

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА** - **Российский технологический университет"**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКИЕ РАБОТЫ № 1-10**

**по дисциплине**

**«Проектирование и обучение нейронных сетей»**

Студент группы: ИКБО-14-20 Вежновец Ф.Ю. *(Ф.И.О.студента)*

Руководитель \_\_Гуличева А.А.\_\_

*(Ф.И.О. преподавателя)*

Москва 2023

ОГЛАВЛЕНИЕ

[введение 8](#_Toc151625557)

[1. ОБУЧЕНИЕ ПО ПРАВИЛАМ ХЕББА 9](#_Toc151625558)

[1.1 Постановка задачи 9](#_Toc151625559)

[1.2 Описание обучения по правилам ХЕББА 9](#_Toc151625560)

[1.3 Результат работы 10](#_Toc151625561)

[2. ДЕЛЬТА ПРАВИЛО 11](#_Toc151625562)

[2.1 Постановка задачи 11](#_Toc151625563)

[2.2 Описание обучения по дельта правилу 11](#_Toc151625564)

[2.3 Результат работы 12](#_Toc151625565)

[3. ОБРАТНОЕ РАССПРОСТАНЕНИЕ ОШИБКИ 14](#_Toc151625566)

[3.1 Постановка задачи 14](#_Toc151625567)

[3.2 Описание алгоритма обратного распространения ошибки 14](#_Toc151625568)

[3.3 Результат работы 15](#_Toc151625569)

[4. РАДИАЛЬНО БАЗИСНЫЕ ФУНКЦИИ 17](#_Toc151625570)

[4.1 Постановка задачи 17](#_Toc151625571)

[4.2 Описание нейросетей с радиально базисными функциями 17](#_Toc151625572)

[4.3 Результат работы 18](#_Toc151625573)

[5. КАРТЫ КОХОНЕНА 20](#_Toc151625574)

[5.1 Постановка задачи 20](#_Toc151625575)

[5.2 Описание карт Кохонена 20](#_Toc151625576)

[5.3 Результат работы 21](#_Toc151625577)

[6. ВСТРЕЧНОЕ РАСПРОСТРАНЕНИЕ 22](#_Toc151625578)

[6.1 Постановка задачи 22](#_Toc151625579)

[6.2 Описание нейронной сети встречного распространения 22](#_Toc151625580)

[6.3 Результат работы 23](#_Toc151625581)

[7. РЕКУРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ 25](#_Toc151625582)

[7.1 Постановка задачи 25](#_Toc151625583)

[7.2 Описание рекуррентной нейронной сети 25](#_Toc151625584)

[7.3 Результат работы 26](#_Toc151625585)

[8. СВЁРТОЧНАЯ НЕЙРОННАЯ СЕТЬ 27](#_Toc151625586)

[8.1 Постановка задачи 27](#_Toc151625587)

[8.2 Описание свёрточой нейронной сети 27](#_Toc151625588)

[8.3 Результат работы 28](#_Toc151625589)

[9. ТРАНСФОРМЕР 30](#_Toc151625590)

[9.1 Постановка задачи 30](#_Toc151625591)

[9.2 Описание трансформера 30](#_Toc151625592)

[9.3 Результат работы 31](#_Toc151625593)

[10. ГРАФОВЫЕ НЕЙРОННЫЕ СЕТИ 33](#_Toc151625594)

[10.1 Постановка задачи 33](#_Toc151625595)

[10.2 Описание графовую нейронную сеть 33](#_Toc151625596)

[10.3 Результат работы 34](#_Toc151625597)

[Вывод 36](#_Toc151625598)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 37](#_Toc151625599)

[ПРИЛОЖЕНИЯ 38](#_Toc151625600)

введение

В современном мире, нейронные сети играют ключевую роль во многих сферах деятельности: анализ данных, распознавание образов, компьютерная визуализация и многое другое. С помощью нейронных сетей можно обрабатывать большие объемы данных, извлекать из них полезную информацию и делать выводы.

Обучение нейронных сетей — это процесс, при котором сеть учится распознавать закономерности в данных и использовать их для принятия решений. Для этого используются различные алгоритмы обучения, такие как правило Хебба, дельта-правило и обратное распространение ошибки, и другие. В данном курсе мы рассмотрим каждый из этих алгоритмов и научимся применять их на практике.

1. ОБУЧЕНИЕ ПО ПРАВИЛАМ ХЕББА
   1. Постановка задачи

Пользуясь лекционными материалами и методическими указаниями реализовать Обучение по правилам Хебба.

* 1. Описание обучения по правилам ХЕББА

Первое правило Хебба – если сигнал персептрона неверен и равен нулю, то необходимо увеличить веса тех входов, на которые была подана единица.

Вытекает первый тип ошибки: на выходе персептрона 0, правильный ответ 1. Для того, чтобы персептрон выдавал правильный ответ необходимо, чтобы скалярное произведение стало больше. Поскольку переменные принимают значения 0 или 1, увеличение суммы может быть достигнуто за счет увеличения весов. Однако нет смысла увеличивать веса при переменных, которые равны нулю. Увеличиваем веса только при тех, которые равны 1. Для закрепления единичных сигналов с весов, следует провести ту же процедуру и на всех остальных слоях.

Второе правило Хебба — если сигнал персептрона неверен и равен единице, то необходимо уменьшить веса тех входов, на которые была подана единица.

Отсюда вытекает второй тип ошибки: на выходе персептрона 1, правильный ответ 0. Для уменьшения скалярного произведения в правой части, необходимо уменьшить веса связей при тех переменных, которые равны 1. Необходимо также провести эту процедуру для всех активных нейронов предыдущих слоев.

Алгоритм обучения по правилу Хебба сводится к следующей последовательности действий:

* 1. Инициализация весовых коэффициентов и порогов случайными значениями, близкими к нулю (чтобы сеть сразу не могла войти в насыщение).
  2. Подача на вход НС очередного входного образа.
  3. Вычисление значения выхода.
  4. Если значение выхода не совпадает с эталонным значением, то происходит модификация коэффициентов в соответствии с формулами при скорости обучения η=1. В противном случае осуществляется переход к пункту 5.
  5. Если не все вектора из обучающей выборки были поданы на вход НС, то происходит переход к пункту 2. Иначе, переход к пункту 6.
  6. Останов.

1.3 Результат работы

Создадим простейший перцептрон для распознавания единицы и нуля. Для обучения модели будем использовать алгоритм обучения Хебба.

Перцептрон реализован на языке Python (Листинг 1.1 – 1.3) результат выполнения приведен на Рисунке 1.1.

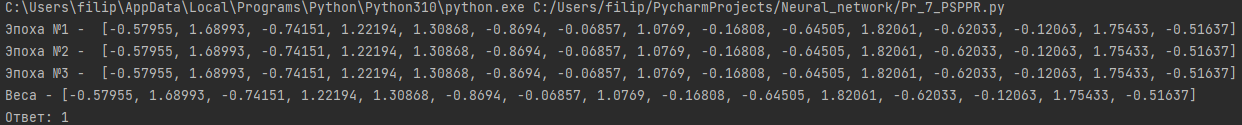


Рисунок 1.1 – Результат обучения

Заданная картинка для распознавания приведена на Рисунке 1.2.



Рисунок 1.2 – Входные данные

2. ДЕЛЬТА ПРАВИЛО

2.1 Постановка задачи

Пользуясь лекционными материалами и методическими указаниями реализовать обучение по дельта правилу.

2.2 Описание обучения по дельта правилу

Алгоритм обучения Розенблатта (дельта-правило) сводится к следующей последовательности действий:

1. Инициализация весовых коэффициентов и порогов значениями, близкими к нулю.
2. Подача на вход нейронной сети очередного входного образа (входного вектора Х), взятого из обучающей выборки, и вычисление суммарного сигнала по всем входам для каждого нейрона j:

, где n – размерность входного вектора, – i-я компонента входного вектора,

– весовой коэффициент связи нейрона j и входа i.

1. Вычисление значения выхода каждого нейрона:

где – порог, соответствующий нейрону j

1. Вычисление значения ошибки обучения для каждого нейрона
2. Проводится модификация весового коэффициента связи по формуле
3. Повторение пунктов 2 – 5 до тех пор, пока ошибка сети не станет меньше заданной.

2.3 Результат работы

Опишем предметную область.

Высказывания:

– сборщик установил систему охлаждения;

– сборщик одел кулер на башню;

– сборщик подключил кулер;

– сборщик прикрутил башню;

a – сборщик нанёс термопасту.

Аксиомы:

– чтобы написать экзамен, надо получить справку и найти кабинет;

Целевая логическая функция:

Для данной логической функции напишем таблицу истинности (Таблица 2.1).

Таблица 2.1 – Таблица истинности для целевой истинности

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

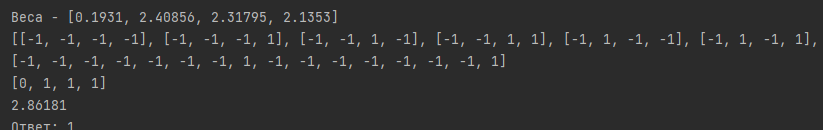


Рисунок 2.1 – Результат обучения

3. ОБРАТНОЕ РАССПРОСТАНЕНИЕ ОШИБКИ

3.1 Постановка задачи

Пользуясь лекционными материалами и методическими указаниями реализовать обучение при помощи обратного распространения ошибки.

3.2 Описание алгоритма обратного распространения ошибки

Обратное распространение ошибки (backpropagation) — это метод, используемый для обучения нейронных сетей. Он основан на принципе градиентного спуска, который позволяет минимизировать ошибку сети путем корректировки весов сети.

**Принцип работы.**

Обратное распространение ошибки работает следующим образом:

* На вход сети подается обучающее значение.
* Сеть производит выходной сигнал.
* Ошибка сети рассчитывается как разница между выходным сигналом и целевым значением.
* Градиент ошибки рассчитывается для каждого веса сети.
* Веса сети обновляются в направлении, противоположном градиенту ошибки.

Процесс повторяется для всех обучающих значений.

**Обучение нейронной сети.**

Обучение нейронной сети с помощью обратного распространения ошибки можно разделить на два этапа:

* Инициализация весов сети: на этом этапе веса сети инициализируются случайными значениями.
* Обучение сети: на этом этапе веса сети обновляются с помощью обратного распространения ошибки.

Инициализация весов сети является важным этапом, который определяет скорость и эффективность обучения сети.

Обучение нейронной сети может быть длительным процессом, особенно для сетей с большим количеством параметров.

3.3 Результат работы

Архитектура сети состоит из 15 входных нейронов, 5 нейронов в скрытом слое и 1 выходной нейрон (рисунок 3.1).

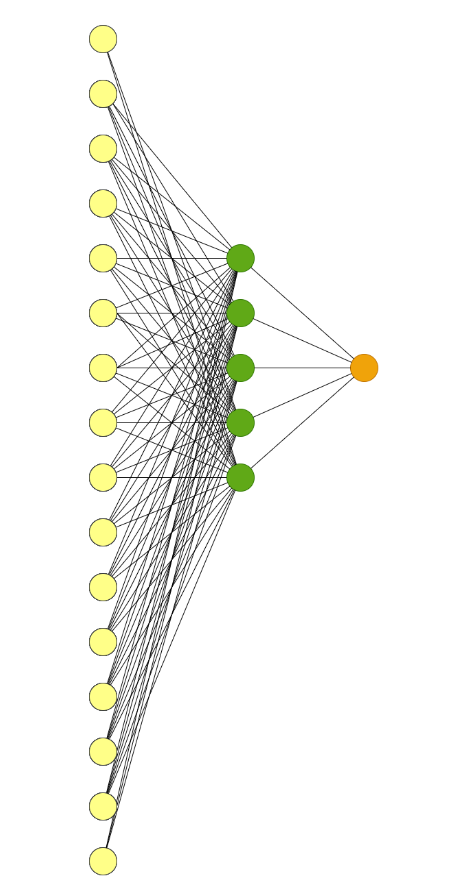


Рисунок 3.1 – Архитектура нейронной сети

Будем распознавать цифру 1 (скобочка). В качестве функции активацию на всех слоях будем использовать сигмойду.

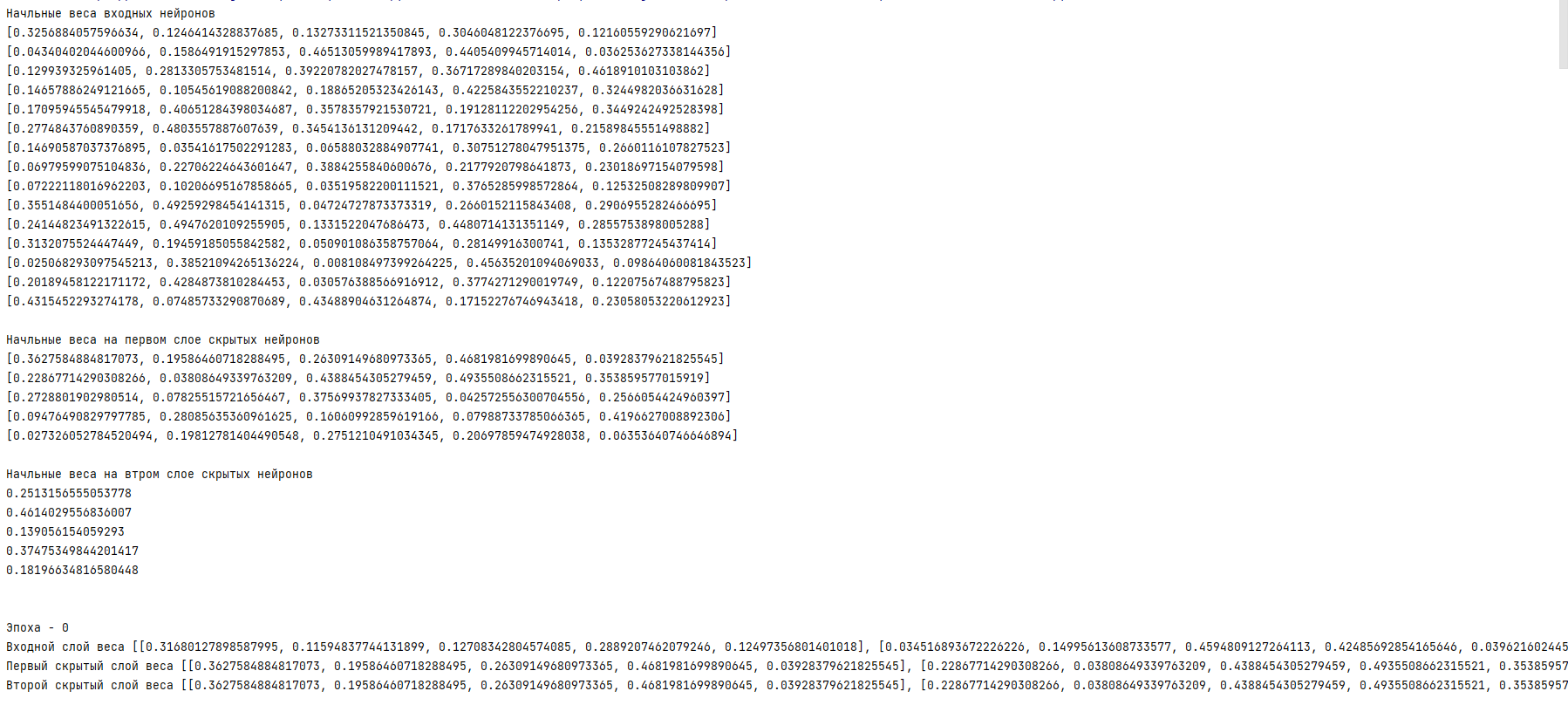


Рисунок 3.2 – Результат работы программы

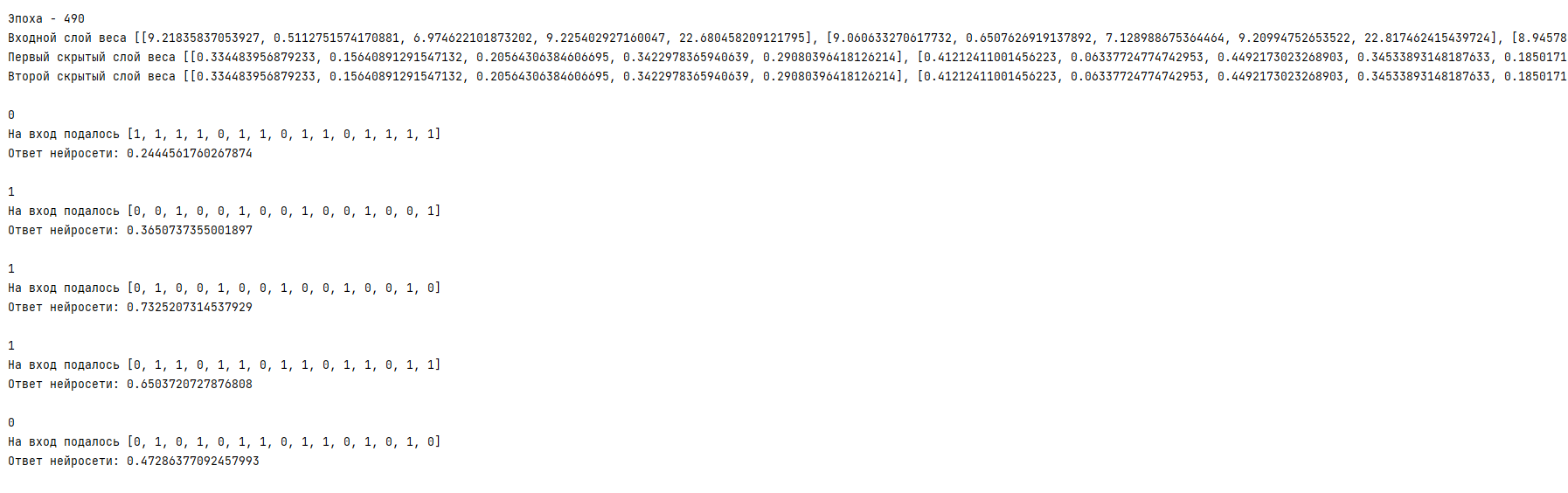


Рисунок 3.3 – Результат работы программы

Нейронная сеть правильно классифицирует единицу и обучается при помощи обратного распространения ошибки.

4. РАДИАЛЬНО БАЗИСНЫЕ ФУНКЦИИ

4.1 Постановка задачи

Пользуясь лекционными материалами и методическими указаниями реализовать нейронную сеть с радиально базисными функциями.

4.2 Описание нейросетей с радиально базисными функциями

Нейронная сеть с радиально базисными функциями (RBF) — это тип искусственной нейронной сети, которая использует радиальные базисные функции (RBF) как функции активации. RBF – это нелинейные функции, которые имеют локальный характер, то есть они имеют большие значения вблизи своего центра и малые значения вдали от него.

Архитектура нейросети с RBF обычно состоит из двух слоев:

* Входной слой: Этот слой получает входные данные.
* Скрытый слой: Этот слой состоит из RBF.
* Выходной слой: Этот слой производит выходной сигнал.

Обучение нейросети с RBF заключается в настройке параметров RBF, чтобы минимизировать ошибку между выходами сети и целевыми значениями.

RBF-сети обладают рядом преимуществ по сравнению с другими типами нейронных сетей. Они являются универсальными аппроксиматорами, то есть они могут аппроксимировать любую непрерывную функцию с произвольной точностью. Кроме того, они устойчивы к шуму в данных.

RBF-сети находят широкое применение в различных областях, включая:

* Функция приближения: RBF-сети могут использоваться для аппроксимации сложных функций, таких как функции распределения вероятностей.
* Классификация: RBF-сети могут использоваться для классификации объектов по их характеристикам.
* Регрессия: RBF-сети могут использоваться для прогнозирования значений зависимой переменной по значениям независимых переменных.
* Системы управления: RBF-сети могут использоваться для управления сложными системами.

Описание обучения нейросети с радиально базисными функциями (RBF):

Этап 1: Выбор центроидов

Центры RBF выбираются из обучающих данных таким образом, чтобы они равномерно покрывали область ввода.

Этап 2: Оптимизация параметров RBF

Параметры RBF, такие как ширина RBF и веса выходного слоя, оптимизируются с помощью методов оптимизации, таких как градиентный спуск или генетические алгоритмы.

В целом, обучение RBF-сетей заключается в настройке параметров RBF, чтобы минимизировать ошибку между выходами сети и целевыми значениями.

4.3 Результат работы

Будем обучать нейронную сеть на датасете ирисов (Рисунок 4.1).

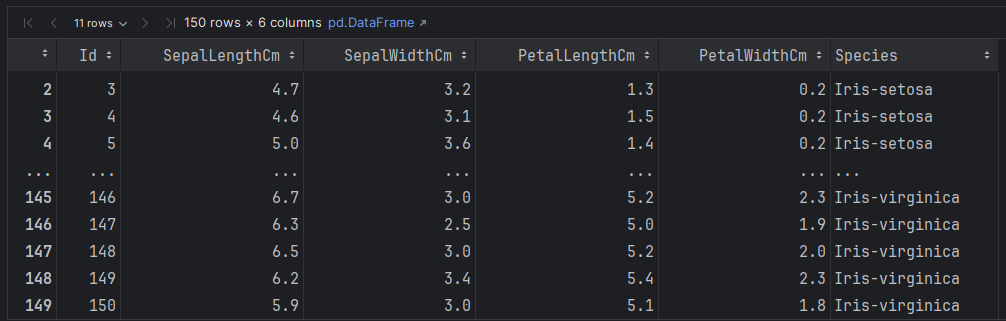


Рисунок 4.1 – Датасет

Обучаться будем 500 эпох, с шагом обучения 0.01 (Рисунок 4.2).

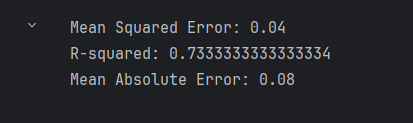


Рисунок 4.2 – Полученная точность на тестовой выборке

5. КАРТЫ КОХОНЕНА

5.1 Постановка задачи

Пользуясь лекционными материалами и методическими указаниями реализовать карты Кохонена.

5.2 Описание карт Кохонена

Карты Кохонена, также известные как самоорганизующиеся карты (SOM), представляют собой тип нейронной сети с обучением без учителя, который используется для задач кластеризации и визуализации.

Карта Кохонена состоит из двух слоев нейронов:

* Входной слой: Этот слой получает входные данные.
* Скрытый слой: Этот слой состоит из нейронов, которые называются нейронами Кохонена.

Нейронные клетки Кохонена имеют нелинейные функции активации, которые обычно принимают форму гауссиана.

**Обучение.**

Обучение карты Кохонена происходит следующим образом:

* Каждый нейрон скрытого слоя назначается одному из кластеров.
* Для каждого входного вектора определяется нейрон Кохонена, который имеет минимальное расстояние до входного вектора. Этот нейрон называется победителем.
* Веса победителя и его соседей корректируются таким образом, чтобы уменьшить расстояние между ними и входным вектором.

Процесс обучения повторяется для всех входных векторов обучающей выборки.

**Принцип работы.**

При обучении карты Кохонена происходит постепенное формирование кластеров в пространстве входных данных. Каждый нейрон Кохонена отвечает за один из кластеров. Веса нейронов Кохонена отражают центр кластера, за который они отвечают. В результате обучения карта Кохонена становится двумерной картой, на которой кластеры представлены отдельными областями.

5.3 Результат работы

Задача будет нетривиальной, будем кластеризовать данные на 2 класса, на вход подаются 2 переменные от 0 до 1 (Рисунок 5.1). Шаг обучения = 0.1, количество эпох 10.

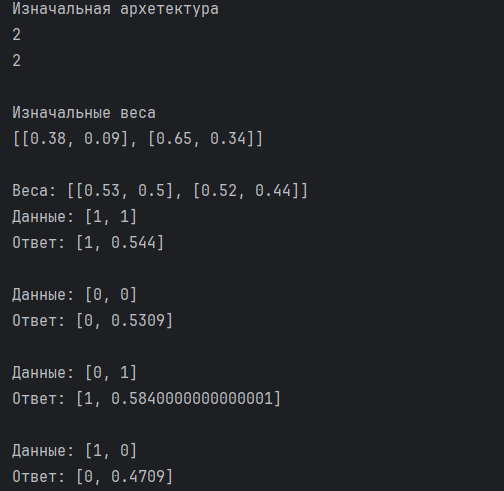


Рисунок 5.1 – Результат работы карт Кохонена.

6. ВСТРЕЧНОЕ РАСПРОСТРАНЕНИЕ

6.1 Постановка задачи

Пользуясь лекционными материалами и методическими указаниями реализовать нейронную сеть встречного распространения.

6.2 Описание нейронной сети встречного распространения

Нейронная сеть встречного распространения — это тип нейронной сети с обучением с учителем, которая используется для решения задач классификации и регрессии.

Сеть встречного распространения состоит из трех слоев нейронов:

* Входной слой: Этот слой получает входные данные.
* Скрытый слой: Этот слой состоит из нейронов, которые обрабатывают входные данные.
* Выходной слой: Этот слой производит выходной сигнал.

Нейронные клетки входного слоя имеют линейные функции активации. Нейронные клетки скрытого слоя имеют нелинейные функции активации, такие как сигмоида или гиперболический тангенс. Нейронные клетки выходного слоя имеют линейные функции активации.

**Обучение.**

Обучение сети ВР происходит следующим образом:

* На вход сети подается обучающий вектор.
* Сеть производит выходной сигнал.
* Сравнивается выходной сигнал с целевым значением.
* Ошибка сети рассчитывается как разница между выходным сигналом и целевым значением.
* Веса сети обновляются таким образом, чтобы уменьшить ошибку сети.
* Процесс обучения повторяется для всех обучающих векторов.

**Принцип работы.**

При обучении сети ВР веса сети настраиваются таким образом, чтобы минимизировать ошибку сети. Ошибка сети рассчитывается как разница между выходным сигналом сети и целевым значением.

Веса сети обновляются с помощью метода градиентного спуска. Градиентный спуск — это метод оптимизации, который использует градиент ошибки для обновления параметров сети.

6.3 Результат работы

Будем обучать нейронную сеть на датасете ирисов (Рисунок 6.1). Количество эпох = 100, изначальные шаги обучения для а и б = 0.1, коэффициент а уменьшается на 20% после каждого примера. Точность на обучающих и тестовых данных представлена на рисунке 6.2 – 6.3.

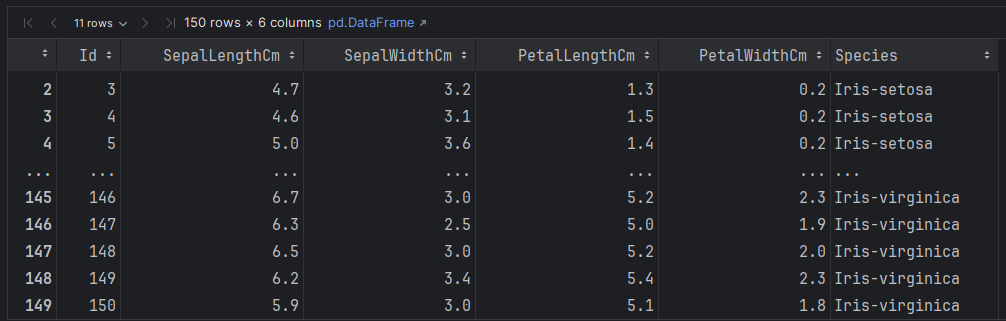


Рисунок 6.1 – Датасет

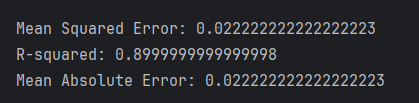


Рисунок 6.2 – Точность на обучающих данных

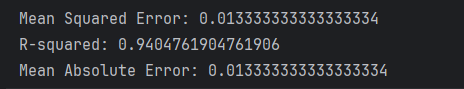


Рисунок 6.3 – Точность на тестовых данных

7. РЕКУРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ

7.1 Постановка задачи

Пользуясь лекционными материалами и методическими указаниями реализовать рекуррентную нейронную сеть.

7.2 Описание рекуррентной нейронной сети

Рекуррентная нейронная сеть — это тип нейронной сети, в которой связи между нейронами образуют направленные петли. Благодаря этому рекуррентная нейронная сеть может обрабатывать последовательные данные, такие как временные ряды или речевые сигналы.

**Структура.**

Рекуррентная нейронная сеть обычно состоит из следующих слоев:

* Входной слой: Этот слой получает последовательные данные.
* Скрытый слой: Этот слой состоит из нейронов, которые обрабатывают последовательные данные.
* Выходной слой: Этот слой производит выходной сигнал.

Нейронные клетки скрытого слоя имеют нелинейные функции активации, такие как сигмоида или гиперболический тангенс.

**Обучение.**

Обучение РНС происходит следующим образом:

* На вход сети подается последовательность обучающих данных.
* Сеть производит выходной сигнал.
* Сравнивается выходной сигнал с целевым значением.
* Ошибка сети рассчитывается как разница между выходным сигналом и целевым значением.
* Веса сети обновляются таким образом, чтобы уменьшить ошибку сети.

Процесс обучения повторяется для всех обучающих последовательностей.

**Принцип работы.**

При обучении РНС веса сети настраиваются таким образом, чтобы минимизировать ошибку сети. Ошибка сети рассчитывается как разница между выходным сигналом сети и целевым значением.

Веса сети обновляются с помощью метода градиентного спуска. Градиентный спуск — это метод оптимизации, который использует градиент ошибки для обновления параметров сети.

7.3 Результат работы

Будем обучать нейронную сеть на своём датасете цены биткоина. Количество эпох = 50, изначальные шаги обучения = 0.01. Результат работы программы представлен на рисунке 7.1.

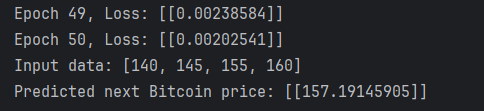


Рисунок 7.1 – Результат обучения рекуррентной нейронной сети

8. СВЁРТОЧНАЯ НЕЙРОННАЯ СЕТЬ

8.1 Постановка задачи

Пользуясь лекционными материалами и методическими указаниями реализовать свёрточную нейронную сеть нейронную сеть.

8.2 Описание свёрточой нейронной сети

Свёрточная нейронная сеть (CNN) – это тип нейронной сети, которая использует сверточные операции для обработки данных. Свёрточные операции позволяют сети обнаруживать пространственные закономерности в данных, что делает их особенно эффективными для задач, связанных с изображениями и видео.

**Структура.**

CNN обычно состоит из следующих слоев:

* Входной слой: Этот слой получает изображения или видео.
* Свёрточный слой: Этот слой выполняет свёрточные операции над входными данными.
* Постсвёрточный слой: Этот слой выполняет операции над выходом свёрточного слоя.
* Выходной слой: Этот слой производит выходной сигнал.

Свёрточный слой состоит из нейронов, которые имеют локальную область ввода. Свёрточная операция выполняется следующим образом:

* Каждый нейрон свёрточного слоя принимает на вход область ввода.
* Веса каждого нейрона свёрточного слоя умножают соответствующие элементы области ввода.
* Результаты умножения суммируются.
* Суммирование производится с использованием функции активации.

**Обучение.**

Обучение CNN происходит следующим образом:

* На вход сети подается обучающее изображение или видео.
* Сеть производит выходной сигнал.
* Сравнивается выходной сигнал с целевым значением.
* Ошибка сети рассчитывается как разница между выходным сигналом и целевым значением.
* Веса сети обновляются таким образом, чтобы уменьшить ошибку сети.

Процесс обучения повторяется для всех обучающих изображений или видео.

**Принцип работы.**

При обучении CNN веса сети настраиваются таким образом, чтобы минимизировать ошибку сети. Ошибка сети рассчитывается как разница между выходным сигналом сети и целевым значением.

Веса сети обновляются с помощью метода градиентного спуска. Градиентный спуск — это метод оптимизации, который использует градиент ошибки для обновления параметров сети.

8.3 Результат работы

Будем обучать нейронную сеть на датасете Fashion-MNIST (Рисунок 8.1).

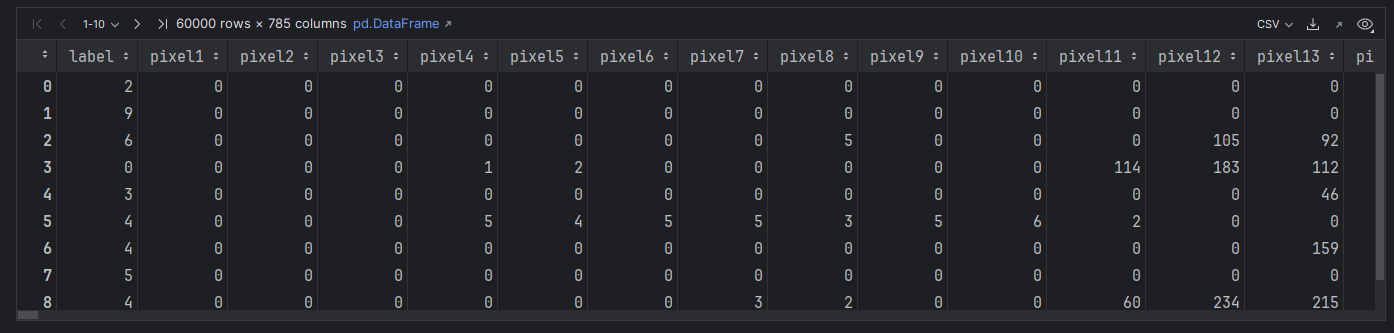


Рисунок 8.1 – Датасет

Обучаться будем всего 1 эпоху, с шагом обучения 0.1 (Рисунок 8.2).

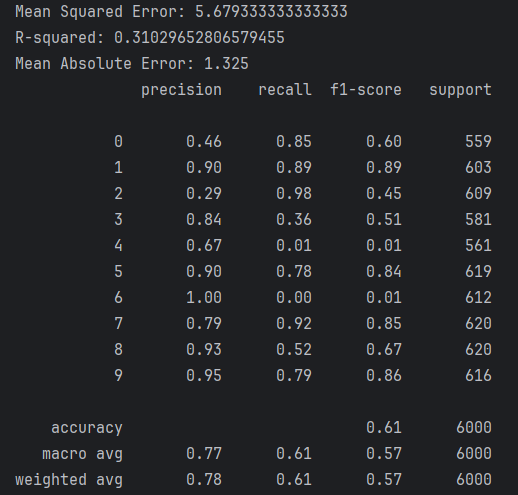


Рисунок 8.2 – Точность предсказаний нейросети



Рисунок 8.3 – Confusion Матрица

9. ТРАНСФОРМЕР

9.1 Постановка задачи

Пользуясь лекционными материалами и методическими указаниями реализовать трансформер.

9.2 Описание трансформера

Трансформер — это тип нейронной сети, который использует механизм внимания для обработки последовательностей данных. Механизм внимания позволяет сети учитывать все элементы последовательности одновременно, что делает его особенно эффективным для задач, связанных с естественным языком.

**Структура.**

Трансформер обычно состоит из следующих слоев:

* Входной слой: Этот слой получает последовательность данных.
* Механизм внимания: Этот слой выполняет операцию внимания над входными данными.
* Выходной слой: Этот слой производит выходной сигнал.

Механизм внимания работает следующим образом:

* Для каждого элемента последовательности рассчитывается внимание к другим элементам последовательности.
* Внимание рассчитывается с использованием функции внимания.
* Отчеты внимания используются для обновления выходов механизма внимания.

**Обучение.**

Обучение трансформера происходит следующим образом:

* На вход сети подается обучающая последовательность данных.
* Сеть производит выходной сигнал.
* Сравнивается выходной сигнал с целевым значением.
* Ошибка сети рассчитывается как разница между выходным сигналом и целевым значением.
* Веса сети обновляются таким образом, чтобы уменьшить ошибку сети.

Процесс обучения повторяется для всех обучающих последовательностей.

**Принцип работы.**

При обучении трансформера веса сети настраиваются таким образом, чтобы минимизировать ошибку сети. Ошибка сети рассчитывается как разница между выходным сигналом сети и целевым значением.

Веса сети обновляются с помощью метода градиентного спуска. Градиентный спуск — это метод оптимизации, который использует градиент ошибки для обновления параметров сети.

9.3 Результат работы

Будем обучать трансформер на данных о температуре (Рисунок 9.1).

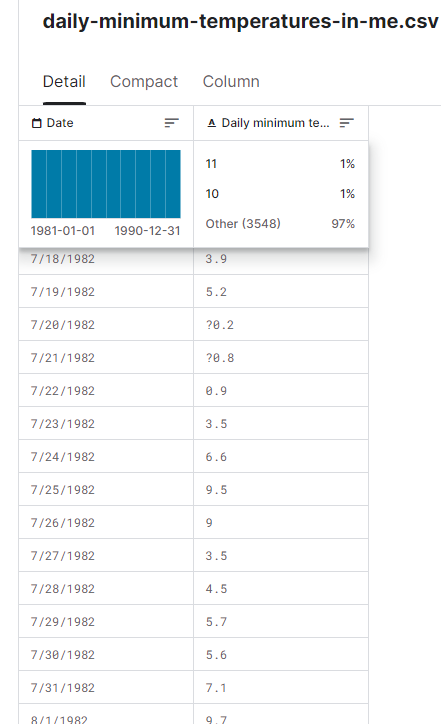


Рисунок 9.1 – Датасет

Обучать датасет будем 100 эпох, результат обучения Рисунок 9.2.

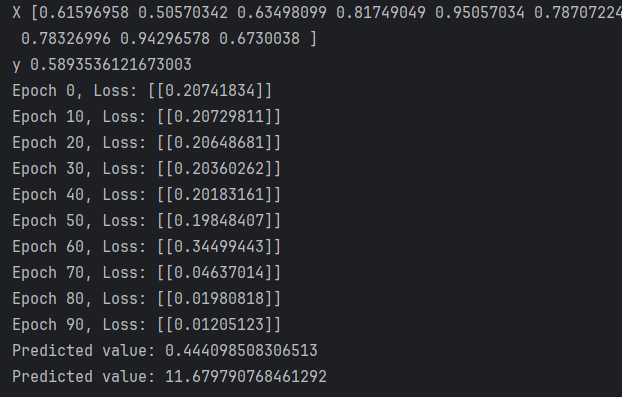


Рисунок 9.2 – Результат работы программы

10. ГРАФОВЫЕ НЕЙРОННЫЕ СЕТИ

10.1 Постановка задачи

Пользуясь лекционными материалами и методическими указаниями реализовать графовую нейронную сеть.

10.2 Описание графовую нейронную сеть

Графовая нейронная сеть — это тип нейронной сети, которая может обрабатывать данные, представленные в виде графа. Граф представляет собой набор узлов, связанных между собой ребрами. Узлы могут представлять объекты, а ребра – отношения между объектами.

**Структура.**

Графовая нейронная сеть обычно состоит из следующих слоев:

* Входной слой: Этот слой получает данные, представленные в виде графа.
* Слой распространения: Этот слой распространяет информацию по графу.
* Выходной слой: Этот слой производит выходной сигнал.

Слой распространения может быть реализован с использованием различных методов, таких как распространение с весами, распространение с использованием градиента и распространение с использованием внимания.

**Обучение.**

Обучение графовая нейронная сеть происходит следующим образом:

* На вход сети подается обучающий граф.
* Сеть производит выходной сигнал.
* Сравнивается выходной сигнал с целевым значением.
* Ошибка сети рассчитывается как разница между выходным сигналом и целевым значением.
* Веса сети обновляются таким образом, чтобы уменьшить ошибку сети.

Процесс обучения повторяется для всех обучающих графов.

**Принцип работы.**

При обучении GNN веса сети настраиваются таким образом, чтобы минимизировать ошибку сети. Ошибка сети рассчитывается как разница между выходным сигналом сети и целевым значением.

Веса сети обновляются с помощью метода градиентного спуска. Градиентный спуск — это метод оптимизации, который использует градиент ошибки для обновления параметров сети.

10.3 Результат работы

Будем обучать графовую нейронную сеть на сгенерированных данных (Рисунок 10.1).

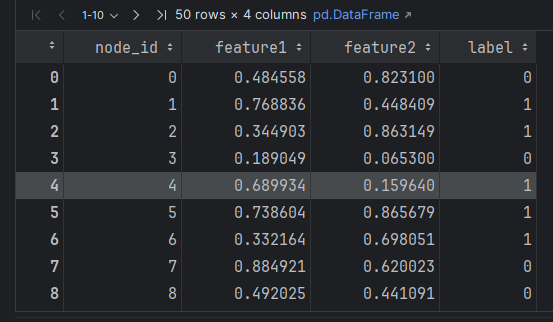


Рисунок 10.1 – Датасет

Будем обучать нейросеть 2000 эпох, шаг обучения = 0.1 (Рисунок 10.2).

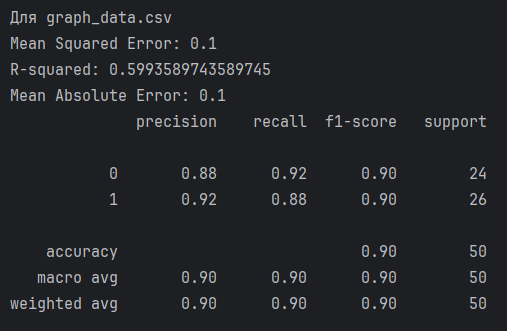


Рисунок 10.2 – Результат обучения нейросети

Вывод

В ходе исследования и практических занятий по проектированию и обучению нейронных сетей были рассмотрены и изучены различные методы и подходы к созданию глубоких моделей обучения. Каждая из представленных тем – от обучения по правилам Хебба до графовых нейронных сетей – демонстрирует уникальные принципы функционирования и применения в контексте разнообразных задач машинного обучения.

Важность разнообразия методов: Изучение различных методов обучения позволяет понять их преимущества и ограничения в различных сценариях. Каждый метод имеет свои особенности и может быть эффективным в определенных условиях.

Эволюция глубокого обучения: Разнообразие тем отражает постоянное развитие области глубокого обучения. От классических алгоритмов, таких как обратное распространение ошибки, до современных моделей, таких как трансформеры и графовые нейронные сети, наблюдается динамичное развитие технологий.

Необходимость практического применения: Эксперименты и практические работы важны для закрепления теоретических знаний. Реализация и тестирование моделей на практике позволяют понять их поведение в реальных условиях и их применимость к различным задачам.

Дальнейшие перспективы и исследования: Развитие глубокого обучения продолжает открывать новые горизонты и вызывать интерес к исследованиям в области нейронных сетей. Понимание основных методов является основой для дальнейших исследований и инноваций в этой области.

Итак, данное исследование позволило ознакомиться с разнообразием методов и подходов к проектированию и обучению нейронных сетей, подчеркивая их важность и актуальность в современной области машинного обучения и искусственного интеллекта.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гончаров, В.А. Методы оптимизации: Учебное пособие для ВУЗов / В.А.Гончаров. - Люберцы: Юрайт, 2016. - 191 c.

2. Горелик, В.А. Исследование операций и методы оптимизации: Учебник / В.А. Горелик. - М.: Academia, 2018. - 384 c.

3. Келлер, И.Э. Методы оптимизации в примерах и задачах: Учебное пособие / И.Э. Келлер. - СПб.: Лань, 2015. - 512 c.

4. Ширяев, В.И. Исследование операций и численные методы оптимизации: Учебное пособие / В.И. Ширяев. - М.: Ленанд, 2015. - 216 c.

5. Зайцев, М.Г. Методы оптимизации управления и принятия решений: примеры, задачи, кейсы / М.Г. Зайцев, С.Е. Варюхин. - М.: Дело АНХ, 2015. - 640 c.

6. Кочегурова, Е.А. Теория и методы оптимизации.: Учебное пособие для академического бакалавриата / Е.А. Кочегурова. - Люберцы: Юрайт, 2016. - 133 c.

ПРИЛОЖЕНИЯ

Приложение А – Листинг кода для обучения Хебба

Приложение Б – Листинг кода для дельта правила

Приложение В – Листинг кода обратного распространения ошибки

Приложение Г – Листинг кода радиально-базисных функций

Приложение Д – Листинг кода карты Кохонена

Приложение E – Листинг кода встречного распространения

Приложение Ж – Листинг кода рекуррентные сети

Приложение З – Листинг кода сверточные сети

Приложение И – Листинг кода трансформер

Приложение К - Листинг кода графовые нейронные сети

Приложение А

Листинг кода для обучения Хебба

Листинг А.1 – Функция main

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

last\_error = 0

step = 0.0001

mass\_value = [[-1, 1, -1, 1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1],

[-1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1]]

mass\_answers = [1, 1]

brain = Neural(mass\_value, random\_weight(), step, mass\_answers, last\_error)

brain.start([-1, 1, -1, 1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1])

Листинг А.2 – Описание класса нейронной сети Neural

class Neural:

def \_\_init\_\_(self, inputs, weight, step, answers, last\_error):

self.inputs = inputs

self.weight = weight

self.step = step

self.answers = answers

self.last\_error = last\_error

self.t = 0

Листинг А.3 – Реализация метода обучения нейронной сети

def train(self):

eph = 0

count = 0

while count != len(self.inputs):

self.summ\_to\_active = 0

for i in range(len(self.inputs[count])):

self.summ\_to\_active += self.inputs[count][i] \* self.weight[i]

self.summ\_to\_active -= self.t

if self.summ\_to\_active > 0:

self.summ\_to\_active = 1

else:

self.summ\_to\_active = -1

if self.summ\_to\_active != self.answers[count]:

for i in range(len(self.weight)):

self.weight[i] = round(self.weight[i] + self.inputs[count][i] \* self.answers[count],

5)

# print(self.weight[i], i)

self.t = self.t - self.answers[count]

count = 0

else:

count += 1

eph += 1

print(f'Эпоха №{eph} - ', self.weight)

Листинг А.4 – Реализация метода для распознавания заданной картинки

def start(self, new\_value):

self.train()

print('Веса -', self.weight)

final\_answer = 0

for i in range(len(new\_value)):

final\_answer += new\_value[i] \* self.weight[i]

final\_answer -= self.t

if final\_answer >= 0:

final\_answer = 1

print('Ответ:', final\_answer)

else:

print('Ответ:', -1)

Приложение Б

Листинг кода для дельта правила

Листинг Б.1 – Функция main

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

last\_error = 0

step = 0.01

epochs = 1

mass\_value = [[-1, -1, -1, -1],

[-1, -1, -1, 1],

[-1, -1, 1, -1],

[-1, -1, 1, 1],

[-1, 1, -1, -1],

[-1, 1, -1, 1],

[-1, 1, 1, -1],

[-1, 1, 1, 1],

[1, -1, -1, -1],

[1, -1, -1, 1],

[1, -1, 1, -1],

[1, -1, 1, 1],

[1, 1, -1, -1],

[1, 1, -1, 1],

[1, 1, 1, -1],

[1, 1, 1, 1]]

mass\_answers = [-1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, 1]

brain = Neural(mass\_value,

random\_weight(len(mass\_value[0])),

step,

mass\_answers,

last\_error,

epochs)

brain.start([0, 1, 1, 1])

Листинг Б.2 – Описание класса нейронной сети Neural

class Neural:

def \_\_init\_\_(self, inputs, weight, step, answers, last\_error):

self.inputs = inputs

self.weight = weight

self.step = step

self.answers = answers

self.last\_error = last\_error

self.t = 0

Листинг Б.3 – Реализация метода обучения нейронной сети

def train(self):

eph = 0

count = 0

while count != len(self.inputs):

self.summ\_to\_active = 0

for i in range(len(self.inputs[count])):

self.summ\_to\_active += self.inputs[count][i] \* self.weight[i]

self.summ\_to\_active -= self.t

if self.summ\_to\_active > 0:

self.summ\_to\_active = 1

else:

self.summ\_to\_active = -1

if self.summ\_to\_active != self.answers[count]:

for i in range(len(self.weight)):

self.weight[i] = round(self.weight[i] + self.inputs[count][i] \* self.answers[count],

Продолжение Листинг 3

5)

# print(self.weight[i], i)

self.t = self.t - self.answers[count]

count = 0

else:

count += 1

eph += 1

print(f'Эпоха №{eph} - ', self.weight)

Листинг 4 – Реализация метода для распознавания заданной картинки

def start(self, new\_value):

self.train()

print('Веса -', self.weight)

final\_answer = 0

for i in range(len(new\_value)):

final\_answer += new\_value[i] \* self.weight[i]

final\_answer -= self.t

if final\_answer >= 0:

final\_answer = 1

print('Ответ:', final\_answer)

else:

print('Ответ:', -1)

Приложение В

Листинг кода для обратного распространения ошибки

Листинг В.1 – Используемые библиотеки

import math

import random

Листинг В.2 – Функция main

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

input\_neurons\_weight = get\_start\_weight()

first\_hidden\_neurons\_weight = get\_first\_hidden\_weight()

second\_hidden\_neurons\_weight = get\_second\_hidden\_weight()

data, y = get\_data\_and\_y()

multi\_layered\_thing = Neural(data, y, input\_neurons\_weight, first\_hidden\_neurons\_weight,

second\_hidden\_neurons\_weight,

learning\_step=0.1, iteration=500)

multi\_layered\_thing.train()

new\_data = [1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1]

print('0')

multi\_layered\_thing.predict(new\_data)

print('1')

new\_data = [0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1]

multi\_layered\_thing.predict(new\_data)

print('1')

new\_data = [0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0]

multi\_layered\_thing.predict(new\_data)

print('1')

new\_data = [0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1]

multi\_layered\_thing.predict(new\_data)

print('0')

new\_data = [0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0]

multi\_layered\_thing.predict(new\_data)

Листинг В.3 – Функция генерации начальных весов

def get\_start\_weight(): # генерируем веса для 15 входных нейронов

input\_neurons\_weight = []

for j in range(15):

weight = []

for i in range(5):

weight.append(random.uniform(0, 0.5))

input\_neurons\_weight.append(weight)

print('Начльные веса входных нейронов')

for i in input\_neurons\_weight:

print(i)

print()

return input\_neurons\_weight

Листинг В.4 – Функция генерации данных

def get\_data\_and\_y():

data = [[1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1],

[0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0],

[1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1],

[0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0],

[1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1],

[0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0]]

y = [[0], [1], [0], [1], [0], [1]]

return data, y

Листинг В.5 – Функция сигмойды

def singmoid(input\_value):

return 1 / (1 + math.e \*\* (-input\_value))

Листинг В.6 – Функция производной сигмойды

def grad\_singmoid(input\_value):

return input\_value \* (1 - input\_value)

Листинг В.7 – Класс нейросети

class Neural:

def \_\_init\_\_(self, data, y, input\_neurons\_weight, first\_hidden\_neurons\_weight, second\_hidden\_neurons\_weight,

learning\_step, iteration):

self.data = data

self.y = y

self.input\_neurons\_weight = input\_neurons\_weight

self.first\_hidden\_neurons\_weight = first\_hidden\_neurons\_weight

self.second\_hidden\_neurons\_weight = second\_hidden\_neurons\_weight

self.learning\_step = learning\_step

self.iteration = iteration

Листинг В.8 –Мметод обчуния

def train(self):

epoch = 0

while epoch != self.iteration:

self.direct\_distribution()

if epoch % 10 == 0:

print()

print(f'Эпоха - {epoch}')

print(f'Входной слой веса {self.input\_neurons\_weight}')

print(f'Первый скрытый слой веса {self.first\_hidden\_neurons\_weight}')

print(f'Второй скрытый слой веса {self.first\_hidden\_neurons\_weight}')

print()

epoch += 1

Листинг В.9 – Обартное распространение ошибки

def direct\_distribution(self):

for d, dd in enumerate(self.data): # проход по данным

values\_on\_input\_second\_layer = [0] \* len(self.second\_hidden\_neurons\_weight)

values\_on\_out\_layer = 0

for i, ii in enumerate(self.first\_hidden\_neurons\_weight):

for j, jj in enumerate(self.first\_hidden\_neurons\_weight):

values\_on\_input\_second\_layer[j] += dd[j] \* self.first\_hidden\_neurons\_weight[j][i]

values\_on\_input\_second\_layer = [singmoid(i) for i in values\_on\_input\_second\_layer]

# print(f'Значения после второго слоя - {values\_on\_input\_second\_layer}')

for j, jj in enumerate(self.second\_hidden\_neurons\_weight):

values\_on\_out\_layer += values\_on\_input\_second\_layer[j] \* self.second\_hidden\_neurons\_weight[j]

values\_on\_out\_layer = singmoid(values\_on\_out\_layer)

# print(f'Значения на выходном слое слое - {values\_on\_out\_layer}')

# print()

# обратное распрастранение ошибки

# расчёт ошибки на всех уровнях

answer\_error = self.y[d][0] - values\_on\_out\_layer

# print('Ошибки сети', answer\_error)

second\_layer\_error = [0] \* len(values\_on\_input\_second\_layer)

# first\_layer\_error = [0] \* len(values\_on\_input\_first\_layer)

for j, jj in enumerate(self.second\_hidden\_neurons\_weight):

second\_layer\_error[j] += answer\_error \* self.second\_hidden\_neurons\_weight[j]

# print('Ошибки на втором скрытом слое', second\_layer\_error)

# обновляем веса

for i, ii in enumerate(self.input\_neurons\_weight[0]):

for j, jj in enumerate(self.input\_neurons\_weight):

self.input\_neurons\_weight[j][i] += \

second\_layer\_error[i] \* grad\_singmoid(values\_on\_input\_second\_layer[i]) \* \

data[d][i] \* self.learning\_step

# print(f'Новые веса для первого слоя')

# for i in input\_neurons\_weight:

# print(i)

# print()

for j, jj in enumerate(self.second\_hidden\_neurons\_weight):

self.second\_hidden\_neurons\_weight[j] += \

answer\_error \* grad\_singmoid(values\_on\_out\_layer) \* \

values\_on\_input\_second\_layer[j] \* self.learning\_step

Приложение Г

Листинг кода для радиально-базисных функций

Листинг Г.1 – Используемые библиотеки

import numpy as np

import random

import pandas as pd  
from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error

Листинг Г.2 – Датасет

data = pd.read\_csv('../Datasets/iris/iris.csv')  
y\_train = pd.DataFrame(data['Species'])

y\_train['Species'] = y\_train['Species'].replace('Iris-setosa', 2)

y\_train['Species'] = y\_train['Species'].replace('Iris-versicolor', 1)

y\_train['Species'] = y\_train['Species'].replace('Iris-virginica', 0)

# y\_train['Species'] = pd.factorize(data['Species'])[0]

y\_train = (y\_train - y\_train.min()) / (y\_train.max() - y\_train.min())  
X\_test = pd.read\_csv('../Datasets/iris/test\_iris.csv')

X\_test = X\_test.drop(['species', 'Unnamed: 0'], axis=1)  
y\_test = pd.read\_csv('../Datasets/iris/test\_iris.csv')

y\_test = y\_test.drop(['Unnamed: 0', 'petal length', 'petal width', 'sepal length', 'sepal width'], axis=1)

y\_test['species'] = pd.factorize(y\_test['species'])[0]

y\_test['species'] = (y\_test['species'] - y\_test['species'].min()) / (

y\_test['species'].max() - y\_test['species'].min()) # нормализация

Листинг Г.3 – Функция main

rbf\_network = RBFNetwork(k=12, learning\_rate=0.01, epochs=500)

rbf\_network.train(X=X\_train, y=y\_train)

y\_test['y\_pred'] = rbf\_network.predict(X=X\_test)

y\_test.loc[y\_test['y\_pred'] < 0.25, 'y\_pred'] = 0.0

y\_test.loc[(0.75 > y\_test['y\_pred']) & (y\_test['y\_pred'] > 0.25), 'y\_pred'] = 0.5

y\_test.loc[y\_test['y\_pred'] > 0.75, 'y\_pred'] = 1.0

y\_test['y\_pred'] = round(y\_test['y\_pred'], 1)

y\_test

#%%

from sklearn.metrics import mean\_squared\_error

y\_test = round(y\_test, 2)

y\_test['y\_pred'] = y\_test['y\_pred']

mean\_squared\_error(y\_test['species'], y\_test['y\_pred'])

#%%

from sklearn.metrics import mean\_squared\_error

y\_train['y\_pred'] = rbf\_network.predict(X=X\_train)

y\_train['y\_pred'] = round(y\_train['y\_pred'], 1)

mean\_squared\_error(y\_train['Species'], y\_train['y\_pred'])

Листинг Г.4 – Подсчёт точности и ошибок сети

y\_true = y\_test['species']

y\_pred = y\_test['y\_pred']

mse = mean\_squared\_error(y\_true, y\_pred)

r2 = r2\_score(y\_true, y\_pred)

mae = mean\_absolute\_error(y\_true, y\_pred)

print(f'Mean Squared Error: {mse}')

print(f'R-squared: {r2}')

print(f'Mean Absolute Error: {mae}')

Приложение Д

Листинг кода для карты Кохонена

Листинг Д.1 – Функция main

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

import random

error = 0.3

learning\_rate = 0.1

epochs = 10

X\_train, Y\_train = get\_dataset() # Получаем тренировочные данные

config = get\_architecture() # Получаем конфигурацию нейронной сети

weight = get\_and\_generate\_weight(config) # Получаем веса нейронной сети

# Создание экземпляра сети

brain = Neural\_network(X\_train, Y\_train, weight, learning\_rate, error, epochs, config)

# Обучаем нейросеть

brain.train\_km()

# Запуск нейрона с новыми входными данными

brain.pred\_km([[1, 1], [0, 0], [0, 1], [1, 0]])

Листинг Д.2 – Получение датасетов

def get\_dataset():

# Входные данные (матрица)

X\_train = [[0, 0],

[0, 1],

[1, 0],

[1, 1]]

# Ожидаемые ответы

Y\_train = []

return X\_train, Y\_train

Листинг Д.3 – Генерации весов

def get\_and\_generate\_weight(configuration):

weight = []

for layer in range(1, len(configuration)):

layer\_weight = []

for i in range(configuration[layer]):

array = []

for j in range(configuration[layer - 1]):

array.append(round(random.uniform(0, 1), 2))

# array.append(0)

layer\_weight.append(array)

weight.append(layer\_weight)

print(f'Изначальные веса')

line\_by\_line\_output(weight)

return weight

Листинг Д.4 – Класс нейросети

class Neural\_network:

def \_\_init\_\_(self, xtrain, answers, weight, learning\_rate, error, epochs, config):

self.xtrain = xtrain # Входные данные (матрица)

self.answers = answers # Ожидаемые ответы (массив)

self.weight = weight # Веса для каждого входа (инициализируются случайными значениями)

self.learning\_rate = learning\_rate # Шаг обучения (learning rate)

self.error = error # Последняя ошибка (не используется)

self.epochs = epochs # Количество эпох обучения (максимальное количество итераций)

self.t = 0 # Порог (threshold) для активации нейрона

self.config = config # Конфигурация нейронной сети

def weight\_update(self, d2, data):

neuron\_answers = [d2.index(max(d2)), max(d2)]

for index\_w, data\_w in enumerate(self.weight[0][neuron\_answers[0]]):

self.weight[0][neuron\_answers[0]][index\_w] += self.learning\_rate \* (data[index\_w] - data\_w)

self.weight[0][neuron\_answers[0]][index\_w] = round(self.weight[0][neuron\_answers[0]][index\_w], 2)

def train\_km(self):

for eph in range(self.epochs):

for index\_x, data\_x in enumerate(self.xtrain):

data\_out = []

for index\_w, data\_w in enumerate(self.weight[0]):

summ = 0

for \_ in range(len(data\_x)):

summ += (data\_w[\_] - data\_x[\_]) \*\* 2

data\_out.append(summ)

self.weight\_update(data\_out, data\_x)

def pred\_km(self, data):

print(f'Веса: {self.weight[0]}')

for index\_x, data\_x in enumerate(data):

data\_out = []

for index\_w, data\_w in enumerate(self.weight[0]):

summ = 0

for \_ in range(len(data\_x)):

summ += (data\_w[\_] - data\_x[\_]) \*\* 2

data\_out.append(summ)

neuron\_answers = [data\_out.index(max(data\_out)), max(data\_out)]

print(f'Данные: {data\_x} \nОтвет: {neuron\_answers}')

print()

Приложение Е

Листинг кода для встречного распространения

Листинг Е.1 – Функция main

import pandas as pd

import math

import numpy as np

data = pd.read\_csv('../Datasets/iris/iris.csv')

data = data.sample(frac=1, random\_state=12).reset\_index(drop=True)

X\_train = data.drop(['Species', 'Id'], axis=1)

y\_train = pd.DataFrame(data['Species'])

y\_train['Species'] = y\_train['Species'].replace('Iris-setosa', 2)

y\_train['Species'] = y\_train['Species'].replace('Iris-versicolor', 1)

y\_train['Species'] = y\_train['Species'].replace('Iris-virginica', 0)

X\_train = X\_train.to\_numpy()

y\_train\_new = y\_train.to\_numpy()

cp\_neurons = CounterPropagation(kohonen\_neurons=4,grossberg\_neurons=3,epoch=100,

learning\_rate\_a=0.1, learning\_rate\_b=0.1)

cp\_neurons.train(X=X\_train, y=y\_train)

X\_test\_pred = cp\_neurons.predict(X=X\_train)

X\_test\_pred = bringing\_the\_response\_back\_to\_normal(X\_test\_pred)

count\_error(y\_train, X\_test\_pred)

X\_test\_pred = cp\_neurons.predict(X=X\_test)

X\_test\_pred = bringing\_the\_response\_back\_to\_normal(X\_test\_pred)

count\_error(y\_test, X\_test\_pred)

Листинг Е.2 – Перевод данных в нужный вид

def bringing\_the\_response\_back\_to\_normal(input\_value\_answer):

for i, ii in enumerate(input\_value\_answer):

for j, jj in enumerate(input\_value\_answer[i]):

if jj != max(input\_value\_answer[i]):

input\_value\_answer[i][j] = 0

else:

input\_value\_answer[i][j] = 1

return input\_value\_answer

def convert\_answer(input\_answer):

y\_convert = []

for i in input\_answer:

if i == 0:

y\_convert.append([1, 0, 0])

if i == 1:

y\_convert.append([0, 1, 0])

if i == 2:

y\_convert.append([0, 0, 1])

return y\_convert

Листинг Е.3 – Метрики ошибок

from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error

def count\_error(answers\_data, predict\_data):

mse = mean\_squared\_error(answers\_data, predict\_data)

r2 = r2\_score(answers\_data, predict\_data)

mae = mean\_absolute\_error(answers\_data, predict\_data)

print(f'Mean Squared Error: {mse}')

print(f'R-squared: {r2}')

print(f'Mean Absolute Error: {mae}')

Листинг Е.4 – Класс нейросети

class CounterPropagation:

def \_\_init\_\_(self, kohonen\_neurons, grossberg\_neurons, epoch, learning\_rate\_a, learning\_rate\_b):

self.kohonen\_neurons = kohonen\_neurons

self.grossberg\_neurons = grossberg\_neurons

self.epoch = epoch

self.learning\_rate\_a = learning\_rate\_a

self.learning\_rate\_b = learning\_rate\_b

self.weight\_w = self.get\_weight\_w()

self.weight\_v = self.get\_weight\_v()

def get\_weight\_w(self):

mass\_weight = []

for i in range(self.kohonen\_neurons):

vector = []

for j in range(self.kohonen\_neurons):

# vector.append(random.uniform(0, 1))

vector.append(1 / (math.sqrt(self.kohonen\_neurons)))

mass\_weight.append(vector)

return mass\_weight

def get\_weight\_v(self):

mass\_weight = []

for i in range(self.kohonen\_neurons):

vector = []

for j in range(self.grossberg\_neurons):

vector.append(1 / (math.sqrt(self.grossberg\_neurons)))

mass\_weight.append(vector)

return mass\_weight

def calculation\_of\_values\_on\_kohonen\_layer(self, example):

result\_kohonen\_layer = []

for i, ii in enumerate(self.weight\_w):

neuron\_output = 0

for index\_value, value in enumerate(example):

neuron\_output += value \* self.weight\_w[i][index\_value]

result\_kohonen\_layer.append(neuron\_output)

for i, ii in enumerate(result\_kohonen\_layer):

if i != np.argmax(result\_kohonen\_layer):

result\_kohonen\_layer[i] = 0

return result\_kohonen\_layer, np.argmax(result\_kohonen\_layer)

def calculation\_of\_values\_on\_grossberg\_layer(self, kohonen\_layer, index\_winning\_neuron):

result\_layer\_grossberg = []

for i, ii in enumerate(self.weight\_v[index\_winning\_neuron]):

value = kohonen\_layer[index\_winning\_neuron] \* self.weight\_v[index\_winning\_neuron][i]

result\_layer\_grossberg.append(value)

return result\_layer\_grossberg

def weight\_adjustment\_kh(self, result\_kohonen\_layer, index\_winning\_neuron, example):

for i, ii in enumerate(self.weight\_w[index\_winning\_neuron]):

self.weight\_w[index\_winning\_neuron][i] += self.learning\_rate\_a \* (

example[i] - self.weight\_w[index\_winning\_neuron][i])

def weight\_adjustment\_gr(self, result\_kohonen\_layer, result\_layer\_grossberg, index\_winning\_neuron, y):

Продолжение - Листинга Е.4

for j, jj in enumerate(y):

self.weight\_v[index\_winning\_neuron][j] += self.learning\_rate\_b \* (

y[j] - result\_layer\_grossberg[j]) \* result\_kohonen\_layer[index\_winning\_neuron]

def train(self, X, y):

for eph in range(self.epoch):

for index\_training\_example, training\_example in enumerate(X):

result\_kohonen\_layer, index\_winning\_neuron = self.calculation\_of\_values\_on\_kohonen\_layer(

training\_example)

self.weight\_adjustment\_kh(result\_kohonen\_layer, index\_winning\_neuron, training\_example)

result\_layer\_grossberg = self.calculation\_of\_values\_on\_grossberg\_layer(result\_kohonen\_layer,

index\_winning\_neuron)

self.weight\_adjustment\_gr(result\_kohonen\_layer, result\_layer\_grossberg, index\_winning\_neuron,

y\_train[index\_training\_example])

self.learning\_rate\_a \*= 0.85

# self.learning\_rate\_b \*= 0.9

def predict(self, X):

predictions = []

for index\_training\_example, training\_example in enumerate(X):

result\_kohonen\_layer, index\_winning\_neuron = self.calculation\_of\_values\_on\_kohonen\_layer(training\_example)

result\_layer\_grossberg = self.calculation\_of\_values\_on\_grossberg\_layer(result\_kohonen\_layer, index\_winning\_neuron)

predictions.append(result\_layer\_grossberg)

return predictions

Приложение Ж

Листинг кода для рекуррентные сети

Листинг Ж.1 – Функция main

import numpy as np  
rnn = RNN(learning\_rate=0.01, epochs=50, hidden\_size=1)

bitcoin\_prices = np.array([100, 108, 110, 115, 105, 112, 120, 125, 135, 140])

rnn.train(bitcoin\_prices)

next\_price = rnn.predict([140, 145, 155, 160])

print(f'Input data: [140, 145, 155, 160]')

print(f'Predicted next Bitcoin price: {next\_price}')

Листинг Ж.2 – Класс нейросети

class RNN:

def \_\_init\_\_(self, learning\_rate, epochs, hidden\_size):

self.learning\_rate = learning\_rate

self.epochs = epochs

self.hidden\_size = hidden\_size

self.Wh\_h = np.random.randn(hidden\_size, hidden\_size)

self.Wh\_x = np.random.randn(hidden\_size, 1)

self.bh = np.zeros((hidden\_size, 1))

self.mean = None

self.std = None

def normalize(self, data):

self.mean = np.mean(data)

self.std = np.std(data)

return (data - self.mean) / self.std

def denormalize(self, data):

return data \* self.std + self.mean

def train(self, bitcoin\_prices):

bitcoin\_prices\_normalized = self.normalize(bitcoin\_prices)

for epoch in range(self.epochs):

loss = 0

h\_prev = 0

for i in range(len(bitcoin\_prices\_normalized) - 1):

# Прямой проход

h = np.tanh(np.dot(self.Wh\_h, h\_prev) + np.dot(self.Wh\_x, bitcoin\_prices\_normalized[i]) + self.bh)

# Предсказание

predicted\_price = np.dot(self.Wh\_x, bitcoin\_prices\_normalized[i + 1])

# Вычисление ошибки

loss += (predicted\_price - bitcoin\_prices\_normalized[i + 1]) \*\* 2

Продолжение – Листинг Ж.2

# Обратный проход

dWxh = np.dot((predicted\_price - bitcoin\_prices\_normalized[i + 1]), bitcoin\_prices\_normalized[i + 1])

dWhh = np.dot((predicted\_price - bitcoin\_prices\_normalized[i + 1]), h\_prev)

dbh = (predicted\_price - bitcoin\_prices\_normalized[i + 1])

# Обновление весов

self.Wh\_x -= self.learning\_rate \* dWxh

self.Wh\_h -= self.learning\_rate \* dWhh

self.bh -= self.learning\_rate \* dbh

h\_prev = h

# Вывод ошибки на каждой эпохе

print(f'Epoch {epoch + 1}, Loss: {loss}')

def predict(self, bitcoin\_prices):

bitcoin\_prices\_normalized = self.normalize(bitcoin\_prices)

h\_prev = 0

for i in range(len(bitcoin\_prices\_normalized) - 1):

h = np.tanh(np.dot(self.Wh\_h, h\_prev) + np.dot(self.Wh\_x, bitcoin\_prices\_normalized[i]) + self.bh)

h\_prev = h

next\_price\_normalized = np.tanh(np.dot(self.Wh\_h, h) + np.dot(self.Wh\_x, bitcoin\_prices\_normalized[-1]) + self.bh)

return self.denormalize(next\_price\_normalized)

Приложение З

Листинг кода для сверточные сети

Листинг З.1 – Функция main

X\_train, y\_train = get\_train\_data()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_train, y\_train, test\_size=0.1, random\_state=123)

simple\_conv\_net = CNN(

input\_size=X\_train[0].shape[0],

num\_classes=y\_train.shape[1], # Update the number of classes based on one-hot encoding shape

count\_convolutional\_layer=1)

simple\_conv\_net.train(X=X\_train, y=y\_train, epoch=1, learning\_rate=0.1)

def metrics\_printing(y\_true, y\_test, y\_pred, predictions):

print(f'Mean Squared Error: {mean\_squared\_error(y\_true, y\_pred)}')

print(f'R-squared: {r2\_score(y\_true, y\_pred)}')

print(f'Mean Absolute Error: {mean\_absolute\_error(y\_true, y\_pred)}')

print(classification\_report(y\_true, y\_pred))

true\_labels, predicted\_labels, predictions = print\_accuracy(simple\_conv\_net, X\_test, y\_test)

metrics\_printing(true\_labels,y\_test, predicted\_labels, predictions)

plt.rcParams['figure.figsize'] = (10, 10)

fig = px.imshow(confusion\_matrix(true\_labels, predicted\_labels), text\_auto=True)

fig.update\_layout(xaxis\_title='Цель', yaxis\_title='Прогноз')

Листинг З.2 – Используемые библиотеки

import numpy as np

import pandas as pd

import random

from sklearn.preprocessing import OneHotEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error, classification\_report, confusion\_matrix

import plotly.express as px

import matplotlib.pyplot as plt

Листинг З.3 – Получение данных

def get\_train\_data():

data\_train = pd.read\_csv('../Datasets/Fashion MNIST/fashion-mnist\_train.csv')

X\_train = data\_train.drop(['label'], axis=1).to\_numpy()

X\_train = X\_train.reshape((X\_train.shape[0], 28, 28, 1))

y\_train = data\_train['label'].to\_numpy()

encoder = OneHotEncoder(sparse=False)

y\_train\_onehot = encoder.fit\_transform(y\_train.reshape(-1, 1))

return X\_train, y\_train\_onehot

Листинг З.4 – Вывод точности

def print\_accuracy(model, X\_test, y\_test):

predictions = model.predict(X\_test)

predicted\_labels = np.argmax(predictions, axis=1)

true\_labels = np.argmax(y\_test, axis=1)

accuracy = np.mean(predicted\_labels == true\_labels)

print(f"Final Accuracy: {accuracy \* 100:.2f}%")

return true\_labels, predicted\_labels, predictions

Листинг З.5 – Класс нейросети

class CNN:

def \_\_init\_\_(self, input\_size, num\_classes, count\_convolutional\_layer):

self.input\_size = input\_size

self.num\_classes = num\_classes # Update the initialization

self.count\_convolutional\_layer = count\_convolutional\_layer

self.weight\_count\_convolutional = self.get\_weight\_count\_convolutional()

self.flatten\_weight = self.get\_flatten\_weight()

def get\_flatten\_weight(self):

flatten\_weight = []

for i in range(self.num\_classes):

mask = []

for j in range(49):

mask.append(random.uniform(0, 1))

flatten\_weight.append(mask)

return np.array(flatten\_weight)

def get\_weight\_count\_convolutional(self):

return np.random.uniform(0, 1, size=(3, 3))

def flatten(self, input\_data):

return input\_data.reshape(-1)

def convolve2d(self, input\_data):

channels = 1

if len(input\_data.shape) == 2:

height, width = input\_data.shape

input\_data = input\_data.reshape((height, width, 1))

elif len(input\_data.shape) == 3:

height, width, channels = input\_data.shape

else:

raise ValueError("Input data should be either 2D or 3D")

mask = np.zeros((height + 2, width + 2, channels))

mask[1:height + 1, 1:width + 1, :] = input\_data

output\_data = []

for i in range(1, height + 1):

s\_array = []

for j in range(1, width + 1):

summ = 0

for ii in range(-1, 2):

for jj in range(-1, 2):

summ += np.sum(mask[i + ii, j + jj, :] \* self.weight\_count\_convolutional[ii + 1][jj + 1])

s\_array.append(summ)

output\_data.append(s\_array)

return np.array(output\_data)

Продолжение - Листинга З.5

def max\_pooling(self, data):

leng = len(data)

i = 0

j = 0

result = []

while i < leng:

result\_str = []

while j < leng:

result\_str.append(max([data[i][j], data[i][j + 1], data[i + 1][j], data[i + 1][j + 1]]))

j += 2

i += 2

j = 0

result.append(result\_str)

return np.array(result)

def fully\_connected(self, flattened\_output):

return np.dot(self.flatten\_weight, flattened\_output)

def sigmoid(self, x):

return 1 / (1 + np.exp(-np.clip(x, -500, 500)))

def train(self, X, y, epoch, learning\_rate):

for eph in range(epoch):

for index\_training\_example, training\_example in enumerate(X):

training\_example = np.array(training\_example)

conv1\_output = self.convolve2d(training\_example)

pool1\_output = self.max\_pooling(conv1\_output)

conv2\_output = self.convolve2d(pool1\_output)

pool21\_output = self.max\_pooling(conv2\_output)

flattened\_output = self.flatten(pool21\_output)

fully\_connected = self.fully\_connected(flattened\_output)

output = self.sigmoid(fully\_connected)

loss\_gradient = output - y[index\_training\_example]

fc\_output\_gradient = np.dot(self.flatten\_weight.T, loss\_gradient)

fc\_output\_gradient[fc\_output\_gradient < 0] = 0

self.flatten\_weight -= learning\_rate \* np.outer(loss\_gradient, flattened\_output)

if index\_training\_example % 1000 == 0:

current\_loss = np.mean(

-y\_train \* np.log(output + 1e-10) - (1 - y\_train) \* np.log(1 - output + 1e-10))

print(f'Epoch {eph}, Example {index\_training\_example}, Loss: {current\_loss}')

def predict(self, X):

predictions = []

for example in X:

conv1\_output = self.convolve2d(example)

pool1\_output = self.max\_pooling(conv1\_output)

conv2\_output = self.convolve2d(pool1\_output)

pool2\_output = self.max\_pooling(conv2\_output)

Продолжение - Листинга З.5

flattened\_output = self.flatten(pool2\_output)

fully\_connected = self.fully\_connected(flattened\_output)

output = self.sigmoid(fully\_connected)

predictions.append(output)

return np.array(predictions)

Приложение И

Листинг кода для трансформера

Листинг И.1 – Функция main

data = pd.read\_csv('../Datasets/Daily-minimum-temperatures/daily-minimum-temperatures-in-me.csv')

data = data.drop('Date', axis=1).to\_numpy()

data = data.flatten()

new\_data = []

for i in data:

new\_data.append(float(i.replace('?', '')))

maximum = max(new\_data)

print('maximum', maximum)

# Нормализация данных

new\_data = np.array(new\_data) / maximum

# Разделение на подмассивы по 10 элементов

data = np.array\_split(new\_data, len(new\_data) / 10)

del data[-1]

print('data', data, '\n')

# Подготовка обучающих данных

data = np.array(data)

X = data[:, :-1] # Входные последовательности (все числа, кроме последнего)

y = data[:, -1] # Выходные числа (следующее число после входной последовательности)

print('data', data[1])

print('X', X[1])

print('y', y[1])

# Гиперпараметры модели

input\_size = 1 # Размерность входных данных

hidden\_size = 64 # Размер скрытого слоя

output\_size = 1 # Размер выходного слоя

learning\_rate = 0.001

num\_epochs = 100

# Инициализация весов

weights = {

'Wq': np.random.randn(input\_size, hidden\_size),

'Wk': np.random.randn(input\_size, hidden\_size),

'Wv': np.random.randn(input\_size, hidden\_size),

'Wo': np.random.randn(hidden\_size, output\_size),

'bq': np.zeros((1, hidden\_size)),

'bk': np.zeros((1, hidden\_size)),

'bv': np.zeros((1, hidden\_size)),

'bo': np.zeros((1, output\_size))

}

Продолжение - Листинга И.1

# Предсказание для новой последовательности чисел

# 10.7, 8.2, 6.1, 4.5, 6.1, 9.8, 9.7, 8.2, 8.4, 8.5

test\_sequence = np.array([10.7, 8.2, 6.1, 4.5, 6.1, 9.8, 9.7, 8.2, 8.4])

test\_sequence = test\_sequence / maximum

predicted\_value = None

# Прямое распространение по обученным весам

for i in range(len(test\_sequence)):

input\_seq = test\_sequence[i].reshape(-1, 1)

q = np.dot(input\_seq, weights['Wq']) + weights['bq']

k = np.dot(input\_seq, weights['Wk']) + weights['bk']

v = np.dot(input\_seq, weights['Wv']) + weights['bv']

scores = np.dot(q, k.T) / np.sqrt(hidden\_size)

attention\_weights = np.dot(scores, v)

attention\_output = np.sum(attention\_weights, axis=0, keepdims=True)

output = np.dot(attention\_output, weights['Wo']) + weights['bo']

predicted\_value = sigmoid(output)

predicted\_value = predicted\_value[0][0]

print(f'Predicted value: {predicted\_value}')

print(f'Predicted value: {predicted\_value \* maximum}')

Листинг И.2 – Используемые библиотеки

import numpy as np

import pandas as pd

Листинг И.3 – Обучение трансформера

# Обучение трансформера

for epoch in range(num\_epochs):

loss = 0

for i in range(len(X)):

# Прямое распространение (forward pass)

input\_seq = X[i].reshape(-1, 1)

target = y[i]

# Attention

q = np.dot(input\_seq, weights['Wq']) + weights['bq']

k = np.dot(input\_seq, weights['Wk']) + weights['bk']

v = np.dot(input\_seq, weights['Wv']) + weights['bv']

scores = np.dot(q, k.T) / np.sqrt(hidden\_size)

attention\_weights = np.dot(scores, v)

# Attention Layer

attention\_output = np.sum(attention\_weights, axis=0, keepdims=True)

# Output Layer

output = np.dot(attention\_output, weights['Wo']) + weights['bo']

prediction = sigmoid(output)

# Loss

loss += np.square(prediction - target)

# Обратное распространение (backpropagation) для обновления весов

d\_output = 2 \* (prediction - target) \* sigmoid\_derivative(prediction)

Продолжение – Листинг И.3

# Обратное распространение attention

d\_attention\_output = np.dot(d\_output, weights['Wo'].T)

d\_attention\_weights = d\_attention\_output \* np.ones\_like(attention\_weights) / len(attention\_weights)

d\_v = np.dot(scores.T, d\_attention\_weights)

d\_scores = np.dot(d\_attention\_weights, v.T)

d\_k = np.dot(d\_scores.T, q) # Используйте транспонированную матрицу d\_scores для умножения на q

d\_q = np.dot(d\_scores.T, k) # Используйте транспонированную матрицу d\_scores для умножения на k

d\_weights = {

'Wo': np.dot(attention\_output.T, d\_output),

'bo': np.sum(d\_output, axis=0, keepdims=True),

'Wv': np.dot(input\_seq.T, d\_v),

'bv': np.sum(d\_v, axis=0, keepdims=True),

'Wk': np.dot(input\_seq.T, d\_k),

'bk': np.sum(d\_k, axis=0, keepdims=True),

'Wq': np.dot(input\_seq.T, d\_q),

'bq': np.sum(d\_q, axis=0, keepdims=True)

}

# Обновление весов

for weight\_name in weights:

weights[weight\_name] -= learning\_rate \* d\_weights.get(weight\_name, 0)

if epoch % 10 == 0:

print(f'Epoch {epoch}, Loss: {loss / len(X)}')

Листинг И.4 – Функции активации и градиенты

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

return x \* (1 - x)

Приложение К

Листинг кода для графовой нейронной сети

Листинг К.1 – Функция main

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

filename = 'graph\_data.csv'

features, labels = get\_data\_from\_csv(filename=filename)

input\_dim = features.shape[1]

hidden\_dim = 15

output\_dim = 1

gnn = GraphNeuralNetwork(input\_dim, hidden\_dim, output\_dim)

gnn.train(features, labels, epochs=2000, learning\_rate=0.1)

y\_pred = gnn.predict(features)

metrics\_printing(labels, y\_pred, filename)

out\_confusion\_matrix(labels, y\_pred)

filename = 'graph\_data\_test.csv'

features, labels = get\_data\_from\_csv(filename=filename)

y\_pred = gnn.predict(features)

metrics\_printing(labels, y\_pred, filename)

out\_confusion\_matrix(labels, y\_pred)

Листинг К.2 – Используемые библиотеки

import numpy as np

import pandas as pd

from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error, classification\_report, confusion\_matrix

import plotly.express as px

Листинг К.3 – Получение данных

def get\_data\_from\_csv(filename):

data = pd.read\_csv(f'../Datasets/for\_gnn/{filename}')

features = data.drop(['label'], axis=1).values

features = (features - features.mean(axis=0)) / features.std(axis=0)

labels = data['label'].values

return features, labels

Листинг К.4 – Вывод точности

def out\_confusion\_matrix(y\_true, y\_pred):

fig = px.imshow(confusion\_matrix(labels, y\_pred), text\_auto=True)

fig.update\_layout(xaxis\_title='Цель', yaxis\_title='Прогноз')

fig.show()  
  
def metrics\_printing(y\_true, y\_pred, filename):

print(f'Для {filename}')

print(f'Mean Squared Error: {mean\_squared\_error(y\_true, y\_pred)}')

print(f'R-squared: {r2\_score(y\_true, y\_pred)}')

print(f'Mean Absolute Error: {mean\_absolute\_error(y\_true, y\_pred)}')

print(classification\_report(y\_true, y\_pred))

Листинг К.5 – Класс нейросети

class GraphNeuralNetwork:

def \_\_init\_\_(self, input\_dim, hidden\_dim, output\_dim):

self.input\_dim = input\_dim

self.hidden\_dim = hidden\_dim

self.output\_dim = output\_dim

self.weights\_hidden = np.random.rand(self.input\_dim, self.hidden\_dim)

self.weights\_output = np.random.rand(self.hidden\_dim, self.output\_dim)

self.bias\_hidden = np.random.rand(self.hidden\_dim)

self.bias\_output = np.random.rand(self.output\_dim)

def sigmoid(self, x):

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(self, x):

return x \* (1 - x)

def forward\_pass(self, inputs):

hidden = self.sigmoid(np.dot(inputs, self.weights\_hidden) + self.bias\_hidden)

output = self.sigmoid(np.dot(hidden, self.weights\_output) + self.bias\_output)

return output

def train(self, inputs, labels, epochs, learning\_rate):

for epoch in range(epochs):

for i in range(len(inputs)):

hidden = self.sigmoid(np.dot(inputs[i], self.weights\_hidden) + self.bias\_hidden)

output = self.sigmoid(np.dot(hidden, self.weights\_output) + self.bias\_output)

output\_error = labels[i] - output

output\_delta = output\_error \* self.sigmoid\_derivative(output)

hidden\_error = np.dot(output\_delta, self.weights\_output.T)

hidden\_delta = hidden\_error \* self.sigmoid\_derivative(hidden)

self.weights\_output += learning\_rate \* np.dot(hidden.reshape(-1, 1), output\_delta.reshape(1, -1))

self.bias\_output += learning\_rate \* output\_delta

self.weights\_hidden += learning\_rate \* np.dot(inputs[i].reshape(-1, 1), hidden\_delta.reshape(1, -1))

self.bias\_hidden += learning\_rate \* hidden\_delta

def predict(self, inputs):

predictions = []

for i in range(len(inputs)):

prediction = self.forward\_pass(inputs[i])

predictions.append(prediction)

predictions = [int(item.flatten()[0] > 0.5) for item in predictions]

return predictions