Sprawozdanie z sieci Kohonena

Aleksandra Wichrowska, 291140

4 maj 2020

"Potwierdzam samodzielność powyższej pracy oraz niekorzystanie przeze mnie z niedozwolonych źródeł.

Aleksandra Wichrowska

Spis treści

1 Bazowa implementacja

1.1 Krótki opis problemu

Pierwsze zadanie polegało na bazowej implementacji sieci Kohonena. Zakładaliśmy, że sieć jest złożona z neuronów w prostokątnej siatce $M \times N$, (M i N to paramatery podawane przez użytkownika).
Implementacja obejmowała dwie funkcje sąsiedztwa:

- funkcję gaussowską
- funkcję mexican-hat (minus druga pochodna funkcji gaussowskiej)

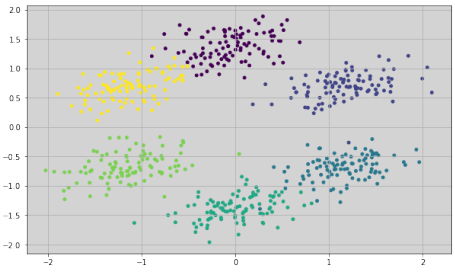
Funkcję wygaszającą uczenie wraz z kolejnymi iteracjami zdefiniowaliśmy jako: $\alpha(t) = e^{-t/\lambda}$

1.2 Testy i wizualizacje

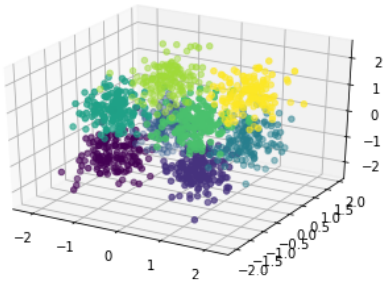
Testy przeprowadziłam na dwóch zbiorach danych:

- danych 2D skupionych w wierzchołkach sześciokąta
- danych 3D skupionych w wierzchołkach sześcianu.

1.2.1 Wizualizacja zbiorów danych



(a) hexagon

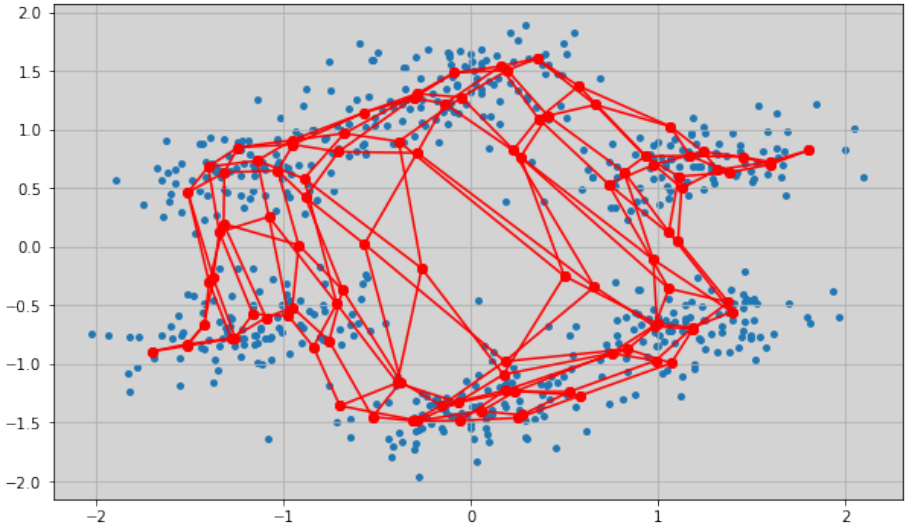


(b) cube

Rysunek 1

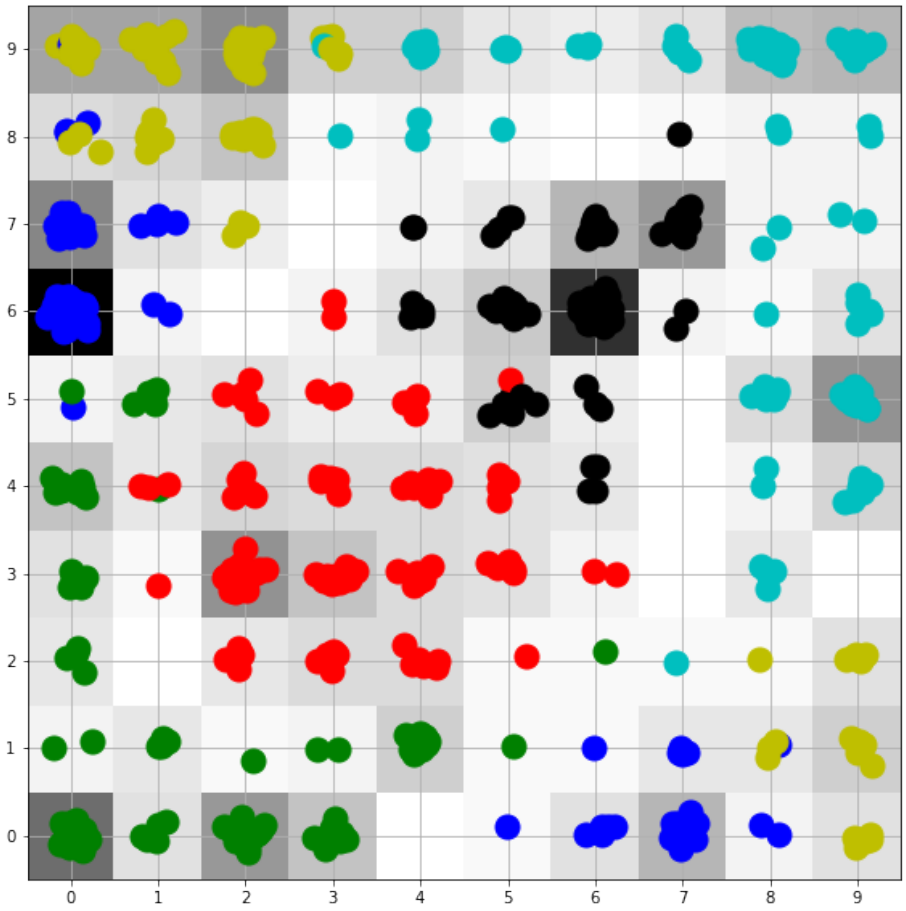
1.2.2 Testy dla zbioru *hexagon*

Poniższa wizualizacja prezentuje neurony w przestrzeni wektorów wejściowych (współrzędne neuronów to ich wagi). Połączone odcinkiem zostały neurony, które w topologii prostokąta sąsiadują ze soba. Można zauważyć zagęszczenie neuronów w pobliżu wierzchołków sześciokąta - sieć poprawnie dopasowała się do danych.



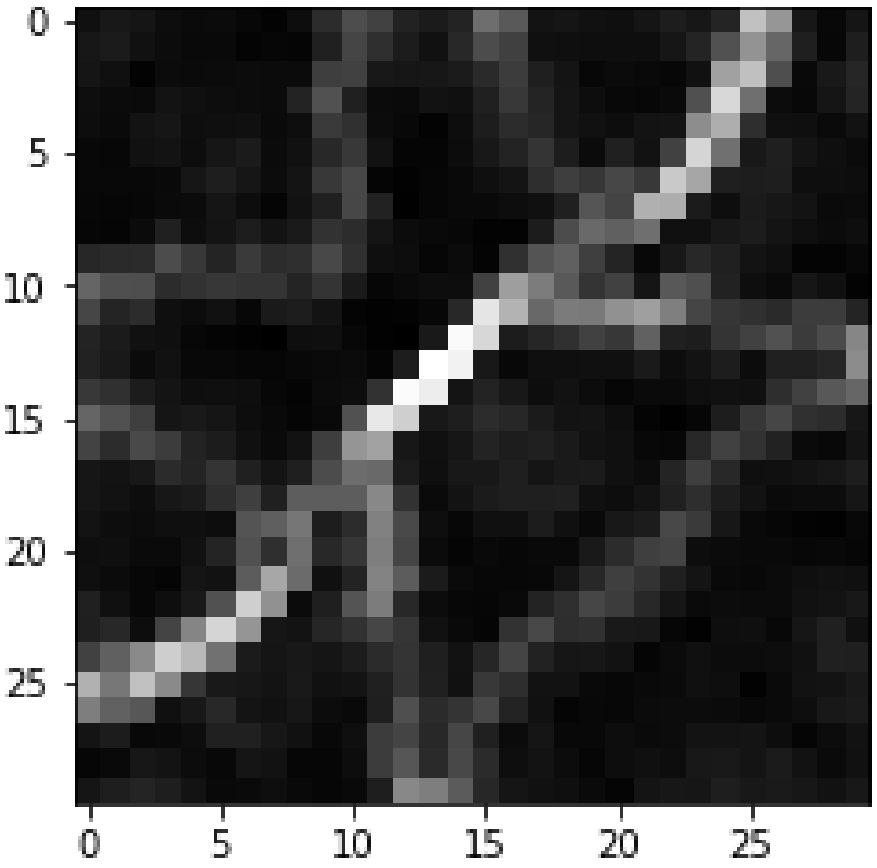
Rysunek 2: Połączenia między neuronami po 1000 iteracji dla sieci 10x10

Poniższa wizualizacja prezentuje wykres gęstości neuronów - dla każdego neuronu zliczamy dla ilu wektorów wejściowych ten neuron jest BMU. Ciemne kolory - duże wartości, jasne kolory - małe wartości. Dodatkowo dla każdego wektora wejściowego stawiamy punkt w odpowiednim kolorze (zależnym od klasy) w obszarze odpowiadającym jego neurowi BMU. Mozna zaobserwować dobry podział na klastry, kolory są dobrze odseparowane.



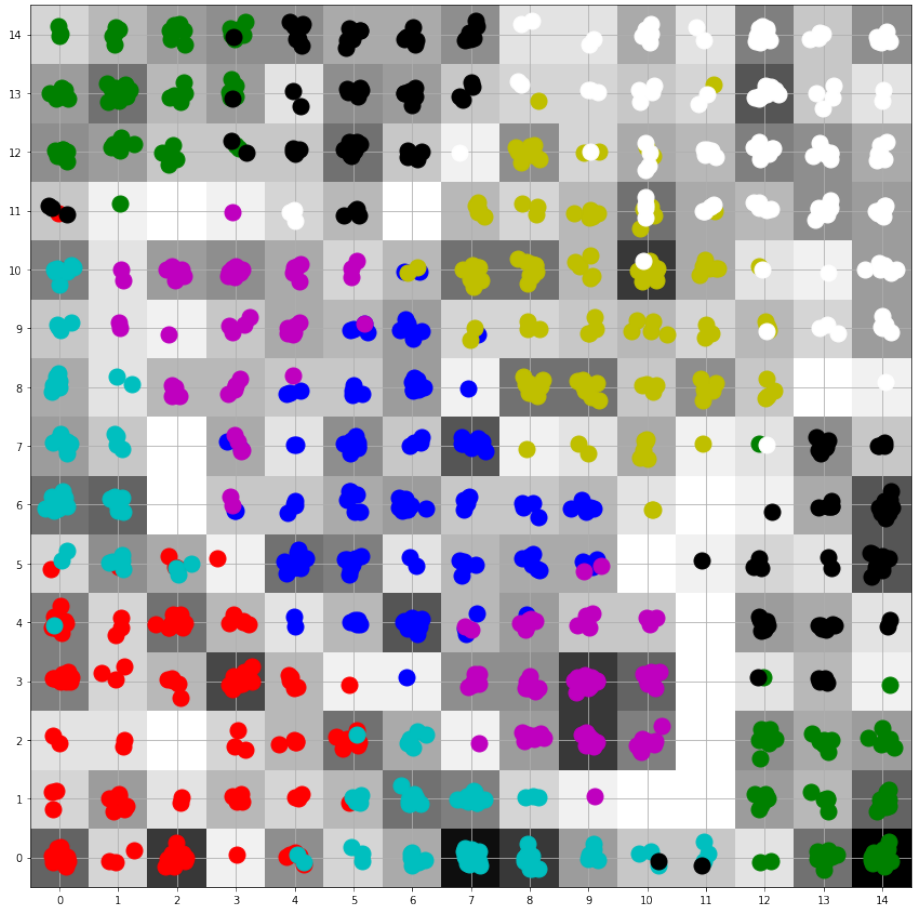
Rysunek 3: 'Density plot' po 100 iteracjach dla sieci 10x10

Poniższa wizualizacja prezentuje wykres zwany $U - matrix$ - dla każdego neuronu obliczamy średnią z odległości tego neuronu to neuronów sąsiadujących. Na podstawie otrzymanych wartości inicjalizujemy mapę ciepła - duże wartości oznaczamy ciemnym kolorem, małe - jasnym. Jasne kolory tworzą granice pomiędzy klastrami. Możemy zauważyć że sieć wyodrębniła kilka klastrów.



Rysunek 4: U-Matrix po 2000 iteracji dla sieci 20x20

1.2.3 Testy dla zbioru *cube*

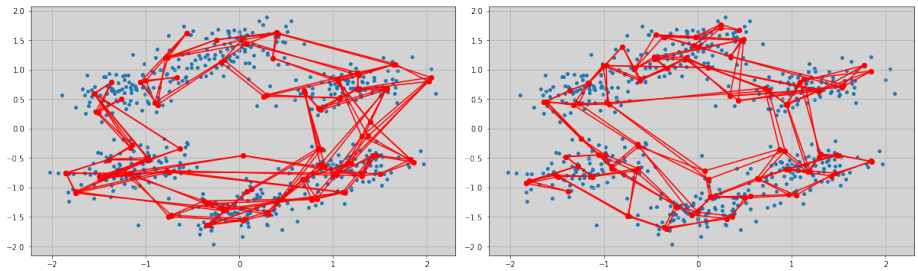


Rysunek 5: 'Density plot' po 500 iteracjach dla sieci 15x15

Dla zbioru *cube* algorytm również osiągnął dobre wyniki - klasy są widocznie odseparowane.

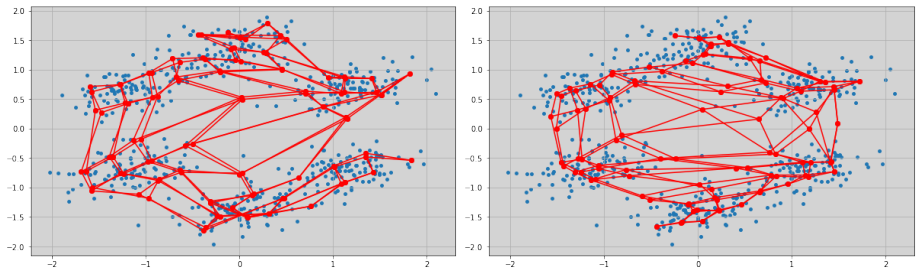
1.2.4 Testy dla zmiany szerokości sąsiedztwa

Dodatkowym parametrem w implementacji sieci Kohonena jest szerokość sąsiedztwa, która wpływa na to jak bardzo wagi neuronów bliskich neuronowi BMU będą korygowane. Sprawdzałam jak zmieniają się wizualizacje dla 4 wartości tego parametru: 0.1, 0.2, 0.5, 1



(a) $radius = 1$

(b) $radius = 0.5$

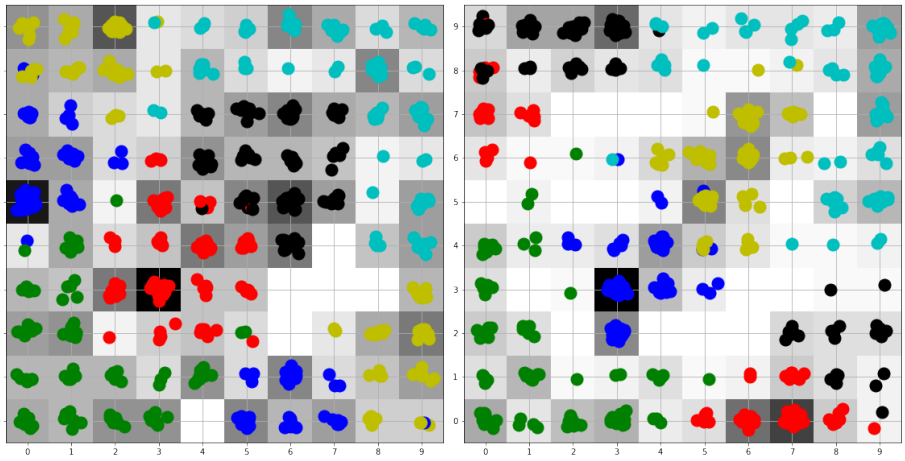


(c) $radius = 0.2$

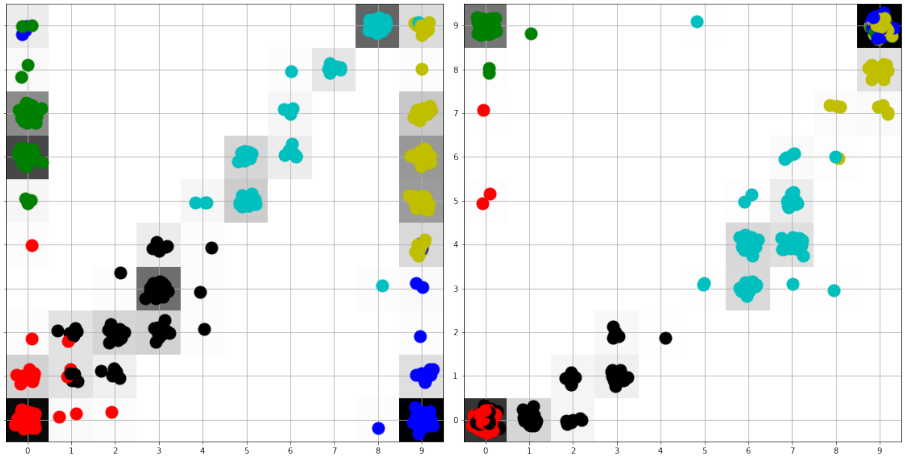
(d) $radius = 0.1$

Rysunek 6: Połączenia między neuronami po 1000 iteracji dla sieci 10x10

Można zauważyć że dla większych wartości parametru *radius* sieć szybciej dopasowuje się do kształtu danych. Jest to spodowodowane większym korygowaniem wag neuronów. Jednocześnie w przypadku innych danych duży *radius* może źle wpłynąć na uczenie - wagi będą się zmieniać dla neuronów zbyt odległych od siebie.



(a) $radius = 1$ (b) $radius = 0.5$



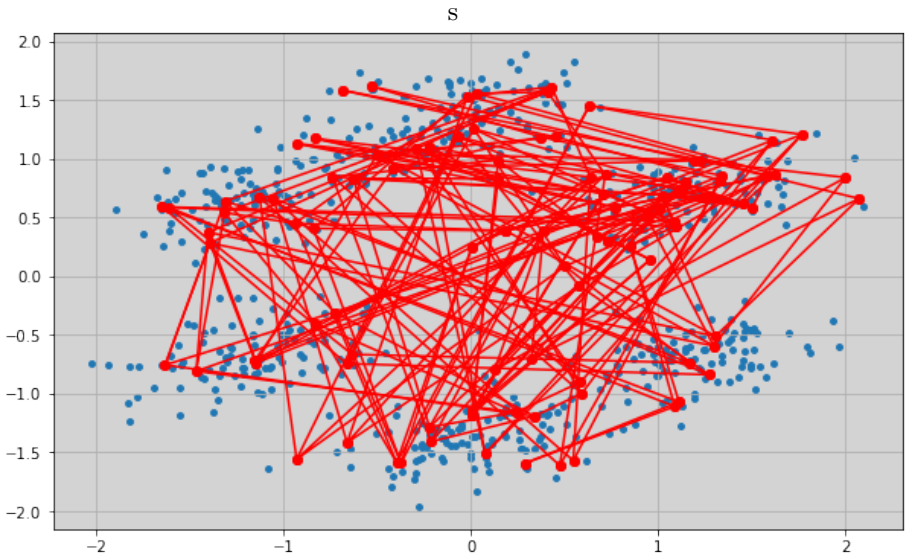
(c) $radius = 0.2$ (d) $radius = 0.1$

Rysunek 7: 'Density plot' po 500 iteracjach dla sieci 15x15

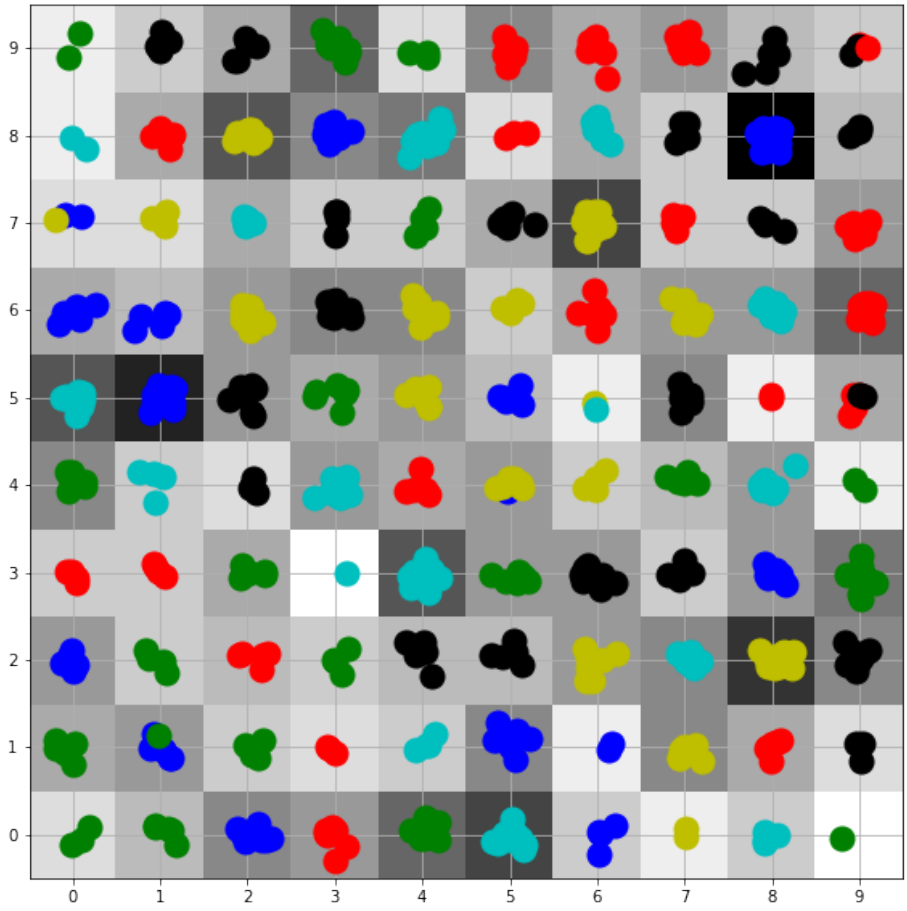
Można zauważyć, że dla większych wartości parametru *radius* wynikowy *density-plot* jest gęstszy, klasy są odseparowane, ale leżą blisko siebie. Dla mniejszych wartości *radius* klasy są dużo mocniej odseparowane od siebie.

1.3 Testy dla funkcji *mexican-hat*

Ze względu na dużą trudność w trenowaniu sieci przy użyciu funkcji sąsiedztwa *mexican-hat* (trudno znaleźć parametr sąsiedztwa, dla którego sieć widocznie separuje dane na klastry) wykonałam jedynie kilka przykładowych wizualizacji dla tej funkcji, a inne testy sieci i jej parametrów przeprowadzałam z funkcją gaussowską.



Rysunek 8: Połączenia między neuronami po 1000 iteracji dla sieci 10x10

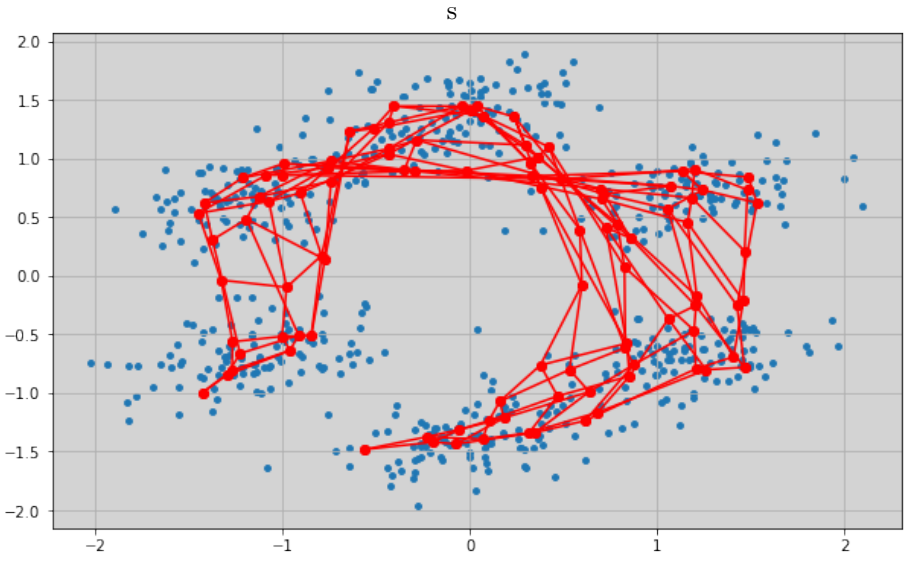


Rysunek 9: Podział na klastry na siatce heksagonalnej

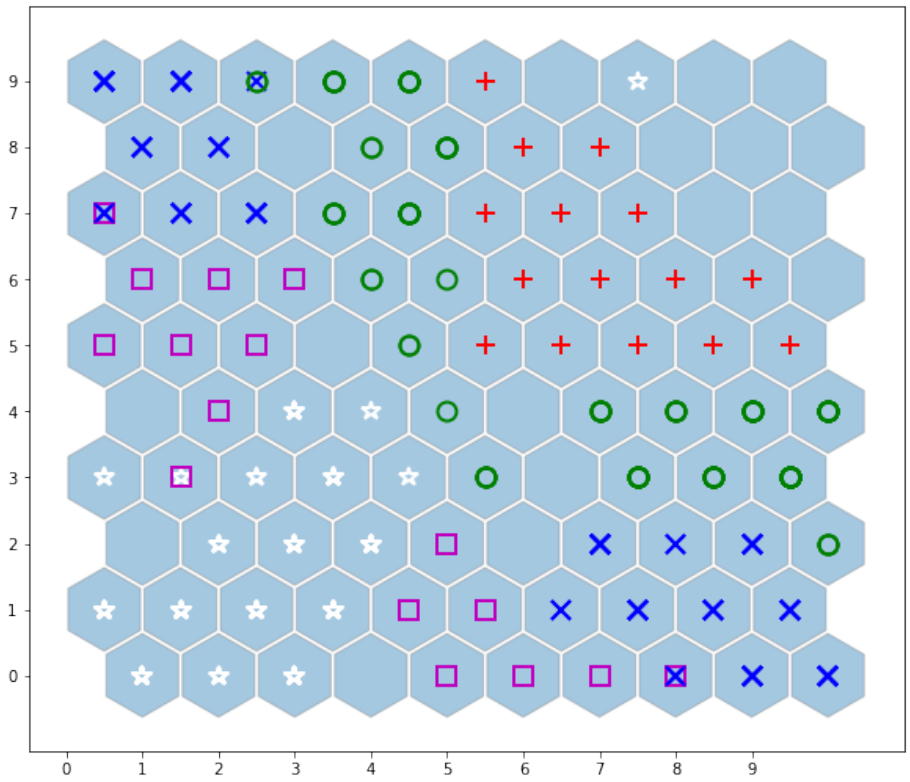
Wagi neuronów dobrze dopasowały się do kształtu danych (rysunek 8), ale podział na klastry nie jest zby dokładny, klastry nie są od siebie odseparowane (rysunek 9).

2 Implementacja topologii heksagonalnej

Jedyną różnicą w topologii heksagonalnej jest obliczanie odległości pomiędzy neuronami na mapie 2D. Przy zastosowaniu tego modelu przeprowadziłam takie same testy i wizualizacje jak poprzednio na zbiorze *cube*.



Rysunek 10: Połączenia między neuronami po 1000 iteracji dla sieci 10x10



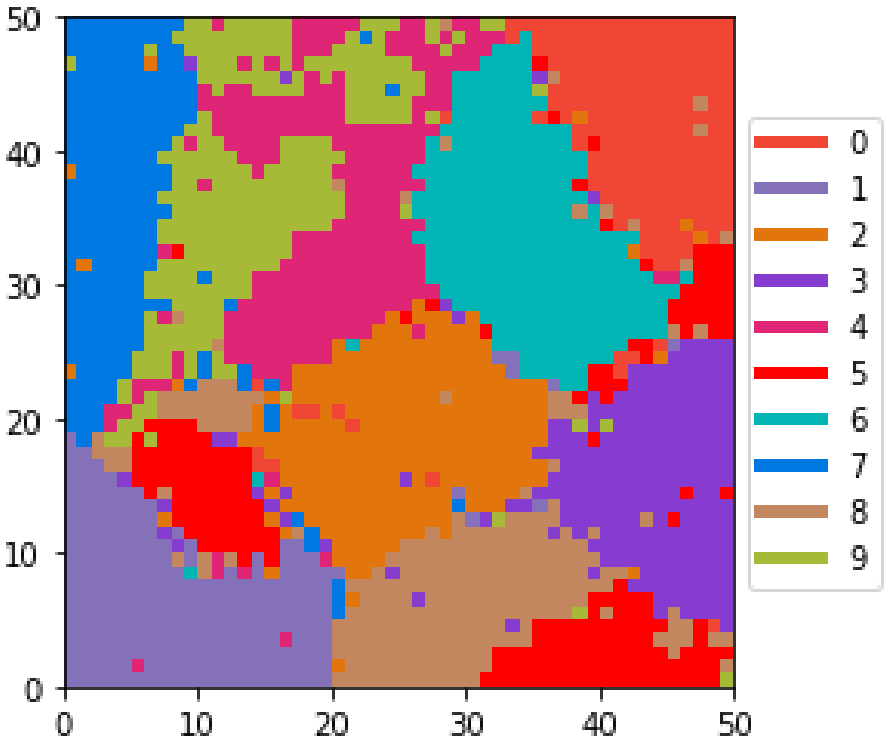
Rysunek 11: Podział na klastry na siatce heksagonalnej

Wniosek: Samouczenie sieci na siatce heksagonalnej również działa bardzo dobrze, klastry są odseparowane od siebie.

3 Testy na dużych zbiorach danych

3.1 Testy na zbiorze MNIST

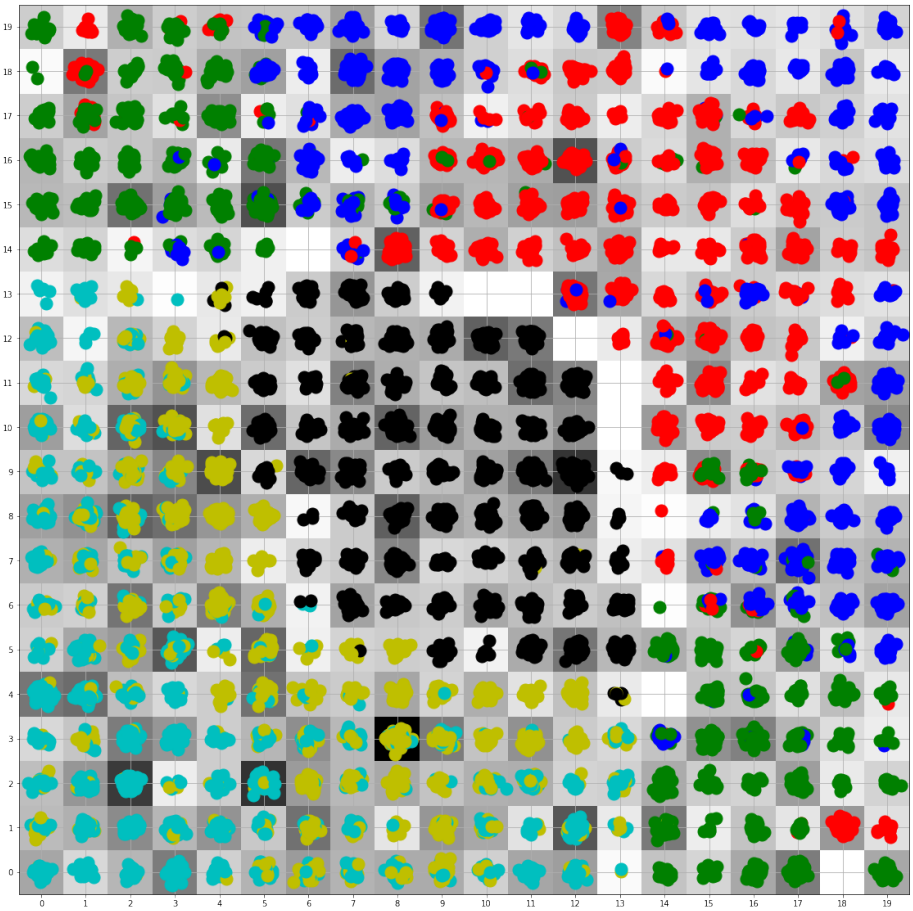
Ze wgzlędu na duży rozmiar zbioru MNIST delikatnie zmodyfikowałam algorytm uczenia oraz sposób wizualizacji wyników. W każdej epoce uczenia brałam pod uwagę jedynie część wektorów inputu (mniejszy szum), natomiast zwiększyłam liczbę epok. Test przeprowadziłam na sieci prostokątnej 50×50. Dla każdego wektora wejściowego wyznaczyłam neuron BMU, dla każdego neuronu zliczałam ile razy jest on BMU dla każdej z cyfr. Odpowiednim kolorem oznaczyłam na poniższej wizulizacji cyfry które dominowały.



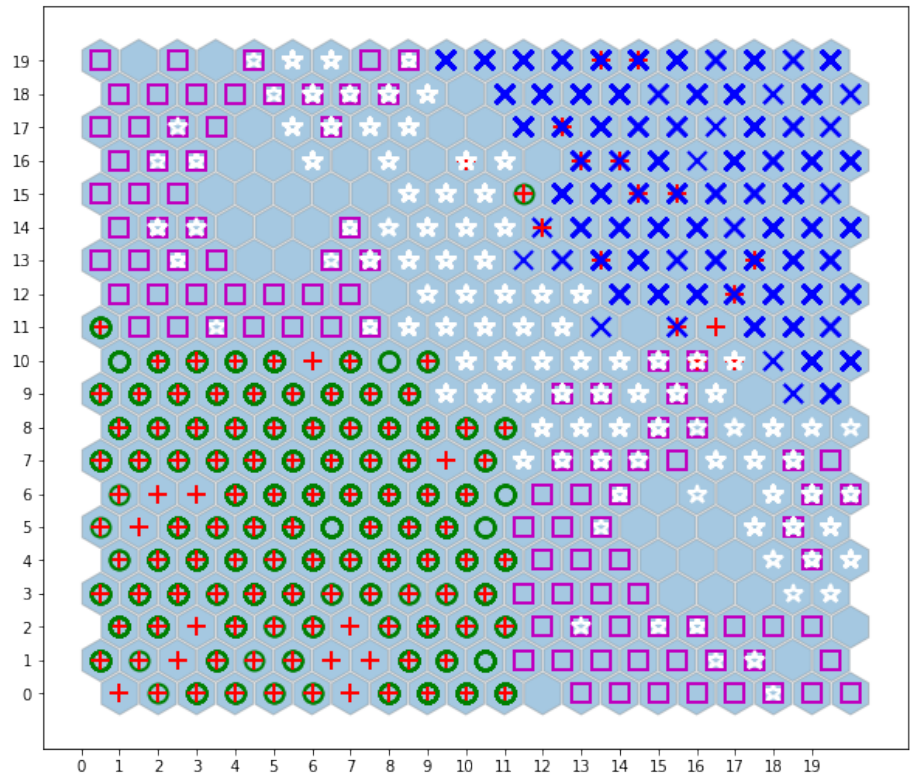
Rysunek 12: Podział na klastry na siatce prostokątnej

3.2 Testy na zbiorze HARUS

Zbiór HARUS jest podzielony na 6 klastrów.



Rysunek 13: Podział na klastry na siatce prostokątnej po 500 epokach dla sieci 20×20



Rysunek 14: Podział na klastry na siatce heksagonalne po 100 epokach dla sieci 20×20

Wniosek: Sieć dobrze odwzorowuje granice pomiędzy klastrami już po niewielkiej liczbie iteracji (epok).