KAMIL STACHOWICZ

Podstawy sztucznej inteligencji

Sieć Kohonena WTM

1. Cel ćwiczenia

Celem ćwiczenia jest poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły WTM do odwzorowywania istotnych cech liter alfabetu

2. Schemat ćwiczenia

1. Zaimplementowanie sieci Kohonena WTM
2. Uczenie sieci
3. Testowanie sieci

3. Wykonanie ćwiczenia

Do zaimplementowania sieci Kohonena WTM wykorzystano wiedzę zawartą w książce[1]. Sieć Kohonena działa na zasadzie współzawodnictwa, z wszystkich neuronów zostają wybrane te których wagi są najbardziej zbliżone do wag wejściowych. W algorytmie typu WTM neuron zwycięski oraz neurony znajdujące się w jego sąsiedztwie mają uaktualniane wagi.

$$w_i(k+1) \leftarrow w_i(k) + G(i,x)\eta(x - w_i)$$

Gdzie:
x - wejście do sieci
w_i - wektor wag i-tego neuronu
η - współczynnik uczenia - z przedziału (0,1)
k - numer iteracji

Obliczanie współczynnika G

$$G(i,x) = exp(\frac{-d^2(i,w)}{2\lambda^2})$$

Gdzie:
i - neuron i
w - neuron zwycięzca
d - odległość pomiędzy punktami
λ - aktualny promień sąsiedztwa

Sygnał wyjściowy oblicza się według wzoru:

$$y_j = \sqrt{\sum_j (x_j - w_{ij})^2}$$

Gdzie:
x_j - j-te wejście neuronu
y_i- wyjście i-tego neuronu
w_{ij} - j-ta waga i-tego neuronu

Uaktualnianie promienia sąsiedztwa:

$$\lambda(k) = \lambda_{max}(\frac{\lambda_{min}}{\lambda_{max}})^{\frac{k}{k_{max}}}$$

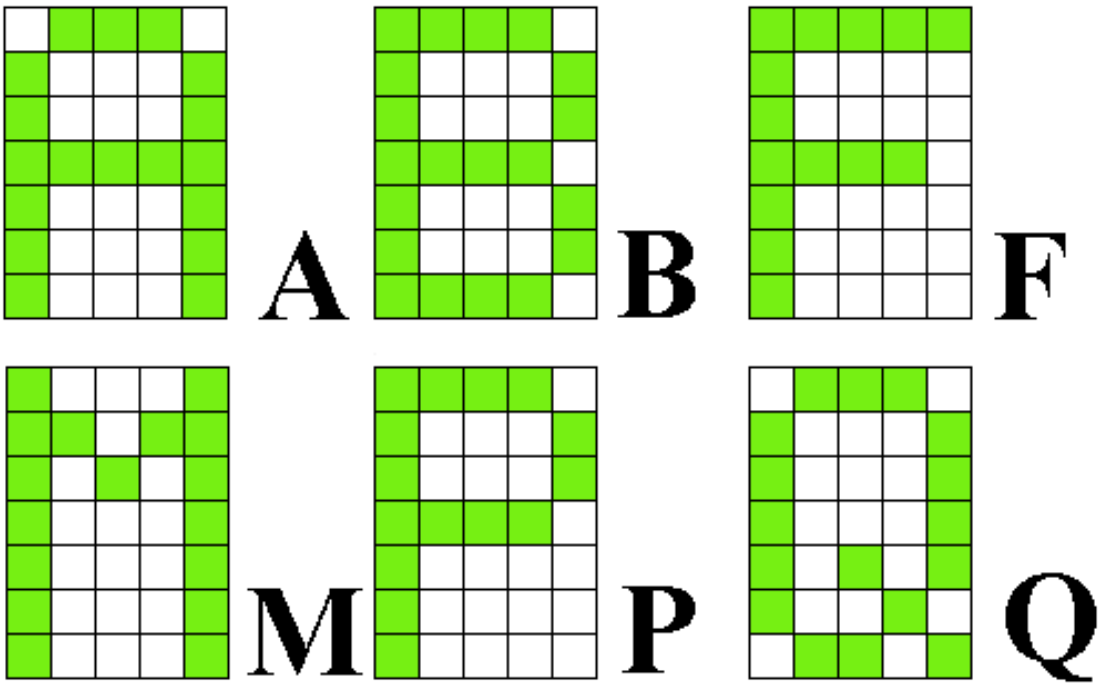
Gdzie:
λ - promień sąsiedztwa
(k - aktualny, min - minmalny, max - maxymalny)
k - numer aktualnej epoki
(max - ilość wszystkich epok)

---

[1] [1] Stanisław Osowski - Sieci neuronowe do przetwarzania danych

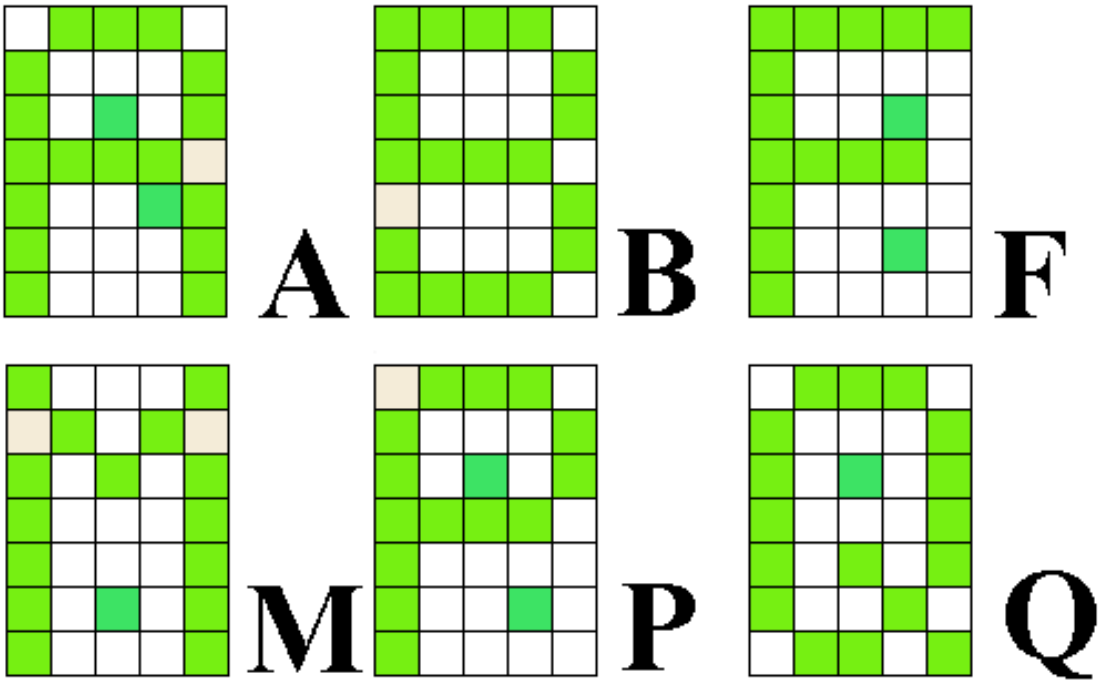
Zestawem testowym było 26 liter z alfabetu, dane zostały pobrane ze strony:
http://www.ai.c-labtech.net/sn/litery.html

Przykładowe litery:



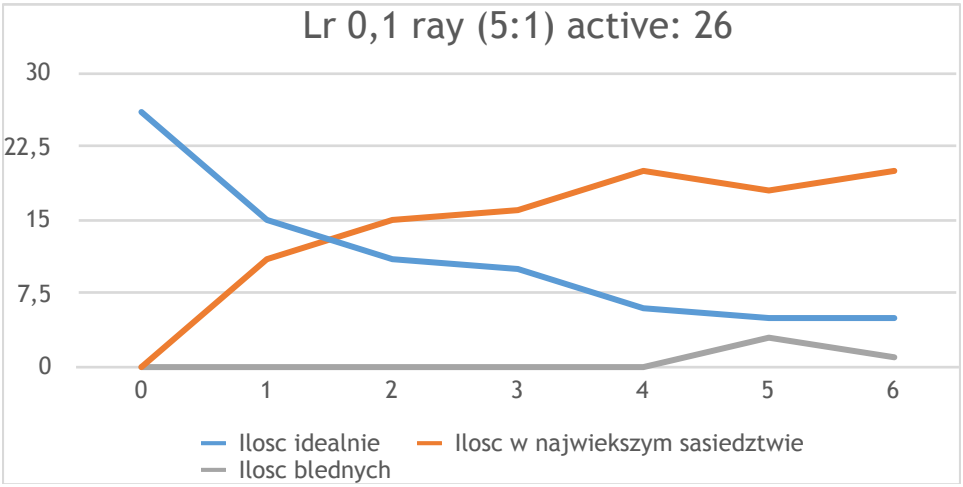
rysunek matryc 5 x 7 przykładowych liter A, B, F, M, Q

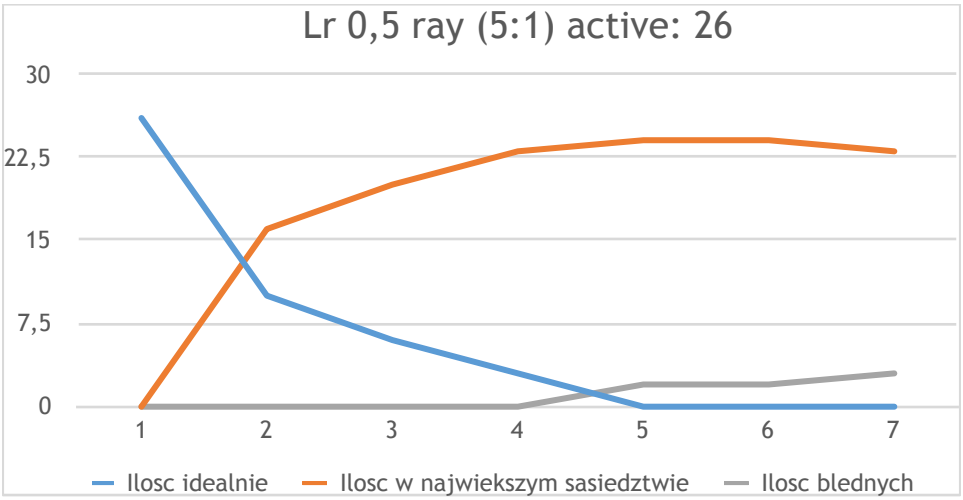
Przykładowe zaszumione litery:

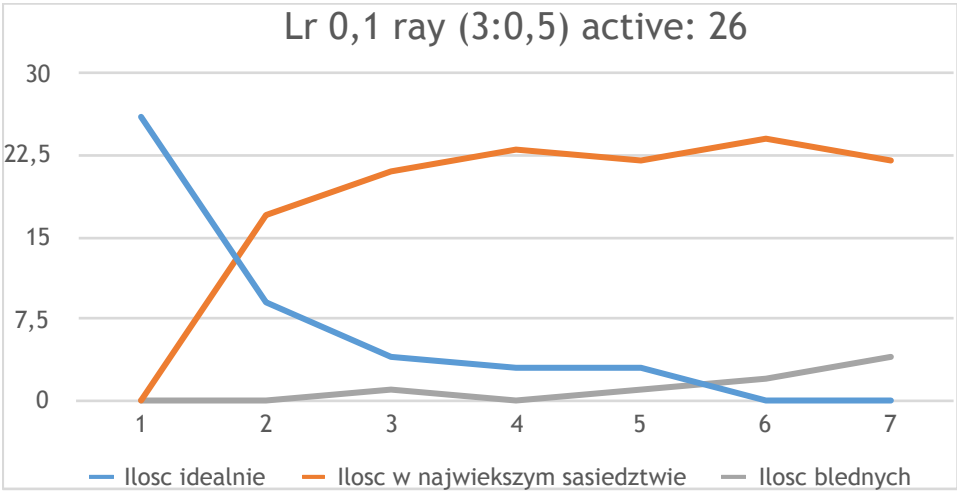


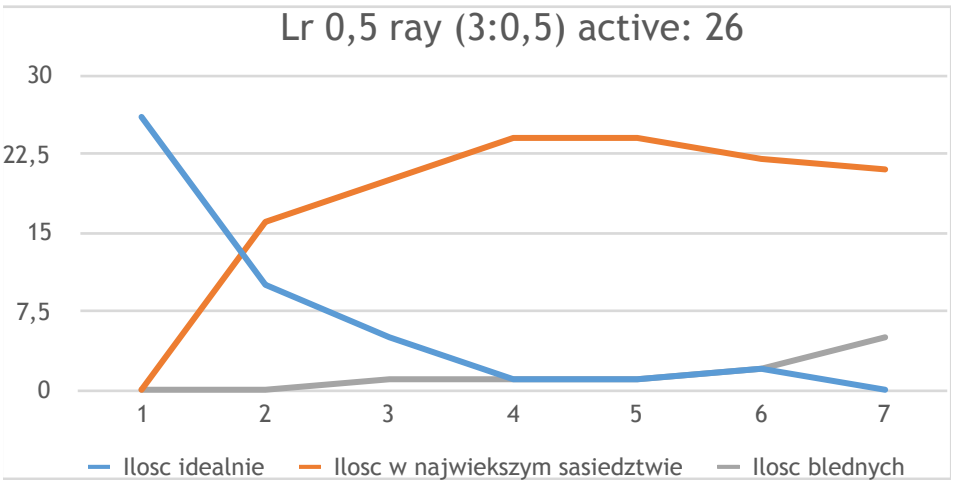
rysunek matryc 5 x 7 przykładowych liter A, B, F, M, Q

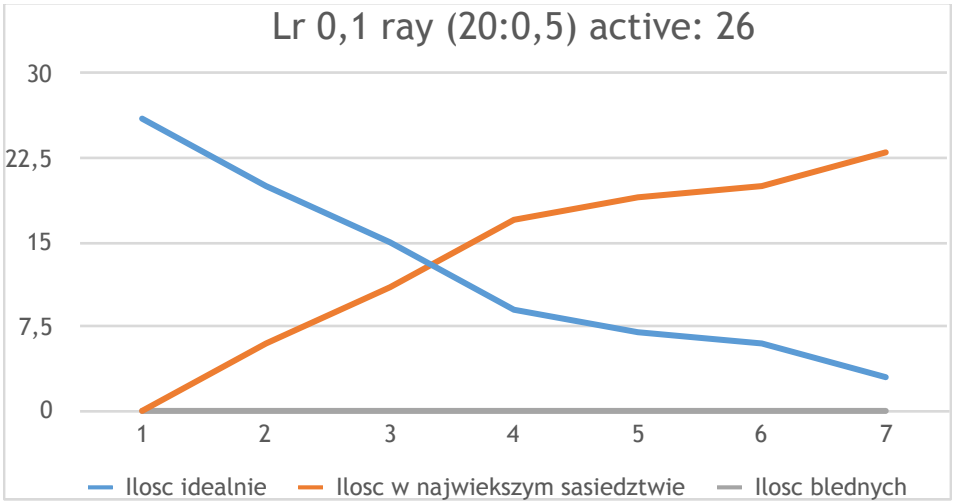
4. Wyniki

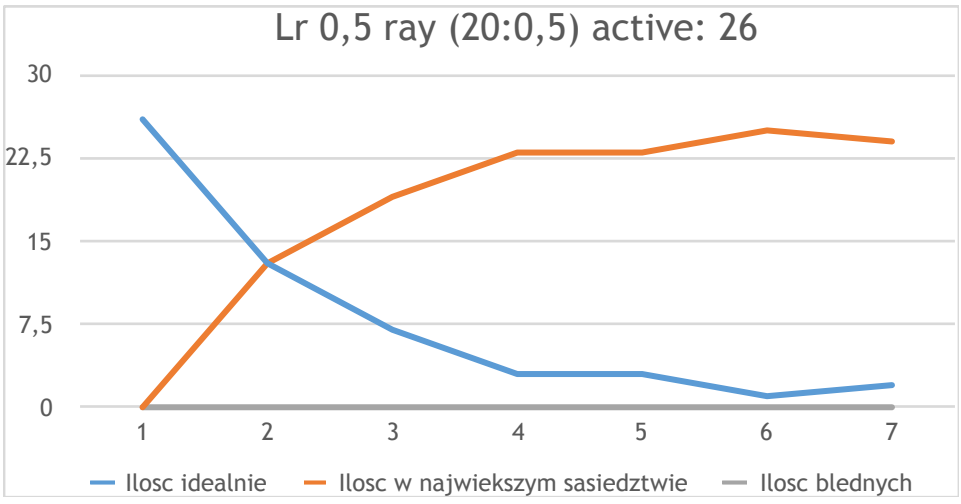

Lr 0,1 ray (5:1) active: 26
30
22,5
15
7,5
0
0
1
2
3
4
5
6
Ilosc idealnie
Ilosc w najwiekszym sasiedztwie
Ilosc blednych

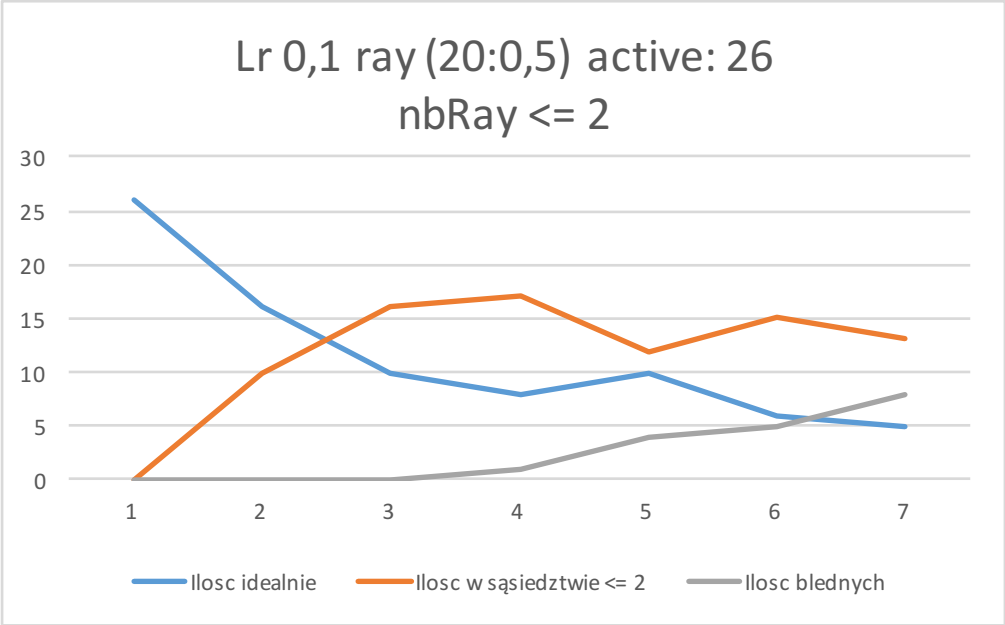

Lr 0,5 ray (5:1) active: 26
30
22,5
15
7,5
0
1
2
3
4
5
6
7
Ilosc idealnie
Ilosc w najwiekszym sasiedztwie
Ilosc blednych


Lr 0,1 ray (3:0,5) active: 26
30
22,5
15
7,5
0
1
2
3
4
5
6
7
Ilosc idealnie
Ilosc w najwiekszym sasiedztwie
Ilosc blednych


Lr 0,5 ray (3:0,5) active: 26
30
22,5
15
7,5
0
1
2
3
4
5
6
7
Ilosc idealnie
Ilosc w najwiekszym sasiedztwie
Ilosc blednych


Lr 0,1 ray (20:0,5) active: 26
30
22,5
15
7,5
0
1
2
3
4
5
6
7
Ilosc idealnie
Ilosc w najwiekszym sasiedztwie
Ilosc blednych


Lr 0,5 ray (20:0,5) active: 26
30
22,5
15
7,5
0
1
2
3
4
5
6
7
Ilosc idealnie
Ilosc w najwiekszym sasiedztwie
Ilosc blednych



Reszta wyników została umieszczona w repozytorium w pliku excel.

5. Omówienie wyników

Sieci o ilości aktywnych neuronów mniejszej niż 26 były uważane za niepoprawnie nauczone.

Zauważono że dla learning rate 0,01 ilość aktywnych neuronów, niezależnie od promienia sąsiedztwa nigdy nie wyniosła 26.

Wraz ze wzrostem zaszumienia danych testowych rośnie ilość liter odgadniętych nie przez neurony wybrane jako zwycięskie, ale przez neurony znajdujące się w jej okolicy.

Ze względu na bardzo duży maksymalny promień, sieć o promieniu 20 -> 0,5 została zbadana pod kątem sąsiedztwa nie większego niż 2.

Sieć ta pomimo dużej rozbieżności pomiędzy promieniem początkowym a końcowym ma największą ilość liter wybieranych przez neurony zwycięskie z uczenia.

6. Wnioski

Sieć Kohonena WTM do grupowania daje nam więcej możliwości zrozumienia grupowania danych wejściowych do programu. W przypadku uczenia literami bardzo łatwo zauważyć jest że wraz ze wzrostem stopnia zaszumienia rośnie ilość danych odgadniętych przez neurony sąsiedzkie, wynika to bezpośrednio z faktu że neurony te uczone są mniej niż neurony zwycięskie, co pozwala na większą rozbieżność w wagach, z czego wynika wyższy sygnał wyjściowy niż dla neuronów zwycięskich.

Podczas analizy danych został popełniony błąd, w trakcie określania testowanego sąsiedztwa, został wybrany promień maksymalny, co w przypadku sieci o maksymalnym promieniu 20 (gdy sieć ma 20x20 neuronów) pokrywa niemalże całą mapę. Błąd ten został skorygowany dla sieci 20 -> 0,5 co widać na ostatnim wykresie. Pokazuje on że początkowy duży wektor, zmniejszany stopniowo aż do 0,5 pozwala na nauczenie sieci w taki sposób najbardziej optymalny.

Łatwo zauważyć że sieć o lr 0,1 i promieniu od 20 do 0,5 ma największą skuteczność uczenia, występuje w nim stosunkowo najwięcej dokładnych dopasowań neuronu zwycięskiego do swojej litery, uwzględniając również zgadywanie w sąsiedztwie mniejszym od 2, radzi sobie lepiej nawet od sieci w których wzięty był największy promień sąsiedztwa równy 3 i 5.

Sieci o lr równym 0,01 nie zostały ujęte w tym sprawozdaniu, wyniki z obserwacji ich działania są zamieszczone w formie załącznika. Żadna z sieci o takich parametrach, nie była w stanie uzyskać 26 różnych neuronów. Wynikać to może z faktu ograniczenia liczby epok do stałej, równej 50.

7. Listing kodu

```
from Grid import *
from testinput import *
from collections import Counter
import csv

if __name__ == '__main__':
    """Wstępna konfiguracja sieci"""
    gridn = 20
    gridm = 20
    epoch = 50
    for neighbourhoodMaxRay, neighbourhoodMinRay in [(3, 0.5), (5, 1), (20, 0.5)]:
        for learningRate in [0.1, 0.5, 0.01]:
            with open('results.csv', 'a', newline="") as csvfile:
                resultWriter = csv.writer(csvfile, dialect='excel')
                gridNxM = (gridn, gridm)
                resultWriter.writerow(['Lr'] + ['Epoch'] + ['GridDim'] + ['maxRay'] + ['minRay'])
                resultWriter.writerow([learningRate] + [epoch] + [gridNxM] + [neighbourhoodMaxRay] + [neighbourhoodMinRay])

                """Generowanie danych testowych"""
                testInputInstance = TestInput()
                testInputs = testInputInstance._testArguments
                noOfInputs = len(testInputs['A'])

                """Utworzenie siatki neuronów"""
                kohonenGrid = Grid(noOfInputs, learningRate, gridn, gridm, neighbourhoodMaxRay, neighbourhoodMinRay, epoch)

                """Trenowanie sieci"""
                winners = {}
                activeNeurons  = []
                for i in range(epoch):
                    for key, value in testInputs.items():
                        winners[key] = kohonenGrid.train(value, i)

                """Sprawdzenie liczby działających w sieci neuronów"""
                uniqueNeurons = len(Counter(winners.values()).keys())
                resultWriter.writerow(['Ilosc aktywnych'] + [uniqueNeurons])
                print("Ilosc aktywnych", str(uniqueNeurons))

                """Zaszumienie danych testowych"""
                noisedInput = []
                for i in range(7):
                    noisedInput.append(testInputInstance.getNoisedLetters(i))

                winnersTest = {}
                matched = Counter()
                almostMatched = Counter()
                badMatched = Counter()
                noisedCounter = 0

                resultWriter.writerow(['Ilosc zaszumionych'] + ['Ilosc idealnie'] + ['Ilosc w najwiekszym sasiedztwie'] + ['Ilosc blednych'])

                """Testowanie wyuczonej sieci"""
                for inputs in noisedInput:
                    for key, value in inputs.items():
                        tempWinner = kohonenGrid.guess(value)
                        if winners[key] == tempWinner:
                            matched[noisedCounter] += 1
                        elif winners[key].getDistanceToOther(tempWinner) <= 2:
                            almostMatched[noisedCounter] += 1
                        else:
                            badMatched[noisedCounter] += 1
                            print("Winner in training: ", winners[key].getNeuronString(), " winner in noised: ", tempWinner.getNeuronString(), " no of noised: ",
noisedCounter)
                    resultWriter.writerow([noisedCounter] + [matched[noisedCounter]] + [almostMatched[noisedCounter]] + [badMatched[noisedCounter]])
                    noisedCounter += 1



                print("Exacly matched")
                print(matched)
                print("Almost matched")
                print(almostMatched)
```

Plik neuronu kohonena

```
import random
```

```
import numpy as np
import math
"""Klasa neuronu według Kohonena"""
class NeuronKohonen:
    """Funkcja inicjalizująca - ustawianie parametrów neuronu"""
    def __init__(self, learning_rate, no_of_inputs, x, y):
        self.__dict__['_no_of_inputs'] = no_of_inputs
        self.__dict__['_weights'] = []
        self.__dict__['_inputs'] = []
        self.__dict__['_learningRate'] = learning_rate
        self.__dict__['_sum'] = None
        self.__dict__['_weightsStart'] = []
        self.__dict__['err'] = None
        self.__dict__['_x'] = x
        self.__dict__['_y'] = y


        for i in range(self._no_of_inputs):     #losowanie wag i zapisanie ich kopii
            weight = random.uniform(0,1)
            self._weights.append(weight)
            self._weightsStart.append(weight)

    """Funkcja procesująca"""
    def guess(self, inputs):
        self._inputs = inputs
        self._sum = 0

        for i in range(len(self._weights)):     #wyjsciem neuronu jest odległość między wagami a danymi wejściowymi
            self._sum += (self._inputs[i] - self._weights[i])**2
        self._sum = math.sqrt(self._sum)

        return self._sum
    """Funkcja trenująca - uaktualniająca wagi"""
    def train(self):
        errTemp = 0
        for i in range(len(self._inputs)):  #aktualizacja wag polega na przyblianiu ich do wektora wejściowego
            self._weights[i] += self._learningRate * (self._inputs[i] - self._weights[i])

    def trainGauss(self, error):
        tmp = error * self._learningRate
        for i in range(len(self._inputs)):  #aktualizacja wag polega na przyblianiu ich do wektora wejściowego
            self._weights[i] += tmp * (self._inputs[i] - self._weights[i])

    """Funkcja zwracająca wagi jako string"""
    def getWeightsAsString(self):
        return str(self._weights)

    """Funkcja resetująca wagi neuronów do początkowych"""
    def resetNeuron(self):
        self._weights = []
        for weight in self._weightsStart:
            self._weights.append(weight)

    """Funkcja odleglosci pomiedzy neuronami"""
    def getDistanceToOther(self, other):
        return math.sqrt((self._x-other.getX())**2 + (self._y - other.getY())**2)

    def getX(self):
        return self._x

    def getY(self):
        return self._y

    def getNeuronString(self):
        return "x: " + str(self._x) + " y: " + str(self._y)
```

Plik sieci Kohonena

```
from NeuronKohonen import *
import sys
import math

sys.path.append('../functions')
from ActivationFunctions import *


"""Klasa sieci Kohonena"""
class Grid:
    """Klasa tworząca neurony i ustawiająca parametry sieci"""
    def __init__(self, noOfInputs, learningRate, height, width, nbMaxRay, nbMinRay, epochs):
        self.__dict__['_noOfInputs'] = noOfInputs
        self.__dict__['_learningRate'] = learningRate
        self.__dict__['_height'] = height
        self.__dict__['_width'] = width
        self.__dict__['_neurons'] = [[NeuronKohonen(self._learningRate, self._noOfInputs, x, y) for x in range(self._width)] for y in range(self._height) ]
        self.__dict__['_neighbourhoodMaxRay'] = nbMaxRay
        self.__dict__['_neighbourhoodMinRay'] = nbMinRay
        self.__dict__['_epochs'] = epochs
        self.__dict__['_gaussDivisor'] = 2 * nbMaxRay**2
        self.__dict__['_currentRay'] = nbMaxRay
```

```
    """Funkcja trenowania sieci"""
    def train(self, inputs, currEpoch):
        winner = self.guess(inputs)

        for i in range(self._height):
            for j in range(self._width):
                tempNeuron = self._neurons[i][j]
                g = self.gaussNeighbourhood(winner=winner, neighbour=tempNeuron)
                tempNeuron.trainGauss(g)    #za każdym razem uczone są wszystkie neurony
                                            #uwzględniając odpowiedni współczynnik G

        self.calculateRay(currEpoch)
        return winner

    """Funkcja wyłaniająca najsilniejszy neuron"""
    def guess(self, inputs):
        lowestOutput = 10
        winnerCoords = (0,0)
        for i in range(self._height):       #wybierany zostaje neuron o najmniejszej odleglosci wektorow: wejscia od wag
            for j in range(self._width):
                tmp = self._neurons[i][j].guess(inputs)
                if tmp < lowestOutput:
                    lowestOutput = tmp
                    winnerCoords = (i, j)

        return self._neurons[winnerCoords[0]][winnerCoords[1]]

    """Funkcja przywracająca początkowe wagi neuronów"""
    def resetNeurons(self):
        for i in range(self._height):
            for j in range(self._width):
                self._neurons[i][j].resetNeuron()

    """Funkcja sąsiedztwa Gaussowskiego"""
    def gaussNeighbourhood(self, winner, neighbour):
        dx = winner.getX() - neighbour.getX()
        dy = winner.getY() - neighbour.getY()

        return math.exp(-(dx**2 + dy**2)/self._gaussDivisor)

    """Funkcja zmnieniająca promień sąsiedztwa"""
    def calculateRay(self, epoch):
        self._currentRay = self._neighbourhoodMaxRay * (self._neighbourhoodMinRay/self._neighbourhoodMaxRay)**(epoch/self._epochs)
        self._gaussDivisor = 2 * self._currentRay ** 2
```

Dane do uczenia/testowania

```
#!/usr/bin/python
import numpy as np
import copy

class TestInput():
    """docstring forTestInput."""
    """
        Test input is a vector which represents capital and small letters like as in
        5x7 table
    """
    def __init__(self):
        self.__dict__['_testArguments'] = {}
        self.__dict__['_lenOfTest'] = 35

        self.getLetter()

    """Stworzenie danych wejściowych"""
    def getLetter(self):
        self._testArguments['A'] = [0,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1]
        self._testArguments['B'] = [1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0]
        self._testArguments['C'] = [0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,1,1,1,0]
        self._testArguments['D'] = [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]
        self._testArguments['E'] = [1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1,1,0,1,0,0,0,0,0,1,0,0,0,0,1,1,1,1,1]
        self._testArguments['F'] = [1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1,1,0,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0]
        self._testArguments['G'] = [0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,0,1,0,1,1,1,1,1,0,0,0,1,1,0,0,0,0,1,0,1,1,1,0]
        self._testArguments['H'] = [1,0,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1]
        self._testArguments['I'] = [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]
        self._testArguments['J'] = [1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0,0,1,0,1,1,1,0]
        self._testArguments['K'] = [1,0,0,0,1,1,0,0,1,0,1,0,1,0,0,1,1,0,0,0,1,0,1,0,0,1,0,0,1,0,1,0,1,0,0,0,1]
        self._testArguments['L'] = [1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,1,1,1,1]
        self._testArguments['M'] = [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]
        self._testArguments['N'] = [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]
        self._testArguments['O'] = [0,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,1,1,1,0]
        self._testArguments['P'] = [1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,0,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0]
        self._testArguments['Q'] = [0,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,1,0,1,1,0,0,1,1,0,1,1,1,1,1]
        self._testArguments['R'] = [1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,0,1,0,1,0,0,1,0,0,1,0,1,0,0,0,1]
        self._testArguments['S'] = [0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,0,1,1,1,1,0,0,0,0,0,1,1,0,0,0,1,0,1,1,1,0]
        self._testArguments['T'] = [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]
        self._testArguments['U'] = [1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,1,1,1,0]
        self._testArguments['V'] = [1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,0,1,0,1,0,0,0,1,0,0]
        self._testArguments['W'] = [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]
        self._testArguments['X'] = [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]
        self._testArguments['Y'] = [1,0,0,0,1,1,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0]
        self._testArguments['Z'] = [1,1,1,1,1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1]
```

```
"""Zaszumienie danych wejściowych"""
def getNoisedLetters(self, noOfNoised):
    testArgumentsCopy = copy.deepcopy(self._testArguments)

    for key, value in testArgumentsCopy.items():
        pixels = np.random.choice(self._lenOfTest, noOfNoised, replace=False)
        for pixel in pixels:
            if value[pixel] == 1:
                value[pixel] = 0
            else:
                value[pixel] = 1

    return testArgumentsCopy
```