Sprawozdanie z realizacji zadania

Sieci Kohonena

Tymoteusz Kwieciński

320637

Kwiecień, Maj 2024

1 Cel zadania

Głównym celem tego zadania było zapoznanie się z budową jednego z pierwszych modeli sieci neuronowej - sieci Kohonena, w literaturze anglojęzycznej zwanej również jako *Self-Organizing Maps*. Model ten został opracowany przez Teuvo Kohonena w latach 80 XX wieku. [2]. To dość krótkie zadanie zostało podzielone na 2 części. W pierwszej części należało zaimplementować sieć od podstaw oraz przetestować jej działanie na bardzo prostych zbiorach. Następnie działanie sieci należało przetestować na zaawansowanych zbiorach, takich jak MNIST.

2 Opis implementacji sieci Kohonena

2.1 Szczegóły implementacji

2.1.1 Funkcja sąsiedztwa

W ramach mojej implementacji wykorzystałem 4 różne funkcje sąsiedztwa

1. Funkcja Gaussowska:
$$f(d,t) = e^{-\frac{d^2}{2\sigma(t)^2}}$$
2. Minus druga pochodna funkcji Gaussowskiej:
$$f(d,t) = -e^{-\frac{d^2}{2\sigma(t)^2}} \cdot (\frac{d^2}{\sigma(t)^2} - 1)$$
3. Funkcja Koła:
$$f(d,t) = \mathbb{1}_{|d| \leqslant \sigma(t)}$$
4. Funkcja Meksykańskiego Kapelusza
$$f(d,t) = (2 - 4d^2) \cdot e^{-d^2}$$

Zgodnie ze wskazówkami w [2], użyte funkcje sąsiedztwa są wygaszane w czasie za pomocą parametru $\sigma(t)$, gdzie t to aktualna iteracja treningu.

W przypadku minus drugiej pochodnej funkcji Gaussowskiej, aby zachować skalę i zapobiec eksplodowaniu wartości wag sieci do nieskończoności pomnożyłem wartość funkcji przez $\sigma(t)^2$. Właściwie obliczona wartość tej funkcji powinna wynosić:
$$f(d,t) = -e^{-\frac{d^2}{2\sigma(t)^2}} \cdot (\frac{d^2}{\sigma(t)^4} - \frac{1}{\sigma(t)^2})$$
.

W ramach eksperymentów nie porównywałem jednak dwóch ostatnich funkcji sąsiedztwa, zgodnie z poleceniem skupiłem się na funkcji gaussowskiej oraz minus drugiej pochodniej funkcji gaussowskiej.

2.1.2 Szerokość sąsiedztwa

Orginalny artykuł [2], w którym wykorzystywana była gaussowska funkcja sąsiedztwa, wspomniane było, aby wygaszać funkcję sąsiedztwa w czasie na podstawie pewnej funkcji malejącej. Na podstawie [1] zastosowałem funkcję wygaszania w czasie postaci:

$$\sigma(t) = \sigma_0 e^{-t/\Lambda}$$

gdzie Λ to stała dla treningu wynosząca $\Lambda = \frac{T}{\log(\sigma_0)}$, a T to łączna liczba epok w treningu.

W obecnej implementacji, można również zrezygnować z zmniejszającej się wartości szerokości sąsiedztwa i pozostawić wartość szerokości sąsiedztwa na stałą, równą początkowemu parametrowi.

2.1.3 *Learning rate*

Kolejnym parametrem, który pozwala wygaszać intensywność sygnałów pod koniec treningu to *Learning rate*, wspomniany (ale bez konkretnej nazwy) w artykule [2]. W implementacji skorzystałem ze zwykłej funkcji *exponential decay* zadanej wzorem

$$\alpha(t) = e^{-\frac{t}{\lambda}}$$

gdzie λ to hiperparametr.

2.1.4 Sposób podziału na klastry

Sieć Kohonena ma niezwykłą własność podziału zbioru wejściowego na klastry. Aby wyznaczyć klastry w najbardziej naiwny sposób, po skończonym treningu każdy element zbioru danych jest mapowany na jeden, najbliższy (BMU) neron z sieci. W ten sposób każdy element zbioru wejściowego zostaje przyporządkowany do oddzielnego klastra. Co warto zauważyć, liczba klastrów do podziału jest równa liczbie neuronów w sieci.

W literaturze udało znaleźć mi się podejścia [3], które wykorzystywały embedding wygenerowany przez sieć Kohonena jako wejście do innego, często prostszego algorytmu klastrującego, który mógłby wykorzystywać położenie na siatce oraz wartości wag neuronów. Ten mechanizm klastrowania wykorzystałem w realizacji drugiego zadania. W trakcie pierwszego używałem jedynie naiwne podejście - rozwiązanie wykorzystujące neurony sieci jako ostateczne klastry.

2.1.5 Organizacja neuronów w sieci

W mojej implementacji wykorzystałem dwa spoosby organizacji neuronów w sieci.

1. Siatka prostokątna
2. Siatka sześciokątna

Dla przejrzystości, odległość pomiędzy sąsiadującymi neuronami została ustawiona na 1. To znaczy, każdy neuron w siatce kwadratowej miał współrzędne równe:

$$(i, j)$$

dla dowolnych wartości $i \in \{1, 2, \ldots, M\}, j \in \{1, 2, \ldots, N\} \in$, zaś w przypadku siatki sześciokątnej, każdy neuron miał współrzędne

$$(i\frac{\sqrt{3}}{2}, j + (i \pmod 2) \cdot \frac{1}{2})$$

dla dowolnych wartości $i \in \{1, 2, \ldots, M\}, j \in \{1, 2, \ldots, N\}$

3 Realizacja zadania 1 i wyniki eksperymentów

3.1 Opis zadania

W ramach pierwszego zadania należało zaimplementować podstawową wersję sieci Kohonena od podstaw, a także przetestować ją na 2 zadanych zbiorach danych.
Wymagania, które należało spełnić, to organizacja neuronów w prostokątnej siatce $M \times N$, gdzie M, N to wejściowe parametry podawane przez użytkownika. Sieć powinna działać dla zbioru wektorów o tej samej długości (wejściowy zbiór danych).
Należało uwzględnić funkcje sąsiedztwa, takie jak:

- funkcję gaussowską
- minus druga pochodna funkcji gaussowskiej

W instrukcji podane było również wymaganie dotyczące zmiany szerokości sąsiedztwa z użyciem parametru z przedziału $[0.1, 1]$. W przypadku mojego rozwiązania sprowadzało się to do zmiany początkowej wartości szerokości sąsiedztwa. Dodatkowym hiperparametrem uczenia, był parametr α określający szybkość wygaszania uczenia. Parametr α to odpowiednik *learning rate*, który zmniejszał wpływ końcowych epok na wynik treningu.

W ramach zadania 1 sieć miała zostać przetestowana na dwóch dostarczonych zbiorach danych - dwuwymiarowym oraz trójwymiarowym zbiorze, które zawierały opisane punkty w wierzchołkach odpowiednio sześciokąta i sześcianu.

Aby zweryfikować jakość sieci oraz implementacji przeprowadzone testy miały odpowiedzieć na pytania:

- *Czy klastry w odwzorowaniu znalezionym przez sieć pokrywają się w liczbą klastrów w faktycznych danych?*
- *Czy znalezione klastry pokrywają się z identyfikatorami wierzchołków?*

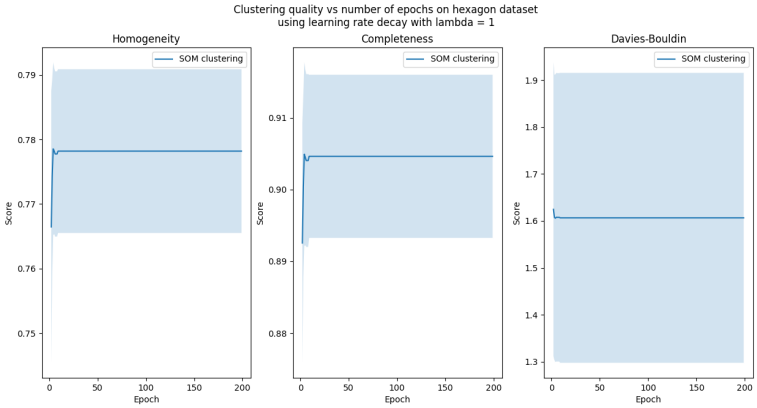
W celu znalezienia odpowiedzi na pierwsze pytanie przeprowadziłem trening sieci z różną liczbą neuronów, a następnie za pomocą miary jakości klastrowania (współczynnik *Davies–Bouldin*'a) [3] zweryfikowałem, przy jakiej liczbie klastrów podział jest najlepszy.

Aby odpowiedzieć na drugie pytanie, przetestowałem zaimplementowaną sieć na zaproponowanych danych oraz wykonałem podział na klastry, przy użyciu rzeczywistej liczby klastrów. Jakość pokrycia weryfikowałem za pomocą miar *Homogenity* oraz *Completeness*, które opisują odpowiednio jak bardzo jednolite są stworzone klastry oraz czy stworzone klastry zawierają wszystkie elementy rzeczywistych klastrów. Wszystkie eksperymenty w ramach tego zadania przeprowadzałem na 10 niezależnych uruchomieniach algorytmu. W tym zadaniu nie wykonywałem podziału zbioru na treningowy i testowy, ze względu na nienadzorowany sposób uczenia testowanego algorytmu.

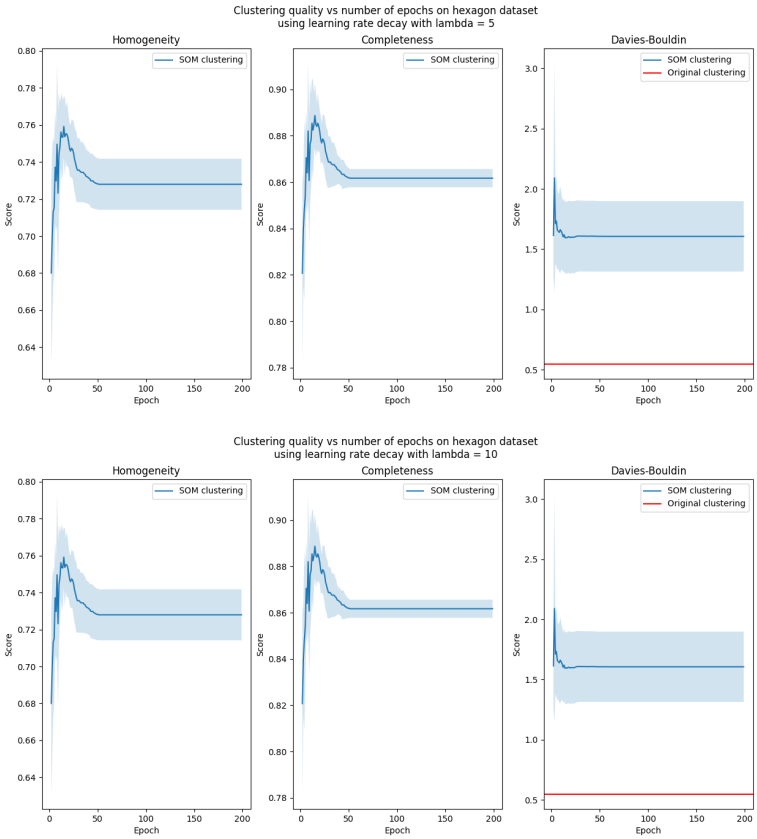
3.2 Wyniki eksperymentów

3.2.1 Długość treningu

Początkowo zweryfikowałem liczbę epok na której powinien odbywać się trening. Przy użyciu domyślnych hiperparametrów ($\sigma_0 = 0.5, \lambda = 1, \alpha_0 = 1$) przetestowałem liczbę epok potrzebną do zbiegnięcia algorytmu. Wyniki porównania znajdują się na 1. Widać, że trening stabilizuje się dość szybko. Jak zauważyłem jednak, stabilizacja następuje szybko z powodu wybranych hiperparametrów. W przypadku zwiększenia parametru λ jak widać na 2 trening zatrzymuje się później, co negatywnie odbija się na wyniku. Podobna sytuacja wystąpiła w przypadku zbioru *cube*, co widać na 3. Pozostałe hiperparametry nie wpływają na długość treningu.



Rysunek 1: Wyniki metryk dotyczące klastrowania dla różnej liczby epok. Jak widać, trening dość szybko się stabilizuje i dalsze epoki nie zmieniają klastrowania, mimo to, że pomiędzy kolejnymi uruchomieniami algorytmu są względnie spore różnice i duży błąd na wykresie

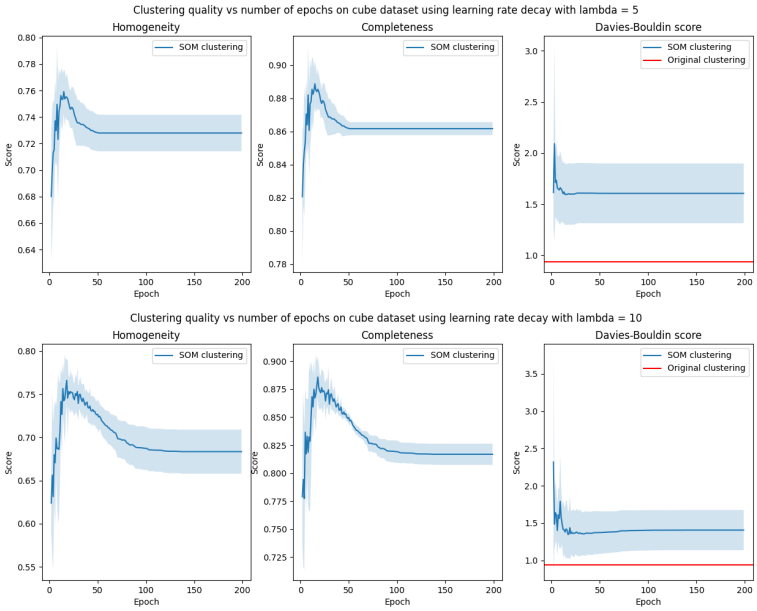


Rysunek 2: Wyniki metryk dotyczące klastrowania dla różnej liczby epok z różnymi wartościami parametru λ. Trening został wygaszony później co negatywnie odbija się na jakości klastrowania.

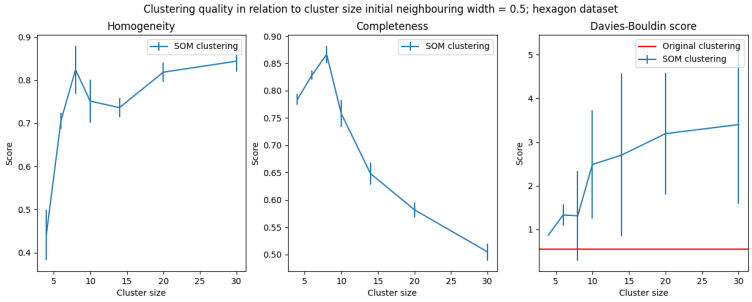
Ostatecznie, mimo rekomendowanej [2] liczbie epok jako $M \cdot N \cdot 500$, gdzie M, N to wymiary sieci, przyjąłem do dalszych testów w ramach tego zadania domyślną liczbę epok jako $M \cdot N \cdot 10$, co pozwoliło mi przetestować większą liczbę przypadków.

3.2.2 Liczba klastrów

Aby zweryfikować hipotezę, czy sieć prawidłowo wybiera liczbę klastrów przeprowadziłem testy jakości klastrowania dla różnych rozmiarów sieci. Jak wspomniane wyżej, przyjąłem jako liczbę klastrów liczbę neuronów sieci, w związku z tym w tej części eksperymentów badałem wpływ rozmiaru sieci na jakość klastrowania w zbiorach *cube* oraz *hexagon*.



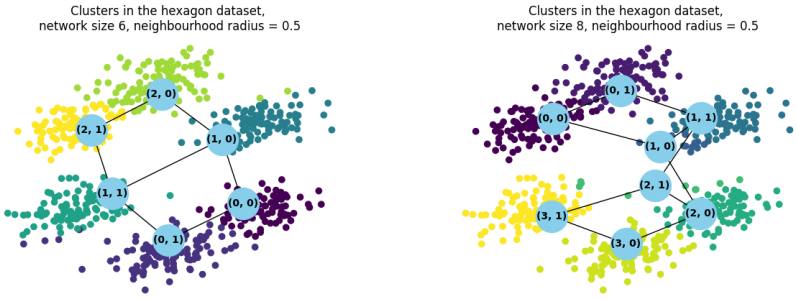
Rysunek 3: Wyniki metryk dotyczące klastrowania dla różnej liczby epok dla zbioru *cube*. Zwiększanie parametru λ wydłuża trening, ale zmniejsza jakość klastrowania



Rysunek 4: Wyniki metryk dotyczące klastrowania dla różnej liczby klastrów. Najelpsze wyniki wydaje się osiągać sieć przy rozmiarze 8 (2×4)

Zaobserwowałem zależność, że dla zbioru *hexagon*, lepszą jakość klastrowania prezentują sieci z 8, a nie 6 neuronami. Problem z jakością klastrów o rozmiarze 6 dla zbioru *hexagon* być może jest spowodowany ściągającym efektem siatki do prostokątnego kształtu, który jest bardziej widoczny przy większej wartości współczynnika sąsiedztwa.

Przy zmniejszonym współczynniku sąsiedztwa, klastry na zbiorze *hexagon* lepiej pokrywają centra klastrów, co widać na porównaniach 5 i 6



Rysunek 5: Porównanie wyników klastrowania dla sieci o najlepszych wynikach - rozmiary 6 i 8

W przypadku zbioru *cube* najlepsze wyniki przy szerokości sąsiedztwa 0.5 zostały osiągnięte dla 8 klastrów, przy szerokości 0.25, te wyniki nie były już aż tak jednoznaczne i jakość reportowanego klastrowania różniła się, co widać na 9

3.2.3 Współczynnik sąsiedztwa

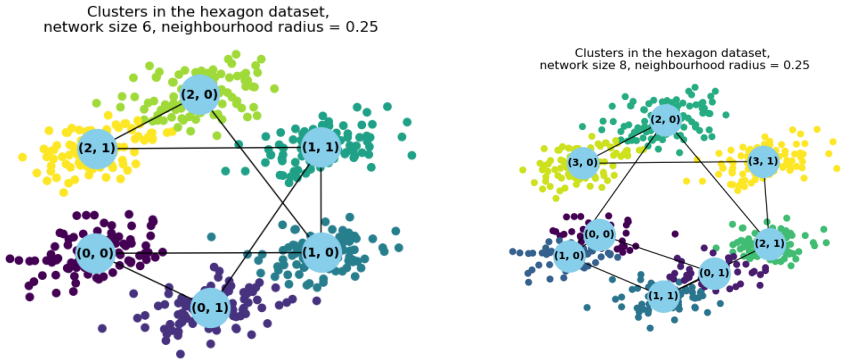
Aby zweryfikować działanie sieci w zależności od szerokości początkowego współczynnika sąsiedztwa wykonałem podobną analizę jak poprzednio, badając wartości z przedziału $[0.1, 1]$. Wyniki analizy dla zbioru *hexagon* znajdują się na 10. Jak widać, jakość klastrowania jest lepsza dla mniejszych początkowych szerokości sąsiedztwa, osiągając najlepsze wyniki dla szerokości 0.2. W przypadku zbioru *cube*, wyniki są podobne, najlepsze rezultaty są otrzymywane dla małej początkowej szerokości sąsiedztwa, wyniki porównania widać na 11

3.2.4 Funkcja sąsiedztwa

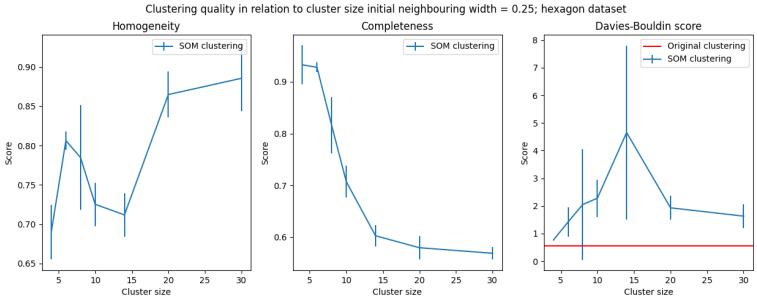
Zweryfikowałem dodatkowo jaki wpływ ma rodzaj funkcji sąsiedztwa wykorzystywanej do treningu sieci. Okazuje się, że minus druga pochodna funkcji Gaussowskiej osiąga najlepsze rezultaty, gdy zmniejszone zostaną parametry *learning rate*. Przykładowe porównanie znajduje się na 12. Okazuje się, że Gaussowska funkcja sąsiedztwa sprawuje się odrobinę lepiej i jest mniej wrażliwa na zmianę hiperparametrów.

3.2.5 Wnioski

W trakcie pierwszego zadania z tej części laboratoriów pomyślnie udało mi się zaimplementować Samoorganizującą Sieć Kohonena. Zapoznałem się z jej działaniem i wykonałem testy jej działania przy użyciu różnych funkcji sąsiedztwa i innych hiperparametrów.



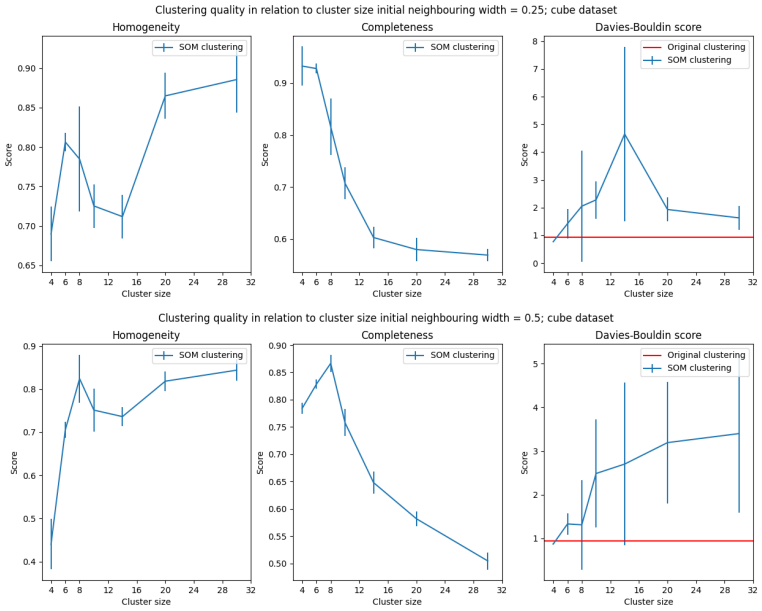
Rysunek 6: Porównanie wyników klastrowania dla sieci o najlepszych wynikach - rozmiary 6 i 8 przy szerokości sąsiedztwa 0.25



Rysunek 7: Wyniki metryk dotyczące klastrowania dla różnej liczby klastrów przy mniejszym początkowym współczynniku sąsiedztwa - 0.25. Sieć otrzymuje lepsze wyniki dla mniejszych rozmiarów

Okazuje się, że w przypadku testowanych zbiorów danych, trening zbiegał bardzo szybko, jego wydłużenie za pomocą dostosowania hiperparametrów związanych z *learning rate* nawet pogarszało wyniki sieci. Modyfikacje parametru α miały sens przy funkcji sąsiedztwa minus drugiej pochodnej funkcji gaussowskiej, gdyż przy domyślnych wartościach, wagi sieci eksplodowały.

Moja implementacja sieci Kohonena raczej poprawnie wykrywała rzeczywistą liczbę klastrów w danych, o ile w przypadku datasetu *hexagon* przejawiała tendencje do zawyżania jej liczby, to otrzymane wartości były bardzo zbliżone do rzeczywistych. Na podstawie przeprowadzonych eksperymentów można wysnuć wniosek, że klastrowanie danych za pomocą sieci Kohonena w sposób taki, jak opisany wyżej sprawuje się ogólnie dość dobrze.



Rysunek 8: Wyniki metryk dotyczące klastrowania dla różnej liczby klastrów dla datasetu *cube*

4 Realizacja zadania 2 i wyniki eksperymentów

4.0.1 Opis zadania

W ramach tego zadania należało przetestować działanie sieci na dwóch rzeczywistych zbiorach danych:

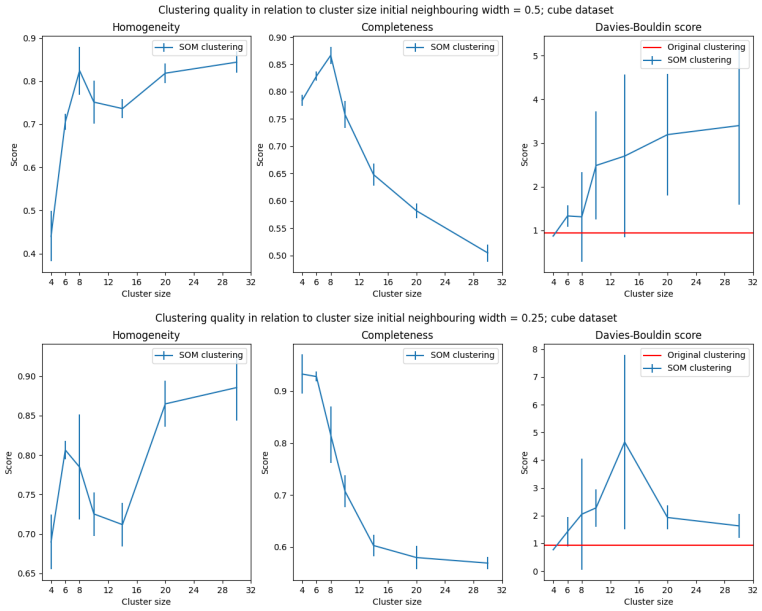
1. MNIST
2. Human Activity Recognition Using Smartphones

Dodatkowo, należało dodać implementacje heksagonalnej sieci sąsiedztwa, co również uczyniłem.

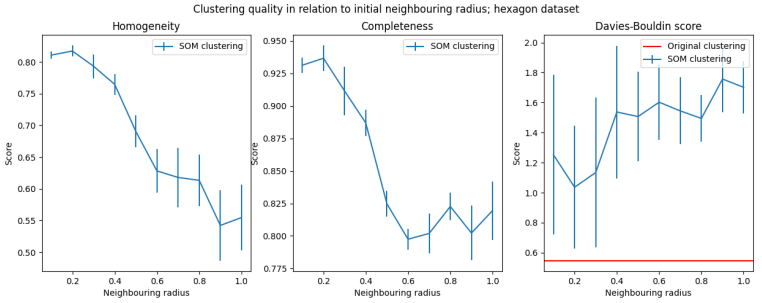
Ponieważ badane zbiory miały znacznie więcej wymiarów niż poprzednie, sztuczne datasety, tym razem zastosowałem inne podejście do klastrowania. Na zadanym zbiorze wykonałem *nienadzorowany* trening sieci Kohonena, a następnie otrzymane mapowanie sklastrowałem przy użyciu algorytmu KMeans i przepuściłem przez klasyfikator lasu losowego. Reprezentacja ukryta generowana przez sieć to mapowanie elementów ze zbioru danych na neurony sieci.

4.0.2 Podjęte kroki i wyniki eksperymentów

Podobnie jak poprzednio jakość klastrowania badałem za pomocą 2 metryk - *Homogenity* oraz *Completeness* (zrezygnowałem z metryki wykorzystujących



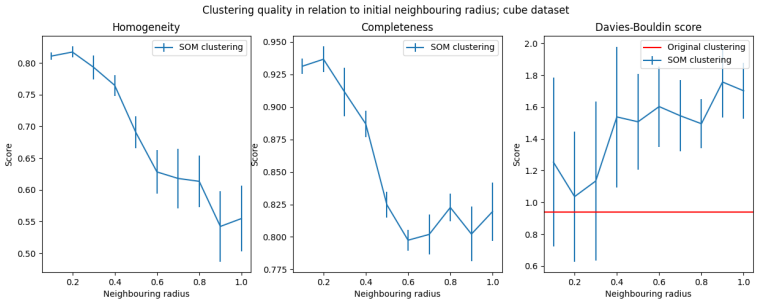
Rysunek 9: Wyniki metryk dotyczące klastrowania dla różnej liczby klastrów dla zbioru *cube*



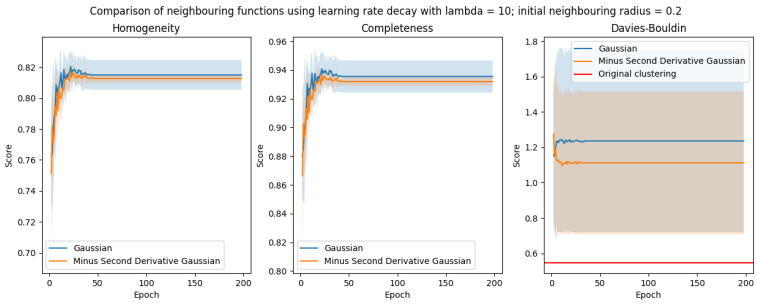
Rysunek 10: Wyniki metryk dotyczące klastrowania dla różnej szerokości sąsiedztwa dla zbioru *hexagon*

pozycje punktów w przestrzeni, gdyż mogłoby to bardzo negatywnie wpłynąć na czas obliczeń, a prawdopodobnie wyniki uzyskane przy jej użyciu byłyby mocno skorelowane z wynikami uzyskanymi przy użyciu pozostałych metryk).

W tym zadaniu zastosowałem podział zbioru na treningowy i testowy. Za pomocą 5 warstwowej kroswalidacji na każdej warstwie trenowana była sieć, na jej wynikach ze zbioru treningowego algorytm klastrowania, a następnie wyliczałem jakość klastrowania za pomocą wspomnianych wyżej metryk na testowej części warstwy. Analogicznie postąpiłem podczas ewaluacji jakości mapowania



Rysunek 11: Wyniki metryk dotyczące klastrowania dla różnej szerokości sąsiedztwa dla zbioru *cube*



Rysunek 12: Porównanie procesu uczenia się dla różnych funkcji sąsiedztwa. Gausowska funkcja sąsiedztwa osiąga lepsze wyniki porównując do *ground-truth*, ale jakość klastrowania na podstawie miary *Davieas-Bouldin*'a jest odrobinę gorsza

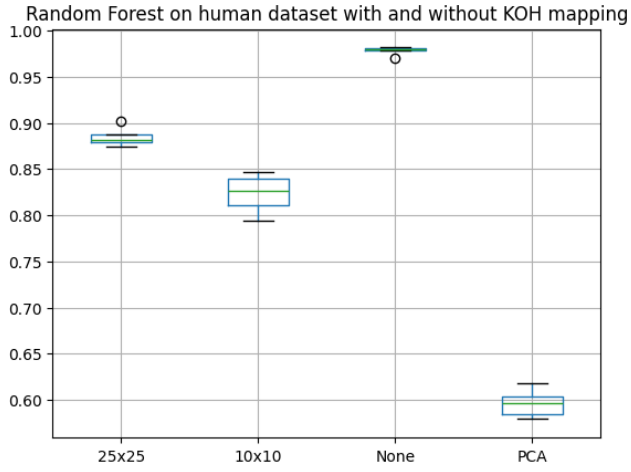
przy użyciu lasu losowego - na mapowaniach zwróconych przez sieć wytrenowałem las losowy używając wartości *ground-truth*, a jego jakość była oceniana po prostu za pomocą *accuracy*. Ponieważ to zadanie było dużo bardziej kosztowne pod względem zasobów oraz czasu, a zależało mi na wykonaniu wielu prób w ramach kroswalidacji, zdecydowałem się kończyć trening już po 30 epokach sieci.

Aby zweryfikować działanie sieci, wykonałem również klastrowanie tylko za pomocą KMeans oraz klasyfikację przy użyciu Random Forest na danych wejściowych, bez mapowania siecią Kohonena w podobny sposób jak opisano powyżej, a także na danych wejściowych po redukcji wymiarowości przy użyciu PCA do 2 wymiarów.

4.0.3 Działanie sieci

W przypadku zbioru *human activity* modele wykorzystujące mapowanie sieci Kohonena sprawdziły się znacznie gorzej, niż te trenowane na oryginalnych danych. Wyniki tego podsumowania znajdują się na 13. W przypadku zbioru

danych MNIST, te różnice nie były aż takie duże 14, prawdopodobnie dalsze zwiększanie rozmiaru sieci lub dłuższy trening przyniósłby o wiele lepsze rezultaty. Mimo to, oba zbiory były na tyle proste, że las losowy dużo lepiej poradził sobie z zadaniem klasyfikacji niż klasyfikator uruchomiony na nienadzorowanej sieci.



Rysunek 13: Klasyfikator wytrenowany na mapowaniu zwróconym przez sieć działa gorzej niż wytrenowany na oryginalnych danych, ale znacznie lepiej niż na mapowaniu zwróconym przez PCA - zbiór *human activity*

4.0.4 Funkcja sąsiedztwa

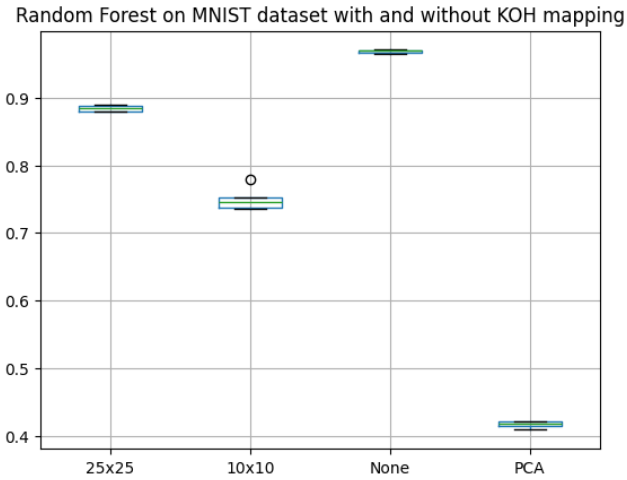
Podobnie jak w poprzednim zadaniu, również tym razem porównałem działanie sieci wykorzystując różne funkcje sąsiedztwa. Okazało się, że na zbiorze *human activity* znacznie lepsze wyniki są osiągane w przypadku zastosowania funkcji gaussowskiej, prawdopodobnie jak w poprzednim zadaniu, z uwagi na wrażliwość minus drugiej pochodnej funkcji gaussowskiej na hiperparametry.

4.0.5 Sposób organizacji sieci

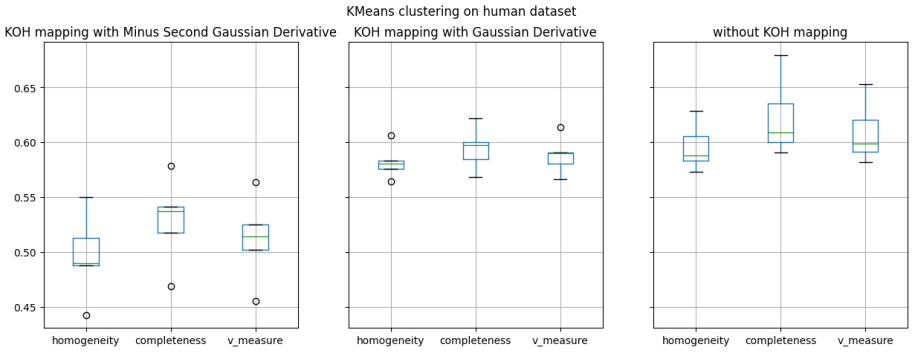
Jak się okazało, sposób organizacji sieci nie ma zbyt dużego wpływu na wynik klastrowania i klasyfikatora. W przypadku zbioru *human activity* jak widać na 16 wyniki osiągane przez sieć zorganizowaną w prostokątną mapę są nieznacznie lepsze od mapy prostokątnej.

4.0.6 Wnioski

Sieci Kohonena są w stanie dosyć sprawnie zredukowość wielowymiarowość dużych zbiorów danych. O ile klasyfikacja i klasteryzacja korzystająca z mapowania zwróconego przez sieć jest gorszej jakości niż ta na oryginalnych danych, to i

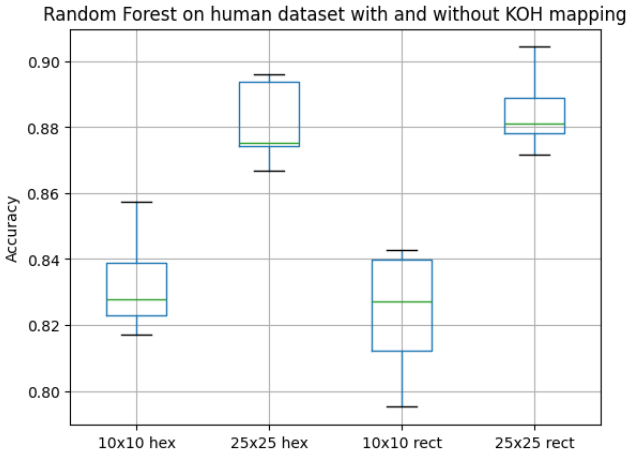


Rysunek 14: Klasyfikator wytrenowany na mapowaniu zwróconym przez sieć działa trochę gorzej niż wytrenowany na oryginalnych danych, ale zdecydowanie lepiej niż na mapowaniu zwróconym przez PCA - zbiór MNIST
. Klasyfikator wytrenowany na mapowaniu PCA jest gorszy niż losowy klasyfikator.



Rysunek 15: Klasteryzacja wyternowana na mapowaniu zwróconym przez sieć z minus drugą pochodną funkcji gaussowskiej działa gorzej niż ta wytrenowana na mapowaniu z funkcją gaussowską lub bez mapowania

tak jest jest satysfakcjonująca z uwagi na ogromną redukcję wymiarowości. Z przeprowadzonych wyżej testów wynika, że sieć Kohonena w przypadku naszych danych najlepiej sprawuje się, gdy jest większego rozmiaru, a funkcja sąsiedztwa to funkcja gaussowska. Ułożenie neuronów nie ma aż tak dużego wpływu na wyniki sieci. Wizualnie, sieć dobrze generuje mapowanie, co możemy zaobserwować na 17



Rysunek 16: Wytrenowany las losowy radzi sobie odrobinę lepiej przy prostokątnej mapie, zwiększenie rozmiaru sieci poprawia rezultaty klasyfikatora

5 Podsumowanie

W trakcie tego krótkiego projektu udało mi się zapoznać i zaimplementować jedną z pierwszych architektur z działu uczenia głębokiego - sieć Kohonena. Było dla mnie zaskakujące, że tak oryginalny pomysł nie uzyskał większego rozgłosu, podobnego do sieci MLP, ale mimo to jest efektywnie wykorzystywany przez specjalistów.

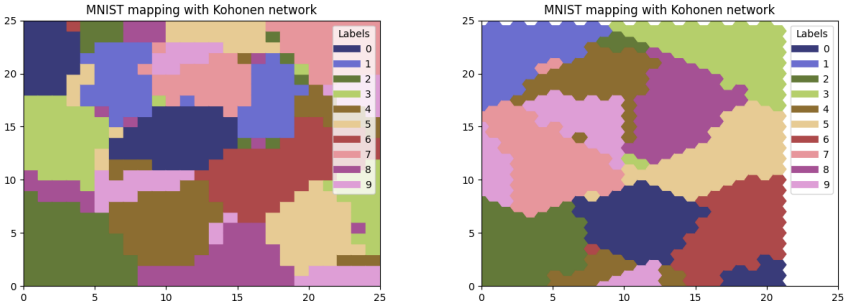
Najbardziej fascynujące dla mnie jest to, że taka sieć jest w stanie wyodrębnić wartościowe informacje z wielowymiarowych danych w dość szybki sposób, co przypomina działanie popularnych w dzisiejszych czasach *foundational models*. Mapowanie uzyskane przez sieć Kohonena, może być z powodzeniem wykorzystane przez *task-specific* algorytmy, podobnie jak finetunowane modele bazujące na embeddingach generowanych przez duże, przetrenowane na ogromnych danych *foundational models*.

Literatura

[1] Le Anh Tu. Improving feature map quality of som based on adjusting the neighborhood function, 2020.

[2] Teuvo Kohonen. Self-organized formation of topologically correct feature maps, 1982.

[3] J. Vesanto and E. Alhoniemi. Clustering of the self-organizing map, 2000.



Rysunek 17: Wyniki mapowania sieci. Każda figura na wykresie odpowiada jednemu neuronowi sieci. Kolor figury odpowiada za wartość *ground truth* obserwacji ze zbioru MNIST, które zostały przyporządkowane do tego neurona największą liczbę razy.