Sprawozdanie

Podstawy sztucznej Inteligencji – Scenariusz 1

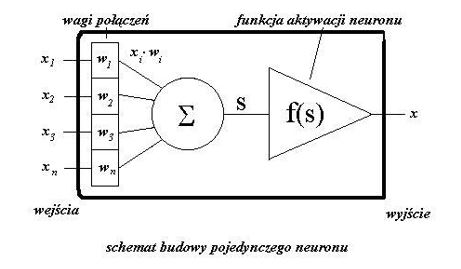
Mikołaj Waśniewski WIMiIP

Projekt napisany w Java

1) Opis budowy oraz algorytmu uczenia

Budowa:

Perceptron składa się z wag poszczególnych danych wejściowych(losowo inicjowanych z przedziału -1,2),funkcji aktywacyjnej(bipolarna) oraz prędkości nauczania i wagi biasu. W momencie podania danych wejściowych perceptron mnoży każdą z danych przez jej odpowiednią wagę(w tym bias) po czym dodaje do siebie wyniki i przesyła do funkcji aktywacyjnej. Wynik funkcji aktywacyjnej określa czy perceptron został wzbudzony czy nie.



Wykorzystany algorytm uczenia:

Zastosowano tu algorytm uczenia z nauczycielem za pomocą reguły Widrowa Hoffa.

Obiekt typu Perceptron wywołuje funkcje train przyjmującą dane wejściowe oraz oczekiwany wynik. Następnie perceptron „zgaduje jaka będzie odpowiedź”, w zależności od tego czy „zgadł” czy nie wagi wejść oraz biasu zostają zmienione lub nie.

2) Testowaną funkcją logiczną był AND

Wyniki:

Funkcja sprawdzająca poprawność sprawdzała ją dla wszystkich 4 możliwych przypadków.

Formatka:Liczba przejść – procent poprawnie

Dla 10000 testów:

a)Przy współczynniku uczenia 1, dla nauczania zestawami po 4 („0 0”, „0 1”, „1 0”, „1 1”).

|  |  |
| --- | --- |
| Liczba przejść (po 4 czyli żeby otrzymać liczbę treningów perceptronu trzeba \*4) | Procent w pełni poprawnie nauczonych perceptronów |
| 0 | 0.64% |
| 1 | 5.47% |
| 2 | 6.63% |
| 3 | 11.91% |
| 4 | 20.80% |
| 5 | 76.99% |
| 6 | 80.32% |
| 7 | 89.55% |
| 8 | 100% |

b)Przy współczynniku uczenia 0.5, dla nauczania zestawami po 4 („0 0”, „0 1”, „1 0”, „1 1”).

|  |  |
| --- | --- |
| Liczba przejść (po 4 czyli żeby otrzymać liczbę treningów perceptronu trzeba \*4) | Procent w pełni poprawnie nauczonych perceptronów |
| 0 | 0.5% |
| 1 | 6.5% |
| 2 | 16.87% |
| 3 | 27.89% |
| 4 | 37.11% |
| 5 | 61.07% |
| 6 | 87.34% |
| 7 | 93.33% |
| 8 | 98.67% |
| 9 | 100% |

c)Przy współczynniku uczenia 0.1, dla nauczania zestawami po 4 („0 0”, „0 1”, „1 0”, „1 1”).

|  |  |
| --- | --- |
| Liczba przejść (po 4 czyli żeby otrzymać liczbę treningów perceptronu trzeba \*4) | Procent w pełni poprawnie nauczonych perceptronów |
| 0 | 0.58% |
| 1 | 3.29% |
| 2 | 8.9% |
| 3 | 18.26% |
| 4 | 29.54% |
| 5 | 39.7% |
| 6 | 50.21% |
| 7 | 60.18% |
| 8 | 69.7% |
| 9 | 80.32% |
| 10 | 87.87% |
| 11 | 93.67% |
| 12 | 97.24% |
| 13 | 99.00% |
| 14 | 99.76% |
| 15 | 99.995% |
| 16 | 100% |

d)Przy współczynniku uczenia 1000, dla nauczania zestawami po 4 („0 0”, „0 1”, „1 0”, „1 1”).

|  |  |
| --- | --- |
| Liczba przejść (po 4 czyli żeby otrzymać liczbę treningów perceptronu trzeba \*4) | Procent w pełni poprawnie nauczonych perceptronów |
| 0 | 0.64% |
| 1 | 1.82% |
| 2 | 8.01% |
| 3 | 7.97% |
| 4 | 7.71% |
| 5 | 83.99% |
| 6 | 86.53% |
| 7 | 94.51% |
| 8 | 100% |

e) Przy współczynniku uczenia 0.1, dla losowych danych wejściowych z wszystkich możliwych przypadków

|  |  |
| --- | --- |
| Liczba przejść | Procent w pełni poprawnie nauczonych perceptronów |
| 0 | 1.07% |
| 1 | 1.55% |
| 2 | 1.63% |
| 3 | 2.36% |
| 4 | 3.10% |
| 5 | 4.16% |
| 10 | 11.36% |
| 20 | 27.51% |
| 30 | 43.25% |
| 40 | 60.76% |
| 50 | 74.86% |
| 60 | 86.18% |
| 70 | 93.54% |
| 80 | 97.68% |
| 90 | 98.98% |
| 100 | 99.81% |
| 120 | ~100%(czasami 99.99%) |

f) Przy współczynniku uczenia 0.5, dla nauczania zestawami po 3 („0 0”, „0 1”, „1 1”):

|  |  |
| --- | --- |
| Liczba przejść (po 3 czyli żeby otrzymać liczbę treningów perceptronu trzeba \*3) | Procent w pełni poprawnie nauczonych perceptronów |
| 0 | 0.49% |
| 1 | 5.41% |
| 2 | 11.81% |
| 3 | 16.40% |
| 4 | 16.52% |

Potem niezależnie od ilości testów wynik pozostaje w tych samych okolicach, czasem rośnie czasem maleje.

g) Przy współczynniku uczenia 0.5, dla nauczania zestawami po 3 („0 0”, „0 1”, „1 0”):

|  |  |
| --- | --- |
| Liczba przejść (po 3 czyli żeby otrzymać liczbę treningów perceptronu trzeba \*3) | Procent w pełni poprawnie nauczonych perceptronów |
| 0 | 0.86% |
| 1 | 6.16% |
| 2 | 6.23% |

Potem niezależnie od ilości testów wynik pozostaje w tych samych okolicach, czasem rośnie czasem maleje.

h) Przy współczynniku uczenia 0.5, dla nauczania zestawami po 3 („0 0”, „1 0”, „1 1”):

|  |  |
| --- | --- |
| Liczba przejść (po 3 czyli żeby otrzymać liczbę treningów perceptronu trzeba \*3) | Procent w pełni poprawnie nauczonych perceptronów |
| 0 | 0.70% |
| 1 | 5.21% |
| 2 | 10.74% |
| 3 | 15.81% |
| 4 | 16.92% |

Potem niezależnie od ilości testów wynik pozostaje w tych samych okolicach, czasem rośnie czasem maleje.

i)Przy współczynniku uczenia 0.5, dla nauczania zestawami po 3 („0 1”, „1 0”, „1 1”):

|  |  |
| --- | --- |
| Liczba przejść (po 3 czyli żeby otrzymać liczbę treningów perceptronu trzeba \*3) | Procent w pełni poprawnie nauczonych perceptronów |
| 0 | 0.76% |
| 1 | 3.79% |
| 2 | 10.81% |
| 3 | 15.03% |
| 4 | 21.84% |
| 5 | 36.56% |
| 6 | 57.89% |
| 7 | 76.85% |
| 8 | 97.13% |
| 9 | 99.37% |
| 10 | 99.41% |
| 11 | 100% |

(Zwiększenie współczynnika nauczania do 1000 powoduje że dopiero dla 12 -100%)

WNIOSKI:

1)Uczenie perceptronu wszystkimi możliwymi danymi podawanymi po kolei jest znacznie szybsze niż wszystkimi możliwymi danych podawanymi losowo.

2)Najważniejszym przykładem dostarczanym do perceptronu uczącego się AND jest (1,1) gdyż bez niego max wydajność wynosi 6.23%. Natomiast najmniej ważnym jest (0,0) gdyż nawet bez niego można uzyskać w pełni nauczony perceptron.

3) Z powyższego – Przykłady podawane do nauki nie są sobie równe i czasami jeden przykład jest zbędny a inny krytyczny do otrzymania działającego perceptronu.

4)Ustawienie zbyt małego współczynnika nauczania znacznie zwiększa ilość treningu przez jaką musi przejść perceptron.

5)Ustawienie bardzo dużego współczynnika nauczania(np. 1000) powoduje dziwny i bardzo niesymetryczny przyrost poprawnie działających perceptronów wraz z ilością treningów. Dzieje się tak najprawdopodobniej z powodu dużych skoków wag co oznacza że trudniej trafić we właściwe ułożenie.

6)Zmiana funkcji aktywacyjnej kompletnie zmienia wyniki

7)Dla niekompletnych danych wejściowych, nie można być pewnym czy perceptron zawsze dobrze się wykona bez testów, podczas gdy jeśli wszystkie przypadki są w danych początkowych to mamy pewność że po skończonej liczbie wywołania uczenia perceptron zawsze będzie działał prawidłowo.

8) Z powyższego sieci neuronowe nie są niezawodne jednak w teorii mogą być (trzeba by wprowadzić wszystkie możliwe przypadki co jest niemożliwe a nawet jeśli byłaby taka możliwość to zajęła by za dużo czasu).

LISTING:

main:

package mypackage;

import Operations.AND;

import Operations.OR;

import Operations.XOR;

import java.util.ArrayList;

import java.util.Collection;

public class Main {

public static Integer test\_AND(Perceptron perceptron)//funkcja testująca

{

ArrayList<Integer> and = new ArrayList<Integer>();

for(int i=0;i<2;i++) {

for(int j=0;j<2;j++) {

and.add(i);

and.add(j);

if (i == 1 && j == 1) {

if (perceptron.guess(and) != 1) {

return 0;

}

and.clear();

} else {

if (perceptron.guess(and) != -1) {

return 0;

}

and.clear();

}

}

}

return 1;

}

public static void train\_AND(Perceptron perceptron,Integer number\_of\_trainings)

{

for(int i=0;i<number\_of\_trainings;i++) {

AND and = new AND();

perceptron.train(and);

}

}

public static void train\_AND\_block(Perceptron perceptron,Integer number\_of\_trainings\_blocks)

{

AND and00 = new AND(0,0);

AND and01 = new AND(0,1);

AND and10 = new AND(1,0);

AND and11 = new AND(1,1);

for(int i=0;i<number\_of\_trainings\_blocks;i++) {

perceptron.train(and00);

perceptron.train(and01);

perceptron.train(and10);

perceptron.train(and11);

}

}

public static void main(String[] args) {

Perceptron perceptron = new Perceptron(2);

perceptron.show();

Perceptron[] per = new Perceptron[10001];

//tests

Integer correct =0;

Integer tests = 10000;

for(int z=0;z<=15;z++) { //z - liczba wykonanych testów

correct=0;

for (int i = 0; i < tests; i++) {

per[i] = new Perceptron(2);

//train\_AND(per[i],125);

*train\_AND\_block*(per[i], z);

correct += *test\_AND*(per[i]);

}

System.*out*.println(correct + " na " + tests + " poprawnie");

}

}

}

Perceptron:

package mypackage;

import Operations.Data;

import java.util.ArrayList;

import java.util.concurrent.ThreadLocalRandom;

*/\*\**

*\* Created by qwerty on 17-Oct-17.*

*\*/*

public class Perceptron {

private ArrayList<Double> weights;

private Double biasweight;

private Double lr = 0.5;

public Double getLr() {

return lr;

}

public void setLr(Double lr) {

this.lr = lr;

}

private int activation(Double sum)

{

if(sum>=0.0)

{

return 1;

}

else

{

return -1;

}

}

Perceptron(int size)

{

weights=new ArrayList<Double>();

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

weights.add(ThreadLocalRandom.*current*().nextDouble(-1, 1 + 1));

}

biasweight=ThreadLocalRandom.*current*().nextDouble(-1, 1 + 1);

}

public void show()

{

System.*out*.println("Wagi: ");

for(int i=0;i<weights.size();i++) {

System.*out*.println(weights.get(i));

}

System.*out*.println("Waga biasu:\n" + biasweight);

}

public Integer guess(ArrayList<Integer> inputs)

{

Double sum= 0.0;

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

{

sum+=inputs.get(i)\*weights.get(i);

}

sum+=biasweight;

Integer out = activation(sum);

return out;

}

public void train(Data data)

{

int guess =guess(data.getIn());

Integer error = data.getLabel() - guess;

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

{

Double a = weights.get(i);

weights.set(i,a+error\*data.getIn().get(i)\*lr);

}

biasweight+=error\*lr;

}

}

AND:

package Operations;

import java.util.ArrayList;

import java.util.concurrent.ThreadLocalRandom;

*/\*\**

*\* Created by qwerty on 17-Oct-17.*

*\*/*

public class AND extends Data{

public AND()

{

x= ThreadLocalRandom.*current*().nextInt(0,2);

y= ThreadLocalRandom.*current*().nextInt(0,2);

if(x.equals(1)&&x.equals(y))

{

label=1;

}

else

{

label=-1;

}

in = new ArrayList<Integer>();

in.add(x);

in.add(y);

}

public AND(int a,int b)

{

x= a;

y= b;

if(x.equals(1)&&x.equals(y))

{

label=1;

}

else

{

label=-1;

}

in = new ArrayList<Integer>();

in.add(x);

in.add(y);

}

}

Data:

package Operations;

import java.util.ArrayList;

public abstract class Data {

protected ArrayList<Integer> in;

protected Integer x;

protected Integer y;

protected int label;

public Integer getX() {

return x;

}

public Integer getY() {

return y;

}

public int getLabel() {

return label;

}

public ArrayList<Integer> getIn() {

return in;

}

public void show()

{

System.*out*.println("X: "+ x +"\nY: "+ y + "\nWynik: " + label);

}

}