Piotr Wiśniewski

Sprawozdanie 1

Data oddania: 20.10.2017r.

Temat ćwiczenia:

Budowa i działanie perceptronu prostego, jednowarstwowego.

Cel ćwiczenia:

Poznanie budowy i działania perceptronu poprzez implementację i uczenie perceptronu realizacji wybranej funkcji logicznej.

Zadania do wykonania

1. Implementacja sztucznego neuronu

2. Wygenerowanie danych uczących dla funkcji logicznej AND

3. Uczenie perceptronu dla różnej liczby danych uczących, różnych współczynników uczenia

4. Testowanie perceptronu

Syntetyczny opis budowy oraz wykorzystanego algorytmu uczenia:

* 1. wymnożenie danych wejściowym przez odpowiadające im wagi- to właśnie one są zasadniczym elementem w procesie uczenia perceptronu. Początkowo są wybrane losowo, a następnie w kolejnych epokach, ich wartość jest optymalizowana przy pomocy wzorów:

w1+=n\*(d-y) \*x1

w2+=n\*(d-y) \*x2

b +=n\*(d-y)

gdzie w1, w2 to wagi, n jest niewielkim współczynnikiem uczenia(n>0), d-oczekiwaną odpowiedzią, y-odpowiedzią neuronu, b-progiem wzbudzenia perceptronu, a x1, x2 wartościami wejściowymi.

* 1. następnie w bloku sumującym iloczyny *xn* i wag *w1,w2,…wn* są sumowane ze sobą. Musimy także uwzględnić w sumie bias (*b*), czyli wartość odchylenia odpowiadająca za nieliniowe przekształcenie wejść w wyjście. Bias jest stałą losową na wejściu, tak jak w przypadku wag.
  2. w bloku aktywacji określa się funkcję aktywacji, która jest prostą służącą do dzielenia przestrzeni, klasyfikująca dane wejściowe do odpowiednich grup. Do naszego zadania wykorzystamy funkcję progową unipolarną:

gdzie θ jest zadaną wartością progową i przyjmujemy, że wynosi 0.

* 1. W ten sposób otrzymujemy wartość wyjściową y, która wynosi 0 lub 1, zależną od sumy ∑.

Zestawienie wyników:

|  |  |  |
| --- | --- | --- |
| Liczba epok | Współczynnik nauki | Ilość błędów |
| 5 | 0,05 | 5 |
| 10 | 0,05 | 10 |
| 15 | 0,05 | 11 |
| 20 | 0,05 | 16 |
| 25 | 0,05 | 2 |
| 30 | 0,05 | 11 |

|  |  |  |
| --- | --- | --- |
| Liczba epok | Współczynnik nauki | Ilość błędów |
| 5 | 0,2 | 3 |
| 10 | 0,2 | 0 |
| 15 | 0,2 | 2 |
| 20 | 0,2 | 0 |
| 25 | 0,2 | 4 |
| 30 | 0,2 | 4 |

|  |  |  |
| --- | --- | --- |
| Liczba epok | Współczynnik nauki | Ilość błędów |
| 5 | 0,3 | 2 |
| 10 | 0,3 | 3 |
| 15 | 0,3 | 2 |
| 20 | 0,3 | 1 |
| 25 | 0,3 | 1 |
| 30 | 0,3 | 3 |

|  |  |  |
| --- | --- | --- |
| Współczynnik nauki | Liczba epok | Ilość błędów |
| 0,05 | 10 | 10 |
| 0,10 | 10 | 7 |
| 0,15 | 10 | 6 |
| 0,20 | 10 | 2 |
| 0,25 | 10 | 0 |
| 0,30 | 10 | 1 |

Jak widać na powyższych tabelach, przy losowym ustawieniu wag liczba błędów zależna jest od wielkości współczynnika nauki. Liczba epok nie ma na to większego wpływu jednak przy małym współczynniku nauki większa ilość epok zapewnia większe szanse na powodzenie nauki.

Powyższy wykres przedstawia jak współczynnik nauki wpływał na ilość błędów przy kolejnych testach

Wnioski:

W zadaniu wykorzystałem algorytm szukania wag – reguła perceptronu, czyli szczególny przypadek reguły Widrowa-Hoffa.

Współczynnik uczenia się perceptronu ma decydujący wpływ na szybkość jego uczenia się - im wyższy tym szybciej on przebiega. Przy eta= 0.5 lub eta = 0.4 sztuczny neuron potrzebował jedynie 3 iteracji, by wyliczane dane wyjściowe były prawidłowe.

Podczas każdego wykonanego testowania, nie wykryto błędu w obliczeniach perceptronu. Można podejrzewać, że stało się tak, ponieważ funkcje logiczne są jednymi z najprostszych operacji i prawdopodobieństwo otrzymania dobrego wyniku jest równe 50%.

Wszystkie 3 czynniki wpływają na skuteczność uczenia się perceptronu: wagi początkowe, współczynnik uczenia się a także ilość powtórzeń danych uczących.

*pętle uczące p z wyświetlaniem wyników po każdej epoce*

Listing kodu:

**import** java.util.Random;  
  
**public class** Perceptron {  
  
 *//losowe wagi początkowe* Random **rand** =**new** Random();  
 **public double weight0**=**rand**.nextDouble();  
 **public double weight1**=**rand**.nextDouble();  
 **public double weight2**=**rand**.nextDouble();

*Wagi losowane są z przedziału od 0 do 1*

*Tworzymy funkcję progową unipolarną zgodnie z algorytmem podanym na początku sprawozdania*

**public int** out(**double** x)  
{  
 **if**(x<0){  
 **return** 0;  
 }  
 **else**{  
 **return** 1;  
 }  
}

*Jest to funkcja sumująca iloczyny wartości xpodanych na wejściu oraz wag wejść. Musimy także uwzględnić w sumie bias (b), czyli wartość odchylenia odpowiadająca za nieliniowe przekształcenie wejść w wyjście. Bias jest stałą losową na wejściu*

**public int** sum ( **int** x0, **int** x1, **int** x2 ) {  
 **double** y\_p = x0 \* **weight0** + x1 \* **weight1** + x2 \* **weight2**;  
 **return** out( y\_p );  
}

*Jako wyjście podawana jest wartość funkcji aktywacji dla obliczonej sumy*

*Tutaj następuje modyfikacja wag perceptronu w przypadku błędu. Jest to funkcja ucząca*

**public void** learn(**int** x0, **int** x1, **int** x2, **double** y, **double** learningRate)  
 {  
 **double** y\_p = sum(x0,x1,x2);  
  
 **weight0** += ( y - y\_p ) \* learningRate \* x0;  
 **weight1** += ( y - y\_p ) \* learningRate \* x1;  
 **weight2** += ( y - y\_p ) \* learningRate \* x2;  
 }  
}

**public class** Main {  
 **public static void** main(String[] args) {  
  
 **int** era = 10;  
 **int** bias = 1;  
 **double** learningRate = 0.3;  
  
 Perceptron p = **new** Perceptron();

*Tablice wejściowe*:  
 **int**[] x1 = {0, 0, 1, 1};  
 **int**[] x2 = {0, 1, 0, 1};

*Tablica wynikowa* – taki wynik powinien podawać nauczony neuron  
 **int**[] y = {0, 1, 1, 1}; *//Bramka OR* System.***out***.println(**"\nWagi wylosowane:"**);  
 System.***out***.println(**"W1 = "** + p.**weight0**);  
 System.***out***.println(**"W2 = "** + p.**weight1**);  
 System.***out***.println(**"W3 = "** + p.**weight2**);  
  
 System.***out***.println(**"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"**);

*Pętle uczące z wyświetlaniem wyników po każdej epoce* **for**(**int** i =0;i<era;i++){  
 **for**(**int** j=0;j<4;j++)  
 {  
 p.learn(bias, x1[j], x2[j], y[j], learningRate);  
 }  
 System.***out***.println(**"Numer epoki: "**+ (i+1) + **"\n"**);

**for** (**int** k = 0; k < 4; k++) {  
 System.***out***.println(**"["** + x1[k] + **" "** + x2[k] + **"] -> "** + y[k] + **" wynik = "** + p.sum(bias, x1[k],x2[k]));  
 }  
  
 System.***out***.println(**"\nWagi:"**);  
 System.***out***.println(**"W1 = "** + p.**weight0**);  
 System.***out***.println(**"W2 = "** + p.**weight1**);  
 System.***out***.println(**"W3 = "** + p.**weight2**);  
 System.***out***.println(**"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"**);  
 }  
  
 }  
}