

Temat ćwiczenia: Budowa i działanie sieci jednowarstwowej

Cel ćwiczenia:

Celem ćwiczenia jest poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

Zadania do wykonania:

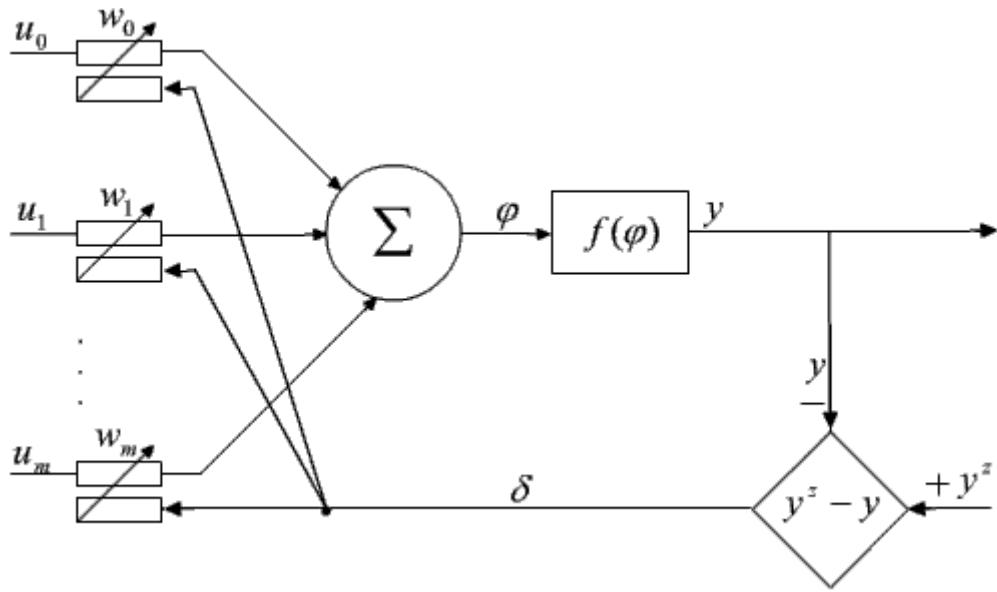
- a) Wygenerowanie danych uczących i testujących, zawierających 10 dużych i 10 małych liter dowolnie wybranego alfabetu w postaci dwuwymiarowej tablicy np. 5x7 pikseli dla jednej litery.
- b) Przygotowanie (implementacja lub wykorzystanie gotowych narzędzi) dwóch jednowarstwowych sieci - każda wg. innego algorytmu podanego na wykładzie.
- c) Uczenie sieci dla przy różnych współczynnikach uczenia.
- d) Testowanie sieci.

Ogólnie uczenie sieci, polega na odpowiednim zmienianiu wag $w(i)$ dla każdego wejścia neuronu. Wagi są zmieniane dopóki, odpowiedź neuronu mieści się w granicy błędu odchyłu od danych uczących.

Algorytm uczenia perceptronu:

1. Wybieramy losowo początkowe wartości wag.
2. Podajemy na wejście neuronu wektor uczący
3. Obliczenie wartości wyjściowej perceptronu y .
4. Porównanie wartości wyjściowej z wartością wzorcową (uczącą).
5. Modyfikacja wag:
 $w_i(\tau+1) = w_i(\tau) + (t - y)x_i(\tau)$, gdzie τ to wektor uczący.
6. Powrót do punktu 2.

Algorytm powtarzany jest dopóty, dopóki błąd na wyjściu nie będzie mniejszy od założonego, dla wszystkich wektorów ciągu uczącego.



Reguła delta

Reguła ta zakłada, że wraz z każdym wektorem wejściowym X do neuronu podawany jest sygnał Z (wymagana odpowiedź neuronu na sygnał X). W j -tym kroku procesu uczenia neuron odpowiada na sygnał $x(j)$ sygnałem

$$y(j) = W(j)^* X(j)$$

oraz jest określany błąd: $\delta(j) = z(j) - y(j)$.

Na podstawie tego sygnału błędu $\delta(j)$ oraz wektora wejściowego $X(j)$ możliwe jest takie skorygowanie wektora wag $W(j)$, by neuron generował sygnał bliższy zadanemu. Nowy wektor wag $W(j+1)$ (w $j+1$ -szym kroku) obliczany jest wg reguły DELTA:

$$W(j+1) = W(j) + \eta(j) \cdot \delta(j) \cdot X(j)$$

gdzie: η - współczynnik uczenia.

Reguła ta daje się łatwo stosować pod warunkiem wprowadzenia początkowego wektora wag $W(0)$, dobranego losowo. Bezwarunkowo trzeba unikać przyjmowania jednakowych wartości dla różnych składowych wektora W na początku procesu uczenia. Niedotrzymanie tego warunku prowadzi do braku postępu w początkowym etapie uczenia. Celem procesu uczenia jest uzyskanie zgodności odpowiedzi neuronu $y(j)$ z wymaganymi wartościami $z(j)$, co daje się sprowadzić do minimalizacji pewnej funkcji kryterialnej:

$$Q = 0.5 * \sum_{j=1}^N (z(j) - y(j))^2$$

Przyjmując do rozważań algorytm spadku gradientu należy zmieniać każdą wagę w_i osobno o pewną wielkość Δw_i . Wzór gradientowego uczenia dla kroku j ma postać:

$$w_i^{(j+1)} - w_i^{(j)} = \Delta w_i^{(j)} = -\eta \frac{\partial Q^{(j)}}{\partial w_i},$$

gdzie: i – indeks oznacza konkretną wagę neuronu.

1. Listing kodu:

b) drugi algorytm

Do poprzedniego listingu kodu dodałem wstawkę:

```
net2.adaptParam.passes = 3; % 3 pętle doboru wag i przesunięcia  
[net2,z,x,Pf,Af,TR] = adapt(net2, litery, wy); % wybór wag i przesunięcia  
net2.trainParam.epochs = 20; % Ustawienie ilości epok podczas treningu  
net2.trainParam.goal = 0.4; % kryterium stopu (sumę kwadratów błędów wyjść  
sieci)  
net2 = train(net2,litery,wy); % Trening sieci neuronowej.
```

2. Dane uczące i testujące.

Dane uczące zostały pobrane ze strony <http://www.ai.c-labtech.net/sn/litery.html>

Reszta danych została wygenerowana samodzielnie.

W moim przypadku:

- macierz A - dane testowe, jedna litera przedstawiona w postaci binarnej,
- macierz in - dane uczące, wszystkie litery a-j (duże i małe), przedstawione w postaci binarnej,
- macierz out - dane uczące, pożądane wyjścia, 1 - wyjście neuronu dla dużej litery, 0 - wyjście neuronu dla małej litery,

3. Wykorzystane algorytmy.

Skorzystałem z gotowego narzędzia newp, tworzącego nowy perceptron.

Implementacja wygląda następująco:

```
net=newp(LW, LN)
```

LW- liczba wejść i wartości jakie może przyjąć każdy neuron,

LN- liczba neuronów w sieci,

```
net=ini(net);
```

```
net.IW{NoN}=[W];
```

NoN- numer nerona,

W- wagi dla każdego wejścia,

```
Y=sim(net, P) %symulujemy sieć
```

P- macierz danych wejściowych,

```
Y=train(net, P, T)
```

P- macierz danych wejściowych,

T- target, pożądany cel,

```
Y=sim(net,P)
```

P- macierz danych wejściowych, ponowna symulacja, tym razem nauczonej już sieci.

Y- zawiera odpowiedź sieci neuronowej na podane zapytanie, wprowadzone dane.

4. Uczenie sieci przez zmienianie wag wejściowych

Wagi dobieram w sposób losowy i obserwuję jak zachowuje się sieć.

- $\text{net2.IW}\{1\}=[22 \ 23 \ 42 \ 51 \ 73 \ 48 \ 59 \ 64 \ 55 \ 33 \ 14 \ 27 \ 21 \ 72 \ 22 \ 35 \ 74 \ 25 \ 86 \ 75 \ 43 \ 74 \ 32 \ 62 \ 42 \ 71 \ 23 \ 14 \ 65 \ 86 \ 75 \ 43 \ 46 \ 24 \ 17];$

Sieci zajęło 24 epoki na naukę danych.

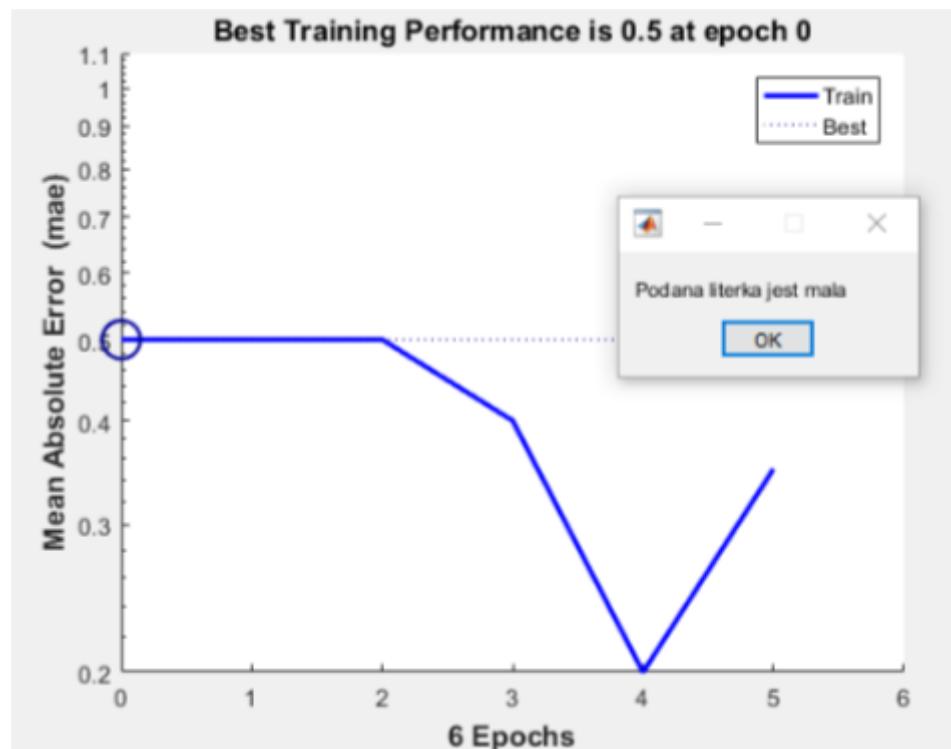


- $\text{net.IW}\{1\}=[2 \ 3 \ 2 \ 1 \ 3 \ 4 \ 5 \ 4 \ 5 \ 3 \ 4 \ 2 \ 1 \ 2 \ 2 \ 5 \ 4 \ 2 \ 6 \ 5 \ 3 \ 4 \ 3 \ 2 \ 2 \ 1 \ 3 \ 1 \ 5 \ 6 \ 5 \ 3 \ 4 \ 2 \ 1];$

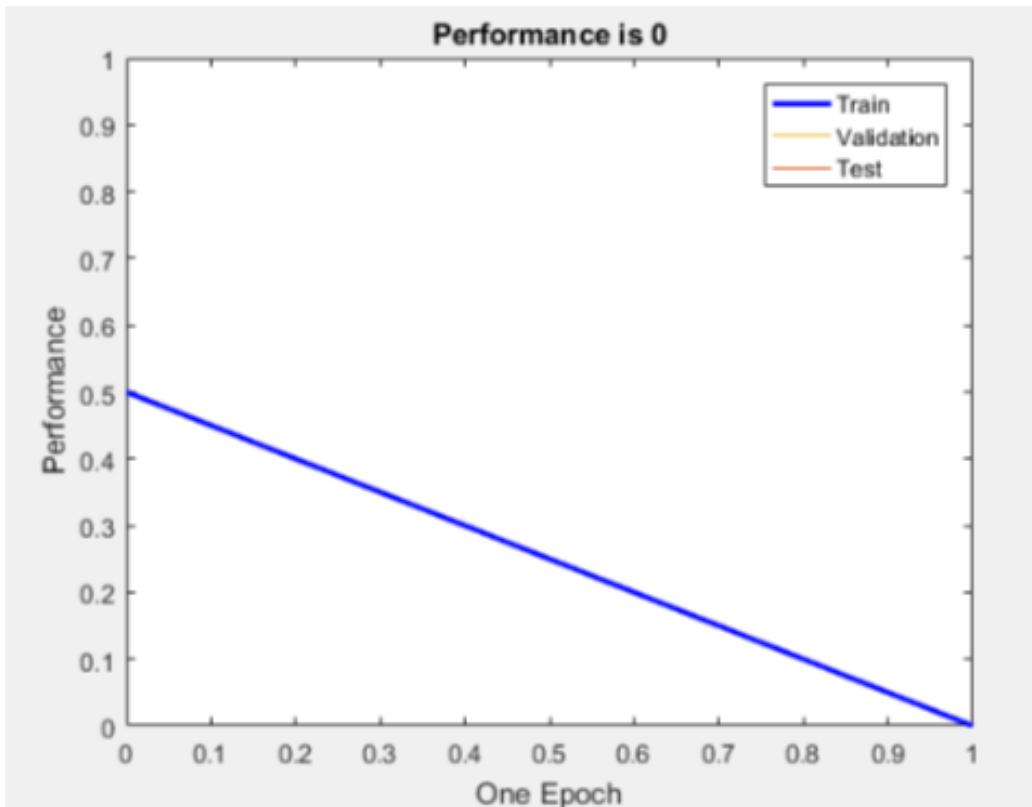
Sieć potrzebowała tylko 4 epoki na naukę przy takim zestawieniu wag.



- bez wag, domyślne ustawienie wag,



- `adapt()` - automatyczne wagи



5. Spostrzeżenia i uwagi

Ustawienie wag na jak najwyższe wydłuża czas uczenia oraz możemy zobaczyć jak odmienny jest wykres przebiegu uczenia od innych. Im mniejsze wagи ustaliliśmy tym proces przebiegał szybciej i lepiej wyglądał. Wniosek z tego taki, że im mniejsze wagи ustalimy tym proces uczenia będzie przebiegał szybciej.

Dla funkcji adapt, proces nauki przebiegał niezwykle szybko. Zakończył się w jednej eoce. Adapt w zadanej liczbie wykoną pętli poprzez parametr `adaptParam.passes`, sam dobiera jakie powinny być wagи na każdym węźle wejściowym neuronu. My nie wiemy jakie są te wagи. Funkcja ustala je samoczynnie i jak widać dopiera je w najlepszy możliwy sposób.

Najlepszym możliwym sposobem na poprawę uczenia jest zastosowanie funkcji adapt. Sieć neuronowa uczy się wtedy najlepiej i najszybciej, co w kontekście użycia sieci w praktyce ma kluczowe znaczenie.

Implementacja algorytmu użyta przeze mnie jest poprawna, gdyż jak możemy zauważać w przypadku podania danej litery to zmiennej i testowaniu sieci, wyskakuje okienko z wiadomością o wielkości litery. Sytuacja ta działa dla każdej z

liter A-J i a-j. Można to łatwo rozwinać o kolejne litery po prostu dodając ich wzór w postaci binarnej do macierzy in, czyli wejściowej macierzy neuronu(danych uczących).

Do algorytmu rozpoznawania wielkości litery wystarczy użyć algorytmu perceptronu. Wtedy w wejściach neuronu podaje się tylko 0 i 1. Graficzne przedstawienie litery i dalsza jego binaryzacja idealnie wpasuje się w rozpoznawanie wielkości liter przez perceptron.