

Podstawy Sztucznej Inteligencji Scenariusz 3:

Budowa i działanie sieci wielowarstwowej.

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

Dane uczące są wygenerowane w postaci dwuwymiarowej tablicy 4x5 pikseli dla jednoitej litery. Zbiór danych uczących to 20 dużych liter, reprezentowanych jako rząd zer i jedynek.

Dla przykładu duża litera A:

```
0 1 1 0
1 0 0 1
1 1 1 1
1 0 0 1
1 0 0 1
```

Taki ciąg znaków zostaje wpisany jako kolumna w wartościach uczących :

$A = [0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1]$;

Zaimplementowana sieć posiada 20 wyjść, po jednym na każdą literę. Wyjścia również przyjmują wartości 0 lub 1 obrazując która kolumna odpowiada danej literze w taki sposób, że 1 oznacza literę której odpowiada dany ciąg znaków - litery są zapisane za kolejnością alfabetyczną, w efekcie daje mi to dane wyjściowe w postaci macierzy 20x20 wypełnioną jedynkami na głównej przekątnej i zerami w pozostałych miejscach:

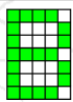
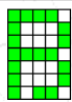
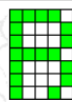

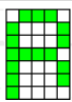
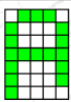
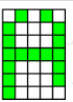
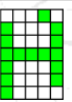
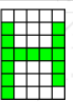
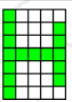
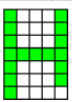
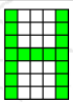
```
%A B C D E F G H I J K L N O P R S T U Y
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; %A
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; %B
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; %C
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; %D
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; %E
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; %F
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; %G
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0; %H
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0; %I
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0; %J
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0; %K
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0; %L
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0; %N
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0; %O
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0; %P
```

```

00000000000000000010000; %R
00000000000000000001000; %S
00000000000000000000100; %T
00000000000000000000010; %U
00000000000000000000001; %Y
];

```

Na podstawie tych danych algorytm będzie w stanie rozpoznawać litery licząc podobieństwo otrzymanego wzoru i porównując go do każdej z 20 liter które otrzymał w danych wejściowych. Podobieństwo określone będzie wg wartości ułamka - im bliższy wartością, tym kod bardziej przypominał odpowiadającą literę.

												
A	0.020985	0.304754	0.553698	0.982967	0.993287	0.984049	0.969733	0.663561	0.059392	0.036503	0.030628	0.011370
B	0.943860	0.901880	0.249863	0.355721	0.979239	0.471022	0.061320	0.000184	0.000960	0.000015	0.000081	0.000020
C	0.000106	0.000014	0.000011	0.000001	0.000021	0.000030	0.000014	0.000024	0.000097	0.000011	0.000001	0.000004
D	0.000378	0.000031	0.000181	0.000031	0.000003	0.000032	0.000006	0.000035	0.000013	0.000159	0.000038	0.000078
E	0.003512	0.005860	0.001913	0.000090	0.000241	0.000172	0.000150	0.000169	0.000808	0.000312	0.000770	0.001932

Algorytm wstecznej propagacji błędów szuka przybliżonego, optymalnego zestawu wag wykorzystując metodę obliczeniową wstecznej propagacji błędów. Jest to metoda umożliwiająca modyfikację wag w sieci o architekturze wielowarstwowej, we wszystkich jej warstwach. Jak działa:

- ustalenie liczby warstw i liczby neuronów w warstwach
- inicjacja wag w sposób losowy.
- dla danego wektora uczącego obliczana jest odpowiedź sieci, warstwa po warstwie
- każdy neuron wyjściowy oblicza swój błąd, oparty na różnicy pomiędzy obliczoną odpowiedzią y oraz poprawną odpowiedzią t.
- błędy propagowane są do wcześniejszych warstw
- każdy neuron modyfikuje wagi na podstawie wartości błędów i wielkości przetwarzanych w tym kroku sygnałów
- Powrót do punktu 3. i kontynuowanie działania dla kolejnych wektorów uczących
- gdy wszystkie wektory zostaną użyte, ich kolejność jest zamieniana losowo i zaczynają być wykorzystywane ponownie
- algorytm zatrzymuje się, gdy średni błąd na danych treningowych przestanie maleć. Aby znaleźć taki zestaw wag, dla którego błąd sieci jest jak najmniejszy, możemy zapisać ten błąd jako funkcję od wartości wag. Oznaczmy przez:

$f: R \rightarrow R$ - funkcję aktywacji w neuronie

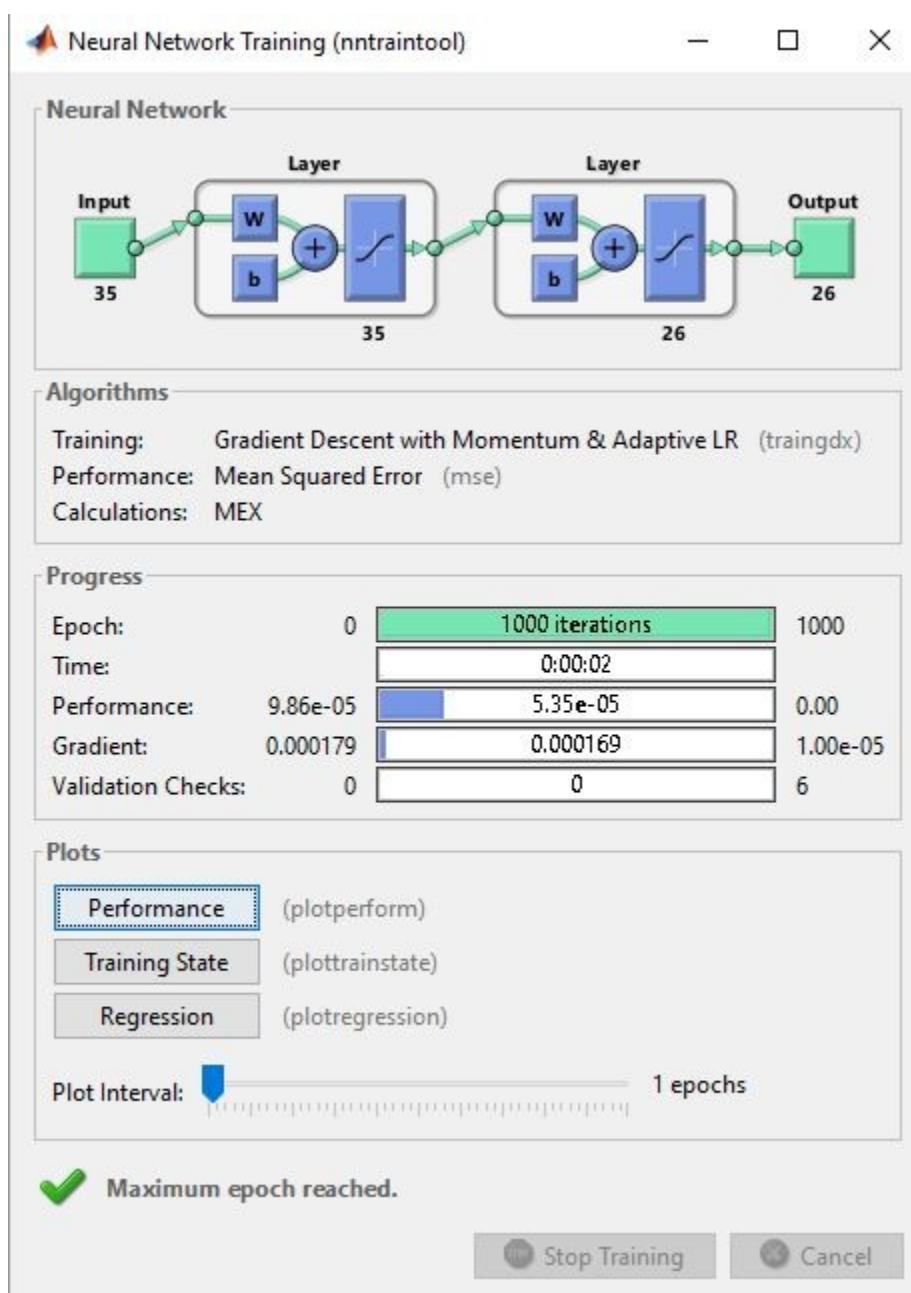
w_1, \dots, w_K - wagi połączeń wchodzących

z_1, \dots, z_K - sygnały napływające do neuronu z poprzedniej warstwy

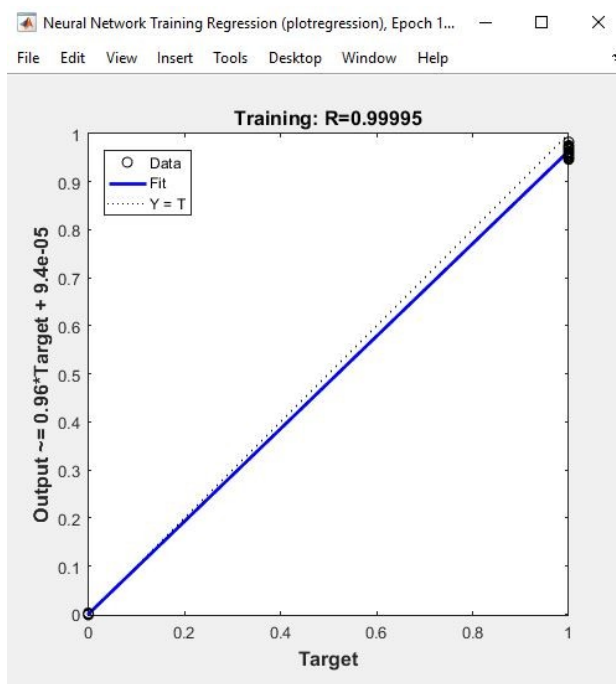
Błąd liczony jest jako kwadrat odchylenia: $d = 1/2 (y-t)^2$, co możemy rozpisać jako:

$$d(w_w, \dots, w_k) = \frac{1}{2} (f(w_1 z_1 + \dots + w_k z_k) - t)^2$$

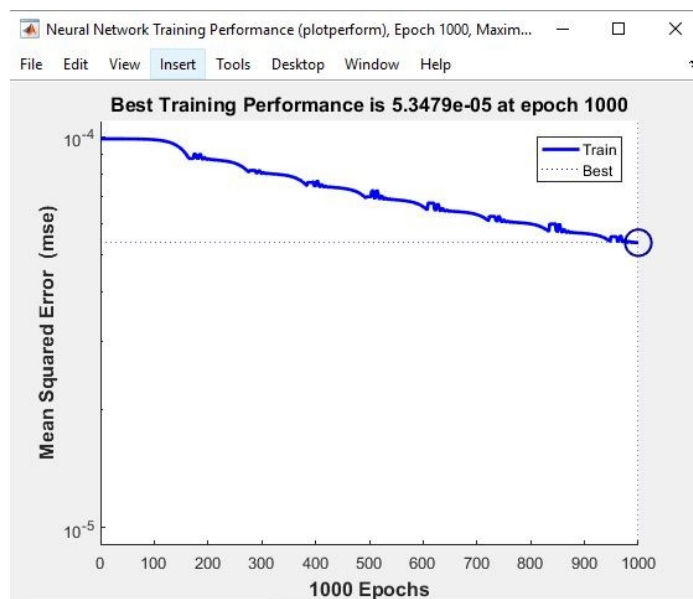
W celu ustalenia, o ile powinna zmienić się waga neuronu, powinno się rozłożyć otrzymany błąd całkowity na połączenia wprowadzające sygnały do danego neuronu. Składową błędu dla każdego j-tego połączenia określamy jako pochodną cząstkową błędu względem j-tej wagi.



Postępy uczenia sieci:



Wydajność uczenia sieci:



Listing kodu programu:

```
close all;clear all;clc
```

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


[illegible]