Wydział Informatyki

Metody sztucznej inteligencji

Laboratorium 01 IUz-22 Urbaniak

Sprawozdanie

Autor: Sergiusz Urbaniak
Grupa: IUz-22
Data: 10 grudnia 2009

1 Uczenie neuronu bramek logicznych

1.1 Bramki logicnzne

1.1.1 Bramka OR

Bramka OR posiada tablice prawdy pokazaną w tablicy 1.

Wejście	Wyjście
0 0	0
0 1	1
1 0	1
1 1	1

Tablica 1: Tablica prawdy bramki OR

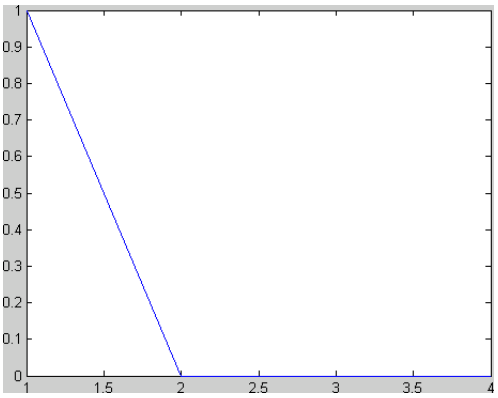
Wartości wejściowe są wpisane do zmiennej `we`. Dane wyjściowe według tablicy prawdy 1 są wpisane w zmienną `wy`. Następnie jest tworzona "sieć" jednego perceptronu i przeprowadzona symulacja działania sieci bez uczenia perceptronu. Kod źródłowy jest dostępny w listingu 1.

```
we = [0 0 1 1; 0 1 0 1];
wy = [0 1 1 1];

net = newp(minmax(we), 1);
y = sim(net, we);

figure(1)
plot(abs(y-wy));

figure(2)
plotpv(we, wy);
plotpc(net.iw{1,1}, net.b{1});
```

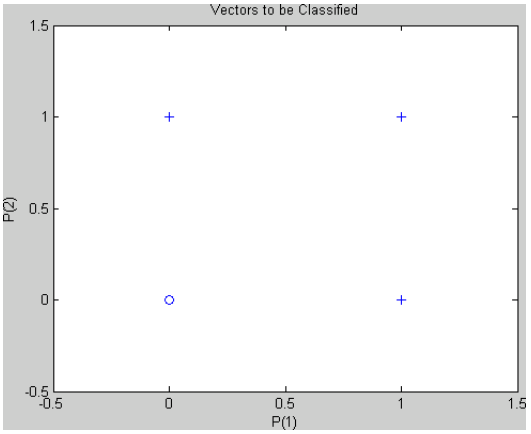
Listing 1: Kod nie uczonej bramki OR

Wykres błędu jak i danych generowany kodu 1 jest widoczny na rysnukach 1 i 2.

Kod zmieniony na uczenie perceptronu jest widoczny w listingu 2. Dodane zostały polecenia `init` i `train`.



Rysunek 1: Wykres błędu nie uczonej bramki OR



Rysunek 2: Linia podziału danych nie uczonej bramki OR

```
we = [0 0 1 1; 0 1 0 1];
wy = [0 1 1 1];

net = newp(minmax(we), 1);
net = init(net);
net.trainParam.epochs = 50;
net = train(net, we, wy);

y = sim(net, we);
figure(1)
plot(abs(y-wy));

figure(2)
plotpv(we, wy);
plotpc(net.iw{1,1}, net.b{1});
```

Listing 2: Kod uczonej bramki OR

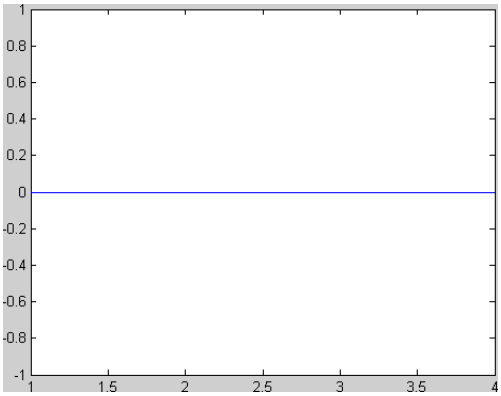
Wykres błędu jak i danych generowany kodu 2 jest widoczny na rysnukach 3 i 4. Można zaobserwowac że uczenie nie pozostawiło żadnego błędu. Perceptron poprawnie nauczył się bramki OR.

1.1.2 Bramka XOR

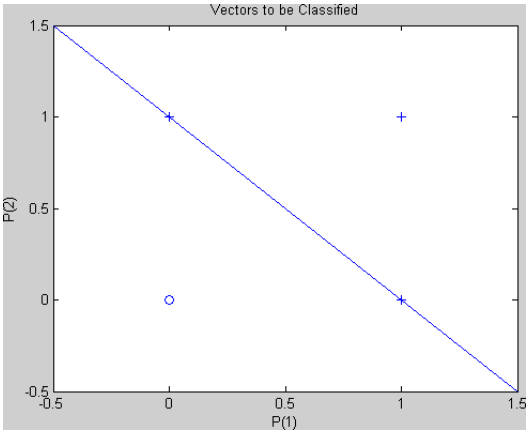
Bramka XOR posiada tablice prawdy pokazaną w tablicy 2.

Wejście	Wyjście
0 0	0
0 1	1
1 0	1
1 1	0

Tablica 2: Tablica prawdy bramki XOR



Rysunek 3: Wykres błędu uczonej bramki OR



Rysunek 4: Linia podziału danych uczonej bramki OR

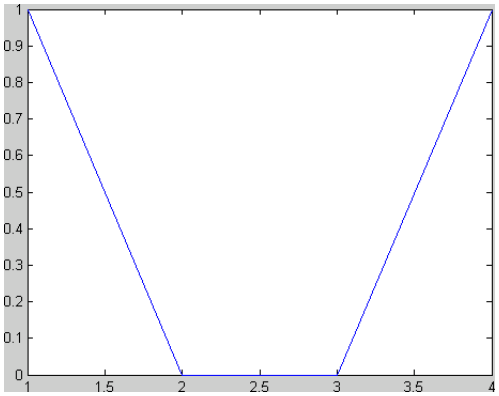
Kod perceptronu nie uczonego jest widoczny w listingu 3. Jak widac została tylko zmieniona zmienna `wy`.

```
we = [0 0 1 1; 0 1 0 1];
wy = [0 1 1 0];

net = newp(minmax(we), 1);
y = sim(net, we);

figure(1)
plot(abs(y-wy));

figure(2)
plotpv(we, wy);
plotpc(net.iw{1,1}, net.b{1});
```

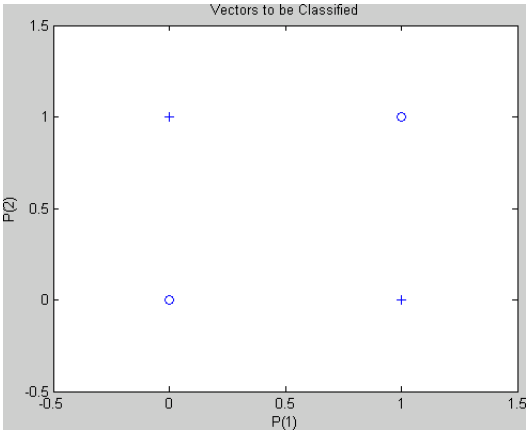
Listing 3: Kod nie uczonej bramki XOR

Wykres błędu jak i danych generowany kodu 3 jest widoczny na rysnukach 5 i 6.

Kod zmieniony na uczenie perceptronu jest widoczny w listingu 4. Znów dodane zostały polecenia `init` i `train`.



Rysunek 5: Wykres błędu nie uczonej bramki XOR



Rysunek 6: Linia podziału danych nie uczonej bramki XOR

```
we = [0 0 1 1; 0 1 0 1];
wy = [0 1 1 0];

net = newp(minmax(we), 1);
net = init(net);
net.trainParam.epochs = 50;
net = train(net, we, wy);

y = sim(net, we);
figure(1)
plot(abs(y-wy));

figure(2)
plotpv(we, wy);
plotpc(net.iw{1,1}, net.b{1});
```

Listing 4: Kod uczonej bramki XOR

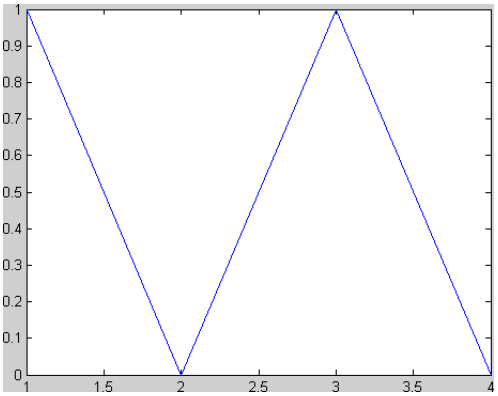
Wykres błędu jak i danych generowany z listingu 4 jest widoczny na rysnukach 7 i 8. Można zaobserwowac że uczenie nie udało się. Nadal istnieją błędy. Powodem jest fakt że podział danych bramki XOR nie jest lineowo separowalny.

1.2 Opracowanie wałsnej uczącą neuron z wzkorzystaniem reguły delta

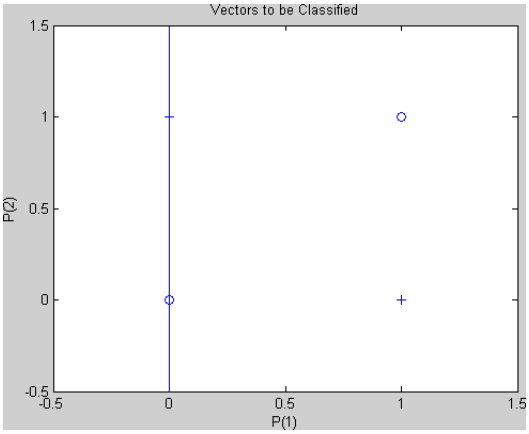
1.2.1 Opis funkcji `delta`

Zrealizowany algorytm operia się na neuronie według wzoru 1 i jest widoczny w listingu 5. Dane wejściowe $x_0, x_1, \cdots, x_n$ są wczytanę zmienną `in`.

Neuron posiada na wyjściu wartośc $y = 1$ albo $y = 0$ kiedy funkcja aktywacjyna φ jest wieksza lub równa albo mniejsza od wartości θ. Oczekiwane wartości wyjściowe są wczytane zmienną `out`. Uczone wartości wyjściowe są oddanę zmienną `learned`.



Rysunek 7: Wykres błędu uczonej bramki XOR



Rysunek 8: Linia podziału danych uczonej bramki XOR

$$
\begin{aligned}
y &= \begin{cases} 1 & \varphi \geq \theta \\ 0 & \varphi < \theta \end{cases} \qquad \text{gdzie} \\
\theta &= 0 \\
\varphi &= w_0 x_0 + w_1 x_1 + \cdots + w_n x_n + w_{n+1} b \\
b &= 1
\end{aligned}
\tag{1}
$$

Wartości początkowe wag $W_0 = (w_0, w_1, \cdots, w_n)$ są wybranę przypadkowo i leżą między $[0, 1]$ (linia 20 listingu 5). W każdej iteracji epoki uczenia są korygowane wartości wag według wzoru 2 (linia 55 listingu 5). Zmienna $0 > \eta \geq 1$ stanowi parametr uczący i jego wartośc została w tym przypadku wybrana jako $\eta = 0.2$.

δ jest różnicą między uczonymi wartościami t danej epoki i wartościami wejściowymi y (linia 49 listingu 5). Uczone dane t w bierzącej epoce są wyliczane za pomocą aktualnej sumy wag i zastosowania wzoru 1 (linie 36-46 listingu 5).

$$
\begin{aligned}
W_i &= W_{i-1} + \Delta W \qquad \text{gdzie} \\
\Delta W &= \eta \ \delta \ x \\
\eta &= 0.2 \\
\delta &= t - y \\
x &= (x_0, x_1, \cdots, x_n, b)
\end{aligned}
\tag{2}
$$

Iteracyjnie wagi W w każdej epoce są korygowane aż do momentu kiedy $t = y$ lub kiedy aktualna epoka jest równa maksymalnej epoce danej jako parametr `epochs` (linia 26/59 listingu 5).

Kod został napisany w środowisku Octave 3.0.5 pod Linux ze względu na fakt że to jest środowisko Open Source i za tym o wiele łatwej dostępne od pakietu Matlab.

1.2.2 Kod funkcji `delta`

```
function [learned,weights]=delta(in,out,epochs)
    % debug input parameters
    % in
    % out
    % epochs

    % specify learning parameter
    eta = 0.2

    % specify the bias. for our case, only 1 and 0 makes sense
    bias = 1

    % specify the threshold (theta)
    % in our case 0 makes since since we have boolean functions
    threshold = 0

    % initialize empty weight vector
    % we have as many weights as rows in "in" + 1
    % the additional entry is the weight for the bias
    weights = rand(1, size(in, 1)+1);

    % the learned variable has the same dimensions as the expected out
        variable
    learned = zeros(size(out,1), size(out,2));

    % loop through all epochs and try to learn
    for current_epoch=1:epochs
        % iterate through all input values in the current epoch
        for i=1:size(in,2)
            % transpose the current values
            cur_in_transposed = in(:,i)';

            % add the bias
            cur_in_and_bias = [cur_in_transposed, bias];

            % calculate the weighted inputs (including the bias)
            weighted_ins = weights .* cur_in_and_bias;

            % sum the weighted inputs
            % this sum is the current output value
            sum_weighted_ins = sum(weighted_ins);

            if (sum_weighted_ins >= threshold)
                learned(i) = 1;
            else
                learned(i) = 0;
            end

            % error in the current epoch
            err = out(i) - learned(i);

            % the delta of the weights
            delta_weight = eta * err * cur_in_and_bias;

            % updates the current weight values with the calculated delta
            weights = weights + delta_weight;
```

```
56          end
57
58          % if the learned vector equals the expected vector, then we are
                finished with learning
59          if (learned == out)
60              break;
61          end
62      end
63
64      if (learned == out)
65          printf("---\nlearning SUCCESS after %d epochs\n", current_epoch)
66      else
67          printf("---\nlearning FAILURE !\n");
68      end
69  end
```

Listing 5: `delta.m`: Funkcja ucząca neuron regułą delta