|  |  |
| --- | --- |
| **Scenariusz 4** | |
| **Temat ćwiczenia:** | Uczenie sieci regułą Hebba |
| **Wykonał:** | Mateusz Mazur, gr 2 |

1. **Cel ćwiczenia**

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

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

Celem budowanej sieci jest podział wzorców uczących na klasy obrazów zbliżonych do siebie i przyporządkowanie każdej klasie osobnego elementu wyjściowego. Podział na grupy odbywa się w takich sposób, by elementy w tej samej grupie były do siebie podobne a jednocześnie jak najbardziej odmienne od elementów z pozostałych grup.

Tab. 1 Przygotowany alfabet

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0110  1001  1111  1001  1001 | D:\Pobrane\Alfabet (1)\Alfabet\A.png | 1110  1001  1110  1001  1110 |
| D:\Pobrane\Alfabet (1)\Alfabet\G.png | 0110  1001  1000  1001  0110 | D:\Pobrane\Alfabet (1)\Alfabet\D.png | 1110  1001  1001  1001  1110 |
| D:\Pobrane\Alfabet (1)\Alfabet\D.png | 1111  1000  1111  1000  1111 | D:\Pobrane\Alfabet (1)\Alfabet\F.png | 1111  1000  1110  1000  1000 |
| D:\Pobrane\Alfabet (1)\Alfabet\F.png | 1111  1000  1011  1001  1111 | D:\Pobrane\Alfabet (1)\Alfabet\H.png | 1001  1001  1111  1001  1001 |
| D:\Pobrane\Alfabet (1)\Alfabet\J.png | 1111  0110  0110  0100  1111 | D:\Pobrane\Alfabet (1)\Alfabet\J.png | 1111  0001  0011  0101  1111 |
| D:\Pobrane\Alfabet (1)\Alfabet\L.png | 1001  1010  1100  1010  1001 | D:\Pobrane\Alfabet (1)\Alfabet\L.png | 1000  1000  1000  1000  1111 |
| D:\Pobrane\Alfabet (1)\Alfabet\B.png | 1001  1001  1101  1011  1001 | D:\Pobrane\Alfabet (1)\Alfabet\B.png | 0110  1001  1001  1001  0110 |
| D:\Pobrane\Alfabet (1)\Alfabet\P.png | 1111  1001  1111  1000  1000 | D:\Pobrane\Alfabet (1)\Alfabet\P.png | 1111  1001  1111  1010  1001 |
| D:\Pobrane\Alfabet (1)\Alfabet\H.png | 1111  1111  0110  0110  0110 | D:\Pobrane\Alfabet (1)\Alfabet\E.png | 1001  1001  1001  1001  1111 |
| D:\Pobrane\Alfabet (1)\Alfabet\E.png | 1001  1001  1111  0110  0110 | D:\Pobrane\Alfabet (1)\Alfabet\G.png | 1111  0001  0110  1000  1111 |

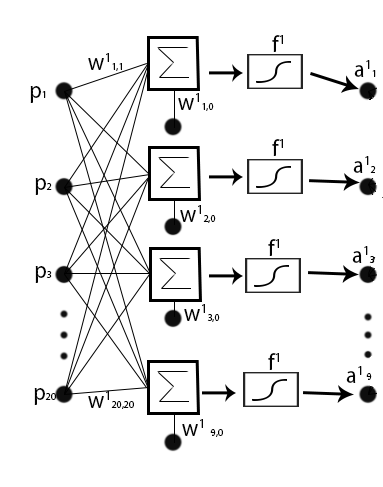
Klasyfikacja liter na poszczególne grupy odbywa się na podstawie obliczonej odległości euklidesowej między poszczególnymi literami według wzoru:

Tab 2. Zestawienie wartości odległości euklidesowych między poszczególnymi literami

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F | G | H | I | J | K | L | N | O | P | R | T | U | Y | Z |
| A | x | 2,24 | 2,65 | 2,45 | 2,45 | 2,45 | 2,45 | 2,00 | 3,74 | 3,00 | 3,16 | 3,16 | 2,45 | 2,45 | 2,00 | 2,00 | 3,74 | 2,83 | 3,46 | 2,83 |
| B | 2,24 | x | 2,00 | 1,73 | 2,24 | 2,24 | 2,24 | 2,65 | 3,32 | 2,83 | 3,32 | 2,65 | 3,16 | 2,24 | 2,24 | 2,65 | 3,00 | 2,83 | 3,00 | 2,24 |
| C | 2,65 | 2,00 | x | 1,73 | 3,00 | 3,00 | 2,65 | 3,32 | 3,87 | 3,16 | 3,61 | 2,65 | 3,32 | 1,00 | 3,00 | 3,32 | 3,16 | 2,65 | 3,32 | 3,00 |
| D | 2,45 | 1,73 | 1,73 | x | 2,45 | 2,83 | 2,00 | 2,83 | 3,74 | 2,65 | 3,46 | 2,45 | 2,83 | 1,41 | 2,45 | 2,83 | 3,46 | 2,00 | 3,16 | 2,83 |
| E | 2,45 | 2,24 | 3,00 | 2,45 | x | 2,00 | 1,41 | 2,45 | 2,83 | 2,65 | 2,83 | 2,45 | 2,83 | 2,83 | 2,00 | 2,00 | 3,16 | 2,45 | 2,83 | 2,00 |
| F | 2,45 | 2,24 | 3,00 | 2,83 | 2,00 | x | 2,45 | 2,45 | 3,16 | 3,32 | 2,45 | 2,83 | 2,83 | 3,16 | 1,41 | 2,00 | 3,16 | 3,16 | 3,16 | 2,45 |
| G | 2,45 | 2,24 | 2,65 | 2,00 | 1,41 | 2,45 | x | 2,45 | 3,16 | 2,24 | 3,16 | 2,45 | 2,83 | 2,45 | 2,45 | 2,45 | 3,46 | 2,00 | 3,16 | 2,45 |
| H | 2,00 | 2,65 | 3,32 | 2,83 | 2,45 | 2,45 | 2,45 | x | 3,74 | 3,00 | 2,45 | 2,83 | 1,41 | 3,16 | 2,00 | 2,00 | 3,74 | 2,00 | 2,83 | 2,83 |
| I | 3,74 | 3,32 | 3,87 | 3,74 | 2,83 | 3,16 | 3,16 | 3,74 | x | 2,65 | 3,16 | 3,46 | 3,74 | 4,00 | 3,46 | 3,46 | 2,00 | 3,74 | 3,16 | 2,45 |
| J | 3,00 | 2,83 | 3,16 | 2,65 | 2,65 | 3,32 | 2,24 | 3,00 | 2,65 | x | 3,87 | 3,32 | 3,32 | 3,00 | 3,00 | 3,00 | 3,00 | 2,65 | 3,00 | 2,24 |
| K | 3,16 | 3,32 | 3,61 | 3,46 | 2,83 | 2,45 | 3,16 | 2,45 | 3,16 | 3,87 | x | 2,45 | 2,00 | 3,74 | 2,83 | 2,45 | 3,46 | 2,83 | 3,16 | 3,16 |
| L | 3,16 | 2,65 | 2,65 | 2,45 | 2,45 | 2,83 | 2,45 | 2,83 | 3,46 | 3,32 | 2,45 | x | 2,83 | 2,83 | 3,16 | 3,16 | 3,74 | 2,00 | 3,16 | 2,83 |
| N | 2,45 | 3,16 | 3,32 | 2,83 | 2,83 | 2,83 | 2,83 | 1,41 | 3,74 | 3,32 | 2,00 | 2,83 | x | 3,16 | 2,45 | 2,00 | 3,74 | 2,00 | 2,83 | 3,16 |
| O | 2,45 | 2,24 | 1,00 | 1,41 | 2,83 | 3,16 | 2,45 | 3,16 | 4,00 | 3,00 | 3,74 | 2,83 | 3,16 | x | 2,83 | 3,16 | 3,46 | 2,45 | 3,16 | 3,16 |
| P | 2,00 | 2,24 | 3,00 | 2,45 | 2,00 | 1,41 | 2,45 | 2,00 | 3,46 | 3,00 | 2,83 | 3,16 | 2,45 | 2,83 | x | 1,41 | 3,16 | 2,83 | 2,83 | 2,83 |
| R | 2,00 | 2,65 | 3,32 | 2,83 | 2,00 | 2,00 | 2,45 | 2,00 | 3,46 | 3,00 | 2,45 | 3,16 | 2,00 | 3,16 | 1,41 | x | 3,16 | 2,83 | 2,83 | 2,83 |
| T | 3,74 | 3,00 | 3,16 | 3,46 | 3,16 | 3,16 | 3,46 | 3,74 | 2,00 | 3,00 | 3,46 | 3,74 | 3,74 | 3,46 | 3,16 | 3,16 | x | 3,74 | 2,45 | 2,83 |
| U | 2,83 | 2,83 | 2,65 | 2,00 | 2,45 | 3,16 | 2,00 | 2,00 | 3,74 | 2,65 | 2,83 | 2,00 | 2,00 | 2,45 | 2,83 | 2,83 | 3,74 | x | 2,83 | 2,83 |
| Y | 3,46 | 3,00 | 3,32 | 3,16 | 2,83 | 3,16 | 3,16 | 2,83 | 3,16 | 3,00 | 3,16 | 3,16 | 2,83 | 3,16 | 2,83 | 2,83 | 2,45 | 2,83 | x | 3,16 |
| Z | 2,83 | 2,24 | 3,00 | 2,83 | 2,00 | 2,45 | 2,45 | 2,83 | 2,45 | 2,24 | 3,16 | 2,83 | 3,16 | 3,16 | 2,83 | 2,83 | 2,83 | 2,83 | 3,16 | x |

Analizując powyższe wyniki podzielono litery na 9 grup:

|  |  |
| --- | --- |
| Lp. grupy | Liczby zawarte w grupie |
| 1 | A |
| 2 | B C D O |
| 3 | E G Z |
| 4 | F P R |
| 5 | I T |
| 6 | J |
| 7 | H N K |
| 8 | L U |
| 9 | Y |



Rys 1. Schemat sieci neuronowej

1. Wybór z zakresu od 0 do 1  
2. Wybór początkowych wartości wag, jako niewielkich liczb losowych z zakresu od -1 do 1  
3. Dla danego wektora uczącego obliczamy odpowiedź sieci. A więc dla pojedynczego neuronu obliczana jest suma ilorazów sygnałów wejściowych i wag - uki. Następnie obliczamy wartość wyjściową neuronu aki wykorzystując tą wartość za pomocą unipolarnej, binarnej funkcji aktywacji.

4. Modyfikacja wag odbywa się według następujących zależności:

η to współczynnik uczenia wybierany z zakresu od 0 do 1

5. W przypadku wykorzystywania współczynnika zapominania modyfikacja wag odbywa się według wzoru:

to współczynnik zapominania wybierany z zakresu 0 do 1 (zwykle 0,1)6. Obliczenie łącznego błędu epoki:

7. Uczenie odbywa się do momentu stabilizacji wag.

1. **Otrzymane wyniki**

Otrzymane przez nas wyniki różnią się mocno dla różnych wartości współczynnika uczenia oraz zapominania. W poniższych tabelach pokazany wpływ modyfikacji tymi współczynnikami na takie czynniki jak: średnia liczba epok potrzebnych do ustabilizowania się wag neuronów, średnia zmiana wartości wag neuronów przy stałej liczbie epok uczenia, jak i średnia zmiana wartości.

Tab 3. Liczba epok potrzeba do ustabilizowania się wartości wag neuronów dla różnych wartości współczynnika zapominania

|  |  |
| --- | --- |
| Wartość współczynnika zapominania | Liczba epok potrzebna do ustabilizowania się wartości wag neuronów |
|
|
| 0,0001 | 641 |
| 0,001 | 480 |
| 0,005 | 157 |
| 0,009 | 94 |
| 0,01 | 80 |

Tab 4. Średnia zmiana wag przy stałej wynoszącej 80 liczbie epok oraz średnia zmiana wag co epokę uczenia dla różnych wartości współczynnika zapominania

|  |  |  |
| --- | --- | --- |
| Wartość współczynnika zapominania | Średnia zmiana wag po 80 epokach uczenia | Średnia zmiana wartości wag co epokę |
|
|
|
| 0,0001 | 268,80 | 6,91 |
| 0,001 | 175,55 | 6,32 |
| 0,005 | 61,27 | 4,47 |
| 0,009 | 35,98 | 3,14 |
| 0,01 | 32,62 | 2,82 |

Tab 5. Średnia zmiana wag przy stałej wynoszącej 80 liczbie epok oraz średnia zmiana wag co epokę uczenia dla różnych wartości współczynnika uczenia

|  |  |  |
| --- | --- | --- |
| Wartość współczynnika uczenia | Średnia zmiana wag po 80 epokach uczenia | Średnia zmiana wartości wag co epokę |
|
|
| 0,01 | 3,76 | 0,09 |
| 0,1 | 35,00 | 1,29 |
| 0,9 | 318,16 | 10,40 |
| 1,1 | 389,44 | 14,80 |
| 1,6 | 565,24 | 22,30 |

Tab 6. Zestawienie otrzymanych wyników dla różnych wartości współczynnika uczenia, jak i współczynnika zapominania w odniesieniu do wartości odległości euklidesowych zawartych w Tab 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Wartość współczynnika zapominania | Wartość współczynnika uczenia | | | | |
| 0,01 | 0,1 | 0,9 | 1,1 | 1,6 |
| Procent poprawności grupowania liter zgodnie z obliczonym rozwiązaniem | | | | |
|
| 0,0001 | 21,00% | 28,32% | 55,55% | 66,66% | 74,34% |
| 0,001 | 24,66% | 36,66% | 51,89% | 70,68% | 77,64% |
| 0,005 | 33,33% | 47,94% | 61,05% | 81,43% | 85,09% |
| 0,009 | 33,33% | 55,26% | 67,02% | 83,27% | 96,20% |
| 0,01 | 47,94% | 61,05% | 77,64% | 92,54% | 100,00% |

1. **Analiza wyników**

Wykres 1. Wykres liczby epok potrzeba do ustabilizowania się wartości wag neuronów dla różnych wartości współczynnika zapominania

Na podstawie powyższego wykresu możemy zauważyć jak duży wpływ ma dobranie odpowiedniej wartości współczynnika zapominania na ilość epok potrzebnych, aby wagi neuronów w naszej sieci się ustabilizowały, a nasza sieć mogła być nauczona i w sposób efektywny realizować swoje zadania. Zwiększenie współczynnika 10 krotnie pozwoliła zredukować o 1/6 ilość epok potrzebnych do nauki przez naszą sieć, jednak to niewielka zmiana współczynnika z 0,001 na 0,005 pozwoliła zmniejszyć ilość epok o nieco ponad 60% co jest największym skokiem wydajnościowym zaobserwowanym podczas testowania. Kolejne zmiany współczynnika uczenia nie skutkowały już takimi zauważalnym efektami, jednak ilość epok sukcesywnie się zmniejszała.

Wykres 2. Wykres średniej zmiany wag przy stałej wynoszącej 80 liczbie epok

Wyniki reprezentowane przez powyższy wykres pokazują nam, że im wyższy współczynnik zapominania tym mniejsza jest średnia zmiana wag neuronów. Im mniejsza różnica tym w mniejszej liczbie epok nasze wagi ulegają stabilizacji, co zresztą jest mocno powiązane z danymi przedstawionymi na wykresie nr 1. Przy wyższym współczynniku zapominania nasze wagi nie osiągają też tak dużych wartości, gdyż o wiele wcześniej się stabilizują.

Wykres 3. Wykres średniej zmiany wag co epokę uczenia dla różnych wartości współczynnika zapominania

Powyższy wykres numer 3 jest mocno powiązany z wykresem numer 2, gdyż oba te wykresy skupiają się na tej samej zależności, jednak w przypadku wykresu numer 3 podano tutaj średnie wartości liczbowe o jakie zmieniają się wagi neuronów pomiędzy epokami uczenia dla poszczególnych wartości współczynnika zapominania i stałej wartości współczynnika uczenia. Widzimy tutaj, iż dla wyższych wartości współczynnika zapominania wahania te są o wiele niższe, w wyniku czego nasze wagi stabilizują się szybciej, a także nie osiągają one tak dużych wartości jak przy mniejszych wartościach tego współczynnika.

Wykres 4. Wykres średniej zmiany wag neuronów po 80 epokach uczenia przy różnych współczynnikach uczenia

Na postawie powyższego wykresu możemy zauważyć jak rośnie średnia zmiana wag neuronów przy stałej liczbie epok uczenia równej 80 oraz stałej wartości współczynnika zapominania równej 0,001 w zależności od wartości współczynnika uczenia. Widzimy tutaj iż wyższa waga współczynnika uczenia powoduje o wiele szybszy wzrost wag, w wyniku czego w momencie ustabilizowania się wagi w sieciach o wyższym współczynniku uczenia cechują się większymi wartościami niż w sieciach o mniejszym współczynniku uczenia.

Wykres 5. Wykres średniej zmiany wartości wag co epokę przy różnych współczynnikach uczenia

Powyższy wykres numer 5 jest mocno powiązany z wykresem numer 4, gdyż oba te wykresy skupiają się na tej samej zależności, jednak w przypadku wykresu numer 5 podano tutaj średnie wartości liczbowe o jakie zmieniają się wagi neuronów pomiędzy epokami uczenia dla poszczególnych wartości współczynnika uczenia i stałej wartości współczynnika zapominania. Widzimy tutaj, iż dla wyższych wartości współczynnika uczenia wahania te są o wiele wyższe, w wyniku czego nasze wagi osiągają o wiele wyższe wartości w momencie kiedy się już one ustabilizują.

Na podstawie danych przedstawionych w tabeli numer 6 możemy stwierdzić, iż po odpowiednim dobraniu zarówno współczynnika zapominania jak i współczynnika uczenia oraz po trwającym wiele epok procesie uczenia sieć neuronowa Hebba jest w stanie podołać zadaniu grupowania przygotowanych przeze mnie liter alfabetu. Wielokrotne testy pokazały, iż dobranie odpowiednich wartości nie jest jednak rzeczą łatwą i wymaga wielu prób, które często są bardzo czasochłonne ze względu na dużą złożoność obliczeniową naszego problemu. Tak jak widzimy w tabeli dla pewnych kombinacji wartości współczynników sieć osiąga słabe rezultaty na poziomie jedynie 21% poprawności kwalifikacji liter, jednak po wielu próbach udało się dobierać te parametry tak, aby sieć była w stanie osiągać coraz to lepsze rezultatu, aż w końcu, aby mogła ona finalnie osiągnąć poprawność na poziomie 100%.

1. **Wnioski**

* Efektywność procesu uczenia dla sieci neuronowej jest silnie powiązana z wartością współczynnika uczenia sieci jak i wartością współczynnika zapominania.

Należy na podstawie wielokrotnych testów dobrać takie wartości współczynników, aby utworzona przez nas sieć była w stanie, grupować alfabet ze skutecznością na akceptowalnym przez nas poziomie.

* Wartość współczynnika zapominania w sposób istotny wpływa na szybkość stabilizowania się wag neuronów. Wyższa wartość współczynnika wymaga mniejszej ilości epok do ustabilizowania się wag.
* Sieć Hebba oparta o proces uczenia z wyłączeniem nauczyciela jest w stanie sama bez naszej pomocy wskazać prawidłowe rozwiązanie postawionego jej przez nas problemu. Wystarczy jedynie ustawić odpowiednio dużą liczbę epok uczenia, tak aby neurony w sieci mogły ustabilizować swoje wagi.
* Ustawienie współczynnika uczenia na większą wartość powoduje, iż sieć osiąga zdecydowanie większe wartości wag dla poszczególnych neuronów, dotyczy to także wartości ustabilizowanych, które to są większe dla wyższych współczynników uczenia.

1. **Kod programu**

**Layer.h:**

#pragma once

#include<vector>

#include"Neuron.h"

using namespace std;

class Layer

{

public:

Layer(int numberOfNeurons, int amountOfDendrites, int amountOfOutputs, double learningCoefficient, double forgettingCoefficient);

vector<Neuron> neurons;

int getNumberOfNeurons() { return \_numberOfNeurons; }

void gatherAnswers();

vector<double> getAnswers() { return \_answers; }

double getAnswer(int index) { return \_answers[index]; }

private:

int \_numberOfNeurons;

vector<double> \_answers;

};

**Layer.cpp:**

#include"Layer.h"

Layer::Layer(int numberOfNeurons, int amountOfDendrites, int amountOfOutputs, double learningCoefficient, double forgettingCoefficient)

{

\_numberOfNeurons = numberOfNeurons;

neurons.resize(numberOfNeurons);

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

neurons[i].Neuron::Neuron(amountOfDendrites, amountOfOutputs, learningCoefficient, forgettingCoefficient);

}

void Layer::gatherAnswers()

{

\_answers.clear();

for (int i = 0; i < \_numberOfNeurons; i++)

\_answers.push\_back(neurons[i].getOutputValue());

}

**Neuron.h:**

#pragma once

#include <iostream>

#include <vector>

using namespace std;

class Neuron

{

public:

void createInputs(int amountOfDendrites, int amountOfSynapses);

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

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

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

int getSynapsesAmount() { return \_synapses.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; }

double getOutputValue() { return \_outputValue; }

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

void sumOfInputs();

void processOutput();

void countNewWeights();

void setErrorSignal(double errorSignal) { \_errorSignal = errorSignal; }

double getErrorSignal() { return \_errorSignal; }

Neuron();

Neuron(int amountOfDendrites, int amountOfOutputs, double learningCoefficient, double forgettingCoefficient);

private:

double \_countFirstWeight();

vector<double> \_dendrites; //inputs

vector<double> \_synapses; //weights

double \_sumOfAllInputs;

double \_outputValue;

double \_learningCoefficient;

double \_forgettingCoefficient;

double \_errorSignal;

};

**Neuron.cpp:**

#include "Neuron.h"

#include <time.h>

#include <math.h>

Neuron::Neuron()

{

\_dendrites.resize(0);

\_synapses.resize(0);

\_sumOfAllInputs = 0.0;

\_outputValue = 0.0;

\_learningCoefficient = 0.0;

\_forgettingCoefficient = 0.0;

}

Neuron::Neuron(int amountOfDendrites, int amountOfOutputs, double learningCoefficient, double forgettingCoefficient)

{

createInputs(amountOfDendrites, amountOfOutputs);

\_learningCoefficient = learningCoefficient;

\_forgettingCoefficient = forgettingCoefficient;

\_sumOfAllInputs = 0.0;

\_outputValue = 0.0;

}

void Neuron::createInputs(int amountOfDendrites, int amountOfOutputs)

{

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

{

\_dendrites.push\_back(0);

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

}

}

void Neuron::sumOfInputs()

{

\_sumOfAllInputs = 0.0;

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

\_sumOfAllInputs += processSingleInput(i);

}

void Neuron::processOutput()

{

if (\_sumOfAllInputs > 0)

\_outputValue = 1.0;

else

\_outputValue = 0.0;

}

void Neuron::countNewWeights()

{

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

\_synapses[i] = \_synapses[i] \* (1.0 - \_forgettingCoefficient) + \_learningCoefficient \* \_dendrites[i] \* \_outputValue;

}

//private methods

double Neuron::\_countFirstWeight()

{

double max = 1.0;

double min = -1.0;

double weight = ((double(rand()) / double(RAND\_MAX)) \* (max - min)) + min;

return weight;

}

**main.cpp:**

#include <iostream>

#include <vector>

#include "Layer.h"

#include <time.h>

#include <fstream>

using namespace std;

void setInputValue(Neuron& neuron, vector< vector<double> > inputData, int numberOfOutputs, int numberOfEntrances, int inputDataRow);

void learn(Layer& layer, vector< vector<double> > inputData);

void test(Layer& layer, vector< vector<double> > inputData);

void readLearningData(vector< vector<double> > &learningInputData, int numberOfEntrances, int amountOfOutputs);

void readTestingData(vector< vector<double> > &testingInputData, int numberOfEntrances, int amountOfOutputs);

fstream OUTPUT\_LEARNING\_FILE;

fstream OUTPUT\_TESTING\_FILE;

fstream LEARNING\_DATA;

fstream TESTING\_DATA;

int main()

{

srand(time(NULL));

vector< vector<double> > learningInputData;

vector< vector<double> > testingInputData;

int numberOfNeurons = 9;

int numberOfEntrances = 21;

int numberOfOutputs = 1;

double learningCoefficient = 1.6;

double forgettingCoefficient = 0.001;

Layer hebbianNetwork(numberOfNeurons, numberOfEntrances, numberOfOutputs, learningCoefficient, forgettingCoefficient);

readLearningData(learningInputData, numberOfEntrances, numberOfOutputs);

readTestingData(testingInputData, numberOfEntrances, numberOfOutputs);

do

{

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

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

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

int choice;

cin >> choice;

switch (choice)

{

case 1:

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

for (int epochNumber = 1, i = 0; i < 80; i++, epochNumber++)

{

learn(hebbianNetwork, learningInputData);

OUTPUT\_LEARNING\_FILE << hebbianNetwork.neurons[2].getSynapse(2) << endl;

//OUTPUT\_LEARNING\_FILE << "EPOCH NUMBER: " << epochNumber << endl;

}

break;

case 2:

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

test(hebbianNetwork, testingInputData);

break;

case 3:

OUTPUT\_LEARNING\_FILE.close();

OUTPUT\_TESTING\_FILE.close();

return 0;

}

} while (true);

return 0;

}

void setInputValue(Neuron& neuron, vector< vector<double> > inputData, int numberOfOutputs, int numberOfEntrances, int row)

{

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

neuron.setInput(i, inputData[row][i]);

}

void learn(Layer& layer, vector< vector<double> > inputData)

{

for (int inputDataRow = 0; inputDataRow < inputData.size(); inputDataRow++)

{

for (int i = 0; i < layer.getNumberOfNeurons(); i++)

{

setInputValue(layer.neurons[i], inputData, layer.getNumberOfNeurons(), layer.neurons[i].getDendritesAmount(), inputDataRow);

layer.neurons[i].sumOfInputs();

layer.neurons[i].processOutput();

layer.neurons[i].countNewWeights();

}

}

}

void test(Layer& layer, vector< vector<double> > inputData)

{

for (int inputDataRow = 0; inputDataRow < inputData.size(); inputDataRow++)

{

for (int i = 0; i < layer.getNumberOfNeurons(); i++)

{

setInputValue(layer.neurons[i], inputData, layer.getNumberOfNeurons(), layer.neurons[i].getDendritesAmount(), inputDataRow);

layer.neurons[i].sumOfInputs();

layer.neurons[i].processOutput();

}

layer.gatherAnswers();

for (int i = 0; i < layer.getNumberOfNeurons(); i++)

OUTPUT\_TESTING\_FILE << layer.neurons[i].getOutputValue() << " ";

OUTPUT\_TESTING\_FILE << endl;

}

}

void readLearningData(vector< vector<double> > &learningInputData, int numberOfEntrances, int amountOfOutputs)

{

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

vector<double> row;

do

{

row.clear();

for (int i = 0, inputTmp = 0; i < numberOfEntrances; i++)

{

LEARNING\_DATA >> inputTmp;

row.push\_back(inputTmp);

}

learningInputData.push\_back(row);

} while (!LEARNING\_DATA.eof());

LEARNING\_DATA.close();

}

void readTestingData(vector< vector<double> > &testingInputData, int numberOfEntrances, int amountOfOutputs)

{

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

vector<double> row;

while (!TESTING\_DATA.eof())

{

row.clear();

for (int i = 0, inputTmp; i < numberOfEntrances; i++)

{

TESTING\_DATA >> inputTmp;

row.push\_back(inputTmp);

}

testingInputData.push\_back(row);

}

TESTING\_DATA.close();

}