**Podstawy Sztucznej Inteligencji – Laboratorium nr 6**

Wykonał: Kamil Wieniecki

Temat ćwiczenia: Budowa i działanie sieci Kohonena dla WTM.

1. Cel ćwiczenia

Celem ćwiczenia jest poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły WTM do odwzorowywania istotnych cech liter alfabetu.

1. Realizacja ćwiczenia

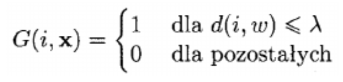
Wybrany przeze mnie język programowania do wykonania laboratorium to **Python.**

Sieć Kohonena nazywana jest inaczej mapą samoorganizującą. Bazuje na algorytmie uczenia bez nauczyciela, wagi na początku są losowane. Charakterystyczne jest tutaj uczenie konkurencyjne tzn. neurony konkurują ze sobą o prawo do reprezentacji danych wejściowych.

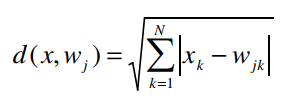
Metoda WTM (Winner Takes Most – „Zwycięzca bierze większość”) działa na zasadzie aktualizacji wag nie tylko neuronu zwycięskiego, ale również neuronów z jego sąsiedztwa. Neuron sąsiadujący to taki, który znajduje się w określonym promieniu od neuronu zwycięskiego. Wartość promienia definiuje się indywidualnie. Neuron którego wagi najmniej różnią się od składowych wektora wejściowego jest wygranym. Relacja opisująca ten stan wygląda następująco:



Neurony, które nie zostają uznane za sąsiednie nie zmieniają swoich wag. Jest to opisane następującą funkcją:



Wykorzystana przeze mnie miara odległości między wektorami to norma Manhattan, która przedstawia się następująco:



Adaptacja wag opisana jest wzorem:



, gdzie **η – współczynnik uczenia**

**Xk – wartość składowej wektora wejściowego**

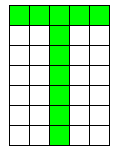
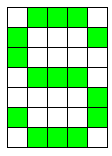
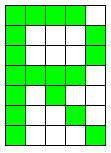
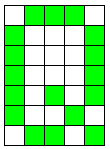
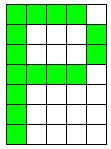
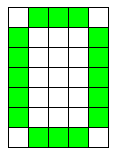
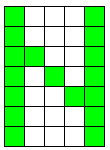
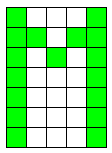
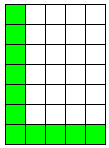
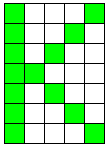
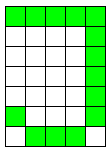
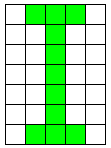
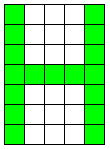
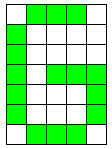
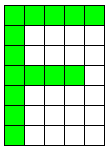
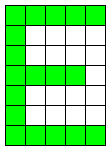
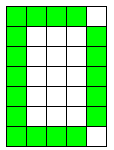
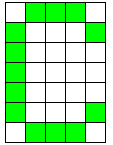
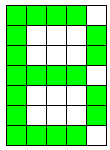
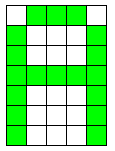
**Wj\*k – wartość składowej wektora wag**

Nadmiarowa ilość neuronów wynika z inicjalizacji wag losowo – przez co część neuronów może znaleźć się w strefie, gdzie nie ma danych lub ich liczba jest znikoma. Neurony te automatycznie zostają wykluczone z sieci, ponieważ nie mają szans na wygraną, nazywamy je neuronami martwymi.

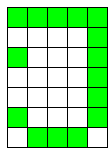
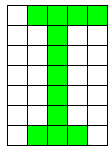
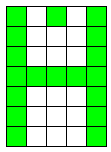
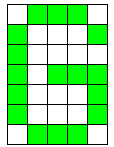
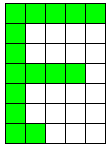
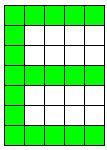
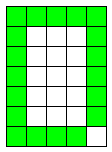
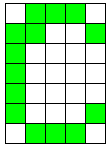
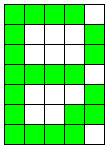
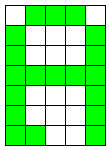
Dane wejściowe tworzą samoistną „bazę wiedzy” na podstawie której sieć uczy się i podejmuje decyzje.

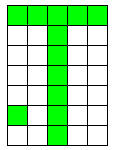
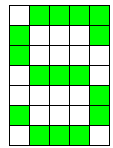
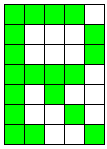
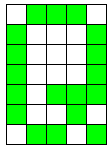
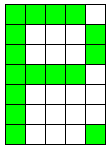
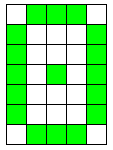
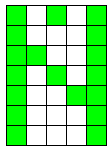
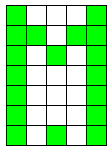
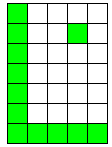
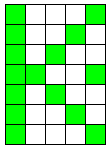
Dane wejściowe zostały zaczerpnięte ze strony zaproponowanej w instrukcji laboratorium. Litery „zaszumione” (testowe) zostały przeze mnie.

Dane wejściowe prezentują się następująco:



Natomiast „zaszumione” dane testowe wyglądają następująco:





Do każdej litery dodany został dokładnie jeden nadmiarowy pixel.

Dane te przedstawione są w systemie binarnym, gdzie zielone pola opisane są liczbą 1, natomiast białe są puste, czyli mają wartość 0.

1. Wyniki

Wyniki przeprowadzone zostały dla trzech różnych współczynników uczenia – 0.01, 0.1 oraz 0.3. Ilość wykorzystanych neuronów do nauki to 20 000, natomiast promienie sąsiedztwa wynosiły 4.0 oraz 6.0 dla każdej z prób. Liter wejściowych jak i testujących było po 20. Wyniki zestawiłem w tabeli:

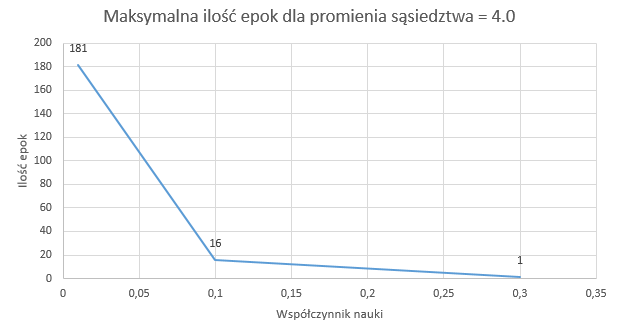
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Współczynnik uczenia | 0,01 | | | | 0,1 | | | | 0,3 | | | |
| Promień sąsiedztwa | 4.0 | | 6.0 | | 4.0 | | 6.0 | | 4.0 | | 6.0 | |
| Lp. | PN [%] | LE | PN [%] | LE | PN [%] | LE | PN [%] | LE | PN [%] | LE | PN [%] | LE |
| 1. | 40 | 1 | 30 | 169 | 30 | 14 | 35 | 16 | 40 | 1 | 35 | 1 |
| 2. | 35 | 173 | 35 | 169 | 60 | 15 | 10 | 16 | 15 | 1 | 50 | 1 |
| 3. | 20 | 172 | 20 | 183 | 35 | 1 | 25 | 13 | 25 | 1 | 25 | 1 |
| 4. | 20 | 162 | 20 | 166 | 20 | 16 | 20 | 16 | 30 | 1 | 30 | 1 |
| 5. | 40 | 170 | 20 | 185 | 15 | 16 | 20 | 1 | 30 | 1 | 30 | 1 |
| 6. | 35 | 1 | 25 | 173 | 35 | 1 | 25 | 16 | 35 | 1 | 15 | 1 |
| 7. | 40 | 158 | 30 | 174 | 15 | 16 | 30 | 16 | 40 | 1 | 25 | 1 |
| 8. | 10 | 181 | 35 | 164 | 45 | 16 | 35 | 14 | 45 | 1 | 40 | 1 |

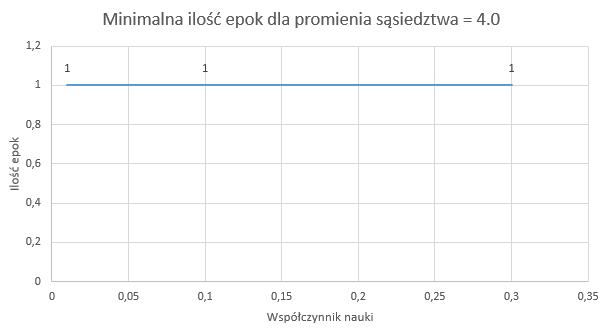
Legenda:

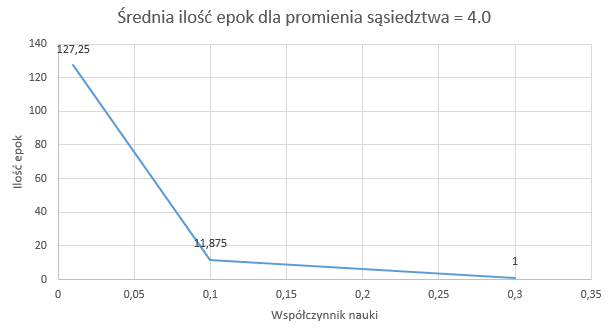
**PN – poprawność nauki [%]**

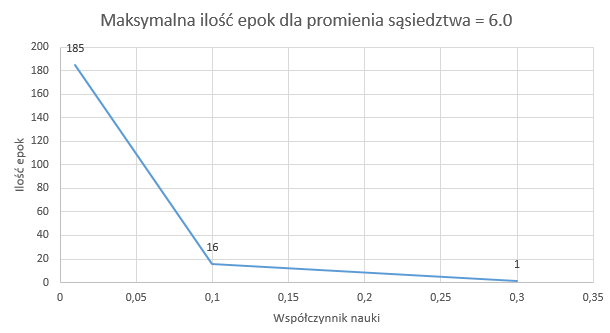
**LE – liczba epok**

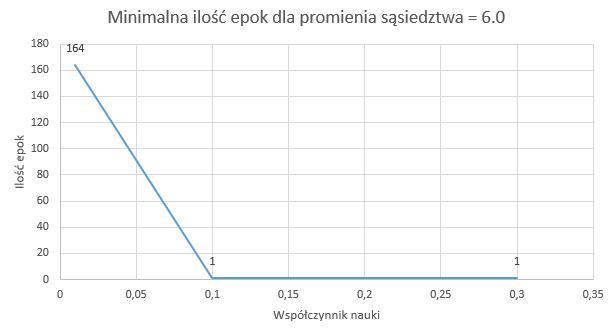
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Współczynnik uczenia | 0,01 | | 0,1 | | 0,3 | |
| Promień sąsiedztwa | 4.0 | 6.0 | 4.0 | 6.0 | 4.0 | 6.0 |
| Maksymalna ilość epok | 181 | 185 | 16 | 16 | 1 | 1 |
| Minimalna ilość epok | 1 | 164 | 1 | 1 | 1 | 1 |
| Średnia ilość epok | 127,25 | 172,875 | 11,875 | 13,5 | 1 | 1 |
| Maksymalna poprawność nauki [%] | 40 | 35 | 60 | 35 | 40 | 50 |
| Minimalna poprawność nauki [%] | 10 | 20 | 15 | 10 | 15 | 15 |
| Średnia poprawność nauki [%] | 30 | 26,875 | 31,875 | 25 | 32,5 | 31,25 |

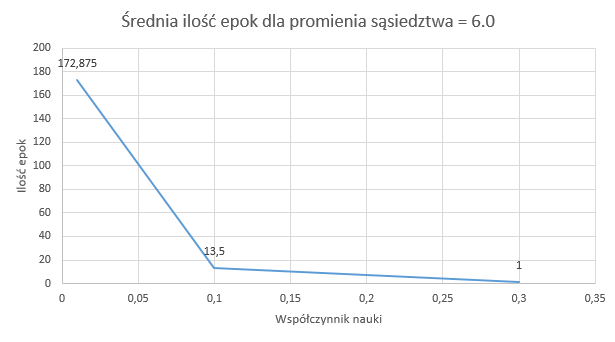


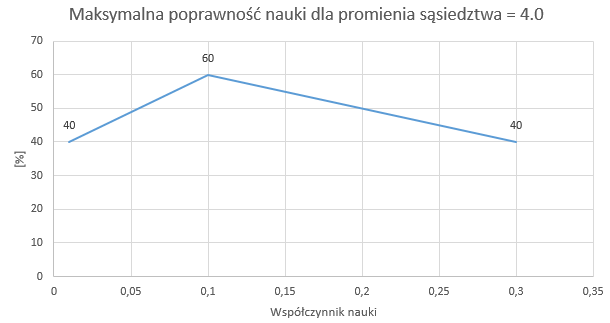


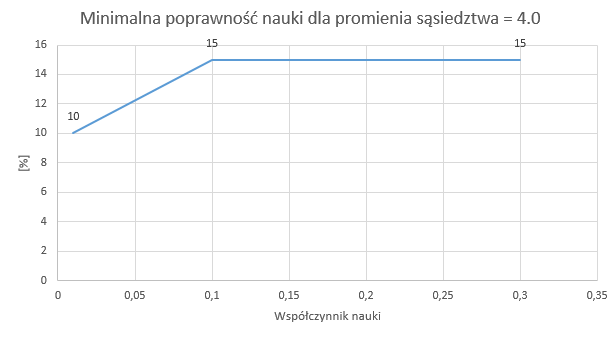


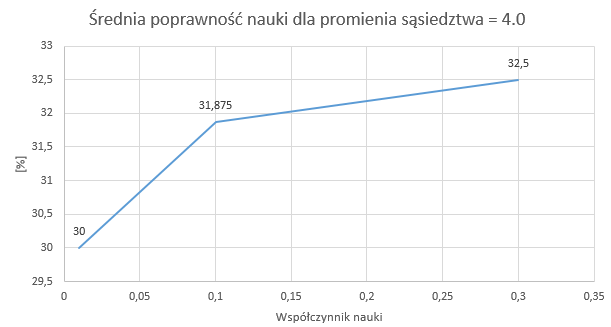


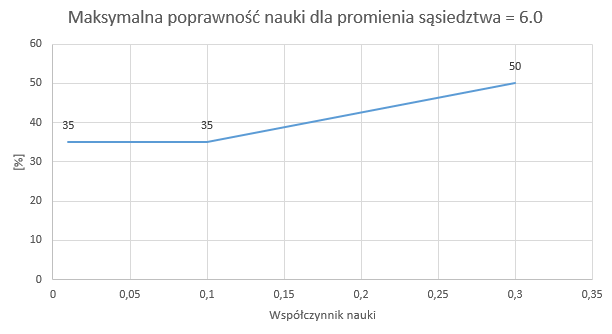


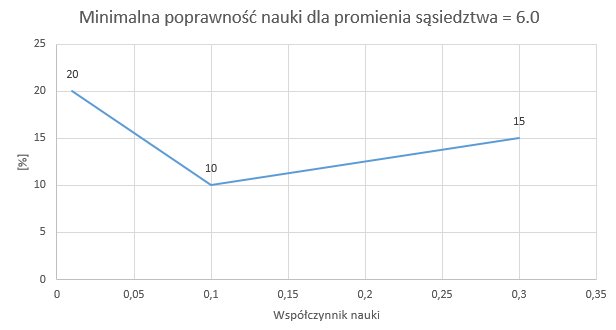


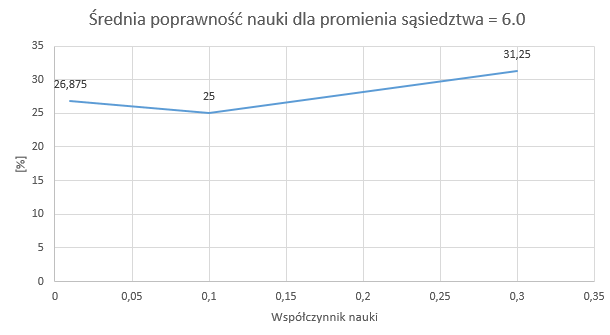












1. Analiza wyników

Niektóre dane ciężko jest zinterpretować. Przykładem takiego wykresu jest minimalna ilość epok dla promienia sąsiedztwa = 4.0. Pozioma linia na poziomie 1 epoki dla różnych współczynników nauki jest dosyć nietypowa.

Widać natomiast zależność dla maksymalnej ilości epok, które wraz ze wzrostem współczynnika nauczania malały. Również zależność tą bardzo dobrze widać dla średniej ilości epok, co potwierdza postawioną tezę.

Najwyższa otrzymana wartość poprawnego nauczenia sieci sięgnęła dla promienia sąsiedztwa 4.0 oraz współczynnika nauczania 0.1 – było to 60%. Nie jest to do końca zadowalający wynik, tym bardziej, że przeciętnie było to w okolicach 30%, natomiast dla promienia 6.0 wartość oscylowała w granicach 25%, czyli około 5% mniej w stosunku do promienia 4.0.

Minimalna wartość poprawnego nauczania dla obu promieni to zaledwie 10% - jest to wynik bardzo daleki od idealnego.

Analizując wykresy można stwierdzić, że promień sąsiedztwa o wartości 4.0 spisał się nieco lepiej niż 6.0.

1. Wnioski

Odpowiedni dobór współczynnika nauki oraz promienia sąsiedztwa, a w dodatku dobór ilości neuronów do tego typu sieci nie jest trywialnym zadaniem.

Po analizie danych otrzymanych z testów można jednoznacznie powiedzieć, że występowało dość dużo błędów. Prawdopodobnie duża rolę odgrywa również ilość wejść (w tym przypadku zaledwie 35). Jeśli ilość ta byłaby większa to myślę, że sieć również lepiej byłaby w stanie wyuczyć się danych. Również większa ilość liter mogłaby poprawić jakoś nauczania się sieci.

Dla tego przypadku najlepiej spisał się promień sąsiedztwa o wartości 4.0 oraz współczynnik nauczania 0.1. Większa wartość promienia powodowała, że większa ilość neuronów zmieniała swoje wagi co zdecydowanie wpływało na wyniki. Należy go odpowiednio dobrać, ale nie może on być też zbyt mały.

W tym przypadku sieć nie spisała się najlepiej. Zdecydowanie lepiej działała sieć Kohonena oparta na metodzie WTA, niż ta oparta na WTM. Jednakże teoretycznie algorytm WTM jest lepszy, jednak doświadczalnie nie spisał się najlepiej.

1. Listing kodu

**Main.py**

from math import sqrt

from Kohonen import Kohonen

from Letters import learn\_letters

from Letters import test\_letters

def learn(kohonens):

"Funkcja uczenia"

counter = 0

winners = []

for i in range(0, LEARN\_SAMPLES):

winners.append(-1)

while(unique(winners) != True):

for i in range(0, LEARN\_SAMPLES):

winner = get\_winner(kohonens, learn\_letters[i])

kohonens[winner].learn(learn\_letters[i], LEARNING\_RATE)

for j in range(0, NEURONS):

"Petla uczaca sasiadow"

if j != winner:

if vector\_distance(kohonens[winner].weights, kohonens[j].weights) <= NEIGHBOURHOOD\_RADIUS:

kohonens[j].learn(learn\_letters[i], LEARNING\_RATE)

for i in range(0, LEARN\_SAMPLES):

winners[i] = get\_winner(kohonens, learn\_letters[i])

counter += 1

if counter == LIMIT:

break

return counter

def unique(winners):

"Funkcja sprawdza czy siec jest juz nauczona"

for i in range(0, LEARN\_SAMPLES):

for j in range(i, LEARN\_SAMPLES):

if i != j:

if winners[i] == winners[j]:

return False

return True

def get\_winner(kohonens, vector):

"Zwraca wygrany neuron"

winner = 0

minimum\_distance = vector\_distance(kohonens[0].weights, vector)

for i in range(1, NEURONS):

current\_distance = vector\_distance(kohonens[i].weights, vector)

if minimum\_distance > current\_distance:

winner = i

minimum\_distance = current\_distance

return winner

def vector\_distance(vector1, vector2):

"Zwraca odleglosc pomiedzy wektorami (miara Manhattan)"

sum = 0

for i in range(0, len(vector1)):

sum += abs(vector1[i] - vector2[i])

return sqrt(sum)

### Dane wejściowe

LEARNING\_RATE = 0.3

INPUTS = 35

NEURONS = 20000

LEARN\_SAMPLES = 20

TEST\_SAMPLES = 20

LIMIT = 3500

NEIGHBOURHOOD\_RADIUS = 6.0

kohonens = []

for i in range(0, NEURONS):

kohonens.append(Kohonen(INPUTS))

learn\_result =[]

test\_result = []

percent = 0

eras = learn(kohonens)

for i in range(0, LEARN\_SAMPLES):

learn\_result.append(get\_winner(kohonens, learn\_letters[i]))

for i in range(0, TEST\_SAMPLES):

test\_result.append(get\_winner(kohonens, test\_letters[i]))

for i in range(0, TEST\_SAMPLES):

if learn\_result[i] != test\_result[i]:

percent += 1

print("Ilosc epok:", eras)

print("Poprawnosc testowania =", ((percent \* 100) / TEST\_SAMPLES), "%")

**Kohonen.py**

from random import uniform

class Kohonen:

def \_\_init\_\_(self, inputs):

self.inputs = inputs

self.weights = []

for i in range(0, inputs):

self.weights.append(uniform(0, 1))

def learn(self, vector, learning\_rate):

"Metoda uczenia"

for i in range(0, self.inputs):

self.weights[i] += learning\_rate \* (vector[i] - self.weights[i])

**Letters.py**

learn\_letters = [

[

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

],

[

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

],

[

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

],

[

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

],

[

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, 1, 1, 1, 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

],

[

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

],

[

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

],

[

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

],

[

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

],

[

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

],

[

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

],

[

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

],

[

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

],

[

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

],

[

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

],

[

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, 0,

0, 1, 1, 0, 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

],

[

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

],

[

1, 1, 1, 1, 1,

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, 0, 0

]

]

test\_letters = [

[

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, 1, 0, 0, 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, 1, 1,

1, 1, 1, 1, 0

],

[

0, 1, 1, 1, 0,

1, 1, 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

],

[

1, 1, 1, 1, 1,

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

],

[

1, 1, 1, 1, 1,

1, 0, 0, 0, 0,

1, 0, 0, 0, 0,

1, 1, 1, 1, 1,

1, 0, 0, 0, 0,

1, 0, 0, 0, 0,

1, 1, 1, 1, 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, 1, 0, 0, 0

],

[

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, 1

],

[

1, 0, 1, 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

],

[

0, 1, 1, 1, 1,

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

],

[

1, 1, 1, 1, 1,

0, 0, 0, 0, 1,

1, 0, 0, 0, 1,

0, 0, 0, 0, 1,

0, 0, 0, 0, 1,

1, 0, 0, 0, 1,

0, 1, 1, 1, 0

],

[

1, 0, 0, 0, 1,

1, 0, 0, 1, 0,

1, 0, 1, 0, 0,

1, 1, 0, 0, 1,

1, 0, 1, 0, 0,

1, 0, 0, 1, 0,

1, 0, 0, 0, 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

],

[

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, 1, 0, 1

],

[

1, 0, 1, 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

],

[

0, 1, 1, 1, 0,

1, 0, 0, 0, 1,

1, 0, 0, 0, 1,

1, 0, 1, 0, 1,

1, 0, 0, 0, 1,

1, 0, 0, 0, 1,

0, 1, 1, 1, 0

],

[

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, 1

],

[

0, 1, 1, 1, 0,

1, 0, 0, 0, 1,

1, 0, 0, 0, 1,

1, 0, 0, 0, 1,

1, 0, 1, 1, 1,

1, 0, 0, 1, 0,

0, 1, 1, 0, 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, 1, 0, 0, 1

],

[

0, 1, 1, 1, 1,

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

],

[

1, 1, 1, 1, 1,

0, 0, 1, 0, 0,

0, 0, 1, 0, 0,

0, 0, 1, 0, 0,

0, 0, 1, 0, 0,

1, 0, 1, 0, 0,

0, 0, 1, 0, 0

]

]

1. Bibliografia

<http://aragorn.pb.bialystok.pl/~gkret/SSN/SSN_w10.PDF>

<https://pl.wikipedia.org/wiki/Sie%C4%87_Kohonena>

<http://galaxy.agh.edu.pl/~vlsi/AI/koho_t/>