Patryk Kurzeja

Inżynieria Obliczeniowa

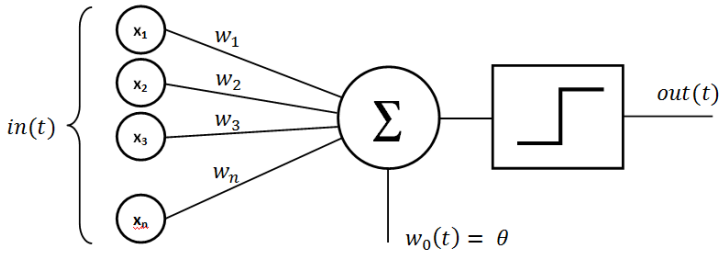
Wydział : WImiIP

Podstawy sztucznej inteligencji

Sprawozdanie : Budowa i działanie perceptronu [Scenariusz 1]

Perceptron – prosty element obliczeniowy, którego zadaniem jest sumowanie ważonych sygnałów wejściowych i porównywanie tej sumy z progiem aktywacji. W zależności od wyniku perceptron może być albo wzbudzony (wynik 1), albo nie (wynik 0). Użycie perceptronu jako klasyfikatora jest możliwe, jednak jego możliwości są ograniczone do zbiorów separowalnych liniowo. W praktyce możemy jednak spróbować wykorzystać perceptron również wtedy, gdy zbiory nie są separowalne liniowo, godząc się na pewną, możliwą liczbę błędnych rozpoznań.

Najprostszą siecią jednokierunkową jest perceptron prosty. Zbudowany jest jedynie z warstwy wejściowej i warstwy wyjściowej. Ponieważ nie istnieją połączenia pomiędzy elementami warstwy wyjściowej, każdy z tych elementów może być traktowany niezależnie jako osobna sieć o m+1 wejściach i jednym wyjściu.



Perceptron złożony z jednego neurona McCullocha-Pittsa

Elementem składowym perceptronu jest sztuczny neuron, którego model matematyczny może być opisany funkcją aktywacji unipolarną

$$y = \begin{cases} 1, & \varphi > 0 \\ 0, & \varphi \leq 0 \end{cases}$$

lub bipolarną

$$\operatorname{sgn}(\varphi) = \begin{cases} +1, & \varphi > 0 \\ -1, & \varphi \leq 0 \end{cases}$$

gdzie

$$\varphi = \sum_{i=1}^{m} w_i u_i - \theta$$

przy czym w_i ; oznacza wagę i-tego połączenia wstępującego do elementu; u_i - wartość *i-tego* wejścia; ϑ - wartość progową funkcji aktywacji.

Algorytm uczenia perceptronu – Metoda uczenia perceptronu należy do grupy algorytmów zwanych uczeniem z nauczycielem lub uczeniem nadzorowanym. Polega na automatycznym doborze wag. Uczenie tego typu polega, że na wyjściu 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. Przedstawia się on następująco:

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, tzn. s>0, dla Ck = 1 lub s<0 dla Ck=-1, wówczas nie rób nic (wektor wag [W] pozostaje bez zmian) czyli [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, tzn.:

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

   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:

$$[W^*] = [W] + Ck\,[Ek]$$

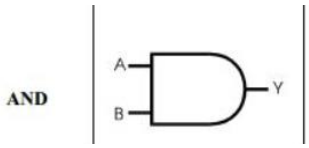
5. Idź do kroku 2.

Algorytm z punktu 2 kończy pracę jeżeli po podaniu kolejno wszystkich wektorów trenujących odpowiedź perceptronu jest prawidłowa.

Przeprowadzenie ćwiczenia :

- Wykorzystanie algorytmu do uczenia bramek logicznych.
- Wywołanie funkcji: NET = NEWP(PR, S, TF, LF)
  - PR - macierz o wymiarach Rx2, gdzie R jest liczbą wejść sieci (liczbą współrzędnych wektorów wejściowych); pierwsza kolumna macierzy R zawiera minimalne wartości kolejnych współrzędnych wektorów wejściowych, druga kolumna – maksymalne wartości tych współrzędnych
  - S - liczba neuronów sieci
  - TF - nazwa funkcji aktywacji neuronów (zmienna tekstowa); nazwa domyślna = 'hardlim'; dopuszczalne wartości parametru TF to: ‘hardlim’ i ‘hardlims’
  - LF - nazwa funkcji trenowania sieci perceptronowej (zmienna tekstowa); nazwa domyślna = 'learnp'; dopuszczalne wartości parametru LF to: ‘learnp’ i ‘learnpn’

1. AND



$A \cdot B = Y$

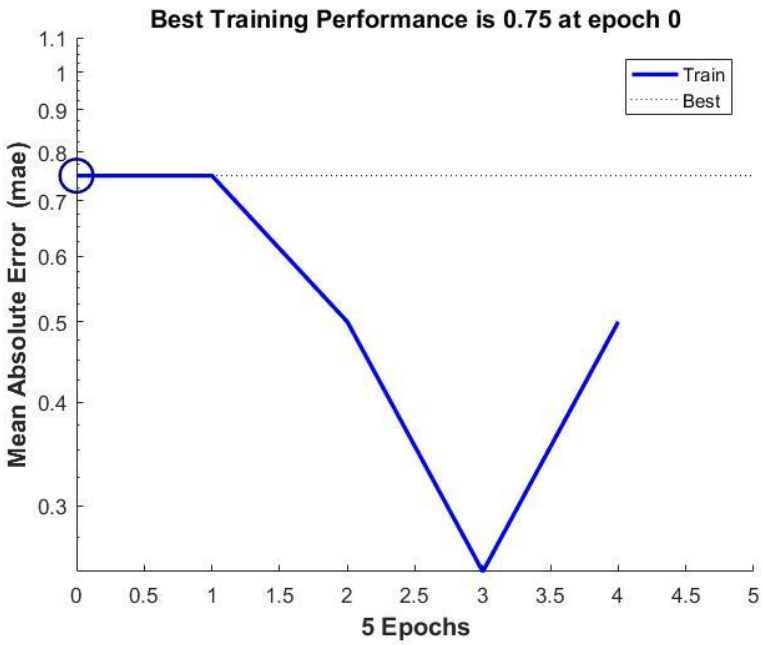
Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

```
net = newp([0 1; -2 2], 1);
A = [0 0 1 1];
B = [0 1 0 1];
P = [A;B];
T = [0 0 0 1];
net = init(net);
Wyj_przed_treningiem = sim(net,P)
net = train(net,P,T)
Wyj_po_treningu = sim(net,P)
```

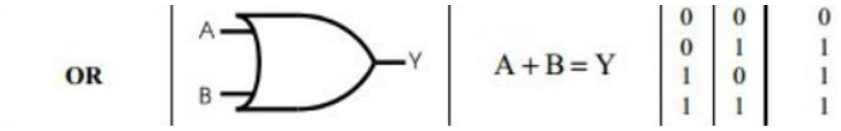
Dane wyjściowe :

Wyj_przed_treningiem = 1 1 1 1

Wyj_po_treningu = 0 0 0 1



2. OR

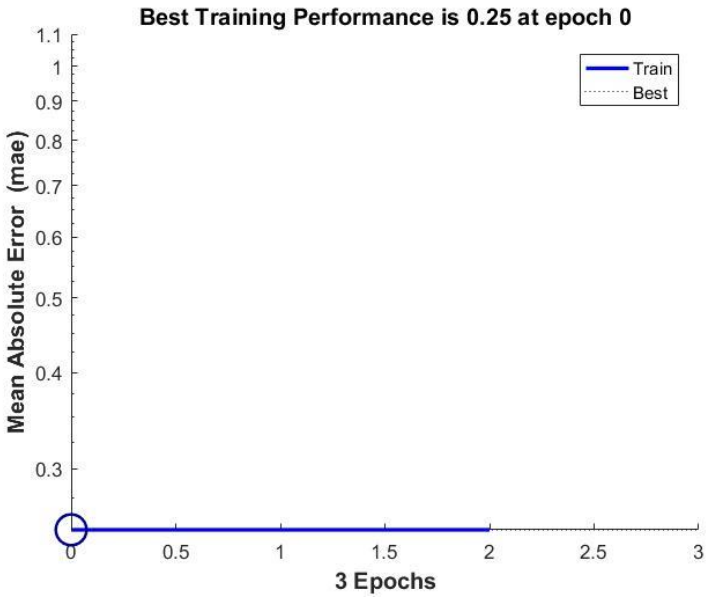



```
net = newp([0 1; -2 2], 1);
A = [0 0 1 1];
B = [0 1 0 1];
P = [A;B];
T = [0 1 1 1];
net = init(net);
Wyj_przed_treningiem = sim(net,P)
net = train(net,P,T)
Wyj_po_treningu = sim(net,P)
```

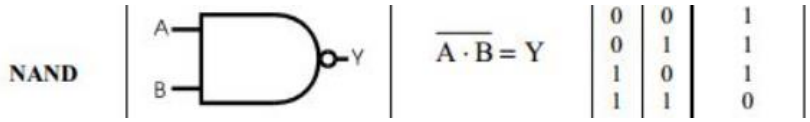
Dane wyjściowe :

Wyj_przed_treningiem = 1 1 1 1

Wyj_po_treningu = 0 1 1 1



3. NAND

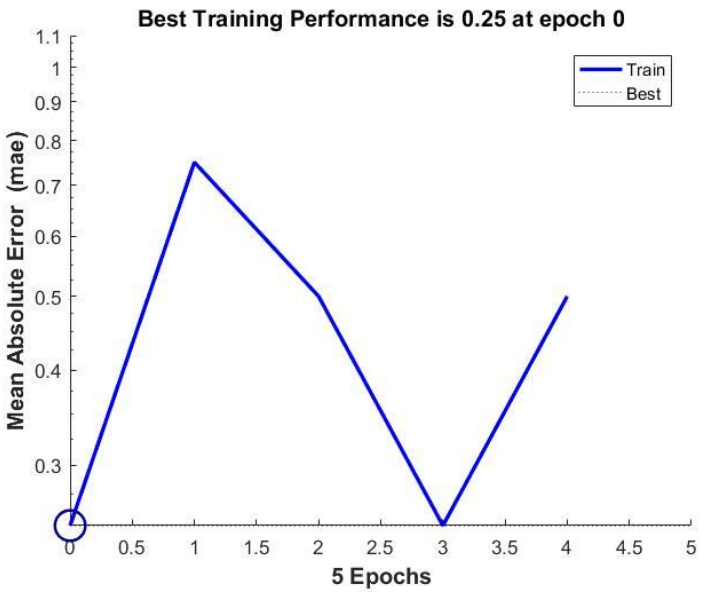



```
net = newp([0 1; -2 2], 1);
A = [0 0 1 1];
B = [0 1 0 1];
P = [A;B];
T = [1 1 1 0];
net = init(net);
Wyj_przed_treningiem = sim(net,P)
net = train(net,P,T)
Wyj_po_treningu = sim(net,P)
```

Dane wyjściowe :

Wyj_przed_treningiem = 1 1 1 1

Wyj_po_treningu = 1 1 1 0



4. NOR

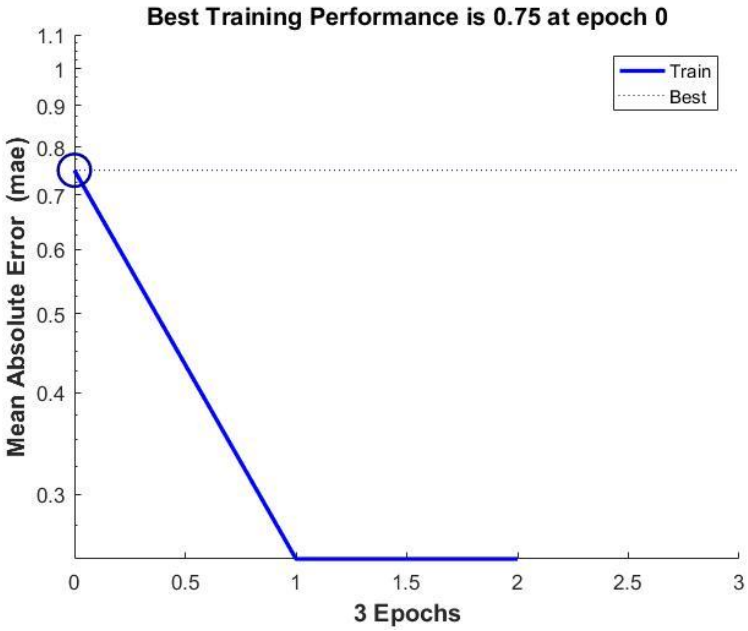
NOR		$\overline{A+B}=Y$	0 0 1 1	0 1 0 1	1 0 0 0

```
net = newp([0 1; -2 2], 1);
A = [0 0 1 1];
B = [0 1 0 1];
P = [A;B];
T = [1 0 0 0];
net = init(net);
Wyj_przed_treningiem = sim(net,P)
net = train(net,P,T)
Wyj_po_treningu = sim(net,P)
```

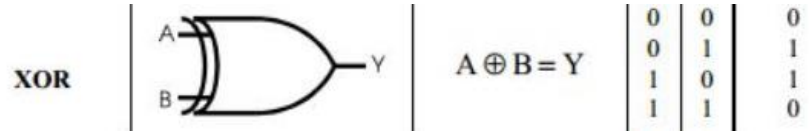
Dane wyjściowe :

Wyj_przed_treningiem = 1 1 1 1

Wyj_po_treningu = 1 0 0 0



5. XOR

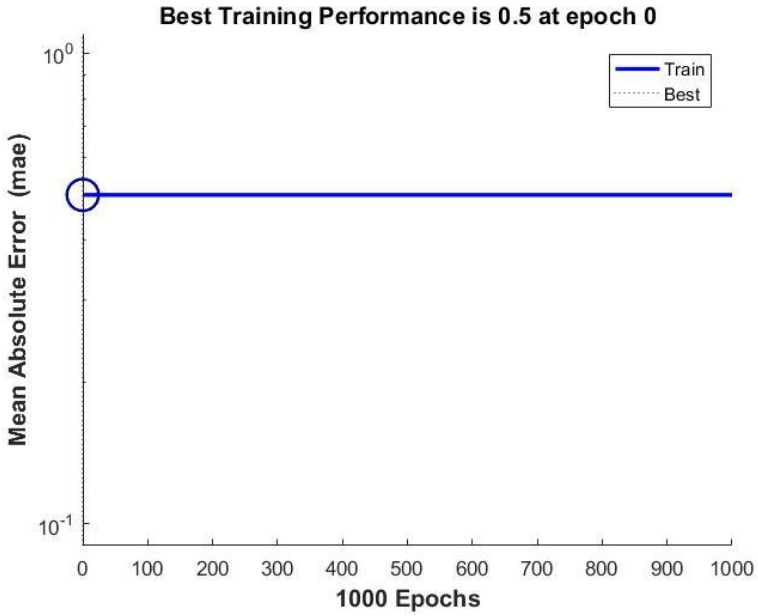



```
net = newp([0 1; -2 2], 1);
A = [0 0 1 1];
B = [0 1 0 1];
P = [A;B];
T = [0 1 1 0];
net = init(net);
Wyj_przed_treningiem = sim(net,P)
net = train(net,P,T)
Wyj_po_treningu = sim(net,P)
```

Dane wyjściowe :

Wyj_przed_treningiem = 1 1 1 1

Wyj_po_treningu = 1 1 0 0



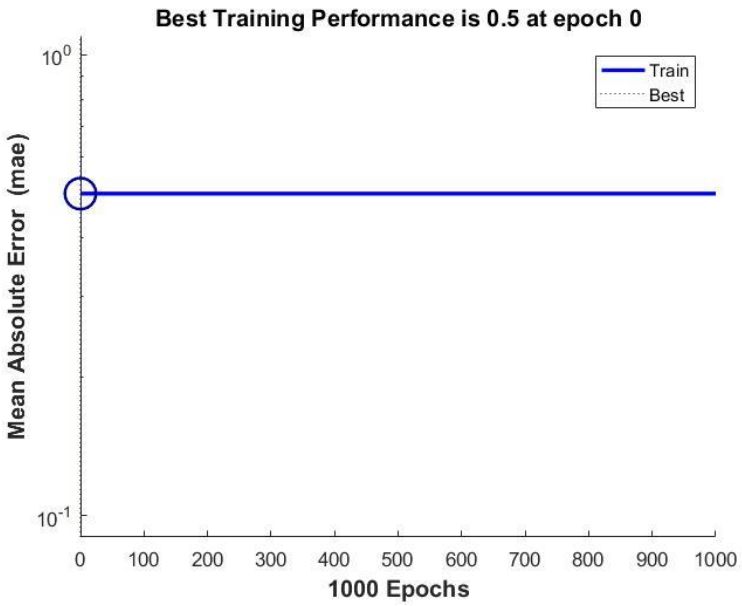
6. XNOR


```
net = newp([0 1; -2 2], 1);
A = [0 0 1 1];
B = [0 1 0 1];
P = [A;B];
T = [1 0 0 1];
net = init(net);
Wyj_przed_treningiem = sim(net,P)
net = train(net,P,T)
Wyj_po_treningu = sim(net,P)
```

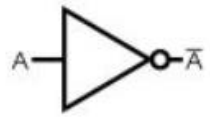
Dane wyjściowe :

Wyj_przed_treningiem = 1 1 1 1

Wyj_po_treningu = 0 0 1 1



7. NOT

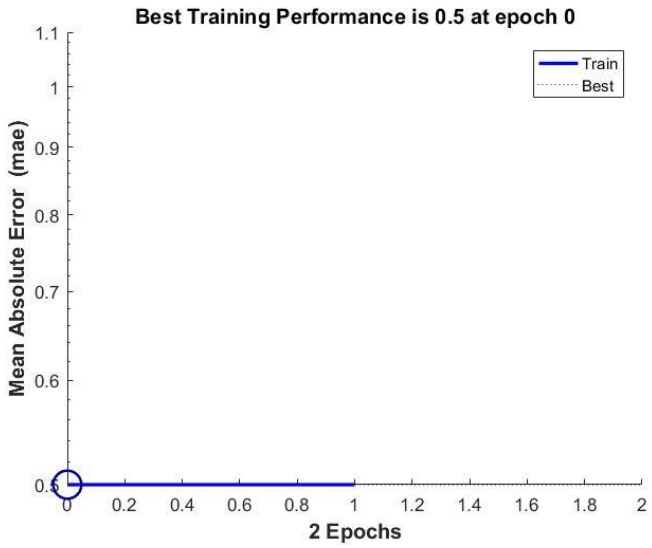
NOT (Inverter)		$A = \overline{A}$	0 1	1 0


```
net = newp([0 1; -2 2], 1);
A = [0 1];
B = [0 0 ];
P = [A;B];
T = [1 0 ];
net = init(net);
Wyj_przed_treningiem = sim(net,P)
net = train(net,P,T)
Wyj_po_treningu = sim(net,P)
```

Dane wyjściowe :

```
Wyj_przed_treningiem = 1  1

Wyj_po_treningu = 1 0
```



Podsumowanie :

Celem naszego ćwiczenia było poznanie budowy i działanie perceptronu poprzez implementację oraz uczenie perceptronu realizującego wybraną funkcję logiczną dwóch zmiennych. Przeprowadziłem ćwiczenie na bramkach : and, or, not, nand, nor, xor oraz xnor.

Jak widzimy na powyższych przykładach, aby zmienić bramkę logiczną wystarczy zmienić wartość A,B oraz T.

Podane niżej wywołanie funkcji tworzy sieć zawierającą pojedynczy neuron o dwóch wejściach. Zakres wartości pierwszego wejścia to [0, 1] drugiego [-2, 2].

Wywołanie funkcji zawiera tylko dwa argumenty – pozostałe dwa przyjmą wartości domyślne:
‘hardlim’ dla funkcji aktywacji i ‘learnp’ dla funkcji treningu sieci:

```
net = newp([0 1; -2 2], 1);
```

Obserwując otrzymane wykresy możemy stwierdzić, że proces nauki został wykonany efektywnie, gdyż wynik otrzymywaliśmy już po kilku iteracjach.

Niepowodzeniem skończyła się próba nauki na bramce XOR oraz XNOR. Nie jest to możliwe do rozwiązania dla sieci jednowarstwowej ponieważ pojedynczy perceptron nie potrafi odróżniać zbiorów nieseparowalnych liniowo, tzn. takich, że między punktami z odpowiedzią pozytywną i negatywną nie da się poprowadzić prostej rozgraniczającej.
Funkcje AND, NAND, OR, NOR, NOT dają zbiory separowalne liniowo, a więc możliwe do realizacji perceptronem.