Podstawy Sztucznej Inteligencji

Mikołaj Stepniewski

Uczenie sieci regułą Kohonena (WTA)

Cel ćwiczenia

Budowa i działanie sieci Kohonena dla WTA.

Opis sieci

Algorytm Kohonena jest jedną z najstarszych metod uczenia sieci samoorganizujących typu WTA i WTM i w chwili obecnej istnieje wiele jego odmian.
W tym przypadku rozważamy grupę neuronów, z których każdy ma przyporządkowane swoje miejsce w przestrzeni i gdzie uaktualniamy tylko wagi zwycięzcy. W odróżnieniu od grupy neuronów Hebba w przypadku Kohonena wyłaniamy zwycięzce na podstawie najmniejszej odległości wektora wag neuronu i wektora wejściowego (do którego neuron ma być przyporządkowany), jest to tzw. miara euklidesowa:

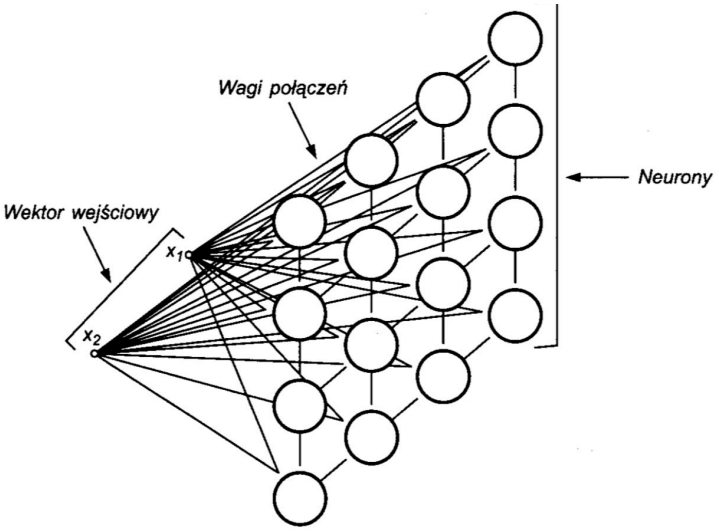
$$d(x, w_i) = \| x - w_i \| = \sqrt{\sum_{j=1}^{N} (x_j - w_{ij})^2}$$

Neuron zwycięzca podlega adaptacji, zmieniając swój wektor wag w kierunku wektora x. Jego wagi uaktualniane są na podstawie poniższej reguły, gdzie η - współczynnik uczenia:

$$w_{k+1} = w_k + \eta(x - w_k)$$

Wykazano, że proces samoorganizacji prowadzi zawsze do spójnego podziału przestrzeni danych, gdy chociaż jeden z wektorów x lub w podlega normalizacji. Przy znormalizowanych wektorach uczących x, wektory wag - nadążając za nimi - stają się automatycznie znormalizowane.
W metodzie samoorganizacji neuronów Kohonena bardzo ważna jest normalizacja danych wejściowych, która obywa się zgodnie z formułą:

$$x_i = \frac{x_i}{\sqrt{\sum_{i=1}^{N} x_i^2}}$$



Rys. 10.1. Struktura sieci samoorganizującej Kohonena

Algorytm WTA (Winner Takes All)

Neurony typu WTA mają stopień wejściowy w postaci standardowego sumatora sumującego iloczyny sygnałów wejściowych z ich odpowiednimi wagami.

$$u_i = \sum_{j=0}^{N} w_{ij} x_j$$

Grupa neuronów współzawodniczących ze sobą otrzymuje te same sygnały wejściowe x_j. W zależności od aktualnych wag, sygnały wyjściowe neuronów różnią się między sobą. Zwycięzcą zostaje neuron, którego sygnał wyjściowy jest największy. Taki neuron ma prawo do aktualizacji swoich wag. Dzięki temu za każdą wygraną epoką wagi neuronu zbliżają się do danego wektora wejściowego. Tym samym wektor przyporządkowuje go do siebie. Neurony, które nie zwyciężyły w danej epoce zostają zablokowane i ich wagi nie ulegają zmianie.

W efekcie takiego współzawodnictwa następuje samoorganizacja procesu uczenia. Neurony dopasowują swoje wagi w taki sposób, że przy prezentacji grup wektorów wejściowych zbliżonych do siebie zwycięża zawsze ten sam neuron. W trybie pracy odtworzeniowej odpowiedni neuron przez zwycięstwo we współzawodnictwie rozpoznaje swoją kategorię. Układy tego typu są stosowane najczęściej do klasyfikacji wektorów.

Dane wejściowe

Wektory wejściowe, czyli parametry irysów pobierane są w programie ze strony:
http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
Zawierają 150 wektorów (po 50 na rodzaj) i wyglądają w następujący sposób:

```
5.1,3.5,1.4,0.2,Iris-setosa
...
5.0,2.0,3.5,1.0,Iris-versicolor
...
6.5,3.2,5.1,2.0,Iris-virginica
...
```

Parametry widoczne powyżej musza zostać znormalizowane. Dokonałem tego za pomocą implementacji odległości euklidesowej:

```
def averageParameters(species, n=50):
    sum = [0.0 for _ in range(4)]
    for row in species:
        sum[0] += row[0]
        sum[1] += row[1]
        sum[2] += row[2]
        sum[3] += row[3]
    return [ceil((sum[i]/n)*100)/100 for i in range(4)]
```

Tabela wartości średnich (**znormalizowanych**):

Rodzaj	Długość kielicha w cm	Szerokość kielicha w cm	Długość płatka w cm	Szerokość płatka w cm
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

Jak widać w powyższej tabeli Iris-versicolor prawie nie różni się od Iris-virginica (jeśli uśrednimy parametry). Jest to duże wyzwanie dla sieci Kohonena i w dalszej części sprawozdania widoczne będzie, że sieć będzie myliła pojedyncze irysy z rodzaju drugiego i trzeciego.

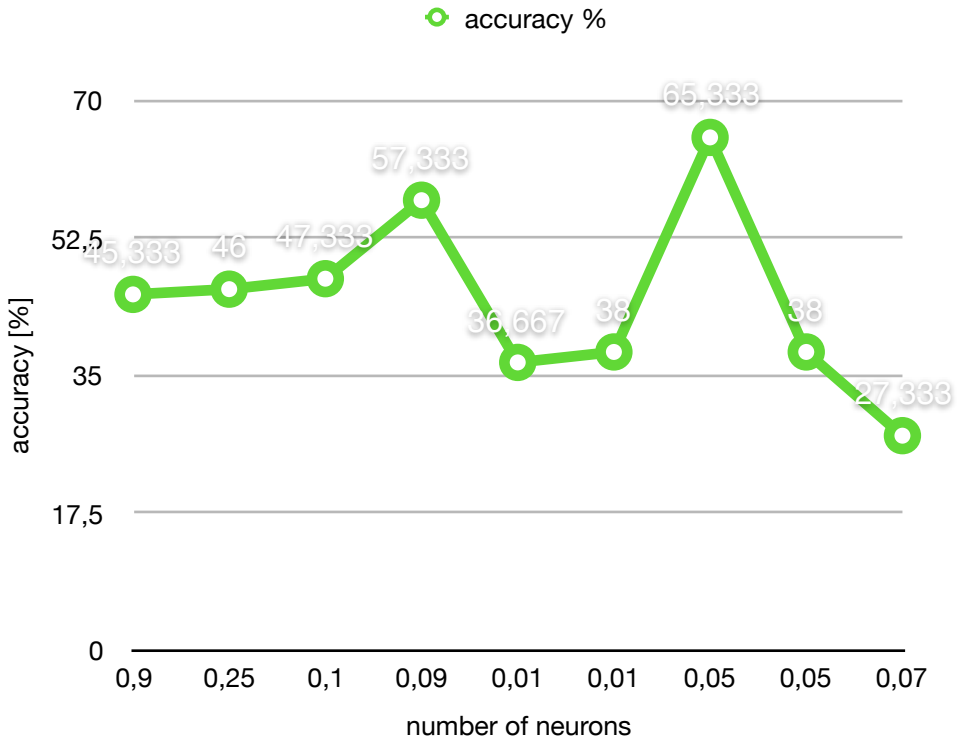
Wyniki

Do uczenia sieci wykorzystałem listę złożoną z 45 wektorów z każdego rodzaju. Metodyka doboru danych treningowych i testowych przedstawia się następująco: Ze zbioru danych (150 irysów) zabieram po 5 z każdego gatunku i umieszczam je na liście danych testowych.

Przeprowadziłem po 10 testów (z uczeniem 25 epok) i w tabeli umieściłem liczbę poprawnie rozpoznanych irysów z grupy 15 irysów testowych. Jeżeli w procesie uczenia uzyskano mniej niż 3 neurony aktywne, wówczas automatycznie liczba poprawnych rozpoznań wynosi 0.

1. Learning rate 17x17 grid

lRate	1	2	3	4	5	6	7	8	9	10	num. of correct results	accuracy %
0,9	13	8	14	13	13	**0**	**0**	**0**	7	**0**	68	45,3333333333333
0,5	14	13	15	12	**0**	**0**	**0**	**0**	**0**	15	69	46
0,25	**0**	**0**	**0**	15	15	15	11	15	**0**	**0**	71	47,3333333333333
0,2	12	**0**	**0**	14	15	**0**	**0**	15	15	15	86	57,3333333333333
0,1	**0**	13	**0**	14	**0**	15	**0**	**0**	**0**	13	55	36,6666666666667
0,09	15	14	**0**	**0**	15	**0**	**0**	**0**	13	**0**	57	38
0,07	15	14	**0**	13	13	14	14	15	**0**	**0**	98	65,3333333333333
0,05	15	15	**0**	**0**	**0**	**0**	15	12	**0**	**0**	57	38
0,01	**0**	15	**0**	13	**0**	13	**0**	**0**	**0**	**0**	41	27,3333333333333



2. Learning rate 20x20 grid

lRate	1	2	3	4	5	6	7	8	9	10	num. of correct results	accuracy %
0,9	8	9	9	13	8	6	8	10	**0**	**0**	71	47,3333333333333
0,5	13	14	13	15	10	15	14	15	**0**	14	123	82
0,25	15	**0**	14	**0**	**0**	**0**	14	14	13	15	85	56,6666666666667
0,2	14	**0**	7	14	8	**0**	14	**0**	15	**0**	72	48
0,1	**0**	15	15	12	0	12	15	14	14	12	109	72,6666666666667
0,09	**0**	**0**	15	15	14	14	15	15	14	15	117	78
0,07	15	13	**0**	14	12	15	14	15	15	13	126	84
0,05	**0**	15	15	13	12	**0**	14	11	**0**	**0**	80	53,3333333333333
0,01	14	**0**	**0**	15	15	15	15	15	14	15	118	78,6666666666667

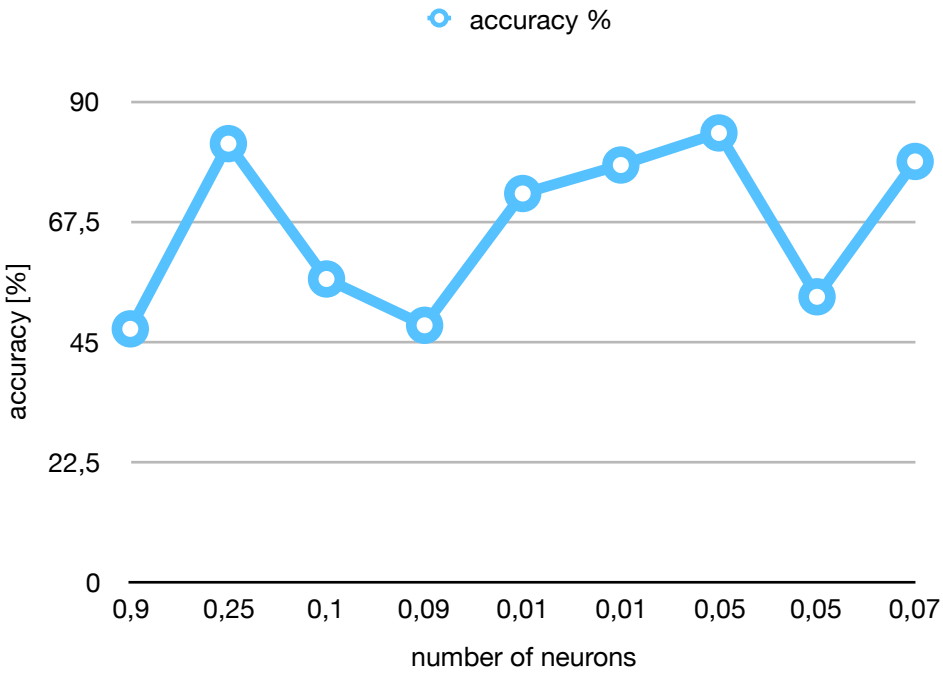
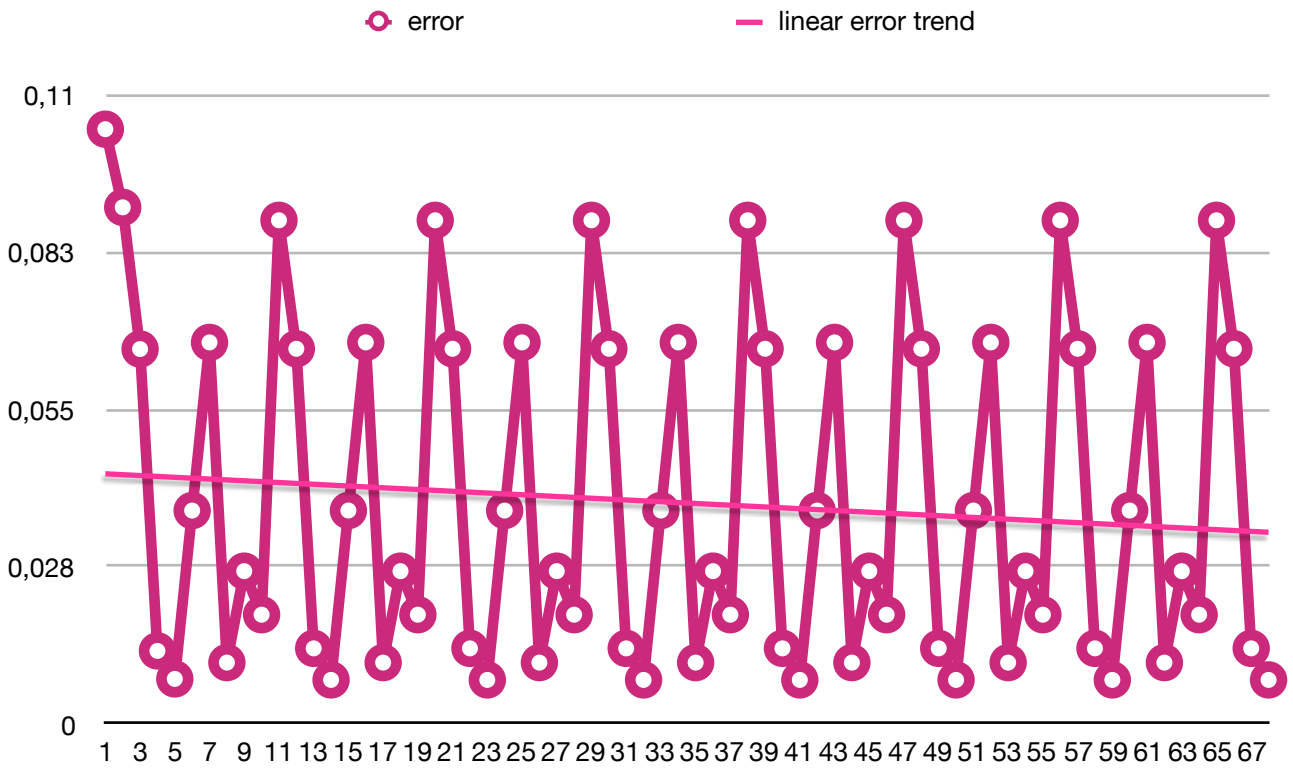


Tabela uśrednionych wartości parametrów znormalizowanych oraz (na zielono) uzyskanych wag neuronów zwycięskich:

Rodzaj	Długość kielicha w cm	Długość kielicha w cm	Szerokość kielicha w cm	Szerokość kielicha w cm	Długość płatka w cm	Długość płatka w cm	Długość płatka w cm	Długość płatka w cm
Iris-setosa	0,81	0,802	0,55	0,545	0,24	0,237	0,04	0,039
Iris-versicolor	0,75	0,751	0,35	0,356	0,54	0,529	0,17	0,164
Iris-virginica	0,71	0,705	0,32	0,327	0,6	0,587	0,22	0,222

Przykładowy wykres błędu (odległości wektora wag od wektora wejściowego):



Wnioski

1. W odróżnieniu od neuronu Hebba, neuron Kohonena zachowuje swoje wagi na określonym poziomie. Wagi dopasowują się wartościowo do wektorów wejściowych. Tym samym neuron o wadze najbardziej zbliżonej do wektora wejściowego będzie mu odpowiadał przy prezentacji.
2. Ucząc neurony wektorami wejściowymi o zbliżonych do siebie wartościach uzyskamy wektor wag o wartościach zbliżonych do uśrednionych wartości wszystkich wektorów wejściowych.
3. Z wykresu zmiany odległości wektora wag od wektora wejściowego wynika, że odległość zmniejsza się do minimalnej wartości już po ok. 5 wygranych (tym samym - aktualizacjach wag). Wykres wygląda na poszarpany, lecz w tak czy inaczej odległość jest stosunkowo mała. Nakładając linie trendu okazuje się, że odległość ma lekką tendencje do zmniejszania się. Oscylacja którą widać na wykresie wynika także ze współczynnika uczenia.
4. Obserwacje wykazały, że dla większej ilości neuronów w siatce, rzadziej zdarza się, że grupa nie podzieli się na przynajmniej 3 neurony. Przy mniejszej ilości często uzyskiwałem na koniec uczenia 2 aktywne neurony (podział na 2 irysy).
5. Najlepsze i najbardziej stabilne wyniki uzyskałem dla współczynnika uczenia równego 0,07. Dla dużych współczynników, często zdarzała się mała ilość poprawnych odpowiedzi dla grupy testowej. Dla za małego, natomiast zdarzało się często, że uzyskiwałem podział na tylko 2 neurony.

Listing

main.py:

```
# @Author: Mikołaj Stępniewski <maikelSoFly>
# @Date:   2017-12-16T02:09:12+01:00
# @Email:  mikolaj.stepniewski1@gmail.com
# @Filename: main.py
# @Last modified by:   maikelSoFly
# @Last modified time: 2017-12-17T15:20:01+01:00
# @License: Apache License  Version 2.0, January 2004
# @Copyright: Copyright © 2017 Mikołaj Stępniewski. All rights reserved.



from math import ceil
from math import floor
from neurons import *
from data import *
from progressBar import *
import random
import copy
from prettytable import PrettyTable


def countUniqueItems(arr):
    return len(Counter(arr).keys())

def getMostCommonItem(arr):
    return Counter(arr).most_common(1)[0][0]

def averageParameters(species, n=50):
    sum = [0.0 for _ in range(4)]
    for row in species:
        sum[0] += row[0]
        sum[1] += row[1]
        sum[2] += row[2]
        sum[3] += row[3]
    return [ceil((sum[i]/n)*100)/100 for i in range(4)]


""" Main training function !!! """
def train(kohonenGroup, trainingData):
    pBar = ProgressBar()
    print('\n {} + {} + {}'.format(speciesNames[0], speciesNames[1], speciesNames[2]))
    pBar.start(maxVal=epochs)

    for i in range(epochs):
        testWinners = kohonenGroup.train(trainingData, histFreq=20)
        pBar.update()

    return testWinners




if __name__ == '__main__':

    """ Training parameters """
    epochs = 30
    decay = (epochs)*13000
    neuronGrid = [20, 20]
    lRate = 0.07    # 0.07 one of the best

    """ Exclude number of irises from total data set
    and add to test data """
    noExcludedIrises = 5

    dataUrl = 'http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
    speciesNames = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

```
data = DataReader(url=dataUrl, delimiter=',').parse()
testData = []

for j in range(len(data)):
        data[j].pop()                       # remove species name
        data[j] = [float(i) for i in data[j]]   # cast str elements to float
        data[j] = normalizeInputs(data[j])      # normalize elements to 0...1 values

irisDict = {'setosa': data[:50], 'versicolor': data[50:100], 'virginica': data[100:]}
speciesArr = np.split(np.array(data), 3)


""" Pop random irises from dict to testData """
for i in range(noExcludedIrises):
    index = np.random.randint(50-i)
    testData.append(irisDict['setosa'].pop(index))
    testData.append(irisDict['versicolor'].pop(index))
    testData.append(irisDict['virginica'].pop(index))


kohonenGroup = KohonenNeuronGroup(
    numOfInputs=4,
    numOfNeurons=neuronGrid,
    processFunc=euklidesDistance,
    lRateFunc=simpleLRateCorrection(decay),
    lRate=lRate
)


print('lRate0: {:.2f}\tdecay: {}\tneurons in group: {:d}\tepochs: {:d}'.format(
    kohonenGroup._lRate, decay, kohonenGroup['totalNumOfNeurons'], epochs
))

print('\n•Averages:')
for i, species in enumerate(speciesArr):
    print('{} \t{}'.format(averageParameters(species), speciesNames[i]))
print()



""" Training & testing """

trainingData = []
trainingData.extend(irisDict['setosa'])
trainingData.extend(irisDict['versicolor'])
trainingData.extend(irisDict['virginica'])

trainingWinners = train(kohonenGroup, trainingData)
numOfActiveNeurons = countUniqueItems(trainingWinners)
trainingWinners = np.split(np.array(trainingWinners), 3)

mostActiveNeurons1 = [getMostCommonItem(row) for row in trainingWinners]
mostActiveNeurons = [getMostCommonItem(row)._iid for row in trainingWinners]
print('\n\n•Training Summary:')
table1 = PrettyTable()
table1.field_names = ['Total active', 'Most active', 'Last lRate']
table1.add_row([numOfActiveNeurons, mostActiveNeurons, kohonenGroup._currentLRate])
print(table1)


testWinners = kohonenGroup.classify(testData)

testWinners = np.split(np.array(testWinners), len(testData)/3)

print('\n\n•Test Results:')
table = PrettyTable()
table.field_names = [speciesNames[0], speciesNames[1], speciesNames[2]]
for row in testWinners:
    table.add_row([neuron._iid for neuron in row ])

print(table)
```

```
    for neuron in mostActiveNeurons1:
        print('\n')
        print('▬' * 25, '   [neuron: {:d}]\n\n'.format(neuron._iid))
        for row in neuron._weights:
            print(row)


    answ = input('Print error history?\ty/n: ')
    if answ == 'y':
        for neuron in mostActiveNeurons1:
            print('\n')
            print('▬' * 25, '   [neuron: {:d}]\n\n'.format(neuron._iid))
            for row in neuron._errorHist:
                print(row)
```

neurons.py:

```
# @Author: Mikołaj Stępniewski <maikelSoFly>
# @Date:   2017-12-12T17:15:51+01:00
# @Email:  mikolaj.stepniewski1@gmail.com
# @Filename: neurons.py
# @Last modified by:   maikelSoFly
# @Last modified time: 2017-12-16T23:31:39+01:00
# @License: Apache License  Version 2.0, January 2004
# @Copyright: Copyright © 2017 Mikołaj Stępniewski. All rights reserved.



import sys
# Add the include folder path to the sys.path list
sys.path.append('../include')
from neuron import *
from supportFunctions import *
from collections import Counter


class KohonenNeuron(Neuron):
    def __init__(self, numOfInputs, processFunc, iid, lRate=0.1):
        Neuron.__init__(self, numOfInputs, iid, activFunc=None, lRate=lRate, bias=0)
        self.__dict__['_winnerCounter'] = 0
        self.__dict__['_processFunc'] = processFunc
        self.__dict__['_startWeights'] = self._weights[:]
        self.__dict__['_errorHist'] = []

    def process(self, vector):
        self._error = self._processFunc(vector, self._weights)
        return self._error

    def train(self, vector):
        for i in range(len(self._weights)):
            self._weights[i] += self._lRate * (vector[i] - self._weights[i])

    def resetWeights(self):
        self._weights = self._startWeights[:]


""" Simple WTA for now...

    Winner is neuron with least distance
    between weights vector and input vector """
class KohonenNeuronGroup:
    def __init__(self, numOfInputs, numOfNeurons, processFunc, lRateFunc, lRate=0.1):
        self.__dict__['_numOfNeurons'] = numOfNeurons
        self.__dict__['_lRate'] = lRate
        self.__dict__['_numOfInputs'] = numOfInputs
        self.__dict__['_neurons'] = None
        self.__dict__['_processFunc'] = processFunc
        self.__dict__['_lRateFunc'] = lRateFunc
        self.__dict__['_currentLRate'] = None
```

```
        self._neurons = [[KohonenNeuron(numOfInputs, processFunc, iid=i*numOfNeurons[0]+j, lRate=lRate)
            for i in range(numOfNeurons[0])]
            for j in range(numOfNeurons[1])
        ]


    def resetWeights(self):
        for row in self._neurons:
            for neuron in row:
                neuron.resetWeights()

    def resetWins(self):
        for row in self._neurons:
            for neuron in row:
                neuron._winnerCounter = 0

    def setLRate(self, lRate):
        self._currentLRate = lRate
        for row in self._neurons:
            for neuron in row:
                neuron._lRate = lRate


    def train(self, vectors, histFreq=1, retMostCommon=False):
        winners = []
        for i, vector in enumerate(vectors):
            winner = None
            for row in self._neurons:
                for neuron in row:
                    neuron.process(vector)
                    if winner == None:
                        winner = neuron
                    elif winner != None:
                        if neuron._error < winner._error:
                            winner = neuron

            """ Winner Takes All """
            if winner._winnerCounter % histFreq == 0:
                winner._errorHist.append(winner._error)
            winner._winnerCounter += 1
            winner.train(vector)
            winners.append(winner)

        self.setLRate(self._lRateFunc(self._lRate))

        if retMostCommon:
            return Counter(winners).most_common(1)[0][0]

        return winners


    """ Basicaly the same as above, but without updating weights """
    def classify(self, vectors):
        winners = []
        for i, vector in enumerate(vectors):
            winner = None
            for row in self._neurons:
                for neuron in row:
                    neuron.process(vector)
                    if winner == None:
                        winner = neuron
                    elif winner != None:
                        if neuron._error < winner._error:
                            winner = neuron

            winners.append(winner)

        return winners


    """ Access methods """
```

```
def __getitem__(self, key):
    if key == 'totalNumOfNeurons':
        return sum(len(x) for x in self._neurons)
```

Źródła

„Sieci neuronowe do przetwarzania informacji” S. Osowski