**Министерства науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования**

**«Заполярный государственный университет»**

**Кафедра информационных систем и технологий**

**Отчет**

**По дисциплине: «Нейронные сети»**

**«Лабораторная работа № 3»**

**Выполнил:**

студент группы ИС-19

Исрафилов Сабухи Мадад оглы

**Проверил преподаватель:**

Дыптан Елена Арнольдовна

Норильск 2022

## Цель работы

Реализовать в программе нейронную сеть Кохонена для классификации изображений.

## Описание архитектуры сети

**Сеть Кохонена** – класс нейронных сетей, основным элементом которых является слой Кохонена. Слой Кохонена состоит из адаптивных линейных сумматоров. Как правило, выходные сигналы слоя Кохонена обрабатываются по правилу «Победитель получает всё»: наибольший сигнал превращается в единичный, остальные обращаются в ноль.

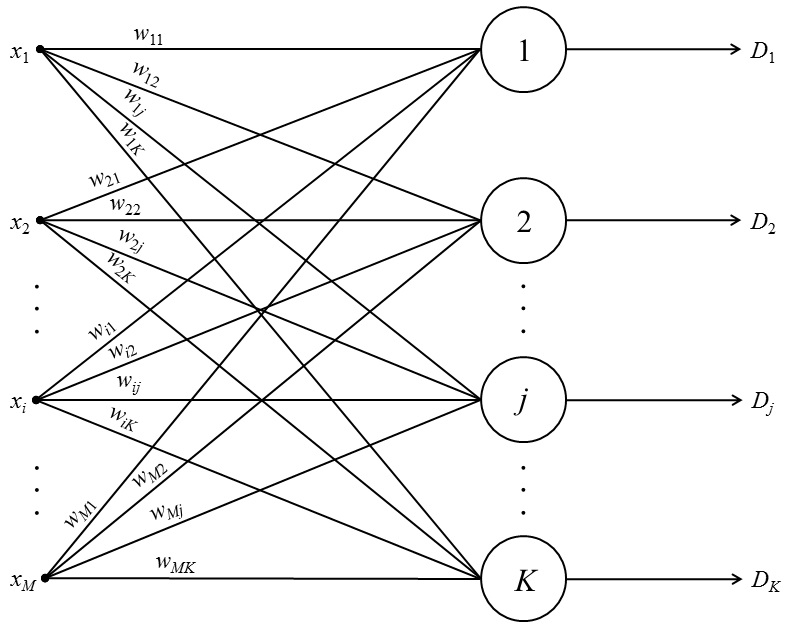


Рисунок 1 – концепция нейронной сети Кохонена

Где, Xn – входной вектор,

D – выходной вектор,

W – матрица весов

## Метод обучения

Алгоритм:

1. Инициализация сети. Весовым коэффициентам присваиваются малые случайные значения [-0.3; 0.3], задаётся начальный темп обучения a и D0 – максимальное расстояние между столбцами матрицы (либо вручную)
2. Подаётся новый входной сигнал
3. Вычисляется расстояние от входного нейрона вектора до всех нейронов сети
4. Выбирается нейрон победитель K с наименьшим расстоянием dk
5. Настройка весов нейрона K и всех нейронов, находящимся от него на расстоянии, не превышающем D
6. Уменьшаем значения a и D после каждого образа либо после эпохи
7. Повторять шаги 2-6, пока веса не перестанут меняться (пока суммарное изменение весов не будет очень мало)
8. Интерпретировать результат нейронной сети

## Практическая часть

**Задача:** реализовать в программе нейронную сеть Кохонена для классификации изображений (5 классов).

Набор данных представляет из себя рисунки в формате PNG размерностью 50 на 50.

В обучающей выборке 100 изображений (20 на каждый класс).

В тестовой выборке 10 изображений (2 на каждый класс).

Изображения нормализуются по rgb пикселям: если пиксель закрашен, то 1, иначе 0.

Образец тестовой выборки:

E:\projects\Kohanen_NN\test\квадрат 19.pngE:\projects\Kohanen_NN\test\круг 18.pngE:\projects\Kohanen_NN\test\треугольник 19.pngE:\projects\Kohanen_NN\test\ромб 20.png

В зависимости от изменения параметров обучения нейронная сеть реагирует по-разному, данные экспериментов представлены в таблице:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a (скорость обучения)** | **Скорость изменения a** | **Скорость изменения D** | **Эпох** | **Порог** | **Ошибка** |
| 0.3 | 0.9 | 0.9 | 71 | 0.05 | 0 |
| 0.3 | 0.99 | 0.99 | 200 | 0.1 | 0.2 |
| 0.5 | 0.9 | 0.9 | 82 | 0.05 | 0.4 |
| 0.7 | 0.9 | 0.9 | 77 | 0.08 | 0 |
| 0.7 | 0.99 | 0.9 | 200 | 0.1 | 0.2 |
| 0.9 | 0.9 | 0.9 | 83 | 0.05 | 0 |
| 0.9 | 0.99 | 0.99 | 200 | 0.05 | 0 |

## Листинг

class KohonenNeural:

    learning\_rate: float

    D: float = 1

    def \_\_init\_\_(self, input, clasters):

        self.weights = np.random.uniform(low=-0.3, high=0.3, size=(clasters, input))

    def predict(self, vector: np.ndarray):

        dist: np.ndarray =  np.power((vector - self.weights), 2).sum(axis=1)

        winner\_index = dist.argmin()

        return winner\_index

    def train(self, vector: np.ndarray):

        winner\_index = self.predict(vector)

        all\_dists: np.ndarray = np.zeros(5, dtype=np.float32)

        rows, \_ = self.weights.shape

        for index in range(0, rows):

            if index == winner\_index:

                continue

            else:

                all\_dists[index] = (np.power((vector - self.weights[index]), 2).sum())

        if self.D is None:

            max\_dist\_index = all\_dists.argmax()

            self.D = all\_dists[max\_dist\_index]

        all\_errors = []

        for index in range(0, len(all\_dists)):

            if index == winner\_index or all\_dists[index] < kn.D:

                delta: np.ndarray = self.learning\_rate \* (vector - self.weights[index])

                self.weights[index] += delta

                all\_errors.append(np.abs(delta))

        all\_errors = np.array(all\_errors)

        return all\_errors.sum()

def normalize(image: np.ndarray):

    new\_image = []

    for rgb in image:

        rgb: np.ndarray

        if (rgb == [255,255,255]).all():

            new\_image.append(0)

        else:

            new\_image.append(1)

    return np.array(new\_image)

dataset: List[tuple] = []

for file in listdir('data'):

    image = np.array(open(f'data/{file}'))

    x\_max, y\_max, \_ = image.shape

    image = image.reshape((x\_max\*y\_max, 3))

    image = normalize(image)

    dataset.append(tuple((file, image)))

test: List[tuple] = []

for file in listdir('test'):

    image = np.array(open(f'test/{file}'))

    x\_max, y\_max, \_ = image.shape

    image = image.reshape((x\_max\*y\_max, 3))

    image = normalize(image)

    test.append(tuple((file, image)))

kn = KohonenNeural(2500, 5)

epoch = 200

kn.learning\_rate = 0.8

all\_deltas = []

epoch\_count = 0

error\_counter = np.zeros(shape=5)

for i in range(epoch):

    shuffle(dataset)

    delta: float = 0

    for \_, image in dataset:

        delta += kn.train(image)

    delta = delta / len(dataset)

    all\_deltas.append(round(delta, 5))

    if (delta < 0.05): break

    epoch\_count += 1

    kn.learning\_rate \*= 0.9

    kn.D \*= 0.9

print('Обучающая выборка:')

all\_class = { 0: {}, 1: {}, 2: {}, 3: {}, 4: {} }

for filename, image in dataset:

    classes = kn.predict(image)

    default\_value = all\_class[classes].get(filename.split(' ')[0], 0)

    new\_value = default\_value + 1

    all\_class[classes][filename.split(' ')[0]] = new\_value

    print(f'{filename}: Класс {classes}')

correct\_classes = {}

for \_class in range(0, len(all\_class)):

    max\_key = max(all\_class[\_class], key=all\_class[\_class].get)

    correct\_classes[\_class] = max\_key

print(correct\_classes)

print(f'\nИзменения на эпохе {all\_deltas}')

print(f'Прошло эпох: {epoch\_count}')

print('Тестовая выборка:')

error = 0

for filename, image in test:

    classes = kn.predict(image)

    if filename.split(' ')[0] != correct\_classes[classes]: error += 1

    print(f'{filename}: Класс {classes}')

print(f'Ошибка на тестовой выборке: {error / len(test)}')

## Приложение

{0: 'ромб', 1: 'квадрат', 2: 'треугольник', 3: 'Прямоугольник', 4: 'круг'}

Прошло эпох: 83

Тестовая выборка:

квадрат 19.png: Класс 1

квадрат 20.png: Класс 1

круг 18.png: Класс 4

круг 19.png: Класс 4

Прямоугольник 19.png: Класс 3

Прямоугольник 20.png: Класс 3

ромб 19.png: Класс 0

ромб 20.png: Класс 0

треугольник 19.png: Класс 2

треугольник 20.png: Класс 2

Ошибка на тестовой выборке: 0.0