

10.02.2018 r.	Izabela Musztyfaga	Podstawy Sztucznej Inteligencji
Scenariusz 4 – Uczenie sieci regułą Hebba.		

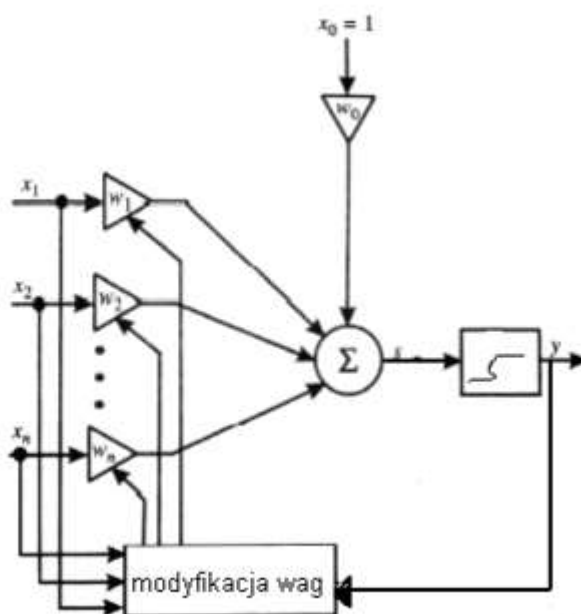
Cel ćwiczenia

Celem ćwiczenia jest poznanie działania reguły Hebba na przykładzie rozpoznawania emotikon.

Syntetyczny opis budowy użytej sieci i algorytmów uczenia ¹

W celu wykonania ćwiczenia, został wykorzystany model neuronu Hebba. Ma on identyczną strukturę jak w przypadku modelu typu Adaline oraz neuronu sigmoidalnego, ale charakteryzuje się specyficzną metodą uczenia, znaną pod nazwą reguły Hebba. *Reguła ta występuje z nauczycielem jak i bez nauczyciela.*

Hebb zauważył, iż połączenie pomiędzy dwiema komórkami jest wzmacniane, jeżeli w tym samym czasie obie komórki są aktywne.



Rys. 1 – Model Hebba

Hebb stworzył algorytm, w którym wagi są modyfikowane w następujący sposób:

$$w_i(t+1) = w_i(t) + \eta y x_i$$

Oznaczenia:

- i-numer wagi neuronu,
- t-numer iteracji w epoce,
- y-sygnał wyjściowy neuronu,
- x-wartość wejściowa neuronu,
- η - współczynnik uczenia (0,1).

¹ http://pracownik.kul.pl/files/31717/public/Model_neuronu_Hebba.pdf

Reguła ta występuje w dwóch odmianach: z nauczycielem oraz bez niego. Dodatkowo, można ją zmodyfikować, uwzględniając **współczynnik zapominania**.

Wygenerowane wartości wag mogą stać się bardzo duże, dlatego można wykorzystać wspomniany współczynnik zapominania – wtedy wzór prezentuje się tak:

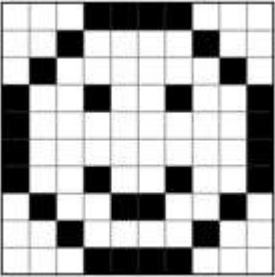
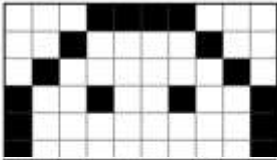
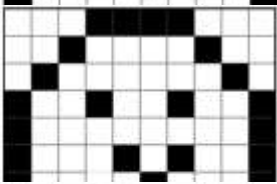
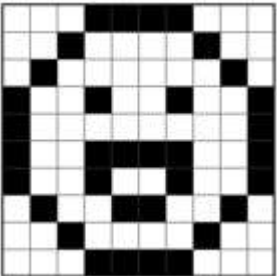
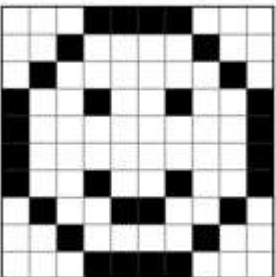
$$w_i(t+1) = (1 - \gamma) \cdot w_i(t) + n d x_i$$

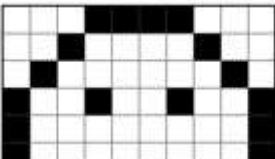
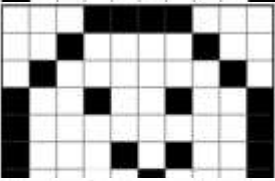
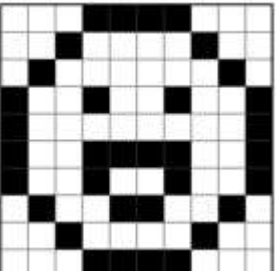
Z reguły przyjmuje on wartości od 0 do 1.

W założeniu, istnieje taka wartość jak błąd testowania dla jednej epoki. Neuron uczy się do momentu, aż błąd ten nie będzie mniejszy niż jego ustalona wartość.

$$E = E + \frac{1}{2} \sum (d_i - y_i)^2$$

Dane uczące i testowe

Dane uczące		
Lp.	Emotka	Ciąg bitów
1		0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 1 0 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 1 1 1 0 0 0
2		0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0
3		0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0
4		0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 0 0 1 1 0 0 1 0 0 1 0 0 1 0 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0
ujące – zamienione niektóre wartości bitów		
Lp.		Ciąg bitów
1		0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0

		0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 1 1 0 0 1 0 0 1 0 0 1 0 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0
2		0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0
3		0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0
4		0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 0 0 1 0 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0

Algorytm uczenia metodą Hebba

1. Wybrać początkowe wartości wag wejściowych z zakresu (0;1>
2. Wylosować początkowe wagi wejść, również z zakresu (0;1>
3. Obliczyć odpowiedź sieci dla podanych zbiorów danych.
4. Modyfikacja wag – proces uczenia

4.1 Metoda Hebba bez zapominania – wzór obliczania wag

$$w_{i,j}^k(t+1) = (w_{i,j}(t)) + \eta * x_i * a_i$$

4.2. Metoda Hebba z zapominaniem – wzór obliczania wag

$$w_{i,j}^k(t+1) = (w_{i,j}(t)) * (1 - \gamma) + \eta * x_i * a_i$$

5. Obliczyć błąd uczenia.

$$E = E + \frac{1}{2} \sum (d_i - y_i)^2$$

6. Uczyć, aż błąd uczenia nie będzie niższy niż określony.

Zestawienie otrzymanych wyników po przeprowadzonym uczeniu i testowania sieci

Uczenie z zapominaniem			
Lp.	Współczynnik uczenia	Współczynnik zapominania	Ilość epok

1	0.01	0.01	3000
2	0.01	0.1	3000
3	0.01	0.5	569
4	0.1	0.01	3000
5	0.1	0.1	3000
6	0.1	0.5	569
7	0.5	0.01	3000
8	0.5	0.1	3000
9.	0.5	0.5	569

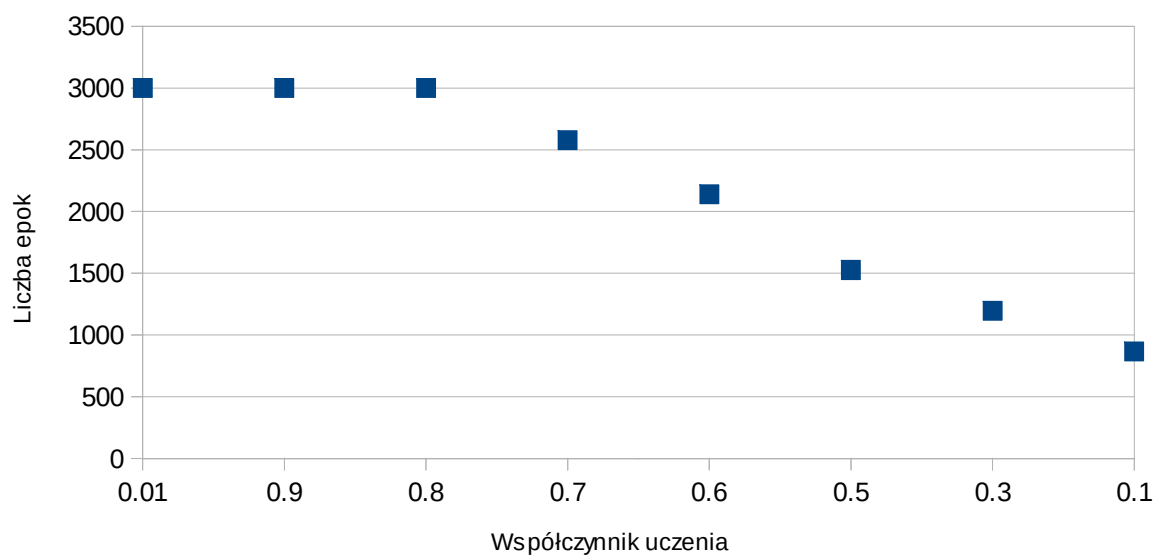
Uczenie bez zapominania		
Lp.	Współczynnik uczenia	Ilość epok
1	0.01	3000
2	0.9	3000
3	0.8	3000
4	0.7	2578
5	0.6	2139
6	0.5	1527
7	0.3	1196
8	0.1	867

Przeprowadzono 10 prób dla każdej sytuacji

Poprawne rozpoznawanie emotikon ze względu na współczynnik zapominania				
Lp.	Współczynnik uczenia	Współczynnik zapominania	Emotikona	Ilość poprawnych rozpoznań
1	0.5	0.1	1	9
			2	10
			3	8
			4	6
2	0.5	0.5	1	5
			2	8
			3	4
			4	3
3	0.5	0.8	1	4
			2	1
			3	4

			4	2
--	--	--	---	---

Uczenie bez zapominania



Analiza otrzymanych wyników

Cały proces uczenia metodą Hebba jest ściśle związany z wartością współczynnika uczenia. Im on mniejszy, tym mniej epok potrzebne, aby sieć zaczęła się efektywnie uczyć.

Trzeba również zwrócić uwagę na współczynnik zapominania – im większy, tym końcowy efekt jest mniej efektywny.

Wnioski

Największy wpływ na cały proces uczenia ma współczynnik uczenia. Im jest on większy (bliższy 1), tym szybciej możemy zaobserwować efektywne uczenie się neuronu.

Mniejszy, ale również pewien wpływ na cały proces uczenia ma współczynnik zapominania (lub jego brak). Twórca sieci powinien dobrać na tyle mały ten współczynnik, aby sieć nie zapominała wszystkich informacji które zdążyła przyswoić.

Błędne rozpoznawanie emotikon nie oznacza, że sieć działa wadliwie. Są one po prostu do siebie bardzo podobne i możliwe, że sieć nie dostrzega drobnych różnic.

Listing kodu

main.cpp

```
Neuron.h  Neuron.cpp  main.cpp  X
Scenariusz 4 version 1  (Global Scope)  ma

1 #include <iostream>
2 #include "Neuron.h"
3 using namespace std;
4
5 int main() {
6
7     const double LEARNING_RATE = 0.1;
8     cout << endl << "Wspolczynnik uczenia: " << LEARNING_RATE << endl << endl;
9
10
11     Neuron hDontForget(LEARNING_RATE);
12     hDontForget.learnDontForget();
13
14     //Neuron hForget(LEARNING_RATE);
15     //hForget.learnForget();
16
17
18
19     system("pause");
20
21 }
```

Neuron.h

```
Neuron.h  X  Neuron.cpp  main.cpp
Scenariusz 4 version 1  Neuron  Neuro

1 #include <ctime>
2 #include <string>
3 #include <fstream>
4 #include <iostream>
5 #include <string>
6
7 #define HOW_MANY_BITS 10*10
8 #define HOW_MANY_EMOTICONS 4
9
10 using namespace std;
11
12 class Neuron
13 {
14 public:
15
16     int emoticons[4][HOW_MANY_BITS]; //Tablica danych uczacych
17     int emoticonsTest[4][HOW_MANY_BITS]; //Tablica danych testujacych
18     double learningRate; //Wspolczynnik uczenia
19     double weights[4][HOW_MANY_BITS]; //Wagi
20     double sumOfInput[HOW_MANY_BITS]; //Suma: wagi * dane uczace
21     double emoticonTest[HOW_MANY_BITS]; //Tablica do przechowywania wynikow w czasie testowania
22
23     Neuron(double); //Konstruktor
24     ~Neuron(); //destruktor
25
26     void learnDontForget(); //Uczenie bez zapominania
27     void learnForget(); //Uczenie z zapominaniem
28     int activationFunction(double); //Funkcja aktywacji (progowa bipolarna)
29     void resetSum(); //Ustawienie sum na 0
30     void test(); //Testowanie rozpoznawania emotikon
31     void result(); //Wypisywanie wyniku testowania
32
33 }
```

Neuron.cpp

```
#include "Neuron.h"

Neuron::Neuron(double learningRate) //konstruktor
{
    this->learningRate = learningRate;
    //Zerowanie tablic wag,sum oraz pomocnicza do testow
    for (int i = 0; i<HOW_MANY_EMOTICONS; i++)
        for (int j = 0; j < HOW_MANY_BITS; j++)
        {
            weights[i][j] = 0;
            sumOfInput[i] = 0;
            emoticonTest[i] = 0;
        }

    //Losowanie wag poczatkowych z przedziału od 0 do 1
    srand(time(NULL));
    for (int i = 0; i<HOW_MANY_BITS; i++) {
        weights[0][i] = (double)rand() / (double)RAND_MAX;//dla emotki 1 dla wszystkich
        "komponentow" itd dalej dla 2,3,4 emotki
        weights[1][i] = (double)rand() / (double)RAND_MAX;
        weights[2][i] = (double)rand() / (double)RAND_MAX;
        weights[3][i] = (double)rand() / (double)RAND_MAX;
    }

    //Wczytanie danych uczacych
    int whichEmoticon = 0, whichBit = 0, bit;//pomocnicze
    fstream file;
    file.open("dane.txt");
    if (file.is_open()) {
        while (!file.eof()) {
            //Jesli wczytano juz wszystkie bity (100=10x10) danej emotikony,
            przejscie do wczytywania bitow kolejnej
            if (whichBit == 225) {
                whichBit = 0; whichEmoticon++;
            }
            //Jesli wczytano wszystkie (4) emotikony, zakoncz wczytywanie
            if (whichEmoticon == HOW_MANY_EMOTICONS)
                break;

            file >> bit;
            emoticons[whichEmoticon][whichBit] = bit;
            whichBit++;
        }
        file.close();
    }

    //Wczytanie danych testujacych
    fstream file2;
    file2.open("dane_testowe.txt");
    if (file2.is_open()) {
        while (!file2.eof()) {

            if (whichBit == 225) {
                whichBit = 0; whichEmoticon++;
            }

            if (whichEmoticon == HOW_MANY_EMOTICONS)
                break;

            file2 >> bit;
            emoticonsTest[whichEmoticon][whichBit] = bit;
        }
    }
}
```

```

        whichBit++;
    }
    file2.close();
}

Neuron::~Neuron()           //destruktor
{
}

//Funkcja aktywacji - progowa unipolarna
int Neuron::activationFunction(double sum)
{
    return (sum >= 0) ? 1 : 0;           //1 gdy x>=wartosc progowa, 0
    gdy x<wart.prog, wart.prog=0 - moze byc inna
}

//Ustawienie sum (wejście*waga) na wartość początkowa = 0
void Neuron::resetSum()
{
    int i = 0;
    for (i = 0; i<HOW_MANY_BITS; i++)
        sumOfInput[i] = 0.;
}

//Uczenie reguła Hebba (bez zapominania)
void Neuron::learnDontForget()
{
    int epoch = 0; //Epok
    double globalError = 0.; //Błąd globalny
    double localErr = 0.; //Błąd lokalny
    double sum = 0.; //Przechowuje sume wejście*waga (sumOfInput)

    resetSum(); //Ustawienie wartości początkowych dla wektora sumOfInput
    cout << "Uczenie reguła Hebba (bez zapominania)" << endl;

    do {
        for (int i = 0; i<HOW_MANY_EMOTICONS; ++i) {
            globalError = 0.;
            for (int j = 0; j<HOW_MANY_BITS; ++j) {
                sum = sumOfInput[j];
                sumOfInput[j] = (weights[i][j] * emoticons[i][j]);

                //Aktualizacja wag
                weights[i][j] = weights[i][j] + this->learningRate*sumOfInput[j]
* emoticons[i][j];

                if (localErr == abs(sum - sumOfInput[j]))
                    break;
                localErr = abs(sum - sumOfInput[j]);
                globalError = globalError + pow(localErr, 2);
            }
        }
        epoch++;
    } while (globalError != 0 && epoch < 3000);

    cout << endl << "Liczba epok: " << epoch << endl << endl << endl;
}

```



```

        test();
    }

//Uczenie reguła Hebba (z zapominaniem)
void Neuron::learnForget() {

    int epoch = 0; //Epoka
    double FORGET_RATE = 0.5; //Wspolczynnik zapominania
    double globalError = 0.; //Bład globalny
    double localErr = 0.; //Bład lokalny
    double sum = 0.; //Przechowuje sume wejscie*waga (sumOfInput)

    resetSum(); //Ustawienie wartosci poczatkowych dla wektora sumOfInput

    cout << "Uczenie reguła Hebba (z zapominaniem)" << endl;

    do {
        for (int i = 0; i<HOW_MANY_EMOTICONS; ++i) {
            globalError = 0.;
            for (int j = 0; j<HOW_MANY_BITS; ++j) {
                sum = sumOfInput[j];

                sumOfInput[j] = (weights[i][j] * emoticons[i][j]);

                //Aktualizacja wag
                weights[i][j] = weights[i][j] * (1 - FORGET_RATE) + this-
>learningRate*sumOfInput[j] * emoticons[i][j];

                if (localErr == abs(sum - sumOfInput[j])) break;
                localErr = abs(sum - sumOfInput[j]);
                globalError = globalError + pow(localErr, 2);
            }
        }
        epoch++;
    } while (globalError != 0 && epoch < 3000);

    cout << endl << "Liczba epok: " << epoch << endl;

    test();

    cout << endl << "Wspolczynnik zapominania = " << FORGET_RATE << endl;
}

//Przetestowanie nauczania danej emotikony
void Neuron::test()
{
    double globalError = 0.;
    double localErr = 0.;
    double sum = 0.;
    int epoch = 0;

    resetSum();

    cout << endl << "Testowana emotikona: 1" << endl;

    do {
        epoch++;
        for (int i = 0; i<HOW_MANY_BITS; i++) {
            sum = sumOfInput[i];

```

```
sumOfInput[i] = (weights[0][i] * emoticonsTest[0][i]);

emoticonTest[i] = activationFunction(sumOfInput[i]);
localErr = abs(sum - sumOfInput[i]);
globalError = globalError + pow(localErr, 2);
}

} while (globalError != 0 && epoch<1);

result();

resetSum();

cout << endl << "Testowana emotikona: 2" << endl;

do {
    epoch++;
    for (int i = 0; i<HOW_MANY_BITS; i++) {
        sum = sumOfInput[i]; //suma wejśc
        sumOfInput[i] = (weights[0][i] * emoticonsTest[1][i]); //s=w*dane_ucz

        emoticonTest[i] = activationFunction(sumOfInput[i]); //f(s)
        localErr = abs(sum - sumOfInput[i]);
        globalError = globalError + pow(localErr, 2);
    }

} while (globalError != 0 && epoch<1);

result();

resetSum();

cout << endl << "Testowana emotikona: 3" << endl;

do {
    epoch++;
    for (int i = 0; i<HOW_MANY_BITS; i++) {
        sum = sumOfInput[i];
        sumOfInput[i] = (weights[0][i] * emoticonsTest[2][i]);

        emoticonTest[i] = activationFunction(sumOfInput[i]);
        localErr = abs(sum - sumOfInput[i]);
        globalError = globalError + pow(localErr, 2);
    }

} while (globalError != 0 && epoch<1);

result();

resetSum();

cout << endl << "Testowana emotikona: 4" << endl;

do {
    epoch++;
    for (int i = 0; i<HOW_MANY_BITS; i++) {
        sum = sumOfInput[i];
        sumOfInput[i] = (weights[0][i] * emoticonsTest[3][i]);
```

```

        emoticonTest[i] = activationFunction(sumOfInput[i]);
        localErr = abs(sum - sumOfInput[i]);
        globalError = globalError + pow(localErr, 2);
    }

    } while (globalError != 0 && epoch<1);

    result();
}

//Sprawdzenie czy dobrze rozpoznaje emotikony
void Neuron::result()
{
    int tmp[HOW_MANY_EMOTICONS];

    for (int i = 0; i<HOW_MANY_EMOTICONS; i++)
        tmp[i] = 0;

    for (int i = 0; i<HOW_MANY_BITS; ++i) {
        if (emoticonTest[i] == emoticons[0][i]) tmp[0]++;
        if (emoticonTest[i] == emoticons[1][i]) tmp[1]++;
        if (emoticonTest[i] == emoticons[2][i]) tmp[2]++;
        if (emoticonTest[i] == emoticons[3][i]) tmp[3]++;
    }

    if (tmp[0] > tmp[1] && tmp[0] > tmp[2] && tmp[0] > tmp[3])
        cout << "Rozpoznana emotikona: 1" << endl;

    if (tmp[1] > tmp[0] && tmp[1] > tmp[2] && tmp[1] > tmp[3])
        cout << "Rozpoznana emotikona: 2" << endl;

    if (tmp[2] > tmp[0] && tmp[2] > tmp[1] && tmp[2] > tmp[3])
        cout << "Rozpoznana emotikona: 3" << endl;

    if (tmp[3] > tmp[0] && tmp[3] > tmp[1] && tmp[3] > tmp[2])
        cout << "Rozpoznana emotikona: 4" << endl;
}

```