Temat: ***Budowa​ ​ i​ ​ działanie​ ​ sieci​ ​ jednowarstwowej – rozpoznawanie wielkości liter***

Data oddania: 24.10.2017

Prowadzący: dr inż. Dorota Wilk - Kołodziejczyk

Przedmiot: Podstawy sztucznej inteligencji

Autor: Przemysław Adamus 286084

Kierunek: Inżynieria Obliczeniowa sem.V

Spis treści:

1. Cel projektu;
2. Wstęp i opis projektu;
3. Budowa wykorzystanego perceptronu;
4. Zestaw uczący;
5. Syntetyczny​ ​opis​ ​budowy​​ pierwszego ​wykorzystanego​ ​algorytmu​ ​uczenia w projekcie;
6. Syntetyczny​ ​opis​ ​budowy​​ drugiego ​wykorzystanego​ ​algorytmu​ ​uczenia w projekcie;
7. Wyniki – tabele;
8. Wnioski i analiza wyników;
9. Źródła;
10. Załączniki.
11. **CEL:**

Celem​ ​ ćwiczenia​ ​ jest​ ​ poznanie​ ​ budowy​ ​ i​ ​ działania​ ​ jednowarstwowych​ ​ sieci​ ​ neuronowych oraz​ ​ uczenie​ ​ rozpoznawania​ ​ wielkości​ ​ liter.

1. **WSTĘP I OPIS PROJEKTU:**

Do rozwiązania zadanego problemu wykorzystałem wiedzę i projekt c++ z wcześniejszego scenariusza.

Zadaniem mojego programu było rozpoznanie wielkości podanej litery. Najpierw przygotowałem zestaw danych uczących. Umieściłem je w 2 plikach .csv. Plik dane.csv zawiera zbiór 20 liter ( 10 małych i 10 dużych ) zapisanych za pomocą 1 i 0. Natomiast plik score.csv zawiera oczekiwane odpowiedzi neuronu dla każdej litery. Do programu dane wczytuje za pomocą funkcji void loadData(bool\*\*) i void loadScore(bool\*). Zdecydowałem się na matrycę wielkości 5x7. 7 pikseli wysokości i 5 pikseli szerokości. Dane uczące można znaleźć w punkcie 3 tego sprawozdania.

W sprawozdaniu zostały umieszczone rozwiązania, kody i wyniki zadania z środowiska C++. Stworzyłem klasę neuron wraz z całą gamą metod do działania na obiektach tej klasy. Znajduje się tam również funkcja do nauki neuronu. Klasę umieściłem w pliku Neurone.h – załącznik 1, a implementacje metod w pliku Neuron.cpp – załącznik2.

W funkcji main() możemy znaleźć proste menu do obsługi i działaniu na naszym neuronie - załącznik 3. Po każdym działaniu możemy zaobserwować czy zwracana wartość i suma membranowa różni się od poprzedniej.

Po zakończonej implementacji przetestowałem mój neuron.

1. **BUDOWA WYKORZYSTANEGO PERCEPTRONU:**

Perceptronem nazywamy prosty element obliczeniowy, który sumuje ważone sygnały wejściowe i porównuję tę sumę z progiem aktywacji - w zależności od wyniku perceptron może być albo wzbudzony (wynik 1), albo nie (wynik 0).

Suma membranowa jest to suma wszystkich iloczynów wejść i ich wag. Jest ona przekazywana dalej do funkcji aktywacji.

Funkcja aktywacji w sztucznej inteligencji określa funkcje, według której obliczana jest wartość wyjścia neuronów sieci neuronowej. Do najczęściej używanych funkcji aktywacji należą: funkcja liniowa, funkcja progowa unipolarna, funkcja progowa bipolarna, funkcja sigmoidalna unipolarna, funkcja sigmoidalna bipolarna.

Rezultat wiadomość zwrotna od neuronu. Może być przekazana dalej, lub być ostatecznym wynikiem. Rezultat jest wartością logiczną więc powinien zawierać tylko informacjię *true* lub *false.*

1. **ZESTAW UCZĄCY:**

Dane.csv:

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;1;1;1;0;

1;0;0;0;1;

1;0;0;0;1;

1;1;1;1;0;

1;0;0;0;1;

1;0;0;0;1;

1;1;1;1;0;

0;1;1;1;0;

1;0;0;0;1;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;1;

0;1;1;1;0;

1;0;0;0;1;

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;1;1;1;1;

0;0;0;0;1;

0;0;0;1;0;

0;0;1;0;0;

0;1;0;0;0;

1;0;0;0;0;

1;1;1;1;1;

1;0;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;1;1;1;

1;1;1;1;1;

1;0;1;0;1;

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;

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;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;0;1;0;1;

1;0;1;0;1;

0;1;0;1;0;

1;1;1;1;1;

1;0;0;0;0;

1;0;0;0;0;

1;1;1;1;0;

1;0;0;0;0;

1;0;0;0;0;

1;1;1;1;1;

0;0;0;0;0;

0;0;0;0;0;

0;0;0;0;0;

0;1;1;0;0;

1;0;0;0;0;

1;0;0;0;0;

0;1;1;0;0;

0;0;0;0;0;

0;0;0;0;0;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;0;

1;1;0;0;0;

0;0;0;0;0;

0;0;0;0;0;

0;0;0;0;0;

0;1;1;0;0;

1;0;0;1;0;

1;0;0;1;0;

0;1;1;0;0;

0;0;0;0;0;

0;0;0;0;0;

0;0;0;0;0;

0;1;1;0;0;

1;0;0;1;0;

1;0;0;1;0;

0;1;1;1;1;

0;0;0;0;0;

0;0;0;0;0;

0;1;0;0;0;

1;1;1;0;0;

0;1;0;0;0;

0;1;0;0;0;

0;1;0;0;0;

0;0;0;0;0;

0;0;0;0;0;

0;0;0;0;0;

1;0;0;0;1;

1;0;0;0;1;

0;1;0;1;0;

0;0;1;0;0;

0;0;0;0;0;

0;0;0;0;0;

0;0;0;0;0;

1;0;0;1;0;

1;0;0;1;0;

1;0;0;1;0;

0;1;1;1;0;

0;0;0;0;0;

0;0;0;0;0;

1;0;0;0;0;

1;0;0;0;0;

1;1;1;0;0;

1;0;0;1;0;

0;1;1;0;0;

0;0;0;0;0;

0;0;0;0;0;

0;0;0;1;0;

0;0;0;1;0;

0;1;1;1;0;

1;0;0;1;0;

0;1;1;1;0;

0;0;0;0;0;

0;0;0;0;0;

0;0;0;0;0;

1;1;1;0;0;

1;0;0;1;0;

1;0;0;1;0;

1;1;1;1;0

Plik score.csv: 1;1;1;1;1;1;1;1;1;1;0;0;0;0;0;0;0;0;0;0

1. **PIERWSZY ALGORYTM UCZENIA W PROJEKCIE C++ - Widrowa – Hoffa:**

Algorytm uczenia perceptronu jest to automatyczny doboru wag na podstawie napływających przykładów. Ogólny przepis (algorytm) uczenia neuronu wynika z zadania minimalizacji błędu uczenia. Pierwszy algorytm to tak zwany algorytmy uczenia z „nauczycielem”. Uczenie z nauczycielem polega na tym, że sieci podaje się przykłady poprawnego działania, które powinna ona potem naśladować w swoim bieżącym działaniu (w czasie egzaminu). Przykład należy rozumieć w ten sposób, że nauczyciel podaje konkretne sygnały wejściowe i wyjściowe, pokazując, jaka jest wymagana odpowiedź sieci dla pewnej konfiguracji danych wejściowych. Mamy do czynienia z parąwartości - przykładowym sygnałem wejściowym i pożądanym (oczekiwanym) wyjściem, czyli wymaganą odpowiedzią sieci na ten sygnał wejściowy. Zbiór przykładów zgromadzonych w celu ich wykorzystaniu w procesie uczenia sieci nazywa się zwykle ciągiem uczącym. Zatem w typowym procesie uczenia sieć otrzymuje od nauczyciela ciąg uczący i na jego podstawie uczy się prawidłowego działania, stosując jedną z wielu znanych dziś strategii uczenia.

W moim programie C++ za uczenie neuronu odpowiada metoda klasy Neuron typu double o nazwie „teachNeuron”. Przyjmuje ona tablice wejść i oczekiwaną odpowiedź, a zwraca średni błąd kwadratowy jednej iteracji. Funkcję uczenia możemy wywoływać wybrną przez nas ilość razy, albo do momentu gdy średni błąd kwadratowy nie wyniesie 0. W moim projekcie C++ możemy pozwolić sobie na zainicjowanie wagi losowo. Najlepiej z przedziału od -0.5 do 0.5, a współczynnik nauczania dobieramy dowolny. Ja przeprowadziłem testy na współczynniku równym: 0,1 ; 0,05 ; 0,01.

* Dla każdego przykładu uczącego obliczamy odpowiedź perceptronu.
* Jeśli odpowiedź perceptronu jest nieprawidłowa, to modyfikujemy wagi według reguły uczenia Widrowa – Hoffa:

w1 += n \* (d-y) \* x1   
w2 += n \* (d-y) \* x2   
…

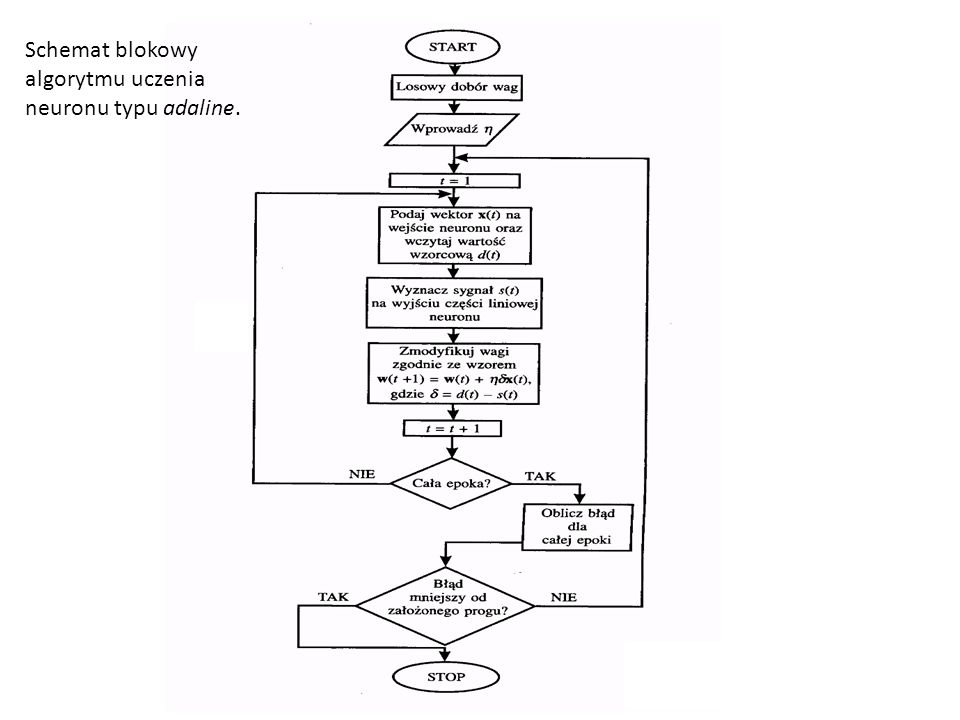
gdzie:

n - współczynnikiem uczenia (n > 0);

d - oczekiwana odpowiedź;

y - odpowiedź neuronu.

Po wyczerpaniu przykładów zaczynamy proces uczenia od początku, do momentu kiedy liczba cykli uczenia (liczba epok) przekracza zdefiniowaną z góry liczbę epok lub średniokwadratowy błąd sieci maleje poniżej progu błędu ε.



**D**

1. **DRUGI ALGORYTM UCZENIA W PROJEKCIE C++ - reguła Hebba - bez nauczyciela :**

Reguła Hebba jest zarazem najprostszą i najwcześniej odkrytą regułą uczenia. Jest ona przeniesieniem stwierdzenia z zakresu neurobiologii, które mówi: „Jeżeli akson neuronu A bierze systematycznie udział w pobudzaniu neuronu B powodując jego aktywację, to wywołuje to zmianę metaboliczną w jednym lub obu neuronach, prowadzącą do wzrostu skuteczności pobudzania neuronu B przez neuron A.” Najprostsza wersja tej reguły odnosi się do uczenia bez nauczyciela i przyjmuje, że sygnał wyjściowy neuronu jest sygnałem uczącym, czyli przyrost wektora wag wynosi:

**∆w =η ⋅ y ⋅ x**

przy założeniu, że η > 0 jest pewnym współczynnikiem uczenia,

yi - wyjściem i-tego neuronu,

xj - j-tym wejściem neuronów.

Nauka polega po prostu na modyfikacji wag pomiędzy każdą parą pobudzonych neuronów. Wynikiem działania tej reguły jest to, że dodatnia wartość składnika korelacyjnego i yj xi powoduje wzrost wagi wij , co w konsekwencji daje silniejszą odpowiedź neuronu przy kolejnej próbie pobudzenia tym samym wzorcem wejściowym. Wzorce często powtarzające się na wejściu sieci dają więc najsilniejszą odpowiedź na jej wyjściu. Reguła ta jest więc często wykorzystywana w sieciach autoasocjacyjnych. Stosowanie reguły Hebba w czystej postaci powoduje nieskończony wzrost wag, więc w praktycznych realizacjach stosuje się często pewien współczynnik normalizujący przeciwdziałający temu nieograniczonemu wzrostowi. Wagi są zazwyczaj aktualizowane po każdym wzorcu uczącym (on-line training). Istnieje wiele rozszerzeń i modyfikacji reguły Hebba, która w efekcie może być stosowana do uczenia z nauczycielem (supervised training) i do uczenia bez nauczyciela (unsupervised training). Dobrze się sprawuje zarówno dla wzorców binarnych jak i bipolranych.

1. **WYNIKI:**

|  |  |  |  |
| --- | --- | --- | --- |
| REGUŁA WIDORWA - HOFFA | | | |
| ILOŚĆ EPOK UCZENIA dla progu aktywacji 0 i wag losowych od -0.5 do 0.5 | | | |
| LP. / STAŁA UCZENIA | **0.1** | **0.05** | **0.01** |
| 1. | **3** | **7** | **15** |
| 2. | **3** | **7** | **11** |
| 3. | **3** | **7** | **14** |
| 4. | **3** | **8** | **15** |
| 5. | **3** | **7** | **17** |
| 6. | **2** | **7** | **15** |
| 7. | **3** | **7** | **16** |
| 8. | **3** | **6** | **14** |
| 9. | **3** | **7** | **13** |
| 10. | **3** | **7** | **16** |
| ŚREDNIA: | **2.9** | **7** | **14.6** |

|  |  |  |  |
| --- | --- | --- | --- |
| REGUŁA WIDORWA – HOFFA | | | |
| ILOŚĆ EPOK UCZENIA dla progu aktywacji 4.5 i wag losowych od 0 do 0.5 | | | |
| LP. / STAŁA UCZENIA | **0.1** | **0.05** | **0.01** |
| 1. | **2** | **3** | **10** |
| 2. | **2** | **3** | **15** |
| 3. | **3** | **3** | **14** |
| 4. | **2** | **4** | **15** |
| 5. | **2** | **4** | **11** |
| 6. | **2** | **3** | **12** |
| 7. | **2** | **3** | **14** |
| 8. | **2** | **3** | **13** |
| 9. | **2** | **3** | **13** |
| 10. | **2** | **3** | **10** |
| ŚREDNIA: | **2.1** | **3.2** | **12.7** |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| REGUŁA WIDORWA – HOFFA | | | | | | | | | |
| BŁĄD KWADRATOWY dla progu aktywacji 4.5 i wag losowych od 0 do 0.5 | | | | | | | | | |
| LP | **0.1** | | | **0.05** | | | **0.01** | | |
| 1. | 6 | 8 | 8 | 8 | 8 | 10 | 10 | 9 | 7 |
| 2. | 0 | 0 | 2 | 4 | 3 | 6 | 9 | 8 | 7 |
| 3. |  |  | 0 | 0 | 0 | 3 | 8 | 7 | 6 |
| 4. |  |  |  |  |  | 0 | 7 | 7 | 5 |
| 5. |  |  |  |  |  |  | 6 | 6 | 4 |
| 6. |  |  |  |  |  |  | 5 | 5 | 3 |
| 7. |  |  |  |  |  |  | 4 | 4 | 3 |
| 8. |  |  |  |  |  |  | 3 | 3 | 2 |
| 9. |  |  |  |  |  |  | 2 | 3 | 1 |
| 10. |  |  |  |  |  |  | 1 | 2 | 1 |
| 11. |  |  |  |  |  |  |  | 2 | 1 |
| 12. |  |  |  |  |  |  |  | 1 | 1 |
| 13. |  |  |  |  |  |  |  | 1 | 1 |
| 14. |  |  |  |  |  |  |  | 1 | 0 |
| 15. |  |  |  |  |  |  |  | 0 |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| REGUŁA WIDORWA – HOFFA | | | | | | | | | |
| BŁĄD KWADRATOWY dla progu aktywacji 0 i wag losowych od -0.5 do 0.5 | | | | | | | | | |
| LP | **0.1** | | | **0.05** | | | **0.01** | | |
| 1. | 7 | 9 | 8 | 8 | 9 | 10 | 10 | 6 | 7 |
| 2. | 3 | 4 | 2 | 6 | 8 | 6 | 9 | 5 | 7 |
| 3. | 0 | 0 | 0 | 5 | 6 | 5 | 8 | 4 | 6 |
| 4. |  |  |  | 4 | 5 | 4 | 7 | 3 | 5 |
| 5. |  |  |  | 2 | 3 | 3 | 6 | 3 | 4 |
| 6. |  |  |  | 1 | 1 | 1 | 4 | 2 | 4 |
| 7. |  |  |  | 0 | 0 | 0 | 4 | 2 | 3 |
| 8. |  |  |  |  |  |  | 3 | 1 | 2 |
| 9. |  |  |  |  |  |  | 2 | 1 | 2 |
| 10. |  |  |  |  |  |  | 2 | 1 | 1 |
| 11. |  |  |  |  |  |  | 2 | 0 | 1 |
| 12. |  |  |  |  |  |  | 1 |  | 1 |
| 13. |  |  |  |  |  |  | 1 |  | 1 |
| 14. |  |  |  |  |  |  | 1 |  | 0 |
| 15. |  |  |  |  |  |  | 0 |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| REGUŁA WIDORWA – HEBBA – dla zestawu uczącego 20 liter | | | |
| ILOŚĆ EPOK UCZENIA dla progu aktywacji 4.5 i wag losowych od 0 do 0.5 | | | |
| LP. / STAŁA UCZENIA | **0.1** | **0.05** | **0.01** |
| 1. | **3** | **7** | **17** |
| 2. | **3** | **7** | **20** |
| 3. | **2** | **7** | **19** |
| 4. | **3** | **8** | **15** |
| 5. | **3** | **8** | **17** |
| 6. | **3** | **8** | **15** |
| 7. | **4** | **7** | **16** |
| 8. | **3** | **6** | **18** |
| 9. | **3** | **9** | **17** |
| 10. | **2** | **7** | **16** |
| ŚREDNIA: | **2.9** | **7.4** | **17.0** |

|  |  |  |  |
| --- | --- | --- | --- |
| REGUŁA WIDORWA – HEBBA – dla zestawu uczącego 10 liter | | | |
| ILOŚĆ EPOK UCZENIA dla progu aktywacji 4.5 i wag losowych od 0 do 0.5 | | | |
| LP. / STAŁA UCZENIA | **0.1** | **0.05** | **0.01** |
| 1. | **6** | **21** | **89** |
| 2. | **6** | **22** | **82** |
| 3. | **6** | **28** | **81** |
| 4. | **6** | **23** | **83** |
| 5. | **7** | **21** | **87** |
| 6. | **6** | **24** | **84** |
| 7. | **6** | **20** | **90** |
| 8. | **8** | **19** | **87** |
| 9. | **8** | **23** | **84** |
| 10. | **7** | **24** | **86** |
| ŚREDNIA: | **6.6** | **22.5** | **85.3** |

1. **WNIOSKI I ANALIZA WYNIKÓW**:

W podpunktach wyżej nie znalazł się opis wszystkich metod i możliwości projektu C++. Osobiście uważam, że najlepszym komentarzem do niego jest jego kod. Starałem się aby sam się komentował.

Wnioski po zestawieniu wyników nasuwają się same. Po pierwsze ilość epok uczących rośnie wraz z malejącym współczynnikiem uczenia. Co widać na 1,2 i 5,6 tabeli. Zarówno w pierwszym jak i w drugim wykorzystanym algorytmie proces nauki się kończy i przebiega pomyślnie. Odrobinę szybszy jest algorytm korzystający z reguły Widorwa – Hoffa. Uważam, że spełnił on lepiej swoje zadanie o czym świadczy , zamieszczony w tabeli 3 i 4, malejący błąd kwadratowy. W tabeli 3 i 4 możemy również zaobserwować różnicę przy zmianie wartości początkowych wag i progu aktywacji. Przy obu tych parach parametrów algorytm uczył się poprawnie. W tabeli 5 i 6 możemy zestawić z sobą wyniki uzyskane po nauce dla różnej ilości danych wejściowych. I tak zmniejszenie ilości danych uczących zwiększa znacząco ilość epok nauki, a czasem uniemożliwia jej poprawność.

Reasumując, mnie bardziej przekonał algorytm uczenia z regułą Widorwa – Hoffa dlatego znalazł się on w ostatecznej wersji programu. Podczas pracy na algorytmie Hebba napotkałem problemy z rozpoznawaniem wielkości litery „I”. Była ona czasem uznawana za małą literę. Jednak niezmierzonym plusem tego algorytmu jest to, że jest on algorytmem „bez nauczyciela”. Dzięki temu nie musimy podawać rozwiązań do naszych danych wejściowych.

1. **ŹRÓDŁA:**

* <https://edux.pjwstk.edu.pl/mat/273/lec/wyklad3/w3.htm>;
* <https://pl.wikipedia.org/wiki/Perceptron>
* <https://forum.pasja-informatyki.pl/73877/sztuczne-sieci-neuronowe>
* <http://www.if.uz.zgora.pl/~mdudek/siecineuronowe.pdf>
* <https://www.youtube.com/watch?v=Wa_9S20SkKw&t=995s>
* http://home.agh.edu.pl/~horzyk/pracedoktorskie/adrianhorzykpracadoktorska.pdf
* Materiały udostępnione przez dr inż. Dorota Wilk - Kołodziejczyk

1. **ZAŁĄCZNIKI:**
   1. Neuron.h – klasa neuron, projekt C++;
   2. Neuron.cpp – implementacja metod, projekt C++;
   3. Main.cpp – funkcja main, projekt C++;

Zał 2) – Neuron.cpp

#include "Neuron.h"

#define constantLearning 0.01 // stała uczęnia

#define amountOfData 20 // ilość danych wejściowych

Neuron::Neuron()

{

size = 20;

input = new bool[size];

weight = new double[size];

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

{

input[i] = 0;

weight[i] = 0.5;

}

}

Neuron::Neuron(int s, bool\* tab)

{

size = s;

input = new bool[size];

weight = new double[size];

srand(time(NULL));

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

{

input[i] = tab[i];

//weight[i] = (rand() % 100) / 100.0 - 0.5;

weight[i] = (rand() % 100) / 200.0;

}

}

Neuron::~Neuron()

{

delete[] input;

delete[] weight;

}

void Neuron::showNeuron()

{

std::cout << "\_\_\_\_\_MÓJ NEURON\_\_\_\_\_" << std::endl;

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

{

std::cout << i << " INPUT: " << input[i] << " WEIGHT: " << weight[i] << std::endl;

}

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "| SUMA: | " << getMembranePotential() << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "| WYNIK: | " << functionUse() << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

}

double Neuron::teachNeuron(bool \*\*tab,bool \*score)

{

double localError = 0.0;

double globalError = 0.0;

int n = Neuron::getSize();

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

{

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

{

input[i] = tab[j][i];

}

localError = score[j] - functionUse();

globalError += localError\*localError;

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

{

setWeight(i, getWeight(i)+constantLearning\*localError\*input[i]);

}

globalError += localError\*localError;

}

globalError = globalError\*0.5;

return globalError;

}

double Neuron::getMembranePotential() //OBLICZANIE SUMY MEMBRANOWEJ

{

if (size == 0) //JEŚLI WEJŚCIA NIE ISTNIEJĄ

return -1;

else

{

double sum = 0;

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

{

sum += input[i]\*weight[i] ;

}

return sum;

}

}

long double Neuron::power(int p) //POTĘGOWANIE NIEZBĘDNE DO WLICZNIA FUNKCJI AKTYWACJI UNIPOLARNEJ

{

const double w = getMembranePotential();

double n = 1;

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

{

n \*= p;

}

long double x = getMembranePotential() / n; //BO POTENGUJEMY PRZEZ -5 WIĘDZ DZIELIMY

return x;

}

//double Neuron::function\_use() //FUNKCJA AKTYWACJI

//{

// if (size == 0) //JEŚLI WEJŚCIA NIE ISTNIEJĄ

// return -1;

//

// else

// {

// float e = 2.71; //WARTOŚĆ LOGARYTMU

// float f = (1 / (1 + (power(e)))); //AKTYWACJA UNIPOLARNA

// return f;

// }

//}

double Neuron::functionUse() //FUNKCJA AKTYWACJI

{

long double x = getMembranePotential();

if (x >= 4.5) return 1.0;

//if (x >= 0.0) return 1.0;

return 0.0;

}

void Neuron::setNeuron(bool\* tab)

{

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

{

input[i] = tab[i];

}

}

Zał 1) – Neuron.h

#pragma once

#include <iostream>

#include <vector>

#include <conio.h>

#include <windows.h>

#include <cstdlib>

#include <fstream>

#include <string>

#include <ctime>

class Neuron

{

private:

int size; // ILOŚ WEJŚĆ

bool \*input; //WEJŚCIA

double \*weight; //WAGI

public:

Neuron(); // Konstruktor domyślny

Neuron(int, bool\*); //Konstruktor z parametrem

~Neuron(); // Destruktor

void showNeuron(); //Wyświetla neuron

//SET

void setSize(int s) { size = s; }

void setInput(int i, bool value) { input[i] = value; }

void setWeight(int i, double value) { weight[i] = value; }

//GET

int getSize() { return size; }

bool getInput(int i) { return input[i]; }

double getWeight(int i) { return weight[i]; }

void setNeuron(bool\*); //Ustawienie wejść neuronu

double teachNeuron(bool \*\*, bool \*); // Uczy neuron

double getMembranePotential(); // Oblicznie sumy membranowej

long double power(int p); //POTĘGOWANIE NIEZBĘDNE DO WLICZNIA FUNKCJI AKTYWACJI UNIPOLARNEJ

double functionUse(); //FUNKCJA AKTYWACJI

};

Zał 3) – Main.cpp

#include "Neuron.h"

#define n 35 // ilość pikseli matrycy - u nas 5x5

#define amountOfData 20 // ilość danych wejściowych

using namespace std;

void LoadData(bool\*\* tab)

{

double a = 0.0;

fstream file;

file.open("dane.csv", ios::in);

int no\_line = 1;

string line;

int i = 0;

int j = 0;

if (file.good() == false)

{

cout << "Error 0" << endl;

exit(0);

}

while (getline(file, line, ';'))

{

a = atoi(line.c\_str());

no\_line++;

tab[i][j] = a;

j++;

if (j % n == 0 && j != 0)

{

i++;

j = 0;

}

}

file.close();

}

void LoadScore(bool\* tab)

{

double a = 0.0;

fstream file;

file.open("score.csv", ios::in);

int no\_line = 1;

string line;

int i = 0;

if (file.good() == false)

{

cout << "Error 0" << endl;

exit(0);

}

while (getline(file, line, ';'))

{

a = atoi(line.c\_str());

no\_line++;

tab[i] = a;

i++;

}

file.close();

}

int main()

{

setlocale(LC\_ALL, ""); // Polskie litery w konsloi

//DANE POMOCNICZE

int index = 0;

double value = 0.0;

double globalError = 0.0;

int counter = 0;

// TABLICE Z DANYMI WEJŚCIOWYMI

bool\*\* data = new bool\*[amountOfData];

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

{

data[i] = new bool[n];

}

bool score[n] = { 0 };

bool input[n] = { 0 };

LoadData(data);

LoadScore(score);

Neuron neuron(n, input); //tworzymy neuron z 25 wejsciami. Wagi ustawiamy losowo

while (true)

{

system("cls");

neuron.showNeuron();

cout << endl << endl;

cout << " ---MENU--- " << endl << endl;

cout << "1) Zmień wybraną wagę " << endl;

cout << "2) Zmień wybrane wejście " << endl;

cout << "3) Ucz neuron " << endl;

cout << "4) Testuj neuron " << endl;

cout << "5) Wypisz zestaw uczący " << endl;

cout << "0) Exit " << endl;

cout << "WYBRAŁEŚ: ";

char z;

cin >> z;

switch (z)

{

case '1':

{

cout << "Podaj indeks wagi: ";

cin >> index;

cout << "Podaj wartosć nowej wagi: ";

cin >> value;

neuron.setWeight(index, value);

}

break;

case '2':

{

cout << "Podaj indeks wejścia: ";

cin >> index;

cout << "Na jaką wartość logiczną chcesz zmienić ? " << endl << "true -> 1" << endl << "false -> 0" << endl << "WYBRAŁEŚ: ";

cin >> value;

if (value == 1)

{

neuron.setInput(index, true);

}

else

{

neuron.setInput(index, false);

}

}

break;

case '0':

{

exit(0);

}

break;

case '4':

{

cout << endl << "TEST: " << endl;

bool A[n] = { 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 };

bool I[n] = { 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 };

bool a[n] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,1,0,1,0,0,1,0,0,1,1,1,1 };

bool b[n] = { 0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0,1,0,0,1,0,0,1,1,0,0 };

neuron.setNeuron(A);

}

break;

case '5':

{

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

{

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

{

cout << data[i][j];

if (j % 5 == 4 && j != 0)cout << endl;

}

}

cout << endl << "Wciśnij ENTER";

\_getch();

}

break;

case '3':

{

counter++;

globalError=neuron.teachNeuron(data,score);

cout << counter <<" EPOKA UCZENIA" << endl << "Średni błąd kwadratowy wynosi: " << globalError << endl;

if (globalError != 0)

{

cout << endl << "Można dalej uczyć - wciśnij ENTER" << endl;

}

else

{

cout << "Nie wskazano dalszego uczenia - wciśnij ENTER" << endl;

}

\_getch();

}

break;

}

}

//DELOKACJA PAMIĘCI

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

{

delete[] data[i];

}

delete[] data;

data = NULL;

neuron.~Neuron();

//system("pause");

return 0;

}