|  |  |
| --- | --- |
| **Scenariusz 1** | |
| **Temat ćwiczenia:** | Budowa i działanie perceptronu |
| **Wykonał:** | Mateusz Mazur, gr 2 |

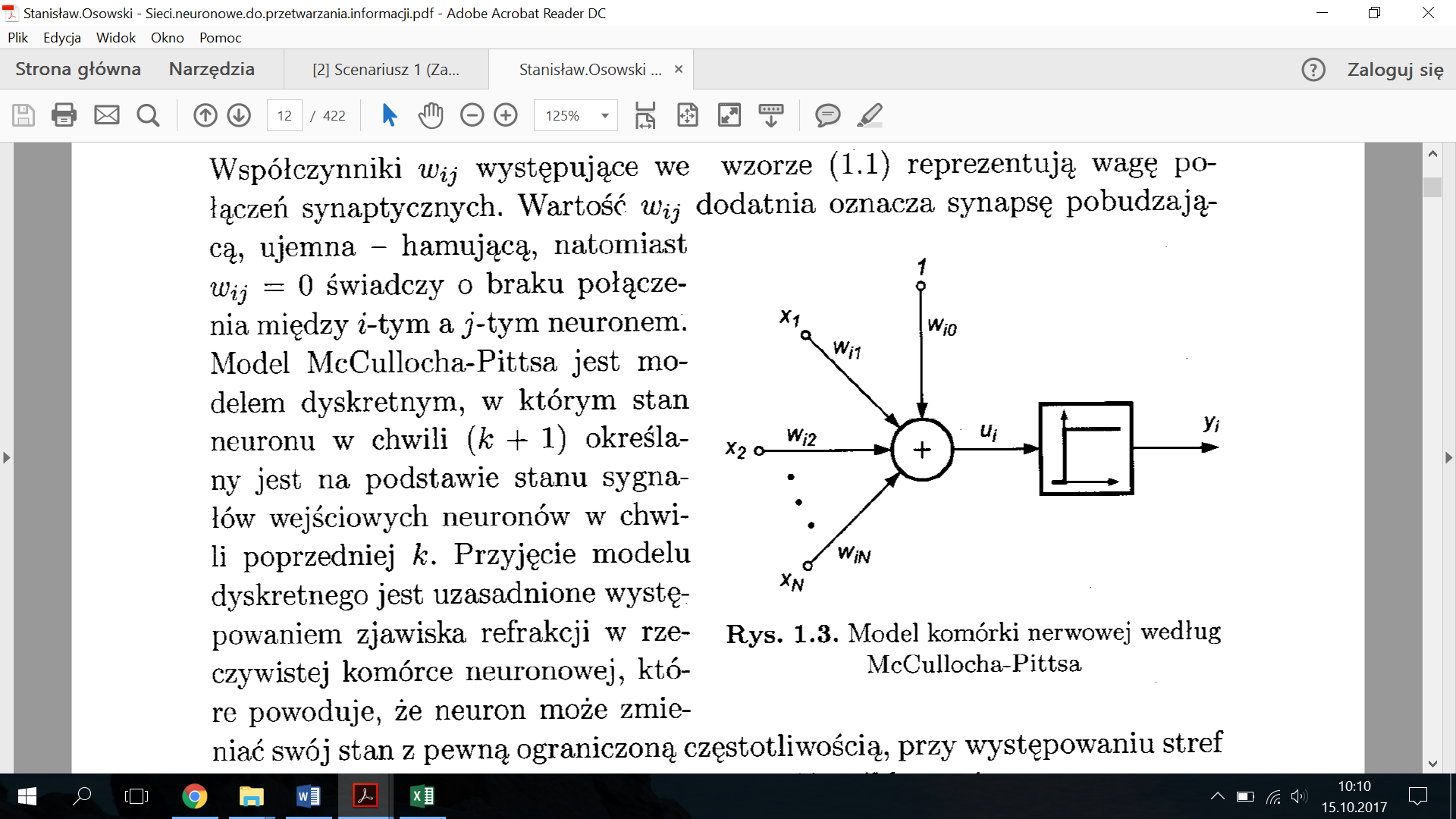
1. **Cel ćwiczenia**

Celem ćwiczenia jest poznanie budowy i działanie perceptronu poprzez implementację oraz uczenie perceptronu realizującego wybraną funkcję logiczną dwóch zmiennych.

1. **Budowa perceptronu i wykorzystany algorytm uczenia**

Perceptron został zaimplementowany zgodnie z modelem McCullocha-Pittsa. Sygnały wejściowe xj (j = 1, 2, …, N) są sumowane z odpowiednimi wagami wij w sumatorze, a następnie jest dodawany sygnał polaryzacji x0 = 1 dla którego waga wynosiwio. Sygnał wyjściowy neuronu yi wyraża się, więc zależnością:

gdzie funkcje wyrażamy wzorem:



Rys. 1 Perceptron według modelu McCullocha-Pittsa[1]

Model ten jest modelem dyskretnym, w którym stan neuronu w chwili (k+1) określony jest na podstawie stanu sygnałów wejściowych neuronów w chwili poprzedniej k.

Do uczenia perceptronu wykorzystano regułę Widrowa-Hoffa uzupełnioną o współczynnik uczenia . Na wstępie wagi dobrano w sposób losowy z zakresu 0-1. Dzięki nim obliczamy wartość sygnału wyjściowego yi. W wyniku porównania aktualnej wartości yi oraz wartości zadanej di dokonuje się aktualizacji wag. Jeżeli wartości otrzymana i zadana się nie różnią to wagi pozostają niezmienione. A więc zgodnie z tą regułą dobór wag odbywa się według zależności

Podczas uczenia wykorzystuje się jedynie informacje o aktualnej wartości sygnału wyjściowego neuronu oraz wartości zadanej[1].

1. **Otrzymane wyniki**

Testowanie zaimplementowanego perceptronu odbywało się przy zmianie liczby danych uczących oraz stałym współczynniku uczenia równym 0,0099. Następnie dla każdego z przypadków uruchamiane były dane testujące w liczbie 50 sprawdzające poprawność procesu uczenia.

Drugim z przeprowadzonych testów było sprawdzenie jak zachowuje się sieć przy różnych współczynnikach uczenia oraz stałej liczbie danych uczących wynoszącej 80. Tutaj również po zakończeniu uczenia sprawdzono jego efektywność dzięki wcześniej przygotowanym danym testującym.

Tab. 1 Dane uczące i testujące wczytywane przez program

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dane uczące | | | Dane testujące | | |
| Wejście 1 | Wejście 2 | Prawidłowa odpowiedź | Wejście 1 | Wejście 2 | Prawidłowa odpowiedź |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 0 |

Tab. 2 Otrzymane wyniki przy zmianie liczby danych uczących

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Statystyki procesu uczenia przy różnej liczbie danych uczących i stałym współczynniku uczenia równym: 0,0099 | | | | | | |
|
| Proces uczenia: | | | | Proces testowania: | | |
| Liczba danych uczących | Liczba dobrych odpowiedzi | Liczba złych odpowiedzi | Procent dobrych odpowiedzi | Liczba dobrych odpowiedzi | Liczba złych odpowiedzi | Procent dobrych odpowiedzi |
|
| 20 | 6 | 14 | 30,0% | 16 | 34 | 32,0% |
| 40 | 15 | 25 | 37,5% | 17 | 33 | 34,0% |
| 60 | 24 | 36 | 40,0% | 19 | 31 | 38,0% |
| 80 | 37 | 43 | 46,3% | 30 | 20 | 60,0% |
| 100 | 58 | 42 | 58,0% | 41 | 9 | 82,0% |

Tab. 3 Otrzymane wyniki przy zmianie współczynnika uczenia

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Statystyki procesu uczenia przy stałej liczbie danych uczących równej 80 i różnym współczynniku uczenia | | | | | | |
|
|
| Proces uczenia: | | | | Proces testowania: | | |
| Współczynnik uczenia | Liczba dobrych odpowiedzi | Liczba złych odpowiedzi | Procent dobrych odpowiedzi | Liczba dobrych odpowiedzi | Liczba złych odpowiedzi | Procent dobrych odpowiedzi |
|
| 0,001 | 29 | 51 | 36,3% | 16 | 34 | 32,0% |
| 0,0099 | 37 | 43 | 46,3% | 30 | 20 | 60,0% |
| 0,01 | 38 | 42 | 47,5% | 39 | 11 | 78,0% |
| 0,1 | 70 | 10 | 87,5% | 50 | 0 | 100,0% |

1. **Analiza wyników**

Wykres nr. 1 Procentowa zależność poprawnych odpowiedzi od liczby danych uczących

Na podstawie wykresu można stwierdzić, że wraz ze wzrostem liczby danych uczących procent dobrych odpowiedzi się zwiększa. Po przeprowadzeniu na naszym nauczonym neuronie procesu testowania zauważamy, iż procentowy współczynnik dobrych odpowiedzi rośnie wraz z liczbą danych uczących, aż do szczytowej wartości równej 100% poprawnych odpowiedzi dla liczby danych uczących równej 100. Na podstawie uzyskanych wyników możemy stwierdzić, iż dla początkowej zmiany liczby danych uczących wzrost efektywności mojego neuronu był raczej niewielki, na poziomie kilku procent, natomiast po zwiększeniu liczby danych uczących do 80 zaobserwowano prawie dwukrotny wzrost liczby poprawnej odpowiedzi, a dodanie kolejnych 20 danych uczących spowodowało kolejny znaczący wzrost umiejętności mojego neuronu. Pozwala to stwierdzić, iż jeżeli potrzebujemy neuronu, który cechuje się bardzo wysoką skutecznością w rozwiązywaniu problemów, musimy przygotować dla niego bardzo dużą bazę do nauki.

Wykres nr. 2 Procentowa zależność poprawnych odpowiedzi od wartości współczynnika uczenia

Na powyższym wykresie możemy obserwować wzrost procentowego udziału poprawnych odpowiedzi w zależności od współczynnika uczenia. Zwiększanie współczynnika uczenia w sposób liniowy wpływa na obliczanie nowych wag na wejściach w przypadku podania przez neuron nieprawidłowej odpowiedzi. W wyniku czego wyższy współczynnik w sposób znaczny wpływa na umiejętność naszego neuronu do wskazywania prawidłowych odpowiedzi. Możemy zauważyć, iż zmiana nawet o kilka tysięcznych może spowodować nawet 100% wzrost efektywności nauki naszego neuronu. Zastosowany przeze mnie neuron przy liczbie danych uczących równej 80, był w stanie osiągnąć 100% skuteczność w wyznaczaniu rozwiązania logicznej funkcji AND już przy współczynniku uczenia równym 0,1.

1. **Wnioski**

* Efektywność procesu uczenia zależna jest od liczby danych uczących. Wraz z wzrostem liczby danych uczących proces ten wykazuje większą skuteczność. Jeśli liczba ta jest dobrana w sposób nieprawidłowy tzn. nasza baza danych uczących jest zbyt mała to otrzymany perceptron nie będzie działał w sposób poprawny, a co za tym idzie nie będzie spełniał swojej funkcji. Trzeba się więc liczyć z tym, że konieczne jest stworzenie dość dużej bazy z danymi uczącymi.
* Drugim czynnikiem wpływającym za efektywność procesu uczenia jest współczynnik uczenia uwzględniany przy obliczaniu nowych wag dla danych. Im jego wartość jest wyższa tym efektywność uczenia jest większa. Jest to wytłumaczalne z faktu, że w takich sytuacjach wagi zmieniają się szybciej i automatycznie szybciej osiągają poprawną wartość.
* Z poprzedniego wniosku wynika, że przy zastosowaniu odpowiednio wysokiego współczynnika uczenia można by zrezygnować z dużej ilości danych uczących, co niewątpliwie usprawniło i przyśpieszyło sam proces uczenia neuronu.
* Dzięki otrzymanym danym można stwierdzić, że jednowarstwowe sieci neuronowe doskonale nadają się do wykonywania prostych funkcji logicznych. Po odpowiednio przeprowadzonym procesie uczenia otrzymano bezbłędnie działający perceptron wykonujący funkcję logiczną AND dwóch zmiennych.

1. **Kod programu**

**main.cpp:**

#include <iostream>

#include <vector>

#include "Neuron.h"

#include <time.h>

#include <fstream>

using namespace std;

void setInputValue(Neuron& neuron, bool testing);

void learn(Neuron& neuron, bool testing);

fstream OUTPUT\_LEARNING\_FILE;

fstream OUTPUT\_TESTING\_FILE;

fstream LEARNING\_DATA;

fstream TESTING\_DATA;

int BAD\_ANSWERS\_AMOUNT;

int RIGHT\_ANSWERS\_AMMOUNT;

int CORRECT\_ANSWER;

int main()

{

srand(time(NULL));

Neuron neuron(3, 0.00999);

OUTPUT\_LEARNING\_FILE.open("output\_learning\_data.txt", ios::out);

OUTPUT\_TESTING\_FILE.open("output\_testing\_data.txt", ios::out);

LEARNING\_DATA.open("learning\_data.txt", ios::in);

TESTING\_DATA.open("testing\_data.txt", ios::in);

do

{

cout << "1. Learn" << endl;

cout << "2. Test " << endl;

cout << "3. Exit" << endl;

int choice;

cin >> choice;

switch (choice)

{

case 1:

int loopIteration;

BAD\_ANSWERS\_AMOUNT = 0;

RIGHT\_ANSWERS\_AMMOUNT = 0;

cout << "Enter number of loops: " << endl;

cin >> loopIteration;

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

{

learn(neuron, false);

}

OUTPUT\_LEARNING\_FILE << "Bad answers amount: " <<  
 BAD\_ANSWERS\_AMOUNT << endl;

OUTPUT\_LEARNING\_FILE << "Right answers amount: " <<   
 RIGHT\_ANSWERS\_AMMOUNT << endl;

break;

case 2:

BAD\_ANSWERS\_AMOUNT = 0;

RIGHT\_ANSWERS\_AMMOUNT = 0;

for (int i = 0; i < 50; i++)

{

learn(neuron, true);

}

OUTPUT\_TESTING\_FILE << "Bad answers amount: " <<   
 BAD\_ANSWERS\_AMOUNT << endl;

OUTPUT\_TESTING\_FILE << "Right answers amount: " <<  
 RIGHT\_ANSWERS\_AMMOUNT << endl;

break;

case 3:

OUTPUT\_LEARNING\_FILE.close();

OUTPUT\_TESTING\_FILE.close();

TESTING\_DATA.close();

LEARNING\_DATA.close();

return 0;

}

} while (true);

return 0;

}

void setInputValue(Neuron& neuron, bool testing)

{

if (testing == false)

{

int input1, input2, input3, answer;

LEARNING\_DATA >> input1;

LEARNING\_DATA >> input2;

LEARNING\_DATA >> input3;

LEARNING\_DATA >> answer;

neuron.setInput(0, input1);

neuron.setInput(1, input2);

neuron.setInput(2, input3);

CORRECT\_ANSWER = answer;

OUTPUT\_LEARNING\_FILE << "IN 0b: " << neuron.getInput(0);

OUTPUT\_LEARNING\_FILE << " IN 1: " << neuron.getInput(1);

OUTPUT\_LEARNING\_FILE << " IN 2: " << neuron.getInput(2);

OUTPUT\_LEARNING\_FILE << " RA: " << CORRECT\_ANSWER;

}

else

{

int input1, input2, input3, answer;

TESTING\_DATA >> input1;

TESTING\_DATA >> input2;

TESTING\_DATA >> input3;

TESTING\_DATA >> answer;

neuron.setInput(0, input1);

neuron.setInput(1, input2);

neuron.setInput(2, input3);

CORRECT\_ANSWER = answer;

OUTPUT\_TESTING\_FILE << "IN 0b: " << neuron.getInput(0);

OUTPUT\_TESTING\_FILE << " IN 1: " << neuron.getInput(1);

OUTPUT\_TESTING\_FILE << " IN 2: " << neuron.getInput(2);

OUTPUT\_TESTING\_FILE << " RA: " << CORRECT\_ANSWER;

}

}

void learn(Neuron& neuron, bool testing)

{

setInputValue(neuron, testing);

neuron.sumOfInputs();

int neuronAnswer = neuron.processOutput();

int \*inputArray = new int[2];

inputArray[0] = neuron.getInput(1);

inputArray[1] = neuron.getInput(2);

if (neuronAnswer) //neuron answer = 1

{

if (CORRECT\_ANSWER == 0)

{

BAD\_ANSWERS\_AMOUNT++;

if (testing == false)

{

OUTPUT\_LEARNING\_FILE << " A: 1";

neuron.countNewWeights(CORRECT\_ANSWER, neuronAnswer,   
 inputArray);

OUTPUT\_LEARNING\_FILE << " -" << " CHANGING WEIGHTS: " <<   
 neuron.getSynapse(0) << " " << neuron.getSynapse(1) << " "

<< neuron.getSynapse(2) << endl;

}

else

{

OUTPUT\_TESTING\_FILE << " A: 1 -" << endl;

}

}

else

{

RIGHT\_ANSWERS\_AMMOUNT++;

if (testing == false)

{

OUTPUT\_LEARNING\_FILE << " A: 1";

OUTPUT\_LEARNING\_FILE << " +" << " KEEPING WEIGHTS: " <<   
 neuron.getSynapse(0) << " " << neuron.getSynapse(1) << " " <<   
 neuron.getSynapse(2) << endl;

}

else

{

OUTPUT\_TESTING\_FILE << " A: 1 +" << endl;

}

}

}

else //neuron answer = 0

{

if (CORRECT\_ANSWER == 1)

{

BAD\_ANSWERS\_AMOUNT++;

if (testing == false)

{

OUTPUT\_LEARNING\_FILE << " A: 0";

neuron.countNewWeights(CORRECT\_ANSWER, neuronAnswer,  
 inputArray);

OUTPUT\_LEARNING\_FILE << " -" << " CHANGING WEIGHTS: " <<   
 neuron.getSynapse(0) << " " << neuron.getSynapse(1) << " "   
 << neuron.getSynapse(2) << endl;

}

else

{

OUTPUT\_TESTING\_FILE << " A: 0 -" << endl;

}

}

else

{

RIGHT\_ANSWERS\_AMMOUNT++;

if (testing == false)

{

OUTPUT\_LEARNING\_FILE << " A: 0";

OUTPUT\_LEARNING\_FILE << " +" << " KEEPING WEIGHTS: " <<   
 neuron.getSynapse(0) << " " << neuron.getSynapse(1) << " "   
 << neuron.getSynapse(2) << endl;

}

else

{

OUTPUT\_TESTING\_FILE << " A: 0 +" << endl;

}

}

}

}

**Neuron.h:**

#pragma once

#include <iostream>

#include <vector>

using namespace std;

class Neuron

{

public:

void createInputs(int amount);

void createDendrite() { \_dendrites.push\_back(0); }

void createSynapse() { \_synapses.push\_back(0); }

int getDendritesAmount() { return \_dendrites.size(); }

double getInput(int index) { return \_dendrites[index]; }

void setInput(int index, double value) { \_dendrites[index] = value; }

double getSynapse(int index) { return \_synapses[index]; }

void setSynapse(int index, double value) { \_synapses[index] = value; }

double getSumOfAllInputs() { return \_sumOfAllInputs; }

int getOutputValue(){ return \_outputValue; }

double processSingleInput(int index) { return \_dendrites[index] \* \_synapses[index]; }

void sumOfInputs();

int processOutput();

void countNewWeights(int rightAnswer, int neuronAnswer, int \*inputArray);

Neuron(int amount, double learningCoefficient)

{

createInputs(amount);

\_learningCoefficient = learningCoefficient;

}

private:

double countFirstWeight();

vector<double> \_dendrites; //inputs

vector<double> \_synapses; //weights

double \_sumOfAllInputs;

int \_outputValue;

double \_learningCoefficient;

int \_size;

};

**Neuron.cpp:**

#include "Neuron.h"

#include <time.h>

void Neuron::createInputs(int amount)

{

double weight = countFirstWeight();

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

{

\_dendrites.push\_back(0);

\_synapses.push\_back(countFirstWeight());

}

}

void Neuron::sumOfInputs()

{

if (getDendritesAmount() == 0)

{

return;

}

else

{

double sum = 0;

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

{

sum += processSingleInput(i);

}

\_sumOfAllInputs = sum;

}

}

int Neuron::processOutput()

{

if (getDendritesAmount() == 0)

{

return -1;

}

else

{

if (\_sumOfAllInputs > 0)

{

\_outputValue = 1;

return \_outputValue;

}

else

{

\_outputValue = 0;

return \_outputValue;

}

}

}

void Neuron::countNewWeights(int rightAnswer, int neuronAnswer, int\* inputArray)

{

\_synapses[0] += \_learningCoefficient \* (rightAnswer - neuronAnswer);

for (int i = 1; i < getDendritesAmount(); i++)

{

\_synapses[i] += \_learningCoefficient \* (rightAnswer - neuronAnswer) \* inputArray[i-1];

}

}

//private methods

double Neuron::countFirstWeight()

{

double weight = ((double)rand() / (RAND\_MAX));

return weight;

}

1. Literatura

[1] Osowski S., Sieci neuronowe do przetwarzania informacji, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2006