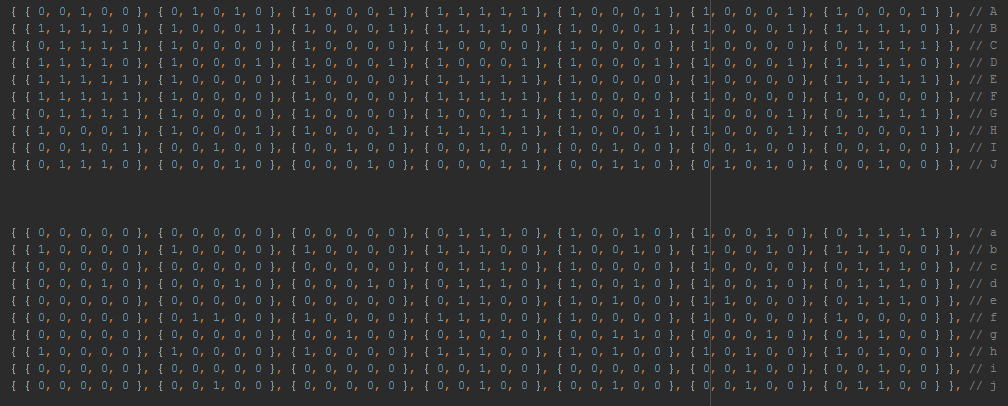
Krzysztof Sekuła

Sprawozdanie z projektu nr 2

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

Do wykonania ćwiczenia stworzyłem 2 jednowarstwowe sieci. Pierwsza na podstawie modelu perceptronu McCullocha-Pittsa, natomiast druga wykorzystuje neuron typu Adaline.

Jako dane uczące i testujące wykorzystałem litery przedstawione jako dwuwymiarowa tablica o rozmiarach 7x5:



Proces uczenia przetestowałem kilka razy dla różnego tempa uczenia. Wagi początkowe zostały dobrane w sposób losowy co miało wpływ na wyniki uczenia się.

**Wyniki:**

|  |  |  |
| --- | --- | --- |
| Tempo Uczenia | Perceptor | Adaline |
| 0,1 | 11 | 2 |
| 0,01 | 18 | 6 |
| 0,01 | 111 | 91 |

**Wnioski:**

Po przeprowadzonych testach możemy stwierdzić że im mniejszy współczynnik uczenia tym więcej kroków było potrzebne do nauki. Kiedy tempo uczenia jest niższe możemy zauważyć że % błędów skacze. Może to być spowodowane podobieństwem niektórych liter jak np.: F f, L l. Na liczbę kroków miało wpływ również to że wagi początkowe były ustawiane losowo, co powodowało że każde uruchomienie programu dawało różnie wyniki.

Z otrzymanych wyników możemy wywnioskować iż neuron typy Adaline jest o wiele bardziej wydajny niż Perceptor. Różnice mogą wynikać z tego iż w przypadku Adaline modyfikowanie wag następuje przed funkcją aktywacji. Oznacza to że jeżeli wartość oczekiwana znacznie różni się od wartości wyliczonej to wagi zostaną znacznie zmienione. Kolejną różnicą są sygnały wejściowe. W przypadku Perceptronu są to 0 lub 1, co powoduje że wagi będą się tylko zwiększać. W Adaline sygnały wejściowe to -1 lub 1, co sprawia że wagi nie tylko mogą się zwiększa ale również zmniejszać.

**Listing kodu:**

public class Litery {  
  
 static int[][][][] *litery* = {  
 {  
 { { 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  
 { { 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  
 { { 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  
 { { 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  
 { { 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  
 { { 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  
 { { 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  
 { { 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  
 { { 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  
 { { 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  
  
 },  
 {  
 { { 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  
 { { 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  
 { { 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  
 { { 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  
 { { 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  
 { { 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  
 { { 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  
 { { 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  
 { { 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  
 { { 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  
  
 }  
 };  
  
  
  
 public static int[] getLetter ( int size, int letter ) {  
 int[] ret = new int[7];  
 Arrays.*fill*( ret, 1, 6, 0 );  
 ret[0] = 1;  
//sektor 1  
 for ( int i = 0; i < 3; i++ )  
 for ( int j = 0; j < 2; j++ )  
 if ( *litery*[size][letter][i][j] == 1 )  
 ret[1] = 1;  
//sektor 2  
 for ( int i = 0; i < 3; i++ )  
 for ( int j = 2; j < 3; j++ )  
 if ( *litery*[size][letter][i][j] == 1 )  
 ret[2] = 1;  
//sektor 3  
 for ( int i = 0; i < 3; i++ )  
 for ( int j = 3; j < 5; j++ )  
 if ( *litery*[size][letter][i][j] == 1 )  
 ret[3] = 1;  
//sektor 4  
 for ( int i = 3; i < 7; i++ )  
 for ( int j = 0; j < 2; j++ )  
 if ( *litery*[size][letter][i][j] == 1 )  
 ret[4] = 1;  
//sektor 5  
 for ( int i = 3; i < 7; i++ )  
 for ( int j = 2; j < 3; j++ )  
 if ( *litery*[size][letter][i][j] == 1 )  
 ret[5] = 1;  
//sektor 6  
 for ( int i = 3; i < 7; i++ )  
 for ( int j = 3; j < 5; j++ )  
 if ( *litery*[size][letter][i][j] == 1 )  
 ret[6] = 1;  
 return ret;  
 }  
}

public class Adaline {  
  
 private int ilosc\_wejsc;  
 private double[] waga;  
 public Adaline ( int numbers\_of\_inputs ) {  
 ilosc\_wejsc = numbers\_of\_inputs;  
 waga = new double[ilosc\_wejsc];  
 for ( int i = 0; i < ilosc\_wejsc; i++ )  
 waga[i] = new Random().nextDouble();  
 }  
  
 private int active ( double y\_p ) {  
 return y\_p <= 0 ? (-1) : 1;  
 }  
  
 public double suma ( int[] x ) {  
 double y\_p = 0;  
 for ( int i = 0; i < ilosc\_wejsc; i++ )  
 y\_p += x[i] \* waga[i];  
 return y\_p;  
 }  
  
 public void learn ( int[] x, double y, double lr ) {  
 double y\_p = suma( x );  
 for ( int i = 0; i < ilosc\_wejsc; i++ )  
 waga[i] += ( y - y\_p ) \* lr \* x[i];  
 }  
  
 public int test ( int[] x )  
 {  
 return ( active( suma( x ) ) );  
 }  
}

public class Perceptron {  
  
 private int ilosc\_wejsc;  
 private double[] waga;  
 public Perceptron ( int numbers\_of\_inputs ) {  
 ilosc\_wejsc = numbers\_of\_inputs;  
 waga= new double[ilosc\_wejsc];  
 for ( int i = 0; i < ilosc\_wejsc; i++ )  
 waga[i] = new Random().nextDouble();  
 }  
  
 private int active ( double y\_p ) {  
 return y\_p < 0 ? 0 : 1;  
 }  
  
 public int suma ( int[] x ) {  
 double y\_p = 0;  
 for ( int i = 0; i < ilosc\_wejsc; i++ )  
 y\_p += x[i] \* waga[i];  
 return active( y\_p );  
 }  
  
 public void learn ( int[] x, double y, double lr ) {  
 double y\_p = suma( x );  
 for ( int i = 0; i < ilosc\_wejsc; i++ )  
 waga[i] += ( y - y\_p ) \* lr \* x[i];  
 }  
}

public class P\_Main {  
  
 public static void main ( String[] args ) {  
 int ilosc\_wejsc = 5; //ilość wejść  
 int nol = 10; //ilość liter  
 int counter = 0; //licznik krokow  
 double lr = 0.01; //tempo uczenia się  
  
 Perceptron[] perc = new Perceptron[ilosc\_wejsc];  
 for (int i = 0; i < ilosc\_wejsc; i++)  
 perc[i] = new Perceptron(ilosc\_wejsc);  
 int[] y = new int[nol \* 2]; //0 - duża litera, 1 - mała litera  
 Arrays.*fill*(y, 0, nol, 0);  
 Arrays.*fill*(y, nol, nol \* 2, 1);  
 int[] wyj = new int[nol \* 2]; //tablica przechowująca wyniki testowania perceptronu  
 Arrays.*fill*(wyj, 0, nol \* 2, 0);  
 int proc;  
 while (!Arrays.*equals*(y, wyj)) {  
 proc=0;  
 for (int i = 0; i < 2; i++) //0 - wielkie litery, 1 - małe litery  
 for (int j = 0; j < nol; j++)  
 *learn*(perc, ilosc\_wejsc, lr, i, j);  
 wyj = *test*(perc, nol, ilosc\_wejsc);  
  
 for ( int i = 0; i < nol \* 2; i++ )  
 if ( wyj[i] != y[i] )  
 proc++;  
 counter++;  
 System.*out*.format( "%.6f%n", ( double ) proc / ( nol \* 2 ) );  
 }  
  
 System.*out*.println();  
  
 //TESTOWANIE  
 for ( int i = 0; i < nol; i++ ) //wyświetlenie liter  
 System.*out*.print((char) (i + 65) + "\t");  
 for ( int i = 0; i < nol; i++ )  
 System.*out*.print((char)(i+97) + "\t");  
  
 System.*out*.println();  
  
 for ( int i = 0; i < 2; i++ ) { //wyświetlenie wynków  
 for ( int j = 0; j < nol; j++ )  
 System.*out*.print( wyj[i \* nol + j] + "\t" );  
  
 }  
 System.*out*.println();  
  
 System.*out*.println( "Ilość kroków do nauczenia się = " + counter );  
 }  
  
  
  
  
 public static void learn (Perceptron[] perc, int noi, double lr, int i, int j ) {  
 int[] vector; //tablica przechowująca wektor sygnałów wejściowych do uczenia pierwszej warstwy sieci  
 vector = Litery.*getLetter*( i, j );  
 int[] vector\_p = new int[noi]; //tablica przechowująca wektor sygnałów wyjściowych pierwszej warstwy sieci  
 vector\_p[0] = 1; //bias  
 for ( int k = 0; k < noi - 1; k++ ) { //uczenie pierwszej warstwy  
 perc[k].learn( vector, i, lr );  
 vector\_p[k + 1] = perc[k].suma( vector ); //pobranie sygnału wyjściowego  
 }  
 perc[noi - 1].learn( vector\_p, i, lr ); //uczenie perceptronu wynikowego  
//na podstawie sygnałów wyjściowych pierwszej warstwy  
 }  
 public static int[] test (Perceptron[] perc, int nol, int noi ) {  
 int[] wyj = new int[nol \* 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++ ) {  
 for ( int j = 0; j < nol; j++ ) {  
 vector = Litery.*getLetter*( i, j );  
 for ( int k = 0; k < noi - 1; k++ )  
 vector\_p[k + 1] = perc[k].suma( vector );  
 wyj[i \* nol + j] = perc[noi - 1].suma( vector\_p );  
 }  
 }  
 return wyj;  
 }  
}

public class A\_Main {  
  
 public static void main ( String[] args ) {  
  
 int ilosc\_wejsc = 5; //ilość wejść  
 int nol = 10; //ilość liter  
 int counter = 0; //licznik krokow  
 double lr = 0.01; //tempo uczenia się  
  
 Adaline[] ada = new Adaline[ilosc\_wejsc];  
 for ( int i = 0; i < ilosc\_wejsc; i++ )  
 ada[i] = new Adaline( ilosc\_wejsc );  
 int[] y = new int[nol \* 2]; //-1 - duża litera, 1 - mała litera  
 Arrays.*fill*( y, 0, nol, - 1 );  
 Arrays.*fill*( y, nol, nol \* 2, 1 );  
  
 int[] wyj = new int[nol \* 2]; //tablica przechowująca wyniki testowania adaline  
 Arrays.*fill*( wyj, 0, nol \* 2, 0 );  
  
 int proc;  
  
 while ( ! Arrays.*equals*( y, wyj ) ) {  
 proc = 0;  
  
 for ( int i = 0; i < 2; i++ ) { //-1 - wielkie litery, 1 - małe litery  
 for ( int j = 0; j < nol; j++ )  
 *learn*( ada, ilosc\_wejsc, lr, i, j );  
 }  
  
 wyj = *test*( ada, nol, ilosc\_wejsc );  
  
 for ( int i = 0; i < nol \* 2; i++ )  
 if ( wyj[i] != y[i] )  
 proc++;  
  
 counter++;  
 System.*out*.format( "%.6f%n", ( double ) proc / ( nol \* 2 ) );  
 }  
 System.*out*.println();  
  
 //TESTOWANIE  
 for ( int i = 0; i < nol; i++ ) //wyświetlenie liter  
 System.*out*.print((char) (i + 65) + "\t");  
 for ( int i = 0; i < nol; i++ )  
 System.*out*.print((char)(i+97) + "\t");  
 System.*out*.println();  
  
 for ( int i = 0; i < 2; i++ ) { //wyświetlenie wynków  
 for ( int j = 0; j < nol; j++ )  
 System.*out*.print( wyj[i \* nol + j] + "\t" );  
  
 }  
 System.*out*.println();  
 System.*out*.println( "Ilość kroków do nauczenia się = " + counter );  
 }  
  
  
 private static void learn ( Adaline[] ada, int noi, double lr, int i, int j ) {  
 int[] vector; //tablica przechowująca wektor sygnałów wejściowych do uczenia pierwszej warstwy sieci  
 vector = Litery.*getLetter*( i, j );  
 *format*( vector );  
 int[] vector\_p = new int[noi]; //tablica przechowująca wektor sygnałów wyjściowych pierwszej warstwy sieci  
 vector\_p[0] = 1; //bias  
 int letter\_size;  
 if ( i == 0 ) letter\_size = - 1;  
 else letter\_size = 1;  
 for ( int k = 0; k < noi - 1; k++ ) { //uczenie pierwszej warstwy  
 ada[k].learn( vector, letter\_size, lr );  
 vector\_p[k + 1] = ada[k].test( vector ); //pobranie sygnału wyjściowego  
 }  
 ada[noi - 1].learn( vector\_p, letter\_size, lr ); //uczenie perceptronu wynikowego  
//na podstawie sygnałów wyjściowych pierwszej warstwy  
 }  
 private static int[] test ( Adaline[] ada, int nol, int noi ) {  
 int[] wyj = new int[nol \* 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++ ) {  
 for ( int j = 0; j < nol; j++ ) {  
 vector = Litery.*getLetter*( i, j );  
 *format*( vector );  
 for ( int k = 0; k < noi - 1; k++ )  
 vector\_p[k + 1] = ada[k].test( vector );  
 wyj[i \* nol + j] = ada[noi - 1].test( vector\_p );  
 }  
 }  
 return wyj;  
 }  
 //w przypadku adaline sygnały wejściowe = 0 muszą być zamienione na sygnały -1  
 private static void format( int[] vector ){  
 for ( int k = 0; k < vector.length; k++ )  
 if ( vector[k] == 0 ) vector[k] = -1;  
 }  
}