Patryk Kurzeja

Inżynieria Obliczeniowa

Nr albumu : 286112

Podstawy sztucznej inteligencji

**Sprawozdanie Uczenie sieci regułą Hebba [ Scenariusz 4 ] :**

# 1. Cel ćwiczenia :

Celem ćwiczenia jest poznanie działania reguły Hebba dla sieci jednowarstwowej na przykładzie grupowania liter alfabetu.

# 2. Pojęcia i algorytmy :

- **Sieci neuronowe** - są sztucznymi strukturami, których budowa i działanie zostały zaprojektowane w sposób modelujący działanie naturalnego układu nerwowego, w szczególności mózgu. Cieszą się bardzo dużym zainteresowaniem. Jednym z głównych czynników mających na to wpływ jest możliwość stosowania ich w bardzo wielu dziedzinach życia do rozwiązywania problemów, gdzie użycie innych metod jest trudne lub niemożliwe. Znakomicie sprawdzają się w problemach klasyfikacji, predykcji, związanych ze sterowaniem, analizą danych.

- **Funkcja newff** - Funkcja tworzy wielowarstwową sieć neuronową; każda warstwa składa się z zadanej liczby neuronów o nieliniowych funkcjach aktywacji (jakkolwiek funkcje aktywacji w poszczególnych warstwach mogą mieć również postać liniową).

Wywołanie funkcji: **NET = NEWFF(PR, [S1 S2...SNL], ...**

**{TF1 TF2...TFNL}, BTF, BLF, PF)**

WEJŚCIE:

**PR** - macierz o wymiarach Rx2, gdzie R jest liczbą wejść sieci (współrzędnych wektorów wejściowych); pierwsza kolumna zawiera minimalne wartości kolejnych współrzędnych wektorów wejściowych, druga kolumna - maksymalne wartości tych współrzędnych

**Si** - liczba neuronów w i-tej warstwie sieci; liczba warstw wynosi N1

**TFi** - nazwa funkcji aktywacji neuronów w i-tej warstwie sieci (zmienna tekstowa); domyślna = 'tansig' (tangens hiperboliczny); dopuszczalne wartości parametru TF to: ‘tansig’ i ‘logsig’ i ‘purelin’

**BTF** - nazwa funkcji, wykorzystywanej do treningu sieci (zmienna tekstowa); domyślnie BTF = ‘trainlm’ (metoda Levenberga-Marquardta)

**BLF** - nazwa funkcji, wykorzystywanej do wyznaczania korekcji wag sieci podczas treningu (zmienna tekstowa); domyślnie BLF = ‘learngd’; dopuszczalne wartości parametru BLF to: ‘learngd’ (gradient prosty) i ‘learngdm’ (gradient prosty z momentum)

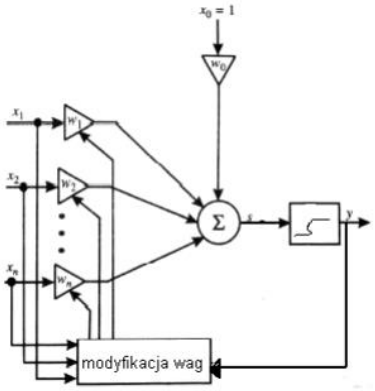
**PF** - funkcja wyznaczająca wartość wskaźnika jakości treningu sieci jednokierunkowej (zmienna tekstowa); domyślnie PF = ‘mse’ (błąd średniokwadratowy); parametr ten może oznaczać dowolną różniczkowalną funkcję błędu, np. ‘msereg’ (suma błędu średniokwadratowego i kwadratów wag sieci – metoda regularyzacji wag) lub ‘sse’ (suma kwadratów błędów)

WYJŚCIE:

**NET** - struktura (obiekt) zawierająca opis architektury, metod treningu, wartości liczbowe wag oraz inne parametry wielowarstwowej sieci jednokierunkowej.

- **Reguła Hebba** - Polega ona na tym, że sieci pokazuje się kolejne przykłady sygnałów wejściowych, nie podając żadnych informacji o tym, co z tymi sygnałami należy zrobić. Sieć obserwuje otoczenie i odbiera różne sygnały, nikt nie określa jednak, jakie znaczenie mają pokazujące się obiekty i jakie są pomiędzy nimi zależności. Sieć na podstawie obserwacji występujących sygnałów stopniowo sama odkrywa, jakie jest ich znaczenie i również sama ustala zachodzące między sygnałami zależności.

  Hebb zauważył podczas badań działania komórek nerwowych, iż połączenie pomiędzy dwiema komórkami jest wzmacniane, jeżeli w tym samym czasie obie komórki są aktywne.




Zaproponował on algorytm, zgodnie z którym modyfikację wag przeprowadza się następująco:

$$w_i(t+1) = w_i(t) + \eta y x_i$$

Oznaczenia:

- $i$-numer wagi neuronu,
- $t$-numer iteracji w epoce,
- $y$-sygnał wyjściowy neuronu,
- $x$-wartość wejściowa neuronu,
- $\eta$ - współczynnik uczenia (0,1).

Po podaniu do sieci neuronowej każdego kolejnego zestawu sygnałów wejściowych tworzy się w tej sieci pewien rozkład sygnałów wyjściowych - niektóre neurony sieci są pobudzone bardzo silnie, inne słabiej, a jeszcze inne mają sygnały wyjściowe wręcz ujemne. Interpretacja tych zachowań może być taka, że niektóre neurony „rozpoznają" podawane sygnały jako „własne" (czyli takie, które są skłonne akceptować), inne traktują je „obojętnie", zaś jeszcze u innych neuronów wzbudzają one wręcz „awersję". Po ustaleniu się sygnałów wyjściowych wszystkich neuronów w całej sieci - wszystkie wagi wszystkich neuronów są zmieniane, przy czym wielkość odpowiedniej zmiany wyznaczana jest na podstawie iloczynu sygnału wejściowego, wchodzącego na dane wejście (to którego wagę zmieniamy) i sygnału wyjściowego produkowanego przez neuron, w którym modyfikujemy wagi. Łatwo zauważyć, że jest to właśnie realizacja postulatu Hebba - w efekcie opisanego wyżej algorytmu połączenia między źródłami silnych sygnałów i neuronami które na nie silnie reagują są wzmacniane.

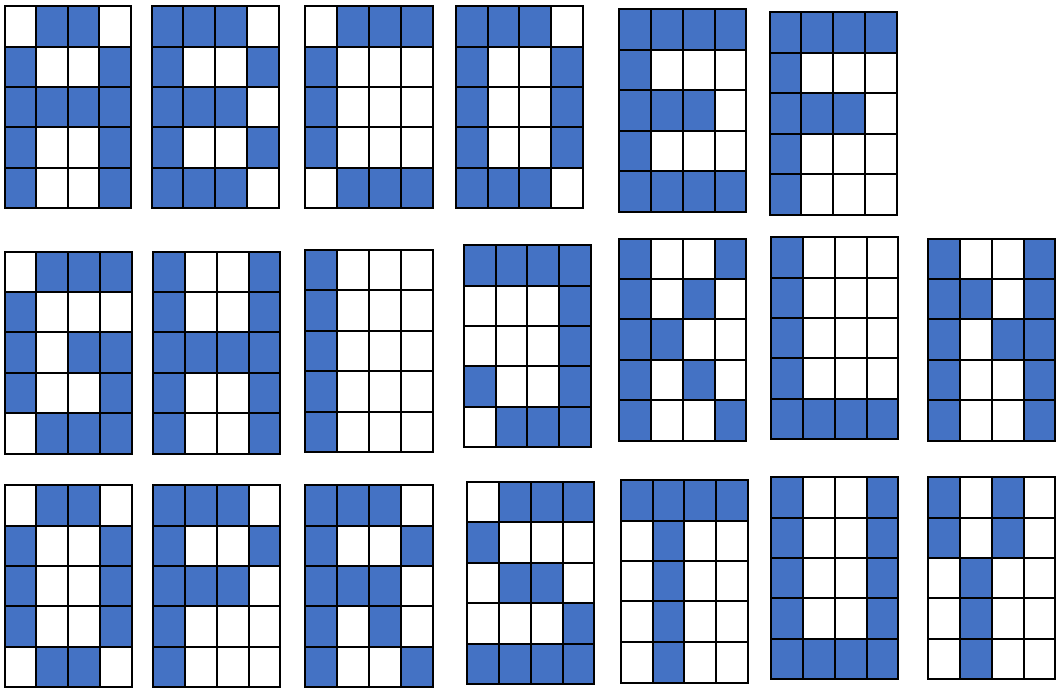
Proces samouczenia ma niestety wady. W porównaniu z procesem uczenia z nauczycielem samouczenie jest zwykle znacznie powolniejsze. Co więcej bez nauczyciela nie można z góry określić, który neuron wyspecjalizuje się w rozpoznawania której klasy sygnałów. Stanowi to pewną trudność przy wykorzystywaniu i interpretacji wyników pracy sieci. Co więcej - nie można określić, czy sieć uczona w ten sposób nauczy się wszystkich prezentowanych jej wzorców. Dlatego sieć przeznaczona do samouczenia musi być większa niż sieć wykonująca to samo zadanie, ale trenowana w sposób klasyczny, z udziałem nauczyciela. - Szacunkowo sieć powinna mieć co najmniej trzykrotnie więcej elementów warstwy wyjściowej niż wynosi oczekiwana liczba różnych wzorów, które sieć ma rozpoznawać.

Bardzo istotną kwestią jest wybór początkowych wartości wag neuronów sieci przeznaczonej do samouczenia. Wartości te mają bardzo silny wpływ na ostateczne zachowanie sieci, ponieważ proces uczenia jedynie pogłębia i doskonali pewne tendencje istniejące w sieci od samego początku

## 3. Wykonanie :

Wygenerowałem dane uczące, czyli 20 dużych liter alfabetu
A, B, C, D, E, F, G, H, I, J, K, L, N, O, P, R, S, T, U, Y.

Litery umieściłem w tablicach o rozmiarze 4x5 a następnie tablice przekształciłem w ciąg zer i jedynek, gdzie 1 oznacza pole czarne a 0 – pole białe ( puste ).



**Procedura przekształcania liter do postaci binarnej :**

- Wybieramy literę : np. A
- Tabelę 4x5 wypełniam 0 i 1
- A = [0; 1; 1; 0; 1; 0; 0; 1; 1; 1; 1; 1; 1; 0; 0; 1; 1; 0; 0; 1];
- Każdą literę zapisuję w postaci 20 znakowego ciągu

**Dane wejściowe :**

```
                %A B C D E F G H I J K L N O P R S T U Y
Input = [0 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1;
         1 1 1 1 1 1 1 0 0 1 0 0 0 1 1 1 1 1 0 0;
         1 1 1 1 1 1 1 0 0 1 0 0 0 1 1 1 1 1 0 1;
         0 0 1 0 1 1 1 1 0 1 1 0 1 0 0 0 1 0 1 0;
         1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1;
         0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0;
         0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1;
         1 1 0 1 0 0 0 0 1 0 1 0 0 1 1 1 1 0 0 1 0;
         1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0;
         1 1 0 0 1 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1;
         1 1 0 0 1 1 1 1 0 0 0 0 1 0 1 1 1 0 0 0;
         1 0 0 1 0 0 1 1 0 1 0 0 1 1 0 0 0 0 1 0;
         1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0;
         0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1;
         0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0;
         1 1 0 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 0;
         1 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 0 0 0;
         0 1 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 1 1 1;
         0 1 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0;
         1 0 1 0 1 0 1 1 0 1 1 1 1 0 0 1 0 0 0 0;
         ];
```

**Dane wyjściowe :**

```
      %A B C D E F G H I J K L N O P R S T U Y
Output =[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
         0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
         0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
         0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
         0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
         0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
         0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0;
         0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0;
         0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0;
         0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0;
         0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1;
         ];
```

Dane wejściowe wprowadziłem do sieci i przeprowadziłem jej uczenie funkcją **newff**.

Dane wyjściowe zawierają wartość 1 w kolumnie odpowiadającej danej literze.

Sieć uczyła się grupowania liter z zestawu 20 podanych w danych uczących.

Program określa wartości wag poszczególnych liter. Litery o takich samych lub zbliżonych wartościach wag uznajemy za takie same lub podobne gdy w rzeczywistości są one różne.

**Listing kodu programu :**

```
%% Sprawozdanie 4
close all; clear all; clc;

% Min i max wejsc sieci :
PR = [0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;
   0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;];

% Ilosc neuronow w kazdej z warstw :
S = 20;

% Funkcja NEWFF :
net = newff(PR,S,{'tansig'},'trainlm','learnh');


     %A B C D E F G H I J K L N O P R S T U Y
Input = [0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 1 1 1;
     1 1 1 1 1 1 1 0 0 1 0 0 0 1 1 1 1 1 0 0;
     1 1 1 1 1 1 1 0 0 1 0 0 0 1 1 1 1 1 0 1;
     0 0 1 0 1 1 1 1 0 1 1 0 1 0 0 0 1 0 1 0;
     1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1;
     0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1;
     1 1 0 1 0 0 0 0 1 0 1 0 0 1 1 1 1 0 0 1 0;
     1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0;
     1 1 0 0 1 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1;
     1 1 0 0 1 1 1 1 0 0 0 0 1 0 1 1 1 0 0 0;
     1 0 0 1 0 0 1 1 0 1 0 0 1 1 0 0 0 0 1 0;
     1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1;
     0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0;
```

```
    1 1 0 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 0;
    1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 0 0 0;
    0 1 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 1 1 1;
    0 1 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0;
    1 0 1 0 1 0 1 1 0 1 1 1 1 0 0 1 0 0 0 0;
    ];

    %A B C D E F G H I J K L N O P R S T U Y
Output =[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0;
    0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0;
    0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0;
    0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
    0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1;
    ];

% Parametry uczenia sie :
lp.dr = 0.5; % Współczynnik zapominania
lp.lr = 0.9; % Współczynnik uczenia się
parameters = learnh([0],Input,[0],[0],Output,[0],[0],[0],[0],[0],lp,[0]);

net.trainParam.epochs = 1000; % Max epochs
net.trainParam.goal = 0.001; % Cel wydajności
net.trainParam.lr = 0.5; % Wskaźnik nauki

% Uczenie :
net = train(net, Input, parameters);

% Dane testowe :
A = [0; 1; 1; 0; 1; 0; 0; 1; 1; 1; 1; 1; 1; 0; 0; 1; 1; 0; 0; 1];
B = [1; 1; 1; 0; 1; 0; 0; 1; 1; 1; 1; 0; 1; 0; 0; 1; 1; 1; 1; 0];
C = [0; 1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1];
D = [1; 1; 1; 0; 1; 0; 0; 1; 1; 0; 0; 1; 1; 0; 0; 1; 1; 1; 1; 0];
E = [1; 1; 1; 1; 1; 0; 0; 0; 1; 1; 1; 0; 1; 0; 0; 0; 1; 1; 1; 1];
F = [1; 1; 1; 1; 1; 0; 0; 0; 1; 1; 1; 0; 1; 0; 0; 0; 1; 0; 0; 0];
G = [0; 1; 1; 1; 1; 0; 0; 0; 1; 0; 1; 1; 1; 0; 0; 1; 0; 1; 1; 1];
H = [1; 0; 0; 1; 1; 0; 0; 1; 1; 1; 1; 1; 1; 0; 0; 1; 1; 0; 0; 1];
I = [1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0];
J = [1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 0; 1; 1; 0; 0; 1; 0; 1; 1; 1];
L = [1; 0; 0; 1; 1; 0; 1; 0; 1; 1; 0; 0; 1; 0; 1; 0; 1; 0; 0; 1];
L = [1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 1; 1; 1];
N = [1; 0; 0; 1; 1; 1; 0; 1; 1; 0; 1; 1; 1; 0; 0; 1; 1; 0; 0; 1];
O = [0; 1; 1; 0; 1; 0; 0; 1; 1; 0; 0; 1; 1; 0; 0; 1; 0; 1; 1; 0];
P = [1; 1; 1; 0; 1; 0; 0; 1; 1; 1; 1; 0; 1; 0; 0; 0; 1; 0; 0; 0];
R = [1; 1; 1; 0; 1; 0; 0; 1; 1; 1; 1; 0; 1; 0; 1; 0; 1; 0; 0; 1];
S = [0; 1; 1; 1; 1; 0; 0; 0; 0; 1; 1; 0; 0; 0; 0; 1; 1; 1; 1; 0];
T = [1; 1; 1; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0];
U = [1; 0; 0; 1; 1; 0; 0; 1; 1; 0; 0; 1; 1; 0; 0; 1; 0; 1; 1; 0];
Y = [1; 0; 1; 0; 1; 0; 1; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0];
```

```
% Symulacje :
effect_1 = parameters;
effect_2 = sim(net, A);

disp('Regula Hebba:')
disp('A = '),disp(sum(effect_1(1,':')));
disp('B = '),disp(sum(effect_1(2,':')));
disp('C = '),disp(sum(effect_1(3,':')));
disp('D = '),disp(sum(effect_1(4,':')));
disp('E = '),disp(sum(effect_1(5,':')));
disp('F = '),disp(sum(effect_1(6,':')));
disp('G = '),disp(sum(effect_1(7,':')));
disp('H = '),disp(sum(effect_1(8,':')));
disp('I = '),disp(sum(effect_1(9,':')));
disp('J = '),disp(sum(effect_1(10,':')));
disp('K = '),disp(sum(effect_1(11,':')));
disp('L = '),disp(sum(effect_1(12,':')));
disp('N = '),disp(sum(effect_1(13,':')));
disp('O = '),disp(sum(effect_1(14,':')));
disp('P = '),disp(sum(effect_1(15,':')));
disp('R = '),disp(sum(effect_1(16,':')));
disp('S = '),disp(sum(effect_1(17,':')));
disp('T = '),disp(sum(effect_1(18,':')));
disp('U = '),disp(sum(effect_1(19,':')));
disp('Y = '),disp(sum(effect_1(20,':')));
effect_1=effect_2;

% Wyznaczone wartosci wyjsciowe dla wszystkich liter :
disp('Wartosci wyjsciowe algorytmu dla wszystkich liter:')
disp('A = '),disp(effect_1(1));
disp('B = '),disp(effect_1(2));
disp('C = '),disp(effect_1(3));
disp('D = '),disp(effect_1(4));
disp('E = '),disp(effect_1(5));
disp('F = '),disp(effect_1(6));
disp('G = '),disp(effect_1(7));
disp('H = '),disp(effect_1(8));
disp('I = '),disp(effect_1(9));
disp('J = '),disp(effect_1(10));
disp('K = '),disp(effect_1(11));
disp('L = '),disp(effect_1(12));
disp('N = '),disp(effect_1(13));
disp('O = '),disp(effect_1(14));
disp('P = '),disp(effect_1(15));
disp('R = '),disp(effect_1(16));
disp('S = '),disp(effect_1(17));
disp('T = '),disp(effect_1(18));
disp('U = '),disp(effect_1(19));
disp('Y = '),disp(effect_1(20));
```

**Opis zmiennych oraz funkcji :**

**PR** - zmienna przechowująca minimalną i maksymalną wartość dla każdego z 20 wejść
**S** - zmienna przechowująca ilość wyjść z sieci
**net = newff(PR,S,{'tansig'},'trainlm','learnh')** - jednowarstwowa sieć neutronowa dla wprowadzonego z wykorzystaniem metody Hebba ('learnh')
**parameters = learnh([0],Input,[0],[0],Output,[0],[0],[0],[0],[0],lp,[0]);** - dostosowanie parametrów dla metody Hebba i przypisanie jej jednokrotnego wykonania do parameters
**Input** - dane wejściowe - 20 dużych liter zapisanych w postaci 0 i 1
**Output** - dane oczekiwane z wartościami 1 w odpowiedniej dla litery kolumnie
**net.trainParam.*** - ustawienie parametrów treningu
**lp.dr, lp.lr** – Współczynnik uczenia oraz współczynnik zapominania
**net = train(net, Input, parameters)** - proces uczenia sieci
**effect_2** = sim(net, A) - symulacja sieci
**disp('A='),disp(sum(effect_1(1,':'))) {...}** - wypisanie efektu działania reguły Hebba
**disp('A='),disp(effect_1(1)) {…}** - wypisanie efektów działania sieci wykorzystującej regułę Hebba

# 4. Wyniki działania :

**Wartości wag dla poszczególnych liter w zależności od współczynnika uczenia po jednokrotnym wywołaniu funkcji metody Hebba :**

| Współczynnik uczenia | 0.01 | 0.1 | 0.5 | 0.9 |
|---|---|---|---|---|
| A | 0.1200 | 1.2000 | 6 | 10.8000 |
| B | 0.1300 | 1.3000 | 6.5000 | 11.7000 |
| C | 0.1000 | 1.0000 | 5 | 9.0000 |
| D | 0.1200 | 1.2000 | 6 | 10.8000 |
| E | 0.1300 | 1.3000 | 6.5000 | 11.7000 |
| F | 0.1000 | 1.0000 | 5 | 9.0000 |
| G | 0.1200 | 1.2000 | 6 | 10.8000 |
| H | 0.1200 | 1.2000 | 6 | 10.8000 |
| I | 0.0500 | 0.5000 | 2.5000 | 4.5000 |
| J | 0.0800 | 0.8000 | 4 | 7.2000 |
| K | 0.1000 | 1.0000 | 5 | 9.0000 |
| L | 0.0800 | 0.8000 | 4 | 7.2000 |
| N | 0.1200 | 1.2000 | 6 | 10.8000 |
| O | 0.1000 | 1.0000 | 5 | 9.0000 |
| P | 0.1000 | 1.0000 | 5 | 9.0000 |
| R | 0.1000 | 1.2000 | 6 | 10.8000 |
| S | -0.0025 | 1.0000 | 5 | 9.0000 |
| T | 0.0100 | 0.7000 | 3.5000 | 6.3000 |
| U | 0.0075 | 1.0000 | 5 | 9.0000 |
| Y | 0.0100 | 0.7000 | 3.5000 | 6.3000 |

**Wyniki działania programu w zależności od współczynnika czyli różnych wag początkowych przydzielonych przez metodę Hebba :**

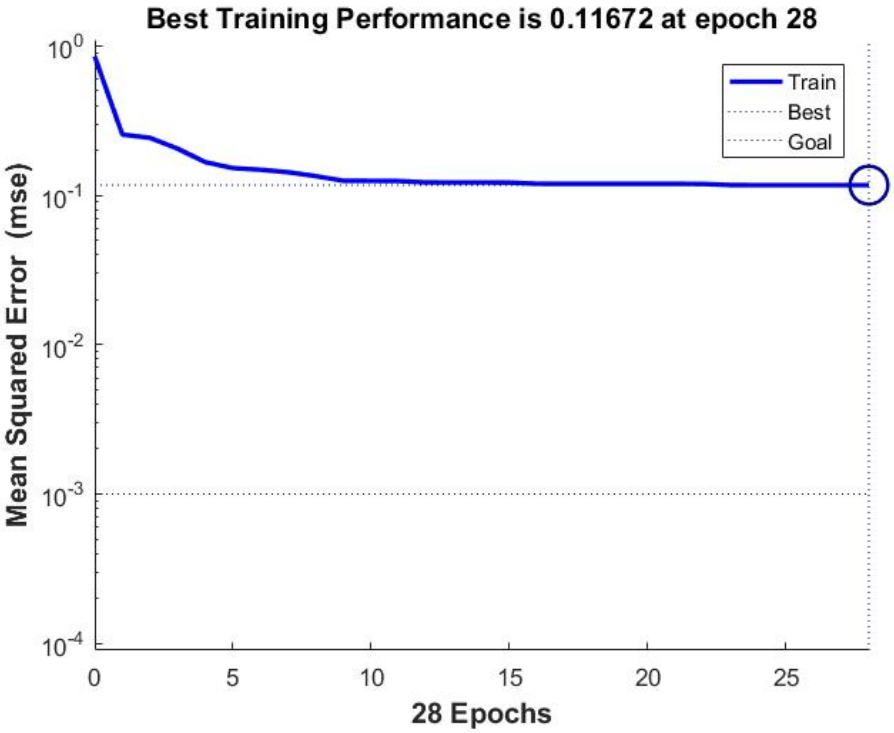
| Współczynnik uczenia | 0.01 | 0.1 | 0.5 | 0.9 |
|---|---|---|---|---|
| A | 0.0025 | 0.0250 | 0.1246 | 0.2191 |
| B | 0.0075 | 0.0751 | 0.4002 | 0.8942 |
| C | 9.9476e-14 | -0.0250 | -0.1246 | -0.2191 |
| D | 0.0075 | 0.0751 | 0.4002 | 0.8942 |
| E | 0.0050 | 0.0500 | 0.2500 | 0.4500 |
| F | 0.0075 | 0.0749 | 0.3591 | 0.5837 |
| G | -0.0025 | -0.0250 | -0.1246 | -0.2191 |
| H | 0.0100 | 0.1000 | 0.5000 | 0.9000 |
| I | 0.0075 | 0.0749 | 0.3591 | 0.5837 |
| J | 0.0100 | 0.1000 | 0.5000 | 0.9000 |
| K | 0.0075 | -1 | 0.5000 | 0.5837 |
| L | 0.0050 | 0.0500 | 0.2500 | 0.4500 |
| N | 0.0100 | 0.1000 | 0.5000 | 0.9000 |
| O | 0.0100 | 1 | -3.5527e-15 | 2.2204e-16 |
| P | 1 | 1 | 0.5000 | 0.9000 |
| R | 0.0100 | 0.1000 | 0.5000 | 0.9000 |
| S | -0.0025 | 5.3291e-15 | -0.1246 | -0.2191 |
| T | 0.0100 | 5.3291e-15 | 0.5000 | 0.9000 |
| U | 0.0100 | 5.3291e-15 | 0.4002 | 0.8942 |
| Y | 0.0100 | 0.1000 | 1.0000 | 0.9000 |

**Przykładowe pogrupowane litery dla współczynnika uczenia = 0.9 :**

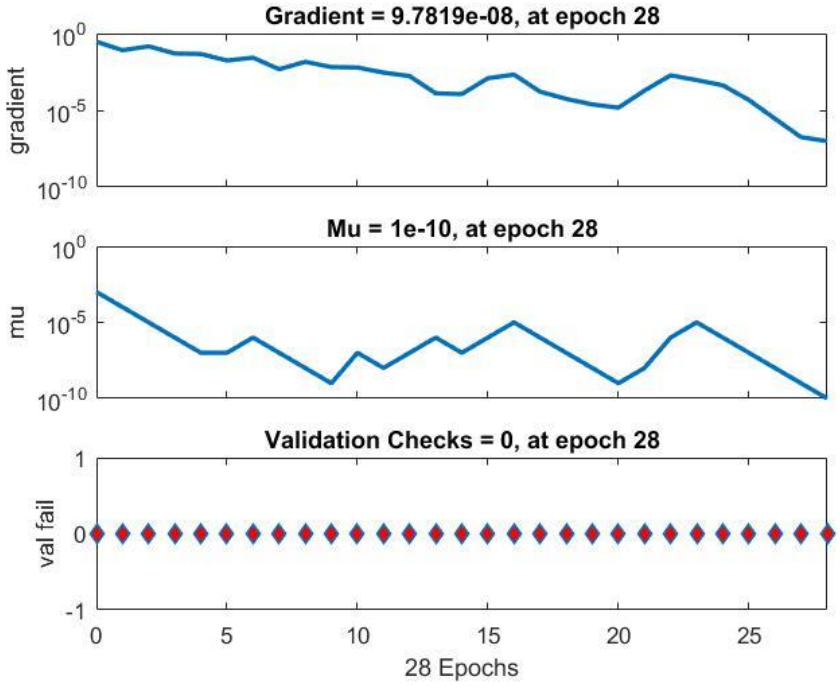
{B,D,U}, {E,L}, {C,G,S}, {H,J,N,P,R,T,Y}, {F,I,K}

Pogrupowane litery posiadają ten sam kolor co pomaga w ich identyfikacji.

## Wykresy wydajności :




Wydajność uczenia sieci – zauważamy bardzo szybki spadek wydajności procesu nauki. Po przekroczeniu 8 iteracji wydajność pozostaje bez zmian.



## 4. Wnioski :

Analizując otrzymane wyniki zauważamy, że do skutecznego procesu nauki wystarczy około 30 iteracji gdy korzystamy z reguły Hebbsa. Z wykresu wydajności zauważamy, że proces nauki przestaje być efektywny już po około 8 iteracjach ( epokach ).
Widzimy bardzo duży wpływ współczynnika uczenia na proces nauki.
Dla współczynnika uczenia równego 0.01 zostają pogrupowane litery, które nawet nie są do siebie podobne. Sytuacja wygląda lepiej podczas gdy współczynnik wynosi 0.1. Wtedy pogrupowane zostają litery takie jak : {C,G} czy {H,K}. Możemy stwierdzić że przy dużym wyidealizowaniu można by uznać je za podobne. Dla współczynnika równego 0.5 podobne litery to np. {P,R}, które już są do siebie rzeczywiście podobne. Jednak do ich grupy zostały przyporządkowane też : {H,J,K,T}, które już nie są za bardzo do nich podobne.
W przypadku zastosowania współczynnika na poziomie 0.9 zauważamy że proces rozpoznawania jest coraz dokładniejszy. Litery pogrupowane ze sobą w tym przypadku to np. : {B,D} czy {P,R} oraz {T,Y}. Te litery rzeczywiście są do siebie podobne i ich pogrupowanie razem ma sens.
Możemy dostrzec jak duży wpływ na wyniki zastosowanej metody mają wagi poszczególnych połączeń, które wynikają ze zastosowanego współczynnika uczenia. Czasami zdarzają się przypadki, gdy niektóre z liter są pogrupowane w ten sam sposób ale nie są one na tyle częste, by można było uznać że wagi nie wpływają na efekt działania sieci. Znaczna większość liter została pogrupowana w zupełnie inny sposób ze względu na odmienne wartości w zależności od zastosowanego współczynnika uczenia.
Metoda Hebba opiera się na uczeniu bez nauczyciela więc sieć jest zmuszona do tego aby samodzielnie decydować o tym jak zadziała dla różnych podanych danych wejściowych.
W tej metodzie wagi rosną w nieskończoność ponieważ nie jest to metoda zbieżna więc sama nie jest w stanie zakończyć nauki. Dość wysoki współczynnik uczenia powodował szybszy wzrost wag.
Reguła Hebba radzi sobie bardzo dobrze podczas grupowania liter za pomocą sieci neuronowych bez użycia nauczyciela.