Sprawozdanie

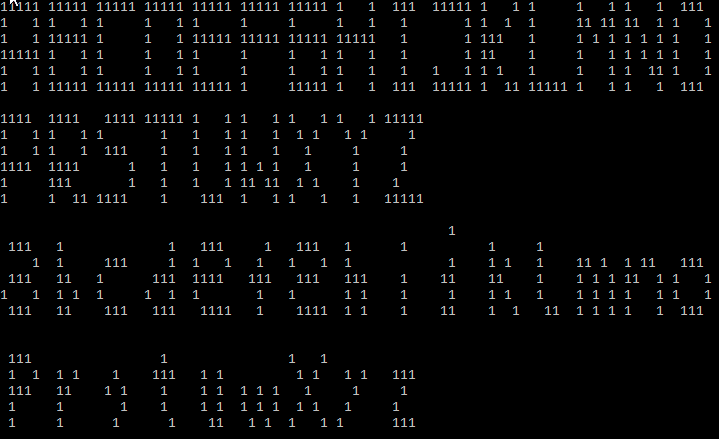
Zadanie z laboratorium numer 2 polegało na nauczeniu sieci perceptronowej rozpoznawania literek złożonych z cyfr 1 i 0 odpowiadających odpowiednio zamalowanemu lub też nie pikselowi w bitmapie. Zgodnie ze wcześniejszym ustaleniem z prowadzącym w swojej implementacji użyłem wszystkich liter dużych i małych alfabetu angielskiego za wyjątkiem v oraz V. Każda litera składała się z 5x6 zer oraz jedynek. Łącznie wyszło 48 literek.

Sieć składa się z 48 perceptronów, a każdy z nich ma w sobie wektor 31 wejść oraz 31 wag (został uwzględniony bias). Funkcja aktywacji jaka została użyta to obliczania poprawności rozpoznawania literek to sigmoidalna funkcja unipolarna o wzorze:2018-01-21 20_02_58-Funkcja aktywacji – Wikipedia, wolna encyklopedia.png

gdzie za β przyjąłem 1 natomiast x to suma iloczynu wag i wejść.

Wybrałem tą funkcje ponieważ wynik podawany jest w przedziale od (0,1).

Zasada działania programu opiera się na wyuczeniu do pewnego stopnia przez jeden perceptron jednej litery(z góry określonej). Za nauczenie przyjąłem próg funkcji aktywacji 0,9 co daje w przybliżeniu wynik na poziomie 90%. Program uczy się tak długo, aż każdy perceptron nie osiągnie wyniku przynajmniej 90%. Następnie testowanych jest kilka liter o zaburzonych pikselach (zaburzałem ok. 3 piksele) i wypisywana jest funkcja aktywacji mówiąca nam na ile procent dana litera jest jakąś literą. Większość wyników to zawsze jest liczba bardzo bliska 0 co daje dobre wyniki ogólne, bo ciężko aby program rozpoznawał jakaś literę jako np. siedem innych. Pierwsza część wypisu jest to test, gdzie dla czytelności, jeżeli aktywacja wyjdzie powyżej 0,9 to wpisuje 1. Jeżeli wszystkie takie testy przejdą wtedy program zatrzymuje swoje wykonywanie.



Literki jakie komputer miał się nauczyć

Tester::Tester() {

tab[0] = "111111000110001111111000110001";

tab[1] = "111111000111111100011000111111";

tab[2] = "111111000010000100001000011111";

tab[3] = "111111000110001100011000111111";

tab[4] = "111111000011111100001000011111";

tab[5] = "111111000011111100001000010000";

tab[6] = "111111000011111100011000111111";

tab[7] = "100011000111111100011000110001";

tab[8] = "011100010000100001000010001110";

tab[9] = "111110000100001000011000111111";

tab[10] = "100011001011100110001010010011";

tab[11] = "100001000010000100001000011111";

tab[12] = "100011101110101100011000110001";

tab[13] = "100011100110101101011001110001";

tab[14] = "011101000110001100011000101110";

tab[15] = "111101000110001111101000010000";

tab[16] = "111101000110001111101110010011";

tab[17] = "011111000001110000010000111110";

tab[18] = "111110010000100001000010000100";

tab[19] = "100011000110001100011000101110";

tab[20] = "100011000110001101011101110001";

tab[21] = "100010101000100001000101010001";

tab[22] = "100010101000100001000010000100";

tab[23] = "111110001000100001000100011111";

tab[24] = "000000111000001011101000101110";

tab[25] = "000000100001000011000101001100";

tab[26] = "000000000001110100001000001110";

tab[27] = "000000001000010011101001001110";

tab[28] = "000000111010001111101000001111";

tab[29] = "000000001000100011100010000100";

tab[30] = "000000111010001011101000001111";

tab[31] = "000000100001000011100101001010";

tab[32] = "000000010000000001000010000100";

tab[33] = "001000000000100011000010001100";

tab[34] = "000000100001010011000101001001";

tab[35] = "000000100001000010000100000110";

tab[36] = "000000000011010101011010110101";

tab[37] = "000000000010110110011000110001";

tab[38] = "000000000001110100011000101110";

tab[39] = "000000111001001011100100001000";

tab[40] = "000000000001010011000100001000";

tab[41] = "000000000000100010100001000100";

tab[42] = "000000010001110001000010000010";

tab[43] = "000000000001010010100101000110";

tab[44] = "000000000000000101011010101010";

tab[45] = "000001000101010001000101010001";

tab[46] = "000000000001010001000100010000";

tab[47] = "000000000001110001000100001110";

tab[48] = "000000000000000000000000000000";

tabTest[0] = "111111000010001111111000110001";

tabTest[1] = "011101000110001111111000110001";

tabTest[2] = "100011100110101101011000110101";

tabTest[3] = "000100001000010010101000001110";

tabTest[4] = "000010000100100010100001000100";

tabTest[5] = "100010111010101001000101010001";

tabTest[6] = "011100000110001110011000111110";

tabTest[7] = "111101000110001111101000010100";

tabTest[8] = "111101000110101111101110000011";

tabTest[9] = "011111000001110010010000111010";

Ich przedstawienie wektorowe.

Liczba prób zanim sieć nauczyła się rozpoznawać litery.

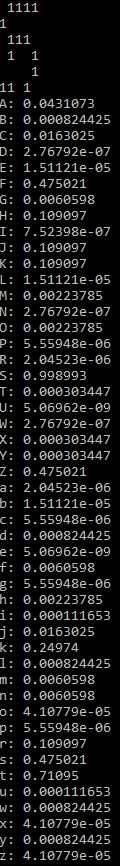
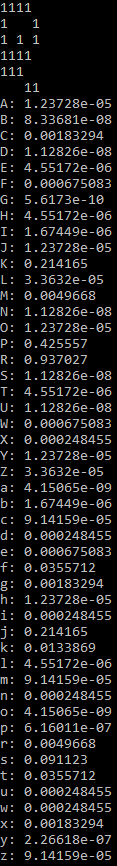
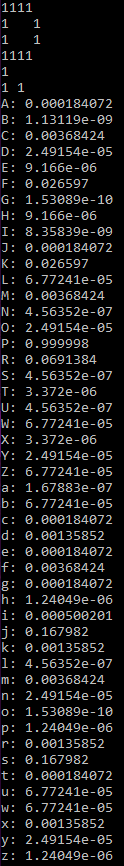
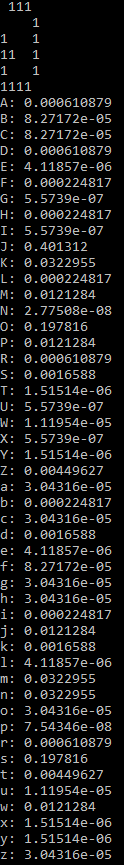
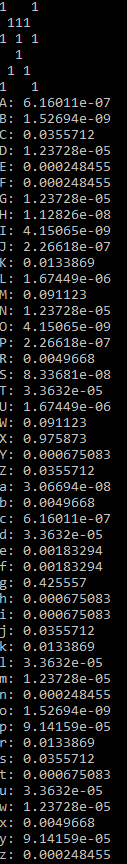
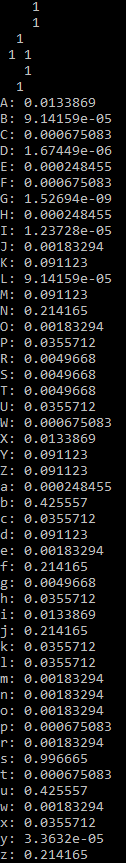
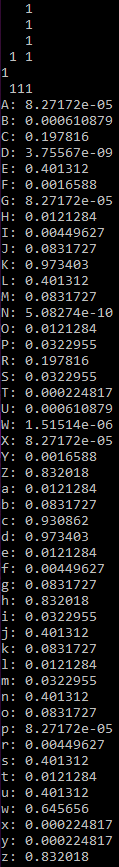
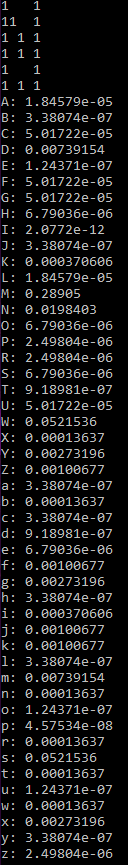
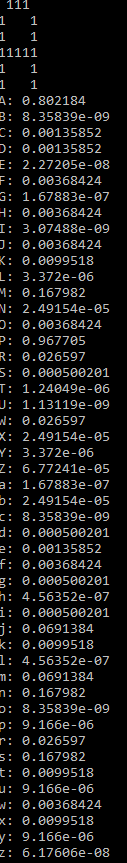
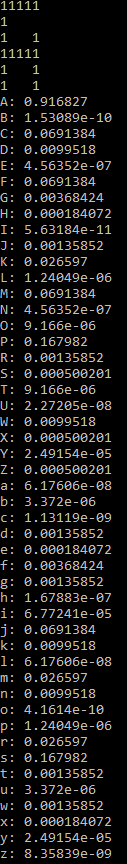
Dla współczynnika 1: 1225,8

Dla współczynnika 1,5: 1235,4

Dla tego współczynnika chociaż średnia wyszła prawie taka sama to wyniki wahały się od 700 do 2200 powtórzeń.

Dla współczynnika 2: 1410,2

Przykładowe rozpoznawanie zniekształconych literek:



Wnioski:

- Nauczanie maszynowe to potężne narzędzie, bo tylko na tym przykładzie można pokazać jak złożona dla człowieka rzecz jak wektor mogło, by się wydawać losowych zer i jedynek komputer jest w stanie przyporządkować do liter, które już rozpoznajemy bez większego problemu,

- Pomimo badania dla 3 różnych współczynników ucznenia wyniki pozostały prawie niezmienne, dla tak niskich współczynników nie ma w zasadzie znaczenia jaki wybierzemy, problemy zaczynają się dopiero przy wyższych (problemy z przeuczeniem),

- Ważniejsze natomiast wydaję mi się jest początkowe ułożenie wag, które ma ważną rolę, gdyż w wynikach bardzo często pojawiały się duże odchyły pomiędzy poszczególnymi próbami np. w jednej próbie dla tego samego współczynnika wyszło, że potrzeba 700 prób natomiast już w następnym wyszło, że tych prób potrzeba ok. 2200,

- Nauczenie się rozpoznawania takich liter zajęło komputerowi zaledwie ok. 1400 iteracji, jest to tylko kolejny dowód na to, że wszystko w przyrodzie opiera się na matematycznych modelach i tak jak dziecko uczy się czytać tak i komputer jest w stanie to zrobić. Różnica polega tylko na szybkości, komputer nauczy się szybciej natomiast człowiek dokładniej.

Listing Kodu:

Percetron.cpp

#include "Perceptron.h";

Perceptron::Perceptron(){

wspolczynnik = 1.0;

ilosc = 31;

x[0] = 1.;

}

void Perceptron::ustal\_wagi() {

srand(time(NULL));

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

w[i] = ((std::rand() % 20) - 10) / 10.;

}

void Perceptron::wypisz\_wagi() {

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

std::cout << "waga " << i + 1 << " = " << w[i] << std::endl;

}

void Perceptron::wypisz\_wejscia() {

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

std::cout << "wejscie " << i + 1 << " = " << x[i] << std::endl;

}

void Perceptron::ustal\_wejscie(std::string tmp) {

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

x[i] = tmp.at(i - 1) - 48;

}

double Perceptron::aktywacja() {

double tmp = 0.;

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

tmp += w[i] \* x[i];

tmp = 1. / (1. + exp(-tmp));

//std::cout << tmp << std::endl;

return tmp;

}

void Perceptron::zmien\_wagi(bool tmp) {

if (tmp == true)

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

w[i] += x[i] \* wspolczynnik;

else

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

w[i] -= x[i] \* wspolczynnik;

}

void Perceptron::licz(bool tmp) {

if (aktywacja() >= 0.9 && tmp == false)

zmien\_wagi(false);

else if (aktywacja() <= 0.9 && tmp == true)

zmien\_wagi(true);

}

bool Perceptron::sprawdz(Tester test, int tmp) {

for (int i = 0; i < 5; i++) {

if (tmp == i)

continue;

if (sprawdz\_bool(test.tab[i]) == true)

return false;

}

if (sprawdz\_bool(test.tab[tmp]) == false)

return false;

return true;

}

bool Perceptron::sprawdz\_bool(std::string tmp) {

ustal\_wejscie(tmp);

if (aktywacja() >= 0.9)

return true;

else

return false;

}

Perceptron.h

#pragma once

#include <iostream>

#include <cstdlib>

#include <ctime>

#include "Tester.h"

#include <string>

#include <cmath>

class Perceptron {

private:

double w[31];

double x[31];

double wspolczynnik;

double ilosc;

public:

Perceptron();

void ustal\_wagi();

void wypisz\_wagi();

void ustal\_wejscie(std::string);

void wypisz\_wejscia();

double aktywacja();

void licz(bool);

void zmien\_wagi(bool);

bool sprawdz(Tester,int);

bool sprawdz\_bool(std::string);

};

Tester.cpp

#include "Tester.h"

Tester::Tester() {

tab[0] = "111111000110001111111000110001";

tab[1] = "111111000111111100011000111111";

tab[2] = "111111000010000100001000011111";

tab[3] = "111111000110001100011000111111";

tab[4] = "111111000011111100001000011111";

tab[5] = "111111000011111100001000010000";

tab[6] = "111111000011111100011000111111";

tab[7] = "100011000111111100011000110001";

tab[8] = "011100010000100001000010001110";

tab[9] = "111110000100001000011000111111";

tab[10] = "100011001011100110001010010011";

tab[11] = "100001000010000100001000011111";

tab[12] = "100011101110101100011000110001";

tab[13] = "100011100110101101011001110001";

tab[14] = "011101000110001100011000101110";

tab[15] = "111101000110001111101000010000";

tab[16] = "111101000110001111101110010011";

tab[17] = "011111000001110000010000111110";

tab[18] = "111110010000100001000010000100";

tab[19] = "100011000110001100011000101110";

tab[20] = "100011000110001101011101110001";

tab[21] = "100010101000100001000101010001";

tab[22] = "100010101000100001000010000100";

tab[23] = "111110001000100001000100011111";

tab[24] = "000000111000001011101000101110";

tab[25] = "000000100001000011000101001100";

tab[26] = "000000000001110100001000001110";

tab[27] = "000000001000010011101001001110";

tab[28] = "000000111010001111101000001111";

tab[29] = "000000001000100011100010000100";

tab[30] = "000000111010001011101000001111";

tab[31] = "000000100001000011100101001010";

tab[32] = "000000010000000001000010000100";

tab[33] = "001000000000100011000010001100";

tab[34] = "000000100001010011000101001001";

tab[35] = "000000100001000010000100000110";

tab[36] = "000000000011010101011010110101";

tab[37] = "000000000010110110011000110001";

tab[38] = "000000000001110100011000101110";

tab[39] = "000000111001001011100100001000";

tab[40] = "000000000001010011000100001000";

tab[41] = "000000000000100010100001000100";

tab[42] = "000000010001110001000010000010";

tab[43] = "000000000001010010100101000110";

tab[44] = "000000000000000101011010101010";

tab[45] = "000001000101010001000101010001";

tab[46] = "000000000001010001000100010000";

tab[47] = "000000000001110001000100001110";

tab[48] = "000000000000000000000000000000";

tabTest[0] = "111111000010001111111000110001";

tabTest[1] = "011101000110001111111000110001";

tabTest[2] = "100011100110101101011000110101";

tabTest[3] = "000100001000010010101000001110";

tabTest[4] = "000010000100100010100001000100";

tabTest[5] = "100010111010101001000101010001";

tabTest[6] = "011100000110001110011000111110";

tabTest[7] = "111101000110001111101000010100";

tabTest[8] = "111101000110101111101110000011";

tabTest[9] = "011111000001110010010000111010";

}

void Tester::wypiszLiterke(int tmp) {

for (int i = 1; i < 31; i++) {

if (tab[tmp].at(i-1) == 49)

std::cout << "1";

else

std::cout << " ";

if (i % 5 == 0)

std::cout << std::endl;

}

}

void Tester::wypiszLiterke(std::string tmp) {

for (int i = 1; i < 31; i++) {

if (tmp.at(i - 1) == 49)

std::cout << "1";

else

std::cout << " ";

if (i % 5 == 0)

std::cout << std::endl;

}

}

void Tester::wypisz\_kilka\_literek(int tmp1, int tmp2, int tmp3, int tmp4, int tmp5, int tmp6, int tmp7, int tmp8, int tmp9, int tmp10, int tmp11, int tmp12, int tmp13, int tmp14, int tmp15) {

int tabtmp[15];

tabtmp[0] = tmp1;

tabtmp[1] = tmp2;

tabtmp[2] = tmp3;

tabtmp[3] = tmp4;

tabtmp[4] = tmp5;

tabtmp[5] = tmp6;

tabtmp[6] = tmp7;

tabtmp[7] = tmp8;

tabtmp[8] = tmp9;

tabtmp[9] = tmp10;

tabtmp[10] = tmp11;

tabtmp[11] = tmp12;

tabtmp[12] = tmp13;

tabtmp[13] = tmp14;

tabtmp[14] = tmp15;

for (int i = 0; i < 30; i += 5) {

for (int k = 0; k < 15; k++) {

for (int j = 0; j < 5; j++) {

if (tab[tabtmp[k]].at(i + j) == 49)

std::cout << "1";

else

std::cout << " ";

}

std::cout << " ";

}

std::cout << std::endl;

}

}

Tester.h

#pragma once

#include <string>

#include <iostream>

class Tester {

public:

Tester();

void wypiszLiterke(int);

void wypiszLiterke(std::string);

void wypisz\_kilka\_literek(int,int,int,int,int, int, int, int, int, int, int, int, int, int, int);

std::string tab[49];

std::string tabTest[10];

};

Źródło.cpp

#include <iostream>

#include "perceptron.h"

using namespace std;

int main() {

Tester t = Tester();

const int ilosc = 48;

Perceptron siec[ilosc];

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

siec[i] = Perceptron();

siec[i].ustal\_wagi();

}

int tmp, licznik = 0, flaga = 0;

while (true) {

tmp = rand() % ilosc;

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

siec[i].ustal\_wejscie(t.tab[tmp]);

if (i != tmp)

siec[i].licz(false);

else

siec[i].licz(true);

}

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

if (siec[i].sprawdz(t, i))

flaga++;

if (flaga == ilosc)

break;

licznik++;

flaga = 0;

}

int tmpTest = ilosc;

string best\_idea\_ever = "ABCDEFGHIJKLMNOPRSTUWXYZ";

best\_idea\_ever += "abcdefghijklmnoprstuwxyz";

cout << "Udalo sie za " << licznik << " podejsciem!" << endl;

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

cout << "Test dla " << best\_idea\_ever.at(i) << ":" << endl;

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

cout << best\_idea\_ever.at(j) << ":" << siec[i].sprawdz\_bool(t.tab[j]) << endl;

}

cout << endl;

}

//for (int i = 0; i < ilosc; i++) {

// t.wypiszLiterke(i);

// cout << endl;

//}

t.wypisz\_kilka\_literek(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14);

cout << endl;

t.wypisz\_kilka\_literek(15, 16, 17, 18, 19, 20, 21, 22, 23, 48, 48, 48, 48, 48, 48);

cout << endl;

t.wypisz\_kilka\_literek(24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38);

cout << endl;

t.wypisz\_kilka\_literek(39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 48, 48, 48, 48, 48);

cout << endl;

t.wypiszLiterke(t.tabTest[0]);

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

siec[i].ustal\_wejscie(t.tabTest[0]);

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

cout << best\_idea\_ever.at(i) << ": " << siec[i].aktywacja() << endl;

}

t.wypiszLiterke(t.tabTest[1]);

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

siec[i].ustal\_wejscie(t.tabTest[1]);

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

cout << best\_idea\_ever.at(i) << ": " << siec[i].aktywacja() << endl;

}

t.wypiszLiterke(t.tabTest[2]);

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

siec[i].ustal\_wejscie(t.tabTest[2]);

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

cout << best\_idea\_ever.at(i) << ": " << siec[i].aktywacja() << endl;

}

t.wypiszLiterke(t.tabTest[3]);

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

siec[i].ustal\_wejscie(t.tabTest[3]);

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

cout << best\_idea\_ever.at(i) << ": " << siec[i].aktywacja() << endl;

}

t.wypiszLiterke(t.tabTest[4]);

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

siec[i].ustal\_wejscie(t.tabTest[4]);

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

cout << best\_idea\_ever.at(i) << ": " << siec[i].aktywacja() << endl;

}

t.wypiszLiterke(t.tabTest[5]);

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

siec[i].ustal\_wejscie(t.tabTest[5]);

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

cout << best\_idea\_ever.at(i) << ": " << siec[i].aktywacja() << endl;

}

t.wypiszLiterke(t.tabTest[6]);

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

siec[i].ustal\_wejscie(t.tabTest[6]);

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

cout << best\_idea\_ever.at(i) << ": " << siec[i].aktywacja() << endl;

}

t.wypiszLiterke(t.tabTest[7]);

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

siec[i].ustal\_wejscie(t.tabTest[7]);

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

cout << best\_idea\_ever.at(i) << ": " << siec[i].aktywacja() << endl;

}

t.wypiszLiterke(t.tabTest[8]);

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

siec[i].ustal\_wejscie(t.tabTest[8]);

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

cout << best\_idea\_ever.at(i) << ": " << siec[i].aktywacja() << endl;

}

t.wypiszLiterke(t.tabTest[9]);

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

siec[i].ustal\_wejscie(t.tabTest[9]);

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

cout << best\_idea\_ever.at(i) << ": " << siec[i].aktywacja() << endl;

}

system("pause");

}