Krzysztof Sekuła

Sprawozdanie z projektu nr 6

Celem ćwiczenia było poznanie budowy działania sieci Kohonena przy wykorzystaniu reguły Winner Takes Most do odwzorowywania istotnych cech liter alfabetu.

Opis budowy algorytmu:

Sieci Kohonena są przykładem sieci uczących się bez nadzoru. Oznacza to że sieci nie są prezentowane wzorcowe odpowiedzi a jedynie dane wejściowe. Neurony w sieci Kohonena są uporządkowane względem siebie przestrzennie. Często realizowane jest to za pomocą siatki dwuwymiarowej. Uczenie polega na wyznaczeniu zwycięskiego neuronu i modyfikacji jego wektora wag w kierunku prezentowanego wektora wejściowego x. zwycięskim neuronem jest ten neuron którego wagi synaptyczne są najbardziej zbliżone do prezentowanego wektora danych wejściowych. Zatem wektor wag zwycięskiego neuronu wyznacza się za pomocą:

$$d(x, w_w) = \min_{1 \le i \le n} d(x, w_i)$$

Norma d może być dowolną normą. W trakcie uczenia zwycięzca oraz jego sąsiedztwo podlega adaptacji wag zgodnie z regułą Kohonena:

$$w_i(k+1) = w_i(k) + \eta_i(k)[x - w_i(k)]$$

Parametr eta krok uczenia, poważnie malejący w trakcie kolejnych iteracji. W przypadku danych wejściowych o małych wymiarach istnieje potrzeba normalizacji wektorów wejściowych.

Algorytm WTM:

W metodzie tej oprócz wag zwycięskiego neuronu zmianie podlegają wagi jego sąsiadów według reguły:

$$w_i(k+1) = w_i(k) + \eta_i(k)G(i,x)[x - w_i(k)]$$

gdzie G(x, i) jest funkcją sąsiedztwa i jest zdefiniowana jako:

$$G(i,x) = \begin{cases} 1 & dla \quad d(i,w) \leq \lambda \\ 0 & dla \quad d(i,w) > \lambda \end{cases}$$

gdzie d(i, w) jest odległością pomiędzy neuronami w przestrzeni sieci, a lambda jest promieniem sąsiedztwa malejącym do zera w trakcie nauki. Jest to tak zwane sąsiedztwo prostokątne.

Zestawienie wyników:

Jako dane uczące wykorzystałem 20 dużych liter alfabetu o rozmiarach 5x7.

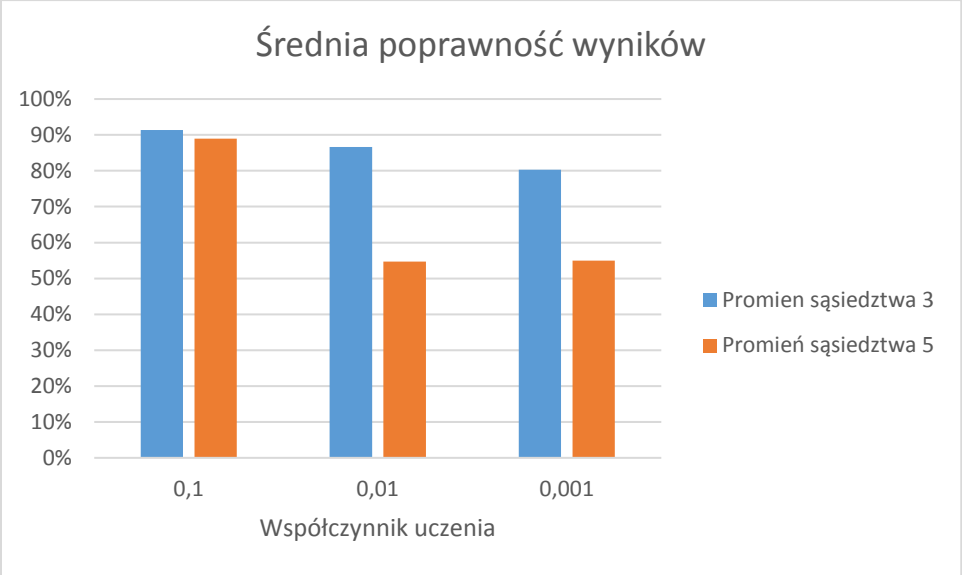
Natomiast jak dane testujące użyłem tych samych liter ale w każdej literze zmieniłem jedną wartość w tablicy.

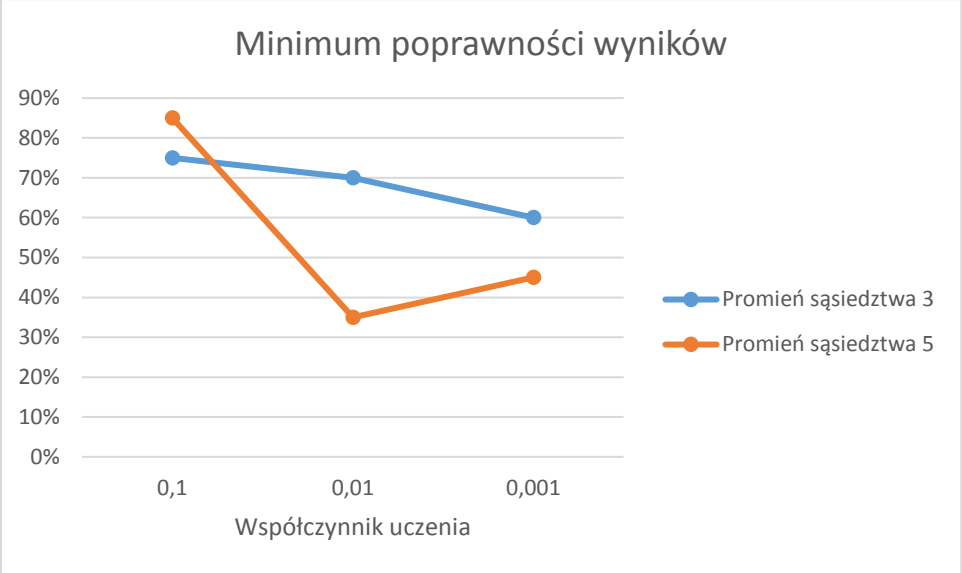
{0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, **0**, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1}//A – dane uczące

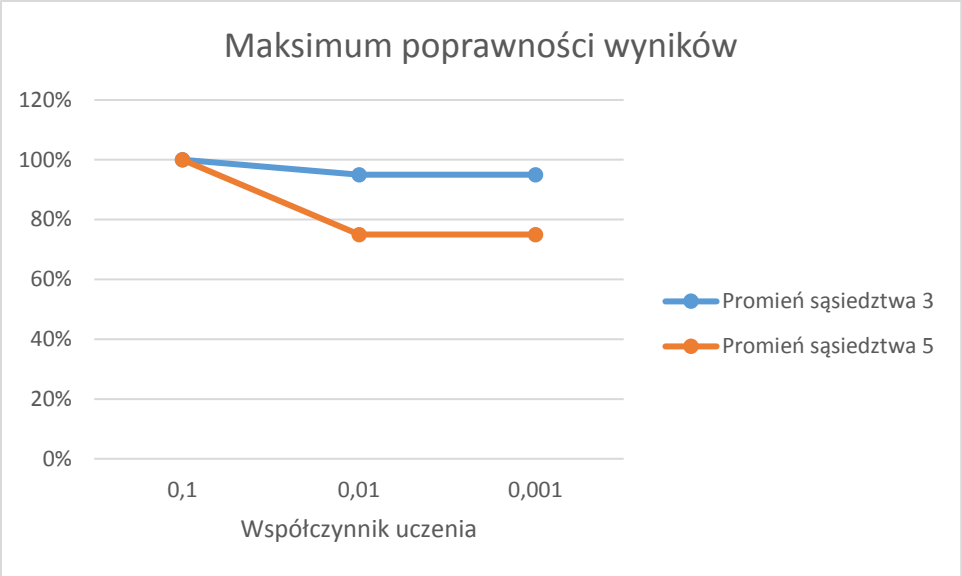
{0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, **1**, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1} //A – dane testujące

Uczenie przeprowadziłem 15 razu dla trzech różnych współczynników uczenia oraz dla dwóch promieni sąsiedztwa.

LP	współczynnik uczenia:	0,1		0,01		0,001	
	promień sąsiedztwa:	3	5	3	5	3	5
1	poprawność:	75%	85%	85%	60%	75%	70%
2	poprawność:	95%	100%	95%	60%	75%	55%
3	poprawność:	80%	85%	95%	45%	70%	45%
4	poprawność:	95%	85%	90%	55%	95%	60%
5	poprawność:	85%	85%	90%	35%	60%	50%
6	poprawność:	95%	100%	80%	75%	90%	45%
7	poprawność:	90%	85%	75%	50%	80%	75%
8	poprawność:	95%	85%	90%	45%	85%	65%
9	poprawność:	100%	85%	85%	45%	85%	55%
10	poprawność:	100%	100%	70%	65%	75%	50%
11	poprawność:	75%	85%	85%	65%	90%	45%
12	poprawność:	100%	85%	95%	60%	95%	45%
13	poprawność:	100%	85%	85%	55%	85%	55%
14	poprawność:	90%	85%	90%	55%	70%	55%
15	poprawność:	95%	100%	90%	50%	75%	55%


Średnia poprawność wyników
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0,1
0,01
0,001
Współczynnik uczenia
Promien sąsiedztwa 3
Promień sąsiedztwa 5


Minimum poprawności wyników
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0,1
0,01
0,001
Współczynnik uczenia
Promień sąsiedztwa 3
Promień sąsiedztwa 5



Wnioski:

Dla każdego testu ilość epok potrzebnych do uczenia wynosiła maksymalną ustawioną liczbę.

Z powyższych wyników możemy zauważyć że dla współczynnika uczenia równego 0.1 poprawność wyników była największa. Natomiast wraz ze zmniejszeniem współczynnika uczenia procent błędnych wyników wzrósł.

Z otrzymanych danych możemy również wnioskować że dla mniejszego promienia sąsiedztwa wyniki były dokładniejsze. Większy promień powodował że zbyt wielka liczba sąsiednich neuronów podlegała modyfikacji. Jedna trzeba pamiętać aby promień nie był zbyt mały. Aby uzyskać zadowalający wynik należ odpowiednio dopasować współczynnik uczenia jak i również promień sąsiedztwa. W tym przypadku najlepszym dopasowaniem okazał się promień 5 oraz współczynnik 0.1.

Z otrzymanych wyników możemy zauważyć że często pojawiały się błędy. Może to być spowodowane za małą ilością danych uczących. Przy zastosowaniu odpowiednio dużego zbiory danych uczących wyniki mogłyby być znacznie lepsze.

Listing kodu:

```
public class WTM {

    private int noi;
    private double[] w;

    public WTM ( int numbers_of_inputs ) {
        noi = numbers_of_inputs;
        w = new double[noi];

        for ( int i = 0; i < noi; i++ )
            w[i] = new Random().nextDouble();
    }


    public void learn ( double[] x, double lr ) {

        for ( int i = 0; i < noi; i++ )
            w[i] += lr * ( x[i] - w[i] );
    }


    public double[] getW () {
        return w;
    }
}
```

```
public class Main {

    private static double learningRate = 0.001;          //współczynnik
uczenia się
    private static int nots = 20;      //liczba danych testujacych
    private static int noi = 35;             //ilość wejść
    private static int nols = 20;      //liczba danych uczących
    private static int ll = 100;                //maksymalna ilosc epok
uczenia
    private static int non = 2000;        //liczba neuronów
    private static double nr = 5.0;        //wartość promienia sąsiedztwa

    public static void main ( String[] args ) {

            WTM[] koh = new WTM[non];
        for (int i = 0; i < non; i++ )
            koh[i] = new WTM(noi);

        int ages = learn( koh );

        int[] winnerLearn = new int[nols], winnerTest = new int[nots];
        int percent = 0;


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

            winnerLearn[i] = getWinner( koh, Letters.lettersLearn[i] );


        for (int i = 0; i < nols; i++ )
            winnerTest[i] = getWinner( koh, Letters.lettersTest[i] );
```

```
        for (int i = 0; i < nots; i++ )
            if ( winnerLearn[i] == winnerTest[i] )
                percent++;

        System.out.println( "Ilość epok = " + ages );
        System.out.println( "Poprawność testowania = " + ( ( percent * 100
) / nots) + "%" );
    }


    private static int learn ( WTM[] kohonens ) {

        int counter = 0;
        int winner;

        int[] winners = new int[nols];
        for (int i = 0; i < nols; i++ )
            winners[i] = - 1;

        while ( ! isUnique( winners ) ) {

            for (int i = 0; i < nols; i++ ) {
                winner = getWinner( kohonens, Letters.lettersLearn[i] );
                kohonens[winner].learn( Letters.lettersLearn[i],
learningRate );

                for (int j = 0; j < non; j++ )
                    if ( j != winner )
                        if ( distanceBetweenVectors(
kohonens[winner].getW(), kohonens[j].getW() ) <= nr)
                            kohonens[j].learn( Letters.lettersLearn[i],
learningRate );
            }

            for (int i = 0; i < nols; i++ )
                winners[i] = getWinner( kohonens, Letters.lettersLearn[i]
);

            if ( ++ counter == ll)
                break;
        }

        return counter;
    }


    private static boolean isUnique ( int[] winners ) {

        for (int i = 0; i < nols; i++ )
            for (int j = i; j < nols; j++ )
                if ( i != j )
                    if ( winners[i] == winners[j] )
                        return false;

        return true;
    }


    private static int getWinner ( WTM[] kohonens, double[] vector ) {

        int winner = 0;
```

```
        double minDistance = distanceBetweenVectors( kohonens[0].getW(),
vector );

        for (int i = 0; i < non; i++ ) {
            if ( distanceBetweenVectors( kohonens[i].getW(), vector ) <
minDistance ) {
                winner = i;
                minDistance = distanceBetweenVectors( kohonens[i].getW(),
vector );
            }
        }

        return winner;
    }


    private static double distanceBetweenVectors ( double[] vector1,
double[] vector2 ) {

        double suma = 0.0;

        for ( int i = 0; i < vector1.length; i++ )
            suma += Math.abs( vector1[i] - vector2[i] ); //miara Manhattan

        return Math.sqrt( suma );
    }

}
```

Źródła:

http://www.michalbereta.pl/dydaktyka/WdoSI/lab_neuronowe_II/Sieci_Neuronowe_2%20Sieci%20Kohonena.pdf

http://www.ai.c-labtech.net/sn/litery.html