Temat: ***Budowa​ ​i​ ​działanie​ ​sieci​ ​wielowarstwowej- algorytm wstecznej propagacji***

Data oddania: 20.10.2017

Prowadzący: dr inż. Dorota Wilk - Kołodziejczyk

Przedmiot: Podstawy sztucznej inteligencji

Autor: Przemysław Adamus 286084

Kierunek: Inżynieria Obliczeniowa sem.V

Spis treści:

1. Cel projektu;
2. Wstęp i opis projektu;
3. Budowa sieci;
4. Zestaw uczący;
5. Syntetyczny​ ​opis​ ​budowy​​ ​wykorzystanego​ ​algorytmu​ ​uczenia w projekcie;
6. Opis wykorzystanych funkcji;
7. Wyniki i analiza;
8. Wnioski;
9. Przykładowa odpowiedź programu;
10. Źródła;
11. Kod programu.
12. **CEL:**

Celem ćwiczenia było utworzenie wielowarstwowej sieci neuronowej wykorzystującej algorytm wstecznej propagacji błędu w celu rozpoznania podstawowych liter polskiego alfabetu.

1. **WSTĘP I KRÓTKI OPIS PROJEKTU:**

Do rozwiązania zadanego problemu wykorzystałem wiedzę i projekt c++ z wcześniejszego scenariusza. Lecz tym razem zdecydowałem się na rozwiązanie problemu w środowisku MatLab R2016a. Do realizacji scenariusza wykorzystałem biblioteke do tworzenia sztucznych sieci neuronowych „Neural Network Toolbox” oraz zawarte w niej gotowe funkcje. Do rozwiązywania zadania zbudowałem sieć trójwarstwową. Pierwsza i druga warstwa ma rozmiar 40, czyli dwukrotność danych wejściowych natomiast trzecia warstwa musi być równa ilości danych wejściowych – czyli 20;

1. **BUDOWA WYKORZYSTANEGO SIECI WIELOWASTROWEJ:**

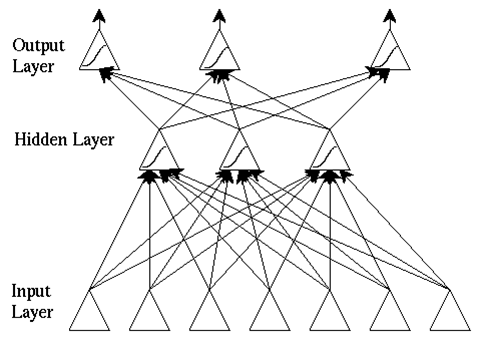
Perceptronem nazywamy prosty element obliczeniowy, który sumuje ważone sygnały wejściowe i porównuję tę sumę z progiem aktywacji - w zależności od wyniku perceptron może być albo wzbudzony (wynik 1), albo nie (wynik 0).

Suma membranowa jest to suma wszystkich iloczynów wejść i ich wag. Jest ona przekazywana dalej do funkcji aktywacji.

Funkcja aktywacji w sztucznej inteligencji określa funkcje, według której obliczana jest wartość wyjścia neuronów sieci neuronowej. Do najczęściej używanych funkcji aktywacji należą: funkcja liniowa, funkcja progowa unipolarna, funkcja progowa bipolarna, funkcja sigmoidalna unipolarna, funkcja sigmoidalna bipolarna.

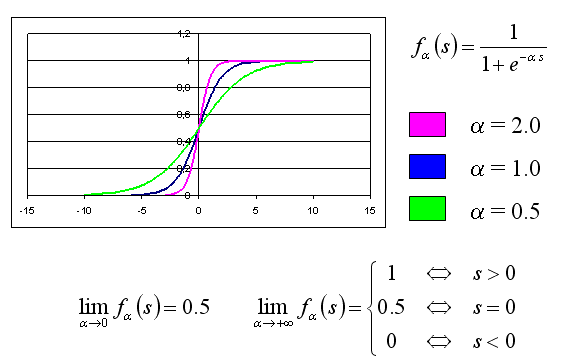
Rezultat wiadomość zwrotna od neuronu. Może być przekazana dalej, lub być ostatecznym wynikiem. Rezultat jest wartością logiczną więc powinien zawierać tylko informacjię *true* lub *false.*

Sieć wielowarstwowa - to siec składająca się z kilku warstw neuronów, warstwy wejściowej, warstw ukrytych i warstwy wyjściowej. Ograniczenia pojedynczych perceptronów spowodowały w latach 80-tych wzrost zainteresowania sieciami wielowarstwowymi i opracowanie algorytmu ich uczenia (propagacja wsteczna). Okazuje się, że już stosunkowo prosta architektura polegająca na tym, że tworzymy kilka (zwykle 3) warstw neuronów połączonych w ten sposób, że wyjścia neuronów należących do warstwy niższej połączone są z wejściami neuronów należących do warstwy wyższej (każdy z każdym), pozwala na tworzenie sieci o niemal dowolnej charakterystyce. Działanie takiej sieci polega na liczeniu odpowiedzi neuronów w kolejnych warstwach - najpierw w pierwszej, do której trafiają sygnały z wejść sieci, potem (na podstawie wyników pierwszej warstwy) liczymy odpowiedzi drugiej warstwy neuronów itd., przy czym odpowiedzi ostatniej warstwy traktowane są jako wyjścia z sieci.

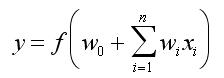


Nie jest znana ogólna metoda projektowania optymalnej architektury sieci neuronowej dla zadanego problemu. W praktyce zwykle wykorzystuje się sieci podobne do powyższej.

Aby sieć wielowarstwowa mogła wykorzystać swe atuty (w szczególności trenować wagi algorytmem propagacji wstecznej), funkcja aktywacji neuronów nie powinna być skokowa. W praktyce wykorzystuje się kilka rodzajów funkcji aktywacji, dbając jedynie, by była ona monotoniczna, różniczkowalna (jak się przekonamy, to cecha ważna podczas trenowania sieci) i z wyglądu przypominała "rozmyty schodek". Przyjżyjmy się funkcji sigmoidalnaej - poniżej przedstawiono własności i przykładowe wykresy tej funkcji (dla różnych wartości dodatkowego parametru):



Ogólny wzór na wartość wyjściową neuronu przedstawia się następująco:



gdzie *w0* jest dodatkową wagą pełniącą rolę analogiczną do progu aktywacji w perceptronie.

1. **ZESTAW UCZĄCY:**

W moim projekcie litery reprezentowane są na matrycach o wymiarach 6x5. Zgodnie z treścią zadania jest ich 20 sztuk. Zdecydowałem się na 10 łatwych do reprezentacji, początkowych liczb naszego alfabetu. Zapisałem je do tablicy dwu wymiarowej „dataIN [20][30]” w kolejności alfabetycznej „AaBbCcDdEeFfHhIiKkLl”

%A a B b C c D d E e F f H h I i K k L l

1. dataIN= [0 0 1 1 0 0 1 0 1 0 1 0 1 1 0 0 1 1 1 0;
2. 1 0 1 0 1 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0;
3. 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 0 0 0 0 0;
4. 1 0 1 0 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 0;
5. 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0;
6. 1 0 1 1 1 0 1 0 1 0 1 0 1 1 0 0 1 1 1 0;
7. 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1;
8. 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1;
9. 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0;
10. 1 0 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0;
11. 1 0 1 1 1 0 1 0 1 1 1 0 1 1 0 0 1 1 1 0;
12. 0 0 1 0 0 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0;
13. 0 0 1 0 0 1 0 0 1 0 1 0 1 1 1 0 1 0 0 1;
14. 0 0 1 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0 0;
15. 1 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0;
16. 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 0;
17. 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 1 0 0;
18. 1 1 0 1 0 0 0 1 0 1 0 1 0 0 1 1 0 1 0 1;
19. 1 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 1 0 0 0;
20. 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0;
21. 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 0;
22. 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0;
23. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1;
24. 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0;
25. 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0;
26. 1 0 1 1 0 0 1 0 1 0 1 0 1 1 0 0 1 1 1 0;
27. 0 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 0 0 0 0;
28. 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 0 1;

0 1 1 0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1;

1. 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0;];
2. **ALGORYTM UCZENIA - ALGORYTM WSTECZNEJ PROPAGACJI :**

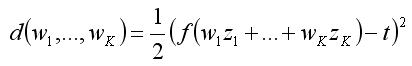
Podobnie jak w przypadku pojedynczego neuronu, główną zaletą sieci neuronowej jest to, że nie musimy "ręcznie" dobierać wag. Możemy te wagi wytrenować, czyli znaleźć ich w przybliżeniu optymalny zestaw za pomocą metody obliczeniowej zwanej wsteczną propagacją błędu. Jest to metoda umożliwiająca modyfikację wag w sieci o architekturze warstwowej (opisanej w poprzednim rozdziale), we wszystkich jej warstwach.

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 wielkości 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 znaleźć 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:



W celu ustalenia, o ile powinna zmienić się waga neuronu, powinniśmy "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.

*Działanie algorytmu i jego opis opracowany na podstawie wykładu zamieszczonego przez Polsko – Japońską Akademią Technik Komputerowych. – Żródła 1*

1. **OPIS WYKORZYSTANYCH FUNKCJI:**

newff – funkcja do tworzenia sieci wielowarstwowej jednokierunkowej.

* PR – liczba wektorów wejścia o [min max] wartościach. Jest ich 30;
* [40 40 20] – liczba neuronów (wyjscia);
* **Tansig** – funkcja aktywacji jako tangens hiperboliczny;
* **Traingda** – nazwa funkcji wykorzystywanej do treningu sieci. W tym przypadku jest to metoda wstecznej propagacji błędów;

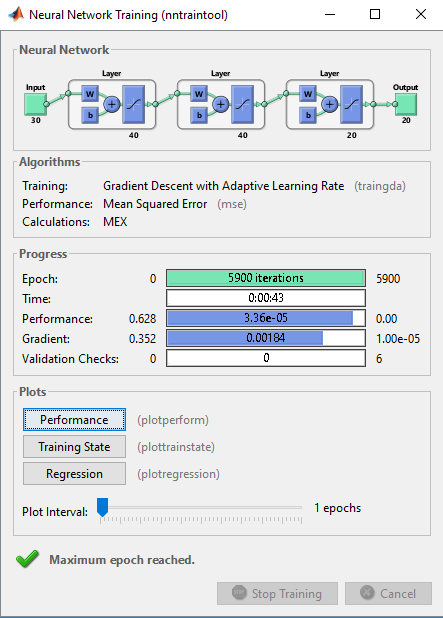
round – funkcja zaokrąglająca. Dzięki zwracanej wartości możemy rozróżnić wielkość liery. 0-mała 1-duża;

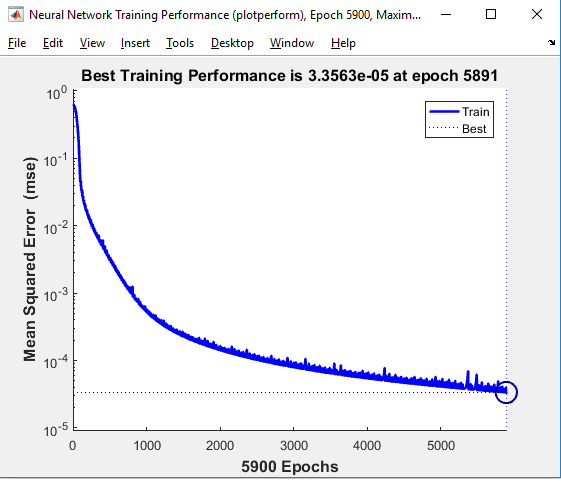
sim – symulacja rezultatu dla podanych danych wejściowych;

train – trening/nauka sieci neuronowej;

1. **WYNIKI I ANALIZA:**

Macierz wyjściowa „dataOUT” jest tego samego rozmiaru co wejściowa. „dataOUT” jest macierzą diagonalną binarną, czyli przyjmuje wartość 1 na diagonali. Taki dobór był konieczny, ponieważ wynik powinien być unikalny dla każdej osobnej litery. Do realizacji scenariusza wykorzystałem biblioteke do tworzenia sztucznych sieci neuronowych „Neural Network Toolbox” oraz zawarte w niej gotowe funkcje opisane w punkcie 5. Do rozwiązywania zadania zbudowałem sieć trójwarstwową. Pierwsza i druga warstwa ma rozmiar 40, czyli dwukrotność danych wejściowych natomiast trzecia warstwa musi być równa ilości danych wejściowych – czyli 20; Testowałem również inne rozmiary warstw. Metodą prób i błędów i według własnej oceny uznałem, że zastosowany rozmiar miał najlepszy stosunek ilości epok i czasu nauczania do dokładności zwracanych rezultatów. Przy użyciu większych matryc na pewno zdecydował bym się na większe warstwy mojej sieci.

Przetestowałem również wpływ innych cech sieci na końcowe wyniki. Godnym odnotowania jest fakt iż sieć z wartościami domyślnymi nie działa optymalnie. Czas nauki takiej sieci jest mocno wydłużony i kończy się dopiero po przekroczeniu maksymalnej ilości epok.

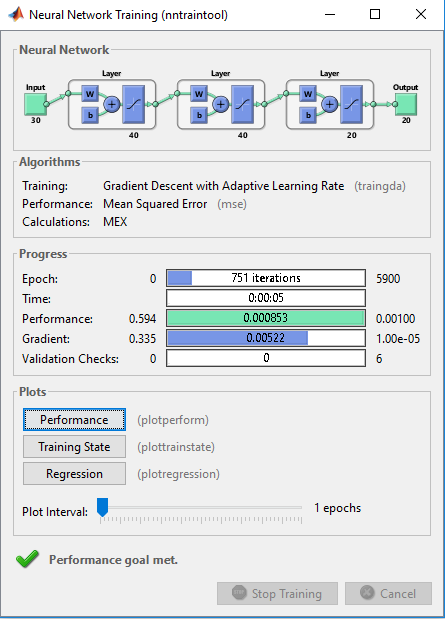
**

Domyślne cechy sieci:

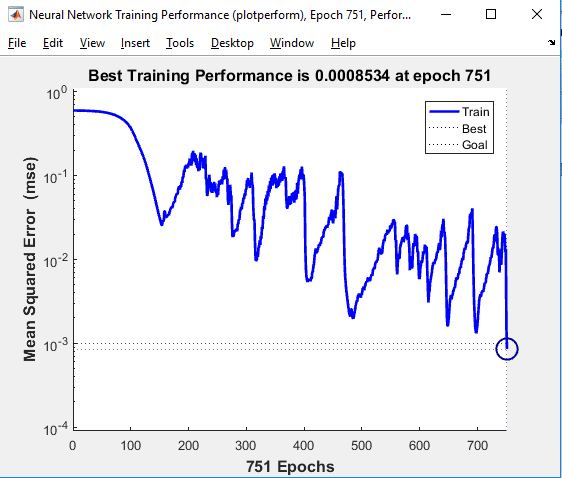
* Learning rate = 0.01
* Maximum performance increase = 1.04
* Goal = 0

Ogromnym minusem rozwiązania domyślnego jest czas nauki, lecz niesprawiedliwym by było niezauważenie dużej dokładności otrzymanych rezultatów. By otrzymać rozwiązanie optymalniejsze, metodą prób i błędów, zmieniałem cechy mojej sieci.

I tak:

* Ustawienie dokładności (goal) na 0.001 pozwalało drastycznie skrócić ilość iteracji sieci do ok. 450 – 752. Wyniki nie były już tak idealnie dokładne. Wahały się między 0.83 ,a 0.87 – co jednak w moim odczuciu dalej spełnia wymogi zadania. Dasze zwiększanie go wciąż zmniejszało ilość epok, lecz by nie tracić dokładności zdecydowałem się na 0.001.
* Zmniejszenie współczynnika nauki z 0.01 na 0.001 zwiększyło nieznacznie ilość epok nauki, ale znacznie wpłynęło na dokładność.
* Maksymalne zwiększenie wydajności ustawiłem na 1.5. Otrzymałem dzięki temu większe różnice między poszczególnymi wynikami dla poszczególnych liter.

*Dla moich wartości cech sieci*

*Wykres wydajności dla moich wartości chech sieci*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| LP | Wartości domyślne | | | Wartości optymalne | | | |
| 1. | L. epok | Dokładność | Wydajność | L. epok | Dokładność | Wydajność | |
| 2. | 5900 | 0.9716 | 3.67e-05 | 751 | 0.8727 | | 0.000853 |
| 3. | 5900 | 0.9845 | 3.71e-05 | 450 | 0.8584 | | 0.000909 |
| 4. | 5900 | 0.9905 | 3.65-e05 | 562 | 0.8729 | | 0.000904 |
| 5. | 5900 | 0.9732 | 3.70e-05 | 752 | 0.8341 | | 0.000925 |

1. **PODSUMOWANIE:**

Po przetestowaniu działania programu mogę stwierdzić, że dobrane prze mnie wartości wszystkich 3 są najbardziej optymalne. Czas nauki sieci jest do zaakceptowania i nie przekracza zazwyczaj 800 epok. Natomiast odpowiedzi i zwracane rezultaty sieci są poprawne. Symulacja przebiega poprawnie i nie występuje błąd zaokrąglenia. Warunki zadania zostały spełnione.

1. **PRZYKŁADOWA ODPOWIEDŹ PROGRAMU:**

Wynik: 'A'

0.8584

###################################################

Wartosci wyjsciowe algorytmu dla wszystkich liter:

A=

0.8584

a=

0.0098

B=

0.0079

b=

-0.0163

C=

0.0042

c=

0.0187

D=

0.0202

d=

-0.0256

E=

0.0090

e=

-0.0164

F=

-0.0351

f=

0.0217

H=

-0.0235

h=

0.0045

I=

0.0361

i=

0.0209

K=

0.0128

k=

0.0295

L=

0.0252

l=

-0.0488

1. **ŹRÓDŁA:**

* <https://edux.pjwstk.edu.pl/mat/273/lec/wyklad3/w3.html>
* <https://www.wikipedia.pl>
* https://www.en.wikipedia.org
* <https://forum.pasja-informatyki.pl/73877/sztuczne-sieci-neuronowe>
* <http://www.if.uz.zgora.pl/~mdudek/siecineuronowe.pdf>
* <https://www.youtube.com/watch?v=Wa_9S20SkKw&t=995s>
* http://home.agh.edu.pl/~horzyk/pracedoktorskie/adrianhorzykpracadoktorska.pdf
* Materiały udostępnione przez dr inż. Dorota Wilk – Kołodziejczyk

1. **KOD PROGRAMU:**
2. close all; clc;
4. 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;
5. 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
6. 0 1; 0 1; 0 1; 0 1];
7. %wejscia do sieci i min/max wartosci wejsc
8. inputs=20; %ilość wejść
10. NET = newff(PR,[2\*inputs 2\*inputs inputs], {'tansig','tansig','tansig'},'traingda');
12. %A a B b C c D d E e F f H h I i K k L l
13. dataIN=[0 0 1 1 0 0 1 0 1 0 1 0 1 1 0 0 1 1 1 0;
14. 1 0 1 0 1 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0;
15. 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 0 0 0 0 0;
16. 1 0 1 0 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 0;
17. 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0;
18. 1 0 1 1 1 0 1 0 1 0 1 0 1 1 0 0 1 1 1 0;
19. 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1;
20. 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1;
21. 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0;
22. 1 0 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0;
23. 1 0 1 1 1 0 1 0 1 1 1 0 1 1 0 0 1 1 1 0;
24. 0 0 1 0 0 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0;
25. 0 0 1 0 0 1 0 0 1 0 1 0 1 1 1 0 1 0 0 1;
26. 0 0 1 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0 0;
27. 1 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0;
28. 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 0;
29. 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 1 0 0;
30. 1 1 0 1 0 0 0 1 0 1 0 1 0 0 1 1 0 1 0 1;
31. 1 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 1 0 0 0;
32. 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0;
33. 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 0;
34. 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0;
35. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1;
36. 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0;
37. 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0;
38. 1 0 1 1 0 0 1 0 1 0 1 0 1 1 0 0 1 1 1 0;
39. 0 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 0 0 0 0;
40. 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 0 1;
41. 0 1 1 0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1;
42. 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0;];
44. dataOUT =[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
45. 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
46. 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
47. 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
48. 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
49. 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
50. 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0;
51. 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0;
52. 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0;
53. 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0;
54. 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0;
55. 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0;
56. 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
57. 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;
58. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0;
59. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0;
60. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0;
61. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0;
62. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0;
63. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1;];
65. NET.trainParam.epochs = 5900; % Maksymalna liczba epok
66. NET.trainParam.goal = 0.001; % Cel wydajności
67. NET.trainParam.max\_perf\_inc = 1.5; % Maksymmalne zwiększenie wydajności
68. NET.trainParam.lr = 0.001; % Współczynnik nauczania
69. NET = train(NET, dataIN, dataOUT); % Uczenie sieci
71. %Dane testowe
72. A = [ 0; 1; 1; 1; 0;
73. 1; 0; 0; 0; 1;
74. 1; 0; 0; 0; 1;
75. 1; 1; 1; 1; 1;
76. 1; 0; 0; 0; 1;
77. 1; 0; 0; 0; 1];
79. l = [0; 0; 0; 0; 0;
80. 0; 0; 1; 0; 0;
81. 0; 0; 1; 0; 0;
82. 0; 0; 1; 0; 0;
83. 0; 0; 1; 0; 0;
84. 0; 0; 1; 1; 0];

87. score=sim(NET,l);%symulacja odpowiedzi sieci
89. alphabet={'A','a','B','b','C','c','D','d','E','e','F','f','H','h','I','i','K','k','L','l'};
90. fprintf('Wynik:');
91. scoreEND = round(score);
93. %szukanie największej wartości
94. for i=1:inputs
95. if(scoreEND(i,1)==1)
96. disp(alphabet(i)); disp(score(i,1));
97. end
98. end
100. fprintf('###################################################');
101. disp(' ');
102. disp('Wartosci wyjsciowe algorytmu dla wszystkich liter:')
103. disp('A='),disp(score(1));
104. disp('a='),disp(score(2));
105. disp('B='),disp(score(3));
106. disp('b='),disp(score(4));
107. disp('C='),disp(score(5));
108. disp('c='),disp(score(6));
109. disp('D='),disp(score(7));
110. disp('d='),disp(score(8));
111. disp('E='),disp(score(9));
112. disp('e='),disp(score(10));
113. disp('F='),disp(score(11));
114. disp('f='),disp(score(12));
115. disp('H='),disp(score(13));
116. disp('h='),disp(score(14));
117. disp('I='),disp(score(15));
118. disp('i='),disp(score(16));
119. disp('K='),disp(score(17));
120. disp('k='),disp(score(18));
121. disp('L='),disp(score(19));
122. disp('l='),disp(score(20));