Podstawy sztucznej inteligencji

Uczenie sieci regułą Hebba

Sprawozdanie ze scenariusza nr 4

1. Cel ćwiczenia

Celem omawianego ćwiczenia było poznawanie uczenia sieci regułą Hebba dla problemu rozpoznawania emotikon

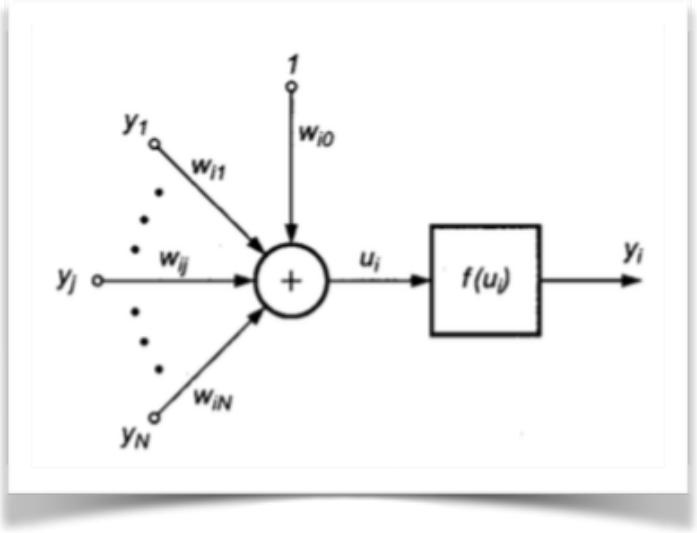
2. Schemat ćwiczenia

1. Zaimplementowanie grupy neuronów Hebba
2. Uczenie sieci wygenerowanymi emotikonami
3. Sprawdzenie wyuczonej sieci zaszumionymi emotikonami

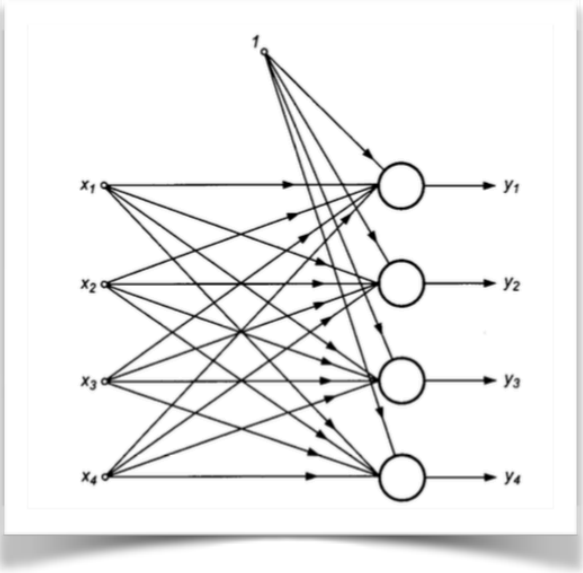
3. Wykonanie ćwiczenia

Do zaimplementowania neuronów Hebba wykorzystane zostały informacje z książki[1]. Sieci Hebba stosowane są do klasyfikowania zbiorów danych. Sieć neuronowa Hebba zauważa najmocniejsze powiązania wag z wejściami. Sieć ta uczy się bez nauczyciela.

Ogólny schemat neuronu:



Schemat sieci:



W regule Hebba wagi aktualizujemy w następujący sposób:

$$w_{ij}(k+1) = (1-\gamma)w_{ij}(k) + \eta y_j y_i$$

Gdzie:

y_i - wyjście neuronu

y_j - j-te wejście neuronu

w_{ij} - j-ta waga i-tego neuronu

γ - współczynnik zapominania - z przedziału (0,1)

η - współczynnik uczenia - z przedziału (0,1)

k - numer iteracji

Sygnał wyjściowy z neuronu oblicza się według następującej reguły:

---

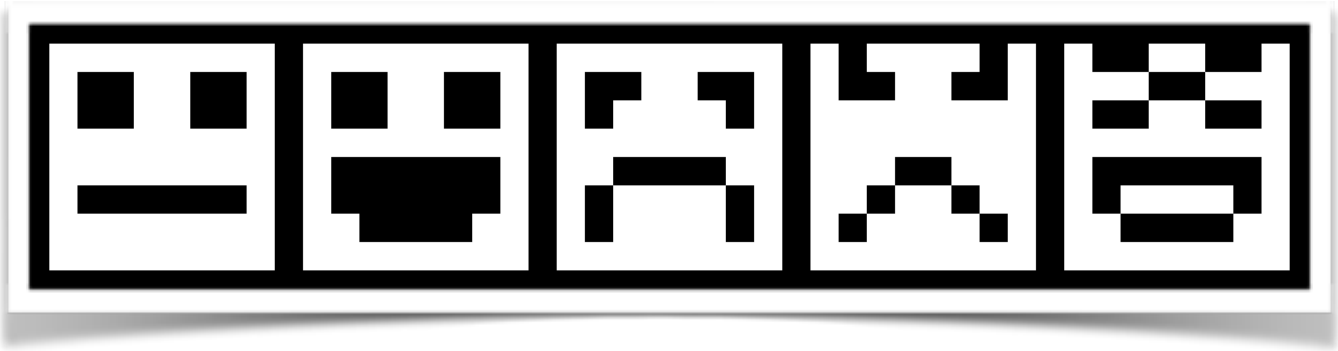
[1] Stanisław Osowski - *Sieci neuronowe do przetwarzania danych* str. 34-37

$$y = \sum_j w_j x_j$$

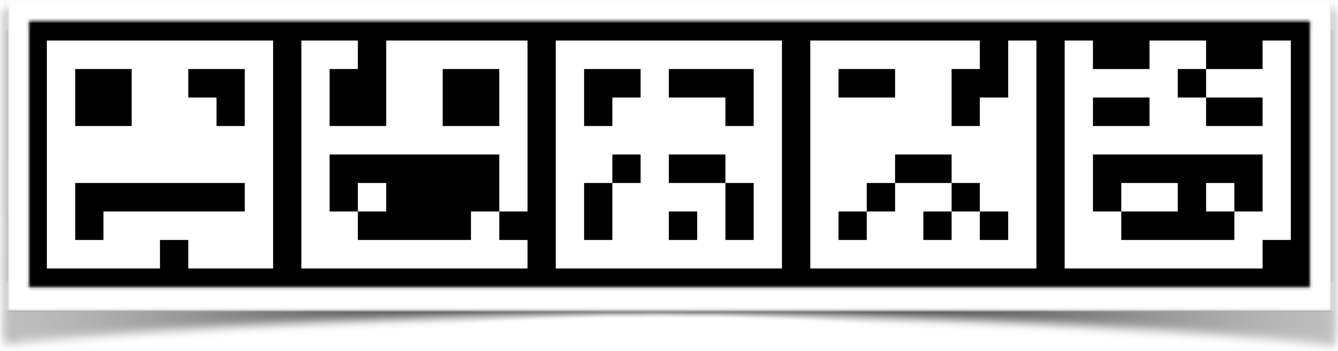
Charakterystyczną cechą neuronu Hebba jest to że wagi rosną do bardzo dużych wartości. Właśnie z tego powodu wprowadzany jest współczynnik zapominania.

W trakcie uczenia z całej sieci wybierany był jeden neuron o najwyższym sygnale wyjściowym dla danej emotikony, następnie uruchamiana była procedura aktualizacji wagi dla wybranego neuronu.

Zestaw emotikon do uczenia sieci:



Przykładowy zaszumiony 3 pikselami zestaw:



Po wybraniu zestawów do uczenia i sprawdzania sieci podjęto próby dobrania odpowiedniego learning rate oraz forget rate do nauki sieci.

Przetestowano następujące ilości neuronów Hebba:

- 5
- 30
- 100

W następujących konfiguracjach lr i fr:

LR	FR
0,01	0,3
0,007	0,4
0,005	0,4

Sieć uznawałem za działającą poprawnie wtedy gdy ten sam neuron odgadywał emotikonę zaszumioną minimalną liczbą 3 pikseli oraz traktowałem jako błąd sieć która jednym neuronem odgadywała więcej niż 1 emotikonę.

4. Wyniki

Ilość neuronów	LR/FR	Ilość epok	% poprawności	Ilość dubli
5	0,01/0,3	100	60%	1
		200	60%	1
		500	40%	1
		1000	20%	1
	0,007/0,4	100	60%	1
		200	40%	1
		500	40%	1
		1000	60%	1
	0,005/0,4	100	60%	0
		200	60%	0
		500	60%	0
		1000	60%	0

Ilość neuronów	LR/FR	Ilość epok	% poprawności	Ilość dubli
30	0,01/0,3	100	60%	0
		200	60%	1
		500	40%	1
		1000	40%	1
	0,007/0,4	100	100%	0
		200	60%	1
		500	60%	1
		1000	40%	1
	0,005/0,4	100	60%	1
		200	80%	0
		500	60%	0
		1000	80%	0

Ilość neuronów	LR/FR	Ilość epok	% poprawności	Ilość dubli
100	0,01/0,3	100	40%	0
		200	60%	0
		500	40%	0
		1000	40%	1
	0,007/0,4	100	80%	0
		200	80%	0
		500	100%	0
		1000	60%	1
	0,005/0,4	100	40%	0
		200	60%	0
		500	60%	0
		1000	100%	0

5. Omówienie wyników

Podczas trenowania i testowania sieci przy różnych konfiguracjach występowały 2 problemy:
- Ten sam neuron zgadywał 2 różne emotikony
- Inny neuron odgadywał zaszumioną emotikonę a inny niezaszumioną

Na podstawie powyższych wyników zauważyć można że najlepszą konfiguracją dla reguły Hebba jest ta w której znajduje się 100 neuronów a learning rate i forget rate ustawione są na odpowiednio na 0,007 i 0,4 neurony.

Zauważono również że sieć 30 neuronów łatwiej jest przetrenować i dokładność odgadywania emotikon spada wraz ze wzrostem liczby epok.

W sieci z 100 neuronami dzieje się analogicznie jak w 30, jednakże potrzeba do tego dużo większej liczby epok.

Sieci z learning rate 0,005 i forget rate równym 0,4 miały najmniej tych samych samych neuronów dla dwóch różnych emotikon

6. Wnioski

Na podstawie wyników zauważono że najlepszą konfiguracją dla danej implementacji jest liczba 100 neuronów w sieci oraz learning rate 0,007 i forget rate 0,4. Jednakże dla 30 neuronów i tych samych ustawień współczynników można zauważyć że sieć jest w stanie nauczyć się jeszcze szybciej.

Niestety forget rate wyższy od learning rate może świadczyć o błędzie w implementacji lub braku normalizacji. Można domyślać się że wynika to z liniowej funkcji aktywacji i tego że wagi rosną

bardzo szybko i właściwie nie istnieje limit liczbowy przez co sieć musi sporo „zapomnieć” w trakcie uczenia aby wagi były bardziej dopasowane.

Dobór odpowiedniej kombinacji learning i forget rate są kluczowe w sieciach neuronowych, przy niektórych konfiguracjach sieć uczyła się bardzo mało lub wcale.

7. Listing kodu

Plik main.py

```
from HebbGroup import *
from testinput import *
from signsigm import *
import numpy as np
import copy

"""Funkcja drukująca emoji na ekran"""
def drawEmoji(emoji):
    for i in range(8):
        for j in range(8):
            print(' ' if emoji[i*8+j] == -1 or emoji[i*8+j] == 0 else 'x', end=' ', flush=True)
        print("\n")

"""Funkcja zaszumiająca losowe piksele w emoji"""
def noiseEmoji(emoji, numOfNoisePixels):
    noisedEmoji = copy.deepcopy(emoji)
    pixels = np.random.choice(64, numOfNoisePixels, replace=False)
    for pixel in pixels:
        if noisedEmoji[pixel] == 1:
            noisedEmoji[pixel] = 0
        else:
            noisedEmoji[pixel] = 1

    return noisedEmoji


if __name__ == '__main__':
    #ustawienie parametrów sieci i uczenia
    no_of_inputs = 64
    learning_rate = 0.008
    forget_rate = 0.25
    num_of_neurons = 30
    epochCnt = 400


    activation_function = Linear()    #ustawienie funkcji aktywacji dla neuronów
    testInput = TestInput()             #wygenerowanie danych do uczenia
    testInputMap = testInput.getInputsMap()
    noisedInputMap = {}

    for key in testInputMap.keys():     #stworzenie zaszumionych emotikon do testów
        noisedInputMap[key] = noiseEmoji(testInputMap[key], 3)

    for key in testInputMap.keys():     #wydruk emoji
        print(str(key))
        drawEmoji(testInputMap[key])
        print("NOISED:")
        drawEmoji(noisedInputMap[key])

    #stworzenie struktury sieci Hebba
    neuronGroup = HebbGroup(learning_rate, num_of_neurons, no_of_inputs, forget_rate, activation_function)

    #uczenie sieci
    for i in range(epochCnt):
        for key in testInputMap.keys():
            neuronGroup.train_without_supervisor(testInputMap[key])

    #sprawdzenie sieci
    winners = {}
    for key in testInputMap.keys():
        winners[key] = neuronGroup.guess(testInputMap[key])

    #sprawdzenie sieci zaszumionymi emoji
    winnersNoised = {}
    for key in noisedInputMap.keys():
        winnersNoised[key] = neuronGroup.guess(noisedInputMap[key])

    #wydruki końcowe
    print("Emoji", "\t", "Norm", "\t", "Noised")
    print("-----------------------")
    for key, winner in winners.items():
        print(key, "\t", winner._iid, "\t", winnersNoised[key]._iid)
```

Plik HebbGroup.py

```
from Neuron import *
class HebbGroup:
    """Inicjalizacja sieci"""
    def __init__(self, learning_rate, no_of_neurons, no_of_inputs, forget_rate, activation_function):
        self._no_of_inputs = no_of_inputs
        self._no_of_neurons = no_of_neurons
        self._learning_rate = learning_rate
        self._forget_rate = forget_rate
        self._neurons = [ Neuron(x, self._learning_rate, self._no_of_inputs, activation_function, self._forget_rate) for x in range(self._no_of_neurons) ]

    """Funkcja trenowania bez nauczyciela"""
    def train_without_supervisor(self, inputs):
        winner = self._neurons[0]
        for neuron in self._neurons:    #wyszukanie neuronu o najwyzszym wyjsciu dla danego zestawu
            temp_winner = neuron
            neuron.guess(inputs)
            if temp_winner._val > winner._val:
                winner = temp_winner    #znalezenie neuronu

        winner.trainWithoutSupervisor(inputs)   #aktualizacja wag neuronu
        return winner

    """Funkcja odgadywania"""
    def guess(self, inputs):
        winner = None       #wyszukanie neuronu o najwyzszym wyjsciu dla danego zestawu
        for neuron in self._neurons:
            temp_winner = neuron
            neuron.guess(inputs)s
            if winner == None:
                winner = neuron
            elif temp_winner._val > winner._val:
                winner = temp_winner

        return winner   #zwrócenie neuronu o najwyzszym wyjsciu
```

Plik Neuron.py

```
import random
from math import exp
from signsigm import *
import numpy as np

class Neuron:
    """Inicjalizacja neuronu i jego ustawień"""
    def __init__(self, iid, learning_rate, no_of_inputs, activation_function, forgetRate):
        self.__dict__['_no_of_inputs'] = no_of_inputs
        self.__dict__['_weights'] = []
        self.__dict__['_inputs'] = []
        self.__dict__['_learningRate'] = learning_rate
        self.__dict__['_activationFunction'] = activation_function
        self.__dict__['_bias'] = 1
        self.__dict__['_forgetRate'] = forgetRate
        self.__dict__['_sum'] = None
        self.__dict__['_error'] = None
        self.__dict__['_val'] = 0
        self.__dict__['_iid'] = iid
        self.__dict__['_forget'] = 1 - forgetRate
        self.__dict__['_in_row_winner'] = -1

        #losowanie wag
        for weight in range(0, self._no_of_inputs):
            self._weights.append(np.random.uniform(0, 1))
        if self._forgetRate == None:
            self._forgetRate = 0

    """Funkcja odgadywania"""
    def guess(self, inputs):
        self._inputs = inputs

        #wymnozenie i zsumowanie wag i wejsc
        self._sum = np.dot(self._weights, self._inputs) + self._bias
        self._val = self._activationFunction(self._sum)     #funkcja aktywacji neuronu f(sum)
        return self._val

    """Funkcja trenowania z nauczycielem"""
    def trainWithSupervisor(self, inputs, desiredOutput):  #∂wij(k+1) = (1-fr)*∂wij(k) + lr*yj*yi (yj to wejście nr j) yi to oczekiwane wyjscie
        output = self.guess(inputs)

        for i in range(len(self._inputs)):  #aktualizacja wag wedlug wzoru powyzej
            self._weights[i] = (1-self._forgetRate) * self._weights[i] + self._learningRate * self._inputs[i] * desiredOutput

    """Funkcja trenowania bez nauczyciela"""
    def trainWithoutSupervisor(self, inputs): #∂wij(k+1) = (1-fr)*∂wij(k) + lr*yj*yi (yj to wejście nr j) yi to wyjście neuronu
        output = self.guess(inputs)

        constant = self._learningRate * output
        for i in range(len(self._inputs)):  #aktualizacja wag edlug wzoru powyzej
            self._weights[i] *= self._forget
            self._weights[i] += constant * self._inputs[i]
```

```
        return output
```

Plik testinput.py

```
"""Klasa z emoji (wejsciami)"""
class TestInput():
    avaliableEmojis = ["sad","D","wrr","xD", "|"]   #lista dostepnych emoji w danych

    def __init__(self):
        self.inputsMap = {}
        self.makeInputs()

    """Wszystkie emoji w formie binarnej"""
    def makeInputs(self):
        for emoji in TestInput.avaliableEmojis:
            if emoji == "|":
                self.inputsMap["|"]=[
                    0,0,0,0,0,0,0,0,
                    0,1,1,0,0,1,1,0,
                    0,1,1,0,0,1,1,0,
                    0,0,0,0,0,0,0,0,
                    0,0,0,0,0,0,0,0,
                    0,1,1,1,1,1,1,0,
                    0,0,0,0,0,0,0,0,
                    0,0,0,0,0,0,0,0,
                    ]
            if emoji == "D":
                self.inputsMap["D"] = [
                    0,0,0,0,0,0,0,0,
                    0,1,1,0,0,1,1,0,
                    0,1,1,0,0,1,1,0,
                    0,0,0,0,0,0,0,0,
                    0,1,1,1,1,1,1,0,
                    0,1,1,1,1,1,1,0,
                    0,0,1,1,1,1,0,0,
                    0,0,0,0,0,0,0,0,
                    ]
            if emoji == "sad":
                self.inputsMap["sad"] = [
                    0,0,0,0,0,0,0,0,
                    0,1,1,0,0,1,1,0,
                    0,1,0,0,0,0,1,0,
                    0,0,0,0,0,0,0,0,
                    0,0,1,1,1,1,0,0,
                    0,1,0,0,0,0,1,0,
                    0,1,0,0,0,0,1,0,
                    0,0,0,0,0,0,0,0,
                    ]
            if emoji == "wrr":
                self.inputsMap["wrr"] = [
                    0,1,0,0,0,0,1,0,
                    0,1,1,0,0,1,1,0,
                    0,0,0,0,0,0,0,0,
                    0,0,0,0,0,0,0,0,
                    0,0,0,1,1,0,0,0,
                    0,0,1,0,0,1,0,0,
                    0,1,0,0,0,0,1,0,
                    0,0,0,0,0,0,0,0,
                    ]
            if emoji == "xD":
                self.inputsMap["xD"] = [
                    0,1,1,0,0,1,1,0,
                    0,0,0,1,1,0,0,0,
                    0,1,1,0,0,1,1,0,
                    0,0,0,0,0,0,0,0,
                    0,1,1,1,1,1,1,0,
                    0,1,0,0,0,0,1,0,
                    0,0,1,1,1,1,0,0,
                    0,0,0,0,0,0,0,0,
                    ]
    def getInputsMap(self): #zwrocenie danych jako mapy
        return self.inputsMap
```