Algorytmy Genetyczne i sztuczne sieci neuronowe

Budowanie generatora do testowania sieci WTM (ang. SOM (Self-organizing map))

Specyfikacja dotycząca generatora:

- Liczba grup danych = 5
- Rozmiar danych = 2D
- Zakres danych [0:100 ; 0:100]
- Liczba obiektów w danej grupie danych = 10
- Promień każdej grupy = 5

Przykładowy wykres wygenerowanych danych poniżej:

UWAGA!!!: Wyświetlanie danych powinno być znormalizowane do zapisu np. [72, 10; ...]. Każda współrzędna kolejna poprzedzona speratorem ,, ; ''

PS: Mieszanie danych podczas uczenia np. w przypadku alfabetu uczenie (a,b,c,d,...), potem uczenie od (g,h,i,j,...). Różne możliwości????

Importowanie potrzebnych bibliotek

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import random
import math
import csv
```

Zmienne potrzebne do wygenerowania danych

- number_of_groups - mienna określająca ilość grup danych
- radius - zmienna określająca promień okręgu, który posłużyć ma nam jako obszar generowanych obiektów danych
- number_of_object - zmienna mówiąca o ilości obiektów jakie muszą być wygenerowane w określonej grupie danych

In [ ]:
```
number_of_groups = 10    #Number of groups in model
radius = 5               #Circle radius
number_of_object = 10    #Number of object in each group
```

Proces generowania koordynatów dla konkretnych grup

- za pomocą funkcji „zeros" zawartej w bibliotece numpy generowana jest tablica 2D wypełniona zerami, która potem będzie wykorzystana do zapisu koordynatów poszczególnych wygenerowanych grup,
- funkcja „coordinate_generator_group(lst_groups)" przekazuje w argumencie stworzoną wcześniej tablice 2D wypełnioną zerami. Następnie poprzez pętle for przechodzimy po poszczególnych elementach tablicy zapisując w niej wygenerowane punkty x i y w odpowiednim zakresie od 0 do 100. Zapis danych jest poprzez „element[0] & element[1]", które zapisują koordynaty w konkretnych miejscach podtablicy.
- wykorzystując bibliotekę pandas jesteśmy w stanie zapisać dane x i y w dwuwymiarowej strukturze danych, które oznaczone są rzędami i kolumnami. Pełni to funkcję przejrzystego wglądu do generowanych danych.

In [ ]:
```
def coordinate_generator_group(temp_lst):
    for element in temp_lst:
        temp_x_point = random.randint(0,100)    #Generate x and y coordinate for
        temp_y_point = random.randint(0,100)
        element[0] = temp_x_point               #Save points in list
        element[1] = temp_y_point
```

In [ ]:
```
lst_groups = np.zeros((number_of_groups,2))    #Create a list of groups coordina

coordinate_generator_group(lst_groups)    #Function to generate coordinates for

groups = pd.DataFrame(lst_groups, columns =['x', 'y'])        #Load data groups
print(groups)
```

```
     x     y
0  26.0  35.0
1  54.0  56.0
2  39.0  19.0
3  52.0  48.0
4  68.0  48.0
5  76.0  84.0
6  64.0  10.0
7  84.0  26.0
8  50.0  15.0
9  21.0  86.0
```

Proces generowania koordynatów do obiektów konkretnych grup danych

- za pomocą wcześniej wspomnianej funkcji „zeros" generowana jest tablica 2D wypełniona zerami,
- wspomniana wyżej tablica nadpisywana jest poprzez wywołaną funkcję „coordinate_generator_object(lst_objects)". W funkcji na początku został stworzony counter, który pomoże podczas identyfkacji danego elementu w tablicy temp_lst, która zawiera number_of_object*number_of_groups elemenetów. Pierwszy człon pętli iteruje po elementach tablicy zawierającej koordynaty grup, następny człon natomiast będzie generował ilość punktów w grupie zdefiniowanej w zmiennej number_of_object. Następnie przy pomocy wzorów na ogrąg generowane są punkty pod konkretnym kątem w okręgu, który jest grupą oraz zapisywane są w zmiennych

temp_x oraz temp_y. Wygenerowane wcześniej koordynaty umieszczane są w konkretnych elementach tablicy jaki wskazuje nam zmienna coutner. Na koniec zwracana jest tablica wraz z zapisanymi koordynatami.

- również z wykorzystaniem biblioteki pandas jesteśmy w sposób przejrzysty wyświetlić oraz manipulować danaymi.

In [ ]:
```
def coordinate_generator_object(temp_lst):
    counter = 0                                                    #counter ele
    for elements_lst_groups in lst_groups:
        for _ in range(number_of_object):
            r = radius * random.random()
            theta = 2 * math.pi *random.random()

            temp_x = elements_lst_groups[0] + r * math.cos(theta)   #Generate x
            temp_y = elements_lst_groups[1] + r * math.sin(theta)

            #print('x', temp_x, 'y', temp_y)
            temp_lst[counter][0] = temp_x
            temp_lst[counter][1] = temp_y
            counter+=1

    return temp_lst
```

In [ ]:
```
lst_objects = np.zeros((number_of_object*number_of_groups, 2))     #List of obje

lst_objects = coordinate_generator_object(lst_objects)

objects = pd.DataFrame(lst_objects, columns =['x', 'y'])        #Load data group
print(objects)
```

```
           x          y
0   26.467184  32.977646
1   29.231520  34.855235
2   26.097313  34.775430
3   23.889346  36.926630
4   26.030078  38.204176
..        ...        ...
95  20.887604  86.107620
96  24.650175  85.141824
97  18.839240  90.286078
98  22.628294  84.851599
99  18.250570  88.257072

[100 rows x 2 columns]
```

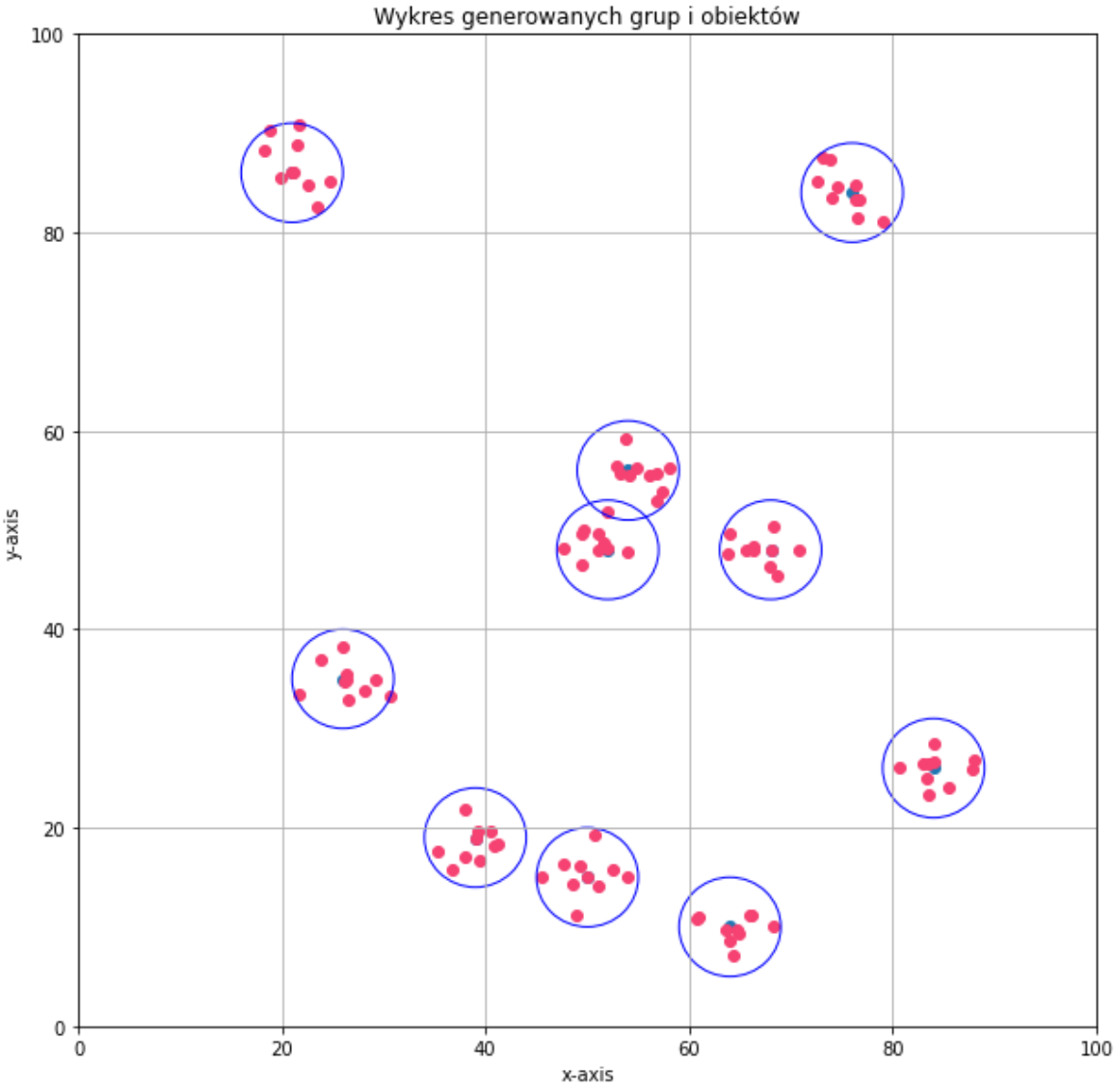
Proces wyświetlania wygenerowanych danych na diagramie z użyciem biblioteki matplotlib

- Funkcja createCircles(item, radius), do której przekazywane są konkretne koordynaty wygenerowanych grup danych oraz promień koła wykorzystywane są do określenia środka oraz promienia rysowanego okręgu.

In [ ]:
```
def  createCircles(centre, radius):
    return plt.Circle((centre[0], centre[1]), radius, color='b', fill=False)
```

In [ ]:
```
# Creating a new figure
fig, ax = plt.subplots(figsize=(10,10))

# Numpy array as scatter plot
plt.scatter(x=groups['x'],y=groups['y'])
plt.scatter(x=objects['x'],y=objects['y'], color= '#f54272')

print(len(lst_groups))

for item in lst_groups:
    ax.add_patch(createCircles(item, radius))

# Adding details to the plot
plt.title('Wykres generowanych grup i obiektów')
plt.xlabel('x-axis')
plt.ylabel('y-axis')

# Displaying the plot
plt.xlim(0,100)
plt.ylim(0,100)
plt.grid()
plt.show()
```

```
10
```



Zapisywanie danych do pliku w celu wczytania go w programie Kohenen

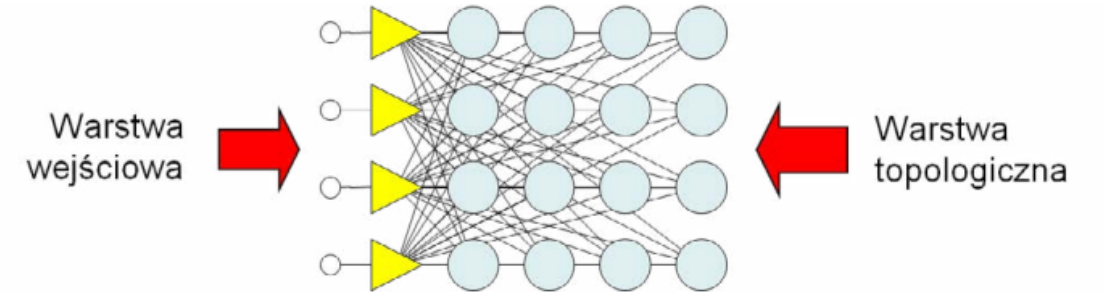
Mean quantization error = 9.9677, topographic error = 0.96364

In [ ]:
```
with open('data_10_temp.dat', 'w', newline='') as f:
    csv.writer(f, delimiter=' ').writerows(lst_groups)
    csv.writer(f, delimiter=' ').writerows(lst_objects)
```

Teoria sieci oraz analiza danych wprowadzonych w programie Kohenen

Neuronowa sieć Kohenena

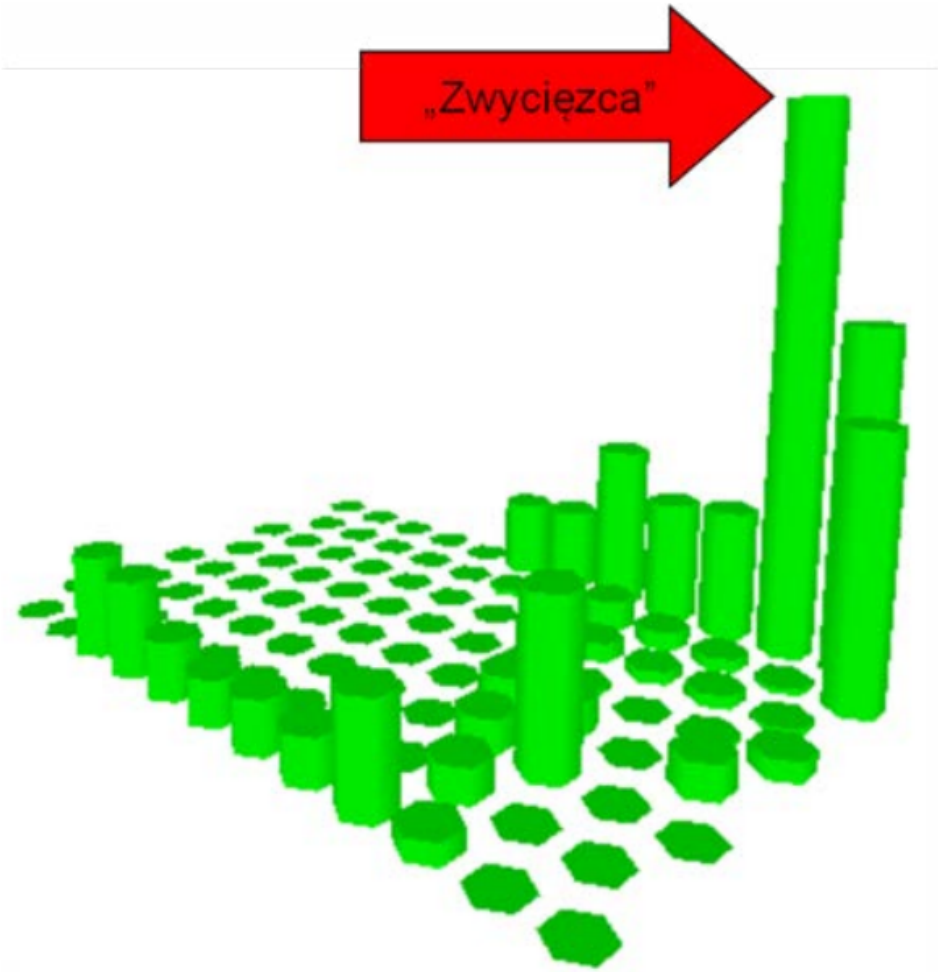
Jest najbardziej znaną i do tego wykorzystywaną siecią samouczącą się, która realizuje zasadę samoorganizacji (SOM). Jest to także najbardziej znany przykład sieci konkurencyjnej wykorzystującej koncepcję sąsiedztwa. W wyniku uczenia powstaje mapa topologiczna, której aprioryczna interpretacja jest niemożliwa, ponieważ sieć uczy się bez nauczyciela oraz użytkownik nie jest w stanie kontrolować tego, co się robi. Natomiast po uczeniu jesteśmy w stanie ustalić jakie poszczególne rejony mapy mają znaczenie na podstawie analizy danych wejściowych. Poniższy rysunek przedstawia strukturę sieci Kohonena, gdzie ma on nakreślić koncepcyjną strukturę schematu, bowiem rzeczywiste sieci Kohonena cechują się tym, że działają w wielowymiarowych przestrzeniach danych wejściowych, w związku z czym warstwa wejściowa zawiere bardzo wiele neuronów (skojarzonych z wieloma sygnałami wejściowymi). Typowa warstwa topologiczna zawiera również bardzo wiele neuronów, dzięki czemu sieć po nauczeniu może prezentować bardzo subtelnie rozróżnienia i klasyfikacje dancyh wejściowych.



Konkurencyjna sieć neuronowa

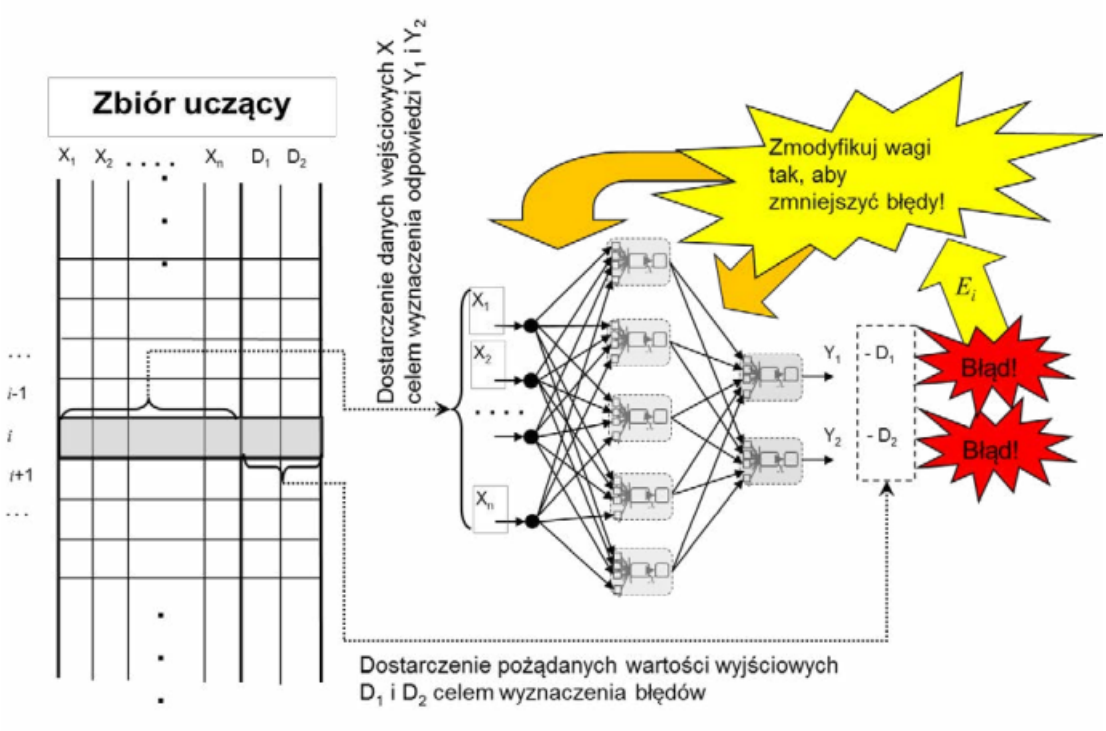
W niektórych sieciach neuronowych wśród neuronów warstwy wyjściowej lub mapy topologicznej wprowadza mechanizm konkurencji, polegający na tym, że sygnały wyjściowe tych neuronów porównuje się ze sobą. Po podaniu określonego sygnału wyjściowego do sieci - na jej wyjściu otrzymuje się sygnały o różnych wartościach pochodzące od różnych neuronów warstwy wyjściowej lub warstwy topologicznej. Wśród tych sygnałów odnajduje się ten, który ma największą wartość i ten neuron zostaje wskazany jako zwycięzca (patrz rysunek). Z faktu, że określony neuron został uznany za

zwycięzcę, wynikają różne konsekwencje. W szczególności w niektórych sieciach na etapie uczenia zmiany wag dotyczą wyłącznie zwycięzcy oraz (niekiedy) jego sąsiedztwa. W sieciach klasyfikacyjnych zwycięski neuron wskazuje poprawną kategoryzację sygnału wejściowego lub poprawne rozpoznanie obiektu reprezentowanego przez ten sygnał.



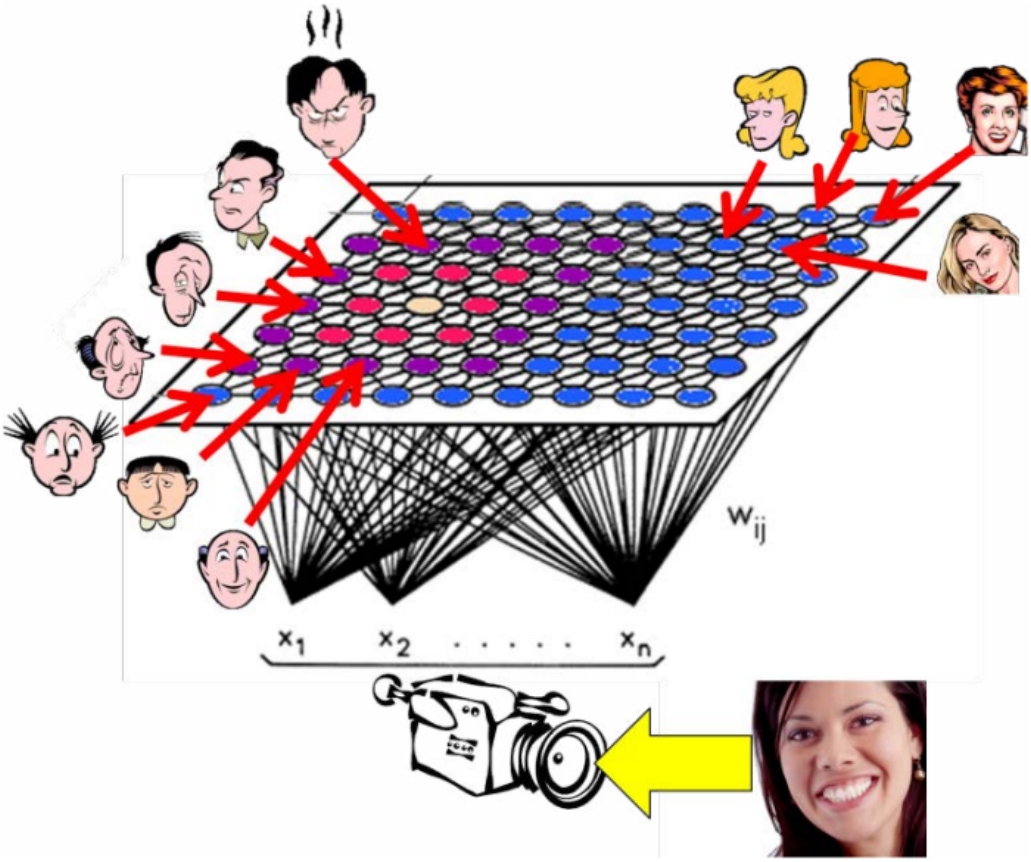
Korekcja błędu

Zmiana wartości parametrów sieci (najczęściej wag) mająca na celu zmniejszenie błędu popełnianego przez sieć. Ponieważ błąd wyznaczany jest podczas jednego kroku procesu uczenia, przeto korekta błędu nie może być zbyt radykalna, bo łatwo jest doprowadzić do sytuacji, w której zmiana parametrów wynikająca z pokazania jednego przypadku uczącego ze zbioru uczącego może popsuć wartości parametrów ustalone wcześniej dla innych przypadków uczących. W praktyce wielkość korekty błędu determinuje współczynnik uczenia. Przebieg typowej korekty błędu przedstawia poniższy schemat.



Mapa topologiczna

W sieci Kohonena ta warstwa, na której prezentowany jest wynik działania sieci, nazywana jest warstwą topologiczną. Neurony należące do tej warstwy specjalizują się w identyfikowaniu poszczególnych obiektów, jakie w trakcie procesu samouczenia były sieci prezentowane na jej wejściu. Każdy neuron warstwy topologicznej ma więc przypisany do siebie obiekt, którego pojawienie się na wejściu sieci powoduje, że ten właśnie neuron zostaje zwycięzcą (patrz hasło Konkurencyjna sieć neuronowa). Rozmieszczenie tych obiektów formuje właśnie mapę topologiczną, pokazaną symbolicznie na rysunku. Znajomość mapy topologicznej ułatwia użytkownikowi interpretację i wykorzystanie wyników obliczeń dostarczanych przez sieć Kohonena.



Przykłady algorytmów

WTA (Winner take all)

Algorytm WTA (Winner Takes All) to jeden z podstawowych algorytmów stosowanych w sieciach neuronowych typu Kohonena, które służą do grupowania danych wejściowych na podstawie podobieństwa. Algorytm ten wykorzystuje konkurencyjną regułę uczenia, która pozwala na wyłonienie zwycięzcy w procesie klasyfikacji.

Algorytm WTA składa się z trzech etapów:

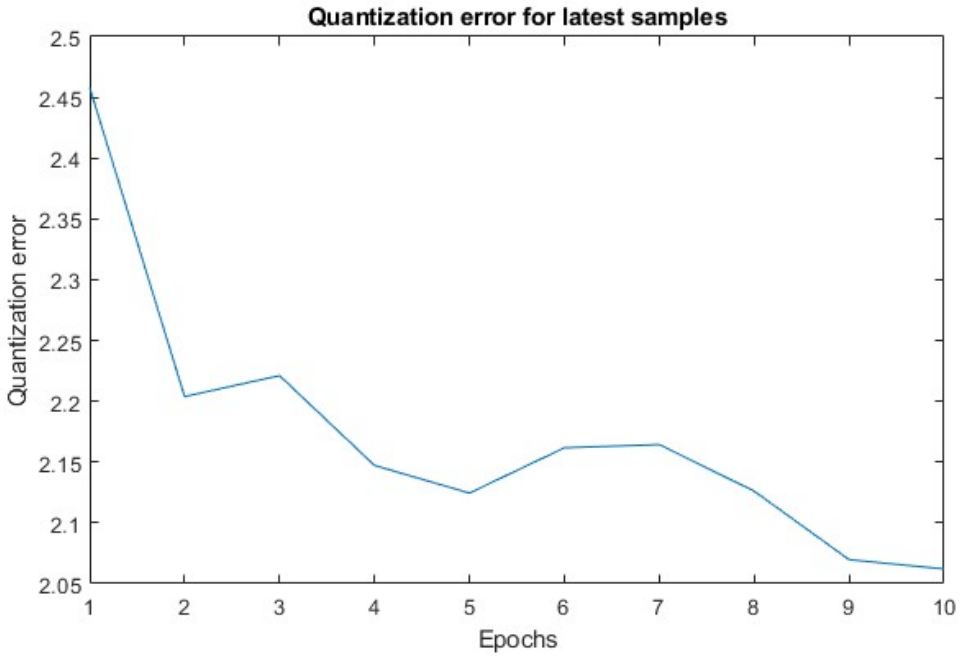
1. Inicjalizacja - losowo inicjuje się położenie neuronów w przestrzeni wejściowej i przypisuje im losowe wagi.
2. Konkurencja - neuron, który jest najbliżej aktualnie prezentowanego wektora wejściowego jest uznawany za zwycięzcę. To właśnie on jest aktywowany, a jego wagi są modyfikowane w kierunku wektora wejściowego. Wagi pozostałych neuronów pozostają niezmienione. Ten etap może być powtarzany wielokrotnie dla różnych wektorów wejściowych.
3. Stabilizacja - po zakończeniu procesu konkurencji, sieć ulega stabilizacji. Wagi neuronów nie są już modyfikowane, a każdy neuron jest przypisany do jednej z grup.

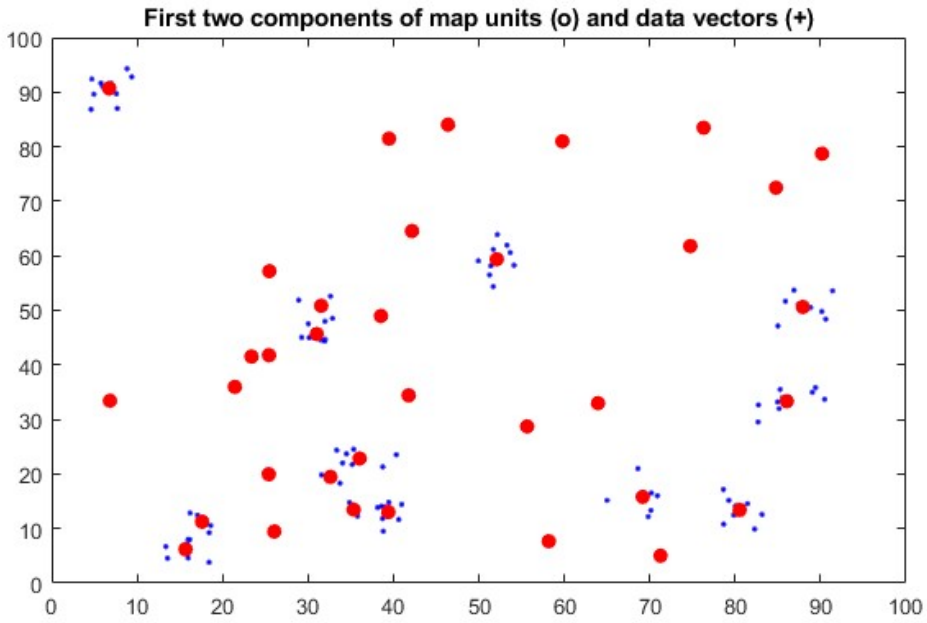
Algorytm WTA wykorzystuje zasadę zwycięzcy zabierającego wszystko - to znaczy, że zwycięzca zabiera całą pulę i jest jedynym neuronem, który jest aktywowany w trakcie prezentacji danego wektora wejściowego. Dzięki temu algorytm WTA umożliwia wyłonienie dominującego wzorca wśród danych wejściowych i grupowanie ich w klastry.

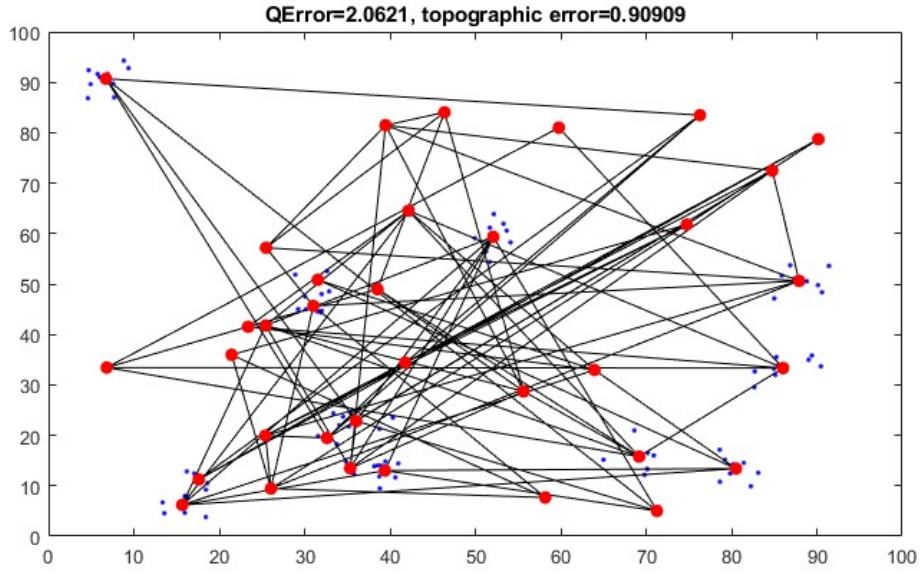
Algorytm WTA znajduje zastosowanie w rozpoznawaniu wzorców, analizie danych, analizie obrazów oraz klasteryzacji danych. Jego zaletami są prostota, szybkość i skuteczność. Jednakże, algorytm WTA ma również pewne wady, takie jak niestabilność w przypadku wystąpienia szumów lub zmian w danych wejściowych, a także niemożność rozpoznawania złożonych wzorców.

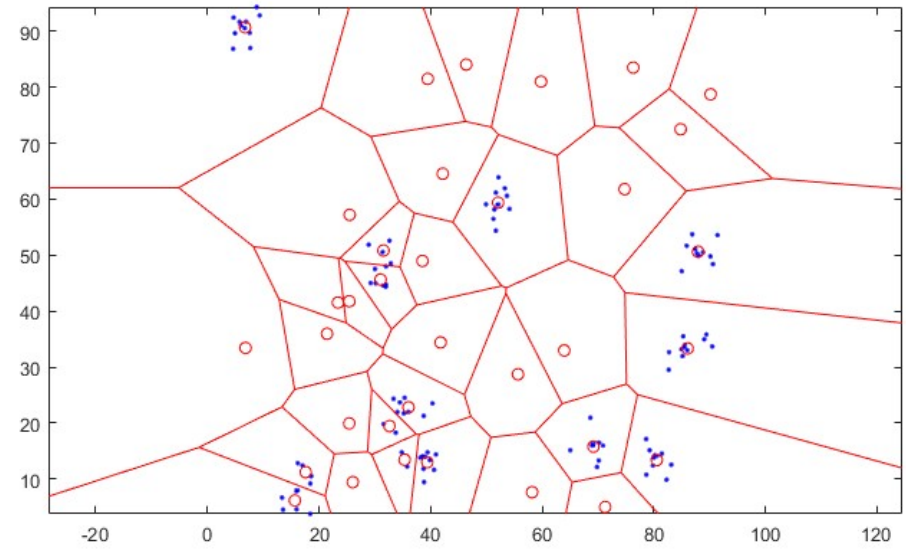
Wykresy dla określonych parametrów

liczba grup = 10
liczba obiektów = 10
liczba epok = 10
wartość początkowa uczenia = 0.5


Quantization error for latest samples
Quantization error
Epochs
2.5
2.45
2.4
2.35
2.3
2.25
2.2
2.15
2.1
2.05
1
2
3
4
5
6
7
8
9
10


First two components of map units (o) and data vectors (+)
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100





CWTA (Conscience Winner Takes All)

lgorytm CWTA (Conscience Winner Takes All) to modyfikacja standardowego algorytmu Winner-Takes-All (WTA), stosowanego w sieciach neuronowych typu Kohonena. WTA wybiera zwycięzcę na podstawie minimalnej odległości między wektorem wejściowym a neuronami SOM. WTA nie uwzględnia jednak dodatkowej wiedzy o stanie sieci, co może prowadzić do niedopasowania zwycięzcy.

Algorytm CWTA wprowadza pojęcie "sumy sumień" dla każdego neuronu, która odzwierciedla aktywność neuronu na przestrzeni czasu. Im częściej neuron jest aktywny, tym wyższa jest jego suma sumień. Dzięki temu, że CWTA uwzględnia historię aktywności neuronów, pozwala na wybór bardziej stabilnego zwycięzcy, który jest w stanie lepiej odzwierciedlić strukturę wejściową.

Algorytm CWTA składa się z dwóch faz:

1. Faza konkurencji - każdy neuron SOM jest aktywowany przez wektor wejściowy, a następnie suma sumień każdego neuronu jest zwiększana o wartość procentową.

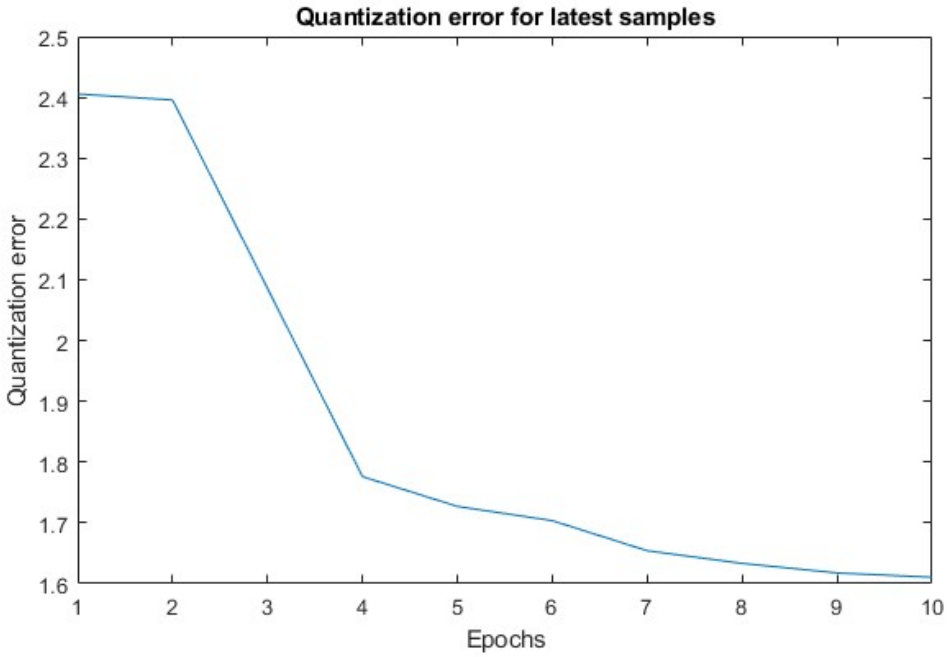
2. Faza selekcji zwycięzcy - neuron z najniższą wartością sumy sumień jest uznawany za zwycięzcę i jest aktualizowany.

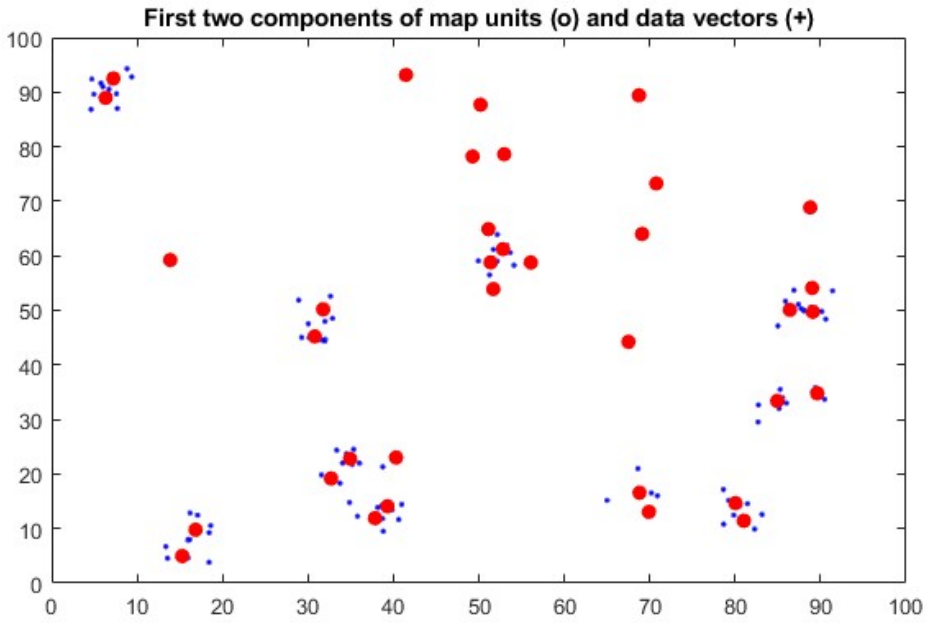
Algorytm CWTA pozwala na uniknięcie efektu "histerii" sieci, gdzie zwycięzca stale wygrywa, a inne neurony są wykluczone. Dzięki uwzględnieniu historii aktywności, algorytm CWTA może przeciwdziałać temu efektowi, wybierając bardziej stabilnych zwycięzców i zapewniając bardziej równomierne rozmieszczenie neuronów SOM w przestrzeni wejściowej.

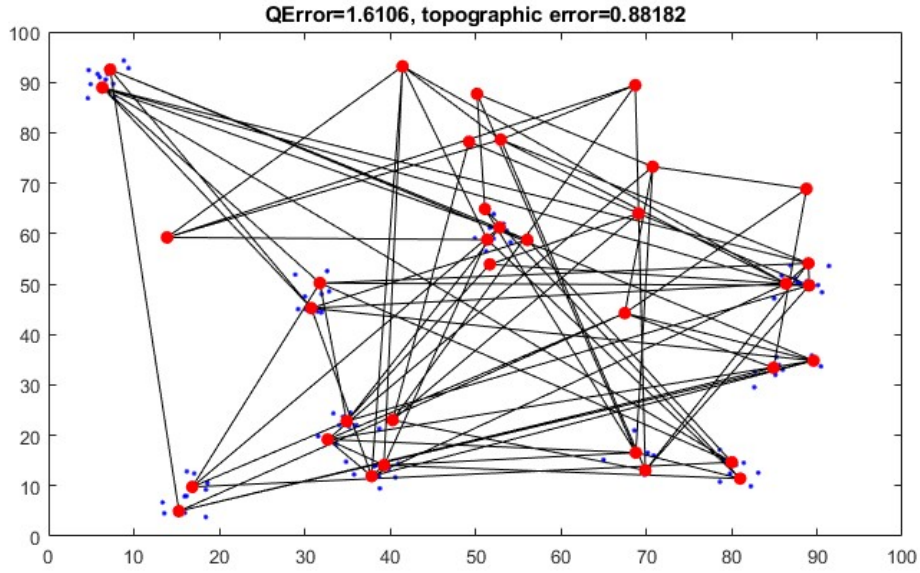
Algorytm CWTA znajduje zastosowanie w rozpoznawaniu wzorców, klasyfikacji obrazów i analizie danych, gdzie ważna jest stabilność wyboru zwycięzcy i równomierność rozkładu neuronów SOM w przestrzeni wejściowej.

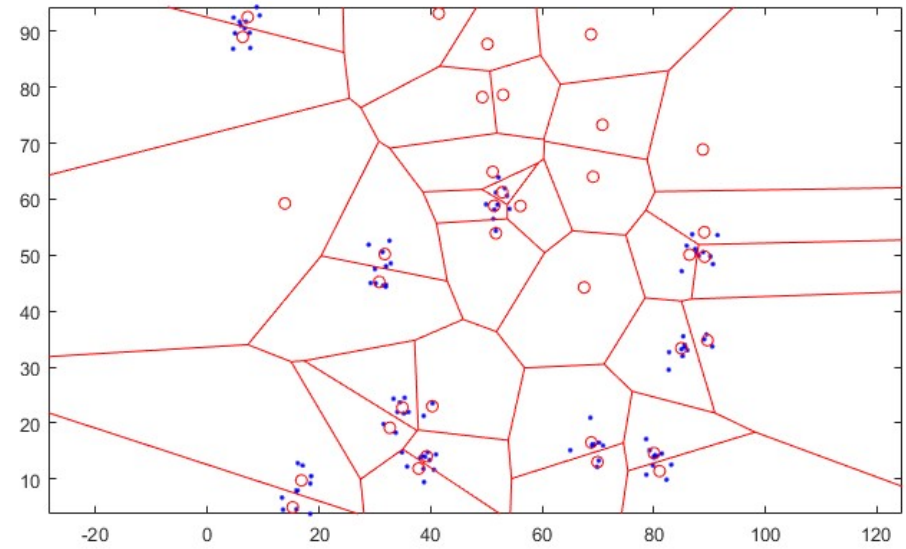
Wykresy dla określonych parametrów

liczba grup = 10
liczba obiektów = 10
liczba epok = 10
wartość początkowa uczenia = 0.5


Quantization error for latest samples
Quantization error
Epochs
2.5
2.4
2.3
2.2
2.1
2
1.9
1.8
1.7
1.6
1
2
3
4
5
6
7
8
9
10


First two components of map units (o) and data vectors (+)
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100





WTM batch (Winner Takes Most Batch)

Algorytm WTM Batch (Winner Takes Most Batch) jest stosowany w sieciach neuronowych typu Kohonena, które służą do grupowania danych wejściowych na podstawie podobieństwa. Algorytm ten składa się z dwóch faz: inicjalizacji i konkurencji.

1. Inicjalizacja:

W fazie inicjalizacji losowo wybierane są wagi dla neuronów SOM (Self-Organizing Map). Wagi te są przypisywane początkowemu rozkładowi w przestrzeni wejściowej.

2. Konkurencja:

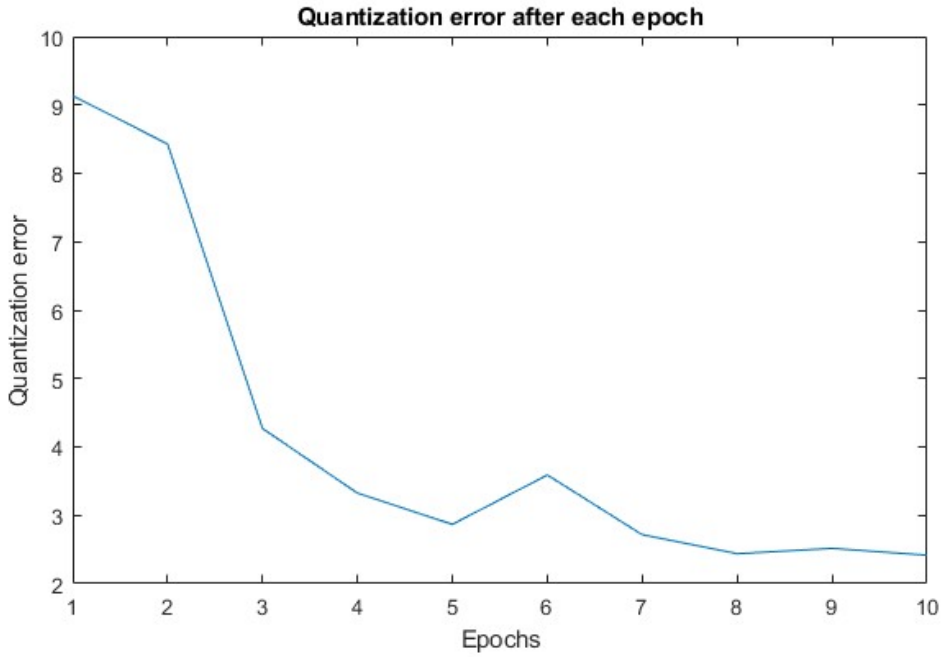
W fazie konkurencji dla każdego wejścia wyznaczany jest neuron z najbliższą wagą, zwany zwycięzcą. Następnie wagi są aktualizowane, tak aby zwycięzca miał jeszcze bliższe wartości do wejścia, a sąsiednie neurony do zwycięzcy również są aktualizowane, ale w mniejszym stopniu.

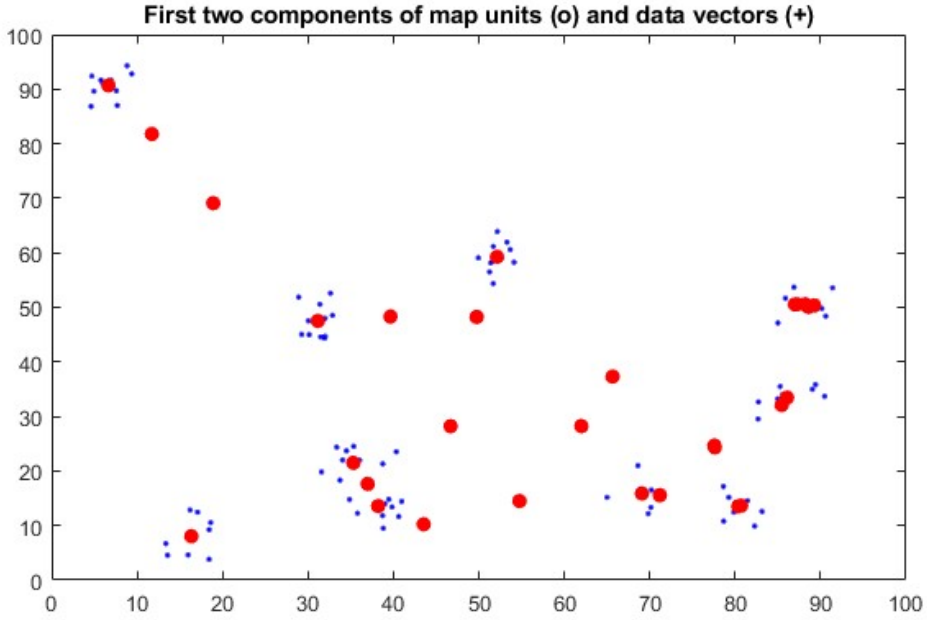
Algorytm WTM Batch wykorzystuje strategię zwycięzca-bierze-więcej (winner-takes-most), co oznacza, że wybrany neuron otrzymuje większą aktualizację swoich wag, a sąsiednie neurony są aktualizowane w mniejszym stopniu. Dzięki temu algorytm może szybciej osiągać stabilizację i lepiej radzić sobie z dużymi zbiorami danych.

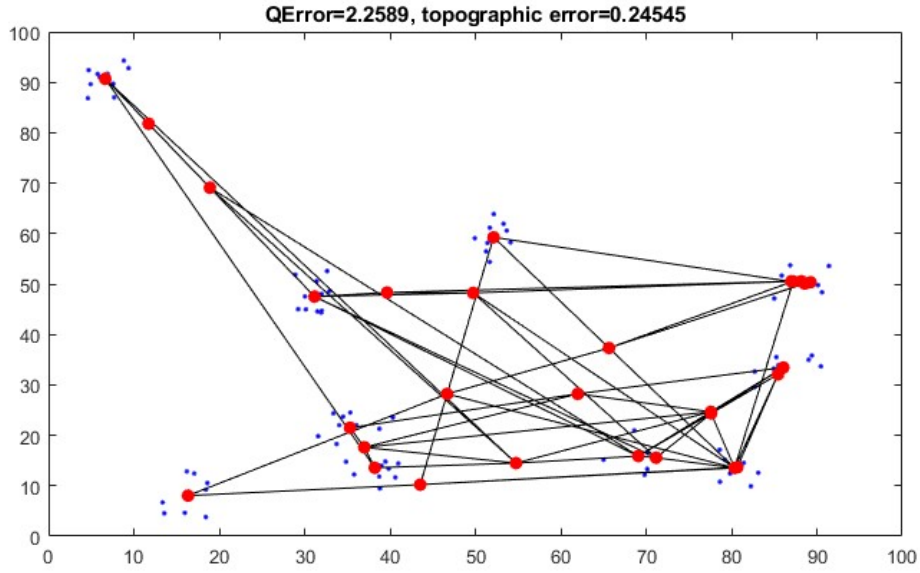
Algorytm WTM Batch jest stosowany w przetwarzaniu obrazów i dźwięku, gdzie sieci Kohonena służą do grupowania pikseli lub cech dźwiękowych w zbiory o podobnych właściwościach.

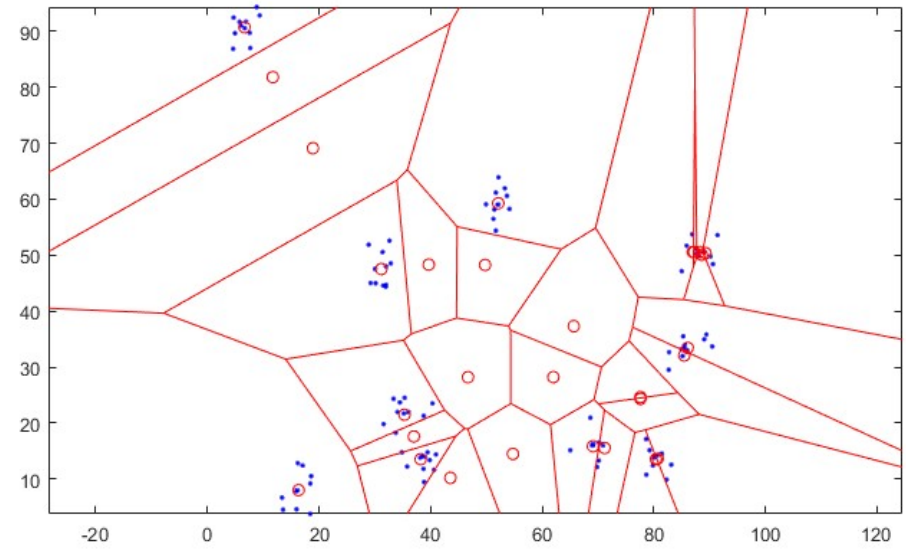
Wykresy dla określonych parametrów

liczba grup = 10
liczba obiektów = 10
liczba epok = 10
wartość początkowa uczenia = 0.2
promień początkowy = 3


Quantization error after each epoch
Quantization error
Epochs
10
9
8
7
6
5
4
3
2
1
2
3
4
5
6
7
8
9
10


First two components of map units (o) and data vectors (+)
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100





WTM seq

Algorytm WTM Seq (Winner Takes Most Sequence) jest rozszerzeniem algorytmu WTM Batch, również stosowanym w sieciach neuronowych typu Kohonena. W odróżnieniu od WTM Batch, WTM Seq uwzględnia sekwencje danych wejściowych i ma zastosowanie w analizie sekwencji tekstu.

Algorytm WTM Seq składa się z trzech faz:

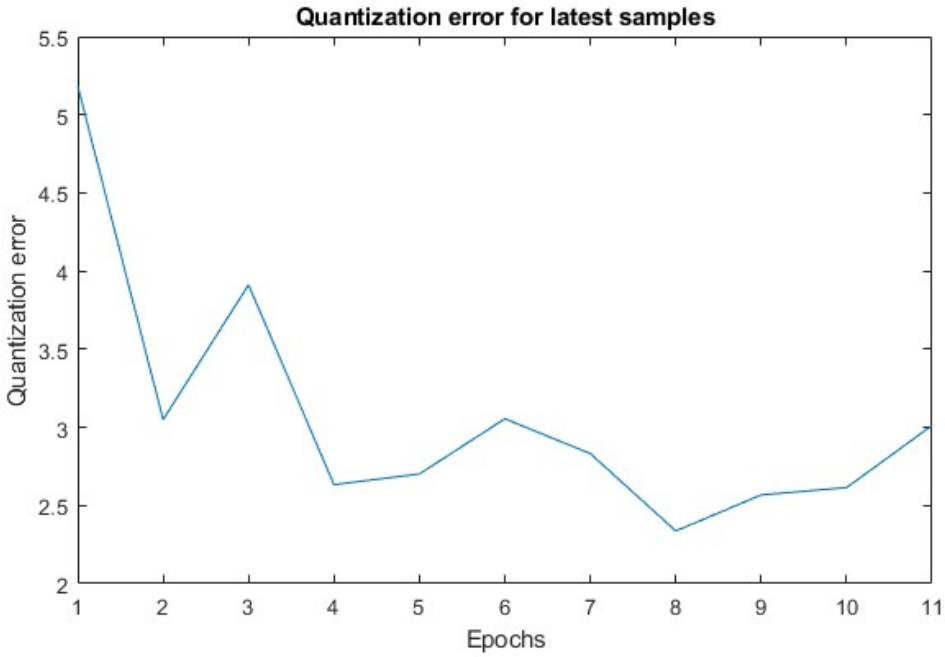
1. Inicjalizacji - losowo wybierane są wagi dla neuronów SOM.
2. Konkurencji - dla każdej sekwencji wyznaczany jest neuron z najbliższą wagą, zwany zwycięzcą. Wagi są aktualizowane, aby zwycięzca miał jeszcze bliższe wartości do sekwencji, a sąsiednie neurony do zwycięzcy również są aktualizowane, ale w mniejszym stopniu.
3. Adaptacji - po każdej iteracji uczącej wagi są aktualizowane w sposób specyficzny dla sekwencji. Zamiast aktualizować wagi neuronów tylko na podstawie jednej sekwencji, algorytm WTM Seq wykorzystuje wagę neuronu, która zależy od sumy

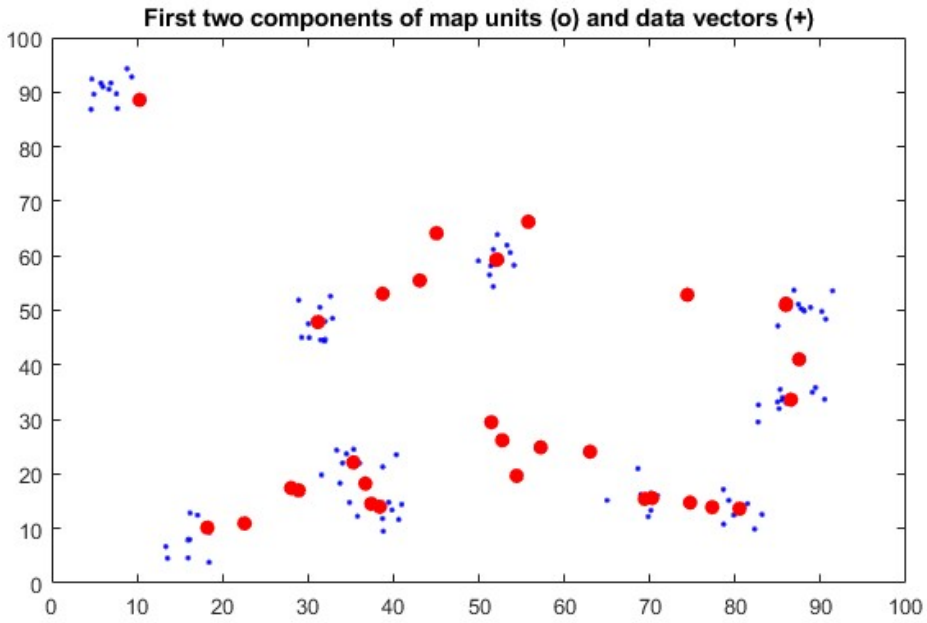
wag zwycięzców dla każdej sekwencji w ciągu uczącym. To znaczy, że sekwencje występujące częściej wpłyną na aktualizację wag w sposób bardziej znaczący.

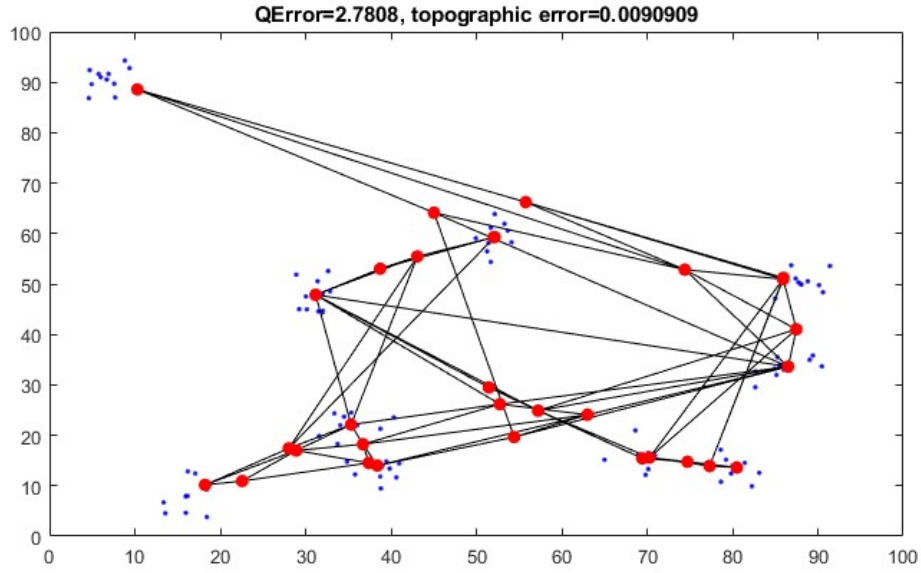
Algorytm WTM Seq jest stosowany w analizie sekwencji tekstu, na przykład do grupowania dokumentów na podstawie podobieństwa. Może być również stosowany w innych dziedzinach, gdzie występują sekwencje danych wejściowych, takich jak sekwencje DNA lub sygnały czasowe.

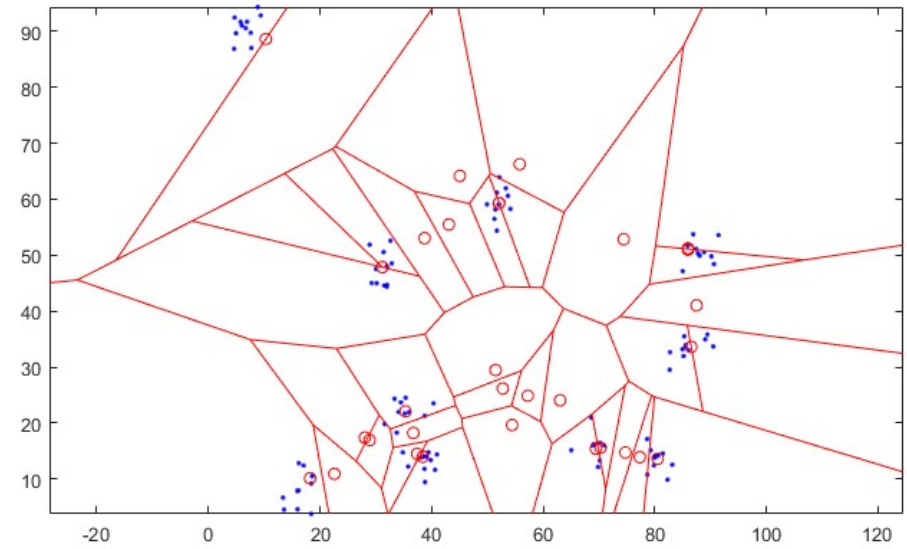
Wykresy dla określonych parametrów

liczba grup = 10
liczba obiektów = 10
liczba epok = 10
wartość początkowa uczenia = 0.05
promień początkowy = 4


Quantization error for latest samples
Quantization error
Epochs
5.5
5
4.5
4
3.5
3
2.5
2
1
2
3
4
5
6
7
8
9
10
11


First two components of map units (o) and data vectors (+)
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100





Neutral gas

Algorytm Neural Gas (NG) jest jednym z algorytmów stosowanych w sieciach neuronowych typu Kohonena, które służą do grupowania danych wejściowych na podstawie podobieństwa. Algorytm ten jest rozszerzeniem algorytmu Kohonena i wykorzystuje podobne założenia co WTM (Winner-Takes-Most).

Algorytm NG składa się z trzech etapów:

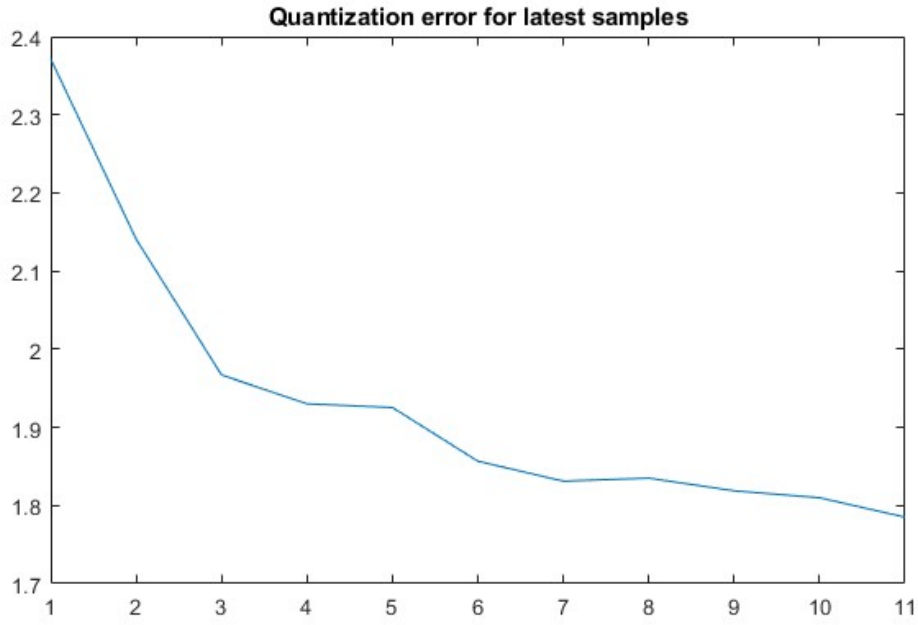
1. Inicjalizacja - losowo inicjuje się położenie neuronów w przestrzeni wejściowej i przypisuje im losowe wagi.
2. Adaptacja - neuron, który jest najbliżej aktualnie prezentowanego wektora wejściowego jest uaktualniany poprzez zmianę swojej pozycji oraz wagi. Sąsiedzi tego neuronu również są uaktualniani, ale w mniejszym stopniu. Proces ten jest powtarzany wielokrotnie aż do osiągnięcia stabilizacji.
3. Redukcja - usuwanie neuronów, które są mniej aktywne i nie przyczyniają się do zdefiniowania skutecznych klastrów.

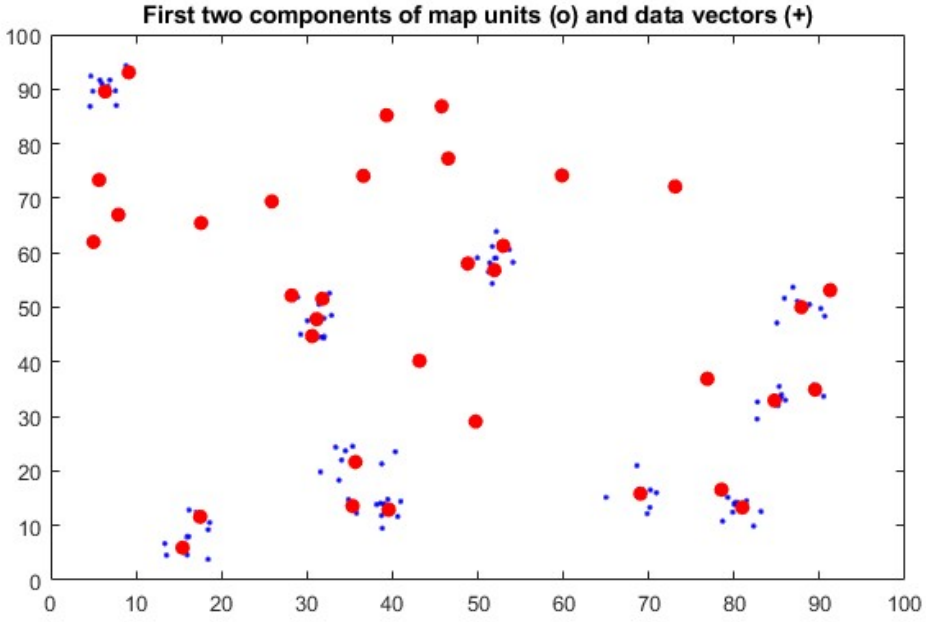
Algorytm NG różni się od standardowej sieci Kohonena tym, że bada on dystrybucję punktów w przestrzeni wejściowej, a nie odległość euklidesową między neuronami. W wyniku tego algorytm NG umożliwia bardziej równomierne rozmieszczenie neuronów w przestrzeni wejściowej oraz lepsze przewidywanie topologii rozkładu klastrów.

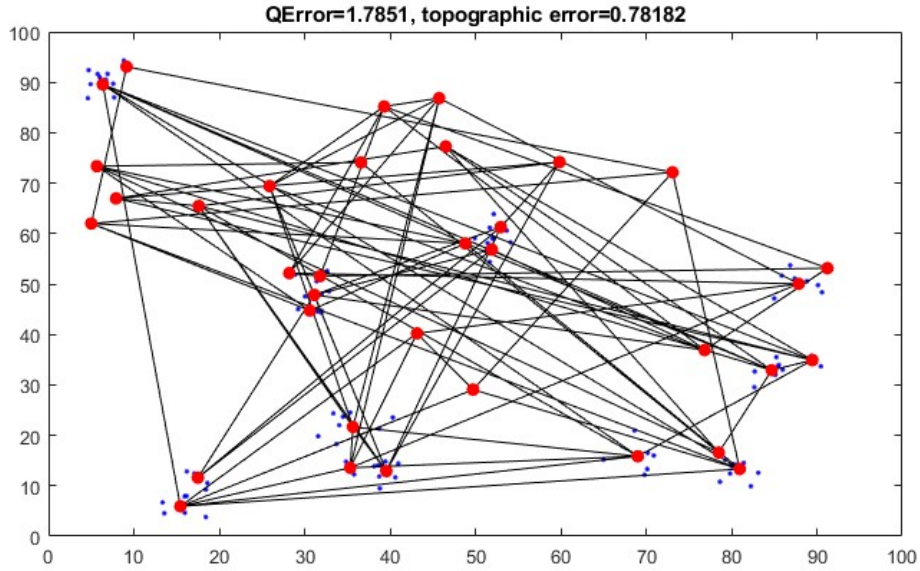
Algorytm Neural Gas znajduje zastosowanie w rozpoznawaniu wzorców, analizie danych, analizie obrazów oraz klasteryzacji danych.

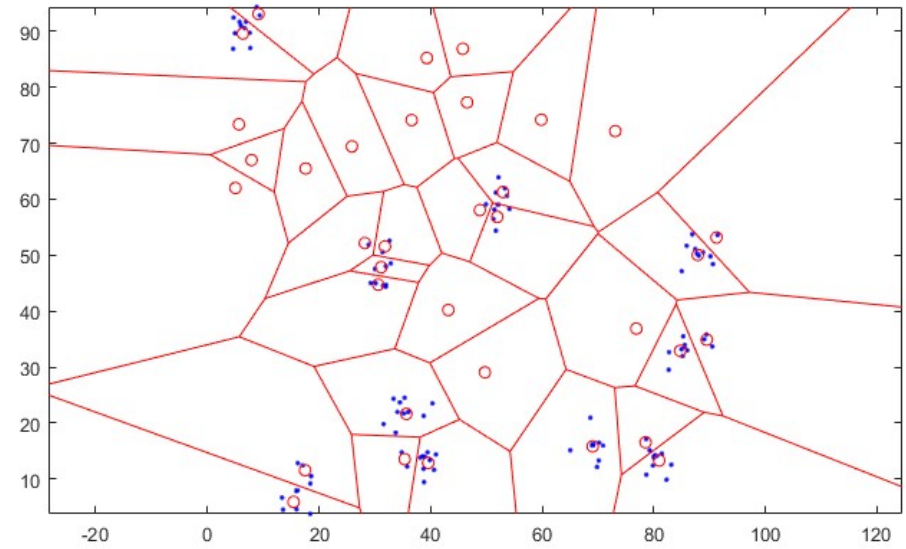
Wykresy dla określonych parametrów

liczba grup = 10
liczba obiektów = 10
liczba epok = 10
wartość początkowa uczenia = 0.5
lambda początkowa = 18


Quantization error for latest samples
2.4
2.3
2.2
2.1
2
1.9
1.8
1.7
1
2
3
4
5
6
7
8
9
10
11


First two components of map units (o) and data vectors (+)
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100





Podsumowanie oraz wnioski

Podsumowując, algorytmy w sieciach neuronowych typu Kohonena służą do grupowania danych wejściowych w klastry. Wyróżniamy kilka algorytmów, takich jak WTA, CWTA, Neural Gas oraz WTM batch i WTM seq, które różnią się między sobą sposobem wyłaniania zwycięzców i modyfikacji wag neuronów.

Niezbędnym elementem w sieciach Kohonena jest proces kwantyzacji, czyli przypisywania wektorów wejściowych do najbliższego neuronu. Jednakże, ten proces może wprowadzić błąd kwantyzacji, co z kolei może prowadzić do destabilizacji sieci i utraty jakości grupowania.

W algorytmie WTA, błąd kwantyzacji jest duży wynika z faktu, że wyłoniony zwycięzca reprezentuje tylko jeden klaster, co może prowadzić do nieprawidłowego grupowania danych wejściowych, szczególnie w przypadku wystąpienia szumów lub zmian w danych.

W algorytmie CWTA, błąd kwantyzacji jest mniejszy, ponieważ mechanizm sumy sumień umożliwia zapobieganie powstawaniu klastrów o niskiej jakości. Niemniej jednak, nadal istnieje ryzyko nieprawidłowego grupowania danych wejściowych.

W algorytmie Neural Gas, błąd kwantyzacji jest redukowany poprzez rozproszenie neuronów w przestrzeni wejściowej, co umożliwia dokładniejsze odwzorowanie danych i wyłonienie mniejszych, bardziej precyzyjnych klastrów.

W algorytmach WTM batch oraz WTM seq, błąd kwantyzacji jest mniejszy niż w algorytmie WTA, ponieważ umożliwiają wyłonienie kilku zwycięzców dla danego wektora wejściowego. To pozwala na zredukowanie ryzyka nieprawidłowego grupowania danych i zwiększenie stabilności sieci.

Wnioskiem z powyższego jest to, że błąd kwantyzacji jest kluczowym problemem w sieciach neuronowych typu Kohonena, a wybór odpowiedniego algorytmu zależy od specyfiki problemu, rodzaju danych wejściowych oraz wymagań dotyczących stabilności i jakości grupowania.