Московский авиационный институт
(Национальный исследовательский университет)
Институт №8 «Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа № 5

по курсу «Нейроинформатика».

Тема: «Сети с обратными связями».

Студент: Вельтман Л.Я.
Группа: 80-407Б
Преподаватели: Тюменцев Ю.В.
Аносова Н.П.

Вариант: 7

Оценка:

Москва, 2020

Цель работы.

Целью работы является исследование свойств сетей Хопфилда, Хэмминга и Элмана, алгоритмов обучения, а также применение сетей в задачах распознавания статических и динамических образов.

Основные этапы работы.

1. Использовать сеть Элмана для распознавания динамических образов. Проверить качество распознавания.
2. Использовать сеть Хопфилда для распознавания статических образов. Проверить качество распознавания.
3. Использовать сеть Хэмминга для распознавания статических образов. Проверить качество распознавания.

Оборудование.

Операционная система: macOS Catalina version 10.15.5

Процессор: 2,3 GHz 2-ядерный процессор Intel Core i5

Оперативная память: 8 ГБ 2133 MHz LPDDR3

Программное обеспечение.

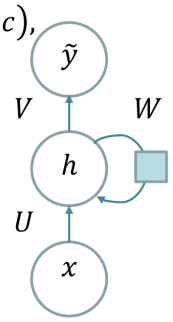
Работа выполнена на Python3 с применением библиотек numpy (для вычислений), pandas и matplotlib (для графиков) при помощи командной оболочки Jupyter Notebook.

Сценарий выполнения работы.

Нейронная сеть Элмана — один из видов рекуррентной сети, которая так же как и сеть Джордана получается из многослойного перцептрона введением обратных связей, только связи идут не от выхода сети, а от выходов внутренних нейронов. Это позволяет учесть предысторию наблюдаемых процессов и накопить информацию для выработки правильной стратегии управления. Эти сети могут применяются в системах управления движущимися объектами, т.к. их главной особенностью является запоминание последовательностей.

Уравнения, описывающие внутреннее состояние и выход сети, которая получена в результате развертывания сети Элмана во времени, имеют следующий вид:

$$a^{(t)} = Ux^{(t)} + Wh^{(t-1)} + b,$$
$$h^{(t)} = f\left(a^{(t)}\right) = f\left(Ux^{(t)} + Wh^{(t-1)} + b\right),$$
$$o^{(t)} = Vh^{(t)} + c, \qquad \tilde{y}^{(t)} = g\left(o^{(t)}\right) = g\left(Vh^{(t)} + c\right),$$



где U, W, V – матрицы весов,

b, c – вектора сдвига,

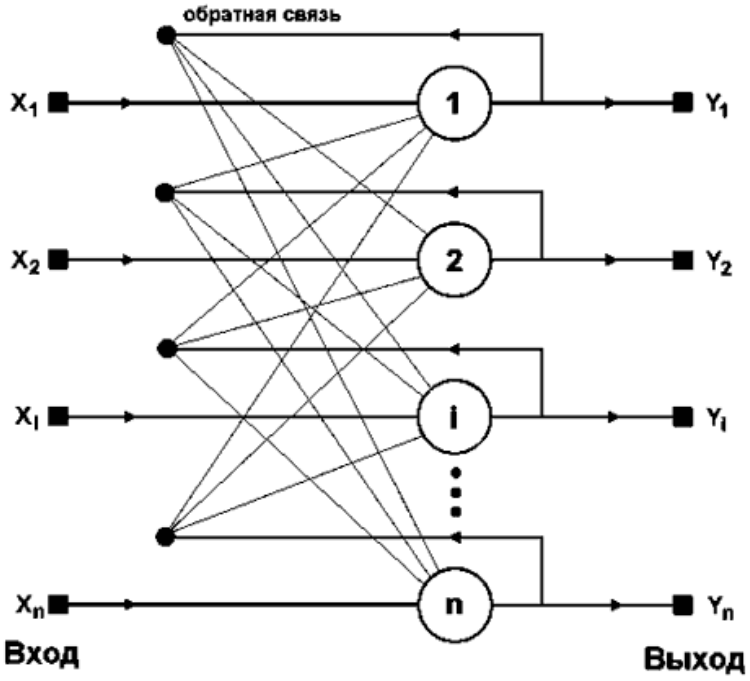
$h^{(t)}$ – вектор скрытых переменных в момент t (при обработке примера с номером t из заданной входной последовательности),

$\tilde{y}^{(t)}$ – выход сети в момент t,

$f(\cdot), g(\cdot)$ – функции активации

Среди различных конфигураций искусственных нейронных сетей (НС) встречаются такие, в которых весовые коэффициенты рассчитываются только однажды перед началом функционирования сети на основе информации об обрабатываемых данных, и все обучение сети сводится именно к этому расчету. Сеть фактически просто запоминает образцы до того, как на ее вход поступают реальные данные, и не может изменять свое поведение, поэтому говорить о звене обратной связи с "миром" (учителем) не приходится. Из сетей с подобной логикой работы наиболее известны сеть Хопфилда и сеть Хэмминга, которые обычно используются для организации ассоциативной памяти.

Структурная схема сети Хопфилда



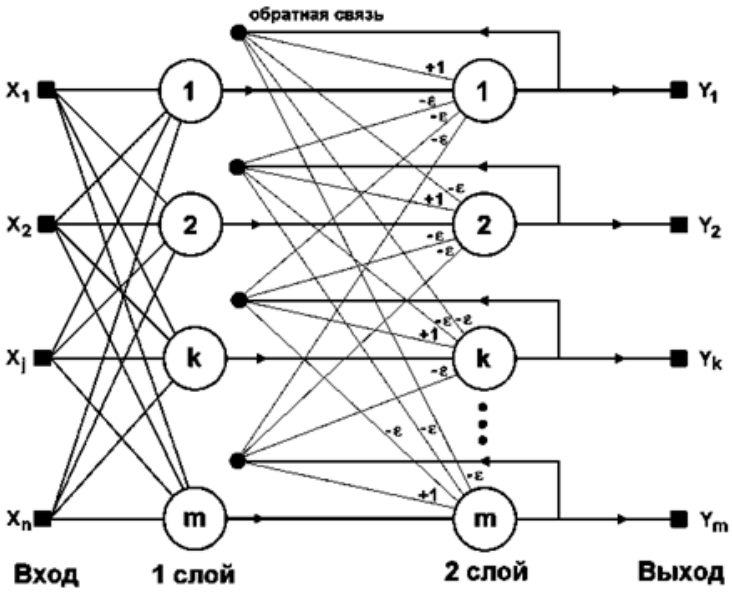
Задача, решаемая данной сетью в качестве ассоциативной памяти формулируется следующим образом. Известен некоторый набор двоичных сигналов, которые считаются образцовыми. Сеть должна уметь из произвольного неидеального сигнала, поданного на ее вход, выделить

("вспомнить" по частичной информации) соответствующий образец (если такой есть) или "дать заключение" о том, что входные данные не соответствуют ни одному из образцов.

Если, например, сигналы представляют собой некие изображения, то, отобразив в графическом виде данные с выхода сети, можно будет увидеть картинку, полностью совпадающую с одной из образцовых (в случае успеха) или же "вольную импровизацию" сети (в случае неудачи).

Когда нет необходимости, чтобы сеть в явном виде выдавала образец, то есть достаточно получать номер образца, ассоциативную память успешно реализует сеть Хэмминга. Данная сеть характеризуется, по сравнению с сетью Хопфилда, меньшими затратами на память и объемом вычислений, что становится очевидным из ее структуры

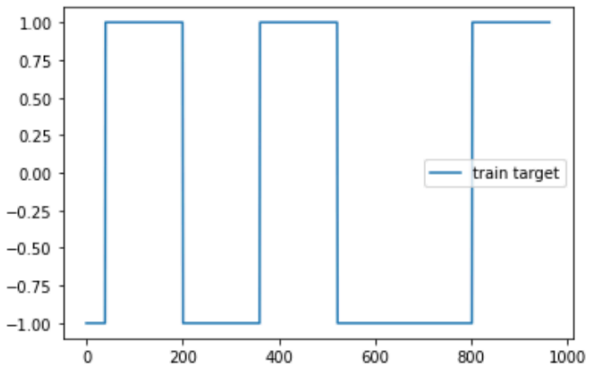
Идея работы сети состоит в нахождении расстояния Хэмминга от тестируемого образа до всех образцов. Расстоянием Хэмминга называется число отличающихся битов в двух бинарных векторах. Сеть должна выбрать образец с минимальным расстоянием Хэмминга до неизвестного входного сигнала, в результате чего будет активизирован только один выход сети, соответствующий этому образцу.



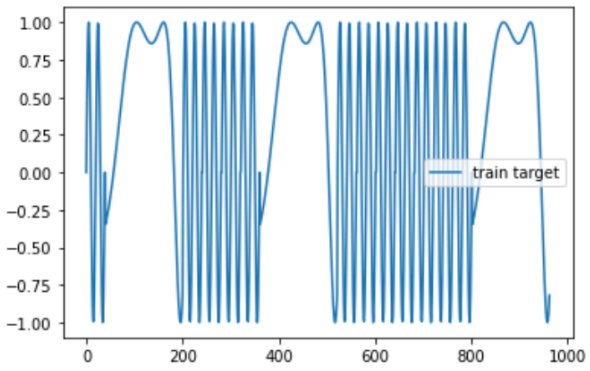
Входные данные и результаты

Задание 1

```
pl.plot(T.reshape(T.shape[0]))
pl.legend(['train target'])
pl.show()
```



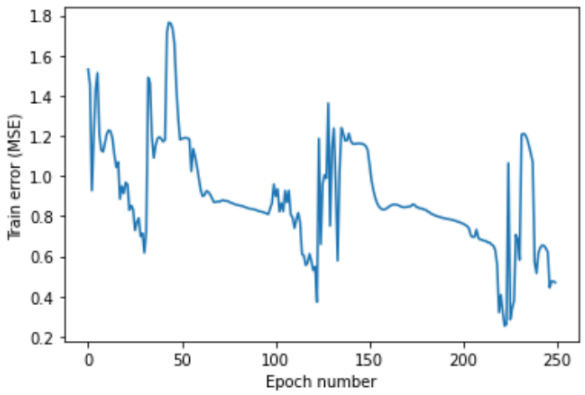

```
pl.plot(P.reshape(P.shape[0]))
pl.legend(['train target'])
pl.show()
```



Отобразим ошибку обучения на графике

In [90]:
```
pl.plot(error)
pl.xlabel('Epoch number')
pl.ylabel('Train error (MSE)')
```

Out[90]: `Text(0, 0.5, 'Train error (MSE)')`

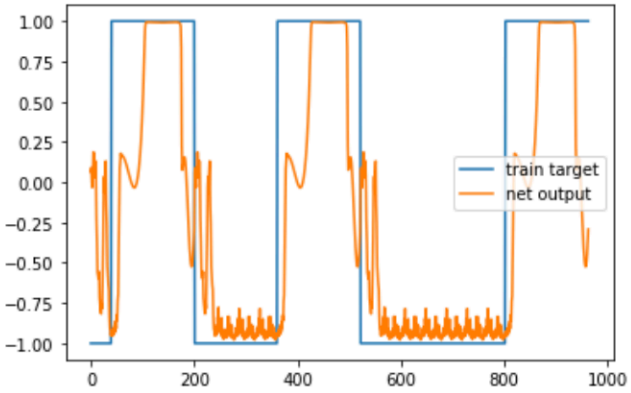


```
:
output2[output2 >= 0] = 1.0
output2[output2 < 0] = -1.0
MSE = mean_squared_error(T2, output2)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))
```

In

```
MSE = 0.5674978503869303
RMSE = 0.7533245319163119
```

```
pl.show()
```



Преобразуем предсказанные значения.

In [92]:

```
output[output >= 0] = 1.0
output[output < 0] = -1.0
```

In [93]:

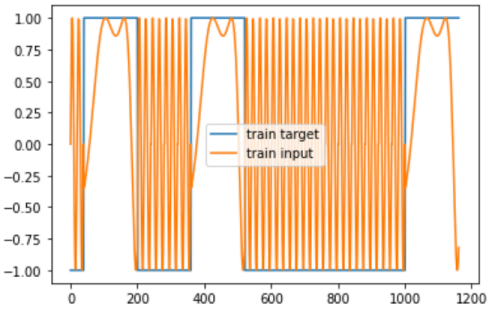
```
MSE = mean_squared_error(T, output)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))
```

```
MSE = 0.6853582554517134
RMSE = 0.8278636696049135
```

Для проверки качества распознавания сформируем новое обучающее множество, изменив одно из значений R.

In [96]:

```
pl.plot(T2.reshape(T2.shape[0]))
pl.plot(P2.reshape(P2.shape[0]))
pl.legend(['train target', 'train input'])
pl.show()
```

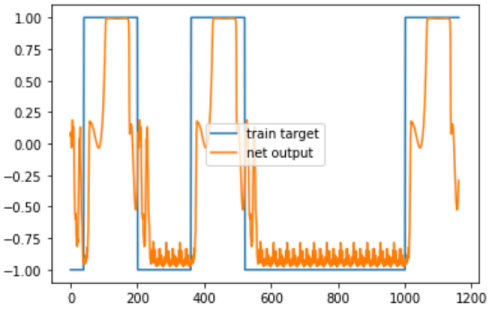


In [97]:

```
# Simulate network
output2 = net.sim(P2)
```

In [98]:

```
pl.plot(T2.reshape(T2.shape[0]))
pl.plot(output2.reshape(output2.shape[0]))
pl.legend(['train target', 'net output'])
pl.show()
```



Задание 2

Подаем в сеть первый образ.

In [21]:
```
result = dhnet.predict(three)
```

Результат распознавания

In [22]:
```
def DrawBinImage(img):
    for row in img.tolist():
        print(' '.join(' #'[val] for val in row))
```

In [23]:
```
DrawBinImage(three.reshape(12,10))
```

```
# # # # # #
# # # # # # #
          # #
          # #
          # #
    # # # #
    # # # #
          # #
          # #
          # #
# # # # # # #
# # # # # #
```

In [24]:
```
DrawBinImage(result.reshape(12, 10))
```

```
# # # # # #
# # # # # # #
          # #
          # #
          # #
    # # # #
    # # # #
          # #
          # #
          # #
# # # # # # #
# # # # # #
```

Зашумление второго образа на 20%

In [26]:
```
noiseOne = Noise(one, 20)
DrawBinImage(noiseOne.reshape(12, 10))
```

```
  # # # #
  # # # #
  # # # #
  # # # #
  # # # #
  # # # #
    #   #
  # # #
  #     #
  # # # #
  #   # #
  # # # #
```

Подаем второй зашумленный образ в сеть

In [27]:
```
result2 = dhnet.predict(noiseOne)
DrawBinImage(result2.reshape(12, 10))
```

```
  # # # #
  # # # #
  # # # #
  # # # #
  # # # #
  # # # #
  # # # #
  # # # #
  # # # #
  # # # #
  # # # #
  # # # #
```

Зашумление третьего образа на 30%

In [28]:
```
noiseZero = Noise(zero, 30)
DrawBinImage(noiseZero.reshape(12, 10))
```

```
    # # #
  # # # # #
# # #     # # #
# # #     # #
#   #       # #
# #         # #
# # #     #
# # #     #   #
  # #     #
    #   #
```

Подаем третий зашумленный образ в сеть

In [29]:
```
result3 = dhnet.predict(noiseZero)
DrawBinImage(result3.reshape(12, 10))
```

```
  # # # #
  # # # # # #
# # #     # # #
# # #     # # #
# # #     # # #
# # #     # # #
# # #     # # #
# # #     # # #
  # # # # # #
    # # # #
```

Задание 3

In [32]:
```
network = nl.net.newhop(a, max_init=600)

network.layers[0].np['w'][:] = LW
network.layers[0].np['b'][:] = 0
```

In [33]:
```
A = IW @ three + b
A
```

Out[33]:
```
array([[124., 140., 168., 240., 168., 116., 160.],
       [124., 140., 168., 240., 168., 116., 160.],
       [124., 140., 168., 240., 168., 116., 160.],
       [124., 140., 168., 240., 168., 116., 160.],
       [124., 140., 168., 240., 168., 116., 160.],
       [124., 140., 168., 240., 168., 116., 160.],
       [124., 140., 168., 240., 168., 116., 160.]])
```

In [34]:
```
res = network.sim(A)
```

In [35]:
```
answerClass = np.argmax(res[0])
print('Result class: {}'.format(answerClass))
```

```
Result class: 3
```

Нарисуем образ по предсказанной позиции

In [36]:
```
check = Patterns[answerClass]
check[check == -1] = 0
DrawBinImage(check.reshape(12, 10))
```

```
  # # # # #
  # # # # # #
          # #
          # #
          # #
    # # # #
    # # # #
          # #
          # #
          # #
# # # # # # #
# # # # # #
```

Получаем верный ответ

Зашумленный на 20% второй образ

In [37]:
```
noiseOne
```

Out[37]:
```
matrix([[0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
         0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0,
         0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
         0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
         0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0,
         1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0]])
```

In [38]:
```
noiseOne = np.asarray(noiseOne)[0]
noiseOne[noiseOne == 0] = -1
A = IW @ noiseOne + b
res = network.sim(A)

answerClass = np.argmax(res[0])
print('Result class: {}'.format(answerClass))

check = Patterns[answerClass]
check[check == -1] = 0
DrawBinImage(check.reshape(12, 10))
```

```
Result class: 1
      # # # #
      # # # #
      # # # #
      # # # #
      # # # #
      # # # #
      # # # #
      # # # #
      # # # #
      # # # #
      # # # #
      # # # #
```

Зашумленный на 30% третий образ

In [39]:
```
noiseZero = np.asarray(noiseZero)[0]
noiseZero[noiseZero == 0] = -1
A = IW @ noiseZero + b
res = network.sim(A)

answerClass = np.argmax(res[0])
print('Result class: {}'.format(answerClass))

check = Patterns[answerClass]
check[check == -1] = 0
DrawBinImage(check.reshape(12, 10))
```

```
Result class: 0

        # # # #
      # # # # # #
    # # #     # # #
    # # #     # # #
    # # #     # # #
    # # #     # # #
    # # #     # # #
    # # #     # # #
      # # # # # #
        # # # #
```

Код программы.

```
import neurolab as nl
import numpy as np
import numpy.matlib
from neupy import algorithms
from scipy.spatial import distance
import pylab as pl
from sklearn.metrics import mean_squared_error


# ### 1 задание
#
# Использовать сеть Элмана для распознавания динамических образов.

p1_k = np.linspace(0, 1, int(1 / 0.025), endpoint=True)

# Основной сигнал
p1 = np.sin(4 * np.pi * p1_k)

t1 = np.ones(len(p1_k)) * (-1)


p2_k = np.linspace(2.16, 4.04, int(4.04 / 0.025), endpoint=True)

# Сигнал, подлежащий распознаванию
p2 = np.cos(np.cos(p2_k) * p2_k * p2_k + 5 * p2_k)

t2 = np.ones(len(p2_k))

# Длительность основного сигнала
R = np.array([1, 4, 7])

p2 = p2.reshape(1, p2.shape[0])
t2 = t2.reshape(1, t2.shape[0])


# Определяем входное множество

P = np.concatenate((numpy.matlib.repmat(p1, 1, R[0]), p2,
                    numpy.matlib.repmat(p1, 1, R[1]), p2,
                    numpy.matlib.repmat(p1, 1, R[2]), p2), axis=1).reshape(-1, 1)

T = np.concatenate((numpy.matlib.repmat(t1, 1, R[0]), t2,
                    numpy.matlib.repmat(t1, 1, R[1]), t2,
                    numpy.matlib.repmat(t1, 1, R[2]), t2), axis=1).reshape(-1, 1)


pl.plot(T.reshape(T.shape[0]))
#pl.plot(output.reshape(output.shape[0]))
pl.legend(['train target', 'net output'])
pl.show()

pl.plot(P.reshape(P.shape[0]))
#pl.plot(output.reshape(output.shape[0]))
pl.legend(['train target', 'net output'])
pl.show()

# Create network with 2 layers
net = nl.net.newelm([[-10, 10]], [8, 1], [nl.trans.TanSig(), nl.trans.TanSig()])
# Set initialized functions and init
net.layers[0].np['w'][:] = 1 # set weight for all input neurons to 1
net.layers[0].np['b'][:] = 0 # set bias for all input neurons to 0
net.init()


# Обучаем сеть и вычисляем метрики обучения, ошибка MSE.

# Train network, error - MSE
error = net.train(P, T, epochs=250, show=10, goal=0.01)

# Simulate network
output = net.sim(P)


# Отобразим ошибку обучения на графике
```

```
pl.plot(error)
pl.xlabel('Epoch number')
pl.ylabel('Train error (MSE)')


# Отобразим на графике этаонные значения и предсказанные сетью.

pl.plot(T.reshape(T.shape[0]))
pl.plot(output.reshape(output.shape[0]))
pl.legend(['train target', 'net output'])
pl.show()


# Преобразуем предсказанные значения.

output[output >= 0] = 1.0
output[output < 0] = -1.0

MSE = mean_squared_error(T, output)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))


# Дя проверки качества распознавания сформируем новое обучающее множество, изменив одно из значений
R.

p1_k = np.linspace(0, 1, int(1 / 0.025), endpoint=True)

# Основной сигнал
p1 = np.sin(4 * np.pi * p1_k)

t1 = np.ones(len(p1_k)) * (-1)


p2_k = np.linspace(2.16, 4.04, int(4.04 / 0.025), endpoint=True)

# Сигнал, подлежащий распознаванию
p2 = np.cos(np.cos(p2_k) * p2_k * p2_k + 5 * p2_k)

t2 = np.ones(len(p2_k))

# Длительность основного сигнала
R = np.array([1, 4, 12])

p2 = p2.reshape(1, p2.shape[0])
t2 = t2.reshape(1, t2.shape[0])

P2 = np.concatenate((numpy.matlib.repmat(p1, 1, R[0]), p2,
                     numpy.matlib.repmat(p1, 1, R[1]), p2,
                     numpy.matlib.repmat(p1, 1, R[2]), p2), axis=1).reshape(-1, 1)

T2 = np.concatenate((numpy.matlib.repmat(t1, 1, R[0]), t2,
                     numpy.matlib.repmat(t1, 1, R[1]), t2,
                     numpy.matlib.repmat(t1, 1, R[2]), t2), axis=1).reshape(-1, 1)

pl.plot(T2.reshape(T2.shape[0]))
pl.plot(P2.reshape(P2.shape[0]))
pl.legend(['train target', 'train input'])
pl.show()

# Simulate network
output2 = net.sim(P2)

pl.plot(T2.reshape(T2.shape[0]))
pl.plot(output2.reshape(output2.shape[0]))
pl.legend(['train target', 'net output'])
pl.show()


# ### 2 задание
#
# Использовать сеть Хопфилда для распознавания статических образов.

# В соответствие с 7 вариантом цифры определились следующим образом: $[3, 1, 0]$
#
# Матрица одной цифры $[10$ x $12]$, где 10 - ширина, 12 - высота

zero = np.matrix([
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
```

```
                0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
                0, 1, 1, 1, 0, 0, 1, 1, 1, 0,
                0, 1, 1, 1, 0, 0, 1, 1, 1, 0,
                0, 1, 1, 1, 0, 0, 1, 1, 1, 0,
                0, 1, 1, 1, 0, 0, 1, 1, 1, 0,
                0, 1, 1, 1, 0, 0, 1, 1, 1, 0,
                0, 1, 1, 1, 0, 0, 1, 1, 1, 0,
                0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
                0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

one = np.matrix([
                0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
                0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
                0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
                0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
                0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
                0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
                0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
                0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
                0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
                0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
                0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
                0, 0, 0, 1, 1, 1, 1, 0, 0, 0])

two = np.matrix([
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0])

three = np.matrix([
                0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
                0, 0, 1, 1, 1, 1, 1, 1, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                0, 0, 0, 0, 1, 1, 1, 1, 0, 0,
                0, 0, 0, 0, 1, 1, 1, 1, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                0, 0, 1, 1, 1, 1, 1, 1, 1, 0,
                0, 0, 1, 1, 1, 1, 1, 1, 0, 0])

four = np.matrix([
                0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
                0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
                0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
                0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
                0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
                0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
                0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 1, 0])

six = np.matrix([
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1,
                1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                1, 0, 0, 0, 0, 0, 0, 0, 0,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
                1, 1, 0, 0, 1, 1, 0, 0, 0, 0,
                1, 1, 0, 0, 1, 1, 0, 0, 0, 0,
                1, 1, 0, 0, 1, 1, 0, 0, 0, 0,
                1, 1, 0, 0, 1, 1, 0, 0, 0, 0,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0])
```

```
nine = np.matrix([
                0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
                0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
                0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
                0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
                0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
                0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
                0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

data = np.concatenate([three, one, zero], axis=0)

dhnet = algorithms.DiscreteHopfieldNetwork(mode = 'async', n_times = 600)
dhnet.train(data)


# Подаем в сеть первый образ.
result = dhnet.predict(three)


# Результат распознавания
def DrawBinImage(img):
    for row in img.tolist():
        print(' '.join(' #'[val] for val in row))

DrawBinImage(three.reshape(12,10))
DrawBinImage(result.reshape(12, 10))


def Noise(img, noisePercent):
    limit = img.shape[1]
    tmp = img.T

    for i in range(noisePercent):
        pos = np.random.randint(1, limit, 1)[0]
        tmp[pos] *= 0
    res = tmp.T
    return res


# Зашумление второго образа на 20%

noiseOne = Noise(one, 20)
DrawBinImage(noiseOne.reshape(12, 10))


# Подаем второй зашумленный образ в сеть

result2 = dhnet.predict(noiseOne)
DrawBinImage(result2.reshape(12, 10))


# Зашумление третьего образа на 30%

noiseZero = Noise(zero, 30)
DrawBinImage(noiseZero.reshape(12, 10))


# Подаем третий зашумленный образ в сеть

result3 = dhnet.predict(noiseZero)
DrawBinImage(result3.reshape(12, 10))


# ### 3 задание
#
# Использовать сеть Хэмминга для распознавания статических образов.

zero = np.array([
               -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
               -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,
               -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,
               -1, 1, 1, 1, -1, -1, 1, 1, 1, -1,
               -1, 1, 1, 1, -1, -1, 1, 1, 1, -1,
               -1, 1, 1, 1, -1, -1, 1, 1, 1, -1,
               -1, 1, 1, 1, -1, -1, 1, 1, 1, -1,
               -1, 1, 1, 1, -1, -1, 1, 1, 1, -1,
```

```
               -1, 1, 1, 1, -1, -1, 1, 1, 1, -1,
               -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,
               -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,
               -1, -1, -1, -1, -1, -1, -1, -1, -1, -1])

one = np.array([
                -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,
                -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,
                -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,
                -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,
                -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,
                -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,
                -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,
                -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,
                -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,
                -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,
                -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,
                -1, -1, -1, 1, 1, 1, 1, -1, -1, -1])

two = np.array([
                1, 1, 1, 1, 1, 1, 1, 1, -1, -1,
                1, 1, 1, 1, 1, 1, 1, 1, -1, -1,
                -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,
                -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,
                -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,
                1, 1, 1, 1, 1, 1, 1, 1, -1, -1,
                1, 1, 1, 1, 1, 1, 1, 1, -1, -1,
                1, 1, -1, -1, -1, -1, -1, -1, -1, -1,
                1, 1, -1, -1, -1, -1, -1, -1, -1, -1,
                1, 1, -1, -1, -1, -1, -1, -1, -1, -1,
                1, 1, 1, 1, 1, 1, 1, 1, -1, -1,
                1, 1, 1, 1, 1, 1, 1, 1, -1, -1])

three = np.array([
                -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,
                -1, -1, 1, 1, 1, 1, 1, 1, 1, -1,
                -1, -1, -1, -1, -1, -1, -1, 1, 1, -1,
                -1, -1, -1, -1, -1, -1, -1, 1, 1, -1,
                -1, -1, -1, -1, -1, -1, -1, 1, 1, -1,
                -1, -1, -1, -1, 1, 1, 1, 1, -1, -1,
                -1, -1, -1, -1, 1, 1, 1, 1, -1, -1,
                -1, -1, -1, -1, -1, -1, -1, 1, 1, -1,
                -1, -1, -1, -1, -1, -1, -1, 1, 1, -1,
                -1, -1, -1, -1, -1, -1, -1, 1, 1, -1,
                -1, -1, 1, 1, 1, 1, 1, 1, 1, -1,
                -1, -1, 1, 1, 1, 1, 1, 1, -1, -1])

four = np.array([
                -1, 1, 1, -1, -1, -1, -1, 1, 1, -1,
                -1, 1, 1, -1, -1, -1, -1, 1, 1, -1,
                -1, 1, 1, -1, -1, -1, -1, 1, 1, -1,
                -1, 1, 1, -1, -1, -1, -1, 1, 1, -1,
                -1, 1, 1, -1, -1, -1, -1, 1, 1, -1,
                -1, 1, 1, 1, 1, 1, 1, 1, 1, -1,
                -1, 1, 1, 1, 1, 1, 1, 1, 1, -1,
                -1, -1, -1, -1, -1, -1, -1, 1, 1, -1,
                -1, -1, -1, -1, -1, -1, -1, 1, 1, -1,
                -1, -1, -1, -1, -1, -1, -1, 1, 1, -1,
                -1, -1, -1, -1, -1, -1, -1, 1, 1, -1,
                -1, -1, -1, -1, -1, -1, -1, 1, 1, -1])

six = np.array([
                1, 1, 1, 1, 1, 1, -1, -1, -1, -1,
                1, 1, 1, 1, 1, 1, -1, -1, -1, -1, 1,
                1, -1, -1, -1, -1, -1, -1, -1, -1, 1,
                1, -1, -1, -1, -1, -1, -1, -1, -1,
                1, 1, 1, 1, 1, 1, -1, -1, -1, -1,
                1, 1, 1, 1, 1, 1, -1, -1, -1, -1,
                1, 1, -1, -1, 1, 1, -1, -1, -1, -1,
                1, 1, -1, -1, 1, 1, -1, -1, -1, -1,
                1, 1, -1, -1, 1, 1, -1, -1, -1, -1,
                1, 1, -1, -1, 1, 1, -1, -1, -1, -1,
                1, 1, 1, 1, 1, 1, -1, -1, -1, -1,
                1, 1, 1, 1, 1, 1, -1, -1, -1, -1])

nine = np.array([
                -1, -1, -1, -1, 1, 1, 1, 1, 1, 1,
                -1, -1, -1, -1, 1, 1, 1, 1, 1, 1,
                -1, -1, -1, -1, 1, 1, -1, -1, 1, 1,
                -1, -1, -1, -1, 1, 1, -1, -1, 1, 1,
                -1, -1, -1, -1, 1, 1, -1, -1, 1, 1,
```

```
                -1, -1, -1, -1, 1, 1, -1, -1, 1, 1,
                -1, -1, -1, -1, 1, 1, 1, 1, 1, 1,
                -1, -1, -1, -1, 1, 1, 1, 1, 1, 1,
                -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
                -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
                -1, -1, -1, -1, 1, 1, 1, 1, 1, 1,
                -1, -1, -1, -1, 1, 1, 1, 1, 1, 1])


Q = 7
Patterns = np.array([zero, one, two, three, four, six, nine])
eps = 1 / (Q - 2)

Result = 10 * 12

IW = np.array([zero.T, one.T, two.T, three.T, four.T, six.T, nine.T])
b = Result * np.ones((Q, 1))

a = np.zeros((Q, Q))
for i in range(Q):
    a[i] = IW[i] @ Patterns[i] + b[i]

LW = np.eye(Q)
LW[LW == 0.0] = -eps

network = nl.net.newhop(a, max_init=600)

network.layers[0].np['w'][:] = LW
network.layers[0].np['b'][:] = 0

A = IW @ three + b
res = network.sim(A)

answerClass = np.argmax(res[0])
print('Result class: {}'.format(answerClass))


# Нарисуем образ по предсказанной позиции

check = Patterns[answerClass]
check[check == -1] = 0
DrawBinImage(check.reshape(12, 10))


# Получаем верный ответ

# Зашумленный на 20% второй образ

noiseOne = np.asarray(noiseOne)[0]
noiseOne[noiseOne == 0] = -1
A = IW @ noiseOne + b
res = network.sim(A)

answerClass = np.argmax(res[0])
print('Result class: {}'.format(answerClass))

check = Patterns[answerClass]
check[check == -1] = 0
DrawBinImage(check.reshape(12, 10))


# Зашумленный на 30% третий образ

noiseZero = np.asarray(noiseZero)[0]
noiseZero[noiseZero == 0] = -1
A = IW @ noiseZero + b
res = network.sim(A)

answerClass = np.argmax(res[0])
print('Result class: {}'.format(answerClass))

check = Patterns[answerClass]
check[check == -1] = 0
DrawBinImage(check.reshape(12, 10))
```

Выводы:

В ходе лабороторной работы я познакомилась с новыми архитектурами: сети Хопфилда и Хэмминга. В отличие от классических задач обучения с учителем, в таких сетях не нужно тестовых примеров с готовыми ответами, они учатся просто из происходящего, запоминая события. Тогда с чего бы ы не начали, мы придем к одному из имеющихся воспоминаний, то есть вызовем самую близкую ассоциацию. Однако, например, у сети Хопфилда есть ряд недостатков. Относительно небольшой объём памяти, величину которого можно оценить выражением:

$$M \approx \frac{N}{2 \ln N}$$

Попытка записи большего числа образов приводит к тому, что нейронная сеть перестаёт их распознавать.
Наряду с запомненными образами в сети хранятся и их негативы.
Достижение устойчивого состояния не гарантирует правильный ответ сети. Это происходит из-за того, что сеть может сойтись к так называемым ложным аттракторам.