**PODSTAWY SZUCZNEJ INTELIGENCJI**

Scenariusz 4

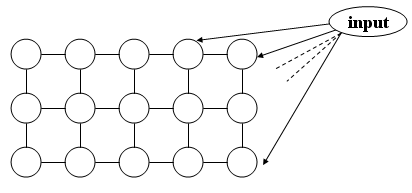
Maria Podkalicka, IS grupa 3

**Temat sieci:** Budowa i działanie sieci Kohonena dla WTA.

**Cel ćwiczenia**: Poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły WTA do odwzorowywania istotnych cech liter alfabetu.

**Sieci Kohonena** są szczególnym przypadkiem algorytmu realizującego uczenie się bez nadzoru. Ich głównym zadaniem jest organizacja wielowymiarowej informacji (np. obiektów opisanych 50 parametrami w taki sposób, żeby można ją było prezentować i analizować w przestrzeni o znacznie mniejszej liczbie wymiarów, czyli mapie (np. na dwuwymiarowym ekranie). Warunek: rzuty "podobnych" danych wejściowych powinny być bliskie również na mapie. Sieci Kohonena znane są też pod nazwami Self-Organizing Maps, Competitive Filters.

Topologia sieci Kohonena odpowiada topologii docelowej przestrzeni. Jeśli np. chcemy prezentować wynik na ekranie, rozsądnym modelem jest prostokątna siatka wezłów (im więcej, tym wyższą rozdzielczość będzie miała mapa):



Zasady działania sieci Kohonena:

* Wejścia (tyle, iloma parametrami opisano obiekty) połączone są ze wszystkimi węzłami sieci
* 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)
* Każdy węzeł może być stowarzyszony z pewnymi innymi, sąsiednimi węzłami - wówczas te węzły również zostają zmodyfikowane, jednak w mniejszym stopniu.

Inicjalizacja wag sieci Kohonena jest losowa. Wektory wejściowe stanowią próbę uczącą, podobnie jak w przypadku zwykłych sieci rozpatrywaną w pętli podczas budowy mapy. 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.

Ciekawym zastosowaniem jest próba wykorzystania sieci Kohonena w eksploracji Internetu. Podobne pod względem treści dokumenty możemy rozłożyć na dwuwymiarowej mapie tak, by leżały w pobliżu siebie - prowadzi to do powstania mapy, na której można wyróżnić obszary tematyczne.

**Uczenie konkurencyjne WTA**

W uczeniu tego typu neurony współzawodniczą ze sobą, aby stać się aktywnymi (pobudzonymi). W odróżnieniu od uczenia Hebba, gdzie dowolna liczba neuronów mogła być pobudzona, w

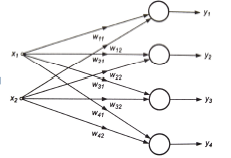
uczeniu konkurencyjnym tylko jeden neuron może być aktywny, a pozostałe są w stanie spoczynkowym.

Grupa neuronów współzawodniczących otrzymuje te same sygnały wejściowe xi. W zależności od aktualnych wartości wag sygnały wyjściowe neuronów



różnią się między sobą. W wyniku porównania tych sygnałów zwycięża neuron, którego wartość ym jest największa. Neuron zwycięzca przyjmuje na swoim wyjściu stan 1, a pozostałe (przegrywające) stan 0.

Neuron typu WTA – mają stopień wejściowy w postaci standardowego sumatora dokonującego sumowania sygnałów wejściowych z odpowiednimi wagami.

****

**Dane uczące**

Dane uczące zawierają numeryczny opis cech kwiatów Irysa. Kwiat ten został określony za pomocą 4 atrybutów - długości i szerokości płatków Irysa: petal i sepal. Do programu wykorzystałam dane opracowane przez Brytyjskiego biologa Ronalda Fishera. Oto przykładowe dane dla gatunku setosa:

|  |
| --- |
| 1. 5.1,3.5,1.4,0.2 |
|  |

|  |
| --- |
| 2. 4.9,3.0,1.4,0.2 |
|  |

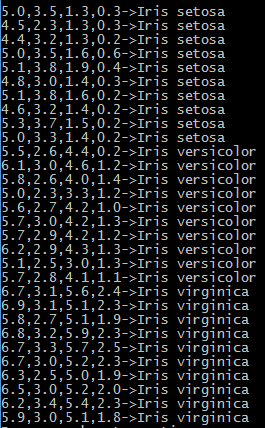
|  |
| --- |
| 3. 4.7,3.2,1.3,0.2 |
|  |

|  |
| --- |
| 4. 4.6,3.1,1.5,0.2 |
|  |

5. 5.0,3.6,1.4,0.2

**Analiza testowania i wyłonionych cech**

Program rozwiązuje problem jakim jest określenie gatunku dla nowego kwiatu irysa w zależności od wielkości jego płatków. Na podstawie 4 podanych parametrów program poda właściwy gatunek obiektu –setosa, versicolor lub virginicacs. Wynik działania przedstawia screen konsoli.



Na wyżej widocznym zdjęciu widać, że sieć została poprawnie nauczona i wszystkie gatunki zostały poprawnie rozpoznane. Uczenie zostało przeprowadzone dla różnych współczynników uczenia. Współczynnik uczenia może przyjmować wartości z przedziału (0,1). Im większa wartość współczynnika uczenia tym mniejsza liczba iteracji, co ma wpływ na działanie programu i powoduje uzyskanie błędnych wyników. Aby zredukować liczbę błędnych wyników można zwiększyć liczbę iteracji.

**Wnioski**

Algorytmy "zwycięzca bierze wszystko" (WTA), w których tylko jeden neuron może podlegać adaptacji w każdej iteracji, są algorytmami słabo zbieżnymi - szczególnie przy dużej liczbie neuronów. Przy konstrukcji skutecznego algorytmu uczenia sieci Kohonena istotny jest dobór wielkości promienia sąsiedztwa oraz stałej uczenia. Tylko neuron wygrywający oraz ewentualnie neurony znajdujące się w jego sąsiedztwie mają możliwość uaktualnienia swoich wag, tak by jeszcze bardziej zbliżyć je do podanego właśnie wektora uczącego. Gdy tylko neuron zwycięski zmienia swoje wagi, mamy do czynienia z najbardziej skrajną formą uczenia konkurencyjnego, zwaną „zwycięzca bierze wszystko”.

Dla sieci samoorganizującej Kohonena bardzo ważna jest ilość danych uczących ponieważ przy zbyt małej ilości tych danych można otrzymać błędne wyniki. W programie zbiór 30 wersów danych okazał się wystarczający.

Listing kodu:

|  |
| --- |
| var synaptic = require('synaptic'); |
|  |

|  |
| --- |
| var fs = require('fs'); |
|  |

|  |
| --- |
| var Layer = synaptic.Layer, |
|  |

|  |
| --- |
| Network = synaptic.Network; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // set up neural network |
|  |

|  |
| --- |
| var inputLayer = new Layer(4); |
|  |

|  |
| --- |
| var hiddenLayer = new Layer(5); |
|  |

|  |
| --- |
| var outputLayer = new Layer(3); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| inputLayer.project(hiddenLayer); |
|  |

|  |
| --- |
| hiddenLayer.project(outputLayer); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| var network = new Network({ |
|  |

|  |
| --- |
| input: inputLayer, |
|  |

|  |
| --- |
| hidden: [hiddenLayer], |
|  |

|  |
| --- |
| output: outputLayer |
|  |

|  |
| --- |
| }); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| var som = require('node-som'); |
|  |

|  |
| --- |
| var somInstance = new som({ |
|  |

|  |
| --- |
| inputLength: 4, |
|  |

|  |
| --- |
| maxClusters: 10, |
|  |

|  |
| --- |
| loggingEnabled:true, |
|  |

|  |
| --- |
| scale:{ |
|  |

|  |
| --- |
| min:0, |
|  |

|  |
| --- |
| max:10 |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| }); |
|  |

|  |
| --- |
| somInstance.train(trainingData); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| var sample = [5.9,3.0,5.1,1.8]; |
|  |

|  |
| --- |
| var group = somInstance.classify(sample); |
|  |

|  |
| --- |
| console.log(getFlowerName2(group)); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| var trainingData = []; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // read training data file |
|  |

|  |
| --- |
| fs.readFile('iris.txt','utf8',function(err, data){ |
|  |

|  |
| --- |
| if(err) throw err; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // load data into training data array |
|  |

|  |
| --- |
| var lines = data.split("\n"); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for(var i = 0; i < lines.length; i++){ |
|  |

|  |
| --- |
| var line = lines[i].trim(); |
|  |

|  |
| --- |
| var splitLine = line.split(","); |
|  |

|  |
| --- |
| var input = splitLine.slice(0, 4); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| var output = splitLine[4]=='Iris-virginica' ? [0,0,1] |
|  |

|  |
| --- |
| : splitLine[4] == 'Iris-versicolor' ? [0,1,0] |
|  |

|  |
| --- |
| : [1,0,0]; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| trainingData.push({ |
|  |

|  |
| --- |
| input: input, |
|  |

|  |
| --- |
| output: output |
|  |

|  |
| --- |
| }); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // train the network |
|  |

|  |
| --- |
| var learningRate = .01; |
|  |

|  |
| --- |
| for (var i = 1; i <= 10000; i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| for(var j = 0; j < trainingData.length; j++){ |
|  |

|  |
| --- |
| network.activate(trainingData[j].input); |
|  |

|  |
| --- |
| network.propagate(learningRate, trainingData[j].output); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| // if(i%1000 == 0) |
|  |

|  |
| --- |
| // console.log("Training... "+i/100+"% complete. "); |
|  |

|  |
| --- |
| //console.log(network.log); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // use the network to classify flowers based on testing data |
|  |

|  |
| --- |
| fs.readFile('testing\_data.txt','utf8',function(err, data){ |
|  |

|  |
| --- |
| if(err) throw err; |
|  |

|  |
| --- |
| console.log("\n\nResults\n===============================\n"); |
|  |

|  |
| --- |
| var lines = data.split("\n"); |
|  |

|  |
| --- |
| for(var i = 0; i < lines.length; i++){ |
|  |

|  |
| --- |
| var input = lines[i].trim().split(","); |
|  |

|  |
| --- |
| var result = getFlowerName(network.activate(input)); |
|  |

|  |
| --- |
| console.log(lines[i].trim()+" => "+result); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| }); |
|  |

|  |
| --- |
| }); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // helper functions |
|  |

|  |
| --- |
| function getLargestIndex(arr){ |
|  |

|  |
| --- |
| var result = 0; |
|  |

|  |
| --- |
| for(var i = 1; i < arr.length; i++) |
|  |

|  |
| --- |
| if(arr[i] > arr[result]) |
|  |

|  |
| --- |
| result = i; |
|  |

|  |
| --- |
| return result; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| function getFlowerName(arr){ |
|  |

|  |
| --- |
| var index = getLargestIndex(arr); |
|  |

|  |
| --- |
| if(index == 0) |
|  |

|  |
| --- |
| return "Iris-setosa"; |
|  |

|  |
| --- |
| if(index == 1) |
|  |

|  |
| --- |
| return "Iris-versicolor"; |
|  |

|  |
| --- |
| return "Iris-virginica"; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| function getFlowerName2(index){ |
|  |

|  |
| --- |
| if(index < 51) |
|  |

|  |
| --- |
| return "Iris-setosa"; |
|  |

|  |
| --- |
| if(index < 101) |
|  |

|  |
| --- |
| return "Iris-versicolor"; |
|  |

|  |
| --- |
| return "Iris-virginica"; |
|  |

}