Sztuczne Sieci Neuronowe

Wykład 4

1. Zdolności uogólniania sieci, weryfikacja procesu uczenia (przypomnienie)
2. Siec wielowarstwowa perceptronowa
3. Algorytmy uczenia sieci metodami propagacji wstecznej bledu.

wykład przygotowany na podstawie.

S. Osowski, "Sieci Neuronowe w ujęciu algorytmicznym ", Rozdz. 3, PWNT, Warszawa 1996
S. Osowski, „Sieci neuronowe do przetwarzania informacji", Oficyna Wydawnicza PW, Warszawa 2000.
R. Tadeusiewicz, "Sieci Neuronowe", Rozdz. 4. Akademicka Oficyna Wydawnicza RM, Warszawa 1993.

Zdolności uogólniania sieci neuronowej

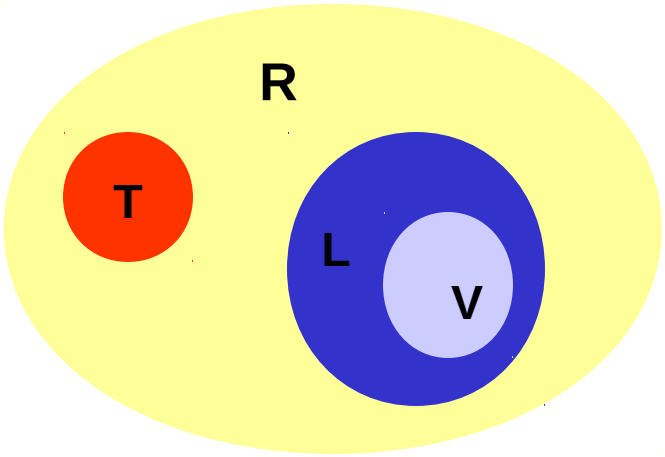
Podstawową cechą sieci neuronowej jest jej zdolność do uogólniania, a więc generowania właściwego rozwiązania dla danych, które nie pojawiły się w zestawie danych uczących.

➔Sieć zostaje poddana uczeniu na zbiorze **L** z bieżącym sprawdzeniem stopnia uczenia na zbiorze **V**.

➔Zdolność odtworzenia zbioru **L** przez sieć jest miarą **zdolności zapamiętania** danych uczących

➔ Zdolność do generowania właściwych rozwiązań dla danych należących do zbioru **T**, na których sieć nigdy nie była trenowana, jest miarą **zdolności uogólniania**.

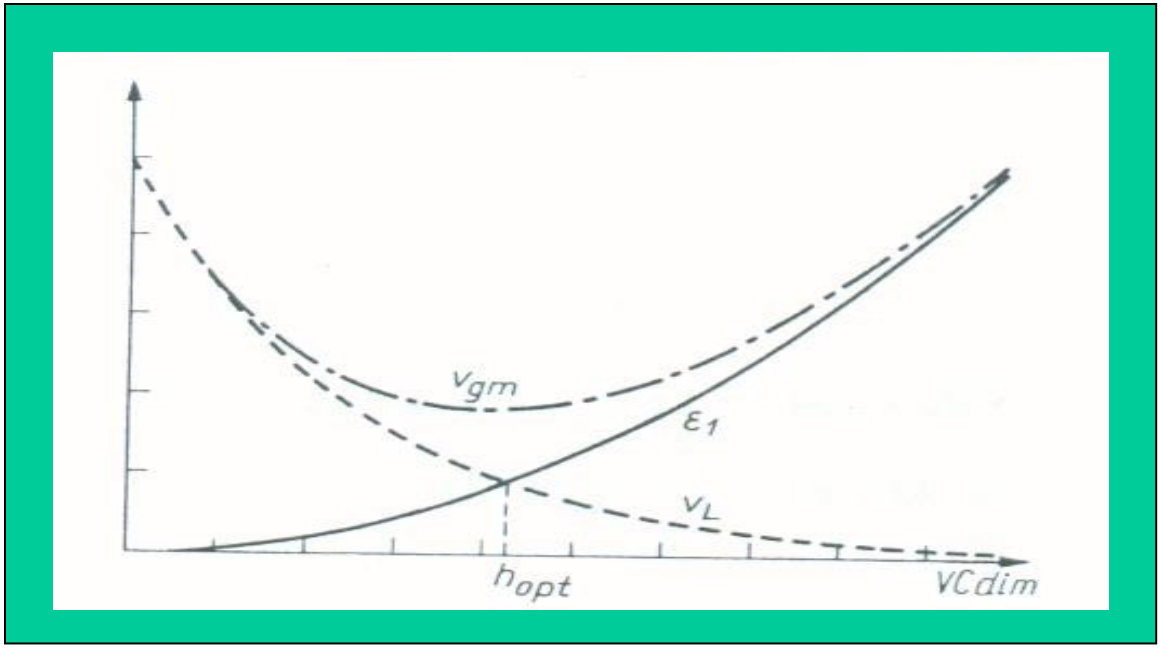
(Zakłada się że dane tworzące zarówno zbiór L jak i zbiór T są typowymi reprezentantami zbioru danych)



R – zbiór danych wejściowych
T - zbiór testujący (testing)
L - zbiór uczący (learning)
V - zbiór danych sprawdzających (validation)

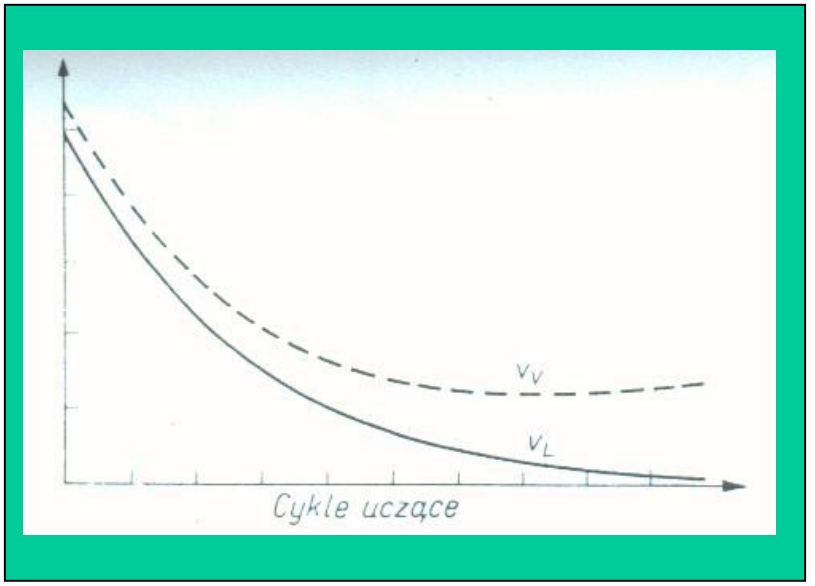
Zdolności uogólniania sieci neuronowej

Przy **stałej liczbie próbek p** i wzrastającej wartości miary ***VCdim*** błąd uczenia $\mathbf{v_L(W)}$ maleje monotonicznie, a przedział ufności ε_1 rośnie. W efekcie maksymalny błąd uogólniania osiąga minimum. Zakres $\mathbf{VCdim < h_{opt}}$ odpowiada nadmiarowości danych bieżących względem aktualnej wartości ***VCdim***. *Zakres* $\mathbf{VCdim > h_{opt}}$ odpowiada zbyt małej liczbie danych uczących przy aktualnej wartości ***VCdim***.



Cykla uczące i błąd veryfikacji

W ogólnym przypadku wraz z upływem czasu uczenia błąd uczenia $v_L(W)$ (learning) maleje i błąd testowania $v_V(W)$ (verification) również maleje (przy ustalonej wartości liczby próbek uczących **p** oraz miary ***VCdim***).



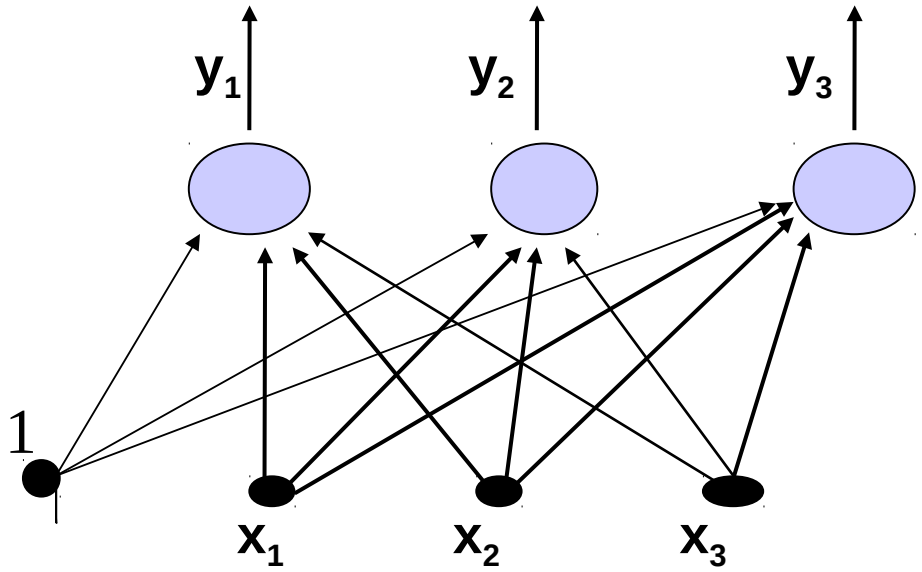
Od pewnego momentu błąd weryfikacji pozostaje stały, natomiast błąd uczenia nadal maleje. W ostatnich fazach procesu uczenia nieregularności w danych odbiegające od cech charakterystycznych danego procesu zaczynają odgrywać role i powodują wzrost błędu testowania.

PERCEPTRON ... raz jeszcze ...

Siecią neuronową, która odegrała historycznie bardzo istotną rolę był

PRECEPTRON

koncepcja w której wprowadzono nieliniowy element przetwarzający informację. Wprowadzenie nieliniowości było uzasadnione, biologiczne układy faktycznie są nieliniowe.



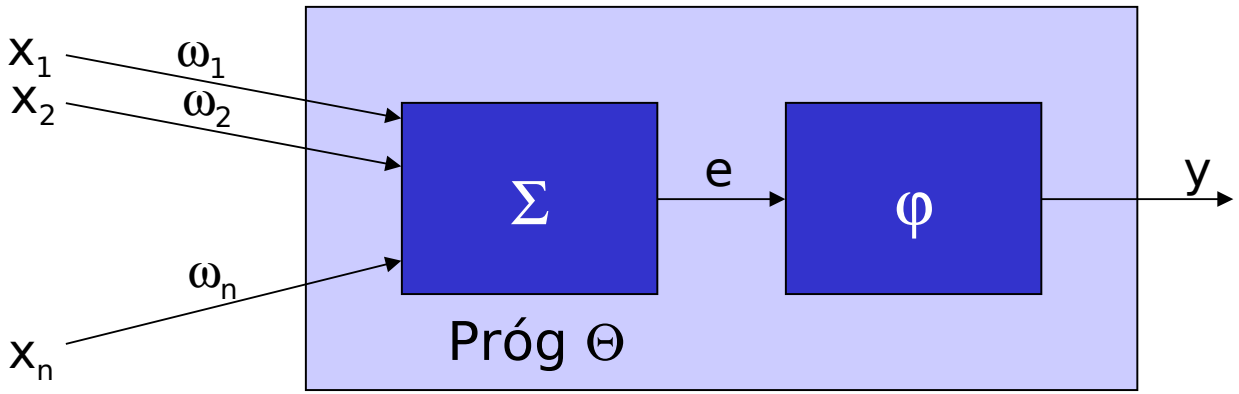
Siec perceptronowa jednowarstwowa

PERCEPTRON ... raz jeszcze ...

Nieliniowy element przyjmowany w sieciach neuronowych może być opisany równaniem.

$$\mathbf{y} = \phi(\mathbf{e})$$

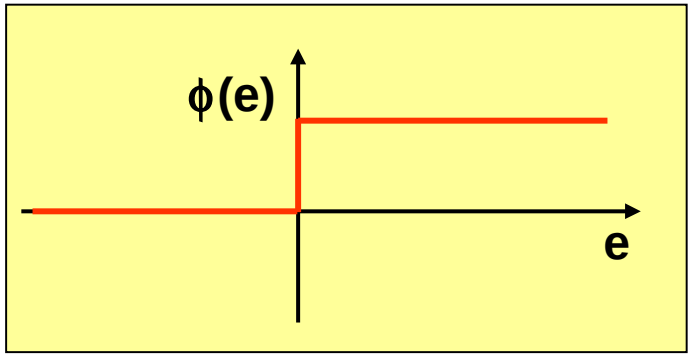
gdzie **φ(e)** jest wybraną funkcją nieliniową a **sygnał *e*** odpowiada łącznemu **pobudzeniu neuronu**.



PERCEPTRON ... raz jeszcze ...

O specyficznych własnościach ***perceptonu*** decyduje funkcja **φ** określającą nieliniowy związek miedzy sygnałem wypadkowego pobudzenia neuronu ***e***, a jego odpowiedzią **y**.
W klasycznym perceptonie funkcja **φ** ma postać progową:

$$\phi(\mathbf{e}) = \mathbf{1}\ \mathbf{gdy}\ \ \mathbf{e} \geq \mathbf{0}$$
$$\phi(\mathbf{e}) = \mathbf{0}\ \mathbf{gdy}\ \ \mathbf{e} < \mathbf{0}$$



Ta postać ma szereg wad ale jest łatwa do wyprowadzenia pewnych intuicji. Ponieważ sygnał wyjściowy przyjmuje wartość (**y=1** lub **y=0**) , może być rozważany w kategoriach określonej ***decyzji***.
Możliwa jest też interpretacja oparta na logice matematycznej, ***prawda*** lub ***fałsz***.

Percepton może być interpretowany jako układ realizujący pewną funkcję logiczną, a więc automat skończony.

Własności nieliniowych sieci

Przyjmując interpretację progowej funkcji **ϕ(*e*)** jako funkcji rozdzielającej przestrzeń wejściowych sygnałów **X** na obszar wyróżniony, w którym **y=1**, oraz na resztę – należy stwierdzić, że przy przyjęciu najczęściej rozważanej reguły scalania wejściowych sygnałów w postaci

$$e = \sum_{i=1}^{n} \omega_i x_i$$

podział ten formułuje granica mająca postać hiperpłaszczyzny.
Istotnie, jeśli **φ(*e*) = 1** gdy **e ≥ 0**; oraz **φ(*e*) = 0** gdy **e < 0**; to obszar w którym neuron podejmuje decyzje **y=1** ogranicza powierzchnia **e=0**, czyli twór o równaniu

$$\sum_{i=1}^{n} \omega_i x_i = 0$$

Dla **n=2** jest to równanie lini prostej, dla **n=3** – równanie płaszczyzny, a dla **n > 3** twór nazywany prawidłowo ***rozmaitością liniową stopnia n-1***, a popularnie traktowany jako płaszczyzna w n-wymiarowej przestrzeni czyli w skrócie ***hiperpłaszczyzna.***

.

Własności nieliniowych sieci

Możemy interpretować działanie neuronu budującego perceptron jako dyskryminatora liniowego.

Może on zrealizować te wszystkie odwzorowania, w których wystarczy oddzielenie podobszaru przestrzeni **X** mającego formę otwartej podprzestrzeni ograniczonej **hiperpłaszczyzną**.

Proces uczenia, polegający zawsze na zmianie wartości współczynników ω_i, pozwala ustalić **graniczną hiperpłaszczyznę** w dowolnym położeniu, nie pozwala jednak na zmianę charakteru **realizowanego odwzorowania**, niezależnie od tego jak długo by się go uczyło.

Separowalność liniowa

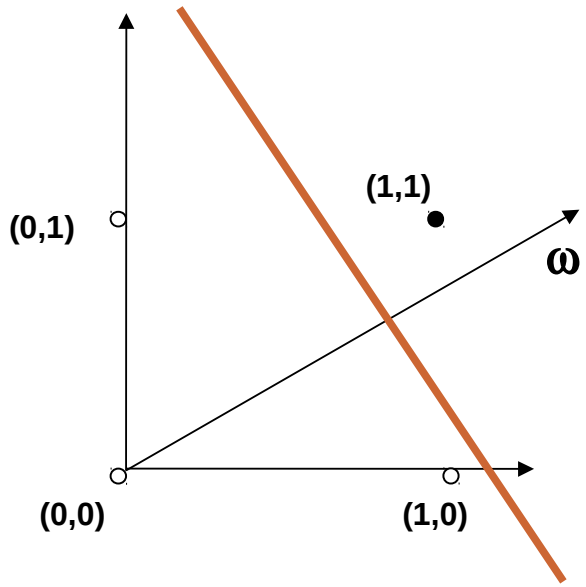
Co sie stanie jeżeli nie istnieje taka płaszczyzna?

Wtedy zadanie nie może być rozwiązane – sieć nie może osiągnąć funkcji celu niezależnie od sposobu jej uczenia. Warunkiem rozwiązania za pomocą perceptronu prostego z jednostkowymi progami jest wymaganie aby dany problem był ***liniowo separowalny***.

$$\mathbf{Y} = \phi(\omega_1 \mathbf{x_1} + \omega_2 \mathbf{x_2} - \omega_0)$$

Funkcja logiczna: **AND**

x_1	x_2	Y
0	0	0
0	1	0
1	0	0
1	1	1



Separowalność liniowa

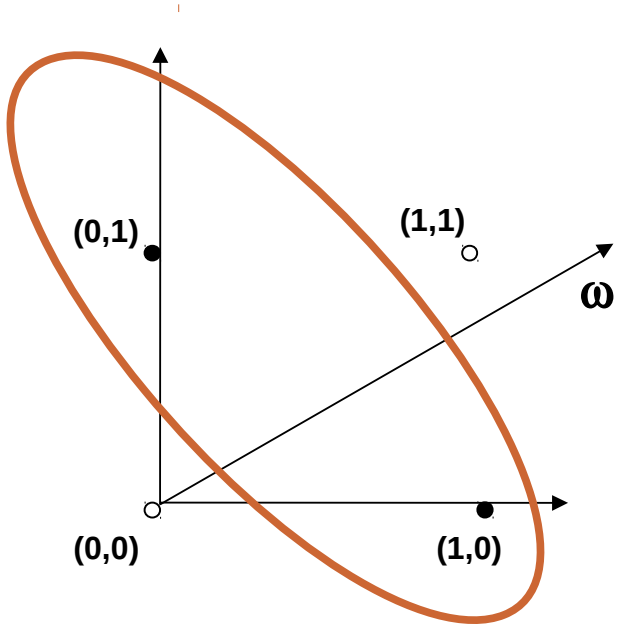
Przykład dla którego brak jest liniowej separowalności: funkcja **XOR**

Nie istnieje rozwiązanie dla takiego układu równań reprezentujacy pojedynczy neuron .

$$\mathbf{Y} = \phi(\omega_1 \mathbf{x_1} + \omega_2 \mathbf{x_2} - \omega_0)$$

Funkcja logiczna: **XOR**

x_1	x_2	Y
0	0	0
0	1	1
1	0	1
1	1	0



Własności nieliniowych sieci

Nierozwiązywalne zadanie: “problem XOR”

Percepton ***nie może*** się nauczyć realizacji odwzorowania

$$y = x_1 \otimes x_2$$

gdzie operator $\oplus$ oznacza alternatywę wyłączającą (eXclusive OR).

Kilkuwarstwowa sieć:

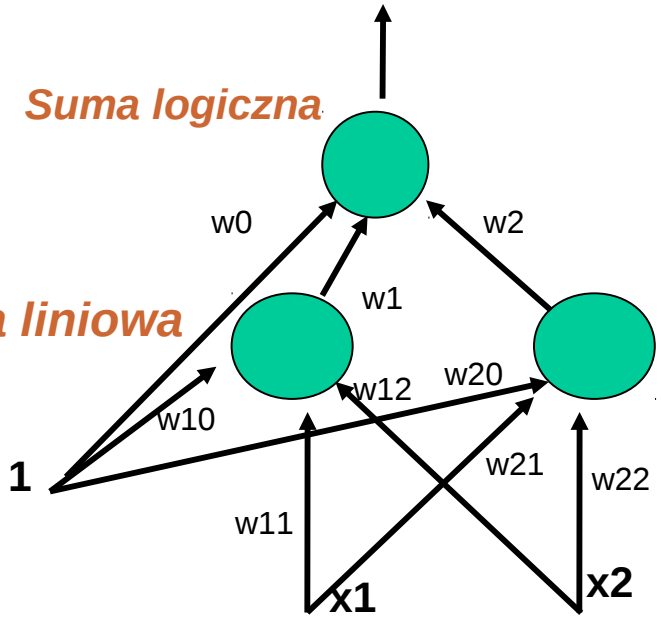
Jednak, czego nie potrafi zrobić jeden neuron, może zrobić kilkuwarstwowa sieć, ponieważ **dla nieliniowych neuronów dodanie nowych warstw istotnie poszerza zakres odwzorowań**, które sieć potrafi zrealizować.

Separowalność dla sieci dwuwarstwowej

Funkcja **XOR:**
rozwiązuje problem nieliniowej separowalności przez złożenie dwóch separowalności liniowych

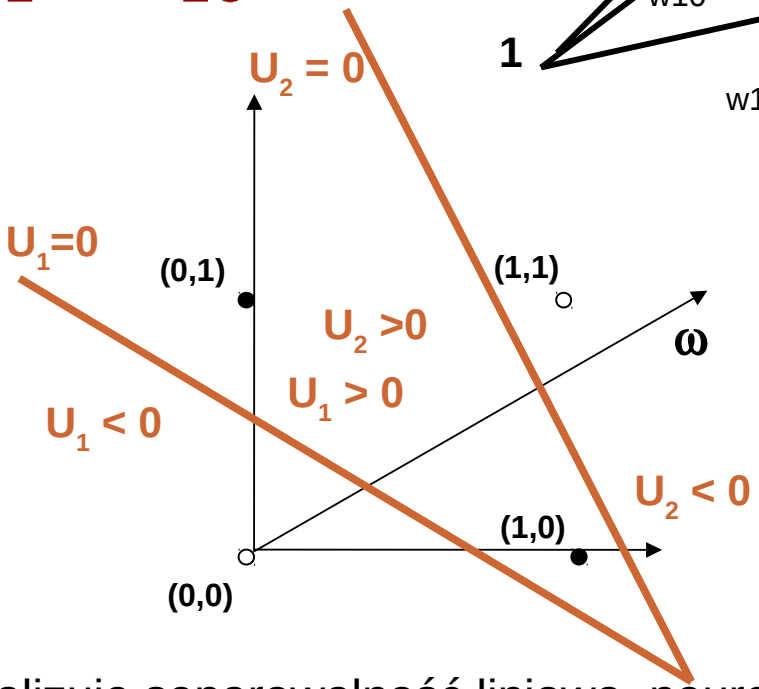
$$U_1 = \omega_{11}x_1 + \omega_{12}x_2 + \omega_{10}$$

$$U_2 = \omega_{21}x_1 + \omega_{22}x_2 + \omega_{20}$$



Funkcja logiczna: **XOR**

x_1	x_2	Y
0	0	0
0	1	1
1	0	1
1	1	0



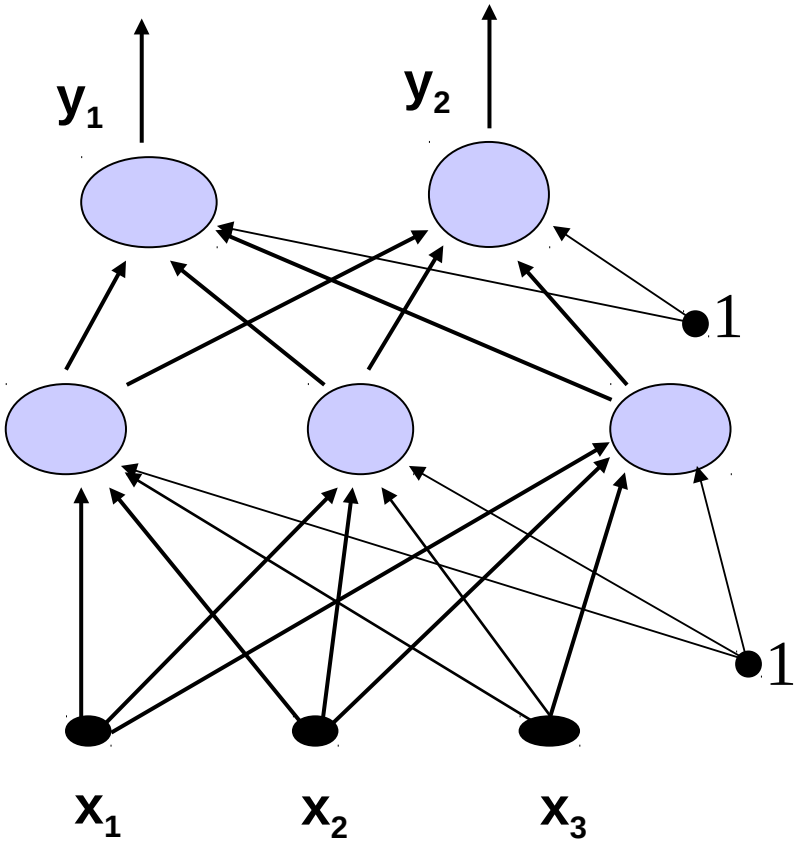
Neuron w warstwie ukrytej realizuje separowalność liniową, neuron w warstwie wyjściowej wykonuje odpowiednią kombinację liniową, np. sumę logiczną

Własności sieci perceptronowych

Rozważmy przykładową ***sieć dwuwarstwową*:**

Pierwsza warstwa, złożona z **k** neuronów otrzymujących sygnały wejściowe **X**, dzieli przestrzeń **X** tych sygnałów za pomocą **k** oddzielnych hiperpłaszczyzn.

Powstaje w ten sposób układ **2k** liniowo rozdzielnych obszarów, które sygnalizowane są przez odpowiednie zestawy **0** i **1** jako wartości sygnałów neuronów pierwszej warstwy.

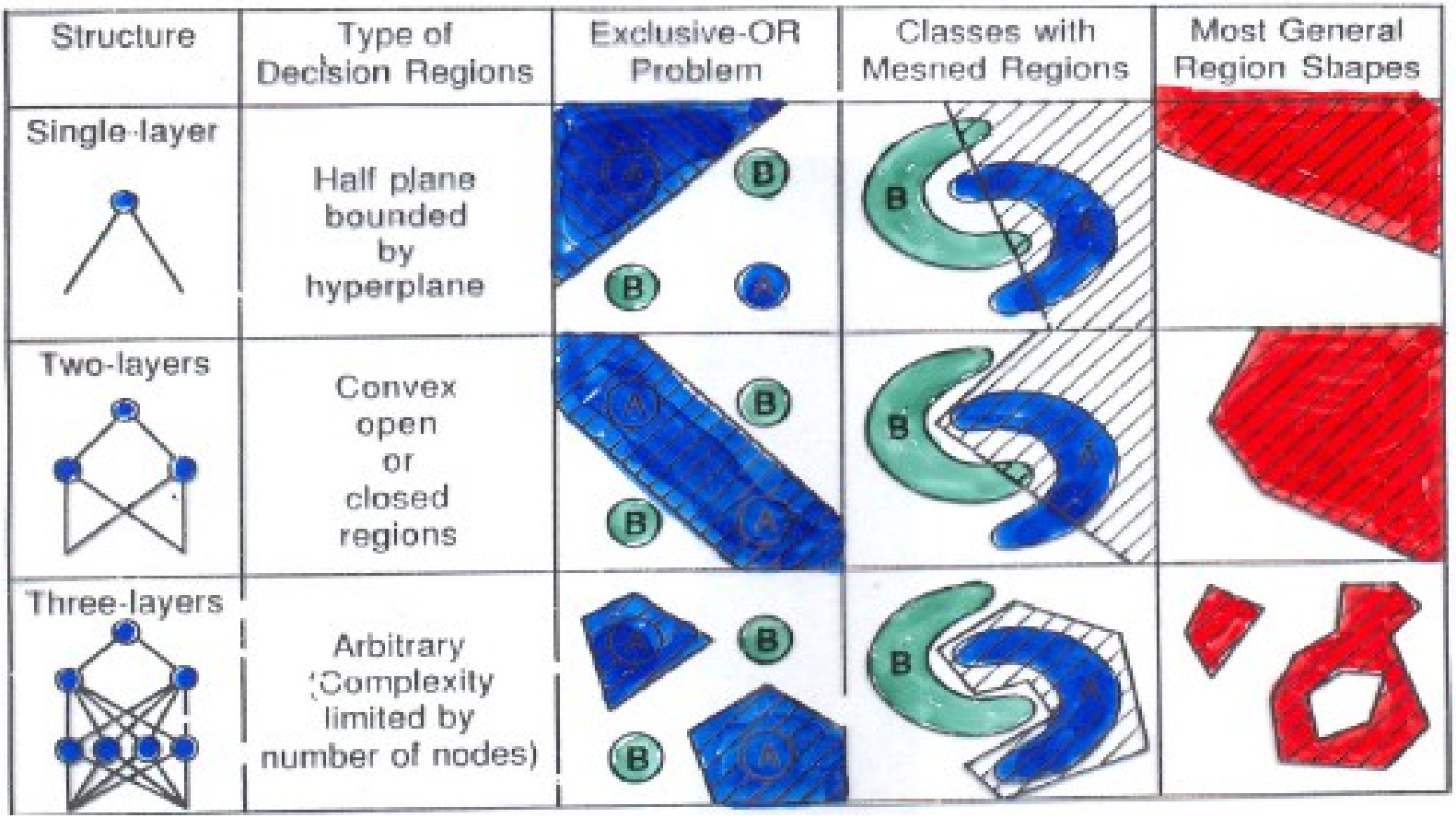


Własności nieliniowych sieci

Sieć dwuwarstwowa nie pozwala jeszcze rozpoznać dowolnego podobszaru przestrzeni X, ponieważ łatwo sprawdzić, że obszary sygnalizowane przez neurony drugiej warstwy musza być ***wypukłe*** oraz ***jednospójne*** (simpleksy).

Jest to dość istotne ograniczenie. Aby się od niego uwolnić należy wprowadzić **trzecią warstwę neuronów**. Dopiero w rezultacie dołączenia trzeciej warstwy możliwe jest utworzenie ***dowolnych*** obszarów.

Schemat Liebmanna



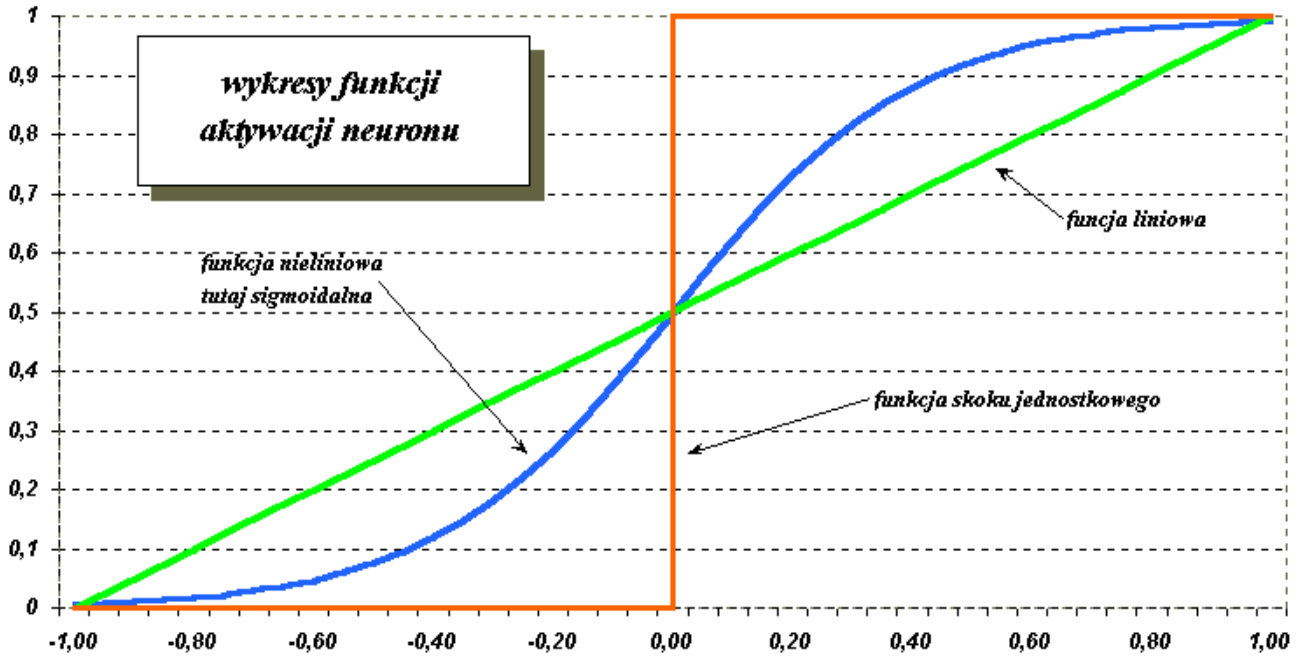
Za pomocą nieliniowej sieci neuronowej o przynajmniej trzech warstwach można zrealizować *dowolne odwzorowanie, wiążące w całkowicie dowolny sposób wejściowe sygnały X z wyjściowymi sygnałami sieci.*

Formy nieliniowości neuronu

Funkcja wiążąca łączne pobudzenie neuronu ***e*** z jego sygnałem wyjściowym **y**

$$\mathbf{y} = \phi(\mathbf{e})$$

Sigmoidalna funkcja wywodzącą się z funkcji logistycznej.



Formy nieliniowości neuronu

Funkcja tangens hiperboliczny:

$$y = \tanh (\beta e)$$

który można rozpisać jako

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

Przy zastosowaniu tej funkcji $y \in (0,1)$
Zaletą tej funkcji jest prosta formuła określająca pochodną tej funkcji w zależności od jej wartości

$$d \varphi / de = (1 + y) (1 - y)$$

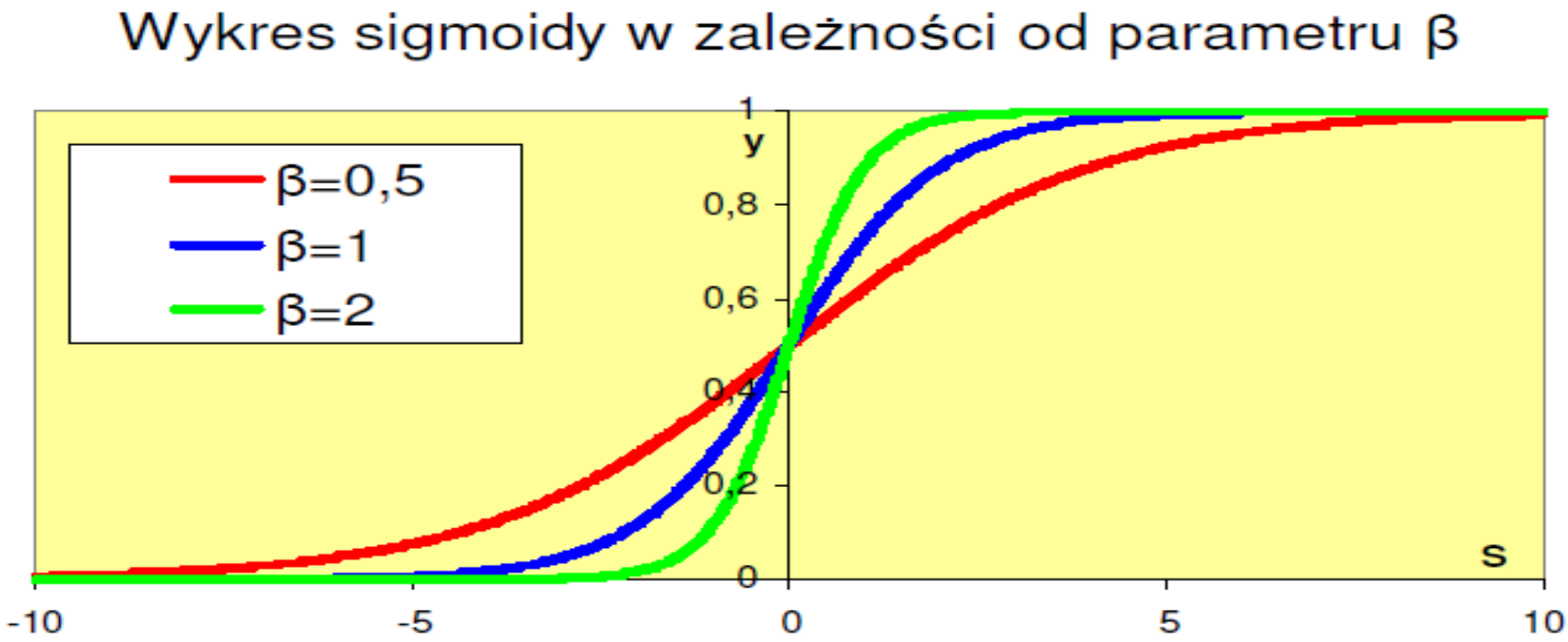
Funkcja sinus:

chętnie stosowana, można doformułować do przedziału zamkniętego [-1,1]:

$$y = \begin{cases} -1 & \text{gdy } e < -\pi/2 \\ \sin(\beta e) & \text{gdy } -\pi/2 < e < \pi/2 \\ 1 & \text{gdy } e > \pi/2 \end{cases}$$

Ta postać funkcji jest szczególnie przydatna przy budowie sieci dokonującej transformaty Fouriera wejściowego sygnału.

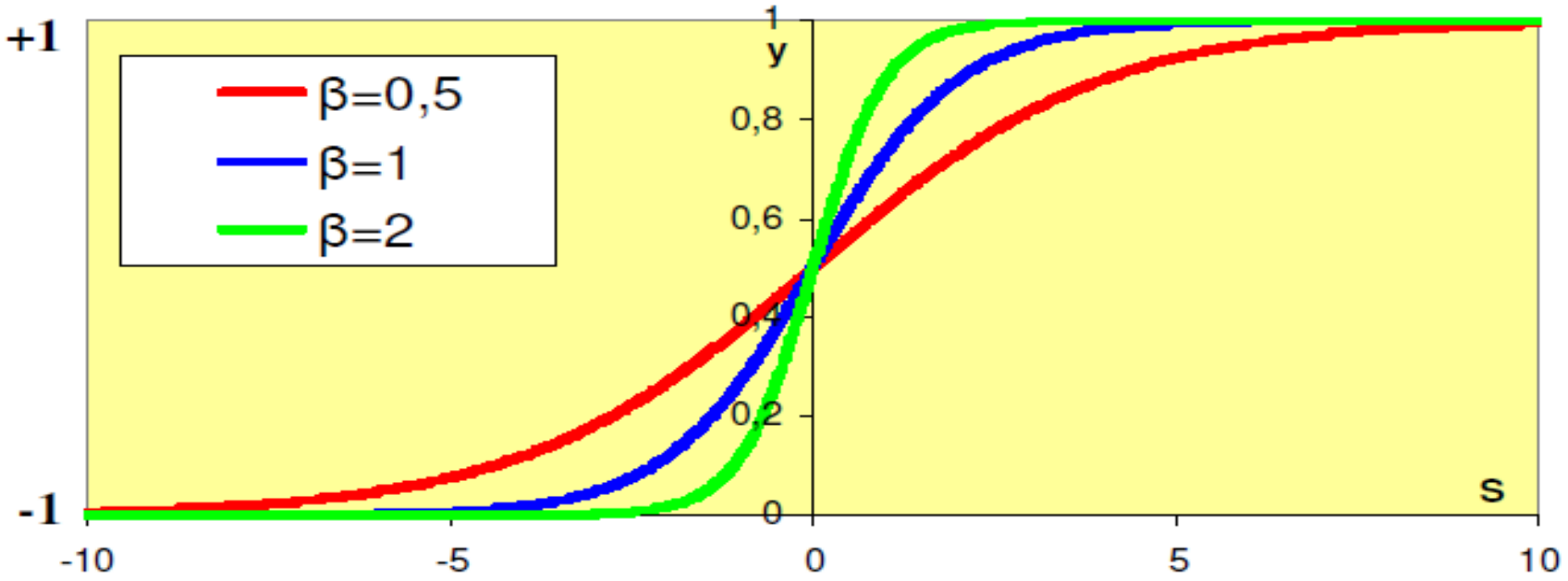
Funkcja sigmoidalna



$$f(s) = \frac{1}{1+\exp(-\beta s)}$$

Funkcja tangs hiperboliczny

Funkcja tangens hiperboliczny ma praktycznie taki sam kształt, tylko jej wartości zmieniają się od **-1** do **+1**, a nie od **0** do **+1** jak w sigmoidzie



$$f(s) = \tanh(\beta s) = \frac{\exp(\beta s) - \exp(-\beta s)}{\exp(\beta s) + \exp(-\beta s)}$$

Formy nieliniowości neuronu

Niekiedy nieliniowość ma postać nie różniczkowalną, przydatną w praktycznych zastosowaniach, ale kłopotliwą do teoretycznej analizy. Z bardziej znanych postaci można wymienić:

Funkcje signum:

$$y = \begin{cases} 1 & \text{gdy } e > 0 \\ 0 & \text{gdy } e = 0 \\ -1 & \text{gdy } e < 0 \end{cases}$$

Funkcja perceptonowa:

$$y = \begin{cases} e & \text{gdy } e > 0 \\ 0 & \text{gdy } e \le 0 \end{cases}$$

Funkcja BSB
(Brain State in a Box)

$$y = \begin{cases} 1 & \text{gdy } e > 1 \\ e & \text{gdy } 1 > e > -1 \\ -1 & \text{gdy } e < -1 \end{cases}$$

Zmodyfikowana funkcje signum:

$$y = \begin{cases} 1 & \text{gdy } e > 0 \\ -1 & \text{gdy } e \le 0 \end{cases}$$

Funkcje signum:

$$y = \begin{cases} 1 & \text{gdy } e > 0 \\ 0 & \text{gdy } e = 0 \\ -1 & \text{gdy } e < 0 \end{cases}$$

Funkcja SPR
(Spatio-Temporal Pattern Recognition)

$y^{(j+1)} = y^{(j)} + A [-a y^{(j)} + b e^{+}]$

gdzie "funkcja ataku"

$$A[u] = \begin{cases} u & \text{gdy } u > 0 \\ \gamma u & \text{gdy } u \le 0 \end{cases}$$

zapis e^{+} oznacza

$$e^{+} = \begin{cases} e & \text{gdy } e > 0 \\ 0 & \text{gdy } e \le \end{cases}$$

Funkcja skoku jednostkowego:

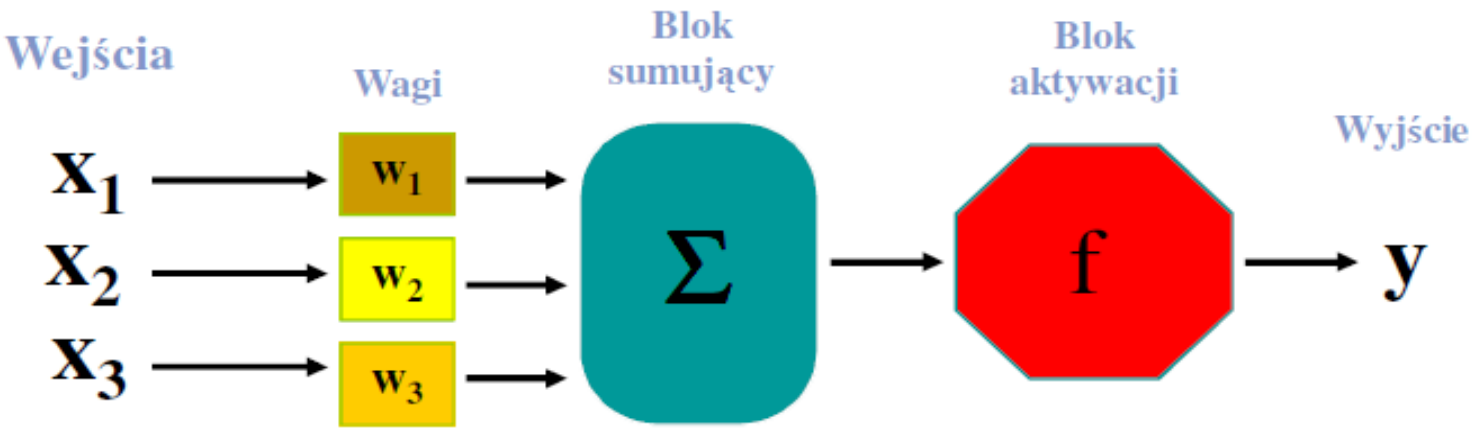
$$y = \begin{cases} 1 & \text{gdy } e > 0 \\ 0 & \text{gdy } e \le 0 \end{cases}$$

Funkcja BAM
(Bidirectorial Associative Memory)

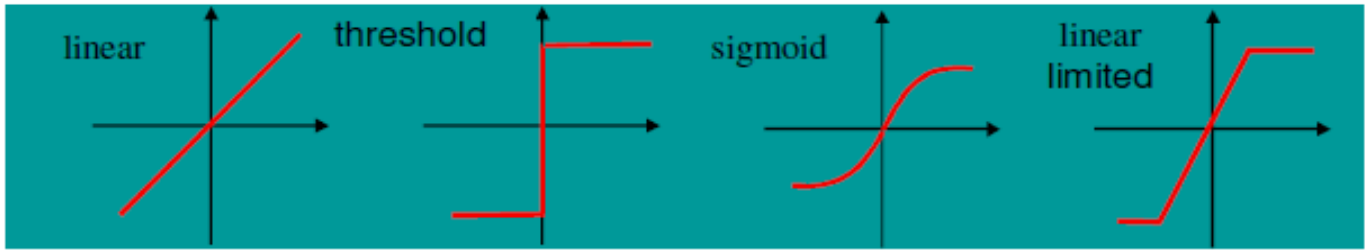
$$y^{(j+1)} = \begin{cases} 1 & \text{gdy } e > 0 \\ y^{(j)} & \text{gdy } e = 0 \\ -1 & \text{gdy } e < 0 \end{cases}$$

Powyższe funkcje są "inżynierskie": opis który pozwala na wygodną analizę matematyczną, łatwą realizację techniczną (perceptron) lub wygodne modelowanie w formie programu symulacyjnego (signum).

Sztuczny neuron sigmoidalny



$$y = f\ (\ \Sigma\ (\ x_i * w_i\ ))$$



f – funkcja nieliniowa

Uczenie nieliniowego neuronu

Rozważmy problem **uczenia** nieliniowych sieci neuronowych. Dla uproszczenia analizujemy wyłącznie **regułę DELTA** w jej podstawowej postaci.

$$\mathbf{U} = << \mathbf{X}^{(1)}, \mathbf{z}^{(1)} >, < \mathbf{X}^{(2)}, \mathbf{z}^{(2)} >, < \mathbf{X}^{(3)}, \mathbf{z}^{(3)} >, ..., < \mathbf{X}^{(N)}, \mathbf{z}^{(N)} >>$$

Formułę uczenia opieramy na **regule minimalizacji** funkcjonału błędu średniokwadratowego:

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

gdzie

$$\mathbf{y}^{(\mathbf{i})} = \phi(\mathbf{\Sigma}_{\mathbf{j}=1}^{\mathbf{N}} \omega_{\mathbf{i}}^{(\mathbf{j})} \mathbf{x}_{\mathbf{i}}^{(\mathbf{j})})$$

Uczenie nieliniowego neuronu

Rozkładając funkcjonał błędu na elementy składowe związane z poszczególnymi krokami procesu uczenia

$$\mathbf{Q} = \Sigma \mathbf{Q}(\mathbf{j}) \quad \text{gdzie} \quad \mathbf{Q}(\mathbf{j}) = \frac{\mathbf{1}}{\mathbf{2}}(\mathbf{z}(\mathbf{j}) - \mathbf{y}(\mathbf{j}))^{\mathbf{2}}$$

Możemy zgodnie z **gradientową strategią** procesu uczenia zapisać algorytm zmian czynników wag

$$\omega_{\mathbf{i}}(\mathbf{j}+\mathbf{1}) - \omega_{\mathbf{i}}(\mathbf{j}) = \Delta\omega_{\mathbf{i}}(\mathbf{j}) = -\eta\, \partial \mathbf{Q}(\mathbf{j}) / \partial \omega_{\mathbf{i}}$$

Uczenie nieliniowego neuronu

Analogiczny wzór wyprowadzono wcześniej dla sieci **ADALINE**, jednak treść tego wzoru jest w tym wypadku bogatsza ze względu na nieliniową funkcję $\phi(e)$.

$$\partial \mathbf{Q}^{(\mathbf{j})} / \partial \omega_{\mathbf{i}} = \partial \mathbf{Q}^{(\mathbf{j})} / \partial \mathbf{y}_{\mathbf{i}}^{(\mathbf{j})} \partial \mathbf{y}_{\mathbf{i}}^{(\mathbf{j})} / \partial \omega_{\mathbf{i}}$$
$$= \partial \mathbf{Q}^{(\mathbf{j})} / \partial \mathbf{y}_{\mathbf{i}}^{(\mathbf{j})} \ \partial \mathbf{y}_{\mathbf{i}}^{(\mathbf{j})} / \partial \mathbf{e}^{(\mathbf{j})} \ \partial \mathbf{e}^{(\mathbf{j})} / \partial \omega_{\mathbf{i}}$$

łatwo możemy obliczyć:

$$\partial \mathbf{Q}^{(\mathbf{j})} / \partial \mathbf{y}_{\mathbf{i}}^{(\mathbf{j})} = -(\mathbf{z}^{(\mathbf{j})} - \mathbf{y}^{(\mathbf{j})}) = -\delta^{(\mathbf{j})}$$

$$\partial \mathbf{e}^{(\mathbf{j})} / \partial \omega_{\mathbf{i}} = \mathbf{x}_{\mathbf{i}}^{(\mathbf{j})}$$

Uczenie nieliniowego neuronu

Problem może być natomiast z wyrażeniem

$$\partial \mathbf{y_i^{(j)}} / \partial \mathbf{e^{(j)}} = \partial \phi(\mathbf{e}) / \partial \mathbf{e^{(j)}}$$

gdzie $\phi(e)$ nie zawsze jest różniczkowalne.

Ostateczny wzór, na podstawie którego prowadzi się proces uczenia ma postać

$$\mathbf{\Delta} \omega_\mathbf{i}^{(\mathbf{j})} = -\eta \; \delta^{(\mathbf{j})} \partial \phi(\mathbf{e}) / \partial \mathbf{e^{(j)}} \mathbf{x_i^{(j)}}$$

Uczenie nieliniowego neuronu

Dość chętnie (bezkrytycznie) stosuje się w rozważaniach funkcje logistyczną

$$\mathbf{y} = \phi(\mathbf{e}) = \mathbf{1}/(\mathbf{1} + \mathbf{exp}(-\beta\mathbf{e}))$$

która ma łatwą postać pochodnej,

$$\partial\phi(\mathbf{e})/\partial\mathbf{e}^{(\mathbf{j})} = \mathbf{y}^{(\mathbf{j})}(\mathbf{1} - \mathbf{y}^{(\mathbf{j})})$$

Ostateczny wzór dla funkcji logistycznej może być zapisany w prostszej postaci

$$\mathbf{\Delta}\omega_{\mathbf{i}}^{(\mathbf{i})} = -\eta(\mathbf{z}^{(\mathbf{j})} - \mathbf{y}^{(\mathbf{j})})(\mathbf{1} - \mathbf{y}^{(\mathbf{j})})\mathbf{y}^{(\mathbf{j})}\mathbf{x}_{\mathbf{i}}^{(\mathbf{j})}$$

Powyższy algorytm uczenia jest możliwy do bezpośredniego zastosowania jedynie w przypadku ***sieci jednowarstwowej***.

Uczenie sieci nieliniowej

Dla ***sieci wielowarstwowych***, które mają istotnie szersze możliwości przetwarzania informacji niż sieci jednowarstwowe, omawiany poprzednio wzór nie daje się zastosować. Dla ***warstw wewnętrznych*** nie ma możliwości bezpośredniego określenia oczekiwanych (wymaganych) wartości sygnałów wejściowych $z^{(j)}$, a tym samych określenia wartości błędu $\delta^{(j)}$. Rozważając ciąg

$$U = << X^{(1)}, z^{(1)} >, < X^{(2)}, z^{(2)} >, < X^{(3)}, z^{(3)} >, ..., < X^{(N)}, z^{(N)} >>$$

mamy do dyspozycji n-wymiarowe wektory wejściowe **X** oraz k-wymiarowe wektory wyjściowe **Z** z neuronów terminalnych.

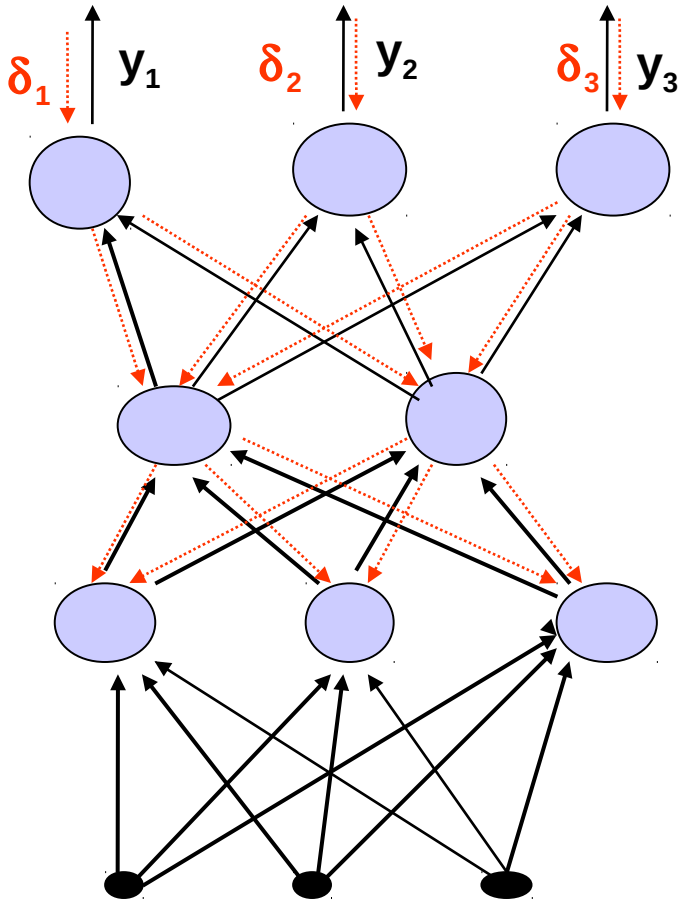
Uczenie sieci nieliniowej

Jeżeli odnotujemy błąd, czyli różnice $(X^{(j)} - Z^{(j)})$, to nie będziemy w stanie ustalić w jaki sposób za pojawienie się błędu odpowiadają neurony warstwy wyjściowej a jaki sposób powstał w elementach wcześniejszych (wewnętrznych) warstw. Noszą one nazwę ***warstw ukrytych***, “hidden layers”.

Przez wiele lat nie było dobrego pomysłu w jaki sposób uczyć warstwy ukryte.

Uczenie sieci nieliniowej

W latach 80-tych zaproponowano algorytm tzw. ***wstecznej propagacji błędów*** (backpropagation), polegający na tym że mając wyznaczony błąd $\delta^{(m)(j)}$ (**j-ty** krok uczenia **m-tego** neuronu) możemy “rzutować” ten błąd wstecz do wszystkich tych neuronów, których sygnały stanowiły wejścia do **m-tego** neuronu.



Uczenie sieci nieliniowej

Przy założeniu ciągłości funkcji kryterialnej, najskuteczniejszymi metodami uczenia pozostają **gradientowe metody optymalizacyjne,** w których uaktualnianie wektora wag odbywa się zgodnie ze wzorem

$$W(k+1) = W(k) + \Delta W$$

gdzie

$$\Delta W = \eta\ p(W)$$

η = wspolczynnik uczenia
$p(W)$ = kierunek w przestrzeni wielowymiarowej W

Uczenie przy zastosowaniu metod gradientowych wymaga wyznaczenia kierunku **p(W),** czyli wyznaczenia wektora gradientu względem wszystkich wag sieci.

∇ **- oznacza gradient**

$$\Delta W = - \eta\ \nabla(W)$$

Uczenie sieci nieliniowej

Na samym początku wyznacza się poprawki dla neuronów stanowiących wyjściową warstwę sieci. Dla poszczególnych sygnałów $y_m^{(j)}$ istnieją w ciągu uczącym wzorcowe (oczekiwane) wartości $z_m^{(j)}$, z którymi można je porównywać, wyznaczając bezpośrednio błąd $\delta_m^{(j)}$.

$$\delta_m^{(j)} = (y_m^{(j)} - z_m^{(j)})$$

$$\Delta\omega_i^{(m)(j)} = \eta\, \delta_m^{(j)}\, d\phi(e)/de_m^{(j)}\, y_i^{(j)}$$

dla funkcji logistycznej wzór ten ulega znacznemu uproszczeniu:

$$\Delta\omega_i^{(m)(j)} = -\eta\, (z_m^{(j)} - y_m^{(j)})\, (1 - y_m^{(j)})\, y_i^{(j)}\, y_m^{(j)}$$

Uczenie sieci nieliniowej

Dla warstwy ukrytej, przez analogię możemy zapisać

$$\Delta\omega_i^{(m)(j)} = \eta\, \delta_m^{(j)}\; d\phi(e)/de_m^{(j)}\, y_i^{(j)}$$

ale teraz nie mamy możliwości bezpośrednego wyznaczenia $\delta_m^{(j)}$.

Załóżmy, że rozważany neuron należy do warstwy ukrytej, ale sygnały od niego docierają tylko do warstwy wyjściowej (dla której potrafimy określić $\delta_k^{(j)}$). Wówczas (***backpropagation***)

$$\delta_m^{(j)} = \Sigma\ \ \omega_m^{(k)(j)}\, \delta_k^{(j)}$$

$\omega_m^{(k)(j)}$ – waga w neuronie o numerze **k**, przy jego wejściu **m**

Rzutowane wstecznie błędy przemnażane są przez te same współczynniki, przez które mnożone były sygnały, tyle tylko, że kierunek przesyłania informacji zostaje w tym przypadku odwrócony; zamiast od wejścia do wyjścia przesyła się je od wyjścia kolejno w kierunku wejścia.

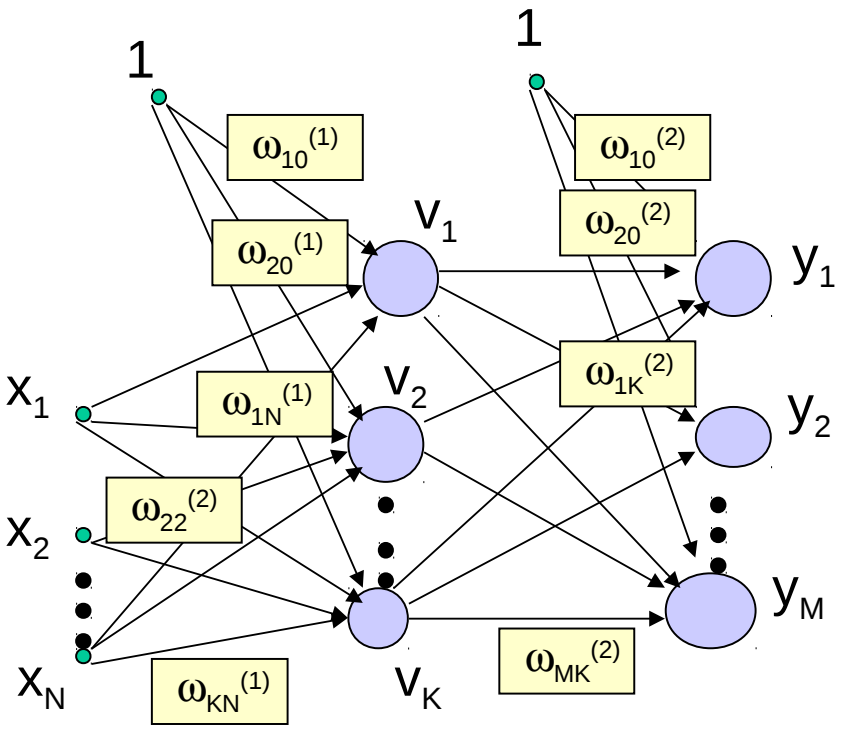
Uczenie sieci nieliniowej

- Powyższą **technikę propagacji wstecznej błędów** należy powtarzać dla kolejno coraz głębszej warstwy sieci.

- Każdy **neuron z warstwy ukrytej** albo przesyła sygnały do wartości wyjściowych, albo znajduje się w jednej z głębszych warstw, wówczas jego błąd można oszacować z chwilą określenia błędów dla wszystkich neuronów w sieci które są odbiorcą jego sygnałów.

- **Uaktualnianie wag** może odbywać się po każdorazowej prezentacji próbki uczącej lub jednorazowo (w sposób skumulowany) po prezentacji wszystkich próbek tworzących cykl uczący.

Uczenie sieci nieliniowej



W sieci ***feedforward*** zawsze daje się określić taką kolejność wstecznej propagacji błędów, która pozwala obliczyć błędy dla wszystkich elementów sieci.

Uczenie tą metoda jest stosunkowo skuteczne ale powolne.

Propagacja wsteczna

Algorytm podstawowy jest wolnozbieżny w sieci wielowarstwowej, liczne wariacje proponują jak go przyspieszyć. Inne cele modyfikacji dotyczą unikania minimów lokalnych i poprawy zdolności generalizacji.

Istnieje **wiele parametrów** które można rozpatrywać w zakresie usprawnienia ogólnej metody propagacji wstecznej, włącznie z **architekturą** (liczbą warstw, liczbą jednostek w warstwie), **wielkością i naturą zbioru treningowego** i **regułą modyfikacji wag**.

Dyskusja dotyczyć będzie przede wszystkim zmian reguł modyfikacji, utrzymując architekturę bez zmian.

Metody gradientowe

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

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

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

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

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

Funkcje kryterialna

Kwadratowa funkcja kryterialna,

$$Q = \frac{1}{2}\Sigma[y - z]^2$$

Nie jest jedyną możliwością.

Możemy zastąpić czynnik $(y-z)^2$ **dowolną inną funkcją różniczkowalną**, która ma minimum jeżeli jej argumenty są sobie równe, oraz możemy wprowadzić odpowiednią regułę modyfikacji.

Bezpośrednie różniczkowanie wskazuje, że tylko wyrażenie bezpośrednio definiujące $\delta^{(m)}$ w warstwie wyjściowej zmienia się, a wszystkie pozostałe równania propagacji wstecznej pozostają bez zmian.

Do funkcji kryterialnej można też dodać parametry zmieniające jej stromość lub pofałdowanie w procesie uczenia.

Spadek gradientu może być bardzo powolny, jeżeli η jest małe, i może mieć duże oscylacje jeżeli η jest duże.

Składnik momentu w funkcji kryterialnej

Najprostsze podejście to dodanie momentu, pomysł polega na **nadaniu każdej wadze pewnej bezwładności**, czyli momentu, w wyniku czego ma ona skłonność do zmian **"kierunku średniej "** zamiast popadania w oscylacje przy każdym małym impulsie.

$$\Delta \omega_{pq}^{(j+1)} = -\eta \, \partial Q/\partial \omega + \alpha \, \Delta \omega_{pq}^{(j)}$$

Jeżeli $\partial Q/\partial \omega$ jest prawie stałe to wówczas

$$\Delta \omega_{pq}^{(j+1)} = -\eta/(1-\alpha) \, \partial Q/\partial \omega \, .$$

Przy wartości współczynnika **$\alpha = 0.9$ oznacza to 10-krotny wzrost efektywności** wartości współczynnika uczenia, a więc przyspieszenie.

Algorytm największego spadku

Nie jest łatwo wybrać **odpowiednie wartości parametrów** η i α dla określonego problemu. Ponadto najlepsze wartości w początkowej fazie uczenia mogą nie być dość dobre później. Spadek gradientu może być bardzo powolny, jeżeli η jest małe, i może mieć duże oscylacje jeżeli η jest duże.

Ważne jest aby **czynnik momentu nie stał się dominujący** w procesie uczenia, gdyż prowadziłoby to do niestabilności algorytmu.

Zwykle kontroluje się zmiany wartości funkcji celu w procesie uczenia, dopuszczając do jej **wzrostu jedynie w ograniczonym zakresie**, np. 5%. Jeżeli kolejna iteracja nie spełnia tego warunku to krok jest pomijany. W tym momencie **składnik gradientowy odzyskuje dominację ponad składnikiem momentu** i proces przebiego zgodnie z kierunkiem minimalizacji wyznaczonym przez wektor gradientu.

Algorytm zmiennej metryki

W metodzie tej wykorzystuje się kwadratowe przybliżenie funkcji kryterialnej **Q(W)** w sasiedztwie znanego rozwiązania W_k.

Możemy więc rozwinąć:

$$Q(W+p) = Q(W) + [g(W)^T]p + 1/2p^T H(W)p$$

Gdzie $g(W) = \nabla Q = [\ \partial Q/\partial \omega_1, \partial Q/\partial \omega_n, ..., \partial Q/\partial \omega_n\ ]^T$ jest wektorem gradientu, a symetryczna macierz kwadratowa **H(W)** jest macierza drugich pochodnych $(\partial^2 Q/(\partial w_1\, \partial w_2))$.

Minimum funkcji wymaga aby $dQ(W_k+p)/dp = 0$. Dokonując odpowiedniego różniczkowania otrzymuje się

$$g(W_k)+H(W_k)p_k=0$$

A więc ostatecznie

$$p_k= -[H(W_k)]^{-1}\, g(W_k)$$

Algorytm zmiennej metryki

W praktyce wyznaczenie **hesjanu H(W)** w każdym kroku nie jest stosowane, stosuje się **przybliżenie hesjanu G(W)** oparte na znajomości wartości **gradientu g(W)**, tak aby

$$G(W_k)\,(W_k - W_{k-1}) = g(W_k) - g\,(W_{k-1})$$

Oznaczmy: $s_k = W_k - W_{k-1}$, $r_k = g(W_k) - g(W_{k-1})$,

Macierz odwrotna przyblizonego hesjanu:

$$V_k = [G(W_k)]^{-1}, \quad V_{k-1} = [G(W_{k-1})]^{-1}$$

To wówczas proces uaktualniania wartości macierzy V_k opisuje się zależnością rekurencyjną

$$V_k = V_{k-1} + (1 + r_k^T V_{k-1} r^k)/(s_k^T r_k) - s_k r_k^T V_{k-1} + V_{k-1} r_k s_k^T/(s_k^T r_k)$$

Algorytm zmiennej metryki

Metoda zmiennej metryki charakteryzuje się **zbieżnością superliniową.** Jest więc **znacznie lepsza od liniowo zbieżnej metody największego spadku**. Fakt, że hesjan w każdym kroku spełnia warunek dodatniej określoności daje pewność, że spełnienie warunku $g(W_k) = 0$ odpowiada **rozwiazaniu problemu optymalizacji**.

Metoda ta jest uważana za jedną z najlepszych metod optymalizacji funkcji wielu zmiennych. Jej **wadą jest stosunkowo duża złożoność** obliczeniowa (konieczność **wyznaczenia n^2** elementów hesjanu), a także **duże wymagania co do pamięci** przy przechowywaniu macierzy hesjanu. Z tego względu **stosuje się ją do niezbyt dużych sieci.**

Metoda gradientów sprzężonych

W metodzie tej podczas wyznaczania kierunku minimalizacyjnego **rezygnuje się z bezpośredniej informacji o hesjanie**. Kierunek poszukiwań p_k jest konstruowany w taki sposób, aby był ortogonalny oraz sprzężony ze wszystkimi poprzednimi kierunkami $p_0, p_1, \ldots, p_{k-1}$, tzn. $p^T G p_j = 0 \quad i \neq j$

Wektor p_k ktory spełnia powyższe założenia ma postać:

$$p_k = - g_k + \sum \beta_{kj} p_j$$

Przy czym $g_k = g(W_k)$ oznacza aktualną wartość wektora gradientu, a sumowanie dotyczy poprzednich kierunków minimalizacyjnych. Korzystając z warunku ortogonalności oraz uwzględniając sprzeżenie między wektorami, możemy zapisać:

$$p_k = -g_k + \beta_{k-1} p_{k-1}$$

Metoda gradientów sprzężonych

Współczynnik sprzężenia odgrywa bardzo ważną rolę, kumulując w sobie informację o poprzednich kierunkach poszukiwań. Istnieje wiele odmiennych reguł wyznaczania tego współczynnika.
Najbardziej znane z nich to:

$$\beta_{k-1} = g_k^T (g_k - g_{k-1}) / (g^T_{k-1}\, g_{k-1})$$

$$\beta_{k-1} = g_k^T (g_k - g_{k-1}) / (-p_{k-1}\, g_{k-1})$$

Ze względu na **kumulację błędów zaokrąglenia** podczas kolejnych cykli obliczeniowych metoda gradientów sprzężonych w praktyce zatraca właściwość ortogonalności między wektorami kierunków minimalizacyjnych. Po n-iteracjach przeprowadza się ponowny start. Metoda wykazuje zbieżność zbliżoną do liniowej, jest mniej skuteczna niż metoda zmiennej metryki, ale zdecydowanie szybsza niż metoda największego spadku.
Stosuje się ją jako skuteczny algorytm przy dużej liczbie zmiennych.

Metody doboru współczynnika uczenia

Po określeniu właściwego kierunku p_k i wyborze na nim nowego rozwiązania W_{k+1}, należy tak dobrać wartość η_k, aby nowy punkt rozwiązania $W_{k+1} = W_k + \eta_k p_k$ leżał możliwie blisko minimum funkcji $Q(W)$ na kierunku p_k.

Właściwy wybór współczynnika η_k ma ogromny wpływ na zbieżność algorytmu optymalizacji do minimum funkcji celu.

$\rightarrow$ przyjęcie zbyt małej wartości η powoduje niewykorzystanie możliwości zminimalizowania wartości funkcji celu w danym kroku i konieczność jego powtórzenia w następnym.

$\rightarrow$ zbyt duży krok powoduje "przeskoczenie" minimum funkcji i podobny efekt jak poprzednio.

Empirycznym "przepisem" stosowanym w sieciach neuronowych jest dobór

$$\eta < \min (1/n_i)$$

gdzie n_i oznacza liczbe wejść i-tego neuronu.

Zestaw pytań do testu

- ✓ Co to znaczy wsteczna propagacja błędów?
- ✓ Podaj znane ci nazwy metod do uczenia wg. techniki propagacji wstecznej
- ✓ Czy metody wstecznej propagacji błędów stosujemy do uczenia sieci liniowych?

PERCEPTRON ... raz jeszcze ...

Dygresja:

Zależnie od postaci przyjętej funkcji **φ(e)** sygnał **y** można rozpatrywać jako

binarny	$y \in \{ 0, 1 \}$
bipolarny	$y \in \{ -1, 1 \}$

Pozornie różnica jest nieistotna, trywialne przeskalowanie. Może mieć jednak poważne konsekwencje ponieważ punkty należące do zbioru **{ 0,1}** są wierzchołkami jednostkowego **hiperszescianu w R^n**, natomiast punkty należące do **zbioru {-1,1}** leżą na powierzchni jednostkowej **sfery R^n**.
W **n**-wymiarowej przestrzeni sześcian i sfera różnią się w sposób zasadniczy.
Porównajmy objętości:

objetość szescianu: $V_s = a^n$

objetość kuli: $V_k = \pi^{n/2} / (n/2)!\ r^n$ gdy n jest parzyste

$V_k = 2^n\ \pi^{(n-1)/2}((n-1)/2)! / n!\ r^n$ gdy n jest nieparzyste

PERCEPTRON ... raz jeszcze ...

Tak więc dla jednostkowego boku a, objętość sześcianu jest stała $V_s = 1$, podczas gdy objętość kuli o jednostkowym promieniu r, $V_k \to 0$ dla $n \to \infty$.

Wszystkie punkty sfery są oczywiście jednakowo odległe od jej środka (odległością jest promień sfery), natomiast dla sześcianu, narożniki są odległe od środka o sqrt(n/2) (odległość rośnie).
Sześcian coraz bardziej przypomina "jeża".

W większych wymiarach..... należy dość ostrożnie podchodzić do intuicji geometrycznych.

Czasami warto jest przejść do układu w ktorym neuron przyjmuje wartość {-1, +1}. Wtedy sieć staje się podobna do układu magnetycznego, w którym momenty magnetyczne atomów mogą mieć dwa przeciwne kierunki. W opisie takich sieci można stosować metody z teorii układów magnetycznych.