**Hubert Bednarski**Podstawy Sztucznej Inteligencji  
IS, semestr V, gr. Proj. 1  
Data oddania: 15.12.2017r.

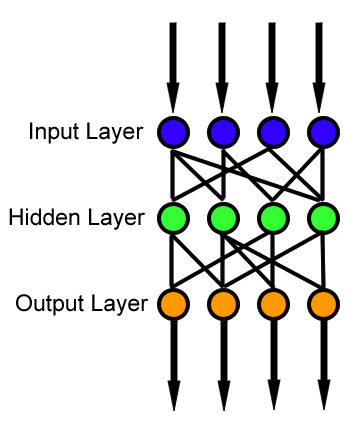
**Budowa i działanie sieci wielowarstwowej typu feedforward**sprawozdanie

Cel ćwiczenia:

Celem ćwiczenia było poznanie budowy i działania wielowarstwowych sieci neuronowych poprzez uczenie kształtu wykresu funkcji matematycznej z użyciem algorytmu wstecznej propagacji błędu.

Opis:

Sieć neuronowa wielowarstwowa typu Feedforward:



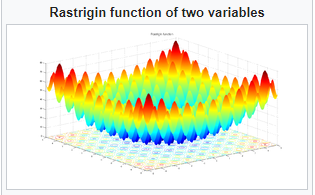
W sieci tej istnieje ściśle określony kierunek przepływu sygnałów, od pewnego ustalonego wejścia do wyjścia, na którym podawane jest rozwiązanie. Neurony są zorganizowane w warstwy tak, że główne połączenia i zawiązane z nimi przepływy sygnałów odbywają się pomiędzy elementami sąsiednich warstw. Sieć składa się z warstwy wejściowej, co najmniej jednej warstwy ukrytej oraz warstwy wyjściowej. Warstwy ukryte służą do przetwarzania sygnałów w sieci neuronowej. Każda warstwa jest powiązana tylko z warstwą poprzednią i następną. Dane wyjściowe każdego neuronu w jednej warstwie są jednocześnie danymi wejściowymi dla neuronów w kolejnej warstwie na zasadzie każdy z każdym. Sygnał wyjściowy nie jest dzielony - jest podawany taki sam na wejściach wszystkich neuronów kolejnej warstwy. Neurony w jednej warstwie nie są ze sobą w żaden sposób połączone.

Algorytm wstecznej propagacji błędu:

Każdy neuron składa się z elementu sumującego iloczyny wag i sygnałów wejściowych oraz z elementu nieliniowego, którego charakterystyka przejściowa jest funkcją aktywacji neuronu. Proces uczenia przebiega iteracyjnie dla kolejnych danych uczących, w każdym kroku współczynniki wagowe połączeń ulegają modyfikacji. Propagacja sygnałów następuje przez warstwę wejściową, warstwę ukrytą i ostatecznie wyjściową. W kolejnym kroku porównujemy wartość oczekiwaną z wartością sygnału wyjściowego. Różnica wartości to sygnał błędu warstwy wyjściowej. Do neuronów leżących w ukrytych warstwach sieci nie ma możliwości bezpośredniego określenia błędu, ponieważ nie są znane dla nich wartości oczekiwane sygnałów wyjściowych. Korzysta się z wyznaczonego wcześniej sygnału błędu rzutując go wstecz do wszystkich neuronów, których sygnały stanowiły wejścia dla rozważanego neuronu. Mając określony błąd sygnału przystępuje się do modyfikacji wag jego połączeń wejściowych.

Funkcja Rastrigin 3D:

Przykład nieliniowej multimodalnej funkcji. Funkcja ta posiada wartość najmniejszą w punkcie (0,0), jednak zanim algorytm przeszukiwania znajdzie to minimum globalne, może napotkać wiele minimów lokalnych, co może stwarzać wiele problemów przy szukaniu minimum globalnego.



**Wykonanie ćwiczenia:**

Ćwiczenie wykonałem przy użyciu oprogramowania NeurophStudio. Stworzyłem w nim sieci wielowarstwowe (Multi Layer Perceptron). Do nauczania i testowania wykorzystywałem dane różniące się od siebie. Dodatkowo dane zostały przeze mnie znormalizowane. Dane uczące i testujące wygenerowałem przy użyciu poniższego kodu (własna implementacja w c++):

#include <iostream>

#include <time.h>

#include <stdlib.h>

#include <fstream>

#define \_USE\_MATH\_DEFINES

#include <math.h>

#define ILE\_REKORDOW 100

**using** **namespace** std**;**

float Rastrigin**(**float**,**float**,**float**);**

int main**()** **{**

srand**(**time**(NULL));**

fstream plik**;**

plik**.**open**(**"dane uczace.txt"**,** std**::**ios**::**out**);**

float**\*** x1 **=** **new** float**[**ILE\_REKORDOW**];**

float**\*** x2 **=** **new** float**[**ILE\_REKORDOW**];**

float**\*** x3 **=** **new** float**[**ILE\_REKORDOW**];**

**for** **(**int i **=** 0**;** i **<** ILE\_REKORDOW**;** i**++)**

**{**

x1**[**i**]** **=** rand**()** **%** 4000 **-** 2000**;**

x1**[**i**]** **/=** 1000**;**

plik **<<** x1**[**i**];**

plik **<<** ";"**;**

x2**[**i**]** **=** rand**()** **%** 4000 **-** 2000**;**

x2**[**i**]** **/=** 1000**;**

plik **<<** x2**[**i**];**

plik **<<** ";"**;**

x3**[**i**]** **=** rand**()** **%** 4000 **-** 2000**;**

x3**[**i**]** **/=** 1000**;**

plik **<<** x3**[**i**];**

plik **<<** ";"**;**

float ras **=** Rastrigin**(**x1**[**i**],** x2**[**i**],** x3**[**i**]);**

plik **<<** ras**;**

plik **<<** endl**;**

**}**

plik**.**close**();**

**return** 0**;**

**}**

float Rastrigin**(**float x1**,** float x2**,** float x3**)**

**{**

double temp**;**

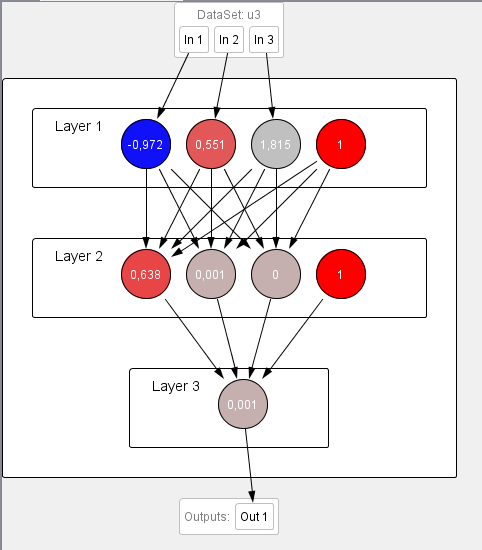
double A **=** 10**;**

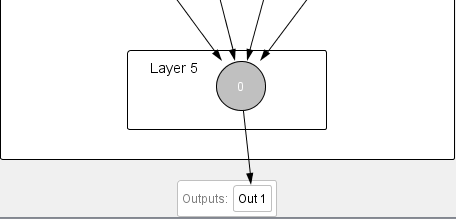
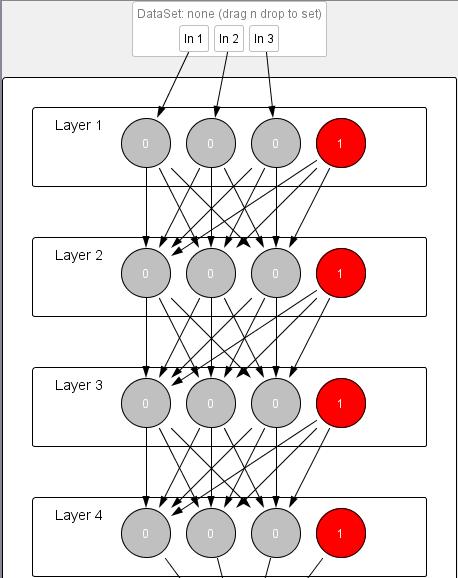
temp **=** 3 **\*** A **+** **(**pow**(**x1**,** 2**)** **-** A **\*** cos**(**2 **\***M\_PI**\*** x1**))** **+** **(**pow**(**x2**,** 2**)** **-** A **\*** cos**(**2 **\*** M\_PI **\*** x2**))** **+** **(**pow**(**x3**,** 2**)** **-** A **\*** cos**(**2 **\*** M\_PI **\*** x3**));**

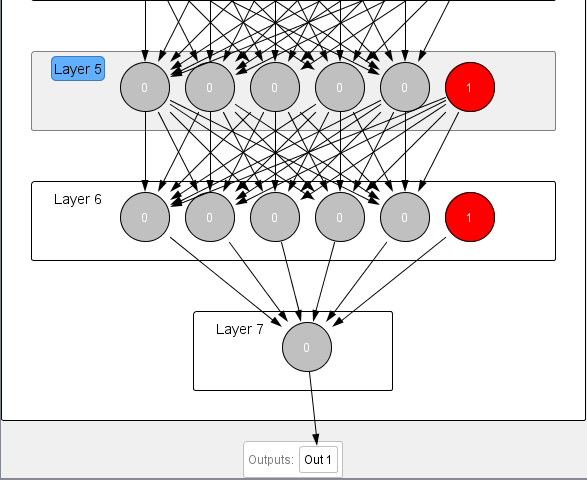
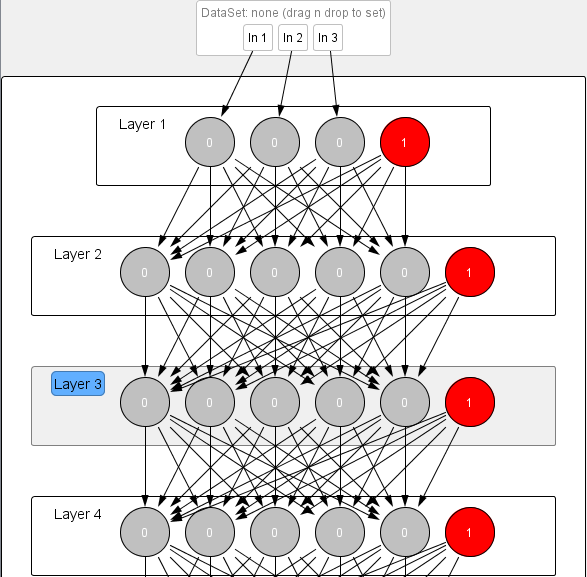
**return** **(**float**)**temp**;**

**}**

Stworzone sieci wielowarstwowe(każda z sieci z 3 wejściami i 1 wyjściem):

  
sieć1 : 1 warstwa ukryta z 3 neuronami.

  
sieć2: 3 warstwy ukryte po 3 neurony każda.

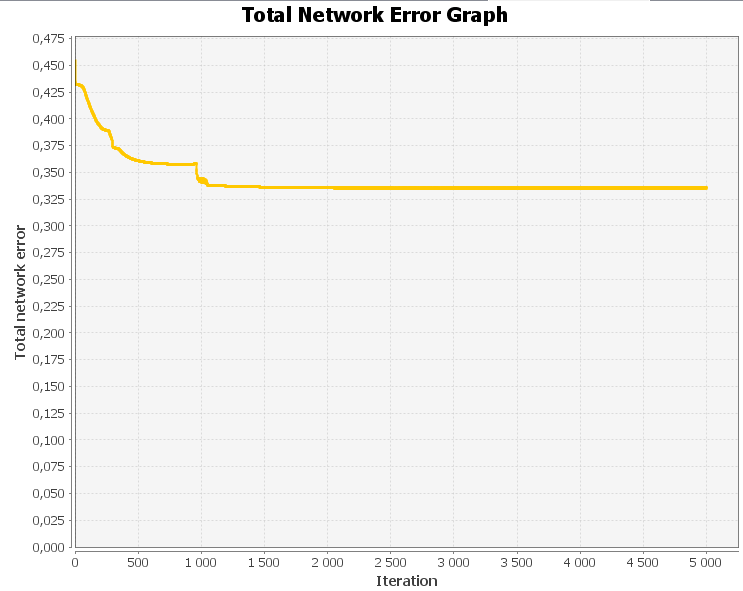
  
sieć3: 5 warstw ukrytych po 5 neuronów każda

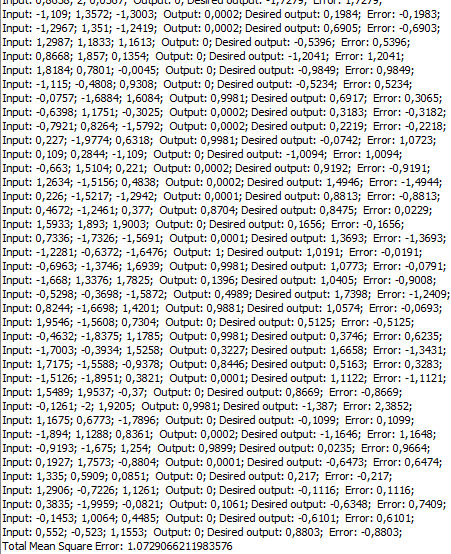
Nauczanie i Testy:

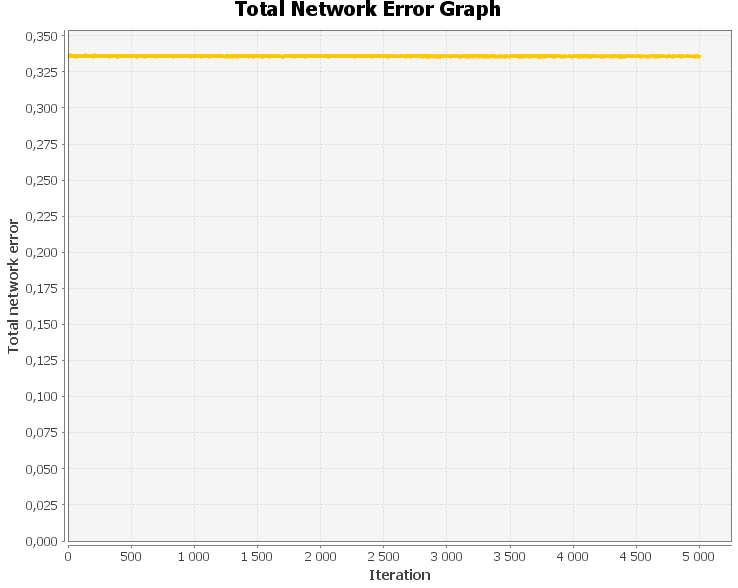
Do nauczania przyjąłem maksymalną wartość liczby iteracji wynoszącą 5000 oraz maksymalną wartość błędu równą 0,01.

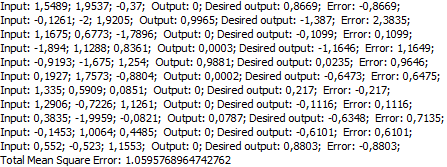
**Dla sieci1:**

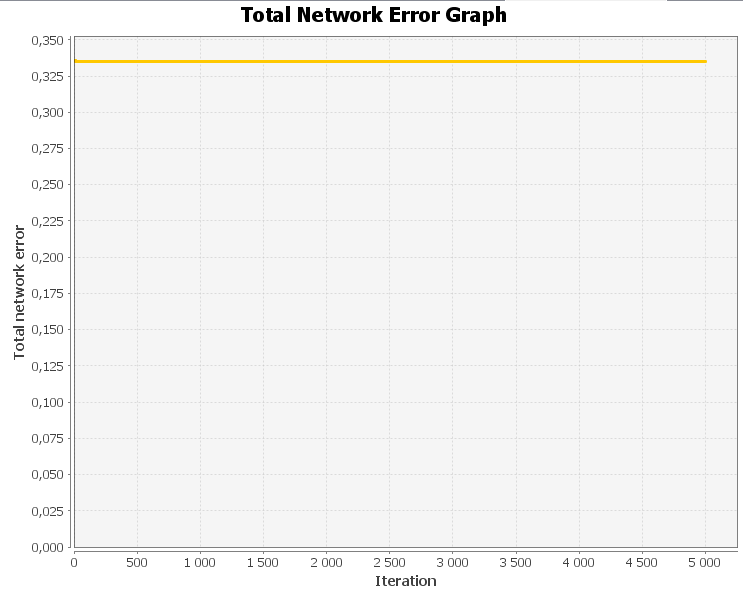
1.learning rate:0,2; liczba danych uczących i testujących: 100.

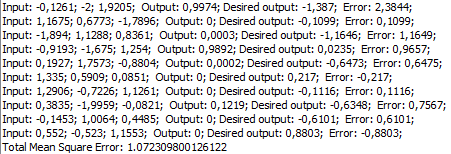
wykres dla nauczania:  
 

Wycinek wartości z testowania:  


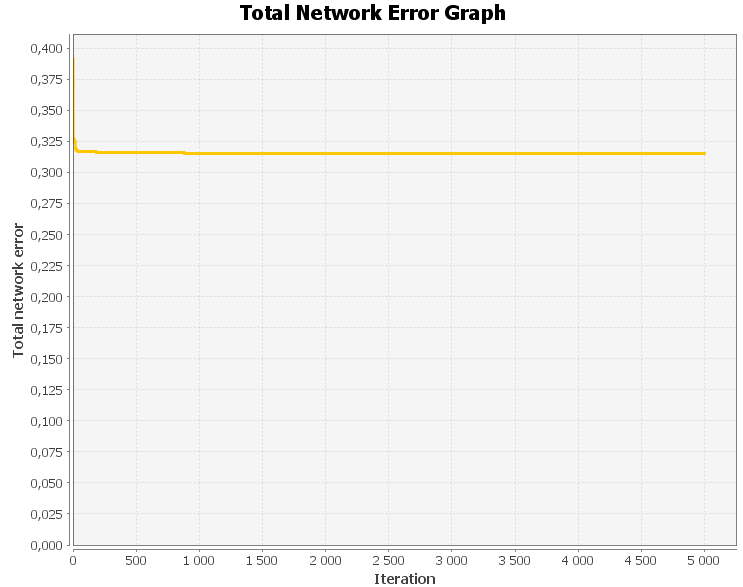
2. learning rate:0,6; liczba danych uczących i testujących: 100.

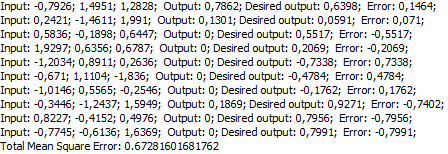


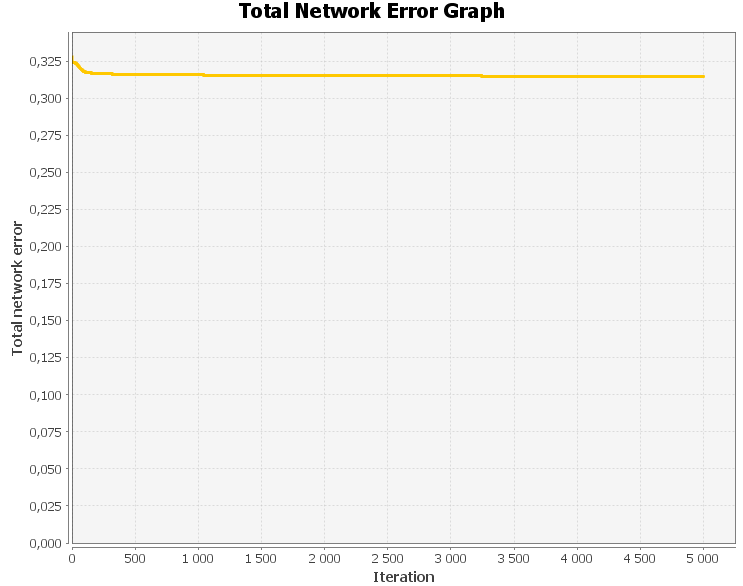
3. learning rate:0,02; liczba danych uczących i testujących: 100.  


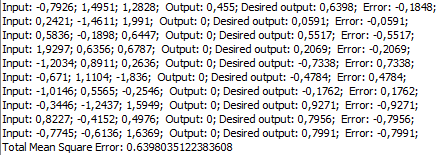


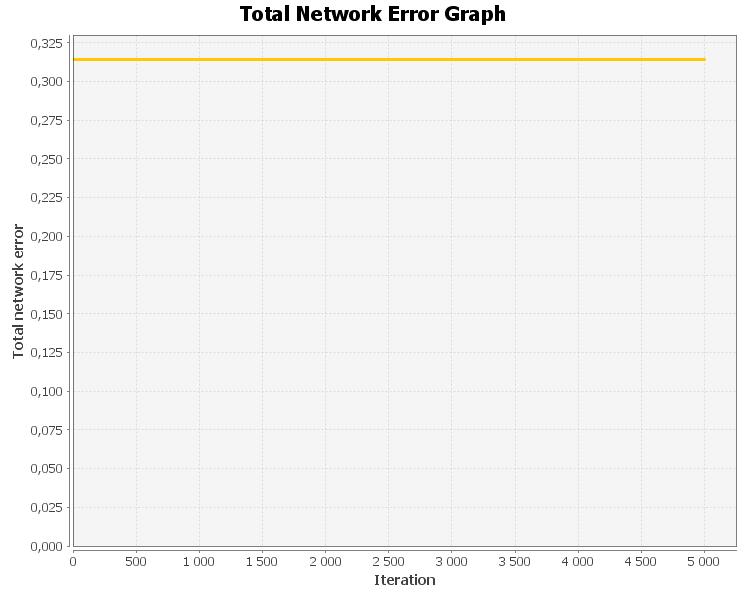
Podczas nauczania drugiego wartość błędu zmieniała się skokowo na przemian rosnąc i malejąc, co ogólnie, w trakcie całego nauczania, nie zmieniało zbytnio jej wartości. Z kolei w trzecim nauczaniu wartość błędu spadała stale, ale bardzo powoli.

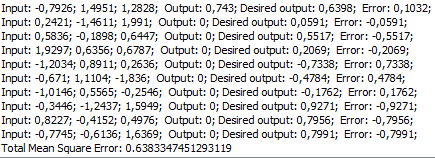
4. learning rate:0,02; liczba danych uczących i testujących: 1000.  


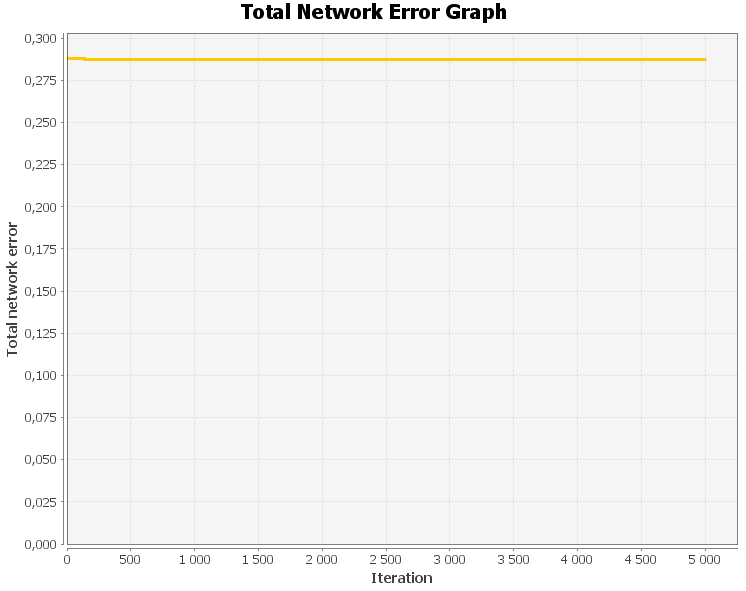


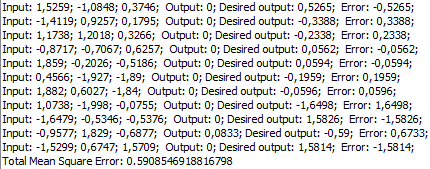
5. learning rate:0,5; liczba danych uczących i testujących: 1000.

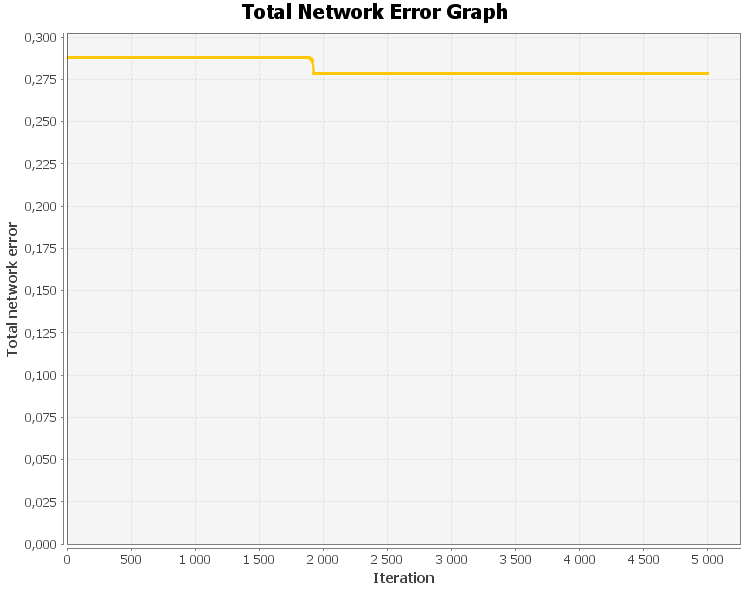


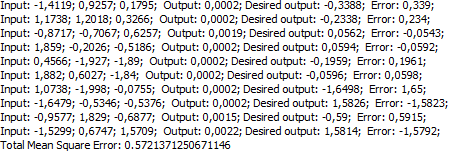
6. learning rate:0,02; liczba danych uczących i testujących: 1000.  


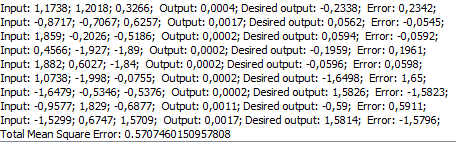
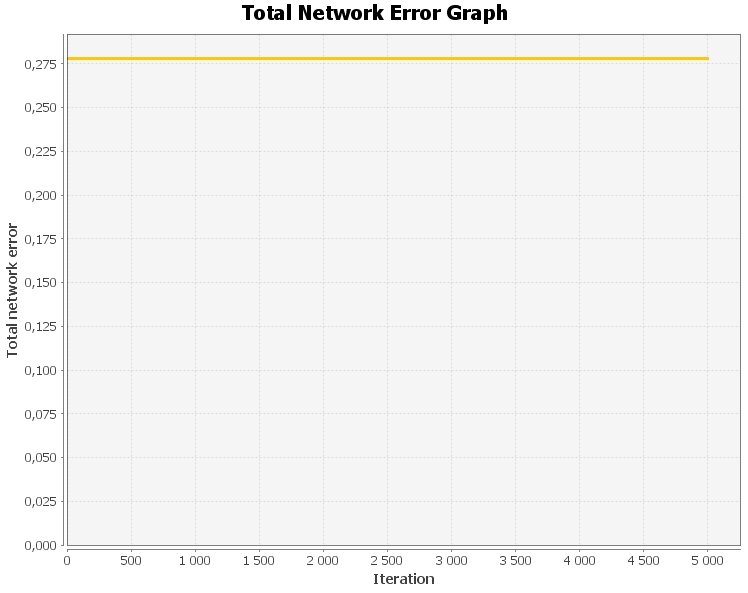


7. learning rate:0,2; liczba danych uczących i testujących: 10000. 



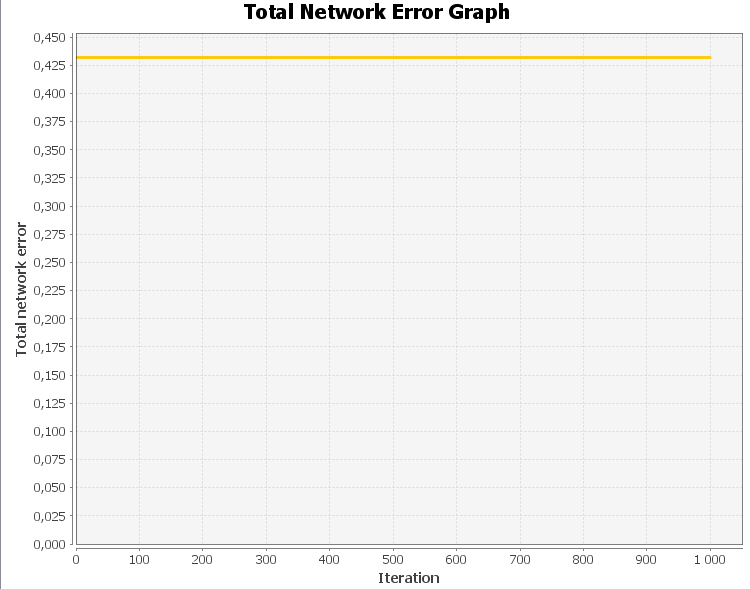
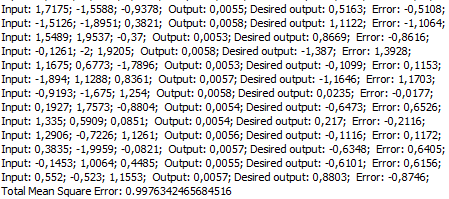
8. learning rate:0,5; liczba danych uczących i testujących: 10000.  


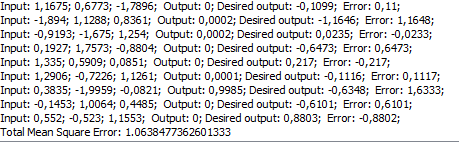
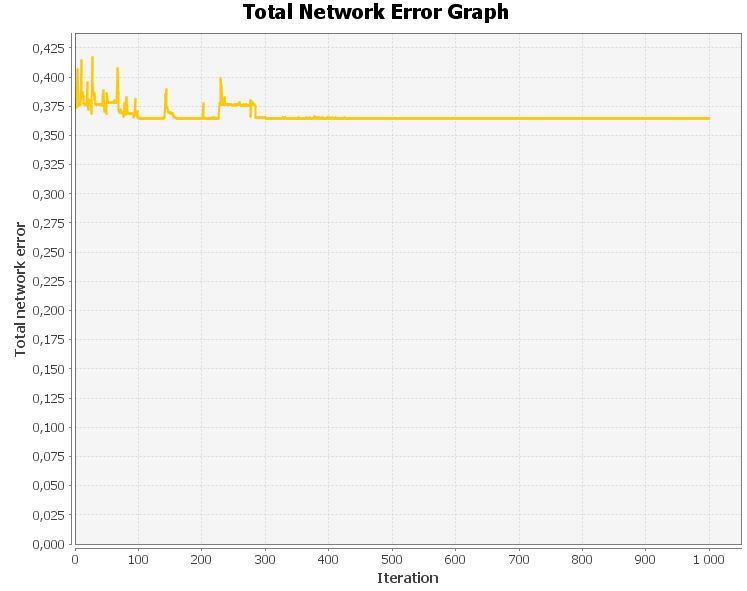


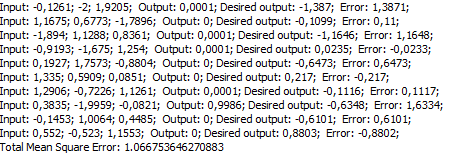
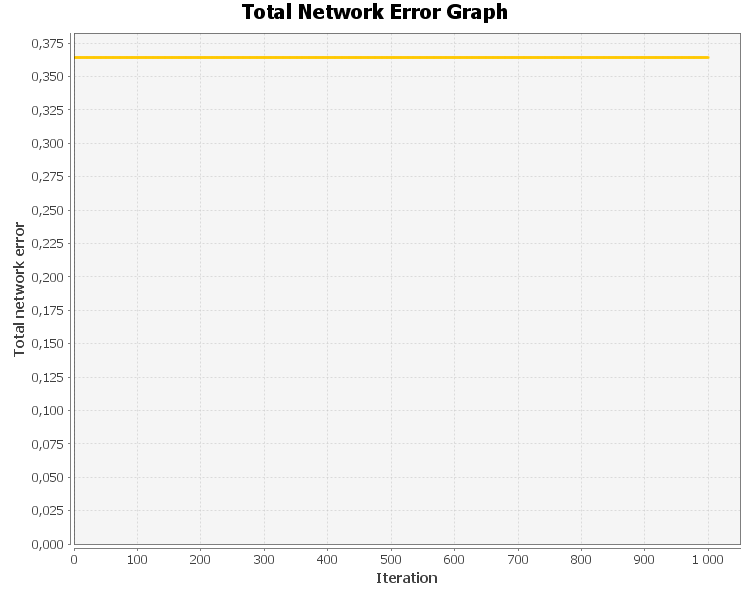
9. learning rate:0,03; liczba danych uczących i testujących: 10000. 

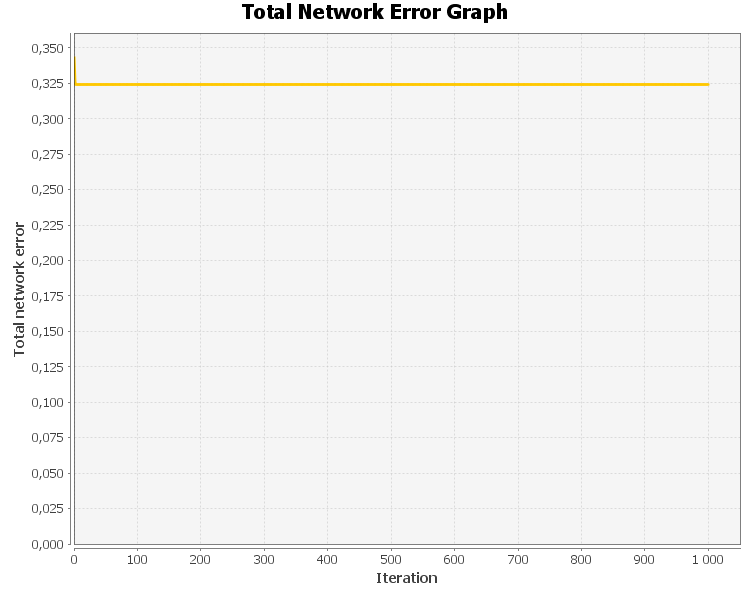
**Dla sieci2(3 warstwy ukryte po 3 neurony):**

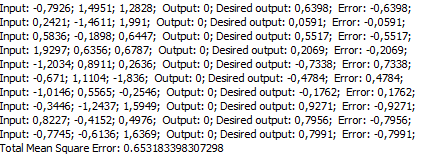
Została zmieniona wartość maksymalna iteracji z 5000 na 1000.

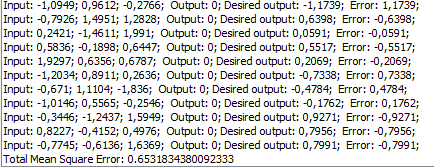
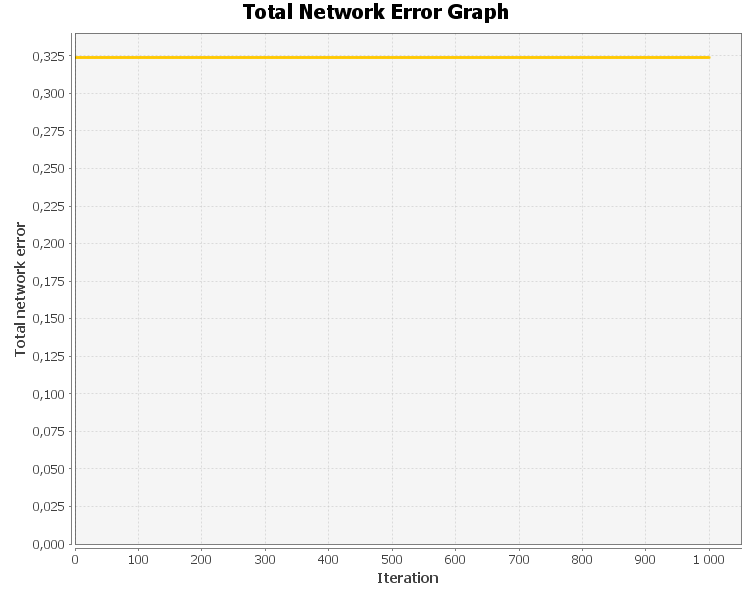
1.learning rate:0,2; liczba danych uczących i testujących: 100.  

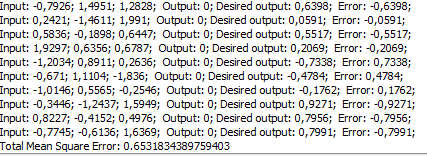
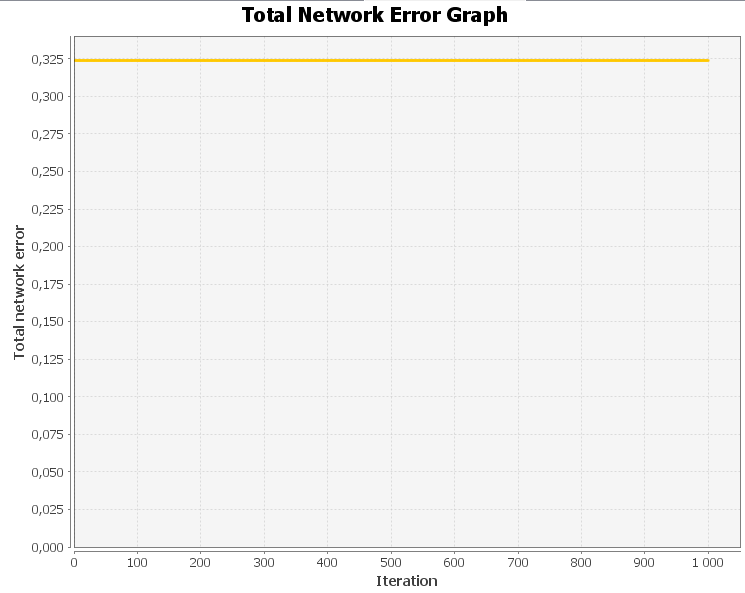
2.learning rate:0,5; liczba danych uczących i testujących: 100. 

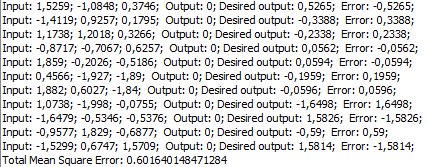
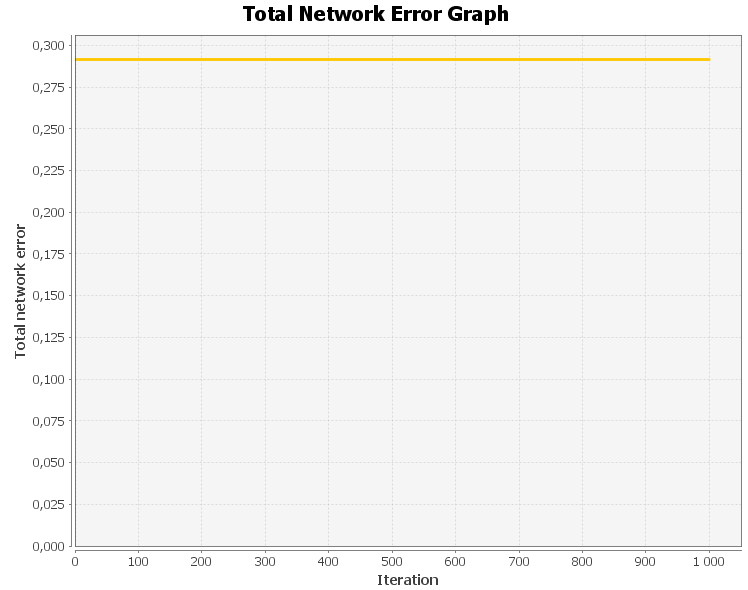
3.learning rate:0,02; liczba danych uczących i testujących: 100. 

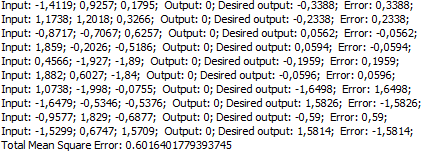
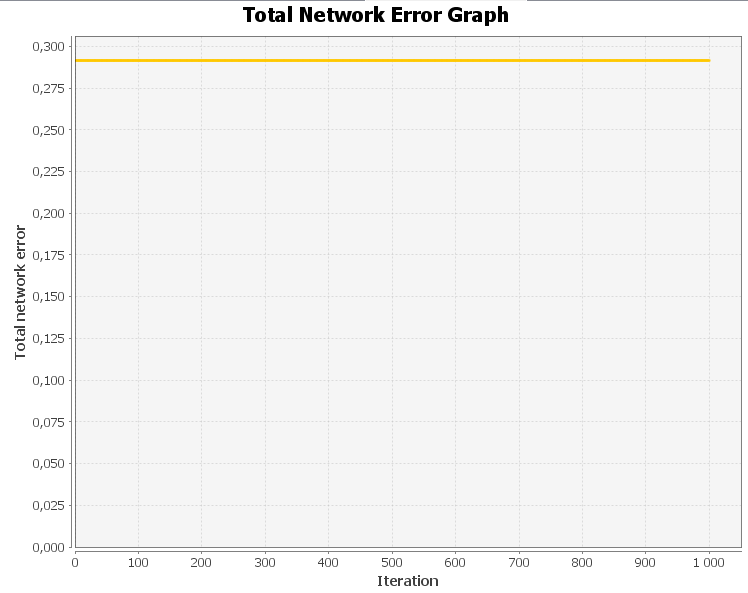
4.learning rate:0,2; liczba danych uczących i testujących: 1000. 

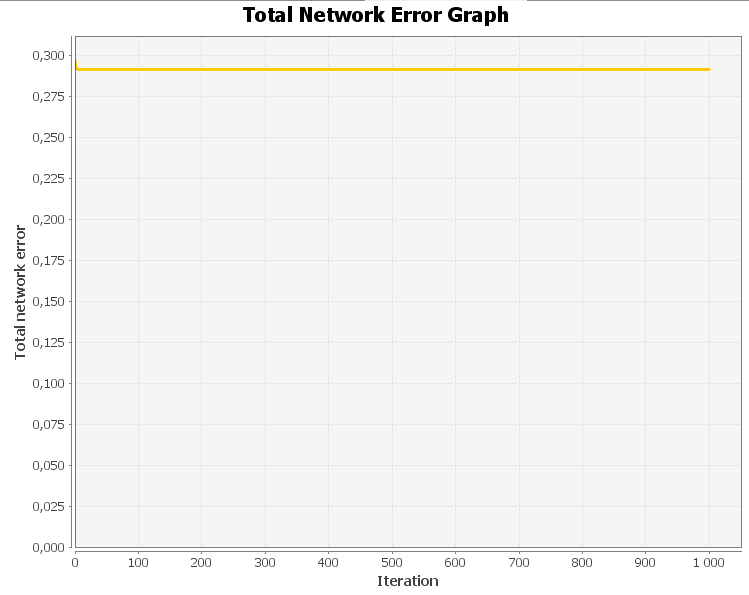
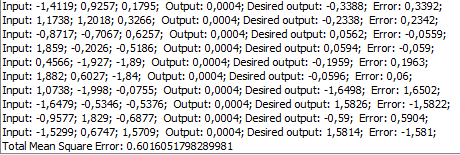


5.learning rate:0,5; liczba danych uczących i testujących: 1000. 

6.learning rate:0,02; liczba danych uczących i testujących: 1000. 

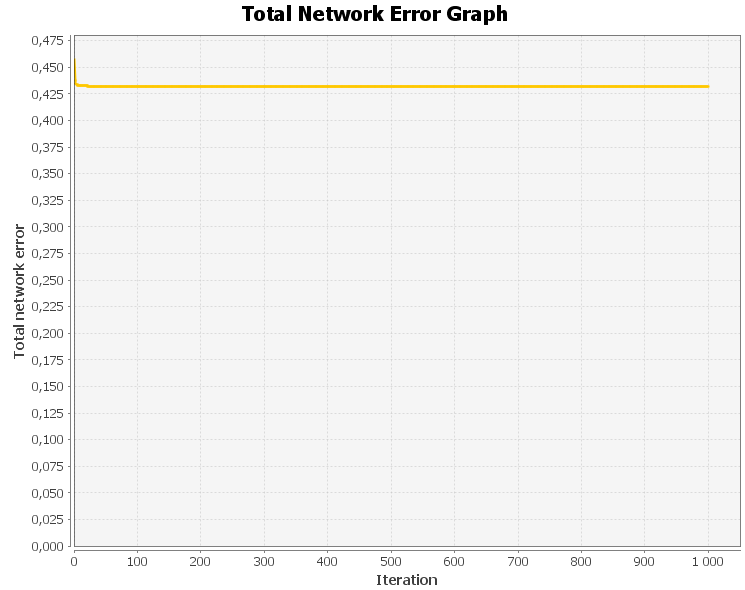
7.learning rate:0,2; liczba danych uczących i testujących: 10000. 

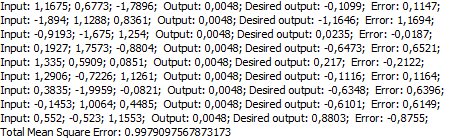
8.learning rate:0,5; liczba danych uczących i testujących: 10000. 

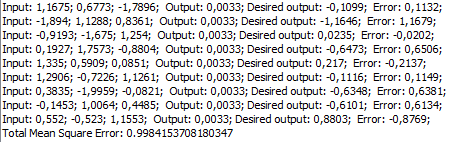
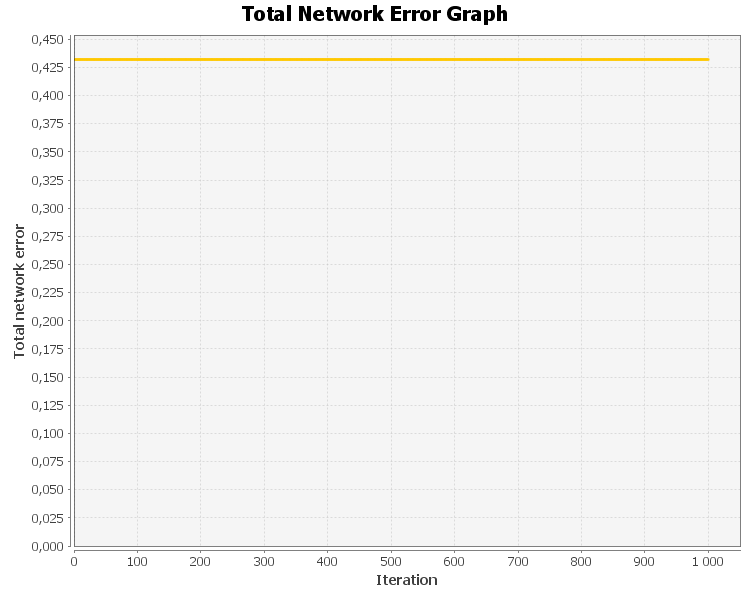
9.learning rate:0,02; liczba danych uczących i testujących: 10000.  

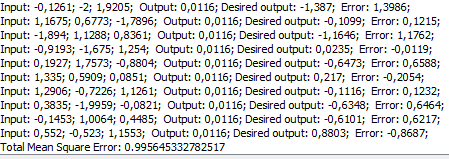
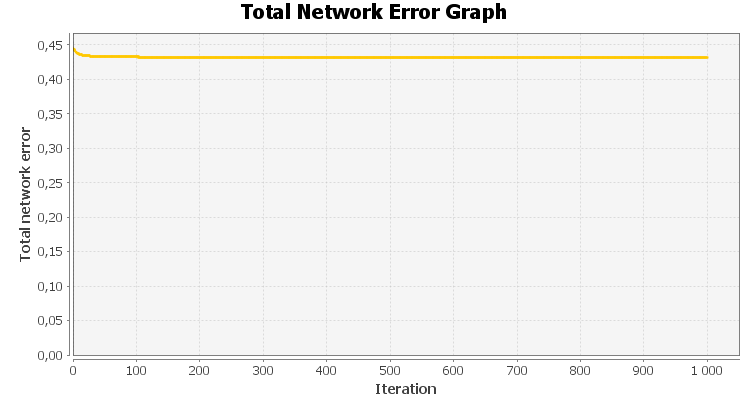
**Dla sieci3(5 warstw ukrytych po 5 neuronów):**

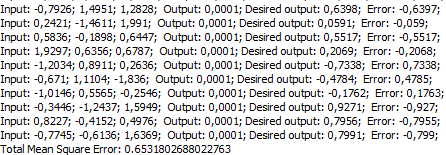
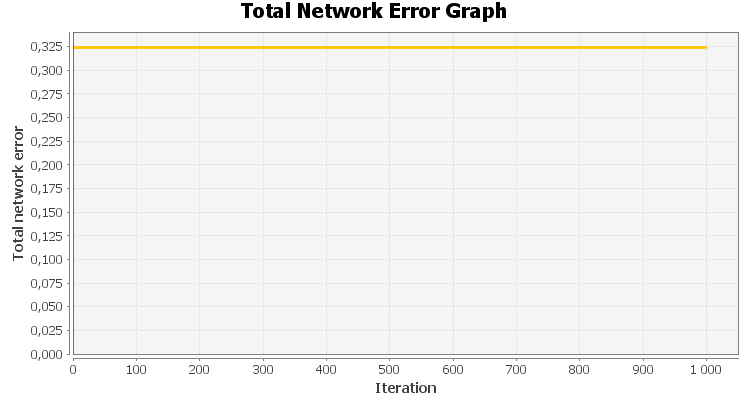
Wartość maksymalna iteracji: 1000.

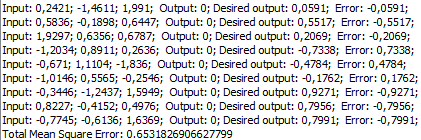
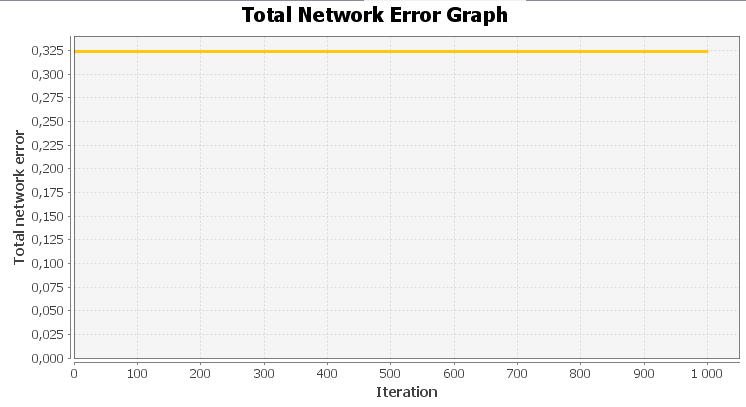
1.learning rate:0,2; liczba danych uczących i testujących: 100. 

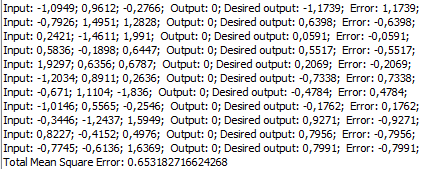
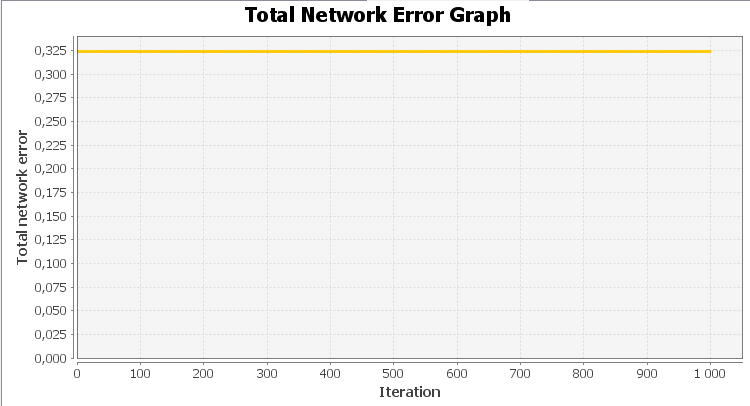


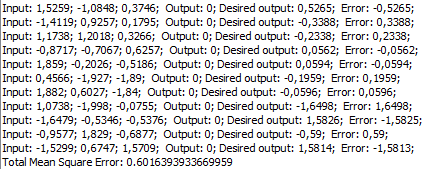
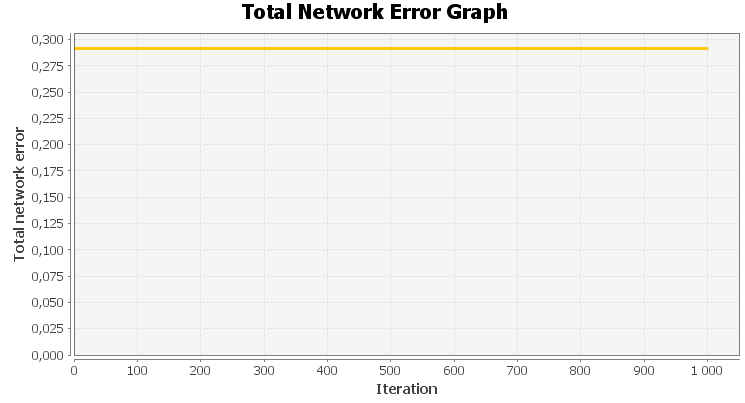
2.learning rate:0,5; liczba danych uczących i testujących: 100. 

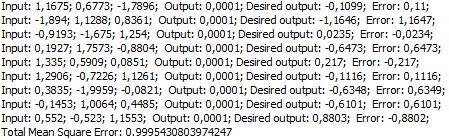
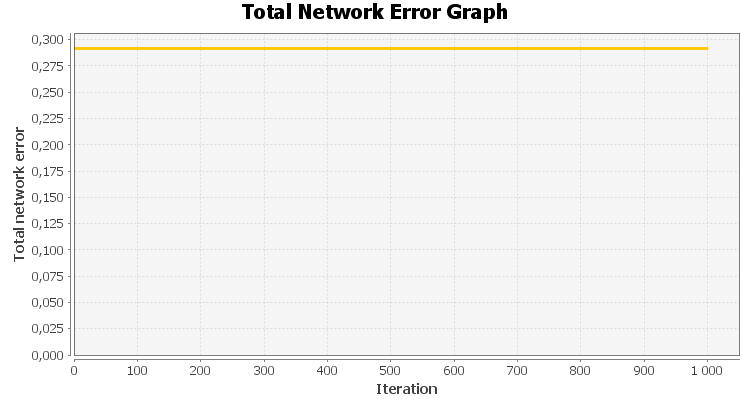
3.learning rate:0,02; liczba danych uczących i testujących: 100. 

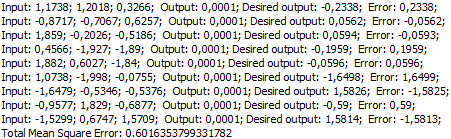
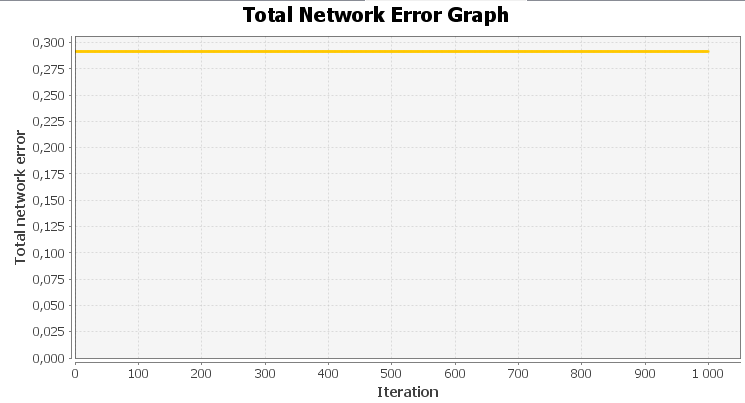
4.learning rate:0,2; liczba danych uczących i testujących: 1000. 

5.learning rate:0,5; liczba danych uczących i testujących: 1000. 

6.learning rate:0,02; liczba danych uczących i testujących: 1000. 

7.learning rate:0,02; liczba danych uczących i testujących: 10000. 

8.learning rate:0,02; liczba danych uczących i testujących: 10000. 

9.learning rate:0,02; liczba danych uczących i testujących: 10000. 

Analiza i wnioski:

Wraz ze wzrostem liczby danych uczących średni błąd podczas testów zmniejsza się. Można zauważyć znaczne obniżenie się wartości błędy podczas testowania, po zwiększeniu liczby danych uczących (i testujących) ze 100 do 1000.  
Najmniejsza sieć osiągnęła nieco lepsze wyniki w trakcie testowania po nauczaniu największą ilością danych niż większe sieci, natomiast sieć z największą liczbą warstw miała najlepsze wyniki dla najmniejszej ilości danych uczących. Jednakże większe różnice nie wystąpiły.  
Wartość learning rate także wpływała zaskakująco słabo na efektywność działania nauczonej sieci.  
Tak jak w przypadku zwykłego Perceptronu wraz ze wzrostem ilości iteracji efektywność działania sieci wzrastała.  
Im więcej danych uczących tym dłużej trwała 1 iteracja uczenia. Przy 10 000 trzeba było chwilę odczekać aż program zakończy proces uczenia. Jeśli użyte zostałyby większe ilości danych niż 100 000 to czas byłby nużąco długi.