Weronika Muszyńska, gr. 03

**Sprawozdanie z ćwiczenia nr 3**

**Temat ćwiczenia:**

Budowa i działanie sieci wielowarstwowej.

**Celem ćwiczenia** jest poznanie budowy i działania wielowarstwowych sieci neuronowych

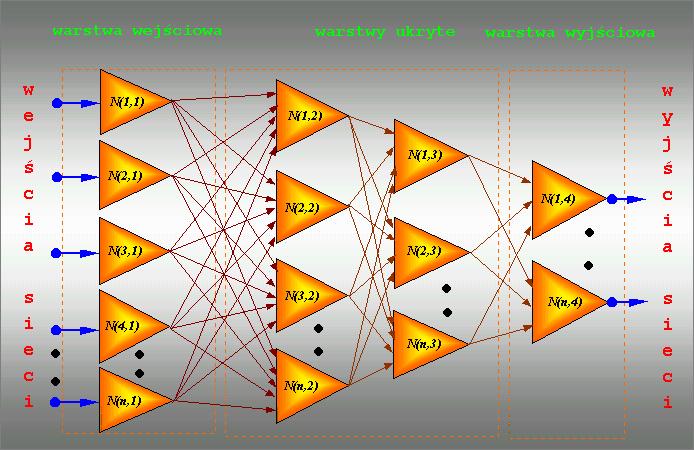
poprzez uczenie z użyciem algorytmu wstecznej propagacji błędu rozpoznawania

konkretnych liter alfabetu.

**Syntetyczny opis budowy sieci wielowarstwowej**

Sieć neuronowa – sieć, w którym wyjścia każdego neuronu są połączone, poprzez wagi, z wejściami wszystkich neuronów, także z jego własnym wejściem.

Podstawową cechą sieci wielowarstwowych jest ich zdolność realizacji dowolnie złożonych odwzorowań wejściowo-wyjściowych lib powierzchni decyzyjnych rozdzielających klasy obrazów. Uczenie sieci wielowarstwowych opiera się, podobnie jak w przypadku sieci jednowarstwowych, na wielokrotnym podawaniu na ich wejście obrazów uczących. Nauczona sieć często wytwarza zaskakujące odpowiedzii uogólnienia w zastosowaniach, w których jawne podanie prze człowieka odwzorowań i wykrycie istniejących zależności jest praktycznie niemożliwe.



***Rys. 1 Schemat uczenia wielowarstwowej sieci neuronowej***

Projektowanie sieci neuronowej zaczyna się już na poziomie analizy sformułowanego problemu. Inaczej mówiąc, jakie i ile danych chcemy lub możemy podać na wejścia sieci (zdeterminuje to wielkość warstwy wejściowej) oraz jaką odpowiedź chcemy uzyskać ( ilość wyjść sieci). Pozostanie zatem do określenia ilość warstw ukrytych i neuronów w tych warstwach. Jest to natrudniejszy moment tego etapu pracy. Przyjmuje się, że sieć z jedną warstwą ukrytą powinna nauczyć się rozwiązywania większości postawionych problemów. Nie znane są problemy wymagające do rozwiązania sieci z więcej niż trzema warstwami ukrytymi. Nie ma natomiast dobrej recepty na dobór właściwej ilości neuronów w warstwie ukrytej. Można próbować według wzoru:

http://edward_ch.republika.pl/IMAGES/ILNEURUK.GIF

gdzie:

*Nwu- ilość neuronów w warstwie ukrytej*

*Nwwe - ilość neuronów w warstwie wejściowej*

*Nwwy - ilość neuronów w warstwie wyjściowej*

Generalnie jednak uczenie rozpoczyna się z małą ich ilością a następnie, obserwując postępy tego procesu, doświadczalnie zwiększa się ich ilość.

Działanie SSN polega na tym, że sygnały pobudzające (wektor wejściowy) podawane na wejścia sieci, przetwarzane są w poszczególnych neuronach. Po tej projekcji na wyjściach sieci otrzymuje się wartości liczbowe, które stanowią odpowiedź sieci na pobudzenie i stanowią rozwiązanie postawionego problemu. Jednak aby takie rozwiązanie uzyskać, należy przejść żmudną drogę uczenia sieci.

W moim projekcie użyłam sieci jednokierunkowej wielowarstwowej.

Moja sieć zawiera 4 warstwy w tym 2 ukryte. Pierwsza warstwa posiada 25 perceptronów, druga 75 perceptronów, trzecia 50, natomiast ostatnia posiada 20 perceptronów

**Algorytm wstecznej propagacji błędów :**

metoda uczenia sieci wielowarstwowej, w której błąd ostatniej warstwy jest przesyłany wstecz i wykorzystywany do zmiany wartości wag w poprzednich warstwach.

Pierwszą czynnością w procesie uczenia jest przygotowanie dwóch ***ciągów*** danych: ***uczącego*** i ***weryfikującego***. ***Ciąg uczący*** jest to zbiór takich danych, które w miarę dokładnie charakteryzują dany problem. Jednorazowa porcja danych nazywana jest ***wektorem uczącym***. W jego skład wchodzi ***wektor wejściowy*** czyli te dane wejściowe, które podawane są na wejścia sieci i ***wektor wyjściowy*** czyli takie dane oczekiwane, jakie sieć powinna wygenerować na swoich wyjściach. Po przetworzeniu wektora wejściowego, nauczyciel porównuje wartości otrzymane z wartościami oczekiwanymi i informuje sieć czy odpowiedź jest poprawna, a jeżeli nie, to jaki powstał błąd odpowiedzi. Błąd ten jest następnie propagowany do sieci ale w odwrotnej niż wektor wejściowy kolejności (od warstwy wyjściowej do wejściowej) i na jego podstawie następuje taka korekcja wag w każdym neuronie, aby ponowne przetworzenie tego samego wektora wejściowego spowodowało zmniejszenie błędu odpowiedzi. Procedurę taką powtarza się do momentu wygenerowania przez sieć błędu mniejszego niż założony. Wtedy na wejście sieci podaje się kolejny wektor wejściowy i powtarza te czynności. Po przetworzeniu całego ***ciągu uczącego***(proces ten nazywany jest ***epoką***) oblicza się błąd dla ***epoki*** i cały cykl powtarzany jest do momentu, aż błąd ten spadnie poniżej dopuszczalnego. Jak to już było zasygnalizowane wcześniej, SSN wykazują tolerancję na nieciągłości, przypadkowe zaburzenia lub wręcz niewielkie braki w zbiorze uczącym. Jest to wynikiem właśnie zdolności do uogólniania wiedzy.

Jeżeli mamy już nauczoną sieć, musimy zweryfikować jej działanie. W tym momencie ważne jest podanie na wejście sieci wzorców z poza zbioru treningowego w celu zbadania czy sieć może efektywnie generalizować zadanie, którego się nauczyła. Do tego używamy ***ciągu weryfikującego***, który ma te same cechy co ***ciąg uczący*** tzn dane dokładnie charakteryzują problem i znamy dokładne odpowiedzi. Ważne jest jednak, aby dane te nie były używane uprzednio do uczenia. Dokonujemy zatem prezentacji ciągu weryfikującego z tą różnicą, że w tym procesie nie rzutujemy błędów wstecz a jedynie rejestrujemy ilość odpowiedzi poprawnych i na tej podstawie orzekamy, czy sieć spełnia nasze wymagania czyli jak została nauczona.

Wagi początkowe, z którymi sieć rozpoczyna naukę z reguły stanowią liczby wygenerowane przypadkowo. Po nauczeniu sieci zawsze warto dla sprawdzenia otrzymanych wyników powtórzyć całą procedurę od wygenerowania wag początkowych.

**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

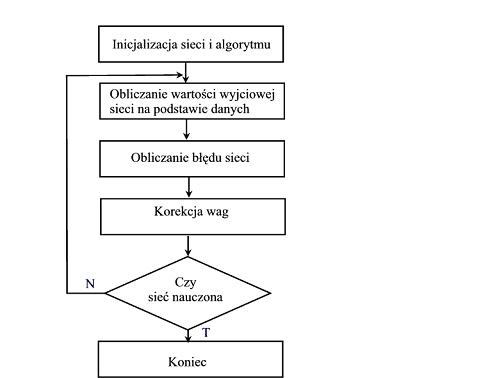
z1 ,..., zK – sygnały napływające do neuronu z poprzedniej warstwy.

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

**d(w1, ... , wk) = (f(w1z1+... + wkzk)-t)2**

Błąd sieci

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



***Rys. 2 Schemat blokowy algorytmu wstecznej propagacji błędu***

**Zestaw danych uczących**

Dane uczące składają się z tablicy 20 liczb reprezentujących 20 wielkich liter:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

new double[25] {1,1,1,1,1,1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1};

new double[25] {1,1,1,0,0,1,0,0,1,0,1,1,1,0,0,1,0,0,1,0,1,1,1,0,0},

new double[25] {1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1},

new double[25] {1,1,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,1,1,0,0},

new double[25] {1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1},

new double[25] {1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,0,0,0,0,1,0,0,0,0},

new double[25] {1,1,1,1,0,1,0,0,0,0,1,0,1,1,0,1,0,0,1,0,1,1,1,1,0},

new double[25] {1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1},

new double[25] {1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0},

new double[25] {1,1,1,0,0,0,0,1,0,0,0,0,1,0,0,1,0,1,0,0,1,1,1,0,0},

new double[25] {1,0,0,1,0,1,0,1,0,0,1,1,0,0,0,1,0,1,0,0,1,0,0,1,0},

new double[25] {1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0},

new double[25] {1,1,0,1,1,1,0,1,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1},

new double[25] {1,1,0,0,1,1,1,1,0,1,1,0,1,1,1,1,0,0,1,1,1,0,0,0,1},

new double[25] {1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1},

new double[25] {1,1,1,1,0,1,0,0,1,0,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0},

new double[25] {1,1,1,1,0,1,0,0,1,0,1,1,1,1,0,1,0,1,0,0,1,0,0,1,0},

new double[25] {1,1,1,0,0,1,0,0,0,0,1,1,1,0,0,0,0,1,0,0,1,1,1,0,0},

new double[25] {1,1,1,1,1,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0},

new double[25] {1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,1,1,1,0}

Dane testujące składają się z tablicy 25 liczb reprezentujących daną literkę:

new double[25] { 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0 }

**Uzyskane wyniki :**

Rozpoznano: M

Litera A wynik : 0,981 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

Litera B wynik : 0,000 1,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

Litera C wynik : 0,000 0,000 0,966 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

Litera D wynik : 0,000 0,000 0,000 1,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

Litera E wynik : 0,000 0,000 0,000 0,000 1,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

Litera F wynik : 0,000 0,000 0,000 0,000 0,000 0,967 0,000 0,000 0,000 0,000 0,000 0,000

0,000 0,000 0,000 0,080 0,000 0,000 0,000 0,000

Litera G wynik : 0,000 0,020 0,000 0,000 0,000 0,000 0,182 0,000 0,000 0,000 0,000 0,000

0,000 0,000 0,000 0,825 0,068 0,000 0,000 0,000

Litera H wynik : 0,051 0,000 0,000 0,000 0,000 0,000 0,000 1,000 0,000 0,000 0,000 0,000

0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

Litera I wynik : 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 1,000 0,000 0,000 0,000

0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

Litera J wynik : 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 1,000 0,000 0,000

0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

Litera K wynik : 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 1,000 0,000

0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

Litera L wynik : 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,142 0,000 0,000 0,000 0,912

0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

Litera M wynik : 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

0,951 0,000 0,000 0,000 0,000 0,000 0,000 0,000

Litera N wynik : 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

0,000 0,991 0,000 0,000 0,000 0,000 0,000 0,000

Litera O wynik : 0,000 0,000 0,076 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

0,000 0,000 1,000 0,000 0,000 0,000 0,000 0,000

Litera P wynik : 0,073 0,000 0,000 0,000 0,000 0,044 0,000 0,000 0,000 0,000 0,000 0,000

0,000 0,000 0,000 0,891 0,000 0,000 0,000 0,000

Litera R wynik : 0,000 0,000 0,000 0,000 0,000 0,000 0,240 0,000 0,000 0,000 0,000 0,000

0,000 0,000 0,000 0,000 0,599 0,000 0,000 0,000

Litera S wynik : 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

0,000 0,000 0,000 0,000 0,000 1,000 0,000 0,000

Litera T wynik : 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

0,000 0,000 0,000 0,000 0,000 0,000 1,000 0,000

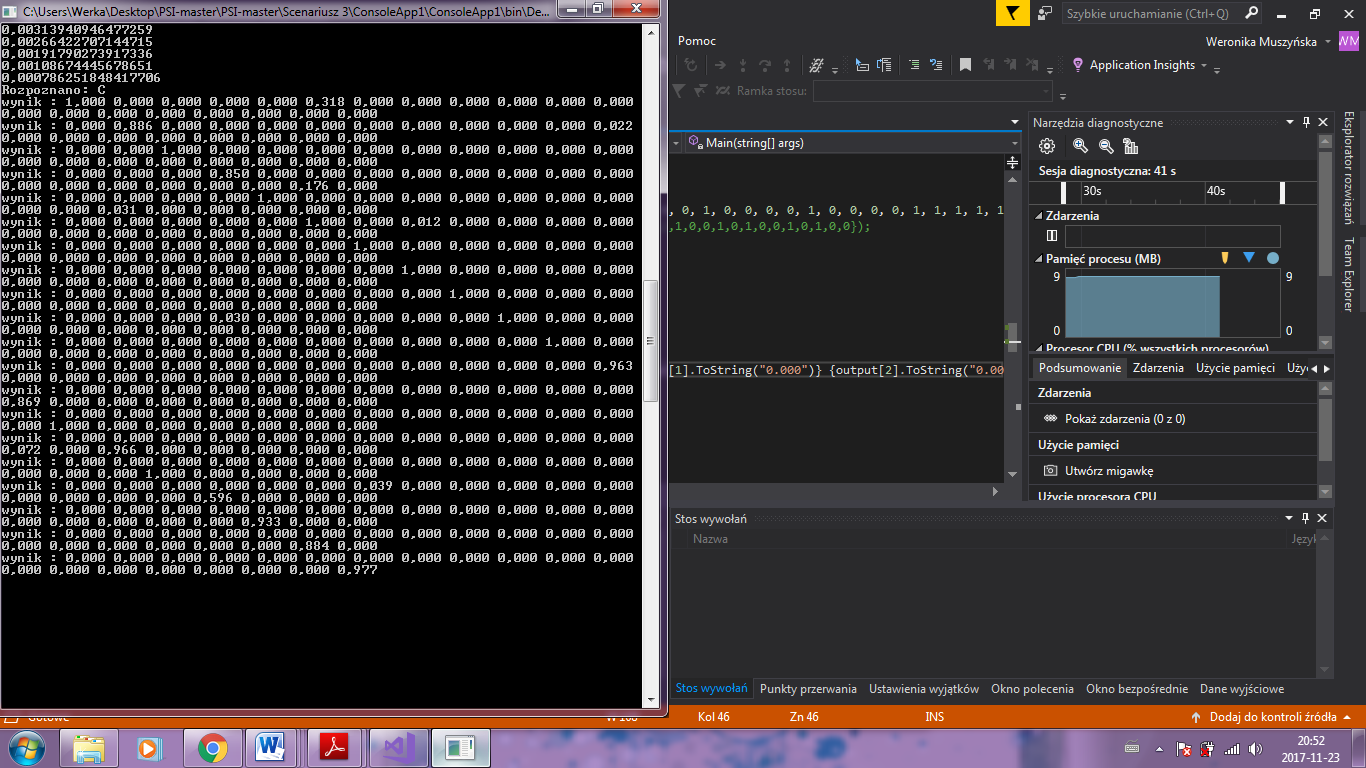
Litera U wynik : 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

0,000 0,000 0,000 0,000 0,000 0,000 0,000 1,000

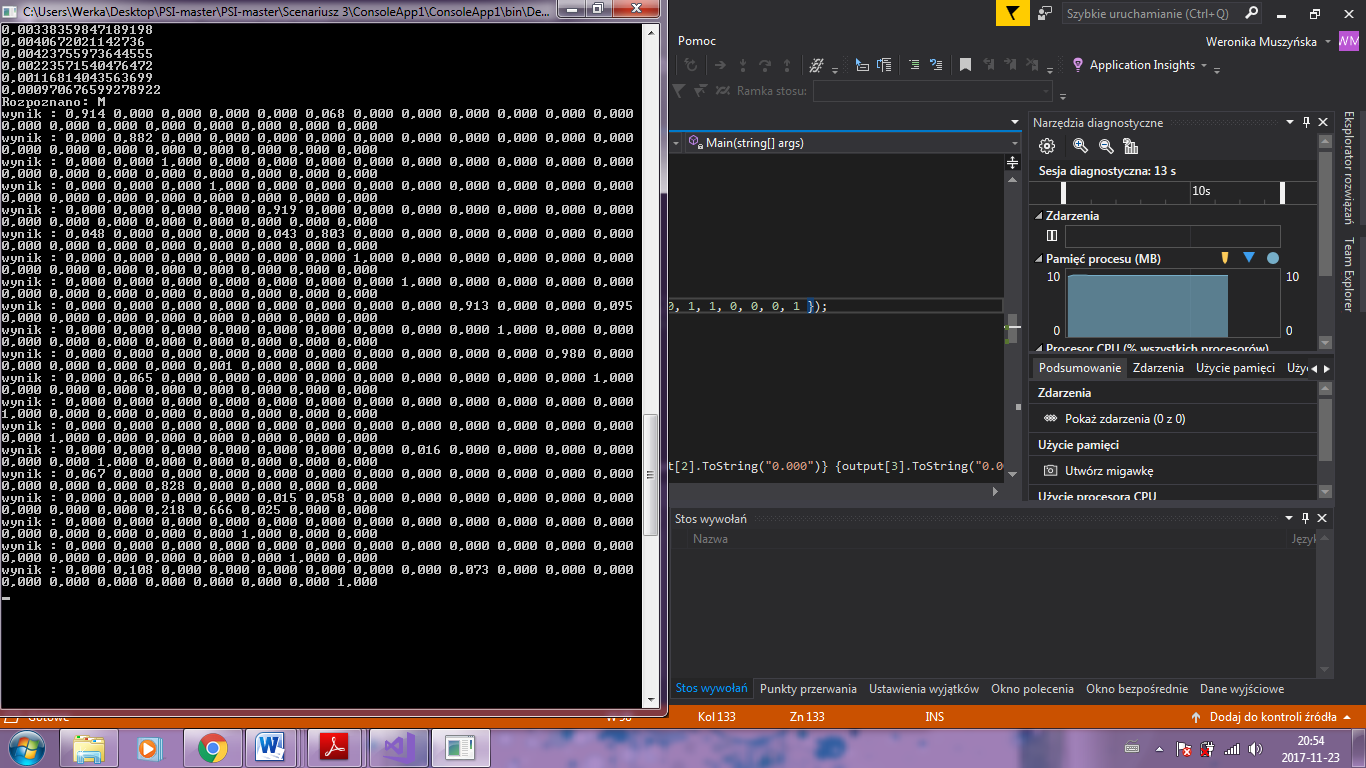
**Zestawienie wyników**

**Rys. 5 Zależność wielkości błędu uczenia od ilości iteracji**

**Przykładowe screeny pokazujące działanie programu oraz odczyt danych :**



***Rys. 3 Ilość danych uczących 20, litera C***



***Rys. 4 Ilość danych uczących 20, litera M***

**Wnioski:**

Im mniejszy współczynnik uczenia tym dokładniejszy wynik. Aby otrzymać wiarygodne dane musimy odpowiednio dobrać współczynniki uczenia a także bezwładności

Wraz ze zwiększaniem wartości współczynnika uczenia zmniejsza się liczba iteracji, co ma wpływ na działanie programu i powoduje uzyskanie błędnych wyników. Aby zredukować liczbę błędnych wyników można zwiększyć liczbę iteracji.

Dobór wag ma bezpośredni wpływ na działanie perceptronów w sieci, w zależności od ich wartości zmienia się poprawność wyników i liczba iteracji. Wartości wag mają największy wpływ na efekt końcowy, często wagi są ustalane losowo.

Dane uczące wpływają na poprawność uczenia perceptronu, przy zbyt małej ilości otrzymujemy błędne wyniki. Aby uzyskać lepsze wyniki powinniśmy dostarczyć wystarczającą liczbę danych wejściowych

Bardzo ważny jest odpowiedni dobór współczynnika uczenia, wag oraz liczby danych uczących, gdyż mają bezpośredni wpływ na działanie sieci.

Dla dużych sieci i ciągów uczących składających się z wielu tysięcy wektorów uczących ilość obliczeń wykonywanych podczas całego cyklu uczenia jest gigantyczna a więc i czasochłonna. Nie zdarza się także aby sieć została dobrze zbudowana od razu. Zawsze jest ona efektem wielu prób i błędów. Ponadto nigdy nie mamy gwarancji, że nawet prawidłowa sieć nie utknie w minimum lokalnym podczas gdy interesuje nas znalezienie minimum globalnego. Dlatego algorytmy realizujące ***SSN*** wyposaża się mechanizmy dające nauczycielowi możliwość regulacji szybkości i jakości uczenia. Są to tzw ***współczynniki***: ***uczenia*** i ***momentum***. Wpływają one na stromość funkcji aktywacji i regulują szybkość wpływu zmiany wag na proces uczenia.

Jednak zawsze istotnym jest zoptymalizowanie algorytmu przetwarzającego lub posiadanie szybkiego komputera, a najlepiej jest mieć te obie rzeczy na raz.

**Źródła:**

1. **S. Osowski. Sieci neuronowe w ujęciu algorytmicznym. WNT, Warszawa 1997.**
2. **J. Żurada, M. Barski, W. Jędruch. Sztuczne sieci neuronowe. PWN, Warszawa 1996.**
3. **R. Tadeusiewicz, Sieci neuronowe. Warszawa : Akademicka Oficyna Wydaw. RM, 1993.**

**Listing kodu :**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Encog;

using Encog.Engine.Network.Activation;

using Encog.ML.Data;

using Encog.ML.Data.Basic;

using Encog.Neural.Data.Basic;

using Encog.Neural.Networks;

using Encog.Neural.Networks.Layers;

using Encog.Neural.Networks.Training;

using Encog.Neural.Networks.Training.Propagation.Back;

using Encog.Neural.Networks.Training.Propagation.Resilient;

using Encog.Neural.NeuralData;

namespace ConsoleApp1

{

class Program

{

private static readonly double AcceptableError = 0.001;

private static readonly int MaxEpoch = 5000;

private static readonly string[] \_literki = { "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "R","S", "T", "U" };

private static readonly double[][] AndInput =

{

new double[25] {1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1},

new double[25] {1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0},

new double[25] {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1},

new double[25] {1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0},

new double[25] {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1},

new double[25] {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0},

new double[25] {1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0},

new double[25] {1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1},

new double[25] {1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0},

new double[25] {1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0},

new double[25] {1,0,0,1,0,1,0,1,0,0,1,1,0,0,0,1,0,1,0,0,1,0,0,1,0},

new double[25] {1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0},

new double[25] {1,1,0,1,1,1,0,1,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1},

new double[25] {1,1,0,0,1,1,1,1,0,1,1,0,1,1,1,1,0,0,1,1,1,0,0,0,1},

new double[25] {1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1},

new double[25] {1,1,1,1,0,1,0,0,1,0,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0},

new double[25] {1,1,1,1,0,1,0,0,1,0,1,1,1,1,0,1,0,1,0,0,1,0,0,1,0},

new double[25] {1,1,1,0,0,1,0,0,0,0,1,1,1,0,0,0,0,1,0,0,1,1,1,0,0},

new double[25] {1,1,1,1,1,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0},

new double[25] {1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,1,1,1,0}

};

private static readonly double[][] AndIdeal = {

new double[20] { 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

new double[20] { 0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

new double[20] { 0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

new double[20] { 0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

new double[20] { 0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

new double[20] { 0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

new double[20] { 0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0},

new double[20] { 0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0},

new double[20] { 0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0},

new double[20] { 0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0},

new double[20] { 0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0},

new double[20] { 0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0},

new double[20] { 0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0},

new double[20] { 0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

new double[20] { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0},

new double[20] { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0},

new double[20] { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0},

new double[20] { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0},

new double[20] { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0},

new double[20] { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1}

};

static void Main(string[] args)

{

INeuralDataSet trainingSet = new BasicNeuralDataSet(AndInput, AndIdeal);

var network = new BasicNetwork();

network.AddLayer(new BasicLayer(new ActivationRamp(), true, 25));

network.AddLayer(new BasicLayer(new ActivationRamp(), true, 75));

network.AddLayer(new BasicLayer(new ActivationRamp(), true, 50));

network.AddLayer(new BasicLayer(new ActivationRamp(), true, 20));

network.Structure.FinalizeStructure();

network.Reset();

ITrain train = new Backpropagation(network, trainingSet,0.02,0.3);

int epoch = 1;

do

{

train.Iteration();

Console.WriteLine($"{train.Error}");

epoch++;

} while ((epoch < MaxEpoch) && (train.Error > AcceptableError));

var input = new BasicMLData(new double[25] { 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1 });

int best = network.Winner(input);

Console.WriteLine($"Rozpoznano: {\_literki[best]}" );

foreach (IMLDataPair pair in trainingSet)

{

IMLData output = network.Compute(pair.Input);

Console.WriteLine($"wynik : { output[0].ToString("0.000")} {output[1].ToString("0.000")} {output[2].ToString("0.000")} {output[3].ToString("0.000")} {output[4].ToString("0.000")} {output[5].ToString("0.000")} {output[6].ToString("0.000")} {output[7].ToString("0.000")} {output[8].ToString("0.000")} {output[9].ToString("0.000")} {output[10].ToString("0.000")} {output[11].ToString("0.000")} {output[12].ToString("0.000")} {output[13].ToString("0.000")} {output[14].ToString("0.000")} {output[15].ToString("0.000")} {output[16].ToString("0.000")} {output[17].ToString("0.000")} {output[18].ToString("0.000")} {output[19].ToString("0.000")}");

}

Console.ReadKey();

}

public static double[] findMaximumIndex(double[][] a)

{

double maxVal = -99999;

double[] answerArray = new double[2];

for (int row = 0; row < a.Length; row++)

{

for (int col = 0; col < a[row].Length; col++)

{

if (a[row][col] > maxVal)

{

maxVal = a[row][col];

answerArray[0] = row;

answerArray[1] = col;

}

}

}

return answerArray;

}

}

}