|  |  |  |
| --- | --- | --- |
| Grupa ćwicz. **2** | Data wykonania 31.10.2017 | Nr. Scenariusza  **2** |
| **Temat ćwiczenia:** Budowa i działanie sieci neuronowej | | |
| Imię i nazwisko  **Andrzej Pawlikowski** | | Ocena i Uwagi |

**Wstęp teoretyczny:**

**Sieci jednokierunkowe**

Sieci jednokierunkowe to sieci neuronowe, w których nie występuje [sprzężenie zwrotne](https://pl.wikipedia.org/wiki/Sprz%C4%99%C5%BCenie_zwrotne), czyli pojedynczy wzorzec lub sygnał przechodzi przez każdy neuron dokładnie raz w swoim cyklu.

**Algorytm Levenberga-Marquardta**

Działanie algorytmu Levenberga-Marquardta opiera się na założeniu że modelowana przez sztuczną sieć neuronową rzeczywista funkcja od wzorująca sygnał wejściowy w sygnał wyjściowy ma charakter liniowy.

Algorytm Levenberga-Marquardta umiejętnie łączy dwie metody: metodę największego spadku oraz algorytm hipotetycznej aproksymacji liniowej (metoda Newtona). W trakcie swojej pracy algorytm wybiera metodę dającą

w danej chwili lepsze rezultaty 2 próbuje hipotetycznej aproksymacji liniowej i jeżeli

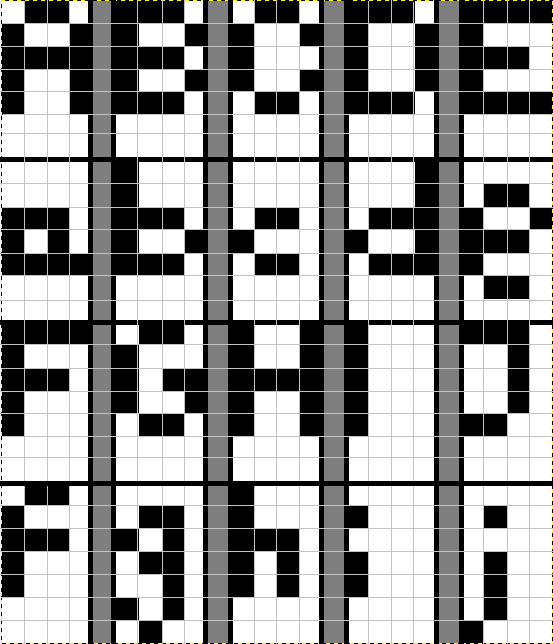
nie daje ona pozytywnych wyników cofa się do metody największego spadku, co umożliwia mu bardzo szybkie działanie.

**Cel ćwiczenia:**

Celem ćwiczenia jest poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

**Wykonanie zadania:**

W celu realizacji projektu przygotowałem dane uczące składające się z dziesięciu dużych i dziesięciu małych liter alfabetu łacińskiego wygenerowanych w tablicy dwuwymiarowej 4x7.



Macierz reprezentującą dane uczące umieściłem w osobnym pliku. Plik ten składa się z 20 kolumn reprezentujących każdą literę oraz 28 wierszy wypełnionych odpowiednio zerami albo jedynkami.

W ramach ćwiczenia użyłem dwóch funkcji newlin oraz newff

**NEWLIN** - Funkcja tworzy jednowarstwową sieć neuronową, złożoną z zadanej liczby neuronów o liniowych funkcjach aktywacji. Tego typu sieć jest zwykle wykorzystywana jako filtr adaptacyjny do przetwarzania sygnałów lub predykcji szeregów czasowych.

**NEWFF –** tworzy kaskadową jednokierunkową sieć wielowarstwową uczoną metodą propagacji wstecznej błędu (wykorzystującej metodę Levenberga-Marquardta).

Parametry potrzebne do utworzenia sieci za pomocą tych funkcji:

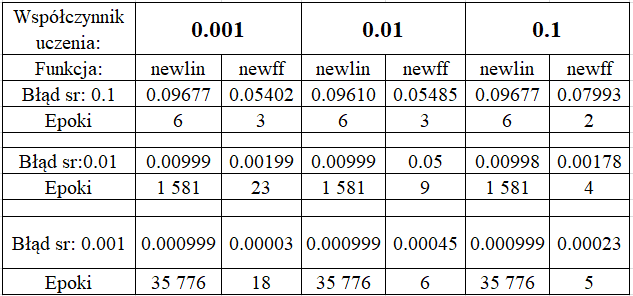
PR – zmienna wejściowa dla funkcji tworzących sieć neuronową, składa się z par 0 1

S - zmienna przechowująca ilość wyjść, w danym ćwiczeniu wystarczy ustawić ją na 1 ponieważ mamy sprawdzić tylko wielkość litery.

Przechodząc do uczenia naszej sieci pierwsze ustawiłem parametry treningu a następnie użyłem funkcji train której przekazujemy dane uczące (tablica z liter przekazana z osobnego pliku) oraz dane wyjściowe. Ustawiamy w nich odpowiednio 0 i 1 dla małych i dużych liter.

WY=[1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0];

Manipulując współczynnikiem uczenia oraz błędem średniokwadratowym uzyskałem poniższe wyniki.



**Analiza otrzymanych danych:**

Uczenie sieci neuronowej polega na minimalizacji funkcji błędu. Od razu można zauważyć ze w funkcji newlin nie można zmieniać współczynnika uczenia. Jest on ustawiony odpowiednio na 0.1.

Dla błędu średnio kwadratowego równego 0.01 ilość epok wynosiła 1581, natomiast dla błędu wynoszącego 0.1 zmniejszyła się do 6 ale metoda zaczęła dawać zdecydowanie mniej dokładne wyniki. Zmniejszanie wartości błędu średnio kwadratowego znacznie zwiększało zarówno ilość epok jak i dokładność metody.

Wartość współczynnika błędu miała również wpływ na newff ale w znacznie mniejszym stopniu. W tym przypadku również widać zależności od zmiany współczynnika uczenia.

Funkcja newff dawała zdecydowanie lepsze efekty. Metoda ta nie potrzebowała wielu epok mimo zmniejszania progu błędu średniokwadratowego. Świadczy to o dobrej optymalizacji metody. Metoda ta wykorzystuje algorytm Levenberga-Marquardta (szybka zbieżność).

Oba algorytmy nie popełniały błędów jeśli otrzymały pełny zestaw uczący, poprawnie rozróżniając duże litery od małych.

Wnioski:

Algorytm wykorzystujący metodę Levenberga-Marquardta okazał się lepszy praktycznie pod każdym względem. Wyniki uzyskiwane za pomocą tej metody były dokładniejsze przy mniejszym zapotrzebowaniu na czas. Problemy pojawiały się gdy wprowadzono do programu literę nieuwzględnioną w danych uczących – algorytmy porównywały ją do wcześniej otrzymanych danych co czasami skutkowało rozpoznaniem małej litery jako dużej lub odwrotnie.

W przypadku, gdy chce się szybko nauczyć sieć powinno się dać w miarę wysoki współczynnik uczenia ale należy liczyć się z tym, że algorytm będzie w mniej dokładny sposób rozróżniał duże litery od małych co w niektórych przypadkach może się przełożyć na niepoprawne wyniki.

W celu poprawienia skuteczności działania programu można wykorzystać macierze o większych rozmiarach co skutkowałoby spowolnieniem programu jednak zmniejszyło by prawdopodobieństwo pomyłki spowodowanych przez większą ilość porównywanych punktów.

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

S=1;

net = newlin(PR,S); %metoda 1

%net = newff(PR,S,{'tansig' 'purelin'}); %metoda 2

%Wczytywanie danych z pliku

filename = 'alfabet.txt';

delimiterIn = ' ';

WE = importdata(filename,delimiterIn);

WY=[1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0];

net.trainParam.epochs = 10000000;

net.trainParam.goal = 0.1;

net.trainParam.lr = 0.1;

net = train(net, WE, WY);

for i=1:20

%zmienna test jest wektorem przedstawiacym litere:

%A a B b C c D d E e F f G g H h I i J j

test=WE(:,i);

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

if round(efekt) == 1

disp('Duza litera');

else

disp('Mala litera');

end

end

Dane uczące:

0 0 1 1 0 0 1 0 1 0 1 0 0 0 1 1 1 0 1 0

1 0 1 0 1 0 1 0 1 0 1 1 1 0 0 0 0 0 1 0

1 0 1 0 1 0 1 0 1 0 1 1 1 0 0 0 0 0 1 0

0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0

1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1

0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0

1 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0

1 1 1 1 1 0 1 0 0 0 1 1 1 1 1 1 1 0 0 0

1 1 1 1 0 1 0 1 1 0 1 1 0 0 1 1 0 0 0 0

1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 0 1 0

1 0 0 0 0 0 1 1 1 1 0 0 1 0 1 0 0 0 0 0

1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 0

0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 1

0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0

1 0 1 1 1 0 1 1 0 1 0 0 1 0 1 0 0 0 0 0

1 1 1 1 0 0 1 0 0 0 1 1 0 0 1 1 1 1 1 0

0 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 1 1

0 1 1 1 1 1 1 1 1 0 0 0 1 1 0 1 0 0 0 0

1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 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 1 0 0 0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0