Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Oświadczam, że niniejsza praca, stanowiąca podstawę do uznania osiągnięcia efektów uczenia się z przedmiotu Sieci neuronowe w zastosowaniach biomedycznych, została wykonana przeze mnie samodzielnie.

Ewa Mergo 300054

Karolina Olszewska 299697

Sieci neuronowe
w zastosowaniach biomedycznych
Projekt etap II

Temat 25: Przewidywanie przewlekłej niewydolności nerek za pomocą sieci MLP

Wykonawca: Ewa Mergo, Karolina Olszewska - zespół 24

Opiekun projektu: Dr. inż. Paweł Mazurek

Warszawa 15.05.2022 r

Spis treści

1. Przetwarzanie danych przed podaniem ich na wejście sieci

Na początku wczytywane do Matlaba są dane. Dane surowe poddane zostały normalizacji min-max, która pozwoliła na „przeskalowanie" tych danych do zakresu [0, 1]. Następnie dane te są dzielone na dane treningowe (80%) oraz testowe (20%) . Normalizacja min-max określa o ile dana wartość jest większa od minimalnej i skaluje tę różnicę przez zakres zgodnie ze wzorem:

$$\bar{X} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

Gdzie: *X* – wartość pierwotnej danej; *min(X)* – wartość minimalna z zakresu badanej danej; *max(X)* – wartość maksymalna z zakresu badanej danej

Dane treningowe składają się z 200 wektorów dla pacjentów chorych i 120 dla zdrowych. Dane testowe to 50 wektorów dla pacjentów chorych i 30 dla zdrowych. Ostatnia kolumna danych zawiera informację, czy dany pacjent choruje na przewlekłą niewydolność nerek czy też nie (wektor kolumnowy stanowiący neuron wyjściowy sieci).

2. Opis implementacji sieci neuronowej

Implementacja sieci neuronowej została przeprowadzona w środowisku Matlab.

Na początku wczytywane do Matlaba są dane oraz są dzielone na dane treningowe i testowe. Następnie dobierane są niewielkie wartości w oraz ω, które zostały dobrano losowo dzięki użyciu generatora liczb losowych. W kodzie wagi zapisane są w zmiennych „w" oraz „omega". Liczba iteracji i współczynnik uczenia się sieci zostały zapisane w zmiennych „K" oraz „eta", których wartości ustalone na podstawie przeprowadzonych testów i wynoszą eta = 0,001 i K = 10 000. Na początku każdej iteracji losowany był wektor uczący z danych treningowych. Do uczenia sieci wykorzystano algorytm wstecznej propagacji błędów (ang. error back-propagation), który został szczegółowo opisany w dalszej części sprawozdania. Dzięki liczeniu wartości pobudzeń, stanów wyjść, sygnałów błędu oraz wag dla warstwy ukrytej i wyjściowej (w kodzie są to wartości „ni", „ksi", „v", „y", „delta_o", „delta_h", „w" i „omega") można było wyznaczyć błąd wzorca w każdej iteracji (wartość „E").

Następnie na podstawie wyznaczonych parametrów przeprowadzono testy poprawności działania sieci i wyznaczono miary charakteryzujące, czyli czułość i specyficzność.

W Matlabie zaimplementowano 3 funkcje. Pierwszą z nich jest funkcja progowania (y = prog(x)) stosowana, by zaklasyfikować odpowiednio obliczone wartości na wyjściu neuronów warstwy wyjściowej (czy należą one do pacjenta chorego (0), czy do zdrowego (1)). Kolejną funkcją jest funkcja aktywacji neuronów (sigmoid_unipolar(x)). Zastosowano tez funkcję, obliczającą pochodna z sigmoidalnej funkcji aktywacji (derivative(x)). Służy ona do obliczania sygnałów błędów dla wartwy ukrytej i wyjściowej.

3. Kod źródłowy zaimplementowanej sieci neuronowej

```
close all
clear all
clc

%% training and testing data

snb_data=readtable("snb_normdata"); % odczytanie danych z pliku xlsx
x=table2array(snb_data(1:400,1:15)); % zamiana tablicy z danymi na macierz
t=table2array(snb_data(1:400,16));   % zamiana tablicy z informacją, czy pacjent jest chory (ckd/notckd)
na wektor
xtrain=[x(1:200,:); x(251:370,:)];   % dane treningowe - 200 wektorów dla pacjentów chorych i 120 dla
pacjentów zdrowych
xtest=[x(201:250,:); x(371:400,:)]; % dane testowe - 50 wektorów dla pacjentów chorych i 30 dla pacjentów
zdrowych
ttrain=[t(1:200,:); t(251:370,:)];   % dane treningowe - 200 informacji, że pacjent jest chory i 120,
że jest zdrowy - kompatybilne z macierzą xtrain
ttest=[t(201:250,:); t(371:400,:)] ; % dane testowe - 50 informacji, że pacjent jest chory i 30, że jest
zdrowy - kompatybilne z macierzą ttrain

%% learning
w = (rand(4,15)*10-1)/100;        % inicjacja macierzy w z niewielkimi początkowymi losowymi wartościami
wag
omega = (rand(1,4)*10-1)/100;          % inicjacja wektora omega z niewielkimi początkowymi losowymi
wartościami wag

K = 10000;     % liczba iteracji
eta = 0.001;   % współczynnik uczenia sieci

for k=1:K
    i = randi([1,320]);  %wybór losowego wektora uczącego (z wektorów treningowych)
    ni = w * xtrain(i,:)';  % obliczenie pobudzenia neuronów warstwy ukrytej
    ksi = sigmoid_unipolar(ni); % obliczenie stanu wyjść neuronów warstwy ukrytej
    v = omega * ksi; % obliczenie pobudzenia neuronów warstwy wyjściowej
    y = sigmoid_unipolar(v);    % obliczenie stanu wyjść neuronów warstwy wyjściowej
    delta_o = (ttrain(i)-y) * derivative(v); % obliczenie sygnału błędu dla warstwy wyjściowej
    delta_h = derivative(ni) .* (omega * delta_o)'; % obliczenie sygnału błędu dla warstwy ukrytej
    omega = omega + eta * delta_o * ksi';   % zmodyfikowanie wagi warstwy wyjściowej
    w = w + eta * delta_h * xtrain(i,:);   % zmodyfikowanie wagi warstwy ukrytej
    E(k) = 0.5 * (ttrain(i)-y)^2;    % obliczenie poziomu błędu
end

%% testing

for l=1:80
    ni = w * xtest(l,:)';  % pobudzenia neuronów warstwy ukrytej
    ksi = sigmoid_unipolar(ni); % stan wyjść neuronów warstwy ukrytej
    v = omega * ksi; % pobudzenia neuronów warstwy wyjściowej
    y(l) = prog(sigmoid_unipolar(v)); % stan wyjść neuronów warstwy wyjściowej
end

%% plots

plot(y,'g*');
hold on;
plot(ttest,'c--');
hold off;
title('Proces uczenia sieci - test');
ylabel('ckd = 0 / notckd = 1');
xlabel('Liczba iteracji');
legend('Wartosci testowe', 'Wartosci prawdziwe','Location','northwest');

%% statistics
TPR=(50-sum(y(1:50)))/50 % czułość

SPC=sum(y(51:80))/30 % specyficzność

%% Wykres błędu
blad=figure();
plot(1:K, E(1:K));
set(blad,'Position',[1 1 2000 1000]);
title('Przebieg bledu w zaleznosci od liczby iteracji');
xlabel('Liczba iteracji');
ylabel('Wartosc bledu');

%% functions
```

```
function y = prog(x) % funkcja progowania
   if x>0.5
       y=1;
   else
       y=0;
   end

end
function y = sigmoid_unipolar(x) % funckja aktywacji - sigmoidalna unipolarna
   beta = 10;
   y = 1./(1+exp(-beta*(x)));
end

function y = derivative(x) %pochodna funkcji aktywacji
   beta = 10;
   y=(beta *exp(-beta*(x)))./(1 + exp(-beta*(x))).^2;
end
```

4. Struktura sieci

4.1. Opis struktury sieci

W procesie modelownia sieci neuronowej została określona liczba warstw sieci. Zamodelowana struktura składała się z trzech warstw: warstwy wejściowej, warstwy ukrytej i warstwy wyjściowej. Warstwy reprezentowane są przez:

Warstwa wejściowa – 15 neuronów
Warstwa ukryta – 4 neurony
Warstwa wyjściowa – 1 neuron

Liczba neuronów w warstwie ukrytej została wyliczona z zależności:

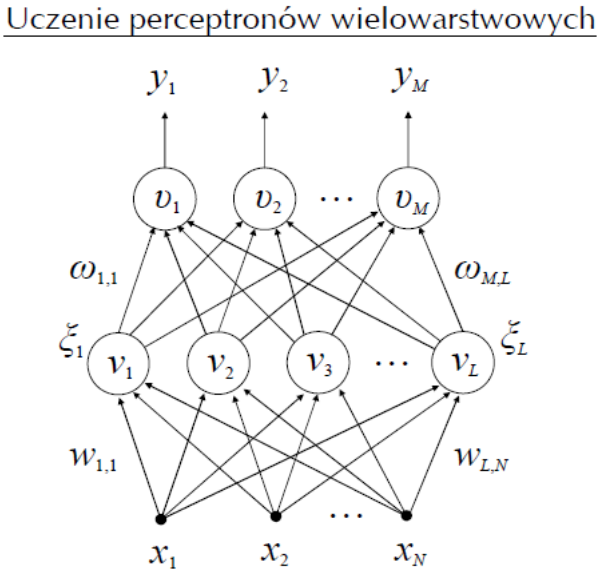
$$N_{ukryte} = \sqrt{N_{WE} * N_{WY}}$$

N_{WE} – liczba neuronów warstwy wejściowej

N_{WY} – liczba neuronów warstwy wyjściowej

Każdy neuron z warstwy poprzedniej posiada połączenie z neuronem warstwy następnej z odpowiednią wagą wyznaczoną w procesie trenowania sieci.

Wstępnie nie planujemy użycia polaryzacji, jednak w trakcie dalszych prac może się okazać, że zastosowanie offset'u może być korzystne i przyniesie lepsze wyniki.



Rysunek 1. Struktura modelowanego perceptronu wielowarstwowego.

4.2. Funkcja aktywacji

W procesie uczenia do algorytmu sieci użyta została ciągła funkcja sigmoidalna unipolarna jako funkcja aktywacji neuronów. Została ona dobrana na podstawie oczekiwanych wartości wyjściowych.

Funkcja ta opisana jest wzorem:

$$y_m = \frac{1}{1 + \exp(-\beta \cdot v_m)}$$

Współczynnik β wynosi 10.

5. Algorytm uczenia sieci neuronowej

Proces uczenie wielowarstwowej sieci MLP został przeprowadzony za pomocą wstecznej propagacji błędów. Jest to algorytm pozwalający na wyznaczenie błędu generowanego przez neurony warstw ukrytych na podstawie błędów warstwy wyjściowej, a następnie wykorzystanie go do poprawy wartości wag neuronów.

Błąd ten określany jest na podstawie sumy kwadratów błędów na wyjściach i wyliczamy za pomocą wzoru dla k-tego wzorca.

$$E^k = \frac{1}{2} \cdot \sum_{m=1}^{M} \left(d_m^k - y_m^k\right)^2$$

Algorytm rozpoczyna się od przyjęcia wartości wag dla połączeń: $w_{l,n}$ dla warstwy wejściowej oraz $\omega_{m,l}$ dla warstwy ukrytej. Wartości te powinny być losowe i niewielkie. Następnie na wejście sieci zostanie podany wejściowy wektor cech zbioru danych treningowych.

$$x^k = [\, x_1^k \; x_2^k \; x_3^k \; \ldots \; x_n^k]^T$$

Na tej podstawie wyznaczane jest pobudzenie neuronów warstwy ukrytej sieci neuronowej. Pobudzenie jest sumą iloczynu skalarnego wektora wag oraz wektora cech wejściowych:

$$v^k = \sum_{n=1}^{N} w_{l,n} \cdot x_n^k$$

Pobudzenie warunkuje stan wyjść neuronów poprzez zastosowanie funkcji aktywacji warstwy ukrytej:

$$\xi_l^k = f\left(v_l^k\right)$$

Kolejnym krokiem jest obliczenie pobudzenia neuronów warstwy wyjściowej jako sumy iloczynu skalarnego wektora wag warstwy ukrytej i wektora stanu wyjść wyznaczonego w poprzednim kroku:

$$v_m^k = \sum_{l=1}^{L} w_{m,l} \cdot \xi_l^k$$

Następnie obliczany jest stan wyjść neuronów warstwy wyjściowej na podstawie jej pobudzenia:

$$y_m^k = f\left(v_m^k\right)$$

Po zakończeniu obliczania stanu wyjść warstw sieci przystępujemy do wyznaczenia błędów pojawiających się w kolejnych warstwach, rozpoczynając od warstwy wyjściowej, dla której sygnał błędu ma postać:

$$\delta_m^{(o)k} = \left(d_m^k - y_m^k\right) \cdot f'\left(v_m^k\right)$$

Zgodnie z ideą wstecznej propagacji błędu wyznaczamy sygnał błędu dla warstwy ukrytej:

$$\delta_l^{(h)k} =\cdot f'(v_l^k) \cdot \sum_{m=1}^{M} \left(\omega_{m,l} \cdot \delta_m^{(o)k}\right)$$

Na podstawie wyznaczonych błędów przy użyciu metody gradientowej modyfikujemy wektor wag warstwy wyjściowej, który przyjmie wartości zgodne ze wzorem:

$$\omega_{m,l}(t+1) = \omega_{m,l}(t) + \eta \cdot \delta_m^{(o)k} \cdot \xi_l^k$$

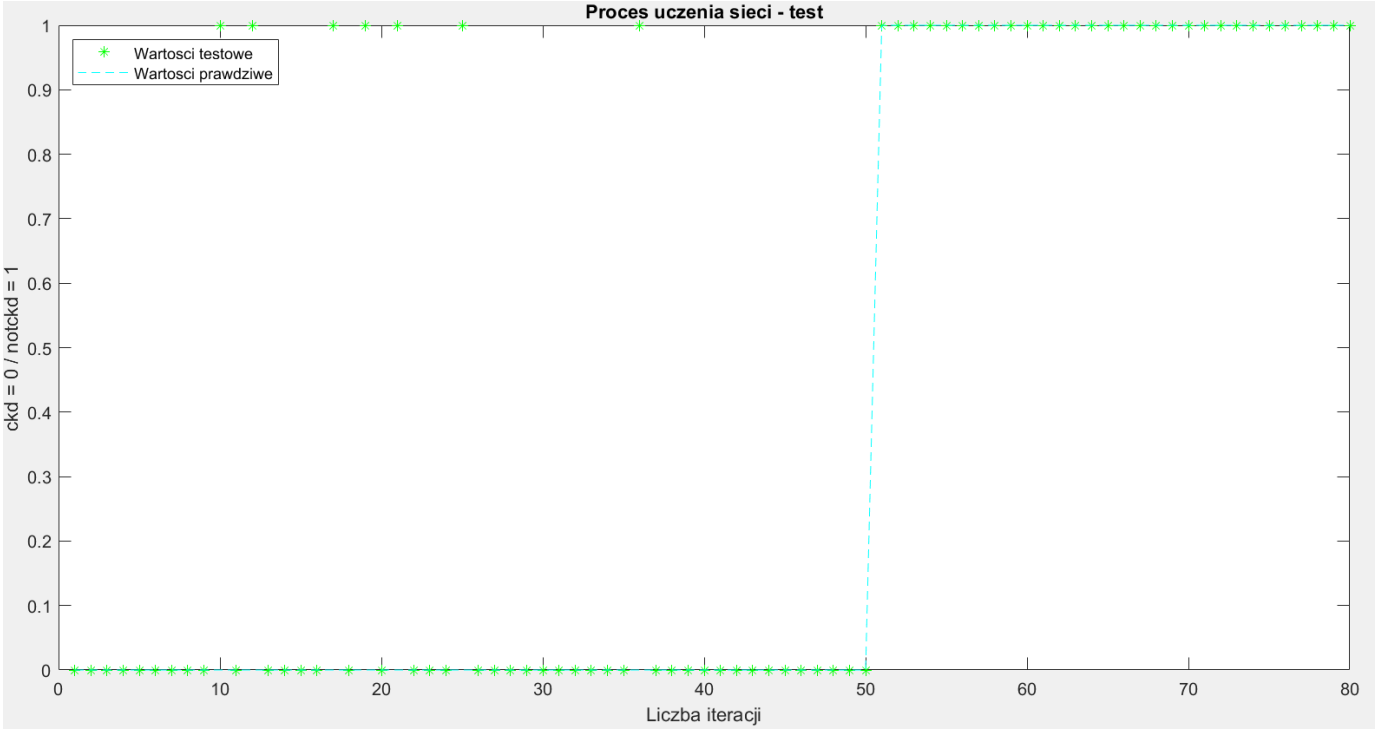
oraz wektor wag warstwy ukrytej, który zostanie uaktualniony zgodnie ze wzorem:

$$w_{l,n}(t+1) = w_{l,n}(t) + \eta \cdot \delta_l^{(h)k} \cdot \mathrm{x}_n^k$$

Proces jest powtarzany wielokrotnie – liczbę iteracji K, która została dobrana na podstawie testów sieci.

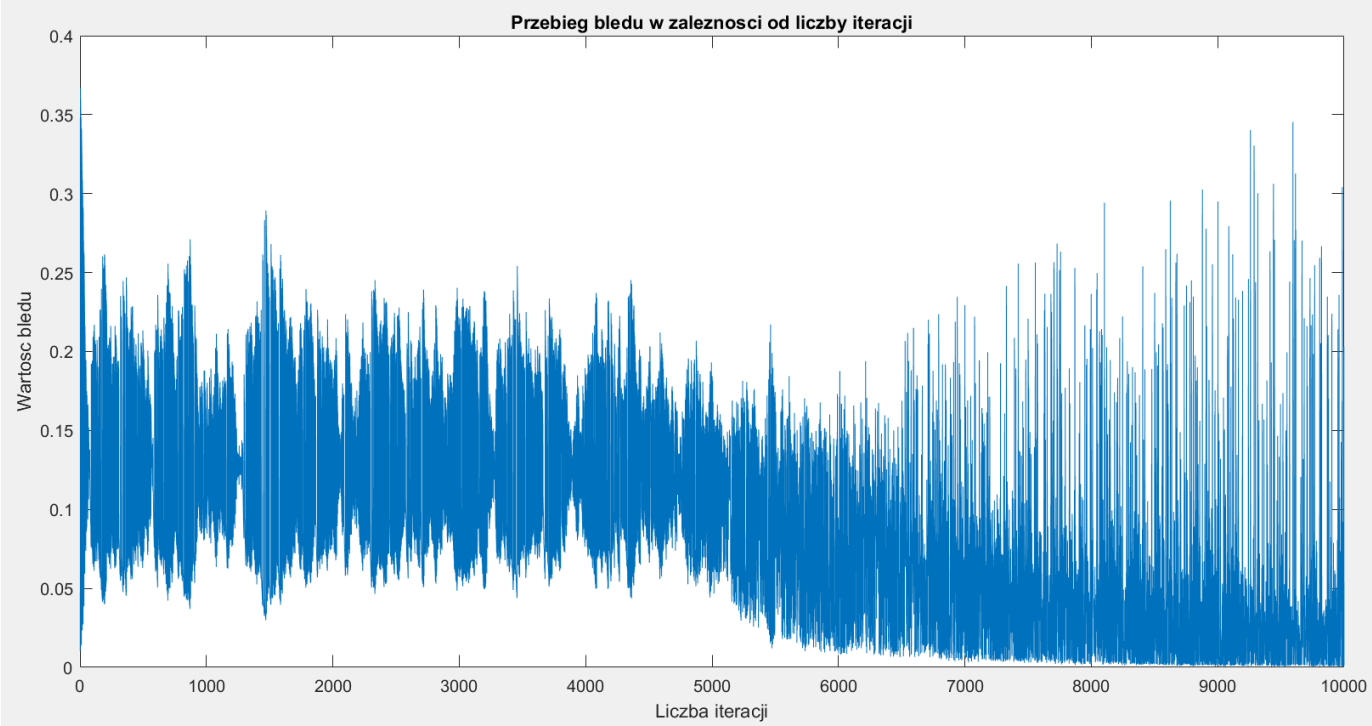
6. Wykresy obrazujące proces uczenia sieci

Na poniższym wykresie zaprezentowano efekty uczenia się sieci neuronowej uzyskane w teście tej sieci oraz proces kształtowania się wartości błędu wyliczanego w każdej z przeprowadzanych iteracji.



Rysunek 2. Test procesu uczenia się sieci neuronowej

Efekt uczenia się sieci jest zadowalający – na wykresie widać, że sieć ta przyjmuje w większości prawidłowe wartości (0 lub 1). Jedynie dla kilku iteracji zamiast wartości zerowej (która świadczyłaby o chorobie przeklekłej niewydolności nerek) ma wartośc jedynkową (dla osoby zdrowej). Rozbieżność ta może wynikać z wartości niektórych dnaych treningowych – składają się one z wyników różnych badań (opisanych w sprawozdaniu z etapu I). Chory pacjent w większości przypadków nie posiada wszystkich patologicznych cech charakterystycznych dla danej choroby. Zatem niektóre wyniki badań mogą wskazywać na wartości w granicach normy pomimo choroby. Wyniki te „zafałszowują" wektory osób chorych i dlatego sieci interpteruje ten zestaw danych (w danym wektorze) jako zestaw danych dla osoby zdrowej.



Rysunek 3. Przebieg funkcji błędu dla k-tego wzorca

Błąd przy każdej iteracji ma niewielką wartość. W miarę postępu procesu uczenia się sieci (zwiększaniu się numeru iteracji) błąd częściej przyjmuje wartości bliskie zeru (efekt pożądany), ale także ma większą amplitudę wartości (efekt niepożadany). Pomimo przyjmowaniu często wartości bliskiej zeru, pojawiają się także wartości błędu wyższe niż na początku uczenia się sieci.

7. Wstępne testy zaimplementowanej sieci

Czułość (ang. sensitivity) sieci neuronowej interpretowana jest jako prawdopodobieństwo poprawnej klasyfikacji pod warunkiem, że przypadek jest pozytywny.
W naszym przypadku oznacza prawdopodobieństwo zakwalifikowania przez sieć przypadków chorych, oznaczonych wartością „0”
Czułość opisana jest wzorem:

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

TP - oznacza liczbę prawidłowych klasyfikacji przypadków pozytywnych
P – oznacza liczbę wszystkich przypadków pozytywnych, na które składają się przypadki pozytywne zakwalifikowane prawidłowo jako pozytywne (*TP*) oraz przypadki z wynikiem fałszywie negatywnym (*FN*), czyli zakwalifikowane jako negatywne pomimo swojej faktycznej wartości pozytywnej
Na podstawie testów przeprowadzonych dla zaimplementowanej sieci obliczona czułość wyniosła:

$\boldsymbol{TPR = 0.8600}$

Na podstawie obliczonej wartości czułości można stwierdzić, iż zaimplementowana sieć dość dobrze radzi sobie z rozpoznawaniem przypadków pozytywnych. Współczynnik czułości jest za zadowalającym wysokim poziomie.

Specyficzność (ang. specificity) określana jest jako prawdopodobieństwo, że klasyfikacja przy użyciu sieci neuronowej będzie poprawna, w przypadku, gdy konkretne dane będą wskazywały na wynik negatywny.
Biorąc pod uwagę nasz zbiór danych, sytuacja ta będzie miała miejsce, sieć zakwalifikuje prawidłowo przypadki negatywne oznaczone jako „1".
Specyficzność opisana jest wzorem:

$$SPC = \frac{TN}{N} = \frac{TN}{FP + TN}$$

TN - oznacza liczbę prawidłowych klasyfikacji przypadków negatywnych
N – oznacza liczbę wszystkich przypadków negatywnych, na które składają się przypadki negatywne fałszywe zakwalifikowane jako pozytywne (*FP*) oraz przypadki prawidłowo zakwalifikowane jako negatywne, czyli zgodnie z faktycznym ich stanem (*TN*).
Specyficzność opracowanej sieci wynosi:

$\boldsymbol{SPC = 1}$

Wartość specyficzności jest bardzo dobrym wynik wskazującym, że sieci udało się 100% przypadków zdrowych.