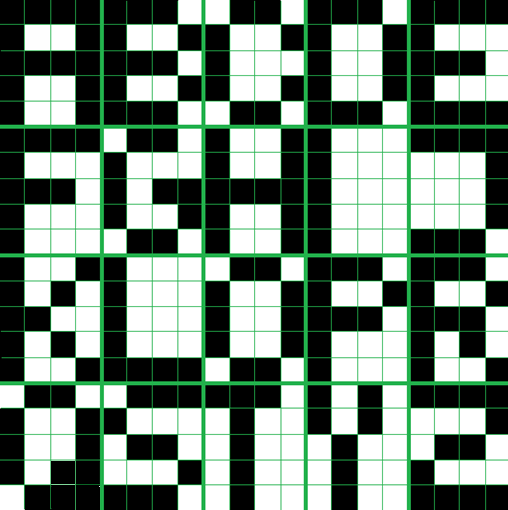
|  |  |  |
| --- | --- | --- |
| Grupa ćwicz. **3** | Data wykonania 17.11.2017 | Nr. Scenariusza  **3** |
| **Temat ćwiczenia:** Budowa i działanie sieci wielowarstwowej. | | |
| Imię i nazwisko  **Andrzej Pawlikowski** | | Ocena i Uwagi |

**Cel ćwiczenia:**

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

Wykonanie zadania:

W celu realizacji projektu przygotowałem dane uczące składające się z dwudziestu dużych liter alfabetu łacińskiego wygenerowanych w tablicy dwuwymiarowej 4x5.



Macierz reprezentującą dane uczące umieściłem w osobnym pliku. Plik ten składa się z 20 kolumn reprezentujących każdą literę oraz 20 wierszy wypełnionych odpowiednio zerami albo jedynkami zależnie od litery.

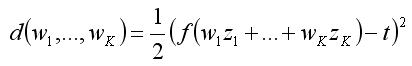
Algorytm wstecznej propagacji jest to metoda obliczeniowa umożliwiająca wytrenowanie optymalnych zestawów wag.

Ogólny schemat procesu trenowania sieci wygląda następująco:

1. Ustalamy topologię sieci, tzn. liczbę warstw, liczbę neuronów w warstwach.
2. Inicjujemy wagi losowo (na małe wartości).
3. Dla danego wektora uczącego obliczamy odpowiedź sieci (warstwa po warstwie).
4. Każdy neuron wyjściowy oblicza swój błąd, oparty na różnicy pomiędzy obliczoną odpowiedzią *y* oraz poprawną odpowiedzią *t*.
5. Błędy propagowane są do wcześniejszych warstw.
6. Każdy neuron (również w warstwach ukrytych) modyfikuje wagi na podstawie wartości błędu i wielkosci przetwarzanych w tym kroku sygnałów.
7. Powtarzamy od punktu 3. dla kolejnych wektorów uczących. Gdy wszystkie wektory zostaną użyte, losowo zmieniamy ich kolejność i zaczynamy wykorzystywać powtórnie.
8. Zatrzymujemy się, gdy średni błąd na danych treningowych przestanie maleć. Możemy też co jakiś czas testować sieć na specjalnej puli nieużywanych do treningu próbek testowych i kończyć trenowanie, gdy błąd przestanie maleć.

Aby znależć taki zestaw wag, dla którego błąd sieci jest jak najmniejszy, możemy zapisać ten błąd jako funkcję od wartości wag. Oznaczmy przez:  
*f: R -> R* – funkcję aktywacji w neuronie  
*w1 ,..., wK* – wagi połączeń wchodzących  
*z1 ,..., zK* – sygnały napływające do neuronu z poprzedniej warstwy.

Zwykle błąd liczony jest jako kwadrat odchylenia: *d = 1/2 (y-t)2*, co możemy rozpisać jako:



**Opis funkcji użytej w ćwiczeniu:**

net = newff(WE,WY,S,T,BTF);

gdzie:

•net–struktura opisująca sieć

•WE –wejścia sieci(uwaga w poziomie cechy, w pionie wektory)

•WY–wyjścia sieci (etykiety wektorów wej.) każdy wiersz opisuje pojedyncze wyjście

•S–współczynnik określający ilość neuronów w poszczególnych warstwach oprócz ostatniej (ostatnia warstwa jest zdeterminowana przez liczbę wyjść)

•T–tablica komórkowa zawierająca informację w postaci stringów opisujących kształt funkcji aktywacji dla poszczególnych warstw. Możliwe funkcje to:

a.tansig –sigmoidalna funkcja aktywacji (+1,-1)

b.satlins-liniowa funkcja aktywacji z ograniczeniami

c.logsig -logarytmiczno sigmoidalna sunkcja aktywacji (0, +1)

d.hardlim –binarna funkcja aktywacji typu y=1(x)



•BTF –algorytm uczenia sieci. Możliwe opcje to:

-traingd-największego spadku z propagacją wsteczną

-traingda -największego spadku z adaptacją wsp. uczenia i propagacją wsteczną

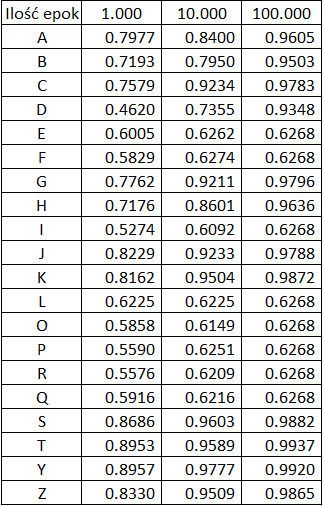
-traingdm -Największego spadku z funkcją momentum z propagacją wsteczną

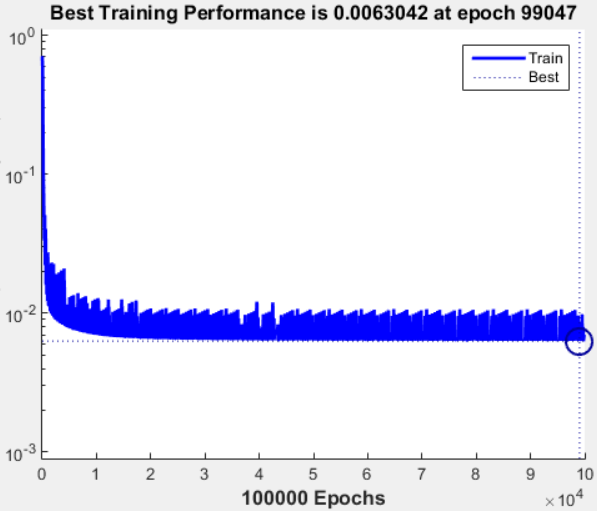
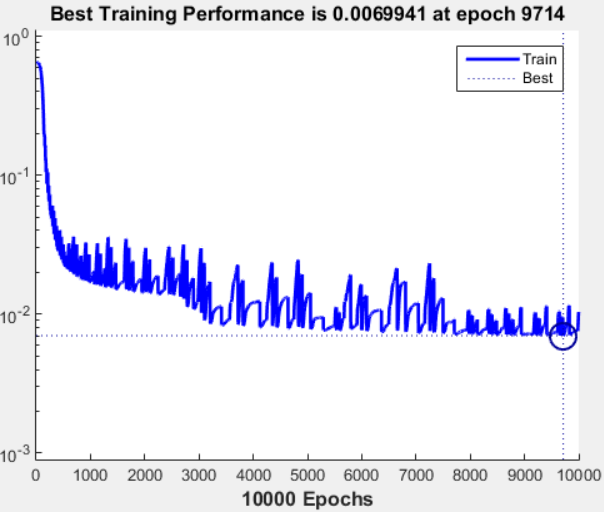
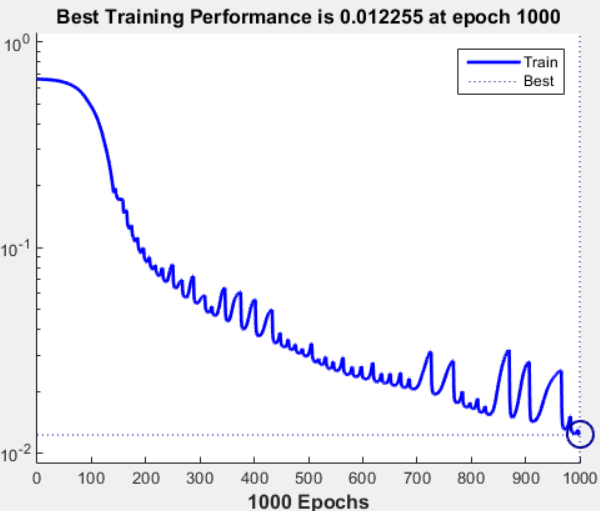
-traingdx -Największego spadku z momentum i adaptacyjnym wsp. uczenia i propagacją wsteczną

-trainlm -algorytm Levenberg-Marquard

Wyniki:

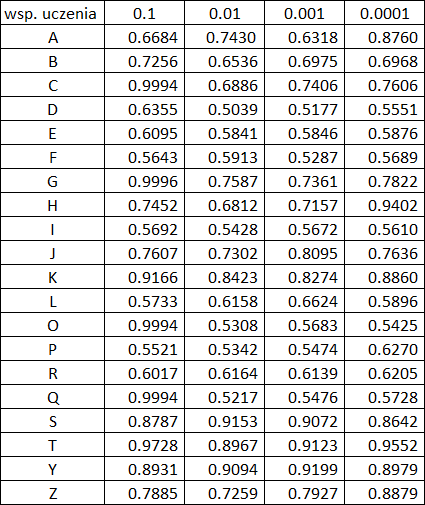
Testowanie algorytmu w zależności od ilości epok:





Wpływ dobranego współczynnika uczenia na wyniki

Manewrując współczynnikiem uczenia nie dało udało się zauważyć żadnej zależności.



**Wnioski:**

Tabela z wynikami pokazuje, że sieć bardzo dobrze radziła sobie z rozpoznawaniem liter, które pojawiały się w danych uczących. Nigdy nie zdarzyło się by dla identycznego kształtu znaku jak w danych uczących sieć nie była w stanie poprawnie rozpoznać litery lub by wystąpił przypadek, gdy kilka znaków ma bardzo wysokie wartości podobieństwa. Zwiększając współczynnik uczenia wpływaliśmy na dokładniejszy proces nauki.

Metoda wstecznej propagacji błędu była pierwszą metod pozwalającą na uczenie sieci wielowarstwowych. W przypadku sieci liczebnie rozbudowanych metoda ta jest stosunkowo wolna. Zobrazowane to zostało w tabeli przedstawiającej efektywność programu w stosunku do maksymalnych ilości epok. Trening dla 10000 epok trwał ok 7 sekund, natomiast dla 100000 epok przy zachowaniu tych samych parametrów treningu czas wynosił aż 5 minut co ważne wartości błędu średniokwadratowego nie zmalała znacząco w porównaniu do treningu przy 10000 epok. Wykres przedstawiający przebieg tego treningu przedstawia widoczny brak efektywności w stosunku do czasu.

close all; clear all; clc;

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;];

%wejscia do sieci i min/max wartosci wejsc

S=20; %ilosc wyjsc z sieci

net = newff(PR,S,{'tansig'},'traingda');

%Wczytywanie danych z pliku

filename = 'alfabet.txt';

delimiterIn = ' ';

WE = importdata(filename,delimiterIn);

WY=eye(20);

%parametry treningu

net.trainParam.epochs = 10000; %max ilosc epok

net.trainParam.lr = 0.00001; %wspolczynnik uczenia

net = train(net, WE, WY);%uczenie

for i=1:20

test=WE(:,i);

efekt=sim(net, test);%testowanie sieci

%wypisywanie jaka to litera

max=1;

for j=1:20

if (efekt(max)<efekt(j))

max=j;

end;

end;

switch max

case 1

disp('A')

case 2

disp('B')

case 3

disp('C')

case 4

disp('D')

case 5

disp('E')

case 6

disp('F')

case 7

disp('G')

case 8

disp('H')

case 9

disp('I')

case 10

disp('J')

case 11

disp('K')

case 12

disp('L')

case 13

disp('O')

case 14

disp('P')

case 15

disp('R')

case 16

disp('Q')

case 17

disp('S')

case 18

disp('T')

case 19

disp('Y')

case 20

disp('Z')

otherwise

disp('Blad!!!')

end

disp(efekt(max))

end