POLITECHNIKA RZESZOWSKA

WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI

KATEDRA INFORMATYKI I AUTOMATYKI

SZTUCZNA INTELIGENCJA

PROJEKT

Temat: Zrealizować sieć LVQ uczącą rozpoznawania zapalenia nerek i zapalenia pęcherza

Karol Baran

II EFDI

L01

1. Przedstawienie problemu

Zbiory danych oraz informacje na ich temat potrzebne do nauczenia sieci neuronowej zostały pobrane ze strony:

http://archive.ics.uci.edu/ml/datasets/Acute+Inflammations

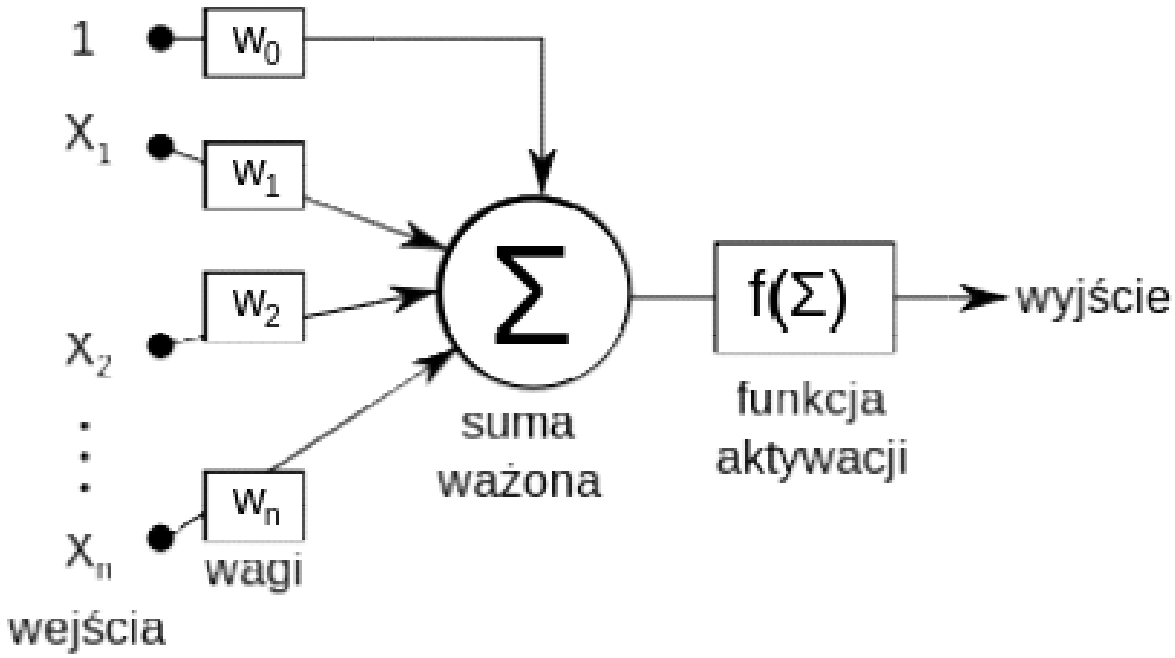
Celem projektu jest nauczenie sieci neuronowej LVQ rozpoznawania czy osoba choruje na zapalenie nerek lub zapalenie pęcherza na podstawie następujących danych:

- Temperatura pacjenta (35,5 – 42,5 °C)
- Występowanie nudności (yes,no)
- Ból lędźwiowy (yes,no)
- Ciągła potrzeba oddawania moczu (yes,no)
- Ból przy oddawaniu moczu (yes,no)
- Pieczenie cewki moczowej (yes,no)

Zbiór danych uczących zawiera 120 rekordów, 6 cech, oraz 2 wyjścia. Pierwsze informuje czy pacjent ma zapalenie pęcherza, drugie czy pacjent ma zapalenie nerek. Temperatura jest daną liczbową, reszta to dane dwuwartościowe{yes,no}

2. Opis sieci neuronowej i algorytmów uczenia

Sieć neuronowa jest modelem matematycznym, który składa się z węzłów obliczeniowych sieci zwanych neuronami oraz ich połączeń. Sieć neuronową skonstruowano na podstawie obserwacji budowy i działania mózgu. Różnicą jest to, że w mózgu neurony działają jednocześnie, a w sztucznej sieci neuronowej działają po kolei.



[https://pl.wikipedia.org/wiki/Neuron_McCullocha-Pittsa#/media/File:Neuron_McCullocha-Pittsa.svg]

Podstawową jednostką obliczeniową sztucznej sieci neuronowej jest neuron. Rysunek przedstawiający sztuczny neuron znajduje się powyżej. Sztuczny neuron składa się z:

- $X_1,X_2,\ldots,X_n$ wejść. Wejściami mogą być zarówno wejścia systemu jak i wyjścia innych neuronów. Każde wejście posiada swoją wagę, która informuje o poziomie wpływu danego wejścia na wyjście neuronu.
- Sumatora ∑, który dodaje iloczyny sygnałów wejściowych, wag oraz wartość przesunięcia w_0

$$S = \sum_{i=1}^{n} (w_i x_i) + w_0$$

- Funkcji aktywacji f, która zwraca wartość dla argumentu wyliczonego w sumatorze. Kształt funkcji aktywacji decyduje, w jakim stopniu informacja wyliczona w sumatorze jest istotna.

$$y = f(S)$$

Wartość zwrócona z funkcji aktywacji jest wyjściem neuronu y.

Uczenie sieci neuronowych

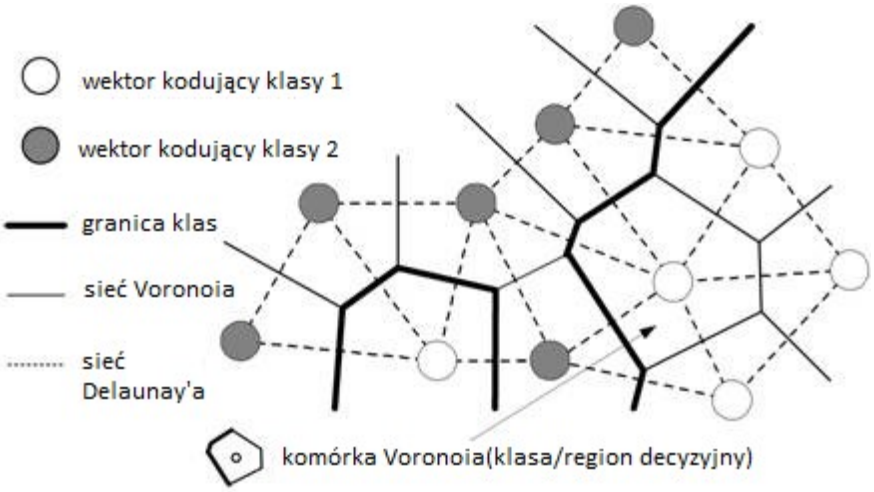
Wyróżniamy dwa główne podejścia do uczenia sieci: uczenie nadzorowane i nienadzorowane. Uczenie nadzorowane zakłada istnienie zbioru danych uczących podzielonych na wektory wejściowe i wyjściowe. Sieć ma za zadanie nauczyć się przewidywać odpowiedzi na podstawie danych uczących. Jeżeli dla danego wejścia otrzymamy odpowiedź inną niż ta z zestawu wyjść, wagi sieci są odpowiednio modyfikowane tak, żeby następnym razem przybliżyć sieć do poprawnej odpowiedzi. Natomiast uczenie nienadzorowane polega na pogrupowaniu podobnych zestawów danych.

Metoda LVQ

Algorytm LVQ jest nadzorowaną wersją algorytmu uczenia Kohonena(VQ). Wejście sieci LVQ stanowi zbiór wektorów wejściowych $X_1,X_2,\ldots,X_n$ Na początku przestrzeń danych wejściowych pokrywana jest wektorami kodującymi. Za wektor kodujący można uważać cały ukryty neuron lub wektor wag dla wszystkich wejść j-tego neuronu. Każdy z wektorów kodujących ma przypisaną etykietę klasy C_i i jest traktowany jako jej prototyp. Wektory kodujące znajdują się po środku komórki decyzyjnej nazywanej też komórką Voronoia. Zbiór wszystkich wektorów kodujących określa sie jako książkę kodowa (codebook) a cała struktura nosi nazwę mapy Kohonena. Ogólna koncepcja algorytmu bazuje na przyciąganiu do siebie wektorów kodujących przez wektory z ta sama etykieta klasy i odpychaniu od siebie wektorów o niezgodnych etykietach klas. Wykorzystywane są do tego algorytmy najbliższego sąsiada, zwycięzca bierze wszystko lub zwycięzca bierze większość.

Algorytm najbliższego sąsiada – dany jest zbiór uczący zawierający sklasyfikowane obiekty oraz obiekt, który chcemy poddać klasyfikacji. W algorytmie najbliższego sąsiada (1-NN) poszukuje sie obiektu najbliższego do obiektu badanego i na jego podstawie podejmuje sie decyzje do której klasy przyjąć nieznany obiekt.

Algorytm WTA (zwycięzca bierze wszystko) tylko zwycięski neuron podlega adaptacji w każdej iteracji. W przypadku dużej liczby neuronów lepiej sprawdza sie algorytm WTM (zwycięzca bierze większość), w którym wokół zwycięskiego neuronu definiuje sie topologiczne sąsiedztwo (okno) i adaptacji podlegają również wszystkie neurony należące do sąsiedztwa.



[http://www.neural-forecasting.com/lvq_neural_nets.htm]

Topologia sieci LVQ

Sieć LVQ charakteryzuje się bardzo prostą topologią. Jest to sieć dwuwarstwowa typu feedforward(nie ma połączenia pomiędzy wyjściami a wejściami sieci). Warstwa ukryta sieci nazywana jest warstwą Kohonena lub warstwą konkurencyjną a jej neurony wektorami kodującymi. Druga warstwa sieci nazywa się warstwą wyjściową lub liniową. W warstwie wyjściowej jest dokładnie tyle neuronów co możliwych wyjść sieci, a ilość neuronów w warstwie Kohonena nie może być mniejsza niż ilość neuronów w warstwie wyjściowej.

Wersje algorytmu LVQ

LVQ 1 - W trybie testowania do określenia podobieństwa wektorów oblicza się odległość od wektora wejściowego od wszystkich wektorów kodujących i wyłaniany jest zwycięzca jako ten leżący najbliżej wektora wejściowego. Najczęściej stosowaną normą jest norma euklidesowa:

$$\boldsymbol{d_i} = \left|\left|\boldsymbol{w_j} - \boldsymbol{x}\right|\right| = \sqrt{\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \left(\boldsymbol{w_{ji}} - \boldsymbol{x_i}\right)^2}$$

W skład zbioru danych wchodzi wektor T, w którym zapisane są poprawne klasy wektorów x. Dzięki niemu sieć może sprawdzić, czy zwycięski neuron należy do badanej klasy i zmienić jego wagi. Zmiana wag powoduje odpowiednio przyciąganie lub odpychanie neuronu od danej klasy. Wagi pozostałych neuronów pozostają bez zmian. Wagi zwycięskiego neuronu są modyfikowane zgodnie ze wzorem

$$\begin{cases} w_k = w_{k-1} + \eta(x - w_{k-1}) & dla\ zwycięzkiego\ neuronu\ należącego\ do\ klasy \\ w_k = w_{k-1} - \eta(x + w_{k-1}) & dla\ zwycięzkiego\ neuronu\ nie\ należącego\ do\ klasy \end{cases}$$

Po modyfikacji wag procedura jest powtarzana. Tym sposobem neurony dzielą się na grupy, które reprezentują poszczególne klasy.

Algorytm LVQ1 jest najprostszym stosowanym w sieciach LVQ. W niektórych przypadkach nie pozwala on na osiągnięcie zadowalającej dokładności sieci. W przypadku błędnego, gdy z powodu

losowego ustawiania wag początkowych dany wektor kodujący jest błędnie zlokalizowany algorytm ten nie posiada mechanizmów poprawiających jakość i szybkość pozbywania się takich złych przyporządkowani. Jego atutem jest natomiast szybki spadek błędu klasyfikacji w początkowych fazach uczenia- w związku z tym zaleca się go do początkowych faz uczenia.

LVQ 2

W algorytmie LVQ 2, wybierany jest zwycięzca biorąc pod uwagę odległość dj oraz drugi posiadający najmniejszą(oprócz zwycięscy) odległość o wektora wejściowego di. Wagi obu neuronów są aktualizowane, jeżeli zwycięzcy należą do różnych klas, aktualnie badany wektor wejściowy x należy do tej samej klasy co drugi zwycięzca oraz odległość d_j i d_i są do siebie zbliżone, co określa się wzorem:

$$\frac{d_i}{d_j} > (1-\varepsilon) \ \wedge \ \frac{d_j}{d_i} < (1+\varepsilon)$$

Jeżeli zachodzą te warunki to wagi neuronów modyfikowane są według wzorów:

$$\begin{cases} w_j = \ w_{j-1} - \eta\left(x - w_{j-1}\right) \ dla \ pierwszego \ zwycięzcy \\ w_i = \ w_{i-1} + \eta(x - w_{i-1}) \ dla \ drugiego \ zwycięzcy \end{cases}$$

LVQ 2.1

Zasada działania algorytmu LVQ 2.1 jest taka sama jak algorytmu LVQ 2 z wyjątkiem wzoru, który określa zbliżenie zwycięzcy globalnego i lokalnego. W tym algorytmie wzór ten ma postać:

$$\min\left[\frac{d_j}{d_i}, \frac{d_i}{d_j}\right] > (1-\varepsilon) \ \wedge \ \max\left[\frac{d_j}{d_i}, \frac{d_i}{d_j}\right] < (1+\varepsilon)$$

LVQ 3

Ta wersja algorytmu uwzględnia sytuację, gdy wektor kodujący znajduję się w optymalnej pozycji, i jego pozycja nie wymaga usprawnienia. Algorytm szuka dwóch najbliższych neuronów W_i i W_j o najmniejszych odległościach d_i i d_j. Następnie sprawdzany jest warunek:

$$\min\left[\frac{d_j}{d_i}, \frac{d_i}{d_j}\right] > (1-\varepsilon)(1+\varepsilon)$$

Jeżeli jeden z wektorów należy do badanej klasy, a drugi nie to modyfikacja wag przebiega tak samo, jak w przypadku LVQ 2.1. Jeżeli oba wektory należą do badanej klasy to wagi obu neuronów modyfikuje się w następujący sposób:

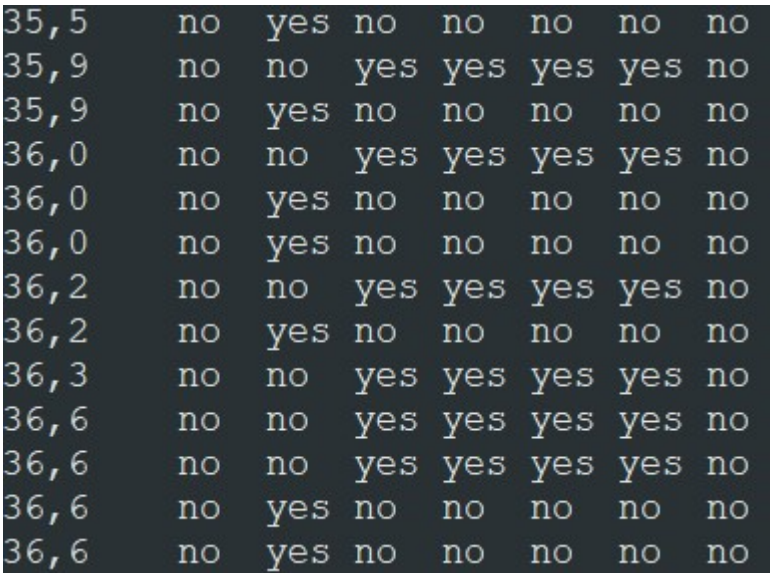
$$w_j = w_{j-1} + m\eta(x - w_{j-1})$$

gdzie $m \in (0.1; 0.5)$

Realizacja praktyczna

Przygotowanie danych

Po pobraniu z adresu http://archive.ics.uci.edu/ml/machine-learning-databases/acute/diagnosis.data pliku z danymi uczącymi dane są zapisane w następującej formie:



```
35,5    no  yes no  no  no  no  no
35,9    no  no  yes yes yes yes no
35,9    no  yes no  no  no  no  no
36,0    no  no  yes yes yes yes no
36,0    no  yes no  no  no  no  no
36,0    no  yes no  no  no  no  no
36,2    no  no  yes yes yes yes no
36,2    no  yes no  no  no  no  no
36,3    no  no  yes yes yes yes no
36,6    no  no  yes yes yes yes no
36,6    no  no  yes yes yes yes no
36,6    no  yes no  no  no  no  no
36,6    no  yes no  no  no  no  no
```

Żeby można było wykorzystać te dane do uczenia w programie matlab 6.5 należy zamienić dane tekstowe na liczbowe oraz znormalizować pierwszą kolumnę.

Zamianę słowa „no” na liczbę 0 oraz słowo „yes” na liczbę 1. Normalizację pierwszej kolumny wykonano w programie Excel według wzoru:

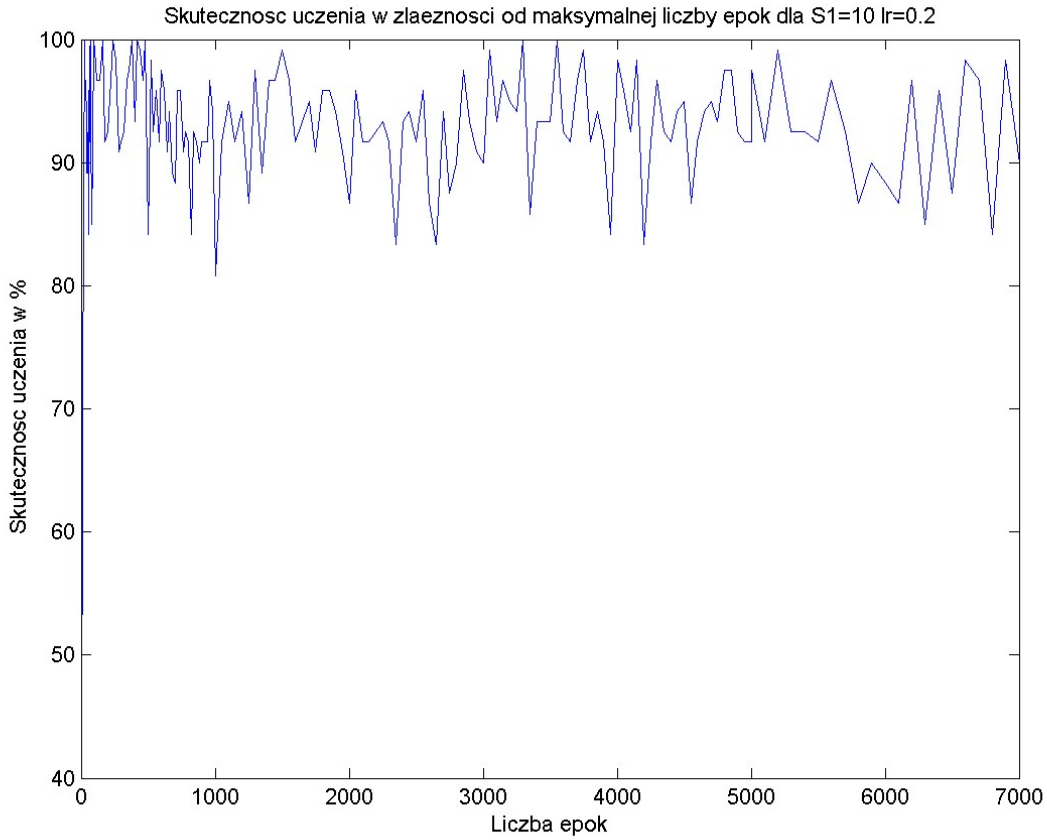
$$y' = \frac{y - y_{min}}{y_{max} - y_{min}}\left(y_{nowy\_max} - y_{nowy\_min}\right) + y_{nowy\_min}$$

Należało jeszcze złączyć dwa wyjścia(dwie ostatnie kolumny) w jedną gdyż funkcja ind2vec musi przyjmować wektor. Zamieniono to w następujący sposób:

Wyjście 1	Wyjście 2	Nowe wyjście	Interpretacja
0	0	1	Badana osoba jest zdrowa
0	1	2	Badana osoba ma zapalenie nerek
1	0	3	Badana osoba ma zapalenie pęcherza
1	1	4	Badana osoba ma obie choroby

Eksperyment 1

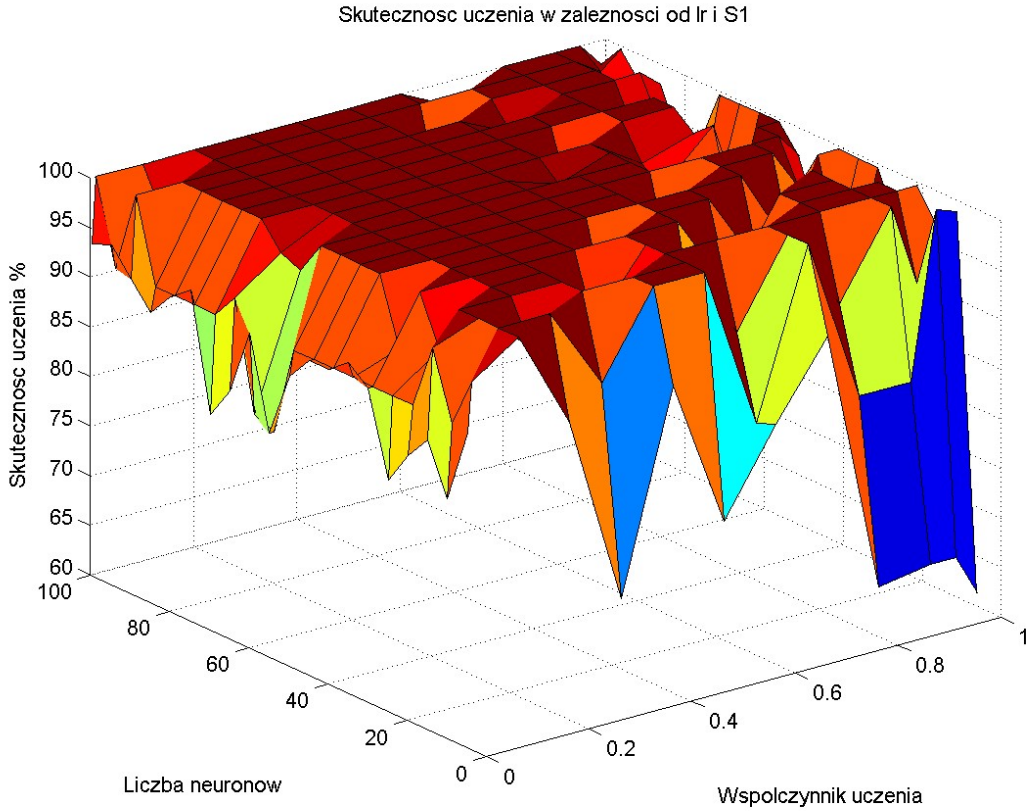
Zbadano zależność skuteczności uczenia od liczby epok dla stałej ilości neuronów w warstwie ukrytej S1 oraz stałego współczynnika uczenia lr oznaczonego w wzorach jako η

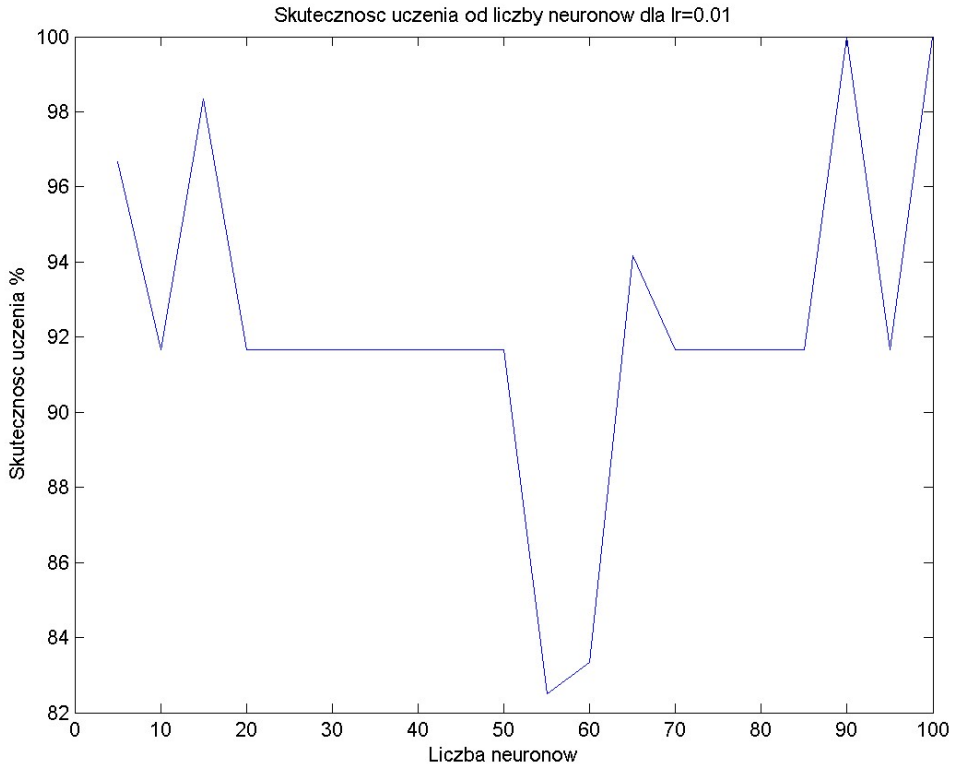


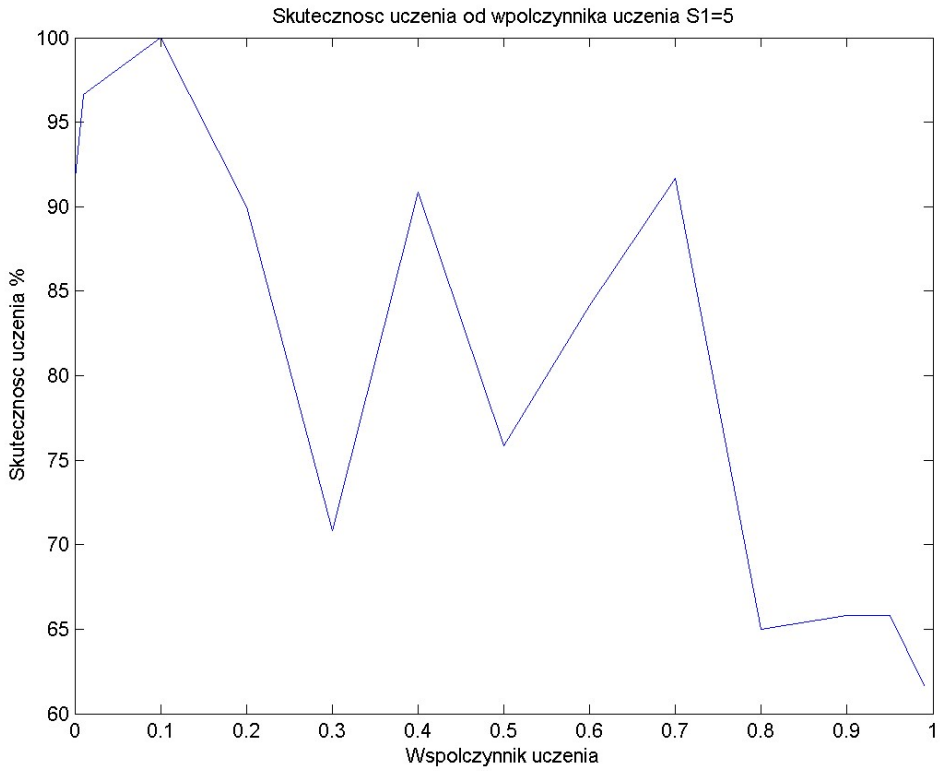
Eksperyment pokazał, że duża liczba epok nie wpływa na skuteczność uczenia. Bez zmiany innych parametrów skuteczność oscyluje pomiędzy wartościami 80% do 100%. Następne eksperymenty zostaną wykonane na małej liczbie epok.

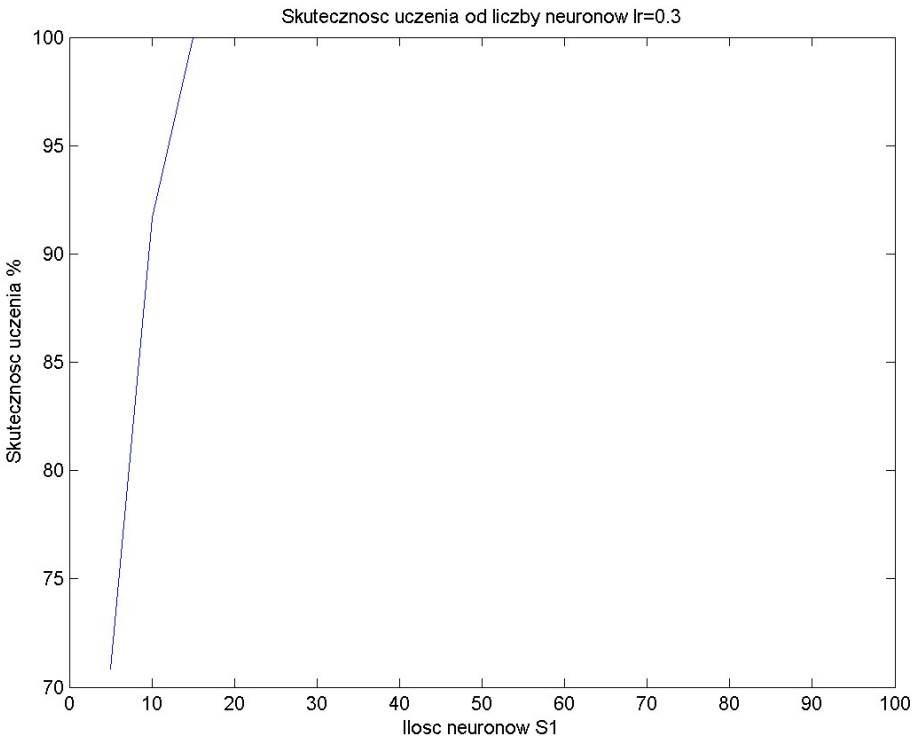
Eksperyment 2

Wykonano wykres skuteczności uczenia od lr i S1







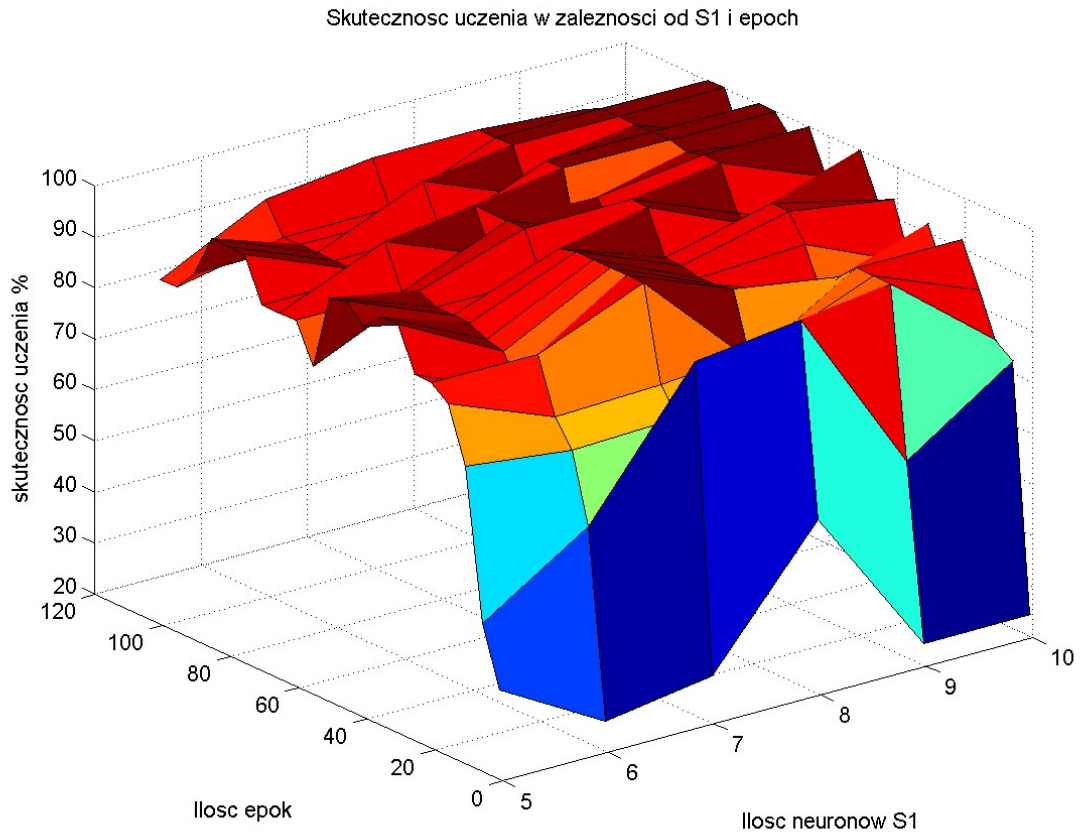


Eksperyment pokazał, że sieć neuronową można nauczyć do 100% skuteczności w bardzo szerokim zakresie parametrów (ciemno-czerwony obszar na pierwszym wykresie). Pokazuje on również, że

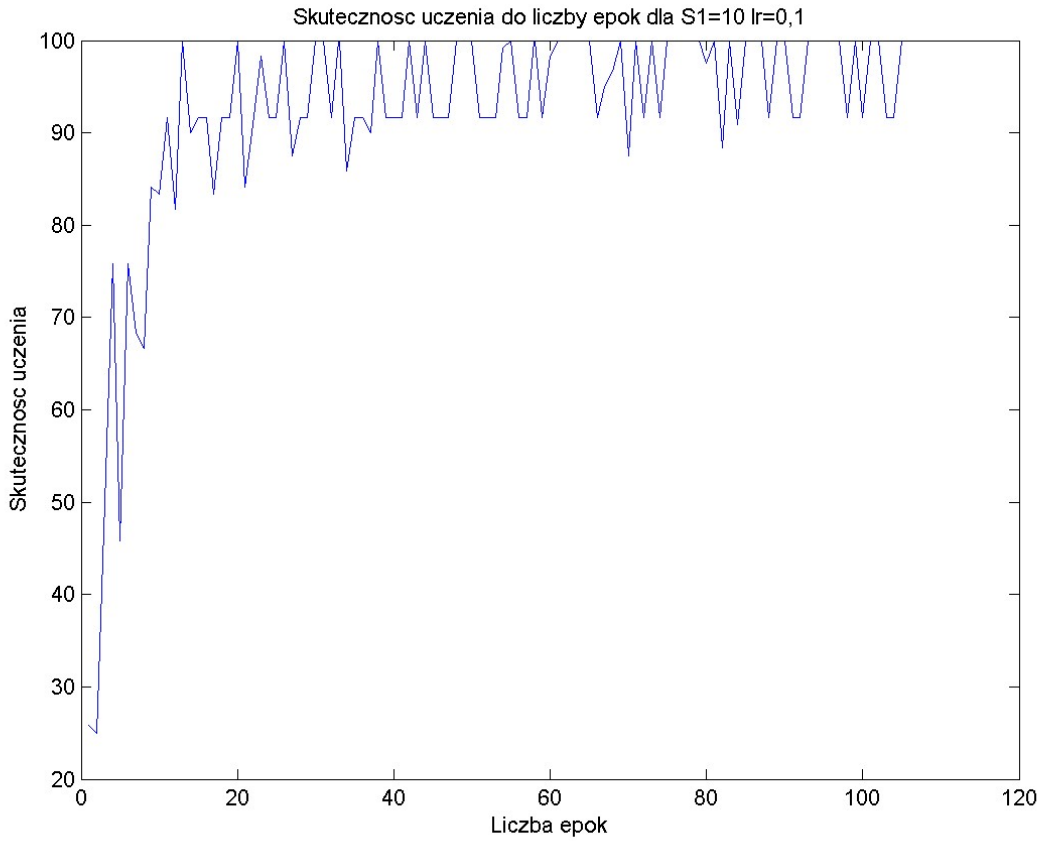
stosowanie dużych współczynników uczenie przy małej ilości neuronów daje najgorsze rezultaty. Drugi wykres to rzut pierwszego względem współczynnika uczenia równego 0,01. Wskazuje on na to, że nie można jednoznacznie stwierdzić, że zwiększanie liczby neuronów będzie prowadzić do wzrostu skuteczność (minimum przy 60 neuronach). Przedostatni wykres pokazuje, że da się osiągnąć maksymalną skuteczność już przy współczynniku uczenia równym 0,1 i 5 neuronach w pierwszej warstwie. Pokazuje on również tendencję spodkową przy zwiększaniu lr. Przy poszukiwaniu najlepszych parametrów powinienem skupić się na małych współczynnikach uczenia. Najlepszym współczynnikiem uczenia okazał się 0,3 gdyż przy zmieniającej się liczbie neuronów najwięcej razy osiągnął maksymalną skuteczność

Eksperyment 3

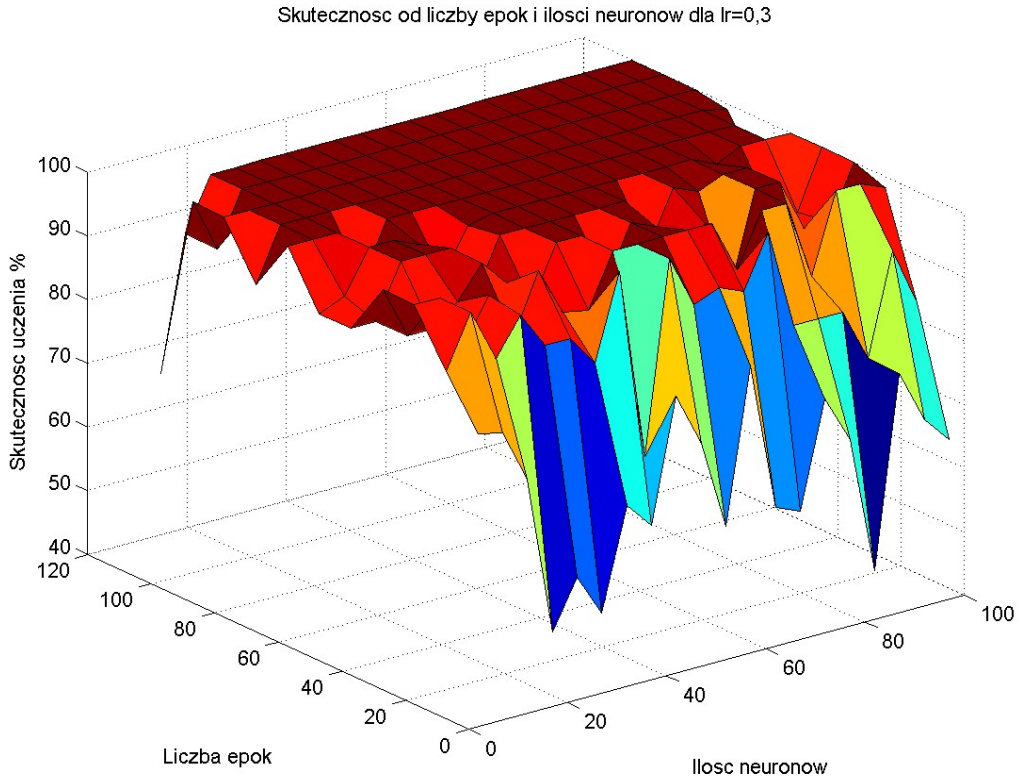
W tym eksperymencie będę starał się znaleźć liczbę epok i minimalną liczbę neuronów dla maksymalnej skuteczności, ponieważ im mniejsza liczba neuronów tym mniejsza szansa, że sieć się przeuczy, tzn. będzie bezbłędnie klasyfikować dane zbioru uczącego, ale dla danych nie należących do zbioru będzie miała znaczący błąd.



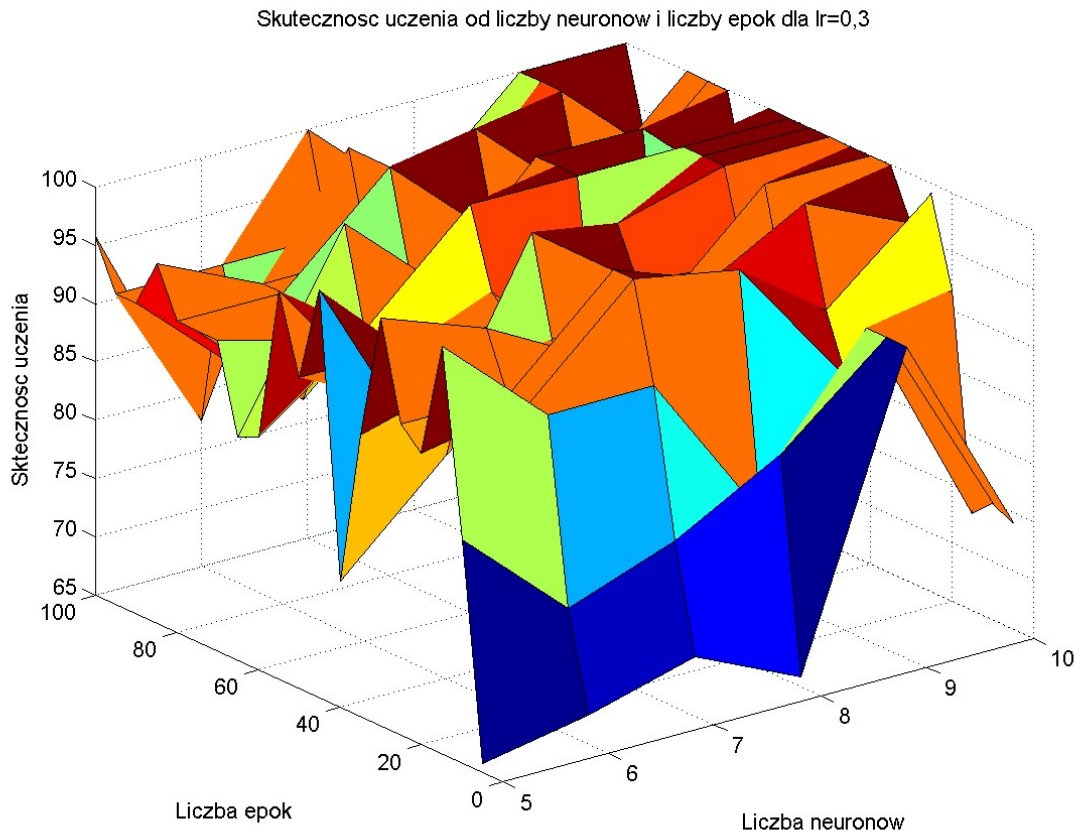
Najszybciej dla lr=0,1 osiągnięto skuteczność 100% dla S1=10 już w 13 epokach



Wykres dla lr=0,3



I dla mniejszej liczby neuronów



Z powyższych wykresów widać, że współczynnik uczenia 0,3 jest lepszy od 0,1 dla dużej ilości neuronów i epok. Dla niskiej ilości neuronów lepiej radzi sobie lr=0,1.

Skrypt

```
% wyczyszczenie przestrzeni ekranowej
clear all

% wyłączanie ostrzeżeń, np. o przestarzałych funkcjach
nntwarn off

format compact

% wczytanie pliku z danymi uczącymi
 load dane

 % transpozycja macierzy
 P = P';
 T = T';
 % liczba neuronów w pierwszej warstwie
 nr=1;

 for S1 = [5 : 5 : 100]
    for lr = [0.001 0.01  0.1 : .1 : .9 .95 .99]
        for epoch = [5:10:105]
            % przekształcenie w macierz zero-jedynkową
```

```
            Tv = ind2vec(T);

            % parametry treningowe
            TP = [5 epoch lr];

            % inicjalizacja sieci LVQ
            [w1, w2] = initlvq(P, S1, Tv);

            % trening sieci
            [w1, w2] = trainlvq(w1, w2, P, Tv, TP);

            % symulacja sieci
            a = simulvq(P, w1, w2);

            % przekształcenie macierzy zero-jedynkowej w wektor indeksowy
            a = vec2ind(a);

            % obliczenie skuteczności uczenia się
            skutecznosc = (1-sum(abs(T-a)>0.5)/length(P))*100;

            tab_w(nr,1) = S1;
            tab_w(nr,2) = lr;
            tab_w(nr,3) = epoch;
            tab_w(nr,4) = skutecznosc;
            nr = nr+1;
        end
    end
end
tablica=[tab_w];
save w tablica
```

Opis funkcji użytych w skrypcie

ind2vec(T) – zamienia wektor P w macierz zero-jedynkową np. wektor w=[1 3 2 4] zamieni na macierz o dla indeksów (ind,w(ind)) będzie przyjmowała 1 w przeciwnym razie 0.

Initlvq(P, S1, Tv) - przyjmuje dane wejściowe i zwraca ustawienia inicjalizacji współczynników wagowych warstw konkurencyjnej i liniowej

trainlvq(w1, w2, Pn, Tv, TP) - szkolenie sieci (W1 - wagi na konkurencyjnej warstwie, W2 - wagi warstwy liniowej, Tv - macierz docelowa, TP - parametry treningowe)

simulvq(Pn, w1, w2) – używane do symulacji wytrenowania sieci,

vec2ind – zwraca indeksy współrzędnych wektorów kolumnowych, które posiadają jednakowa wartość, a pozostałe współrzędne każdego wektora są zerowane.

Wnioski

Do dokładnego znalezienia parametrów sieci korzystałem z wielu doświadczeń, a te zawarte w projekcie mają tylko pokazać ogólne zmiany, gdyż nie było sensu zawierania większej ilości wykresów Wstępne eksperymenty pokazują, że nie ma sensu robić dalszych eksperymentów dla dużych zakresów zmiennych, gdyż nie poprawiają one zbieżności metody. Wpływ parametrów na siec ostatecznie okazał się nie tak prosty jak oczekiwałem na początku doświadczeń i czas

przeprowadzania samych symulacji przerósł moje oczekiwania, gdyż zajęło to dziesiątki godzin. Eksperyment 1 pokazał, że liczba epok ma niewielki wpływ na dokładność uczenia. Kluczowy wpływ mają liczba neuronów w warstwie Kohenena oraz współczynnik uczenia. A mianowicie skuteczny trening sieci odbywał się zarówno przy niskiej(dla wybranych współczynników uczenia) jak i wysokiej liczby neuronów. Jeśli chodzi o współczynnik uczenia to sieć dawała najlepsze wyniki gdy lr był w pewnym przedziale(ok. od 0.2 do 0.4). Z powodu małej ilości danych wejściowych udało się osiągnięcie stuprocentowej skuteczności nie było trudne.

Bibliografia

Leszek Rutkowski „Metody i techniki sztucznej inteligencji”

http://home.agh.edu.pl/~asior/stud/doc/LVQ_CP_10.pdfW

http://ccy.dd.ncu.edu.tw/~chen/course/neural/ch4/index.htm

https://pl.wikipedia.org/wiki/Neuron_McCullocha-Pittsa

https://www.ii.uni.wroc.pl/~aba/teach/NN/w7som.pdf