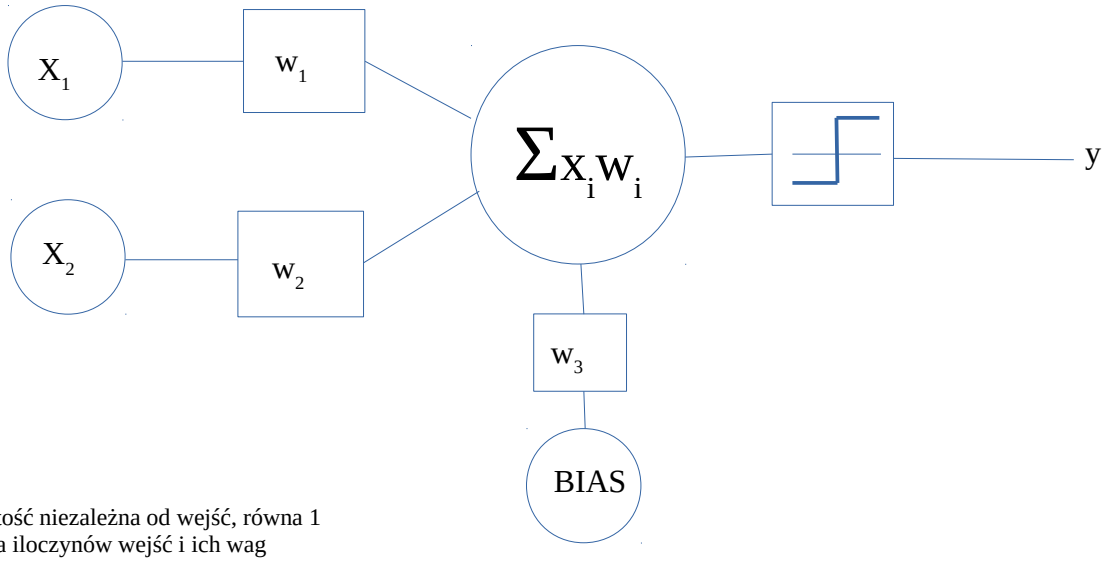
Imię Nazwisko Grupa Michał Słowikowski gr 4	Temat **Scenariusz I**	Data 20.10.2017.r

Do ćwiczenia wykorzystałem perceptron (jedną warstwę)

Budowa Perceptronu



Gdzie:
x_i – wejścia
w_i – wagi
BIAS – wartość niezależna od wejść, równa 1
$\Sigma x_i w_i$ – suma iloczynów wejść i ich wag

 - funkcja aktywacji (dla sumy >= 0 przyjmuje 1, dla sumy < 0 przyjmuje 0)

Do nauki perceptronu wykorzystałem regułę perceptronu poznaną na wykładzie:

$w_{i+1} = w_i + \Delta w$

$\Delta w = \eta * (y - y') * x_i$

Gdzie:
w_{i+1} to kolejna waga
y – oczekiwany wynik
y' - uzyskany wynik
η – współczynnik uczenia

Algorytm był wykonywany aż do momentu gdy suma |(y-y')| dla wszystkich zestawów danych była równa zero, lub gdy ilość wykonanych treningów przekroczyła z góry ustaloną wartość.

Pomiary:
W ćwiczeniu wykonałem próby nauczenia perceptronu następujących bramek (X oznacza brak danej wejściowej):

- AND
- AND(00X1)
- AND(0XX1)
- OR
- OR(0X11)
- OR(X111)
- XOR

  Do każdej bramki wykonałem 10 pomiarów, które różniły się wartością współczynnika uczenia. Przyjmował on wartości od 0.000001 do 10000 (z każdą iteracją był zwiększany dziesięciokrotnie).

AND

η	Ilość iteracji
0.000001	416743
0.00001	11210
0.0001	4967
0.001	435
0.01	45
0.1	6
1	8
10	6
100	6
1000	6

AND0XX1

η	Ilość iteracji
0.000001	20839 (fail)
0.00001	54192 (fail)
0.0001	4366
0.001	622 (fail)
0.01	64 (fail)
0.1	8 (fail)
1	2 (fail)
10	3 (fail)
100	3 (fail)
1000	3 (fail)

AND00X1

η	Ilość iteracji
0.000001	97937 (fail)
0.00001	39969
0.0001	4420
0.001	408
0.01	42
0.1	5
1	2
10	4
100	4 (fail)
1000	4 (fail)

OR

η	Ilość iteracji
0.000001	620272
0.00001	96080
0.0001	181
0.001	19
0.01	3
0.1	2
1	4
10	4
100	4
1000	4

ORX111

η	Ilość iteracji
0.000001	1 (fail)
0.00001	1 (fail)
0.0001	1 (fail)
0.001	1 (fail)
0.01	1 (fail)
0.1	1 (fail)
1	1 (fail)
10	1 (fail)
100	1 (fail)
1000	1 (fail)

OR0X11

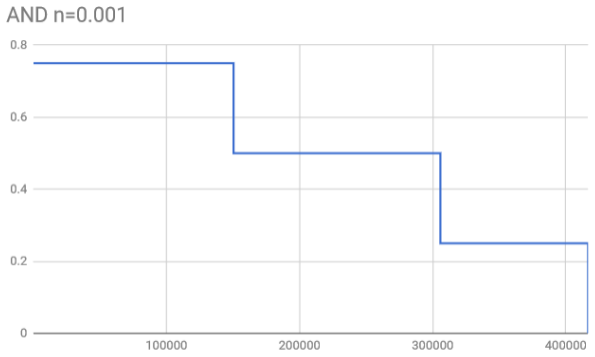
η	Ilość iteracji
0.000001	201784
0.00001	59953
0.0001	3525
0.001	538
0.01	55
0.1	7
1	3
10	3
100	3
1000	3

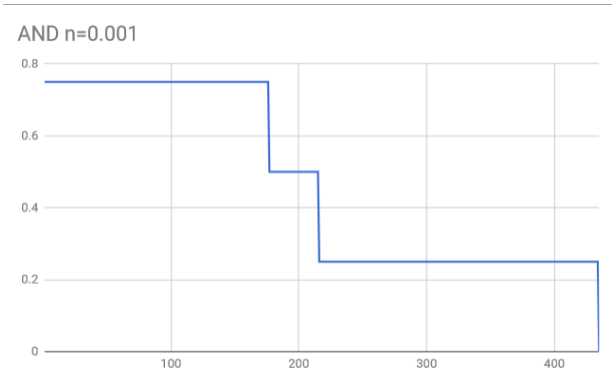
XOR

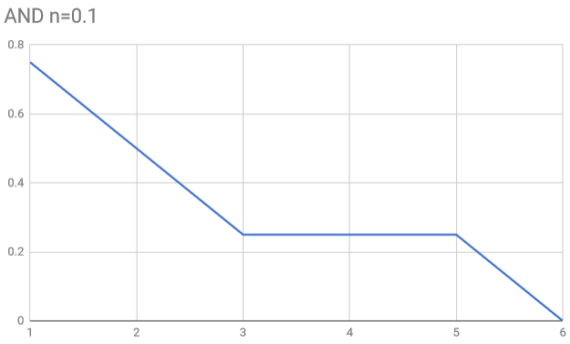
η	Ilość iteracji
0.000001	1000000 (fail)
0.00001	1000000 (fail)
0.0001	1000000 (fail)
0.001	1000000 (fail)
0.01	1000000 (fail)
0.1	1000000 (fail)
1	1000000 (fail)
10	1000000 (fail)
100	1000000 (fail)
1000	1000000 (fail)

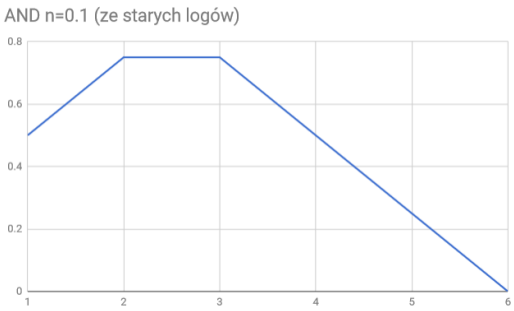
Na podstawie otrzymanych wyników możemy zaobserwować jaki wpływ miał dobór kroku na ilość iteracji, zanim perceptron nauczył się danej bramki logicznej (o ile udało mu się to zrobić). Możemy zauważyć, że bramki AND oraz OR, które miały pełen zestaw danych, posiadają 100% poziom sukcesu nauki. Bramki, w których brakowało pewnych danych potrafiły nauczyć się danego zadania, jednak nie zawsze (podczas nauki zgłaszały gotowość, jednak podczas testów wyniki brakujących danych były błędne). Możemy również zobaczyć, że zgodnie z oczekiwaniami bramka XOR nie została nauczona (jest to niemożliwe przy udziale jednego perceptronu), wyczerpała limit iteracji. Z tabel możemy również zauważyć, że optymalny krok iteracji dla tych bramek logicznych i reguły perceptronu wynosi od 0.01 do 1.

Porównanie błędów na wykresach uzyskanych z bramki AND









Ilość błędów była liczona na podstawie jednej epoki danych (ilość błędnych wyników/całkowitą liczbę danych). Jak można zauważyć bardzo duży wpływ na uzyskane wyniki (a w szczególności na czas trwania nauki) ma rozmiar dobranego kroku. Zbyt mały krok powoduje, że zanim zbliżymy się do oczekiwanego wyniku będziemy musieli wykonać wiele iteracji. Z drugiej strony Zbyt duży krok może powodować efekt „przestrzelenia” (widoczny na 4 wykresie, ilość błędów wzrosła zamiast zmaleć).

Wnioski:

- Możliwe jest nauczenie perceptronu prostych bramek logicznych
- Bramka logiczna AND, przy mniejszych krokach, średnio uczyła się szybciej od bramki OR
- Próby w których usunięto część danych wejściowych nie zawsze były nauczone poprawnie wykonywać zadanie
- Dobór wielkości kroku jest najważniejszym czynnikiem wpływającym na długość uczenia się
- Można zauważyć, że zbyt mały krok powoduje znaczne wydłużenie czasu nauki
- Nie jest możliwe zbudowanie bramki XOR za pomocą jednego perceptronu (XOR nie jest rozłączny liniowo)

```
namespace Scen1
{
    class Perceptron
    {
        private double[] weights;
        private double bias = 1;
        private double treshold = 0;

        public Perceptron(double[] weights)
        {
            this.weights = weights;
        }
        public double[] getWeights()
        {
            return weights;
        }
        public void setWeights(double[] weights)
        {
            this.weights = weights;
        }
        public int getResult(int[] input)
        {
            double sum = inputSummary(input);
            return perceptronActivation(sum);
        }
        public double inputSummary(int[] input)
        {
            double sum = 0;
            for (int i = 0; i < input.Length; i++)
            {
                sum += weights[i] * input[i];
            }

            return sum + weights[input.Length] * bias;
        }
        private int perceptronActivation(double sum)
        {
            int active = 1;
            int inactive = 0;

            if (sum >= treshold)
            {
                return active;
            }
            else
            {
                return inactive;
            }
        }
    }
}
```

```
using System;

namespace Scen1
{
    class Trainer
    {

        private int[][] inputDataSets;
        private int[] expectedOutputs;
        private double learningRate;
        private const double maxNumberOfSeasions = 1000000;

        public Trainer(int[][] inputDataSets, int[] expectedOutputs, double learningRate)
        {
            this.inputDataSets = inputDataSets;
            this.expectedOutputs = expectedOutputs;
            this.learningRate = learningRate;
        }

        public bool train(ref Perceptron perceptronToTrain)
        {
            bool passed = false;
            int tryNumber = 0;

            do
            {
                if (perceptronRule(ref perceptronToTrain) == 0)
                {
                    passed = true;
                }

                tryNumber++;

            } while (!passed && tryNumber < maxNumberOfSeasions);
            Console.WriteLine("After: " + tryNumber + " generations");
            return passed;
        }

        private double perceptronRule(ref Perceptron perceptron)
        {
            int error;
            int biasID = inputDataSets[0].Length;
            double weightModyfier = 0.0;
            double totalError = 0;
            double[] weights = perceptron.getWeights();

            for (int i = 0; i < expectedOutputs.Length; i++)
            {
                error = expectedOutputs[i] - perceptron.getResult(inputDataSets[i]);
                for (int j = 0; j < inputDataSets[0].Length; j++)
                {
                    weightModyfier = learningRate * error * inputDataSets[i][j];
                    weights[j] += weightModyfier;
                    totalError += Math.Abs(weightModyfier);
                }
                weightModyfier = learningRate * error;
                weights[biasID] += weightModyfier;
                totalError += Math.Abs(weightModyfier);
            }

            perceptron.setWeights(weights);
            printDifferenceBetweenTargetAndPerceptronOutput(perceptron);
            return totalError;
        }

        private void printDifferenceBetweenTargetAndPerceptronOutput(Perceptron perceptronToTrain)
        {
            for (int i = 0; i < expectedOutputs.Length; i++)
            {
                double inputSummary = perceptronToTrain.inputSummary(inputDataSets[i]);
                int afterActivation = perceptronToTrain.getResult(inputDataSets[i]);
                Console.Write("In:");
                foreach (int input in inputDataSets[i])
                {
                    Console.Write(input);
                }
                Console.WriteLine("\tExp:\t" + expectedOutputs[i] +
                    "\tActv:\t" + afterActivation +
                    "\tGot:\t" + inputSummary +
                    "\tErr:\t" + Math.Abs(((double)expectedOutputs[i] - inputSummary)));
            }
        }
    }
}
```