Piotr Wiśniewski

Sprawozdanie 4

Data oddania: 11.01.2018r.

Temat ćwiczenia:

Uczenie sieci regułą Hebba.

Cel ćwiczenia:

Celem ćwiczenia jest poznanie działania reguły Hebba na przykładzie rozpoznawania

emotikon.

Zadania do wykonania

1. Wygenerowanie danych uczących i testujących, zawierających 4 różne emotikony np. czarno-białe, wymiar 8x8 pikseli dla jednej emotikony.

2. Przygotowanie sieci oraz reguły Hebba w wersji z i bez współczynnika zapominania.

3. Uczenie sieci dla różnych współczynników uczenia i zapominania.

4. Testowanie sieci.

Syntetyczny opis budowy oraz wykorzystanego algorytmu uczenia:

Do zrealizowania ćwiczenia wykorzystałem neurony o strukturze modelu sigmoidalnego z metodą uczenia Hebba. Metoda uczenia występuje w dwóch wersjach – z nauczycielem oraz bez nauczyciela. Sposoby modyfikacji wag opisane są wzorami:

**Δwij = learning\_rate \* yj \* yi** gdzie:

1. Learning\_rate – współczynnik uczenia
2. yj – sygnał wejściowy
3. yi – sygnał wyjściowy
4. Ze współczynnikiem zapominania:

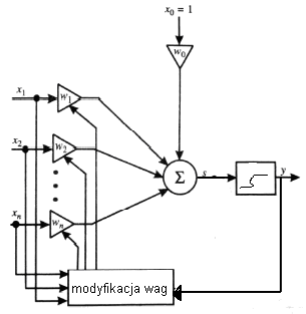
**wij(k+1) = (1-γ) \***  **wij(k) + Δwij** gdzie:

1. γ – współczynnik zapominania
2. Bez współczynnika zapominania:

**wij(k+1) = wij(k) + Δwij**

Modyfikacja wag jak widać w powyższych wzorach zależna jest od sygnału podanego na wejściu jak i sygnału wyjściowego.

Schemat modelu Hebba przedstawiony jest następująco:



Dodatkowo podczas uczenia normalizuję wagi, aby zapobiec nieustającemu ich wzrostowi.

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



Realizacja zadania:

Klasa Hebb składa się z następujących metod:

Metoda ***active,*** wykorzystuje unipolarną sigmoidalną funkcję aktywacji:



Metoda ***sum,*** zwraca sumę iloczynów wag oraz sygnałów wejściowych:

y = ∑ wj \* xj

Metoda ***learn,*** uczenie poprzez modyfikację wag neuronu, zarówno w wersji ze współczynnikiem zapominania, jak i bez niego.

**wij(k+1) = (1-γ) \***  **wij(k) + Δwij**

**wij(k+1) = wij(k) + Δwij**

Metoda ***test,*** zwraca sygnał wyjściowy neuronu.

Metoda ***normalize,*** normalizuje wagi neuronu, czyli zapobiega nadmiernemu wzrostowi wag. Realizuje się to za pomocą wzoru:

Użyte emotikony:

Emotikony stworzony na tablicach 8x8, ich odwzorowanie w programie to tablica jedynek i zer.

(1 oznacza czarne pole, 0 białe).

Zestaw uczący:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

Zestaw testujący:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

Dane do testów wykonane zostały na podstawie danych uczących , jednak dodane zostało zaszumienie dla maksymalnie dwóch pikseli aby odpowiednio dobrać rozmiar zaburzeń do wielkości macierzy.

Proces uczenia i testowania wykonałem zarówno dla wersji modyfikacji wag ze współczynnikiem

zapominania jak i bez niego. Dla obu wersji wykonałem po kilka testów dla różnych

współczynników uczenia oraz zapominania.

Zestawienie otrzymanych wyników:

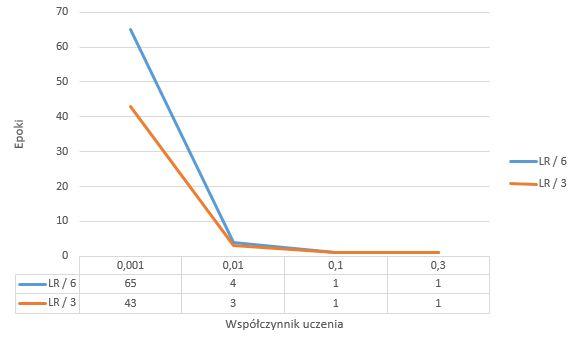
Wersja ze współczynnikiem zapominania

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Współczynnik uczenia | 0.001 | | 0.01 | | 0.1 | | 0.3 | |
| Lp. | Współczynnik zapominania | LR / 6 | LR / 3 | LR / 6 | LR / 3 | LR / 6 | LR / 3 | LR / 6 | LR / 3 |
| 1 | % poprawności [%] | 100 | 25 | 50 | 75 | 75 | 25 | 50 | 50 |
| Ilość epok | 123 | 101 | 6 | 3 | 2 | 15 | 1 | 1 |
| 2 | % poprawności [%] | 75 | 75 | 75 | 75 | 50 | 25 | 25 | 50 |
| Ilość epok | 65 | 192 | 114 | 9 | 2 | 2 | 6 | 1 |
| 3 | % poprawności [%] | 25 | 75 | 50 | 25 | 50 | 50 | 75 | 50 |
| Ilość epok | 119 | 43 | 17 | 115 | 3 | 2 | 2 | 1 |
| 4 | % poprawności [%] | 50 | 50 | 75 | 75 | 75 | 75 | 25 | 50 |
| Ilość epok | 111 | 125 | 14 | 7 | 2 | 2 | 1 | 3 |
| 5 | % poprawności [%] | 50 | 100 | 75 | 75 | 50 | 50 | 50 | 25 |
| Ilość epok | 121 | 127 | 16 | 7 | 3 | 2 | 2 | 1 |
| 6 | % poprawności [%] | 50 | 75 | 50 | 25 | 50 | 50 | 75 | 50 |
| Ilość epok | 112 | 98 | 4 | 14 | 1 | 1 | 6 | 6 |
| 7 | % poprawności [%] | 50 | 25 | 25 | 50 | 75 | 100 | 75 | 50 |
| Ilość epok | 77 | 82 | 8 | 117 | 1 | 1 | 1 | 1 |
| 8 | % poprawności [%] | 50 | 75 | 50 | 75 | 75 | 25 | 75 | 25 |
| Ilość epok | 90 | 156 | 8 | 10 | 1 | 14 | 1 | 2 |
| 9 | % poprawności [%] | 100 | 75 | 25 | 25 | 75 | 50 | 75 | 25 |
| Ilość epok | 78 | 81 | 25 | 13 | 3 | 2 | 2 | 1 |
| 10 | % poprawności [%] | 50 | 25 | 50 | 50 | 50 | 75 | 75 | 50 |
| Ilość epok | 97 | 75 | 112 | 6 | 14 | 2 | 1 | 1 |

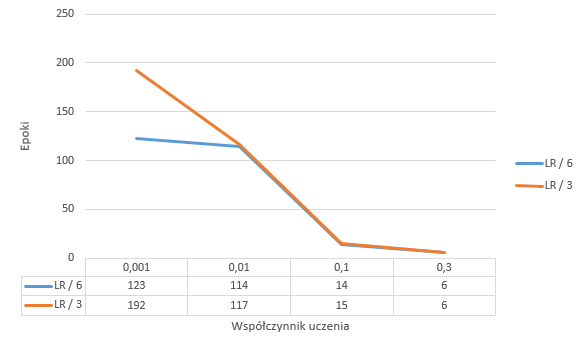
Wersja bez współczynnika zapominania

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lp. | Współczynnik uczenia | 0.001 | 0.01 | | 0.1 | | 0.3 | |
| 1 | % poprawności [%] | 100 | 75 | 75 | | 75 | |
| Ilość epok | 79 | 112 | 2 | | 1 | |
| 2 | % poprawności [%] | 50 | 25 | 75 | | 75 | |
| Ilość epok | 19 | 5 | 1 | | 1 | |
| 3 | % poprawności [%] | 75 | 100 | 75 | | 50 | |
| Ilość epok | 85 | 107 | 2 | | 1 | |
| 4 | % poprawności [%] | 75 | 75 | 75 | | 50 | |
| Ilość epok | 41 | 114 | 2 | | 1 | |
| 5 | % poprawności [%] | 50 | 75 | 50 | | 75 | |
| Ilość epok | 127 | 117 | 1 | | 1 | |
| 6 | % poprawności [%] | 50 | 75 | 75 | | 75 | |
| Ilość epok | 100 | 6 | 1 | | 6 | |
| 7 | % poprawności [%] | 50 | 50 | 75 | | 100 | |
| Ilość epok | 150 | 10 | 1 | | 2 | |
| 8 | % poprawności [%] | 75 | 50 | 75 | | 50 | |
| Ilość epok | 121 | 9 | 2 | | 1 | |
| 9 | % poprawności [%] | 50 | 75 | 50 | | 75 | |
| Ilość epok | 31 | 9 | 2 | | 6 | |
| 10 | % poprawności [%] | 100 | 75 | 25 | | 75 | |
| Ilość epok | 52 | 6 | 2 | | 1 | |

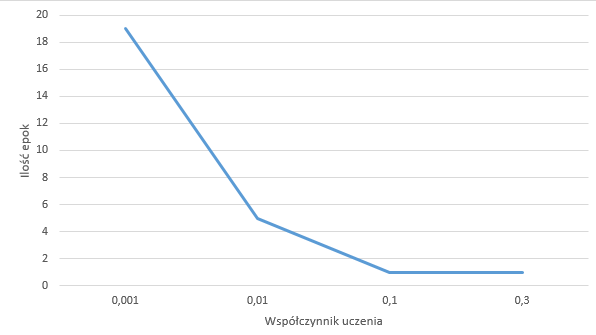
Minimalna ilość epok potrzebnych do nauczenia ze współczynnikiem zapominania



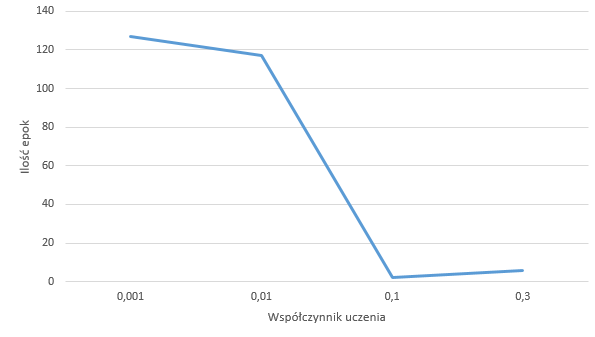
Maksymalna ilość epok potrzebna do nauki ze współczynnikiem zapominania



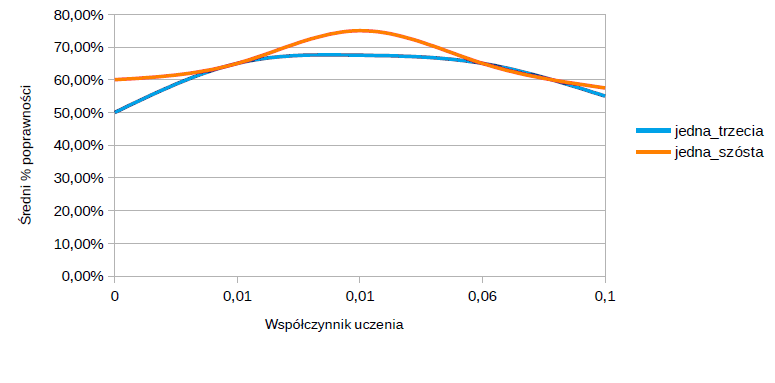
Minimalna ilość epok potrzebna do nauki bez współczynnika zapominania



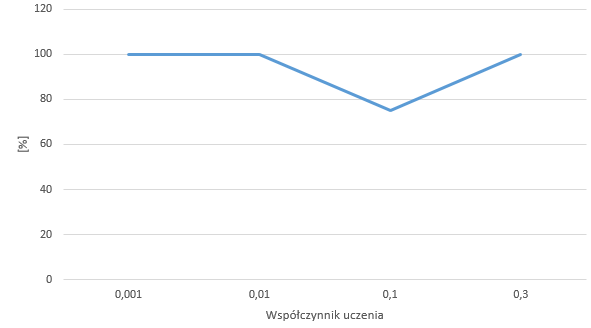
Maksymalna ilość epok potrzebna do nauki bez współczynnika zapominania



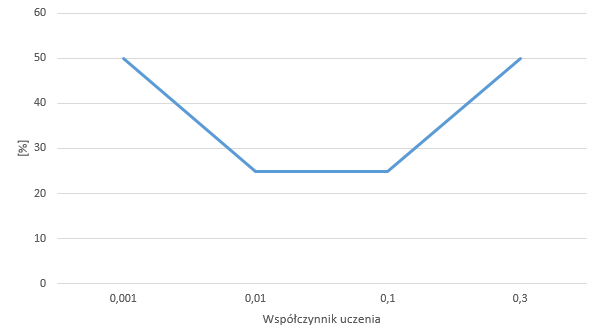
Średni % poprawności testowania w zależności od współczynnika uczenia i współczynnika zapominania



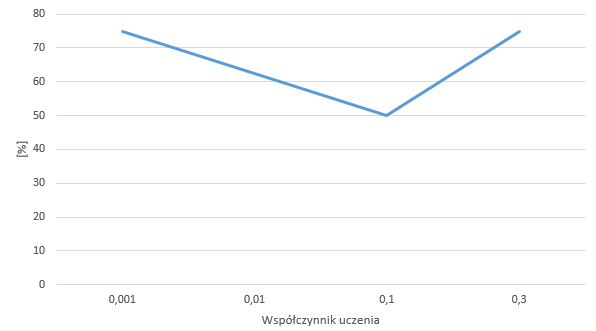
Maksymalna poprawność nauki bez współczynnika zapominania



Minimalna poprawność nauki bez współczynnika zapominania



Średnia poprawność nauki bez współczynnika zapominania



Analiza i dyskusja błędów uczenia i testowania oraz wyników rozpoznawania dla opracowanej sieci w zależności od wartości współczynnika uczenia i zapominania

Po przeanalizowaniu otrzymanych wyników od razu można stwierdzić, że ilość epok jaka była potrzebna do nauczenia sieci znacząco różniła się w poszczególnych przypadkach. Niekiedy wynik wynosił tylko 1 epokę, a niekiedy było to ponad 200 epok. Ciężko przy takiej rozbieżności jednoznacznie stwierdzić skuteczność nauki sieci na podstawie współczynnika nauczania. Co więcej Podczas testów zdarzały się przypadki w których sieć nie była w stanie się nauczyć natomiast w następnym kroku sieć była nauczona w kilku lub kilkunastu krokach. Jest to spowodowane

losowością początkowych wartościami wag neuronów. Jednak wraz ze wzrostem współczynnika nauczania ilość epok potrzebnych do nauczenia sieci za każdym razem spada. Niezależnie od opcji ( z ustawionym bądź nieustawionym współczynnikiem zapominania ) jego zmniejszenie pogarsza otrzymywane wyniki. Wpływało to na większą ilość popełnianych błędów.

Na wykresie ze średnim %poprawności uczenia się w zależności od współczynnika uczenia oraz współczynnika zapominania widać wyraźnie, że najlepsze efekty nauczania wychodzą przy współczynniku uczenia równym 0,1. Również rzuca się w oczy przewaga współczynnika zapominania o wartości 1/6 nad współczynnikiem o wartości 1/3.

Wnioski:

Zaprezentowana sieć neuronowa nie w każdym przypadku była w stanie nauczyć się wprowadzonych emotikon dlatego, aby zapobiec nieskończonemu wykonywaniu się programu został narzucony przeze mnie limit maksymalnej ilości 1000 epok. Zastosowano również normalizację wag, która zabezpiecza nas przed nieskończonym wzrostem wartości wag.

Na podstawie powyższych wyników można śmiało wnioskować, iż najlepsze wyniki można uzyskać stosując metodę modyfikacji wag ze współczynnikiem zapominania. Najlepszy wynik osiągnęła sieć o współczynniku uczenia równym 0,1 oraz o współczynniku zapominania równym 1/6 wartości.

Jeżeli chodzi o to która z wersji dawała lepsze rezultaty to opierając się na szczegółowej analizie stwierdzam, że nieznacznie lepsze wyniki dawała opcja z ustawionym współczynnikiem zapominania. Jednak największy wpływ na ilość epok niezbędnych do nauki miał współczynnik uczenia. Dobranie odpowiedniej wartości współczynnika zapominania jest bardzo istotną kwestią, ważne, aby ta wartość nie była zbyt duża, ponieważ sieć zapominałaby szybko tego czego się nauczyła.

Przy analizowaniu błędnych wyników można również dojść do wniosku, że sieć ma trudność z rozpoznaniem emotikon na obszarze 8x8. Prawdopodobnie gdyby macierz była większa sieć lepiej poradziłaby sobie z problemem z powodu zmniejszonego zaszumienia danych.

Listing kodu:

**public class Emoji {**

**public static double[][] emoji = {**

***//pierwszy input to bias***

**{ 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,**

**0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0 }, // :)**

**{ 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,**

**1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0 }, // :(**

**{ 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,**

**0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0 }, // :|**

**{ 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,**

**1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0 }, // :D**

**};**

**public static double[][] emojiNoised = {**

***//pierwszy input to bias***

**{ 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,**

**0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0 }, // :)**

**{ 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,**

**1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0 }, // :(**

**{ 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0,**

**0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0 }, // :|**

**{ 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,**

**1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0 }, // :D**

**};**

**public static String[] emojiType = { ":)", ":(", ":|", ":D" };**

**}**

**import java.util.Random;**

**public class Hebb {**

**private int noi; *//ilość wejść***

**private double[] w; *//wagi***

**public static boolean HEBB\_WITH\_FORGETTING = true; *//flaga do uczenia ze współczynnikiem zapominania***

**public static boolean HEBB\_WITHOUT\_FORGETTING = false; *//flaga do uczenia bez współczynnika zapominania***

**public Hebb ( int numbers\_of\_inputs ) {**

**noi = numbers\_of\_inputs;**

**w = new double[noi];**

**for ( int i = 0; i < noi; i++ )**

**w[i] = new Random().nextDouble(); *//wagi początkowe sa losowane***

**normalize();**

**}**

***//funkcja aktywacji***

**private double active ( double y\_p ) {**

**return ( 1.0 / ( 1 + Math.pow( Math.E, - y\_p ) ) ); *//unipolarna sigmoidalna***

**}**

***//zwraca sumę iloczynów wag i sygnałów wejściowych***

**private double sum(double[] x ) {**

**double y\_p = 0.0;**

**for ( int i = 0; i < noi; i++ )**

**y\_p += x[i] \* w[i];**

**return y\_p;**

**}**

***//uczenie***

**public double learn(double[] x, double lr, double fr, boolean version ) {**

**double y\_p = active( sum( x ) );**

***//w zależności od podanej wersji, nauka będzie z lub bez współczynnika zapominania***

**for ( int i = 0; i < noi; i++ )**

**if ( version ) w[i] = ( 1 - fr ) \* w[i] + lr \* x[i] \* y\_p; *//ze współczynnikiem zapominania***

**else w[i] += lr \* x[i] \* y\_p; *//bez współczynnika zapominania***

**normalize();**

**return active( sum( x ) );**

**}**

***//zwraca output neuronu***

**public double test ( double[] x ) {**

**return active( sum( x ) );**

**}**

***//normalizuje wagi***

**private void normalize() {**

**double dl = 0.0;**

**for ( int i = 0; i < w.length; i++ )**

**dl += Math.pow( w[i], 2 );**

**dl = Math.sqrt( dl );**

**for ( int i = 0; i < w.length; i++ )**

**if ( w[i] > 0 && dl != 0 )**

**w[i] = w[i] / dl;**

**}**

**}**

**public class Main {**

**static int numberOfInputs = 64 + 1; *//ilość wejść (+1 bo bias)***

**static double learningRate = 0.1; *//współczynnik uczenia się***

**static double forgettingRate = learningRate / 6.0; *//współczynnik zapominania***

**static int numberOfEmoji = 4; *//liczba emotikonów***

**static int numberOfNeurons = 5; *//liczba neuronów***

**public static void main ( String[] args ) {**

**int winner;**

**Hebb[] hebbs = new Hebb[numberOfNeurons];**

**for ( int i = 0; i < numberOfNeurons; i++ )**

**hebbs[i] = new Hebb( numberOfInputs );**

**int ages = learn( hebbs );**

**System.out.println( "Nasze emotikony:\n 1. :)\n 2. :(\n 3. :|\n 4. :D" );**

**System.out.println( "PO UCZENIU" );**

**for ( int i = 0; i < numberOfEmoji; i++ ) {**

**winner = testHebb( hebbs, Emoji.emoji[i] );**

**System.out.println( " " + Emoji.emojiType[i] + " - zwycięza = " + winner );**

**}**

**System.out.println( "\nTESTOWANIE" );**

**for ( int i = 0; i < numberOfEmoji; i++ ) {**

**winner = testHebb( hebbs, Emoji.emojiNoised[i] );**

**System.out.println( " " + Emoji.emojiType[i] + " - zwycięzca = " + winner );**

**}**

**System.out.println( "\nIlość epok = " + ages );**

**}**

***//uczenie neuronów***

**public static int learn ( Hebb[] hebbs ) {**

**int counter = 0;**

**int limit = 1000;**

**int[] winners = new int[numberOfNeurons];**

**for ( int i = 0; i < numberOfNeurons; i++ )**

**winners[i] = - 1;**

**while ( ! isUnique( winners ) ) {**

**for ( int j = 0; j < numberOfNeurons; j++ ) {**

***//uczenie neuronów każdej emotikony***

**for ( int k = 0; k < numberOfEmoji; k++ )**

**hebbs[j].learn( Emoji.emoji[k], learningRate, forgettingRate, Hebb.HEBB\_WITH\_FORGETTING );**

***//tesotowanie sieci celem sprawdzenia, czy sieć jest już nauczona***

**for ( int l = 0; l < numberOfEmoji; l++ )**

**winners[l] = testHebb( hebbs, Emoji.emoji[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[] hebbs, double[] emoji ) {**

**double max = hebbs[0].test( emoji );**

**int winner = 0;**

**for ( int i = 1; i < numberOfNeurons; i++ ) {**

**if ( hebbs[i].test( emoji ) > max ) {**

**max = hebbs[i].test( emoji );**

**winner = i;**

**}**

**}**

**return winner;**

**}**

**}**

Bibliografia:

Stanisław Osowski : Sieci neuronowe do przetwarzania informacji.

https://pl.wikipedia.org