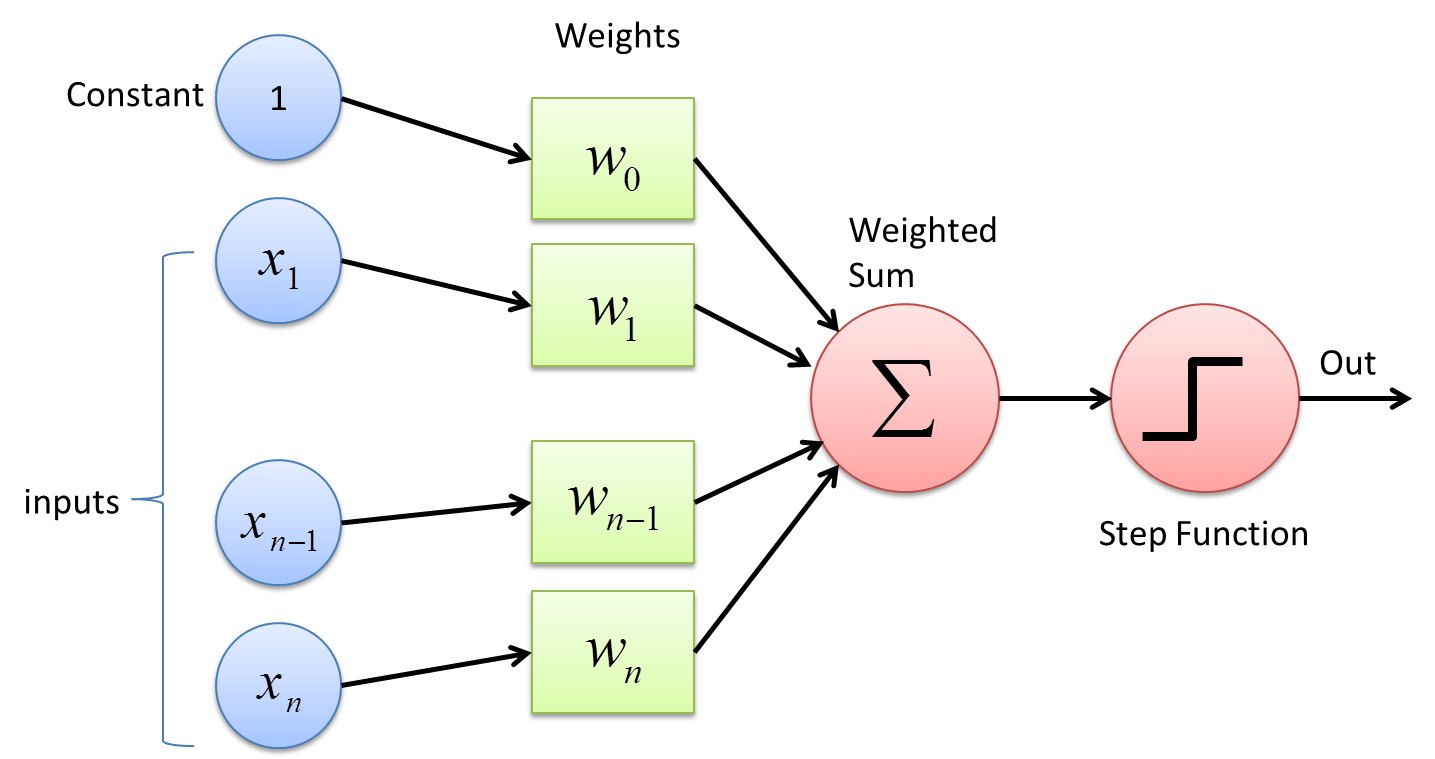
Piotr Smuga gr 2

Podstawy sztucznej inteligencji

Sprawozdanie nr 1

Cel: Poznanie budowy i działanie perceptronu poprzez implementację oraz uczenie perceptronu realizującego wybraną funkcję logiczną dwóch zmiennych.

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. Pierwowzorem zbudowania sztucznego neuronu był biologiczny neuron.



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. W uproszczonym przypadku algorytm wygląda następująco:

1. Inicjujemy wagi losowo.
2. Dla każdego przykładu uczącego obliczamy odpowiedź perceptronu.
3. Jeśli odpowiedź perceptronu jest nieprawidłowa, to modyfikujemy wagi:

w1 += n \* (d-y) \* x1

w2 += n \* (d-y) \* x2

b += n \* (d-y)

gdzie:

n - niewielki współczynnik uczenia (n > 0),

d - oczekiwana odpowiedź,

y - odpowiedź neuronu.

Po wykorzystaniu wszystkich przykładów rozpoczyna się proces uczenia na nowo, do momentu gdy następują jakiekolwiek zmiany wag połączeń.

Do uczenia perceptronu użyłem algorytm 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.  
  
- Moja sztuczna sieć neuronowa, realizuje funkcję logiczną „And”  
- Przy tworzeniu mojego perceptronu wykorzystałem bibliotekę Encog C# przeznaczoną do tworzenia sztucznych sieci neuronowych. <http://www.heatonresearch.com/encog/>

- wagi są ustalane poprzez losowanie   
- W programie użyłem funkcję sigmoidalną jako funkcję aktywacji.  
 - dane uczące składają się z czterech par liczb:

private static readonly double[][] AndInput ={

new double[2] { 0.0, 0.0 },

new double[2] { 1.0, 0.0 },

new double[2] { 0.0, 1.0 },

new double[2] { 1.0, 1.0 } };

- dane testujące składają się z czterech liczb:

private static readonly double[][] AndIdeal = {

new double[1] { 1.0 },

new double[1] { 0.0 },

new double[1] { 0.0 },

new double[1] { 1.0 } };

Wyniki:  
Przy epoch: 94 oraz błędzie mniejszym od 0,001  
0 AND 0 powinno być: 1 aktualna wartość: 0,973352417369545  
1 AND 0 powinno być: 0 aktualna wartość: 0,0330859756133677  
0 AND 1 powinno być: 0 aktualna wartość: 0,030818130296159  
1 AND 1 powinno być: 1 aktualna wartość: 0,969405155368819

Przy epoch: 50 oraz błędzie mniejszym od 0,0001  
0 AND 0 powinno być: 1 aktualna wartość: 0,993834096589738  
1 AND 0 powinno być: 0 aktualna wartość: 0,0147131510740629  
0 AND 1 powinno być: 0 aktualna wartość: 0,0118925875883704  
1 AND 1 powinno być: 1 aktualna wartość: 0,996079986390473

Wnioski:

* z każdą następną iteracją, błąd maleje na skutek uczenia perceptronu.
* dokładność nie jest uzależniona od liczby iteracji
* nie udało mi się nauczyć perceptron, aby działał bez dozwolonego małego błędu, tzn. wartość wyjściowa nigdy nie osiągnie 1.0 lub 0.0;
* dobór wag ma bezpośredni wpływ na działanie perceptronu, w zależności od ich wartości zmienia się poprawność wyników i liczba iteracji
* działanie perceptronu jest uzależnione od kilku czynników: współczynnika uczenia, wag oraz liczby danych uczących,
* dane uczące wpływają na poprawność uczenia perceptronu, przy zbyt małej ilości otrzymujemy błędne wyniki,
* zbyt duża liczba danych uczących powoduje, iż uzyskane wyniki są błędne,
* bardzo ważny jest odpowiedni dobór współczynnika uczenia, wag oraz liczby danych uczących, gdyż mają bezpośredni wpływ na działanie perceptronu.

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.Neural.Data.Basic;

using Encog.Neural.Networks;

using Encog.Neural.Networks.Layers;

using Encog.Neural.Networks.Training;

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 double[][] AndInput ={

new double[2] { 0.0, 0.0 },

new double[2] { 1.0, 0.0 },

new double[2] { 0.0, 1.0 },

new double[2] { 1.0, 1.0 } };

private static readonly double[][] AndIdeal = {

new double[1] { 1.0 },

new double[1] { 0.0 },

new double[1] { 0.0 },

new double[1] { 1.0 } };

static void Main(string[] args)

{

INeuralDataSet trainingSet = new BasicNeuralDataSet(AndInput, AndIdeal);

var network = new BasicNetwork();

network.AddLayer(new BasicLayer(new ActivationSigmoid(), true, 2));

network.AddLayer(new BasicLayer(new ActivationSigmoid(), true, 2));

network.AddLayer(new BasicLayer(new ActivationSigmoid(), true, 1));

network.Structure.FinalizeStructure();

network.Reset();

ITrain train = new ResilientPropagation(network, trainingSet);

int epoch = 1;

do

{

train.Iteration();

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

epoch++;

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

Console.WriteLine("\nAnd function Results:");

foreach (IMLDataPair pair in trainingSet)

{

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

Console.WriteLine($"{pair.Input[0]} AND {pair.Input[1]} should be: {pair.Ideal[0]} actual value is: {output[0]}");

}

Console.ReadKey();

}

}

}