Paweł Nowak,
Scenariusz 4

Temat ćwiczenia:

Uczenie sieci regułą Hebba.

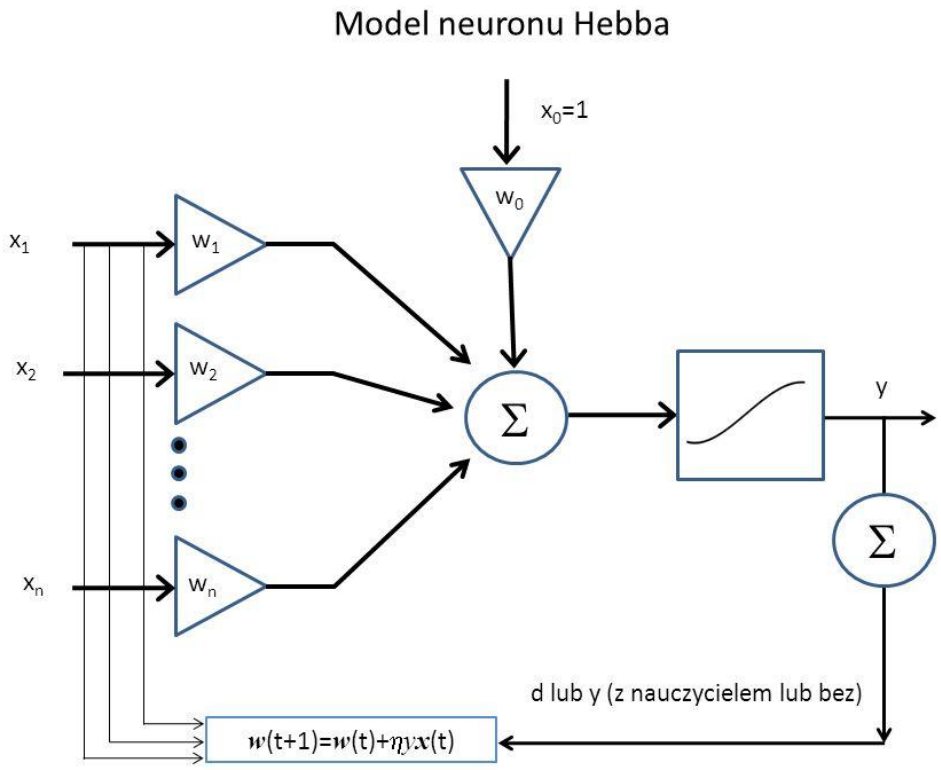
Cel ćwiczenia:

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

Regula Hebba

Jest to jedna z najpopularniejszych metod samouczenia sieci neuronowych. Polega ona na tym, że sieci pokazuje się kolejne przykłady sygnałów wejściowych, nie podając żadnych informacji o tym, co z tymi sygnałami należy zrobić. Sieć obserwuje otoczenie i odbiera różne sygnały, nikt nie określa jednak, jakie znaczenie mają pokazujące się obiekty i jakie są pomiędzy nimi zależności. Sieć na podstawie obserwacji występujących sygnałów stopniowo sama odkrywa, jakie jest ich znaczenie i również sama ustala zachodzące między sygnałami zależności.

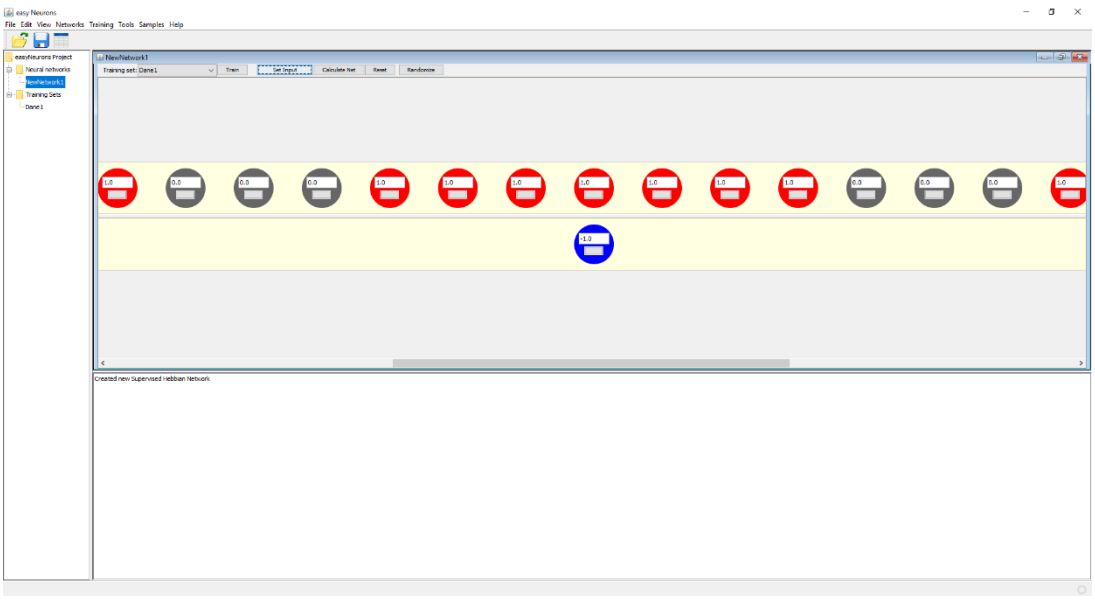
Po podaniu do sieci neuronowej każdego kolejnego zestawu sygnałów wejściowych tworzy się w tej sieci pewien rozkład sygnałów wyjściowych - niektóre neurony sieci są pobudzone bardzo silnie, inne słabiej, a jeszcze inne mają sygnały wyjściowe wręcz ujemne. Interpretacja tych zachowań może być taka, że niektóre neurony „rozpoznają" podawane sygnały jako „własne" (czyli takie, które są skłonne akceptować), inne traktują je „obojętnie", zaś jeszcze u innych neuronów wzbudzają one wręcz „awersję". Po ustaleniu się sygnałów wyjściowych wszystkich neuronów w całej sieci - wszystkie wagi wszystkich neuronów są zmieniane, przy czym wielkość odpowiedniej zmiany wyznaczana jest na podstawie iloczynu sygnału wejściowego, wchodzącego na dane wejście (to którego wagę zmieniamy) i sygnału wyjściowego produkowanego przez neuron, w którym modyfikujemy wagi. Łatwo zauważyć, że jest to właśnie realizacja postulatu Hebba - w efekcie opisanego wyżej algorytmu połączenia między źródłami silnych sygnałów i neuronami które na nie silnie reagują są wzmacniane.



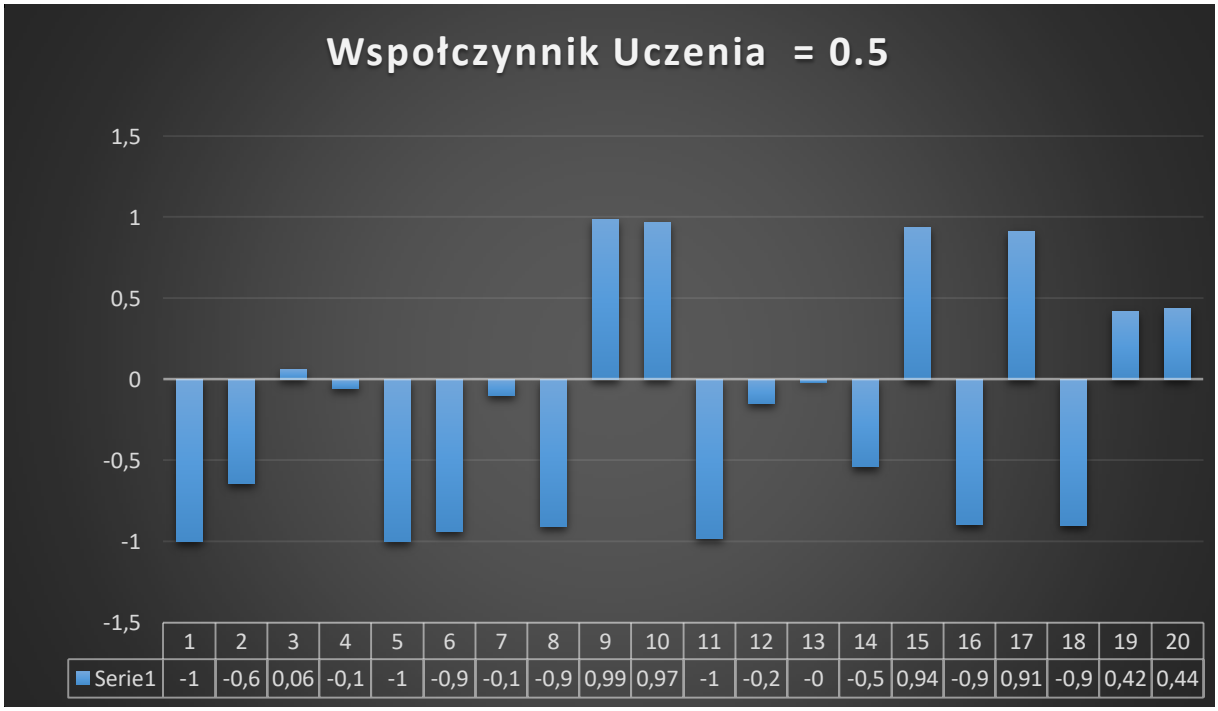
Do rozwiązania tego zagadnienia użyłem programu Easy Neurons.

Przyjąłem ciąg znaków dla pojedynczej litery równy 35. Po nauczeniu sieci złozonej z 35 input'ów oraz 1 output'u sprawdzałem wyniki dla każdej z 20 liter podając ciąg znaków pojedynczej litery.

Wygląd sieci:

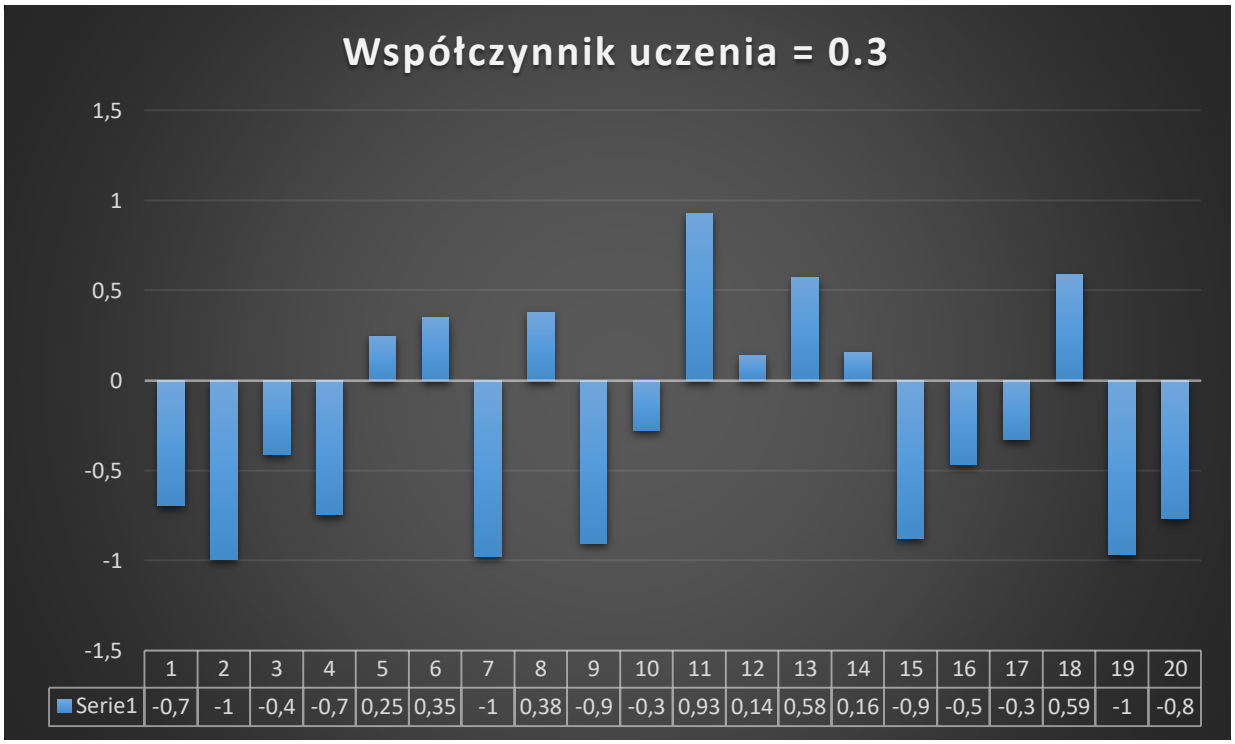


Poniższe wykresy prezentują wyniki testowania programów dla różnych współczynników uczenia. Dla każdego wariantu pętla przy trenowaniu została ograniczona do 100 000 iteracji.

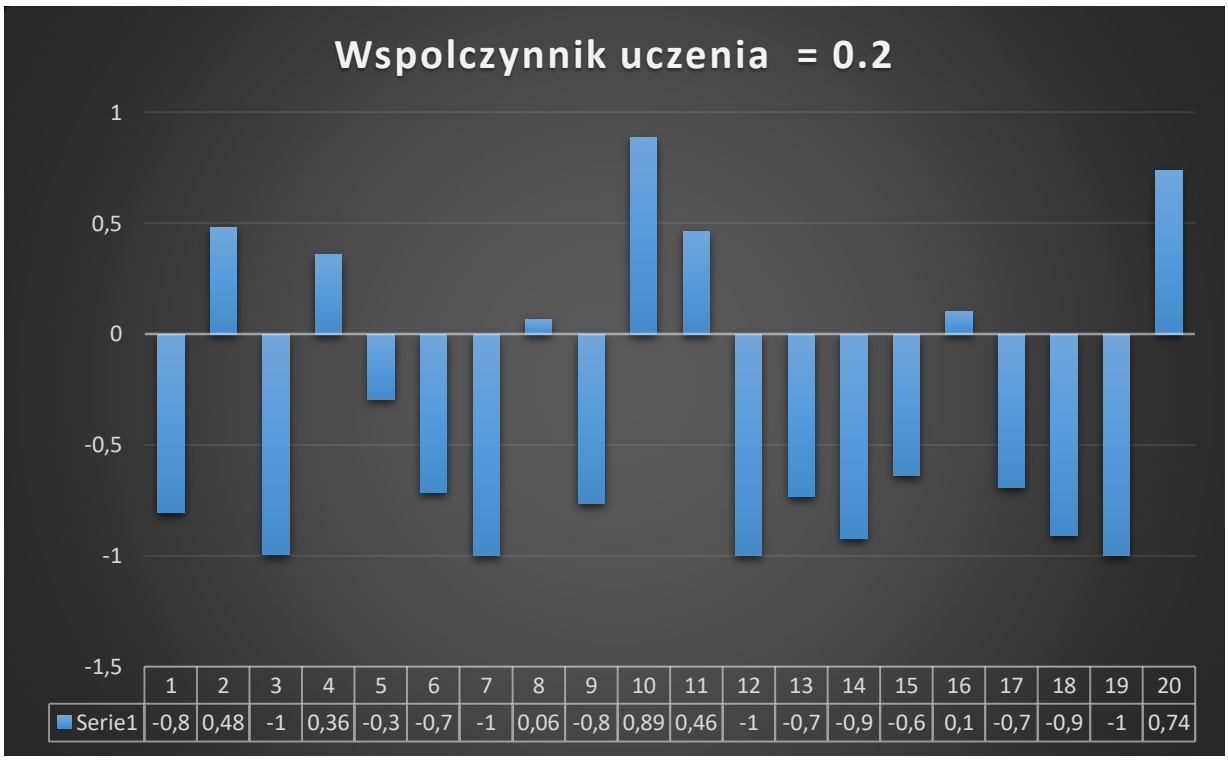


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Serie1	-1	-0,6	0,06	-0,1	-1	-0,9	-0,1	-0,9	0,99	0,97	-1	-0,2	-0	-0,5	0,94	-0,9	0,91	-0,9	0,42	0,44

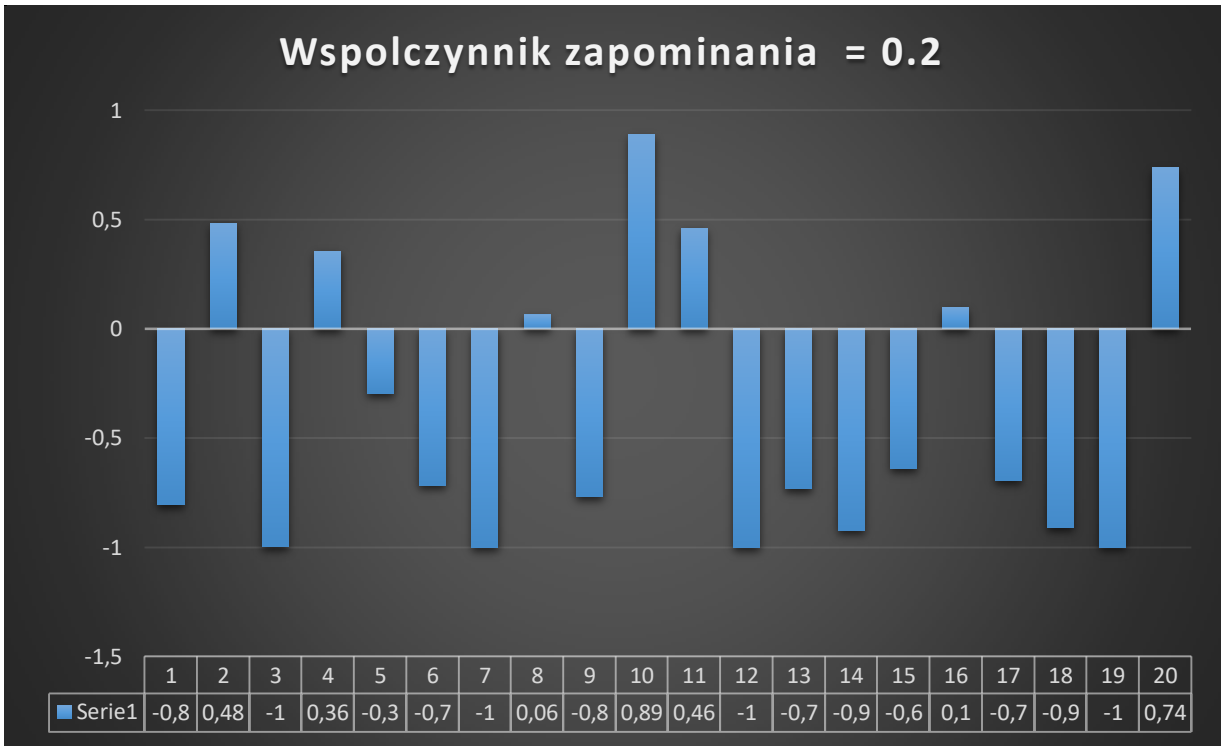
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Serie1	0,87	0,98	0,91	0,98	0,66	0,4	0,98	0,91	0,69	0,87	0,4	0,9	0,85	0,87	0,64	0,41	0,95	0,03	0,89	-0,6



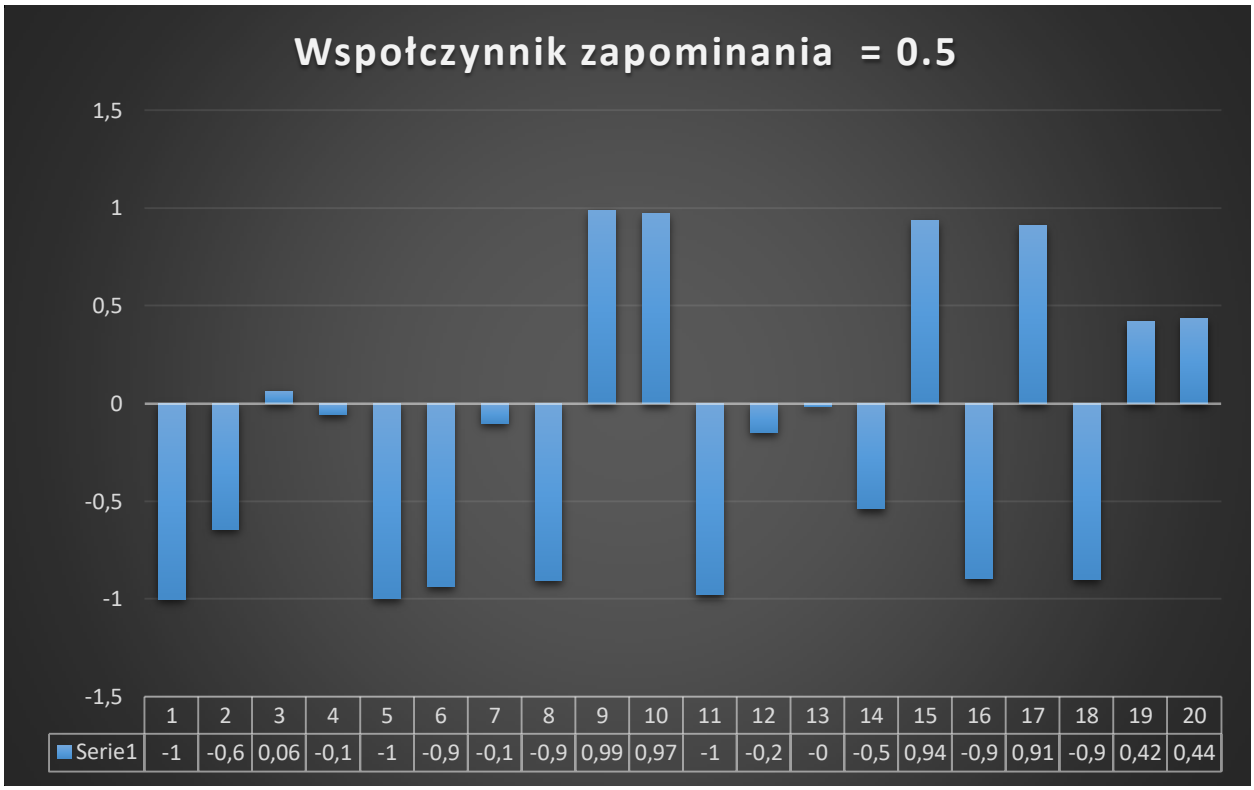
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Serie1	-0,7	-1	-0,4	-0,7	0,25	0,35	-1	0,38	-0,9	-0,3	0,93	0,14	0,58	0,16	-0,9	-0,5	-0,3	0,59	-1	-0,8



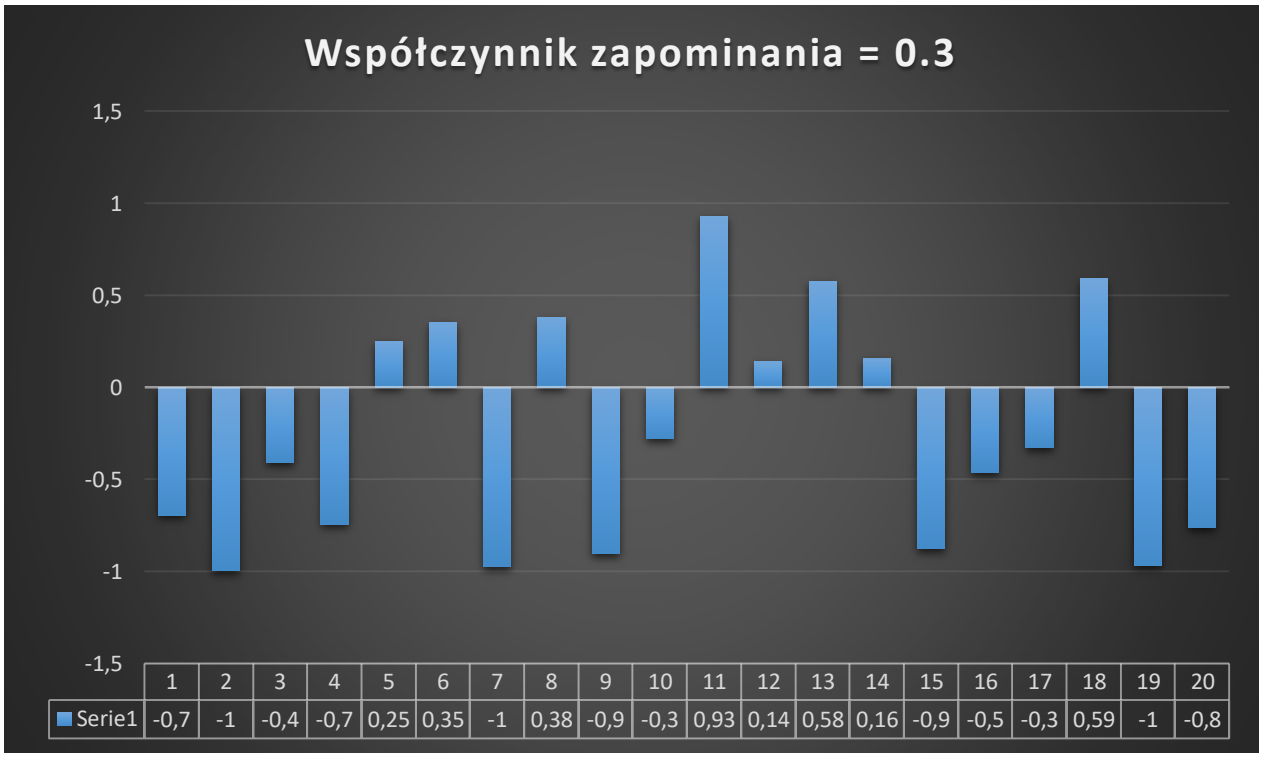
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Serie1	-0,8	0,48	-1	0,36	-0,3	-0,7	-1	0,06	-0,8	0,89	0,46	-1	-0,7	-0,9	-0,6	0,1	-0,7	-0,9	-1	0,74


Wspolczynnik zapominania = 0.2
1
0,5
0
-0,5
-1
-1,5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Serie1 -0,8 0,48 -1 0,36 -0,3 -0,7 -1 0,06 -0,8 0,89 0,46 -1 -0,7 -0,9 -0,6 0,1 -0,7 -0,9 -1 0,74


Współczynnik zapomiania = 0.7
1,2
1
0,8
0,6
0,4
0,2
0
-0,2
-0,4
-0,6
-0,8
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Serie1 0,87 0,98 0,91 0,98 0,66 0,4 0,98 0,91 0,69 0,87 0,4 0,9 0,85 0,87 0,64 0,41 0,95 0,03 0,89 -0,6



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Serie1	-1	-0,6	0,06	-0,1	-1	-0,9	-0,1	-0,9	0,99	0,97	-1	-0,2	-0	-0,5	0,94	-0,9	0,91	-0,9	0,42	0,44



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Serie1	-0,7	-1	-0,4	-0,7	0,25	0,35	-1	0,38	-0,9	-0,3	0,93	0,14	0,58	0,16	-0,9	-0,5	-0,3	0,59	-1	-0,8

WNIOSKI :

Proces samouczenia ma niestety wady. W porównaniu z procesem uczenia z nauczycielem samouczenie jest zwykle znacznie powolniejsze. Co więcej bez nauczyciela nie można z góry określić, który neuron wyspecjalizuje się w rozpoznawania której klasy sygnałów. Stanowi to pewną trudność przy wykorzystywaniu i interpretacji wyników pracy sieci. Co więcej - nie można określić, czy sieć uczona w ten sposób nauczy się wszystkich prezentowanych jej wzorców. Dlatego sieć przeznaczona do samouczenia musi być większa niż sieć wykonująca to samo zadanie, ale trenowana w sposób klasyczny, z udziałem nauczyciela. - Szacunkowo sieć powinna mieć co najmniej trzykrotnie więcej elementów warstwy wyjściowej niż wynosi oczekiwana liczba różnych wzorów, które sieć ma rozpoznawać. Bardzo istotną kwestią jest wybór początkowych wartości wag neuronów sieci przeznaczonej do samouczenia. Wartości te mają bardzo silny wpływ na ostateczne zachowanie sieci, ponieważ proces uczenia jedynie pogłębia i doskonali pewne tendencje istniejące w sieci od samego początku, przeto od jakości tych początkowych, „wrodzonych" właściwości sieci silnie zależy, do czego sieć dojdzie na końcu procesu uczenia. Nie wiedząc z góry, jakiego zadania sieć powinna się uczyć, trudno wprowadzać jakikolwiek zdeterminowany mechanizm nadawania początkowych wartości wag, jednak pozostawienie wszystkiego wyłącznie mechanizmom losowym może powodować, że sieć (zwłaszcza mała) może nie zdołać wystarczająco zróżnicować swego działania w początkowym okresie procesu uczenia i wszelkie późniejsze wysiłki, by znaleźć w strukturze sieci reprezentację dla wszystkich występujących w wejściowych sygnałach klas, mogą okazać się daremne.

KOD PROGRAMU:

```
"Import
funkcji
losujacej
Pythona"
            from random import uniform
            import math
            class Hebb:
                """Klasa HEBB"""
                def __init__(self, inputs):
                    "Konstruktor"
                    self.inputs = inputs
                    self.weights = []
                    for i in range(0, inputs):
                        self.weights.append(uniform(0, 1))
                    self.normalize_weights()
                @staticmethod
                def activation(y_p):
                    "Funkcja aktywacji"
```

```
                return (1.0 / (1 + math.pow(math.e, - y_p)))
            def sum(self, vector):
                "Sumator"
                y_p = 0
                for i in range(0, self.inputs):
                    y_p += vector[i] * self.weights[i]
                return y_p
            def learn_without_supervising(self, vector, learning_rate,
        forgetting_rate, is_forgetting):
                "Nauka bez nauczyciela"
                y_p = self.activation(self.sum(vector))
                for i in range(0, self.inputs):
                    if is_forgetting:
                        self.weights[i] = (1 - forgetting_rate) * self.weights[i] +
        learning_rate * vector[i] * y_p
                    else:
                        self.weights[i] += learning_rate * vector[i] * y_p
                self.normalize_weights()
                return self.activation(self.sum(vector))
            def test(self, vector):
                "Funkcja testujaca i zwracajaca wynik z neuronu"
                return self.activation(self.sum(vector))
            def normalize_weights(self):
                "Funkcja normalizujaca wagi"
                dl = 0
                for i in range(0, self.inputs):
                    dl += math.pow(self.weights[i], 2)
                dl = math.sqrt(dl)
                for i in range(0, self.inputs):
                    if self.weights[i] > 0 and dl != 0:
                        self.weights[i] = self.weights[i] / dl

##
pierwsza
wartość
to BIAS
    emoji = [
                    [1,
                     0, 0, 1, 1, 1, 1, 1, 0, 0,
                     0, 1, 0, 0, 0, 0, 0, 1, 0,
                     1, 0, 1, 1, 0, 1, 1, 0, 1,
                     1, 0, 1, 1, 0, 1, 1, 0, 1,
                     1, 0, 0, 0, 0, 0, 0, 0, 1,
                     1, 0, 1, 0, 0, 0, 1, 0, 1,
                     1, 0, 0, 1, 1, 1, 0, 0, 1,
                     0, 1, 0, 0, 0, 0, 0, 1, 0,
```

```
             0, 0, 1, 1, 1, 1, 1, 0, 0
            ],
            [1,
             0, 0, 1, 1, 1, 1, 1, 0, 0,
             0, 1, 0, 0, 0, 0, 0, 1, 0,
             1, 0, 1, 1, 0, 1, 1, 0, 1,
             1, 0, 1, 1, 0, 1, 1, 0, 1,
             1, 0, 0, 0, 0, 0, 0, 0, 1,
             1, 0, 0, 1, 1, 1, 0, 0, 1,
             1, 0, 1, 0, 0, 0, 1, 0, 1,
             0, 1, 0, 0, 0, 0, 0, 1, 0,
             0, 0, 1, 1, 1, 1, 1, 0, 0
             ],
            [1,
             0, 0, 1, 1, 1, 1, 1, 0, 0,
             0, 1, 0, 0, 0, 0, 0, 1, 0,
             1, 0, 1, 1, 0, 1, 1, 0, 1,
             1, 0, 1, 1, 0, 1, 1, 0, 1,
             1, 0, 0, 0, 0, 0, 0, 0, 1,
             1, 0, 1, 1, 1, 1, 1, 0, 1,
             1, 0, 0, 0, 0, 1, 1, 0, 1,
             0, 1, 0, 0, 0, 0, 0, 1, 0,
             0, 0, 1, 1, 1, 1, 1, 0, 0
            ],
            [1,
             0, 0, 1, 1, 1, 1, 1, 0, 0,
             0, 1, 0, 0, 0, 0, 0, 1, 0,
             1, 0, 1, 1, 0, 1, 1, 0, 1,
             1, 0, 1, 1, 0, 1, 1, 0, 1,
             1, 0, 0, 0, 0, 0, 0, 0, 1,
             1, 0, 1, 1, 1, 1, 1, 0, 1,
             1, 0, 0, 1, 1, 1, 0, 0, 1,
             0, 1, 0, 0, 0, 0, 0, 1, 0,
             0, 0, 1, 1, 1, 1, 1, 0, 0
            ]
        ]
confused_emoji = [
                    [1,
                     0, 0, 1, 1, 1, 1,
1, 0, 0,
                     0, 1, 0, 0, 0, 0,
0, 1, 0,
                     1, 0, 1, 1, 0, 1,
1, 0, 1,
                     1, 0, 1, 1, 0, 1,
1, 0, 1,
```

```
                    1, 0, 0, 0, 0, 0,
0, 0, 1,
                    1, 0, 1, 0, 0, 0,
1, 0, 1,
                    1, 0, 0, 1, 1, 1,
0, 0, 1,
                    0, 1, 0, 0, 0, 0,
1, 1, 0,
                    0, 0, 1, 1, 1, 1,
1, 0, 0
                  ],
                  [1,
                    0, 0, 1, 1, 1, 1,
1, 0, 0,
                    0, 1, 0, 0, 0, 1,
0, 1, 0,
                    1, 0, 1, 1, 0, 1,
1, 0, 1,
                    1, 0, 1, 1, 0, 1,
1, 0, 1,
                    1, 0, 0, 0, 0, 0,
0, 0, 1,
                    1, 0, 0, 1, 1, 1,
0, 0, 1,
                    1, 0, 1, 0, 0, 0,
1, 0, 1,
                    0, 1, 0, 0, 0, 0,
0, 1, 0,
                    0, 0, 1, 1, 1, 1,
1, 0, 0
                  ],
                  [1,
                    0, 0, 1, 1, 1, 1,
1, 0, 0,
                    0, 1, 0, 0, 0, 0,
0, 1, 0,
                    1, 0, 1, 1, 0, 1,
1, 0, 1,
                    1, 0, 1, 1, 0, 1,
1, 0, 1,
                    1, 0, 0, 0, 0, 0,
0, 0, 1,
                    1, 0, 1, 1, 1, 1,
1, 1, 1,
                    1, 0, 0, 0, 0, 1,
1, 0, 1,
```

```
                        0, 1, 0, 0, 0, 0,
    0, 1, 0,
                        0, 0, 1, 1, 1, 1,
    1, 0, 0
                       ],
                       [1,
                        0, 0, 1, 1, 1, 1,
    1, 0, 0,
                        0, 1, 0, 0, 0, 0,
    0, 1, 0,
                        1, 0, 1, 1, 0, 1,
    1, 0, 1,
                        1, 1, 1, 1, 0, 1,
    1, 0, 1,
                        1, 0, 0, 0, 0, 0,
    0, 0, 1,
                        1, 0, 1, 1, 1, 1,
    1, 0, 1,
                        1, 0, 0, 1, 1, 1,
    0, 0, 1,
                        0, 1, 0, 0, 0, 0,
    0, 1, 0,
                        0, 0, 1, 1, 1, 1,
    1, 0, 0
                       ]
                   ]
    emoji_type = [":)", ":(", ":P", ":D"]


from
Emoji
import
emoji
        from Emoji import confused_emoji
        from Emoji import emoji_type
        from Hebb import Hebb
        HEBB_FORGETTING = True
        HEBB_WITHOUT_FORGETTING = False
        def learn(hebbs):
            "Funkcja uczenia"
            era = 0
            winners = []
            for i in range(0, NEURONS):
                winners.append(-1)
            while unique(winners) != True:
                for i in range(0, NEURONS):
```

```
            for j in range(0, EMOJI):
                hebbs[i].learn_without_supervising(emoji[j], LEARNING_RATE,
FORGETTING_RATE, HEBB_FORGETTING)
            for j in range(0, EMOJI):
                winners[j] = test_hebb(hebbs, emoji[j])
        era += 1
        if era == LIMIT:
            break
    return era
def unique(winners):
    "Funkcja sprawdza czy elementy w tablicy sa unikalne, pomoc w nauce"
    for i in range(0, NEURONS):
        for j in range(0, NEURONS):
            if i != j:
                if winners[i] == winners[j]:
                    return False
    return True
def test_hebb(hebbs, emoji):
    "Funkcja zwraca wartosc zwyciezkiego neuronu dla emotikony"
    max = hebbs[0].test(emoji)
    winner = 0
    for i in range(1, NEURONS):
        test = hebbs[i].test(emoji)
        if test > max:
            max = test
            winner = i
    return winner
## Dane wejsciowe
INPUTS = 82
LEARNING_RATE = 0.3
FORGETTING_RATE = LEARNING_RATE / 3.0
NEURONS = 5
EMOJI = 4
LIMIT = 1000
print("lr =", LEARNING_RATE, "forgetting=", FORGETTING_RATE)
HEBBS = []
for i in range(0, NEURONS):
    HEBBS.append(Hebb(INPUTS))
ERAS = learn(HEBBS)
print("Po nauce")
for i in range(0, EMOJI):
    winner = test_hebb(HEBBS, emoji[i])
    print("Wygrany dla emotikony", emoji_type[i], "neuron:", winner)
print("Testowanie")
for i in range(0, EMOJI):
    winner = test_hebb(HEBBS, confused_emoji[i])
```

```
        print("Wygrany dla emotikony", emoji_type[i], "neuron:", winner)
    print("Ilosc epok =", ERAS)
```