - \frac{1}{\sqrt{2\pi}} \int_{x}^{\infty} e^{-\frac{y^2}{2}} dy = \\ &= 1 - 2\frac{1}{\sqrt{2\pi}} \int_{x}^{\infty} e^{-\frac{y^2}{2}} dy = \\ &= 1 - 2\Phi(-x) = 1 - 2\left(1 - \Phi(x)\right). \end{aligned}$$

□

4.4 Algebra

4.4.1 Wektory i wartości własne macierzy

Przekształcenia liniowe przestrzeni wektorowej takie jak np. obrót, //tutu// można interpretować graficznie pokazując ich wpływ na wektory (pokazując jak zmieniają się wektory). Stanowią one funkcje wektorowe. Możemy powiedzieć, że w przestrzeni wektorowej L definiujemy funkcję wektorową A jeśli dla każdemu wektorowi $x \in L$ odpowiada tylko jeden wektor $y \in L$ taki, że $y = A(x)$ co często zapisywane jest jako $y = Ax$. Funkcja wektorowa A jest liniowa jeśli spełnione są warunki:

- (ang. *additivity*): $A(x + y) = Ax + Ay$,
- (ang. *homogeneity*): $A(\alpha x) = \alpha Ax$,

gdzie x i y są dowolnymi wektorami z przestrzeni L a α jest wartością skalarną. Taka funkcja nazywana jest transformacją liniową (ang. *linear transformation*), operatorem liniowym (ang. *linear operator*) lub liniowym endomorfizmem (ang. *linear endomorphism*) przestrzeni L.

Niech A będzie przekształceniem liniowym. Niezerowy wektor x nazywamy **wektorem własnym** przekształcenia (macierzy) A jeśli spełnione jest równanie

$$Ax = \lambda x \tag{4.7}$$

dla pewnej wartości skalarnej λ. Współczynnik λ nazywamy wartością własną przekształcenia (macierzy) A odpowiadającą wektorowi własnemu x.

Własności

- Jedna wartość własna może być związana z kilkoma lub nawet nieskończoną ilością wektorów własnych. Jeśli jednak wektor własny jest określony, wówczas odpowiadająca jemu wartość własna określona jest jednoznacznie.
- Geometrycznie równanie (4.7) oznacza, że transformacja A wpływa tylko na długość wektora x i jego zwrot. Wartość własna jest miarą tego o ile należy wektor „rozciągnąc" lub „skrócić".

- Jeśli x jest wektorem własnym przekształcenia A z wartością własną λ, wówczas wektor $y = \alpha x$ także jest wektorem własnym z taką samą wartością własną. A zatem o tym czy dany wektor jest wektorem własnym nie decyduje jego długość czy zwrot, ale jego kierunek. Z tego powodu, najczęściej posługujemy się wektorami własnymi znormalizowanymi do długości równej 1.
- Wektory własne odpowiadające różnym wartościom własnym są liniowo niezależne. W szczególności oznacza to, że w przestrzeni n-wymiarowej transformacja A nie może mieć więcej niż n wektorów własnych z różnymi wartościami własnymi.
- Wektory własne istnieją tylko dla macierzy kwadratowych, ale nie każda taka macierz wektory własne posiada.
- Macierz symetryczna ma wartość własną.
- Dla macierzy kwadratowej wymiaru $n \times n$ istnieje n wektorów własnych.
- Wszystkie wektory własne macierzy A są ortogonalne (prostopadłe do siebie). Oznacza to, że w rozważanej przestrzni otrzymujemy drugi, alternatywny, układ współrzędnych w którym możemy wyrazić rozpatrywane wektory (patrz rysunek //tutu//).
- Wartości własne macierzy są pierwiastkami jej wielomianu charakterystycznego

$$det(A - I\lambda) = 0$$

4.4.2 Określoność macierzy

Macierz określona dodatnio

Rzeczywista macierz A jest macierzą **dodatnio określoną**, jeśli jest symetryczna i dla każdego niezerowego wektora $x \in R^n$ zachodzi

$$xAx > 0$$

Równoważnie

- Macierz A jest dodatnio określona jeśli wszystkie wartości własne macierzy A są dodatnie.
- Macierz A jest dodatnio określona jeśli wyznaczniki wszystkich minorów głównych macierzy A są dodatnie.

Macierz dodatnio określona jest zawsze odwracalna i jej odwrotność jest również dodatnio określona. Jeśli A i B są dodatnio określone, to ich suma $A + B$ także jest dodatnio określona.

4.5 Analiza matematyczna

4.5.1 Wzór Taylora

Definicja 4.13 (Wzór Taylora)**.** *Wzór Taylora pozwala na przedstawienie funkcji $n+1$-razy różniczkowalnej przy pomocy wielomianu zależnego od kolejnych jej pochodnych oraz*

dostatecznie małej reszty. Mówiąc precyzyjniej, jeśli założymy, że Y jest przestrzenią unormowaną oraz $f : [a, b] \to Y$ jest funkcją $n+1$-razy różniczkowalną w sposób ciągły, wówczas dla każdego $x \in (a, b)$ mamy:

$$f(x) = f(a) + \frac{x-a}{1!}f^1(a) + \frac{(x-a)^2}{2!}f^2(a) + \cdots + \frac{(x-a)^n}{n!}f^n(a) + R_n(x, a)$$

gdzie $R_n(x, a)$ spełnia warunek

$$\lim_{x \to a} \frac{R_n(x, a)}{(x-a)^n} = 0$$

$R_n(x, a)$ nazywane jest resztą (Peano) we wzorze Taylora. Jeśli $a = 0$, to wzór Taylora nazywany jest wzorem Maclaurina.

4.5.2 Przestrzeń unitarna

Definicja 4.14. Przestrzenią unitarną *nazywamy rzeczywistą przestrzeń liniową V nad ciałem liczb rzeczywistych K wraz z określonym w niej funkcjonałem dwuliniowym f spełniającym następujące warunki:*

- **symetria***: dla wszystkich x, y należących do V zachodzi*

$$f(x, y) = f(y, x)$$

- **liniowość** *ze względu na pierwszą zmienną: dla wszystkich x, y należących do V oraz dowolnego a należącego do K zachodzi*

$$f(ax) = af(x)$$

$$f(x + y, z) = f(x, z) + f(y, z)$$

- **dodatnia określoność***:*

$$f(x, x) > 0, \text{dla} x \neq 0$$

Funkcjonał $f(\cdot, \cdot)$ nazywa się **iloczynem skalarnym** *lub* **iloczynem wewnętrznym** *wektorów x i y i oznaczany jest jako $\langle \cdot, \cdot \rangle$.*

Najprostszym przykładem przestrzeni unitarnej jest przestrzeń liczb rzeczywistych ze (standardowym) iloczynem skalarnym zdefiniowanym jako

$$\langle x, y \rangle = x \cdot y$$

co czasem, jeśli nie prowadzi to do niejednoznaczności, zapisujemy także jako xy.

Ogólniej, w przestrzeni euklidesowej R^n jeśli $u = (u_1, \ldots, u_n)$ oraz $v = (v_1, \ldots, v_n)$ standardowy iloczyn skalarny określony jest wzorem

$$u \cdot v = u_1 v_1 + \cdots + u_n v_n.$$

Ze względu na własności iloczynu skalarnego, funkcja $\| \cdot \|$ taka, że

$$\|u\| = \sqrt{\langle u, u \rangle}$$

spełnia aksjomaty normy. Normę tę nazywamy **normą generowaną przez iloczyn skalarny**. Z tego też względu każda przestrzeń unitarna jest także unormowana. W tym przypadku, norma interpretowana jest jako długość wektora u.

Interpretacja geometryczna iloczynu skalarnego

W przestrzeni euklidesowej istnieje silna zależność między iloczynem skalarnym a długością i kątem. Dla wektora u, $u \cdot u$ jest kwadratem jego długości. Ogólniej, jeśli v jest innym wektorem, wówczas

$$u \cdot v = |u||v| \cos \alpha,$$

gdzie $|u|$, $|v|$ oznaczają długość wektorów u oraz v a α jest kątem między nimi. Ze względu na to, że norma wektora jest jego długością, często wzór ten występuje w postaci

$$u \cdot v = \|u\|\|v\| \cos \alpha.$$

Iloczyn $|u| \cos \alpha$ jest rzutem skalarnym wektora u na v, tj. odcinkiem o początku w punkcie p_1 będącym początkiem wektora v i końcu w punkcie $p_2 = p_1 + (|u| \cos \alpha)v$ znajdującym się na półprostej $p_1 + \eta v$, gdzie $\eta \in R_+$.

4.5.3 Nierówność Cauchy'ego-Schwarza

Nierówność Cauchy'ego-Schwarza – podstawowa nierówność dla iloczynu skalarnego w przestrzeni unitarnej. Nierówność ta znana jest pod wieloma innymi nazwami, mi.in. **Schwarza**, **Buniakowskiego-Schwarza** lub **Cauchy'ego-Buniakowskiego-Schwarza**.

Twierdzenie 4.3 (Nierówność Cauchy'ego-Schwarza)**.** *Niech $\langle x, y \rangle$ oznacza iloczyn skalarny wektorów x, y danej przestrzeni unitarnej. Zachodzi następująca nierówność*

$$|\langle x, y \rangle|^2 \leq \langle x, x \rangle \langle y, y \rangle$$

lub, wyrażona za pomocą norm,

$$|\langle x, y \rangle| \leq \|x\|\|y\|.$$

Rozdział 5

Trochę klasyki

Zanim przejdziemy do „inteligentnych" tematów poświęcimy troszkę czasu klasycznym (numerycznym) sposobom rozwiązywania pewnych problemów. Czynimy tak, ze względu na ogólne zalecaną metodologię postepowania. Otóż, jeśli tylko znane jest analityczne rozwiązanie problemu działające w sposób nas satysfakcjonujący i dające wyniki w akceptowalnym przez nas czasie radzi się je wybierać. Gdy takowe nie istnieje, dopiero wtedy można próbować innych metod. Związane jest to z wieloma, jak będziemy mieli jeszcze niejednokrotnie okazję przekonać się, niewiadomymi towarzyszącymi „inteligentnym algorytmom".

5.1 Przeszukiwanie grafu

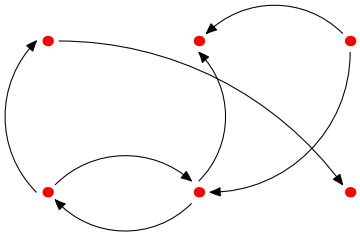
Jako pierwszy rozważymy problem przeszukiwania grafu: dla zadanego grafu G sprawdzić, czy istnieje ścieżka łącząca wierzchołki $W1$ oraz $W2$.

W zaproponowanych sposobach rozwiązania trudno doszukać się inteligentnego działania. Dają one nam jednak możliwość przećwiczenia pewnych technik algorytmiczno - programistycznych, przydatnych w kolejnych zadaniach. Poza tym sam problem przeszukiwania grafu, bliski jest problemowi odnajdywania optymalnej ścieżki łączącej dwa punkty. Ostatnie zadanie natomiast często stanowi podstawę problemów decyzyjnych, np. przy poruszaniu postacią w grze komputerowej.

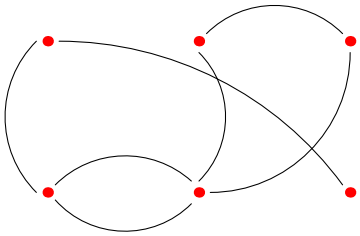
Ponadto pamiętajmy, że graf może reprezentować różne rzeczy: zarówno dwa pola na planszy jak i drzewo decyzyjne w grze czy też bardziej skomplikowaną przestrzeń poszukiwań.

5.1.1 Niezbędne definicje

Na początku podamy kilka niezbędnych definicji. **Graf skierowany** G jest opisany parą (V, E), gdzie V jest zbiorem skończonym, a E jest relacją binarną w V. Zbiór V jest nazywany **zbiorem wierzchołków** grafu G, a jego elementy nazywane są **wierzchołkami**. Zbiór E nazywamy **zbiorem krawędzi** grafu G, a jego elementy **krawędziami**. Przyjmuje się także inną równoważną definicję zbioru E, w której to określa się go jako zbiór uporządkowanych par wierzchołków. Rysunek 5.1 przedstawia przykład grafu skierowanego. **Graf nieskierowany** różni się od skierowanego sposobem konstrukcji zbioru E, który w tym przypadku jest zbiorem nieuporządkowanych par wierzchołków. Przyjmujemy, że w grafie nieskierowanym nie mogą występować pętle. Rysunek 5.2 przedstawia przykład grafu nieskierowanego. Jeśli (a, b) jest krawędzią grafu $G = (V, E)$, to mówimy,



Rysunek 5.1: Przykładowy graf skierowany.



Rysunek 5.2: Przykładowy graf nieskierowany.

że wierzchołek a jest sąsiedni do wierzchołka b (jest jego sąsiadem). Pod pojęciem **grafu nieskończonego** rozumieć będziemy graf o nieskończonej ilości wierzchołków, z których jednak każdy ma skończoną ilość sąsiadów.

5.1.2 Przechowywanie grafów

Istnieje wiele sposobów przechowywania grafu w pamięci komputera. Przedstawiamy tutaj dwa najbardziej popularne dzięki swej prostocie.

Sposób 1 – macierz sąsiedztwa

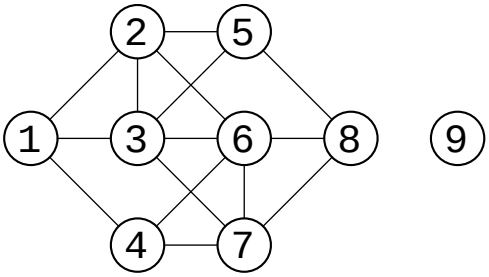
Graf przechowywany jest w tablicy bitowej wymiaru $n \times n$, gdzie n-ilość wierzchołków grafu; 1 w danej komórce oznacza istnienie połączenia między odpowiednimi wierzchołkami. Na przykład 1 w komórce (2,3) oznacza, iż wierzchołek 2 jest połączony z wierzchołkiem 3. Spójrzmy na powstałą według tego sposobu tabelę (5.1) odpowiadającą grafowi z rysunku 5.3

Sposób 2 – lista

Graf przechowywany jest z wykorzystaniem listy. Każdy wiersz stanowi zbiór numerów wierzchołków połaczonych z wierzchołkiem o numerze równym numerowi wiersza. Spójrzmy na odpowiednią dla naszego przykładu strukturę (tablica 5.2). Mamy tam między innymi nastepujący wiersz:

4	-	1	-	6	-	7

Oznacza on, iż wierzchołek 4 łaczy się z wierzchołkami 1, 6 i 7.



Rysunek 5.3: Przykładowy graf.

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	0	0	0	0	0
2	1	0	1	0	1	1	0	0	0
3	1	1	0	0	1	1	1	0	0
4	1	0	0	0	0	1	1	0	0
5	0	1	1	0	0	0	0	1	0
6	0	1	1	1	0	0	1	1	0
7	0	0	1	1	0	1	0	1	0
8	0	0	0	0	1	1	1	0	0
9	0	0	0	0	0	0	0	0	0

Tabela 5.1: Przechowywanie grafu w postaci macierzy bitowej.

1	-	2	-	3	-	4				
2	-	1	-	3	-	5	-	6	-	
3	-	1	-	2	-	5	-	6	-	7
4	-	1	-	6	-	7				
5	-	2	-	3	-	8				
6	-	2	-	3	-	4	-	7	-	8
7	-	3	-	4	-	6	-	8		
8	-	5	-	6	-	7				
9	-									

Tabela 5.2: Przechowywanie grafu z wykrzystaniem listy.

5.1.3 Rozwiązanie zadania

Istnieją dwa popularne sposoby rozwiązania postawionego zadania. Pierwszy z nich nosi nazwę **przeszukiwania grafu wszerz** drugi zaś **przeszukiwania wzdłuż** lub też **w głąb**. W rozwiązaniach tych wykorzystuje się struktury danych nazywane **stosem** (**kolejką LIFO** (ang. last in, first out)) i **listą** (**kolejką FIFO** (ang. first in, first out)). Wspomnianych struktur danych nie będziemy omawiać, odsyłając do literatury ([3], [27]).

Aby zbadać istnienie połączenia w grafie podajemy dwa wierzchołki: pierwszy – zwany **stanem początkowym**, oraz drugi – zwany **stanem końcowym**. W przypadku przeszukiwania grafów skończonych omawiane algorytmy są równoważne, tzn. zbieżność algorytmu przeszukiwania w głąb pociąga za sobą zbieżność algorytmu przeszukiwania wszerz i odwrotnie[1]. Jeżeli przeszukujemy graf nieskończony, algorytm przeszukiwania wszerz jest zbieżny, natomiast algorytm przeszukiwania w głąb nie. Brak zbieżności ma miejsce wówczas, gdy przeszukujemy krawiędź nieskończoną, która nie prowadzi do wierzchołka końcowego.

Przeszukiwanie grafu wszerz

W pierwszym kroku algorytmu odwiedzamy wszystkie wierzchołki grafu sąsiadujące z wierzchołkiem początkowym. Następnie (krok drugi) odwiedzamy wszystkie wierzchołki sąsiadujące z uprzednio odwiedzonymi. W kroku trzecim odwiedzamy wszystkie wierzchołki sąsiadujące z wierzchołkami odwiedzonymi w kroku drugim. I tak dalej. Jeśli podczas odwiedzania natrafimy na szukany wierzchołek końcowy, to oznacza to, że istnieje ścieżka łącząca wierzchołek początkowy i końcowy. W przypadku, gdy odwiedzimy wszystkie możliwe wierzchołki i nie znajdziemy wierzchołka końcowego, oznacza to, że nie istnieje droga miedzy zadanymi wierzchołkami. Odwiedzając kolejne wierzchołki należy pamiętać, żeby nie odwiedzać wierzchołków wcześniej odwiedzonych, tzn. każdy wierzchołek możemy odwiedzić dokładnie jeden raz.

Jako dane wejściowe algorytm przyjmuje graf G i dwa wierzchołki $W1$ i $W2$, o których chcemy się dowiedzieć, czy są ze sobą połączone, czy też nie.

1. Start. Utwórz kolejkę FIFO i oznacz ją jako K.
2. Dodaj $W1$ do kolejki K.
3. Czy kolejka K jest pusta? Tak – idź do 8, nie – kontynuuj.
4. Pobierz element W z kolejki K. Jeśli W był już odwiedzony idź do 3, jeśli nie to oznacz go jako odwiedzony i idź do 5.
5. Czy $W = W2$? Tak – idź do 9, nie – kontynuuj.
6. Dodaj do kolejki K wszystkie nieodwiedzone wierzchołki, z którymi W sąsiaduje a które w K jeszcze się nie znajdują.
7. Idź do 3.
8. Zwróć `BRAK_ROZWIAZANIA` i zakończ algorytm.

[1] Mówiąc o zbieżności mamy tutaj na myśli skończoność algorytmu w skończonym czasie. Inaczej mówiąc, zbieżność w tym kontekście, oznacza, że znajdziemy rozwiązanie (oczywiście jeśli tylko istnieje) w skończonym czasie.

9. Zwróć `SUKCES` i zakończ algorytm.

Tabela 5.3 zawiera kilka kroków tego algorytmu dla grafu G z rysunku 5.3 oraz wierzchołków $W1 := 1$ oraz $W2 := 8$. W przykładzie korzystać będzimy z następujących funkcji

- `MakeFIFO()` – funkcja tworzy kolejkę FIFO.
- `Push(x)` – funkcja dodaje element `x` do utworzonej kolejki.
- `Pop()` – funkcja pobiera element z kolejki (jednocześnie usuwając go z niej).

Przeszukiwanie grafu w głąb

W pierwszym kroku algorytmu odwiedzamy jeden wierzchołek grafu sąsiadujący z wierzchołkiem początkowym. Następnie (krok drugi) odwiedzamy jeden wierzchołek sąsiadujące z uprzednio odwiedzonym. W kroku trzecim odwiedzamy wierzchołek sąsiadujące z wierzchołkiem odwiedzonymi w kroku drugim. I tak dalej. Jeśli podczas odwiedzania natrafimy na szukany wierzchołek końcowy, to oznacza to, że istnieje ścieżka łącząca wierzchołek początkowy i końcowy. Jeśli odwiedziliśmy już wszystkie wierzchołki sąsiednie dla danego wierzchołka to cofamy się do wierzchołka poprzedzającego ten, w którym jesteśmy. W przypadku, gdy odwiedzimy wszystkie możliwe wierzchołki i nie znajdziemy wierzchołka końcowego, oznacza to, że nie istnieje droga miedzy zadanymi wierzchołkami. Odwiedzając kolejne wierzchołki należy pamiętać, żeby nie odwiedzać wierzchołków wcześniej odwiedzonych, tzn. każdy wierzchołek możemy odwiedzić dokładnie jeden raz.

Można powiedzieć, że przeszukiwanie grafu w głąb polega na przeszukiwaniu poszczególnych krawędzi grafu. Przechodzimy krawędz najdalej jak tylko można. Jeżeli dana ścieżka nie doprowadziła nas do wierzchołka końcowego wówczas cofamy sie do momentu, z którego możemy pójść inną scieżką.

Jako dane wejściowe algorytm przyjmuje graf G i dwa wierzchołki *W1* i *W2*, o których chcemy się dowiedzieć, czy są ze sobą połączone, czy też nie.

1. Start. Utwórz kolejkę LIFO i oznacz ją jako S.
2. Dodaj $W1$ do kolejki S.
3. Czy kolejka S jest pusta? Tak – idź do 8, nie – kontynuuj.
4. Pobierz element W z kolejki K. Jeśli W był już odwiedzony idź do 3, jeśli nie to oznacz go jako odwiedzony i idź do 5.
5. Czy $W = W2$? Tak – idź do 9, nie – kontynuuj.
6. Dodaj do kolejki S wszystkie nieodwiedzone wierzchołki, z którymi W sąsiaduje, a które w K jeszcze się nie znajdują.
7. Idź do 3.
8. Zwróć `BRAK_ROZWIAZANIA` i zakończ algorytm.
9. Zwróć `SUKCES` i zakończ algorytm.

Krok algorytmu	Stan kolejki	Operacja	Wierzchołki odwiedzone								
			1	2	3	4	5	6	7	8	9
1	Pusta	`MakeFIFO()`									
2	-1-	`Put(W1)`	x								
3	-1-		x								
4	Pusta	`W:=Get()` ($W := 1$)	x								
5	Pusta	$W! = W2$	x								
6	-2-3-4-	`Put(2)`, `Put(3)`, `Put(4)`	x	x	x	x					
7	-2-3-4-	`Goto 3`	x	x	x	x					
3	-2-3-4-		x	x	x	x					
4	-3-4-	`W:=Get()` ($W := 2$)	x	x	x	x					
5	-3-4-	$W! = W2$	x	x	x	x					
6	-3-4-5-6-	`Put(5)`, `Put(6)` Nie `Put(3)`, bo już był odwiedzony	x	x	x	x	x	x			
7	-3-4-5-6-	`Goto 3`	x	x	x	x	x	x			
3	-3-4-5-6-		x	x	x	x	x	x			
4	-4-5-6-	`W:=Get()` ($W := 3$)	x	x	x	x	x	x			
5	-4-5-6	$W! = W2$	x	x	x	x	x	x			
6	-4-5-6-7-	`Put(7)` Nie `Put(1)`, `Put(2)`, `Put(5)`, `Put(6)` bo już były odwiedzone	x	x	x	x	x	x	x		
7	-4-5-6-7-	`Goto 3`	x	x	x	x	x	x	x		
3	-4-5-6-7-		x	x	x	x	x	x	x		
4	-5-6-7-	`W:=Get()` ($W := 4$)	x	x	x	x	x	x	x		
5	-5-6-7-	$W! = W2$	x	x	x	x	x	x	x		
6	-5-6-7-	Nic nie dodajemy, bo już wszystko było	x	x	x	x	x	x	x		
7	-5-6-7-	`Goto 3`	x	x	x	x	x	x	x		
⋮		⋮									

Tabela 5.3: Przebieg wykonania algorytmu przeszukiwania wszerz grafu G z rysunku 5.3 i wierzchołków $W1 := 1$, $W2 := 8$.

Krok algorytmu	Zawartość stosu	Operacja	Wierzchołki odwiedzone								
			1	2	3	4	5	6	7	8	9
1	Pusta	`MakeLIFO()`									
2	1	`Push(W1)`	x								
3	1		x								
4	Pusta	`W:=Pop()` ($W = 1$)	x								
5	Pusta	$W! = W2$	x								
6	2,3,4	`Push(2)`, `Push(3)`, `Push(4)`	x	x	x	x					
7	2,3,4	`Goto 3`	x	x	x	x					
3	2,3,4		x	x							
4	2,3	`W:=Pop()` ($W := 4$)	x	x	x	x					
5	2,3	$W! = W2$	x	x							
6	2,3,6,7	`Push(6)`, `Push(7)` Nie `Push(1)` bo już był	x	x	x			x	x		
7	2,3,6,7	`Goto 3`	x	x	x			x	x		
3	2,3,6,7		x	x	x			x	x		
4	2,3,6	`W:=Pop()` ($W := 7$)	x	x	x			x	x		
5	2,3,6	$W! = W2$	x	x	x			x	x		
6	2,3,6,8	`Push(8)` Nie `Push(4)`, `Push(6)` bo już były	x	x	x			x	x		
7	2,3,6,8	`Goto 3`	x	x	x			x	x		
3	2,3,6,8		x	x	x			x	x		
4	2,3,6	`W:=Pop()` ($W := 8$)	x	x	x			x	x	x	
5	2,3,6	$W = W2$	x	x	x			x	x	x	
10	2,3,6	`SUKCES`	x	x	x			x	x	x	

Tabela 5.4: Przebieg wykonania algorytmu przeszukiwania w głąb grafu G z rysunku 5.3 i wierzchołków $W1 := 1$, $W2 := 8$.

Tabela 5.4 zawiera kilka kroków tego algorytmu dla grafu G z rysunku 5.3 oraz wierzchołków $W1 := 1$ oraz $W2 := 8$. W przykładzie korzystać będzimy z następujących funkcji

- `MakeLIFO()` – funkcja tworzy kolejkę FIFO.
- `Push(x)` – funkcja dodaje element `x` do utworzonej kolejki.
- `Pop()` – funkcja pobiera element z kolejki (jednocześnie usuwając go z niej).

Uwaga 5.1. Czy te algorytmy są identyczne?

Jak można zauważyć jedyna różnica w obu zaprezentowanych algorytmach polega na zmianie struktury danych, na której operujemy. „Tylko" ta zmiana pociąga za sobą diametralnie inne zachowanie obu „identycznych" algorytmów.

Uwaga 5.2. Inna wersja przeszukiwania w głąb.

1. *Start. Utwórz kolejkę LIFO i oznacz ją jako S.*
2. *Dodaj $W1$ do kolejki S.*
3. *Czy kolejka S jest pusta? Tak – idź do 8, nie – kontynuuj.*
4. *Pobierz element W z kolejki K. Jeśli W był już odwiedzony idź do 3, jeśli nie to oznacz go jako odwiedzony i idź do 5.*
5. *Czy $W = W2$? Tak – idź do 9, nie – kontynuuj.*
6. *Jeśli W jest połączony z jakimś wierzchołkiem, który nie był jeszcze odwiedzony, to dodaj W i ten wierzchołek (w takiej właśnie kolejności) do S.*
7. *Idź do 3.*
8. *Zwróć* `BRAK_ROZWIAZANIA` *i zakończ algorytm.*
9. *Zwróć* `SUKCES` *i zakończ algorytm.*

Tabela 5.5 zawiera kilka kroków tego algorytmu dla grafu G z rysunku 5.3 oraz wierzchołków $W1 := 1$ oraz $W2 := 8$.

Ćwiczenie 5.1.

Proszę napisać programy realizujące przedstawione powyżej algorytmy. Zakładamy, że program wczytuje graf z pliku o podanej w linii poleceń nazwie. Następnie pyta o numery wierzchołków do sprawdzenia. Programy powinny odczytywać niezbędne do działania dane z pliku w jednym z dwóch poniże przedstawionych formatów (proszę sobie wybrać). Maksymalna ilość wierzchołków wynosi 100 i numerujemy je kolejnymi liczbami całkowitymi od 1 do 100. Przykłady plików odpowiadające grafowi z rysunku 5.3):

`FORMAT 1 (format "bitowy")`

```
9
1 8
0 1 1 1 0 0 0 0 0
1 0 1 0 1 1 0 0 0
1 1 0 0 1 1 1 0 0
1 0 0 0 0 1 1 0 0
0 1 1 0 0 0 0 1 0
0 1 1 1 0 0 1 1 0
0 0 1 1 0 1 0 1 0
0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0
```

`FORMAT 2 (format "listowy")`

```
9
1 8
2 3 4
1 3 5 6
1 2 5 6 7
1 6 7
2 3 8
2 3 4 7 8
3 4 6 8
5 6 7
```

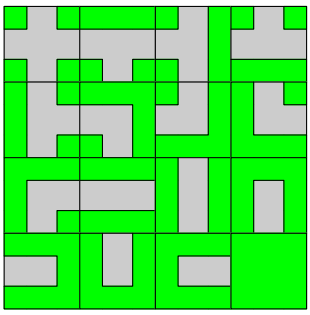
Numer linii	*Opis*
1	*Ilość wierzchołków w grafie*
2	*Węzeł początkowy (W1) i końcowy (W2)*
3, ...	*Dane w postaci „bitowej” lub „listowej”*

Programy powinien wyświetlać listę aktualnie odwiedzonych wierzchołków oraz zawartość listy lub stosu a na zakończenie wyświetlać informację, czy wierzchołki są połączone czy też nie

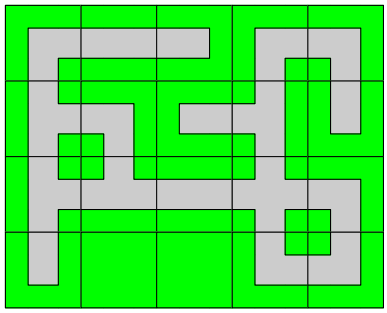
---

Krok algorytmu	Zawartość stosu	Operacja	Wierzchołki odwiedzone								
			1	2	3	4	5	6	7	8	9
1	Pusta	`MakeLIFO()`									
2	1	`Push(W1)`	x								
3	1		x								
4	Pusta	`W:=Pop()` ($W = 1$)	x								
5	Pusta	$W! = W2$	x								
6	1,2	`Push(1)`, `Push(2)`	x	x							
		(`Push(2)`, bo 2 to pierwszy	x	x							
		węzł połaczony z węzłem 1	x	x							
		wg. tabeli 5.2)	x	x							
7	1,2	`Goto 3`	x	x							
3	1,2		x	x							
4	1	`W:=Pop()` ($W := 2$)	x	x							
5	1	$W! = W2$	x	x							
6	1,2,3	`Push(2)`, `Push(3)`	x	x	x						
7	1,2,3	`Goto 3`	x	x	x						
3	1,2,3		x	x	x						
4	1,2	`W:=Pop()` ($W := 3$)	x	x	x						
5	1,2	$W! = W2$	x	x	x						
6	1,2,3,5	`Push(3)`, `Push(5)`	x	x	x		x				
7	1,2,3,5	`Goto 3`	x	x	x		x				
3	1,2,3,5		x	x	x		x				
4	1,2,3	`W:=Pop()` ($W := 5$)	x	x	x		x				
5	1,2,3	$W! = W2$	x	x	x		x				
6	1,2,3,5,8	`Push(5)`, `Push(8)`	x	x	x		x			x	
7	1,2,3,5,8	`Goto 3`	x	x	x		x			x	
3	1,2,3,5,8		x	x	x		x			x	
4	1,2,3,5	`W:=Pop()` ($S := 8$)	x	x	x		x			x	
5	1,2,3,5	$W = W2$	x	x	x		x			x	
10	1,2,3,5,8	`SUKCES`	x	x	x		x			x	

Tabela 5.5: Przebieg wykonania zmodyfikowaneg algorytmu przeszukiwania w głąb grafu G z rysunku 5.3 i wierzchołków $W1 := 1$, $W2 := 8$.



Rysunek 5.4: Zestaw pól tworzących labirynt (lewe górne pole ma numer 0, prawe dolne - 15, numeracja wierszami).



Rysunek 5.5: Przykładowy labirynt.

Ćwiczenie 5.2.

Proszę napisać program sprawdzający, w oparciu o podane algorytm, czy możliwe jest przejście w labiryncie od jednego miejsca do drugiego. Zakładamy że program wczytuje labirynt z pliku o podanej w linii poleceń nazwie. Następnie pyta się o współrzędne pola startowego i końcowego. Maksymalny rozmiar planszy to 100 wierszy i 100 kolumn. Każde pole na planszy ma numer z przedziału [0, 15]. Numer ten oznacza jekiego typu jest pole, to znaczy gdzie możemy się z niego przemieścić. Dostępne pola przedstawia rysunek 5.4 (lewe górne pole ma numer 0, prawe dolne - 15, numeracja wierszami): W pliku z opisem labiryntu pierwsza linia zawiera liczbę wierszy, druga – liczbę kolumn, kolejne natomiast to opis pól w danym wierszu. Tak więc linia 3 zawiera opis pól wiersza 1, linia 4 – opis pól wiersza drugiego, itd. Kolejne pola w wierszu rozdzielone są spacjami. Poniżej umieszczono przykładowy pliku z danymi odpowiadający labiryntowi z rysunku 5.5:

```
4
5
8 9 12 8 5
4 5 14 2 13
4 3 9 0 5
13 15 15 7 6
```

Realizacja graficzna powinna innym kolorem zaznaczac pola odwiedzone a innym pola oczekujące na odwiedzenie (czyli te z listy lub stosu). W realizacji tekstowej program powinien wyświetlać (lub zapisywać do pliku) współrzędne odwiedzanych pól.

Uwaga

Jak łatwo zauważyć labirynt można utożsamiać z grafem. Poszczególne pola labiryntu są wierzchołkami, a rodzaj pola jednoznacznie definiuje listę wieszchołków sąsiadujących z rozważanym. Dla powyższego labiryntu plik (w formacie „listowym") opisujący go jako graf wygląda następująco (numeracja pol wierszami od góry i od lewej do prawej):

```
20
2 6
1 3
2
5 9
4 10
1 7 11
6 12
9
4 8 14
5
6 12 16
7 11 13
12 14
9 13 15 19
14 20
11


14 20
15 19
```

Pytanie 5.1.

Przedstawione sposoby przechowywania grafów mają swoje wady i zalety. Jakie?

Rozdział 6

Algorytmy zachłanne

6.1 Wprowadzenie

Jedną z najmniej wyrafinowannych metod rozwiązywania problemów są metody zachłanne (ang. *greedy methods*). Dlaczego taka nazwa? Otóż strategie te wykorzystują lokalną optymalizację. Na rozwiązywany problem patrzą z lokalnego punktu widzenia, wybierając najlepsze w danym momencie rozwiązanie. Oczywiście najlepsze w danym momencie nie oznacza najlepsze w ogóle. Trzeba jednak przyznać, iż pewne zagadnienia można w ten sposób rozwiązać i, co może wydawać się dziwne, są to rozwiązania optymalne. Jednym z nich jest problem wydawania reszty.

6.1.1 Problem wydawania reszty

Dane są następujące rodzaje monet: 1 zł (13 szt.), 2 zł (7 szt.) i 5 zł (3 szt.). Należy obliczyć ile i jakich monet należy wydać, aby reszta wynosiła 13 zł a ilość monet była jak najmniejsza. Postępowanie zachłanne w tym przypadku można opisać jako wybieranie monety, która najszybciej zbliża nas do uzyskania wymaganej reszty.

1. Rozpoczęcie algorytmu. Reszta jaka pozostała do wydania: 13 zł. Ilość wydanych monet: 0.
2. Wybieramy, największą możliwą w tym przypadku, monetę o wartości 5 zł jako, że pozwala ona najszybciej zredukować pozostałą do wydania resztę do wielkości 8 zł. Ilość wydanych monet: 1 szt.
3. Wybieramy, największą możliwą w tym przypadku, monetę o wartości 5 zł jako, że pozwala ona najszybciej zredukować pozostałą do wydania resztę do wielkości 3 zł. Ilość wydanych monet: 2 szt.
4. Wybieramy, największą możliwą w tym przypadku, monetę o wartości 2 zł jako, że pozwala ona najszybciej zredukować pozostałą do wydania resztę do wielkości 1 zł. Ilość wydanych monet: 3 szt.
5. Wybieramy, największą możliwą w tym przypadku, monetę o wartości 1 zł jako, że pozwala ona najszybciej zredukować pozostałą do wydania resztę do wielkości 0 zł. Ilość wydanych monet: 4 szt.
6. Rozwiązaniem jest reszta wydana przy pomocy dwóch monet 5 zł, jednej 2 zł i jednej 1 zł.

Otrzymane w ten sposób rozwiązanie jest optymalne. Można wykazać, że dla pewnych zbiorów nominałów, np. {1, 2, 5, 10, 20, 50, 100} strategia zachłanna daje rozwiązanie optymalne. Łatwo można jednak zmienić ten problem tak, aby pokazać, że nie zawsze tak musi być.

Rozwiązanie nieoptymalne

Załóżmy, że dane są następujące rodzaje monet: 1 zł (14 szt.), 2 zł (7 szt.), 5 zł (3 szt.), 9 zł (2 szt.) i 10 zł (2 szt.). Należy obliczyć ile i jakich monet należy wydać, aby reszta wynosiła 14 zł a ilość monet była jak najmniejsza.

Gdy dobór pierwszej monety będzie zachłanny, algorytm wybierze jedną monetę 10 zł (taki wybór pozwala najszybciej zredukować pozostałą do wydania sumę). W związku z tym pozostanie do wydania 4 zł. Postępując dalej z duchem algorytmu zachłannego, najszybciej redukujemy resztę do wartości 2 zł jeśli wybierzemy monetę o nominale 2 żł. W ostatnim kroku pozostaje wybrać monetę 2 zł. Tak więc otrzymaliśmy resztę w postaci jednej monety 10 zł i dwóch 2 zł podczas, gdy identyczną resztę można otrzymać za pomocą jednej monety 9 zł i jednej 5 zł.

Porażka algorytmu zachłannego

Dane są następujące rodzaje monet: 2 zł (3 szt.) i 5 zł (3 szt.). Należy obliczyć ile i jakich monet należy wydać, aby reszta wynosiła 6 zł a ilość monet była jak najmniejsza.

Gdy dobór pierwszej monety będzie zachłanny, algorytm wybierze jedną monetę 5 zł (taki wybór pozwala najszybciej zredukować pozostałą do wydania sumę) zamiast 2 zł. Jednak już w następnym kroku okazuje się, że droga zachłanna była w tym przypadku drogą ślepą — wartość jaka pozostała do wydania to 1 zł podczas gdy najmniejszą dostępną monetą jest moneta o wartości 2 zł.

6.1.2 Problem plecakowy

Dyskretny problem plecakowy

Danych jest n przedmotów przy czym i-ty przedmiot wart jest c_i i waży w_i. Zadanie: do plecaka o nośności W należy zapakować jak najcenniejszy ładunek. Takie sformułowanie problemu, nazywane dyskretnym problemem plecakowym, bliższe jest rzeczywistości, gdyż nie dopuszczamy części „ułamkowych" produktów, co ma miejsce w ciągłym problemie plecakowym.

Dla ciągłego problemu plecakowego algorytm zachłanny jest optymalny. Najcenniejszy ładunek zabierzemy wtedy, gdy najpierw będziemy plecak napełniać wybierając najdroższy przedmiot. Gdy się on skończy a plecak nadal nie będzie pełny, wybieramy drugi co do wartości produkt i probujemy nim zapełnić plecak, itd.

Dla problemu dyskretnego algorytm zachłanny, podobnie jak to miało miejsce w przypadku monet, może ale często nie daje optymalnego rozwiązania. Jeśli np. ustalimy pojemność plecaka na $W = 50$ oraz założymy istnienie trzech produktów (po jednej sztuce każdy)

- $c_1 = 60$, $w_1 = 10$ (cena jednostkowa wynosi $60/10 = 6$),
- $c_2 = 100$, $w_2 = 20$ (cena jednostkowa wynosi $100/20 = 5$),
- $c_3 = 120$, $w_3 = 30$ (cena jednostkowa wynosi $120/30 = 4$).

wówczas algorytm zachłanny wybierze najpierw produkt pierwszy (jako ten o najwyższej cenie jednostkowej) a potem drugi (jako kolejny produkt o najwyższej cenie jednostkowej). Przy takim wyborze nośność plecaka została zmniejszona do 20, podczas gdy ostatni produkt waży 30 a więc już się nie zmieści. Wartość ładunku w plecaku wyniesie 160. Optymalny wybór w tym przypadku uzyskamy wybierając produkt drugi i trzeci co pozwoli wypełnić plecak maksymalnie a wartość przenoszonego ładunku wyniesie 220.

6.2 Charakterystyka metod zachłannych

Algorytm zachłanny wymaga określenia

1. Zbioru **operatorów** czyli czynności, akcji itp., których ciąg pozwala uzyskać rozwiązanie. W przykładzie z monetami operatorami mogą być:
   - op_1 – wydaj jako resztę monetę o wartości 10 zł;
   - op_2 – wydaj jako resztę monetę o wartości 5 zł;
   - itd.
2. Funkcji oceniającej koszt wyboru operatora i opłacalność całego rozwiązania (tzw. **funkcja oceny**).

Algorytmy zachłanne cechuje „krótkowzroczność" i „uparte" podążanie wybranym rozwiązaniem. Krótkowzroczność rozumiemy jako całkowite pominięcie informacji o stanach przyszłych do jakich prowadzi obecnie dokonywany wybór i posługiwanie się jedynie lokalną informacją w celu jak najszybszego sprowadzenia problemu do mniejszego podproblemu. Iteracyjnie dokonują one „zachłannych" wyborów jeden po drugim, redukując zadany problem do co raz mniejszego. Uparte podążanie wybranym rozwiązaniem to niezdolność do wznowienia poszukiwania rozwiązania w innym punkcie, gdy wybrane postępowanie do niego nie prowadzi. Mówiąc inaczej, algorytm zachłanny, nigdy się nie cofa celem „przemyślenia" podjętej uprzednio decyzji.

Najlepiej spełniają swoje zadanie w problemach cechujących się optymalną substrukturą (ang. *optimal substructure*).

Problem ma optymalną substrukturę jeśli optymalne rozwiązanie dla problemu zawiera rozwiązanie optymalne dla podproblemu.

Problem ma optymalną substrukturę jeśli najlepsze w danym momencie rozwiązanie jest częścią rozwiązania optymalnego.

Bardzo pożądaną cechą algorytmów tego typu jest prędkość działania, przewyższająca inne metody optymalizacyjne (np. programowanie dynamiczne). Jeśli więc tylko pozwalają one na uzyskanie choćby rozwiązania przybliżającego w zadowalającym stopniu rozwiązanie optymalne, często stają się naturalnym wyborem.

6.3 Hill climbing

Jednym z najprostszych algorytmów należących do grona algorytmów zachłanych jest algorytm przeszukiwania przestrzeni (*ang. hill climbing*). Skąd taka nazwa? Otóż strategia te wykorzystuje lokalną optymalizację. Na rozwiązywany problem patrzy z lokalnego

punktu widzenia, wybierając najlepsze w danym momencie rozwiązanie. Oczywiście najlepsze w danym momencie nie oznacza najlepsze w ogóle. Tak właśnie jest z turystą wchodzącym na górę (stąd angielska nazwa). Wybierając najlepszy w danym momencie szlak wcale nie musi przejść najoptymalniejszą trasą z punktu widzenia całej wędrówki. Do znajdywania najkrótszej ścieżki łączącej dwa wierzchołki metoda ta się nie nadaje. Owszem można osiągnąć poprawne rezultaty, ale są to raczej wyjątki niż reguła.

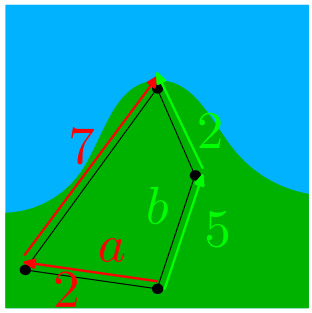
Chcąc zastosować algorytm *hill climbing* musimy wprowadzić **funkcję oceniającą**. Postać tej funkcji jest ściśle zależna od rozwiązywanego problemu jako, że informuje ona o „opłacalności" wybrania takiego a nie innego rozwiązania. Ponadto musimy określić **operator(y)** a więc **sposób generowania kolejnych stanów** (a więc stanów następujących po stanie obecnym). Stan można utożsamiać zarówno z punktem (wówczas generowanie kolejnych stanów oznacza wyznaczenie punktów sąsiednich) jak też np. z grą w szachy (wówczas generowanie kolejnych stanów oznacza wyznaczenie wszystkich możliwych do wykonania ruchów). Dla uproszczenia formułowania myśli, powiemy, że stan A wywodzi się ze stanu B jeśli stan A został wygenerowany ze stanu B. Ogólna postać tego algorytmu wygląda następująco:

1. Start. Wygeneruj stan początkowy SS: `SS:=Init()`.
2. Przyjmij jako bieżący stan CS wartość stanu SS: `CS:=SS`.
3. Czy można wygenerować nowy (jeszcze nie generowany ze stanu CS) stan pochodzący od stanu CS? Jeśli nie – zwróć `BRAK_ROZWIAZANIA` i zakończ algorytm. Jeśli tak – kontynuuj.
4. Wygeneruj stan pochodzący ze stanu CS i nazwij go NS: `NS:=Generate(CS)`.
5. Jeśli
   (a) NS jest stanem końcowym zwróć `SUKCES` i zakończ algorytm.
   (b) NS nie jest stanem końcowym, ale jest lepszy od CS, to `CS:=NS`.
6. Powrót do 3.

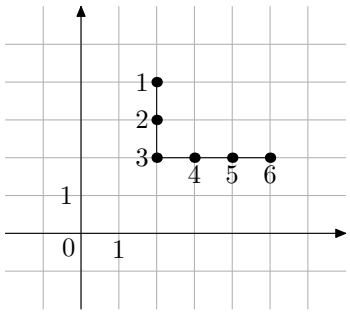
Punkt 4 można zastąpić następującą jego wersją

4. Wygeneruj wszystkie stany pochodzące ze stanu CS, wybierz z nich najlepszy i nazwij go NS: `NS:=Generate(CS)`.

Jak więc widać jest to algorytm lokalny, kierujący się przy wyborze kolejnego stanu tylko chwilowym polepszeniem wyników. Nie ma żadnej gwarancji, że prowadzi nas to do globalnie lepszego wyniku. Przypomina to troszkę turystę wchodzącego na górę i podejmującego decyzję odnośnie dalszej wędrówki na podstawie wyceny tylko kolejnego odcinka. Oczywiście taki sposób postępowania nie zawsze jest optymalny co pokazuje rusunek 6.1. Wybrana według takiego algorytmu doroga (czerwona) kosztuje 9 MU (mountain unit, taka wymyślona jednostka), podczas gdy droga odrzucona (zielona) kosztowałaby nas 7 MU. Odrzucona została tylko dlatego, że w momencie startu koszt odcinka b był większy od kosztu odcinka a. Co więcej, jak pokazał przykład z monetami, może zaistnieć sytuacja gdy nie zostanie znalezione żadne rozwiązanie, pomimo, że takowe istnieje. Przyjrzyjmy się teraz temu algorytmowi w konkretnej sytuacji. Niech dane będą punkty rozmieszczone tak jak pokazuje to rysunek 6.2. Jako funkcję oceny przyjmijmy odległość



Rysunek 6.1: Przykład nieoptymalnego działania algorytmu Hill Climbing. Wybrana według takiego algorytmu doroga (czerwona) kosztuje 9 MU (mountain unit), podczas gdy droga odrzucona (zielona) kosztowałaby 7 MU.



Rysunek 6.2: Przykładowy graf dla algorytmu Hill Climbing.

euklidesową do punktu docelowego. Punkt jest tym lepszy im wartość tej funkcji jest mniejsza. Funkcja generująca stany będzie zwracała kolejne punkty sąsiadujące z właśnie rozważanym. Tabela 6.1 zawiera kilka kroków tego algorytmu dla punktu startowego 2 oraz końcowego 5. W przykładzie korzystać będzimy z następujących funkcji

- `Init()` – funkcja generująca stan początkowy.
- `GenerateNext(x)` – funkcja generująca ze stanu x kolejny stan.
- `Value(x)` – funkcja oceniająca stan x.

Ćwiczenie 6.1.

Proszę napisać program realizujący przedstawiony powyżej algorytm. Zakładamy, że program wczytuje graf z pliku o podanej w linii poleceń nazwie. Następnie pyta o numery wierzchołków do sprawdzenia. Program powinien odczytywać niezbędne do działania dane z pliku o określonym poniże formacie. Maksymalna ilość wierzchołków wynosi 100 i numerujemy je kolejnymi liczbami całkowitymi od 1 do 100.

Krok algorytmu	Stan bieżący	Operacja	Wierzchołki odwiedzone					
			1	2	3	4	5	6
1		`SS:=Init()`, $(SS = 2)$						
2	2	`CS:=SS`, $(CS = 2)$		x				
3	2	Można wygenerować stany: 1,3		x				
4	2	`NS:=GenerateNext(CS)`, (NS=1)	x	x				
5(a)	2	$NS! = 5$	x	x				
5(b)	2	`n:=Value(NS)`, `c:=Value(CS)` $n = 2\sqrt{2} \approx 2.82$, $c = \sqrt{5} \approx 2.23$ $CS < NS$	x	x				
6	2	`Goto 4`	x	x				
3	2	Można wygenerować stany: 3	x	x				
4	2	`NS:=GenerateNext(CS)`, (NS=3)	x	x	x			
5(a)	2	$NS! = 5$	x	x	x			
5(b)	2	`n:=Value(NS)`, `c:=Value(CS)` $n = 2$, $c = \sqrt{5} \approx 2.23$ $CS > NS$, `CS:=NS`	x	x	x			
6	3	`Goto 4`	x	x	x			
3	3	Można wygenerować stany: 4	x	x	x			
4	3	`NS:=GenerateNext(CS)`, (NS=4)	x	x	x	x		
5(a)	3	$NS! = 5$	x	x	x	x		
5(b)	3	`n:=Value(NS)`, `c:=Value(CS)` $n = 1$, $c = 2$ $CS > NS$, `CS:=NS`	x	x	x	x		
6	4	`Goto 4`	x	x	x	x		
3	4	Można wygenerować stany: 5	x	x	x	x		
4	4	`NS:=GenerateNext(CS)`, (NS=5)	x	x	x	x	x	
5(a)	4	$NS = 5$	x	x	x	x	x	
	4	`SUKCES`	x	x	x	x	x	

Tabela 6.1: Przebieg wykonania algorytmu Hill Climbing dla punktów z rysunku 6.2 oraz wierzchołków 2 i 5.

Numer linii	*Opis*
1	*Ilość wierzchołków w grafie (n)*
2	*Współrzędne wierzchołka 1 na płaszczyźnie.*
...	...
$n+1$	*Współrzędne wierzchołka n na płaszczyźnie.*
$n+2$	*Listowy opis grafu dla wierzcholka 1.*
...	...
$2n+1$	*Listowy opis grafu dla wierzcholka n.*

Zarówno współrzędne jak i elementy listy rozdzielone są znakiem spacji. Programy powinien w każdej iteracji wyświetlać listę aktualnie odwiedzonych wierzchołków a na zakończenie wyświetlać informację, czy wierzchołki są połączone czy też nie. Oto przykładowy wygląd pliku dla przykładu rozpatrywanego powyżej:

```
6
2 4
2 3
2 2
3 2
4 2
5 2
2
1 3
2 4
3 5
4 6
5
```

Ćwiczenie 6.2.

Proszę napisać program sprawdzający, w oparciu o podany algorytm, czy możliwe jest przejście w labiryncie od jednego miejsca do drugiego.

Rozdział 7

Poszukiwanie optymalnej ścieżki

7.1 Best First Search

Jak to zostało powiedziane wcześniej, zadanie przeszukiwania grafu można uogólnić, na zadania przeszukiwania pewnej przestrzeni w celu znalezienia ścieżki łączącej dwa punkty. W tym podrozdziale przedstawimy kolejne algorytmy, które można wykorzystać także do przeszukiwania grafów. Wydzieliliśmy je do osobngo podrozdziału, ponieważ różnią się one od poprzednich w pewnych założeniach.

Po pierwsze zakładają istnienie pewnej metryki i miary pozwalającej mierzyć odległość dwóch wierzchołków (czasem nawet takich, które bezpośredniu lub nawet wcale nie są połączone). Po drugie, zakłada się, że każde połączenie dwóch wierzchołków ma jakąś wagę (koszt przejścia tym połączeniem od jednego wierzchołka do drugiego).

Tak więc algorytmy te będziemy mogli wykorzystać dla grafu, jeśli:

- Umieścimy graf w jakiejś przestrzeni, np. euklidesowej, przypisując każdemu wierzchołkowi jakieś współrzędne.
- Ustalimy koszty przejścia po krawędziach (może być on np. stały i wynosić zawsze 1).

Miejmy jednak na uwadze, że sposób rozmieszczenia grafu w przestrzeni będzie miał istotny wpływ na to, czy uda się znaleźć rozwiązanie, czy nie.

Na potrzeby tego i kolejnych rozdziałów jako przestrzeń naszych rozważań przyjmujemy przestrzeń $\mathbf{R}^2$ wraz z określonymi w niej punkatmi (wierzchołkami grafu). Pomiędzy punktami istnieje bezpośrednie połączenie jeśli połączone są odpowiadające im wierzchołki grafu. Koszt przejścia równy jest długości krawędzi. Jeśli nie będzie powiedziane inaczej, krawędź jest odcinkiem. Tak więc krawędz równa jest odległości euklidesowej punktów.

7.1.1 Opis algorytmu

Do tej pory omawiane były dwie metody systematycznego przeszukiwania rozwiązań (traktowanych jako wierzchołki grafu): przeszukiwanie w głąb i przeszukiwanie wszerz[1]. Połączmy teraz ich zalety:

[1] Przeszukiwań typu Hill Climbing nie klasyfikujemy jako metody systematycznego przeszukiwania, gdyż mogą one zatrzymać się po odwiedzeniu nawet niewielkiej ilości stanów z rozważanej przestrzeni.

- przeszukiwanie w głąb pozwala na znalezienie rozwiązania bez potrzeby przeglądania wszystkich „konkurencyjnych" rozwiązań;
- przeszukiwanie wszerz chroni przed zabrnięciem w „ślepą uliczkę".

Jeden z możliwych sposobów to podążanie jedną ścieżką (jak w głąb), ale „przełączanie się" na inną jeśli tylko okaże się ona lepsza od bieżącej. Ponieważ powyżej wystąpiło pojęcie „lepsza" zatem musimy określić sposób porównania czy też pewną miarę. Wprowadzamy w tym celu dwie funkcje oceniające:

- $f1$, podającej koszt przejścia od stanu początkowego do stanu bieżącego; wartość ta jest dokładna, gdyż dotyczy ona stanów, w których już byliśmy i dokładnie znamy koszty związane z przejściami między nimi;
- $f2$, podającej koszt przejścia od stanu bieżącego do końcowego; wartość ta jest jedynie przybliżona.

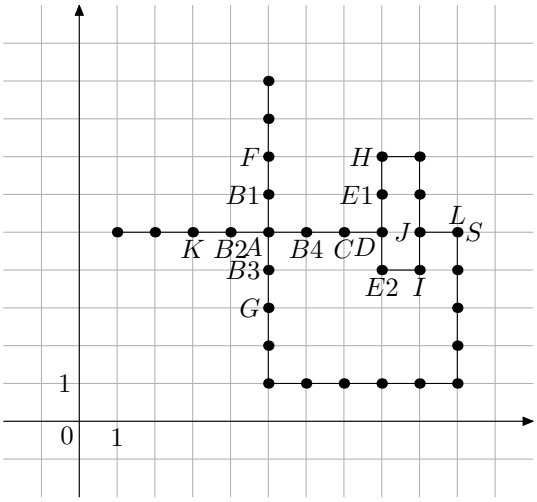
Ogólna postać tego algorytmu wygląda następująco:

1. Start. Utwórz kolejkę priorytetową PQ.
2. Wygeneruj stan początkowy SS: `SS:=Init()`.
3. Przyjmij jako bieżący stan CS wartość stanu SS: `CS:=SS`.
4. Wygeneruj ze stanu CS wszystkie możliwe stany po nim następujące. Każdemu z nich przypisz liczbę V zależną od wartości zwróconych przez funkcje $f1$ i $f2$ (np. ich sumę). Liczba ta określa priorytet stanu w kolejce. Dodaj do kolejki wszystkie te stany, które jeszcze się w niej nie znajdują lub nie znajdowały sie[2].
5. Pobież stan z PQ i nazwij go CS. Jeśli PQ jest pusta, wówczas przyjmij jako CS wartość $NULL$.
6. Jeśli
   (a) CS jest stanem końcowym zwróć `SUKCES` i zakończ algorytm.
   (b) $CS = NULL$ zwróć `BRAK_ROZWIAZANIA` i zakończ algorytm.
7. Powrót do 4.

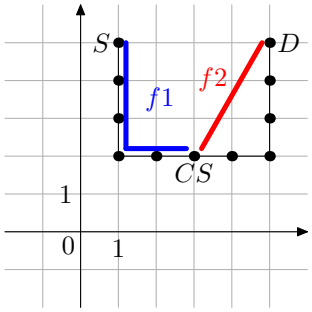
Zauważmy, że algorytm ten może nie dawać optymalnego rozwiązania w sytuacji, gdy jest kilka sposobów przejścia pomiędzy punktami o różnych wartościach funkcji oceny. Spowodowane jest to tym, że do kolejki dodajemy tylko takie stany, które się w kolejce nie zanjdują bądź też nie znajdowały. Nasuwająca się tutaj prosta modyfikacja, polegająca na dodawaniu również stanów, które już kiedyś były lub są w kolejce, prowadzi oczywiście do zapętlenia algorytmu. O tym jak poprawić ten algorytm powiemy w dalszej części.

Przyjrzyjmy się teraz temu algorytmowi w konkretnej sytuacji. Niech dane będą punkty rozmieszczone tak jak pokazuje to rysunek 7.1. Jako funkcję oceny przyjmijmy (patrz rys. 7.2):

[2]Czyli każdy stan trafia do kolejki tylko raz, bez względu na ilość możliwych sposobów jego wygenerowania.



Rysunek 7.1: Przykładowy graf dla algorytmu Best First Search.



Rysunek 7.2: Ilustracja sposobu wyznaczania wartości dla funkcji $f1$ oraz $f2$.

- $f1$ – wartością funkcji jest doległość od punktu startowego S do bieżącego węzła CS, liczona jako, suma odległości pomiędzy punktami, przez które przechodzi ścieżka;
- $f2$ – odległość euklidesowa pomiędzy bieżącym punktem CS a punktem docelowym D.

Liczba V przypisana każdemu stanowi jest sumą wartości zwróconych przez $f1$ i $f2$. Przyjmujemy, że stany o mniejszej wartości V mają większy priorytet. Przez A oznaczmy punkt $(5, 5)$ traktowany jako stan początkowy, zaś przez S punkt $(10, 5)$ traktowany jako stan końcowy. Dodatkowo jeśli na bieżąco będziemy tworzyć drzewo, w którym każdy węzeł reprezentować będzie stan, w którym się znaleźliśmy to dzięki temu łatwo będziemy mogli wypisać scieżkę prowadzącą od stanu początkowego do końcowego. Zauważmy, że w drzewie tym istotne jest, aby każdy następca wskazywał też na swojego poprzednika. Stan drzewa przy wykonywaniu kolejnych kroków algorytmu przedstawiają rysunki 7.3–7.14 (kolorem czerwonym wyróżniono najlepszą ścieżkę).

Przebieg algorytmu.

1. Wygenerowanie wszystkich stanów dostępnych ze stanu A, tj. stanów: $B1$, $B2$,

$B3$, $B4$. Koszt przejścia do nich jest taki sam i wynosi 1. Odległość od stanu S to odpowiednio: $\sqrt{26}$, 6, $\sqrt{26}$ i 4. Zatem koszt przejscia do poszczególnych stanów jest następujący:

- $B4$: $1 + 4 = 5$;
- $B1$: $1 + \sqrt{26} \approx 6.09$;
- $B3$: $1 + \sqrt{26} \approx 6.09$;
- $B2$: $1 + 6 = 7$.

W tej sytuacji najlepiej będzie gdy przejdziemy do stanu $B4$.

2. Wygenerowanie wszystkich stanów dostępnych ze stanu $B4$, tj. stanu C i dołaczenie ich do listy:

   - C: $2 + 3 = 5$;
   - $B1$: $1 + \sqrt{26} \approx 6.09$;
   - $B3$: $1 + \sqrt{26} \approx 6.09$;
   - $B2$: $1 + 6 = 7$.

   Przejście do stanu C.

3. Wygenerowanie wszystkich stanów dostępnych ze stanu C, tj. stanu D i dołączenie ich do listy:

   - D: $3 + 2 = 5$;
   - $B1$: $1 + \sqrt{26} \approx 6.09$;
   - $B3$: $1 + \sqrt{26} \approx 6.09$;
   - $B2$: $1 + 6 = 7$.

   Przejście do stanu D.

4. Ze stanu D możemy przejść do stanu $E1$ lub $E2$. Stąd w liście stanów mieć będziemy:

   - $B1$: $1 + \sqrt{26} \approx 6.09$;
   - $B3$: $1 + \sqrt{26} \approx 6.09$;
   - $E1$: $4 + \sqrt{5} \approx 6.23$;
   - $E2$: $4 + \sqrt{5} \approx 6.23$;
   - $B2$: $1 + 6 = 7$.

   Przejście do stanu $B1$.

5. Wygenerowanie wszystkich stanów dostępnych ze stanu $B1$, tj. stanu F i dołączenie ich do listy:

   - $B3$: $1 + \sqrt{26} \approx 6.09$;
   - $E1$: $4 + \sqrt{5} \approx 6.23$;
   - $E2$: $4 + \sqrt{5} \approx 6.23$;

- $B2$: $1 + 6 = 7$;
- F: $2 + \sqrt{29} \approx 7.38$.

Przejście do stanu $B3$.

6. Wygenerowanie wszystkich stanów dostępnych ze stanu $B3$, tj stanu G i dołączenie ich do listy:

   - $E1$: $4 + \sqrt{5} \approx 6.23$;
   - $E2$: $4 + \sqrt{5} \approx 6.23$;
   - $B2$: $1 + 6 = 7$;
   - F: $2 + \sqrt{29} \approx 7.38$;
   - G: $2 + \sqrt{29} \approx 7.38$.

   Przejście do stanu $E1$.

7. Wygenerowanie wszystkich stanów dostępnych ze stanu $E1$, tj stanu H i dołączenie ich do listy:

   - $E2$: $4 + \sqrt{5} \approx 6.23$;
   - $B2$: $1 + 6 = 7$;
   - F: $2 + \sqrt{29} \approx 7.38$;
   - G: $2 + \sqrt{29} \approx 7.38$;
   - H: $5 + \sqrt{8} \approx 7.82$.

   Przejście do stanu $E2$.

8. Wygenerowanie wszystkich stanów dostępnych ze stanu $E2$, tj stanu I i dołączenie ich do listy:

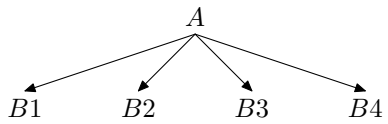
   - I: $5 + \sqrt{2} \approx 6.41$;
   - $B2$: $1 + 6 = 7$;
   - F: $2 + \sqrt{29} \approx 7.38$;
   - G: $2 + \sqrt{29} \approx 7.38$;
   - H: $5 + \sqrt{8} \approx 7.82$.

   Przejście do stanu I.

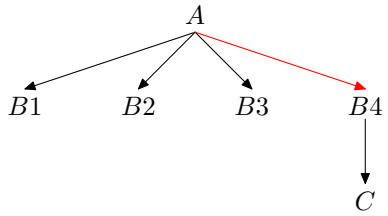
9. Wygenerowanie wszystkich stanów dostępnych ze stanu I, tj. stanu J i dołączenie ich do listy:

   - $B2$: $1 + 6 = 7$;
   - J: $6 + 1 = 7$;
   - F: $2 + \sqrt{29} \approx 7.38$;
   - G: $2 + \sqrt{29} \approx 7.38$;
   - H: $5 + \sqrt{8} \approx 7.82$.

   Przejście do stanu $B2$.



Rysunek 7.3: Drzewo odwiedzanych węzłów po 1 kroku algorytmu.



Rysunek 7.4: Drzewo odwiedzanych węzłów po 2 kroku algorytmu.

10. Wygenerowanie wszystkich stanów dostępnych ze stanu $B2$, tj. stanu K i dołączenie ich do listy:
    - J: $6 + 1 = 7$;
    - F: $2 + \sqrt{29} \approx 7.38$;
    - G: $2 + \sqrt{29} \approx 7.38$;
    - H: $5 + \sqrt{8} \approx 7.82$;
    - K: $2 + 7 = 9$.

    Przejście do stanu J.

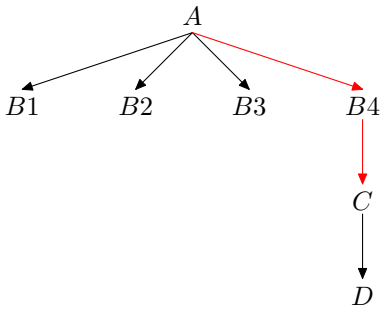
11. Wygenerowanie wszystkich stanów dostępnych ze stanu J, tj. stanu $L1$ oraz $L2$ i dołączenie ich do listy:
    - $L1$: $7 + 0 = 7$;
    - F: $2 + \sqrt{29} \approx 7.38$;
    - G: $2 + \sqrt{29} \approx 7.38$;
    - H: $5 + \sqrt{8} \approx 7.82$;
    - $L2$: $7 + \sqrt{2} \approx 8.41$;
    - K: $2 + 7 = 9$.

    Przejście do stanu $L1$.

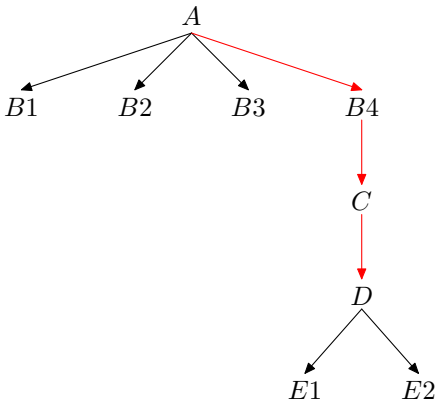
12. Stan $L1$ jest stanem końcowym – koniec algorytmu.

Ćwiczenie 7.1.

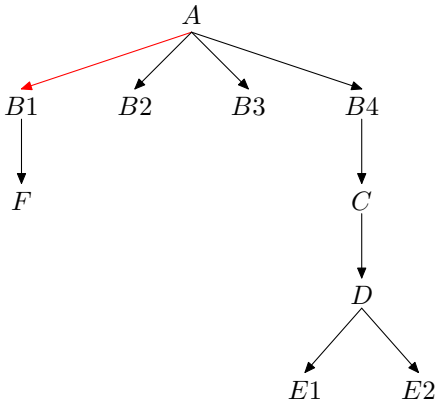
Proszę napisać program realizujący przedstawiony powyżej algorytm. Zakładamy, że program wczytuje graf z pliku o podanej w linii poleceń nazwie. Następnie pyta o numery wierzchołków do sprawdzenia. Program powinien odczytywać niezbędne do działania dane z pliku o określonym dla poprzednich ćwiczeń formacie. Programy powinien w każdej iteracji wyświetlać listę aktualnie odwiedzonych wierzchołków wraz z wartością V funkcji



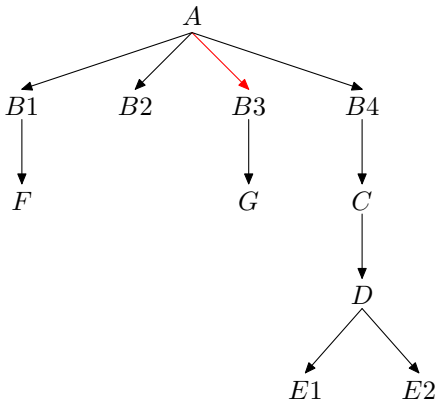
Rysunek 7.5: Drzewo odwiedzanych węzłów po 3 kroku algorytmu.



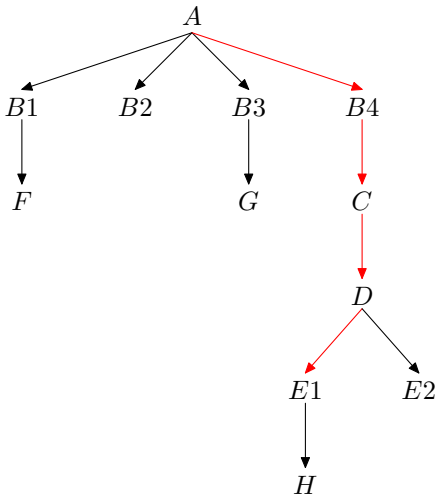
Rysunek 7.6: Drzewo odwiedzanych węzłów po 4 kroku algorytmu.



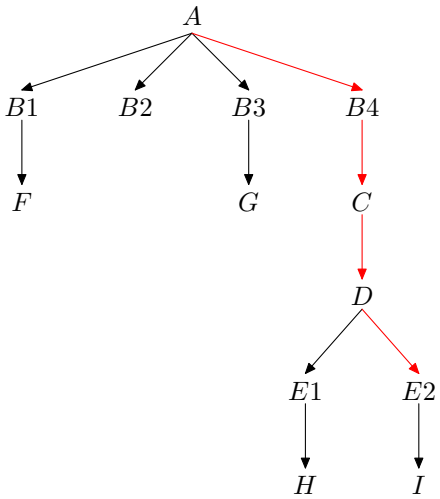
Rysunek 7.7: Drzewo odwiedzanych węzłów po 5 kroku algorytmu.



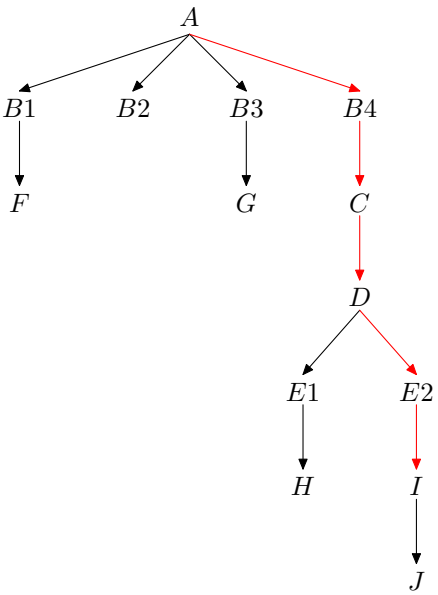
Rysunek 7.8: Drzewo odwiedzanych węzłów po 6 kroku algorytmu.



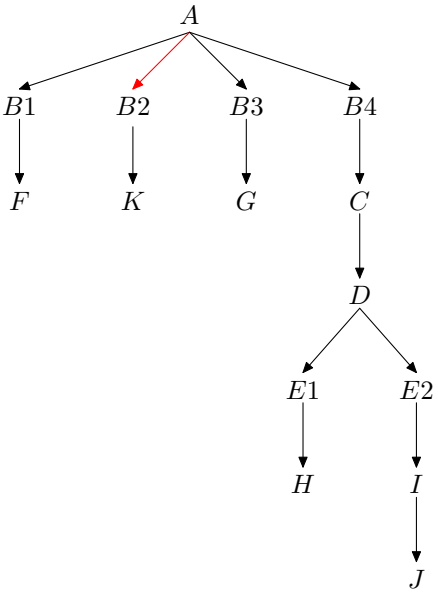
Rysunek 7.9: Drzewo odwiedzanych węzłów po 7 kroku algorytmu.



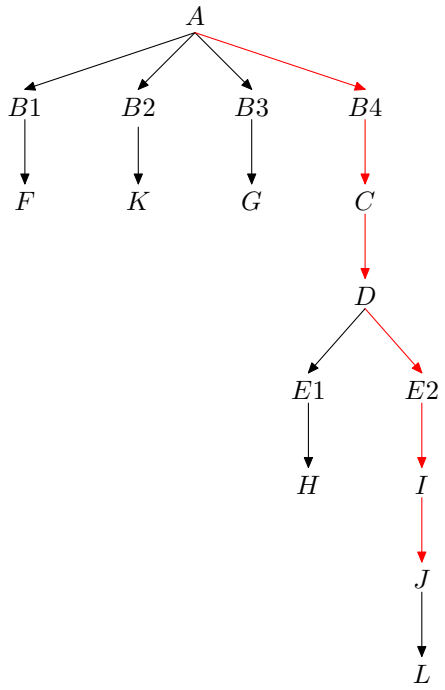
Rysunek 7.10: Drzewo odwiedzanych węzłów po 8 kroku algorytmu.



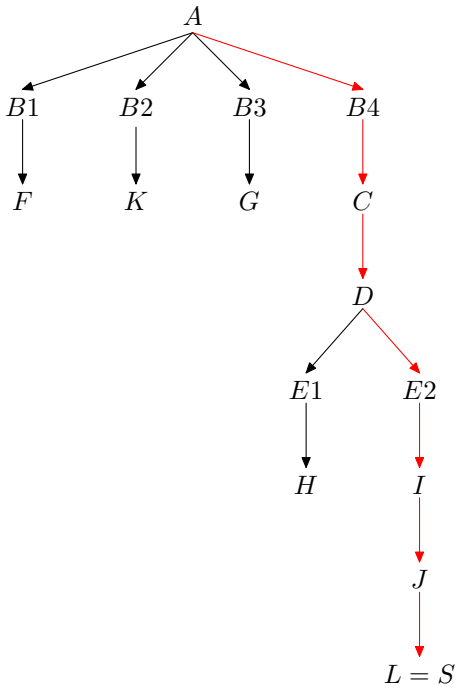
Rysunek 7.11: Drzewo odwiedzanych węzłów po 9 kroku algorytmu.



Rysunek 7.12: Drzewo odwiedzanych węzłów po 10 kroku algorytmu.



Rysunek 7.13: Drzewo odwiedzanych węzłów po 11 kroku algorytmu.



Rysunek 7.14: Drzewo odwiedzanych węzłów po 12 kroku algorytmu.

oceny, numer wierzchołka bieżącego a na zakończenie wyświetlać informację, czy wierzchołki są połączone czy też nie. Dodatkowo program powinien podawać ścieżkę łączącą zadane punkty.

Ćwiczenie 7.2.

Proszę napisać program sprawdzający, w oparciu o podany algorytm, czy możliwe jest przejście w labiryncie od jednego miejsca do drugiego.

Ćwiczenie 7.3.

Napisać program znajdujący na planszy np. takiej jak na rysunku //uzup// najtańszą drogę pomiędzy dowolnie wskazanymi punktami. Każde pole można utożsamiać z wierzchołkiem grafu. Sąsiednie wierzchołki są połączone, jeśli tylko, istnieje przejście z jednego pola na drugie. Koszt przejścia zależny jest od rodzaju terenu i kształtować się może następująco:

7.2 Algorytmy typu brute force

Algorytm brute force (ang. „brutalna siła" lub „brutalny ata') nie mają sobie nic z finezji i „wyrachowania" poprzedniej metody. określenie algorytmu, który opiera się na sukcesywnym sprawdzeniu wszystkich możliwych kombinacji w poszukiwaniu rozwiązania problemu, zamiast skupiać się na jego szczegółowej analizie.

Rozdział 8

Means-Ends Analysis

Wszystkie zaprezentowane do tej pory strategie rozpoczynały działanie ze stanu początkowego i podążały w kierunku stanu końcowego. Zatem ścieżka prowadząca do rozwiązania wyznaczana była od początku do końca. Pokażemy teraz inny sposób podejścia do zagadnienia, wzorowany na tym jak pewne problemy rozwiązują ludzie.

Załóżmy, że obiekt `CZŁOWIEK` jest w stanie **głodny** i chciałby znaleźć się w stanie **najedzony**. Wychodząc od stanu **głodny** można podać następującą sekwencję *operatorów* (czyli w tym przypadku czynności) przeprowadzających nas do stanu **najedzony** (stany **pogrubiono**, operatory *pochylono*, normalną czcionką zapisane są polecenia sterujące przebiegiem naszego dalszego postępowania):

1. **Głodny (Start)**
2. *Idź do kuchni*
3. *Otwórz lodówkę*
4. *Weź coś z półki*
5. Jeśli ostatni operacja się udała kontynuuj, w przeciwnym razie idź do 10.
6. *Zobacz czy świeże*
7. Jeśli wynik ostatniej operacji był pozytywny – kontynuuj, w przeciwnym razie idź do 4.
8. *Zjedz*
9. **Najedzony (Stop)**
10. *Ubierz się*
11. *Idź do sklepu*
12. Jeśli masz przy sobie pieniądze - kontynuuj, jeśli nie idź do 16.
13. *Kup*
14. *Wróć do domu*
15. Idź do 8.

16. *Wróć do domu*
17. *Weź pieniądze*
18. Idź do 11.

Mimo iż powyższa sekwencja jest poprawna to jednak powinniśmy dokładniej przyjrzeć się temu jak powstała. Odzwierciedla ona faktyczną kolejność operacji jakie należy wykonać, ale nie odzwierciedla naturalnego sposobu ustawiania reguł w taką kolejność aby osiągnąć cel. Inaczej mówiąc, taka kolejność nie jest naturalną kolejnością rozwiązywania problemu. Bo niby skąd obiekt `CZŁOWIEK` ma wiedzieć, że będąc w stanie **głodny** i wybierając jako pierwszy operator *idź do kuchni* uda się jemu po iluś krokach dojść do stanu **najedzony**? Przecież gdy będziemy już w kuchi to mamy możliwość wykonania wielu różnych czynności: zmywanie naczyń, gotowanie, obieranie ziemniaków... Tylko dzięki naszemu doświadczeniu spodziewamy się, że takie działanie, przy odpowiednim dobraniu działań **kolejnych**, przyniesie oczekiwany efekt. Jednak maszyny nie posiadają (jeszcze) intuicji i nie mogą wesprzeć się doświadczeniem. Przeanalizujmy zatem jeszcze raz ten sam problem, postępując tak jak gdybyśmy i my nie mieli żadnego doświadczenia związanego z postawionym zadaniem.

Zatem jesteśmy **głodni** a chcemy być **najedzeni**. Logiczna czynność jaką należy w takim przypadku wykonać, to coś *zjeść* (właśnie *zjeść* a nie *iść do kuchni*!!!!).

```
Głodny->...->jedz->Najedzony
```

W naszej sytuacji to właśnie wybranie operacji *jeść* jest najbardziej narzucające się, gdyż to ona gwarantuje najszybsze dotarcie do stanu docelowego. Teraz dopiero możemy zastanowić się co zrobić aby coś zjeść. Można iść do kuchni:

```
Głodny->idź do kuchni->...->jedz->Najedzony
```

Będąc w kuchni zajrzymy do lodówki:

```
Głodny->idź do kuchni->zajrzyj do lodówki->...->jedz->Najedzony
```

Jeśli coś znajdziemy w lodówce, spróbujemy to zjeść; jeśli nie - idziemy do sklepu

```
                ------------<----------------------------(N)
Głodny          |                                        |
    |           |                                        |
idź do kuchni   |                                        |
    |           |                                        |
zajrzyj do lodówki->coś jest->(T)->bierz z półki->czy świeże->(T)->jedz->Najedzony
                    |                                                    |
                    |                                                    |
                  (N)->idź do sklepu------>...-----------------------><-
```

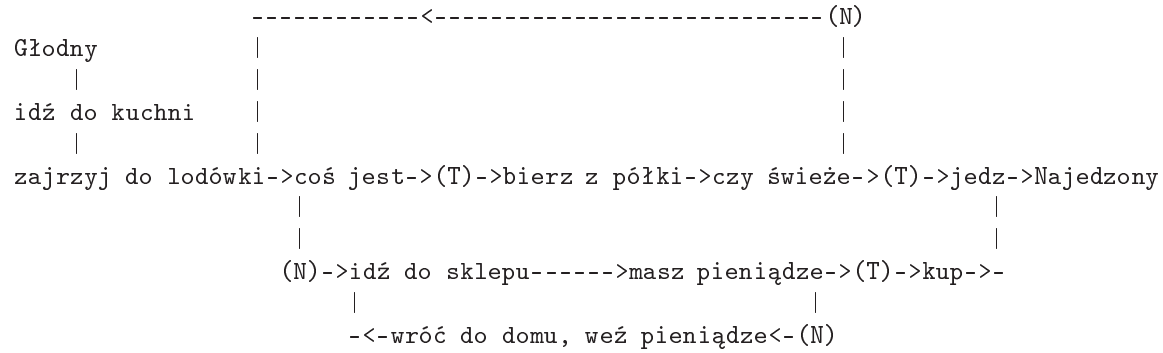
Będąc w sklepie staramy się coś kupić

```
                ------------<----------------------------(N)
Głodny          |                                        |
    |           |                                        |
idź do kuchni   |                                        |
    |           |                                        |
zajrzyj do lodówki->coś jest->(T)->bierz z półki->czy świeże->(T)->jedz->Najedzony
                    |                                                    |
                    |                                                    |
                  (N)->idź do sklepu------>...----------------->kup->-
```

Operator	Warunki wstępne	Warunki spełnione
jedz	**głodny**, jest co jeść	**najedzony**
idź do kuchni	brak	jestem w kuchni
zajrzyj do lodówki	jestem w kuchni	coś jest w lodówce lub lodówka pusta
bierz z półki	coś jest w lodówce	produkt w ręce
zobacz czy świeże	produkt w ręce	jest co jeść lub jestem w kuchni
idź do sklepu	lodówka pusta	jestem w sklepie
kup	jestem w sklepie, mam pieniądze przy sobie	coś jest w lodówce
wróć do domu i zabierz pieniądze	brak	mam pieniądze

Tabela 8.1: Lista przykładowych operatorów wraz z warunkami jakie powinny zostać spełnione i jakie zostaną spełnione.

Na zakupy potrzeba jednka pieniędzy



Ostatecznie otrzymujemy podobny (jeśli nie identyczny) schemat jak za pierwszym razem. Różnica polega jednak na tym, że w drugim przypadku analizę problemu rozpoczęliśmy niejako „od środka" wybierając najistotniejsze zagadnienie. Chcąc się bowiem najeść najistotniejsze jest zjedzenie czegoś a nie przjście do kuchni. Gdy już wiemy, że musimy coś zjeść, zastanawiamy się co zrobić aby można było zastosować operator *jeść*. W ten sposób problem rozbijamy na podproblemy. Każdy z podproblemów traktujemy następnie jak problem główny rozbijając je na jeszcze mniejsze zagadnienia. Można powiedzieć, że ta technika analizy skupia swoją uwagę na **znalezieniu i redukowaniu różnic pomiędzy dwoma stanami**. Przejścia od jednego problemu do innego umożliwiają operatory. Z każdym operatorem związane są dwie listy.

- Lista warunków wstępnych – lista ta opisuje wszystkie warunki jakie MUSZĄ być spełnione aby móc zastosować dany operator.
- Lista warunków jakie będą spełnione po zastosowaniu tego operatora.

Dla naszego przykładu lista ta mogłaby przyjąć postać z tabeli 8.1.

Poniżej przedstawiamy ogólną postać algorytmu Means-Ends Analysis. Algorytm ten rozwiązuje postawione zadanie wyszukując i redukując różnice pomiędzy dwoma

zadanymi stanami. Problem wyjściowy rozbijany jest na podproblemy mniejsze tak długo aż dojdziemy do problemu rozwiązywalnego. W oparciu o rozwiązane problemy, próbujemy następnie rozwiązać pozostałe problemy. Taki sposób działania typowy jest dla algorytmów typu *dziel i zwyciężaj* i naturalnym narzędziem w tym przypadku wydaje się rekurencja. Ze względu na ogólność zapisu, od problemu zależy co rozumieć będziemy pod pojęciem „wybierz najbardziej istotną różnicę".

Wywołanie algorytmu przyjmuje postać `MEA(START,STOP)`. Jako dane wejściowe podajemy stan początkowy i końcowy. Wyjściem z algorytmu jest ciąg operatorów (w odpowiedniej kolejności), których zastosowanie pozwoli na przejscie od zadanego stanu początkowego do zadanego stanu końcowego. Nie jest to jednak równoznaczne z wyznaczeniem ścieżki od węzła początkowego do końcowego.

1. Jeśli `START=STOP` zwróc strukturę

   ```
   {
      OPERATORS:=NULL
      RESULT:=SUCCESS
   }
   ```

2. W przeciwnym razie **wybierz najbardziej istotną różnicę pomiędzy stanami** `START` i `STOP` i usuń je powtarzając następujące kroki dopóki nie będzie zwrócona wartość `SUCCESS` lub `FAILURE`:

   (a) Wybierz nieużywany jeszcza dla stanów `START` i `STOP` a możliwy do zastosowania operator `O`, który pozwoli zlikwidować różnicę wskazaną w punkcie 2. Jeśli takiego operatora nie ma, zwróć strukturę

   ```
   {
      OPERATORS:=NULL
      RESULT:=FAILURE
   }
   ```

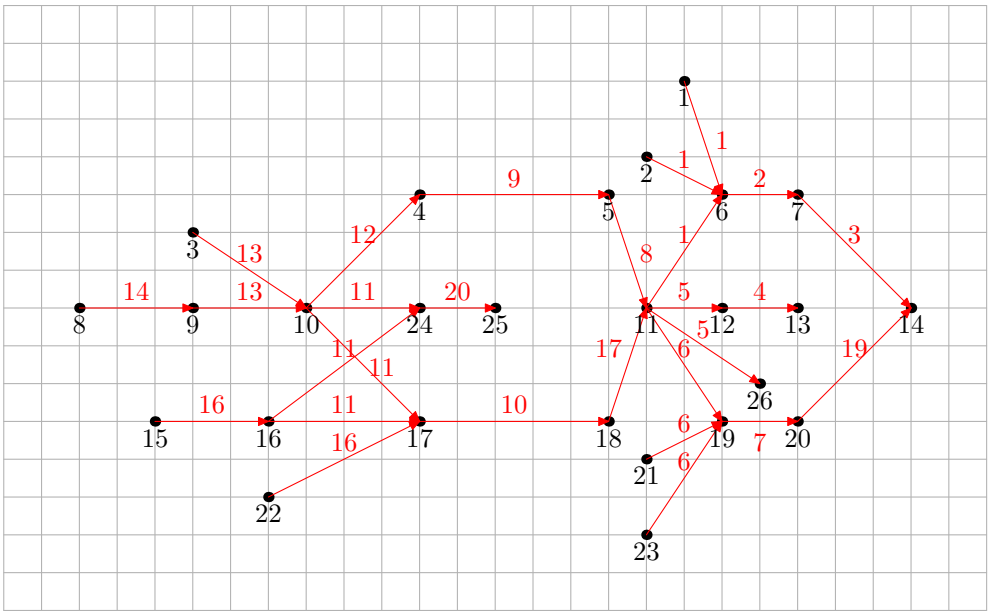
   (b) Spróbuj zastosować `O`. W tym celu sporządź listę warunków wstępnych `O-START`, których spełnienie pozwoli dopiero na zastosowanie operatora, oraz listę warunków końcowych `O-RESULT` jekie zostaną spełnione po zastosowaniu reguły.

   (c) `FIRST-PART:=MEA(START,O-START)`. Aby to wywołanie mogło zwrócić `SUCCESS` wszystkie wywołania `MEA(START,o-start)`, gdzie `o-start` jest elementem z `O-START`, muszą zwrócić `SUCCESS`. Jeśli dla pewnego `o-start` wywołanie zwróci `FAILURE` to wykonaj `FIRST-PART:=MEA(GOOD,o-start)`. `GOOD` reprezentuje wszystkie stany, które są spełnione z założenia. Wystarczy, że `MEA()` zwróci `SUCCESS` dla jednego elementu z `GOOD`. `FIRST-PART.OPERATORS` jest sumą wszystkich wartości zwróconych przez wszystkie podrzędne wywołania[1].

   (d) `LAST-PART:=MEA(O-RESULT,STOP)`. Wystarczy, że `MEA()` zwróci `SUCCESS` dla jednego elementu z `O-RESULT`.

   (e) Jeśli `FIRST-PART.RESULT=SUCCESS` i `LAST-PART.RESULT=SUCCES` to zwróć strukturę

[1] Wartości `NULL` można pominąć.



Rysunek 8.1: Przykładowy graf dla algorytmu Means-Ends Analysis. Liczby koloru czarnego oznaczają wierzchołki. Numery operatorów są koloru czerwonego.

```
{
   OPERATORS:=FIRST-PART.OPERATORS,O,LAST-PART.OPERATORS
   RESULT:=SUCCESS
}
```

W przeciwnym przypadku zwróć

```
{
   OPERATORS:=NULL
   RESULT:=FAILURE
}
```

Przyjrzyjmy się działaniu tego algorytmu w następującym przykładzie. Dostępne operatory wraz z warunkami wstępnymi i końcowymi zawiera tabela 8.2. Warunki 1, 2, 3, 8, 15, 21, 22, 23 uznajemy za już spełnione. Ponadto jako spełniony przyjmujemy stan startowy (wydaje się to logiczne, bo skoro zaczynamy z pewnego stanu to oznacza to, że musiał zajść taki ciąg operacji, który spowodował, że w tym stanie się znaleźliśmy). Operatory można wykorzystać do przechodzenia pomiędzy węzłami grafu skierowanego z rysunku 8.1. Możliwości te zebrano w tabeli 8.3, w której wiersz oznacza węzeł początkowy, kolumna końcowy, a wartości wpisane w komórki to operatory jakie możemy do danego przejścia zastosować (oczywiście po spełnieniu wymaganych warunków). Przyjmujemy umowę, że operatory te pozwalają nam zlikwidować „najbardziej istotna różnicę". Obecnie dla każdego przejścia jest jeden operator, ale na podstawie rysunku grafu można tą tabelę uzupełnić o dowolne inne operatory leżące na ścieżce łączącej dwa węzły.

Poniżej przedstawiono rezultat wywołania opisywanego algorymtu przy tak przyjętych założeniach dla węzła początkowego równego 16 i końcowego równego 14.

```
MEA(16,14) o6 (11,21,23 => 19)
FIRST:
   MEA(16,11) o10 (17 => 18)
   FIRST:
      MEA(16,17) o11 (10,16 => 17,24)
      FIRST:
         MEA(16,10) => FAILURE
         === dodatkowe sprawdzenie ===
            MEA(1,10) => FAILURE
            MEA(2,10) => FAILURE
            MEA(3,10) o13 (3,9 => 10)
            FIRST:
               MEA(3,3) => SUCCESS
               MEA(3,9) => FAILURE
               === dodatkowe sprawdzenie ===
                  MEA(1,9) => FAILURE
                  MEA(2,9) => FAILUR
                  MEA(3,9) => FAILURE
                  MEA(8,9) o14 (8 => 9)
                  FIRST:
                     MEA(8,8) => SUCCESS
                  FIRST-PART.OPERATORS:=NULL
                  LAST:
                     MEA(9,9) => SUCCESS
                  LAST-PART.OPERATORS:=NULL
                  OPERATORS:=NULL,o14,NULL
               === ostatecznie MEA(3,9) => SUCCESS
            FIRST-PART.OPERATORS:=o14
            LAST:
               MEA(10,10) => SUCCESS
            LAST-PART.OPERATORS:=NULL
            OPERATORS:=o14,o13,NULL
         === ostatecznie MEA(16,10) => SUCCESS
         MEA(16,16) => SUCCESS
      FIRST-PART.OPERATORS:=o14,o13
      LAST:
         MEA(17,17) => SUCCESS
         === break
      LAST-PART.OPERATORS:=NULL
      OPERATORS:=o14,o13,o11,NULL
   FIRST-PART.OPERATORS:=o14,o13,o11
   LAST:
      MEA(18,11) o17 (18 => 11)
      FIRST:
         MEA(18,18) => SUCCESS
      FIRST-PART.OPERATOR:=NULL
      LAST:
         MEA(11,11) => SUCCESS
      LAST-PART.OPERATORS:= NULL
      OPERATORS:=NULL,o17,NULL
   LAST-PART.OPERATORS:=o17
   OPERATORS:=o14,o13,o11,o10,o17
   MEA(16,21) => FAILURE
   === dodatkowe sprawdzenie ===
```

```
      MEA(1,21) => FAILURE
      MEA(2,21) => FAILURE
      MEA(3,21) => FAILURE
      MEA(8,21) => FAILURE
      MEA(15,21) => FAILURE
      MEA(21,21) => SUCCESS
   === ostatecznie MEA(16,21) => SUCCESS
   OPERATORS:=NULL
   MEA(16,23) => FAILURE
   === dodatkowe sprawdzenie ===
      MEA(1,23) => FAILURE
      MEA(2,23) => FAILURE
      MEA(3,23) => FAILURE
      MEA(8,23) => FAILURE
      MEA(15,23) => FAILURE
      MEA(21,23) => FAILURE
      MEA(22,23) => FAILURE
      MEA(23,23) => SUCCESS
   === ostatecznie MEA(16,23) => SUCCESS
   OPERATORS:=NULL
FIRST-PART.OPERATORS:=o14,o13,o11,o10,o17
LAST:
   MEA(19,14) o19 (20 => 14)
   FIRST:
      MEA(19,20) o7 (19 => 20)
      FIRST:
         MEA(19,19) => SUCCESS
      FIRST-PART.OPERATOR:=NULL
      LAST:
         MEA(20,20) => SUCCESS
      LAST-PART.OPERATOR:=NULL
   FIRST-PART.OPERATOR:=NULL,o7,NULL
   LAST:
      MEA(14,14) => SUCCESS
   LAST-PART.OPERATORS:= NULL
   OPERATORS:=o7,o19,NULL
LAST-PART.OPERATORS:=:
OPERATORS:=o14,o13,o11,o10,o17,o6,o7,o19
```

Ćwiczenie 8.1.

Zadanie polega na napisaniu programu podającego ciąg operatorów (w odpowiedniej kolejności), których zastosowanie pozwoli na przejscie od zadanego stanu początkowego do zadanego stanu końcowego.

Operator	Warunki wstępne	Warunki spełnione
o1	1,2,11	6
o2	6	7
o3	7	14
o4	12	13
o5	11	12,26
o6	11,21,23	19
o7	19	20
o8	5	11
o9	4	5
o10	17	18
o11	10,16	17,24
o12	10	4
o13	3,9	10
o14	8	9
o15	15	16
o16	22	17
o17	18	11
o18	12	7
o19	20	14
o20	24	25

Tabela 8.2: Lista przykładowych operatorów wraz z warunkami jakie powinny zostać spełnione i jakie zostaną spełnione. operatory wraz z warunkami wstępnymi i końcowymi

	4	5	6	7	9	10	11	12	13	14	16	17	18	19	20	24	25	26
1	-	-	1	2	-	-	-	-	-	2	-	-	-	-	-	-	-	-
2	-	-	1	2	-	-	-	-	-	3	-	-	-	-	-	-	-	-
3	12	9	1	2	-	13	10	5	4	3	-	11	10	6	7	13	11	5
4	-	9	8	2	-	-	8	5	5	3	-	-	-	6	6	-	-	5
5	-	-	1	1	-	-	8	8	8	2	-	-	-	6	6	-	-	5
6	-	-	-	2	-	-	-	-	-	3	-	-	-	-	-	-	-	-
7	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-
8	12	9	8	2	14	14	9	8	4	3	-	11	10	6	7	11	11	5
9	12	9	8	2	-	13	9	8	4	2	-	11	10	6	7	11	20	5
10	12	12	1	1	-	-	12	5	5	2	-	11	10	6	7	11	20	5
11	-	-	1	1	-	-	-	5	5	3	-	-	-	6	6	-	-	5
12	-	-	-	18	-	-	-	-	4	3	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
15	-	-	10	17	-	-	17	5	4	7	15	11	10	6	7	-	-	17
16	-	-	10	17	-	-	10	17	5	6	-	11	10	17	6	-	-	5
17	-	-	1	1	-	-	17	17	17	17	-	-	10	10	10	-	-	17
18	-	-	1	1	-	-	17	17	17	17	-	-	-	17	17	-	-	17
19	-	-	-	-	-	-	-	-	-	19	-	-	-	-	7	-	-	-
20	-	-	-	-	-	-	-	-	-	19	-	-	-	-	-	-	-	-
21	-	-	-	-	-	-	-	-	-	7	-	-	-	6	6	-	-	-
22	-	-	1	1	-	-	10	10	10	10	-	16	16	17	7	-	-	5
23	-	-	-	-	-	-	-	-	6	-	-	-	-	6	6	-	-	-
24	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	20	-
25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Tabela 8.3: Zbiór operatorów usuwających najbardziej istotną różnicę pomiędzy stanem początkowy (wiersz) a stanem końcowym (kolumna). Z tabeli usunięto kolumny reprezentujące stany końcowe dla których nie istnieje żaden operator przprowadzający z pewnego stanu początkowego (kolumny 1,2,3,8,15,21,22,23).

Rozdział 9

Problem wyboru optymalnego ruchu

9.1 Minimax

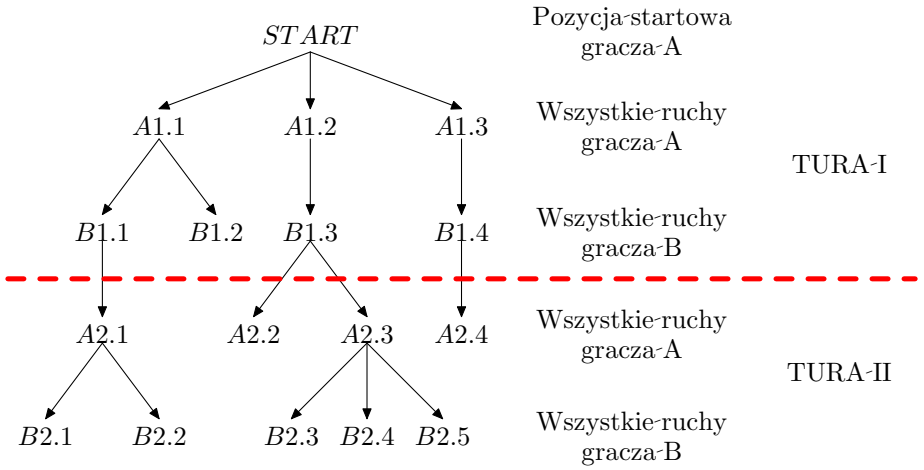
Bardzo wygodną strukturą danych pozwalającą reprezentować stan i przebieg gry (szczególnie gier dwuosobowych) jest drzewo. Węzły drzewa reprezentują stan gry po wykonaniu ruchu przez jednego z graczy. Gałąź przechodząca od węzła x do węzła y oznacza, że przyjmując za stan bieżący stan x jeden z graczy może wybrać taki ruch, który spowoduje, że stan gry (czyli na przykład sytuacja na planszy) będzie równy y. Poziomy drzewa reprezentują wszystkie ruchy, jakie może wykonać jeden z graczy w danej turze. Każdy węzeł ma przypisaną liczbę wyrażającą wartość reprezentowanej pozycji z punktu widzenia gracza, dla którego zaczęliśmy konstruować drzewo. W przykładowym drzewie gry z rysunku 9.1 graczem tym jest gracz A.

Zadanie nasze polega teraz na wyznaczeniu ruchu leżącego na ścieżce prowadzącej od stanu początkowego do tego ze stanów, który daje nam pewność, że nie zdobędziemy mniej jak x punktów, przy czym x jest największą z najmniejszych wartości jakie możemy zdobyć[1]. Brzmi to trochę zawile, ale ma wiele sensu.

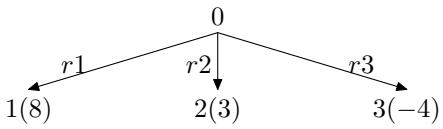
Rozważmy drzewo gry z rysunku 9.2 (każdy stan oceniany jest od -10 - przegrana gracza A do +10 - wygrana gracza A) Będąc w stanie 0 i mając wykonać ruch, gracz A z pewnością wybierze przjście (czyli wykona ruch powodujący zmianę na planszy) do stanu 1, gdyż stan ten zapewnia mu największą ilość punktów. Rozwińmy jednak drzewo gry o jeden dodatkowy poziom, który powstanie po wykonaniu ruchu przez gracza B (rysunek 9.3, wszystkie stany oceniane są z punktu widzenia gracza A)

Otóż w odpowiedzi na ruch $r1$ gracz B może odpowiedzieć ruchami $r4$, $r5$ i $r6$. Zakładając, że gracz B stosuje kryterium zdrowego rozsądku, to znaczy wybiera najkorzystniejszy dla siebie ruch, wykona on ruch $r5$ a to oznacza dla gracza A zdecydowane pogorszenie poprzedniej świetnej sytuacji. Zdecydowanie lepiej dla A wykonać taki ruch, który zagwarantuje, że po odpowiedzi gracza B gracz A straci najmniej. Ruchem tym jest $r2$. W najgorszym razie znajdziemy się bowiem w sytuacji ocenianej na -2. Owszem, wybierając $r1$ gracz A mógłby znaleźć się w stanie ocenianym na 9, ale pamiętajmy, że B nie gra przeciw sobie więc raczej ruchu $r4$ nie wykona. Przykład ten pokazuje także inną rzecz. Bardzo istotna jest głębokość analizy, czyli wysokość drzewa. Najlepiej, jeśli możnaby wygenerować i przeszukać całe drzewo, co dla większości gier jest... niemożliwe. Problem stanowi tutaj czas potrzebny na realizację takiego zadania (tj. generowania i

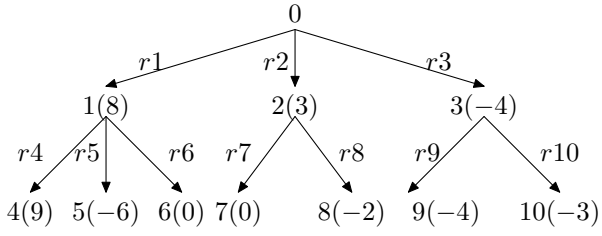
[1] Wyznaczamy ruch a nie całą ścieżkę, gdyż zależy ona od decyzji przeciwnika. Jedyne co możemy zrobić to właśnie wyznaczyć ruch na tej ścieżce leżący.



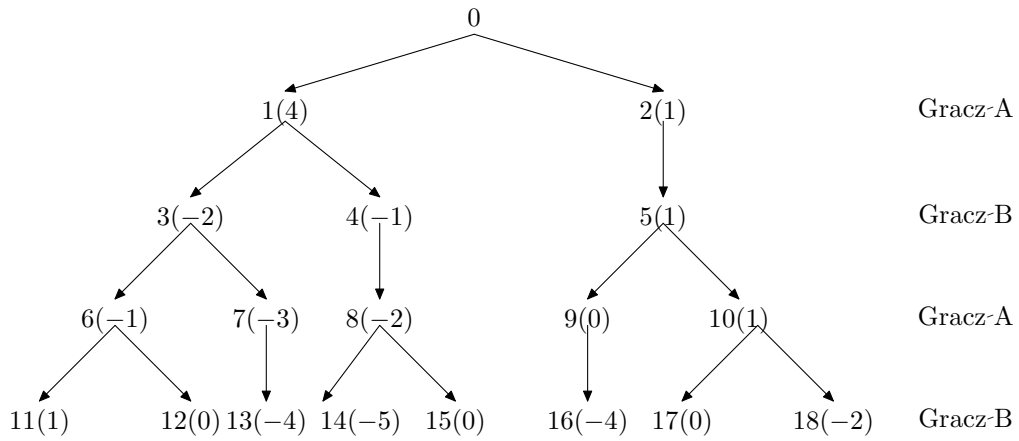
Rysunek 9.1: Przykładowe drzewo gry dla gracza A.



Rysunek 9.2: Przykładowy pierwszy poziom drzewa gry dla gracza A.



Rysunek 9.3: Dalsze rozwinięcie drzewa gry z rysunku 9.3.



Rysunek 9.4: Drzewo gry wraz z ocenami stanów z punktu widzenia gracza A.

przeszukiwania drzewa). Na szczęście dla prostych gier, jak ta z zadania, wystarczają 2-3 poziomy.

Przyjmijmy więc, że drzewo (utworzone dla gracza A) ma kilka poziomów odpowiadających kilku turom w pewnej grze. Jak zatem wybrać odpowiednią ścieżkę? Analizę drzewa rozpoczniemy od jego liści i posuwać będziemy się w kierunku korzenia. Liściom przypisjemy wartości gry odpowiadające reprezentowanym przez nie stanom. Dla pozostałych węzłów powstałych w wyniku ruchu gracza B będzie to **maksimum** (max) z wartości oceny dzieci, natomiast dla węzłów odpowiadających ruchowi gracza A **minimum** (min) z wartości oceny dzieci (max i min liczone jest osobno dla każdego węzła).

Postępowanie przeanalizujemy dla przykładowego drzewa gry z rysunku 9.4.

1. Postępując zgodnie z powyższym opisem rozpoczynamy jego analizę od liści (węzłów o numerach 11-18). Każdemu z nich przypiszemy wartość reprezentowanego przez niego stanu gry. Stąd drzewo przyjmie postać jak na rysunku 9.5.

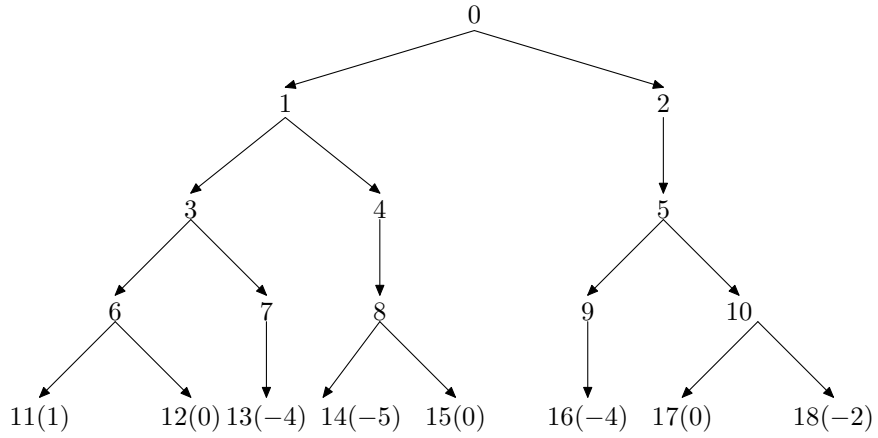
2. Wartości dla węzłów z tury II odpowiadających ruchowi gracza A będą minimami z wartości przechowywanych przez dzieci. Stąd drzewo przyjmie postać jak na rysunku 9.6.

3. Wartości dla węzłów odpowiadających ruchowi gracza B w turze I to maksima z wartości ich dzieci (rysunek 9.7):

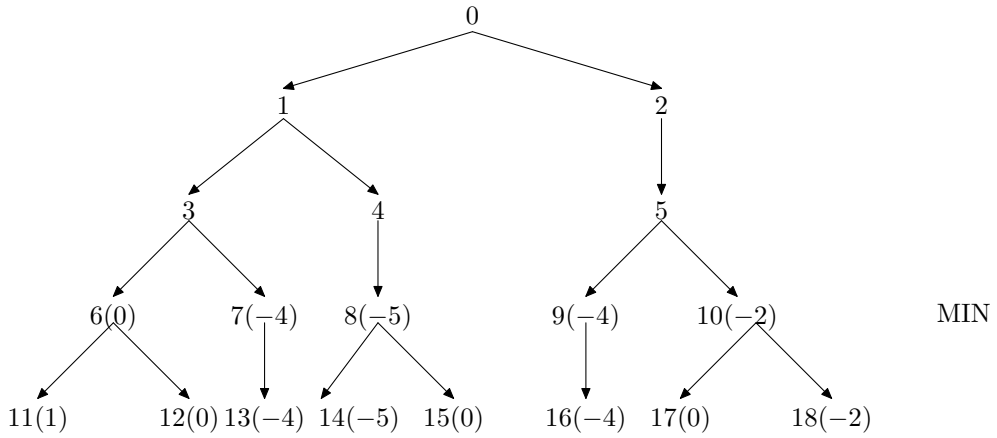
4. Wartości dla węzłów odpowiadających ruchowi gracza A w turze I to ponownie minima z wartości ich dzieci (rysunek 9.8):

5. Ostatecznie, do korzenia zwracamy maxymalną wartości jego dzieci, czyli -2 (rysunek 9.9):

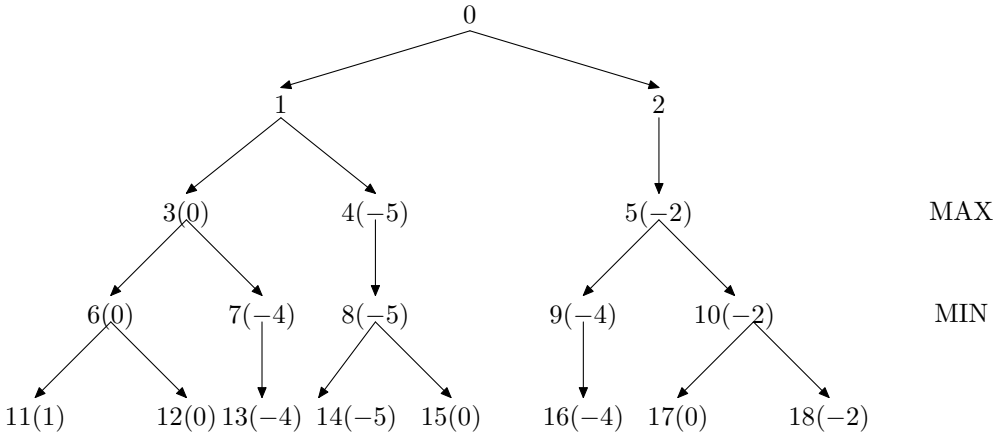
Jeśli więc wykonamy ruch, który spowoduje, że znajdziemy się w stanie 2 to w najgorszym przypadku znajdziemy się w stanie ocenianym na -2 (przy oczywistym założeniu, że nie gramy przeciw sobie i staramy się uzyskać jak najwięcej punktów).



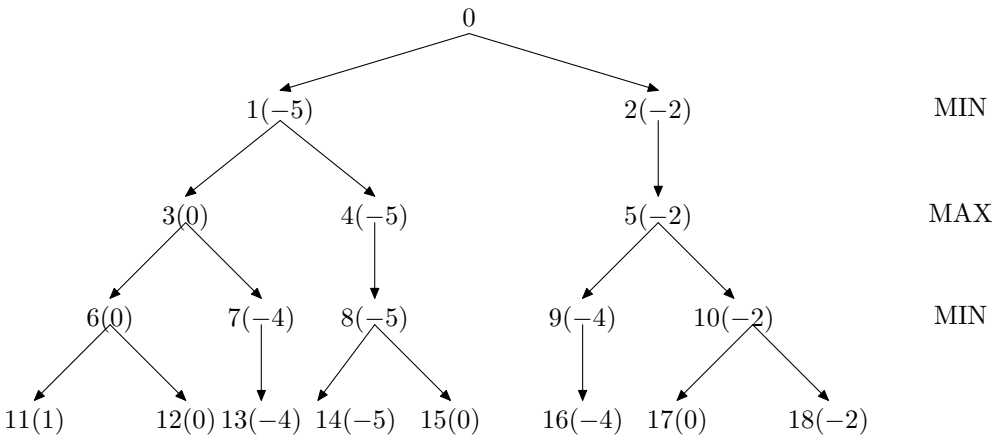
Rysunek 9.5: Pierwszy krok algorytmu Mini-Max.



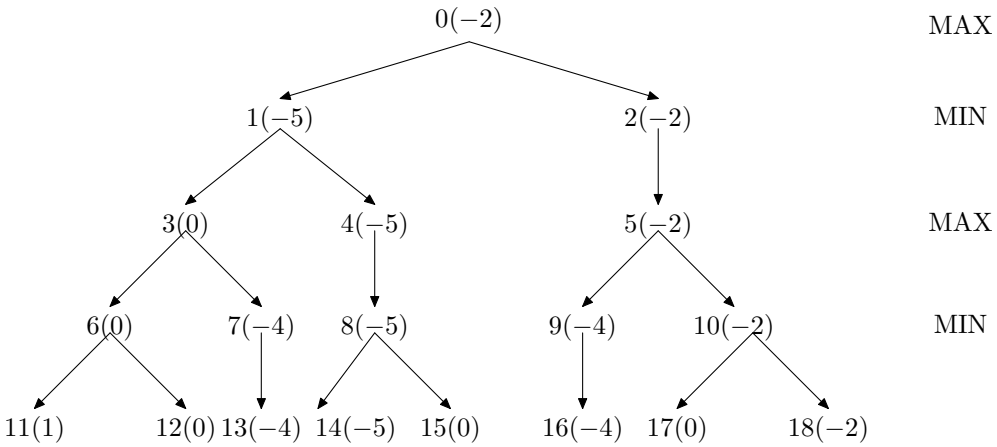
Rysunek 9.6: Drugi krok algorytmu Mini-Max.



Rysunek 9.7: Trzeci krok algorytmu Mini-Max.



Rysunek 9.8: Czwarty krok algorytmu Mini-Max.



Rysunek 9.9: Piąty krok algorytmu Mini-Max.

9.2 Opis algorytmu

Przedstawimy teraz ten algorytm w bardziej formalnej postaci. Korzystać będziemy w tym opisie z następujacych funkcji i stałych

- `MOVEGEN(Position, Player)` – funkcja ta generuje wszystkie ruchy jakie może wykonać gracz `Player` traktując stan `Position` jako stan początkowy. Zwraca ona listę, w której każda pozycja zawiera nowy stan wraz z opisem ruchu, który do niego doprowadził. Dla naszego przykładu wywołanie `MOVEGEN(1,B)` zwraca listę `{{3,r3},{4,r4}}`.
- `ASSESS(Position,Player)` – funkcja, która ocenia stan `Position` według kryteriów gracza `Palyer`, na przykład `ASSESS(3,B)` zwróci wartość -2.
- Ponieważ jest to algorytm rekursywny zatem istotne jest przerwanie w pewnym momencie ciągu „samowywołań". Uczynimy tak, gdy funkcja `DEEP_ENOUGH(Position, Depth)` zwróci wartości `TRUE`.
- Stałe `THE_BEST` i `THE_WORSE` oznaczają największą i najmniejszą wartość jaką może zwócić funkcja `ASSESS`.

Algorytm Minimax

`MINIMAX(Position, Depth, Player)`

1. Jeśli `DEEP_ENOUGH(Position,Depth)=TRUE` to zwróć strukturę

   ```
   {
      VALUE:=ASSESS(Position,PLAYER_A)
      PATH:=NULL
   }
   ```

2. W przeciwnym razie wygeneruj kolejny poziom drzewa wywołując funkcję

   - `MOVGEN(Position, PLAYER_B)` jeśli `Player=PLAYER_A`
   - `MOVGEN(Position, PLAYER_A)` jeśli `Player=PLAYER_B`

   a zwrócone przez nią wartości zapisz w `SUCCESSORS`.

3. Jeśli `SUCCESSORS=NULL`, oznacz to, że nie ma już więcej możliwych do wykonania ruchów. Zwracamy zatem strukturę

   ```
   {
      VALUE:=ASSESS(Position,PLAYER_A)
      PATH:=NULL
   }
   ```

4. W przeciwnym razie jeśli

   (a) `Player=PLAYER_A` to wykonaj:

```
PATH:=NULL
VALUE:=THE_BEST;
Dla każdego stanu S z SUCCESSORS będącego wynikiem ruchu R wykonaj:
(T_VALUE,T_PATH):=MINIMAX(S,Depth+1,PLAYER_B)

Jeśli VALUE > T_VALUE to
   VALUE:=T_VALUE
   PATH:=R+T_PATH
```

(b) `Player=PLAYER_B` to wykonaj:

```
PATH:=NULL
VALUE:=THE_WORSE;
Dla każdego stanu S z SUCCESSORS będącego wynikiem ruchu R wykonaj:
(T_VALUE,T_PATH):=MINIMAX(S,Depth+1,PLAYER_A)

Jeśli VALUE < T_VALUE to
   VALUE:=T_VALUE
   PATH:=R+T_PATH
```

5. Zwróć strukturę

```
{
   VALUE
   PATH
}
```

Algorytm został przedstawiony wyjątkowo nie w postaci ogólnej aby można było zauważyć momenty wybierania wartości minimalnej lub maksymalne. Ponad to jest to postać odpowiednia dla gracza A. Chcąc użyć go dla gracza B należy zamienić wystąpienia `PLAYER_A` i `PLAYER_B` na `PLAYER_B` i `PLAYER_A`.

Pozostał jeszcze technicznych „drobiazg": jak z tgo algorytmu skorzystać wywolac? Popronujemy tak:

1. Generujemy wszystkie możliwe ruchy dla `PLAYER_A` jakie może wykonać dla bieżącego układu planszy.
2. Dla każdego ruchu wygenerowanego w poprzednik kroku wywołujemy `MINIMAX(kolejny_wygenerowany_ruch,1,PLAYER_A)`.
3. Z wartości zwróconych przez wszystkie wywolanie minimax-u wybieramy największą.

Ćwiczenie 9.1.

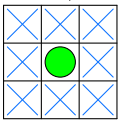
Zadanie polega na napisaniu gry, w której poczynaniami jednego z graczy steruje komputer. Kolejny ruch do wykonania przez komputer ma być znajdowany w oparciu o przedstawiony algorytm. Zasady gry są następujace

1. *Gramy na planszy podobnej do planszy szachowej. Rozmiar dowolny, ale nie mniejszy niż 8x8 (nie musi to być kwadrat). Wszystkie pola mają jednakowy kolor.*
2. *Dwóch graczy posługuje się pionkami dwóch różnych kolorów.*

3. *Na każdym polu może stać tylko jeden pionek.*

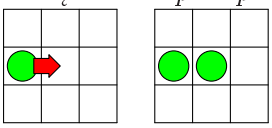
4. *Początkowo na planszy każdy z graczy ma po jednym pionku; ustawione one są w przeciwległych rogach planszy.*

5. *Sąsiedztwem pionka nazywamy wszystkie 8 pól, które bezpośrednio do niego przylegają (krzyżyki na rysunku).*

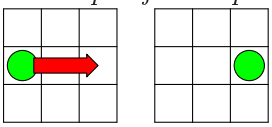


6. *Możliwe są dwa rodzaje ruchów:*

   Rozmnożenie: *przejście z pola bieżącego na sąsiednie polegające na postawieniu na sąsiednim polu pionka.*

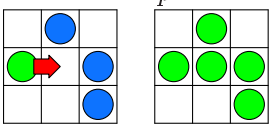


   Przeskok: *przejście z pola bieżącego na pole oddalone o 1 pole.*



7. *Ruch możliwy jest w pionie i poziomie (ale jeśli ktoś chce może także dodać skosy).*

8. *Podczas fazy ruchu jednego z graczy wszystkie pionki przeciwnika, które znajdą się w otoczeniu przemieszczonego pionka zmieniają kolor na kolor wykonującego ruch.*



9. *Celem gry jest zdobycie przewagi w postaci większej liczby pionków na planszy. Gra toczy się do momentu zapełnienia wszystkich pól na planszy lub niemożności wykonania ruchu przez jednego z graczy.*

Rozdział 10

Symulacja pożaru

//wersja wstepna!!! wymaga sprawdzenia!!!//

W ćwiczeniu tym wykorzystamy koncepcję automatu komórkowego do symulowania rozprzestrzeniania się ognia.

10.1 Podejście pierwsze

Najprostsze podejście do tego tematu polega na zdefiniowaniu następujących warunków

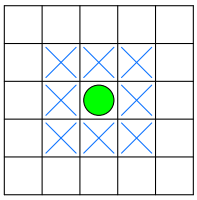
1. Każda komórka jest w jednym z dwóch stanów: **pali się** lub **nie pali się**.
2. Sąsiedztwem dla komórek są wszystkie komórki które bezpośrednio z nimi sąsiadują (rysunek 10.1).
3. Jeśli w sąsiedztwie komórki, która się nie pali, znajduje się choć jedna komórka paląca się, wówczas komórka ta się zapala.

I choć pożar rządzony takimi prawami nie zachowuje w sposób naturalny, to jednak warto od tego zacząć tworzenie aplikacji a następnie stopniowo wprowadzać udoskonalenia i modyfikacje. Takie podejście ma kilka oczywistych wad.

- Pożar rozprzestrzenia się bardzo nienaturalnie, bo „koncentrycznie" względem punktu w którym się rozpoczął.
- Zajęcie się kolejnych komórek jest pewne o ile tylko w ich sąsiedztwie jest choć jedna komórka paląca się. Tak więc „siła", czy „zagęszczenie" ognia nie ma żadnego znaczenia.
- Nie ma rozróżnienia pomiędzy komórkami jeśli chodzi o ich łatwopalność, czy też czas palenia (brak zdefiniowanej reguły określającej kiedy komórka się wypala).

10.2 Podejście drugie

Spróbujemy teraz wyeliminować wady wskazane w poprzednim podrozdziale. Zwiększmy zatem ilość stanów w jakich może znaleźć się dana komórka.



Rysunek 10.1: Najbardziej podstawowy przykład sąsiedztwa (pola „z krzyżykiem").

Nie pali się. Komórka jeszcze się nie paliła. Zawiera pewną określoną ilość materiału palnego. Najprościej jest przyjąć, że ilość „paliwa" U jest liczbą z przedziału $[0, 255]$, przez co łatwo będzie można prezentować stan takiej komórki.

Pali się. Komórka pali się. Oznacza to, że w każdej iteracji ubywa pewna część paliwa. Siłę ognia określamy za pomocą liczby z przedziału $[0, 255]$.

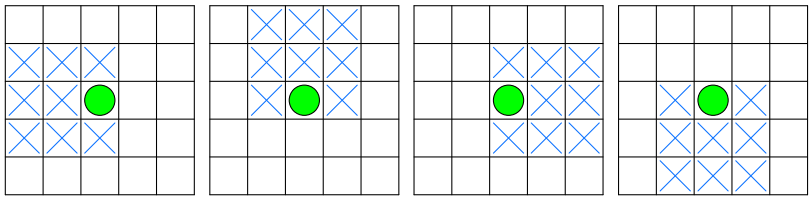
Wypalona. Komórka wypalona. Oznacza to, że w wyniku pożaru paliwo znajdujące się w komórce zostało zużyte i komórka taka nie może ponownie się zapalić.

Wprowadzamy następujące zasady rządzące zmianą stanu komórek

1. Prawdopodobieństwo zapłonu komórki o współrzędnych (i, j) jest wprost proporcjonalnie do sumarycznej siły ognia w komórkach sąsiednich. Obliczamy je według wzoru
$$P_Z(i,j) = \begin{cases} \frac{S_F(i,j)}{1600} & \text{dla} \quad S_F(i,j) \leq 1600 \\ 1 & \text{dla} \quad S_F(i,j) > 1600 \end{cases}$$
gdzie $S_F(i, j)$ jest wyrażeniem obliczanym jako
$$S_F(i,j) = \sum_{k=1}^{8} F_k(i,j)$$
a $F_k(i, j)$ to siła ognia w k-tej komórce sąsiadującej z komórką (i, j). Zauważmy, że maksymalna wartość dla $S_F(i, j)$ to 2040 a próg 1600 został wybrany arbitralnie. Przyjęto, że powyżej niego „stężenie" ognia jest tak duże, że zapłon następuje już zawsze.

2. Jeśli komórka się pali i jest w niej jeszcze paliwo, to ogień zwiększa się (ogień „rozpala się") o pewną stałą wartość f. Zamiast stałej można używać pewnej zależności od siły ognia w komórkach sąsiednich, np.
$$\Delta F(i,j) = \begin{cases} +f & \text{dla} \quad S_F(i,j) \in (0, 500] \\ +f \cdot 3 & \text{dla} \quad S_F(i,j) \in (500, 1300] \\ +f \cdot 10 & \text{dla} \quad S_F(i,j) \in (1300, 2040] \end{cases}$$
gdzie $\Delta F(i, j)$ oznacza przyrost siły ognia w komórce (i, j). Pamiętamy przy tym, że górnym ograniczeniem dla siły ognia jest wartość 255.

3. Jeśli komórka się pali a nie ma w niej paliwa, to ogień zmniejsza swoją siłę albo o stałą wartość f, albo podobnie jak to było określone w punkcie powyżej, według



Rysunek 10.2: Modelowanie czynnika wiatru przez przemieszczanie sąsiedztwa: (a) – wiatr zachodni, (b) – wiatr północny, (c) – wiatr wschodni, (d) – wiatr południowy.

pewnej zależności od siły ognia w komórkach sąsiednich (żar lub ogień z komórek sąsiednich utrzymuje temperaturę w komórce) , np.

$$\Delta F(i,j) = \begin{cases} -f \cdot 10 & \text{dla} \quad S_F(i,j) \in (0,500] \\ -f \cdot 3 & \text{dla} \quad S_F(i,j) \in (500,1300] \\ -f \cdot & \text{dla} \quad S_F(i,j) \in (1300,2040] \end{cases}$$

Pamiętamy przy tym, że dolnym ograniczeniem dla siły ognia jest wartość 0.

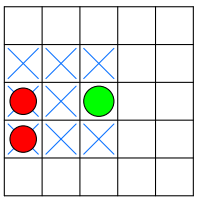
4. Ogień w komórce powoduje, że ubywa paliwa. Szybkość ubywania paliwa jest proporcjonalna do wielkości ognia w komórce

$$\Delta U(i,j) = \begin{cases} -g & \text{dla} \quad F(i,j) \in (0,100] \\ -g \cdot 3 & \text{dla} \quad F(i,j) \in (100,200] \\ -g \cdot 10 & \text{dla} \quad F(i,j) \in (200,255] \end{cases}$$

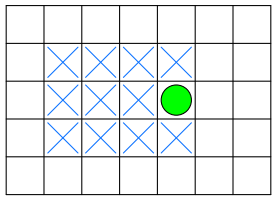
gdzie g jest pewną stałą.

10.3 Podejście trzecie

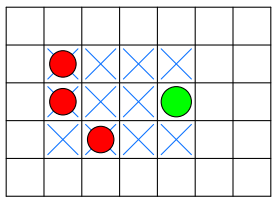
Choć efekt uzyskany w poprzednim podrozdziale przedstawia się już całkiem rzeczywiście, to można go prostymi zabiegami wzbogacić. Spróbujmy dodać czynnik wiatru, który będzie powodował, że w pewnych kierunkach ogień będzie rozchodził się szybciej niż w pozostałych. Chcąc wprowadzić czynnik wiatru najprościej jest zdefiniować nowe (inaczej wyglądające) sąsiedztwo. Wygląd sąsiedztwa „tradycyjnego", tzn. bez wiatru i odpowiadającego czterem podstawowym kierunkom przedstawia rysunek 10.2. Zauważmy, że sąsiedztwo przemieszcza się w kierunku z którego wieje wiatr. Postępujemy tak dlatego, aby na stan danej komórki wpływały komórki znajdujące się po stronie nawietrznej. Mówiąc inaczej, wiatr będzie przenosił ogień z tych komórek. Na rysunku 10.3 przedstawiono hipotetyczną komórkę (kolor zielony) i „palące się" komórki (kolor czerwony), które mimo, że nie mają bezpośredniej styczności z rozważaną komórką to jednak będą miały wpływ na jej stan. Jak można się łatwo przekonać, tak niewielka modyfikacja wystarcza już do tego, aby uzyskać bardzo ciekawy efekt. Dodatkowo można wprowadzić siłę wiatru, od której będzie zależało sąsiedztwo. Powinno się ono zwiększać w kierunku z którego wieje wiatr (patrz rysunek 10.4). W takim przypadku ogień powinien przenosić się na jeszcze większe odległości (patrz rysunek 10.5).



Rysunek 10.3: Hipotetyczna komórka (kolor zielony) i „palące się" komórki (kolor czerwony) znajdujące się w jej otoczeniu (w przypadku wiatru zachodniego). Mimo, że „palące się" komórki nie mają bezpośredniej styczności z rozważaną komórką to jednak będą miały wpływ na jej stan.



Rysunek 10.4: Sąsiedztwo dla silnego wiatru zachodniego.



Rysunek 10.5: Sąsiedztwo dla silnego wiatru zachodniego. Jak widać wpływ na stan rozważanej komórki (kolor zielony) będą miały nawet odległe „palące się" komórki (kolor czerwony).

Rozdział 11

Automaty

W rozdziale tym pozostaniemy w kręgu zagadnień wykorzystujących grafy czy to do reprezentacji danych jako takie czy też jako narzędzie opisu ich wzajemnych zależności. Rozszerzymy go jednak o kolejne pole zastosowań a mianowicie modelowanie za pomocą grafów schematów działań.

11.1 Automat skończony

Automat skończony (ang. *finite state machine*, FSM lub *finite state automaton* (pl: *automata*), FSA lub *state machine*, SM) to abstrakcyjny, matematyczny, iteracyjny model zachowania systemu dynamicznego złożonego ze

- skończonej ilości **stanów**,
- tablicy (funkcji) definiującej przejścia między tymi stanami,
- akcji.

Ponieważ bieżący stan w takim automacie w pewnym sensie jest determinowany przez stany wcześniejsze, stąd można mówić, że automaty te posiadają pewną formę pamięci (która pozwala odtworzyć sekwencję przejść od stanu początkowego do bieżącego). Tablica przejść opisuje pod wpływem jakich zdarzeń (warunków) może nastąpić przejście z jednego stanu do innego. Ze względu na charakter przejść między stanami, wyróżnia się **deterministyczne** i **niedeterministyczne** automaty skończone; w dalszej części mówiąc *automat skończony* będziemy mieć na myśli jego wersję deterministyczną. Akcja jest opisem działania jakie ma zostać wykonane w wyniku stosowania tablicy przejść. Wyróżnić możemy

- *Akcje wejściowe* (ang. *entry action*), wykonywane wówczas gdy wchodzimy do danego stanu. Dla przykładu, akcje tego rodzaju wykorzystuje automat Moore'a (ang. *Moore machine*), tzn. jego akcja zależy od stanu w jakim znalazł się automat, ale zupełnie nie jest istotne jakie zdarzenie to spowodowało.
- *Akcje wyjściowe* (ang. *exit action*), wykonywane wówczas gdy wychodzimy z danego stanu
- *Akcje związane z wejściem* (ang. *input action*), wykonywane wówczas gdy wyboru przejścia dokonujemy biorąc pod uwagę stan bieżący i sygnał wejściowy. Dla przykładu, akcje tego rodzaju wykorzystuje automat Mealy'ego (ang. *Mealy machine*),

tzn. jego akcja zależy od stanu w jakim znajduje się automat zanim przejście nastąpi i symbolu wejściowego.

Ogólna zasada działania automatów skończonych jest następująca.

1. Automat, symbol po symbolu, wczytuje pewne słowo zbudowane w oparciu o zadany alfabet[1].
2. Każdy wczytany symbol może spowodować, zgodnie z tabelą przejść, przejście do innego stanu i podjęcie pewnej akcji.
3. Automat może przyjmować skończoną ilość różnych stanów.
4. Powiemy, że automat akceptuje jakieś słowo, gdy po jego całkowitym wczytaniu znajdzie się w jednym z tzw. stanów końcowych (końcowych akceptujących).

Intuicyjnie automat skończony możemy wyobrazić sobie jako urządzenie składające się z taśmy i znajdującej się nad nią głowicy czytającej. Taśma (może być nieskończona z jednej strony) podzielona jest na komórki. W każdej komórce wpisany jest jeden symbol ze słowa. Głowica odczytuje kolejne symbole z taśmy (w danym momencie zawartość jednej i tylko jednej komórki), zmienia stan automatu zgodnie z tablicą przejść i przesuwa głowicę o jedną komórkę w stronę końca słowa. Jeśli po wczytaniu wszystkich symboli danego słowa, zapisanego na taśmie, głowica znajduje się w jednym ze stanów końcowych, oznacza to, że słowo jest akceptowane przez automat.

Formalnie automat przedstawiamy jako piątkę uporządkowaną

$$\{Q, \Sigma, \delta, q_0, F\},$$

gdzie:

- $Q = \{q_1, \ldots, q_n\}$ – skończony zbiór n stanów;
- Σ – skończony zbiór symboli wejściowych nazywany alfabetem. Może mieć on różne postacie np.: binarną $\Sigma = \{0, 1\}$, wtedy dane wejściowe mogą mieć postać 01101100, alfabetyczną $\Sigma = \{a, b, c, d\}$ z wejściem postaci *abcccdaa* lub jak ma to miejsce w przypadku gier, składać się z wydarzeń jakie mogą zajść w czasie rozgrywki $\Sigma = \{strzal,\ blysk,\ krzyk\}$;
- δ – funkcja przejścia, odwzorowująca $Q \times \Sigma$ w Q czyli taka, która każdemu stanowi $q \in Q$ i każdemu symbolowi wejściowemu $a \in \Sigma$ przyporządkowuje nowy stan $\delta(q, a)$;
- q_0 – należące do Q, stan początkowy, czyli taki od którego automat rozpoczyna działanie;
- $F \subseteq Q$ – zbiór stanów końcowych.

Graficzną reprezentacją automatu skończonego jest graf skierowany, którego wierzchołki odpowiadają stanom automatu, krawędzie zaś przejściom ze stanu do stanu, zgodnie ze zwrotem strzałki. Każda krawędź opatrzona jest etykietą symbolu wejściowego, dla którego zachodzi dane przejście.

Mówimy, że automat akceptuje dany łańcuch wejściowy, jeśli ciąg przejść odpowiadający poszczególnym symbolom łańcucha prowadzi od stanu początkowego do jakiegoś stanu końcowego.

[1] Pojęcia *słowa* i *alfabetu* zostaną wyjaśnione w dalszej części a w tym miejscu zakładamy ich intuicyjne rozumienie.

11.1.1 Automat skończony jako model działania

Maszyna Turinga jest generalizacją automatu skończonego operującą na nieskończonej pamięci. Automat skończony jako model działania urządzenia – Maszyna Turinga.

Automat skończony jako model działania w sytuacji awaryjnej.

11.2 Automat Moore'a.

Automatem Moore'a nazywamy automat z wyjściem rozszerzonym. Ten rodzaj automatów (określany również mianem **maszyny stanów**) różni się od wcześniej omówionego tym, że sygnały wyjściowe generowane są każdorazowo po przetworzeniu symbolu wejściowego. Nie występuje tu pojęcie stanu końcowego.

Automat Moore'a definiujemy jako szóstkę uporządkowaną

$$\{Q, \Sigma, \Delta, \delta, \lambda, q_0\}$$

gdzie:

- Q, Σ, δ, q_0 są określone tak jak w przypadku automatów skończonych z wyjściem binarnym;
- Δ – alfabet wyjściowy;
- λ – funkcja zwracająca wyjście związane z każdym ze stanów, czyli odwzorowująca Q w Δ. Wyjściem zwróconym dla łańcucha wejściowego

  $$a_1 a_2 \dots a_n, \quad n \geq 1$$

  jest

  $$\lambda(q_1)\lambda(q_2)\dots\lambda(q_n)$$

  gdzie q_0, q_1, q_2,..., q_n jest ciągiem stanów takim, że $\delta(q_{i-1}, a_i) = q_i$ dla $1 \leq i \leq n$.

Należy zauważyć, że model ten zwraca sygnał wyjściowy dla łańcucha wejściowego dowolnej długości, w szczególności 0. Mamy wtedy do czynienia z tzw. pustym wejściem, któremu odpowiada wyjście $\lambda(q_0)$.

Adaptując dla potrzeb gry komputerowej dowolny typ automatu z wyjściem rozszerzonym, możemy traktować symbole wyjściowe jako akcje (lub zbiory akcji), które należy podjąć w związku z przejściem z jednego stanu do drugiego.

Rozdział 12

Symulacja zachowania

Celem projektu jest napisanie symulacji życia 4 osobników poruszających się po zadanej planszy na podstawie zdefiniowanych reguł. Narzędziem sterującym będzie automat Moore'a (patrz rozdział 11). Planszą, po której poruszają się osobniki jest prostokąt podzielony na komórki równej wielkości tak, że możemy ją reprezentować jako macierz wymiaru $n \times m$. Każdy z osobników jest na początku umieszczony w innym rogu planszy. Osobniki poruszają się w pionie, poziomie i po skosie o jedno pole. Każdy osobnik ma tzw. wartość punktów życia. Maksymalna wartość typ punktów jest określona zmniejszana wraz z wykonaniem ruchu przez osobnika. Na planszy znajdują się specjalne pola z pożywieniem. Jeżeli osobnik wejdzie na takie pole regeneruje punkty życia i dodatkowo zapamiętuje gdzie jest punkt z pożywieniem i ile jest w nim pożywienia. Każdy punkt pożywienia ma określoną wartość, która jest pomniejszana przy regeneracji punktów życia przez osobnika. Nie jest to wartość odnawialna, tzn. w czasie gry wartość pożywienia w danym punkcie może się tylko zmniejszać, aż osiągnie wartość zero czyli zniknie.

Jeżeli osobnik wejdzie na pole, w którym znajduje się inny osobnik następuje rozmowa w trakcie której osobniki przekazują sobie informacje o miejscach z pożywieniem. Osobnik zapamiętuje informacje o nowych punktach pożywienia lub aktualizuje swoje informacje. Może się bowiem zdarzyć, że dwa osobniki mają informacje o jednym punkcie z żywnością. Informacją bardziej aktualną jest ta, która ma mniejszą wartość pożywienia dla tego samego punktu.

Dla osobnika korzystną sytuacją jest ta, w której ma maksymalną liczbę punktów. Jeśli natomiast liczba punktów życia osiąga minimalny poziom osobnik powinien pójść do punktu z pożywieniem i zregenerować punkty życia. Dokonuje tego na podstawie posiadanych informacji. Jeśli ma do wyboru kilka możliwości wybiera najbardziej dla siebie korzystną (w sensie odległości i liczby punktów do zregenerowania). Ponadto istotne jest aby osobnik posiadał jak najwięcej informacji o punktach z pożywieniem i aby te informacje były na bieżąco aktualizowane. Osobnik umiera gdy liczba punktów życia osiąga wartość 0. Ruch osobnika nie poszukującego pożywienia nie jest określony (może być losowy lub podlegać innym regułom).

W zależności od ilości punktów i sytuacji w jakiej się znajdujemy możemy określić stan osobnika. Automat Moore'a, dla powyższego zadania jest zdefiniowany poniżej:

$$Q = \{PATROL,\ POSZUKIWANIE,\ JEDZENIE,\ ROZMOWA,\ SMIERC\}$$

gdzie:

- PATROL – stan początkowy polegający na beztroskim zwiedzaniu okolicy.
- ROZMOWA – po napotkaniu osobnika dysponującego nowymi informacjami o punktach regeneracji, następuje krótka konwersacja i wymiana danych.
- POSZUKIWANIE – stan, w którym osobnik korzystając z wiedzy zdobytej w czasie PATROLU lub wymiany informacji z innymi osobnikami w stanie ROZMOWA, szuka miejsca, w którym mógłby odnowić swój zasób punktów życia.
- JEDZENIE – regeneracja sił.
- SMIERC – po wyczerpaniu puli punktów życia osobnik ginie.

Alfabet wejściowy jest zbiorem komunikatów postaci:

- K1: mała liczba punktów życia (1 - 4999),
- K2: duża liczba punktów życia (5000 - 9999),
- K3: maksymalna liczba punktów życia (10000),
- K4: wyczerpany zasób punktów życia (0),
- K5: spotkanie,
- K6: koniec rozmowy,
- K7: odnaleziono miejsce z pożywieniem,
- K8: wyczerpanie zasobów pożywienia.

Funkcja przejścia określona następująco:

$$(biezacy\ stan,\ symbol\ wejsciowy) \rightarrow (nowy\ stan)$$

- $(PATROL,\ K5) \rightarrow (ROZMOWA)$;
- $(PATROL,\ K1) \rightarrow (POSZUKIWANIE)$;
- $(ROZMOWA,\ K6) \rightarrow (PATROL)$;
- $(POSZUKIWANIE,\ K7) \rightarrow (JEDZENIE)$;
- $(POSZUKIWANIE,\ K4) \rightarrow (SMIERC)$;
- $(JEDZENIE,\ K3,\ K8 \wedge K2) \rightarrow (PATROL)$;
- $(JEDZENIE,\ K8 \wedge K1) \rightarrow (POSZUKIWANIE)$.

Brakujące parametry należy dodefiniować samodzielnie a całość najlepiej zilustrować w formie graficznej.

Rozdział 13

Drzewo decyzyjne jako reprezentacja bazy reguł

Drzewo decyzyjne jest takim drzewem, w którym:

- węzły odpowiadają testom przeprowadzanym na wartościach atrybutów reguł,
- gałęzie są możliwymi wynikami testów,
- liście reprezentują cześć decyzyjną.

Drzewa decyzyjne jako forma reprezentacji zbioru reguł zdobyły wśród programistów sztucznej inteligencji dużą popularność przede wszystkim ze względu na swą czytelność oraz znaczne zmniejszenie wymaganych do przechowania bazy, zasobów pamięci. Fakt ten zilustrowany jest tablicą 13.1 zawierającą przykładową bazę reguł, oraz rysunkiem 13.1 przedstawiającym odpowiadające jej drzewo decyzyjne. Wspomniana baza składa się z formuł ogólnej postaci

JEŻLI ... **I** ... **I** ... **TO** ....

W tym szczególnym przypadku przyjmują one postać

JEŻLI *natura* postaci jest *zła* **I** *uroda* postaci jest *pospolita* **I** *charyzma* postaci jest *duża* **I** *inteligencja* postaci jest *niska* **TO** moim *stosunkiem* do postaci jest *nienawiść*.

na podstawie których bot[1] potrafi określić swój stosunek do innego osobnika. Klasycznym przykładem zastosowania bazy reguł są gry. Baza reguł stanowi zbiór na podstawie którego tworzy się relacje definiujące zasady gry.

13.1 Algorytm konstruowania drzewa decyzyjnego

Algorytm konstruowania drzewa decyzyjnego jest realizowany przez funkcję rekurencyjną, która jako argument przyjmuje bazę reguł. Baza reguł składa się z atrybutów oraz przypisanych im wartości. U nas są to

[1] Bot jest programem wykonującym pewne czynności w zastępstwie człowieka. Czasem jego funkcją jest udawanie ludzkiego zachowania lub wykonywanie zautomatyzowanych czynności.

Lp.	Natura	Uroda	Charyzma	Inteligencja	Stosunek
1	dobra	piękna	duża	niska	uwielbienie
2	dobra	piękna	duża	wysoka	uwielbienie
3	neutralna	piękna	duża	niska	obojętność
4	zła	pospolita	duża	niska	nienawiść
5	dobra	pospolita	duża	niska	uwielbienie
6	neutralna	brzydka	mała	wysoka	obojętność
7	zła	brzydka	mała	niska	nienawiść
8	zła	brzydka	duża	wysoka	obojętność
9	neutralna	pospolita	mała	wysoka	obojętność
10	dobra	brzydka	mała	niska	obojętność
11	neutralna	piękna	mała	niska	obojętność
12	neutralna	brzydka	duża	niska	obojętność
13	dobra	pospolita	mała	wysoka	obojętność
14	zła	pospolita	duża	niska	nienawiść
15	zła	pospolita	mała	wysoka	obojętność

Tabela 13.1: Przykładowa baza reguł.

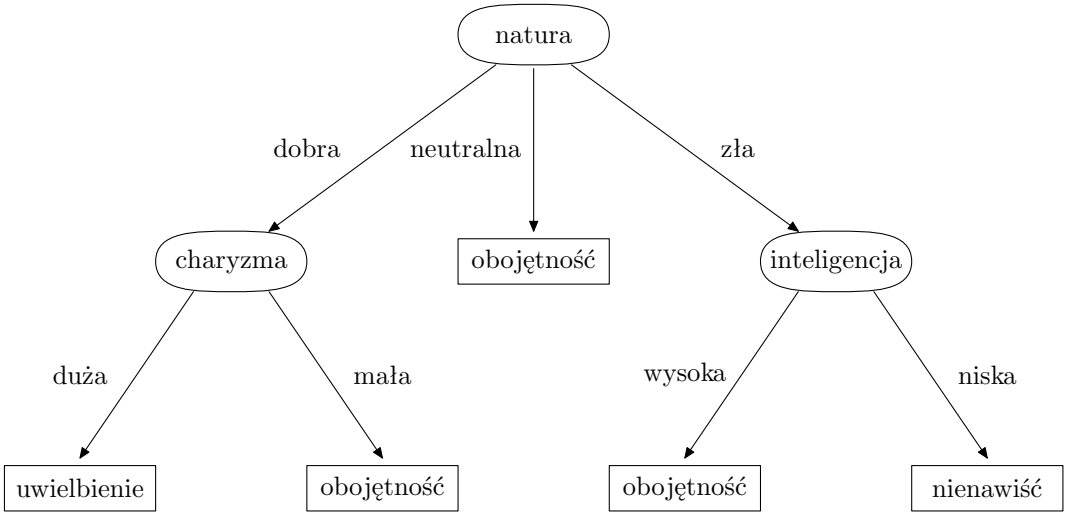
- Natura (dobra, neutralna, zła);
- Uroda (piękna, pospolita, brzydka);
- Charyzma (duża, mała);
- Inteligencja (niska, wysoka).

Pierwszy krok polega na wybraniu atrybutu, który będzie korzeniem drzewa. Tworzony jest nowy węzeł dla którego należy wybrać najlepszy atrybut, czyli taki który zapewni prostotę tworzonego drzewa. Prostota ta rozumiana jest jako możliwie najmniejsza liczba węzłów i liści oraz jak najkrótsze ścieżki łączące korzeń z liśćmi. Funkcję oceny poszczególnych atrybutów, tzw. **entropie** zbioru reguł P ze względu na atrybut t, definiujemy nastepujaco:

$$E_t(P) = \sum_{r \in R_t} \frac{\mid P_{tr} \mid}{\mid P \mid} \sum_{d \in C} -\frac{\mid P_{tr}^d \mid}{\mid P_{tr} \mid} log_2 \frac{\mid P_{tr}^d \mid}{\mid P_{tr} \mid},$$

gdzie

- R_t – zbiór wartości jakie może przyjmować atrybut t;
- P_{tr} – podzbiór zbioru P zawierający te reguły, dla których atrybut t przyjmuje wartość r;
- C – zbiór wszystkich kategorii;
- P_{tr}^d – podzbiór zbioru P_{tr}, zawierający reguły, których cześć decyzyjna jest kategorii d.



Rysunek 13.1: Drzewo decyzyjne odpowiadające bazie reguł.

Jako obowiązujący dla danego węzła test wybieramy atrybut o najmniejszej entropii. Mówimy tutaj o teście, gdyż ze względu na wartość wybranego atrybutu będziemy podejmować decycję związaną z wyborem kolejnego węzła. Zanim policzymy entropię dla powyższych atrybutów zauważmy jeszcze tylko, że

- rozpatrujemy cały zbiór reguł (a więc wszystkie reguły od 1 do 15 z tablicy 13.1),

- zbiór testów, zawierający wszystkie te atrybuty, na rzecz których może odbyć się testowanie reguł składa się z atrybutów *natura*, *uroda*, *charyzma* i *inteligencja*,

- domyślna część decyzyjna przyjmuje wartość *obojętność* gdyż, to ona zwracana jest jako rezultat większości reguł ze zbioru.

Lp.	Uroda	Charyzma	Inteligencja	Stosunek
1	piękna	duża	niska	uwielbienie
2	piękna	duża	wysoka	uwielbienie
5	pospolita	duża	niska	uwielbienie
10	brzydka	mała	niska	obojętność
13	pospolita	mała	wysoka	obojętność

Tabela 13.2: Przykładowa baza reguł dla o atrybutu *natura=dobra*.

$$E_{natura}(P) = \frac{\mid P_{natura,\ dobra} \mid}{\mid P \mid} \left(-\frac{\mid P^{uwielbienie}_{natura,\ dobra} \mid}{\mid P_{natura,\ dobra} \mid} log_2 \frac{\mid P^{uwielbienie}_{natura,\ dobra} \mid}{\mid P_{natura,\ dobra} \mid} + \right.$$
$$- \frac{\mid P^{obojtno}_{natura,\ dobra} \mid}{\mid P_{natura,\ dobra} \mid} log_2 \frac{\mid P^{obojtno}_{natura,\ dobra} \mid}{\mid P_{natura,\ dobra} \mid} - \frac{\mid P^{nienawi}_{natura,\ dobra} \mid}{\mid P_{natura,\ dobra} \mid} log_2 \frac{\mid P^{nienawi}_{natura,\ dobra} \mid}{\mid P_{natura,\ dobra} \mid} \Bigg) +$$
$$+ \frac{\mid P_{natura,\ neutralna} \mid}{\mid P \mid} \left(-\frac{\mid P^{uwielbienie}_{natura,\ neutralna} \mid}{\mid P_{natura,\ neutralna} \mid} log_2 \frac{\mid P^{uwielbienie}_{natura,\ neutralna} \mid}{\mid P_{natura,\ neutralna} \mid} + \right.$$
$$- \frac{\mid P^{obojtno}_{natura,\ neutralna} \mid}{\mid P_{natura,\ neutralna} \mid} log_2 \frac{\mid P^{obojtno}_{natura,\ neutralna} \mid}{\mid P_{natura,\ neutralna} \mid} - \frac{\mid P^{nienawi}_{natura,\ neutralna} \mid}{\mid P_{natura,\ neutralna} \mid} log_2 \frac{\mid P^{nienawi}_{natura,\ neutralna} \mid}{\mid P_{natura,\ neutralna} \mid} \Bigg) +$$
$$+ \frac{\mid P_{natura,\ za} \mid}{\mid P \mid} \left(-\frac{\mid P^{uwielbienie}_{natura,\ za} \mid}{\mid P_{natura,\ za} \mid} log_2 \frac{\mid P^{uwielbienie}_{natura,\ za} \mid}{\mid P_{natura,\ za} \mid} + \right.$$
$$- \frac{\mid P^{obojtno}_{natura,\ za} \mid}{\mid P_{natura,\ za} \mid} log_2 \frac{\mid P^{obojtno}_{natura,\ za} \mid}{\mid P_{natura,\ za} \mid} - \frac{\mid P^{nienawi}_{natura,\ za} \mid}{\mid P_{natura,\ za} \mid} log_2 \frac{\mid P^{nienawi}_{natura,\ za} \mid}{\mid P_{natura,\ za} \mid} \Bigg) =$$
$$= \frac{5}{15} \cdot \left(-\frac{3}{5} log_2 \frac{3}{5} - \frac{2}{5} log_2 \frac{2}{5} - \frac{0}{5} log_2 \frac{0}{5} \right) + \frac{5}{15} \cdot \left(-\frac{0}{5} log_2 \frac{0}{5} - \frac{5}{5} log_2 \frac{5}{5} - \frac{0}{5} log_2 \frac{0}{5} \right) +$$
$$+ \frac{5}{15} \cdot \left(-\frac{0}{5} log_2 \frac{0}{5} - \frac{2}{5} log_2 \frac{2}{5} - \frac{3}{5} log_2 \frac{3}{5} \right) = 0.647$$

Ponieważ atrybutem o najmniejszej wartości entropii jest atrybut *natura*, więc korzeniem budowanego drzewa będzie ten właśnie atrybut (on będzie testem w pierwszym węźle). Krawędziami poprowadzonymi z tego węzła będą gałęzie: *dobra*, *neutralna* i *zła* – czyli wszystkie wartości atrybutu *natura*.

Kolejne wywołanie funkcji będzie realizowane ze względu na pomniejszoną bazę reguł. Będzie to baza pomniejszona o atrybut *natura* oraz zawierająca tylko te wiersze, które odpowiadają danej gałęzi, tzn. jeżeli wywołujemy funkcję w węźle, do którego prowadzi gałąź $natura = dobra$ bierzemy pod uwagę reguły dla atrybutów: uroda, charyzma i inteligencja, ale tylko te, dla których $natura = dobra$ (patrz tabela 13.2).

Ze względu na kolejne wywoływania funkcji przy realizacji algorytmu, możemy stwierdzić, że jest to algorytm rekurencyjny. Należy zatem określić warunek zatrzymania działania tego algorytmu. Będzie nim wyczerpanie listy atrybutów lub podanie jako argument takiej bazy reguł, dla której atrybut wyjściowy (*stosunek*) przyjmuje takie samą wartość.

13.1.1 Zadanie

Na podstawie podanej bazy reguł zbudować drzewo decyzyjne w oparciu o przedstawiony algorytm.

13.2 Dodatek - entropia

Dla dociekliwych 13.1. Entropia

Entropia w ramach teorii informacji jest definiowana jako średnia ilość informacji, przypadająca na znak symbolizujący zajście zdarzenia z pewnego zbioru. Zdarzenia w tym zbiorze mają przypisane prawdopodobieństwa wystąpienia.

Wzór na entropię:

dd

gdzie p(i) - prawdopodobieństwo zajścia zdarzenia i. W przypadku kodowania ciągu znaków jest to prawdopodobieństwo wystąpienia i-tego znaku. W teorii informacji najczęściej stosuje się logarytm o podstawie r=2, wówczas jednostką entropii jest bit. Dla r=e jednostka ta nazywa się nat(nit), natomiast dla r=10 - dit lub hartley.

Entropię można interpretować jako niepewność wystąpienia danego zdarzenia elementarnego w następnej chwili. Jeżeli następujące zdarzenie występuje z prawdopodobieństwem równym 1, to z prostego podstawienia wynika, że entropia wynosi 0, gdyż z góry wiadomo co się stanie - nie ma niepewności.

Własności entropii:

** jest nieujemna * jest maksymalna, gdy prawdopodobieństwa zajść zdarzeń są takie same * jest równa 0, gdy stany systemu przyjmują wartości 0 albo 1 * własność superpozycji - gdy dwa systemy są niezależne to entropia sumy systemów równa się sumie entropii.*

Definicja informacyjna była pierwotnie próbą ujęcia tradycyjnego pojęcia entropii znanego z termodynamiki w kategoriach teorii informacji. Okazała się jednak, że definicja ta jest przydatna w ramach samej teorii informacji.

Pojęcie entropii jest bardzo przydatne w np: dziedzinie kompresji informacji. Entropię zerowego rzędu można obliczyć znając histogram ciągu symboli. Jest to iloczyn entropii i ilości znaków w ciągu. Osiągi kodowania Huffmana są często zbliżone do tej granicy, ale istnieją lepsze sposoby np. kodowanie arytmetyczne.

Przyjęcie modelu, w którym uwzględnia się kontekst znaku, pozwala zwykle na bardzo duże obniżenie entropii.

Rozdział 14

Programowanie dynamiczne

14.1 Istota programowania dynamicznego, czyli kiedy programowanie nie jest programowaniem

Występujące w tytule słowo *programowanie* jest trochę mylące, gdyż nie do końca mówić będziemy o programowaniu. Programowanie (w połączeniu z komputerem) jest tutaj raczej narzędziemy, które pozwala efektywnie wykorzystać pewną metodologię rozwiązywania zagadnień optymalizacyjnych. Ponieważ metoda programowania dynamicznego może być stosowana do zagadnień optymalizacyjnych o różnorodnej postaci matematycznej, więc należy ją traktować jako pewien sposób podejścia do problemu a nie gotowy algorytm[1].

Najogólniej mówiąc, programowanie dynamiczne zajmuje się optymalnym planowaniem sterowalnych procesów wieloetapowych. Efektem zastosowania tej metody jest ustalenie optymalnego planu działania (optymalnej strategii). Strategię rozumiemy w tym miejscu intuicyjnie jako plan działania, przebieg (kontrolowanie i sterowanie) danego procesu.

Mówiąc o programowaniu dynamicznym rozważamy etapy (stadia) pewnego procesu. Proces decyzyjny może być wieloetapowy ze swojej natury lub dać się (czasem w sposób sztuczny) na pewne etapy rozłożyć. Każdy etap jest charakteryzowany przez pewien stan rozpatrywanego procesu. Pojęcie etapu także rozumiemy tutaj intuicyjnie jako „moment" w którym możemy opisać proces i opis ten ma sens, tzn. jest istotny z punktu widzenia rozwiązywanego problemu. Jako pojedynczy etap można np. przyjąć pewien odcinek czasu (moment gdy ten odcinek upłynie). Charakteryzujemy (opisujemy) wówczas proces po upływie zadanego czasu. Stadium procesu może oznaczać także pewien etap w myśleniu o problemie, przy czym sam proces techniczno-ekonomiczny w ogóle nie musi rozwijać się w czasie. Przejście do następnego stadium może oznaczać np. uruchomienie kolejnego urządzenia czy rozważenie następnej inwestycji.

Programowanie dynamiczne opiera się na podziale rozwiązywanego problemu na podproblemy względem kilku parametrów. Każdy z podproblemów stanowi cząstkę rozwiązania problemu wyjściowego w tym sensie, że znajomość rozwiązań podproblemów pozwala znaleźć rozwiązanie problemu zasadniczego. Klucz do zaprojektowania algorytmu tą techniką leży w znalezieniu równania rekurencyjnego opisującego optymalną wartość funkcji celu dla danego problemu jako funkcji optymalnych wartości funkcji celu dla podproblemów o mniejszych rozmiarach. Programowanie dynamiczne znajduje optymalną

[1] Choć zastosowanie tej techniki prowadzi zwykle do znalezienia algorytmu dla konkretnego zadania.

wartość funkcji celu dla całego zagadnienia rozwiązując podproblemy od najmniejszego do największego w ten sposób, że rozwiązania podproblemów małych pozwalają (łatwo) znaleźć rozwiązania problemów większych. Jeśli teraz do przechowywania wyników pośrednich (tzn. rozwiązań mniejszych problemów) użyjemy tablicy, to pozwala to zastąpić wywołania rekurencyjne odwołaniami do odpowiednich komórek owej tablicy co gwarantuje, że każdy podproblem jest rozwiązywany tylko raz. Rozwiązanie ostatniego z rozpatrywanych podproblemów jest na ogół wartością rozwiązania rozpatrywanego zagadnienia.

W odróżnieniu od techniki **dziel i zwyciężaj** podproblemy w programowaniu dynamicznym nie są na ogół rozłączne, ale musi je cechować własność optymalnej podstruktury (ang. *optimal substructure*). Drugą istotną cechą jest (ang. *overlapping subproblems*).

14.1.1 Optymalna podstruktura

Własnośc optymalnej podstruktury jest niejako istotą programowania dynamicznego. Własność ta mówi, że rozwiązanie problemu optymalizacyjnego może zostać znalezione w oparciu o rozwiązania podproproblemów. Z taką sytuacją mamy do czynienia np. rozpatrując zagadnienie najkrótszej ścieżki w grafie. Jeśli pewna ścieżka w grafie od wierzchołka v_1 do v_n jest optymalną ścieżką łączących te dwa wierzchołki, wówczas ścieżka od dowolnego wierzchołka v_i, $i = 2, \ldots, n-1$ leżącego na tej ścieżce do v_n także musi być optymalna.

Inaczej mówiąc, optymalna podstruktura oznacza że, wartość uzyskana na n-tym etapie optymalizacji zależy od stanu na etapie poprzednim oraz od decyzji podjętej na n-tym etapie. Nie powinna natomiast zależeć od tego, jaką drogą system doszedł do stanu $n-1$. Jest to tzw. **własność Markowa**. Do procesu decyzyjnego posiadającego własność Markowa stosuje się **zasadę optymalności** R. Bellmana (1956), która stanowi podstawową własność strategii optymalnej

Definicja 14.1. *Polityka optymalna ma tę własność, że niezależnie od początkowego stanu i początkowej decyzji pozostałe decyzje muszą stanowić politykę optymalną ze względu na stan wynikający z pierwszej decyzji.*

Konsekwencją tej definicji jest to, że aby ciąg decyzji $d_1, d_2, \ldots, d_n$ był strategią optymalną w procesie o n etapach przy pewnym stanie początkowym s_0 potrzeba, aby ciąg decyzji $d_2, \ldots, d_n$ był strategią optymalną w procesie o $n-1$ etapach przy stanie początkowym wynikającym z decyzji d_1.

Innym przykładem zagadnienia posiadającego optymalną podstrukturę jest poszukiwanie podtablicy o największej sumie elementów[2]. Jeśli tablicą jest $[1, 2, -5, 4, 7, -2]$ wówczas taką podtablicą jest $[4, 7]$. Jeśli zagadnienie to posiada cechę optymalnej podstruktury, wówczas wiedząc, że pewna podtablica o długości i jest taką podtablicą, że suma jej elementów jest największa spośród wszystkich podtablic i elementowych, wykorzystując ją powinniśmy móc znaleźć podtablicę $i+1$ elementową o tej samej cesze.

//tutu dokonczyc//

[2]Podtablicę rozumiemy tutaj jako ciągły „wycinek” tablicy.

14.1.2 Overlapping subproblems

Termin *overlapping subproblems* oznacza, że przestrzeń jaką tworzą podproblemy musi być „mała". Mówiąc inaczej, rekurencyjny algorytm rozwiązania takiego problemu powinien charakteryzować się tym, że w celu rozwiązania kolejnych podproblemów wciąż oblicza te same wartości lub stosunek wartości nowych do uprzednio obliczonych zdecydowanie przemawia na korzyść tych drugich. Najprostszym przykładem jest tutaj ciąg Fibonacciego. Jak wiadomo każdy wyraz (z wyjątkiem dwóch pierwszych) jest w nim sumą dwóch wyrazów poprzednich. W konsekwencji, aby obliczyć np. wyraz 10 musimy znać wyrazy 9 i 8. Zauważmy jednak, że aby obliczyć wyraz 9 musimy znać 8 i 7. Oznacza to, że wyraz 8 będzie obliczony dwukrotnie: pierwszy raz aby możliwe było otrzymanie wyrazu 9 i drugi raz aby obliczyć przy jego pomocy wyraz 10.

Rekurencja jest w tym przypadku „wskaźnikiem", ale nie wszystkie problemy rekurencyjne tą cechę posiadają. Na przykład silnia obliczana jest według rekurencyjnej zależności

$$n! = n \cdot (n-1)!$$

Tak więc aby obliczyć n-ty wyraz musimy znać wyraz $n-1$. Zauważmy, że dla dowolnego n musimy „poznać" wszystkie wyrazy od 1 do $n-1$, ale każdy potrzebny jest tylko jeden raz. Gdybyśmy chcieli to jakoś zobrazować, wówczas drzewo wywołań będzie wyglądać jak na przykładzie

```
s_r(5)
    |
    5*s_r(4)
          |
          4*s_r(3)
                |
                3*s_r(2)
                      |
                      2*s_r(1)
                            |
                            1*s_r(0)
```

W przypadku ciągu Fibonacciego najbardziej narzucające się rozwiązanie postaci

```
long fib_r(int n)
{
if(n==0)
return 0;
else if(n==1)
return 1;
else
return fib_r(n-1)+fib_r(n-2);
}
```

okazuje się najwolniejszym przez wielokrotne wyliczanie tych samych wartości, co można pokazać na następującym drzewie wywołań

```
f_r(5)
    |
```

```
    f_r(4)+f_r(3)
        |      |
        |      f_r(2)+f_r(1)
        |          |
        |          f_r(1)+f_r(0)
        |
        f_r(3)+f_r(2)
            |      |
            |      f_r(1)+f_r(0)
            |
            f_r(2)+f_r(1)
                |
                f_r(1)+f_r(0)
```

Oczywiście można podać algorytm iteracyjny obliczania wyrazów ciągu Fibonacciego, np.

```
long fib_i(int n)
{
long i,n1,n2,tmp;

i=0;
n1=0;
n2=1;

while (i<n)
{
tmp=n2;
n2=n1+n2;
n1=tmp;

i=i+1;
}

return n1;
}
```

Chcąc jednak pozostać przy idei rekurencji można wprowadzić niewielką modyfikację, która znacznie przyspieszy działanie programu. Utwórzmy tablicę pomocniczą, w której zapisywać będziemy kolejne wartości ciągu. Zanim zaczniemy liczyć jakąś wartość, sprawdzmy najpierw w tablicy, czy nie została już ona przypadkiem policzona przez jedno z wywołań rekurencyjnych. Jeśli nie to musimy policzyć i wynik zapisać w tablicy, a jeśli tak to nic nie musimy liczyć i wystarczy pobrać wartość z tablicy. Taki właśnie algorytm jest przedstawiony poniżej

```
long tab[100];

for(i=0;i<100;i++)
tab[i]=-1;
tab[0]=0;
```

```
tab[1]=1;

long fib_r_i(int n)
{
if(tab[n]!=-1)
return tab[n];
else
tab[n]=fib_r_i(n-1)+fib_r_i(n-2);

return tab[n];
}
```

14.2 Przykłady zastosowania

W poprzednim podrozdziale podaliśmy elementarne przykłady związane z programowaniem dynamicznym. Tutaj spróbujemy rozwiązać bardziej złożone (i posiadające praktyczne zastosowanie) zagadnienia.

14.2.1 Dopasowywanie ciągów – algorytm Needlemana-Wunscha

Algorytm ten może być wykorzystywany do dopasowywania dowolnych ciągów znaków, w szczególności sekwencji DNA, które opisywane są znakami $\{A, C, G, T\}$. Polski termin dopasowywanie będący odpowiednikiem angielskiego *alignment* często tłumaczy się też jako uliniowianie sekwencji.

W bioinformatyce, dopasowanie sekwencji jest sposobem dopasowania struktur pierwszorzędowych DNA, RNA, lub białek do zidentyfikowania regionów wykazujących podobieństwo, mogące być konsekwencją funkcjonalnych, strukturalnych, lub ewolucyjnych powiązań pomiędzy sekwencjami. Zestawione sekwencje nukleotydów lub aminokwasów są zazwyczaj przedstawione jako wiersze macierzy. Pomiędzy reszty wprowadzane są przerwy, tak że reszty zbliżonych do siebie sekwencji tworzą kolejne kolumny. Jeśli dwie dopasowywane sekwencje mają wspólne pochodzenie, niedopasowania mogą być interpretowane jako mutacje punktowe, a przerwy jako indele (mutacje polegające na delecji lub insercji), które zaszły w jednej lub obu liniach od czasu, kiedy obie sekwencje uległy rozdzieleniu. [3]

Dla przykładu, mając dwie sekwencje

```
            0 0 0 0 0 0 0 0 0 1 1 1 1
            1 2 3 4 5 6 7 8 9 0 1 2 3
sekwencja 1: A G A T T G A T A C C C A
sekwencja 2: A G A C A T T A A C T A
```

można je dopasować w następujący sposób

```
            0 0 0 0 0 0 0 0 0 1 1 1 1 1 1
            1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
sekwencja 1: A G A - - T T G A T A C C C A
```

[3] `http://wapedia.mobi/pl/Uliniowianie`, stan z 2 maja 2009.

```
sekwencja 2: A G A C A T T A A - - - C T A
             |               |       |
             match           gap     mismatch
```

przy czym elementy występujące np. w sekwencji 2 na pozycjach 10 czy 12 nazywamy przerwami[4] (ang. *gap*), elementy występujące np. na pozycji 7 w obu sekwencjach nazwiemy pasującymi (ang. *match*) a elementy z pozycji np. 14 nazwiemy niepasującymi (ang. *mismatch*).

Aby móc dopasować dwie sekwencje do siebie należy wprowadzić jakąś miarę dopasowania aby móc oceniać na ile są one do siebie podobne. Najprostszy sposób to przyjęcie, że jesli dwa elementy na tej samej pozycji w dwóch różnych ciagach są identyczne to oceniamy tę pozycję na $+1$, jeśli różne to na -1 a w przypadku wystąpienia przerwy przyjmujemy wartość 0. Dla przykładu sekwencje

```
A T G G C G T
A T G - A G T
```

ocenone zostaną na

$$1+1+1+0-1+1+1=4$$

natomiast

```
A T G G C G T
A - T G A G T
```

na

$$1+0-1+1-1+1+1=2.$$

W zwarty sposób zaproponowaną „punktację" można przedstawić za pomocą macierzy podstawień[5] (ang. *substitution matrix*), np.

```
   C  T  A  G
C  1 -1 -1 -1
T -1  1 -1 -1
A -1 -1  1 -1
G -1 -1 -1  1
```

Oczywiście jest to jedna z wielu możliwości. W przypadku gdy wiemy (zakładamy), że pewne odstępstwa są mniej lub bardziej spotykane można wprowadzić rozróżnianie ich kosztów, np.

```
   C  T  A  G
C  2  1 -1 -1
T  1  2 -1 -1
A -1 -1  2  1
G -1 -1  1  2
```

Do tej pory pominęliśmy najistotniejsze chyba zagadnienie a mianowicie wstawianie pustych elemntów (gap). To one tworzą przesunięcia i to dzięki nim można najlepiej dopasować dwie sekwencje. Za wystąpienie elementu typu gap przyjmowaliśmy 0 punktów.

[4] Mówi się także o przesunięciu (ang. alignment).

[5] Czasem też nazywanej macierzą podobieństwa (ang. *similarity matrix*).

Pamiętajmy jednak, że gap to wprowadzenie pewnej zmiany z sekwencji, zmiany o tyle istotnej, że zmieniamy jej długość. Tak więc za każdy element typu gap powinna być przyznana kara, czyli punkty ujemne. Nawet jednak z tą modyfikacją nadal będziemy mieć problem z dopasowaniem sekwencji. Punktacja pozwala ocenić nam dwie sekwencje, ale nie daje żadnych wskazówek gdzie należy wstawić elementy gap. Rozwiązania siłowe ze względu na bardzo długi czas obliczeń nie wchodzą tutaj w grę. I właśnie tutaj z pomocą przychodzi algorytm Needlemana-Wunscha, który stosując zasadę programowania dynamicznego buduje optymalne przesunięcia w sekwencjach w oparciu o optymalne przesunięcia w podsekwencjach.

W algorytmie rozważamy wszystkie możliwe pary elementów obu sekwencji co prowadzi do powstania dwóch dwuwymiarowych macierzy. Jedna z macierzy zawiera punkty druga natomiast służyć będzie do wyznaczenia miejsc wstawienia elementów typu gap. Na algorytm składają się trzy zasadnicze etapy

1. Inicjalizacja macierzy (ang. *initialisation*).
2. Wypełnienie macierzy (ang. *scoring*).
3. Wyznaczenie przesunięć (ang. *traceback*).

Wyjaśnimy działanie tego algorytmu na przykładzie dwóch sekwencji

```
sekwencja 1: SEND
sekwencja 2: AND
```

W tym przypadku łatwo będzie nam sprawdzić czy opisywany algoytm daje optymalne rozwiązanie, gdyż możemy wypisać wszystkie cztery możliwe sposoby dopasowania

```
SEND
-AND  =>  +1
A-ND  =>  +3 (najlepsze)
AN-D  =>  -3
AND-  =>  -8
```

przy czym przyjmujemy następującą macierz kosztu

```
   A  D  E  N  S
A  4 -2 -1 -2  1
D -2  6  2  1  0
E -1  2  5  0  0
N -2  1  0  6  1
S  1  0  0  1  4
```

i karę za element *gap* wynoszącą -10.

Inicjalizacja macierzy

Jeśli m oznacza długość pierwszej sekwencji zaś n drugiej, wówczas należy utworzyć dwie pustą macierze C i T o wymiarze $(n+1) \times (m+1)$. Kolejne wiersze (kolumny) odpowiadają kolejnym elementom sekwencji. Pierwszy wiersz pierwszej macierzy (C) i pierwsza jej kolumna wypełniona zostaje wartościami, które mają być karą za wstawienie elementu typu gap. Przyjmując, że kara za jeden element gap wynosi -10 otrzymamy następującą macierz

```
        S   E   N   D
    0 -10 -20 -30 -40
A -10   .   .   .   .
N -20   .   .   .   .
D -30   .   .   .   .
```

Drugą macierz (T) wypełniamy w następujący sposób

```
         S    E    N    D
  end left left left left
A  up    .    .    .    .
N  up    .    .    .    .
D  up    .    .    .    .
```

Wypełnienie macierzy

Elementy macierzy wypełniamy począwszy od elementu $c_{2,2}$. Wartość każdego elementu jest równa

$$c_{i,j} = \max\{v_{diag}, v_{left}, v_{up}\},$$

gdzie

$$v_{diag} = c_{i-1,j-1} + s(i,j)$$
$$v_{left} = c_{i,j-1} + g$$
$$v_{up} = c_{i-1,j} + g$$

wyrażenie g to koszt wstawienia elementu gap (przyjmujemy ten koszt równy -10), natomiast $s(i,j)$ to koszt z macierzy podstawień jaki ponieść musimy zamieniając i-ty znak jednej sekwencji na j-ty drugiej. Jak więc widać wartość elemntu $c_{i,j}$ zależy tylko od trzech elementów które z nim sąsiadują (od góry, z lewej i od lewej górnej przekątnej).

Policzymy teraz wartość dla elementu $c_{2,2}$

$$v_{diag} = c_{1,1} + s(S,A) = 0 + 1 = 1$$
$$v_{left} = c_{2,1} + g = -10 - 10 = -20$$
$$v_{up} = c_{1,2} + g = -10 - 10 = -20$$

A zatem element $c_{2,2}$ macierzy C przyjmie wartość 1 a odpowiadający element macierzy T ($t_{2,2}$) przyjmie wartość *diag*

```
        S   E   N   D              S    E    N    D
    0 -10 -20 -30 -40       end left left left left
A -10   1   .   .   .     A  up diag    .    .    .
N -20   .   .   .   .     N  up    .    .    .    .
D -30   .   .   .   .     D  up    .    .    .    .
```

Policzymy teraz wartość dla elementu $c_{2,3}$

$$v_{diag} = c_{1,2} + s(E,A) = -10 - 1 = -11$$
$$v_{left} = c_{2,2} + g = 1 - 10 = -9$$
$$v_{up} = c_{1,3} + g = -20 - 10 = -30$$

A zatem element $c_{2,2}$ macierzy C przyjmie wartość -9 a odpowiadający element macierzy T ($t_{2,2}$) przyjmie wartość $left$

```
        S   E   N   D              S    E    N    D
    0 -10 -20 -30 -40        end left left left left
A -10   1  -9   .   .      A  up diag left    .    .
N -20   .   .   .   .      N  up    .    .    .    .
D -30   .   .   .   .      D  up    .    .    .    .
```

Stosując dalej analogiczne postępowanie na koniec powinniśmy uzyskać następujące macierze

```
        S   E   N   D              S    E    N    D
    0 -10 -20 -30 -40        end left left left left
A -10   1  -9 -19 -29      A  up diag left left left
N -20  -9  -1  -3 -13      N  up diag diag diag left
D -30 -19 -11   2   3      D  up   up diag diag diag
```

Wyznaczenie przesunięć

Wyznaczanie optymalnych przesunięć odbywa się w etapie nazywanym wyznaczaniem śladu powrotnego lub też ścieżki powrotnej (ang. *traceback path*). Algorytm rozpoczyna swoje działanie z ostatniej komórki macierzy T (tzn. elementu $(n+1, m+1)$). Kolejne elementy ścieżki wybierane są w oparciu o zawartość macierzy T a ich wartość określa ewentualne przesunięcia według poniższych zasad.

- Jeśli w komórce jest wartość *diag* wówczas znaki z obu sekwencji uznajemy za pasujące.
- Jeśli w komórce jest wartość *left* wówczas w sekwencji znajdującej się po lewej stronie macierzy T (tworzącej wiersze) wstawiany jest element *gap*.
- Jeśli w komórce jest wartość *up* wówczas w sekwencji znajdującej się na górze macierzy T (tworzącej kolumny) wstawiany jest element *gap*.

Zobaczmy jak wygląda to w naszym przypadku.

1. Rozpoczynamy z komórki $(4, 5)$. Znajduje się w niej wartość *diag* a więc tworzymy dwie sekwencje rozpoczynające (a właściwie to kończące się, bo sekwencje tworzone są od końca) się od znaków odpowiadających współrzędnym, czyli D i D.

   ```
            S    E    N    D                   D
      end left left left left                   |
   A   up diag left left left                   D
   N   up diag diag diag left
   D   up   up diag diag DIAG
   ```

   Zawartość tej komórki wskazuje nam kolejną komórkę, czyli w tym przypadku element macierzy T o współrzędnych $(3, 4)$.

2. W komórce $(3, 4)$ znajduje się wartość *diag* a więc do istniejących sekwencji dodajemy (z przodu) znaki odpowiadające współrzędnym, czyli N i N.

```
          S    E    N    D                ND
  end left left left left                 ||
A  up diag left left left                 ND
N  up diag diag DIAG left
D  up   up diag diag DIAG
```

Zawartość tej komórki wskazuje nam kolejną komórkę, czyli w tym przypadku element macierzy T o współrzędnych $(2, 3)$.

3. W komórce $(2, 3)$ znajduje się wartość *left* a więc do istniejących sekwencji dodajemy (z przodu) element *gap* (dla „lewej" sekwencji) i element E (dla „górnej" sekwencji).

```
          S    E    N    D               END
  end left left left left                  ||
A  up diag LEFT left left                 -ND
N  up diag diag DIAG left
D  up   up diag diag DIAG
```

Zawartość tej komórki wskazuje nam kolejną komórkę, czyli w tym przypadku element macierzy T o współrzędnych $(2, 2)$.

4. W komórce $(2, 2)$ znajduje się wartość *diag* a więc do istniejących sekwencji dodajemy (z przodu) znaki odpowiadające współrzędnym, czyli S i A.

```
          S    E    N    D              SEND
  end left left left left                | ||
A  up DIAG LEFT left left                A-ND
N  up diag diag DIAG left
D  up   up diag diag DIAG
```

Zawartość tej komórki wskazuje nam kolejną komórkę, czyli w tym przypadku element macierzy T o współrzędnych $(1, 1)$.

5. W komórce $(1, 1)$ znajduje się wartość *end* co oznacza koniec algorytmu. Zatem optymalne dopasowanie łańcuchów, przy zdefiniowanym przez nas koszcie to

```
SEND
| ||
A-ND
```

Rozdział 15

Elementarne wiadomości z biologii

15.1 Mózg jako biologiczne CPU

Naturalnym jest dla nas, że „my" jako ludzie, możemy wykonywać wiele różnych akcji. Często, jeśli nawet nie częściej, pewne akcje odbywają się „w nas" bez udziału naszej świadomości. Możemy, świadomie lub nie:

- widzieć, słyszeć, czuć itp, czyli przyjmować i przetwarzać agromną lawinę informacji powodowaną różnymi zmysłami (wzrok, słuch, smak, zapach, dotyk itp);
- kontrolować naszą ogólnie pojmowaną motorykę gdy chodzimy, rozmawiamy, gestykulujemy;
- myśleć, wnioskować, zapamiętywać, uczyć się;
- marzyć;
- kontrolować ciepłotę ciała, ciśnienie krwi, tętno czy szybkość oddechu.

Wszystkie te zadania są koordynowane i kontrolowane przez jeden, mały rozmiarowo, organ – mózg. Można powiedzieć, że mózg jest głównym elementem przetwarzającym w naszym ciele. To takie biologiczne CPU w biologicznym hardware jakim jest ludzkie ciało.

U zwierząt tutu mózg zwykle jest umiejscowiony w głowie. Chroniony czaszką znajduje się bardzo blisko najważniejszych zródeł informacji: wzroku, słuchu, smaku i zapachu, ośrodka równowagi (jak to inaczej okreslic???!!).

(tutu Cos o zdecentralizowanym ukladzie nerwowym.)

15.2 Malutkie cegiełki – neurony

Mózg zbudowany jest z dwóch zasadniczych rodzajów komórek

- nerwowych, tzw. neuronów (ang. *neurons, neurones, nerve cells*),
- glejowych (ang. *glial cells*; glia z gr. *klej*).

Ilość neuronów u przeciętnego człowieka szacowana jest na ok. 100 miliardów (10^{11}, 1mld = 10^9).

Neurony postrzegane mogą być jako aktywne elektrycznie komórki. Posiadają one bowiem zdolność do gromadzenia i transmitowania sygnałów elektrochemicznych na duże, rzędu kilku metrów, odległości[1].

Można powiedzieć, że neurony są bardzo wąsko wyspecjalizowanymi komórkami, których głównym zadaniem jest przetwarzanie i przekazywanie sygnałów. Mechanizmy dzięki którym się to odbywa są bardzo złożone, ale sam proces, czyli gromadzenie sygnałów i ich przekazanie dalej, wydaje się być prosty. W pewnym sensie są podobne do bramek logicznych tworzących „zwykłe" komputery.

Połączone neurony tworzą sieć neuronową. Sieć taka podobna jest do sieci komputerowej, gdzie neurony (odpowiedniki węzłów, np. komputer) połączone są włóknami nerwowymi (np. przewód UTP). Zauważmy jednak, że stopień złożoności w biologicznych sieciach jest znacznie większy. Niezwykle rzadko mamy tutaj do czynienia z połączeniem jeden-do-jednego. Zwykle jeden neuron połączony jest z co najmniej tysiącem innych[2].

Komunikacja między neuronami odbywa się w procesie transmisji synaptycznej (ang. *synaptic transmission*) za pośrednictwem reakcji elektro-chemicznych. Podstawą tego procesu jest potencjał czynnościowy, „przemieszczający" sygnał elektryczny od ciała komórki wzdłuż aksonu do innych komórek. Zjawisko to znane jest także pod nazwą fali depolaryzacyjnej (ang. *wave of depolarization*).

Komórki glejowe, mniej więcej 10 razy liczniejsze niż neurony, stanowią swego rodzaju „wypełnienie", czy też przestrzeń, w której znajdują się neurony. Zasadniczym ich zadaniem jest ochrona tkanki nerwowej, zaopatrywanie jej w substancje odżywcze, udział w jej procesach metabolicznych i regeneracyjnych.

15.3 Budowa komórki nerwowej

Pomimo dużego zróżnicowania neuronów pod względem kształtu, rozmiaru czy też własności elektrochemicznych, w każdym z nich wyróżnić można pewne wspólne elemeny.

Ciało komórki (soma). (ang. *cell body*) Podobnie jak w przypadku innych komórek, także i w tym przypadku ciało komórki nerwowej zawiera wszystkie niezbędne do funkcjonowania komórki struktury: jądro komórkowe (zawiera informacje genetyczne), rybosomy (odpowiadają za biosyntezę białek), mitochondrium (jako źródło energii). Obumarcie jądra komórki oznacza śmierć samej komórki. Soma zwykle przyjmuje rozmiary z przedziału 4-100 mikrometrów (1 mikrometr = 10^{-6}). Z informatycznego punktu widzenia pełnią rolę układu przetwarzającego dane.

Akson. Przekazuje sygnał elektrochemiczny (impuls nerwowy, potencjał czynnościowy) z somy na zewnątrz komórki[3]. Większość neuronów posiada tylko jeden akson, który jednakże może się znacznie rozgałęziać, przez co sygnał przekazywany jest do znacznej ilości innych komórek (zwykle co najmniej tysiąc). Aksony niektórych neuronów bywają pokryte osłonką mielinową (jak gdyby odpowiednik ziolacji przewodu elektrycznego) przyspieszającą znacznie proces transmisji impulsu wzdłuż aksonu. Długość aksonu jest zwykle od kilku do kilkudzisięciu tysięcy razy większa od średnicy somy (zawiera się w przedziale od ok. 0.01–1m).

[1] Co przy samym rozmiarze ciała komórki wynoszącym ok. 4 do 100 mikrometrów jest odległością bardzo dużą (1 mikrometr = 10^{-6}, a więc od kilku tysięcy do ok. miliona razy większą).

[2] Mamy tutaj na myśli połączenia bezpośrednie.

[3] Szybkość propagacji mieści się w zakresie 1-100m/s a czas trwania impulsu to ok. 1-2ms.

Akson zakończony jest **synapsami** — specjalnymi strukturami, w których uwalniane są chemiczne neurotransmitery w celu przekazania sygnału do innego neuronu (jego dendrytu lub somy a znacznie rzadziej aksonu).

Synapsy mogą być pobudzające lub hamujące (blokujące, spowalniające (ang. *inhibit*)), to znaczy zwiększające lub zmniejszające aktywność (potencjał) jakiegoś neuronu. Transmisja sygnału między komórkami, za pośrednictwem synaps, jest złożonym procesem chemicznym, w którym są uwalniane specyficzne substancje, tzw. transmitery po stronie nadawczej złącza. W rezultacie rośnie, lub maleje, potencjał elektryczny ciała komórki odbiorczej. Jeżeli ten potencjał osiągnie wartość progową, to w kierunku aksonu jest wysyłany impulsowy potencjał czynnościowy o stałej wartości i czasie trwania (mówimy wówczas, że nastąpił zapłon komórki). Impuls rozchodzi się po drzewku wyjściowym aksonu i poprzes złącze synaptyczne dociera do innych komórek. Po zapłonie komórka musi odczekać pewien czas, zwany okresem refrakcji, zanim będzie zdolna do ponownego zapłonu.

W znacznym skrócie proces transmisji synaptycznej można przedstawić w następujących etapach. (tutu etapy)

Z informatycznego punktu widzenia pełnią rolę układu wyjściowego.

Przeciętnie każdy neuron ma około 7000 połączeń synaptycznych z innymi neuronami. Szacuje się, że dorosła osoba ma łacznie od 10^{15} do 5×10^{15} takich połączeń, podczas gdy trzyletnie dziecko ma ich ok. 10^{16}.

Dendryty. Stanowią niejako mocno rozgałęziającą się „dobudowę" ciała komórki. Są zasadniczym miejscem odbierania sygnałów. Z informatycznego punktu widzenia pełnią rolę układu wejściowego. Ilość dendrytów waha się od kilku do kilku tysięcy. Jednak każdy dendryt może otrzymywać sygnały od tysięcy synaps.

Zasada „wszystko albo nic"

Przekazywanie impulsu nerwowego odbywa się według zasady wszystko, albo nic (ang. *all-or-none principle*). Oznacza to, że jeśli neuron zareaguje na pobudzenie to odpowiada na to pobudzenie w pełni. Istotne jest tutaj to, że mocniejszemu pobudzeniu odpowiada nie silniejszy sygnał (silniejsza reakcja), ale więcej sygnałów w jednostce czasu.

15.4 Przydatne klasyfikacje

Komórki nerwowe można dzielić i klasyfikować ze względu na różne kryteria. Poniżej przedstawiam tylko te, które choć w małym stopniu mogą być zrozumiałe dla kogoś ukierunkowanego na nauki ścisłe.

Klasyfikacja funkcjonalna – kierunek przekazywania informacji.

Afektory. (ang. afferent neurons) Przekazują informacje z tkanek i organów do centralnego układu nerwowego (sensory, receptory). W ogólności są to te komórki nerwowe, które dostarczają informacji do mózgu.

Efektory. (ang. *efferent neurons* lub *motor neurons*) Przekazują informacje z centralnego układu nerwowego do układów wykonawczych (efektorów). W ogólności są to te komórki nerwowe, które wyprowadzają informacje z mózgu.

Międzyneuronowe. (and. *interneurons connect neurons*) Przekazują informację pomiędzy różnymi obszarami centralnego układu nerwowego.

Klasyfikacja funkcjonalna – sposób oddziaływania na inne komórki.

Neurony pobudzające. (ang. *excitatory neurons*) Powodują pobudzenie (zwiększenie potencjału) kolejnych neuronów.

Neurony hamujące. (ang. *inhibitory neurons*) Hamują (spowalniają) kolejne neurony.

Neurony modelujące (ang. *modulatory neurons*). (tutu)

Charakterystyka elektrofizjologiczna – sposób wyładowania (ang. *discharge patterns*)

Tonic or regular spiking. Some neurons are typically constantly (or tonically) active.

Phasic or bursting. Neurons that fire in bursts are called phasic.

Fast spiking. Neurony o dużej częstotliwości reakcji („odpaleń", ang. *fast firing rates*).

Thin-spike. Action potentials of some neurons are more narrow compared to the others.

15.5 Mózg a rozum

Rozróżnienie pomiędzy rozumem a mózgiem jest czymś zasadniczym w filozofii mysłu. Problem umysł-ciało od wieków zajmował filozofów. Mózg jest tworem biologicznym mieszczącym się wewnątrz czaszki, odpowiedzialnym za szereg procesów elektrochemicznych. Rozum natomiast jest czymś niematerialnym, duchowym: myślami, pragnieniami, sposobem postrzegania rzeczywistości czy świadomością. Można pokazać pewne korelacje zachodzące pomiędzy zdarzeniami mentalnymi a sygnałami neuronów. Pytaniem pozostaje natomiast, jaka faktycznie łączy je relacja – czy rozum i mózg to jest to samo czy może jednak nie. Filozoficzne rozważania na ten temat chilowo jednak odsuwamy.

Rozdział 16

Sztuczna sieć neuronowa

Tradycyjnie termin sieć neuronowa odnosił się do struktury połączonych neuronów biologicznych. Obecnie oznacza on często także sztuczne sieci neuronowe zbudowane ze sztucznych neuronów, czyli „tworów" naśladujacych działanie biologicznego neuronu[1]. Sztuczne sieci neuronowe mogą być używane zarówno w celu lepszego i głębszego poznania zasad działania ich biologicznych protoplastów, jak i w celu rozwiazywania problemów z zakresu sztucznej inteligencji.

Sztuczna inteligencja i modelowanie kognitywne / poznawcze (ang. *cognitive modeling*) próbują naśladować / symulować pewne własności sieci neuronowych. Choć oba używają podobnych narzędzi i technik, to pierwszy z nich skupia się na rozwiązywaniu konkretnych zadań, drugi natomiast na budowaniu matematycznego modelu biologicznego systemu nerwowego.

Na polu zastosowań sztuczne sieci neuronowe z powodzeniem wykorzystano w takich zadanich jak:

- klasyfikacja i rozpoznawanie wzorców – przy klasyfikacji i rozpoznawaniu wzorców sieć uczy się podstawowych cech tych wzorców, takich jak odwzorowanie geometryczne układu pikselowego wzorca, rokład składników głównych wzorca czy innych jego właściwości; podkreśla się różnice występujące w różnych wzorcach, stanowiące podstawę podjęcia odpowiedniej decyzji klasyfikacyjnej;

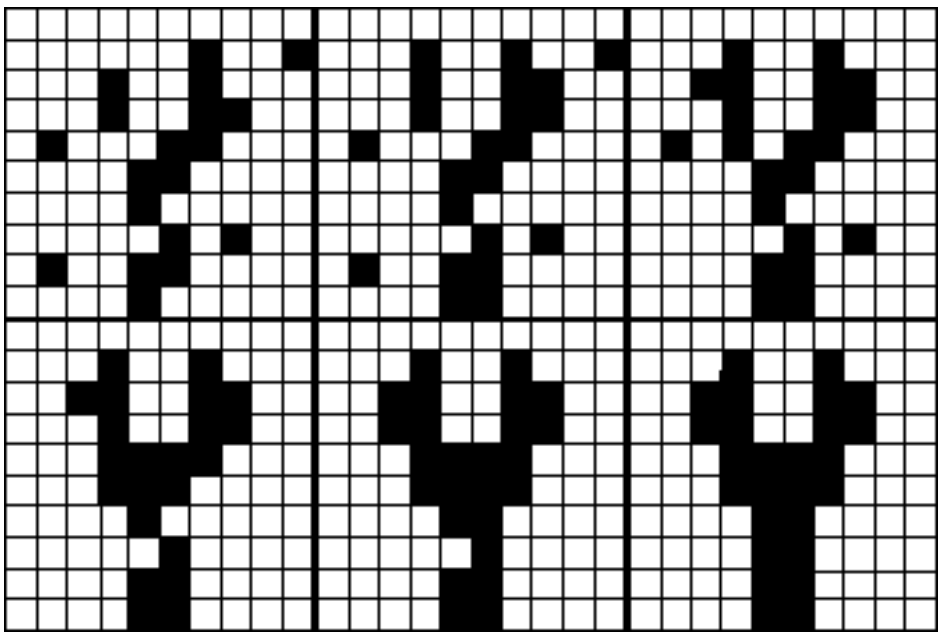
- aproksymacja funkcji, modelowanie – sieć aproksymująca odgrywa rolę uniwersalnego aproksymatora funkcji wielu zmiennych;

- asocjacji – gdzie sieć neuronowa spełnia zadanie pamięci asocjacyjnej; można tu wyróżnić pamięci:

  - autoasocjacyjną, gdzie skojarzenie dotyczy tylko poszczególnych składowych wektora wejściowego
  - heteroasocjacyjną, gdzie zadaniem sieci jest skojarzenie ze sobą dwu wektorów

  W przypadku podania na wejście sieci wektora zaszumionego bądź pozbawionego pewnych fragmentów danych, sieć potrafi odtworzyć pełny wektor oryginalny, pozbawiony szumów, generując przy tym pełną postać drugiego, odpowiadającego mu, wektora (patrz także rys. 16.1).

[1] W kolejnych rozdziałach często zamiast pojęcia „sztuczna sieć neuronowa" będzie używane pojęcie „sieci neuronowej'.



Rysunek 16.1: Przykład odtworzenia oryginlanego obrazu (prawy dolny obraz) na podstawie zaszumionych danych wejściowych (lewy górny obraz).

- optymalizacja;
- predykcja wartości sygnałów – sieć ma za zadanie określenie przyszłych zachowań systemu na podstawie ciągu wartości z przeszłości;
- filtrowanie danych;
- grupowanie (klastrowanie) i wydobywanie cech;
- sterowania – w tym przypadku sieć neuronowa pełni zwykle kilka funkcji:
  - stanowi model nieliniowy tego procesu, identyfikując jego podstawowe parametry niezbędne do wypracowania odpowiedniego sygnału sterującego;
  - pełni funkcje układu śledzącego adaptując się do zmiennych warunków środowiska;
  - może również stanowić bezpośredni neuroregulator, zastępując klasyczne rozwiązania;
  - ważną rolę, zwłaszcza w sterowaniu robotów, odgrywa klasyfikacja i podejmowanie decyzji co do dalszego przebiegu procesu

Praktyczne zastosowania obejmują zadania kontroli i identyfikacji (sterowanie pojazdami, procesami technologicznymi), sterowania robotami czy agentami w grach, rozpoznawania wzorców (sygnały radarowe, identyfikacja twarzy, wykrywanie obiektów), rozpoznawanie wzorców-sekwencji (gesty, mowa, tekst pisany), diagnostyki medycznej, analizy finansowej, *data mining* (KDD, *knowledge discovery in databases*), wizualizacji, filtrowania (np. spamu).

Najistotniejszą cechą sztucznych sieci neuronowych, z punktu widzenia zastosowań, jest możliwość ich tworzenia w oparciu o obserwacje. Oznacza to, że w sytuacji gdy dane są niepełne, lub na tyle złożone, że związków pomiędzy nimi nie znamy lub są one bardzo skomplikowane, możemy, jedynie w oparciu o przykłady typu „w tej sytuacji należy wykonać to", zbudować poprawnie reagujący układ. Mówimy, że sztuczne sieci neuronowe **możemy uczyć**.

Ponadto cechuje je zdolność do **uogólniania** nabytej wiedzy. Duża ilość elementów przetwarzających powoduje, że jest ona do pewnego stopnia **odporna na uszkodzenia** jej struktury.

Istotną zaletą sieci jest możliwość ich realizacji w technice o wielkim stopniu scalenia. Zróżnicowanie elementów sieci jest niewielkie, a ich powtarzalność ogromna. Stwarza to perspektywę zbudowania uniwersalnego procesora, który, być może w przyszłości wspomoże procesory klasyczne[2].

Wciąż jednak nie znamy odpowiedzi na pytanie o stopień złożoności i własności jakie pojedyńczy neuron powinien posiadać aby przy jego pomocy móc odwzorować coś co by przypominało inteligentne zachowania. Co więcej, nie wiemy nawet czy ścisłe „podążanie" za biologią ma sens – jak wiemy samoloty latają a jednak w istotny sposób inaczej niż ptaki.

Sztuczna sieć neuronowa (ang. *artificial neural network (ANN)*, *simulated neural network (SNN)*, *neural network (NN)*) jest połączeniem pewnej ilości sztucznych neuronów wykorzystujących pewien matematyczny model obliczeniowy w celu przetwarzania informacji. Sieci neuronowe wykorzystywane w sztucznej inteligencji są znacznie uproszczonym modelem analogicznych struktur z mózgu. Współczesne komputery wyewoluowały w oparciu o założenia architektury von Neumannowskiej i jako takie bazują na **sekwencyjnym** przetwarzaniu i wykonywaniu **jasno** i **precyzyjnie** określonych instrukcji. Dla odróżnienia sztuczne sieci neuronowe, przynajmniej teoretycznie, jako, że modelują ich biologiczne odpowiedniki, bazują w znacznej mierze na **przetwarzaniu równoległym** i **niejawnych instrukcjach / informacjach** pochodzących z zewnętrznych sensorów. W pewnym sensie, sztuczna sieć neuronowa staje się w ten sposób złożonym procesorem statystycznym. Na tyle złożonym, że właściwie sieć traktowana jest jak **czarna skrzynka**, która przyjmuje sygnały wejściowe, daje oczekiwany sygnały wyjściowe, ale nie wiadomo jak i dlaczego.

Ujmując zagadnienie trochę bardziej formalnie, sztuczne sieci neuronowe są narzędziem do nieliniowego statystycznego modelowania danych.

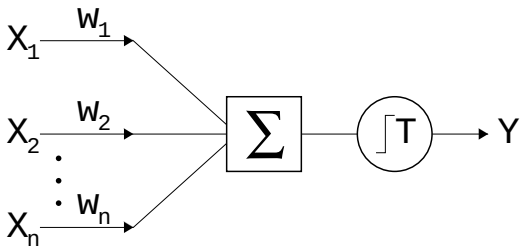
16.1 Model neuronu dla informatyka

Używajac terminologii informatycznej w budowie neuronu wyróżnić możemy następujące elementy

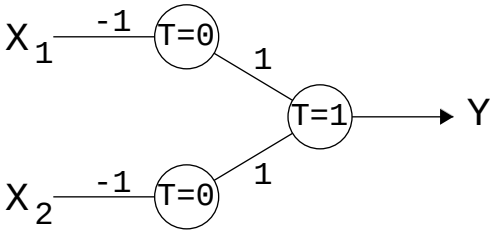
- wejście dostarczające sygnałów wejściowych (odpowiednik dendrytów),
- element przetwarzający sygnały (ciało komórki),
- wyjście (jedno) przekazujące informacje o wynikach działania.

(tutu: rys:03-01)

[2]Raczej nigdy nie zastąpi ze względu na kompletną nieprzydatność rozwiązań neuronowych do wykonywania obliczeń.



Rysunek 16.2: Model neuronu zaproponowany przez McCullocha i Pittsa.



Rysunek 16.3: Realizacja bramki NAND.

16.2 Elementarny model neuronu

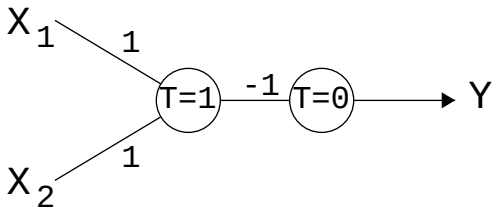
Pierwszą formalną definicję sztucznego neuronu, opartą na uproszczonym modelu biologicznym opisanym powyżej, podali McCulloch i Pitts (1943). Zaproponowany przez nich model przedstawiony jest na rysunku 16.2. Sygnały wejściowe x_i, $i = 1, \ldots, n$ mają wartość 1 lub 0, zależnie od tego czy w chwili k impuls wejściowy pojawił się, czy też nie. Sygnał wyjściowy neuronu oznaczany jest przez y. Reguła aktywacji neuronu ma dla tego modelu postać:

$$y^{k+1} = \begin{cases} 1, & \text{gdy} \quad \sum_{i=1}^{n} w_i x_i^k \geq T \\ 0, & \text{gdy} \quad \sum_{i=1}^{n} w_i x_i^k < T \end{cases}$$

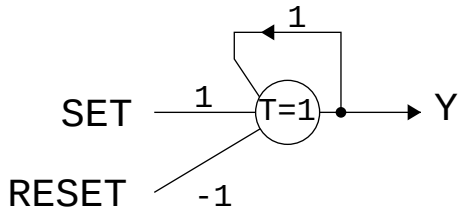
gdzie: $k = 0, 1, \ldots$ – kolejne momenty czasu,
w_i – waga przypisana połączeniu wejścia i z „ciałem" neuronu,
T – wartość progowa (poniżej której neuron nie zadziała).

Ponadto przyjmujemy $w_i = +1$ dla synaps pobudzających i $w_i = -1$, dla synaps hamujących.

Mimo całej prostoty, model ten wykazuje duże potencjalne możliwości i przy odpowiednim doborze wag i progów można z jego pomocą zrealizować funkcje logiczne NOT, OR oraz AND bądź NOR i NAND, a to z kolei, jak wiadomo z algebry Boole'a, umożliwia zbudowanie dowolnie złożonego układu kombinacyjnego. Wprowadzenie do przedstawionego modelu sygnału sprzężania zwrotnego umożliwia nam zbudowanie obwodów zachowujących się jak jednobitowy rejestr zdolny zapamiętać stan wejścia, czyli inaczej mówiąc, komórkę pamięci (rys. 16.5). Jedynka na wejściu pobudzającym powoduje aktywację komórki, a jedynka na wejściu hamującym wprowadza ją w stan nieaktywności.



Rysunek 16.4: Realizacja bramki NOR.



Rysunek 16.5: Komórka pamięci.

Przy braku sygnałów wejściowych, sygnał wyjściowy jest podtrzymywany nieskończenie długo dzięki sygnałowi sprzężenia zwrotnego.

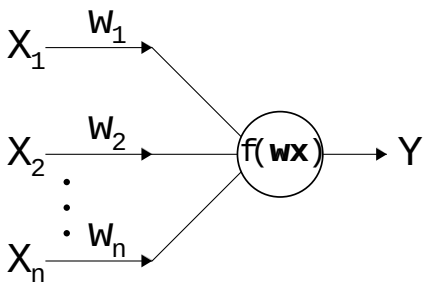
Należy jednak zdawać sobie sprawę, iż zaprezentowany powyżej model pomija wiele właściwości jakie charakteryzują rzeczywiste neurony. Najważniejsze z nich to:

- Rzeczywiste neurony, nawet w przybliżeniu, nie są elementami progowymi. Wręcz przeciwnie, reagują na wejście w sposób ciągły. Uniwersalną cechą jest natomiast nieliniowa zależność między wejściem a wyjściem komórki.
- Rzeczywisty neuron generuje ciąg impulsów, a nie sygnał wyjściowy o stałym poziomie. Przedstawienie stanu pobudzenia za pomocą jednej liczby, czy nawet w postaci ciągłej, pomija wiele informacji, jak np. faza impulsu.
- Nie wszystkie neurony mają taką samą skalę opóźnienia. Nie są też synchronizowane przez zegar centralny.
- Ilość substancji transmitera w synapsie może zmieniać się w sposób nieprzewidywalny.
- Wartości „wag" i „progów" ulegają zmianom dzięki wzajemnemu oddziaływaniu neuronów (które tu występuje jedynie w postaci przepływu sygnałów).

Dlatego też obecnie model ten stanowi jedynie punkt wyjścia do dalszych rozważań nad neuronem przydatnym do budowy sieci. Poniżej podany zostanie zasadniczy model sztucznego neuronu, używany w dalszej części.

Każdy model neuronu składa się z elementu przetwarzającego połączonego z wejściami synaptycznymi i jednym wyjściem (rys. 16.6). Przepływ sygnałów we wszystkich połączeniach jest jednokierunkowy. Sygnał wyjściowy neuronu dany jest zależnością:

$$y = f(\mathbf{w}x) = f\left(\sum_{i=1}^{n} x_i w_i\right)$$



Rysunek 16.6: Ogólny model neuronu.

gdzie w– wektor wag, x – wektor wejściowy, n - ilość sygnałów wejściowych dla danego neuronu. Funkcja $f(wx)$ nazywana jest funkcją aktywacji. Często łączne pobudzenie oznacza się symbolem net. Wówczas możemy tę funkcję zapisać jako

$$f(net) = f(wx)$$

Typowymi funkcjami aktywacji są:

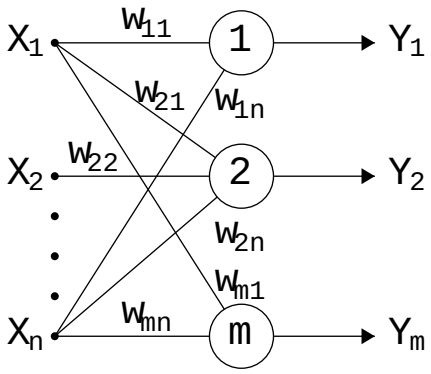
- funkcja liniowa
$$f(x) = a \cdot x, \tag{16.1}$$
gdzie a jest współczynnikiem nachylenia prostej. Na ogół przyjmuje się $a = 1.0$.

- funkcja progowa bipolarna
$$f(x) = \begin{cases} 1 & \text{dla} \quad x \geq 0, \\ -1 & \text{dla} \quad x < 0. \end{cases} \tag{16.2}$$
Czasami funkcja ta, przy dodatkowym założeniu, że dla $x = 0$, $f(x) = 0$ oznaczana jest jako $sgn(\cdot)$.

- funkcja progowa unipolarna
$$f(x) = \begin{cases} 1 & \text{dla} \quad x \geq 0, \\ 0 & \text{dla} \quad x < 0. \end{cases} \tag{16.3}$$

- funkcja sigmoidalna bipolarna
$$f(x) = \frac{2}{1 + \exp(-\lambda x)} - 1 \tag{16.4}$$
Parametr λ wpływa na „stromość" wykresu. Im ten parametr jest większy, np. rzędu 7.0, tym wykres jest bliższy wykresowi funkcji progowej. Dla małych wartości, np. 0.3, wykres przypomina funkcję liniową. Istotną zaletą tej funkcji jest jej różniczkowalność. Na ogół przyjmuje się $\lambda = 1.0$.

- funkcja sigmoidalna unipolarna
$$f(x) = \frac{1}{1 + \exp(-\lambda x)} \tag{16.5}$$



Rysunek 16.7: Siec jednokierunkowa jednowarstwowa.

- funkcja radialna

$$f(x) = \exp\left(\frac{\|x-c\|^2}{r^2}\right), \tag{16.6}$$

gdzie c i r są ustalonymi stałymi decydującymi o środku wykresu i jego szerokości (patrz rys. //uzup//).

16.3 Podstawowe architektury sieci neuronowych

Sposoby połączenia neuronów między sobą i ich wzajemnego współdziałania spowodowały powstanie różnych typów sieci. Każdy typ sieci jest z kolei ściśle powiązany z odpowiednią metodą doboru wag (uczenia). Z wielu różnych rodzajów sieci ograniczono się tutaj do kilku najczęściej stosowanych.

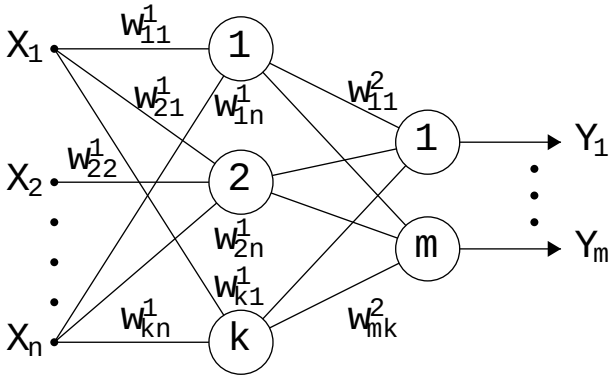
Sieć jednokierunkowa jednowarstwowa

W sieci tej neurony są ułożone w jednej warstwie, zasilanej jedynie z węzłów wejściowych (rys. 16.7). Połączenie węzłów wejściowych z neuronami warstwy wyjściowej jest zwykle pełne (każdy węzeł jest połączony z każdym neuronem). Przepływ sygnałów występuje w jednym kierunku, od wejścia do wyjścia. Węzły wejściowe nie tworzą warstwy neuronów, gdyż nie zachodzi w nich żaden proces obliczeniowy[3].

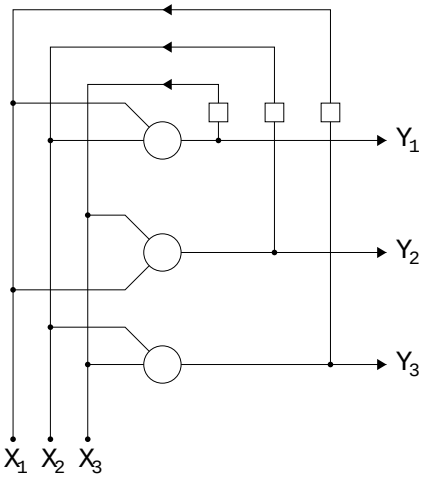
Sieć jednokierunkowa wielowarstwowa

Cechą charakterystyczną sieci jednokierunkowej wielowarstwowej jest występowanie co najmniej jednej warstwy ukrytej neuronów, pośredniczącej w przekazywaniu sygnałów między węzłami wejściowymi a warstwą wyjściową. Sygnały wejściowe są podawane na pierwszą warstwę ukrytą neuronów, a te z kolei stanowią sygnały źródłowe dla kolejnej warstwy. Typowy przykład takiej sieci przedstawia rysunek 16.8. W tym przypadku, o wiele częściej niż poprzednio, zdarza się, iż pewne połączenia międzyneuronowe nie występują. Neurony warstw ukrytych stanowią bardzo istotny element sieci, umożliwiający uwzględnienie związków między sygnałami, wynikającymi z zależności statystycznych wyższego rzędu.

[3]Choć niektórzy traktują węzły wejściowe jako neurony.



Rysunek 16.8: Siec jednokierunkowa wielowarstwowa.



Rysunek 16.9: Siec rekurencyjna.

Sieci rekurencyjne

Sieci rekurencyjne różnią się od sieci jednokierunkowych występowaniem sprzężenia zwrotnego między warstwami wyjściową i wejściową. Na rys 16.9 przedstawiono typową sieć rekurencyjną, jednowarstwową, w której sygnały wyjściowe neuronów tworzą jednocześnie wektor wejściowy sieci dla następnego cyklu (z - oznacza operator opźnienia //uzup// nie ma na rys!!!).

16.4 Nauka sieci – podstawowe modele

Chcąc nauczyć sieć neuronową, musimy dysponować zbiorem uczącym $L = l_1, ..., l_n$, gdzie n określa ilość elementów zbioru uczącego. Każdy z elementów l_i tego zbioru składa się natomiast z dwóch elementów: wektora z danymi wejściowymi p_i oraz odpowiadającego jemu wektora oczekiwanych odpowiedzi t_i. Elementy p_i oraz t_i tworzą odpowiednio zbiór **wzorcy uczących** P oraz odpowiadający im zbiór **prawidłowych**

odpowiedzi T.

Pod pojęciem uczenia sieci, będziemy rozumieli wymuszanie na niej określonego reagowania na zadane sygnały wejściowe. Osiąga się to przez adaptacyjny dobór wag, „krok po kroku", bądź etapowo według schematu

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

gdzie:

t	– numer cyklu uczącego,
$w_{ij}(t+1)$	– nowa wartość wagi synaptycznej łączącej neuron j warstwy p, z neuronem i warstwy $p+1$,
$w_{ij}(t)$	– poprzednia wartość wagi synaptycznej,
$\Delta w_{ij}(t)$	– wielkość o jaką zmodyfikuje się wartość wagi,

lub poprzez jednorazowy zapis.

Jednorazowy zapis ma miejsce wtedy, gdy wagi dostrajane są w wyniku jednej sesji treningowej, podczas której korzysta się z całego zbioru wzorcowych par wejście/wyjście. Sygnały sprzężenia zwrotnego wytwarzane w sieci nie biorą udziału w jej dostrajaniu. Sytuacja taka zachodzi, gdy możliwe jest obliczenie wartości wag według określonego wzoru. Nie mamy wówczas tak naprawdę doczynienia z procesem nauki, ale z jednorazowym ich wyznaczeniem. Ta technika uczenia jest typowa dla pamięci i występuje na przykład w sieciach Hopfielda.

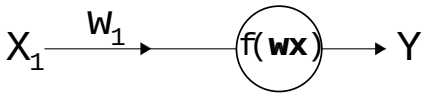
Dla innych sieci neuronowych typowe jest raczej uczenie z wykorzystaniem sprzężenia zwrotnego z otoczeniem, umownie zwane nauczaniem, realizowane etapowo i nazywane nauczaniem przyrostowym. Występują zasadniczo trzy modele nauki sieci.

Uczenie nadzorowane nazywane czasami uczeniem z nauczycielem (ang. *supervised learning*). Podczas uczenia z nauczycielem przy nowych danych wejściowych podpowiada on pożądaną odpowiedź. Odległość pomiędzy rzeczywistą a pożądaną odpowiedzią sieci jest miarą błędu używaną do korekcji parametrów sieci. Dostrajanie elementów macierzy wag może być oparte na systemie nagród i kar, jak to ma miejsce w naturalnym nauczaniu. Zestaw obrazów wejściowych i wyjściowych użytych w czasie nauki nazywamy zbiorem uczącym. Często zbiór uczący jest realizacją procesu przypadkowego i procedura minimalizacji błędu musi uwzględniać jego własności statystyczne. W rezultacie większość algorytmów uczenia z nauczycielem sprowadza się do statystycznej minimalizacji błędu w wielowymiarowej przestrzeni wag.

Uczenie nienadzorowane nazywane czasami uczeniem bez nauczyciela (ang. *unsupervised learning*). Podczas uczenia bez nauczyciela pożądana odpowiedź nie jest znana. Wobec braku informacji o poprawności czy niepoprawności odpowiedzi sieć musi się uczyć poprzez analizę reakcji na pobudzenie, o których naturze wie bardzo mało, albo wręcz nic. Wszelkie ogólne charakterystyki danych wejściowych sieć musi wykryć sama. W trakcie ich wykrywania parametry sieci podlegają zmianom, co nazywamy samoorganizacją[4].

Uczenie ze wzmocnieniem (z krytykiem) (ang. *reinforcement learning*).

[4]Termin *samoorganizacja* jest w powszechnym użyciu, ale nie tyle chodzi w tym przypadku o zmianę organizacji sieci (np. zmiana ilości warstw czy sposobu połączeń), ile o zmienianie wartości wag bez dodatkowych informacji „z zewnątrz".



Rysunek 16.10: Prosty model neuronu.

16.5 Podstawowe metody uczenia nadzorowanego w sieciach jednokierunkowych

16.5.1 Reguła perceptronowa

Jako pierwszą metodą nauki z nauczycielem zajmiemy się regułą percepronową wprowadzoną przez Rosenblatta wraz z ideą neuronu (1958).

Zanim jednak przjdziemy do opisu metody wykonajmy kilka doświadczeń.

Doświadczenie 1

Przyjmujemy model neuronu z rysunku 16.10, w którym funkcja aktywacji przyjmuje postać

$$f(x) = \begin{cases} -1 & gdy \quad x < 0 \\ 0 & gdy \quad x = 0 \\ 1 & gdy \quad x > 0 \end{cases} \tag{16.7}$$

Model ten nazywany jest **perceptronem**. Spróbujemy ustalić, czy przy zadanym modelu możliwe jest osiągnięcie odwzorowania określonego przez tabele 16.1 a następnie co należy zmienić aby realizować odwzorowanie z tabeli 16.2.

wejście	-4	-2	0	2	4
wyjście	-1	-1	0	1	1

Tabela 16.1:

wejście	-4	-2	0	2	4
wyjście	1	1	0	-1	-1

Tabela 16.2:

Zauważmy, że odwzorowanie postaci $g(x_1) = w_1 x_1$, przy ustalonym $w_1 > 0$, realizuje następujące przekształcenie

$$\begin{aligned} (-\infty, 0) &\rightarrow N \quad \text{gdy} \quad x_1 < 0, \\ 0 &\rightarrow 0, \quad \text{gdy} \quad x_1 = 0, \\ (0, +\infty) &\rightarrow P \quad \text{gdy} \quad x_1 > 0, \end{aligned}$$

gdzie N oznacza zbiór liczb ujemnych a P zbiór liczb dodatnich. Stosując teraz wzór

16.7 do argumentu $g(x_1) = w_1 x_1 = net$ otrzymujemy następujące przekształcenie

$$\begin{aligned}(-\infty, 0) &\rightarrow -1,\\ 0 &\rightarrow 0,\\ (0, +\infty) &\rightarrow +1.\end{aligned}$$

Tak więc chcąc realizować odwzorowanie opisane tabelą 16.1 wystarczy przyjąć jako w_1 dowolną liczbę dodatnią. Przyjmując np. $w_1 = 1.5$ dla $x_1 = -4$ orzymujemy:

$$net = x_1 w_1 = (-4) \cdot 1.5 = -6$$

$$f(net) = f(-6) = -1$$

co zgodne jest z definicją z tabeli.

Analogicznie, chcąc realizować odwzorowanie opisane tabelą 16.2, wystarczy przyjąć jako w_1 dowolną liczbę ujemną.

Doświadczenie 2

Ponownie przyjmujemy model neuronu z rysunku 16.10, z funkcją aktywacji zadaną wzorem (16.7). Niech teraz „poszukiwane" odwzorowanie opisuje tabela 16.3. Przy

wejście	-6	-4	-2	0	2
wyjście	-1	-1	0	1	1

Tabela 16.3:

takich założeniach, dla $w_1 > 0$, odwzorowanie $g(x_1) = w_1 x_1$ realizuje przekształcenie

$$\begin{aligned}(-\infty, 0) &\rightarrow N \quad \text{gdy} \quad x_1 < 0,\\ 0 &\rightarrow 0, \quad \text{gdy} \quad x_1 = 0,\\ (0, +\infty) &\rightarrow P \quad \text{gdy} \quad x_1 > 0,\end{aligned}$$

przez co po zastosowaniu wzoru (16.7) np. punktowi -2 będzie przypisana niezgodna z tabelą 16.3 wartość -1 a w ogólności otrzymamy

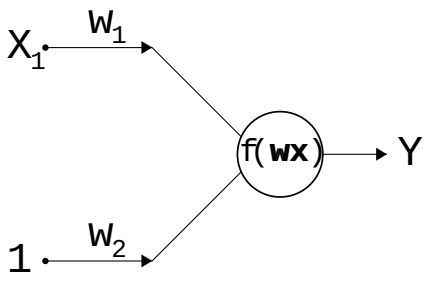
$$\begin{aligned}net &\rightarrow -1 \quad \text{gdy} \quad net < 0,\\ 0 &\rightarrow 0, \quad \text{gdy} \quad net = 0,\\ net &\rightarrow 1 \quad \text{gdy} \quad net > 0,\end{aligned} \tag{16.8}$$

Przyglądając się postaci (16.8) i mając jednocześnie na uwadze „poszukiwane" odwzorowanie określone tabelą 16.3 widzimy, że odpowiednie wyniki uzysakmy jeśli spełnione będzie

$$\begin{aligned}net &\rightarrow -1 \quad \text{gdy} \quad net < -2,\\ 0 &\rightarrow 0, \quad \text{gdy} \quad net = 0,\\ net &\rightarrow 1 \quad \text{gdy} \quad net > -2,\end{aligned}$$

Łatwo możemy to uzyskać, modyfikując funkcję aktywacji (16.7) do postaci

$$f(x) = \begin{cases} -1 & \text{gdy} \quad x < -2\\ 0 & \text{gdy} \quad x = -2\\ 1 & \text{gdy} \quad x > -2 \end{cases}$$



Rysunek 16.11:

Modyfikowanie funkcji aktywacji nie jest jednak najlepszym pomysłem. Może i dla jednego neuronu się sprawdza, ale gdy jest ich kilkadziesiąt lub kilkaset, wówczas każdy może mieć troszkę inną funkcję aktywacji co stanowiłoby pewną uciążliwość w pracy jako, że wciąż trzeba by pamietać jakiej funkcji użyć do jakiego neuronu. Zamiast tego można lekko zmodyfikować naszą sieć (tą złożoną z jednego neuronu) przenosząc zadanie „pamiętania" o zmienionym progu z funkcji aktywacji na sam neuron. W tym celu przyjmujemy model neuronu z rusunku 16.11 przy czym jego funkcja aktywacji pozostaje w pierwotnej postaci, czyli obowiązuje wzór (16.7). Chcąc teraz realizować odwzorowanie opisane tabelą 16.1 należy tak ustalić wartości wag w_1 oraz w_b aby spełnione były zależności

$$\underset{x_1\in(-\infty,-2)}{\forall} x_1 w_1 + w_b < 0$$

$$\text{gdy} \quad x_1 = -2 \quad \text{to} \quad x_1 w_1 + w_b = 0$$

$$\underset{x_1\in(-2,+\infty)}{\forall} x_1 w_1 + w_b > 0$$

Korzystając z zależności „środkowej" otrzymujemy

$$-2w_1 + w_b = 0$$

a stąd

$$w_b = 2w_1.$$

Wstawiając powyższe do zależności pierwszej otrzymujemy

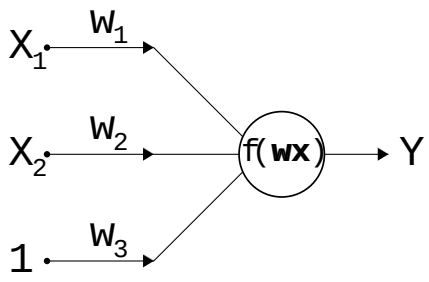
$$\underset{x_1\in(-\infty,-2)}{\forall} x_1 w_1 + 2w_1 < 0$$

$$\underset{x_1\in(-\infty,-2)}{\forall} w_1(x_1 + 2) < 0$$

z czego wnosimy, że w_1 musi być liczbą dodatnią. Ostatecznie otrzymujemy, że odwzorowanie będzie spełnione jeśli

$$w_b = 2w_1 \wedge w_1 > 0 \wedge w_b > 0.$$

Choć najbardziej naturalnym wyborem wydaje się przyjęcie $w_1 = 1$, $w_b = 2$, to jednak celem sprawdzenia przyjmijmy 2 jako wartosć wagi w_1 a 4 jako wartość wagi w_b. Otrzymujemy wówczas:



Rysunek 16.12:

- dla $x_1 = -2$

$$net = x_1 w_1 + w_b = (-2) \cdot 2 + 4 = 0$$
$$f(net) = f(0) = 0$$

- dla $x_1 = -6$

$$net = x_1 w_1 + w_b = (-6) \cdot 2 + 4 = -8$$
$$f(net) = f(-8) = -1$$

- dla $x_1 = 0$

$$net = x_1 w_1 + w_b = 0 \cdot 2 + 4 = 4$$
$$f(net) = f(4) = +1$$

co zgodne jest z definicją z tabeli.

Doświadczenie to pokazało, że oprócz „normalnych" wejść, neuron powinien posiadać jedno dodatkowe o znaczeniu specjalnym, tj. odpowiedzialne za przesuwanie progu funkcji aktywacji. Istotne jest, aby na tym specjalnym wejściu zawsze pojawiał się ten sam sygnał. Może być dowolny. W jednej sieci może to być -1, w innej 12 a jeszcze innej -72.5. Nie ma znaczenia. Ważne natomiast, aby zawsze (to znaczy przez cały czas jak działa sieć) był to ten sam sygnał. Oczywiście w zależności od wartości tego sygnału różne będą odpowiadające mu wagi pozwalające ne osiągnięcie tego samego efektu. To dodatkowe wejście, ze względu na pełnioną funkcję, nazywa się **biasem**.

Doświadczenie 3

Mamy zatem model neuronu wzbogacony o dodatkowe wejście o stałym sygnale. Umiemy także dobrać wagi tak, aby spełnione zostały odpowiednie założenia (te z tabeli). Jednakże robimy to recznie. Dla prostego zadania jest to łatwe, ale przypatrzmy się trochę bardziej złożonemu: proszę ustalić wagi tak, aby neuron z rysunku 16.12 realizował odwzorowanie zadane przez tabelę 16.4.

W tym przypadku jest to jeszcze bardziej kłopotliwe niż w doświadczeniu 1 i 2. Na podstawie zdobytej do tej pory wiedzy możemy próbować rozstrzygnąć, czy w ogóle jest to możliwe. Narysujmy w tym celu wszystkie punkty określone w tabeli (patrz rysunek //tutu). Jak widać dwie grupy punktów, tj. czarnych i białych kółek, możemy rozdzielić linią prostą. Stąd wniosek, że powinno udać się znaleźć odpowiednie wartości wag dla tego zadania. Pytanie jak?

wejście 1	-7	-5	-5	-2	1	1	3
wejście 2	-1	1	-2	0	-1	2	1
wyjście	1	1	-1	0	-1	1	-1

Tabela 16.4:

Wybierzmy dwa punkty leżące na tej prostej i na ich podstawie wyznaczmy równanie ogólne prostej. Niech punkatmi tymi będą

$$p_1 = (1, 1) \quad p_2 = (4, 2).$$

Równanie prostej wyznaczonej na ich podstawie ma postać

$$\frac{1}{3}x_1 - x_2 + \frac{2}{3} = 0.$$

Jak wiadomo dla prostej o równaniu ogólnym

$$Ax + By + C = 0$$

wektor $[A, B]$ wskazuje dodatnią półpłaszczyznę. Niestety okazuje się, że w naszym przypadku jest na odwrót (rysunek //tutu), co możemy też sprawdzić wykonując obliczenia dla dowolnego punktu z tabeli. W takim razie, wystarczy obie strony równania prostej pomnożyć przez -1

$$-\frac{1}{3}x_1 + x_2 - \frac{2}{3} = 0.$$

Nadal będzie to dokładnie taka sama prosta, ale tym razem dodatnia półpłaszczyzna będzie zawierała te punkty o które nam chodzi. Tak więc dla dowolnego punktu $p = (x_1, x_2)$ znajdującego się na znalezionej prostej mamy, że

$$f(p) = -\frac{1}{3}x_1 + x_2 - \frac{2}{3} = 0.$$

Ponad to mamy

$$f(p) = -\frac{1}{3}x_1 + x_2 - \frac{2}{3} > 0,$$

dla p leżących na dodatniej półpłaszczyźnie oraz

$$f(p) = -\frac{1}{3}x_1 + x_2 - \frac{2}{3} < 0,$$

dla p leżących na ujemnej półpłaszczyźnie. Ponieważ argument funkcji aktywacji rozważanego neuronu opisany jest wzorem

$$net = x_1 w_1 + x_2 w_2 + w_b$$

więc wystarczy przyjąć: $w_1 = -\frac{1}{3}$, $w_2 = 1$ oraz $w_b = -\frac{2}{3}$ aby otrzymać neuron o oczekiwanym działaniu. Sprawdźmy działanie neuronu na kilku wybranych przypadkach

- dla $x_1 = -7$, $x_2 = -1$

$$net = (-7) \cdot \left(-\frac{1}{3}\right) + (-1) \cdot 1 + \left(-\frac{2}{3}\right) = \frac{7}{3} - 1 - \frac{2}{3} = \frac{2}{3}$$

$$f(net) = f\left(\frac{2}{3}\right) = +1$$

- dla $x_1 = -2$, $x_2 = 0$

$$net = (-2) \cdot \left(-\frac{1}{3}\right) + (0) \cdot 1 + \left(-\frac{2}{3}\right) = \frac{2}{3} + 0 - \frac{2}{3} = 0$$

$$f(net) = f(0) = 0$$

- dla $x_1 = 3$, $x_2 = 1$

$$net = 3 \cdot \left(-\frac{1}{3}\right) + 1 \cdot 1 + \left(-\frac{2}{3}\right) = -1 + 1 - \frac{2}{3} = -\frac{2}{3}$$

$$f(net) = f\left(-\frac{2}{3}\right) = -1$$

co zgodne jest z definicją z tabeli.

Jak więc widać, możliwe jest „ręczne" znalezienie wartości wag, co jednak poprzedzone jest żmudną fazą obliczeń i analizy. Przejdźmy zatem teraz do omówienie **reguły perceptronowej**, dzięki której zautomatyzujemy proces doboru wartości wag.

Reguła perceptronu

Sygnałem uczącym r jest różnica pomiędzy odpowiedzią pożądaną $t = t_k$ a rzeczywistą y otrzymaną na wyjściu sieci po podaniu wzorca uczącego p_k, dla wybranej próbki uczącej k

$$r = t - y. \tag{16.9}$$

Korekcja wag neuronu odbywa się według zależności

$$\Delta w_i = \eta[t - f(net)]x_i, \tag{16.10}$$

gdzie zakresem zmienności i jest liczba wszystkich wejść, net jest łącznym pobudzeniem neuronu pod wpływem sygnału wejściowego $x = p_k$ a funkcja $f(\cdot)$ jest określoną wzorem (16.2) bipolarną funkcją progową

$$f(x) = \begin{cases} -1 & gdy \quad x < 0 \\ 0 & gdy \quad x = 0 \\ 1 & gdy \quad x > 0 \end{cases}$$

Stała η nazywana jest **współczynnikiem uczenia** i określa, w pewnym sensie, szykość uczenia. Zwykle jej wartości mieszczą się w przedziale $(0.01, 0.5)$.

Zilustrujemy tę metodę na przykładzie. Chcemy aby neuron realizował odwzorowanie zadane przez tabelę 16.4. Do jego nauki wykorzystamy zatem wektory

$$\begin{aligned} p_1 &= [-7 \quad -1] \\ p_2 &= [-5 \quad 1] \\ p_3 &= [-5 \quad -2] \\ p_4 &= [-2 \quad 0] \\ p_5 &= [1 \quad -1] \\ p_6 &= [3 \quad 1] \\ p_7 &= [1 \quad 2]. \end{aligned}$$

Każdy z wektorów p_1-p_7 rozszerzymy o dowolnie przez nas wybraną stałą, która odpowiadać będzie za realizacje przesunięcia. Zatem zbiór wektorów uczących przyjmie ostatecznie postać

$$\begin{aligned} p_1 &= [-7 \quad -1 \quad 1] \\ p_2 &= [-5 \quad 1 \quad 1] \\ p_3 &= [-5 \quad -2 \quad 1] \\ p_4 &= [-2 \quad 0 \quad 1] \\ p_5 &= [1 \quad -1 \quad 1] \\ p_6 &= [3 \quad 1 \quad 1] \\ p_7 &= [1 \quad 2 \quad 1]. \end{aligned}$$

Każdemu z wektorów wejściowych p_i odpowiada wektor oczekiwanej (poprawnej) odpowiedzi t_i; w naszym przypadku jest to po prostu liczba.

$$\begin{aligned} t_1 &= -1 \\ t_2 &= -1 \\ t_3 &= 1 \\ t_4 &= 0 \\ t_5 &= 1 \\ t_6 &= 1 \\ t_7 &= -1. \end{aligned}$$

Stałą uczenia η przyjmiemy równą 0.2 zaś początkowy stan wektora wag ustalimy na $w = [-1 \quad -1 \quad -1]$. Możemy teraz rozpocząć pierwszy cykl uczenia. W każdym cyklu prezentujemy jeden raz każdy z wzorców, badamy reakcję sieci i dokonujemy korekty wag. Proszę zwrócić uwagę na chwilową zmianę oznaczeń w stosunku do wzorów 16.9 i 16.10. Teraz indeks przy t nie oznacza wyjścia z i-tego neuronu, ale odpowiedź na i-ty obraz uczący. Pominięto także indeks przy w jako, że mamy tylko jeden taki wektor. Z kolei indeks przy x oznacza, że jest to i-ty obraz uczący.
Dla wzorca p_1 mamy:

$$\begin{aligned} net &= wx_1 = 7 \\ y = f(net) &= 1 \\ r &= t_1 - y = -2 \\ \Delta w &= \eta r x_1 \\ &= 0.2(-2)x_1 \end{aligned}$$

i ostatecznie

$$\begin{aligned} w &= w + \Delta w \\ &= [-1 \quad -1 \quad -1] + [2.8 \quad 0.4 \quad -0.4] \\ &= [1.8 \quad -0.6 \quad -1.4] \end{aligned}$$

(proszę porównać także z rysunkiem //uzupelnic//).
Dla wzorca p_2 mamy:

$$\begin{aligned} net &= wx_2 = -11 \\ y = f(net) &= -1 \\ r &= t_2 - y = 0 \\ \Delta w &= \eta r x_2 \\ &= 0.2(0)x_2 \end{aligned}$$

i ostatecznie

$$\begin{aligned} w &= w + \Delta w \\ &= [1.8 \quad -0.6 \quad -1.4] + [0 \quad 0 \quad 0] \\ &= [1.8 \quad -0.6 \quad -1.4] \end{aligned}$$

Dla wzorca p_3 mamy:

$$\begin{aligned} net &= wx_3 = -9.2 \\ y = f(net) &= -1 \\ r &= t_3 - y = 2 \\ \Delta w &= \eta r x_3 \\ &= 0.2(1-(-1))x_3 \end{aligned}$$

i ostatecznie

$$\begin{aligned} w &= w + \Delta w \\ &= [1.8 \quad -0.6 \quad -1.4] + [-2 \quad -0.8 \quad 0.4] \\ &= [-0.2 \quad -1.4 \quad -1.0] \end{aligned}$$

Dla wzorca p_4 mamy:

$$\begin{aligned} net &= wx_4 = -0.6 \\ y = f(net) &= -1 \\ r &= t_4 - y = 1 \\ \Delta w &= \eta r x_4 \\ &= 0.2(0-(-1))x_4 \end{aligned}$$

i ostatecznie

$$\begin{aligned} w &= w + \Delta w \\ &= [-0.6 \quad -1.4 \quad -0.8] \end{aligned}$$

Dla wzorca p_5 mamy:

$$\begin{aligned} net &= wx_5 = 0 \\ y = f(net) &= 0 \\ r &= t_5 - y = 1 \\ \Delta w &= \eta r x_5 \\ &= 0.2(1-(0))x_5 \end{aligned}$$

i ostatecznie

$$\begin{aligned} w &= w + \Delta w \\ &= [-0.6 \quad -1.4 \quad -0.8] \end{aligned}$$

Dla wzorca p_6 mamy:

$$\begin{aligned} net &= wx_6 = -4.0 \\ y = f(net) &= -1.0 \\ r &= t_6 - y = 2 \\ \Delta w &= \eta r x_6 \\ &= 0.2(1-(-1))x_6 \end{aligned}$$

i ostatecznie

$$\begin{aligned} w &= w + \Delta w \\ &= [0.6 \quad -1.0 \quad -0.4] \end{aligned}$$

Dla wzorca p_7 mamy:

$$\begin{aligned} net &= wx_7 = -1.8 \\ y = f(net) &= -1 \\ r &= t_7 - y = 0 \\ \Delta w &= \eta r x_7 \\ &= 0.2(1-(0))x_7 \end{aligned}$$

i ostatecznie

$$\begin{aligned} w &= w + \Delta w \\ &= [0.6 \quad -1.0 \quad -0.4] \end{aligned}$$

Tym samym zakończyliśmy pierwszy cykl uczenia. Zwykle cykli tych potrzeba znacznie więcej, zatem całą procedurę należy powtórzyć. Zaleca się przy tym losowy dobór próbek uczących czyli wymieszanie w ramach każdego cyklu zbioru próbek.

Dowód zbieżności reguły perceptronu

Z przeprowadzonych dotąd obserwacji, na podstawie podanych wiadomości, wnioskować można, że perceptron zawsze dokonuje podziału płaszczyzny na dwie półpłaszczyzny za pomocą prostej[5]. Oznacza to, że tylko te problemy, dla których można rozdzielić punkty na płaszczyźnie za pomocą prostej, można za pomocą perceptronu rozwiązać. Inaczej mówiąc, rozwiązywalne są tylko problemy **separowalne liniowo**.

Będziemy teraz chcieli pokazać, że metoda perceptronu dla problemów separowalnych liniowo jest zbieżna. Zbieżność w tym kontekście oznacza otrzymanie rozwiązania, to znaczy takich wartości składowych wektora wagowego, które gwarantują poprawną klasyfikację punktów ze zbioru uczącego. Wprowadźmy najpierw formalną definicję separowalności liniowej i wykorzystywanej w dowodzie absolutnej separowalności liniowej.

Definicja 16.1 (Separowalność liniowa)**.** *Dwa zbiory punktów P oraz N, takie że $P, N \subset \mathbf{R}^n$ nazywamy* **separowalnymi liniowo** *jeśli istnieje $n+1$ liczb rzeczywistych $w_1, \ldots, w_{n+1}$, takich, że dla każdego punktu $x^P = (x_1^P, \ldots, x_n^P) \in P$ i dla każdego $x^N = (x_1^N, \ldots, x_n^N) \in N$ zachodzi*

$$\sum_{i=1}^{n} w_i x_i^P \geq w_{n+1}, \tag{16.11}$$

$$\sum_{i=1}^{n} w_i x_i^N < w_{n+1}. \tag{16.12}$$

Definicja 16.2 (Absolutna separowalność liniowa)**.** *Dwa zbiory punktów P oraz N, takie że $P, N \subset \mathbf{R}^n$ nazywamy* **absolutnie separowalnymi liniowo** *jeśli istnieje $n+1$ liczb rzeczywistych $w_1, \ldots, w_{n+1}$ takich, że dla każdego punktu $x^P = (x_1^P, \ldots, x_n^P) \in P$ i dla każdego $x^N = (x_1^N, \ldots, x_n^N) \in N$ zachodzi*

$$\sum_{i=1}^{n} w_i x_i^P > w_{n+1}, \tag{16.13}$$

$$\sum_{i=1}^{n} w_i x_i^N < w_{n+1}. \tag{16.14}$$

Pokażemy teraz że obie definicje są równoważne, jeśli tylko zbiory P i N są skończone.

Lemat 16.1. *Jeśli dwa skończone zbiory punktów, P oraz N, takie że $P, N \subset \mathbf{R}^n$ są separowalne liniowo, to są także absolutnie separowalne liniowo.*

Dowód. Ponieważ zbiory P i N są separowalne liniowo, więc istnieją liczb rzeczywiste $w_1, \ldots, w_{n+1}$, takie że zachodzą wzory (16.11) oraz (16.12). Niech

$$\varepsilon = \max_{x^N \in N} \Big\{ \sum_{i=1}^{n} w_i x_i^N - w_{n+1} \Big\}.$$

[5] W ogólności, przestrzeń n wymiarowa dzielona jest dwie podprzestrzenie za pomocą jednej $n-1$ wymiarowej hiperpłaszczyzny.

Oczywiście zachodzi

$$\varepsilon < \frac{1}{2}\varepsilon < 0.$$

Niech teraz

$$w'_{n+1} = w_{n+1} + \frac{1}{2}\varepsilon.$$

Dla wszystkich punktów ze zbioru P zachodzi

$$\sum_{i=1}^{n} w_i x_i^P - w_{n+1} = \sum_{i=1}^{n} w_i x_i^P - (w'_{n+1} - \frac{1}{2}\varepsilon) \geq 0,$$

co oznacza, że

$$\sum_{i=1}^{n} w_i x_i^P - w'_{n+1} \geq -\frac{1}{2}\varepsilon > 0$$

i w konsekwencji

$$\sum_{i=1}^{n} w_i x_i^P > w'_{n+1}. \tag{16.15}$$

Dla wszystkich punktów ze zbioru N zachodzi

$$\sum_{i=1}^{n} w_i x_i^N - w_{n+1} = \sum_{i=1}^{n} w_i x_i^N - (w'_{n+1} - \frac{1}{2}\varepsilon) \leq \varepsilon,$$

co oznacza, że

$$\sum_{i=1}^{n} w_i x_i^N - w'_{n+1} \leq \frac{1}{2}\varepsilon < 0$$

i w konsekwencji

$$\sum_{i=1}^{n} w_i x_i^N < w'_{n+1}. \tag{16.16}$$

Wzory (16.15) oraz (16.16) dowodzą absolutnej separowalności liniowej zbiorów P oraz N. Tak więc pokazaliśmy, że modyfikując wagę w_{n+1} o pewną dowolnie małą liczbę (bo zamiast $\varepsilon/2$ równie dobrze można przyjąć $\varepsilon/1000$), możemy dokonać absolutnej separacji liniowej skończonych zbiorów separowalnych liniowo.

Zależność w drugą stronę jest oczywista. □

Twierdzenie 16.1 (Dowód zbieżności reguły perceptronu). *Jeśli zbiory P oraz N są absolutnie separowalne liniowo, wówczas reguła perceptronu, startując od wektora w_0, aktualizuje wektor wagowy w skończoną ilość razy.*

Dowód. Bez straty ogólności rozważań możemy przyjąć następujące, ułatwiające rozumowanie, założenia:

- Zbiory P i N są absolutnie separowalne liniowo. Dla uproszczenia zapisu możemy przez S oznaczyć zbiór będący sumą elementów z P i zanegowanych elementów z N (tak więc S zawiera elementy dodatnie).
- Wektory x tworzące zbiór S mogą zostać poddane normalizacji. Jeśli bowiem prawdą jest, że dla ustalonego wektora w zachodzi $w \cdot x > 0$, to prawdą jest także, że $w \cdot \eta x > 0$, gdzie η jest dowolną stałą dodatnia. Wektory normalizujemy w taki sposób aby ich długość była równa 1.

- Analogicznie normalizacji poddajemy wektor wagowy w. Ponieważ zakładamy absolutną separowalność liniową zbiorów P i N a zatem istnieje taki wektor w, dla którego zachodzi (16.13) oraz (16.14). Przez w^* oznaczamy jego znormalizowaną postać.

Załóżmy teraz, że jesteśmy w chwili czasowej $t+1$ i obliczyliśmy nową postać wektora wagowego w_{t+1}. Oznacza to, że

- wykonaliśmy $n+1$ kroków algorytmu;
- w chwili czasowej t pewien wektor $p_t \in S$ został niepoprawnie zaklasyfikowany, co spowodowało uaktualnienie wektora wagowego według formuły

$$w_{t+1} = w_t + \rho p_t,$$

gdzie $\rho = \eta r$ (porównaj wzory (16.9) oraz (16.10) ze strony 143).

Kosinus kąta pomiędzy wektorami w^* oraz w_{t+1} obliczmy, korzystając z definicji iloczynu skalarnego, jako

$$\cos\alpha = \frac{w^* \cdot w_{t+1}}{\|w_{t+1}\|}. \tag{16.17}$$

Niech $\delta = \min_{p \in S}(w^* \cdot p)$. Oczywiście $\delta > 0$. Mamy

$$w^* w_{t+1} = w^*(w_t + \rho p_t) = w^* w_t + \rho w^* p_t \geq w^* w_t + \rho\delta.$$

Posługując się indukcją otrzymujemy

$$w^* w_{t+1} \geq w^* w_0 + \rho(t+1)\delta. \tag{16.18}$$

Wyrażenie $\|w_{t+1}\|$ szacujemy w następujący sposób

$$\|w_{t+1}\|^2 = (\|w_t\| + \rho p_t)(\|w_t\| + \rho p_t) = \|w_t\|^2 + 2\rho w_t p_t + \|\rho p_t\|^2 \leq \|w_t\|^2 + \rho.$$

Nierówność wynika stąd, że

- $w_t p_t \leq 0$ (gdyby tak nie było, nie dokonywali byśmy korekty wag);
- $\|p_t\| = 1$.

Posługując się indukcją otrzymujemy

$$\|w_{t+1}\|^2 \geq \|w_0\|^2 + \rho(t+1). \tag{16.19}$$

Ze wzorów (16.17), (16.18) oraz (16.19) otrzymujemy nierówność

$$\cos\alpha \geq \frac{w^* w_0 + \rho(t+1)\delta}{\sqrt{\|w_0\|^2 + \rho(t+1)}}. \tag{16.20}$$

Prawa strona wyrażenia (16.20) rośnie proporcjonanie do $\sqrt{t}$ i może osiągać dowolnie duże wartości[6]. Z drugiej jednak strony $\cos\alpha \leq 1$, co z kolei implikuje, że t jest ograniczone z góry przez pewną wartość t_{max}. □

[6]Zmienna t, jako „numer" kolejnych chwil czasowych, przyjmuje wartości naturalne.

16.5.2 Przypadek ciągłej funkcji aktywacji

Zasadniczą niedogodnością w rozważanym do tej pory modelu neuronu jest nieciągłość jego funkcji aktywacji, nie pozwalająca na modelowanie zjawisk i danych typu „trochę". Binarna funkcja aktywacji działa na zasadzie „tak" (gdy jej wartość wynosi 1), „nie" (dla -1) i ewentualnie „niewiem" (dla 0). Nie pozwala jednak na modelowanie odpowiedzi typu „trochę tak" (0.4) lub „prawie jestem pewien, ale mam wątpliwości" (0.8). „Uciągloną" wersją tejże funkcji jest sigmoidalne funkacj aktywacji opisana wzorami (16.4) oraz (16.5).

Ciągła funkcja aktywacji, oprócz zdolności do modelowania danych „rozmytych", czyli nieprecyzyjnych, ma jeszcze jedną zaletę. Przy jej stosowaniu można korzystać z gradientowych (czyli posługujacych się pochodną) metod optymalizacji, co szczególnie istotne stanie się w przypdaku sieci wielowarstwowych.

Reguła delta

W regule delta sygnał uczący definiuje się jako

$$r = \delta = [t - f(net)]f'(net),$$

zaś korekta wektora wag ma postać

$$\Delta w_i = \eta(t - y)f'(net)x.$$

Reguła ta daje się wyprowadzić jako wynik minimalizacji kwadratowego kryterium błedu. Przyjmijmy model sieci przedstawiony na rysunku 16.7. Zakładamy, że neurony mają dowolną różniczkowalną funkcję aktywacji. Oznaczmy przez t pożądany sygnał wyjściowy, y faktyczny sygnał wyjściowy, x – pobudzenie sieci. Przyjmijmy ilość wejść równą n. Błąd klasyfikacji jednego obrazu wejściowego wynosi

$$E = \frac{1}{2}\sum_{k=1}^{m}(t_k - y_k)^2, \tag{16.21}$$

gdzie m jest ilością wyjść. Do wyznaczenia wartości wag minimalizujących błąd E zastosujemy metodę gradientową numerycznego wyznaczania minimum funkcji. Ustalamy pewien dowolnie wybrany punkt w i obliczamy gradient $\nabla E(w)$. Nową wartość w uzyskujemy przesuwając w w kierunku najszybszego spadku wartości funkcji, to znaczy w kierunku ujemnego gradientu. Korekta wag przyjmie więc postać

$$\Delta w = -\eta\nabla E(w).$$

W naszym przypadku wzór ten wygląda następująco

$$\Delta w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}}$$

co prowadzi do obliczeń

$$\frac{\partial E}{\partial w_{ij}} = \Big[\frac{1}{2}\sum_{k=1}^{m}(t_k - y_k)^2\Big]'_{w_{ij}} = \frac{1}{2}2\sum_{k=1}^{m}\Big[(t_k - y_k)(t_k - y_k)'_{w_{ij}}\Big]$$

$$= -\sum_{k=1}^{m}\Big[(t_k - y_k)(y_k)'_{w_{ij}}\Big] = -\sum_{k=1}^{m}\Big[(t_k - y_k)f'_{net_k}(wx)\Big]$$

$$= -\sum_{k=1}^{m}\left[(t_k - y_k)f'(net_k)(\sum_{l=1}^{n}(x_l w_{kl}))'_{w_{ij}}\right]$$
$$= -(t_i - y_i)f'(net_i)x_j.$$

Wyrażenie

$$(t_i - y_i)f'(net_i)$$

nazywa się *sygnałem błędu delta*. Ostatecznie korekta wag przyjmie postać

$$w_{ij} = w_{ij} + \eta(t_i - y_i)f'(net_i)x_j.$$

Jak widać do stosowania tej metody konieczna jest znajomość pochodnej funkcji aktywacji. Dobrze aby cały proces jej obliczania nie był zbyt długotrwały. W tym kontekście szczególnie dobrze nadają się do tego celu funkcje sigmoidalne, których pochodne wyrażają się przez nie same. Dla funkcji 16.5 jej pochodna ma postac

$$f'(net) = \lambda f(net)(1 - f(net)), \tag{16.22}$$

zaś dla funkcji 16.4

$$f'(net) = \lambda\frac{1}{2}(1 - f^2(net)). \tag{16.23}$$

Wyprowadzenie wzorów na pochodne dla funkcji sigmoidalnej

Choć wyprowadzenie wzorów na pochodne funkcji sigmoidalnej to zwykłe przekształcenia wykorzystujące elementarne własności, to jednak zaskakująco często można znaleźć błędne postaci tych wzorów. Dlatego umieścimy poniżej ich wyprowadzenie, aby każdy mógł łatwo wyjaśnić ewentualne wątpliwości.

Wykażemy najpierw prawdziwość wzoru (16.22). Wprowadźmy oznaczenie

$$f_u(x) = \frac{1}{1 + e^{-\lambda x}}.$$

Wówczas

$$f'_u(x) = \left(\frac{1}{1 + e^{-\lambda x}}\right)' = \frac{\lambda e^{-\lambda x}}{(1 + e^{-\lambda x})^2} = \lambda\frac{1}{1 + e^{-\lambda x}} \cdot \frac{1 - 1 + e^{-\lambda x}}{1 + e^{-\lambda x}} =$$

$$= \lambda\frac{1}{1 + e^{-\lambda x}}\left(\frac{-1}{1 + e^{-\lambda x}} + \frac{1 + e^{-\lambda x}}{1 + e^{-\lambda x}}\right) = \lambda f_u(x)(1 - f_u(x)).$$

Wykażemy teraz prawdziwość wzoru (16.23). Wprowadźmy oznaczenie

$$f_b(x) = \frac{2}{1 + e^{-\lambda x}} - 1.$$

Ponieważ mamy następującą zależność

$$f_b(x) = 2f_u(x) - 1$$

więc

$$f'_b(x) = (2f_u(x) - 1)' = 2f'_u(x) = 2\lambda f_u(x)(1 - f_u(x)).$$

Tak więc można wyrazić wartość pochodnej funkcji bipolarnej za pomocą wartości funkcji unipolarnej. Pokażemy teraz, że jest to możliwe także przy wykorzystaniu funkcji bipolarnej

$$f_b'(x) = \cdots = 2\lambda f_u(x)(1 - f_u(x)) = 2\lambda f_u(x) - 2\lambda f_u^2(x) = \frac{2\lambda}{1+e^{-\lambda x}} - \frac{2\lambda}{(1+e^{-\lambda x})^2} =$$

$$= \frac{2\lambda(1+e^{-\lambda x}) - 2\lambda}{(1+e^{-\lambda x})^2} = \frac{2\lambda e^{-\lambda x}}{(1+e^{-\lambda x})^2} = \frac{1}{2}\frac{4\lambda e^{-\lambda x}}{(1+e^{-\lambda x})^2} = \frac{1}{2}\frac{\lambda(2e^{-\lambda x} + 2e^{-\lambda x})}{(1+e^{-\lambda x})^2} =$$

$$= \frac{1}{2}\frac{\lambda(1 - 1 + (e^{-\lambda x})^2 - (e^{-\lambda x})^2 + 2e^{-\lambda x} + 2e^{-\lambda x})}{(1+e^{-\lambda x})^2} =$$

$$= \frac{1}{2}\frac{\lambda(1 + 2e^{-\lambda x} + (e^{-\lambda x})^2 - 1 + 2e^{-\lambda x} - (e^{-\lambda x})^2)}{(1+e^{-\lambda x})^2} =$$

$$= \frac{1}{2}\frac{\lambda\left[(1 + 2e^{-\lambda x} + (e^{-\lambda x})^2) - (1 - 2e^{-\lambda x} + (e^{-\lambda x})^2)\right]}{(1+e^{-\lambda x})^2} =$$

$$= \frac{1}{2}\frac{\lambda\left[(1+e^{-\lambda x})^2 - (1-e^{-\lambda x})^2\right]}{(1+e^{-\lambda x})^2} = \frac{1}{2}\lambda\left[1 - \left(\frac{1-e^{-\lambda x}}{1+e^{-\lambda x}}\right)^2\right] =$$

$$= \frac{1}{2}\lambda\left[1 - \left(\frac{2-1-e^{-\lambda x}}{1+e^{-\lambda x}}\right)^2\right] = \frac{1}{2}\lambda\left[1 - \left(\frac{2}{1+e^{-\lambda x}} - \frac{1+e^{-\lambda x}}{1+e^{-\lambda x}}\right)^2\right] =$$

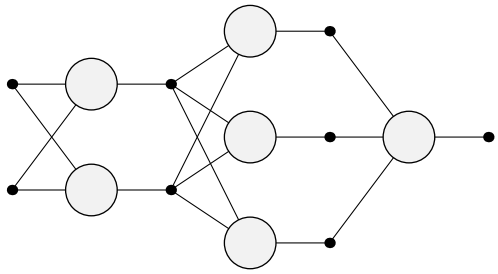
$$= \frac{1}{2}\lambda\left[1 - \left(\frac{2}{1+e^{-\lambda x}} - 1\right)^2\right] = \frac{1}{2}\lambda[1 - f_b^2(x)].$$

16.5.3 Zasada propagacji wstecznej

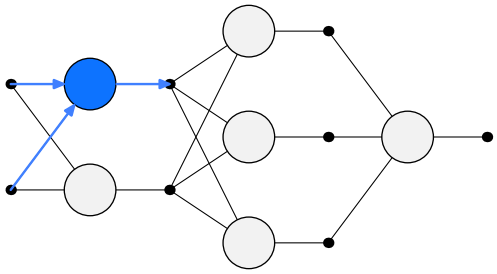
Algorytm propagacji wstecznej błędu jest podstawą większości bieżących prac na temat uczenia sieci neuronowych. Został on odkryty niezależnie przez Brysona i Ho, Werbosa, Parkera, Rumelharta, Hitona i Williamsa; pokrewny sposób podał Le Cun. Zasada **propagacji wstecznej** stanowiła istotny krok w rozwoju sieci neuronowych. Umożliwiła uczenie sieci wielowarstwowych niezbędnych do rozwiązania większości problemów. Nawet prosty wydawałoby się problem XOR (patrz rozdział 26) wymaga stosowania sieci dwuwarstwowej. Sama metoda, z formalnego punktu widzenia, nie jest niczym więcej jak gradientową metodą minimalizacji funkcji wielu zmiennych i dziwić może fakt, że aż tak długo trzeba było czekać na jej zauważenie. Podamy teraz interpretację zdarzeń zachodzących podczas procesu propagacji wstecznej, co powinno ułatwić śledzenie wywodów formalnych w dalszej części. Opiszemy przepływ sygnałów dla jednej próbki uczącej w ramach jednego cyklu. W praktyce działanie takie należy wykonać dla wszystkich próbek w ramach jednego cyklu a sam cykl powtórzyć wielokrotnie.

W celu łatwiejszego śledzenie wywodów przyjmujemy konkretny model sieci, przedstawiony na rysunku 16.13[7]. Ponadto, zgodnie z opisem w 16.4 ze strony 136, zakładamy, że mamy pewien zbiór uczący L z którego wybieramy dowolną parę (p_i, t_i), gdzie $p_i = \{p_{i1}, p_{i2}\}$, $t_i = \{t_i\}$.

[7]Uwaga: na rysunkach nie uwzględniono połączeń z biasem, które występują jednak we wzorach (sygnały x i wagi w przy których występuje indeks o wartości 0).



Rysunek 16.13: Ogólny wygląd sieci do przykładu ilustrującego zasadę działania propagacji wstecznej.



$$net_1^1 = w_{10}^1 x_0^1 + w_{11}^1 x_1^1 + w_{12}^1 x_2^1$$
$$y_1^1 = f(net_1^1)$$
$$x_1^2 = y_1^1$$

Rysunek 16.14: Przebieg sygnałów pobudzających pierwszy neuron w warstwie pierwszej.

- Propagacja sygnałów w warstwie pierwszej (wejściowej) (porównaj rysunki 16.14–16.15).
  - Obliczenie pobudzenia net_1^1, net_2^1 neuronów numer 1 i 2 z warstwy pierwszej:

$$\begin{aligned} net_1^1 &= w_{10}^1 x_0^1 + w_{11}^1 x_1^1 + w_{12}^1 x_2^1, \\ net_2^1 &= w_{20}^1 x_0^1 + w_{21}^1 x_1^1 + w_{22}^1 x_2^1. \end{aligned}$$

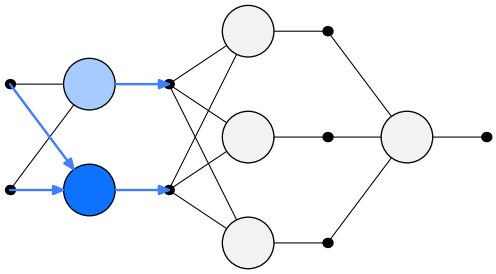
  - Obliczenie wartości funkcji aktywacji:

$$\begin{aligned} y_1^1 &= f(net_1^1), \\ y_2^1 &= f(net_2^1). \end{aligned}$$

  - Sygnały wyjściowe z tej warstwy stają się sygnałami wejściowymi dla warstwy kolejnej:

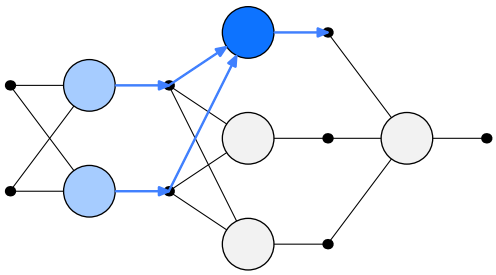
$$\begin{aligned} x_1^2 &= y_1^1, \\ x_2^2 &= y_2^1. \end{aligned}$$

- Propagacja sygnałów w warstwie drugiej (ukrytej) (porównaj rysunki 16.16–16.18).



$$net_2^1 = w_{20}^1 x_0^1 + w_{21}^1 x_1^1 + w_{22}^1 x_2^1$$
$$y_2^1 = f(net_2^1)$$
$$x_2^2 = y_2^1$$

Rysunek 16.15: Przebieg sygnałów pobudzających drugi neuron w warstwie pierwszej.



$$net_1^2 = w_{10}^2 x_0^2 + w_{11}^2 x_1^2 + w_{12}^2 x_2^2$$
$$y_1^2 = f(net_1^2)$$
$$x_1^3 = y_1^2$$

Rysunek 16.16: Przebieg sygnałów pobudzających pierwszy neuron w warstwie drugiej.

– Obliczenie pobudzenia net_1^2, net_2^2 i net_3^2 neuronów numer 1, 2 i 3 z warstwy drugiej:

$$\begin{aligned} net_1^2 &= w_{10}^2 x_0^2 + w_{11}^2 x_1^2 + w_{12}^2 x_2^2, \\ net_2^2 &= w_{20}^2 x_0^2 + w_{21}^2 x_1^2 + w_{22}^2 x_2^2, \\ net_3^2 &= w_{30}^2 x_0^2 + w_{31}^2 x_1^2 + w_{22}^2 x_2^2. \end{aligned}$$

– Obliczenie wartości funkcji aktywacji:

$$\begin{aligned} y_1^2 &= f(net_1^2), \\ y_2^2 &= f(net_2^2), \\ y_3^2 &= f(net_3^2). \end{aligned}$$

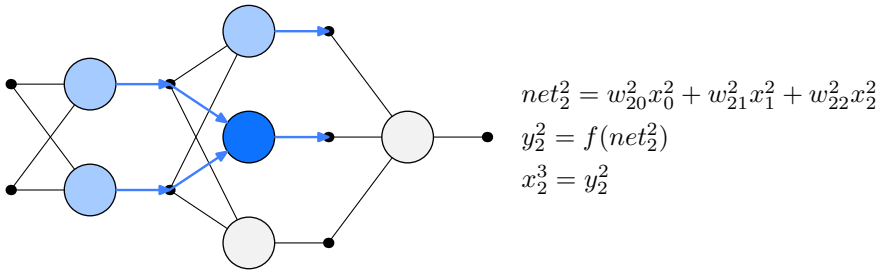
– Sygnały wyjściowe z tej warstwy stają się sygnałami wejściowymi dla warstwy kolejnej:

$$\begin{aligned} x_1^3 &= y_1^2, \\ x_2^3 &= y_2^2, \\ x_3^3 &= y_3^2. \end{aligned}$$

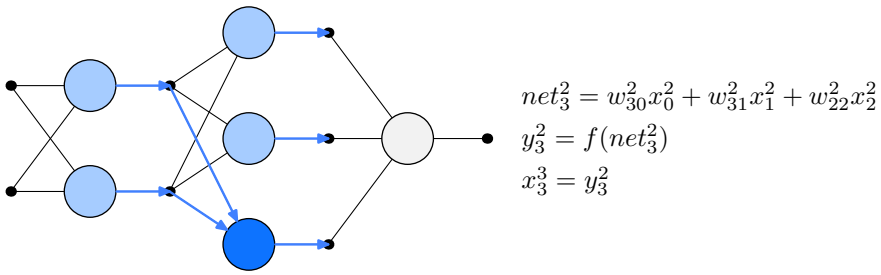
- Propagacja sygnałów w warstwie trzeciej (wyjściowej) (porównaj rysunek 16.19).

– Obliczenie pobudzenia net_1^3 neuronu numer 1 z warstwy trzeciej:

$$net_1^3 = w_{10}^3 x_0^3 + w_{11}^3 x_1^3 + w_{12}^3 x_2^3 + w_{13}^3 x_3^3.$$



Rysunek 16.17: Przebieg sygnałów pobudzających drugi neuron w warstwie drugiej.



Rysunek 16.18: Przebieg sygnałów pobudzających trzeci neuron w warstwie drugiej.

– Obliczenie wartości funkcji aktywacji:

$$y_1^3 = f(net_1^3).$$

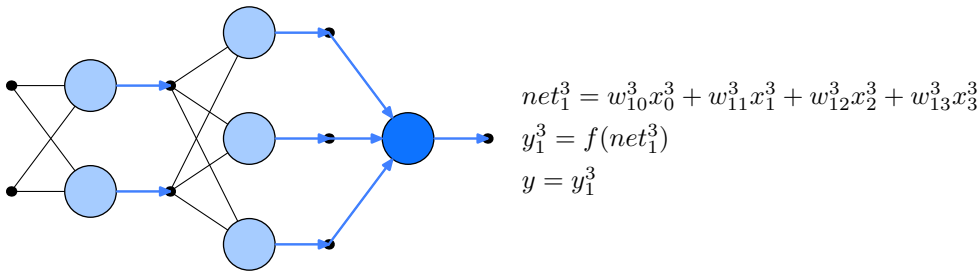
– Sygnały wyjściowe z tej warstwy stają się odpowiedzią sieci

$$y = y_1^3.$$

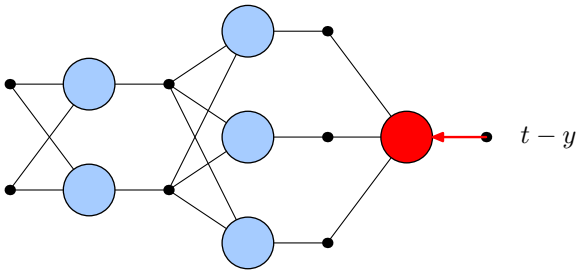
- Obliczenie wartości błędu jaki popełniła sieć (rysunek 16.20)

$$\delta = t - y.$$

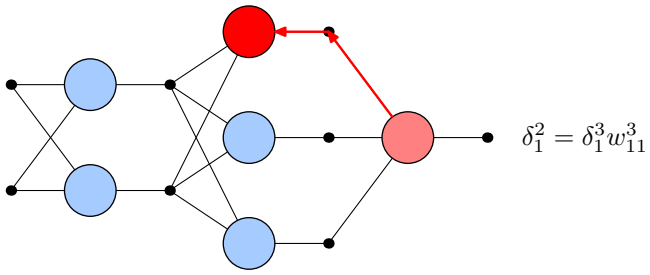
Sygnał błędu ma o tyle duże znaczenie, że dzięki niemu wiadomo jaką poprawkę należy wprowadzić do wektora wagowego.



Rysunek 16.19: Przebieg sygnałów pobudzających pierwszy neuron w warstwie trzeciej.



Rysunek 16.20: Obliczenie wartości błędu popełnionego przez sieć.



Rysunek 16.21: Obliczenie wartości błędu popełnionego przez neuron pierwszy z warstwy drugiej.

- Propagowanie sygnału błędu do warstwy ukrytej (porównaj rysunki 16.21–16.23).

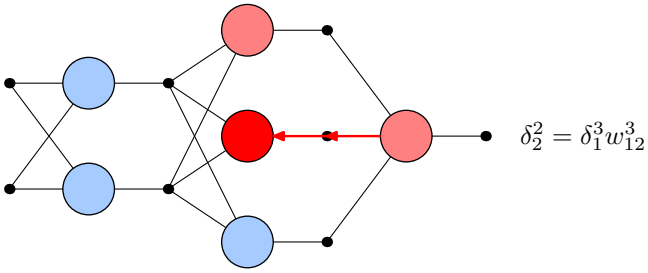
  Nie jest możliwe bezpośrednie obliczenie wartości sygnału błędu dla neuronów warstw innych niż wyjściowa. Wychodząc jednak ze słusznego założenia, że na błąd końcowy miały także wpływ neurony z innych warstw a nie tylko wyjściowe, można próbować „obiążyć" je częścią „odpowiedzialności". Odpowiedzialność będzie tym większa im większą wartość bezwzględną miała waga na połączeniu neuronu warstwy ukrytej i wyjściowej. Dlatego oznaczając δ przez δ_1^3 otrzymujemy

$$\begin{aligned} \delta_1^2 &= \delta_1^3 w_{11}^3, \\ \delta_2^2 &= \delta_1^3 w_{12}^3, \\ \delta_3^2 &= \delta_1^3 w_{13}^3. \end{aligned}$$

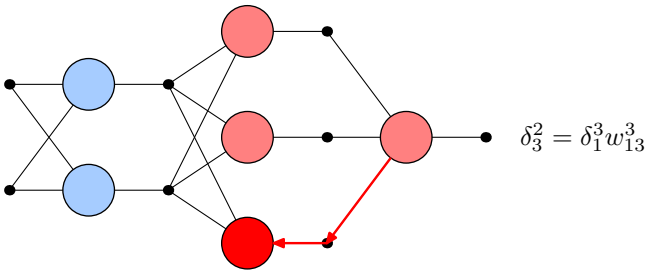
- Propagowanie sygnału błędu do warstwy wejściowej (porównaj rysunki 16.24–16.25).

  Na podobnej zasadzie obliczamy sygnały błędu dla neuronów z warstwy wejściowej.

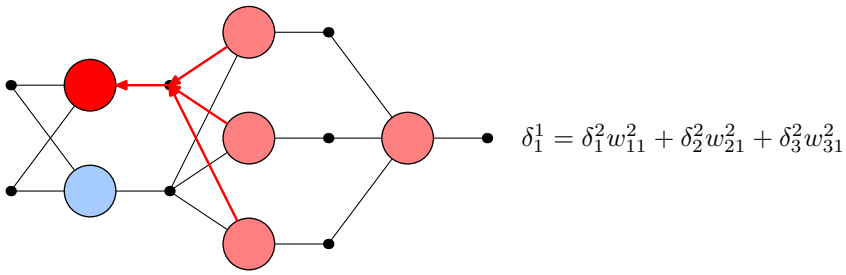
$$\begin{aligned} \delta_1^1 &= \delta_1^2 w_{11}^2 + \delta_2^2 w_{21}^2 + \delta_3^2 w_{31}^2, \\ \delta_2^1 &= \delta_1^2 w_{12}^2 + \delta_2^2 w_{22}^2 + \delta_3^2 w_{32}^2. \end{aligned}$$



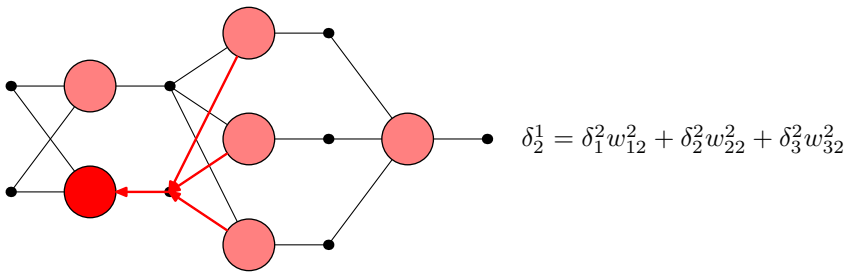
Rysunek 16.22: Obliczenie wartości błędu popełnionego przez neuron drugi z warstwy drugiej.



Rysunek 16.23: Obliczenie wartości błędu popełnionego przez neuron trzeci z warstwy drugiej.



Rysunek 16.24: Obliczenie wartości błędu popełnionego przez neuron pierwszy z warstwy pierwszej.



Rysunek 16.25: Obliczenie wartości błędu popełnionego przez neuron drugi z warstwy pierwszej.

- Uaktualnienie wag.

  Ponieważ teraz dla każdego neuronu znana jest wartość błędu, więc można policzyć poprawki dla wszystkich wag. Kolejność liczenia nie ma teraz znaczenia (można uaktualniać wagi zaczynając albo od początku albo od końca).

16.5.4 Uogólniona reguła delta

Podstawę algorytmu stanowi spadek gradientu (reguła delta) opisany na stronie 149. Podane poniżej wzory oraz ich wyprowadzenie stanowią klasykę w dziedzinie sieci neuronowych.

Podobnie jak poprzednio n oznacza liczbę neuronów wejściowych, m – liczbę neuronów w warstwie wyjściowej oraz k – liczbę neuronów w warstwie ukrytej. Podstawę algorytmu stanowi przyjęcie funkcji celu w postaci sumy kwadratów różnic między aktualnymi wartościami sygnałów wyjściowych sieci a wartościami żądanymi. Uaktualnianie wag może odbywać się po każdorazowej prezentacji próbki uczącej lub jednorazowo, po prezentacji wszystkich próbek tworzących cykl uczący. W celu uproszczenia do dalszych rozważań przyjęto funkcję celu 16.21, która odpowiada aktualizacji wag po każdorazowej prezentacji próbki. Ponadto ograniczamy się tylko do dwóch warstw, ale nic nie stoi na przeszkodzie aby w ramach ćwiczenia wyprowadzić sobie wzory dla sieci 10-cio warstwowej...i przekonać się o ich analogicznej postaci.

Przy oznaczeniach pokazanych na rys 16.8 ze strony 136, funkcję błędu E można opisać zależnością:

$$E = \frac{1}{2}\sum_{p=1}^{m}[t_p - y_p]^2 = \frac{1}{2}\sum_{p=1}^{m}[t_p - f(net_p^2)]^2 =$$

$$= \frac{1}{2}\sum_{p=1}^{m}\left[t_p - f\left(\sum_{q=1}^{k} w_{pq}^2 f(net_q^1)\right)\right]^2 =$$

$$= \frac{1}{2}\sum_{p=1}^{m}\left[t_p - f\left(\sum_{q=1}^{k} w_{pq}^2 f\left(\sum_{r=1}^{n} w_{qr}^1 x_r\right)\right)\right]^2$$

gdzie net_q^1, net_p^2 oznaczają odpowiednio pobudzenie q-tego neuronu w warstwie 1 i p-tego w warstwie 2.

Obliczymy teraz $\frac{\partial E}{\partial w_{ij}^1}$, $1 \leq i \ \leq k$, $1 \leq j \leq n$:

$$\frac{\partial E}{\partial w_{ij}^1} = \frac{1}{2}2\sum_{p=1}^{m}\left\{[t_p - f(net_p^2)](-f'(net_p^2))\sum_{q=1}^{k} w_{pq}^2 f'(net_q^1)\sum_{r=1}^{n}\delta_{qi}\delta_{rj}x_r\right\} =$$

$$= -\sum_{p=1}^{m}\left\{[t_p - f(net_p^2)]f'(net_p^2)w_{pi}^2 f'(net_i^1)x_j\right\}$$

Podobnie jak dla sieci jednowarstwowej, przez δ oznaczamy następujące fragmenty ostatniego wzoru

$$\delta_p^2 = [t_p - f(net_p^2)]f'(net_p^2)$$

dla $p = 1, \ldots, k$, oraz

$$\delta_i^1 = [t_p - f(net_p^2)]f'(net_p^2)w_{pi}^2 f'(net_i^1)$$

Stąd też

$$\delta_i^1 = \sum_{p=1}^{m} \delta_p^2 w_{pi} f'(net_i^1)$$

gdzie $i = 1, \ldots, m$ i ostatecznie

$$\frac{\partial E}{\partial w_{ij}^1} = \delta_i^1 x_j$$

Otrzymaliśmy w ten sposób wzór na zmianę wartości wag warstwy pierwszej. Wzory na zmianę wag warstwy drugiej są analogiczne do tych wyprowadzonych na stronie 149.

Znajomość powyższych wzorów „na pamięć" nie jest konieczna do zrozumienia tej metody. Co więcje, znając zasadę działania metody opisaną w 16.5.3 bardzo łatwo można powyższe wzory odtworzyć. Pokazują one bowiem, że sama metoda nie została wyssana z palca a wręcz przeciwnie, posiada silne podstawy teoretyczne.

Istotnym problemem związanym z sieciami wielowarstwowymi jest dobór ich struktury, to znaczy ułożenia neuronów w warstwy i ich wzajemne połączenie. Niestety nie ma uniwersalnej metody pozwalającej określić architekturę sieci. Na temat doboru struktury powstało wiele prac, jednakże żadna z nich nie daje rozstrzygającej odpowiedzi. Zwykle podkreśla się, że najlepszym wyjściem jest przebadanie pewnej ilości sieci i wybranie tej najlepiej działającej lub dobra intuicja.

Rozdział 17

Algorytmy gradientowe uczenia sieci wielowarstwowych skierowanych do przodu

Reguła delta, choć zgodnie uważana za wręcz klasyczną, nie jest jedyną metodą nauki sieci wielowarstwowych skierowanych do przodu. Istnieje dosyć duża grupa alternatywnych metod. Przyjrzymy się im w tym rozdziale.

17.1 Metoda najszybszego spadku

Poznana reguła delta, wraz z jej uogólnieniem na przypadek sieci wielowarstwowej, należy do tak zwanych **gradientowych metod nauki**. W ogólności mówimy o gradientowych metodach minimalizacji funkcji. Cechą charakterystyczną tych metod jest modyfikowanie wag według wzoru

$$\Delta w_{ij}(t) = \eta v(w_{ij}(t)),$$

gdzie v jest kierunkiem na którym będzie się poszukiwać minimalnej wartości zadanej funkcji zaś współczynnik[1] $\eta \in R$ określa wielkość kroku wykonanego w wyznaczonym przez v kierunku. Kierunek v, jak sugeruje ogólna nazwa tych metod, wyznaczany jest w oparciu o gradient a czasem także i składowe wyższych rzędów (np. hesjan). Różnice między algorytmami występują w sposobie wyznaczania kierunku poszukiwań v oraz wartości kroku η.

Dla reguły delta wektor v określany jest jako $-\nabla E(w)$. Stąd też i druga nazwa tego algorytmu: **algorytm najszybszego spadku**. Gradient wskazuje kierunek najszybszego (lokalnego) wzrostu wartości funkcji. Tak więc minus gradient wskazuje kierunek najszybszego spadku wartości funkcji.

Fakt iż gradient faktycznie wskazuje kierunek najszybszego wzrostu (spadku) wartości funkcji uzasadnić można w następujący sposób.

Twierdzenie 17.1. *Niech f będzie funkcją różniczkowalną w pewnym otoczeniu punktu w_0. Wówczas wektor $v = -\nabla f(x_0)$ wskazuje kierunek najszybszego spadku wartości funkcji. Przy takim wyborze wektora v szybkość zmian wartości funkcji $f(\cdot)$ wynosi dokładnie $-\|\nabla f(x_0)\|$.*

[1] Nazywany też współczynnikiem uczenia lub krokiem. Zwykle jego wartości określa się jako małą liczbę dodatnią z przedziału $(0, 1]$, ale w praktyce bardziej nietypowe wartości też się zdarzają.

Dowód. Niech v będzie wektorem jednostkowym. Oznaczmy przez g funkcję taką, że

$$g(\eta) = f(x_0 + \eta v), \quad \eta \geq 0.$$

Funkcja $g(\eta)$ jest zbiorem tych wartości funkcji $f(\cdot)$ dla których argumenty leżą na półprostej o początku w punkcie w_0 i kierunku wyznaczonym przez wektor v. Różniczkując funkcję $g(\eta)$ otrzymujemy

$$g'(\eta) = \nabla f(x_0 + \eta v)v,$$

a w szczególności dla $\eta = 0$ otrzymujemy

$$g'(0) = \nabla f(x_0)v$$

czyli tak zwaną pochodną kierunkową funkcji $f(\cdot)$ w punkcie x_0 w kierunku wektora v oznaczaną jako $\nabla_v f(x_0)$. Z elementarnego kursu analizy rzeczywistej wiadomo, że pochodna, mówiąc obrazowo, niesie informacje o stopniu nachylenia funkcji, a więc o tym jak szybko zmieniają się wartości funkcji. Im większa (dodatnia) wartość pochodnej tym szybciej rosną wartości funkcji, im mniejsza (ujemna) tym szybciej malają. Poszukujemy zatem takiego kierunku v, dla którego wartość $\nabla_v f(x_0)$ będzie najmniejsza. Korzystając z nierówności Cauchy'ego–Schwarz'a (twierdzenie 4.3) otrzymujemy

$$-\|\nabla f(x_0)\| = -\|\nabla f(x_0)\|\|v\| \leq \nabla f(x_0)v = \nabla_v f(x_0) \leq \|\nabla f(x_0)\|\|v\| = \|\nabla f(x_0)\|,$$

czyli

$$-\|\nabla f(x_0)\| \leq \nabla_v f(x_0) \leq \|\nabla f(x_0)\|.$$

Pierwsza nierówność jest konsekwencją następującego rozumowania. Z twierdzenia 4.3 (nierówność Cauchy'ego–Schwarz'a) a także definicji iloczynu skalarnego (patrz rozdział 4) mamy

$$|x \cdot y| \leq \|x\|\|y\|$$

co przekształcone daje

$$-\|x\|\|y\| \leq -|x \cdot y|.$$

Ponieważ dla dowolnego $x \in R$ zachodzi

$$-|x| \leq x$$

więc

$$-\|x\|\|y\| \leq x \cdot y.$$

Druga z nierówności jest prostą konsekwencją twierdzenia 4.3.

Z powyższego oszacowania widzimy, że pochodna kierunkowa $\nabla_v f(x_0)$ osiąga swoją najmniejszą możliwą wartość równą

$$-\|\nabla f(x_0)\| = \nabla_v f(x_0) \tag{17.1}$$

wtedy, gdy v jest wektorem jednostkowym postaci

$$v = -\frac{\nabla f(x_0)}{\|\nabla f(x_0)\|}. \tag{17.2}$$

Istotnie, zgodnie z definicjami z rozdziału 4 dotyczącymi iloczynu skalarnego, dla dowolnego wektora x zachodzi

$$\|x\| = \sqrt{x \cdot x}. \tag{17.3}$$

Niech u i v będą wektorami i oznaczmy przez a i b następujące wyrażenia

$$a = \|u\|,$$

$$b = u \cdot v$$

oraz

$$v = \frac{u}{\|u\|}. \tag{17.4}$$

Wówczas

$$b = u \cdot v = u \cdot \frac{u}{\|u\|} = \frac{u \cdot u}{\|u\|} = \dots$$

a uwzględniając (17.3) otrzymujemy

$$\dots = \frac{u \cdot u}{\|u\|} = \|u\| = a.$$

Zatem jeśli tylko przyjmiemy, że v jest określone jak w (17.4) wówczas zachodzi równość $a = b$ a w konsekwencji równość (17.1) dla v określonego jak w (17.2). Przy takim wyborze wektora v szybkość zmian wartości funkcji $f(\cdot)$ wynosi dokładnie $-\|\nabla f(x_0)\|$. □

Niestety metoda najszybszego spadku (reguła delta) jest bardzo wrażliwa na dobór parametru η co ilustrują rysunki //uzup//. Dlatego też bardzo często bywa ona łączona z minimalizacją kierunkową. Minimalizacja kierunkowa oznacza każdorazowy dobór takiej wartości parametru η aby osiągnięte zostało minimum funkcji $f(\cdot)$ obciętej do kierunku v.

17.1.1 Metoda najszybszego spadku z minimalizacją kierunkową

Definicja 17.1. *Niech $f(x)$, $x \in R^n$ ma ciągłe pochodne cząstkowe pierwszego rzędu. Wówczas dla pewnego punktu początkowego $x_0 \in R^n$ metoda najszybszego spadku z minimalizacją kierunkową określa ciąg $\{x_k\}_{k \in N}$ według zależności*

$$x_{k+1} = x_k - \eta_k \nabla f(x_k), \tag{17.5}$$

gdzie η_k jest taką wartością parametru $\eta > 0$, która minimalizuje wyrażenie

$$g(\eta) = f(x_k - \eta \nabla f(x_k)), \quad \eta \geq 0. \tag{17.6}$$

W dalszej części mówiąc „metoda najszybszego spadku" będziemy mieć zawsze na myśli jej zmodyfikowaną o minimalizaję kierunkową wersje.

Tak sformułowana metoda najszybszego spadku ma dosyć ciekawą własność.

Twierdzenie 17.2. *Kierunki generowane przez metodę najszybszego spadku z minimalizacją kierunkową są do siebie prostopadłe. Precyzyjniej, jeśli $\{x_k\}_{k \in N}$ jest ciągiem punktów wygenerowanych przez tą metodę, wówczas wektor o początku w punkcie x_k oraz końcu w x_{k+1} jest prostopadły do wektora o początku w x_{k+1} oraz końcu w x_{k+2}.*

Dowód. Korzystając z definicji 17.1 mamy

$$(x_{k+1} - x_k)(x_{k+2} - x_{k+1}) = \eta_k \nabla f(x_k) \eta_{k+1} \nabla f(x_{k+1}).$$

Aby wykazać prostopadłość wektorów $[x_k, x_{k+1}]$ i $[x_{k+1}, x_{k+2}]$ wystarczy pokazać, że zachodzi

$$\nabla f(x_k) \nabla f(x_{k+1}) = 0.$$

Ponieważ zgodnie z definicją η_k jest taką liczbą dla której zachodzi (17.5) oraz która minimalizuje wyrażenie (17.6), więc

$$0 = g'(\eta_k) = -\nabla f(x_k - \eta_k \nabla f(x_k)) \nabla f(x_k) = -\nabla f(x_{k+1}) \nabla f(x_k).$$

□

Istotną zaletą minimalizacji na kierunku jest znacznie szybsza zbieżność na płaskich obszarach (tam gdzie pochodna bliska jest zeru). Wykazana prostopadłość często powoduje jednak powolną zbieżność w końcowym etapie – jego zbieżność w pobliże minimum jest szybka (rzędu kilka, kilkanaście kroków), ale już znalezienie minimum z założonym ε może wymagać kilkuset kroków. Ponadto sam proces minimalizacji kierunkowej może zajmować znaczną ilość czasu i mocy obliczeniowej. //uzup//rysunki

Metoda ta ma jednak jedną bardzo przyjemną własność: wartości minimalizowanej funkcji obliczane w kolejnych punktach wskazanych przez metodę są co raz mniejsze, co pokazuje poniższe twierdzenie.

Twierdzenie 17.3. *Jeśli ciąg $\{x_k\}_{k \in N}$ składa się z punktów wygenerowanych przez metodę najszybszego spadku i jeśli $\nabla f(x_k)$ jest różne od 0 dla pewnego k, wówczas $f(x_{k+1}) < f(x_k)$.*

Dowód. Zgodnie z definicją 17.1 zachodzi wzór 17.5, gdzie η_k minimalizuje wyrażenie 17.6. Stąd

$$f(x_{k+1}) = g(\eta_k) \leq g(\eta).$$

Różniczkując funkcję $g(\cdot)$ otrzymujemy

$$g'(0) = -\nabla f(x_k - 0 \cdot \nabla f(x_k)) \cdot \nabla f(x_k) = -\nabla f(x_k) \cdot \nabla f(x_k) = -\|\nabla f(x_k)\|^2 < 0.$$

Oznacza to, że istnieje $\tilde{\eta} > 0$ takie, że dla wszystkich $0 < \eta \leq \tilde{\eta}$ zachodzi

$$g(0) > g(\eta).$$

W konsekwencji otrzymujemy

$$f(x_{k+1}) = g(\eta_k) \leq g(\tilde{\eta}) < g(0) = f(x_k).$$

□

17.2 Metoda Newtona

Metoda Newtona nie ma bezpośredniego zastosowania w uczeniu sieci neuronowych, ale jest podstawą do wyprowadzenia i zrozumienia działania innych metod. Z tego też powodu podajemy jej krótką charakterystykę.

Definicja 17.2 (Metoda Newtona rozwiązywania równań)**.** *Niech $g : R^n \to R^n$ będzie funkcją różniczkowalną. Wówczas ciąg punktów $\{x_k\}_{k\in N}$ generowany prze metodę Newtona, dla równania $g(x) = 0$ i pewnego punktu początkowego x_0 określony jest przez formułę postaci*

$$\nabla g(x_k)(x_{k+1} - x_k) = -g(x_k)$$

lub równoważnie

$$x_{k+1} = x_k - [\nabla g(x_k)]^{-1} g(x_k).$$

Zauważmy, że:

- Druga postać formuły jest znacznie czytelniejsza – jasno widać w niej jak x_{k+1} zależy od x_k. W zastosowaniach praktycznych znacznie częściej jednak używa się pierwszej formuły jako, że nie wymaga ona jawnego obliczania odwrotności dla macierzy $\nabla g(\cdot)$.

- Geometryczną interpretację metody bardzo łatwo zrozumieć jeśli przyjmiemy, że funkcja g działa ze zbioru R w zbiór R. Wówczas otrzymujemy formułę postaci

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}.$$

  Punkt x_{k+1} jest wówczas punktem przecięcia prostej stycznej do wykresu funkcji $g(\cdot)$ w punkcie $(x_k, g(x_k))$ z osią OX. Prosta ta ma równanie

$$y - g(x_k) = g'(x_k)(x - x_k).$$

- Można wykazać, że metoda Newtona jest zbieżna do pewnego rozwiązania x^* jeśli:
  - punkt początkowy x_0 nie jest „zbyt daleko" od punktu x^*;
  - wykres funkcji $g(\cdot)$ jest „porządny".

W oparciu o metodę rozwiązywania równań bardzo łatwo można już wskazać metodę minimalizacji funkcji.

Definicja 17.3 (Metoda Newtona minimalizacji funkcji)**.** *Niech $f : R^n \to R^n$ będzie funkcją dwukrotnie różniczkowalną w sposób ciągły. Wówczas ciąg punktów $\{x_k\}_{k\in N}$ generowany prze metodę Newtona, dla zadania $\min_x f(x)$ i pewnego punktu początkowego x_0 określony jest przez formułę postaci*

$$Hf(x_k)(x_{k+1} - x_k) = -\nabla f(x_k)$$

lub równoważnie

$$x_{k+1} = x_k - [Hf(x_k)]^{-1} \nabla f(x_k),$$

gdzie H jest hesjanem funkcji $f(\cdot)$.

Zauważmy, że:

- Minimalizacja jest bezpośrednią konsekwencją definicji **??**, w której za $g(\cdot)$ przyjmujemy $\nabla f(\cdot)$.

- Metoda Newtona jako kierunek poszukiwań wartości minimalnej wybiera wektor

$$v = -\frac{\nabla f(x_k)}{Hf(x_k)}. \tag{17.7}$$

Jak widać do wyznaczenia kierunku w każdym cyklu należy określić wartość gradientu $\nabla f(\cdot)$ oraz hesjanu $Hf(\cdot)$ w punkcie ostatniego znanego rozwiązania x_k

- Metoda Newtona minimalizacji funkcji może zostać łatwo wyprowadzona także w następujący sposób. Rozwijając funkcję $f(\cdot)$ w szereg Taylora w punkcie x_k i ograniczając się jedynie do pierwszych trzech składników otrzymujemy

$$\tilde{f}(x) = f(x_k) + \nabla f(x_k)(x - x_k) + \frac{1}{2}(x - x_k)Hf(x_k)(x - x_k),$$

czyli funkcję kwadratową najlepiej przybliżającą funkcję $f(\cdot)$ w rozważanym punkcie x_k. Najlepsze przybliżenie jest w sensie identycznych wartości funkcji $f(\cdot)$ i $\tilde{f}(\cdot)$, ich pierwszych oraz drugich pochodnych w punkcie x_k. Jeśli funkcja kwadratowa $\tilde{f}(\cdot)$ osiąga minimum w punkcie x_{min}, punkt ten musi być punktem krytycznym tej funkcji, a więc

$$0 = \nabla \tilde{f}(x_{min}) = \nabla f(x_k) + Hf(x_k)(x_{min} - x_k).$$

Jeśli hesjan jest dodatnio określony //uzup// wówczas funkcja $\tilde{f}(\cdot)$ jest ściśle wypukła i ma minimum globalne właściwe w punkcie x_{min}. Podstawiając teraz w poprzednim równaniu x_{k+1} zamiast x_{min} (ponieważ x_{k+1} jest przybliżeniem właśnie poszukiwanego x_{min}) otrzymujemy wcześniejszą formułę rekurencyjną

$$Hf(x_k)(x_{k+1} - x_k) = -\nabla f(x_k).$$

Dodatnia określoność hesjanu jest w tym przypadku istotna co pokazuje następujące twierdzenie.

Twierdzenie 17.4. *Niech $\{x_k\}_{k \in N}$ będzie ciągiem punktów wygenerowanych przez metodę Newtona dla pewnej funkcji $f(\cdot)$. Jeśli hesjan $Hf(x_k)$ jest dodatnio określony oraz jeśli $\nabla f(x_k)$ jest różne od zera, wówczas kierunek*

$$v = -\frac{\nabla f(x_k)}{Hf(x_k)}$$

jest kierunkiem spadku dla funkcji $f(\cdot)$ w takim sensie, że istnieje $\varepsilon > 0$, dla którego zachodzi

$$f(x_k + \eta v) < f(x_k),$$

dla wszystkich $\eta \in (0, \varepsilon)$.

Dowód. Określmy finkcję $g(\cdot)$ jak poniżej

$$g(\eta) = f(x_k + \eta v).$$

Stąd

$$g'(\eta) = \nabla f(x_k + \eta v)v,$$

a następnie

$$g'(0) = \nabla f(x_k)v = -\nabla f(x_k)[Hf(x_k)]^{-1}\nabla f(x_k) < 0,$$

gdyż zgodnie z założeniem $\nabla f(x_k)$ jest różne od zera a hesjan jest dodatnio określony. Stąd wnioskujemy, że istnieje $\varepsilon > 0$, taki że $g(\eta) < g(0)$ dla wszystkich $\eta \in (0, \varepsilon)$, czyli

$$f(x_k + \eta v) < f(x_k).$$

□

17.3 Adaptacyjny dobór współczynnika uczenia

Zanim przejdziemy do bardziej wyrafinowanych metod minimalizacji funkcji, przyjrzyjmy się dwóm bardzo prostym modyfikacjom...

17.4 Metoda momentu

Do tej pory najmniejszej uwagi nie zwracaliśmy na bardzo istotny parametr – stałą uczenia η. Decyduje ona o ile w każdym kroku nauki zmieniamy wartości wag. Od jej właściwego doboru wiele zależy. Zbyt mała wartość powoduje, że postępy w nauce są niewielkie. Zbyt duża natomiast doprowadzić może do całkowitej rozbieżności procesu nauki lub jego haotycznego zachowania. Najprostrza metoda pozwalająca „uspokoić" zachowanie wag polega na dodaniu **członu momentu** do wyrażenia określającego przyrost wag. Pomysł polega na nadaniu każdej wadze pewnej bezwładności, czyli właśnie momentu, w wyniku czego zmienia się ona w kierunku wartości średniej zmian z poprzednich kroków. W takiej sytuacji nagłe niewielkie zmiany kierunku zostaną złagodzone, natomiast jeśli kierunek będzie ten sam, to proces ulegnie przyspieszeniu.

Stosując metodę momentu, wzór na zamianę wektora wag przyjmie postać

$$\Delta w_{ij}(t+1) = -\eta\frac{\partial E}{\partial w_{ij}} + \alpha\Delta w_{ij}(t),$$

gdzie $\alpha\Delta w_{ij}(t)$ to człon momentu, zaś α współczynnikiem momentu leżącym pomiędzy 0 i 1. Dla $\alpha = 0$ otrzymujemy zwykłą metodę aktualizacji wag. Pokażemy teraz na kilku przykładach znaczenie współczynników η i α.

Załóżmy, że powierzchnia błędu opisana jest wzorem

$$E(w) = 2w^2.$$

Wówczas

$$\Delta w = -\eta\frac{\partial E}{\partial w} = -\eta 4w.$$

Przyjmijmy ponadto $w = 8$.

Sytuacja 1 a) – cykl bez momentu: $\eta = 0.5$, $\alpha = 0.0$

Krok 1: $\Delta w = -0.5\ 4\ 8 = -16$, zatem $w = -8$.
Krok 2: $\Delta w = -0.5\ 4\ (-8) = 16$, zatem $w = 8$.
Krok 3: $\Delta w = -0.5\ 4\ 8 = -16$, zatem $w = -8$.
Krok 4: $\Delta w = -0.5\ 4\ (-8) = 16$, zatem $w = 8$.
Krok 5: $\Delta w = -0.5\ 4\ 8 = -16$, zatem $w = -8$.
Krok 6: $\Delta w = -0.5\ 4\ (-8) = 16$, zatem $w = 8$.

Sytuacja 1 b): – zbieżność z momentem$\eta = 0.5$, $\alpha = 0.2$

Krok 1: $\Delta w = -0.5\ 4\ 8 + 0.2\ 0.0 = -16$, zatem $w = -8$.
Krok 2: $\Delta w = -0.5\ 4\ (-8) + 0.2\ (-16) = 12.8$, zatem $w = 4.8$.
Krok 3: $\Delta w = -0.5\ 4\ 8 + 0.2\ 12.8 = -7.04$, zatem $w = -2.24$.
Krok 4: $\Delta w = -0.5\ 4\ (-8) + 0.2\ (-7.04) = 3.072$, zatem $w = 0.832$.
Krok 5: $\Delta w = -0.5\ 4\ 8 + 0.2\ 3.072 = -1.0496$, zatem $w = -0.2176$.
Krok 6: $\Delta w = -0.5\ 4\ (-8) + 0.2\ (-1.0496) = 0.22528$, zatem $w = 0.00768$.

Sytuacja 2 a) – zbieżność bez momentu: $\eta = 0.4$, $\alpha = 0.0$

Krok 1: $\Delta w = -0.4\ 4\ 8 = -12.8$, zatem $w = -4.8$.
Krok 2: $\Delta w = -0.4\ 4\ (-4.8) = 7.68$, zatem $w = 2.88$.
Krok 3: $\Delta w = -0.4\ 4\ 2.88 = -4.608$, zatem $w = -1.728$.
Krok 4: $\Delta w = -0.4\ 4\ (-1.728) = 2.7648$, zatem $w = 1.0368$.
Krok 5: $\Delta w = -0.4\ 4\ 2.7648 = -1.65888$, zatem $w = -0.62208$.
Krok 6: $\Delta w = -0.4\ 4\ (-1.65888) = 0.995328$, zatem $w = 0.373248$.

Sytuacja 2 b) – szybsza zbieżność z momentem: $\eta = 0.4$, $\alpha = 0.15$

Krok 1: $\Delta w = -0.4\ 4\ 8 + 0.15\ 0.0 = -12.8$, zatem $w = -4.8$.
Krok 2: $\Delta w = -0.4\ 4\ (-4.8) + 0.15\ (-12.8) = 5.76$, zatem $w = 0.96$.
Krok 3: $\Delta w = -0.4\ 4\ 0.96 + 0.15\ 5.76 = -0.672$, zatem $w = 0.288$.
Krok 4: $\Delta w = -0.4\ 4\ 0.288 + 0.15\ (-0.672) = -0.5616$, zatem $w = -0.2736$.
Krok 5: $\Delta w = -0.4\ 4\ (-0.2736) + 0.15\ (-0.672) = 0.35352$, zatem $w = 0.07992$.
Krok 6: $\Delta w = -0.4\ 4\ 0.07992 + 0.15\ 0.35352 = -0.074844$, zatem $w = 0.005076$.

Sytuacja 3 a) – zbieżne: $\eta = 0.1$, $\alpha = 0.0$

Krok 1: $\Delta w = -0.1\ 4\ 8 = -3.2$, zatem $w = 4.8$.
Krok 2: $\Delta w = -0.1\ 4\ 4.8 = -1.92$, zatem $w = 2.88$.
Krok 3: $\Delta w = -0.1\ 4\ 2.88 = -1.152$ zatem $w = 1.728$.
Krok 4: $\Delta w = -0.1\ 4\ 1.728 = -0.6912$, zatem $w = 1.0368$.
Krok 5: $\Delta w = -0.1\ 4\ 1.0368 = -0.41472$, zatem $w = 0.62208$.
Krok 6: $\Delta w = -0.1\ 4\ 0.62208 = -0.472392$, zatem $w = 0.373248$.

Sytuacja 3 b) – popsuta zbieżność: $\eta = 0.1$, $\alpha = 0.5$ **(poprawic na** $\alpha = 0.9$ **bo to dla większej liczby iteracji okazuje się zbieżne)**

Krok 1: $\Delta w = -0.1\ 4\ 8 + 0.5\ 0.0 = -3.2$, zatem $w = 4.8$.
Krok 2: $\Delta w = -0.1\ 4\ 4.8 + 0.5\ (-3.2) = -3.52$, zatem $w = 1.28$.
Krok 3: $\Delta w = -0.1\ 4\ 1.28 + 0.5\ (-3.52) = -2.272$, zatem $w = -0.992$.
Krok 4: $\Delta w = -0.1\ 4\ (-0.992) + 0.5\ (-2.272) = -0.7392$, zatem $w = -1.7312$.
Krok 5: $\Delta w = -0.1\ 4\ (-1.7312) + 0.5\ (-0.7392) = 0.32288$, zatem $w = -1.40832$.
Krok 6: $\Delta w = -0.1\ 4\ (-1.40832) + 0.5\ 0.32288 = 0.724768$, zatem $w = -0.683552$.

17.5 Metody zmiennej metryki

Niestety wzór (17.7), stanowiący istotę algorytmu Newtona, bardzo często okazuje się jedynie zależnością czysto teoretyczną. Wymaga on bowiem dodatniej określoności hesjanu w każdym kroku, co w ogólnym przypadku jest na ogół bardzo trudne do spełnienia. Z tego powodu metody, które faktycznie mogą być użyteczne rezygnują z dokładnego wyznaczania wartości hesjanu na rzecz jego przybliżenia.

Jedną z metod wzorowanych na metodzie Newtona, która przy tym nie oblicza dokładnej wartości hesjanu, jest metoda siecznych Broydena. Osoby zainteresowane wyprowadzeniem odsyłamy do [21], podając w tym miejscu jedynie sformułowanie metody.

Definicja 17.4 (Metoda Broydena rozwiązywania układów równań)**.** *Niech $g : R^n \to R^n$ będzie funkcją o ciągłych pochodnych cząstkowych pierwszego rzędu, $x_0 \in R^n$ oraz D_0 będzie macierzą wymiaru $n \times n$. Wówczas ciąg punktów $\{x_k\}_{k \in N}$ generowany prze metodę Broydena, dla równania $g(x) = 0$ i pewnego punktu początkowego x_0 określony jest przez formułę postaci*

$$x_{k+1} = x_k - D_k^{-1} g(x_k).$$

Jedna iteracja metody wymaga nastepujących kroków

1. *Rozwiązać równanie $D_k(x - x_k) = -g(_k)$ ze względu na zmienną x i wykonać podstawienie $x_{k+1} = x$.*
2. *Wykonać podstawienia*
   - $d_k = x_{k+1} - x_k$,
   - $y_k = g(x_{k+1}) - g(x_k)$.
   - *Dokonać aktualizacji macierzy D według wzoru*

$$D_{k+1} = D_k + \frac{(y_k - D_k d_k) \otimes d_k}{d_k d_k},$$

gdzie $\otimes$ oznacza iloczyn tensorowy.//uzup//(sprawdzic powyzej)

Poprawka dla macierzy D dobierana jest tak aby spełniona była zależność

$$D_{k+1}(x_{k+1} - x_k) = g(x_{k+1}) - g(x_k).$$

Zależność ta wynika z narzucenia warunku, aby krzywiznę funkcji $g(\cdot)$ aproksymować sieczną przechodzącą przez punkty $(x_{k-1}, g(x_{k-1}))$ oraz $(x_k, g(x_k))$ skąd pochodzi nazwa metody.

Znajomość metody Broydena pozwala na wyprowadzenie opartych o nią metod minimalizacji funkcji nazywanych metodami zmiennej metryki. Podamy w tym miejscu jedynie zależności rekurencyjne na aktualizację macierzy D odsyłając zainteresowanych do [21]. I tak według algorytmu

BFGS (Broyden–Hletcher–Goldfarb–Shanno)

$$D_{k+1} = D_k + \frac{y_k \otimes y_k}{d_k y_k} - \frac{D_k d_k \otimes D_k d_k}{d_k D_k d_k},$$

DFP (Davidon–Fletcher–Powell)

$$D_{k+1} = D_k + \frac{d_k \otimes d_k}{y_k d_k} - \frac{D_k y_k \otimes D_k y_k}{y_k D_k y_k},$$

gdzie $d_k = x_{k+1} - x_k$ oraz $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$. Zarówno w przypadku metody BFGS jak i DFP konieczne jest dobranie odpowiedniego kroku (minimalizacja kierunkowa) η_k w formule

$$x_{k+1} = x_k - \eta_k D_k^{-1}(\nabla f(x_k)).$$

Pomimo teoretycznej równoważności metod (teoretycznie macierz D_k otrzymana w wyniku stosowania obu metod jest dodatnio określona), metoda BFDS jest znacznie bardziej odporna na wszelkiego rodzaju błędy zaokrągleń i w praktyce to właśnie ona jest używana. Metoda ta jest obecnie uważana za jedną z najlepszych metod optymalizacji funkcji wielu zmiennych. Do jej wad zaliczyć można stosunkowo dużą złożoność obliczeniową związaną z koniecznością wyznaczenia elementów macierzy D_k a także duże zapotrzebowanie na pamięć związane z koniecznością przechowywania elementów macierzy D_k.

17.6 Algorytm gradientów sprzężonych

// tutu DOKOŃCZYĆ !!! // najszybszy spadek z minimalizacją kierunkową ale nowy kierunek jako kompromis pomiędzy kierunkiem gradientu a poprzednim kierunkiem szukania

$$v(t+1) = -\nabla E(t+1) + \beta v(t)$$

β dobrana jest tak aby nowy kierunek $v(t+1)$ jak najmniej psuł minimalizację osiągnętą przez poprzedni kierunek. Nowy kierunek powinien być taki, żeby nie zmieniał składowej gradientu wzdłuż poprzedniego kierunku, która właśnie została wyzerowana gdyż mamy warunek

$$\frac{\partial E(x + \eta v(t))}{\partial \eta} = v(t+1) \cdot v(t) = 0$$

Reguła Polaka-Ribiere'a

$$\beta = \frac{(\nabla E(t+1) - \nabla E(t)) \cdot \nabla E(t+1)}{(\nabla E(t+1))^2}$$

17.7 Algorytm Levenberga-Marquardta

W podrozdziale tym omówimy algorytm, którego istotną cechą jest odmienny sposób aktualizacji wag. Większość algorytmów uaktualnia wagi bezpośrednio po podaniu każdego wzorca uczącego (por. np. **??**//tutu//). Teoretycznie możliwe jest uaktualnianie wag dopiero po zaprezentowaniu wszystkich danych uczących w danym cyklu, ale praktyka wskazuje na znaczne pogorszenie procesu nauki w takim przypadku. O ile grupowe modyfikowanie wag w poprzednich algorytmach było opcjonalne, to omawiany tutaj algorytm Levenberga-Marquardta działa inaczej. Zakłada on (a wręcz wymaga) tylko grupową modyfikację wag. Założenie to znajduje swoje odzwierciedlenie w formułach na modyfikację wag.

O samym algorytmie można powiedzieć, że jest jednym z najefektywniejszych, gdyż łączy w sobie zbieżność algorytmu Newtona blisko minimum oraz metodę najszybszego

spadku, która bardzo szybko zmniejsza błąd gdy rozwiązanie jest dalekie. W praktyce rozwiązanie otrzymywane jest w kilku, kilkudziesięciu krokach. Niestety algorytm ten ma bardzo duże zapotrzebowanie na zasoby pamięciowe, co powoduje, że jest on praktycznie niemożliwy do wykorzystania w większości praktycznych zastosowań. Wymiar jednej z używanych macierzy wynosi $(o^* \cdot l^*) \times w^*$, gdzie o^* – ilość wyjść z sieci, l^* – ilość danych uczących, w^* – ilość wag. Rozpatrując dla przykładu zadanie kompresji obrazu (patrz ćwiczenie z rozdziału 28) mamy

- ilość wag wynosi 4160:
  - w warstwie pierwszej (wejściowej) są 32 neurony, a każdy jest połączony z 64 wejściami i jednym biasem, co daje łącznie $32 \cdot 64 + 32 = 2080$ wag,
  - w warstwie drugiej (wyjściowej) są 64 neurony, a każdy jest połączony z 32 neuronami z warstwy pierwszej i jednym biasem, co daje łącznie $64 \cdot 32 + 32 = 2080$ wag;
- ilość wyjść wynosi 64,
- ilosc danych uczacych wynosi 1024

a więc wymiar macierzy to ostatecznie $(64 \cdot 1024) \times 4192 = 65536 \times 4192$ co daje 274726912 liczb rzeczywistych które trzeba zapamiętać. Biorąc pod uwagę, że liczba rzeczywista zwykle zapamiętywana jest na czterech bajtach, otrzymujemy 1098907648 bajtów czyli ponad 1GB. A trzeba mieć na uwadze, że zadanie to jest jednym z prostszych jeśli chodzi o stopień złożoności sieci (ilość neuronów i połączeń między nimi) i wielkość zbioru uczącego.

W podrozdziale 17.1 przedstawiliśmy najprostsze chyba podejście do zagadnienia minimalizacji funkcji, czyli metodę najszybszego spadku. Przypomnijmy, że w metodzie tej ciąg punktów $\{x_k\}_{k \in N}$ mający być zbieżny do poszukiwanego minimum funkcji $f(\cdot)$, generowany jest według wzoru

$$x_{k+1} = x_k - \eta \nabla f(x_k). \tag{17.8}$$

Następnie, w podrozdziale 17.2 przedstawiliśmy metodę Newtona, która generuje ciąg $\{x_k\}_{k \in N}$ za pomocą formuły postaci

$$x_{k+1} = x_k - [Hf(x_k)]^{-1} \nabla f(x_k). \tag{17.9}$$

Przyglądając się równaniu (17.8) oraz (17.9) widzimy, że mają one taką samą formę a różnią się „czynnikiem" występującym przed $\nabla f(x_k)$. W zależności os sytuacji, korzystniejsze jest użycie albo pierwszej, albo drugiej metody. W przypadku, gdy funkcja zachowuje się liniowo w otoczeniu punktu x_k, wówczas nową wartość x_{k+1} lepiej obliczać przy pomocy (17.9). W przeciwnym razie lepiej skorzystać z wolno zbieżnej, ale generującej przynajmniej właściwe przybliżenie dla x_{k+1} formuły (17.8). Jak więc widać, obie metody uzupełniają się i intuicyjnie uzasadnione jest ich połączenie w taki sposób, aby zależnie od potrzeb móc w prosty sposób realizować jedną z nich. Stąd wynika następująca postać formuły generującej ciąg $\{x_k\}_{k \in N}$

$$x_{k+1} = x_k - ([Hf(x_k)] + \eta I)^{-1} \nabla f(x_k), \tag{17.10}$$

gdzie I jest macierzą jednostkową. Wzór (17.10) zapisany przy zastosowaniu wprowadzonej do tej pory konwencji nazewniczej dotyczącej sieci, przyjmuje postać

$$w(t+1) = w(t) - ([HE(w(t))] + \eta I)^{-1} \nabla E(w(t)). \tag{17.11}$$

Pokażemy teraz jak można wyznaczyć wartość wektora $\nabla E(\cdot)$ i macierzy $HE(\cdot)$. Przyjmijmy, że funkcja celu dana jest wzorem

$$E(w) = \frac{1}{2} \sum_{l \in L} \sum_{o \in O} (t_o^l - y_o^l)^2, \tag{17.12}$$

gdzie L jest zbiorem uczącym zaś O jest zbiorem neuronów warstwy wyjściowej. Równoważna postać, ale prostsza w zapisie to

$$E(w) = \frac{1}{2} \sum_{i \in A} (t_i - y_i)^2, \tag{17.13}$$

gdzie liczność zbioru A równa jest liczności zbioru $L \times O$ a indeks i przebiega kolejne elementy tego zbioru.

Oznaczmy przez $e_o^l = (t_o^l - y_o^l)$ błąd l-tej próbki w o-tym neuronie oraz przez $e(w(t))$ wektor błędu poszczególnych próbek na każdym neuronie w warstwie wyjściowej

$$e(w) = \begin{bmatrix} e_1^1 \\ \vdots \\ e_{o^*}^1 \\ e_1^2 \\ \vdots \\ e_{o^*}^2 \\ \vdots \\ e_1^{l^*} \\ \vdots \\ e_{o^*}^{l^*} \end{bmatrix} \tag{17.14}$$

gdzie l^* jest licznoscią zbioru L oraz o^* jest licznością zbioru O. Ponadto niech J będzie macierzą pierwszych pochodnych cząstkowych błędu każdej próbki dla poszczególnych

neuronów ostatniej warstwy e_l^o po wszystkich wagach w sieci:

$$J(w) = \begin{bmatrix} \frac{\partial e_1^1}{\partial w_1} & \cdots & \frac{\partial e_1^1}{\partial w_i} & \cdots & \frac{\partial e_1^1}{\partial w_{w^*}} \\ \vdots & & & & \\ \frac{\partial e_{o^*}^1}{\partial w_1} & \cdots & \frac{\partial e_{o^*}^1}{\partial w_i} & \cdots & \frac{\partial e_{o^*}^1}{\partial w_{w^*}} \\ \vdots & & & & \\ \frac{\partial e_1^l}{\partial w_1} & \cdots & \frac{\partial e_1^l}{\partial w_i} & \cdots & \frac{\partial e_1^l}{\partial w_{w^*}} \\ \vdots & & & & \\ \frac{\partial e_{o^*}^l}{\partial w_1} & \cdots & \frac{\partial e_{o^*}^l}{\partial w_i} & \cdots & \frac{\partial e_{o^*}^l}{\partial w_{w^*}} \\ \vdots & & & & \\ \frac{\partial e_1^{l^*}}{\partial w_1} & \cdots & \frac{\partial e_1^{l^*}}{\partial w_i} & \cdots & \frac{\partial e_1^{l^*}}{\partial w_{w^*}} \\ \vdots & & & & \\ \frac{\partial e_{o^*}^{l^*}}{\partial w_1} & \cdots & \frac{\partial e_{o^*}^{l^*}}{\partial w_i} & \cdots & \frac{\partial e_{o^*}^{l^*}}{\partial w_{w^*}} \end{bmatrix} \tag{17.15}$$

Zanim przejdziemy dalej, skomentujemy krótko postać wektora e i macierzy J. W celu czytelniejszego przedstawienia prezentowanych wzorów, przyjęto założenie, iż posługujemy się tylko jednym indeksem do wskazywania składowej wektora wagowego. Tak więc wykorzystywany w innych przypadkach zapis składowych z wieloma indeksami[2] należy wstępnie poddać procesowi „linearyzacji". Z punktu widzenia omawianej metody zupełnie nie ma znaczenia jak taka linearyzacja zostanie przeprowadzona.

W wektorze $e(w)$ poszczególne elementy są błędami kolejnych neuronów wyjściowych liczonych dla kolejnych elementów zbioru uczącego. Stąd wymiar tego wektora to $(o^* \cdot l^*) \times 1$.

Macierz $J(w)$ jest zbudowana w ten sposób, że tworzą ją l^* grup po o^* wierszy. Liczba wierszy w każdej grupie odpowiada liczbie wyjść z sieci. Pierwsza grupa jest liczona dla pierwszego elementu ze zbioru uczącego. Stąd pierwszy wiersz jest błędem jaki generuje pierwsze wyjście z sieci liczone dla pierwszego wzorca. Dla tego błędu są liczone pochodne po wszystkich wagach sieci. Dlatego liczba kolumn macierzy jest równa liczbie wag w całej sieci. Analogicznie, drugi wiersz jest wektorem pochodnych błędu po wszystkich wagach sieci jaki generuje drugie wyjście z sieci liczone dla pierwszego wzorca. Procedurę tę powtarzamy dla wszystkich elementów należących do zbioru uczącego. Stąd wymiar tej macierzy wynosi $(o^* \cdot l^*) \times w^*$.

Zauważmy jeszcze, że przyjmując uproszczoną notację, wzory (17.14) oraz (17.15) przyjmą prostszą postać (odpowiednio wzory (17.16) oraz (17.17))

$$e(w) = \begin{bmatrix} e_1 \\ \vdots \\ e_{a^*} \end{bmatrix} = \begin{bmatrix} t_1 - y_1 \\ \vdots \\ t_{a^*} - y_{a^*} \end{bmatrix} \tag{17.16}$$

[2]Zwykle są to trzy indeks: numer warstwy, numer neuronu w rozważanej warstwie, numer neuronu w warstwie poprzedniej.

oraz

$$J(w) = \begin{bmatrix} \frac{\partial e_1}{\partial w_1} & \cdots & \frac{\partial e_1}{\partial w_{w^*}} \\ \vdots & & \\ \frac{\partial e_{a^*}}{\partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*}} \end{bmatrix} \tag{17.17}$$

gdzie a^* jest licznością zbioru A. Przyjmując, że funkcja błędu wyraża się wzorem (17.13), ze wzoru na pochodną funkcji złożonej mamy

$$\nabla E(w) = \begin{bmatrix} \left[\frac{1}{2}\sum_{i\in A}(t_i - y_i)^2\right]'_{w_1} \\ \vdots \\ \left[\frac{1}{2}\sum_{i\in A}(t_i - y_i)^2\right]'_{w_{w^*}} \end{bmatrix} = \begin{bmatrix} \sum_{i\in A}(t_i - y_i)(t_i - y_i)'_{w_1} \\ \vdots \\ \sum_{i\in A}(t_i - y_i)(t_i - y_i)'_{w_{w^*}} \end{bmatrix} =$$

$$\begin{bmatrix} (t_1 - y_1)(t_1 - y_1)'_{w_1} + \cdots + (t_{a^*} - y_{a^*})(t_{a^*} - y_{a^*})'_{w_1} \\ \vdots \\ (t_1 - y_1)(t_1 - y_1)'_{w_{w^*}} + \cdots + (t_{a^*} - y_{a^*})(t_{a^*} - y_{a^*})'_{w_{w^*}} \end{bmatrix} =$$

$$\begin{bmatrix} (t_1 - y_1){e_1}'_{w_1} + \cdots + (t_{a^*} - y_{a^*}){e_{a^*}}'_{w_1} \\ \vdots \\ (t_1 - y_1){e_1}'_{w_{w^*}} + \cdots + (t_{a^*} - y_{a^*}){e_{a^*}}'_{w_{w^*}} \end{bmatrix} =$$

$$\begin{bmatrix} (t_1 - y_1)\frac{\partial e_1}{\partial w_1} + \cdots + (t_{a^*} - y_{a^*})\frac{\partial e_{a^*}}{\partial w_1} \\ \vdots \\ (t_1 - y_1)\frac{\partial e_1}{\partial w_{w^*}} + \cdots + (t_{a^*} - y_{a^*})\frac{\partial e_{a^*}}{\partial w_{w^*}} \end{bmatrix} =$$

$$\begin{bmatrix} \frac{\partial e_1}{\partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_1} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*}} \end{bmatrix} \cdot \begin{bmatrix} t_1 - y_1 \\ \vdots \\ t_{a^*} - y_{a^*} \end{bmatrix} = \begin{bmatrix} \frac{\partial e_1}{\partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_1} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*}} \end{bmatrix} \cdot \begin{bmatrix} e_1 \\ \vdots \\ e_{a^*} \end{bmatrix} = J^T(w)\cdot e(w)$$

Zatem gradient funkcji błędu obliczany jest według zależności

$$\nabla E(w) = J^T(w)e(w).$$

Wyznaczymy teraz analogiczną formułę na HE funkcji błędu.

$$\nabla(\nabla E(w)) = \left[(\nabla E(w))'_{w_1} \ldots (\nabla E(w))'_{w_{w^*}}\right] =$$

$$\left[\left[\left[\begin{bmatrix} \frac{\partial e_1}{\partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_1} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*}} \end{bmatrix} \cdot \begin{bmatrix} e_1 \\ \vdots \\ e_{a^*} \end{bmatrix}\right]'_{w_1} \cdots \left[\begin{bmatrix} \frac{\partial e_1}{\partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_1} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*}} \end{bmatrix} \cdot \begin{bmatrix} e_1 \\ \vdots \\ e_{a^*} \end{bmatrix}\right]'_{w_{w^*}}\right] =$$

$$\begin{bmatrix} \frac{\partial e_1}{\partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_1} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*}} \end{bmatrix}'_{w_1} \cdot \begin{bmatrix} e_1 \\ \vdots \\ e_{a^*} \end{bmatrix} + \begin{bmatrix} \frac{\partial e_1}{\partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_1} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*}} \end{bmatrix} \cdot \begin{bmatrix} e_1 \\ \vdots \\ e_{a^*} \end{bmatrix}'_{w_1}$$

$$\ldots$$

$$\left[\begin{array}{ccc} \frac{\partial e_1}{\partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_1} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*}} \end{array}\right]'_{w_{w^*}} \cdot \left[\begin{array}{c} e_1 \\ \vdots \\ e_{a^*} \end{array}\right] + \left[\begin{array}{ccc} \frac{\partial e_1}{\partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_1} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*}} \end{array}\right] \cdot \left[\begin{array}{c} e_1 \\ \vdots \\ e_{a^*} \end{array}\right]'_{w_{w^*}} =$$

$$\left[\left[\begin{array}{ccc} \frac{\partial e_1}{\partial w_1 \partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_1 \partial w_1} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*} \partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*} \partial w_1} \end{array}\right] \cdot \left[\begin{array}{c} e_1 \\ \vdots \\ e_{a^*} \end{array}\right] + \left[\begin{array}{ccc} \frac{\partial e_1}{\partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_1} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*}} \end{array}\right] \cdot \left[\begin{array}{c} \frac{\partial e_1}{\partial w_1} \\ \vdots \\ \frac{\partial e_{a^*}}{\partial w_1} \end{array}\right]\right.$$

$$\cdots$$

$$\left.\left[\begin{array}{ccc} \frac{\partial e_1}{\partial w_1 \partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_1 \partial w_{w^*}} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*} \partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*} \partial w_{w^*}} \end{array}\right] \cdot \left[\begin{array}{c} e_1 \\ \vdots \\ e_{a^*} \end{array}\right] + \left[\begin{array}{ccc} \frac{\partial e_1}{\partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_1} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*}} \end{array}\right] \cdot \left[\begin{array}{c} \frac{\partial e_1}{\partial w_{w^*}} \\ \vdots \\ \frac{\partial e_{a^*}}{\partial w_{w^*}} \end{array}\right]\right] =$$

$$\left[\left[\begin{array}{ccc} \frac{\partial e_1}{\partial w_1 \partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_1 \partial w_1} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*} \partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*} \partial w_1} \end{array}\right] \cdot e(w) + J^T(w) \cdot \left[\begin{array}{c} \frac{\partial e_1}{\partial w_1} \\ \vdots \\ \frac{\partial e_{a^*}}{\partial w_1} \end{array}\right]\right.$$

$$\cdots$$

$$\left.\left[\begin{array}{ccc} \frac{\partial e_1}{\partial w_1 \partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_1 \partial w_{w^*}} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*} \partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*} \partial w_{w^*}} \end{array}\right] \cdot e(w) + J^T(w) \cdot \left[\begin{array}{c} \frac{\partial e_1}{\partial w_{w^*}} \\ \vdots \\ \frac{\partial e_{a^*}}{\partial w_{w^*}} \end{array}\right]\right] =$$

$$\left[\left[\left[\left[\begin{array}{ccc} \frac{\partial e_1}{\partial w_1 \partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_1 \partial w_1} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*} \partial w_1} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*} \partial w_1} \end{array}\right] \cdots \left[\begin{array}{ccc} \frac{\partial e_1}{\partial w_1 \partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_1 \partial w_{w^*}} \\ \vdots & & \\ \frac{\partial e_1}{\partial w_{w^*} \partial w_{w^*}} & \cdots & \frac{\partial e_{a^*}}{\partial w_{w^*} \partial w_{w^*}} \end{array}\right]\right]\right] \cdot e(w) +$$

$$J^T(w) \cdot \left[\left[\left[\begin{array}{c} \frac{e_1}{\partial w_1} \\ \vdots \\ \frac{e_{a^*}}{\partial w_1} \end{array}\right] \cdots \left[\begin{array}{c} \frac{e_1}{\partial w_{w^*}} \\ \vdots \\ \frac{e_{a^*}}{\partial w_{w^*}} \end{array}\right]\right]\right] =$$

$$\nabla(\nabla e(w)) \cdot e(w) + J^T(w) \cdot J(w).$$

Zatem hesjan funkcji błędu obliczany jest według zależności

$$HE(w) = J^T(w) \cdot J(w) + \nabla(\nabla e(w)) \cdot e(w).$$

Oznaczmy przez $R(w)$ wyrażenie $\nabla(\nabla e(w)) \cdot e(w)$. Zauważmy, że jego wartość jest bliska macierzy zerowej w dwu przypadkach.

1. Gdy wszystkie otrzymane odpowiedzi y_i są bliski odpowiedziom oczekiwanym t_i, wówczas wektor $e(w)$ jest wektorem bliskim wektorowi zerowemu. Sytuacja taka występuje, gdy sieć jest już praktycznie nauczona.

2. Gdy funkcje $e_i(w)$ można aproksymować funkcjami liniowymi, wówczas wyrażenie $\nabla(\nabla e(w))$ bliskie jest macierzy zerowej (gdyż druga pochodna funkcji liniowej równa jest zeru).

Przyjmując założenie, że zachodzi co najmniej jeden z powyższych warunków, mamy

$$R(w(t)) \approx \mu I,$$

gdzie μ jest małą liczbą a więc zachodzi

$$R(w(t)) \approx 0,$$

i wzór (17.11) przyjmuje postać

$$w(t+1) = w(t) - [J^T(w)J(w) + \eta I]^{-1} J^T(w)e(w). \tag{17.18}$$

//tutu sprawdzic!!!//

Zauważmy, że identyczny wzór otrzymujemy, jeśli w rozważaniach wyjdziemy tylko od metody Newtona (bez łączenia jej z metodą najszybszego spadku, jak to było opisane na początku tego podrozdziału). Przyjmijmy zatem, że znana jest nam metoda Newtona generująca ciąg punktów $\{x_k\}_{k \in N}$ mający być zbieżny do poszukiwanego minimum funkcji $f(\cdot)$, co zapisane w stosowanej symbolice sieci neuronowych przyjmie postać

$$w(t+1) = w(t) - [HE(w(t))]^{-1} \nabla E(w(t)).$$

Jak wiemy z powyższych rozważań wzór ten przyjmuje postać

$$w(t+1) = w(t) - [J^T(w)J(w) + R(w)]^{-1} J^T(w)e(w),$$

gdzie wyrażenie $J^T(w)J(w) + R(w)$ pozwala obliczyć wartość hesjanu funkcji $E(w)$. Zakładając liniową aproksymację funkcji $e(w)$ wartość czynnika $R(w(t))$ bliska jest zeru a więc czynnik ten można albo pominąć albo zastąpić innym wyrażeniem. Pomijając go niejako skazujemy się na liniową aproksymację funkcji $e(w)$, która wcale nie musi być taka dobra i tym samym możemy wprowadzać dosyć duży błąd do obliczeń. Pozostawienie tego czynnika i jednocześnie przyjęcie, że $R(w(t)) \approx 0$ pozwala na zastąpienie go innym wyrażeniem W, znacznie prostszym w obliczeniach, ale mającym własność $W \approx 0$. Powoduje to, że informacje o aproksymacji nie wprowadzają tak dużego błędu jak w przypadku pominięcia tego czynnika a jenocześnie samo wyrażenie jest znacznie prostsze do obliczenia. Nie zapominajmy także o tym, że metoda Newtona wymaga aby macierz $HE(w(t))$ była dodatnio określona. Wyrażenie W daje nam możliwość na takie manipulowanie aby $J^T(w)J(w) + W$ było dostatecznie bliskim przybliżeniem $HE(w(t))$ a jednocześnie otrzymana w ten sposób macierz była dodatnio określona. Wygodnie jest przyjąć

$$W = \eta I,$$

gdzie η jest liczbą dodatnią bliską zeru a I to macierz jednostkowa.

Algorytm Levenberga - Marquardta

Przebieg algorytmu Levenberga - Marquardta:

1. Ustawić początkową wartość η oraz β (np. $\eta = 0.01$ oraz $\beta = 10$).
2. Obliczyć wektor błędu sieci dla wszystkich próbek (wzór (17.14)) dla bieżących wag.

3. Obliczyć miarę błędu sieci (wzór (17.12)) dla bieżących wag.
4. Zapamiętać wszystkie wagi sieci neuronowej.
5. Obliczyć macierz J daną wzorem (17.15).
6. Obliczyć nowe wagi sieci używając wzoru (17.18).
7. Obliczyć wektor błędu sieci dla wszystkich próbek (wzór (17.14)) dla nowych wag.
8. Obliczyć miarę błędu sieci (wzór (17.12)) dla nowych wag.
9. Jeżeli nowy błąd jest
   (a) mniejszy, niż obliczony w punkcie 3 to wykonać podstawienie $\eta = \frac{\eta}{\beta}$;
   (b) większy, niż obliczony w punkcie 3 to wykonać podstawienie $\eta = \eta\beta$ i przejść do punktu 6.
10. Jeśli spełniony jest warunek końca, to przejdź do kroku 12.
11. Zaakceptować zmianę wartości wag (nowe wagi stają się bieżącymi wagami) i przejść do kroku 2.
12. Koniec.

Warunek końca może być uznany za spełniony po wykonaniu zadanej ilości iteracji lub gdy błąd spadnie poniżej założonego minimum. W oryginalnym algorytmie Levenberga - Marquardta zamiast macierzy I stosuje się macierz $diag(J^T(w)J(w))$. Wykonanie kroku 9a oznacza, że założenie o liniowości minimalizowanej funkcji w otoczeniu punktu $w(t)$ okazało się trafne (wystarczająco dokładne) i można wpływ metody Newtona zwiększyć. Wykonanie kroku 9b oznacza, że założenie o liniowości minimalizowanej funkcji w otoczeniu punktu $w(t)$ okazało się nie dość ścisłe w związku z czym zwiększamy wpływ metody największego spadku.

Rozdział 18

Sieci samouczące

18.1 Wprowadzenie

Sieci samouczące (samoorganizujące) stanowią niejako dopełnienie sieci uczonych w oparciu o zadane wzorce. Nie wymagają one nadzoru na etapie uczenia. Same wykrywają istotne cechy powiązań między sygnałami, ucząc się ich, aby w fazie pracy skorzystać z tych właściwości do wyznaczenia rozwiązania dla wzorców nie uczestniczących w procesie samouczenia.

Podstawą samoorganizacji jest obserwacja, że globalne uporządkowanie sieci jest możliwe przez działania samoorganizujące prowadzone lokalnie w różnych punktach sieci, niezależnie od siebie. Sygnały wejściowe aktywują neurony w różnym stopniu, zależnym od częstości wystąpienia (aktywności) danego sygnału. Ponieważ zmiany zachodzące w sieci dążą do tego aby po ponownym podaniu na wejście tego samego sygnału, jej reakcja była silniejsza, dlatego im częściej dany sygnał występuje, tym większy ma wpływ na pobudzenie (aktywację) neuronów. Następuje przy tym naturalne zróżnicowanie wśród grup neuronów. Neurony lub nawet całe ich grupy specjalizują się w uaktywnianiu na ściśle określone wzorce. W tym przypadku można wręcz mówić zarówno o współpracy pomiędzy neuronami tej samej grupy, jak i konkurencji występującej zarówno w samej grupie jak i pomiędzy nimi. W samym procesie uczenia ważną rolę odgrywa „nadmiarowość" danych. Jest to tylko nadmiarowość pozorna. Owszem, pewne dane mogą występować wielokrotnie, ale jest to konieczne aby uczenie w ogóle było możliwe (konieczność wielokrotnego pokazywania wzorców).

Ogólnie mówiąc, zadania z powodzeniem rozwiązywane przez sieci samouczące są zadaniami wymagającymi **analizy skupień**. Analiza skupień (grupowanie) (ang. *data clustering*) jest pojęciem z zakresu eksploracji danych oraz uczenia się maszyn wywodzącym się z szerszego pojęcia jakim jest **klasyfikacja bezwzorcowa**. Jest ona metodą tzw. **klasyfikacji bez nadzoru** (ang. *unsupervised learning*), dokonującą grupowania elementów we względnie jednorodne klasy. Podstawą grupowania w większości algorytmów jest podobieństwo pomiędzy elementami wyrażane przy pomocy wybranej metryki (funkcji podobieństwa). Ponieważ grupowanie polega na wyodrębnianiu grup (klas, podzbiorów) podobnych (według wybranej metryki), zatem poprzez grupowanie można rozwiązać także problemy z gatunku odkrywania struktury w danych oraz dokonywanie uogólniania.

Grupowanie może być wykonywane w celu uzyskania różnorodnych informacji.

Klastrowanie. Inaczej właśnie grupowanie. Zadanie polega w tym przypadku na po-

dziale danych wejściowych na grupy o względnie najmniejszym stopniu zróżnicowania (według wybranego kryterium). Przedmiotem badania w tym przypadku jest grupa jako całość, np. jej rozkład w przestrzeni, położenie względem innych grup. Utworzone klastry pozwalają także na wydobycie innych informacji o danych (patrz następne punkty).

Kwantyzacja. Kwantyzacja to proces polegający na przypisaniu wartości analogowych do najbliższych im poziomów reprezentacji. Zwykle z kwantyzacją mamy do czynienia, gdy ciągłemu sygnałowi analogowemu należy przypisać pewną wartość dyskretną ze skończonego zbioru wartości.

// przyklad z sygnalem analogowym

//przyklad z przypisaniem punktom numerow

Ekstrakcja cech. Zwykle metody klasyfikacji bezwzorcowej stosujemy wówczas, gdy wiedza o naturze badanych sygnałów jest bardzo niewielka. Wówczas, grupując dane w klastry według zadanego kryterium, może się okazać, że wszystkie one (tzn. dane wchodzące w skład jednego klastra) posiadają jeszcze jakieś inne cechy wspólne, które do tej pory nie zostały wzięte przez nas pod uwagę. Tym samym proses grupowania może pozwolić na wyodrębnienie nowych cech opisujących dane.

Spośród mechanizmów samoorganizacji można wyróżnić dwie podstawowe klasy

- mechanizm współzawodnictwa pomiędzy neuronami oparty o ideę Kohonena;
- mechanizm samoorganizacji oparty na regule asocjacji Hebba.

Uwaga 18.1.

W literaturze zarówno światowej jak i polskiej konsekwentnie używany jest termin samoorganizacja. *Naszym zdaniem nie do końca odpowiada on temu co faktycznie się dzieje z siecią neuronową.*

Wszyscy są zgodni co do tego, że proces **nauki** *sieci neuronowej polega na zmianie wartości jej wag. I kiedy mówimy o uczeniu mamy na myśli zmienianie wag. Jeśli natomiast mówimy o ilości warstw, sposobie połączenia neuronów itp, to mówimy raczej o* **strukturze** *(***organizacji***) takiej sieci. Stąd termin* samoorganizująca *sugeruje, że przede wszystkich chodzi tutaj o zmianę organizacji (struktury) sieci neuronowej. Tym czasem jedyne co się zmienia w omawianych sieciach samoorganizujących, to wagi tych sieci.*

Oczywiście można powiedzieć, że np. sieci typu Kohonena, organizują neurony przez co otrzymujemy tak zwane mapy Kohonena. To jest jednak konsekwencja zmiany wag i ustalonej topologii sieci[1]*. Dlatego o wiele właściwsze wydaje nam się użycie terminu* samouczenie *choć stosować będziemy także i rozpowszechniony w tym kontekście termin* samoorganizacji.

[1]Przez topologię rozumiemy tutaj określenie który neuron z którym łączy się bezpośrednio. Czasem określenie to jest jeszcze bardziej dokładne, gdyż określa „stronę" sąsiedztwa: np. neuron n_1 łączy się z neuronem n_2 w taki sposób, że n_1 jest na górze n_2.

18.2 Sieci Kohonena

18.2.1 Mechanizm współzawodnictwa

Wśród sieci samoorganizujących istotną rolę odgrywają sieci, działające w oparciu o konkurencję między neuronami (sieci konkurencyjne, ang. *competitive network*). Najczęściej sieci tego typu są jednowarstwowe i skierowane do przodu. Każdy neuron sieci połączony jest ze wszystkimi składowymi n-wymiarowego wektora wejściowego. Zasadniczy mechanizm pracy sieci opiera się na typowaniu tak zwanego **zwycięzcy**. Zwycięzcą jest ten neuron $i \in \{1, \ldots, n\}$, dla którego (w danym cyklu pracy lub nauki) zachodzi nierówność:

$$d(x, w_{winner}) = d(x, w_i) = \min_{j \in \{1,\ldots,n\}} d(x, w_j), \tag{18.1}$$

gdzie $d(\cdot, \cdot)$ jest wybraną metryką (miarą podobieństwa). Oznacza to, że zwycięzcą jest neuron i-ty ($winner$), którego wagi w są najbliższe wektorowi wejściowemu x **w sensie wybranej metryki** d.

Uwaga 18.2.

Neuron winner jest neuronem, którego wagi w są najbliższe wektorowi wejściowemu x **w sensie wybranej metryki**. *Oznacza to, że w praktyce formuła (18.1) przyjmuje postać albo*

$$d(x, w_{winner}) = d(x, w_i) = \min_{j \in \{1,\ldots,n\}} d(x, w_j),$$

albo

$$d(x, w_{winner}) = d(x, w_i) = \max_{j \in \{1,\ldots,n\}} d(x, w_j).$$

Pierwsze z powyższych stosujemy, gdy metryka ma tą własność, że dla bliższych sobie wektorów zwraca mniejsze wartości (np. odległość euklidesowa). Drugie stosujemy, gdy metryka ma tą własność, że dla bliższych sobie wektorów zwraca wartość większą (np. iloczyn skalarny). Możliwa jest także sytuacja, gdy zarówno zamiast min *jak i* max *użyta zostanie inna funkcja pozwalająca wyłonić zwycięzcę. Nie zmienia to jednak ideii działania tego typu sieci.*

Zwykle odpowiedź jest obliczana w oparciu o neuron zwycięski[2] lub tylko on podlega procesowi uczenia[3]. Prowadzi to do „specjalizacj" neuronów i zwiększania ich stopnia reakcji po podaniu specyficznych wzorców wejściowych. Na ogół inny neuron reaguje na inne dane wejściowe a podczas uczenia stopień tej reakcji zwiększa się co pozwala zaobserwować pewnego rodzaju współzawodnictwo (konkurencję) między neuronami.

Sieci tego typu wprowadzone zostały przez Teuvo Kohonena ([8], [9], [10]) na początku lat 80-tych i stąd noszą taką nazwę. Przez ich twórcę nazwane zostały **samoorganizującym odwzorowaniem** (ang. *self-organizing map* – SOM) lub samoorganizującym odwzorowaniem cech (ang. *self-organizing feature map* – SOFM).

Zasadniczą ideą pracy sieci tego typu jest typowanie neuronu zwycięzcy według zależności (18.1). Wagi neuronu zwycięzcy są zmieniane w kierunku wektora x (tzn. wektor w zbliża się do wektora x) według zależności:

$$w_{winner}(t+1) = w_{winner}(t) + \eta[x - w_{winner}(t)],$$

[2] W sieciach tego typu nie ma raczej klasycznego obliczania wartości sygnału wyjściowego jak to ma miejsce w sieciach skierowanych do przodu.

[3] A jeśli nie tylko on, to najsilniej on.

gdzie η jest, jak zwykle, współczynnikiem uczenia. Zbliżanie się wektorów przekładać powinno się na wzmocnienie typowania jako *winner*-a danego neuronu przy ponownym podaniu jako obrazu wektora x.

Bardzo często ten elementarny mechanizm rozszerzany jest o występowanie tzw. sąsiedztwa. W najprostszym ujęciu, sąsiedztwo definiowane jest w przestrzeni sygnałów wejściowych jako miara odległości innych neuronów od neuronu zwycięzcy. Mając tak określone sąsiedztwo, zmianie możemy poddać wagi nie tylko neuronu zwycięzcy, ale wszytkich jego sąsiadów:

$$w_i(t+1) = w_i(t) + \eta N(i, winner)[x - w_i(t)], \quad i \in N(winner)$$

gdzie:

- $N(winner)$ – jest zbiorem wszystkich sąsiadów neuronu zwycięzcy (wraz z nim samym),
- $N(i, winner)$ – jest miarą sąsiedztwa (bliskości) neuronu i i neuronu $winner$. Miara ta zwykle jest tym mniejsza im dalej od neuronu $winner$ znajduje się neuron i a ponadto na ogół maleje z czasem. W efekcie w początkowym etapie nauki zmianiane są wagi zwycięzcy i pewnej ilości neuronów z jego otoczenia. Następnie otoczenie to ulega systematycznemu zawężaniu aż do momentu, gdy zbiór $N(winner)$ stanie się jednoelementowy (będzie zawierał tylko neuron $winner$).

Zachowanie takie krótko można sformułować jako wprowadzanie podobnych zmian w podobnych elementach i jak pokazują rysunki //tutu// ma on swoje praktyczne uzasadnienie.

Algorytm nauki, w którym adaptacji podlega tylko wektor wagowy zwycięzcy, nazywany jest algorytmem typu WTA (ang. *winner takes all*). Algorytm, w którym oprócz zwycięzcy uaktualniają swoje wagi również neurony z jego sąsiedztwa, nazywany jest algorytmem typu WTM (ang. *winner takes most*). Proces adaptacji wag w przypadku gdy określimy lub nie sąsiedztwo przedstawiają rysunki //tutu//.

18.2.2 Miary odległości

Istotnym elementem w przypadku sieci samouczących na zasadzie konkurencj, jest wybór metryki, czyli sposobu mierzenia odległości pomiędzy wektorem wejściowym a wektorem wagowym. Do najpopularniejszych należą:

- Miara euklidesowa

  $$d(x, w) = \|x - w\| = \sqrt{\sum_{i=1}^{n}(x_i - w_i)^2},$$

  gdzie x i w są wektorami n wymiarowymi, zaś x_i i w_i są ich i-tą składową.

- Iloczyn skalarny

  $$d(x, w) = x \cdot w = \|x\|\|w\| \cos(x, w).$$

- Miara według normy L_1 (Manhattan)

  $$d(x, w) = \sqrt{\sum_{i=1}^{n} |x_i - w_i|}.$$

- Miara według normy L_∞

$$d(x,w) = \max_{i\in\{1,\ldots,n\}} |x_i - w_i|.$$

Na rysunkach //tutu// zestawiono porównanie obszarów atrakcji poszczególnych neuronów przy różnych miarach odległości. W szczególności zaobserwować można, że:

- wykorzystując miarę euklidesową uzyskujemy podział przestrzeni na obszary dominacji neuronów równoważny mozaice Voronoia, w której przestrzeń wokół punktów centralnych jest strefą dominacji danego neuronu;
- zastosowanie iloczynu skalarnego, bez dodatkowych zabiegów najczęściej prowadzi do niespójnego podziału przestrzeni.

Przyjrzyjmy się prostemu przykładowi pokazującemu, dlaczego iloczyn skalarny stosowany do oryginalnych danych powodować może zachowanie sieci inne od oczekiwanego.
Przykład 18.1.

---

iloczyn skalarny w przestrzeni z roznie umieszczonymi punktami Przykład pokazujący konieczność „ingerencji" przyklad ze dla iloczynu skalarnego normalizacja jest konieczna (dla pozostalych miar nie) iloczyn skalarny w przestrzeni z rozkładem równomiernym dwuwymiarowych danych uczących Przykład pokazujący konieczność „ingerencji" – podział przestrzeni W istocie, iloczyn skalarny prowadzi do podziału przestrzeni jak na rysunku...

Środkiem zaradczym takiego, nieporządanego w przypadku wyboru iloczynu skalarnego jako metryki, działania sieci jest **normalizacja**. Normalizacja może być dokonanan na dwa sposoby.

- Przedefiniowanie wszystkich n składowych wektora x tak aby długość nowootrzymanego wektora x' wynosiła 1:

$$x' = \frac{x}{\|x\|},$$

  czyli

$$x'_i = \frac{x_i}{\sqrt{\sum_{j=1}^{n} x_j^2}}.$$

- Zwiększenie wymiaru przestrzeni o jeden przez taki wybór dodatkowej $n+1$ składowej, aby spełniona była równość:

$$\sum_{i=1}^{n+1} x_i^2 = 1.$$

  Zwykle wiąże się to z koniecznością wcześniejszego przeskalowania wektora $x \in R^n$, tak aby było możliwe spełnienie powyższej równości.

Normalizacji powinien być poddany wektor uczący x lub wektor wagowy w. Jeśli bowiem normalizacji poddamy:

- wektory uczące x wówczas wektory wagowe, jako dążące do nich, też staną się znormalizowane;

- wektor wagowy w wówczas o wartości iloczynu skalarnego dla wybranego wektora x decyduje wartość $\cos(x, w)$ (bo $\|w\|$ jest stała), przez co staje się wspólną miarą dla całej sieci.

Badania eksperymentalne potwierdzają potrzebę stsowania normalizacji w przypadku wektorów z małych przestrzeni. Znaczenie normalizacji maleje, gdy wymiar przestrzeni jest duży, tj. $n > 200$. I choć wykazano //tutu - gdzie?//, że proces normalizacji prowadzi zawsze do spójnego podziału przestrzeni danych to jednak normalizacja nie jest „cudownym lekiem" – przyjrzyjmy się kolejnemu przykładowi.

Przykład 18.2.

sss

Na rysunkach //tutu// zestawiono porównanie obszarów atrakcji poszczególnych neuronów przy różnych miarach odległości po normalizacji (porównaj z rysunkami //tutu//).

Uwaga 18.3.

Można by zapytać, dlaczego jako miarę wykorzystuje się iloczyn skalarny, skoro powoduje on różne problemy? Otóż, jeśli założyć normalizację, wówczas wartość iloczynu skalarnego jest tym większa im bardziej wektor w przypomina wektor x. W tym przypadku, przekłada się to na bardziej naturalną interpretację: zwycięzcą jest ten neuron, który zareaguje najsilniej na zadane pobudzenie. Rozpatrywanie po prostu odległości jako miary trudno uzasadniać biologicznie (bo czy neuron „mierzy" jakąś odległość?).

18.2.3 Mechanizm zmęczenia neuronów

Tak jak w przypadku sąsiedztwa także i ten mechanizm zapobiec ma preferowaniu pewnych neuronów i pozostawieniu innych niezmienionych przez proces uczenia. Chodzi o to, aby dać szansę na zwycięstwo także i neuronom mniej aktywnym.

Algorytm uczący liczbę zwycięstw poszczególnych neuronów wyrażaną przez współczynnik p nazywany **potencjałem**. Wartość tego współczynnika podlega modyfikacji po każdorazowej prezentacji wzorca uczącego, według zależności

$$p_i(t+1) = \begin{cases} p_i(t) + \frac{1}{n}, & \text{gdy} \quad i \neq winner \\ p_i(t) - p_{min}, & \text{gdy} \quad i = winner. \end{cases}$$

Współczynnik p_{min} oznacza minimalny potencjał zezwalający na udział we współzawodnictwie. Jeśli aktualna wartość potencjału spadnie poniżej p_{min}, wówczas dany neuron „odpoczywa". Opisany proces ma także miejsce w przypadku neuronów biologicznych. Każdy neuron po uaktywnieniu przechodzi w stan refrakcji („odpoczynku") kiedy to pozostaje niewrażliwy na pobudzenie. Jako maksymalną wartość p_{min} przyjmuje się 1. Doświadczalnie ustalona optymalna wartość tej stałej wynosi około 0.75. Zauważmy, że gdy $p_{min} = 0$ wówczas nie występuje zmęczenie neuronów i w każdej chwili współzawodniczą wszystkie neurony. Jeśli natomiast $p_{min} = 1$, wtedy neurony zwyciężają kolejno, gdyż w dowolnym momencie tylko jeden z nich gotowy jest do współzawodnictwa.

18.2.4 Miara organizacji sieci

SSS

18.2.5 Algorytmy uczenia

Zadaniem procedury nauki sieci samouczącej jest dobór takich wartości wektorów wag, które zminimalizują błąd popełniany przy aproksymacji wektora wejściowego x wektorem wagowym neuronu zwycięzcy. Zadanie to nosi także nazwę **kwantowania wektorowego** (ang. *vector quantization*). Numery neuronów zwyciężających podczas kolejnych prezentacji wektorów wejściowych stanowią niejako kod przypisany danemu wektorowi. Pozwala to np. na zastąpienie tych wektorów ich kodami przez co zmniejsza się objętość przetwarzanych danych (mechanizm taki wykorzystać można np. przy zadaniu kompresji – patrz rozdział 34).

Algorytm Kohonena

Algorytm Kohonena stanowi klasykę wśród algorytmów dla sieci samouczących. Do określenia zwycięzcy wykorzystuje się tutaj miarę euklidesową. Wyróżnia się dwa typy sąsiedztwa.

Sąsiedztwo prostokątne.

$$N(i, winner) = \begin{cases} 1, & \text{gdy} \quad d(i, winner) \leq \lambda \\ 0, & \text{gdy} \quad d(i, winner) > \lambda, \end{cases}$$

Sąsiedztwo gaussowskie.

$$N(i, winner) = \exp\left(-\frac{d^2(i, winner)}{2\lambda^2}\right)$$

W obu przypadkach parametr λ jest promieniem sąsiedztwa i maleje wraz z czasem uczenia. Zauważmy, że w przypadku sąsiedztwa gausowskiego stopień adaptacji jest zmienny w zależności od odległości od neuronu zwycięzcy. W przypadku sąsiedztwa prostokątnego albo neuron podlega adaptaci w takim samym stopniu jak neuron zwycięzca albo wcale nie podlega temu procesowi.

18.2.6 Funkcja błędu

Jak można zaobserwować na podanych wcześniej przykładach i wywnioskować z podanych informacji, sieci konkurencyjne dokonują podziału przestrzeni sygnałów wejściowych na grupy, nazywane także klastrami. Klaster tworzą te punkty z przestrzeni sygnałów wejściowych, które według pewnej miary podobieństwa są do siebie podobne. Zwykle miarą podobieństwa jest odległość od pewnego zadanego punktu, nazywanego centrum klastra (grupy). Popularnym kryterium pomiaru jakości klasteryzacji jest określenie następującej funkcji błędu

$$E = \sum_{i \in L} \|w_{winner} - x_i\|^2,$$

gdzie L jest zbiorem sygnałów wejściowych, x_i jest i-tym sygnałem wejściowym a w_{winner} jest neuronem zwycięzcą dla sygnału x_i. Pokażemy teraz, że procedura samoouczenia sieci konkurencyjnej prowadzi do minimalizacji funkcji E.

Twierdzenie 18.1. *Aktualizacja wektora wagowego według wzoru*

$$w_{winner}(t+1) = w_{winner}(t) + \eta[x - w_{winner}(t)],$$

prowadzi do minimalizacji funkcja błędu E dla i-tego wzorca

$$E_i = \frac{1}{2}\sum_{j=1}^{n} \|w_{winner,j} - x_{i,j}\|^2.$$

Dowód. Postępując analogicznie jak w przypadku reguły delta, obliczając pochodną funkcji E_i po zmiennej w_{ab}, otrzymujemy, formułę na zmianę wartości wag

$$\Delta w_{ab} = \begin{cases} -\eta\frac{\partial E_i}{\partial w_{ab}} & \text{gdy} \quad a = winner \\ \\ 0, & \text{gdy} \quad a \neq winner, \end{cases}$$

Wartość wyrażenia $\eta\frac{\partial E_i}{\partial w_{ab}}$ dla $a = winner$ wynosi:

$$\frac{\partial E_i}{\partial w_{ab}} = w_{ab} - x_{ib}.$$

Zatem

$$\Delta w_{ab} = -\eta(w_{ab} - x_{ib}) = \eta(x_{ib} - w_{ab}),$$

co jest odpowiednikiem wzoru

$$w_{winner}(t+1) = w_{winner}(t) + \eta[x - w_{winner}(t)],$$

dla b-tej składowej wektora w. Pokazaliśmy zatem, że metoda uczenia konkurencyjnego wynika z zastosowania gradientowej metody minimalizacji funkcji. □

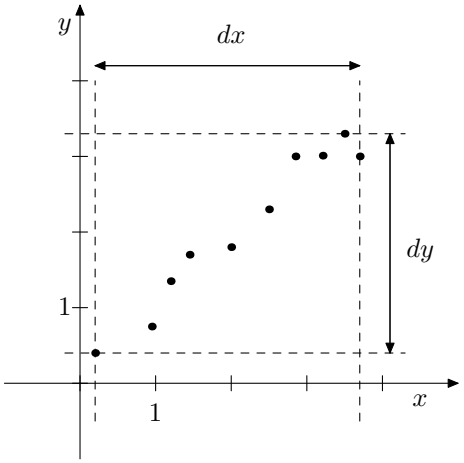
18.2.7 Sieć odwzorowań – mapy

Spróbujemy najpierw wyjaśnić dlaczego o sieciach Kohonena mówi się także **samoorganizujące mapy cech**. Mianowicie wokół neuronu zwycięzcy definiuje się topologiczne sąsiedztwo. Oznacza to, że określamy domyślne rozłożenie neuronów w przestrzeni, określając np. że neuron n_1 jest sąsiadem neuronu n_2. Co ważne, sąsiedztwo to zachodzi bez względu na faktyczne wartości wag. Może więc okazać się, że odległe w naszej przestrzeni neurony są sąsiadami[4]. Zauważmy, że w takim przypadku odległości pomiędzy neuronami określamy nie w odniesieniu do metryki zdefiniowanej w przestrzeni sygnałów wjściowych ale w odniesieniu do topologii samej sieci.

Sieci tego typu wprowadzone zostały przez Teuvo Kohonena ([8], [9], [10]) na początku lat 80-tych i stąd noszą taką nazwę (tj. sieci Kohonena). Przez ich twórcę natomiast nazwane zostały **samoorganizującym odwzorowaniem** (ang. *self-organizing map* – SOM) lub samoorganizującym odwzorowaniem cech (ang. *self-organizing feature map* – SOFM).

Odległości pomiędzy neuronami można wyznaczyć zarówno w odniesieniu do metryki zdefiniowanej w przestrzeni sygnałów wejściowych, jak również odpowiednio w odniesieniu topologii samej sieci. To drugie podejście pozwala określać sąsiedztwo danego neuronu w postaci zbioru neuronów powiązanych między sobą odpowiednimi relacjami topologicznymi. Na rysunku poniżej pokazano sąsiedztwo różnych rozmiarów //tutu uzupełnic//.

[4]To tak jak z ludźmi: nie przestajemy być sąsiadami tylko dlatego, że np. wakacje spędzamy na przeciwległych krańcach ziemi.



Rysunek 18.1: Przykładowe dane wraz z zaznaczonymi zakresami zmienności.

18.3 Uogólniona reguła Hebba

Działanie sieci zaprezentowanych w punkcie 18.2 może być rozumiane jako transformacja mapująca wektory wejściowe na pewien wektor binarny (z jedną jedynką odpowiadającą zwycięzcy i zerami na pozostałych liczbach) lub jako kwantyzacja tych wektorów (przypisywanie każdemu z nich pewnej liczby całkowitej z ograniczonego zbioru wartości). Wektory wagowe neuronów były aktualizowane w taki sposób, że mogły pełnić rolę prototypów dla tych danych wejściowych dla których zwyciężał ten sam neuron.

Samoorganizacja opisana w tym podrozdziale prowadzi do transformacji (obrotu) przestrzeni sygnałów wejściowych w taki sposób aby sygnały wyjściowe były ze sobą nieskorelowane a wariancja sygnałów wejściowych skupiona w kilku z nich. Przyjrzyjmy się następującemu przykładowi.

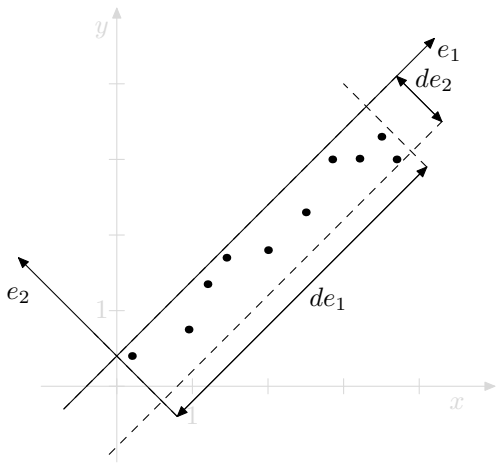
Rozważmy punkty o współrzędnych (x, y) przedstawione na rysunku 18.1. Łatwo można zauważyć, że x i y są ze sobą związane w taki sposób, że znając x względnie łatwo możemy „odgadnąć" wartość y (zależność bliska liniowej). Wprowadzając nowy układ współrzędnych (e_1, e_2) możemy w nim opisać dokładnie te same punkty (rysunek 18.2). Zauważmy, że

- nie ma żadnej zależności pomiędzy współrzędnymi punktów opisanych w nowym układzie;
- zmniejszona została wariancja (porównaj wielkości dx i dy z de_1 i de_2);
- można pominąc informację o drugiej współrzędnej w nowym układzie gdyż zmienia się ona nieznacznie w porównaniu z pierwszą współrzędną.

Mówiąc inaczej, dążymy do takiego przekształcenia sygnałów wejściowych aby wydobyć z nich istotne cechy wpływające na kształt sygnałów bez potrzeby rozważania wszystkich informacji.

18.3.1 Analiza głównych składowych

Intuicyjnie rozumiana analiza głównych składowych (ang. *PCA – principal component analysis*) opisana została w poprzednim podrozdziale. Bardziej formalnie, analiza głów-



Rysunek 18.2: Przykładowe dane wraz z zaznaczonymi zakresami zmienności.

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}
x	0.2	0.95	1.2	1.45	2.0	2.5	2.85	3.21	3.5	3.7
y	0.4	0.75	1.35	1.7	1.8	2.3	3.0	3.01	3.3	3.0

Tabela 18.2: Przykładowy zbiór danych.

nych składowych jest metodą statystyczną określającą przekształcenie liniowe

$$y = wx$$

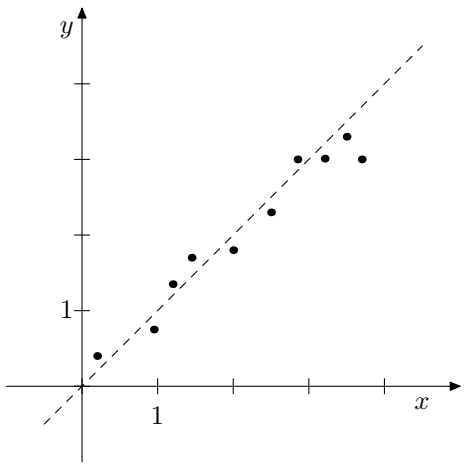
przekształcające wektor $x \in R^n$ opisujący pewien proces stochastyczny w wektor $y \in R^m$ przy pomocy macierzy $w \in R^{m \times n}$, przy czym $m \ll n$. Przekształcenie to powinno zostać wybrane w taki sposób aby zachowane zostały najważniejsze informacje dotyczące procesu. Odnosząc się do przykładu, można powiedzieć, że przekształcenie to zmniejsza ilość informacji zawartej we wzajemnie skorelowanych danych wejściowych i odwzorowuje go w zbiór statystycznie niezależnych składników według ich „ważności". Jest to więc pewnego rodzaju metoda kompresji.

Tak więc analiza głównych składowych jest procesem pozwalającym na wykrycie pewnych prawidłowości (istotnych różnic i podobieństw) we wzorcach. W teorii sieci neuronowych operacja analizy głównej składowej realizowana jest przez regułę uczenia Oji o czym powiemy w dalszej części tego rozdziału. Zanim to pokażemy, przybliżmy najpierw samą metodę analizy głównej składowej. Aby możliwa była dalsza lektura tego punktu należy z rozdziału 4 przypomnieć sobie informacje związane z:

- wartością oczekiwaną,
- wariancją,
- odchyleniem standardowym,
- korelacją,
- wektorami własnymi.

Analiza głównych składowych – przykład

Krok 1: pozyskanie danych. Występowanie tego kroku jest oczywiste. Jako jedyne słowo komentarza w tym miejscu nadmieńmy, że w przykładzie wykorzystamy dane pochodzące z przestrzeni dwuwymiarowej ze względu na łatwość ich prezentacji.



Rysunek 18.3: Rozmieszczenie przykładowych danych w przestrzeni R^2.

D ma postać:

$$D = \begin{bmatrix} 0.2 & 0.4 \\ 0.95 & 0.75 \\ 1.2 & 1.35 \\ 1.45 & 1.7 \\ 2.0 & 1.8 \\ 2.5 & 2.3 \\ 2.85 & 3.0 \\ 3.21 & 3.01 \\ 3.5 & 3.3 \\ 3.7 & 3.0 \end{bmatrix}$$

Krok 2: obliczenie średniej. W kroku tym liczymy średnią wartość dla każdej składowej (w naszym przypadku średnie obliczamy dla x i y oznaczając je przez $\bar{x}$ i $\bar{y}$) i odejmujemy ją od odpowiednich składowych otrzymując nowy zbiór $D' = \{[x'_1, y'_1], \ldots, [x'_{10}, y'_{10}]\}$:

$$\bar{x} = \frac{\sum_{i=1}^{10} x}{10}, \quad \bar{y} = \frac{\sum_{i=1}^{10} y}{10}$$

$$x'_i = x_i - \bar{x}, \quad y'_i = y_i - \bar{y}, \quad i = 1, \ldots, 10.$$

Przez D_{mean} oznaczamy wektor średnich:

$$D_{mean} = [\bar{x}, \bar{y}]$$

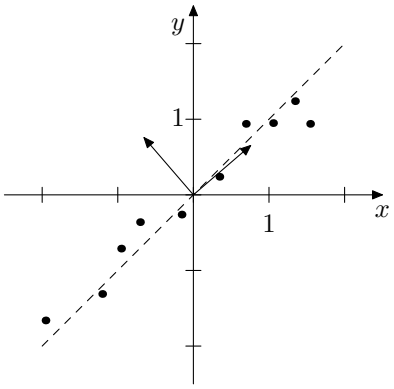
Dla danych z tabeli 18.2 otrzymujemy średnie: $\bar{x} = 2.156$ oraz $\bar{y} = 2.061$. Wartości zmiennych x' oraz y' zebrano w tabeli 18.3.

Krok 3: obliczenie macierzy kowariancji. Ponieważ dane pochodzą z przestrzeni dwuwymiarowej więc macierz kowariancji jest wymiaru 2×2.

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) \\ cov(y,x) & cov(y,y) \end{pmatrix},$$

	d'_1	d'_2	d'_3	d'_4	d'_5	d'_6	d'_7	d'_8	d'_9	d'_{10}
x'	-1.956	-1.206	-0.956	-0.706	-0.156	0.344	0.694	1.054	1.344	1.544
y'	-1.661	-1.311	-0.711	-0.361	-0.261	0.239	0.939	0.949	1.239	0.939

Tabela 18.3: Przykładowa zbiór danych po odjęciu średnich.



Rysunek 18.4: Rozkład elementów zbioru D' wraz z zaznaczonymi wektorami własnymi.

gdzie

$$\begin{aligned} cov(x,x) &= \frac{\sum_{i=1}^{10} x'_i x'_i}{9} \\ cov(x,y) &= \frac{\sum_{i=1}^{10} x'_i y'_i}{9} \\ cov(y,x) &= \frac{\sum_{i=1}^{10} y'_i x'_i}{9} \\ cov(y,y) &= \frac{\sum_{i=1}^{10} y'_i y'_i}{9} \end{aligned}$$

Dla danych z tabeli 18.3 otrzymujemy:

$$C = \begin{pmatrix} 1.402026 & 1.183826 \\ 1.183826 & 1.048654 \end{pmatrix}$$

Krok 4: obliczenie wektorów i wartości własnych macierzy kowariancji. Ponieważ macierz kowariancji jest prostokątna, więc możemy obliczyć jej wektory i wartości własne (odpowiednio e_1, e_2 oraz λ_1, λ_2).

$$e_1 = [0.757501, 0.652834], \quad \lambda_1 = 2.42228,$$

$$e_2 = [-0.652834, 0.757501], \quad \lambda_2 = 0.0284014.$$

Na rysunku 18.4 pokazano rozkład elementów zbioru D' wraz z zaznaczonymi wektorami własnymi. Wektory własne wyznaczają niejako „kierunki" zmian danych. Dalsze etapy dokonują takiej transformacji danych, aby można je opisać przy pomocy wyznaczonych wektorów.

Krok 5: wybór składowych. Przyglądając się wektorom i wartościom własnym obliczonym w kroku 4 zauważyć można, że wektor własny, dla którego obserwjemy największą zmienność danych ma największą wartość własną. Wektor ten nazywamy **składnikiem głównym** (ang. *principle component*). I nie jest to przypadek. W ogólności, po obliczeniu wektorów własnych i odpowiadających im wartości

własnych, porządkujemy je według malejących wartości własnych. W ten sposób ustawiamy składowe od najważniejszej do najmniej istotnej. Im wartość własna jest mniejsza, tym mniej istotna jest dana składowa. Wybierając teraz tylko niektóre ze składowych – te najważniejsze – możemy oryginalne dane przekształcić w nowy zbiór (złożony z elementów o mniejszym wymiarze), który będzie opisywał tylko te – najważniejsze naszym zdaniem cechy – które wybraliśmy uprzednio. Wybrane w ten sposób wektory własne tworzą macierz nazywaną wektorem (macierzą) cech (ang. *feature vector*).

W naszym przypadku możemy zbudować trzy takie wektory:

$$f_0 = [e_1, e_2], \quad f_1 = [e_1], \quad f_2 = [e_2].$$

Jakie konsekwencje pociąga za sobą wybór każdej z nich, pokażemy w następnym kroku.

Krok 6: transformacja danych. W kroku tym dokonujemy transformacji danych według składowych (wektorów własnych) które wybraliśmy:

$$D_{fin} = f^T \times D'^T,$$

gdzie D_{fin} to dane po transformacji, f to wybrany wektor cech. Na rysunku 18.5 przedstawiono zbiór danych D' i jego postać po transformacji w zależności od wybranego wektora cech. W ten oto sposób wyraziliśmy oryginalne dane w terminach najważniejszych składowych.

Krok 7: odzyskanie danych oryginalnych. Krok 6 zasadniczo zakończył proces analizy głównych składowych i wyrażania danych w ich terminach. Jak można zauważyć, wybierając najważniejsze składowe dokonujemy kompresji danych, bo zmniejszamy wymiar docelowej przestrzeni zachowując informację o najważniejszych cechach. Dane oryginalne możemy przywrócić posługując się formułą

$$D^T_{dec} = (f \times D^T_{fin}) + D^T_{mean}$$

Pamiętać musimy tylko o tym, że redukcja danych na etapie „kompresji" jest nieodwracalna i nie ma możliwości przywrócenia oryginalnych danych w 100% jeśli pominięta została choćby jedna składowa. Na rysunku 18.6 przedstawiono oryginalny zbiór danych i jego postać po „dekompresji" w zależności od wybranego uprzednio wektora cech.

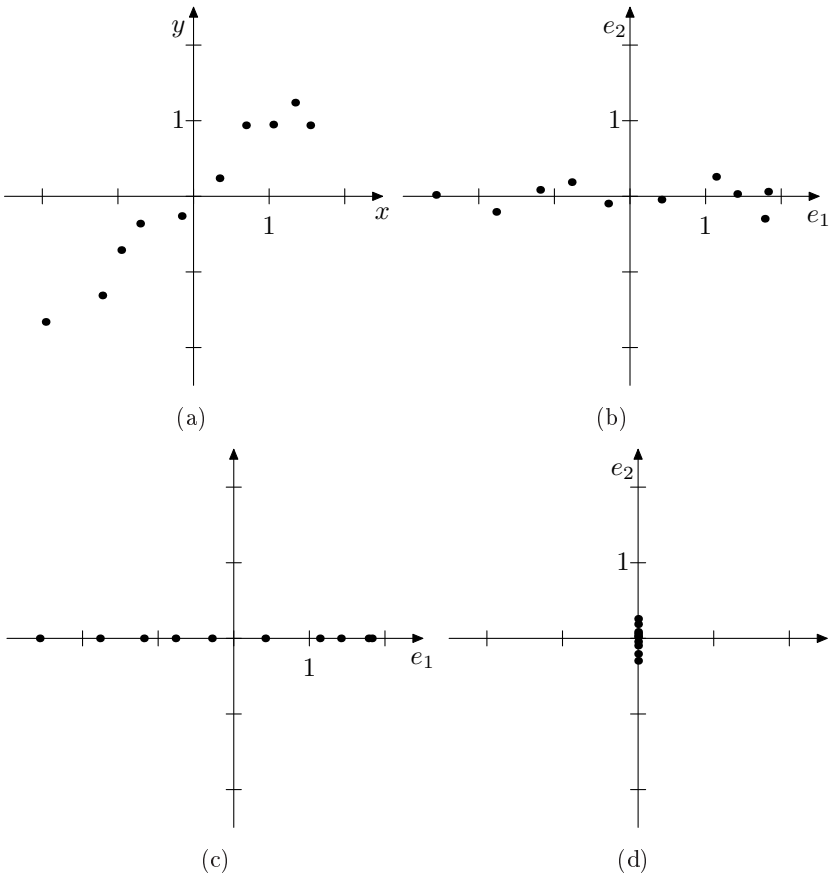
18.3.2 Siec samoorganizująca się w oparciu o uogólnioną regułę Hebba

Rozważamy sieć jednowarstwową o n wejściach, m wyjściach, z liniowymi modelami neuronu (z liniową funkcją aktywacji) uczoną metodą Hebba.

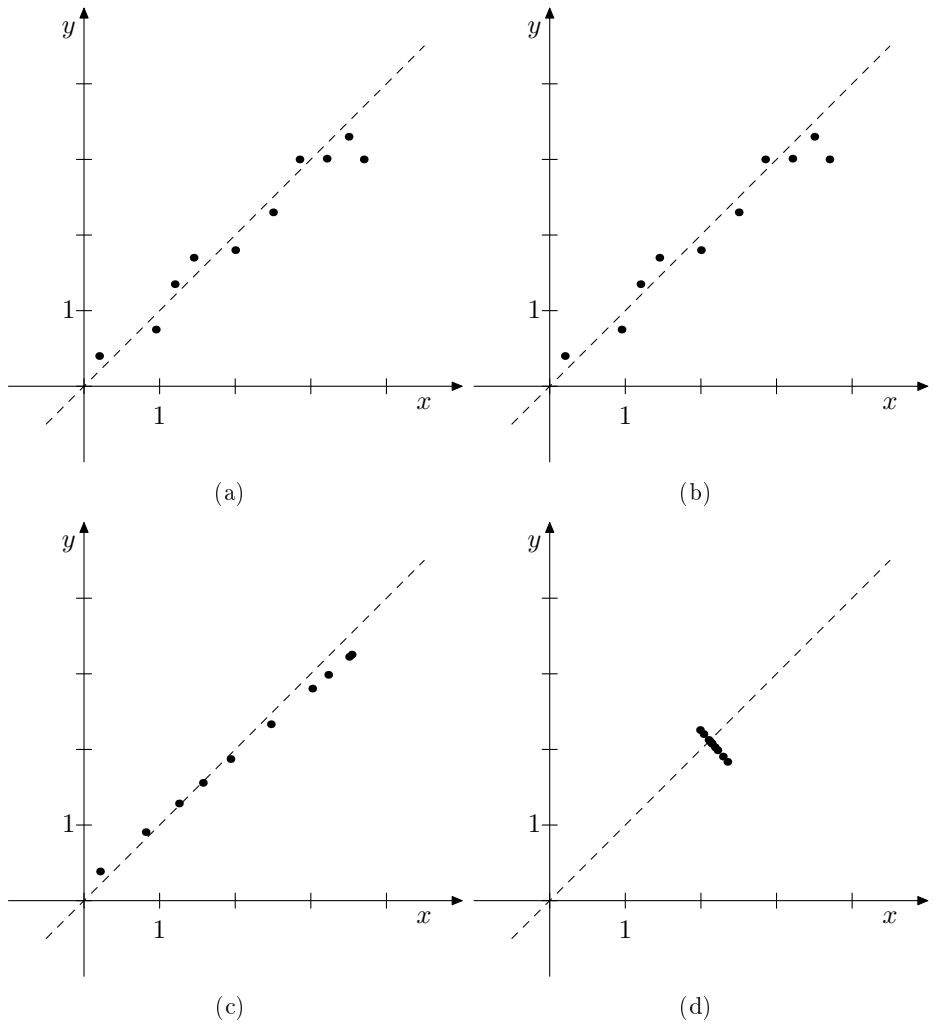
Definicja 18.1 (Metoda Hebba)**.** *Metoda Hebba wykorzystuje wyniki obserwacji neurobiologicznych. Zgodnie z nimi waga powiązań pomiędzy dwoma nauronami wzrasta przy jednoczesnym wzajemnym pobudzeniu obu neuronów. W przeciwnym przypadku maleje.*

Zastosowanie tej obserwacji prowadzi do wzoru na aktualizację wag postaci:

$$\Delta w_{ij} = \eta x_j y_i,$$



Rysunek 18.5: Zbiór danych D' i jego postać po transformacji w zależności od wybranego wektora cech: (a) – dane tworzące zbiór D', (b) – zbiór D' po transformacji przy pomocy wektora f_0, (c) – zbiór D' po transformacji przy pomocy wektora f_1, (d) – zbiór D' po transformacji przy pomocy wektora f_2.



Rysunek 18.6: Zbiór danych D i jego postać po transformacji i „dekompresji" w zależności od wybranego wektora cech: (a) – dane tworzące zbiór D, (b) – dla wektora f_0, (c) – dla wektora f_1, (d) – dla wektora f_2.

gdzie x_j jest sygnałem wejściowym przechodzącym przez połączenie o wadze w_{ij} zaś y_i jest wyjściem z i-tego neuronu. A zatem metoda ta może być wykorzystana do uczenia bez nauczyciela. Jeśli jednak znane są prawidłowe odpowiedzi, wówczas powyższy wzór przyjmuje postać:

$$\Delta w_{ij} = \eta x_j t_i,$$

gdzie t_i jest oczekiwaną odpowiedzią i-tego neuronu. Taki sposób modyfikacji wartości wag może prowadzić do ich nadmiernego wzrostu w przypadku wielokrotnego prezentowania takiego samego wzorca. Stąd też zwykle stosuje się pewne modyfikacje tej metody, np.:

- *dodając współczynnik zapominania:*

  $$\Delta w_{ij} = \eta x_j y_i + \gamma w_{ij} y_i,$$

  gdzie γ jest współczynnikiem zapominania;

- *wprowadzając normalizacje wektora wagowego:*

  $$w_{ij}(t+1) = \frac{w_{ij}(t) + \eta x_j y_i}{\|w_i(t) + \eta x y_i\|}, \tag{18.2}$$

  gdzie $w_i(t) = [w_{i1}(t), \ldots, w_{in}(t)]$ oraz $xy_i(t) = [x_1 y_i, \ldots, x_n y_i]$.

Traktując normę jako operator zwracający długość wektora[5], oznaczmy go przez L:

$$L(x) = \|x\|.$$

Korzystając ze wzoru Taylora (definicja 4.13 na stronie 46) możemy teraz rozwinąć funkcje $L'(\eta) = L(w_i(t) + \eta x y_i)$ dla $i = 1, \ldots, m$ w punkcie $\eta = 0$ (przyjmując w definicji 4.13: $a = 0$ i dowolne $b > 0$; funkcja $L(\eta)$ jest klasy C^∞) otrzymujemy:

$$L'(\eta) = L'(0) + \frac{\eta}{1!}L'^1(0) + \frac{(\eta)^2}{2!}L'^2(0) + \cdots + \frac{(x)^n}{n!}L'^n(0) + R_n(x, 0)$$

lub równoważnie:

$$L'(\eta) = L'(0) + \frac{\eta}{1!}L'^1(0) + O(\eta^2).$$

Z ostatniego i ze zmodyfikowanej reguły Hebba (18.2) otrzymujemy:

$$\begin{aligned}
w_{ij}(t+1) &= \frac{w_{ij}(t) + \eta x_j y_i}{\|w_i(t) + \eta x y_i\|} \\
&= \frac{w_{ij}(t) + \eta x_j y_i}{L(w_i(t) + \eta x y_i)} \\
&= \frac{w_{ij}(t) + \eta x_j y_i}{L'(\eta)} \\
&= \frac{w_{ij}(t) + \eta x_j y_i}{L'(0) + \frac{\eta}{1!}L'^1(0) + O(\eta^2)} \\
&= \frac{w_{ij}(t) + \eta x_j y_i}{L(w(t)) + \eta L'^1(0) + O(\eta^2)} \\
&= \frac{w_{ij}(t) + \eta x_j y_i}{1 + \eta L'^1(0) + O(\eta^2)}.
\end{aligned}$$

[5]Ponieważ interesuje nas długość, więc używać będziemy normy euklidesowej.

Zauważmy teraz, że dla małych $\eta > 0$ zachodzi następujący związek:

$$\frac{1}{1+\eta a} = 1 - \eta a + O(\eta^2).$$

Stąd:

$$\begin{aligned} w_{ij}(t+1) &= \frac{w_{ij}(t) + \eta x_j y_i}{1 + \eta L'^1(0) + O(\eta^2)} \\ &= (w_{ij}(t) + \eta x_j y_i)\frac{1}{1 + \eta L'^1(0) + O(\eta^2)} \\ &= (w_{ij}(t) + \eta x_j y_i)(1 - \eta L'^1(0) + O(\eta^2)) \end{aligned}$$

co po odrzuceniu reszty prowadzi do wyrażenia[6]

$$w_{ij}(t+1) = (w_{ij}(t) + \eta x_j y_i)(1 - \eta L'^1(0)).$$

Policzymy teraz (pierwszą) pochodną funkcji L' w punkcie $\eta = 0$:

$$\begin{aligned} L'^1(\eta)|_{\eta=0} &= L(w_i(t) + \eta x y_i)^1|_{\eta=0} \\ &= \left(\sqrt{\sum_{q=1}^{n}(w_{iq} + \eta x_q y_i)^2}\right)^1\Bigg|_{\eta=0} \\ &= \frac{1}{2}\frac{1}{\left(\sqrt{\sum_{q=1}^{n}(w_{iq} + \eta x_q y_i)^2}\right)} \cdot \left(\sum_{q=1}^{n}(w_{iq} + \eta x_q y_i)^2\right)^1\Bigg|_{\eta=0} \\ &= \frac{1}{2}\frac{1}{\left(\sqrt{\sum_{q=1}^{n}(w_{iq} + \eta x_q y_i)^2}\right)} \cdot \left(\sum_{q=1}^{n} 2(w_{iq} + \eta x_q y_i) \cdot (x_q y_i)\right)\Bigg|_{\eta=0} \\ &= \frac{1}{2}\frac{1}{\left(\sqrt{\sum_{q=1}^{n}(w_{iq})^2}\right)} \cdot \left(\sum_{q=1}^{n} 2(w_{iq}) \cdot (x_q y_i)\right) \end{aligned}$$

Uwzględniając teraz, że[7]:

$$\sqrt{\sum_{q=1}^{n}(w_{iq})^2} = \|w_i\| = 1$$

oraz

$$\sum_{q=1}^{n} w_{iq} x_q = y_i$$

otrzymujemy ostatecznie:

$$L'^1(\eta)|_{\eta=0} = y_i^2.$$

[6] W tym przypadku znak *równości* należy bardziej rozumieć jako *staje się* a nie symbol oznaczający takie same wartości wyrażeń po lewej i prawej stronie.

[7] Związek ten jest wynikiem wprowadzenia normalizacji w regule Hebba.

A zatem:

$$\begin{aligned} w_{ij}(t+1) &= (w_{ij}(t) + \eta x_j y_i)(1 - \eta L'^1(0)) \\ &= (w_{ij}(t) + \eta x_j y_i)(1 - \eta y_i^2) \\ &= w_{ij}(t) - \eta w_{ij} y_i^2 + \eta x_j y_i - \eta^2 x_j y_i^3 \end{aligned}$$

co po ograniczeniu się do pierwszych trzech czynników pozwala określić wzór na zmianę wartości wag:

$$w_{ij}(t+1) = w_{ij}(t) + \eta y_i (x_j - w_{ij} y_i).$$

Wzór ten nosi nazwę **wzoru (reguły) Oji**.

Przyjmijmy teraz, że nasza sieć składa się z tylko jednego neuronu (czyli $m = 1$). Dwa alternatywne zdania na temat procesu uczenia są prawdziwe:

- Uczenie zmierza do minimalizacji funkcji błędu.
- Uczenie zmierza do ustabilizowania się wektorów wagowych.

Pierwsze ze zdań pojawiało się już dosyć często, więc przyjrzyjmy się drugiemu z nich.

Drugie ze zdań szczególnie mocno widoczne jest w przypadku sieci samouczących. Oto bowiem rozpoczynając naukę od pewnych losowych wartości wag, dążymy do takiego stanu aż wektor wagowy przestanie się zmieniać. Oznaczać to będzie, że sieć „jest już nauczona" a raczej, że „się nauczyła". Tak więc po dostatecznie długim procesie uczenia wektor wag pozostawałby w sąsiedztwie pewnego punktu – punktu równowagi. Oczywiście gdyby proces nauki był nadal kontynuowany to, proporcjonalnie do współczynnika uczenia, wektor ten zmieniałby nieznacznie swoją wartość, pozostając jednak cały czas wokół swojego punktu równowagi. Tak więc w punkcie równowagi wszelkie zmiany wag, w sensie wartości średniej, równają się zero.

Niech teraz x oznacza n-wymiarowy wektor sygnałów wejściowych, taki że $E(x) = 0$. Zauważm, że krok 2 w przedstawionym wcześniej przykładzie realizował właśnie ten postulat – przekształcał dane w taki sposób, aby ich wartość oczekiwana (średnia) wynosiła 0. Dalej, niech R będzie określone według wzoru

$$R = E[(x - Ex)(x - Ex)]. \tag{18.3}$$

Zauważmy, że założenie to realizowane było przez krok 3 wspomnianego algorytmu, w którym to liczyliśmy macierz kowariancji postaci

$$cov(X, Y) = E[(X - EX)(Y - EY)],$$

co przy przyjęciu założenia, że X jest pierwszą składową wektora x a Y – drugą prowadzi do (18.3). Wówczas zachodzi następujące twierdzenie:

Twierdzenie 18.2. *W wyniku modyfikacji wag według reguły Oji wektor wagowy dąży do takiego wektora w, dla którego:*

- *Wektor w leży w kierunku maksymalnego wektora własnego*[8] *macierzy R.*
- $\|w\| = 1$.

[8] Pisząc „maksymalny wektor własny" mamy na myśli ten z wektorów własnych, któremu odpowiada największa wartość własna.

Dowód. □

Poczynione do tej pory założenia wraz z powyższym twierdzeniem pozwalają znaleźć pierwszą główną składową. Przyjrzyjmy się teraz jak można wyznaczyć kolejne składowe. Rozważmy pewien sygnał x', który możemy wyrazić w przestrzeni rozpiętej na wektorach własnych macierzy R jako:

$$x' = \sum_{i=1}^{n} \alpha'_i e_i.$$

Odejmując teraz od sygnału x' składową na kierunku e_1 i oznaczając taką różnicę przez x''

$$x'' = x' - \alpha'_1 e_1 = \sum_{i=1}^{n} \alpha''_i e_i$$

mamy pewność, że poddanie sygnału x'' procesowi analizy głównej składowej spowoduje, że współczynnik α''_1 będzie równy 0. Stanie się tak, gdyż w sygnale x'' nie będzie już żadnych informacji związanych ze składową e_1, która właśnie została odjęta. A zatem dodając do sieci kolejny (drugi) neuron i ucząc go na sygnale x'' znajdziemy drugą składową. Kontynuując to postępowanie, można zbudować sieć znajdującą wszystkie lub określoną ilość składowych. Dokładny opis algorytmu zamieszczono w ćwiczeniu z rozdziału 33.

Rozdział 19

Sieci neuronowe o radialnych funkcjach bazowych

19.1 Matematyczne podstawy

Sieci z radialnymi funkcjami aktywacji stanowią w pewnym sensie uzupełnienie sieci sigmoidalnych. Pojedyńczy neuron sigmoidalny reprezentuje w przestrzeni wielowymiarowej S hiperpłaszczyznę dzielącą tę przestrzeń na dwie części (klasy) P oraz N w taki sposób aby zachodziły warunki 16.11 oraz 16.12 z rozdziału 16:

$$\sum_{i=1}^{n} w_i x_i^P \geq w_{n+1},$$

$$\sum_{i=1}^{n} w_i x_i^N < w_{n+1}.$$

Konsekwencją tego są trudności w przypadku rozwiązywania zadań gdzie punkty tworzą wyraźnie odróżniające się i wymieszane skupiska punków (porównaj rysunek //uzup//). W tego typu przypadkach najlepiej jeśli można wykorzystać narzędzie dokonujące np. „kołowego" podziału przestrzeni wokół pewnego punktu centralnego[1]. Właśnie takie odwzorowanie realizują sieci radialne.

Definicja 19.1. Radialna funkcja bazowa *(ang. radial basis function - RBF), jest funkcją, której wartość zależy zwykle wyłącznie od odległości argumentu x od określonego punktu c w taki sposób, że zachodzi*

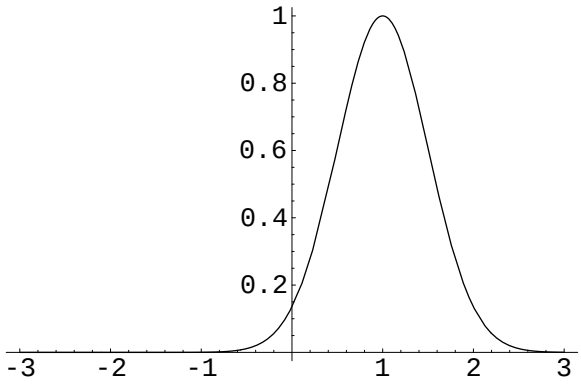
$$\phi(x, c) = \phi(\|x - c\|).$$

Punkt c nazywany jest **centrum funkcji radialnej** *i na ogół jest parametrem a nie zmienną, dlatego czasem pisze się*

$$\phi(x) = \phi(\|x - c\|).$$

Przykładami najczęściej wykorzystywanych funkcji radialnych są

[1] Termin „kołowy" nie musi być koniecznie potraktowany dosłownie, może to być na elipsa, lub jakiś inny kształt, ale skupiony wokół pewnego punktu centralnego.



Rysunek 19.1: Wykres funkcji $\phi(x) = \exp(-2|x-1|^2)$.

gaussowska (ang. *gaussian*)

- $$\phi(x) = \exp(-\beta\|x-c\|^2), \quad \beta > 0,$$
- $$\phi(x) = \exp\left(-\frac{\|x-c\|^2}{r^2}\right) = \exp\left(-\frac{\|x-c\|^2}{2\sigma^2}\right), \quad \sigma > 0,$$

wielokwadratowa (ang. *multiquadratic*)

$$\phi(x) = \sqrt{\|x-c\|^2 + r^2},$$

wielomianowa (ang. *polynomial*)

$$\phi(x) = \|x-c\|^2 + r^2,$$

liniowa

$$\phi(x) = \|x-c\|,$$

cienkiej płytki (ang. *thin-plate spline*)

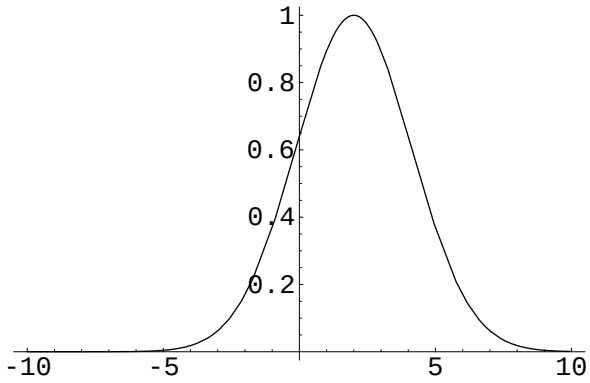
$$\phi(x) = \|x-c\|^2 \log(\|x-c\|).$$

Wprowadźmy teraz definicję analogiczną do separowalności liniowej, ale dotyczącą neuronów radialnych.

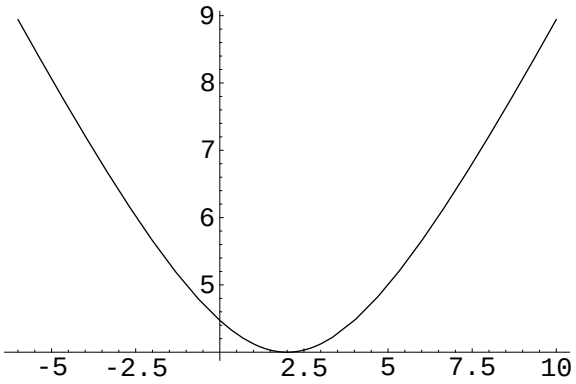
Definicja 19.2 (f-separowalność)**.** *Niech $f = [f_1, \ldots, f_k]$ będzie funkcją działającą z przestrzeni R^n w przestrzeń R^k. Każdy zbiór S składający się z s punktów należących do dwóch różnych klas P i N nazywamy f-separowalnym jeśli zbiór $f(x) : x \in S$ jest liniowo separowalny.*

f-separowalność oznacza, że istnieje k wymiarowy wektor w taki, że dla wszystkich $x^P \in P$ oraz $x^N \in N$ zachodzi

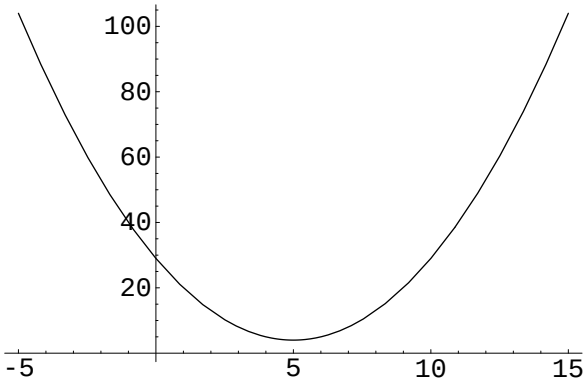
$$\sum_{i=1}^{k} w_i f_i(x^P) \geq 0,$$



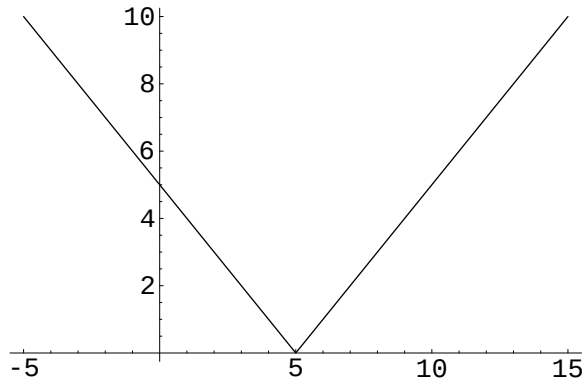
Rysunek 19.2: Wykres funkcji $\phi(x) = \exp(-\frac{|x-2|^2}{3^2})$.



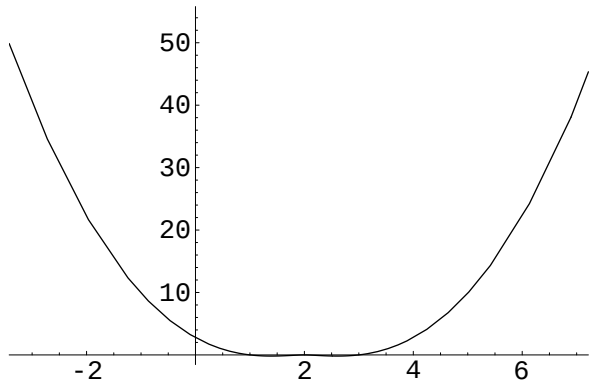
Rysunek 19.3: Wykres funkcji $\phi(x) = \sqrt{|x-2|^2 + 4^2}$.



Rysunek 19.4: Wykres funkcji $\phi(x) = |x-5|^2 + 2^2$.



Rysunek 19.5: Wykres funkcji $\phi(x) = |x - 5|$.



Rysunek 19.6: Wykres funkcji $\phi(x) = |x - 2|^2 \log(|x - 2|)$.

$$\sum_{i=1}^{k} w_i f_i(x^N) < 0.$$

Definicja 19.3. *Niech $\phi = [\phi_1, \ldots, \phi_k]$ będzie wektorem funkcji radialnych, takich że $\phi_i : R^n \to R$ dla $i = 1, \ldots, k$. Dwa zbiory punktów P oraz N, takie że $P, N \subset \mathbf{R}^n$ nazywamy* **nieliniowo ϕ-separowalnymi** *jeśli dla każdego punktu $x^P = (x_1^P, \ldots, x_n^P) \in P$ oraz $x^N = (x_1^N, \ldots, x_n^N) \in N$ istnieje k liczb rzeczywistych $w_1, \ldots, w_k$, takich, że*

$$\sum_{i=1}^{k} w_i \phi_i(x^P) \geq 0, \tag{19.1}$$

$$\sum_{i=1}^{k} w_i \phi_i(x^N) < 0. \tag{19.2}$$

Podstawą matematyczną funkcjonowania sieci radialnych, jest następujące twierdzenie Covera:

Twierdzenie 19.1 (Cover, 1965). *Niech f będzie funkcją działającą z przestrzeni R^n w przestrzeń R^k. Każdy zbiór S składający się z s losowo wybranych punktów z przestrzeni R^n jest f-separowalny z prawdopodobieństwem*

$$\sum_{j=0}^{k-1} \binom{s-1}{j} / 2^{s-1}.$$

Jak widać dla ustalonego s prawdopodobieństwo wzrasta wraz ze wzrostem k a więc ze wzrostem wymiaru przestrzeni rzutowania. Jeśli $k \approx s$ wówczas prawdopodobieństwo tego, że elementy zbioru S są f-separowalne liniowo jest równe 1. Mówiąc inaczej, prawdopodobieństwo separowalności elementów zbioru S wzrasta wraz ze wzrostem wymiaru przestrzeni, na którą rzutujemy te elementy.

W kontekście radialnych sztucznych sieci neuronowych oznacza to, że jeśli przyjmiemy dostateczną ilość neuronów z radialną funkcją aktywacji ϕ_i to uda się z prawdopodobieństwem równym 1 znaleźć taki wektor wagowy w (czyli nauczyć sieć) aby spełnione były zależności 19.1 oraz 19.2. Innymi słowy uda się znaleźć taką funkcję f postaci

$$f(x) = \sum_{i=1}^{k} w_i \phi_i(x), \tag{19.3}$$

że spełnione będą zależności 19.1 oraz 19.2.

W ogólności nauka sztucznej sieci neuronowej z matematycznego punktu widzenia równoważna jest zadaniu poszukiwania takiej funkcji f, dla której zachodzi

$$f(x_i) = t_i, \quad i = 1, \ldots, s$$

gdzie t_i jest oczekiwaną wartością w punkcie x_i a s określa ilość tych punktów. Jest to więc typowe zadanie interpolacyjne.

Sieci poznane do tej pory rozwiązywały to zadanie na zasadzie aproksymacji globalnej jako, że dotyczyła ona wielu neuronów naraz.

Odmiennym sposobem postępowania jest lokalne dopasowanie wielu pojedynczych funkcji aproksymacji do zadanych wartości. W takim przypadku odwzorowanie pełnego zbioru danych jest sumą odwzorowań lokalnych. W tym przypadku mówimy o

aproksymacji lokalnej, ponieważ oddziaływanie pojedynczych tzw. **funkcji bazowych** obserwowane jest tylko w wąskim obszarze przestrzenni danych.

Stąd możemy powiedzieć, że sieć neuronowa typu radialnego działa na zasadzie wielowymiarowej interpolacji s różnych wektorów wejściowych z n wymiarowej przestrzeni w zbiór k liczb rzeczywistych. Przyjmując $k = s$ oraz centra funkcji radialnych ϕ (np. funkcji gaussowskich) jako kolejne elementy x_i (tzn. $\phi_i(\cdot) = \phi(\cdot, x_i)$), równanie przyjmie postać układu równań liniowych

$$\begin{matrix} \phi_{11} & \phi_{21} & \dots & \phi_{s1} \\ \phi_{12} & \phi_{22} & \dots & \phi_{s2} \\ \dots & \dots & \dots & \dots \\ \phi_{1s} & \phi_{2s} & \dots & \phi_{ss} \end{matrix} \cdot \begin{matrix} w_1 \\ w_2 \\ \dots \\ w_s \end{matrix} = \begin{matrix} t_1 \\ t_2 \\ \dots \\ t_s \end{matrix} \tag{19.4}$$

gdzie ϕ_{ij} oznacza wartość i-tej funkcji ϕ w j-tym punkcie x_i. W wielu przypadkach układ ten posiada rozwiązanie. Niestety rozwiązanie to jest tylko teoretyczne. Z praktycznego punktu widzenia prowadzi ono bowiem do niesatysfakcjonujących właściwości uogólniających sieci. i znacznych nakładów obliczeniowych (macierz złożona z funkcji ϕ ma wymiar $s \times s$). Problem postawiony w ten sposób staje się przewymiarowany, gdyż liczba zmiennych w równaniu przewyższa wielokrotnie wszelkie stopnie swobody modelowanego procesu. Dzieje się tak, ponieważ dla każdego z s wektorów wejściowych sieć stworzyłaby jeden neuron. Rezultatem nadmiarowości wag będzie dopasowanie modelu (sieci) do wszelkich szumów i niedokładności występujących w danych. W efekcie otrzymana hiperpłaszczyzna interpolująca będzie mocno niegładka co przełoży się na niewielkie zdolności uogólniania. Dlatego też należy przeprowadzić redukcje liczby funkcji bazowych tak, aby znaleźć optymalne, przybliżone rozwiązanie w przestrzeni o mniejszym wymiarze.

19.2 Neuronowa sieć radialna

Konkluzją z poprzedniego rozdziału jest konieczność poszukiwania rozwiązania suboptymalnego, aproksymującego rozwiązanie dokładne, w przestrzeniach o mniejszym wymiarze niż s. Przy ograniczeniu się do k funkcji bazowych równanie (19.4) przyjmie postać

$$\begin{matrix} \phi_{11} & \phi_{21} & \dots & \phi_{k1} \\ \phi_{12} & \phi_{22} & \dots & \phi_{k2} \\ \dots & \dots & \dots & \dots \\ \phi_{1s} & \phi_{2s} & \dots & \phi_{ks} \end{matrix} \cdot \begin{matrix} w_1 \\ w_2 \\ \dots \\ w_k \end{matrix} = \begin{matrix} t_1 \\ t_2 \\ \dots \\ t_s \end{matrix}$$

a rozwiązanie aproksymujące można przedstawić w postaci

$$f(x) = \sum_{i=1}^{k} w_i \phi_i(x, c_i), \tag{19.5}$$

gdzie c_i, $i = 1, \dots, k$ jest zbiorem centrów jakie należy wyznaczyć. W szczególności przyjmując $k = s$ mamy sytuację z poprzedniego podrozdziału i otrzymujemy rozwiązanie dokładne.

Tak więc z teoretycznego punktu widzenia zadanie polega na dobraniu odpowiedniej liczby parametrów funkcji radialnych (tj. c_i) oraz takim doborze wag w_i, $i = 1, \dots, k$

aby rozwiązanie funkcji $f(x)$ najlepiej przybliżało rozwiązanie dokładne. Problem ten sprowadza się do minimalizacji funkcji celu E opisanej równaniem

$$E = \sum_{i=1}^{s} \left[\sum_{j=1}^{k} w_j \phi(x_i, c_j) - t_i \right]^2,$$

gdzie

- k jest liczbą funkcji radialnych,
- s liczbą próbek uczących,
- x_i wektorem wejściowym,
- t_i oczekiwaną (zadaną) wartością.

Rozwiązanie (19.5) można intepretować w postaci neuronowej sieci radialnej (RBF, ang. *radial basis function*) o strukturze przedstawionej na rysunku //uzup// (dla uproszczenia przyjęto jedno wyjście). Siec RBF ma strukturę dwuwarstwową. Neurony warstwy pierwszej reprezentują odwzorowanie nieliniowe realizowane przez neurony o radialnych funkcjach bazowych. Neurony wyjściowe są liniowe a ich rolą jest sumowanie ważonych sygnałów z warstwy pierwszej.

19.3 Metody uczenia

Mówiąc o metodach nauki w przypadku sieci radialnych możemy mówić o:

- metodach doboru parametrów funkcji bazowych;
- metodach doboru wag neuronów warstwy wyjściowej;
- kompleksowych metodach pozwalających na dobór zarówno parametrów jak i wartości wag.

19.3.1 Dobór parametrów

Jest to zasadniczy problem związany z nauką sieci radialnych wpływający na ostateczną jakość nauki sieci. Sposród wielu metod doboru parametrów (centrów) funkcji radialnych w dalszej częsci ograniczymy się do dwóch:

- wybór losowy,
- wybór przy zastosowaniu procesu samoorganizacji.

Wybór losowy

Jest to najprostsze rozwiązanie, w którym wybór parametrów funkcji dokonywany jest losowo przy rozkładzie równomiernym. Przy wyborze gaussowskiej funkcji radialnej przyjmujemy wartość odchylenia standardowego funkcji zależną od rozrzutu losowo wybranych centrów

$$\sigma = \frac{d}{\sqrt{2n}},$$

gdzie d oznacza maksymalną odległość między centrami a n jest ilością centrów.

Wybór przy zastosowaniu procesu samoorganizacji

Znacznie lepsze wyniki niż wybór losowy uzyskuje się stosując wstępny podział danych wejściowch na klastry przy pomocy jednej z wielu procedur samoorganizacji. Można w tym celu wykorzystać zarówno algorytmy przeznaczone dla sieci samoorganizujących się na zasadzie rywalizacji jak i inne np. K-uśrednień[2]. Proces samoorganizacji zastosowany do danych uczących powoduje podział przestrzeni sygnałów wejściowych na obszary Voronoia (obszary atrakcji), reprezentujące oddzielne grupy danych. Centrum klastra utożsamiamy z centrum odpowiedniej funkcji bazowej. Liczba funkcji jest równa liczbie klastrów. Dodatkowe parametry wyznaczamy w oparciu o klastry, np. szerokość funkcji jest równa rozrzutowi danych w klastrze.

19.3.2 Dobór wag

Dobór wag, zgodnie z podanymi wcześniej informacjami, dotyczy tylko warstwy wyjściowej. Stąd można w tym przypadku użyć właściwie dowolnego algorytmu nauki sieci jednowarstwowych. Można także użyć metod bezpośrednich polegających na rozwiązaniu odpowiedniego układu równań liniowych.

19.3.3 Metody kompleksowe

Do metod kompleksowych, a więc pozwalających ustalić zarówno parametry funkcji bazowych jak i wartości wag, zaliczać będziemy wszelkie metody gradientowe uczenia nadzorowanego zaprezentowane np. w rozdziałach 16 czy **??**. Ostateczna postać wzorów będzie odbiegała w swej formie od tam podanych, co spowodowane jest koniecznością uwzględnienia większej liczby zmiennych[3], jednak metody stosuje się analogicznie jak w przypadku zwykłych sieci skierowanych do przodu.

19.4 Porównanie z sieciami sigmoidalnymi

Pomimo pozornego podobieństwa do sieci wielowarstwowych sigmoidalnych sieci RBF różnią się od nich w sposób istotny.

- Sieć radialna ma strukturę ustaloną o dwu warstwach. Pierwsza warstwa, wejściowa, składa się z neuronów radialnych, druga, wyjściowa, składa się z neuronów liniowych.
- Znacznie szerszy jest repertuar dostępnych funkcji bazowych (porównaj wzory //tutu// i rysunki //tutu//).
- Każda funkcja bazowa ma indywidualnie dobierane parametry.
- Argumentem funkcji radialnej jest odległość próbki od centrum.
- Odmienny przebieg procesu uczenia, który musi uwzględniać także konieczność doboru centrów i parametrów kształtu funkcji bazowych. Mimo to proces uczenia jest prostszy.

[2] Algorytm ten lepiej chyba znany jest pod angielsą nazwą K-means.

[3] Oprócz wektora wag zmiennymi są także centra oraz ewentualne dodatkowe parametry funkcji bazowych.

- Sieci radialne bazujące na funkcjach mających wartość niezerową jedynie w określonej przestrzeni wokół centrów, realizują aproksymację typu lokalnego o ograniczonym zasięgu działania. Lokalność działania funkcji bazowych umożliwia też znacznie łatwiejsze powiązanie parametrów funkcji bazowych z fizycznym rozmieszczeniem danych uczących w rozważanej przestrzeni. Dzięki temu znacznie łatwiej dobrać optymalne parametry startowe dla algorytmów uczących.
- Zmniejszenie znaczenia problemu doboru liczby neuronów ze względu na lokalny charakter aproksymacji reprezentowany przez poszczególne funkcje bazowe.

Rozdział 20

Sieci rekurencyjne

20.1 Wejście zależne od wyjścia

Można powiedzieć, że sieci poznane we wcześniejszych rozdziałach opierają się na przetwarzaniu statycznych danych. Mówiąc „statyczne" mamy tutaj na myśli to, że kolejno następujące po sobie sygnały nie są w żaden sposób od siebie zależne. Oto bowiem podajemy na wejście jakieś sygnały i obliczamy wyjście. Jeśli teraz podamy kolejne sygnały wejściowe to zupełnie nie ma znaczenia czy coś przed chwilą było liczone, czy nie. To znaczy, że kolejne sygnały wyjściowe nie zależą od siebie i właściwie to mogły by być obliczane w dowolnej kolejności.

Często jednak zdarza się, że sygnały jakie mamy do przetworzenia są ze swojej natury sygnałami „zależnymi" w tym sensie, że na wartość sygnału w pewnej chwili czasowej mają wpływ wartości tego sygnału z chwil go poprzedzających. W jaki sposób można modelować tego typu zależności? Najbardziej narzucające się rozwiązanie, biorąc pod uwagę poznane do tej pory narzędzia, jest następujące.

Jednokierunkowy przepływ sygnałów

Załóżmy, że wektor wejściowy $x(t)$ w chwili czasowej t skutkuje sygnałem wyjściowym $y(t)$. Statyczny zbiór uczący (o m elementach) wygląda następująco:

$$
\begin{aligned}
&l_1 = ((x_1), (t_1)),\\
&l_2 = ((x_2), (t_2)),\\
&l_3 = ((x_3), (t_3)),\\
&l_4 = ((x_4), (t_4)),\\
&l_5 = ((x_5), (t_5)),\\
&l_6 = ((x_6), (t_6)),\\
&\dots\\
&l_m = ((x_m), (t_m)),
\end{aligned}
$$

a wektor $x(t)$, dla ustalonej chwili czasowej t przyjmuje jako swoją wartość pierwszy element pary l_t, natomiast $y(t)$ jest drugim elementem. Zauważmy, że w tym przypadku wszystkie sygnały t_i zależą tylko i wyłącznie od odpowiadających im sygnałów x_i.

Jeśli teraz chcemy oddać zależności czasowe pomiędzy kolejnymi sygnałami, wówczas

zbiór uczący możemy skonstruować np. w następujący sposób

$$
\begin{aligned}
&l'_1 = ((0,0,0,x_1),(t_1)),\\
&l'_2 = ((0,0,x_1,x_2),(t_2)),\\
&l'_3 = ((0,x_1,x_2,x_3),(t_3)),\\
&l'_4 = ((x_1,x_2,x_3,x_4),(t_4)),\\
&\ldots\\
&l'_m = ((x_{m-3},x_{m-2},x_{m-1},x_m),(t_m)).
\end{aligned}
$$

Z danych tych widzimy, że o ile w pierwszym przypadku x_3 ma wpływ tylko na wartość t_3 to w drugim wpływa na wartość t_3, t_4, t_5 oraz t_6. Tak więc teraz wektor $x(t)$ został rozszerzony o dodatkowe informacje pochodzące z innych (poprzednich) chwil czasowych. Wadą takiego rozwiązania jest konieczność zwiększenia wymiaru wektora wejściowego $x(t)$ tyle razy ile poprzednich sygnałów wejściowych chcemy uwzględnić (w powyższym przykładzie wymiar wektora wejściowego wzrósł czterokrotnie) co natychmiast odbije sie (zwykle negatywnie) na szybkości i jakości uczenia sieci.

Sygnał sprzężenia zwrotnego

Pomimo iż poprzednie podejście może okazać się wystarczające, to jednak zdecydowanie bardziej naturalne jest postępowanie następujące. Przy analogicznym jak poprzednio założeniu, że wektor wejściowy $x(t)$ w chwili czasowej t skutkuje sygnałem wyjściowym $y(t)$ konstruujemy następujący zbiór uczący:

$$
\begin{aligned}
&l_1 = ((x_1,0,0,0),(t_1)),\\
&l_2 = ((x_2,y_1,0,0),(t_2)),\\
&l_3 = ((x_3,y_2,y_1,0),(t_3)),\\
&l_4 = ((x_4,y_3,y_2,y_1),(t_4)),\\
&l_5 = ((x_5,y_4,y_3,y_2),(t_5)),\\
&l_6 = ((x_6,y_5,y_4,y_3),(t_6)),\\
&\ldots\\
&l_m = ((x_m,y_{m-1},y_{m-2},y_{m-3}),(t_m)).
\end{aligned}
$$

Jak widać w tym przypadku zbiór uczący zawiera jeden sygnał właściwy dla danej chwili czasowej t (czyli x_t) oraz kilka ostatnich odpowiedzi (reakcji) sieci (czyli y_{t-1}, y_{t-2}, y_{t-3}. Takie postępowanie jest znacznie bardziej naturalne: oto bowiem sygnały wyjściowe z chwil poprzednich ($t-1$, $t-2$, $t-3$) mają wpływ na bieżącą odpowiedź w chwili t. W tym momencie pojawiają się następujące, powiązane ze sobą, kwestie:

- Powstaje konieczność potraktowania wyjść z sieci jako wejść. Mówiąc inaczej, musimy sygnał wyjściowy potraktować jako wejście, czyli pojawia się tak zwane **sprzężenie zwrotne**[1].
- Jak konstruować zbiór uczący, skoro zawiera on elementy y_i? Nie będziemy w tym momencie tego jeszcze wyjaśniać, ale powiemy tutaj tylko tyle, że w praktyce w zbiorze uczącym nie ma elementów y_i.

[1] Sprzężenie zwrotne (ang. *feedback*) – oddziaływanie sygnałów stanu końcowego (wyjściowego) procesu (systemu, układu), na jego sygnały referencyjne (wejściowe). Polega na otrzymywaniu przez układ (proces, system) informacji o własnym działaniu (o wartości wyjściowej).

- Jak aktualizować stany neuronów? W sieciach ze sprzężeniem zwrotnym możemy bowiem stany neuronów zmieniać na dwa sposoby: **asynchroniczny** i **synchroniczny**.

  Asynchroniczna zmiana stanu neuronów oznacza, że w chili t zmieniamy stan tylko jednego, losowo wybranego neuronu.

  Synchroniczna zmiana stanu neuronów oznacza, że w chili t zmieniamy stan wszystkich neuronów.

  W sieciach skierowanych do przodu asynchroniczność raczej nie ma zastosowania co spowodowane jest głównie problememi natury teoretycznej. Jak do tej pory wszystkie znane i stosowane sieci skierowane do przodu są synchroniczne.

Powyższe rozumowanie doprowadziło nas w naturalny sposób do sieci o strukturze rekurencyjnej.

20.2 Sieci typu RTRN

Może mało klasycznie[2], ale prezentację sieci rekurencyjnych, jako najbardziej logiczną kontynuację wcześniejszych wywodów, rozpoczniemy od sieci typu RTRN (ang. *Real Time Recurrent Network*, 1989, [31]). Ogólną strukturę sieci przedstawia rysunek 20.1. Liczba neuronów wynosi m i każdy z nich tworzy sprzężenie zwrotne, przy czym sygnały zwrotne podawane na wejście sieci są opóźnione o jeden cykl pracy sieci. Wśród m neuronów tylko m' z nich stanowi faktyczne wyjście z sieci. Pozostałe pełnią rolę jednostek ukrytych. Sieć ma n wejść „zewnętrznych" (przy czym jednym z nich może być sygnał biasu). Stąd wektor wejściowy ma postać:

$$x(t) = [x_1(t), x_2(t), \ldots, x_m(t), x_{m+1}(t), \ldots, x_{m+n}(t)].$$

Uwzględniając, że część sygnałów wejściowych pochodzi ze zprzężenia zwrotnego i jest opóźniona o jeden cykl, wektor ten można zapisać w postaci:

$$x(t) = [y_1(t-1), y_2(t-1), \ldots, y_m(t-1), x_{m+1}(t), \ldots, x_{m+n}(t)].$$

Przyjmując ciągłą funkcję aktywacji f stan wszystkich neuronów w sieci w chili t określamy analogicznie jak do tej pory, tj.

$$y_i(t) = f(net_i(t)), \quad i = 1, \ldots, m,$$

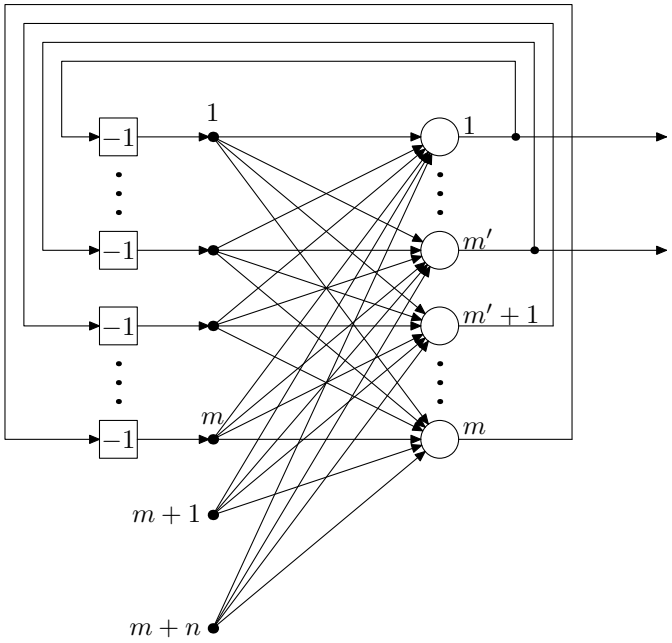
gdzie net_i jest łącznym pobudzeniem:

$$net_i(t) = \sum_{j=1}^{m+n} w_{ij} x_j(t).$$

Zdefiniujmy teraz funkcję błędu:

$$E(t) = \frac{1}{2} \sum_{i=1}^{m} (e_i(t))^2,$$

[2]Zwykle omawianie sieci rekurencyjnych rozpoczyna się od „klasyka" czyli sieci Hopfielda.



Rysunek 20.1: Schemat sieci RTRN.

gdzie

$$e_i = \begin{cases} t_i(t) - y_i(t), & \text{gdy} \quad i = 1, \ldots, m' \\ 0, & \text{gdy} \quad i = m'+1, \ldots, m \end{cases}$$

Ponieważ w sieciach RTRN przyjmujemy, że połączenia zwrotne mają wagę o stałej wartości +1 zatem mimo rekurencyjnych zależności modyfikacji podlegają tylko wagi z jednej warstwy. Tak więc można do nauki takiej sieci wykorzystać praktycznie dowolny algorytm przeznaczony dla sieci skierowanych do przodu.

20.2.1 Algorytm Williamsa-Zipsera

Jednak najciekawszą własnością przyjętej struktury jest możliwość wyprowadzenia rekurencyjnych wzorów na aktualizację wag, co znacznie pozwala przyspieszyć proces uczenia. Przy zastosowaniu metody najszybszego spadku (porównaj rozdział **??** punkt **??**), o aktualizacji wartości wag decyduje wektor gradientu funkcji celu względem tych wag:

$$\nabla E(t) = \frac{\partial E(t)}{\partial w_{pq}}.$$

Policzmy teraz jego wartość[3]:

$$\begin{aligned}
\frac{\partial E(t)}{\partial w_{pq}} &= \frac{1}{2}\sum_{i=1}^{m}(e_i(t))^2\Big|_{w_{pq}} = \frac{1}{2}\sum_{i=1}^{m}\left[2e_i(t)e_i(t)\big|_{w_{pq}}\right] = -\sum_{i=1}^{m}\left[e_i(t)y_i(t)\big|_{w_{pq}}\right] \\
&= -\sum_{i=1}^{m}\left[e_i(t)f(net_i(t))\big|_{w_{pq}}\right] = -\sum_{i=1}^{m}\left[e_i(t)f'(net_i(t))net_i(t)\big|_{w_{pq}}\right] \\
&= -\sum_{i=1}^{m}\left[e_i(t)f'(net_i(t))\Big[\sum_{k=1}^{m+n}w_{ik}x_k(t)\Big]\Big|_{w_{pq}}\right] \\
&= -\sum_{i=1}^{m}\left[e_i(t)f'(net_i(t))\Big[\sum_{k=1}^{m+n}w_{ik}|_{w_{pq}}x_k(t) + \sum_{k=1}^{m+n}w_{ik}x_k(t)|_{w_{pq}}\Big]\right] \\
&= -\sum_{i=1}^{m}\left[e_i(t)f'(net_i(t))\Big[\delta_{ip}x_q(t) + \sum_{k=1}^{m}w_{ik}x_k(t)|_{w_{pq}} + \sum_{k=m+1}^{m+n}w_{ik}x_k(t)|_{w_{pq}}\Big]\right] \\
&= \ldots
\end{aligned}$$

Uwzględniając, że sygnały $x_{m+1},\ldots,x_{m+n}$ nie zależą od wartości w_{ij}, gdyż są to wejścia do sieci, mamy:

$$\begin{aligned}
\ldots &= -\sum_{i=1}^{m}\left[e_i(t)f'(net_i(t))\Big[\delta_{ip}x_q(t) + \sum_{k=1}^{m}w_{ik}x_k(t)|_{w_{pq}}\Big]\right] \\
&= \ldots
\end{aligned}$$

Biorąc pod uwagę, że wejścia od x_1 do x_m w chili t mają dokładnie takie same wartości jak wyjścia y_1 do y_m o jeden cykl pracy wcześniejsze (czyli $t-1$) otrzymujemy dalej:

$$\begin{aligned}
\ldots &= -\sum_{i=1}^{m}\left[e_i(t)f'(net_i(t))\Big[\delta_{ip}x_q(t) + \sum_{k=1}^{m}w_{ik}y_k(t-1)|_{w_{pq}}\Big]\right] \\
&= -\sum_{i=1}^{m}\left[e_i(t)f'(net_i(t))\Big[\delta_{ip}x_q(t) + \sum_{k=1}^{m}w_{ik}f(net_k(t-1))|_{w_{pq}}\Big]\right].
\end{aligned}$$

Z powyższego widać, że:

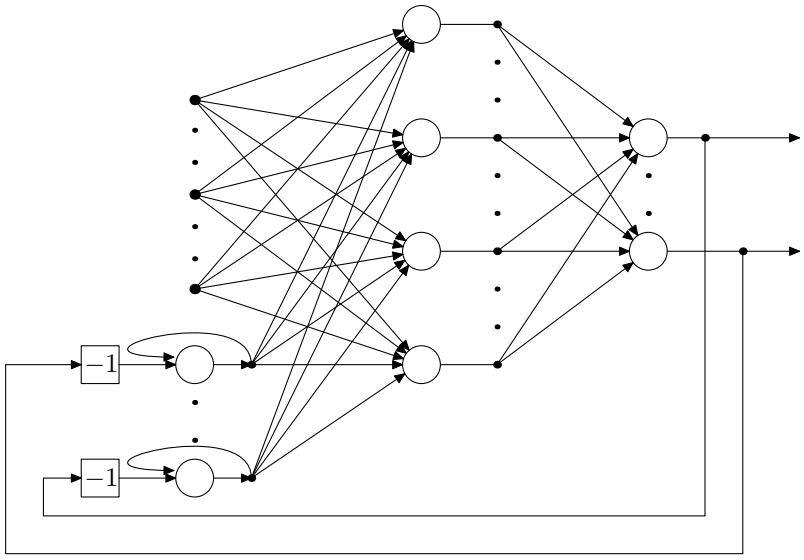
$$\sum_{i=1}^{m}\left[e_i(t)f(net_i(t))\big|_{w_{pq}}\right] = \sum_{i=1}^{m}\left[e_i(t)f'(net_i(t))\Big[\delta_{ip}x_q(t) + \sum_{k=1}^{m}w_{ik}f(net_k(t-1))|_{w_{pq}}\Big]\right],$$

więc mamy następującą zależność rekurencyjną:

$$f(net_i(t))\big|_{w_{pq}} = f'(net_i(t))\Big[\delta_{ip}x_q(t) + \sum_{k=1}^{m}w_{ik}f(net_k(t-1))|_{w_{pq}}\Big].$$

Jak więc widać wartość pochodnej funkcji f po zmiennej w_{pq} w chili t zależy od wartości pochodnej tejże funkcji z poprzedniego cyklu pracy. Spostrzeżenie to pozwala na obliczanie wartości wyrażenia $\frac{\partial E(t)}{\partial w_{pq}}$ właściwie w czasie rzeczywistym i stąd właśnie pochodzi nazwa sieci. Zwykle przyjmuje się, że w chwili $t=0$ wszystkie pochodne funkcji f były równe 0. Wyprowadzona formuła na aktualizację wektora wagowego nosi nazwę algorytmu Williamsa-Zipsera.

[3]Przyjmujemy, że zapis typu $xy\big|_i z$ oznacza iloczyn trzech składników: x, z oraz pochodnej y liczonej po zmiennej i.



Rysunek 20.2: Schemat sieci Jordana.

20.3 Sieć Jordana

Z roku 1986 pochodzi architektura sieci rekurencyjnej, nazywanej siecią Jordana. Ogólną strukturę sieci przedstawia rysunek 20.2. Zwykle stosuje się dwie warstwy, ale nic nie stoi na przeszkodzie aby zastosować ich więcej. W sieci Jordana sygnał wyjściowy z sieci stanowi jednocześnie sygnał wejściowy poprzez podanie jego na tzw. **neurony stanu** (ang. *state units*) z liniową funkcją aktywacji. Dodatkowo każdy neuron stanu na jedno ze swoich wejść pobiera własny sygnał wyjściowy. Neuronów stanu jest tyle ile wyjść z sieci. Połączenia zwrotne mają wagę o stałej wartości +1. Do uczenia sieci tego typu można wykorzystać algorytm będący połęczniem zwykłej propagacji wstecznej (czyli np. metody największego spadku) oraz algorytmu Williamsa-Zipsera.

20.3.1 Algorytm nauki

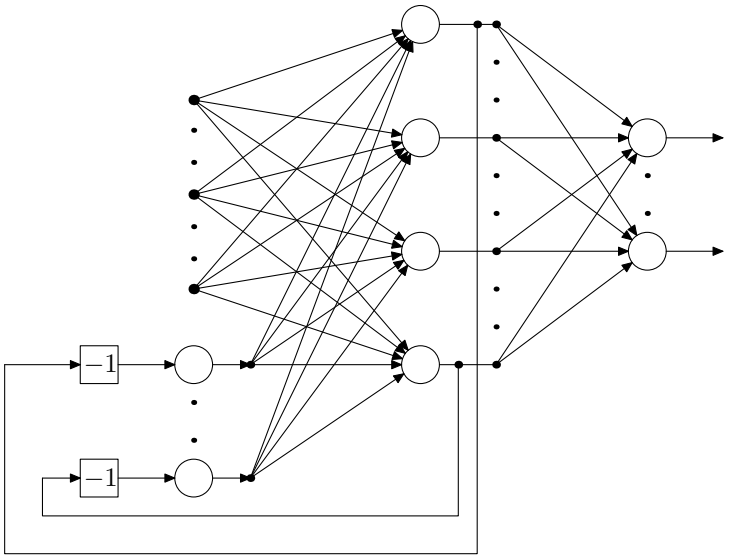
20.4 Sieć Elmana

Sieć Elmana (1990) jest siecią częściowo rekurencyjną o strukturze dwuwarstwowej, w której sprzężenie zwrotne dotyczy tylko warstwy pierwszej. Ogólną strukturę sieci przedstawia rysunek 20.3. Sieć przypomina swoją strukturą sieć Jordana z tym wyjątkiem, że:

- sygnał zwrotny nie pochodzi z neuronów wyjściowych ale z neuronów warstwy pierwszej,
- neurony stanu, nazywane w tym przypadku neuronami kontekstowymi (ang. *context neurons*) nie są połączone same ze sobą.

20.4.1 Algorytm nauki

Niech:



Rysunek 20.3: Schemat sieci Elmana.

- n – liczba wejść do sieci;
- k – liczba neuronów w warstwie pierwszej;
- z – liczba wyjść z sieci (neuronów w drugiej warstwie).

Wektor wejściowy do sieci Elmana przyjmuje postać:

$$[1, x_1^1(t), \ldots, x_n^1(t), x_{n+1}^1(t), \ldots, x_{n+k}^1(t)],$$

gdzie

$$[x_{n+1}^1(t), \ldots, x_{n+k}^1(t)] = [y_1^1(t-1), \ldots, y_k^1(t-1)]$$

jest wektorem wyjść z warstwy ukrytej otrzymany w poprzedniej iteracji. Sygnały wyjściowe neuronów poszczególnych warstw wyliczane są następująco:

$$y_i^2(t) = f\left(\sum_{j=0}^{z} y_j^1(t) w_{ij}^2\right), \quad i = 1, \ldots, z;$$

$$y_i^1(t) = f\left(\sum_{j=0}^{n+k} x_j^1(t) w_{ij}^1\right) = f\left(\sum_{j=0}^{n} x_j^1(t) w_{ij}^1 + \sum_{j=1}^{k} y_j^1(t-1) w_{ij+k}^1\right), \quad i = 1, \ldots, k.$$

Przyjmuje się, że $[x_{n+1}^1(0), \ldots, x_{n+k}^1(0)] \equiv 0$.

Do uczenia sieci Elmana stosujemy te same metody uczenia co do sieci jednokierunkowej. Uczenie sieci jest procedurą modyfikowania wartości wag mającą na celu minimalizację funkcji błędu:

$$E = \frac{1}{2} \sum_{i=1}^{z} (t - y_i^2)^2,$$

dla ustalonej pary uczącej $(p, t) \in L$.

Zastosowanie metody uczenia sieci jednokierunkowej do sieci Elmana jest możliwe ze względu na sposób przetwarzania informacji. Zauważmy, że identyczny przepływ sygnałów w sieci uzyskamy budując „zwykłą" sieć o $n+k+1$ wejściach skierowaną do przodu (a więc bez sprzężeń zwrotnych) taką, że wektor sygnałów wejściowych równy będzie

$$[1, x_1^1(t), \ldots, x_n^1(t), x_{n+1}^1(t), \ldots, x_{n+k}^1(t)]^T = [1, p_1, \ldots, p_n, y_1^1(t-1), \ldots, y_k^1(t-1)].$$

Zatem przy odpowiednim przygotowaniu danych uczących (a raczej ich przygotowywaniu, gdyż każda nowo podawana próbka ucząca musi zostać uzupełniona o wyjścia z warstwy ukrytej otrzymane w poprzedniej iteracji) możliwe jest wykorzystanie algorytmów nauki przeznaczonych dla sieci skierowanych do przodu. Metoda ta nie zawsze jest wygodna, dlatego w ćwiczeniu 30 podajemy wyprowadzenia wzorów w sytuacji, gdy nie chcemy ingerować w dane uczące.

20.5 Sieć Hopfielda

Jednym z najbardziej znanych przedstawicieli sieci rekurencyjnych jest sieć Hopfielda (1982). Składa się ona (patrz rysunek 20.4) z n w pełni połączonych ze sobą neuronów (tzn. na zasadzie każdy z każdym) asynchronicznie i niezależnie uaktualniających swoje wyjście. Każdy neuron pełni zarówno rolę neuronu wejściowego jak i wyjściowego, tzn.

$$x_i(t) = y_i(t-1),$$

gdzie i jest numerem neuronu i jednocześnie wejścia oraz wyjścia, t oznacza zmienną czasową, x_i jest i-tym sygnałem wejsciowym (identycznym dla wszystkich neuronów) a y_i sygnałem wyjściowym z i-tego neuronu. Sygnały wejściowe i wyjściowe są wartościami binarnymi. Oryginalnie były to wartości 0 i 1, ale zostały one zastąpione przez -1 oraz 1 jako, że dają one lepsze wyniki. Łączne pobudzenie i-tego neuronu w czasie $t+1$ wyraża się wzorem

$$net_i(t) = \sum_{k=1, k\neq i}^{n} w_{ik} x_k(t) - \theta_i,$$

gdzie

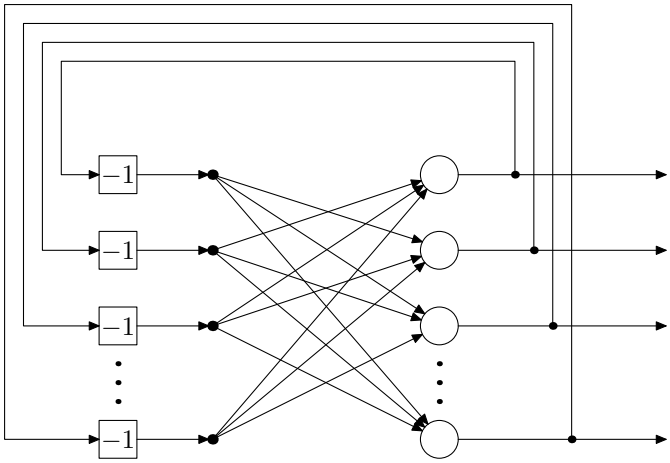
- w_{ik} – waga łącząca i-ty neuron z k-tym wejściem (wyjściem k-tego neuronu),
- $x_k(t)$ – k-te wejście w chwili t (wyjście w k-tego neuronu z chwili $t-1$[4]),
- θ_i – bias neuronu.

Jako funkcja aktywacji stosowana jest funkcja progowa postaci

$$f(t) = \begin{cases} +1, & \text{gdy } net(t) > 0 \\ -1, & \text{gdy } net(t) < 0 \\ f(t-1), & \text{gdy } net(t) = 0 \end{cases} \tag{20.1}$$

Ponadto przyjmujemy symetryczną postać macierzy wag, tzn. $w_{ij} = w_{ji}$. Jak widać ze wzoru na wartość $net_i(t)$ nie występują samosprzężenia (suma jest od $k=1$ do n z wyłączeniem przypadku gdy $k=i$ co można osiągnąc przyjmując $w_{ii}=0$ i licząc sumę od $k=1$ do n).

[4]Nie jest to do końca ścisłe, ale pisząc tak mamy na myśli to, że sygnał wyjściowy pochodzi z wcześniejszych cykli pracy.



Rysunek 20.4: Schemat sieci Hopfielda.

20.5.1 Cykl pracy sieci

Zasadniczy cykl pracy sieci Hopfielda polega na asynchronicznym uaktualnianiu wyjść neuronów. Bardziej szczegółowo działanie sieci Hopfielda przebiega następująco:

1. Podanie na wejście wektora x. Wektor x podawany jest na wejście sieci tylko raz aby zainicjalizować jej działanie. W kolejnych krokach rolę sygnału (wektora) wejściowego połnią sygnały sprzężenia zwrotnego.
2. Obliczenie wartości sygnałów wyjściowych dla wszystkich neuronów.
3. Sygnały wyjściowe stają się nowym sygnałem wejściowym.
4. Losowy wybór i-tego neuronu.
5. Dla neuronu i obliczamy wartość jego wyjścia. Sygnał wyjścia aktualizuje wektor wejściowy.
6. Jeśli nie jest spełniony warunek stopu, to powrót do punktu 4. W przeciwnym razie zakończ algorytm.

Przedstawiony powyżej sposób pracy sieci Hopfielda jest kłopotliwe z dwóch powodów.

- Kłopotliwe staje się ustalenie warunku zatrzymania algorytmu. Zwykle jako warunek stopu określa się taki stan sieci, w którym jej wyjście już się nie zmienia. Jak jednak stwierdzić, czy wyjście się nie zmienia jeśli aktualizacja stanu sieci jest asynchroniczna? Nie wystarczy wtedy zwykłe porównanie wartości wyjścia neuronu w chili t oraz $t-1$. Wybór jako warunku zatrzymania po wykonaniu zadanej ilości cykli może spowodować, że nie otrzymamy rozwiąznia – sieć nie zdąży dotrzeć do punktu w którym stabilizuje się (nie zmienia się) jej wyjście.
- Zasygnalizowany powyżej kłopot z analizowaniem działania sieci asynchronicznej.

Z wymienionych powyżej powodów zwykle do analizy działania wybiera się synchroniczny wariant sieci Hopfielda. Wówczas działanie sieci Hopfielda przebiega następująco:

1. Podanie na wejście wektora x. Wektor x podawany jest na wejście sieci tylko raz aby zainicjalizować jej działanie. W kolejnych krokach rolę sygnału (wektora) wejściowego połnią sygnały sprzężenia zwrotnego.
2. Obliczenie wartości sygnałów wyjściowych dla wszystkich neuronów.
3. Sygnały wyjściowe stają się nowym sygnałem wejściowym.
4. Porównanie poprzedniego i obecnego sygnału wejściowego. Jeśli są one identyczne, to kończymy działanie sieci. W przeciwnym przypadku powracamy do punktu 2.

20.5.2 Stabilność sieci

Powiemy, że i-ty neuron w sieci Hopfielda jest **stabilny** w chwili t, jeśli:

$$f_i(t) = f_i(t-1)$$

lub inaczej:

$$y_i(t) = f(net_i(t-1)).$$

Sieć jest stabilna (sieć jest w stanie stabilnym) gdy każdy nauron jest stabilny.

Definicja 20.1. *Funkcję postaci:*

$$E(x) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j w_{ij} - \sum_{i=1}^{n}\theta_i x_i. \tag{20.2}$$

nazywamy funkcją energii sieci Hopfielda dla stanu x.

Twierdzenie 20.1. *Sieć Hopfielda złożona z n neuronów i uaktualniająca swój stan asynchronicznie, dla zadanego stanu wejściowego x zbiega do stanu stabilnego będącego minimum (często lokalnym) funkcji energii.*

Dowód. Wartość funkcji energii dla stanu $x = [x_1, \ldots, x_n]$ wyraża się wzorem (20.2). Jeśli do aktualizacji stanu wybrano neuron k, to możliwe są dwie sytuacje:

1. Neuron nie zmienił swojego stanu na wyjściu. Wówczas także nie zmieni się wartość funkcji energii.
2. Neuron zmienił swój stan na wyjściu. Zatem mamy nowy wektor wejściowy

   $$x' = [x'_1, \ldots, x'_k, \ldots, x'_n]$$

   i nową wartość funkcji energii: $E(x')$, przy czym ponieważ $x_i = x'_i$ dla $i \neq k$, więc

   $$x' = [x_1, \ldots, x'_k, \ldots, x_n].$$

   Policzmy teraz różnicę pomiędzy stanem $E(x)$ a $E(x')$:

$$\begin{aligned} E(x) - E(x') &= \left(-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j w_{ij} - \sum_{i=1}^{n}\theta_i x_i\right) \\ &\quad - \left(-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} x'_i x'_j w_{ij} - \sum_{i=1}^{n}\theta_i x'_i\right) \\ &\quad \ldots \end{aligned}$$

W różnicy tej istotne są tylko te składniki które zawierają x_k lub x'_k (pozostałe się redukują):

$$\begin{aligned} \ldots &= \left(-\sum_{j=1}^{n} x_k x_j w_{kj} - \theta_k x_k \right) \\ &\quad - \left(-\sum_{j=1}^{n} x'_k x_j w_{kj} - \theta_k x'_k \right) \\ &\quad \ldots \end{aligned}$$

Poza tym usuwamy współczynnik $\frac{1}{2}$, gdyż w sumach podwójnych, ze względu na symetrię macierzy wag (czyli fakt iż $w_{ij} = w_{ji}$) składnik $x_k x_j w_{kj}$ występuje dwa razy. Ponieważ $w_{ii} = 0$ (brak samosprzężenia neuronów), więc:

- dla wszystkich $j \neq k$ mamy:

$$x_k x_j w_{kj} - x'_k x_j w_{kj} = (x_k - x'_k) x_j w_{kj}$$

- gdyby nie założona równość, wówczas dla $j = k$ jedynym wspólnym czynnikiem było by w_{kk}:

$$x_k x_k w_{kk} - x'_k x'_k w_{kk} = (x_k x_k - x'_k x'_k) w_{kk};$$

dzięki przjętemu założeniu także i w tym przypadku można napisać

$$x_k x_k w_{kk} - x'_k x'_k w_{kk} = (x_k - x'_k) x_k w_{kk},$$

gdyż z jednej strony mamy, że

$$x_k x_k w_{kk} - x'_k x_k w_{kk} = x_k \cdot 0 - x'_k \cdot 0 = 0$$

z drugiej zaś

$$(x_k - x'_k) x_k w_{kk} = (x_k - x'_k) \cdot 0 = 0.$$

A zatem mamy dalej:

$$\begin{aligned} \ldots &= -(x_k - x'_k) \sum_{j=1}^{n} w_{kj} x_j + \theta_k (x_k - x'_k) \\ &= -(x_k - x'_k) \left(\sum_{j=1}^{n} w_{kj} x_j - \theta_k \right) \\ &= -(x_k - x'_k) net_k \end{aligned}$$

Otrzymany iloczyn jest zawsze dodatni:

- jeśli $x_k = +1$ wówczas $x'_k = -1$ i net_k musiało być liczbą ujemną (bo nastąpiła zmiana stanu). Tak więc

$$-(1 - (-1)) \cdot a = -2 \cdot a > 0,$$

gdzie $a = sgn(net_k) < 0$;

- jeśli $x_k = -1$ wówczas $x'_k = +1$ i net_k musiało być liczbą dodatnią (bo nastąpiła zmiana stanu). Tak więc

$$-(-1-1)) \cdot a = -2 \cdot a > 0,$$

gdzie $a = sgn(net_k) > 0$.

Oznacza to, że iloczyn $-(x_k - x'_k)net_k$ jest zawsze większy od zera[5], co z kolei oznacza, że także różnica $E(x) - E(x')$ jest większa od zera, czyli $E(x) > E(x')$. Pokazaliśmy w ten sposób, że za każdym razem gdy stan wyjściowy neuronu ulegnie zmianie, oznacza to zmniejszenie wartości funkcji energii. Z drugiej strony wartości funkcji energii są ograniczone z dołu (wagi w_{ij} oraz próg θ_i są stałe, a x_i przyjmuje wartość -1 lub 1) a liczba różnych możliwych do przyjęcia przez konkretną sieć wartości jest skończona. Stąd wnioskujemy, że po skończonej ilości kroków (uaktualnień wartości wyjściowych sieci), funkcja energi nie będzie już malała, czyli znajdzimy się w stanie stabilnym sieci.

□

Uwaga 20.1.

Zauważmy, że powyższe twierdzenie dotyczy też sieci działającej synchronicznie. Różnica pomiędzy asynchronicznym a synchronicznym działaniem jest taka, że w danym cyklu uaktualniamy stan jednego losowo wybranego nauronu zamiast wszystkich. Jeśli uaktualniamy stan wszystkich neuronów to (w ramach jednego cyklu) wyjścia neuronów już uaktualnionych nie mają wpływu na neurony których stan dopiero (w danym cyklu) uaktualnimy. Tak więc można powiedzieć, że działanie synchroniczne jest powtórzeniem działania asynchronicznego dla wszystkich neuronów. Tak długo jak nie zostaną zaktualizowane wyjścia wszystkich neuronów, „zamrożony" pozostaje wektor wejściowy. Podanie jako wektora wejściowego wektora wyjściowego następuje dopiero po uaktualnienu stanu wszystkich neuronów.

Uwaga 20.2.

W trybie odtworzeniowym na podstawie zaszumionych próbek stanowiących stan początkowy neuronów sieci Hopfielda dążymy do uzyskania właściwego stanu końcowego odpowiadającego jednemu z zapamiętanych wzorców. Niestety w wielu przypadkach proces iteracyjny nie osiąga rozwiązania właściwego, ale rozwiązanie fałszywe. Jest kilka przyczyn tego zjawiska.

- *Wartość funkcji energetycznej zależy od iloczynu stanu dwu neuronów i jest symetryczna względem polaryzacji. Ten sam stan energetyczny jest przypisany obu polaryzacjom x_i i x_j, pod warunkiem, że obie zmieniają się na przeciwny w tej samej chili czasowej t. Dlatego też np. stan $[+1, +1, -1]$ charakteryzuje się identyczną wartością energii co stan $[-1, -1, +1]$ i oba stany stanowią jednakowo dobre rozwiązanie. Przejście z jednego stanu do drugiego jest możliwe przez zwykłą zamianę polaryzacji stanów wszystkich neuronów jednocześnie. Mówiąc inaczej, dla sieci obraz oryginalny i jego negatyw są dokładnie tym samym i sieć ich nie rozróżnia.*

[5]Dla przypadku gdy nastąpiła zmiana na wyjściu neuronu.

20.5.3 Nauka sieci

Proces uczenia sieci Hopfielda ma za zadanie taki dobór wartości wag, aby dla stanu (wektora) początkowego bliskiego jednemu z wektorów uczących x_i stan sieci zbiegał do takiego stanu stabilnego, w którym wektor wyjściowy sieci równy jest zapamiętanemu wektorowi w_i. Najprostsza, klasyczna, metoda doboru wartości wag bazuje na metodzie Hebba (patrz punkt 18.1 strona 189).

Zakładając, że sygnały wejściowe przyjmując wartości ze zbioru $\{-1, +1\}$, to przy prezentacji p wzorców uczących $x_i = [x_i^1, \ldots, x_i^n]$, $i = 1, \ldots, p$ wagi w_{ij} dla $i \neq j$ są dobierane według formuły

$$w_{ij} = \frac{1}{n} \sum_{p}^{k=1} x_i^k x_j^k.$$

Dla $i = j$ waga ma wartość równą 0. Przy takim trybie uczenia wagi przyjmują wartości uśrednione wielu próbek uczących. Nie jest to efektywna metoda nauki, ale w oparciu o nią można stosunkowo łatwo przeprowadzić analizę pojemności sieci, która prowadzi do dosyć interesujących wniosków.

20.5.4 Pojemność sieci

W celu uproszczenia dalszych rozważań łączne pobudzenie neuronu obliczać będziemy według wzoru

$$x_i(t+1) = sgn\left(\sum_{j=1}^{n} w_{ij} x_j(t)\right), \quad i = 1, ..., n, \tag{20.3}$$

a funkcja aktywacji przyjmie postać

$$sgn(x) = \begin{cases} 1 & gdy \quad x \geq 0, \\ -1 & gdy \quad x < 0 \end{cases}.$$

Załóżmy, że chcemy przechowywać m wzorów x^α, $\alpha = 1, ..., m$ tak aby były stabilne, tzn. aby zachodziła równość[6]

$$x_i^\alpha = sgn\left(\sum_{j=1}^{n} w_{ij} x_j^\alpha\right), \quad \alpha = 1, ..., m. \tag{20.4}$$

Dla wyznaczenia wartości wag użyjemy wzoru opartego o regułę Hebba

$$w_{ij} = \frac{1}{n} \sum_{\alpha=1}^{m} \left(x_i^\alpha x_j^\alpha - \delta_{ij}\right), \quad j = 1, ..., n, \tag{20.5}$$

gdzie

$$\delta_{ij} = \begin{cases} 0 & gdy \quad i \neq j, \\ 1 & gdy \quad i = j. \end{cases} \tag{20.6}$$

[6]Zauważmy, że wzór (20.3) należy rozumieć jako przypisanie, natomiast (20.4) jako porównanie.

Zauważmy, że macierz w jest macierzą symetryczną z zerami na głównej przekątnej. Ponieważ można pokazać, że ilość zapamiętanych wzorców jest co najwyżej równa ilości neuronów, czyli zachodzi $m \leq n$, więc podzielenie przez n sprowadza wszystkie wagi do przedziału $[-1, 1]$.

$i-$ty bit wzorca x_i^α będzie stabilny jeżeli równanie (20.4) będzie prawdziwe dla ustalonego i. Będzie to spełnione wówczas, gdy $\sum_{j=1}^n w_{ij} x_j^\alpha$ będzie tego samego znaku co x_i^α. Dwie liczby różne od zera są tego samego znaku, jeżeli ich iloczyn jest dodatni, a zatem warunek stabilności dla $i-$tego bitu przyjmuje postać

$$\left(\sum_{j=1}^{n} w_{ij} x_j^\alpha \right) x_i^\alpha \geq 0. \tag{20.7}$$

Poszukujemy największej liczby wzorców m wyrażonej jako funkcja liczby neuronów n, dla której warunek (20.7) jest spełniony dla wszystkich n bitów.

Z (20.5) i (20.7) mamy:

$$\frac{1}{n} \sum_{j=1}^{n} \left(\sum_{\beta=1}^{m} x_i^\beta x_j^\beta - \delta_{ij} \right) x_j^\alpha x_i^\alpha \geq 0 \iff \tag{20.8}$$

$$\frac{1}{n} \sum_{j \neq i}^{n} \sum_{\beta=1}^{m} x_i^\beta x_j^\beta x_j^\alpha x_i^\alpha \geq 0 \iff$$

$$\left(\frac{1}{n} \sum_{j \neq i}^{n} x_i^\alpha x_j^\alpha x_j^\alpha + \frac{1}{n} \sum_{j \neq i}^{n} \sum_{\beta \neq \alpha}^{m} x_i^\beta x_j^\beta x_j^\alpha \right) x_i^\alpha \geq 0 \iff$$

$$\left(\frac{n-1}{n} x_i^\alpha + \frac{1}{n} \sum_{j \neq i}^{n} \sum_{\beta \neq \alpha}^{m} x_i^\beta x_j^\beta x_j^\alpha \right) x_i^\alpha \geq 0. \tag{20.9}$$

Dla prostoty obliczeń zakładamy teraz, że wzorce, które chcemy przechowywać są losowe. Dla skorelowanych wzorców obliczenia są podobne lecz bardziej skomplikowane. Przyjmujmy zatem, że bit x_i^α jest wartością losową ξ_i^α, otrzymaną z prawdopodobieństwem

$$P(\xi_i^\alpha = -1) = P(\xi_i^\alpha = 1) = \frac{1}{2}. \tag{20.10}$$

Ponieważ zajmujemy się teraz wartościami losowymi nie możemy powiedzieć, że nierówność (20.9) jest prawdziwa lub nie. Możemy natomiast obliczyć prawdopodobieństwo tego, że ta nierówność będzie prawdziwa. Chcemy, by prawdopodobieństwo to było równe 1. Potrzebne w dalszej części elementarne wiadomości z zakresu rachunku prawdopodobieństwa zebrane zostały w rozdziale 4.

Nierówność (20.9) zawiera teraz $(n-1)(m-1)$ losowych wartości $\xi_i^\beta \xi_j^\beta \xi_i^\alpha \xi_j^\alpha$, $j = 1, ..., n$, $j \neq i$, $\beta = 1, ..., m$, $\beta \neq \alpha$. o których zakładamy, że są niezależne. Dla wartości losowych ξ_i^α, wartość oczekiwana wynosi 0 a wariancja 1.

Istotnie, z definicji 4.4 mamy

$$E\xi_i^\alpha = 1 \cdot P(\xi_i^\alpha = 1) + (-1) \cdot P(\xi_i^\alpha = -1) = \frac{1}{2} - \frac{1}{2} = 0.$$

natomiast z definicji 4.5 otrzymujemy

$$D^2\xi_i^\alpha = E(\xi_i^\alpha - E\xi_i^\alpha)^2 = (-1-0)^2 \cdot \frac{1}{2} + (1-0)^2 \cdot \frac{1}{2} = 1$$

Takie same parametry uzyskujemy dla wszystkich czterech wartości losowych: $\xi_i^\beta \xi_j^\beta \xi_i^\alpha \xi_j^\alpha$. Zgodnie z centralnym twierdzeniem granicznym (patrz twierdzenie 4.1 w rozdziale 4), dla $(n-1)(m-1) \to \infty$,

$$\sum_{j\neq i}^{n} \sum_{\beta\neq\alpha}^{m} \xi_i^\beta \xi_j^\beta \xi_i^\alpha \xi_j^\alpha \tag{20.11}$$

jest zamienną losową o rozkładzie Gaussa o wartości oczekiwanej 0 oraz wariancji $(n-1)(m-1)$, jako sumy indywidualnych wariancji. Ponieważ n jest stałą ustaloną liczbą oraz (20.11) jest zmienną losową o rozkładzie Gaussa to:

$$\frac{1}{n} \sum_{j\neq i}^{n} \sum_{\beta\neq\alpha}^{m} \xi_i^\beta \xi_j^\beta \xi_i^\alpha \xi_j^\alpha \tag{20.12}$$

również jest zmienną losową o rozkładzie Gaussa. Ponadto korzystając z własności wartości oczekiwanej i wariancji, wartość oczekiwana zmiennej losowej (20.12) wynosi 0 (patrz własność 4.1), zaś wariancja (patrz własność 4.2)

$$\sigma^2 = \frac{(n-1)(m-1)}{n^2} \approx \frac{m-1}{n} \tag{20.13}$$

Warunkiem zastosowania centralnego twierdzenia granicznego jest aby $(n-1)(m-1) \to \infty$. Dodatkowo załóżmy, że $n \to \infty$. Nie wynika to automatycznie, gdyż nie wiemy jakiej postaci jest m (jako funkcja, której argumentem jest n. np. może się zdarzyć, że $m = \frac{1}{n^2}$).

Chcemy, by prawdopodobieństwo spełnienia warunku (20.9) dla wartości losowych było równe 1. Mamy więc:

$$\lim_{\substack{n\to\infty,\\ (n-1)(m-1)\to\infty}} Pr\left[\left(\frac{n-1}{n}\xi_i^\alpha + \frac{1}{n}\sum_{j\neq i}^{n}\sum_{\beta\neq\alpha}^{m} \xi_i^\beta \xi_j^\beta \xi_j^\alpha\right) \xi_i^\alpha \geq 0\right] = 1 \quad \Longleftrightarrow \tag{20.14}$$

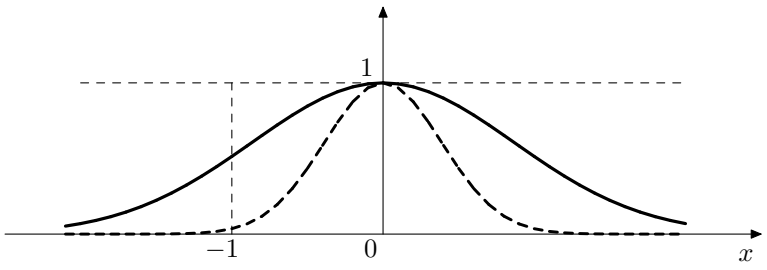
$$\lim_{\substack{n\to\infty,\\ (n-1)(m-1)\to\infty}} Pr\left[\left(\frac{n-1}{n}(\xi_i^\alpha)^2 + \frac{1}{n}\sum_{j\neq i}^{n}\sum_{\beta\neq\alpha}^{m} \xi_i^\beta \xi_j^\beta \xi_i^\alpha \xi_j^\alpha\right) \geq 0\right] = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\ (n-1)(m-1)\to\infty}} Pr\left[\left(\frac{n-1}{n} + \frac{1}{n}\sum_{j\neq i}^{n}\sum_{\beta\neq\alpha}^{m} \xi_i^\beta \xi_j^\beta \xi_i^\alpha \xi_j^\alpha\right) \geq 0\right] = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\ (n-1)(m-1)\to\infty}} Pr\left(\frac{1}{n}\sum_{j\neq i}^{n}\sum_{\beta\neq\alpha}^{m} \xi_i^\beta \xi_j^\beta \xi_i^\alpha \xi_j^\alpha \geq -1\right) = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\ (n-1)(m-1)\to\infty}} \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-1}^{\infty} e^{-\frac{x^2}{2\sigma^2}} dx = 1. \tag{20.15}$$

Ostatnia równoważność zachodzi w myśl definicji 4.8 oraz 4.10. Niech ilustrację stanowi



Rysunek 20.5: Linia ciągła przedstawia wykres funkcji $f(x) = exp(-(\frac{x}{1.5})^2)$ natomiast linia przerywana przedstawia wykres funkcji $f(x) = exp(-(\frac{x}{0.3})^2)$.

tutaj rysunek 20.5, z którego widać, że granica będzie równa 1 gdy wariancja σ będzie bardzo mała. Warunek stabilności (20.15) musi być spełniony dla n niezależnych bitów, a więc cały wzorzec α będzie stabilny gdy

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(\frac{1}{\sqrt{2\pi\sigma^2}} \int_{-1}^{\infty} e^{-\frac{x^2}{2\sigma^2}} dx \right)^n = 1. \tag{20.16}$$

Pokażemy teraz, że

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \sigma^2 = 0.$$

Zakłóżmy, że

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \sigma^2 = c > 0.$$

Z (20.16) uzyskujemy

$$\lim_{n\to\infty} \left(\frac{1}{\sqrt{2\pi c}} \int_{-1}^{\infty} e^{-\frac{x^2}{2c}} dx \right)^n = 1, \tag{20.17}$$

a to jest niemożliwe ponieważ

$$\frac{1}{\sqrt{2\pi c}} \int_{-1}^{\infty} e^{-\frac{x^2}{2c}} dx < 1.$$

Prawdziwe jest więc, że

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \sigma^2 = 0.$$

Wracając do (20.16) mamy

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(\frac{1}{\sqrt{2\pi\sigma^2}} \int_{-1}^{\infty} e^{-\frac{x^2}{2\sigma^2}} dx \right)^n = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(\frac{1}{\sqrt{2\pi\sigma^2}} \int_{-1}^{0} e^{-\frac{x^2}{2\sigma^2}} dx + \frac{1}{\sqrt{2\pi\sigma^2}} \int_{0}^{\infty} e^{-\frac{x^2}{2\sigma^2}} dx \right)^n = 1$$

Ponieważ całka

$$\int_{-1}^{1} e^{-\frac{x^2}{2\sigma^2}} dx$$

jest symetryczna względem osi OY, więc

$$\int_{-1}^{0} e^{-\frac{x^2}{2\sigma^2}} dx = \frac{1}{2} \int_{-1}^{1} e^{-\frac{x^2}{2\sigma^2}} dx.$$

Ponadto

$$\frac{1}{\sqrt{2\pi\sigma^2}} \int_{0}^{\infty} e^{-\frac{x^2}{2\sigma^2}} dx = \frac{1}{2},$$

a więc w konsekwencji otrzymujemy dalej

$$\lim_{\substack{n\to\infty,\\ (n-1)(m-1)\to\infty}} \left(\frac{1}{2} + \frac{1}{2} \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-1}^{1} e^{-\frac{x^2}{2\sigma^2}} dx \right)^n = 1 \tag{20.18}$$

Całkując przez podstawienie

$$g(x) = \frac{x}{\sigma} = t = h(t)$$

$$\frac{1}{\sigma}\, dx = dt$$

$$g(-1) = -\frac{1}{\sigma}, \quad g(1) = \frac{1}{\sigma}$$

otrzymujemy równość równoważną do (20.18)

$$\lim_{\substack{n\to\infty,\\ (n-1)(m-1)\to\infty}} \left(\frac{1}{2} + \frac{1}{2} \frac{1}{\sqrt{2\pi}} \int_{-\frac{1}{\sigma}}^{\frac{1}{\sigma}} e^{-t^2} dt \right)^n = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\ (n-1)(m-1)\to\infty}} \left(\frac{1}{2} + \frac{1}{2} erf\left(\frac{1}{\sigma}\right) \right)^n = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\ (n-1)(m-1)\to\infty}} \left(\frac{1}{2} \left(1 + erf\left(\frac{1}{\sigma}\right) \right) \right)^n = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\ (n-1)(m-1)\to\infty}} \left(\frac{1}{2} \left(1 + 1 - 2\left(1 - \Phi\left(\frac{1}{\sigma}\right) \right) \right) \right)^n = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\ (n-1)(m-1)\to\infty}} \left(\frac{1}{2} \left(1 + 1 - 2\frac{\sigma}{\sqrt{2\pi}}\, e^{-\frac{1}{2\sigma^2}} \right) \right)^n = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\ (n-1)(m-1)\to\infty}} \left(1 - \frac{\sigma}{\sqrt{2\pi}}\, e^{-\frac{1}{2\sigma^2}} \right)^n = 1 \quad \Longleftrightarrow \tag{20.19}$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(1+\frac{-n\frac{\sigma}{\sqrt{2\pi}}\ e^{-\frac{1}{2\sigma^2}}}{n}\right)^n = 1 \quad \Longleftrightarrow$$

$$e^{\lim\limits_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(-n\frac{\sigma}{\sqrt{2\pi}}\ e^{-\frac{1}{2\sigma^2}}\right)} = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(-\frac{n\sigma}{\sqrt{2\pi}}\ e^{-\frac{1}{2\sigma^2}}\right) = 0 \quad \Longleftrightarrow \tag{20.20}$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(n\sigma\ e^{-\frac{1}{2\sigma^2}}\right) = 0 \quad \Longleftrightarrow \tag{20.21}$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} n\sqrt{\frac{m-1}{n}}e^{-\frac{n}{2(m-1)}} = 0. \tag{20.22}$$

Ostatnia równość domyślnie określa m jako funkcję zależną od n. Nie można w oczywisty sposób określić m, dlatego dokonamy pewnego przypuszczenia i sprawdzimy, czy granica jest równa zero.

Przypuśćmy najpierw, że $m-1=n$. Wtedy

$$\lim_{n\to\infty} ne^{-\frac{1}{2}} = \infty, \tag{20.23}$$

dlatego zredukujmy m: $m-1=n/lnn$. Otrzymamy

$$\lim_{n\to\infty} n\frac{1}{\sqrt{lnn}}e^{-\frac{lnn}{2}} = \lim_{n\to\infty} n\frac{1}{\sqrt{lnn}}\left(e^{lnn}\right)^{-\frac{1}{2}} = \lim_{n\to\infty}\frac{n}{\sqrt{n\ lnn}} = \lim_{n\to\infty}\sqrt{\frac{n}{lnn}} = \infty, \tag{20.24}$$

Teraz niech $m-1=n/(2\ lnn)$. Granica jest równa

$$\lim_{n\to\infty} n\frac{1}{\sqrt{2\ lnn}}\ e^{-lnn} = \lim_{n\to\infty}\frac{1}{\sqrt{2\ lnn}} = 0. \tag{20.25}$$

Więc znaleźliśmy odpowiednią postać dla m. Zauważmy, że $m-1=n/(2\ lnn)$, $n\to\infty$ oznacza że $(n-1)(m-1)\to\infty$.

Jako wniosek z powyższych rozważań możemy stwierdzić, że udowodniliśmy następujące twierdzenie:

Twierdzenie 20.2. *Dla sieci Hopfielda, w której wektor wag dobrany jest według metody hebba, wzorzec będzie stabilny z prawdopodobieństwem 1, jeśli $n\to\infty$ oraz ogólna ilość zapamiętanych wzorcy m spełnia nierówność:*

$$m-1 \le \frac{n}{2\ lnn}. \tag{20.26}$$

20.6 Pojemność zależna od reprezentacji

Jak dotąd stany neuronów przyjmowały wartości +1 i −1. Najbardziej naturalną reprezentacją są jednak wartości binarne 0 i 1 (tak też było w oryginalnej pracy Hopfielda). Zawsze oczywiście można zmienić wzorzec

$$v^\alpha \in \{0,1\}^n$$

na wzorzec

$$x^\alpha \in \{-1,+1\}^n$$

za pomocą przekształcenia

$$2v_i^\alpha - 1 = x_i^\alpha, \quad i = 1, ..., n$$

W dalszej części, v będzie używane jako wzorzec z bitami o wartościach 0 i 1, a x dla wzorców o wartościach +1 i −1. Używając notacji v wzór (20.5) przyjmie postać:

$$w_{ij} = \frac{1}{n}\sum_{\alpha=1}^{m}\left((2v_i^\alpha - 1)(2v_j^\alpha - 1) - \delta_{ij}\right) \quad i = 1, ..., n. \tag{20.27}$$

Postępując analogicznie warunek (20.7) stabilności $i-$tego bitu jest teraz następujący:

$$\left(\sum_{j=1}^{n} w_{ij}(2v_i^\alpha - 1)\right)(2v_j^\alpha - 1) \geq 0. \tag{20.28}$$

Takie przedstawienie pozwala nam jedynie otrzymać wzorce o składowych 0, 1. Zastosujmy przekształcenie, które pozwoli otrzymać wartości wyjściowe neuronów należące do przedziału $[-1, 1]$. Stan j-tego neuronu można teraz zmieniać za pomocą następującej reguły:

$$\frac{1}{2}\left[(1-\lambda) + (1+\lambda)(2v_j - 1)\right], \quad 0 \leq \lambda \leq 1.$$

Zauważmy, że dla $\lambda = 0$, na wyjściu otrzymujemy v_j czyli wartość binarną. Natomiast dla $\lambda = 1$, na wyjściu mamy $2v_j - 1 = x_j$, wartość bipolarną +1 lub −1.

Twierdzenie 20.3. *Dla sieci Hopfielda z wartościami wyjść neuronów*

$$\frac{1}{2}\left[(1-\lambda) + (1+\lambda)(2v_j - 1)\right], \quad 0 \leq \lambda \leq 1,$$

wzorzec będzie stabilny z prawdopodobieństwem 1, *jeżeli* $n \to \infty$ *oraz dla* m *zostanie spełniona nierówność*

$$m - 1 = \frac{n\ (1+\lambda)^2}{lnn\ 4(1+\lambda^2)} \tag{20.29}$$

Dowód. Warunek stabilności $i-$tego bitu ma postać:

$$\left(\sum_{j=1}^{n} w_{ij}\frac{1}{2}\left[(1-\lambda) + (1+\lambda)(2v_j^\alpha - 1)\right]\right)(2v_j^\alpha - 1) \geq 0 \tag{20.30}$$

Podobnie jak w poprzednim podrozdziale przejdziemy z v_i^α do jednostajnie rozłożonych, losowych bitów γ_i^α. Wstawiając do powyższej nierówności wartości wag i biorąc pod uwagę, że:

$$(2\gamma_i^\beta - 1)(2\gamma_j^\beta - 1) - \delta_{ij} = 0$$

otrzymujemy:

$$Pr\left\{\left[\sum_{j=1}^{n}\frac{1}{n}\sum_{\beta=1}^{m}\Big((2v_i^\beta - 1)(2v_j^\beta - 1) - \delta_{ij}\Big)\frac{1}{2}\left[(1-\lambda) + (1+\lambda)(2v_j^\alpha - 1)\right]\right]\times \quad (20.31)$$

$$\times(2v_i^\alpha - 1) \geq 0\right\} = 1 \quad \Longleftrightarrow .$$

$$Pr\left\{\left[\frac{1}{2n}\sum_{j\neq i}^{n}(2\gamma_i^\alpha - 1)(2\gamma_i^\alpha - 1)(2\gamma_j^\alpha - 1)\left[(1-\lambda) + (1+\lambda)(2\gamma_j^\alpha - 1)\right] + \right.\right. \quad (20.32)$$

$$\left.\left. + \frac{1}{2n}\sum_{j=1}^{n}\sum_{\beta\neq\alpha}^{m}(2\gamma_i^\alpha - 1)(2\gamma_i^\beta - 1)(2\gamma_j^\beta - 1)\left[(1-\lambda) + (1+\lambda)(2\gamma_j^\alpha - 1)\right]\right] \geq 0\right\} = 1 \quad \Longleftrightarrow$$

$$Pr\left\{\left[\frac{1}{2n}\sum_{j\neq i}^{n}\left[(1-\lambda) + (1+\lambda)(2\gamma_j^\alpha - 1)\right] + \right.\right. \quad (20.33)$$

$$\left.\left. + \frac{1}{2n}\sum_{j=1}^{n}\sum_{\beta\neq\alpha}^{m}(2\gamma_i^\alpha - 1)(2\gamma_i^\beta - 1)(2\gamma_j^\beta - 1)\left[(1-\lambda) + (1+\lambda)(2\gamma_j^\alpha - 1)\right]\right] \geq 0\right\} = 1.$$

Przeanalizujmy pierwszy składnik:

$$\frac{1}{2n}\sum_{j\neq i}^{n}\left[(1+\lambda) + (1-\lambda)(2\gamma_j^\alpha - 1)\right]$$

Zmienna losowa $2\gamma_j^\alpha - 1$ posiada wartość oczekiwaną równą 0, stąd zmienna losowa $(1+\lambda) + (1-\lambda)(2\gamma_j^\alpha - 1)$ ma wartość oczekiwaną $1+\lambda$. Wariancja tej zmiennej losowej wynosi

$$\sum_{\gamma_j^\alpha}^{+1}\left[(1-\lambda)(2\gamma_j^\alpha - 1)\right]2 \;\; Pr(\gamma_j^\alpha) = (1-\lambda)^2$$

Z tego wynika, że $\frac{1}{2n}\Big[(1+\lambda) + (1-\lambda)(2\gamma_j^\alpha - 1)\Big]$ posiada wartość oczekiwaną

$$(1+\lambda)/2n$$

oraz wariancję

$$(1-\lambda)^2/4n^2.$$

Możemy teraz użyć centralnego twierdzenia granicznego do określenia wartości wyrażenia:

$$\frac{1}{2n}\sum_{j\neq i}^{n}\left[(1+\lambda) + (1-\lambda)(2\gamma_j^\alpha - 1)\right] \quad (20.34)$$

jako rozkładu Gaussa o wartości oczekiwanej

$$\mu = \frac{n-1}{2n}(1+\lambda) \tag{20.35}$$

i wariancji

$$\frac{n-1}{4n2}(1-\lambda)^2 \tag{20.36}$$

Składnik

$$\frac{1}{2n}\sum_{j=1}^{n}\sum_{\beta\neq\alpha}^{m}(2\gamma_i^\alpha-1)(2\gamma_i^\beta-1)(2\gamma_j^\beta-1)\left[(1-\lambda)+(1+\lambda)(2\gamma_j^\alpha-1)\right] \tag{20.37}$$

jest sumą $(n-1)(m-1)$ wartości losowych

$$\frac{1}{2n}(2\gamma_i^\alpha-1)(2\gamma_i^\beta-1)(2\gamma_j^\beta-1)\left[(1-\lambda)+(1+\lambda)(2\gamma_j^\alpha-1)\right]$$

Tak jak w poprzednim podrozdziale zakładamy, że wartości te są niezależne oraz osiągają wartość oczekiwaną 0 i wariancję

$$\frac{1}{4n^2}\left[\frac{1}{2}[(1-\lambda)+(1+\lambda)]^2+\frac{1}{2}[(1-\lambda)+(1+\lambda)]^2\right]=\frac{1+\lambda^2}{2n^2} \tag{20.38}$$

Używając centralnego twierdzenia granicznego stwierdzamy, że składnik przesłuchu posiada rozkład Gaussa z wartością oczekiwaną 0 i wariancją

$$\sigma^2=\frac{(n-1)(m-1)}{2n^2}(1+\lambda^2) \tag{20.39}$$

Warunek (20.33) zachowuje się podobnie jak (20.14). Tylko teraz drugi składnik posiada wariancję (20.36), której granica przy $n\to\infty$ wynosi 0. Jeśli przyjmujemy graniczny warunek o zerowej wariancji, warunek (20.33) można przedstawić podobnie jak na rysunku //tutu//10 i rysunku //tutu//11, tylko teraz mamy μ zamiast -1 dla warunku (20.15). Jeżeli weźmiemy także pod uwagę, że wszystkie n niezależnych bitów muszą być stabilne, warunek(20.36) będzie równoważny z warunkiem (podobnie jak (20.16))

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}}\left(\frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\mu}^{\infty}e^{-\frac{x^2}{2\sigma^2}}dx\right)^n=1 \quad\Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}}\left(\frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\mu}^{0}e^{-\frac{x^2}{2\sigma^2}}dx+\frac{1}{\sqrt{2\pi\sigma^2}}\int_{0}^{\infty}e^{-\frac{x^2}{2\sigma^2}}dx\right)^n=1 \quad\Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}}\left(\frac{1}{2}+\frac{1}{2}\frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\mu}^{\mu}e^{-\frac{x^2}{2\sigma^2}}dx\right)^n=1 \quad\Longleftrightarrow \tag{20.40}$$

Dokonując w (20.40) całkowania przez podstawienie

$$g(x)=\frac{x}{\sigma}=t=h(t)$$

$$\frac{1}{\sigma}\,dx=dt$$

$$g(-\mu) = -\frac{\mu}{\sigma}, \quad g(\mu) = \frac{\mu}{\sigma}$$

otrzymujemy równość równoważną do (20.40)

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(\frac{1}{2} + \frac{1}{2}\frac{1}{\sqrt{2\pi}} \int_{-\frac{\mu}{\sigma}}^{\frac{\mu}{\sigma}} e^{-t^2} dt\right)^n = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(\frac{1}{2} + \frac{1}{2} erf\left(\frac{\mu}{\sigma}\right)\right)^n = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(\frac{1}{2}\left(1 + erf\left(\frac{\mu}{\sigma}\right)\right)\right)^n = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(\frac{1}{2}\left(1 + 1 - 2\left(1 - \Phi\left(\frac{\mu}{\sigma}\right)\right)\right)\right)^n = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(\frac{1}{2}\left(1 + 1 - 2\frac{\sigma}{\sqrt{2\pi}\mu}\, e^{-\frac{\mu^2}{2\sigma^2}}\right)\right)^n = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(1 - \frac{\sigma}{\sqrt{2\pi}\mu}\, e^{-\frac{\mu^2}{2\sigma^2}}\right)^n = 1 \quad \Longleftrightarrow \tag{20.41}$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(1 + \frac{-n\frac{\sigma}{\sqrt{2\pi}\mu}\, e^{-\frac{\mu^2}{2\sigma^2}}}{n}\right)^n = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} e^{\left(-n\frac{\sigma}{\sqrt{2\pi}\mu}\, e^{-\frac{\mu^2}{2\sigma^2}}\right)} = 1 \quad \Longleftrightarrow$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} \left(-\frac{n\sigma}{\sqrt{2\pi}\mu}\, e^{-\frac{\mu^2}{2\sigma^2}}\right) = 0 \quad \Longleftrightarrow \tag{20.42}$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} n\frac{\sigma}{\mu}\, e^{-\frac{\mu^2}{2\sigma^2}} = 0 \quad \Longleftrightarrow \tag{20.43}$$

$$\lim_{\substack{n\to\infty,\\(n-1)(m-1)\to\infty}} n\frac{\sqrt{1+\lambda^2}}{1+\lambda}\sqrt{\frac{m-1}{n}} e^{-\frac{1(1+\lambda)^2(n-1)}{4(1+\lambda^2)(m-1)}} = 0. \tag{20.44}$$

Dla rozwiązania tego równania poszukujemy m jako funkcji n. Weźmy (ponownie odgadując)

$$m - 1 = \frac{n}{lnn}\frac{(1+\lambda)^2}{4(1+\lambda^2)}$$

Podstawiając takie m do (20.44) mamy

$$\lim_{n\to\infty} \frac{n}{2\sqrt{lnn}}\, e^{-lnn} = \lim_{n\to\infty} \frac{1}{2\sqrt{lnn}} = 0. \tag{20.45}$$

□

Uwaga 20.3.

- *Dla* $\lambda = 1$, *mamy*
$$m - 1 \leq \frac{n}{2\ lnn},$$
co jest analogicznym wynikiem do (20.26) wyprowadzonym dla sieci ze stanami neuronów -1 *i* $+1$.
- *Dla* $\lambda = 0$, *mamy*
$$m - 1 \leq \frac{n}{4\ lnn},$$
co jest pojemnością sieci dla neuronów binarnych o stanach 0 *i* 1.

Uwaga 20.4.

Zauważmy, że wartości wyjściowe neuronów wpływają na pojemność sieci neuronowej. Na przykład pojemność sieci o stanach neuronów -1 , $+1$ *jest większa niż pojemność sieci o neuronach binarnych.*

Rozdział 21

Sieci Fourier'a

Sieci Fourier'a są kontynuają idei zapoczątkowanej w sieciach samoorganizujących i sieciach radialnych a mianowicie „uneurawiania" idei matamatycznych i ich (czasem sztucznego) przenoszenia na grunt terminologii neuronowej. Zauważmy, że

- W przypadku sieci samoorganizujących właściwie neuron nie istnieje. Nie istnieje, bowiem nie ma on, jako element przetważający, żadnego znaczenia. Liczą się tylko jego wagi.
- W przypadku sieci radialnych dobrze znane narzędzie matematyki „ubrane" zostało w neurony. Nie mamy w tym przypadku praktycznie żadnej swobody, gdyż struktura sieci jest „narzucona" przez teorię.

Powyższe, to może nie tyle zarzut, co stwierdzenie faktów, które mają miejsce i z którymi trudno dyskutować. Oczywiście poparcie konstrukci sieci na drodze analizy teoretycznej jest jak najbardziej porządane, gdyż ułatwia jej wykorzystanie. Naturalną jednak kolejnością działania i myślenia jest porządek: sieć – uzasadnienie teoretyczne. Gdy bowiem najpierw zaczyna się od rozwijania teorii a potem nazywa się ją siecią, to mam dziwne wrażenie, że ktoś, na fali popularności jednej dziedziny, próbuje sprzedać dobrze znane rzeczy. Nie inaczej ma się sprawa z siecimi Fourier'a. Ich angielska nazwa (FSNNs, ang. *Fourier Series Neural Networks*) zdecydowanie lepiej oddaje ich pochodzenie. Nie są to bowiem sieci wymyślone przez Fourier'a, ale sieci, które są realizacją koncepcji szeregów Fourier'a. A prawdę powiedziawszy, to szeregiem Fourier'a wyrażonym w terminologi neuronowej. Lepszym odpowiednikiem była by nazwa Neuronowe Sieci Fourier'a, ale ponieważ do tej pory w polsko języcznej literaturze nie udało mi się znaleźć opisu tych sieci, więc proponuje dla skrócenia zapisu nazwę Sieci Fourier'a.

21.1 Podstawy matematyczne

Rozdział 22

Dobór architektury sieci

22.1 Ogólnie o strukturze sieci

Wymiar wejść i wyjść jest narzucony. Twierdzenie Kołmogorowa 2N+1, jedna warstwa ukryta (a wlasciwie, to zadna jesli traktowac wejsciowa jako ukryta). Twierdzenie dowodzi istnienia rozwiązania problemu aproksymacji funkcji wielu zmiennych przez superpozycję wielu funkcji jednej zmiennej i daje podstawy teoretyczne do określenia architektury sieci neuronowej.

22.2 Uogólnianie

Pzykład 1D i 2D.

22.3 Algorytm kaskadowej korelacji Fahlmana

problemy „poruszającego się celu" – ciągle zmieniający się cel ze względu na lokalny dostęp do informacji, nieskoordynowane zmiany

oraz efekt „stada". (w pewnym sensie samouczenie z sasiedztwem)

przypisanie na każdym etapie uczenia aktywnej roli tylko niektorym neuronom i połączeniom wagowym.

korelacja pomiedzy bledem sieci a wyjsciem (aktywnoscia) neuronu.

22.4 Sieć neuronowa z rozszerzeniami funkcyjnymi Pao

neurony ukryte jako elementy poszerzające informację o położeniu wzorca w przestrzeni. to jest OK gdy ilość wejść jest większa niż ilość wyjść.

nie wprowadza to nowej informacji, ale wzbogaca istniejącą

przyklad z XOR-em

Rozdział 23

Rozmyte sieci neuronowe

23.1 Idea

Jak napisaliśmy w punkcie 44.8 rozdzialu 44, możliwość reprezentacji dowolnej funkcji ciągłej wielu zmiennych za pomocą sumy funkcji rozmytych jednej zmiennej, uzasadnia ideę ich wykorzystania przy konstruowaniu sieci neuronowej. Połączenie sieci neuronowych z logiką rozmytą daje nam wymierne korzyści.

Optymalny dobór parametrów. Mamy możliwość dobrania optymalnej struktury sterownika poprzez proces uczenia sieci. W trakcie uczenia modyfikacji podlegają na przykład parametry funkcji przynależności, przez co bardziej prawdopodobne jest to, że będą one miały postać odpowiednią dla rozwiązywanego problemu. W tradycyjnym podejściu parametry te ustalane są przez eksperta i jego wiedza a czasem wręcz intuicja decyduje o przyszłej sprawności projektowanego układu.

Uzyskanie informacji o regułach. Można zaprojektować sieć według pewnych ogólnych zasad ich budowy (opisanych poniżej) jednak bez użycia żadnych konkretnych reguł (reguły rozumiane jako zbiór „warunków" i „akcji" typu `IF ... THEN ...`). Wówczas reguły te otrzymuje się z nauczonej sieci.

Z połączenia sieci neuronowej i logiki rozmytej otrzymujemy **rozmytą sieć neuronową** (ang. *neuro-fuzzy network*).

23.2 Schemat sieci

Ogólny schemat rozmytej sieci neuronowej jest ściśle powiązany ze strukturą sterownika rozmytego. Omówimy tutaj dwie najpopularniejsze postacie bazujące na modelu Mamdaniego oraz Takagi-Sugeno.

23.2.1 Przekształcenie rozmytego modelu Mamdaniego w rozmytą sieć neuronową

Przekształcenie bloku fuzyfikacji

Blok fuzyfikacji odpowiedzialny jest za rozmywanie a więc za określenie stopnia przynależności danej wejściowej do poszczególnych zbiorów rozmytych. Jednemu zbiorowi rozmytemu odpowiada jedna funkcja przynależności o pewnych parametrach zależnych od

jej postaci. Przyjmijmy, że mamy n wejść. Przez $F_i = \{\mu_{A_{i,1}}, \ldots, \mu_{A_{i,m_i}}\}$ określmy zbiór funkcji przynależności związanych z i-tym wejściem. Jak widać dla każdego $i = 1, \ldots, n$ ilość funkcji wynosi m_i i może być różna. Tak więc rolę bloku fuzyfikacji pełni $\sum_{i=1}^{n} m_i$ neuronów. Każdy neuron połączony jest z tylko jednym wejściem – wejściem dostarczającym sygnał dla definiowanego przez neuron zbioru rozmytego. Wszystkie neurony mogą mieć inną funkcję przynależności, choć często celem uproszczenia wybiera się takie same funkcje dla wszystkich neuronów. Wybór różnych funkcji przynależności komplikuje nieco procedurę nauki, gdyż wymaga odrębnych wzorów na pochodne. Pomimo tego, że teoretycznie warunkiem koniecznym, ze względu na stosowanie gradientowych metod nauki, wymaganym przy konstruowaniu sieci neuronowej jest różniczkowalna w sposób ciągły funkcja aktywacji, to z powodzeniem stosuje się odcinkowo-liniowe funkcje aktywacji (np. [22], s. 352).

Przekształcenie bazy reguł

Wyjściem bloku fuzyfikacji są stopnie przynależności i tego sygnału x_i do zbioru rozmytego $A_{i,j}$, $j = 1, \ldots, m_i$. Są one jednocześnie miarami spełnienia przesłanek składowych będących elementami części `IF` reguł postaci

```
IF
(x1 jest A) AND ... AND (xn jest A)
OR
(x1 jest A) AND ... AND (xn jest A)
OR
...
THEN (y jest B)
```

gdzie przez `A` oznaczono ogólnie zbiory rozmyte z lingwistycznej przestrzeni rozważań odpowiedniego sygnału. W bloku bazy reguł następuje obliczenie stopni spełnienia poprzednika reguł (część `IF`), które to stopnie aktywizują funkcję przynależności znajdującą się w następniku (część `THEN`) reguły. Zwykle stosuje się wnioskowanie typu MAX-MIN. Fragment sieci neuronowej przedstawiający poprzednik reguły pokazany jest na rysunku //tutu//. Jeden węzeł `AND` realizuje operację z użyciem T-normy (najczęściej operator min) na fragmencie reguły postaci

```
(x1 jest A) AND ... AND (xn jest A)
```

Tak więc węzłów tych jest tyle ile elementów połączonych operatorem `OR`. Węzeł `OR` realizuje operację z użyciem S-normy (najczęściej operator max) i jest tylko jeden. Może się jednak zdarzyć, że kilka różnych reguł aktywuje ten sam zbiór wyjściowy `B` występujący w następniku reguł. Wówczas istnieje konieczność dodania dodatkowego węzła realizującego operację max ze szystkich reguł aktuwujących dany zbiór rozmyty `B`.

Przekształcenie bloku defuzyfikacji

Wejściami bloku defuzyfikacji są stopnie aktywizacji $\tau^{(i)}$ poszczególnych zbiorów rozmytych wyjścia modelu. Struktura sieci neuronowej realizującej ten blok zależy od przyjętej metody defuzyfikacji. Jedną z najczęściej stosowanych metod jest metoda singletonów. Singletony $y^{(i)}$ umieszcza się albo w wierzchołkach albo w ich środku ciężkości funkcji przynależności znajdującej się we wniosku i-tej reguły. Wówczas wyjście modelu y

obliczane jest według wzoru (porównaj ćwiczenie z rozdziału 45)

$$y = \frac{\sum_{i=1}^{n} \tau^{(i)} y^{(i)}}{\sum_{i=1}^{n} \tau^{(i)}}.$$

Tak więc blokowi temu odpowiadają trzy węzły: dwa sumatory i węzeł dokonujący dzielenie wartości otrzymanych z sumatorów. Sumator odpowiadający mianownikowi sumuje stopnie aktywacji reguł. Sumator odpowiadający licznikowi sumuje iloczyny stopni aktywacji reguł pomnożonych przez wartość odpowiedniego singletona.

Przykład

Niech dany będzie model rozmyty z bazą reguł określoną jak poniżej i funkcjami przynależności przedstawionymi na rysunku //tutu//.

Kontynuacją tego przykładu jest ćwiczenie z rozdziału 38.

Krok 1.

23.2.2 Przekształcenie rozmytego modelu Takagi-Sugeno w rozmytą sieć neuronową

Zasadnicza różnica pomiędzy modelem Mamdaniego a Takagi-Sugeno polega na postaci nasyępników reguł w których to zamiast zbiorów rozmytych występują zależności funkcyjne. Najczęsciej są to funkcje liniowe, choć może być wykorzystana także i dowolna funkcja nieliniowa. W konsekwencji blok fuzyfikacji i baza reguł pozostają niezmienione. Modyfikacji musi ulec blok defuzyfikacji jako, że wyjście modelu y obliczane jest według wzoru (porównaj ćwiczenie z rozdziału 45)

$$y = \frac{\sum_{i=1}^{n} \tau^{(i)} f^{(i)}(x)}{\sum_{i=1}^{n} \tau^{(i)}},$$

gdzie $\tau^{(i)}$ jest stopniem aktywizacji i-tej reguły natomiast $f^{(i)}(x)$ jest funkcją występującą we wniosku tejże reguły. Tak więc blokowi temu odpowiadają trzy węzły: dwa sumatory i węzeł dokonujący dzielenie wartości otrzymanych z sumatorów. Inaczej jednak niż w modelu Mamdaniego określone zostają węzły sumujące a dokładnie węzeł liczący sumę licznika. Sumator odpowiadający licznikowi sumuje iloczyny stopni aktywacji reguł pomnożonych przez wartość odpowiedniej funkcji.

Przykład

Niech dany będzie model rozmyty z bazą reguł określoną jak poniżej i funkcjami przynależności przedstawionymi na rysunku //tutu//.

Kontynuacją tego przykładu jest ćwiczenie z rozdziału 38.

Krok 1.

23.3 Nauka

Uogólniając można powiedzieć, że do nauki otrzymanych sieci można z powodzeniem użyć dowolny z algorytmów wykorzystywany do nauki sieci wielowarstwowych jednokierunkowych. Wyprowadzenie wszystkich niezbędnych wzorów zawarte zostało w ćwiczeniu z rozdziału 38.

Rozdział 24

Badanie możliwości klasyfikacji

24.1 Wprowadzenie

Jednym z najczęściej powierzanych sieciom neuronowym zadań jest klasyfikacja obiektów. Przez „obiekt" rozumieć będziemy tutaj wiele różnych rzeczy: litery, obrazy, dźwięki itd. Każdy z takich obiektów reprezentowany jest przez ciąg liczb. Skoro tak, to ciąg ten możemy utożsamiać z pewnym wektorem, czyli punktem w przestrzeni odpowiedniego wymiaru. Zatem możemy inaczej powiedzieć, że sieć uczy się klasyfikowania punktów pewnej przestrzeni do różnych klas. Interesującym zagadnieniem jest:

- jak sposób rozmieszczenia punktów w przestrzeni wpływa na konstrukcję sztucznej sieci neuronowej;
- w jakim stopniu przyjęte założenia co do sposobu konstrukcji determinują możliwą do rozwiązania klasę problemów.

Aby znaleźć przynajmniej częściową odpowiedź na powyższe pytania, przebadamy kilka różnych sieci zwracając bacznie uwagę na sposób podziału przestrzeni sygnałów wejściowych.

W kolejnych podrozdziałach będziemy badać:

- sieci jednowarstwowe:
  - z neuronami o liniowej funkcji aktywacji,
  - z neuronami o progowej funkcji aktywacji,
  - z neuronami o sigmoidalnej funkcji aktywacji;
- sieci wielowarstwowe:
  - z neuronami o liniowej funkcji aktywacji,
  - z neuronami o progowej funkcji aktywacji,
  - z neuronami o sigmoidalnej funkcji aktywacji;
- sieci radialne.

24.2 Sieć jednowarstwowa

24.2.1 Przygotowanie

Przyjmujemy najprostszy z możliwych modeli sztucznej sieci neuronowej, tj. model sieci złożonej z jednego neuronu. Przyjmujemy, że neuron posiada dwa wejścia oraz bias. Ponieważ mamy dwa wejścia, więc wektory sygnałów wejściowych będą miały w tym przypadku tylko dwie zmieniające się współrzędne. Skoro tak, to łatwo możemy je interpretować jako pewien punkt na płaszczyźnie. Ponadto przyjmujemy dalej następujące założenia:

1. sygnały wejściowe zmieniają się w zakresie od -5 do $+5$ z pewnym, wybranym indywidualnie krokiem, np. 0.1;
2. wagi są liczbami losowymi z przedziału $[-5, 5]$.

24.2.2 Przebieg eksperymentu

1. Ustalmy wartość biasu na 0.0.
2. Wybieramy jedną z funkcji aktywacji: (16.1),(16.2) lub (16.4).
3. Losujemy wagi. W tym przypadku będą to wagi: w_1 związana z pierwszym sygnałem wejściowym x_1, w_2 związana z drugim sygnałem wejściowym x_2 oraz w_0 związana z biasem x_0.
4. Podajemy na wejście sieci parę punktów (x_1, x_2) z przestrzeni sygnałów wejściowych $([-5, 5] \times [-5, 5])$.
5. Dla pary sygnałów wejściowych obliczamy wartość wyjścia neuronu dla przyjętej funkcji aktywacji.
6. W zależności od wartości otrzymanej na wyjściu, w punkcie odpowiadającym wartości podanych na wejście sygnałów, stawiamy kropkę o odpowiednim kolorze. Sposób kolorowania podany zostanie poniżej.
7. Postępowanie z punktów $2 - 5$ kontynuujemy tak długo aż wyczerpiemy wszystkie punkty z zadanego obszaru przy przyjętym kroku.
8. Jeśli nie przebadaliśmy jeszcze zachowania dla trzech funkcji aktywacji to powracamy do punktu 2, gdzie wybieramy kolejną funkcję aktywacji.
9. Jeśli przebadaliśmy zachowanie sieci dla trzech funkcji aktywacji, to powracamy do punktu 2, ustalając wartość biasu na 1.0.

Całe powyższe postępowanie należy powtórzyć kilkakrotnie, aby móc zaobserwować pewne prawidłowości.

Pozostało jeszcze ustalić sposób kolorowania.

W przypadku funkcji progowej jako, że przyjmuje ona tylko dwie wartości, używamy tylko dwóch kolorów: czerwonego dla wartości równych 1 i niebieskiego dla wartosci równych -1.

Liniowa funkcja aktywacji może przyjmujmować dowolną wartość rzeczywistą z przedziału $[-55.0, 55.0]$. Dlatego przyjmujemy następujący sposób kolorowania (lub dowolny

inny będący jego rozszerzeniem – im więcej kolorów, tym lepiej będzie widać zdolności sieci do podziału płaszczyzny)

- dla wartości większych lub równych 1.0 – czerwony;
- dla wartości z przedziłu $[0.0, 1.0)$ – żółty;
- dla wartości z przedziłu $[-1.0, 0.0)$ – zielony;
- dla wartości mniejszych od -1.0 – niebieski.

Sigmoidalna funkcja aktywacji może przyjmujmować dowolną wartość rzeczywistą z przedziału $(-1.0, 1.0)$. Dlatego przyjmujemy następujący sposób kolorowania (uwaga o sposobie kolorowania j.w.)

- dla wartości większych lub równych 0.8 – czerwony;
- dla wartości z przedziłu $[0.0, 0.8)$ – żółty;
- dla wartości z przedziłu $[-0.8, 0.0)$ – zielony;
- dla wartości mniejszych od -0.8 – niebieski.

24.3 Sieć dwuwarstwowa

Po przebadaniu możliwości jednego neuronu (sieci jednowarstwowej), zajmiemy się siecią dwuwarstwową. Przyjmujemy następującą strukturę sieci: dwa neurony w pierwszej warstwie oraz jeden w drugiej – ostatniej (wyjściowej), połączone na zasadzie „każdy z każdym". Dla tak zbudowanej sieci powtarzamy kroki $1-9$ wykonywane podczas badania sieci jednowarstwowej.

24.4 Sieć radialna

Dla sieci radialnej przyjmujemy strukturę taką jak przedstawiona w rozdziale **??**, przy czym w warstwie radialnej używać będziemy trzech neuronów. Będziemy stosować bardzo prosty sposób kolorowania: czerwony dla wartości większych lub równych 0 i niebieski dla wartosci mniejszych od 0.

24.5 Wnioski

Oto jakie wnioski powinno dać się wysnuć w wyniku przeprowadzonych eksperymentów.

1. Pojedyńczy neuron z progową funkcją aktywacji zawsze dzieli przestrzeń na dwie podprzestrzenie za pomocą prostej. Prosta ta zawsze przechodzi przez punkt $(0, 0)$ jeśli bias ma wartość 0.0 (rys. **??**,**??**).
2. Sieć wielowarstwowa złożona z neuronów progowych dzieli przestrzeń za pomocą łamanej o ile bias ma wartość różną od 0.0 (rys. **??**).
3. Sieć złożona z jednostek liniowych zawsze dzieli przestrzeń w ten sam sposób niezależnie od ilości warstw. Podział ten wyznaczany jest przez równoległe do siebie linie proste (rys. **??**).

4. Pojedyńczy neuron z sigmoidalną funkcją aktywacji dzieli przestrzeń podobnie do neuronu liniowego.

5. Sieć wielowarstwowa z sigmoidalnymi funkcjami aktywacji dzieli przestrzeń na podobszary za pomocą krzywych. Jeśli bias ma wartość 0.0, wówczas podział ten jest symetryczny względem punktu $(0, 0)$ (rys. **??**,**??**).

6. Sieć radialna dzieli przestrzeń na podprzestrzenie będące „kombinacją" kół (rys. **??**).

Wnioski te będą troche inne, jeśli zamiast bipolarnych funkcji aktywacji użyjemy funkcji unipolarnych.

Rozdział 25

Najprostsza sieć

25.1 Zadanie

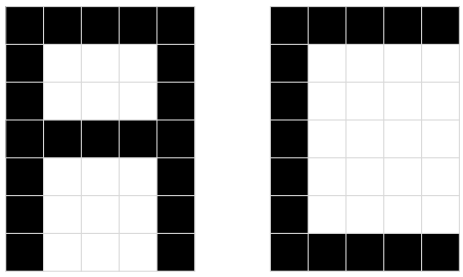
Zadanie postawione przed nami jest następujące: skonstruować sieć, która będzie mogła rozpoznawać wzorce graficzne należące do dwóch różnych klas. Jako wzorcy graficznych używać będziemy liter zapisanych w prostokątnym obszarze podzielonym na 7 wierszy i 5 kolumn. Każda komórka leżąca na przecięciu pewnego wiersza i kolumny może być czarna lub biała. W ten sposób, przyjmując za rozpatrywane wzorce litery A oraz C i przypisując im odpowiednio klasy oznaczane przez 0 oraz 1, możemy przedstawić je tak jak na rysunku 25.1.

Jeśli teraz nauczymy sieć tych dwóch wzorcy wówczas oczekiwać będziemy, że podanie zniekształconego wzorca A skutkować będzie odpowiedzią bliską 0, natomiast w przypadku zniekształconego wzorca C odpowiedzią bliską 1.

25.2 Konstrukcja sieci i przebieg nauki

Aby osiągnąć pożądane zachowanie sieci, którego przejawem jest zdolność do klasyfikowania obiektów, musimy:

- określić architekturę sieci
  - liczbę warstw,
  - liczbę neuronów w każdej warstwie,
  - sposób połączenia neuronów,



Rysunek 25.1: Litery A i C jako przykładowe wzorce.

– rodzaj funkcji aktywacji;

- określić sposób uczenia.

25.2.1 Architektura

Przyjmiemy najprostszą z możliwych architektur. Sieć będzie składała się z jednej warstwy. W warstwie tej będzie tylko jeden neuron. Ilość wejść określamy na równą ilości punktów tworzących wzorce graficzne, plus jedno stałe wejście, na które bez względu na wzorzec zawsze podawany jest sygnał o wartości 1.0 (bias). Zatem neuron będzie miał $5 \cdot 7 + 1 = 36$ wejść numerowanych od 0 do 35, przy czym wejście numer 0 jest wejściem o stałym sygnale. Jako funkcję aktywacji wybieramy funkcję liniową (16.1) lub sigmoidalną unipolarna (16.4).

25.2.2 Nauka

Celem nauki jest dobranie takich wartości wag sieci aby zachowywała się ona zgodnie z naszymi oczekiwaniami przypisując wzorcom podobnym do A wartość bliską 0 a wzorcom podobnym do C wartość bliską 1.

Na początku musimy zdefiniować dwa zbiory.

1. Zbiór pierwszy, to zbiór sygnałów jakie będą podawane na wejście neuronu; w naszym przypadku: sygnały reprezentujące litery A i C.

2. Zbiór drugi, to zbiór prawidłowych sygnałow wyjściowych neuronu; w naszym przypadku: odpowiednio 0 i 1.

Oznaczmy pierwszy z tych zbiorów przez P, drugi przez T. Elementy tych zbiorów oznaczać będziemy małymi literami – odpowiednio p i t, z ineksem wskazujacym na numer. Tak wiec mamy: $P = \{p_1, p_2\}$ oraz $T = \{t_1, t_2\} = \{0, 1\}$ gdzie p_1 i p_2 to wektory powstałe w oparciu o obrazy naszych wzorcy, według zasady: czarne pole = 1, białe = -1. Przyjmują one postać (odstępy co 5 dodane są dla czytelnosci):

```
p1 =11111   1-1-1-1 1   1-1-1-1 1   1 1 1 1 1   1-1-1-1 1   1-1-1-1 1   1-1-1-1 1
p2 =11111   1-1-1-1-1   1-1-1-1-1   1-1-1-1-1   1-1-1-1-1   1-1-1-1-1   1 1 1 1 1
```

Oba zbiory P oraz T tworzą zbiór uczący $L = \{P, T\}$ (porównaj 16.4 ze strony 136).

Dążymy do tego, aby wyjście z sieci y zależne od podanego wzorca p_i, $i = 1, 2$ różniło się jak najmniej od odpowiedzi oczekiwanej przez nas, czyli aby różnica $(t_i - y)$ dla $i = 1, 2$ była jak najmniejsza. Inaczej mówiąc, dążymy do minimalizacji określonej tym sposobem funkcji. Skoro jednak ma to być funkcja, to powinna mieć jakąś zmienną (lub zmienne). Ponieważ jedyne co może się w neuronie zmieniać to jego wagi, więc ostatecznie otrzymujemy nastepującą definicję funkcji (nazywanej funkcją błędu)

$$E(w) = (t_i - y), \qquad \text{gdzie} \quad w = [w_0, ..., w_{35}]\,.$$

Po prawej stronie nie widać aby y zależało od w, ale łatwo jest to zmienić (a raczej pokazać). Zgodnie z wcześniej podanymi informacjami y definiujemy jako

$$y = f(net),$$

przy czym *net* określone jest jako

$$net = \sum_{k=0}^{35}(x_k^i w_k),$$

gdzie x_k^i oznacza wartość k-tego wejścia w wyniku podania i-tego wzorca. Ostatecznie

$$E(w) = \left(t_i - f\left(\sum_{k=0}^{35} x_k^i w_k\right)\right).$$

Zwykle aby zapewnić sobie, że funkcja E ma minimum i aby otrzymać w wyniku dalszych działań „przyjemną" postać, przyjmuje się ją jako

$$E(\mathbf{w}) = \frac{1}{2}(t_i - y)^2.$$

Tak więc określiliśmy pewną funkcję i poszukujemy jej minimum. Do tego celu, przy założeniu różniczkowalności badanej funkcji, nadaje sie metoda gradientowa wyznaczania minimum funkcji. Pochodna funkcji policzona w pewnym punkcie, oznaczmy go przez $w(t)$[1], jej dziedziny wskazuje kierunek najszybszego wzrostu jej wartości. Tak więc minus pochodna w danym punkcie wskazuje kierunek najszybszego spadku jej wartości. Zatem szukając minimum przesuwamy się nieznacznie w tak wyznaczonym kierunku z punktu w którym jesteśmy obecnie, otrzymując nowy punkt oznaczany przez $w(t+1)$. Czyli, przyjmując za η niewielką liczbę rzeczywistą (np. 0.01) dokonujemy następującej modyfikacji:

$$w(t+1) = w(t) - \eta \cdot kierunek,$$

czyli

$$w(t+1) = w(t) - \eta E'(w(t)).$$

Współczynnik η nazywany jest **współczynnikiem uczenia**.

Policzmy teraz gradient[2] funkcji E

$$\nabla E = \left[\frac{\partial E}{\partial w_0}, ..., \frac{\partial E}{\partial w_{35}}\right]$$

(przyjąć musimy, że obliczenia prowadzone są dla pewnej pary (p_i, t_i)).

$$\frac{\partial E}{\partial w_k} = E'_{w_k}(\mathbf{w}) = \left[\frac{1}{2}\left(t_i - f(\sum_{m=0}^{35} x_m^i w_m)\right)^2\right]'_{w_k} =$$

$$\frac{1}{2}\cdot 2\left(t_i - f(\sum_{m=0}^{35} x_m^i w_m)\right)\cdot\left(t_i - f(\sum_{m=0}^{35} x_m^i w_m)\right)'_{w_k} =$$

$$-\left(t_i - f(\sum_{m=0}^{35} x_m^i w_m)\right) f'\left(\sum_{m=0}^{35} x_m^i w_m\right)\left(\sum_{m=0}^{35} x_m^i w_m\right)'_{w_k} =$$

[1]Zmienna t oznacza w tym przypadku pewną, intuicyjnie rozumianą, zależność czasową, polegającą na tym, że wartość $w(t)$ jest obliczana przed $w(t+1)$.

[2]Gradient, czyli wektor pochodnych cząstkowych funkcji wielu zmiennych.

$$-\left(t_i - f\left(\sum_{m=0}^{35} x_m^i w_m\right)\right) f'\left(\sum_{m=0}^{35} x_m^i w_m\right) x_k^i =$$
$$-(t_i - f(net))f'(net)x_k^i.$$

Iloczyn $(t_i - f(net))f'(net)$ oznaczamy przez δ i nazywamy **sygnałem błędu delta**.

W zależności od wyboru funckji aktywacji, inaczej będzie wyglądało wyrażenie $f'(x)$. Przyjmując liniową funkcję aktywacji (patrz wzór (16.1)) otrzymujemy

$$f'(x) = a.$$

Dla sigmoidalnej unipolarnej funkcję aktywacji (wzór (16.5)) otrzymujemy (patrz strona 150)

$$f'(x) = f(x)(1 - f(x)),$$

zaś dla bipolarnej (patrz wzór (16.4))

$$f'(x) = 0.5 \cdot (1 - f^2(x)).$$

Jedna i druga postać jest o tyle ciekawa, iż pochodna w punkcie x wyraża się przez wartość tejże funkcji w tym samym punkcie x. Oznacza to, że nie musimy przeprowadzać prawie żadnych dodatkowych obliczeń w celu obliczenia pochodnej.

25.3 Algorytm

Niech dany będzie zbiór uczący L postaci $L = \{P, T\}$, gdzie $P = \{p_1, p_2\}$ oraz $T = \{t_1, t_2\}$ (zgodnie z opisem powyżej).

1. Wybór
   (a) $\eta > 0$ – współczynnik uczenia (np. 0.2),
   (b) $E_{max} > 0$ – maksymalny błąd jaki chcemy osiągnąć (np. 0.01),
   (c) $C_{max} > 0$ – ilość kroków uczenia (np. 100).
2. Losowy wybór początkowych wartości wag (zmienne $w_0, ..., w_{35}$) jako niewielkich liczb (np. z przedziału $[-1, 1]$).
3. Przyjęcie $c := 1$.
4. Przyjęcie $E := 0$.
5. Uporządkowanie w losowej kolejności indeksów elementów zbioru P, otrzymując nowy zbiór I.
6. Wybór liczby k jako kolejnej liczby ze zbioru I.
7. Podanie k-tego obrazu ze zbioru P na wejscia neuronu: $x_i = p_{k,i}$, $i = 0, \ldots, 35$.
8. Obliczenie sygnału wyjściowego neuronu $y = f(net)$, gdzie $net = \sum_{i=0}^{35} x_i w_i$.
9. Uaktualnienie wartości wag według wzoru
$$w_i(t+1) = w_i(t) - \eta \frac{\partial E}{\partial w_i}, \quad i = 0, \ldots, 35.$$

10. Obliczenie błędu $b = (t_k - y)$ oraz zaktualizowanie sumarycznej wartości błędu dla danego cyklu uczącego: $E = E + \frac{1}{2}b^2$.

11. Jeśli k nie jest ostatnim elementem to przejście do kroku 6.

12. Jeśli $E < E_{max}$, to koniec algorytmu.

13. Jeśli $c < C_{max}$, to $c := c + 1$ i przejście do kroku 4. W przeciwnym razie koniec algorytmu.

25.4 Zadanie

Należy zaimplementować zaprezentowany algorytm dla identycznego zadania jak rozważanego w tekście (rozpoznawanie pewnych wzorcy graficznych). Program powinien mieć możliwość, po nauczeniu sieci, podawania przez użytkownika wartości sygnałów wejściowych i obliczenia dla nich odpowiedzi neuronu. Ilość rozpoznawanych klas można rozszerzyć wedle uznania.

Rozdział 26

Zadanie XOR

26.1 Zadanie

XOR (ang. *eXclusive OR*), czyli różnica symetryczna (lub też jedno z: alternatywa wykluczająca, alternatywa rozłączna, kontrawalencja), to logiczny (dwuargumentowy) funktor zdaniotwórczy, prawdziwy wtedy i tylko wtedy, gdy dokładnie jedno ze zdań jest prawdziwe.

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 26.1: Tablica prawdy XOR.

Przyglądając się powyższej tabeli, wyróżnić możemy obiekty dwóch klas. Pierwsza klasa składa się z obiektów $(0,0)$ oraz $(1,1)$ i przypisujemy jej wartość liczbową 0, natomiast drugiej, złożonej z elementów $(0,1)$ oraz $(1,0)$, przypisujemy wartość liczbową 1.

Tak sformułowane zadanie wygląda dosyć prosto i niewinnie, ale w rzeczywistości dla sztucznych sieci neuronowych jest nie lada wyzwaniem.

Zastanawiając się nad doborem sieci potrafiącej rozwiązać to zadanie możemy oprzeć się na wiadomościach i obserwacjach z rozdziału 24. Mówiąc bardzo obrazowo, szukamy takiej sieci, dla której będzie można ustalić wartości wag w ten sposób, aby punkty należące do klasy 0 znalazły się na obszarze jednego koloru, natomiast punkty należące do klasy 1, na obszarze innego koloru. Żadna ze znanych nam sieci jednowarstwowych zadania tego nie rozwiąże. Sieci te bowiem rozwiązują jedynie problemy liniowoseparowalne a problem XOR nie jest takim problemem. Zatem konieczna będzie sieć wielowarstwowa. Wystarczy nam struktura z dwiema warstwami: dwoma neuronami w pierwszej i jednym w drugiej (rys. **??**). Jako funkcję aktywacji przyjmujemy funkcję sigmoidalną unipolarną.

Do uczenia takiej sieci wykorzystamy algorytm z rozdziału 25, wprowadzając do niego jedynie pewne, raczej kosmetyczne zmiany, prowadzące do uogólnienia na sieci wielowarstwowe; nie będzie to jakościowo nowa metoda.

Zanim przejdziemy do opisu algorytmu poczynić musimy jeszcze małą uwagę. Otóż

teraz do oznaczenia wagi używać będziemy aż trzech indeksów. Zapis w_{bc}^{a} oznacza, że mówimy o wadze z warstwy a łączącej neuron o numerze c z warstwy $a-1$ z neuronem o numerze b z warstwy a.

Postąpimy teraz analogicznie jak przy wyprowadzaniu wzorów na modyfikację wag w rozdziale 25. Policzymy pochodną funkcji celu po kolejnych wagach. Jak się przekonamy wzór na zmianę wag warstwy wyjściowej pozostanie bez zmian. Wzór na zmianę wag w warstwie wcześniejszej uwzględniać będzie natomiast sygnał błędu, nazywany **sygnałem delta**. Wynika to stąd, że dla wszystkich warstw z wyjątkiem wyjściowej nie znamy prawidłowej odpowiedzi sieci, której znajomość wymagana jest aby wiedzieć jak duże poprawki wprowadzić. W związku z tym, wychodząc ze słusznego założenia, że neurony z warstw poprzednich mają wpływ na błąd w warstwie wyjściowej, będziemy ich błąd obliczać w oparciu o błąd warstwy wyjściowej (dokładniej mówiąc, to błąd warstwy I będzie obliczany na podstawie błędu warstwy $I+1$).

Oto jak będzie przebiegało wyprowadzenie potrzebnych wzorów. Przyjmujemy analogiczną postać funkcji błędu

$$E(\mathbf{w}) = \frac{1}{2}[t - f(net_1^2)]^2,$$

gdzie net_1^2 to pobudzenie neuronu 1 z warstwy 2.

Dla warstwy wyjściowej otrzymujemy

$$\frac{\partial E}{\partial w_{1p}^2} = \frac{\partial E}{\partial net_1^2}\frac{\partial net_1^2}{\partial w_{1p}^2} = \frac{1}{2}\left([t - f(net_1^2)]^2\right)'_{w_{1p}^2} =$$

$$\left[t - f(\sum_{k=0}^{2} x_k^2 w_{1k}^2)\right]\left[t - f(\sum_{k=0}^{2} x_k^2 w_{1k}^2)\right]'_{w_{1p}^2} = ...$$

gdzie:

- p zmienia się od 1 do ilości wejść dla warstwy 2 (w naszym przypadku do 3);
- suma po k jest od 0 do 2, gdyż sieć ma 2 wejścia +1 sygnał stały, razem 3;
- x_k^2 oznacza $k-ty$ sygnał wejściowy dla warstwy 2;
- w_{1k}^2 oznacza wagę łączącą neuron 1 z warstwy 2 z k-tym wejściem dla warstwy 2 (czyli na ogół z k-tym neuronem);

$$... = -(t - f(net_1^2))f'(net_1^2)x_p^2.$$

Iloczyn

$$(t - f(net_1^2))f'(net_1^2)$$

oznaczamy przez

$$\delta_1^2 = (t - f(net_1^2))f'(net_1^2).$$

W tym przypadku oznacza to, że jest to sygnał delta dla 1 neuronu z warstwy 2. Oczywiście

$$net_1^2 = \sum_{p=0}^{2} x_p^2 w_{1p}^2,$$

gdzie

$$x_0^2 = 1,$$

$$x_1^2 = f(\sum_{m=0}^{2} x_m^1 w_{1m}^1),$$

$$x_2^2 = f(\sum_{m=0}^{2} x_m^1 w_{2m}^1).$$

Teraz zajmijmy się warstwą pierwszą.

$$\frac{\partial E}{\partial w_{qp}^1} = \frac{1}{2}\left([t - f(net_1^2)]^2\right)'_{w_{qp}^1} = \left[t - f(net_1^2)\right]\left[t - f(net_1^2)\right]'_{w_{qp}^1} =$$

$$= -\left[t - y\right] f'(net_1^2)(net_1^2)'_{w_{qp}^1} = -\left[t - y\right] f'(net_1^2)(x_0^2 w_{10}^2 + x_1^2 w_{11}^2 + x_2^2 w_{12}^2)'_{w_{qp}^1} =$$

$$= -\left[t - y\right] f'(net_1^2)\left(w_{10}^2 + f(net_1^1)w_{11}^2 + f(net_2^1)w_{12}^2\right)'_{w_{qp}^1} =$$

$$= -\left[t - y\right] f'(net_1^2) f'(net_q^1) w_{1q}^2 \left(net_q^1\right)'_{w_{qp}^1} =$$

$$= -\left[t - y\right] f'(net_1^2) f'(net_q^1) w_{1q}^2 \left(x_0^1 w_{q0}^1 + x_1^1 w_{q1}^1 + x_2^1 w_{q2}^1\right)'_{w_{qp}^1} =$$

$$= -\left[t - y\right] f'(net_1^2) f'(net_q^1) w_{1q}^2 x_p^1 = -\delta_1^2 f'(net_q^1) w_{1q}^2 x_p^1 = -\delta_q^1 x_p^1$$

gdzie

$$\delta_q^1 = \delta_1^2 f'(net_q^1) w_{1q}^2.$$

Jeśli teraz sieć miałaby więcej warstw, to analogiczne obliczenia należałoby przeprowadzić dla kolejnych warstw poprzedzających te dwie, dla których właśnie wyprowadziliśmy odpowiednie wzory.

26.2 Algorytm

Mamy dany zbiór uczący L postaci $L = \{P, T\}$, gdzie $P = \{p_1, p_2, p_3, p_4\}$ oraz $T = \{t_1, t_2, t_3, t_4\}$. p_i oraz t_i, dla $i = 1, \ldots, 4$ określamy jak poniżej (argument 0 funkcji XOR został zastąpiony wartością -1)

$$\begin{aligned} p_1 &= \{-1, -1\} \\ p_2 &= \{-1, 1\} \\ p_3 &= \{1, -1\} \\ p_4 &= \{1, 1\} \\ t_1 &= \{0\} \\ t_2 &= \{1\} \\ t_3 &= \{1\} \\ t_4 &= \{0\} \end{aligned}$$

1. Wybór

   (a) $\eta > 0$ – współczynnik uczenia (np. 0.2),
   (b) $E_{max} > 0$ – maksymalny błąd jaki chcemy osiągnąć (np.0.01),
   (c) $C_{max} > 0$ – ilość kroków uczenia (np. 100).

2. Losowy wybór początkowych wartości wag jako niewielkich liczb (np. z przedziału $[-1, 1]$).
3. Przyjęcie $c := 1$.
4. Przyjęcie $E := 0$.
5. Uporządkowanie w losowej kolejności indeksów elementów zbioru P, otrzymując nowy zbiór I.
6. Wybór liczby k jako kolejnej liczby ze zbioru I.
7. Podanie k-tego obrazu ze zbioru P na wejscia neuronów z warstwy pierwszej.
8. Obliczenie sygnału wyjściowego z sieci, czyli y_1^2.
9. Obliczenie sygnałów delta

$$\delta_1^2 = (t - f(net_1^2))f'(net_1^2)$$

$$\delta_1^1 = \delta_1^2 w_{11}^2 f'(net_1^1)$$

$$\delta_2^1 = \delta_1^2 w_{12}^2 f'(net_2^1)$$

10. Uaktualnienie wartości wag według wzoru

$$w_{bc}^a = w_{bc}^a + \eta \delta_b^a x_c^a.$$

11. Obliczenie błędu $b = (t_k - y_1^2)$ oraz zaktualizowanie sumarycznej wartości błędu dla danego cyklu uczącego: $E = E + \frac{1}{2}b^2$.
12. Jeśli k nie jest ostatnim elementem to przejście do kroku 6.
13. Jeśli $E < E_{max}$, to koniec algorytmu.
14. Jeśli $c < C_{max}$, to $c := c + 1$ i przejście do kroku 4. W przeciwnym razie koniec algorytmu.

26.3 Zadanie

Należy zaimplementować zaprezentowany algorytm dla przedstawionego problemu XOR.

Rozdział 27

Scorch

27.1 Gra

Może niektórzy z Państwa mieli okazję grać w grę Scorch. Wątpię, gdyż jest to gra pamiętająca jeszcze czasy MS-DOS, ale może...

Gra w swoim pomyśle była wręcz banalna. Na ekranie ustawione są (widziane z boku) dwa czołgi. Zmieniając parametry strzału, a więc siłę i kat podniesienia działa, należy trafić przeciwnika. I to już wszystko. Dodatkową atrakcją był bardzo przemyślny arsenał środków bojowych, który czynił tę grę niesamowicie wciągającą.

I właśnie takiej gry (uproszczonej, ale nic nie stoi na przeszkodzie aby Państwo zadanie sobie utrudnili), dotyczy ten rozdział. W naszym przypadku przeciwnikiem będzie komputer sterowany przez sieć neuronową.

27.2 Założenia

Przyjmujemy, że na planszy, na tym samym poziomie stoja dwa czołgi w losowo dobieranej odległości. Jedyną akcją jaką możemy w stosunku do nich podjąć jest zmiana kąta podniesienia działa, przy stałej sile strzału (a dokładnie prędkości wylotowej v_0). Po ustaleniu przez gracza kąta strzelamy i sprawdzamy czy udało nam się trafić przciwnika. W tym miejscu przydaje się wzór na rzut ukośny:

$$x(t) = v_0 \cos(\alpha) t$$

$$y(t) = v_0 \sin(\alpha) t - \frac{1}{2} g t^2$$

gdzie:

- $\alpha \in [0, 45]$ – kąt nachylenia działa (patrz także punkt 27.4),
- v_0 – prędkość początkowa (stała, dobrana tak aby przy każdym kącie nachylenia działa tor lotu pocisku był krzywą),
- t – czas,
- g – przyspieszenie ziemskie (w przybliżeniu równe 9.81).

Wyznaczając z pierwszego równania zmienną t i podstawiając do drugiego otrzymujemy zależność y od x

$$y(x) = \tan(\alpha)x - \frac{g}{2v_0^2\cos^2(\alpha)}x^2.$$

Jeśli nie trafimy to próbuje nasz przeciwnik. I tak na zmianę aż ktoś uzyska trafienie. Każdy kolejny strzał komputera w ramch tej samej tury (z wyjątkiem pierwszego) jest niewielką modyfikacją poprzedniego strzału o losową wielkość. Po uzyskaniu trafienia (wszystko jedno przez kogo) zapamiętujemy odległość i parametry strzału dla jakich udało się trafić. W ten sposób otrzymujemy próbkę uczącą, którą dodajemy do naszego zbioru uczącego. Zbiór uczący będzie się powiększał, gdyż zawiera parametry celnych strzałów od początku gry. Po zakończeniu każdej tury uczymy sieć, która steruje kątem nachylenia działa czołgu kontrolowanego przez komputer.

27.3 Zadanie

Należy zaimplementować zaprezentowaną grę wykorzystując sieć neuronową do sterowania poczynaniami komputerowego przeciwnika. Sieć powinna składać się z co najmniej 2 warstw. Poza tym sieć musi mieć jedno wyjście. Ilość warstw (nie mniej jednak jak 2) oraz ilość neuronów należy ustalić samemu. Z uwagi na to, że stosowaną funkcją aktywacji jest funkcja sigmoidalna należy pamiętać o przeskalowaniu zbioru wartości funkcji na zakres zmienności kąta nachylenia działa założony w zadaniu tak, aby we właściwy sposób interpretować dane wyjściowe z sieci.

27.4 Uwagi do realizacji

Kąt α ograniczony został do przedziału $[0, 45]$. Jeśli jako przedział zmienności α przyjmiemy przedział $[0, 90]$, wówczas dla zadanej odległości istnieją dwa kąty, przy których można uzyskać trafienie: jeden mniejszy od 45 stopni a drugi większy. Mamy wówczas sytuację, gdy dane uczące mogą okazać się sprzeczne. Jeśli udział danych sprzecznych w ogólnej ilości danych uczących nie jest duży (kilka procent), to nie stanowi to problemu. Pouczającym doświadczeniem jest sprawdzenie jaka jest sensowna granica stosunku wszystkich danych do danych sprzecznych.

Chcąc dopuścić zmienność α w całym przedziale od 0 do 90 stopni, powinniśmy dodać do sieci dodatkowe wejście służące do wybierania rodzaju strzału: strzał płaski (kąty do 45 stopni) czy stromotorowy (powyżej 90 stopni). Kryterium wyboru jednego z rodzajów strzału mogło by być np. następujące. Najpierw próbujemy trafić przeciwnika oddając strzał płaski. Jeśli okaże się, że spowodowało to trafienie w cel inny od przeciwnika (np. górę znajdująca się na linii strzału), wówczas zaczynamy strzelać stromotorowo.

Oczywiście można jeszcze bardziej uogólnić zadanie przyjmując dwa zmienne parametry: kąt nachylenia działa i siłę strzału. Z tego co napisano powyżej wywnioskować można jednak, że tak postawione zadanie jest jeszcze trudniejsze z punktu widzenia uczenia sieci neuronowej, gdyż mamy jeszcze większą niejednoznaczność odpowiedzi: dla tych samych danych wejsciowych (odległość) mamy nieskończenie wiele par postaci *(siła,kąt)*.

Rozdział 28

Kompresja obrazu – sieć jednokierunkowa

28.1 Przedstawienie zadania

Sztuczne sieci neuronowe mogą okazać się całkiem przydatne, o czym postaramy się przekonać tym razem. Wykorzystamy je bowiem do zadania kompresji obrazu.

Kompresja obrazu możliwa do uzyskania przy pomocy sieci neuronowej jest kompresją stratną. Założenia jakie przyjmujemy wymienimy w kilku podpunktach.

Podział obrazu. Zwykle dzielimy obraz na znacznie mniejsze fragmenty nazywane blokami, które stanowią podstawową jednostkę poddawaną kompresji. Na potrzeby tego zadania przyjmujemy, że blok ma stały rozmiar wynoszący 8 na 8 pikseli a kompresowany obraz zarówno na wysokość jak i szerokość jest całkowitą wielokrotnością bloku (czyli zarówno wysokość jak i szerokość obrazka podzielna jest bez reszty przez 8, np. 256 na 256 pikseli). Założenie takie znacznie upraszcza implementację. Dodatkowo możemy przyjąć, że kompresowane obrazy zawsze mają taki sam wymiar.

Składowe RGB. Kolorowy obraz można opisać na wiele sposobów. Jedną z możliwości jest podanie nasycenia podstawowymi składowy przypadającymi na piksel, tj. czerwoną (R), zieloną (G) oraz niebieską (B). Nasycenie wyrażane jest przy pomocy liczby całkowitej z przedziału $[0, 255]$ a jeden piksel opisują trzy takie liczby. Będziemy traktować te składowe niezależnie. Można powiedzieć, że w istocie rozważamy trzy obrazy: obraz czerwony, zielony i niebieski. Kompresować będziemy niezależnie każdy z tych obrazów.

Wybór sieci. Użyjemy sieci skierowanej do przodu z wieloma warstwami.

Funkcja aktywacji. Jako funkcję aktywacji wybieramy funkcję sigmoidalną. Konsekwencją takiego wyboru jest konieczność przeskalowania danych będących wejściem i wyjściem z sieci do przedziału $[0.1, 0.9]$.

Struktura sieci. Struktura sieci ściśle odzwierciedla zdolności do kompresji. Przyjmujemy sieć o 64 wejściach, pewnej ilości neuronów w warstwie 1 – kompresującej (dla ustalenia uwagi przyjmijmy, że 32) oraz 64 neurony w warstwie 2 – wyjściowej.

Zauważmy, że ilość wejść jest równa ilości wyjść. Jest to chyba zrozumiałe jako, że nasza sieć ma przekształcać obraz oryginalny podany na wejście w możliwie najbardziej zbliżony do niego obraz na wyjściu, który jest wynikiem dekompresji. Rolę elementu kompresującego pełni warstwa 1 a warstwa 2 dekompresującego. Jeśli bowiem zapamiętamy wyjścia z warstwy 1 (czyli 32 liczby) wówczas mamy (teoretycznie) o połowę mniej informacji opisującej obraz. Jeśli teraz podamy na wejście warstwy 2 wektor złożony z 32 liczb, wówczas zostanie na jego podstawie odtworzony odpowiadający jemu wektor 64 liczbowy.

Tak więc trenując sieć celem kompresji przy jej pomocy obrazów jednocześnie trenujemy obie jej części – kompresującą i dekompresującą. Chcąc jej użyć używamy tylko jednej z jej warstw.

Zbiór uczący. Zbiór danych wejściowych zawiera 64-elementowe wektory nasycenia barw RGB powstałe w wyniku podziału obrazu. Zbiór oczekiwanych wartości wyjściowych z sieci pokrywa się ze zbiorem danych wejściowych. Jest tak dlatego, że żądamy aby obraz po kolejno wykonanych kompresji i dekompresji pozostał bez zmian.

Dyskretyzacja. Wyjście z warstwy 1 stanowi skompresowaną informację. Zauważmy, iż wyjście to stanowią 32 liczby rzeczywiste. Do zapamiętania liczby rzeczywistej potrzeba więcej niż 1 bajt – zwykle 4. Na wejściu sieci mamy zaś 64 liczby całkowite z przedziału $[0, 255]$ opisujące jedną ze składowych modelu RGB. Liczby te zapisać można na 1 bajcie. Prosty rachunek daje, że ilość informacji nieskompresowanej (wejściowej) wynosi 64 x 1 bajt = 64 bajty zaś skompresowanej 32 x 4bajty = 128 bajtów. Czyli po przeprowadzonej w ten sposób kompresji wzrośnie nam rozmiar obrazu skompresowanego.

Jedynym rozwiązaniem w takiej sytuacji jest zmniejszenie ilości bajtów na których zapisywana jest liczba rzeczywista. Najczęściej odbywa się to przez dyskretyzację liczb będących wynikiem kompresji.

Przystępując do dyskretyzacji musimy ustalić liczbę bitów jaką będziemy używać do zapisu liczby rzeczywistej. Następnie dzielimy przedział z jakiego pochodzą dyskretyzowane liczby na 2^n równych podprzedziałów, gdzie n jest liczbą bitów. Jeśli np. przyjmiemy $n = 2$ a przedziałem zmienności dla liczb rzeczywistych będzie $[0.0, 1.0]$ wówczas otrzymamy następujące podprzedziały

- $[0.0, 0.25)$ każda liczba rzeczywista należąca do tego przedziału będzie zapisywana jako 0;
- $[0.25, 0.5)$ każda liczba rzeczywista należąca do tego przedziału będzie zapisywana jako 1;
- $[0.5, 0.75)$ każda liczba rzeczywista należąca do tego przedziału będzie zapisywana jako 2;
- $[0.75, 1.0]$ każda liczba rzeczywista należąca do tego przedziału będzie zapisywana jako 3.

Teraz skompresowany obraz będzie miał rozmiar 32 x 2 bity = 64 bity = 8 bajtów co w porównaniu z wielkością informacji wejściowej równej 64 bajty daje kompresję 8-krotną.

Pamiętać należy o tym aby dyskretyzacji dokonywać także na etapie uczenia. Pominięcie tego może skutkować słabym dekompresowaniem obrazów gdyż sieć będzie uczyła się na liczbach rzeczywistych a pracować będzie na całkowitych będących przybliżeniem (często kiepskim) rzeczywistych.

Stopień kompresji. Na stopień kompresji mają wpływ następujące czynniki:

- stosunek ilości wejść do rozmiaru warstwy 1;
- dyskretyzacja wyjść z warstwy 1;
- wagi sieci. Niestety wagi są liczbami rzeczywistymi a również powinny być zapamiętane razem z obrazem. Im większy obraz, tym „narzut" związany z koniecznością ich zapamiętania okazuje się mniejszy. Jeśli jednak do kompresji używamy zawsze takiej samej sieci, wówczas możemy z zapamiętywania wag zrezygnować. Alternatywnym rozwiązaniem jest zapamiętywanie wag w osobnym pliku w sytuacji gdy tego samego zestawu wag używamy wielokrotnie. Wówczas poddając kompresji np. 10 obrazów otrzymujemy 10 plików skompresowanych i jeden, wspólny, z jednym zestawem wag.

28.2 Realizacja

Z założenia kompresji podlegać mają kolorowe obrazy zapisane przy pomocy trzech składowych RGB. Każdy obraz będziemy rozpatrywać trzykrotnie – ze względu na nasycenie każdej z barw: czerwonej, zielonej i niebieskiej. Zbiór uczący konstruujemy w następujący sposób:

1. Plik graficzny dzielimy na bloki o zadanym rozmiarze: 8×8 pikseli.
2. Każdy blok określa 3 wektory 64 liczbowe - wektor odpowiadający składowej czerwonej, zielonej i niebieskiej.
3. Zapamiętujemy te wektory – ich zbiór stanowić będzie zbiór uczący.

Uwaga. Całkiem dobre efekty można osiągnąć konstruując zbiór uczący w oparciu o jedną składową, ale warto poeksperymentować z tym samemu.

Mając dany zbiór uczący, po uprzednim określeniu rozmiaru warstwy kompresującej oraz stopnia dyskretyzacji, można przystąpić do uczenia sieci.

Nauczoną sieć wykorzystujemy do kompresji i dekompresji obrazu używając odpowiednio tylko jednej warstwy.

28.2.1 Zadanie

Należy zaimplementować sieć neuronową o strukturze opisanej powyżej. Sieć należy nauczyć przy pomocy jednego lub kilku wybranych obrazów, a następnie sprawdzić jak działa jako narzędzie kompresujące i dekompresujące. Testy wykonywać dla obrazów użytych w procesie uczenia oraz innych. Zbadać wpływ wielkości warstwy ukrytej oraz dokonywanej dyskretyzacji na straty kompresji. Na ile obraz odtworzony różni się od oryginalnego obliczymy według wzoru

$$MSE_{RGB} = \frac{(MSE_R + MES_G + MSE_B)}{3},$$

gdzie

$$MSE_X = \frac{1}{np} \sum_{l=1}^{p} \sum_{i=1}^{k} \sum_{j=1}^{k} (x_{ij}^{(l)} - \hat{x}_{ij}^{(l)})^2,$$

- x, $\hat{x}$ – odpowiednio wartość składowej bloku oryginalnej i odtworzonej,
- k – wymiar bloku,
- p – liczba bloków,
- n – wymiar wektora tworzącego blok,
- X – przyjmuje jedną z wartości: R, G lub B.

Rozdział 29

Sieć Hopfielda

29.1 Rekurencja w sieci

W sieciach neuronowych rozważanych do tej pory, wyjścia neuronów z pewnej warstwy mogły być połączone jedynie z wejściami neuronów warstwy następnej. Konsekwencją tego był jednokierunkowy przepływ informacji. Sygnały były przesyłane od warstwy wejściowej, przez pośrednie warstwy ukryte aż do warstwy wyjściowej. Stąd też nazwa tego rodzaju sieci – sieci jednokierunkowe (ang. *feedforward networks*).

Istnieją również sieci, w których dopuszcza się istnienie sygnałów sprzężenia zwrotnego, to znaczy sygnałów, które z wyjścia neuronu kierowane są na wejście neuronu z tej samej warstwy lub nawet wcześniejszej (patrz rysunek 29.1).

Rysunek 29.1: Model sieci Hopfielda.

Doskonałym polem zastosowań tego typu sieci są pamięci skojarzeniowe. Koncepcja pamięci skojarzeniowych wiąże się z jedną z podstawowych funkcji mózgu – zdolnością do odtworzenia pewnej informacji na podstawie jedynie jej zarysu a ściślej mówiąc jej zniekształconej postaci. Tak oto potrafimy rozpoznawać pismo, często bardzo odbiegające od wyuczonych w pierwszych latach szkoły „wzorcowych" liter. Rozwiązujemy krzyżówki, gdzie dysponując jedynie kilkoma literami potrafimy dopasować słowo. Potrafimy na zdjęciu rozpoznać znajomych nawet gdy znacznie odbiegają od znanych nam wzorców.

Wyróżniać będziemy dwa rodzaje tego typu pamięci:

- **pamięci autoasocjacyjne** – są to pamięci, gdzie na podstawie zaszumionego sygnału wejściowego odtwarzamy jego wzorcową, idealną postać (np. rozwiązywanie krzyżówki);

Rysunek 29.2: Działanie pamięci autoasocjacyjnej.

- **pamięci heteroasocjacyjne** – są to pamięci, gdzie na podstawie zaszumionego sygnału wejściowego odtwarzamy idealną postać, ale innego sygnału, który z tym jest skojarzony (związany).

Przyjrzymy się teraz tego typu pamięci w oparciu o najbardziej znane sieci rekurencyjne, nazywane sieciami Hopfielda.

Sieć zaproponowana przez Hopfielda składa się z n elementów przetwarzających (neuronów), pracujących asynchronicznie. Oznacza to, że w danej chwili zmieniany jest stan wyjścia tylko jednego (losowo wybranego) neuronu. Przyjmujemy, że funkcja wyjścia jest postaci:

$$y(t+1) = \begin{cases} +1 & \text{gdy } net > 0 \\ y(t) & \text{gdy } net = 0 \\ -1 & \text{gdy } net < 0 \end{cases}$$

a o macierzy wag zakładamy, że jest symetryczna i posiada zera na głównej przekątnej (zera na głównej przekątnej oznaczają, że wyjście żadnego z neuronów nie jest wejściem dla niego samego). Ponieważ każdy element jest połączony ze wszystkimi pozostałymi, w sieci Hopfielda nie ma wyraźnie wydzielonych warstw. Na ogół taką sieć przedstawia się jako jednowarstwową.

Wymiar wektora wejściowego jest równy ilości neuronów w sieci. Podając wektor wejściowy do sieci, podajemy każdą jego współrzędną jako wejście do jednego neuronu.

29.2 Algorytm uczenia

Jak we wszystkich sieciach neuronowych, cała wiedza również w tej sieci ukryta jest w wartościach wag. Istnieje kilka algorytmów pozwalających wyznaczyć wartości wag dla sieci Hopfielada mającej pracować jako pamięć autoasocjacyjna; przyjrzymy się tutaj najprostrzej z nich.

Zapisujemy do pamięci wektory wzorcowe s^m, $m = 1, ..., p$ o składowych -1 lub $+1$. Wagi wyznaczamy według wzoru:

$$w_{ij} = (1 - \delta_{ij}) \sum_{m=1}^{p} s_i^m s_j^m,$$

gdzie δ oznacza deltę Kroneckera:

$$\delta_{ij} = \begin{cases} 1 & gdy \quad i = j \\ 0 & gdy \quad i \neq j. \end{cases}$$

Uwaga 29.1.

Ważna jest zależność między ilością neuronów w sieci n a ilością zapamiętanych wzorców p. Przez pojemność sieci rozumiemy liczbę efektywnie zapamiętanych wzorców (patrz rozdział 20).

29.3 Algorytm odczytu

1. Ustalenie stanu początkowego przez podanie sygnału na wejścia neuronów; najczęściej jest to właśnie zaszumiona informacja, zniekształcony obraz itp.

2. Ustalenie losowej kolejnosci w jakiej neurony będą obliczały swoje sygnały wyjściowe.

3. Obliczenie sygnałów wyjściowych dla wszystkich neuronów (w kolejnosci ustalonej w poprzednim punkcie).

4. Jeśli dla żadnego neuronu nie nastąpiła zmiana sygnału wyjściowego $y(t)$ porównywanego z sygnałem wyjściowym $y(t-1)$, to kończymy algorytm. W przeciwnym razie wracamy do punktu 2.

Uwaga 29.2.

Podczas pracy algorytmu może się zdarzyć, że sieć odtworzy obraz będący dopełnieniem obrazu zapamiętanego (to znaczy zamiast -1 bedą $+1$ i odwrotnie). Jest to jak najbardziej „normalne" zachowanie dla tego typu sieci.

29.4 Zadanie

Zaimplementować zaprezentowany algorytm dla zadania odtwarzania liter. Przyjmujemy postać liter analogiczną jak w rozdziale 25. Do celów dydaktycznych wystarczy jak sieć zapamięta, powiedzmy 3 - 5 liter. Program powinien wczytywać zniekształconą literę i na jej podstawie odtworzyć zapamiętany obraz.

Rozdział 30

Sieć Elmana (metoda gradientowa dla zadania XOR)

Sieć tego typu wprowadziliśmy w podrozdziale 20.4. Przypomnijmy, że jest ona przykładem sieci neuronowej ze sprzężeniem zwrotnym. W swojej konstrukcji składa się ona z dwóch warstw. Pierwsza (wejściowa) jest warstwą, której sygnały wyjściowe są kierowane jako wejścia dla neuronów warstwy drugiej oraz jako sygnały zwrotne na wejścia tzw. neuronów kontekstowych. Wyjście z neuronów kontekstowych (które mają liniową funkcję aktywacji) stanowi wejście dla warstwy pierwszej (porównaj z rysunkeim 20.3 ze strony 213). Sygnały dla warstwy kontekstowej nie mają przypisanych wag lub inaczej, co na jedno wychodzi, wagi te wynoszą 1.0.

Jeżeli wejście do sieci jest n wymiarowe, oraz liczba neuronów w warstwie ukrytej wynosi n, to w kolejnym przejściu przez sieć wektor wejściowy będzie miał wymiar $n+k$. Pamiętając, że k sygnałów o indeksach od $n+1$ do $n+k$ pochodzi z poprzedniej iteracji otrzymujemy następujący wektor wejściowy $x(t)$

$$x(t) = [x_0(t), \ldots, x_n(t), x_{n+1}(t), \ldots, x_{n+k}(t)] = [x_0(t), \ldots, x_n(t), y_1(t-1), \ldots, y_k(t-1)]$$

gdzie $[y_1(t-1), \ldots, y_k(t-1)]$ jest wektorem wyjściowym warstwy pierwszej z poprzedniej (tj. $t-1$) iteracji. W czasie $t = 0$ przyjmujemy $[x_{n+1}(t), \ldots, x_{n+k}(t)] \equiv 0$.

W ćwiczeniu tym zastosujemy sieć Elmana do zadania XOR. Sieć Elmana rozwiązująca zadanie XOR będzie miała dwuwymiarowy wektor wejściowy danych zasadniczych plus bias, dwa neurony w warstwie ukrytej oraz dwa neurony kontekstowe. Ponadto sieć bezie miała jeden neuron w warstwie wyjściowej.

x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

Do nauki sieci Elmana zastosujemy gradientową metodę uczenia. Postępowanie w tym przypadku jest analogiczne jak w przypadku metody gradientowej stosowanej dla

sieci jednokierunkowej wielowarstwowej. Istotna różnica polega na modyfikowaniu wartości wag warstwy wejściowej, co jest związane ze sprzężeniem zwrotnym występującym w omawianej sieci.

Niech błąd sieci po podaniu jednego wzorca uczącego będzie dany wzorem

$$E = \frac{1}{2}(t - y_1^2)^2,$$

gdzie t jest elementem ze zbioru uczącego, zaś y_1^2 jest wyjściem z sieci (druga warstwa, pierwszy neuron). Ogólny wzór na modyfikowanie wag w metodzie gradientowej dany jest wzorem

$$w_{bc}^a = w_{bc}^a - \eta \frac{\partial E}{\partial w_{bc}^a}.$$

W naszym przypadku indeksy a, b oraz c przyjmują następujące wartości.

- Jeśli $a = 1$ to $b = 1, 2$, $c = 0, 1, 2, 3, 4$ (gdzie $c = 3, 4$ to wejścia pochodzące od neuronów kontekstowych).
- Jeśli $a = 2$ to $b = 1$, $c = 0, 1, 2$.

Wyznaczmy $\frac{\partial E}{\partial w_{bc}^a}$. W tym celu rozważamy dwa przypadki: pierwszy dla wag warstwy wyjściowej i drugi dla wag warstwy wejściowej.

Dla **wag warstwy wyjściowej** wartość $\frac{\partial E}{\partial w_{bc}^a}$ będzie taka sama jak w przypadku propagacji wstecznej dla sieci jednokierunkowej wielowarstwowej. Mamy wówczas

$$\frac{\partial E}{\partial w_{1i}^2} = -(t - y_1^2) f'(net_1^2) \cdot x_i^2, \quad i = 0, 1, 2.$$

Dla **wag warstwy ukrytej** wartość $\frac{\partial E}{\partial w_{bc}^1}$ będzie liczona z uwzględnieniem sprzężenia zwrotnego. Rozważmy najpierw konkretny przypadek dla ustalonej wagi, np. w_{21}^1. Mamy wówczas

$$\frac{\partial E}{\partial w_{21}^1} = \left[\frac{1}{2}\left(t - y_1^2(t)\right)^2\right]\Big|_{w_{21}^1} = \left[\frac{1}{2}\left(t - f(net_1^2(t))\right)^2\right]\Big|_{w_{21}^1} =$$

$$-(t - f(net_1^2(t))) \cdot f(net_1^2(t))\big|_{w_{21}^1} = -(t - f(net_1^2(t))) \cdot f'(net_1^2(t)) \cdot net_1^2(t)\big|_{w_{21}^1} =$$

$$-(t - f(net_1^2(t))) \cdot f'(net_1^2(t)) \cdot \underbrace{\left(\sum_{i=0}^{2} x_i^2(t) w_{1i}^2\right)}_{net_1^2(t)}\Bigg|_{w_{21}^1} = \ldots$$

W tym miejscu kończą się obliczenia dla warstwy wyjściowej a my liczymy dalej. Ponieważ

$$x_i^2(t) = y_i^1 \quad \text{dla} \quad i = 1, 2$$

(to znaczy: sygnał wejściowy dla warstwy drugiej jest tym samym co wyjściowy z warstwy pierwszej) oraz

$$y_i^1(t) = f\underbrace{\left(\sum_{j=0}^{4} x_j^1(t) w_{ij}^1\right)}_{net_i^1(t)} = f\left(\sum_{j=0}^{2} x_j^1(t) w_{ij}^1 + \sum_{j=1}^{2} y_j^1(t-1) w_{i2+j}^1\right) \tag{30.1}$$

zatem

$$\cdots = -(t - f(net_1^2(t))) \cdot f'(net_1^2(t)) \cdot \left(w_{10}^2 + x_1^2(t)w_{11}^2 + x_2^2(t)w_{12}^2\right)\Big|_{w_{21}^1} =$$

$$-(t - f(net_1^2(t))) \cdot f'(net_1^2(t)) \cdot \left(y_1^1(t)w_{11}^2 + y_2^1(t)w_{12}^2\right)\Big|_{w_{21}^1} =$$

$$-(t - f(net_1^2(t))) \cdot f'(net_1^2(t)) \cdot \left(f(net_1^1(t))w_{11}^2 + f(net_2^1(t))w_{12}^2\right)\Big|_{w_{21}^1} = \ldots$$

Ze wzoru (30.1) widać, że zarówno $net_1^1(t)$ jak i $net_2^1(t)$ zależą od w_{21}^1 przez sygnały sprzężenia zwrotnego, więc

$$\cdots = -(t - f(net_1^2(t))) \cdot f'(net_1^2(t)) \cdot \left(w_{11}^2 f'(net_1^1(t))\, net_1^1(t)\Big|_{w_{21}^1} + w_{12}^2 f'(net_2^1(t))\, net_2^1(t)\Big|_{w_{21}^1}\right)$$

Otrzymalismy więc

$$\frac{\partial E}{\partial w_{21}^1} = -(t - f(net_1^2(t))) \cdot f'(net_1^2(t)) \cdot \left(w_{11}^2 f'(net_1^1(t))\, net_1^1(t)\Big|_{w_{21}^1} + w_{12}^2 f'(net_2^1(t))\, net_2^1(t)\Big|_{w_{21}^1}\right) \tag{30.2}$$

Policzmy teraz ile wynosi $net_1^1(t)\big|_{w_{21}^1}$ oraz $net_2^1(t)\big|_{w_{21}^1}$. Najpierw $net_1^1(t)\big|_{w_{21}^1}$

$$net_1^1(t)\Big|_{w_{21}^1} = \left(\sum_{j=0}^{2} x_j^1(t)w_{1j}^1 + \sum_{j=1}^{2} y_j^1(t-1)w_{12+j}^1\right)\Bigg|_{w_{21}^1} = 0 + w_{13}^1 y_1^1(t-1)\Big|_{w_{21}^1} + w_{14}^1 y_2^1(t-1)\Big|_{w_{21}^1} =$$

$$w_{13}^1 f(net_1^1(t-1))\Big|_{w_{21}^1} + w_{14}^1 f(net_2^1(t-1))\Big|_{w_{21}^1} =$$

$$w_{13}^1 f'(net_1^1(t-1)) net_1^1(t-1)\Big|_{w_{21}^1} + w_{14}^1 f'(net_2^1(t-1)) net_2^1(t-1)\Big|_{w_{21}^1}$$

A więc mamy zależność

$$net_1^1(t)\Big|_{w_{21}^1} = w_{13}^1 f'(net_1^1(t-1)) net_1^1(t-1)\Big|_{w_{21}^1} + w_{14}^1 f'(net_2^1(t-1)) net_2^1(t-1)\Big|_{w_{21}^1} \tag{30.3}$$

Zajmijmy się teraz $net_2^1(t)\big|_{w_{21}^1}$

$$net_2^1(t)\Big|_{w_{21}^1} = \left(\sum_{j=0}^{2} x_j^1(t)w_{2j}^1 + \sum_{j=1}^{2} y_j^1(t-1)w_{22+j}^1\right)\Bigg|_{w_{21}^1} =$$

$$x_1^1(t) + w_{23}^1 y_1^1(t-1)\Big|_{w_{21}^1} + w_{24}^1 y_2^1(t-1)\Big|_{w_{21}^1} =$$

$$x_1^1(t) + w_{23}^1 f(net_1^1(t-1))\Big|_{w_{21}^1} + w_{24}^1 f(net_2^1(t-1))\Big|_{w_{21}^1} =$$

$$x_1^1(t) + w_{23}^1 f'(net_1^1(t-1)) net_1^1(t-1)\Big|_{w_{21}^1} + w_{24}^1 f'(net_2^1(t-1)) net_2^1(t-1)\Big|_{w_{21}^1}$$

A więc mamy zależność

$$net_2^1(t)\Big|_{w_{21}^1} = x_1^1(t) + w_{23}^1 f'(net_1^1(t-1)) net_1^1(t-1)\Big|_{w_{21}^1} + w_{24}^1 f'(net_2^1(t-1)) net_2^1(t-1)\Big|_{w_{21}^1} \tag{30.4}$$

Wzór (30.2) wraz z rekurencyjnymi zależnościami (30.3) i (30.4) pozwala nam modyfikować wagę w_{21}^1. Zauważmy, że otrzymujemy w tym przypadku zależność rekurencyjną na pochodną jak w przypadku sieci RTRN (patrz podrozdział 20.2). W ogólności wspomniane wzory występują w postaciach

- ogólna postać wzoru (30.2)

$$\frac{\partial e_1^1}{\partial w_{bc}^1} = -(t - f(net_1^2(t))) \cdot f'(net_1^2(t)) \cdot \left(w_{11}^2 f'(net_1^1(t))\ net_1^1(t)\big|_{w_{bc}^1} + w_{12}^2 f'(net_2^1(t))\ net_2^1(t)\big|_{w_{bc}^1} \right)$$

- ogólna postać wzoru (30.3)

$$net_1^1(t)\big|_{w_{bc}^1} = x_c^1 \delta_{1b} + w_{13}^1 f'(net_1^1(t-1)) net_1^1(t-1)\big|_{w_{bc}^1} + w_{14}^1 f'(net_2^1(t-1)) net_2^1(t-1)\big|_{w_{bc}^1}$$

- ogólna postać wzoru (30.4)

$$net_2^1(t)\big|_{w_{bc}^1} = x_c^1 \delta_{2b} + w_{23}^1 f'(net_1^1(t-1)) net_1^1(t-1)\big|_{w_{bc}^1} + w_{24}^1 f'(net_2^1(t-1)) net_2^1(t-1)\big|_{w_{bc}^1}$$

Rozdział 31

Sieć Elmana (metoda Levenberga - Marquardta dla zadania XOR)

Algorytm Levenberga-Marquardta (LM) został zaprezentowany w podrozdziale **??** i poświęcone zostało jemu ćwiczenie 35. Istniejące opis odnoszą się do sieci jednokierunkowej wielowarstwowej, jednak ten sam algorytm może być z powodzeniem stosowany dla sieci Elmana. Z uwagi na analogie w przypadku realizacji algorytmu z wymienionym wcześniej materiałem, podamy tutaj jedynie wskazówki pomocne przy jego implementacji. Celem dokonania porównania z innymi zbadanymi przypadkami, problemem jaki przy jego pomocy rozwiążemy będzie problem XOR (jego opis pojawił się w rozdziale 26)[1].

Przyjmowany model sieci przedstawia rysunek **??**. Wynika z niego, że sygnały wyjściowe z neuronów opisane są przez zależności

$$\begin{aligned} y_1^1 &= f(x_0^1 w_{10}^1 + x_1^1 w_{11}^1 + x_2^1 w_{12}^1 + x_{13}^1 w_{13}^1 + x_4^1 w_{14}^1), \\ y_2^1 &= f(x_0^1 w_{20}^1 + x_1^1 w_{21}^1 + x_2^1 w_{22}^1 + x_{13}^1 w_{23}^1 + x_4^1 w_{24}^1), \\ y_1^2 &= f(x_0^2 w_{10}^2 + x_1^2 w_{11}^2 + x_2^2 w_{12}^2), \end{aligned}$$

gdzie

$$\begin{aligned} x_{13}^1(t) &= y_1^1(t-1), \\ x_{14}^1(t) &= y_2^1(t-1) \end{aligned}$$

są sygnałami wyjściowymi neuronów pierwszej warstwy z poprzedniej iteracji. Dodatkowo zakłada się, że w czasie $t = 0$ zachodzi

$$y_1^1(t) = y_2^1(t) = 0.$$

Ponadto sygnał $y_1^1 = x_1^2$ jest wyjściem z neuronu pierwszego warstwy pierwszej, ale jednocześnie jest pierwszym sygnałem wejściowym warstwy drugiej. Podobna zależność zachodzi dla sygnału y_2^1, który jest drugim sygnałem wejściowym warstwy drugiej (x_2^2). Funkcja f jest sigmoidalną funkcją aktywacji.

Zbiór uczący L składa się z następujących par

$$\begin{aligned} (p_1, t_1) &= ((0,0),(0)), \\ (p_2, t_2) &= ((0,1),(1)), \\ (p_3, t_3) &= ((1,0),(1)), \\ (p_4, t_4) &= ((1,1),(0)). \end{aligned}$$

[1] Problem ten nie jest najwłaściwszym problemem dla sieci Elmana bo nie występują w nim zależności (czasowe) pomiędzy danymi uczącymi.

Wektor błędu poszczególnych próbek uczących na każdym neuronie w warstwie wyjściowej (wzór **??**) będzie miał postać[2]

$$e(w) = \begin{bmatrix} e_1^1 \\ e_1^2 \\ e_1^3 \\ e_1^4 \end{bmatrix} \tag{31.1}$$

czyli

$$e(w) = \begin{bmatrix} t_1 - y_1^2(p_1) \\ t_2 - y_1^2(p_2) \\ t_3 - y_1^2(p_3) \\ t_4 - y_1^2(p_4) \end{bmatrix}$$

gdzie e_o^l, $o \in \{1\}$, $l \in \{1,2,3,4\}$ oznacza błąd l-tej próbki w o-tym neuronie wyjściowym. Zapis $y_1^2(p_l)$ rozumiemy jako wyjście z sieci otrzymane po prezentacji wzorca p_l.

Mając postać wektora $e(w)$ musimy znaleźć teraz postać macierzy $J(w)$. Ponieważ mamy łącznie trzynaście wag, zatem jej postać jest następująca

$$J(w) = \left[\begin{array}{ccccccccccc|ccc} \frac{\partial e_1^1}{\partial w_{10}^1} & \frac{\partial e_1^1}{\partial w_{11}^1} & \frac{\partial e_1^1}{\partial w_{12}^1} & \frac{\partial e_1^1}{\partial w_{13}^1} & \frac{\partial e_1^1}{\partial w_{14}^1} & \frac{\partial e_1^1}{\partial w_{20}^1} & \frac{\partial e_1^1}{\partial w_{21}^1} & \frac{\partial e_1^1}{\partial w_{22}^1} & \frac{\partial e_1^1}{\partial w_{23}^1} & \frac{\partial e_1^1}{\partial w_{24}^1} & \frac{\partial e_1^1}{\partial w_{10}^2} & \frac{\partial e_1^1}{\partial w_{11}^2} & \frac{\partial e_1^1}{\partial w_{12}^2} \\ \frac{\partial e_1^2}{\partial w_{10}^1} & \frac{\partial e_1^2}{\partial w_{11}^1} & \frac{\partial e_1^2}{\partial w_{12}^1} & \frac{\partial e_1^2}{\partial w_{13}^1} & \frac{\partial e_1^2}{\partial w_{14}^1} & \frac{\partial e_1^2}{\partial w_{20}^1} & \frac{\partial e_1^2}{\partial w_{21}^1} & \frac{\partial e_1^2}{\partial w_{22}^1} & \frac{\partial e_1^2}{\partial w_{23}^1} & \frac{\partial e_1^2}{\partial w_{24}^1} & \frac{\partial e_1^2}{\partial w_{10}^2} & \frac{\partial e_1^2}{\partial w_{11}^2} & \frac{\partial e_1^2}{\partial w_{12}^2} \\ \frac{\partial e_1^3}{\partial w_{10}^1} & \frac{\partial e_1^3}{\partial w_{11}^1} & \frac{\partial e_1^3}{\partial w_{12}^1} & \frac{\partial e_1^3}{\partial w_{13}^1} & \frac{\partial e_1^3}{\partial w_{14}^1} & \frac{\partial e_1^3}{\partial w_{20}^1} & \frac{\partial e_1^3}{\partial w_{21}^1} & \frac{\partial e_1^3}{\partial w_{22}^1} & \frac{\partial e_1^3}{\partial w_{23}^1} & \frac{\partial e_1^3}{\partial w_{24}^1} & \frac{\partial e_1^3}{\partial w_{10}^2} & \frac{\partial e_1^3}{\partial w_{11}^2} & \frac{\partial e_1^3}{\partial w_{12}^2} \\ \frac{\partial e_1^4}{\partial w_{10}^1} & \frac{\partial e_1^4}{\partial w_{11}^1} & \frac{\partial e_1^4}{\partial w_{12}^1} & \frac{\partial e_1^4}{\partial w_{13}^1} & \frac{\partial e_1^4}{\partial w_{14}^1} & \frac{\partial e_1^4}{\partial w_{20}^1} & \frac{\partial e_1^4}{\partial w_{21}^1} & \frac{\partial e_1^4}{\partial w_{22}^1} & \frac{\partial e_1^4}{\partial w_{23}^1} & \frac{\partial e_1^4}{\partial w_{24}^1} & \frac{\partial e_1^4}{\partial w_{10}^2} & \frac{\partial e_1^4}{\partial w_{11}^2} & \frac{\partial e_1^4}{\partial w_{12}^2} \end{array}\right] \tag{31.2}$$

Pionową kreską rozdzielono pochodne cząstkowe liczone według różnych schematów. Zajmijmy się najpierw wagami warstwy wyjściowej (grupa po prawej stronie pionowej kreski) a więc wagami postaci $\frac{\partial e_1^l}{\partial w_{ij}^2}$, $l \in \{1,2,3,4\}$, $i \in \{1\}$, $j \in \{0,1,2\}$. Policzmy np. pochodną $\frac{\partial e_1^2}{\partial w_{11}^2}$

$$\frac{\partial e_1^2}{\partial w_{11}^2} = (t_2 - y_1^2(p_2))'_{w_{11}^2} = (t_2 - f(net_1^2(p_2))'_{w_{11}^2} = -f'(net_1^2(p_2))(net_1^2(p_2))'_{w_{11}^2} =$$

$$-f'(net_1^2(p_2))(x_0^2(p_2)w_{10}^2 + x_1^2(p_2)w_{11}^2 + x_2^2(p_2)w_{12}^2)'_{w_{11}^2} = -f'(net_1^2(p_2))x_1^2(p_2)$$

gdzie np. zapis $net_1^2(p_2)$ oznacza wartość net_1^2 policzoną dla sygnału p_2. Uogólniając otrzymany wynik na dowolną wagę warstwy wyjściowej w_{1i}^2, $i \in \{0,1,2\}$ dla l-tej próbki, otrzymujemy ogólny schemat dla tych wag

$$\frac{\partial e_1^l}{\partial w_{1i}^2} = -f'(net_1^2(p_l))x_i^2(p_l).$$

Zajmijmy się teraz wagami warstwy pierwszej (grupa po lewej stronie pionowej kreski) a więc wagami postaci $\frac{\partial e_1^l}{\partial w_{ij}^2}$, $l \in \{1,2,3,4\}$, $i \in \{1,2\}$, $j \in \{0,\ldots,4\}$.

[2]Mamy jeden neuron i cztery próbki uczące.

Dla wag warstwy ukrytej wartość $\frac{\partial e_1^1}{\partial w_{bc}^1}$ będzie liczona z uwzględnieniem sprzężenia zwrotnego. Rozważmy najpierw konkretny przypadek dla wagi ustalonej wagi, np. w_{21}^1. Mamy wówczas

$$\frac{\partial e_1^1}{\partial w_{21}^1} = \left[t - y_1^2(t)\right]\Big|_{w_{21}^1} = \left[t - f(net_1^2(t))\right]\Big|_{w_{21}^1} =$$

$$-f(net_1^2(t))\Big|_{w_{21}^1} = -f'(net_1^2(t)) \cdot net_1^2(t)\Big|_{w_{21}^1} =$$

$$-f'(net_1^2(t)) \cdot \underbrace{\left(\sum_{i=0}^{2} x_i^2(t) w_{1i}^2\right)}_{net_1^2(t)}\Bigg|_{w_{21}^1} = \ldots$$

W tym miejscu kończą się obliczenia dla warstwy wyjściowej a my liczymy dalej. Ponieważ

$$x_i^2(t) = y_i^1 \quad \text{dla} \quad i = 1, 2$$

(to znaczy: sygnał wejściowy dla warstwy drugiej jest tym samym co wyjściowy z warstwy pierwszej) oraz

$$y_i^1(t) = f\underbrace{\left(\sum_{j=0}^{4} x_j^1(t) w_{ij}^1\right)}_{net_i^1(t)} = f\left(\sum_{j=0}^{2} x_j^1(t) w_{ij}^1 + \sum_{j=1}^{2} y_j^1(t-1) w_{i2+j}^1\right) \tag{31.3}$$

zatem

$$\cdots = -f'(net_1^2(t)) \cdot \left(w_{10}^2 + x_1^2(t) w_{11}^2 + x_2^2(t) w_{12}^2\right)\Big|_{w_{21}^1} =$$

$$-f'(net_1^2(t)) \cdot \left(f(net_1^1(t)) w_{11}^2 + f(net_2^1(t)) w_{12}^2\right)\Big|_{w_{21}^1} = \ldots$$

Ze wzoru (31.3) widać, że zarówno $net_1^1(t)$ jak i $net_2^1(t)$ zależą od w_{21}^1 przez sygnały sprzężenia zwrotnego, więc

$$\cdots = -f'(net_1^2(t)) \cdot \left(w_{11}^2 f'(net_1^1(t))\, net_1^1(t)\Big|_{w_{21}^1} + w_{12}^2 f'(net_2^1(t))\, net_2^1(t)\Big|_{w_{21}^1}\right)$$

Otrzymalismy więc

$$\frac{\partial e_1^1}{\partial w_{21}^1} = -f'(net_1^2(t)) \cdot \left(w_{11}^2 f'(net_1^1(t))\, net_1^1(t)\Big|_{w_{21}^1} + w_{12}^2 f'(net_2^1(t))\, net_2^1(t)\Big|_{w_{21}^1}\right) \tag{31.4}$$

Policzmy teraz ile wynosi $net_1^1(t)\big|_{w_{21}^1}$ oraz $net_2^1(t)\big|_{w_{21}^1}$. Najpierw $net_1^1(t)\big|_{w_{21}^1}$

$$net_1^1(t)\Big|_{w_{21}^1} = \left(\sum_{j=0}^{2} x_j^1(t) w_{1j}^1 + \sum_{j=1}^{2} y_j^1(t-1) w_{12+j}^1\right)\Bigg|_{w_{21}^1} = 0 + w_{13}^1 y_1^1(t-1)\Big|_{w_{21}^1} + w_{14}^1 y_2^1(t-1)\Big|_{w_{21}^1} =$$

$$w_{13}^1 f(net_1^1(t-1))\Big|_{w_{21}^1} + w_{14}^1 f(net_2^1(t-1))\Big|_{w_{21}^1} =$$

$$w_{13}^1 f'(net_1^1(t-1)) net_1^1(t-1)\Big|_{w_{21}^1} + w_{14}^1 f'(net_2^1(t-1)) net_2^1(t-1)\Big|_{w_{21}^1}$$

A więc mamy zależność

$$net_1^1(t)\big|_{w_{21}^1} = w_{13}^1 f'(net_1^1(t-1))net_1^1(t-1)\big|_{w_{21}^1} + w_{14}^1 f'(net_2^1(t-1))net_2^1(t-1)\big|_{w_{21}^1} \tag{31.5}$$

Zajmijmy się teraz $net_2^1(t)\big|_{w_{21}^1}$

$$net_2^1(t)\big|_{w_{21}^1} = \left(\sum_{j=0}^{2} x_j^1(t)w_{2j}^1 + \sum_{j=1}^{2} y_j^1(t-1)w_{22+j}^1\right)\Bigg|_{w_{21}^1} =$$

$$x_1^1(t) + w_{23}^1 y_1^1(t-1)\big|_{w_{21}^1} + w_{24}^1 y_2^1(t-1)\big|_{w_{21}^1} =$$

$$x_1^1(t) + w_{23}^1 f(net_1^1(t-1))\big|_{w_{21}^1} + w_{24}^1 f(net_2^1(t-1))\big|_{w_{21}^1} =$$

$$x_1^1(t) + w_{23}^1 f'(net_1^1(t-1))net_1^1(t-1)\big|_{w_{21}^1} + w_{24}^1 f'(net_2^1(t-1))net_2^1(t-1)\big|_{w_{21}^1}$$

A więc mamy zależność

$$net_2^1(t)\big|_{w_{21}^1} = x_1^1(t) + w_{23}^1 f'(net_1^1(t-1))net_1^1(t-1)\big|_{w_{21}^1} + w_{24}^1 f'(net_2^1(t-1))net_2^1(t-1)\big|_{w_{21}^1} \tag{31.6}$$

Wzór (31.4) wraz z rekurencyjnymi zależnościami (31.5) i (31.6) pozwala nam modyfikować wagę w_{21}^1. Zauważmy, że otrzymujemy w tym przypadku zależność rekurencyjną na pochodną. W ogólności wspomniane wzory występują w postaciach

- ogólna postać wzoru (31.4)

$$\frac{\partial e_1^1}{\partial w_{bc}^1} = -f'(net_1^2(t)) \cdot \left(w_{11}^2 f'(net_1^1(t))\ net_1^1(t)\big|_{w_{bc}^1} + w_{12}^2 f'(net_2^1(t))\ net_2^1(t)\big|_{w_{bc}^1}\right)$$

- ogólna postać wzoru (31.5)

$$net_1^1(t)\big|_{w_{bc}^1} = x_c^1\delta_{1b} + w_{13}^1 f'(net_1^1(t-1))net_1^1(t-1)\big|_{w_{bc}^1} + w_{14}^1 f'(net_2^1(t-1))net_2^1(t-1)\big|_{w_{bc}^1}$$

- ogólna postać wzoru (31.6)

$$net_2^1(t)\big|_{w_{bc}^1} = x_c^1\delta_{2b} + w_{23}^1 f'(net_1^1(t-1))net_1^1(t-1)\big|_{w_{bc}^1} + w_{24}^1 f'(net_2^1(t-1))net_2^1(t-1)\big|_{w_{bc}^1}$$

Pozostaje jeszcze do wyjaśnienia sprawa odpowiednich wymiarów macierzy i wektorów we wzorze **??** ze strony **??**. Ze wzoru (31.1) widzimy, że wektor e jest wymiaru 4×1 natomiast ze wzoru (31.2) widzimy, że macierz J jest wymiaru 4×13. Jeśli więc przyjmiemy, że wektor wagowy w jest wektorem kolumnowym postaci

$$w = \begin{bmatrix} w_{10}^1 \\ w_{11}^1 \\ w_{12}^1 \\ w_{13}^1 \\ w_{14}^1 \\ w_{20}^1 \\ w_{21}^1 \\ w_{22}^1 \\ w_{23}^1 \\ w_{24}^1 \\ w_{10}^2 \\ w_{11}^2 \\ w_{12}^2 \end{bmatrix}$$

wówczas wymiary we wzorze (**??**) są zgodne

$$\underbrace{w(t+1)}_{13\times1} = \underbrace{w(t)}_{13\times1} - \underbrace{[\underbrace{\underbrace{J^T}_{13\times4}\underbrace{J}_{4\times13}}_{13\times13} + \underbrace{\eta I}_{13\times13}]^{-1}\underbrace{\underbrace{J^T}_{13\times4}\underbrace{e}_{4\times1}}_{13\times1}}_{13\times1}$$

Powyższe wyjaśnienia powinny rozwiać wszelkie wątpliwoście związane z praktyczną implementacją algorytmu LM dla postawionego problemu.

Rozdział 32

Sieci samoorganizujące się na zasadzie współzawodnictwa

32.1 Samoorganizacja

Obserwując działanie poprzednich programów mieliśmy okazję przekonać się, że sieć neuronowa jest w stanie nauczyć się pewnego odwzorowania z przestrzeni sygnałów wejściowych w przestrzeń sygnałów wyjściowych, korzystając z informacji o właściwych (oczekiwanych) odpowiedziach.

Pokażemy teraz, że sieć potrafi wykryć pewne związki bez dodatkowej pomocy z zewnątrz, czyli bez posiadania informacji o prawidłowej klasyfikacji danych wejściowych. W tym celu wykorzystywać będziemy mechanizm konkurencji. Po obliczeniu sygnałów wyjściowych neuronów, wybierzemy z nich tego o najsilniejszej odpowiedzi (to znaczy o największej wartości liczbowej na wyjściu). Jego i tylko jego wagi zmodyfikujemy następnie tak, aby po ponownej prezentacji tego wzorca, jego sygnał wyjściowy miał jeszcze większą wartość w stosunku do pozostałych niż ma to miejsce obecnie. Wagi pozostałych neuronów pozostaną bez zmian. Mechanizm ten, znany pod nazwą **WTA** (ang. *winner takes all – wygrywający bierze wszystko*), wymaga aby wagi i sygnały wejściowe były znormalizowane. Sieci tego typu są jednowarstwowe, często z liniową funkcją aktywacji; neurony nie mają biasu. Ilość neuronów wyznacza zdolności do wykrywania grup – maksymalna ilość wykrytych grup nie może przekroczyć ilości neuronów.

Sieci samouczące[1] wykazują swoją przydatność wszędzie tam, gdzie nie możemy z góry udzielić informacji dotyczących przetwarzanych danych, lub właśnie tych informacji poszukujemy. Sieci te najczęściej wykorzystywane są do:

- grupowania sygnałów wejściowych i łączenia ich w klasy;
- wykrywania zależności pomiędzy znalezionymi grupami/klasami;
- wykrywania własności statystycznych danych wejściowych przez przeznaczenie większej ilości neuronów do klasyfikowania częściej pojawiających się danych;
- poznanie topologii danych wejsciowych; neurony położone blisko siebie odpowiadają na podobne sygnały.

[1] Ze względu na sposób działania właśnie termin *samouczące* wydaje się bardziej właściwy niż *samoorganizujące*, choć ten drugi, dokładnie w tym kontekście, wykorzystywany jest znacznie częście.

32.2 Algorytm

1. Ustalenie struktury sieci. Decydujemy ile neuronów ma posiadać sieć – determinuje to zdolności sieci do wykrywania różnej ilości grup (jeden neuron odpowiada jednej grupie; może się zdarzyć, że różne neurony będą odpowiadać jednej grupie wzorców a także, że kilku grupom wzorców będzie odpowiadał jeden neuron).

2. Wylosowanie wartości wag dla każdego neuronu. Wagi neuronu, tworzące wektor, możemy interpretować jako punkt (neuron) w pewnej przestrzeni.

3. Czasem wymagane jest dokonanie normalizacji wag i sygnałów wejściowych (wzorców). Można tego dokonać np. według wzoru:

$$x_i = \frac{x_i}{\sqrt{\sum_{j=1}^{n} x_j^2}},$$

   gdzie n jest ilością wejść, $i = 1, \ldots, n$ indeksem współrzędnej.

4. Powtarzamy założoną ilość razy następujące kroki:

   (a) Wybierz losowo wzorzec p.

   (b) Oblicz dla wszystkich neuronów sygnały wyjściowe dla wzorca p. Sposób liczenia wartości na wyjściu może być różny. Na przykład może to być iloczyn skalarny wag i sygnałów wejściowych lub odległość wektora wag i syganałów wejściowych.

   (c) Znajdź neuron o największej wartości sygnału na wyjściu. Oznacz go jako *winner*. Jeśli takich neuronów jest kilka, to wybieramy tylko jednego z nich (w dowolny sposób: losowo, pierwszy, ostatni itp).

   (d) Zmień wagi neuronu *winner* tak aby były bliższe wzorcowi p według wzoru:

$$w_i = w_i + \eta[x_i - w_i],$$

   gdzie $i = 1, \ldots, n$, n jest ilością wejść, η jest współczynnikiem uczenia. Współczynnik uczenia należy zmniejszać wraz z przebiegiem procesu uczenia. Na początku może być większy, na przykład 0.2, a następnie powinien dążyć do zera.

5. Na zakończenie sprawdzamy do jakich grup należą rozważane punkty. W tym celu każdemu punktowi przypisujemy numer neuronu, który zareagował najsilniej.

32.3 Sąsiedztwo

Bardzo często zdarza się, że metoda WTA nie daje nam dobrego rozwiązania, w sensie przyporządkowania jednego neuronu do jednej grupy. Przykłady takich sytuacji przedstawiono na rysunkach **??**.

W takiej sytuacji bardzo dobre efekty przynosi złagodzona wersja metody WTA zwanej **WTM** (ang. *winner takes most – wygrywający bierze więcej*). Różnica polega na tym, że oprócz wag neuronu zwycięskiego, modyfikowane są także wagi wszystkich neuronów sąsiednich. Sąsiedztwo neuronów można określić na wiele sposobów.

- Jako odległość euklidesowa pomiędzy wektorami wag neuronu zwycięzcy i neuronu kandydującego do miana sąsiada. Wówczas, jeśli odległość ta jest mniejsza od pewnej ustalonej liczby, zwanej promieniem sąsiedztwa, neuron kandyujący staje się sąsiadem. Wagi sąsiada modyfikowane są w kierunku tego samego punktu, który powoduje modyfikowanie wag zwycięzcy. Stopień modyfikacji jest jednak tym mniejszy im dalszym sąsiadem okazuje się neuron kandydujący.

- Jako pewna relacja wiążąca ze sobą neurony, które nie koniecznie muszą być „blisko" siebie. Można na przykład przyjąć, że sąsiadem neuronu 1 jest neuron 5, neuronu 2 jest neuron 3 a neuron 4 nie ma sąsiada. Przyjęcie takiego sąsiedztwa prowadzi do sieci Kohonena i znanych jako efekt ich wykorzystania map Kohonena.

Jak więc widać odległości pomiędzy neuronami można wyznaczyć zarówno w odniesieniu do metryki zdefiniowanej w przestrzeni sygnałów wejściowych, jak również w odniesieniu do topologii samej sieci. To drugie podejście pozwala określać sąsiedztwo danego neuronu w postaci zbioru neuronów powiązanych między sobą odpowiednimi relacjami topologicznymi (relacjami zachowującymi strukturę sieci).

32.4 Zadanie

Przedmiotem naszych rozważań będą punkty pochodzące obszaru $[-150, 150]\times[-150, 150]$ pikseli. W obszarze tym umieszczamy kilkanście punktów tworzących zbiór P w kilku skupiskach oraz kilka neuronów.

Ustalając liczbę kroków, będziemy podawać na wejście sieci losowo wybrane punkty ze zbioru P realizując przedstawiony wcześniej algorytm bez dokonywania normalizacji. Wyjściem neuronu jest odległość euklidesowa wektora wejściowego od wektora wag neuronu. Neuronem zwycięskim jest neuron o najmniejszej wartości wyjściowej. Efektem powinno być przemieszczanie się neuronów w kierunku skupisk punktów. Dodatkowo należy udostępnić możliwość włączenia lub wyłączenia mechanizmu sąsiedztwa.

Rozdział 33

Sieci samoorganizujące się na podstawie reguły Hebba

Rozdział 34

Kompresja obrazu – sieć samoorganizująca

34.1 Wprowadzenie

Bieżące zadanie wykorzystuje sieć samoorganizującą do kompresji obrazu (porównaj rozdział 28 oraz 32).

W celu skompresowania obrazu tworzymy sieć jednokierunkową jednowarstwową, która składa się z 64 wejść oraz n neuronów pierwszej warstwy. Liczba n wpływa na stopień kompresji pliku (im większa tym mniejszy współczynnik kompresji) i powinna być zmienną, którą możemy ustalać w momencie uruchamiania programu.

Jak zatem wykorzystać sieć samoorganizującą do omawianego zagadnienia i czym będzie kompresja? Pierwszym krokiem, jaki należy wykonać jest wygenerowanie zbioru wektorów wejściowych. Będzie to zbiór $3 \cdot p$ wektorów 64−elementowych. Sposób ich generowania został omówiony w rozdziale 28. Mając zbiór wejściowy należy postępować godnie z algorytmem:

1. Ustalenie struktury sieci, to znaczy ilości neuronów, gdyż ona determinuje zdolności sieci do wykrywania różnej ilości grup.
2. Wybór funkcji aktywacji jako funkcji liniowej.
3. Wczytanie zbioru wejść.
4. Wylosowanie wartości wag dla każdego neuronu.
5. Normalizacja wag i wczytanych sygnałów wejściowych według wzoru:

$$x_i = \frac{x_i}{\sqrt{\sum_{j=1}^{n} x_j^2}}$$

   gdzie n jest ilością wejść, $i = 1, \ldots, n$ indeksem współrzędnej.
6. Powtarzanie założoną ilość kroków:
   (a) Wybór wzorca p.
   (b) Obliczenie sygnału wyjściowego dla wzorca p.

(c) Znalezienie neuronu o największej wartości sygnału na wyjściu. Oznaczenie go jako *winner*.

(d) Zmiana wag *winner'a* tak aby były bliższe wzorcowi p według wzoru:

$$w_i = w_i + \eta[x_i - w_i]$$

gdzie $i = 1, \ldots, n$, n jest ilością wejść, η jest współczynnikiem uczenia. Współczynnik uczenia należy zmniejszać. Na początku może być większy, na przykład 0.2, a następnie powinien dążyć do zera.

7. Na zakończenie dla każdego wektora wejściowego zapamiętujemy numer neuronu, który zareagował najsilniej (*winner'a*).

Algorytm uczenia sieci dokona podziału wektorów wchodzących w skład zbioru uczącego na grupy. Grup tych będzie co najwyżej tyle z ilu neuronów składała się sieć. Na tej podstawie dokonamy procedury tworzenia pliku, będącego kompresją rozważanego obrazu. Plik będzie miał strukturę:

- n - liczba neuronów,
- n linii, z których każda będzie wektorem wag kolejnych neuronów,
- $3 \cdot p$ linii, z których każda będzie zawierała numer neuronu zwycięzcy dla kolejnych wektorów wejściowych.

Tak utworzony plik zawiera pełną informację o sieci. Jest przechowywana liczba neuronów. Znamy długość wektora wejściowego oraz macierz wag. Wiemy również, jak zostały sklasyfikowane wektory wejściowe. Aby dokonać dekompresji należy odtworzyć na podstawie *winnerów* wektory wejściowe. Nie będą to jednak dokładnie te same wektor, które podlegały kompresji. Dla każdej grupy wektorów odpowiadającej poszczególnym neuronom generujemy jeden wektor, jako reprezentant tej grupy. Będzie to wektor wag przypisanych neuronowi. Oznacza to, że wszystkie wektory wejściowe, dla których zwyciężył i-ty neuron zostaną zdekompresowane jako ten sam wektor – wektor wag i-tego neuronu. Wektor ten jest czymś w rodzaju średniej wektorów wejściowych generujących obszar wpływów rozważanego neuronu. Łatwo zauważyć, że zdekompresowany obraz będzie składał się z takiej ilości różnych klastrów ile mieliśmy neuronów winnerów.

Jeżeli wektory wejściowe należące do obszaru jednego neuronu nie różniły się między sobą, nie wystąpią straty podczas kompresji. W sytuacji przeciwnej – gdy różnicę będą występowały, wszystkie (mimo że różne między sobą) wektory zostaną odtworzone jako ten sam wektor. Wówczas wystąpią duże straty kompresji. Można im zapobiec, dokonując zmian wektorów wag o pewną wartość losową. Wartość ta będzie zależała od stopnia rozproszenia wektorów wejściowych należących do jednego obszaru. Użyjemy w tym celu miary rozrzutu.

W wyniku procesu uczenia wektory wejściowe zostały podzielone na grupy. Dla każdej z tych grup wyliczamy stopień rozproszenia wektorów wejściowych zgodnie ze wzorem:

$$s_k^i = \sqrt{\frac{\sum_{j=1}^{n} (x_j^k - \bar{x^k})^2}{n}},$$

gdzie

- i – numer grupy ($i \leq n$),
- k – współrzędna wektora wejściowego ($k = 1, ..., 64$),
- n – liczba wektorów w i-tej grupie,
- $\bar{x^k}$ – średnia arytmetyczna x_j^k, $j = 1, ..., n$.

Tak wyliczone miary rozrzutu będą wektorami 64-elementowymi, z których każdy przypisany jest do innego neuronu zwycięzcy. Informacja o rozrzucie musi być przechowywana w pliku kompresji. Teraz będzie on miał strukturę:

- n – liczba neuronów,
- n linii, z których każda będzie wektorem wag kolejnych neuronów,
- $3 \cdot p$ linii, z których każda będzie zawierała numer neuronu zwycięzcy dla kolejnych wektorów wejściowych,
- *liczba neuronw winnerw* linii, z których każda będzie zawierała wektor rozrzutu odpowiadających mu danych wejściowych.

Mając wyliczoną miarę rozrzutu zmieni się sposób dekompresji. Dekompresją każdego wektora wejściowego będzie wektor wag odpowiadającego mu neuronu zwycięzcy zmodyfikowany o miarę rozrzutu. Dokonamy tego zgodnie ze wzorem:

$$x_k^i = w_k^i + \rho s_k^i,$$

gdzie

- i – numer wzorca,
- k – numer współrzędnej,
- ρ – wartość losowa z przedziału $[-1, 1]$.

Tak dokonana modyfikacja pozwoli zastąpić jeden wektor reprezentujący dany neuron zwycięzcę kilkoma wektorami stanowiącymi jego modyfikację. Modyfikacja będzie tym większa im większy jest rozrzut wektorów wejściowych.

34.2 Zadanie

Zadanie będzie polegało na dokonaniu kompresji i dekompresji obrazu wykorzystując powyższą sieć i algorytm. Należy napisać dwa programy, z których jeden będzie wykorzystywał miarę rozrzutu, a drugi nie. Pozwoli to na zaobserwowanie wpływu modyfikacji wektora wag w zależności od rozproszenia danych wejściowych.

Rozdział 35

Metoda Levenberga - Marquardta dla zadania XOR

Algorytm Levenberga-Marquardta (LM) został opisany w podrozdziale 17.7, dlatego tutaj podamy jedynie wskazówki pomocne przy jego implementacji. Celem dokonania porównania z innymi zbadanymi przypadkami, problemem jaki przy jego pomocy rozwiążemy będzie problem XOR (jego opis pojawił się w rozdziale 26).

Przyjmowany model sieci przedstawia rysunek **??**. Wynika z niego, że sygnały wyjściowe z neuronów opisane są przez zależności

$$\begin{aligned} y_1^1 &= f(x_0^1 w_{10}^1 + x_1^1 w_{11}^1 + x_2^1 w_{12}^1), \\ y_2^1 &= f(x_0^1 w_{20}^1 + x_1^1 w_{21}^1 + x_2^1 w_{22}^1), \\ y_1^2 &= f(x_0^2 w_{10}^2 + x_1^2 w_{11}^2 + x_2^2 w_{12}^2), \end{aligned}$$

przy czym sygnał y_1^1 jest wyjściem z neuronu pierwszego warstwy pierwszej, ale jednocześnie jest pierwszym sygnałem wejściowym warstwy drugiej (x_1^2). Podobna zależność zachodzi dla sygnału y_2^1, który jest drugim sygnałem wejściowym warstwy drugiej (x_2^2). Funkcja f jest sigmoidalną funkcją aktywacji.

Zbiór uczący L składa się z następujących par

$$\begin{aligned} (p_1, t_1) &= ((0,0),(0)), \\ (p_2, t_2) &= ((0,1),(1)), \\ (p_3, t_3) &= ((1,0),(1)), \\ (p_4, t_4) &= ((1,1),(0)). \end{aligned}$$

Wektor błędu poszczególnych próbek uczących na każdym neuronie w warstwie wyjściowej (wzór 17.16) będzie miał postać[1]

$$e(w) = \begin{bmatrix} e_1^1 \\ e_1^2 \\ e_1^3 \\ e_1^4 \end{bmatrix} \tag{35.1}$$

[1] Mamy jeden neuron i cztery próbki uczące.

czyli

$$e(w) = \begin{bmatrix} t_1 - y_1^2(p_1) \\ t_2 - y_1^2(p_2) \\ t_3 - y_1^2(p_3) \\ t_4 - y_1^2(p_4) \end{bmatrix}$$

gdzie e_o^l, $o \in \{1\}$, $l \in \{1,2,3,4\}$ oznacza błąd l-tej próbki w o-tym neuronie wyjściowym. Zapis $y_1^2(p_l)$ rozumiemy jako wyjście z sieci otrzymane po prezentacji wzorca p_l.

Mając postać wektora $e(w)$ musimy znaleźć teraz postać macierzy $J(w)$. Ponieważ mamy łącznie dziewięć wag, zatem jej postać jest następująca

$$J(w) = \left[\begin{array}{cccccc|ccc} \frac{\partial e_1^1}{\partial w_{10}^1} & \frac{\partial e_1^1}{\partial w_{11}^1} & \frac{\partial e_1^1}{\partial w_{12}^1} & \frac{\partial e_1^1}{\partial w_{20}^1} & \frac{\partial e_1^1}{\partial w_{21}^1} & \frac{\partial e_1^1}{\partial w_{22}^1} & \frac{\partial e_1^1}{\partial w_{10}^2} & \frac{\partial e_1^1}{\partial w_{11}^2} & \frac{\partial e_1^1}{\partial w_{12}^2} \\ \frac{\partial e_1^2}{\partial w_{10}^1} & \frac{\partial e_1^2}{\partial w_{11}^1} & \frac{\partial e_1^2}{\partial w_{12}^1} & \frac{\partial e_1^2}{\partial w_{20}^1} & \frac{\partial e_1^2}{\partial w_{21}^1} & \frac{\partial e_1^2}{\partial w_{22}^1} & \frac{\partial e_1^2}{\partial w_{10}^2} & \frac{\partial e_1^2}{\partial w_{11}^2} & \frac{\partial e_1^2}{\partial w_{12}^2} \\ \frac{\partial e_1^3}{\partial w_{10}^1} & \frac{\partial e_1^3}{\partial w_{11}^1} & \frac{\partial e_1^3}{\partial w_{12}^1} & \frac{\partial e_1^3}{\partial w_{20}^1} & \frac{\partial e_1^3}{\partial w_{21}^1} & \frac{\partial e_1^3}{\partial w_{22}^1} & \frac{\partial e_1^3}{\partial w_{10}^2} & \frac{\partial e_1^3}{\partial w_{11}^2} & \frac{\partial e_1^3}{\partial w_{12}^2} \\ \frac{\partial e_1^4}{\partial w_{10}^1} & \frac{\partial e_1^4}{\partial w_{11}^1} & \frac{\partial e_1^4}{\partial w_{12}^1} & \frac{\partial e_1^4}{\partial w_{20}^1} & \frac{\partial e_1^4}{\partial w_{21}^1} & \frac{\partial e_1^4}{\partial w_{22}^1} & \frac{\partial e_1^4}{\partial w_{10}^2} & \frac{\partial e_1^4}{\partial w_{11}^2} & \frac{\partial e_1^4}{\partial w_{12}^2} \end{array}\right] \tag{35.2}$$

Pionową kreską rozdzielono pochodne cząstkowe liczone według różnych schematów. Zajmijmy się najpierw wagami warstwy wyjściowej (grupa po prawej stronie pionowej kreski) a więc wagami postaci $\frac{\partial e_1^l}{\partial w_{ij}^2}$, $l \in \{1,2,3,4\}$, $i \in \{1\}$, $j \in \{0,1,2\}$. Policzmy np. pochodną $\frac{\partial e_1^2}{\partial w_{11}^2}$

$$\frac{\partial e_1^2}{\partial w_{11}^2} = (t_2 - y_1^2(p_2))'_{w_{11}^2} = (t_2 - f(net_1^2(p_2))'_{w_{11}^2} = -f'(net_1^2(p_2))(net_1^2(p_2))'_{w_{11}^2} =$$

$$-f'(net_1^2(p_2))(x_0^2(p_2)w_{10}^2 + x_1^2(p_2)w_{11}^2 + x_2^2(p_2)w_{12}^2)'_{w_{11}^2} = -f'(net_1^2(p_2))x_1^2(p_2)$$

gdzie np. zapis $net_1^2(p_2)$ oznacza wartość net_1^2 policzoną dla sygnału p_2. Uogólniając otrzymany wynik na dowolną wagę warstwy wyjściowej w_{1i}^2, $i \in \{0,1,2\}$ dla l-tej próbki, otrzymujemy ogólny schemat dla tych wag

$$\frac{\partial e_1^l}{\partial w_{1i}^2} = -f'(net_1^2(p_l))x_i^2(p_l).$$

Zajmijmy się teraz wagami warstwy pierwszej (grupa po lewej stronie pionowej kreski) a więc wagami postaci $\frac{\partial e_1^l}{\partial w_{ij}^2}$, $l \in \{1,2,3,4\}$, $i \in \{1,2\}$, $j \in \{0,1,2\}$. Policzmy np. pochodną $\frac{\partial e_1^2}{\partial w_{11}^1}$

$$\frac{\partial e_1^2}{\partial w_{11}^1} = (t_2 - y_1^2(p_2))'_{w_{11}^1} = (t_2 - f(net_1^2(p_2))'_{w_{11}^1} = -f'(net_1^2(p_2))(net_1^2(p_2))'_{w_{11}^1} =$$

$$-f'(net_1^2(p_2))(x_0^2(p_2)w_{10}^2 + x_1^2(p_2)w_{11}^2 + x_2^2(p_2)w_{12}^2)'_{w_{11}^1} = -f'(net_1^2(p_2))(x_1^2(p_2)w_{11}^2)'_{w_{11}^1} =$$

$$-f'(net_1^2(p_2))(x_1^2(p_2))'_{w_{11}^1}w_{11}^2 = -f'(net_1^2(p_2))f(net_1^1(p_2))'_{w_{11}^1}w_{11}^2 =$$

$$-f'(net_1^2(p_2))f'(net_1^1(p_2))(net_1^1(p_2))'_{w_{11}^1}w_{11}^2 =$$

$$-f'(net_1^2(p_2))f'(net_1^1(p_2))(x_0^1(p_2)w_{10}^1 + x_1^1(p_2)w_{11}^1 + x_2^1(p_2)w_{12}^1)'_{w_{11}^1} w_{11}^2 =$$
$$-f'(net_1^2(p_2))f'(net_1^1(p_2))x_1^1(p_2)w_{11}^2.$$

Uogólniając otrzymany wynik na dowolną wagę warstwy wejściowej w_{ij}^1, $i \in \{1,2\}$, $j \in \{0,1,2\}$ dla l-tej próbki, otrzymujemy ogólny schemat dla tych wag

$$\frac{\partial e_1^l}{\partial w_{ij}^1} = -f'(net_1^2(p_l))f'(net_i^1(p_l))x_j^1(p_l)w_{1i}^2.$$

Pozostaje jeszcze do wyjaśnienia sprawa odpowiednich wymiarów macierzy i wektorów we wzorze (17.18) ze strony 174. Ze wzoru (35.1) widzimy, że wektor e jest wymiaru 4×1 natomiast ze wzoru (35.2) widzimy, że macierz J jest wymiaru 4×9. Jeśli więc przyjmiemy, że wektor wagowy w jest wektorem kolumnowym postaci

$$w = \begin{bmatrix} w_{10}^1 \\ w_{11}^1 \\ w_{12}^1 \\ w_{20}^1 \\ w_{21}^1 \\ w_{22}^1 \\ w_{10}^2 \\ w_{11}^2 \\ w_{12}^2 \end{bmatrix}$$

wówczas wymiary we wzorze (17.18) są zgodne

$$\underbrace{w(t+1)}_{9\times1} = \underbrace{w(t)}_{9\times1} - \underbrace{[\underbrace{\underbrace{J^T}_{9\times4}\underbrace{J}_{4\times9}}_{9\times9} + \underbrace{\eta I}_{9\times9}]^{-1}\underbrace{\underbrace{J^T}_{9\times4}\underbrace{e}_{4\times1}}_{9\times1}}_{9\times1}$$

Powyższe wyjaśnienia powinny rozwiać wszelkie wątpliwoście związane z praktyczną implementacją algorytmu LM dla postawionego problemu.

Rozdział 36

Parkowanie ciężarówki za pomocą sieci neuronowej

Zagadnienie parkowania ciężarówki polega na stworzeniu sterownika, który będzie w określony wcześniej sposób sterował ruchem ciężarówki. Aby zrealizować to zadanie należy precyzyjnie sformułować co w naszym przypadku będziemy rozumieć pod pojęciem „sterowania ciężarówką".

36.1 Sforułowanie problemu

Niech dany będzie obszar $\Omega = [-300, 300] \times [0, 400]$, nazywany obszarem parkingu. Obszar ten podzielimy prostą przechodzącą przez punkty $(0, 0)$ i $(0, 400)$ – będzie to oś parkingu. W obszarze Ω definujemy prostokąt wymiaru $a \times b$, gdzie $b > a > 0$[1]. Prostokąt ten to ciężarówka, którą mamy za zadanie sterować. Sterując ciężarówką możemy jeździć tylko do tyłu. Niech (x, y) oznacza współrzędne środka końca ciężarówki. Oznacza to że $(x, y) \in \Omega$. W celu jednoznacznego określenia położenia ciężarówki oznaczmy przez ϕ kąt jaki ciężarówka tworzy z osią parkingu. Przyjmijmy, że $\phi \in [-180, 180]$. Kąt 0 oznacza, że oś ciężarówki pokrywa się z osią parkingu, zaś tył ciężarówki zwrócony jest do górnej krawędzi Ω. Z wprowadzonych oznaczeń wynika, że położenie ciężarówki jest jednoznacznie wyznaczone przez wektor (x, y, ϕ).

Rysunek 36.1: Zagadnienie parkowania ciężarówki.

Ciężarówka posiada dodatkowo parametr $\theta \in [-45, 45]$, który oznacza kąt skręcenia kół ciężarówki. Wartosct $\theta = 0$ oznacza, że ciężarówka jedzie prosto. Dla danego położenia $(x(t), y(t), \phi(t))$ dobierana jest wartość kąta skręcenia kół $\theta(t)$. Na podstawie tych wartości wyliczane jest nowe położenie ciężarówki $(x(t+1), y(t+1), \phi(t+1))$:

$$x(t+1) = x(t) + \sin(\theta(t) + \phi(t)) - \sin(\theta(t))\cos(\phi(t)),$$

$$y(t+1) = y(t) - \cos(\theta(t) + \phi(t)) - \sin(\theta(t))\sin(\phi(t)),$$

$$\phi(t+1) = \phi(t) - \arcsin\left(\frac{2\sin(\theta(t))}{b}\right).$$

[1] Parametr b jest długością ciężarówki, a jej szerokością. Oczywiście oba parametry powinny być na tyle małe aby możliwe było poruszanie się w obszarze Ω (np. $a = 10$, $b = 20$).

Dobierając dowolnie kąt skręcenia kół ciężarówki trajektoria ruchu jest losowa. Chodzi nam o taki dobór parametru θ, aby ciężarówka zaparkowała w zadanym miejscu. Ciężarówka będzie zaparkowana, jeżeli jej położenie $(x, y, \theta) = (0, 0, 0)$. Chcemy stworzyć narzędzie, które w zależności od położenia ciężarówki będzie generowało odpowiedni kąt skręcenia kół. W efekcie sekwencja wykonanych przez ciężarówkę ruchów doprowadzi ją do położenia $(0, 0, 0)$, czyli zrealizujemy cel.

36.2 Kostrukcja sterownika

Narzędziem wykorzystywanym jako sterownik ciężarówki może być sieć neuronowa. Trzeba określić jej topologię oraz ją nauczyć. Nauczona sieć dla danego położenia będzie liczyła kąt skręcenia kół.

Siecią spełniającą wymagania zadania jest sieć jednokierunkowa, która ma 3 wejścia i dwie warstwy - pierwszą składającą się z 2 neuronów i drugą składającą się z 1 neuronu. Wejściem do sieci będzie wektor określający położenie ciężarówki, wyjściem z sieci będzie kąt skręcenia kół. Stworzona sieć jest więc aproksymatorem odwzorowania $f : ([-300, 300] \times [0, 400] \times [-180, 180]) \rightarrow [-45, 45]$.

Uwaga 36.1.

Przy konstrukcji i uczeniu sieci należy pamiętać o przeskalowaniu wartości w zbiorze uczącym. Jeżeli używamy sigmoidalnej unipolarnej funkcji aktywacji należy tak przekształcić dane, by wejścia do sieci były z zakresu $[-1, 1]$*, zaś wyjścia z zakresu* $(0, 1)$.

Aby nauczyć sieć trzeba skonstruować zbiór uczących. Należy go wygenerować doświadczalnie w następujący sposób.

1. Napisać program, w którym samodzielnie dobieramy kąt skręcenia kół ciężarówki realizuąc tym samym ruch.

2. Wylosować położenie ciężarówki i zmieniać kąt tak, by dojechać do celu.

3. Jeżeli uda nam się dojechać do celu należy zapamiętać parametry wszystkich wykonanych ruchów. Każdy ruch ma wektor czterech wartości, pierwsze trzy określają położenie ciężarówki, zaś ostatnia właściwy dla tego położenia kąt skręcenia kół. Przyjmując pierwsze trzy współrzędne jako wejście do sieci, zaś ostatnią jako żądane wyjście otrzymujemy kolejne elementy zbioru uczącego.

4. Jeżeli nie uda nam się dojechać do celu, tzn. w wyniku wykonywanych ruchów wyjedziemy poza obszar parkingu, wówczas losujemy położenie jeszcze raz. Nie pamiętamy paramatrów żadnego ruchu, nie jesteśmy bowiem w stanie stwirdzić, któr z nich były dobre a które złe.

5. Należy wygenerować jak najwięcej dróg dla różnych położeń początkowych.

Na podstawie wygenerowaniego zbioru uczącego uczymy sieć. Nauczona sieć jest gotowym sterownikiem dla ciężarówki.

36.3 Uproszczenie zagadnienia

Przedstawiony powyżej problem można uprościć. Można bowiem przyjąć, że zaparkowana ciężarówka to taka, której położenie $(x, y, \theta) = (0, y', 0)$, gdzie $y' \in (0, 400]$. Takie położenie oznacza, że oś ciężarówki pokrywa się z osią parkingu oraz ciężarówka jest zwrócona tyłem do górnej krawędzi Ω. Aby z położenia $(0, y', 0)$ przejechać do położenia $(0, 0, 0)$ wystarczy przyjąć $\theta = 0$ i wykonać odpowiednią liczbę kroków.

Dla tak zdefionowanego zadania upraszcza się konstrukcja sieci. Ma ona bowiem jedynie dwa wejścia określające położenie x oraz kąt θ. Parametr y jest wyliczany jedynie do sprawdzenia, czy ciężarówka znajduje się w obszarze parkingu.

Sterownik otrzymany przy uproszczonych założeniach nie uwzględnia wogóle współrzędnej y. Przez to będzie się źle zachowywał dla położeń zajdujących się w górnej i dolnej części Ω.

36.4 Zadanie

Napisać program, w którym po obszarze Ω porusza się ciężarówka. Losujemy dla niej położenie początkowe. Ruch ciężarówki odbywa się w sposób automatyczny sterowany siecią neuronową. Sieć jest nauczona na podstawie wcześniej wygenerowanych poprawnych sekwencji ruchów.

Rozdział 37

Sieć nauronowa radialna

37.1 Zadanie

Sieć neuronowa radialna może być wykorzystywana do aproksymowania funkcji. W celu zrealizowania tego zadania należy zdefiniować zbiór uczący a następnie w oparciu o ten zbiór skonstruować sieć i ją nauczyć.

Przyjmijmy, że aproksymowana przez nas funkcja f działa z R do R. Zdefiniujmy zbiór uczący jako

$$P = [-1, -0.9, -0.8, \ldots, 0.8, 0.9, 1.0]$$

$$\begin{aligned} T = [&-0.9602, -0.5770, -0.0729, 0.3771, 0.6405, 0.6600, 0.4609, \\ &0.1336, -0.2013, -0.4344, -0.5000, -0.3930, -0.1647, 0.0988, \\ &0.3072, 0.3960, 0.3449, 0.1816, -0.0312, -0.2189, -0.3201] \end{aligned}$$

gdzie P jest zbiorem wejść zaś T odpowiadających im wartości wyjściowych.

Uczenie sieci radialnej (patrz rozdział 19) przebiega następująco:

1. Wybór jako radialnej funkcji aktywacji funkcji postaci:

$$\phi(x) = \exp\left(-\frac{\|x - c\|^2}{r^2}\right).$$

2. Wyznaczenie na podstawie zbioru P ilości neuronów radialnych oraz ich parametrów c oraz r. Realizujemy to przy użyciu sieci samoorganizującej (patrz np. rozdział 18 lub 32). Tworzymy sieć, która ma za zadanie podzielić zbiór P na kilka grup. Ilość grup wyznaczonych przez sieć samoorganizującą stanowi liczbę neuronów radialnych n. Każdy neuron radialny odpowiada jednemu z neuronów identyfikujących grupę w sieci samoorganizującej. Jego wagi wyznaczają centrum c neuronu radialnego. Wartość r jest odległością c od najbliższego elementu danej grupy.

3. Określenie topologii sieci radialnej. W rozważanym zadaniu sieć posiada jedno wejście, n neuronów radialnych z parametrami c oraz r określonymi w poprzednim kroku oraz jedno wyjście.

4. Wylosowanie wag dla drugiej warstwy.

5. Nauczenie sieci zbioru $L = \{P, T\}$ dowolną metodą uczenia sieci jednokierunkowych[1].

[1]Zauważmy, że ponieważ wagi występują jedynie w warstwie drugiej, dlatego modyfikujemy jedynie wagi warstwy drugiej (wyjściowej).

Rozdział 38

Rozmyta sieć neuronowa

Ćwiczenie to jest kontynuacją przykładu z punktu 23.2.1 rozdziału 23.

Rozdział 39

Algorytmy genetyczne

39.1 Charakterystyka

Algorytm genetyczny to rodzaj algorytmu przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych. W istotny sposób różni się on od wcześniej poznanych metod przeszukiwania przestrzeni (rozdział 5) jako, że zgodnie z ideą twórcy Johna Henryego Hollanda inspiracje czerpią z biologii. Algorytmy genetyczne zrodziły się z prób naśladowania naturalnych procesów zachodzących w świecie organizmów żywych związanych z ewolucją i selekcją zachodzącą wśród osobników danej populacji. Umożliwiają rozwiązanie zagadnień optymalizacyjnych w oparciu o mechanizm doboru naturalnego i dziedziczenia. W związku z dość ścisłym związkiem z genetyką do algorytmów genetycznych przeniknęły takie terminy biologiczne jak gen, chromosom czy osobnik.

Mówiąc ogólnie, problem każdorazowo definiuje środowisko, w którym istnieje pewna **populacja** osobników. Każdy z osobników ma przypisany pewien zbiór informacji stanowiących jego **genotyp**, a będących podstawą do utworzenia **fenotypu**. Fenotyp to zbiór cech podlegających ocenie funkcji przystosowania modelującej środowisko. Innymi słowy — genotyp opisuje proponowane rozwiązanie problemu, a funkcja przystosowania ocenia, jak dobre jest to rozwiązanie.

Genotyp składa się z **chromosomów**, gdzie zakodowany jest fenotyp i ewentualnie pewne informacje pomocnicze dla algorytmu genetycznego. Chromosom składa się z **genów**.

Poniżej zebrano najważniejsze terminy wraz z krótkim objaśnieniem ich znaczenie.

Populacją nazywamy zbiór osobników o określonej liczebności.

Generacja to kolejna iteracja w algorytmie genetycznym, a o nowo utworzonej populacji osobników mówi się też nowe pokolenie lub pokolenie potomków.

Osobnikami populacji w algorytmach genetycznych są zakodowane w postaci chromosomów zbiory parametrów zadania, czyli rozwiązania, określone też jako punkty przestrzeni poszukiwań. Osobniki czasami nazywa się organizmami.

Chromosomy – inaczej łańcuchy lub ciągi kodowe – to uporządkowane ciągi genów.

Gen (nazywany też cechą, znakiem, detektorem) stanowi pojedynczy element genotypu, w szczególności chromosomu.

Genotyp , czyli struktura, to zespół chromosomów danego osobnika. Zatem osobnikami populacji mogą być genotypy albo pojedyncze chromosomy (jeśli genotyp składa się tylko z jednego chromosomu, a tak się często przyjmuje).

Fenotyp jest zestawem wartości odpowiadających danemu genotypowi, czyli zdekodowaną strukturą, a więc zbiorem parametrów zadania (rozwiązaniem, punkt przstrzeni poszukiwań).

Allel to wartość danego genu, określona jako wartość cechy lub wariant cechy.

Locus to pozycja – wskazuje miejsce położenia danego genu w łańcuchu, czyli chromosomie.

Funkcja przystosowania (nazywana też funkcją dopasowania lub funkcją oceny) stanowi miarę przystosowania (dopasowania) danego osobnika w populacji. Funkcja ta jest niezwykle istotna, gdyż pozwala ocenić stopień przystosowania poszczególnych osobników w populacji i na tej podstawie wybrać osobniki najlepiej przystosowane (czyli o największej wartości funkcji przystosowania), zgodnie z ewolucyjną zasadą przetrwania „najsilniejszych" (najlepiej przystosowanych). Funkcja przystosowania również przyjęła swą nazwę bezpośrednio z genetyki. Ma ona duży wpływ na działanie algorytmów genetycznych i musi być odpowiednio zdefiniowana.

Algorytmy genetyczne posiadają kilka bardzo istotnych cech, dzięki którym zyskały szerokie grono zarówno zwolenników jak i przeciwników.

1. Konkretna postać algorytmu ściśle zależy od rozwiązywanego zadania.

2. Stosowanie operatorów genetycznych, które dostosowane są do postaci zadania i poszukiwanego rozwiązania.

3. Przeszukują w sposób równoległy całą przestrzeń rozwiązań badając jednocześnie wiele jej punktów.

4. Stosują probabilistyczne reguły działania. Wiążą się z tym spore kłopoty natury teoretycznej. Trudno na przykład określić, czy uda się znaleźć rozwiązanie optymalne jakiegoś problemu.

5. Wykorzystują one funkcje celu tylko w swej pierwotnej postaci, bez odwoływania się do innych informacji jakie można z niej uzyskać, na przykład pochodnych. Dzięki temu staje się możliwe rozwiązywanie zagadnień źle uwarunkowanych numerycznie, czyli takich, gdzie klasyczne podejścia wypracowane przez metody numeryczne nie dają dobrych rezultatów lub wręcz nie można ich zastosować.

6. Konieczność odpowiedniej reprezentacji parametrów zadania w postaci zakodowanego łańcucha. Dobór sposobu kodowania i zestawu operatorów genetycznych decyduje o sukcesie lub porażce w stosowaniu tej metody.

7. W związku z przeszukiwaniem całej przestrzeni rozwiązań mamy dużo większe prawdopodobieństwo znalezienia rozwiązania optymalnego, ale też potrzeba na to znacznie więcej czasu.

8. Algorytm genetyczny nie daje nam jednego, konkretnego rozwiązania, ale całą serię rozwiązań optymalnych. Jeśli okaże się, że z jakiś powodów rozwiązanie reprezentowane przez najlepiej przystosowanego osobnika nie może być zastosowane, to możemy wziąć drugiego z najlepiej przystosowanych, albo jeszcze wcześniejszego. Zasada działania daje nam gwarancję (przynajmniej teoretycznie), że kolejne rozwiązania mimo, że nie optymalne będą bliskie temu optimum.

39.2 Ogólna postać algorytmu genetycznego

Algorytm genetyczny w swej klasycznej postaci przedstawia się następująco:

1. Start
2. Wybór chromosomów do początkowej populacji.
3. Ocena przystosowania każdego z chromosomów (osobników).
4. Czy spełniony jest warunek zatrzymania? TAK – idź do 8, NIE – kontynuuj.
5. Selekcja chromosomów i zastosowanie operatorów genetycznych.
6. Utworzenia nowej populacji.
7. Idź do 3.
8. Rozwiązanie problemu stanowi najlepiej przystosowany chromosom.
9. Stop.

W celu zilustrowania poszczególnych kroków, rozważmy następujące zadanie.
Znaleźć największą wartość funkcji

$$f(x) = -(x-2)^2 + 1$$

w przedziale $[1.0, 3.0)$. Dokładność otrzymanego wyniku powinna być nie mniejsza niż 0.05 (tzn. najmniejsza mozliwa do wyrażenia liczba powinna być mniejsza niż 0.05).

39.2.1 Ad. 0 – prace wstępne

Na początek należy zdecydować z ilu genów składa się chromosom (osobnik) i co każdy gen reprezentuje (jaką koduje informacje i w jaki sposób). W naszym przypadku najnaturalniej przyjąć, że chromosom reprezentuje binarny zapis liczby rzeczywistej z przedziału [0.0,1.0) a następnie skalować tę liczbę aby należała do wymaganego przedziału. W związku z tym każdy chromosom powinien mieć co najmniej 6 genów. Wówczas przyjmując, że wartość v ciągu bitów postaci

$$b_6 b_5 b_4 b_3 b_2 b_1$$

odczytujemy jako

$$v = \sum_{i=1}^{6} b_i \cdot 2^{-i},$$

otrzymujemy

Numer	Chromosom	Odkodowany chromosom		
i	c_i	z_i	x_i	$f(x_i)$
1	1,0,0,0,1,1	35/64	2.09375	0.991211
2	0,1,1,0,1,0	13/32	1.8125	0.964844
3	0,1,1,1,0,1	29/64	1.90625	0.991211
4	1,1,1,0,0,0	7/8	2.75	0.4375
5	1,1,0,1,1,0	27/32	2.6875	0.527344
6	1,0,1,1,1,1	47/64	2.46875	0.780273
7	0,1,0,1,0,0	5/16	1.625	0.859375
8	1,1,0,1,1,1	55/64	2.71875	0.483398
9	0,1,1,0,0,0	3/8	1.75	0.9375
10	1,0,1,0,1,1	43/64	2.34375	0.881836
11	0,0,1,0,0,1	9/64	1.28125	0.483398
12	1,1,1,1,1,1	63/64	2.96875	0.0615234
13	1,1,1,0,1,0	29/32	2.8125	0.339844
14	0,0,1,1,0,1	13/64	1.40625	0.647461
15	1,1,0,0,1,0	25/32	2.5625	0.683594
16	0,0,0,1,1,1	7/64	1.21875	0.389648
17	1,0,1,1,0,1	45/64	2.40625	0.834961
18	0,1,0,1,0,0	5/16	1.625	0.859375
19	1,1,1,1,0,0	15/16	2.875	0.234375
20	1,0,1,0,0,0	5/8	2.25	0.9375

Tabela 39.1:

```
000000 = 0.0
000001 = 0.015625
...
111111 = 0.984375
```

co przy stosowaniu skalowania według wzoru

$$x = 2z + 1 \quad dla \quad z \in [0, 1)$$

daje wartości

```
000000 = 1.0
000001 = 1.03125
...
111111 = 2.96875
```

i dokładność 0.03125.

39.2.2 Ad. 2 – wybór populacji początkowej

W wyniku losowej generacji chromosomów, otrzymujemy populację początkową np. taką jak przedstawiona w tabeli 39.1 (druga kolumna). Tym samym wykonany zostaje 2 krok algorytmu – wybrana została populację początkową.

39.2.3 Ad. 3 – ocena osobników

Teraz, zgodnie z algorytmem, dokonujemy oceny przystosowania każdego z chromosomów, obliczając po prostu wartość funkcji $f(x)$ (tabela 39.1, kolumna 5) dla $x = 2z + 1$, gdzie z to liczba dziesiętna otrzymana z chromosomu za pomocą wzoru

$$z = \sum_{n=1}^{6} \frac{1}{2^n}$$

gdzie n – ilość genów w chromosomie. Istotne jest od której strony liczymy geny – umawiamy się, że od lewej.

39.2.4 Ad. 4 – sprawdzenie warunku zatrzymania algorytmu

Jako warunek zatrzymania przyjmujemy w tym przykładzie ilość wykonanych iteracji; na razie wynosi ona 1 i jest mniejsza od założonej liczby 50.

39.2.5 Ad. 5 – selekcja chromosomów i zastosowanie operatorów genetycznych

Przystępujemy teraz do realizacji najważniejszej części algorytmu genetycznego – selekcji chromosomów i zastosowania na wybranych osobnikach operatorów genetycznych.

Podczas selekcji postępujemy zgodnie, jak się powszechnie uważa, ze wskazówkami otrzymanymi od Matki Natury. W żadnym razie nie odrzucamy któregokolwiek z chromosomów. Szansę na stanie się rodzicem ma każdy, dzięki temu unikamy pułapki podobnej do tej, w którą wpadają algorytmy zachłanne. Lepsza wartość funkcji przystosowania ujawnia się w większym prawdopodobieństwie wybrania danego chromosomu jako rodzica. Podobnie jak w naturze, osobniki najlepiej przystosowane mają największe szanse przetrwania. Jednak osobniki gorzej przystosowane nie są całkiem bez szans i z czasem może okazać się, że to właśnie one przetrwają.

Najczęściej (zapewne ze względu na prostotę) przy wyborze chromosomu stosuje się *metodę ruletki*. Każdemu chromosomowi przydziela się pewien przedział odcinka $[0, 1)$ lub, jak kto woli, wycinek koła ruletki. Przedział (wycinek) jest tym większy im lepiej dany chromosom jest przystosowany w stosunku do pozostałych. Granice przedziałów dla każdego chromosomu wyznaczamy w następujący sposób.

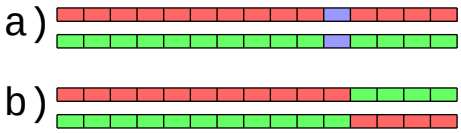
1. Sumujemy wartości funkcji przystosowania wszystkich chromosomów

$$F = \sum_{i=1}^{20} f(c_i), \tag{39.1}$$

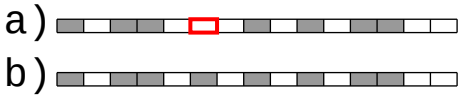
2. Następnie dla $i = 1, 2, \ldots, 20$ obliczamy

$$p_i = \frac{f(c_i)}{F}, \tag{39.2}$$

$$q_i = \sum_{j=1}^{i} p_j. \tag{39.3}$$



Rysunek 39.1: Krzyżowanie. Chromosomy wyznaczone do krzyżowania przedstawione są na rysunku a). Niebieski gen oznacza miejsce, za którym nastąpi cięcie. b) – wygląd chromosomów po krzyżowaniu.



Rysunek 39.2: Mutacja. Chromosom wyznaczony do zmutowania przedstawiony jest na rysunku a). Czerwona ramka oznacza gen, który zostanie poddany mutacji. b) – wygląd chromosomu po mutacji.

3. Dla pierwszego chromosomu szukanym przedziałem jest $[0, q_1)$, dla pozostałych zaś $[q_{i-1}, q_i)$.

Mając określone przedziały (wycinki koła) losujemy jedną liczbę z przedziału $[0, 1)$ (puszczamy kulkę na kole ruletki). Chromosom, któremu odpowiada przedział w jakim znalazła się ta liczba, wybieramy do dalszego postępowania.

Stosując tę metodę losujemy 20 par i 10 pojedyńczych chromosomów. Pary posłużą nam do reprodukcji, natomiast pojedyńcze osbniki do mutacji.

Reprodukcja (krzyżowanie)

Podczas reprodukcji następuje wymiana pewnych odcinków łańcuchów genów pomiędzy dwoma chromosomami. Najprostszy przypadek został zilustrowany na rysunku 39.1. Losowo wybieramy gen, który stanie się miejscem cięcia (rysunek 39.1 a), niebieski gen; cięcie nastąpi za nim) a nastepnie dokonujemy zamiany odpowiednich odcinków (rysunek 39.1 b)).

Mutacja

Mutacja jest mniej skomplikowaną operacją (rysunek 39.2). Ponownie losujemy gen (rysunek 39.2 a); gen wylosowany otoczony jest czerwoną ramką) lecz tym razem zmieniamy tylko jego wartość (rysunek 39.2 b)).

W tym miejscu należy poczynić pewne uwagi.

1. Wybór sposobów działania operatorów krzyżowania i mutacji ściśle zależy od rozwiązywanego zadania a ściślej rzecz biorąc, sposobu reprezentacji danych przez geny. Należy szczególnie uważać, czy w wyniku ich zastosowania otrzymuje się osobnika z danej populacji. W naszym przypadku zadanie jest na tyle proste, że trudności te nie występują, ale są przypadki gdy bez dodatkowych zabiegów

operatory genetyczne mogą generować złe osobniki (tzn. osobniki nie będące poprawnymi reprezentacjami poszukiwanego rozwiązania; proszę zajrzeć do ćwiczenia **??**).

2. Nowo powstała populacja składa się ze starych i nowych osobników (dlatego też chwilowo rozmiar populacji zwiększa się).

3. Fakt, iż do krzyżowania wybrano 20 par a do mutacji 10 osobników, nie oznacza, że proces ten faktycznie u wszystkich zajdzie. Dla każdej pary (osobnika) losuje się liczbę. Jeśli jest ona mniejsza od przyjętego prawdopodobieństwa krzyżowania CP lub mutacji MP, wówczas dopiero mogą one zajść. Zwykle przyjmuje się, że prawdopodobieństwo mutacji jest o rząd lub dwa mniejsze niż prawdopodobieństwo krzyżowania (np. krzyżowanie 0.2, mutacja 0.2).

4. Wybór par może przebiegać według pewnych dodatkowych kryteriów. W opisie dla prostoty nie wprowadzamy żadnych ograniczeń, dlatego też każdy osobnik może krzyżować się więcej niż jeden raz.

5. Liczby 20 i 10 są jedynie przykładem i zwykle dobiera się je zależnie od zadania i wielkości populacji.

Przedziały odpowiadające wygenerownaym przez nas chromosomom zawiera tabela 39.2. Kolumna druga to granice przedziału, trzecia – jego długość. Czwarta kolumna zawiera informacje o numerze pary, do której wylosowano dany chromosom w celu przeprowadzenia krzyżowania, piąta zaś, jako który został wytypowany do mutacji.

Tabele 39.3, 39.4 i 39.5 zawierają zestawienie, przeprowadzonych w wyniku realizacji algorytmu, krzyżowań i mutacji. Ze względu na przyjęte prawdopodobieństwa krzyżowania CP=0.3 i mutacji MP=0.01, operacje te nie zawsze mają miejsce.

39.2.6 Ad. 6 – utworzenie nowej populacji

Prawa przyrody są sprawiedliwe, ale i okrutne zatem należy „pozbyć się" kilku osobników – dokładnie tylu o ile zwiększyła się chwilowa liczebność populacji. Osobniki mające zginąć wybieramy ponownie zgodnie z zasadą ruletki tylko trochę zmodyfikowaną – do wzorów 39.1–39.3 zamiast $f(c_i)$ wstawiamy $1/f(c_i)$.

39.2.7 Ad. 7 – zakończenie jednego kroku algorytmu

W ten oto sposób zakończyliśmy jeden przebieg algorytmu genetycznego. Teraz następuje powrót do kroku 3 i ponowne wykonanie opisanych czynności.

39.2.8 Ad. 8 – zakończenie algorytmu

Ostatecznie, po //uzup// krokach otrzymujemy populację z tabeli //uzup//.

Ćwiczenie 39.1.

Opisany powyżej algorytm zrealizować w postaci programu. Proszę dowolnie wybrać wszystkie istotne parametry, a także optymalizowaną funkcję.

Numer chromosomu	Podprzedział	Długość podprzedziału	Numer pary do krzyżowania	Który do mutacji
1	[0.0,0.0743808)	0.0743808	1,2,7	10
2	[0.0743808,0.146783)	0.0724022	3,8,18	
3	[0.146783,0.221164)	0.0743808	6,11,16	
4	[0.221164,0.253994)	0.0328301	15,17,18	7
5	[0.253994,0.293566)	0.0395721		3
6	[0.293566,0.352118)	0.0585519	13,17	
7	[0.352118,0.416606)	0.0644878	4,16	9
8	[0.416606,0.45288)	0.0362743	8	
9	[0.45288,0.52323)	0.0703503	20	5
10	[0.52323,0.589403)	0.0661732	10,19	1,4
11	[0.589403,0.625678)	0.0362743		
12	[0.625678,0.630295)	0.00461673	10	2
13	[0.630295,0.655797)	0.025502	5,12	
14	[0.655797,0.704382)	0.0485857	2	
15	[0.704382,0.755679)	0.0512971	4,11,13	8
16	[0.755679,0.784919)	0.0292393	9,14	
17	[0.784919,0.847574)	0.0626557	1,3,9,12,19	6
18	[0.847574,0.912062)	0.0644878	15,20	
19	[0.912062,0.92965)	0.0175876		
20	[0.92965,1.0]	0.0703503	5,6,7,14	

Tabela 39.2:

Numer pary	Chromosom pierwszy	Chromosom drugi
1	1,0,0,0,1,1	1,0,1,1,0,1
2	1,0,0,0,1,1	0,0,1,1,0,1
3	0,1,1,0,1,0	1,0,1,1,0,1
4	0,1,0,1,0,0	1,1,0,0,1,0
5	1,1,1,0,1,0	1,0,1,0,0,0
6	0,1,1,1,0,1	1,0,1,0,0,0
7	1,0,0,0,1,1	1,0,1,0,0,0
8	0,1,1,0,1,0	1,1,0,1,1,1
9	0,0,0,1,1,1	1,0,1,1,0,1
10	1,0,1,0,1,1	1,1,1,1,1,1
11	0,1,1,1,0,1	1,1,0,0,1,0
12	1,1,1,0,1,0	1,0,1,1,0,1
13	1,0,1,1,1,1	1,1,0,0,1,0
14	0,0,0,1,1,1	1,0,1,0,0,0
15	1,1,1,0,0,0	0,1,0,1,0,0
16	0,1,1,1,0,1	0,1,0,1,0,0
17	1,1,1,0,0,0	1,0,1,1,1,1
18	0,1,1,0,1,0	1,1,1,0,0,0
19	1,0,1,0,1,1	1,0,1,1,0,1
20	0,1,1,0,0,0	0,1,0,1,0,0

Tabela 39.3: Pary wytypowane do krzyżowania.

Numer pary	Chromosom pierwszy	Chromosom drugi	Miejsce Krzyżowania
1	1,0,0,0,1,1 1,0,0,1,0,1	1,0,1,1,0,1 1,0,0,1,0,1	3
5	1,1,1,0,1,0 1,1,1,0,0,0	1,0,1,0,0,0 1,1,1,0,0,0	3
6	0,1,1,1,0,1 0,1,1,0,0,0	1,0,1,0,0,0 0,1,1,0,0,0	2
8	0,1,1,0,1,0 0,1,1,0,1,1	1,1,0,1,1,1 0,1,1,0,1,1	4
10	1,0,1,0,1,1 1,1,1,1,1,1	1,1,1,1,1,1 1,1,1,1,1,1	5
18	0,1,1,0,1,0 0,1,1,0,0,0	1,1,1,0,0,0 0,1,1,0,0,0	1

Tabela 39.4: Pary poddane krzyżowaniu.

Numer chromosomu	Chromosom	Numer mutowanego genu	Chromosom po mutacji
10	1,0,1,0,1,1		
12	1,1,1,1,1,1		
5	1,1,0,1,1,0		
10	1,0,1,0,1,1		
9	0,1,1,0,0,0		
17	1,0,1,1,0,1		
4	1,1,1,0,0,0		
15	1,1,0,0,1,0		
7	0,1,0,1,0,0	4	0,1,0,0,0,0
1	1,0,0,0,1,1		

Tabela 39.5: Chromosomy wybrane do mutacji.

Ćwiczenie 39.2.

Proszę napisać program wykorzystujący algorytm genetyczny znajdujący najkrótszą drogę między zadanymi węzłami w grafie (zakładamy, że połączenie takie istnieje).

Rozdział 40

Magiczne kwadraty

40.1 Przedstawienie zadania

Zagadnienie magicznych kwadratów polega na wygenerowaniu kwadratu $n \times n$, w którym elementami są liczby $1, 2, ..., n^2$ ułożone tak, by w każdej kolumnie, wierszu i przekątnej suma wartości liczb była taka sama i wynosiła $\frac{1+n^2}{2} \cdot n$. Przykładowy kwadrat dla $n = 3$ przedstawia rysunek 40.1.

Jest to zadanie łatwe jeżeli n jest stosunkowo małe. W przypadku dużych n zagadnienie jest trudne do rozwiązania. Jeżeli chcielibyśmy sprawdzić wszystkie możliwości rozmieszczenie liczb w kwadracie to otrzymamy $n^2!$ różnych możliwych rozmieszczeń.

Ze względu na wielkość przestrzeni poszukiwań dosyć naturalnym wydaje się próba rozwiązania tego zadania za pomocą algorytmu genetycznego.

40.2 Adaptacja zadania

Fenotypem w tym zadaniu jest kwadrat o boku n z rozmieszczonymi liczbami naturalnymi od 1 do n^2. Pojedynczy genotyp (chromosom) wchodzący w skład populacji jest ciągiem n^2-elementowym o genach należących do zbioru $\{1, 2, ..., n^2\}$ (przy czym wartość genu nie powtarza się - pojedynczy chromosom jest permutacją ciągu $(1\ 2\ ...\ n^2)$.

Tworzenie potomstwa odbywa się przy wykorzystaniu operatorów genetycznych mutacji i krzyżowania oraz dodatkowej operacji zamiany segmentów genów.

40.2.1 Krzyżowanie

Krzyżowanie polega na wymianie segmentów genów między dwoma chromosomami. Załóżmy, że mamy dwóch rodziców: $R1$ i $R2$. Aby skrzyżować dwa chromosomy dokonujemy podzielenia każdego z nich na trzy segmenty w taki sposób aby segment środkowy

8	1	6
3	5	7
4	9	2

Rysunek 40.1: Magiczny kwadrat dla $n = 3$.

Para rodziców:	[1 3 5 | 8 6 4 2 | 7 9]	[4 | 5 6 8 1 | 2 3 9 7]
I iteracja:	[1 3 8 | 5 6 4 2 | 7 9]	[4 | 8 6 5 1 | 2 3 9 7]
II iteracja:	[1 3 8 | 5 6 4 2 | 7 9]	[4 | 8 6 5 1 | 2 3 9 7]
III iteracja:	[1 3 4 | 5 6 8 2 | 7 9]	[5 | 8 6 4 1 | 2 3 9 7]
Para potomków:	[2 3 4 | 5 6 8 1 | 7 9]	[5 | 8 6 4 2 | 1 3 9 7]

Tabela 40.1: Przykład działania operatora krzyżowania dla $n = 3$.

był jednakowej długości w obu chromosomach. Długość segmentu może być jedną z wartości $\{1, 2, ..., \frac{n^2}{2}\}$. Tworzymy dwa nowe chromosomy takie same jak $R1$ i $R2$.

Nowe chromosomy modyfikujemy w taki sposób, aby wymienić między nimi środkowe segmenty. Ponieważ geny nie są jedynie liczbami 0 i 1 więc wymiana nie może być bezpośrednim zastąpieniem jednego segmentu drugim, gdyż otrzymany chromosomy mógłby zawierać powtórzone geny. Aby tego uniknąć stosujemy zamianę elementów przeprowadzoną w sposób następujący. Modyfikujemy pierwszego potomka. Sprawdzamy jaki gen znajduje się na pierwszym miejscu w środkowym segmencie drugiego rodzica. Zamieniamy w pierwszym potomku gen o tej samej wartości z genem znajdującym się na pierwszym miejscu w segmencie środkowym. Tę samą procedurę powtarzamy dla wszystkich genów segmentu środkowego. W analogiczny sposób modyfikujemy drugiego potomka. Przykład działania operatora krzyżowania jest przedstawiony w tabeli 40.1.

40.2.2 Mutacja

Mutacja chromosomu to zamiana miejscami dwóch losowo wybranych genów. Taki sposób mutacji wynika z wartości genów chromosomu. Ponieważ genami są różne liczby naturalne, a nie wartości binarne, to nie ma możliwości zmiany wartości genu na przeciwną. Ponieważ chromosom jest ciągiem n^2 liczb naturalnych, wiec mutacja jest losową transpozycją.

Dodatkowo stosujemy operatory zamiany segmentów genów długości n odpowiadających kolumnom, wierszom lub przekątnym. Jest to na przykład zamiana miejscami dwóch wierszy, dwóch kolumn, wiersza i kolumny lub przekątnej z wierszem czy kolumną. Omówione operatory genetyczne są stosowane w udziale procentowym:

- transpozycja: 50%,
- krzyżowanie: 30%,
- zamiany: 20%.

40.2.3 Funkcja oceny

Po wygenerowaniu potomstwa dokonujemy oceny potomków. Wykorzystując funkcję oceny tworzymy nową populację główną. Do populacji tej dołączane są te chromosomy, dla których wartość funkcji oceny jest największa. Funkcja oceny przyjmuje wartości odpowiadające ilości kolumn, wierszy i przekątnych, dla których suma wartości genów wynosi $\frac{1+n^2}{2} \cdot n$. Maksymalną wartością funkcji oceny jest wartość $2n+2$ odpowiadająca wygenerowaniu chromosomu spełniającego warunki zadania, tzn. chromosomu, któremu odpowiada kwadrat z jednakową sumą elementów w każdym wierszu, kolumnie i na przekątnych.

Jeżeli nowowygenerowane chromosomy powtarzają się z chromosomami już istniejącymi to je usuwamy.

Po utworzeniu nowej populacji dokonujemy sortowania chromosomów znajdujących się w populacji w taki sposób, aby chromosomy były ułożone od największej do najmniejszej wartości funkcji oceny. Do sortowania chromosomów użyć możemy np. działającego w czasie liniowym algorytmu sortowania przez zliczanie.

40.2.4 Warunek końca

Program kończy działanie kiedy generuje chromosom, dla którego funkcja oceny wynosi $2n + 2$. Jeżeli funkcja oceny nie osiągnie takiej wartości, wówczas program kończy działanie po wykonaniu maksymalnej ilości iteracji. Maksymalna ilość iteracji jest ustalona w opcjach programu. Można ponadto przyjąć dodatkowo następujące działanie. W momencie wykonania maksymalnej ilości iteracji następuje sprawdzenie wartości funkcji oceny w ostatnich iteracjach (w połowie maksymalnej liczby iteracji). Jeżeli wzrastała wartość funkcji oceny wraz ze wzrostem ilości iteracji, to czas działania programu zostaje wydłużony o połowę maksymalnej ilości iteracji.

40.3 Zadanie

Zaimplementować algorytm genetyczny do rozwiązania zaprezentowanego problemu.

Rozdział 41

Zadanie optymalizacyjne

41.1 Przedstawienie zadania

Mamy do rozwiązania „typowe" zadanie optymalizacyjne. Na pewnym obszarze $\Omega \in \mathbf{R}^2$ jest określona funkcja $f : \Omega \to \mathbf{R}$. Należy tak zmodyfikować obszar Ω, aby zmaksymalizować wartość funkcji na nim określonej. Jedynym dopuszczalnym sposobem modyfikowania obszaru jest wycinanie w nim dziur. Wycięcie dziury rozumiemy tutaj jako określenie wartości funkcji f na pewnym podobszarze obszaru Ω jako równych zero. Ponadto zakładamy, że dziur tych nie można wyciąć więcej jak 3. Jest to więc zadanie typowe dla zastosowań algorytmów genetycznych, gdyż mamy tutaj do czynienia z dużą przestrzenią poszukiwań i potencjalnie z wieloma rozwiązaniami optymalnymi lub ε-optymalnymi, tj. różniącymi się od optymalnego o nie więcej jak pewne ustalone ε[1].

41.2 Sprecyzowanie warunków zadania

Na potrzeby praktycznej realizacji zadania przyjmijmy, że obszar Ω jest określony jak poniżej

$$\Omega = [0, 8] \times [0, 8].$$

Na obszarze tym określmy funkcję f jak poniżej

$$f(x) = \begin{cases} -20 & \text{gdy} \quad x \in [5, 5.5] \times [2, 2.5] \cup [6, 6.5] \times [4.5, 5] \cup [2, 2.5] \times [4, 4.5] \\ -10 & \text{gdy} \quad x \in [4, 4.5] \times [5.5, 6] \\ +10 & \text{gdy} \quad x \in [1.5, 2] \times [2, 2.5] \cup [1.5, 2] \times [6.5, 7] \cup [7, 7.5] \times [1.5, 2] \\ 0 & \text{w pozostaych przypadkach.} \end{cases}$$

Rysunek **??** przedstawia rozkład obszarów na które funkcja f dzieli obszar Ω. Jak widać w tym konkretnym zadaniu dla uproszczenia można na obszar nałożyć siatkę, podobnie jak na wspomnianym rysunku. Dzięki temu zawężimy dziedzinę poszukiwań do zbioru liczb $\{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5\}$, które będą określały odpowiednie „kwadraty" z tak podzielonego obszaru. I tak np.

- kwadratowi o współrzędnych $(2, 5.5)$ odpowiada podobszar $[2, 2.5] \times [5.5, 6]$;
- kwadratowi o współrzędnych $(0, 0)$ (a więc w lewym górnym rogu) odpowiada podobszar $[0, 0.5] \times [0, 0.5]$;

[1] Oczywiście nie możemy zagwarantować, że algorytm genetyczny znajdzie rozwiązanie ε-optymalne, ale z praktyki wiemy, że są duże szanse, iż tak się stanie.

- kwadratowi o współrzędnych $(0, 7.5)$ (a więc w prawym górnym rogu) odpowiada podobszar $[0, 0.5] \times [7.5, 8]$;
- kwadratowi o współrzędnych $(7.5, 0)$ (a więc w lewym dolnym rogu) odpowiada podobszar $[7.5, 8] \times [0, 0.5]$;
- kwadratowi o współrzędnych $(7.5, 7.5)$ (a więc w prawym dolnym rogu) odpowiada podobszar $[7.5, 8] \times [7.5, 8]$.

Wartością funkcji oceny będzie całka z funkcji f na obrszarze Ω. W tym zadaniu jej obliczenie jest bardzo proste: jest to suma odpowiedniej ilości „słupków" o wartościach -20, -10 oraz 10 z pominięciem słupków znajdujących się na obszarze wyciętym. Tak więc najmniejsza możliwa wartość to -70 (gdy „wytniemy wszystkie obszary o wartości 10) a największa to 20 (gdy „wytniemy wszystkie obszary o wartości -20).

Mając tak uproszczone zadanie, możemy określić jak wygląda chromosom. Przyjmijmy, że jeden chromosom składa się z 6 genów. Sześciu, bo chcemy wyciąć 3 otwory a jeden otwór ma dwie współrzędne. Gen natomiast jest liczbą. Jako liczby przyjmijmy liczby rzeczywiste z przedziału $[0, 7.5]$ reprezentowane z dokładnością 0.5. Oznacza to, ze do reprezentacji liczb można użyć reprezentacji stałoprzecinkowej, przeznaczając 3 na zapisanie bitów reprezentujących część całkowitą i 1 na zapisanie bitów reprezentujących część ułamkową. Zatem format ten wygląda jak poniżej

$$c_2 c_1 c_0 u_1,$$

gdzie u_1 – część ułamkowa, $c_1, \ldots, c_3$ – część całkowita. Wartość w takiego ciągu bitów wynosi:

$$w = \sum_{i=0}^{2} c_i 2^i + \sum_{i=1}^{1} u_i 0.5^i.$$

Oto wszystkie możliwe do reprezentowania w tak przyjętym formacie liczby rzeczywiste:

```
0000 = 0.0    1000 = 4.0
0001 = 0.5    1001 = 4.5
0010 = 1.0    1010 = 5.0
0011 = 1.5    1011 = 5.5
0100 = 2.0    1100 = 6.0
0101 = 2.5    1101 = 6.5
0110 = 3.0    1110 = 7.0
0111 = 3.5    1111 = 7.5
```

Pozostają jeszcze do określenia operatory genetyczne jakie będziemy stosować i jak one będą działać.

41.2.1 Mutacja

Operacji mutacji podlegają pojedyńcze bity genu (liczby rzeczywistej). Aby przeprowadzić operację mutacji należy określić dwa prawdopodobieństwa: p_1 i p_2 oraz dwie liczby n_1 i n_2. Wartość p_1 określa z jakim prawdopodobieństwem gen (liczba) wybierany jest do mutacji. Wartość p_2 określa z jakim prawdopodobieństwem mutowany jest wybrany bit genu podlegającego mutacji. Wartość n_1 określa ile razy wybieramy bit do muatcji z części całkowitej, natomiast n_2 określa ile razy wybieramy bit do muatcji z części ułamkowej (w obu przypadkach ten sam bit może być wybrany kilka razy).

41.2.2 Krzyżowanie

Wybieramy bardzo prostą metodę krzyżowania – prześledźmy ją na poniższym przykładzie. Dane mamy dwa chromosomy c_1 i c_2:

$$c_1 = g_{1,1}g_{1,2}g_{1,3}g_{1,4}g_{1,5}g_{1,6}$$

oraz

$$c_2 = g_{2,1}g_{2,2}g_{2,3}g_{2,4}g_{2,5}g_{2,6}$$

gdzie $g_{1,i}$ oraz $g_{2,i}$ dla $i = 1, \ldots, 6$ są genami. Wybieramy losowo pewien punkt krzyżowania – powiedzmy, że wylosowaliśmy 4. Wykonujemy teraz krzyżowanie zgodnie z zadanym prawdopodobieństwem, otrzymując w jego wyniku dwa nowe chromosomy C_1 i C_2:

$$C_1 = g_{2,1}g_{2,2}g_{2,3}g_{1,4}g_{1,5}g_{1,6}$$

oraz

$$C_2 = g_{1,1}g_{1,2}g_{1,3}g_{2,4}g_{2,5}g_{2,6}.$$

41.2.3 Selekcja

Również do selekcji osobników wchodzących w skład populacji użyjemy najprostszej metody – metody ruletki. Metoda ta każdemu chromosomowi przypisuje pewien fragment koła – fragment zależny od wartości funkcji oceny dla danego chromosomu. Im lepiej chromosom odpowiada warunkom zadania, tym większy fragment odpowiada jemu na kole. Następnie losuje się pewną liczbę z zadanego przedziału, powiedzmy, że od 0 dp 100, i przyjmując, że cały obwód koła ma 100, patrzymy do którego obszaru wpadnie wylosowana liczba. Obszar, do którego wpadła wyznacza nam chromosom. Taka metoda wyboru jest sprawiedliwa o tyle, że największe szanse wyboru mają osobniki najlepiej przystosowane, ale nie odrzuca się tych gorzej przystosowanych.

Przy tak przyjętej metodzie wyboru należy pamiętać, że daje się ona stosować tylko wtedy, gdy wartości funkcji oceny są dodatnie. Jeśli takiej gwarancji nie mamy, a tak jest właśnie w omawianym zadaniu, wówczas musimy dokonywać tymczasowego przesunięcia otrzymanych wartości funkcji oceny. Tak więc algorytm utworzenia koła ruletki dla zadanej populacji o rozmiarze n wygląda następująco.

1. Przpisanie każdemu chromosomowi, wartości funkcji oceny v_i, $i = 1, \ldots, n$.
2. Wybranie najmniejszej z wartości v_i i oznaczenie jej jako v_{min}.
3. Przypisanie każdemu chromosomowi tymczasowej wartości funkcji oceny, wg. wzoru

   $$v_i^{tmp} = v_i - v_{min} + 0.1$$

   Jak widać z powyższego wzoru, najmniejsza tymczasowa wartość funckji oceny wynosi 0.1.
4. Wyznaczenie przedziałów p_i, $i = 1, \ldots, n$, odpowiadających chromosomom, wg. poniższych wzorów

   $$p_1 = \left[0.0, \frac{v_1^{tmp}}{sum}\right), \tag{41.1}$$

   $$p_i = \left[p_i^g, \frac{v_i^{tmp}}{sum}\right), \qquad \text{dla} \quad i = 2, \ldots, n, \tag{41.2}$$

gdzie

- sum określone jest wzorem $sum = \sum_{i=1}^{n} v_i^{tmp}$,
- p_i^g określa górny kraniec przedziału p_i.

41.3 Szkic algorytmu

Poniżej przedstawiamy szkic algorytmu genetycznego z uwzględnieniem jego najważniejszych etapów.

1. Ustalenie wartości zmiennej t na 1 (t jest zmienną czasową określającą kolejne populacje).
2. Utworzenie populacji $pop(t)$ liczącej n osobników.
3. Ocena chromosomów (wyliczenie wartości v_i, $i = 1, \ldots, n$).
4. Wyliczenie tymczasowych wartości funkcji oceny v_i^{tmp}, $i = 1, \ldots, n$.
5. Wyliczenie przedziałów p_i, $i = 1, \ldots, n$.
6. Krzyżowanie k par chromosomów. Po tym kroku, populacja liczy $n + 2kosobnikw$. Zakładamy, że chromosomy potomne wchodzą w skład populacji a nie zastępują chromosomów rodziielskich, czyli populacja składa się z rodziców (których jest n) oraz ich potomków (których jest $2k$).
7. Ocena chromosomów (wyliczenie wartości v_i, $i = 1, \ldots, n + 2k$)[2].
8. Wyliczenie tymczasowych wartości funkcji oceny v_i^{tmp}, $i = 1, \ldots, n + 2k$.
9. Wyliczenie przedziałów p_i, $i = 1, \ldots, n + 2k$.
10. Mutacja przeprowadzona z pewnym prawdopodobieństwem na wszystkich $n + 2k$ osobnikach.
11. Ocena chromosomów (wyliczenie wartości v_i, $i = 1, \ldots, n + 2k$)[3].
12. Wyliczenie tymczasowych wartości funkcji oceny v_i^{tmp}, $i = 1, \ldots, n + 2k$.
13. Wyliczenie przedziałów p_i, $i = 1, \ldots, n + 2k$.
14. Utworzenie populacji $pop(t + 1)$ składającej się z n losowo wybranych osobników populacji $pop(t)$.
15. Jeśli t jest mniejsze od zadanej ilości kroków, to powrót do punktu 3.

41.4 Zadanie

Zaimplementować algorytm genetyczny do rozwiązania zaprezentowanego problemu.

[2] Prawdę mówiąc, w tym przypadku wystarczy ocenić chromosomy od $n + 1$ do $n + 2k$

[3] W tym przypadku trzeba ocenić chromosomy od 1 do $n + 2k$

Rozdział 42

Metoda Monte Carlo

42.1 Wprowadzenie

Losowe działanie, choć z pozoru wydaje się pozbawione sensu, zdaje się być jedynym słusznym podejściem w problemach o dużej złożoności obliczeniowej. Zauważmy, że rzadko kiedy zależy nam na uzyskaniu wyniku dokładnego. I tak zamiast $\frac{1}{3}$ wystarczyć nam może 0.33. Wynika to w znacznej mierze z niedokładności urządzeń pomiarowych przez obliczeniowe na wykonawczych kończąc. Losowe próbkowanie przestrzeni zdarzeń ma tą zaletę, że zwiększa znacznie szanse na znalezienie rozwiązania suboptymalnego, tj. takiego rozwiązania, które nie jest faktycznym rozwiązaniem optymalnym, ale z naszego punktu widzenia za takie może uchodzić. Poznaliśmy zalety takiego działania przy okazji omawiania algorytmów genetycznych (rozdział 39) a teraz poznamy kolejną metodę z serii metod losowych[1].

Metoda Monte Carlo jest algorytmem obliczeniowym opartym na wielokrotnym losowym próbkowaniu przestrzeni w celu obliczenia ostatecznego wyniku. Losowość oraz wielokrotne powtarzanie pewnych procedur obliczeniowych sprawiają, że metody te szczególnie dobrze nadają się do rozwiązywania problemów matematyczno-fizycznych z wykorzystaniem komputerów. Doskonale sprawdza się wszędzie tam, gdzie jest mało praktyczne lub bardzo trudne znalezienie rozwiązania dokładnego opartego o algorytmy deterministyczne.

Metody typu Monte Carlo pierwotnie stosowane były pod nazwą próbkowania statystycznego (ang. *statistical sampling*). Nazwa „Monte Carlo" została wprowadzona przez fizyków pracujących w latach 40-tych nad projektem bomby atomowej w Los Alamos National Laboratory[2]. Nazwa odnosi się do znanego kasyna w Monako a jest tym bardziej adekwatne, że gra w kasynie wydaje się być procesem składjącym się z powtarzających się czynności typu zakład-gra, jest wysoce losowa a na końcowy wynik składają się wszystkie wyniki cząstkowe. Zauważmy przy okazji, że prawidłowe i efektywne korzystanie z metod Monte Carlo wymaga dostępu do znacznej ilości liczb losowych. Zainteresowanie tymi metodami stało się znacznym bodźcem do rozwoju generatorów liczb pseudolosowych, które są o wiele wydajniejsze niż pierwotnie stosowane tablice liczb losowych.

Prawdę powiedziawszy, nie istnieje jedna metoda Monte Carlo. Jest to raczej termin opisujący szeroką klasę różnych rozwiązań posiadających pewne cechy wspólne:

1. Rozważają deterministyczną i skończoną przestrzeń danych wejściowych (prze-

[1] W sensie metod opartych w znacznej części na czynnikach losowych.

[2] A byli to m.in. Stanisław Ulam, Enrico Fermi, John von Neumann, Nicholas Metropolis.

n	Próby					Średnia
10	3,2	3,6	3,2	2,8	2,4	3,04
100	3,0	3,12	2,92	3,04	2,84	2,984
1000	3,08	3,16	3,12	3,18	3,16	3,14
10000	3,15	3,11	3,16	3,11	3,11	3,128
1000000	3,14	3,14	3,13	3,14	3,13	3,136

Tabela 42.1: Pole powierzchni koła wpisanego w kwadrat o długości boku równej 2 dla różnych wartości $n = 3$. Wyniki prób zostały obcięte po drugiej liczbie po przecinku bez zaokrąglania.

strzeń rozważań).

2. Generują losowo punkty z przestrzeni danych wejściowych i dla każdego z nich wykonują deterministyczną procedurę obliczeniową otrzymując wyniki pośrednie (cząstkowe).

3. Agregują wyniki pośrednie w celu uzyskania wyniku ostatecznego.

Jak się okazuje metodą typu Monte Carlo można obliczyć z dosyć dobrym przybliżeniem liczbę π[3]. Narysujmy na piasku kwadrat o boku o długości 2 (to będzie przestrzeń rozważań) a następnie wpiszmy w niego okrąg. Teraz rozsypmy nad kwadratem (i wpisanym weń kołem) równomiernie kilka garści ryżu, które staną się tym samym losowo wybranymi z przestrzeni rozważań punktami. Ponieważ punkty rozkładają się równomiernie w obrębie pola kwadratu, to stosunek liczby punktów wewnątrz koła do liczby wszystkich punktów w kwadracie jest równy stosunkowi pola koła do pola kwadratu. Stąd przez zwykłą proporcję mamy

$$\frac{P_s}{P_c} = \frac{n_s}{n_c},$$

gdzie P_s oraz P_c to pola kwadratu i koła a n_s oraz n_c to liczba punktów należących do kwadratu i koła. Ostatecznie pole koła wyraża się wzorem

$$P_c = \frac{n_c \cdot P_s}{n_s}.$$

Ponieważ skądinąd wiadomo, że $P_c = \pi \cdot r^2$, gdzie r jest promieniem, więc jeśli r=1 to wówczas pole takiego koła równe jest liczbie π. Oczywiście im większa będzie liczba punktów wybranych z przestrzeni, tym wynik będzie dokładniejszy. Tabela 42.1 przedstawia wyniki zebrane dla kilku przykładowych wartości n (łącznej ilości punktów z przestrzeni).

42.2 Całkowanie metodą Monte Carlo

W ogólności metody Monte Carlo znajdują zastosowanie w matematyce wszędzie tam, gdzie rozwiązanie postawionego problemu, jeśli w ogóle możliwe analitycznie, jest zbyt

[3]Wartość liczby π z dokładnością do 100 miejsc po przecinku: π=3,14159 26535 89793 23846 26433 83279 50288 41971 69399 37510 58209 74944 59230 78164 06286 20899 86280 34825 34211 70679...

skomplikowane lub czasochłonne. Do jednych z takich zadań należy obliczanie wartości całki oznaczonej.

Deterministyczne metody całkowania numerycznego dokonują obliczeń w jednostajnie rozmieszczonych punktach. Takie podejście sprawdza się bardzo dobrze dla funkcji jednej zmiennej. Jednak efektywność metody drastycznie maleje wraz ze wzrostem wymiaru rozważanej przestrzeni. Numeryczne całkowanie funkcji dwóch zmiennych wymaga jednostajnej dwuwymiarowej siatki, co przy wyborze 10 punktów dla każdej zmiennej (a jest to wartość zdecydowanie za mała) daje 100 punktów w których należy przeprowadzić obliczenia. A jeśli wymiar przestrzeni jest rzedu dziesiątek lub setek (co wcale nie jest taką rzadością w wielu praktycznie rozważanych problemach) wówczas policzenie całki w ten sposob, jeśli w ogóle możliwe, będzie bardzo czasochłonne.

Metoda Monte Carlo jest sposobem obejścia lawinowego tempa wzrostu złożoności obliczeniowej. Tak długo jak funkcja jest „porządna" obliczanie całki oznaczonej można oprzeć na wartościach funkcji policzonych w losowo wybranych punktach. Zgodnie z prawem wielkich liczb, takie podejście daje zbieżność do rozwiązania rzędu $1/\sqrt{n}$, czyli zwiększenie ilości punktów próbkowania z kwadratem, powoduje zmniejszenie błędu o połowę i to bez względu na wymiar przestrzeni.

Zasada obliczania całki oznaczonej metodą Monte Carlo jest następująca. Dla danej funkcji $f(x)$, której całkę oznaczoną chcemy obliczyć w przedziale całkowania $[a, b]$, wyznaczamy prostokąt obejmujący pole pod wykresem tej funkcji o wysokości h i długości podstawy $b - a$. W dalszej kolejności losujemy n punktów i zliczamy te punkty, które wpadają w pole pod wykresem funkcji oznaczając ich liczbę przez n_f. Wartość całki wyraża się wzorem przybliżonym:

$$\int_a^b f(x)dx \approx \frac{n_f}{n} h(b-a)$$

Tak otrzymany wzór nie jest pozbawiony wad.

- W ogólnym przypadku trudno wyznaczyć wysokość h. Wiąże się to bowiem z koniecznścią znalezienia maksimum i minimum funkcji f co samo w sobie może okazać się zadaniem bardzo trudnym.
- Pewne problemy mogą wystąpić także wtedy, gdy funkcja zmienia znak w przedziale całkowania.

Dlatego częściej jako metodę Monte Carlo przyjmuje się metodę, która wyznacza średnią z wartości funkcji w przedziale całkowania na podstawie serii losowo wybranych współrzędnych x. Następnie średnia ta jest mnożona przez długość przedziału całkowania i otrzymujemy przybliżoną wartość całki oznaczonej. Wzór ma następującą postać:

$$\int_a^b f(x)dx \approx \frac{\sum_{i=1}^n f(x_i)}{n}(b-a)$$

gdzie x_i jest wartością losową przedziału $[a, b]$.

42.3 Opis algorytmu

Niech f będzie funkcją dla której poszukujemy wartości całki oznaczonej.

Dane wejściowe • a, b – początek i koniec przedziału całkowania.

- n – liczba losowych punktów.

Dane wyjściowe • Obliczona wartość całki.

Algorytm

1. Start.
2. Pobranie wartości zmiennych a, b oraz n.
3. Przyjmijmy $i = 1$, $s = 0$.
4. Dopóki $i <= n$ powtarzaj:
   (a) Wybierz losowy punkt x należący do przedziału $[a, b]$.
   (b) Wykonaj: $s := s + f(x)$.
5. Oblicz: $r := \frac{s}{n}(b - a)$.
6. Wynikiem jest r.
7. Stop.

42.4 Zadanie

Zaimplementować algorytm obliczania całki metodą Monte Carlo dla następujących funkcji:

1. $$f(x) = \begin{cases} \frac{\sin(x)}{x}, & \text{gdy} \quad x \neq 0 \\ 1, & \text{gdy} \quad x = 0 \end{cases}$$

   Wartość całki dla $a = -10$, $b = 10$ wynosi około 3,317.

Rozdział 43

Boidy

43.1 Boid — co to jest?

Boid – termin dosyć dziwny, ale mający duży związek z rzeczywistością, a co ważniejsze z bardzo liczną rzeczywistością. Twórcą idei Boidów był Craig Reynolds. W 1986 stworzył on model skoordynowanego ruchu dużej liczby zwierząt jak np. ławica ryb czy stado ptaków. Pojedyńczy „osobnik" wchodzący w skład „stada" nazwany został właśnie Boidem.

Aby zrozumieć, jak wygląda świat z punktu widzenia Boidów, zróbmy pewien eksperyment myślowy. Załóżmy, że zależy nam na modelowaniu ruchu pojedyńczego osobnika. Można starać się obliczyć i podać dokładnie trajektorię jaką ma on podążać. Nie jest to zadanie łatwe gdyż nawet w najprostszym przypadku, czyli poruszanie się po prostej, żaden żywy organizm nie przemieszcza się po idealnej prostej, ale możliwe. Przychodzi mi tutaj na myśl pewna secena z filmu „Piękny umysł", w której to Russel Crowe grający rolę Johna Nasha ugania się za stadem gołębi chcąc opisać ich ruch. Może więc jest to możliwe?

Pójdźmy dalej – nie jeden osobnik, ale 2 lub 3 „podróżujące razem". Ot, choćby rodzina na spacerze. Cóż, można dla nich opracować analogiczne równania ruchu, dla każdego inne, ale jednak podobne. Inne, bo muszą one różnić się na tyle, aby uniknąć efektu sztuczności jak choćby wykonywanie skrętu w tym samym momencie lub podążanie cały czas w tym samym porządku. Podobne, bo jednak wszyscy pokonują tą samą drogę. Jest to już zadanie trudniejsze od poprzedniego. Nie dosyć bowiem, że ruch pojedyńczego osobnika musi być naturalny, to uwzględnić trzeba też ich wzajemne relacje. Wciąż jednak jest to zadanie wykonalne. Co jednak jeśli tych osobników będą dziesiątki czy setki jak choćby stado ptaków czy ławica ryb. Dla każdego inne równanie? Nie tędy droga.

Proszę zwrócić uwagę na jedną prostą rzecz jaka daje się zaobserwować przy ruchu na przykład stada ptaków. Stado, ogólnie, podąża w określonym kierunku; jako zbiorowość wykonuje pewne manewry. W jego wnętrzu panuje jednak pewien uporządkowany bałagan. Uporządkowany, bo jednak stado pozostaje stadem. Bałagan bo osobniki nie są ciągle na tych samych pozycjach. Przemieszczają się wewnątrz stada. A ich manewry w żadnym razie nie są zsynchronizowane czy zaprogramowane wcześniej. Zmieniają się one dynamicznie, zależnie od sytuacji. Ruchem tym rządzą bardzo poste prawa, które w ogólności (pisząc „boid" zamiast „ptak") można na przykład przedstawić w następujący sposób:

Podążaj za grupą. Każdy boid tak wyznacza swój kierunek poruszania aby stanowił on mniej więcej średnią kierunków poruszania obiektów z jego najbliższego sąsiedztwa. Postępowanie takie wydaje się uzasadnione – także ludzie znajdujący się w przemieszczającym się i dążącym do tego samego celu tłumie poruszają się tak jak najbliżsi sąsiedzi (np. wycieczka turystyczna, ludzie spieszący się do metra).

Staraj się być w centrum grupy jaką tworzą najbliżsi sąsiedzi. Boidy dążą do pozostawania pośrodku grupy złożonej z najbliższych sąsiadów.

Nie wpadaj na nikogo. Boidy dążą do nie przekraczania pewnej minimalnej odległości od najbliższych sąsiadów.

Każde zachowanie boida, pozostające w zgodzie z tymi regułami było akceptowalne.

I właściwie to już wszystko, reszta „robi się" sama. Określone w ten sposób lokalne reguły pozwoliły na symulowanie bardzo naturalnie wyglądających zachowań stadnych. Tak więc skomplikowany proces globalnego analizowania złożonych zależności pomiędzy wszystkimi, bardzo licznymi, obiektami tworzącymi stado, zastąpiono lokalnym kontrolowaniem zależności prostych. Co ciekawe, relatywnie niewielkim nakładem pracy, można modelować inne zachowania stadne, np. rozproszenie grupy boidów na wiele mniejszych w momencie natrafienia na dużą przeszkodę i ponowne ich połączenie po jej pokonaniu. W tym przypadku wystarczy dodać regułę mówiącą o tym, że każdy boid stara się nie zbliżyć do przeszkody na odległość mniejszą niż pewna określona wartość. Ogólnie rzecz ujmując, bazując tylko na lokalnych regułach, na jakich zresztą muszą nieświadomie bazować żywe oranizmy, uzyskujemy bardzo naturalnie wyglądający model ruchu stadnego. Od reguł jakie przyjmiemy zależy co „nasze" stado będzie robiło. W dalszej części proponujemy kilka przykładowych modeli wraz z regułami.

43.2 Kaczuszki

Kaczuszki to pierwsza z propozycji wykorzystania koncepcji boidów do symulowania zachowania stadnego. W naszej implementacji przyjmiemy istnienie dwóch rodzajów osobników

- przywódcy – tylko jeden,
- obywatela – wiele.

Każdy osobnik opisany jest przez następującą strukturę

```
typedef struct
{
  float x,y;//polozenie
  float kierunek;//liczba z przedziału [0,360], 0 - dodatnia półoś osi OX
  float predkosc;
}
osobnik;
```

Lider

Przywódca, jak to przywódca ma całkowitą swobodę w poruszaniu się po placu (placem nazywam ograniczony obszar okna w którym przmieszczają się osobniki). Jego ruch będzie sterowany w następujący sposób:

1. Wylosuj liczbę z przedziału *[-max_ kat_ skretu_ przywodca, max_ kat_ skretu_ przywodca]* i oznacz ją przez α.

2. Zmień bieżący kierunek przemieszczania przywódcy o α stopni: $kierunek = kierunek + \alpha$.

3. Przesuń bierzącą pozycję lidera, wyznaczaną przez współrzędne (x, y) o wektor o długości *predkosc* którego kierunek zgodny jest z *kierunek*.

//tutu//rysunek Jak znaleźć wektor potrzebny w punkcie 3? Jak zwykle można to zrobić na wiele sposobów; proponujemy następujący: dokonać obrotu punktu $(predkosc, 0)$ wokół środka układu współrzędnych o kąt *kierunek*. W ten sposób otrzymujemy współrzędne szukanego wektora.

W powyższym zachowaniu należy jeszcze dodać obsługę sytuacji wyjątkowej, to jest takiej gdy Przywódca dojdzie do brzegu placu. Wówczas zmiana kierunku zgodnie z zasadami opisanymi w punkcie 1 może nie być wystarczająca; najlepiej w takim razie za kierunek przyjąć kierunek do środka placu.

Obywatel

Obywatel, jak to obywatel istnieje tylko po to aby za przywódcą podążać. Jego ruch będzie sterowany według następującego algorytmu:

1. Obliczamy wartości kat_1 i kat_2 (patrz rysunek //tutu//).

2. Dokonujemy obrotu osobnika (inaczej zmieniamy jego kierunek) w stronę wyznaczoną przez mniejszą z liczb kat_1 oraz kat_2. Pamiętamy przy tym, że jeśli $kat = min(kat_1, kat_2)$ jest mniejsze od *max_ kat_ skretu* to zmieniamy kierunek o wielkość *kat*. W przeciwnym razie zmieniamy kierunek o wielkość *max_ kat_ skretu*.

3. Próbujemy zmienić bieżącą pozycję obywatela, wyznaczaną przez współrzędne (x, y) o wektor o długości odpowiadającej prędkości obywatela, którego kierunek zgodny jest z kierunkiem otrzymanym w poprzednim kroku.

4. Jeśli nowe położenie obywatela powoduje konflikt z innymi już istniejącymi, czyli jeśli odległość od przynajmniej jednego jest mniejsza niż pewna wartość, to zmniejszamy prędkość, na przykład $predkosc = predkosc * 0.9$ i powracamy do poprzedniego kroku. Jeśli takich prób wykonamy *max_ dec* uznajemy, że danego osobnika nie można w tym cyklu przemieścić.

Cykl ruchowy uznajemy za zakończony gdy przemieścimy wszystkich obywateli. Kolejność wyboru osobników najlepiej ustalić jako losową.

Jak łatwo się zorientować przy tak przyjętych założeniach żaden z obywateli nie może znaleźć się na żadnym innym; uwaga ta nie dotyczy lidera.

PS Zadanie to zostało nazwane kaczuszki, bo bardzo przypomina nam zachowanie malych kaczuszek podążających za mamą kaczką.

43.3 Japońscy turyści

Mam brata. Mieszka on, jak sam to określa, w takiej małej wiosce w Zachodniej Europie – w Paryżewie. Dosyć specyficzną cechą tego miasta jest to, że na wiosnę nagle, nie

wiadomo skąd, pojawiają się masy japońskich turystów i wręcz zalewają miasto robiąc zdjęcia wszystkim i wszystkiemu. Taka japońska wycieczka jest jak lawina pogrążająca w sobie wszystko co napotka na drodze. I teraz spróbujemy zasymulować takie właśnie zachowanie.

W naszej implementacji przyjmiemy istnienie jednego rodzaju osobnika, turysty, dla którego obowiązywać powinny następujące reguły:

1. Podążaj do celu wycieczki (czyli do pewnego określonego punktu).
2. Nie przekraczaj pewnej minimalnej odległości od najbliższych sąsiadów (choć z tym można polemizować, gdy mówimy o rzeczywistej wycieczce).
3. Omijaj przeszkody.

Ruch turysty będzie sterowany według następującego algorytmu:

1. Czy w polu widzenia boida znajduje się przeszkoda? Jeśli tak, idź do 2, jeśli nie, idź do 4.
2. Obróć się w losowo wybraną stronę o taki kąt aby przeszkoda zniknęła z pola widzenia. Kąt ten nie może jednak przekroczyć wartości *max_ kat_ skretu*.
3. Idź do 6.
4. Znajdź wartość kąta pomiędzy kierunkiem boida a prostą łączącą boida z celem i nazwij ją α.
5. Sprawdź, czy możesz wykonać obrót aby zmniejszyć ten kąt? Wielkość obrotu jest liczbą losowo wybraną, której wartość bezwzględna należy do przedziału *[0,max_ kat_ skretu]*, gdy $\alpha <$*max_ kat_ skretu*, oraz równą co do wartości bezwzględnej *max_ kat_ skretu*, gdy $\alpha >$*max_ kat_ skretu*.
6. Możesz wykonać krok w wyznaczonym kierunku biorąc pod uwagę jego *predkosc* (prędkość powinna być mniejsza niż długość pola widzenia). Jeśli nie możesz (bo spowoduje to wejscie na innego boida lub przeszkodę stała), to sprawdź, czy jest to możliwe przy zmniejszonej wartości prędkości, na przykład $predkosc = predkosc * 0.9$. Jeśli nie, to ponownie zmniejsz prędkość. Próbę tą powtórz maksymalnie *max_ dec* razy.

I jeszcze kilka dodatkwych założeń:

- Prdkość boida powinna być mniejsza niż długość jego pola widzenia.
- Przyjmujemy, że boidy poruszają się wzdłuż długiego korytarza, z lewej strony na prawą. Po prawej stronie znajduje się małe wyjście, które stanowi cel. W korytarzu rozmieszaczamy losowo kilka okręgów symulujących przeszkody (np. mogą to być obiekty muzealne). Przykładowy wygląd takiego korytarza przedstawia rysunek //tutu//.

Rozdział 44

Logika rozmyta

44.1 Wprowadzenie

Impulsem dla powstania logiki rozmytej i teorii zbiorów rozmytych była chęć precyzyjniejszego opisania zjawisk oraz pojęć mających ze swojej natury nieprecyzyjny lub wieloznaczny charakter. Za ojca tej dziedziny nauki uważa się L. A. Zadeha, który przedstawił jej zarys w pracy *Fuzzy Sets* [32].

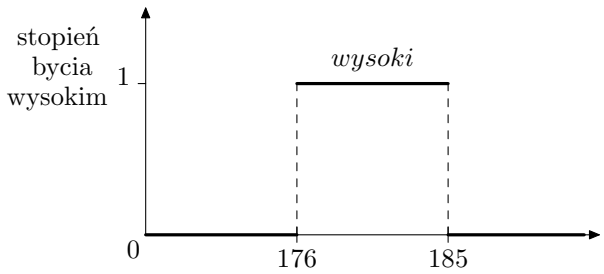
Logika rozmyta (ang. *fuzzy logic*) jest jedną z logik wielowartościowych (ang. *multi-valued logic*) i stanowi uogólnienie klasycznej logiki dwuwartościowej. Jest ściśle powiązana z teorią zbiorów rozmytych i teorią prawdopodobieństwa. W logice rozmytej między stanem 0 (fałsz) a stanem 1 (prawda) rozciąga się szereg wartości pośrednich, które określają stopień przynależności elementu do zbioru.

Znaczna ilość „zjawisk" z jakimi spotyka się człowiek posiada znacznie więcej niż tylko dwa stany je opisujące. Pod pojęciem „zjawiska" możemy rozumieć zarówno temperature jak i kolor czy też wagę. W takiej sytuacji posługiwanie się logiką dwuwartościową jest nienaturalne a przede wszystkim kłopotliwe. Załóżmy na przykład, że w zbiorze osób musimy oznaczyć osoby *wysokie*. Ale co to znaczy: *wysoka* osoba? Oczywiście możemy przyjąć umowę, że wszystkie osoby o wzroście z przedziału $[176, 185]$ (cm) są *wysokie*. Osoby, których wzrost przekracza 185 cm uważać będziemy za *bardzo wysokie* a te o wzroście mniejszym niż 176 cm za niskie. Intuicyjnie wyczuwamy jednak, że taki podział jest bardzo sztuczny. Powodów jest kilka.

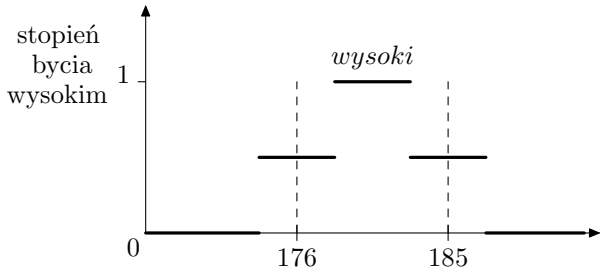
- Po pierwsze, nawet wśród *wysokich* osób będą osoby wyższe i niższe. Klasyfikując w identyczny sposób wszystkie osoby zaliczane do *wysokich* automatycznie skazujemy siebie na utratę informacji.

- Po drugie, logika dwuwartościowa zmusza nas do nienaturalnych rozgraniczeń. Według naszego, restrykcyjnego progu klasyfikacji, osoba o wzroście 185.2 cm nie jest zaliczana do grupy *wysokich*.

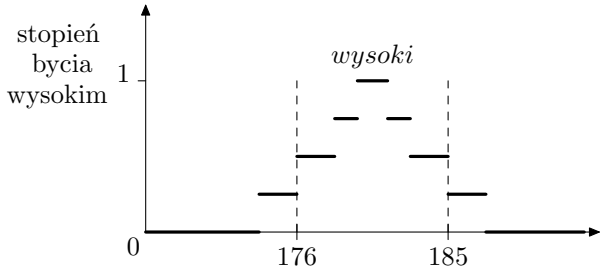
A przecież zdecydowanie bardziej naturalne jest stwierdzenie, że osoby o wzroście 180 cm i 185 cm są *wysokie*, ale ta o wzroście 185 cm jest *wysoka* w stopniu większym niż ta o wzroście 180 cm. Co więcej, ta o wzroście 185.2 cm mimo, że zaczyna być już zaliczana do *bardzo wysokich* to wciąż może być zaliczana do *wysokich*, choć w już trochę mniejszym stopniu. I w tak naturalny sposób dochodzimy do potrzeby określania stopnia przynależności. Nie mówimy bowiem, że coś należy (w całości) lub nie do jakiegoś zbioru, ale, że należy w jakimś stopniu; w jakimś stopniu spełnia „założenia" opisujące



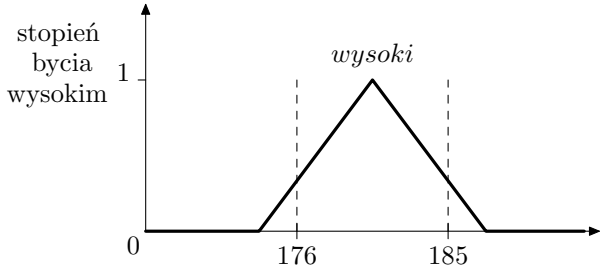
Rysunek 44.1: Progowe kryterium wyboru osób wysokich. Przynależność w stopniu 1.0 oznacza „być osobą wysoką", przynależność w stopniu 0.0 oznacza „nie być osobą wysoką".



Rysunek 44.2: Progowe kryterium wyboru osób wysokich uwzględniające możliwość tylko częściowej przynależności do zbioru osób wysokich. A zatem o osobie przynależącej do zbioru osób wysokich w stopniu 0.5, powiemy „wysoka" ze znacznie mniejszym „przekonaniem".



Rysunek 44.3: Progowe kryterium wyboru osób wysokich uwzględniające możliwość tylko częściowej przynależności do zbioru osób wysokich – poszerzenie koncepcji z rysunku 44.2.



Rysunek 44.4: Funkcja przynależności do zbioru osób wysokich.

Przedział [cm]	50-150	150-176	176-180	180-185	185-210	210-250
Przynależność	NIE	NIE	TAK	TAK	NIE	NIE

Tabela 44.1: Podział przestrzeni rozważań (wzrost człowieka) na podprzedziały opisywane wartościami 0 lub 1.

Przedział [cm]	50-150	150-160	160-170	170-176	176-178	178-180	180-182	182-184	184-185	185-190	190-200	210-250
Przynależność w stopniu	0.0	0.0	0.4	0.6	0.7	0.75	0.85	0.9	1.0	0.9	0.75	0.3

Tabela 44.2: Podział przestrzeni rozważań (wzrost człowieka) na podprzedziały opisywane wartościami z przedziału $[0, 1]$.

zbiór do którego przynależy. W ten oto sposób zamiast podziału dwustanowego (patrz tabela 44.1) możemy wprowadzić następujący podział opisujący przynależność do zbioru osób *wysokich* (patrz tabela 44.2). Dokonując podziału przestrzeni na co raz mniejsze podprzedziały uzyskamy ostatecznie stopnie przynależności dające się opisać za pomocą funkcji ciągłej (porównaj rysunki 44.1–44.4).

44.2 Podstawowe pojęcia i definicje

Chcąc mówić o zbiorach rozmytych musimy określić **obszar rozważań** (ang. *the universe of discourse*) nazywany także **zbiorem** lub **przestrzenią**. Obszar rozważań to nic innego jak zakres wartości definiujących dany zbiór rozmyty. Pamiętajmy, że jest to zbiór nierozmyty. Zbiór ten może być zarówno zbiorem liczb jak i osób, przedmiotów czy pojęć.

Definicja 44.1. *Zbiorem rozmytym A w pewnej (niepustej) przestrzeni X, co zapisujemy jako $A \subseteq X$, nazywamy zbiór par*

$$A = \{(x, \mu_A(x)); x \in X\},$$

w którym

$$\mu_A : X \to [0, 1]$$

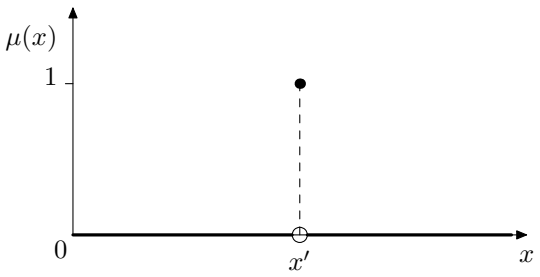
nazywamy **funkcją przynależności** *zbioru rozmytego A. Funkcja ta każdemu elementowi $x \in X$ przyporządkowywuje jego stopień przynależności do zbioru rozmytego A.*

Do zapisu zbiorów rozmytych stosuje się symbolikę, która może okazać się lekko myląca. Jeżeli X jest przestrzenią o skończonej liczbie elementów, to znaczy $X = \{x_1, \ldots, x_n\}$, to zbiór rozmyty $A \subseteq X$ symbolicznie zapisujemy w następującej postaci

$$A = \frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \cdots + \frac{\mu_A(x_n)}{x_n} = \sum_{i=1}^{n} \frac{\mu_A(x_i)}{x_i}$$

gdzie zapis

$$\frac{\mu_A(x_i)}{x_i}, \quad i = 1, \ldots, n$$



Rysunek 44.5: Funkcja typu *singleton*.

należy rozumieć jako

$$(x_i, (\mu_A(x_i)), \quad i = 1, \ldots, n$$

Jeśli X jest przestrzenią o nieskończonej liczbie elementów, to zbiór rozmyty $A \subseteq X$ symbolicznie zapisujemy jako

$$A = \int_X \frac{\mu_A(x)}{x}.$$

Przykład 44.1.

Określimy pojęcie „mniej więcej 5". Niech przestrzenią rozważań X będzie zbiór liczb naturalnych. Wówczas zdefiniować możemy następujący zbiór rozmyty $A \subseteq X$

$$A = \frac{0.1}{1} + \frac{0.3}{2} + \frac{0.7}{3} + \frac{0.9}{4} + \frac{1}{5} + \frac{0.9}{6} + \frac{0.7}{7} + \frac{0.3}{8} + \frac{0.1}{9}$$

44.2.1 Typowe funkcje przynależności

Funkcja typu *singleton*

Jest to dosyć nietypowa funkcja przynależności przyjmująca wartość 1 (całkowiat przynależność) tylko w jednym punkcie przestrzeni rozważań. Dla pozostałych punktów wartość tej funkcji wynosi 0 (rysunek 44.5).

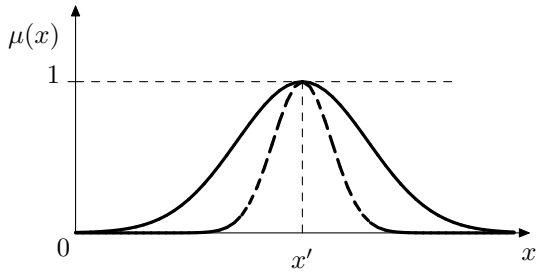
$$\mu_A(x) = \begin{cases} 1 & \text{dla} \quad x = x' \\ 0 & \text{dla} \quad x \neq x' \end{cases}$$

Funkcja *Gaussa*

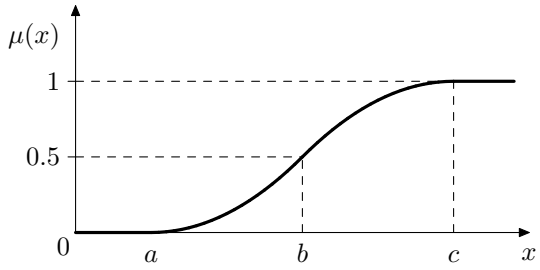
Funkcja tego typu opisana jest wzorem (rysunek 44.6)

$$\mu_A(x) = exp\left(-\left(\frac{x - x'}{\rho}\right)^2\right)$$

w którym x' jest środkiem, a ρ określa szerokość krzywej gaussowskiej.



Rysunek 44.6: Funkcja *Gaussa* (wykres linią ciągłą dla parametru $\rho = 1.5$, linią przerywaną dla $\rho = 0.3$).



Rysunek 44.7: Funkcja *klasy s*.

Funkcja *klasy s*

Funkcję tę definiujemy jako (rysunek 44.7)

$$\mu_A(x;a,b,c) = \begin{cases} 0 & \text{dla} \quad x \leq a \\ 2\left(\frac{x-a}{c-a}\right)^2 & \text{dla} \quad x \in (a,b] \\ 1-2\left(\frac{x-c}{c-a}\right)^2 & \text{dla} \quad x \in (b,c] \\ 1 & \text{dla} \quad x > c \end{cases}$$

W punkcie $x = b = (a+c)/2$ funkcja przyjmuje wartość 0.5.

Funkcja *klasy* γ

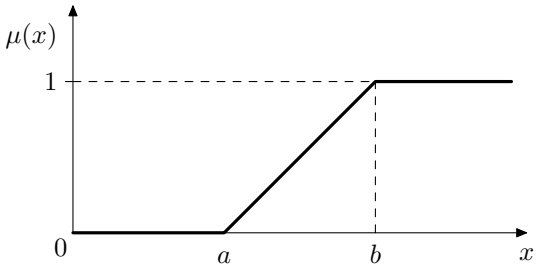
Funkcję tę definiujemy jako (rysunek 44.8)

$$\mu_A(x;a,b) = \begin{cases} 0 & \text{dla} \quad x \leq a \\ \frac{x-a}{b-a} & \text{dla} \quad x \in (a,b] \\ 1 & \text{dla} \quad x > b \end{cases}$$

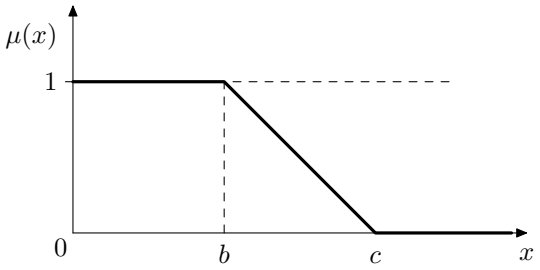
Funkcja *klasy L*

Funkcję tę definiujemy jako (rysunek 44.9)

$$\mu_A(x;b,c) = \begin{cases} 1 & \text{dla} \quad x \leq b \\ \frac{c-x}{c-b} & \text{dla} \quad x \in (b,c] \\ 0 & \text{dla} \quad x > c \end{cases}$$



Rysunek 44.8: Funkcja *klasy* γ.

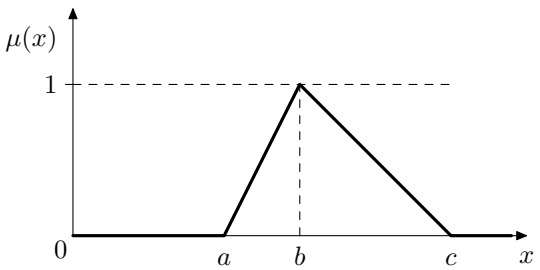


Rysunek 44.9: Funkcja *klasy* L.

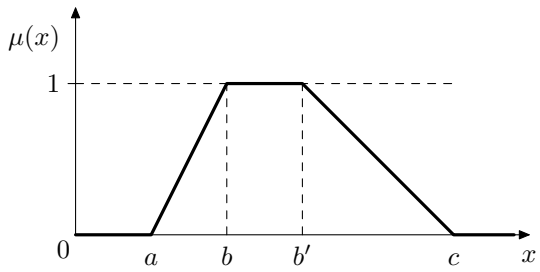
Funkcja *klasy t*

Funkcję tę definiujemy jako (rysunek 44.10)

$$\mu_A(x;a,b,c) = \begin{cases} 0 & \text{dla} \quad x \leq a \\ \frac{x-a}{b-a} & \text{dla} \quad x \in (a,b] \\ \frac{c-x}{c-b} & \text{dla} \quad x \in (b,c] \\ 0 & \text{dla} \quad x > c \end{cases}$$



Rysunek 44.10: Funkcja *klasy t*.



Rysunek 44.11: Funkcja „*trapezowa*".

Funkcja „*trapezowa(?)*"

Funkcję tę definiujemy jako (rysunek 44.11)

$$\mu_A(x;a,b,b',c) = \begin{cases} 0 & \text{dla} \quad x \leq a \\ \frac{x-a}{b-a} & \text{dla} \quad x \in (a,b] \\ 1 & \text{dla} \quad x \in (b,b'] \\ \frac{c-x}{c-b'} & \text{dla} \quad x \in (b',c] \\ 0 & \text{dla} \quad x > c \end{cases}$$

Funkcje przynależności dla pojęć związanych z prędkością.

Definicja 44.2. *Zbiór elementów przestrzeni X, dla których $\mu_A(X) > 0$ nazywamy* **nośnikiem** *zbioru rozmytego A i oznaczamy $suppA$.*

Definicja 44.3. Wysokością *zbioru rozmytego A nazywamy liczbę $h(A)$ zdefiniowaną jak poniżej*

$$h(A) = \sup_{x \in A} \mu_A(x)$$

Definicja 44.4. *Zbiór rozmyty A nazywamy* **normalnym** *jeśli $h(A) = 1$. Zauważmy, że jeśli zbiór nie jest normalny to zawsze można znormalizować go za pomocą przekształcenia*

$$\mu_{A'}(x) = \frac{\mu_A(x)}{h(A)}$$

Definicja 44.5. *Zbiór rozmyty nazywamy* **pustym**, *co zapisujemy $A = \emptyset$ //tutu// jeśli $\mu_A = 0$ dla każdego $x \in X$.*

Definicja 44.6. *Zbiór rozmyty A* **zawiera** *się w zbiorze rozmytym B, co zapisujemy $A \subset B$ jeśli*

$$\mu_A \leq \mu_B$$

dla każdego $x \in X$.

Definicja 44.7. *Zbiór rozmyty A jest równy zbiorowi rozmytemu B, co zapisujemy $A = B$ jeśli*

$$\mu_A = \mu_B$$

dla każdego $x \in X$.

Definicja 44.8. **α-przekrojem** *zbioru rozmytego $A \subseteq X$, co zapisujemy A_α, nazywamy ziór nierozmyty*

$$A_\alpha = \{x \in X : \mu_A(x) \geq \alpha\}$$

dla każdego α z przedziału $[0, 1]$

Definicja 44.9. *Zbiór rozmyty $A \subseteq X$ jest* **wypukły** *jeśli dla dowolnych $x_1, x_2 \in X$ i $\lambda \in [0, 1]$ zachodzi*

$$\mu_A(\lambda x_1 + (1 - \lambda)x_2) \geq \min(\mu_A(x_1), \mu_A(x_2))$$

Definicja 44.10. *Zbiór rozmyty $A \subseteq X$ jest* **wklęsły** *jeśli istnieją $x_1, x_2 \in X$ oraz $\lambda \in [0, 1]$ takie, że zachodzi*

$$\mu_A(\lambda x_1 + (1 - \lambda)x_2) < \min(\mu_A(x_1), \mu_A(x_2))$$

44.3 Operacje na zbiorach rozmytych

Definicja 44.11. Sumą *zbiorów rozmytych $A, B \subseteq X$ jest zbiór rozmyty $A \cup B$ o funkcji przynależności*

$$\mu_{A \cup B}(x) = \mu_A(x) \vee \mu_B(x) = (\mu_A(x) \quad OR \quad \mu_B(x)) = max(\mu_A(x), \mu_B(x))$$

dla każdego $x \in X$.
Przykład 44.2.

Niech $X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ oraz

$$\begin{aligned} A &= \frac{0.2}{2} + \frac{0.7}{5} + \frac{1}{6} + \frac{0.85}{9} \\ B &= \frac{0.5}{2} + \frac{0.4}{3} + \frac{0.75}{5} + \frac{1}{9} \end{aligned}$$

Zgodnie z powyższą definicją sumą zbiorów rozmytych $A, B \subseteq X$ jest

$$A \cup B = \frac{0.5}{2} + \frac{0.4}{3} + \frac{0.75}{5} + \frac{1}{6} + \frac{1}{9}$$

Definicja 44.12. Przecięciem *zbiorów rozmytych $A, B \subseteq X$ jest zbiór rozmyty $A \cap B$ o funkcji przynależności*

$$\mu_{A \cap B}(x) = \mu_A(x) \wedge \mu_B(x) = (\mu_A(x) \quad AND \quad \mu_B(x)) = min(\mu_A(x), \mu_B(x))$$

dla każdego $x \in X$.
Przykład 44.3.

Niech $X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ oraz

$$\begin{aligned} A &= \frac{0.2}{2} + \frac{0.7}{5} + \frac{1}{6} + \frac{0.85}{9} \\ B &= \frac{0.5}{2} + \frac{0.4}{3} + \frac{0.75}{5} + \frac{1}{9} \end{aligned}$$

Zgodnie z powyższą definicją przecięciem zbiorów rozmytych $A, B \subseteq X$ jest

$$A \cap B = \frac{0.2}{2} + \frac{0.7}{5} + \frac{0.85}{9}$$

Uwaga 44.1.

Zamiast definicji 44.11 i 44.12 można spotkać alternatywne definicje sumy i przecięcia zbiorów rozmytych. W ogólności przecięcie zbiorów rozmytych można zdefiniować za pomocą tak zwanej T-normy *natomiast sumę zbiorów rozmytych za pomocą tak zwanej* S-normy.

Twierdzenie 44.1 (Twierdzenie o dekompozycji)**.** *Każdy zbiór rozmyty $A \subseteq X$ można przedstawić w postaci*

$$A = \bigcup_{\alpha \in [0,1]} \alpha A_\alpha$$

gdzie αA_α oznacza zbiór rozmyty, którego elementom przypisano następujące stopnie przynależności

$$\mu_{\alpha A_\alpha}(x) = \begin{cases} \alpha & dla \quad x \in A_\alpha \\ 0 & dla \quad x \notin A_\alpha \end{cases}$$

Przykład 44.4.

Rozważmy zbiór rozmyty $A \subseteq X$

$$A = \frac{0.2}{2} + \frac{0.7}{5} + \frac{1}{6} + \frac{0.85}{9}$$

Zgodnie z powyższą definicją otrzymujemy

$$\begin{aligned} A &= 0.2 \cdot A_{0.2} + 0.7 \cdot A_{0.7} + 0.85 \cdot A_{0.85} + 1 \cdot A_1 \\ &= \left(\frac{0.2}{2} + \frac{0.2}{5} + \frac{0.2}{6} + \frac{0.2}{9}\right) \cup \left(\frac{0.7}{5} + \frac{0.7}{6} + \frac{0.7}{9}\right) \cup \left(\frac{0.85}{6} + \frac{0.85}{9}\right) \cup \left(\frac{1}{6}\right) \end{aligned}$$

Definicja 44.13. Dopełnieniem *zbioru rozmytego $A \subseteq X$ jest zbiór rozmyty $\overline{A}$ o funkcji przynależności*

$$\mu_{\overline{A}} = 1 - \mu_A(x)$$

dla każdego $x \in X$.

Przykład 44.5.

Niech $X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Rozważmy zbiór rozmyty $A \subseteq X$

$$A = \frac{0.2}{2} + \frac{0.7}{5} + \frac{1}{6} + \frac{0.85}{9}$$

Zgodnie z powyższą definicją otrzymujemy

$$\overline{A} = \frac{1}{0} + \frac{1}{1} + \frac{0.8}{2} + \frac{1}{3} + \frac{1}{4} + \frac{0.3}{5} + \frac{0}{6} + \frac{1}{7} + \frac{1}{8} + \frac{0.15}{9}$$

Zauważmy, przy okazji, że w przypadku zbiorów rozmytych nie zachodzą równości $A \cup \overline{A} = X$ *oraz* $A \cap \overline{A} = \not{0}$*, czyli nie są spełnione prawa dopełnienia. Można natomiast wykazać, że definicje sumy, przecięcia i dopełnienia mają cechy przemienności, łączności i rozdzielności a ponadto spełniają prawa de Morgana oraz absorpcji.*

Definicja 44.14. Iloczynem kartezjańskim *zbiorów rozmytych* $A \subseteq X$ *i* $B \subseteq Y$*, oznaczanym* $A \times B$*, nazywamy*

$$\mu_{A\times B} = min(\mu_A(x), \mu_B(y)) \tag{44.1}$$

lub

$$\mu_{A\times B} = \mu_A(x)\mu_B(y) \tag{44.2}$$

dla każdego $x \in X$ *i* $y \in Y$*.*

Przykład 44.6.

Niech $X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ *oraz*

$$\begin{aligned} A &= \frac{0.2}{2} + \frac{0.7}{5} \\ B &= \frac{0.5}{2} + \frac{0.4}{3} + \frac{0.75}{5} \end{aligned}$$

Zgodnie z powyższą definicją iloczyn kartezjański zbiorów rozmytych $A, B \subseteq X$ *wynosi (wg. wzoru 44.1)*

$$A \times B = \frac{0.2}{(2,2)} + \frac{0.5}{(2,5)} + \frac{0.2}{(3,2)} + \frac{0.4}{(3,5)} + \frac{0.2}{(5,2)} + \frac{0.7}{(5,5)}$$

lub (wg. wzoru 44.2)

$$\begin{aligned} A \times B &= \frac{0.2 \cdot 0.5}{(2,2)} + \frac{0.5 \cdot 0.7}{(2,5)} + \frac{0.2 \cdot 0.4}{(3,2)} + \frac{0.4 \cdot 0.7}{(3,5)} + \frac{0.2 \cdot 0.75}{(5,2)} + \frac{0.7 \cdot 0.75}{(5,5)} \\ &= \frac{0.1}{(2,2)} + \frac{0.35}{(2,5)} + \frac{0.08}{(3,2)} + \frac{0.28}{(3,5)} + \frac{0.15}{(5,2)} + \frac{0.525}{(5,5)} \end{aligned}$$

44.4 Liczby rozmyte

Definicja 44.15. Liczbą rozmytą *nazywamy zbiór rozmyty A określony w zbiorze liczb rzeczywistych R,* $A \subseteq R$ *taki, że*

1. *A jest normalny,*
2. *A jest wypukły,*
3. $\mu_A(x)$ *jest przedziałami ciągła.*

Definicja 44.16. *Liczba rozmyta* $A \subseteq R$ *jest*

- *dodatnia, jeżeli* $\mu_A(x) = 0$ *dla wszystkich* $x < 0$*.*

- *ujemna, jeżeli* $\mu_A(x) = 0$ *dla wszystkich* $x > 0$.

Definicja 44.17. *Niech dane będą dwie liczby rozmyte* $A_1, A_2 \subseteq R$. *Określamy dla nich operacje*

- **dodawania**, *co zapisujemy* $A_1 + A_2 = B$, *gdzie funkcja przynależności zbioru* B *przyjmuje postać*

$$\mu_B(y) = \sup_{\substack{x_1, x_2 \\ y = x_1 + x_2}} \min(\mu_{A_1}(x_1), \mu_{A_2}(x_2)),$$

- **odejmowanie**, *co zapisujemy* $A_1 - A_2 = B$, *gdzie funkcja przynależności zbioru* B *przyjmuje postać*

$$\mu_B(y) = \sup_{\substack{x_1, x_2 \\ y = x_1 - x_2}} \min(\mu_{A_1}(x_1), \mu_{A_2}(x_2)),$$

- **mnożeniem**, *co zapisujemy* $A_1 \cdot A_2 = B$, *gdzie funkcja przynależności zbioru* B *przyjmuje postać*

$$\mu_B(y) = \sup_{\substack{x_1, x_2 \\ y = x_1 \cdot x_2}} \min(\mu_{A_1}(x_1), \mu_{A_2}(x_2)),$$

- **dzieleniem**, *co zapisujemy* $A_1 : A_2 = B$, *gdzie funkcja przynależności zbioru* B *przyjmuje postać*

$$\mu_B(y) = \sup_{\substack{x_1, x_2 \\ y = x_1 : x_2}} \min(\mu_{A_1}(x_1), \mu_{A_2}(x_2)).$$

Przykład 44.7.

Niech dane będą dwie liczby rozmyte

$$\begin{aligned} A &= \frac{1}{1} + \frac{0.2}{2} + \frac{0.1}{3} \\ B &= \frac{0.5}{2} + \frac{1}{3} + \frac{0.75}{4} \end{aligned}$$

Aby policzyć sumę tych liczb wypiszmy najpierw wszystkie możliwe układy x_1 *i* x_2. *Patrząc na liczbę* A *widzimy, że niezerowe wartości funkcji przynależności określone są dla* $x_1 = \{1, 2, 3\}$. *Podobnie dla* B *otrzymujemy* $x_2 = \{2, 3, 4\}$. *Stąd wszystkie możliwe układy pary* (x_1, x_2) *to*

Stąd funkcja przynależności $\mu_{A+B}(y)$ *przyjmuje niezerowe wartości tylko dla* $y \in$

x_1	x_2	$x_1 + x_2$
1	2	3
1	3	4
1	4	5
2	2	4
2	3	5
2	4	6
3	2	5
3	3	6
3	4	7

$\{3, 4, 5, 6, 7\}$. *A zatem*

$$\begin{aligned}
\mu_{A+B}(3) &= \sup_{\substack{x_1, x_2 \\ x_1 + x_2 = 3}} \{\min(\mu_{A_1}(1), \mu_{A_2}(2))\} \\
&= \max\{\min(1, 0.5)\} = 0.5
\end{aligned}$$

$$\begin{aligned}
\mu_{A+B}(4) &= \sup_{\substack{x_1, x_2 \\ x_1 + x_2 = 4}} \{\min(\mu_{A_1}(1), \mu_{A_2}(3)), \min(\mu_{A_1}(2), \mu_{A_2}(2))\} \\
&= \max\{\min(1, 1), \min(0.2, 0.5)\} = 1
\end{aligned}$$

$$\begin{aligned}
\mu_{A+B}(5) &= \sup_{\substack{x_1, x_2 \\ x_1 + x_2 = 5}} \{\min(\mu_{A_1}(1), \mu_{A_2}(4)), \min(\mu_{A_1}(2), \mu_{A_2}(3)), \min(\mu_{A_1}(3), \mu_{A_2}(2))\} \\
&= \max\{\min(1, 0.75), \min(0.2, 1), \min(0.1, 0.5)\} = 0.75
\end{aligned}$$

$$\begin{aligned}
\mu_{A+B}(6) &= \sup_{\substack{x_1, x_2 \\ x_1 + x_2 = 6}} \{\min(\mu_{A_1}(2), \mu_{A_2}(4)), \min(\mu_{A_1}(3), \mu_{A_2}(3))\} \\
&= \max\{\min(0.2, 0.75), \min 0.1, 1\} = 0.2
\end{aligned}$$

$$\begin{aligned}
\mu_{A+B}(7) &= \sup_{\substack{x_1, x_2 \\ x_1 + x_2 = 7}} \{\min(\mu_{A_1}(3), \mu_{A_2}(4))\} \\
&= \max\{\min(0.1, 0.75)\} = 0.75
\end{aligned}$$

Ostatecznie otrzymujemy

$$A + B = \frac{0.5}{3} + \frac{1}{4} + \frac{0.75}{5} + \frac{0.2}{6} + \frac{0.75}{7}$$

Definicja 44.18. *Niech dana będzie liczba rozmyta $A \subseteq R$. Określamy dla niej operacje*

- **zmiany znaku**, *co zapisujemy $-A$, a jej funkcja przynależności przyjmuje postać*

$$\mu_{-A}(x) = \mu_A(-x)$$

- **odwrotności**, *co zapisujemy* A^{-1}, *a jej funkcja przynależności przyjmuje postać*

$$\mu_{A^{-1}}(x) = \mu_A(x^{-1})$$

- **skalowania**, *co zapisujemy* λA, $\lambda \in R$ *a jej funkcja przynależności przyjmuje postać*

$$\mu_{\lambda A}(x) = \mu_A(\lambda x)$$

Uwaga 44.2.

Zauważmy, że suma liczb przeciwnych (na ogół) nie daje zera a iloczyn liczb odwrotnych nie jest równy 1.

Uwaga 44.3.

Nie zawsze wynikiem operacji arytmetycznych na liczbach rozmytych jest liczba rozmyta. Jednak w przypadku liczb mających ciągłe funkcje przynależności zachodzi

Twierdzenie 44.2 (Dubois i Prade)**.** *Jeśli liczby rozmyte mają ciągłe funkcje przynależności, to wynikiem operacji arytmetycznych dodawania, odejmowania, mnożenia i dzielenia są liczby rozmyte.*

Definicja 44.19. *Relacją rozmytą R między dwoma niepustymi zbiorami (nierozmytymi) X i Y nazywamy zbiór rozmyty określony na iloczynie kartezjańskim* $X \times Y$.

44.5 Reguły wnioskowania

Jako postawowe reguły wnioskowania w logice dwuwartościwej stosowane są

- Reguła **modus ponens**[1] określona przez następujący schemat wnioskowania

```
Przesłanka     A
Implikacja     A implikuje B
-----------------------------------
Wniosek        B
```

- Reguła **modus tollens**[2] określona przez następujący schemat wnioskowania

```
Przesłanka     not(B)
Implikacja     A implikuje B
-----------------------------------
Wniosek        not(A)
```

[1] Nazywana też *modus ponendo ponens* (łac. *sposób potwierdzający przez potwierdzenie*) lub *regułą odrywania*.

[2] Nazywana też *modus tollendo tollens*, (łac. *sposób zaprzeczający przy pomocy zaprzeczenia*).

W obu regułach jeśli spełnione jest (to znaczy, jeśli jest prawdziwe) wszystko to co jest nad kreską, wówczas prawdziwe jest to co znajduje się pod kreską. Mówiąc inaczej, z prawdziwości przesłanki i implikacji wynika prawdziwość wniosku.

Uogólniając reguły wnioskowania logiki dwuwartościowej otrzymujemy ich odpowiedniki wyrażone w logice rozmytej.

Definicja 44.20. *Uogólnioną (rozmytą) regułą wnioskowania* modus ponens *określa następujący schemat wnioskowania*

```
Przesłanka      x jest A'
Implikacja      IF x jest A THEN y jest B
-----------------------------------
Wniosek         y jest B'
```

gdzie $A, A' \subseteq X$ oraz $B, B' \subseteq Y$ są zbiorami rozmytymi, natomiast x i y są zmiennymi lingwistycznymi *czyli zmiennymi przyjmującymi jako swoją wartość słowa lub zdania wypowiedziane w języku naturalnym.*

Definicja 44.21. *Uogólnioną (rozmytą) regułą wnioskowania* modus tollens *określa następujący schemat wnioskowania*

```
Przesłanka      y jest B'
Implikacja      IF x jest A THEN y jest B
-----------------------------------
Wniosek         x jest A'
```

gdzie $A, A' \subseteq X$ oraz $B, B' \subseteq Y$ są zbiorami rozmytymi, natomiast x i y są zmiennymi lingwistycznymi *czyli zmiennymi przyjmującymi jako swoją wartość słowa lub zdania wypowiedziane w języku naturalnym.*

W obu przypadkach implikacja jest tej samej postaci. Jednakże zdanie A w implikacji reguły nierozmytej widnieje również w przesłance tej reguły. Natomiast w przypadku reguły rozmytej przesłanka nie dotyczy zbioru rozmytego A ale pewnego zbioru A', który może być identyczny z A, w jakimś sensie podobny do A lub wręcz całkowicie inny od A. Występowanie przesłanki nie koniecznie dokładnie takiej jak w regule oznacza, że wniosek (najprawdopodobniej) także nie będzie odnosił się do zbioru B, ale pewnego zbioru B', który ponownie może być identyczny z B, w jakimś sensie podobny do B lub wręcz całkowicie inny od B.

Przykład 44.8.

Przykład na wnioskowanie i stosowanie reguły

Definicja 44.22. *Niech A i B będą zbiorami rozmytymi, $A \subseteq X$ oraz $B \subseteq Y$. Rozmytą implikacją $A \rightarrow B$ nazywamy relację R określoną w $X \times Y$ i zdefiniowaną za pomocą jednej z podanych reguł*[3]

reguła Mamdaniego (minimum)

$$\mu_{A \rightarrow B}(x, y) = \mu_R(x, y) = \mu_A(x) \wedge \mu_B(y) = min[\mu_A(x), \mu_B(y)]$$

[3]Reguł tych jest znacznie więcej – ze względu na charakter zajęć ograniczamy się jedynie do tych dwóch.

reguła Larsena (iloczyn)

$$\mu_{A\to B}(x,y) = \mu_R(x,y) = \mu_A(x) \cdot \mu_B(y)$$

44.6 Sterowniki rozmyte

Jednym z praktycznych pól zastosowań logiki rozmytej są zadania kontroli i sterowania. Aby móc sterować przebiegiem pewnych procesów lub też pracą urządzeń niezbędne jest stworzenie odpowiedniego modelu, na podstawie którego można będzie podejmować decyzje związane ze sterowaniem. Bardzo często zdarza się, iż znalezienie odpowiedniego modelu jest procesem trudnym i czasochłonnym a nierzadko wymagającym przyjęcia dodatkowych założeń upraszczających zagadnienie. W takim przypadku idealnym narzędziem są sterowniki rozmyte. Zamiast wyznaczać pewien model formułujemy jedynie reguły postępowania w postaci rozmytych zdań warunkowych typu

```
IF ...  THEN ...
```

Rysunek **??** przedstawia klasyczny sterownik rozmyty. Składa się on z czterech elementów:

- **Bazy reguł**
  Bazę reguł stanowi zbiór rozmytych reguł np. postaci[4]

  ```
  IF (x1 jest A1) AND ... AND (xn jest An)
  THEN (y1 jest B1) AND ... AND (ym jest Bm)
  ```

  gdzie A_i, B_j, $i = 1,...,n$, $j = 1,...,m$ – są zbiorami rozmytymi, x_i – zmiennymi wejściowymi, y_j – zmiennymi wyjściowymi.

- **Bloku rozmywania**
  Ponieważ system sterowania z logiką rozmytą operuje na zbiorach rozmytych, dlatego konkretne wartości sygnału wejściowego podlegają operacji rozmywania, w wyniku której zostają one odwzorowane w zbiór rozmyty. Blok rozmywania nazywany jest także **fuzyfikatorem**.

- **Bloku wnioskowania**
  Na podstawie zbioru reguł rozmytych w oparciu o uogólnione reguły wnioskowania znajdujemy odpowiedni zbiór rozmyty będący wnioskiem powstałym w oparciu o podane przesłanki. Blok wnioskowania nazywany jest także **blokiem inferencji**.

- **Bloku wyostrzania**
  Wielkością wyjściową bloku wnioskowania jest zbiór rozmyty. Zbiór ten należy odwzorować w jedną wartość, która będzie poszukiwanym sygnałem sterującym. Blok wyostrzania nazywany jest także **defuzyfikatorem**.

[4]Konkretna postać reguł zależy od przyjętego modelu, co jednak nie wpływa na ogólną strukturę sterownika rozmytego.

44.6.1 Rozmywanie

44.6.2 Blok inferencji

44.6.3 Wyostrzanie

44.7 Podstawowe modele systemów rozmytych

Omówimy tutaj dwa najpopularniejsze modele: Mamdaniego oraz Takagi-Sugeno.

44.7.1 Model Mamdaniego

Model Mamdaniego jest reprezentowany przez zbiór reguł postaci

```
IF
(x1 jest A) AND ... AND (xm jest A)
OR
(x1 jest A) AND ... AND (xm jest A)
OR
...
THEN (y1 jest B)
```

W modelu tym istotne jest to, że zarówno w poprzedniku reguły (część `IF`) jak i następniku (część `THEN`) występują zbiory rozmyte (w powyższym ogólnie oznaczone jako `A`, `B`). Wyjście modelu y oblicza się jako

$$y = \frac{\sum_{i=1}^{n} \tau^{(i)} y^{(i)}}{\sum_{i=1}^{n} \tau^{(i)}},$$

gdzie $\tau^{(i)}$ jest stopniem aktywności i-tej reguły, $y^{(i)}$ wartością otrzymaną w wyniku defuzyfikacji zbioru rozmytego znajdującego się we wniosku i-tej reguły a n ogólną liczbą reguł w modelu.

44.7.2 Model Takagi-Sugeno

Model Takagi-Sugeno różni się od modelu Mamdaniego postacą reguł. W tym przypadku w następniku reguły nie występuje zbiór rozmyty, ale funkcja

```
IF
(x1 jest A) AND ... AND (xm jest A)
OR
(x1 jest A) AND ... AND (xm jest A)
OR
...
THEN (y jest f(x1,...,xm))
```

Tak więc w modelu tym zbiór rozmyty występuje jedynie w poprzedniku reguły (część `IF`). W następniku (część `THEN`) występują natomiast występują zależności funkcyjne. Wyjście modelu y oblicza się jako

$$y = \frac{\sum_{i=1}^{n} \tau^{(i)} f^{(i)}(x_1, \ldots, x_m)}{\sum_{i=1}^{n} \tau^{(i)}},$$

gdzie $\tau^{(i)}$ jest stopniem aktywności i-tej reguły, $f^{(i)}(x_1, \ldots, x_m)$ jest funkcją występującą we wniosku tejże reguły, n jest ogólną liczbą reguł w modelu a m ogólną liczbą sygnałów wejściowych.

44.8 Aproksymacja przy pomocy funkcji rozmytych

W pracy [29] wykazano, że każda funkcja ciągła określona w R^n może być aproksymowana przez układ w którym defuzyfikator jest w jednej z dwóch postaci:

Rozdział 45

Tworzenie zbioru reguł rozmytych

45.1 Tworzenie zbioru reguł

Poniżej podajemy algorytm tworzenia zbioru reguł rozmytych, co stanowi praktyczną ilustrację materiału podanego w rozdziale 44. Dla uproszczenia przyjmujemy, że naszym celem jest stworzenie sterownika rozmytego o dwóch wejściach i jednym wyjściu. Założenie takie nie zmienia ogólności podejścia (postępujemy dokładnie w taki sam sposób jeśli ilość wejść i/lub/albo wyjść jest inna) za to upraszcza rozważania i pozwala w łatwy sposób przedstawić całą bazę. Odpowiednie reguły ustalimy w oparciu o zebrane przykładowe dane, nazywane dalej danymi uczącymi. Zgodnie z powyższym założeniem zbiór danych uczących jest zbiorem par

$$(in(i), out(i)), \quad i = 1, 2, \ldots$$

gdzie $in(i) = (x_1(i), x_2(i))$ jest sygnałem wejściowym, podawanym na wejście sterownika natomiast $out(i) = y(i)$ to wzorcowa wartość sygnału wyjściowego.

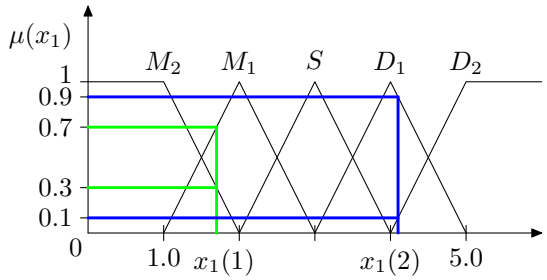
- **Krok 1 – podział przestrzeni wejściowej i wyjściowej na obszary**
  Określamy przedziały, w których zawierają się dopuszczalne wartości dla $in(i)$ oraz $out(i)$. Każdy z przedziałów dzielimy na $2N + 1$ obszarów (odcinków). Dla każdego z sygnałów N może być różne; różne mogą być także długości odcinków. Poszczególne obszary oznaczamy następująco[1]:

  $$M_N, \ldots, M_1, S, D_1, \ldots, D_N$$

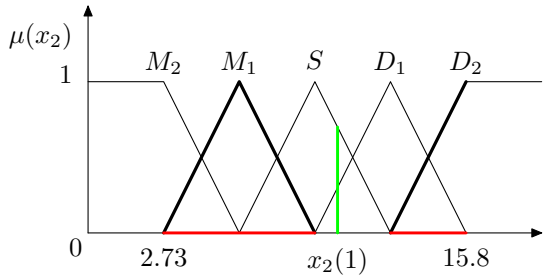
  i dla każdego z nich określamy jedną funkcję przynależności (trójkątną dla odcinków wewnętrznych i trapezową na brzegach). Przykład takiego podziału przedstawiono na rysunkach 45.1–45.3. Jak widać w stosunku do trzech sygnałów: x_1, x_2, y przyjęto następujące założenia:

Sygnał	Minimalna wartość	Maksymalna wartość	N
x_1	1.0	5.0	2
x_2	2.73	15.8	2
y	0.4	13.7	3

[1] M oznacza *mały*, S – *średni*, D – *duży*.



Rysunek 45.1: Podział na obszary i określenie funkcji przynależności dla sygnału x_1.

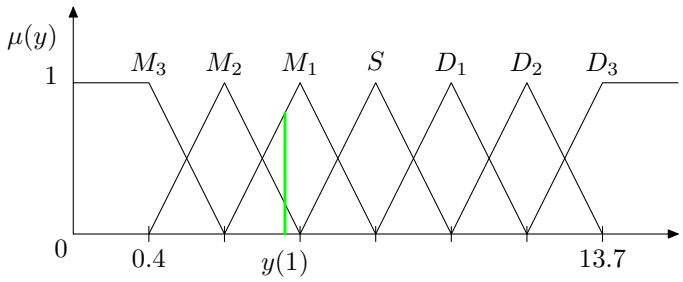


Rysunek 45.2: Podział na obszary i określenie funkcji przynależności dla sygnału x_2. Wyraźniej zaznaczono funkcje przynależności dla zbiorów M_1 i D_2, dla których też zaznaczono odpowiadajace im odcinki leżące na osi OX.

Uwaga: Zaprezentowany sposób jest tylko przykładem. Można wybrać zarówno inny sposób wydzielenia odcinków jak i zupełnie inny kształt funkcji przynależności.

- **Krok 2 – tworzenie reguł rozmytych na podstawie danych uczących.**

  1. Wyznaczamy stopnie przynależności danych uczących do każdego z obszarów utworzonych w kroku 1. Biorąc pod uwagę rysunek 45.1 stwierdzamy, że



Rysunek 45.3: Podział na obszary i określenie funkcji przynależności dla sygnału y.

stopień przynależności danej $x_1(1)$ do obszaru M_1 wynosi 0.7, do obszaru M_2 – 0.3 a do pozostałych obszarów 0. Dla danej $x_1(2)$ otrzymujemy przynależność do obszaru D_2 w stopniu 0.1, do obszaru D_1 w stopniu 0.9 a do pozostałych obszarów 0. Na podobnej zasadzie wyznaczamy stopnie przynależności dla wszystkich danych uczących (tj. $x_1(i)$, $x_2(i)$ i $y(i)$).

2. Przyporządkowujemy dane uczące do obszarów w których mają one maksymalne stopnie przynależności. W ten sposób dla każdej pary uczącej możemy napisać jedną regułę postaci (przykład na podstawie rysunków 45.1–45.3):

   ```
   para ucząca: ((x1(1),x2(1)),y(1))

   i odpowiadająca jej

   reguła: IF (x1 jest M_1) AND (x2 jest S) THEN y jest M_1
   ```

- **Krok 3 – przyporządkowanie stopni prawdziwości do każdej z reguł**
  W oparciu o każdą parę danych uczących możemy sformułować jedną regułę. W przypadku występowania dużej ilości par, wiele reguł będzie miało te same przesłanki i te same (bądź też różne) wnioski. W celu umożliwienia podjęcia decyzji, którą z nich wybrać, do każdej reguły przyporządkowujemy stopień jej prawdziwości i wybieramy tą spośród reguł o tych samych przesłankach, która ma ten stopień najwyższy. Dla reguł postaci

  ```
  IF (x1 jest A1) AND (x2 jest A2) THEN y jest B
  ```

  stopień prawdziwości sp definiujemy jako

  $$sp = \mu_{A_1}(x_1)\mu_{A_2}(x_2)\mu_B(y),$$

  gdzie zapis $\mu_X(a)$ oznacza wartość funkcji przynależności do zbioru X policzoną dla argumentu a.

- **Krok 4 – utworzenie bazy reguł rozmytych**
  W naszym przypadku bazę reguł stanowi tablica, którą wypełniamy regułami rozmytymi w następujący sposób: jeśli reguła jest postaci

  ```
  IF (x1 jest A_i) AND (x2 jest A_j) THEN y jest B_z
  ```

  to na przecięciu kolumny A_i oraz wiersza A_j wpisujemy nazwę zbioru rozmytego występującego we wniosku, czyli B_z.

45.2 Wyostrzanie

Pokażemy teraz jak w oparciu o stworzoną bazę reguł rozmytych można wyciągać wnioski. W celu określenia liczbowej wartości stanowiącej odpowiedź sterownika należy przyjąć pewną metodę wyostrzania, tzn. zamiany wartości rozmytych na liczby. W tym przypadku wyostrzanie odbywać się będzie według wzoru

$$y = \frac{\sum_{i=1}^{n} \tau^{(i)} y^{(i)}}{\sum_{i=1}^{n} \tau^{(i)}},$$

gdzie

- n jest ilością reguł w bazie;
- $\tau^{(i)}$ jest stopniem prawdziwości i-tej reguły obliczanym według wzoru:

$$\tau^{(i)} = \mu_{A_1}(x_1)\mu_{A_2}(x_2);$$

- A_1 oraz A_2 są zbiorami rozmytymi występującymi w przesłance i-tej reguły;
- $x1$, $x2$ – sygnały wejściowe;
- $y^{(i)}$ – jest wartością dla której funkcja przynależności będąca we wniosku i-tej reguły osiąga wartość maksymalną;
- y – sygnał wyostrzony.

45.3 Zadanie

Zaimplementować podany algorytm tworzenia bazy reguł rozmytych w oparciu o informacje wczytywane z pliku. Format pliku jest następujący

```
linia 1: n
linia 2: od1 do1 podzial1
linia 3: od2 do2 podzial2
linia 4: od3 do3 podzial3
linia 5: x1(1) x2(1) y(1)
linia 6: x1(2) x2(2) y(2)
...
linia n+4: x1(n) x2(n) y(n)
```

gdzie:

- `n` – ilość danych na podstawie których będzie tworzona baza reguł;
- `od1`, `do1` oraz `podzial1` – zakres zmienności oraz wielkość podziału (czyli parametr N z Kroku 1 w powyższym opisie) parametru x1;
- `od2`, `do2` oraz `podzial2` – zakres zmienności oraz wielkość podziału (czyli parametr N z Kroku 1 w powyższym opisie) parametru x2;
- `od3`, `do3` oraz `podzial3` – zakres zmienności oraz wielkość podziału (czyli parametr N z Kroku 1 w powyższym opisie) parametru y.
- `x1(n) x2(n) y(n)` – n-ta próbka.

Wynikiem działania programu ma być spis wszystkich reguł oraz oraz wypisana w trybie tekstowym baza reguł. I tak dla następującgo zbioru danych

```
4
5 9 1
2 8 1
1 3 1
5 7.5 2.8
6.5 5 2.6
6.5 5 2
8.5 2 1.2
```

w wyniku przeprowadzonych obliczeń

```
Maksymalny stopien przynaleznosci dla x1=5.0 wynosi 1.0 i jest w obszarze -1
Maksymalny stopien przynaleznosci dla x2=7.5 wynosi 0.83 i jest w obszarze 1
Maksymalny stopien przynaleznosci dla y=2.8 wynosi 0.8 i jest w obszarze 1
Stopien prawdziwosci reguly: 0.6
Wpis reguly: x1 jest -1 AND x2 jest 1 => y jest 1
Maksymalny stopien przynaleznosci dla x1=6.5 wynosi 0.75 i jest w obszarze 0
Maksymalny stopien przynaleznosci dla x2=5.0 wynosi 1.0 i jest w obszarze 0
Maksymalny stopien przynaleznosci dla y=2.6 wynosi 0.6 i jest w obszarze 1
Stopien prawdziwosci reguly: 0.45
Wpis reguly: x1 jest 0 AND x2 jest 0 => y jest 1
Maksymalny stopien przynaleznosci dla x1=6.5 wynosi 0.75 i jest w obszarze 0
Maksymalny stopien przynaleznosci dla x2=5.0 wynosi 1.0 i jest w obszarze 0
Maksymalny stopien przynaleznosci dla y=2.0 wynosi 1.0 i jest w obszarze 0
Stopien prawdziwosci reguly: 0.75
Wpis reguly: x1 jest 0 AND x2 jest 0 => y jest 0
Maksymalny stopien przynaleznosci dla x1=8.5 wynosi 0.75 i jest w obszarze 1
Maksymalny stopien przynaleznosci dla x2=2.0 wynosi 1.0 i jest w obszarze -1
Maksymalny stopien przynaleznosci dla y=1.2 wynosi 0.8 i jest w obszarze -1
Stopien prawdziwosci reguly: 0.6
Wpis reguly: x1 jest 1 AND x2 jest -1 => y jest -1
```

otrzymujemy następujące reguły

```
Reguly:
Jesli x1 jest M_1 AND x2 jest D_1  => y jest D_1

Jesli x1 jest S   AND x2 jest S    => y jest S

Jesli x1 jest D_1 AND x2 jest M_1  => y jest M_1
```

i tablicę bazy reguł postaci

```
Baza regul:
(wiersze - x1, kolumny - x2)
*   *   D_1
*   S   *
M_1 *   *
```

Rozdział 46

Prolog

SSS

Dodatek A

Czy grozi nam intelektualny upadek?

James Flynn zajmuje się badaniem ludzkiej inteligencji. Odkrył on, że w przeciągu minionych 100 lat każda kolejna generacja miała wyższy iloraz inteligencji. Dziś jednak ludzkość doszła do punktu zwrotnego. Im więcej bodźców dociera do nas ze świata zewnętrznego, tym bardziej wyłączamy swój umysł. Czy w związku z tym naszym wnukom grozi intelektualny upadek[1]?

- **W Niemczech trwa debata na temat sensu i celowości testów na inteligencję. Dyskusję sprowokowała kadra uniwersytetu we Freiburgu, gdy zrodził się tam pomysł, żeby studenci z IQ wyższym niż 130 byli zwalniani z opłat za studia. Sąd nie zezwolił jednak na wprowadzenie przepisu w życie.**

- Cieszę się, że sąd wydał tak rozsądny wyrok. Sprawy nie mogły potoczyć się inaczej. Lepiej by było, gdyby wykładowcy uniwersyteccy przyjrzeli się szkolnym osiągnięciom swoich studentów, są one bowiem dużo lepszym wykładnikiem motywacji i zdyscyplinowania, a właśnie głównie od tych czynników zależy akademicki sukces. Poza tym testy na inteligencję nie mówią nic o ludzkiej kreatywności - aby ją ocenić, trzeba by kazać studentom napisać na przykład jakiś esej. Wszystkie trzy wskaźniki razem - test na inteligencję, oceny z egzaminów i praca pisemna - mogłyby dać obraz potencjalnych możliwości danego studenta.

- **A jak wysoki jest Pański iloraz inteligencji?**

- Test robiłem raz, ale nie znam wyniku. Chyba nie było źle, skoro zostałem przyjęty na University of Chicago.

- **Gdyby Pańskie IQ byłoby wyższe niż 130, mógłby Pan zostać członkiem elitarnego stowarzyszenia Mensa.**

- Na tę sprawę mam podobny pogląd jak Karl Popper. Mensa zapraszała go, aby został jej członkiem, on jednak odmówił. Twierdził, że nigdy nie chciałby przystąpić

[1] Tłumaczenie artykułu z „Die Zait" umieszczone na stronie `http://wiadomosci.gazeta.pl/Wiadomosci/1,80281,4977808.html` z datą ostatniej aktualizacji 2008-02-29, 18:21. Rozmawiał Max Rauner.

do stowarzyszenia, któremu prawo do egzystencji daje wyłącznie wysoki iloraz inteligencji jego członków. Gratulować sobie nawzajem wysokiego IQ? Nie, dziękuję, to nie dla mnie.

- **Czy uważa się Pan za osobę inteligentną?**

- W środowisku, w którym dorastałem, dało się zauważyć, że niektóre zadania rozwiązywałem szybciej i lepiej niż moi rówieśnicy. Patrząc z tego punktu widzenia, można uznać, że byłem bardziej inteligentny niż reszta. Inteligencja jest jednak pojęciem względnym. Przyjrzyjmy się na przykład Robinsonowi Crusoe, który znalazł się sam na wyspie. Zapominał on wiele istotnych szczegółów, więc szybko uświadomił sobie, jak wielką rolę odgrywa dobra pamięć. Wiedział też, co znaczy uczyć się, bo musiał przyswoić sobie pewne nowe umiejętności. Jednak dopiero gdy na wyspie pojawił się Piętaszek, który uczył się dużo szybciej, Crusoe pojął, że jego kompan był dużo bardziej inteligentny niż on sam.

- **I nie potrzebował dowodzić tego testem na inteligencję.**

- Piętaszek po prostu umiał szybciej rozwiązywać problemy i z wieloma sytuacjami radził sobie dużo lepiej niż Robinson. Obaj startowali mniej więcej z tej samej pozycji, dlatego można bezsprzecznie stwierdzić, że iloraz inteligencji Piętaszka był rzeczywiście wyższy.

- **Powszechnie znana definicja inteligencji mówi, że jest ona tym, co mierzy się za pomocą testu IQ.**

- To skrót myślowy. Tak samo można by powiedzieć, że upał to to, co mierzy się za pomocą termometru. Kto rozumuje w ten sposób, stanie w miejscu i już nigdy nie zrobi kroku naprzód.

- **Udowodnił Pan, że iloraz inteligencji naszych przodków z przełomu XIX i XX wieku wynosił średnio między 50 a 70 i od tego czasu systematycznie wzrastał. Ten fenomen nazywa się dziś „efektem Flynna". Czy mógłby Pan go wyjaśnić?**

- Nasi przodkowie nie byli głupi. To, że ludzie z czasem zaczynali zdobywać w testach coraz więcej punktów nie znaczy wcale, że byli coraz bardziej inteligentni. Proces ten raczej obrazuje przepaść między naszym sposobem rozumowania a rozumowaniem naszych przodków. Dziś patrzymy na świat z zupełnie innej perspektywy. Jesteśmy przekonani o tym, że aby zrozumieć rzeczywistość, należy ją sklasyfikować. U współczesnego człowieka logika góruje nad konkretem.

- **Co to znaczy?**

- Jedno z zadań testowych, w którym dziś ludzie osiągają dużo lepsze wyniki niż wcześniej, opiera się na wskazywaniu podobieństw. Co wspólnego mają pies i zając? Przed 100 laty każdy powiedziałby, że psy polują na zające. Dziś nikt nie wpadłby na taką odpowiedź. Od współczesnego człowieka oczekuje się, że powie, że i pies i zając to ssaki. Każdy z nas uważa podział rzeczywistości na kategorie za rzecz absolutnie oczywistą i nie ocenia jej według kryterium użyteczności. Nasz mentalny świat uległ diametralnej zmianie, dlatego nie wiadomo, w jaki sposób można by skutecznie i sensownie zbadać poziom ludzkiej inteligencji.

- **Czy mógłby Pan konkretnie powiedzieć, co się zmieniło?**

- Zmieniło się na przykład podejście do dzieci. Na przełomie XIX i XX wieku każdy wychodził z założenia, że człowiek sam uczy się codziennego życia. Dziś dorośli wierzą, że dzieciom należy wpoić tzw. mentalną kompetencję, która wykracza daleko poza sferę dziecięcej codzienności i zwykłych potrzeb: rodzina wychodzi na spacer, dziecko widzi krowę i mówi: „To jest krowa." Dawniej powiedziano by malcowi: „Nie odzywaj się, póki nikt cię nie zapyta!" W dzisiejszych czasach rodzice cieszą się i wołają rozradowani: „Bardzo dobrze, Jasiu! To jest krówka! A jak ona robi?" W ten sposób opiekunowie starają się stymulować dziecięcą ciekawość świata.

- **Oceniając przeszłość na podstawie współczesnych kryteriów, doszlibyśmy więc do wniosku, że przeciętnie inteligentna osoba z przełomu wieków była upośledzona umysłowo.**

- I to właśnie pokazuje, że nie można porównywać testów na inteligencję z tym, co naprawdę powinniśmy rozumieć pod tym pojęciem. Szczerze mówiąc, termin inteligencja zaczął mnie już nudzić. Nie mamy żadnych podstaw, aby twierdzić, że nasze umysły funkcjonują lepiej od umysłów naszych przodków.

- **Od którego momentu w historii można zaobserwować, że iloraz inteligencji w społeczeństwie stawał się coraz wyższy?**

- Stało się to z pewnością wraz z nadejściem rewolucji przemysłowej pod koniec XIX wieku. Wszędzie pojawiły się maszyny i każdy, kto pracował w fabryce i obserwował ich działanie, mógł lepiej zrozumieć teorię przyczyny i skutku. W ten sposób zaczęła rozwijać się myśl naukowa.

- **Niektórzy naukowcy twierdzą, że wzrost ilorazu inteligencji idzie w parze ze zmianami w nawykach żywieniowych.**

- Zanim odkryłem „efekt Flynna", żaden z poważnych naukowców zajmujących się odżywianiem nie twierdził, że dziś młodzi ludzie jedzą lepiej niż przed 40 laty. Jeżeli w ogóle jest jakaś różnica, to właśnie współczesna młodzież niszczy swoje zdrowie fast-foodami.

- **A jaki wpływ na wzrost inteligencji w społeczeństwie miała zmiana modelu rodziny?**

- Rodzina jest coraz mniejsza i pociąga to za sobą poważne skutki. Rodzice jedynaka mają dużo więcej czasu, aby pobudzać umysł swojego dziecka oraz dawać mu coraz więcej językowych i intelektualnych bodźców. Zmienił się również sposób spędzania wolnego czasu. Dawniej, gdy mój tata wracał do domu, był tak zmęczony, że nie miał siły na żadne intelektualne rozrywki. Dziś w czasie wolnym gra się w gry planszowe czy gry wideo, w których operuje się abstrakcyjnymi symbolami i figurami, które nie mają żadnego odniesienia do realnej rzeczywistości. Na przełomie XIX i XX wieku jedynymi abstrakcyjnymi symbolami były karty do gry. Jednak osoby głęboko religijne wyklinały nawet ten sposób spędzania wolnych chwil. Moja prababcia na przykład paliła każdą talię kart do gry, którą znalazła. Dziś symbole nie funkcjonujące w konkretnej rzeczywistości otaczają nas ze wszystkich stron i pomagają naszemu rozumowi pojąć świat abstrakcji.

- **Czy o każdym, kto opanował umiejętność abstrakcyjnego myślenia, można powiedzieć, że jest inteligentny?**

- Nie. Ludzie, którzy potrafią abstrahować od konkretnych sytuacji i posiadają zdolność logicznego argumentowania, mają pewien rodzaj inteligencji, który pozwala im osiągać dobre wyniki w testach. Jednak nie jest to inteligencja krytyczna. Również osoby z religijnymi uprzedzeniami mogą być święcie przekonane o tym, że Bóg stworzył świat. Krytyczna inteligencja to zdolność podawania aksjomatów w wątpliwość. Niestety bardzo niewiele uniwersytetów oferuje nam wykształcenie wystarczająco dobre, aby opanować tę umiejętność.

- **Dzisiejsza młodzież spędza długie godziny, grając w gry wideo czy sterując awatarami w wirtualnych światach. Czy można podejrzewać, że ci młodzi ludzie będą kiedyś bardziej inteligentni niż ich rodzice?**

- Ludzki umysł powoli dociera do momentu, kiedy krzywa ilorazu inteligencji osiągnie swój najwyższy punkt i zacznie spadać. Nie ma już praktycznie na świecie ludzi, którzy nie chodziliby do szkoły, a ramy naszego wolnego czasu nie znajdą już miejsca dla większej ilości bodźców poznawczych. Przyczyny wzrostu ilorazu inteligencji w społeczeństwie zaczynają więc powoli zanikać. Współczesny świat bombarduje nas coraz to nowymi środkami pobudzającymi, chociażby takimi jak strzelanina czy morderstwo w grze komputerowej. Jak więc można oczekiwać od młodych ludzi, żeby czytali Karola Dickensa? Oni nie potrafią już przebrnąć przez 400 stron książki, na których nie dochodzi do żadnego trzymającego w napięciu pościgu za mordercą. Najbardziej martwi mnie to, że sami nie zauważają, ile w ten sposób tracą.

- **Tak myślą tylko ludzie, którzy wychowali się na książkach. Świat jest jednak dynamiczny i ustawicznie się zmienia.**

- Na uniwersytecie wykładam filozofię moralności. Czy gdyby moi studenci wierzyli, że średni zarobek jest pięć razy wyższy niż w rzeczywistości, to byłoby to z pożytkiem dla ich umiejętności dokonywania oceny moralnej pewnych czynów? Lub gdyby sądzili, że pracownicy fizyczni są leniwi? Jeżeli przeczytaliby książkę o pracy w kopalni, wiedzieliby, że nie do każdego życie uśmiechnęło się tak szeroko jak do nich. Nie jestem zwolennikiem książki samej w sobie, ale niezwykle cenię ludzi, których horyzonty przekraczają granice ich grupy społecznej.

- **Jednak musi Pan przyznać, że dzięki portalom takim jak Google czy Wikipedia, można łatwo zdobyć wiele cennych informacji.**

- Rzeczywiście. Można na przykład sprawdzić, ilu Żydów zginęło w czasach Holocaustu. Jednak nigdy nie będzie to to samo, co przeczytać „Dziennik" Anne Frank. Żadne inne medium nie kształci naszej zdolności odczuwania tak, jak robi to literatura. Moim zdaniem chaotyczne nagromadzenie faktów w internecie nigdy nie dotrzyma kroku książkom.

- **Czy możemy więc jeszcze uratować nasze dzieci przed upadkiem intelektualnym, czytając im książki?**

- Dziecka nie da się oszukać. Ono bardzo wyraźnie widzi, czy rodzice robią coś z poczucia obowiązku, czy wkładają w to całe serce. Wszystkim rodzicom zawsze radzę, żeby starali się rozwijać w swoich pociechach pasje. Kiedy pytają, jak mają to robić, mówię im, żeby po prostu sami mocno wierzyli we własne ideały. Dzieciom nie można kazać jeść szpinaku, a samemu krzywić się z obrzydzenia, mając go na talerzu.

- **A czy inteligencja, którą nasze dzieci odziedziczą po starszych pokoleniach, nie może uchronić ich przed upadkiem?**

- Rzeczywiście pewne elementy naszej inteligencji są dziedziczne. Jednak najprawdopodobniej wpływ uwarunkowań genetycznych na funkcjonowanie mózgu wynosi zaledwie 25 proc. Geny są istotne, ale wiara we własne ideały odgrywa jeszcze ważniejszą rolę.

- **Do czego będziemy w przyszłości potrzebować testów na inteligencję?**

- W połączeniu z innymi testami będą one zawsze miały istotne znaczenie, na przykład w procedurze przyjmowania nowych studentów na uczelnie. Nie róbmy sobie jednak zbyt wielkich nadziei. Testy na inteligencję nie mogą zdradzić wiele o człowieku.

Dodatek B

Pytania

Rozdział ten zawiera listę pytań, na które, uwzględniając poniższe uwagi, powinno znać się odpowiedzi po zapoznaniu z określoną partią materiału.

Uwaga: Na przestrzeni lat w trakcie egzaminów powstało wiele niejasności nie tyle co do materiału ile co do sposobu jego nauki. Proszę zatem pamiętać, że

1. W większości przypadków, pytanie w stylu: „wskaż", „znajdź" itp. wymaga umiejętności uzasadnienia podanej odpowiedzi! Zawsze musimy wiedzieć, dlaczego tak odpowiadamy. Uzasadnienie w stylu: „*Bo tak było napisane w książce*" wystawia jak najgorsze świadectwo odpowiadającemu.

2. Uczymy się z sensem!!! Znajmość definicji czy twierdzenia (nawet z dowodem) bez rozumienia ich i umiejętności zastosowania jest bez sensu i trudno w takiej sytuacji mówić o ich znajomości!

3. Uczymy się kompleksowo! Jeśli w jakimś temacie pojawią się inne terminy to ich też się uczymy. Jeśli np. w założeniach twierdzenia o zbieżności reguły perceptronu pojawia się separowalność liniowa, to ucząc się tego twierdzenia uczymy się też co to jest separowalność liniowa. Niestety nie dla wszystkich jest to oczywiste.

O sztucznej inteligencji ogólnie

1. Co to jest inteligencja? Jak ją możemy zdefiniować?
2. Co to jest sztuczna inteligencja?
3. Co należy do głównych zadań sztucznej inteligencji?
4. Co oznacza termin „maszyna myśląca w sensie Turinga"? Na czym polega test Turinga?
5. Czym różni się rachunek zdań od rachunku predykatów?
6. Podaj przykłady zdań atomowych i złożonych rachunku predykatów i rachunku zdań.
7. Przekształcić wyrażenie rachunku zdań do postaci klauzulowej.
8. Co oznacza całkowita falsyfikowalność rezolucji? Jak możemy z niej skorzystać?
9. Co to jest klauzula Horna?

Algorytmy przeszukiwania

1. Dla zadanego grafu wypisz wierzchołki jakie zostaną odwiedzone w wyniku stosowania przeszukiwania wszerz lub w głąb.

2. Opisz algorytm Hill Climbing, Best First Search.
3. Dla zadanego „grafu" wypisz wierzchołki jakie zostaną odwiedzone w wyniku stosowania algorytmu Hill Climbing, Best First Search.
4. Opisz idee algorytmu Means-Ends Analysis.
5. Opisz idee algorytmu Mini-Max.
6. Dla zadanego drzewa gry wybierz optymalny ruch stosując algorytm Mini-Max.

Sztuczne sieci neuronowe

1. Co to jest neuron? Opisz budowę neuronu biologicznego i jego matematyczny model.
2. Wymień przykładowe zastosowania sztucznych sieci neuronowych.
3. Wymień cechy sztucznych sieci neuronowych.
4. Co to jest funkcja aktywacji. Podaj przykłady.
5. Opisz podstawowe architektury sieci neuronwych.
6. Opisz podstawowe modele nauki sieci neuronowych.
7. Omów regułę perceptronu.
8. Co oznacza separowalność (absolutna separowalność) liniowa? Kiedy zbiór separowalny liniowo jest absolutnie separowalny liniowo?
9. Udowodnij, że reguła perceptronu jest zbieżna.
10. Wyprowadź wzory dla reguły delta.
11. Omów zasadę propagacji wstecznej.
12. Dlaczego algorytm najszybszego spadku tak właśnie się nazywa?
13. Podaj przykład błędnego działania metody najszybszego spadku (rozbieżność metody, oscylacje itp).
14. Metoda najszybszego spadku z minimalizacją kierunkową posiada pewne interesujące własności. Napisz jakie to są własności i je wykaż. (prostopadłość kierunków, generowanie siągu punktów w których minimalizwoana funkcja ma co raz mniejsze wartości).
15. Opisz metodę Newtona minimalizacji funkcji.
16. Opisz metodę momentu.
17. Co to jest kwantyzacja, klastrowanie?
18. Na czym polega mechanizm współzawodnictwa pomiędzy neuronami?
19. Kiedy i dlaczego stosujemy normalizację? Czy zwsze przynosi ona oczekiwane efekty?
20. Omów algorytm Kohonena.
21. Na czym polega analiza głównych składowych?
22. Opisz mechanizm uczenia sieci w oparciu o regułę Hebba.
23. Jaki jest związek sieci samoorganizującej się w oparciu o regułę Hebba i analizy głównych składowych?
24. Co to jest radialna funkcja bazowa?
25. Co to jest f-separowalność?

26. O czym mówi twierdzenie Covera?
27. Opisz budowę neuronowej sieci radialnej?
28. Opisz budowę sieci Hopfielda.
29. Opisz cykl pracy sieci Hopfielda?
30. Co oznacza pojęcie stabilności sieci?
31. Co to jest funkcja energii? Co mozna powidzieć o zachowaniu się funkcji energii w sieci Hopfielda?
32. Wyjaśnij dlaczego sieć Hopfielda „nie odróżnia" obrazu od jego negatywu?
33. Co nazywamy pojemnością sieci? Co można powiedzieć o pojemności sieci Hopfielda uczonej klasyczną metodą opartą o regułę Hebba.
34. Dokonaj porównania pomiędzy sieciami radialnymi, samoorganizującymi się, rekurencyjnymi i skierowanymi do przodu.
35. Omów algorytm kaskadowej korelacji Fahlmana.
36. Co to są rozszerzenia funkcyjne Pao?
37. Podaj definicję zbioru rozmytego.
38. Omów rolę poszczególnych bloków sterownika rozmytego i podaj sposób ich realizacji w technologii sieci neuronowych.
39. Znając funkcje przynależności i reguły zaprojektuj sieć neuronową pełniącą rolę sterownika rozmytego.

Algorytmy genetyczne

1. Opisz poszczególne kroki algorytmu genetycznego.
2. Zaproponuj reprezentację liczb rzeczywistych z zakresu $[-7.5, 7.5]$ możliwą do użycia w algorytmie genetycznym.
3. Na czym polega krzyżowanie/mutacja?

Zbiory rozmyte

1. Podaj definicję zbioru rozmytego.
2. Podaj przykładowe postaci funkcji przynależności.
3. Określ działania sumy, iloczynu i dopełnienia zbioru rozmytego.
4. Co nazywamy liczbą rozmytą. Określ operacje „arytmetyczne" na liczbach rozmytych.
5. Przedstaw budowę sterownika rozmytego i opisz jego bloki.
6. Opisz algorytm konstruowania bazy reguł rozmytych.
7. Co nazywamy wyostrzaniem? Jak je liczymy?

Prolog

1. Czym jest prolog? Na czym polega istota programowania w tym języku?
2. Jak należy rozumieć obiekt w sensie Prologa?
3. Co to jest relacja?
4. Podaj przykład reguły Prologa.

Inne

1. Na czy polega metoda Monte-Carlo.
2. Czym jest Boid? Przedstaw przykładowe zadnia i reguły rządzące zachowaniem Boidów.

3. Dla zadanego automatu Moore'a i sekwencji sygnałów wejściowych wygenerować ciąg sygnałów wyjściowych.
4. Co to jest drzewo decyzyjne? Wyjaśnij i jeśli to konieczne, podaj odpowiedni przykład.

Bibliografia

[1] Dreyfus S., *Richard Bellman on the birth of Dynamic Programming*, materiał dostępny pod adresem
`http://www.eng.tau.ac.il/~ami/cd/or50/1526-5463-2002-50-01-0048.pdf`
(kwiecień 2009).

[2] Driankov D., Hellendoorn H., Reinfrank M., *Wprowadzenie do sterowania rozmytego*, Wydawnictwa Naukowo-Techniczne, Warszawa 1996.

[3] Drozdek Adam, Simon Donald L., *Struktury danych w języku C++*, Wydawnictwa Naukowo-Techniczne.

[4] Duch W., *Fascynujący świat komputerów*, Poznań.

[5] William Feller, „*Wstęp do rachunku prawdopodobieństwa i jego zastosowań*", PWN Warszawa, 1960.

[6] Hertz John , Krogh Anders , Palmer Richard G., *Wstęp do teorii obliczeń neuronowych*, WNT, Warszawa 1995.

[7] Kasperski Marek Jan, *Sztuczna inteligencja. Droga do myślących maszyn*, Helion, Gliwice, 2003

[8] Kohonen Teuvo, *Self-organized formation of topologically correct feature maps*, Biological Cybernetics, 43, p. 59-69, 1982.

[9] Kohonen Teuvo, *Self-Organization and Associative Memory*, Springer, Berlin, 1984.

[10] Kohonen Teuvo, *Self-Organizing Maps. Third, extended edition*, Springer, 2001.

[11] Korbicz Józef, Obuchowicz A., Uciński D., *Sztuczne sieci neuronowe*, Akademicka Oficyna Wydawnicza PLJ, Warszawa 1994.

[12] Krose Ben, Smagt Patrick, *Introduction to Neural Networks*, Eight edition, The University of Amsterdam, November 1996.

[13] Lindsay I. Smith, *A tutorial on Principal Components Analysis*, `www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf`, February 26, 2002 (stan z 26 kwietnia 2008).

[14] Luger George f., *Artificial Intelligence. Structures and Strategies for Complex Problem Solving*, Addison Wesley, 5th ed. 2005.

[15] Mehrotra Kishan, Mohan Chilukuri K., Ranka Sanjay, *Elements of Artificial Neural Networks*, MIT Press (October 11, 1996).

[16] Nałęcz Maciej (red.), *Biocybernetyka i inżynieria biomedyczna 2000*, Akademicka Oficyna Wydawnicza EXIT, Warszawa 2000.

[17] Oja, Erkki, *Simplified neuron model as a principal component analyzer*, Journal of Mathematical Biology 15 (3), s. 267-273, November 1982.

[18] Orr Mark J. L., *Introduction to Radial Basis Function Networks*, Centre for Cognitive Science, University of Edinburg, Scotland, April 1996.

[19] Osowski Stanisław, *Sieci neuronowe w ujęciu algorytmicznym*, WNT, Warszawa 1996.

[20] Penrose Roger, *Nowy umysł cesarza. O komputerach, umyśle i prawach fizyki*, Wydawnicwo Naukowe PWN, Warszawa 2000.

[21] Peressini Anthony L., Sullivan Francis E., Uhl J. J., Jr *The Mathematics of Nonlinear Programming*, Springer-Verlag, New York, 1988.

[22] Piegat Andrzej, *Modelowanie i sterowanie rozmyte*, Akademicka Oficyna Wydawnicza EXIT, Warszawa 1999.

[23] Piliński Maciej, Rutkowska Danuta, Rutkowski Leszek, *Sieci neuronowe, algorytmy genetyczne i systemy rozmyte*, Wydawnictwo Naukowe PWN, Warszawa 1997.

[24] Rich Elaine, Knight Kevin, *Artificial Intelligence*, McGraw-Hill, Inc., 1991.

[25] Rojas Raul, *Neural Networks. A Systematic Introduction*, Springer-Verlag, Berlin, 1996 (wersja dostępna w Internecie).

[26] Rutkowski Leszek, *Metody i techniki sztucznej inteligencji*, Wydawnictwo Naukowe PWN, Warszawa, 2005.

[27] Sedgewick Robert, *Algorytmy w C++*, Wydawnictwo RM, Warszawa 1999.

[28] Shannon Claude, *Programming a Computer for Playing Chess*. Philosophical Magazine 41 (314), (1950).

[29] Wang L.-X., *Adaptive Fuzzy Systems and Control. Design and Satbility Analysis*, Prentice Hall, New Jersey, 1994.

[30] De Wilde Philippe, *Neural Network Models. An Analysis*, Springer-Verlag, London, 1996.

[31] Williams R., Zipser D., *A learning algorithm for continually running fully recurent neural networks*, Neural Computers, 1989, Vol. 1, s. 270-280.

[32] Zadeh, L. A., *Fuzzy Sets*, Information And Control, 1965, vol. 8, p. 338-353.

[33] Żurada J., Barski M., Jędruch W., *Sztuczne sieci neuronowe*, Wydawnictwo Naukowe PWN, Warszawa 1996.