Podstawy sztucznej inteligencji

Budowa i działanie sieci Kohonena dla WTA

Sprawozdanie ze scenariusza nr 5

1. Cel ćwiczenia

Celem omawianego ćwiczenia było poznanie budowy oraz sposobu działania sieci Kohonena dla WTA.

2. Schemat ćwiczenia

1. Zaimplementowanie sieci neuronowej Kohonena dla WTA[1]
2. Uczenie sieci zestawem testowym[2]
3. Zestawienie wyników

3. Wykonanie ćwiczenia

Sieć została zaimplementowana z pomocą informacji zawartych w książce (nazwa zamieszczona w przypisie).
Sieć Kohonena działa na zasadzie współzawodnictwa, z wszystkich neuronów zostają wybrane te których wagi są najbardziej zbliżone do wag wejściowych. W algorytmie typu WTA dla jednego typu danych do sklasyfikowania wybierany jest jeden neuron który po zidentyfikowaniu jako zwycięzca ma uaktualnione wagi.

W algorytmie Kohonen WTA wagi uaktualniane są według następującego schematu:

$$w_{ij}(k+1) \leftarrow w_{ij}(k) + \eta(x_j - w_{ij})$$

Gdzie:
x_j - j-te wejście neuronu
w_{ij} - j-ta waga i-tego neuronu
η - współczynnik uczenia - z przedziału (0,1)
k - numer iteracji

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

Dane wejściowe (średnie wartości):

Gatunek	Długość działki kielicha	Szerokość działki kielicha	Długość płatka	Szerokość płatka
Iris setosa	5,01	3,42	1,47	0,25
Iris versicolor	5,94	2,78	4,26	1,33
Iris-virginica	6,59	2,98	5,56	2,03

[1] Stanisław Osowski - *Sieci neuronowe do przetwarzania danych* str. 289

[2] https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

4. Wyniki

Podczas uczenia sieci, dane wejściowe zostały znormalizowane do wartości (0,1)

Gatunek	Długość działki kielicha	Szerokość działki kielicha	Długość płatka	Szerokość płatka
Iris setosa	0,81	0,55	0,24	0,04
Iris versicolor	0,75	0,35	0,54	0,17
Iris-virginica	0,71	0,32	0,6	0,22

Lr 0,01

Gatunek	Długość działki kielicha	Szerokość działki kielicha	Długość płatka	Szerokość płatka	Odgadnietych testowych
Iris setosa	0,81	0,55	0,24	0,04	Zestaw testowy
Iris versicolor	0,75	0,35	0,54	0,17	18
Iris virginica	0,71	0,32	0,6	0,22	
Lr: 0,01	Epoch: 10	Grid: 20x20			13
Iris setosa	0,78144079	0,58835309	0,22063701	0,06407736	
Iris versicolor	0,74962809	0,34745333	0,53581163	0,16546400	
Iris virginica	0,70226140	0,32118289	0,59248844	0,22052745	
Lr: 0,01	Epoch: 50	Grid: 20x20			17
Iris setosa	0,79995774	0,54894505	0,23466900	0,03953890	
Iris versicolor	0,75367114	0,34907882	0,53036055	0,16394479	
Iris virginica	0,70514856	0,32386216	0,59074993	0,21604293	
Lr: 0,01	Epoch: 100	Grid: 20x20			18
Iris setosa	0,80159725	0,54600162	0,23506891	0,04004370	
Iris versicolor	0,75367114	0,34907882	0,53036055	0,16394479	
Iris virginica	0,70504565	0,32255068	0,59220665	0,21407754	
Lr: 0,01	Epoch: 100	Grid: 20x20			17
Iris setosa	0,80159725	0,54600162	0,23506891	0,04004370	
Iris versicolor	0,72669455	0,33569182	0,56391624	0,19197855	
Iris virginica	0,72669455	0,33569182	0,56391624	0,19197855	

Lr 0,1

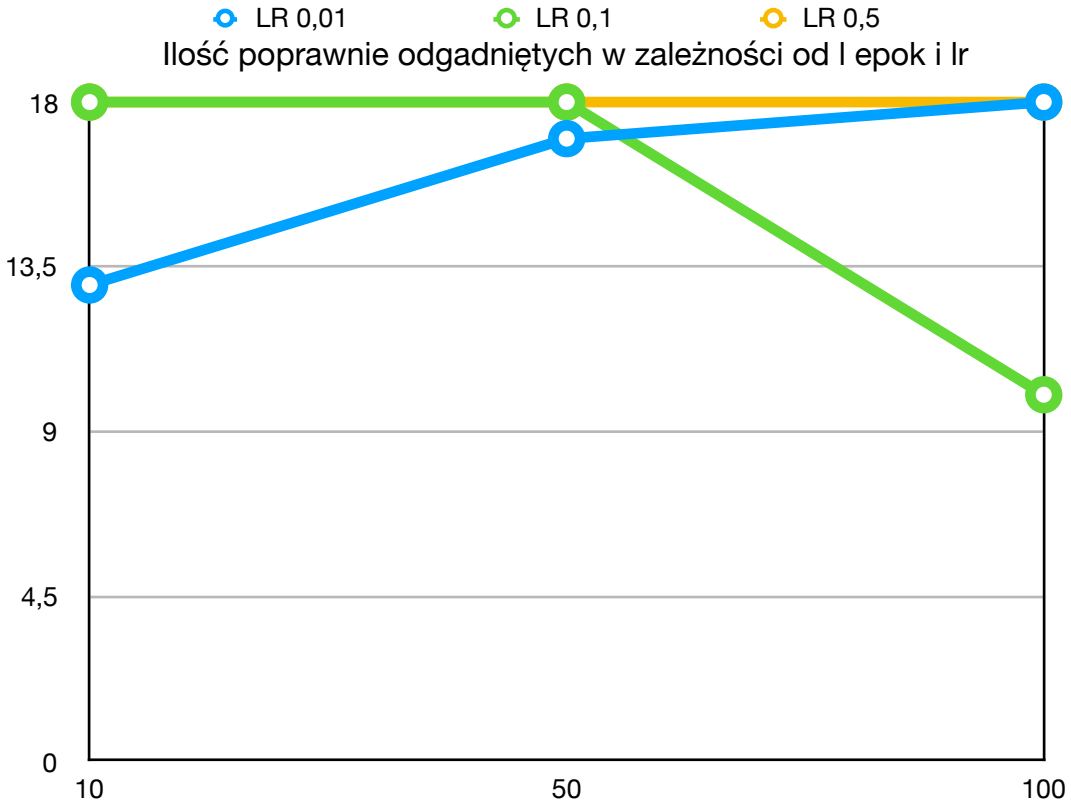
Gatunek	Długość działki kielicha	Szerokość działki kielicha	Długość płatka	Szerokość płatka	Odgadnietych testowych
Iris setosa	0,81	0,55	0,24	0,04	Zestaw testowy
Iris versicolor	0,75	0,35	0,54	0,17	18
Iris virginica	0,71	0,32	0,6	0,22	
Lr 0,1	Epoch: 10	Grid 20x20			18
Iris setosa	0,79826757	0,54958664	0,23797498	0,04299638	
Iris versicolor	0,75389660	0,35492546	0,52558237	0,16556584	
Iris virginica	0,70546743	0,32837078	0,58704887	0,21840310	
Lr 0,1	Epoch: 50	Grid 20x20			18
Iris setosa	0,80237989	0,54194774	0,23933251	0,04463979	
Iris versicolor	0,75389660	0,35492546	0,52558237	0,16556584	
Iris virginica	0,70546743	0,32837078	0,58704887	0,21840310	
Lr 0,1	Epoch: 100	Grid 20x20			10
Iris setosa	0,78534734	0,57290855	0,22558482	0,04695660	
Iris versicolor	0,72387748	0,36364673	0,55592579	0,18277776	
Iris virginica	0,69670802	0,32990396	0,59057596	0,23632691	

Lr 0,5

Gatunek	Długość działki kielicha	Szerokość działki kielicha	Długość płatka	Szerokość płatka	Odgadnietych testowych
Iris setosa	0,81	0,55	0,24	0,04	Zestaw testowy
Iris versicolor	0,75	0,35	0,54	0,17	18
Iris virginica	0,71	0,32	0,6	0,22	
Lr 0,5	Epoch: 10	Grid 15x15			18
Iris setosa	0,78306077	0,55703914	0,26439658	0,06425585	
Iris versicolor	0,75371570	0,36763294	0,51675989	0,16756498	
Iris virginica	0,69388384	0,33249641	0,58926046	0,24523021	
Lr 0,5	Epoch:50	Grid 15x15			18
Iris setosa	0,78306077	0,55703914	0,26439658	0,06425585	
Iris versicolor	0,75371570	0,36763294	0,51675989	0,16756498	
Iris virginica	0,69388384	0,33249641	0,58926046	0,24523021	
Lr 0,5	Epoch:100	Grid 15x15			18
Iris setosa	0,78306077	0,55703914	0,26439658	0,06425585	
Iris versicolor	0,75371570	0,36763294	0,51675989	0,16756498	
Iris virginica	0,69388384	0,33249641	0,58926046	0,24523021	

Tabela 1-2

Gatunek	Długość działki kielicha	Szerokość działki kielicha	Długość płatka	Szerokość płatka	Odgadnietych testowych
Iris setosa	0,81	0,55	0,24	0,04	Zestaw testowy
Iris versicolor	0,75	0,35	0,54	0,17	18
Iris virginica	0,71	0,32	0,6	0,22	
Lr 0,1	Epoch: 10	Grid 5x5			18
Iris setosa	0,80237988775825	0,5419477416324254	0,2393325122377685	0,0446397926522230365	
Iris versicolor	0,7241234115250796	0,3435096933611997	0,5591096180733784	0,2011136246480528 2	
Iris virginica	0,7241234115250796	0,3435096933611997	0,5591096180733784	0,2011136246480528 2	
Lr 0,01	Epoch: 10	Grid 5x5			18
Iris setosa	0,7989460286396434	0,5476513075262899	0,2347902943296499	0,0406458587138 3504	
Iris versicolor	0,7264899468053399	0,3347368084721 8507	0,5646271761230767	0,1921680147111 7734	
Iris virginica	0,7264899468053399	0,3347368084721 8507	0,5646271761230767	0,1921680147111 7734	
Lr 0,5	Epoch: 10	Grid 5x5			18
Iris setosa	0,7830607680386272	0,5570391400236537	0,2643965826735419	0,0642558502 5821608	
Iris versicolor	0,7105592175761939	0,3465669099523 9114	0,5663817905928215	0,2237059705 3028432	
Iris virginica	0,7105592175761939	0,3465669099523 9114	0,5663817905928215	0,2237059705 3028432	



5. Analiza wyników - obserwacje

Dla learning rate równego 0,01 sieć uczy się liniowo i jej dokładność rośnie.
Te same neurony dla różnych gatunków wybierane są sporadycznie przy przetrenowaniu sieci za dużą ilością epok.
Wektor wag zbliżony jest do wektora danych wejściowych.

Sieć o learning rate 0,1 zostaje przetrenowana przy 100 epokach. Do 50 działa bardzo sprawnie, a wektory wag są wystarczająco zbliżone do wektorów danych wejściowych aby poprawnie pogrupować zestaw testowy.

Sieć o learning rate 0,5 już po 10 epokach jest nauczona poprawnie rozpoznawać zestaw testowy. Każde zwiększenie liczby epok nie skutkuje dalszymi zmianami wag.
Sieć dokładnie grupuje dane testowe do zbiorów w których się zawierają.

Wszystkie próby dotychczas były przeprowadzane na siatce 20x20 neuronów, daje to sumę 400 różnych neuronów o bardzo zróżnicowanych wagach.

Spróbowano stworzyć mniejszą siatkę o rozmiarach 5x5 co daje 25 różnych neuronów, siatka ta jest przetrenowana po 10 epokach niezależnie od wybranego współczynnika uczenia. Zostaje wybrany jeden neuron dla dwóch różnych gatunków kwiatów. Kolejne próby zwiększania liczby epok nie zmieniają rezultatów.

6. Wnioski

W sieci Kohonena WTA wartości wag dążą do bycia jak najbardziej zbliżonymi do wartości wektora danych wejściowych. Podczas ćwiczenia zauważono że dane wejściowe dla gatunku „Iris versicolor” oraz „Iris virginica” są do siebie bardzo zbliżone, co zaostrza się w wypadku normalizacji danych wejściowych (dane zostały przekształcone tak aby cechy zawierały się w przedziale (0,1)).

W trakcie wykonywania ćwiczenia zauważono że najlepszą konfiguracją jest ta o współczynniku uczenia 0,5 oraz o siatce 20x20. Sieć bardzo szybko była w stanie 100% rozpoznać poprawnie gatunki z danych testujących (tych które nie brały udziału w uczeniu), pozwala to na zaoszczędzenie mocy obliczeniowej maszyny na którym dana siatka pracuje.

Sieć o współczynniku uczenia 0,01 dawała wyniki jeszcze dokładniej zbliżone do średniej z danych wejściowych, pozwala to na stwierdzenie że mimo tego że potrzebuje 10x krotnie więcej czasu na naukę, pozwoli poźniej na dokładniejsze dopasowania.

Sprawdzono również jak radzą sobie sieci o niższej liczbie neuronów - 25 - sieć ta wybierała ten sam neuron dla dwóch różnych gatunków sieci, co czyni jej grupowanie niedokładne w zestawieniu do wyników sieci 400 neuronów.
Wynika to bezpośrednio ze zróżnicowania wag wejściowych, w sieci 400 neuronów bardziej prawdopodobne jest znalezienie dwóch różnych neuronów których wagi będą zbliżone do cech danych gatunków, zwłaszcza że różnica pomiędzy dwoma gatunkami jest stosunkowo niewielka. Pozwala to stwierdzić że już w pierwszej epoce zostaje wybierany ten sam neuron i jego wagi zostają uaktualnione dla dwóch różnych gatunków.

Sieć Kohonena typu WTA w odpowiedniej konfiguracji radzi sobie dość dobrze z zadanym problemem. Wymaga ona jednak nadzoru i dość dokładnej analizy wyników jej działania.

W sieci tej pozostaje również problem martwych neuronów, jako że z sieci 400 wybranych zostaje 3 jest to zaledwie 0,75% wszystkich neuronów. O ile uczenie odbywa się jednorazowo i można poświęcić większą ilość czasu i pamięci, o tyle w trakcie używania sieci do grupowania kwiatów, przechowywanie tak dużej ilości neuronów w pamięci mija się z celem.

7. Listing kodu

Main

```
from Grid import *
from Inputs import *
from NeuronKohonen import *

if __name__ == '__main__':
    #ustawienia i parametry sieci neuronowej
    learningRate = 0.1
    epoch = 100
    noOfInputs = 4
    width = 20
    height = 20

    inputs = Inputs()   #utworzenie danych wejściowych

    """Przypisanie danych róznych kwiatow to zmiennych"""
    species = {}
    for i in range(3):
        species[inputs.getInputData(i)[0]] = inputs.getInputData(i)[1]

    #zainicjalizowanie siatki neuronów Kohonena WTA
    grid = Grid(noOfInputs, learningRate, width, height)

    #tablica zwycięzców Kohonena
    winner = {}

    #wśród wszystkich neuronów w siatce odnajdywany jest zwycięzca
    for i in range(epoch):
        for j in range(len(species["Iris-setosa"])):
            for key in species.keys():
                winner[key] = grid.train(species[key][j])

    for key, value in winner.items():
        print(key)
        for val in value.getWeightsAsString().replace('[','').replace(']','').split(', '):
            print(val)

    testData = {}
    for i in range(3):
        testData[inputs.getTestData(i)[0]] = inputs.getTestData(i)[1]

    #sprawdzanie jak wiele z danych testowych zostanie poprawnie odgadnięte
    matched = 0
    winnerTest = {}
    for key, value in testData.items():
        for j in range(len(testData["Iris-setosa"])):
            winnerTest[key] = grid.guess(value[j])
            if winnerTest[key] == winner[key]:
```

```
            matched += 1

    print(matched)
    """średnie z danych wejściowych"""
    #averages:
    #Iris-setosa [5.01, 3.42, 1.47, 0.25]
    #Iris-versicolor [5.94, 2.78, 4.26, 1.33]
    #Iris-virginica [6.59, 2.98, 5.56, 2.03]

    """średnie z danych znormalizowanych"""
    #averages:
    #[0.81, 0.55, 0.24, 0.04]
    #[0.75, 0.35, 0.54, 0.17]
    #[0.71, 0.32, 0.6, 0.22]
```

Neuron

```
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):
        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

        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])

    """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)
```

Grid - siatka

```
from NeuronKohonen import *

"""Klasa sieci Kohonena"""
class Grid:
    """Klasa tworząca neurony i ustawiająca parametry sieci"""
    def __init__(self, noOfInputs, learningRate, height, width):
        self.__dict__['_noOfInputs'] = noOfInputs
        self.__dict__['_learningRate'] = learningRate
        self.__dict__['_height'] = height
        self.__dict__['_width'] = width
        self.__dict__['_neurons'] = [[NeuronKohonen(self._learningRate, self._noOfInputs) for x in range(self._width)] for y in range(self._height) ]

    """Funkcja trenowania sieci"""
    def train(self, inputs):
        winner = self.guess(inputs)
        winner.train() #wagi zwycięzcy zostają uaktualnione

        return winner

    """Funkcja wyłaniająca najsilniejszy neuron"""
    def guess(self, inputs):
        lowestOutput = 10
        winner = (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
                winner = (i, j)

    return self._neurons[winner[0]][winner[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()
```