

**Podstawy Sztucznej Inteligencji**

Scenariusz 4: Uczenie sieci regułą Hebba

Wykonała:

**Magdalena Migas**

**Wydział Inżynierii Metali i Informatyki Przemysłowej**

**Informatyka Stosowana**

1. **Cel ćwiczenia**

Celem ćwiczenia było poznanie działania reguły Hebba dla sieci jednowarstwowej na

przykładzie grupowania liter alfabetu. W omawianym zagadnieniu przygotowano jednowarstwową sieć neuronową, która wyznaczyła swoje współczynniki wagowe z wykorzystaniem reguły Hebba i następnie grupowała podane jej litery polskiego alfabetu.

1. **Reguła Hebba**

Uczenia z wykorzystaniem reguły Hebba jest przykładem uczenia nienadzorowanego. Sieci neuronowe nie mają przedstawianych wzorców, których mają się nauczyć, lecz same organizują przedstawiane im wektory danych.

Reguła ta opiera się na stwierdzeniu Hebba z roku 1949 o następującej treści:

„Jeżeli akson komórki A bierze systematycznie udział w pobudzaniu komórki B powodującym jej aktywację, to wywołuje to zmianę metaboliczną w jednej lub w obu komórkach, prowadzącą do wzrostu skuteczności pobudzania B przez A.”

Można wyróżnić kilka modyfikacji reguły Hebba. W niniejszym ćwiczeniu wykorzystano dwie najpopularniejsze z nich:

1. Prosta reguła Hebba

W tym przypadku korekta poszczególnych wag sieci odbywa się według następujących wzorów:

1. Reguła Hebba wykorzystująca współczynnik zapominania

Modyfikacja ta wynika z faktu, że wagi mogą przybierać dowolnie duże wartości gdyż w każdym cyklu uczącym następuje proces sumowania. Dlatego też dąży się do stabilizacji procesu uczenia. Jedną z wykorzystywanych metod jest wprowadzenie współczynnika zapominania, który przyjmuje wartości z zakresu od 0 do 1. Wzór na zmianę wag przyjmuje wówczas postać:

Wady samouczenia metodą Hebba:

* Stosunkowo niska efektywność uczenia;
* Przemnożony wpływ początkowych wartości wag;
* Możliwość pomijania niektórych klas w nauczonej sieci;
* Powstawanie reduntantnych nadprezentacji klas.

1. **Zestaw danych uczących**

Przykładowe dane uczące znajdują się w pliku zbior\_uczacy1.txt. W pliku tym zawarto 20 liter polskiego alfabetu, które z matryc 35-elementowych rozwinięto na odpowiadające im wektory, dla przykładu litera A:

**0 1 1 1 0  
1 0 0 0 1  
1 0 0 0 1  
1 1 1 1 1** 🡪 **0 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1  
1 0 0 0 1   
1 0 0 0 1  
1 0 0 0 1**

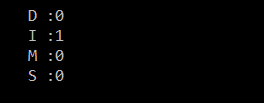
1. **Uzyskane wyniki**

Napisany program poddano testom z wykorzystaniem różnych współczynników uczenia oraz zapominania a także różnej ilości iteracji zadanej do wyuczenia sieci. Do zbioru testującego wybrano 4 litery: D, I, M, S. W każdym przypadku sprawdzano jak stworzona sieć grupuje podane litery. Dla każdego przypadku przeprowadzono po 20 testów grupowania.

**Wariant I:**

* Zestaw uczący zawiera 20 liter polskiego alfabetu.
* Zastosowano algorytm z binarną, bipolarną funkcją aktywacji w postaci sgn(u);
* Testy przeprowadzono dla 4 różnych współczynników uczenia oraz różnych współczynników zapominania;
* W testach sprawdzano jak sieć grupuje litery: **D**, **I**, **M, S**.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Numer wariantu** | **Współczynnik uczenia c** | **Współczynnik zapominania γ** | **Liczba danych uczących** | **Liczba potrzebnych iteracji do wyuczenia** | **Liczba błędów przy testowaniu** |
| **1** | 0.1 | 0.1 | 20 | 50 | 17/20 |
| **2** | 0.2 | 0.2 | 20 | 50 | 14/20 |
| **3** | 0.5 | 0.3 | 20 | 50 | 11/20 |
| **4** | 0.7 | 0.5 | 20 | 50 | 12/20 |



Rysunek 1: Test dla współczynnika uczenia równego 0.7 i współczynnika zapominania 0.5

Wykres 1: Ilość błędów podczas grupowania w zależności od wartości współczynnika uczenia

Wykres 2: Ilość błędów podczas grupowania w zależności od wartości współczynnika zapominania

**Wariant II:**

* Zestaw uczący zawiera 20 liter polskiego alfabetu.
* Zastosowano algorytm z binarną, bipolarną funkcją aktywacji w postaci sgn(u);
* Testy przeprowadzono dla 4 różnych współczynników uczenia oraz różnych współczynników zapominania;
* W testach sprawdzano jak sieć grupuje litery: **D**, **I**, **M, S**.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Numer wariantu** | **Współczynnik uczenia c** | **Współczynnik zapominania γ** | **Liczba danych uczących** | **Liczba potrzebnych iteracji do wyuczenia** | **Liczba błędów przy testowaniu** |
| **1** | 0.1 | 0.1 | 20 | 250 | 11/20 |
| **2** | 0.2 | 0.2 | 20 | 250 | 9/20 |
| **3** | 0.5 | 0.3 | 20 | 250 | 8/20 |
| **4** | 0.7 | 0.5 | 20 | 250 | 10/20 |

Wykres 3: Ilość błędów podczas grupowania w zależności od wartości współczynnika uczenia

Wykres 4: Ilość błędów podczas grupowania w zależności od wartości współczynnika zapominania

**Wariant III:**

* Zestaw uczący zawiera 20 liter polskiego alfabetu.
* Zastosowano algorytm z binarną, bipolarną funkcją aktywacji w postaci sgn(u);
* Testy przeprowadzono dla 4 różnych współczynników uczenia ale bez uwzględnienia współczynnika zapominania;
* W testach sprawdzano jak sieć grupuje litery: **D**, **I**, **M, S**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Numer wariantu** | **Współczynnik uczenia c** | **Liczba danych uczących** | **Liczba potrzebnych iteracji do wyuczenia** | **Liczba błędów przy testowaniu** |
| **1** | 0.1 | 20 | 250 | 13/20 |
| **2** | 0.2 | 20 | 250 | 11/20 |
| **3** | 0.5 | 20 | 250 | 11/20 |
| **4** | 0.7 | 20 | 250 | 12/20 |

Wykres 5: Zależność ilości błędów testowania od ilości iteracji

1. **Wnioski**

* Wykonane ćwiczenie polegało na implementacji jednowarstwowej sieci neuronowej, której zadaniem było grupowanie liter polskiego alfabetu przy użyciu reguły Hebba.
* W trakcie testów zaimplementowanej sieci przeprowadzono trzy warianty, różniące się ilością iteracji w procesie uczenia, współczynnikiem uczenia i współczynnikiem zapominania. W pierwszym wariancie liczba iteracji wynosiła 50, w drugim wariancie 250, a w trzecim 500. Ponadto w poszczególnych wariantach wprowadzono rozróżnienie ze względu na przyjęty współczynnik uczenia oraz zapominania. W efekcie każdy wariant obejmuje po 4 przypadki testowe.
* Wariant numer jeden to proces uczenia trwający 50 iteracji. Współczynnik uczenia zmienia się od 0.1 do 0.7 a współczynnik zapominania od 0.1 do 0.5. Dla każdego przypadku przeprowadzono po 20 testów i policzono błędy. Za błąd uznano sytuację kiedy sieć nie grupuje podanych znaków na dwie odrębne grupy ale przyporządkowuje je wszystkie do jednej grupy. Na podstawie uzyskanych wyników sporządzono wykresy oznaczone numerami 1 i 2. Wynika z nich, że początkowo wzrost współczynników uczenia i zapominania powoduje zmniejszenie błędów przy grupowaniu liter przez sieć jednakże po przekroczeniu określonej wartości tych współczynników ilość błędów znowu zaczyna rosnąć. Na tej podstawie możemy stwierdzić, że optymalne ich wartości dla 50 iteracji to : c=0.5 i γ=0.3. Należy jednak tutaj zaznaczyć, że 50 iteracji to za mało do wyuczenia sieci gdyż uzyskano bardzo dużo błędów podczas testowania.
* W wariancie numer 2 przyjęto, że ilość iteracji w procesie uczenia wynosi 250. W tym przypadku zaobserwowano zależności analogiczne do wariantu numer 1, a mianowicie zmniejszenie liczby błędów grupowania liter przy rosnących współczynnikach uczenia i zapominania aż do osiągnięcia ekstremum w określonym punkcie i następnie ponowny wzrost ilości popełnianych błędów przez sieć. Opisana zależność przedstawiona jest na wykresach nr 3 i nr 4. W porównaniu do wariantu numer 1 średnio uzyskano mniejszą ilość błędów podczas testowania co wynika bezpośrednio ze zwiększonej liczby iteracji podczas uczenia sieci.
* Wariant numer 3 opisuje przypadek, w którym nie uwzględniono współczynnika zapominania a liczba iteracji wynosi 250. Tutaj należy zauważyć pogorszenie wyników w stosunku do analogicznego wariantu z uwzględnionym współczynnikiem zapominania. Jest to spowodowane faktem iż współczynnik zapominania służy do stabilizacji rozwiązania, bez niego wagi sieci mogą rosnąć w nieskończoność.
* Wykres numer 5 pokazuje zależność ilości błędów testowania od ilości iteracji w procesie uczenia. Jak się można było spodziewać rosnąca ilość iteracji w procesie uczenia wpływa korzystnie na wyniki uzyskiwane podczas testów jednakże taka zależność utrzymuje się jedynie do pewnej określonej liczby iteracji. W dalszym etapie ilość błędów systematycznie co rośnie co może świadczyć o tym, ze mamy do czynienia ze zjawiskiem przeuczenia.

1. **Listing kodu programu**

#include<iostream>

#include<cstdlib>

#include<fstream>

#include<list>

#include<time.h>

using namespace std;

class HebbRule {

public:

int size;

int inputSize;

int iterations;

double \*\*inputs;

double \*weights;

double eta=0.1;

double calculateValue(double sum);

void InitializeWeights();

void learnNeuron();

void testNetwork(int \*testTable, char znak, int index);

int letterID;

char letter;

double output;

HebbRule();

};

HebbRule::HebbRule(){

fstream plik;

plik.open("zbior\_uczacy1.txt");

if (plik.good())

{

inputSize = 35;

size = 20;

inputs = new double\*[size];

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

inputs[i] = new double[inputSize];

double \*result = new double[inputSize];

weights = new double[inputSize];

for (int k = 0; k < size; k++)

{

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

{

plik >> inputs[k][i];

}

}

}

else

{

cout << "blad otwarcia pliku!";

}

}

double HebbRule::calculateValue(double sum) {

if (sum >= 0)

return 1;

else

return 0;

}

void HebbRule::InitializeWeights() {

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

{

weights[i] = -1 + (double)rand() / RAND\_MAX\* (2);

}

}

void HebbRule::testNetwork(int \*testTable, char znak, int index)

{

double sum = 0;

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

sum += (testTable[i] \* weights[i]);

cout << " " << znak << " :" << calculateValue(sum) << endl;

}

void HebbRule::learnNeuron() {

InitializeWeights();

double y = 0;

double s = 0;

iterations = 500;

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

{

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

{

for (int k = 0; k < inputSize; k++)

{

y += (inputs[j][k] \* weights[k]);

}

s = calculateValue(y);

for (int k = 0; k < inputSize; k++)

{

weights[k] = weights[k]\*1+eta\*s\*inputs[j][k];

}

}

}

}

int main()

{

srand(time(NULL));

HebbRule A,B,C,D,E;

A.learnNeuron();

int tab3[35] = { 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0 }; //D

int tab8[35] = { 0, 1, 1, 1, 0, 0, 0, 1, 0, 0 ,0, 0, 1, 0 ,0 ,0, 0 ,1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0 }; //I

int tab12[35] = { 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1 }; //M

int tab18[35] = { 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0 }; //S

A.testNetwork(tab3, 'D', 2);

A.testNetwork(tab8, 'I', 8);

A.testNetwork(tab12, 'M', 12);

A.testNetwork(tab18, 'S', 18);

getchar();

}

1. **Literatura**

* <https://www.ii.uni.wroc.pl/~aba/teach/NN/w6pca.pdf>
* <http://pracownik.kul.pl/files/31717/public/Model_neuronu_Hebba.pdf>
* <http://home.agh.edu.pl/~asior/stud/doc/Kohonen_12.pdf>
* <http://sknbo.ue.poznan.pl/neuro/ssn/pliki/samoucz/samoucz1.html>
* <http://aragorn.pb.bialystok.pl/~gkret/SSN/Ssn_w2.PDF>
* <http://www.neurosoft.edu.pl/media/pdf/jbartman/sztuczna_inteligencja/NTI4.pdf>