Sieci Neuronowe

Marcin Buchowiecki

January 7, 2013

Spis treści

1. Modelowanie układu nerwowego
2. Elementy progowe i ich sieci.
3. Perceptron
4. Algorytm uczenia perceptronu
5. Sieci jedno - i dwuwarstwowe
6. Algorytm propagacji wstecznej
7. Sieci neuronowe i statystyka
8. Logika rozmyta
9. Samo-organizacja (sieci Kohonena)
10. Algorytmy genetyczne
11. Hardware dla sieci neuronowych

Naturalne i sztuczne sieci neuronowe

- Matematyczna inspiracja - **sformalizowanie ludzkiej myśli** (logika, teoria prawdopodobieństwa, kognitywistyka).
- Procesy przebiegające w ludzkim umyśle - bardziej skomplikowane niż jakakolwiek formalna teoria.
- Procesy poznania (kognitywistyka) modelowano naśladując fizjologię neuronów od lat 40tych (szczególnie lata 80te).
- 1943 - **Warren Mc Culloch i Walter Pitts** - pierwszy model sztucznych neuronów.
- Biologia - studiowanie neuronów, ich połączeń i mózgu.
- Dowiedziono, że sieci adaptacyjne nadają się do rozwiązywania dużych i złożonych problemów.
- 1901 – 1991: 10% nagród Nobla (Fizjologia i Medycyna) za badanie mózgu.

Naturalne i sztuczne sieci neuronowe

- **Podstawowe pytanie** - relacja sztucznych sieci neuronowych do naturalnych, które własności uwzględnić w modelach? Jaka jest relacja między naturalnymi i sztucznymi neuronami?
- Próbujemy modelować możliwości systemu nerwowego w zakresie **przetwarzania informacji**.
- Istnieje wiele modeli ale najważniejszą ideą jest **"kontrola poprzez komunikację"**; systemy nerwowe to często miliony połączonych ze sobą komórek przetwarzających przychodzące sygnały.
- **Czas reakcji na sygnał**: elektroniczne bramki logiczne - kilka nanosekund, neurony - kilkanaście milisekund; mimo to mózg potrafi rozwiązywać problemy, z którymi komputery nadal nie radzą sobie zbyt dobrze.
- Hierarchiczna, złożona i równoległa struktura mózgu wydaje się być fundamentalna dla powstania **złożonych zachowań oraz świadomości**.

Naturalne i sztuczne sieci neuronowe

- Główna różnica między sieciami neuronowymi a komputerami - **silne zrównoleglenie i nadmiarowość** (redundancja) - własności te pomagają radzić sobie z **zawodnością** poszczególnych jednostek (= neuronów).
- Biologiczne sieci - **samoorganizacja** + każdy neuron ma możliwość samoorganizacji i przetwarzania sygnałów na różne sposoby.
- **Uczenie systemu** (naturalnego lub sztucznego) - dzieki samoorganizacji przez modyfikacje parametrów w procesie uczenia (**algorytm uczący**).

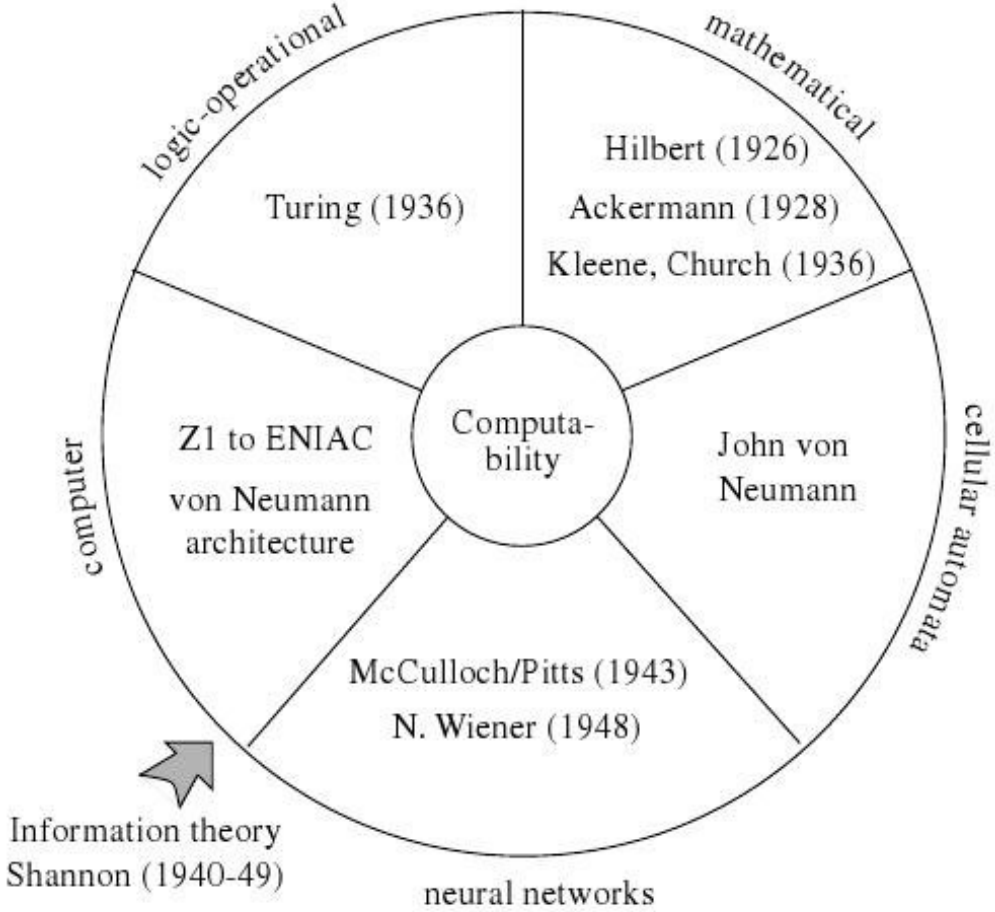
Obliczenia i obliczalność

- Sztuczne sieci neuronowe można rozważać jako kolejne podejście do problemu obliczeń.
- W naukach komputerowych - zwycięzcą został **komputer von Neumanna**.
- Matematyka nie zajmowała sie problemem obliczeń aż do początku 20go wieku.
- Nie bylo definicji **obliczalności** - pojęcie to jest względne i zależy od narzędzi jakimi dysponujemy (np. rozwiązania równania piątego stopnia nie można przedstawić wyłącznie przez funkcje algebraiczne).
- Przyjmuje się, że pewne **funkcje są podstawowe i działania** na nich definiują obliczalność.
- **David Hilbert** wysunął hipotezę, że wszystkie intuicyjnie obliczane funckcje są **prymitywnymi funkcjami rekursyjnymi** (= powstającymi złożenie, rzutowanie i określoną liczbę iteracji).
- 1927 - **Wilhelm Ackerman** znajduje funkcję obliczalną nie będącą prymitywnie rekursyjną (zawiera nieokreśloną z góry liczbę iteracji).

Obliczenia i obliczalność

- Zdefiniowno **funkcje ogólne rekursyjne** - operator μ równoważny nieokreślonej rekursji lub przeszukiwaniu nieskończonej tablicy.
- **Teza Churcha** - zbiór funkcji obliczalnych jest równy zbiorowi funkcji ogólne rekursyjnych. David Deutsch - twierdzenie to powinno mieć status zasady fizycznej.

Obliczenia i obliczalność



Five models of computation

Obliczenia i obliczalność

- **Maszyna Turinga** - operacyjny, mechaniczny model obliczalności; nieskończona taśma na której symbole mogą być zapisywane i odczytywanie; głowica odczytująco-zapisująca porusza się zgodnie ze swoim stanem wewnętrznym, który zmienia się w każdym kroku.
  - **Teza Turinga** - obliczalne funkcje to takie, które można obliczyć takim urządzeniem.
  - Turing dowiódł, że jego teza jest równoważna z tezą Churcha.
  - Podejście Turinga można nazwać **programowaniem** (komputery nie istniały w jego czasach).
- **Model komputera** - pierwsze elektroniczne maszyny liczące powstały w latach 30-40tych, **obliczalność za pomocą komputera**.
  - Pierwsi inżynierowie nie znali prac Turinga ani Churcha.
  - Maszyny **Z1 i Z3** (1938 – 1944- Berlin, Zuse) - programowalne ale nie uniwersalne - wykonywały sekwencje operacji ale nie iterowały (nie było pętli).
  - **ENIAC** - nie wykonywał nieokreślonej pętli (WHILE); iteracje - wyznaczone przez połączenia między modułami komputera.
  - **Mark I (Manchester)** - pierwszy uniwersalny komputer.

Obliczenia i obliczalność

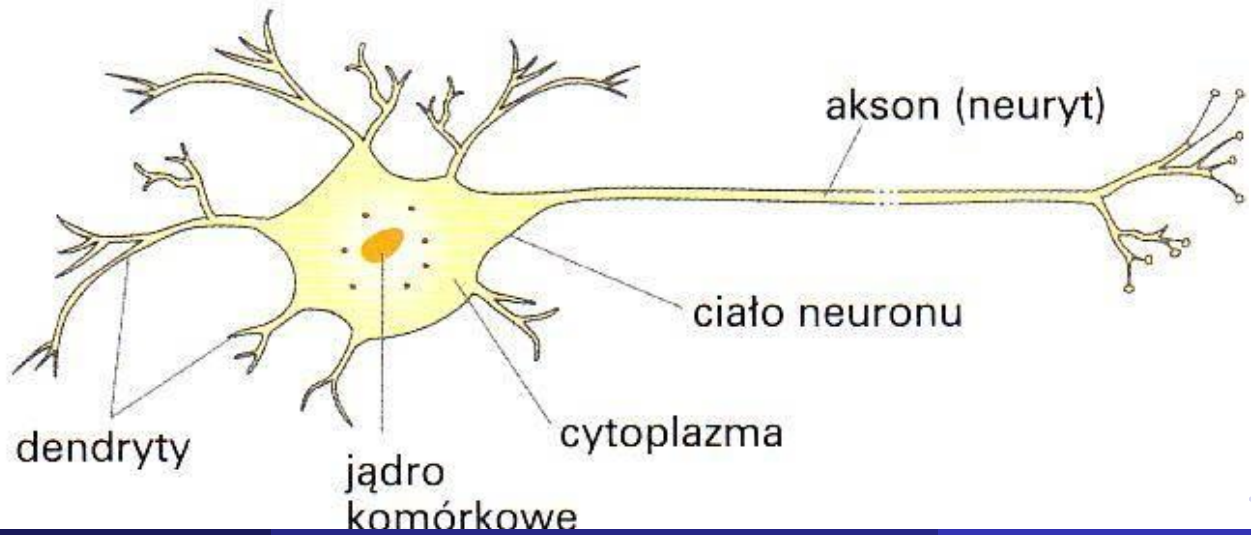
- **Automaty komórkowe** - dane mogą być przetwarzane jednocześnie; problemem jest komunikacja i koordynacja pomiędzy komórkami obliczeniowymi - gwarantują to odpowiednie algorytmy i konwencje.
  - Funkcje obliczalne w sensie Turinga mogą być obliczone przez automat komórkowy (nawet jednowymiarowy o kilku stanach).
  - Automaty k. przypominają silnie równoległe systemy wieloprocesorowe.
- **Model biologiczny = sieci neuronowe** - możliwe po wyjaśnieniu fizjologi neuronów.
  - Hierarchiczna wielowarstwowa struktura odróżnia je od automatów komórkowych (informacja przekazywana nie tylko do sąsiednich jednostek ale i bardziej odległych).
  - Od zwykłych komputerów odróżnia je brak programu; tworzy się on przez adaptacyjne ustalanie się wolnych parametrów sieci.
  - Biologiczne sieci neuronowe - miliony lat ewolucji.

Obliczenia i obliczalność

- Każdy model obliczeniowy można opisać następująco:
  obliczenie = pamięć + transmisja danych + przetwarzanie
- Przechowywanie inforamcji może być transmisją (jako obieg prądu).
- Maszyna Turinga - pamięć to taśma.
- Automaty komórkowe - każda komórka zarówno przechowuje informacje jak i jest procesorem.

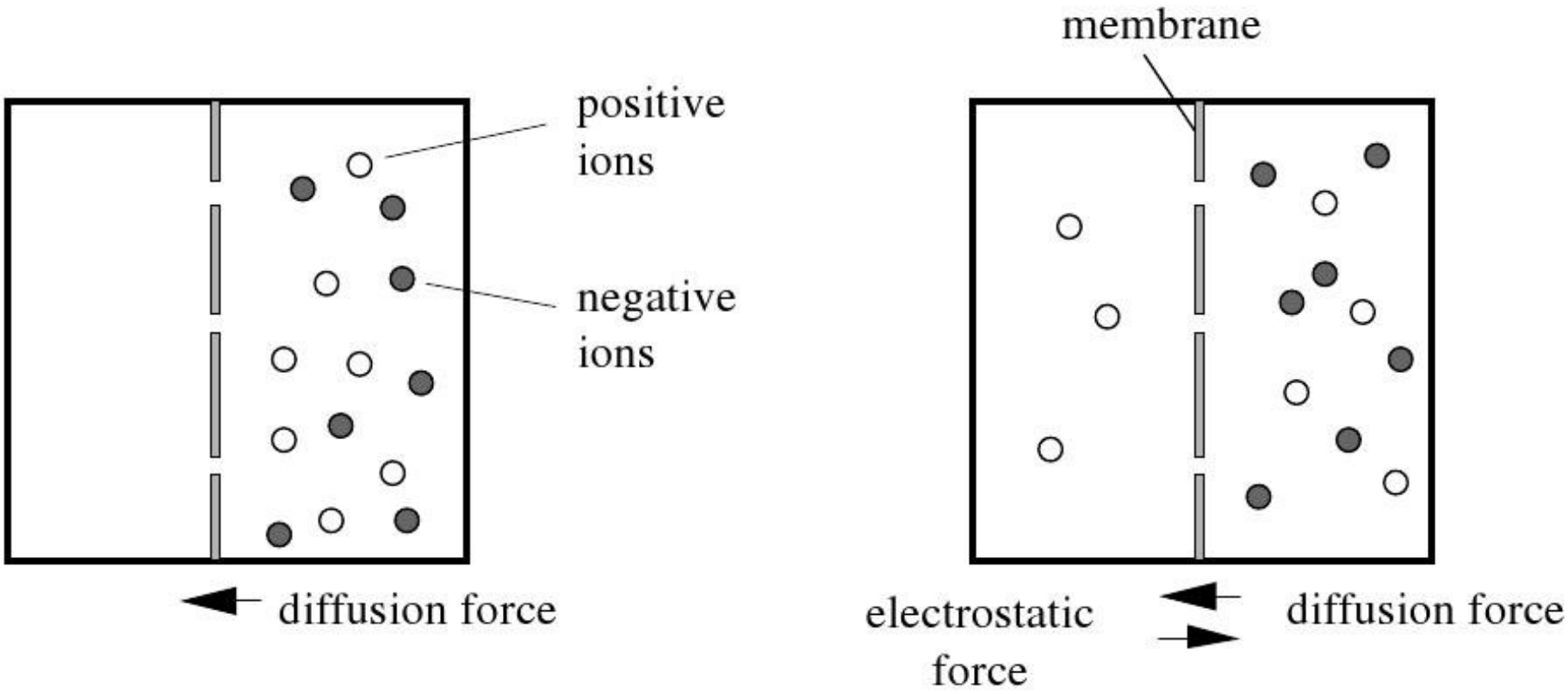
Modelowanie układu nerwowego

- **Synapsy** - przechowują informacje (w postaci przekaźników chemicznych), **dendryty** - informacja przychodząca do neuronu, **Axony** - informacja wychodząca z neuronu.
- **Minimalna struktura sztucznego neuronu** - dendryty, synapsy (z nimi są kojarzone wagi w sztucznym neuronie), ciało komórkowe i axon.
- **Transmisja informacji** - fundamentalny problem każdego systemu przetwarzania informacji (pamięć można przedstawić jako transmisję informacji).



Modelowanie układu nerwowego

- Naturalne neurony do przekazywania informacji używają sygnałów elektrycznych.



Diffusion of ions through a membrane

Modelowanie układu nerwowego

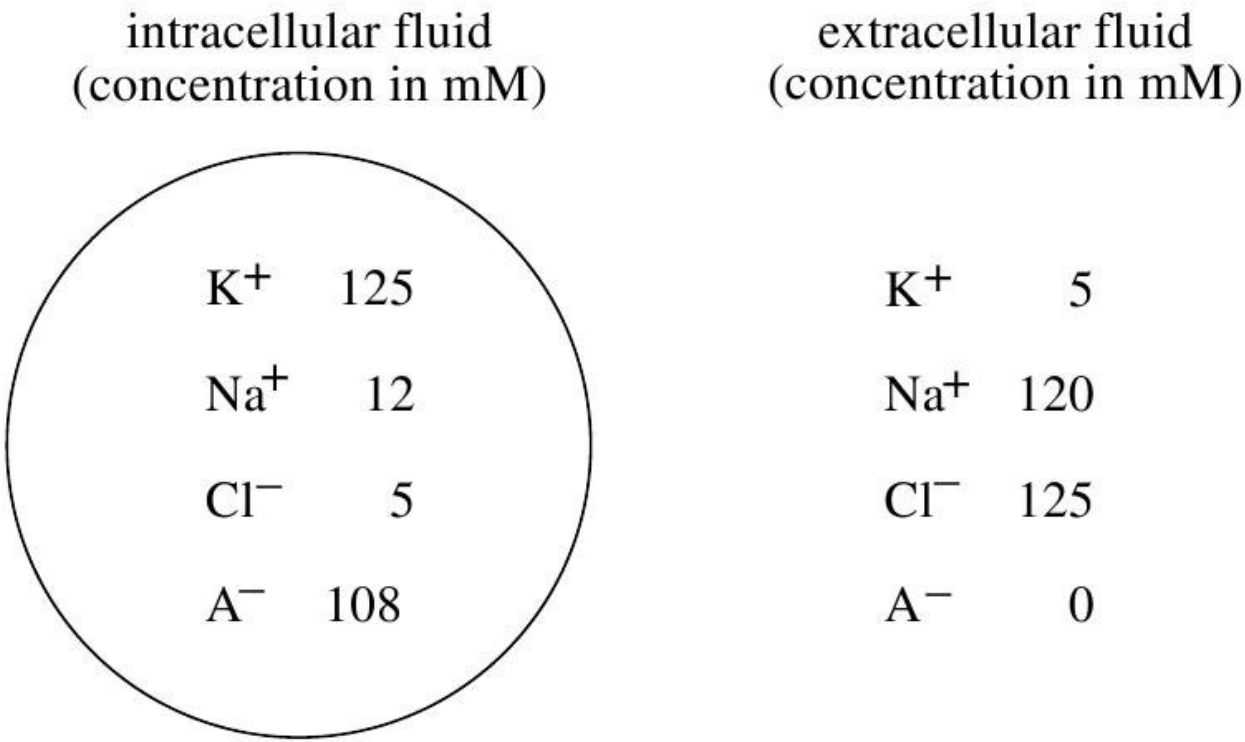
- Jony dodatnie Na^+, K^+, Ca^{2+} - przenikają przez membranę przez **kanały jonowe**.
- **Potencjał odwrotny** - różnica potencjałów w stanie równowagi.
- Wnętrze komórki naładowane jest **ujemnie**.
- Różnica potencjałów dana jest **wzorem Nernsta**

$$E = k(\ln c_0 - \ln c_i),$$

c_i - stężenie jonu i w komórce, c_0 - stężenie jonu w płynie międzykomórkowym. Np. dla K^+: $-80mV$.

Modelowanie układu nerwowego

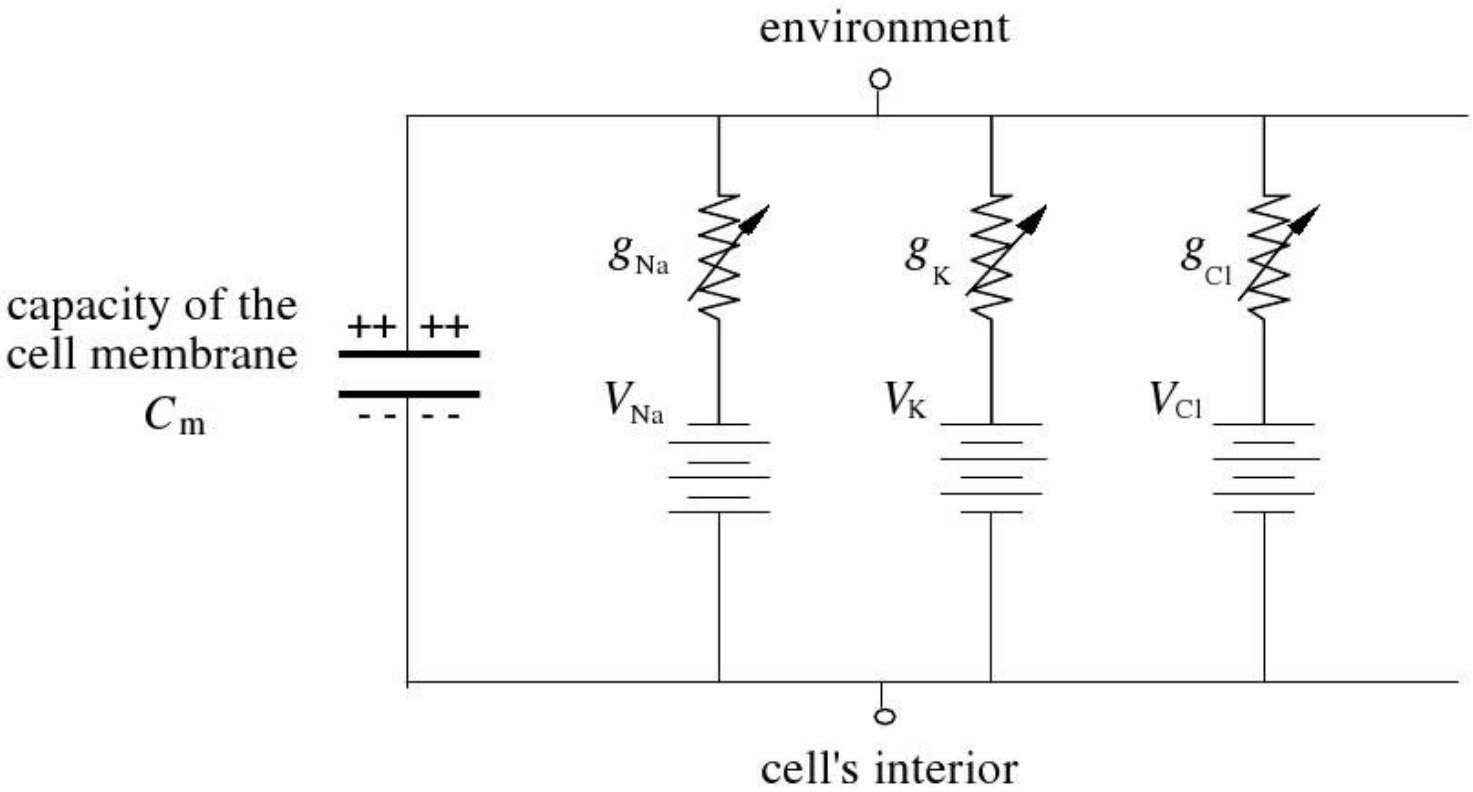
- Równowaga poniższych jonów (głównie) daje potencjał $-70mV$.



Ion concentrations inside and outside a cell

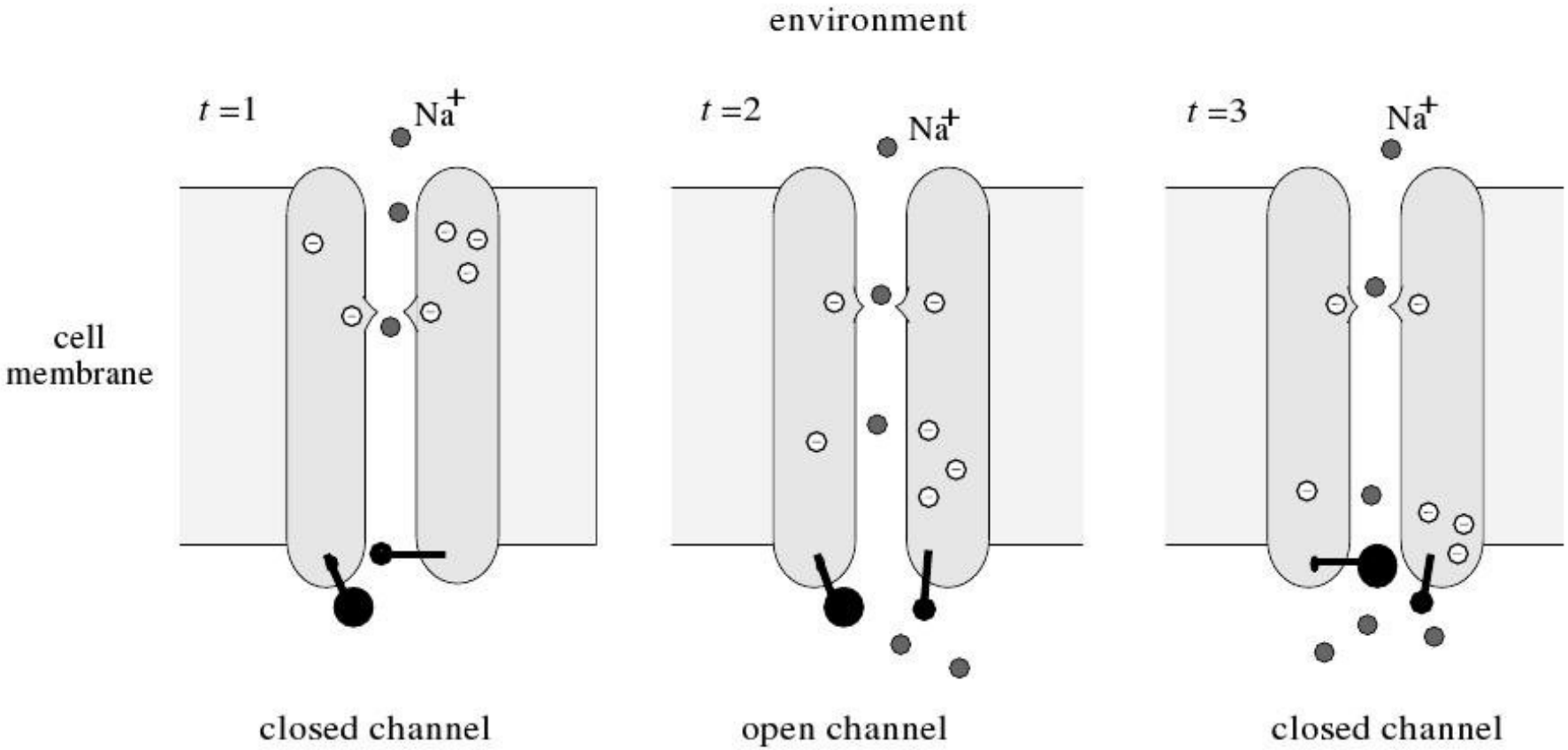
Modelowanie układu nerwowego

- **Alan Hodgkin i Andrew Huxley** - prosty model elektryczny błony komórkowej, która zachowuje się jak **kondensator** utworzony przez dwie lipidowe warstwy izolujące. **Przepuszczalność** membrany przedstawiona jest jako **przewodność** (1/opór).



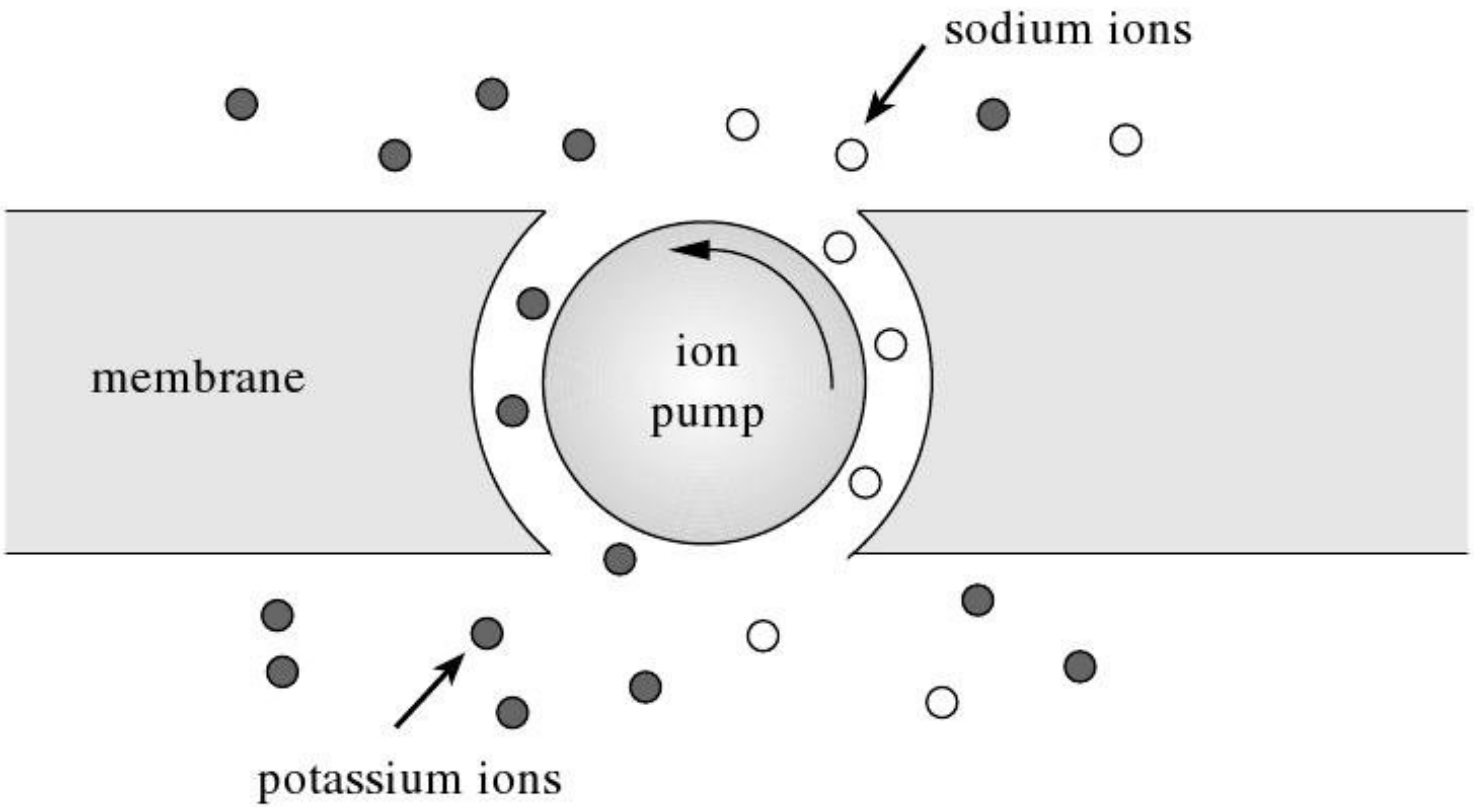
The Hodgkin–Huxley model of a cell membrane

Modelowanie układu nerwowego



Electrically controlled ionic channels

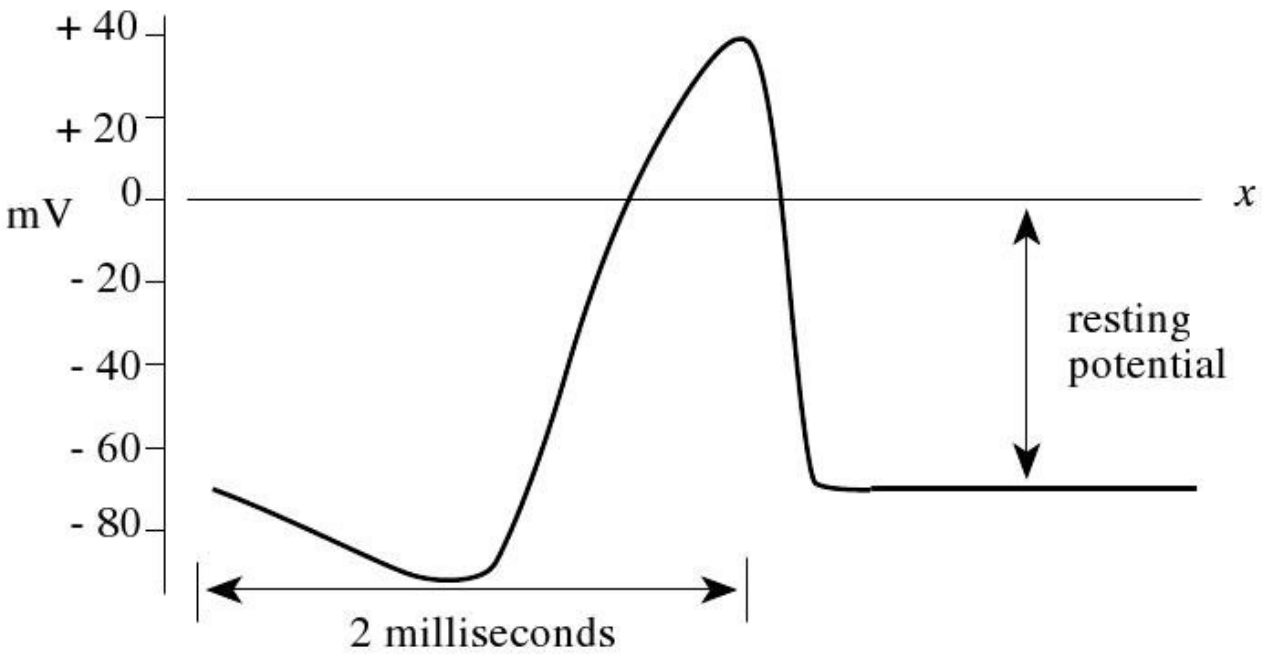
Modelowanie układu nerwowego



Sodium and potassium ion pump

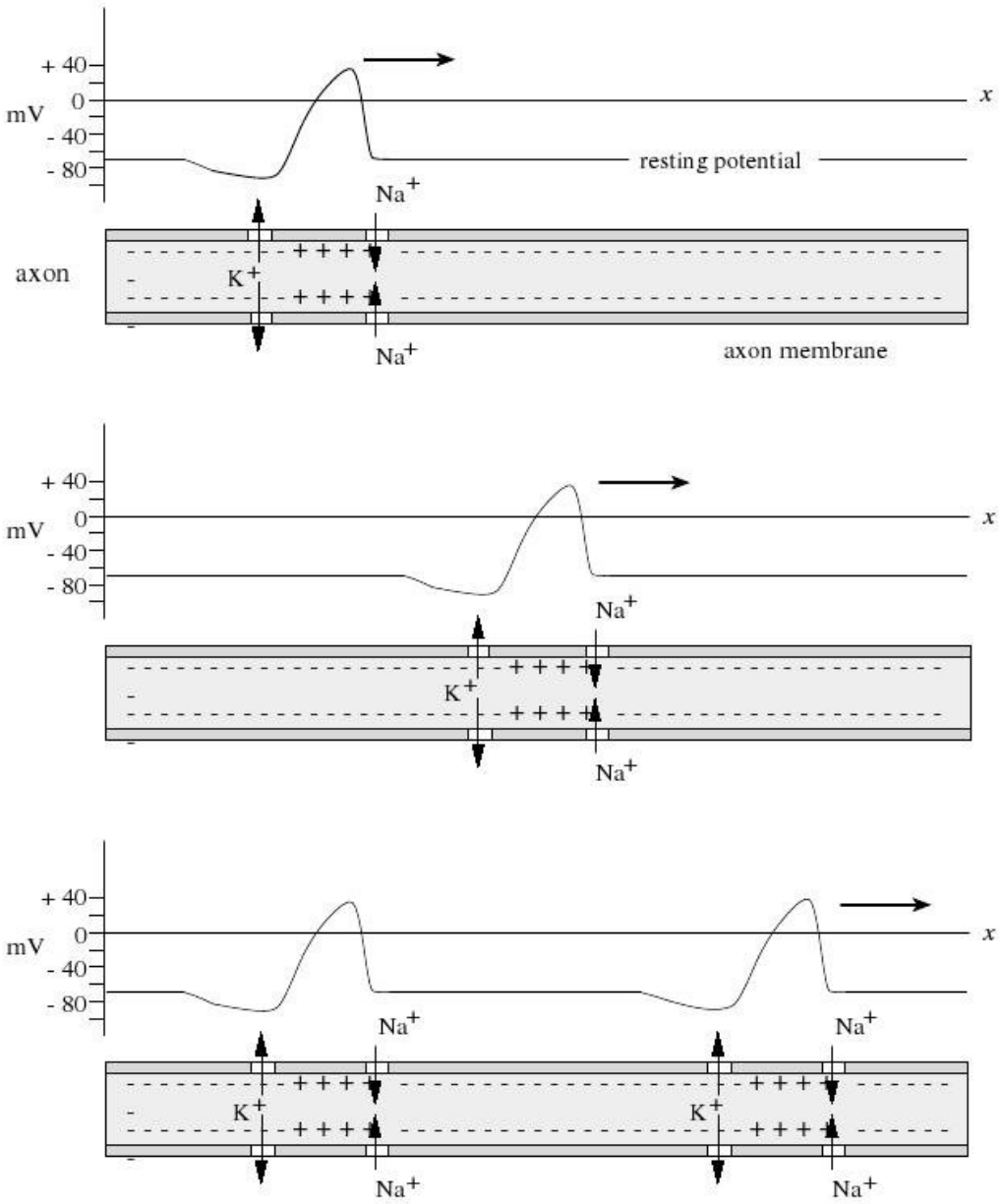
Modelowanie układu nerwowego

- **Sygnały** - fale depolaryzacji wędrujące po aksonie ($-70mV \rightarrow +40mV$). **Czas przełączania** układu opornik-kondensator dany jest stałą RC; neurony - $2.4ms$.
- Sygnał jest albo go nie ma - **cyfrowe** przekazywanie informacji.



Typical form of the action potential

Modelowanie układu nerwowego



Modelowanie układu nerwowego

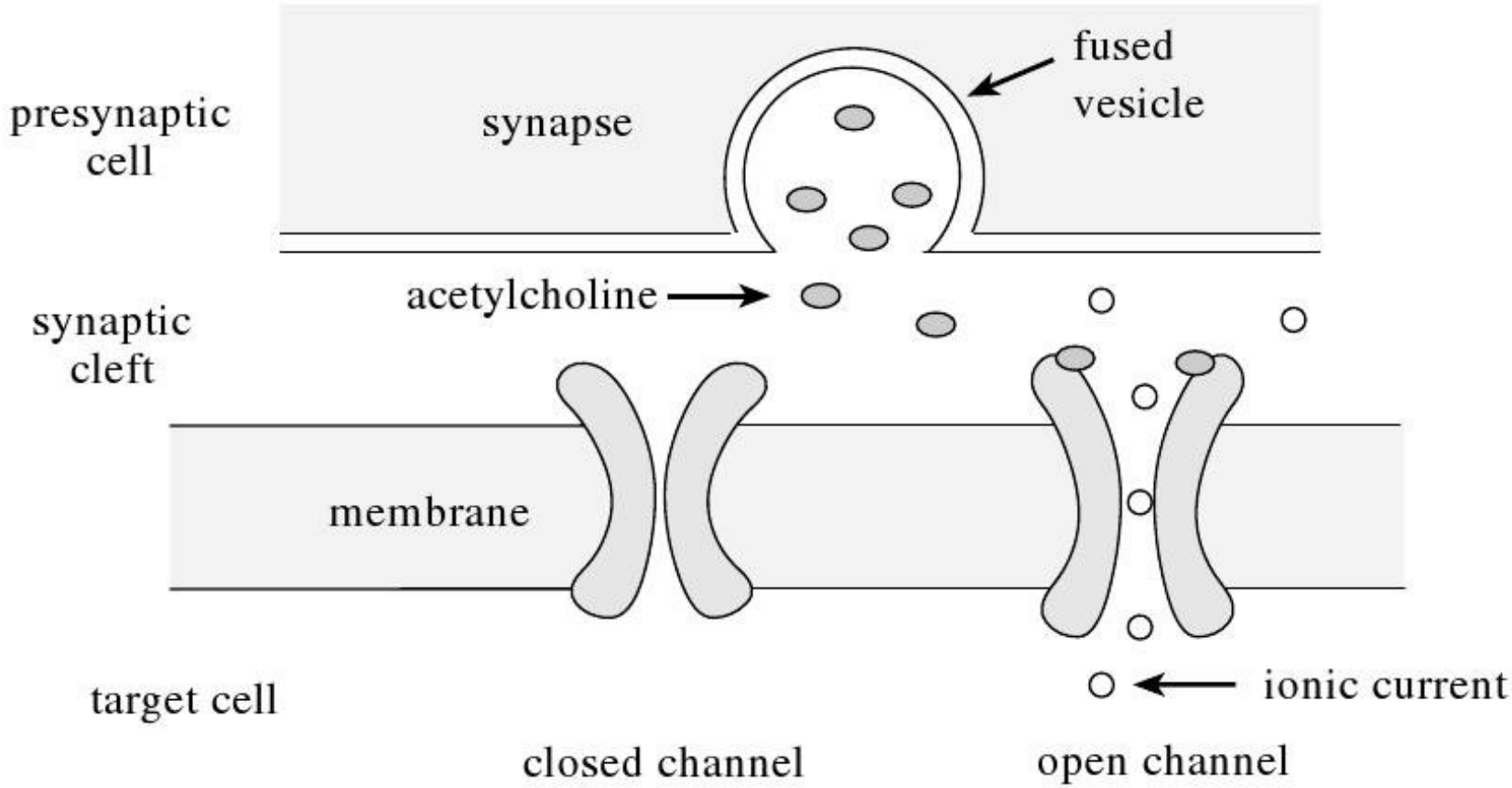
- Równanie różniczkowe Hodgkina-Huxleya

$$\frac{dV}{dt} = \frac{1}{C_m}(I - g_{Na}(V - V_{Na}) - ...),$$

  C_m - pojemność membrany, $V - V_{Na}$ - siła elektromotoryczna działająca na dany rodzaj jonów.
- Przewodności g - dane są przez r.r. opisujące ich oscylacje.
- **Silniejszy bodziec $\Rightarrow$ większa częstotliwość impulsów**.
  - Inżynierowie nazywają to **modulacją częstotliwości** np. fale radiowe FM.
  - Zwiększa to dokładność transmisji i zmniejsza zużycie energii.

Modelowanie układu nerwowego

- Przekaźnik chemiczny w synapsie - powstanie potencjału - informacja przekazywana jest dalej.



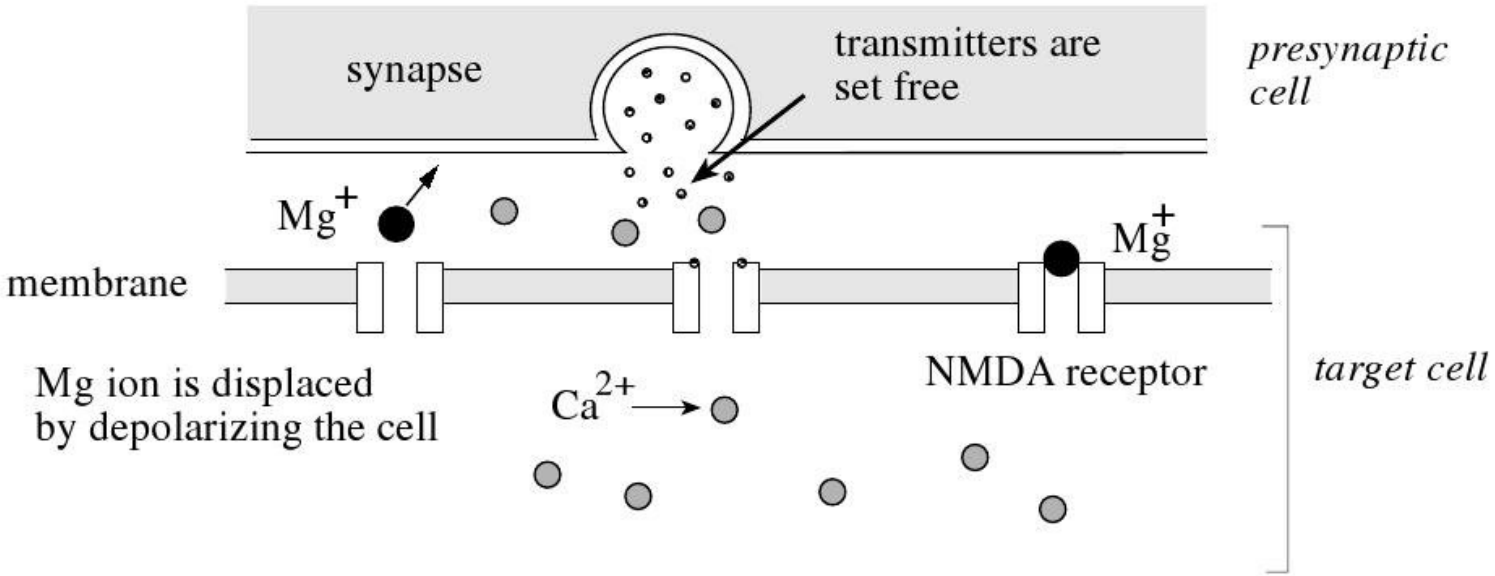
Chemical signaling at the synapse

Modelowanie układu nerwowego

- Istnieją synapsy powodujące **inhibicję** sygnału.
- Synapsa wyznacza kierunek transmisji informacji - w sztucznych sieciach opisywane to jest **grafem skierowanym** (w układach cyfrowych - diody i wzmacniacze kierunkowe).
- Potencjał za synapsą powstaje gdy kilka sygnałów dochodzi jednocześnie lub w krótkim czasie i **wartość progowa** zostaje przekroczona.
- Sygnały cyfrowe i ich złożenie na zasadzie pobudzenia-inhibicji pozwalają uzyskać dowolną **funkcję logiczną**.

Modelowanie układu nerwowego

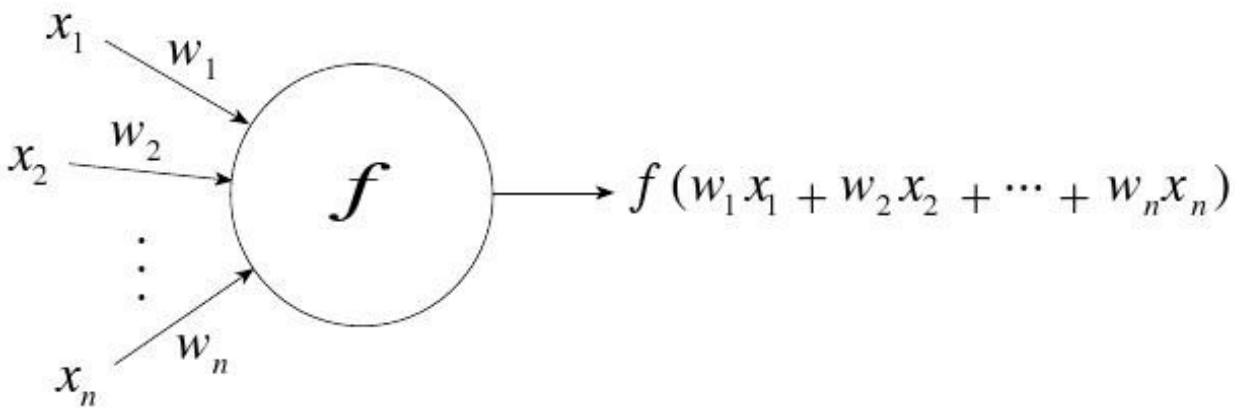
- Receptory NMDA - pozwalają zmieniać próg dla sygnałów przychodzących - proces **uczenia** sieci.
- Tą własność synaps można modelować zmianą własności połączeń jednostek sztucznej sieci.



Unblocking of an NMDA receptor

Modelowanie układu nerwowego - sztuczne sieci neuronowe

- Komputer - potrzebny minimalny zbiór instrukcji dla implementacji wszystkich funkcji obliczalnych.
- Sztuczna sieć neuronowa - funkcje elementarne zlokalizowane są w węzłach sieci a reguły ich składania są zawarte w połączeniach między nimi, synchroniczności lub asynchroniczności transmisji oraz obecności lub braku cykli.
- **Abstrakcyjny neuron** o n wejściach obliczający **funkcję elementarną** f dla danych **wartości na wejścu x** i **wag w**:



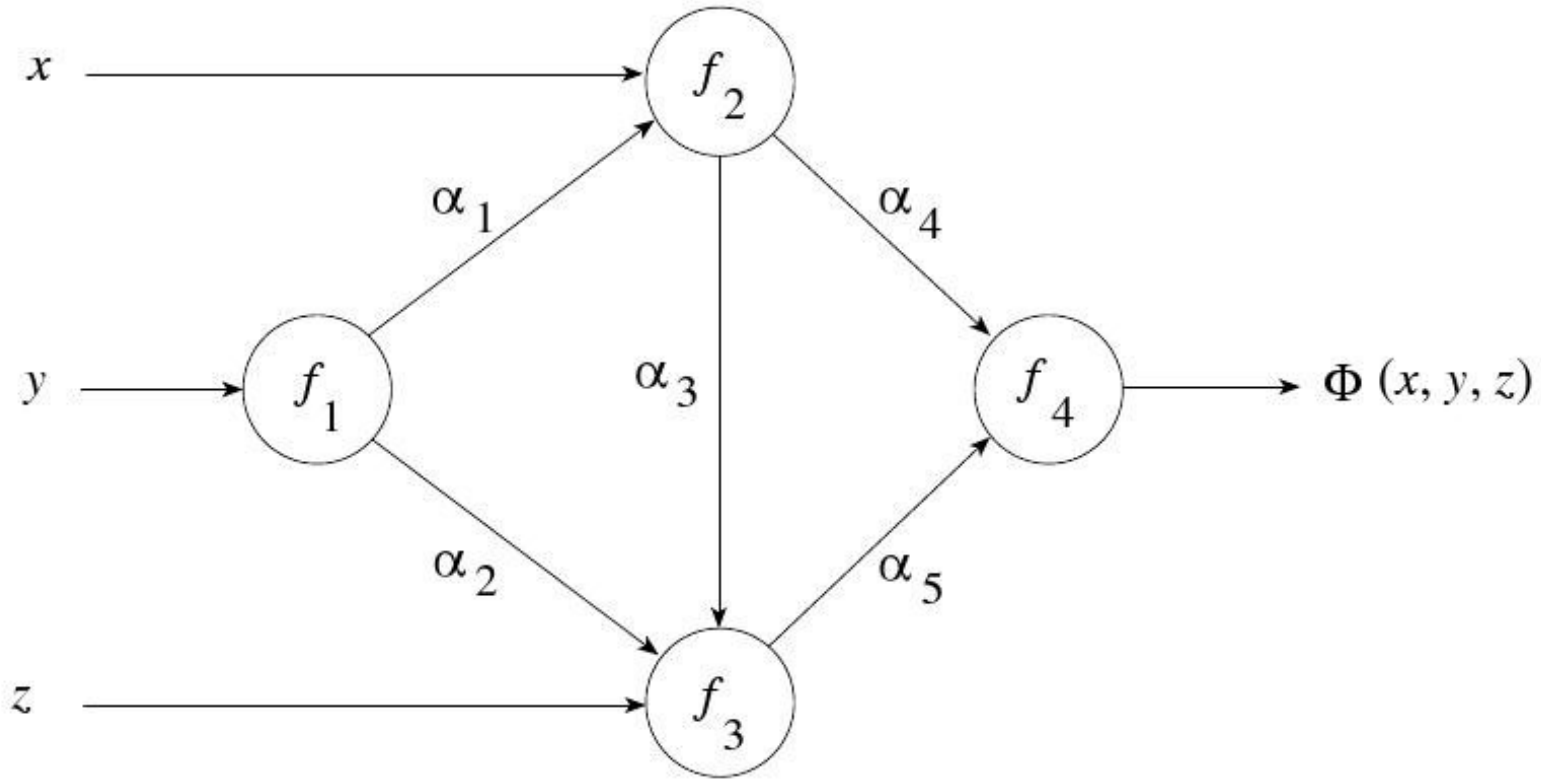
An abstract neuron

Modelowanie układu nerwowego - sztuczne sieci neuronowe

- Informacje ze wszystkich wejść są najczęściej scalane przez **sumowanie**.
- **Różne modele** sztucznych sieci neuronowych różnią się głównie:
  - założeniami o funkcjach elementarnych,
  - wzorcem połączeń,
  - czasem kiedy informacja jest transmitowana.

Modelowanie układu nerwowego - sztuczne sieci neuronowe

- O sieci można zatem myśleć jak o obliczaniu wartości funkcji Φ tzw. **funkcji sieci**.



Functional model of an artificial neural network

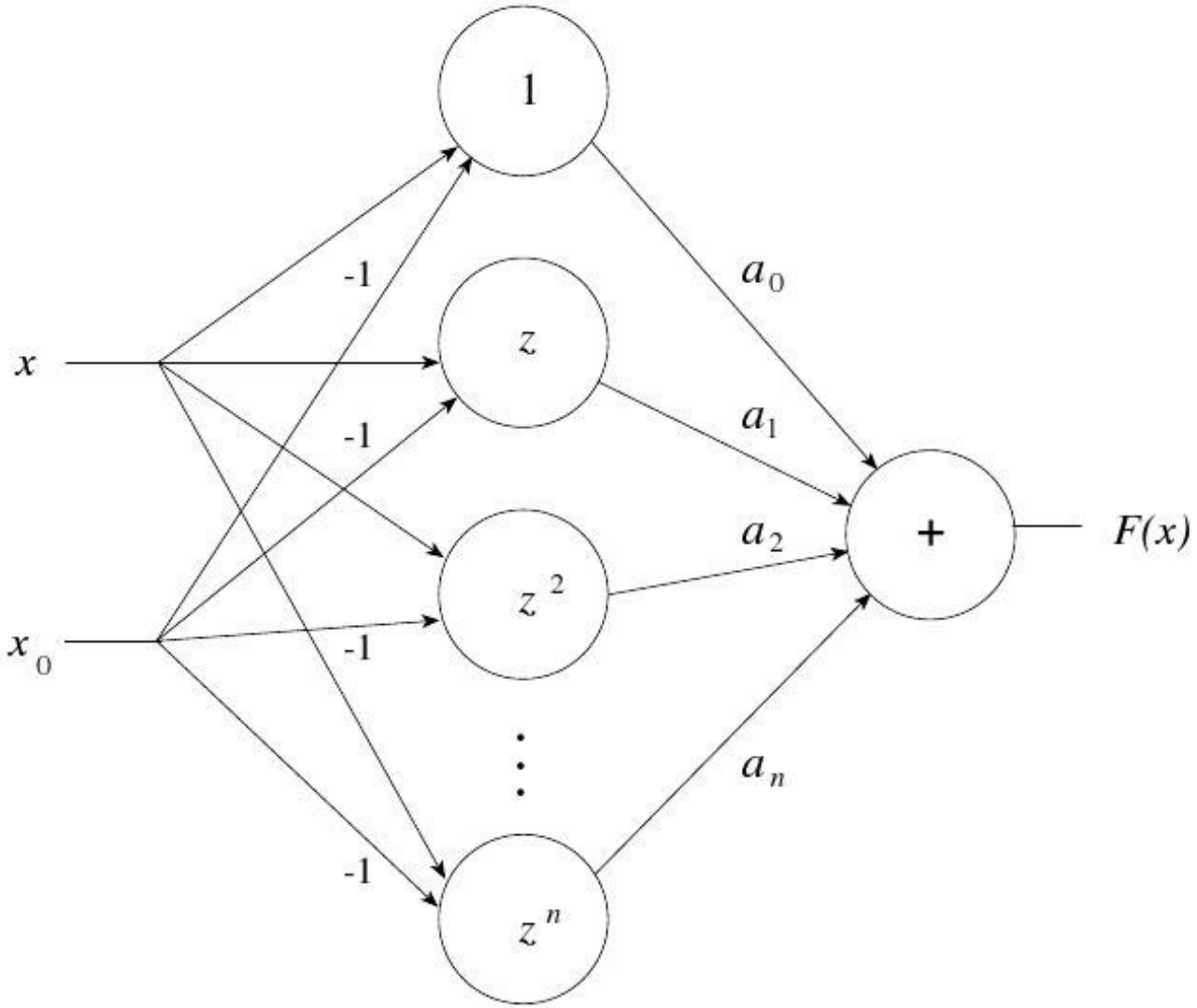
- Dobór wag α zmienia funkcję Φ.
- **Ważne parametry**:
  - struktura węzła,
  - topologia sieci,
  - algorytm uczenia (sposób dobierania wag sieci).
- **Aproksymacja funkcji** - otworzyć funkcję dokładnie lub w przybliżeniu przez obliczenie pewnego zbioru funkcji elementarnych np. przybliżanie wielomianami lub szeregiem Fouriera.

$$F(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + ... + a_n(x - x_0)^n + ...,$$

$$F(x) = \sum_{i=0}^{\infty}(a_i \cos(ix) + b_i \sin(ix)).$$

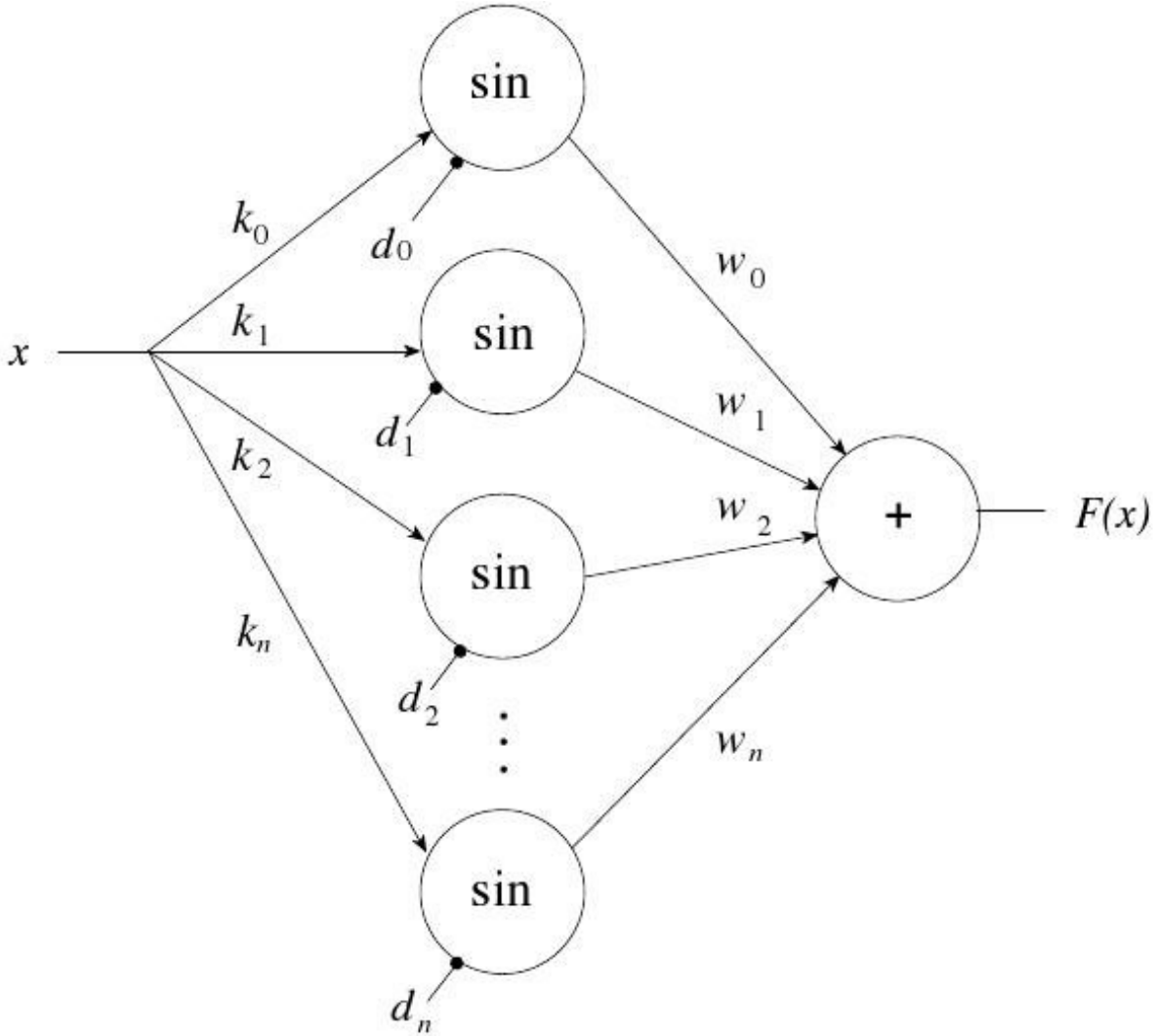
- Parametry można obliczyć analitycznie lub za pomocą algorytmu uczenia.

Modelowanie układu nerwowego - sztuczne sieci neuronowe



A Taylor network

Modelowanie układu nerwowego - sztuczne sieci neuronowe



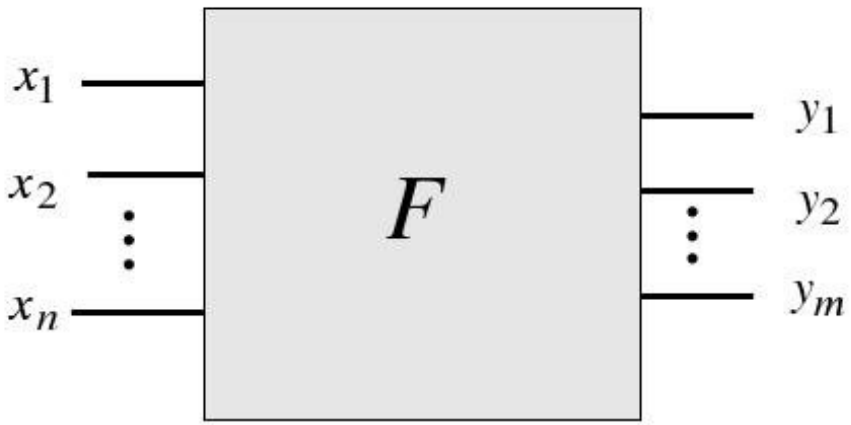
A Fourier network

Modelowanie układu nerwowego - sztuczne sieci neuronowe

- Sztuczne sieci neuronowe - funkcja F dana jest przez zbiór przykładów wejście-wyjście, które w procesie uczenia mają zostać uogólnione do funkcji F.

Elementy progowe i ich sieci

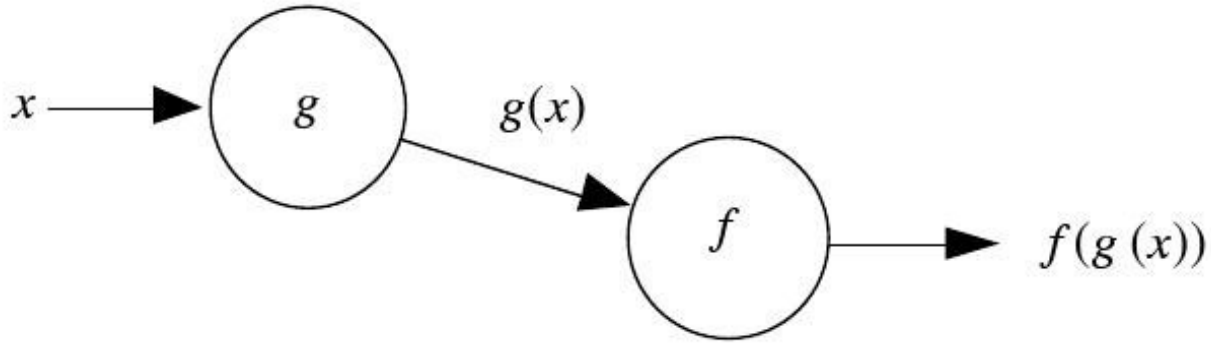
- **Jednostki obliczeniowe** sieci neuronowych są uogólnieniem bramek logicznych.
- Najprostszy rodzaj porównuje wejście z wartością progową - stąd nazwa **logika progowa**.
- Sieć neuronową można często traktować jako **czarną skrzynkę**; rezultat działania sieci pozostawia się samoorganizującemu procesowi.
- Poniższa sieć oblicza funkcję $F : \mathbb{R}^n \to \mathbb{R}^m$.



A neural network as a black box

Elementy progowe i ich sieci

- Poniższy graf skierowany obrazuje **złożenie funkcji**:

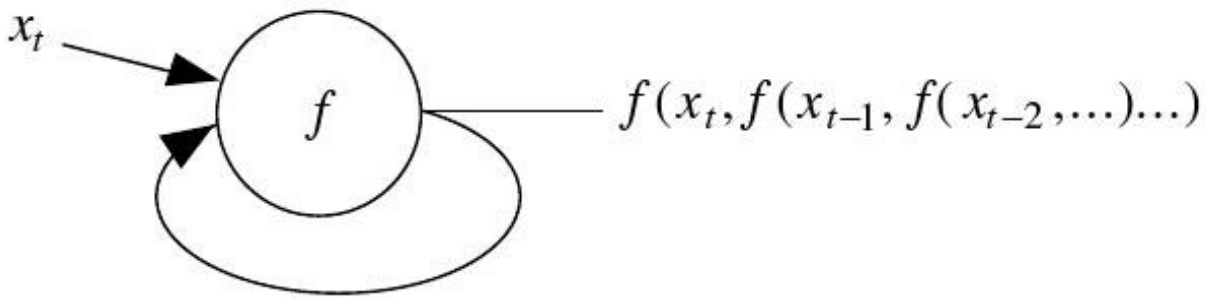


Function composition

- Gdy nie ma cykli rezultat działania sieci jest dobrze określony i nie występuje problem synchronizacji jednostek; zakłada się, że obliczenia następują **bez opóźnienia**.

Elementy progowe i ich sieci

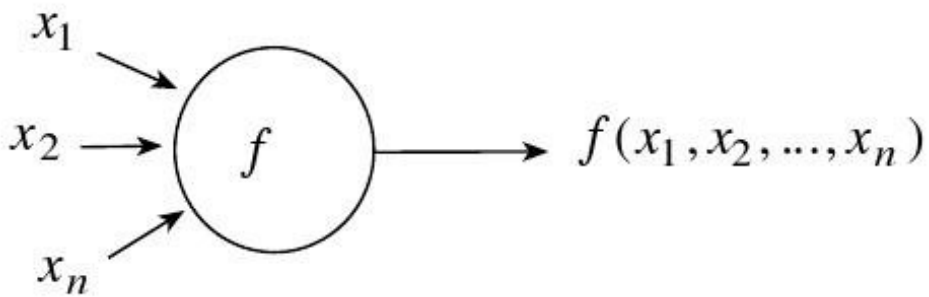
- Gdy sieć zawiera **cykle** zdefiniowanie połączeń nie wystarcza.
- Jeżeli sprzeżenie zwrotne następuje do tej samej jednostki nie ma oczywistego warunku na **zatrzymanie** działania sieci; pytanie: który obrót cyklu daje żądaną wartość.
- Założenie - **każde obliczenie zabiera jednostkę czasu**.
  - Argument dostarczony do jednostki w chwili t daje rezultat na wyjściu w chwili $t + 1$.
  - Obliczenie można zatrzymać po pewnej liczbie kroków.



Recursive evaluation

Elementy progowe i ich sieci

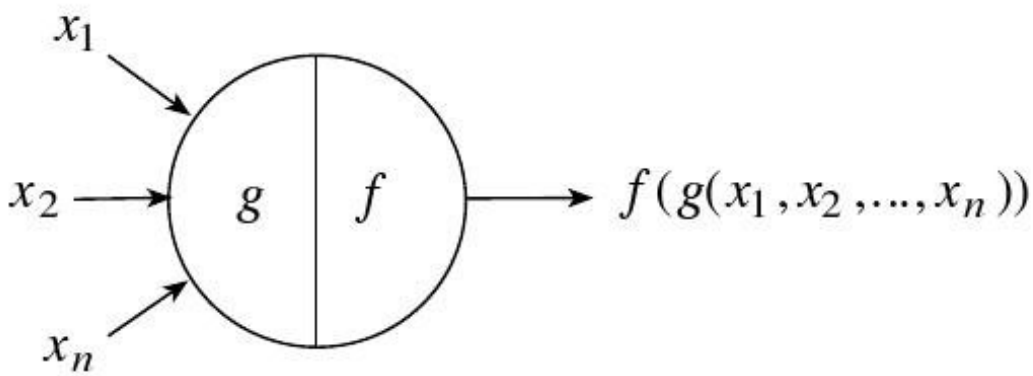
- **Definicja. Sieć neuronowa** to sieć funkcji dla której synchronizację można rozważać jawnie lub nie.
- Elementy sieci nazywamy **jednostkami** (obliczeniowymi) lub (sztucznymi) **neuronami**.
- Zakładamy też, że połączenia sieci przekazują informację w **określonym z góry kierunku** oraz że **liczba wejść** do węzła sieci **nie jest ograniczona**.



Evaluation of a function of n arguments

Elementy progowe i ich sieci

- Jednostki dzielimy ponadto na dwie funkcjonalne części:
  1. Redukującą wejścia do jednej liczby g.
  2. **Funkcję wyjścia (aktywacji)** f, która jest wtedy prostą funkcją jednej zmiennej.



Generic computing unit

Elementy progowe i ich sieci

- **Sieci McCullocha-Pittsa** są jeszcze prostsze - posługują się tylko sygnałami binarnymi (**0** i **1**).
- Połączenia w tych sieciach nie posiadają wag i mogą być typu pobudzającego lub hamującego (inhibicyjnego; **oznaczenie - małe kółko** na końcu połączenia).
- Wejścia dochodzą do białej części węzła, wyjścia do czarnej i mogą się rozgałęziać.

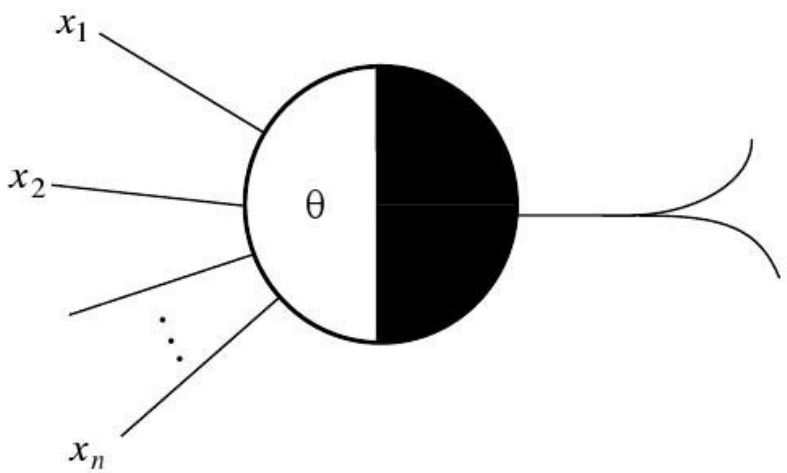


Diagram of a McCulloch–Pitts unit

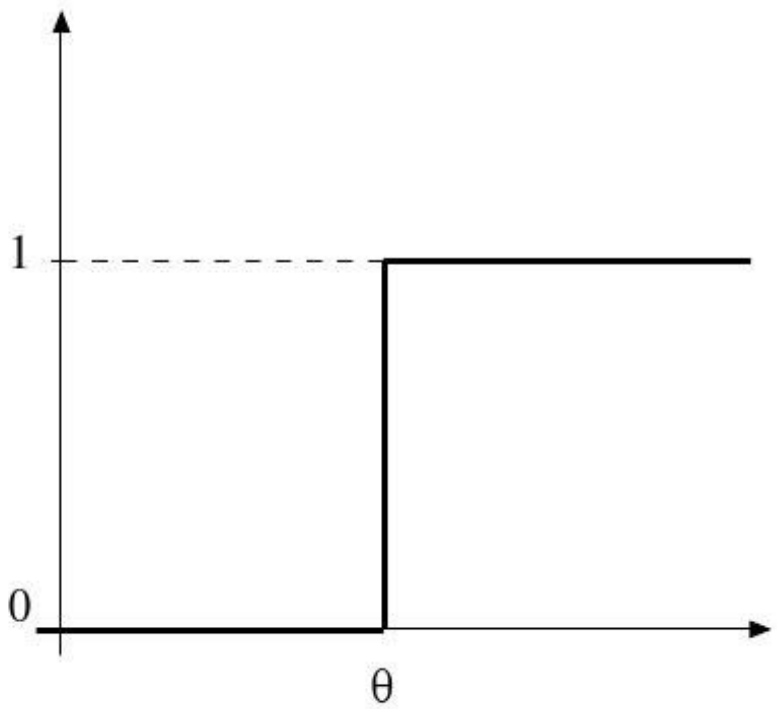
- **Zasada działania jednostki McCullocha-Pittsa**:
  1. Wejście - n liczb $x_1, ..., x_n$ przychodzących przez połączenia pobudzające oraz m liczb $y_1, ..., y_m$ poprzez połączenia hamujące.
  2. $m \geq 1$ i $\exists_i y_i = 1 \Rightarrow$ wynik obliczenia jest 0.
  3. W przeciwnym wypadku obliczane jest całkowite wzbudzenie $x = x_1 + ... + x_n$ i następnie porównywane z progiem jednostki θ (jeżeli $n = 0$ to $x = 0$):

$$x \geq \theta \Rightarrow 1,$$

$$x < \theta \Rightarrow 0.$$

Elementy progowe i ich sieci

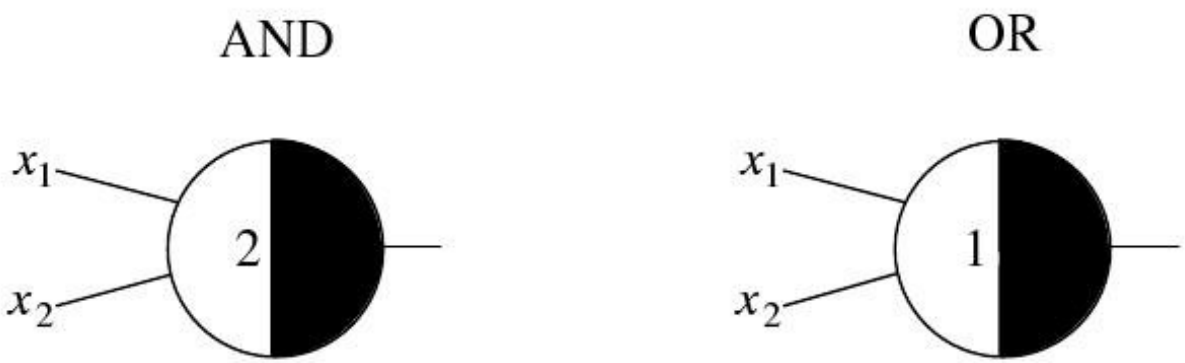
- **Funkcja schodkowa** jest funkcją aktywacji takiej jednostki (w przypadku nieobecności sygnału) hamującego.



The step function with threshold θ

Elementy progowe i ich sieci - funkcje Boole'a

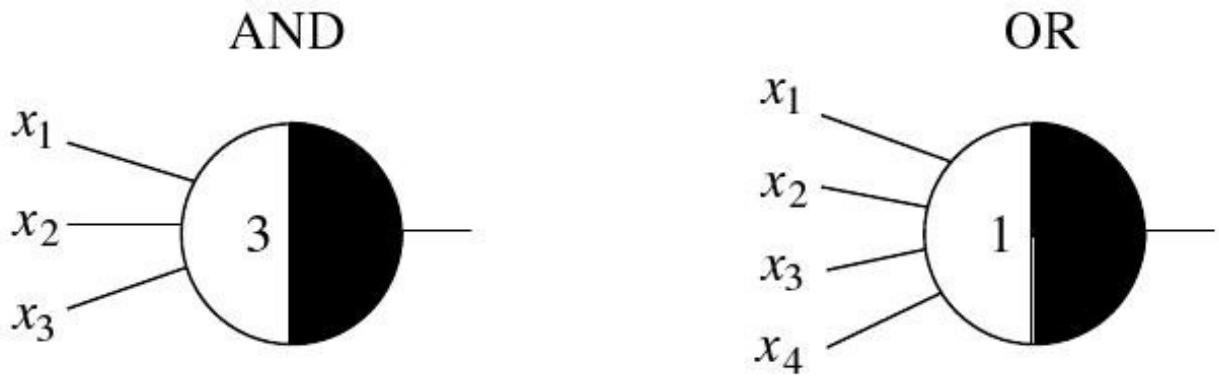
- **Funkcje logiczne (Boole'a)** - odwzorowania $\{0,1\}^n \to \{0,1\}$.
- Proste funkcje logiczne można przedstawić jako jedną jednostkę McCullocha-Pittsa, gdzie wartość 1 oznacza *prawdę* a wartość 0 *fałsz*.



Implementation of AND and OR gates

Elementy progowe i ich sieci - funkcje Boole'a

- Uogólnienie do przypadku n agrumentów (wymagało by to kilku klasycznych bramek logicznych o dwóch wejściach).
- **Wniosek** - progowe elementy logiczne mogą **zredukować złożoność** układu.



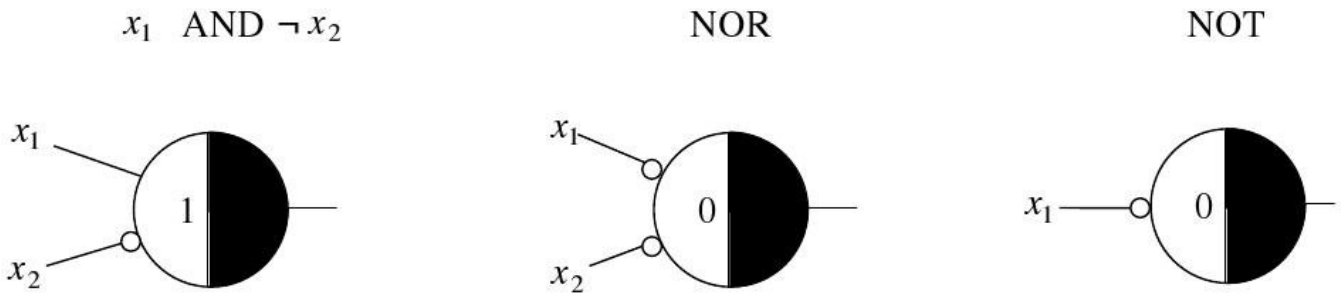
Generalized AND and OR gates

Elementy progowe i ich sieci - funkcje Boole'a

- Wiadomo że przez złożenie bramek *AND* i *OR* **nie** można otrzymać dowolnej funkcji logicznej n argumentów.
- Progowe elementy logiczne są ogólniejsze, powstaje **pytanie** czy ich złożenie daje dowolną funkcję logiczna?
- Odpowiedź to **nie**, możliwe jest to tylko dla monotoniczych funkcji logicznych.
- **Definicja**. Funkcję logiczną n agrumentów nazywamy **monotoniczną** jeżeli dla dwóch punktów $(x = (x_1, ..., x_n)$ i $y = (y_1, ..., y_n))$ $f(x) \geq f(y)$ zawsze gdy zbiór jedynek we współrzędnych punktu y jest pozbiorem jedynek w x.
- Przykład niemonotonicznej funkcji logicznej jednej zmiennej - **negacja** $(\{0\}_1 \subset \{1\}_1 \Rightarrow 1 = f(0) < f(1) = 0)$.

Elementy progowe i ich sieci - funkcje Boole'a

- **Twierdzenie**. Progowe elementy logiczne bez inhibicji typu McCullocha-Pittsa mogą obliczać jedynie funkcje monotoniczne.
- **Dowód**.
  1. Załóżmy, że $(1, 1, ..., 1) \rightarrow 0$,
  2. Dowolny wektor jest odwzorowywany na 0 (nie można większą ilością połączeń przesyłać wartości 1).
  3. Ogólnie: jeżeli $\{y\}_1 \subset \{x\}_1 \Rightarrow f(x) \geq f(y)$ (rysunek).
- Przykłady funkcji niemonotonicznych - do realizacji wymagają połączeń inhibujących.

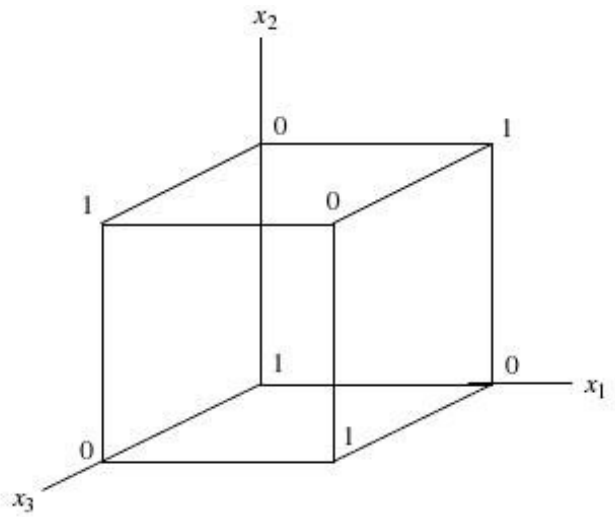


Logical functions and their realization

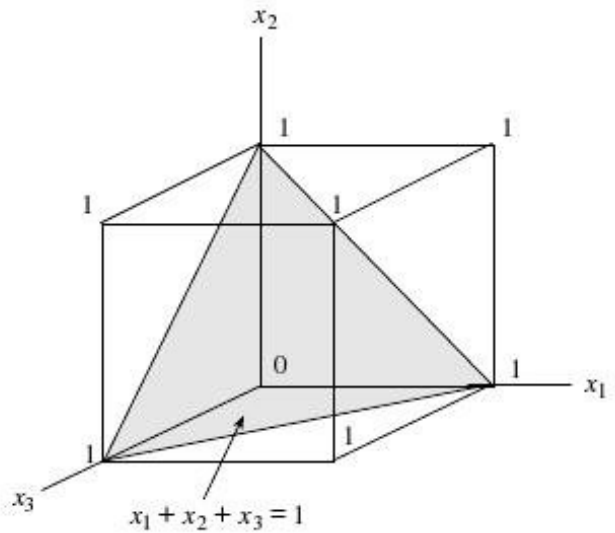
Elementy progowe i ich sieci - funkcje Boole'a

- **Wizualizacja** funkcji obliczalnych jednostkami McCullocha-Pittsa.
  - każda zmienna logiczna przedstawiana jest na osi,
  - 8 możliwych kombinacji dla ich wartości - wierzchołki sześcianu.
- **Funkcja logiczna** to przypisanie wartości do 0 lub 1 do wierzchołków.
- Jednostka McCullocha-Pittsa dzieli przestrzeń na **dwie pół-przestrzenie** ($x_1 + x_2 + x_3 \geq \theta$ oraz $x_1 + x_2 + x_3 < \theta$) płaszczyzną $x_1 + x_2 + x_3 = \theta$.
- **Przykład**: $\theta = 1$, czyli funkcja *OR*.

Elementy progowe i ich sieci - funkcje Boole'a



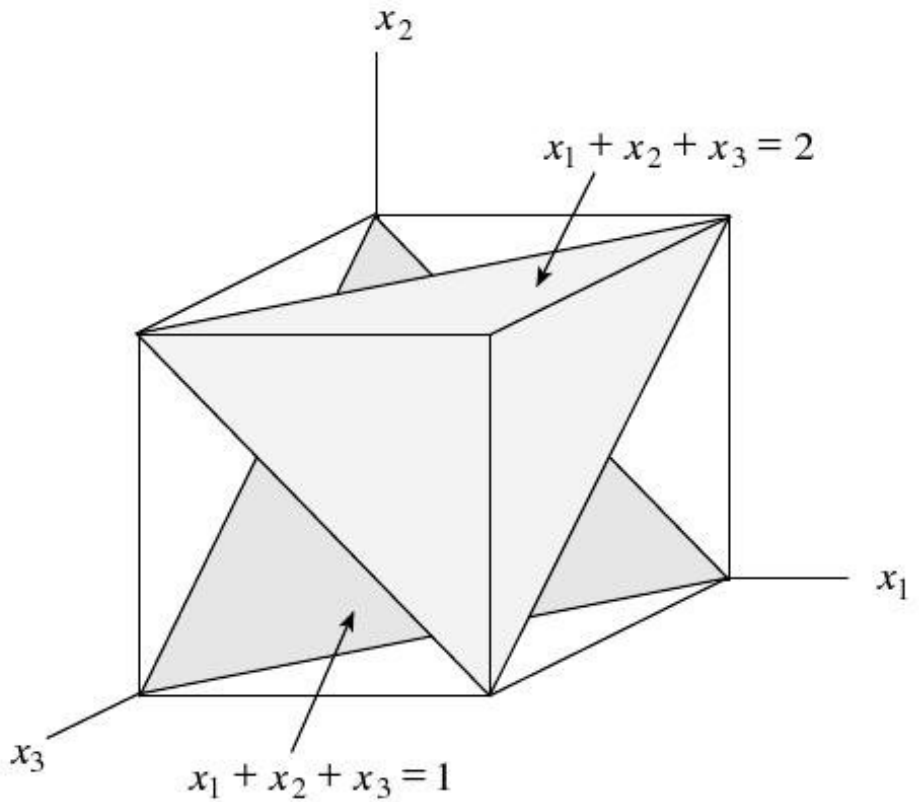
Function values of a logical function of three variables



Separation of the input space for the OR function

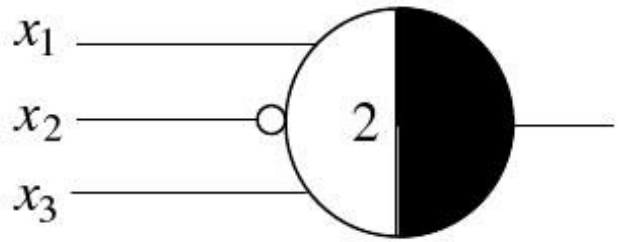
Elementy progowe i ich sieci - funkcje Boole'a

- Funkcja **większości** - **majority function** (fałsz jeżeli $n/2$ lub więcej argumentów jest fałszywe, w przeciwnym wypadku - prawda) odpowiada $\theta = 2$.
- Płaszczyzny są zawsze **równoległe** w przypadku jednostek McCullocha-Pittsa.



Elementy progowe i ich sieci - funkcje Boole'a

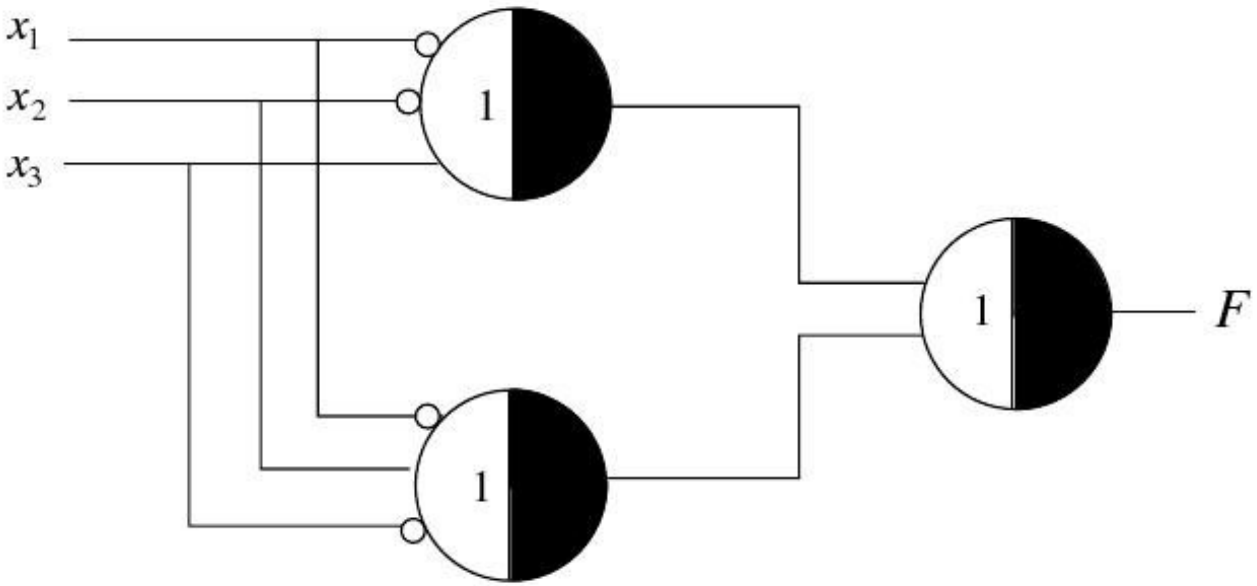
- Każda funkcja logiczna n zmiennych może być zapisana w postaci **tabeli**.
- Sieć do obliczania tej funkcji - n wejść i jedno wyjście; jeżeli nie ograniczamy liczby jednostek **zawsze** można zbudować sieć obliczającą zadaną funkcję.
  - Np. tylko wektor $(1, 0, 1)$ spełnia warunek $x_1 \wedge \neg x_2 \wedge x_3$.
  - Poniższa jednostka daje 1 tylko dla $(1, 0, 1)$ - jest **dekoderem** tego wektora (**próg** = liczba jedynek).
  - Niech **F** będzie dana tabelą - do obliczenia tej funkcji wystarczy zdekodować wektory dające wartość 1.



Decoder for the vector $(1, 0, 1)$

Elementy progowe i ich sieci - funkcje Boole'a

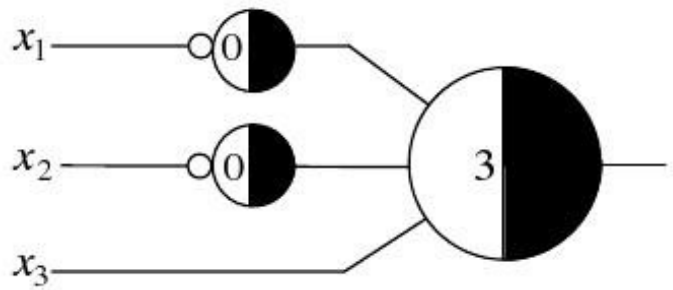
input vectors	F
(0,0,1)	1
(0,1,0)	1
all others	0



Synthesis of the function F

Elementy progowe i ich sieci - funkcje Boole'a

- **Wniosek.** Każdą funkcję logiczną $F : \{0,1\}^n \to \{0,1\}$ można obliczyć dwuwarstwową siecią jednostek McCullocha-Pittsa.
- Potrzeba tylu dekoderów ile jest jedynek w tabeli. Alternatywą dla tej prostej konstrukcji jest **analiza harmoniczna funkcji logicznych**.
- Poniższa jednostka jest alternatywnym dekoderem zbudowanym z bramek *AND* i *NOT*.



A composite decoder for the vector $(0,0,1)$

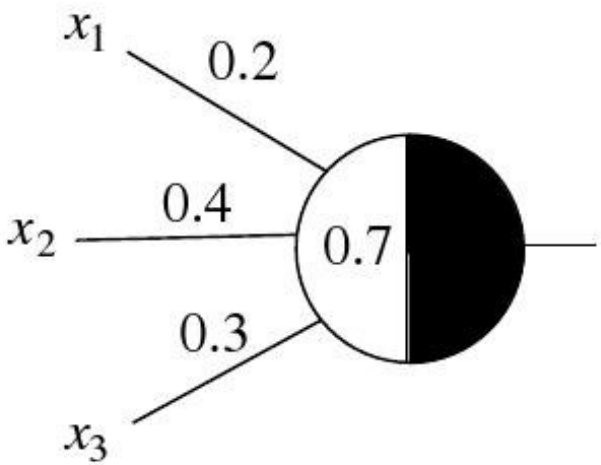
Elementy progowe i ich sieci

- **Twierdzenie.** Wszystkie funkcje logiczne mogą być obliczone siecią jednostek *AND*, *OR* i *NOT*.
- *AND*, *OR* i *NOT* nazywamy **bazą logiczną**.
- *OR* można przedstawić jako złożenie *AND* oraz *NOT*. Te dwie bramki także tworzą bazę logiczną.
- Podobnie jest z *OR* i *NOT*.
- John von Neumann pokazał, że dzięki kodowaniu nadmiarowemu (zmienna przekazywana dwoma liniami) *AND* i *OR* tworzą bazę logiczną.

Elementy progowe i ich sieci - sieci równoważne

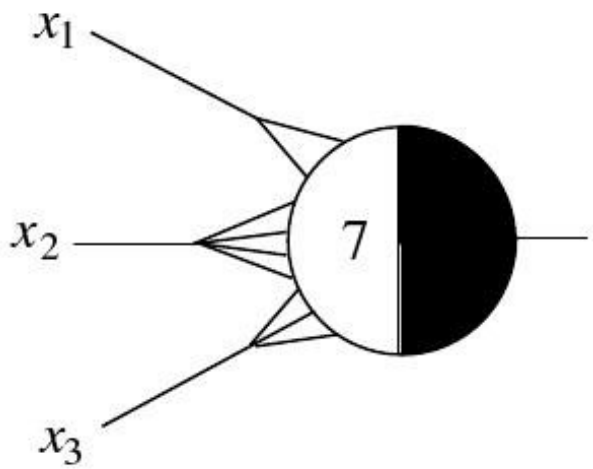
- **Prostsze sieci** - wymagają ogólniejszych jednostek np. z wejściami z wagą lub inhibicji względnej.
- Jednostki McCullocha-Pittsa nie używają wag; **pytanie** - czy sieci z wagami są ogólniejsze niż bez wag.
- Odpowiedź to nie - poniższe jednostki są równoważne:

$$0.2x_1 + 0.4x_2 + 0.3x_3 \geq 0.7 \quad \Leftrightarrow \quad 2x_1 + 4x_2 + 3x_3 \geq 7.$$



Weighted unit

Elementy progowe i ich sieci - sieci równoważne



Equivalent computing unit

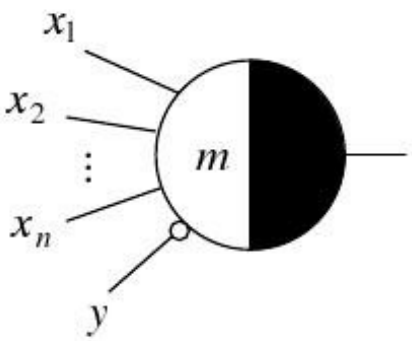
- Przykład ten pokazuje, że **wagi wymierne dodatnie** można symulować zwielokrotniając wejścia.

Elementy progowe i ich sieci - sieci równoważne

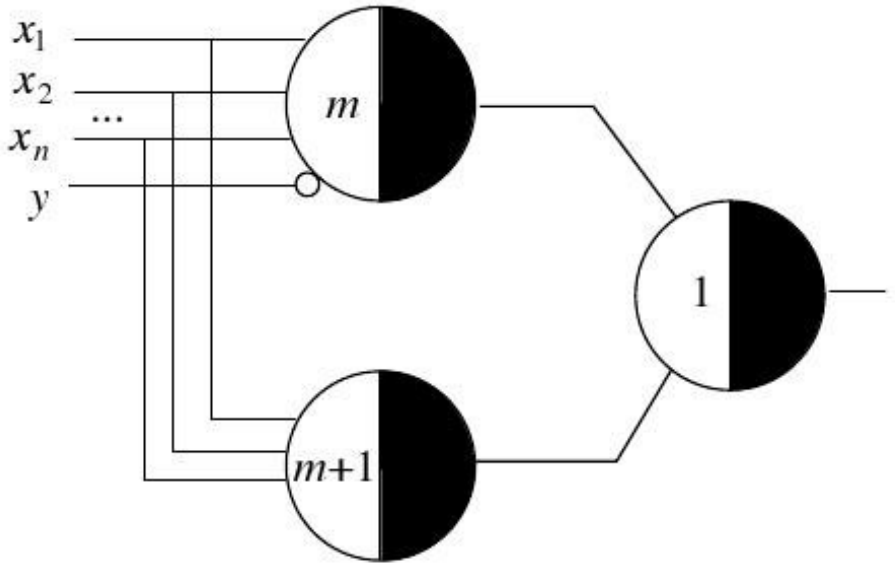
- **Inhibicja absolutna** - używana w jednostkach McCullocha-Pittsa.
- **Inhibicja względna** - wejścia mają ujemną wagę, czego efektem jest podwyższenie progu.
- **Twierdzenie.** Sieci jednostek McCullocha-Pittsa są równoważne sieciom z inhibicja względną.
- Ze względu na powyższe równoważności można wybrać albo prostszą topologię sieci albo prostsze jednostki.
  - Inhibicja: $n' - 1|m \Leftrightarrow n'|m + 1$ (dolna jednostka; górna - zawsze zero - inhibicja absolutna).
  - Bez inhibicji: $n'|m \Leftrightarrow n'|m$ (górna jednostka; dolna - jeżeli górna daje zero to dolna też - wyższy próg; jeżeli górna daje jeden dolna nie ma wpływu).

Elementy progowe i ich sieci - sieci równoważne

relative inhibition



equivalent circuit with absolute inhibition



Two equivalent networks

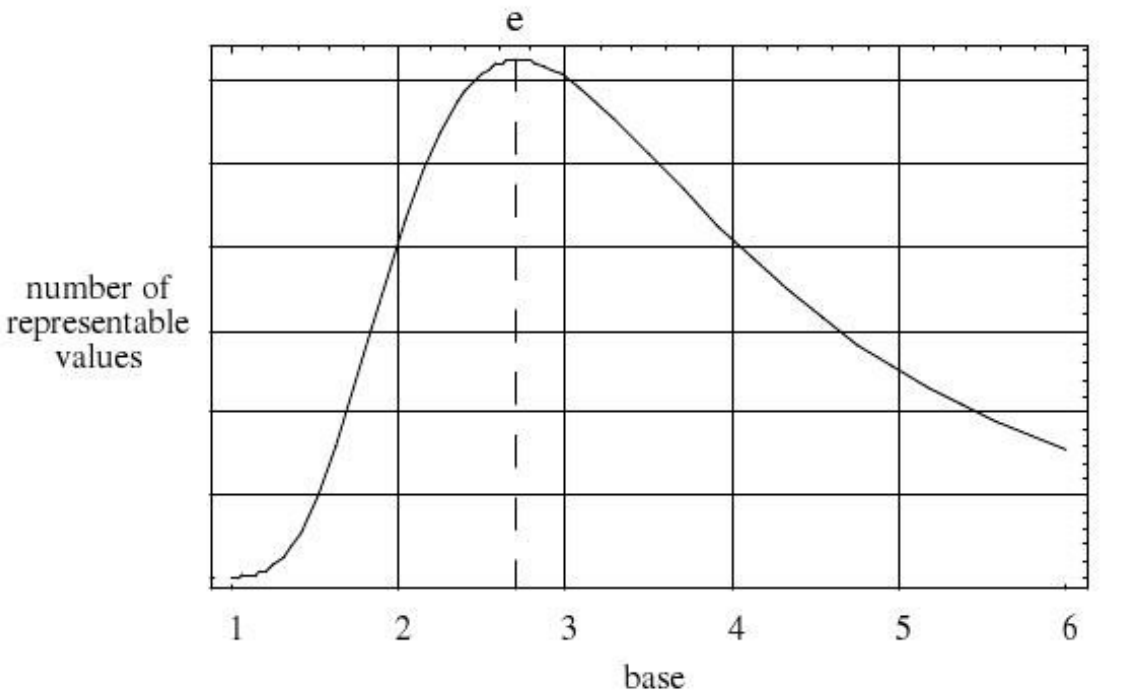
- Zatem każdą funkcję logiczną można przedstawić siecią jednostek bez wag. Jednak używając wag można zbudować prostszą sieć równoważną.

Elementy progowe i ich sieci - sygnały binarne

- Kolejne pytanie - **czy sygnały binarne nie są ograniczoną strategią** kodowania sygnałów?
- Każdy kolejny stan ma swoją cenę, która należy wziąć pod uwagę.

1. Oznaczmy przez b **liczbę stanów** na kanał komunikacyjny oraz przez c **liczbę kanałów** wejściowych.
2. **Koszt implementacji** to $K = \gamma bc$.
3. **Liczba wartości** którą można reprezentować to $b \cdot ... \cdot b = b^c$.
4. $c = K/(\gamma b)$; oznaczając $\kappa = K/\gamma$ szukamy wartości b, która maksymalizuje funkcję $\mathbf{b}^{\kappa/\mathbf{b}}$ (koszt κ stały).

Elementy progowe i ich sieci - sygnały binarne



Number of representable values as a function of the base

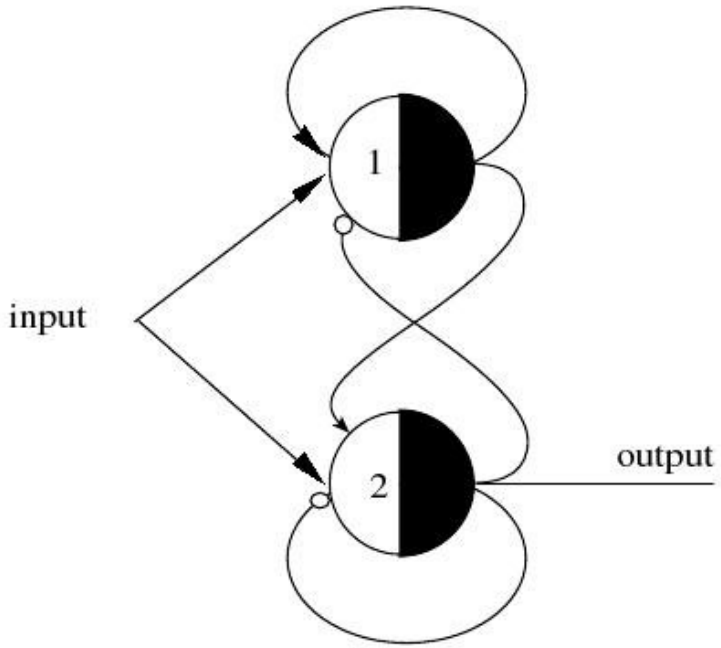
- Wartością optymalną jest stała Eulera, zatem 3 stany to wartość optymalna.
- Ze wzgledu na binarność układów biologicznych i elektronicznych wybieramy 2 stany.
  - Dla uproszczenia analizy założymy możliwość przekazywania liczb rzeczywistych.

Elementy progowe i ich sieci - sieci rekurencyjne

- Do tej pory rozmiar danych wejściowych i wyjściowych był określony z góry, często chcemy aby wejście miało **zmienny rozmiar** np. dodawanie dwóch dowolnych liczb.
- Sieć nie-rekurencyjna nie może rozwiązać problemu **dodawania** - nie może przechowywać rezulatatów poprzednich działań.
- Sieci rekurencyjne mogą **przechowywać** częściowe rezultaty.
- Cykle w topologii sieci pozwalają przechowywać wyniki i ponownie ich używać.
- Jednostki McCullocha-Pittsa mogą funkcjonować w sieciach rekurencyjnych zakładając, że **obliczenie trwa jednostkę czasu**.
  - Sygnał dochodzący do jednostki w czasie t daje rezultat w czasie $t + 1$.
  - Możliwości sieci bez rekurencji mogą być także uzyskane w sieciach z rekurencją.
  - Należy **skoordynować czas dostarczenia sygnałów** wejściowych do jednostki.

Elementy progowe i ich sieci - sieci rekurencyjne

- **Opóźnienie czasowe** można uzyskać przez włączenie dodatkowych elementów (ten sam problem co w każdym komputerze z zegarem).
- Przykład prostej sieci rekurencyjnej - przetwarza sekwencję bitów tak, że dwie kolejne jedynki są zamieniane w 10.
  - np. 00110110 $\Rightarrow$ 00100100.



Network for a binary scaler

Elementy progowe i ich sieci - automaty komórkowe

- Poprzednia sieć jest przykładem **automatu komórkowego**.
- Jest to abstrakcyjne urządzenie mogące przyjmować różne stany. Stan urządzenia zmienia się w zależności od sygnału wejściowego. Sygnał wyjściowy także zależy także od stanu urządzenia.
  - W przykładzie stan to kombinacja sygnałów krążących w sieci w danym momencie.
- **Automaty skończone** - skończona liczba stanów i może reagować na skończoną liczbę sygnałów.
- Przejścia między stanami i sygnały wyjściowe można przedstawić w tabeli.

Elementy progowe i ich sieci - automaty komórkowe

- Automat o dwóch stanach Q_1 oraz Q_2.
- Sygnał 1 powoduje przejście układu w stan Q_1. 0 powoduje przejście w stan Q_0.
- Wartość na wyjściu zależy od stanu niezależnie od sygnału wejściowego.

state transitions

input \ state	Q_0	Q_1
0	Q_0	Q_0
1	Q_1	Q_1

output table

input \ state	Q_0	Q_1
0	0	1
1	0	1

State tables for a binary delay

Elementy progowe i ich sieci - automaty komórkowe

- Diagram obrazuje przejścia pomiędzy stanami - wartość na początku strzałki to sygał wejściowy a na środku to sygnał wyjściowy.
- Automat ten zatem przechowuje ostatni bit na wejściu w postaci swojego obecnego stanu.

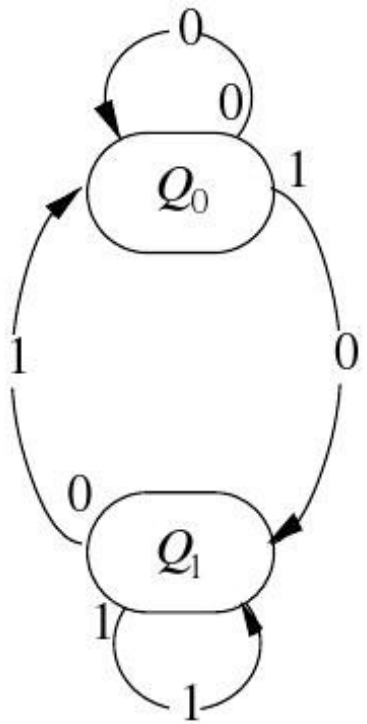


Diagram of a finite automaton

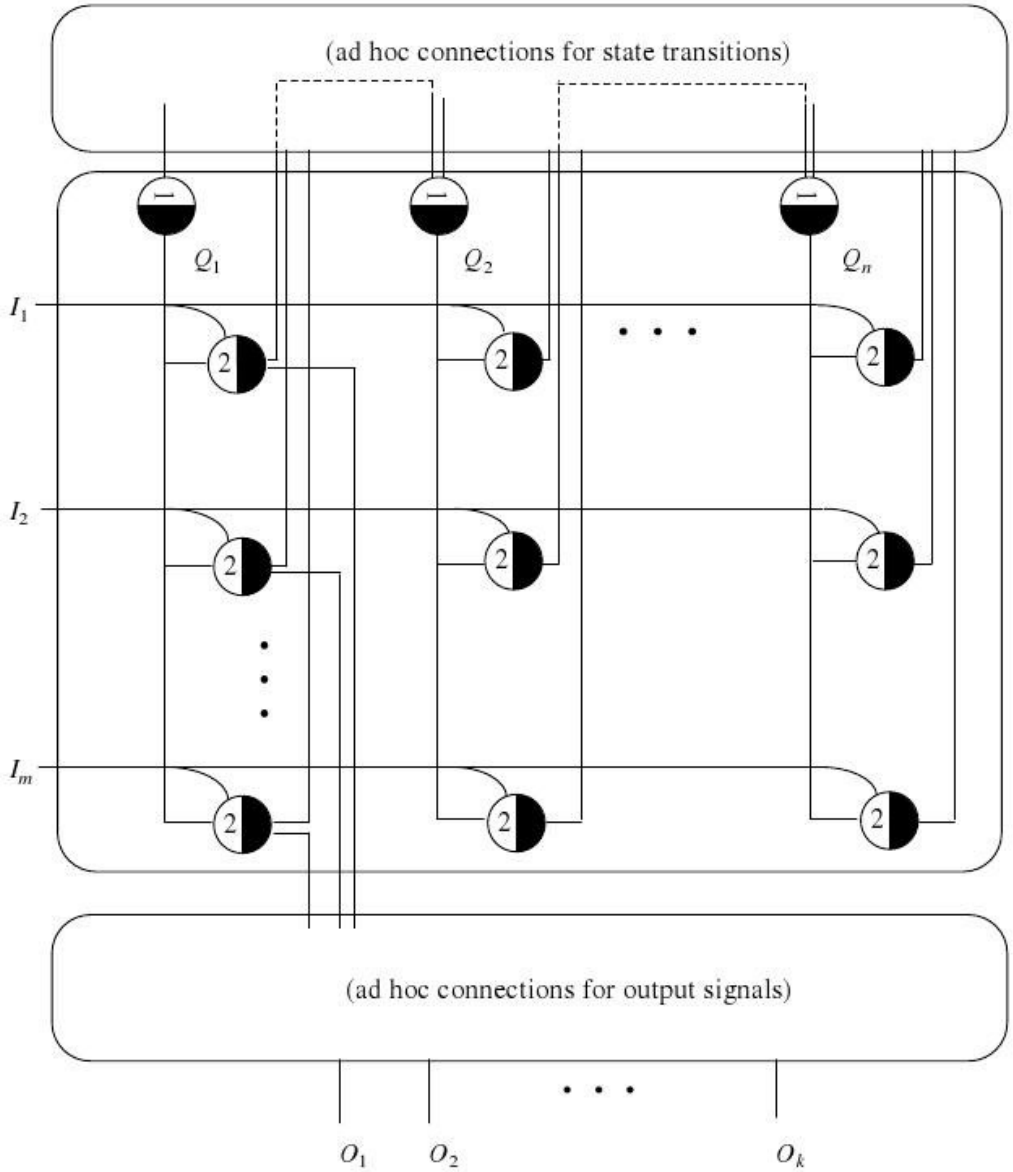
Elementy progowe i ich sieci - automaty komórkowe

- Skończony automat komórkowy nie może wykonywać działań wymagających nieskończonej liczby stanów, a więc **nie może realizować wszystkich funkcji obliczalnych**.
- **Komputery** są skończonymi automatami komórkowymi - duża liczba możliwych stanów zapewnia wystarczającą uniwersalność do wszystkich praktycznych zastosowań.

Elementy progowe i ich sieci - automaty skończone

- **Twierdzenie.** Każdy automat skończony może być symulowany siecią jednostek McCullocha-Pittsa.
- Dowód.
  - Sygnały wejściowe $I_1,\ldots,I_m$.
  - W każdej chwili t jedynie jedna z tych linii przekazuje 1 (pozostałe 0).
  - Załóżmy, że stanem początkowym sieci jest Q_1, tzn. tylko jedna z linii $Q_1,\ldots,Q_n$ ma wartość 1.
  - W chwili $t+1$ tylko jedna jednostka *AND* (tzn. o progu 2) daje 1 (linie wejścia i stanu są równe 1).
  - Połączenia w górnej części diagramu kontrolują zmiany stanu układu np. jeżeli kombinacja sygnałów 1 na Q_1 i I_1 w chwili t powoduje przejście do stanu Q_2 w czasie $t+1$, to należy połączyć odpowiednią jednostkę *AND* z jednostką *OR* (tzn. o progu 2) odpowiadającą stanowi Q_2; na wyjściu Q_2 pojawi się w chwili $t+2$.
  - Jeżeli w czasie $t+2$ pojawi się nowy sygnał wejściowy np. I_2 to nowym stanem będzie Q_n.
  - Dolna część diagramu zawiera połączenia potrzebne do otrzymania odpowiedniego sygnału wyjściowego.

Elementy progowe i ich sieci - automaty skończone

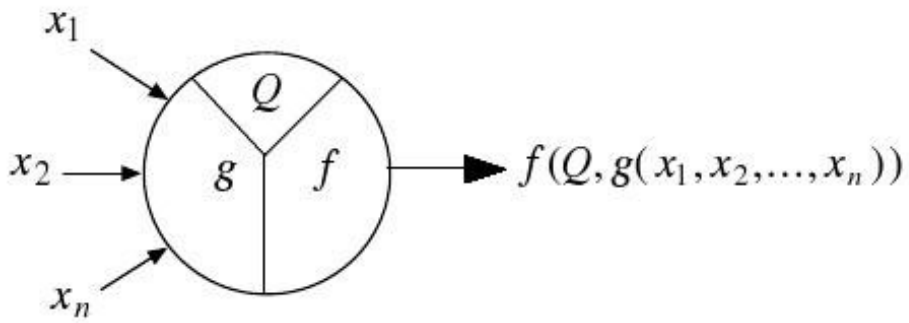


Implementation of a finite automaton with McCulloch–Pitts units

Elementy progowe i ich sieci - klasyfikacja sieci

1. **Sieci z wagami i bez wag** (pokazaliśmy że są równoważne).
   - Główna różnica - rodzaj algorytmu uczącego (bez wag - można zmienić tylko progi i topologię; w sieciach z wagami topologia zwykle nie jest modyfikowana).
2. **Sieci synchroniczne i asynchroniczne**.
   - Modele synchroniczne - wyjścia są obliczane jednocześnie (zawsze możliwe w sieci bez cykli)
   - W sieciach warstwowych - możliwe obliczanie synchroniczne w warstwach, które obliczane są kolejno.
   - Modele asynchroniczne - każda jednostka jest obliczana niezależnie od innych (w sieciach Hopfielda w losowo wybranych momentach); cykle nie są problemem.
3. **Modele z pamięcia stanu lub bez niej**.
   - Przykład. Jednostka, w której stan Q jest zapisywany po każdym obliczeniu.

Elementy progowe i ich sieci - klasyfikacja sieci



A unit with stored state Q

- Stan Q może modyfikować sygnał wyjściowy w kolejnej aktywacji.
- Jeżeli liczba wejść i stanów jest skończona jest to automat skończony. Ponieważ każdy automat można symulować siecią jednostek bez pamięci, zatem sieci z pamięcią stanu są **równoważne** sieciom bez pamięci (dane są przechowywane w postaci wzorca rekursji).

* Sieci o zmiennych progach i wagach są równoważne sieciom o stałych parametrach, jeżeli dopuścić rekursję.

Elementy progowe i ich sieci - analiza harmoniczna

- **Analiza harmoniczna funkcji logicznych** - badanie rozkładu wartości niezerowych.
- Można wypisać wartości funkcji w ciągu, zatem można myśleć o niej jak o jednowymiarowej funkcji.
- Przedstawmy funkcję $f : \mathbb{R}^m \to \mathbb{R}$ jako kombinację liniową n funkcji

$$f = a_1 f_1 + a_2 f_2 + ... + a_n f_n$$

  - dziedzina taka sama jak dziedzina funkcji bazy
- Dla danych stałych błąd kwadratowy w całej dziedzinie V wynosi

$$E = \int_V (f - (a_1 f_1 + a_2 f_2 + ... + a_n f_n))^2 \, dV$$

  - założenie - funkcje bazy są całkowalne na V

Elementy progowe i ich sieci - analiza harmoniczna

- Aby zminimalizować błąd E potrzebne są pochodne

$$0 = \frac{dE}{da_i} = -2 \int_V f_i \left(f - a_1 f_1 - a_2 f_2 - \ldots - a_n f_n \right) \right) dV, \quad i = 1, \ldots, n$$

- Prowadzi to do układu n równań

$$a_1 \int f_i f_1 + a_2 \int f_i f_2 + \ldots + a_n \int f_i f_n = \int f_i f$$

- W postaci macierzowej

$$\begin{pmatrix} \int f_1 f_1 & \int f_1 f_2 & \cdots & \int f_1 f_n \\ \int f_2 f_1 & \int f_2 f_2 & & \int f_2 f_n \\ \vdots & & \ddots & \vdots \\ \int f_n f_1 & \int f_n f_2 & \cdots & \int f_n f_n \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \int f_1 f \\ \int f_2 f \\ \vdots \\ \int f_n f \end{pmatrix}$$

Elementy progowe i ich sieci - analiza harmoniczna

- Powyższe równanie jest bardzo ogólne - jedyne założenie to całkowalność $f_i f_j$ oraz $f_i f$.
- Wersja dyskretna ($\sum f_i f_j = \sum_{k=1}^{m} f_i(x_k) f_j(x_k)$):

$$\begin{pmatrix} \sum f_1 f_1 & \sum f_1 f_2 & \cdots & \sum f_1 f_n \\ \sum f_2 f_1 & \sum f_2 f_2 & & \sum f_2 f_n \\ \vdots & & \ddots & \vdots \\ \sum f_n f_1 & \sum f_n f_2 & \cdots & \sum f_n f_n \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \sum f_1 f \\ \sum f_2 f \\ \vdots \\ \sum f_n f \end{pmatrix}$$

- W przypadku przybliżania wielomianami $x^0, ..., x^{n-1}$ punktów $(x_1, y_1), ..., (x_m, x_m)$

$$\begin{pmatrix} m & \sum x_i & \cdots & \sum x_i^{n-1} \\ \sum x_i & \sum x_i^2 & & \sum x_i^n \\ \vdots & & \ddots & \vdots \\ \sum x_i^{n-1} & \sum x_i^n & \cdots & \sum x_i^{2n-2} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum x_i y_i \\ \vdots \\ \sum x_i^{n-1} y_i \end{pmatrix}$$

Elementy progowe i ich sieci - analiza harmoniczna

- Jeżeli funkcje bazy są ortogonalne - macierz diagonalna.
  - Transformata Fouriera - $sin(k_i x)$ oraz $cos(k_i x)$
  - Jeżeli żadna z par sinusów nie ma tej samej liczby falowej k_i, to $\int_0^{2\pi} sin(k_i x) sin(k_j x) = \pi \delta_{ij}$ (podobnie cosinusy), wtedy

$$\pi \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \int f_1 f \\ \int f_2 f \\ \vdots \\ \int f_n f \end{pmatrix}$$

Elementy progowe i ich sieci - transformata Hadamarda-Walsha

- Fałsz: -1, prawda 1.
- Dla funkcji logicznej n zmiennych $x_1, ..., x_n$ można użyć poniższego zbioru 2^n funkcji elementarnych:
  - Funkcja stała $(x_1, ..., x_n) \mapsto 1$
  - $\binom{n}{k}$ jednomianów $(x_1, ..., x_n) \mapsto x_{l_1} x_{l_2} \cdots x_{l_k}$
- Funkcje są ortogonalne.
- W przypadku dwóch zmiennych **transformata Hadamarda-Walsha** dana jest relacją

$$4 \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}$$

Elementy progowe i ich sieci - transformata Hadamarda-Walsha

- Ogólnie 2^n argumentów oblicza się następująco

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{2^n} \end{pmatrix} = H_n \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{2^n} \end{pmatrix}$$

gdzie macierz H_n definuje się rekurencyjnie

$$H_n = \frac{1}{2} \begin{pmatrix} H_{n-1} & H_{n-1} \\ -H_{n-1} & H_{n-1} \end{pmatrix}$$

$$H_1 = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

Elementy progowe i ich sieci - transformata Hadamarda-Walsha

- Funkcję *AND* można przedstawić jako

$$x_1 \wedge x_2 = \frac{1}{4}(-2 + 2x_1 + 2x_2 + 2x_1 x_2)$$

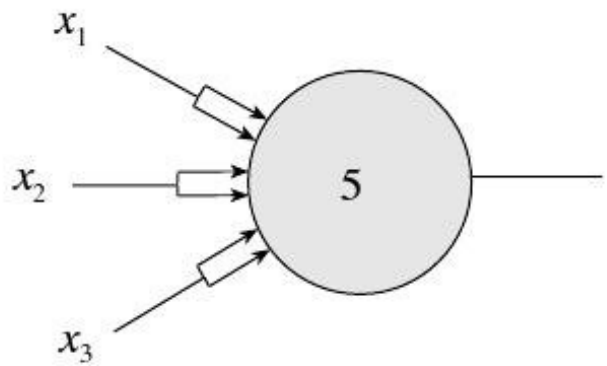
gdzie współczynniki wynikają z równania:

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}$$

- Powyższe wyrażenie oznacza, że dowolną funkcję logiczną można przedstawić siecią neuronów wykonujących dodawanie i mnożenie binarne.
- Następnie techniki kombinatoryczne pozwalają na optymalizację sieci.

Elementy progowe i ich sieci - zastosowania

- Ogólnie: elementy progowe - efektywnie obliczają funkcje logiczne (nieduża liczba jednostek; 2 warstwy); niektóre funkcje nie mogą być obliczone w ustalonej liczbie warstw bez wykładniczego zwiększania liczby bramek konwencjonalnych (nawet w przypadku nieograniczonej liczby wejść).
- Konwencjonalne sieci nie gwarantują stałego opóźnienia.
- Logika progowa daje łatwą metodę otrzymania **tolerancji na błędy**:



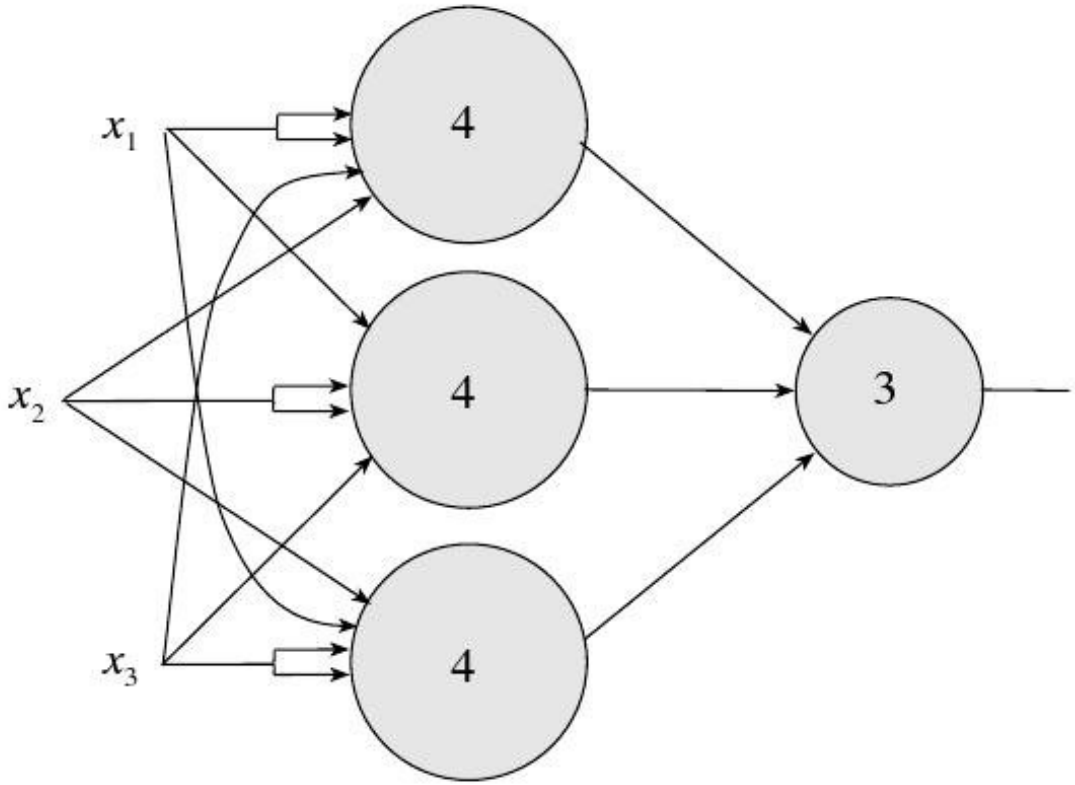
Fault-tolerant gate

Elementy progowe i ich sieci - zastosowania

- Powyższa jednostka oblicza $x_1 \wedge x_2 \wedge x_3$.
- Bez wejść nadmiarowych - Jeżeli na każdym wejściu może wystąpić błąd z prawdopodobieństwem p, to prawdopodobieństwo błędu wyniku (przypadku niezależnie transmitowanych i równych x_1, x_2, x_3) jest $3p$.
- **Z wejściami nadmiarowymi** - dzięki progowi 5 wynik będzie prawidłowy nawet w przy wystąpieniu błędu.
  - Prawdopodobieństwo, że dwie 1 dotrą do jednostki jako zera jest p^2, a kombinacji 2 wejść z 6 jest 15 prawdopodobieństwo uzyskania złej odpowiedzi to $15p^2$.
  - Dla małego p taka bramka jest lepsza: $15p^2 < 3p$.
  - Podobnie można analizować inne kombinacje wartości wejściowych.

Elementy progowe i ich sieci - zastosowania

- Jeżeli neurony powodują błędy można zbudować **sieć nadmiarową**.
- W przypadku bezbłędnych neuronów można ją zredukować do poprzedniej bramki.



A fault-tolerant AND built of noisy components

Elementy progowe i ich sieci - zastosowania

- Pierwsze 3 jednostki dają sygnał wyjściowy z prawdopodobieństwem 1 dla progu θ oraz z prawdopodobieństwem p dla progu $\theta - 1$.
- Podwójne połączenia dają nadmiarowość.
- Ostatnia jednostka dają sygnał wyjściowy z prawdopodobieństwem 1 gdy zadziałają wszystkie 3 jednostki pierwszej warstwy oraz z prawdopodobieństwem p gdy zadziałają dwie.

Elementy progowe i ich sieci - historia

- 1927 - Warren McCulloch zaczął rozważać sieci sztucznych neuronów (problemy - inhibicja i sieci rekurencyjne nie były jeszcze potwierdzone do 1930).
- Norbert Wiener i Arturo Rosenblueth - dyskusja ważnej roli sprzężenia zwrotnego w systemach biologicznych.
- "Cybernetyka" Wienera - pierwszy best-seller w dziedzinie sztucznej inteligencji.
- Model McCullocha-Pittsa doczekał się wielu modyfikacji.
- Teoretycznie elementy progowe są ogólniejsze ale technologia ma problem z nieograniczoną liczbą wejść - nadzieja w elementach optycznych.
- Lata 60te - komputer DONUT (Lewis i Coates): 614 bramek o maksymalnie 15 wejściach.
- John von Neumann - pracował nad zagadnieniem jak zbudować niezawodny mechanizm liczący z zawodnych elementów.

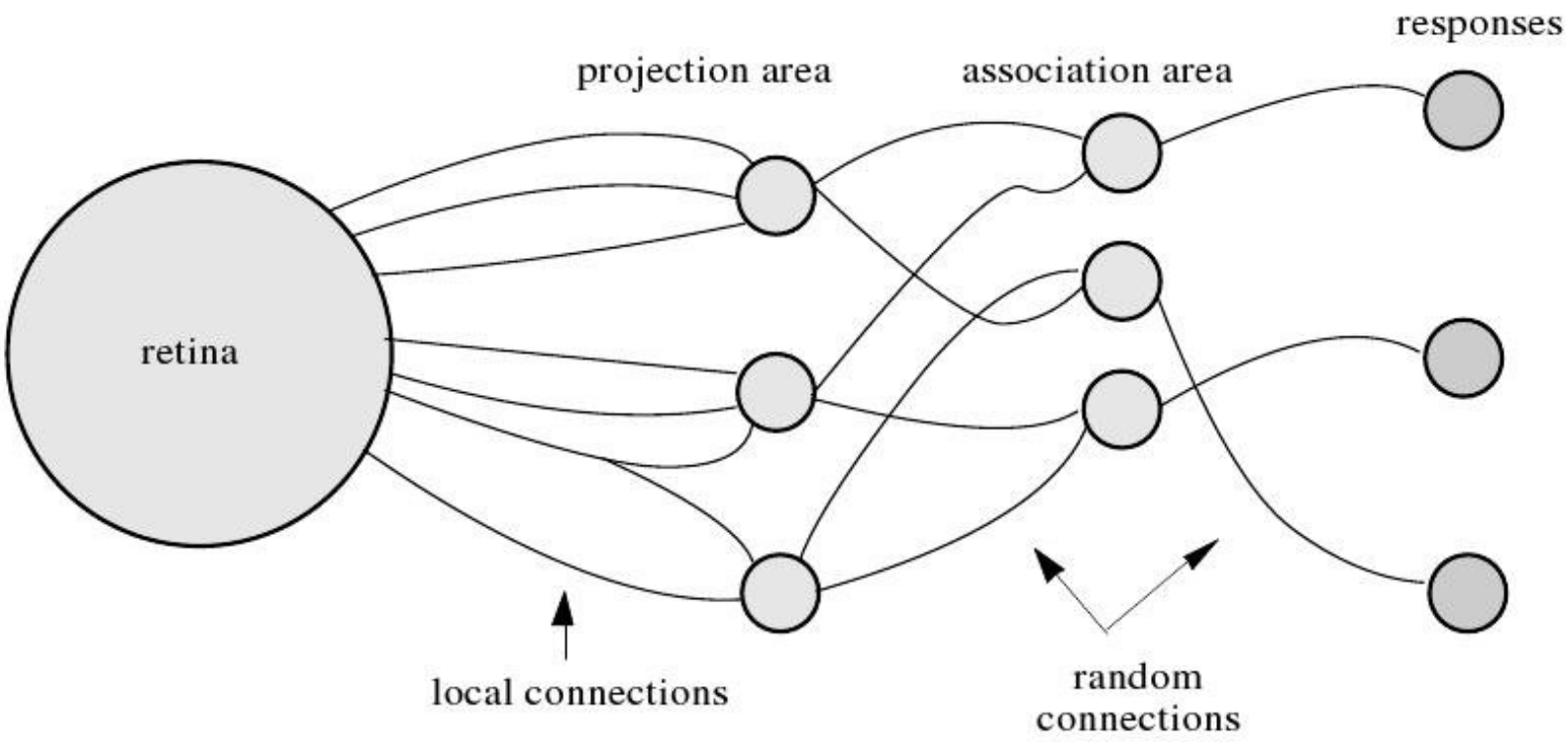
Perceptron

- Jednostki McCullocha-Pittsa pozwalają zbudować sieci obliczające dowolną funkcję logiczną i symulować dowolony automat skończony - jednak nie ma **wolnych parametrów**, które można by zmieniać na potrzeby różnych problemów.
- **Uczenie** można wprowadzić jedynie przez modyfikowanie **połączeń** i **progów**, co nie jest praktyczne.
- Dlatego tak ważne są **neurony z wagami**.

Perceptron

- 1958 - Frank Rosenblatt (psycholog) - zaproponował perceptron - model obliczeniowy ogólniejszy niż neurony McCullocha-Pittsa.
- Istotną innowacją było wprowadzenie wag oraz specjalnego wzorca połączeń.
- Oryginalny model - **perceptron klasyczny** - elementy progowe i połączenia określane stochastycznie a uczenie następuje dzięki zmianie wag przy pomocy algorytmu.
- **Perceptron** - model analizowany i ulepszany w latach 60-tych (Minsky i Papert).
- Klasyczny perceptron był siecią do rozwiązywania pewnych **problemów rozpoznawania wzorców**.
- **Retina** - powierzchnia rzutowania wzorca - transmituje wielkości binarne do **warstwy rzutowania** perceptronu przez połączenia, które są **deterministyczne i niezmienne**. Pozostałe połączenia są wybierane **stochastycznie** aby model był **biologicznie wiarygodny**.

Perceptron



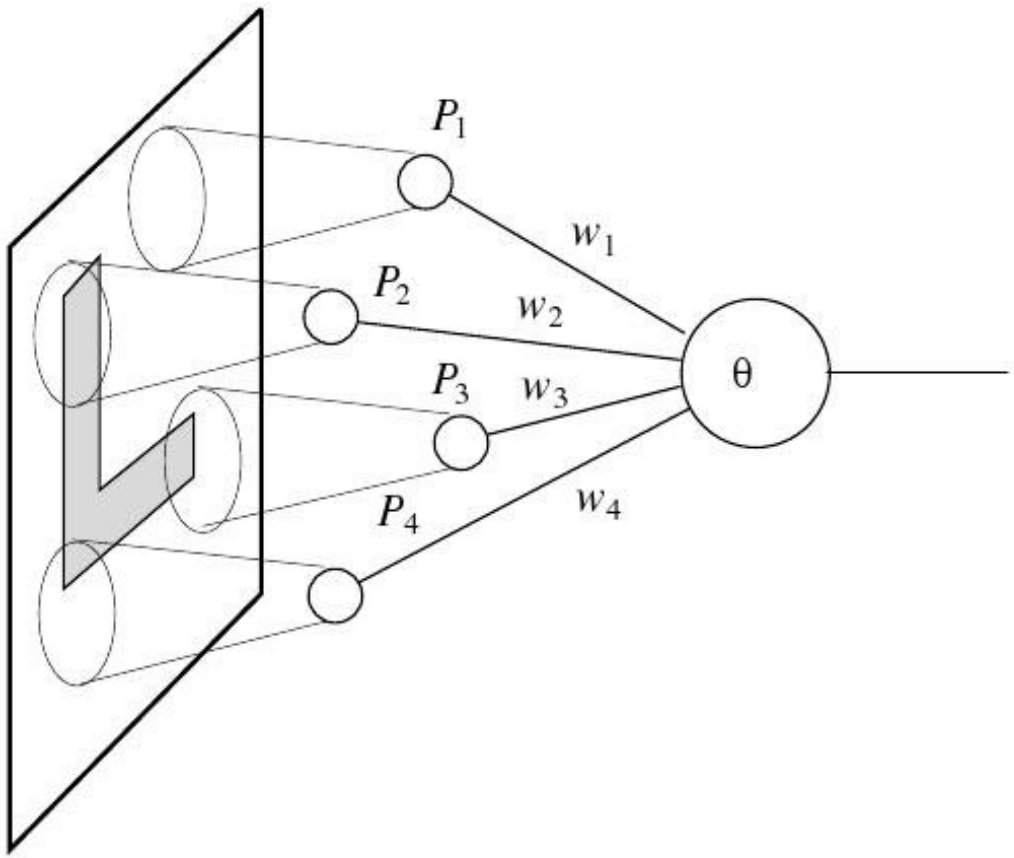
The classical perceptron [after Rosenblatt 1958]

Perceptron

- **Idea** - wytrenowanie układu do rozpoznawania wzorców wejściowych w rejonie zmiennych połączeń; algorytm uczący musi znaleźć opdowiednie wagi.
- Formalnie - **jedyna różnica** pomiędzy neuronami McCullocha-Pittsa a perceptronami to obecność wag w sieci.
- Rosenblatt - studiował też modele z innymi modyfikacjami np. ograniczeniem na maksymalną liczbę wejść jednostki.
- Minsky i Papert - określili istotne cechy modelu Rosenblatta - badali możliwości obliczeniowe perceptronu z różnymi założeniami.

Perceptron

- W modelu Minsky'ego-Paperta - pewne piksele retiny (o wartościach binarnych) połączone są z elementami logicznymi tzw. **predykatami** (mogą one być dowolnie skomplikowane).



Predicates and weights of a perceptron

Perceptron

- Predykaty - ograniczenia - liczba punktów retiny, które może jednocześnie analizować dany predykat lub dystans pomiędzy tymi punktami.
- Predykaty przesyłają wartości binarne do elementu progowego z wagą - ostateczna decyzja w problemie rozpoznawania wzorca.
- **Pytania.** Jakie rodzaje wzorców można rozpoznać w ten równoległy sposób? Czy jest granica na to co można obliczyć równolegle używając nieograniczonej mocy obliczeniowej dla każdego predykatu, jeżeli nie może on widzieć całej retiny?
- Drugie pytanie jest podobne do zagadnienia przyspieszenia w obliczeniach równoległych - jaką część zadania można zrównoleglić a jaka jest nieuniknienie sekwencyjna?

Perceptron - ograniczenia

- **Ograniczenia** na możliwości obliczeniowe wynikają z:
  - pole widziane przez jeden predykat nie pokrywa całej retiny,
  - na wyjściu z predykatu - wartość binarna.
- Przed Minskym i Papertem nie wiadomo było jakie klasy problemów rozpoznawania wzorców perceptron może rozwiązać efektywnie.
- Liczba predykatów - ustalona ale mimo nieograniczonej mocy obliczeniowej każdego z nich jest końcowy etap z jednym elementem progowym.
- Wszystkie predykaty muszą współpracować produkując częściowe rezultaty dla globalnej decyzji.
- Jakie problemy można tak rozwiązać?

Perceptron - ograniczenia

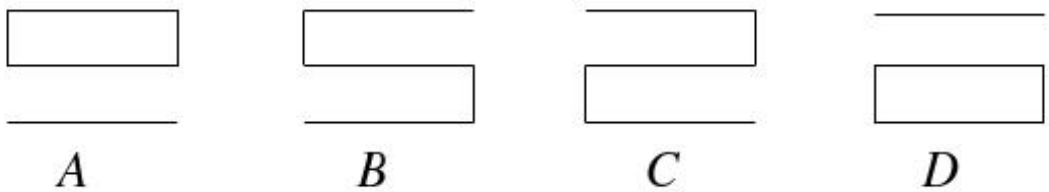
- Może sie wydawać że model Minsky-Papert jest uproszczeniem procesu równoległego.
- Zawiera najważniejsze idee odróżniające równoległość od sekwencyjności.
- Niektóre algorytmy po zrównolegleniu zawierają część nieredukowalnie sekwencyjną, co ogranicza maksymalne możliwe przyspieszenie po zrównolegleniu.
- **Prawo Amdahla** - matematyczna relacja pomiędzy przyspieszeniem a nieredukowalnie sekwencyjną częścią algorytmu.
- Istnieją problemy, których nie można rozwiązać przy pomocy pojedynczej jednostki decyzyjnej.
- Ograniczenia na pole widziane przez jeden predykat są realistyczne. Predykaty są wcześniej ustalone i problem może być dowolnie duży (retinę można powiększyć).

Perceptron - ograniczenia

- **Podział perceptronów** ze względu na liczbę punktów i ich połączeń do predykatu (Minsky-Papert):
  - perceptrony o ograniczonej średnicy - każdy predykat widzi pole o ograniczonej średnicy,
  - preceptrony ograniczonego rzędu - każdy predykat widzi ograniczoną liczbę punktów,
  - perceptrony stochastyczne - predykat widzi pewną ilość losowo wybranych punktów.
- Pewne wzorce są trudniejsze do rozpoznania niż inne a powyższa klasyfikacja jest pierwszą próbą zdefiniowania **klas złożoności** dla rozpoznawania wzorców.
- **Ciągłość** - nie może być rozpoznana przez ograniczone układy.

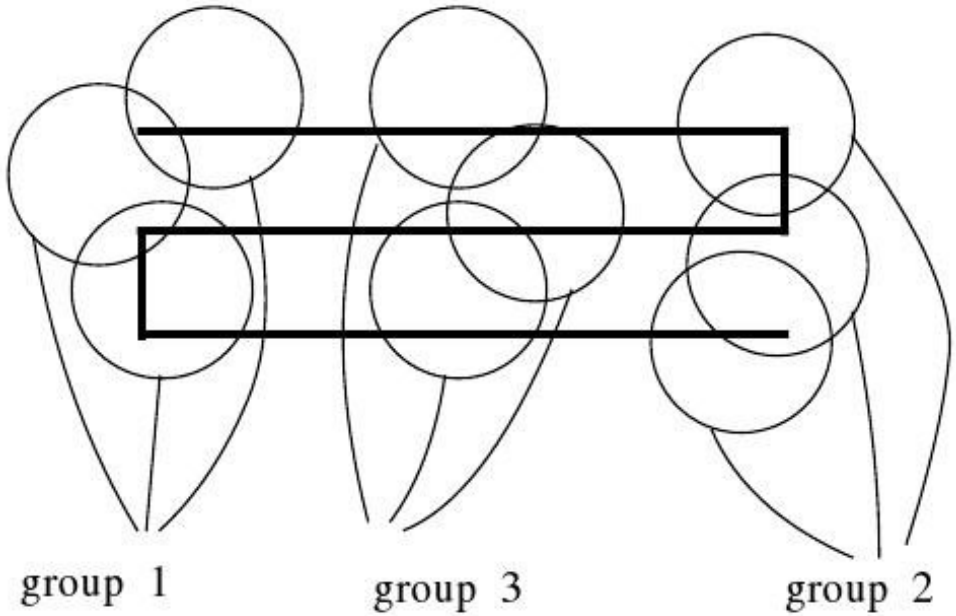
Perceptron - ograniczenia

- **Twierdzenie.** Żaden perceptron o ograniczonej średnicy nie może zdecydować czy figura geometryczna jest ciągła.
- **Dowód.** Przez zaprzeczenie - przyjmijmy założenie że jest to możliwe.
- Poniższe figury można rozciągnąć tak, że predykaty nie mogą widzieć jednocześnie jej lewej i prawej strony.



Perceptron

- Predykaty można podzielić na grupy:



Receptive fields of predicates

- Perceptron decyduje o ciągłości figury obliczając ($S \geq 0$ - ciągła)

$$S = \sum_{P_i \in G1} w_{1i} P_i + \sum_{P_i \in G2} w_{2i} P_i + \sum_{P_i \in G3} w_{3i} P_i - \theta \geq 0.$$

Perceptron

- Dla A: $S < 0$. $\mathbf{A} \rightarrow \mathbf{B}$: grupa 3 daje ten sam rezultat (środek wygląda tak samo).
- Jedynie predykaty grupy 2 zmienią swoje wyjście o $\Delta_2 S$:

$$S + \Delta_2 S \geq 0 \Rightarrow \Delta_2 S \geq -S.$$

- Rozważmy $\mathbf{A} \rightarrow \mathbf{C}$: predykaty grupy 1 muszą się dostosować

$$S + \Delta_1 S \geq 0 \Rightarrow \Delta_1 S \geq -S.$$

- $\mathbf{A} \rightarrow \mathbf{D}$: grupa 1 nie odróżnia C od D; grupa 2 nie odróżnia B od D a grupa 3 nie zmieni się zatem całkowita zmiana S to:

$$\begin{aligned} \Delta S = \Delta_2 S + \Delta_1 S &\geq -2S \\ S + \Delta S &\geq -S > 0 \end{aligned}$$

- Ale D nie jest figurą ciągłą - sprzeczność.

Perceptron

- Ciągłość jest zatem wielkością globalną i nie można jej rozpoznać bez patrzenia na całość.
- Rozwiązaniem problemu może być sekwencyjne przetwarzanie danych z predykatów.
- Inne problemy trudne dla perceptronów np. nie można zdecydować czy zbiór punktów zawiera ich parzystą czy nieparzystą ilość jeżeli perceptron widzi ograniczoną liczbę punktów.

Perceptron - funkcje logiczne

- Poprzednio rozważaliśmy implementację funkcji logicznych w sieciach McCullocha-Pittsa.
- Teraz uwzględnienie wag pozwoli osiągnąć ten sam cel mniejszą liczbą jednostek.
- **Które funkcje można zaimplementować z użyciem pojedynczej jednostki?**
- Za perceptron będziemy uważać pojedynczy element progowy obliczający wartość wyjściową bez opóźnienia.

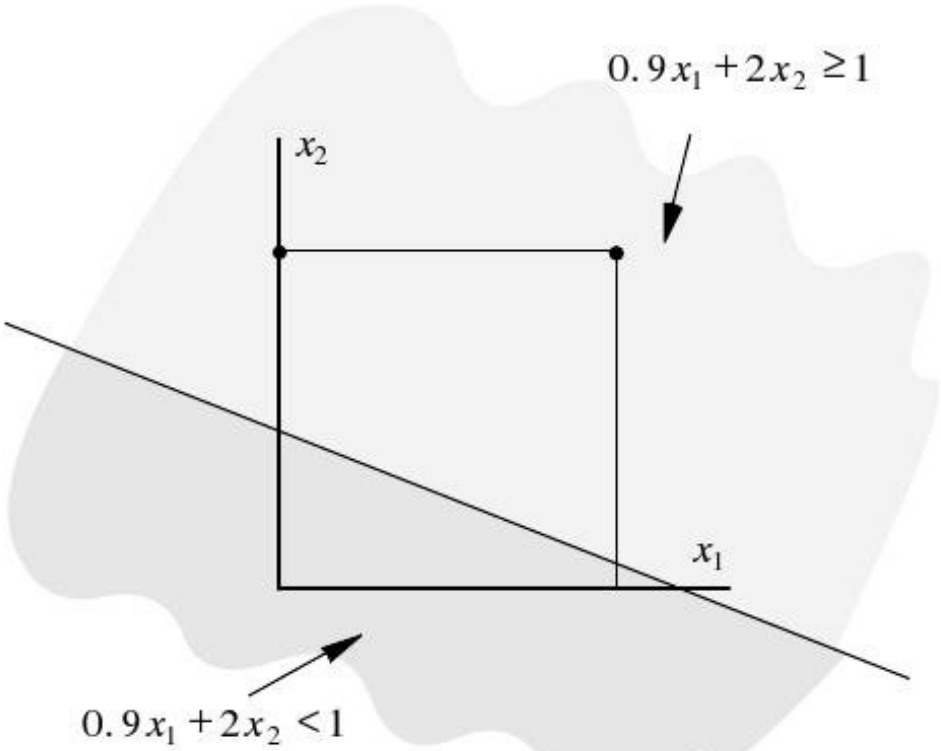
Perceptron - funkcje logiczne - interpretacja geometryczna

- **Definicja. Perceptron prosty** jest to jednostka progowa o progu θ, która otrzymując n wartości $x_1, ..., x_n \in \mathbb{R}$ poprzez wejścia o wagach $w_1, ..., w_n$ daje na wyjściu 1 jeżeli zachodzi warunek $\sum_{i=1}^{n} w_i x_i \geq \theta$, a 0 w przeciwnym wypadku.
- Wejścia mogą pochodzić od innych perceptronów lub też innego rodzaju jednostek obliczeniowych - nie ma to znaczenia.
- **Interpretacja geometryczna** - analogiczna do jednostek McCullocha-Pittsa z uogólnieniem w postaci wag np. perceptron o progu 1 i wagach 0.9 oraz 2.0 będzie sprawdzał warunek

$$0.9x_1 + 2x_2 \geq 1.$$

- Wagi pozwalają wygenerować dowolny podział przestrzeni płaszczyzną.

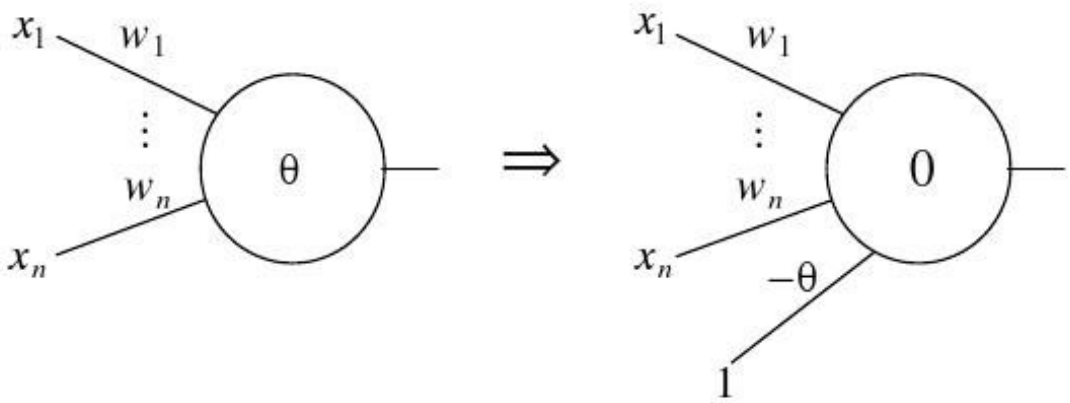
Perceptron - funkcje logiczne - interpretacja geometryczna



Separation of input space with a perceptron

Perceptron - funkcje logiczne - interpretacja geometryczna

- Wygodnie jest posługiwać się tylko **perceptronami o zerowym progu** - odpowiednia płaszczyzna przechodzi zawsze przez **środek układu współrzędnych**.
- Poniższe perceptrony są równoważne:



A perceptron with a bias

- Większość algorytmów uczenia sieci lepiej jest przedstawić w postaci o zerowym progu, wtedy n wymiarowy wektor wejść rozszerza się do $n+1$-wymiarowego $(x_1, ..., x_n, 1)$; wektor wag roszerza się do $(w_1, ..., w_n, w_{n+1})$, gdzie $w_{n+1} = -\theta$.

Perceptron - funkcje logiczne - problem XOR

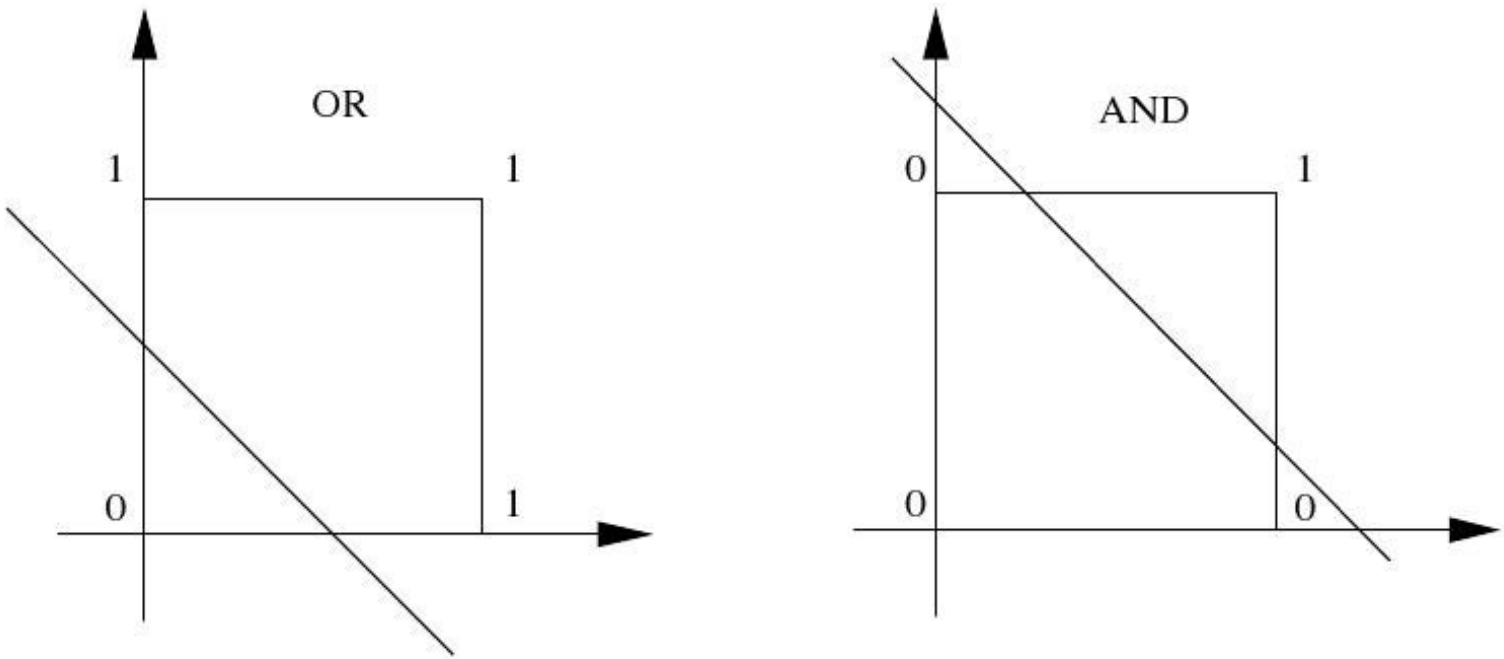
- Które funkcje logiczne są implementowalne jednym perceptronem?
- Sieć perceptronów będzie bardziej wszechstronna niż sieć jednostek McCullocha-Pittsa.
- Rozważymy jako przykład funkcje 2 zmiennych - jest ich $16 = 2^{2 \cdot 2}$.
- f_0 - funkcja stała, $f_{14} = OR$, $f_6 = XOR$.

The 16 Boolean functions of two variables

x_1	x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Perceptron - funkcje logiczne - problem XOR

- Funkcje obliczalne perceptronem - punkty o wartości 0 i 1 można oddzielić prostą.



Separations of input space corresponding to OR and AND

- Analiza tabeli - f_6 i f_9 nie są obliczane perceptronem.

Perceptron - funkcje logiczne - problem XOR

- Rozważając *XOR* analitycznie (w_1, w_2 - wagi) spełnione muszą być zależności (gdzie prawa strona wynika z definicji *XOR*):

$$\begin{array}{llll} x_1 = 0 \; x_2 = 0 & w_1 x_1 + w_2 x_2 = 0 & \Rightarrow & 0 < \theta \\ x_1 = 1 \; x_2 = 0 & w_1 x_1 + w_2 x_2 = w_1 & \Rightarrow & w_1 \geq \theta \\ x_1 = 0 \; x_2 = 1 & w_1 x_1 + w_2 x_2 = w_2 & \Rightarrow & w_2 \geq \theta \\ x_1 = 1 \; x_2 = 1 & w_1 x_1 + w_2 x_2 = w_1 + w_2 & \Rightarrow & w_1 + w_2 < \theta \end{array}$$

- Ponieważ $\theta > 0$ i $w_{1,2} \geq \theta$ ostatnia nierówność nie może być spełniona ($\theta > w_1 + w_2 \geq 2\theta \;\Rightarrow\; 1 > 2$).

Perceptron - funkcje liniowo separowalne

- Omówimy narzędzia do efektywnego opisu funkcji n zmiennych.
- Znając przypadek *XOR* można domyślać się, że istnieje więcej funkcji takiego typu.
- Geometria n-wymiarowego hiperześcianu.
- **Definicja.** Dwa zbiory punktów A i B w przestrzeni n-wymiarowej nazywamy **liniowo separowalnymi** jeżeli istnieje $n+1$ liczb rzeczywistych $w_1, ..., w_{n+1}$ takich, że każdy punkt $(x_1, ..., x_n) \in A$ spełnia warunek $\sum_{i=1}^{n} w_i x_i \geq w_{n+1}$ a każdy punkt $(x_1, ..., x_n) \in B$ spełnia warunek $\sum_{i=1}^{n} w_i x_i < w_{n+1}$.
- W naszym przypadku oczywiście $w_{n+1} = \theta$
- **Pytanie.** Perceptron oblicza jedynie funkcje liniowo separowalne - ile ich jest dla n argumentów.
  - $n = 2$: 14 z 16 możliwych,
  - $n = 3$: 104 z 256,
  - $n = 4$: 1882 z 65536.
- Mimo intensywnych badań żadnego ogólnego wzoru nie ma (można jedynie podać górne ograniczenie).

Perceptron - funkcje liniowo separowalne

- Perceptron liniowo dzieli przestrzeń wejść, jednakże proces szukania wag może być lepiej przedstawiony w **przestrzeni wag**.
- Dualność tą dla 3 wymiarów pokazuje rysunek:

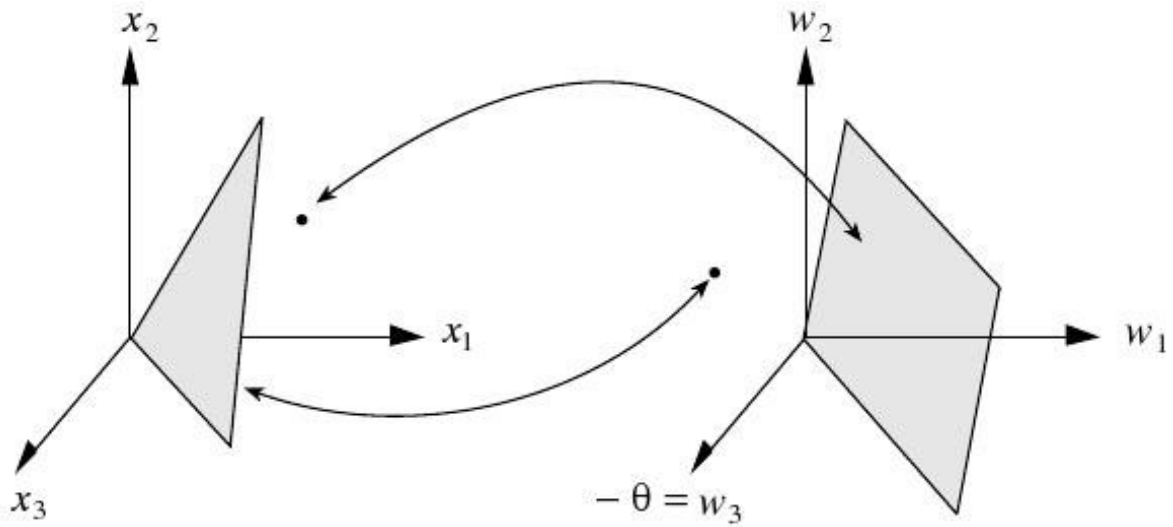


Illustration of the duality of input and weight space

Perceptron - funkcje liniowo separowalne

- Dla perceptronu o n wejściach szukanie liniowego podziału, jest szukaniem punktu w $(n+1)$-wymiarowej **przestrzeni wag** (wagi + θ).
- Każdy punkt $(n+1)$-wymiarowej przestrzeni wag odpowiada płaszczyźnie w $(n+1)$-wymiarowej rozszerzonej przestrzeni wejść. Punkt (w_1, w_2, w_3) definiuje płaszczyznę $w_1x_1 + w_2x_2 + w_3x_3 = 0$.
- Odwrotnie - jeżeli punkt (x_1, x_2, x_3) ma być w dodatniej półprzestrzeni to $w_1x_1 + w_2x_2 + w_3x_3 \geq 0$.
- **Algorytm uczenia perceptronu** - dobiera wagi i próg automatycznie.
- A i B dwa zbiory n-wymiarowych wektorów wejściowych, które mają być rozdzielone; perceptron oblicza funkcję binarną $f_w(x)$ taką, że $f_w(x) = 1$ dla $x \in A$ oraz $f_w(x) = 0$ dla $x \in B$. Funkcja taka zależy od wag i progu - pozwala zdefiniować **funkcję błędu** jako liczbę fałszywie sklasyfikowanych punktów przy danym wektorze wag w:

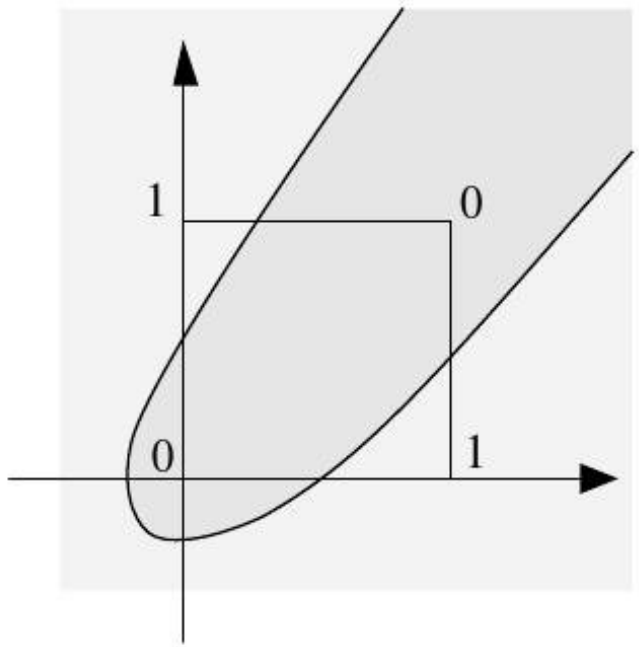
$$E(w) = \sum_{x \in A} (1 - f_w(x)) + \sum_{x \in B} f_w(x).$$

Perceptron - funkcje liniowo separowalne

- $E(w)$ - z definicji **dodatnia lub zero** i jest zdefiniowana na całej przestrzeni wag.
- Celem uczenia perceptronu jest jej **minimalizacja** - chcemy osiągnąć minimum globalne $E(w) = 0$; startuje się w losowo wybranego wektora wag i przeszukuje przestrzeń obliczając w każdym kroku $E(w)$.

Perceptron - funkcje liniowo separowalne

- Linowy podział przestrzeni przez perceptron zawęża ilość problemów rozwiązywalnych pojedynczym perceptronem.
- Można wprowadzić ogólniejszy podział przestrzeni - rozwiązanie problemu *XOR*:



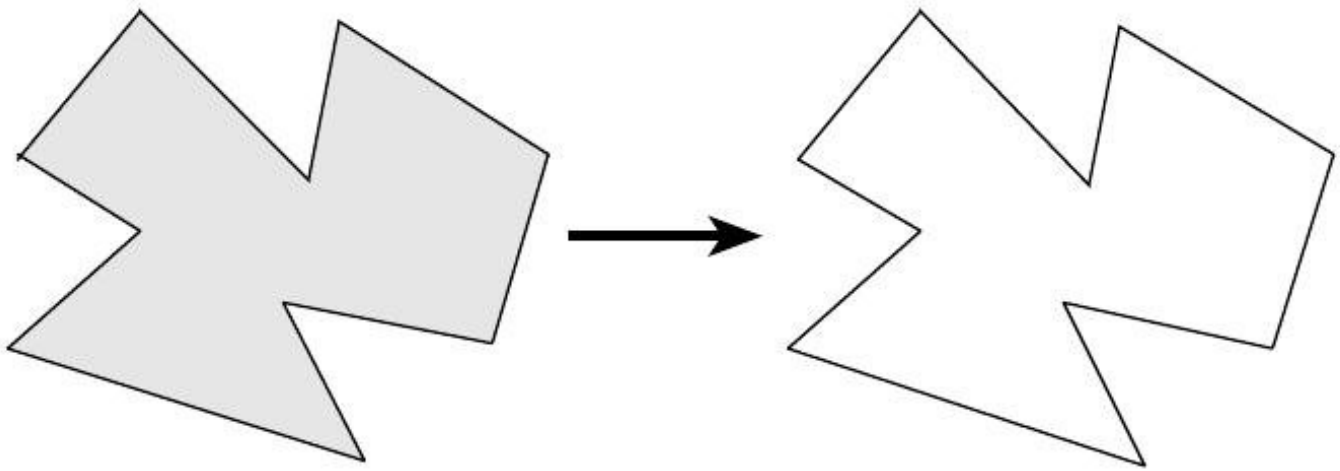
Non-linear separation of input space

Perceptron - funkcje liniowo separowalne

- **Krzywe decyzji** np. badano wielomiany i funkcje kawałkami wielomianowe (splines).
- Problem rozpoznawania wzorców - zakłada się, że wzorce są pogrupowane w klastry w przestrzeni wejść a odpowiednia kombinacja krzywych decyzji ma **wyizolować** jeden z klastrów.
- Kombinacja kilku perceptronów także pozwala wyizolować wypukły obszar przestrzeni.
- Ogólna idea - rozróżnianie **obszarów przestrzeni**; związany z tym **problem** - czy wolne parametry odpowiednich obszarów mogą być znalezione algorytmem uczącym.
- Kolejny wykład - **zawsze można znaleźć parametry dla prostych** jeżeli kasyfikowane wzorce są liniowo separowalne.
- Znajdywanie alorytmów dla innych krzywych decyzji jest tematem badań.

Perceptron - analogia biologiczna

- Perceptron jako prosty model znalazł liczne zastosowania np. przetwarzanie obrazów.
- Przykład - **wykrywanie brzegu** - każdy piksel figury po lewej porównuje się z najbliższymi sąsiadami - jeżeli czarny piksel będzie miał białego sąsiada to zostanie zaklasyfikowany jako należący do brzegu.



Example of edge detection

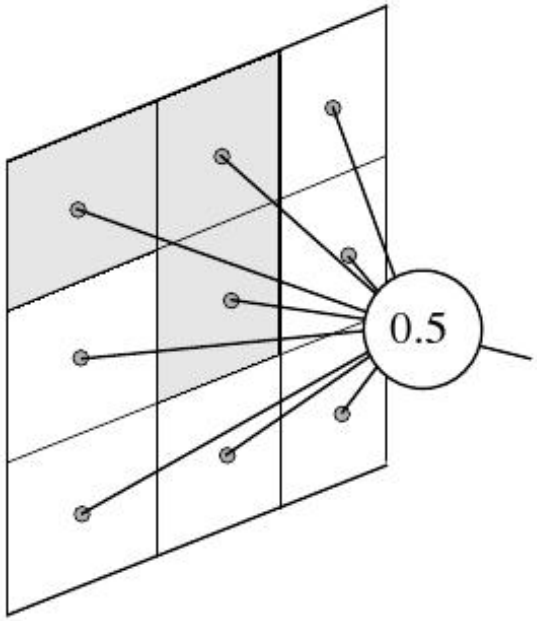
Perceptron - analogia biologiczna

- Algorytm wykrywania brzegu - łatwo stosować równolegle (potrzebna informacja lokalna).
- Każdy piksel ekranu (na który rzutowana jest figura) połączony jest z perceptronem tak, że perceptron 'widzi' piksel i jego otoczenie.
- Na rysunku podano wagi; w dziedzinie przetwarzania obrazów nazywa się to **operatorem konwolucji** (używamy go ustawiając w centrum dany piksel; operator ma maksimum w centrum a na obrzeżach lokalne minima).

-1	-1	-1
-1	8	-1
-1	-1	-1

Edge detection operator

Perceptron - analogia biologiczna



Connection of a perceptron to the projection grid

Perceptron - analogia biologiczna

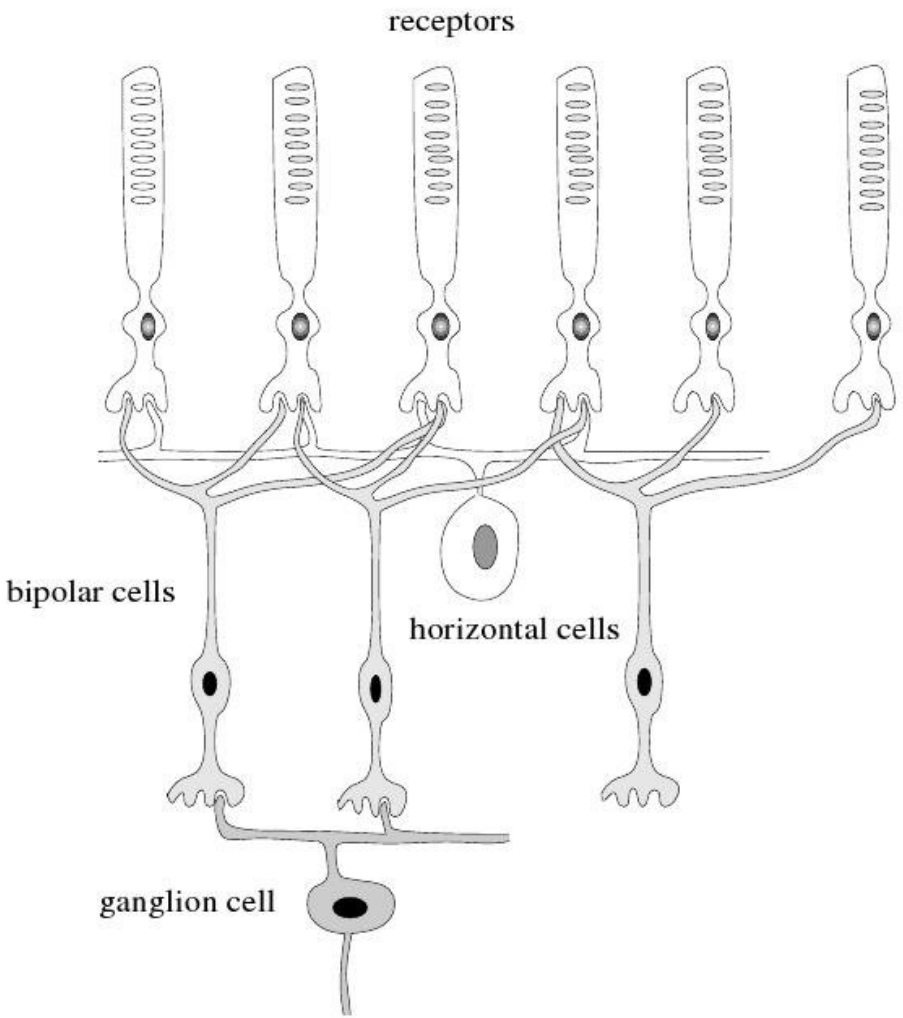
- Inne operatory - wykrywania poziomych lub pionowych linii, zamazywania lub wyostrzania obrazu.
- Wielkość otoczenia danego piksela można dostosowywać do konkretnego zastosowania.
- Operator do wykrywania pionowej krawędzi pomiędzy białą powierzchnią po lewej i czarną po prawej.

$$\begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix}$$

Perceptron - analogia biologiczna - struktura retiny

- **Ścieżka wzrokowa** - najlepiej poznana część ludzkiego mózgu.
- **Retina** jest początkiem tej drogi - to komórki nerwowe zoragnizowane w warstwy; wstępnie przetwarzają informację niezbędną do widzenia.
- Żaby - pewne neurony retiny regują na obecność małej plamki w polu widzenia.
- Komórki w retinie - połączone w taki sposób, że nerw łączący oko z mózgiem koduje sumę informacji z kilku fotoreceptorów - podobnie jak operator konwolucji.

Perceptron - analogia biologiczna - struktura retiny



The interconnection pattern in the retina

Perceptron - analogia biologiczna - struktura retiny

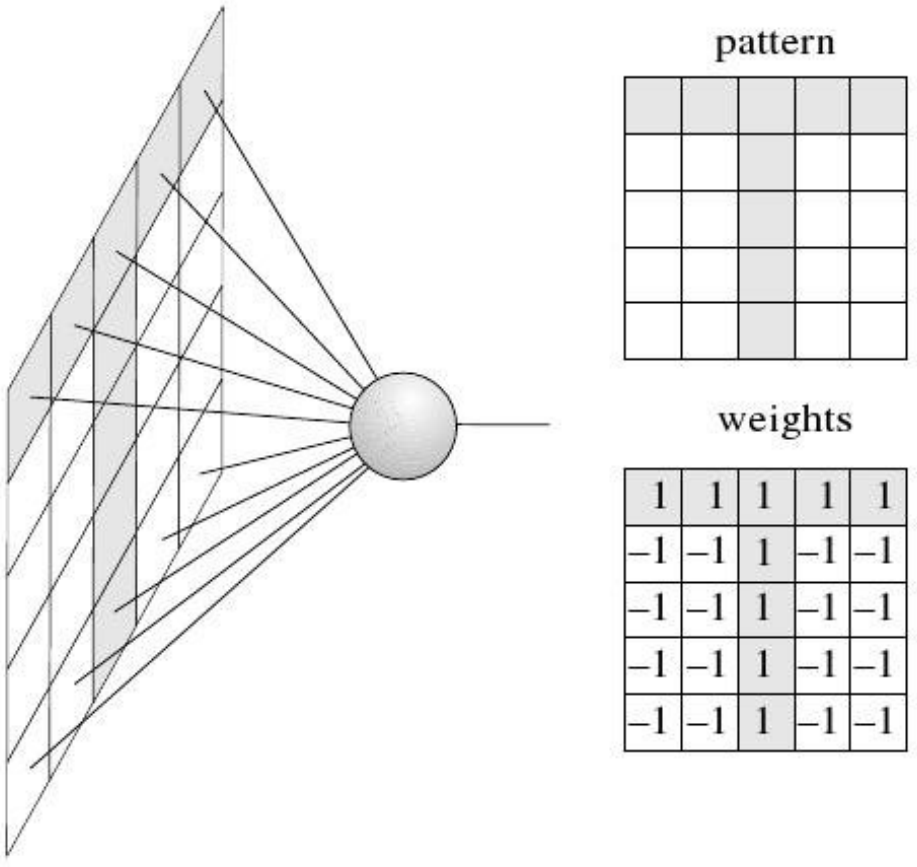
- **Fotoreceptory** - reagują na fotony depolaryzując się.
- **Komórki horyzontalne** - obliczają średnią luminancję (ilość światła emitowaną przez określoną powierzchnię w dany kąt bryłowy) w danym obszarze.
- **Komórki bipolarne i ganglionowe** - sygnał wyjściowy jeżeli luminacja w danym punkcie jest znacząco wyższa od średniej.
- Każdy receptor w retinie jest częścia mniej więcej **kolistego pola odbiorczego**; pola te nakładają się.
- Podobnie jak z krawędzią - otoczenie powoduje inhibicję neuronu a fotoreceptor wzbudzenie (lub odwrotnie); funkcja meksykański kapelusz.

Perceptron - analogia biologiczna - struktura retiny

- David Marr podsumował wiedzę o ścieżce wzrokowej i podzielił proces na **3 etapy**.
  1. Mózg rozkłada obraz na cechy np. krawędzie.
  2. Z uzyskanych cech budowana jest interpretacja powierzchni, relacje głębi, grupowania.
  3. Pełna interpretacja objektów w polu widzenia (szkic pierwotny).
- Marr próbował wyjaśnić strukturę retiny z punktu widzenia zasobów obliczeniowych potrzebnych do widzenia - zaproponował że na pewnym etapie retina zamazuje obraz a następnie wydobywa informacje o kontraście.
- Zamazywanie - uśrednienie wartości danego piksela i jego sąsiadów; można użyć gausowskiego rozkładu wag.

Perceptron - analogia biologiczna - neokognitron

- Pojedyncze perceptrony - **detektory cech**.



Feature detector for the pattern T

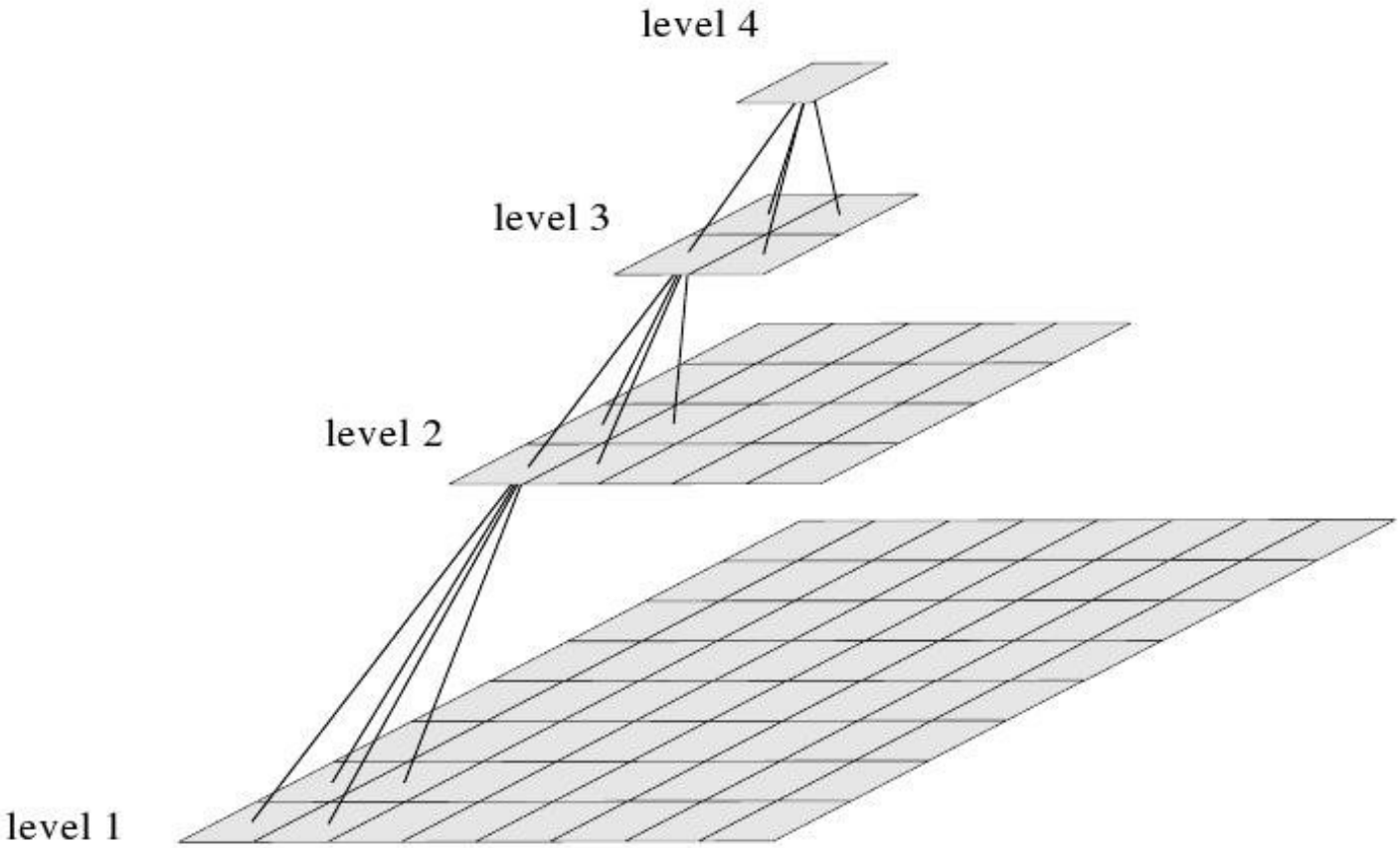
Perceptron - analogia biologiczna - neokognitron

- Rozpoznanie litery T: t pikseli jest czarne.
- Jeżeli brakuje piksela czarnego lub biały staje się czarny - wzbudzenie $t-1$.
- $t-1$ - próg perceptronu - prawidłowe działanie mimo szumu jednego piksela.
- Podobnie więszy szum - perceptron oblicza **podobieństwo** wzorca do wzorca idealnego a próg jest minimalnym podobieństwem jakie jest wymagane.
- **Problem** - wzorce muszą być znormalizowane - wycentrowanie (translacja i rotacja jest niedozwolona) oraz odpowiedni rozmiar.
- **Rozwiązanie** - rozpoznanie w kilku krokach.
  - Pismo ręczne - można próbować rozpoznać cechy jak obecność linii o pewnej orientacji niezależnie od ich położenia.

Perceptron - analogia biologiczna - neokognitron

- Kognitron i neokognitron (Fukushima) - naśladowanie struktury ludzkiej ścieżki wzrokowej.
- **Neokognitron** - transformuje obraz na ekranie do następnego wzmacniając pewne cechy aż do ostatecznej decyzji; rozdzielczość kolejnych ekranów może się zmieniać.
- Ogólna **struktura układu nerwowego** zaproponowana przez Fukushimę - wariant znanej w przetwarzaniu obrazów **architektury piramidalnej** o rozdzielczości obrazu redukowanej z płaszczyzny do płaszczyzny.
- Rysunek pokazuje piramidę o rozdzielczości redukowanej o czynnik 4.
- Obliczenie ustalające wartość danego piksela może być dowolne np. obliczenia progowe.
- Pola odbiorcze nie zachodzą na siebie.
- Architektury - studiowano jako **struktury danych** dla algorytmów równoległych.

Perceptron - analogia biologiczna - neokognitron

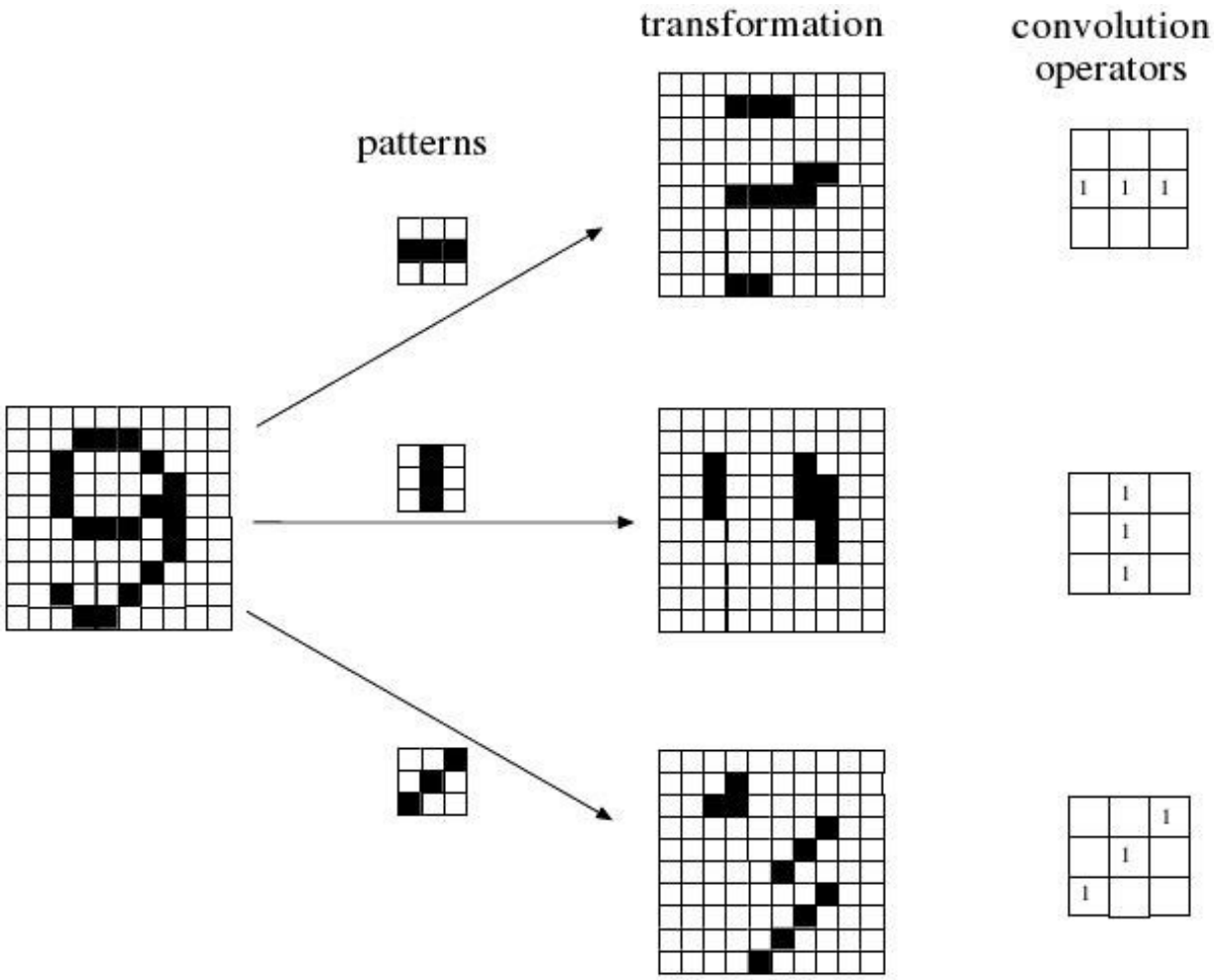


Pyramidal architecture for image processing

Perceptron - analogia biologiczna - neokognitron

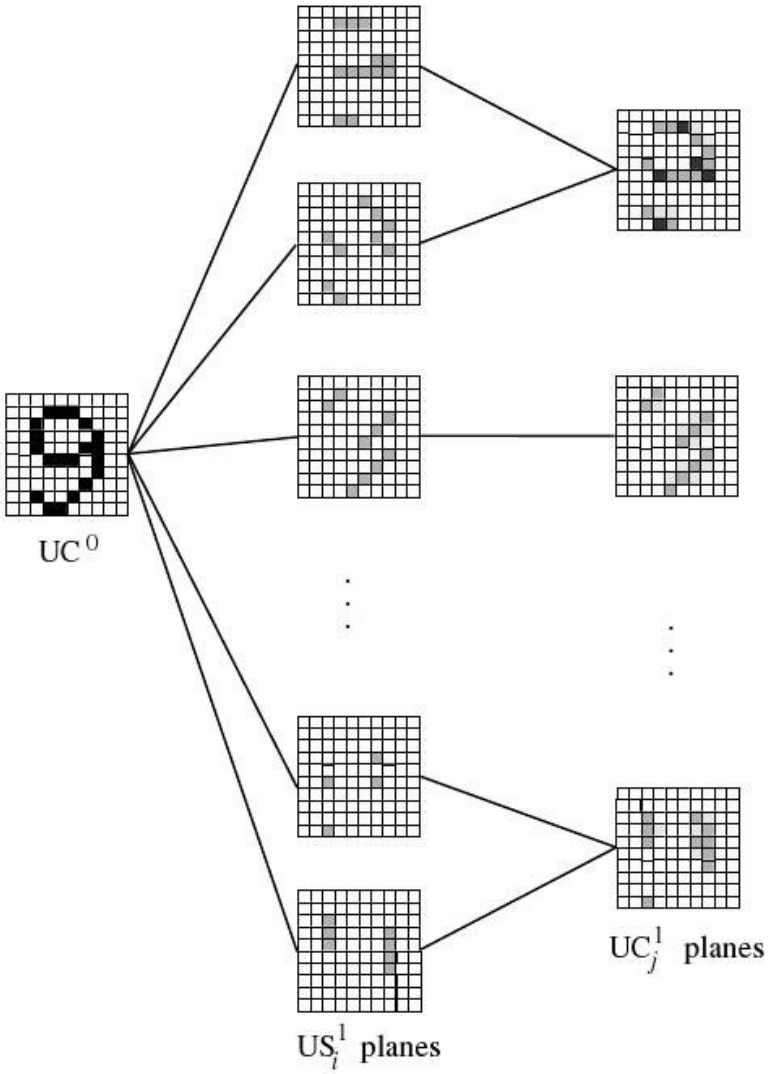
- **Neokognitron** - bardziej skomplikowana architektura; obraz transformowany jest z płaszczyzny pierwotnej na inne w poszukiwaniu specyficznych cech.
- Bit po bicie - decyzja czy biały czy czarny, na podstawie **identyfikacji wzorca** (piksel i otoczenie).
- Neokognitron Fukushimy - jednostki liniowe a nie perceptrony.
  - obliczają ważone wejście - interpretowane jako **korelacja** ze wzorcem, który jednostka ma zidentyfikować
  - czarno-białe wzorce $\Rightarrow$ odcienie szarości

Perceptron - analogia biologiczna - neokognitron



Feature extraction in the neocognitron

Perceptron - analogia biologiczna - neokognitron



The architecture of the neocognitron

Perceptron - analogia biologiczna - neokognitron

- Warstwa wejściowa UC^0.
- Konwersja do 12 obrazów $US_0^1,...,US_{11}^1$ o takiej samej rozdzielczości (indeks górny - numer warstwy).
- Powyższa konwersja - każdy piksel z UC^0 jest stowarzyszony z polem odbiorczym 3×3.
- Jedna płaszczyzna - jeden wzorzec.
- Płaszczyzny UC_j^1 - każdy piksel połączony z polem odbiorczym w jednej lub dwóch płaszczyznach US_i^1 - ich aktywacje nakładają się zamazując obraz - wzorzec staje się trochę szerszy (średnia ważona piksela i jego sąsiadów).
- Następnie płaszczyzny US_i^2 - połączone w tej samej pozycji do każdego z UC_j^1, rozdzielczość może być zmniejszona.
- Rysunek - rozmiary płaszczyzn użytych przez Fukushimę do rozpoznawania pisma ręcznego.
  - US i UC przemiennie aż do UC^4
  - klasyfikacja jako jedna z $0, .., 9$
  - znalezienie wag w procesie uczenia

Perceptron - analogia biologiczna - neokognitron

- Główna **zaleta** neokognitronu - tolerancja na przesuniecia i zniekształcenia (dzieki zamazywaniu obrazu).
- Układ ten jest bardzo wrażliwy na metodę uczenia i w rzeczywistości nie jest lepszy od innych prostych sieci.
- Badano podobne systemy a neokognitron jest **przykładem** klasy sieci opierających się na operatorach konwolucji i rozpoznawaniu wzorców w małych polach odbiorczych.

Perceptron - krzemowa retina

- Carver Mead (California Institute of Technology) - rozwija tzw. **inżynierię neuromorficzną** - budowa urządzeń emulujących odpowiedź ludzkich organów odpowiedzialnych za zmysły.
- **Retina silikonowa** - model 3 pierwszych warstw retiny: fotoreceptorów, komórek horyzontalnych i komórek bipolarnych.
  - komórki horyzontalne - sieć oporników,
  - fotoreceptory (na rysunku - czarne kropki) połączone są z 6 sąsiadami i wytwarzają różnicę potencjałów proporcjonalną do luminancji,
  - sieć oporników osiąga równowagę elektryczną gdy ustali się średnia różnica potencjałów,
  - neuron daje sygnał gdy różnica pomiędzy jego potencjałem a średnią osiągnie pewien próg.

Perceptron - krzemowa retina

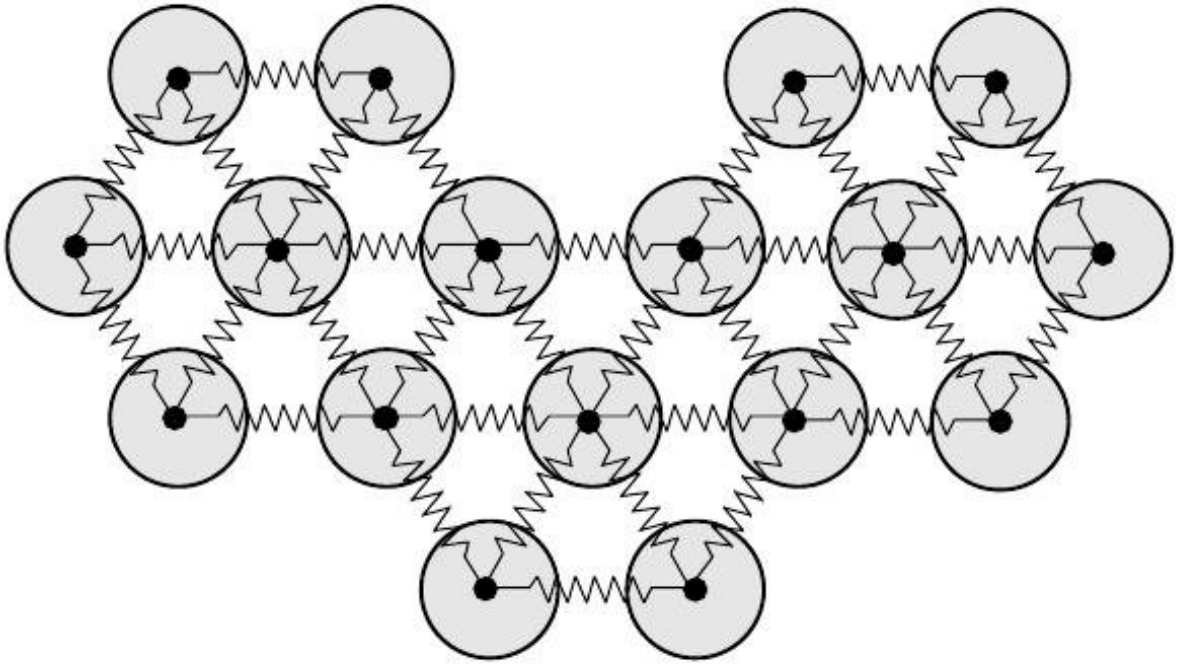


Diagram of a portion of the silicon retina

Perceptron - krzemowa retina

- Średni potencjał S jako natężenie światła H_i:

$$\begin{aligned} S &= \frac{1}{n}\sum_{i=1}^{n} S_i = \frac{1}{n}\sum_{i=1}^{n} \log H_i, \\ S &= \frac{1}{n}\log\left(H_1 H_2 \cdots H_n\right) = \log\left(H_1 H_2 \cdots H_n\right)^{1/n} \end{aligned}$$

- Zatem średni potencjał można przedstawić jako średnią geometryczną natężeń.
- Warunek na aktywację i-tego perceptronu o progu γ to

$$\log(H_i) - log(H_1 \cdots H_n)^{1/n} \geq \gamma,$$

lub

$$\log \frac{H_i}{(H_1 \cdots H_n)^{1/n}} \geq \gamma.$$

Perceptron - krzemowa retina

- Jednostka aktywuje się gdy różnica z tłem jest duża.
- W słoneczny dzień czarne litery w książce odbijają więcej fotonów niż biały papier w pomieszczeniu.
- Dzieki kompensacji luminacji tła można czytać zarówno w pomieszczeniu jak i w świetle słonecznym.

Perceptron - uwagi historyczne

- Perceptron - pierwsza sieć neuronowa produkowana komercyjnie (pierwsze prototypy używane były w laboratoriach).
- Frank Rosenblat - perceptron do rozwiązania pewnych problemów rozpozawania obrazów.
- Perceptron można uważać za pierwszy poważny model komórki nerwowej.
- Model powstał w latach 50-tych - wtedy też Hubel i Wiesel zbadali strukturę retiny.
- Początek lat 70-tych - architektura ludzkiego oka była dobrze poznana.
- Książka "Perceptrony" Minsky'ego i Paperta - wpływowa w społeczności badaczy sztucznej inteligencji (AI); mowi się że wpływała na finansowanie badań.
  - jedna z najlepiej napisanych,
  - krytykowana za podkreślanie nieobliczanlości a nie obliczalności.

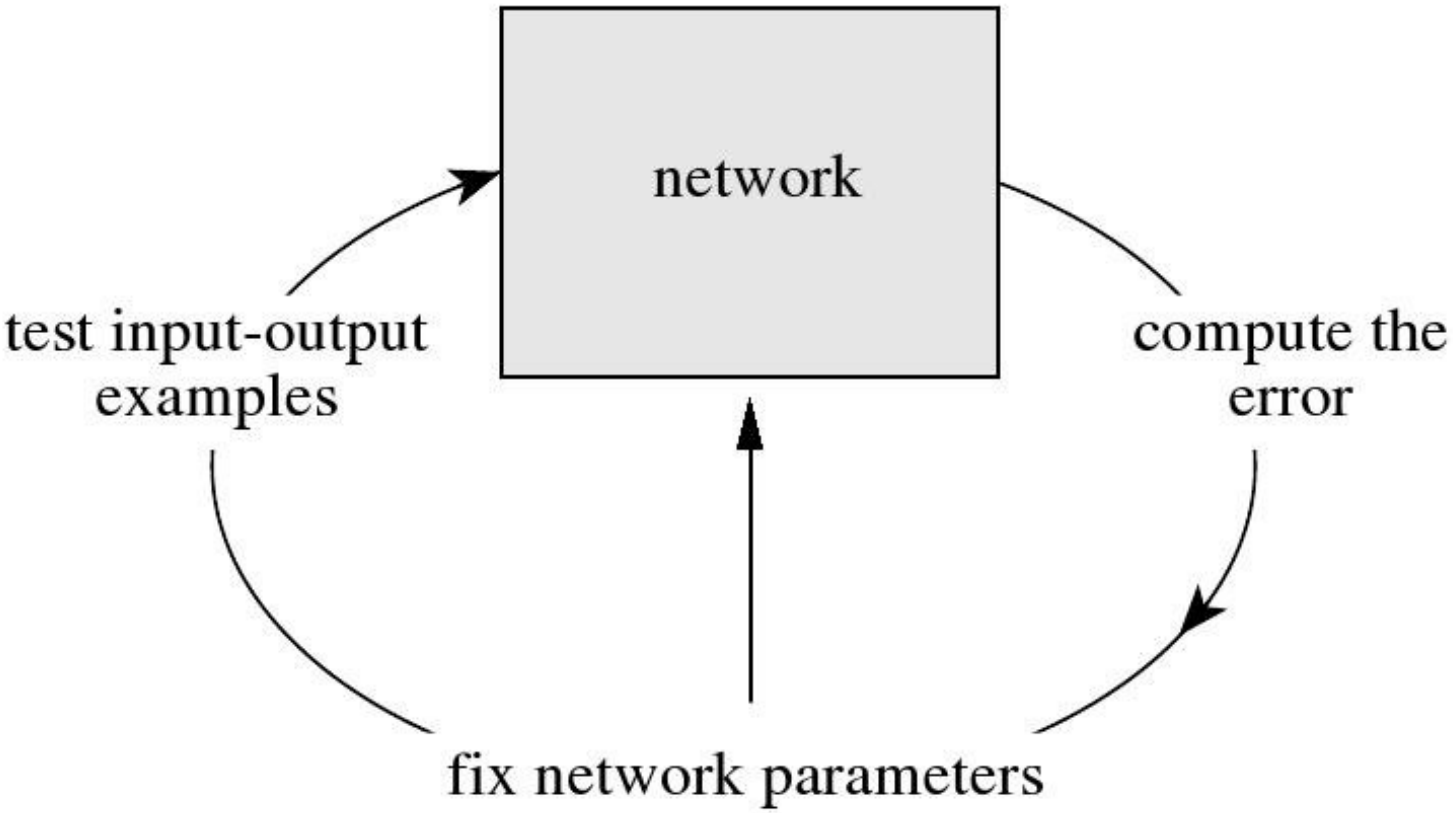
Perceptron - uwagi historyczne

- Operatory konwolucji - używane od wielu lat i są standardowymi metodami.
- Urządzenia typu krzemowej retiny są produkowane i mogą być używane w budowie robotów.
- Praktyczny cel - przywrócenie wzroku niewidomym ludziom jeżeli nerwy wzrokowe są nieuszkodzone.

Algorytm uczenia perceptronu

- Algorytm uczenia perceptronu - mechanizm znajdywania wag odpowiednich do danego zadania.
- Ogólnie **algorytm uczenia** to metoda adaptywna - następuje samoorganizacja.
- Można prezentować sieci przykłady par wejście-wyjście.
- Uczenie zawiera krok korekcyjny wykonywany aż sieć nauczy się dawać prawidłowe wyjście.
- Pewne przypadki - szukanie wag przez testowanie stochastycznie generowanych przykładów - nie następuje adaptacja wynikająca z poprzednich doświadczeń więc nie jest to prawdziwe uczenie.

Algorytm uczenia perceptronu

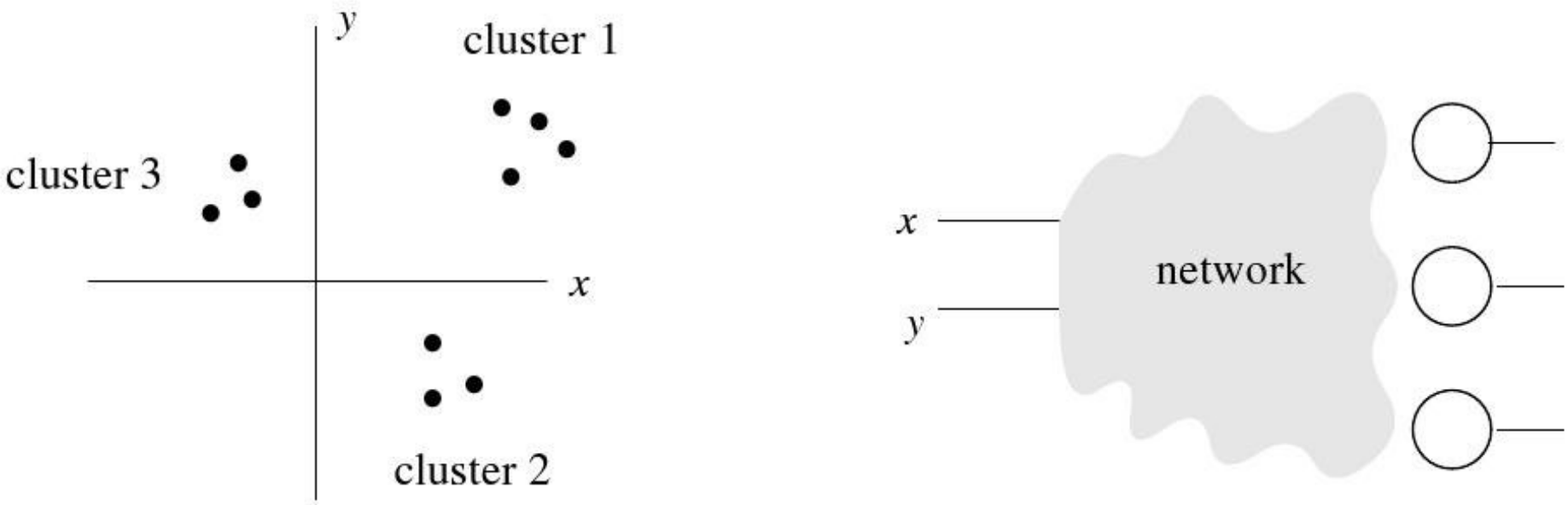


Learning process in a parametric system

Algorytm uczenia perceptronu - rodzaje algorytmów

1. **Uczenie nadzorowane** (nazywane tez uczeniem z nauczycielem - proces uczący zna odpowiedź)
   1. wektory wejściowe są prezentowane sieci
   2. odchylenie od oczekiwanej odpowiedzi mierzone
   3. wagi są korygowane w zależności od wielkości błędu w sposób określony przez algorytm
2. **Uczenie nienadzorowane** - dla danego wejścia dokładna odpowiedź nie jest znana.
   - Przykład - punkty w przestrzeni 2-wymiarowej mają być zaklasyfikowane do 3 klastrów - można zaprojektować odpowiednią sieć klasyfikującą o 3 liniach wyjściowych. Nie wiadomo który neuron wyspecjalizuje się dla danego kastra. W ogólności może nie być wiadomo ile jest klastrów.

Algorytm uczenia perceptronu



Three clusters and a classifier network

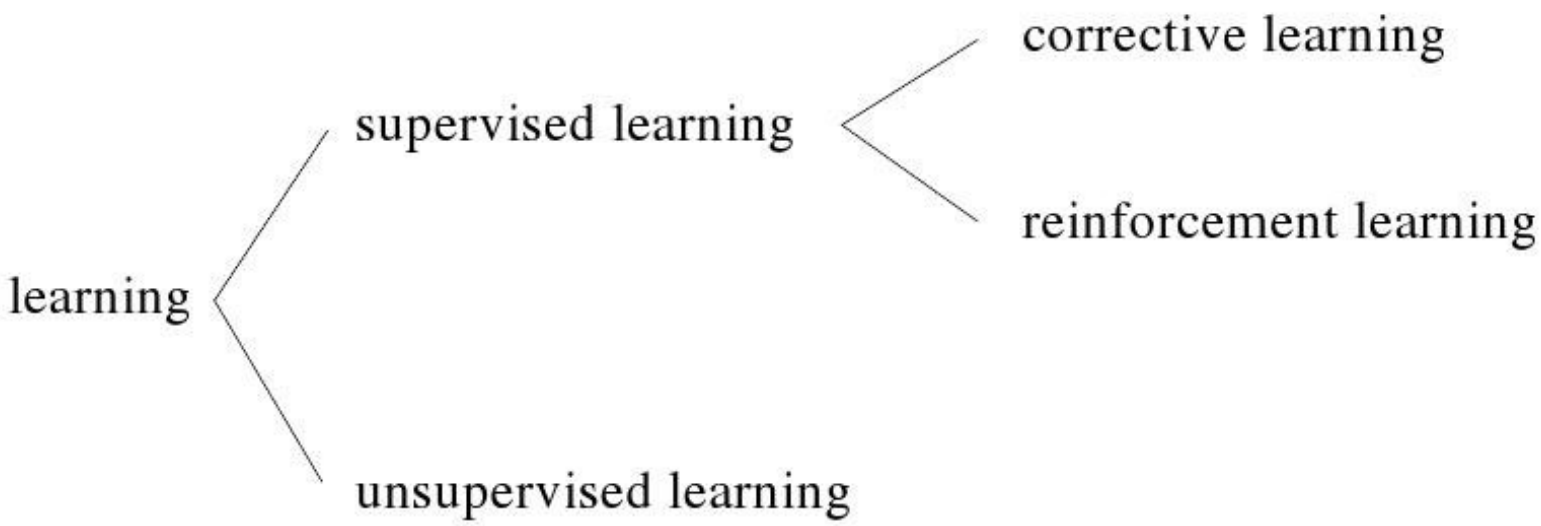
Algorytm uczenia perceptronu - rodzaje algorytmów

Uczenie nadzorowane można podzielić dalej na:

1. **Uczenie wymuszone** - po każdej prezentacji pary wejście-wyjście wiadomo jedynie czy wyjście sieci było prawidłowe czy nie. Do korekcji może być użyty tylko wektor wejścia.
2. **Uczenie przez korekcję błędu** - zarówno wielkość błędu jak i wektor wejściowy wyznaczają wielkość korekcji wag sieci. Często można próbować usunąć błąd w jednym kroku.

- Algorytm uczenia perceptoronu - uczenie nadzorowane z wymuszeniem (niektóre warianty używają korekcji błędu).

Algorytm uczenia perceptronu



Classes of learning algorithms

Algorytm uczenia perceptronu - notacja wektorowa

- Wektor wejściowy $(x_1, x_2, ..., x_n)$.
- θ jest progiem.
- $\mathbf{w} = (w_1, w_2, ..., w_n, w_{n+1})$ $(w_{n+1} = -\theta)$ - **rozszerzony wektor wag**.
- $(x_1, x_2, ..., x_n, 1)$ - **rozszerzony wektor wejścia**.
- Perceptron sprawdza czy $\mathbf{w} \cdot \mathbf{x} \geq \theta$ lub (w notacji rozszerzonej) $\mathbf{w} \cdot \mathbf{x} \geq 0$.
- W przypadku operatora $\wedge$ wejścia i wyjścia to (notacja zwykła i rozszerzona):
  $(0,0) \mapsto 0 \quad (0,0,1) \mapsto 0$
  $(0,1) \mapsto 0 \quad (0,1,1) \mapsto 0$
  $(1,0) \mapsto 0 \quad (1,0,1) \mapsto 0$
  $(1,1) \mapsto 1 \quad (1,1,1) \mapsto 1$
  Szukamy 3 wag (w_1, w_2, w_3).

Algorytm uczenia perceptronu - absolutna liniowa separowalność

- Do dowodu zbieżności algorytmu uczenia perceptronu zakłada się sprawdzanie nierówności ostrej $\mathbf{w} \cdot \mathbf{x} > 0$.
- Przypadki równości ostrej i nieostrej są równoważne jeżeli zbiór treningowy jest **skończony**.
- **Definicja.** Dwa zbiory punktów w przestrzeni n-wymiarowej A i B są nazywane **absolutnie liniowo separowalnymi** jeżeli istnieje $n+1$ liczb rzeczywistych $w_1, ..., w_{n+1}$ takich, że każdy punkt $(x_1, ..., x_n) \in A$ spełnia nierówność $\sum_{i=1}^{n} w_i x_i > w_{n+1}$ a każdy punkt $(x_1, ..., x_n) \in B$ spełnia nierówność $\sum_{i=1}^{n} w_i x_i < w_{n+1}$.
- Jeżeli perceptron może liniowo rozdzielić dwa skończone zbiory wektorów wejściowych, to jedynie małe dostosowanie wag jest potrzebne do otrzymania absolutniej liniowej separowalości. Wynika to z twierdzenia:
- **Twierdzenie.** Dwa liniowo separowalne skończone zbiory punktów A i B w przestrzeni n-wymiarowej są także absolutnie liniowo separowalne.

Algorytm uczenia perceptronu - absolutna liniowa separowalność

Dowód.

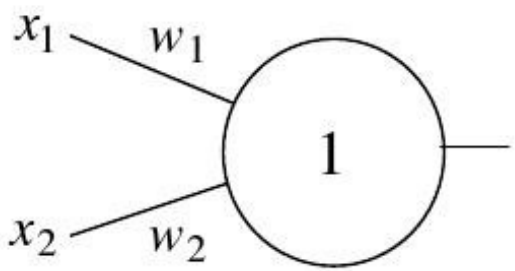
- Dwa zbiory są liniowo separowalne, tzn. istnieją wagi $w_1, ..., w_{n+1}$ takie, że $\sum_{i=1}^{n} w_i a_i \geq w_{n+1}, \ (a_1, ..., a_n) \in A$
  $\sum_{i=1}^{n} w_i b_i < w_{n+1}, \ (b_1, ..., b_n) \in B$
- Jeżeli $\varepsilon = \max\{\sum_{i=1}^{n} w_i b_i - w_{n+1} | (b_1, ..., b_n) \in B\}$. Z poprzedniego punktu: $\varepsilon < \varepsilon/2 < 0$.
- Niech $w' = w_{n+1} + \varepsilon/2$, dla wszystkich punktów ze zbioru A:
  $\sum_{i=1}^{n} w_i a_i - (w' - \varepsilon/2) \geq 0 \quad \Rightarrow \quad \sum_{i=1}^{n} w_i a_i - w' \geq -\varepsilon/2 > 0 \quad \Rightarrow$
  $\sum_{i=1}^{n} w_i a_i > w'$.
- Podobnie dla punktów zbioru B:
  $\sum_{i=1}^{n} w_i b_i - w_{n+1} = \sum_{i=1}^{n} w_i b_i - (w' - \varepsilon/2) \leq \varepsilon < 0 \quad \Rightarrow$
  $\sum_{i=1}^{n} w_i b_i - w' \leq \varepsilon/2 < 0 \quad \Rightarrow \quad \sum_{i=1}^{n} w_i b_i < w'$.

Algorytm uczenia perceptronu - powierzchnia błędu

- Zwykle pierwszym etapem w procesie uczenia sieci jest **losowe przypisanie wag** sieci a następnie poprawianie parametrów (każdy krok - sprawdzamy, czy lepsza liniowa separacja została osiągnięta).
- **Definicja. Otwarta (zamknięta) dodatnia półprzestrzeń** odpowiadająca n-wymiarowemu wektorowi wag $\mathbf{w}$ jest zbiór wszystkich punktów $\mathbf{x} \in \mathcal{R}^n$ dla których $\mathbf{w} \cdot \mathbf{x} > 0$ $(\mathbf{w} \cdot \mathbf{x} \geq 0)$.
  Otwarta (zamknięta) ujemna półprzestrzeń odpowiadająca n-wymiarowemu wektorowi wag $\mathbf{w}$ jest zbiór wszystkich punktów $\mathbf{x} \in \mathcal{R}^n$ dla których $\mathbf{w} \cdot \mathbf{x} < 0$ $(\mathbf{w} \cdot \mathbf{x} \leq 0)$.
- P, N - skończone zbiory punktów w przestrzni $\mathcal{R}^n$, które chcemy rozseparować liniowo. Szukamy odpowiedniego **wektora wag**.
- **Błąd** perceptronu o wektorze wag $\mathbf{w}$ = liczba nieprawidłowo zaklasyfikowanych punktów.
- Algorytm uczący musi minimalizować funkcję błędu $E(\mathbf{w})$.

Algorytm uczenia perceptronu - powierzchnia błędu

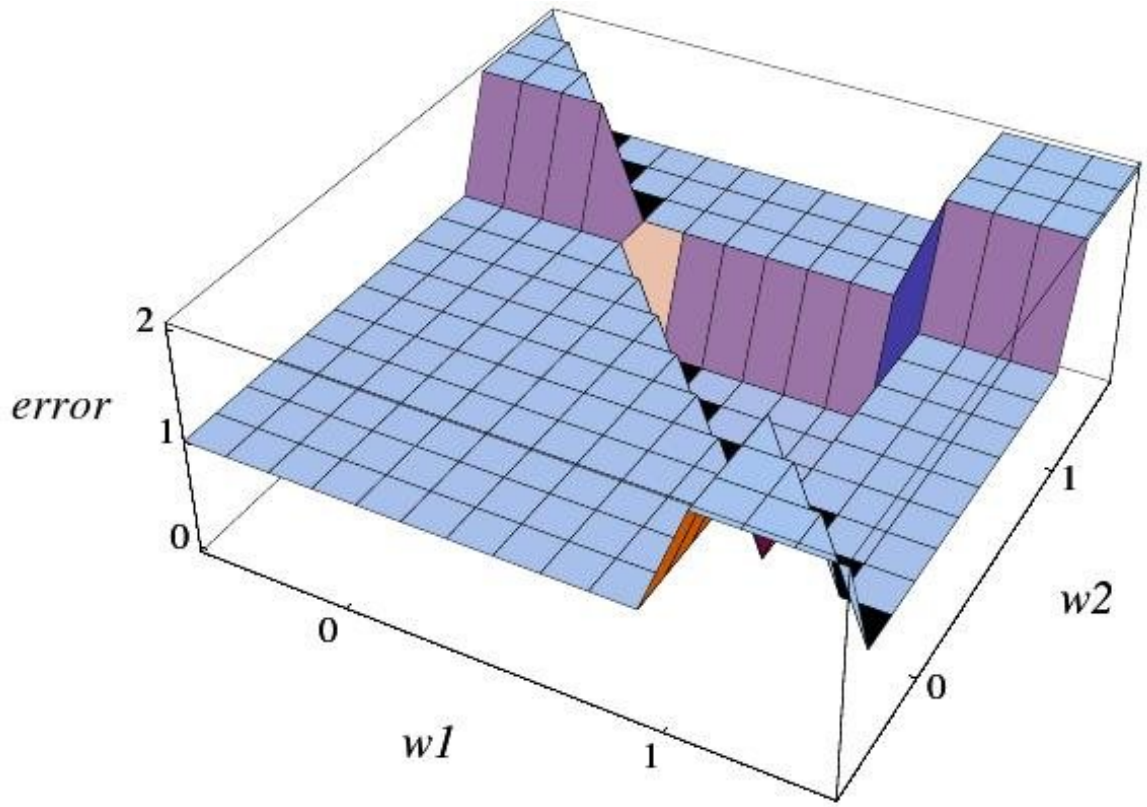
- Możliwa **strategia** - obliczamy błąd a następnie określamy kierunek w przestrzeni wag w którym należy przemieścić wektor.
- Można to zwizualizować w przestrzeni wag.
- Rozważmy perceptron o 2 wagach i progu $\theta = 1$. Szukamy wag które zamienią go w bramkę **AND**.



Perceptron with constant threshold

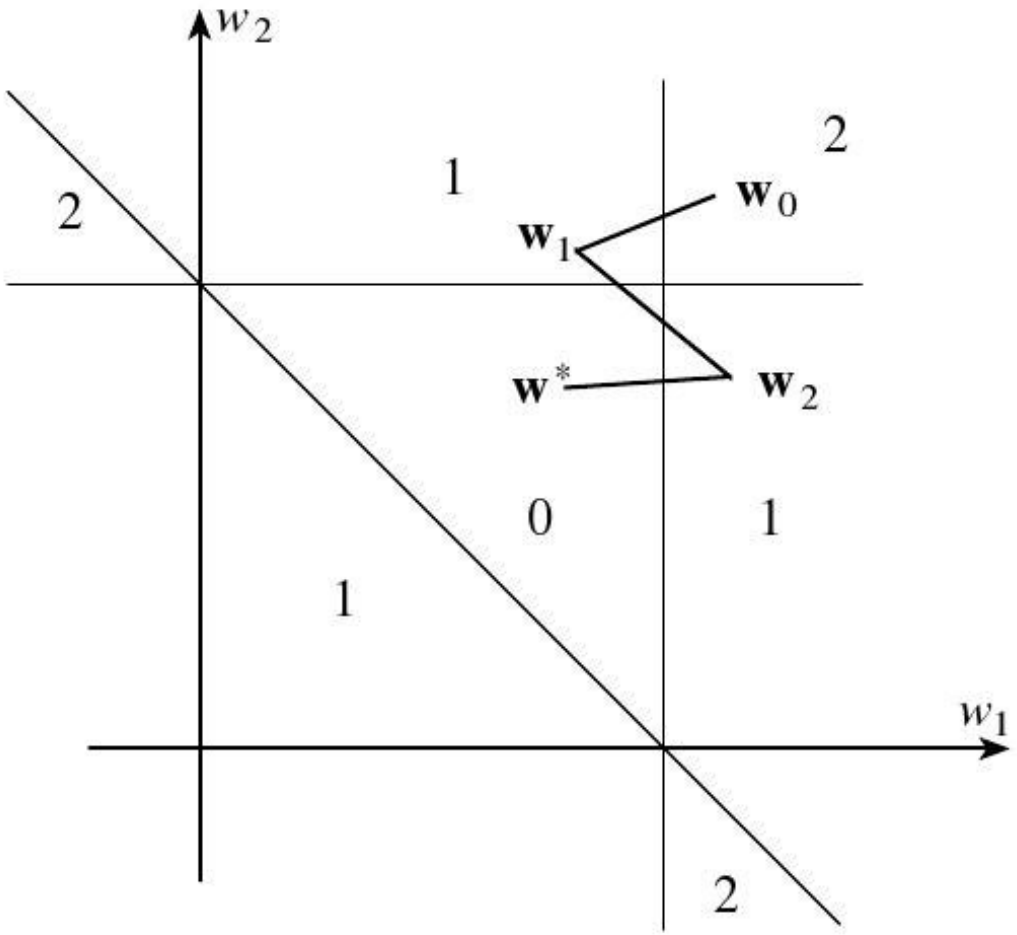
Algorytm uczenia perceptronu - powierzchnia błędu

- Rysunek pokazuje funkcję błędu dla wag z przedziału $[-0.5, 1.5]$.
- Rozwiązaniem jest trójkątny rejon w środku - algorytm powinien go znaleźć zaczynając od dowolnego punktu (w tym przypadku jest to możliwe z użyciem w każdym kroku lokalnych decyzji).



Algorytm uczenia perceptronu - powierzchnia błędu

- Rysunek z góry - rozwiązanie to trójkąt na poziomie zero.



Iteration steps to the region of minimal error

Algorytm uczenia perceptronu - powierzchnia błędu

- Szukanie funkcji $\wedge$ oznacza, że należy rozseparować absolutnie zbiory $N = \{(0,0),(1,0),(0,1)\}$, $P = \{(1,1)\}$.
- P - musi być zakwalifikowany do dodatniej półprzestrzeni a N do ujemnej.
- Potrzebne są 3 wagi $w_1, w_2, w_3 = -\theta$ i rozszerzenie wektora wejściowego o $x_3 = 1$, muszą być spełnione:
  $(0,0,1) \cdot (w_1, w_2, w_3) < 0$
  $(1,0,1) \cdot (w_1, w_2, w_3) < 0$
  $(0,1,1) \cdot (w_1, w_2, w_3) < 0$
  $(1,1,1) \cdot (w_1, w_2, w_3) > 0.$

Algorytm uczenia perceptronu - powierzchnia błędu

- W postaci macierzowej:

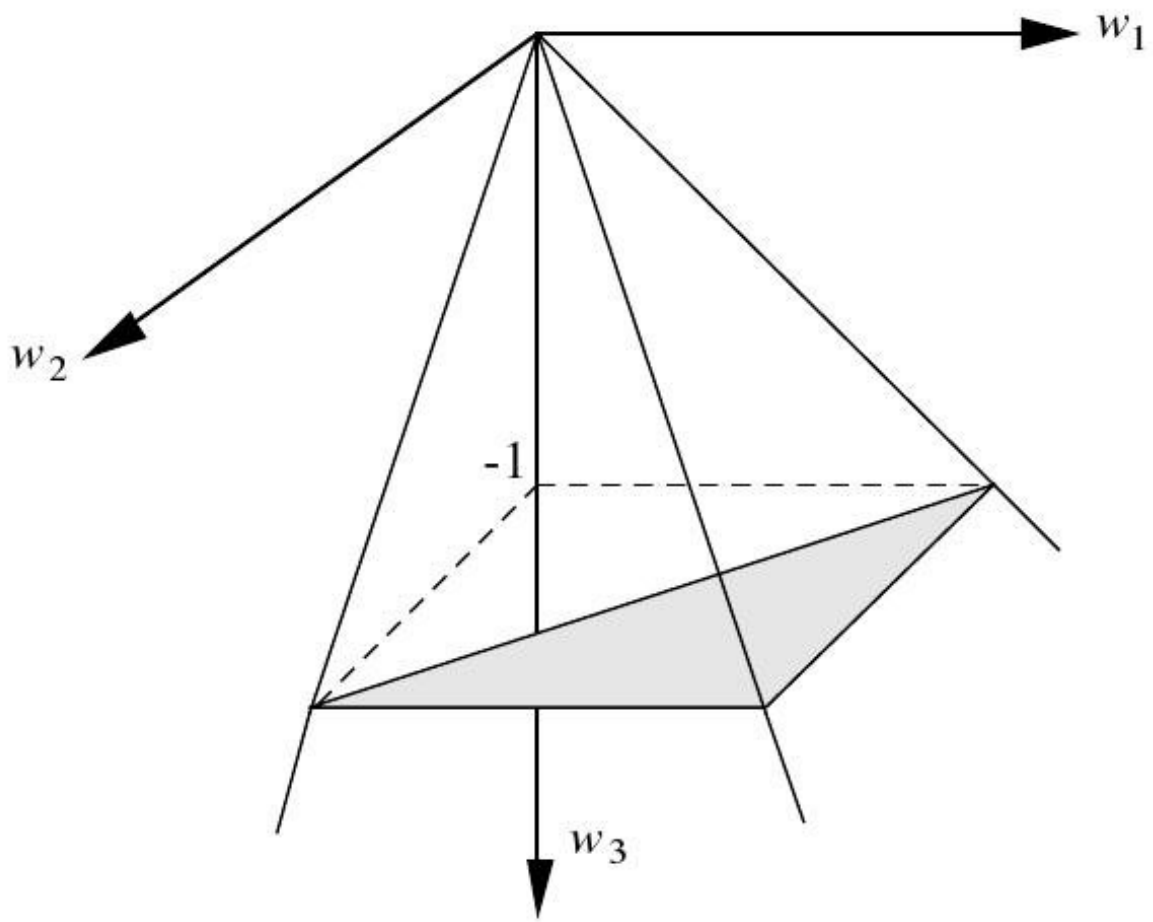
$$\begin{pmatrix} 0 & 0 & -1 \\ -1 & 0 & -1 \\ 0 & -1 & -1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} > \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

  lub $\mathbf{Aw} > 0$, gdzie $\mathbf{A} \in M_{4\times 3}$.

- Powyższe równanie wyznacza **punkty wnętrza wielościanu wypukłego** - ściany wyznaczone są przez odpowiednie równania powstałe z powyższych nierówności [$(0, 0, 1)$ itd. są wektorami normalnymi do wyznaczanych przez nie płaszczyzn].
- Każdy w wewnętrznych punktów jest rozwiązaniem problemu uczenia sieci.

Algorytm uczenia perceptronu - powierzchnia błędu

- Gdy próg (tzn. w_3) może się zmieniać, rozwiązanie dla funkcji $\wedge$ jest wnętrzem poniższego wielościanu:



Solution polytope for the AND function in weight space

Algorytm uczenia perceptronu - powierzchnia błędu

- Wielościan jest **nieograniczony** w kierunku ujemnych wartości w_3. Dla dowolnie dużego progu można znaleźć rozwiązanie.
- Do szukania punktów wnętrza można użyć metod **programowania liniowego**.

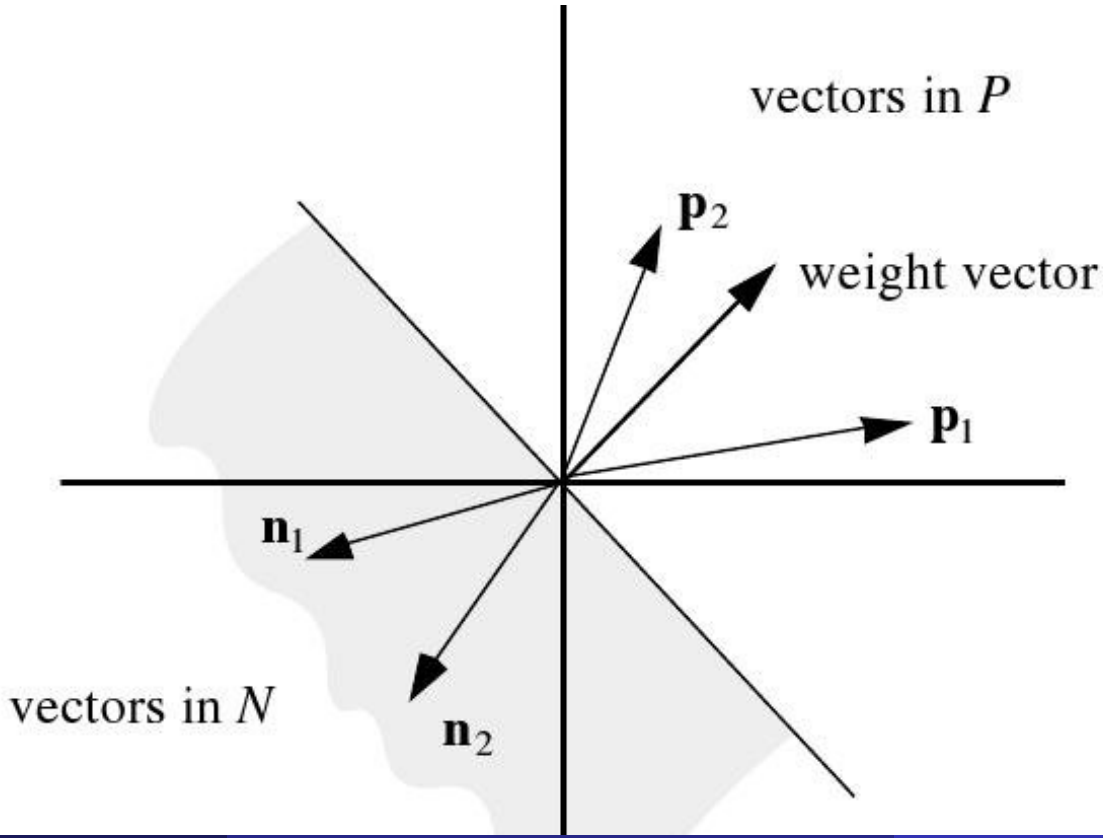
Algorytm uczenia perceptronu

- **Start**: Losowo generujemy wektor wag $\mathbf{w}_0$; t=0.
- **Test**: Wybieramy losowo wektor $\mathbf{x} \in P \cup N$:
  - jeżeli $\mathbf{x} \in P$ i $\mathbf{w}_t \cdot \mathbf{x} > 0$ idź do *test*,
  - jeżeli $\mathbf{x} \in P$ i $\mathbf{w}_t \cdot \mathbf{x} \leq 0$ idź do *dodaj*,
  - jeżeli $\mathbf{x} \in N$ i $\mathbf{w}_t \cdot \mathbf{x} < 0$ idź do *test*,
  - jeżeli $\mathbf{x} \in N$ i $\mathbf{w}_t \cdot \mathbf{x} \geq 0$ idź do *odejmij*.
- **Dodaj**: $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{x}$ i $t = t + 1$, idź do *test*.
- **Odejmij**: $\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{x}$ i $t = t + 1$, idź do *test*.

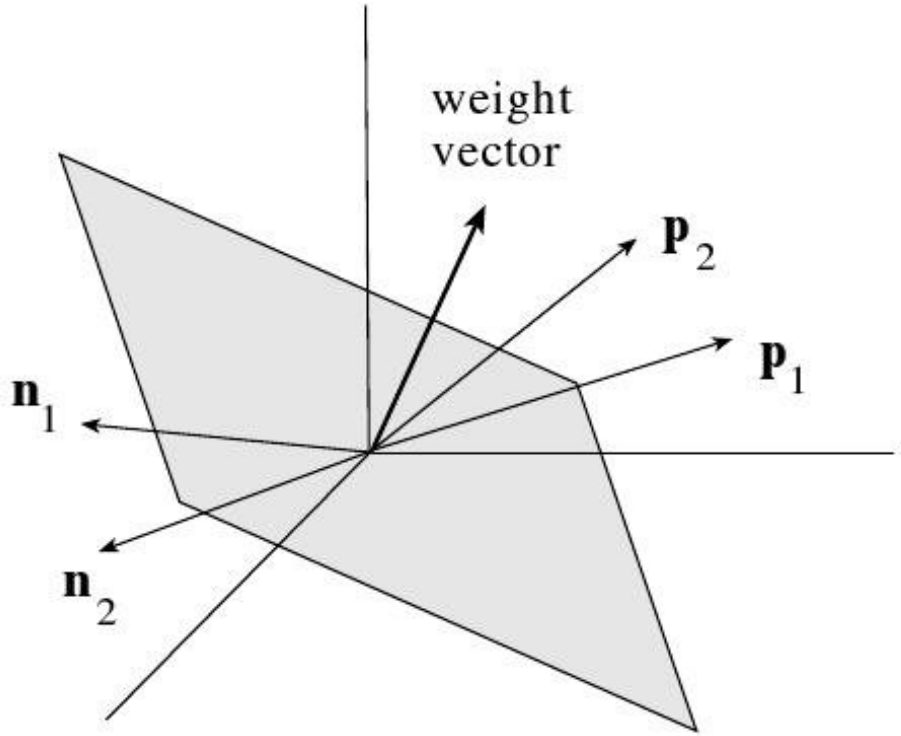
- Algorytm koryguje wektor wag kiedykolwiek nie jest zaklasyfikowany prawidłowo.
- $\mathbf{w}_{t+1} \cdot \mathbf{x} = \mathbf{w}_t \cdot \mathbf{x} + \mathbf{x} \cdot \mathbf{x} > \mathbf{w}_t \cdot \mathbf{x}$, bo $\mathbf{x} \cdot \mathbf{x} > 0$.
- **Twierdzenie o zbieżności** - dla skończonych zbiorów P i N wektor wag będzie aktualizowany skończoną ilość razy.

Algorytm uczenia perceptronu - wizualizacja geometryczna

- Są dwie alternatywne metody wizualizacji procesu uczenia: w przestrzeni wejść lub w rozszerzonej przestrzeni wejść (wtedy punkty separujemy liniowo przechodząc przez środek układu współrzędnych: $w_1 x_1 + w_2 x_2 + w_3 = 0$).
- $\mathbf{w} = (w_1, w_2, w_3)$ - wektor normalny do płaszczyzny.



Algorytm uczenia perceptronu - wizualizacja geometryczna

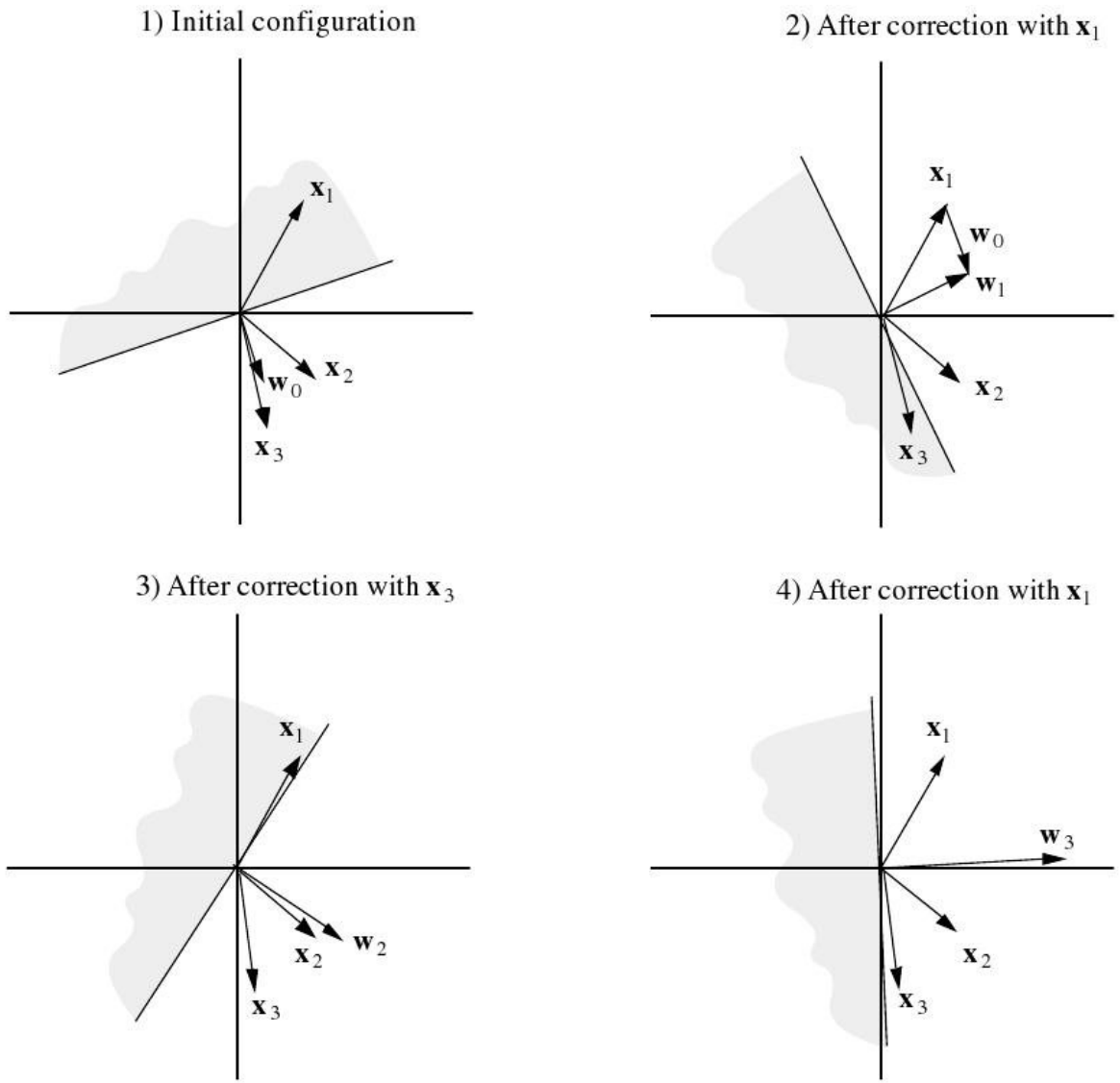


Visualization in extended input space

Algorytm uczenia perceptronu - wizualizacja geometryczna

- Należy znaleźć wagi takie, że $\mathbf{w} \cdot \mathbf{x} > 0$ dla $\mathbf{x} \in P$ oraz $\mathbf{w} \cdot \mathbf{x} < 0$ dla $\mathbf{x} \in N$.
- Algorytm dodaje/odejmuje $\mathbf{x}$ od $\mathbf{w}$ powodując zbliżenie/oddalenie (pomniejszenie/powiększenie kąta) się wektora wag od danego $\mathbf{x}$.
- Można zacząć od wektora będącego średnią dodatnich wektorów wejściowych minus średnią ujemnych wektorów wejściowych.
- Nie normalizujemy wektora wag, zatem za każdym razem $\mathbf{w} \pm \mathbf{x}$ oznacza inną rotację.
- Jeżeli $\mathbf{x} \in P$ i $\|x\| \gg \|w\|$ to nowy wektor wag $\mathbf{w} + \mathbf{x}$ jest prawie równy $\mathbf{x}$.
- Na poniższym przykładzie można zaobserwować zwiększanie się długości $\mathbf{x}$ oraz to, że każda korekcja wag przybliża wektor do danago $\mathbf{x}$ powodując coraz mniejszą zmianę kąta:

Algorytm uczenia perceptronu - wizualizacja geometryczna



Convergence behavior of the learning algorithm

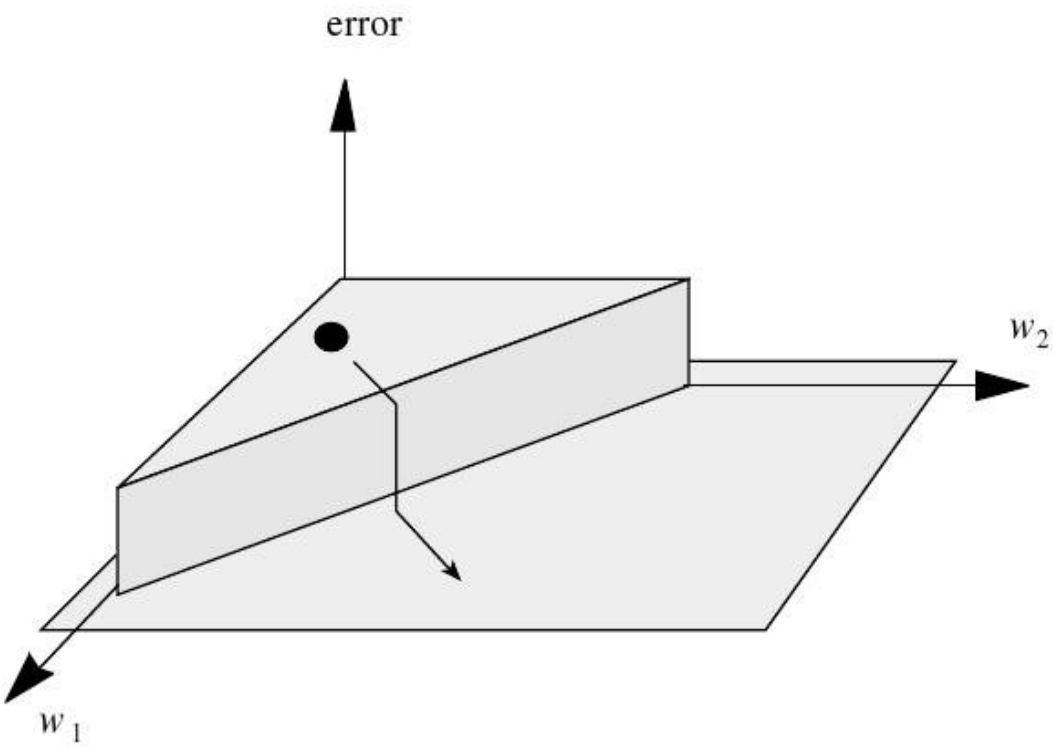
Algorytm uczenia perceptronu - wizualizacja geometryczna

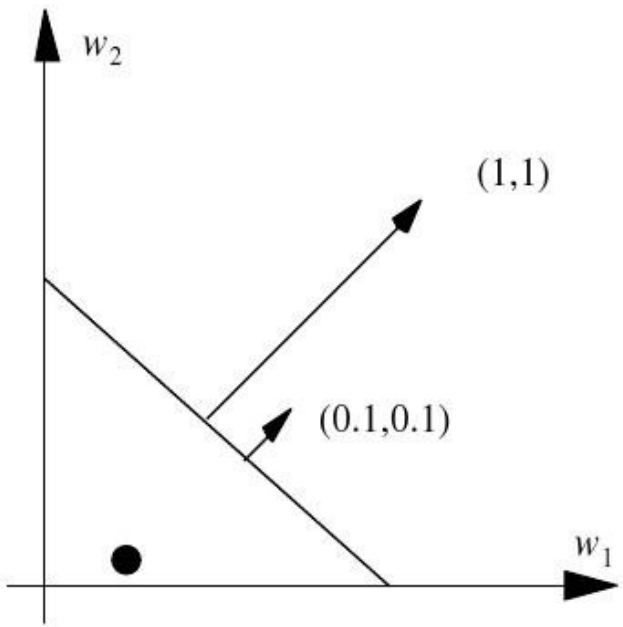
- **Szybkość uczenia** = szybkość zmiany wektora **w** - maleje w czasie i jeżeli kontynuujemy uczenie po uzyskaniu separacji zbliża się do zera.
- Wiele algorytmów uczenia posiada **stałą uczenia**, która dąży do zera w procesie uczenia.
- Algorytm uczenia perceptronu dostarcza zatem takiej stałej, która wyznacza też plastyczność sieci.

Algorytm uczenia perceptronu - przyspieszanie zbieżności

- Liczba iteracji może być bardzo duża - wektory mogą być nieznormalizowane lub niekorzystnie rozłożone.
- Istnieją szybsze metody znajdowania wektora wag.
- Dynamika uczenia perceptronu na przykładzie funkcji $\vee$:
  - $(1, 1)$ na wejściu musi dać 1, tzn. $w_1 + w_2 \geq 1$,
  - na rysunku przeciwprostokątna trójkąta to $w_1 + w_2 = 1$,
  - dodając do $\mathbf{x}$ wektor $(1, 1)$ otrzymujemy rozwiązanie w jednym kroku,
  - dodając do $\mathbf{x}$ wektor $(0.1, 0.1)$ potrzeba kilku kroków.

Algorytm uczenia perceptronu - przyspieszanie zbieżności





A step on the error surface

Algorytm uczenia perceptronu - przyspieszanie zbieżności

- Jeżeli w iteracji t wektor wejściowy $\mathbf{x} \in P$ został zakwalifikowany błędnie, tzn. $\mathbf{w}_t \cdot \mathbf{x} \leq 0$, błąd definiujemy następująco

$$\delta = -\mathbf{w}_t \cdot \mathbf{x}.$$

- Definiując nowe wagi jako

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \frac{\delta + \varepsilon}{\|\mathbf{x}\|^2}\mathbf{x},$$

gdzie ε to mała liczba rzeczywista dodatnia.

- Tak zdefiniowana waga klasyfikuje wektor $\mathbf{x}$ prawidłowo

$$\mathbf{w}_{t+1} \cdot \mathbf{x} = \left(\mathbf{w}_t + \frac{\delta + \varepsilon}{\|\mathbf{x}\|^2}\mathbf{x}\right) \cdot \mathbf{x} = \mathbf{w}_t \cdot \mathbf{x} + (\delta + \varepsilon) = -\delta + \delta + \varepsilon = \varepsilon > 0.$$

Algorytm uczenia perceptronu - przyspieszanie zbieżności

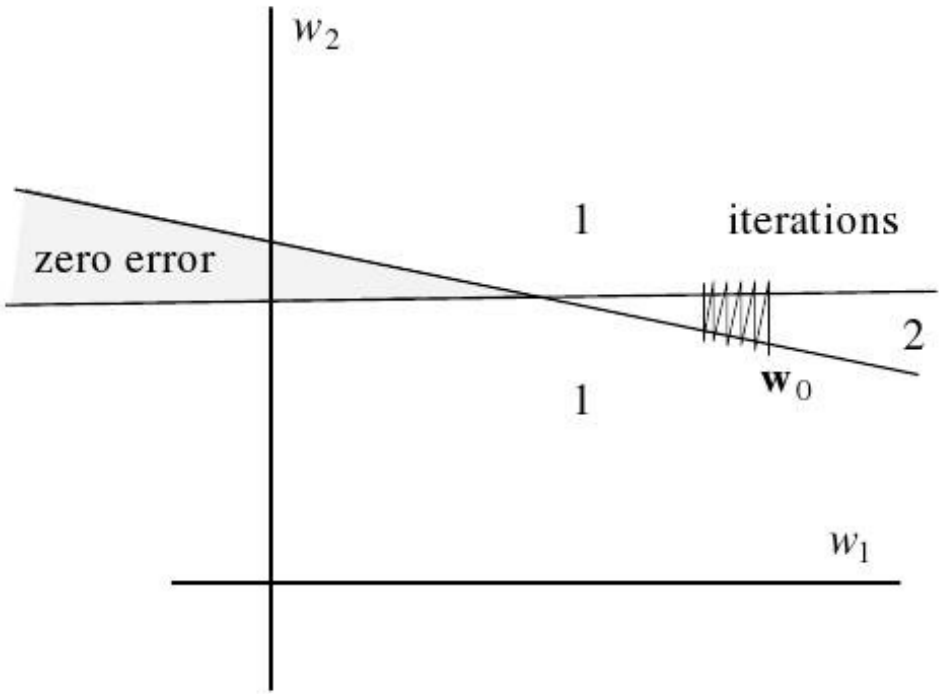
- Dzięki ε nowy wektor jest prawidłowo klasyfikowany w jednym kroku w minimalny sposób.
- Ważne jest aby ε by nie przeskoczyć to obszaru o wyższym błędzie.
- Gdy $\mathbf{x} \in N$ postępuje się analogicznie używając $\delta - \varepsilon$.
- Algorytm ten to przykład, to uczenie przez korekcję błędu.
- Istnieje wariant ze stałą proporcjonalności dążącą do zera w procesie uczenia $\gamma(\delta + \varepsilon)\mathbf{x}$.

Algorytm uczenia perceptronu - przyspieszanie zbieżności

- Jeżeli zbiór treningowy nie jest liniowo separowalny algorytm uczenia perceptronu nie będzie mógł sie zakończyć.
- Jednak może wystarczyć znalezienie separacji liniowej prawidłowo możliwie największej liczby wektorów.
- Odpowiedni wariant algorytmu uczenia perceptronu przechowuje najlepszy w danej chwili wektor wag (w "kieszonce") i nadal stara się znaleźć lepszy, który jeżeli zostanie znaleziony zastępuje poprzedni.
- Algorytm "kieszonkowy":
  1. Zainicjalizuj wektor $\mathbf{w}$ losowo. Zdefiniuj przechowywany wektor $\mathbf{w}_s = \mathbf{w}$. Ustaw historię h_s wektora $\mathbf{w}_s$ na zero.
  2. Aktualizuj $\mathbf{w}$ używając pojedynczego kroku algorytmu uczenia perceptronu. Aktualizuj liczbe h wektorów testowanych z sukcesem (cykl = wszystkie $\mathbf{x}$). Jeżeli $h > h_s$ podstaw $\mathbf{w}$ za $\mathbf{w}_s$ oraz h za h_s.
  3. Algorytm ten czasem może zamienić wektor lepszy na gorszy, bo rozważana jest tylko informacja z ostatniego zbioru przykładów.
  4. Jeżeli zbiór treningowy jest skończony i wagi oraz wektory są wymierne można pokazać, że algorytm ten **dąży do optymalnego rozwiązania** z prawdopodobieństwem 1.

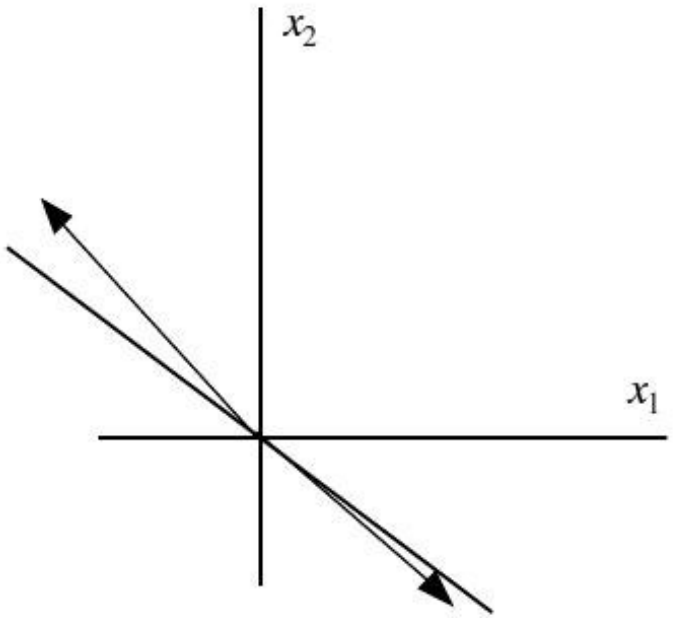
Algorytm uczenia perceptronu - złożoność uczenia perceptronu

- Algorytm wybiera kierunek w przestrzeni wag bazując na ostatniej nieprawidłowej klasyfikacji wektora testowego.
- Nie ma **globalnej informacji** o funkcji błędu.
- Czasem prowadzi to do bardzo wykładniczo rosnącej liczby aktualizacji wektora wag:



Algorytm uczenia perceptronu - złożoność uczenia perceptronu

- Zalety wizualizacji - poniższy rysunek pokazuje problem a poprzedni wyjaśnia dlaczego jest trudny do rozwiązania.
- Obydwa wektory należy zaklasyfikować do dodatniej półprzestrzeni - trzeba obracać prostą o bardzo mały kąt.



Worst case for perceptron learning (input space)

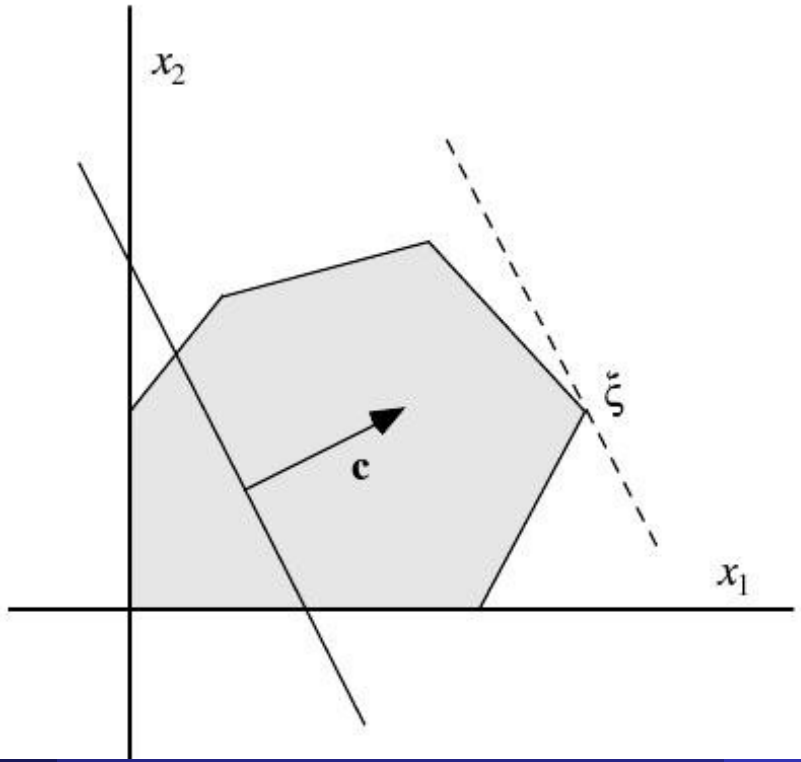
Algorytm uczenia perceptronu - programowanie liniowe

- Szukamy wnętrza wielościanu - zatem musza istnieć punkty należące do wnętrza.
- **Programowanie liniowe nadaje** się dla takiego zagadnienia. Zostało ono rozwinięte dla rozwiązania następującego problemu:
  - Mając dany zbiór n zmiennych $x_1, x_2, ..., x_n$ należy zminimalizować lub zmaksymalizować funkcję $c_1x_1 + c_2x_2 + ... + c_nx_n$.
  - Zmienne te muszą dodatkowo spełniać warunki:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n &\leq b_2 \\ &... \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n &\leq b_m \end{aligned}$$

Algorytm uczenia perceptronu - programowanie liniowe

- **Algorytm simpleks** - sprawdza kolejne wierzchołki, poruszając się w kierunku wzrastającej wartości optymalizowanej funkcji.
- **Kryterium** przejścia do kolejnego wierzchołka jest rzut gradientu optymalizowanej funkcji na ściany wielościanu.
- **Przy dużej liczbie wierzchołków** lepiej jest poruszać się przez środek wielościanu.



Algorytm uczenia perceptronu - programowanie liniowe

- Problem znalezienia punktu wnętrza można przekształcić w zagadnienie programowania liniowego.
- Wymagane nierówności można przedstawić w postaci macierzowej $\mathbf{Ax} \leq \mathbf{b}$.
- Dla uproszczenia zakładamy $\mathbf{b} \geq \mathbf{0}$ oraz $\mathbf{x} \geq \mathbf{0}$.
- Wprowadzając m-wymiarowy wektor $\mathbf{y}$ dodatkowych zmiennych, można napisać

$$\mathbf{Ax} + \mathbf{Iy} = \mathbf{b}.$$

- Szukamy rozwiązania zagadnienia minimalizacji

$$\min \left\{ \sum_{i=1}^{m} y_i | \mathbf{Ax} + \mathbf{Iy} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}, \mathbf{y} \geq \mathbf{0} \right\}.$$

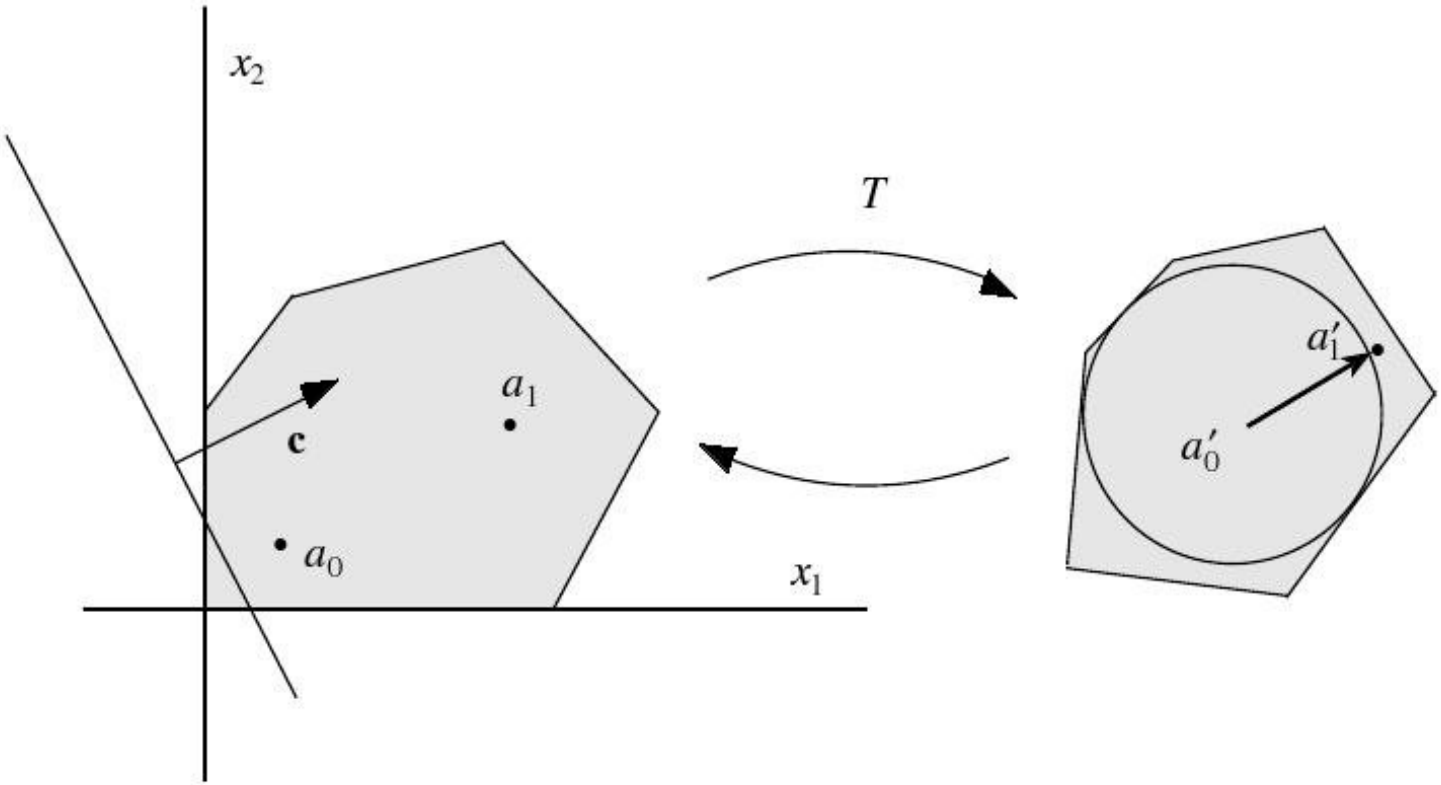
- Początkowe rozwiązanie spełniające te warunki, to $\mathbf{x} = \mathbf{0}$, $\mathbf{y} = \mathbf{0}$. Zaczynając od niego **szukamy minimum**.

Algorytm uczenia perceptronu - programowanie liniowe

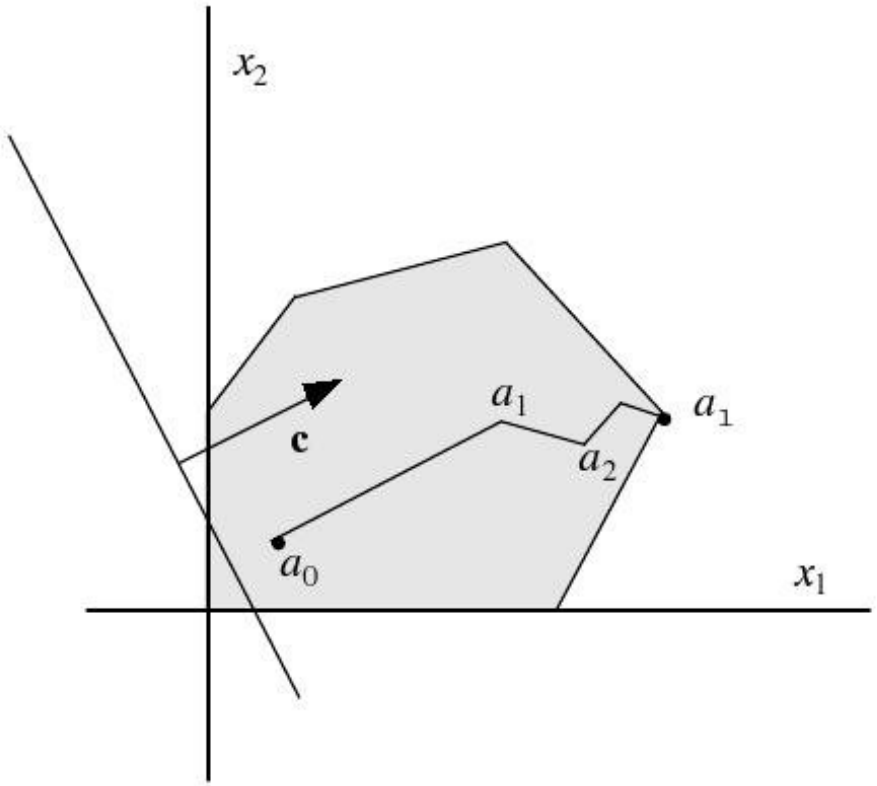
- Jeżeli minimum jest ujemne problem **nie ma rozwiązania**, gdyż $\mathbf{Ax} > \mathbf{b}$ (zbiór $\mathbf{Ax} \leq \mathbf{b}$ jest pusty).
- Można ominąć warunki $\mathbf{x} \geq \mathbf{0}$, $\mathbf{y} \geq \mathbf{0}$ - istnieje **transformacja** ogólnego problemu do przedstawionej tu postaci kanonicznej.
- Szukanie wektora wag dla **perceptronu** to szukanie punktów wewnętrznych wielościanu wypukłego, to jego uczenie może się odbywać także w ten sposób.
- Jeżeli zbiory wektorów **nie są liniowo separowalne** algorytm programowania liniowego może to wykryć.
- Algorytm uczenia perceptronu **nie** jest najefektywniejsza metodą uczenia (wiemy, że zdarzają się trudne przypadki!).
- W teorii programowania liniowego istnieją algorytmy potrzebujące **wielomianowej liczby iteracji i znajdujące optymalne rozwiązanie lub wykrywające, że takowe nie istnieje**.

Algorytm uczenia perceptronu - programowanie liniowe

- **Algorytm Karmarkara** (1984) - przykład szybkiego algorytmu (czas wielomianowy).
- Startuje od punktu wewnętrznego i stosuje metodę najszybszego spadku (w przypadku maksymalizacji) upewniając się, że nie wyjdzie poza dozwolony rejon.



Algorytm uczenia perceptronu - programowanie liniowe



Example of a search path for Karmarkar's algorithm

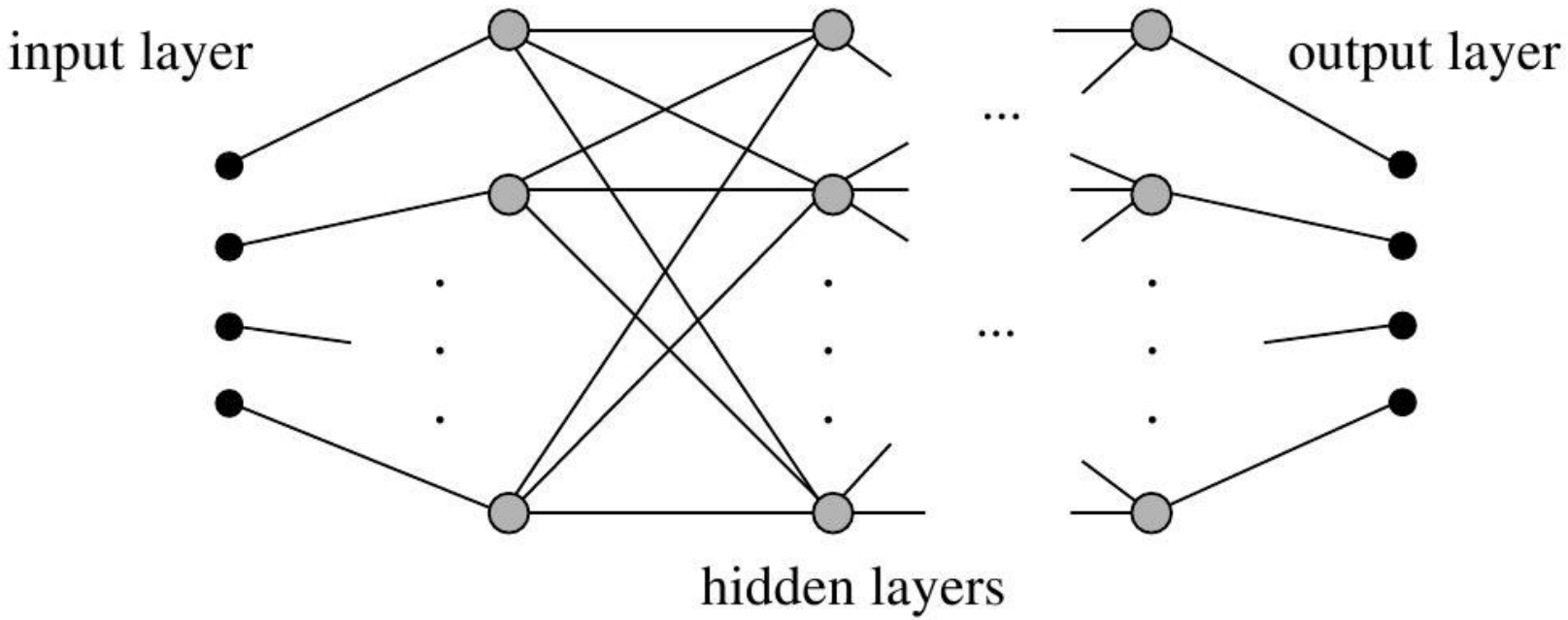
Sieci jedno - i dwuwarstwowe

- Przeanalizowaliśmy własności obliczeniowe pojedynczych elementów progowych.
- Rozważymy sieci o strukturze warstwowej - warstwy neuronów.
- Architektura sieci - jednostki i ich połączenia.
  - funkcja integrująca $\Psi : \mathbb{R}^n \to \mathbb{R}$ - oblicza całkowite wzbudzenie.
  - funkcja aktywacji $\Phi : \mathbb{R} \to \mathbb{R}$
  - może być $\Phi : \mathbb{R} \to [0, 1]$; Ψ nie musi być sumą - sieć może wtedy obliczyć trudne funkcje używając mniej neuronów.

Sieci jedno - i dwuwarstwowe - architektura warstwowa

- **Definicja. Architektura sieci** jest to czwórka (I, N, O, E) składająca się ze zbioru wejść I, zbioru jednostek obliczeniowych N, zbioru wyjść O oraz zbioru skierowanych krawędzi E. **Krawędź skierowana** to trójka (u, v, w) taka, że $u \in I \cup N$, $v \in N \cup O$, $w \in \mathbb{R}$.
- Wejścia i wyjścia nie wykonują obliczeń ale traktujemy je jak neurony.
- Wszystkie krawędzie mają wagi.
- **Architektura warstwowa** $N = N_1 \cup ... \cup N_l$ (N_i - **warstwa** sieci) tak, że połączenia z N_1 idą do N_2, z N_2 do N_3 itd.
- Tylko N_1 są połączone z wejściami, N_l tylko z wyjściami (nazywamy je jednostkami wyjściowymi sieci)
- Wejścia tworzą **warstwę wejść** a wyjścia **warstwę wyjść**; warstwy bez połączeń do wyjścia to **warstwy ukryte**.

Sieci jedno - i dwuwarstwowe - architektura warstwowa



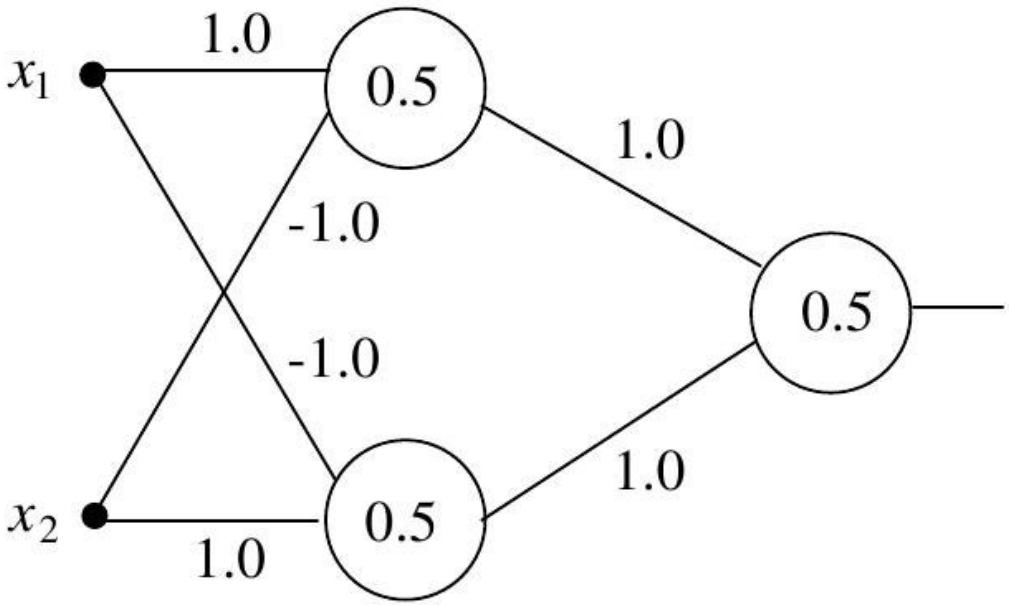
A generic layered architecture

Sieci jedno - i dwuwarstwowe - architektura warstwowa

- Zwykle jednostki w warstwie nie są ze sobą połączone a wyjścia są pomijane na rysunku.
- Jednostki z sąsiadujących warstw są połączone każda z każdą; liczba wag to mn (ilości neuronów: $N_i = m$, $N_{i+1} = n$).
- Dużo połączeń - powstaje zagadnienie ich minimalizacji.

Sieci jedno - i dwuwarstwowe - problem XOR

- XOR pozwala przeanalizować własności sieci 1- i 2-warstwowych.
- Pojedynczy perceptron nie może obliczyć tej funkcji ale sieć dwuwarstwowa tak.
- Sieć **3-warstwowa** obliczająca XOR (liczymy warstwę wejściową a wyjściową nie):



A three-layered network for the computation of XOR

Sieci jedno - i dwuwarstwowe - problem XOR

- Są tu 3 jednostki $x_1 \wedge \neg x_2$, $\neg x_1 \wedge x_2$: $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$.
- **Pytanie** - ile istnieje różnych rozwiązań problemu XOR taką siecią? Dokładniej: ile wyrażeń na funkcję XOR można napisać z użyciem 3 z 14 możliwych funkcji Boole'a dwóch zmiennych (XOR i jego zaprzeczenia nie liczymy).
- Można wypisać możliwe funkcje boolowskie.
  - Notacja - są 4 możliwe wejścia, które szereguje się (1,1), (0,1), (1,0), (0,0) i indeksem przy funkcji f oznacza się wartość funkcji dla kolejnych wejść:

Sieci jedno - i dwuwarstwowe - problem XOR

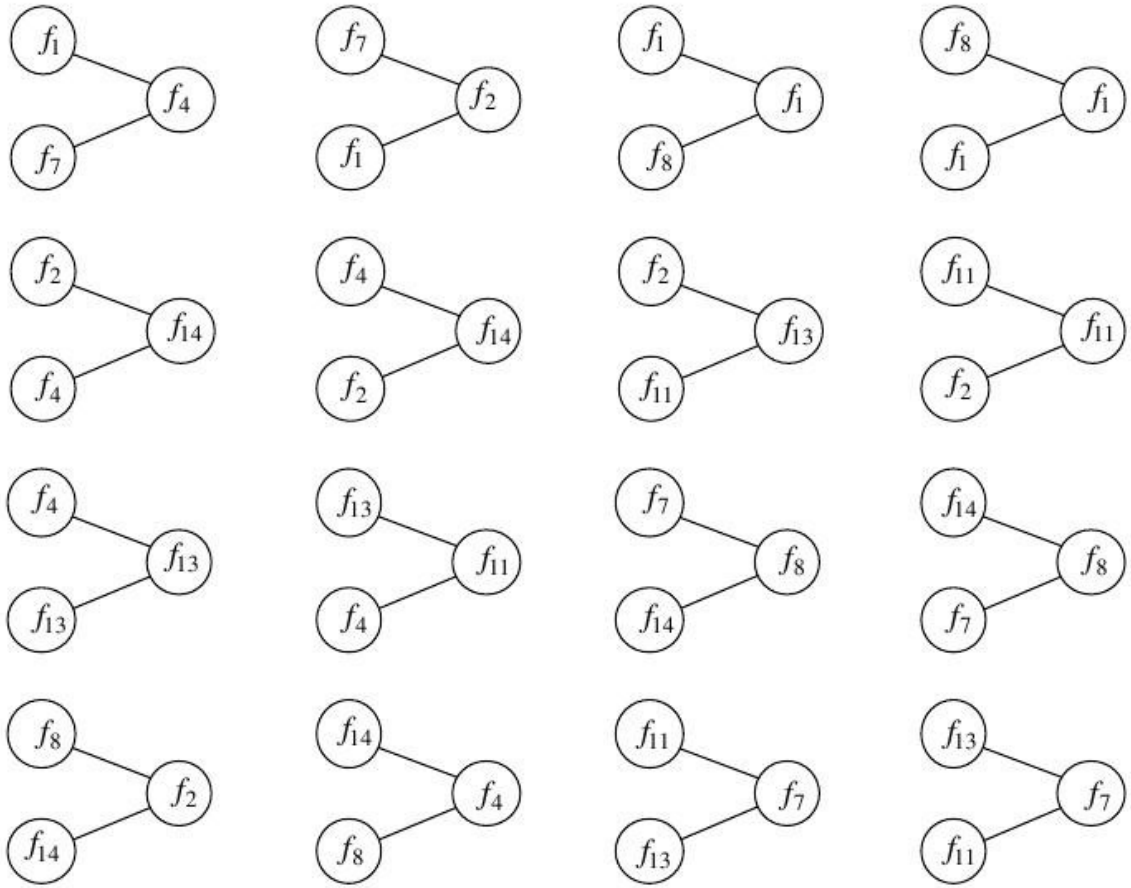
$$
\begin{aligned}
f_0(x_1, x_2) &= f_{0000}(x_1, x_2) = 0 \\
f_1(x_1, x_2) &= f_{0001}(x_1, x_2) = \neg(x_1 \vee x_2) \\
f_2(x_1, x_2) &= f_{0010}(x_1, x_2) = x_1 \wedge \neg x_2 \\
f_3(x_1, x_2) &= f_{0011}(x_1, x_2) = \neg x_2 \\
f_4(x_1, x_2) &= f_{0100}(x_1, x_2) = \neg x_1 \wedge x_2 \\
f_5(x_1, x_2) &= f_{0101}(x_1, x_2) = \neg x_1 \\
f_6(x_1, x_2) &= f_{0110}(x_1, x_2) = x_1 \oplus x_2 \\
f_7(x_1, x_2) &= f_{0111}(x_1, x_2) = \neg(x_1 \wedge x_2)
\end{aligned}
$$

$$
\begin{aligned}
f_8(x1, x_2) &= f_{1000}(x_1, x_2) = x_1 \wedge x_2 \\
f_9(x1, x_2) &= f_{1001}(x_1, x_2) = x_1 \equiv x_2 \\
f_{10}(x1, x_2) &= f_{1010}(x_1, x_2) = x_1 \\
f_{11}(x1, x_2) &= f_{1011}(x_1, x_2) = x_1 \vee \neg x_2 \\
f_{12}(x1, x_2) &= f_{1100}(x_1, x_2) = x_2 \\
f_{13}(x1, x_2) &= f_{1101}(x_1, x_2) = \neg x_1 \vee x_2 \\
f_{14}(x1, x_2) &= f_{1110}(x_1, x_2) = x_1 \vee x_2 \\
f_{15}(x1, x_2) &= f_{1111}(x_1, x_2) = 1
\end{aligned}
$$

- f_0 to funkcja stała $(x_1, x_2) \mapsto 0$; $f_8 = f_{1000}$ to AND; f_6 to XOR.

Sieci jedno - i dwuwarstwowe - problem XOR

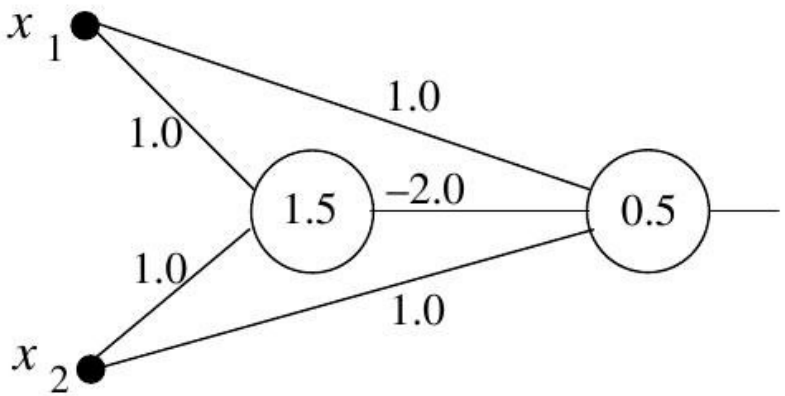
- 16 możliwości rozwiązania problemu XOR; można odszukać $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) = f_{14}(f_2(x_1, x_2), f_4(x_1, x_2))$.



The 16 solutions for the computation of XOR with three computing units

Sieci jedno - i dwuwarstwowe - problem XOR

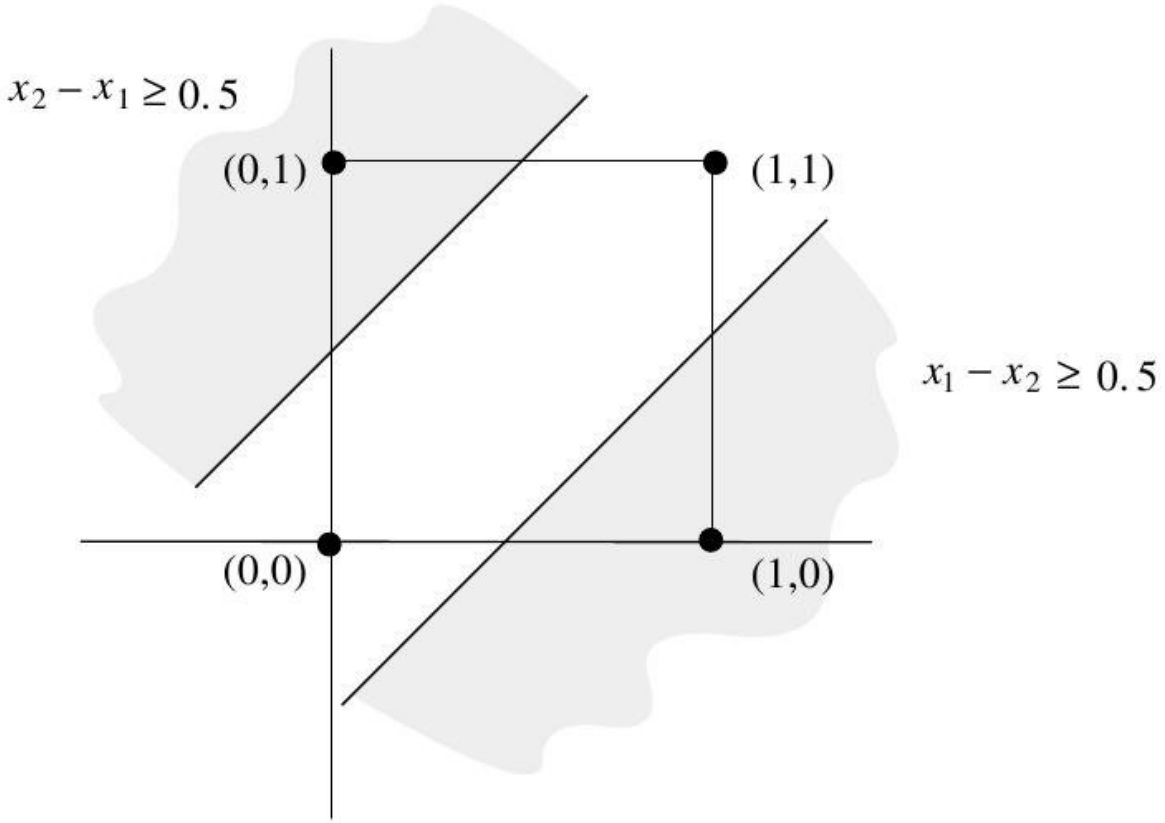
- Widoczne **symetrie** nie są przypadkowe.
- Zwiększając liczbę jednostek w warstwach ukrytych - więcej możliwych kombinacji - sieć ma większą **pojemność**.
- Przykład sieci, która nie jest czysto warstwowa;
  - ostatnia jednostka oblicza OR ale jeśli $x_1 = x_2 = 1$ następuje inhibicja.



Two unit network for the computation of XOR

Sieci jedno - i dwuwarstwowe - wizualizacja geometryczna

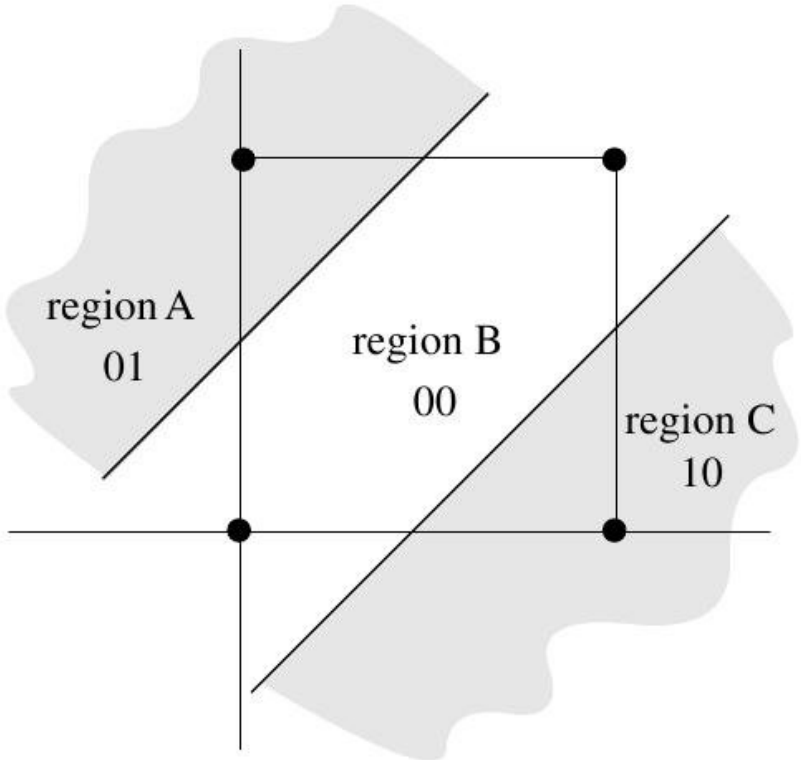
- Symetrię 16-tu rozwiązań problemu XOR można zrozumieć patrząc na **obszary przestrzeni wag** wyznaczane przez sieć 2-warstwową.
- Poniższy rysunek pokazuje obszary dla sieci XOR (str. 168), zaznaczono dodatnie półprzestrzenie.



Space separation defined by a two layered network

Sieci jedno - i dwuwarstwowe - wizualizacja geometryczna

- Powstałe 3 obszary mogą być identyfikowane z użyciem 2 bitów - każdy bit oznacza czy dany obszar należy do dodatniej czy ujemnej półprzestrzeni związaną z daną prostą.

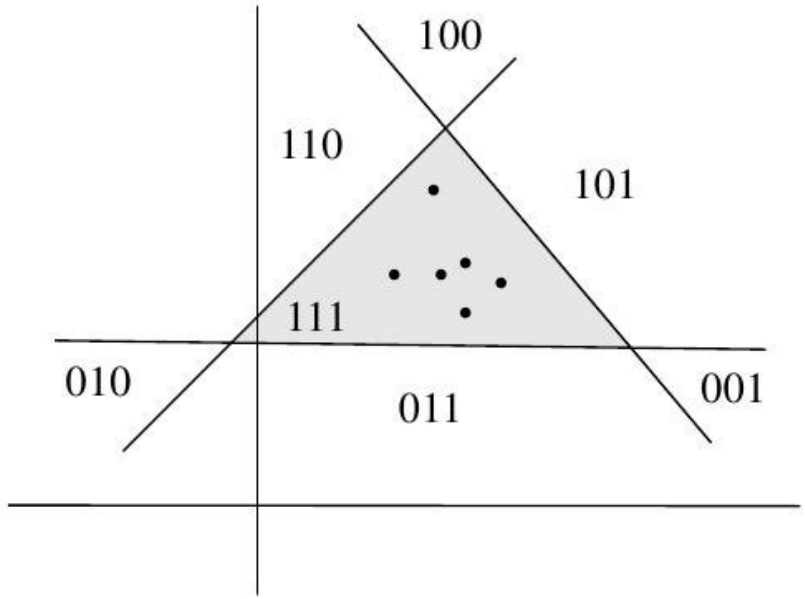


Labeling of the regions in input space

Sieci jedno - i dwuwarstwowe - wizualizacja geometryczna

- Jeżeli zaznaczonym obszarom przypiszemy wartość 1, a niezaznaczonemu 0 - problem XOR rozwiązany.
- Ogólna cecha sieci warstwowych - pierwsza warstwa jednostek obliczeniowych odwzorowuje wektor wejściowy na drugą przestrzeń tzw. **przestrzeń klasyfikacji lub cech**, która dekoduje klasyfikację wyznaczoną przez jednostki ukryte obliczając ostateczny wynik.
- Każda jednostka pierwszej warstwy ukrytej oblicza liniową separację przestrzeni wejściowej
  - w przestrzeni 2-wymiarowej można wyizolować **klastry** punktów używając 3 linii prostych:

Sieci jedno - i dwuwarstwowe - wizualizacja geometryczna



Delimiting a cluster with three linear separations

Sieci jedno - i dwuwarstwowe - wizualizacja geometryczna

- Jeżeli szukamy sieci dającej 1 dla punktów klastra - potrzebne 3 jednostki ukryte i na końcu jedna obliczająca *AND*.
  - ostatnia jednostka dekoduje (111) dając wartość 1.
- Aby wyizolować **wypukły klaster** punktów w przestrzeni **n-wymiarowej** potrzeba przynajmniej $n + 1$ jednostek ukrytych.
- Aby zidentyfikować **sumę** 2 **klastrów** i otrzymać 1:
  - potrzebne $2x3$ jednostki do indentyfikacji każdego z klastrów
  - 2 jednostki *AND* w warstwie ukrytej
  - jednostka obliczająca *OR*
- Opisany sposób izolacji klastrów **nie jest optymalny** - nie była rozpatrywana możliwość ponownego użycia zdefiniowanych poprzednio hiperpłaszczyzn.
- **Ogólnie** - nie wiadomo ile klastrów zawierają dane wejściowe ani czy są one wypukłe.

Sieci jedno - i dwuwarstwowe - wizualizacja geometryczna

- **Problem** - ile obszarów można zdefiniować z użyciem przecinających się półprzestrzeni i dlaczego w pewnych wypadkach sieć nie jest wystarczająco **plastyczna** aby rozwiązać dany problem.
- Znajdźmy wektor wag dla perceptronu obliczającego *AND*:

$$\begin{aligned}
(0,0) &: 0w_1 + 0w_2 + 1w_3 < 0, \quad \text{wyjście} = 0 \\
(0,1) &: 0w_1 + 1w_2 + 1w_3 < 0, \quad \text{wyjście} = 0 \\
(1,0) &: 1w_1 + 0w_2 + 1w_3 < 0, \quad \text{wyjście} = 0 \\
(1,1) &: 1w_1 + 1w_2 + 1w_3 \geq 0, \quad \text{wyjście} = 1
\end{aligned}$$

- Otrzymuje się 4 płaszczyzny przechodzące przez środek układu współrzędnych.

Sieci jedno - i dwuwarstwowe - wizualizacja geometryczna

$$
\begin{aligned}
1 &: \quad w_3 = 0 \\
2 &: \quad w_2 + w_3 = 0 \\
3 &: \quad w_1 + w_3 = 0 \\
4 &: \quad w_1 + w_2 + w_3 = 0
\end{aligned}
$$

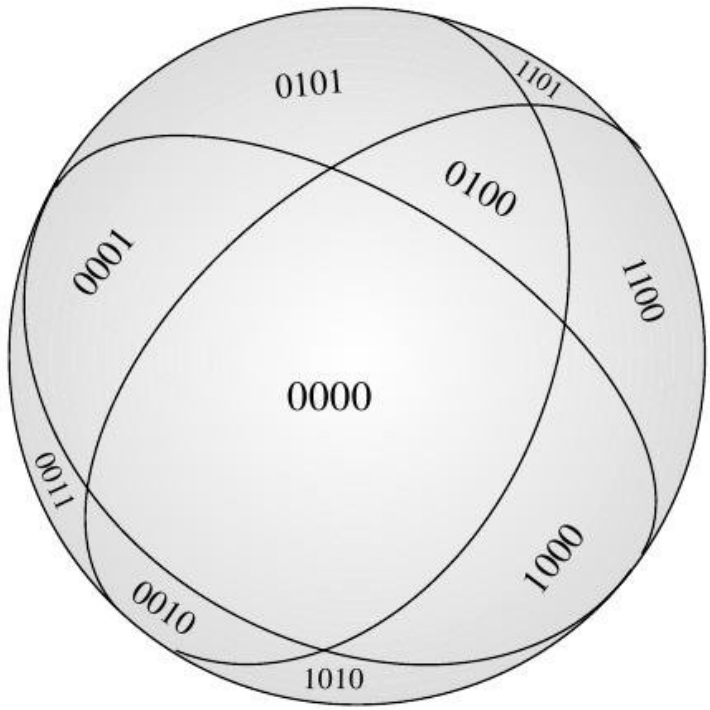
- 3 płaszczyzny w przestrzeni 3-wymiarowej definiują 8 obszarów a 4 płaszczyzny - 14 obszarów.
- Ponieważ jest 16 funkcji boolowskich 2 zmiennych 2 nie mogą być obliczone perceptronem (wiemy już, że chodzi o XOR i $\neg XOR$).

Sieci jedno - i dwuwarstwowe - wizualizacja geometryczna

- Te 14 obszarów można przedstawić na sferze.
  - 4 nierówności związane z punktami (1,1), (0,1), (1,0) i (0,0) definiują wielościan w przestrzeni wag
  - nakłada się warunek normalizacji wektorów wag, co powoduje że końce wszystkich wektorów leżą na sferze
  - nie wpływa to na działanie perceptronu
  - obszary zdefiniowane przez hiperpłaszczyzny odpowiadające funkcjom logicznym stają się obszarami na sferze jednostkowej

Sieci jedno - i dwuwarstwowe - wizualizacja geometryczna

- **Oznaczenia** - bity wyjściowe.
  - np. $f_{1000} = f_8$ to funkcja AND.

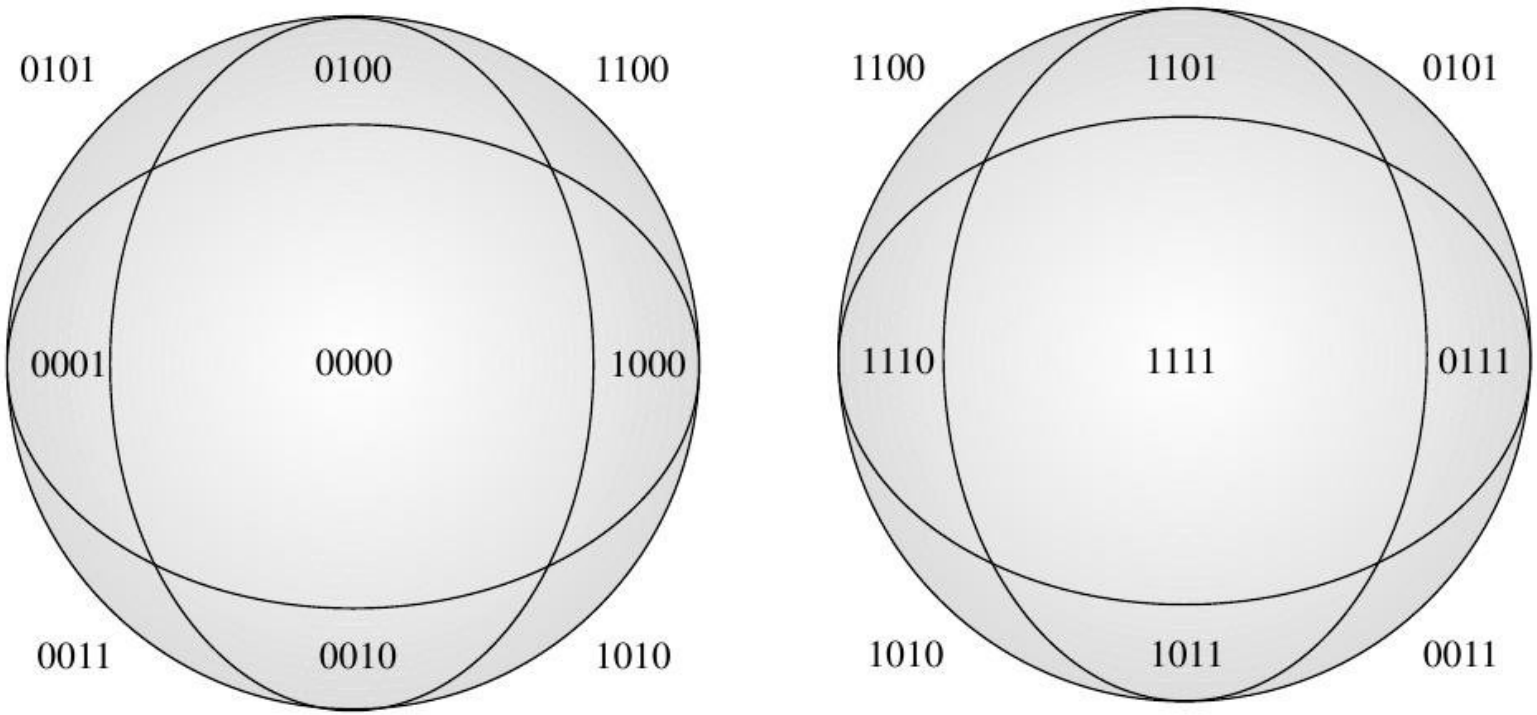


The Boolean sphere

Sieci jedno - i dwuwarstwowe - wizualizacja geometryczna

- Obszary oddzielone kołem wielkim - różnią się 1 bitem.
- Każdy obszar sąsiaduje z 3 lub 4 innymi.
- Np. obszar AND (1000) ma 3 sąsiadów 0000, 1100, 1010.
- Na sferze nie ma obszaru 1001 - funkcja nieobliczalna.

Sieci jedno - i dwuwarstwowe - wizualizacja geometryczna



Two opposite sides of the Boolean sphere

Sieci jedno - i dwuwarstwowe - wektory bipolarne

- **Wektory bipolarne** - używają -1 zamiast zera (wektory binarne - 0 i 1)
- Nie zmienia to własności perceptronu ale zmienia symetrię obszarów związanych z rozwiązaniem.
- Odpowiednie równania dla funkcji *AND*:

$$\begin{aligned} 1 &: \quad -w_1 - w_2 + w_3 = 0 \\ 2 &: \quad -w_1 + w_2 + w_3 = 0 \\ 3 &: \quad w_1 - w_2 + w_3 = 0 \\ 4 &: \quad w_1 + w_2 + w_3 = 0 \end{aligned}$$

- Płaszczyzny spotykają się w środku układu współrzędnych - tworzą **wielościany foremne**
  - wektory normalne - parami iloczyn skalarny 1 lub -1

Sieci jedno - i dwuwarstwowe - wektory bipolarne

- **Rozmiary** obszarów są różne i związane z **trudnością nauki** perceptronu.
- Dobrze by było gdyby obszary miały podobne rozmiary - może być potrzeba nauczenia perceptronu dowolnej funkcji.
- Tabela pokazuje rozmiary otrzymane metodą Monte Carlo dla funkcji boolowkich 2 zmiennych.

Table Relative sizes of the regions on the Boolean sphere as percentage of the total surface

Coding	*Boolean function number*							
	0	1	2	3	4	5	6	7
binary	26.83	2.13	4.18	4.13	4.17	4.22	0.00	4.13
bipolar	8.33	6.29	6.26	8.32	6.24	8.36	0.00	6.22

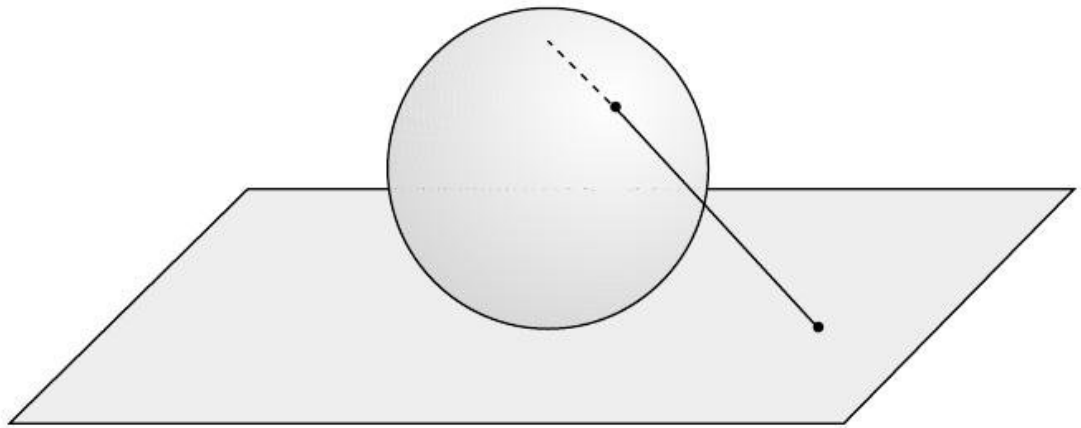
Coding	*Boolean function number*							
	8	9	10	11	12	13	14	15
binary	4.28	0.00	4.26	4.17	4.17	4.14	2.07	27.12
bipolar	6.16	0.00	8.42	6.33	8.27	6.31	6.25	8.23

Sieci jedno - i dwuwarstwowe - wektory bipolarne

- Maksymalna **zmienność** w kodowaniu bipolarnym - 1.44, natomiast w binarnym 12.5.
- Także **empirycznie** zaobserwowano, że łatwiej jest nauczyć wielowarstwową sieć neuronową w reprezentacji bipolarnej.
- Można pokazać, że pod względem rozmiarów obszarów **kodowanie bipolarne jest optymalne**.
- Kodowanie bipolane jest też lepsze dla przestrzeni wielowymiarowej o dużym n.

Sieci jedno - i dwuwarstwowe - projekcja obszarów rozwiązań

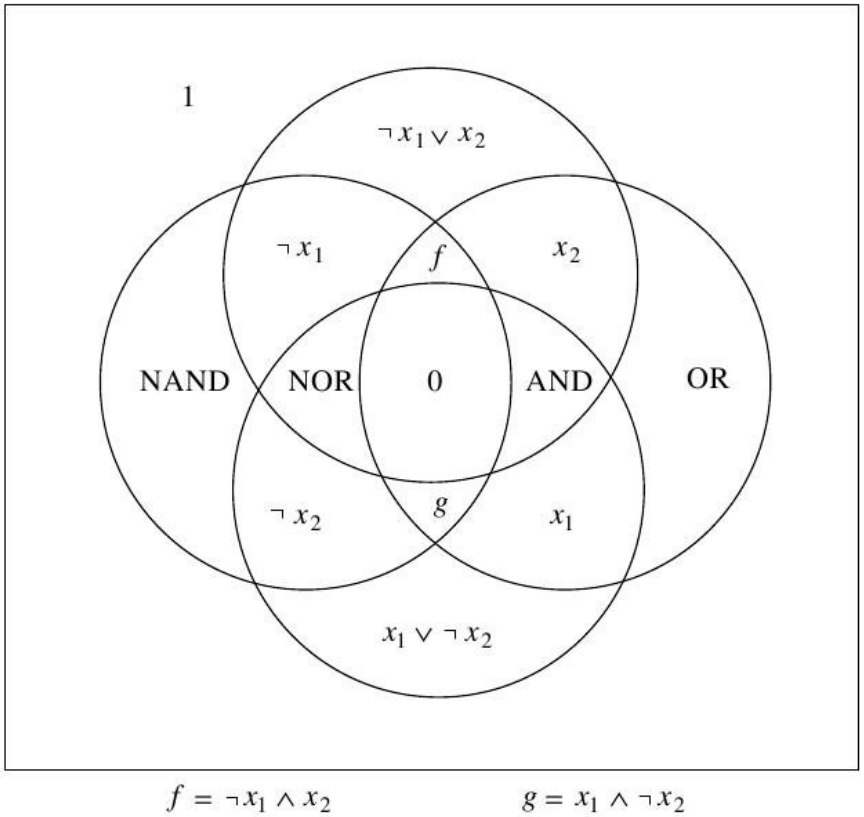
- Innym sposobem analizy obszarów rozwiązań jest projekcja stereograficzna sfery na płaszczyznę (biegun północny - odwzorowany na ∞).



Stereographic projection

Sieci jedno - i dwuwarstwowe - projekcja obszarów rozwiązań

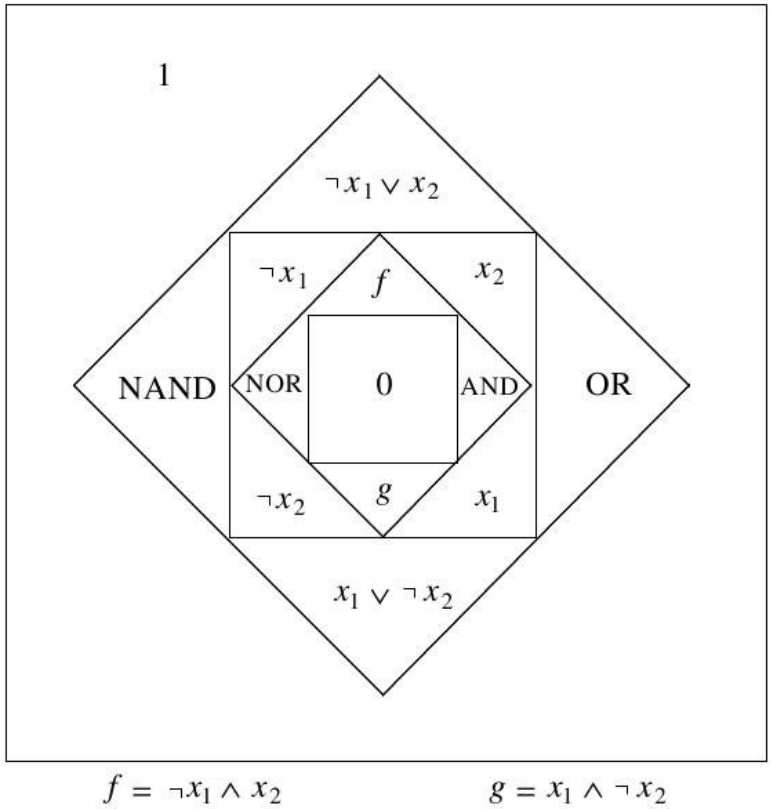
- **Projekcja stereograficzna** rzutuje okręgi na elipsy - kształt nie jest istotny więc można elipsy przekształcić w okręgi (centrum obszaru 1111 wybrano jako biegun północny)



Projection of the solution regions of the Boolean sphere

Sieci jedno - i dwuwarstwowe - projekcja obszarów rozwiązań

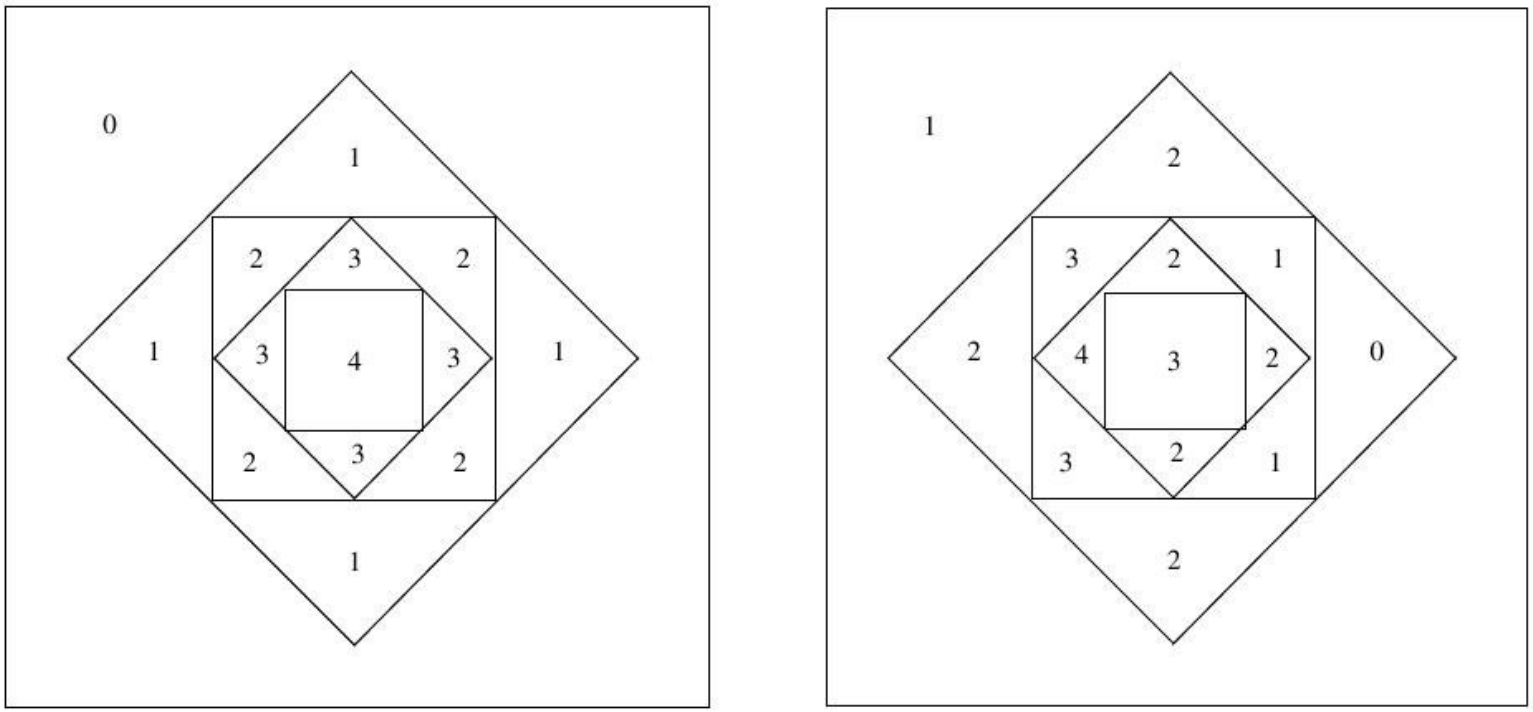
- Symetria jest lepiej widoczna w stylizowanej reprezentacji - obszary funkcji f i $\neg f$ są położone symetrycznie.



Stylized representation of the projected solution regions

Sieci jedno - i dwuwarstwowe - projekcja obszarów rozwiązań

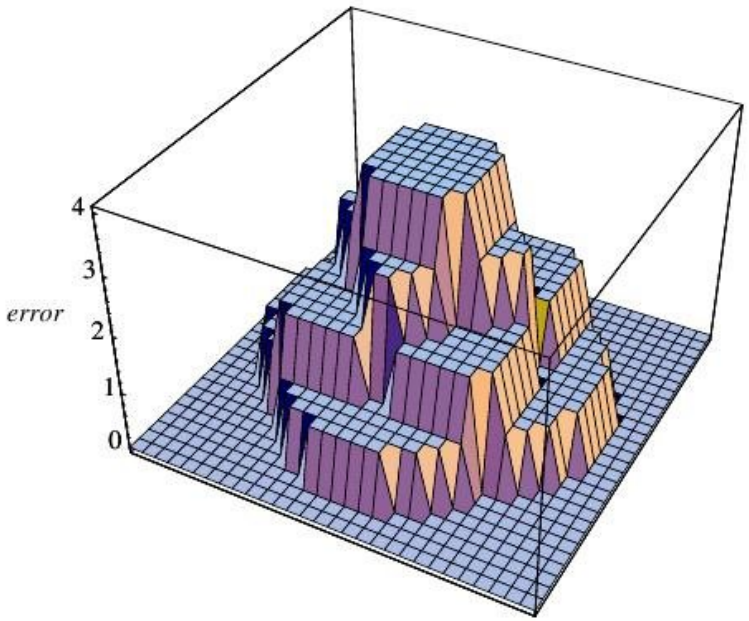
- Liczba sąsiednich obszarów ma znaczenie jeżeli algorytm zaczyna z losowego punktu i przechodzi z obszaru do obszaru.
- Funkcja błędu (po lewej: dla f_{1111} - minimum a dla f_{0000} - maksimum)



Error functions for the computation of f_{1111} and $x_1 \vee x_2$

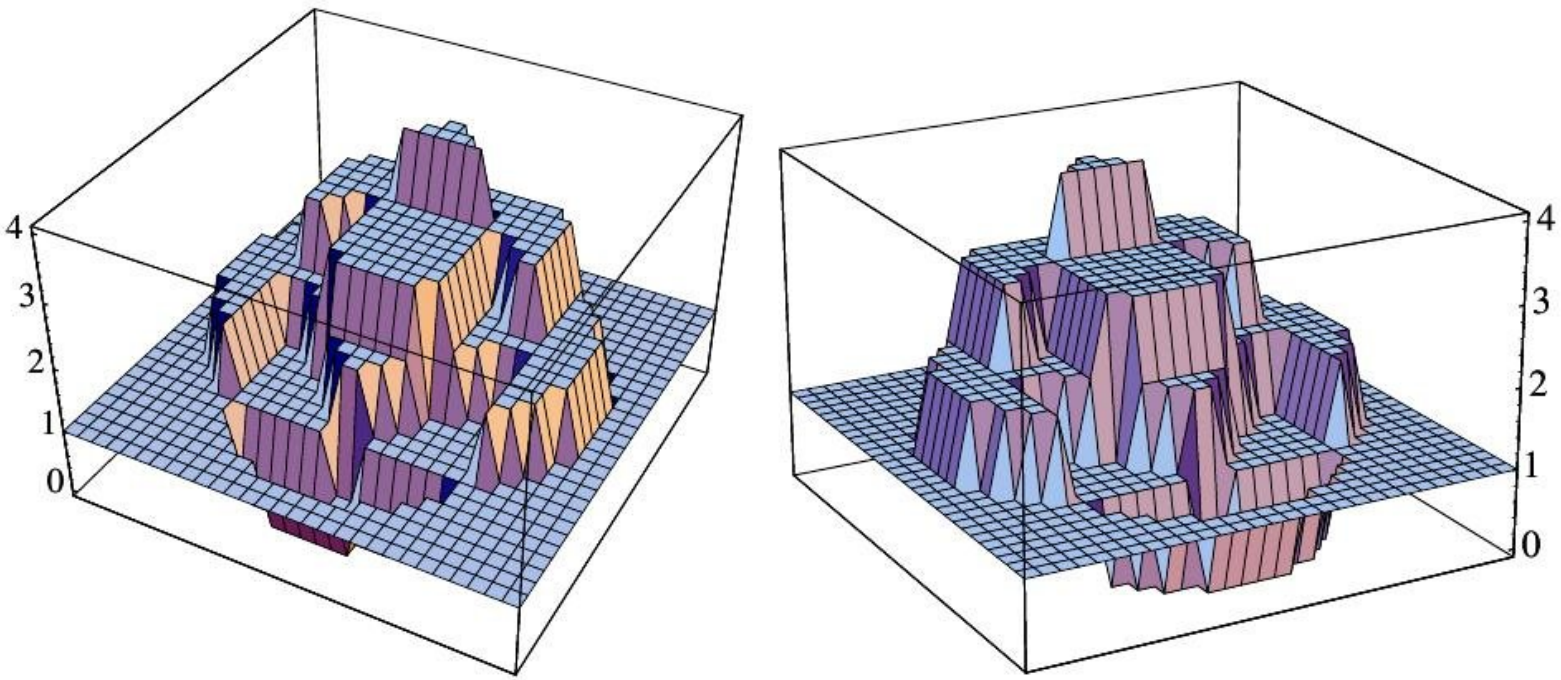
Sieci jedno - i dwuwarstwowe - projekcja obszarów rozwiązań

- Im większy błąd tym więcej możliwych ścieżek do obszaru o zerowym błędzie.



The error function for f_{1111}

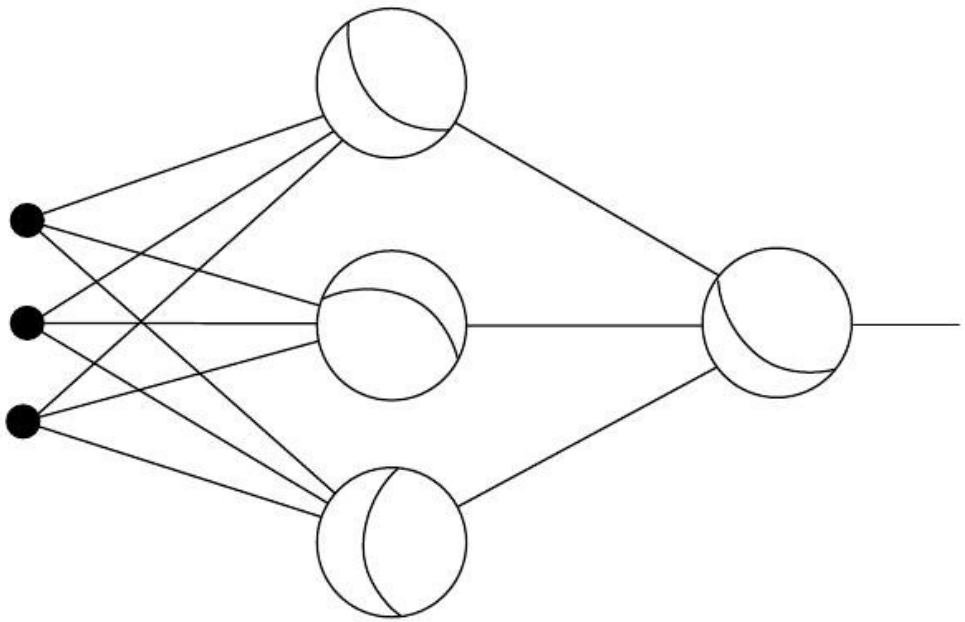
Sieci jedno - i dwuwarstwowe - interpretacja geometryczna



Two perspectives of the error function for OR

Sieci jedno - i dwuwarstwowe - interpretacja geometryczna

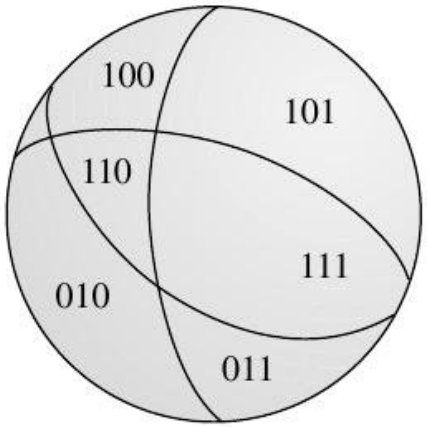
- Sfera boolowska pozwala interpretować działanie sieci z warstwą ukrytą.
- W poniższej sieci z 3 perceptronami w warstwie ukrytej każda jednostka dzieli przestrzeń wejściową na 2 półprzestrzenie.



Stylized representation of the input space separations

Sieci jedno - i dwuwarstwowe - interpretacja geometryczna

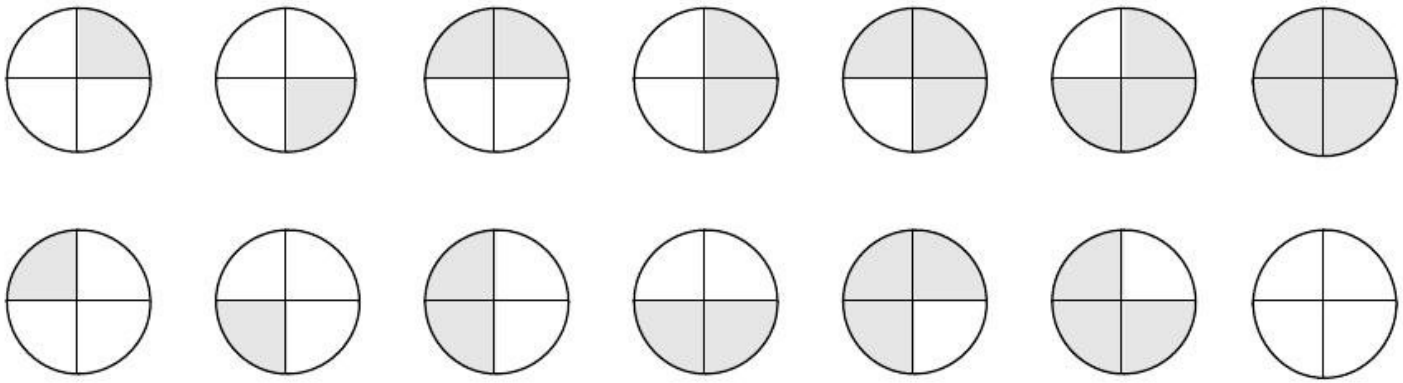
- Podział przestrzeni można podsumować na jednej sferze
  - 3 jednostki generują 3-bitowe oznaczenie każdego obszaru
  - jednostka wyjściowa dekoduje te bity przypisując im wartość 0 lub 1



Labeling of the regions in input space

Sieci jedno - i dwuwarstwowe - interpretacja geometryczna

- Jakie podziały przestrzeni generuje sieć z dwoma jednostkami w warstwie ukrytej?
  - są 4 obszary
  - wartość oznaczona zacienieniem obszaru
  - z 16 możliwości pokolorowania 2 odrzucamy jako nieobliczalne - jednostka wyjściowa nie może zdekodować XOR [przypomnienie - dekodery - inhibicja]



Coloring of the regions in input space

Sieci jedno - i dwuwarstwowe - interpretacja geometryczna

- Rozważania dotyczące liczby obszarów można sformalizować.
- $R(m, n)$ - liczba różnych obszarów zdefiniowanych przez m powierzchni o wymiarze $n - 1$.

$$R(m, n) = 2 \sum_{i=0}^{n-1} \binom{m-1}{i}$$

dimension

m \ n	0	1	2	3	4
1	0	2	2	2	2
2	0	2	4	4	4
3	0	2	6	8	8
4	0	2	8	14	16
5	0	2	10	22	30

Recursive calculation of $R(m, n)$

Sieci jedno - i dwuwarstwowe - interpretacja geometryczna

- $T(2^n, n)$ - liczba funkcji progowych n zmiennych.
- Tabela pokazuje też górne ograniczenie na $R(m, n)$.

Table Comparison of the number of Boolean and threshold functions of n inputs and two different bounds

n	2^{2^n}	$T(2^n, n)$	$R(2^n, n)$	$\lfloor 2^{n^2+1}/n! \rfloor$
1	4	4	4	4
2	16	14	14	16
3	256	104	128	170
4	65,536	1,882	3,882	5,461
5	4.3×10^9	94,572	412,736	559,240

Sieci jedno - i dwuwarstwowe - konsekwencje

- **Liczba funkcji progowych** w przestrzeni n-wymiarowej rośnie wielomianowo względej zmiennej 2^n (wszystkich funkcji logicznych - wykładniczo).
  - procent funkcji progowych w relacji do wszystkich logicznych dąży z n do zera
- Sieci z **dwoma lub więcej warstw** mają także problemy z uczeniem.
  - jeżeli warstwa ukryta zawiera m jednostek a wektor wejściowy ma wymiar n - maksymalna liczba obszarów to $R(m, n)$
  - jeżeli wektorów wejściowych jest więcej - może się zdarzyć, że nie będzie wystarczającej ilości obszarów dla obliczenia danej funkcji logicznej

Sieci jedno - i dwuwarstwowe - historia

- Sieci warstwowe zaproponowano już w początkach rozwoju sieci neuronowych nie znano dobrej metody ich uczenia.
- Rosenblatt eksperymentował z uczeniem, w którym błąd na wyjściu propagował się do pierwszych warstw sieci.
- Problem - położenie i liczba lokalnych minimów.
- Funkcje progowe studiowano intensywnie w latach 60-tych otrzymując ograniczenia na ich liczbę.
- Wykonano wiele badań własności topologicznych wielościanów i obszarów na sferach.
- Valiant - niezależna od modelu teoria uczenia.

Algorytm propagacji wstecznej

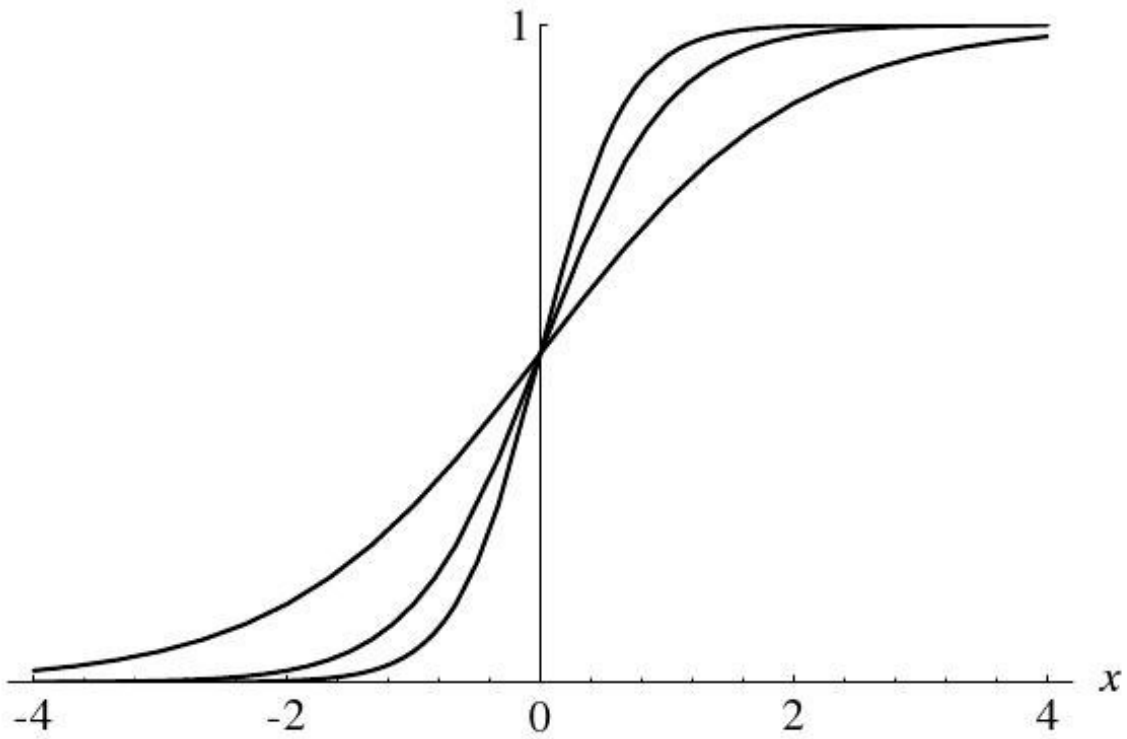
- **Sieci wielowarstwowe** mogą obliczać wiecej funkcji Boolowskich niż jednowarstwowe
- **Koszt** znalezienia wag **wzrasta** z ilością parametrów i bardziej skomplikowaną topologią sieci.
- **Algorytm propagacji wstecznej** - może poradzić sobie z takim złożonym problemem uczenia sieci.
- Używany był w różnych sytuacjach, ponownie odkrywany; jeden z najbardziej studiowanych i używanych algorytmów uczenia sieci.
- **Dowód** można oprzeć na podejściu **graficznym** - problem opisania grafu.
- Dowód analityczny - mniej ogólny (opisuje specjalne topologie sieci) i trudniejszy.
- Algorytm propagacji wstecznej - skuteczny także gdy w sieć może przekazywać tylko informację lokalną.

Algorytm propagacji wstecznej - funkcje aktywacji

- Szukamy **minimium funkcji błędu** w przestrzeni wag - metoda gradientowa.
- Rozwiązanie problemu uczenia - kombinacja wag odpowiadająca minimum globalnemu.
- **Gradient** - w każdym kroku funkcja błędu musi być różniczkowalna.
- **Funkcja aktywacji neuronu** - inna niż funkcja schodkowa znana z perceptronu.
- **Złożenie** funkcji na wyjściu z perceptronu będzie **nieciągłe** $\Rightarrow$ funkcja błędu nieciągła.
- Popularna funkcja dla algorytmu propagacji wstecznej - **sigmoid**, czyli $s_c : \mathcal{R} \to (0, 1)$

$$s_c = \frac{1}{1 + e^{-cx}}$$

Algorytm propagacji wstecznej - funkcje aktywacji



Three sigmoids (for $c = 1$, $c = 2$ and $c = 3$)

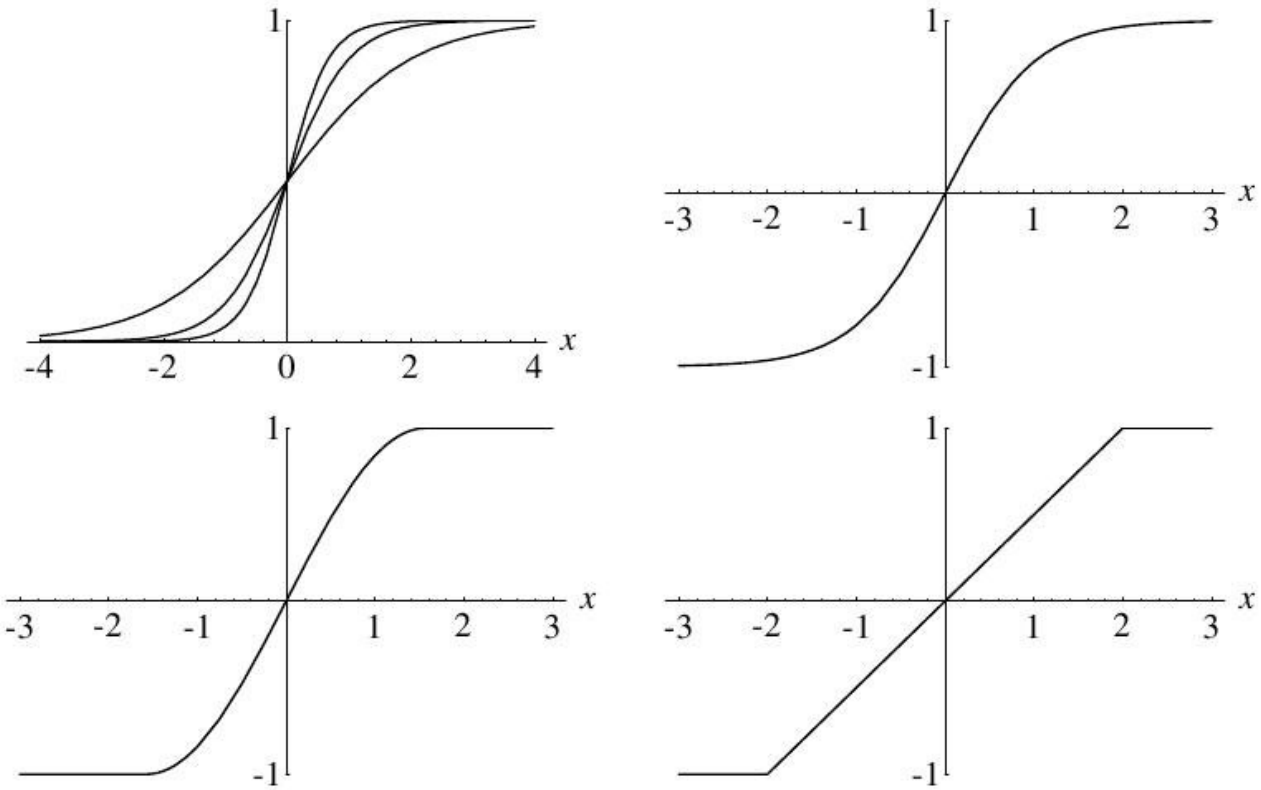
Algorytm propagacji wstecznej - funkcje aktywacji

- Parametr c można wybrać dowolnie, gdy $\mathbf{c} \to \infty$ funkcja staje się schodkowa.
- W sieciach stochastycznych - $1/c$ nazywa się parametrem temperaturowym.
- Dla uproszczenia wyrażeń założymy $c = 1$. Wtedy $s(x) \equiv s_1(x)$.
- **Pochodna**:

$$\frac{d}{dx}s(x) = \frac{e^{-x}}{(1+e^{-x})^2} = s(x)(1-s(x)).$$

- Alternatywą dla sigmoidu może być tzw. **sigmoid symetryczny**
  - $S(x) = 2s(x) - 1 = \frac{1-e^{-x}}{1+e^{-x}}$ $(S(-x) = -S(x))$.

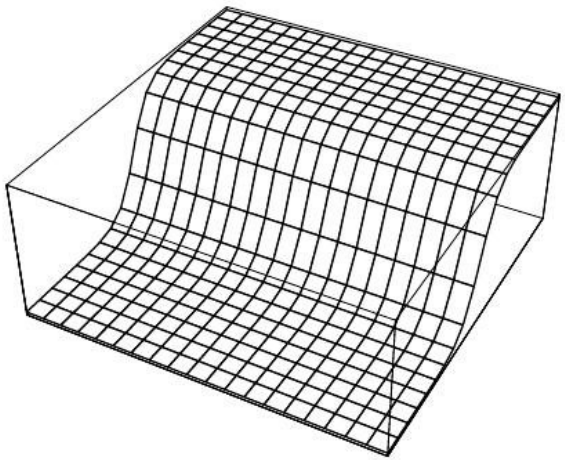
Algorytm propagacji wstecznej - funkcje aktywacji



Graphics of some "squashing" functions

Algorytm propagacji wstecznej - funkcje aktywacji

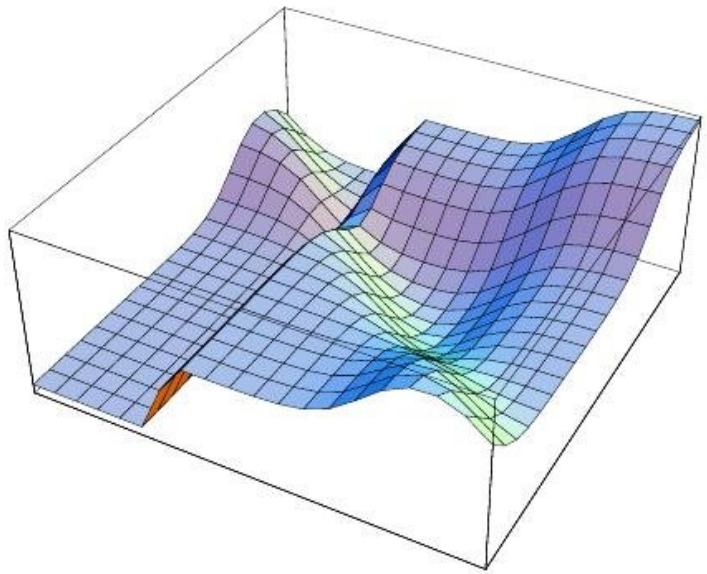
- Do algorytmu propagacji wstecznej - wiele różnych funkcji aktywacji zostało zaproponowanych.
- Rysunek pokazuje **wygładzenie** funkcji błędu - użycie sigmoidu.
  - Dla gradientu - ważne, że nie ma zupełnie płaskich rejonów - konsekwencja $s'(x) > 0$.



A step of the error function

Algorytm propagacji wstecznej - minima lokalne

- Ceną za różniczkowalność sigmoidu jest pojawianie się **minimów lokalnych** funkcji błędu.
- Nie występowały by one gdyby użyć funkcji schodkowej.
- Funkcja błędu na rysunku - obliczona dla pojedynczej jednostki o 2 wagach stałym progu i 4 przykładami w zbiorze treningowym.



A local minimum of the error function

Algorytm propagacji wstecznej - minima lokalne

- W wielu przypadkach minima lokalne powstają gdy docelowym wyjściem są **wartości inne niż** 0 lub 1.
- Gdy uczymy sieć do obliczania XOR aby dawała wartość 0.9 dla $(0, 1)$ i $(1, 0)$ - pojawiają się minima lokalne.
- Jednak nawet gdy **wartości docelowe są binarne** - minima lokalne są obecne.
- Lisboa i Perantonis - znaleźli analitycznie wszystkie minima dla funkcji XOR.

Algorytm propagacji wstecznej - zagadnienie uczenia

- Wyprowadzenie propagacji wstecznej = obliczanie gradientu odpowiednim opisaniem grafu.
- Dowód dla **ogólnych topologii** - nie ma ograniczenia do sieci warstwowych.
- Sieć neuronowa oblicza funkcję sieci ϕ; **uczenie** - szukanie wag najlepiej przybliżającą w najlepszy możliwy sposób szukaną funkcję f (daną niejawnie ale w postaci zbioru przykładów).
- Rozważmy sieć o n **wejściach** i m **wyjściach** o **dowolnej** liczbie jednostek ukrytych i **dowolnych** połączeniach.
  - Zbiór treningowy: $\{(\mathbf{x_1}, \mathbf{t_1}), ..., (\mathbf{x_p}, \mathbf{t_p})\}$
  - Funkcje na każdym węźle sieci - ciągłe i różniczkowalne.
  - Wagi - losowe liczby rzeczywiste.
  - $\mathbf{x_i} \rightarrow \mathbf{o_i} \neq \mathbf{t_i}$ - chcielibyśmy uzyskać równość albo ogólniej...

Algorytm propagacji wstecznej - zagadnienie uczenia

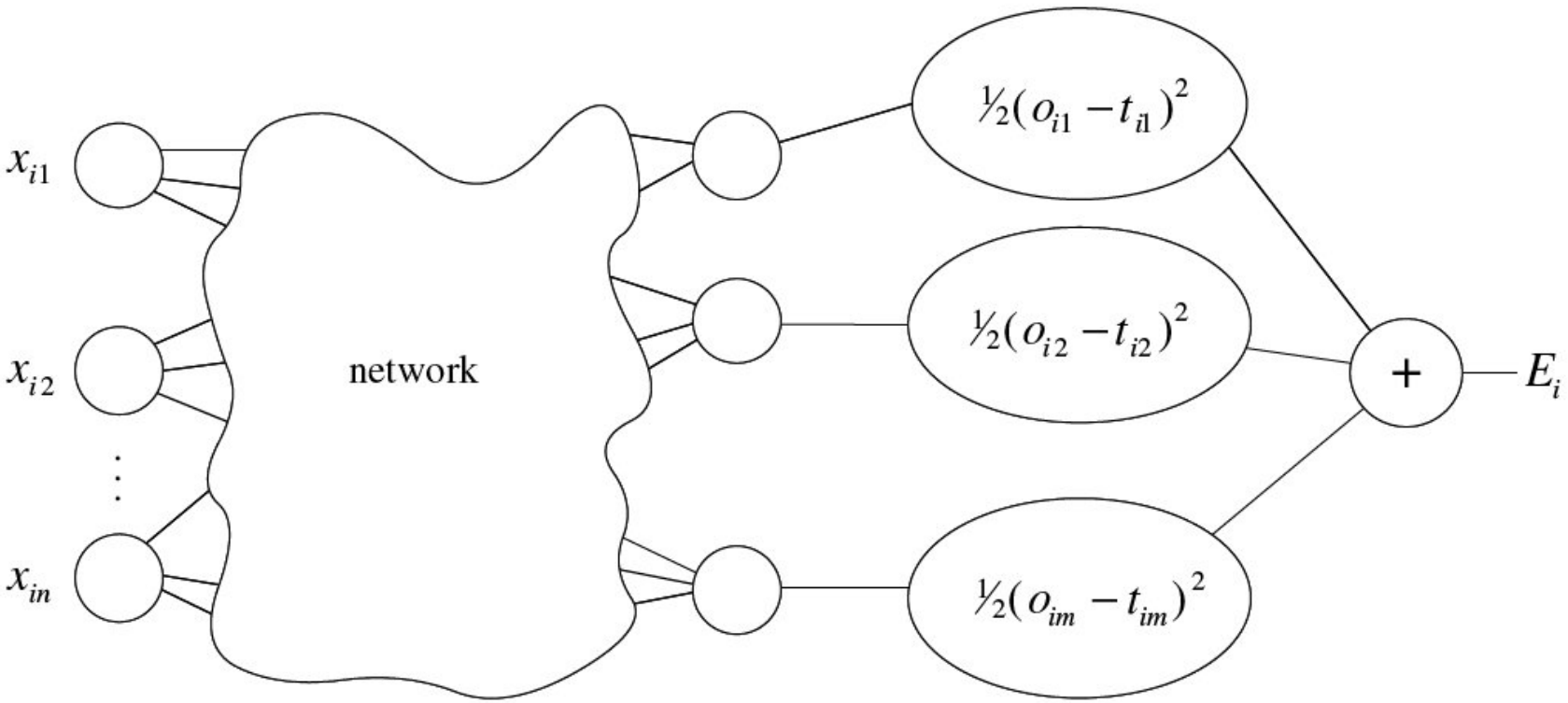
- ... zminimalizować błąd zdefiniowany jako

$$E = \frac{1}{2} \sum_{i=1}^{p} \|\mathbf{o_i} - \mathbf{t_i}\|^2$$

- Po treningu sprawdzamy, czy sieć potrafi **interpolować** - na nowych przykładach sprawdzamy czy są one rozpoznawane jako podobne do już znanych i czy wartości na wyjściu są podobne.
- Algorytm propagacji wstecznej - znajduje **minima lokalne** funkcji błędu - inicjacja sieci **losowymi wagami**.
- Gradient funkcji błędu jest obliczany i używany do korekcji początkowych wag.

Algorytm propagacji wstecznej - zagadnienie uczenia

- **Rozszerzamy sieć** aby obliczyć funkcję błędu.
- Podobnie dla każdego $\mathbf{t_i}$.



Extended network for the computation of the error function

Algorytm propagacji wstecznej - zagadnienie uczenia

- Funkcja błędu jest różniczkowalna - minimalizacja metodą gradientową

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_l} \right)$$

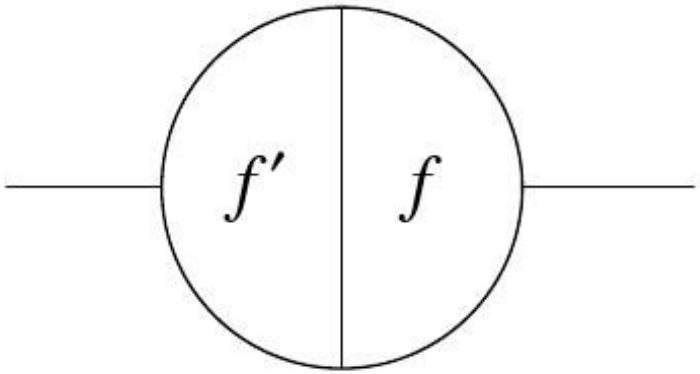
- Aktualizacja wag (λ - stała uczenia):

$$\Delta w_i = -\lambda \frac{\partial E}{\partial w_i}, \quad i = 1, ..., l.$$

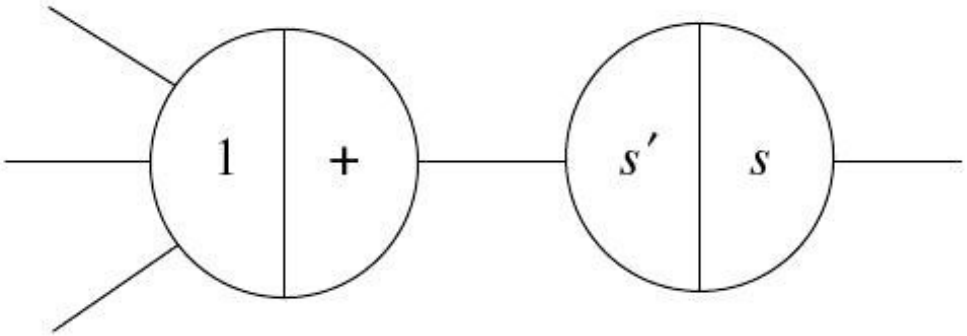
Algorytm propagacji wstecznej - pochodne funkcji sieci

- Cel - metoda efektywnego wyznaczania **gradientu funkcji sieci względem wag sieci**.
- Sieć jest równoważna skomplikowanemu złożeniu funkcji - **reguła łancuchowa** dla znajdowania gradientu.
- Węzłom sieci nadaje się złożoną strukturę, którą reprezentuje się **B-diagramem** (backpropagation diagram): prawa strona oblicza funkcję związaną z węzłem, lewa strona jej pochodną.
- Funkcja scalająca może być odseparowana od funkcji aktywacji - rozdzielamy węzeł sieci na 2 części.

Algorytm propagacji wstecznej - B-diagram



The two sides of a computing unit

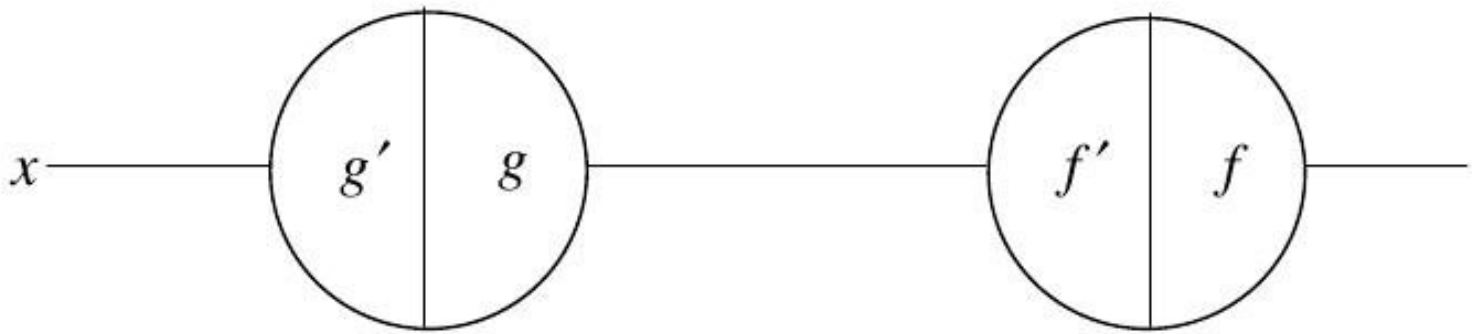


Separation of integration and activation function

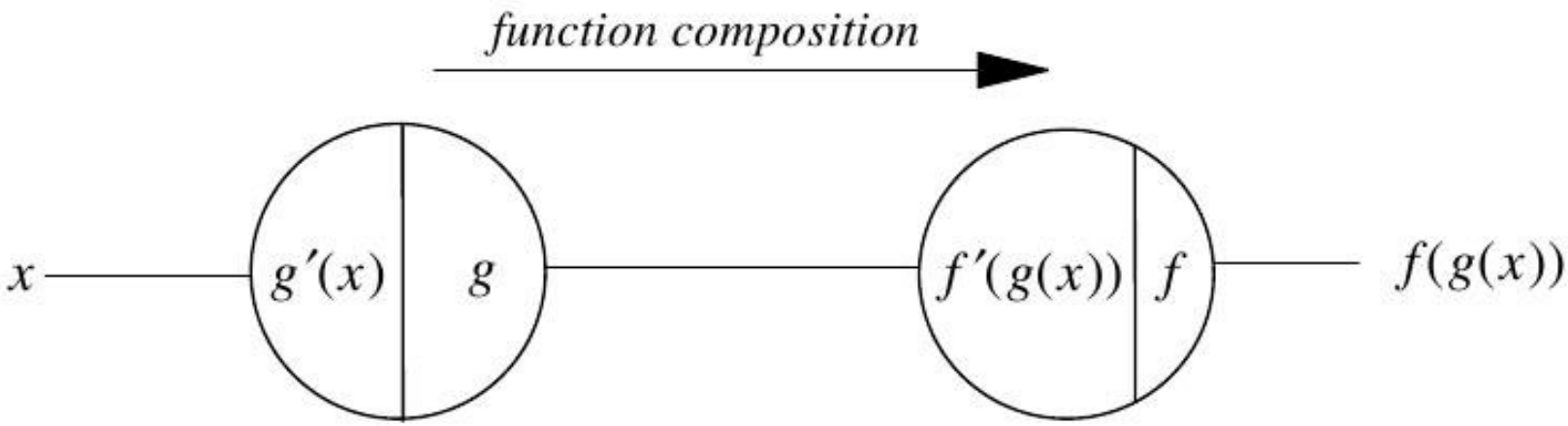
Algorytm propagacji wstecznej - B-diagram

- Sieć jest obliczana 2-etapowo:
  1. Informacja przepływa ze strony lewej do prawej - każda jednostka oblicza swoją funkcję aktywacji f oraz pochodną f'; obydwa rezultaty są przechowywane ale tylko f przekazywane jest do kolejnych jednostek po prawej.
  2. Propagacja wsteczna - informacja w sieci przekazywana jest z powrotem z prawej do lewej strony sieci (przechowywane wartości są użyte).

Algorytm propagacji wstecznej - złożenie funkcji

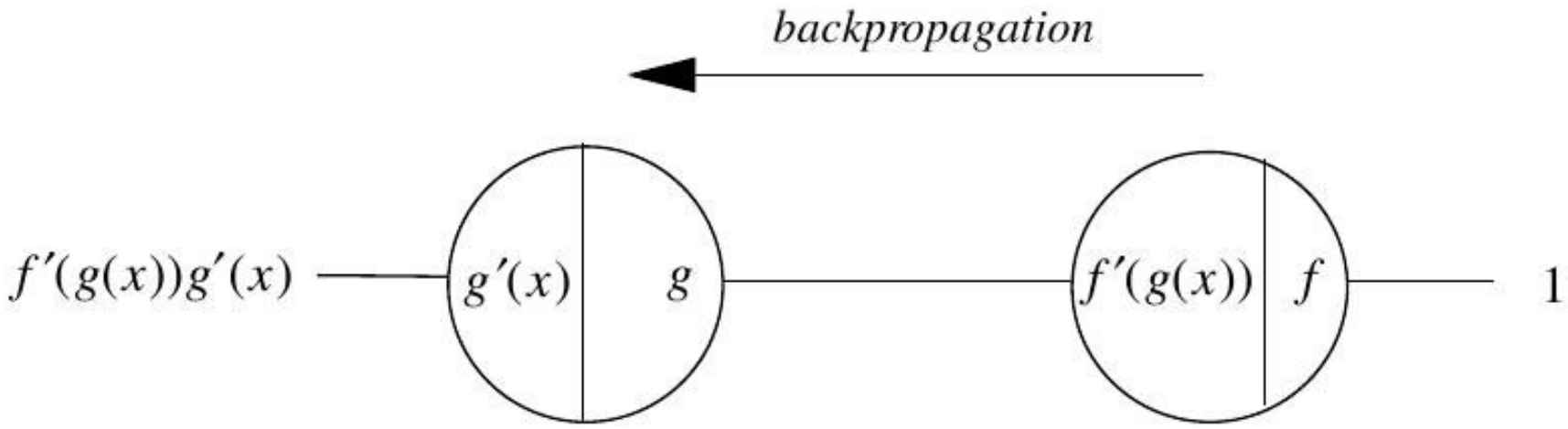


Network for the composition of two functions



Result of the feed-forward step

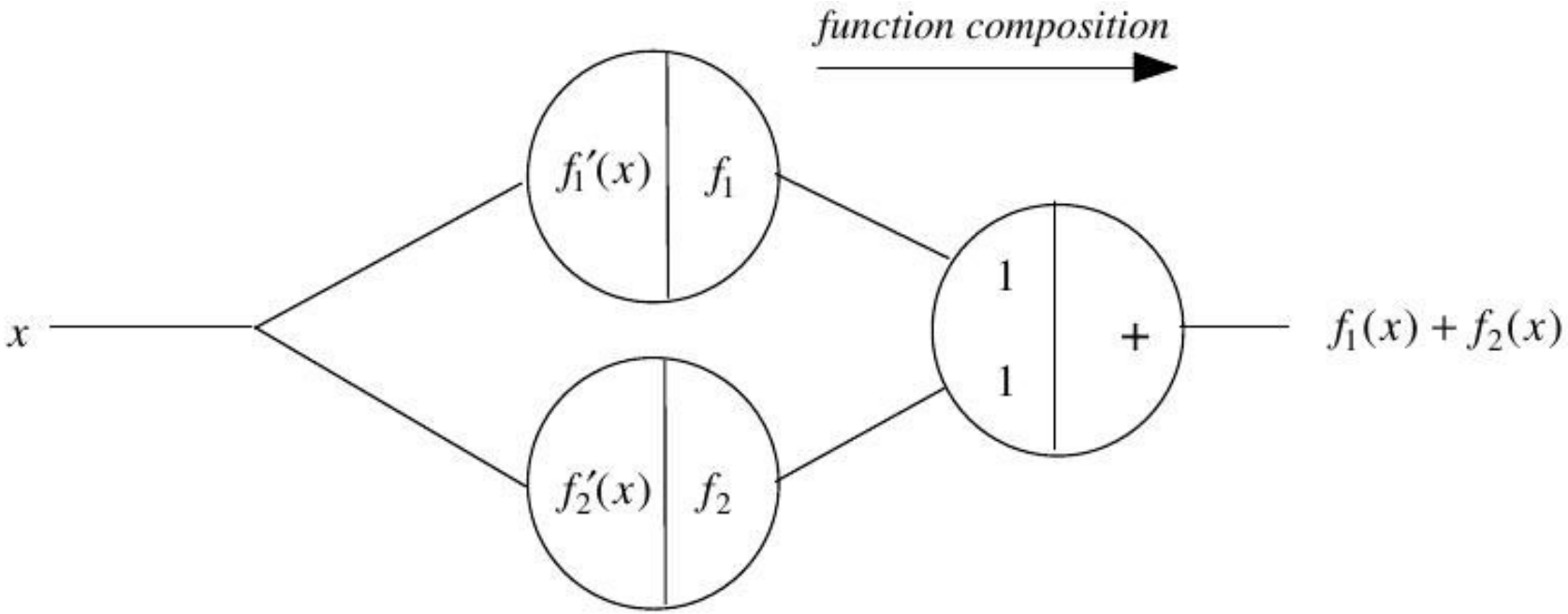
Algorytm propagacji wstecznej - złożenie funkcji



Result of the backpropagation step

- Krok propagacji wstecznej - wejście 1, informacja jest mnożona przez wartość zapisaną po lewej stronie jednostki.

Algorytm propagacji wstecznej - dodawanie funkcji

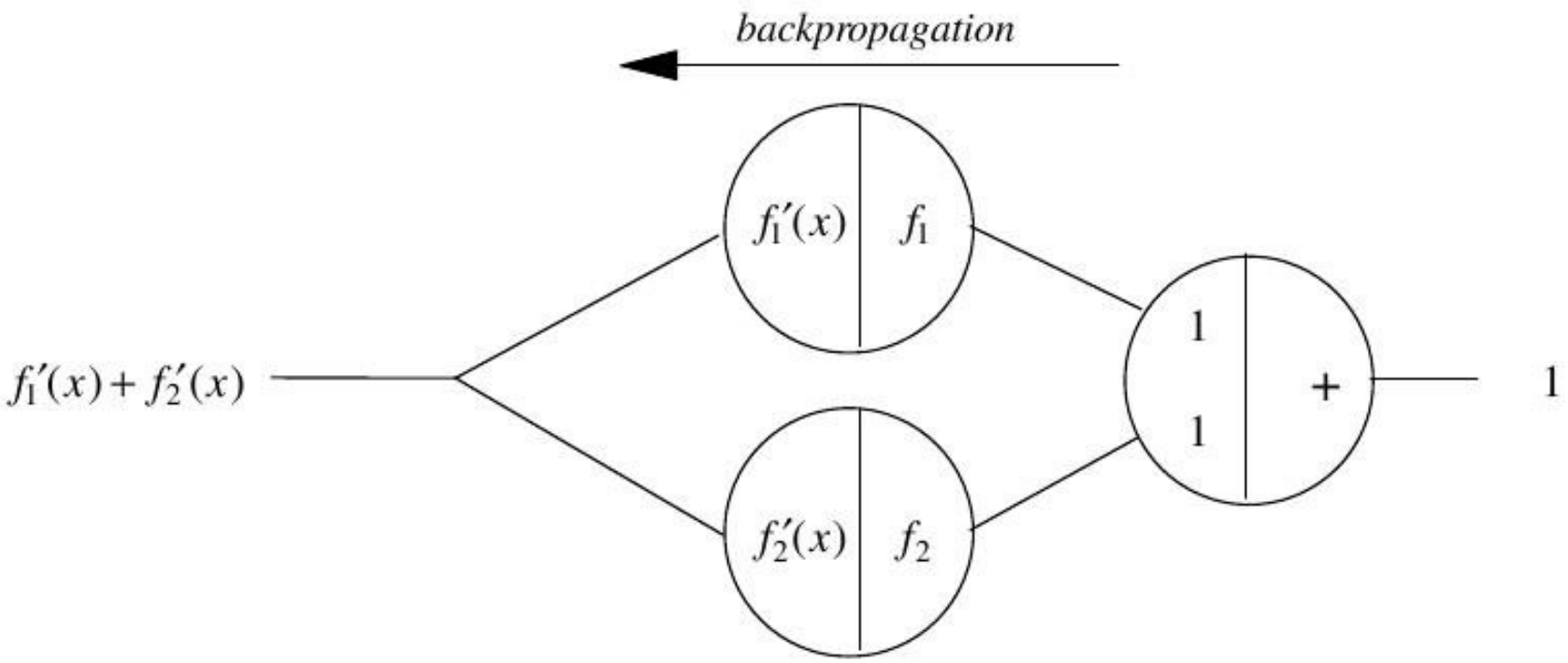


Addition of functions

- Pochodna z sumy funkcji względem którejkolwiek z nich to 1.
- Gdy dwie prawa-do-lewej ścieżki spotykają się - wartości są dodawane.

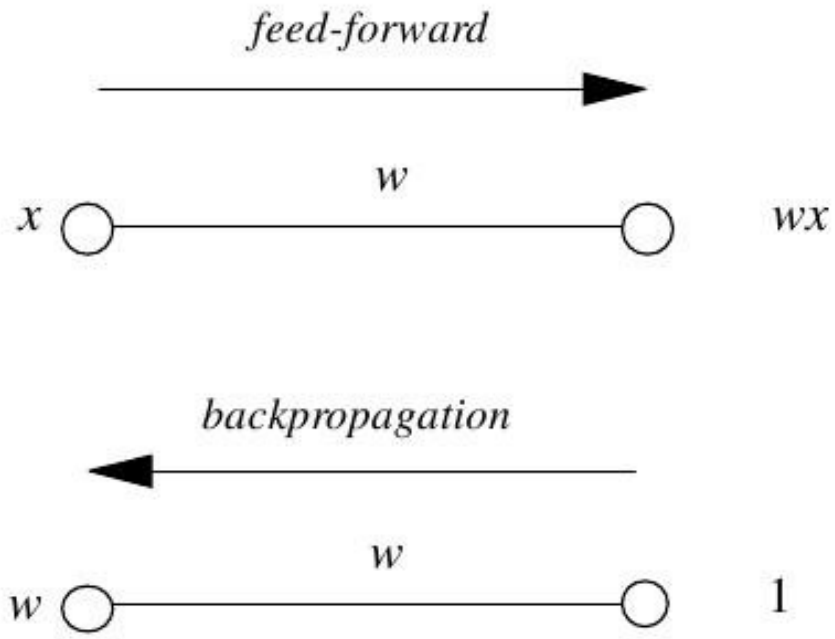
Algorytm propagacji wstecznej - dodawanie funkcji

- Gdy dwie prawa-do-lewej ścieżki spotykają się - wartości są dodawane.



Result of the backpropagation step

Algorytm propagacji wstecznej - wagi



Forward computation and backpropagation at an edge

- Wartość x mnożona jest przez wagę w.
- Propagacja wsteczna - wartość 1 mnożona jest przez w - rezultatem jest w czyli pochodna wx.

Algorytm propagacji wstecznej

Algorytm. Rozważmy sieć z jednym wejściem $x \in \mathbb{R}$ i funkcją sieci F. Pochodna $F'(x)$ obliczana jest w dwóch krokach:

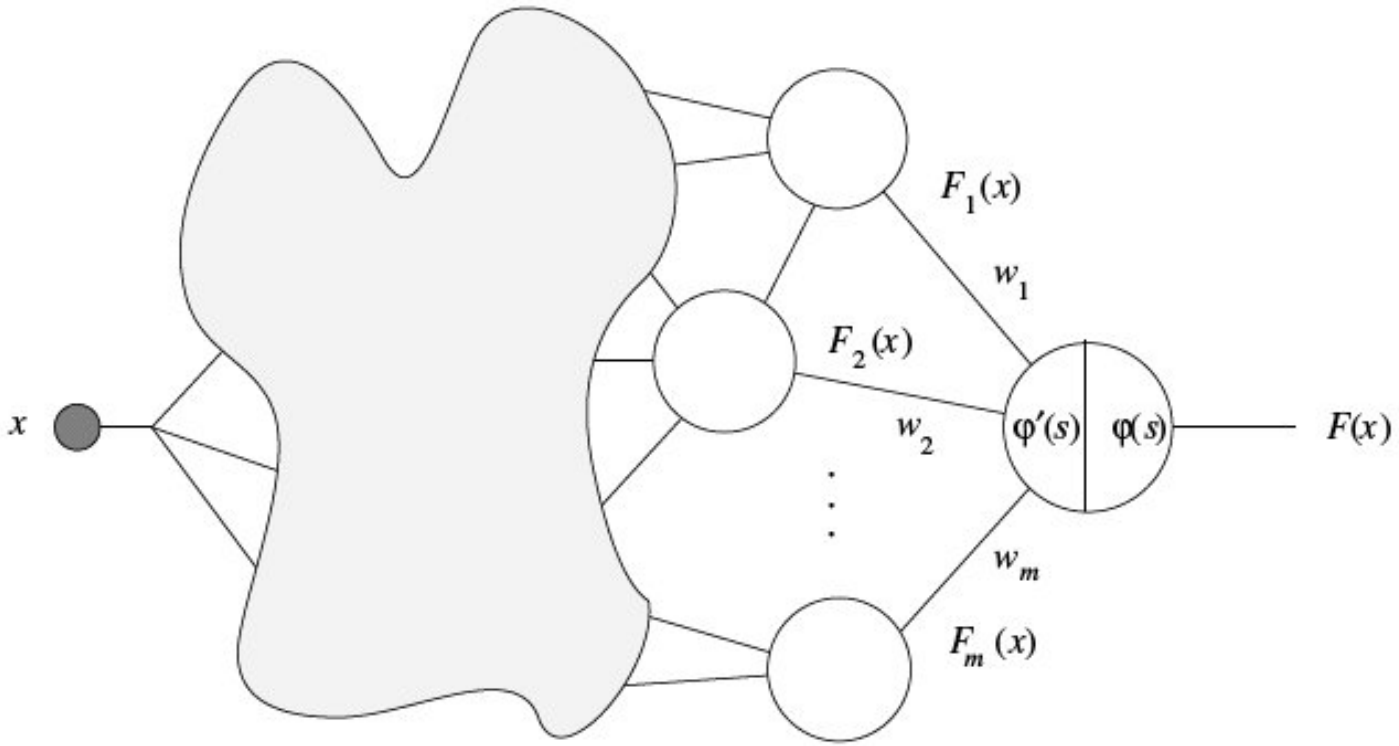
1. Podajemy do sieci wartość x; obliczane są funkcje aktywacji i ich pochodne; pochodne są zapamiętywane.
2. Stałą 1 podajemy do wyjścia sieci - informacja propaguje się wstecznie; informacja przychodząca do danej jednostki jest dodawana i mnożona przez wartość zapamiętaną w lewej części jednostki; wynik tranmitowany jest do jednostki na lewo; rezultat pojawiający się na jednostce wejściowej jest pochodną funkcji sieci po x.

- **Twierdzenie.** Powyższy algorytm oblicza prawidłowo pochodną po x funkcji F.

Algorytm propagacji wstecznej

Dowód. Indukcja, założenie różniczkowalności funkcji aktywacji, nie ma cykli.

- Pokazaliśmy, że algorytm działa dla jednostek połączonych szeregowo i równolegle oraz dla krawędzi z wagami.
- Zakładamy, że algorytm działa dla sieci n lub mniej węzłów.
- Rozważmy B-diagram $n + 1$ węzłów.

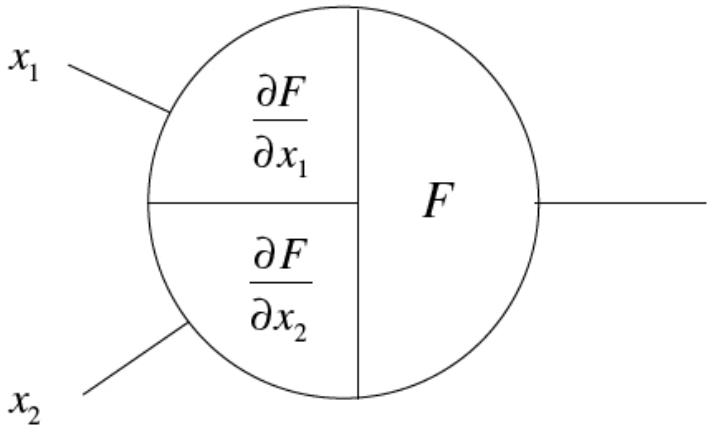


Algorytm propagacji wstecznej

- Funkcja sieci F obliczana z m wyjść $F_1(x), ..., F_m(x)$
- $F(x) = \varphi(w_1 F_1(x) + w_2 F_2(x) + ... + w_m F_m(x))$
- $F'(x) = \varphi'(s)(w_1 F_1'(x) + w_2 F_2'(x) + ... + w_m F_m'(x))$, $s = w_1 F_1(x) + w_2 F_2(x) + ... + w_m F_m(x)$
- F_1 - definiuje podsieć n lub mniej jednostek, dzięki *założeniu indukcyjnemu* możemy obliczyć pochodną przez wprowadzenie 1 i jej propagację wsteczną.
- Podobnie dla jednostek o wyjściach $F_2(x), ..., F_m(x)$.
- Jeżeli zamiast 1 wprowadzimy stałą $\varphi'(s)$ na wejściach otrzymamy: $w_1 F_1'(x)\varphi'(s)$,...,$w_m F_m'(x)\varphi'(s)$ (równoważne wprowadzeniu 1 do ostatniej jednostki sieci).
- Zatem propagacja wsteczna całą siecią daje $\varphi'(s)(w_1 F_1'(x) + w_2 F_2'(x) + ... + w_m F_m'(x))$, czyli pochodną - algorytm działa dla $n+1$ jednostek.

Algorytm propagacji wstecznej

- Domyślnie - wszystkie wejścia są dodawane przed obliczeniem funkcji aktywacji jednej zmiennej w każdym węźle.
- Możliwe funkcje aktywacji F wielu zmiennych - lewa strona jednostki przechowuje pochodne cząstkowe.
- W propagacji wstecznej - propagowana wartość jest mnożona przez odpowiednią pochodną i trasportowana odpowiednią krawędzią.



Algorytm propagacji wstecznej

- Propagacja wsteczna działa także dla sieci o kilku jednostkach wejściowych i kilku zmiennych.
  - 2 zmienne - 2 **podsieci** - każda łączy dane wejście z wyjściem.
  - Na wejściach otrzmujemy odpowiednie pochodne cząstkowe.
  - W praktyce nie ma przeszkody aby obie propagacje wsteczne nakładały się i można je zrobić w jednym kroku.

Algorytm propagacji wstecznej - uczenie

- Minimalizowana funkcja błędu E zależy od wag - należy uwzględnić wszystkie wagi za jednym razem.
- Przechowujemy też wyjścia każdej jednostki i w rozszerzonej sieci obliczającej błąd przeprowadzamy propagację wsteczną.
- w_{ij} - waga związana z jednostkami i oraz j - można jednostki następujące po niej traktować jako podsieć.
- Sygnał docierający do podsieci zaczynającej się na jednostce i to $o_i w_{ij}$ (o_i - zachowane wyjście jednostki i).
- **Propagacja wsteczna oblicza pochodną** $\partial E / \partial o_i w_{ij}$:

$$\frac{\partial E}{\partial w_{ij}} = o_i \frac{\partial E}{\partial o_i w_{ij}}$$

Algorytm propagacji wstecznej - uczenie

- Wszystkie podsieci obliczane są jednocześnie.
- **Propagowany wstecznie błąd** δ_j - skumulowany rezultat propagacji wstecznej na węźle j (*uwaga*: jest to więc pochodna a nie sam błąd).

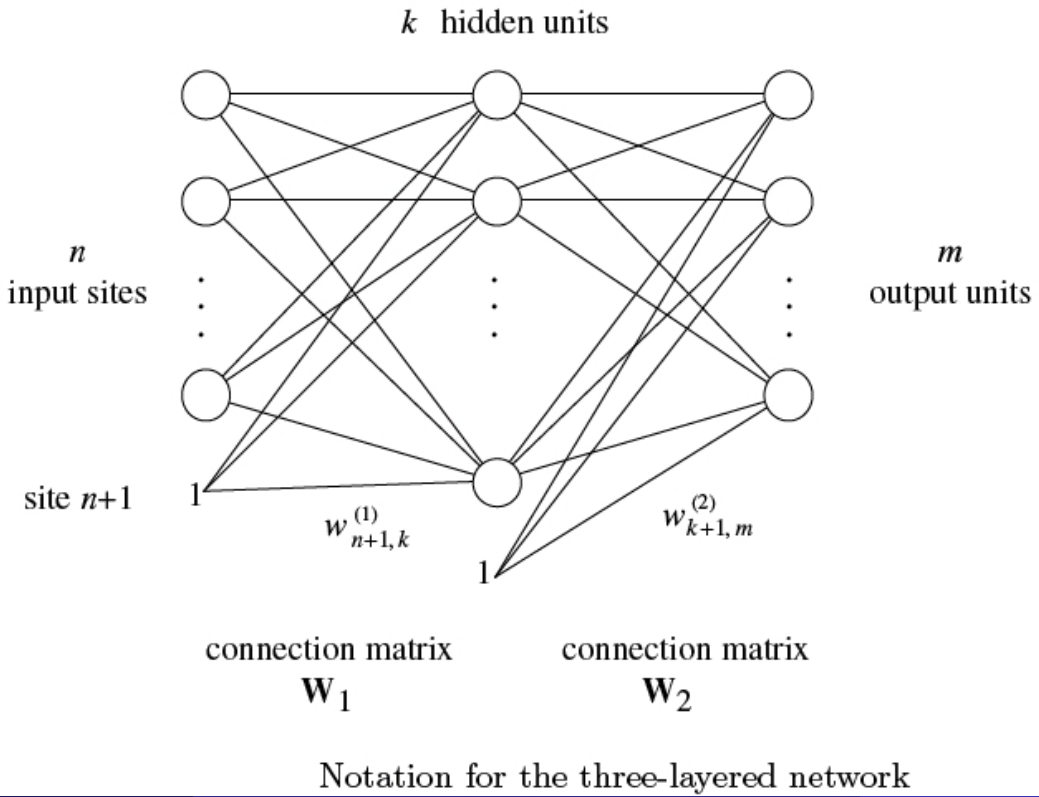
$$\frac{\partial E}{\partial w_{ij}} = o_i \delta_j$$

- Krok korygujący wagi - przekształca algorytm w metodę uczenia sieci neuronowych.

$$\Delta w_{ij} = -\gamma o_i \delta_j$$

Algorytm propagacji wstecznej - sieci warstwowe

- W przypadku sieci warstwowych (z pewną ilością warstw ukrytych) - wzory na aktualizacje wag można podać w terminach liniowych operacji algebraicznych.
- W poniższej sieci oznaczmy wagi między wejściami a jednostkami ukrytymi $w_{ij}^{(1)}$ a między jednostkami ukrytymi a wyjściami $w_{ij}^{(2)}$



Notation for the three-layered network

Algorytm propagacji wstecznej - sieci warstwowe

- Można wprowadzić próg $-\theta$ jako dodatkową krawędź z wagą.
  - Odpowiednie wagi (rysunek) to $w^{(1)}_{n+1,j}$ oraz $w^{(2)}_{k+1,j}$.
- Oznaczmy: $\overline{\mathbf{W}}_1$ - $(n+1) \times k$ macierz wag $w^{(1)}_{ij}$; podobnie $\overline{\mathbf{W}}_2$ - $(k+1) \times m$ macierz wag $w^{(2)}_{ij}$.
  - kreska - oznacza dołączenie θ
- n-wymiarowy wektor wejściowy $\mathbf{o} = (o_1, ..., o_n)$, rozszerzony $\hat{\mathbf{o}} = (o_1, ..., o_n, 1)$.
- **Wzbudzenie** j-tej jednostki ukrytej to

$$wzb_j = \sum_{i=1}^{n+1} w^{(1)}_{ij} \hat{o}_i.$$

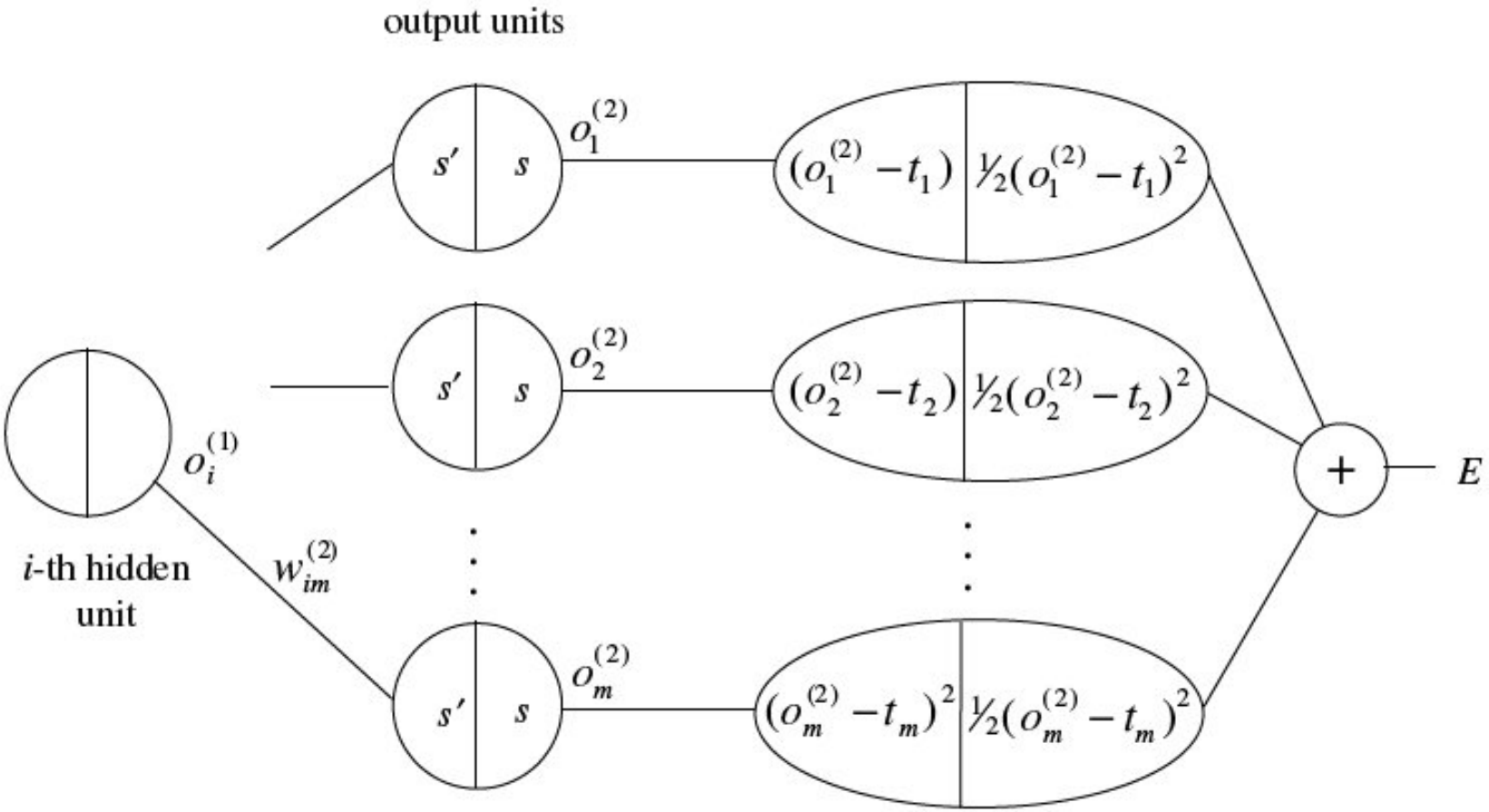
- Funkcja aktywacji - sigmoid - **odpowiednie wyjścia** to

$$o^{(1)}_j = s(wzb_j) = s\left(\sum_{i=1}^{n+1} w^{(1)}_{ij} \hat{o}_i\right).$$

Algorytm propagacji wstecznej - sieci warstwowe

- **Postać macierzowa**: $\mathbf{o}^{(1)} = s(\hat{\mathbf{o}}\overline{\mathbf{W}}_1)$.
- Dalej, wyjścia całej sieci: $\mathbf{o}^{(2)} = s(\hat{\mathbf{o}}^{(1)}\overline{\mathbf{W}}_2)$.
  - wyrażenia takie można napisać dla dowolnej ilości warstw
- Rozważmy **sieć rozszerzoną** i jedną parę wejście-wyjście $(\mathbf{x}, \mathbf{t})$ jako zbiór treningowy.
- Wyjścia oryginalnej sieci to $o_i^{(2)}$ obliczone jednostkami sigmoidalnymi.
- Dodanie odchyleń kwadratowych daje **błąd** E.
- W przypadku p przykładów **wejście-wyjście** - można utworzyć p takich sieci i dodać wyjścia dla otrzymania błędu dla całego zbioru treningowego.

Algorytm propagacji wstecznej - sieci warstwowe



Extended multilayer network for the computation of E

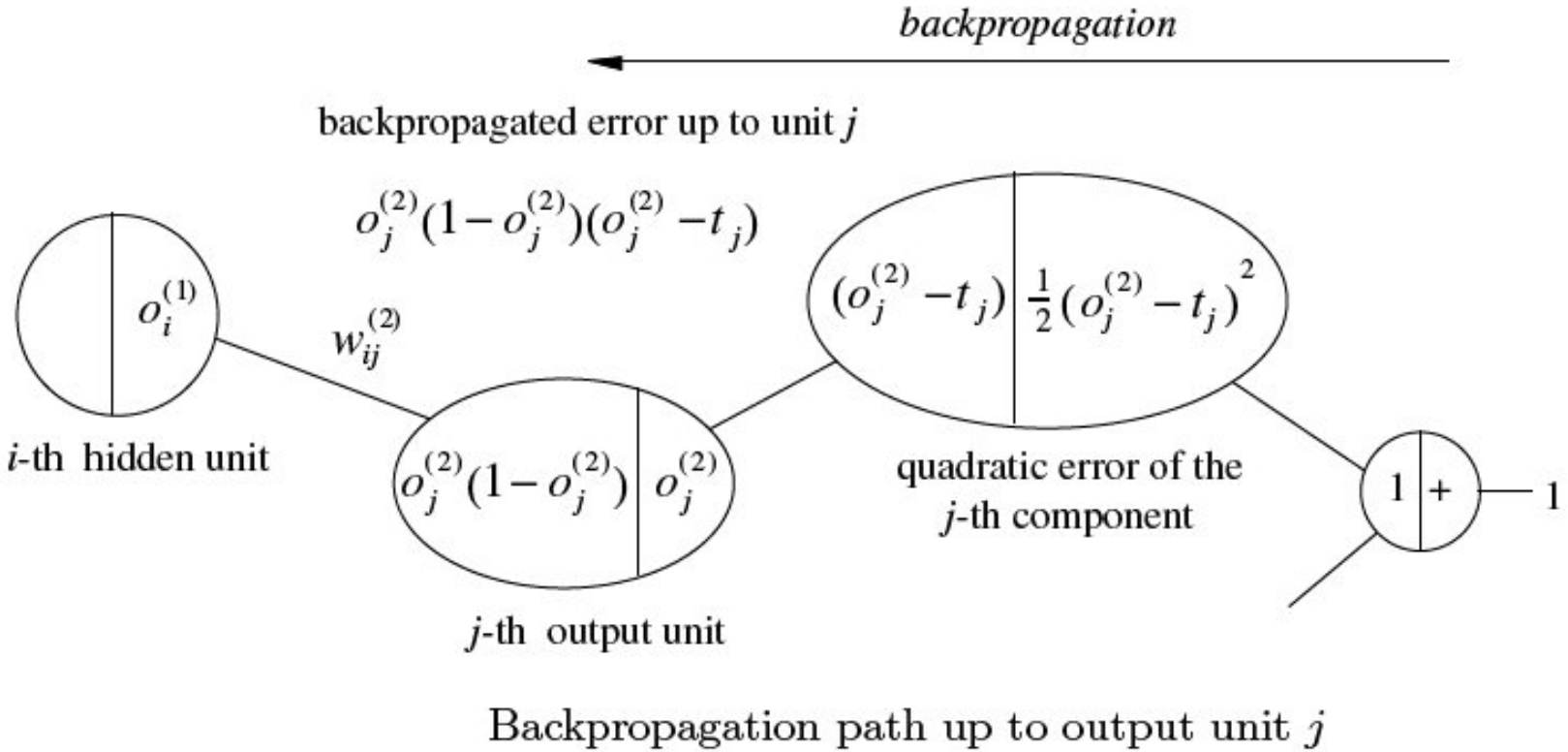
Algorytm propagacji wstecznej - sieci warstwowe

- Po wybraniu **przypadkowych wag** sieci - propagacja wsteczna oblicza korekcje dla wag.
- Można wyróżnić **4 kroki**:
  1. Normalne działanie sieci.
  2. Propagacja wsteczna do warstwy wyjściowej.
  3. Propagacja wsteczna do warstwy ukrytej.
  4. Aktualizacja wag sieci.
- Algorytm zatrzymuje się gdy błąd jest wystarczająco mały.

Krok 1: Podajemy wektor wejściowy $\mathbf{o}$ - obliczane i przechowywane są wektory $\mathbf{o}^{(1)}$ i $\mathbf{o}^{(2)}$. Pochodne funkcji aktywacji także są przechowywane.
Krok 2: Propagacja wsteczna do warstwy wyjściowej - ma dać zbiór pochodnych $\partial E/\partial w_{ij}^{(2)}$.

Algorytm propagacji wstecznej - sieci warstwowe

$$o_j^{(2)} = s(w_{ij}^{(2)} o_i^{(1)});\ s' = s(1-s)$$



Backpropagation path up to output unit j

Algorytm propagacji wstecznej - sieci warstwowe

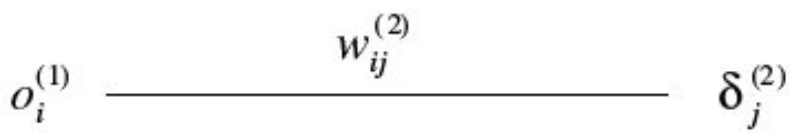
- Patrząc na powyższy rysunek można napisać propagowany wstecznie błąd na tym etapie:

$$\delta_j^{(2)} = o_j^{(2)}(1 - o_j^{(2)})(o_j^{(2)} - t_j) \quad (*)$$

- Odpowiednia pochodna to

$$\frac{\partial E}{\partial w_{ij}^{(2)}} = [o_j^{(2)}(1 - o_j^{(2)})(o_j^{(2)} - t_j)]o_i^{(1)} = \delta_j^{(2)} o_i^{(1)}$$

- Sytuacja w propagacji wstecznej - w jest zmienną:



Input and backpropagated error at an edge

Algorytm propagacji wstecznej - sieci warstwowe

Krok 3: Propagacja wsteczna do warstwy ukrytej - pochodna $\partial E/\partial w_{ij}^{(1)}$.

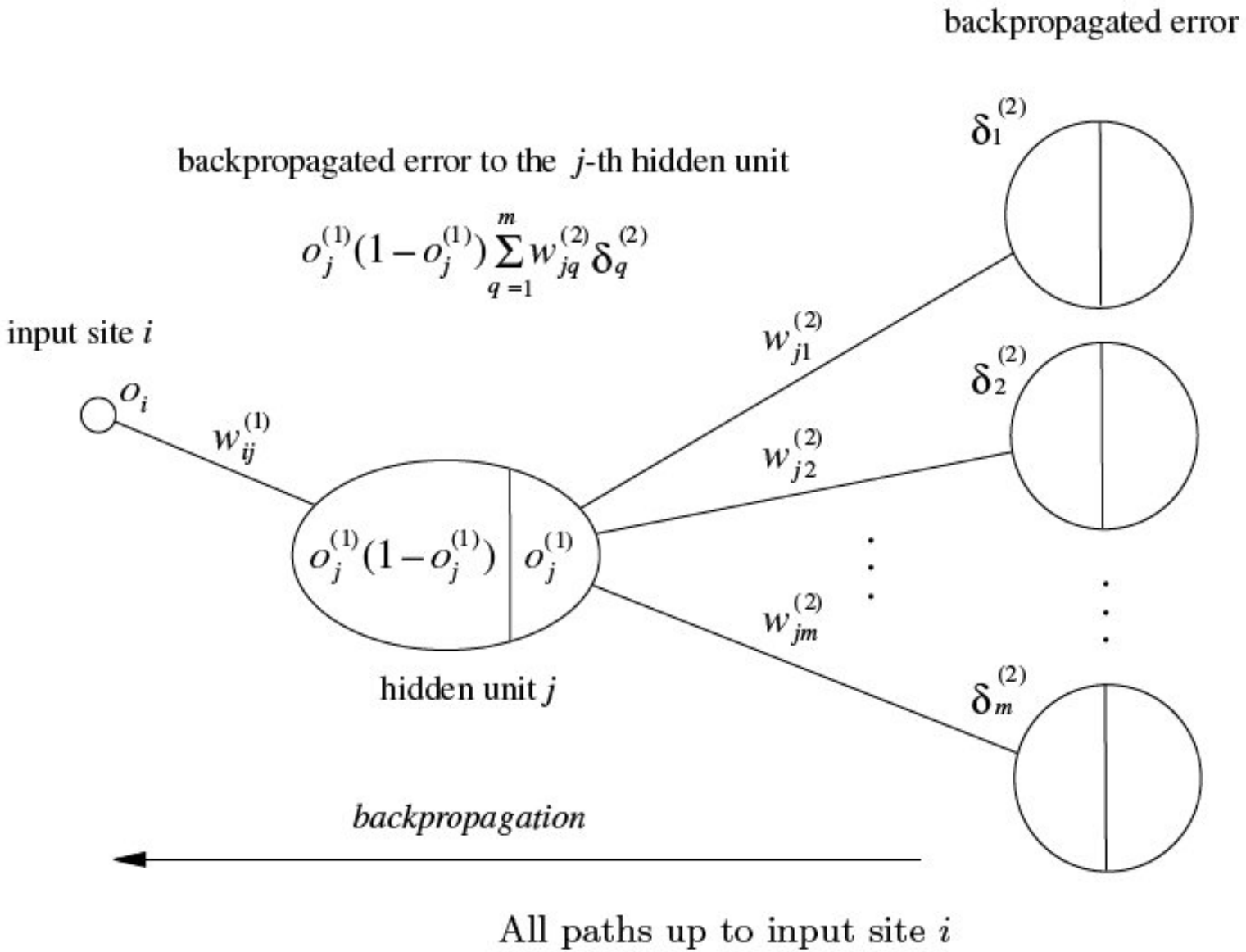
- Każda jednostka warstwy ukrytej - połączona z każdą jednostką warstwy wyjściowej krawędziami o wagach $w_{jq}^{(2)}$, dla $q = 1, .., m$.
- Błąd na danej jednostce j warstwy ukrytej - wszystkie ścieżki wsteczne (rysunek).

$$\delta_j^{(1)} = o_j^{(1)}(1 - o_j^{(1)}) \sum_{q=1}^{m} w_{jq}^{(2)} \delta_q^{(2)} \quad (**)$$

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \delta_j^{(1)} o_i$$

Proces taki sam dla dowolnej ilości warstw ukrytych.

Algorytm propagacji wstecznej - sieci warstwowe



Krok 4: Aktualizacja wag.

- Jeżeli γ jest wielkością kroku korekcji:

$$\Delta w_{ij}^{(2)} = -\gamma o_i^{(1)} \delta_j^{(2)}, \ i = 1, ..., k+1; \ j = 1, ..., m.$$

$$\Delta w_{ij}^{(1)} = -\gamma o_i \delta_j^{(1)}, \ i = 1, ..., n+1; \ j = 1, ..., k.$$

- $o_{n+1} = o_{k+1}^{(1)} = 1$
- **Ważne:** Wagi korygujemy po propagacji wstecznej w całej sieci - inaczej korekcje mieszają się z błędami i obliczane korekcje przestają odpowiadać gradientowi.
- Propagowany wstecznie błąd można też zdefiniować z minusem - wtedy nie potrzeba go już przy korekcji wag.

Algorytm propagacji wstecznej - sieci warstwowe

- W przypadku więcej niż jednej pary treningowej $p > 1$ - dla każdej z nich błąd oblicza się osobno.
- Przykładowo dla wagi $w_{ij}^{(1)}$:

$$\Delta_1 w_{ij}^{(1)}, \Delta_2 w_{ij}^{(1)}, ..., \Delta_p w_{ij}^{(1)},$$

- Korekcja zgodna z gradientem:

$$\Delta w_{ij}^{(1)} = \Delta_1 w_{ij}^{(1)} + \Delta_2 w_{ij}^{(1)} + ... + \Delta_p w_{ij}^{(1)}.$$

- Takie postępowanie - aktualizacje **off-line**.
- Często wagi aktualizuje się po prezentacji każdej pary - **on-line**.
  - Korekcje nie są dokładnie zgodne z kierunkiem gradientu.
  - Dla przypadkowego uczenia - kierunek oscyluje wokół dokładnego gradientu i średnio algorytm działa prawidłowo.
  - Dodanie pewnego szumu do algorytmu - pozwala uniknąć wpadania w lokane płytkie minima funkcji błędu.
  - Zbiór treningowy duży - obliczanie dokładnego gradientu jest drogie (1 krok off-line - wymaga prezentacji wszystkich przykładów).

Algorytm propagacji wstecznej - sieci warstwowe

Postać macierzowa.

- Przyjmujemy jak poprzednio: n wejść, k jednostek ukrytych, m jednostek wyjściowych.
- $\mathbf{o}^{(2)} = s(\hat{\mathbf{o}}^{(1)}\overline{\mathbf{W}}_2)$, $\mathbf{o}^{(1)} = s(\hat{\mathbf{o}}\overline{\mathbf{W}}_1)$
- $\mathbf{W}_1$ - $n \times k$, $\mathbf{W}_2$ - $k \times m$ - nie trzeba propagować wstecznie do wejść ze stałą wartością
- Odpowiednie pochodne zapisane w jednostkach wyjściowych i ukrytych - zapisujemy jako macierze:

Algorytm propagacji wstecznej - sieci warstwowe

$$\mathbf{D}_2 = \begin{pmatrix} o_1^{(2)}(1-o_1^{(2)}) & 0 & \cdots & 0 \\ 0 & o_2^{(2)}(1-o_2^{(2)}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & o_m^{(2)}(1-o_m^{(2)}) \end{pmatrix},$$

$$\mathbf{D}_1 = \begin{pmatrix} o_1^{(1)}(1-o_1^{(1)}) & 0 & \cdots & 0 \\ 0 & o_2^{(1)}(1-o_2^{(1)}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & o_k^{(1)}(1-o_k^{(1)}) \end{pmatrix}.$$

Algorytm propagacji wstecznej - sieci warstwowe

- Wektor pochodnych odchyleń kwadratowych:

$$\mathbf{e} = \begin{pmatrix} (o_1^{(2)} - t_1) \\ (o_2^{(2)} - t_2) \\ \vdots \\ (o_m^{(2)} - t_m) \end{pmatrix}$$

- m-wymiarowy **wektor propagowanego wstecznie błędu do jednostek wyjściowych**

$$\delta^{(2)} = \mathbf{D}_2\mathbf{e}. \quad (*)$$

- k-wymiarowy **wektor propagowanego wstecznie błędu do jednostek ukrytych**

$$\delta^{(1)} = \mathbf{D}_1\mathbf{W}_2\delta^{(2)}. \quad (**)$$

Algorytm propagacji wstecznej - sieci warstwowe

- Macierze współczynników korygujemy następująco

$$\Delta \overline{\mathbf{W}}_2^T = -\gamma \delta^{(2)} \hat{\mathbf{o}}^{(1)}$$

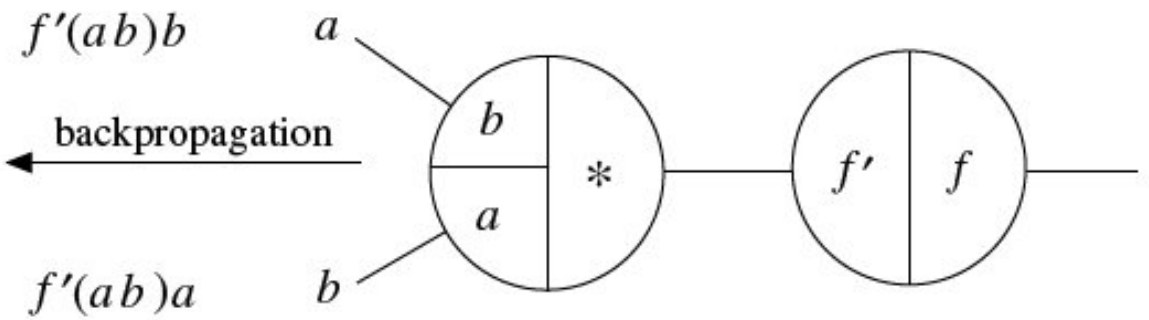
$$\Delta \overline{\mathbf{W}}_1^T = -\gamma \delta^{(1)} \hat{\mathbf{o}}.$$

- Transponowanie: $\overline{\mathbf{W}}_2 \in M_{(k+1)\times m}$ ale $M_{m\times 1} M_{1\times(k+1)} = M_{m\times(k+1)}$.
- **Obliczeniowo** - potrzeba mnożenia macierzy - są architektury komputerów wyspecjalizowane w takich operacjach.
- **Uogólnienie** do l warstw ($\overline{\mathbf{W}}_{i+1}$ - współczynniki pomiędzy wastwami i oraz $i+1$, zero - wejścia):
  - wyjścia: $\delta^{(l)} = \mathbf{D}_l \mathbf{e}$
  - i-ta warstwa obliczeniowa: $\delta^{(i)} = \mathbf{D}_i \mathbf{W}_{i+1} \delta^{(i+1)}$, $i = 1, ..., l-1$ lub

$$\delta^{(i)} = \mathbf{D}_i \mathbf{W}_{i+1} \cdots \mathbf{W}_{l-1} \mathbf{D}_{l-1} \mathbf{W}_l \mathbf{D}_l \mathbf{e}.$$

Algorytm propagacji wstecznej - sieci warstwowe - lokalność

- Używając B-diagramu można dowieść, że **dodawanie** jest jedyną funkcją scalającą zachowującą **lokaność** uczenia propagacją wsteczną.
- Sieci które rozważaliśmy - wykorzystywały informację lokalną - każda jednostka przesyła daną krawędzią (w którymkolwiek kierunku) i przechowuje tylko informacje bezpośrednio przychodzącą do niej.
- Przykład **mnożenia** jako funkcji scalającej - rysunek - w propagacji wstecznej wartość b musi być transportowana górną krawędzią a dostarczona została dolną.



Multiplication as integration function

Algorytm propagacji wstecznej - sieci warstwowe - lokalność

Twierdzenie. Dla jednostki o n wejściach $x_1, ..., x_n$ jedynie funkcje scalające postaci
$I(x_1, ..., x_n) = F_1(x_1) + F_2(x_2) + ... + F_n(x_n) + C$,
gdzie C - stała, zachowują lokalność algorytmu propagacji wstecznej w takim sensie, że na krawędzi $i \neq j$ nie jest jawnie przechowywana żadna informacja o x_j.

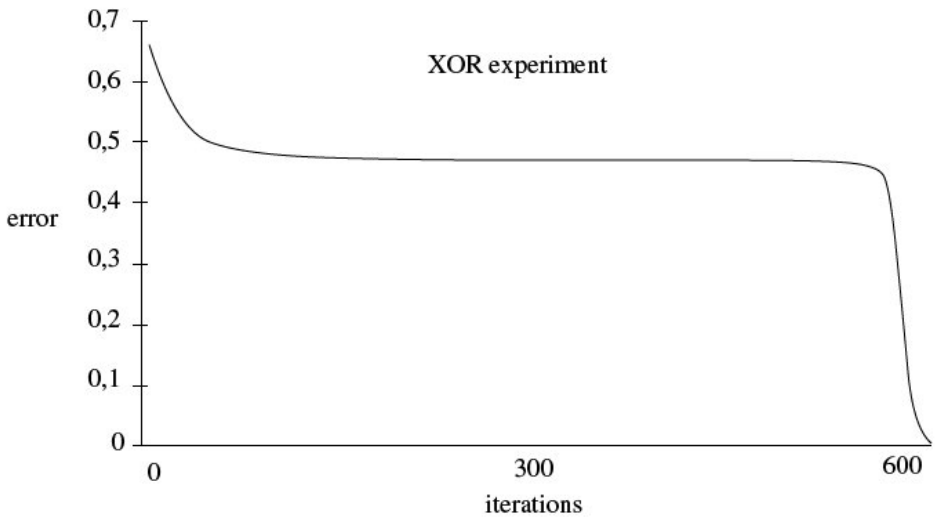
Dowód. Jeżeli w propagacji wstecznej jedynie funkcja $f_i(x_i)$ może być przechowywana w jednostce i tansmitowana przez i-tą krawędź wejściową. Wtedy $\frac{\partial I}{\partial x_i} = f_i(x_i), \quad i = 1, ..., n$. Zatem

$$I(x_1, x_2, ..., x_n) = F_1(x_1) + G_1(x_2, ..., x_n)$$
$$I(x_1, x_2, ..., x_n) = F_2(x_2) + G_2(x_1, x_3, ..., x_n)$$
$$I(x_1, x_2, ..., x_n) = F_n(x_n) + G_n(x_2, ..., x_{n-1}),$$

gdzie F_i - pochodna f_i, G - funkcje rzeczywiste $n-1$ argumentów. Jedyną możliwością jest funkcja podana w twierdzeniu.

Algorytm propagacji wstecznej - błąd

- Przy omawianiu sieci 1- i 2-warstwowych - błąd dla funkcji XOR.
- Rysunek - ewolucja błędu całkowitego podaczas treningu sieci z 3 jednostkami dla algorytmu propagacji wstecznej.
  - Po 600 krokach algorytm znajduje dokładne rozwiązanie.
  - Duża część to prawie płaski rejon - gdyby do aktywacji użyć funkcji schodkowej rejon ten byłby zupełnie płaski - problem. Sigmoid daje małe nachylenie.
  - Istnieją metody przyspieszania propagacji wstecznej.



Error function for 600 iterations of backpropagation

Algorytm propagacji wstecznej - sieci rekurencyjne

- Algorytm propagacji wstecznej - można rozszerzyć tak aby uwzględnić cykle.
- Wprowadzamy dyskretny czas: t - obliczenie, $t+1$ - transmisja wyniku.
- Wartości wejściowe mogą być stałe lub zmienne.
- Odpowiedni algorytm to nazywamy **propagacją wsteczną w czasie** (BPTT - *backpropagation through time*).
- Jeżeli stan sieci stabilizuje się w czasie można rozważać nieskończony czas.
  - Istnienie takiego stanu zależy od funkcji aktywacji i topologii sieci.

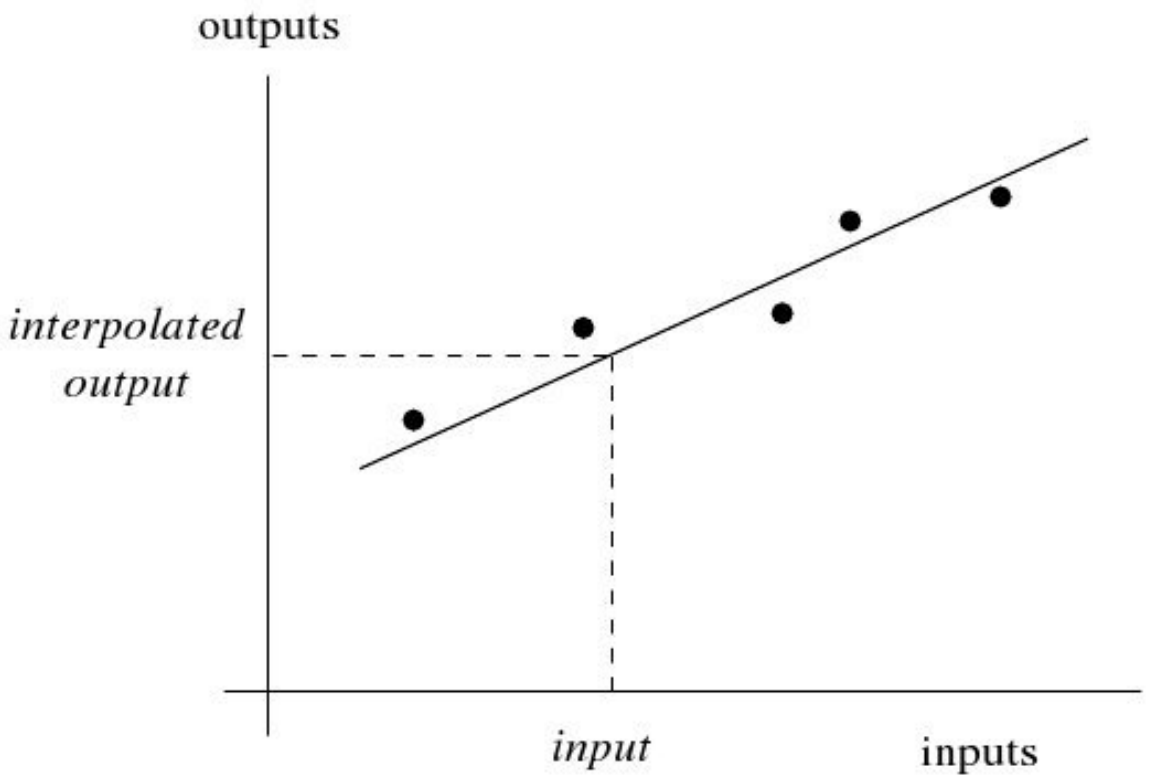
Algorytm propagacji wstecznej - historia

- Koniec lat 60-tych - dla skomplikowanych wielowarstwowych architektur sieci potrzebne było wyjście poza perceptron - nowy algorytm uczenia.
- Odpowiedni algorytm istniał w dziedzinie kontroli optymalnej - wariacyjna optymalizacja funkcji kosztu z dodatkowymi warunkami.
- Ok. 1960 - Kelly i Bryson - rozwineli numeryczne metody rozwiązywania tego problemu - rekursywne obliczanie gradientu funkcji kosztu.
- 1962 - Dreyfus - przedstawił problem w postaci wielo-etapowego systemu i podał wyprowadzenie tego co dzisiaj nawywamy algorytmem propagacji wstecznej.
- Algorytm był ponownie rozwijany w dziedzinach statystyki, rozpoznawania wzorców i sztucznej inteligencji.

Sieci neuronowe i statystyka - regresja

- Sieci neuronowe mogą służyć do **znalezienia funkcji** najlepiej dopasowanej do danych.
- Co znaczy **optymalnie**? Chcemy **odzwzorować znane wejscia** na wyjscia z największa możliwą dokładnością ale jednocześnie sieć powinna **uogolniać**, czyli nieznanym wejściom przypisywać wartości = interpolować.

Sieci neuronowe i statystyka - dobre uogólnienie

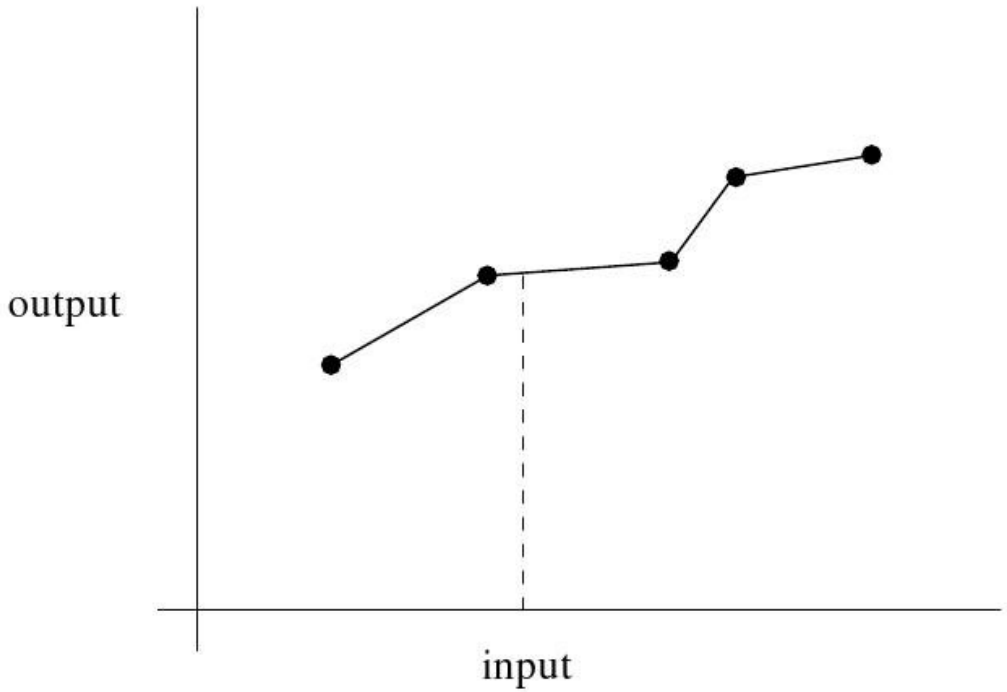


Linear approximation of the training set

- Kropki = zbiór treningowy

Sieci neuronowe i statystyka - dobre uogólnienie

- Funkcja kawałkami liniowa - gorsza, gdyż odwzorowuje błedy zawarte w danych mimo że zbiór treningowy odzworowuje bezbłednie.



Approximation of the training set with linear splines

Sieci neuronowe i statystyka - dobre uogólnienie

- Przybliżanie funkcji ma dwa sprzeczne cele:
  1. minimalizacja błędu uczenia
  2. minimalizacja błędu nieznanych wejść
- Liczba stopni swobody wyznacza **plastyczność** sieci (= zdolność aproksymacji zbioru treningowego).
- Większa plastyczność to mniejszy błąd treningowy ale może spowodować wzrost błedu na zbiorze testowym.
- Nie ma uniwersalnej metody wyznaczenia optymalnej liczby parametrów, zależy ona od **struktury problemu**.
- Najlepiej gdy topologia sieci wybrana jest w zależnosci od znanych relacji wejscie-wyjscie;
  - przykład powyżej - jeżeli analiza teoretyczna problemu potwierdza zależność liniową - wiadomo że przybliżenie liniowe będzie najlepsze.

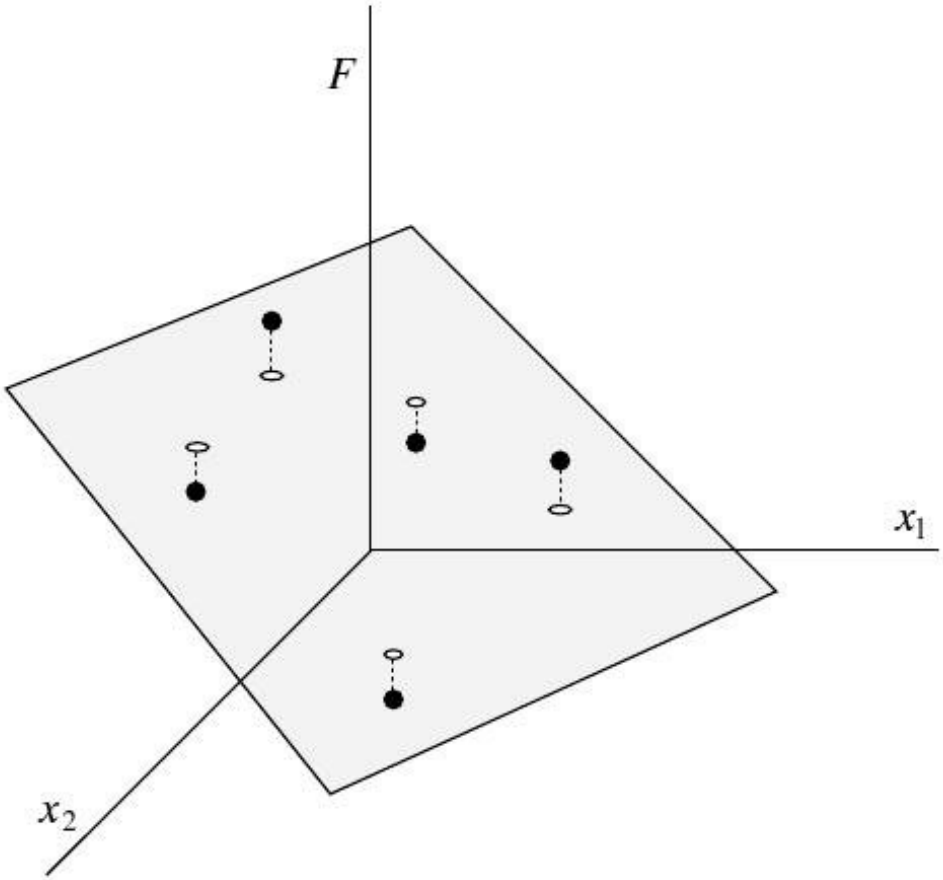
Sieci neuronowe i statystyka - regresja liniowa

- **Asocjator liniowy** - dodaje ważone wejścia (można go uważać za część neuronu nieliniowego); n-wymiarowe wejście $(x_1, x_2, ..., x_n)$, wektor wag $(w_1, w_2, ..., w_n)$:

$$y = w_1 x_1 + w_2 x_2 + ... + w_n x_n.$$

- Funkcja wyjściowa jest równaniem hiperpowierzchni w (n+1)-wymiarowej przestrzeni.
- **Problem uczenia** asocjatora liniowego - odwzorować wyjście dla wektorów zbioru treningowego (czarne punkty).
- Należy zminimalizować błąd (odległość zbioru od hiperpowierzchni) - algorytm propagacji wstecznej.

Sieci neuronowe i statystyka - regresja liniowa



Learning problem for a linear associator

Sieci neuronowe i statystyka - regresja liniowa

- Rozważmy zbiór treningowy $T = \{(\mathbf{x}^1, a_1), ..., (\mathbf{x}^m, a_m)\}$; wejściami są n-wymiarowe wektory $\mathbf{x}^1, ..., \mathbf{x}^m$, a wyjściami liczby rzeczywiste $a_1, ..., a_m$.
- Należy znaleźć wektor $\mathbf{w}$ minimalizujący błąd kwadratowy:

$$E = \frac{1}{2}\left[\left(a_1 - \sum_{i=1}^{n} w_i x_i^1\right)^2 + ... + \left(a_m - \sum_{i=1}^{n} w_i x_i^m\right)^2\right]$$

x_i^j - i-ta składowa wektora $\mathbf{x}^j$.

- Składowe gradientu

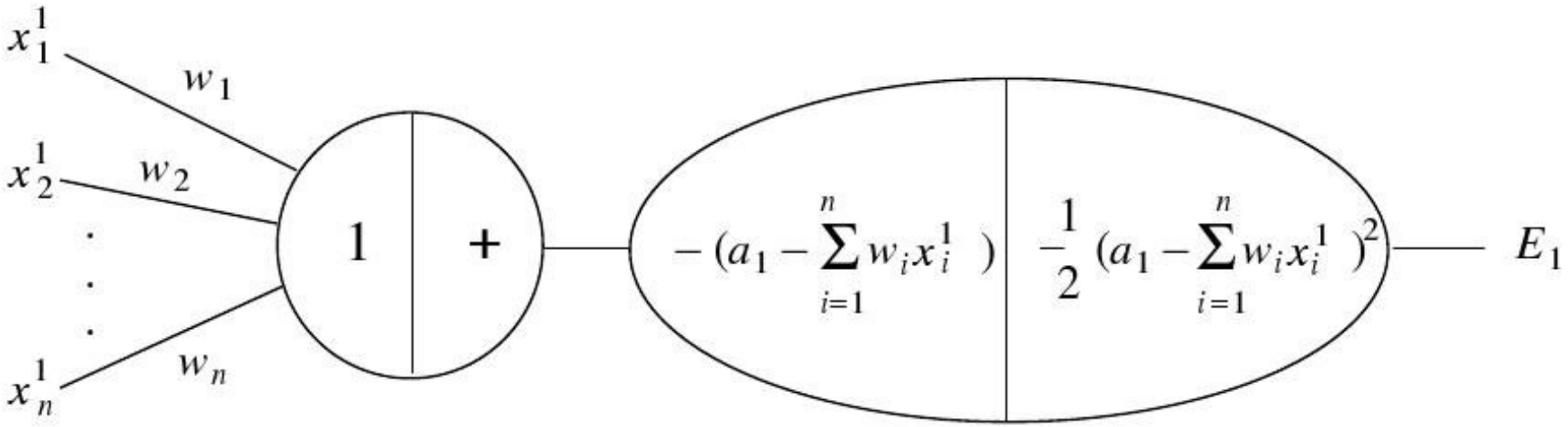
$$\frac{\partial E}{\partial w_j} = -\left(a_1 - \sum_{i=1}^{n} w_i x_i^1\right) x_j - ... - \left(a_m - \sum_{i=1}^{n} w_i x_i^m\right) x_j^m$$

$j = 1, 2, ..., n$.

Sieci neuronowe i statystyka - regresja liniowa

- Rozwiązanie - analityczne ($\nabla E = \mathbf{0}$) lub iteracyjne metodą gradientową.
- Funkcja błędu jest kwadratowa - minimum globalne można znaleźć zaczynając od dowolnie wybranych wag i używając w każdym kroku poprawki $\Delta w_j = -\gamma \partial E / \partial w_j$
- B-diagram asocjatora liniowego (użyto wektora $\mathbf{x}^1$):

Sieci neuronowe i statystyka - regresja liniowa



Backpropagation network for the linear associator

Sieci neuronowe i statystyka - regresja liniowa

- Problem znalezienia optymalnych wag dla danego zbioru treningowego dla asocjatora liniowego w statystyce - **wielokrotna regresja liniowa**.
- Szukamy wag takich, że wartości y można obliczyć z x z błędem ε_i

$$y = w_0 + w_1 x_1^i + w_2 x_2^i + \ldots + w_n x_n^i.$$

- Należy zminimalizować błąd kwadratowy $\sum_{i=1}^{n} \varepsilon_i^2$ - metoda algebraiczna.
- $\mathbf{X}$ - macierz $m \times (n+1)$

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^1 & \ldots & x_n^1 \\ 1 & x_1^2 & \ldots & x_1^2 \\ \ldots & \ldots & \ldots & \ldots \\ 1 & x_1^m & \ldots & x_1^m \end{pmatrix}$$

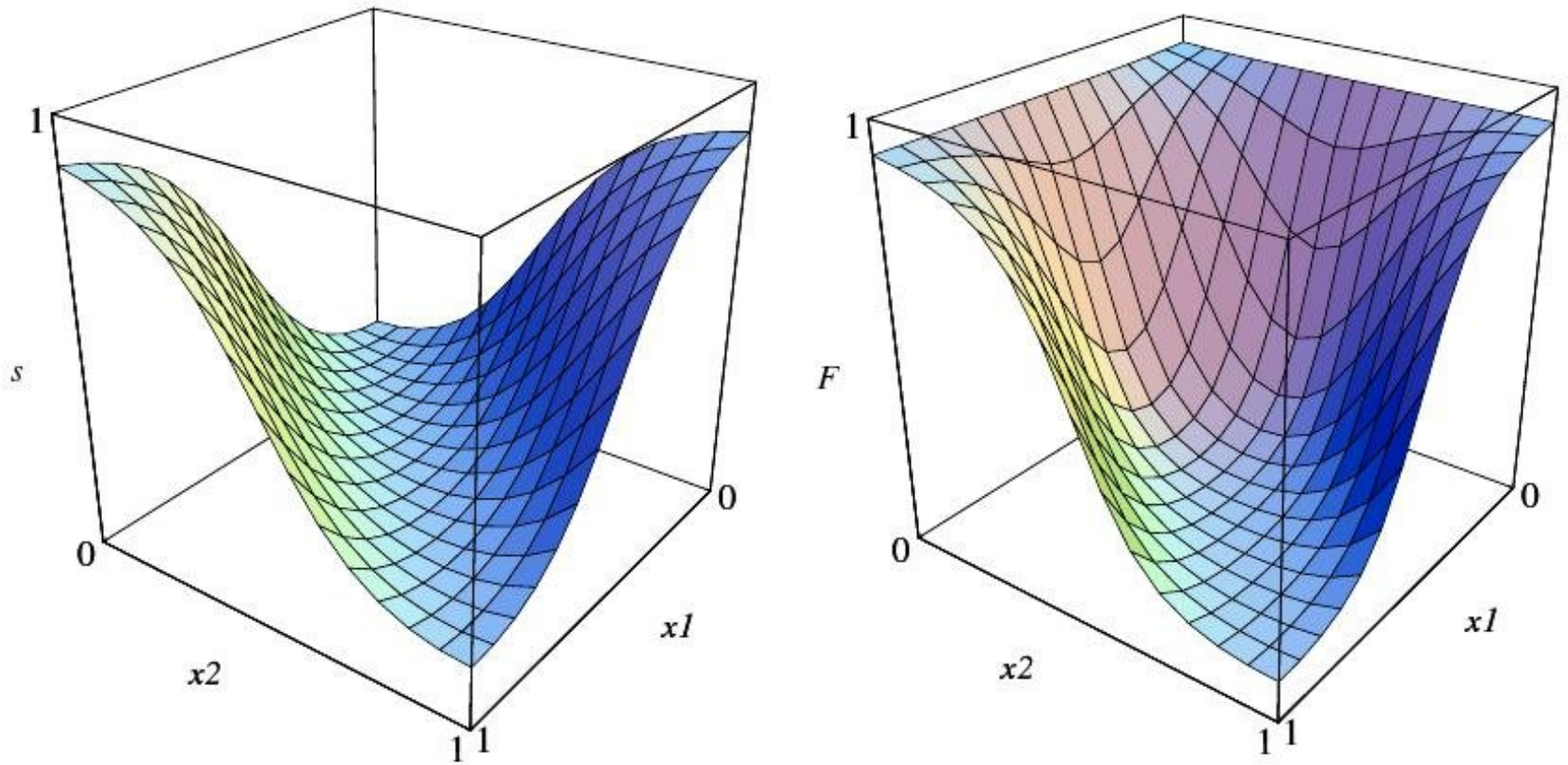
Sieci neuronowe i statystyka - regresja liniowa

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \dots \\ a_m \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_n \end{pmatrix}, \quad \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \dots \\ \varepsilon_m \end{pmatrix}$$

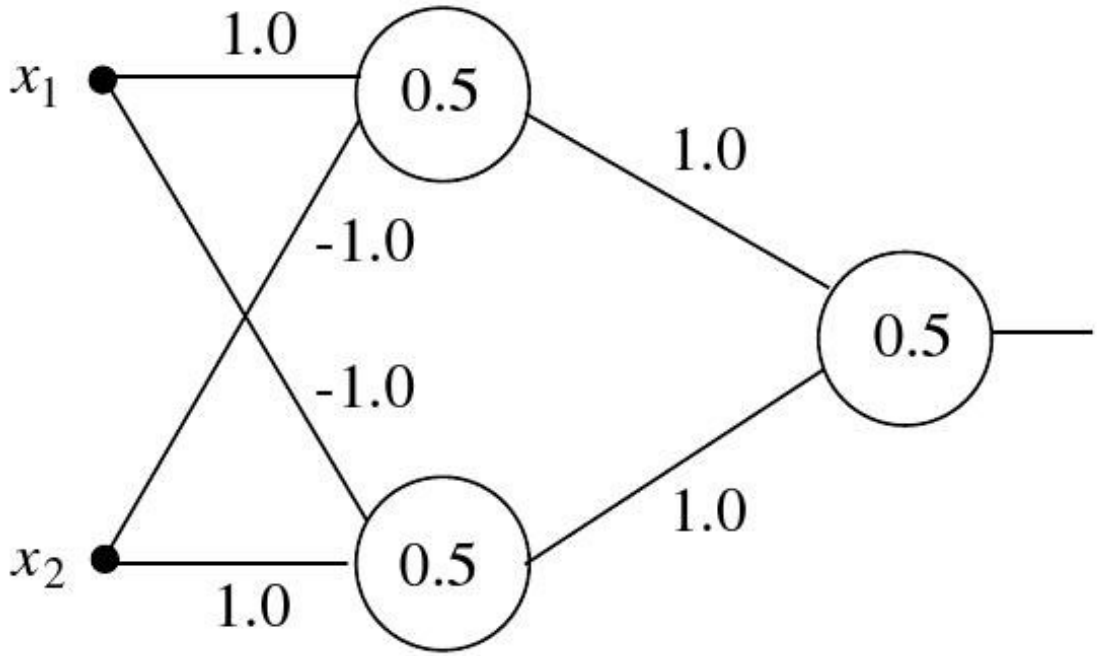
- $$\mathbf{a} = \mathbf{X}\mathbf{w} + \varepsilon$$
- Minimalizujemy $\|\varepsilon\|^2 = (\mathbf{a} - \mathbf{X}\mathbf{w})^T(\mathbf{a} - \mathbf{X}\mathbf{w}) \Rightarrow \mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{a}$.

Sieci neuronowe i statystyka - neurony nieliniowe

- Sigmoid jako funkcja aktywacji zmienia funkcję na wyjściu sieci.
- Przykłady ciągłego wyjścia małych sieci jednostek sigmoidalnych:



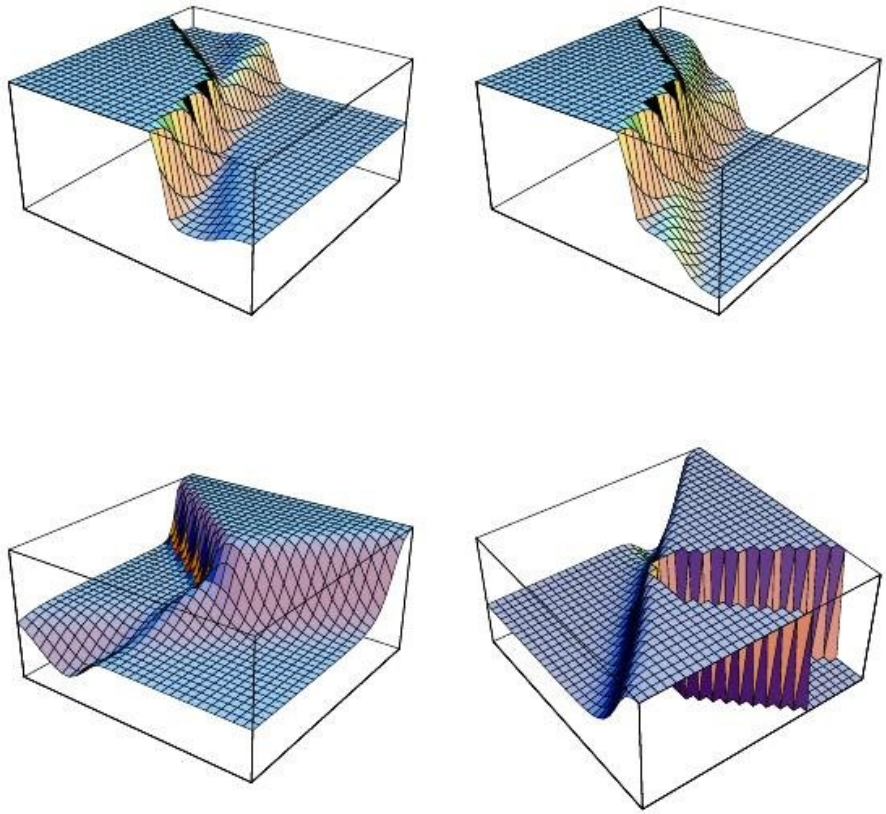
Output of networks for the computation of XOR (left) and NAND (right)



A three-layered network for the computation of XOR

Sieci neuronowe i statystyka - neurony nieliniowe

- Znacznie bardziej skomplikowane funkcje mogą być obliczane przez dość proste sieci; na rysunku przykłady dla sieci z 3 i 4 ukrytymi neuronami:



Network functions of networks with one hidden layer

Sieci neuronowe i statystyka - neurony nieliniowe

- Niewielkie zmiany parametrów sieci - duża zmienność kształtu funkcji na wyjściu.
- Hipoteza - każda funckcja ciągła może być przybliżona.
- Liczba pofałdowań wykresu funkcji zależy od liczby ukrytych neuronów; podobnie wielomiany - stopień wyznacza liczbę stopni swobody.

Sieci neuronowe i statystyka - regresja logistyczna

- **Regresja logistyczna** - typ regresji nielinowej pozwala obliczyć prawdopodobnieństwo zdarzenia gdy wyjściowo dana jest funkcja Boola; stosowana w biologii i ekonomii.
- Jeżeli T jest zbiorem treningowym, szukamy n-wymiarowego wektora $\mathbf{w}$ minimalizującego błąd (s - sigmoid):

$$E = \sum_{i=1}^{m} (a_i - s(\mathbf{w} \cdot \mathbf{x}^i))^2$$

- Algorytm propagacji wstecznej rozwiązuje ten problem.
- Przybliżenie można znaleźć odwracając sigmoid:

$$E' = \sum_{i=1}^{m} (s^{-1}(a_i) - \mathbf{w} \cdot \mathbf{x}^i)^2$$

- Asocjator liniowy musi przybliżyć wyjścia (transformacja logit - ln stosunku liczby sukcesów do porażek)

$$a'_i = s^{-1}(a_i) = \ln\left(\frac{a_i}{1 - a_i}\right), \quad i = 1, 2, ..., m.$$

- Podejście to upraszcza problem kosztem dodatkowego błędu (tradycyjna propagacja wsteczna jest dokładniejsza; transformacja logit modyfikuje wagi).

Sieci neuronowe i statystyka - błąd przewidywania

- Najważniejsza kwestia w przybliżaniu funkcji to **błąd przewidywania** (uogólnienia) nowych wartości prezentowanych sieci.
- Regresja liniowa - intensywnie studiowana, istnieją wzory na błąd i jego wariancję.
- Regresja nieliniowa - uzyskanie wzorów bardzo trudne lub niemożliwe.
- Za spodziewany błąd uogólnienia można by przyjąć pierwiastek średniego błędu uczenia (N danych punktów, E - całkowity błąd kwadratowy po zbiorze treningowym):

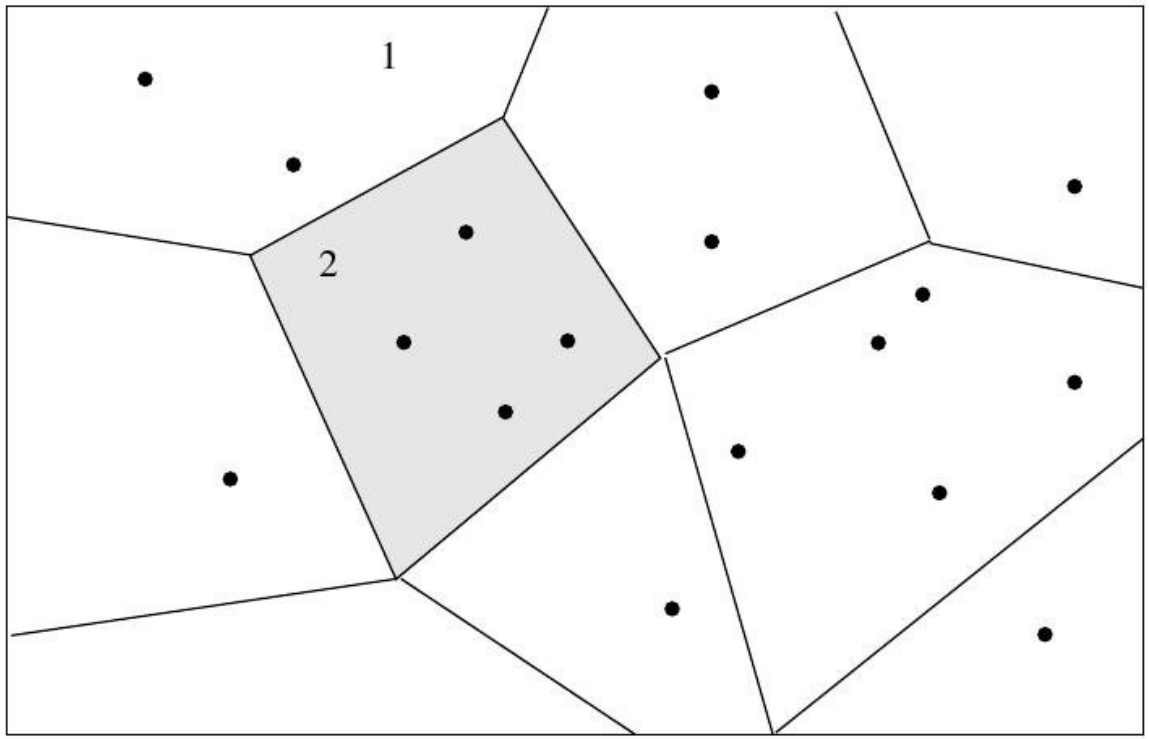
$$\tilde{E} = \sqrt{E/N}.$$

- Tak obliczony błąd jest jednak **mniejszy** niż prawdziwy - sieć została dostosowana właśnie do tych elementów.

Sieci neuronowe i statystyka - błąd przewidywania

- **Metoda bootstrap** - Efron, 1979; kluczowa obserwacja - jeżeli dane mogą służyć do znalezienia linii regresji, to zawierają też informacje o przyszłych wejściach sieci.
  - Gdyby dostępne były nowe dane można by obliczyć z nich błąd uogólnienia.
  - Symuluje się powyższą sytuację tworząc różne zbiory treningowe z istniejących danych przez próbkowanie losowe.
- Zbiór danych $X = \{x_1, x_2, ..., x_n\}$, $\hat{\theta}$ - wielkość statystyczna obliczona z tych danych (jest to estymator prawdziwej wartości θ po całym zbiorze).
- Problem: obliczenie odchylenia standardowego $\hat{\theta}$.
- Dane pochodzą z nieznanego rozkładu prawdopodobieństwa, który przybliżamy **próbkując losowo ze zbioru X ze zwracaniem**.
- Tak wygenerowane nowe zbiory danych X^* dają nowe wartości $\hat{\theta}^*$.
- Odchylenie standardowe θ jest przybliżane odchyleniem standardowym $\hat{\theta}^*$.

Sieci neuronowe i statystyka - błąd przewidywania



Distribution of data in input space

- Obszar 2 będzie reprezentowany dwa razy częściej w generowanych próbkach niż 1 - zatem próbkowanie zachodzi z pewnego przybliżenia dokładnego rozkładu (**założenie**: rozkład danych jest wystarczająco dobrym przybliżeniem dokładnego rozkładu prawdopodobieństwa).

Sieci neuronowe i statystyka - błąd przewidywania

- Przy dobrym przybliżeniu rozkładu można obliczyć odchylenia standardowe wartości funkcji określonych na zbiorze danych.
- **Bootstrap algorytm**:
  - Wybierz N niezależnych próbek $\mathbf{x}^{*1}, \mathbf{x}^{*2}, ..., \mathbf{x}^{*N}$ (każda po n wartości wybranych z zamianą ze zbioru X)
  - Oblicz żądaną wielkość statystyczną S odpowiadającą każdej z próbek

$$\hat{\theta}^*(b) = S(\mathbf{x}^{*b}), \quad b = 1, 2, ..., N.$$

  - Oblicz odchylenie standardowe na podstawie tych N próbek

$$\hat{s}_N = \left(\sum_{b=1}^{N} [\hat{\theta}^*(b) - \tilde{\theta}]^2 / (N-1) \right)^{1/2},$$

gdzie $\tilde{\theta} = \sum_{b=1}^{N} \hat{\theta}^*(b)/N$.

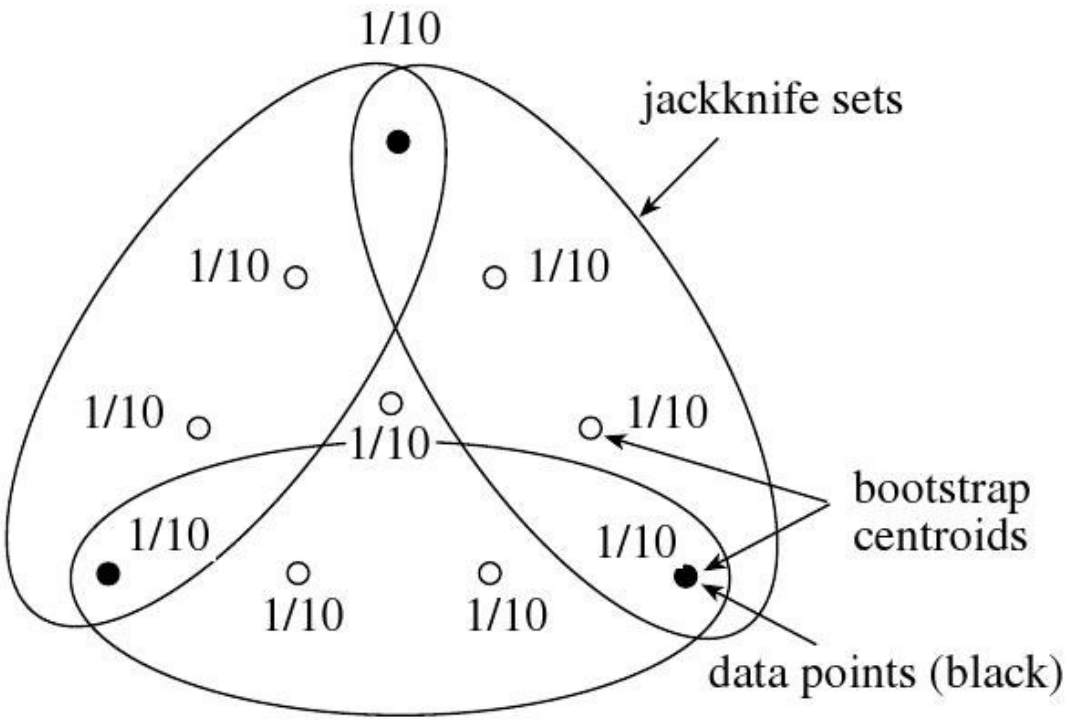
Sieci neuronowe i statystyka - błąd przewidywania

- Zwykle uczenie sieci neuronowej jest niedeterministyczne - metoda gradientowa prowadzi do jednego z wielu obecnych minimów.
- Uczenie na różnych zbiorach danych dać całkowicie **różne wagi** a więc i różne oszacowania średniego kwadratowego odchylenia.
- Jeżeli chcemy analizować **ogólną sytuację** to jest to prawidłowe bo w ogólności nie wiadomo, które lokalne minimum zostanie osiągnięte z danego zbioru treningowego.
- Jeżeli jednak interesuje nas konkretne **minimum** uczenie musi zapewnić zbieżność do **podobnego** minimum funkcji błędu (kształt funkcji błędu zależy od zbioru treningowego).

Sieci neuronowe i statystyka - błąd przewidywania

- **Metoda jackknife** - może być uważana za poprzednika metody bootstrap; z oryginalnych danych (n) generujemy nowe próbki przez odrzucanie jednego punktu.
- Z pozostałych $n-1$ obliczamy wielkość statystyczną a rezultatem końcowym jest średnia po n różnych zbiorach danych.
- Na rysunku - porównanie w przypadku 3 punktów dla średniego położenia; w metodzie bootstrap 10 możliwych zbiorów (kółka z prawdopodobieństwmi); w metodzie jackknife - 3 różne zbiory (elipsy); średnie z obydwu metod w tym prostym przypadku pokrywają się.

Sieci neuronowe i statystyka - błąd przewidywania



Comparison of the bootstrap and jackknife sampling points for $n = 3$

Sieci neuronowe i statystyka - błąd przewidywania

- **d-jackknife** - odrzucamy nie jeden ale d punktów.
- Wartości średnie i odchylenia standardowe obliczamy jak w metodzie bootstrap.
- **Metoda krzyżowego potwierdzenia** (cross-validation) - ze zbioru treningowego rezerwuje się $5 - 10\%$ par wejście-wyjście, których nie używa się do uczenia sieci; wytrenowana sieć jest testowana tym zerezerwowanym zbiorem a średni błąd uzyskany z tych par jest oszacowaniem dokładnej wartości błędu.
  - Oszacowanie jest dobre jeżeli zarówno zbiór treningowy jak i testowy dobrze oddają rzeczywisty rozkład prawdopodobieństwa.
- k-**krotne krzyżowe potwierdzenie** - metoda ulepszona; dzielimy zbiór danych na k przypadkowe podzbiory tej samej wielkości i uczymy sieć k-krotnie za każdym razem pomijając jeden z podzbiorów a następnie testuje się na nim błąd (ponownie średnia tych k wielkości jest oszacowaniem średniego błędu).
- Metody te są czasochłonne - sieć uczymy wielokrotnie; problem nadaje się do obliczeń równoległych.

Sieci neuronowe i statystyka - sieci klasyfikujące

- Sieci wielowarstwowe - popularne narzędzie w robotyce, rozpoznawaniu mowy lub wzorców, kodowaniu.
- Pewne zagadnienia są rozwijane dzięki teorii a inne dzieki dużej ilości danych (ale nie ma rozwiniętej teorii).
- Sieci neuronowe mogą rozpoznać zależności statystyczne oraz dostosowywać się do zmiennych warunków.
- Statystyczne własności sieci znalazły zastosowanie w sieciach klasyfikujących.

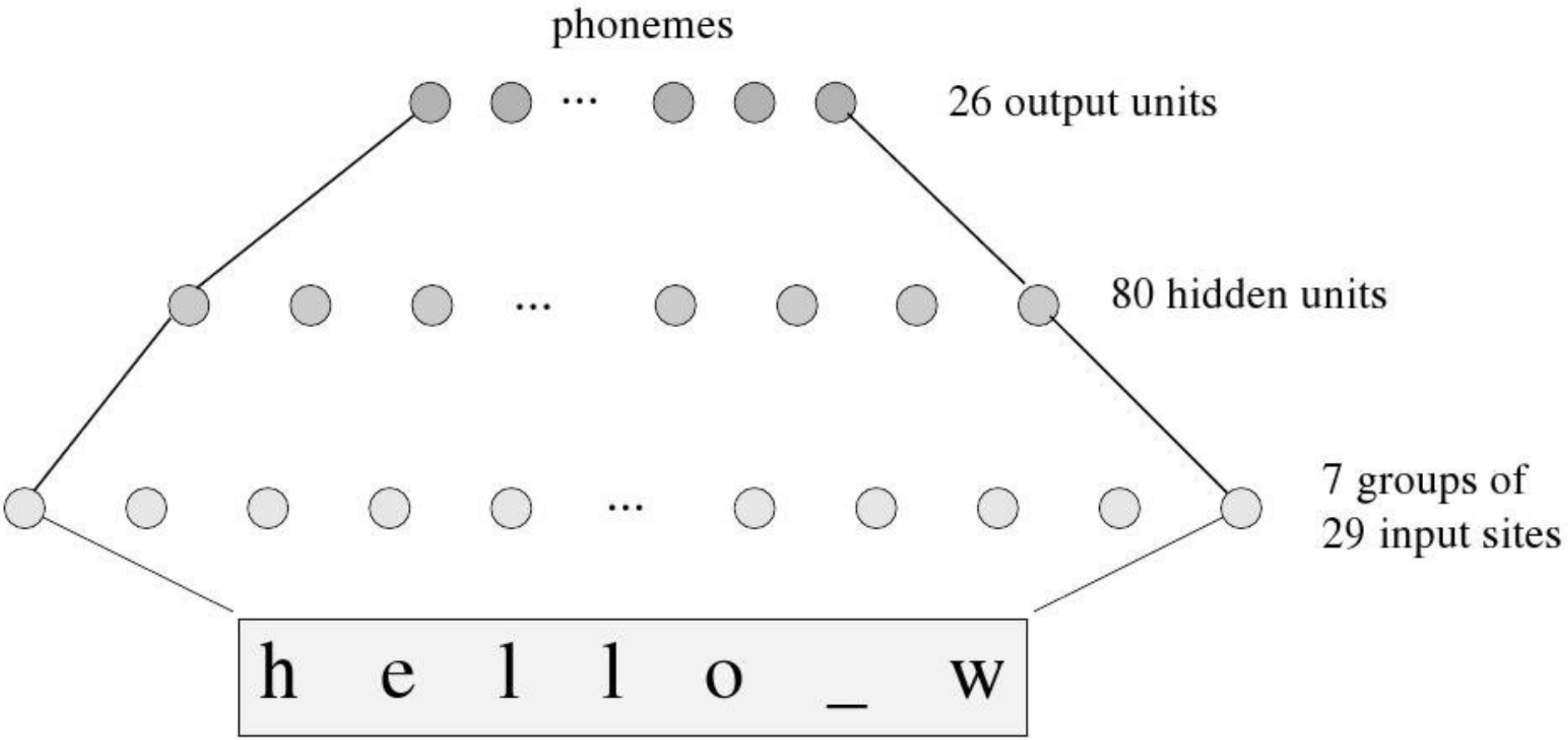
Sieci neuronowe i statystyka - sieci klasyfikujące

- W wielu zastosowaniach wejście musi być zaklasyfikowane do jednej z k klas - problemy takie rozwiązują **sieci klasyfikujące** o k neuronach wyjściowych.
- Gdy na wejściu podawany jest wektor $\mathbf{x}$ wyjście skojarzone z prawidłową jego klasyfikacją musi dać wartość 1 a wszystkie pozostałe 0.
- Jednym z zastosowań są **systemy syntezy mowy** - przekszałcają znaki do fonemów przez zastosowanie reguł transforamcji lingwistycznych (jest wiele takich reguł i związków między nimi; nietrywialne).

Sieci neuronowe i statystyka - sieci klasyfikujące

- Sejnowski i Resenberg (1980) opracowali sieć z propagacją wsteczną dającą dobrą jakość mowy bez jawnego zastosowania transformacji lingwistycznych (**NETtalk**).
  - System skanuje tekst używając 7 znakowego przesuwanego okienka.
  - Każdy znak jest kodowany jako jedna z 29 liter ($7 \times 29 = 203$ wejścia; jedno z wejść = 1, pozostałe 0).
  - Na wyjściu otrzymuje się fonem dla środkowego znaku (po 3 znaki po lewej i prawej tworzą kontekst dla znaku).
  - Sieć jest podłączona do elektronicznego syntezatora mowy o 26 fonemach (późniejsze wersje - więcej).

Sieci neuronowe i statystyka - sieci klasyfikujące



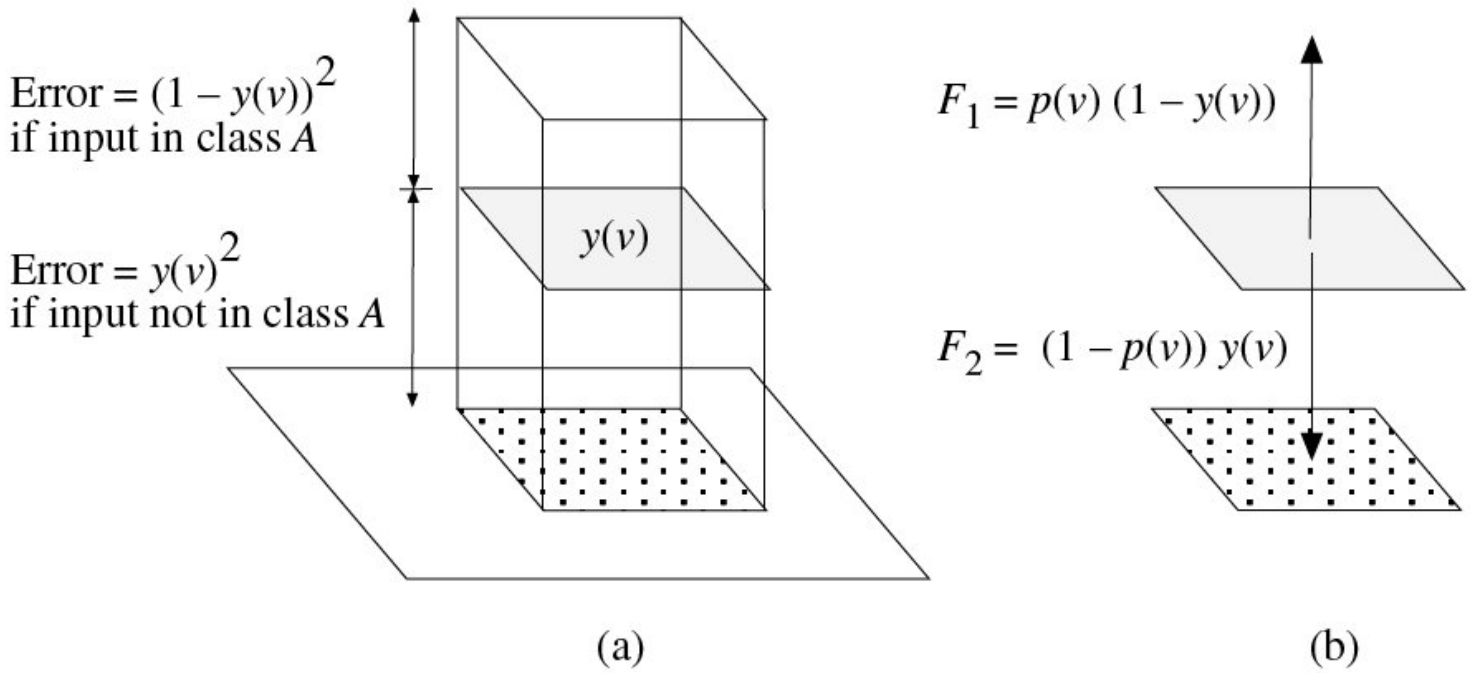
The NETtalk architecture

Sieci neuronowe i statystyka - sieci klasyfikujące

- Sieć NETtalk posiada około 1800 wag.
- Sieć powinna wyekstrahować statystyczne regularności dzieki uczeniu się.
- Zbiór treningowy - setki słów z transkrypcją fonetyczną.
- Po zakończeniu nauki nieznany tekst jest skanowany i jednostka z o maksymalnym wyjściu jest wybierana (na wyjściu nie będzie dokładnie wartości 0 lub 1; wybieramy tą o najwięszej wartoęści).
- Generowana mowa jest jakości porównywalnej do znacznie bardziej skomplikowanych systemów opartych na regułach.
- W początkowych fazach nauki sieć popełniała takie same błędy jak dzieci uczące się mówić.
- Zmiana (= uszkodzenie) niektórych wag powodowała specyficzne problemy z mową.
- Analizując wyjścia ukrytych jednostek autorzy stwierdzili, że sieć nauczyła niektórych ze znanych reguł lingwistycznych.

Sieci neuronowe i statystyka - sieci klasyfikujące - prawdopodobieństwa

- Wartości na wyjściu ($\in [0,1]$) - interpretacja jako prawdopodobieństwa.
- Sieć uczymy wartościami binarnymi - jak uczy się prawdopodobieństw? Fakt ten można udowodnić:



Sieci neuronowe i statystyka - sieci klasyfikujące - prawdopodobieństwa

- Punkty w przestrzeni wejść - klasyfikacja jako należące do zbioru A lub jego dopełnienia - wystarczy rozważyć jedną klasę.
- Ogólnie - liczba wyjść = liczba klas; na wyjściu ma pojawić się 1, jeżeli wejście należy do odpowieniej klasy.
- Błąd całkowity - suma błędów dla każdej klasy, które można **minimalizować indywidualnie**.

Twierdzednie. Sieć neuronowa klasyfikująca doskonale wytrenowana i o wystarczającej elastyczności może nauczyć się prawdopodobieństw zbioru danych empirycznych.

Dowód.

- Podzielmy przestrzeń wejść na różnicznkowe objętości dv o środku w n-wymiarowym punkcie v.
- Na wyjściu przypisanym klasie A sieć oblicza wartość $y(v) \in [0,1]$ (powinno być 1 zatem błąd to $1 - y(v)$).
- $p(v)$ - prawdopodobieństwo, że $v \in V(v)$ należy do A.
- Wartość oczekiwana kwadratu błędu:

$$E_A = \sum_V \{p(v)(1 - y(v))^2 + (1 - p(v))y(v)^2\} dv,$$

  suma przebiega po wszystkich objętościach różniczkowych.
- Zakładając, że wartości $y(v)$ mogą być obliczane indywidualnie - można minimalizować błędy dla każdego v niezależnie.
- Różniczkując po $y(v)$:

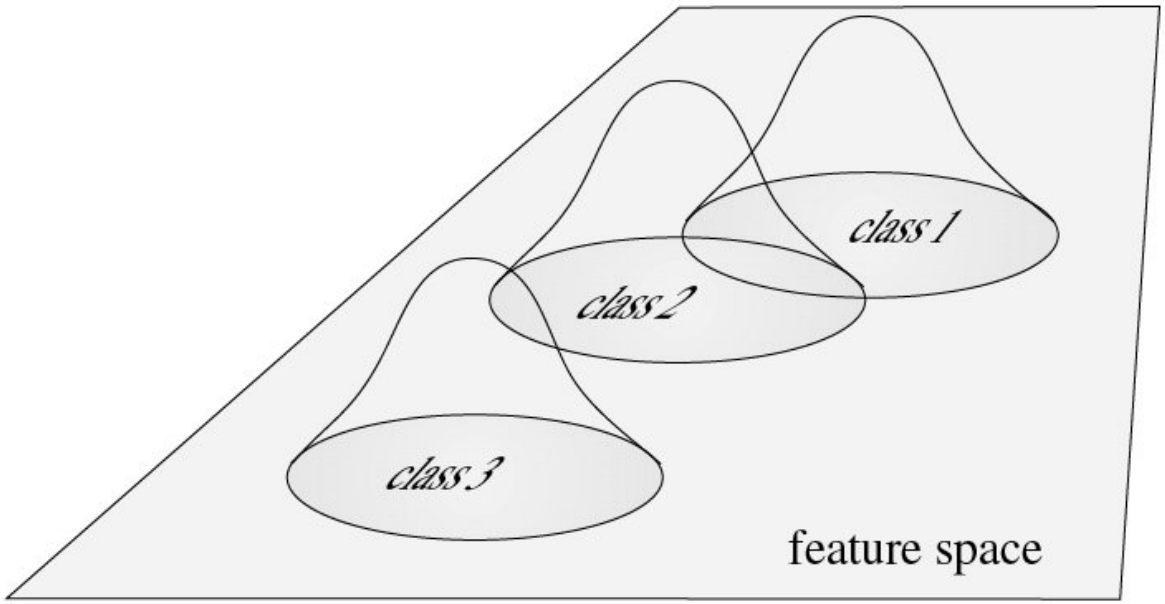
$$-2p(v)(1 - y(v)) + 2(1 - p(v))y(v) = 0 \Rightarrow \mathbf{p(v)} = \mathbf{y(v)}.$$

Sieci neuronowe i statystyka - sieci klasyfikujące - prawdopodobieństwa

- Wartość oczekiwana błędu to $p(v)(1-p(v))^2 + (1-p(v))p(v)^2 = p(v)(1-p(v))$ a E_A staje się wariancją dla klasy A.
- Doskonałe wytrenowanie i elastyczność są istotne aby dobrze przybliżyć rozkład prawdopodobieństwa różniczkowymi objętościami.
- Część **(b)** rysunku pokazuje jeszcze prostszy dowód.
  - otrzymaną funkcję $y(v)$ poddajemy "działaniu siły" proporcjonalnej do pochodnej błędu $1-y(v)$ i prawdopodobieństwa $p(v)$ oraz drugiej "siły" proporcjonalnej do $y(v)$ i $1-p(v)$.
  - siły te są w równowadze gdy $p(v) = y(v)$ (funkcja ma się nie zmieniać $\Rightarrow$ równowaga).

Sieci neuronowe i statystyka - sieci klasyfikujące

- **Wizualizacja** - 3 klasy (niekoniecznie rozłączne); przynależność każdego wektora wejściowego do danej klasy jest dana jako prawdopodobieństwo.



Probability distribution of several classes in feature space

Sieci neuronowe i statystyka - sieci klasyfikujące

- **Przykład** - n-wymiarowe wektory o wartościach przypisanych n możliwym symptomom chorób, które z kolei są reprezentowane przez klasy.
- Rozłączność klas można uzyskać tylko dodając więcej informacji (symptomów).
- Istniejące banki danych - zbiór treningowy i możliwość obliczenia błędów klasyfikacji.
- Sieć może obliczyć wstępną diagnozę a lekarz decyduje czy brać to pod uwagę.

Sieci neuronowe i statystyka - sieci klasyfikujące

- **Rozpoznawanie mowy** - problem odwrotny: mając sekwencję sygnałów akustycznych należy przekształcić je w tekst.
- Jest to zadanie znacznie trudniejsze niż synteza mowy (rozpoznawanie pełni decydującą rolę).
- Niezwykła czułość na **kontekst** - niektórych fonemów może nie być w mowie i nie sprawia to różnicy; ludzie bez problemu oddzielają szum tła a podczas rozpoznawania mowy jest to duży problem.
- Można przypuszczać zatem, że **reguły deterministyczne będą znacznie gorsze** niż system statystyczny.
- Pierwsze eksperymenty w zakresie rozpoznawania mowy - lata 50-te; w latach 60-tych istniały systemy rozpoznające głoski wymawiane przez różnych ludzi.
- Do dziś nie wszystkie oczekiwania zostały w pełni spełnione.
- Istnieją **komercyjne** systemy - zwykle z ograniczonym słownictwem i jakoś zależne od mówcy.

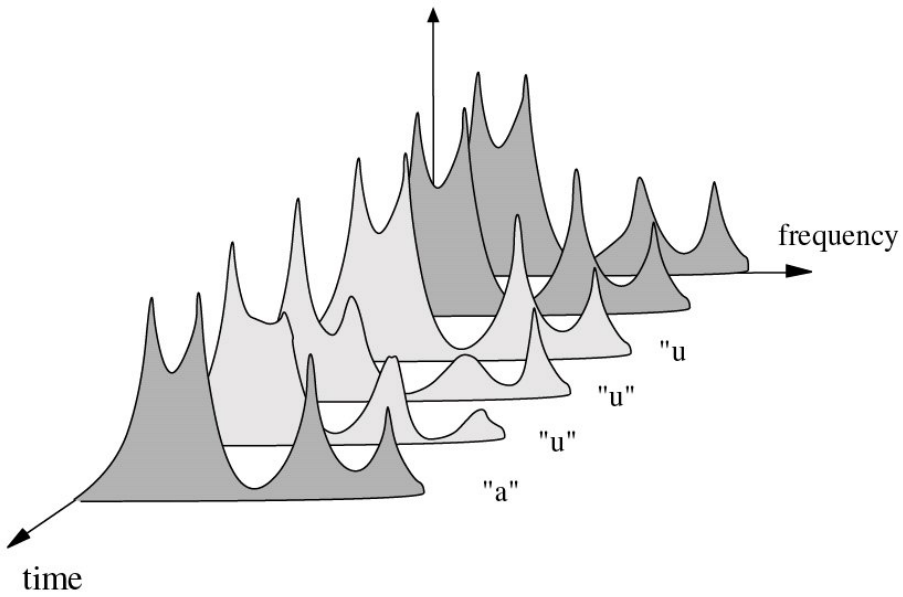
Sieci neuronowe i statystyka - sieci klasyfikujące

- **Docelowo**: rozpoznawanie bez sztucznych pauz oraz zrozumienie każdego bez kontekstu rozmowy czy monologu.
- Wiedza o procesie rozpoznawania mowy jest ciągle ograniczona.
- Dominuje podejście **pragmatyczne** - łączenie najlepszych cech sieci neuronowych z tradycyjnymi algorytmami (lub innymi pojęciami statystycznymi); stąd nazwa **hybrydowe systemy rozpoznawania mowy**.

- **Wyodrębnianie cech**
  - pierwszy problem - znalezienie odpowiedniej reprezentacji mowy
  - aby rozpoznanie było dobre - odpowiednio gęste próbkowanie ale też dobrze mieć umiarkowaną liczbę punktów zachowując wystarczającą ilość informacji
  - podczas powstawania głosu - sygnał wytwarzany przez struny głosowe - rezonans powoduje wzmocnienie pewnych częstotliwości oraz wygaszenie innych
  - należy wykryć odpowiednie częstotliwości

Sieci neuronowe i statystyka - sieci klasyfikujące

- **Formanty** - maksima w widmie.
- Każdy fonem ma charakterystyczną sygnaturę formantów.
- Analiza spektralna mowy - wiele metod; niektóre o podstawach psychofizycznych (oparte na fizjologii słyszenia); inne pochodzą z inżynierii; jedną z najprostszych i najpotężniejszych jest **analiza fourierowska** krótkich odcinków czasowych.



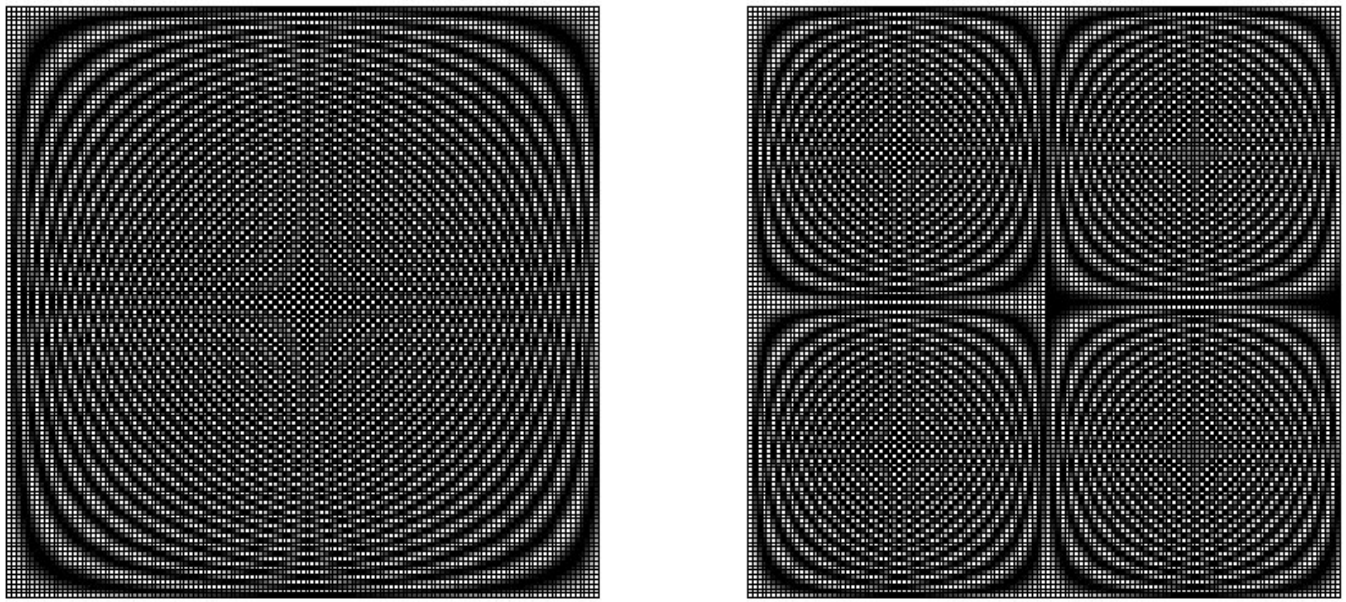
Temporal variation of the spectrum of the speech signal

Sieci neuronowe i statystyka - sieci klasyfikujące

- Analiza Fouriera = struktura okresowa zbioru danych.
  - okienko o długości n (ilość próbek np. 10 milisekund)
  - transformata daje n amplitud
  - okienko przesuwamy do n następnych danych itd.
  - z tej informacji algorytm powinien odtworzyć prawidłową sekwencję fonemów.
- Najciekawsza jest analiza w **czasie rzeczywistym** - ważna redukcja liczby operacji.
  - transformata Fouriera (dyskretna, mnożenie macierz-wektor) ok. $\mathcal{O}(n^3)$ mnożeń
  - **szybka transformata Fouriera**- FFT = fast Fourier transform - polega na odpowiednim uporządkowaniu.

Sieci neuronowe i statystyka - sieci klasyfikujące

- Część rzeczywista elementów macierzy transformacji $Re(\mathbf{F}_n)$
- Po prawej - parzyste kolumny po lewej a nieparzyste po prawej
  - 4 macierze $n/2$ x $n/2$ ($\mathbf{F}_{n/2}$)
  - istnieje relacja pomiędzy $\mathbf{F}_n$ i $\mathbf{F}_{n/2}$
  - n musi być potęgą 2

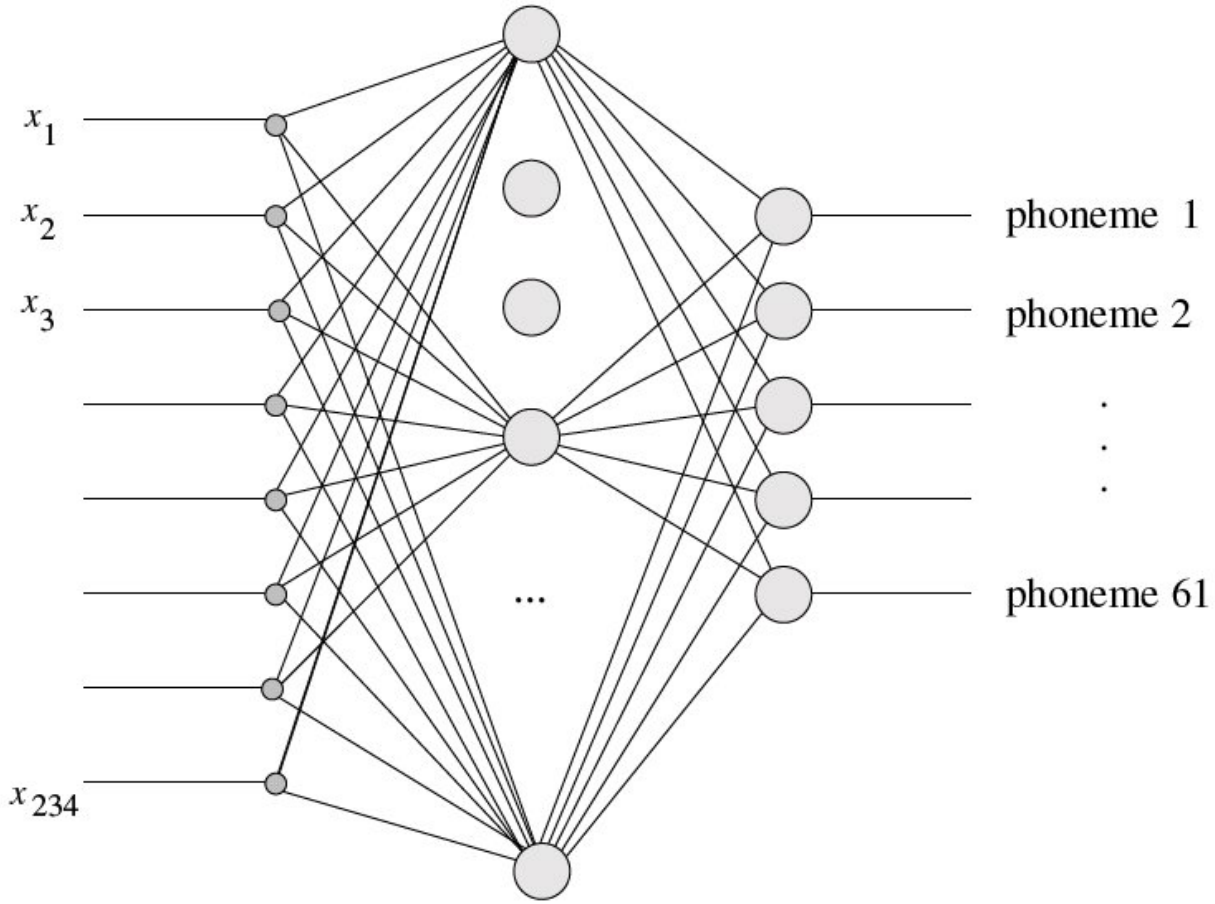


The Fourier matrix and the permuted Fourier matrix

Sieci neuronowe i statystyka - sieci klasyfikujące

- **Trening klasyfikatora** - cel - obliczenie prawdopodobieństwa, że fonem odpowiada danemu spektrum.
  - sygnał dzielimy na $10ms$ odcinki
  - obliczamy spektrum i kwantujemy (18 współczynników)
  - trenujemy sieć w rozpoznawaniu fonemów
  - używamy 6 poprzednich i 6 kolejnych odcinków (kontekst) $\Rightarrow$ wymiar wektora wejściowego $18(6+1+6) = 234$
  - rozważając 61 fonemów otrzymujemy następującą sieć:

Sieci neuronowe i statystyka - sieci klasyfikujące

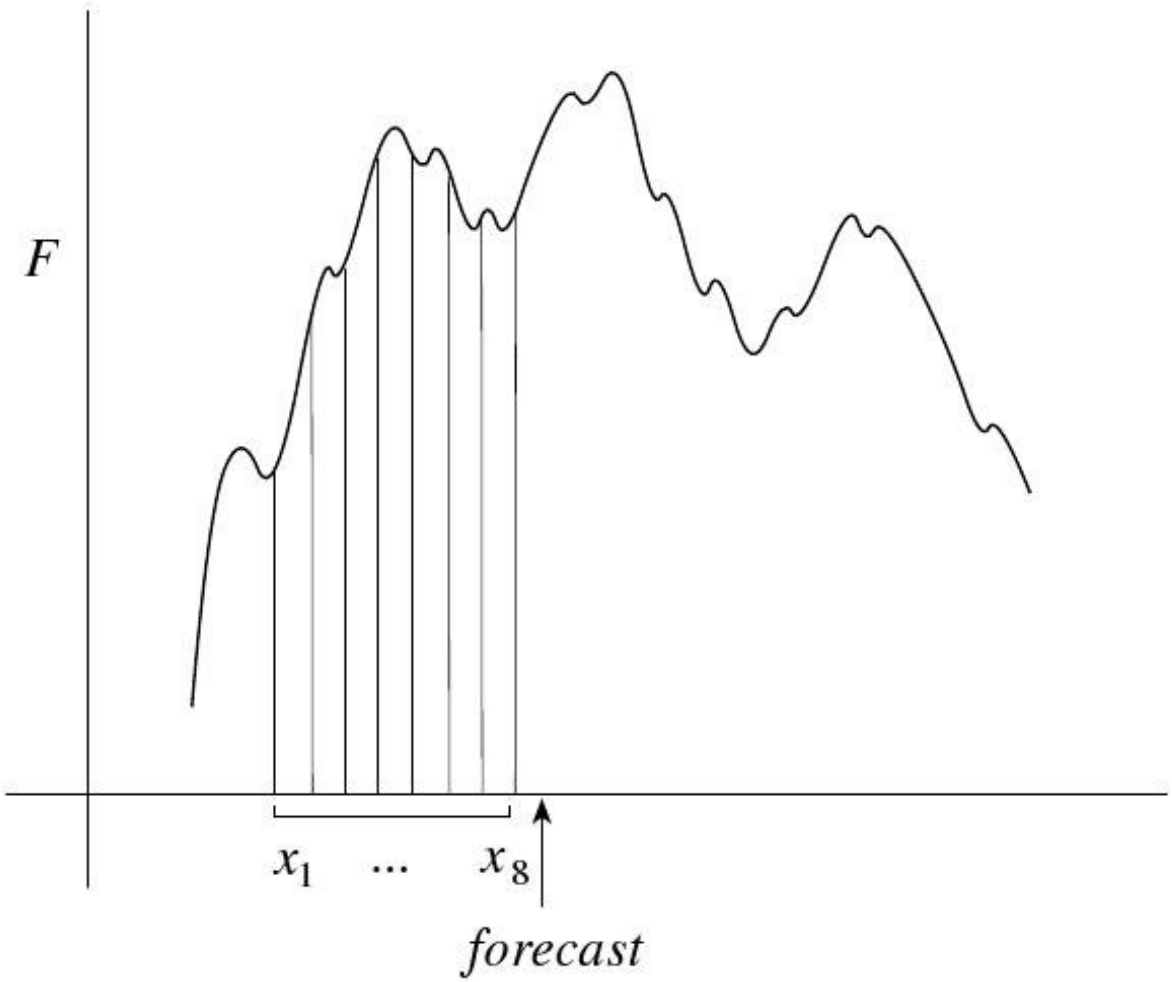


Classification network for 61 phonemes

Sieć uczymy znanymi zestawami danych - istnieją odpowiednie bazy danych.

Sieci neuronowe i statystyka - analiza szeregów czasowych

- Ważną kwestią jest rozwijanie dobrych **technik przewidywania dla szeregów czasowych** w ekonomii i statystyce.
- Zmienna stochastyczna X daje sekwencje danych $x_1, x_2, ..., x_t$, którą można użyć do przewidzenia wartości w czasie $t + 1$.
- Zwykle **modele liniowe** relacji pomiędzy kolejnymi wartościami były preferowane ze wzgledu na zgromadzone doświadczenie i literaturę.
- Sieć neuronową trenujemy dostępnymi danymi i testujemy na nowych wartościach.
- Można użyć $x_1, x_2, ..., x_8$ to przewidzenia x_9 i dalej otrzymujemy $n - 8$ próbek treningowych z serii n danych.
- Na tych danych sieć uczy się przewidywać x_t na podstawie $x_{t-8}, ..., x_{t-1}$.



Time series and a training window

Sieci neuronowe i statystyka - analiza szeregów czasowych

- Technika ta odpowiada modelom **autoregresywnym** znanym w statystyce.
- Stosujemy przybliżenie (jest to model **liniowy**)

$$x_t = \alpha_0 + \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + ... + \alpha_p x_{t-p} + Z_t,$$

$\alpha_1, ..., \alpha_p$ - stałe; Z_t - **zmienna losowa**.

- Można zrobić więcej niż jeden krok w przyszłość

$$x_{t+q} = \alpha_0 + (\beta_0 x_t + ... + \beta_{q-1} x_{t+q-1}) + (\alpha_1 x_{t-1} + ... + \alpha_p x_{t-p}).$$

- Możliwe modele **nieliniowe**, ogólnie stosuje się różne testy statystyczne i techniki obróbki wstępnej przed wyborem najlepszej metody.
- Szeregi czasowe w **ekonomii** odpowiadają sytemom z ogromną liczną stopni swobody - należy ograniczyć znacząco wymiar problemu.
  - Często nie można oprzeć prognozy na samym szeregu czasowym.
  - Trzeba wziąść pod uwagę stopę inflacji, sytuację międzynarodową.
  - Proste modele zawodzą; najlepiej sprawdzają sie modele ekonometryczne pewnej ilości zmiennych z zależnościami funkcyjnymi

Sieci neuronowe i statystyka - historia

- Przybliżanie funkcji z użyciem sieci neuronowych - potrzebna uwaga i wiedza z dziedziny
  - znane przykłady błędów spowodowanych złym użyciem sieci neuronowych
- Efron - metoda bootstrap (modeluje nieznany rozkład prawdopodobieństwa przez ponowne próbkowanie zbioru danych).
- Tukey - studiował metodę jackknife i nadał jej nazwę.
- NETtalk - pierwszy przykład osiągnięcia przez system statystyczny doskonałych wyników metod opartych na regułach znacznie łatwiej oraz mniejszą ilością założeń.
  - uszkadzając sieć modelowano np. dysleksję - "neuropsychologia" sieci.
- Neurogammon - program gra na poziomie mistrzowskim, pierwszy system uczący się, który wygrał turniej komputerów.
  - pokonał systemy oparte na regułach zaprojektowane z dużym nakładem pracy

Sieci neuronowe i statystyka - historia

- Wielu badaczy marzyło o przewewidywaniu rynków finansowych.
  - niektóre banki - projekty aby porównać metody - wyniki niejednoznaczne (były to eksperymenty off-line)
  - podjęcie wysokiego ryzyka czasem może dać zaskakująco dobre rezultaty - realnie taka sytuacja była by zabroniona
  - gdyby każdy miał dobrą metodę - ustaliła by się nowa równowaga
  - systemy próbujące przewidzieć bankructwa

Logika rozmyta

- Wyjście sieci z propagacją wsteczną - wartości nie tylko 0 lub 1.
- Interpretacja takiego wyjścia - metody **logiki rozmytej**.
- Uogólnienie logiki klasycznej - **Lofti Zadeh** (lata 60-te); modelowanie problemów o nieprecyzyjnych danych lub nieprecyzyjnych regułach.
- W logice rozmytej wartość logiczna **zdania** jest kontinuum wartości prawda.
- Przykładowo: zdanie A może mieć wartość prawda 0.4, a A^C - 0.5 $(0.4 + 0.5 \neq 1!)$.
- Istnieje związek między logiką rozmytą a teorią prawdopodobieństwa (nie jest konieczne takie usprawiedliwianie jej).
- Zwyczajowe podejście - uogólnienie logiki zachowujące część struktury algebraicznej.

Logika rozmyta

- Teoria zbiorów rozmytych odpowiada logice rozmytej a semantykę operatorów rozmytych można przedstawić graficznie.
- Interpretacja geometryczna logiki rozmytej pozwala połączyć ją z teorią sieci neuronowych.
- Logika rozmyta może być modelem interpretacji dla własności sieci neuronowych.
- Operatory rozmyte - uogólnione funkcje wyjściowe neuronów.
- Czasem pozwalają na pominięcie uczenia - czasem prościej jest podać prosty zbiór reguł dla danego układu niż uczyć sieć (przykład: zagadnienie parkowania samochodu).
- Logika rozmyta jest używana w wielu produktach komercyjnych: przemysł i elektronika użytkowa w zastosowaniach gdzie potrzebny jest dobry ale nie koniecznie optymalny system kontroli.

Logika rozmyta - zbiory rozmyte

- Niech $X = \{x_1, x_2, ..., x_n\}$. Podzbiór $A = \{x_1\} \subset X$ może być wektorem **przynależności**

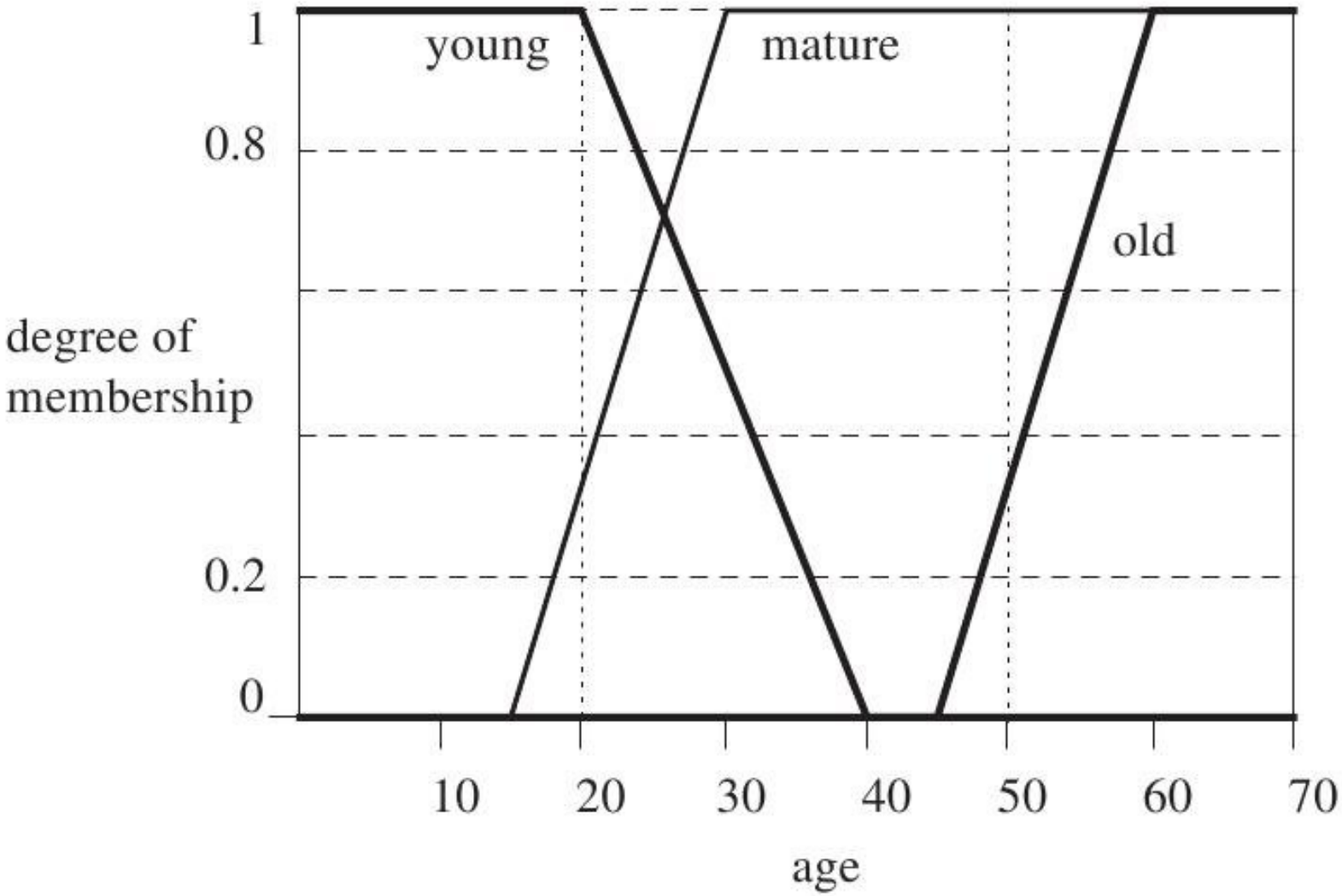
$$Z(A) = (1, 0, ..., 0)$$

- Analogicznie można opisać dowolny zbiór klasyczny.
- Jeżeli przyjąć, że składowe Z nie muszą mieć wartości binarnych, to przkładowy **zbiór rozmyty** C ma opis wektorowy

$$Z(C) = (0.5, 0, ..., 0).$$

- Można powiedzieć, że element x_1 tylko częściowo należy do zbioru C (stopień przynależności to $0.5 \in [\mathbf{0}, \mathbf{1}]$).
- Jest to analogiczne do stwierdzienia typu "osoba x_1 jest dorosła" - nie ma absolutnego progu, stawanie się dorosłym jest procesem ciągłym.
- Przynależność osoby do zbioru osób dorosłych wzrasta od 0 do 1.

Logika rozmyta - zbiory rozmyte



Membership functions for the concepts young, mature and old

Logika rozmyta - zbiory rozmyte

- Inne przykłady stwierdzeń rozmytych stary-młody, szybki-wolny.
- W niektórych zastosowaniach (systemy ekspertowe) potrzebne są formalne metody radzenia sobie z tego typu zdaniami tak, aby komputer oparty na logice Boolowskiej mógł je przetwarzać.
- Definicja. Niech X będzie klasycznym zbiorem. Funkcję rzeczywistą $\mu_A : X \to [0,1]$ nazywamy **funkcją przynależności** A i definiuje **zbiór rozmyty A** oparty o zbiór X. Jest on zbiorem par $(x, \mu_A(x))$, gdzie $x \in X$.
- Funkcja przynależności całkowicie określa zbiór rozmyty.
- Powyższa definicja obejmuje też przypadek gdy X nie jest skończony.
- **Zbiorem podparcia** zbioru rozmytego A jest zbiór wszystkich elementów $x \in X$ dla których $(x, \mu_A(x)) \in A$ oraz $\mu_A(x) > 0$.
- Zbiór rozmyty A o skończonym zbiorze podparcia $\{a_1, a_2, ..., a_m\}$ można opisać symbolicznie

$$A = \mu_1/a_1 + \mu_2/a_2 + ... + \mu_m/a_m, \quad \mu_i = \mu_A(a_i),\ i = 1, 2, ..., m.$$

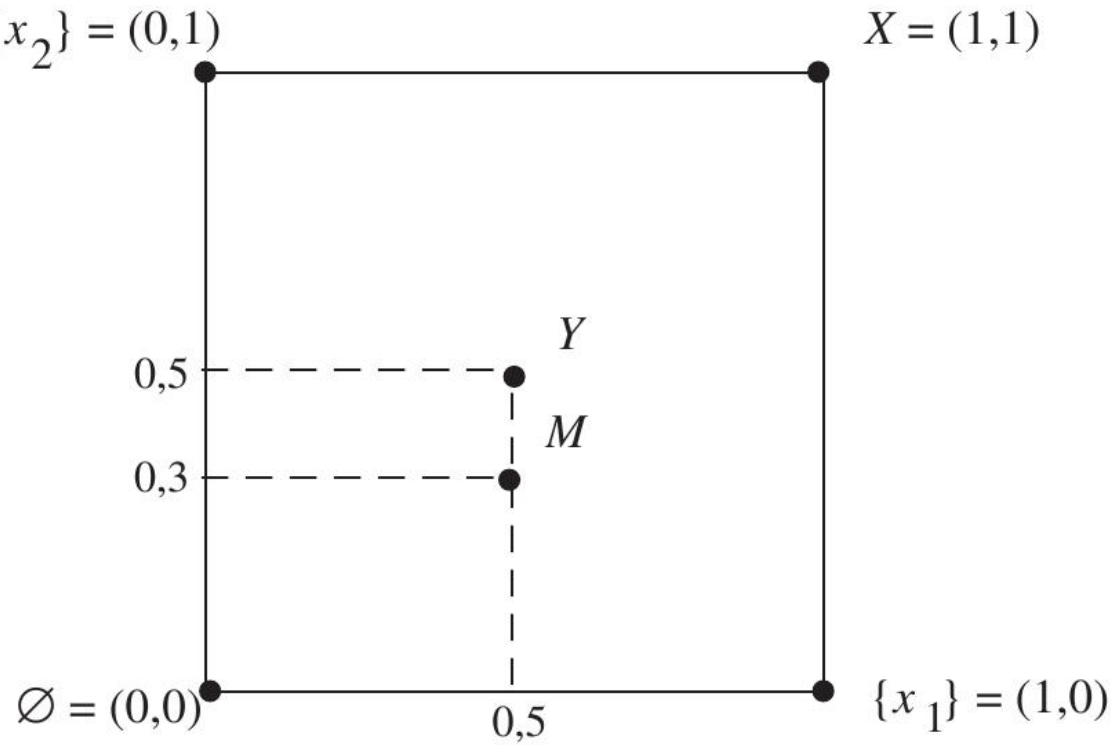
- Uwaga - "/" i "+" mają znaczenie symboliczne.

Logika rozmyta - działania na zbiorach rozmytych

- $X = \{x_1, x_2, x_3\}$. **Sumę** $A = \{x_1, x_2\}$ i $B = \{x_2, x_3\}$ tworzy się biorąc maksimum przynależności każdego z elementów obu zbiorów
  $A = 1/x_1 + 1/x_2 + 0/x_3$
  $B = 0/x_1 + 1/x_2 + 1/x_3$
  $A \cup B = 1/x_1 + 1/x_2 + 1/x_3$
- Dla zbiorów rozmytych podobnie
  $C = 0.5/x_1 + 0.6/x_2 + 0.3/x_3$
  $D = 0.7/x_1 + 0.2/x_2 + 0.8/x_3$
  $C \cup D = 0.7/x_1 + 0.6/x_2 + 0.8/x_3$
- **Przekrój zbiorów** - minimum funkcji przynależności.
- Istnieją alternatywne definicje działań na zbiorach.

Logika rozmyta - interpretacja geometryczna zbiorów rozmytych

- **Reprezentacja graficzna zbiorów rozmytych** (w tym klasycznych) - każda współrzędna odpowiada funkcji przynależności jednego z elementów.



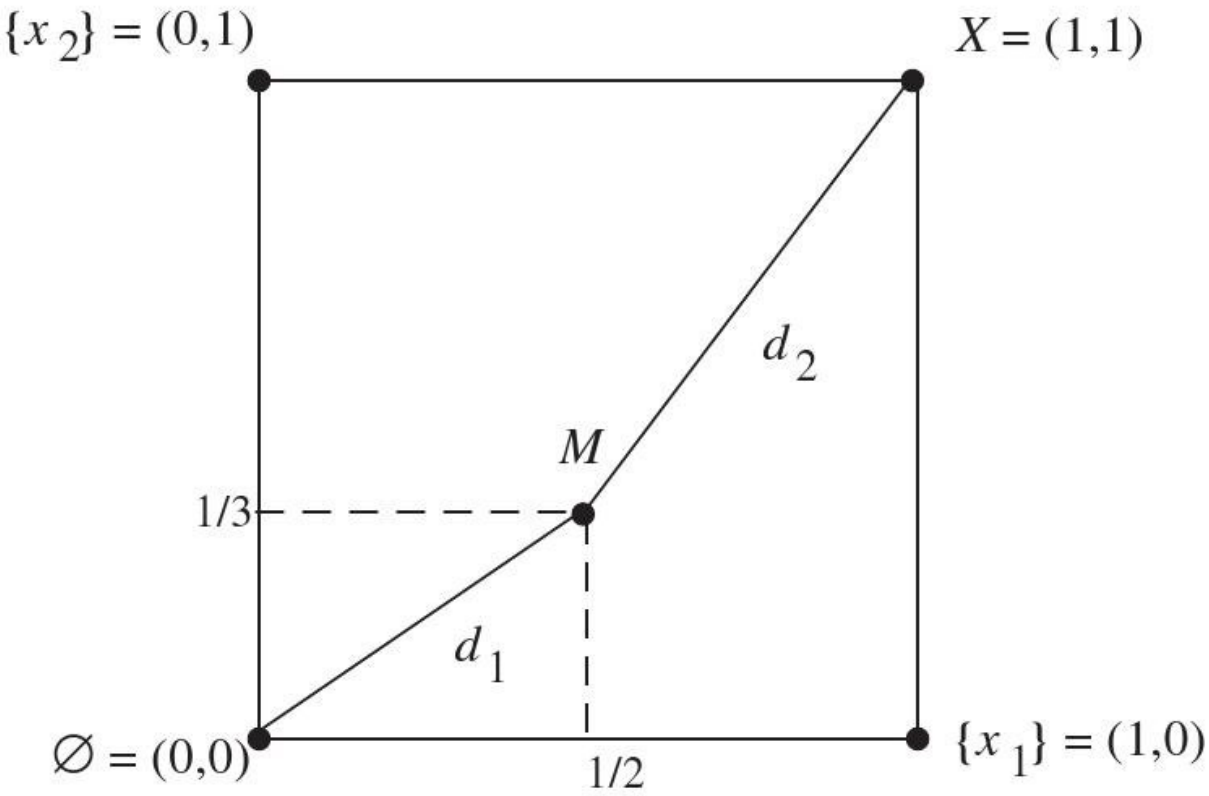
Logika rozmyta - interpretacja geometryczna zbiorów rozmytych

- $M = 0.5/x_1 + 0.3/x_2$
- Naturalne jest uważanie zbioru $Y = 0.5/x_1 + 0.5/x_2$ za najbardziej rozmyty.
- Rozmycie można mierzyć **entropią** zbioru, która geometrycznie zależy od odwrotności odległości punktu reprezentującego zbiór od środka kwadratu jednostkowego.
- Wierzchołki mają entropię zero.
- Ta entropia nie ma związku z fizyczną - używa się też nazwy **indeks rozmytości**.
- Formalnie:

$$E(M) = \frac{d_1}{d_2} \in [0, 1],$$

d_1 - odległość od najbliższego wierzchołka, d_2 - odległość od najdalszego wierzchołka.

Logika rozmyta - interpretacja geometryczna zbiorów rozmytych



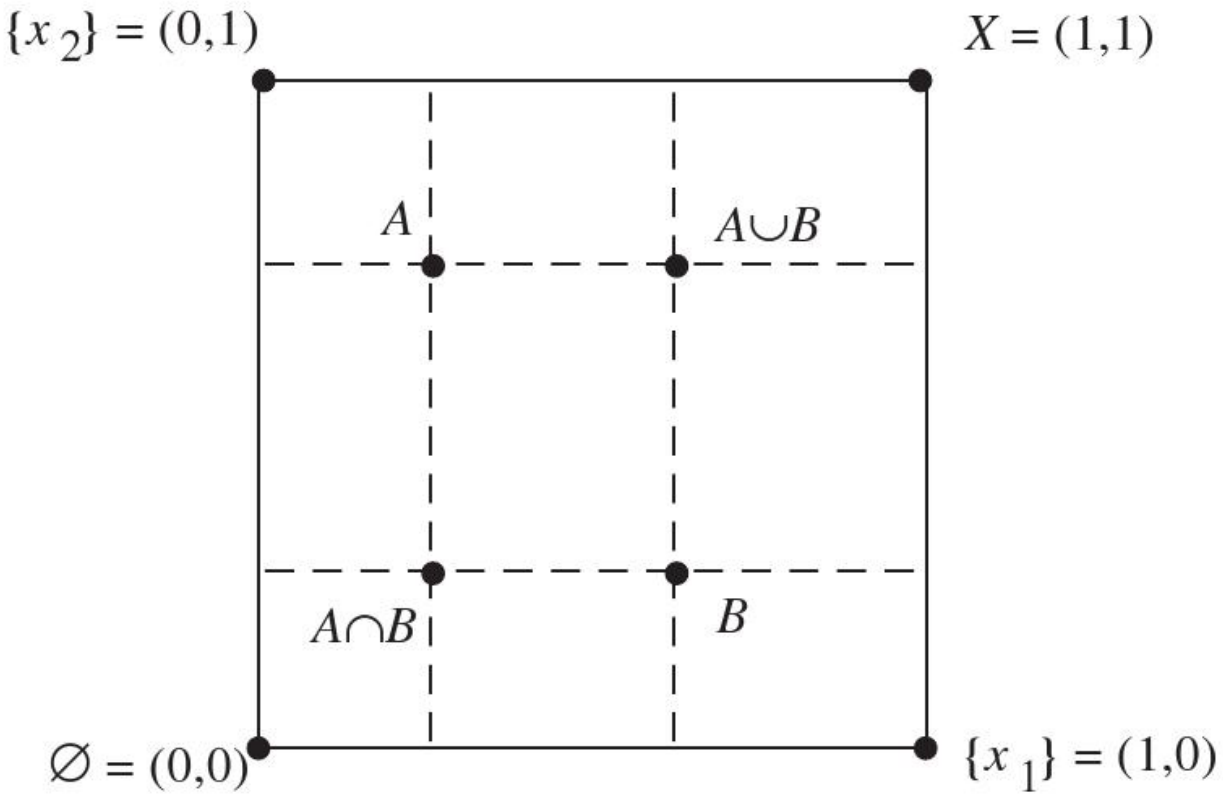
Distance of the set M to the universal and to the void set

Logika rozmyta - interpretacja geometryczna zbiorów rozmytych

- Można graficznie zinterpretować sumę i przekrój.

$$\mu_{A\cup B}(x) = \max(\mu_A(x), \mu_B(x)) \quad \forall_{x\in X}$$

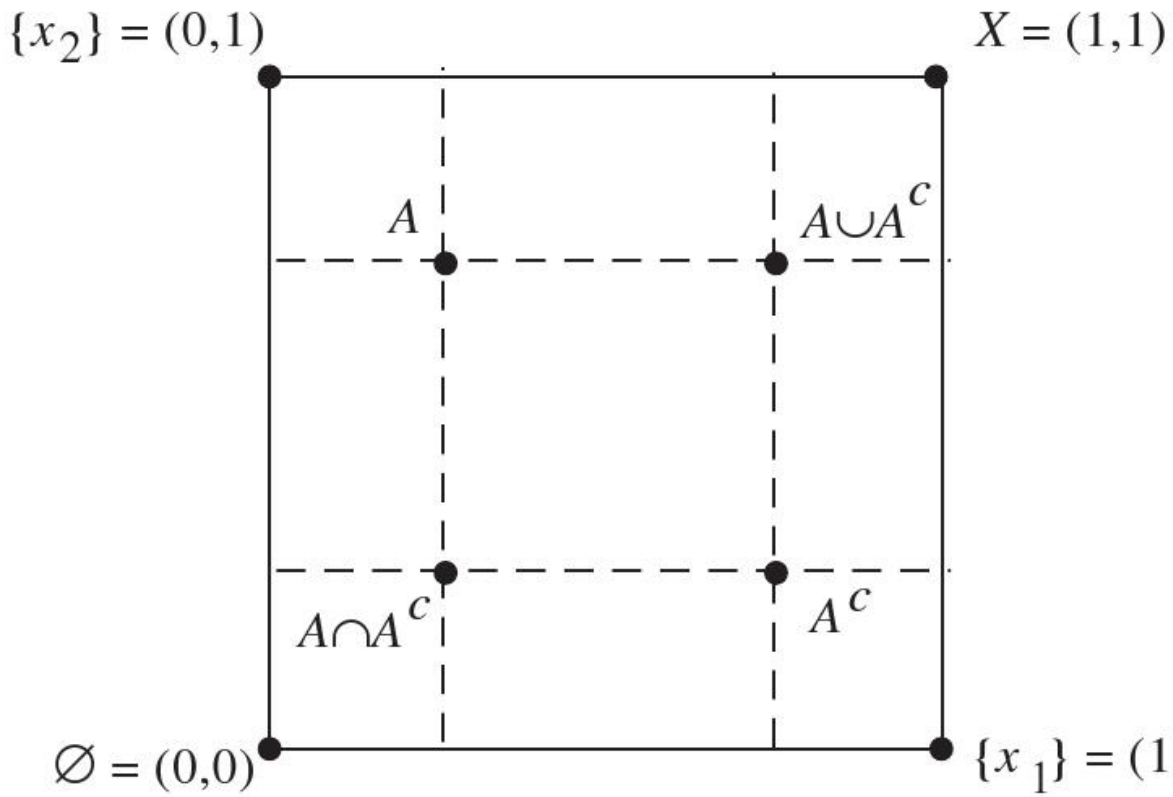
$$\mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x)) \quad \forall_{x\in X}$$



Logika rozmyta - interpretacja geometryczna zbiorów rozmytych

- A^C - uzupełnienie zbioru A do X:

$$\mu_{A^C}(x) = 1 - \mu_A(x) \quad \forall_{x \in X}$$



Logika rozmyta - interpretacja geometryczna zbiorów rozmytych

- W odróżnieniu od teorii klasycznej

$$A \cup A^C \neq X$$

$$A \cap A^C \neq \varnothing$$

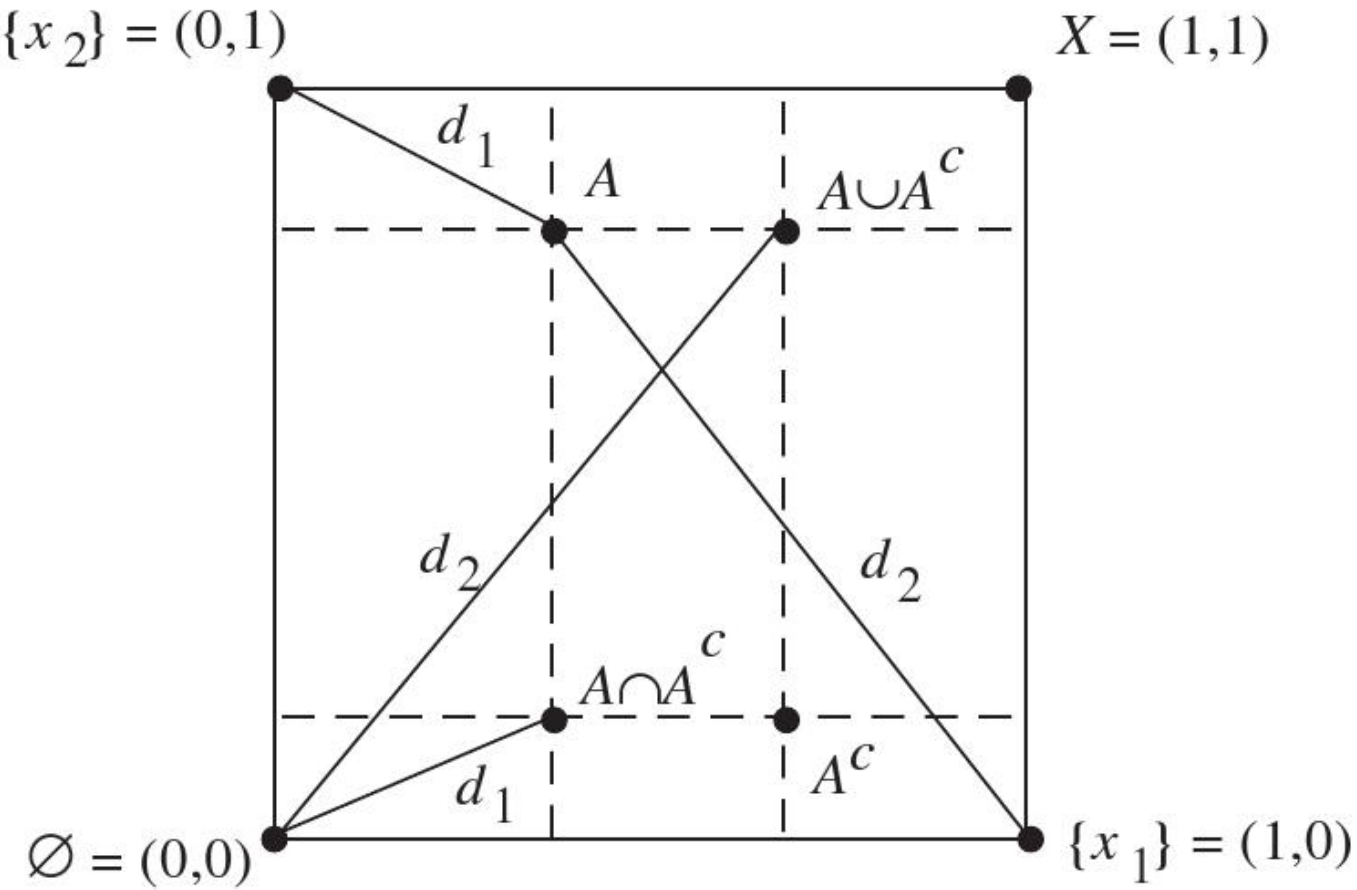
Logika rozmyta - interpretacja geometryczna zbiorów rozmytych

- **Definicja**. Niech A będzie podzbiorem zbioru X. **Kardynalnością** $|A|$ zbioru A jest suma wartości funkcji członkostwa wszystkich elementów zbioru X względem zbioru A

$$|A| = \sum_{x \in X} \mu_A(x).$$

- Definicja ta opisuje odległość w metryce Manhattan.
- Rysunek pokazuje jak zdefiniować entropię z użyciem kardynalności zbiorów $A \cup A^C$ i $A \cap A^C$.

Logika rozmyta - interpretacja geometryczna zbiorów rozmytych



Logika rozmyta - interpretacja geometryczna zbiorów rozmytych

- **Definicja.** Liczbę rzeczywistą

$$E(A) = \frac{|A \cap A^C|}{|A \cup A^C|},$$

nazywamy **entropią** zbioru rozmytego A.

- Entropia zbioru klasycznego jest zawsze zerowa, gdyż $A \cap A^C = \varnothing$.
- $E(A) \in [0, 1]$.
- Czasem entropię definiuje się geometrycznie i powyższą definicje wyprowadza jako twierdzenie o rozmytej entropii.

Logika rozmyta - operatory logiczne

- Istnieje izomorfizm pomiędzy teorią zbiorów a logiką klasyczną.
- Operacje na zbiorach $\cup$, $\cap$ i C odpowiadają operatorom logicznym $\vee$, $\wedge$ i $\neg$.
- Mając dwa zbiory klasyczne A i B o funkcjach członkostwa $\mu_A, \mu_B : X \to \{0, 1\}$, odpowiednia funkcja dla **sumy** to

$$\mu_{A \cup B}(x) = \mu_A(x) \vee \mu_B(x) \quad \forall_{x \in X},$$

gdzie 0 interpretujemy jako wartość logiczną *fałsz* a 1 jako *prawda*.

- Podobnie **przecięcie** i **dopełnienie**

$$\mu_{A \cap B}(x) = \mu_A(x) \wedge \mu_B(x) \quad \forall_{x \in X},$$

$$\mu_{A^C}(x) = \neg \mu_A(x) \quad \forall_{x \in X}.$$

Logika rozmyta - operatory logiczne

- Izomorfizm powoduje przenoszenie się relacji takich jak **prawa de Morgana**

$$(A \cup B)^C = A^C \cap B^C \quad \neg(A \vee B) = \neg A \wedge \neg B$$

$$(A \cap B)^C = A^C \cup B^C \quad \neg(A \wedge B) = \neg A \vee \neg B.$$

Logika rozmyta - operatory logiczne

- Rozmyty operator **lub** ($\tilde{\vee}$) identyfikujemy z funkcją maksimum, operator **i** ($\tilde{\wedge}$) z funkcją minimum a **negację** ($\tilde{\neg}$) z funkcją $x \mapsto 1 - x$:

$$\mu_{A \cup B}(x) = \mu_A(x) \tilde{\vee} \mu_B(x) \quad \forall_{x \in X},$$

$$\mu_{A \cap B}(x) = \mu_A(x) \tilde{\wedge} \mu_B(x) \quad \forall_{x \in X},$$

$$\mu_{A^c}(x) = \tilde{\neg} \mu_A(x) \quad \forall_{x \in X}.$$

- Operatory rozmyte mają pewne własności znane z logiki klasycznej (np. funkcje *min* i *max* są przemienne i łączne).
- Inne reguły są złamane; jeżeli zdanie A ma wartość prawda równą 0.4, to

$$A \tilde{\wedge} \tilde{\neg} A = min(0.4, 1 - 0.4) \neq 0,$$

$$A \tilde{\vee} \tilde{\neg} A = max(0.4, 1 - 0.4) \neq 1.$$

Logika rozmyta - operatory logiczne

- Podane definicje operatorów rozmytych nie są jedynymi możliwościami - można zdefiniować **rodziny operatorów**.
- **Definicja aksjomatyczna** - wymusza pożądane własności.
- Operator rozmyte **OR**:
  1. Warunki brzegowe:
     $0\tilde{\vee}0 = 0$
     $1\tilde{\vee}0 = 1$
     $0\tilde{\vee}1 = 1$
     $1\tilde{\vee}1 = 1$
  2. Przemienność: $a\tilde{\vee}b = b\tilde{\vee}a$
  3. Monotoniczność: $a \leq a'$ i $b \leq b' \Rightarrow a\tilde{\vee}b \leq a'\tilde{\vee}b'$
  4. Łączność: $a\tilde{\vee}(b\tilde{\vee}c) = (a\tilde{\vee}b)\tilde{\vee}c$
- Maksimum spełnia te warunki ale też tzw. ograniczona suma

$$B(a, b) = min(1, a + b)$$

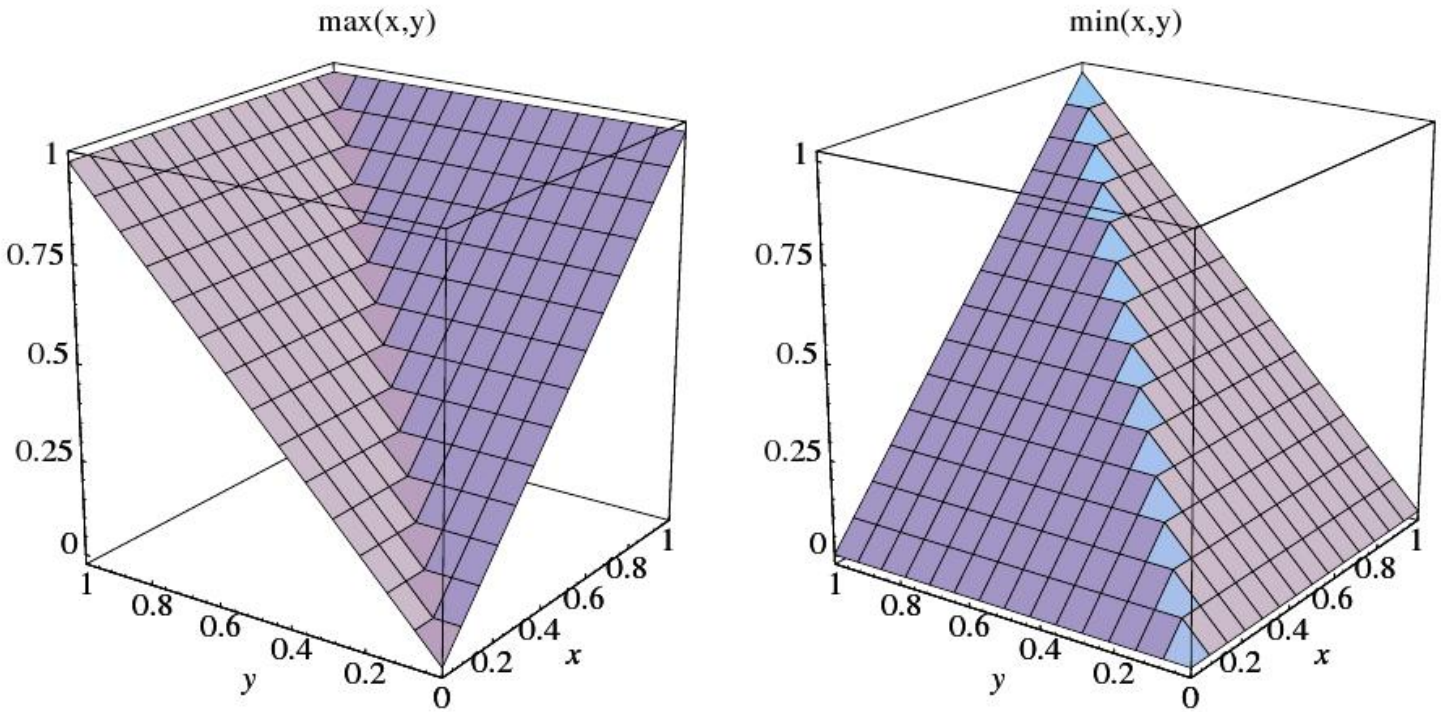
- Funkcja ta nie jest idempotentna tzn. $B(a, a) \neq a$, można wykluczyć takie operatory dodatkowym aksjomatem:
  - Idempotentność: $a\tilde{\vee}a = a$

Logika rozmyta - operatory logiczne

- **Wniosek.** Logika rozmyta jest rodziną teorii.
- Rozmyte **AND**:
  - Przemienność, monotoniczność, łączność
  - Warunki brzegowe:
    $0\tilde{\wedge}0 = 0$
    $1\tilde{\wedge}0 = 0$
    $0\tilde{\wedge}1 = 0$
    $1\tilde{\wedge}1 = 1$
  - Można dołączyć idempotentność.
- Rozmyta **negacja** (NOT):
  - Warunki brzegowe:
    $\tilde{\neg}1 = 0$
    $\tilde{\neg}0 = 1$
  - Monotoniczność: $a \leq b \Rightarrow \tilde{\neg}b \leq \tilde{\neg}a$
  - Inwolucja: $\tilde{\neg}\tilde{\neg}a = a$

Logika rozmyta - operatory logiczne

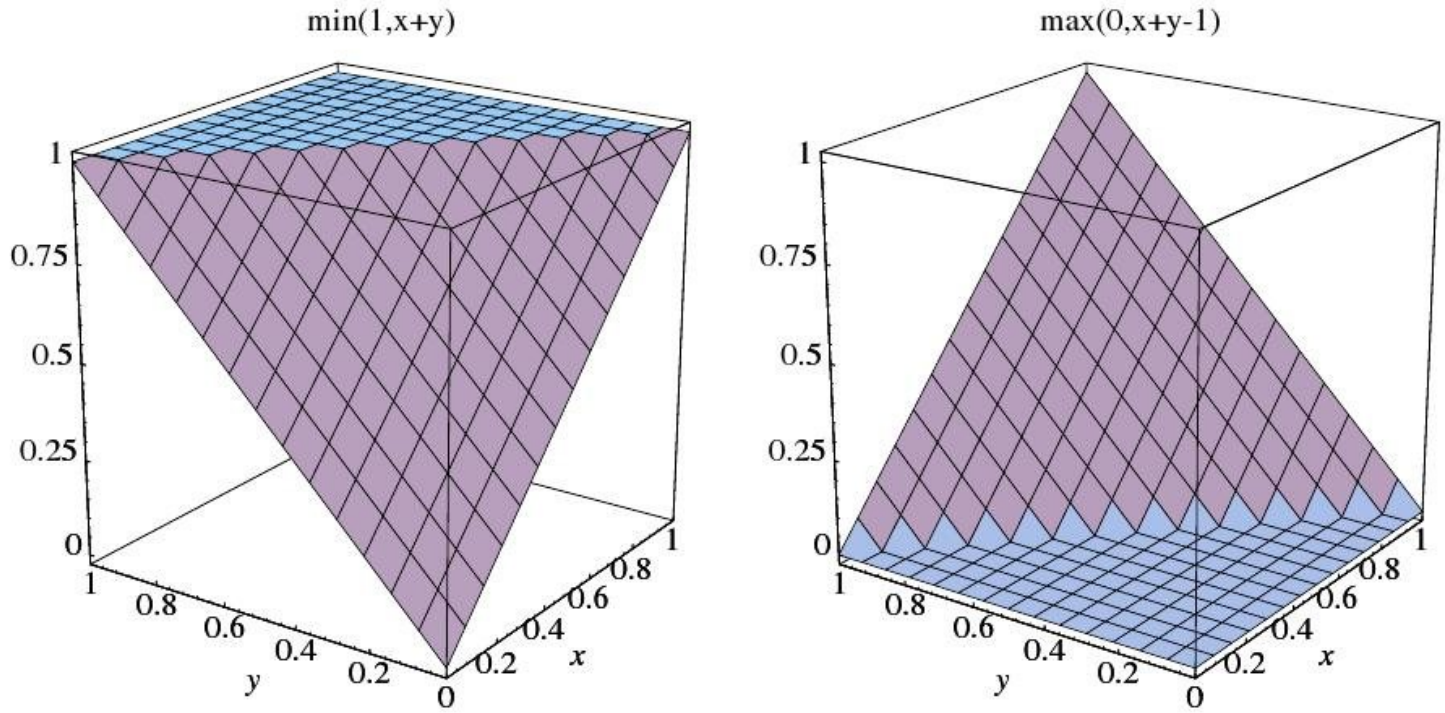
- max i min - możliwe OR i AND:



- można ich użyć jako **funkcji aktywacji** neuronów - możliwa interpretacja logiczna wyjścia sieci

Logika rozmyta - operatory logiczne

Ograniczona suma i ograniczona różnica - OR i AND bez warunku idempotencji:



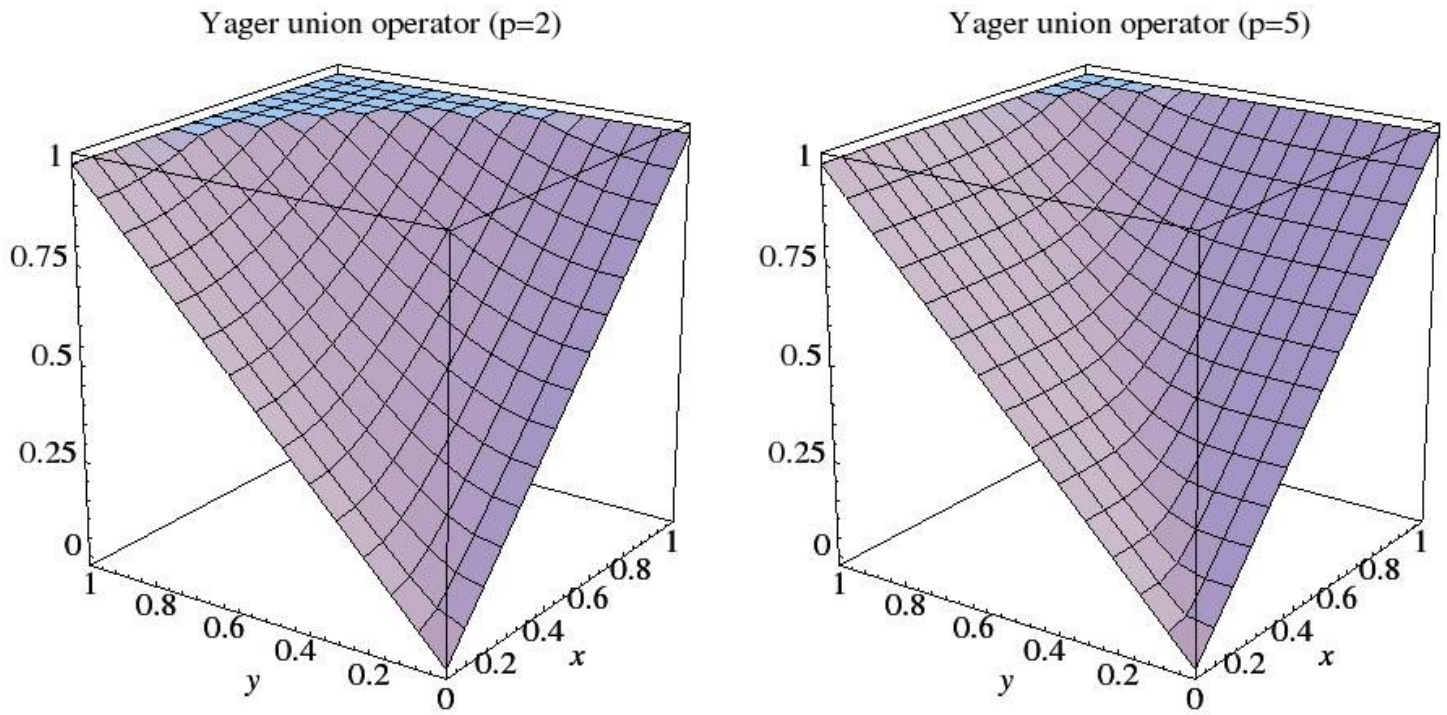
The fuzzy operators *bounded sum* and *bounded difference*

Logika rozmyta - operatory logiczne

Można definiować rodziny parametryczne operatorów rozmytych np. operatory OR Yagera

$$Y_p(a,b) = min(1,(a^p + b^p)^{1/p}), \quad p \geq 1, \quad a,b \in [0,1]$$

$p = 2$ - przybliżenie sumy ograniczonej, $p \gg 1$ - przybliżenie max



Two variants of the Yager union operator

Logika rozmyta - operatory logiczne

- Operatory Yagera - nie są idempotentne - linia łącząca punkty (0,0) i (1,1) musi należeć do wykresu.
- Można pokazać, że min i max są jedynymi funkcjami spełniającymi wszystkie 5 aksjomatów dla OR i AND.
- Własności wykresów wynikają z aksjomatów:
  - warunki brzegowe - wyznaczają 4 punkty wykresu
  - przemienność - symetria względem płaszczyzny diagonalnej i normalnej do p. xy
  - monotoniczność - płaszczyzna nie zawija się
  - łączność - trudna do wizualizacji - nie ma rejonów gdzie funkcja gwałtownie rośnie.
- Odwrotnie - jeżeli wykres pewnej funkcji spełnie te warunki - funkcja spełnia aksjomaty i nadaje się na definicje operatora rozmytego.
- Symetria - prowadzi do użytecznych własności algebraicznych operatorów.

Logika rozmyta - wnioskowanie rozmyte

- Wnioskowanie na podstawie operatorów rozmytych - systemy ekspertowe.
- Zasady, które można przedstawić w sposób rozmyty - przedstawiamy regułami przetwarzalnymi przez komputer.
- **Reguły wnioskowania rozmytego** - struktura taka jak reguł klasycznych:
  R_1: Jeżeli $(A \tilde{\wedge} B)$ to C
  R_2: Jeżeli $(A \tilde{\vee} B)$ to D
  - zakładamy: $\tilde{\wedge}$, $\tilde{\vee}$ - min i max
- Dla wartości prawda(A)=0.4, prawda(B)=0.7:
  $A \tilde{\wedge} B = min(0.4, 0.7) = 0.4$
  $A \tilde{\vee} B = max(0.4, 0.7) = 0.7$
- **Interpretacja**: R_1 i R_2 mogą być zastosowane odpowiednio w 40% i 70%.
  - **rezultat** - odpowiednia kombinacja C i D.

Logika rozmyta - wnioskowanie rozmyte

- Przykład - kontroler temperatury reguluje grzejnik elektryczny, można użyć 3 reguł:
  R_1: Jeżeli (temperatura = zimno) to grzej
  R_2: Jeżeli (temperatura = normalna) to utrzymuj
  R_3: Jeżeli (temperatura = ciepło) to zredukuj moc
- Jeżeli 12° ma wartość funkcji członkostwa 0.5 w zbiorze temperatur `zimno` i 0.3 w zbiorze `normalna`.
- **Kategoria rozmyta** - lista wartości członkostwa elementu w stosunku do uprzednio wybranych zbiorów
  12° = T = zimno/0.5 + normalna/0.3 + ciepło/0.0
- Ogólnie: $x = A/\mu_A + B/\mu_B + C\mu_C$
  Uwaga: zapis ten jest inny niż zapis zbioru rozmytego (NIE: $x = \mu_A/A + \mu_B/B + \mu_C/C$).

Logika rozmyta - wnioskowanie rozmyte

- Używając kategorii T, można równolegle obliczyć reguły R_1, R_2, R_3 - każda prawdziwa do pewnego stopnia.
- Mamy kombinację 3 możliwych konsekwencji - ważonych stopniem ich prawdziwości - **rozmyte wnioskowanie**.
- Wynikiem romytego wnioskowania jest zatem kategoria rozmyta:
  - wnioskujemy: `działanie` = grzej/0.5 + utrzymuj/0.3 + redukuj/0.0
- **Systemy wnioskowania rozmytego** - obliczają takie wielkości:
  - **nieprecyzyjne dane** (kategorie rozmyte) $\Rightarrow$ **wniosek** (nowa kategoria rozmyta)
- **Systemy ekspertowe** - wniosek dalej przetwarzany lub zamieniany na konwencjonalną wartość logiczną (konieczne w przypadku elektronicznego kontrolera rozmytego).
- W wielu przypadkach - złożone problemy kontroli - **niewielka ilość reguł**.
  - dokładne reguły są bardziej skomplikowane i potrzeba ich więcej.

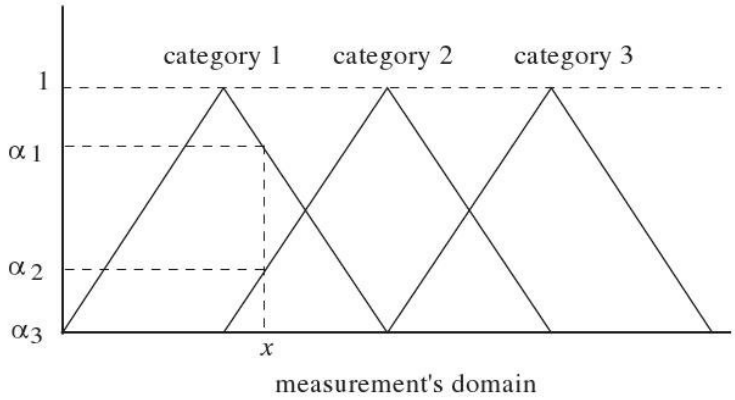
Logika rozmyta - wnioskowanie rozmyte

Działanie kontrolera rozmytego:

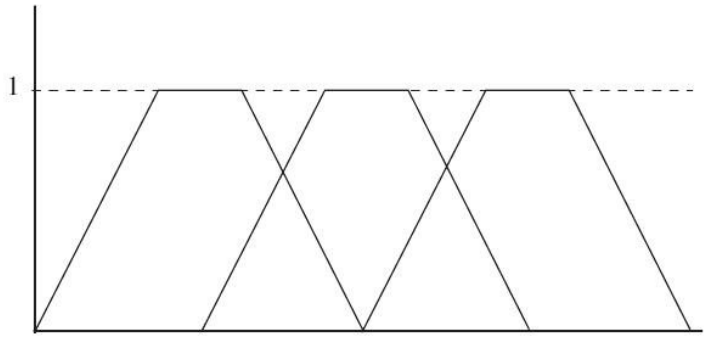
1. Pomiar przekształcany jest w kategorię rozmytą używając funkcji członkostwa dla wszystkich zdefiniowanych kategorii.
2. Obliczane są wszystkie reguły wnioskowania systemy kontrolującego - otrzymujemy wniosek jako kategorię rozmytą.
3. Wniosek rozmyty przekształcamy w wartość klasyczną.

Logika rozmyta - liczby rozmyte i odwrotność

- **Pytanie 1**: Jak przekształcić pomiar w kategorię rozmytą?
- Często - trójkątne lub trapezoidalne funkcje członkostwa.



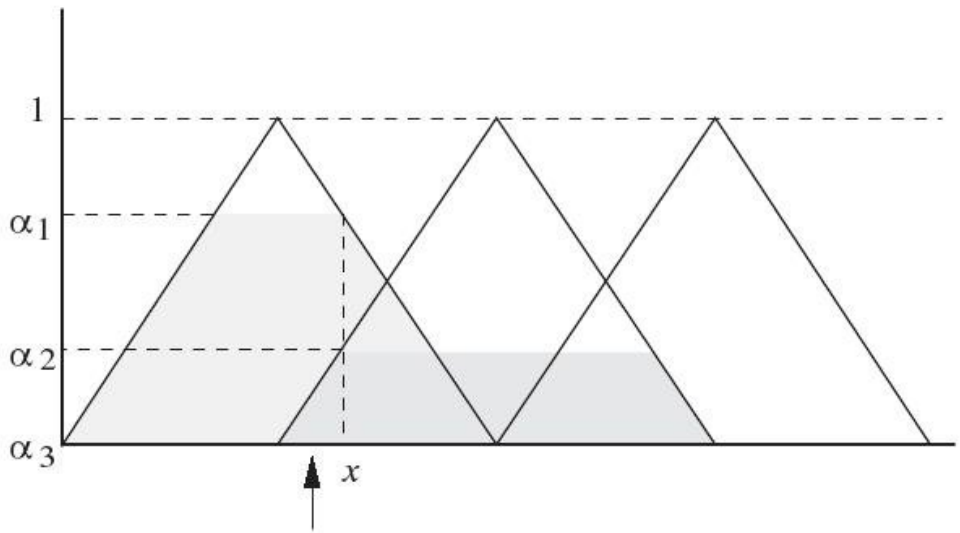
Categories with triangular membership functions



Categories with trapezium-shaped membership functions

Logika rozmyta - liczby rozmyte i odwrotność

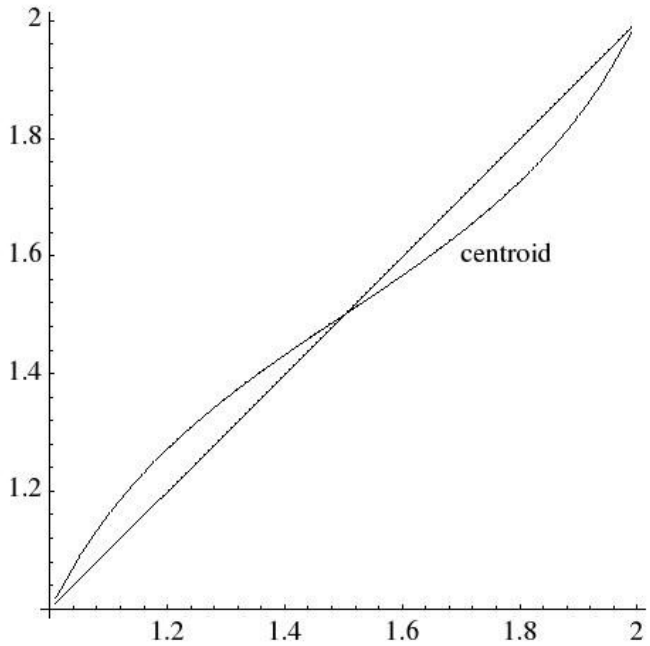
- α_1, α_2, α_3 - wartości funkcji członkostwa definiujące kategorię rozmytą.
- **Pytanie 2**: Jak przekształcić wartości α_1, α_2, α_3 w pomiar x?
  - metoda oparta na **centroidzie** - wartości α wyznaczają powierzchnie - składnik horyzontalny centroidu całkowitej powierzchni.
  - jeżeli przynajmniej 2 z 3 wartości są niezerowe - metoda ta daje dobrze przybliżenie.



The centroid method

Logika rozmyta - liczby rozmyte i odwrotność

- Różnica pomiędzy x i jego przybliżeniem - podstawa trójkątnych obszarów = 2, wysokość = 1; rozmieszczenie jak na poprzednim rysunku.
  - $x \in [1, 2]$.
  - różnica nie przekracza 10%.
  - błąd zależy od rozmieszczenia obszarów; użycie wag związanych z rozmieszczeniem może wpłynąć na wyniki.

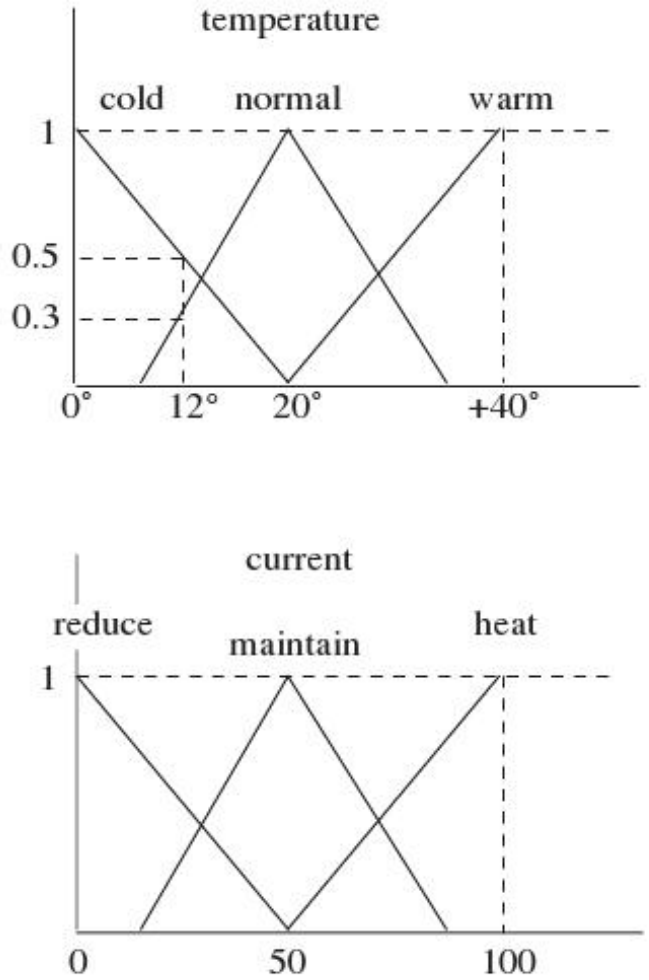


Reconstruction error of the centroid method

Logika rozmyta - kontrola z użyciem logiki rozmytej

- **Kontroler rozmyty** - system regulujący o działaniu danym regułami rozmytymi
- W omówionym kontrolerze temperatury:
  - trzeba określić zakresy zmiennych użytych w problemie
  - $T \in [0, 40]$
  - kontoler zmienia używaną moc elektryczną w zakresie $[0, 100]$ jednostek (50 - wartość normalna, stand-by).
- Na rysunku - funkcje członkostwa dla kategorii `zimno`, `normalna`, `ciepło` temperatury oraz kategorie kontroli `redukuj`, `utrzymuj`, `grzej`.

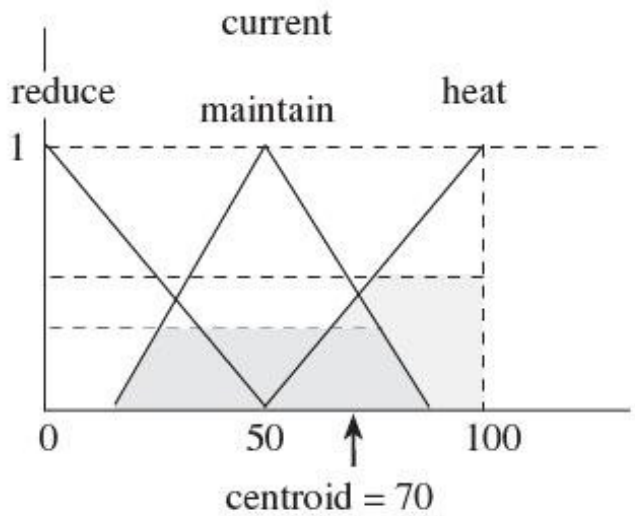
Logika rozmyta - kontrola z użyciem logiki rozmytej



. Membership functions for temperature and electric current categories

Logika rozmyta - kontrola z użyciem logiki rozmytej

- Liczba rozmyta $T = 12^\circ$ przekształcana jest we wniosek (kategorię rozmytą) `działanie`=grzej/0.5 + utrzymuj/0.3 + redukuj/0.0.
- Metodą centroidu - `działanie`=70.

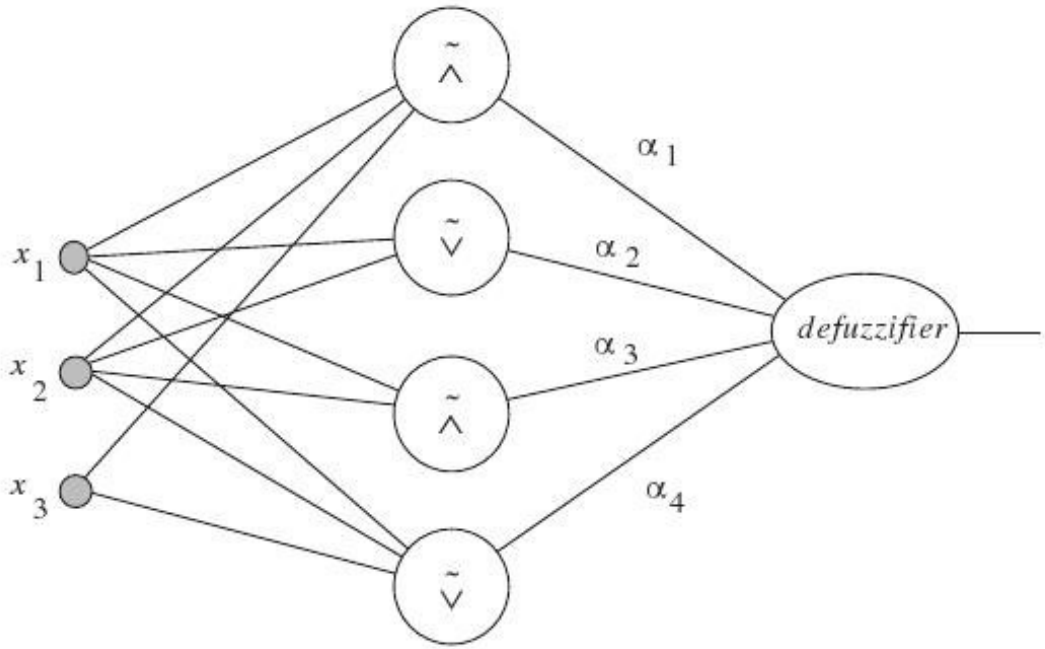


Centroid computation

Można sformułować bardziej skomplikowane reguły o więcej niż 2 zmiennych.

Logika rozmyta - sieci rozmyte

- Systemy rozmyte można reprezentować **sieciami rozmytymi** - są to sieci, których jednostki obliczają operatory rozmyte.
  - Operatory rozmyte obliczane przez warstwę ukrytą odpowiadają zbiorowi reguł wnioskowania.
  - Ostatnia jednostka przekształca uzyskaną kategorię rozmytą w klasyczną zmienna kontrolną - każdy operator ma przypisaną wagę α (możliwość użycia ważonego centroidu).



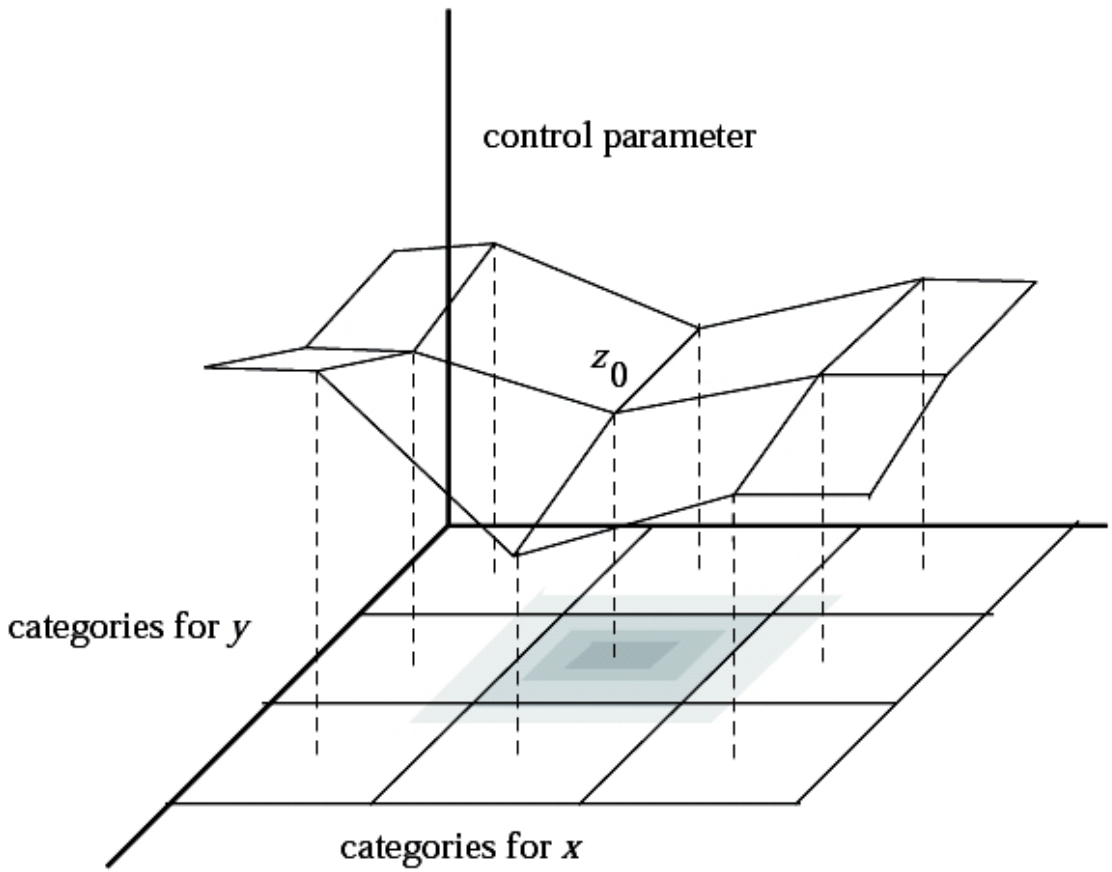
Example of a fuzzy network

Logika rozmyta - sieci rozmyte

- Można zbudować skomplikowaną sieć wielowarstwową - systemy rozmyte zwykle nie prowadzą do bardzo skomplikowanych sieci.
  - nie jest to też polecane - każde wnioskowanie rozmyte **zmniejsza dokładność** wyniku.
- Operatorów rozmytych nie można obliczyć dokładnie jednostkami sigmoidalnymi - dobre przybliżenia: ograniczona suma i różnica.
- Zamiana na wartość klasyczną może być przybliżona standardowymi jednostkami.
  - jeżeli funkcje członkostwa są trójkątami - powierzchnia rośnie kwadratowo z wysokością - funkcje kwadratowe w danych przedziałach mogą być przybliżone sigmoidami.
  - parametry przybliżenia można są określane przez **algorytm uczący**.

Logika rozmyta - aproksymacja funkcji

- Kontroler rozmyty - system do szybkiego obliczania aproksymacji gruboziarnisto zdefiniowanej powierzchni kontrolnej.
  - x i y - zamieniane są w kategorie rozmyte; na rysunku są **3** co daje **9** punktów powierzchni.

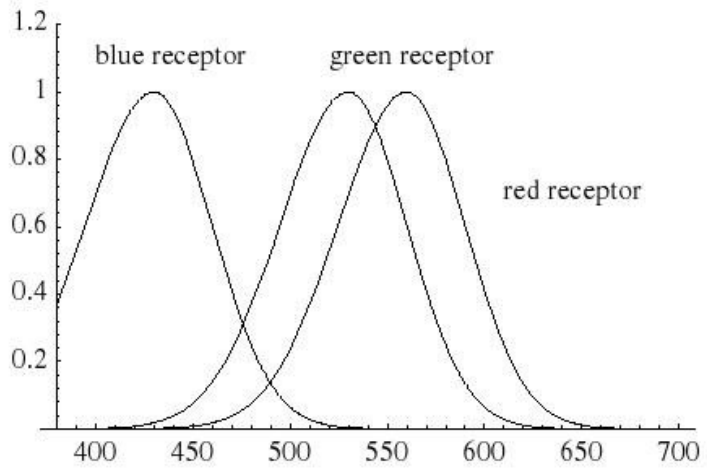


Logika rozmyta - aproksymacja funkcji

- Poszczególne pary wartości (x, y) należą w rożnym stopniu do każdej z 9 kategorii.
- Cieniowanie na rysunku - pokazuje członkostwo wejscia w kategorii dla której zmienna kontrolna przyjmuje wartość z_0.
- Wartości odpowiadające słabszemu zacieniowaniu - wartość zmiennej kontronej jest interpolacją sąsiednich wartości.
- Jeżeli funkcja kontrolna jest **gładka** - kilka punktów wystarcza do dobrego przybliżania innych wartości.
- **Niewielka liczba punktów** = niewielka liczba reguł wnioskowania.

Logika rozmyta - oko jako system rozmyty

- Oko można potraktować jak kontroler rozmyty w przypadku **widzenia kolorów**.
- Receptory - tylko dla 3 kolorów o **maksymalnym** wzbudzeniu dla niebieskiego, zielonego i czerwonego.
- Dużo pikseli $\Rightarrow$ potrzebna redukcja liczby typów detektorów.
- Światło o określonej długości - pobudza wszystkie typy receptorów ale z innym skutkiem $\Rightarrow$ powstaje kategoria rozmyta (członkostwo danej długości fali w kategoriach `niebieski`, `zielony` i `czerwony`).



Response function of the three receptors in the retina

Logika rozmyta - oko jako system rozmyty

- Dalsze przetwarzanie - oparte na powstałej kategorii oraz na dodatkowym kodowaniu (np. porównywaniu kolorów komplementarnych).
- Cały proces powoduje, że mieszanina kolorów postrzegana jest jako nowy kolor.
- Analiza oparta na fizjologii pokazuje, że 3 typy receptorów wystarczają do dobrego rozróżniania (dyskryminacji) kolorów.
- Ponownie - 3 receptory = mała ilość reguł.

Logika rozmyta - historia

- Arystoteles - zastanawiał się czy wszystkim stwierdzeniom można przypisać tylko wartość prawda lub fałsz.
- Jan Łukasiewicz (lata 20te XX w.) - pierwszy system logiki wielowartościowej; wprowadził trzecią wartość; aksjomatyzacja przez innych autorów.
- Od lat 30-tych - Gödel, Brouwer, von Neumann - możliwość nieskończonej ilości wartości.
- Logika rozmyta - Zadeh 1965 - kontinuum wartości; termin ten odnosi sie do całej rodziny wartości o różnych definicjach operatorów logicznych.
- Lata 70-te - wzrost zainteresowania logiką rozmytą i jej zastosowaniem w systemach ekspertowych; wtedy też pierwsze próby wykorzystania w systemach kontroli (Mamdani 1977).

Logika rozmyta - historia

- Obecnie - kontrolery rozmyte - w przemyśle i elektronice użytkowej.
- Istnieją czipy z operatorami rozmytymi zaimplementowanymi sprzętowo.
- Rynek wart miliardy dolarów.
- Operatory rozmyte - można włączyć do zbioru instrukcji normalnych procesorów.
- Badania w dziedzinie algorytmów uczenia dla systemów rozmytych.

Samo-organizacja

- **Sieci samo-organizujące się** - prawidłowa odpowiedź nie jest znana, nie ma zatem ilościowej miary błędu.
- Stosując odpowiedni proces uczenia prowadzi do wyznaczenia dobrze zdefiniowanych parametrów sieci.
- Poznaliśmy już system samoorganizujący się - sieć identyfikującą klastry wektorów.
- Dane wejściowe powodują adaptację parametrów - jeżeli odpowiednio kontrolować ten proces sieć zbuduje **wewnętrzną reprezentację** środowiska.
- Reprezentacja ta może być uaktualniania w sposób ciągły.
- Najbardziej znany i najpopularniejszy model samo-organizujących się sieci - **zachowujące topologię mapowanie Teuvo Kohonena**.

Samo-organizacja

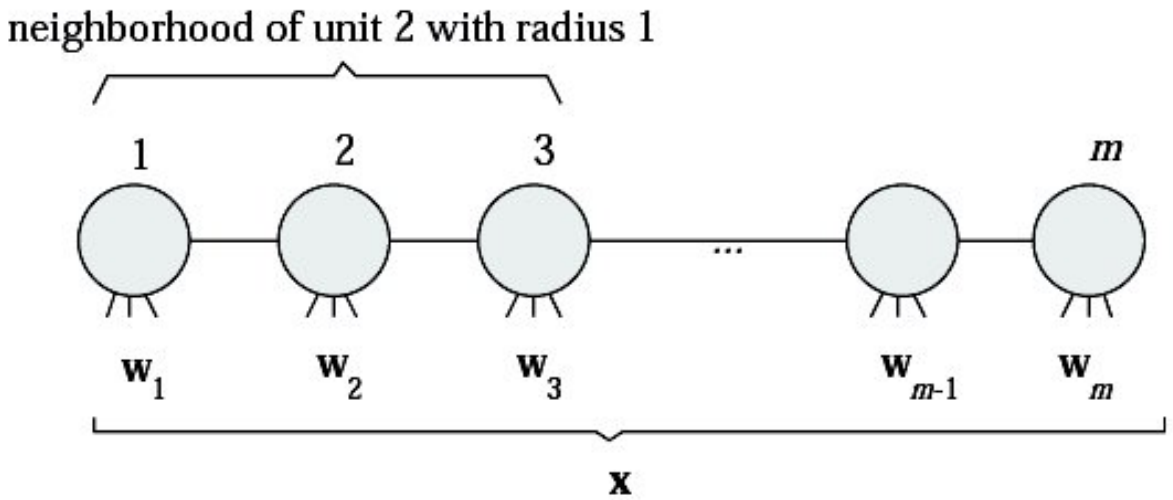
- Model Kohonena - podobnie mózg tworzy mapy np. dwu-wymiarową mapę wielowymiarowej przestrzeni w przypadku widzenia.
  - 3 rodzaje receptorów w widzeniu kolorowym + struktura, rodzaj powierzchni i położenie obiektów.
- Zachowywanie topologii - w przetwarzaniu sygnałów motorycznych - obszar odpowiadający ramionom bedzie znajdował sie blisko obszaru odpowiadającego dłoniom itp.
- Ważne jest aby podczas uczenia aktualizować także wagi sąsiednich neuronów - uczy to reakcji na podobne bodźce.

Samo-organizacja

- Chcemy stworzyć mapę n-wymiarowej przestrzeni używając łańcucha m jednostek
  - tzn. każda jednostka specjalizuje się w pewnym obszarze przestrzeni wejściowej.
- Do każdej jednostki podajemy wektor wejściowy $\mathbf{x}$ i obliczamy wzbudzenie.
- Odpowiednia jednostka powinna dać dla danego wektora wejściowego maksymalne wzbudzenie.
- **Jednostki Kohonena** - obliczają odległość euklidesową pomiędzy $\mathbf{x}$ oraz $\mathbf{w}$ - taka definicja jest lepsza dla pewnych zastowowań oraz łatwiejsza do wizualizacji.
- **Promień sąsiedztwa** równy r dla jednostki k definiujemy jako jednostki $k - r$ oraz $k + r$.
  - jednostki na końcach łańcucha mają sąsiedztwo asymetryczne

Samo-organizacja

- Definiujemy także **funkcję sąsiedztwa** ϕ; wartość $\phi(i, k)$ reprezentuje siłę sprzeżenia danych jednostek podczas treningu.
- Prostym wyborem jest zdefiniowanie $\phi(i, k) = 1$ dla jednostek i w sąsiedztwie o promieniu r oraz $\phi(i, k) = 0$ dla pozostałych jednostek.



A one-dimensional lattice of computing units

Algorytm Kohonena:

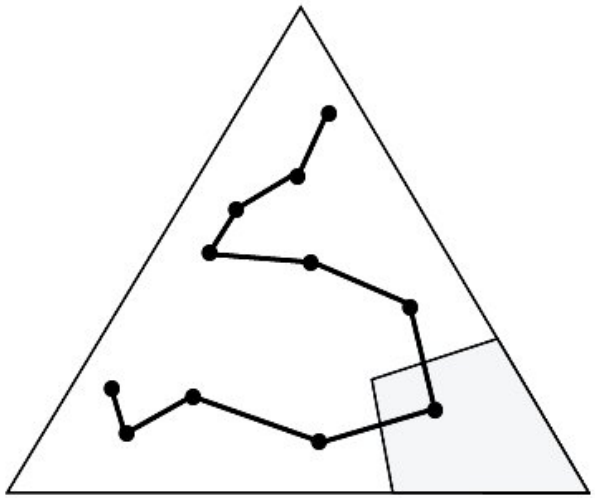
1. n-wymiarowe wektory wag $\mathbf{w_1}, \mathbf{w_2}, ..., \mathbf{w_m}$ dla m jednostek obliczeniowych - wybierane przypadkowo; ponadto wybieramy - początkowy promień r, stałą uczenia η oraz funkcję sąsiedztwa ϕ.
2. wybieramy wektor wejść ξ używając żądanego rozkładu prawdopodobieństwa na przestrzeni wejść.
3. wybieramy jednostkę k o najwyższym wzbudzeniu tzn. o minimalnym dystansie pomiędzy w_i oraz ξ $(i = 1, ..., m)$.
4. wektory wag są uaktualniane zgodnie z regułą: $\mathbf{w}_i \leftarrow \mathbf{w}_i + \eta\phi(i,k)(\xi - \mathbf{w}_i), \quad i = 1, ..., m.$
5. stop - jeżeli osiągnieta została maksymalna liczba iteracji; w przeciwnym razie zmodyfikuj zgodnie z *planem* η oraz ϕ i przejdź do kroku 2.

Samo-organizacja

- Modyfikacja wag przyciąga je w kierunku wejścia ξ; powtarzając ten proces spodziewamy się otrzymania rozkładu jednorodnego wag w przestrzeni wejść (jeżeli wejścia także wybralismy wg. rozkładu jednorodnego).
- Promień redukujemy zgodnie z wcześniej przyjętym **planem**.
- Gdy wektor wag danej jednostki przyciągany jest do pewnego obszaru przestrzeni wejściowej - wektory sąsiednich jednostek także są przyciągane (w mniejszym stopniu).
- W procesie uczenia - zmniejsza się zarówno sąsiedztwo jak i ϕ oraz η - wpływ na jednostki sąsiednie jest stopniowo redukowany.

Samo-organizacja

- Rysunek pokazuje rezultat uczenia Kohonena dla obszaru trójkątnego. **Kropka = jednostka**.
- Wektory wag osiągają rozkład który zamienia jednostkę w reprezentację małego obszaru przestrzeni wejść.
- Punkt w prawym dolnym rogu - odpowiada najwyższym wzbudzeniem w przypadku wektorów w zacienionym obszarze.
  - Dla strategii aktywacji typu zwyciezca-bierze-wszystko - bedzie on jedynym aktywującym się neuronem.



Map of a triangular region

Samo-organizacja - historia

- Od czasów Kohonena - wiele zastosowań.
- Prosty model - wyjaśnia niektóre zjawiska biologiczne.
- Kombinatoryka np. problem komiwojażera (poszczególne miasta okrąża się pierścieniem jednostek).
- Lata 80-te - pierwsze zastosowania w robotyce.
- Rozpoznawania mowy np. system rozpoznający słowa na podstawie obserwacji ścieżki w przestrzeni fonemów.
- Matematyka - badania procesów samo-organizujących się.

Algorytmy genetyczne

- **Algorytmy genetyczne** - algorytmy optymalizacji zainspirowane **doborem naturalnym i genetyką**.
  - znajdują zastosowanie w wielu dziedzinach
  - algorytmy są proste do zrozumienia i zaprogramowania
  - mimo zalet nie są tak rozpowszechnione jak sieci neuronowe
- Idea użycia **populacji rozwiązań** do problemów optymalizacji w inżynierii - lata 50te i 60te.
- John Holland (psycholog i inżynier) - 1975 - Adaptation in Natural and Artifical Systems.
- Znacznie szersze zastosowania niż estymacja nieznanych parametrów modelu fizycznego.

Algorytmy genetyczne

Typowy algorytm genetyczny składa się z:

1. Pewnej liczby (= **populacji**) przypuszczalnych **rozwiązań** problemu.
2. Metody obliczania **jak dobre jest rozwiązanie**.
3. **Metody mieszania/krzyżowania** fragmentów lepszych rozwiązań - powstają nowe, średnio biorąc lepsze rozwiązania.
4. **Operatora mutacji** zapobiegający utracie różnorodności.

Nie ma skomplikowanej matematyki ani trudnych algorytmów.

Algorytmy genetyczne

- Przykłady zastosowań:
  - przetwarzanie obrazu
  - struktura białek
  - obwody scalone
  - trajektorie statków kosmicznych
  - analiza szeregów czasowych
  - fizyka ciała stałego
  - robotyka
  - automatyczna ewolucja programów komputerowych
  - rozpoznawanie twarzy
  - uczenie i projektowanie sieci neuronowych
  - kontrola

Algorytmy genetyczne - przestrzenie rozwiązań

- **Przestrzeń rozwiązań** - należy zdefiniować **miarę odległości między rozwiązaniami** oraz dla każdego rozwiązania **miarę sukcesu/przystosowania** jako rozwiązania.
- Przykład - znalezienie dwóch parametrów w modelu matematycznym maksymalizującym siłę nośną działającą na skrzydło samolotu.
- Jeżeli odpowiedni wykres posiada maksimum/maksima to lepiej przystosowane maksima będą zajmowały te obszary.
- **Większość zastosowań** - celem jest dokładna identyfikacja optimum.
    - czasem np. kontrola w czasie rzeczywistym - wystrcza znalezienie jakiegokolwiek punktu powyżej ustalonej granicy przystosowania
    - inne przypadki np. architektura - potrzebne znalezienie dużej liczby różnych dobrze przystosowanych punktów

Algorytmy genetyczne

- Algorytm GA inicjalizujemy populacją rozwiązań - zwykle przypadkowo rozproszoną w przestrzeni rozwiązań (potem selekcja, krzyżowanie i mutacje).
- Typowo populacje zapisujemy **binarnie**.
  - czasem jako wartości rzeczywiste lub w sposób naśladujący naturalną strukturę danych problemu
- **Selekcja** - wywiera na populację presję podobnie do doboru naturalnego - lepiej przystosowane osobniki mają większą szansę przenieść informację którą zawierają (DNA) do następnego pokolenia.
- **Krzyżowanie** - powzala wymieniać informacje podobnie do procesu rozmnażania.
  - krzyżowanie jednopunktowe - jedna z metod - w wybranych przez operator selekcji osobnikach losowany jest punkt ciągu binarnego a osobniki wymieniają informację na prawo od niego.
- **Mutacja** - losowo zmienia wartość pojedynczych bitów.
- Po selekcji, krzyżowaniu i mutacji początkowej populacji powstaje nowa populacja (nowe pokolenie) - proces powtarzamy określoną ilość razy lub do spełnienia odpowiedniego kryterium.

Algorytmy genetyczne - przykład

- Przed zastosawaniem GA do konkretnego problemu potrzeba określić:
  - metodę kodowania parametrów np. binarnie
  - jak wymieniać informację między członkami populacji
  - wielkość populacji - typowo $20 - 1000$ (może być mniejsza lub znacznie większa)
  - jak stosować ideę mutacji
  - kryterium zatrzymania algorytmu
- Istnieją publikacje na temat różnych sposobów kodowania, wyboru wielkości populacji, różnych mechanizmów krzyżowania i różnych częstości mutacji - nie wiadomo na ile są ogólne.
- Praktycznie - analizuje się podejścia zastosowane w podobnych problemach i wybiera to które wydaje się najodpowiedniejsze.
- Trywialny przykład - znaleźć maksimum funkcji $f(x) = x^2$ dla całkowitego $0 \leq x \leq 4095$.

Algorytmy genetyczne - przykład

Jeden z możliwych algorytmów:

1. Utwórz populację 8 losowych ciągów liczb binarnych długości 12.
2. Zdekoduj ciągi do liczb całkowitych np. $000000000111 = 7$, $111111111111 = 4095$.
3. Przetestuj otrzymane liczby jako rozwiązanie problemu - funkcja przystosowania $f(x)$ (np. $x = 7$ ma przystosowanie 49).
4. Wybierz najlepszą połowę populacji do następnego pokolenia.
5. Pogrupuj losowo ciągi-rodziców w pary i losowo zastosuj jednopunktowe krzyżowanie - powstają ciągi-dzieci.
6. Zastosuj do dzieci mutacje - z prawdopodobieństwem 1/6 zmień 0 na 1 lub odwrotnie.
7. Nowe ciągi wraz z rodzicami tworzą nową populację (8 ciągów).
8. Wróć do kroku 2 i powtarzaj do uzyskania 50 pokoleń.

Algorytmy genetyczne - przykład

- Przykładowe krzyżowanie - wylosowany 3 bit:

$$000/100011100 \Rightarrow 000001101010$$

$$111/001101010 \Rightarrow 111100011100$$

- Przykładowa populacja początkowa:

Nr	*ciag*	*x*	*przystosowanie*
1	110101100100	3428	11751184
2	010100010111	1303	1697809
3	101111101110	3054	9326916
4	010100001100	1292	1669264
5	011101011101	1885	3553225
6	101101001001	2889	8346321
7	101011011010	2778	7717284
8	010011010101	1237	1530169

Algorytmy genetyczne - przykład

1, 3, 6, 7 - największe przystosowanie

Nr	*ciag*	*x*	*przystosowanie*
1	110101100100	3428	11751184
2	101111101110	3054	9326916
3	101101001001	2889	8346321
4	101011011010	2778	7717284

Pary wybierane przypadkowo np. 1 z 2 oraz 3 z 4.
Zaniedbując mutacje tworzymy nowe pokolenie:

Algorytmy genetyczne - przykład

Nr	*ciag*	*x*	*przystosowanie*
1	11/0101100100	3428	11751184
2	10/1111101110	3054	9326916
3	101101/001001	2889	8346321
4	101011/011010	2778	7717284
5	111111101110	4078	16630084
6	100101100100	2404	5779216
7	101101011010	2906	8444836
8	101011001001	2761	7623121

- Wyjściowa populacja - średnie przystosowanie 5065797, maksymalne - 11751184.
- Drugie pokolenie - średnie - 8402107, maksymalne - 16630084.

Algorytmy genetyczne - przykład

Rodzicami kolejnego pokolenia będą:

Nr	*ciag*	*x*	*przystosowanie*
1	110101100100	3428	11751184
2	101111101110	3054	9326916
3	101101011010	2906	8444836
4	101011001001	2761	7623121

- Nie występuje ciąg z 1 na końcu - **maksimum nie zostanie osiągnięte**.
- Może powstać 111111111110, który zdominuje populację - wskazówka, że **brak mutacji** może być problemem.
- Mutacja powoduje zachowanie **różnorodności** pierwotnej populacji.
  - pierwotna populacja zawierała 1 na każdej pozycji - wiec mutacja niekoniecznie wynajduje nową informację
  - to samo z mutacją - otrzymuje się globalne optimum 111111111111.

Algorytmy genetyczne - podsumowanie

Różnice między GA i tradycyjnymi algorytmami:

- Równoległa manipulacja populacją rozwiązań.
- Specjalne kodowanie rozwiązań np. binarne.
- Używanie losowych operatorów.

Algorytmy genetyczne

Terminologia biologiczna.

- chromosom - ciąg binarny (rozważamy pojedyncze ciągi więc jest to także genotyp)
- fenotyp (organizm) - rezultat ekspresji genotypu w środowisku - rozwiązanie lub zbiór nieznanych parametrów

Uniwersalność.

- GA - w naturalny sposób uniwersalne, przy odpowiednim dostosowaniu operatorów i odpowiednim kodowaniu danych mogą być bardzo wydajne.
  - przy wystarczającej informacji o przestrzeni rozwiązań zawsze można skonstruować metodę lepszą niż GA
  - jednak otrzymanie takie informacji w wielu przypadkach jest niemal tak trudne jak rozwiązanie problemu
  - specjalny algorytm ma mały zakres zastosowania a GA szeroki
  - niewielkie zmiany w GA poprawiają wydajność z jednoczesnym zachowaniem uniwersalności

Algorytmy genetyczne - zmienne rzeczywiste

- Użycie zmiennych rzeczywistych w algorytmie - zmiana definicji operatorów.
- Najpopularniejsze podejście - liniowe odwzorowanie pomiędzy zbiorami liczb rzeczywistych i reprezentacji binarnych o ustalonej długości.
- Ciąg binarny transformujemy na dziesiętną liczbę całkowitą z; linowa transformacja na liczbę rzeczywistą:

$$r = mz + c.$$

- m i c wynikają z minimalnych i maksymalnych wartości r i z

$$r_{min} = mz_{min} + c$$

$$r_{max} = mz_{max} + c,$$

Algorytmy genetyczne - zmienne rzeczywiste

- ogólne wzory:

$$m = \frac{r_{max} - r_{min}}{z_{max} - z_{min}}$$

$$c = r_{min} - m z_{min}$$

- Najmniejsza liczba binarna 000...0, tzn. $z_{min} = 0$; 1111...1 - $z_{max} = 2^l + 1$ (l - długość ciągu binarnego); w tym przypadku:

$$m = \frac{r_{max} - r_{min}}{2^l - 1}$$

$$c = r_{min},$$

- a odpowiednia transformacja to:

$$r = \frac{r_{max} - r_{min}}{2^l - 1} z + r_{min}.$$

Algorytmy genetyczne - zmienne rzeczywiste

- Przykład - szukamy parametru x w przedziale $[2.2, 3.9]$.
- Liczbie binarnej 10101 odpowiada dzisiętna $z = 21$ a rzeczywista

$$r = \frac{3.9 - 2.2}{2^5 - 1} 21 + 2.2 = 3.3516.$$

- Kolejną liczbą binarną jest $10110 = 22 \Rightarrow r = 3.4065$ - nie można określić liczb z przedziału $[3.356, 3.4065]$.
- Dokładność można zwiększyć
  1. redukując rozmiar przestrzeni rozwiązań
  2. zwiększając liczbę bitów w ciągu
  3. inna reprezentacja liczb - w przypadku większości problemów nie jest to konieczne
- $l = 20$ daje dokładność ponad 10^{-6}.

Algorytmy genetyczne - problemy z wieloma zmiennymi

- Jeżeli M zmiennych reprezentowanych jest pod-ciągami długości l - łączenie w populację ciągów długości L

$$L = \sum_{j=1}^{M} l_j.$$

- Nie jest wymagane by pod-ciągi były tej samej długości $\Rightarrow$ kontrola dokładności parametrów $\Rightarrow$ możliwość znacznego przyspieszenia poszukiwań.
- Punkt krzyżowania musi być wewnątrz pod-ciągu opisującego parametr a nie pomiędzy parametrami.

Algorytmy genetyczne - mutacje

- Łatwe do zaimplementowania w reprezentacji binarnej - każdy bit zmieniany jest z prawdopodobieństwem mutacji P_m.
- $P_m = 0.001$ - średnio jeden bit na 1000 ulegnie mutacji.
- P_m zależy od problemu; często stosuje się $P_m = 1/L$ lub $P_m = 1/(N\sqrt{L})$ (N - wielkość populacji).
- Uważa się, że w większości problemów zbyt niska częstość mutacji jest mniej szkodliwa niż zbyt wysoka.

Algorytmy genetyczne - selekcja

- Odrzucanie 50% populacji jest bardzo prostą selekcją i nie najczęściej używaną.
  - nie ma odróżnienia między dobrymi a bardzo dobrymi
  - nie najlepiej dostosowane osobniki są zawsze odrzucane (redukujemy różnorodność genetyczną) a mogłyby przechodzić do kolejnego pokolenia z niższym prawdopodobieństwem
- Popularnym operatorem selekcji jest operator proporcjonalny do przystosowania (zwany też ruletką) - prowdopodobieństwo wyboru jest proporcjonalne do przystosowania danego osobnika.
  - ruletka - można wyobrazić sobie populację tworzącą ruletkę, gdzie część koła ruletki odpowiadająca danemu osobnikowi jest proporcjonalna do jego przystosowania
  - metodę ruletki stosujemy 2 razy aby wybrać parę podlegają krzyżowaniu
  - metoda ta nie gwarantuje wyboru żadnego elementu nawet najlepiej przystosowanego - dla pewnych problemów może być to korzystne bo algorytm jest spowalniany pozwalając lepiej przeszukać przestrzeń rozwiązań
  - elitaryzm - najlepiej przystosowany zawsze przeżywa (nie podlega też mutacji ani krzyżowaniu) - często przyspiesza algorytm

Algorytmy genetyczne - krzyżowanie

- W algorytmie zwanym mały algorytm genetyczny (LGA - Little Genetic Algorithm) używane jest krzyżowanie jednopunktowe (w naturze stwierdzono od 1 do 8 punktów krzyżowania).
- Para wybranych osobników podlega krzyżowaniu z prawdopodobieństwem P_c (generowana jest liczba losowa R_c, krzyżwanie następuje jeżeli $R_c \leq P_c$).
  - typowo $0.4 \leq P_c \leq 0.9$
  - $P_c = 0.5$ - połowa nowej populacji tworzona jest przez selekcję i krzyżowanie a druga połowa tylko przez selekcję

Algorytmy genetyczne - LGA

1. Wygeneruj początkową populację ($g = 1$) losowych ciągów binarnych długości $\sum_{k=1}^{M} l_k$ (M - liczba niewiadomych).
2. Zdekoduj każdego osobnika i do liczby całkowitej $z_{i,k}$ i dalej rzeczywistej $r_{i,k}$ (szukane parametry).
3. Przetestuj wszystkie osobniki otrzymując przystosowania f_i.
4. Użyj selekcji proporcjonalnej do przystosowania do wyboru par osobników i zastosuj z prawdopodobieństwem P_c krzyżowanie jednopunktowe $\Rightarrow$ nowa populacja.
5. Do każdego bitu zastosuj mutację z prawdopodobieństwem P_m.
6. Jeżeli stosujemy elitaryzm - jeżeli nowa populacja nie zawiera elementu o przystosowaniu co najmniej takim jak najlepiej przystosowany element wyjściowej populacji, zastąp losowo wybrany element najlepiej przystosowanym.
7. Zastąp starą populację nową.
8. $g = g + 1$ i przejdź do kroku 2 (chyba że osiągnięto zakładaną liczbę pokoleń G).

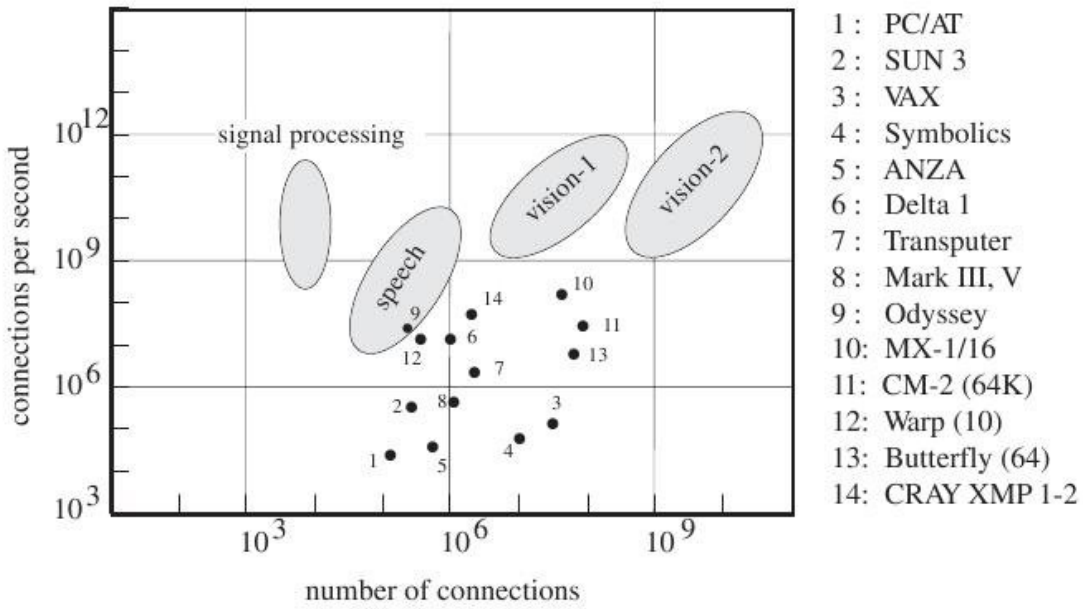
Hardware dla sieci neuronowych

- Jak najlepiej przetwarzać informacje w sieci neuronowej z użyciem **urządzeń elektronicznych**?
  - jak najlepiej emulować silną równoległość świata biologicznego?
- Sieci neuronowe są atrakcyjną opcją w zastosowaniach jeżeli ich niejawną równoległość można uczynić jawną z użyciem odpowiedniego sprzętu.
  - dobrze jest ograniczyć komunikację do lokalnej wymiany danych
- Istnieją dwie alternatywy:
  1. **programowa symulacja** na konwencjonalnych komputerach,
  2. **specjany hardware** (zapewnia drastyczne zmniejszenie czasu wykonywania zadań).
- Mimo to symulacja może być przydatna do **rozwijania i testowania nowych algorytmów**.
- **Duże sieci** - symulacja nie wystarcza - **czas uczenia sieci** może wzrastać wykładniczo z jej rozmiarem.

Hardware dla sieci neuronowych - wydajność

- W wielu zastosowaniach sieci neuronowe mogą być wydajniejsze niż konwencjonalne metody - rozpoznawanie obrazów i mowy, robotyka, przetwarzanie sygnałów, kompresja danych, analiza statystyczna i optymalizacja.
- Najistotniejszym czynnikiem w budowie sieci neuronowej jest - **rozmiar sieci i moc obliczeniowa**.
  - **złożoność/pojemność** sieci najlepiej określić przez **liczbę wag** (nie liczbę jednostek - uczenie), wielkość ta poza złożonością informuje o tym jak trudne będzie uczenie sieci.
  - **wydajność implemen** - **liczba połączeń na sekundę** (ang. connections per second, cps) - liczba paczek danych przekazywana przez połączenia sieci na sekundę.
- **Wydajność procesu uczenia** - mierzy się w liczbie uaktualnień połączeń sieci na sekundę (connections updates per second, cups).

Hardware dla sieci neuronowych - wydajność



Performance and capacity of different implementations of neural networks and performance requirements for some applications [Ramacher 1991]

- PC/AT - stary komputer osobisty: 10*Kcps*, CM-2 (równoległy SIMD, 64K procesorów): 10*Mcps* - może radzić sobie z prostym rozpoznawaniem mowy ale już nie z wersją wyrafinowaną lub rozpoznawaniem obrazów (symulacją widzenia).
- Stosunek **cena/wydajność** - nie jest dobry w przypadku konwencjonalnych komputerów.

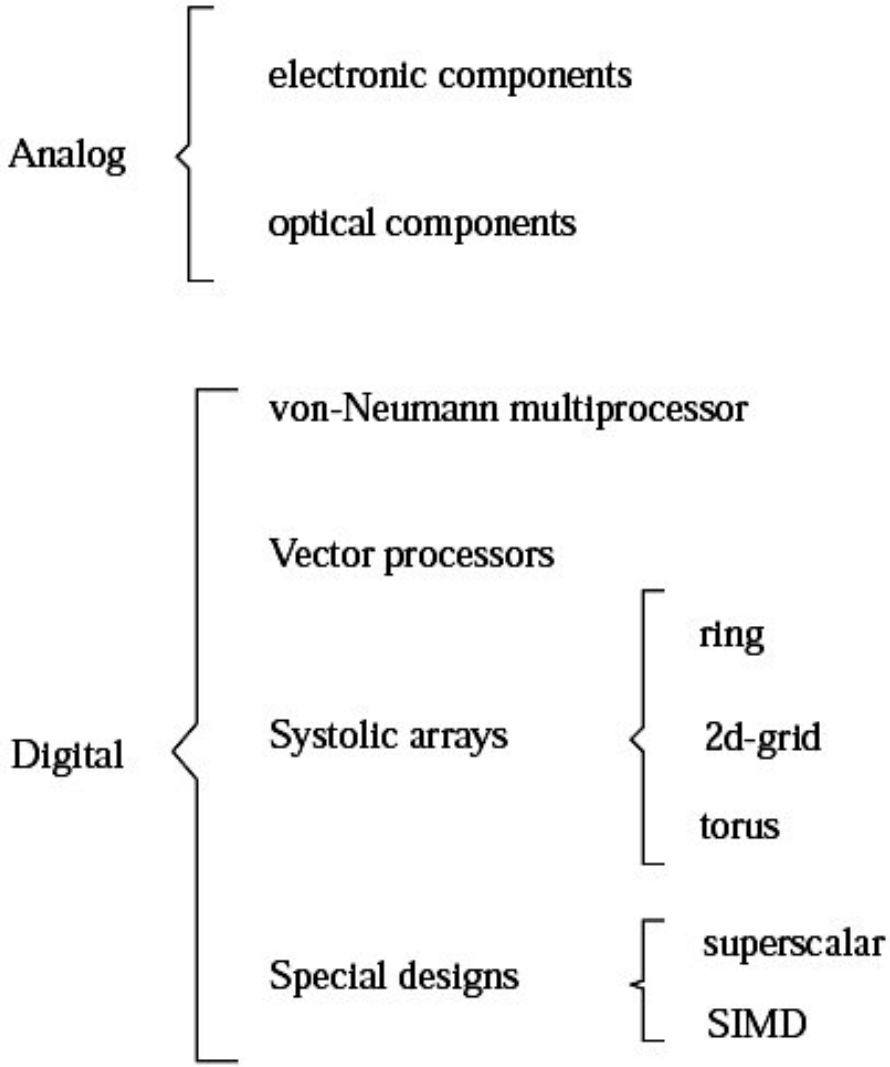
Hardware dla sieci neuronowych - typy neurokomputerów

- W klasyfikacji ważne są 3 czynniki:
  1. **rodzaj sygnałów** użytych w sieci
  2. implementacja **wag** sieci
  3. funkcje **scalające** i funkcje **wyjścia** jednostek

Ad.1. Sygnały - analogowe (natężenie prądu lub różnica potencjałów) lub cyfrowe (wartości dyskretne).
Ad.2. Wagi analogowe - oporniki lub tranzystory o liniowej funkcji odpowiedzi w pewnym zakresie; wagi cyfrowe - mnożenie cyfrowe.

- Neurokomputery **hybrydowe** - połączenie obwodów cyfrowych i analogowych.
- Podejście analogowe - układy elektroniczne lub optyczne (prototypy).
- Podejście cyfrowe - architektóry równoległe lub szeregowe.
  - każdy procesor - może zajmować się częścią zbioru treningowego
  - procesory wektorowe - zoptymalizowane do operacji macierzowych

Hardware dla sieci neuronowych - typy neurokomputerów



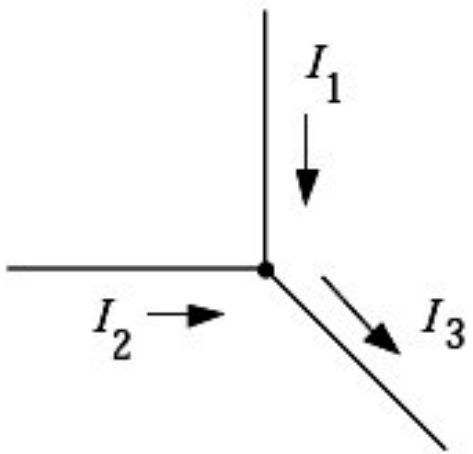
Taxonomy of neurosystems

Hardware dla sieci neuronowych - typy neurokomputerów

- Macierze systoliczne - regularne układy jednostek obliczeniowych; komunikują się tylko z bezpośrednimi sąsiadami.
  - zaproponowane do szybkiego mnożenia macierzy - ważne w sieciach warstwowych.
- specjalne czipy typu superskalarnego lub systemy zawierające wiele identycznych procesorów używanych w trybie SIMD.
- **Porównanie:**
  - **analogowe** - większa gęstość układu oraz potrzebują mniejszą moc.
  - **cyfrowe** - większa precyzja i elastyczność programowania, możliwość pracy z **sieciami wirtualnymi** - czyli nie odzwierciedlonymi fizycznie w strukturze sieci (możliwość zwiekszenia liczby jednostek).

Hardware dla sieci neuronowych - sieci analogowe

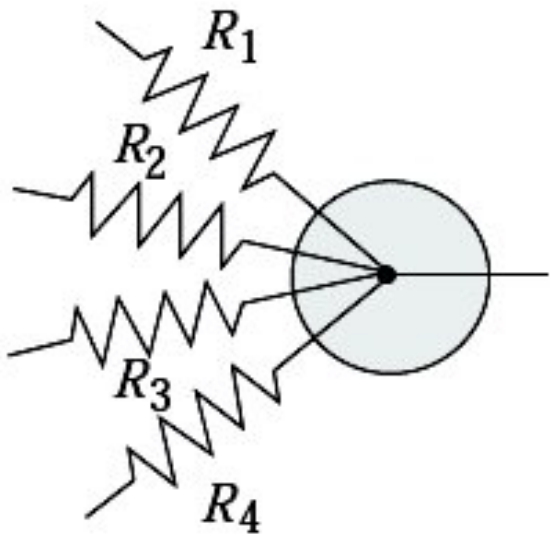
- Sygnały reprezentowane prądami (tzn. natężeniami) lub napięciami - można myśleć o nich, że operują na liczbach rzeczywistych.
- **Kodowanie** - prądy - dodawanie jest łatwo obliczyć:



Addition of electric current

Hardware dla sieci neuronowych - sieci analogowe

- Dodawanie natężeń - bardziej skomplikowane - aby dodać V_1 i V_2 - wyjście o napięciu V_1 musi być odniesieniem dla drugiego z napięć (wiecej potencjałów - nie jest to łatwe do zaimplementowania).
- **Funkcja scalająca** - połączenie wejść do wspólnego punktu - suma prądów będzie przetworzona przez jednostkę.
- **Wagi sieci** - użycie zmiennych oporów (tak zrobił Rosenblatt w pierwszych perceptronach) - $V = RI$:

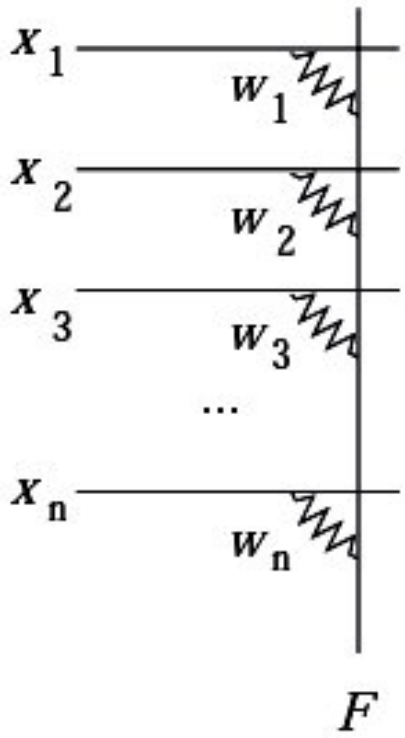


Network of resistances

Hardware dla sieci neuronowych - sieci analogowe

- Lata 70-te i 80-te - powstała pewna ilość projektów analogowych.
- **Grupa Carvera Meada** (Caltech) - studiowali jak zmniejszyć rozmiar sieci oraz probór mocy.
  - tranzystory
  - krzemowa *retina*
  - krzemowa *colchea*
- Lata 50-te - Karl Steinbuch - model uczenia - **Learnmatrix** - n wejść jako napięcia, następnie opory zmieniają napięcie w ważone natężenia.
  - zbiór kolumn tego typu pozwala obliczyć w sposób równoległy iloczyn wektor-macierz.
  - Steinbuch zbudował pierwsza pamięć skojarzeniową jako hardware.

Hardware dla sieci neuronowych - sieci analogowe



A column of the *Lernmatrix*

Hardware dla sieci neuronowych - sieci analogowe

- **Tranzystory VLSI** (Very Large Scale Integration).
- Mogą używać półprzewodnikowych **tranzystorów polowych** (FET - field effect transistor) - nieliniowa krzywa odpowiedzi napięcie-natężenie.
  - nieliniowość $\Rightarrow$ dogodne do zastosowania jako *przełączniki cyfrowe*.

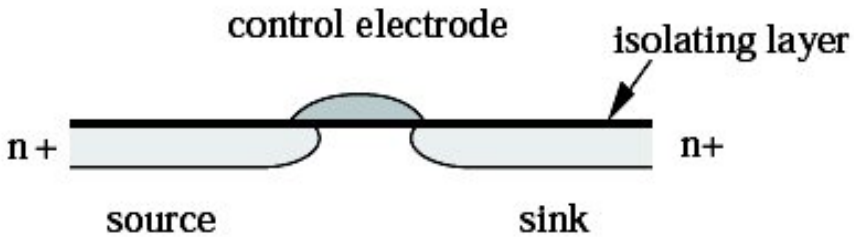
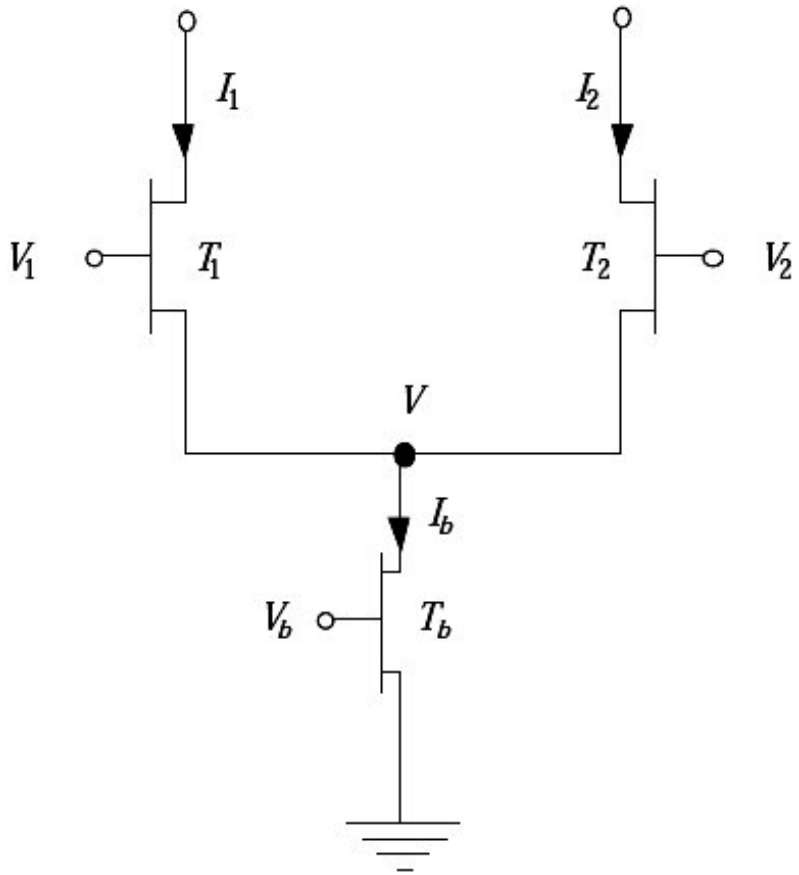


Diagram of a field effect transistor (FET)

* $n+$ oznacza nadmiar ładunku dodatniego (odpowiednie domieszkowanie półprzewodnika)
* przepływ ładunku - gdy elektroda kontrolna jest naładowana dodatnio powyżej pewnego progu (eletrony są przyciągane i mogą przeskoczyć przez przerwe)

Hardware dla sieci neuronowych - sieci analogowe

- Tranzystor FET **zamienia** więc **różnicę potencjałów** (pomiędzy źródłem a elektrodą kontrolną) **w prąd**.
- Można zbudować mały obwód FET-ów mnożący wartość prądu i wartość różnicy potencjałów.



Hardware dla sieci neuronowych - sieci analogowe

- Prąd wytworzony na tranzystorze FET zależy wykładniczo od różnicy potencjałów:

$$I_1 = I_0 e^{cV_1 - V}, \quad I_2 = I_0 e^{cV_2 - V}$$

Zatem:

$$I_b = I_1 + I_2 = I_0 e^{-V}(e^{cV_1} + e^{cV_2}),$$

$$I_1 - I_2 = I_b \frac{e^{cV_1} - e^{cV_2}}{e^{cV_1} + e^{cV_2}} = I_b \tanh \frac{c(V_1 - V_2)}{2}$$

- Zatem - różnica $V_1 - V_2$ daje nieliniowy rezultat $I_1 - I_2$.
- Przypomnienie - tangens hiperboliczny jest symetrycznym sigmoidem używanym w algorytmie propagacji wstecznej; dla **małych argumentów** $\tanh(x) \approx x$, więc:

$$I_1 - I_2 = \frac{cI_b(V_1 - V_2)}{2}.$$

- **Podsumowanie** - **FET** - ważą sygnały (wykładnicza zależność), scalają (I_b) oraz dają nieliniowe wyjście ($I_1 - I_2$).

Hardware dla sieci neuronowych - sieci analogowe

- **Tranzystory przechowujące ładunek** - mogą posłużyć do mnożenia.
- Problem FET - potrzebne i napięcia i natężenia, wagi trzeba przekształcić w różnice potencjałów i utrzymywać ją niezmienioną jeżeli nie ma aktualizacji wag - **łatwiej** było by wagi przechować i użyć gdy będą potrzebne.
- Schemat tranzystora przechowującego ładuenk jest podobny do FET:

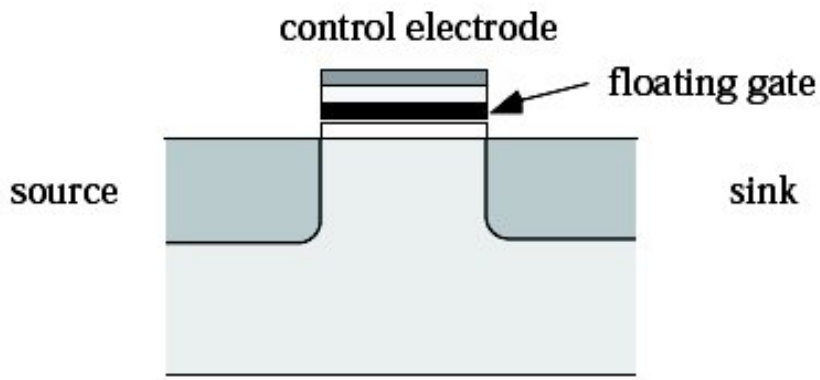


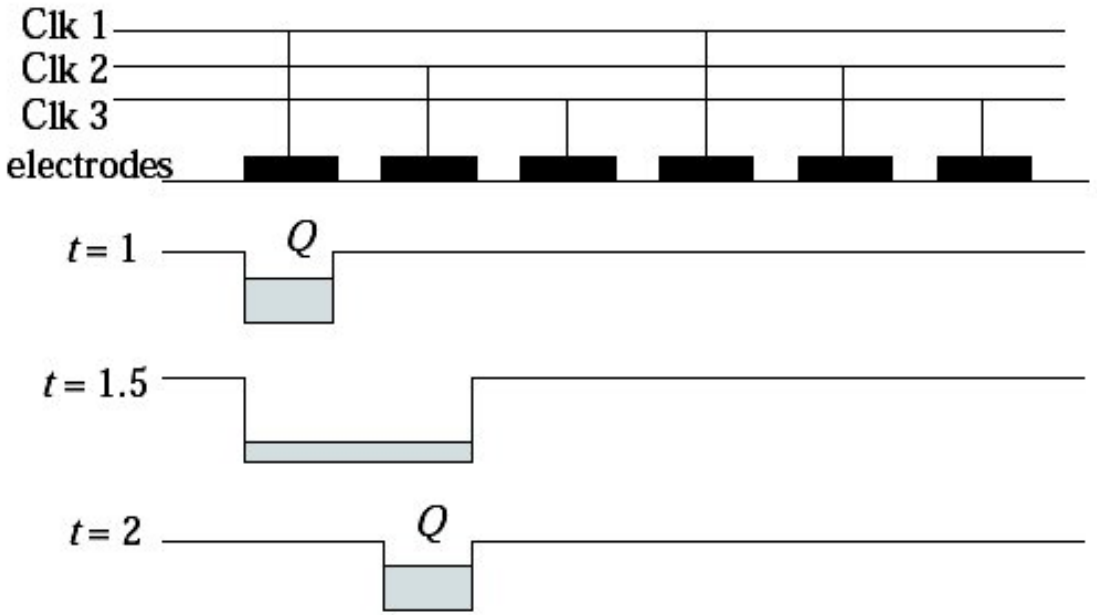
Diagram of a floating gate transistor

Hardware dla sieci neuronowych - sieci analogowe

- Różnica - warstwa metaliczna - izoluje elektrodę kontrolną i resztę tranzystora.
- Dzięki izolacji - warstwa ta może przechowywać ładunek (przez długi czas) jak kondensator.
- Ładowanie - przyłożenie różnicy potencjałów.
- Jest to pamięć analogowa.
- Ładunek modyfikuje efektywność działania elektrody kontrolnej - przepływający prąd zależy liniowo (w pewnym przedziale od zgromadzonego ładunku) - różnica potencjałów jest ważona i przekształcana w prąd.
- **Uczenie**: zmiany wag = zmiany zgromadzonego ładunku.

Matryce CCD (charge coupled devices) - sygnał elektryczny proporcjonalny do padającego światła.

- Przechowują ładunek dynamicznie - wagi jako ładunek mogą krążyć w pętli - w pewnym jej punkcie są czytane i mnoóone przez wejście.
- Obwód CCD transportujący ładunki - łańcuch elektrod na powierzchni półprzewodnika - synchronizacja zegarem tak, że w danym czasie tylko jedna elektroda osiąga maksymalny potencjał.



Hardware dla sieci neuronowych - sieci analogowe

- Ładunek Q - zgromadzony w materiale półprzewodnika - tworzy studnię potencjału, która z czasem rozmywa się.
- Kolejna elektroda tworzy studnię w kolejnym położeniu.
- Można zbudować pamięć.

Hardware dla sieci neuronowych - sieci cyfrowe

- Sygnały i parametry sieci - cyfrowe.
- **Dowolna dokładność** - zwiększanie długości słowa.
- W algorytmach uczenia opartych na gradiencie - potrzebna większa dokładność niż można uzyskać w sieciach analogowych (8,9 bitów)
  - inna wady sieci analogowych - duża zmienność charakterystyki elektrycznej tranzystorów (nawet na tym samym czipie).
  - sieci analogowe - do zastosowań gdzie odchylenia statystyczne można tolerować np. widzenie.
- Obecne komputery są cyfrowe - sieć może być jedną z części większego urządzenia $\Rightarrow$ łatwiejsza integracja oprogramowania (**koprocesor neuronowy**).

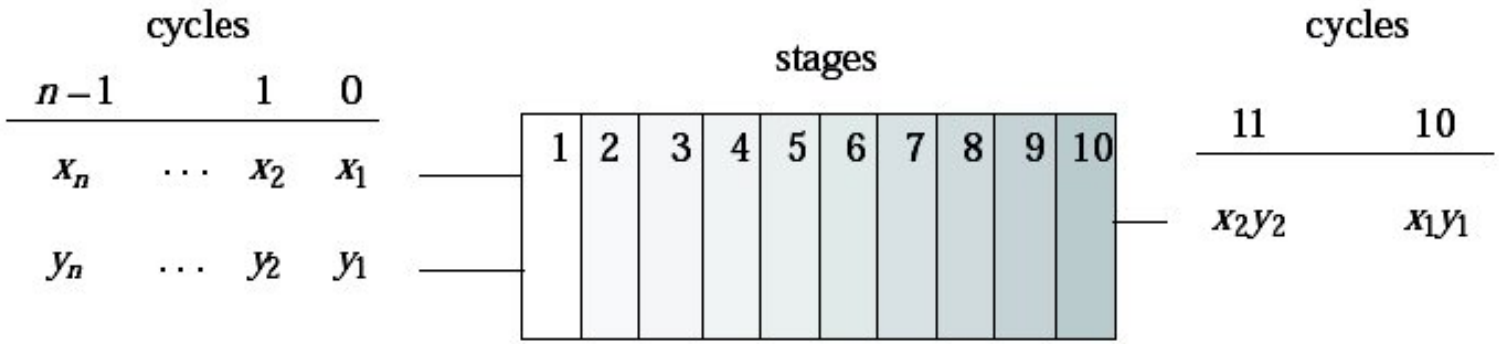
Hardware dla sieci neuronowych - sieci cyfrowe

- **Podstawowe zagadnienie - ile bitów** użyć do reprezentacji wag i sygnałów.
- Implementacja czysto programowa - reprezentacja zmienno-przecinkowa.
- Operacje zmienno-przecikowe - potrzeba więcej cykli do obliczenia niż liczby całkowite (chyba, że użyje się bardzo skomplikowanych układów) $\Rightarrow$ wiele neurokoputerów używa stało-przecinkowej reprezentacji - nie może to wpływać na zbieżność algorytmów uczenia sieci.

- Prawie wszystkie modele sieci - potrzebny iloczyn skalarny wektora wag i wejścia - **procesory wektorowe** (przetwarzają w pojedynczych cyklach całe wektory) zostały zbudowane z myślą o takich zastosowaniach.
- Rejestry skalarne oraz wektorowe (przechowują całe wektory).
- Operacje na wektorach - składowe wektora są czytane z rejestru kolejno i przekazywane do jednostki arytmetycznej.

Hardware dla sieci neuronowych - sieci cyfrowe

Jednostka arytmetyczna - podzielona na etapy:



Multiplier with 10 pipeline sections

Hardware dla sieci neuronowych - sieci cyfrowe

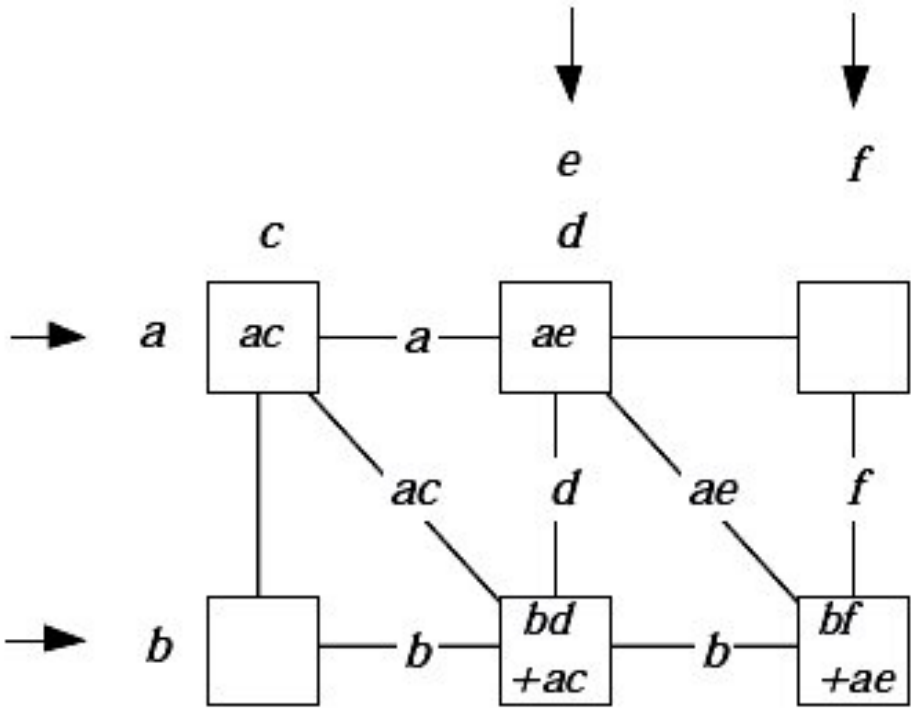
- **Macierze systoliczne** - bardzo szybko wykonują operacje algebry liniowej.
- Systolic - podobieństwo do przepływu krwi w sercu - indicating the maximim arterial pressure occuring during contraction of the left ventricle of the heart
- Regularne struktury jednostek VLSI (najczęściej 1- lub 2-wymiarowe) - komunikują sie tylko **lokalnie**.
- Informacja dostarczana jest na obrzeżach i transportowana synchronicznie od jednego etapu do drugiego.
- Mnożenie **wektor-macierz** - mniej cykli niż procesor wektorowy.
  - n-wymiarowy wektor i $M_{n\times n}$: **2n** (procesor wektorowy n^2)
- 2-wymiarowa macierz systoliczna; macierz:

$$\mathbf{W} = \begin{pmatrix} c & d \\ e & f \end{pmatrix},$$

którą mnożymy razy wektor $[a, b]$ (dokładnie jej transpozycję).

Hardware dla sieci neuronowych - sieci cyfrowe

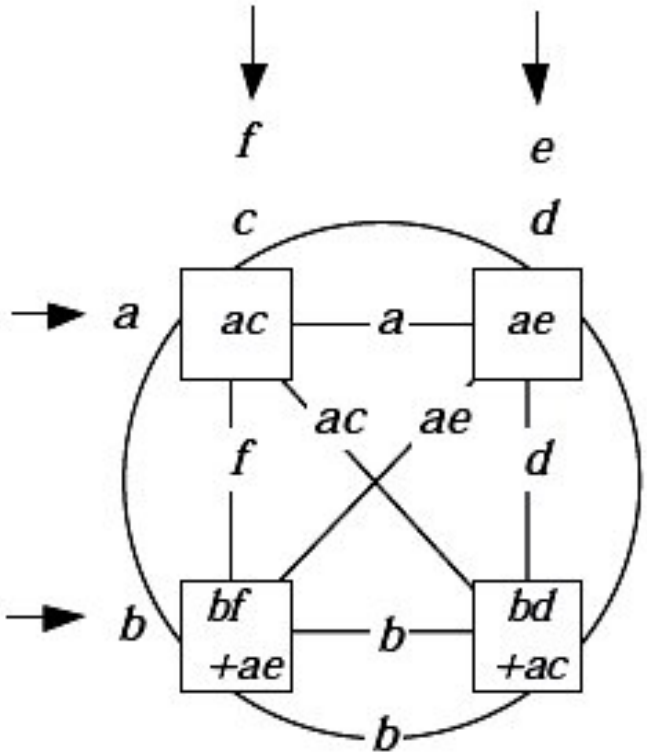
- Rzędy macierzy podawane są od góry.
- Wartości przekazywane na prawo. Wyniki działań ukośnie.



Planar systolic array for vector-matrix multiplication

Hardware dla sieci neuronowych - sieci cyfrowe

- Łącząc brzegi sieci do utworzenia torusa - redukcja liczby elementów z $n(2n-1)$ do n^2.
- **Architektura toroidalna** - często wybierana do neurokomputerów.



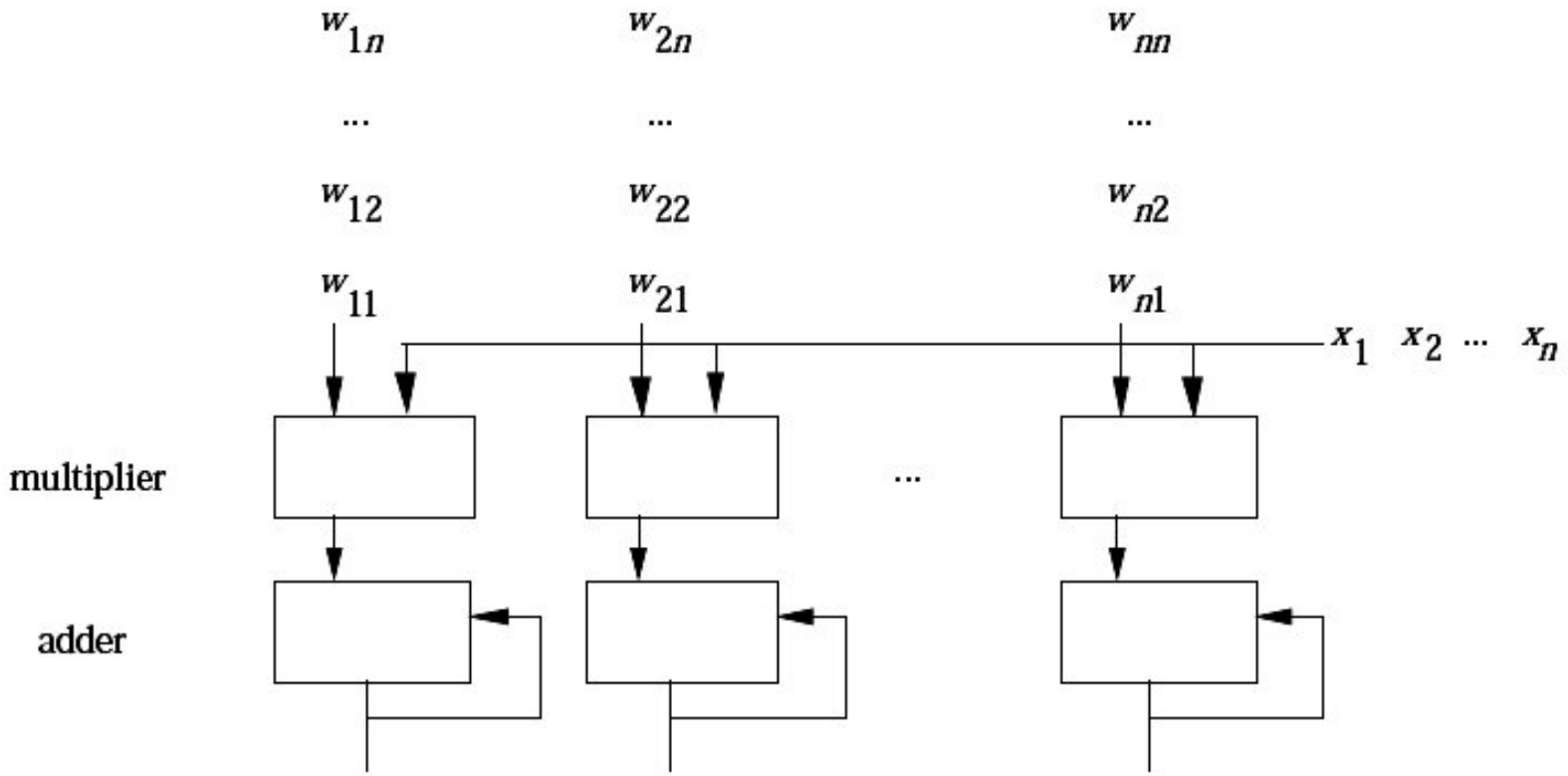
Systolic array with toroidal topology

Hardware dla sieci neuronowych - sieci cyfrowe

SPECJALNE ARCHITEKTURY.

- Doświadczenie w implementacji sieci na zwykłych komputerach $\Rightarrow$ rozwój specjanych projektów mikroprocesorów dla neurokomputerów.
- **Model SIMD** popularny - łatwo znacznie przyspieszyć mnożenie wektor-macierz kosztem niewielkiej inwestycji w hardware.
- Studiowano optymalne transformowanie modeli sieci neuronowych na maszyny SIMD.
- Implementacja SIMD - iloczynu macierz-wektor:
  - składowe wektora $x_1, ..., x_n$ - podawane sekwencyjnie do wszystkich procesorów
  - w każdym cyklu podawana jest też jedna kolumna macierzy $\mathbf{W}$
  - rezultat n mnożeń akumuluje się w jednostkach dodających; w tym samym czasie jednostki mnożące zaczynają obliczać kolejne iloczyny

Hardware dla sieci neuronowych - sieci cyfrowe



SIMD model for vector-matrix multiplication

Hardware dla sieci neuronowych - sieci cyfrowe

- W powyższym modelu - jednoczesna transmisja n argumentów z pamięci do jednostek arytmetycznych - potrzebna szeroka szyna danych.
- Torrent chip - 1995 - Berkeley - architektura SIMD - pierwszy procesor wektorowy na pojedynczym czipie (zaprojektowany dla sieci neuronowych oraz innych zadań z zakresu cyfrowego przetwarzania sygnałów).

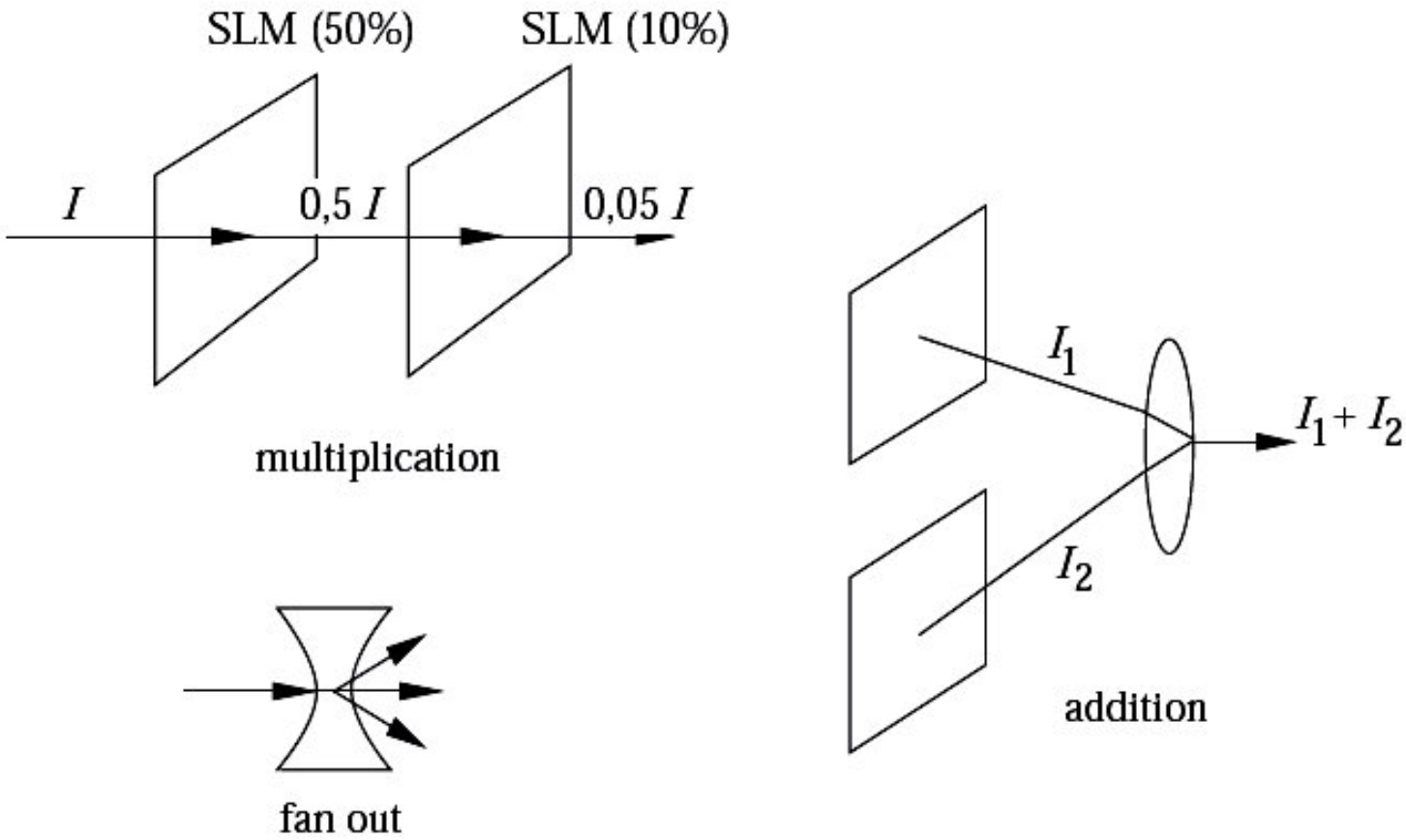
Hardware dla sieci neuronowych - sieci cyfrowe

- Ważna techniczna trudność sieci neuronowych - implementacja kanałów komunikacyjnych.
- **Komputery optyczne** - kanały komunikacyjne nie są częścią sprzętu a sygnały są transmitowane między komponentami jako światło.
- Promienie świtlne mogą się przecinać - nie wpływa to na przenoszone informacje.
- Potrzebna energia jest niewielka.
- Można osiągnąć czas przełączania - do $30 GHz$.

Hardware dla sieci neuronowych - sieci cyfrowe

- Łatwo zaimplementować podstawowe operacje - **przestrzenne modulatory światła** SLM (Spatial Light Modulators).
- SLM - maski optyczne kontolowane elektrodami; w zależności od napięcia - przepuszczają mniej światła - **możenie**.
  - kolejne mnożenia można wykonać "jednocześnie" - przepuszczenie światła przez kilka SLM-ów.
- **Dodawanie** - soczewki, pryzmaty.
- **Rozdzielenie sygnału** - kryształy lub odpowiednie soczewki.

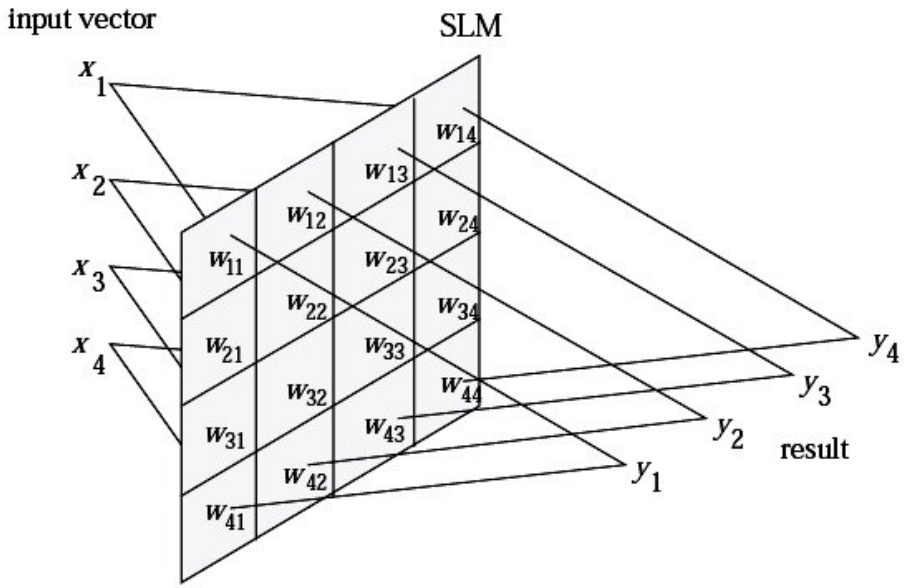
Hardware dla sieci neuronowych - sieci cyfrowe



Optical implementation of multiplication, addition, and signal splitting

Hardware dla sieci neuronowych - sieci cyfrowe

- Operacje algebry liniowej - łatwe do równoległej implementacji.
- Mnożenie macierz-wektor - podział SLM na $n \times n$ pól (niezależna regulacja).

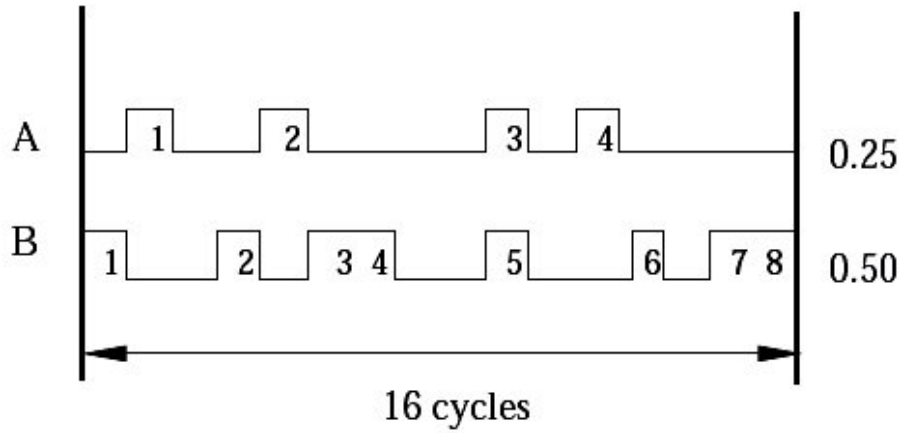


Matrix-vector multiplication with an SLM mask

- Wychodzące światło zbierane jest kolumna po kolumnie - rezultat = składowe iloczynu.
- Inne operacje - także łatwo - transformata Fouriera obrazu (przydatne w rozpoznawaniu obrazów).

Hardware dla sieci neuronowych - kodowanie pulsowe

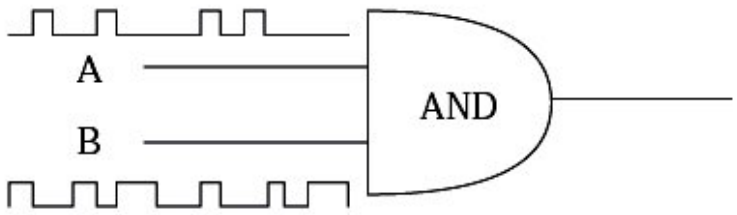
- Bliższe biologicznemu modelowi jest używanie **dyskretnych pulsów** (jak potencjały neuronów).
- Sterowanie częstotliwością sygnałów.
- Kodowanie pulsowe można zaimplementować w technologii analogowej oraz cyfrowej.
- Tomlinson - neuroczip.



Pulse coded representation of signals

Hardware dla sieci neuronowych - kodowanie pulsowe

- A=0.25, B=0.5
- Pulsy - reprezentowane 16-cyklowymi interwałami, które generator wytwarza losowo tak, że dla A puls pojawia się przez 1/4 czasu a B połowę czasu.
- Pulsy nie są skorelowane.
- Dekoder - rekonstuuje liczby zliczając pulsy w 16-cyklowym interwałach.
- **Iloczyn** A i B - bramka AND - wynik = łańcuch $0.25 \times 0.5 \times 16$ pulsów, czyli liczba 0.25×0.5
  - jest to rezultat statystyczny - liczba pulsów może różnić się od średniej
  - dokładność można zwiększyć wydłużając interwał np. z 24 do 256 (co odpowiada 8 bitom).



Multiplication of two pulse coded numbers

Hardware dla sieci neuronowych - kodowanie pulsowe

- **Sumowanie** (wynik ograniczony do przedziału $[0, 1]$) - bramka OR - otrzymujemy $C = 1 - (1 - A)(1 - B)$ (zero otrzymujemy A i B są jednocześnie zero).
- Dla 10 liczb $A_1, ..., A_{10}$ podobnie, jeżeli liczby są małe:

$$C = 1 - (1 - A_1) \cdots (1 - A_{10}) \approx 1 - \left(1 - \sum_{i=1}^{10} A_i\right) \approx \sum_{i=1}^{10} A_i.$$

- Większe wartości i wystarczająco szeroki interwał - przybliżenie funkcją podobną do **sigmoidu**:

$$C = 1 - \exp(-\sum_{i=1}^{10} A_i)$$

- Nie obejmuje to sygnałów ujemnych (nie pasuje do interpretacji probabilistycznej) - potrzebne do aktywacji jednostki - dodatkowy hardware.
- Wiele badań nad pulsową implementacją uczenia.

Hardware dla sieci neuronowych - historia

- Lata 50-te - pierwsze próby budowania specjalnego hardware.
- 1951 - Marvin Minsky - system symulujący adaptacyjne wagi potencjometrami.
- 1957 - Rosenblatt - perceptron - wagi jako opory (podejście Minsky'ego).
- Urządzenia Rosenblatta - pierwsze komercyjne neurokomputery.
- 1960 - Widrow, Hoff - pierwsze adaptacyjne systemy przetwarzające sygnały.
- Karl Steinbuch - zbudował pamięci z sieci oporów.
- Lata 70-te - niewielki postęp w Japonii cognitron i neokognitron.
- Lata 70-te - H.T. Kung (Carnegie Mellon) - macierze systoliczne.
- Warp - Princeton - pierwsza architektura systoliczna dla sieci neuronowych.
- 1980 - wiele prac w celu adaptacji konwencjonalnych systemów wieloprocesorowych do potrzeb sieci neuronowych.
- Czekamy na największy przełom - komputery optyczne.