

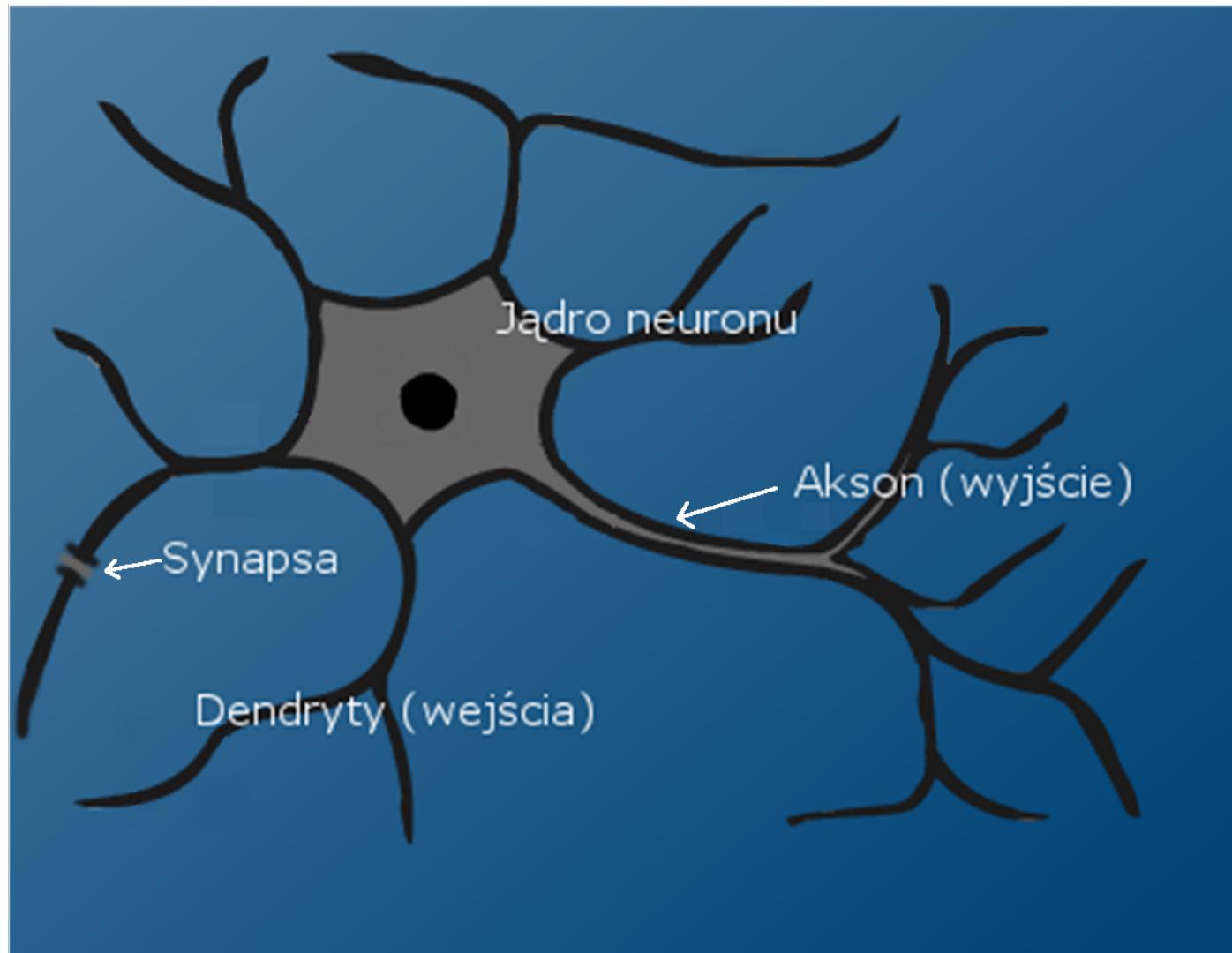
Sieci neuronowe

Sztuczne sieci neuronowe

- Sieci neuronowe to matematyczne i obliczeniowe modele, których budowa została zainspirowana strukturą i funkcjonowaniem biologicznych sieci neuronowych. Zbudowane są z połączonych grup tzw. sztucznych neuronów. W większości przypadków są układami adaptacyjnymi, które potrafią zmieniać swoją strukturę i parametry w oparciu o zewnętrzne lub wewnętrzne informacje, które przepływają przez sieć w fazie uczenia. Podstawowym elementem obliczeniowym takiej sieci jest **sztuczny neuron**.

- Sieci neuronowe są trenowane i wykorzystywane do wykonywanie złożonych zadań w takich dziedzinach jak:
 - rozpoznawanie obrazów (wzorców)
 - rozpoznawania mowy
 - identyfikacja
 - klasyfikacja
 - systemy kontroli
 - systemy ekspertowe (wspomaganie decyzji)

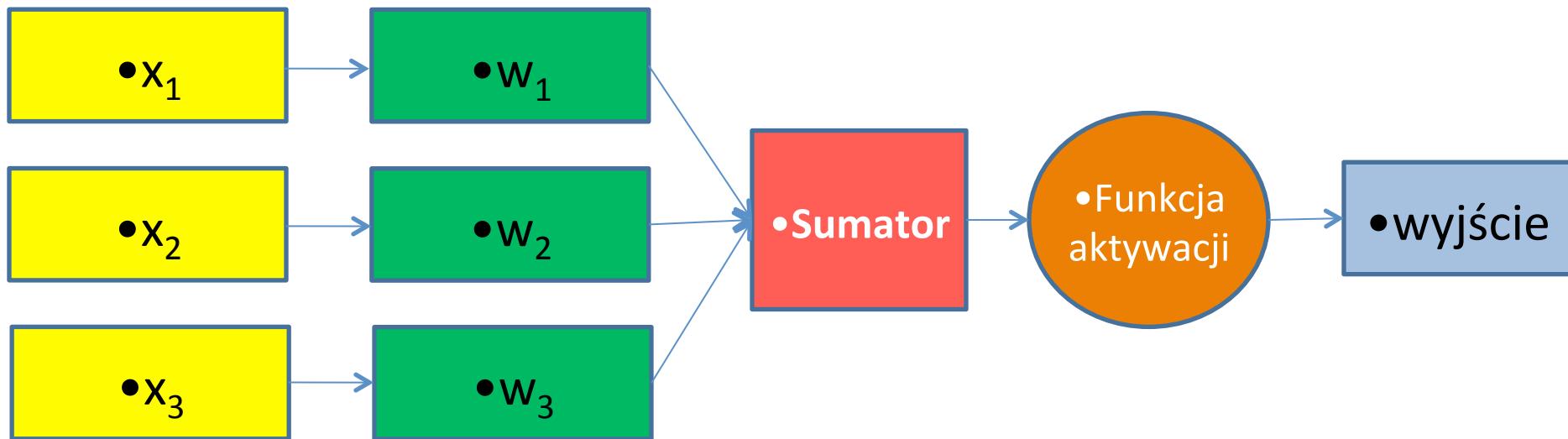
Uproszczony schemat neuronu biologicznego



•Ogólny schemat neuronu

$$y = f(\cdot)$$

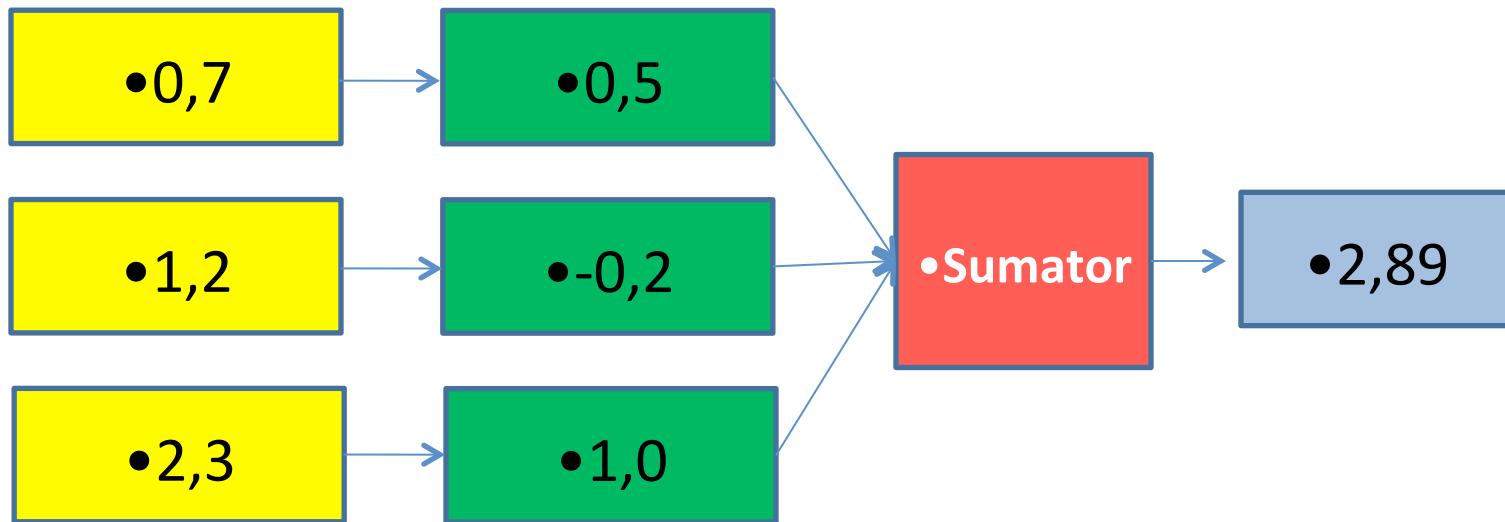
•dowolna (na ogół nieliniowa) *funkcja aktywacji*



$$\text{wzbudzenie} = \sum_{i=1}^n w_i x_i$$

$$y = f(\text{wzbudzenie})$$

Przykład



- Jest to przykład neuronu liniowego.
- Funkcja aktywacji $f(s)=s$, więc została pominięta

- Zauważmy, że w przypadku gdy funkcja aktywacji ma postać

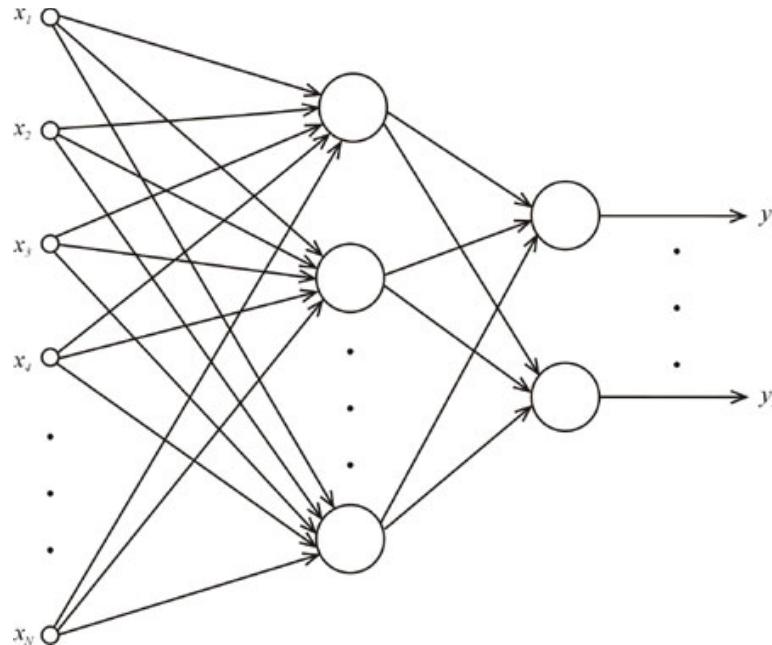
$$f(s) = s,$$

- to neuron nieliniowy staje się liniowym, gdyż

$$\text{wzbudzenie} = s = \sum_{i=1}^n w_i x_i$$

$$y = f(s) = s = \sum_{i=1}^n w_i x_i.$$

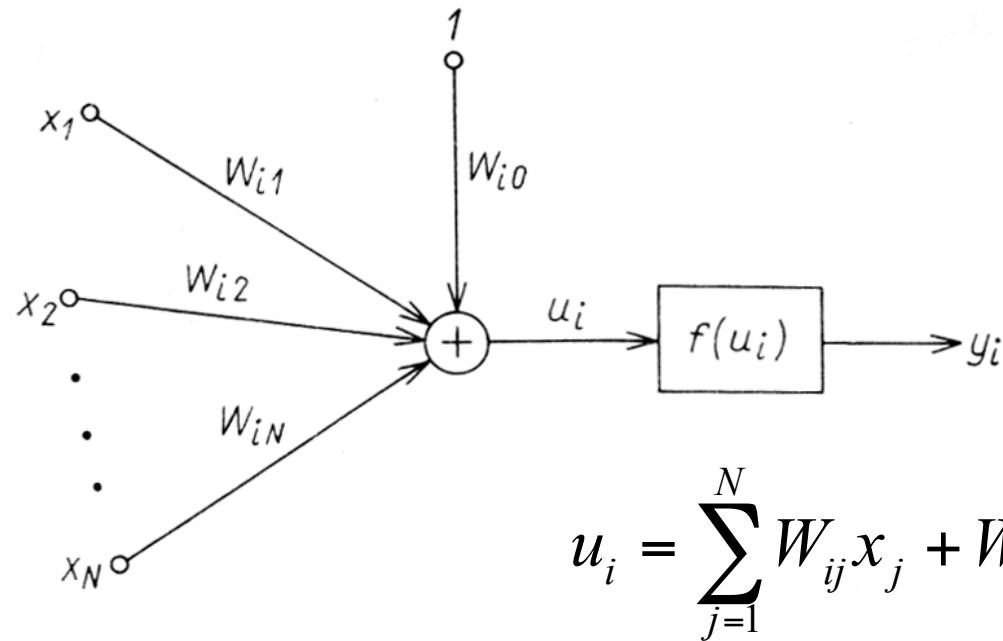
1. Wstęp



- Jednokierunkowe sieci wielowarstwowe radzą sobie z szerszą klasą problemów niż sieci liniowe.
- Algorytmy uczenia są bardziej skomplikowane.
- Zwykle stosuje się uczenie z nauczycielem.

2. Budowa jednokierunkowej sieci wielowarstwowej

2.1 Model neuronu



$$u_i = \sum_{j=1}^N W_{ij} x_j + W_{i0} \quad (2.1)$$

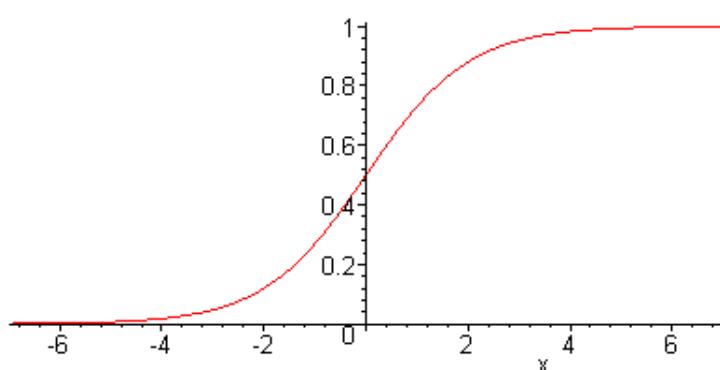
$\mathbf{x} = [x_1, x_2, \dots, x_N]^T$ - wektor wejściowy

$\mathbf{W}_i = [W_{i1}, W_{i2}, \dots, W_{iN}]^T$ - wektor wag i-tego neuronu

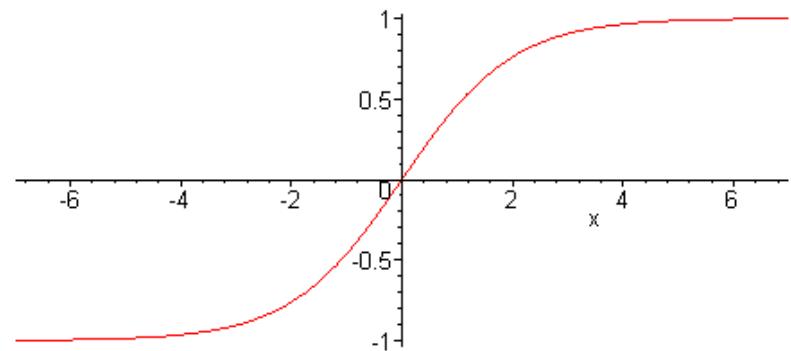
$y_i = f(u_i)$ - funkcja aktywacji

W_{i0} - próg

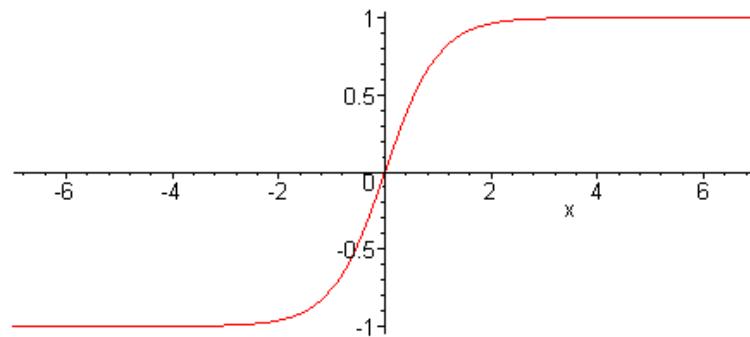
2.2 Funkcje aktywacji neuronu



$$f_u(u_i) = \frac{1}{1 + \exp(-\beta u_i)}$$



$$f_b(u_i) = 2f_u(u_i) - 1$$



$$f_b(u_i) = \tanh(\beta u_i)$$

Wyliczmy pochodną funkcji aktywacji dla $f_u(x) = \frac{1}{1 + \exp(-\beta x)}$

$$\begin{aligned}\frac{df_u(x)}{dx} &= \beta \frac{\exp(-\beta x)}{(1 + \exp(-\beta x))^2} = \\ &= \beta \frac{1}{1 + \exp(-\beta x)} \frac{1 + \exp(-\beta x) - 1}{1 + \exp(-\beta x)} = \beta f_u(x)(1 - f_u(x))\end{aligned}$$

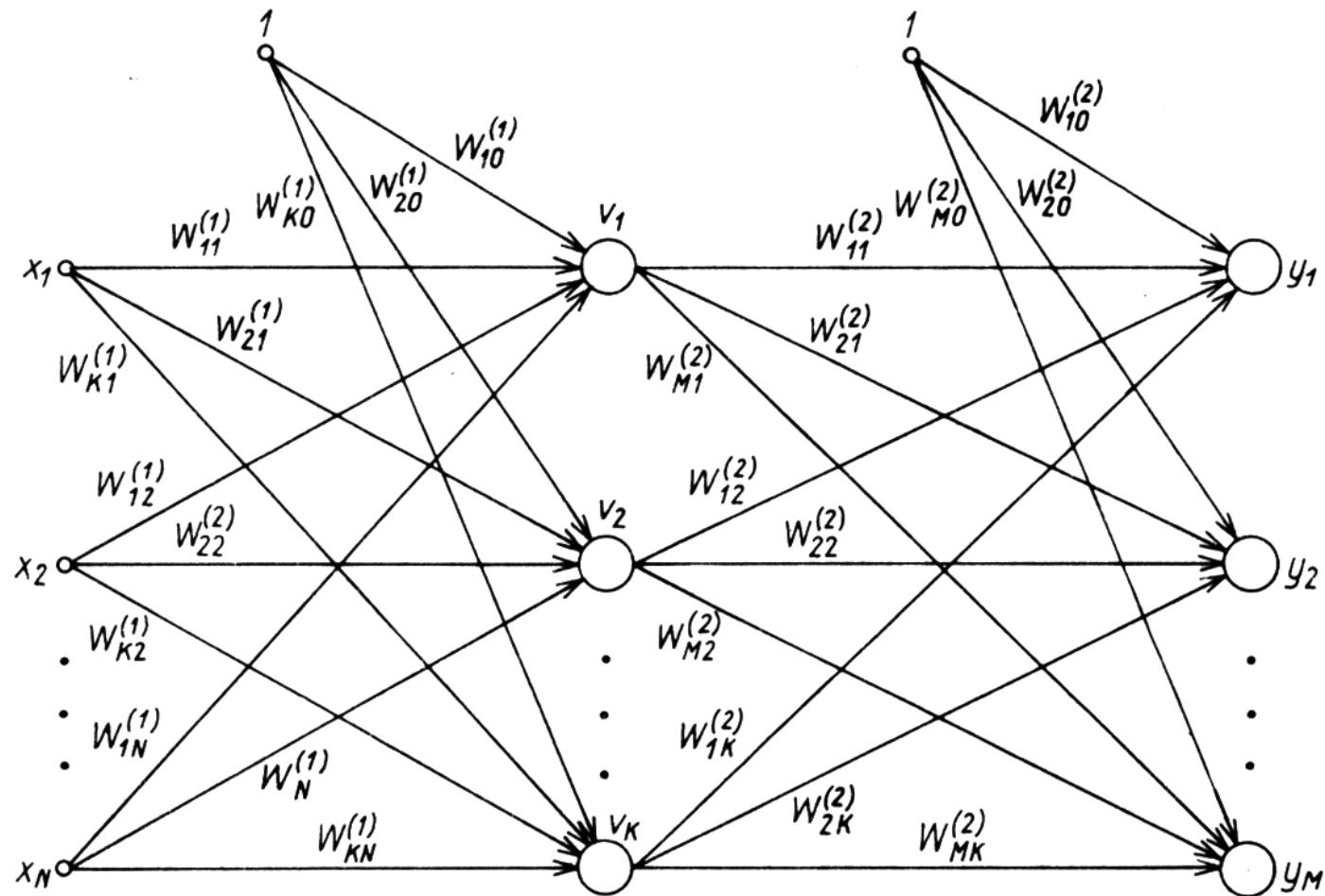
analogicznie dla $f_b(x) = 2f_u(x) - 1$

$$\frac{df_b(x)}{dx} = 2\beta f_u(x)(1 - f_u(x)) = \beta \frac{1 - f_b^2(x)}{2}$$

oraz dla $f_b(x) = \tanh(\beta x)$

$$\frac{df_b(x)}{dx} = \beta(1 - f_b^2(x))$$

2.3 Struktura sieci



3. Funkcja celu

Najskuteczniejszą metodą uczenia sieci jednokierunkowych jest optymalizacja funkcja celu, minimalizująca błąd między wartościami żądanymi a aktualnie otrzymywanyimi na wyjściu sieci.

Oznaczenia:

$\mathbf{d} = [d_1, d_2, \dots, d_M]^T$ - wektor znanych wartości żądanych na wyjściu

$U = \{(x(j), \mathbf{d}(j)) \mid j = 1 \dots p\}$ - p-elementowy zbiór uczący

$\mathbf{W} = [W_1, W_2, \dots, W_n]^T$ - wektor wszystkich wag w sieci

E - funkcja celu

3.1 Definicje

Dla jednej pary uczącej (\mathbf{x}, \mathbf{d}) najprostsza definicja funkcji celu ma postać błędu średniokwadratowego

$$E = \frac{1}{2} \sum_{i=1}^M (y_i - d_i)^2$$

Dla p -elementowego ciągu uczącego możemy przyjąć

$$E = \frac{1}{2} \sum_{j=1}^p \sum_{i=1}^M (y_i(j) - d_i(j))^2$$

Tak zdefiniowane funkcje celu są ciągłe i różniczkowalne, ponieważ funkcje aktywacji są ciągłe i różniczkowalne. Umożliwia to wykorzystanie metod gradientowych do ich minimalizacji.

3.2 Minimalizacja funkcji celu

W gradientowych metodach optymalizacji wykorzystuje się informacje o gradiencie funkcji celu

$$\nabla E = \left[\frac{\partial E}{\partial W_1}, \frac{\partial E}{\partial W_2}, \dots, \frac{\partial E}{\partial W_n} \right]^T$$

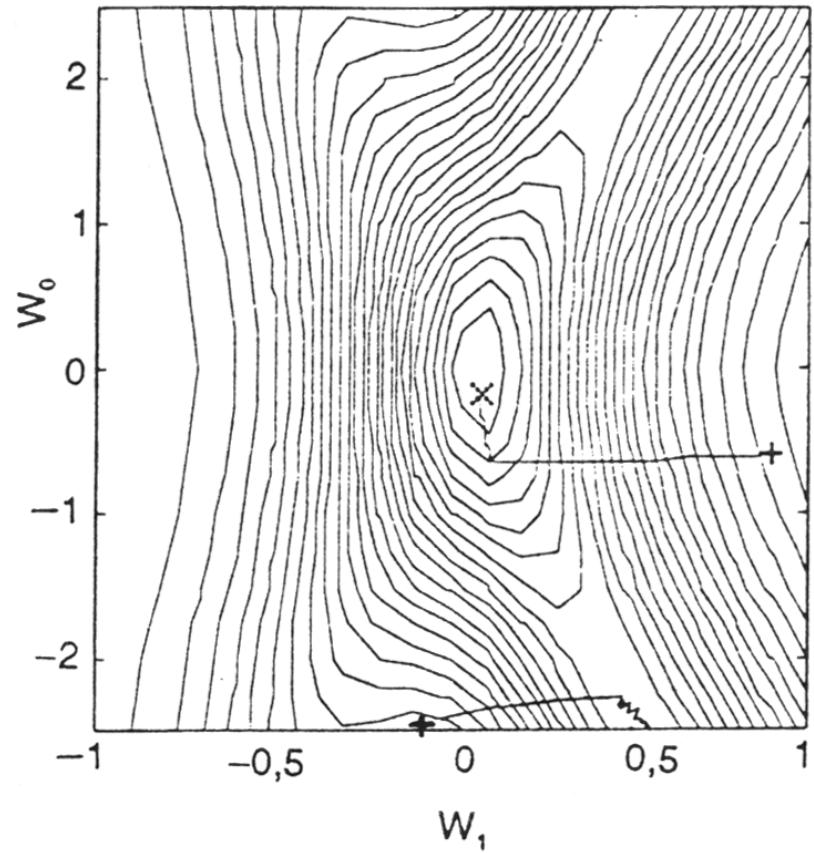
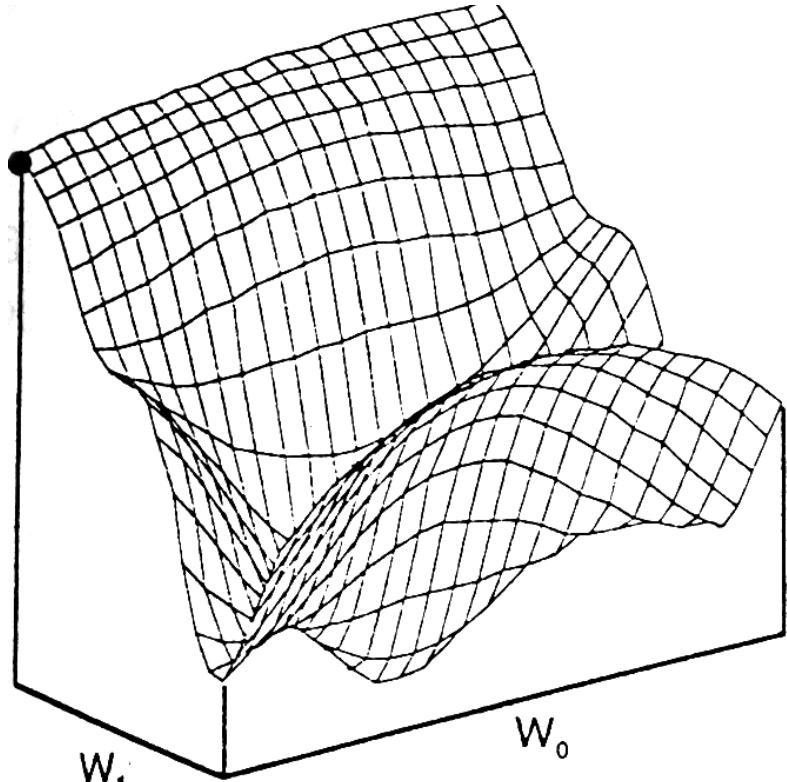
Na jego podstawie, w każdym kroku, określa się kierunek minimalizacji

$$p(W(k))$$

Następnie aktualizuje się wektor wag według formuły

$$W(k+1) = W(k) + \eta p(W(k))$$

gdzie η to współczynnik uczenia



Przykładowy wykres funkcji celu, dla sieci złożonej z jednego neuronu o dwóch wagach.

Jakość i prędkość minimalizacji zależy od metody wyznaczania kierunku oraz od metody wyznaczenia współczynnika uczenia.

3.3 Inne funkcje celu

W zależności od zastosowania, możemy definiować inne funkcje celu:

$$E(\mathbf{W}) = \sum_{i=1}^M |y_i - d_i|$$

Funkcja ta zapewnia bardziej równomierny udział poszczególnych składników błędu w ogólnej definicji funkcji celu.

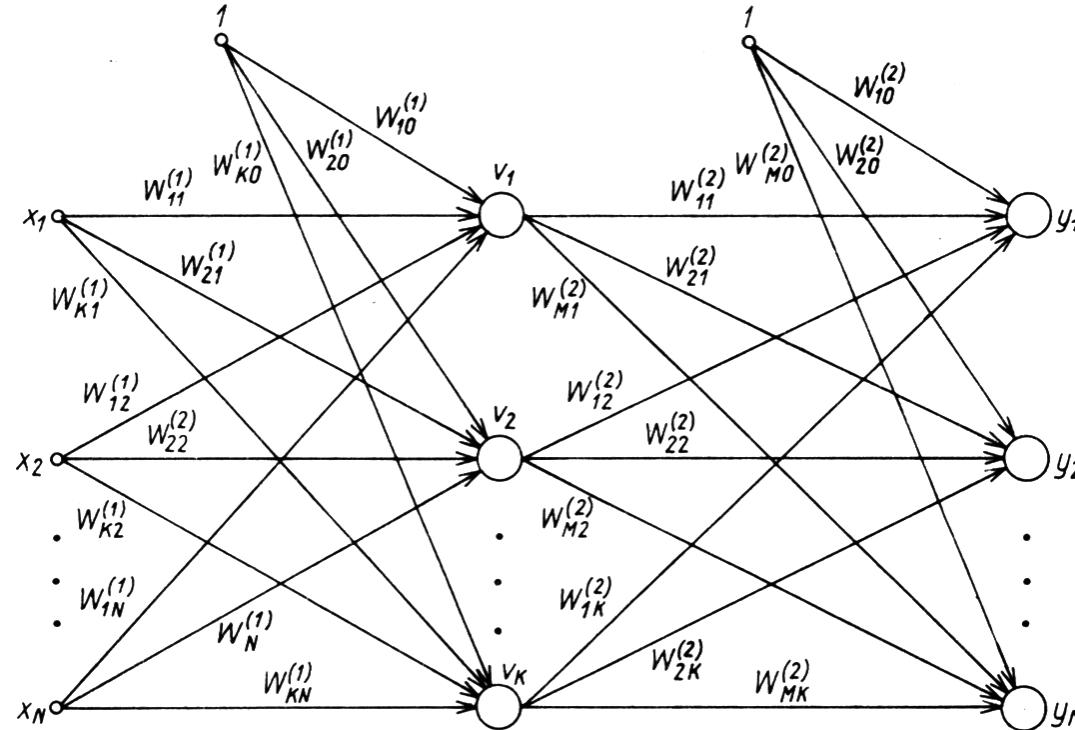
$$E(\mathbf{W}) = \sum_{i=1}^M (y_i - d_i)^{2K}$$

Funkcja minimalizuje największego odchylenia odpowiedzi od wielkości żądanej.

4. Algorytm wstecznej propagacji błędu

1. Analiza sieci neuronowej o zwykłym kierunku przepływu sygnałów, dla aktualnego wektora \mathbf{x} , w wyniki której otrzymujemy sygnały wyjściowe wszystkich warstw, a także pochodne
$$\frac{df(u_i^{(1)})}{du_i^{(1)}}, \frac{df(u_i^{(2)})}{du_i^{(2)}}, \dots, \frac{df(u_i^{(m)})}{du_i^{(m)}}$$
2. Utworzenie sieci propagacji wstecznej przez odwrócenie kierunków sygnałów, zastąpienie funkcji aktywacji ich pochodnymi, i podanie do byłego wejścia różnic między wartościami otrzymanymi w punkcie 1 a żądanymi. Obliczenie gradientu.
3. Na podstawie wyników z 1 i 2 dokonujemy adaptacji wag (uczenie) w sieci.
4. Powtarzamy cały proces dla wszystkich wzorców tak długo aż norma gradientu będzie mniejsza niż zadany stopień dokładności ε

4.1 Wyznaczenie gradientu



Przyjmijmy za funkcję celu dla powyższej sieci

$$E = \frac{1}{2} \sum_{j=1}^p \sum_{i=1}^M (y_i(j) - d_i(j))^2$$

Składowe gradientu otrzymujemy różniczkując funkcję celu

$$E = \frac{1}{2} \sum_{k=1}^M \left[f\left(\sum_{i=0}^K W_{ki}^{(2)} v_i\right) - d_k \right]^2 = \frac{1}{2} \sum_{k=1}^M \left[f\left(\sum_{i=0}^K W_{ki}^{(2)} f\left(\sum_{j=0}^N W_{ij}^{(1)} x_j\right)\right) - d_k \right]^2$$

Dla warstwy wyjściowej otrzymujemy

$$\frac{\partial E}{\partial W_{ij}^{(2)}} = (y_i - d_i) \frac{df(u_i^{(2)})}{du_i^{(2)}} v_j$$

Oznaczmy

$$\delta_i^{(2)} = (y_i - d_i) \frac{df(u_i^{(2)})}{du_i^{(2)}}$$

Więc ostatecznie

$$\frac{\partial E}{\partial W_{ij}^{(2)}} = \delta_i^{(2)} v_j$$

W wyliczeniu składowych gradientu dla warstwy ukrytej korzystamy z rezultatów obliczeń w warstwie wyjściowej

$$\frac{\partial E}{\partial W_{ij}^{(1)}} = \delta_i^{(1)} x_j$$

przy czym

$$\delta_i^{(1)} = \sum_{k=1}^M \delta_k^{(2)} W_{ki}^{(2)} \frac{df(u_i^{(1)})}{du_i^{(1)}}$$

Powyższe wzory można przez analogię rozszerzyć na sieć posiadającą więcej niż jedną warstwę ukrytą.

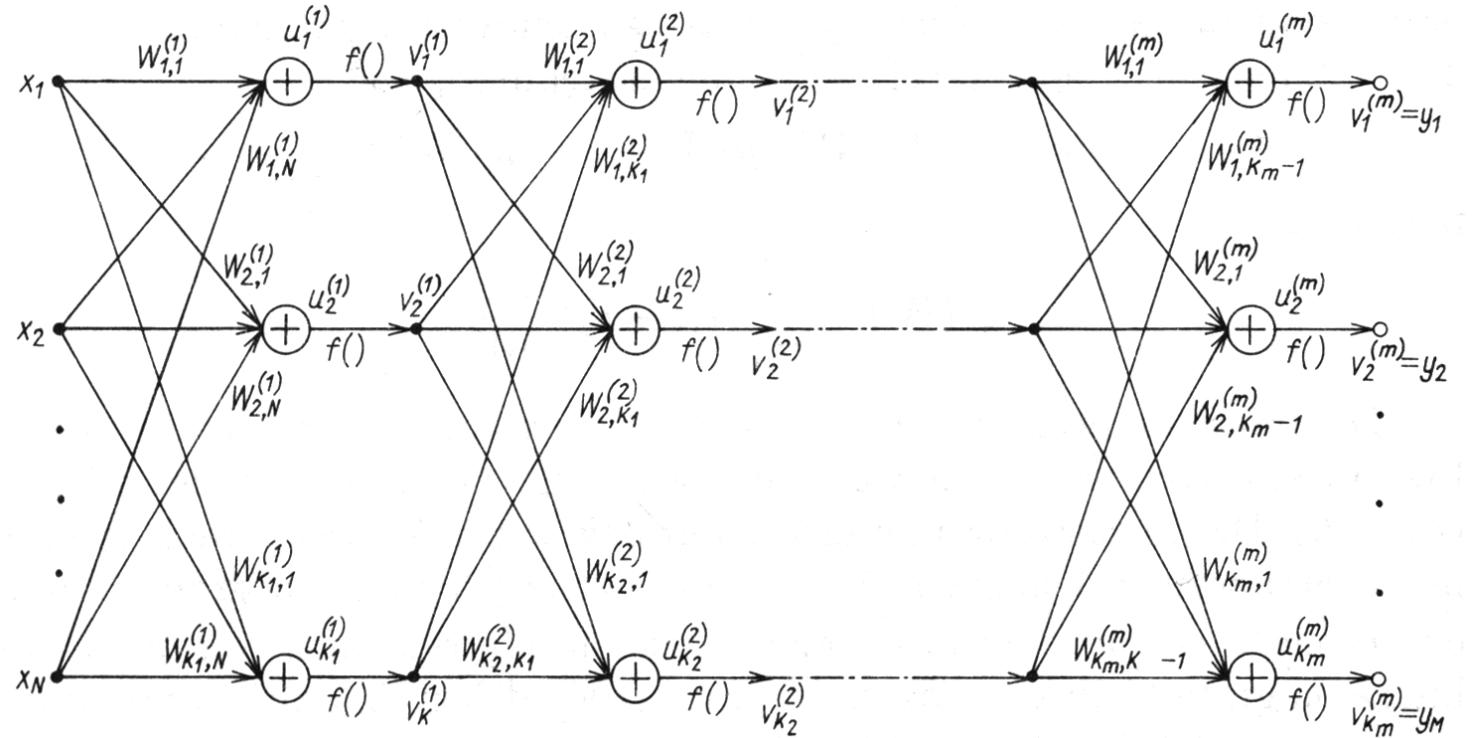
Posiadając wszystkie składowe gradientu można przystąpić do aktualizacji wag przyjmując

$$\Delta \mathbf{W} = -\eta \nabla E(\mathbf{W})$$

4.2 Wyznaczenie gradientu metodą grafów przepływowych

Graf przepływowy graf skierowany w którym:

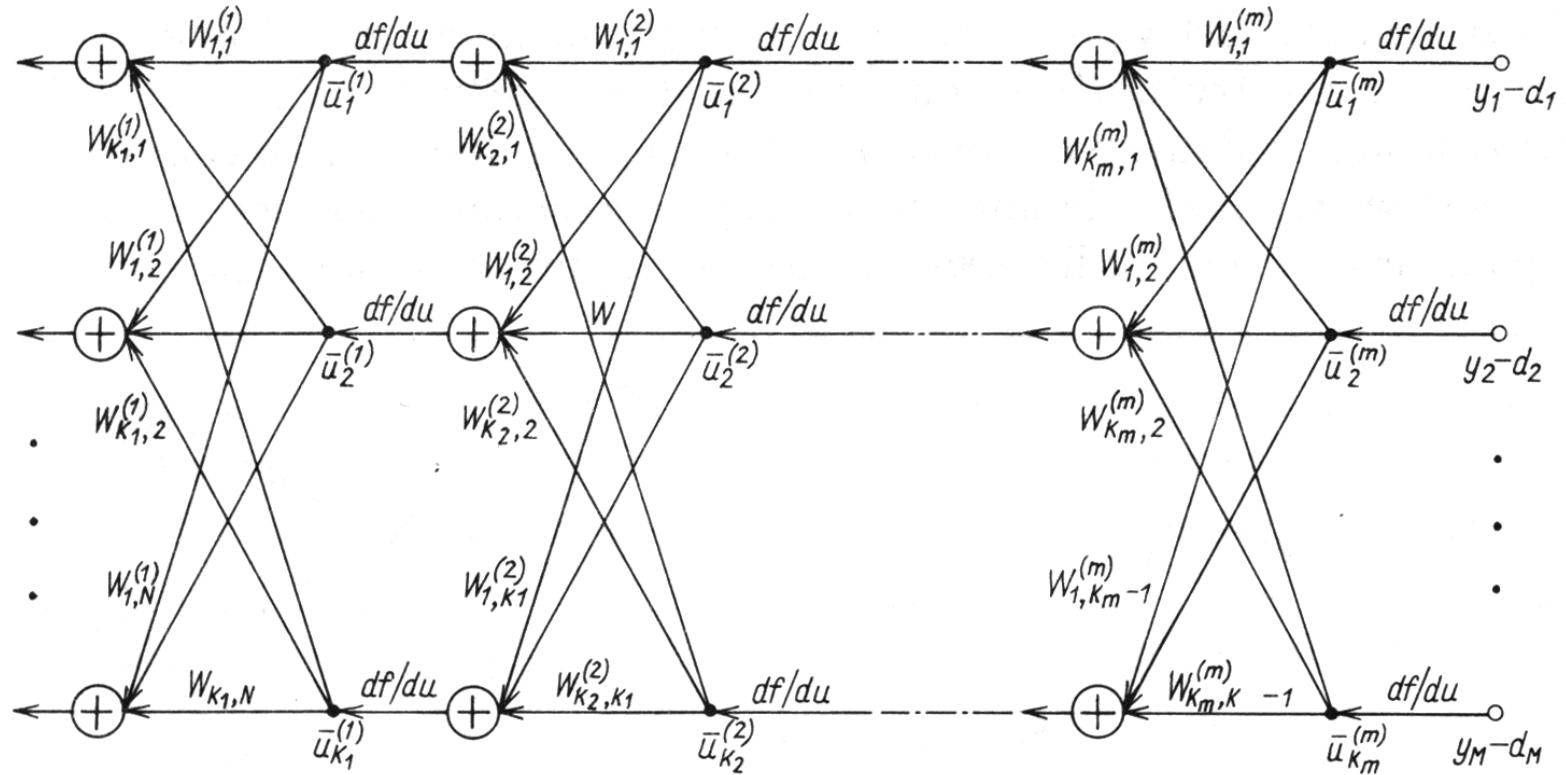
1. Sygnał przepływa wzduż krawędzi tylko w kierunku zgodnym z kierunkiem krawędzi.
2. Przepływający przez krawędź przepuszczany jest przez funkcję przypisaną tej kawędzi.
3. Wartość węzła jest sumą wartości wszystkich sygnałów wchodzących do węzła
4. Wartość każdego węzła jest przesyłana przez wszystkie krawędzie wychodzące z węzła



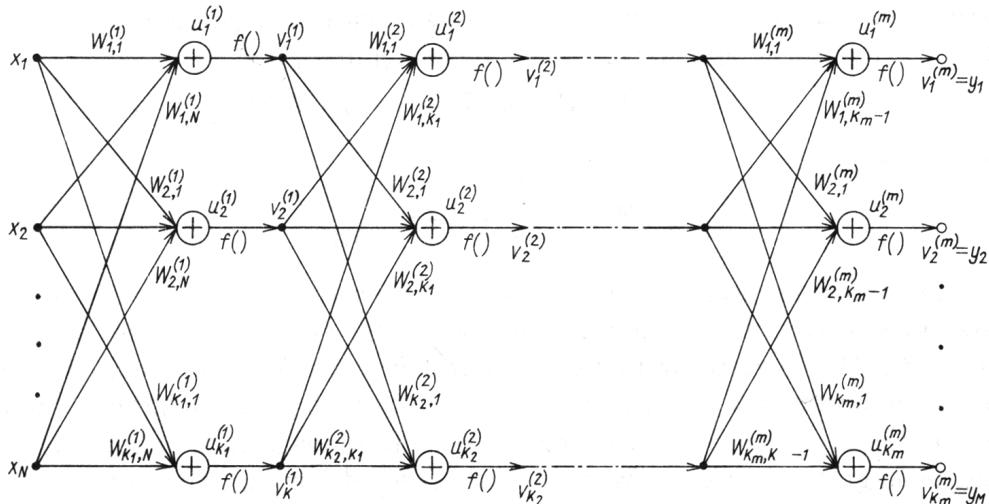
Dla sieci przedstawionej jako graf przepływowo, tworzymy graf dołączony

Graf G	Graf \hat{G}
x_1 \xrightarrow{W} x_2	\hat{x}_1 \xleftarrow{W} \hat{x}_2
$x_2 = f(x_1, k)$ x_1 $\xrightarrow{\quad}$ x_2	β $\beta = \frac{\partial f(x, k)}{\partial x} \Big _{x=x-1}$ \hat{x}_1 $\xleftarrow{\beta}$ \hat{x}_2

Do wejścia otrzymanego grafu dołączonego podajemy różnice sygnałów otrzymanych i oczekiwanych w grafie oryginalnym.



Analizując oba grafy, wyznaczamy składowe wektora gradientu

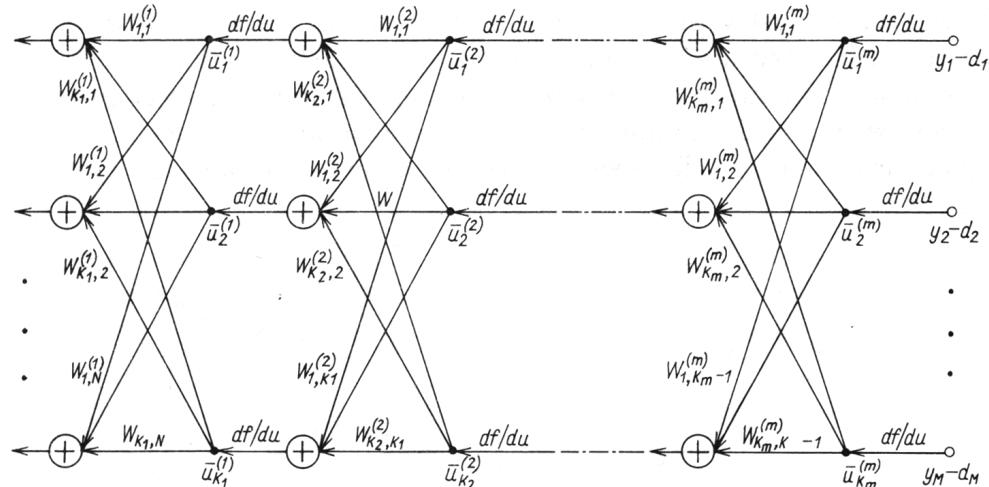


Dla warstwy wyjściowej

$$\frac{\partial E(\mathbf{W})}{\partial W_{ij}^{(m)}} = v_j^{(m-1)} \hat{u}_i^{(m)}$$

Dla k-tej warstwy ukrytej

$$\frac{\partial E(\mathbf{W})}{\partial W_{ij}^{(k)}} = v_j^{(k-1)} \hat{u}_i^{(k)}$$



Dla warstwy pierwszej
warstwy ukrytej

$$\frac{\partial E(\mathbf{W})}{\partial W_{ij}^{(1)}} = x_j u_i^{(1)}$$

5. Algorytmy gradientowe optymalizacji w zastosowaniu do uczenia sieci

Algorytmy te bazują na informacji o gradientie i na rozwinięciu w szereg Taylora funkcji celu, w najbliższym sąsiedztwie znanego rozwiązania \mathbf{W} .

$$E(\mathbf{W} + \mathbf{p}) = E(\mathbf{W}) + [\mathbf{g}(\mathbf{W})]^T \mathbf{p} + \frac{1}{2} \mathbf{p}^T \mathbf{H}(\mathbf{W}) \mathbf{p} + O(h^3)$$

$$\mathbf{g}(\mathbf{W}) = \nabla E \qquad \mathbf{H}(\mathbf{W}) = \begin{bmatrix} \frac{\partial E}{\partial W_1 \partial W_1} & \cdots & \frac{\partial E}{\partial W_n \partial W_1} \\ \vdots & & \vdots \\ \frac{\partial E}{\partial W_1 \partial W_n} & \cdots & \frac{\partial E}{\partial W_n \partial W_n} \end{bmatrix}$$

Celem działania tych algorytmów jest minimalizacja funkcji celu poprzez wyznaczanie, w każdym kroku, kierunku poszukiwań \mathbf{p}_k oraz wartości η_k takich że

$$E(\mathbf{W}_k + \eta_k \mathbf{p}_k) < E(\mathbf{W}_k)$$

5.1 Algorytm największego spadku

W tym algorytmie jako kierunek poszukiwań obieramy ujemny gradient

$$\mathbf{p}_k = -\mathbf{g}(\mathbf{W}_k)$$

Wadą tego algorytmu jest niewykorzystanie informacji o krzywiźnie funkcji zawartej w hesjanie, co powoduje że metoda ta jest wolno zbieżna.

W okolicy punktu optymalnego gradient przyjmuje małe wartości, co wydłuża czas działania algorytmu.

Poprawę efektywności może przynieść umiejętne dobieranie parametru η_k oraz zastosowanie czynnika *momentum* uzależniającego wykonanie kroku algorytmu od kroku wykonanego w poprzedniej iteracji.

$$\Delta \mathbf{W}_k = \eta_k \mathbf{p}_k + \alpha (\mathbf{W}_k - \mathbf{W}_{k-1}) \quad \alpha \in [0, 1]$$

Ulepszenie to przyspiesza działanie algorytmu nawet 10-krotnie

5.2 Algorytm zmiennej metryki

W algorytmie tym, wyznaczając wektor poszukiwań, wykorzystujemy kwadratowe przybliżenie funkcji celu, a wektor poszukiwań wyznaczamy korzystając z formuły

$$\mathbf{p}_k = - [\mathbf{H}(\mathbf{W}_k)]^{-1} \mathbf{g}(\mathbf{W}_k)$$

Z przyczyn praktycznych rezygnuje się w tym algorytmie z dokładnego Wyznaczenia hesjanu, a w zamian stosuje się jego przybliżenie stosując metodę zmiennej metryki.

Oznaczmy

$$\mathbf{s}_k = \mathbf{W}_k - \mathbf{W}_{k-1}$$

$$\mathbf{r}_k = \mathbf{g}(\mathbf{W}_k) - \mathbf{g}(\mathbf{W}_{k-1})$$

$$\mathbf{V}_k = [\mathbf{G}(\mathbf{W}_k)]^{-1}$$

Reguła Broydena-Goldfarba-Fletcher-Shanno (BFGS)

$$\mathbf{V}_k = \mathbf{V}_{k-1} + \left[1 + \frac{\mathbf{r}_k^T \mathbf{V}_{k-1} \mathbf{r}_k}{\mathbf{s}_k^T \mathbf{r}_k} \right] \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{r}_k} - \frac{\mathbf{s}_k \mathbf{r}_k^T \mathbf{V}_{k-1} + \mathbf{V}_{k-1} \mathbf{r}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{r}_k}$$

Reguła Reguła Davidona-Fletcher-Powella (DFP)

$$\mathbf{V}_k = \mathbf{V}_{k-1} + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{r}_k} - \frac{\mathbf{V}_{k-1} \mathbf{r}_k \mathbf{r}_k^T \mathbf{V}_{k-1}}{\mathbf{r}_k^T \mathbf{V}_{k-1} \mathbf{r}_k}$$

Za wartość startową przyjmuje się zwykle $V_0 = I$ oraz pierwszy krok algorytmu wykonuje zgodnie z algorytmem największego spadku.

Metoda zmiennej metryki charakteryzuje się zbieżnością superliniową. Jest obecnie uważana za jedną z najlepszych metod optymalizacji funkcji wielu zmiennych. Jednakże konieczność aproksymowania macierzy hesjanu ogranicza jej zastosowanie do problemów z niewielką ilością zmiennych (<1000).

5.3 Algorytm Levenberga- Marquardta

Przyjmijmy definicję funkcji celu odpowiadającej istnieniu jednego wzorca uczącego

$$E(\mathbf{W}) = \frac{1}{2} \sum_{i=1}^M [e_i(\mathbf{W})]^2$$

oraz oznaczmy

$$e_i = [y_i(\mathbf{W}) - d_i]$$

$$\mathbf{e}(\mathbf{W}) = \begin{bmatrix} e_1(\mathbf{W}) \\ e_2(\mathbf{W}) \\ \dots \\ e_M(\mathbf{W}) \end{bmatrix}$$

$$\mathbf{J}(\mathbf{W}) = \begin{bmatrix} \frac{\partial e_1}{\partial W_1} & \frac{\partial e_1}{\partial W_2} & \dots & \frac{\partial e_1}{\partial W_n} \\ \frac{\partial e_2}{\partial W_1} & \frac{\partial e_2}{\partial W_2} & \dots & \frac{\partial e_2}{\partial W_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial e_M}{\partial W_1} & \frac{\partial e_M}{\partial W_2} & \dots & \frac{\partial e_M}{\partial W_n} \end{bmatrix}$$

Algorytm LM również wyznacza wektor zgodnie z regułą

$$\mathbf{p}_k = -[\mathbf{H}(\mathbf{W}_k)]^{-1} \mathbf{g}(\mathbf{W}_k)$$

Jednakże zarówno gradient jak i przybliżoną macierz hesjanu $\mathbf{G}_k(\mathbf{W}_k)$ wyznaczamy w oparciu o jakobian

$$\mathbf{g}(\mathbf{W}) = [\mathbf{J}(\mathbf{W})]^T \mathbf{e}(\mathbf{W})$$

$$\mathbf{G}(\mathbf{W}_k) = [\mathbf{J}(\mathbf{W}_k)]^T \mathbf{J}(\mathbf{W}_k) + v_k \mathbf{1}$$

Wartość skalarna v_k jest dobierana arbitralnie i zmniejszana w trakcie uczenia do zera w okolicy rozwiązania minimalnego.

Wartość v_k nazywamy czynnikiem regularizacyjnym.

Na początku działania algorytmu gdy wartość v_k jest duża, można przyjąć że

$$\mathbf{G}(\mathbf{W}_k) \approx v_k \mathbf{1}$$

Wtedy poszukiwanie odbywa się zgodnie z metodą największego spadku

$$\mathbf{p}_k = -\frac{\mathbf{g}(\mathbf{W}_k)}{v_k}$$

W miarę zbliżania się do rozwiązania v_k jest zmniejszane i pierwszy składnik wzoru na przybliżony hesjan zaczyna odgrywać coraz większą rolę.

5.5 Metoda gradientów sprzężonych

W metodzie tej rezygnuje się z bezpośredniej informacji o hesjanie. Kierunek poszukiwań konstruowany jest tak aby był ortogonalny i sprzężony ze wszystkimi kierunkami wyznaczonymi w poprzednich krokach algorytmu.

Wektor spełniający te założenia ma postać

$$\mathbf{p}_k = -\mathbf{g}_k + \beta_{k-1} \mathbf{p}_{k-1}$$

Przy czym

$$\mathbf{g}_k = \mathbf{g}(\mathbf{W}_k)$$

Zaś kumuluje informację o ortogonalności i sprzężoności wszystkich wektorów Kierunku wyznaczanych w poprzednich k-1 krokach.

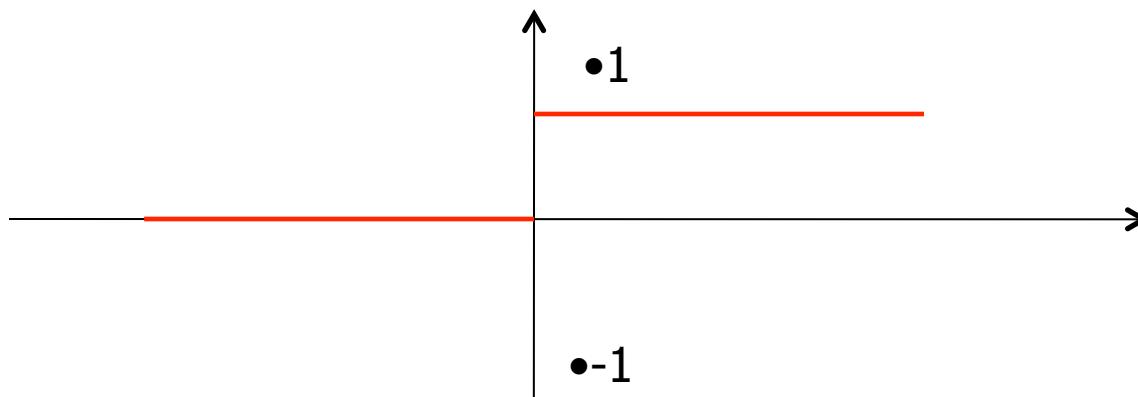
$$\beta_{k-1} = \frac{\mathbf{g}_k^T (\mathbf{g}_k - \mathbf{g}_{k-1})}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$

$$\beta_{k-1} = \frac{\mathbf{g}_k^T (\mathbf{g}_k - \mathbf{g}_{k-1})}{-\mathbf{p}_{k-1}^T \mathbf{g}_{k-1}}$$

Ze względu na kumulację błędów zaokrągleń, metoda zatracza własność ortogonalności między wektorami, zatem co określoną ilość iteracji należy przeprowadzać jej ponowny start.

Efektywność tej metody jest słabsza niż efektywność metody zmiennej metryki, jednakże jej prostota i niewielkie wymagania pamięciowe czynią ją najlepszą metodą optymalizacji funkcji o dużej ilości zmiennych.

Neuron z funkcją progową



$$f(s) = \begin{cases} 1 & \text{dla } s \geq 0, \\ 0 & \text{dla } s < 0. \end{cases}$$

- (Inne nazwy to *funkcja unipolarna* lub *funkcja Heaviside'a*)

Przykłady funkcji aktywacji

- Funkcja bipolarna

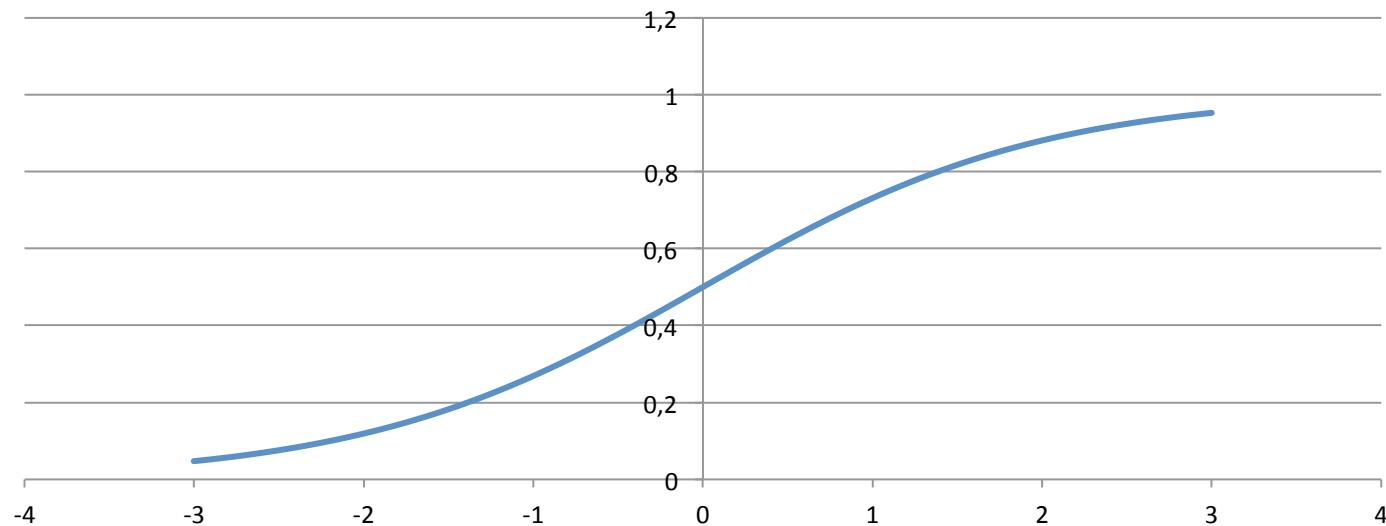
$$f(s) = \begin{cases} 1 & \text{gdy } s \geq 0, \\ -1 & \text{gdy } s < 0. \end{cases}$$

- Funkcja unipolarna

$$f(s) = \begin{cases} 1 & \text{gdy } s \geq 0, \\ 0 & \text{gdy } s < 0. \end{cases}$$

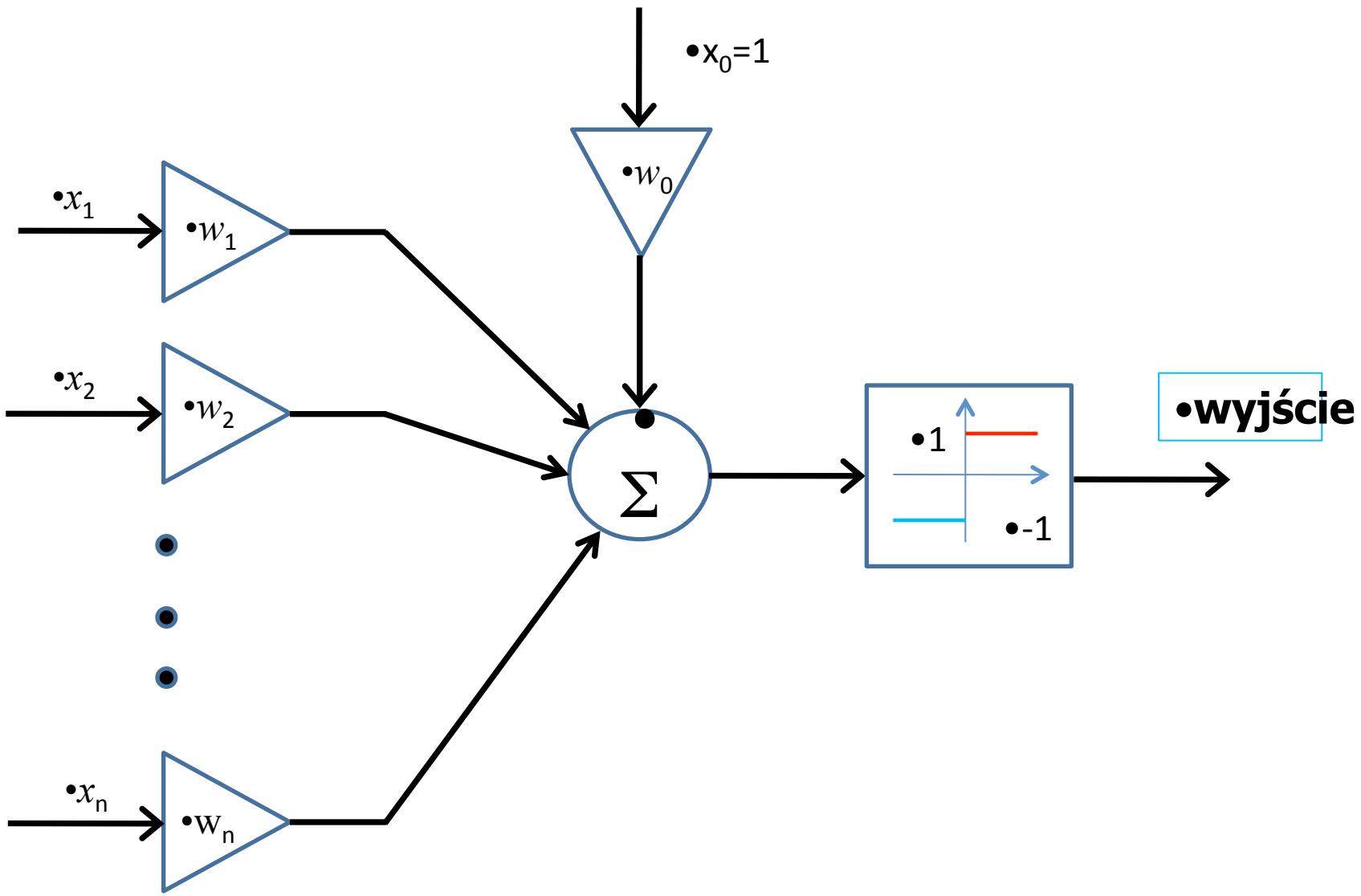
- Funkcja sigmoidalna

$$f_{\beta}(x) = \frac{1}{1 + e^{-\beta x}} \quad x \in \mathbb{R}.$$



- Wykres dla $\beta=1$.

- Poniżej jest przedstawiony schemat *perceptronu*



•Funkcja aktywacji perceptronu

- Rysunkowy schemat perceptronu sugeruje następującą funkcję aktywacji

$$f(s) = \begin{cases} +1, & \text{gdy } s \geq 0, \\ -1, & \text{gdy } s < 0. \end{cases}$$

- W literaturze spotyka się także (może nawet częściej) inną definicję

$$f(s) = \begin{cases} 1, & \text{gdy } s \geq 0, \\ 0, & \text{gdy } s < 0. \end{cases}$$

- W gruncie rzeczy nie ma tu zasadniczej różnicy, gdyż perceptron służy do klasyfikacji: czy sygnał wejściowy (reprezentowany przez punkt lub wektor kolumnowy)

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbf{R}^n$$

- należy do jednej klasy czy do drugiej.

- Cześć liniowa perceptronu (sumator, Σ)

- Dla danych wag perceptronu w_1, \dots, w_n oraz progu $-b$, gdy impulsy wejściowe są równe x_1, \dots, x_n , to pobudzenie neuronu jest równe

$$s = w_1x_1 + w_2x_2 + \dots + w_nx_n + b = \sum_{i=1}^n w_i x_i + b.$$

- Używając symbolu iloczynu skalarnego możemy to zapisać też tak

$$s = w \otimes x + b,$$

- gdzie

$$w = (w_1, \dots, w_n), x = (x_1, \dots, x_n) \in \mathbf{R}^n, b \in \mathbf{R}.$$

• Istota działania perceptronu

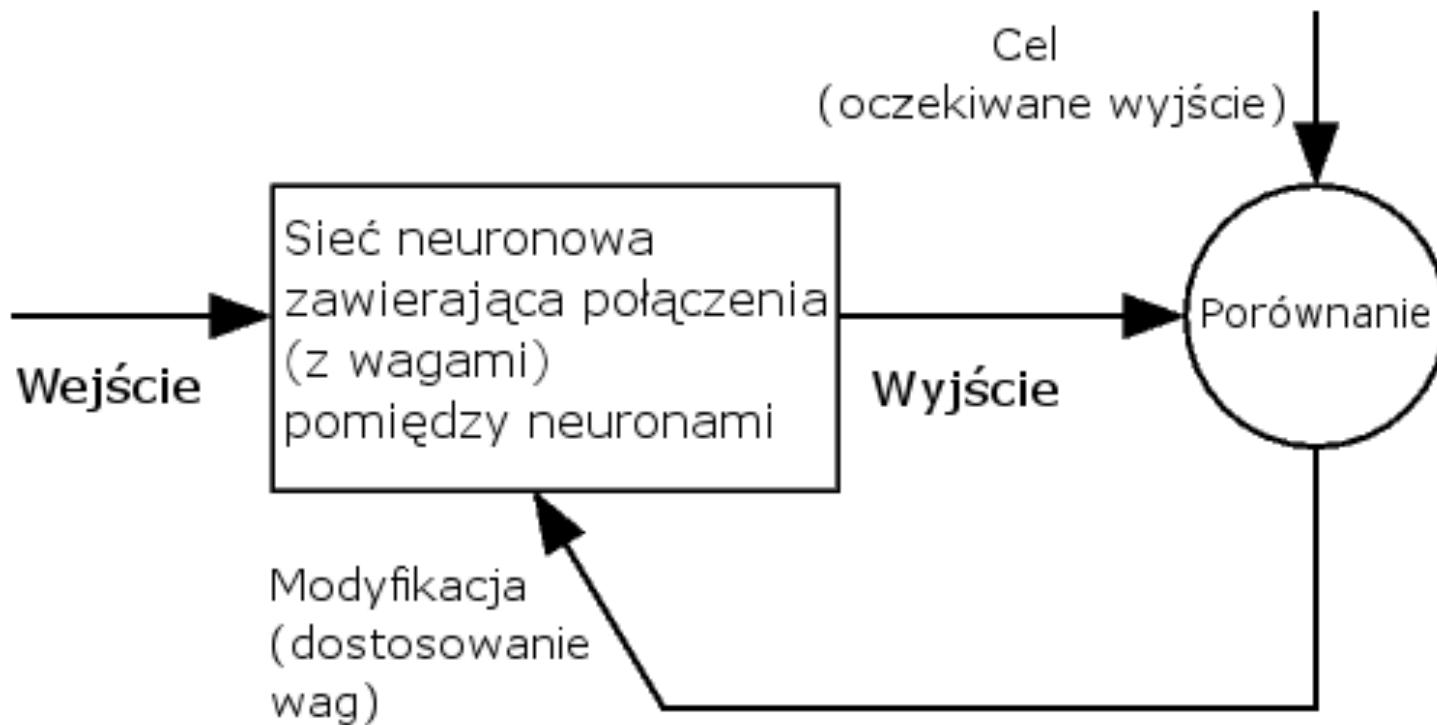
- Funkcja aktywacji rozróżnia dwa przypadki: (i) $s > 0$, (ii) $s \leq 0$, zatem to co jest kluczowe w działaniu perceptronu sprowadza się do klasyfikacji punktów (wektorów) $x = (x_1, \dots, x_n)$ wg poniższych nierówności:

$$(i) \quad \sum_{i=1}^n w_i x_i + b > 0,$$

$$(ii) \quad \sum_{i=1}^n w_i x_i + b \leq 0.$$

- Oznacza to, że punkty x spełniające nierówność (i) będą klasyfikowane do jednej kategorii, a punkty spełniające nierówność (ii) do drugiej.

- Ogólny schemat uczenia jednokierunkowej sieci neuronowej z nauczycielem (ang. *supervised*)



Algorytm uczenia perceptronu

- Uczenie sieci polega na dobieraniu wag tak, aby dla zadanych impulsów wejściowych otrzymywać oczekiwane wartości wyjściowe z neuronów. Za chwilę zajmiemy się prostym przypadkiem (mamy tylko jeden neuron) poszukiwania wag w_i oraz progu b dla perceptronu.
- Dany mamy ciąg uczący, który składa się z dowolnej skończonej liczby wektorów oraz dodatkowych informacji mówiących do której z dwóch klas te wektory należą. Tę dodatkową informację będziemy reprezentować symbolem d ($d=+1$ (pierwsza klasa) lub $d=-1$ (druga klasa)). Przy numeracji kolejnych wektorów uczących użyjemy indeksów górnych, np. $x^{(2)}$, aby odróżniać się od numerowania składowych wektora (indeksy dolne). Mamy więc ciąg uczący

$$x^{(1)}, x^{(2)}, \dots, x^{(T)} \in \mathbf{R}^n, d_1, \dots, d_T \in \{-1, +1\}.$$

- Liczba elementów ciągu uczącego (długość ciągu) oznaczamy przez $T = \text{liczba epok}$.

Algorytm uczenia perceptronu

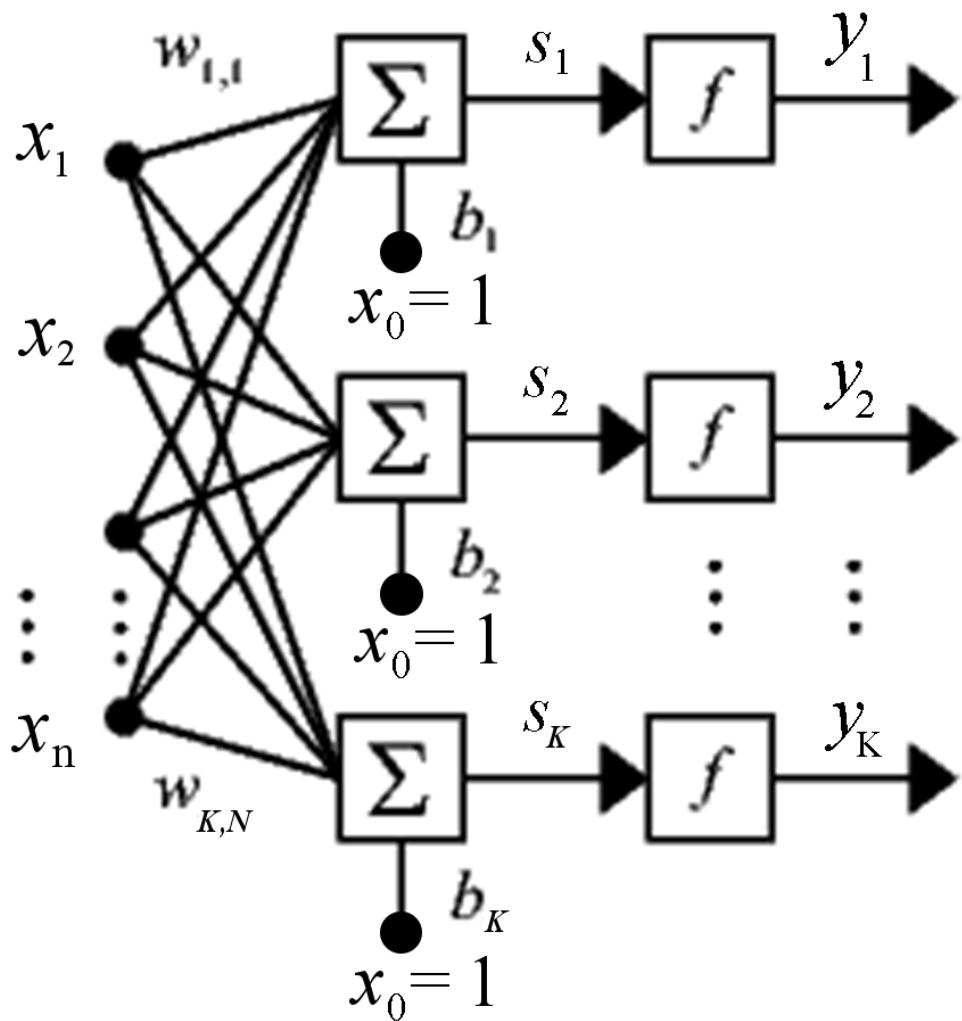
1. Losujemy wagi początkowe w_1, \dots, w_n oraz próg b .
2. Dla $t=1$ do $t=T$ wykonujemy 3. i 4.:
3. Na wejście podajemy kolejny wektor uczącego $x=x^{(t)}$ i obliczamy $y=f(s(x))$.
4. Porównujemy wartość wyjściową y z oczekiwana wartością $d=d_t$ z ciągu uczącego.
5. Dokonujemy modyfikacji wag:
 - 4.1 Jeżeli $y \neq d$, to $w := w + d \cdot x$, $b = b + d$.
 - 4.2 Jeżeli $y = d$, to wagi pozostają bez zmian.
5. Jeżeli w 4. była choć jedna modyfikacja, to wracamy do 2.
6. Koniec.

• Uwagi:

- 1) Zauważmy, że w p. 4.1) operacja $w := w + d \cdot x$ oznacza tak naprawdę $w := w + x$, $b = b + 1$ lub $w := w - x$, $b = b - 1$.
- 2) Tak naprawdę to nie musimy obliczać wartości $y = f(s)$ w p. 2. Wystarczy sprawdzać warunek: $s = w \circ x + b > 0$.

- Pojedynczy neuron to oczywiście za mało, aby oczekiwany, że taki system obliczeniowy był w praktyce użyteczny. Prawdziwą siłę obliczeniową możemy spodziewać się uzyskać dopiero gdy zastosujemy zbiór wielu sztucznych neuronów odpowiednio połączonych. Wtedy powstaje **sztuczna sieć neuronowa** o pewnej liczbie wejść i pewnej liczbie wyjść. W zastosowaniach spotyka się wiele struktur takich sieci (czyli tzw. topologii) scharakteryzowanych przez sposób połączenia poszczególnych neuronów. W zasadzie możemy te sieci podzielić na dwie duże klasy: **sieci jednokierunkowe** (ang. *feed forward*) oraz **sieci rekurencyjne**.
- W sieciach jednokierunkowych (jedno- lub wielowarstwowych) sygnały płyną tylko w jednym kierunku od wejścia do wyjścia. Oznacza to, że w takiej sieci nie ma połączeń powrotnych. Notomiast w sieciach rekurencyjnych sygnały mogą zawracać i przebiegać wielokrotnie przez sieć. W takiej sieci występują więc połączenia powrotne – sygnały wyjściów pewnych neuronów mogą być kierowane na wyjścia tych samych lub innych neuronów wielokrotnie.

• Jednowarstwowa i jednokierunkowa sieć



- N =liczba wejść
- K =liczba wyjść

$$s_k = \sum_{i=0}^N w_{k,i} x_i, \\ y_k = f(s_k) \quad \text{dla } k = 1, K, K.$$

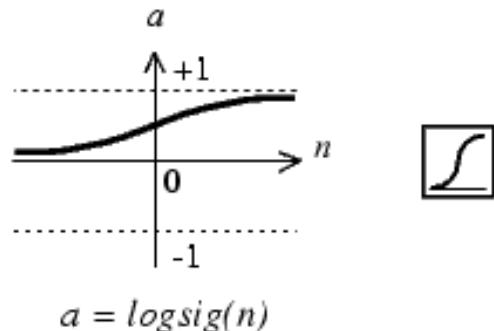
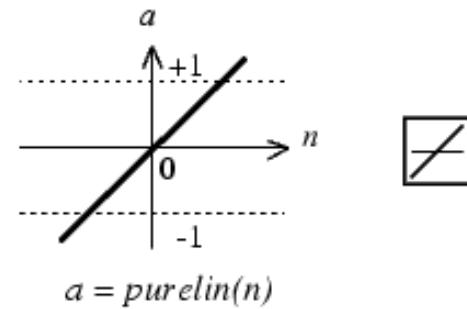
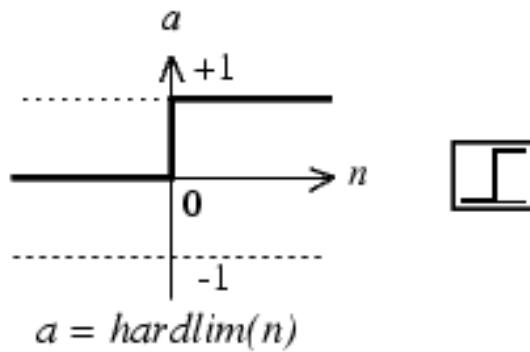
- Dla wygody progi (b_i) włączamy w powyższej sumie jako wagi odpowiadające impulsowi $x_0=1$. Tak więc $w_{k,0}=b_k$.

Sieci neuronowe w MATLAB-e

- MATLAB jest środowiskiem do obliczeń numerycznych. Pierwotnie powstał jako pakiet wspomagający obliczenia w algebrze liniowej, stąd posiada ogromne możliwości operacji na macierzach. Jego nazwa nawiązuje do tych źródeł (MATLAB od **Matrix Laboratory**). Obecnie jest to jednak wszechstronne środowisko obliczeniowe oferujące możliwości obliczeń w praktycznie każdej dziedzinie metod numerycznych.
- Niezwykle cennym elementem w MATLAB-a są tzw. **toolbox-y**, czyli wyspecjalizowane pakiety do obliczeń i zastosowań w konkretnych obszarach. Jednym z takich pakietów jest *Neural Network*.

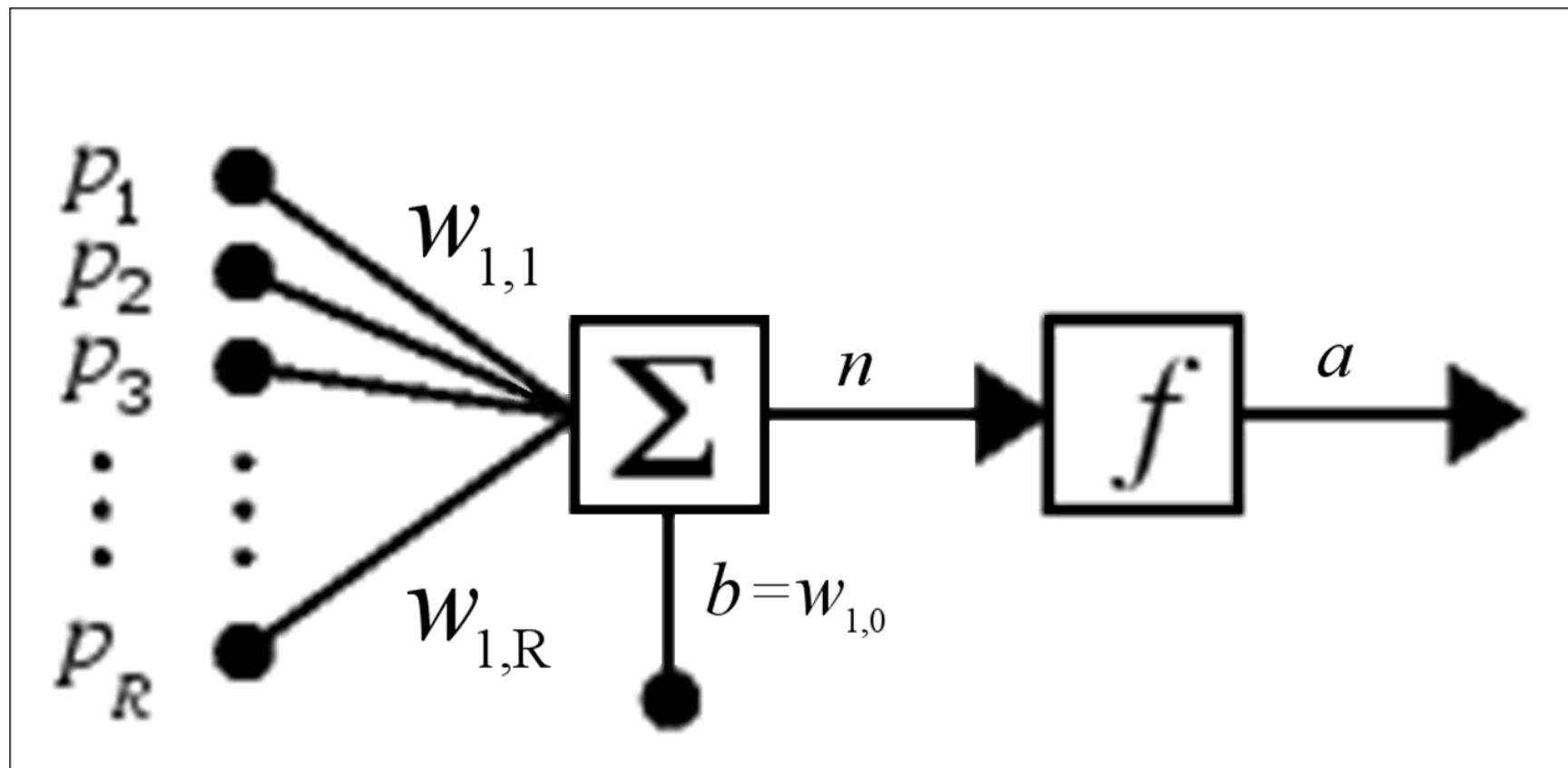
- Podstawowe funkcje aktywacji neuronów – nazwy zgodne z pakietem Neural Network MATLAB-a

- hardlim = funkcja unipolarna (Heaviside'a)
- purelin = funkcja liniowa, $f(s)=s$
- logsig = funkcja sigmoidalna

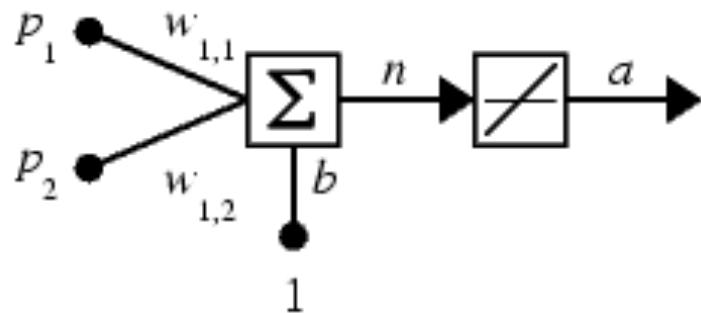


- Standardowa symbolika używana w opisie neuronów w MATLAB-ie

- Dla pojedynczego neuronu nie ma w zasadzie potrzeby używania dla wag notacji z podwójnym indeksem $w_{1,j}$. Można posługiwać się pojedynczym indeksem j oznaczając wagi przez w_j .



Przykład (neuron liniowy)



- Ciąg uczący składa się z czterech par ($Q=4$):
- Wejście: $p^{(1)} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $p^{(2)} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $p^{(3)} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$, $p^{(4)} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$,
- Wyjście: $T = [-100, 50, 50, 100]$.
- Wagi początkowe i próg:
 $w_{1,1} = 1.0$, $w_{1,2} = 2.0$, $b = 0$.

•Komendy MATLAB-a

- $P = [1 \ 2 \ 2 \ 3; \ 2 \ 1 \ 3 \ 1];$
- $T = [-100 \ 50 \ 50 \ 100];$
- $\text{net} = \text{newlin}(P, T);$
- $\text{sim}(\text{net}, P);$
- $\text{net} = \text{train}(\text{net}, P, T)$

• Przygotowanie tablic z danymi. P =wejścia, T =oczekiwane wartości na wyjściu.

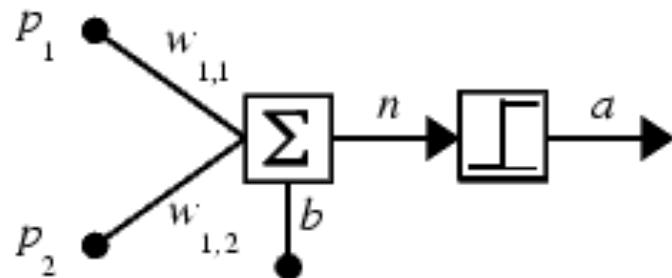
• Komenda tworząca nowy neuron liniowy. Obiekt jest zapisany do zmiennej net .

• Obliczenie co zwraca sieć (symulacja). Na razie sieć jest tylko zainicjalizowana – nie była uczona.

• Uczenie neuronu (z nauczycielem – T).

Przykład

(perceptron z dwoma wejściami, jednym wyjściem,
funkcja aktywacji progowa)



- Problem funkcji logicznej **OR**

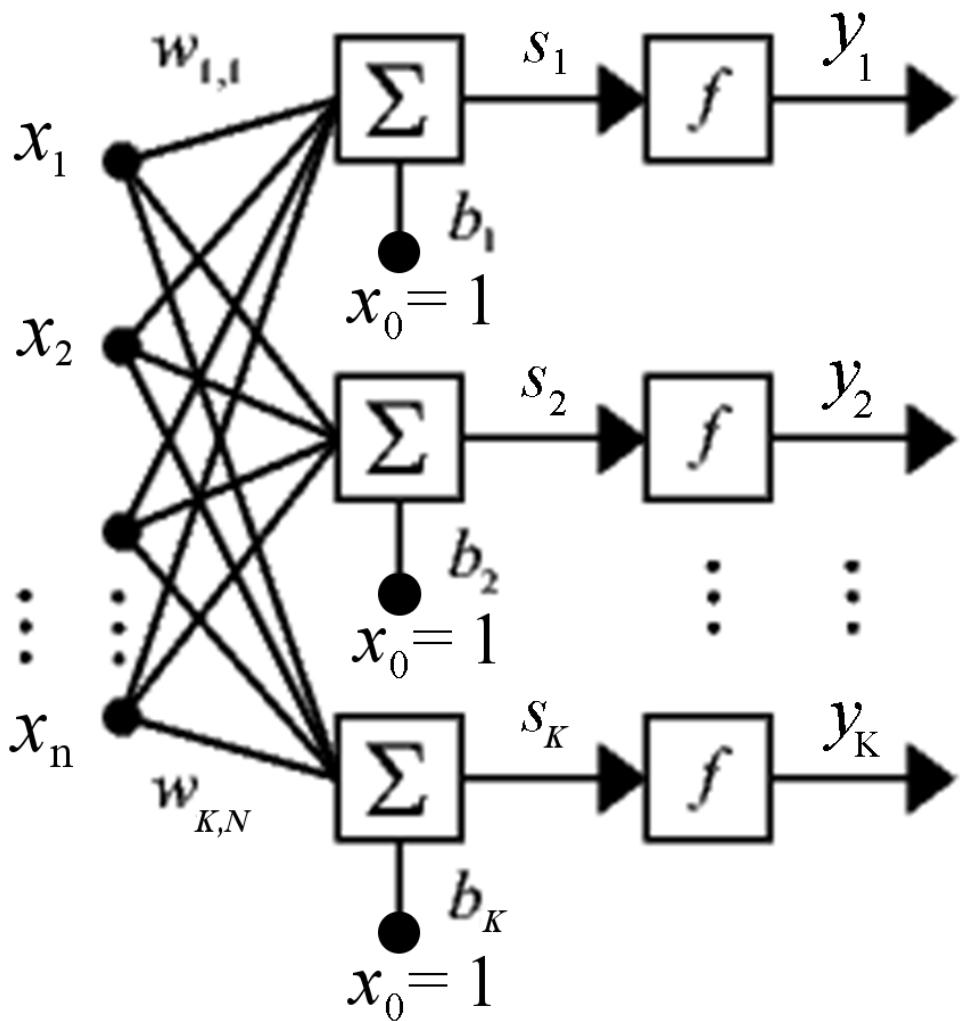
$$\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, 1\right), \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, 1\right), \left(\begin{bmatrix} 0 \\ 1 \end{bmatrix}, 1\right), \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, 0\right).$$

Komendy MATLAB-a

- Tym razem użyjemy perceptronu a nie neuronu liniowego. Dlatego funkcja tworząca sieć będzie newp.

- `net=newp([-2 2;-2 2],1);`
- `P=[1 1 0 0; 1 0 1 0];`
- `T=[1 1 1 0];`
- `net=train(net,P,T)`
- `sim(net,P)`

• Jednowarstwowa i jednokierunkowa sieć



- $N = \text{liczba wejścia}$
- $K = \text{liczba wyjścia}$

$$s_k = \sum_{i=0}^N w_{k,i} x_i, \\ y_k = f(s_k) \quad \text{dla } k = 1, K, K.$$

- Dla wygody progi (b_i) włączamy w powyższej sumie jako wagi odpowiadające impulsowi $x_0 = 1$. Tak więc $w_{k,0} = b_k$.

- Jednowarstwowa sieć jest opisana przez *macierz wag*, $W=[w_{i,j}]$:

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & K & w_{1,N} \\ w_{2,1} & w_{2,2} & K & w_{2,N} \\ M & M & & M \\ w_{K,1} & w_{K,2} & K & w_{K,N} \end{bmatrix},$$

- gdzie $w_{i,j}$ jest wagą pomiędzy j -tym wejściem a i -tym neuronem.

- Czasami – dla wygody i jednolitości opisu – macierz wag może zawierać także progi. Będą one wtedy umieszczone w macierzy jako pierwsza kolumna:

$$W = \begin{bmatrix} w_{1,0} & w_{1,1} & w_{1,2} & K & w_{1,K} \\ w_{2,0} & w_{2,1} & w_{2,2} & K & w_{2,K} \\ M & M & M & & M \\ w_{N,0} & w_{N,1} & w_{N,2} & K & w_{N,K} \end{bmatrix}.$$

- Jednowarstwowa sieć perceptronowa

- Sieć jednowarstwową jednokierunkową uczymy metodą analogiczną do algorytmu, które opisywaliśmy dla pojedynczego perceptronu. Jak pamiętamy do uczenia musimy mieć dany ciąg uczący

$$(x^{(1)}, d^{(1)}), K, (x^{(T)}, d^{(T)}),$$

- gdzie $x^{(t)}$ oraz $d^{(t)}$ są w ogólnym przypadku wektorami:

$$x^{(t)} = N \text{ elementów}, d^{(t)} = K \text{ elementów}.$$

- Przebieg uczenia jest następujący: podajemy na wejście sieci kolejny wektor wejściowy $x^{(t)}$, obliczamy wartości na wyjściu y_1, \dots, y_K , a następnie porównujemy z wartościami których oczekujemy, czyli $d^{(t)} = [d^{(t)}_1, \dots, d^{(t)}_K]$. Na tej podstawie modyfikujemy wagę (patrz dalej).

- Jeżeli $x=[x_1, \dots, x_N]$ oraz $d=[d_1, \dots, d_K]$ oznaczają aktualny wektor uczący i wektor żądanego odpowiedzi odpowiednio, to korekta wg algorytmu perceptronowego, przy założeniu, że funkcja aktywacji jest unipolarna ($f(s)=1$ lub 0)

for $i = 1, K, N$

for $j = 1, K, K, \{$

$$\Delta w_{i,j} = \eta(d_i - y_i)x_i$$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

}

- Tak naprawdę chodzi to o to, że modyfikacja jest tylko wtedy, gdy $d_i \neq y_i$. W tym przypadku wykonujemy $w + \eta x$ lub $w - \eta x$.

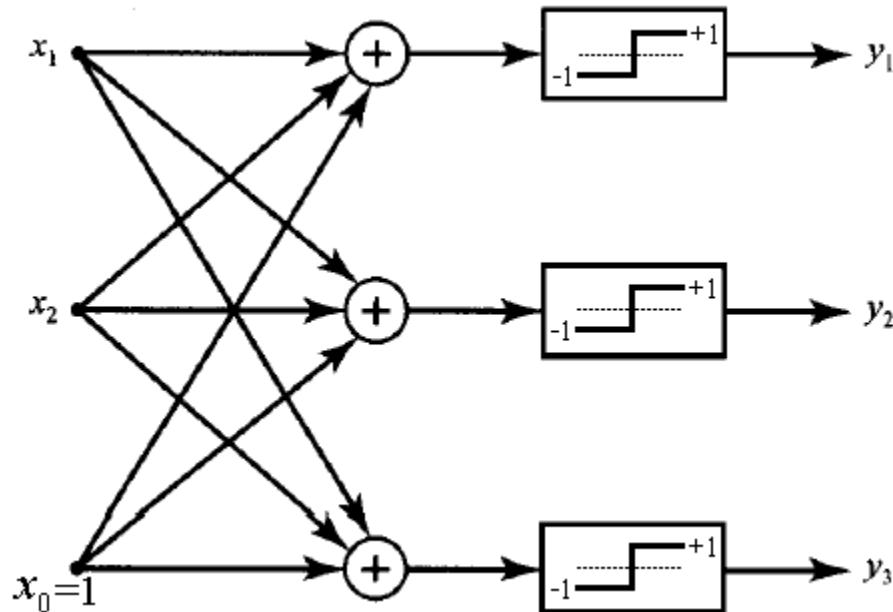
- Zauważmy, że gdy w perceptronie jako funkcji aktywacji używamy funkcji bipolarnej ($f(s)=+1$ lub -1), to formuła na modyfikację, $\Delta w_{i,j}$, będzie następująca

$$\Delta w_{i,j} = \frac{1}{2}\eta(d_i - y_i)x_i$$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

- Przykład

- Rozważmy sieć jednowarstwową perceptronów jak na rys. poniżej. Składa się ona z $N=2$ normalnych wejść ($x_0=1$ odpowiada progowi) oraz $K=3$ wyjść. W przykładzie użyto funkcji bipolarnej ($f(s)=\pm 1$).



- Ciąg uczący składa się z następujących par

$$(x^{(1)}, d^{(1)}) = \left(\begin{bmatrix} 10 \\ 2 \end{bmatrix}, \begin{bmatrix} +1 \\ -1 \\ -1 \end{bmatrix}\right), (x^{(2)}, d^{(2)}) = \left(\begin{bmatrix} 2 \\ -5 \end{bmatrix}, \begin{bmatrix} -1 \\ +1 \\ -1 \end{bmatrix}\right), (x^{(3)}, d^{(3)}) = \left(\begin{bmatrix} -5 \\ 5 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix}\right).$$

- Jeżeli dla wygody dołączymy do wektorów wejściowych wartość sygnału dla progu ($x_0=1$), to ciąg będzie miał postać

$$(x^{(1)}, d^{(1)}) = \left(\begin{bmatrix} 1 \\ 10 \\ 2 \end{bmatrix}, \begin{bmatrix} +1 \\ -1 \\ -1 \end{bmatrix}\right), (x^{(2)}, d^{(2)}) = \left(\begin{bmatrix} 1 \\ 2 \\ -5 \end{bmatrix}, \begin{bmatrix} -1 \\ +1 \\ -1 \end{bmatrix}\right), (x^{(3)}, d^{(3)}) = \left(\begin{bmatrix} 1 \\ -5 \\ 5 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix}\right).$$

- Macierz wag (zwykła i rozszerzona o prógi) na początku

$$\begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} \rightarrow \begin{bmatrix} w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \\ w_{3,0} & w_{3,1} & w_{3,2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & -2 \\ -2 & 0 & -1 \\ 1 & 1 & 3 \end{bmatrix}.$$

• Krok 1.

• Bierzemy parę

$$x^{(1)} = \begin{bmatrix} 1 \\ 10 \\ 2 \end{bmatrix}, \quad d^{(1)} = \begin{bmatrix} +1 \\ -1 \\ -1 \end{bmatrix},$$

• i obliczamy:

$$s = w_1 \text{ o } x^{(1)} = [w_{1,0} \quad w_{1,1} \quad w_{1,2}] \cdot \begin{bmatrix} 1 \\ 10 \\ 2 \end{bmatrix} = [0 \quad 1 \quad -2] \cdot \begin{bmatrix} 1 \\ 10 \\ 2 \end{bmatrix} = 0 \cdot 1 + 1 \cdot 10 + (-2) \cdot 2 = 6,$$

$$y_1 = f(s) = \text{sgn}(6) = 1.$$

$$s = w_2 \text{ o } x^{(1)} = [w_{2,0} \quad w_{2,1} \quad w_{2,2}] \cdot \begin{bmatrix} 1 \\ 10 \\ 2 \end{bmatrix} = [-2 \quad 0 \quad -1] \cdot \begin{bmatrix} 1 \\ 10 \\ 2 \end{bmatrix} = (-2) \cdot 1 + 0 \cdot 10 + (-1) \cdot 2 = -4,$$

$$y_2 = f(s) = \text{sgn}(-4) = -1.$$

$$s = w_3 \text{ o } x^{(1)} = [w_{3,0} \quad w_{3,1} \quad w_{3,2}] \cdot \begin{bmatrix} 1 \\ 10 \\ 2 \end{bmatrix} = [1 \quad 1 \quad 3] \cdot \begin{bmatrix} 1 \\ 10 \\ 2 \end{bmatrix} = 1 \cdot 1 + 1 \cdot 10 + 3 \cdot 2 = 17,$$

$$y_3 = f(s) = \text{sgn}(17) = 1.$$

- Mamy więc dla wektora $x^{(1)}$ następujące wartości na wyjściach

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} +1 \\ -1 \\ +1 \end{bmatrix}.$$

- Jeżeli porównamy teraz ten wynik z oczekiwany, czyli $d(1)$:

$$d^{(1)} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} +1 \\ +1 \\ -1 \end{bmatrix},$$

- to widzimy, że w tym kroku tylko wagi prowadzące do neuronu numer 3 będą zmienione

$$w_1 \leftarrow w_1, \quad w_2 \leftarrow w_2,$$

$$w_3 \leftarrow w_3 + \frac{1}{2}(d_3 - y_3)x^{(1)} = w_3 - x^{(1)} = [1 \quad 1 \quad 3] - [1 \quad 10 \quad 2] = [0 \quad -9 \quad 1].$$

- Ostatecznie po pierwszym kroku („przepuszczenie” przez sieć pierwszego wektora z ciągu uczącego) macierz wag (zawierająca w pierwszej kolumnie progi) ma postać

$$\begin{bmatrix} w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \\ w_{3,0} & w_{3,1} & w_{3,2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & -2 \\ -2 & 0 & -1 \\ 0 & -9 & 1 \end{bmatrix}.$$

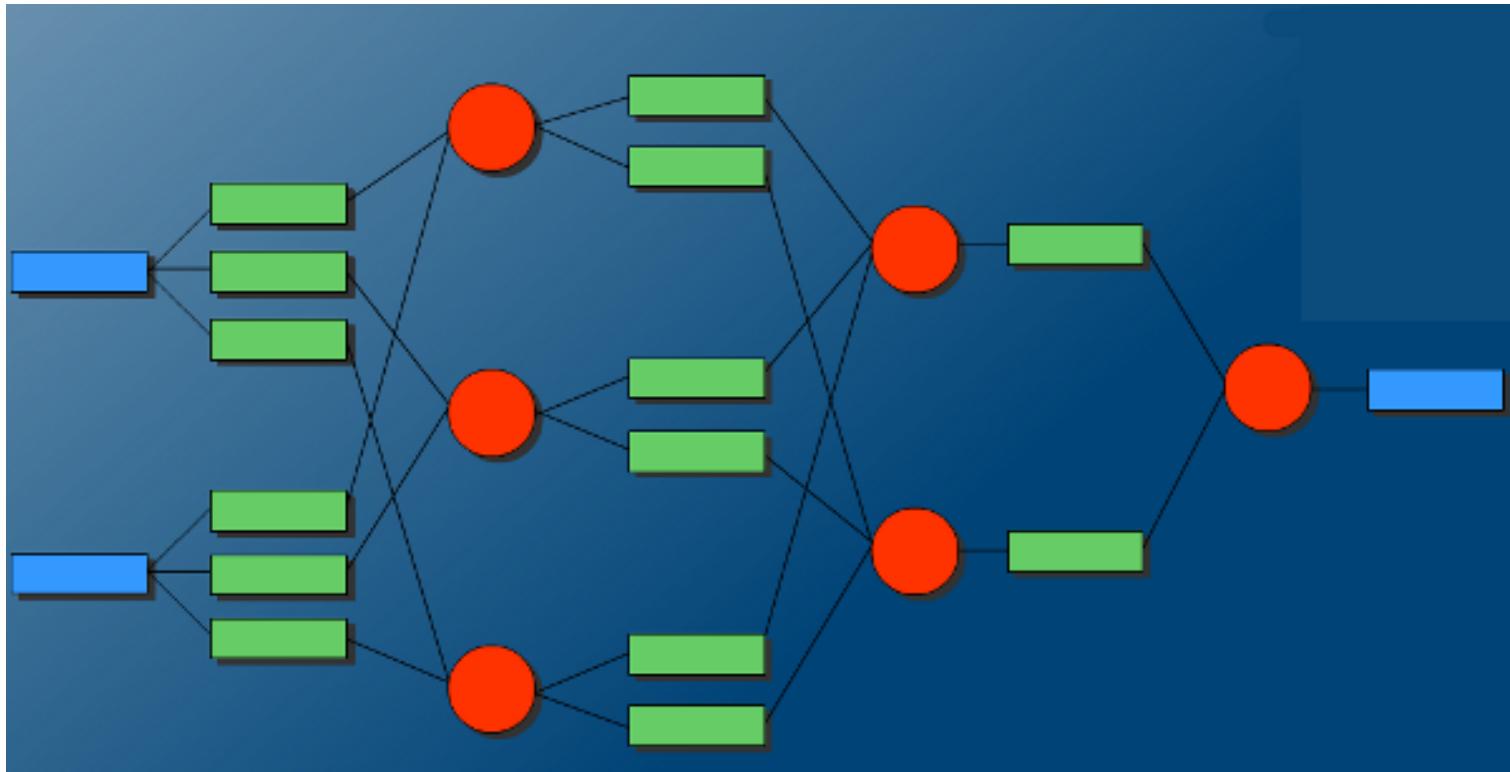
- Wykonując dalsze obliczenia (podając kolejne dwa wektory) i w razie potrzeby wykonując dalsze epoki otrzymujemy ostatecznie następującą macierz wag

$$\begin{bmatrix} w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \\ w_{3,0} & w_{3,1} & w_{3,2} \end{bmatrix} = \begin{bmatrix} -5 & 5 & 3 \\ -2 & 0 & -1 \\ 0 & -9 & 1 \end{bmatrix}.$$

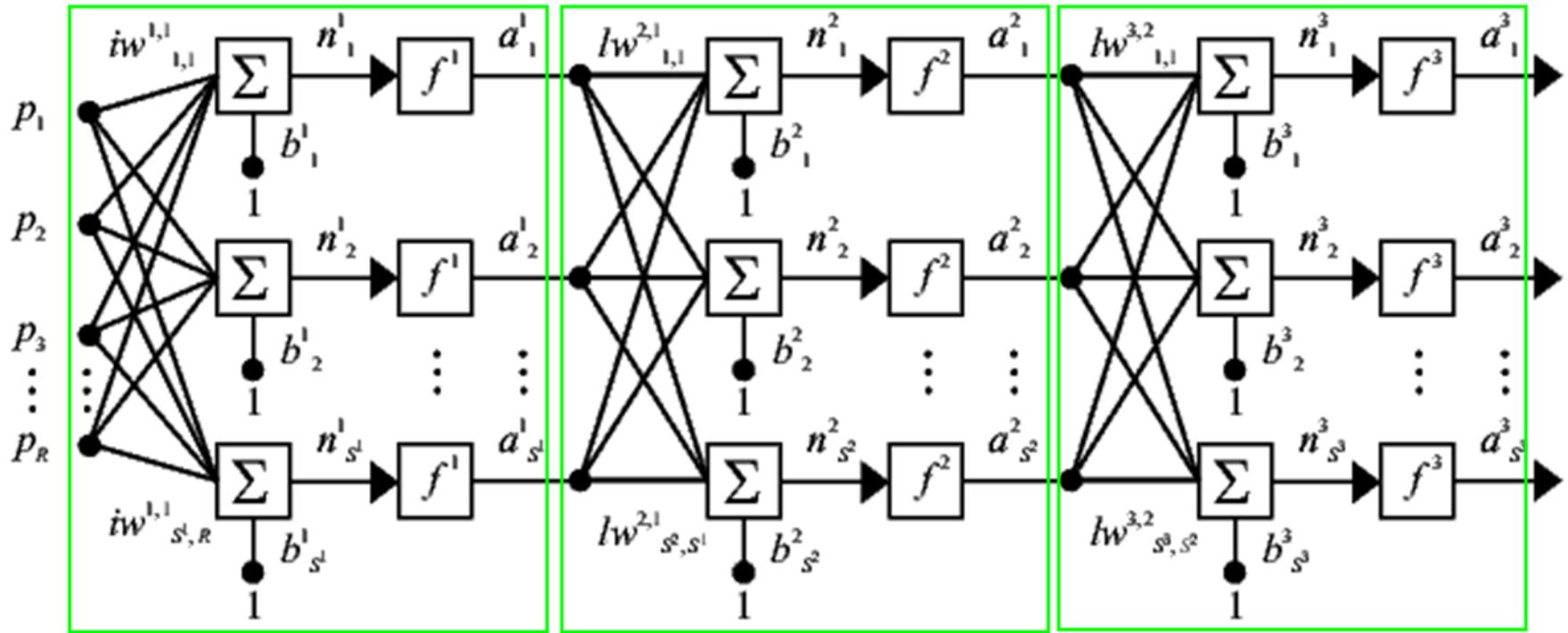
- (Pamiętajmy: pojedynczy wiersz o numerze i zawiera wagi dla połączeń prowadzących do neuronu o numerze i .)

- Sieci wielowarstwowe jednokierunkowe (ang. *multi-layer, feedforward*)
- Sieć wielowarstwową tworzą neurony ułożone w wielu warstwach, przy czym oprócz warstwy wejściowej i wyjściowej istnieje co najmniej jedna warstwa ukryta. Czasami sieć taka określa się mianem perceptronu wielowarstwowego.
- Teraz przeanalizujemy dokładniej szczególny przypadek – sieć o jednej warstwie ukrytej.

Sieć wielowarstwowa



- Przykładowy schemat sieci wielowarstwowej jednokierunkowej (ang. *multilayer feedforward network*)



- W zielonych ramkach zaznaczone są poszczególne warstwy: wejściowa, ukryta oraz wyjściowa. W tym przypadku jest jedna warstwa ukryta – ale w ogólności może ich być więcej.

•Oznaczenia

- $w_{ij}^{(1)}$ – •wagi w warstwie numer 1 (ukryta) dla połączenia: j -ty sygnał do i -tego neuronu w warstwie 2
- $w_{ij}^{(2)}$ – •wagi w warstwie numer 2 (wyjściowa) dla połączenie: od j -tego neuronu w warstwie 1 do i -tego w warstwie 2
- N – •liczba sygnałów wejściowych do sieci
- K – •liczba neuronów w warstwie ukrytej (warstwa 1)
- M – •liczba neuronów w warstwie wyjściowej (= liczba wyjść z sieci)
- (x_0, x_1, K, x_N) – •rozszerzony wektor wejść ($x_0=1$)
- (d_1, K, d_M) – •oczekiwane wyjście
- (y_1, K, y_M) – •wartości aktualnie generowane przez sieć na wyjściu
- (v_1, K, v_K) – •wartości aktualnie generowane przez warstwę ukrytą

- Podstawowa idea przy konstrukcji procedury uczenia sieci wielowarstwowej jednokierunkowej jest oparta o minimalizację funkcji błędu. Dla danej pary uczącej

$$\{(x_0, x_1, \dots, x_N), (d_1, \dots, d_M)\}$$

- Definiujemy błąd sieci

$$E(W) := \frac{1}{2} \sum_{k=1}^K (y_k - d_k)^2,$$

- gdzie $y = (y_1, \dots, y_K)$ są aktualnymi odpowiedziami sieci dla zadanego wektora wejściowego $x = (x_0, x_1, \dots, x_N)$. Funkcja błędu zależy w dość skomplikowany sposób od wszystkich wag:

$$W = \left\{ w_{i,j}^{(1)}, w_{i,j}^{(2)} \right\}.$$

- Na przykład aby obliczyć y_k stosujemy sygnały wyjściowe pierwszej warstwy v_1, \dots, v_M jako wejścia do drugie warstwy:

$$y_j = f \left(\sum_{k=0}^K w_{j,k}^{(2)} v_k \right), \quad v_k = f \left(\sum_{i=0}^N w_{k,i}^{(1)} x_i \right).$$

- Tak zdefiniowana funkcję błędu chcemy w kolejnych krokach minimalizować. Jedną z procedur jest metoda najszybszego spadku (prosa gradientowa), która wymaga policzenia gradientu funkcji $E(W)$, czyli w praktyce wszystkich pochodnych cząstkowych względem $w_{i,j}^{(1)}, w_{i,j}^{(2)}$ wzg

- Jeżeli następnie wstawimy te pochodne do metody najszybszego spadku, to otrzymamy procedurę modyfikowanie wag, która nazywana jest w sieciach neuronowych *metodą wstecznej propagacji błędu*.
- Podstawowa idea wstecznej propagacji błędów jest taka. Dla warstwy wyjściowej znamy korekty wag, gdyż znamy oczekiwane wyjście. Zatem

$$\delta_{k,\text{output}} : d_k - y_k$$

- Dla warstwy ukrytej nie znamy błędów, bo nie znamy oczekiwanych wartości na wyjściach neuronów z tej warstwy. Dlatego używamy średniej wartości błędów z warstwy wyjściowej z wagami, które odpowiadają połączeniom danego neuronu ukrytego z wszystkimi neuronami warstwy wyjściowej. Zatem

$$\delta_{i,\text{hidden}} : \sum_k w_{k,i} \delta_{k,\text{output}}$$

- Przykład
- (wycena wartości nieruchomości)

- Wykorzystamy przykładowe dane zapisane w plikach *houseInputs.txt* oraz *housePrices.txt*. Zawierają one dane na temat rynku nieruchomości. Parametry opisujące sprzedaną nieruchomość (*houseInputs.txt*) oraz cenę sprzedaży (*housePrices.txt*). Dane wejściowe zawierają 506 przykładowych wektorów, każdy o 13-tu elementach opisujących cechy danej nieruchomości. Z kolei tablica *housePrices* zawiera 506 elementów (macierz 1x506), które są wartościami nieruchomości dla odpowiadających im zestawów parametrów z tablicy *houseInputs*.

```
•P=load('houseInputs.txt','-ascii');
•T=load('houseOutputs.txt','-ascii');
•size(P);
•size(T);
•P(:,1), T(1,1)
•net = newff(P,T,20);
•net = train(net,P,T);
```

6. Algorytmy heurystyczne

6.1 Algorytm Quickprop

$$\Delta W_{ij}(k) = -\eta_k \left[\frac{\partial E(\mathbf{W}(k))}{\partial W_{ij}} + \gamma W_{ij}(k) \right] + \alpha_{ij}^{(k)} \Delta W_{ij}(k-1)$$

Czynnik γ redukuje wartości wag i nie pozwala na ich duży przyrost.
Typowa wartość to 10^{-4}

Czynnik η_k przyjmuje dwie wartości:

$$\eta_0 \in [0.01, 0.6]$$

Na starcie uczenia, lub gdy $\left[\frac{\partial E(\mathbf{W}(k))}{\partial W_{ij}} + \gamma W_{ij}(k) \right] \Delta W_{ij} > 0$

albo zero w przeciwnym wypadku

Istotną rolę pełni współczynnik momentu dobierany dla każdej wagi indywidualnie.

$$\alpha_{ij}^{(k)} = \begin{cases} \alpha_{\max} & \text{gdy } \beta_{ij}^{(k)} > \alpha_{\max} \\ \beta_{ij}^{(k)} & \text{wpw.} \end{cases} \quad \text{lub} \quad S_{ij}(k) \Delta W_{ij}(k-1) \beta_{ij}^{(k)} < 0$$

$$S_{ij}(k) = \frac{\partial E(\mathbf{W}(k))}{\partial W_{ij}} + \gamma W_{ij}(k)$$

$$\beta_{ij}^{(k)} = \frac{S_{ij}(k)}{S_{ij}(k-1) - S_{ij}(k)}$$

$$\alpha_{\max} = 1,75$$