Jakub Sowiński

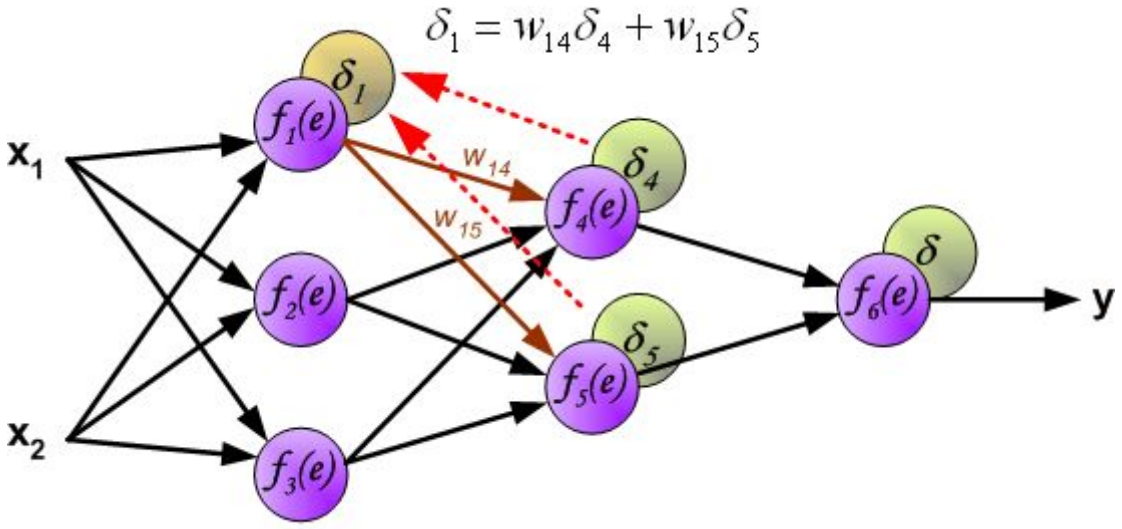
Temat ćwiczenia: Budowa i działanie sieci wielowarstwowej.

Cel ćwiczenia:

Celem ćwiczenia jest poznanie budowy i działania wielowarstwowych sieci neuronowych poprzez uczenie z użyciem algorytmu wstecznej propagacji błędu rozpoznawania konkretnych liter alfabetu.

Zadania do wykonania:

a) Wygenerowanie danych uczących i testujących, zawierających 20 dużych liter dowolnie wybranego alfabetu w postaci dwuwymiarowej tablicy np. 5x7 pikseli dla jednej litery. przykładowe znaki: http://www.ai.c-labtech.net/sn/litery.html
b) Przygotowanie (implementacja lub wykorzystanie gotowych narzędzi) wielowarstwowej sieci oraz algorytmu wstecznej propagacji błędu.
c) Uczenie sieci dla różnych współczynników uczenia i bezwładności.
d) Testowanie sieci.



Propagacja wsteczna – oparty jest on na minimalizacji sumy kwadratów błędów (lub innej funkcji błędu) uczenia z wykorzystaniem optymalizacyjnej metody największego spadku. Dzięki zastosowaniu specyficznego sposobu propagowania błędów uczenia sieci powstałych na jej wyjściu, tj. przesyłania ich od warstwy wyjściowej do wejściowej, algorytm propagacji wstecznej stał się jednym z najskuteczniejszych algorytmów uczenia sieci.

1. **Korzystam z metody trainrp i traingdx**

Metoda Rprop

Algorytm został zaproponowany przez Riedmillera, Brauna w 1992, nazwa pochodzi od nazwy angielskiej Resilent backPROPagation. Istota metody polega na uwzględnianiu jedynie znaku gradientu bez uwzględniania jego wartości - wówczas wartości wag w następnym kroku można wyznaczyć zgodnie z zależnością:

$$w_{ji}(t) = w_{ji}(t-1) - \eta(t-1) sgn\left(\frac{\delta E(t-1)}{\delta w_{ji}}\right)$$

W metodzie tej współczynnik uczenia η zależy od zmian znaku gradientu. Jeżeli w kolejnych iteracjach znak pochodnej jest taki sam następuje wzrost współczynnika, jeżeli natomiast znak się zmienia następuje jego redukcja.

$$\eta_{ji}(t) = \begin{cases} min\left(a * \eta_{ji}(t-1), \eta_{max}\right) & gdy \quad \left(\frac{\delta E(t)}{\delta w_{ji}}\right)\left(\frac{\delta E(t-1)}{\delta w_{ji}}\right) > 0 \\ max\left(b * \eta_{ji}(t-1), \eta_{min}\right) & gdy \quad \left(\frac{\delta E(t)}{\delta w_{ji}}\right)\left(\frac{\delta E(t-1)}{\delta w_{ji}}\right) < 0 \\ \eta_{ji}(t-1) & gdy \quad \left(\frac{\delta E(t)}{\delta w_{ji}}\right)\left(\frac{\delta E(t-1)}{\delta w_{ji}}\right) = 0 \end{cases}$$

gdzie: E - błąd

$$E_l = \frac{1}{2}\sum_{k=1}^{K}(d_{lk} - z_{lk})^2 \rightarrow \min$$

a - współczynnik zwiększania (a ≅1,2),

b – współczynnik zmniejszania (b ≅0,5),

ηmin – dolne ograniczenie współczynnika uczenia,

ηmax – górne ograniczenie współczynnika uczenia.

Zastosowanie wartości minimalnej i maksymalnej współczynnika uczenia zapobiega zbytniemu ograniczeniu oraz nadmiernemu wzrostowi zmian wartości wag. Algorytm Rprop powoduje przyspieszenie procesu nauczania zwłaszcza w obszarach o niewielkim nachyleniu funkcji błędu, gdzie wartość pochodnej ma bardzo małą wartość.

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).

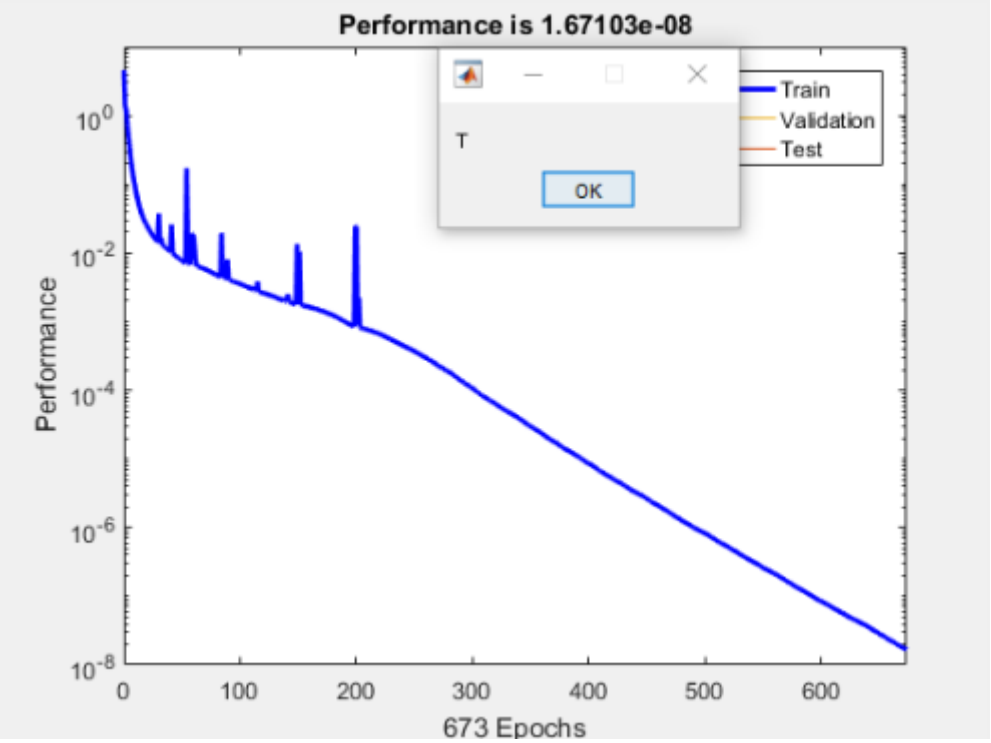
Wywołanie funkcji:
NET = NEWFF(PR, [S1 S2...SNL],{TF1 TF2...TFNL}, BTF, BLF, PF)
WEJŚCIE:
PR - macierz o wymiarach Rx2, gdzie R jest liczbą wejść sieci (współrzędnych wektorów wejściowych); pierwsza kolumna zawiera minimalne wartości kolejnych współrzędnych wektorów wejściowych, druga kolumna – maksymalne 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 moŜe oznaczać dowolną róŜniczkowalną 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.
Uwaga: zestaw funkcji treningu sieci jednokierunkowych (tj. zbiór dopuszczalnych wartości zmiennej BTF) obejmuje m.in.:
traingda metoda propagacji wstecznej błędu z adaptacyjną zmianą stałej szybkości uczenia,
traingdm metoda propagacji wstecznej błędu z momentum,
traingdx metoda propagacji wstecznej błędu z momentum i adaptacyjną zmianą stałej szybkości uczenia,
trainlm metoda Levenberga-Marquardta,
trainscg metoda skalowanego gradientu sprzęŜonego.

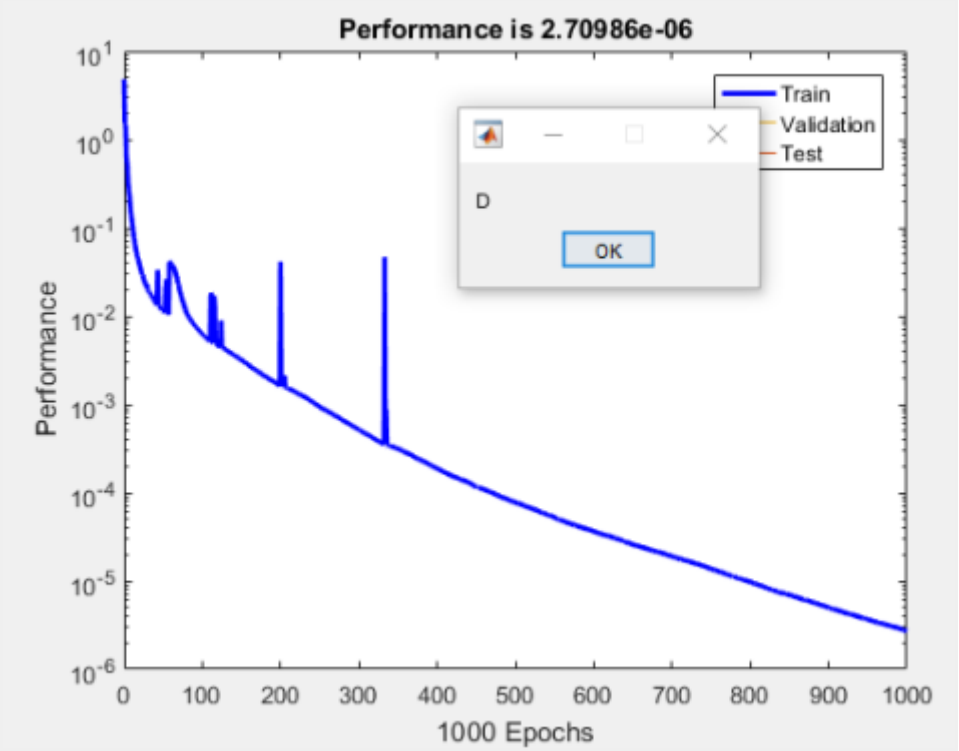
Uczenie z momemtum- Istotą metody jest wprowadzenie do procesu uaktualniania wagi pewnej bezwładności tzw. "momentu", proporcjonalnego do zmiany tej wagi w poprzedniej iteracji.

2. Trainrp

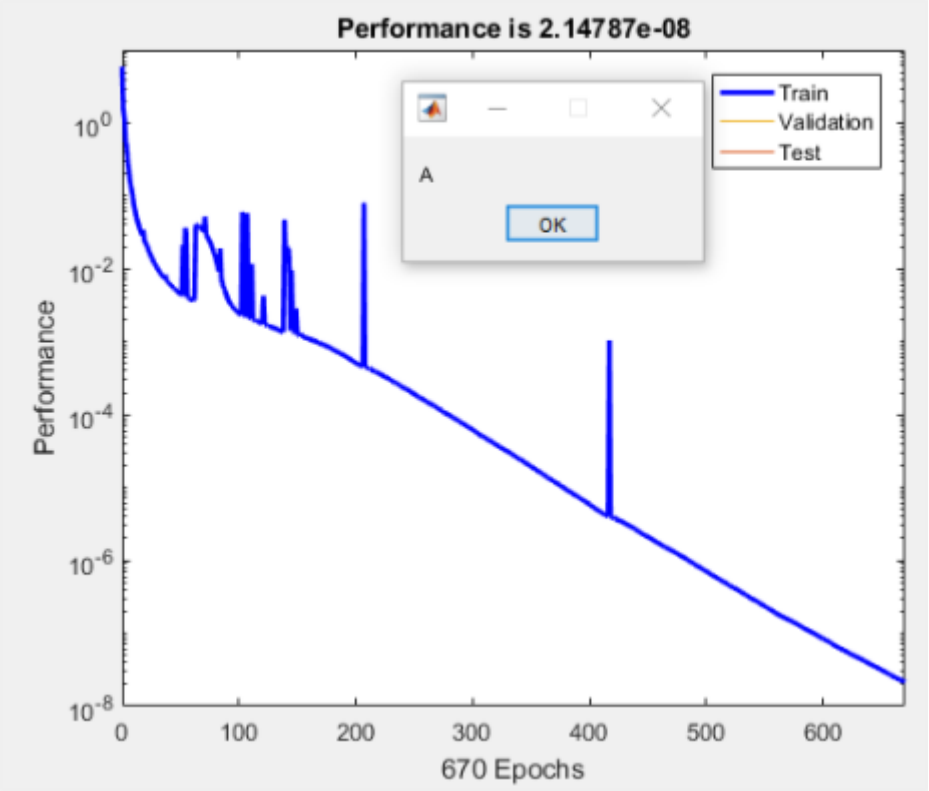
Wprowadzona litera: T, zwrócona przez sieć: T, sieć nauczyła się w 637 epok.



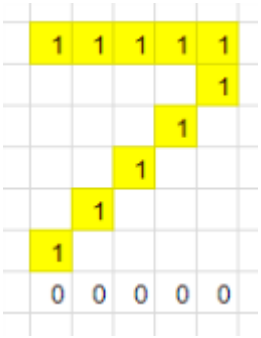
Wprowadzona litera: D, zwrócona przez sieć: D, sieć osiągnęła limit 1000 epok.

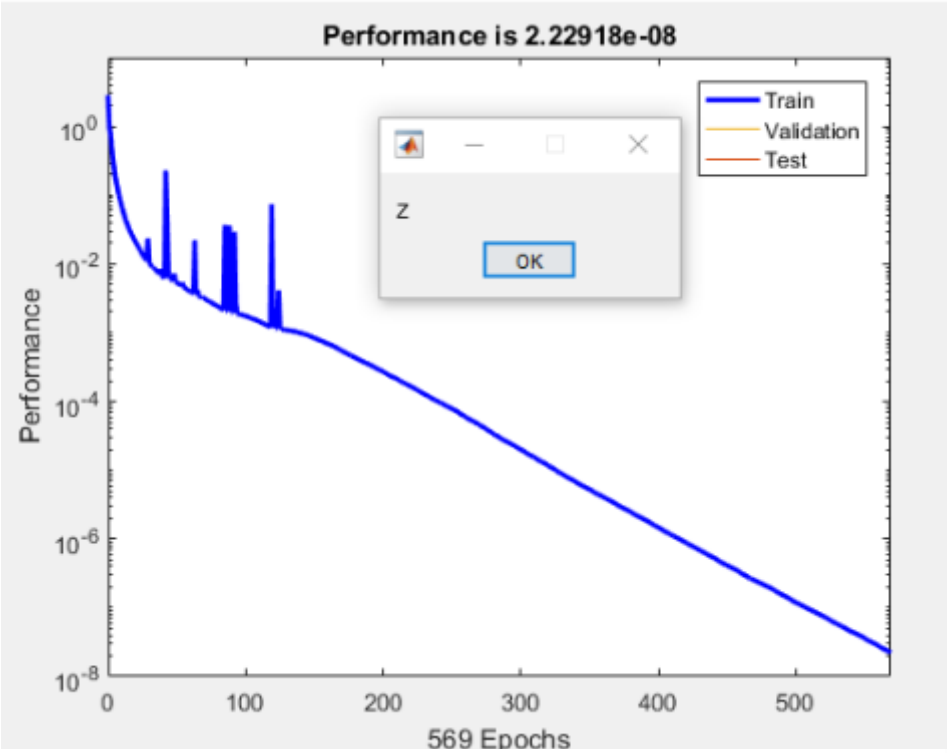


Wprowadzona litera: A, zwrócona przez sieć: A, sieć nauczyła sie w 670 epok.



Wprowadzona litera: Z bez dolnej nóżki, zwrócona litera: Z.





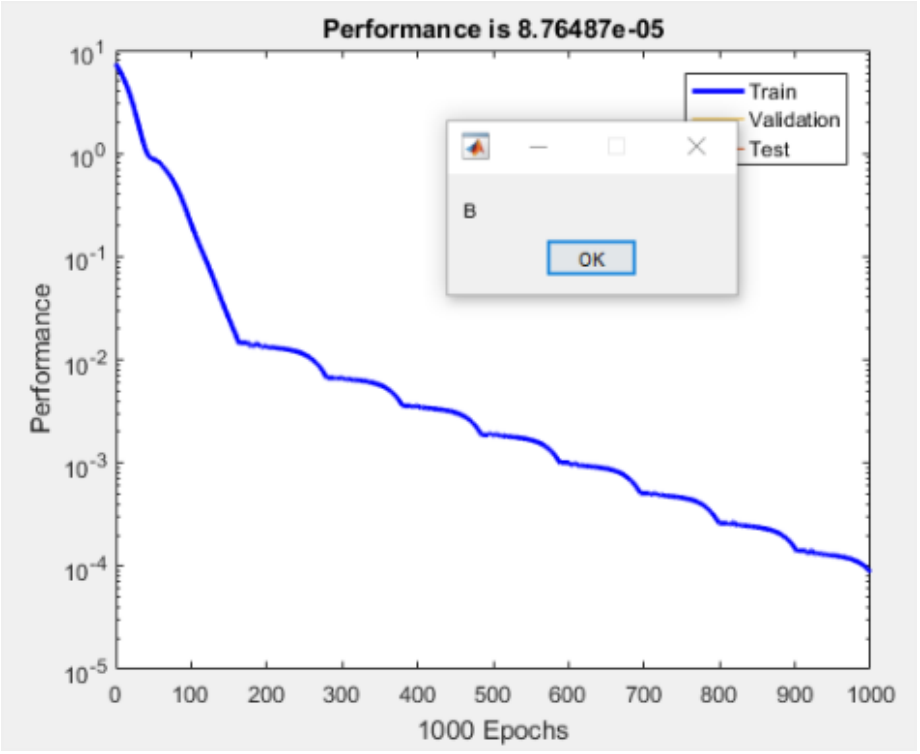
U

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	T	Z
0	0	0,40926	0	0	0,65278	0,49846	0	0	0	0,45217	0,58604	0	0,11834	0	0,61021	0,25518	0,11629	0,68644	0,80043

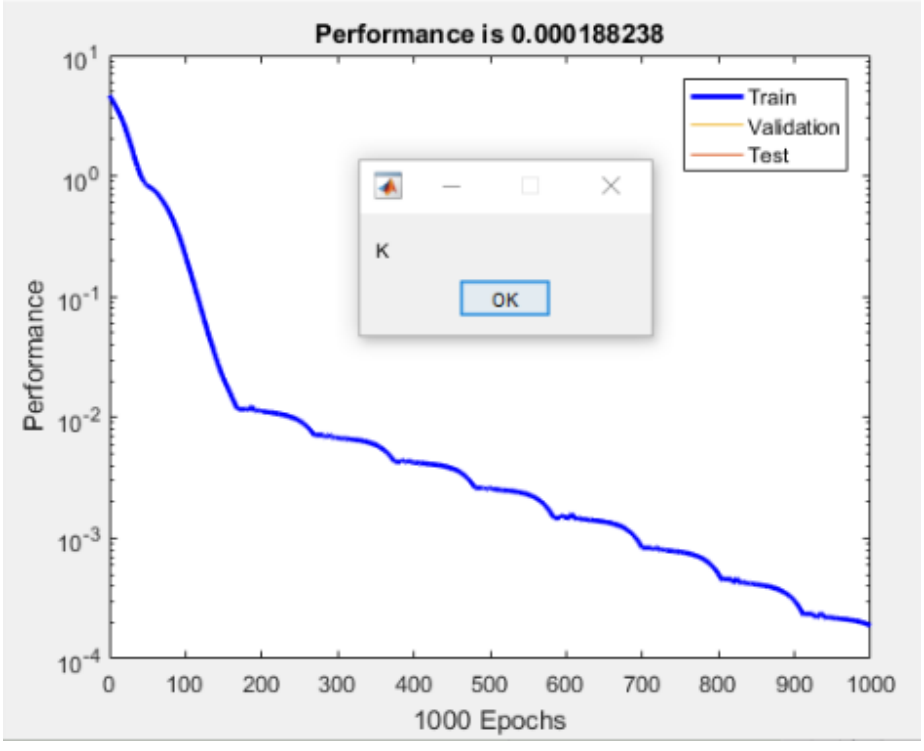
3. Traingdx

traingdx - metoda propagacji wstecznej błędu z momentum i adaptacją współczynnika

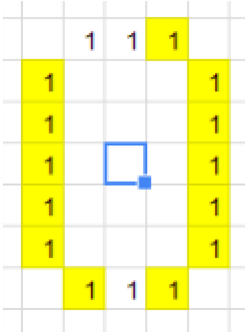
Wprowadzona litera: B, zwrócona przez sieć: B, sieć osiągnęła limit 1000 epok.

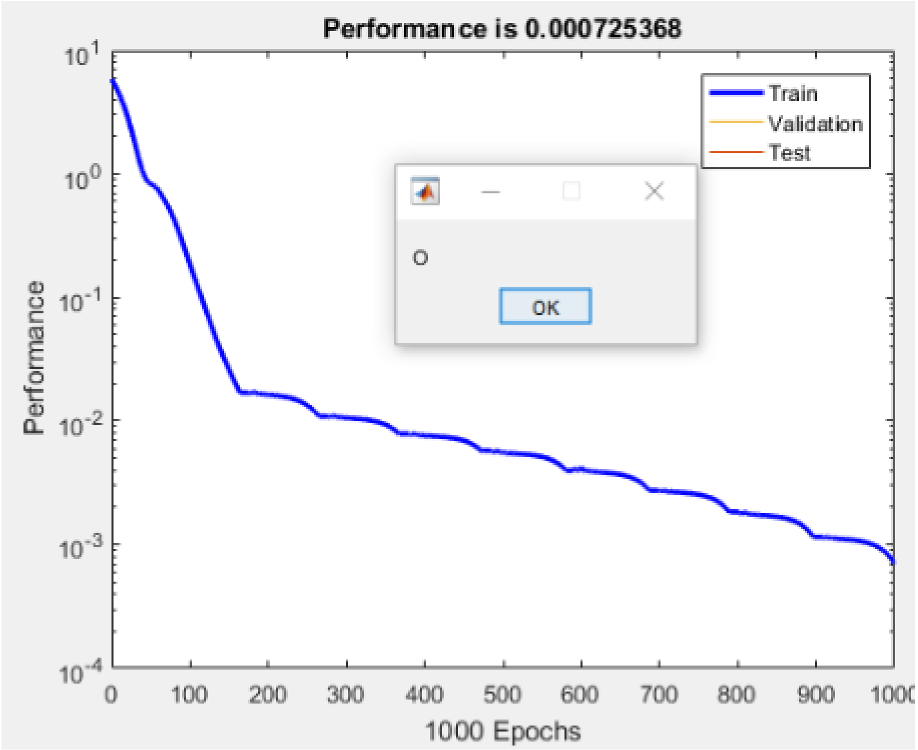


Wprowadzona litera: K, zwrócona przez sieć: K, sieć osiągnęła limit 1000 epok.



Wprowadzona litera: O, zwrócona przez sieć: O, sieć osiągnęła limit 1000 epok.





A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	T	Z
0	0	0,59519	0	0	0,76485	0	0	0	0	0	0	0	0,59026	0,94107	0	0	0,10816	0	0,36000

Listing kodu:

inputs=[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];

targets=[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];

net=newff(inputs, [35 20] , {'logsig', 'purelin', 'traingdx'}); **tworzymy nową sieć wielowarstwową**

litery=[A;B;C;D;E;F;G;H;I;J;K;L;M;N;O;P;R;S;T;Z]; **macierz liter**

litery=litery';

wy=zeros(20); **macierz jedynek na odpowiedniej pozycji dla każdej litery**

for i=1:20

  wy(i,i)=1;

end

wy=wy';

```
Y=sim(net,litery);
net.trainParam.max_fail=8;
net.trainParam.goal = 0.0001;
net.trainParam.mu=0.001;
 net.trainParam.lr=0.1;
 net.trainParam.epochs=2000;

 [net tr1] =traingdx(net,litery,wy) trenujemy sieć traingdx////albo trainrp
 plotperf(tr1);
 Y=sim(net,litery);
```

4. Spostrzeżenia i wnioski

Proces uczenia odbywał się dla trainrp oraz traingdx. Można było zauważyć że trainrp wykonuje się w dużej liczbie epok(500-1000) ale zawsze nauczył się w 100% danej literki natomiast podczas stosowania traingdx z propagacja wsteczną prawie wszędzie występowały małe wartości dla każdej litery.
Testy przeprowadziłem dla 1 sieci. Ta sieć nie posiadała warstwy ukrytej (35 neuronów - 20 neuronów. Obserwując procesy uczenia można było stwierdzić że dużo lepiej działała trainrp, gdyż sieć nauczyła się w mniej niż 1000 epokach oraz dawała znakomite wyniki natomiast traingdx potrzebował ok 2800 iteracji do pełnej nauki oraz nie dostawaliśmy dobrego przybliżenia oraz można było zaobserwować przeuczenie się sieci.
Zmieniając wskaźnik uczenia się (learning rate) otrzymywaliśmy lepsze przybliżenie wyników ale w efekcie sieć musiała się uczyć dłużej.

Funkcja która trenuje nam sieć jest ona w tym momencie kluczowym elementem gdyż może ona nam zmniejszyć liczbę epok lub zwrócić lepszy rezultat. Natomiast przez mniejszy learning rate wzrosła liczba epok lecz wyniki stały się dużo lepsze.

Tworząc tego typu sieci nie można skupić się na jednym czynniku trzeba pamiętać o doborze odpowiednich trenerów, stworzenie odpowiedniej sieci neuronów oraz doborze parametrów lr, gdyż jeśli zbudujemy większą sieć i lr również będzie wysoki to może nam się nie nauczyć albo otrzymamy mało satysfakcjonujący efekt.