**Podstawy Sztucznej Inteligencji – Laboratorium nr 4**

Wykonał: Kamil Wieniecki

Temat ćwiczenia: Uczenie sieci regułą Hebba.

1. Cel ćwiczenia

Celem ćwiczenia jest poznanie działania reguły Hebba na przykładzie rozpoznawania emotikon

1. Realizacja ćwiczenia

Wybrany przeze mnie język programowania do wykonania laboratorium to **Python.**

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:

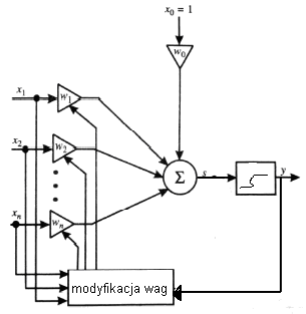
**wij(k+1) = (1-γ) \***  **wij(k) + Δwij** gdzie:

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:



Normalizacja wag (metoda normalize\_weights) polega na podziale każdej składowej wektora przez długość tego wektora, co zapobiega nadmiernemu wzrostowi wag. Wzór jest postaci:



Model Hebba wykorzystuje funkcje aktywacji postaci funkcji unipolarnej sigmoidalnej, która przedstawia się następująco:

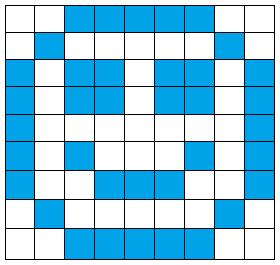
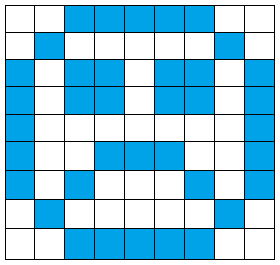
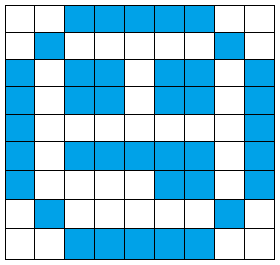
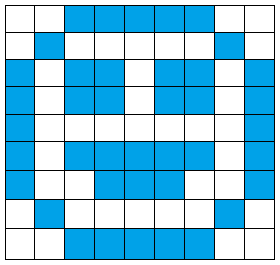
Metoda sumująca klasy Hebb zwraca sumę iloczynów wag oraz sygnałów wejściowych:

**y = ∑ wj \* xj**

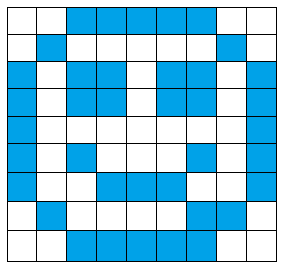
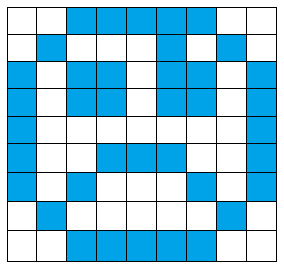
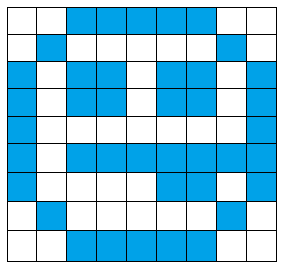
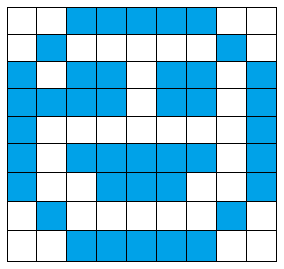
Metoda learn\_without\_supervising uczy poprzez modyfikacje wag neuronu ze współczynnikiem zapominania jak i również bez niego.

Metoda test zwraca sygnał wyjściowy.

Dane wejściowe w postaci wygenerowanych przeze mnie emotikonów przedstawiają się następująco:



Dodałem jeden pixel do każdej emotikony, który powodował zniekształcenie. Szablony z dodatkowym pixelem wyglądają następująco:



Każdy pixel to jeden sygnał wejściowy, więc wejść dla każdego neuronu jest 9x9 + 1 (BIAS) = 82.

Proces uczenia oraz testów przeprowadziłem dla wersji ze współczynnikiem zapominania jak i bez niego. Oto wyniki testów dla różnych wartości współczynników uczenia i zapominania.

1. Wyniki

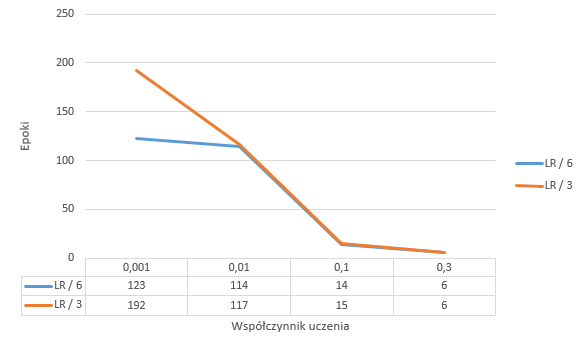
Tabela z modyfikacją wag wraz ze współczynnikiem zapominania.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Learning Rate | 0.001 | | 0.01 | | 0.1 | | 0.3 | |
| Lp. | Forgetting Rate | 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 |

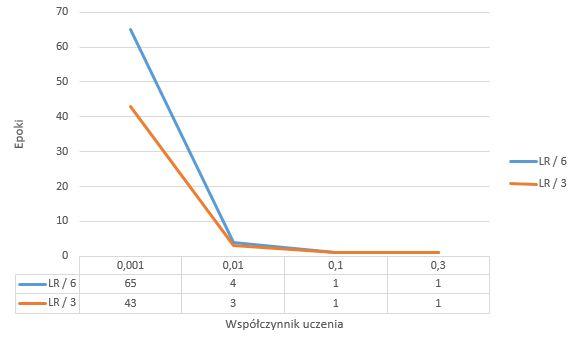
Tabela modyfikacji wag bez współczynnika zapominania

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lp. | Learning Rate | 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 | |

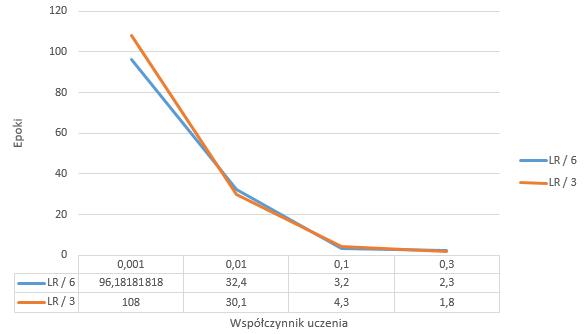
Maksymalna ilość epok potrzebna do nauki ze współczynnikiem zapominania



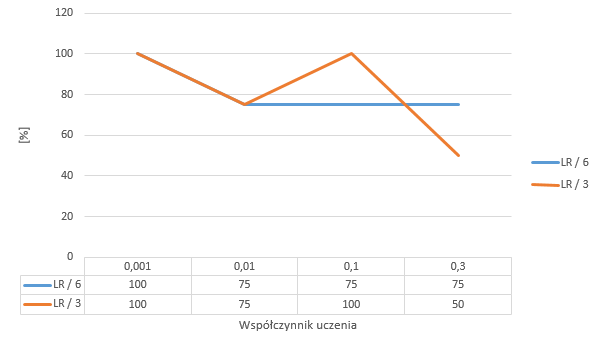
Minimalna ilość epok potrzebna do nauczenia ze współczynnikiem zapominania



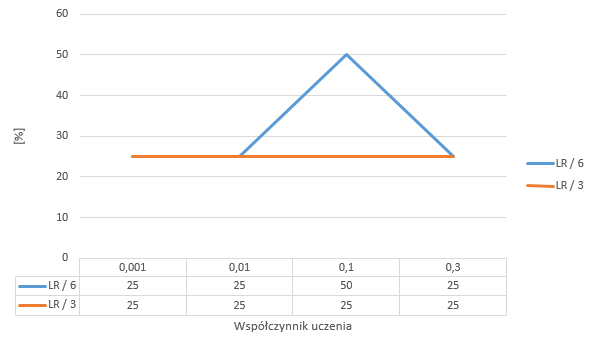
Średnia ilość epok potrzebnych do nauczenia ze współczynnikiem zapominania



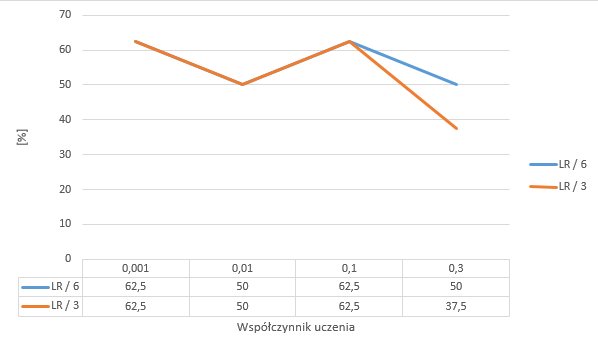
Maksymalna poprawność uczenia sieci ze współczynnikiem zapominania



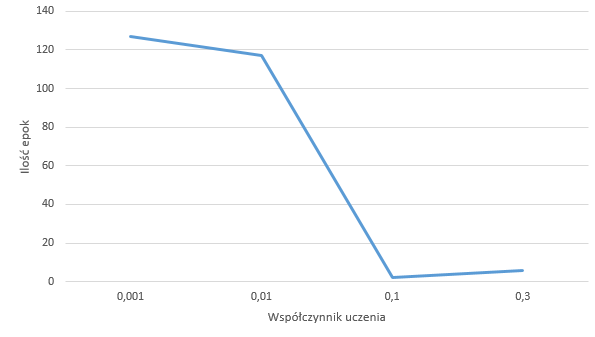
Minimalna poprawność uczenia sieci ze współczynnikiem zapominania



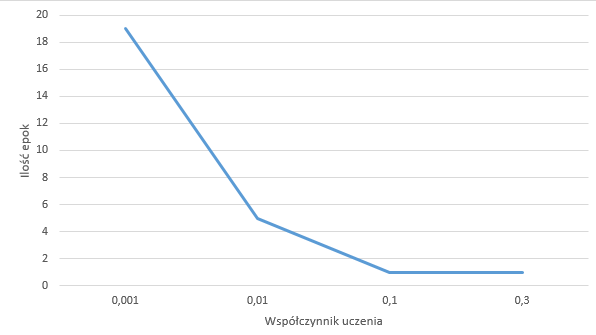
Średnia poprawność uczenia sieci ze współczynnikiem zapominania



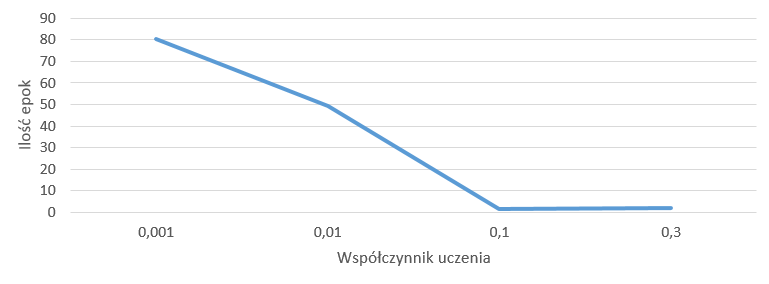
Maksymalna ilość epok potrzebna do nauki bez współczynnika zapominania



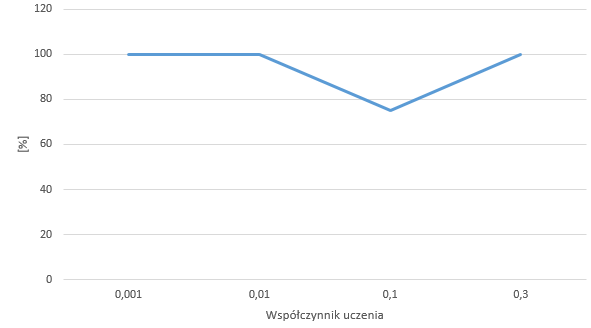
Minimalna ilość epok potrzebna do nauki bez współczynnika zapominania



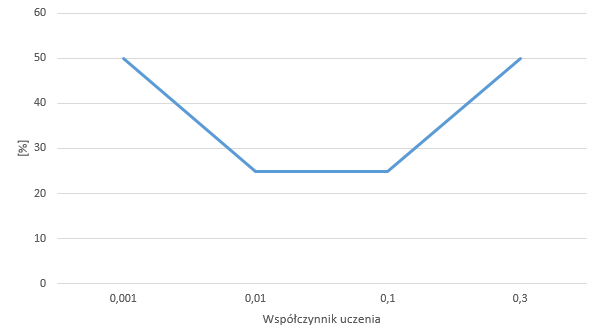
Średnia ilość epok potrzebna do nauki bez współczynnika zapominania



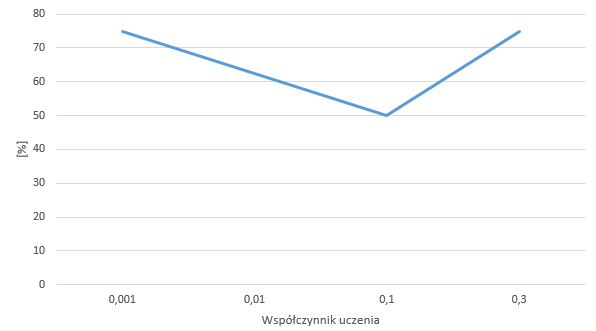
Maksymalna poprawność nauki bez współczynnika zapominania



Minimalna poprawność nauki bez współczynnika zapominania



Średnia poprawność nauki bez współczynnika zapominania



1. Analiza wyników

W przedstawionych wyżej wynikach analizując najpierw model ze współczynnikiem zapominania - maksymalna ilość epok, a minimalna bardzo się waha i są to nieco skrajne wartości. Ciężko przy takiej rozbieżności jednoznacznie stwierdzić skuteczność nauki sieci na podstawie współczynnika nauczania. Losowanie za każdym razem różnych wag zdecydowanie wpływa na wyniki. Jednak wraz ze wzrostem współczynnika nauczania ilość epok za każdym razem spada, jednak większa wartość powoduje też obniżenie poprawności wyników, co sugeruje, że model najlepiej uczyć jest powoli (tzn. mały współczynnik uczenia i wiele epok), wtedy poprawność wyników jest lepsza.

Wartość współczynnika uczenia 0,1 uzyskała najlepsze efekty tzn. dość wysoką poprawność wyników w stosunku do potrzebnej ilości epok. Współczynnik zapominania o wartości 1/6 współczynnika uczenia dał lepsze efekty niż 1/3.

Dla modelu bez współczynnika zapominania ilość epok potrzebna do wyuczenia była mniejsza niż w pierwszym przypadku. Również poprawność nauki prezentowała się nieco lepiej niż w przypadku pierwszym. Także tutaj wartość współczynnika uczenia miała znaczenie i wraz z jej wzrostem ilość potrzebnych epok malała.

1. Wnioski

Czasami sieć nie była w stanie nauczyć się wprowadzonych emotikon dlatego, aby zapobiec nieskończonemu wykonywaniu się programu narzuciłem limit maksymalnej ilości 1000 epok.

Zdecydowanie ilość wejść ma wpływ na rozróżnianie emotikon, ponieważ większa rozdzielczość niż 81 pixeli pozwala na dokładniejsze opisanie ekspresji emotikony. Widać to po tym, że zaszumione emotikony są nadal bardzo podobne – dlatego też dokładność wyników była rozbieżna. Sieć dla większej rozdzielczości uczyła by się bardziej poprawniej, gdyż wtedy emotikony coraz bardziej by się od siebie różniły.

Normalizacja wag jest istotna, ponieważ zabezpiecza przed nieskończonym wzrostem wartości wag.

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.

Nauka przy pomocy modelu Hebba jest całkiem efektywna, daje obiecujące wyniki o całkiem wysokiej dokładności.

1. Listing kodu

**Emoji.py**

emoji = [

[1,

0, 0, 1, 1, 1, 1, 1, 0, 0,

0, 1, 0, 0, 0, 0, 0, 1, 0,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 0, 0, 0, 0, 0, 0, 1,

1, 0, 1, 0, 0, 0, 1, 0, 1,

1, 0, 0, 1, 1, 1, 0, 0, 1,

0, 1, 0, 0, 0, 0, 0, 1, 0,

0, 0, 1, 1, 1, 1, 1, 0, 0

],

[1,

0, 0, 1, 1, 1, 1, 1, 0, 0,

0, 1, 0, 0, 0, 0, 0, 1, 0,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 0, 0, 0, 0, 0, 0, 1,

1, 0, 0, 1, 1, 1, 0, 0, 1,

1, 0, 1, 0, 0, 0, 1, 0, 1,

0, 1, 0, 0, 0, 0, 0, 1, 0,

0, 0, 1, 1, 1, 1, 1, 0, 0

],

[1,

0, 0, 1, 1, 1, 1, 1, 0, 0,

0, 1, 0, 0, 0, 0, 0, 1, 0,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 0, 0, 0, 0, 0, 0, 1,

1, 0, 1, 1, 1, 1, 1, 0, 1,

1, 0, 0, 0, 0, 1, 1, 0, 1,

0, 1, 0, 0, 0, 0, 0, 1, 0,

0, 0, 1, 1, 1, 1, 1, 0, 0

],

[1,

0, 0, 1, 1, 1, 1, 1, 0, 0,

0, 1, 0, 0, 0, 0, 0, 1, 0,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 0, 0, 0, 0, 0, 0, 1,

1, 0, 1, 1, 1, 1, 1, 0, 1,

1, 0, 0, 1, 1, 1, 0, 0, 1,

0, 1, 0, 0, 0, 0, 0, 1, 0,

0, 0, 1, 1, 1, 1, 1, 0, 0

]

]

confused\_emoji = [

[1,

0, 0, 1, 1, 1, 1, 1, 0, 0,

0, 1, 0, 0, 0, 0, 0, 1, 0,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 0, 0, 0, 0, 0, 0, 1,

1, 0, 1, 0, 0, 0, 1, 0, 1,

1, 0, 0, 1, 1, 1, 0, 0, 1,

0, 1, 0, 0, 0, 0, 1, 1, 0,

0, 0, 1, 1, 1, 1, 1, 0, 0

],

[1,

0, 0, 1, 1, 1, 1, 1, 0, 0,

0, 1, 0, 0, 0, 1, 0, 1, 0,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 0, 0, 0, 0, 0, 0, 1,

1, 0, 0, 1, 1, 1, 0, 0, 1,

1, 0, 1, 0, 0, 0, 1, 0, 1,

0, 1, 0, 0, 0, 0, 0, 1, 0,

0, 0, 1, 1, 1, 1, 1, 0, 0

],

[1,

0, 0, 1, 1, 1, 1, 1, 0, 0,

0, 1, 0, 0, 0, 0, 0, 1, 0,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 0, 0, 0, 0, 0, 0, 1,

1, 0, 1, 1, 1, 1, 1, 1, 1,

1, 0, 0, 0, 0, 1, 1, 0, 1,

0, 1, 0, 0, 0, 0, 0, 1, 0,

0, 0, 1, 1, 1, 1, 1, 0, 0

],

[1,

0, 0, 1, 1, 1, 1, 1, 0, 0,

0, 1, 0, 0, 0, 0, 0, 1, 0,

1, 0, 1, 1, 0, 1, 1, 0, 1,

1, 1, 1, 1, 0, 1, 1, 0, 1,

1, 0, 0, 0, 0, 0, 0, 0, 1,

1, 0, 1, 1, 1, 1, 1, 0, 1,

1, 0, 0, 1, 1, 1, 0, 0, 1,

0, 1, 0, 0, 0, 0, 0, 1, 0,

0, 0, 1, 1, 1, 1, 1, 0, 0

]

]

emoji\_type = [":)", ":(", ":P", ":D"]

**Hebb.py**

"Import funkcji losujacej Pythona"

from random import uniform

import math

class Hebb:

"""Klasa HEBB"""

def \_\_init\_\_(self, inputs):

"Konstruktor"

self.inputs = inputs

self.weights = []

for i in range(0, inputs):

self.weights.append(uniform(0, 1))

self.normalize\_weights()

@staticmethod

def activation(y\_p):

"Funkcja aktywacji"

return (1.0 / (1 + math.pow(math.e, - y\_p)))

def sum(self, vector):

"Sumator"

y\_p = 0

for i in range(0, self.inputs):

y\_p += vector[i] \* self.weights[i]

return y\_p

def learn\_without\_supervising(self, vector, learning\_rate, forgetting\_rate, is\_forgetting):

"Nauka bez nauczyciela"

y\_p = self.activation(self.sum(vector))

for i in range(0, self.inputs):

if is\_forgetting:

self.weights[i] = (1 - forgetting\_rate) \* self.weights[i] + learning\_rate \* vector[i] \* y\_p

else:

self.weights[i] += learning\_rate \* vector[i] \* y\_p

self.normalize\_weights()

return self.activation(self.sum(vector))

def test(self, vector):

"Funkcja testujaca i zwracajaca wynik z neuronu"

return self.activation(self.sum(vector))

def normalize\_weights(self):

"Funkcja normalizujaca wagi"

dl = 0

for i in range(0, self.inputs):

dl += math.pow(self.weights[i], 2)

dl = math.sqrt(dl)

for i in range(0, self.inputs):

if self.weights[i] > 0 and dl != 0:

self.weights[i] = self.weights[i] / dl

**Main.py**

from Emoji import emoji

from Emoji import confused\_emoji

from Emoji import emoji\_type

from Hebb import Hebb

HEBB\_FORGETTING = True

HEBB\_WITHOUT\_FORGETTING = False

def learn(hebbs):

"Funkcja uczenia"

era = 0

winners = []

for i in range(0, NEURONS):

winners.append(-1)

while unique(winners) != True:

for i in range(0, NEURONS):

for j in range(0, EMOJI):

hebbs[i].learn\_without\_supervising(emoji[j], LEARNING\_RATE, FORGETTING\_RATE, HEBB\_FORGETTING)

for j in range(0, EMOJI):

winners[j] = test\_hebb(hebbs, emoji[j])

era += 1

if era == LIMIT:

break

return era

def unique(winners):

"Funkcja sprawdza czy elementy w tablicy sa unikalne, pomoc w nauce"

for i in range(0, NEURONS):

for j in range(0, NEURONS):

if i != j:

if winners[i] == winners[j]:

return False

return True

def test\_hebb(hebbs, emoji):

"Funkcja zwraca wartosc zwyciezkiego neuronu dla emotikony"

max = hebbs[0].test(emoji)

winner = 0

for i in range(1, NEURONS):

test = hebbs[i].test(emoji)

if test > max:

max = test

winner = i

return winner

## Dane wejsciowe

INPUTS = 82

LEARNING\_RATE = 0.3

FORGETTING\_RATE = LEARNING\_RATE / 3.0

NEURONS = 5

EMOJI = 4

LIMIT = 1000

print("lr =", LEARNING\_RATE, "forgetting=", FORGETTING\_RATE)

HEBBS = []

for i in range(0, NEURONS):

HEBBS.append(Hebb(INPUTS))

ERAS = learn(HEBBS)

print("Po nauce")

for i in range(0, EMOJI):

winner = test\_hebb(HEBBS, emoji[i])

print("Wygrany dla emotikony", emoji\_type[i], "neuron:", winner)

print("Testowanie")

for i in range(0, EMOJI):

winner = test\_hebb(HEBBS, confused\_emoji[i])

print("Wygrany dla emotikony", emoji\_type[i], "neuron:", winner)

print("Ilosc epok =", ERAS)

1. Bibliografia

<https://en.wikipedia.org/wiki/Hebbian_theory>

<https://pl.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-vectors/a/vector-magnitude-normalization>

<https://pl.wikipedia.org/wiki/Funkcja_aktywacji>

http://www.inf.ed.ac.uk/teaching/courses/dcfc/cecn04.pdf