Katarzyna Mlicka
Inżynieria Obliczeniowa WIMiIP
Nr indeksu 286120

Podstawy Sztucznej Inteligencji
Scenariusz 5:

Budowa i działanie sieci Kohonena dla WTA.

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

Sztuczna sieć neuronowa uczona w sposób nadzorowany modyfkuje swoje działanie za pomocą informacji z zewnątrz. Takiej korekty nie dostaje sieć uczona w sposób nienadzorowany (bez nauczyciela), która sama wypracowuje funkcje przetwarzania danych np. uporządkowywania, wykrywania regularności w danych, ich klasyfkacji, kodowania itd. Metody uczenia nienadzorowanego są ważnym narzędziem przetwarzania danych w sytuacjach, gdy nie można zorganizować nadzorowanego procesu uczenia.

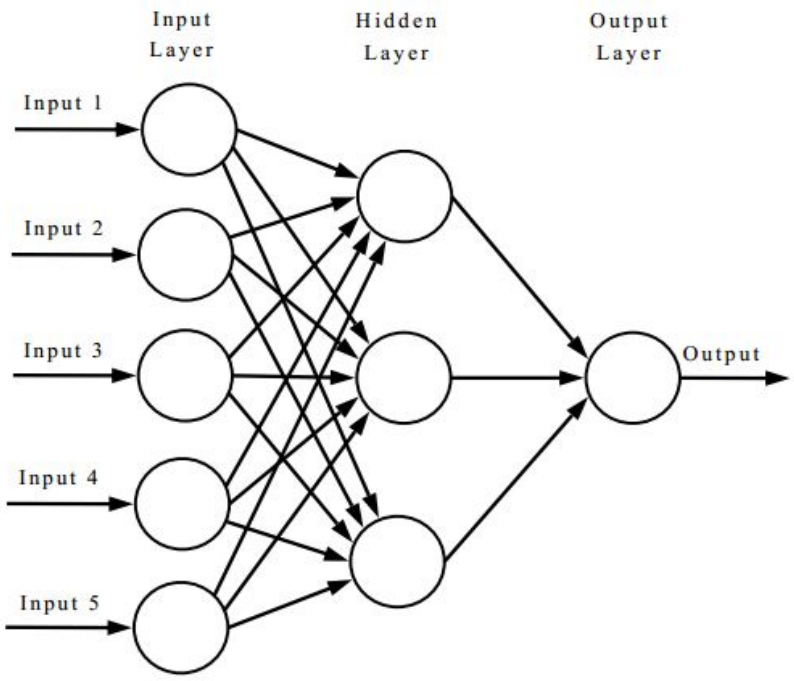
Przykładowe zadania, które mogą być realizowane przez sieci samoorganizujące się:

- wykrywanie podobieństwa - ocena stopnia podobieństwa pomiędzy wzorcami (wg wybranego kryterium)
- analiza składowych głównych - jest to metoda redukcji wymiarowości danych, w której wektory wzorców wejściowych są rzutowane na najistotniejsze kierunki ich zmian w przestrzeni o zredukowanej liczbie wymiarów
- tworzenie map cech - porządkowanie danych według wybranego kryterium podobieństwa
- klasyfkacja - grupowanie danych, w trakcie którego kryterium jest podobieństwo wybranych cech
- określanie prototypu - wskazanie typowego reprezentanta grupy, tzw. Prototypu
- kodowanie - wyznaczanie zbioru prototypów o liczności znacznie mniejszej niż cały zbiór danych wejściowych, zbiór prototypów w możliwie jak najlepszy sposób reprezentuje dane wejściowe (uzyskuje się efekt kompresji stratnej danych).

Jednym z podstawowych sposobów uczenia nienadzorowanego jest tzw. **uczenie konkurencyjne (ang. competitive learning).** W tej metodzie uczenia sieci, poszczególne neurony konkurują ze sobą o prawo do reprezentacji danych wejściowych.

Cechy charaktersytyczne uczenia konkurencyjnego:

- zwykle sieci składające się z jednej warstwy wejściowej i jednej warstwy wyjściowej
- każdy neuron jest połączony ze wszystkim składowymi K-wymiarowego wektora wejściowego x
- neurony są liniowe (funkcja przejścia/aktywacji ma charakter liniowy)



SIECI KOHONENA

Nazwa pochodzi od nazwiska fińskiego uczonego, Teuvo Kohonena. Kohonen zaproponował regułę uczenia konkurencyjnego, w której, w odróżnieniu od reguły standardowej, modyfkacji wag dokonuje się nie tylko dla neuronu zwycięskiego, lecz również dla pewnej liczby neuronów z jego sąsiedztwa.

Sąsiedztwem tym są inne neurony leżące blisko danej jednostki w strukturze grafowej neuronów.

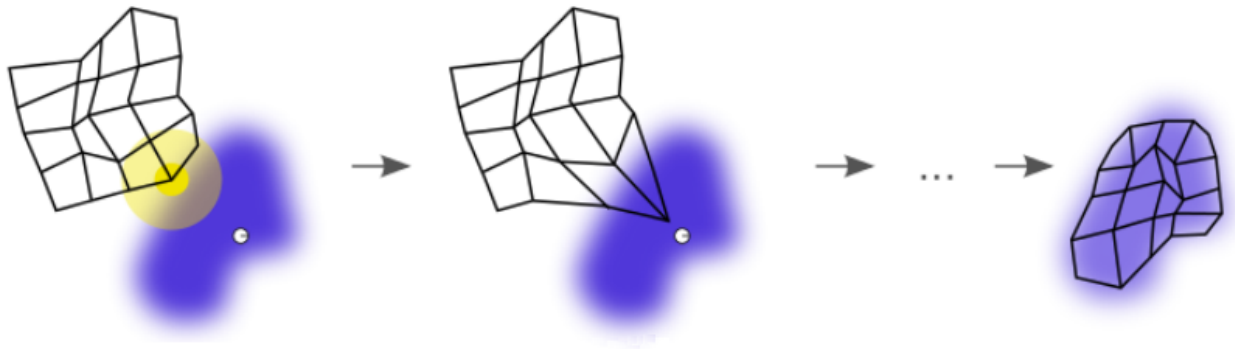
3 etapy funkcjonowania sieci Kohonena:

1. Konstrukcja
2. Uczenie
3. Rozpoznawanie

Sieć neuronowa uczona w trybie bez nauczyciela w celu wytworzenia niskowymiarowej (przeważnie dwuwymiarowej) zdyskretyzowanej reprezentacji przestrzeni wejściowej. Sieć Kohonena wyróżnia się tym od innych sieci, że **zachowuje odwzorowanie sąsiedztwa przestrzeni wejściowej**. Wynikiem działania sieci jest klasyfikacja przestrzeni w sposób grupujący zarówno przypadki ze zbioru uczącego, jak i wszystkie inne wprowadzenia po procesie uczenia.

Zasada działania sieci Kohonena - dane wejściowe trafiają do neuronów a następnie są odwzorowywane na warstwę topologiczną, co daje siatkę neuronow z efektem działania sieci. Neurony mogą być ułożone w siatkę hexagonalną albo prostokątną.

Jak wygląda trening sieci Kohnena:



Ilustracja przedstawiająca trening sieci Kohonena. Niebieska plama jest rozkładem przykładów zestawu uczącego sieć, biała kropka obrazuje aktualnie wylosowany przypadek z tej dystrybucji. W pierwszej fazie (od lewej) sieć jest losowo umieszczona w przestrzeni danych. Węzeł najbliższy wylosowanemu przypadkowi (podświetlony na żółto) jest przesuwany w jego kierunku. Jego sąsiedzi także w mniejszym stopniu zmieniają swoje pozycje (ma to na celu zachowanie równomiernego rozkładu siatki). Po wielu krokach siatka ma tendencję do odwzorowania analizowanych przykładów (po prawej)

WINNER TAKES ALL

W konkurencyjnej metodzie uczenia sieci, tylko **jeden** element wyjściowy może znajdować się w stanie aktywnym. Nazywany jest on zwycięzcą, a schemat takiej reguły aktywacji neuronów określany jest mianem "zwycięzca bierze wszystko" (ang. Winner Takes All = WTA). Tylko jego waga jest dofyfikowana.

Neuron, który wygrał, jeszcze bardziej jest upodobniany do przykładu, którym uczymy.

Każda komórka w warstwie wyjściowej jest połączona ze wszystkimi elementami x_k wzorca wejściowego x za pomocą połączeń wagowych, neuron m-ty jest połączony z warstwą wejściową za pomocą wektora wag $w_m = [w_{m1},w_{m2},...,w_{mK}]$ (K = wymiar wektora wejściowego).

W wyniku zastosowania takiej struktury połączeń m-ty neuron warstwy wyjściowej sieci otrzymuje sygnał pobudzenia:

$$y_m = \sum_{k=1}^{K} w_{mk} x_k = w_m^T x, \quad m = 1, 2, ..., M,$$

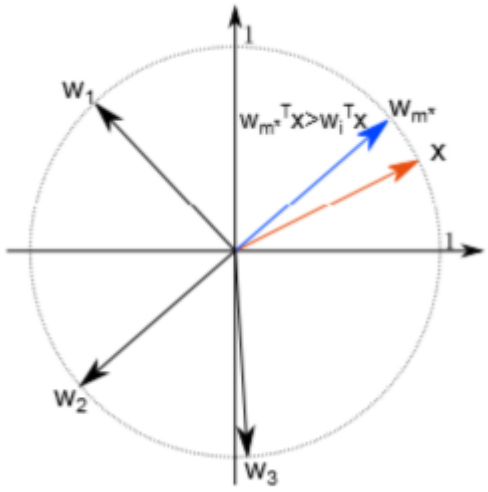
Gdzie K to długość wzorca, a M – ilość neuronów.

Komórka zwycięska warstwy wyjściowej sieci, oznaczona indeksem $m*$, jest to neuron otrzymujący najsilniejszy sygnał pobudzenia y_m. Zatem dla komórki zwycięskiej spełniona jest

nierówność:

$$w_{m^*}^T \cdot x > w_m^T \cdot x, \quad m = 1, 2, ..., M.$$

W procesie samoorganizacji wagi zwycięzcy są modyfikowane. Stosuje się algorytm zmęczenia gdy neuron wygrywa zbyt często. Przestaje on być brany pod uwagę w rywalizacji.



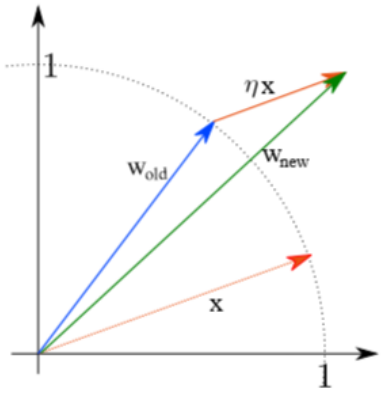
ALGORYM UCZENIA

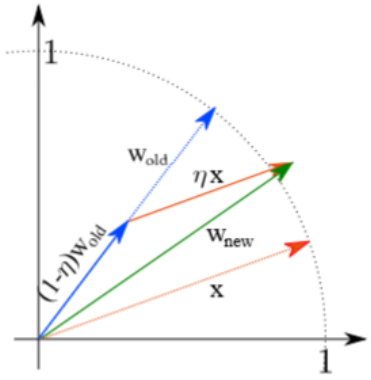
Zakładamy, że przykłady uczące są znormalizowane. Do uczenia sieci używamy następującego algorytmu:

1. *Generujemy losowo znormalizowane wektory wag w_m*
2. *Losujemy wektor x oraz liczymy dla niego aktywację $y_m = w_m^T * x$*
3. *Modyfikujemy wektor zwycięzcy (wizualizacja na wykresach poniżej), a następnie go normalizujemy.*
4. *Zatrzymujemy algorytm po odpowiednio dużej liczbie iteracji*

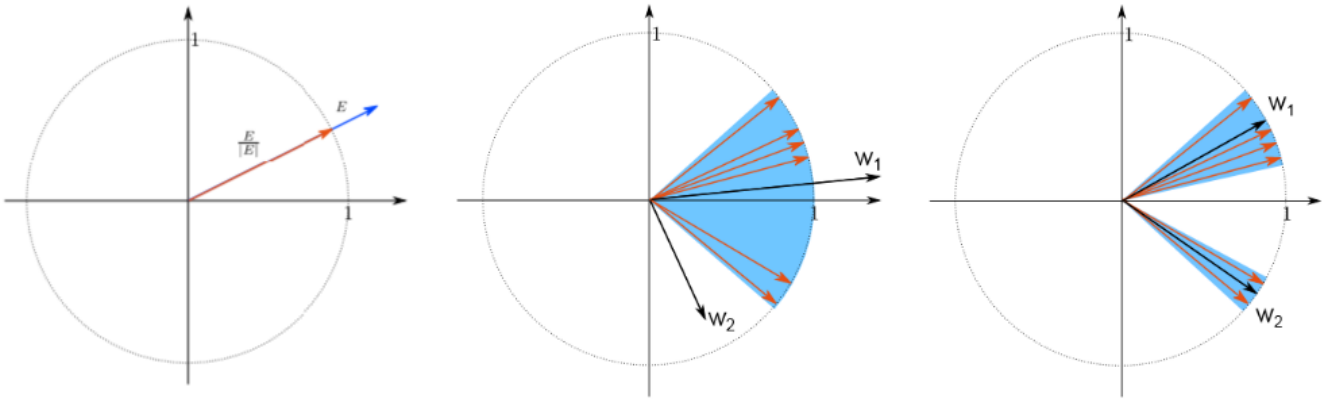
Modyfikacja wekora wag:

$$w_{m^*} = w_{m^*} + \eta x \qquad w_{m^*} = w_{m^*} + \eta(x - w_{m^*})$$



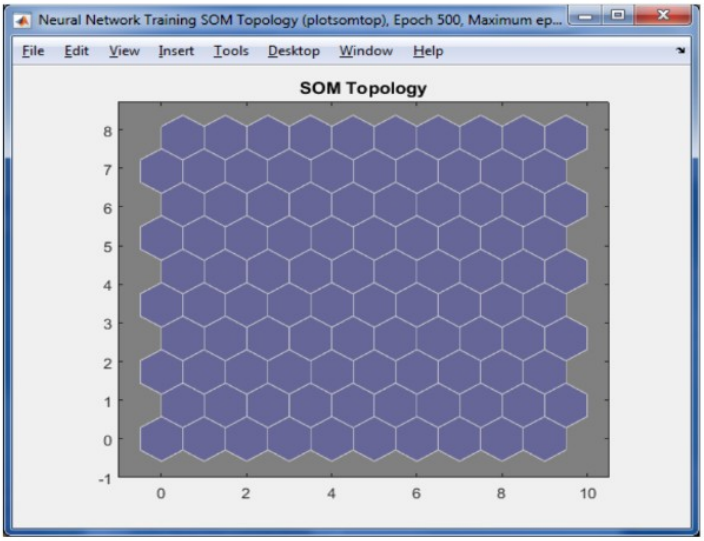


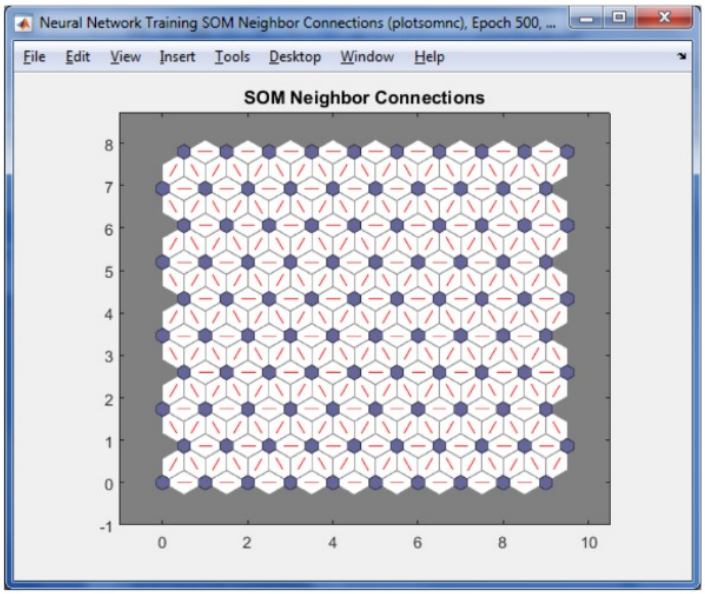
Normalizacja. Stosowanie metody WTA skutkuje podzieleniem się neuronów na tak zwane strefy wpływów. Używając normalizacji wektorów, uniemożliwiamy jednemu wektorowi stanie się tak dużym, że mógłby wygrywać współzawodnictwo zbyt często. Konsekwencją byłoby to, że inne neurony nigdy by nie zwyciężyły współzawodnictwa i ich wagi nie byłyby modyfkowane. Takie jednostki są nazywane martwymi.



Normalizacja gwarantuje, że jednostką, która jest zwycięzcą dla wektora wejściowego, jest ta jednostka, która leży najbliżej wektora wejściowego (cosinus kąta między wektorem wejściowym a zwycięzcą jest największy).

Modyfkacja wag powoduje, że wektor wag zostaje przyciągnięty w kierunku wektora wejściowego. To znaczy, że neuron zwycięski w przyszłości jeszcze silniej będzie reagował na ten wektor wejściowy (reguła Hebba).





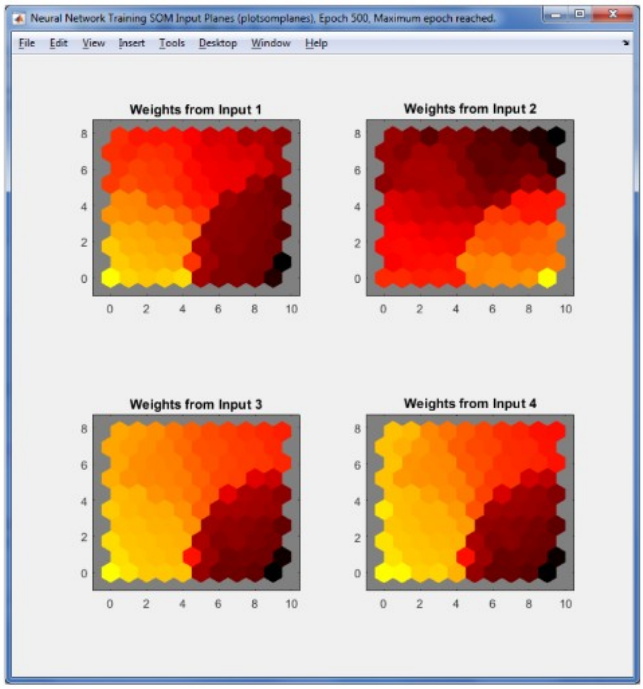
WYNIKI DZIAŁANIA PROGRAMU ORAZ WNIOSKI

Topologia sieci dla heksagonalnej mapy neuronów i połączenia między poszczególnymi neuronami.

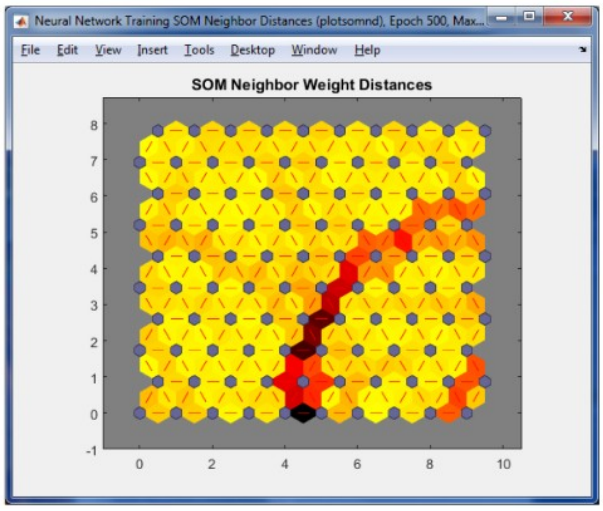
Program powinien odwzorowywać istotne cechy kwiatu irysa na podstawie otrzymanych danych.

Została użyta siatka heksagonalna która ze względu na dużą ilość połączeń umożliwia lepsze dopasowanie wag.

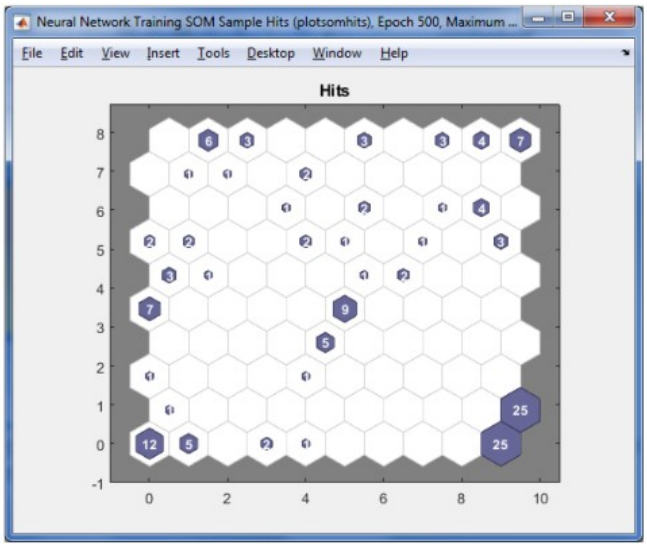
Wejścia to: (1) długość i (2) szerokość płatka, (3) długość i (4) szerokość kielicha. Tutaj można zaobserwować tworzenie się strefy wpływów. Ciemniejszy kolor oznacza wyższe wagi czyli wagi bardziej typowe dla irysa.

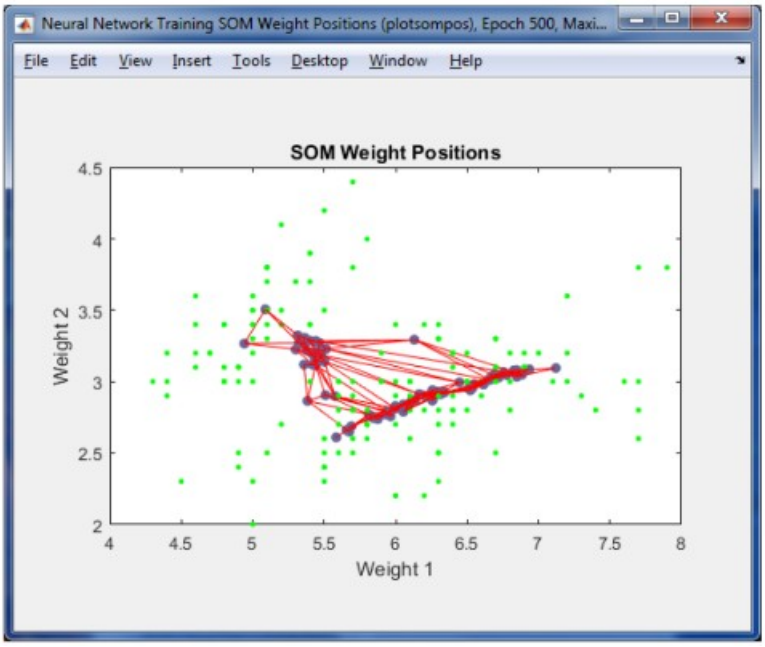


Odległości pomiędzy wagami:



Winner Takes All – zwycięzcy. Efekt działania normalizacji. Odległość neuronów wag są powiązane z tym ile razy dany neuron wygrał rywalizację.





Sieć w poprawny sposób wyznaczyła typowe cechy irysa. Zielone kropki przedstawiają typowe cechy irysa, niebieskie to te kwiaty, których cechy najbardziej pasowały do cech irysa.

LISTING KODU MATLAB

```
close all; clear all; clc;

WE = iris_dataset;
size(WE);
plot(WE(1,:),WE(2,:),'b.',WE(3,:),WE(4,:),'g.');%wagi
hold on; grid on;%wykresy w bieżących osiach; wyświetlanie głównych linii siatki
%parametry sieci
dimensions   = [10 10];
coverSteps   = 100;
initNeighbor = 0;
%topologie:
topologyFcn  = 'hextop';
distanceFcn  = 'dist';
%tworzenie SOM (self organisation-map):
net = selforgmap(dimensions,coverSteps,initNeighbor,topologyFcn,distanceFcn);
net.trainParam.epochs = 500;%liczba epok
%liczba epok:
%mniejsza = mniejsza skuteczność
%większa = dłuższy czas treningu, bez lepszych efektów
%trenowanie sieci:
[net,tr] = train(net,WE);
y = net(WE);
grid on
```