Politechnika Warszawska

MSI

11. Sztuczne sieci neuronowe

Włodzimierz Kasprzak

Treść

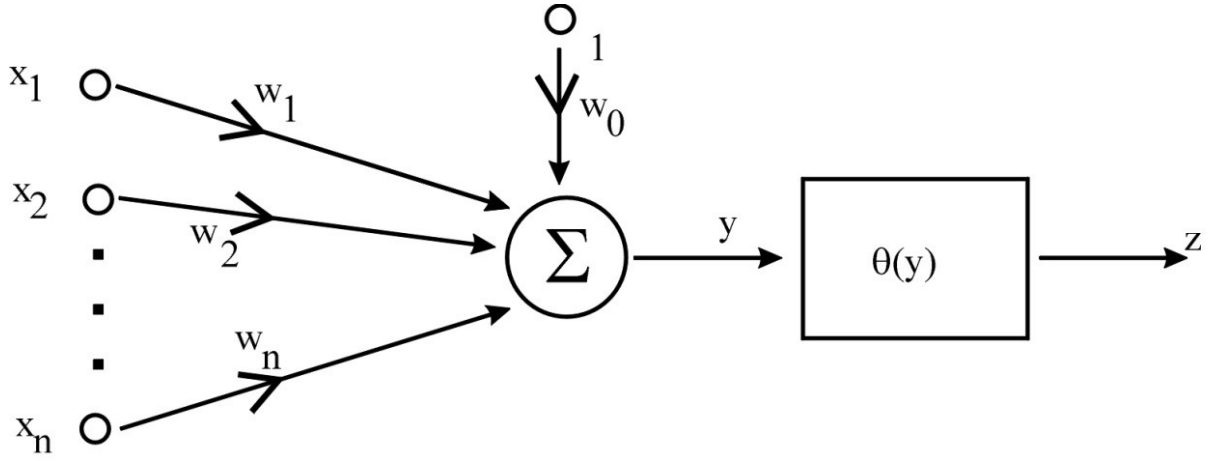
1. Wstęp
2. Wielowarstwowy perceptron MLP
3. Uczenie sieci neuronowych
4. Poprawianie generalizacji sieci
5. Hiperparametry
6. Neuronowe sieci rekurencyjne
7. Sieciowe modele klasycznych technik uczenia maszynowego

1. Wstęp

Sztuczne sieci neuronowe są dogodnym narzędziem stosowanym tam, gdzie podanie rozwiązania o analitycznej postaci funkcji jest trudne lub niemożliwe.

Dzięki algorytmom uczenia sieci na podstawie zadanego zbioru próbek możemy wyznaczyć sieć neuronową dobrze *aproksymującą* nieznaną postać funkcji.

Model pojedynczego neuronu (Rosenblatt, 1957)



Model neuronu

Wejścia: $(x_1,x_2,\ldots,x_n)$.

Wyjście: z.

Funkcja pobudzenia (wyjściowa): $y = \sum_{i=1}^{n} (w_i \cdot x_i) - w_0 \cdot 1$

gdzie w_0 jest wagą dodatkowego wejścia progowego.

Funkcja aktywacji: $z = \theta(y)$.

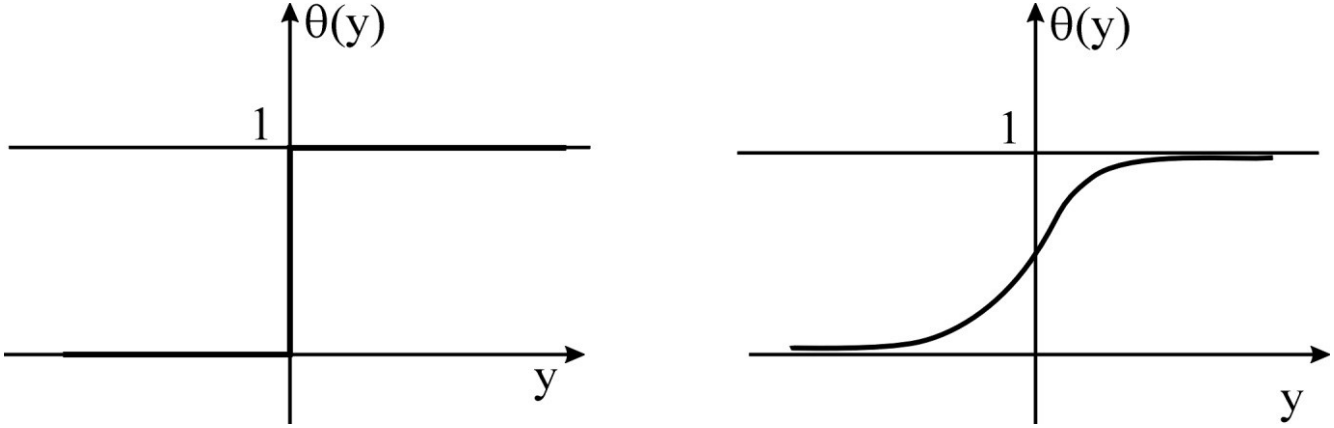
Proste funkcje aktywacji to:

1. Funkcja liniowa, $z = k\ y$, gdzie k jest zadanym stałym współczynnikiem.
2. Funkcja skoku jednostkowego: $z = \begin{cases} 1 & \text{gdy } y_i > y_T \\ 0 & \text{, przeciwnie} \end{cases}$

   gdzie y_T jest zadanym progiem.

Funkcje aktywacji neuronu

Najczęściej stosuje się funkcję aktywacji o ciągłej pochodnej: zamiast funkcji *skoku jednostkowego* (lewa strona) stosujemy funkcję *logistyczną* (*sigmoidalną*) (prawa strona):



- Funkcja *logistyczna* (*sigmoidalna unipolar.*): gdzie β jest zadanym parametrem. Wyjście neuronu przyjmuje wartości z przedziału $[0, 1]$.

$$\theta(y) = \frac{1}{1+\exp(-\beta y)}$$

- Funkcja *tangens hiperboliczny* (*bipolarna*): gdzie α jest zadanym parametrem. Wyjście przyjmuje wartości z przedziału $[-1, 1]$.

$$\theta(y) = \mathrm{tgh}(\frac{\alpha\, y}{2})$$

Sigmoidalna funkcja aktywacji

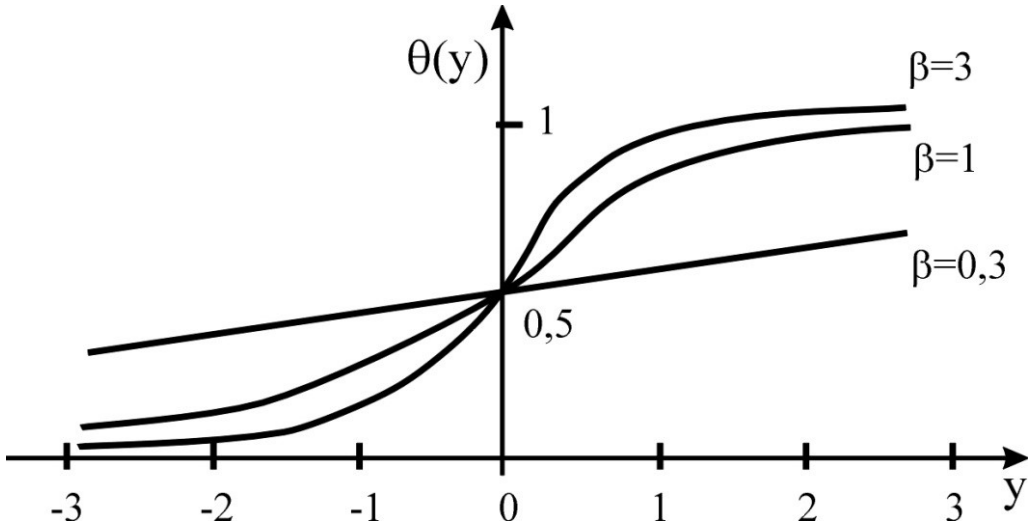
Dla *logistycznej* (*sigmoidalnej*) funkcji aktywacji o postaci:

$$z = \theta(y) = \frac{1}{1+\exp(-y)}$$

pochodna tej funkcji jest postaci:

$$\frac{\partial z}{\partial y} = z(1-z)$$

Wpływ wartości parametru β na kształt sigmoidalnej funkcji aktywacji:

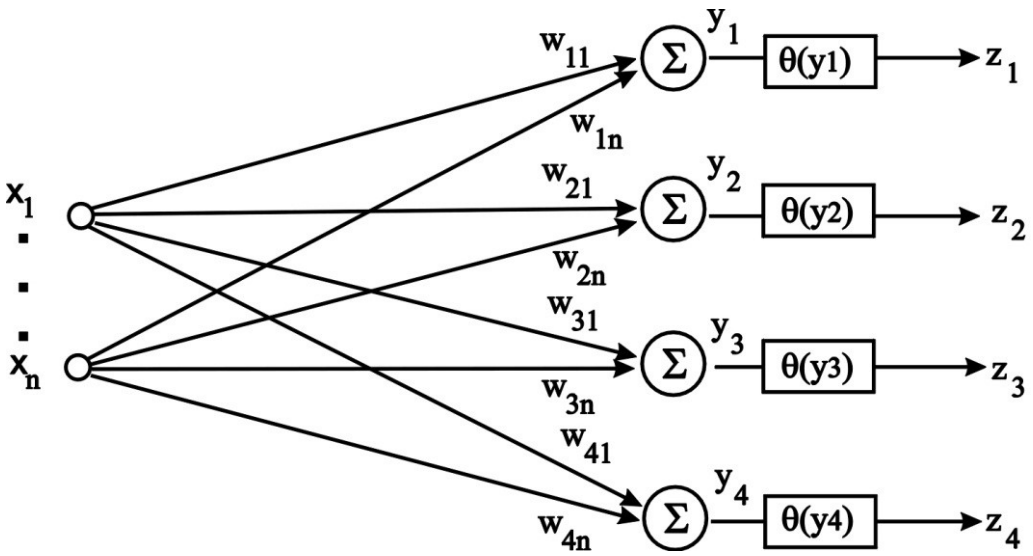


2. Wielowarstwowy perceptron

W sieci **jednowarstwowej** neurony ułożone są w jednej warstwie. W sieci typu **perceptron** (sieć **jednokierunkowa**, ang. „*feed-forward*”) sygnały wejściowe są przesyłane od wejścia do wyjścia poprzez połączenia **pobudzające**.

Połączenie wektora wejściowego z neuronami warstwy wyjściowej jest zwykle pełne co reprezentuje macierz wag połączeń $\mathbf{W}$. Funkcja pobudzenia neuronów warstwy wyjściowej:

$$\mathbf{y} = \mathbf{W} \cdot \mathbf{x}$$



Wielowarstwowy perceptron MLP

Wielowarstwowy perceptron (ang. *multilayer perceptron*, MLP) posiada warstwę wejściową, warstwę ukrytą (jedną, lub więcej) i neurony warstwy wyjściowej.

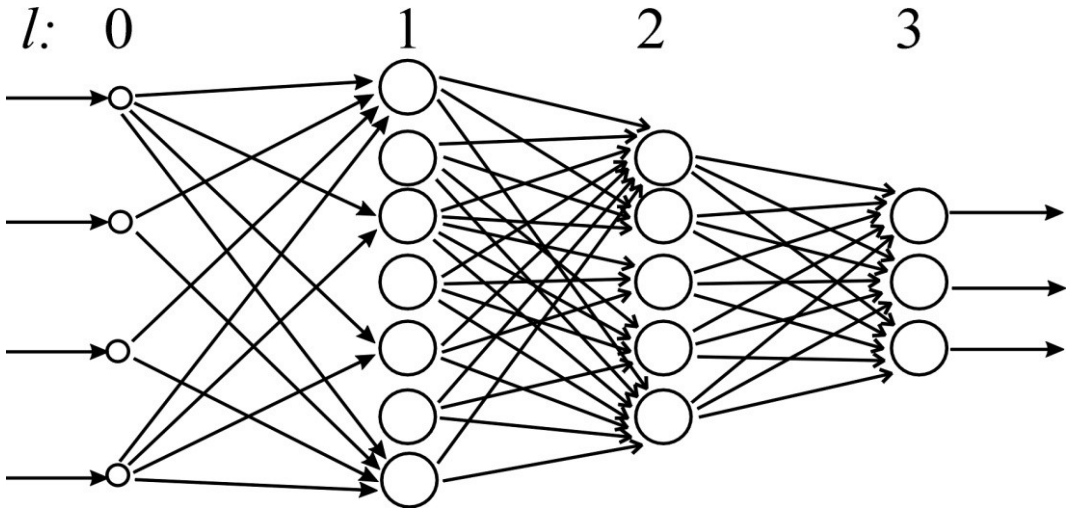
Funkcja każdej warstwy o indeksie l $(=0,1,2,\ldots)$:

$$\mathbf{z}^{(l)} = \theta(\mathbf{W}^{(l)}\mathbf{z}^{(l-1)} - \mathbf{w}_0^{(l)})$$

przy czym $\mathbf{z}(0) = \mathbf{x}$, $\theta(\mathbf{y})$ jest nieliniową funkcją aktywacji, a $\mathbf{w}_0(l)$ to wektor wag dodatkowych wejść progowych.

Struktura sieci MLP

o czterech warstwach:



MLP jako aproksymator funkcji

Oryginalna sieć perceptronu zastosowana została do aproksymacji funkcji binarnych, tzn. wyjście sieci zawierało aktywację: $\hat{y} = sign(W \cdot X)$, typową dla problemu klasyfikacji.

Jednak obecnie sieć MLP jest uniwersalnym aproksymatorem funkcji o wartościach rzeczywistych.

Można pokazać (Cybenko, 1989), że już 3-warstwowy perceptron (czyli jednej warstwie ukrytej) o sigmoidalnej funkcji aktywacji i n_{hidden} neuronach w warstwie ukrytej, przy $n_{\text{hidden}} \rightarrow \infty$ (dowolnie duża liczba neuronów) może:

- aproksymować dowolne zbiory w przestrzeni $\Re^n$, albo
- dowolną funkcję ciągłą zdefiniowaną w tej przestrzeni.

Funkcje logiczne

Pojedynczy neuron może m.in. realizować liniowe funkcje logiczne.

Neuron realizuje funkcję iloczynu AND, gdy ustalimy dla niego wagi: $w_0 = 1.5$, $w_1=1$, $w_2=1$, a jako funkcję aktywacji wybierzemy funkcję skoku.

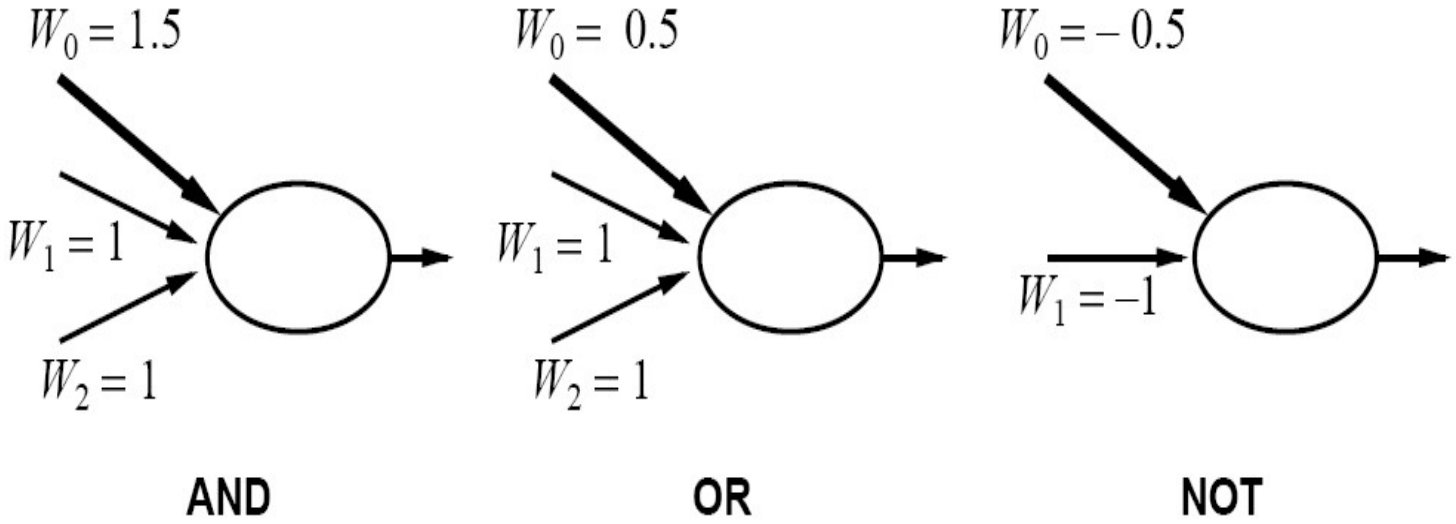
Neuron realizuje funkcję sumy OR, gdy ustalimy dla niego wagi: $w_0 = 0.5$, $w_1=1$, $w_2=1$, a jako funkcję aktywacji wybierzemy funkcję skoku.

Neuron realizuje funkcję negacji NOT, gdy ustalimy dla niego wagi: $w_0= -0.5$, $w_1= -1$, a jako funkcję aktywacji wybierzemy funkcję skoku.

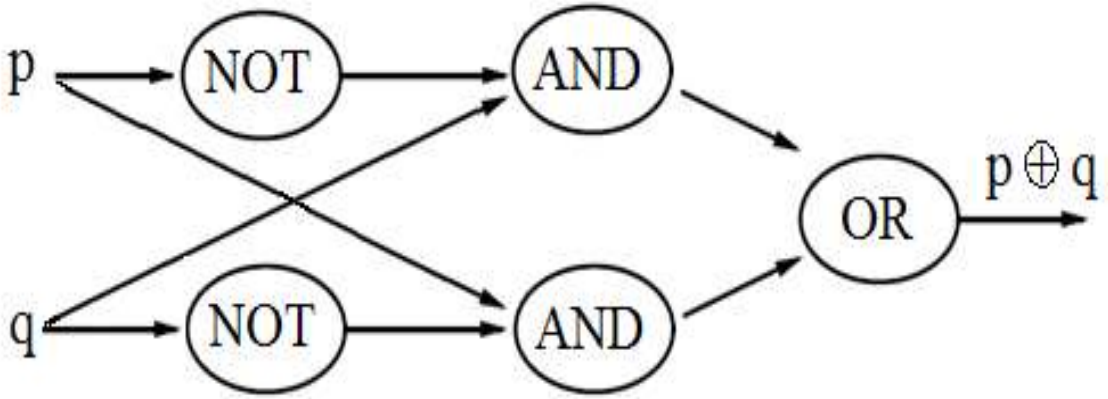
Prosta sieć neuronowa jednokierunkowa może m.in. realizować funkcję XOR.

Funkcje logiczne (c.d.)

- Neurony dla funkcji AND, OR, NOT



- Sieć dla funkcji XOR



Parametry sieci

Rozważmy sieć o jednym wejściu x, dwóch neuronach ukrytych n_1 i n_2 i jednym neuronie wyjściowym n_3 o liniowym wyjściu y_3. Nieliniowa funkcja aktywacji $\theta(y)$ jest postaci tangensa hiperbolicznego. Czyli aproksymujemy funkcję $f: R \rightarrow R$. Opisuje ją wzór:

$$z_3 = y_3 = w_{30} + w_{31} \cdot \theta(w_{10} + w_{11}x) + w_{32} \cdot \theta(w_{20} + w_{21}x)$$

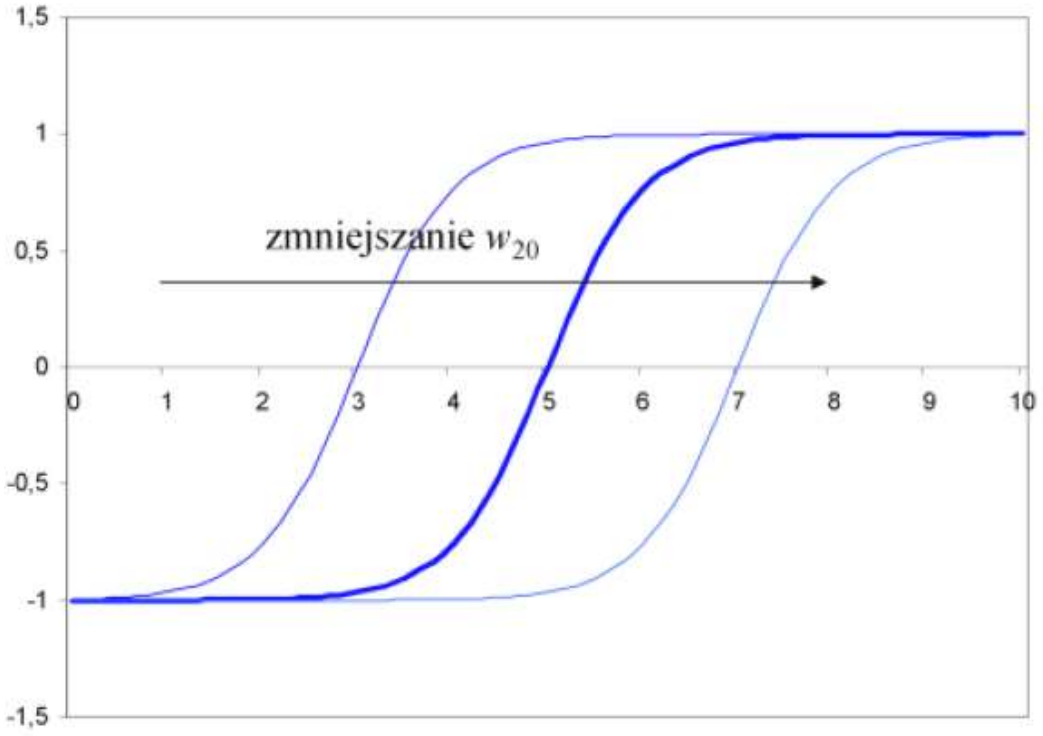
Znaczenie parametrów pojedynczego neuronu warstwy ukrytej jest następujące:

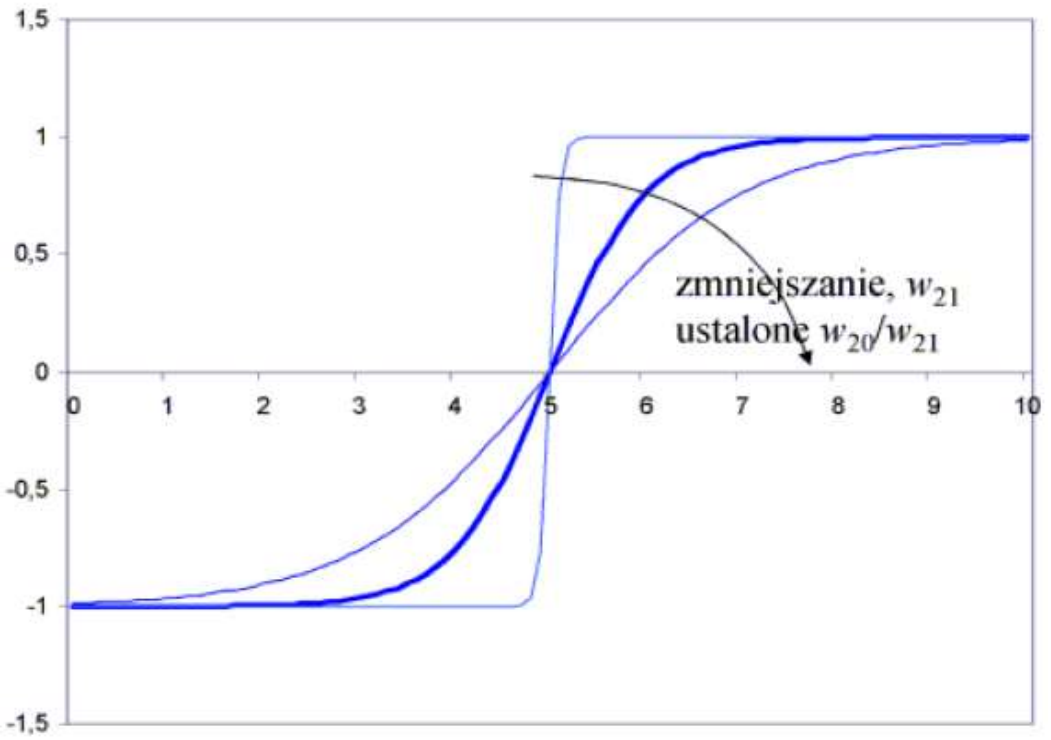
- wartości w_{10}/w_{11}, w_{20}/w_{21} służą do przesuwania wykresu funkcji $\theta(.)$ wzdłuż osi rzędnych.
- parametry w_{11}, w_{21} wpływają na „stromość” wykresu funkcji $\theta(y_1)$, $\theta(y_2)$.

Parametry warstwy ukrytej

Wartości w_{10}/w_{11}, w_{20}/w_{21} służą do przesuwania wykresu funkcji $\theta(.)$ wzdłuż osi rzędnych (ilustracja lewa).

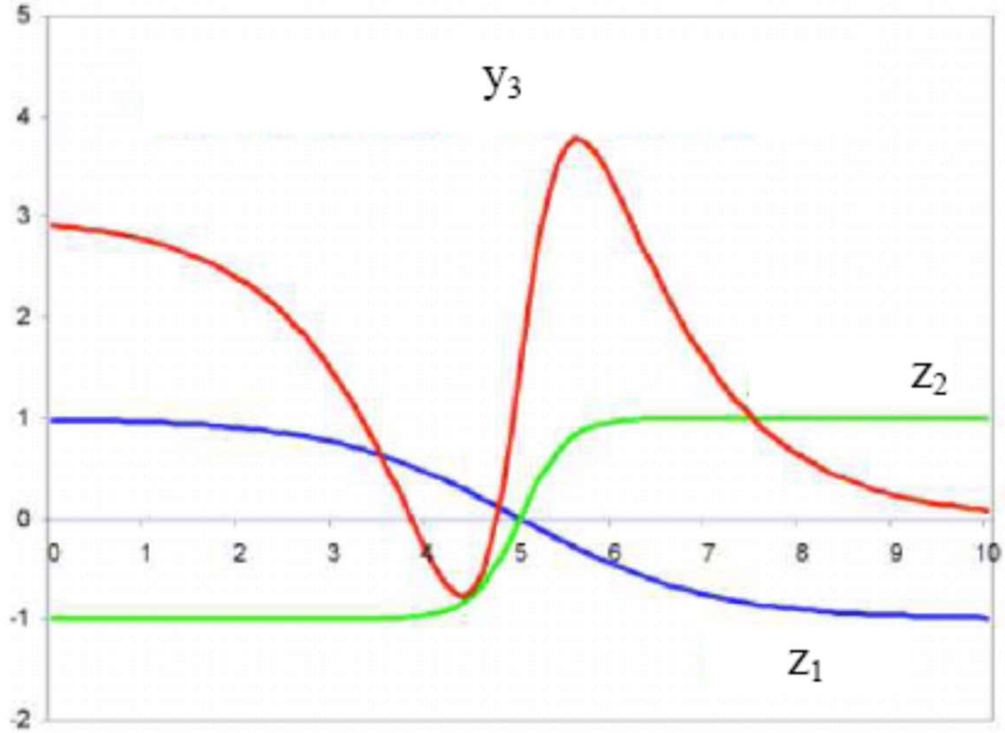
Parametry w_{11}, w_{21} wpływają na „stromość” wykresu funkcji $\theta(y_1)$, $\theta(y_2)$ (ilustracja prawa).





Parametry warstwy wyjściowej

Parametry warstwy wyjściowej służą określeniu stopnia wymieszania wyjść neuronów ukrytych. Im większa wartość wagi w_{31}, wzgl. w_{32} tym większy mnożnik użyty do wykresu wyjścia neuronu 1-szego wzgl. 2-ego.



3. Uczenie sieci („trening”)

Najczęściej spotykane kryteria w uczeniu sieci neuronowych polegają na minimalizacji **funkcji błędu wyjść** wyrażonej jako:

1. Błąd średniokwadratowy pomiędzy oczekiwanym (zadanym) wynikiem a wynikiem obserwowanym (typowa dla aproksymacji funkcji, np. **siecią MLP**);
2. Entropia krzyżowa z funkcją logistyczną – typowa dla wielokrotnej klasyfikacji za pomocą niezależnych od siebie wielu klasyfikacji binarnych;
3. Softmax z entropią krzyżową między rozkładem prawdopodobieństwa pożądanym a rzeczywistym – typowa dla wieloklasowej klasyfikacji (tzw. one-hot encoding).

- Podstawowym algorytmem uczenia wag sieci jest algorytm wstecznej propagacji błędu („error backpropagation”)

3.1 Uczenie sieci MLP

Uczenie perceptronu polega na optymalizacji wartości funkcji straty, $E(\mathbf{W})$, czyli minimalizacji średniego błędu kwadratowego dla zbioru próbek uczących.

Poszukiwana jest minimalna wartość tego błędu iteracyjną metodą *spadku w kierunku gradientu* $\nabla E(\mathbf{W})$ (ang. *gradient descent*).

W każdej kolejnej iteracji t wyznaczana jest chwilowa wartość błędu $E(\mathbf{W}, t)$ i wyznaczane są gradienty tej funkcji względem aktualnych wartości wag.

Jedna iteracja może obejmować pojedynczą próbkę uczącą lub podzbiór (tzw. *batch*) próbek. W drugim przypadku wyznaczany jest średni gradient dla aktualnego podzbioru.

Wyznaczone ostatecznie wartości wag sieci odpowiadają minimum funkcji $E(\mathbf{W})$ (minimum globalnemu w przypadku sieci liniowej lub najczęściej lokalnemu minimum dla sieci nieliniowej).

Średni błąd kwadratowy

Niech dla wektora wejść $\mathbf{x}_t$ oczekiwany (prawidłowy) wektor wyjściowy warstwy wynosi $\mathbf{s}_t$.

Miarą jakości sieci jest suma kwadratów błędów wyjść, $E(\mathbf{W})$ (błędem jest różnica między rzeczywistą a żądaną wartością), uśredniona dla zbioru treningowego (może być wyznaczana dla podzbioru uczącego, walidacyjnego lub testującego):

$$E(\mathbf{W}) = \frac{1}{N_{\mathrm{T}}} \sum_{t \in \mathrm{T}} \sum_{i} e_{t,i}^{\ 2} = \frac{1}{N_{\mathrm{T}}} \sum_{t \in \mathrm{T}} \sum_{i} (s_{t,i} - z_{t,i}(y_i(\mathbf{x}_t)))^2$$

Minimalizacja błędu średniokwadratowego - poszukiwanie minimalnej wartość tego błędu iteracyjną metodą *w kierunku spadku gradientu* $\nabla E(\mathbf{W})$ (ang. *gradient descent*):

$$w_{ij}(t+1) = w_{ij}(t) - \eta(t) \frac{\partial E(\mathbf{W}(t))}{\partial w_{ij}}$$

Reguła modyfikacji wag sieci

$$\frac{\partial E}{\partial \boldsymbol{w}_{ij}} = \frac{\partial E}{\partial z_i}\frac{\partial z_i}{\partial y_i}\frac{\partial y_i}{\partial \boldsymbol{w}_{ij}} = -2(s_i - z_i)\frac{\partial z_i}{\partial y_i}x_j$$

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)(s_i - z_i)\frac{\partial z_i}{\partial y_i}x_j$$

Jest to iteracyjna reguła modyfikacji wag sieci będąca uogólnieniem tzw. reguły „delta" zaproponowanej dla sieci perceptronu o binarnym wyjściu.

Reguła Widrowa-Hoffa (reguła „delta")

Według niej waga połączenia *j*-tego wejścia z *i*-tym (neuronem) wyjściowym jest wzmacniana proporcjonalnie do różnicy pożądanej i rzeczywistej aktywacji oraz wartości wejścia:

$$\Delta w_{ij} = \eta(s_i - z_i)x_j$$

gdzie s_i to pożądana aktywacja *i*-tego wyjścia.

Uczenie sieci MLP (1)

W postaci macierzowej **reguła modyfikacji wag** warstwy l:

$$\mathbf{W}^{(l)}(t+1) = \mathbf{W}^{(l)}(t) + \eta \cdot \Delta^{(l)} \cdot \left[\boldsymbol{x}^{(l)}\right]^T, \quad \boldsymbol{x}^{(1)} = \boldsymbol{x}$$

gdzie $\boldsymbol{x}^{(l)} = z^{(l-1)}$, $l = 1, 2, \ldots, L$; a $\Delta^{(l)}$ jest wektorem korekcji pobudzenia neuronów l-tej warstwy.

Modyfikacja wag rozpoczyna się od ostatniej warstwy ($l = L$) i przemieszcza się wstecz warstwa po warstwie, aż zakończy się na warstwie o indeksie l=1.

Podstawą tego procesu są propagowane „wstecz” wartości korekty, obliczone początkowo dla najwyższej warstwy ($l = L,\ldots,1$):

$$\Delta^{(L)} = (\mathbf{s} - \mathbf{z}^{(L)}).* \left[\frac{\partial z}{\partial y}\right] \qquad \Delta^{(l-1)} = (\mathbf{W}^{(l)})^T \Delta^{(l)}.* \left[\frac{\partial z}{\partial y}\right]$$

Uczenie sieci MLP (2)

Uwaga: symbol .* we wzorach na korektę wag oznacza mnożenie Hadamarda – mnożenie elementu pierwszego wektora z odpowiednim elementem drugiego wektora (element-po-elemencie).
W szczególności dla logistycznej funkcji aktywacji

$$\mathrm{z} = \theta(y) = \frac{1}{1 + \exp(-y)}$$

wartości korekty, dla $l = L,\ldots,1$, obliczamy jako:

$$\Delta^{(L)} = (\mathbf{s} - \mathbf{z}^{(L)}).* \left[1 - \mathbf{z}^{(L)}\right].* \mathbf{z}^{(L)}$$

$$\Delta^{(l-1)} = (\mathbf{W}^{(l)})^T \Delta^{(l)}.* \left[1 - \mathbf{z}^{(l-1)}\right].* \mathbf{z}^{(l-1)}$$

3.2 Entropia krzyżowa

W problemie klasyfikacji dla dwóch klas (K_1, K_2) zazwyczaj stosowane jest pojedyncze wyjście z funkcją sigmoidalną a jako miara błędu – entropia krzyżowa.

Podejście to może być rozszerzone na wiele wyjść, z których każde jest wynikiem niezależnie wykonanej klasyfikacji binarnej. Odpowiada to sytuacji, gdy próbka wejściowa może jednocześnie należeć do różnych klas.

Niech klasa K_1 będzie klasą „*pozytywną*” – wtedy wyjście sieci „z” stanowi aproksymację prawdopodobieństwa klasy K_1 dla wejścia sieci $\boldsymbol{x}$ przy wagach $\boldsymbol{W}$, a dopełnienie do 1, wartość prawdopodobieństwa klasy K_2:

$$z = P(K_1|\boldsymbol{x}, \boldsymbol{W}); \qquad (1-z) = P(K_2|\boldsymbol{x}, \boldsymbol{W})$$

Problem klasyfikacji binarnej opisany jest zmienną losową s o rozkładzie Bernoulliego o dwóch realizacjach $s \in \{0, 1\}$:

$$P(s|\boldsymbol{x}, \boldsymbol{W}) = z^s \cdot (1-z)^{1-s}$$

Entropia krzyżowa

Entropia krzyżowa (cross-entropy).

W teorii informacji jest to miara różnicy dwóch rozkładów prawdopodobieństwa, np. pożądanego $\mathbb{R}$ i rzeczywistego $\mathbb{Z}$. W przypadku klasyfikacji binarnej pierwszy rozkład dany jest pożądaną wartością wyjścia, $S = [s, 1-s]$, a drugi – rzeczywistą wartością $\mathbb{Z} = [z, 1-z]$:

$$\mathcal{L}(S, \mathbb{Z}) = -\sum_{i=1}^{2} S_i \cdot \ln \mathbb{Z}_i \quad = -(s \ln z + (1-s) \ln(1-z))$$

Celem procesu uczenia jest minimalizacja średniej wartości entropii krzyżowej dla zbioru próbek uczących.

Gradient entropii krzyżowej względem wag warstwy wyjściowej w połączeniu z logistyczną (sigmoidalną) funkcją aktywacji posiada ciekawą własność – zanika w nim zależność od pochodnej funkcji sigmoidalnej.

Gradient entropii krzyżowej

Wyznaczmy pochodną funkcji „straty” (entropii krzyżowej) względem wyjścia z:

$$\frac{\partial \mathcal{L}}{\partial z} = -\frac{s}{z} + \frac{1-s}{1-z} = \frac{-(1-z)s + z(1-s)}{z(1-z)} = \frac{(z-s)}{z(1-z)}$$

Wyznaczamy pochodną funkcji „straty” względem pojedynczej wagi warstwy wyjściowej z sigmoidalną funkcją aktywacji :

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial w_j} = \frac{(z-s)}{z(1-z)} \cdot z(1-z) \cdot x_j = (z-s) \cdot x_j$$

Modyfikacja wartości wag neuronu wyjściowego sieci wynosi:

$$\Delta \boldsymbol{w}^{(L)} = -\eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^{(L)}} = (s-z)\boldsymbol{x}^{(L)},$$

czyli sama „korekta”, $\Delta^{(L)} = (s-z)$

3.3 Funkcja „softmax”

Jest to funkcja aktywacji odwzorowująca wektory iczb, która zapewnia normalizację wyjść sieci dla N **klas**. Normalizacja polega na tym, że suma wartości wyjść wynosi 1. Pozwala to interpretować znormalizowane wyjścia (np. s_k) jako wartości prawdopodobieństwa klas dla jednej zmiennej losowej (s) - próbka wejściowa (x) może należeć jedynie do **jednej** klasy.

Funkcja softmax: $s_k = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)}$, gdzie

$$z_k = (y_k) = \left(\sum_{i=1}^{N} w_{ki} x_i + w_{k0} \right)$$

„Bezpieczna” forma funkcji softmax – zabezpieczająca przed nadmiernymi wartościami funkcji wykładniczej i powstaniem błędu nadmiaru (overflow): $s_k = \frac{\exp(z_k - \max(z))}{\sum_{j=1}^{K} \exp(z_j - \max(z))}$

Gradient entropii krzyżowej (1)

W procesie uczenia sieci potrzebna jest pochodna funkcji *softmax.* Wprawdzie nie zawiera ona parametrów wymagających uczenia ale odpowiada za propagację funkcji straty do niższych warstw sieci.

Wejście ($\mathbf{z}$) jak i wyjście warstwy softmax ($\mathbf{s}$) są wektorami o długości K (liczba klas) – stąd pochodne cząstkowe tworzą macierz:

$$\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{z}} = \begin{bmatrix} \frac{\partial s_1}{\partial z_1} & \cdots & \frac{\partial s_1}{\partial z_K} \\ \vdots & \ddots & \vdots \\ \frac{\partial s_K}{\partial z_1} & \cdots & \frac{\partial s_K}{\partial z_K} \end{bmatrix}$$

Oznaczmy: $s_k(\mathbf{z}) = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)} = \frac{f}{g}$.

Wiemy, że $\frac{\partial}{\partial z}\left(\frac{f}{g}\right) = \frac{\frac{\partial f}{\partial z}g - \frac{\partial g}{\partial z}f}{g^2}$. Stąd: $\frac{\partial s_k}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\frac{f}{g}\right) = \frac{\frac{\partial f}{\partial z}g - \frac{\partial g}{\partial z}f}{g^2}$

Gradient entropii krzyżowej (2)

Oznaczmy przez δ_{ki} deltę Kroneckera:

$$\delta_{ki} = 1\ (k = i)\,, \qquad \delta_{ki} = 0\ (k \neq i)$$

$$\frac{\partial f_k}{\partial z_i} = \frac{\partial(\exp(z_k))}{\partial z_i} = \exp(z_k)\frac{\partial z_k}{\partial z_i} = \exp(z_k)\,\delta_{ki}$$

$$g = \sum_{j=1}^{K} \exp(z_j)$$

$$\frac{\partial g}{\partial z_i} = \sum_{j=1}^{K} \frac{\partial f_i}{\partial z_i} = \exp(z_i)$$

$$f_k = \exp(z_k)$$

Stąd:

$$\frac{\partial s_k}{\partial z_i} = \frac{\frac{\partial f}{\partial z} g - \frac{\partial g}{\partial z} f}{g^2} = \frac{\exp(z_k)\,\delta_{ki} \sum_{j=1}^{K} \exp(z_j) - \exp(z_i)\exp(z_k)}{\left(\sum_{j=1}^{K} \exp(z_j)\right)^2} =$$

$$= s_k \delta_{ki} - s_i s_k = s_k(\delta_{ki} - s_i)$$

Jakobian wektora *softmax*

Wyznaczamy macierz jakobianu (gradientów) funkcji softmax mając dany wektor wyjść *softmax*, $\boldsymbol{s} = [s_1, s_2, \dots, s_K]^T$,

$$\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{z}} = \begin{bmatrix} \frac{\partial s_1}{\partial z_1} & \cdots & \frac{\partial s_1}{\partial z_K} \\ \vdots & \ddots & \vdots \\ \frac{\partial s_K}{\partial z_1} & \cdots & \frac{\partial s_K}{\partial z_K} \end{bmatrix} = diag(\boldsymbol{s}) - \boldsymbol{s}\,\boldsymbol{s}^{\boldsymbol{T}}$$

Funkcja straty

Dla sieci o warstwie „*softmax*” jako *funkcja „straty*” w procesie uczenia sieci stosowana jest entropia krzyżowa (*cross-entropy*). Jak wiemy, jest to miara różnicy dwóch rozkładów prawdopodobieństwa.

W przypadku funkcji softmax jeden rozkład (rzeczywisty) dany jest wektorem wartości wyjść, $\boldsymbol{s} = [s_1, s_2, \ldots, s_K]^T$, a drugi (pożądany) - wektorem prawidłowej klasyfikacji - np. dla próbki klasy „l” (l=1), $\boldsymbol{k}(l) = [1,0,\cdots,0]^T$ („one-hot encoding”):

$$\mathcal{L}(\boldsymbol{k}(l), \boldsymbol{s}) = -\sum_{i=1}^{K} k_i \cdot \ln s_i = -\sum_{i=1}^{K} \delta(i,l) \cdot \ln s_i = -\ln s_l$$

Celem uczenia wag sieci jest minimalizacja (średniej wartości) funkcji straty dla zbioru próbek ze zbioru uczącego.

Gradient funkcji straty

Na początek kroku wstecznej propagacji należy wyznaczyć gradient *funkcji straty* względem wag warstwy z aktywacją *softmax*.

$$\frac{\partial \mathcal{L}(\boldsymbol{k},\boldsymbol{s})}{\partial z_i} = \sum_{j=1}^{K} \frac{\partial \mathcal{L}}{\partial s_j}\frac{\partial s_j}{\partial z_i} = \frac{\partial \mathcal{L}}{\partial s_i}\frac{\partial s_i}{\partial z_i} + \sum_{j=1, j\neq i}^{K} \frac{\partial \mathcal{L}}{\partial s_j}\frac{\partial s_j}{\partial z_i}$$

$$= -\frac{k_i}{s_i}\cdot s_i(1-s_i) - \sum_{j=1, j\neq i}^{K} \frac{k_j}{s_j}(-s_j s_i) = -k_i + s_i \sum_{j=1}^{K} k_j$$

$$= (s_i - k_i)$$

Czyli gradient funkcji straty nie zależy od gradientu funkcji *softmax*. Gradient funkcji straty względem wag ostatniej warstwy:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \sum_{j=1}^{K} \frac{\partial \mathcal{L}}{\partial z_i}\frac{\partial z_i}{\partial w_{ij}} = (s_i - k_i)\cdot x_j$$

Zbiór danych

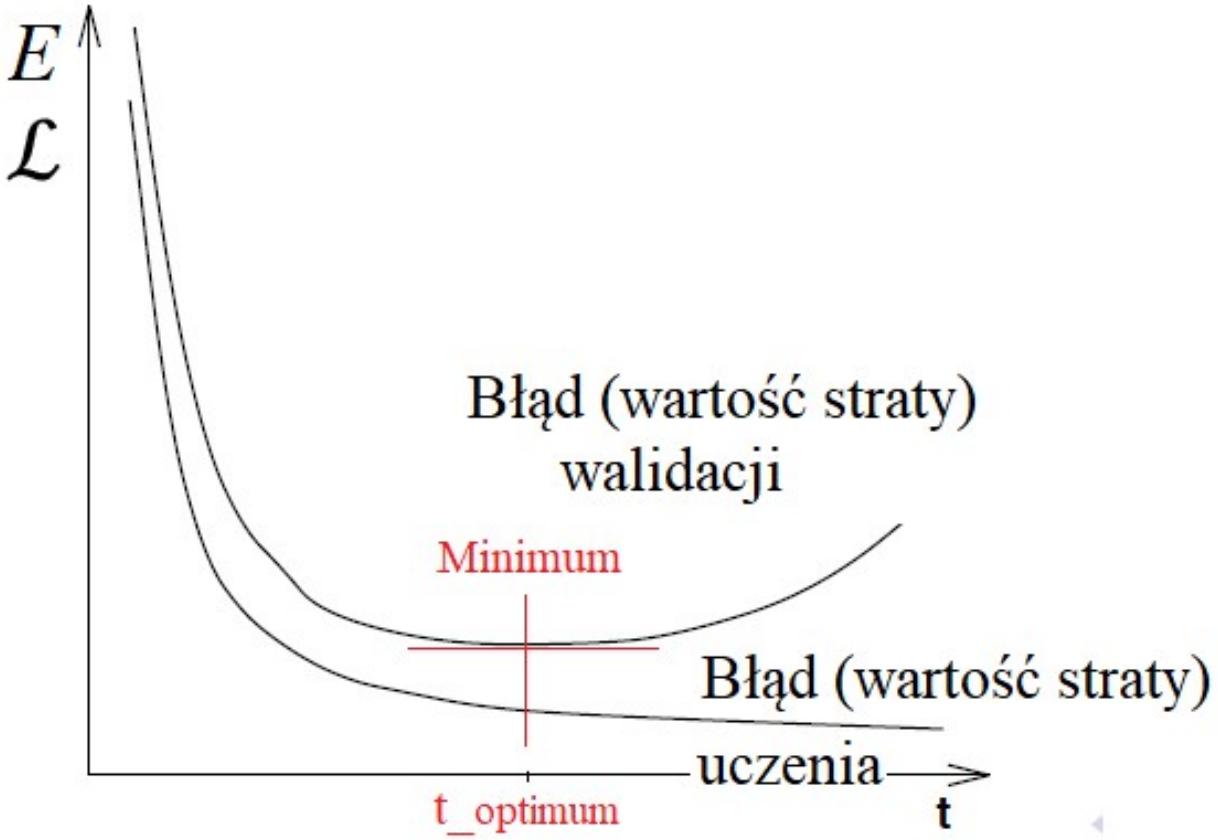
Podział zbioru danych:

- Podzbiór uczący – stosowany do uczenia sieci;
- Podzbiór walidujący – stosowany do określenia błędu sieci podczas uczenia na „nieznanych” danych (np. po każdej epoce);
- Podzbiór testowy – stosowany do określenia błędu sieci na „nieznanych” danych – idealnie tylko raz - po procesie uczenia.

Wczesne kończenie procesu uczenia („*early stopping*”):

- Zbiór walidujący stosowany jest do określenia wcześniejszego niż założono zakończenia procesu uczenia;
- Podczas uczenia błąd sieci na zbiorze uczącym monotonicznie maleje, podczas gdy błąd sieci na zbiorze walidującym osiąga minimum i ponownie zaczyna rosnąć;
- Zakłada się najlepsze działanie sieci w stanie odpowiadającym minimum błędu walidacji.

„Early stopping”



Walidacja krzyżowa

Błąd generalizacji - przewidywany błąd sieci dla „nieznanych” danych. Np. na podstawie zbioru testowego lub walidacyjnego:

$$\mathcal{L}_{train} = -\sum_{n=1}^{N_{train}} \sum_{i=1}^{K} k_i(n) \cdot \ln z_i(n); \quad \mathcal{L}_{valid} = -\sum_{n=1}^{N_{valid}} \sum_{i=1}^{K} k_i(n) \cdot \ln z_i(n)$$

N-krotna walidacja krzyżowa:

- Podziel zbiór danych na N części (partycji);
- Powtórz N-krotnie proces uczenia, za każdym razem zostawiając inną partycję jako dane walidujące a ucząc na pozostałych (N-1) partycjach;
- Po zakończeniu każdego procesu uczenia wyznacz błąd generalizacji na podzbiorze walidującym.
- Określ średni błąd generalizacji na wszystkich podzbiorach walidujących.

4. Poprawianie generalizacji sieci

Techniki poprawiające efekt generalizacji uczonej sieci:

- Regularyzacja - zapobiega eksplozji wartości wag – dodatkowy element funkcji straty zależny od wartości wag – wersje L2, L1;
- Augmentacja danych – generowanie dodatkowych (zaszumionych) danych do trenowania sieci – efekt generalizacji rośnie wraz ze wzrostem liczby danych;
- Łączenie modeli – np. uśrednianie wyniku kilku sieci, ważone łączenie wyników;
- Dropout – losowe „zamrażanie” podzbiorów ukrytych neuronów w kolejnych etapach (dla „mini-zbiorów danych);
- Dzielenie się modelami – np. sieciami splotowymi.

Regularyzacja L2

Dodatkowy element $\mathcal{L}_W$ w funkcji straty $\mathcal{L}(n)$ (względnie błędu E) (dla próbki o indeksie n) związany z wartością kwadratową wag (w iteracji t):

$$\mathcal{L}(n) = \mathcal{L}_{train}(n) + \beta \mathcal{L}_W(t)$$

$\mathcal{L}_W$ powinno być różniczkowalne, stąd może nim być wyrażenie kwadratowe:

$$\mathcal{L}_W(t) = \mathcal{L}_{L2}(t) = \frac{1}{2}\sum_i w_i^2(t)$$

$$\frac{\partial \mathcal{L}_{L2}}{\partial w_i} = w_i$$

Regularyzacja L2

Wersja reguły modyfikacji wag uwzględniająca regularyzację L2 (indeks iteracji „*t*” został pominięty):

$$\frac{\partial \mathcal{L}(n)}{\partial w_i} = \frac{\partial (\mathcal{L}_{train}(n) + \mathcal{L}_{L2})}{\partial w_i} = \left(\frac{\partial \mathcal{L}_{train}(n)}{\partial w_i} + \beta \frac{\partial \mathcal{L}_{L2}}{\partial w_i} \right) =$$

$$= \left(\frac{\partial \mathcal{L}_{train}(n)}{\partial w_i} + \beta w_i \right)$$

Stąd:

$$\Delta w_i = -\eta \left(\frac{\partial \mathcal{L}_{train}(n)}{\partial w_i} + \beta w_i \right)$$

Regularyzacja L1

Regularyzacja L1: dodatkowy element w funkcji straty będący sumą wartości bezwzględnych wag:

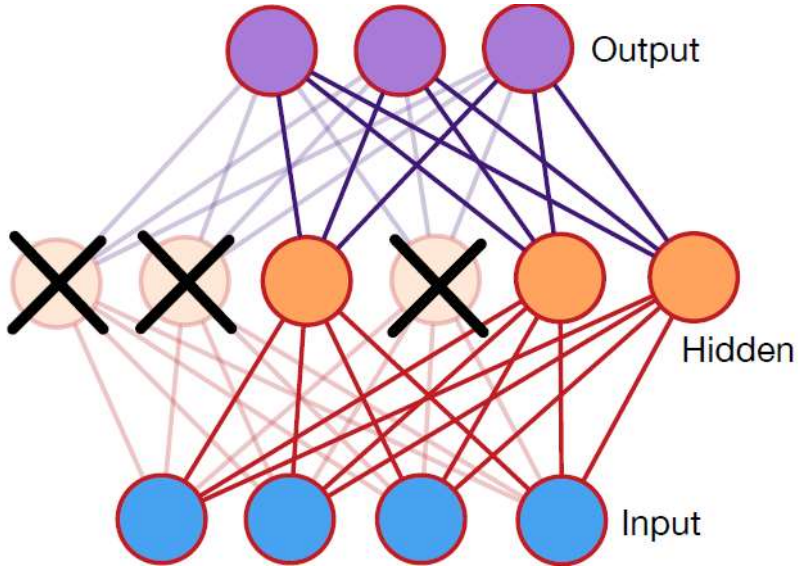
$$\mathcal{L}(n) = \mathcal{L}_{train}(n) + \beta \mathcal{L}_{L1}(t) = \mathcal{L}_{train}(n) + \beta |W(t)|$$

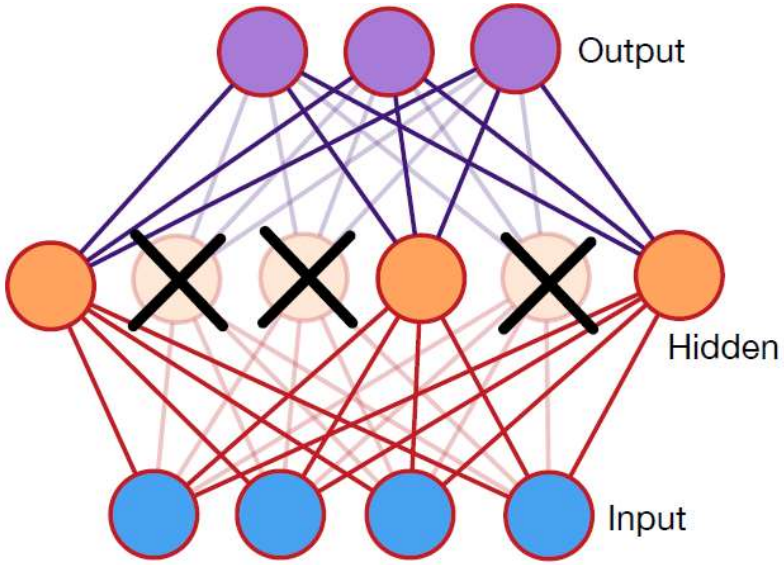
Stąd gradient funkcji straty względem wagi wynosi:

$$\frac{\partial \mathcal{L}(n)}{\partial w_i} = \left(\frac{\partial \mathcal{L}_{train}(n)}{\partial w_i} + \beta \frac{\partial \mathcal{L}_{L1}}{\partial w_i} \right) = \frac{\partial \mathcal{L}_{train}(n)}{\partial w_i} + \beta \, \mathrm{sgn}(w_i)$$

Dropout

Technika uczenia, w której opuszcza się tymczasowo część neuronów w warstwie ukrytej, co modeluje uczenie dla wielu różnych konfiguracji sieci.





5. Hiperparametry

Współczynnik uczenia $\eta(t)$ w regule ujemnego gradientu (*gradient descent*) stosowanej dla modyfikacji wag:

$$\Delta w_{ij}(t) = w_{ij}(t) - w_{ij}(t-1) = -\eta(t)\frac{\partial \mathcal{L}(\mathbf{W}(t))}{\partial w_{ij}(t)}$$

Strategie zależności współczynnika uczenia od czasu:

1. Odwrotnie proporcjonalny do czasu uczenia: $\eta(t) \sim \frac{1}{t}$
2. Odcinkami stała wartość: stała w ramach epoki;
3. Wykładnicza zmiana: $\eta(t) = \eta(0)exp\left(-\frac{t}{N}\right)$, (*N*-liczba próbek)
4. Odwrotność czasu: $\eta(t) = \eta(0)\left(1+\frac{t}{N}\right)^{-c}$, ($c$ ~1)

Momentum

Dodatkowy składnik (momentum, „moment pędu”) w regule modyfikacji wag:

$$\Delta w_{ij}(t) = -\eta(t)\frac{\partial \mathcal{L}\left(\mathbf{W}(t)\right)}{\partial w_{ij}(t)} + \alpha \cdot \Delta w_{ij}(t-1)$$

Hiperparametr „momentum”: $\alpha \sim 0.9$

„Momentum” wprowadza pewną bezwładność kierunku zmian wag dzięki uwzględnieniu, obok aktualnego, także dotychczasowego kierunku zmian.

Adaptacyjny współczynnik uczenia

Adaptacyjny współczynnik uczenia

1. AdaGrad - normalizacja zmiany każdej wagi
2. RMSProp – przeciwdziała ciągłemu zmniejszaniu się wag w procesie AdaGrad;
3. Adam – to RMSProp z dodanym „momentum„ - wprowadza pewną bezwładność kierunku zmian wag, uwzględniając dotychczasowy kierunek zmian.

1. AdaGrad

Stosuje osobną normalizację każdej wagi – podział przez sumę dotychczasowych gradientów danej wagi:

$$S_{ij}(0) = 0;\ S_{ij}(t) = S_{ij}(t-1) + {d_{ij}}^2(t);\ \text{gdzie}\ d_{ij}(t) = \frac{\partial \mathcal{L}\left(\mathbf{W}(t)\right)}{\partial w_{ij}(t)}$$

$$\eta_{ij}(t) = \frac{\eta}{\sqrt{S_{ij}(t)} + \epsilon}$$

RMSProp

2. RMSProp

Zmniejsza szybkość zanikania współczynnika uczącego przez zastosowanie w miejsce sumy kwadratu gradientu, średniej kroczącej kwadratu gradientu z parametrem zapominania $\beta \sim 0.9$:

$$S_{ij}(0) = 0;$$

$$S_{ij}(t) = \beta S_{ij}(t-1) + (1-\beta)\, {d_{ij}}^2(t);\ \text{gdzie}\ d_{ij}(t) = \frac{\partial \mathcal{L}\left(\mathbf{W}(t)\right)}{\partial w_{ij}(t)}$$

$$\eta_{ij}(t) = \frac{\eta}{\sqrt{S_{ij}(t) + \epsilon}}\ ;\ \Delta w_{ij}(t) = \frac{-\eta}{\sqrt{S_{ij}(t) + \epsilon}}\, d_{ij}(t);$$

Okazało się, że wprowadzenie średniej kroczącej w połączeniu z istniejącym *momentum* dla gradientu prowadziło do sytuacji braku gwarancji zanikania współczynnika uczenia w ogóle. Stąd wzięła się koncepcja rozwiązania „Adam”.

Adam

3. Adam

Jest to wariant RMSProp uwzgledniający zarówno średnie kroczące kwadratu gradientu jak i wartości gradientu (czyli wygładzany gradient z *momentum*) w równaniu współczynnika uczenia dla każdej wagi:

$$S_{ij}(0) = 0;\ M_{ij}(0) = 0;$$

$$S_{ij}(t) = \beta S_{ij}(t-1) + (1-\beta)\, {d_{ij}}^2(t);\ \text{gdzie}\ d_{ij}(t) = \frac{\partial \mathcal{L}\left(\mathbf{W}(t)\right)}{\partial w_{ij}(t)}$$

$$M_{ij}(t) = \alpha M_{ij}(t-1) + (1-\alpha)\, d_{ij}(t)$$

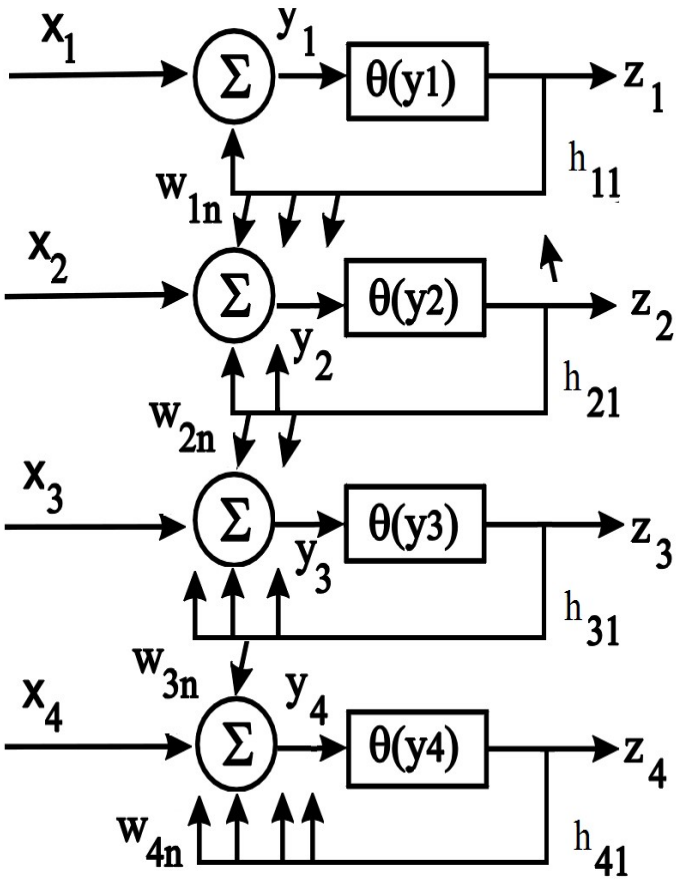
$$\eta_{ij}(t) = \frac{\eta}{\sqrt{S_{ij}(t)} + \epsilon};\ \Delta w_{ij}(t) = \frac{-\eta}{\sqrt{S_{ij}(t)} + \epsilon}\, M_{ij}(t);$$

Autorzy tego rozwiązania (*Kingma & Ba*) rekomendują wartości: $\alpha \sim 0.9,\ \beta \sim 0.999$

6. Neuronowe sieci rekurencyjne

W **sieciach rekurencyjnych** istnieją połączenia wyjść z wejściami neuronów – są to tzw. połączenia **hamujące** (ang. *inhibitory links*).

Aktywacje w sieciach rekurencyjnych ustalają się w wyniku relaksacji (proces dynamicznych zmian) Ponieważ połączenia hamujące realizują sprzężenie zwrotne pomiędzy wyjściem a wejściem sieci rekurencyjnej, więc po jej pobudzeniu stan sieci może oscylować aż do chwili osiągnięcia stanu stabilnego.



Liniowa sieć rekurencyjna

Specyficznym przypadkiem jest liniowa **sieć rekurencyjna. Zapiszmy** wagi połączeń **hamujących** między neuronami warstwy w postaci macierzy wag **H**. Funkcja pobudzenia neuronów takiej sieci ma postać:

$$\mathbf{y} = \mathbf{x} - \mathbf{H} \cdot \mathbf{y}$$

W stanie stabilnym liniowa sieć rekurencyjna realizuje funkcję równoważną funkcji **liniowej sieci jednokierunkowej** o postaci:

$$\mathbf{y} = (\mathbf{I} + \mathbf{H})^{-1} \cdot \mathbf{x}$$

Przykłady

Przykłady sieci rekurencyjnych

- Sieć jednowarstwowa Hopfielda – pamięć asocjacyjna. Odtwarza ona pełny „najlepszy” wzorzec przy jej pobudzeniu niepełnym wzorcem.
- Maszyna wielowarstwowa Boltzmana – o binarnych wyjściach i ze stochastyczną regułą aktywacji tzw. „symulowanego wychładzania” (ang. *simulated annealing*) zależną od energii własnej i temperatury sieci. Rozwiązuje ona problemy globalnej optymalizacji.

Zastosujemy warstwę sieci rekurencyjnej do wyznaczenia:

1. wejścia o największej wartości (ang. *Winner Takes All*);
2. K wejść o największej wartości (ang. *kWTA*)

Sieć WTA

Sieć WTA („zwycięzca bierze wszystko”) jest to jednowarstwowa sieć rekurencyjna, która:

1. Posiada nieliniową aktywację w postaci funkcji RELU;
2. Jej wagi hamujące są postaci:

$$h_{im} = \begin{cases} -\varepsilon & gdy \quad i \neq m \\ +1 & gdy \quad i = m \end{cases}$$

co oznacza, że wyjście neuronu dodaje się do sumy na jego wejściu a odejmuje się wartości wyjść wszystkich pozostałych neuronów przemnożone przez parametr ε .

W dynamicznym procesie oscylacji po kolei wszystkie wyjścia o mniejszych wartościach zostają zmniejszone do wartości zero i pozostaje jedno wyjście niezerowe odpowiadające wejściu o największej wartości.

Uwaga: wszystkie wartości wejściowe powinny być nieujemne.

Neuronowa klasteryzacja

Sieć Kohonena jest siecią dwuwarstwową, w której wagi pierwszej warstwy uczone są metodą „w warunkach konkurencji” (ang. *competitive learning*).

Pierwsza warstwa jest liniowa i jednokierunkowa (ang. *feed-forward*) a jej wagi docelowo wyznaczają środki klastrów danych wejściowych.

Druga warstwa jest rekurencyjna i służy jedynie w procesie uczenia do wyznaczenia wyjścia pierwszej warstwy o największej wartości (problem WTA).

W pierwszej warstwie każde z wyjść ***i*** jest połączone z każdym wejściem ***j***, zaś funkcja ***i***-tego wyjścia wynosi: $y_i = \mathbf{w}_i^T \mathbf{x}$.

Wyjścia powinny zostać znormalizowane długością wektora wag:

$$z_i = f(y_i) = \frac{y_i}{|\mathbf{w}_i|}$$

Uczenie „w warunkach konkurencji”

W procesie uczenia dla każdej próbki wejściowej wybierany jest więc neuron wyjściowy o najwyższej aktywacji i jego wagi $\mathbf{w}_l$ są korygowane „w kierunku” aktualnego wektora wejściowego $\mathbf{x}(t)$:

$$\mathbf{w}_l(t+1) = \mathbf{w}_l(t) + \eta_l\,[\mathbf{x}(t) - \mathbf{w}_l(t)].$$

Jednocześnie umożliwia on częściową aktywację neuronów ze swojego „sąsiedztwa” w stopniu zależnym od odległości ich wektorów wag $\mathbf{w}_k$ od wag neuronu wygrywającego:

$$\mathbf{w}_k(t+1) = \mathbf{w}_l(t) + \eta_k G(k, l, \mathbf{x}(t))\,[\mathbf{x}(t) - \mathbf{w}_k(t)].$$

Funkcja sąsiedztwa $G(k, l, \mathbf{x}(t))$ wyznacza stopień „uczenia” (wartość z przedziału [0,1]) sąsiada o indeksie k zwycięskiego neuronu l w iteracji t.

Dla oryginalnej sieci Kohonena zachodzi: $G(k,l,x(t)) = \begin{cases} 0 & gdy \quad k \neq l \\ +1 & gdy \quad k = l \end{cases}$

co oznacza, że do modyfikacji nie dopuszczano żadnych neuronów sąsiednich (strategia „*winner takes all*”).

Sieć kWTA

Sieć WTA może być też zastosowana do znalezienia k wejść o największych wartościach (problem kWTA, „k zwycięzców”) spośród N wejść, gdzie k = 1, 2, …, N-1.

W tym przypadku potrzebna jest druga warstwa, która zlicza liczbę wyjść o zerowej wartości. W momencie, gdy liczba zerowych wyjść osiągnie N-k, pozostaje k wyjść o dodatnich wartościach i proces oscylacji należy przerwać.

Sieć kWTA

Program w Matlabie:

```
function [z, iternum] = kWTA( u, maxiter, k )
% Argumenty: u -  wektor „n” liczb (sygnały wejściowe dla sieci),
%                 maxiter – maksymalna liczba iteracji,
%                 k – liczba szukanych zwycięzców.
% Zwracany wynik: z – wektor „n” liczb (sygnały wyjściowe) –
% niezerowa wartość na i-tym wyjściu wskazuje, że odpowiednie i-te
% wejście należy do zbioru „k zwycięzców”.
        [c, n] = size(u);
        if (k> (n-1)) % błędna wartość argumentu k
            iternum = 0;
            z = u;
           return;
        end
        eps = 1.0/(n+k);
```

Sieć kWTA (c.d.)

```
% Inicjalizacja
z = u;   newz = z;
iternum = maxiter; % maksymalna liczba iteracji
for i=1 : maxiter %
    sumuvec = sum(z);      zeronum = 0;
    for j=1:n
        newz(j) = u(j) + z(j) - eps *(sumuvec - z(j)); % reguła modyfikacji
        if newz(j) < 0  % nieliniowość RELU
            newz(j) = 0;          zeronum = zeronum +1;
        end
    end
    z = newz; % synchroniczna modyfikacja wszystkich wyjść
    % Sprawdź aktualną liczbę niezerowych wyjść u(j):
    if zeronum >= (n-k)
        iternum = i;
        break; % zakończ
    end
end
```

7. Sieciowe modele klasycznych technik uczenia maszynowego

Wiele klasycznych modeli uczenia maszynowego stosuje metody optymalizacji ciągłej dziedziny. Wszystkie te modele mogą być też wyrażone jako specjalne przypadki *płytkich* sieci neuronowych.

Np.

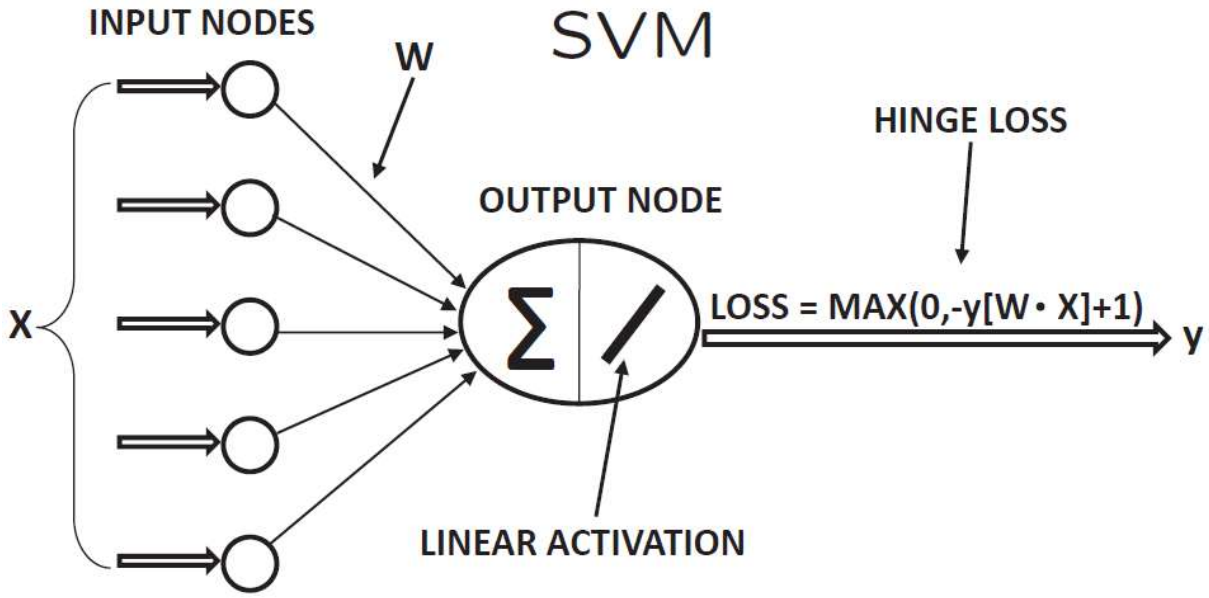
- Klasyfikator SVM → Sieć SVM
- Regresja (liniowa, logistyczna, itd.) → Perceptron z odpowiednią funkcją aktywacji
- Dekompozycja wg. wartości własnych (SVD) → liniowy autoenkoder
- Faktoryzacja niezupełnej macierzy → autoenkoder z warstwą ukrytą

Sieć SVM

Reguła modyfikacji wag sieci SVM:

$$W(n+1) = W + \alpha\, y\, X$$

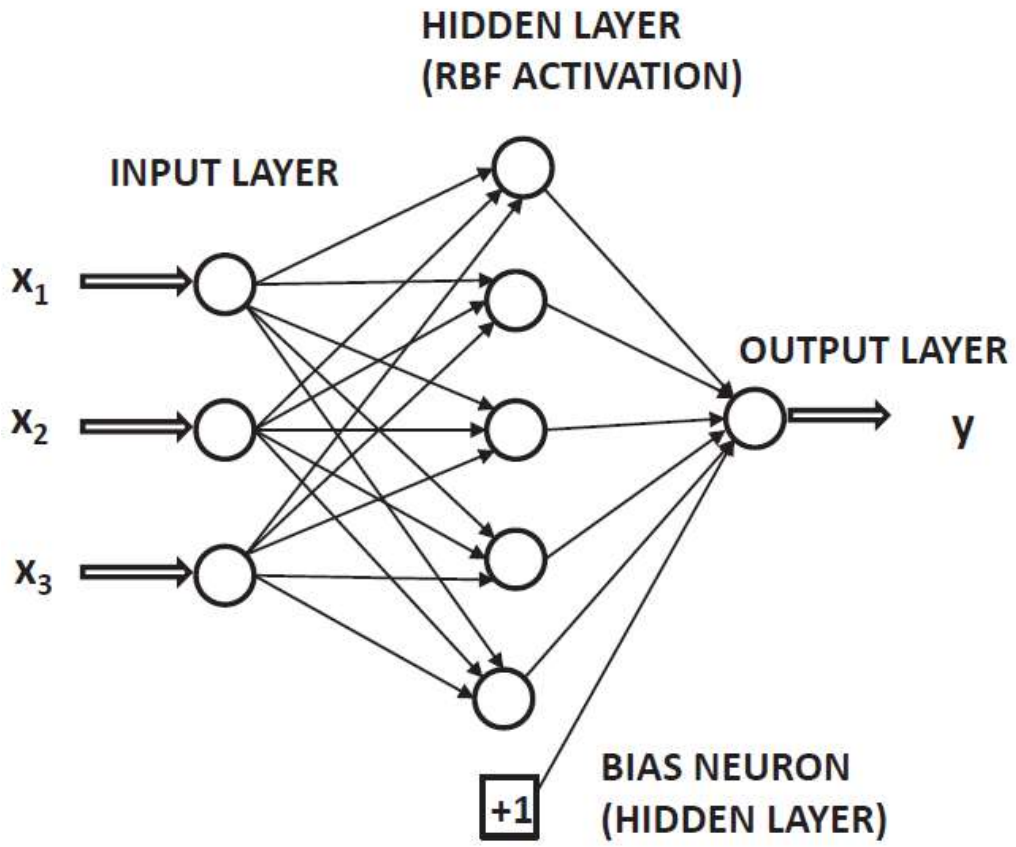
wykonywana dla błędnie klasyfikowanych próbek i także dla „marginalnie poprawnych” próbek uczących.



$$\text{Loss} = \max\{0, 1 - y(\overline{W} \cdot \overline{X})\}$$

Sieć RBF

Sieć perceptronowa z funkcją aktywacji RBF („Radial Basis Function”) może realizować klasyfikację „kernel SVM”.



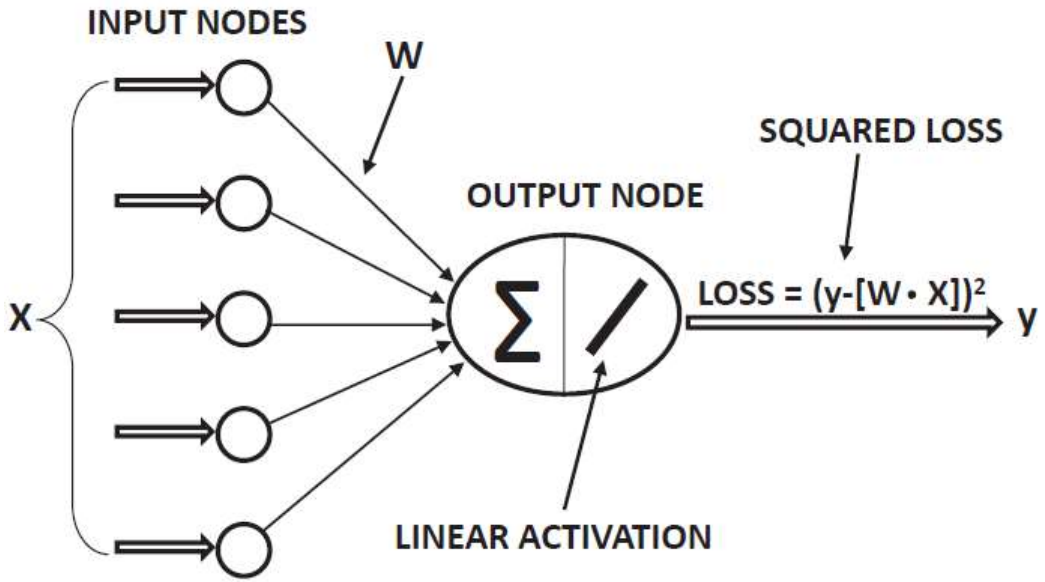
Regresja liniowa

Sieć perceptronowa z liniowym wyjściem oraz optymalizowaną funkcją straty („loss”) wyrażającą błąd kwadratowy aproksymacji:

$$\hat{y}_i = W \cdot X_i\,, \qquad i = 1,2,\ldots,N$$

Loss: $L_i = (y_i - \hat{y}_i)^2$. Gradientowa reguła modyfikacji:

$$W(n+1) = W(n) - \alpha \frac{\partial L_i}{\partial W} = W(n) + \alpha(y_i - \hat{y}_i)X_i$$



Pytania

1. Omówić model neuronu.
2. Omówić pojęcie wielowarstwowego perceptronu MLP.
3. Przedstawić kryterium optymalizacji procesu uczenia i podstawowy algorytm uczenia sieci MLP.
4. Omówić rolę „entropii krzyżowej” w procesie uczenia sieci klasyfikującej.
5. Przedstawić funkcję „softmax” i kryterium uczenia sieci stosującej „softmax”.
6. Przedstawić sieć rekurencyjną i jej zastosowanie jako kWTA (*k-winners-take-all*).
7. Podać przykłady płytkich sieci neuronowych realizujących klasyczne modele uczenia maszynowego (np. SVM, regresja liniowa).