Aleksandra Spiecha

Inżynieria Obliczeniowa

Podstawy sztucznej inteligencji

**Sprawozdanie nr 3**

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

Sieci jednokierunkowe których typowym przykładem jest perceptron jednowarstwowy (rys. perceptronu) składają się z neuronów ułożonych w warstwach o jednym kierunku przepływu sygnałów i połączeniach między warstwowych jedynie między kolejnymi warstwami. Sieć tego typu posiada warstwę wejściową, wyjściową i warstwy ukryte. Z funkcjonalnego punktu widzenia układ taki można traktować jako układ aproksymacji funkcji nieliniowej wielu zmiennych y = f(u).

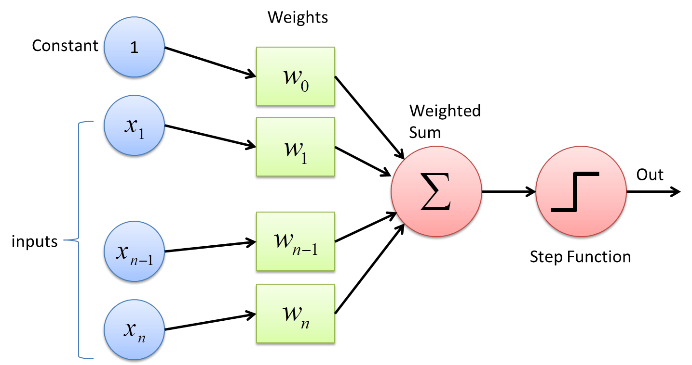


Figure 1 Perceptron

W moim projekcie użyłam sieci jednokierunkowej wielowarstwowej, zgodnie z założeniami.

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

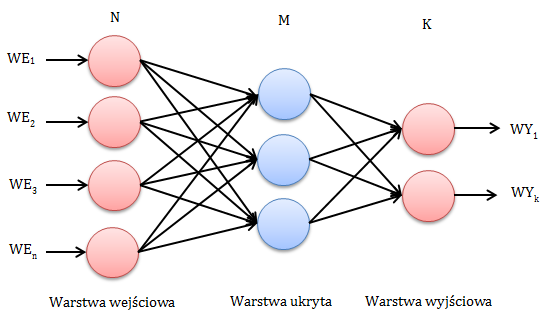


Figure 2 Przykład ilustrujący jednowarstwową jednokierunkową sieć neuronową

Sztuczny neuron to prosty system przetwarzający wartości sygnałów wprowadzanych na jego wejścia w pojedynczą wartość wyjściową, wysyłaną na jego jedynym wyjściu. To podstawowy element sieci neuronowych, która jest jedną z metod sztucznej inteligencji. Perceptron to prosty element obliczeniowy. Sumuje ważone sygnały wejściowe oraz porównuje ją z progiem aktywacji. Zależnie od wyniku perceptron może być wzbudzony albo nie. Do uczenia perceptronu można użyć algorytmu uczenia perceptronu, czyli automatycznego doboru wag na podstawie kolejnych przykładów.

Do uczenia perceptronu użyłam algorytmu RPROP (od ang. Resilient backPROPagation). Jest to algorytm przeznaczony dla pełnego (wsadowego) trybu korekcji parametrów (wag). Oznacza to, że jedno skorygowanie parametrów (wag) następuje dopiero po przeglądnięciu przez sieć całego zbioru uczącego i obliczeniu sumarycznego a tym samym dokładnego gradientu. Kluczowymi elementami algorytmu RPROP są: wykorzystywanie jedynie samego znaku każdej składowej gradientu (natomiast wartości są pomijane), a także modyfikowanie współczynnika (współczynników) uczenia w każdym kroku. Współczynnik uczenia jest zwiększany, gdy znaki kolejnych gradientów pozostają zgodne, natomiast zmniejszany (a dokładnie połowiony), gdy są różne.

Ogólny schemat procesu trenowania sieci wygląda następująco:

* Ustalamy topologię sieci, tzn. liczbę warstw, liczbę neuronów w warstwach.
* Inicjujemy wagi losowo (na małe wartości).
* Dla danego wektora uczącego obliczamy 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 (również w warstwach ukrytych) modyfikuje wagi na podstawie wartości błędu i wielkosci przetwarzanych w tym kroku sygnałów.
* Powtarzamy od punktu 3. dla kolejnych wektorów uczących. Gdy wszystkie wektory zostaną użyte, losowo zmieniamy ich kolejność i zaczynamy wykorzystywać powtórnie.
* 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.

- W programie użyłam funkcję Ramp jako funkcję aktywacji

- każda litera reprezentowana jest jako tablica zawierająca 5x5 liczb reprezentującą piksele  
 - dane uczące składają się z tablicy 20 liczb reprezentujących 20 pierwszych wielkich liter alfabetu angielskiego:

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 }

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

Wyniki:

Wyniki uczenia:

A: 0,632 0,000 0,000 0,000 0,000 0,291 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

B: 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

C: 0,000 0,000 0,983 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,010 0,000 0,000 0,000 0,000 0,000

D: 0,000 0,000 0,000 1,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,030 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

E: 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

F: 0,356 0,000 0,000 0,000 0,000 0,663 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

G: 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 0,000

H: 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,917 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

I: 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,978 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

J: 0,000 0,000 0,000 0,000 0,000 0,062 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

K: 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

L: 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,952 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000

M: 0,000 0,000 0,000 0,000 0,036 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

N: 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 0,000 0,000 0,000 0,000

O: 0,000 0,000 0,017 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,000 0,942 0,004 0,000 0,000 0,000 0,000

P: 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 0,000 0,000

R: 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 0,000

S: 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

T: 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

U: 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

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.
* Zastosowanie sieci wielowarstwowych umożliwia uzyskanie dokładniejszych wyników w krótszym czasie obliczeń.

Listing kodu napisanego w języku C#:

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);

//ITrain train = new Backpropagation(network, trainingSet);

int epoch = 1;

do

{

train.Iteration();

//Console.WriteLine($"Epoch no {epoch}. Error: {train.Error}");

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

epoch++;

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

//sprawdzenie

var input = new BasicMLData(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 });

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

int best = network.Winner(input);

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

foreach (IMLDataPair pair in trainingSet)

{

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

Console.WriteLine($"values: {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")}");

}

//var wyniki = new double[20][];

//int index = 0;

//Console.WriteLine("\nResults:");

//foreach (IMLDataPair pair in trainingSet)

//{

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

// wyniki[index] = new double[20];

// for (int i = 0; i < output.Count - 1; i++)

// {

// if (output[i] <= 1)

// wyniki[index][i] = output[i];

// else

// {

// wyniki[index][i] = 0;

// }

// }

// index += 1;

// //Console.WriteLine($" actual value is: {output[0]} {output[1]} {output[2]}");

//}

//for (int i = 0; i < wyniki.Length; i++)

//{

// for (int j = 0; j < 20; j++)

// {

// Console.Write(wyniki[i][j].ToString("0.000") + " ");

// }

// Console.WriteLine();

//}

//double[] wyn = findMaximumIndex(wyniki);

//Console.WriteLine("Rozpoznano: " + \_literki[(int)wyn[0]]);

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;

}

}

}