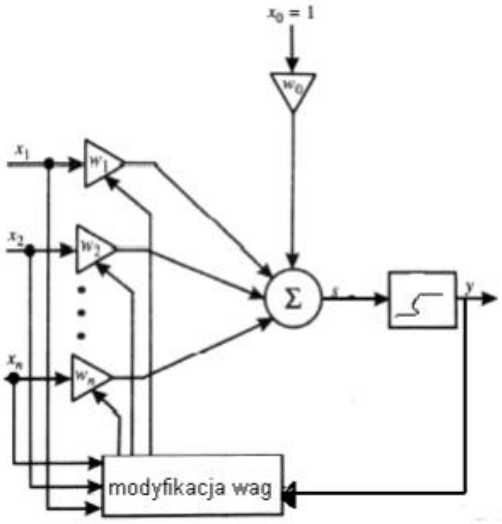
Krzysztof Sekuła
Sprawozdanie z projektu nr 4

Celem ćwiczenia było poznanie działania reguły Hebba dla sieci jednowarstwowej na przykładzie grupowania liter alfabetu.

Opis budowy algorytmu:
Model neuronu Hebba ma 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 podczas badań działania komórek nerwowych, że połączenie pomiędzy dwiema komórkami jest wzmacniane, jeśli w tym samym czasie obie komórki są aktywne.



Ogólna reguła Hebba mówi, że zmiany wag powinny odbywać się według reguły:

$\Delta w(k) = F(x(k), y(k))$

czyli że przyrost wag $\Delta w(k)$ powinien zależeć zarówno od wielkości wzorca presynaptycznego x(k) jak i od wytworzonego wzorca postsynaptycznego y(k). Do wyrażenia funkcji F służy tzw. Prosta reguła Hebba która uproszcza funkcję F do funkcji iloczynowej:

$\Delta w(k) = \eta \cdot x(k) \cdot y(k)$

Przy użyciu funkcji iloczynowej zastosowanej w prostej regule Hebba wagi wzrastają wykładniczo wraz z postępem uczenia i wielokrotną prezentacją tego samego wzorca, co jest zjawiskiem bardzo niepożądanym. Aby temu zapobiec, wprowadzono reguły ograniczające przyrosty wag. Jedną tak z nich jest reguła zapominania.

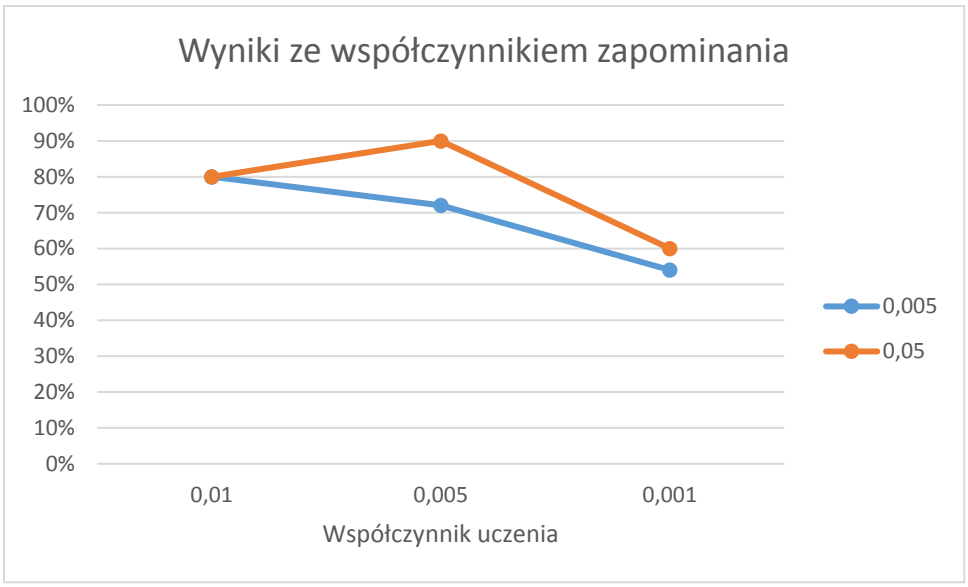
Zestawienie wyników:

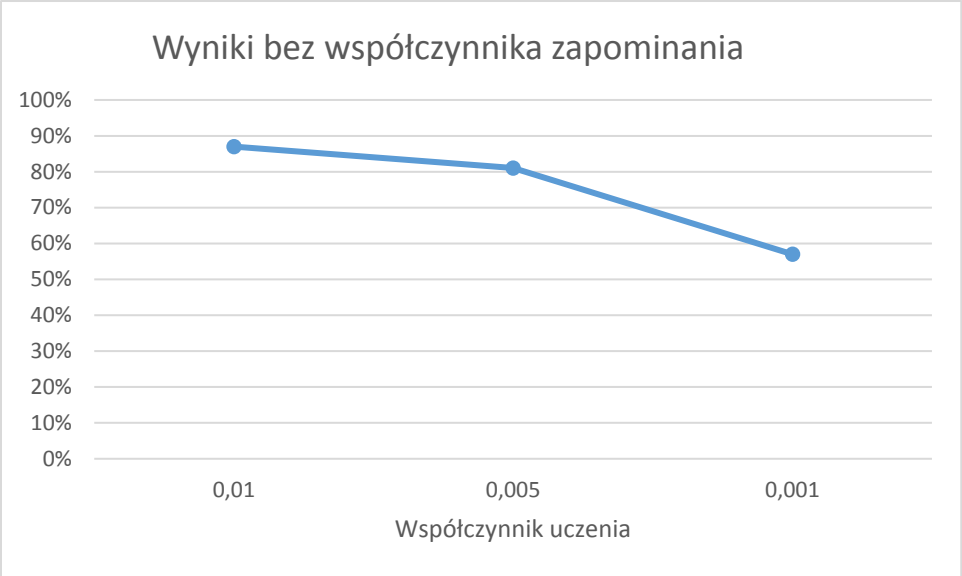
Jako dane uczące wykorzystałem 20 dużych liter alfabetu o rozmiarach 5x7. Natomiast jak dane testujące użyłem tych samych liter ale w każdej literze zmieniłem jedną wartość w tablicy.
{0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, **0**, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1}//A – dane 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 po 10 razy dla uczenia z zapominaniem i bez, każdy dla trzech różnych współczynników uczenia. W wersji z zapominaniem zastosowałem dwa różne współczynniki zapominania.

Wyniki ze współczynnikiem zapominania							
LP	Współczynnik uczenia	0,01	0,005	0,001	0,01	0,005	0,001
	Współczynnik zapominania	0,005	0,005	0,005	0,05	0,05	0,05
1	poprawność:	100%	60%	40%	80%	100%	60%
2	poprawność:	70%	80%	30%	80%	100%	50%
3	poprawność:	70%	60%	70%	60%	90%	60%
4	poprawność:	70%	90%	60%	80%	60%	70%
5	poprawność:	80%	80%	70%	100%	100%	50%
6	poprawność:	70%	60%	60%	90%	100%	50%
7	poprawność:	70%	70%	60%	80%	100%	60%
8	poprawność:	80%	50%	50%	80%	90%	70%
9	poprawność:	100%	70%	40%	80%	70%	50%
10	poprawność:	90%	100%	60%	70%	90%	80%

Wyniki bez współczynnika zapominania				
LP	Współczynnik uczenia	0,01	0,005	0,001
1	poprawność:	100%	60%	50%
2	poprawność:	90%	80%	40%
3	poprawność:	80%	70%	60%
4	poprawność:	100%	90%	50%
5	poprawność:	100%	80%	70%
6	poprawność:	80%	80%	50%
7	poprawność:	90%	100%	90%
8	poprawność:	80%	100%	60%
9	poprawność:	70%	90%	50%
10	poprawność:	80%	60%	50%





Wnioski:

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

W powyższych wynikach możemy zauważyć współczynnik uczenia oraz współczynnik zapominania wpływał na poprawność uczenia.

Kiedy współczynnik uczenie malał spadała również poprawność wyników.

W przypadku kiedy był brany pod uwagę współczynnik zapominania, najlepsze wyniki otrzymaliśmy dla współczynnika uczenia 0,005 oraz współczynnika zapominania 0,05. W innych przypadkach wyniki nie były już aż tak dobre.

Z otrzymanych wyników możemy wnioskować iż odpowiedni dobór współczynników jest bardzo ważny aby otrzymać poprawne dane.

Należy pamiętać że zbyt duży współczynnik zapomnienia może niekorzystnie wpłynąć na dane końcowe, ponieważ sieć w trakcie nauki zbyt szybko zapomni czego się nauczyła.

Listing kodu:

```
public class Letters {


    public static double[][] lettersLearn = {
            {1, 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
            {1, 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
            {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},//B
            {1, 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},//C
            {1, 1, 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, 0, 0, 1, 1, 1, 1, 1, 0},//D
            {1, 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, 0, 0, 0, 0},//E
            {1, 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, 0, 0, 0, 0},//F
            {1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1,
1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0},//G
            {1, 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},//H
            {1, 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},//I
            {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, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0},//J
            {1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1},//K
            {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, 0, 1, 1, 1, 1, 1},//L
            {1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1,
1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1},//M
            {1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1},//N
            {1, 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, 0, 0, 1, 0, 1, 1, 1, 0},//O
            {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, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0},//P
            {1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1,
1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1},//Q
            {1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0,
1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1},//R
            {1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0,
0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0},//S
            {1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0},//T
    };


    public static double[][] lettersTest = {
            {1, 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
            {1, 1, 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, 1, 1, 1, 0},//B
            {1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0},//C
            {1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1,
1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0},//D
            {1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 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},//E
            {1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0,
1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0},//F
            {1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1,
1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0},//G
            {1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 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},//H
            {1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 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},//I
            {1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0},//J
            {1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1},//K
            {1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1},//L
            {1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1,
1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1},//M
            {1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1},//N
            {1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1,
1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0},//O
            {1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0,
1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0},//P
            {1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1,
1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1},//Q
            {1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0,
1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1},//R
            {1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0,
0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0},//S
            {1, 1, 1, 1, 1, 1, 0, 0, 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},//T
    };

    public static String[] letter = { "A", "B", "C", "D", "E", "F", "G",
"H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T"};


}
```

```
public class Hebb {

    private int noi;                                //ilość wejść
    private double[] w;                             //wagi
    public static boolean forget = false;      //współczynnik zapominania
    public static boolean not_forget = true;    //bez współczynnika
zapominania

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

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

```
    private double active ( double y_p ) {
        return ( 1.0 / ( 1 + Math.pow( Math.E, - y_p ) ) );
    }

    private void normalizeWeights () {
        double dl = 0.0;
        for ( int i = 0; i < w.length; i++ )
            dl += Math.pow( w[i], 2 );

        dl = Math.sqrt( dl );

        for ( int i = 0; i < w.length; i++ )
            if ( w[i] > 0 && dl != 0 )
                w[i] = w[i] / dl;
    }

    private double sumator ( double[] x ) {
        double y_p = 0.0;
        for ( int i = 0; i < noi; i++ )
            y_p += x[i] * w[i];

        return y_p;
    }




    public double test ( double[] x ) {
        return active( sumator( x ) );
    }


    public double learnUnsupervised ( double[] x, double lr, double fr,
boolean version ) {
        double y_p = active( sumator( x ) );


        for ( int i = 0; i < noi; i++ )
            if ( version ) w[i] = ( 1 - fr ) * w[i] + lr * x[i] * y_p;
            else w[i] += lr * x[i] * y_p;

        normalizeWeights();

        return active( sumator( x ) );
    }

}
```

```
 public class Main {

    static int noi = 36;               //ilość wejść
    static double learningRate = 0.001;               //współczynnik
uczenia się
    static double forget = 0.0001; //współczynnik zapominania
    static int nols = 10;                 //liczba danych uczacych
    static int non = 11;                  //liczba neuronów

    public static void main ( String[] args ) {
```

```
        int winner;
        Hebb[] hebbs = new Hebb[non];
        for (int i = 0; i < non; i++ )
            hebbs[i] = new Hebb(noi);

        int ages = learn( hebbs );

        System.out.println( "PO UCZENIU" );
        for (int i = 0; i < nols; i++ ) {
            winner = testHebb( hebbs, Letters.lettersLearn[i] );
            System.out.println( "Litera " + Letters.letter[i] + " - neuron
" + winner );
        }

        System.out.println( "\nTESTOWANIE" );
        for (int i = 0; i < nols; i++ ) {
            winner = testHebb( hebbs, Letters.lettersTest[i] );
            System.out.println( "Litera " + Letters.letter[i] + " - neuron
" + winner );
        }

        System.out.println( "\nIlość epok = " + ages );

    }


    public static int learn ( Hebb[] hebbs ) {

        int counter = 0;
        int limit = 1000;
        int[] winners = new int[non];
        for (int i = 0; i < non; i++ )
            winners[i] = - 1;

        while ( ! isUnique( winners ) ) {

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

                for (int k = 0; k < nols; k++ )
                    hebbs[j].learnUnsupervised( Letters.lettersLearn[k],
learningRate, forget, Hebb.forget);


                for (int l = 0; l < nols; l++ )
                    winners[l] = testHebb( hebbs, Letters.lettersLearn[l]
);
            }

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

        return counter;
    }


    public static boolean isUnique ( int[] winners ) {

        for (int i = 0; i < non; i++ )
            for (int j = 0; j < non; j++ )
                if ( i != j )
```

```
                    if ( winners[i] == winners[j] )
                        return false;

        return true;
    }


    public static int testHebb ( Hebb[] hebbs, double[] letters ) {

        double max = hebbs[0].test( letters );
        int winner = 0;

        for (int i = 1; i < non; i++ ) {
            if ( hebbs[i].test( letters ) > max ) {
                max = hebbs[i].test( letters );
                winner = i;
            }
        }

        return winner;
    }

}
```

Źródła:

https://www.ii.uni.wroc.pl/~aba/teach/NN/w6pca.pdf

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

http://nanophysics.pl/teaching/sieci_neuronowe/sieci_neuronowe_05.pdf