Kamil Wasilonek

Podstawy Sztucznej Inteligencji

Sprawozdanie z projektu nr 2

Celem ćwiczenia było poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

1. **Syntetyczny opis budowy wykorzystanego algorytmu uczenia:**

Do wykonania ćwiczenia wykorzystałem model perceptronu McCullocha-Pittsa oraz algorytm

neuronu typu adaline. Każda z sieci składają się z siedmiu neuronów(warstwa pierwsza). Sześć pierwszych przesyła sygnały wyjściowe do siódmego wyjściowego neuronu.

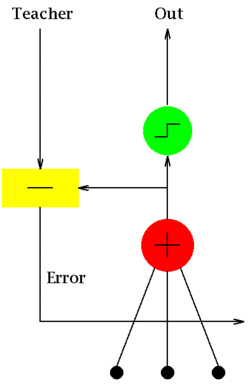
Dla modelu McCullocha-Pittsa sygnały dochodzące do neuronu (I1, I2, ...IN) mnożone są przez tzw. wagi Wi(i - numer neuronu). Dodatkowy sygnał jednostkowy nazwano biasem. Na podstawie wag i dochodzących sygnałów obliczany jest stan wewnętrzny neuronu. Odpowiedź neuronu zależy od tego czy jego stan wewnętrzny (pobudzenie) przekracza pewien poziom. Odpowiada za to unipolarna funkcja aktywacji która w modelu McCullocha-Pittsa jest funkcją progową.



Rys. 1 Schemat neuronu McCullocha-Pittsa

ADALINE ( Adaptive Linear Neuron lub later Adaptive Linear Element ) to wczesna jednowarstwowa sztuczna sieć neuronowa i nazwa fizycznego urządzenia, które zaimplementowało tę sieć. Składa się z wagi, funkcji obciążenia i sumowania.

Różnica między perceptronem Adaline i standardowym ( McCulloch-Pitts ) polega na tym, że w fazie uczenia wagi są korygowane zgodnie z ważoną sumą danych wejściowych .W standardowym perceptronie sieć jest przekazywana do funkcji aktywacji a wyjście funkcji służy do regulacji wag.



*Rys. 2 Schemat neuronu Adaline*

Klasa „Perceptron” odpowiadająca za perceptron składa się z 3 funkcji:

• „**calculateOutput**” odpowiada za sumowanie iloczynów wartości „input” podanych na wejściu oraz wag „weights” .

• „**activationFunction**” jest aktywacyjną funkcja progową unipolarną. Na jej podstawie obliczana jest wartość wyjścia perceptronu.

• „**learnFunction**” wykonuje modyfikacje wag wykorzystując dane otrzymane z powyższych funkcji. W argumentach metody podajemy również oczekiwaną prawidłową wartość oraz „krok uczenia”. Obie te czynniki wykorzystujemy do modyfikacji wag.

Klasa „Adeline” składa się z 4 funkcji:

• „**calculateOutput**” odpowiada za sumowanie iloczynów wartości „input” podanych na wejściu oraz wag „weights” .

• „**activationFunction**” jest aktywacyjną funkcja która zwraca 1 lub - 1.

• „**learnFunction**” wykonuje modyfikacje wag wykorzystując dane otrzymane z funkcji „calculateOutput”. W argumentach metody podajemy również oczekiwaną prawidłową wartość oraz „krok uczenia”. Obie te czynniki wykorzystujemy do modyfikacji wag.

• „**checkActivationFunction**” uruchamia funkcje aktywacji( argumentem jest wartość zwracana z funkcji „calculateOutput”).

1. **Zestawienie otrzymanych wyników:**

Do wykonania ćwiczenia wykorzystałem 10 dużych oraz 10 małych liter reprezentowanych jako tablica 7x5 wypełniona odpowiednio wartościami 0 lub 1. Dla wypełniania tablicy sygnałów wyjściowych dziele tablice znaków na 6 podobszarów. Jeśli w danych sektorze została choć raz wykryta wartość 1 oznaczająca jeden piksel to do odpowiedniej komórki tablicy sygnałów zapisywana jest wartość 1. Połowa wprowadzanych danych została użyta do uczenia zaś druga połowa do testowania.

Przy zastosowaniu współczynnika uczenia wynoszącego **0.1**

Wykres 1. Stosunek liczby epok do procentu popełnianych błędów

Przy zastosowaniu współczynnika uczenia wynoszącego **0.01**

Wykres 2. Stosunek liczby epok do procentu popełnianych błędów

Przy zastosowaniu współczynnika uczenia wynoszącego **0.001**

Wykres 2. Stosunek liczby epok do procentu popełnianych błędów

**3) Analiza i dyskusja błędów uczenia i testowania opracowanych sieci w zależności od**

**wartości współczynnika uczenia oraz wybranego algorytmu:**

Niezależnie od rodzaju użytego neuronu najbardziej znaczącą zmienną w kontekście długości uczenia jest „współczynnik uczenia”. Obserwujemy wyraźny wzrost liczby epok wraz ze zmniejszaniem wartości współczynnika.

W zależności od współczynnika uczenia:

Dla 0.1 - perceptron 5 epok, adaline 1 epoka

Dla 0.01 – perceptron 15 epok, adaline 5 epoki

Dla 0.001 – perceptron 119 epok, adaline 31 epok

Dla lepszego zaobserwowania działania uczenia neuronów przeprowadziłem test w którym pierwsza połowa(5 liter) tablicy posłużyła do nauki rozpoznawania wielkości liter, zaś druga została wykorzystana do testowanie czy neuron jest w stanie odpowiednio rozpoznać wielkość. W moim przypadku niezależnie od użytego współczynnika uczenia oraz użytego algorytmu procent popełnionych błędów wynosił 0%. Program bezbłędnie rozpoznawał wielkość liter. Należy jednak zwrócić uwagę ze wynik ten może być spowodowany dużym podobieństwem użytych liter jak również zbyt małą ilością testowanych znaków. Warto również zauważyć ze przy małych wartościach współczynnika uczenia procent popełnianych bledów nie zmniejszał się proporcjonalnie do liczby epok. Obserwujemy skokowe zmiany np. z 50% do 60% lub 50% na 70%.

1. **Sformułowanie wniosków:**

Na podstawie otrzymanych wyników mogę jednoznacznie stwierdzić że model neuronu adaline jest wielokrotnie wydajniejszy od algorytmu perceptronu McCullocha-Pittsa. W każdym z przeprowadzonych przeze mnie testów liczba epok potrzebnych do nauczenia neuronu rozpoznawania wielkości liter była mniejsza w przypadku adaline. Gdy odpowiednio dobrałem współczynnika uczenia było możliwe prawidłowe nauczenie już w 1 epoce. Na otrzymane wyniki wpłynęła metoda modyfikacji wag w algorytmie adaline. Zadanie polega na tym aby tak dobrać wagę neuronu liniowego, aby na zadany sygnał wejściowy odpowiedź neuronu była możliwie jak najbliższa wartości oczekiwanej.  
Zgodnie z tą regułą, po prezentacji wzorca następuje modyfikacja wag poprzez korektę wagi poprzedniej o iloczyn różnicy między żądaną wartością na wejściu, a otrzymaną wartością na wyjściu neuronu oraz wartość wejścia, z którym powiązana jest ta waga.   
Im większa różnica między żądaną wartością na wejściu, a otrzymaną wartością na wyjściu neuronu, tym korekta wag jest większa. Dla różnicy tej równej zeru nie ma żadnej korekty.

Bibliografia:

https://www.slawop.net/blog/metody-uczenia-liniowych-sieci-neuronowych

<https://en.wikipedia.org/wiki/ADALINE>

1. **Listing całego kodu**

Klasa Adaline

package Adeline;  
  
import java.util.Random;  
  
// Klasa implementująca perceptron  
public class Adaline {  
 public double[] getWeights() {  
 return weights;  
 }  
  
 // Pola używane w klasie  
 private double[] weights;  
 private int numerOfInput;  
 Random random = new Random();  
  
  
 // Konstruktor klasy nadający początkowe wagi  
 public Adaline(int numberOfInput) {  
 this.numerOfInput = numberOfInput;  
 weights = new double[numberOfInput];  
 for (int i = 0; i < weights.length; i++) {  
 //weights[i] = 3;  
 weights[i] = random.nextDouble();  
// System.out.println("Start weight:" + i + " " + weights[i]);  
// System.out.println();  
 }  
 }  
  
  
 //Funkcja aktywacji  
 public int activationFunction(double output) {  
 if(output > 0) {  
 return 1;  
 }else {  
 return -1;  
 }  
 }  
  
  
 //Funkcja sumująca  
 public double calculateOutput(int[] input) {  
 double output = 0;  
 for (int i = 0; i < this.numerOfInput; i++) {  
 output += (input[i] \* weights[i]);  
 }  
 return output;  
 }  
  
 //Funkcja zwracająca wartość w zależności od funkcji aktywacji  
 public int checkActivationFunction(int [] input){  
 double output = 0;  
 for (int i = 0; i < this.numerOfInput; i++) {  
 output += (input[i] \* weights[i]);  
 }  
 return activationFunction(output);  
 }  
  
  
 // Funkcja ucząca, która modyfikuje wagi przy każdym kroku uczenia  
 public void learnFunction(int[] input, double y, double learn\_rate) {  
 double outputSum = calculateOutput(input);  
 for (int i = 0; i < weights.length; i++) {  
 weights[i] += (y - outputSum) \* learn\_rate \* input[i];  
 }  
 }  
}

Klasa Alphabet

package Adeline;  
  
import java.util.Arrays;  
  
*/\*\*  
 \* Created by Kamil on 2017-11-15.  
 \*/*public class Alphabet {  
  
 // Wielkie litery  
 static int[][][] *bigLetters* =  
 {  
 {{0, 0, 1, 0, 0}, {0, 1, 0, 1, 0}, {1, 0, 0, 0, 1}, {1, 1, 1, 1, 1}, {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}}, // A 0  
 {{1, 1, 1, 1, 0}, {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {1, 1, 1, 1, 0}, {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {1, 1, 1, 1, 0}}, // B 1  
 {{0, 1, 1, 1, 1}, {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}, {0, 1, 1, 1, 1}}, // C 2  
 {{1, 1, 1, 1, 0}, {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {1, 1, 1, 1, 0}}, // D 3  
 {{1, 1, 1, 1, 1}, {1, 0, 0, 0, 0}, {1, 0, 0, 0, 0}, {1, 1, 1, 1, 1}, {1, 0, 0, 0, 0}, {1, 0, 0, 0, 0}, {1, 1, 1, 1, 1}}, // E 4  
 {{1, 1, 1, 1, 1}, {1, 0, 0, 0, 0}, {1, 0, 0, 0, 0}, {1, 1, 1, 1, 1}, {1, 0, 0, 0, 0}, {1, 0, 0, 0, 0}, {1, 0, 0, 0, 0}}, // F 5  
 {{0, 1, 1, 1, 1}, {1, 0, 0, 0, 0}, {1, 0, 0, 0, 0}, {1, 0, 0, 1, 1}, {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {0, 1, 1, 1, 1}}, // G 6  
 {{1, 0, 0, 0, 1}, {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}, {1, 0, 0, 0, 1}}, // H 7  
 {{0, 0, 1, 0, 1}, {0, 0, 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, 1, 0, 0}}, // I 8  
 {{0, 1, 1, 1, 0}, {0, 0, 0, 1, 0}, {0, 0, 0, 1, 0}, {0, 0, 0, 1, 1}, {0, 0, 1, 1, 0}, {0, 1, 0, 1, 0}, {0, 0, 1, 0, 0}}, // J 9  
  
 };  
  
 // Małe litery  
 static int[][][] *smallLetters* = {  
 {{0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}, {0, 1, 1, 1, 0}, {1, 0, 0, 1, 0}, {1, 0, 0, 1, 0}, {0, 1, 1, 1, 1}}, // a 0  
 {{1, 0, 0, 0, 0}, {1, 0, 0, 0, 0}, {1, 0, 0, 0, 0}, {1, 1, 1, 0, 0}, {1, 0, 0, 1, 0}, {1, 0, 0, 1, 0}, {1, 1, 1, 0, 0}}, // b 1  
 {{0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}, {0, 1, 1, 1, 0}, {1, 0, 0, 0, 0}, {1, 0, 0, 0, 0}, {0, 1, 1, 1, 0}}, // c 2  
 {{0, 0, 0, 1, 0}, {0, 0, 0, 1, 0}, {0, 0, 0, 1, 0}, {0, 1, 1, 1, 0}, {1, 0, 0, 1, 0}, {1, 0, 0, 1, 0}, {0, 1, 1, 1, 0}}, // d 3  
 {{0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}, {0, 1, 1, 0, 0}, {1, 0, 1, 0, 0}, {1, 1, 0, 0, 0}, {0, 1, 1, 1, 0}}, // e 4  
 {{0, 0, 0, 0, 0}, {0, 1, 1, 0, 0}, {1, 0, 0, 0, 0}, {1, 1, 1, 0, 0}, {1, 0, 0, 0, 0}, {1, 0, 0, 0, 0}, {1, 0, 0, 0, 0}}, // f 5  
 {{0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}, {0, 0, 1, 0, 0}, {0, 1, 0, 1, 0}, {0, 0, 1, 1, 0}, {0, 0, 0, 1, 0}, {0, 1, 1, 0, 0}}, // g 6  
 {{1, 0, 0, 0, 0}, {1, 0, 0, 0, 0}, {1, 0, 0, 0, 0}, {1, 1, 1, 0, 0}, {1, 0, 1, 0, 0}, {1, 0, 1, 0, 0}, {1, 0, 1, 0, 0}}, // h 7  
 {{0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}, {0, 0, 1, 0, 0}, {0, 0, 0, 0, 0}, {0, 0, 1, 0, 0}, {0, 0, 1, 0, 0}}, // i 8  
 {{0, 0, 0, 0, 0}, {0, 0, 1, 0, 0}, {0, 0, 0, 0, 0}, {0, 0, 1, 0, 0}, {0, 0, 1, 0, 0}, {0, 0, 1, 0, 0}, {0, 1, 1, 0, 0}}, // j 9  
 };  
  
  
 //zwraca wektor sygnałów wejściowych zależny od danej litery  
 public static int[] getLetter ( int size, int letter ) {  
 int[] ret = new int[7];  
 Arrays.*fill*( ret, 1, 6, 0 );  
 ret[0] = 1;  
  
 int sizeLetter[][][];  
 if (size == 0) {  
 sizeLetter = *bigLetters*;  
 } else {  
 sizeLetter = *smallLetters*;  
 }  
  
  
 //sektor 1  
 for ( int i = 0; i < 3; i++ )  
 for ( int j = 0; j < 2; j++ )  
 if ( sizeLetter[letter][i][j] == 1 )  
 ret[1] = 1;  
  
 //sektor 2  
 for ( int i = 0; i < 3; i++ )  
 for ( int j = 2; j < 3; j++ )  
 if ( sizeLetter[letter][i][j] == 1 )  
 ret[2] = 1;  
  
 //sektor 3  
 for ( int i = 0; i < 3; i++ )  
 for ( int j = 3; j < 5; j++ )  
 if ( sizeLetter[letter][i][j] == 1 )  
 ret[3] = 1;  
  
 //sektor 4  
 for ( int i = 3; i < 7; i++ )  
 for ( int j = 0; j < 2; j++ )  
 if ( sizeLetter[letter][i][j] == 1 )  
 ret[4] = 1;  
  
 //sektor 5  
 for ( int i = 3; i < 7; i++ )  
 for ( int j = 2; j < 3; j++ )  
 if ( sizeLetter[letter][i][j] == 1 )  
 ret[5] = 1;  
  
 //sektor 6  
 for ( int i = 3; i < 7; i++ )  
 for ( int j = 3; j < 5; j++ )  
 if ( sizeLetter[letter][i][j] == 1 )  
 ret[6] = 1;  
  
 return ret;  
 }  
  
}

Adaline\_main

package Adeline;  
  
import java.util.Arrays;  
  
public class Main\_adaline {  
  
 public static void main ( String[] args ) {  
  
 int numberOfInputs = 7; //ilość wejść  
 int numberOfLetters = 5; //ilość liter MAX 26  
 int counter = 0; //licznik ilości epok uczenia się  
 double leranRate = 0.1; //krok uczenia się  
  
 Adaline[] adaline = new Adaline[numberOfInputs];  
 for ( int i = 0; i < numberOfInputs; i++ )  
 adaline[i] = new Adaline( numberOfInputs );  
  
 int[] lettersArray = new int[numberOfLetters \* 2]; //-1 - duża litera, 1 - mała litera  
 for (int i = 0; i < numberOfLetters; i++) {  
 lettersArray[i] = -1;  
 }  
  
 for (int i = numberOfLetters; i < numberOfLetters \* 2; i++) {  
 lettersArray[i] = 1;  
 }  
  
 int[] outputArray = new int[numberOfLetters \* 2]; //tablica przechowująca wyniki testowania adaline  
  
 while ( ! Arrays.*equals*( lettersArray, outputArray ) ) {  
 int licznik = 0;  
 for ( int i = 0; i < 2; i++ ) { //-1 - wielkie litery, 1 - małe litery  
 for ( int j = 0; j < numberOfLetters; j++ )  
 *learn*( adaline, numberOfInputs, leranRate, i, j );  
 }  
  
 outputArray = *test*( adaline, numberOfLetters, numberOfInputs );  
  
 for (int i = 0; i < numberOfLetters \* 2; i++) {  
 if (outputArray[i] != lettersArray[i]) {  
 licznik++;  
 }  
 }  
  
 System.*out*.println((licznik \* 100) / (2 \* numberOfLetters)+",00%");  
 counter++;  
 }  
  
 System.*out*.println( "Ilość kroków do nauczenia się = " + counter );  
 }  
  
  
 private static void learn ( Adaline[] adaline, int numberOfLetters, double learnRate, int i, int j ) {  
 int[] inVector; //tablica przechowująca wektor sygnałów wejściowych do uczenia pierwszej warstwy sieci  
 inVector = Alphabet.*getLetter*( i, j );  
 *format*( inVector );  
  
 int[] outVector = new int[numberOfLetters]; //tablica przechowująca wektor sygnałów wyjściowych pierwszej warstwy sieci  
 outVector[0] = 1; //bias  
  
 int letter\_size;  
 if ( i == 0 ) letter\_size = - 1;  
 else letter\_size = 1;  
  
 for ( int k = 0; k < numberOfLetters - 1; k++ ) { //uczenie pierwszej warstwy  
 adaline[k].learnFunction( inVector, letter\_size, learnRate );  
 outVector[k + 1] = adaline[k].checkActivationFunction( inVector ); //pobranie sygnału wyjściowego  
 }  
 adaline[numberOfLetters - 1].learnFunction( outVector, letter\_size, learnRate ); //uczenie perceptronu wynikowego na podstawie sygnałów wyjściowych pierwszej warstwy  
 }  
  
 private static int[] test ( Adaline[] adaline, int numberOfLetters, int numberOfInpiut ) {  
 int[] wyj = new int[numberOfLetters \* 2];  
 int[] inVector; //tablica przechowująca wektor sygnałów wejściowych do testowania pierwszej warstwy sieci  
 int[] outVector = new int[numberOfInpiut]; //tablica przechowująca wektor sygnałów wyjściowych pierwszej warstwy sieci  
 outVector[0] = 1; //bias  
  
 for ( int i = 0; i < 2; i++ ) { //testowanie, celem upewnienia się, czy sieć już nauczona  
 for ( int j = 0; j < numberOfLetters; j++ ) {  
 inVector = Alphabet.*getLetter*( i, j );  
 *format*( inVector );  
  
 for ( int k = 0; k < numberOfInpiut - 1; k++ )  
 outVector[k + 1] = adaline[k].checkActivationFunction( inVector );  
  
 wyj[i \* numberOfLetters + j] = adaline[numberOfInpiut - 1].checkActivationFunction( outVector );  
 }  
 }  
 return wyj;  
 }  
  
 //Zmiana sygnału wejściowego 0 na sygnały -1  
 private static void format( int[] vector ){  
 for ( int k = 0; k < vector.length; k++ )  
 if ( vector[k] == 0 ) vector[k] = -1;  
 }  
}

Klasa Perceptron\_Main

package PerceptronPackage;  
  
  
import java.util.Arrays;  
  
public class Main\_perceptron {  
  
 public static void main(String[] args) {  
  
 int numberOfInputs = 7; //ilość wejść  
 int numberOfLetters = 5; //ilość liter do uczenia  
 int counter = 0; //licznik epok uczenia się  
 double leranRate = 0.1; //krok uczenia się  
  
 Perceptron[] perceptron = new Perceptron[numberOfInputs];  
 for (int i = 0; i < numberOfInputs; i++)  
 perceptron[i] = new Perceptron(numberOfInputs);  
  
 int[] lettersArray = new int[numberOfLetters \* 2]; //0 - duża litera, 1 - mała litera  
  
 for (int i = 0; i < numberOfLetters; i++) {  
 lettersArray[i] = 0;  
 }  
  
 for (int i = numberOfLetters; i < numberOfLetters \* 2; i++) {  
 lettersArray[i] = 1;  
 }  
  
  
 int[] outputArray = new int[numberOfLetters \* 2]; //tablica wynikowa testowania  
  
  
 while (!Arrays.*equals*(lettersArray, outputArray)) {  
 int licznik = 0;  
 for (int i = 0; i < 2; i++) //0 - wielkie litery, 1 - małe litery  
 for (int j = 0; j < numberOfLetters; j++)  
 *mainLearn*(perceptron, numberOfInputs, leranRate, i, j);  
  
 outputArray = *test*(perceptron, numberOfLetters, numberOfInputs);  
  
 for (int i = 0; i < numberOfLetters \* 2; i++) {  
 if (outputArray[i] != lettersArray[i]) {  
 licznik++;  
 }  
 }  
  
 System.*out*.println((int)(licznik \* 100) / (2 \* numberOfLetters)+",00%");  
 counter++;  
  
 }  
  
 System.*out*.println("Ilość kroków do nauczenia się = " + counter);  
  
  
 //Testowanie czy rozpoznaje pozostały przedział liter  
 int licznik = 0;  
  
 for (int i = 0; i < 2; i++) { //0 - wielkie litery, 1 - małe litery  
 for (int j = numberOfInputs; j < numberOfLetters \* 2; j++){  
 outputArray = *test*(perceptron, numberOfLetters, numberOfInputs);  
 }  
  
 }  
  
 for (int i = 0; i < numberOfLetters \* 2; i++) {  
 if (outputArray[i] != lettersArray[i]) {  
 licznik++;  
 }  
 }  
  
 System.*out*.println("Procent błedów testowania " + (licznik \* 100.0) / (2.0 \* numberOfLetters));  
 }  
  
  
 public static void mainLearn(Perceptron[] perceptron, int numberOfInputs, double lr, int i, int j) {  
 int[] InputVector; //tablica sygnałów wejściowych pierwszej warstwy  
 InputVector = Alphabet.*getLetter*(i, j);  
  
 int[] vector\_p = new int[numberOfInputs]; //tablica sygnałów wyjściowych pierwszej warstwy  
 vector\_p[0] = 1; //bias  
  
 for (int k = 0; k < numberOfInputs - 1; k++) { //uczenie pierwszej warstwy  
 perceptron[k].learnFunction(InputVector, i, lr);  
 vector\_p[k + 1] = perceptron[k].calculateOutput(InputVector); //pobranie sygnału wyjściowego  
 }  
 perceptron[numberOfInputs - 1].learnFunction(vector\_p, i, lr); //uczenie perceptronu wynikowego na podstawie sygnałów wyjściowych pierwszej warstwy  
 }  
  
 public static int[] test(Perceptron[] perc, int numberOfLetters, int noi) {  
 int[] wyj = new int[numberOfLetters \* 2];  
 int[] vector; //tablica przechowująca wektor sygnałów wejściowych do testowania pierwszej warstwy sieci  
 int[] vector\_p = new int[noi]; //tablica przechowująca wektor sygnałów wyjściowych pierwszej warstwy sieci  
 vector\_p[0] = 1; //bias  
  
 for (int i = 0; i < 2; i++) { //testowanie, celem upewnienia się, czy sieć już nauczona  
 for (int j = 0; j < numberOfLetters; j++) {  
 vector = Alphabet.*getLetter*(i, j);  
 for (int k = 0; k < noi - 1; k++)  
 vector\_p[k + 1] = perc[k].calculateOutput(vector);  
  
 wyj[i \* numberOfLetters + j] = perc[noi - 1].calculateOutput(vector\_p);  
 }  
 }  
 return wyj;  
 }  
}

Klasa Perceptron

package PerceptronPackage;  
  
import java.util.Random;  
  
// Klasa implementująca perceptron  
public class Perceptron {  
 public double[] getWeights() {  
 return weights;  
 }  
  
 // Pola używane w klasie  
 private double[] weights;  
 private int numerOfInput;  
 Random random = new Random();  
  
  
 // Konstruktor klasy nadający początkowe wagi  
 public Perceptron(int numberOfInput) {  
 this.numerOfInput = numberOfInput;  
 weights = new double[numberOfInput];  
 for (int i = 0; i < weights.length; i++) {  
 //weights[i] = 3;  
 weights[i] = random.nextDouble();  
// System.out.println("Start weight:" + i + " " + weights[i]);  
// System.out.println();  
 }  
 }  
  
  
 //Funkcja aktywacji oparta na unipolarnej funkcji progowej  
 public int activationFunction(double output) {  
 if (output < 0) return 0;  
 else return 1;  
 }  
  
  
 //Funkcja sumująca  
 public int calculateOutput(int[] input) {  
 double output = 0;  
 for (int i = 0; i < this.numerOfInput; i++) {  
 output += (input[i] \* weights[i]);  
 }  
 return activationFunction(output);  
 }  
  
  
 // Funkcja ucząca, która modyfikuje wagi przy każdym kroku uczenia  
 public void learnFunction(int[] input, double y, double learn\_rate) {  
 double output = calculateOutput(input);  
 for (int i = 0; i < weights.length; i++) {  
 weights[i] += (y - output) \* learn\_rate \* input[i];  
 }  
 }  
}