Michał Habigier
GĆP 01
Podstawy Sztucznej Inteligencji

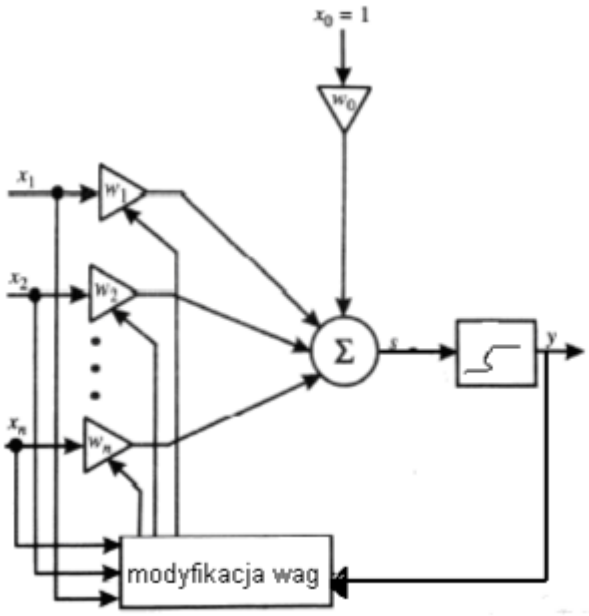
Projekt 4 – Sprawozdanie

1. Cel ćwiczenia:

Celem ćwiczenia jest poznanie działania reguły Hebba na przykładzie rozpoznawania emotikon.

2. Opis budowy wykorzystanego algorytmu:

W ćwiczeniu wykorzystałem jednakowe neurony oparte na modelu sigmoidalnym oraz regułę Hebba w wersji bez nauczyciela.



Ogólny model neuronu Hebba, przedstawiony powyżej odpowiada standardowej postaci modelu neuronu. Waga w_{ij} włączona jest między sygnałem wejściowym y_j a węzłem sumacyjnym i-tego neuronu o sygnale wyjściowym y_i. W przypadku pojedynczego neuronu w trakcie uczenia będziemy modyfikować wartość wag proporcjonalnie zarówno do wartości sygnału podanego na i-te wejście, jak i sygnału wyjściowego y z uwzględnieniem współczynnika uczenia. Ważne , że w przypadku tym nie podajemy wzorcowej wartości wyjściowej. Uczenie neuronu z zastosowaniem reguły Hebba może odbywać się w trybie bez nauczyciela i z nauczycielem. W zastosowanym przez nas trybie bez nauczyciela używa się aktualnej wartości y_i sygnału wyjściowego. Poniżej przedstawiam dwie metody modyfikacji wag

$$\Delta w_i^{(m)(j)} = \eta x_i^{(j)} y_m^{(j)}$$

Ze współczynnikiem zapominania:

$$w_{ij}(k+1) = (1-\gamma) w_{ij}(k) + \Delta w_{ij}$$

Bez współczynnika zapominania:

$$w_{ij}(k+1) = w_{ij}(k) + \Delta w_{ij}$$

Normalizacja polega na podzieleniu każdej ze składowej wektora przez długość tego wektora:

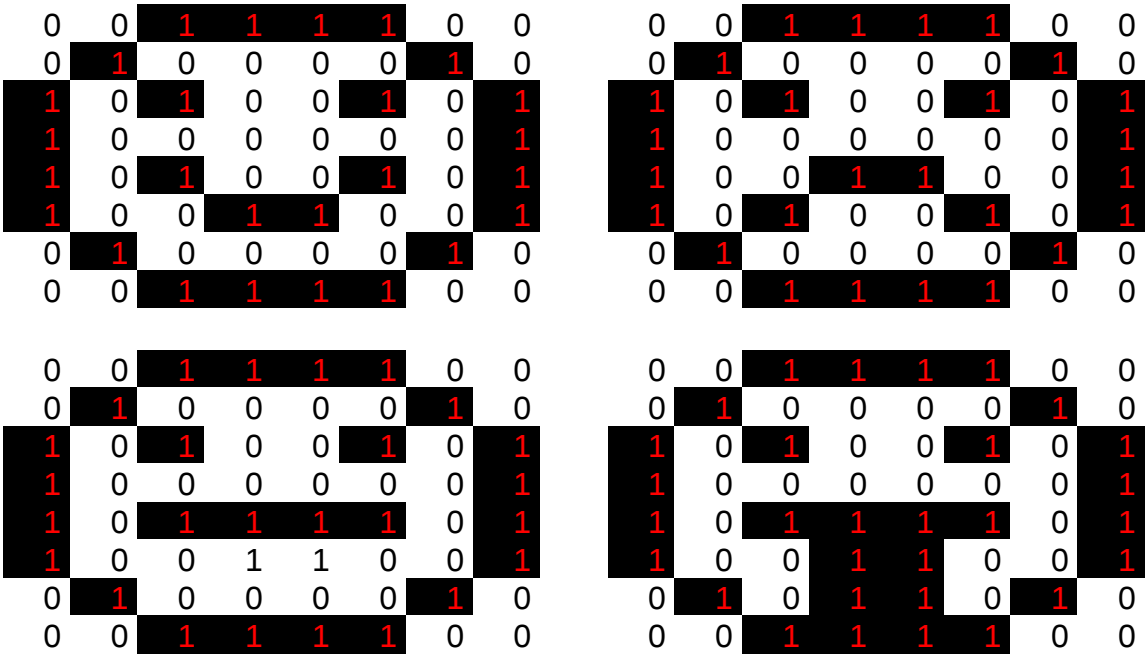
$$\hat{u} = \frac{\vec{u}}{||\vec{u}||}$$

Metoda aktywacji, wykorzystuje funkcje unipolarną sigmoidalną:

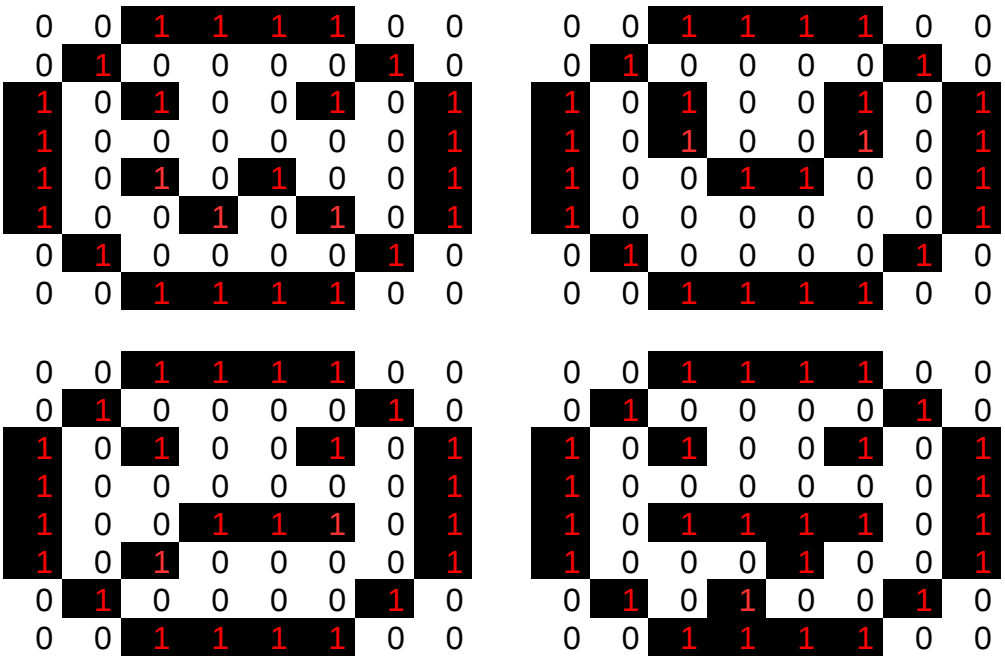
$$f_\beta(x) = \frac{1}{1+e^{-\beta x}}$$

Zestawienie i analiza otrzymanych wyników:

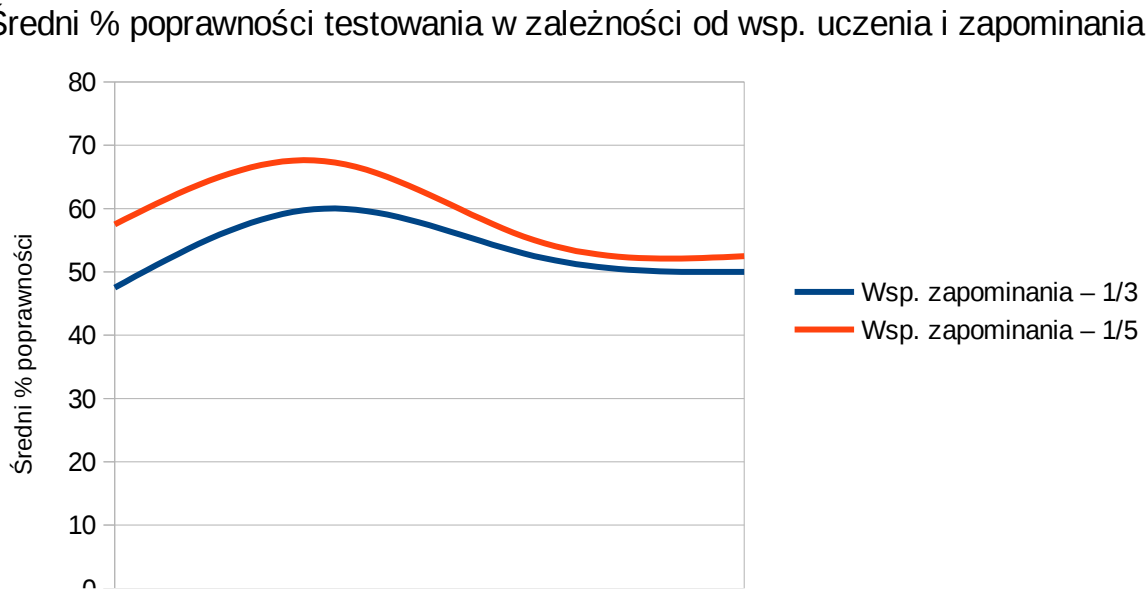
Na potrzeby ćwiczenia utworzyłem kolejno dane uczące oraz do testowania w postaci macierzy 8x8

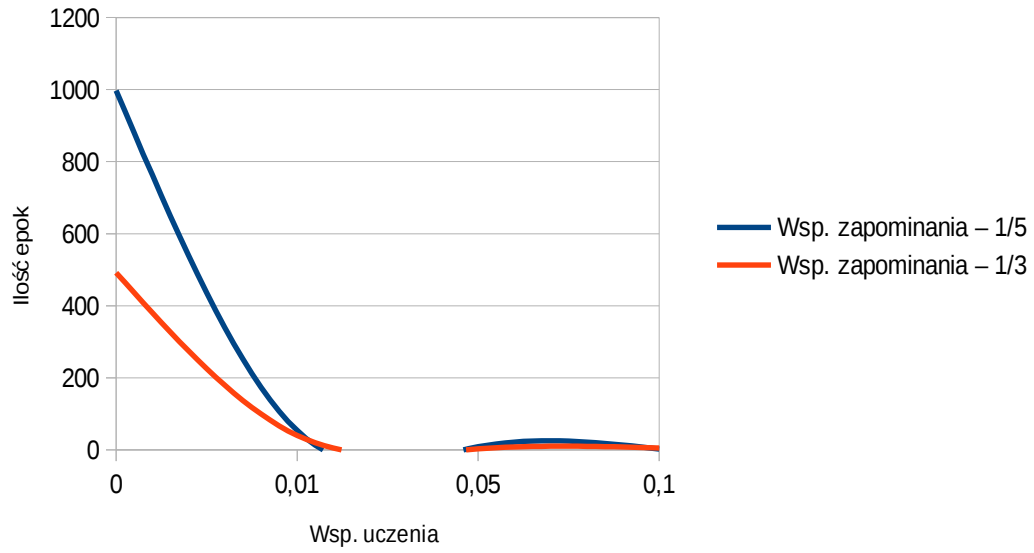


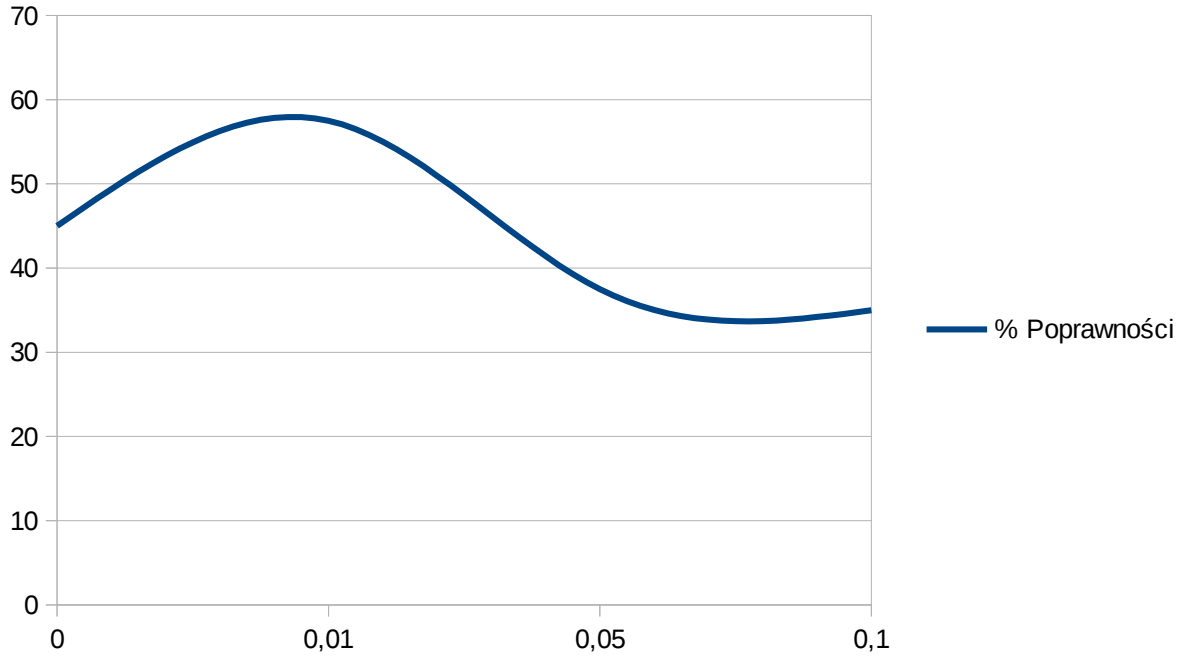
Dane uczące:

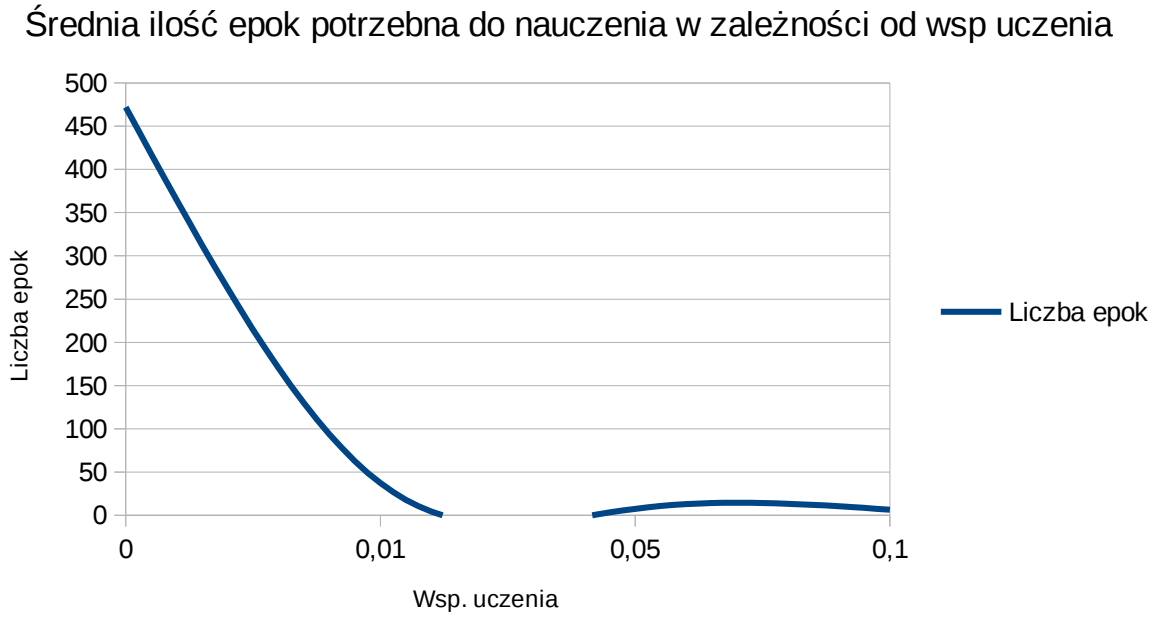


Dane testujące









Jak widać na wykresach ilość epok potrzebna do nauczenia sieci różni się znacząco w poszczególnych przypadkach. Niekiedy wynik wynosił tylko 1 epokę, a innym razem ta liczba przekraczała ponad 1000.
Jeśli chodzi o ocenę jakości uczenia sieci to liczba epok nie jest miarodajnym wynikiem ponieważ zależy ona wyłącznie od początkowych wartości wag neuronów, które są losowe.
Jeśli spojrzymy jednak na wykresy to widać, że wraz ze wzrostem współczynnika uczenia liczba epok potrzebnych do nauczenia sieci stale spada.

Jeśli chodzi o skuteczność uczenia widać, że najlepiej wychodzi ono dla współczynnika 0,01. Zależność ta zachodzi w obu algorytmach. Ponadto lepiej spisał się tutaj współczynnik zapominania wynoszący 0.2 aniżeli 0.33.

Wnioski:

- Na podstawie powyższych wyników można wywnioskować, że należy rozsądnie wybierać współczynnik zapominania. Jeśli współczynnik zapominania będzie zbyt duży spadnie wydajność sieci, gdyż zacznie ona zapominać swoje wyniki.

- Sieć ucząca bez współczynnika zapominania jest mniej wydajna

- Należy zwrócić szczególną uwagę na normalizację wag neuronów, ponieważ bez niej te mogą rosnąć w nieskończoność.

- Podczas testów wystąpiły błędy. Spowodowane były one tym, że sam rozmiar emotikonów jest niewielki. 64 piksele nie pozwalają na dużą różnorodność emotikonów, a nawet mogą być mylące dla neuronów.

Listing kodu:

```
public class Main {
        static int numberOfInputs = 64 + 1;                 //ilość wejść
        static double learningRate = 0.1;                   //współczynnik uczenia się
        static double forgettingRate = learningRate / 5.0;  //współczynnik zapominania
        static int numberOfEmoticon = 4;                        //liczba emotikonów
        static int numberOfNeurons = 5;                     //liczba neuronów

        public static void main ( String[] args ) {

            int winner;
            Hebb[] hebbMethods = new Hebb[numberOfNeurons];
            for ( int i = 0; i < numberOfNeurons; i++ )
                hebbMethods[i] = new Hebb( numberOfInputs );
            System.out.println("\n");
//
            for (int i = 0; i < numberOfEmoticon; i++ ) {
                winner = testHebb(hebbMethods, Emoticon.emoticonToLearn[i] );
                System.out.println( "Zwycięzca przed uczeniem = " + winner );
            }

            int ages = learn(hebbMethods);
            System.out.println("\n");
//
            for (int i = 0; i < numberOfEmoticon; i++ ) {
                winner = testHebb(hebbMethods, Emoticon.emoticonToLearn[i] );
                System.out.println( "Zwycięzca po uczeniu = " + winner );
            }

            System.out.println( "\n\nIlość epok = " + ages );
            System.out.println("\n");
//
            for (int i = 0; i < numberOfEmoticon; i++ ) {
                winner = testHebb(hebbMethods, Emoticon.EmoticonToTest[i] );
                System.out.println( "Zwycięzca po testowaniu = " + winner );
            }

        }
```

```
        winners[i] = -1;
    while ( ! isUnique( winners ) ) {
        for ( int j = 0; j < numberOfNeurons; j++ ) {
            //uczenie neuronów każdej emotikony
            for (int k = 0; k < numberOfEmoticon; k++ )
                hebbMethods[j].learnMethod( Emoticon.emoticonToLearn[k], learningRate, forgettingRate, Hebb.hebbUnforgetting);
            //tesotowanie sieci
            for (int l = 0; l < numberOfEmoticon; l++ )
                winners[l] = testHebb(hebbMethods, Emoticon.emoticonToLearn[l] );
        }
        if ( ++ counter == limit )
            break;
    }
    return counter;
}

//funkcja pomocnicza w procesie uczenie
//zwraca true jeśli każdy element w tablicy jest unikalny
public static boolean isUnique ( int[] winners ) {
    for ( int i = 0; i < numberOfNeurons; i++ )
        for ( int j = 0; j < numberOfNeurons; j++ )
            if ( i != j )
                if ( winners[i] == winners[j] )
                    return false;
    return true;
}

//zwraca wartość zwycięzkiego neuronu dla podanej emotikony
public static int testHebb (Hebb[] hebbMethods, double[] emoticon ) {
    double max = hebbMethods[0].neuronOutput(emoticon);
    int winner = 0;
    for ( int i = 1; i < numberOfNeurons; i++ ) {
        if ( hebbMethods[i].neuronOutput(emoticon) > max ) {
            max = hebbMethods[i].neuronOutput(emoticon);
            winner = i;
        }
    }
    return winner;
}
```