Podstawy Sztucznej Inteligencji Monika Michalik – gr.2

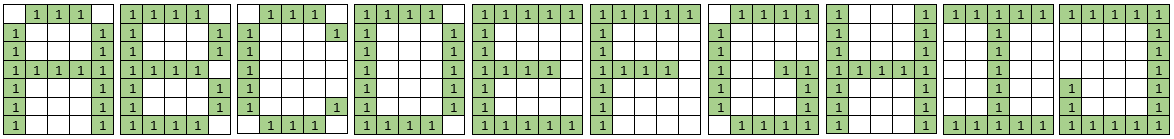
**Scenariusz 4. Uczenie sieci regułą Hebba.**

**Cel ćwiczenia:**

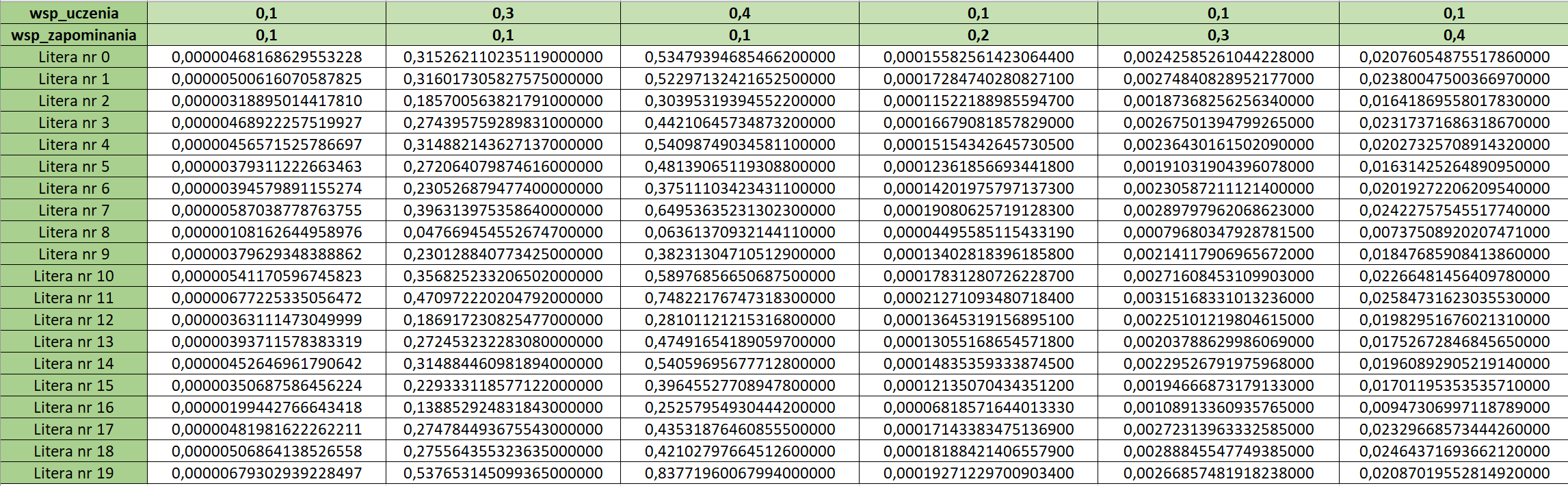
Celem ćwiczenia jest poznanie działania reguły Hebba dla sieci jednowarstowej na przykładzie grupowania liter alfabetu.

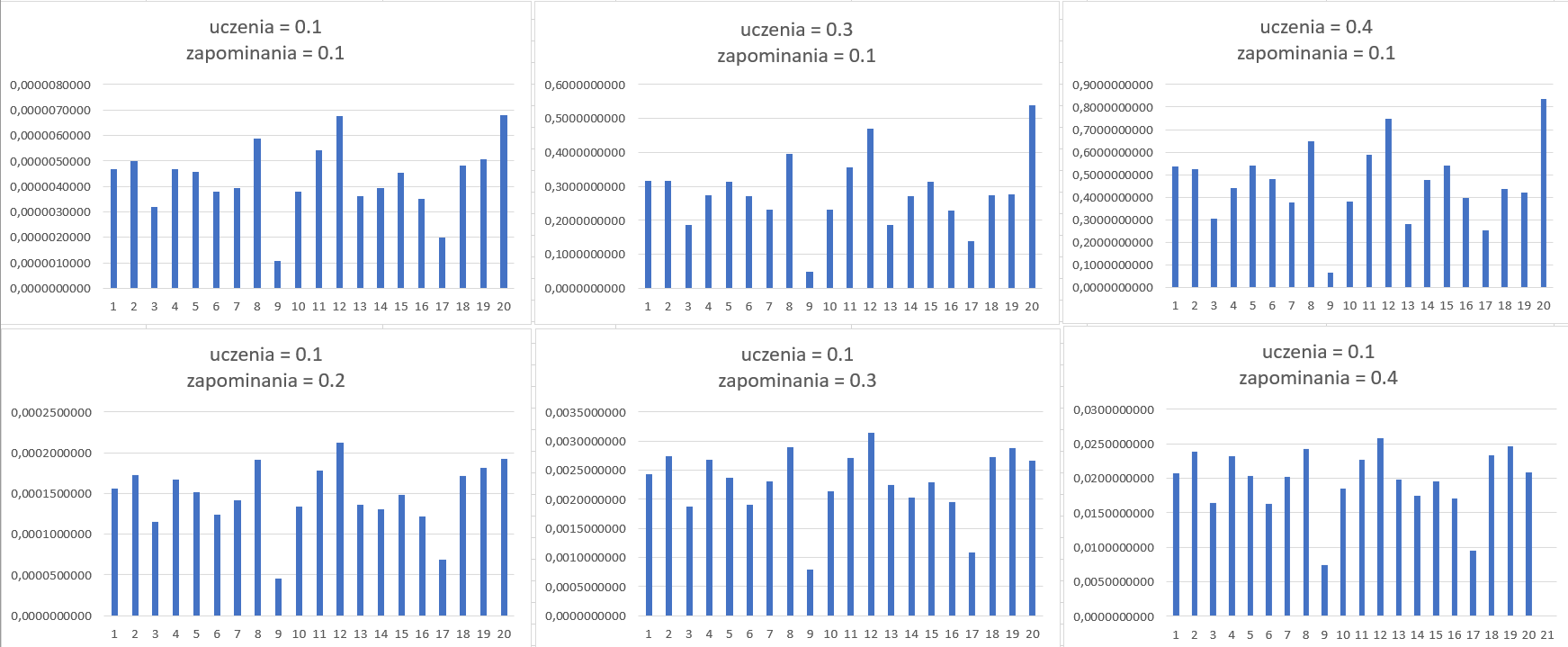
**Zadania do wykonania:**

**1. Wygenerowanie danych uczących i testujących, zawierających 20 dużych liter alfabetu w postaci dwuwymiarowej tablicy 5x7 pikseli dla jednej litery.**

****

**2. Przygotowanie jednowarstwowej sieci oraz reguły Hebba z i bez współczynnika zapominania. Uczenie sieci dla różnych współczynników uczenia i zapominania. Testowanie.**





**3. Obserwacje i wnioski.**

Obserwacje:

* Litera nr 9 (J) najbardziej wyróżnia się na tle wyników uzyskanych u pozostałych liter – jej wartości są najmniejsze.
* Pierwsze trzy wykresy przedstawiają wyniki uzyskane przy stałej wartości współczynnika zapominania i zmieniającej się wartości współczynnika uczenia. Zaobserwowano, że dla wartości współczynnika uczenia większej lub równej 0.5 otrzymywano błędne wyniki.
* Kolejne wykresy przedstawiają wyniki uzyskane przy stałej wartości współczynnika uczenia i zmieniającej się wartości współczynnika zapominania. Litera nr 9 (J) oraz nr 17 (U) należą do odrębnej grupy niż pozostałe litery.

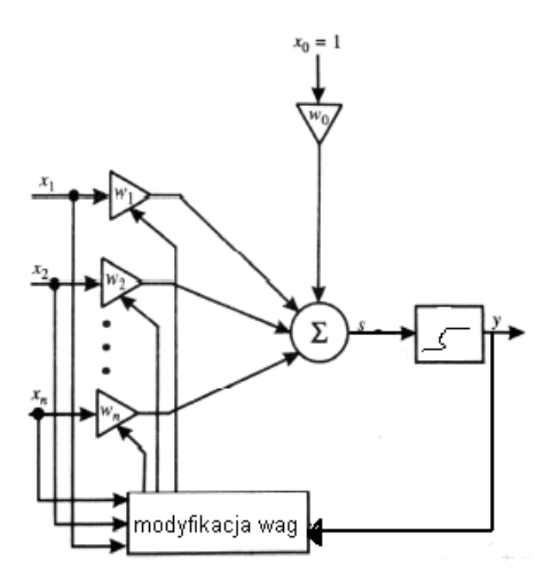
Wnioski:

* Prawidłowy dobór współczynnika uczenia wpływa na poprawność uczenia: np. gdy wartość ta jest zbyt duża otrzymywane wyniki są błędne.
* Współczynnik zapominania pozwala na odpowiednie zmiany wag, tak aby nie osiągały ogromnych wartości. Bez stosowania współczynnika zapominania - wagi wzrastają bardzo gwałtownie, co uniemożliwia ich stabilizację.
* Próby wykonane ze współczynnikiem zapominania były bardziej jednoznaczne i umożliwiały szybkie wyróżnienie grup przynależności liter.

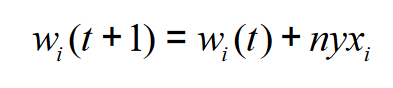
**4. Syntetyczny opis budowy sieci oraz algorytmu uczenia sieci.**

Model Hebba ma identyczną strukturę jak w przypadku modelu typu Adaline, ale charakteryzuje się specyficzną metodą uczenia, znaną pod nazwą *reguły Hebba*. Reguła ta występuje z nauczycielem jak i bez nauczyciela.

Hebb zauważył podczas badać dzaiałania komórek nerwowych, iż połączenie pomiędzy dwiema komórkami jest wzmacniane, jeżeli w tym samym czasie obie komórki są aktywne.



Zaproponował on algorytm, zgodnie z którym modyfikację wag przeprowadza się następująco:



Oznaczenia:

i – numer wagi neuronu

t – numer iteracji w epoce

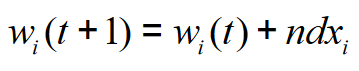
y – sygnał wyjściowy neuronu

x – wartość wejściowa neuronu

n – współczynnik uczenia (0,1)

W przypadku pojedynczego neuronu w trakcie uczenia będziemy modyfikować wartość wag proporcjonalnie zarówno do wartości sygnału podanego na i-te wyjście, jak i sygnału wyjściowego y z uwzględnieniem współczynnika uczenia. W przypadku tym nie podajemy wzorcowej wartości wyjściowej, stosujemy więc tu mętodę uczenia bez nauczyciela.

Niewielka modyfikacja

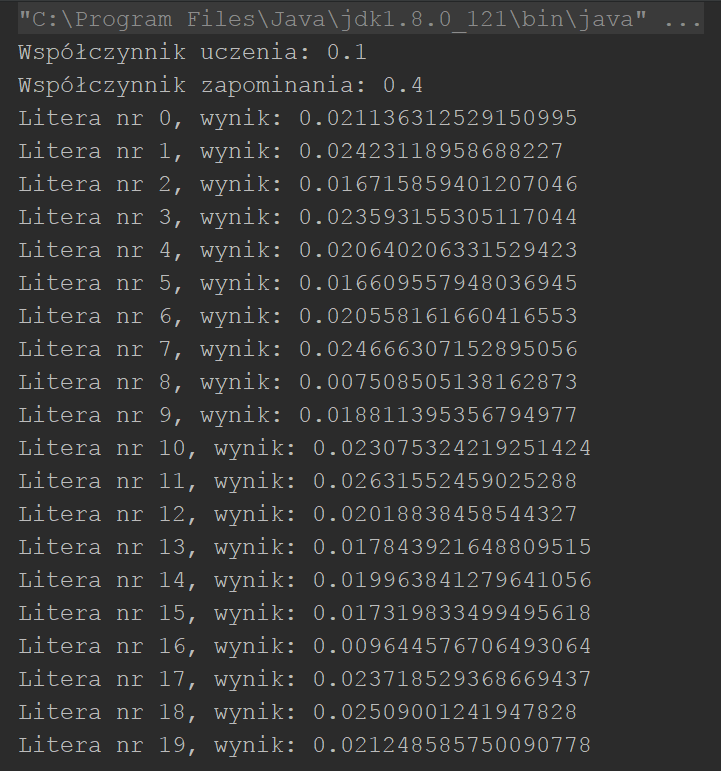


prowadzi do drugiej metody uczenia neuronu Hebba – z nauczycielem

(d – sygnał wzorcowy). Pewną wadą omawianego przez nas algorytmu jest to, iż wartości wag mogą wzrastać do dowolnie dużych liczb.

**5. Listing kodu.**

Przykładowy wydruk:



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  
 {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  
 {1,0,0,0,1,1,0,0,0,1,1,1,0,0,1,1,0,1,0,1,1,0,0,1,1,1,0,0,0,1,1,0,0,0,1}, // N  
 {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  
 {1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0}, // P  
 {1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0,1,0,1,0,0,1,0,0,1,0,1,0,0,0,1}, // R  
 {0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,0,1,1,1,0,0,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, // S  
 {1,1,1,1,1,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0}, // T  
 {1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, // U  
 {1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,1,0,1,1,0,1,0,1,0,1,0,1,0}, // W  
 {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  
 };  
 Hebba neuronHebba = new Hebba(letters);  
 neuronHebba.learn();  
 neuronHebba.test();  
 }  
}

import java.util.Random;  
  
public class Hebba {  
  
 private int[][] lettersMy;  
 public double weight[];  
  
 double learning\_ratio = 0.1;  
 double forgeting\_ratio = 0.4;  
 private int amountOfLetters = 20;  
  
  
 public Hebba(int[][] letters) {  
 this.lettersMy = letters;  
 this.weight = new double[35];  
  
 Random random = new Random();  
 for (int i = 0; i < weight.length; i++) {  
 weight[i] = random.nextDouble();  
 }  
  
 System.*out*.println("Współczynnik uczenia: " + learning\_ratio);  
 System.*out*.println("Współczynnik zapominania: " + forgeting\_ratio);  
 }  
  
  
 public void learn() {  
 // Wykonuj dla każdej litery uczącej  
 for (int i = 0; i < amountOfLetters; i++) {  
 double membranePotential;  
 int[] oneLetterTab = lettersMy[i]; // Prześlij adres początku tablicy litery  
  
 membranePotential = getMembranePotential(oneLetterTab, weight);  
  
 // Modyfikacja wag wg wzoru  
 for (int j = 0; j < weight.length; j++) {  
 weight[j] = weight[j] \* forgeting\_ratio + learning\_ratio \* activationBlock(membranePotential) \* (oneLetterTab[j] - weight[j]);  
 }  
 }  
 }  
  
 public void test(){  
 // Wykonuj dla każdej litery uczącej  
 for (int i = 0; i <amountOfLetters ; i++) {  
 double result;  
 int[] oneLetterTab = lettersMy[i]; // Prześlij adres początku tablicy litery  
  
 double membranePotential = getMembranePotential(oneLetterTab, weight);  
 result = activationBlock(membranePotential);  
  
 System.*out*.print("Litera nr " + i + ", wynik: "+ result + "\n");  
 }  
 }  
  
 // 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  
 public double activationBlock(double membranePotential){  
 double result = (1 - Math.*exp*(-membranePotential)) / (1 + Math.*exp*(-membranePotential));  
 return result;  
 }  
}