**Podstawy Sztucznej Inteligencji**

Scenariusz 4: Uczenie​​ sieci​​ regułą​​ Hebba

Wszystkie projekty znajdują się w jednym programie na repozytorium

**Wykonał:**

**Bartłomiej Leja**

**Wydział Inżynierii Metali i Informatyki Przemysłowej**

**Informatyka Stosowana**

1. **Cel ćwiczenia**

Celem​​ ćwiczenia​​ jest​​ poznanie​​ działania​​ reguły​​ Hebba​​ dla​​ sieci​​ jednowarstwowej​​ na przykładzie​​ grupowania​​ liter​​ alfabetu.

1. **Reguła Hebba**

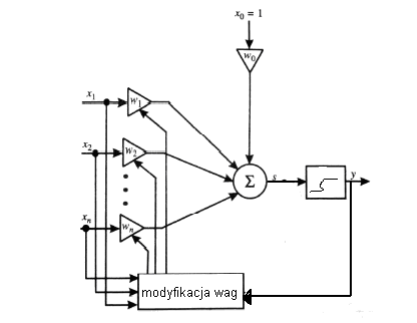
Reguła Hebba jest to jedna z najpopularniejszych metod samouczenia sieci neuronowych. Polega ona  na tym, że sieci pokazuje się kolejne przykłady sygnałów wejściowych, nie podając żadnych informacji o tym, co z tymi sygnałami należy zrobić**.**Sieć obserwuje otoczenie i odbiera różne sygnały, nikt nie określa jednak, jakie znaczenie mają pokazujące się obiekty i jakie są pomiędzy nimi zależności. Sieć na podstawie obserwacji występujących sygnałów stopniowo sama odkrywa, jakie jest ich znaczenie i również sama ustala zachodzące między sygnałami zależności.

Po podaniu do sieci neuronowej każdego kolejnego zestawu sygnałów wejściowych tworzy się w tej sieci pewien rozkład sygnałów wyjściowych - niektóre neurony sieci są pobudzone bardzo silnie, inne słabiej, a jeszcze inne mają sygnały wyjściowe wręcz ujemne. Interpretacja tych zachowań może być taka, że niektóre neurony „rozpoznają” podawane sygnały jako „własne” (czyli takie, które są skłonne akceptować), inne traktują je „obojętnie”, zaś jeszcze u innych neuronów wzbudzają one wręcz „awersję”. Po ustaleniu się sygnałów wyjściowych wszystkich neuronów w całej sieci - wszystkie wagi wszystkich neuronów są zmieniane, przy czym wielkość odpowiedniej zmiany wyznaczana jest na podstawie iloczynu sygnału wejściowego, wchodzącego na dane wejście (to którego wagę zmieniamy) i sygnału wyjściowego produkowanego przez neuron, w którym modyfikujemy wagi. Łatwo zauważyć, że jest to właśnie realizacja postulatu Hebba - w efekcie opisanego wyżej algorytmu połączenia między źródłami silnych sygnałów i neuronami które na nie silnie reagują są wzmacniane.

Bardzo istotną kwestią jest wybór początkowych wartości wag neuronów sieci przeznaczonej do samouczenia. Wartości te mają bardzo silny wpływ na ostateczne zachowanie sieci, ponieważ proces uczenia jedynie pogłębia i doskonali pewne tendencje istniejące w sieci od samego początku, przeto od jakości tych początkowych, „wrodzonych” właściwości sieci silnie zależy, do czego sieć dojdzie na końcu procesu uczenia. Nie wiedząc z góry, jakiego zadania sieć powinna się uczyć, trudno wprowadzać jakikolwiek zdeterminowany mechanizm nadawania początkowych wartości wag, jednak pozostawienie wszystkiego wyłącznie mechanizmom losowym może powodować, że sieć (zwłaszcza mała) może nie zdołać wystarczająco zróżnicować swego działania w początkowym okresie procesu uczenia i wszelkie późniejsze wysiłki, by znaleźć w strukturze sieci reprezentację dla wszystkich występujących w wejściowych sygnałach klas, mogą okazać się daremne.

1. **Model neuronu Hebba**

Model ten ma identyczną strukturę jak w przypadku modelu typu Adaline oraz neuronu sigmoidalnego, ale charakteryzuje się specyficzną metodą uczenia, znaną pod nazwą reguły Hebba. Reguła ta występuje z nauczycielem jak i bez nauczyciela. Hebb zauważył podczas badań działania komórek nerwowych, iż połączenie pomiędzy dwiema komórkami jest wzmacniane, jeśli w tym samym czasie obie komórki są aktywne.



Rys. 1

Zaproponował on algorytm, zgodnie z którym modyfikację wag przeprowadza się następująco:



Oznaczenia:

• i-numer wagi neuronu,

• t-numer iteracji w epoce,

• y-sygnał wyjściowy neuronu,

• x-wartość wejściowa neuronu,

• η - współczynnik uczenia (0,1).

1. **Zestaw danych uczących**

Dane uczące zostały zawarte w pliku HebbLetterLearningData.txt, który zawiera 20 liter polskiego alfabetu w postaci 35 elementowych wektorów utworzonych na bazie matryc 7x5. Dla przykładu litera A to:

**-1 1 1 1 -1  
1 -1 -1 -1 1  
1 -1 -1 -1 1  
1 1 1 1 1** 🡪 **-1 1 1 1-1 1-1-1-1 1 1-1-1-1 1 1 1 1 1 1 1-1-1-1 1 1-1-1-1 1 1-1-1-1 1  
1 -1 -1 -1 1   
1 -1 -1 -1 1  
1 -1 -1 -1 1**

1. **Uzyskane wyniki**

Analizę reguły Hebba zaczęto od różnych współczynników uczenia (1, 0.5 oraz 0,1) dla współczynnika zapomnienia 0 , 100 powtórzeń uczenia oraz binarnej funkcji aktywacji sgn. Wyniki dla wszystkich badanych przypadków zostały przedstawione w tabeli 1 oraz zobrazowane na wykresie 1 dla litery A.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Litera A | Litera B | Litera C | Litera D | Litera I | Wszystkie wartości -1 | Wszystkie wartości +1 |
| Sygnał wyjściowy wsp. Ucz. 1 | 33002,49 | 34203,82 | 28197,29 | 33801,87 | -7404,78 | 2185,50 | -2185,50 |
| Sygnał wyjściowy wsp. Ucz. 0,5 | 16501,82 | 17102,22 | 14095,71 | 16900,76 | -3706,48 | 1085,53 | -1085,53 |
| Sygnał wyjściowy wsp. Ucz. 0,1 | 3300,874 | 3422,354 | 2818,273 | 3381,185 | -746,796 | 203,250 | -203,25 |

Tab.1

Rys. 1

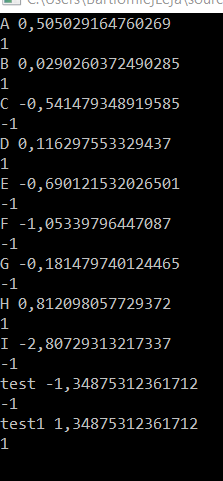
Następnie sporządzono tabele różnych współczynników zapomnienia (1, 0,9 oraz 0.1) dla współczynnika uczenia 0.1, 100 powtórzeń uczenia oraz binarnej funkcji aktywacji sgn.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Litera A | Litera B | Litera C | Litera D | Litera I | Wszystkie wartości -1 | Wszystkie wartości +1 |
| Sygnał wyjściowy wsp. zapomnienia 0,9 | 0,5 | 0,02 | -0,54 | 0,11 | -2,8 | -1,38 | 1,38 |
| Sygnał wyjściowy wsp. zapomnienia 0,5 | 1,19 | 1,88 | 2,15 | 1,37 | 3,17 | 1,53 | -1,53 |
| Sygnał wyjściowy wsp. zapomnienia 0,1 | 15,69 | 15,67 | 12,79 | 15,43 | -4,78 | 0,4 | -0,4 |

Tab. 2

Wykres. 2

Na screnshotcie pokazany został przykładowy wynik dla współczynnika zapomnienia 0.9 współczynnika uczenia 0.1 oraz 100 powtórzeń uczenia.



Rys. 2

1. **Wnioski**

Najważniejszym wnioskiem tego scenariusza jest to iż reguła Hebba czyli nauczanie neuronów bez nauczyciela działa. Wniosek jest o tyle ważny iż do tej pory uczyliśmy neurony w naturalny dla komputera, wręcz maszynowy sposób dając im przykłady do czego mają dążyć. Wykonane ćwiczenie pokazuję, że neurony potrafią same wzmacniać swój sygnał i dzięki temu grupować podobne litery. Jedno z najważniejszych zagadnień w regule Hebba to dobranie odpowiednich parametrów dzięki którym neuron może poprawnie nauczyć się zadanego przykładu. Jak widać na rys. 2 udało się te parametry odpowiednio dobrać. Dzięki wykonanym eksperymentom stwierdzoną iż optymalnym zakresem początkowych wag jest zakres od 0 do 1. Dodatkowo zauważono, że metoda ze współczynnikiem zapomnienia pokazuje dokładniejszy podział liter. Czym wyższy współczynnik zapomnienia tym lepszy grupowanie liter. Współczynnik zapomnienia zapobiegał również dużemu narastaniu wag. Dzięki sygnałowi wyjściowemu możemy zaobserwować więcej grup niż wynika to tylko z binarnej funkcji aktywacji. Dobrym przykładem jest litera I mająca sygnał wyjściowy -2.8 oraz litera G mająca sygnał wyjściowy -0.18. obie te litery zostały zaklasyfikowane do jednej grupy. Biorąc pod uwagę różnice pomiędzy ich wyglądem oraz sygnałem wyjściowym powinny być w innych grupach. Najlepszym współczynnikiem uczenia była wartość 0.1 co potwierdza tezę większości dostępnych materiałów. Podsumowując reguła Hebba nie jest idealna ale jest to wielki krok w kierunku samo uczeniu neuronów.

1. **Literatura**

<http://pracownik.kul.pl/files/31717/public/Model_neuronu_Hebba.pdf>

<http://www.poltynk.pl/marcin/reguly.html>

Wykłady

1. **Listing kodu**

using System;

namespace PerceptronLerning.HebbRule

{

class HebbRule

{

private double[] \_weight { get; set; }

private double a;

private double s;

private double c = 0.1;

private double [] cap;

private double gamma = 0.9;

void adjustWeight(double [] input)

{

for(int i=0;i<input.Length;i++)

{

s += \_weight[i]\* input[i];

}

a = (s > 0) ? 1 : -1;

cap = new double[\_weight.Length];

for(int i=0;i< \_weight.Length;i++)

{

cap[i] = (c \* a) \* input[i];

}

for (int i = 0; i < \_weight.Length; i++)

{

\_weight[i] = \_weight[i]\*(1-gamma) + cap[i];

}

s = 0;

a = 0;

}

public void adjustWeights(double [][] input, int timeOfLerning)

{

drawWeight();

string textForOutput = String.Empty;

for (int i =0; i<timeOfLerning;i++)

{

for(int j =0;j<20;j++)

{

adjustWeight(input[j]);

}

}

}

public void test(double [] testInput)

{

double testResult=0;

for (int i = 0; i < testInput.Length; i++)

{

testResult += (testInput[i] \* \_weight[i]);

}

Console.WriteLine($"{testResult}");

testResult = (testResult > 0) ? 1 : -1;

Console.WriteLine($"{testResult}");

}

private void drawWeight()

{

\_weight = new double[35];

var randomNumber = new Random();

for (int i = 0; i < 35; i++)

\_weight[i] = randomNumber.NextDouble() ;

}

}

}

using System;

namespace PerceptronLerning.HebbRule

{

class HebbRuleHandler

{

public void HebbRuleShower()

{

var inputTest = new HebbLetterDataTextFileReader();

var array = inputTest.ConverTextForTestDataArray(@"C:\Users\BartlomiejLeja\source\repos\PerceptronLerning\PerceptronLerning\HebbLetterLearningData.txt");

var hebb = new HebbRule();

hebb.adjustWeights(array, 100);

Console.Write("A ");

hebb.test(new double[] { -1, 1, 1, 1, -1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1 });

Console.Write("B ");

hebb.test(new double[] { 1, 1, 1, 1, -1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, 1, 1, 1, -1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, 1, 1, 1, -1 });

Console.Write("C ");

hebb.test(new double[] { -1, 1, 1, 1, -1, 1, -1, -1, -1, 1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1, -1, -1, -1, 1, -1, 1, 1, 1, -1 });

Console.Write("D ");

hebb.test(new double[] { 1, 1, 1, 1, -1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, 1, 1, 1, -1 });

Console.Write("E ");

hebb.test(new double[] { 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1, 1, 1, 1, -1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1, 1, 1, 1, 1 });

Console.Write("F ");

hebb.test(new double[] { 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1, 1, 1, 1, -1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1 });

Console.Write("G ");

hebb.test(new double[] { -1, 1, 1, 1, -1, 1, -1, -1, -1, 1, 1, -1, -1, -1, -1, 1, -1, 1, 1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, -1, 1, 1, 1, -1 });

Console.Write("H ");

hebb.test(new double[] { 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1 });

Console.Write("I ");

hebb.test(new double[] { -1, 1, 1, 1, -1, -1, -1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1, -1, -1, -1, 1, 1, 1, -1 });

Console.Write("test ");

hebb.test(new double[] { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 });

Console.Write("test1 ");

hebb.test(new double[] { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 });

}

}

}