Sprawozdanie Scenariusz 4 Aleksandra Karaś

Uczenie sieci regułą Hebba

Cel ćwiczenia:

Celem ćwiczenia jest poznanie działania reguły Hebba na przykładzie rozpoznawania emotikon.

2. Zadania do wykonania:

a) Wygenerowanie danych uczących i testujących, zawierających 4 różne emotikony np. czarno-białe, wymiar 8x8 pikseli dla jednej emotikony.

b) Przygotowanie (implementacja lub wykorzystanie gotowych narzędzi) sieci oraz reguły Hebba w wersji z i bez współczynnika zapominania.

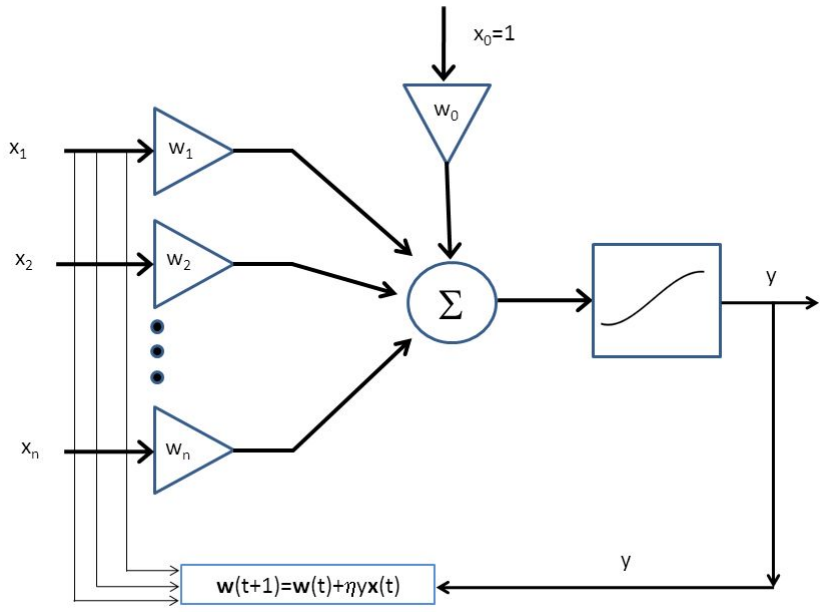
c) Uczenie sieci dla różnych współczynników uczenia i zapominania.

d) Testowanie siec

3. Opis budowy użytej sieci i algorytmu uczenia:

Model sieci neuronowej Hebba ma podobną strukturę w porównaniu do modelu Adaline, jednakże różnią się one samą regułą Hebba. Reguła ta występuje w dwóch wersjach: nauka z nauczycielem i bez nauczyciela. Przy implementacji można zastosować, lecz nie trzeba, współczynnik zapominania, który wspomaga uczenie sieci.

Model neuronu Hebba



Model sieci Hebba

Algorytm uczenia

Proces uczenia przebiega według następującego schematu:

1. Wybór współczynnika uczenia η z zakresu (0; 1>

2. Wylosowanie początkowych wartości wag z zakresu <0; 1)

3. Dla danego zbioru uczącego obliczamy odpowiedź sieci – dla każdego pojedynczego neuronu obliczana jest suma ilorazów sygnałów wejściowych oraz wag.

4. Modyfikacja wag odbywa się według następujących zależności:

a) Reguła Hebba (bez zapominania)

$$w_{i,j}^{k}(t+1) = \left(w_{i,j}(t)\right) + \eta * x_i * a_i$$

b) Reguła Hebba (z zapominaniem)

$$w_{i,j}^{k}(t+1) = \left(w_{i,j}(t)\right) * (1-\gamma) + \eta * x_i * a_i$$

η – współczynnik uczenia

x – zbiór danych wejściowych

γ – współczynnik zapominania

a – suma wyjściowa

5. Obliczenie łącznego błędu epoki

$$E = E + \frac{1}{2}\sum (d_i - y_i)^2$$

6. Uczenie odbywa się aż błąd nie będzie niższy niż określony próg

Przebieg programu:

Wykorzystanie funkcji newff(start, wyjscia_s, {'tansig'}, 'trainlm', 'learnh');, która tworzy sieć neuronową złożoną z jednej warstwy, a jej argumentami są:

– minmaxInput – wartości minimalne i maksymalne dla elementów wejściowych sieci dla funkcji tworzących sieć neuronową, które odpowiednio oznaczają wartość maksymalną - 1 oraz minimalną - 0 na wejściu,

– amount – liczba elementów wektora wyjściowego sieci,

– tansig – parametru określającego funkcję aktywacji neuronów (w tym przypadku tangens hiperboliczny),

– trainlm – funkcja szkolenia sieciowego, która aktualizuje wartości wagi i odchylenia, – learnh – jest wagą uczenia dla funkcji Hebba.

– net – zmienna, do której będzie przypisywana nowa tworzona sieć neutronowa,

– lp.dr – wartość współczynnika zapominania dla metody Hebba,

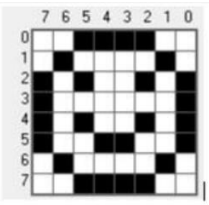
– lp.lr – wartość współczynnika uczenia dla metody Hebba,

– net.trainParam.x – określenie parametrów treningu sieci, gdzie x oznacza określenie:

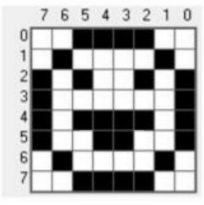
- net.trainParam.epochs – maksymalnej liczby epok,
- net.trainParam.goal – celu wydajności sieci,
- net.trainParam.lr – wartości współczynnika uczenia się sieci.

– train(net, inputData, heb) – uczenie (trening) sieci net z wykorzystaniem danych wejściowych i wartości określonych wcześniej wag dla reguły Hebba,

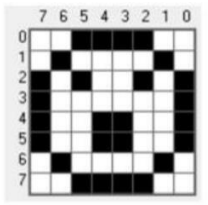
Użyte emotikony:



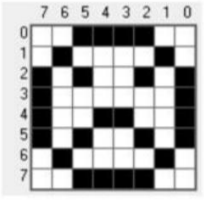
0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
1	0	0	0	0	0	0	1
1	0	1	0	0	1	0	1
1	0	0	1	1	0	0	1
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0



0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
1	0	0	0	0	0	0	1
1	0	1	1	1	1	0	1
1	0	0	1	1	0	0	1
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0



0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0



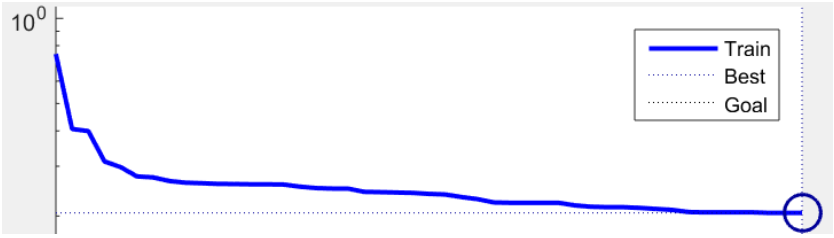
0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	1
1	0	1	0	0	1	0	1
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0

Wyniki:

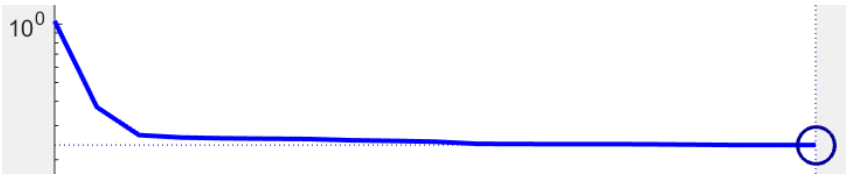
Współczynnik uczenia 0.5 współczynnik zapominania 0.5				
Smile	0.9529	1	0.9983	0.5723
Shock	0.9070	-1	-0.9996	0.3935
Happy	0.8267	0.9978	-0.6227	0.7774
Sad	0.1489	-1	0.9999	0.3696
Iteracje	42	34	16	46

Współczynnik uczenia 1 współczynnik zapominania 0				
Smile	1	-1	0.8501	1
Shock	1	-1	0.8770	1
Happy	1	-1	0.9775	1
Sad	1	-1	0.7532	1
Iteracje	15	20	17	18

Ostatnia próba Współczynnik uczenia 0.5 współczynnik zapominania 0.5



Ostatnia próba Współczynnik uczenia 1 współczynnik zapominania 0



Wnioski:

Skuteczność procesu uczenia zależy od współczynnika uczenia. Wraz z jego wzrostem proces uczenia jest efektywniejszy. Jest to wytłumaczalne z tego względu, że im ta wartość jest większa tym przyrost wag, które na samym początku są niewielkie jest szybszy, więc proces uczenia przebiega szybciej. Sieci wykorzystujące regułę Hebba są odporne na zaszumienie (do kilkudziesięciu bitów). Zbyt duże zaszumienie powoduje błędne odpowiedzi dawane przez sieć, niezależnie od zastosowanej reguły. Przy projektowaniu sieci neuronowej trzeba wybrać odpowiedni jej model. Oprócz samego modelu na jej efektywność ma wpływ sama jej struktura, np. zastosowana funkcja aktywacji. W samym modelu z regułą Hebba wpływ na efektywność uczenia sieci ma zastosowany rodzaj – z współczynnikiem zapominania lub bez. Reguła z współczynnikiem zapominania jest bardziej efektywna aniżeli reguła bez tego współczynnika. Jednakże trzeba uważać przy samym doborze wartości współczynnika zapominania, ponieważ wartości bliższe jedynce powodują, że sieć zapomina to, co przed chwilą nauczyła się.

Kod:

```
minmax=[0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;
    0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;
    0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;
    0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;
    0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;
    0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;
    0 1;0 1;0 1;0 1];

numerOfOutput=64;
net=newff(minmax,numerOfOutput,{'tansig'},'trainlm','learnh');

input=[ 0 0 0 0;
    0 0 0 0;
    1 1 1 1;
    1 1 1 1;
    1 1 1 1;
    1 1 1 1;
    0 0 0 0;
```

(..)

```
    1 1 1 1;
    0 0 0 0;
    0 0 0 0;
    ];

output =[ 1 0 0 0 % smile
          0 1 0 0 % shock
          0 0 1 0 % happy
          0 0 0 1 % sad
        ];

lp.dr = 0.5;
lp.lr =0.5;

Heb=learnh([],input,[],[],output,[],[],[],[],[],lp,[]);
heb=Heb';

net.trainParam.epochs = 100;
net.trainParam.goal = 0.1;
net=train(net,input,heb);
```

```
smile=[ 0 0 1 1 1 1 0 0;       happy=[ 0 0 1 1 1 1 0 0;
        0 1 0 0 0 0 1 0;               0 1 0 0 0 0 1 0;
        1 0 1 0 0 1 0 1;               1 0 1 0 0 1 0 1;
        1 0 0 0 0 0 0 1;               1 0 0 0 0 0 0 1;
        1 0 1 0 0 1 0 1;               1 0 1 1 1 1 0 1;
        1 0 0 1 1 0 0 1;               1 0 0 1 1 0 0 1;
        0 1 0 0 0 0 1 0;               0 1 0 0 0 0 1 0;
        0 0 1 1 1 1 0 0];              0 0 1 1 1 1 0 0];
shock=[ 0 0 1 1 1 1 0 0;         sad=[ 0 0 1 1 1 1 0 0;
        0 1 0 0 0 0 1 0;               0 1 0 0 0 0 1 0;
        1 0 1 0 0 1 0 1;               1 0 1 0 0 1 0 1;
        1 0 0 0 0 0 0 1;               1 0 0 0 0 0 0 1;
        1 0 0 1 1 0 0 1;               1 0 0 1 1 0 0 1;
        1 0 0 1 1 0 0 1;               1 0 1 0 0 1 0 1;
        0 1 0 0 0 0 1 0;               0 1 0 0 0 0 1 0;
        0 0 1 1 1 1 0 0];              0 0 1 1 1 1 0 0];

testSmile = sim(net, smile);disp('Smile ='),disp(testSmile(1));
testShock = sim(net, shock);disp('Shock ='),disp(testShock(1));
testHappy = sim(net, happy);disp('Happy ='),disp(testHappy(1));
testSad = sim(net, sad);    disp('Sad ='),  disp(testSad(1));
```