

Projekt z przedmiotu

Sztuczna Inteligencja

**Zrealizować sieć neuronową LVQ uczącą się rozpoznawania kwiatów irysa**

**Imię i nazwisko:** Rafał Nazarko

**Grupa projektowa:** P5

**Numer indeksu:** 163982

Rzeszów 2021

**Spis treści**

[1. Opis projektu 2](#_Toc72682011)

[2. Zbiór danych 2](#_Toc72682012)

[2.1 Przedstawienie zbioru 2](#_Toc72682013)

[2.2 Przygotowanie i normalizacja danych 2](#_Toc72682014)

[3. Wprowadzenie teoretyczne 4](#_Toc72682015)

[3.1 Sieć neuronowa 4](#_Toc72682016)

[3.1.1 Model sztucznego neuronu 5](#_Toc72682017)

[3.1.2 Sieć jednowarstwowa 6](#_Toc72682018)

[3.1.3 Sieć wielowarstwowa 7](#_Toc72682019)

[3.1.4 Sieć jednokierunkowa 8](#_Toc72682020)

[3.1.5 Sieć rekurencyjna 9](#_Toc72682021)

[3.1.6 Uczenie sieci neuronowej 9](#_Toc72682022)

[3.2 Algorytm LVQ 11](#_Toc72682023)

[3.2.1 LVQ1 11](#_Toc72682024)

[3.2.2 LVQ2 11](#_Toc72682025)

[3.2.3 LVQ2.1 11](#_Toc72682026)

[3.2.4 LVQ3 11](#_Toc72682027)

[3.2.5 Warstwa Kohonena 11](#_Toc72682028)

[3.3 Walidacja krzyżowa 11](#_Toc72682029)

[4. Kod programu 11](#_Toc72682030)

[5. Eksperymenty 11](#_Toc72682031)

[6. Wnioski 11](#_Toc72682032)

[7. Bibliografia 12](#_Toc72682033)

# Opis projektu

Celem projektu jest stworzenie sieci neuronowej LVQ uczącej się rozpoznawania kwiatów irysa. Zakres realizacji obejmował przygotowanie danych, stworzenie algorytmu w języku Python oraz przeprowadzenie niezbędnych eksperymentów, w ramach których poszukiwane były najbardziej optymalne konfiguracje parametrów sieci a także najkrótszy czas wykonania.

# Zbiór danych

## Przedstawienie zbioru

Baza danych Iris Data Set [1] wykorzystana w tym projekcie pochodzi z repozytorium uczenia maszynowego UC Irvine. Jest to prawdopodobnie najbardziej znana baza danych, jaką można znaleźć w literaturze dotyczącej rozpoznawania wzorców. Zawiera 3 klasy po 50 instancji. Nie posiada brakujących/nieokreślonych wartości dlatego nie jest konieczne usuwanie rekordów. Każda z instancji składa się z 4 atrybutów w formacie liczb rzeczywistych oraz atrybutu klasy, która odnosi się do typu rośliny.

**Informacje o atrybutach:**

1. **Długość kielicha** (w centymetrach)
2. **Szerokość kielicha** (w centymetrach)
3. **Długość płatka** (w centymetrach)
4. **Szerokość płatka** (w centymetrach)
5. **Klasa:**
   * Iris Setosa
   * Iris Versicolour
   * Iris Virginica

## Przygotowanie i normalizacja danych

Klasy należało przekształcić z literałów na wartości liczbowe aby ujednolicić dane oraz sprawniej przeprowadzać operacje porównywania. Literały klas wymienione w poprzednim podpunkcie przekształcono odpowiednio w liczby naturalne z przedziału od 1 do 3. Separator atrybutów zamieniono z przecinka (‘,’) na średnik (‘;’).

Kolejnym krokiem była normalizacja danych w celu umożliwienia ich wzajemnego porównywania i dalszej analizy. Wszystkie atrybuty poza klasami zostały przeliczone na odpowiadające im wartości w przedziale od -1 do 1.

Poniżej zostały zamieszczone dane oryginalne oraz odpowiadające im dane po przygotowaniu i normalizacji, aby zobrazować zaistniałe zmiany.

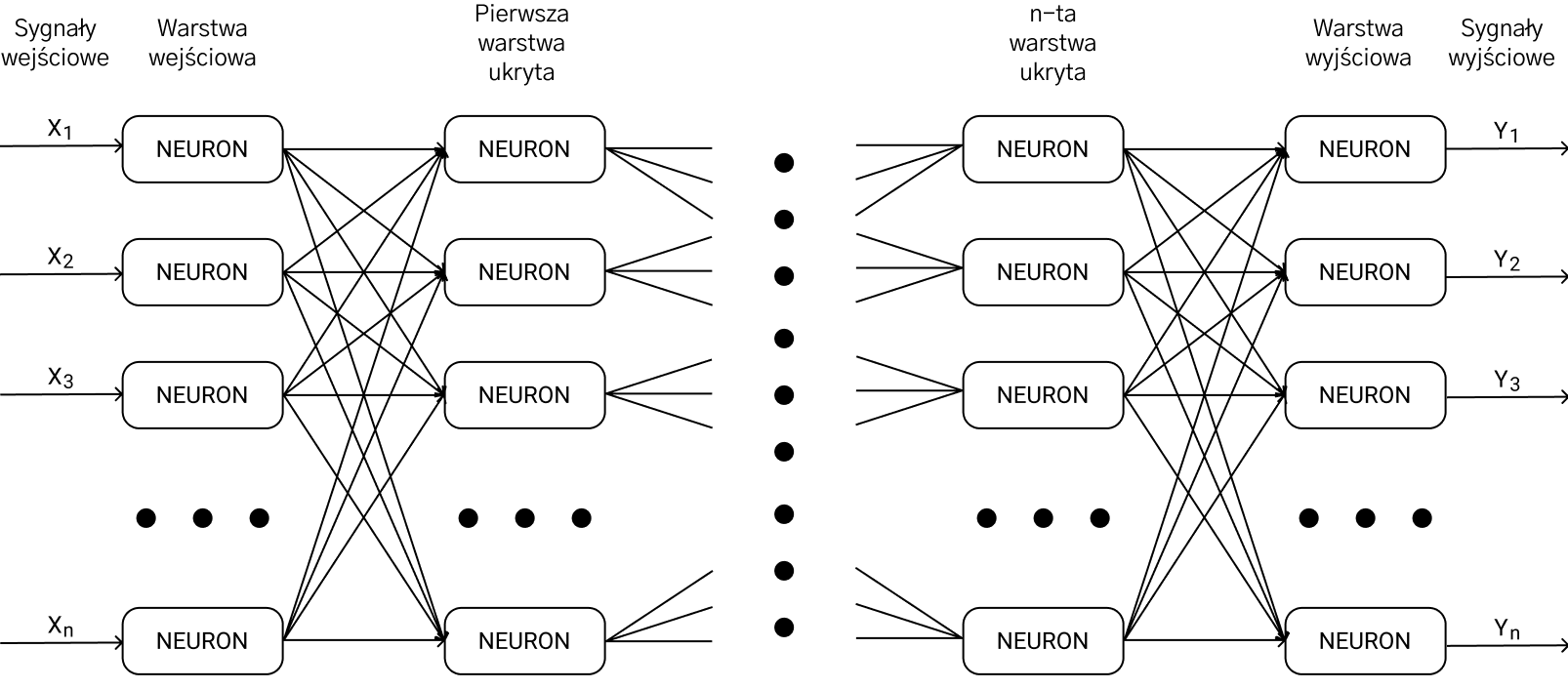
|  |
| --- |
| **Dane oryginalne** |
| 5.0,3.3,1.4,0.2,Iris-setosa  7.0,3.2,4.7,1.4,Iris-versicolor |

|  |
| --- |
| **Dane po przygotowaniu i normalizacji** |
| -0.611111;0.083333;-0.864407;-0.916667;1  0.5;0.0;0.254237;0.083333;2 |

# Wprowadzenie teoretyczne

## Sieć neuronowa

Termin sztuczne sieci neuronowe (SSN) lub również często określane jako po prostu sieci neuronowe (NN), obejmują rodzinę nieliniowych metod obliczeniowych, które przynajmniej we wczesnej fazie ich rozwoju były inspirowane funkcjonowaniem ludzkiego mózgu. Są to bardzo wyrafinowane narzędzia obliczeniowe zdolne do modelowania niezwykle złożonych funkcji. Składa się ona z dużej liczby elementów przetwarzających informacje. Elementy te, zwane *neuronami*, powiązane są ze sobą w sieć wzajemnie zależnych od siebie współczynników. Połączenia oraz ich parametry są podstawą działania sieci a jej topologia może różnić się w zależności od obranego algorytmu. Strukturę taką dzieli się na warstwy: *wejściową, wyjściową* oraz warstwy *ukryte*. W każdej sieci obligatoryjnie znajduje się warstwa wejściowa – przyjmująca zestaw danych, który ma zostać poddany analizie, oraz wyjściową – zwracającą wyniki działania algorytmu. Warstw ukrytych może istnieć dowolna ilość a decyduje o tym model wybranej sieci. W większości współczesnych zastosowań realizuje się sieci o modelu wielowarstwowym. *Sygnały* przekazywane są pomiędzy neuronami poszczególnych warstw aż do najwyższej warstwy dającej odpowiedź, a ich wpływ na odpowiedź neuronu jest zależna od *wag*, które są odpowiednio dopasowywane w procesie *uczenia sieci.*

**

*Rysunek 1. Przykładowy model sieci neuronowej.*

*Źródło: opracowanie własne*

### Model sztucznego neuronu

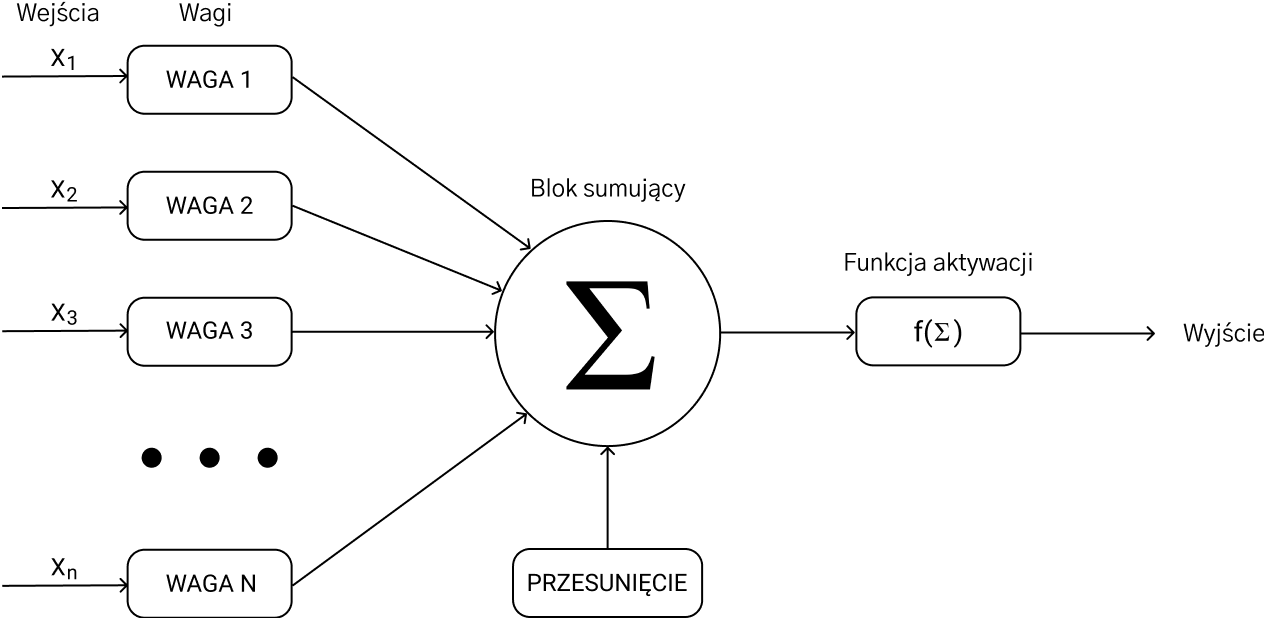
Mianem sztucznego neuronu określa się system przetwarzający sygnały przekazane do jego wejścia, na pojedynczą wartość wyjściową. Model sztucznego neuronu zwany *perceptronem* został stworzony na podstawie biologicznego neuronu.

Może on przyjmować dowolną ilość sygnałów wejściowych, najczęściej oznaczanych jako . Sygnały z wejść mnożone są z odpowiednimi wagami , a następnie sumowane wraz z przesunięciem w bloku sumującym [5]. Obliczona wartość poddawana jest działaniu funkcji aktywacji, która ostatecznie zwraca sygnał wyjściowy zwany *pobudzeniem neuronu*. Ogólny wzór na pobudzenie neuronu przedstawia poniższa zależność:

(3.1)

gdzie:

* – wartość i-tego sygnału wejściowego
* – waga i-tego sygnału wejściowego
* – ilość sygnałów wejściowych
* – przesunięcie
* – pobudzenie neuronu

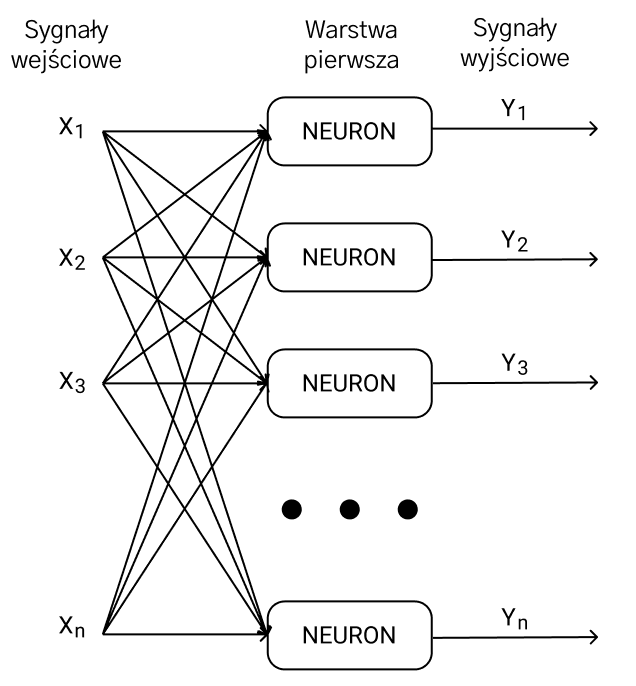


*Rysunek 2. Model sztucznego neuronu.*

*Źródło: opracowanie własne*

### Sieć jednowarstwowa

Sieć jednowarstwowa ma bardzo prymitywną budowę. Składa się jedynie z jednej warstwy neuronów oraz sygnałów wejściowych i wyjściowych. Nie posiada żadnych neuronów w warstwach ukrytych. Ma niewielkie znaczenie praktyczne, lecz nadal znajduje zastosowanie wszędzie tam, gdzie występuje niska złożoność problemu a istnienie jednej warstwy jest wystarczające do rozwiązania określonego zagadnienia. Neurony ułożone w pojedynczej warstwie działają niezależnie od siebie, stąd możliwości takiej sieci są ograniczone do możliwości pojedynczych neuronów [2]. Każdy z nich realizuje odwzorowanie funkcyjne pobudzenia neuronu (3.1). Przykładowa struktura sieci jednowarstwowej została zobrazowana poniżej.



*Rysunek 3. Model sieci jednowarstwowej.*

*Źródło: opracowanie własne*

### Sieć wielowarstwowa

Sieci wielowarstwowe znajdują zastosowanie w większości współczesnych realizacji sieci neuronowych. Tworzone są przez neurony ułożone w wielu warstwach [2]. Są rozszerzeniem sieci jednowarstwowej, jednak oprócz warstwy wejściowej i wyjściowej istnieje pomiędzy nimi co najmniej jedna warstwa ukryta. Sygnały wejściowe podawane są na wejścia neuronów warstwy wejściowej. Przetworzone dane są przekazywane na wyjścia neuronów, skąd trafiają na wejścia neuronów kolejnej warstwy. Proces ten jest powtarzany tyle razy ile jest warstw ukrytych. Na koniec, wyjścia ostatniej warstwy ukrytej są sygnałami wejściowymi ostatniej warstwy – warstwy wyjściowej. Działanie poszczególnych warstw można przedstawić według poniższej zależności:

(3.2)

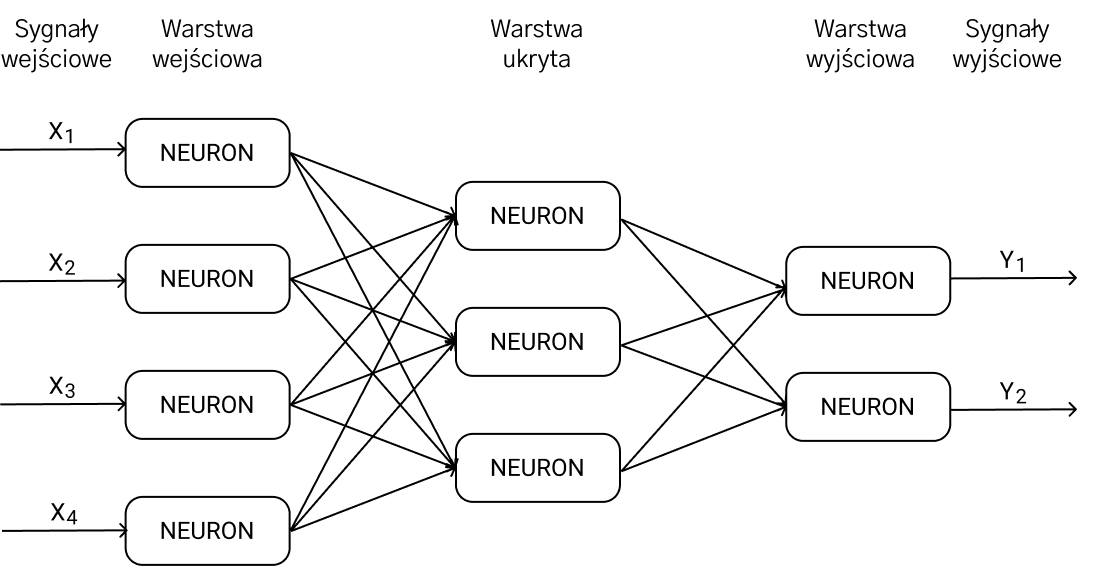
gdzie:

* – macierz wag dla n-tej warstwy
* – macierz wektorów wejściowych
* – wektor przesunięć dla n-tej warstwy
* – funkcja aktywacji dla n-tej warstwy
* – pobudzenie neuronu na n-tej warstwie

Dla zobrazowania działania sieci wielowarstwowych, poniżej został przedstawiony opis działania sieci trójwarstwowej [3]:

(3.3)

Do czego przydatne są warstwy ukryte? Powodem jest to, że jeśli nie ma ukrytych warstw to odwzorowywane dane wejściowe, nie są ze sobą w żaden sposób powiązane na wyjściach poszczególnych neuronów. W rzeczywistych problemach zmienne wejściowe są zwykle wysoce współzależne i wpływają na wynik w skorelowany sposób. Neurony warstwy ukrytej pozwalają uchwycić subtelne interakcje pomiędzy danymi wejściowymi, które ostatecznie wpływają wyjście. Ponadto, ukryte warstwy reprezentują cechy lub atrybuty wyższego poziomu naszych danych. Każdy z neuronów w warstwie ukrytej waży dane wejściowe inaczej, ucząc się innej pośredniczącej charakterystyki danych, a neuron wyjściowy jest wtedy ich funkcją zamiast surowych danych wejściowych. Włączając więcej niż jedną ukrytą warstwę, sieć dostaje możliwość nauczenia się wielu poziomów abstrakcji oryginalnych danych wejściowych przed uzyskaniem ostatecznego wyniku. Dzięki wykorzystaniu wielu warstw, sieć potrafi rozwiązywać problemy, z którymi nie radzi sobie sieć jednowarstwowa. Funkcje aktywacji zwiększają jej zdolność do wychwytywania nieliniowych relacji między wejściami a wyjściami. Łącząc ze sobą wiele nieliniowych transformacji za pomocą warstw, radykalnie zwiększa to elastyczność i zastosowanie sieci neuronowych [4].



*Rysunek 4. Przykładowa struktura sieci trójwarstwowej.*

*Źródło: opracowanie własne*

### Sieć jednokierunkowa

Sieć jednokierunkowa jest jednym ze sposobów działanie sieci neuronowych. Taką architekturę posiadają najprostsze sieci neuronowe. Informacje są podawane tylko w jednym kierunku – każdy neuron reaguje na jeden zestaw danych wejściowych tylko raz. Są częściej stosowane ze względu na prostotę w analizie. Sieci takie nazywane są czasami *heteroasocjacyjnymi* i z założenia są rozpatrywane jako twory statyczne, ponieważ ewentualne procesy przejściowe, zachodzące w sieci podczas jej pracy, nie mają znaczenia z punktu widzenia celu funkcjonowania sieci i mogą być pomijane[5].

### Sieć rekurencyjna

Bardziej złożone sieci są rekurencyjne, co znaczy, że dane wychodzące z ukrytych warstw są podawane pośrednio lub bezpośrednio do jednej z poprzednich warstw. Sprzężenie zwrotne powoduje, że sieć po jakimś czasie osiąga stan równowagi i „decyduje się” na konkretną odpowiedź. Takie sieci określane bywają jako *autoasocjacyjne* i odznaczają się bogatymi własnościami dynamicznymi, ponieważ sygnały mogą krążyć w sieci dowolnie długo, tym samym powodując powstanie różnorodnych i ciekawych przebiegów sieci przejściowych [5].

### Uczenie sieci neuronowej

Uczenie sieci to proces dobierania odpowiedniego zestawu wag dla danego zadania. Można je podzielić na uczenie nadzorowane oraz uczenie bez nauczyciela. Metodę uczenia pod nadzorem stosuje się gdy problem do rozwiązania jest zrozumiały dla człowieka oraz spodziewana jest określona odpowiedź od sieci. Wiedząc jakie sygnały są poprawne na wyjściu sieci dla każdego sygnału możemy je porównać z aktualną odpowiedzią sieci, po czym w razie potrzeby zmodyfikować wagi neuronów. Dobór odpowiednich wag odpowiada procesowi minimalizacji funkcji błędu, którą można zapisać jako:

(3.4)

gdzie:

* – odpowiedź n-tego neuronu
* – poprawny sygnał dla n-tego neuronu
* – ilość neuronów
* – błąd dopasowania

W zależności od wybranego algorytmu, funkcja skorygowania wag sygnałów może wyglądać w inny sposób. Jednakże w większości przypadków funkcja ta będzie przybierać postać zbliżoną do poniższej:

(3.5)

gdzie:

* – nowa macierz wektorów wag
* – aktualna macierz wektorów wag
* – współczynnik uczenia
* – błąd dopasowania
* – macierz wektorów wejściowych

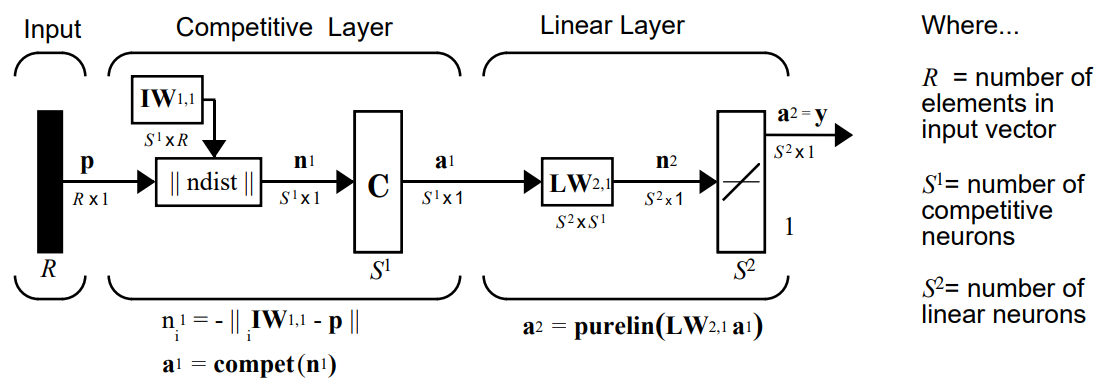
W sytuacjach gdy problem jest zbyt skomplikowany albo określone przez ludzi cechy prowadzą do niesatysfakcjonującego rozwiązania, można używać sposobu nauczania bez nauczyciela. Sieć ucząca się w taki sposób musi być odpowiednio złożona, wielowarstwowa, żeby mogła wykrywać w danych wejściowych różnorodne zależności. Ponieważ sieci uczone bez nadzoru nie dążą do zadanego wyniku, mają potencjał wykrywania prawidłowości i zależności, które wcześniej nie były znane. To powoduje, że są użyteczne do analizy zjawisk, których model jest niedokładny albo niekompletny – a to można powiedzieć prawie o wszystkich zjawiskach społecznych albo finansowych. Jednak stopień złożoności architektury takiej sieci jest wysoko zaawansowany a również nie pokrywa się z tematem projektu, toteż nie zostanie bardziej szczegółowo opisany w tym raporcie.

## Algorytm LVQ

Algorytm Nauki Kwantyzacji Wektorów (ang. Learning Vector Quantization) to adaptacyjna metoda klasyfikacji danych oparta na danych szkoleniowych z żądanymi informacjami o klasie. Został opracowany i jest najlepiej rozumiany jako algorytm klasyfikacji. Jest siecią jednokierunkową wielowarstwową. Obsługuje zarówno binarne (dwuklasowe), jak i wieloklasowe problemy klasyfikacji. Jest bardzo podobny do algorytmu k-najbliższego sąsiada. Jest prekursorem samoorganizujących się map funkcji (SOFM) [6].

Neurony zwane tutaj również wektorami kodującymi są w rzeczywistości wektorami zdefiniowanymi w przestrzeni wejściowej z przypisanymi dla nich etykietami klas, przy czym liczba neuronów jest znacznie mniejsza od liczby wektorów treningowych. Koncepcja algorytmu LVQ bazuje na przyciąganiu do siebie wektorów kodujących przez wektory z tą samą etykietą klasy i odpychaniu od siebie wektorów kodujących o niezgodnych etykietach klas. Wektory kodujące są inicjalizowane dla losowo wybranych wartości z uczącego zestawu danych. Następnie, na przestrzeni kilku epok, są one dostosowywane, aby jak najlepiej podsumować dane szkoleniowe za pomocą algorytmu uczącego. To jaką liczbę epok, neuronów czy wartość współczynnika uczenia przyjąć, aby algorytm jak najlepiej klasyfikował dane, należy rozstrzygnąć eksperymentalnie. Zrobione to zostanie w rozdziale 5 tego raportu.

W architekturze sieci LVQ można wyróżnić dwie kluczowe warstwy – warstwę konkurencyjną oraz warstwę liniową. Warstwa konkurencyjna uczy się klasyfikować wektory wejściowe w podobny sposób, jak robią to samoorganizujące się mapy funkcji. Warstwa liniowa przekształca warstwę konkurencyjną na docelowe klasyfikacje zdefiniowane przez użytkownika. Warto zauważyć, że struktura nie uwzględnia przesunięcia (tzw. biasu ).



*Rysunek 5. Struktura budowy sieci LVQ.*

*Źródło: Neural Network Toolbox 5 User’s Guide*

### LVQ1

Algorytm uczenia sieci typu LVQ w wersji pierwszej polega na iteracyjnej aktualizacji położenia wektorów kodujących tak by zminimalizować błąd klasyfikacji. Jest to wersja algorytmu, którą wykorzystano w tym projekcie. Proces iteracyjny składa się z głównej pętli, która wykonywana jest określoną liczbę epok, oraz pętli w której iteracja następuje po elementach zbioru treningowego. Elementy tego zbioru losowane są z książki kodowej a ich ilość jest równa ilości ustanowionych neuronów . Wówczas dla każdego wektora treningowego poszukiwany jest najbliżej leżący wektor kodujący. Odległość wektorów obliczana jest według wzoru na odległość Euklidesową, która ma postać:

(3.6)

gdzie:

* – wektor treningowy
* – wektor uczący
* – odległość Euklidesowa

Warstwa konkurencyjna opiera się na typie algorytmu „zwycięzca bierze wszystko” dlatego, że wybierany jest tylko jeden neuron z całego zbioru, którego pobudzenie będzie wynosiło 1 – reszta neuronów na wyjście podaje sygnał 0. Jeżeli obydwa wektory (kodujący i treningowy) należą do tej samej klasy wówczas aktualizacja wag wektora kodującego odbywa się wg. zależności (3.7) jeżeli natomiast klasy wektora kodującego i treningowego są różne aktualizacja wag odbywa się wg. zależności (3.8) [6].

(3.7)

(3.8)

gdzie:

* – nowy i-ty wektor wag
* – aktualny i-ty wektor wag
* – aktualny współczynnik uczenia (maleje liniowo w każdej epoce, obliczany ze wzoru:
* – j-ty wektor treningowy

### LVQ2.1

LVQ2.1 opiera się na takiej samej zasadzie klasyfikacji jak wersja pierwsza lecz różni się metodą wybierania wygranego neuronu. Podczas uczenia wybierane są dwa wektory z książki kodowej, które są najbliższymi sąsiadami wektora treningowego. To właśnie one będą podlegać modyfikacji wag. Aby jednak mogło się to dokonać muszą zostać spełnione trzy warunki:

1. Klasa najbliższego wektora kodującego musi być różna od klasy wektora treningowego,
2. Klasa drugiego najbliższego wektora musi być taka sama jak klasa wektora treningowego,
3. Wektor treningowy musi należeć do strefy wartości zwanej „oknem”, zdefiniowanym jako:

(3.9)

gdzie:

* – odległość Euklidesowa do najbliższego wektora
* – odległość Euklidesowa do drugiego najbliższego wektora
* – zmienna definująca szerokość okna (zalecana wartość zawiera się w przedziale od 0.2 do 0.3 [7])

Jeśli te warunki są spełnione, nieprawidłowy wektor kodujący jest odsunięty od wektora treningowego, podczas gdy prawidłowy wektor kodujący zbliża się do wektora treningowego, zgodnie odpowiednio z modyfikacją wag w algorytmie LVQ1 (zależności (3.7) oraz (3.8)).

Algorytm LVQ w wersji drugiej pozwala osiągnąć lepsze dokładności dopasowania, jednak **należy go stosować tylko po uprzednim zastosowaniu wersji pierwszej!** LVQ2.1 powinien być używany tylko w sposób zrównoważony, wykorzystując niewielką wartość współczynnika uczenia się i stosunkowo małą liczbę epok.

### LVQ3

Kolejną wariancją algorytmu LVQ jest wersja trzecia. Powstała z powodu problemów, które wykazuje wersja druga. Chociaż naukowcy odnotowali dobre wyniki z algorytmem LVQ2.1, zgłosili również pewne problemy. Po pierwsze dlatego, że poprawki LVQ2.1 są proporcjonalne do różnicy między wektorami kodującymi a wektorem trenującym , a korekta wektora o właściwej klasie ma znacznie większą wielkość niż poprawka na wektor o złej klasie. Rozwiązaniem tego problemu jest rozszerzenie warunków algorytmu w wersji drugiej o dodatkowy warunek sformułowany następująco:

Jeśli obydwa najbliższe wektory kodujące mają taką samą klasę jak wektor treningowy, zmodyfikuj wagi obydwu wektorów według następującego wzoru:

(3.10)

gdzie:

* – nowy i-ty oraz k-ty wektor wag
* – aktualny i-ty oraz k-ty wektor wag
* – aktualny współczynnik uczenia (maleje liniowo w każdej epoce, obliczany ze wzoru:
* – j-ty wektor treningowy
* – zmienna definująca wpływ modyfikacji na wagi (zalecana wartość zawiera się w przedziale od 0.1 do 0.5 [8])

Pozostałe zasady odnoszące się do szerokości okna oraz modyfikacji wag najbliższych wektorów kodujących, w przypadku gdy jeden z nich odpowiada klasie wektora treningowego a drugi nie, nie ulegają zmianie w porównaniu do poprzedniej wersji algorytmu. LVQ3 podobnie jak LVQ1 definiują bardziej niezawodny proces, w których wektory kodujące przyjmują właściwe wartości nawet po dłuższych okresach uczenia się. Warto w tym miejscu zauważyć, że algorytm LVQ1 modyfikuje tylko jeden wektor kodujący jednocześnie, podczas gdy LVQ2.1 oraz LVQ3 modyfikują jednocześnie dwa wektory.

### Warstwa Kohonena

* 1. **Działanie sieci nauczonej**

## Walidacja krzyżowa

# Kod programu

# Eksperymenty

* 1. **Eksperyment 1**

Znalezienie kresu górnego i dolnego

* 1. **Eksperyment 2**

Znalezienie optymalnych parametrów Lr oraz S1

* 1. **Eksperyment 3**

Znalezienie odpowiedniej ilości epok i przedziałów

# Wnioski

Obliczenia przeprowadzone „na kartce” pokrywają się z tymi obliczonymi w programie Octave. Wyniki dla poszczególnych metod plasują się następująco:

* Metoda bisekcji:
* Metoda regula falsi:
* Metoda siecznych:

Można więc stwierdzić, że największą dokładnością wykazuje się metodą siecznych. Cechuje się najmniejszą ilością kroków do osiągnięcia zbieżności, a co za tym idzie, również najmniejszą złożonością czasową. Jednakowoż metoda bisekcji jest najprostszą w obliczaniu metodą rozwiązywania równań nieliniowych.

# Bibliografia