Wykład 2

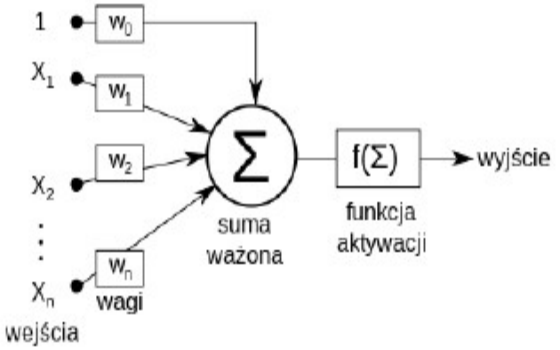
Model Neuronu McCulocha-Pittsa
Perceptron
Liniowe Sieci Neuronowe

wykład przygotowany na podstawie.

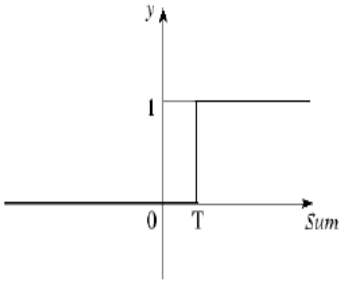
R. Tadeusiewicz, “Sieci Neuronowe”, Rozdz. 3. Akademicka Oficyna Wydawnicza RM, Warszawa 1993.
S. Osowski, „Sieci neuronowe do przetwarzania informacji”, Oficyna Wydawnicza PW, Warszawa 2000.
J. Żurada, M. arc, W. Jędruch, „Sztuczne sieci neuronowe”, Wydawnictwo naukowe PWN, Warszawa 1996.

Neuron McCullocha-Pittsa

Pierwsza formalna definicja sztucznego neutronu opartą na uproszczonym modelu biologicznym podali McCulloch i Pitts (1943).
Sygnały wejściowe mają wartość 1 lub 0, reguła aktywacji ma postać



$$y^{k+1} = 1 \text{ gdy } \Sigma w_i x_i^k \geq T$$
$$= 0 \text{ gdy } \Sigma w_i x_i^k < T$$

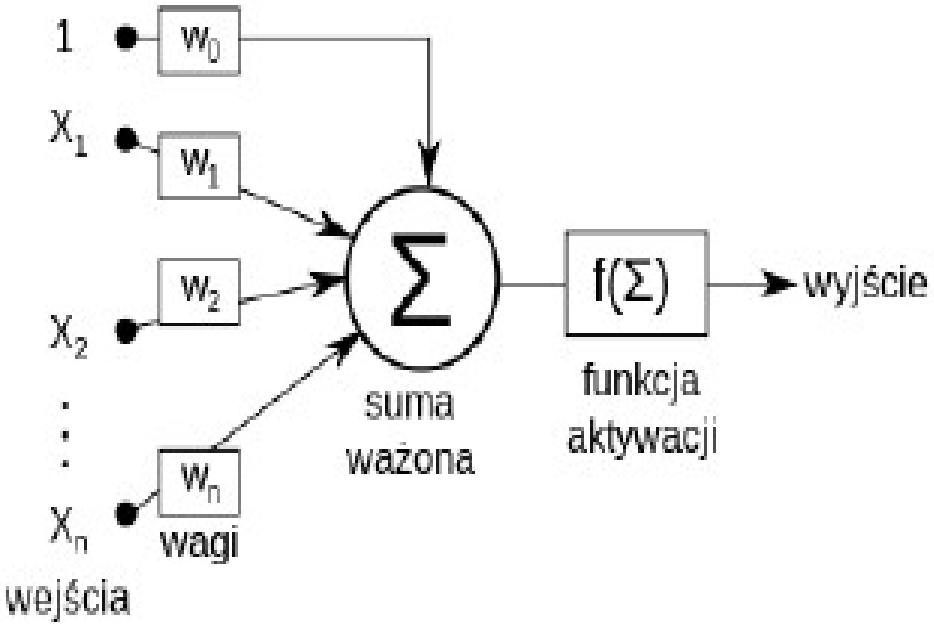


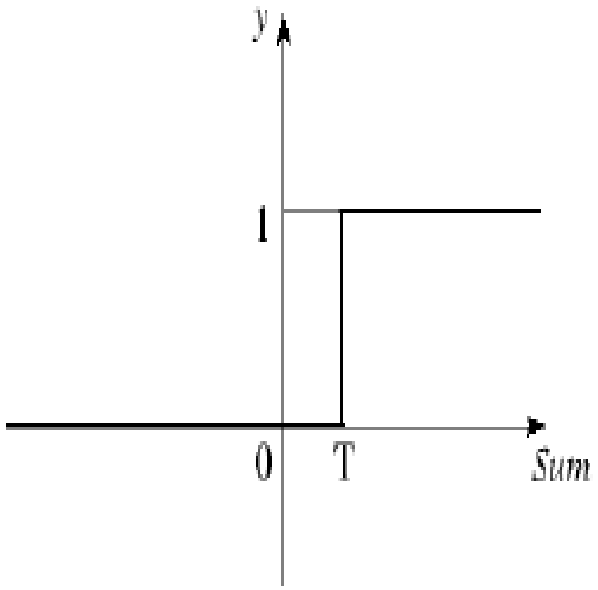
gdzie k = 0,1,2,… oznacza kolejne momenty czasu, w_i zaś jest multiplikatywną wagą przypisaną połączeniu.

Mimo swojej prostoty neuron ten wykazuje duże potencjalne możliwości. Przy odpowiednim doborze wag i progów można przy jego pomocy zrealizować funkcje logiczne

NOT, OR, NOR, NAND

Neuron McCullocha-Pittsa





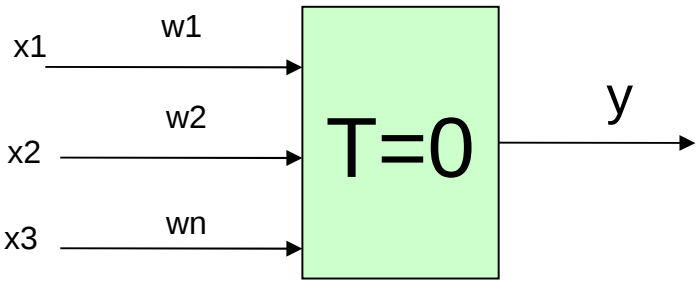
Neuron McCullocha-Pittsa

Model McCullocha-Pittsa cechuje elegancja i precyzja matematycznej definicji. Ale zawiera szereg drastycznych uproszczeń.

- Dopuszcza tylko binarne stany 0,1
- Zakłada dyskretność czasu i synchronizacje neuronów w sieci
- Wagi i progi są niezmienne

Perceptron

Prosty perceptron jest zwykłym modelem McCullocha-Pittsa o odpowiednio przyjętej strategii uczenia.



$$y^k(u^k) = 1 \text{ gdy } u^k \geq 0$$
$$= 0 \text{ gdy } u^k < 0$$

Gdzie $u^k = \Sigma\, w_i x_i^k$, $w_i = \pm 1$ dla $i=1, 2, ..n$

Najpopularniejszą metodą uczenia (przeprogramowania) jest tzw. *metoda perceptronu, zgodnie z którą dobór wag odbywa się w następującym cyklu:*

(0) *Przy zadanych wstępnie najczęściej losowo wartościach wag, podajemy wektor uczący* x^k *, referencyjny sygnał wyjściowy* d^k *oraz obliczamy sygnał wyjściowy* y^k*.*

(1) *W wyniku porównania dokonujemy uaktualnienia wektora wag:*

jeżeli $y^k = d^k$***, wagi pozostają niezmienione***

jeżeli $y^k = 0$ ***i*** $d^k = 1$, $w^k_i(t+1) = w^k_i(t) + x_i$

jeżeli $y^k = 1$ ***i*** $d^k = 0$, $w^k_i(t+1) = w^k_i(t) - x_i$

Perceptron

Po uaktualnieniu wag, podajemy nowy wektor x i skojarzoną z nim wartość d.

Krok (1) *powtarza się wielokrotnie, aż uzyska się minimalizację różnice między wszystkimi wartościami* y^k *i odpowiadającymi im wartościami żądanych* d^k.

Minimalizacja różnic pomiędzy odpowiedziami aktualnymi neuronu y^k i wartościami żądanymi d^k odpowiada minimalizacji określonej funkcji błędu (funkcji celu) **E**, definiowanej najczęściej jako

$$E = \Sigma_m (y^{k\,(m)} - d^{k\,(m)})^2$$

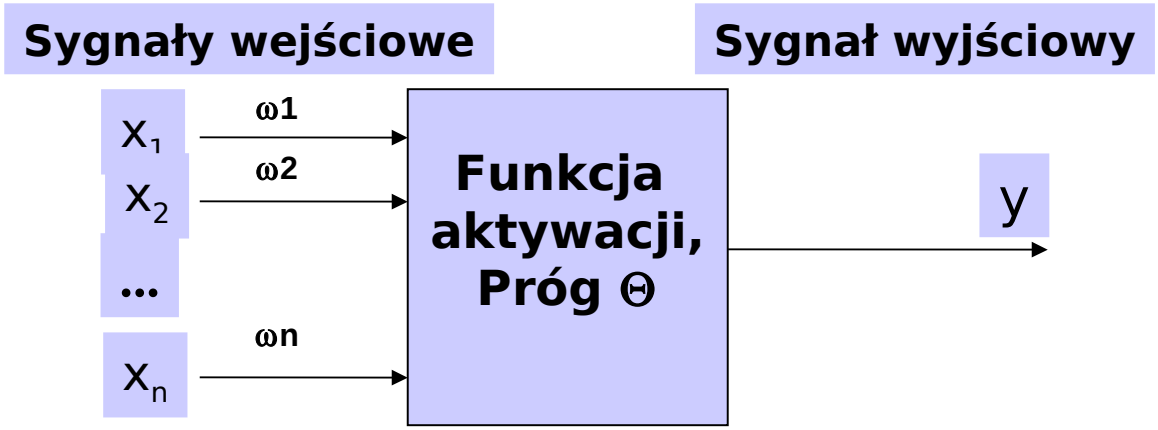
Gdzie sumowanie po indeksie m odpowiada sumowaniu po wszystkich wzorach uczących.

Perceptron

Minimalizacja ta w regule perceptronu odbywa się zgodnie z metodą *bezgradientowej* optymalizacji. Efektywność metody przy dużej liczbie wzorców jest stosunkowo niewielka, a liczba cykli uczących i czas uczenia wzrasta szybko, nie dając przy tym gwarancji uzyskania minimum funkcji celu. Usunąć te wady można jedynie przez przyjęcie funkcji aktywacji ciągłej, przy której **funkcja celu E** staje się również ciągła, co umożliwia wykorzystanie w procesie uczenia informacji o wartości grandientu.

Liniowy Sztuczny Neuron

Element opisany powyższym równaniem liniowym jest między innymi zdolny do ***rozpoznawania wejściowych sygnałów***.

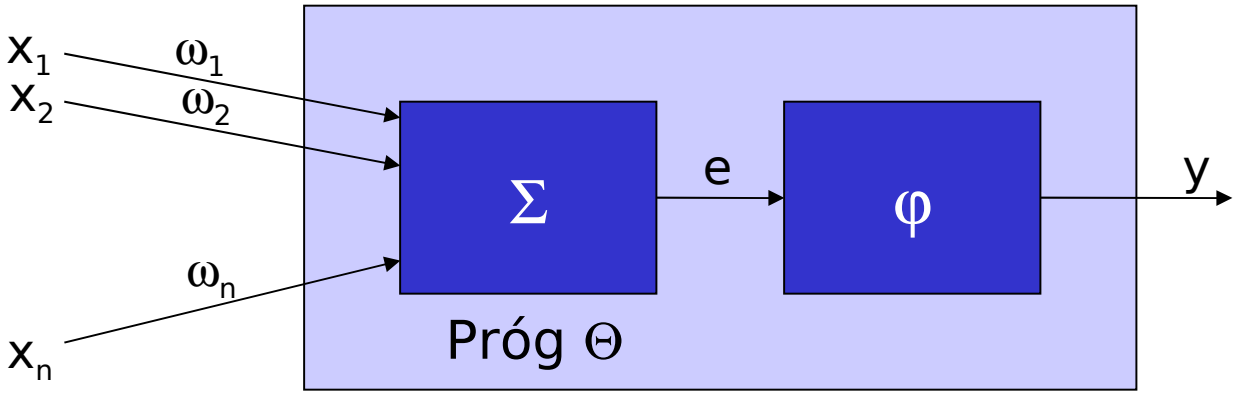


Współczynniki ω_i, nazywane ***wagami synaptycznymi***, podlegają modyfikacjom w trakcie ***procesu uczenia***, który stanowi jeden z zasadniczych wyróżników sieci neuronowych jako adaptacyjnych systemów przetwarzania informacji.

Liniowy Sztuczny Neuron

Niech zestaw sygnałów wejściowych neutronu stanowi wektor

$$x = \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{pmatrix}$$



Wektor ten można interpretować jako punkt w **n-**wymiarowej przestrzeni **X**, nazywanej przestrzenią wejść.
Wektor ten możemy też zapisać jako

$$X = < x_1, x_2, ..., x_n >^T$$

Zestaw **n** współczynników wagowych możemy również rozpatrywać jako wektor

$$W = < \omega_1, \omega_2, ..., \omega_n >^T$$

wyznaczający punkt w **n-**wymiarowej przestrzeni zwanej przestrzenią wag:

$y = W \otimes X$ (iloczyn skalarny)

Rozpoznawanie sygnału

Z własności iloczynu skalarnego, można powiedzieć, że wartość y będzie tym większa im bardziej położenie wektora **X** będzie przypominać położenie wektora wag.
Jeżeli założymy, że wektory **X, W** są znormalizowane do 1, to wówczas

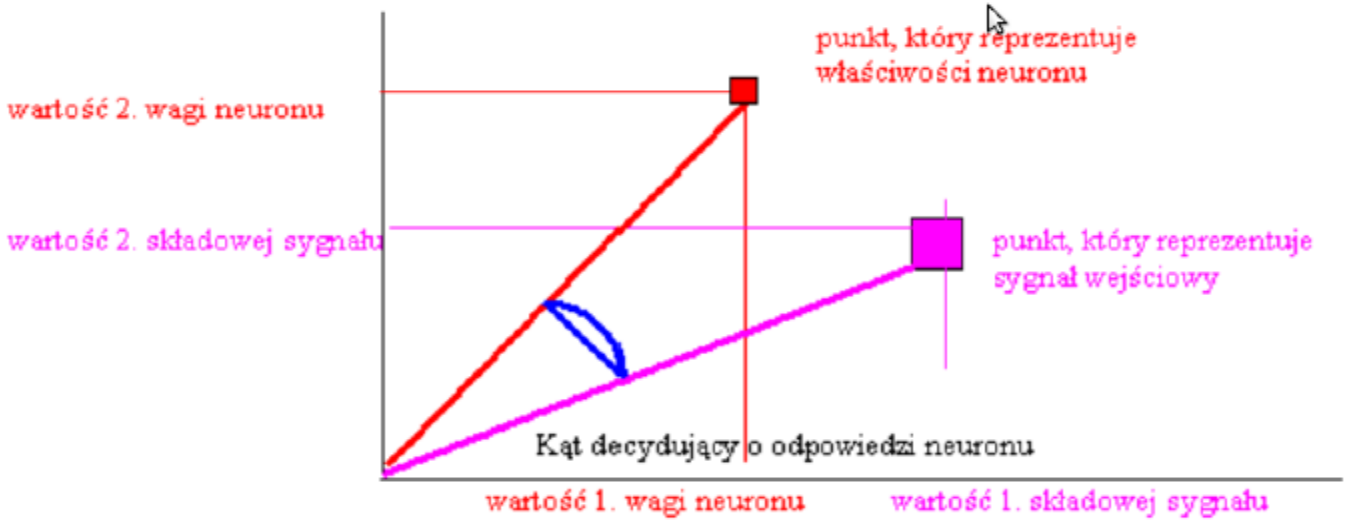
$$y = \cos(\phi)$$

gdzie ϕ jest katem pomiędzy wektorem **X** i **W**.

Jeżeli na wejście neuronu będziemy podawali różne sygnały, **X**, to sygnał na wyjściu neuronu będzie miał tym większą wartość, im bardziej podany sygnał **X** będzie podobny do sygnału wzorcowego, który neuron ***"pamięta"*** w postaci swojego zestawu wag **W**.

Rozpoznawanie sygnału

Pojedynczy neuron w typowych przypadkach realizuje (z matematycznego punktu widzenia) operacje iloczynu skalarnego wektora sygnałów wejściowych oraz wektora wag. W efekcie, odpowiedź neuronu zależy od wzajemnych stosunków geometrycznych pomiędzy wektorami sygnałów i wektorami wag.



Warstwa neuronów jako najprostsza sieć

Rozważmy warstwę neutronów, z których każdy ma ***ten sam*** zestaw sygnałów wejściowych $X = < x_1, x_2, ..., x_n >^T$, natomiast każdy ma ***swój własny*** wektor wag.

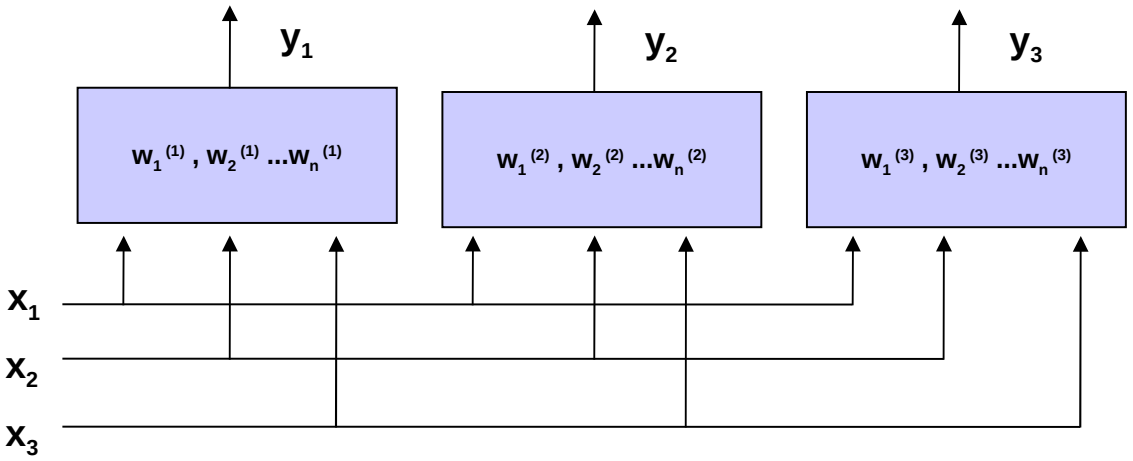
Ponumerujmy neurony i oznaczmy jako $W^{(m)} = < \omega_1, \omega_2, ..., \omega_n >^T$ wektor wag **m-**tego neuronu (m = 1, 2, ..., k).

Sygnał wyjściowy **m-**tego neuronu można wyznaczyć ze wzoru:

$$y_m = W^{(m)} \otimes X = \Sigma_{i=1}^{N} \omega_i^{(m)} x_i$$

Omawiana struktura stanowi najprostszy przykład sieci neuronowej. Działanie sieci polega na tym, że pojawienie się określonego wektora wejściowego **X** powoduje powstanie sygnałów wyjściowych y_m na wszystkich (m) neuronach wchodzących w skład rozważanej warstwy.

Warstwa neuronów jako najprostsza sieć



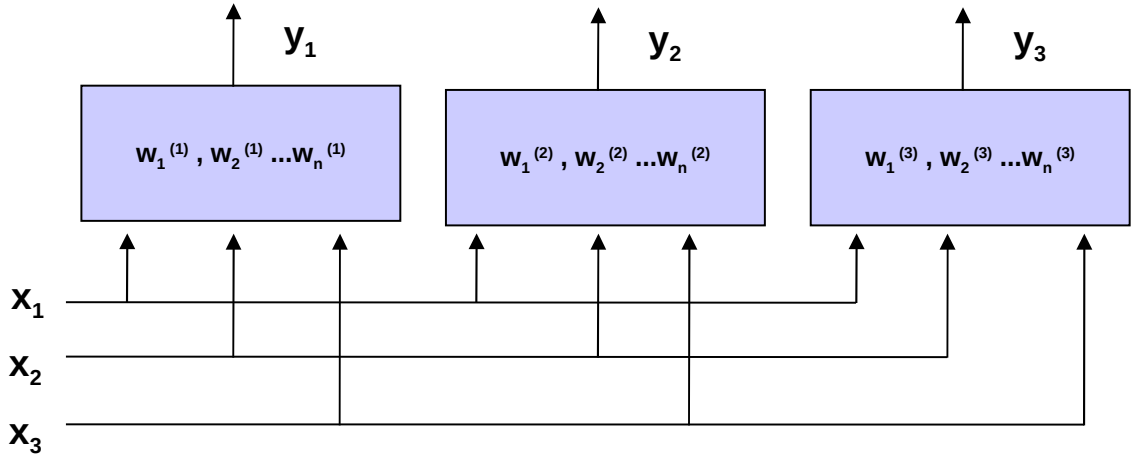
Oczekujemy maksymalnego sygnału w tym neuronie, którego wektor wag $W^{(m)}$ najbardziej przypomina **X**. Sieć tego typu może ***rozpoznawać*** k różnych klas obiektów, gdyż każdy neuron zapamiętuje jeden wzorcowy obiekt na którego pojawienie się jest **"*uczulony*".** Wzorce różnych klas zawarte są w poszczególnych neuronach w postaci ich wektorów wag.
Stosując notacje wektorową możemy zapisać

$$Y = < y_1, y_2, ..., y_k >^T$$

Warstwa neuronów jako najprostsza sieć

Możemy wprowadzić macierz $\mathbf{W_k}$ o wymiarach $\mathbf{k \times n}$, utworzoną w taki sposób, że jej kolejnymi wierszami są (transponowane) kolejne wektory $\mathbf{W^{(m)}}$ (dla m = 1,2,...,k).
Macierz $\mathbf{W^{(m)}}$ ma więc następującą budowę:

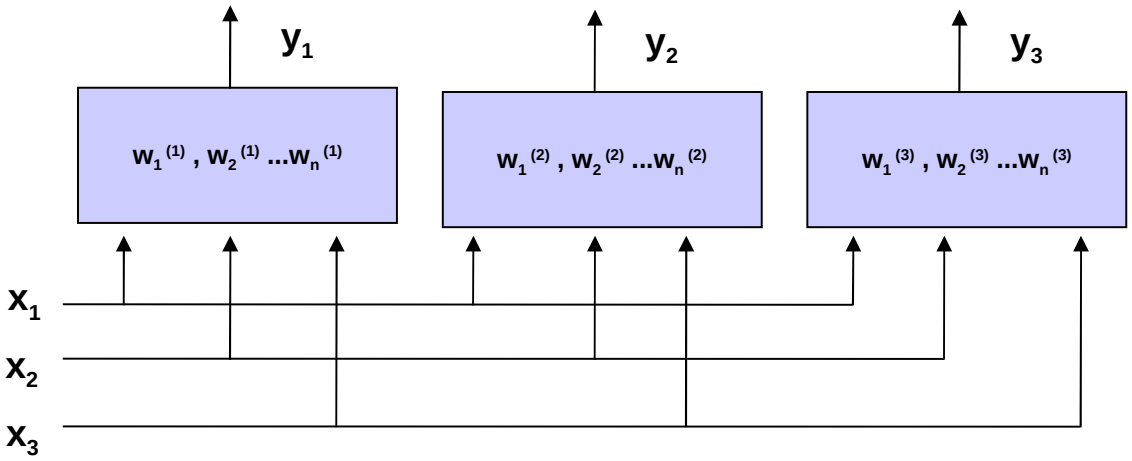
$$\begin{pmatrix} \omega^{(1)}_1 & \omega^{(1)}_2 & \ldots\ldots & \omega^{(1)}_n \\ \omega^{(2)}_1 & \omega^{(2)}_2 & \ldots\ldots & \omega^{(2)}_n \\ \ldots & & & \\ \omega^{(k)}_1 & \omega^{(k)}_2 & \ldots\ldots & \omega^{(k)}_n \end{pmatrix}$$



Warstwa neuronów jako najprostsza sieć

Wykorzystując macierz W_k można zapisać funkcję realizowaną przez całą sieć w formie

$$Y = W_k X$$



Macierz W_k zadaje ***odwzorowanie liniowe*** sygnału $X \in R^n$ w sygnał $Y \in R^k$. Odwzorowanie to może być w zasadzie dowolne (np. transformacja Fouriera). Przekształcenie sygnału **X** w sygnał **Y** można też interpretować jako ***filtrację***, w związku z tym o sieci neuronowej dokonującej takiego przekształcenia możemy mówić jako o ***filtrze***.

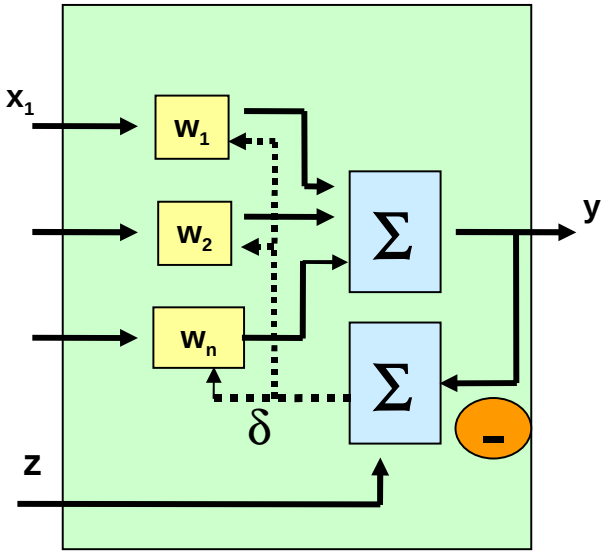
Uczenie pojedynczego neuronu

O zachowaniu pojedynczego neuronu decydował

wektor wag **W**,

a o działaniu sieci –

macierz wag $\mathbf{W_k}$.



Istnieje możliwość zastąpienia jednorazowego aktu ***zaprogramowania*** sieci, iteracyjnym, wieloetapowym procesem jej ***uczenia***.
Aby zapewnić możliwość uczenia, trzeba wprowadzony już model neuronu uzupełnić o dwa dodatkowe elementy: ***procesor zmiany wag*** oraz ***detektor błędu***.

Tak zmodyfikowany neuron nazywany bywa **ADALINE** (ADAptive LINear Element), i wykazuje zastanawiająco bogate możliwości w zakresie dostosowywania swojego działania do wymagań wynikających z postawionego zadania.

ADALINE, MADALINE

Neurony liniowe **ADALINE** i zbudowane z nich sieci liniowe, znane w literaturze pod **MADALINE** są najprostsze w budowie, a równocześnie są w wielu wypadkach wysoce użyteczne.

Neurony

Elementy z których buduje się sieci, charakteryzują się występowaniem wielu wejść i jednego wyjścia.

Sygnały wejściowe x_i (i=1, 2, ...n) oraz sygnał wyjściowy y mogą przyjmować wartości z pewnego ograniczonego przedziału.

Z dokładnością do prostej funkcji skalującej można przyjąć, że

$$x_i \in [-1, 1]$$

dla każdego i, a także

$$y \in [-1, 1]$$

Zależność

$$y = f(x_1, x_2, ..., x_n)$$

w najprostszym przypadku może być rozważana jako liniowa $y = \sum_{i=1}^{n} \omega_i x_i$

Uczenie pojedynczego neuronu

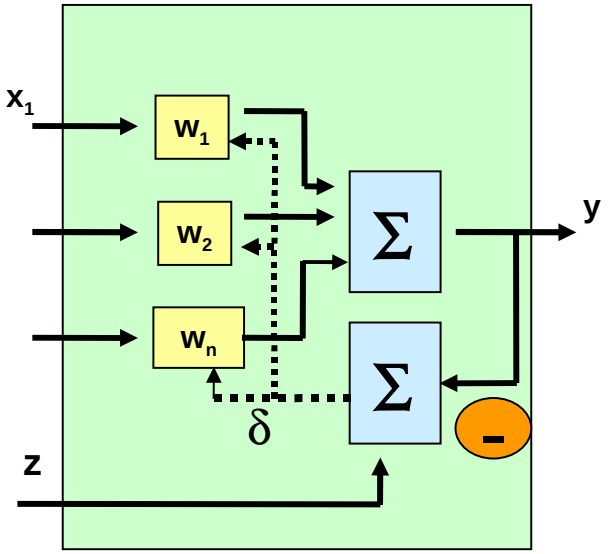
Załóżmy, że zadanie stawiane **ADALINE** polega na tym, by sygnał wyjściowy **y** był związany z sygnałami wejściowymi **X** pewną zależnością funkcyjną

$$y = f(X)$$

Funkcja **f** nie musi być zadana w sposób jawny; wystarczy, że dla każdego konkretnego wektora wejściowego potrafimy wskazać konkretną wartość

$$z = f(X)$$

stanowiącą ***referencyjną wartość*** odnośnie sygnału wyjściowego y.



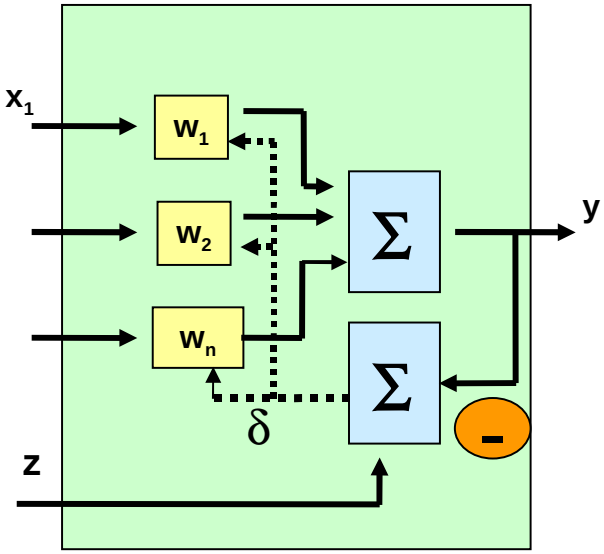
Uczenie pojedynczego neuronu

Zasada działania ADALINE przy rozwiązaniu tego zadania oparta jest na podstawowym algorytmie uczenia wprowadzonym przez Widrowa i Hoffa, zwanym **regułą DELTA.**
Wraz z każdym wektorem wejściowym **X**, do neuronu podany jest również sygnał **z**, czyli żądana (wymagana) odpowiedź neuronu na sygnał **X**.
Neuron odpowiada na sygnał **X** sygnałem

$$y = W \otimes X,$$

przy czym jeśli neuron nie jest nauczony, sygnał ten jest inny niż wymagany, czyli $y \neq z$.
Wewnątrz neuronu **ADALINE** istnieje blok składający się z sumatora i inwertora oceniający wielkość błędu

$$\delta = z - y$$

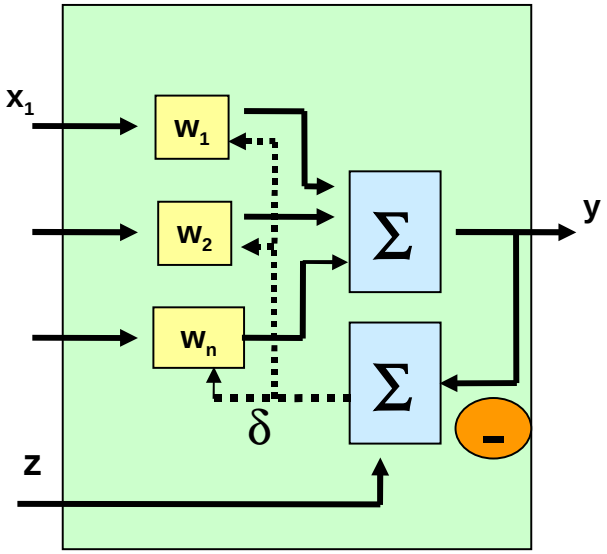


Uczenie pojedynczego neuronu

Na podstawie ***sygnału błędu*** δ oraz wektora wejściowego **X** możliwe jest takie skorygowanie wektora wag **W**, by neuron lepiej realizował zadaną funkcję **y = f(X)**. Nowy wektor wag obliczany jest ze wzoru

$$W' = W + \eta\, \delta\, X$$

gdzie η jest współczynnikiem liczbowym, decydującym o szybkości uczenia.
Korekta **W** jest tym silniejsza im większy został odnotowany błąd.



Matematyczne aspekty procesu uczenia sieci

Wprowadźmy pojecie ***ciągu uczącego***. Ciąg ten ma następującą budowę:

$$U = < < X^{(1)}, z^{(1)} >, < X^{(2)}, z^{(2)} >, ... , < X^{(N)}, z^{(N)} > >$$

czyli składa się z par postaci $< X^{(j)}, z^{(j)} >$ zawierających wektor $X^{(j)}$ podany w **j-**tym kroku procesu uczenia i informacje o wymaganej odpowiedzi neutronu $z^{(j)}$ w tym kroku.

Regułę uczenia
możemy napisać następująco:

$$W^{(j+1)} = W^{(j)} + \eta^{(j)} \delta^{(j)} X^{(j)}$$

gdzie

$$\delta^{(j)} = z^{(j)} - y^{(j)}$$

oraz

$$y^{(j)} = W^{(j)} \otimes X^{(j)}$$

Reguła ta daje się łatwo stosować pod warunkiem wprowadzenia początkowego wektora wag $W^{(1)}$; zwykle zakłada się że wektor ten ma składowe wybrane losowo.

Matematyczne aspekty procesu uczenia sieci

Celem procesu uczenia
jest uzyskanie zgodności odpowiedzi neutronu $y^{(j)}$ z wymaganymi wartościami $z^{(j)}$, co daje się sprowadzić do problemu minimalizacji **funkcji kryterialnej**

$$Q = \frac{1}{2} \sum_{j=1}^{N} (z^{(j)} - y^{(j)})^2$$

powyższy wzór możemy też zapisać jako $Q = \sum_{j=1}^{N} Q^{(j)}$,
gdzie

$$Q^{(j)} = \frac{1}{2} (z^{(j)} - y^{(j)})^2$$

Matematyczne aspekty procesu uczenia sieci

Ponieważ **Q = Q(W)**, zatem poszukiwanie minimum może być dokonywane **metodą gradientową**. Skupiając uwagę na **i**-tej składowej wektora **W**, możemy więc zapisać:

$$\omega'_i - \omega_i = \Delta \omega_i = -\eta \frac{\partial Q}{\partial \omega_i}$$

Wzór ten można interpretować w sposób następujący: poprawka $\Delta \omega_i$ jakiej powinna podlegać **i**-ta składową wektora **W**, musi być proporcjonalna do **i**-tej składowej gradientu funkcji **Q**.
Znak **–** w omawianym wzorze wynika z faktu, że gradient **Q** wskazuje kierunek najszybszego wzrastania tej funkcji, podczas gdy w omawianej metodzie zależy nam na tym, by zmieniać **W** w kierunku najszybszego malenia błędu popełnianego przez sieć, czyli w kierunku najszybszego malenia funkcji **Q**. Współczynnik proporcjonalności $\eta^{(i)}$ określa wielkość kroku $\Delta \omega_i$ i może być w zasadzie wybierany dowolnie.

Matematyczne aspekty procesu uczenia sieci

Przy wyborze współczynnika $\eta^{(i)}$ mogą pojawić się pewne subtelności:
a) Z teorii aproksymacji stochastycznej można wyprowadzić wniosek, że $\eta^{(i)}$ powinny spełniać warunki

$$\sum_{j=1}^{\infty} \eta^{(i)} = \infty , \qquad \sum_{j=1}^{\infty} (\eta^{(i)})^2 < \infty$$

W najprostszym przykładzie warunki te spełnia ciąg

$$\eta^{(i)} = \eta^{(0)} / j$$

ale szybkie malenie $\eta^{(i)}$ może ograniczać skuteczność procesu uczenia.
b) w celu ograniczenia malenia $\eta^{(i)}$ oraz ograniczenia rośnięcia modułu wektora **W**, często proponuje się

$$\eta^{(i)} = \lambda \, / \, || X^{(j)} ||^2 \ ,$$

gdzie λ jest pewną ustaloną stałą (zwykle $0.1 < \lambda < 1$).

Matematyczne aspekty procesu uczenia sieci

Gradientowa minimalizacja funkcji błędu (funkcji kryterialnej)

Rozpisując wzór gradientowego uczenia dla kroku **(j)**, otrzymujemy:

$$\omega_i^{(j+1)} - \omega_i^{(j)} = \Delta\omega_i^{(j)} = -\frac{\partial Q^{(j)}}{\partial w_i}\eta$$

Uwzględniając fakt, że **Q** zależne jest od **y**, a dopiero **y** jest funkcją wektora wag **W**, możemy zapisać wzór, odpowiadający pochodnej funkcji złożonej

$$\frac{\partial Q^{(j)}}{\partial \omega_i} = \frac{\partial Q^{(j)}}{\partial y^{(j)}}\frac{\partial y^{(j)}}{\partial \omega_i}$$

Na podstawie zależności $Q^{(j)} = \frac{1}{2}\left(z^{(j)} - y^{(j)}\right)^2$ można ustalić że,

$$\frac{\partial Q^{(j)}}{\partial \omega_i} = -\left(z^{(j)} - y^{(j)}\right) = -\delta^{(j)}$$

Matematyczne aspekty procesu uczenia sieci

Natomiast liniowa funkcja wiążąca sygnał wyjściowy $y^{(j)}$ z wektorem wag $W^{(j)}$ powoduje, że

$$\frac{\partial y^{(j)}}{\partial \omega_i} = X_i^{(j)}$$

Zbierając razem wszystkie powyższe rozważania, końcowa formuła uczenia neuronu typu **ADALINE** wygląda następująco:

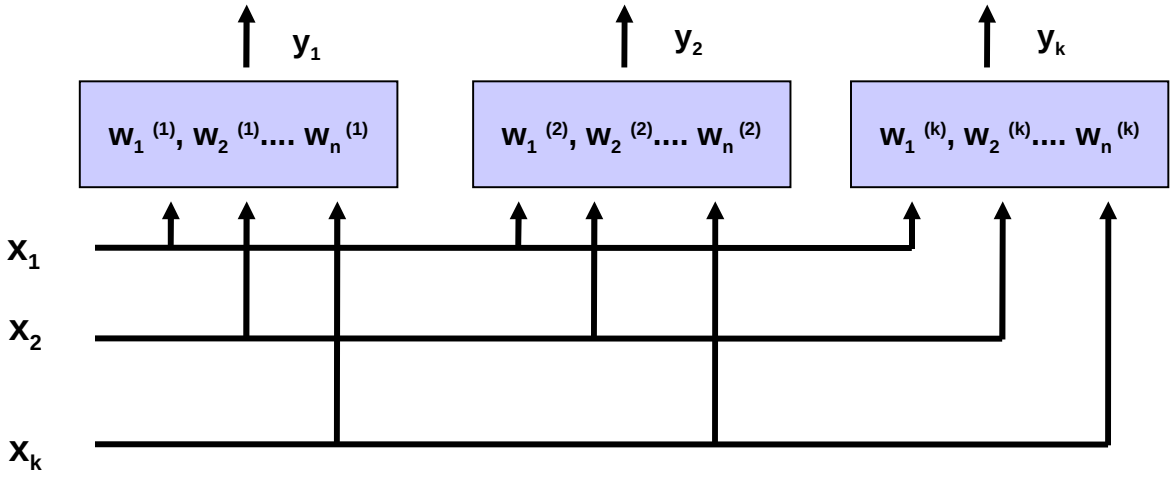
$$\Delta \omega_{\mathbf{i}}^{(\mathbf{j})} = \eta \; \delta^{(\mathbf{j})} \mathbf{X}_{\mathbf{i}}^{(\mathbf{j})}$$

co potwierdza jej poprawność.

Proces uczenia jest zbieżny i pozwala wyznaczyć potrzebny wektor wag **W***, zapewniający dokładną realizację przez neuron wymaganej funkcji **y = f(X)**, jeśli **f** jest funkcją liniową, lub gwarantujący optymalną (w sensie minimum średniokwadratowego) aproksymację tej funkcji, jeżeli jest ona nieliniowa.

Uczenie sieci elementów liniowych

W sposób analogiczny do opisanego algorytmu uczenia pojedynczego neuronu można uczyć także całą ***sieć*** elementów liniowych.



Uczenie sieci elementów liniowych

Obiektem podlegającym uczeniu jest w tym wypadku macierz W_k, a ciąg uczący ma postać:

$$\mathbf{U} = (\mathbf{X}^{(1)}, \mathbf{Z}^{(1)}), (\mathbf{X}^{(2)}, \mathbf{Z}^{(2)}), ..., (\mathbf{X}^{(N)}, \mathbf{Z}^{(N)})$$

gdzie $Z^{(j)}$ są k-elementowymi wektorami oznaczającymi wymagane zestawy odpowiedzi sieci na wymuszenia danych odpowiednimi wektorami $X^{(j)}$.

Sieć taka w literaturze nazywana jest **MADALINE** (*Many* ADALINE s). Uczenie sieci **MADALINE** odbywa się w sposób zupełnie analogiczny do uczenia neuronu **ADALINE**, z tą różnicą że formuła uczenia ma postać macierzową

$$\mathbf{W_k^{(j+1)}} = \mathbf{W_k^{(j)}} + \eta(\mathbf{Z^{(j)}} - \mathbf{Y^{(j)}})(\mathbf{X^{(j)}})^{\mathbf{T}}$$

Uczenie sieci elementów liniowych: filtry

Przypominając interpretację funkcjonowania sieci o **n-**wejściach i **k-**wyjściach jako **filtra przetwarzającego** sygnały **X** na odpowiadające im (zgodnie z określonym odwzorowaniem) sygnały **Y** – można interpretować proces uczenia sieci jako **adaptację filtra do określonych potrzeb**. Wkraczamy w ten sposób w krąg zagadnień ***filtracji adaptacyjnej*** oraz problemu poszukiwania ***optymalnego filtra*** *(z punktu widzenia określonych kryteriów),* ***samoczynnie dostosowującego się*** *do potrzeb.*

Sieci neuronowe (zwłaszcza typu MADALINE) były i są chętnie stosowane jako filtry adaptacyjne (na przykład do eliminacji efektu "echa" w liniach telefonicznych).

Filtry tworzone w wyniku uczenia sieci neuronowej mogą być wykorzystywane do typowych zadań (dolno lub górnoprzepustowa filtracja sygnału, eliminacja zakłóceń, polepszanie stosunku sygnału do szumu, wydobywanie określonych cech sygnału, analiza widmowa, itp.), ale mogą też mieć zupełnie nowe zastosowania.

Uczenie sieci elementów liniowych: filtry

Filtry tego typu mogą służyć do odtwarzania kompletnego sygnału na podstawie jego fragmentu. Takie zadanie nazywa się ***"pamięcią adresowaną kontekstowo"*** lub ***"pamięcią asocjacyjną "*.**

Pamięć taka pozwala odtworzyć cały zestaw zapamiętanych informacji w przypadku przedstawienia informacji niekompletnej lub niedokładnej. Tego rodzaju urządzenie, działając w powiązaniu z ***systemem ekspertowym***, może doskonale odpowiadać na pytania użytkowników. Zadaniem sieci jest w tym wypadku uzupełnienie wiadomości podanych przez użytkownika w taki sposób, by system ekspertowy mógł
(w oparciu o bazę wiedzy) efektywnie szukać rozwiązania.

Innym zastosowaniem omawianych tu filtrów może być tzw. ***"filtr nowości"***, czyli tej informacji która uległa nagłej zmianie (ochrona przed włamaniem, automatyka przemysłowa).

Uczenie “z” lub “bez nauczyciela”

Opisany poprzednio ***schemat uczenia*** sieci **MADALINE** opierał się na założeniu że istnieje ***zewnętrzny arbiter*** (“nauczyciel”), który podaje poprawne odpowiedzi **Z**, w wyniku czego korekta macierzy wag W_k następuje w sposób sterowany i zdeterminowany przez cel, jakim jest minimalizacja błędu **Q**.

Taki schemat nazywamy ***“supervised learning”*** albo ***“delta rule”.*** Jest on często niewygodny, ponadto mało “wiarygodny biologicznie”, gdyż wielu czynności mózg uczy się bez świadomego i celowego instruktażu.

Uczenie “z” lub “bez nauczyciela”

Często stosowaną metodą jest technika uczenia “***bez nauczyciela***”, zwaną ***“unsupervised learning”*** lub ***“hebbian learning”.*** Zasada tego uczenia polega na tym, że waga $\omega_i^{(m)}$, **i-**tego wejścia **m-**tego neuronu wzrasta podczas prezentacji **j-**tego wektora wejściowego $X^{(j)}$ proporcjonalnie do iloczynu **i-**tej składowej sygnału wejściowego tego $x_i^{(j)}$ docierającego do rozważanej synapsy i sygnału wyjściowego rozważanego neuronu.

$$\omega_i^{(m)(j+1)} = \omega_i^{(m)(j)} + \eta\, x_i^{(j)} y_m^{(j)}$$

przy czym oczywiście

$$y_m^{(j)} = \Sigma_{i=1}^{m} \omega_i^{(m)(j)} x_i^{(j)}$$

Wzmocnieniu w sieci ulegają te wagi, które są aktywne (duże $x_i^{(j)}$) w sytuacji gdy “ich” neuron jest pobudzony (duże $y_m^{(j)}$). Tego typu sieć jest zatem ***“autoasocjacyjna”:*** jeśli pewien wzór pobudzeń **X** jest sygnalizowany przez pewne **m-**te wyjście sieci, to w miarę upływu czasu ta sygnalizacja staje się coraz bardziej wyraźna.

Uczenie “z” lub “bez nauczyciela”

Sieć uczy się rozpoznawać bodźce oraz grupować w pewne kategorie (clusters), gdyż neuron wytrenowany do rozpoznawania pewnego sygnału **X**, będzie zdolny do rozpoznawania także tych sygnałów które ***są podobne*** do wzorcowego.

Zdolność do uogólniania zdobytego już doświadczenia jest jedną z najważniejszych cech sieci neuronowej.

Regułę uczenia pochodzącą od praw Hebba nazywa się także niekiedy ***uczeniem korelacyjnym*** (“correlation learning”), ponieważ zmierza ona do takiego dopasowania wag aby uzyskać najlepszą korelację pomiędzy sygnałami wejściowymi, a zapamiętanym w formie wartości wag ***“wzorcem sygnału”,*** na który określony neuron ma reagować. Eksponuje ona fakt ***“uśredniania”*** wejściowych sygnałów przez sieć uczoną wg. reguły Hebba.

Uczenie “z” lub “bez nauczyciela”

W wyniku takiego uczenia sieć jest zdolna do **rozpoznawania sygnałów** których wcześniej nigdy jej nie pokazywano: **zniekształcone, niekompletne, zaburzone** przez szum.

Sieć uczona wg. metody Hebba na ogół uzyskuje dobre wyniki i **samoczynnie grupuje sygnały wejściowe w “kategorie”**. Efekt ten nie jest jednak nigdy pewny, gdyż istotnie zależy od stanu początkowego sieci.

Często też **wiele neuronów uczy się rozpoznawać te same kategorie**, czyli na ogół ilość neuronów w sieci musi być większa niż liczba oczekiwanych kategorii.

Warianty metod uczenia i samouczenia

Przyrostowe samouczenie (Hebba):

Proces samouczenia i samoorganizacji można uczynić bardziej efektywnym poprzez zastosowanie tak zwanego ***przyrostowego samouczenia*** ("*differential hebbian learning*") polegającego na uzależnieniu procesu zmiany wag od przyrostów sygnałów wejściowych na danej synapsie i wyjściowych na danym neuronie:

$$\omega_{\mathbf{i}}^{(\mathbf{m})(\mathbf{j}+\mathbf{1})} = \omega_{\mathbf{i}}^{(\mathbf{m})(\mathbf{j})} + \eta\,[(\mathbf{x}_{\mathbf{i}}^{(\mathbf{j})} - \mathbf{x}_{\mathbf{i}}^{(\mathbf{j}-\mathbf{1})})(\mathbf{y}_{\mathbf{m}}^{(\mathbf{j})} - \mathbf{y}_{\mathbf{m}}^{(\mathbf{j}-\mathbf{1})})]$$

W niektórych wypadkach ta przyrostowa strategia daje znacznie lepsze rezultaty niż "czysty" algorytm Hebba.

Warianty metod uczenia i samouczenia

Metoda "gwiazdy wejść" (Grossberga):

W metodzie tej, "instar training", wybiera się pewien neuron i narzuca mu się taką strategie uczenia, by zapamiętał i potrafił rozpoznać aktualnie wprowadzany sygnał **X**. Inne neurony sieci są w tym czasie bezczynne.

$$\omega_i^{(m)(j+1)} = \omega_i^{(m)(j)} + \eta^{(j)}\left(x_i^{(j)} - \omega_i^{(m)(j)}\right)$$

Z praktyki stosowania tej metody wynikają pewne zalecenia odnośnie wyboru wartości $\eta^{(j)}$, które powinny się zmieniać zgodnie z empiryczna regułą:

$$\eta^{(j)} = 0,1 - \lambda j$$

przy czym współczynnik λ należy wybrać na tyle mały, by w ciągu całego uczenia zachodził warunek $\eta^{(j)} > 0$.

Warianty metod uczenia i samouczenia

Metoda gwiazdy wyjść (Grossberga):

Na zasadzie analogii można wprowadzić koncepcje “outgoing stars”. W koncepcji tej rozważa się wagi wszystkich neuronów całej warstwy, jednak wybiera się wyłącznie wagi łączące te neurony z ***pewnym ustalonym wejściem***.
W sieciach wielowarstwowych wejście to pochodzi od pewnego ustalonego neuronu wcześniejszej warstwy i to właśnie ten neuron staje się ***“gwiazdą wyjść”*** (outstar).

$$\omega_{\mathbf{i}}^{(\mathbf{m})(\mathbf{j}+\mathbf{1})} = \omega_{\mathbf{i}}^{(\mathbf{m})(\mathbf{j})} + \eta^{(\mathbf{j})}(\mathbf{y}_{\mathbf{m}}^{(\mathbf{j})} - \omega_{\mathbf{i}}^{(\mathbf{m})(\mathbf{j})})$$

W powyższym wzorze **i** jest ustalone, natomiast **m** jest zmienne i przebiega wszelkie możliwe wartości (m = 1,2,.....,k).
Reguła zmieniania $\eta^{(j)}$ jest dana wzorem

$$\eta^{(\mathbf{j})} = \mathbf{1} - \lambda\mathbf{j}$$

Warianty metod uczenia i samouczenia

Metoda ***"instar"*** stosowana jest w przypadku, kiedy trzeba sieć nauczyć rozpoznawania określonego sygnału **X**, natomiast metoda ***"outstar"*** znajduje zastosowanie przy uczeniu sieci wytwarzania określonego wzorca zachowań **Y** w odpowiedzi na określony sygnał inicjujący x_i.

Reguły uczenia podane przez Hebba i Grossberga były przez wielu badaczy wzbogacane i modyfikowane.

Warianty metod uczenia i samouczenia

Dyskryminacja wejściowych i wyjściowych sygnałów.

Polega na dzieleniu sygnałów $x_i^{(j)}$ i $y_m^{(j)}$ na **"aktywne"** i **"nieaktywne"**.

W tym celu wprowadza się pewna ustaloną lub zależną od j **wartość progową ε** i wprowadza się nowe zmienne

$$\hat{x}_i^{(j)} = \begin{cases} 1 & \text{gdy } x_i^{(j)} > \varepsilon \\ 0 & \text{w przeciwnym wypadku} \end{cases}$$

oraz

$$\hat{y}_i^{(j)} = \begin{cases} 1 & \text{gdy } y_i^{(j)} > \varepsilon \\ 0 & \text{w przeciwnym wypadku} \end{cases}$$

Przy tych oznaczeniach nową technikę uczenia wg. **zmodyfikowanego algorytmu Hebba** zapisać można w postaci:

$$\omega_{\mathbf{i}}^{(\mathbf{m})(\mathbf{j}+\mathbf{1})} = \omega_{\mathbf{i}}^{(\mathbf{m})(\mathbf{j})} + \eta^{(\mathbf{j})}\left(\hat{\mathbf{x}}_{\mathbf{i}}^{(\mathbf{j})}\hat{\mathbf{y}}_{\mathbf{m}}^{(\mathbf{j})}\right)$$

Czy zmiana wagi jest dokonana decyduje logiczny warunek podany powyżej.

Warianty metod uczenia i samouczenia

Reguła Hebb/Anti-Hebb

Podany wzór pozwala jedynie na zwiększanie wartości wag, co może prowadzić do osiągania przez nie bardzo dużych wartości.
Proponuje się niekiedy uogólniony algorytm, w którym przy aktywnym wyjściu $\hat{y}_i^{(j)}$ synapsy obsługujące aktywne wejścia $\hat{x}_i^{(j)}$ uzyskują wzmocnienie, a synapsy obsługujące nieaktywne wyjścia są osłabiane (zakłada się że $\hat{y}_i^{(j)} \in \{0,1\}$)

$$\omega_{\mathbf{i}}^{(\mathbf{m})(\mathbf{j}+\mathbf{1})} = \omega_{\mathbf{i}}^{(\mathbf{m})(\mathbf{j})} + \eta^{(\mathbf{j})}\left(\mathbf{2}\hat{\mathbf{y}}_{\mathbf{m}}^{(\mathbf{j})}-\mathbf{1}\right)$$

Dalsze poszerzenie rozważanej metody uczenia daje **wzór Hopfielda**, w podobny sposób traktujący zarówno $\hat{x}_i^{(j)}$ jak i $\hat{y}_i^{(j)}$

$$\omega_{\mathbf{i}}^{(\mathbf{m})(\mathbf{j}+\mathbf{1})} = \omega_{\mathbf{i}}^{(\mathbf{m})(\mathbf{j})} + \eta^{(\mathbf{j})}\left(\mathbf{2}\hat{\mathbf{x}}_{\mathbf{i}}^{(\mathbf{j})}-\mathbf{1}\right)\left(\mathbf{2}\hat{\mathbf{y}}_{\mathbf{m}}^{(\mathbf{j})}-\mathbf{1}\right)$$

Wszystkie współczynniki wagowe podlegają przy nim bardzo intensywnemu treningowi, niezależnie od tego, czy dotyczą wejść aktywnych i niezależnie od tego, czy odpowiednie wyjścia są aktywne, natomiast oczywiście kierunek zmian aktywności wag zależy od aktywności wejść i wyjść.

Uczenie z rywalizacją i sieci Kohonena

Uczenie z rywalizacja **(competitive learning)**

wprowadził Kohonen przy tworzeniu sieci neuronowych uczących się realizacji ***dowolnych odwzorowań*** **X ⇒ Y**.
Zasada uczenia z rywalizacja jest formalnie identyczna z regułą "instar"

$$\omega_{\mathbf{i}}^{(\mathbf{m}*)(\mathbf{j}+\mathbf{1})} = \omega_{\mathbf{i}}^{(\mathbf{m}*)(\mathbf{j})} + \eta^{(\mathbf{j})}\left(\mathbf{x}_{\mathbf{i}}^{(\mathbf{j})} - \omega_{\mathbf{i}}^{(\mathbf{m}*)(\mathbf{j})}\right)$$

z dwoma dość istotnymi uzupełnieniami.

⇒Wektor wejściowy X jest przed procesem uczenia normalizowany tak, aby ||X|| = 1.
⇒ Index poddawanego treningowi neuronu **m*** nie jest przypadkowy czy arbitralnie wybierany, jest to bowiem ten (i tylko ten) **neuron którego sygnał wyjściowy $y_{m*}^{(j)}$** jest największy. Przy każdorazowym podaniu sygnału wejściowego $X^{(j)}$ neurony rywalizują ze sobą i wygrywa ten, który uzyskał największy sygnał wyjściowy $y_{m*}^{(j)}$.
Tylko ten **zwycięski neuron podlega uczeniu**, którego efektem jest jeszcze lepsze dopasowanie wag $W^{(m*)(j+1)}$ do rozpoznawania obiektów podobnych do $X^{(j)}$.

Uczenie z rywalizacją i sieci Kohonena

Reguła uczenia Kohonena bywa często wzbogacana o dodatkowy element związany z topologią uczącej się sieci. Neurony w sieci są ***uporządkowane***, można więc wprowadzić pojęcie ***sąsiedztwa***. Uogólniona metoda samoorganizującej się sieci Kohonena polega na tym, że uczeniu podlega nie tylko neuron **m*** wygrywający w konkurencji z innymi neuronami sieci, ale także neurony które z nim sąsiadują.
Formalnie regule można zapisać wzorem:

$$\omega_i^{(m*)(j+1)} = \omega_i^{(m*)(j)} + \eta^{(j)} h(m, m*)(x_i^{(j)} - \omega_i^{(m)(j)})$$

formuła uczenia może być zapisana w formie:

$$\omega_i^{(m*)(j+1)} = \omega_i^{(m*)(j)} + \eta^{(j)} x_i^{(j)} \ (2\, y_m^{(j)} - 1)$$

Uczenie z rywalizacją i sieci Kohonena

Funkcjonowanie powyższego wzoru w istotny sposób oparto na fakcie, że $y_m^{(j)} \in \{0,1\}$.

Wzór ten nazywamy *regułą Hebb/Anti-Hebb*.

Funkcje **h(m,m*)** można definiować na wiele różnych sposobów, na przykład:

$$h(m,m^*) = \begin{cases} 1 & \text{dla } m=m^* \\ 0.5 & \text{dla } |m-m^*| =1 \\ 0 & \text{dla } |m-m^*| > 1 \end{cases}$$

$$h(m,m^*) = 1/\rho(m,m^*)$$

$$h(m,m^*) = \exp\left(- [\rho(m,m^*)]^2 \right)$$

Uczenie z forsowaniem

Omawiane dotychczas techniki **uczenia "bez nauczyciela"** mają bardzo interesującą odmianę polegającą na wykorzystaniu przytoczonych powyżej metod wówczas kiedy wektor wymaganych wartości sygnałów wyjściowych sieci $Z^{(j)}$ jest znany .

Wszystkie wymienione powyżej metody uczenia dadzą się łatwo zastosować poprzez zamianę **y** przez stosowne **z**. Takie uczenie ma charakter ***"forsowania"*** poprawnych rozwiązań bez względu na to co robi sieć.

Uczenie z forsowaniem

Wyróżnić możemy następujące metody:

- ***metoda autoasocjacji***:

$$\omega_i^{(m)(j+1)} = \omega_i^{(m)(j)} + \eta^{(j)} z_m^{(j)}$$

- ***metoda przyrostowej autoasocjacji***:

$$\omega_i^{(m)(j+1)} = \omega_i^{(m)(j)} + \eta^{(j)} [(x_i^{(j)} - x_i^{(j-1)})(z_m^{(j)} - z_m^{(j-1)})]$$

- ***metoda zbliżania wektora wag do wektora odpowiedzi***:

$$\omega_i^{(m)(j+1)} = \omega_i^{(m)(j)} + \eta^{(j)} (z_m^{(j)} - \omega_m^{(j)})$$

Wybór jednej z różnych możliwości podyktowany musi być ocena ich przydatności w konkretnym zadaniu. Brak jest tutaj konkretnej teorii, konieczne są eksperymenty i poszukiwania oparte na badaniach empirycznych.

Uczenie z forsowaniem

Sieć może służyć jako pamięć

Wystartujmy od równania autoasocjacji

$$\omega_{\mathbf{i}}^{(\mathbf{m})(\mathbf{j}+\mathbf{1})} = \omega_{\mathbf{i}}^{(\mathbf{m})(\mathbf{j})} + \eta^{(\mathbf{j})} \mathbf{X}_{\mathbf{i}}^{(\mathbf{j})} \mathbf{Z}_{\mathbf{m}}^{(\mathbf{j})}$$

i przepiszmy go do postaci macierzowej

$$\mathbf{W}_{\mathbf{k}}^{(\mathbf{j}+\mathbf{1})} = \mathbf{W}_{\mathbf{k}}^{(\mathbf{j})} + \eta^{(\mathbf{j})} \mathbf{Z}_{\mathbf{m}}^{(\mathbf{j})} [\mathbf{X}_{\mathbf{i}}^{(\mathbf{j})}]^{\mathbf{T}}$$

Efekt uczenia można zapisać w postaci sumarycznego wzoru

$$\mathbf{W}_{\mathbf{k}} = \Sigma_{\mathbf{j}=\mathbf{1}}^{\mathbf{N}} \eta \, \mathbf{Z}^{(\mathbf{j})} [\mathbf{X}^{(\mathbf{j})}]^{\mathbf{T}} + \mathbf{W}_{\mathbf{k}}^{(\mathbf{1})}$$

Załóżmy, że $W_k^{(1)} = 0$ oraz przyjmijmy że wszystkie wektory wejściowe w ciągu uczącym są ortonormalne. Wówczas sieć nauczy się wiernie odtwarzać wymagane sygnały wyjściowe dla wszystkich rozważanych sygnałów wejściowych.

Uczenie z forsowaniem

Sieć ma zdolność uogólniania:

Sieć jest zdolna do uogólniania sygnałów wejściowych:

$$\mathbf{X}^{(\mathbf{j})} = \mathbf{X} + \zeta^{(\mathbf{j})}$$

gdzie $\zeta^{(j)}$ reprezentuje "szum" zniekształcający sygnały wejściowe w każdym kolejnym przykładzie.

Taki model odpowiada np. procedurze odczytywania ręcznie pisanych liter.

Jeżeli dla każdego wczytywanego $\mathbf{X}^{(j)}$ będziemy podawać ten sam wektor wyjściowy $\mathbf{Z}$ (np. uczmy sieć rozpoznawania litery A), to w wyniku uczenia macierz wag zostanie zbudowana następująco:

$$\mathbf{W_k} = \Sigma_{\mathbf{j=1}}^{\mathbf{N}} \eta \mathbf{Z}[\mathbf{X}^{(\mathbf{j})} + \zeta^{(\mathbf{j})}]^{\mathbf{T}} = \eta \mathbf{Z}(\mathbf{N}[\mathbf{X}]^{\mathbf{T}} + \Sigma_{\mathbf{j=1}}^{\mathbf{N}}[\zeta^{(\mathbf{j})}]^{\mathbf{T}})$$

Uczenie z forsowaniem

W macierzy wag manifestować się będzie głownie ***"idealny" wzorzec X***, podczas gdy

$$\Sigma_{j=1}^{N}\left[\zeta^{(j)}\right]^{T}$$

na ogół ma niewielką wartość, ponieważ poszczególne składniki $\zeta^{(j)}$ mogą się wzajemnie kompensować. Oznacza to, że ***sieć ma zdolność uśredniania*** sygnałów wejściowych i może ***"sama odkryć"*** nowy wzorzec **X**.

Oba omawiane efekty są niezależne od siebie i mogą być ***"nałożone",*** co w wyniku daje sieć która ma zdolność zapamiętania różnych reakcji **Z** na różne sygnały wejściowe X wydobywane z serii przypadkowo zniekształconych obserwacji.
Organiczeniem może być **"pojemność pamięci"** związana z liczbą neuronów w sieci:

$$N_{max} \sim k/(2\log k)$$

Przyspieszanie procesu uczenia

W bardziej złożonych i rozbudowanych sieciach istotną rolę odgrywa sprawność procesu uczenia. Jak optymalizujemy potrzebne CPU?

- Odpowiedni dobór wartości $\eta^{(j)}$.
- Wprowadzanie do wzoru na korektę wektora wag dodatkowego składnika uwzględniającego **"bezwładność" procesu** zmiany wag w postaci tzw. ***momentu***.

$$\mathbf{W}_{\mathbf{k}}^{(\mathbf{j}+\mathbf{1})} = \mathbf{W}_{\mathbf{k}}^{(\mathbf{j})} + \eta_{\mathbf{1}}(\mathbf{Z}^{(\mathbf{j})} - \mathbf{Y}^{(\mathbf{j})})[\mathbf{X}^{(\mathbf{j})}]^{\mathbf{T}} + \eta_{\mathbf{2}}\mathbf{M}^{(\mathbf{j})}$$

gdzie moment $\mathbf{M}^{(j)}$ wyliczane jest ze wzoru:

$$\mathbf{M}^{(\mathbf{j})} = \mathbf{W}_{\mathbf{k}}^{(\mathbf{j})} - \mathbf{W}_{\mathbf{k}}^{(\mathbf{j}-\mathbf{1})}$$

Przyspieszanie procesu uczenia

➢**Ograniczanie procesu uczenia** wyłącznie do *dużych poprawek*. Oznacza to, że reguła uczenia ma dodatkowy parametr η_3 i działa następująco:

$$\omega_i^{(m)(j+1)} = \begin{cases} \omega_i^{(m)(j)} + \eta_1 \, x_i^{(j)} \, (z_m^{(j)} - y_m^{(j)}) & \text{gdy } (z_m^{(j)} - y_m^{(j)}) \geq \eta_3 \\ \omega_i^{(m)(j)} & \text{gdy } (z_m^{(j)} - y_m^{(j)}) < \eta_3 \end{cases}$$

➢Wybór współczynników η_1, η_2, η_3 jest sprawą doświadczenia/intuicji osoby która stosuje specyficzny algorytm.

➢ Wygładzanie wykładnicze:

$$W_k^{(j+1)} = W_k^{(j)} + \eta_1 \{ (1 - \eta_1) (Z^{(j)} - Y^{(j)}) [X^{(j)}]^T + \eta_2 (W_k^{(j)} - W_k^{(j-1)}) \}$$

➢ Istotnym problemem jest również odpowiednia randomizacja zbioru uczącego, kolejne elementy $X^{(j)}$ powinny być losowo wybierane (a nie cyklicznie).

Uwagi końcowe

Sieci **MADALINE** były pierwszymi efektywnie zastosowanymi sieciami neuronowymi i wciąż pozostają używanym narzędziem przy budowaniu systemów rozpoznających, filtrów, pamięci asocjacyjnych. Jednakże:

Możliwości systemów budowanych z elementów liniowych są ograniczone.

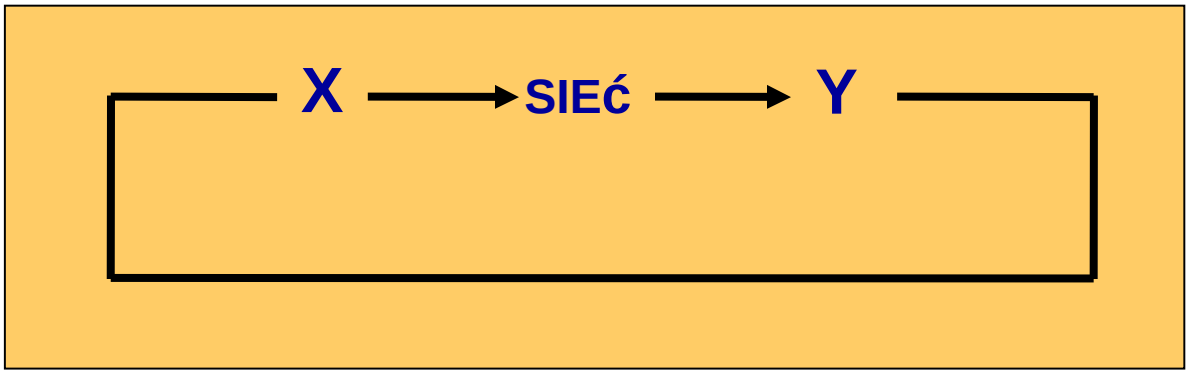
- sieć może realizować tylko odwzorowania liniowe
- klasa dostępnych odwzorowań nie zależy od tego czy mamy odczynienia z siecią jednowarstwową czy wielowarstwową. Budowa wielowarstwowej sieci liniowej nie ma sensu, ponieważ można ją zastąpić efektywną siecią jednowarstwową.

$$Y = W_n X \qquad (W_n = W_k W_l)$$

Uwagi końcowe

W ramach sieci liniowych, rozważa się dwa schematy funkcjonowania takiej sieci

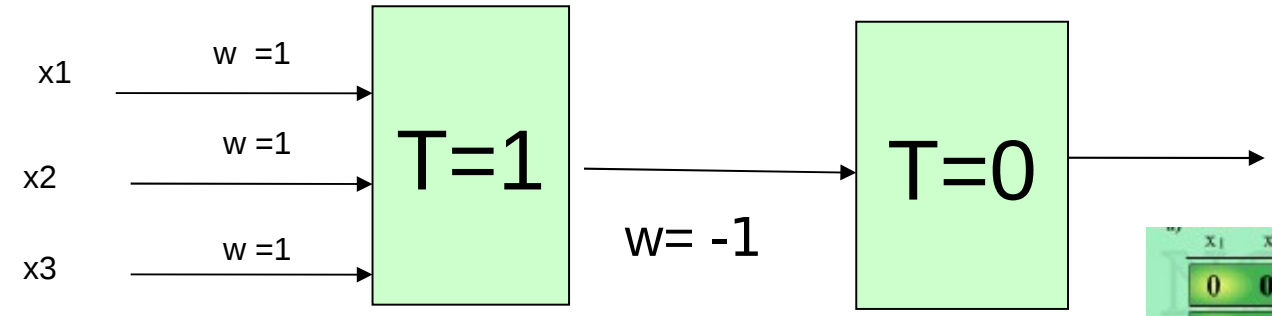
- jednokierunkowy przepływ sygnałów **(feedforward)** sygnały **X** pojawiające się na wejściu są niezależne od sygnałów na wyjściu **Y**
- sieci ze sprzężeniem zwrotnym **(feedback)** w których sygnały wyjściowe **Y** są pośrednio lub bezpośrednio podawane na wejście **X**. Takie sieci nazywane są ***autoasocjacyjnymi*** i oznaczają się bogatymi własnościami dynamicznymi.



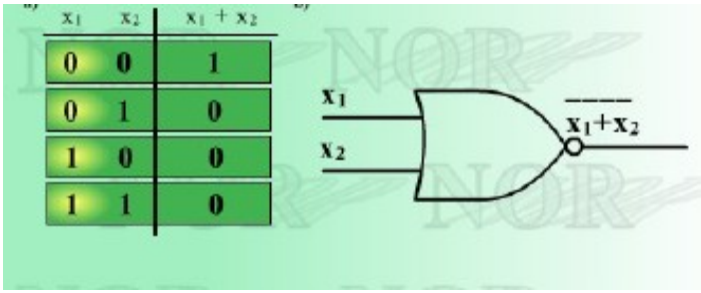
Zestaw pytań do testu

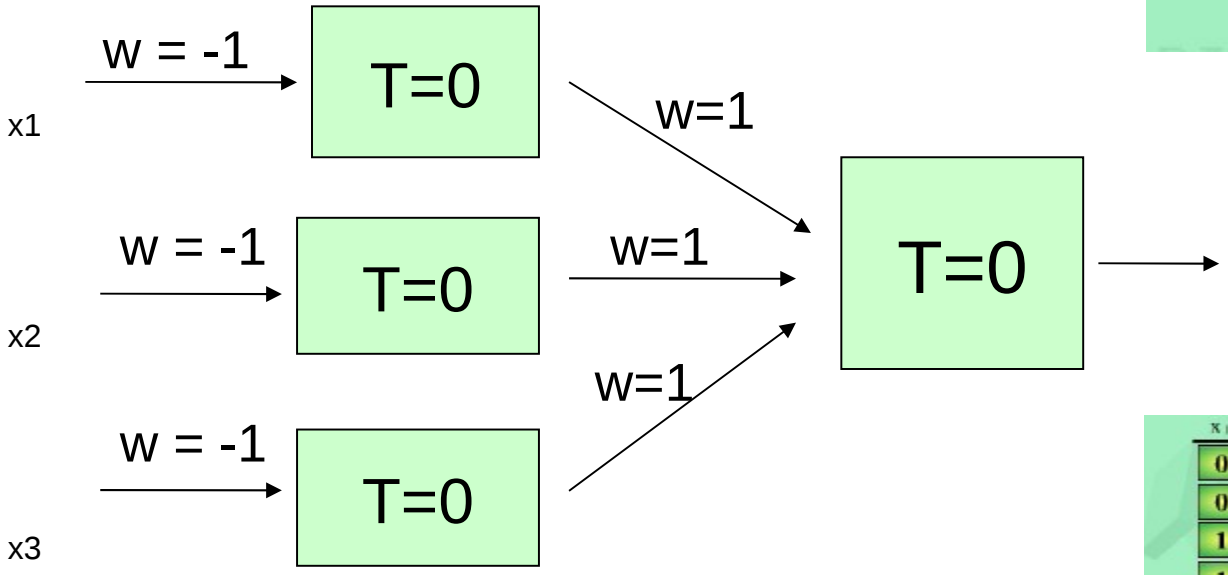
1. Co to jest neuron McCullocha-Pittsa i jakie układy logiczne można przy jego pomocy zrealizować?
2. Jakie elementy musi zawierać neutron ADALINE?
3. Napisz wzór na sygał wyjściowy neuronu liniowego.
4. Co to jest reguła delta?
5. Napisz końcowy wzór na zmianę wag neuronu ADALINE.
6. Co to jest sieć MADALINE i czy może być stosowana jako filter?
7. Dlaczego w sieci liniowej nie wprowadzamy warstw?
8. Na czym polega idea samouczenia Hebba?
9. Napisz wzór na zmianie wag w sieci Kohonena uczonej metodą Hebb/Anti-Hebb.
10. Napisz wzór na zmianę wag przy uczeniu z forsowaniem.

Elementarne układy logiczne



NOR





NAND