

## Scenariusz 4

Białek Tomasz, gr. 1

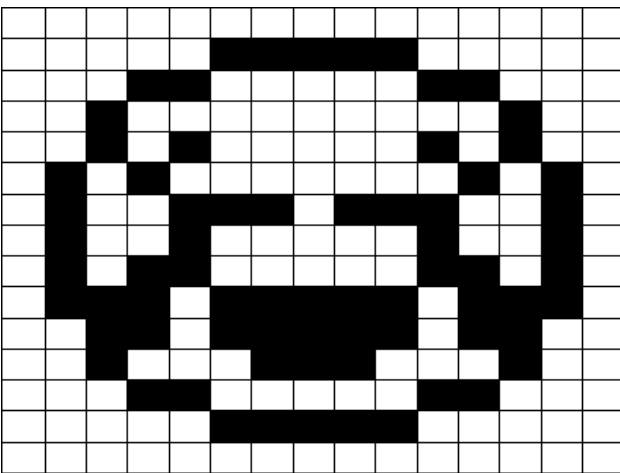
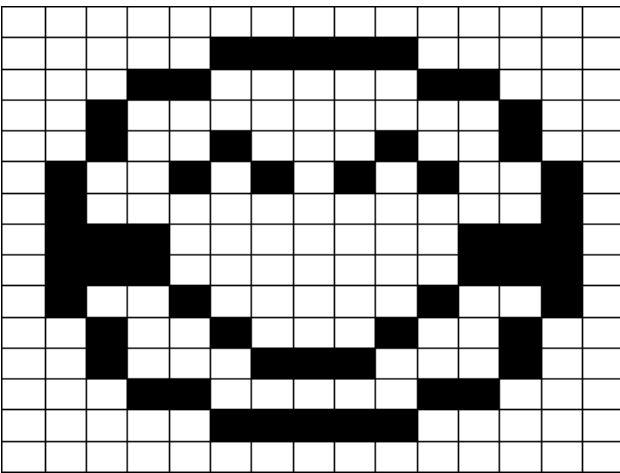
**Temat ćwiczenia: Uczenie sieci regułą Hebba**

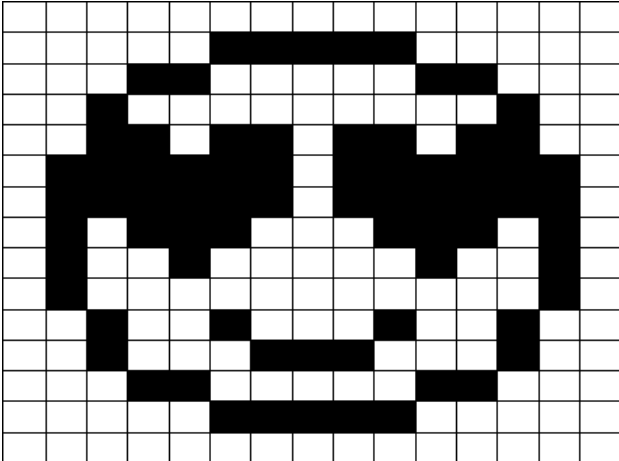
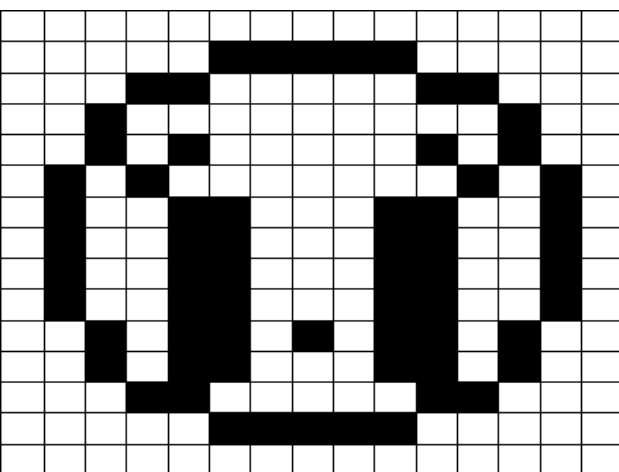
### 1. Cel ćwiczenia

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

### 2. Opis budowy sieci i algorytmów uczenia.

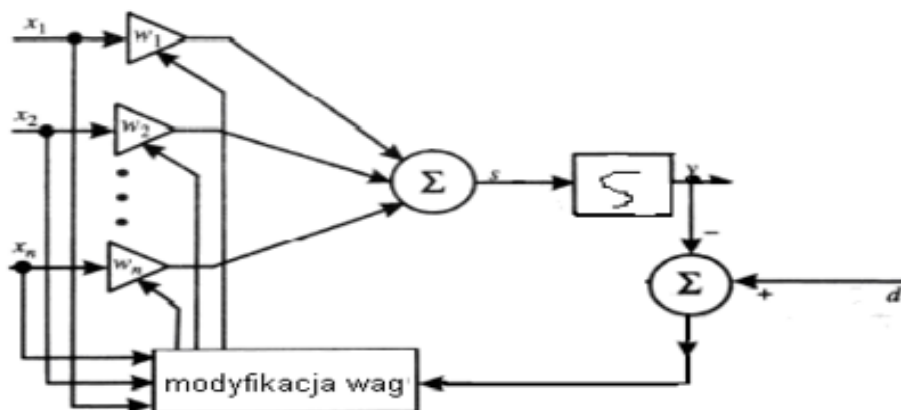
Emotikony wykorzystane w ćwiczeniu: ŚMIECH, UŚMIECH, SERCE, PŁACZ

ŚMIECH		<pre> -1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1 1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 1 -1 -1 1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 1 -1 -1 1 -1 -1 1 1 1 1 -1 1 1 1 1 -1 1 1 -1 1 -1 -1 1 -1 -1 -1 -1 -1 1 1 -1 -1 1 -1 1 -1 1 1 -1 -1 -1 -1 -1 1 1 -1 -1 1 -1 1 -1 1 1 -1 -1 -1 -1 -1 1 1 1 -1 1 -1 1 1 1 -1 1 1 1 1 1 1 1 1 1 -1 -1 -1 1 1 -1 1 1 1 1 1 -1 1 1 -1 -1 -1 -1 1 1 -1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 </pre>
UŚMIECH		<pre> -1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 1 -1 -1 1 -1 1 1 -1 1 1 -1 -1 1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 1 -1 -1 1 -1 -1 -1 -1 -1 1 1 -1 1 1 -1 -1 1 -1 -1 1 -1 -1 -1 1 -1 1 1 -1 -1 -1 -1 1 -1 -1 -1 1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 </pre>

SERCE		-1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 1 -1 1 1 -1 1 1 -1 1 1 -1 -1 -1 1 1 1 1 1 1 -1 1 1 1 1 1 -1 -1 1 1 1 1 1 1 -1 1 1 1 1 1 -1 -1 1 -1 1 1 1 -1 -1 -1 1 1 1 -1 1 -1 -1 1 -1 1 1 -1 -1 -1 -1 -1 1 -1 1 1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 1 -1 -1 1 -1 -1 -1 1 -1 1 1 -1 -1 -1 -1 1 -1 -1 -1 1 1 1 -1 -1 -1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
PŁĄCZ		-1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 1 -1 1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 1 -1 1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 1 -1 1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 1 -1 1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 -1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 -1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 -1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 -1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

### Syntetyczny opis sieci

Model sieci neuronowej Hebba ma podobną strukturę w porównaniu do modelu Adaline, jednakże różnią się one samą regułą Hebba. Reguła ta występuje w dwóch wersjach: nauka z nauczycielem i bez nauczyciela. Przy implementacji można zastosować, lecz nie trzeba, współczynnik zapominania, który wspomaga uczenie sieci.



Model sieci Hebba

### Algorytm uczenia

Proces uczenia przebiega według następującego schematu:

1. Wybór współczynnika uczenia  $\eta$  z zakresu  $(0; 1)$
2. Wylosowanie początkowych wartości wag z zakresu  $(0; 1)$
3. Dla danego zbioru uczącego obliczamy odpowiedź sieci – dla każdego pojedynczego neuronu obliczana jest suma ilorazów sygnałów wejściowych oraz wag.
4. Modyfikacja wag odbywa się według następujących zależności:
  - a) Reguła Hebba (bez zapominania)

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

- b) Reguła Hebba (z zapominaniem)

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

- c) Reguła Oji

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

$\eta$  – współczynnik uczenia

$x$  – zbiór danych wejściowych

$\gamma$  – współczynnik zapominania

$a$  – suma wyjściowa

5. Obliczenie łącznego błędu epoki

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

6. Uczenie odbywa się aż błąd nie będzie niższy niż określony próg

### 3. Zestawienie otrzymanych wyników

```
Liczba epok uczenia [Hebb bez zapominania]: 935
Do testu emotikona SMIECH
----- PODOBIENSTWO [ZMODYFIKOWANE W 20%] -----
Smiech: 88.8889
Usmiech: 71.1111
Serce: 73.7778
Placz: 81.3333
Wspolczynnik uczenia: 0.1
```

```

Liczba epok uczenia [Hebb z zapominaniem]: 54

Do testu emotikona SERCE
----- PODOBIENSTWO [ZMODYFIKOWANE W 20%] -----
Smiech: 69.7778
Uśmiech: 71.5556
Serce: 84
Placz: 70.2222

Współczynnik uczenia: 0.1

```

```

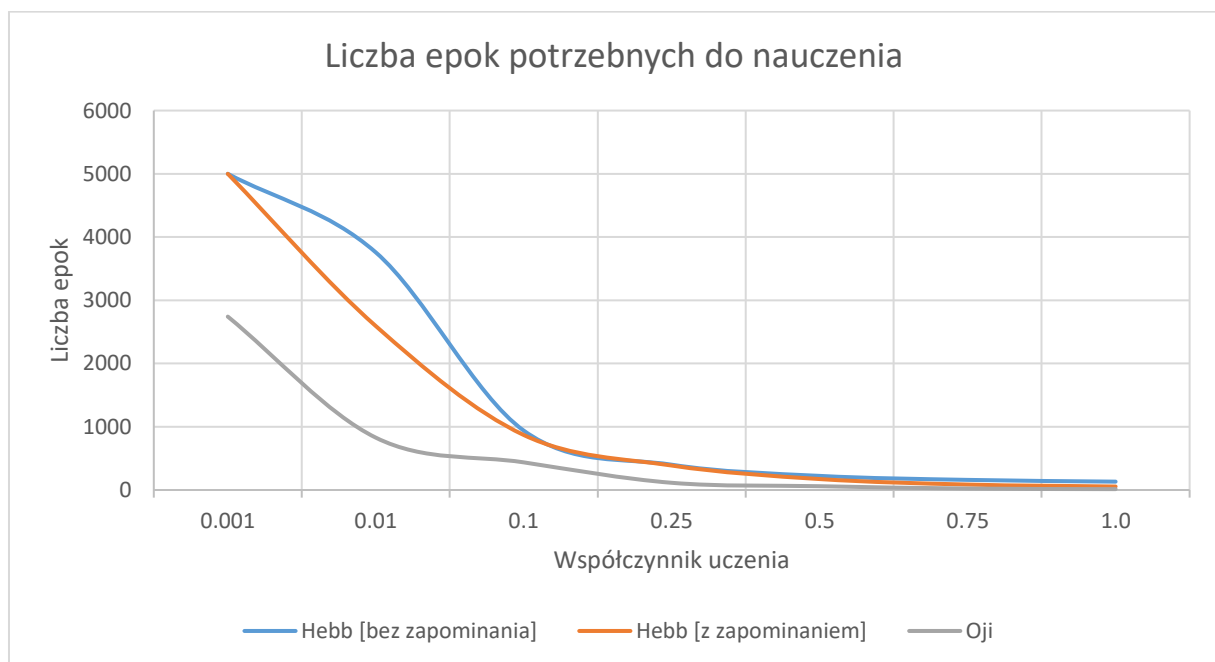
Liczba epok uczenia [Oji]: 109

Do testu emotikona SERCE
----- PODOBIENSTWO [ZMODYFIKOWANE W 20%] -----
Smiech: 86.2222
Uśmiech: 70.2222
Serce: 75.5556
Placz: 74.2222

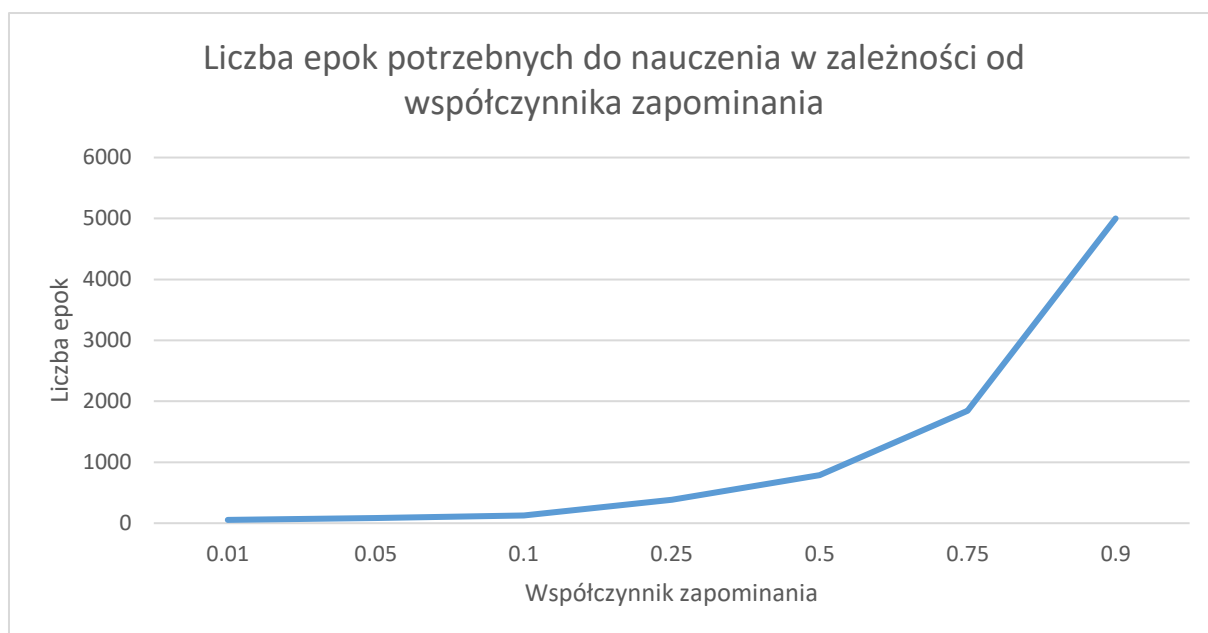
Współczynnik uczenia: 0.1

```

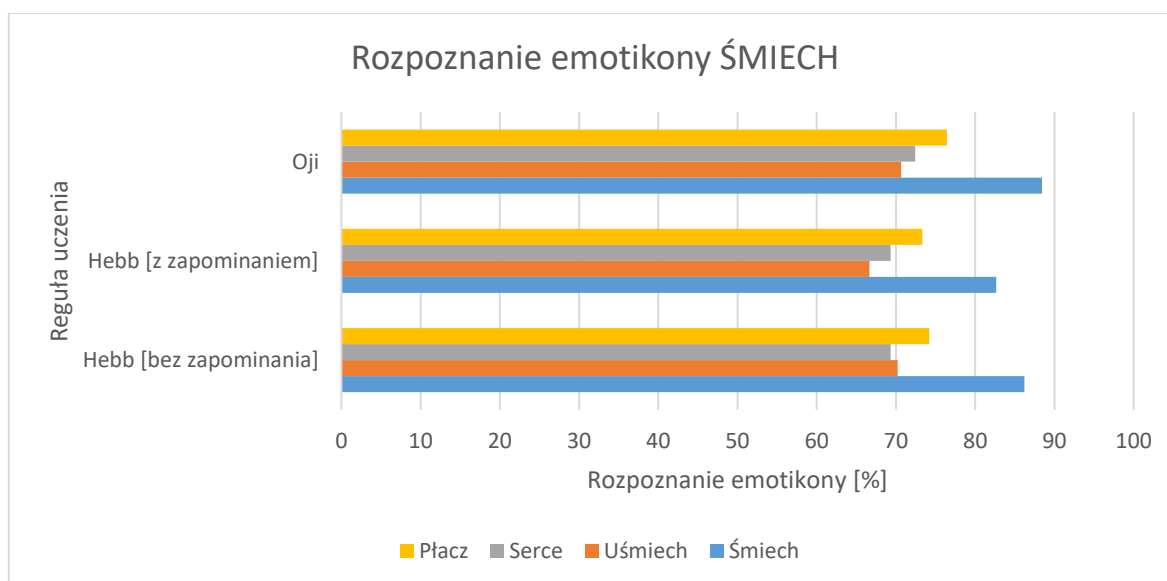
### Analiza

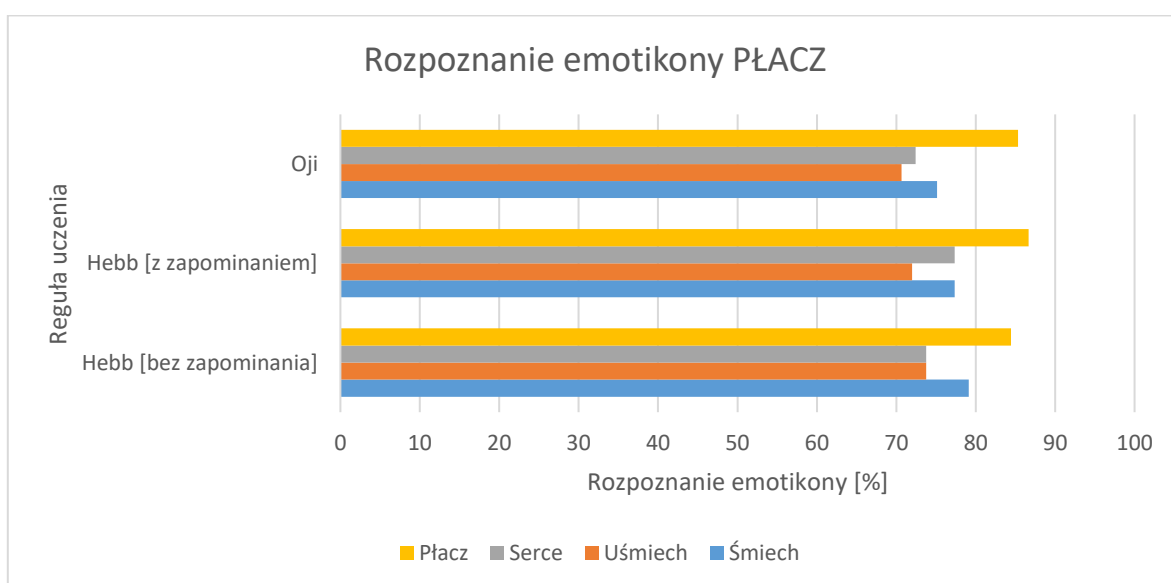
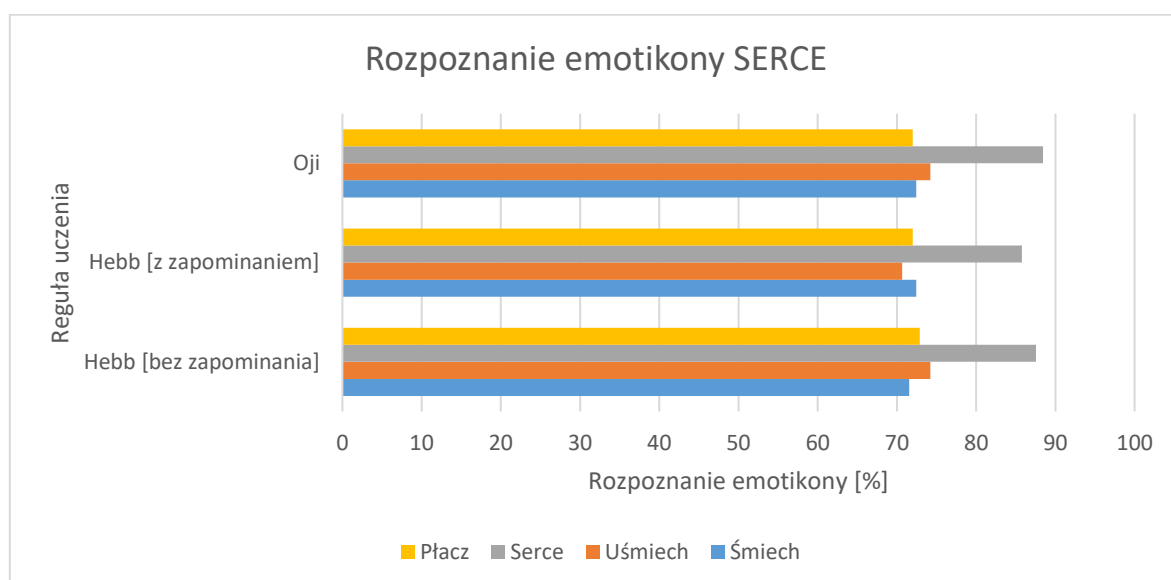
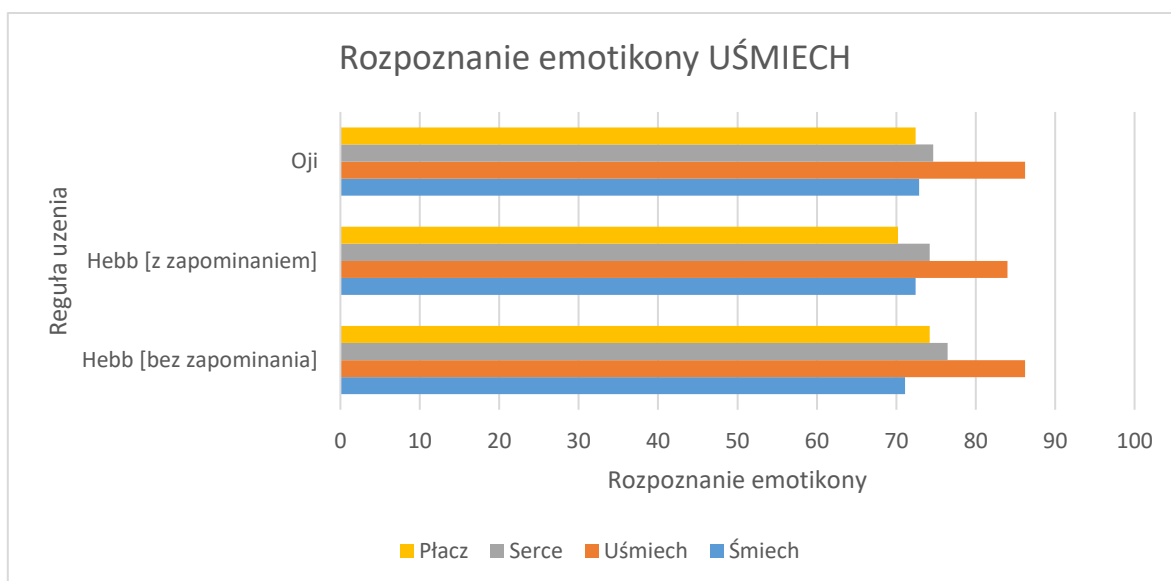


Z powyższego wykresu wynika, że efektywność uczenia w metodzie Hebba jest ściśle powiązana ze współczynnikiem uczenia. Niezależnie od zastosowanej reguły, im większy współczynnik uczenia, tym mniej epok potrzebnych do nauki. Dla bardzo małych współczynników uczenia, efektywność nauki dla reguły Hebba jest bardzo niska (tutaj ograniczona do 5000 epok, po czym nauka zostaje przerwana z powodu oszczędności czasu). Spośród trzech reguł najefektywniejszą jest reguła Oji, gdzie dopiero reguły Hebba osiągają taką efektywność jak Oji przy dużych współczynnikach uczenia (powyżej 0.75).

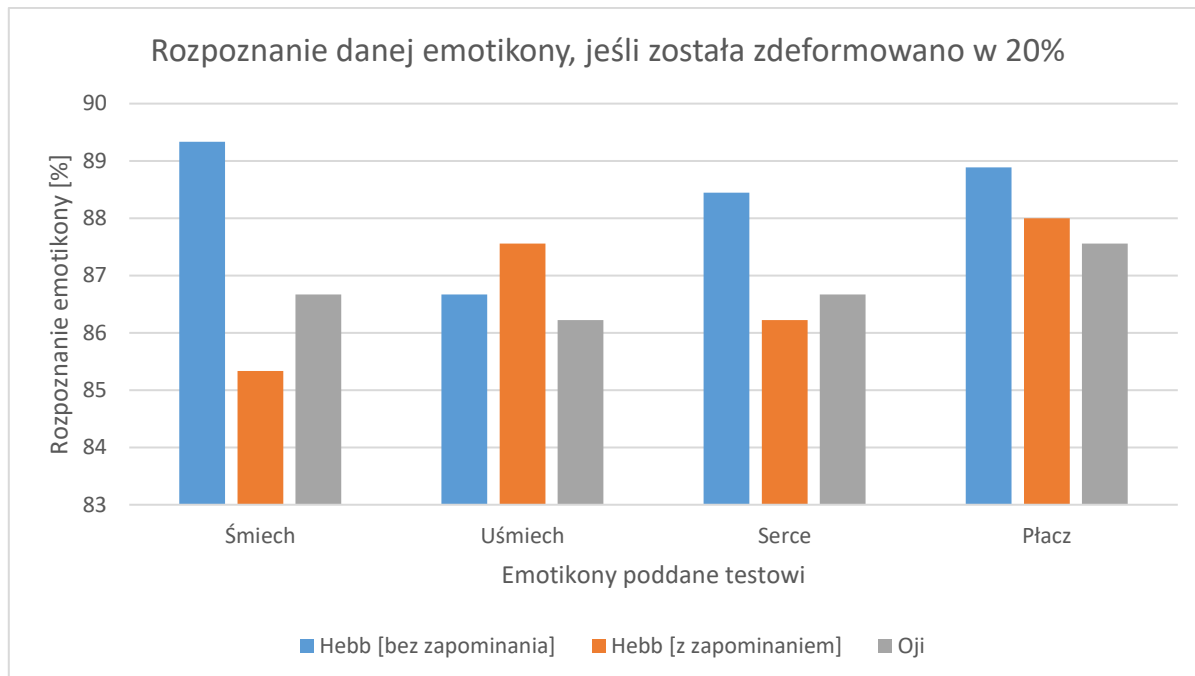


Wykres ten dotyczy tylko jednej z reguł (reguła Hebb, ze współczynnikiem zapominania). Z wykresu można wywnioskować, że reguła Hebb posiada bardziej efektywne uczenie, jeśli współczynnik zapominania jest niewielki, co pozwala osiągnąć stabilność wag. Dla większych współczynników zapominania sieć uczy się mniej efektywnie, ponieważ to co zostało nauczone, zostaje zapomniane.





Z powyższych czterech wykresów można odczytać, że sieć podczas testów odgadywała poprawnie jaką emotikona została podana. W każdym przypadku emotikona testująca (zdeformowana w 20%) osiągnęła największe podobieństwo do emotikony, którą rzeczywiście jest. Taka spora niepewność przy podobieństwie może być spowodowana tym, że emotikony wyglądają podobnie (szczególnie, gdy są tylko reprezentowane w kolorze białym i czarnym).



Wykres ten jest podsumowaniem czterech poprzednich wykresów. Zostały w nim zebrane wszystkie testy. Można z niego odczytać, że największą efektywność w rozpoznawaniu rodzaju emotikony ma sieć wykorzystująca regułę Hebba (z wariantem bez zapominania), jednakże sieć taka uczy się dłużej niż pozostałe, a różnice w testach wyniosły maksymalne około 4%.

#### 4. Podsumowanie

Skuteczność procesu uczenia zależy od współczynnika uczenia. Wraz z jego wzrostem proces uczenia jest efektywniejszy. Jest to wytłumaczalne z tego względu, że im ta wartość jest większa tym przyrost wag, które na samym początku są niewielkie jest szybszy, więc proces uczenia przebiega szybciej. Sieci wykorzystujące regułę Hebba oraz Oji są odporne na zaszumienie (do kilkudziesięciu bitów). Zbyt duże zaszumienie powoduje błędne odpowiedzi dawane przez sieć, niezależnie od zastosowanej reguły. Przy projektowaniu sieci neuronowej trzeba wybrać odpowiedni jej model. Oprócz samego modelu na jej efektywność ma wpływ sama jej struktura, np. zastosowana funkcja aktywacji. Model wykorzystujący regułę Oji pozwala na o wiele sprawniejsze uczenie sieci w porównaniu do modelu wykorzystującego regułę Hebba. W samym modelu z regułą Hebba wpływ na efektywność uczenia sieci ma zastosowany rodzaj – z współczynnikiem zapominania lub bez. Reguła z współczynnikiem zapominania jest bardziej efektywna aniżeli reguła bez tego współczynnika. Jednakże trzeba uważać przy samym doborze wartości współczynnika zapominania, ponieważ wartości bliższe jedynce powodują, że sieć zapomina to, co przed chwilą nauczyła się.

## 5. Kod programu

„Source.cpp”

```
#include "Hebb.h"

int main() {

    double LEARNING_RATE = 0.1;

    //Hebb hebbWithoutForgetting(LEARNING_RATE);
    //hebbWithoutForgetting.learnWithoutForgetting();

    //Hebb hebbWithForgetting(LEARNING_RATE);
    //hebbWithForgetting.learnWithForgetting();

    Hebb hebbOjiRule(LEARNING_RATE);
    hebbOjiRule.learnOjiRegule();

    cout << endl << "Wspolczynnik uczenia: " << LEARNING_RATE << endl << endl;

    system("pause");

    return 0;
}
```

„Hebb.h”

```
#include "Include.h"

class Hebb {
public:
    // dane uczace:
    int emoticons[4][225];
    //0 - smiech
    //1 - usmiech
    //2 - serce
    //3 - placz

    double learningRate; //wspolczynnik uczenia

    //cztery zdeformowane emotikony testujace
    int inputSmiech80pr[225];
    int inputUsmiech80pr[225];
    int inputSerce80pr[225];
    int inputPlacz80pr[225];

    double weights[4][225]; //wagi
    double sumOfInput[225]; //suma: wagi * dane uczace
    double emoticonTest[225]; //wyniki dla danych testujacych (przechowuje wynik z
funkcji aktywacji)

    int activationFunction(double); //funkcja aktywacji (progowa bipolarna)

    void generateWeights(); //losowanie wag z zakresu <0;1>
    void generateTestInputs(); //stworzenie czterech zdeformowanych emotikon
testujacych
    void resetSum(); //ustawienie sum na 0
    void readFromFile(); //wczytanie z pliku bitow danych uczacych i zapisanie do
zmiennych

    void learnWithoutForgetting(); //uczenie Hebb'em bez zapominania
    void learnWithForgetting(); //uczenie Hebb'em z zapominaniem
}
```



```

void learnOjiRegule(); //uczenie regula Oji

void test(); //przetestowanie zdeformowanych emotikon
void compare(); //w ilu % siec rozpozнала dobrze dana emotikone

Hebb(double); //konstruktor
};

                                                                    „Hebb.cpp”

#include "Hebb.h"

Hebb::Hebb(double learningRate){
    this->learningRate = learningRate;

    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;
        }
}

//funkcja progowa bipolarna
int Hebb::activationFunction(double sum){
    return (sum >= 0) ? 1 : -1;
}

//losowanie wag dla kazdej z emotikon [przedzial 0;1]
void Hebb::generateWeights(){
    srand(time(NULL));
    for (int i = 0; i<HOW_MANY_BITS; i++) {
        weights[0][i] = (double)rand() / (double)RAND_MAX;
        weights[1][i] = (double)rand() / (double)RAND_MAX;
        weights[2][i] = (double)rand() / (double)RAND_MAX;
        weights[3][i] = (double)rand() / (double)RAND_MAX;
    }
}

//ustawienie sum (wejście*waga) na wartosc poczatkowa = 0
void Hebb::resetSum(){
    int i = 0;
    for (i = 0; i<HOW_MANY_BITS; i++)
        sumOfInput[i] = 0.;
}

//wczytanie z pliku danych do nauki
void Hebb::readFromFile(){
    int whichEmoticon = 0, whichBit = 0, bit;
    fstream file;
    file.open("dane.txt");
    if (file.is_open()) {
        while (!file.eof()) {
            //jesli wczytano juz wszystkie bity (225) danej emotikony,
            //przejdzie 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();
}

//uczenie regula Hebba (bez zapominania)
void Hebb::learnWithoutForgetting() {
    int epoch = 0; //epoka

    double globalError = 0.; //blad globalny
    double localError = 0.; //blad lokalny
    double MSE = 0.; //blad sredniokwadratowy
    double MAPE = 0.; //blad bezwzględny

    double sum = 0.; //przechowuje sume wejscie*waga (sumOfInput)

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

    generateWeights(); //losowanie wag z przedziału 0;1
    readFromFile(); //wczytanie do 4 wektorów wszystkich (225) bitów w celu uczenia

    do {
        cout << "Epoka: " << epoch << endl;

        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 (localError == abs(sum - sumOfInput[j]))
                    break;
                localError = abs(sum - sumOfInput[j]);
                globalError = globalError + pow(localError, 2);

            }

            MSE = pow(globalError, 2) / HOW_MANY_BITS;
            MAPE = (globalError * 10) / HOW_MANY_BITS;
            cout << "MSE: " << MSE << "\tMAPE: " << MAPE << "%" << endl;
        }
        epoch++;

    } while (globalError != 0 && epoch<5000);

    cout << endl << "Liczba epok uczenia [Hebb bez zapominania]: " << epoch << endl;

    test();
}

//uczenie regułą Hebba (z zapominaniem)
void Hebb::learnWithForgetting() {
    int epoch = 0; //epoka
    double FORGET_RATE = 0.05; //wspolczynnik zapominania

```

```

double globalError = 0.; //bład globalny
double localError = 0.; //bład lokalny
double MSE = 0.; //bład sredniokwadratowy
double MAPE = 0.; //bład bezwzględny

double sum = 0.; //przechowuje sume wejście*waga (sumOfInput)

resetSum(); //ustawienie wartosci początkowych dla wektora sumOfInput

generateWeights(); //losowanie wag z przedziału 0;1
readFromFile(); //wczytanie do 4 wektorów wszystkich (225) bitów w celu uczenia

do {
    cout << "Epoka: " << epoch << endl;

    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 (localError == abs(sum - sumOfInput[j])) break;
            localError = abs(sum - sumOfInput[j]);
            globalError = globalError + pow(localError, 2);
        }

        MSE = pow(globalError, 2) / (HOW_MANY_BITS);
        MAPE = (globalError * 10 / HOW_MANY_BITS);
        cout << "MSE: " << MSE << "\tMAPE: " << MAPE << "%"<<endl;
    }

    epoch++;

} while (globalError != 0 && epoch<5000);

cout << endl << "Liczba epok uczenia [Hebb z zapominaniem]: " << epoch << endl;

test();
}

//uczenie regułą Oji
void Hebb::learnOjiRegule() {

    int epoch = 0; //epoka

    double globalError = 0.; //bład globalny
    double localError = 0.; //bład lokalny
    double MSE = 0.; //bład sredniokwadratowy
    double MAPE = 0.; //bład bezwzględny

    double sum = 0.; //przechowuje sume wejście*waga (sumOfInput)

    resetSum(); //ustawienie wartosci początkowych dla wektora sumOfInput

    generateWeights(); //losowanie wag z przedziału 0;1
    readFromFile(); //wczytanie do 4 wektorów wszystkich (225) bitów w celu uczenia

```

```

do {
    cout << "Epoka: " << epoch << endl;

    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 wg
            weights[i][j] = weights[i][j] + (this-
>learningRate*sumOfInput[j] * (emoticons[i][j] - sumOfInput[j] * weights[i][j]));

            if (localError == abs(sum - sumOfInput[j])) break;
            localError = abs(sum - sumOfInput[j]);
            globalError = globalError + pow(localError, 2);
        }

        MSE = pow(globalError, 2) / (HOW_MANY_BITS);
        MAPE = (globalError * 10 / HOW_MANY_BITS);
        cout << "MSE: " << MSE << "\tMAPE: " << MAPE << "%" <<endl;
    }

    epoch++;

} while (globalError != 0 && epoch<5000);

cout << endl << "Liczba epok uczenia [0ji]: " << epoch << endl;

test();
}

//przetestowanie nauczania danej emotikony (do test sluza zmodyfikowane w 20%
emotikony)
void Hebb::test() {
    fstream file;
    file.open("test.txt", ios::app);

    double globalError = 0.;
    double localError = 0.;
    double MSE = 0.;
    double MAPE = 0.;
    double sum = 0.;
    int epoch = 0;

    generateTestInputs();

    cout << endl << "Do testu emotikona SMIECH" << endl;

    do {
        epoch++;
        for (int i = 0; i<HOW_MANY_BITS; i++) {
            sum = sumOfInput[i];
            sumOfInput[i] = (weights[0][i]*inputSmiech80pr[i]);

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

```

```

        MSE = pow(globalError, 2) / (HOW_MANY_BITS);
        MAPE = (globalError * 10 / HOW_MANY_BITS);

        //cout << "Uczenie: MSE: " << MSE << "\tMAPE: " << MAPE << "%" << endl;

        file << "MSE: " << MSE << "\tMAPE: " << MAPE << "%" << endl;
    } while (globalError != 0 && epoch<1);

    file.close();

    compare();
}

//sprawdzenie w ilu procentach siec rozpoznaje dana emotikone
void Hebb::compare() {
    fstream file;
    file.open("compare.txt", ios::app);
    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]++;
    }

    cout << "----- PODOBIENSTWO [ZMODYFIKOWANE W 20%] -----" << endl;
    cout << "Smiech: " << (double(tmp[0]) / 225.0)*100.0 << endl;
    cout << "Usmiech: " << (double(tmp[1]) / 225.0)*100.0 << endl;
    cout << "Serce: " << (double(tmp[2]) / 225.0)*100.0 << endl;
    cout << "Placz: " << (double(tmp[3]) / 225.0)*100.0 << endl;

    file << "Smiech: " << (double(tmp[0]) / 225.0)*100.0 << endl;
    file << "Usmiech: " << (double(tmp[1]) / 225.0)*100.0 << endl;
    file << "Serce: " << (double(tmp[2]) / 225.0)*100.0 << endl;
    file << "Placz: " << (double(tmp[3]) / 225.0)*100.0 << endl;

    file.close();
}

void Hebb::generateTestInputs() {
    srand(time(NULL));

    //przepisanie bitow do danych testujacych
    for (int i = 0; i<HOW_MANY_BITS; i++) {

        inputSmiech80pr[i] = emoticons[0][i];
        inputUsmiech80pr[i] = emoticons[1][i];
        inputSerce80pr[i] = emoticons[2][i];
        inputPlacz80pr[i] = emoticons[3][i];
    }

    int wrongBits = 45; // (1 - 0.8) * 225
    int j;

    //zdeformowanie danych testujacych o 20%
    for (int i = 0; i<wrongBits; i++) {
        j = rand() % 225;
        inputSerce80pr[j] = !inputSerce80pr[j];
    }
}

```

```

        j = rand() % 225;
        inputSmiech80pr[j] = !inputSmiech80pr[j];
        j = rand() % 225;
        inputUsmiech80pr[j] = !inputUsmiech80pr[j];
        j = rand() % 225;
        inputPlacz80pr[j] = !inputPlacz80pr[j];
    }
}

```

„Include.h”

```

#pragma once
#include<ctime>
#include<string>
#include<fstream>
#include<iostream>

#define HOW_MANY_BITS 15*15
#define HOW_MANY_EMOTICONS 4

using namespace std;

```