|  |  |  |
| --- | --- | --- |
| Grupa ćwicz. **2** | Data wykonania 21.10.2017 | Nr. Scenariusza  **2** |
| **Temat ćwiczenia:** Budowa i działanie sieci jednowarstwowej | | |
| Imię i nazwisko  **Kamil Szczurkowski** | | Ocena i Uwagi |

**Cel ćwiczenia:**

Celem ćwiczenia jest poznanie budowy i działania jednowarstwowych sieci neuronowych

oraz uczenie rozpoznawania wielkości liter.

**Perceptron** – najprostsza sieć neuronowa, składająca się z jednego bądź wielu niezależnych neuronów McCullocha-Pittsa, implementująca algorytm uczenia nadzorowanego klasyfikatorów binarnych. Perceptron jest funkcją, która potrafi określić przynależność parametrów wejściowych do jednej z dwóch klas. Może być wykorzystywany tylko do klasyfikowania zbiorów liniowo separowalnych.

**Działanie perceptronu** polega na klasyfikowaniu danych pojawiających się na wejściu i ustawianiu stosownie do tego wartości wyjścia. Przed używaniem perceptron należy wytrenować, podając mu przykładowe dane na wejście i modyfikując w odpowiedni sposób wagi wejść i połączeń między warstwami neuronów, tak aby wartość na wyjściu przybierała pożądane wartości. Perceptrony mogą klasyfikować dane na zbiory, które są liniowo separowalne. Własność ta uniemożliwia na przykład wytrenowanie złożonego z jednego neuronu perceptronu, który wykonywałby logiczną operację XOR na wartościach wejść.



Rysunek 1- Perceptron złożony z jednego neurona McCullocha-Pittsa

**Algorytm uczenia perceptronu:**

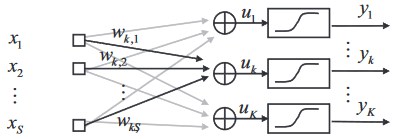
1. Wybieramy losowo początkowe wartości wag.
2. Podajemy na wejście neuronu wektor uczący   
   **x=x(**τ**)=[**x0(τ), x1(τ),..., xn(τ)]T, τ=1,2,....
3. Obliczenie wartości wyjściowej perceptronu y.
4. Porównanie wartości wyjściowej *y(*τ*)* z wartością wzorcową (uczącą) *t=t*(**x**(τ)).
5. Modyfikacja wag:  
    *wi(*τ*+1) = wi(*τ*) + (t – y)xi(*τ*)*
6. Powrót do punktu 2.

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

**(Warunek:** dane uczące reprezentują klasy liniowo separowalne)

**Neuron** (wg. McCullocha-Pittsa) - to podstawowy blok sztucznych sieci neuronowych. Jest on uproszczonym matematycznym modelem biologicznego neuronu. Neuron posiada wiele wejść i jedno bądź kilka wyjść. Każdemu z wejść przyporządkowana jest liczba rzeczywista - tak zwana waga wejścia.

**Sieć neuronowa** (*sztuczna sieć neuronowa*) – ogólna nazwa struktur matematycznych i ich programowych lub sprzętowych modeli, realizujących obliczenia lub przetwarzanie sygnałów poprzez rzędy elementów, zwanych sztucznymi neuronami, wykonujących pewną podstawową operację na swoim wejściu.



Rysunek 2- Sieć jednowarstwowa

**Sieci jednokierunkowe** - Ogólnie ujmując są to takie sieci, w których nie występują żadne sprzężenia zwrotne. W sieciach jednokierunkowych sygnały są przesyłane od warstwy wejściowej poprzez warstwy ukryte (jeśli występują) do warstwy wyjściowej. Sposób działania tego rodzaju sieci określa jednocześnie ich nazwę



Rysunek 3-Sieci jednokierunkowe uproszczony schemat

**Algorytm uczenia ciągłej jednowarstwowej sieci:**

Dane jest p par uczących:   
 *{(y1,d1),(y2,d2),...,(yp,dp)}*

gdzie *yi* ma rozmiar *J ×1, di* ma rozmiar *K ×1*, a sygnał progowy przyjmuje wartość *yiJ = -1, i = 1, 2,..., p*. Niech parametr *l* oznacza numer kroku cyklu uczenia.

1. Wybór *h > 0, Emax > 0*.
2. Wybór początkowych wartości elementów macierzy wag *W* jako niewielkich liczb losowych. Macierz *W* ma wymiar K ×J.
3. Ustawienie wartości początkowej licznika kroków oraz wyzerowanie wartości błędu: *l = 1, E = 0*.
4. Podanie obrazu na wejście i obliczenie sygnału wyjściowego: *y = yl*, *d=dl, zk = j(wkT y), k=1,2, ..., K* gdzie *wkT* jest *k*-tym wierszem macierzy *W*.
5. Dla bipolarnej (tanh) funkcji aktywacji uaktualnienie wag wg zależności:   
   

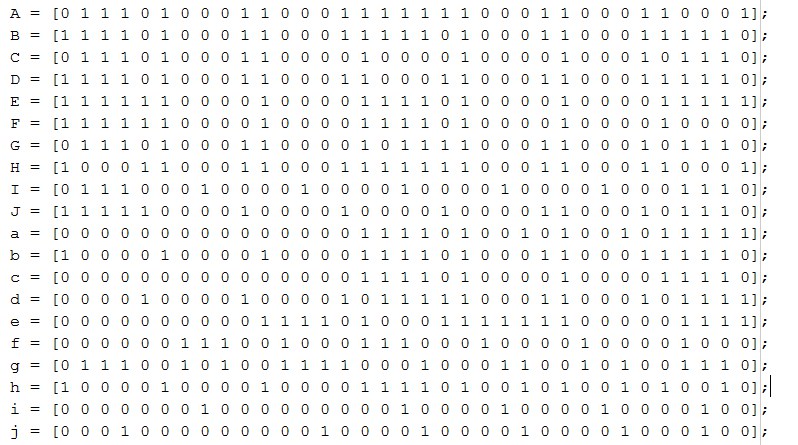


1. Obliczenie błędu łącznego:  
    
2. Jeżeli *l < p*, to *l=l+1* i przejście do kroku 4.
3. Cykl uczenia został zakończony. Jeżeli *E < Emax*, to zakończenie uczenia. W przeciwnym razie rozpoczęcie nowego cyklu uczenia począwszy od kroku 3.

**a) Wygenerowanie danych uczących i testujących, zawierających 10 dużych i 10**

**małych liter dowolnie wybranego alfabetu w postaci dwuwymiarowej tablicy np. 4x5**

**pikseli dla jednej litery.**



Rysunek 4- Tabela danych uczących się



Rysunek 5- Zdefiniowany wektor wyjść sieci.

**b) Przygotowanie (implementacja lub wykorzystanie gotowych narzędzi) dwóch**

**jednowarstwowych sieci - każda wg. innego algorytmu podanego na wykładzie.**

Wykorzystałem w tym celu Matlab’a i skorzystałem z takich funkcji:

**newp** - Tworzenie jednowarstwowej sieci złożonej z „twardych” perceptronów.

NEWP Funkcja tworzy jednowarstwoą sieć neuronową, złożoną z zadanej liczby

neuronów o funkcjach aktywacji „twardego” perceptronu (ang. *hardlimit*

*perceptron*).  
WYJŚCIE:*NET* - struktura (obiekt) zawierająca opis architektury, metod treningu, wartości

liczbowe wag i progów oraz inne parametry sieci perceptronowej.

**init** - Inicjalizacja struktury sieci neuronowej.

INIT Funkcja inicjalizuje wartości wag i progów neuronów sieci, wykorzystując

funkcję inicjalizującą, zdefiniowaną w polu NET.initFcn i wartości parametrów,

podane w polu NET.initParam wejściowej struktury *NET*. Zazwyczaj pole

NET.initFcn ma wartość 'initlay', co powoduje uŜycie do inicjalizacji kaŜdej

warstwy funkcji zdefiniowanej w polu NET.layers{i}.initFcn (indeks *i* oznacza

numer warstwy). Dla sieci jednokierunkowych (utworzonych np. za pomocą

funkcji ‘newff’) domyślna wartość pola NET.layers{i}.initFcn = 'initnw'; tak więc

inicjalizacja wag jest dokonywana za pomocą algorytmu Nguyena-Widrowa  
WYJŚCIE:*NET* - struktura wielowarstwowej sieci jednokierunkowej, z nowymi wartościami

progów i wag synaptycznych, obliczonych za pomocą funkcji inicjalizacyjnej

**sim** - Symulacja sieci neuronowej dla zadanych danych wejściowych.

SIM Funkcja służy do wyznaczenia wyjść sieci neuronowej dla zadanej macierzy

danych wejściowych. Argumenty *Pi*, *Ai*, *Pf*, *Af* są opcjonalne i nie będą używane

przez sieci nieliniowe, wykorzystywane w ćwiczeniach

**train** - Trening sieci neuronowej.

TRAIN Funkcja realizuje trening sieci neuronowej, wykorzystując funkcję treningu, której

nazwa została podana w polu NET.trainFcn, zaś wartości niezbędnych

parametrów – w polu NET.trainParam. Funkcja treningu jest uniwersalna,

wywoływana w jednolity sposób dla wszystkich typów sieci neuronowych, stąd

też niektóre argumenty wejściowe nie mają znaczenia dla sieci

jednokierunkowych (bez sprzężeń zwrotnych i opóźnień w torach sygnałów

wejściowych)

**adapt** - Adaptacja wag sieci neuronowej.

ADAPT Funkcja realizuje w swej podstawowej postaci jeden krok (tzw. epokę) treningu

sieci neuronowej, dokonując adaptacji wag synaptycznych sieci. Do adaptacji jest

wykorzystywana funkcja, której nazwa została podana w polu NET.adaptFcn, zaś

wartości niezbędnych parametrów tej funkcji – w polu NET.adaptParam. Funkcja

adaptacji jest uniwersalna, wywoływana w jednolity sposób dla wszystkich typów

sieci neuronowych, stąd też niektóre argumenty wejściowe nie mają znaczenia dla

sieci jednokierunkowych (bez sprzężeń zwrotnych i opóźnień w torach sygnałów

wejściowych)

**newff** - Tworzenie wielowarstwowej jednokierunkowej sieci neuronowej,

złożonej z neuronów o nieliniowych funkcjach aktywacji. ( wykorzystamy do stworzenia sieci jednowarstwowej)

NEWFF Funkcja tworzy wielowarstwową sieć neuronową; każda warstwa składa się z

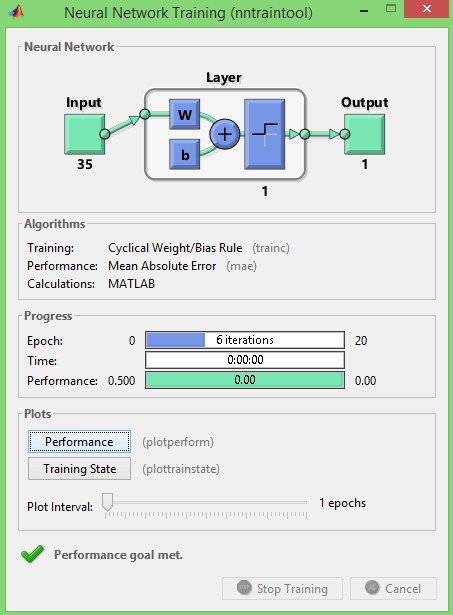
zadanej liczby neuronów o nieliniowych funkcjach aktywacji (jakkolwiek funkcje

aktywacji w poszczególnych warstwach mogą mieć również postać liniową).

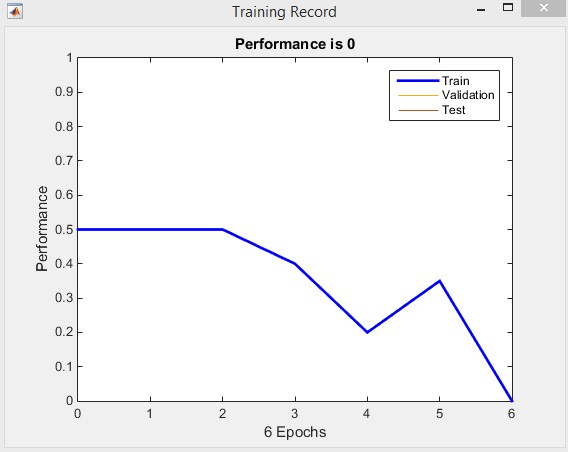
**c) Uczenie sieci dla przy różnych współczynnikach uczenia dla sieci newp.**

**Przy pomocy funkcji train:**

Dla podstawowych wag:



Rysunek 6- Otrzymane wyniki

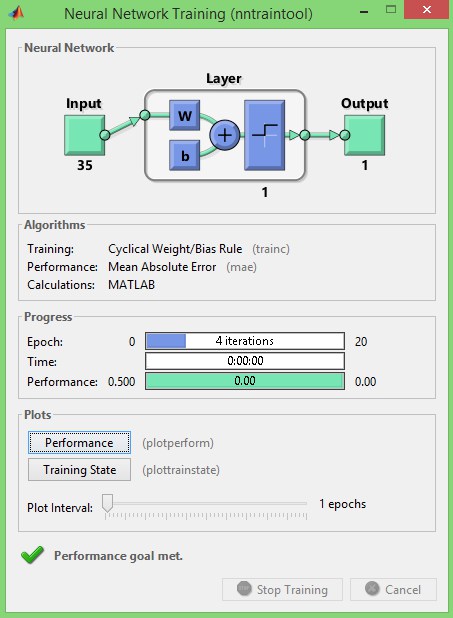


Rysunek 7 - Wykres wydajności.

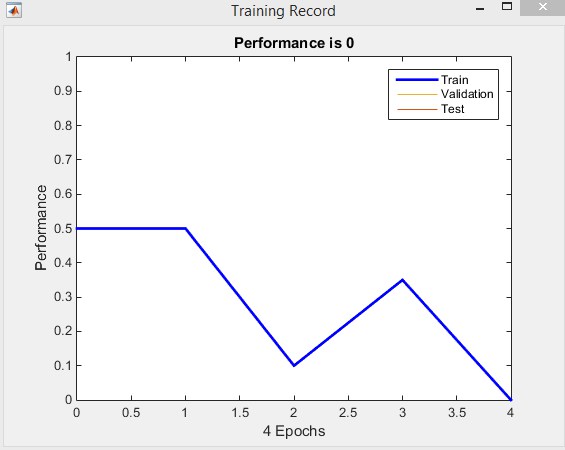
Dla ustalonych przez nas współczynników uczeni:



Rysunek 8- Wagi



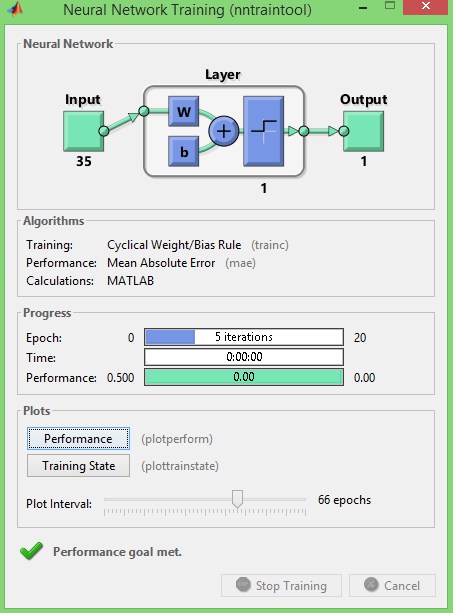
Rysunek 9 - Otrzymane wyniki



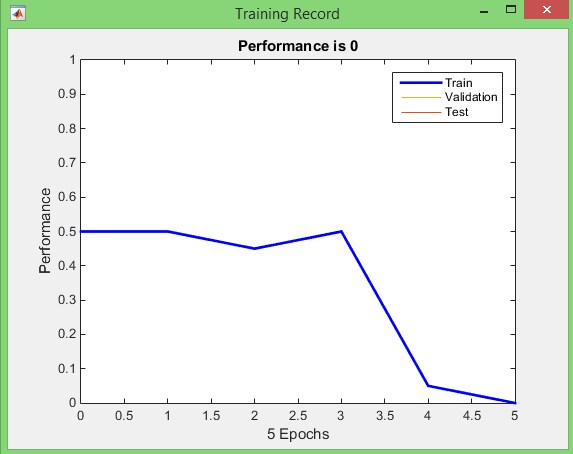
Rysunek 10 - Wykres wydajności.



Rysunek 11 - Wagi



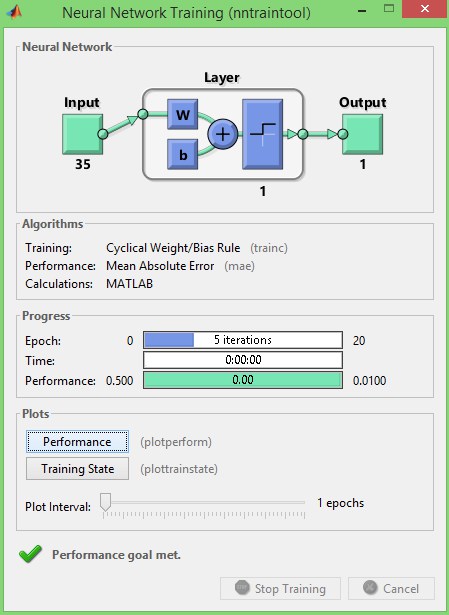
Rysunek 12 - Otrzymane wyniki



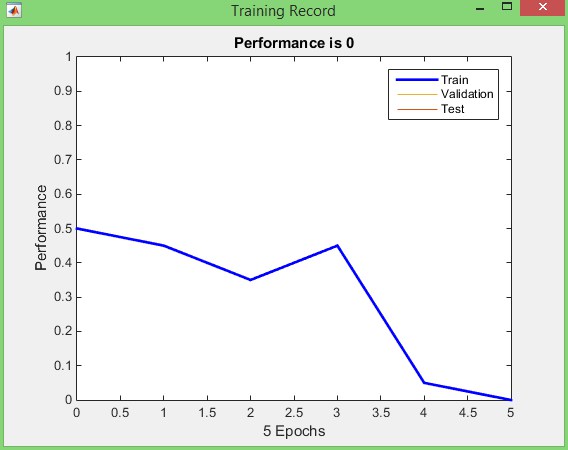
Rysunek 13 - Wykres wydajności.



Rysunek 14 - Wagi



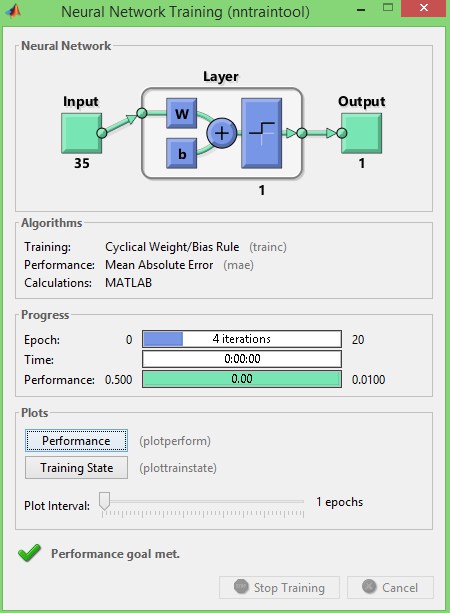
Rysunek 15- Otrzymane wyniki



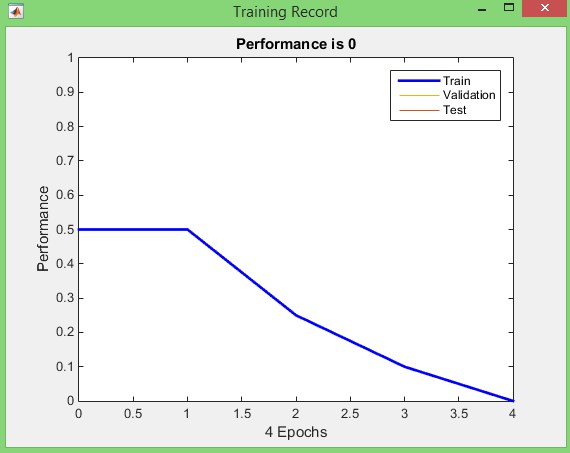
Rysunek 16 - Wykres wydajności.



Rysunek 17- Wagi

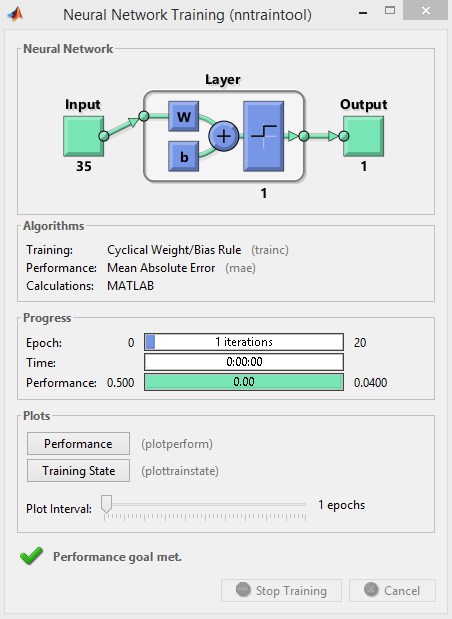


Rysunek 18- Otrzymane wyniki

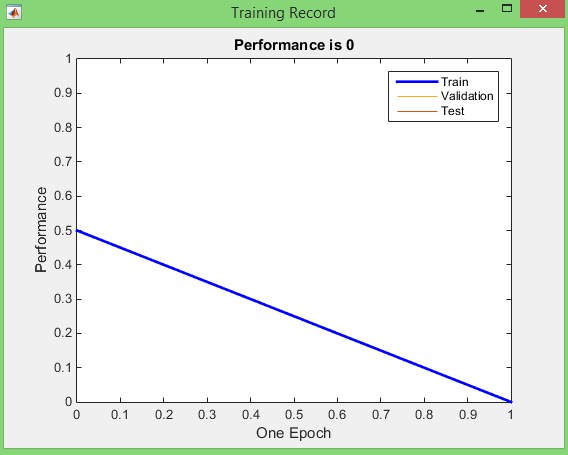


Rysunek 19 - Wykres wydajności.

**Przy pomocy funkcji adapt i train :**

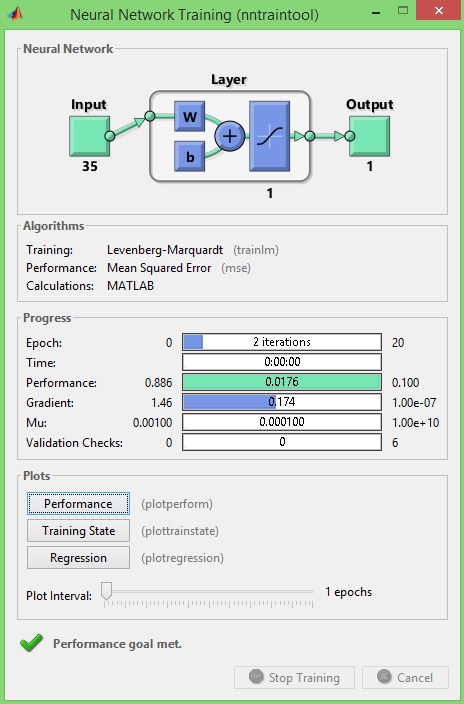


Rysunek 20 - Otrzymane wyniki

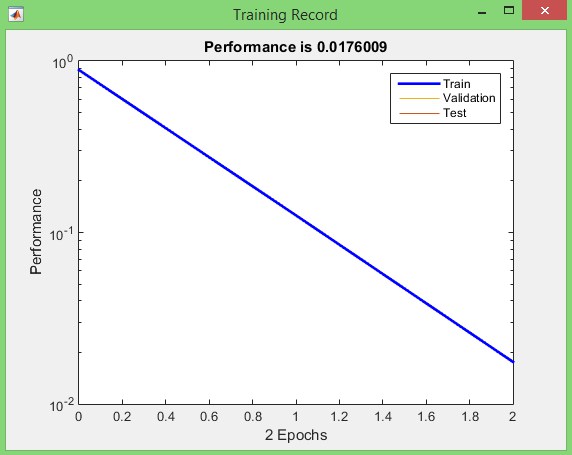


Rysunek 21 - Wykres wydajności.

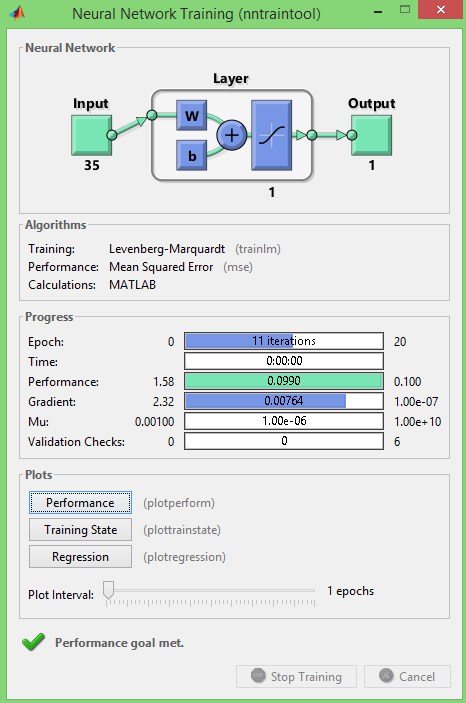
d) Uczenie sieci dla przy różnych współczynnikach uczenia dla sieci newff.



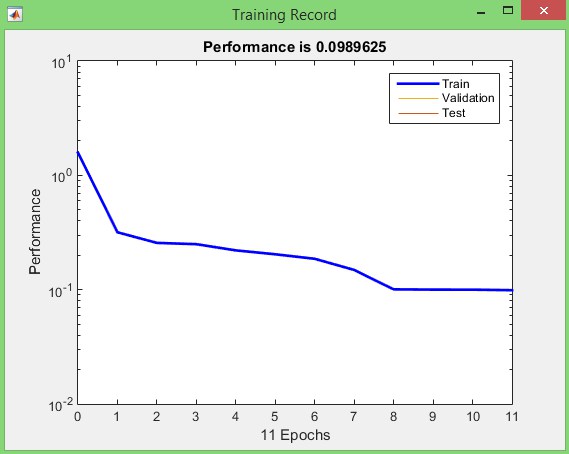
Rysunek - Otrzymane wyniki



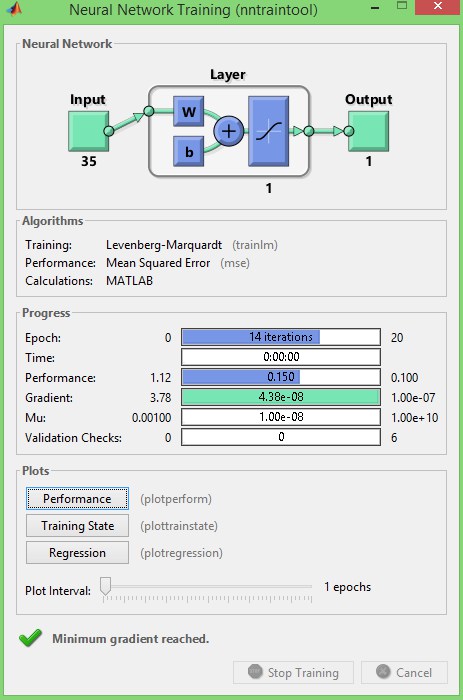
Rysunek - Wykres wydajności.



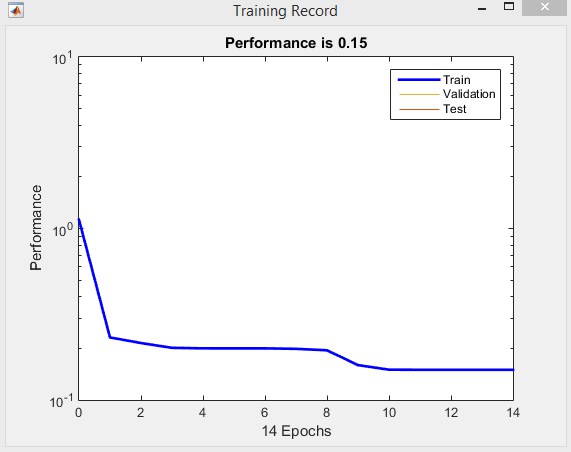
Rysunek - Otrzymane wyniki



Rysunek - Wykres wydajności.



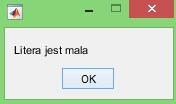
Rysunek - Otrzymane wyniki



Rysunek - Wykres wydajności.

**e) Testowanie sieci.**

Sieć można było przetestować w bardzo prosty sposób, mianowicie po udanej nauce sprawdzaliśmy sobie czy podana przez nas litera na początku w zmiennej jest mała czy duża. Otrzymany wynik :



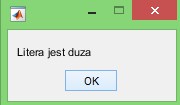
Rysunek 28 - testowanie

Inny przykład :

Dane do sprawdzenia:



Otrzymany wynik:



Rysunek 29 – Testowanie

f) Listing całego kodu programu lub zrzuty konfiguracji i wykorzystania programu.

**Program pierwszy z funkcja newp:**

close all;clear all;clc

% Lista danych uczacych sie

A = [0 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1];

B = [1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0];

C = [0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 1 0];

D = [1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0];

E = [1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1];

F = [1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0];

G = [0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 1 0 1 1 1 0];

H = [1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1];

I = [0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 0];

J = [1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0];

a = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 1 0 1 0 0 1 0 1 1 1 1 1];

b = [1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0];

c = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0];

d = [0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 1 1 1 1 0 0 0 1 1 0 0 0 1 0 1 1 1 1];

e = [0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1];

f = [0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0];

g = [0 1 1 1 0 0 1 0 1 0 0 1 1 1 1 0 0 0 1 0 0 0 1 1 0 0 1 0 1 0 0 1 1 1 0];

h = [1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 1 0 1 0 0 1 0 1 0 0 1 0];

i = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0];

j = [0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0];

litery=[A;B;C;D;E;F;G;H;I;J;a;b;c;d;e;f;g;h;i;j];

litery=litery';

spr =j'; % litera która będziemy sprawdzać czy nauczył się określać wielkość liter

wy=[1;1;1;1;1;1;1;1;1;1;0;0;0;0;0;0;0;0;0;0];% oczekiwane wartości sygnałów wyjściowych

wy=wy';

net2 = newp([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;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1], 1); % utworzenie perceptronu

net2 = init(net2);%Inicjalizacja struktury sieci neuronowej.

Y = sim(net2,litery); %Symulacja sieci neuronowej dla zadanych danych wejściowych.

net2 = adapt(net2, litery, wy); % Adaptacja wag sieci neuronowej.

net2.trainParam.epochs = 20; % Ustawienie ilości epok podczas treningu

net2.trainParam.goal = 0.04;%kryterium stopu (sumę kwadratów błędów wyjść sieci)

[net2 tr] = train(net2,litery,wy); % Trening sieci neuronowej.

Y = sim(net2,litery); %Symulacja sieci neuronowej dla zadanych danych wejściowych.

Y=sim(net2,spr) %Symulacja sieci neuronowej dla zadanych danych wejściowych.

Y=Y';

plotperf(tr) % Wykreśla przebieg treningu sieci neuronowej

if Y==1

msgbox('Litera jest duza');

else

msgbox('Litera jest mala');

end

**Program pierwszy z funkcja newff:**

close all;clear all;clc

% Lista danych uczacych sie

A = [0 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1];

B = [1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0];

C = [0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 1 0];

D = [1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0];

E = [1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1];

F = [1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0];

G = [0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 1 0 1 1 1 0];

H = [1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1];

I = [0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 0];

J = [1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0];

a = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 1 0 1 0 0 1 0 1 1 1 1 1];

b = [1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0];

c = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0];

d = [0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 1 1 1 1 0 0 0 1 1 0 0 0 1 0 1 1 1 1];

e = [0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1];

f = [0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0];

g = [0 1 1 1 0 0 1 0 1 0 0 1 1 1 1 0 0 0 1 0 0 0 1 1 0 0 1 0 1 0 0 1 1 1 0];

h = [1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 1 0 1 0 0 1 0 1 0 0 1 0];

i = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0];

j = [0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0];

litery=[A;B;C;D;E;F;G;H;I;J;a;b;c;d;e;f;g;h;i;j];

litery=litery';

spr =d'; % litera która będziemy sprawdzać czy nauczył się określać wielkość liter

wy=[1;1;1;1;1;1;1;1;1;1;0;0;0;0;0;0;0;0;0;0];% oczekiwane wartości sygnałów wyjściowych

wy=wy';

net2 = newff([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;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1], [1]); % utworzenie sieci jednowarstwowej

net2 = init(net2);%Inicjalizacja struktury sieci neuronowej.

Y = sim(net2,litery); %Symulacja sieci neuronowej dla zadanych danych wejściowych.

net2.trainParam.epochs = 20; % Ustawienie ilości epok podczas treningu

net2.trainParam.goal = 0.1;%kryterium stopu (sumę kwadratów błędów wyjść sieci)

[net2 tr] = train(net2,litery,wy); % Trening sieci neuronowej.

Y = sim(net2,litery); %Symulacja sieci neuronowej dla zadanych danych wejściowych.

Y=sim(net2,spr) %Symulacja sieci neuronowej dla zadanych danych wejściowych.

Y=Y';

plotperf(tr) % Wykreśla przebieg treningu sieci neuronowej

if Y>0.7

msgbox('Litera jest duza');

else

msgbox('Litera jest mala');

end

**Spostrzeżenia:**

Patrząc na rysunki 6,9,12,15,18 które były otrzymane za pomoca funkcji train zauważyłem ze sieć nauczyła się określać wielość liter w bardzo dobrym czasie bo wąchało się to w 4-6 iteracjach. Natomiast jeśli spojrzymy na rysunek 20 który otrzymaliśmy za pomaca train i adapt dostaliśmy ten sam efekt w zaledwie1 iteracji.

Zauważyłem ze podczas wprowadzania własnych wag liczba iteracji się zmniejszała rysunek 13,10

Zauważyłem ze wykres uczenia się przy pomocy funkcji train był najładniejszy podczas podania wag z rysunku 17.

Zauważyłem ze najlepiej uczył się przy pomocy train i adapt ponieważ otrzymaliśmy liniowy wykres rysunek 21.

Zauważyłem ze funkcja newff działa podobnie do funkcji newp.

Zauważyłem ze w funkcji newff występują niestabilności i czasem może się nie nauczyć rysunek 26.

**Wnioski:**

Przy pomocy funkcji train mogliśmy otrzymać zadowalające efekty ale jednak dzięki adapt otrzymaliśmy najlepsze wyniki podczas uczenia

Funkcja adapt dużo lepiej wpływa na postępy w uczeniu.

Dzięki dobieraniu wag otrzymujemy lepsze wyniki uczenia.

Funkcja newff nie zawsze nam się nauczy.

**Bibiliografia:**

[www.wikipedia.org](http://www.wikipedia.org)

Wykłady