Michał Radziejowski – Scenariusz 1 SPRAWOZDANIE

Budowa i działanie perceptronu

Cele ćwiczenia.

Głównym celem zadania było poznanie budowy i działania perceptronu poprzez implementację oraz uczenie perceptronu realizującego wybraną funkcję logiczną dwóch zmiennych.

Algorytm uczenia perceptronu:

- Zadeklarowanie współczynnika uczenia, liczby danych, oraz threshold;

- Wprowadzenie danych wejściowych oraz wyjściowych (w tym przypadku dane zgodne z bramką logiczną AND);

- Inicjalizacja wag liczbami rzeczywistymi (w tym przypadku zakres -1 do 1)

- Rozpoczęcie cyklu poniższych instrukcji do momentu uzyskania satysfakcjonującego wyniku (nauczenie perceptronu):

    - Wykonanie tak zwanych threshold computations. Można je sprowadzić do wyznaczania sumy iloczynu skalarnego wi*xi i sprawdzenie nierówności w*x >= θ;

    - Wyznaczenie błędu lokalnego, globalnego oraz średniej kwadratowej błędów, czyli średniej różnicy pomiędzy estymatorem i wartością estymowaną;

    - Przypisanie nowych wag, poprzez dodanie iloczynu błędu lokalnego, danych wejściowych oraz współczynnika uczenia.

    - Skonfrontowanie wyniku z wartościami oczekiwanymi, bądź sprawdzenie czy nie skończyły się dane uczące.

- Po poprawnym nauczeniu (root mean square error = 0), sprawdzenie na 20 kolejnych ‘strzałach’ poprawności działania perceptronu.

Przykłady działania

```
=======
New Random Point:
x = 0,y = 1
Works? = 0
Iteration 6 : RMSE = 0.7071067811865476

=======
New Random Point:
x = 1,y = 0
Works? = 1
Iteration 7 : RMSE = 0.7071067811865476

=======
New Random Point:
x = 1,y = 1
Works? = 0
Iteration 8 : RMSE = 0.7071067811865476
```

‘Strzały’ losowymi wartościami w trakcie nauki perceptronu. Wysoka wartość średniej kwadratowej błędów.

```
Iteration 10 : RMSE = 0.7071067811865476

=======
New Random Point:
x = 1,y = 1
Works? = 1
Iteration 11 : RMSE = 0.5

=======
New Random Point:
x = 0,y = 1
Works? = 0
---------------------------NAUCZONY---------
Iteration 80 : RMSE = 0.0

=======
New Random Point:
x = 1,y = 0
Works? = 0
Iteration 81 : RMSE = 0.0
```

Widzimy jak wartość średniej kwadratowej błędów maleje, aż w końcu jest równa 0. Jest to oznaką że perceptron został nauczony (automatyczny przeskok do iteracji MAX – 20 aby przeprowadzić 20 testów kontrolnych).

```
=======
New Random Point:
x = 1,y = 1
Works? = 1
Iteration 89 : RMSE = 0.0

=======
New Random Point:
x = 0,y = 0
Works? = 0
Iteration 90 : RMSE = 0.0

=======
New Random Point:
x = 0,y = 0
Works? = 0
Iteration 91 : RMSE = 0.0

=======
New Random Point:
x = 1,y = 1
Works? = 1
Iteration 92 : RMSE = 0.0
```

W czasie 20 prób kontrolnych, widzimy iż perceptron poprawnie rozpoznaje dane bramki logicznej AND. Średnia kwadratowa błędów wynosi przez cały czas 0.

Wyniki

W czasie użytkowania programu, niezmienne pozostały dane wejściowe, potrzebne do przekazania informacji perceptronowi na temat bramki AND. Zakresy wag zostały ustawione na zakresach od -1 do 1.

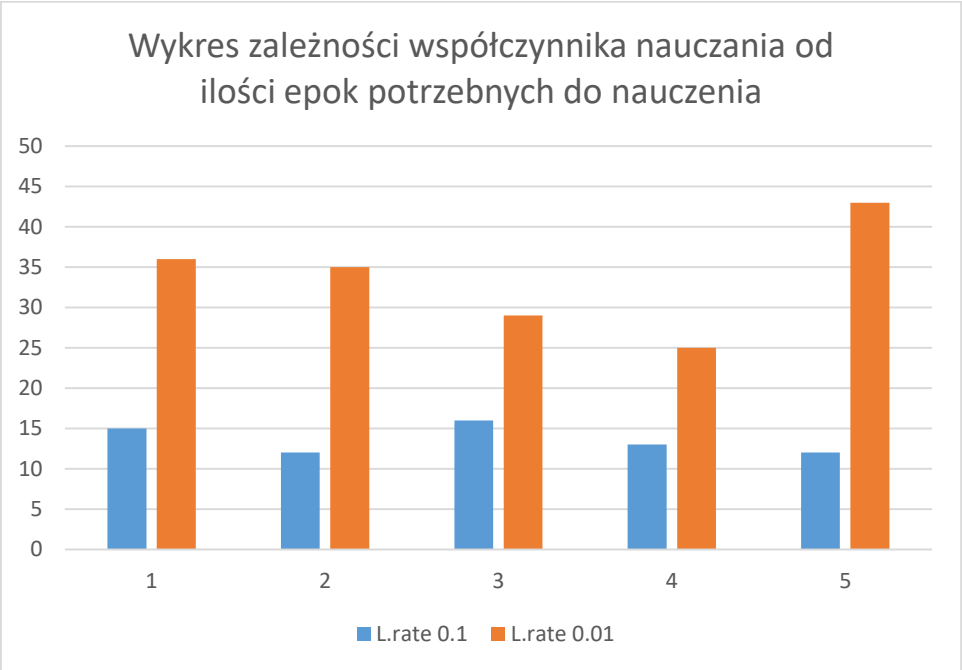
```
x[0] = 0; y[0] = 0;
x[1] = 1; y[1] = 0;
x[2] = 0; y[2] = 1;
x[3] = 1; y[3] = 1;

outputs[0] = 0;
outputs[1] = 0;
outputs[2] = 0;
outputs[3] = 1;

weights[0] = randomNumber(-1, 1);
weights[1] = randomNumber(-1, 1);
weights[2] = randomNumber(-1, 1);
```

W zagadnieniu użyłem unipolarnej funkcji progowej (z zadaną początkową wartością a = 0.) gdyż jej działanie dobrze współpracuje z tematem pracy:

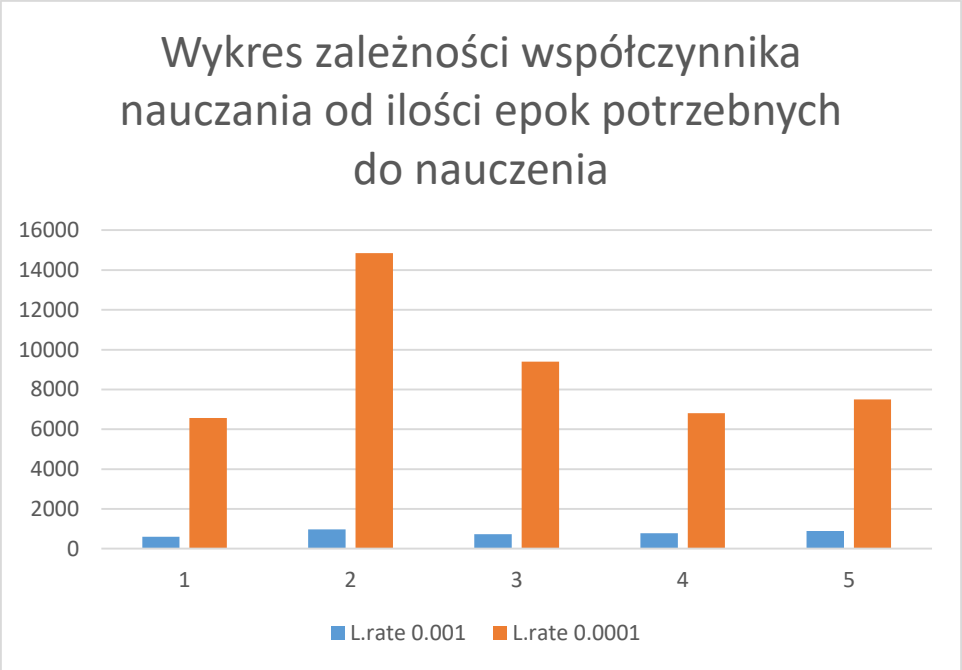
$$y(x) = \begin{cases} 0 & dla \quad x < a \\ 1 & dla \quad x \geq a \end{cases}$$

W zależności od przeprowadzonego testu, zmieniano współczynnik uczenia. Każdy test zawierał 5 prób na podstawie których powstały wykresy porównawcze zależności wspomnianego wcześniej współczynnika uczenia od ilości epok po których perceptron nauczył się poprawnych odpowiedzi na przedstawione zagadnienie bramki logicznej.

Learn Rate	Epoki zadeklarowane	L.P. pomia	1	2	3	4	5
0.1	100		15	12	16	13	12
0.01	100		36	35	29	25	43
0.001	15000		603	978	735	778	898
0.0001	15000		6563	14853	9393	6803	7494



Analizując wstępnie wykres, widzimy iż dziesięciokrotne zmniejszenie współczynnika nauki w programie, znacznie (około dwa-trzy razy) zwiększa ilość epok wymaganych do otrzymania poprawnego wyniku logicznego.



Powyższy wykres ukazuje nam iż dalsze zmniejszanie współczynnika nauczania, w niektórych przypadkach nawet kilkunastokrotnie zwiększa ilość epok potrzebnych do nauczenia perceptronu.

Analiza oraz wnioski:

Przy korzystaniu z narzędzia jakim jest perceptron, musimy bardzo uważnie dobierać wartość epok, których ilość w zależności od współczynnika nauki może mieć bardzo duże wahania. Gdy zada się ich zbyt małą ilość, program nie osiągnie odpowiednio niskiego błędu umożliwiającego poprawne przewidywanie wyników.

Kolejnym wartym omówienia kryterium, jest wcześniej wspomniany współczynnik nauki. Jego zmiana posiada znaczący wpływ na ilość epok(iteracji) potrzebnych do skutecznego nauczenia sztucznego neuronu odpowiedniej reakcji na dane wejściowe.

Przedstawiony w zagadnieniu perceptron jednowarstwowy (single layer perceptron) jest bardzo skutecznym narzędziem do klasyfikowania danych na zbiory liniowo separowalne. Oznacza to, że niemożliwym jest przykładowo stworzenie jednowarstwowego perceptronu który nauczyłby się wykonywać logiczną operację XOR na zadanych wartościach wejść. Do takich przypadków, należy zbudować sieć składającą się z więcej niż jednego neurona.

Dzięki temu ćwiczeniu zaznajomiłem się z podstawowym zagadnieniem jakim jest Neuron McCullocha-Pittsa, oraz najprostszą siecią neuronową jaką jest perceptron. Umiejętność wykorzystania tej wiedzy w praktyce będzie z pewnością pomocna, przy kolejnych zagadnieniach związanych ze sztuczną inteligencją.

Listing kodu

```
public class perceptron {

    static int MAX_ITER = 15000;
    static double LEARNING_RATE = 0.0001;
    static int NUM_INSTANCES = 4;
    static int theta = 0;

    public static void main(String[] args){
        Random randGenerator = new Random();

        //three variables (features)
        int[] x = new int[NUM_INSTANCES];
        int[] y = new int[NUM_INSTANCES];
        int[] outputs = new int[NUM_INSTANCES];

        x[0] = 0; y[0] = 0;
        x[1] = 1; y[1] = 0;
        x[2] = 0; y[2] = 1;
        x[3] = 1; y[3] = 1;

        outputs[0] = 0;
        outputs[1] = 0;
        outputs[2] = 0;
        outputs[3] = 1;
```

```
double[] weights = new double[3]; //2 for input variables and one for bias
double localError, globalError;
int p, iteration, output;

weights[0] = randomNumber(-1, 1);
weights[1] = randomNumber(-1, 1);
weights[2] = randomNumber(-1, 1);

int test = 1;
iteration = 0;
do{
    iteration++;
    globalError = 0;
    //loop through all instances (complete one epoch)
    for(p = 0;p<NUM_INSTANCES;p++){

        //calculate predicted class
        output = calculateOutput(theta, weights, x[p], y[p]);
        //difference between predicted and actual class values
        localError = outputs[p] - output;
        //update weights
        weights[0] += LEARNING_RATE*localError*x[p];
        weights[1] += LEARNING_RATE*localError*y[p];

        //update bias
        weights[2] += LEARNING_RATE*localError;

        globalError += (localError*localError);
    }
    /*Root Mean Squared Error*/ if(test == 1 && globalError == 0){test = 0;
        System.out.println("----------------------------NAUCZONY--------------------------------");
        iteration = MAX_ITER -20;}
    System.out.println("Iteration "+iteration+" : RMSE = "+Math.sqrt(globalError/NUM_INSTANCES));
    int x1 = randGenerator.nextInt( bound: 2);
    int y1 = randGenerator.nextInt( bound: 2);
```

```
            output = calculateOutput(theta, weights, x1, y1);
            System.out.println("\n========\nNew Random Point:");
            System.out.println("x = "+x1+",y = "+y1);
            System.out.println("Works? = "+output);

        }while(iteration < MAX_ITER);

        System.out.println("\n========\nDecision boundary equation:");
        System.out.println(weights[0]+"*x "+weights[1]+"*y + "+weights[2]+" = 0");

    }

    public static double randomNumber(double min, double max){
        double d = min+Math.random()*(max-min);
        return d;
    }

    static int calculateOutput(int theta, double weights[], double x, double y){
        double sum  = x*weights[0] + y*weights[1] + weights[2];
        return sum>=theta ? 1:0;
    }
}
```

Materiały źródłowe wykorzystane w ćwiczeniu:
http://www.codebytes.in/2015/07/perceptron-learning-algorithm-java.html
http://edu.pjwstk.edu.pl/wyklady/nai/scb/wyklad3/w3.htm
http://computing.dcu.ie/~humphrys/Notes/Neural/single.neural.html
http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html
http://www.cs.put.poznan.pl/rklaus/assn/percep.htm