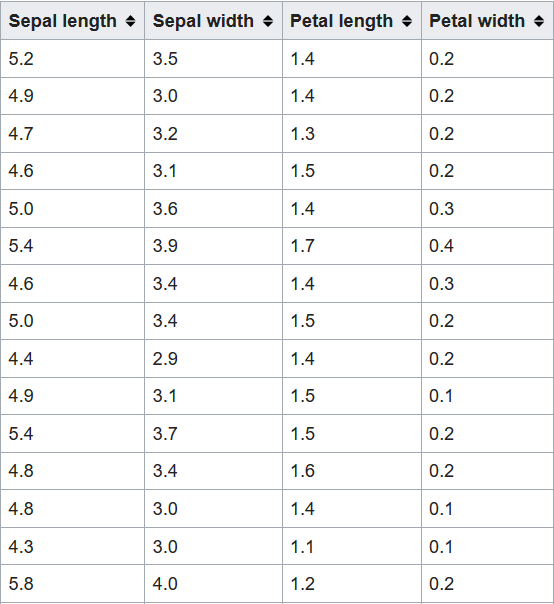
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Jarosz Wojciech | Gr. 1 | PSI | Sprawozdanie 5 | Temat: Budowa i działanie sieci Kohonena dla WTA. |

1. Cel ćwiczenia:

Celem ćwiczenia jest poznanie budowy i działanie sieci Kohenena przy wykorzystaniu reguły WTA do odwzorowywania istotych cech kwiatów.

1. Wykonane zadania:
   1. Przygotowano dane uczące i testujące zawierające numeryczny opis cech kwiatów.

Wykorzystano zestaw: <https://en.wikipedia.org/wiki/Iris_flower_data_set>



* 1. Przygotowano (wykorzystano bibliotekę neuroph) sieć Kohenena i algorytm uczenia oparty o regułę Winner Takes All (WTA).
  2. Nauczono sieć dla różnych współczynników uczenia.
  3. Przetestowano sieć.

1. Syntetyczny opis budowy oraz wykorzystanego algorytmu uczenia:

Sieć Kohennena wykorzystuje podejście, które jest przeciwieństwem uczenia się z nadzorem (z nauczycielem). W podejściu tym są problemy, w których mamy podzielić zbiór obiektów na grupy, jednak nie mamy żadnych danych treningowych, z których moglibyśmy wywnioskować reguły tego podziału

Zasady:

* Liczba wejść jest równa liczbie parametrów, którymi opisano obiekty
* Każdy węzeł przechowuje wektor wag o wymiarze identycznym z wektorami wejściowymi
* Każdy węzeł oblicza swój poziom aktywacji jako iloczyn skalarny wektora wag i wektora wejściowego (podobnie jak w zwykłym neuronie)
* Ten węzeł, który dla danego wektora wejściowego ma najwyższy poziom aktywacji, zostaje zwycięzcą i jest uaktywniony
* Wzmacniamy podobieństwo węzła-zwycięzcy do aktualnych danych wejściowych poprzez dodanie do wektora wag wektora wejściowego (z pewnym współczynnikiem uczenia)

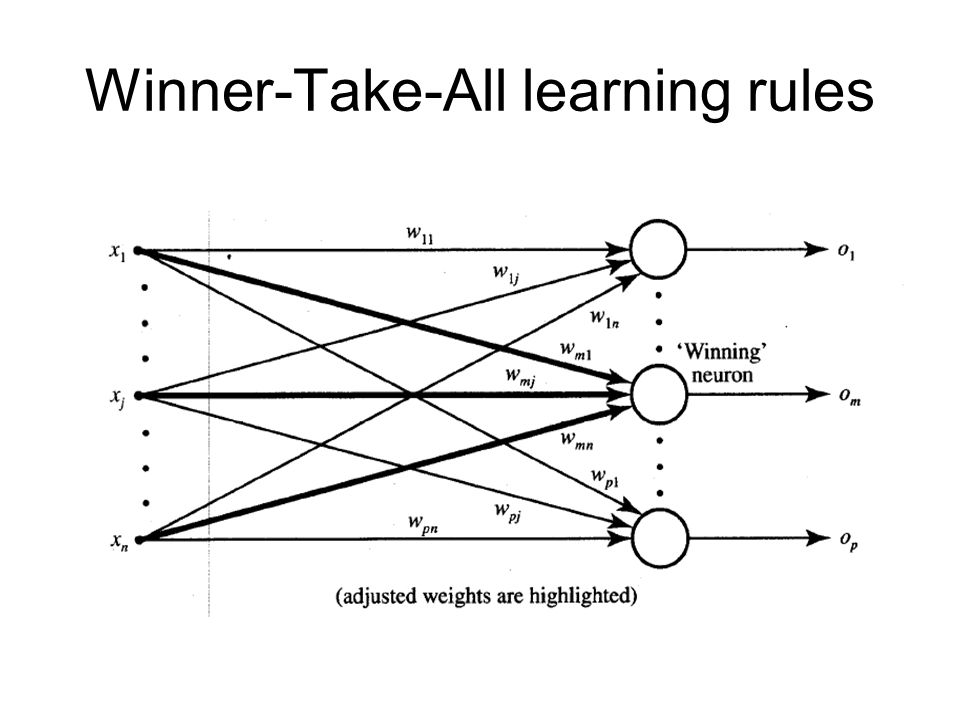
Wykorzystanie utworzonej w ten sposób mapy polega na tym, że zbiór obiektów umieszczamy na wejściu sieci i obserwujemy, które węzły sieci się uaktywniają. Obiekty podobne powinny trafiać w podobne miejsca mapy.

Algorytm WTA:

* Inicjalizacja wag sieci.
* Normalizacja danych wejściowych
* Obliczenie odległości wektora wejściowego do wag każdego z neuronów:
* Wybranie neuronu zwycięzcy (wygrywającego) dla którego odległość wag od wektora wejściowego jest najmniejsza (na podstawie Euklidesowej miary odległości).
* Zmiana wartości poszczególnych wag tego neuronu przyjmując.

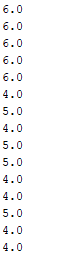
wt+1 – aktualnie obliczana waga   
wt – poprzednia waga  
lrate – współczynnik uczenia  
xi – wejście

* powtórzenie kroków 2-5 dla wszystkich przykładów uczących.



1. Zestawienie otrzymanych wyników:

Przykładowe output:



Do testowania wybrano po 5 kwiatów z każdej grupy.

Otrzymane wyniki należy interpretować w następujący sposób:

- 5 kolejnych wyników należy do jednej grupy

- jeżeli wszystkie 5 kolejnych wyników należy do tej samej grupy według sieci, to otrzymamy tą samą liczbę(numer neurony, który został przyporządkowany do danej grupy)

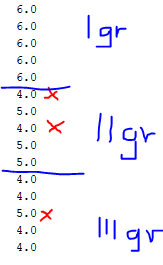
- jeżeli otrzymaliśmy w 5 kolejnych wynikach inne liczby, oznacza to błąd uczenia

W powyższym przykładzie:

- pierwsze pięć wyników zostało dobrze przyporządkowane do jednej grupy( neuron nr 6)

- w kolejnych pięciu wynikach pojawił się błąd – złe przyporządkowanie do grupy. Dwa na pięć kwiatów nie pasują do reszty – został aktywowany niewłaściwy neuron

- w ostatnich pięciu wynikach tylko jeden kwiat nie został dobrze przyporządkowany



Zestawienie wyników dla różnych współczynników uczenia:

Współczynnik uczenia = 0.5

Współczynnik uczenia = 0.1:

Rozpoznawanie tylko 2 grup kwiatów:



Pierwszy typ kwiatów(pierwsze pięć wyników) przyporządkowane zostawały do jednej z grup, natomiast nie rozróżniane były pozostałe 2 grupy (kolejne 10 wyników)

Sporadycznie pojawiała się 3 grupa (maxymalnie dla jednego wyniku)

Współczynnik uczenia = 0.01:

Sytuacja taka jak w poprzednim przypadku:



Współczynnik uczenia = 0.001;

* + 1. W większości przypadków rozpoznawanie tylko 1 grup ,
    2. ale w niektórych przypadkach otrzymywaliśmy drugą grupę, co ciekawe wszystkie 5 wyników pierwszego typu kwiatów zostawało przyporządkowywane p0prawnie

1. 2)

Współczynnik uczenia =0.5:

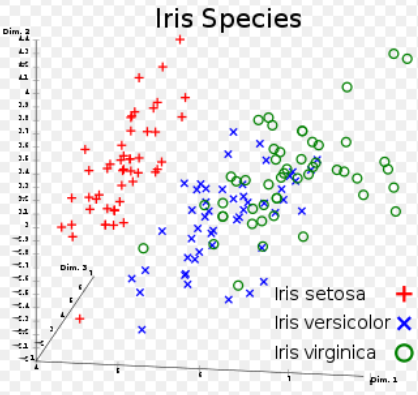


1. Analiza i dyskusja błędów:

Skuteczność uczenia sieci silnie wpływa od współczynnika uczenia sieci.

Najczęściej występującym błędem był problem przyporządkowania kwiatów z gatunków I*ris versicolor* i *Iris virginica*. Gatunek *Iris setosa* był poprawnie przyporządkowany w zdecydowanej większości przypadków.

Powodem tego może być sposób rozmieszczenia tych gatunków ( źródło wikipednia.org)



Dla współczynnika uczenia 0.5 dostaliśmy gorsze wyniki od wyników dla współczynnika uczenia 0.1

Wnioski:

1. Możemy zauważyć na powyższym obrazku, że wiele kwiatów z gatunków I*ris versicolor* i *Iris virginica*, nachodzi na siebie, co wiąże się z większym problemem przyporządkowania do oddzielnych grup tych dwóch gatunków.

2. Im mniejszy współczynnik uczenia tym gorsze rezultaty uczenia sieci.

3. Dla współczynnika uczenia 0.5 otrzymywane wyniki pogarszały się , stąd wniosek, że najbardziej optymalnym współczynnikiem uczenia się tej sieci był współczynnik uczenia = 0.1.

4. Przy współczynniku uczenia = 0.1 sieć spełniała swoje zadanie i poprawnie rozwiązywała problem przyporządkowywania kwiatów do odpowiednich gatunków, jednak występowały przypadki, że nawet przy tym współczynniku otrzymywaliśmy błędne przyporządkowania z powodu tego, że ich charakterystyka (dane wejściowe) były podobne do siebie.

1. Listning kodu:

WTA.java

**import** org.neuroph.core.Connection;  
**import** org.neuroph.core.Layer;  
**import** org.neuroph.core.Neuron;  
**import** org.neuroph.core.data.DataSet;  
**import** org.neuroph.core.data.DataSetRow;  
**import** org.neuroph.core.learning.LearningRule;  
**import** java.util.Iterator;  
*/\*\*  
 \* Created by Wojtek on 03.01.2018.  
 \*/***public class** WTA **extends** LearningRule {  
  
  
 **private static final long *serialVersionUID*** = 1L;  
  
 **private double learningRate** = 0.5; *//współćzynnik czenia* **private int iteration** = 10000; *//ilosc iteracji* **public** WTA(){  
 **super**();  
 }  
  
 *// funkcja ucząca siec* @Override  
 **public void** learn(DataSet trainingSet) {  
  
 **for**(**int** i = 0; i < **iteration**; i++) {  
 Iterator<DataSetRow> iterator = trainingSet.iterator();  
 **while** (iterator.hasNext() && !isStopped()) {  
 DataSetRow dataSetRow = normalize(iterator.next());  
 learnPattern(dataSetRow);  
 }  
 **learningRate**=**learningRate**\*Math.*exp*(-(**double**)i/**iteration**);  
 }  
 }  
  
 *//funkcja normalizyjąca dane wejściowe* **private** DataSetRow normalize(DataSetRow dataSetRow){  
 **double**[] input = dataSetRow.getInput();  
 **double** l = Math.*sqrt*(Math.*pow*(input[0], 2)+Math.*pow*(input[1], 2)+Math.*pow*(input[2], 2)+Math.*pow*(input[3], 2));  
 input[0] = input[0]/l;  
 input[1] = input[1]/l;  
 input[2] = input[2]/l;  
 input[3] = input[3]/l;  
 DataSetRow dataSetRow1 = **new** DataSetRow();  
 dataSetRow1.setInput(input);  
 **return** dataSetRow1;  
 }  
 *// węzeł oblicza swój poziom aktywacji jako iloczyn skalarny wektora wag i wektora wejściowego (podobnie jak w zwykłym neuronie)  
 //Ten węzeł, który dla danego wektora wejściowego ma najwyższy poziom aktywacji, zostaje zwycięzcą i jest uaktywniony* **private void** learnPattern(DataSetRow dataSetRow){  
 **neuralNetwork**.setInput(dataSetRow.getInput());  
 **neuralNetwork**.calculate();  
 Neuron winner = getClosestNeuron();  
 **if**(winner.getOutput() == 0)  
 **return**;  
 Layer map = **neuralNetwork**.getLayerAt(1);  
 **int** indexOfWinner = map.indexOf(winner);  
 **for**(**int** i = 0; i < map.getNeuronsCount(); i++){  
 **if**(i == indexOfWinner) **continue**;  
 map.getNeurons()[i].setOutput(0);  
 }  
 **for**(**int** i = 0; i < map.getNeuronsCount(); i++){  
 **if**(map.getNeurons()[i].getOutput() == 0) **continue**;  
 changeWeights(map.getNeurons()[i]);  
 }  
 }  
  
 *//uaktualnienie wag* **private void** changeWeights(Neuron neuron){  
 **for**(Connection conn : neuron.getInputConnections()) {  
 **double** dWeight = **learningRate**\*(conn.getInput() - conn.getWeight().getValue());  
 conn.getWeight().inc(dWeight);  
 }  
 }  
  
 **private** Neuron getClosestNeuron() {  
 Neuron winner = **new** Neuron();  
 **double** max = 0;  
 **for**(Neuron n: **this**.**neuralNetwork**.getLayerAt(1).getNeurons()){  
 **if**(n.getOutput() > max){  
 max = n.getOutput();  
 winner = n;  
 }  
 }  
 **return** winner;  
 }  
  
 **public void** setIteration(**int** iteration){  
 **this**.**iteration** = iteration;  
 }  
 **public int** getIteration(){  
 **return iteration**;  
 }  
 **public double** getLearningRate() {  
 **return learningRate**;  
 }  
  
 **public void** setLearningRate(**double** learningRate) {  
 **this**.**learningRate** = learningRate;  
 }  
  
}

Main.java

**import** org.neuroph.core.data.BufferedDataSet;  
**import** org.neuroph.core.data.DataSetRow;  
**import** org.neuroph.nnet.Kohonen;  
  
**import** java.util.ArrayList;  
  
  
**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.util.List;  
  
**public class** Main {  
  
 **public static void** main(String[] args) {  
 File learningDataFile = **new** File(**"learningData.txt"**);  
 File testingDataFile = **new** File(**"testingData.txt"**);  
 BufferedDataSet dataSet;  
 **try** {  
 dataSet = **new** BufferedDataSet(learningDataFile, 4, 10,**";"**);  
 WTA wta = **new** WTA();  
 Kohonen kohonen = **new** Kohonen(4, 10);  
 kohonen.randomizeWeights();  
 kohonen.setLearningRule(wta);  
 kohonen.learn(dataSet);  
  
 dataSet = **new** BufferedDataSet(testingDataFile, 4, 10, **";"**);  
 List<Double> winners = **new** ArrayList<>();  
 **for**(DataSetRow row: dataSet.getRows()){  
 kohonen.setInput(row.getInput());  
 kohonen.calculate();  
 **double**[] output = kohonen.getOutput();  
 **double** tmp = 0.0;  
 **double** winner = 0.0;  
 **for**(**int** i = 0; i < output.**length**; i++){  
 **if**(output[i] > tmp){  
 tmp = output[i];  
 winner = i;  
 }  
 }  
 winners.add(winner);  
 }  
 **for**(**double** d: winners){  
 System.***out***.println(d);  
 }  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
}