Temat: ***Budowa​ ​i​ ​działanie​ ​sieci​ ​Kohonena​ ​dla​ ​WTM.***

Data oddania: 11.12.2017

Prowadzący: dr inż. Dorota Wilk - Kołodziejczyk

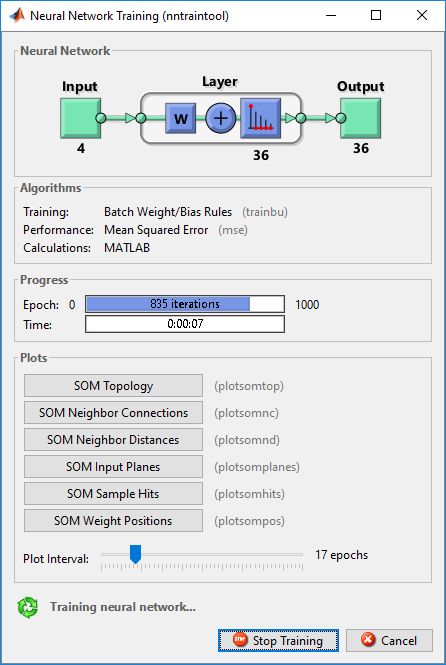
Przedmiot: Podstawy sztucznej inteligencji

Autor: Przemysław Adamus 286084

Kierunek: Inżynieria Obliczeniowa sem.V

Spis treści:

1. Cel projektu;
2. Wstęp i opis projektu;
3. Budowa sieci;
4. Zestaw uczący;
5. Syntetyczny​ ​opis​ ​budowy​​ ​wykorzystanego​ ​algorytmu​ ​uczenia w projekcie;
6. Opis wykorzystanych funkcji;
7. Wyniki i analiza działania programu;
8. Wnioski;
9. Źródła;
10. Kod programu.
11. **CEL:**

Celem ćwiczenia było​ poznanie​ ​budowy​ ​i​ ​działania​ ​sieci​ ​Kohonena​ ​przy​ ​wykorzystaniu​ ​reguły WTM​ ​do​ ​odwzorowywania​ ​istotnych​ ​cech​ dużych liter alfabetu.

1. **WSTĘP I KRÓTKI OPIS PROJEKTU:**

Do rozwiązania zadanego problemu wykorzystałem wiedzę i kody źródłowe z wcześniejszych projektów. Wiedzę na temat zadanego problemu pozyskałem z wielu stron WWW i internetowych tutorialii (linki w bibliografii), oraz z wykładów.

Do rozwiązania problemu użyłem środowiska **MatLab R2016a**. Do realizacji scenariusza wykorzystałem bibliotekę do tworzenia sztucznych sieci neuronowych „**Neural Network Toolbox**” oraz zawarte w niej gotowe funkcje. Celem ćwiczenia było stworzenie SOM czyli samoorganizującej się mapy Kohonena i użycie jej do grupowania gatunków kwiatów z odpowiednich danych wejściowych. O danych wejściowych możemy przeczytać w punkcie 3.

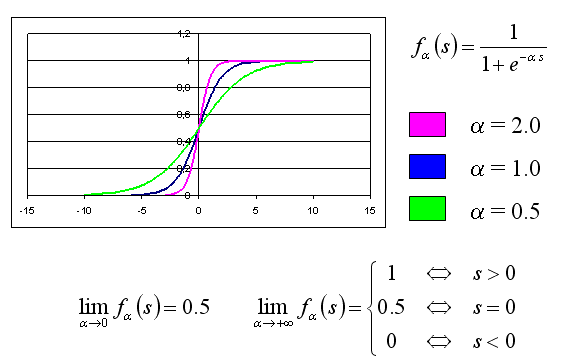
1. **PODSTAWOWE POJĘCIA I BUDOWA WYKORZYSTANEJ SIECI:**

Perceptronem nazywamy prosty element obliczeniowy, który sumuje ważone sygnały wejściowe i porównuję tę sumę z progiem aktywacji - w zależności od wyniku perceptron może być albo wzbudzony (wynik 1), albo nie (wynik 0).

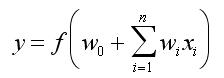
Suma membranowa jest to suma wszystkich iloczynów wejść i ich wag. Jest ona przekazywana dalej do funkcji aktywacji.

Funkcja aktywacji w sztucznej inteligencji określa funkcje, według której obliczana jest wartość wyjścia neuronów sieci neuronowej. Do najczęściej używanych funkcji aktywacji należą: funkcja liniowa, funkcja progowa unipolarna, funkcja progowa bipolarna, funkcja sigmoidalna unipolarna, funkcja sigmoidalna bipolarna.

Rezultat wiadomość zwrotna od neuronu. Może być przekazana dalej, lub być ostatecznym wynikiem. Rezultat jest wartością logiczną więc powinien zawierać tylko informacjię *true* lub *false.*



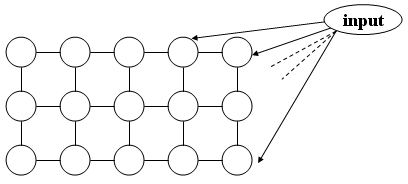
Ogólny wzór na wartość wyjściową neuronu przedstawia się następująco:



gdzie *w0* jest dodatkową wagą pełniącą rolę analogiczną do progu aktywacji w perceptronie.

**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):



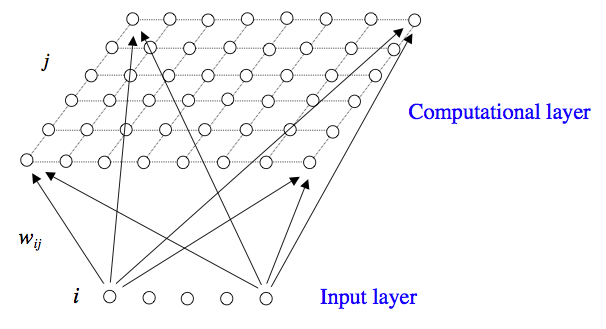
**Zasady działania sieci Kohonena:**

* 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.

**Uczenie** jest po prostu mechanizmem który dla każdego neuronu określa stopień podobieństwa jego wag do danego sygnału wejściowego oraz wyznacza jednostkę największym dopasowaniem – czyli zwycięzcę. Wtedy sieć powinna zadaptować wartości wag neuronu zwycięzcy i jego sąsiadów w zależności od siły, z jaką odpowiedział on na dane wejście. SOM jest siecią uczącą się bez nadzoru. Oznacza to, że w trakcie uczenia opierają się wyłącznie na obserwacji danych wejściowych, nikt im nie mówi, co z tych danych wejściowych powinno wynikać, sama ma to ustalić. Zatem struktura sieci SOM wygląda tak:

* jedna warstwa wejściowa – 1w
* co najmniej warstwa wyjściowa -2w
* mapa topologiczna
* sieć jest zwykle jednokierunkowa
* każdy neuron jest połączony ze wszystkimi składowymi N-wymiarowego wektora wejściowego x.
* wagi połączeń neuronów tworzą wektor **wi=[wi1, wi2, ..., win]T**



Dokładniejsze i więcej informacji o wykorzystywanych algorytmach przeczytamy w punkcie 4.

1. **ZESTAW UCZĄCY:**

W moim projekcie litery reprezentowane są na matrycach o wymiarach 6x5. Zgodnie z treścią zadania jest ich 20 sztuk. Zdecydowałem się na 10 łatwych do reprezentacji, początkowych liczb naszego alfabetu. Zapisałem je do tablicy dwu wymiarowej „dataIN [20][30]” w kolejności alfabetycznej „A B C D E F G H I J K L M N O P R S T U”.

%A B C D E F G H I J K L M N O P R S T U

dataIN =[0 1 0 1 1 1 0 1 0 1 1 0 1 1 0 1 1 0 1 1;

1 1 1 1 1 1 1 0 1 0 0 1 0 0 1 1 1 1 1 0;

1 1 1 1 1 1 1 0 1 0 0 1 0 0 1 1 1 1 1 0;

1 1 1 1 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0;

0 0 1 0 1 1 0 1 0 1 0 1 1 1 0 0 0 0 1 1;

1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1;

0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0;

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0;

0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0;

1 1 0 1 0 0 1 1 0 0 0 1 1 1 1 1 1 0 0 1;

1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 0 1;

0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0;

0 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 0 1 0;

0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0;

1 0 0 1 0 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1;

1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 0 1;

1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0;

1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 1 1 0;

1 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 1 0 0 0;

1 1 0 1 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1;

1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1;

0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0;

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0;

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0;

1 1 0 1 0 0 1 1 0 1 0 1 1 1 1 0 1 0 0 0;

1 1 0 1 1 1 0 1 0 1 1 0 1 1 0 1 1 0 0 0;

0 1 1 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1;

0 1 1 1 1 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1;

0 1 1 1 1 1 1 0 1 0 0 1 0 0 1 0 0 0 0 1;

1 0 1 0 1 0 0 1 0 1 1 0 1 1 0 0 1 0 0 0;];

1. **ALGORYTM UCZENIA - ALGORYTMY WTA & WTM:**

W trakcie pokazywania elementów zbioru uczącego każdorazowo wyłaniany jest neuron „zwycięski”. Najczęściej właśnie ten „zwycięzca” jest uczony. Zwycięski jest ten neuron, którego wektor wag jest najbliższy w stosunku do aktualnego wektora wejściowego. Na skutek uczenia zwycięzcy neuron który „lubił” określone sygnały wejściowe (bo przy ich pojawianiu się dawał silny pozytywny sygnał wyjściowy), po każdym kroku procesu uczenia będzie „lubił” te sygnały coraz bardziej. Taki neuron po wytrenowaniu będzie więc rozpoznawał pewną klasę sygnałów. Ponieważ tylko on będzie rozpoznawał te klasę, będzie, wiec mógł służyć jako swoisty detektor wszelkich sygnałów podobnych do tych, jakie w trakcie uczenia powodowały, że był on zwycięzcą.

Metodyka zmian wag neuronów wyjściowych nazywa się strategią. Dla SOM dwie podstawowe strategie to:

* **WTA** - Winner Takes All - Zwycięzca bierze wszystko. Po przedstawieniu sieci wektora wejściowego, neuron najbardziej podobny do elementu prezentowanego (którego wagi są najbardziej podobne składowym wektora wejściowego) zostaje zmodyfikowany zgodnie z funkcją f tak aby jego wagi były jak najbardziej zbliżone do wektora wejściowego.
* **WTM** - Winner Takes Most - Zwycięzca bierze najwięcej. W tej strategii nie tylko neuron najbardziej podobny, ale także jego otoczenie zostają zmodyfikowane. Najczęściej ta modyfikacja jest zależna od odległości sąsiada od zwycięzcy.

W naszym projekcie skorzystaliśmy z algorytmu **WTM**:

1. Przypisz wagom sieci o M neuronach warstwy wyjściowej i N wejściach niewielkie liczby losowe. Ustal liczbę neuronów należących do początkowego otoczenia neuronu.

2. Dołącz nowy wektor uczący x=[x1 , x2 , ..., xN ] do wejścia.

3. Oblicz odległości.

4. Znajdź neuron zwycięski. Mając próbkę x ze zbioru uczącego X w danym kroku k fazy uczenia znajdujemy element mapy najbardziej zbliżony do wektora prezentowanego c(x): **c(x) = arg min{||x-mi||}.**

5. Wyznacz nowe wartości wag dla neuronu zwycięzcy i jego sąsiedztwa korzystając z jednej z dwóch strategii. W naszym przypadku **WTM**: **wi(k+1)=wi(k)+η(k)G(i,c) x(k)−wi(k).** Funkcja G oznacza tu wpływ sąsiedztwa. Algorytm działa w sposób analogiczny jak WTA.

6. Zmień odpowiednio wartości współczynnika uczenia i sąsiedztwa.

7. Znormalizuj wagi.

8. Powtórz 2-7 dla następnych wzorców wejściowych aż do chwili ustalenia się odpowiedzi sieci.

1. **OPIS WYKORZYSTANYCH FUNKCJI:**

Do realizacji scenariusza wykorzystałem bibliotekę do tworzenia sztucznych sieci neuronowych „**Neural Network Toolbox**” oraz zawarte w niej gotowe funkcje. Na temat podstawowych funkcji pisałem już w wcześniejszym sprawozdaniu. Teraz opiszę wykorzystanie funkcji „selforgmap”.

Funkcjia **selforgmap()** pozwala na stworzenie mapy neuronów z odpowiednimi parametrami. Jako argumenty przyjmuje:

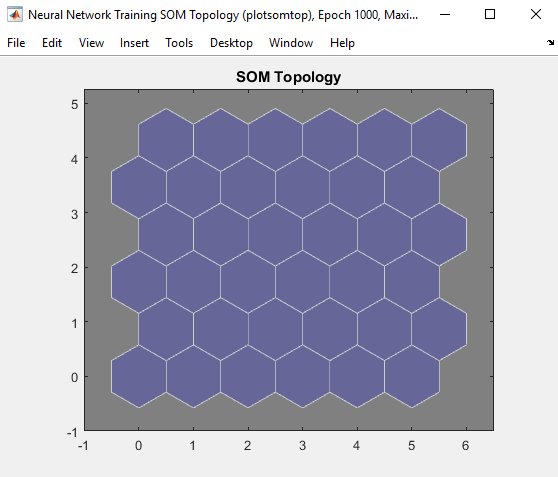
• Dimensions – wymiar sieci (np. 6x6);

• InitNeighbor – ilość neuronów tworzących sąsiedztwo;



• topologyFcn – funkcja topologii:

1. Gridtop – topologia przy użyciu prostokątów;
2. Hextop – przy użyciu sześciokątów;
3. Randtop – losowo rozrzucone neurony;



• distanceFcn – funkcja obliczająca dystans pomiędzy neuronami:

a) dist - odległość Euklidesowa.

b) linkdist – kilka sposobów liczenia, w tym Euklidesowa, zależnych od

ilości wprowadzanych wektorów.

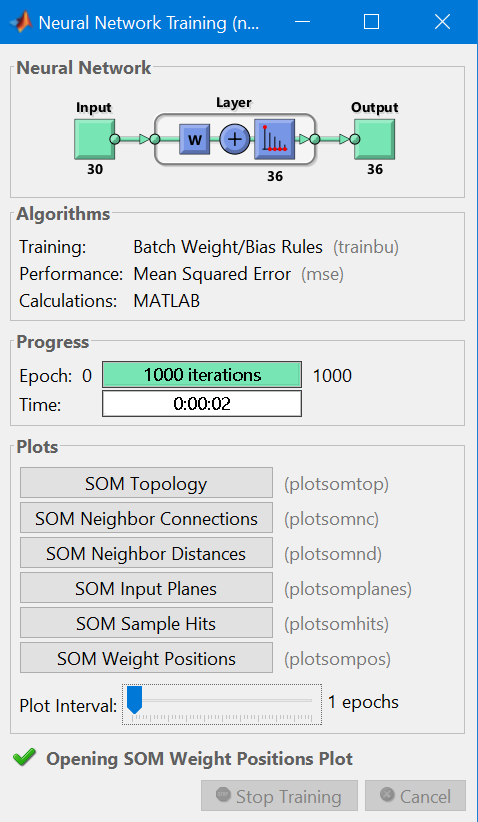
Użyte parametry w mojej sieci:

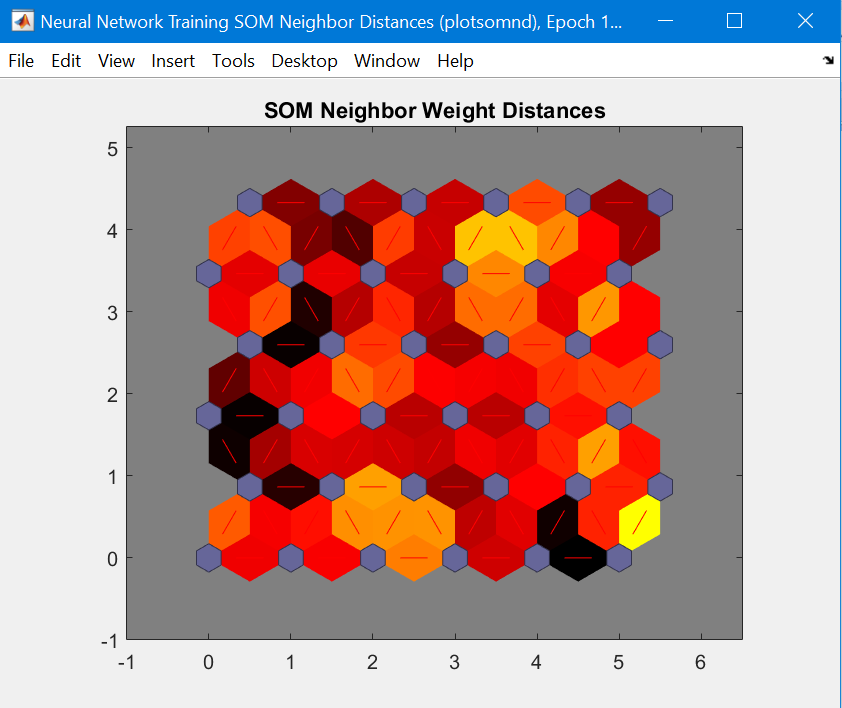
* dimensions = [6 6]; lub dimensions = [6];
* coverSteps = 100;
* initNeighbor = 1;
* topologyFcn = 'hextop';
* distanceFcn = 'linkdist';

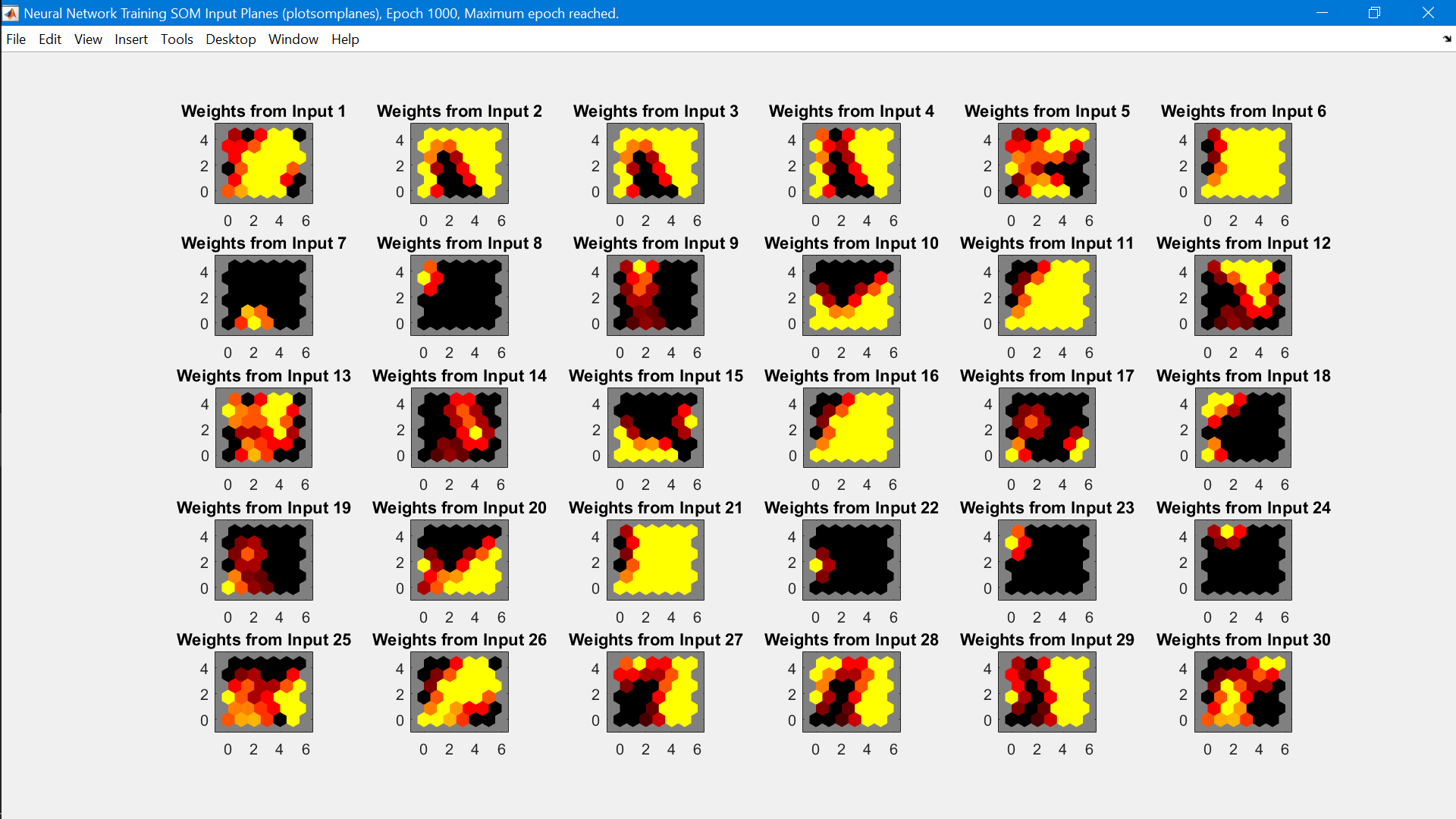
Inicjalizacja:

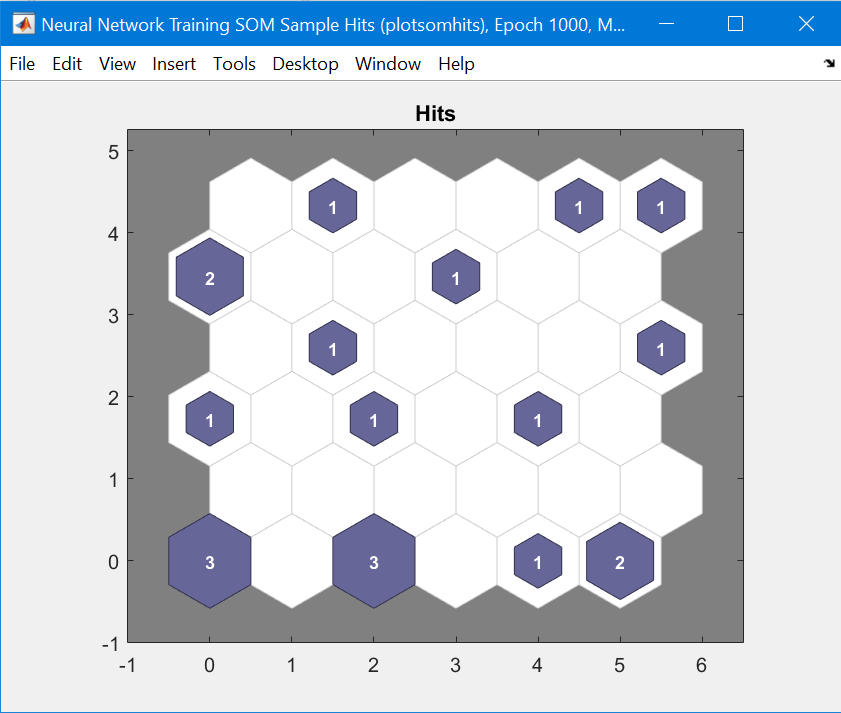
net = **selforgmap**(*dimensions,coverSteps,initNeighbor,topologyFcn,distanceFcn*);

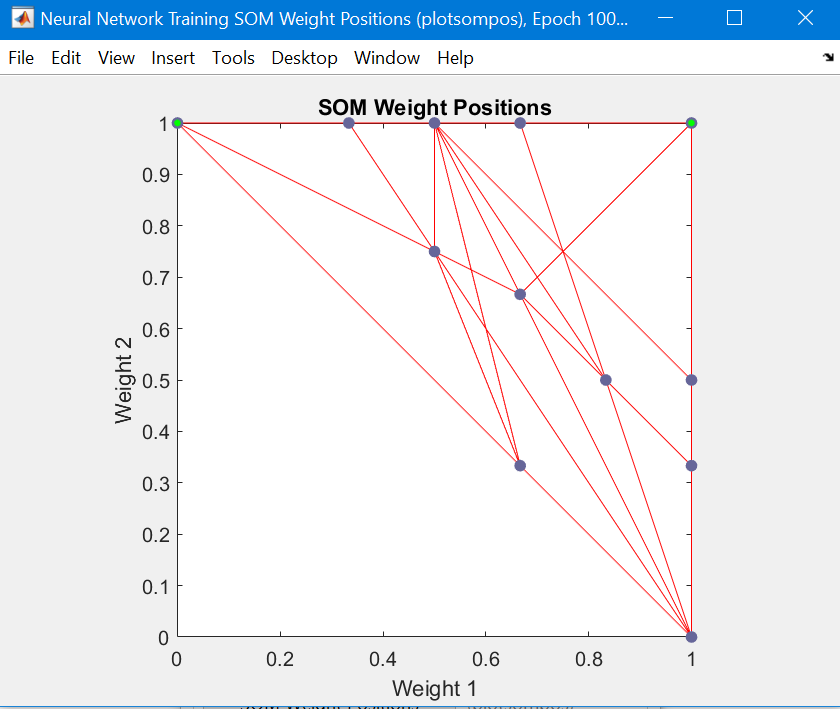
1. **WYNIKI I ANALIZA DZIAŁANIA PROGRAMU:**

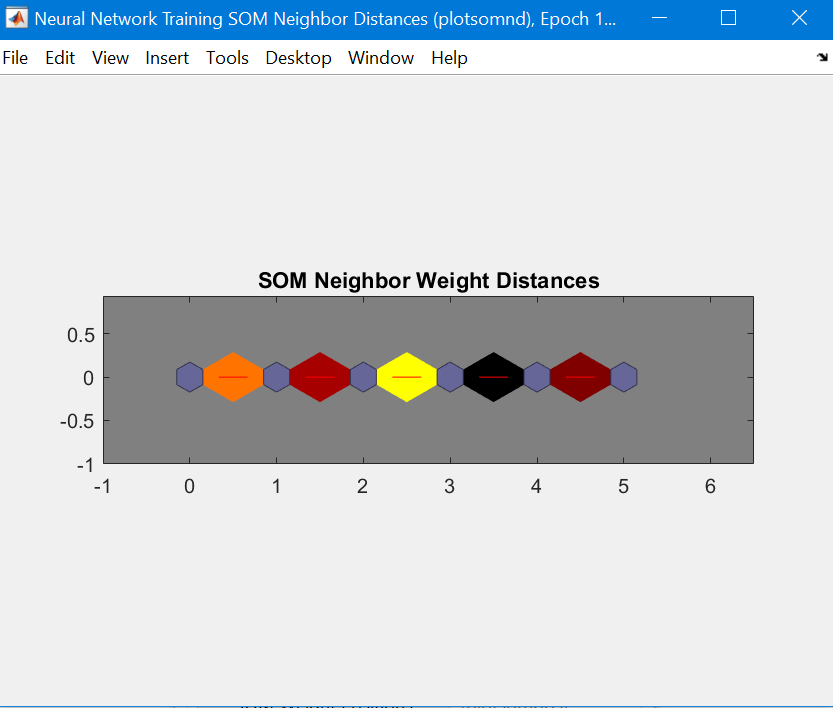


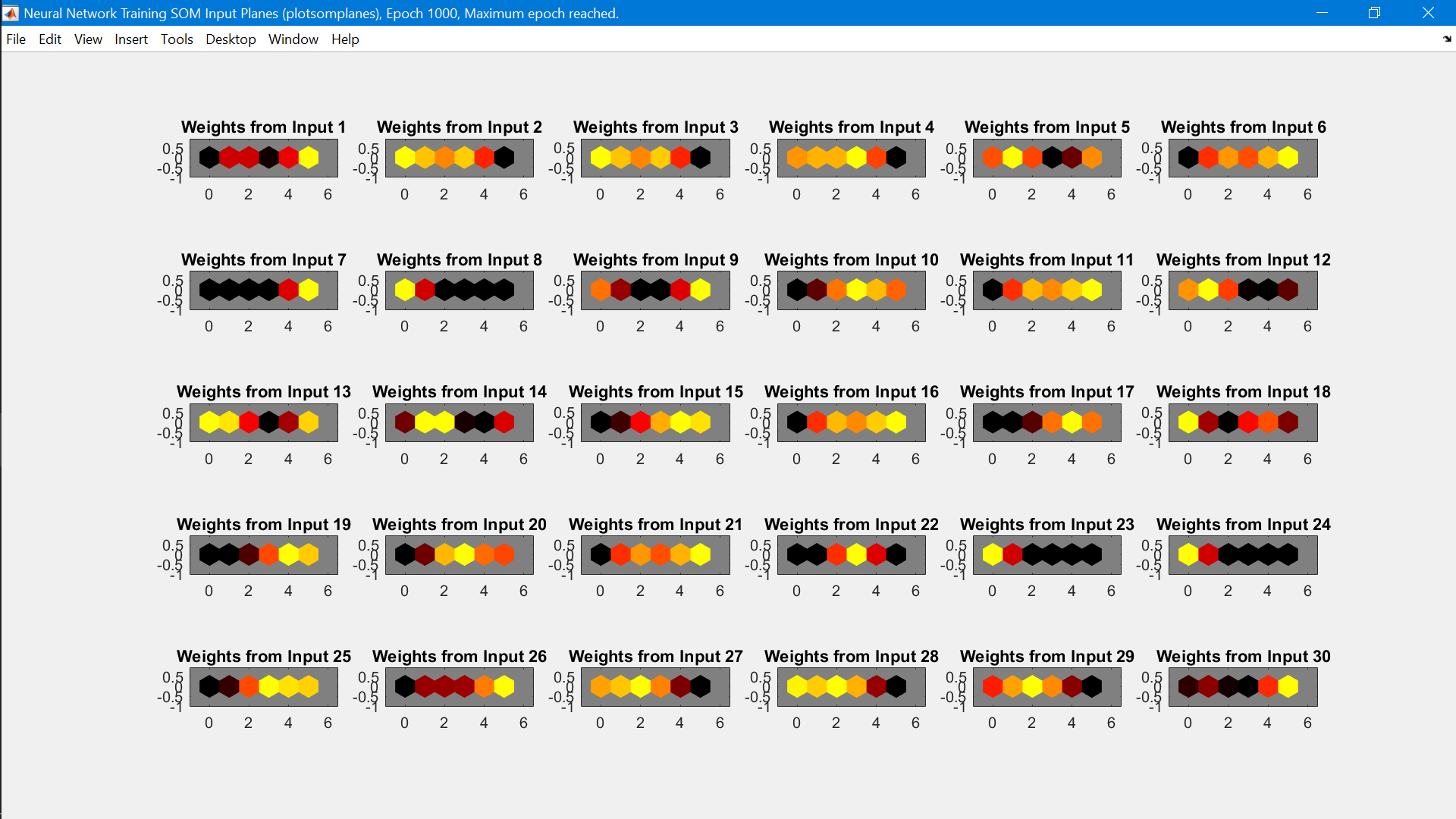


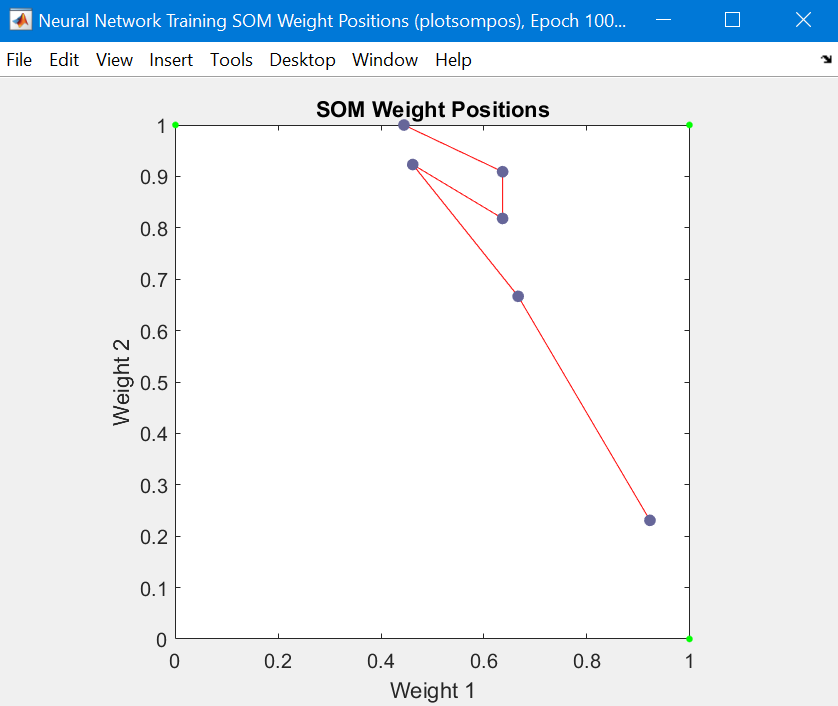


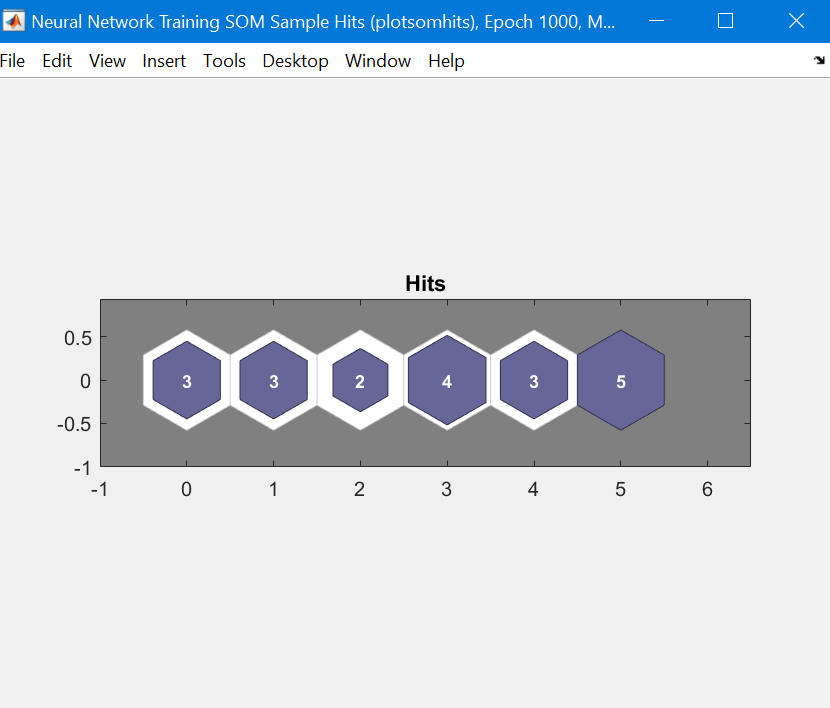
****

****

**

**

**

**

Analiza: Odpowiednie wykresy i mapy odpowiadają za:

* Wynik działania programu;
  + -Czerwone to połączenia
  + -Zielone to dane wejściowe
  + -Niebieskie to neurony.
* Ilość zwycięzców;
* Rozłożenie wag na dwóch różnych wejściach - im ciemniejszy kolor tym większa waga;
* Dystans pomiędzy neuronami - im ciemniejszy kolor tym dalszy dystans;

1. **WNIOSKI:**
   1. Odpowiedni dobór ilości neuronów jest najważniejszy przy wykorzystaniu SOM. WTM powoduje konkurencje między neuronami. Lecz jednak WTM nie pozwala na dominacje jednego neuronu, przez co rozkład sygnałów wejściowych jest bardziej równomiernie rozłożony. Jest to pierwsza różnica między WTM, a WTA. WTM nie powinno prowadzić do zagłodzenia neuronu.
   2. Wielkość mapy nie ma znaczenia na końcowy wynik, chyba, że użyjemy za małej. Dla naszej sieci testowałem wielkość 3x3 , 6x6 , 12x12;
   3. Najczęściej używanymi topologiami są sieci złożone z kwadratów lub sześcianów. W naszym przypadku obie topologie działają poprawnie, również dla różnych tablic danych. Różnica występuje gdy skorzystamy z topologii randomowej.

* 1. W tym rodzaju sieci nie mamy możliwości zmiany współczynnika uczenia. Może dlatego, że jest on ważną częścią zasad działania SOM-ów. Kontroluje on bezpośrednio wielkość wag. Współczynnik uczenia nie jest stały i zmniejsza się wraz z kolejnymi epokami nauki.
  2. Tym razem w porównaniu do metody WTA widać, że rozkład danych na grupy jest dużo bardziej uśredniony, grupy są znacznie mniejsze i więcej neuronów było w stanie zacząć je grupować. Dzięki temu nie ma żadnego neuronu który zagarnia wszystkie sygnały dla siebie. Lecz pojawia się pytanie jak na wyniki wpłynie jednowymiarowa topologia sieci. Sprawdziłem odpowiedź na to pytanie doświadczalnie tworząc sieć jednowymiarową. Odległości między neuronami i samo położenie neuronów jest w obu przypadkach bardzo podobne, co upewnia nas co do poprawności działania sieci.

1. **ŹRÓDŁA:**

* <https://edux.pjwstk.edu.pl/mat/273/lec/wyklad3/w3.html>
* <https://forum.pasja-informatyki.pl/73877/sztuczne-sieci-neuronowe>
* <http://www.if.uz.zgora.pl/~mdudek/siecineuronowe.pdf>
* <https://www.youtube.com/watch?v=Wa_9S20SkKw&t=995s>
* http://home.agh.edu.pl/~horzyk/pracedoktorskie/adrianhorzykpracadoktorska.pdf
* http://pracownik.kul.pl/files/31717/public/Model\_neuronu\_WTM.pdf
* Materiały udostępnione przez dr inż. Dorota Wilk – Kołodziejczyk
* Notatki z Wykładu „Podstawy Sztucznej Inteligencji”

1. **KOD PROGRAMU:**

close all;

%clear all;

clc;

%A B C D E F G H I K L J M N O P R S T U

dataIN =[0 1 0 1 1 1 0 1 0 1 1 0 1 1 0 1 1 0 1 1;

1 1 1 1 1 1 1 0 1 0 0 1 0 0 1 1 1 1 1 0;

1 1 1 1 1 1 1 0 1 0 0 1 0 0 1 1 1 1 1 0;

1 1 1 1 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0;

0 0 1 0 1 1 0 1 0 1 0 1 1 1 0 0 0 0 1 1;

1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1;

0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0;

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0;

0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0;

1 1 0 1 0 0 1 1 0 0 0 1 1 1 1 1 1 0 0 1;

1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 0 1;

0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0;

0 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 0 1 0;

0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0;

1 0 0 1 0 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1;

1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 0 1;

1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0;

1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 1 1 0;

1 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 1 0 0 0;

1 1 0 1 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1;

1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1;

0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0;

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0;

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0;

1 1 0 1 0 0 1 1 0 1 0 1 1 1 1 0 1 0 0 0;

1 1 0 1 1 1 0 1 0 1 1 0 1 1 0 1 1 0 0 0;

0 1 1 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1;

0 1 1 1 1 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1;

0 1 1 1 1 1 1 0 1 0 0 1 0 0 1 0 0 0 0 1;

1 0 1 0 1 0 0 1 0 1 1 0 1 1 0 0 1 0 0 0;];

[m, n] = size(dataIN);

% SOM parametry

dimensions = [6];

%dimensions = [6,6];

coverSteps = 100;

initNeighbor = 1;

topologyFcn = 'hextop';

distanceFcn = 'linkdist';

% Tworzenie SOM

net = selforgmap(dimensions,coverSteps,initNeighbor,topologyFcn,distanceFcn);

net.trainParam.epochs = 200;

net.trainFcn = 'trainbu';

% Trenowanie sieci

[net,tr] = train(net,dataIN);

dataOUT = net(dataIN);

classes = vec2ind(dataOUT);

%plotsompos(net,dataIN);

%grid on