Ćwiczenie nr 4

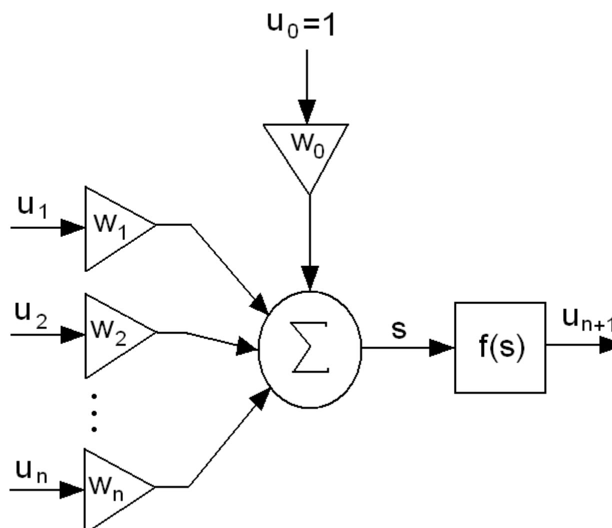
Sztuczne sieci neuronowe

Sztuczna komórka neuronowa, perceptron, funkcja aktywacji, algorytmy trenowania perceptronu: perceptronowy, kieszeniowy, kieszeniowy z zatrzaskiem

1. Wprowadzenie

Sztuczne sieci neuronowe stanowią intensywnie rozwijającą się dziedzinę wiedzy stosowaną w wielu obszarach nauki. Zajmują lekarzy i biologów, zainteresowanych modelowaniem biologicznych sieci neuronowych, oraz fizyków widzących analogię pomiędzy modelami sieci neuronowych, a nieliniowymi układami dynamicznymi. Matematyków interesują analogie pomiędzy opisami formalnymi sieci, a modelowaniem systemów złożonych. Inżynierowie elektronicy widzą sieci neuronowe jako układy przetwarzające sygnały. Są także zainteresowani wytwarzaniem inteligentnych maszyn wykorzystujących elektroniczne układy scalone. Psycholodzy patrzą na sztuczne sieci neuronowe jak na możliwe wzorce struktur przetwarzania informacji przez człowieka. Wreszcie informatycy zainteresowani są możliwościami otwieranymi przez równoległe struktury obliczeniowe w dziedzinach sztucznej inteligencji, teorii obliczeń, czy symulacji komputerowej.

Na rys. 1 przedstawiono model sztucznej komórki nerwowej.



Rys. 1 – Model sztucznego neuronu

Gdzie: *n* – liczba wejść w neuronie, $u_1, u_2, ..., u_n$ – sygnały wejściowe, $w_0, w_1, ..., w_n$ – wagi synaptyczne, u_{n+1} – wartość wyjściowa neuronu, u_0 – wartość progowa, *f* – funkcja aktywacji, *s* – wartość sumy ważonej.

Formuła opisująca działanie neuronu wyraża się zależnością:

$$u_{n+1} = f(s) \quad (1)$$

w której

$$s = \sum_{i=0}^{n} u_i \cdot w_i \quad (2)$$

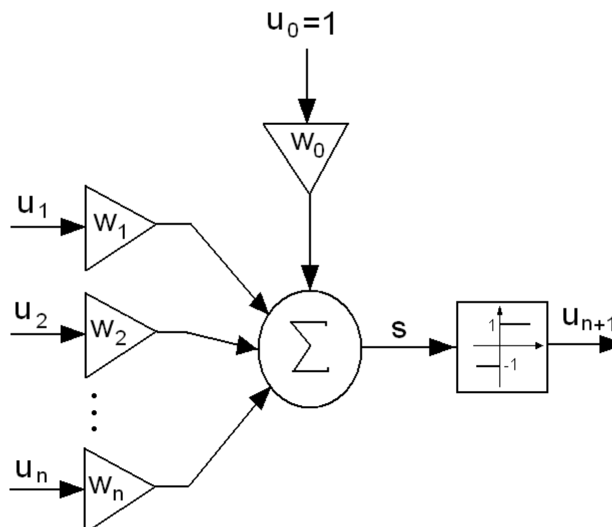
Funkcja aktywacji *f* może przybierać różną postać w zależności od konkretnego modelu neuronu.

Jak widać z powyższych wzorów, działanie neuronu jest bardzo proste. Najpierw sygnały wejściowe $u_0, u_1, ..., u_n$ zostają pomnożone przez odpowiadające im wagi $w_0, w_1, ..., w_n$.

Otrzymane w ten sposób wartości należy następnie zsumować. W wyniku powstaje sygnał *s* odzwierciedlający działanie części liniowej neuronu. Sygnał ten jest poddawany działaniu funkcji aktywacji, najczęściej nieliniowej. Zakładamy, że wartość sygnału u_0 jest równa 1, natomiast wagę w_0 nazywa się progiem (ang. bias). Gdzie zatem kryje się wiedza w tak opisanym neuronie ? Otóż wiedza zapisana jest właśnie w wagach. Największym zaś fenomenem jest to, iż w łatwy sposób (za pomocą algorytmów trenujących) można neurony uczyć, a więc odpowiednio dobierać wagi. Na rys. 1 przedstawiono ogólny schemat neuronu, jednakże w sieciach stosuje się różne jego modele. Należy jeszcze wspomnieć, iż podobnie jak w mózgu komórki nerwowe łączą się ze sobą, tak i w przypadku tworzenia modeli matematycznych sztuczne neurony przedstawione na rys. 1 łączy się ze sobą tworząc wielowarstwowe sieci neuronowe. Najprostszym modelem neuronu jest perceptron, który opisano w punkcie 2.

2. Perceptron

Na rys. 2 przedstawiono schemat perceptronu.



Rys. 2 – Schemat perceptronu

Działanie perceptronu można opisać zależnością:

$$u_{n+1} = f\left(\sum_{i=0}^{n} w_i \cdot u_i\right) \tag{3}$$

Funkcja *f* może być nieciągłą funkcją skokową – bipolarną (przyjmuje wartości –1 lub 1) lub unipolarną (przyjmuje wartości 0 lub 1). Do dalszych rozważań przyjmiemy, iż funkcja aktywacji jest bipolarna:

$$f(s) = \begin{cases} 1, & gdy \;\; s > 0 \\ -1, & gdy \;\; s < 0 \\ 0, & gdy \;\; s = 0 \end{cases} \tag{3}$$

Perceptron ze względu na swą funkcję aktywacji przyjmuje tylko dwie różne wartości wyjściowe, może więc klasyfikować sygnały podane na jego wejście w postaci wektorów $\boldsymbol{u} = [u_1, ..., u_n]^T$ do jednej z dwóch klas. Na przykład perceptron z jednym wejściem może oceniać, czy sygnał wejściowy jest dodatni, czy ujemny. W przypadku dwóch wejść u_1 i u_2 perceptron dzieli płaszczyznę na dwie części. Podział ten wyznacza prosta o równaniu:

$$w_0 \cdot u_0 + w_1 \cdot u_1 + w_2 \cdot u_2 = 0 \tag{4}$$

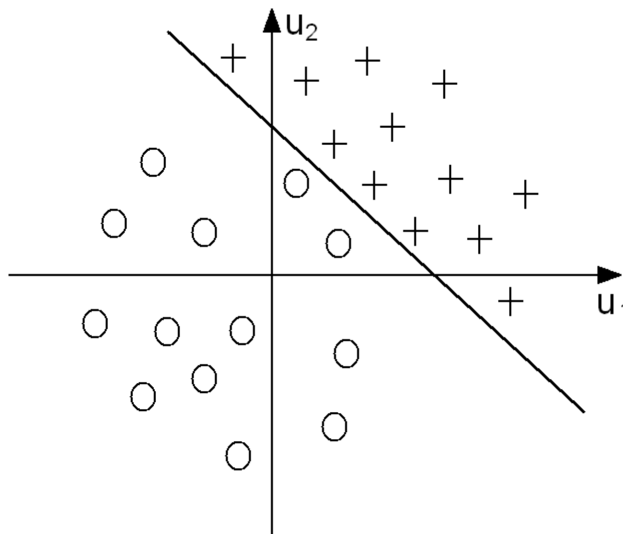
Zatem równanie można zapisać:

$$u_2 = -\frac{w_1}{w_2} \cdot u_1 - \frac{w_0 \cdot u_0}{w_2} \tag{5}$$

W ogólnym przypadku, gdy perceptron ma *n* wejść, wówczas dzieli *n* – wymiarową przestrzeń wektorów wejściowych ***u*** na dwie półprzestrzenie. Są one rozdzielone *n* – 1 - wymiarową hiperpłaszczyzną, nazywaną granicą decyzyjną, daną wzorem:

$$\sum_{i=0}^{n} w_i \cdot u_i = 0 \tag{6}$$

Na rys. 3 przedstawiono przykładową granicę decyzyjną dla *n* = 2.



Rys. 3 – Przykładowa granica decyzyjna dla n=2

3. Algorytmy uczące pojedynczy perceptron

Zgodnie z tym co zostało napisane wcześniej, perceptron można uczyć. W czasie tego procesu jego wartości wag są modyfikowane. Metoda uczenia perceptronu należy do grupy algorytmów zwanych uczeniem z nauczycielem lub uczeniem nadzorowanym. Uczenie tego typu polega, że na wejście perceptronu podaje się sygnały wejściowe, dla których znamy prawidłowe wartości sygnałów wyjściowych, zwanych sygnałami wzorcowymi. Zbiór takich próbek wejściowych wraz z odpowiadającymi im wartościami sygnałów wzorcowych nazywamy ciągiem uczącym. Podstawowym algorytmem uczenia perceptronu jest algorytm perceptronowy, który przedstawiono na rys. 4.

1. Przyjmij zerowy wektor wag [W] = [0];
2. Wybierz parę trenującą [Ek] i Ck; wybór ten może być kolejny ze wzrostem k od 1 do N, lub losowy, z uwzględnieniem wszystkich par trenujących
3. Jeżeli dla bieżącego wektora wag [W] następuje poprawa klasyfikacyjna, tj.
   s>0, dla Ck = 1 lub s<0 dla Ck=-1,
   wówczas nie rób nic (wektor wag [W] pozostaje bez zmian) tj. [W*]=[W];
   inaczej: zmodyfikuj wektor wag [W] przez dodanie lub odjęcie od niego wektora trenującego [Ek] w zależności od tego czy: Ck=1 czy Ck=-1, tj.:

   $$[W^*] = [W] + \frac{Ck - \mathrm{sgn}(s)}{2} \cdot [Ek],$$

   gdzie funkcja signum sgn(s) opisana jest zależnością:

$$\mathrm{sgn}(s)=\begin{cases}1, & dla\ \ s>0\\ -1, & dla\ \ s<0\end{cases}$$

4. Dla s=0, zmień wagi według zależności:

[W*]=[W]+Ck [Ek]

5. Idź do kroku 2.

Rys. 4 – Algorytm trenowania perceptronu jednokomórkowego

Algorytm z punktu 2 kończy pracę, gdy po podaniu kolejno wszystkich wektorów trenujących odpowiedź perceptronu jest prawidłowa. Poniżej przedstawiono przykładowe zastosowanie algorytmu z rys. 4 do wytrenowania 2 wejściowego perceptronu do spełnienia funkcji logicznej AND. Zestaw par trenujących jest następujący:

	E			C
	u_0	u_1	u_2	
E1	1	-1	-1	-1
E2	1	-1	1	-1
E3	1	1	-1	-1
E4	1	1	1	1

Natomiast kod algorytmu perceptronowego w Scilabie do wytrenowania pojedynczego perceptronu do funkcji logicznej AND jest następujący:

```
//---- perceptron trenowanie
//---- okreslenie par wektorow trenujacych
//---- dla funktora AND
A=ones(4,4);
A(1,2)=-1; A(1,3)=-1; A(1,4)=-1;
A(2,2)=-1; A(2,4)=-1;
A(3,3)=-1; A(3,4)=-1;
//---- wykreslenie obszaru klasyfikacji
for i=1:4
    if A(i,4)==1
        plot(A(i,2),A(i,3),'ko:');
    else
        plot(A(i,2),A(i,3),'r+:');
    end
end
mtlb_axis([-2 2 -2 2])
//---- ustalenie poczatkowych wartosci wag
W=[0 0 0];
//---- proces trenowania
disp(W);
Zmiana=1;
while (Zmiana==1)
    Zmiana=0;
    for i=1:4 //--- kolejno pobiera wektory trenujace
        S=A(i,1)*W(1)+A(i,2)*W(2)+A(i,3)*W(3);
        Sig=0;
        if S>0
            Sig=1;
        else
            Sig=-1;
        end
        if ((Sig>0) & (A(i,4)==1)) | ((Sig<0) & (A(i,4)==-1))
            W=W;
        else
            Zmiana=1;
```

```
         if (S~=0)
            for j=1:3
               W(j)=W(j)+0.5*(A(i,4)-Sig)*A(i,j);
            end
         end
      end
      if S==0
         Zmiana=1;
         for j=1:3
            W(j)=W(j)+(A(i,4)*A(i,j));
         end
      end
   disp(W);
   end
end
//---- wykreslenie otrzymanej linii podzialu
k=0;
for i=-2:0.01:2
   k=k+1;
   XX(k)=i;
   YY(k)=-((W(2)/W(3))*i)-(W(1)*1)/W(3);
end
plot(XX,YY);
mtlb_axis([-2 2 -2 2])
```

Rys. 5a – Przykładowa implementacja algorytmu perceptronowego w środowisku Scilab do wytrenowania funkcji logicznej AND

Natomiast ten sam kod w języku Python (Spyder) przedstawia się następująco:

```
#---- perceptron trenowanie
#---- okreslanie par wektorow trenujacych
#---- dla funktora AND
import numpy as np
import matplotlib.pyplot as plt
A=np.ones((4,4))
A[0,1]=-1; A[0,2]=-1; A[0,3]=-1;
A[1,1]=-1; A[1,3]=-1;
A[2,2]=-1; A[2,3]=-1;
#---- wykreslenie obszaru klasyfikacji
for i in range(0,4):
    if A[i,3]==1:
        plt.plot(A[i,1],A[i,2],'ko:')
    else:
        plt.plot(A[i,1],A[i,2],'r+:')
plt.axis([-2,2,-2,2])
#---- ustalenie poczatkowych wartosci wag
W=[0,0,0]
#---- proces trenowania
print(W)
Zmiana=1
while (Zmiana==1):
    Zmiana=0
    for i in range(0,4):
        S=A[i,0]*W[0]+A[i,1]*W[1]+A[i,2]*W[2]
        Sig=0
        if S>0:
            Sig=1
        else:
            Sig=-1
        if (Sig>0 and A[i,3]==1) or (Sig<0 and A[i,3]==-1):
            W=W
        else:
            Zmiana=1
            if S!=0:
```

```
                for j in range(0,3):
                    W[j]=W[j]+0.5*(A[i,3]-Sig)*A[i,j]
        if S==0:
            Zmiana=1
            for j in range(0,3):
                W[j]=W[j]+A[i,3]*A[i,j]
        print(W)
#---- wykreslenie otrzymanej linii podzialu
k=-2;
XX=np.zeros((401))
YY=np.zeros((401))
for i in range(0,401):
    XX[i]=k
    YY[i]=-((W[1]/W[2])*k)-(W[0]*1)/W[2];
    k=k+0.01;
plt.plot(XX,YY)
```

Rys. 5b – Przykładowa implementacja algorytmu perceptronowego w środowisku Spyder (Python) do wytrenowania funkcji logicznej AND

Problem trenowania perceptronu do funkcji logicznej AND jest liniowo separowalny tzn. można przy użyciu jednej linii decyzyjnej (w przypadku problemu 2 wejściowego) dokonać poprawnej klasyfikacji. Natomiast w przypadku problemów liniowo nieseparowalnych (np. XOR) korzystanie z algorytmu przedstawionego na rys. 4 doprowadzi do niemożności określenia odpowiedniego wektora wag, gdyż algorytm będzie zachowywał się cyklicznie.

Uniknąć tego można przez zastosowanie „algorytmu kieszeniowego". Algorytm ten zapamiętuje wybrany wektor wag umieszczając go w pamięci (kieszeni). Umieszczenie w kieszeni nowego wektora [W] następuje wówczas, gdy liczba iteracji, w których [W] klasyfikuje poprawnie jest większa od liczby iteracji dla wektora [Wk] znajdującego się w kieszeni.

Algorytm ten przedstawiono na rys. 6. Jego modyfikacja w stosunku do algorytmu z rys. 4 polega na rozbudowie kroku 3 i polega na zliczeniu liczby iteracji, dla których następuje poprawna klasyfikacji i ewentualnym umieszczeniu w pamięci nowego wektora wag. Również w kroku 2 przykłady trenujące są wybierane wyłącznie losowo.

1. Przyjmij zerowy wektor wag [W] = [0] i zerowy wektor wag w kieszeni [Wk] = 0
2. Wybierz losowo parę trenującą [Ek] i Ck
3. Jeżeli dla bieżącego wektora wag [W] następuje poprawa klasyfikacyjna, tj. s>0, dla Ck = 1 lub s<0 dla Ck=-1, wówczas:
   3a. Jeżeli bieżąca liczba iteracji, dla których kolejne klasyfikacje są poprawne przy danym wektorze [W] jest większa niż liczba iteracji poprawnych klasyfikacji dla wektora wag [Wk] w kieszeni, to zastąp wagi w kieszeni [Wk] przez [W] i zapamiętaj liczbę tych iteracji.
4. Gdy klasyfikacja nie jest poprawna wówczas zmodyfikuj wektor wag [W] poprzez dodanie lub odjęcie od niego wektora trenującego [Ek] w zależności od tego czy: Ck=1 lub Ck=-1, tj.:
$$[W^*] = [W] + \frac{Ck - \operatorname{sgn}(s)}{2} \cdot [Ek]$$
5. Dla s=0, zmień wagi według zależności:
$$[W^*]=[W]+Ck\,[Ek]$$
6. Idź do kroku 2

Rys. 6 – Algorytm „kieszeniowy" trenowania perceptronu jednokomórkowego

Ponieważ przy działaniu algorytmu kieszeniowego przykłady trenujące Ek są wybierane losowo to może się zdarzyć, szczególnie przy większej liczbie wejść, jak np. dla funkcji PAR-n (funkcja parzystości z n wejściami), że gorsze zestawy wag klasyfikujące poprawnie mniej przykładów niż jest to możliwe będą się powtarzały dłużej niż optymalne zestawy wag (klasyfikujące poprawnie najwięcej możliwych przykładów) i wejdą do kieszeni (zostaną zapamiętane).

Aby uniknąć niedogodności algorytmu kieszeniowego wprowadza się jego modyfikację w postaci algorytmu kieszeniowego z „zatrzaskiem” (ang. ratchet). Głównym celem zmodyfikowanego algorytmu jest badanie, czy „kandydat” do umieszczenia w kieszeni klasyfikuje poprawnie więcej przykładów trenujących Ek niż wektor wag aktualnie znajdujący się w kieszeni. Jeśli tak wówczas do zostaje on zapisany w „kieszeni”. Algorytm kieszeniowy z „zatrzaskiem” przedstawiono na rys. 7.

1. Przyjmij zerowy wektor wag [W] = [0] i zerowy wektor wag w kieszeni [Wk] = 0
2. Wybierz losowo parę trenującą [Ek] i Ck
3. Jeżeli dla bieżącego wektora wag [W] następuje poprawa klasyfikacyjna, tj.
   s>0, dla Ck = 1 lub s<0 dla Ck=-1,
   wówczas:
   3a. Jeżeli bieżąca liczba iteracji, dla których kolejne klasyfikacje są poprawne przy danym wektorze [W] jest większa niż liczba iteracji poprawnych klasyfikacji dla wektora wag [Wk] w kieszeni, to sprawdź czy wektor bieżący [W] poprawnie klasyfikuje więcej przykładów trenujących niż wektor [Wk] będący w kieszeni, jeśli tak, to zastąp wagi w kieszeni [Wk] przez [W] i zapamiętaj liczbę tych iteracji.
4. Gdy klasyfikacja nie jest poprawna wówczas zmodyfikuj wektor wag [W] poprzez dodanie lub odjęcie od niego wektora trenującego [Ek] w zależności od tego czy: Ck=1 lub Ck=-1, tj.:
$$[W^*] = [W] + \frac{Ck - \mathrm{sgn}(s)}{2} \cdot [Ek]$$
5. Dla s=0, zmień wagi według zależności:
   [W*]=[W]+Ck [Ek]
6. Idź do kroku 2

Rys. 7 – Algorytm „kieszeniowy z zatrzaskiem” trenowania perceptronu jednokomórkowego

4. Zadania do wykonania

a) przepisać i uruchomić program z rys. 5
b) na podstawie algorytmu z rys. 4 i programu z rys. 5 napisać program trenujący perceptron do problemu klasyfikacji funkcji logicznej OR, przyjąć następujące wektory trenujące:

	E			C
	u_0	u_1	u_2	
E1	1	-1	-1	-1
E2	1	-1	1	1
E3	1	1	-1	1
E4	1	1	1	1

W programie narysować przestrzeń klasyfikacyjną oraz wyznaczyć linię decyzyjną (podobnie jak w programie z rys. 5)

c) dopisać do programu z punktu 4b, możliwość sprawdzenia wytrenowania perceptronu przez użytkownika, tzn. po przeprowadzeniu trenowania i narysowania linii decyzyjnej program powinien poprosić użytkownika o podanie z klawiatury wartości u_1 oraz u_2 a następnie wygenerować odpowiedź perceptronu na podane przez użytkownika wartości wejściowe. Np. dla u_1=–1 i u_2=–1 odpowiedz perceptronu powinna wynosić –1. Sprawdzić poprawność generowanych wyników.
d) na podstawie algorytmu z rys. 4 i programu z rys. 5 napisać program trenujący perceptron do problemu klasyfikacji funkcji logicznej (3 wejściowej), przyjąć następujące wektory trenujące:

	E				C
	u_0	u_1	u_2	u_3	
E1	1	-1	-1	-1	-1
E2	1	-1	-1	1	-1
E3	1	-1	1	-1	-1
E4	1	-1	1	1	-1
E5	1	1	-1	-1	-1
E6	1	1	-1	1	1
E7	1	1	1	-1	-1
E8	1	1	1	1	-1

Po wytrenowaniu perceptronu sprawdzić poprawność klasyfikacji, umożliwiając użytkownikowi wprowadzanie danych wejściowych (u_1, u_2, u_3) z klawiatury oraz wyprowadzając na ekran wartość odpowiedzi generowanej przez perceptron.

e) zastosować algorytm z rys. 6, do wytrenowania perceptronu do problemu klasyfikacji funkcji logicznej o charakterze XOR, przyjąć następujące wektory trenujące (wykreślić przestrzeń klasyfikacyjną oraz linię decyzyjną będącą wynikiem wytrenowania perceptronu). Sprawdzić poprawność klasyfikacji.

	E			C
	u_0	u_1	u_2	
E1	1	-1	-1	-1
E2	1	-1	1	1
E3	1	1	-1	1
E4	1	1	1	-1
E5	1	0.9	0.9	-1
E6	1	0.9	0.8	-1
E7	1	0.8	0.9	-1
E8	1	0.95	0.95	-1
E9	1	0.95	0.9	-1
E10	1	-0.8	0.8	1
E11	1	-0.85	0.9	1
E12	1	0.8	-0.8	1

f) zastosować algorytm z rys. 7 do wytrenowania perceptronu do problemu klasyfikacji funkcji logicznej XOR z punktu 4e (wykreślić przestrzeń klasyfikacyjną oraz linię decyzyjną). Sprawdzić poprawność klasyfikacji.