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

**«Заполярный Государственный Университет имени Н. М. Федоровского»**

**Кафедра информационных систем и технологий**

**Отчет**

**По дисциплине: «Нейронные сети»**

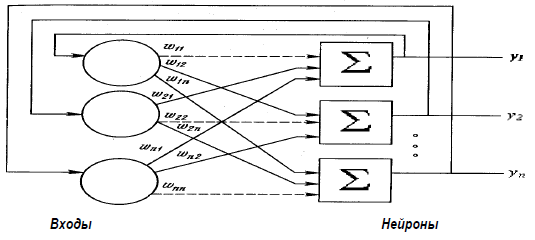
**«Лабораторная работа № 4. Сеть Хопфилда»**

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил студент группы ИС-21 |  | Сидельников Максим Эдуардович | |
| Преподаватель: |  | Дыптан Елена Арнольдовна | |
|  |  |  | |
|  |  |
|  |  | | |
|  | Дата защиты: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Оценка: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Норильск 2024

1. Теоретическая часть

Нейронная сеть Хопфилда (англ. Hopfield network) — полносвязная нейронная сеть с симметричной матрицей связей. В процессе работы динамика таких сетей сходится (конвергирует) к одному из положений равновесия. Эти положения равновесия определяются заранее в процессе обучения, они являются локальными минимумами функционала, называемого энергией сети. Такая сеть может быть использована как авто ассоциативная память, как фильтр, а также для решения некоторых задач оптимизации. В отличие от многих нейронных сетей, работающих до получения ответа через определённое количество тактов, сети Хопфилда работают до достижения равновесия, когда следующее состояние сети в точности равно предыдущему: начальное состояние является входным образом, а при равновесии получают выходной образ.



1. Схема сети Хопфилда с тремя нейронами.

Алгоритм обучения:

1. Вычисление коэффициентов основано на следующем правиле: для всех запомненных образов матрица связи должна удовлетворять уравнению

поскольку именно при этом условии состояния сети будут устойчивы — попав в такое состояние, сеть в нём и останется.

2. Запоминаемые векторы должны иметь бинарный вид. Расчёт весовых коэффициентов проводится по следующей формуле:

где N — размерность векторов, m — число запоминаемых выходных векторов, d — номер запоминаемого выходного вектора, — i-я компонента запоминаемого выходного j-го вектора.

3 Это выражение может стать более ясным, если заметить, что весовая матрица W может быть найдена вычислением внешнего произведения каждого запоминаемого вектора с самим собой и суммированием матриц, полученных таким образом. Это может быть записано в виде

где — i-й запоминаемый вектор-столбец.

Расчёт этих весовых коэффициентов и называется обучением сети, которое проводится только за одну эпоху.

Замечания, повышающие качество нейронной сети:

При обучении сети больше чем на двух образах могут получаться «химеры».

При выдачи сетью «химер» нужно проводить разобучение.

1. Практическая часть

Задача: написать программу, который сможет восстанавливать 3 изображения (единица, двойка и четвёрка). Реализация на языке программирования Python с преобразованием картинок попиксельно во входной вектор. Обучение сети — это построение одной большой матрицы по средству сложения матриц, полученных в итоге перемножения строк чисел с их транспонированным видом. По итогу обучения сеть сможет восстанавливать зашумлённые изображения размером 20 на 20 пикселей. В случае выдачи сетью химер производится разобучение, вносится изменение в матрицу весов во избежание появления химер.

  
Рисунок 2. Пример химеры



Рисунок 3. Зашумлённое изображение двойки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |

## Листинг

import sys

import os

import numpy as np

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QVBoxLayout, QHBoxLayout, QLabel, QSizePolicy

from PyQt5.QtGui import QPixmap, QImage, QFont

from PyQt5.QtCore import Qt

# Класс, реализующий сеть Хопфилда

class HopfieldNetwork:

    def \_\_init\_\_(self, size):

        self.size = size  # Задаем размер сети

        self.weights = np.zeros((size, size))  # Инициализируем матрицу весов нулями

    def train(self, patterns):

        for pattern in patterns:  # Для каждого паттерна

            self.weights += np.outer(pattern, pattern)  # Обновляем матрицу весов

        np.fill\_diagonal(self.weights, 0)  # Обнуляем диагональные элементы матрицы весов

    def update(self, state):

        return np.sign(np.dot(self.weights, state))  # Обновляем состояние сети

    def run(self, state):

        previous\_state = np.zeros\_like(state)  # Инициализируем предыдущее состояние

        flag = True  # Флаг для выхода из цикла

        while flag:  # Пока состояние не стабилизируется

            new\_state = self.update(state)  # Обновляем состояние

            if np.array\_equal(new\_state, previous\_state):  # Если состояние не изменилось

                flag = False  # Выходим из цикла

            previous\_state = state  # Обновляем предыдущее состояние

            state = new\_state  # Устанавливаем новое состояние

        return state  # Возвращаем стабилизированное состояние

    def unlearn(self, pattern):

        self.weights -= np.outer(pattern, pattern)  # Обновляем матрицу весов, вычитая вклад паттерна

        np.fill\_diagonal(self.weights, 0)  # Обнуляем диагональные элементы матрицы весов

# Функция для добавления шума в изображение

def add\_noise(image, noise\_level=0.1):

    noisy\_image = image.copy()

    num\_noisy\_pixels = int(noise\_level \* image.size)

    noisy\_indices = np.random.choice(image.size, num\_noisy\_pixels, replace=False)

    noisy\_image[noisy\_indices] \*= -1

    return noisy\_image

# Класс, реализующий графический интерфейс приложения

class HopfieldApp(QWidget):

    def \_\_init\_\_(self, network):

        super().\_\_init\_\_()

        self.network = network

        self.patterns = []

        self.current\_image = None

        self.current\_image\_index = -1

        self.noise\_level = 0.1

        self.initUI()

        self.load\_images()

    def initUI(self):

        layout = QVBoxLayout()

        self.label = QLabel(self)

        layout.addWidget(self.label)

        hbox = QHBoxLayout()

        self.image\_buttons = []

        for i in range(3):

            btn = QPushButton(f'Изображение {i + 1}', self)

            btn.clicked.connect(lambda \_, x=i: self.select\_image(x))

            hbox.addWidget(btn)

            self.image\_buttons.append(btn)

        layout.addLayout(hbox)

        self.selected\_image\_label = QLabel('Выбрано изображение: Нет', self)

        self.selected\_image\_label.setFont(QFont('Arial', 24))

        self.selected\_image\_label.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Preferred)

        self.selected\_image\_label.setAlignment(Qt.AlignCenter)

        layout.addWidget(self.selected\_image\_label)

        self.train\_button = QPushButton('Обучить', self)

        self.train\_button.clicked.connect(self.train)

        layout.addWidget(self.train\_button)

        self.run\_button = QPushButton('Запустить', self)

        self.run\_button.clicked.connect(self.run)

        layout.addWidget(self.run\_button)

        self.noise\_button = QPushButton('Добавить шум', self)

        self.noise\_button.clicked.connect(self.add\_noise)

        layout.addWidget(self.noise\_button)

        self.unlearn\_button = QPushButton('Разобучить', self)

        self.unlearn\_button.clicked.connect(self.unlearn)

        layout.addWidget(self.unlearn\_button)

        self.setLayout(layout)

        self.setWindowTitle('Сеть Хопфилда')

        self.show()

    def load\_images(self):

        folder = 'hob\_image'

        patterns = []

        for filename in os.listdir(folder):

            if filename.endswith('.bmp'):

                filepath = os.path.join(folder, filename)

                image = QImage(filepath)

                if image.width() == 20 and image.height() == 20:

                    pattern = self.image\_to\_pattern(image)

                    patterns.append(pattern)

        self.patterns = patterns

        print("Изображения загружены")

    def select\_image(self, index):

        if index < len(self.patterns):

            self.current\_image = self.patterns[index]

            self.current\_image\_index = index

            self.noise\_level = 0.1

            self.selected\_image\_label.setText(f'Выбрано изображение: {index + 1}')

            self.display\_image(self.current\_image)

    def image\_to\_pattern(self, image):

        gray\_image = image.convertToFormat(QImage.Format\_Grayscale8)

        buffer = gray\_image.bits()

        buffer.setsize(gray\_image.byteCount())

        array = np.frombuffer(buffer, dtype=np.uint8).reshape((20, 20))

        pattern = np.where(array > 127, 1, -1).flatten()

        return pattern

    def train(self):

        self.network.train(self.patterns)

        print("Обучение завершено")

    def run(self):

        if self.current\_image is not None:

            result = self.network.run(self.current\_image)

            self.display\_image(result)

    def add\_noise(self):

        if self.current\_image is not None:

            self.noise\_level += 0.1

            noisy\_image = add\_noise(self.current\_image, self.noise\_level)

            self.display\_image(noisy\_image)

    def unlearn(self):

        if self.current\_image is not None:

            self.network.unlearn(self.current\_image)

            print("Разобучение завершено")

    def display\_image(self, image):

        img = (image.reshape(20, 20) \* 255).astype(np.uint8)

        qimage = QImage(img, 20, 20, QImage.Format\_Grayscale8)

        pixmap = QPixmap.fromImage(qimage).scaled(100, 100, Qt.KeepAspectRatio)

        self.label.setPixmap(pixmap)

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    network = HopfieldNetwork(400)

    ex = HopfieldApp(network)

    sys.exit(app.exec\_())