Sprawozdanie - Sieci Kohonena

Jakub Lis
305744

Streszczenie

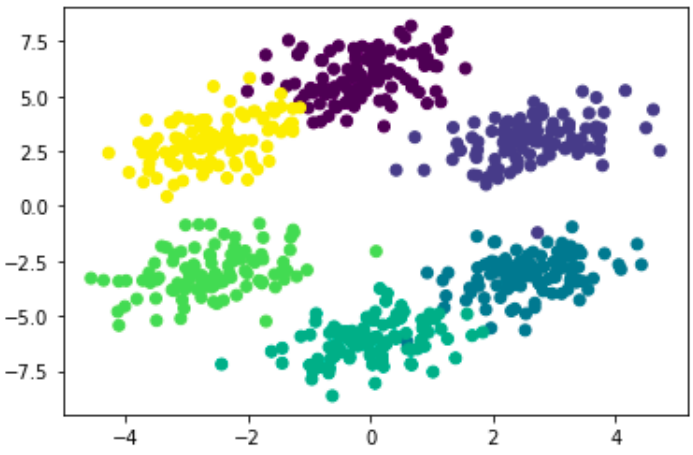
Laboratorium dotyczyło implementacji sieci Kohonena. W pierwsze dwa tygodnie powstała bazowa implementacja sieci Kohonena złożona z neuronów w prostokątnej siatce. Następnie implementacja została rozszerzona o siatkę sześciokątną. W sprawozdaniu został opisany wpływ różnych parametrów na działanie sieci.

1 Podstawowa sieć Kohonena

Podstawowa implementacja sieci Kohonena złożona jest z prostokątnej siatki M x N, gdzie M, N są parametrami programu. Końcowo algorytm dzieli wektory właśnie na $N \cdot M$ klastrów. Używanymi funkcjami sąsiedztwa są funkcja gaussowska oraz meksykański kapelusz, czyli minus druga pochodna funkcji gaussowskiej.

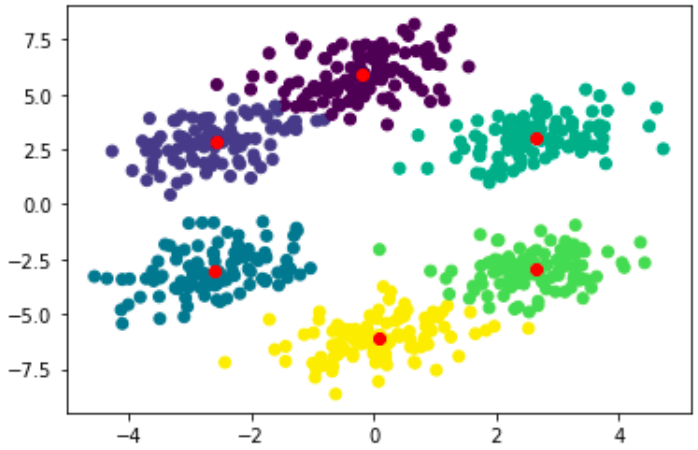
1.1 Zbiór danych hexagon

Zbiór hexagon zawiera dane 2d skupione w wierzchołkach sześciokąta. Na rysunku poniżej kolorami zostały oznaczone klasy punktów, które dla sieci Kohonena będą niedostępne i spróbujemy je odtworzyć.



Rysunek 1: Zbiór hexagon

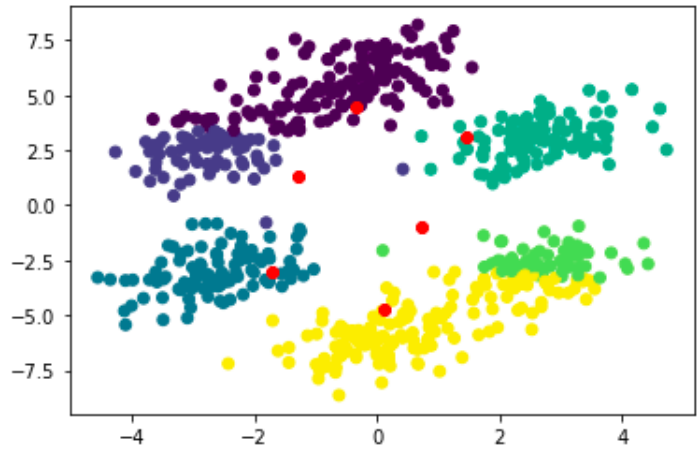
Na początku sprawdzamy jak zadziała sieć z szerokością sąsiedztwa 0.1, przy użyciu funkcji gaussowskiej i przy 100 epokach. W tym przypadku przyjmujemy $M = 3, N = 2$.



Rysunek 2: Przydzielone klasy po zastosowaniu sieci Kohonena

Czerwonym kolorem zostały oznaczone neurony sieci, które zostały przyciągnięte w odpowiednie miejsca przez zbitki punktów. Widzimy, że sieć poradziła sobie bardzo dobrze i uzyskaliśmy bardzo podobny rysunek jak wcześniej. Niespójność kolorów wynika jedynie z tego, że sieć nie zna poprawnych klas i z dużym prawdopodobieństwem przypisze je w innej kolejności niż oryginalnie zostały nadane.

Dla porównania zwizualizujemy jak sieć Kohonena na tych samych ustawieniach radzi sobie dla 10 epok.



Rysunek 3: Działanie przy 10-ciu epokach

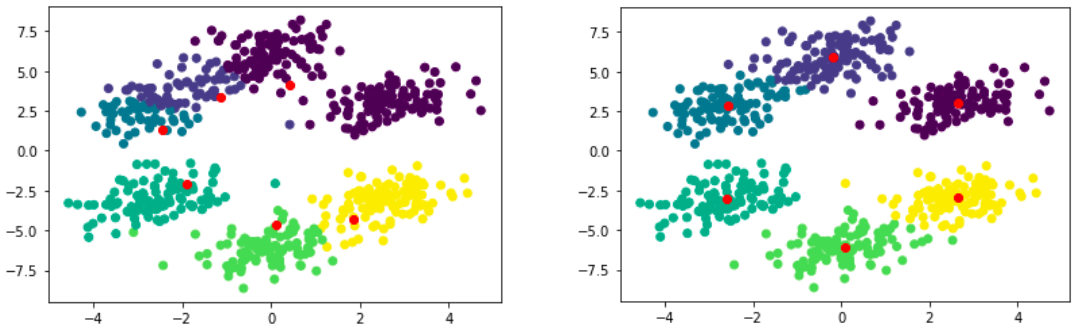
Widać, że 10 epok to za mało, ale neurony idą w dobrym kierunku. Dalej przeprowadzimy testy wyznaczając wartość accuracy dla 20-tu uruchomień sieci zmieniając ziarno losowości. Losowym elementem jest pierwsze rozlosowanie punktów prostokąta $3x2$, przy czym losowanie współrzędnych odbywa się przy ograniczeniu wartości od percentylu 25% do percentylu 75% ze zbioru danych. Ograniczenie takie pomaga dobrać punkty tak, aby żaden nie odstawał na tyle, że nigdy nie zbliży się do skupiska punktów.

lp.	accuracy po 10 epokach	accuracy po 20 epokach
1	0.93	0.99
2	0.93	0.98
3	0.95	0.99
4	0.95	0.98
5	0.92	0.99
6	0.93	0.98
7	0.92	0.98
8	0.92	0.98
9	0.92	0.99
10	0.91	0.98
11	0.93	0.98
12	0.94	0.99
13	0.94	0.98
14	0.93	0.98
15	0.93	0.99
16	0.94	0.99
17	0.94	0.99
18	0.93	0.98
19	0.94	0.99
20	0.92	0.98

Tabela 1: Uzyskane wartości accuracy podczas testowania sieci

Wobec tego przy wcześniej wspomnianych ustawieniach sieci Kohonena uczenie przebiega poprawnie niezależnie od ziarna losowości. Uśredniona wartośc accuracy po 20 epokach wynosi 0.986.

Dalej przetestujemy jak wpływa zmiana parametrów na działanie sieci. Zacznimy od zmiany parametrów M, N. Wiemy, że klas jest 6, przyjmijmy więc jeszcze $M = 6, N = 1$ (zamiast jak wcześniej $M = 3, N = 2$).



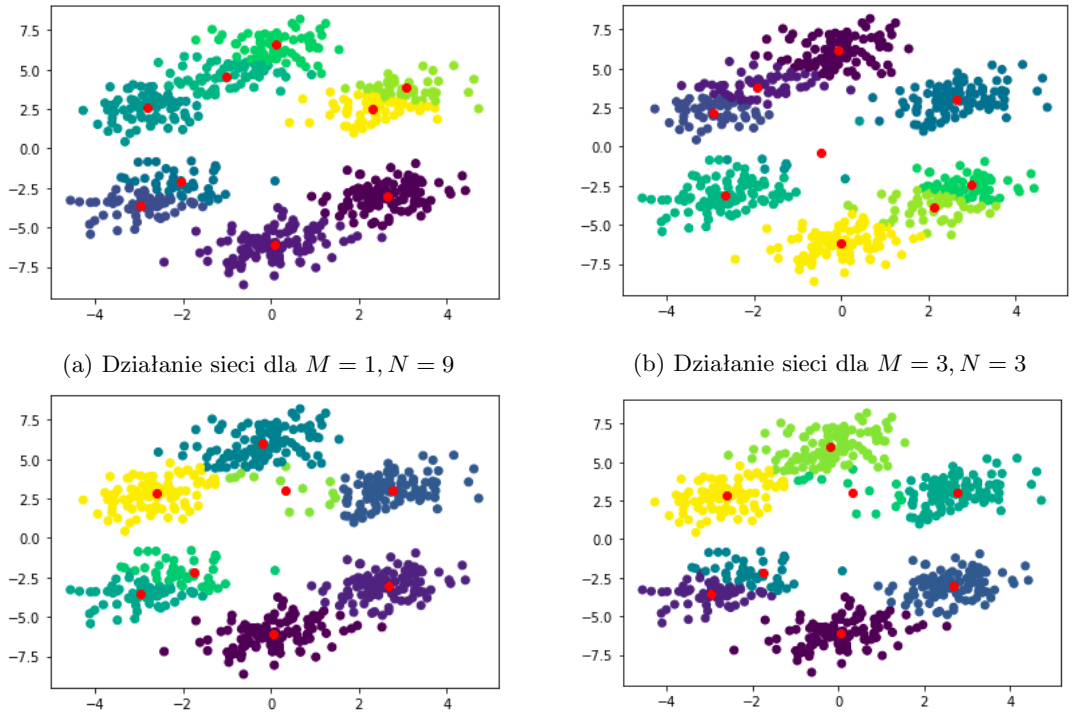
(a) Po 5-ciu epokach

(b) Po 100-tu epokach

Rysunek 4: Działanie sieci Kohonena z parametrami $M = 6, N = 1$

Dla powyższych wyników test został przeprowadzony 10-cio krotnie i za każdym razem otrzymywane wyniki były bardzo podobne, więc prezentowane jest jedno z uruchomień algorytmu.

Dalej spróbujemy zmodyfikować parametry M, N, ale wybierając je nie biorąc pod uwagę prawdziwej liczby klas.



(a) Działanie sieci dla $M = 1, N = 9$

(b) Działanie sieci dla $M = 3, N = 3$

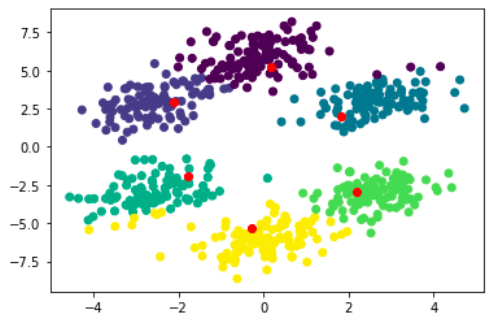
(c) Działanie sieci dla $M = 2, N = 4$

(d) Działanie sieci dla $M = 4, N = 2$

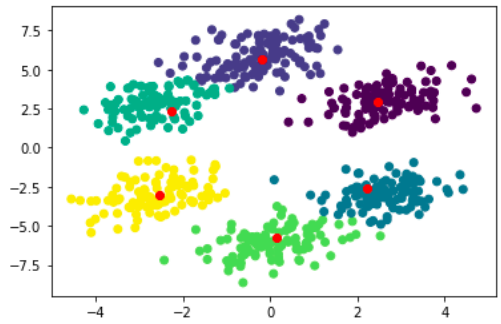
Rysunek 5: Działanie sieci Kohonena przy różnych wartościach M, N

Rysunki 5a, 5b pokazują, że pomimo ustalenia tej samej liczby docelowych klas ($M \cdot N$) dobranie różnych parametrów M, N może wpłynąć na końcowy wynik. Prezentowane wyniki są po uczeniu przy stu epokach. Rysunki 5c, 5d są przykładem, że pomimo zamiany miejscami parametrów można otrzymać identyczny końcowy wynik.

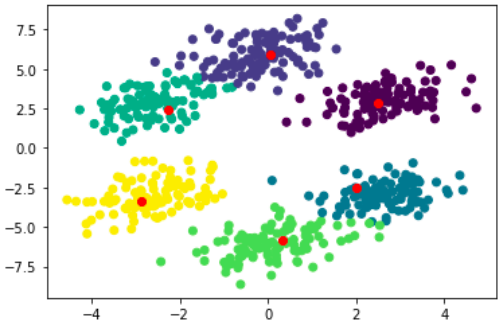
Sprawdzimy jeszcze wpływ zmian parametru learning_rate na naukę sieci. Wracamy do ustawień $M = 3, N = 2$, pozostając dalej przy funkcji gaussowskiej jako funkcji sąsiedztwa.



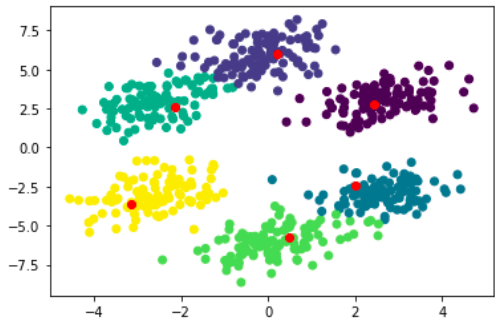
(a) Działanie sieci przy learning_rate=0.01



(b) Działanie sieci przy learning_rate=0.1



(c) Działanie sieci przy learning_rate=0.5

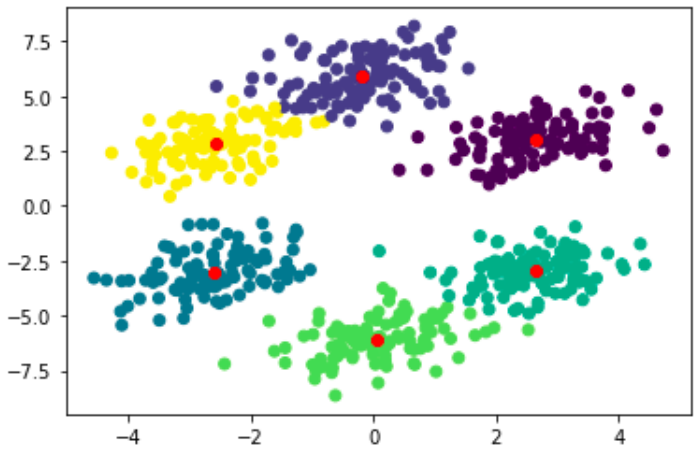


(d) Działanie sieci przy learning_rate=1

Rysunek 6: Działanie sieci Kohonena przy różnych wartościach kroku uczenia

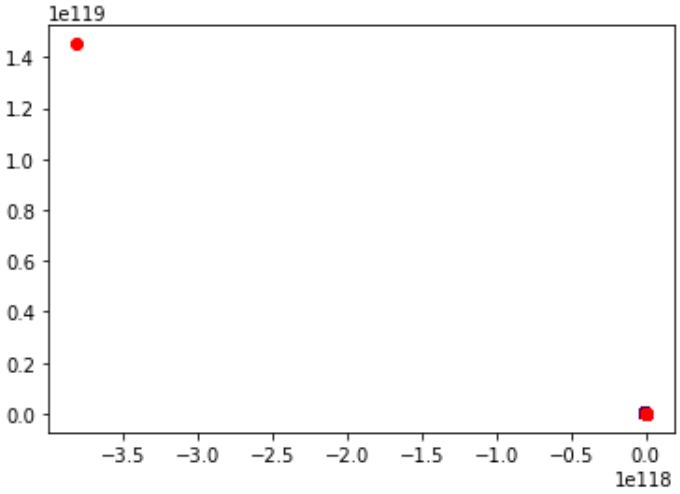
Ponownie, wyniki zostały przestowane przy różnych ziarnach losowości i nie różniły się od siebie znacząco. Widzimy z rysunków, że niezależnie od parametru learning_rate sieć radziła sobie bardzo dobrze. Pokazane wyniki są po 20 epokach uczenia i jedynie dla learning_rate=0.01 było to za mało, ale sieć można douczyć i uzyskać identyczne wyniki jak dla pozostałych wartości.

Kolejny rysunek (rysunek 7) prezentuje już działanie sieci Kohonena przy użyciu naszej drugiej funkcji sąsiedztwa - meksykańskiego kapelusza. Wykorzystana siatka



Rysunek 7: Przydzielone klasy po zastosowaniu sieci Kohonena z funkcją sąsiedztwa meksykański kapelusz

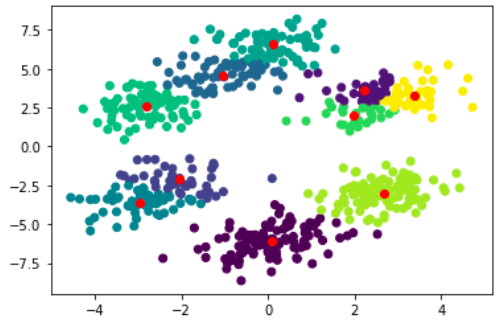
Warto podkreślić, że przy tej funkcji sąsiedztwa trzeba było mocno zminimalizować wartość szerokości sąsiedztwa, tzn. przyjąć tę wartość na poziomie 10^{-4}. Poniższy rysunek przedstawia zachowanie sieci, gdy parametr ten jest równy 1.



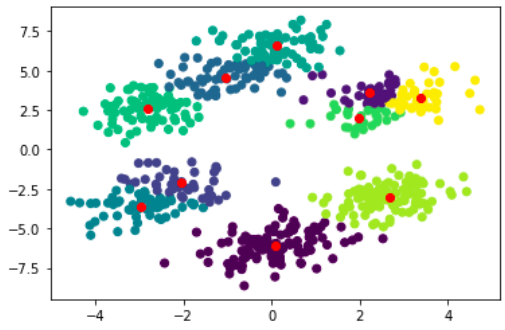
Rysunek 8: Uczenie sieci Kohonena, gdy szerokość sąsiedztwa jest równa 1

Widać, że gdy szerokość sąsiedztwa nie jest odpowiednio mała, to jeden z neuronów bardzo oddala się od punktów z naszego zbioru danych. Dalej w testach będziemy już przyjmować tę szerokość na poziomie 10^{-4}. Zacznijmy ponownie od modyfikacji parametrów M, N przy użyciu funkcji meksykański kapelusz. Dla przypadków $M = 3, N = 2$; $M = 2, N = 3$; $M = 6, N = 1$; $M = 1, N = 6$

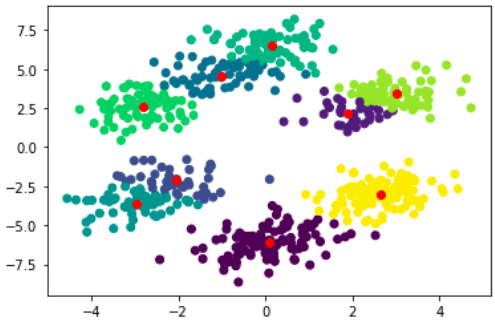
otrzymujemy zawsze te same wyniki niezależnie od ziarna losowości. Nie będziemy prezentować jeszcze raz tego samego, wszystkie uruchomienia dały wyniki, które są na rysunku 7. Dodatkowo wyznaczone i uśrednione accuracy we wszystkich przypadkach wynosio 0.988. Zaprezentujemy natomiast działanie algorytmu, gdy $M \cdot N \neq 6$.



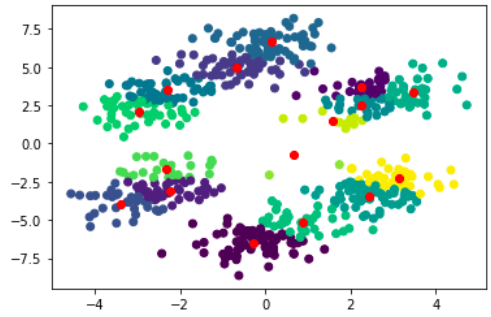
(a) Działanie sieci na siatce 2x5



(b) Działanie sieci na siatce 5x2



(c) Działanie sieci na siatce 3x3

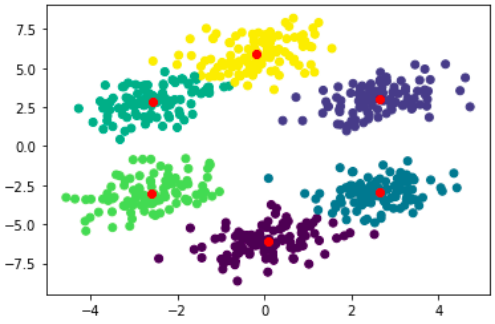


(d) Działanie sieci na siatce 4x4

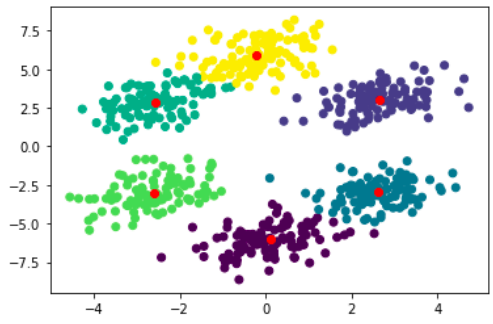
Rysunek 9: Działanie sieci Kohonena przy różnych wartościach parametrów M, N

W przypadku parametrów $M = 2, N = 5$ oraz $M = 5, N = 2$, które są odpowiednio na rysunkach 9a i 9b nie ma znaczącej różnicy. Dodatkowo bardzo podobnie wyglądają klastry dla $M = 3, N = 3$, czyli gdy założyliśmy podział na 1 klasę więcej (rysunek 9c). W przypadku $M = 4, N = 4$ (rysunek 9d) także mamy naturalne zachowanie algorytmu - w końcu sami wymusiliśmy podział na 16 klas.

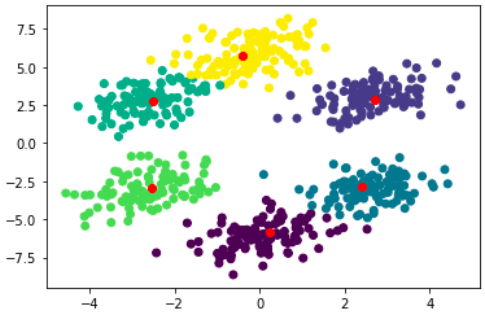
Dalej sprawdzimy wpływ parametru learning_rate na zachowanie algorytmu, gdy używamy funkcji meksykański kapelusz. W tym celu wracamy do $M = 3, N = 2$.



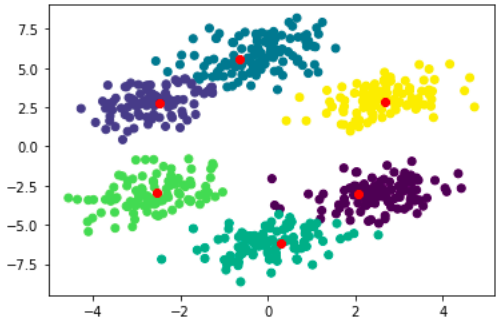
(a) Działanie sieci przy learning_rate=0.01



(b) Działanie sieci przy learning_rate=0.1



(c) Działanie sieci przy learning_rate=0.5



(d) Działanie sieci przy learning_rate=1

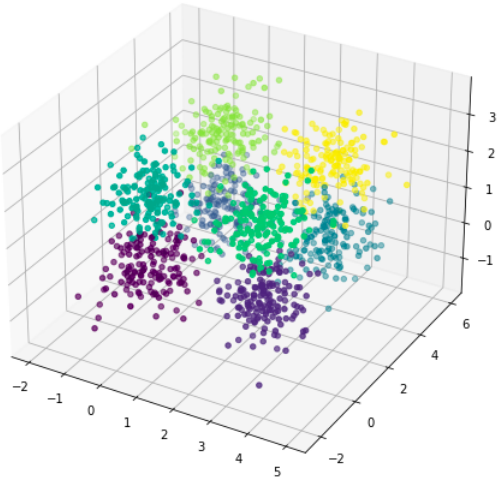
Rysunek 10: Działanie sieci Kohonena przy różnych wartościach learning_rate

Wszystkie rysunki powyżej przedstawiają jak zadziałała sieć Kohonena przy 20-tu epokach. Widzimy, że nawet dla tak małej liczby epok wszystkie wybrane wartości learning_rate poradziły sobie niemalże idealnie. Widzimy także, że podział na klastry we wszystkich przypadkach jest identyczny.

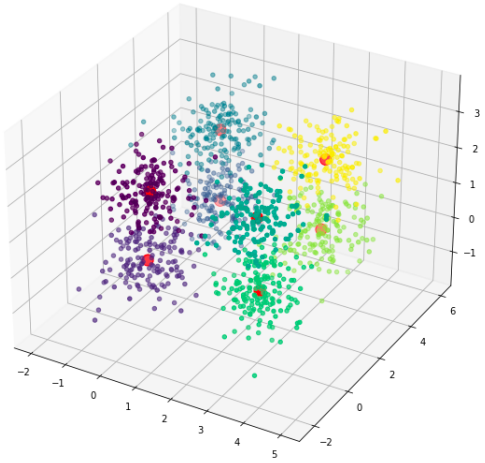
Podsumowując, obie funkcje sąsiedztwa działały dla zbioru hexagon bardzo podobnie i bardzo dobrze klastrowały te dane.

1.2 Zbiór danych cube

Zbiór cube zawiera dane 3d skupione w wierzchołkach sześcianu. Na rysunku 11a kolorami zostały oznaczone klasy punktów, które dla sieci Kohonena będą niedostępne i spróbujemy je odtworzyć analogicznie jak w przypadku poprzedniego zbioru. Do wizualizacji zbioru będziemy używać wykresów 3D implementowanych za pomocą bibliotek mpl_toolkits oraz matplotlib.



(a) Zbiór cube



(b) Przydzielone klasy po zastosowaniu sieci

Rysunek 11: Porównanie oryginalnych klastrów i klastrowania dokonanego przez sieć Kohonena

Tym razem mamy 8 klas, wobec tego domyślnie będziemy korzystać z siatki $2x4$. Ponownie, czerwonymi punktami oznaczymy neurony sieci. Wykres prezentujący uruchomienie sieci z funkcją sąsiedztwa gaussowską jest na rysunku 11b.

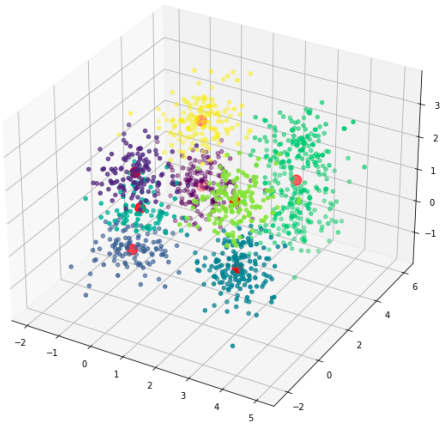
Poniższa tabelka przedstawia accuracy uzyskane za pomocą sieci Kohonena z parametrami opisanymi powyżej. Testowane były różne ziarna losowości przy uczeniu po 10-ciu epokach i następnie po douczeniu o kolejne 10 epok.

lp.	accuracy po 10 epokach	accuracy po 20 epokach
1	0.9	0.92
2	0.95	0.96
3	0.93	0.94
4	0.93	0.95
5	0.93	0.93
6	0.95	0.93
7	0.93	0.94
8	0.95	0.94
9	0.96	0.96
10	0.9	0.92
11	0.93	0.95
12	0.94	0.94
13	0.93	0.91
14	0.94	0.93
15	0.94	0.94
16	0.9	0.92
17	0.9	0.94
18	0.94	0.91
19	0.96	0.95
20	0.94	0.94

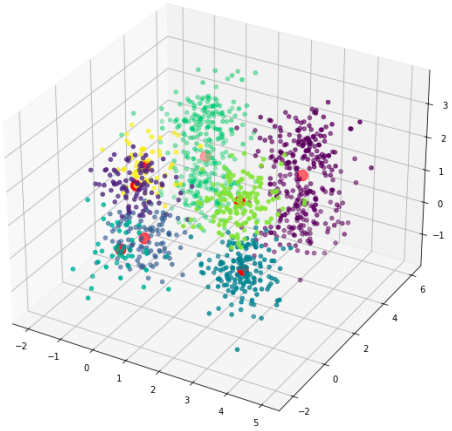
Tabela 2: Uzyskane wartości accuracy podczas testowania sieci

Widzimy zatem, że niezależnie od rozlosowania neuronów początkowych - sieć osiąga accuracy na poziomie wyższym niż 90%.

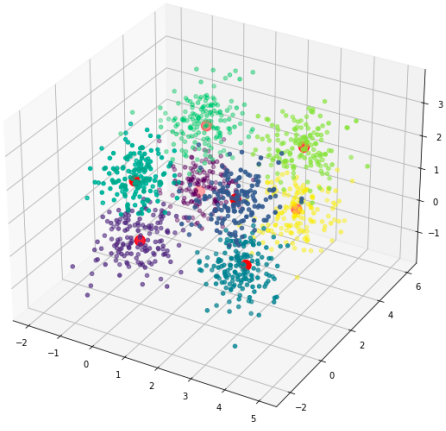
Dalej zbadamy wpływ learning_rate na naukę sieci na danych 3D.



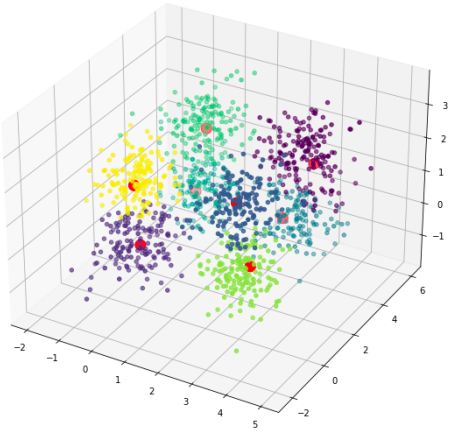
(a) Działanie sieci przy learning_rate=0.01



(b) Działanie sieci przy learning_rate=0.1



(c) Działanie sieci przy learning_rate=0.5



(d) Działanie sieci przy learning_rate=1

Rysunek 12: Działanie sieci Kohonena przy różnych wartościach learning_rate na zbiorze Cube

Zauważmy, że poprawne klastrowanie (tzn. zgodne z oryginalnymi klastrami) otrzymaliśmy jedynie w przypadku learning_rate 0.5 oraz 1. W przypadku 0.01 oraz 0.1 klastrowanie jest także niezłe, ale niezgodne z oryginalnymi klasami.

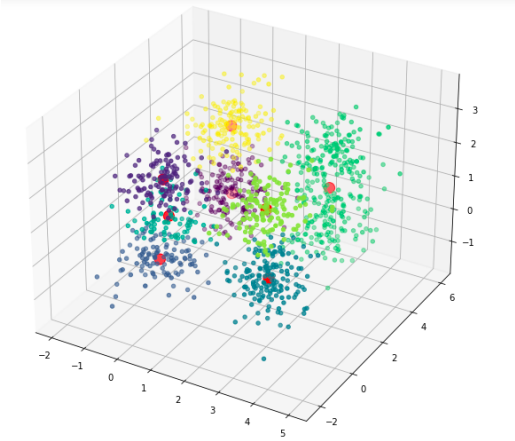
Dalej sprawdzimy działanie drugiej funkcji sąsiedztwa, tzn. meksykańskiego kapelusza. Zacznijmy od siatki $M = 2, N = 4$ i learning_rate równego 0.5. Tabelka poniżej pokazuje uzyskiwane przy różnych ziarnach losowości różne wyniki accuracy po odpowiedniu 10-ciu i 20-tu epokach.

lp.	accuracy po 10 epokach	accuracy po 20 epokach
1	0.98	0.97
2	0.97	0.97
3	0.98	0.97
4	0.98	0.98
5	0.97	0.97
6	0.97	0.97
7	0.97	0.98
8	0.97	0.96
9	0.97	0.98
10	0.97	0.97
11	0.96	0.98
12	0.98	0.98
13	0.98	0.98
14	0.98	0.97
15	0.98	0.98
16	0.98	0.97
17	0.97	0.97
18	0.97	0.97
19	0.97	0.97
20	0.97	0.98

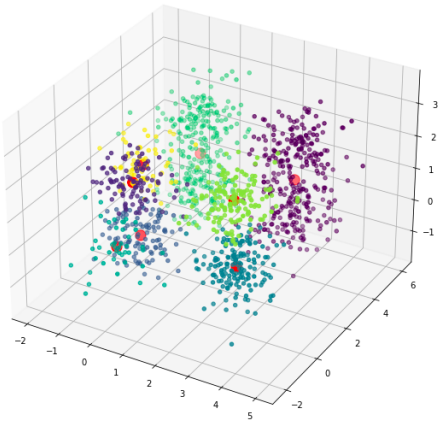
Tabela 3: Uzyskane wartości accuracy podczas testowania sieci

Widzimy, że uzyskiwane wartości accuracy są lepsze niż w przypadku funkcji gaussa. Czyli funkcja meksykański kapelusz lepiej zadziałała dla danych 3D.

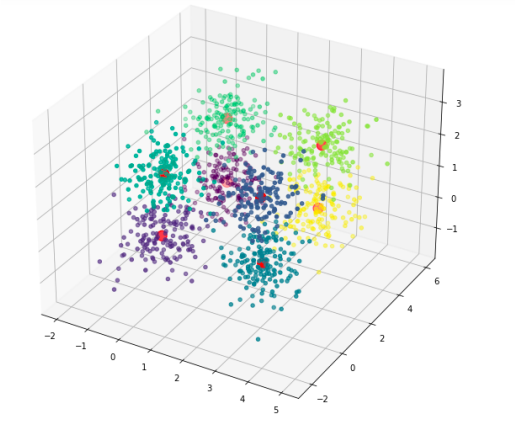
Następnie ponownie przetestujemy działanie dla różnych learning_rate i pokażemy wizualizacje uzyskiwanych wyników.



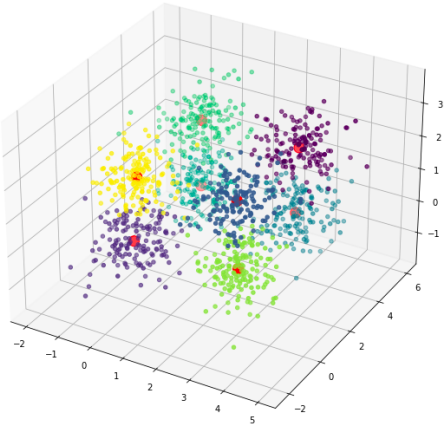
(a) Działanie sieci przy learning_rate=0.01



(b) Działanie sieci przy learning_rate=0.1



(c) Działanie sieci przy learning_rate=0.5



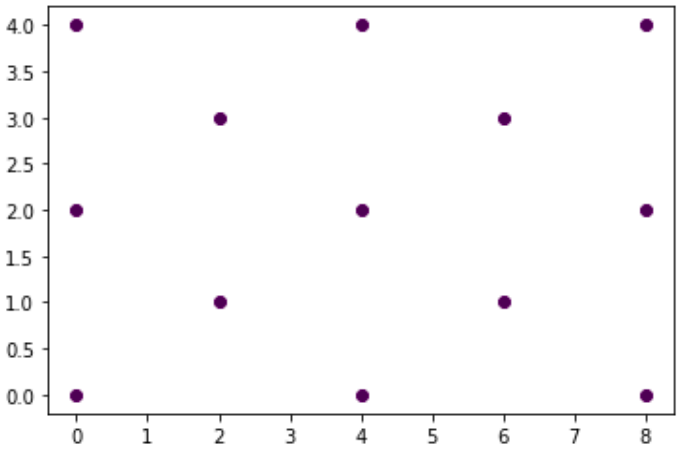
(d) Działanie sieci przy learning_rate=1

Rysunek 13: Działanie sieci Kohonena przy różnych wartościach learning_rate na zbiorze Cube

Otrzymujemy analogiczną sytuację, jak w przypadku używania funkcji gaussowskiej. Ponownie dla większych learning_rate uzyskaliśmy w przybliżeniu oryginalne klastrowanie, przy czym po wcześniej policzonym accuracy wiemy, że prawie idealnie odwzorowane.

2 Sieć Kohonena na siatce sześciokątnej

Podstawową implementację sieci Kohonena rozszerzyliśmy o możliwość ułożenia neuronów na siatce sześciokątnej. Poniżej przedstawiony jest przykład rozłożenia neuronów na takiej siatce.



Rysunek 14: Siatka sześciokątna

2.1 Zbiór MNIST

Zbiór MNIST zawiera macierze 28x28 odpowiadające obrazkom przedstawiającym liczby. Sprawdzimy, czy nasza sieć Kohonena na siatce sześciokątnej poradzi sobie z klastrowaniem takich danych. W tym celu tworzymy siatkę z 10-cioma neuronami, czyli tyloma ile jest cyfr. Następnie używamy sieci Kohonena do klastrowania i za pomocą funkcji znajdującej dla punktu najbliższy neuron dokonujemy klasyfikacji. Otrzymaną klasyfikację przekształcamy na accuracy, tzn. za pomocą napisanej funkcji rezerwujemy zajęte neurony przez największą liczbę punktów z prawdziwej klasy i w ten sposób odnajdujemy powiązanie pomiędzy stworzonymi klastrami, a prawdziwym podziałem zbioru na cyfry. Można powiedzieć, że klastry klasyfikujemy w taki sposób, aby maksymalizować metrykę accuracy. Podobnie liczyliśmy to już wcześniej dla zbiorów hexagon i cube.

lp.	accuracy po 10 epokach	accuracy po 20 epokach
1	0.72	0.77
2	0.71	0.73
3	0.72	0.77
4	0.72	0.77
5	0.74	0.77
6	0.72	0.77
7	0.69	0.75
8	0.72	0.77
9	0.72	0.77
10	0.71	0.75
11	0.71	0.73
12	0.72	0.77
13	0.72	0.77
14	0.71	0.75
15	0.72	0.79
16	0.74	0.79
17	0.72	0.77
18	0.75	0.81
19	0.72	0.77
20	0.71	0.73

Tabela 4: Uzyskane wartości accuracy podczas testowania sieci

Powyższa tabela to uzyskiwane wartości accuracy przy użyciu funkcji gaussowskiej. Widzimy zatem, że sieć niezależnie od ziarna losowości radzi sobie całkiem dobrze z klasteryzacją. Uśredniona wartość accuracy wyniosła 0.72. Nie prezentujemy już tabeli dla funkcji meksykański kapelusz, ponieważ wartości dla niej wyszły bardzo podobne, a uśrednione accuracy otrzymaliśmy dokładnie takie samo, tj. 0.72.

2.2 Human Activity Recognition Using Smartphones

Zbiór Human Activity Recognition Using Smartphones zawierał dane zebrane za pomocą smartfonów, łącznie 561 kolumn danych treningowych. Analogicznie przedstawimy tabele otrzymywanych accuracy. Ze względu na ponownie duże podobieństwa otrzymywanych wyników zmniejszymy liczbę uruchomień kodu do 10.

lp.	accuracy po 10 epokach	accuracy po 20 epokach
1	0.85	0.85
2	0.85	0.85
3	0.84	0.85
4	0.85	0.85
5	0.84	0.85
6	0.84	0.85
7	0.85	0.85
8	0.83	0.84
9	0.84	0.85
10	0.84	0.85

Tabela 5: Uzyskane wartości accuracy podczas testowania sieci z funkcją gaussowską

Końcowo uśrednione accuracy dla funkcji gaussowskiej jest równe 0.85. Poniżej jeszcze tabela dla drugiej funckji sąsiedztwa - meksykańskiego kapelusza.

lp.	accuracy po 10 epokach	accuracy po 20 epokach
1	0.84	0.84
2	0.85	0.85
3	0.85	0.85
4	0.84	0.85
5	0.84	0.85
6	0.83	0.84
7	0.84	0.85
8	0.83	0.84
9	0.84	0.85
10	0.84	0.85

Tabela 6: Uzyskane wartości accuracy podczas testowania sieci z funkcją meksykański kapelusz

W przypadku tej funkcji także otrzymujemy uśrednioną wartość accuracy równą 0.85.

3 Podsumowanie i wnioski

Podsumowując, na wszystkich używanych do testów zbiorach udawało się otrzymywać dobre wyniki. Korzystaliśmy z dwóch funkcji sąsiedztwa: gaussowskiej oraz meksykańskiego kapelusza. Dla obu działanie sieci Kohonena było niemalże identyczne, z taką różnicą, że dla meksykańskiego kapelusza należało dobrać odpowiednio małą wartość szerokości sąsiedztwa. Oprócz tego, obserwowaliśmy także, że na siatce prostokątnej często zamiana parametrów M z N nie wpływała na zmianę klasteryzacji. Co więcej, dla wszystkich testów, o ile $M \cdot N$ było równe prawdziwej liczbie klas, to otrzymywaliśmy po pewnej liczbie epok taką samą klasteryzację. Także learning_rate nie wpływał mocno na działanie sieci, za wyjątkiem sytuacji ze zbiorem cube, gdy zbyt mały learning_rate dzielił zbiory lekko inaczej niż były one podzielone oryginalnie. Siatka sześciokątna w sieci Kohonena pozwoliła uzyskać dobre klasteryzacje dla zbiorów o dużej liczbie kolumn, co potwierdzało uzyskiwane

accuracy.