

## Podstawy sztucznej inteligencji Sprawozdanie do scenariusza 3

### Zadanie:

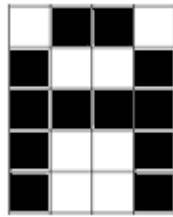
Przygotowanie wielowarstwowej sieci neuronowej rozróżniającej duże litery i wykorzystującej algorytm wstecznej propagacji błędów.

Podczas realizacji scenariusza korzystałem z oprogramowania MATLAB R2016a.

### Dane uczące:

W celu realizacji scenariusza przygotowałem dane uczące składające się z 20 dużych liter: A, B, C, D, E, F, G, H, I, J, K, L, N, O, P, R, S, T, U, Y. Litery są zapisywane na tablicy 4x5 wg zaprezentowanego sposobu:

-literę rysuję w tablicy 4x5:



-zamieniam czarne pola na wartość 1 a białe na wartość 0

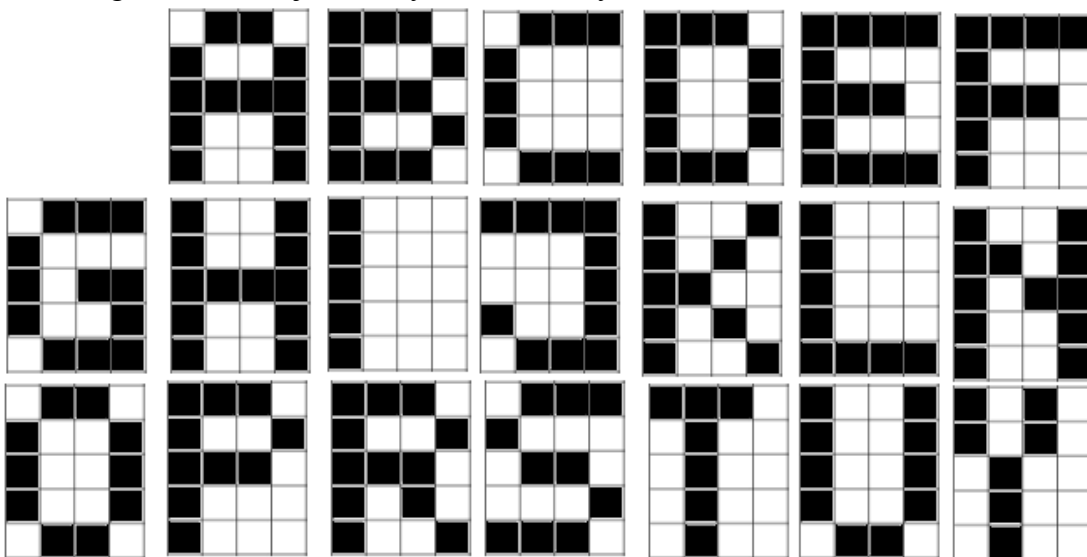
```
0 1 1 0
1 0 0 1
1 1 1 1
1 0 0 1
1 0 0 1
```

-zapisuję wszystkie wiersze po kolei zyskując literę jako ciąg zer i jedynek:

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.

Poniższa grafika obrazuje kształty liter, które wybrałem:



Takie litery skutkują następującymi danymi uczącymi, gdzie każda kolumna odpowiada kodowi innej litery:

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

Zaimplementowana przeze mnie sieć neuronowa będzie posiadała 20 wyjść – po jednym na każdą literę. Wyjście przyjmuje 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 więc dla wartości A kod wyjściowy będzie w postaci:

```
%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; %A
```

Natomiast dla litery B kod wyjściowy przyjmie postać:

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

itd. 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; %A
 0 1 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; %C
 0 0 0 1 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; %E
 0 0 0 0 0 1 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; %G
 0 0 0 0 0 0 0 1 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; %I
 0 0 0 0 0 0 0 0 0 1 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; %K
 0 0 0 0 0 0 0 0 0 0 0 1 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; %N
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 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; %P
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0; %R
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0; %S
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0; %T
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0; %U
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1; %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 on będzie 1 tym otrzymany kod będzie bardziej przypominał odpowiadającą literę.

### **Algorytm uczenia – propagacji wstecznej błędu:**

Algorytm ten szuka przybliżonego, optymalnego zestawu wag wykorzystując metodę obliczeniową propagacji wstecznej błędu. Jest to metoda umożliwiająca modyfikację wag w sieci o architekturze warstwowej, we wszystkich jej warstwach.

Schemat działania algorytmu:

1. Ustalenie liczby warstw i liczby neuronów w warstwach.
2. Inicjacja wag w sposób losowy.
3. Dla danego wektora uczącego obliczana jest 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 modyfikuje wagi na podstawie wartości błędu i wielkości przetwarzanych w tym kroku sygnałów.
7. 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 повторно.
8. 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:

$f: \mathbb{R} \rightarrow \mathbb{R}$  – funkcję aktywacji w neuronie

$w_1, \dots, w_K$  – wagi połączeń wchodzących

$z_1, \dots, z_K$  – sygnały napływające do neuronu z poprzedniej warstwy.

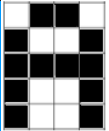
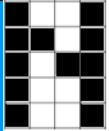

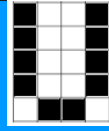
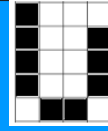
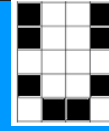

Zwykle błąd liczony jest jako kwadrat odchylenia:  $d = 1/2 (y-t)^2$ , co możemy rozpisać jako:

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

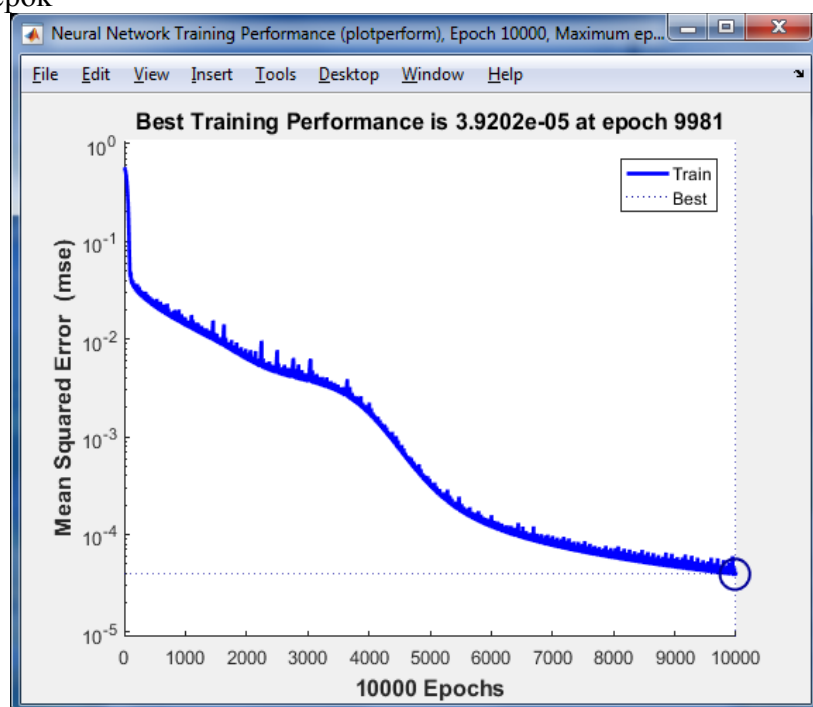
W celu ustalenia, o ile powinna zmienić się waga neuronu, powinno się rozłożyć otrzymany błąd całkowity na połączenia wprowadzające sygnały do danego neuronu. Składową błędu dla każdego  $j$ -tego połączenia określamy jako pochodną cząstkową błędu względem  $j$ -tej wagi.

### **Wyniki działania programu:**

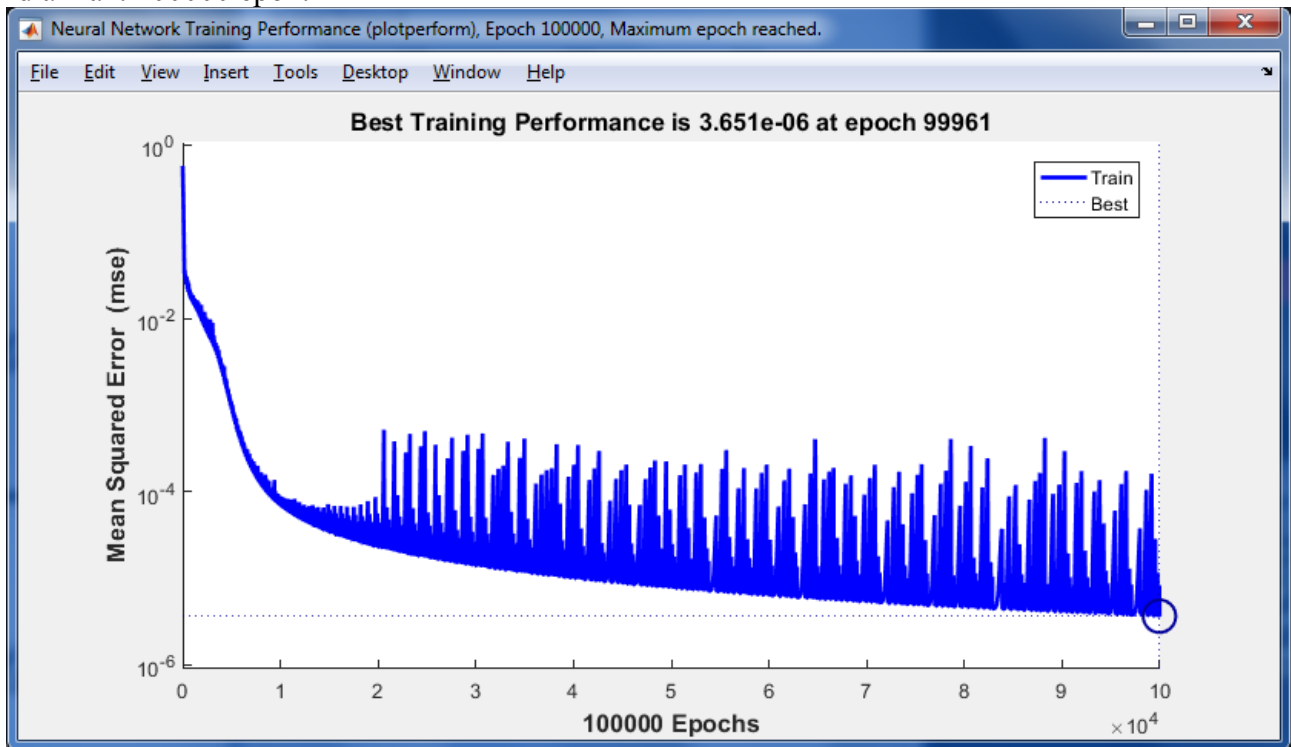
Poniższa tabela prezentuje wyniki działania programu dla kilku przykładowych liter występujących w danych uczących oraz dla kilku symboli, których nie ma w danych uczących ale przypominają kształtem litery znajdujące się w zestawie uczącym. Na ostatnim polu każdej kolumny znajduje się litera, która według algorytmu w najwyższym stopniu pokrywa się z wprowadzonym kształtem. Tabela została przygotowana w oparciu o efekty działania programu dla współczynnika uczenia równego 0.001, dla tej wartości program poprawnie rozpoznawał wszystkie znaki znajdujące się w danych uczących oraz w zadowalający sposób przyrównywał symbole niewystępujące w danych uczących do ich odpowiedników.

Wprowadzony kształt -> Litery z danych uczących:							
<b>A</b>	<b>0.9560</b>	-0.0032	-0.0058	-0.2403	-0.2401	-0.2402	<b>0.9492</b>
<b>B</b>	-0.0103	-0.0066	-0.0039	0.0006	0.0006	0.0005	-0.9475
<b>C</b>	0.0111	-0.0004	0.0051	-0.0004	-0.0004	-0.0004	-0.7360
<b>D</b>	0.0104	0.0086	0.0065	0.0004	0.0004	0.0004	-0.2287
<b>E</b>	-0.0079	-0.0044	-0.0061	-0.0001	-0.0001	-0.0001	0.5816
<b>F</b>	-0.0123	-0.0142	-0.0074	-0.0001	-0.0001	-0.0001	0.2527
<b>G</b>	0.0207	0.0185	0.0121	0.0005	0.0006	0.0005	0.6206
<b>H</b>	0.2453	0.0052	0.0050	0.2404	0.2402	0.2399	0.4061
<b>I</b>	-0.0050	0.0034	0.0022	0.0004	-0.0004	-0.0002	-0.5955
<b>J</b>	0.0124	0.0130	-0.0041	-0.0086	0.0073	0.0080	0.5884
<b>K</b>	0.0042	0.0086	0.0084	0.0008	0.0008	0.0008	0.2146
<b>L</b>	-0.0015	0.0008	0.0005	-0.0001	0.0002	0.0001	0.1824
<b>N</b>	-0.0070	<b>0.9796</b>	-0.0043	-0.0004	-0.0004	-0.0005	0.8052
<b>O</b>	0.2258	-0.0117	-0.0083	0.2396	0.2396	0.2394	-0.7504
<b>P</b>	0.0252	0.0215	0.0191	0.0004	0.0004	0.0004	0.4141
<b>R</b>	-0.0035	-0.0038	<b>0.9804</b>	-0.0009	-0.0011	-0.0009	-0.9831
<b>S</b>	0.0033	0.0028	0.0028	-0.0001	-0.0002	-0.0002	0.8717
<b>T</b>	-0.0074	-0.0107	-0.0124	-0.0134	-0.0115	-0.0128	-0.3936
<b>U</b>	-0.2363	-0.0001	0.0014	<b>0.9834</b>	<b>0.9714</b>	<b>0.9710</b>	-0.5050
<b>Y</b>	-0.0205	-0.0242	0.0148	-0.0125	0.0106	0.0118	-0.8313
Litera rozpoznana przez algorytm:	<b>A</b>	<b>N</b>	<b>R</b>	<b>U</b>	<b>U</b>	<b>U</b>	<b>A</b>

Wykresy przedstawiające wydajność uczenia sieci:  
-dla max. 10000 epok

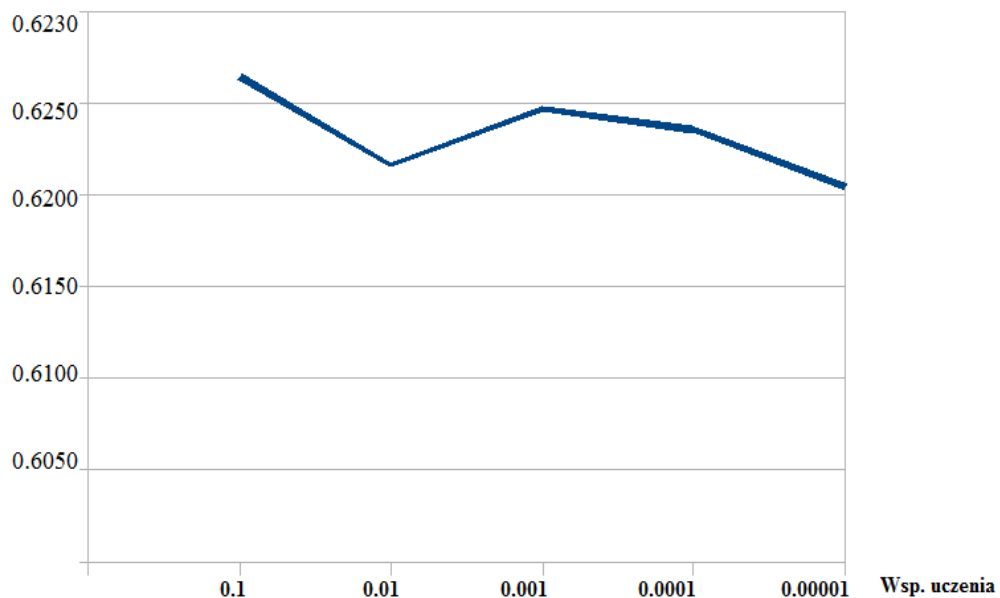


-dla max. 100000 epok:

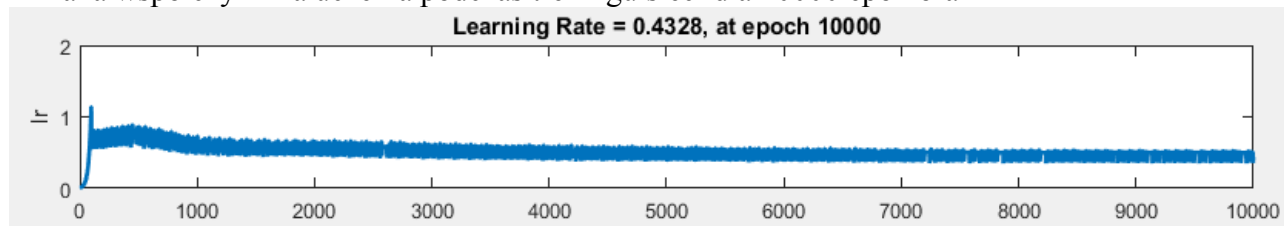


Wpływ dobranego współczynnika uczenia na wyniki (im większa różnica pomiędzy wynikiem poprawnym a wynikami niepoprawnymi tym niedokładność jest niższa):

Niedokładność  
wyniku



Zmiana współczynnika uczenia podczas treningu sieci dla 10000 epok oraz



### Analiza:

Analizę wyników przeprowadzam dla różnych współczynników uczenia przy zachowaniu jednakowego limitu epok wynoszącego 10000.

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. Nawet dla znaków różniących się w niewielkim stopniu sieć nie miała dużych problemów z rozpoznaniem liter, można to zaobserwować na przykładzie trzech kształtów liter U, umieszczonych w tabeli: wartość podobieństwa dla litery U zmienia się nieznacznie, podobnie jak podobieństwo innych liter. Program zaczął mieć problemy podczas próby rozpoznania ostatniego symbolu: można zauważyć że dla kilku liter wartości są bardzo wysokie, wiele przekracza 0.5 co sugeruje, że być może liczba epok była zbyt mała by można było rozróżniać kształty bardzo niepodobne do danych wejściowych.

Analizując wykres przedstawiający wydajność uczenia sieci można zauważyć że błąd średniokwadratowy posiada tendencję do spadku wraz ze wzrostem liczby epok. Zwiększenie limitu epok faktycznie skutkuje lepszymi efektami treningu ale niestety efektywność tej metody spada wraz z każdą kolejną epoką w wyniku czego zwiększanie liczby epok do bardzo dużych wielkości nie poprawi efektywności programu w dużym stopniu a w dodatku spowoduje, że czas treningu znacząco się wydłuży np. trening dla 10000 epok trwał ok 7 sekund, natomiast dla 100000 epok przy zachowaniu tych samych parametrów treningu czas wynosił aż 5 minut natomiast wartość błędu średniokwadratowego nie zmalała znacząco w porównaniu do treningu przy 10000 epok. W dodatku wykres dla 100000 epok pokazuje, że po przekroczeniu 86502 epoki wydajność zaczęła spadać – sieć osiągnęła wtedy prawdopodobnie najniższy możliwy dla niej do osiągnięcia w procesie treningu błąd średniokwadratowy.

Ostatni wykres pokazuje, jak zmiana współczynnika uczenia wpływała na efektywność programu. Wraz ze wzrostem współczynnika rosła również niedokładność wyników co mogło skutkować tym że przy zbyt dużym współczynniku sieć mogłaby zacząć rozpoznawać litery w sposób niepoprawny. Zmniejszanie wsp. uczenia doprowadzało do otrzymania nieco lepszych wyników (rosła przepaść pomiędzy literą poprawną a pozostałymi) jednak ta zmiana była bardzo niewielka, zmniejszenie współczynnika 100 razy spowodowało zmianę na poziomie ok 0.003. Modyfikacja wsp. uczenia skutkowała także niewielkimi zmianami w czasie trwania treningu: 100 krotne zmniejszenie wydłużyło czas uczenia o ok 10%.

Analizując wykres zmiany współczynnika uczenia podczas treningu sieci można zauważyć, że największe wahania wartości współczynnika występują na początku treningu, po przekroczeniu 1000 epok zmiana jest niewielka, choć widać że wartości powoli dążą do 0 a właściwie do wartości wsp. uczenia podanej w parametrach treningu sieci – w tym przypadku wynosi on 0.001.

### Wnioski:

Obserwując i analizując efekty działania sieci zaobserwowałem że sieć może w bardzo skuteczny

sposób rozróżniać litery jeśli otrzyma odpowiedniej jakości dane wejściowe a także dobrze dobrane parametry treningu, w szczególności duży wpływ ma tutaj ilość epok: ustawienie zbyt niskiego limitu będzie skutkowało zbyt wczesnym przerywaniem procesu uczenia co doprowadzi do niepoprawnego działania sieci. Sieć wielowarstwowa okazała się bardzo skuteczna: w tabelce można zauważyć np. że dla litery N program nie miał problemu w odróżnieniu jej od litery H pomimo tego, że różnica między nimi wynosiła zaledwie 2 kratki. Sieć potrafiła w poprawny sposób rozpoznawać litery znane z danych uczących z podobieństwem nie mniejszym niż 0.9, choć jak widać w tabeli zwykle było ono wyższe ( $>0.95$ ). Zaskakująco dobre były efekty rozpoznawania liter niekompletnych, takich jak np. 3 wersje litery U znajdujące się w tabeli. Sieć nie miała wątpliwości co do tego, jaką literę otrzymała pomimo nawet kilku brakujących kratek.

Problemy w działaniu sieci pojawiać się zaczęły gdy próbowałem rozpoznawać kształty bardzo niepodobne do tych, które umieściłem w danych wejściowych: sieć dawała wyniki o bardzo wysokim podobieństwie do wielu liter co doprowadzać może do tego, że sieć w błędny sposób zacznie rozpoznawać litery zbyt bardzo odbiegające od wzorców. Problem ten prawdopodobnie można rozwiązać na kilka sposobów: poprzez znaczne zwiększenie ilości epok treningu, zmianę współczynnika uczenia oraz zwiększenie rozmiaru tablic przechowujących kształty liter, gdyż większa ilość danych dla każdej litery spowoduje, że sieć będzie miała więcej punktów odniesienia podczas rozróżniania symboli. Niestety zastosowanie tych rozwiązań doprowadzi do wydłużenia (często znacznego) czasu potrzebnego na trening więc w zależności od zapotrzebowania nie koniecznie będzie to opłacalne.

Podczas analizowania wydajności uczenia zaobserwowałem, zgodnie z oczekiwaniami, że istnieje punkt, po przekroczeniu którego jakość treningu przestanie wzrastać co nasuwa wniosek, że w celu osiągnięcia jak najlepszych efektów działania sieci (takich jak zależność pomiędzy jakością a czasem treningu) należy postarać się zlokalizować punkt, w którym proces treningu osiąga najlepszą wydajność i w okolicach tego punktu zaprzestać uczenia sieci, gdyż dalsze uczenie nie będzie skutkowało znaczącą poprawą działania sieci.

Podobnie jak z ilością epok, modyfikacja współczynnika uczenia będzie dawać zadowalające efekty tylko do pewnego momentu, zmniejszenie współczynnika zbyt bardzo nie poprawi znacząco efektywności sieci a może skutkować znacznym wydłużeniem czasu uczenia; być może nie jest to szczególnie uciążliwe dla przypadków które analizowałem ale przy bardziej zaawansowanych sieciach neuronowych czas treningu mógłby wzrosnąć o bardzo duże wartości, sięgające czasem wielu godzin.

### Listing programu wraz z opisem:

```
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;];
%wejścia do sieci i min/max wartości wejść
S=[40 20 20]; %ilosc neuronow w kazdej z warstw
net = newff(PR,S,{'tansig','tansig','tansig'},'traingda');
```

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

```

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

net.name='Rozroznianie liter';

%parametry treningu
net.trainParam.epochs = 10000; %max ilość epok
net.trainParam.mu = 0.001; %współczynnik uczenia

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

%litery do testu:
testa = [0; 1; 1; 0; %A
1; 0; 0; 1;
1; 1; 1; 1;
1; 0; 0; 1;
1; 0; 0; 1];
testb = [1; 1; 1; 0; %b
1; 0; 0; 1;
1; 1; 1; 0;
1; 0; 0; 1;
1; 1; 1; 0];
testc = [0; 1; 1; 1; %c
1; 0; 0; 0;
1; 0; 0; 0;
1; 0; 0; 0;
0; 1; 1; 1];
testd = [1; 1; 1; 0; %d
1; 0; 0; 1;
1; 0; 0; 1;

```



```

1; 0; 0; 1;
1; 1; 1; 0];
teste = [1; 1; 1; 1; %e
1; 0; 0; 0;
1; 1; 1; 0;
1; 0; 0; 0;
1; 1; 1; 1];
testf = [1; 1; 1; 1; %f
1; 0; 0; 0;
1; 1; 1; 0;
1; 0; 0; 0;
1; 0; 0; 0];
testg = [0; 1; 1; 1; %g
1; 0; 0; 0;
1; 0; 1; 1;
1; 0; 0; 1;
0; 1; 1; 1];
testh = [1; 0; 0; 1; %h
1; 0; 0; 1;
1; 1; 1; 1;
1; 0; 0; 1;
1; 0; 0; 1];
testi = [1; 0; 0; 0; %i
1; 0; 0; 0;
1; 0; 0; 0;
1; 0; 0; 0;
1; 0; 0; 0];
testj = [1; 1; 1; 1; %j
0; 0; 0; 1;
0; 0; 0; 1;
1; 0; 0; 1;
0; 1; 1; 1];
testk = [1; 0; 0; 1; %k
1; 0; 1; 0;
1; 1; 0; 0;
1; 0; 1; 0;
1; 0; 0; 1];
testl = [1; 0; 0; 0; %l
1; 0; 0; 0;
1; 0; 0; 0;
1; 0; 0; 0;
1; 1; 1; 1];
testn = [1; 0; 0; 1; %n
1; 1; 0; 1;
1; 0; 1; 1;
1; 0; 0; 1;
1; 0; 0; 1];
testo = [0; 1; 1; 0; %o
1; 0; 0; 1;
1; 0; 0; 1;
1; 0; 0; 1;
0; 1; 1; 0];
testp = [1; 1; 1; 0; %p
1; 0; 0; 1;
1; 1; 1; 0;
1; 0; 0; 0;
1; 0; 0; 0];
testr = [1; 1; 1; 0; %r
1; 0; 0; 1;
1; 1; 1; 0;
1; 0; 1; 0;
1; 0; 0; 1];
tests = [0; 1; 1; 1; %s
1; 0; 0; 0;

```

```

0; 1; 1; 0;
0; 0; 0; 1;
1; 1; 1; 0];
testt = [1; 1; 1; 0; %t
0; 1; 0; 0;
0; 1; 0; 0;
0; 1; 0; 0;
0; 1; 0; 0;
0; 1; 0; 0];
testu = [1; 0; 0; 1; %u
1; 0; 0; 1;
1; 0; 0; 1;
1; 0; 0; 1;
0; 1; 1; 0];
testy = [1; 0; 1; 0; %y
1; 0; 1; 0;
0; 1; 0; 0;
0; 1; 0; 0;
0; 1; 0; 0];
%ksztalt nie wystepujacy w danych uczacych
test = [0; 1; 0; 0;
1; 0; 1; 0;
1; 1; 1; 1;
1; 0; 0; 0;
1; 0; 0; 0];

efekt=sim(net, testa);%testowanie sieci
%testX gdzie x to litera

%szukanie największej wartosci
max=1;
for i=1:1:20
    if (efekt(max)<efekt(i))
        max=i;
    end;
end

%wypisywanie wartosci dla poszczegolnych liter
disp('Wartosci wyjsciowe algorytmu dla wszystkich liter:')
disp('A='),disp(efekt(1));
disp('B='),disp(efekt(2));
disp('C='),disp(efekt(3));
disp('D='),disp(efekt(4));
disp('E='),disp(efekt(5));
disp('F='),disp(efekt(6));
disp('G='),disp(efekt(7));
disp('H='),disp(efekt(8));
disp('I='),disp(efekt(9));
disp('J='),disp(efekt(10));
disp('K='),disp(efekt(11));
disp('L='),disp(efekt(12));
disp('N='),disp(efekt(13));
disp('O='),disp(efekt(14));
disp('P='),disp(efekt(15));
disp('R='),disp(efekt(16));
disp('S='),disp(efekt(17));
disp('T='),disp(efekt(18));
disp('U='),disp(efekt(19));
disp('Y='),disp(efekt(20));

%wypisywanie jaka to litera
switch max
case 1
    disp('Wpisana litera to A')

```

```

case 2
    disp('Wpisana litera to B')
case 3
    disp('Wpisana litera to C')
case 4
    disp('Wpisana litera to D')
case 5
    disp('Wpisana litera to E')
case 6
    disp('Wpisana litera to F')
case 7
    disp('Wpisana litera to G')
case 8
    disp('Wpisana litera to H')
case 9
    disp('Wpisana litera to I')
case 10
    disp('Wpisana litera to J')
case 11
    disp('Wpisana litera to K')
case 12
    disp('Wpisana litera to L')
case 13
    disp('Wpisana litera to N')
case 14
    disp('Wpisana litera to O')
case 15
    disp('Wpisana litera to P')
case 16
    disp('Wpisana litera to R')
case 17
    disp('Wpisana litera to S')
case 18
    disp('Wpisana litera to T')
case 19
    disp('Wpisana litera to U')
case 20
    disp('Wpisana litera to Y')
otherwise
    disp('Bład dzialania programu')
end

```

Działanie programu: w funkcji sim jako drugi parametr umieszcza się literę/kształt który ma być przeanalizowany przez sieć, następnie program wypisuje wartości podobieństwa do każdej z liter, jakie otrzymał w danych uczących i wybiera najwyższą z tych wartości na podstawie której daje komunikat jaka to litera.

Opis zmiennych/funkcji w programie:

PR – zmienna wejściowa dla funkcji tworzących sieć neuronową, składa się z 20 par 0 1 które obrazują minimalną (0) i maksymalną (1) wartość dla każdego z 20 wejść (czyli pól składających się na kształt litery)

S - zmienna przechowująca ilość neuronów w poszczególnych warstwach sieci, ostatnia wartość stanowi ilość wyjść z sieci, wynosi 20 ponieważ potrzebuję po jednym wyniku na każdą literę do której porównuję badany kształt; S przechowuje 3 wartości więc utworzona sieć będzie się składać z 3 warstw

net – zmienna do której będzie przypisana tworzona sieć neuronowa

`net = newff(PR,S,{'tansig','tansig','tansig'},'traingda')` – utworzenie wielowarstwowej sieci neuronowej, dla podanego `S` składającego się z 3 wartości będzie to skutkowało stworzeniem sieci o 3 warstwach; `tansig` – parametr określający funkcję aktywacji neuronów w poszczególnych warstwach sieci, tu: tangens hiperboliczny; `traingda` – funkcja wykorzystywana przy treningu sieci, tu: algorytm propagacji wstecznej błędu

`WE` – zmienna przechowująca dane uczące (wejściowe)

`WY` – zmienna przechowująca dane wyjściowe odpowiadające danym uczącym, jest to tablica 20x20 ponieważ na każdą możliwą literę potrzebuję po 1 wyjściu \* ilość liter; wynika to z działania sieci która wyprowadza wyłącznie 0 i 1 więc poprzez umieszczenie 1 w odpowiednim miejscu umożliwiam sieci rozróżnianie liter z danych uczących

`net.trainParam.x` – określenie parametrów treningu:

\*`epochs` – maksymalna liczba epok

\*`mu` – wartość współczynnika uczenia

`train(net, WE, WY)` – uczenie sieci `net` z wykorzystaniem danych wejściowych i wyjściowych oraz przy pomocy zdefiniowanych wcześniej parametrów oraz metody określonej w funkcji `newff` (algorytmu propagacji wstecznej błędu)

`testx` – dane testujące, modele zero-jedynkowe liter wykorzystywane do testu, w miejscu `x` znajduje się litera, która odpowiada wartości tej zmiennej

`efekt=sim(net, testx)` - symulacja sieci `net` przy pomocy funkcji `sim()`, `testx` to litera która ma być rozpoznana, efekt symulacji zostaje wpisany do zmiennej `efekt`

pętla `for` – szuka najwyższej wartości wyjścia

`disp('A='),disp(efekt(1))` itp. - wypisuje wartość podobieństwa dla poszczególnych liter

`switch` – instrukcja pozwalająca na wypisanie jaka litera zdaniem sieci została podana podczas symulacji