Michalina Nikiel
WIMiIP, Inżynieria Obliczeniowa
rok 3, grupa laboratoryjna nr 2

Podstawy sztucznej inteligencji – projekt nr 5 – sprawozdanie

Budowa i działanie sieci Kohonena dla WTA.

Cel ćwiczenia:

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

Zrealizowane kroki:

- Przygotowanie danych uczących zawierających numeryczny opis cech kwiatów.
- Przygotowanie sieci Kohonena i algorytmu uczenia opartego o regułę Winner Takes All (WTA).
- Uczenie sieci.
- Testowanie sieci.

Sprawozdanie zawiera kolejno:

- Zagadnienia teoretyczne: opis budowy użytej sieci i odpowiednich algorytmów.
- Zestawienie otrzymanych wyników wraz z zrzutami ekranu i przebiegiem ćwiczenia.
- Wnioski wraz z analiza i dyskusją błędów uczenia opracowanej sieci.
- Listing całego kodu programu.

Zagadnienia teoretyczne:

Działanie sieci Kohonena:
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 (SOM) lub Competitive Filters.

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.

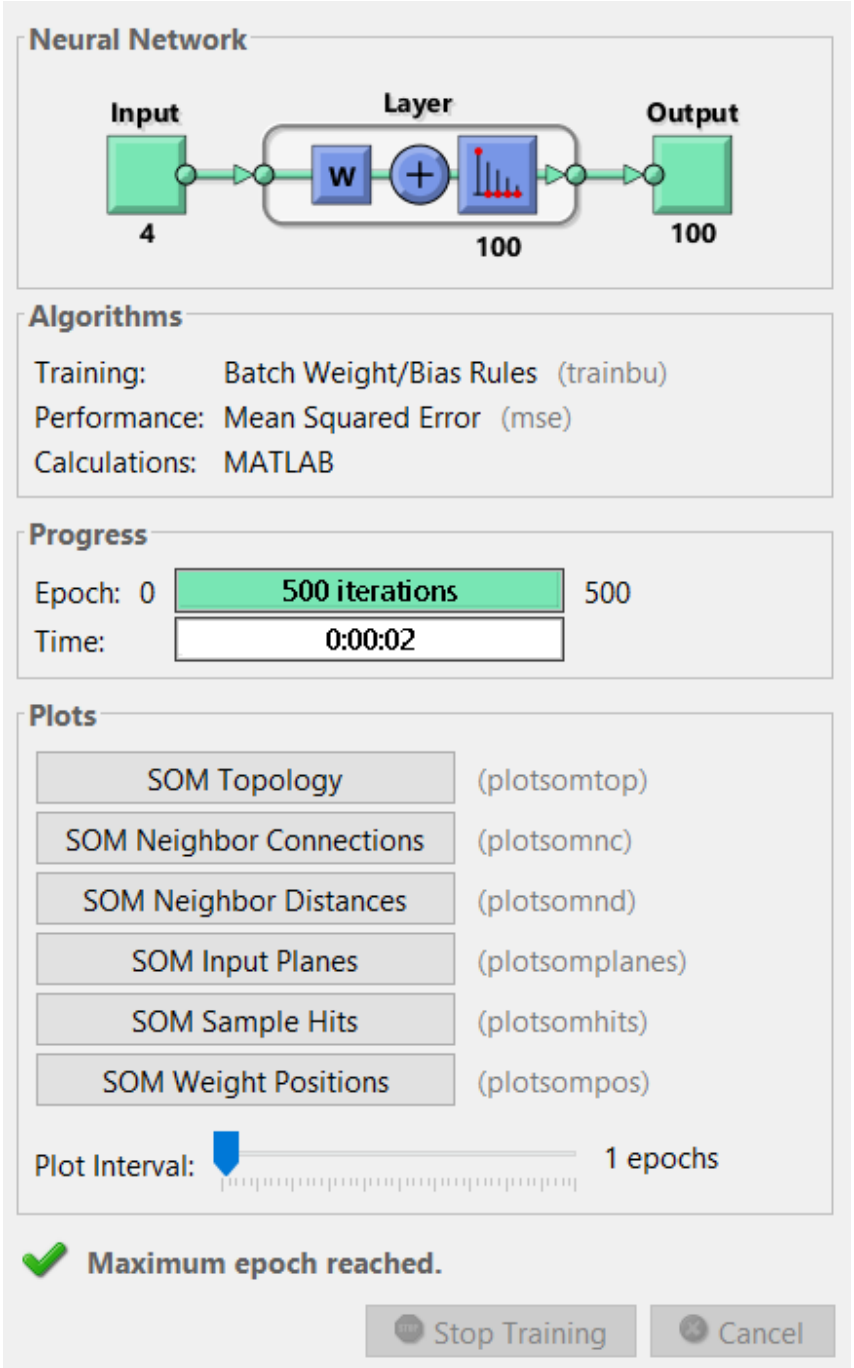
Mechanizm WTA (Winner Takes All):
Jest to reguła aktywacji neuronów w sieci neuronowej, która pozwala na aktywację tylko jednego neuronu w danej chwili. W konsekwencji w jednym momencie zostaje zmodyfikowana waga tylko jednego neuronu. Aby uniknąć dominacji jednego neuronu często stosuje się mechanizm zmęczenia, który polega na tym, że jeśli jakiś neuron wygrywa zbyt często, to przez pewien czas przestaje być brany pod uwagę podczas rywalizacji (jest odsuwany/usypiany w celu wylosowania innego neuronu).
Stosowanie tego mechanizmu prowadzi do podzielenia mapy neuronów na tzw. „strefy wpływów”. Są to obszary, które znajdują się pod dominacją konkretnego silnego neuronu, który uniemożliwia modyfikację wag neuronów z jego otoczenia (brak normalizacji może doprowadzić do sytuacji, w której na niewielkim obszarze znajduje się kilka silnych neuronów lub kilka niewielkich stref wpływów, natomiast pozostały obszar nie posiada żadnego silnego neuronu - nierównomierny rozkład sił).

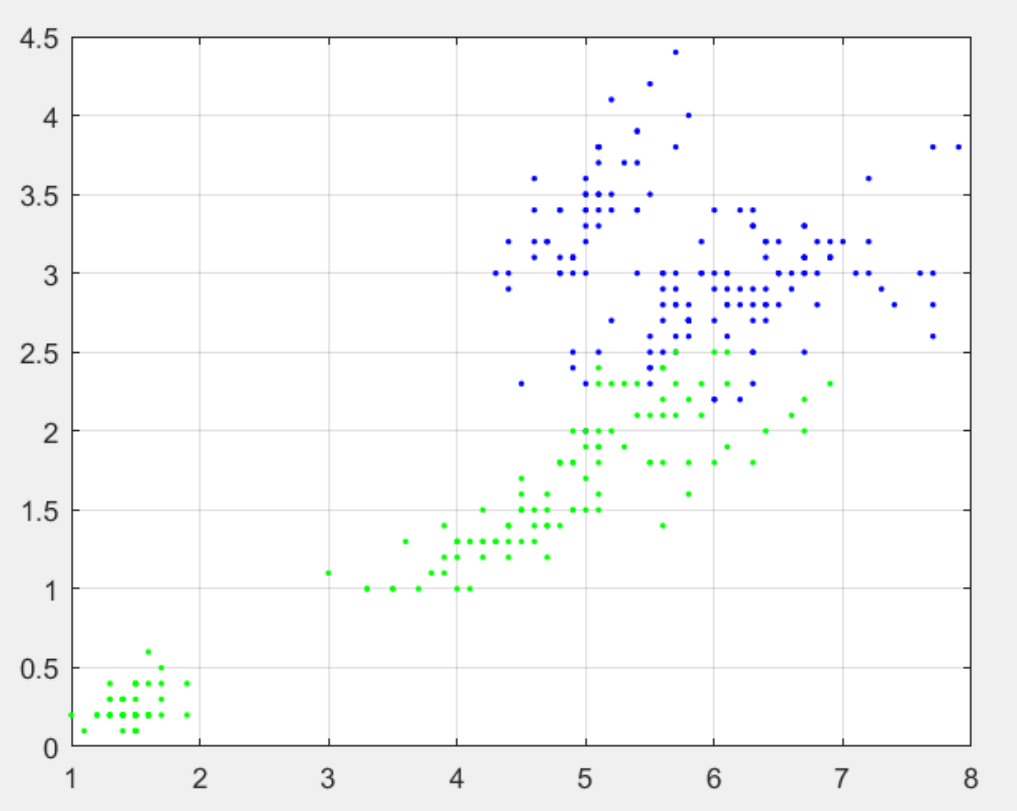
Użyty algorytm uczenia sieci:
1) Generowanie losowo znormalizowanych wektorów wag.
2) Losowanie wektora X oraz obliczanie dla niego aktywację Y dla wszystkich neuronów.
3) Szukanie neuronu zwycięzcy.
4) Modyfikacja wektora wag neuronu zwycięzcy, a następnie jego normalizacja (sprawdzenie warunków WTA).
5) Zatrzymanie algorytmu po odpowiednio dużej ilości iteracji.

Zestawienie otrzymanych wyników:

W programie wykorzystałam heksagonalną siatkę neuronów, uczenie według reguły Kohonena i WTA. Program w efekcie powinien odwzorowywać istotne cechy kwiatów irysa na podstawie otrzymanych danych.

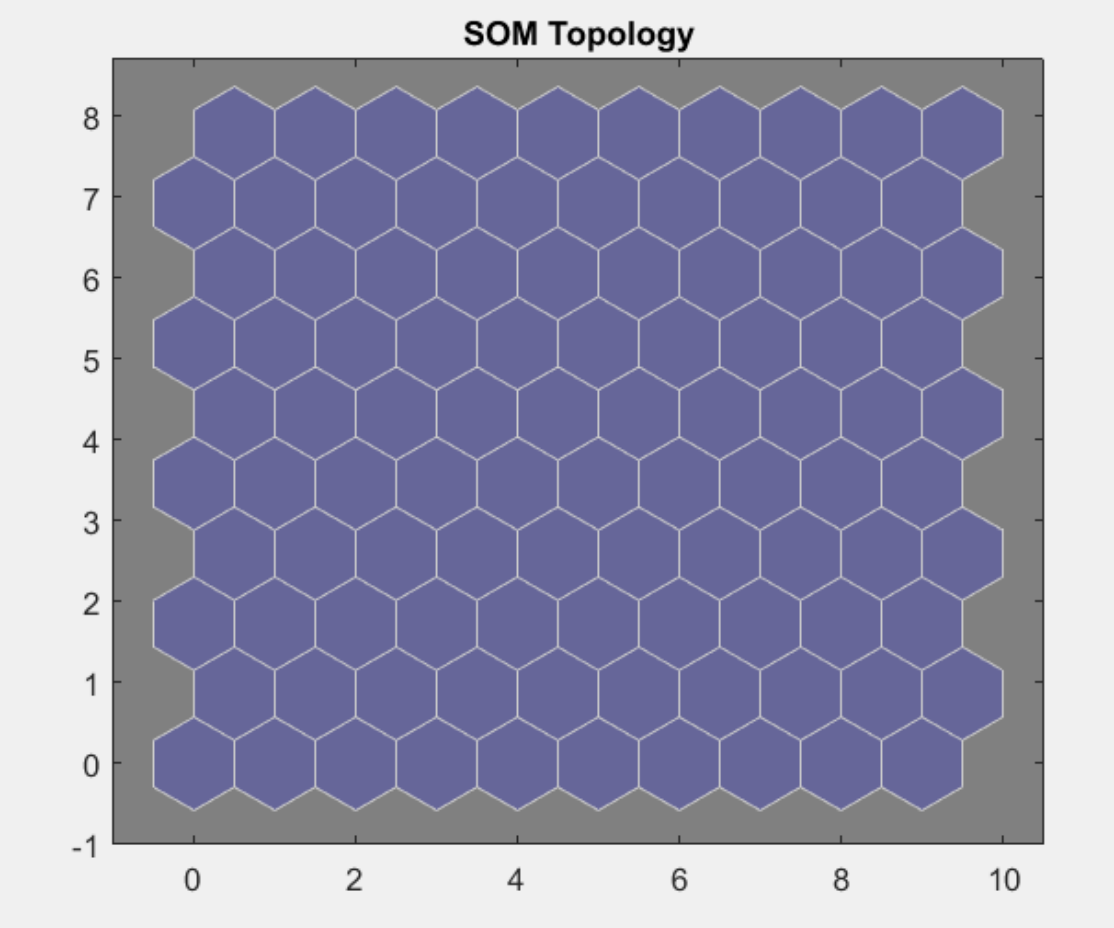
Szkolenie obejmuje maksymalną liczbę epok, która wynosi 500, czas wykonania zadania to 0:00:02.



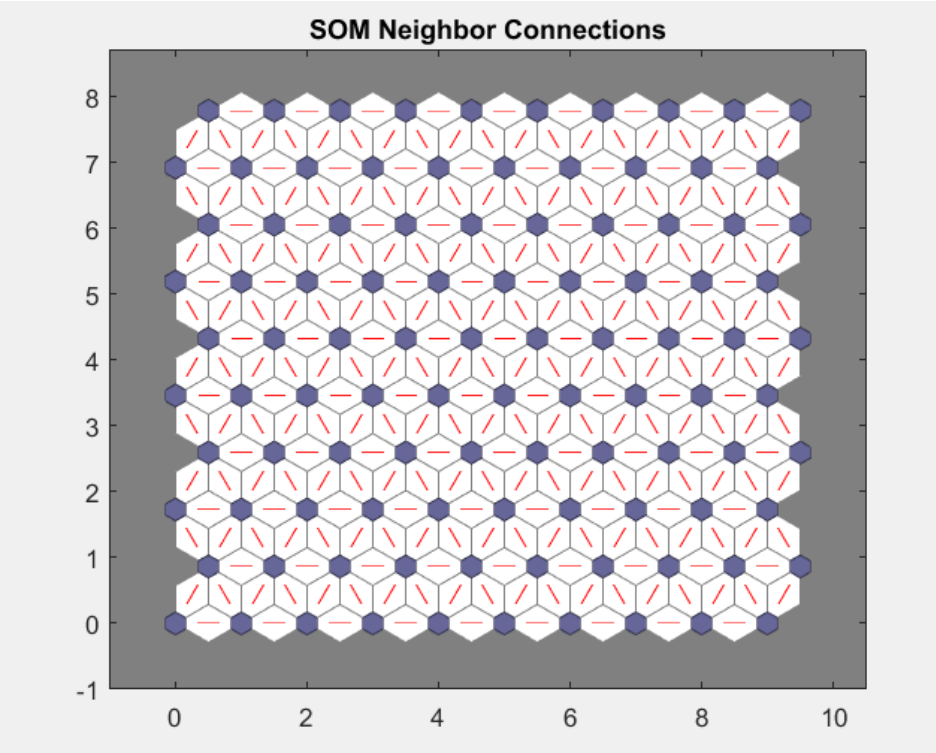


SOM Topology:

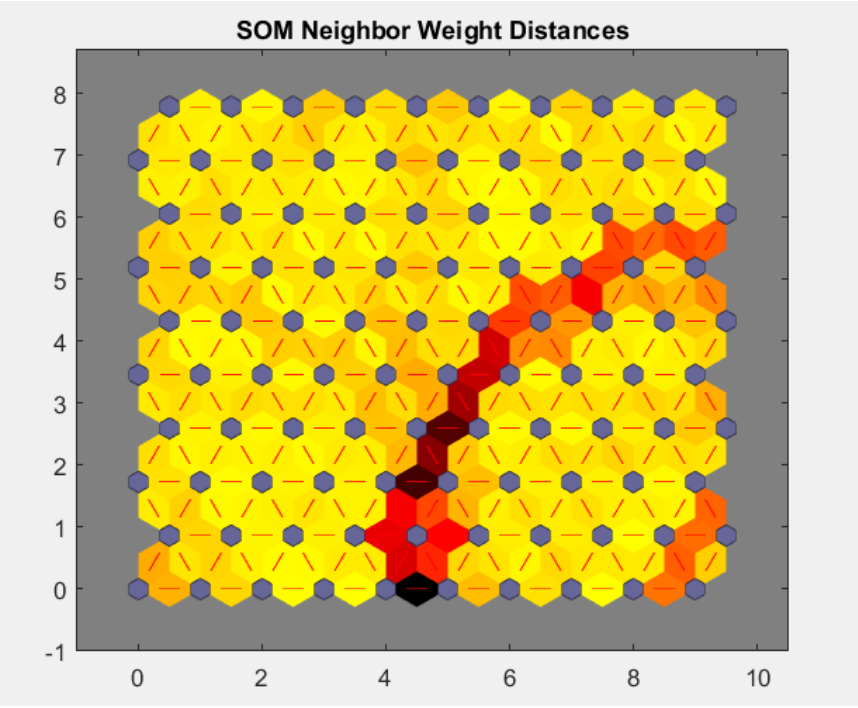
Na tej figurze każdy z sześciokątów reprezentuje neuron. Siatka to 8 na 10, więc w tej sieci jest łącznie 80 neuronów. W każdym wektorze wejściowym znajdują się cztery elementy, więc przestrzeń wejściowa jest czterowymiarowa. Wektory masy (centra skupień) mieszczą się w tej przestrzeni.



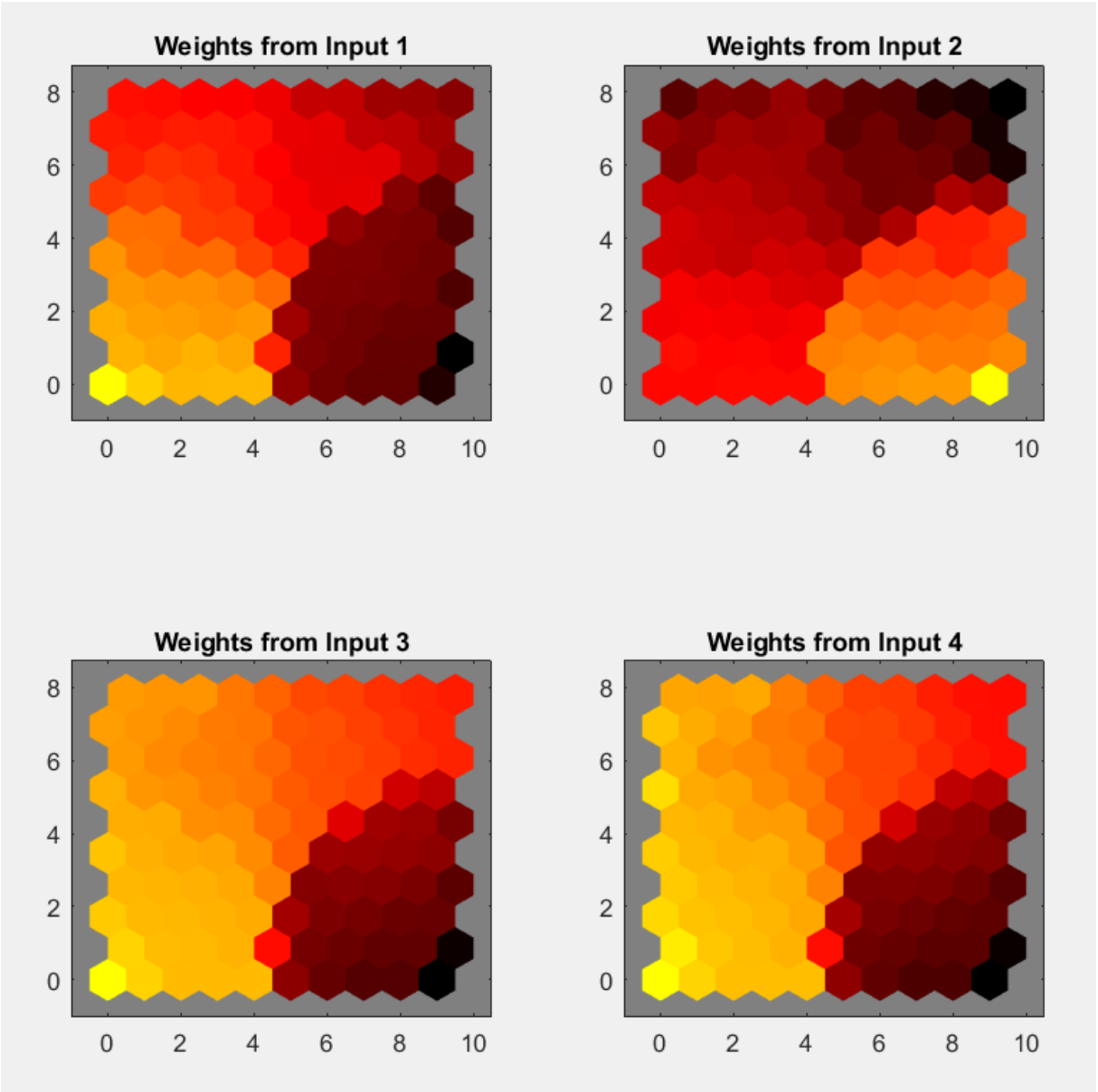
SOM Neighbor Connections:
Niżej umieszczona mapa przedstawia powiązanie sąsiedzkie.



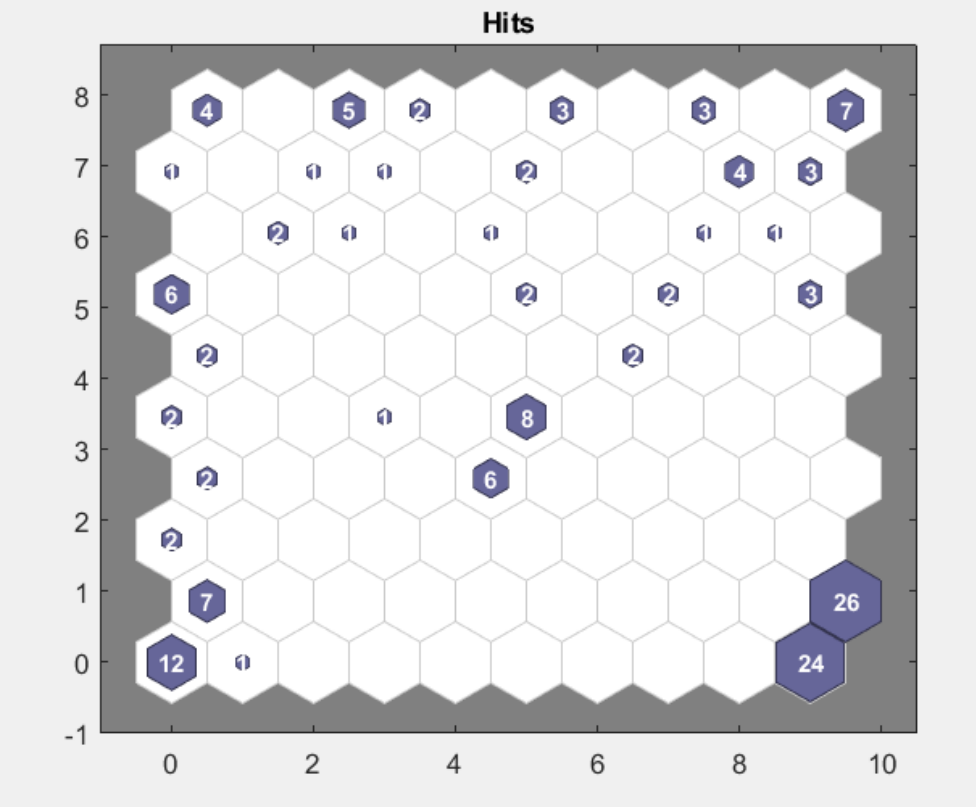
SOM Neighbor Distance:
Na tej figurze niebieskie sześciokąty reprezentują neurony. Czerwone linie łączą sąsiednie neurony. Kolory w obszarach zawierających czerwone linie wskazują odległości między neuronami. Ciemniejsze kolory reprezentują większe odległości, a jaśniejsze kolory oznaczają mniejsze odległości. Pasmo ciemnych segmentów przechodzi od górnego w dół.



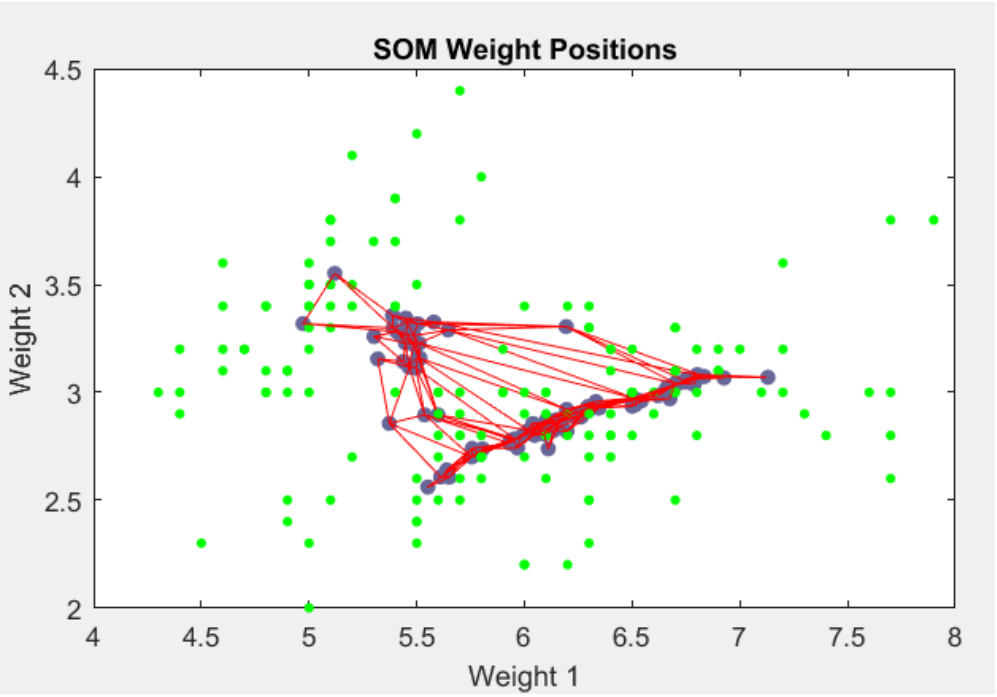
SOM Input Planes:
Ta figura przedstawia płaszczyznę ciężkości dla każdego elementu wektora wejściowego (w tym przypadku cztery). Są to wizualizacje wag, które łączą każde wejście z każdym z neuronów. (Ciemniejsze kolory oznaczają większe wagi.) Jeśli schematy połączeń dwóch wejść były bardzo podobne, można założyć, że dane wejściowe są wysoce skorelowane.



SOM Sample Hits:
W przypadku SOM training wektor wagowy związany z każdym neuronem przesuwa się, aby stać się centrum skupienia wektorów wejściowych. Ponadto, neurony, które sąsiadują ze sobą w topologii, powinny również poruszać się blisko siebie w przestrzeni wejściowej, dlatego możliwe jest zwizualizowanie wielowymiarowej przestrzeni wejściowej w dwóch wymiarach topologii sieci. Zdjęcie to przedstawia ile razy dany neuron zwyciężył podczas rywalizacji.



SOM Weight Positions:



Opis użytych funkcji:
WEJSCIE = iris_dataset – dane uczące, wartości implementowane przez oprogramowanie, zawiera numeryczny zapis czterech cech kwiatu irysa umieszczonych w tablicy 4x150,
– **size(WEJSCIE)** – określenie rozmiaru poprzez przypisanie danych wejściowych,
– **hold on** – utrzymuje wykresy w bieżących osiach, aby nowe wykresy dodane do osi nie usuwały istniejących wykresów,
– **grid on** – wyświetla główne linie siatki dla bieżących osi lub wykresu. Główne linie siatki rozciągają się od każdego znaku podziałki,
– **dimensions** – wektor rzędów wymiarów,
– **coverSteps** – liczba kroków szkoleniowych dla początkowego pokrycia przestrzeni wejściowej (domyślnie = 100),
– **initNeighbor** – początkowy rozmiar sąsiedztwa (domyślnie = 3),
– **topologyFcn** – funkcja topologii warstw (domyślnie = 'hextop'),
– **hextop** – funkcja topologii sześciokątnej,
– **distanseFcn** – funkcja odległości neuronowej (domyślnie = "linkdist"),
– **dist** – funkcja wagi odległości euklidesowej,
– **net = selforgmap(…)** – zmienna, do której będzie przypisywana nowo tworzona sieć neuronowa za pomocą algorytmu Kohonena z wykorzystaniem wcześniej zdefiniowanych parametrów sieci,
– **net.trainParam.epochs** – ustalenie maksymalnej liczby epok treningowych utworzonej sieci

Wnioski:

- Heksagonalna siatka neuronów umożliwia utworzonej sieci stworzenie większej liczby połączeń pomiędzy neuronami, niż w przypadku sieci prostokątnej (w konsekwencji sieć ma więcej możliwości w doborze odpowiednich wag dla poszczególnych neuronów).
- Na podstawie ostatniego wykresu można zauważyć, że sieć neuronowa zdefiniowała te punkty jako irysy, które posiadały parametry bliskie lub równe średnim wartościom. Średnie wartości były interpretowane jako typowe wartości dla irysów.
- Wykres przedstawiający rozkład wag kolejnych wejść doskonale obrazuje wcześniej opisane zjawisko strefy wpływów. Analiza rozkładu barw pozwala określić jak może wyglądać irys według szkolonej sieci neuronowej. Dla przypomnienia im kolor jest ciemniejszy, tym jest bardziej bliski cechy irysa (według wyznaczonych danych przez sieć można stwierdzić, że zdefiniowany kwiat posiada zarówno długie i szerokie płatki jak i kielich, co pokrywa się z rzeczywistym wyglądem kwiatu).
- Na podstawie wykresu pokazującego rozkład sił neuronów można zauważyć, że sieć korzystała z mechanizmu WTA (sieć nie modyfikowała

wag tylko jednego neuronu, wynika to z zastosowania normalizacji, gdyby nie ona zapewne dwa najmocniejsze neurony w prawym dolnym rogu zwyciężałyby za każdym razem).

- Zestawiając wyniki można zauważyć, że wykres przedstawiający rozkład sił neuronów jest powiązany z wykresem pokazującym odległości pomiędzy wagami poszczególnych neuronów. Ciemniejsze kolory na wykresie drugim pokrywają się z występowaniem neuronów zwyciężających w wykresie pierwszym.
- Pomimo treningu bez nadzoru sieć Kohonena (wraz z mechanizmem WTA) prawidłowo odwzorowała cechy typowe dla wybranego kwiatu, przy stosunkowo niewielkiej liczbie narzuconych epok treningowych (w moim przypadku już przy zmniejszeniu liczby do 500 epok wyniki stawały się coraz bardziej klarowne, przy odpowiednio krótkim czasie działania programu - więcej epok zapewniało na pewno większą dokładność, jednak zdecydowanie wydłużało czas nauki).
- Bardzo ważnym czynnikiem przy tworzeniu takiej sieci jest dobór odpowiedniej liczby neuronów - dla małej liczby rośnie ryzyko wystąpienia błędu, natomiast zbyt duża liczba neuronów znacznie wydłuży czas potrzebny na naukę sieci.

Listing kodu:

```
WEJSCIE = iris_dataset;      %dane wejsciowe
size(WEJSCIE);               %okreslenie rozmiaru tablicy
plot(WEJSCIE(1, :), WEJSCIE(2, :), 'b.', WEJSCIE(3, :),
WEJSCIE(4, :), 'g.');
hold on; grid on;            %do wyświetlenia (trzymanie
wykresu i siatka)

% PARAMETRY SIECI KOHONENA
dimensions   = [10 10];      %wymiar wektora
coverSteps   = 100;          %liczba kroków szkoleniowych
dla początkowego pokrycia przestrzeni wejściowej
initNeighbor = 0;            %początkowy rozmiar sąsiedztwa
topologyFcn  = 'hextop';     %funkcja topologii warstw
distanceFcn  = 'dist';       %funkcja odległości neuronowej

% TWORZENIE SIECI KOHONENA
net = selforgmap(dimensions, coverSteps, initNeighbor,
topologyFcn, distanceFcn);
net.trainParam.epochs = 500;%ustalenie maksymalnej liczby
epok treningowych utworzonej sieci

% TRENING SIECI
[net, tr] = train(net, WEJSCIE);
y = net(WEJSCIE);

grid on
```