Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа № 5

по курсу «Нейроинформатика»

Тема: Сети с обратными связями.

Студент: Куликов А.В.
Группа: М80-408Б-17
Преподаватель: Аносова Н.П.
Дата: 11 декабря 2020
Оценка:

Москва, 2020

Цель работы: 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:

Процессор: AMD Ryzen 5 Mobile 3550H

Объем оперативной памяти: 8 Гб

Программное обеспечение:

Python 3.8.5, MATLAB r2020

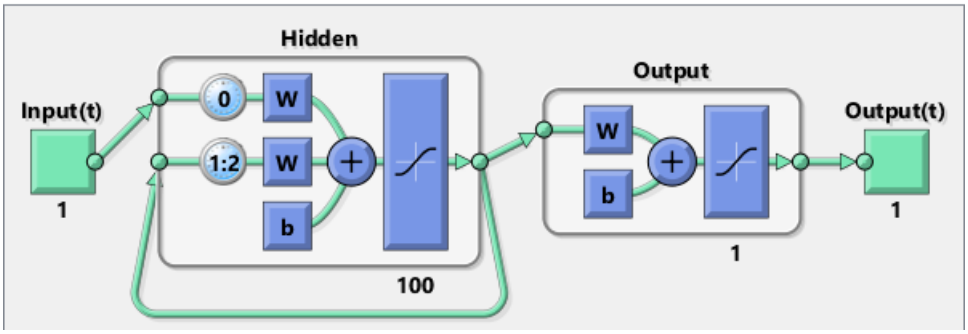
Сценарий выполнения работы:

Задание №1

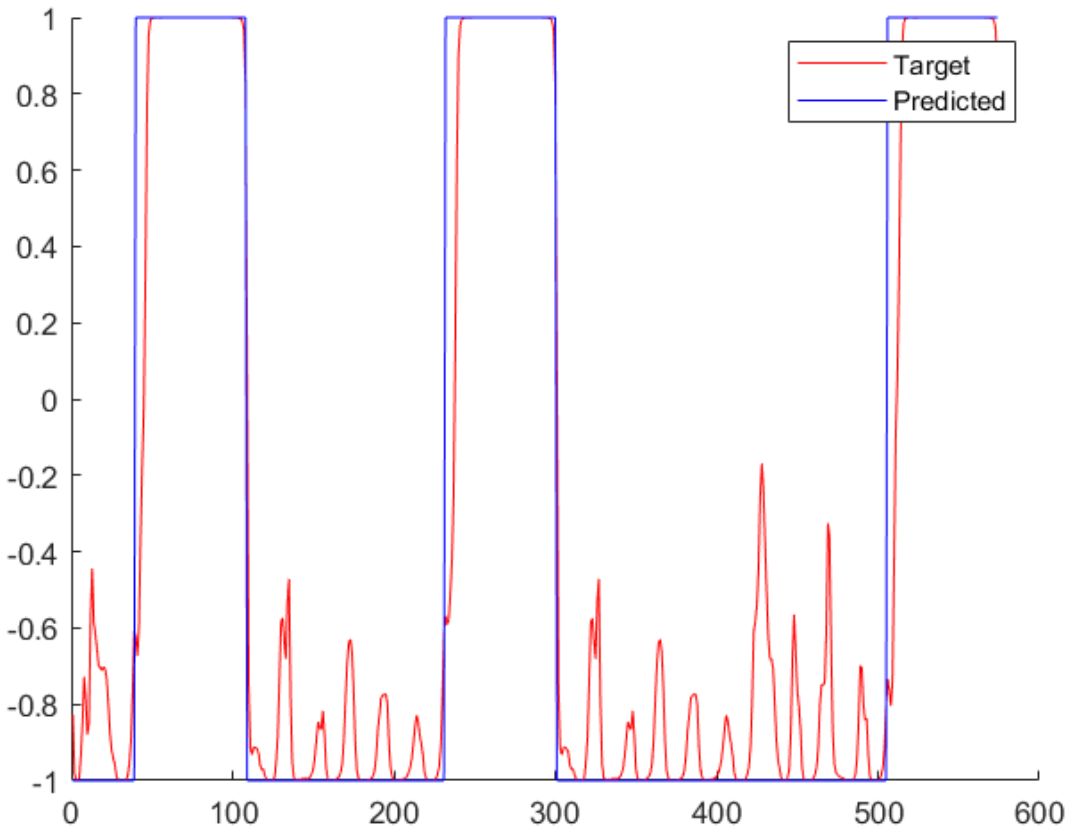
```
% Основной сигнал
k_1 = 0:0.025:1;
p_1 = sin(4*pi*k_1);

% Сигнал для распознавания распознаванию
k_2 = 2.38:0.025:4.1;
p_2 = cos(cos(k_2).*k_2.*k_2 + 5*k_2);

% Целевой выход
t_2 = ones(size(p_2));

% Целевой выход основного сигнала
t_1 = -ones(size(p_1));

% Длительности основного сигнала
R = {1; 3; 5};

% Формирование входного множества
P = [repmat(p_1, 1, R{1}), p_2, repmat(p_1, 1, R{2}), p_2, repmat(p_1, 1, R{3}),
p_2];
T = [repmat(t_1, 1, R{1}), t_2, repmat(t_1, 1, R{2}), t_2, repmat(t_1, 1, R{3}),
t_2];
Pseq = con2seq(P);
Tseq = con2seq(T);

% Создание и конфигурация сети
net = layrecnet(1 : 2, 100, 'trainoss');
net.layers{1}.transferFcn = 'tansig';
net.layers{2}.transferFcn = 'tansig';
net = configure(net, Pseq, Tseq);
```

```
% Подготовка массивов ячеек
[p, Xi, Ai, t] = preparets(net, Pseq, Tseq);

% Задаем параметры обучения:
% число эпох обучения, предельное значение критерия обучения
net.trainParam.epochs = 100;
net.trainParam.goal = 1.0e-5;

% Обучение сети
net = train(net, p, t, Xi, Ai);
Y = sim(net, p, Xi, Ai);

% Отображение структуры сети
view(net);

figure;
hold on;

rLine = plot(cell2mat(Y), 'r');
pLine = plot(cell2mat(t), 'b');
legend([rLine,pLine],'Target', 'Predicted');

% Преобразование значений
tc = sign(cell2mat(Y));

fprintf('Correctly recognized (train set): %d\\%d\n',nnz(tc == T(3 : end)),
length(T)-3);

% Формирование тестового множества
R = {1; 4; 5};
P = [repmat(p_1, 1, R{1}), p_2, repmat(p_1, 1, R{2}), p_2, repmat(p_1, 1, R{3}),
p_2];
T = [repmat(t_1, 1, R{1}), t_2, repmat(t_1, 1, R{2}), t_2, repmat(t_1, 1, R{3}),
t_2];

Pseq = con2seq(P);
Tseq = con2seq(T);

[p, Xi, Ai, t] = preparets(net, Pseq, Tseq);

% Рассчет выходя для тестового множетсва
Y = sim(net, p, Xi, Ai);

figure;
hold on;

rLine = plot(cell2mat(Y), 'r');
pLine = plot(cell2mat(t), 'b');
legend([rLine,pLine],'Target', 'Predicted');

% Преобразование значений
tc = sign(cell2mat(Y));

fprintf('Correctly recognized (test set): %d\\%d\n',nnz(tc == T(3 : end)), length(T)-
3);
```

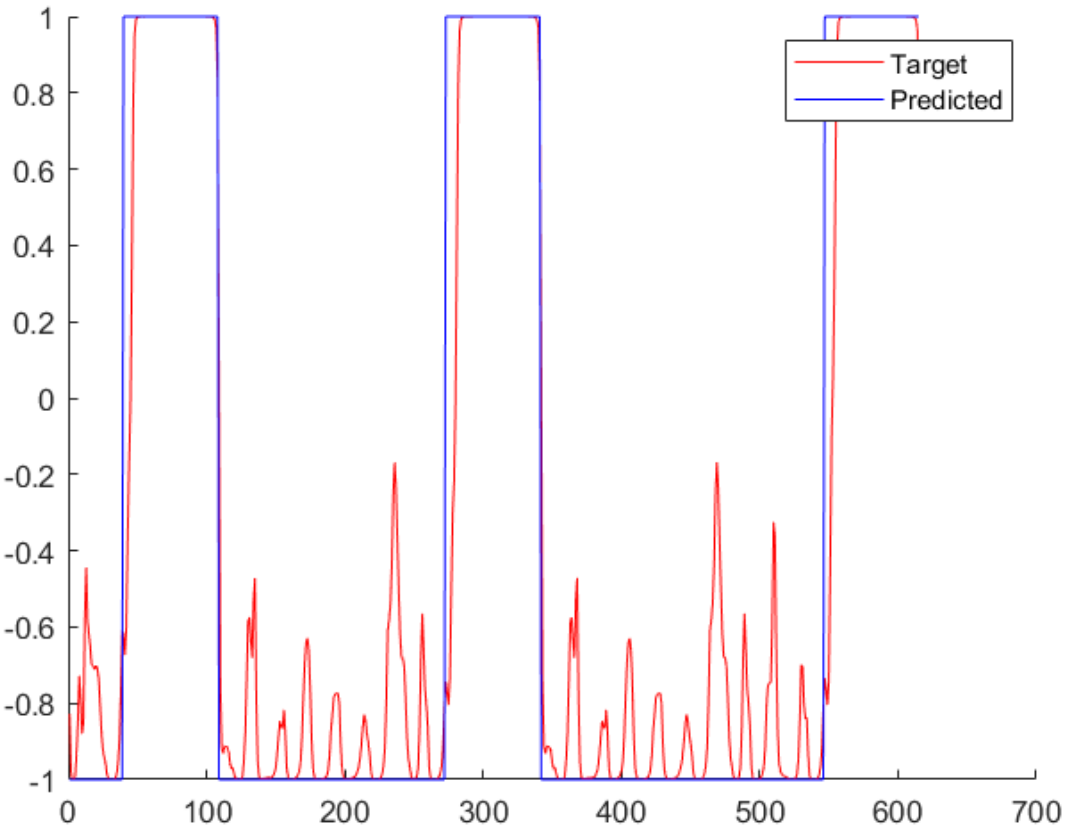
Структура сети



Выход сети для обучающего подмножества



Выход сети для тестового множества



Результаты распознавания:
Correctly recognized (train set): 554\573
Correctly recognized (test set): 593\614

Реализация сети Хопфилда

```
import numpy as np

class hopfield_nn:
    def fit(self, x_train):
        n_examples = x_train.shape[0]
        n_neurons = x_train.shape[1]
        self.W = np.zeros((n_neurons, n_neurons))
        for ex in x_train:
            self.W += np.outer(ex, ex)
        self.W /= n_neurons

    def predict_single(self, x):
        print(f'{x.shape=}')
        print(f'{self.W.shape=}')

        return np.sign(x @ self.W)

    def predict(self, x, epochs=600):
```

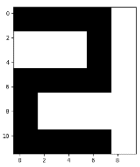
```
        n_examples = x.shape[0]
        converged = [False] * n_examples
        n_converged = 0

        cur = np.copy(x)
        for _ in range(epochs):
            for i in range(n_examples):
                if converged[i]:
                    continue

                pred = self.predict_single(cur[i])

                if (cur[i] == pred).all():
                    converged[i] = True
                    n_converged += 1

                if n_converged == n_examples:
                    break

                cur[i] = pred

        return cur
```

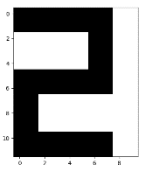
Задание №2

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import PIL

from hopfield_nn import hopfield_nn

def noise(x, noise_portion):
    size = x.shape[0]
    index = np.random.randint(0, size, int(noise_portion * size))
    res = np.copy(x)
    for i in index:
        res[i] = -1 if x[i] == 1 else 1

    return res

digits = [0, 1, 2, 3, 4, 6, 9]

images = []

for d in digits:
    img = PIL.Image.open(str(d) + '.bmp').convert('L')
    images.append(np.where(np.asarray(img) > 128, 1, -1))

plt.imshow(images[6], cmap=plt.get_cmap('Greys_r'))
```

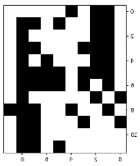
```
n_examples = len(digits)

x_train = []
for i in range(n_examples):
    x_train.append(images[i].flatten())
x_train = np.array(x_train)
print(f'{x_train.shape=}')

filter = hopfield_nn()
filter.fit(x_train)

initial_shape = (12, 10)

images_to_process = [2, 4, 1]
epochs = 600

img1 = x_train[images_to_process[0]][np.newaxis]
print(f'{img1.shape=}')
result = filter.predict(img1, epochs=epochs).reshape(initial_shape)
plt.imshow(result, cmap=plt.get_cmap('Greys_r'))
plt.show()

img2 = noise(x_train[images_to_process[1]], 0.2)[np.newaxis]
result = filter.predict(img2, epochs=epochs).reshape(initial_shape)
plt.imshow(result, cmap=plt.get_cmap('Greys_r'))
plt.show()

img3 = noise(x_train[images_to_process[2]], 0.3)[np.newaxis]
result = filter.predict(img3, epochs=epochs).reshape(initial_shape)
plt.imshow(result, cmap=plt.get_cmap('Greys_r'))
plt.show()
```

Тестовый образ 1

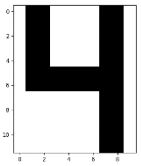




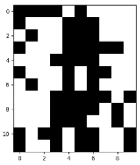
Тестовый образ 2 (зашумленный на 20%)



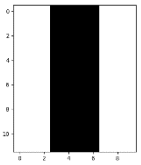
Результат фильтрации образа 2



Тестовый образ 3 (зашумленный на 20%)



Результат фильтрации образа 3



Реализация сети Хемминга

```
import numpy as np

def poslin(x):
    return np.maximum(0, x)

class hamming_nn:
    def fit(self, x_train):
        Q, R = x_train.shape
        self.IW = x_train.T
        self.b1 = np.full((1, Q), R)
        eps = 1 / (Q - 1)
        self.LW = np.full((Q, Q), -eps)
        for i in range(Q):
            self.LW[i][i] = 1.0

    def predict_single(self, x):
        return poslin(x @ self.LW)


    def predict(self, x, epochs=600):
        n_examples = x.shape[0]
        converged = [False] * n_examples
        n_converged = 0

        cur = x @ self.IW + self.b1
        for _ in range(epochs):
            for i in range(n_examples):
                if converged[i]:
                    continue
```

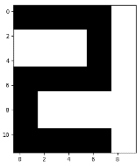
```
                pred = self.predict_single(cur[i])

                if (cur[i] == pred).all():
                    converged[i] = True
                    n_converged += 1

                if n_converged == n_examples:
                    break

                cur[i] = pred

        return cur
```

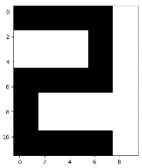
Задание №3

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import PIL

from hamming_nn import hamming_nn

def noise(x, noise_portion):
    size = x.shape[0]
    index = np.random.randint(0, size, int(noise_portion * size))
    res = np.copy(x)
    for i in index:
        res[i] = -1 if x[i] == 1 else 1

    return res

digits = [0, 1, 2, 3, 4, 6, 9]

images = []

for d in digits:
    img = PIL.Image.open(str(d) + '.bmp').convert('L')
    images.append(np.where(np.asarray(img) > 128, 1, -1))

plt.imshow(images[6], cmap=plt.get_cmap('Greys_r'))

n_examples = len(digits)

x_train = []
for i in range(n_examples):
    x_train.append(images[i].flatten())
x_train = np.array(x_train)
print(f'{x_train.shape=}')

filter = hamming_nn()
filter.fit(x_train)
```

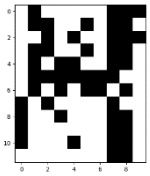
```
initial_shape = (12, 10)

images_to_process = [2, 4, 1]
epochs = 600

img1 = x_train[images_to_process[0]][np.newaxis]
print(f'{img1.shape=}')
result = filter.predict(img1, epochs=epochs)
result = images[np.argmax(result)]
plt.imshow(result, cmap=plt.get_cmap('Greys_r'))
plt.show()

img2 = noise(x_train[images_to_process[1]], 0.2)[np.newaxis]
result = filter.predict(img2, epochs=epochs)
result = images[np.argmax(result)]
plt.imshow(result, cmap=plt.get_cmap('Greys_r'))
plt.show()

img3 = noise(x_train[images_to_process[2]], 0.3)[np.newaxis]
result = filter.predict(img3, epochs=epochs)
result = images[np.argmax(result)]
plt.imshow(result, cmap=plt.get_cmap('Greys_r'))
plt.show()
```

Тестовый образ 1

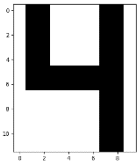




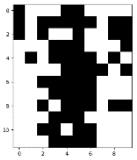
Тестовый образ 2 (зашумленный на 20%)

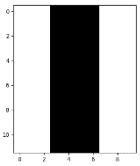


Результат фильтрации образа 2



Тестовый образ 3 (зашумленный на 20%) Результат фильтрации образа 3





Выводы:

Рекуррентные нейронные сети — вид нейронных сетей с циклическими связями. Благодаря им появляется возможность обработки последовательных данных, таких как временные ряды или тексты на естественном языке.
В отличие от многослойных перцептронов, рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины.

Процесс обучения сетей Элмана называется обратное распространение ошибки во времени. Алгоритм представляет собой градиентный спуск в пространстве весов в развернутой по времени реккурентной сети.
Обучение же сетей Хопфилда и Хемминга – довольно простые операции над матрицами.

Сеть Элмана подходит для анализа временныых рядов.

Сети Хопфилда и Хемминга отлично подходят для организации ассоциативной памяти.

Стоит отметить, существуют так же более продвинутые реккурентные нейронные сети такие как LSTM и GRU, которые более успешно решают задачи, связанные с моделированием и анализом временных рядов.