Podstawy Sztucznej Inteligencji Monika Michalik – gr.2

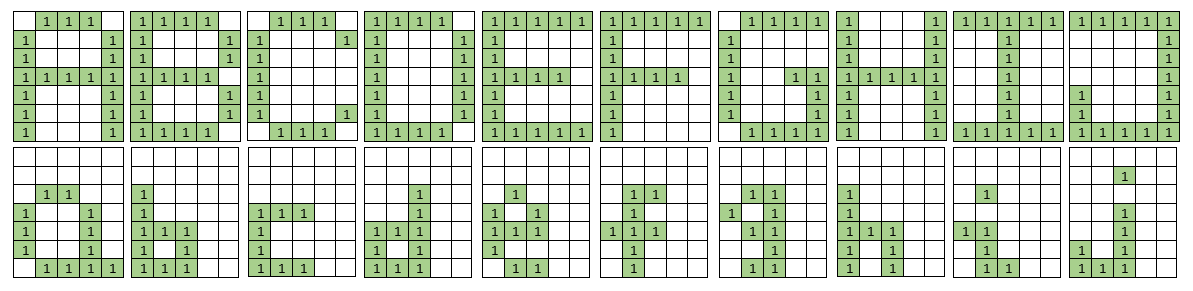
**Scenariusz 2. Budowa i działanie sieci jednowarstwowej.**

**Cel ćwiczenia:**

Celem ćwiczenia jest poznanie budowy i działania jednowarstowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

**Zadania do wykonania:**

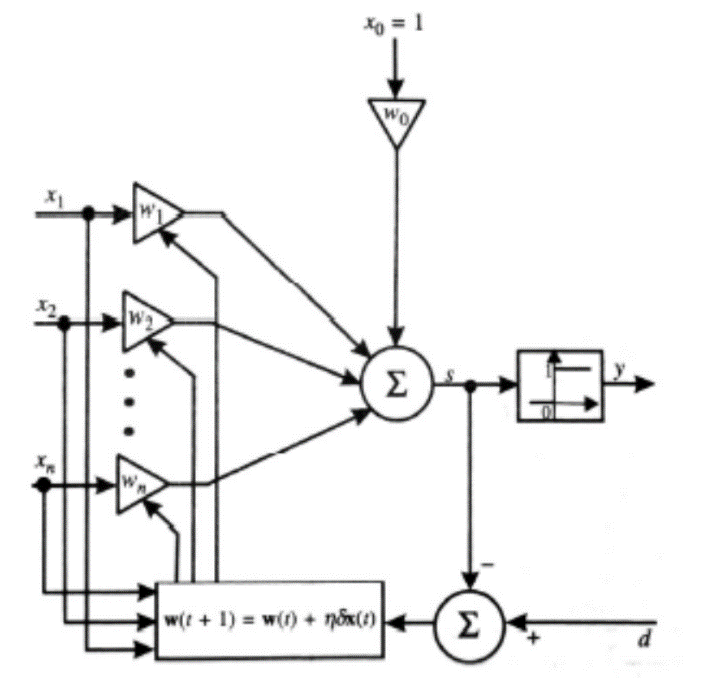
**1. Wygenerowanie danych uczących i testujących, zawierających 10 dużych i 10 małych liter alfabetu w postaci dwuwymiarowej tablicy 5x7 pikseli dla jednej litery.**

****

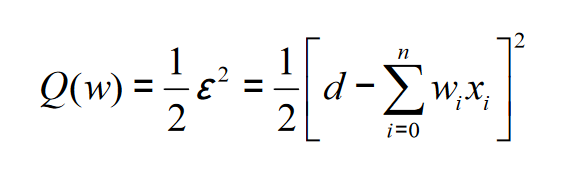
**2. Przygotowanie jednowarstwowej sieci wg algorytmu Adaline.**

**Syntetyczny opis budowy oraz algorytmu uczenia.**

Schemat neuronu Adaline przedstawiono na poniższym rysunku. Budowa tego neuronu jest bardzo podobna do modelu perceptronu, a jedyna różnica dotyczy algorytmu uczenia. Sygnał wyznacza się w ten sam sposób, co w przypadku uczenia perceptronu. Jednak w przypadku neuronu typu Adaline porównuje się sygnał wzorcowy d z sygnałem s, na wyjściu części liniowej neuronu (nie uwzględnia się funkcji aktywacji).

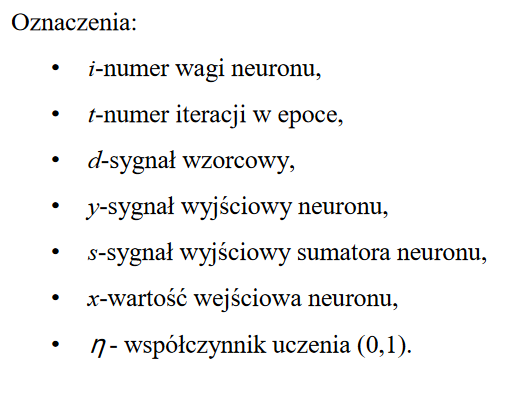
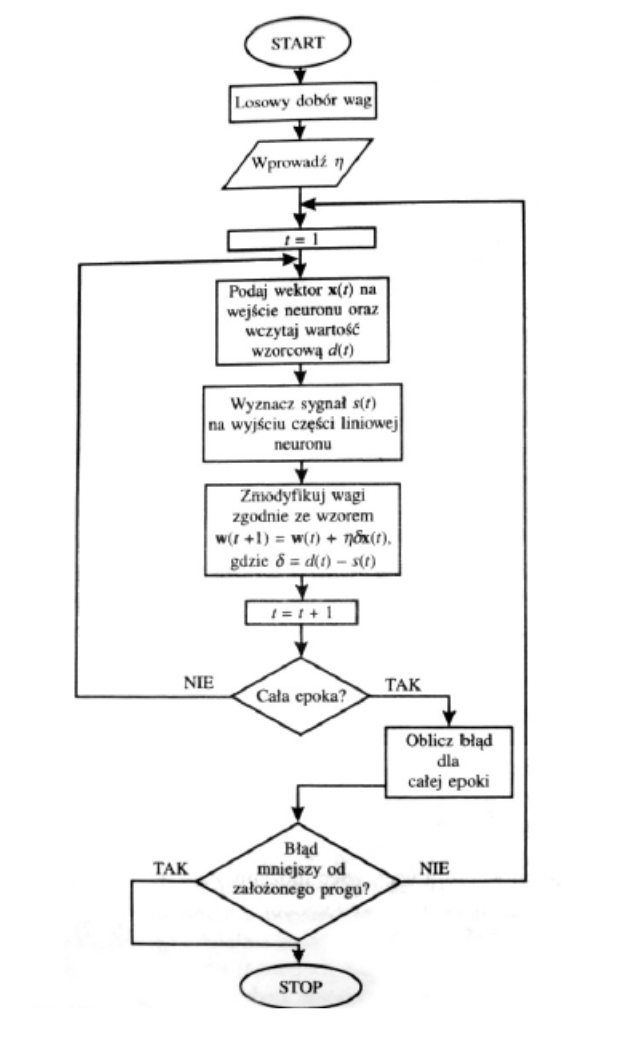
****

Różnice między sygnałem wzorcowym, a sygnałem s nazywamy błędem = d – s. Uczenie neuronu, czyli dobór wag, sprowadza się do minimalizacji funkcji określonej w sposób następujący:



Miarę błędu określa się mianem błędu średniego kwadratowego.

**Algorytm uczenia neuronu typu Adaline w postaci schematu blokowego:**

****

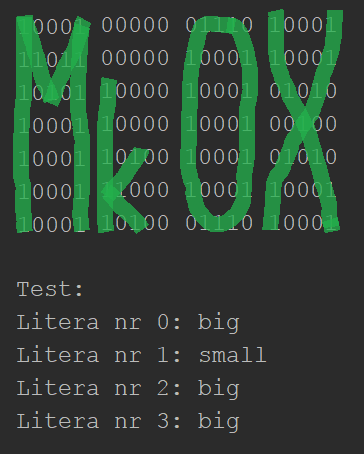
**3. Uczenie sieci przy różnych współczynnikach uczenia.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Współczynnik uczenia** | **Liczba iteracji** | **Błąd przy DUŻYCH literach** | **Błąd przy małych literach** |
| **0.01** | **16** | **1** | **1** |
| **0.01** | **100** | **0** | **0** |
| **0.01** | **30** | **0** | **1** |
| **0.01** | **42** | **0** | **0** |
| **0.01** | **3** | **3** | **7** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Współczynnik uczenia** | **Liczba iteracji** | **Błąd przy DUŻYCH literach** | **Błąd przy małych literach** |
| **0.1** | **3** | **4** | **0** |
| **0.1** | **90** | **0** | **0** |
| **0.0001** | **200** | **1** | **3** |
| **0.00001** | **2000** | **0** | **4** |

**4. Testowanie sieci dla danych zawierających 4 litery.**

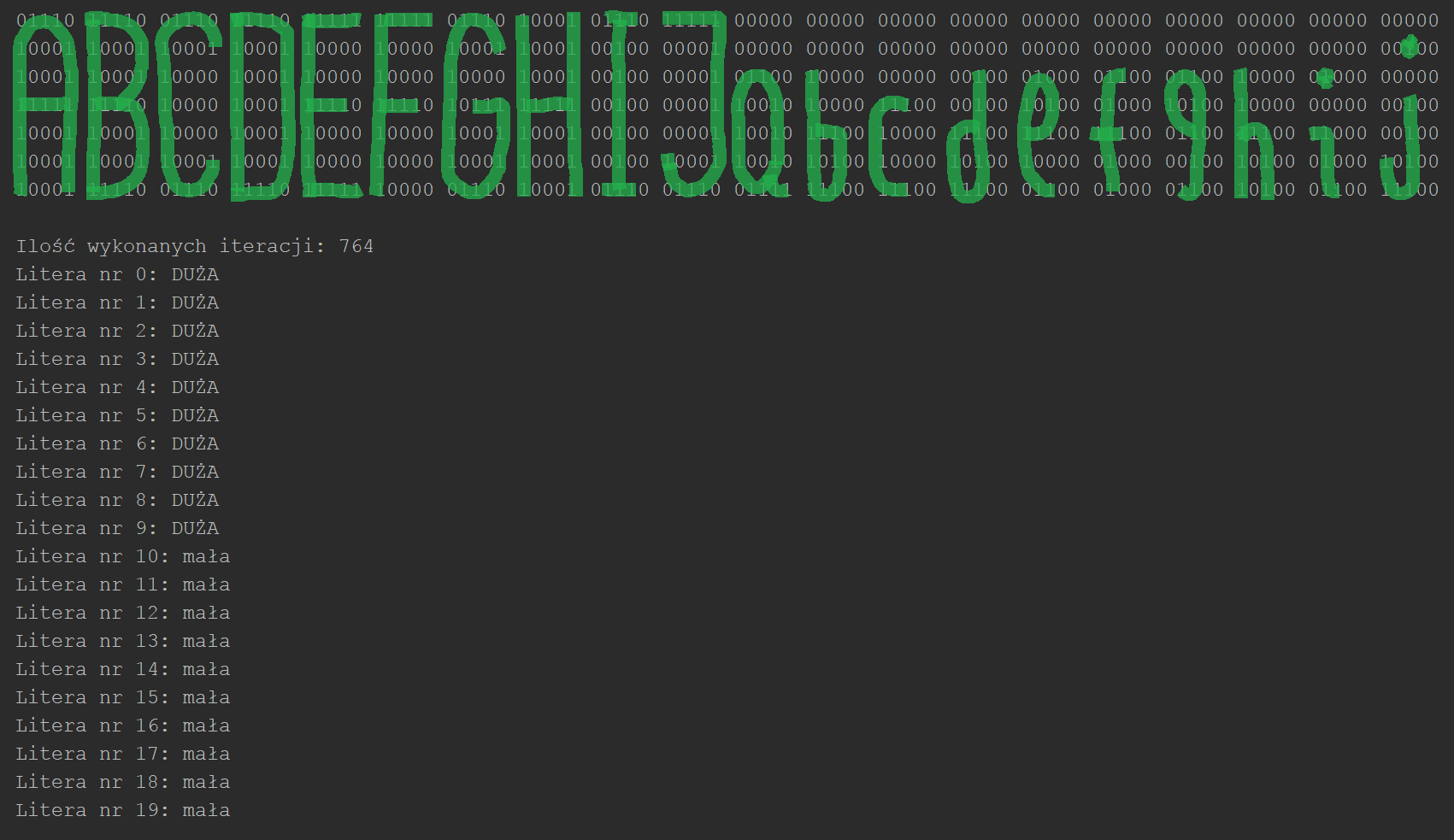
Został wykonany test dla liter: MkOX, wynik prezentuje poniżej:

****

**5. Wnioski.**

Im mniejszy współczynnik uczenia, tym więcej ilości iteracji potrzeba do uzyskania poprawnego wyniku. Wagi są przydzielane losowo, stąd liczba iteracji potrzebna do nauczenia sieci jest rozbieżna. Jak widać w pierwszej tabeli, przy współczynniku 0.01 potrzebowaliśmy 100, z kolejnym wynikiem już tylko 42 iteracji do uzyskania oczekiwanego rezultatu. Duża rozbieżność ma związek z losowym doborem wag.

**6. Przykładowy wynik programu:**

****

**7. Listing kodu:**

Main.java:

public class Main {  
  
 public static void main(String[] args) {  
  
 int[][] letters = {  
 {0,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1}, // 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  
 {0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,1,1,1,0}, // 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  
 {1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1}, // E  
 {1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0}, // F  
 {0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, // 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  
 {0,1,1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,1,1,0}, // I  
 {1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, // J  
  
 {0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,0,1,1,1,1}, // a  
 {0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0,1,0,1,0,0,1,1,1,0,0}, // b  
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0}, // c  
 {0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,1,1,0,0,1,0,1,0,0,1,1,1,0,0}, // d  
 {0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,0,0,1,1,1,0,0,1,0,0,0,0,0,1,1,0,0}, // e  
 {0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,1,0,0,0}, // f  
 {0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,1,0,0,0,1,1,0,0,0,0,1,0,0,0,1,1,0,0}, // g  
 {0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0,1,0,1,0,0,1,0,1,0,0}, // h  
 {0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0,1,1,0,0}, // i  
 {0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,0,1,0,0,1,1,1,0,0} // j  
 };  
  
 int[] bigOrSmall = {1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0};  
  
 int[][] test\_letters = {  
 {1,0,0,0,1,1,1,0,1,1,1,0,1,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1}, // M  
 {0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,1,0,0,1,1,0,0,0,1,0,1,0,0}, // k  
 {0,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, // O  
 {1,0,0,0,1,1,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,1,1,0,0,0,1}, // X  
 };  
  
  
 AdalineNeuron adaline = new AdalineNeuron(letters, bigOrSmall);  
 adaline.printLetters(letters);  
 adaline.learn(adaline);  
  
 System.*out*.println("\n");  
  
 adaline.printLetters(test\_letters);  
 adaline.test(test\_letters, adaline);  
 }  
}

AdalineNeuron.java:

public class AdalineNeuron {  
  
 private int[][] lettersMy;  
 private int[] bigOrSmall;  
 private double[] weight;  
 double[] membranePotential;  
  
 private double learning\_ratio = 0.01;  
 private double maxIteration = 2000;  
 private int amountOfLetters = 20;  
  
 public AdalineNeuron(int[][] letters, int[] bigOrSmall){  
 this.lettersMy = letters;  
 this.bigOrSmall = bigOrSmall;  
  
 this.weight = new double[35];  
 membranePotential = new double[amountOfLetters]; // Zawiera potencjał membranowy dla każdej litery  
 }  
  
 public void learn(AdalineNeuron adaline)  
 {  
 int[] expectedResults = bigOrSmall; //wyniki oczekiwane [ 1 - duża,

0 - mała litera ]  
 double error = 0;  
  
 for(int i=0; i<35; i++){  
 weight[i] = Math.*random*();  
 }  
  
 int iteration = 0;  
 for(int i=0; i<maxIteration; i++)  
 {  
 iteration++;  
  
 // Wykonuj dla każdej litery uczącej  
 for (int j=0; j<amountOfLetters; j++)  
 {  
 int[] oneLetterTab = lettersMy[j]; // Prześlij adres początku tablicy litery

membranePotential[j]=adaline.getMembranePotential(oneLetterTab, weight);  
  
 // Modyfikacja wag wg wzoru  
 for ( int k = 0; k < weight.length; k++ ) {  
 weight[k] += oneLetterTab[k] \* (expectedResults[j] - membranePotential[j]) \* learning\_ratio;  
 }  
 error = 0.5\*(expectedResults[j] - membranePotential[j])\*(expectedResults[j] - membranePotential[j]);  
 }  
  
 // Jeśli błąd mniejszy od założonego progu - zakończ!  
 if(error<0.0001) {  
 break;  
 }  
 }  
 resultWithActivation(adaline, iteration);  
 }  
  
  
 // Wynik z użyciem funkcji aktywacji  
 public void resultWithActivation(AdalineNeuron adaline, int iteration) {  
 System.*out*.println("Ilość wykonanych iteracji: " + iteration);  
 for(int i=0; i<20;i++){  
 System.*out*.print("Litera nr " + i + ": ");  
 if(adaline.activationBlock(membranePotential[i])==1) {  
 System.*out*.println("DUŻA");  
 }else{  
 System.*out*.println("mała");  
 }  
 }  
 }  
  
 // Zwraca potencjał membranowy, sumuje każdy znak\*waga  
 public double getMembranePotential(int[] tab, double[] weight){  
 double membranePotential = 0;  
 for(int i=0; i < tab.length; i++)  
 membranePotential += tab[i] \* weight[i];  
 return membranePotential;  
 }  
  
 // Funkcja aktywacji  
 private int activationBlock(double membranePotential) {  
 if(membranePotential <= 0.5)  
 return 0;  
 else  
 return 1;  
 }  
  
 public void test(int[][] test\_letters, AdalineNeuron adaline) {  
  
 double[] sum = new double[amountOfLetters];  
 int[] tab;  
 System.*out*.println("Test:");  
 for (int i = 0; i<4; i++){  
 tab = test\_letters[i];  
 sum[i] = adaline.getMembranePotential(tab, weight);  
 System.*out*.print("Litera nr " + i + ": ");  
 if(adaline.activationBlock(sum[i])==1) {  
 System.*out*.println("big");  
 }else{  
 System.*out*.println("small");  
 }  
 }  
 }  
  
 // Wyświetla tablicę liter  
 public void printLetters(int[][] letters){  
 for(int i =0; i<7; i++){  
 for(int j = 0; j< letters.length; j++){  
 for(int k=0; k<5; k++){  
 System.*out*.print(letters[j][(i\*5) + k]);  
 }  
 System.*out*.print(" ");  
 }  
 System.*out*.println();  
 }  
 System.*out*.println();  
 }  
}