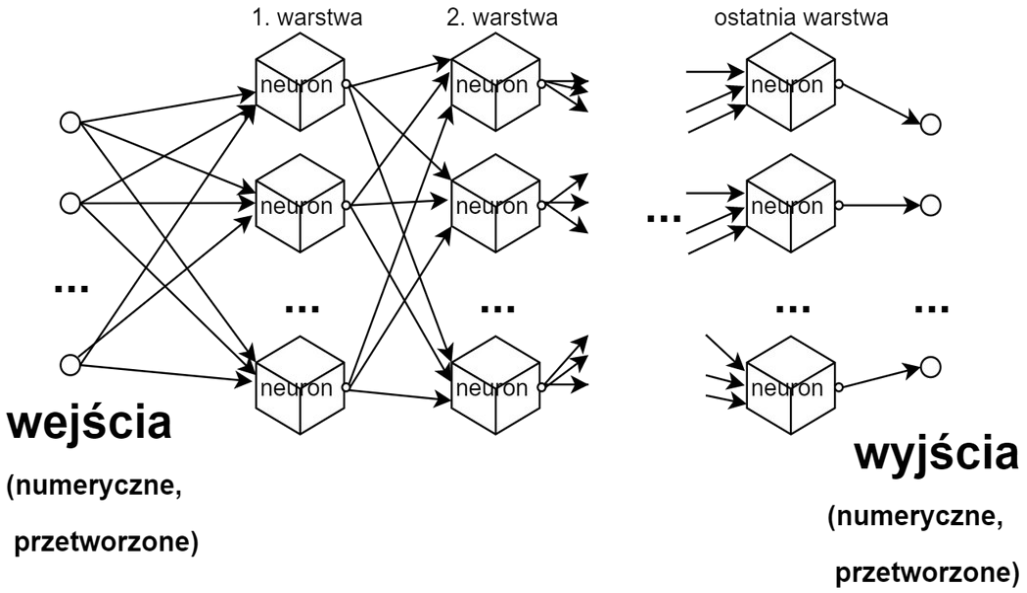
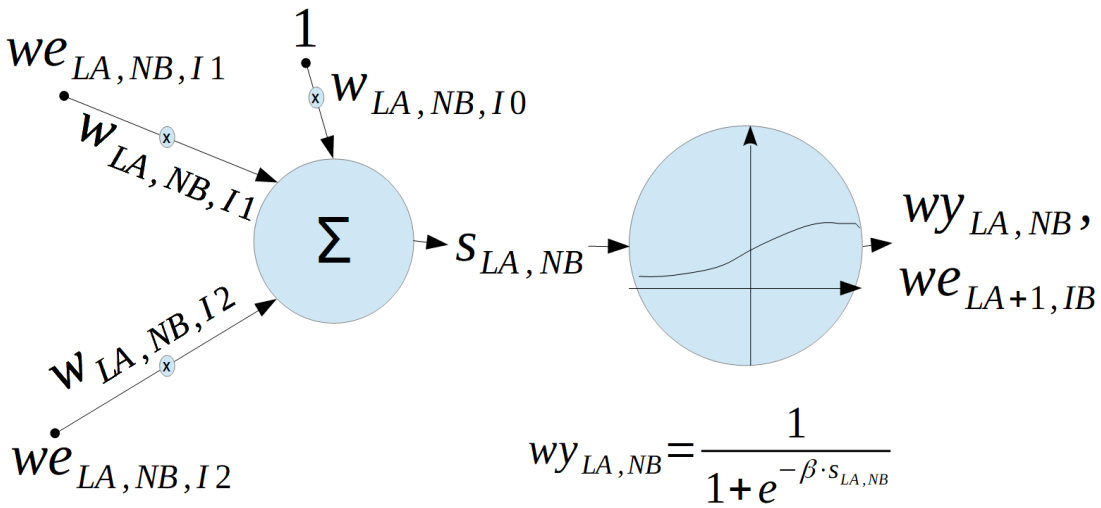
Wielowarstwowa sieć neuronowa, algorytm wstecznej propagacji błędów

Sieci neuronowe wielowarstwowe

Jedną najczęściej wykorzystywanych struktur w inteligencji obliczeniowej na początku tego wieku była wielowarstwowa sieć neuronowa. Najczęściej neurony są ułożone w kolejne warstwy (na rysunku jedna kolumna neuronów to jedna warstwa) ułożone od strony lewej do prawej zgodnie z rysunkiem widocznym poniżej.



Najczęściej używane były neurony sigmoidalne, które mają kilka wejść plus jedno wejście ukryte oraz dokładnie jedno wyjście. Zwykle każdy neuron warstwy pierwszej podłączony jest z każdym wejściem sieci, a wejściami każdego z neuronów warstwy drugiej i kolejnych są wszystkie wyjścia neuronów z warstwy wcześniejszej. Wyjścia neuronów ostatnich warstw są jednocześnie wyjściami globalnymi sieci. Oto schemat budowy ***B-tego*** neuronu sigmoidalnego unipolarnego należącego do ***A*-tej** warstwy posiadającego 2 wejścia jawne + 1 ukryte:



$$wy_{LA,NB}=\frac{1}{1+e^{-\beta\cdot s_{LA,NB}}}$$

$$S_{LA,NB}=w_{LA,NB,I0}+\sum_{i=1,\dots} we_{LA,I1}*w_{LA,NB,I1}$$

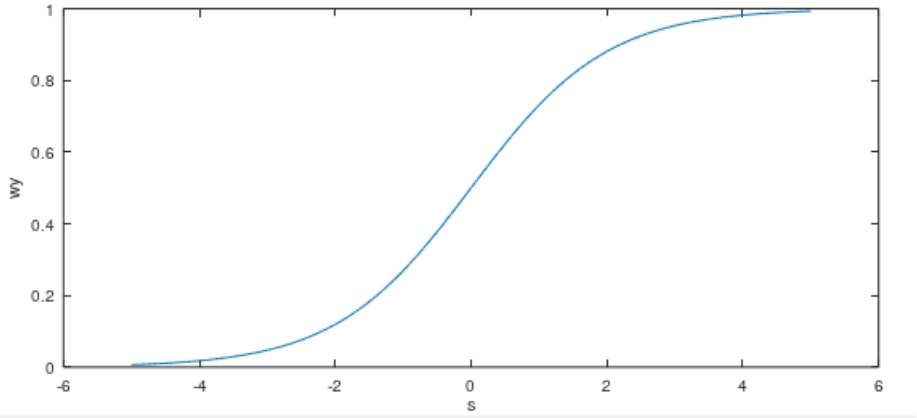
Warto zauważyć, że wyjście aktualnego neuronu należącego do nieostatniej warstwy jest jednocześnie wejściem neuronu z kolejnej warstwy. Oto wzory użyte do wyliczenia odpowiednich wartości:

$S_{LA,NB}=w_{LA,NB,I0}+\sum_{i=1,\dots} we_{LA,I1}*w_{LA,NB,I1}$, $wy_{LA,NB}=\frac{1}{1+e^{-\beta\cdot s_{LA,NB}}}$,

gdzie $S_{LA,NB}$ to suma aktywacji *B*-tego neuronu należącego do *A*-tej warstwy, $w_{LA,NB,I0}$ to waga dla

wejścia ukrytego B-tego neuronu będącego w A-tej warstwie, $w_{LA,NB,I1}$ to waga dla 1-go wejścia B-tego neuronu będącego w A-tej warstwie, i to indeks wejścia danego neuronu, $wy_{LA,NB}$ to wyjście B-tego neuronu w A-tej warstwie, β to parametr określający kształt funkcji aktywacji domyślnie równy 1, $\beta>0$.

Wzór $wy_{LA,NB}=\frac{1}{1+e^{-\beta \cdot s_{LA,NB}}}$ określa funkcję aktywacji neuronu i jest charakterystyczny dla unipolarnego neuronu sigmoidalnego. Wartość wyjściowa mieści się w zakresie $(0;1)$. Warto zwrócić uwagę na wartość pochodnej $\frac{\partial wy_{LA,NB}}{\partial s_{LA,LB}}=\beta \cdot wy_{LA,NB} \cdot (1-wy_{LA,NB})$. Oto kształt tej funkcji dla $\beta=1$:



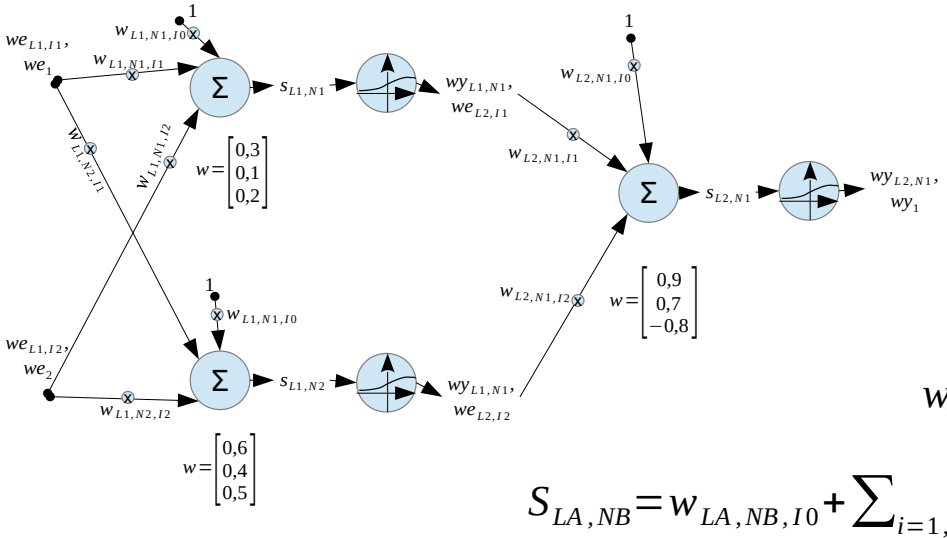
Wartości wejściowe i wyjściowe tej sieci neuronowej są liczbami rzeczywistymi, a zakładając użycie tej funkcji aktywacji wartości wyjściowe tej sieci mieszczą się w przedziale (0; 1).

Zwracanie wartości wyjściowej

Aby zwrócić wartości wyjściowe najpierw podawane są wartości na wejściach sieci, a następnie uruchamiane są kolejne warstwy sieci. Wyjścia neuronów należących do warstwy nie ostatniej służą jako wejścia dla neuronów z kolejnej warstwy, a wyjścia z warstwy ostatniej służą jako wartości wyjściowe całej sieci neuronowej.

Poniżej znajduje się przykład działania dla sieci neuronowej składającej się z 2+1 neuronów, 2 wejść o wartościach [1; 0] i jednego wyjścia, $\beta=1$.

Oto stan sieci przed uruchomieniem sieci (wagi początkowe zostały dobrane przypadkowo):



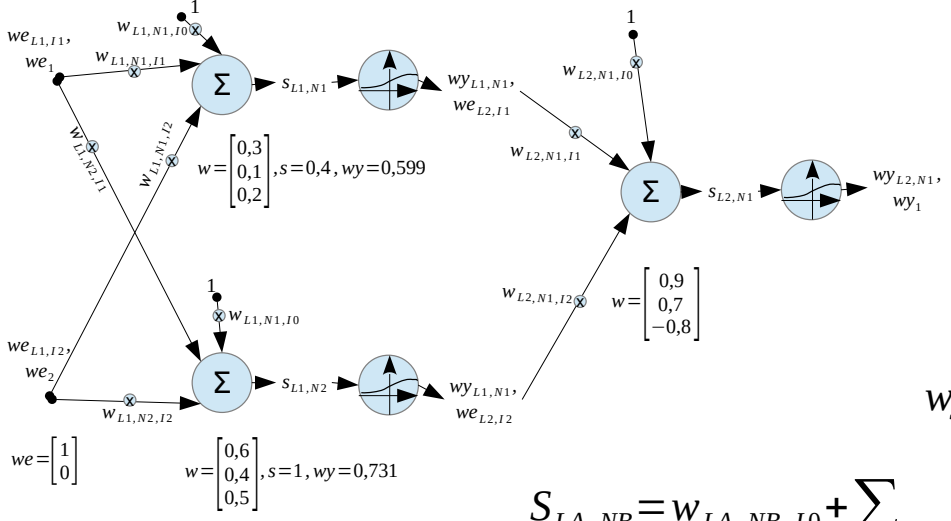
$\beta=1$

$$wy_{LA,NB}=\frac{1}{1+e^{-\beta \cdot s_{LA,NB}}}$$

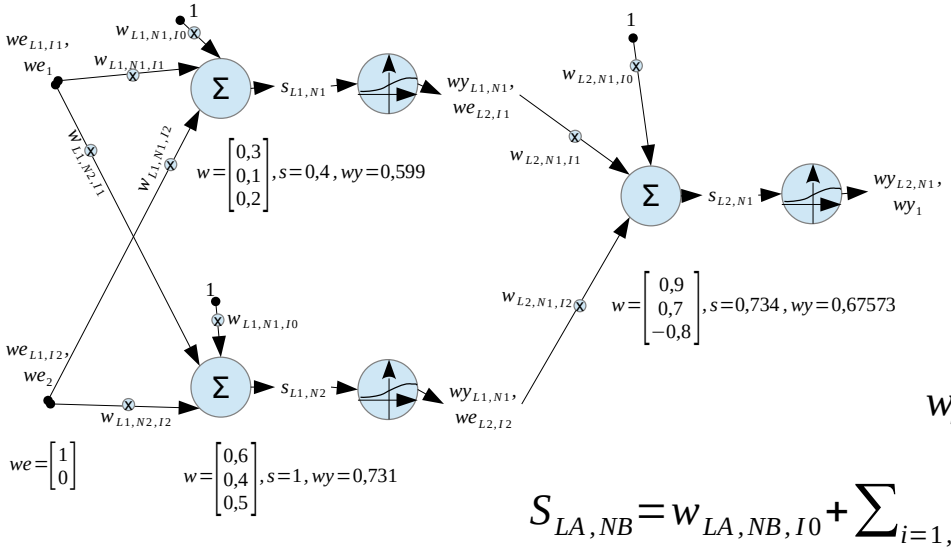
$$S_{LA,NB}=w_{LA,NB,I0}+\sum_{i=1,\ldots} we_{LA,I1} * w_{LA,NB,I1}$$

,

Oto stan sieci po uruchomieniu pierwszej warstwy:



Oto stan sieci po uruchomieniu drugiej (ostatniej) warstwy:



Jak widać po podaniu próbki podaniu wartości wejściowych [1; 0] sieć zwróciła wartość około 0,676.

Wsteczna propagacja błędów

Algorytm ten jest głównym algorytmem uczenia sieci neuronowych. Najpierw należy zademonstrować, jak będzie wyglądało wykorzystanie tego algorytmu:

Użycie algorytmu wstecznej propagacji błędów

Aby nauczyć sieć neuronową przy użyciu tego algorytmu trzeba wykonać następujące czynności:

1. Stwórz sieć neuronową z wybraną lub zadaną liczbą neuronów w poszczególnych warstwach, wylosuj wagi początkowe.
2. Zgromadź próbki uczące i na podstawie każdej z nich stwórz parę {wartości na wejściach sieci; wartości na wyjściach sieci}. Na przykład dla problemu xor próbki to [1 0 1], [1 1 0], [0 1 1], [1 1 1], a pary {[1 0]; 1}, {[1 1]; 0}, {[0 1]; 0}, {[1 1]; 1}
3. Ucz sieć przez zadaną liczbę epok
    1. Zmień losowo kolejność próbek i tym samym par {wartości na wejściach sieci; wartości na wyjściach sieci}.
    2. Ucz sieć algorytmem wstecznej propagacji błędów za pomocą kolejnych próbek, czyli par {wartości na wejściach sieci; wartości na wyjściach sieci}.

Na przykład dla problemu xor, gdyby liczba epok wynosiła 2, to sieć mogłaby być uczona za pomocą następujących sygnałów:

Epoka 1-sza:

1. wartości wejściowe: [1; 0], pożądana wartość wyjściowa: 1,

2. wartości wejściowe: [0; 0], pożądana wartość wyjściowa: 0,
3. wartości wejściowe: [1; 1], pożądana wartość wyjściowa: 0,
4. wartości wejściowe: [0; 1], pożądana wartość wyjściowa: 1,

   Epoka 2-ga:

5. wartości wejściowe: [0; 1], pożądana wartość wyjściowa: 1,
6. wartości wejściowe: [1; 0], pożądana wartość wyjściowa: 1,
7. wartości wejściowe: [1; 1], pożądana wartość wyjściowa: 0,
8. wartości wejściowe: [0; 0], pożądana wartość wyjściowa: 0.

Taka sieć byłaby uczona algorytmem wstecznej propagacji błędów 2*4 razy.

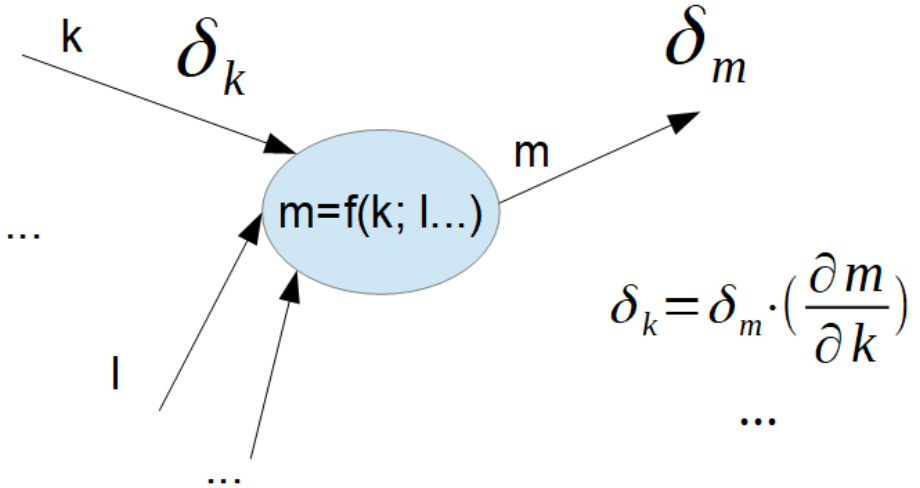
Właściwy algorytm wstecznej propagacji błędów

Polega on na uczeniu sieci pojedynczą próbką uczącą, czyli parą {wartości na wejściach sieci; wartości na wyjściach sieci}. Wartości wejściowe tej próbki podawane są na wejścia sieci, a wartości wyjściowe próbki widoczne są używane jako wartości pożądane (zadane) i na ich podstawie sieć jest uczona. Uczenie w podstawowej wersji sterowane jest za pomocą parametru $\mu, \mu \in (0;1)$, który domyślnie wynosi 0,1. Oto schemat uczenia za pomocą tego algorytmu.

- Podaj wartości na wejścia sieci i wylicz wartości wyjściowe.
- Wylicz poprawki na wyjściach stosując wzór $\delta_{numerWyjścia} = \mu \cdot (d_{numerWyjścia} - wy_{numerWyjścia})$, gdzie $\delta_{numerWyjścia}$ to wyliczona korekta dla danego wyjścia, μ to parametru uczenia, $d_{numerWyjścia}$ to wartość pożądana dla tego wyjścia (wynika z wartości parametrów wyjściowych próbki uczącej), $wy_{numerWyjścia}$ to otrzymana wartość wyjściowa całej sieci w poprzednim kroku.
- Dla każdego neuronu w ostatniej warstwie wylicz poprawki w kolejnych miejscach sieci, tak aby udało się doprowadzić poprawki od wyjścia neuronów (będące jednocześnie wyjściami całej sieci), aż po wagi i wejścia. Szczegóły będą podane później.
- Czynność powtórz dla pozostałych warstw rozpoczynając od warstwy przedostatniej a kończąc na warstwie pierwszej.
- Zwiększ wartość wszystkich wag sieci o wyliczone wcześniej poprawki.
- Krok niekonieczny. Ponownie wylicz wartości wyjściowe dla tej samej próbki. Moduł błędu $|d_{numerWyjścia} - wy_{numerWyjścia}|$ w większości przypadków powinien się zmniejszyć w stosunku do tego przed uczeniem. Ten krok jest bardzo istotny do samokontroli i powinien być zaimplementowany w pierwszej wersji algorytmu.

Należy zwrócić uwagę, że propagowanie poprawek dokonywane jest wstecz, a wzór podobny jest do metody rozwiązywania pochodnych przez wstawianie.

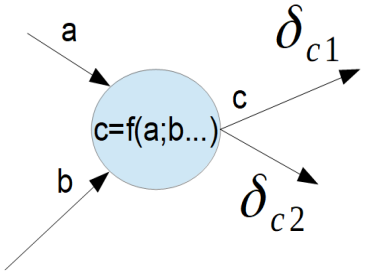
Poniższy przykład pokazuje ogólny sposób propagacji wstecz poprawek, od poprawki znanej dla *m*, do poprawki dla *k*.



Jeśli *m* wynika z *k* to wartość poprawki δ_k jest równa iloczynowi poprawki δ_m i pochodnej *m* względem

k, czyli $\delta_k = \delta_m \cdot (\frac{\partial m}{\partial k})$.

Gdyby do jednego miejsca propagowane było wiele poprawek, to trzeba je wcześniej zsumować.



$\delta_c = \delta_{c1} + \delta_{c2}$

$\delta_a = \delta_c \cdot (\frac{\partial c}{\partial a})$,

$\delta_b = \delta_c \cdot (\frac{\partial c}{\partial b})$

...

W przypadku neuronu sigmoidalnego mającego numer *B* w warstwie *A* z należy wyliczyć poprawki w następującej kolejności:

1. Poprawki liczone dla sumatora poszczególnych neuronów:

   $\delta_{s,LA,NB} = \delta_{wy,LA,NB} \cdot \left(\frac{\partial wy_{LA,NB}}{\partial s_{LA,NB}} \right) = \ldots$, gdzie *LA* to numer warstwy, *NB* to numer neuronu w warstwie *LA*.

2. Poprawki liczone dla wag ukrytych wejść (przy założeniu domyślnym, że wartość wejść ukrytych to +1) :

   $\delta_{w,LA,NB,I0} = \delta_{s,LA,NB} \cdot \left(\frac{\partial s_{LA,NB}}{\partial w_{LA,NB,I0}} \right) = \delta_{s,LA,NB}$.

3. Poprawki liczone dla wag nieukrytych wejść:

   $\delta_{w,LA,NB,Ii} = \delta_{s,LA,NB} \cdot \left(\frac{\partial s_{LA,NB}}{\partial w_{LA,NB,Ii}} \right) = \delta_{s,LA,NB} \cdot we_{LA,NB,Ii}$, gdzie *Ii* to numer wejścia.

4. Poprawki liczone dla nieukrytych wejść neuronów:

   $\delta_{we,LA,NB,Ii} = \delta_{s,LA,NB} \cdot \left(\frac{\partial s_{LA,NB}}{\partial we_{LA,NB,Ii}} \right) = \delta_{s,LA,NB} \cdot w_{LA,NB,Ii}$. Tego działania nie ma sensu wykonywać dla pierwszej warstwy.

Po wyliczeniu poprawek dla każdego neuronu w warstwie *A,* jeśli nie jest ona ostatnią warstwą należy wyliczyć poprawki dla wyjść wszystkich neuronów warstwy poprzedniej.

- Przykład 1 (najczęściej spotykany; taki jak dla struktur widocznych w tej instrukcji): Jeśli wszystkie wyjścia poprzedniej warstwy są podłączone do każdego neuronu w warstwie aktualnej wtedy wzór ten można zapisać jako:

  $\delta_{wy,LA-1,Na} = \sum_{b=1...} \delta_{we,LA,Nb,Ia}$, gdzie *b* to indeksy kolejnych, wszystkich neuronów w warstwie *LA,* $\delta_{wy,LA-1,Na}$ to poprawka liczona dla wyjścia *a*-tego neuronu *(A-1)*-tej warstwy, $\delta_{we,LA,Nb,Ia}$ to poprawka liczona na a-tym wejściu *b*-tego neuronu *A*-tej warstwy.

- Przykład 2 (prostszy): Jeśli wyjście neuronu *Na* z warstwy *LA-1* jest podłączone z wejściem *Ib* neuronu *Nc* warstwy *LA* i wejściem *Id* neuronu *Ne* warstwy *LA,* to poprawka ta wynosi:
  $\delta_{wy,LA-1,Na} = \delta_{we,LA,Nb,Ic} + \delta_{we,LA,Nd,Ie}$.

Zadanie 1. XOR

Proszę stworzyć sieć składającą się z 2 wejść, 1 wyjścia oraz 2+1 neuronów. Sieć ta będzie uczona następującymi próbkami:

```
0 0 0
0 1 1
1 0 1
1 1 0
```

, gdzie pierwsze dwie kolumny to wejścia, a ostatnia to wyjście. Proszę dobrać takie parametry uczenia β, μ, liczba epok , aby po nauczeniu dla każdej z próbek uczących moduł błędu $|d - wy_{całejSieci}|$ był mniejszy niż 0,3.

Wskazówki

- Na początku proponuje się przetestować dla $\beta = 1$, $\mu = 0{,}1$, liczba epok $= 20000$, a wartości początkowe wag wylosować z przedziałów $[-1;+1]$.
- Czasami niestety, dla bardzo pechowo wylosowanych początkowych wartości wag, po przeprowadzaniu 20000 epok uczenia sieć może wciąż nie działać prawidłowo.
- Przy pierwszej implementacji zaleca się nie pomijać ostatniego kroku uczenia, czyli samokontroli.

Zadanie 2. (niekonieczne) XOR + NOR, 2-2-2-2

Proszę stworzyć sieć składającą się z 2 wejść, 2 wyjść i trzech warstw neuronów, po dwa neurony na każdą warstwę. Sieć ta będzie uczona następującymi próbkami:

Wejście 1.	Wejście 2.	Wyjście 1.	Wyjście 2.
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	0

Zbiór próbek xor_nor.txt:
0 0 0 1
0 1 1 0
1 0 1 0
1 1 0 0
Nazwa atrybutów xor_nor-names.txt:
we_1 b
we_2 b
wy_1 b
wy_2 b

Proszę dobrać takie parametry uczenia β, μ, liczba epok , aby po nauczeniu dla każdej z próbek uczących moduł błędu $|d_{wy1} - wy_{calejSieci1}|$ i $|d_{wy2} - wy_{calejSieci2}|$ był mniejszy niż 0,4.

Zadanie 3. (niekonieczne) Sumator, 3-3-2-2

Proszę stworzyć sieć składającą się z 3 wejść, 2 wyjść i trzech warstw neuronów: z trzema neuronami w warstwie pierwszej, i dwoma neuronami w warstwach drugiej i trzeciej. Sieć ta będzie uczona następującymi próbkami:

Wejście 1.	Wejście 2.	Wejście 3.	Wyjście 1.	Wyjście 2.
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1

0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Zbiór próbek sumatorek.txt:

0 0 0 0 0
0 1 0 1 0
1 0 0 1 0
1 1 0 0 1
0 0 1 1 0
0 1 1 0 1
1 0 1 0 1
1 1 1 1 1

Nazwa atrybutów sumatorek-names.txt:

we_1 b
we_2 b
we_3 b
wy_1 b
wy_2 b

Proszę dobrać takie parametry uczenia β, μ, liczba epok , aby po nauczeniu dla każdej z próbek uczących moduł błędu $\left|d_{wy1} - wy_{calejSieci1}\right|$ i $\left|d_{wy2} - wy_{calejSieci2}\right|$ był mniejszy niż 0,4.