СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc168021072)

[1. ПОСТАНОВКА ЗАДАЧИ 6](#_Toc168021073)

[2. РАЗРАБОТКА АЛГОРИТМОВ 8](#_Toc168021074)

[2.1 Алгоритм матричного произведения 9](#_Toc168021075)

[2.2. Алгоритм инициализации нейронной сети 10](#_Toc168021076)

[2.3. Алгоритм прямого распространения в нейронной сети 11](#_Toc168021077)

[2.4. Алгоритм обратного распространения ошибки 12](#_Toc168021078)

[2.5. Алгоритм обновления весов 13](#_Toc168021079)

[2.6. Алгоритм матричного произведения 13](#_Toc168021080)

[3. ВЕДЕНИЕ В НЕЙРОННЫЕ СЕТИ 15](#_Toc168021081)

[4. РАЗРАБОТКА ПРОГРАММЫ 26](#_Toc168021082)

[4.1. Спецификация программы 26](#_Toc168021083)

[4.2. Описание класса Matrix 27](#_Toc168021084)

[4.3. Описание класса ActivateFunction 29](#_Toc168021085)

[4.4. Описание класса NetWork 31](#_Toc168021086)

[4.5. Описание функции main 33](#_Toc168021087)

[5. ТЕСТИРОВАНИЕ 36](#_Toc168021088)

[5.1. Описание входных и выходных данных 36](#_Toc168021089)

[5.2. Результаты тестирования 37](#_Toc168021090)

[ЗАКЛЮЧЕНИЕ 41](#_Toc168021091)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 43](#_Toc168021092)

**ПРИЛОЖЕНИЕ А Текст программы**

**ПРИЛОЖЕНИЕ Б Графический материал**

# ВВЕДЕНИЕ

В современном мире компьютерные технологии активно внедряются в различные сферы жизни, что требует постоянного совершенствования алгоритмов машинного обучения и разработки эффективных методов обработки данных. Одной из ключевых задач в этой области является распознавание рукописных символов, что находит применение в различных сферах, от почтовой сортировки до обработки банковских документов.

Целью данной курсовой работы является разработка и реализация нейронной сети для распознавания рукописных цифр на языке программирования C++. Для достижения данной цели необходимо решить следующие задачи:

1. Изучение основных принципов работы нейронных сетей и методов машинного обучения, используемых для распознавания образов;

2. Анализ существующих алгоритмов распознавания рукописных символов и выбор наиболее подходящего подхода для реализации;

3. Разработка архитектуры нейронной сети, включая выбор типов слоев, количество нейронов и функции активации;

4. Написание программного кода на языке C++ для обучения и тестирования нейронной сети;

5. Проведение экспериментов для оценки качества работы разработанной нейронной сети и сравнение ее производительности с другими методами распознавания рукописных цифр;

Реализация данного проекта позволит создать эффективное средство для автоматизации процесса распознавания рукописных цифр, что имеет большое значение в сфере компьютерного зрения и обработки информации. Возможности для работы с консольными приложениями, что особенно важно для данного проекта.

Вкратце о процессе создания нейронной сети. Для её создания цифр на языке программирования C++ необходимо пройти ряд этапов.

Первым этапом является сбор и подготовка данных для обучения нейронной сети. Это включает в себя получение набора изображений рукописных цифр, их предобработку (например, нормализацию размеров и улучшение контраста) и разделение на обучающую и тестовую выборки.

Далее следует выбор подходящей архитектуры нейронной сети. Это может быть, н0.566апример, свёрточная нейронная сеть (CNN), которая показывает высокую эффективность в обработке изображений. Необходимо определить количество слоев, их типы и конфигурацию, а также выбрать функции активации и метод оптимизации.

После определения архитектуры следует написание программного кода для реализации нейронной сети на языке C++. Это включает в себя создание классов и функций для работы с нейронами, слоями и обучением сети. Важно также реализовать алгоритмы обратного распространения ошибки и обучения с использованием выбранного метода оптимизации, такого как градиентный спуск или его вариации.

После завершения реализации необходимо провести обучение нейронной сети на обучающей выборке и оценить ее производительность на тестовой выборке. Этот этап включает в себя тщательное настройку гиперпараметров сети, таких как скорость обучения и количество эпох, для достижения оптимальных результатов.

Наконец, после успешного обучения нейронной сети проводится ее тестирование на новых данных для оценки ее способности к распознаванию рукописных цифр с высокой точностью и надежностью.

# ПОСТАНОВКА ЗАДАЧИ

При разработке курсовой работы по созданию нейронной сети для распознавания рукописных цифр на языке C++ необходимо сформулировать следующую подробную постановку задачи:

1. Сбор данных:

- провести поиск и сбор набора данных рукописных цифр, содержащего изображения цифр от 0 до 9;

-подготовить данные, включая их нормализацию и разделение на обучающую и тестовую выборки;

2. Изучение методов машинного обучения:

- провести обзор и изучение основных методов машинного обучения, используемых для задачи распознавания образов;

- изучить принципы работы нейронных сетей и их применение в задачах компьютерного зрения;

3. Анализ существующих алгоритмов:

- проанализировать существующие алгоритмы распознавания рукописных символов, включая сверточные нейронные сети (CNN), многослойные перцептроны и другие;

- выбрать наиболее подходящий подход для реализации в данной курсовой работе.

4. Разработка архитектуры нейронной сети:

- разработать архитектуру нейронной сети, определить количество слоев, их типы и конфигурацию;

- выбрать подходящие функции активации и методы оптимизации.

5. Написание программного кода:

- написать программный код на языке C++ для реализации нейронной сети согласно выбранной архитектуре.

- реализовать алгоритмы обратного распространения ошибки, функции активации, методы оптимизации и другие необходимые компоненты.

6. Обучение и тестирование нейронной сети:

- обучить разработанную нейронную сеть на обучающей выборке с использованием выбранных методов обучения.

- провести тестирование нейронной сети на тестовой выборке для оценки ее производительности и точности распознавания рукописных цифр.

7. Оценка результатов:

- оценить качество работы разработанной нейронной сети с помощью метрик, таких как точность, полнота и F-мера.

-сравнить результаты с другими существующими методами распознавания рукописных цифр для оценки эффективности разработанной нейронной сети.

8. Документация и отчет:

- написать документацию к программному коду с описанием его структуры и принципов работы.

- подготовить курсовой отчет, включающий в себя введение, постановку задачи, описание методов, результаты экспериментов и выводы.

Реализация этих задач позволит создать эффективную нейронную сеть для распознавания рукописных цифр на языке C++, а также провести анализ ее производительности и точности.

# 2. РАЗРАБОТКА АЛГОРИТМОВ

Разработка алгоритмов для курсовой работы по созданию нейронной сети для распознавания рукописных цифр на языке C++ включает в себя следующие этапы:

1. Выбор архитектуры нейронной сети:

- рассмотрение различных архитектур нейронных сетей, таких как свёрточные нейронные сети (CNN), многослойные перцептроны (MLP) и комбинации этих архитектур;

- принятие решения о выборе наиболее подходящей архитектуры с учетом специфики задачи распознавания рукописных цифр.

2. Определение структуры сети:

- определение количества слоев и их типов (например, свёрточные, пулинговые, полносвязанные).

- определение размерности и активации каждого слоя.

- решение о наличии слоев нормализации и регуляризации.

3. Инициализация весов:

- выбор метода инициализации начальных весов нейронов, например, случайной инициализации или методов Xavier/Glorot и He.

- реализация соответствующего кода для инициализации весов в каждом слое сети.

4. Реализация прямого прохода (forward pass):

- написание кода для передачи входных данных через сеть.

- реализация вычислений в каждом слое, включая операции свертки (для CNN), активации и пулинга.

- передача полученного выхода на последний слой для получения результата.

5. Реализация обратного прохода (backpropagation):

- написание кода для вычисления градиентов функции потерь по параметрам сети.

- реализация обратного распространения ошибки от выходного слоя к входным слоям.

- обновление весов сети в соответствии с вычисленными градиентами, используя выбранный метод оптимизации (например, стохастический градиентный спуск).

6. Выбор функции потерь и метрик оценки:

- выбор подходящей функции потерь, например, кросс-энтропии или среднеквадратичной ошибки.

- определение метрик оценки качества модели, таких как точность, точность по классам, полнота.

7. Реализация обучения:

- написание кода для цикла обучения сети, включая передачу обучающих данных через сеть, вычисление функции потерь и обновление весов.

- реализация контроля процесса обучения, такого как вывод информации о прогрессе и остановка обучения по достижении определенного критерия.

8. Тестирование и оценка:

- написание кода для тестирования обученной сети на тестовой выборке.

- оценка качества работы сети с использованием ранее определенных метрик.

- анализ результатов и сравнение с другими методами распознавания рукописных цифр.

## **2.1 Алгоритм матричного произведения**

1. Проверка входных данных:

- выполняется проверка, что количество столбцов в матрице m1 (m1.col) равно размеру вектора neuron (n). Если размеры не совпадают, выбрасывается исключение std::runtime\_error с сообщением "Error Multi!\n".

1. Основной цикл умножения:

- внешний цикл for (int i = 0; i < m1.row; i++) проходит по строкам матрицы m1;

-внутри этого цикла инициализируется временная переменная tmp для накопления суммы произведений элементов строки матрицы на соответствующие элементы вектора.

1. Вычисление суммы произведений:

- внутренний цикл for (int j = 0; j < m1.col; j++) проходит по столбцам текущей строки матрицы m1;

-для каждого элемента строки матрицы выполняется умножение m1.matrix[i][j] на соответствующий элемент вектора neuron[j], и результат добавляется к tmp.

1. Запись результата:

-после завершения внутреннего цикла, накопленное значение tmp записывается в выходной массив c в позицию c[i].

(см. приложение Б, рисунок 2.1)

## **2.2. Алгоритм инициализации нейронной сети**

1. Настройка функции активации:

-вызывается метод actFunc.set(), который настраивает функцию активации.

1. Инициализация генератора случайных чисел:

-функция srand(time(NULL)) инициализирует генератор случайных чисел для последующей генерации случайных весов.

1. Установка количества слоев:

-переменной L присваивается значение data.L, указывающее на количество слоев в сети.

1. Создание массива размеров слоев:

-выделяется память для массива size, который будет содержать количество нейронов в каждом слое;

-заполнение массива size значениями из data.size.

1. Инициализация матриц весов и смещений:

-выделяется память для массива weights длиной L-1, так как количество матриц весов на одну меньше количества слоев;

-выделяется память для массива указателей на массивы смещений bios;

- вцикле от 0 до L-2: для каждой матрицы весов вызывается метод Init, который инициализирует матрицу размером size[i+1] на size[i]; для каждой строки массива bios выделяется память; вызывается метод Rand для случайной инициализации весов; для каждого элемента массива смещений bios генерируется случайное значение в заданном диапазоне.

1. Инициализация значений нейронов и ошибок:

-выделяется память для массивов указателей neurons\_val и neurons\_err, длиной L;

- цикле от 0 до L-1: для каждого слоя выделяется память под массив значений нейронов и массив ошибок.

1. Инициализация значений смещений нейронов:

- выделяется память для массива neurons\_bios\_val, длиной L-1.

- цикле от 0 до L-2: каждому элементу массива neurons\_bios\_val присваивается значение 1.

(см. приложение Б, рисунок 2.2)

## **2.3. Алгоритм прямого распространения в нейронной сети**

1. Прямое распространение через каждый слой:

-цикл for (int k = 1; k < L; k++) итерируется по каждому слою сети, начиная со второго слоя (первый скрытый слой) и до последнего слоя (выходного слоя).

1. Матричное умножение:

-для каждого слоя k выполняется операция матричного умножения между матрицей весов weights[k - 1] и вектором значений нейронов предыдущего слоя neurons\_val[k - 1]. Результат умножения сохраняется в векторе значений нейронов текущего слоя neurons\_val[k];

-вызов функции Matrix::Multi(weights[k - 1], neurons\_val[k - 1], size[k - 1], neurons\_val[k]) реализует это матричное умножение.

1. Сложение со смещением:

- каждому элементу вектора значений нейронов текущего слоя neurons\_val[k] прибавляется соответствующий элемент вектора смещений bios[k - 1];

- вызов функции Matrix::SumVector(neurons\_val[k], bios[k - 1], size[k]) реализует это сложение.

1. Применение функции активации:

-к каждому элементу вектора значений нейронов текущего слоя neurons\_val[k] применяется функция активации;

-вызов функции actFunc.use(neurons\_val[k], size[k]) выполняет это преобразование. Функция активации преобразует линейную комбинацию входов в нелинейное значение, что позволяет сети моделировать сложные нелинейные зависимости.

1. Определение предсказания:

-после прохождения данных через все слои сети (от второго слоя до выходного слоя), сеть формирует предсказание. Предсказание определяется как индекс максимального значения вектора значений нейронов выходного слоя;

-вызов функции SearchMaxIndex(neurons\_val[L - 1]) возвращает индекс наибольшего элемента в векторе neurons\_val[L - 1], который соответствует классу с наибольшей вероятностью.

1. Возврат предсказания:

-функция возвращает индекс класса, который был определен на выходном слое как предсказание.

(см. приложение Б, рисунок 2.3)

## **2.4. Алгоритм обратного распространения ошибки**

1. Вычисление ошибок выходного слоя:

- для каждого нейрона в выходном слое рассчитывается ошибка. Ошибка вычисляется на основе разницы между предсказанным значением и ожидаемым значением;

- если индекс текущего нейрона не совпадает с индексом ожидаемого класса (expect), то ошибка нейрона рассчитывается как отрицательное значение предсказанного выхода, умноженное на производную функции активации: -neurons\_val[L - 1][i] \* actFunc.useDer(neurons\_val[L - 1][i]);

- если индекс текущего нейрона совпадает с индексом ожидаемого класса, то ошибка рассчитывается как разница между 1 и предсказанным выходом, умноженная на производную функции активации: (1.0 - neurons\_val[L - 1][i]) \* actFunc.useDer(neurons\_val[L - 1][i]).

1. Вычисление ошибок скрытых слоев:

- цикл for (int k = L - 2; k > 0; k--) итерируется по скрытым слоям сети, начиная с последнего скрытого слоя и заканчивая первым скрытым слоем;

- для каждого слоя k выполняется матричное умножение транспонированной матрицы весов текущего слоя weights[k] и вектора ошибок следующего слоя neurons\_err[k + 1]. Результат умножения сохраняется в векторе ошибок текущего слоя neurons\_err[k];

- вызов функции Matrix::Multi\_T(weights[k], neurons\_err[k + 1], size[k + 1], neurons\_err[k]) реализует это матричное умножение;

- для каждого нейрона текущего слоя k ошибка умножается на производную функции активации для соответствующего значения нейрона: neurons\_err[k][j] \*= actFunc.useDer(neurons\_val[k][j]).

Матричное умножение транспонированной матрицы (Matrix::Multi\_T): функция принимает на вход транспонированную матрицу весов текущего слоя, вектор ошибок следующего слоя, размер следующего слоя и записывает результат умножения в вектор ошибок текущего слоя.

Производная функции активации (actFunc.useDer): используется для вычисления градиента ошибки для каждого нейрона, что необходимо для корректного обновления весов.

Обратное распространение ошибки: этот процесс позволяет обновлять веса сети таким образом, чтобы минимизировать ошибку предсказания. Обратное распространение ошибки выполняется путем перемещения от выходного слоя к первому скрытому слою, вычисляя градиенты ошибок для каждого слоя.

(см. приложение Б, рисунок 2.4)

## **2.5. Алгоритм обновления весов**

1. Обновление весов:

- внешний цикл по слоям: for (int i = 0; i < L - 1; ++i) итерируется по всем слоям сети, кроме последнего выходного слоя, так как веса присутствуют только между слоями;

- средний цикл по нейронам текущего слоя: for (int j = 0; j < size[i + 1]; ++j) итерируется по всем нейронам слоя, следующего за текущим;

-внутренний цикл по нейронам предыдущего слоя: for (int k = 0; k < size[i]; ++k) итерируется по всем нейронам текущего слоя;

- обновление весов: для каждого веса между нейронами слоев i и i+1 выполняется обновление по правилу градиентного спуска: weights[i](j, k) += neurons\_val[i][k] \* neurons\_err[i + 1][j] \* lr;

- neurons\_val[i][k] — значение активации нейрона k в слое i;

- neurons\_err[i + 1][j] — ошибка нейрона j в слое i+1;

- lr — скорость обучения (learning rate), определяющая шаг обновления весов.

1. Обновление биосов:

- внешний цикл по слоям: for (int i = 0; i < L - 1; i) итерируется по всем слоям сети, кроме последнего выходного слоя;

- внутренний цикл по нейронам текущего слоя: for (int k = 0; k < size[i + 1]; k) итерируется по всем нейронам слоя i+1;

- обновление биосов: для каждого биоса нейрона в слое i+1 выполняется обновление по правилу градиентного спуска: bios[i][k] += neurons\_err[i + 1][k] \* lr;

- neurons\_err[i + 1][k] — ошибка нейрона k в слое i+1;

- lr — скорость обучения (learning rate).

(см. приложение Б, рисунок 2.5)

## **2.6. Алгоритм матричного произведения**

1. Выбор функции активации:

- функция использует оператор switch для определения, какая функция активации используется в текущий момент (задается через переменную actFunc).

1. Вычисление производной для функции сигмоида (sigmoid):

- производная сигмоида: f'(x) = f(x) \* (1 - f(x));

- для каждого элемента массива value выполняется вычисление value[i] = value[i] \* (1 - value[i]);

- цикл по элементам массива: for (int i = 0; i < n; i).

1. Вычисление производной для функции ReLU (Rectified Linear Unit):

- производная ReLU: f'(x) = 0.01, если x < 0 или x > 1 или

f'(x) = 1, если 0 <= x <= 1;

- для каждого элемента массива value выполняется проверка условия: если value[i] < 0 или value[i] > 1, устанавливается value[i] = 0.01, противном случае, устанавливается value[i] = 1.

1. Вычисление производной для гиперболического тангенса (tanh):

- производная tanh: ;

- для каждого элемента массива value выполняется вычисление: если value[i] < 0, value[i] = 0.01 \* (1 - value[i] \* value[i]) или, если value[i] >= 0, value[i] = 1 - value[i] \* value[i].

1. Обработка ошибок:

-если значение actFunc не соответствует ни одному из предопределенных значений (sigmoid, ReLU, tanh), выбрасывается исключение с сообщением об ошибке: throw std::runtime\_error("Error activate derivative of function\n").

(см. приложение Б, рисунок 2.5)

# вВЕДЕНИЕ В НЕЙРОННЫЕ СЕТИ

Компьютеры, в своей сути, являются машинами для выполнения арифметических операций с высокой скоростью. Они успешно выполняют широкий спектр задач, требующих численных расчетов, таких как определение цены с учетом скидки, расчет долговых процентов и генерация графиков на основе данных.

Даже простые действия, такие как просмотр телевизионных программ или прослушивание музыки с использованием компьютера, фактически сводятся к выполнению множества арифметических операций. Например, отрисовка каждого кадра изображения, полученного из интернета в виде последовательности битов, требует вычислительных операций, сравнимых по сложности с простыми арифметическими задачами.

Однако, не следует путать способность компьютеров к выполнению миллионов операций в секунду с искусственным интеллектом. Работа с числами на таком уровне не требует высоких когнитивных способностей, а скорее подчиняется заранее определенным алгоритмам.

С компьютерами все ясно. Однако, важно понимать, что, в отличие от них, человек обладает способностью к анализу и интерпретации сложных ситуаций, а также к творческому мышлению.

Посмотрите на рисунок 3.1 ниже и определите, что на них изображено:



Рисунок 3.1 – Люди, кошачья морда, дерево

При рассмотрении изображений, определение объектов на них человеком происходит быстро и с высокой точностью. Например, на первой картинке видны лица людей, на второй - морда кошки, на третьей - дерево. Этот процесс происходит моментально и требует минимальных умственных усилий. Человек редко допускает ошибки в таких задачах распознавания объектов на изображениях.

Несмотря на легкость выполнения этой задачи для человека, она представляет собой сложную задачу для компьютера. Компьютерам значительно сложнее автоматически распознавать объекты на изображениях с такой же точностью и скоростью, как у человека.

У компьютеров, несмотря на их скорость и надежность, отсутствует ключевое качество - интеллект, характерный для человека.

Целью исследований в области искусственного интеллекта является разработка алгоритмов, позволяющих компьютерам решать сложные задачи, подобные тем, которые успешно выполняются людьми. Это включает в себя автоматическое распознавание объектов на изображениях, анализ текста, принятие решений и другие когнитивные функции.

Хотя компьютеры по-прежнему базируются на микросхемах и электронных компонентах, задача искусственного интеллекта заключается в создании новых алгоритмов и методов, которые позволят им эмулировать поведение человека при решении сложных задач. В результате применения этих алгоритмов компьютеры могут проявлять поведение, которое ассоциируется с человеческим интеллектом.

Рассмотрим простую модель предсказательной машины, способной преобразовывать данные на входе в результаты на выходе. Компьютер, не проявляя "мыслительной" активности, выполняет заранее определенные арифметические операции, обрабатывая полученные данные и выдавая окончательный результат. Для иллюстрации данного процесса рассмотрим пример перевода километров в мили.

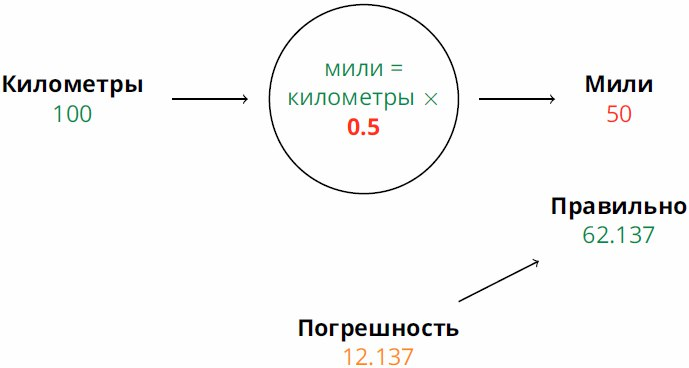
Предположим, что зависимость между километрами и милями линейна, то есть выражается формулой миль = километры × C, где C представляет собой постоянную, значение которой нам неизвестно.

На данный момент известно лишь несколько заранее измеренных расстояний в километрах и милях (см. таблицу 3.1).



Таблица 3.1 – Результаты измерений

Путем случайного выбора значения C = 0,5 и последующего преобразования расстояния в километрах в мили, мы получаем некоторую величину, которая оказывается ниже правильного ответа, что приводит к возникновению погрешности (см. рисунок 3.2).

 Рисунок 3.2. – Первый подбор параметра С

Зная величину погрешности, мы корректируем значение C, сделав его ближе к правильному ответу. Этот процесс продолжается до тех пор, пока не будет достигнута приемлемая точность результата (см. рисунок 3.3).

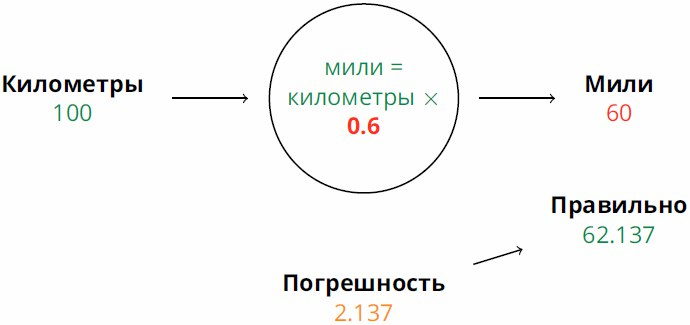


Рисунок 3.3 – Второй подбор параметра С

Подход, используемый в данном примере, демонстрирует принцип работы нейронных сетей, которые обучаются постепенно улучшать точность результатов.

Важно отметить, что в данном примере корректировка значения C зависит от величины погрешности: чем больше погрешность, тем сильнее корректировка. Однако по мере приближения к правильному ответу изменения параметров становятся менее значительными.

Теперь рассмотрим вопрос о том, что такое нейрон. Нейроны — природные вычислительные машины Исследования демонстрируют, что даже птицы, такие как голуби, обладают интеллектуальными способностями, превосходящими возможности цифровых компьютеров. Эти компьютеры состоят из множества вычислительных элементов, способны хранить значительные объемы данных и производить вычисления с большей скоростью и точностью, чем биологические мозги.

Традиционные компьютеры выполняют конкретные и четкие команды, исключая неопределенность и двойственность. В отличие от них, мозги животных функционируют иначе: они работают медленнее, но обрабатывают несколько сигналов одновременно, что часто приводит к неопределенностям.

Основным вычислительным элементом любого биологического мозга является нейрон. Существует множество типов нейронов, однако все они передают электрический сигнал от дендритов к терминалам аксона. Этот сигнал передается от одного нейрона к другому, обеспечивая восприятие света, звуков, прикосновений, тепла и других стимулов. Сигналы принимаются сенсорными нейронами и передаются через нервную систему в мозг, который в значительной степени состоит из нейронов. На рисунке 3.4 представлено общее строение нейрона.

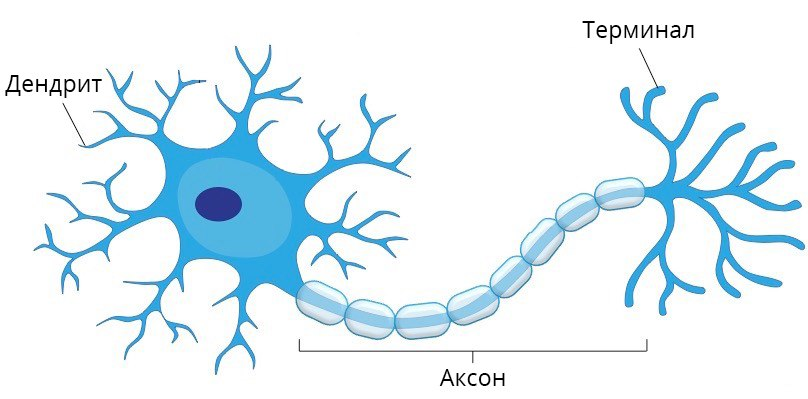


Рисунок 3.4 – Строение биологического нейрона

На иллюстрации ниже (рисунок 3.5) представлены различные типы нейронов мозга голубя, выполненные испанским нейробиологом в 1889 году. Основные части нейрона, такие как дендриты и терминалы аксона, четко видны.

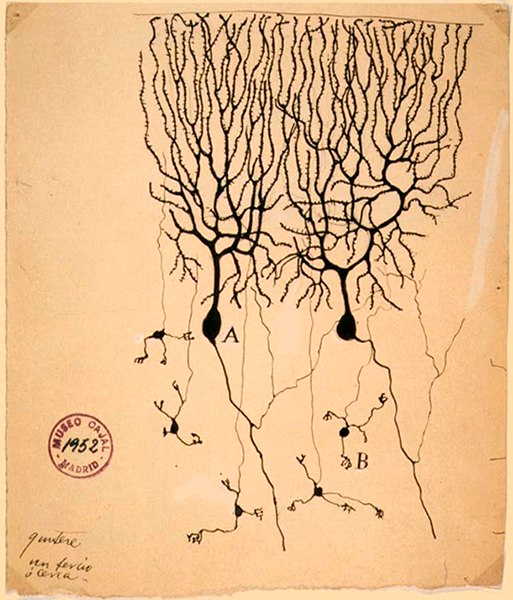


Рисунок 3.5 – Нейроны мозга голубя

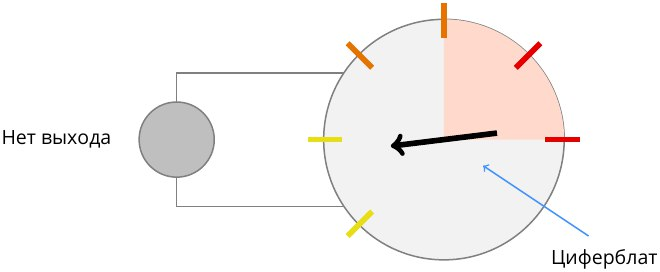
Сколько нейронов необходимо для выполнения сложных задач? В среднем человеческий мозг содержит около 100 триллионов нейронов. У плодовых мушек примерно 100 000 нейронов, что позволяет им летать, питаться, избегать опасности, находить пищу и выполнять множество других сложных задач. Современные компьютеры способны симулировать работу 100 000 нейронов. У круглых червей всего 302 нейрона, что является незначительным количеством по сравнению с компьютерами. Однако, несмотря на это, черви могут выполнять действия, с которыми не справятся традиционные компьютерные программы.

Почему биологические мозги демонстрируют такую высокую интеллектуальную производительность, несмотря на их более медленную работу и зачастую меньшее количество вычислительных элементов по сравнению с компьютерами? На данный момент мы не имеем полного понимания того, как функционирует мозг в целом. Природа сознания остается для нас загадкой. Однако мы достаточно хорошо изучили структурные компоненты мозга — нейроны. Эти знания позволяют нам создавать различные модели для решения разнообразных задач.

Рассмотрим механизм работы нейрона. Нейрон принимает электрический сигнал, обрабатывает его и выдает измененный сигнал. Этот процесс можно сравнить с работой предсказательной машины или классификатора, которые также принимают входные данные, обрабатывают их и выдают результат.

Можно ли представить нейроны в виде линейных функций, как это делается в некоторых моделях? На первый взгляд идея кажется логичной, но она ошибочна. Ответ биологического нейрона не является результатом применения линейной функции к входу.

Исследования показывают, что нейроны не реагируют на входные сигналы сразу. Они подавляют входные сигналы до тех пор, пока их величина не достигнет определенного порога, вызывающего испускание выходного сигнала. Это можно представить как необходимость преодоления некой границы, чтобы нейрон дал ответ. Такая модель интуитивно понятна: нейроны не должны реагировать на незначительные шумы, а только на достаточно сильные сигналы, которые вызывают реакцию. На приведенной ниже иллюстрации (рисунок 3.6) показано, что нейрон выдает ответ только тогда, когда входной сигнал превышает определенный порог.



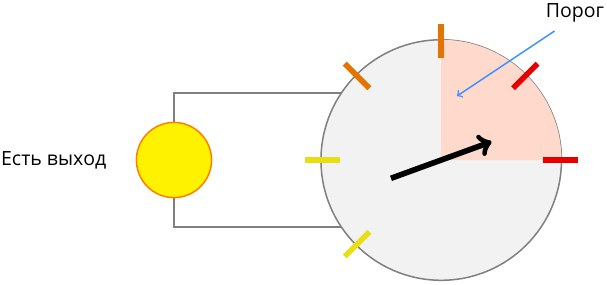


Рисунок 3.6 – Порог входного сигнала

Функция, принимающая входной сигнал и генерирующая выходной сигнал с учетом определенного порога, называется функцией активации. Существует множество различных функций активации. Одной из них является функция единичного скачка (см. рисунок 3.7).

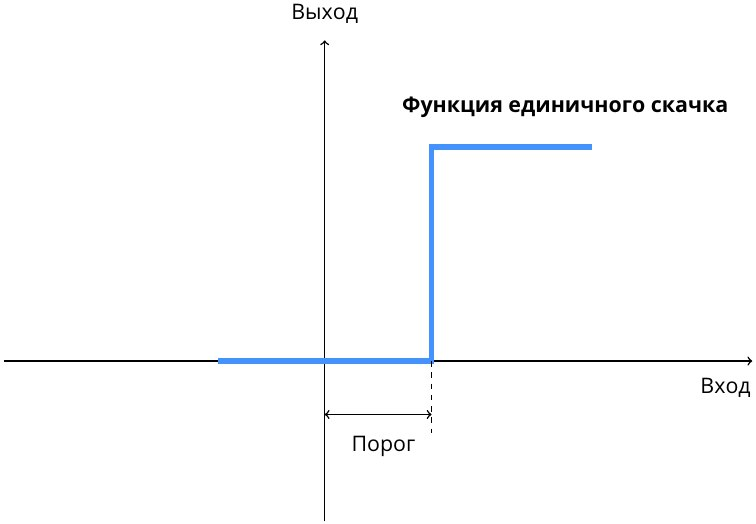
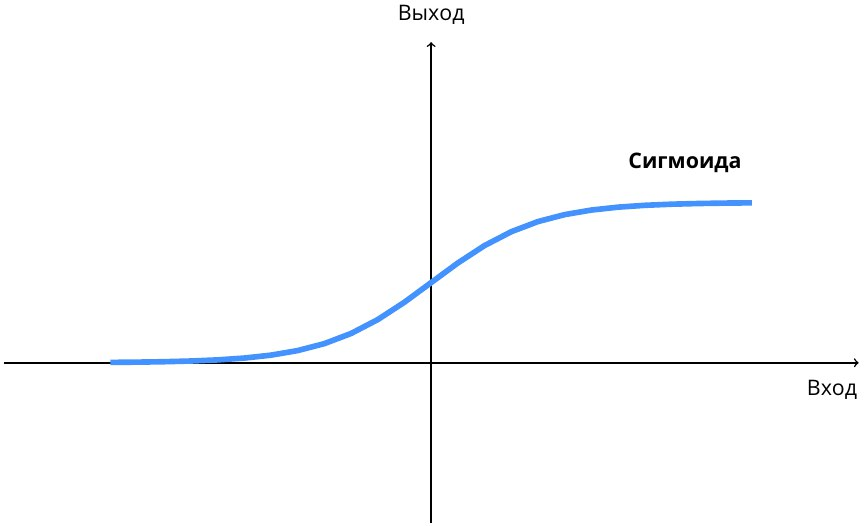


Рисунок 3.7 – График функции единичного скачка

Если значение входного сигнала мало, выходной сигнал равен нулю. Однако, как только входное значение достигает или превышает пороговое значение, сразу генерируется выходной сигнал. Искусственный нейрон с такой функцией активации более схож с реальным биологическим нейроном.

Функцию единичного скачка можно усовершенствовать. S-образная функция на графике ниже (рисунок 3.8), называемая сигмоидой, является более гладкой по сравнению с угловатой функцией единичного скачка.

Рисунок 3.7 – График сигмоиды

Именно такую S-образную сигмоиду мы будем использовать в наших нейронных сетях. Специалисты в области искусственного интеллекта иногда используют другие, схожие функции, однако сигмоида является простой и широко распространенной.

Уравнение сигмоиды, также известной как логистическая функция, выглядит следующим образом:

(уравнение 3.1)

Есть еще одна весомая причина для использования сигмоиды вместо множества других ​S-образных функций: её удобство для вычислений, которые нам предстоит проводить. С другими функциями этот процесс значительно усложняется.

Теперь давайте попробуем построить модель искусственного нейрона.

Во-первых, биологические нейроны имеют множество входов. Это не является новой концепцией для нас, поскольку логические функции принимают два аргумента.

Следует сложить сумму входов и передать сеё в качестве аргумента сигмоидной функции. Значение функции и будет выходом нейрона. Так и функционируют биологические нейроны. Диаграмма (рисунок 3.8) ниже прекрасно иллюстрирует принцип работы нейрона.

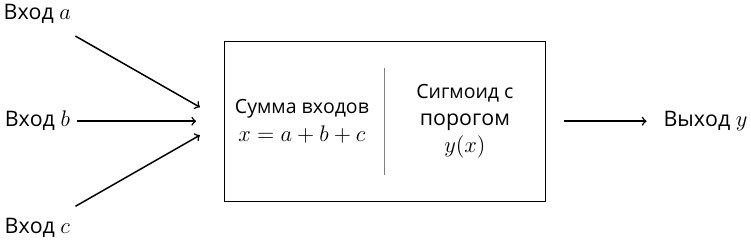


Рисунок 3.8 – Принцип работы нейрона

Если сумма всех входных сигналов меньше порогового значения, нейрон подавляет проходящий через него сигнал. Если сумма входных сигналов достигает или превышает порог, нейрон пропускает сигнал через сигмоидную функцию и выдает результат. Важно отметить, что нейрон может активироваться при поступлении сильного сигнала на один или несколько входов, в то время как на остальные входы поступают слабые сигналы. Либо нейрон может активироваться, если на все входы поступают не очень сильные сигналы, но их суммарное значение превышает порог. Именно эта особенность обуславливает двойственность и неопределенность, присущие как биологическим нейронам, так и компьютерным нейронным сетям.

В реальных нейронных сетях электрический сигнал принимается дендритами. Если сумма сигналов от всех дендритов превышает пороговое значение, нейрон передает измененный сигнал по аксону к дендритам других нейронов.

Иллюстрация ниже (рисунок 3.9) демонстрирует принцип связи между нейронами.

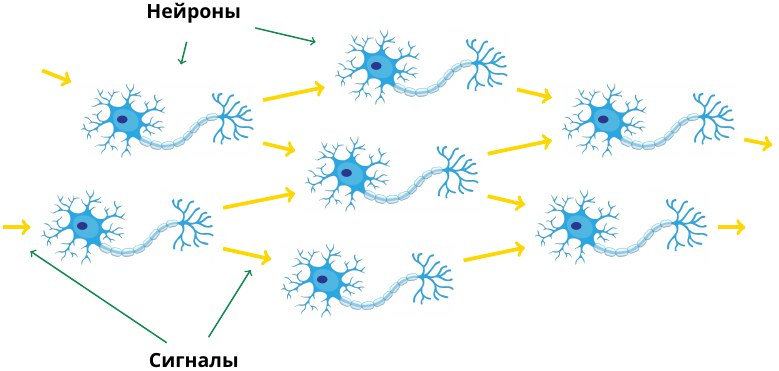


Рисунок 3.9 – Принцип связи между нейронами

Каждый нейрон получает сигналы от предыдущих нейронов и, если сигналы превышают порог, передает сигнал следующему множеству нейронов.

На рисунке 3.9 присутствует избыточное количество деталей. Для нашей искусственной модели нейронной сети неважно, какая форма у нейронов и их расположение. Мы будем считать, что наша искусственная нейросеть состоит из слоев нейронов, где каждый нейрон в одном слое связан со всеми нейронами предыдущего и следующего слоев.

Следующая диаграмма (см. рисунок 3.10) показывает три слоя, каждый из которых содержит по три искусственных нейрона.

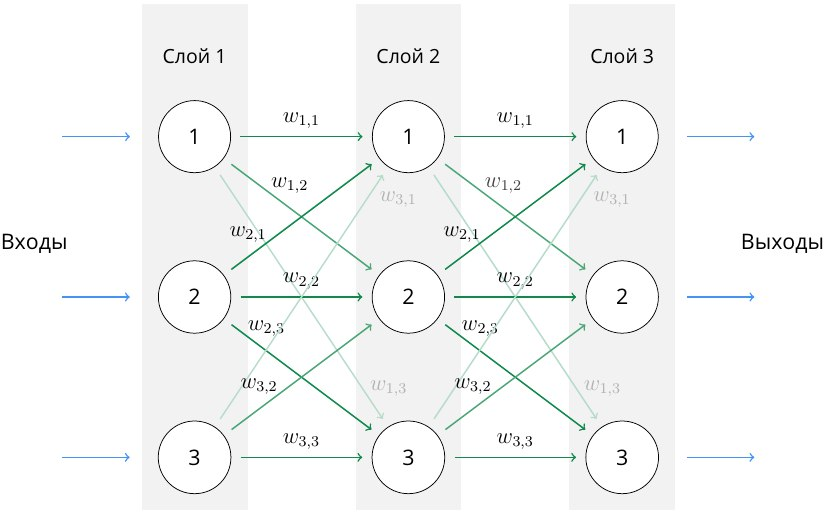


Рисунок 3.10 – Модель искусственной нейронной сети

Каждый нейрон связан со всеми нейронами предыдущего и следующего слоев. Однако, что именно изображено на этой диаграмме, относится к процессу обучения? Необходимо иметь параметры, которые можно изменять для минимизации ошибок.

Возможно изменять сумму входящих сигналов в нейрон. Возможно также настраивать параметры сигмоиды. Однако в практике эти вычисления сложны. Наиболее простой способ — изменять веса связей между искусственными нейронами.

Вес связи определяет его влияние. Увеличение веса усиливает проходящий через связь сигнал, а уменьшение веса уменьшает его.

Возникает вопрос: зачем связывать каждый нейрон со всеми нейронами в предыдущем и следующем слоях? В принципе, можно соединять нейроны любым образом. Однако чаще всего этого не делают, так как сложно предсказать, сколько связей потребуется для решения конкретной задачи. Кроме того, в процессе обучения нейронная сеть сама отбросит ненужные связи.

Во время обучения веса некоторых связей становятся равными нулю или очень близкими к нулю. Это означает, что такие связи просто не передают сигнал, поскольку любое число, умноженное на ноль, равно нулю. Такие "мертвые" связи не играют никакой роли. По мере обучения нейросеть сама определяет, какие связи ей нужны, а какие можно удалить. Поэтому наилучшим подходом является связывание всех нейронов друг с другом и предоставление сети решать, какие связи важны, а какие нет.

# Разработка программы

## **4.1. Спецификация программы**

Целью данного проекта является разработка и реализация нейронной сети, способной распознавать рукописные цифры. Программа должна быть способна обучаться на предоставленном наборе данных и предсказывать цифры на новых изображениях с высокой точностью.

Исходные данные: набор данных для обучения. Данные представляют собой изображения рукописных цифр, каждый из которых помечен соответствующей цифрой (от 0 до 9).

Формат данных: изображения представлены в виде одномерных массивов пикселей; каждое изображение имеет фиксированный размер 28x28 пикселей (784 пикселя в одномерном массиве); файлы конфигурации: файлы, содержащие параметры конфигурации сети и веса, сохраненные после обучения.

Требования к результату: точность распознавания. Программа должна достигать высокой точности распознавания (не менее 95% на тестовом наборе данных).

Время обучения: обучение должно завершаться в разумное время, обычно в течение нескольких минут на стандартном оборудовании.

Сохранение и загрузка весов: программа должна сохранять веса после обучения и загружать их для использования в последующих запусках без необходимости повторного обучения.

Архитектура нейронной сети: нейронная сеть должна иметь не менее трех слоев. Входной слой (784 нейрона для входных пикселей). Несколько скрытых слоев (количество и размер скрытых слоев определяется конфигурацией сети). Выходной слой (10 нейронов для каждой цифры от 0 до 9).

Функции активации: должны быть реализованы 3 функции активации, выбор которых при тестировании нейронной сети определяет пользователь.

Функциональные требования: инициализация сети с заданной конфигурацией слоев; инициализация весов и смещений.

Обучение сети: обработка обучающего набора данных; прямое распространение (forward propagation) для получения предсказаний; обратное распространение (backpropagation) для вычисления градиентов и обновления весов; регулярное вычисление точности на обучающем наборе.

Тестирование сети: обработка тестового набора данных; прямое распространение для получения предсказаний; вычисление точности на тестовом наборе; сохранение и загрузка весов; сохранение текущих весов сети в файл; загрузка весов из файла для продолжения работы или тестирования.

Особые случаи: если файл с данными не удается открыть или прочитать, программа должна выводить сообщение об ошибке и завершать работу корректно; программа должна проверять корректность формата данных и выдавать соответствующие сообщения об ошибках при необходимости; программа должна запрашивать подтверждение действий у пользователя (например, начинать ли обучение, тестирование или завершение работы).

Поведение программы в особых случаях: при отсутствии файла конфигурации или его повреждении программа должна выдавать сообщение об ошибке и завершать работу; при недостатке оперативной памяти для загрузки больших наборов данных программа должна обрабатывать ошибки, связанные с выделением памяти, и корректно завершать работу.

Аппаратные требования:

- минимальные: процессор с тактовой частотой 2 ГГц, 4 ГБ оперативной памяти;

- рекомендуемые: процессор с тактовой частотой 3 ГГц или выше, 8 ГБ оперативной памяти.

Программные требования:

- операционная система: Windows, macOS, Linux;

- компилятор: GCC, Clang, MSVC с поддержкой стандартов C++11 или выше;

- библиотеки: Стандартная библиотека C++, библиотека для работы с файлами (например, `<fstream>`).

## **4.2. Описание класса Matrix**

Поля класса:

1. double \*\*matrix:

- назначение: представляет собой двумерный массив для хранения элементов матрицы. Каждый элемент матрицы является вещественным числом (тип `double`);

- реализация: matrix реализован как указатель на массив указателей. Каждый указатель в этом массиве указывает на массив вещественных чисел, представляющий строки матрицы.

2. int row:

- назначение: хранит количество строк в матрице;

- реализация: это целочисленное поле, инициализируемое при создании матрицы.

3. int col:

- назначение: хранит количество столбцов в матрице;

- реализация: это целочисленное поле, также инициализируемое при создании матрицы.

Методы класса:

1. void Init(int row, int col):

- назначение: инициализирует матрицу с заданными количеством строк и столбцов, выделяя необходимую память;

-описание: вначале сохраняет переданные значения row и col в соответствующие поля класса; выделяет память для массива указателей размером row, где каждый указатель будет указывать на строку матрицы; затем для каждой строки выделяется память под массив элементов размером col; все элементы матрицы инициализируются нулями для обеспечения корректного начального состояния.

2. void Rand():

- назначение: заполняет матрицу случайными значениями;

- описание: проходит по всем элементам матрицы с помощью двойного цикла for. Для каждого элемента генерируется случайное значение с использованием функции rand(), которое затем масштабируется для обеспечения разумного диапазона значений. Случайные значения нормализуются с учетом количества строк, чтобы результаты были сопоставимы независимо от размеров матрицы.

3. static void Multi(const Matrix& m, const double\* b, int n, double\* c):

- назначение: умножает матрицу на вектор и сохраняет результат в новом векторе;

- описание: метод принимает три параметра: матрицу m, вектор b и размер вектора n; проверяет, совпадает ли количество столбцов матрицы m с размером вектора b. Если нет, выбрасывает исключение runtime\_error; для каждой строки матрицы вычисляет скалярное произведение строки и вектора, результат сохраняется в массиве c.

4. static void Multi\_T(const Matrix& m, const double\* b, int n, double\* c):

- назначение: умножает транспонированную матрицу на вектор;

- описание: метод аналогичен `Multi`, но работает с транспонированной матрицей; принимает те же параметры и проверяет, совпадает ли количество строк матрицы с размером вектора; для каждого столбца матрицы вычисляется скалярное произведение столбца и вектора, результат сохраняется в массиве c.

5. static void SumVector(double\* a, const double\* b, int n):

- назначение: складывает два вектора и сохраняет результат в первом векторе;

- описание: принимает два вектора a и b одинакового размера n; для каждого элемента векторов выполняет сложение и сохраняет результат в соответствующем элементе вектора a.

6. double& operator() (int i, int j):

- назначение: обеспечивает доступ к элементам матрицы по индексам, используя перегрузку оператора ();

- описание: принимает индексы строки i и столбца j; возвращает ссылку на элемент матрицы, что позволяет использовать синтаксис matrix(i, j) для доступа и изменения элементов матрицы.

7. friend std::ostream& operator << (std::ostream& os, const Matrix& m):

- назначение: перегружает оператор << для вывода матрицы в поток;

- описание: принимает поток вывода os и матрицу m; проходит по всем элементам матрицы, выводя их в поток с пробелом между элементами и новой строкой между строками матрицы; возвращает поток os для возможности цепочки операций вывода.

8. friend std::istream& operator >> (std::istream& is, Matrix& m):

- назначение: перегружает оператор >> для ввода матрицы из потока;

- описание: принимает поток ввода is и матрицу m; для каждой строки и столбца матрицы считывает значение элемента из потока ввода и сохраняет его в соответствующем элементе матрицы; возвращает поток is для возможности цепочки операций ввода.

Этот класс является ключевым элементом для выполнения матричных операций в процессе работы нейронной сети, используемой для распознавания рукописных цифр. Операции матричного умножения и сложения составляют основу вычислений, выполняемых в нейронных сетях при передаче данных через слои.

## **4.3. Описание класса ActivateFunction**

Поля класса:

1. activateFunc actFunc:

- назначение: Переменная actFunc`хранит текущую активирующую функцию, выбранную пользователем. Активирующая функция определяет, как будут обрабатываться значения в процессе вычислений нейронной сети;

- реализация: Тип данных activateFunc является перечислением (enum), включающим три возможных значения: sigmoid, ReLU, и thx. Это обеспечивает удобный и понятный способ задания активирующей функции.

Методы класса

1. void set():

-назначение: метод set() позволяет пользователю выбрать активирующую функцию из предложенных вариантов;

- описание: сначала метод выводит на экран приглашение выбрать одну из трех активирующих функций; затем считывает введенное пользователем значение и использует оператор switch для установки поля actFunc в одно из значений sigmoid, ReLU, или thx; в случае ввода некорректного значения выбрасывается исключение runtime\_error с сообщением об ошибке.

2. void use(double\* value, int n):

- назначение: метод use() применяет выбранную активирующую функцию к массиву значений;

- описание: vетод принимает указатель на массив value и его размер n; в зависимости от текущего значения поля actFunc метод применяет соответствующую активирующую функцию ко всем элементам массива; для функции sigmoid каждому элементу массива присваивается значение, рассчитанное по формуле 1 / (1 + exp(-value[i])); для функции ReLU каждому элементу массива присваивается значение: если элемент меньше нуля, то он умножается на 0.01; если элемент больше одного, то он корректируется по формуле 1 + 0.01 \* (value[i] - 1); для функции thx каждому элементу массива присваивается значение гиперболического тангенса: если элемент меньше нуля, то он умножается на 0.01; иначе применяется стандартная формула гиперболического тангенса.

3. void useDer(double\* value, int n):

-назначение: метод useDer() применяет производную выбранной активирующей функции к массиву значений;

- описание: метод принимает указатель на массив value и его размер n; в зависимости от текущего значения поля actFunc, метод применяет соответствующую производную активирующей функции ко всем элементам массива; для функции sigmoid каждому элементу массива присваивается значение производной сигмоиды value[i] \* (1 - value[i]); для функции ReLU каждому элементу массива присваивается значение: если элемент меньше нуля или больше одного, то он устанавливается в 0.01; иначе он устанавливается в 1; для функции thx каждому элементу массива присваивается значение производной гиперболического тангенса: если элемент меньше нуля, то он умножается на 0.01; иначе применяется стандартная формула производной гиперболического тангенса.

4. double useDer(double value):

- назначение: метод useDer(double value) применяет производную выбранной активирующей функции к одному значению;

- описание: метод принимает одно вещественное число value и возвращает значение производной активирующей функции, примененной к этому числу; в зависимости от текущего значения поля actFunc, метод рассчитывает и возвращает соответствующую производную; для функции sigmoid возвращается значение value \* (1 - value); для функции ReLU возвращается значение: если элемент меньше нуля или больше одного, то он устанавливается в 0.01; иначе он устанавливается в 1; для функции thx возвращается значение производной гиперболического тангенса: если элемент меньше нуля, то он умножается на 0.01; иначе применяется стандартная формула производной гиперболического тангенса.

В основной программе создается объект класса ActivateFunction, пользователь выбирает активирующую функцию через метод set(). Затем методы use() и useDer() применяются к массиву значений для демонстрации работы выбранной активирующей функции и её производной. Этот класс является важной частью процесса обучения и работы нейронной сети, обеспечивая гибкость в выборе и применении различных активирующих функций, что позволяет оптимизировать производительность сети под конкретные задачи.

## **4.4. Описание класса NetWork**

Поля класса:

1. int L:

- назначение: переменная L хранит количество слоев в нейронной сети;

- реализация: инициализируется и задается в методе Init(data\_NetWork data).

2. int\* size:

- назначение: указатель size указывает на массив, в котором хранится количество нейронов в каждом слое сети;

- реализация: размерность массива определяется количеством слоев L и инициализируется в методе Init(data\_NetWork data).

3. ActivateFunction actFunc:

- назначение: объект класса ActivateFunction используется для выбора и применения активирующих функций в нейронной сети;

- реализация: активирующая функция устанавливается методом set().

4. Matrix\* weights:

- назначение: указатель weights указывает на массив объектов класса Matrix, представляющих матрицы весов для каждого слоя сети;

- реализация: инициализируется в методе Init(data\_NetWork data).

5. double\*\* bios:

- назначение: указатель `bios` указывает на массив массивов, содержащих значения смещений (biases) для каждого слоя сети;

- реализация: инициализируется и задается в методе Init(data\_NetWork data).

6. double\*\* neurons\_val:

- назначение: указатель neurons\_val указывает на массив массивов, содержащих значения нейронов для каждого слоя сети;

- реализация: инициализируется и задается в методе Init(data\_NetWork data).

7. double\*\* neurons\_err:

- назначение: указатель neurons\_err указывает на массив массивов, содержащих значения ошибок для каждого нейрона в сети;

- реализация: инициализируется и задается в методе Init(data\_NetWork data).

8. double\* neurons\_bios\_val:

- назначение: указатель neurons\_bios\_val указывает на массив значений смещений для каждого слоя;

- реализация: инициализируется и задается в методе Init(data\_NetWork data).

Методы класса:

1. void Init(data\_NetWork data):

- назначение: метод Init инициализирует параметры нейронной сети на основе структуры data\_NetWork;

-описание: устанавливает активирующую функцию методом actFunc.set(); инициализирует количество слоев L и массив size; создает массивы для весов, смещений, значений нейронов и их ошибок; заполняет матрицы весов случайными значениями и инициализирует смещения случайными значениями в диапазоне.

2. void PrintConfig():

- назначение: метод PrintConfig выводит конфигурацию нейронной сети, включая количество слоев и количество нейронов в каждом слое;

- описание: использует стандартный поток вывода cout для отображения конфигурации сети.

3. void SetInput(double\* values):

- назначение: метод SetInput устанавливает входные значения нейронов для первого слоя сети;

- описание: копирует значения из входного массива `values` в массив `neurons\_val` для первого слоя.

4. double ForwardFeed():

- назначение: метод ForwardFeed выполняет прямое распространение (forward propagation) входных данных через сеть;

- описание: последовательно умножает матрицы весов на значения нейронов предыдущего слоя, добавляет смещения и применяет активирующую функцию; возвращает индекс нейрона с максимальным значением в последнем слое сети, что представляет собой предсказанный класс.

5. int SearchMaxIndex(double\* value):

- назначение: метод SearchMaxIndex находит индекс максимального значения в массиве;

- описание: проходит по массиву value и возвращает индекс наибольшего элемента.

6. void PrintValues(int L):

- назначение: метод PrintValues выводит значения нейронов для указанного слоя L;

- описание: использует стандартный поток вывода cout для отображения значений нейронов слоя.

7. void BackPropogation(double expect):

- назначение: метод BackPropogation выполняет обратное распространение ошибки (backpropagation) для обновления весов и смещений сети;

- описание: вычисляет ошибки для нейронов выходного слоя на основе ожидаемого значения expect; последовательно вычисляет ошибки для нейронов предыдущих слоев, умножая ошибки следующего слоя на транспонированные матрицы весов и применяя производные активирующей функции.

8. void WeightsUpdater(double lr):

- назначение: метод WeightsUpdater обновляет веса и смещения сети на основе рассчитанных ошибок и заданного коэффициента обучения lr;

- описание: проходит по всем весам и смещениям, обновляя их значения на основе ошибок нейронов и значений коэффициента обучения.

9. void SaveWeights():

- назначение: метод SaveWeights сохраняет текущие значения весов и смещений сети в файл;

- описание: открывает файл Weights.txt для записи и сохраняет в него значения всех матриц весов и массивов смещений; закрывает файл после завершения записи.

10. void ReadWeights():

- назначение: метод ReadWeights загружает значения весов и смещений сети из файла;

- описание: открывает файл Weights.txt для чтения и загружает из него значения всех матриц весов и массивов смещений; закрывает файл после завершения чтения.

## **4.5. Описание функции main**

Структуры:

1. struct data\_info:

- назначение: Структура data\_info используется для хранения информации о каждом примере из обучающего набора данных;

- поля: double\* pixels: Указатель на массив, хранящий значения пикселей изображения; int digit: Цифра, изображенная на изображении.

2. struct data\_NetWork:

- назначение: структура data\_NetWork используется для хранения информации о конфигурации нейронной сети;

-поля: int L: Количество слоев в нейронной сети; int\* size: Указатель на массив, в котором хранится количество нейронов в каждом слое.

Функции:

1. void displayAsciiImage(string filename):

- назначение: функция displayAsciiImage выводит ASCII изображение, сохраненное в файле;

- описание: открывает файл filename для чтения; читает файл построчно и выводит каждую строку на экран; закрывает файл после завершения чтения. 2. data\_NetWork ReadDataNetWork(string path):

- назначение: функция ReadDataNetWork считывает конфигурацию нейронной сети из файла;

- описание: открывает файл path для чтения; если файл не удается открыть, выводит сообщение об ошибке и отображает ASCII изображение ошибки; читает данные из файла и заполняет структуру data\_NetWork; закрывает файл после завершения чтения;

-возвращает: структуру data\_NetWork, содержащую конфигурацию сети. 3. data\_info ReadData(string path, const data\_NetWork& data\_NW, int& examples):

-назначение: функция ReadData считывает данные примеров из файла;

-описание: открывает файл path для чтения; если файл не удается открыть, выводит сообщение об ошибке и отображает ASCII изображение ошибки; читает количество примеров и инициализирует массив структур data\_info; заполняет массив данных, считывая значения пикселей и соответствующие цифры; закрывает файл после завершения чтения;

-возвращает: указатель на массив структур data\_info, содержащих данные примеров.

Функция main:

1. Инициализация и настройка нейронной сети:

- создает объект NetWork NW;

- считывает конфигурацию сети из файла Config.txt с помощью функции ReadDataNetWork;

- инициализирует нейронную сеть с помощью метода Init класса NetWork;

- выводит конфигурацию сети с помощью метода PrintConfig.

2. Цикл обучения и тестирования:

- запрашивает у пользователя, начинать ли обучение сети;

- если пользователь выбирает обучение, считывает данные из файла lib\_MNIST\_edit.txt с помощью функции ReadData;

- запускает цикл обучения, в котором повторяется процесс прямого распространения (forward propagation), обратного распространения ошибки (backpropagation) и обновления весов;

- вычисляет количество правильных ответов и выводит процент правильных ответов и время обучения;

- сохраняет веса сети в файл после завершения обучения.

3. Тестирование сети:

- запрашивает у пользователя, начинать ли тестирование сети;

- если пользователь выбирает тестирование, считывает данные из файла lib\_10k.txt с помощью функции ReadData;

- выполняет прямое распространение для каждого тестового примера и сравнивает предсказанный результат с правильным ответом;

- выводит процент правильных ответов и отображает ASCII изображения правильных цифр.

4. Завершение работы программы:

- запрашивает у пользователя, повторить ли процесс обучения и тестирования;

- завершает работу программы после выхода из цикла.

Вспомогательные действия:

1. вывод ASCII изображений: процессе выполнения программы используются ASCII изображения для визуального отображения процесса загрузки данных, ошибок и других событий;

2. работа с файлами: открытие, чтение и запись данных в файлы реализованы с использованием стандартных потоков ввода-вывода C++ (ifstream и ofstream);

3. обработка ошибок: при возникновении ошибок ввода-вывода выводятся соответствующие сообщения и отображаются ASCII изображения ошибки.

# 5. ТЕСТИРОВАНИЕ

## **5.1. Описание входных и выходных данных**

Входные данные:

1. Изображения рукописных цифр:

- формат: изображения предоставляются в виде массивов пикселей. Обычно изображения имеют размер 28x28 пикселей, что соответствует 784 входным признакам для нейронной сети;

- тип данных: значения пикселей представлены в виде массивов double размером 784 для каждого изображения.

1. Метаданные изображений:

- метка класса (цифра): каждое изображение сопровождается меткой, указывающей, какая цифра (от 0 до 9) изображена. Это целочисленные значения, представляющие классы для классификации;

- формат: метки хранятся в виде целочисленного значения для каждого изображения.

1. Конфигурация нейронной сети:

- количество слоев (L): Целочисленное значение, представляющее количество слоев в сети;

- размеры слоев: массив целых чисел, представляющий количество нейронов в каждом слое сети.

Выходные данные:

1. Предсказанные метки классов:

- формат: для каждого входного изображения сеть генерирует предсказание, представляющее метку класса (от 0 до 9);

- тип данных: целочисленные значения, представляющие предсказанные классы.

1. Проценты правильных предсказаний:

- формат: процентное соотношение правильно предсказанных меток к общему количеству тестовых изображений;

- тип данных: вещественное число, представляющее точность сети.

1. Ошибки нейронов:

- формат: массивы значений ошибок для каждого нейрона на каждом слое, используемые для обратного распространения ошибки;

- тип данных: массивы double.

1. Веса и смещения:

- формат: матрицы весов и векторы смещений для каждого слоя сети;

- тип данных: массивы double для весов и смещений.

## **5.2. Результаты тестирования**

Среда тестирования: операционная система Windows 10, среда разработки Visual Studio 2022, компилятор: Microsoft C++ Compiler.

Тест 1. Вывод ошибки при открытии файла: проверка корректности работы программы при выводе сообщения об ошибке при открытии файла. (см. рисунок 5.1.)

Ожидаемый результат: корректный вывод сообщения об ошибке при открытии файла.

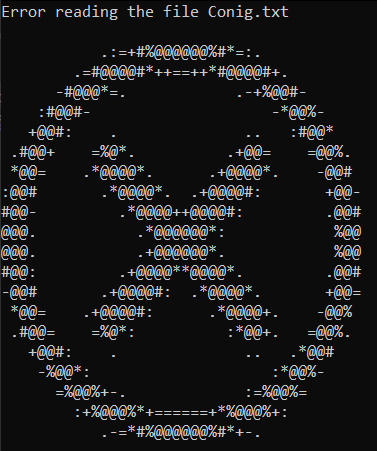


Рисунок 5.1. – Вывод сообщения об ошибке при открытии файла

Вывод: результат соответствует ожиданиям. Тест 1 успешно пройден.

Тест 2. Вывод сообщения об успешной загрузке файла с данными для обучения (см. рисунок 5.2).

Ожидаемый результат: корректная загрузка файла с данными для обучения и вывод соответствующего сообщения.



Рисунок 5.2. – Вывод сообщения об успешной загрузке данных для обучения

Вывод: результат соответствует ожиданиям. Тест 2 успешно пройден.

Тест 3. Обучение нейронной сети (см. рисунок 5.3).

Ожидаемый результат: правильные ответы, выданные нейронной сетью, должны составлять не менее 95% процентов от всех поданных на обучение данных (количество эпох равно 20).

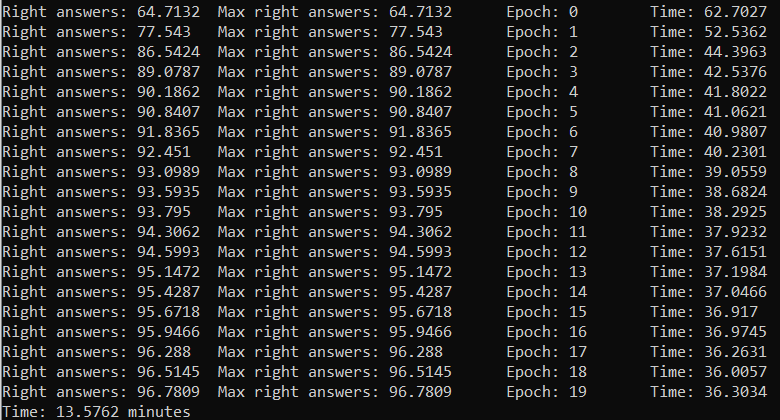


Рисунок 5.3. – Результаты обучения нейронной сети

Вывод: результат соответствует ожиданиям. Тест 3 успешно пройден.

Тест 4. Проверка работы нейронной сети на тестовых данных (см. рисунок 5.4).

Ожидаемый результат: правильные ответы, выданные нейронной сетью, должны составлять не менее 85% процентов от всех поданных на тестирование данных.

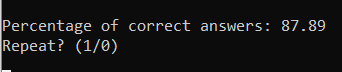


Рисунок 5.4. – Результат тестирования нейронной сети

Вывод: результат соответствует ожиданиям. Тест 4 успешно пройден.

Тест 5. Вывод изображения угаданного числа (см. рисунок 5.5).

Ожидаемый результат: программа должна выводить изображения угаданных чисел.

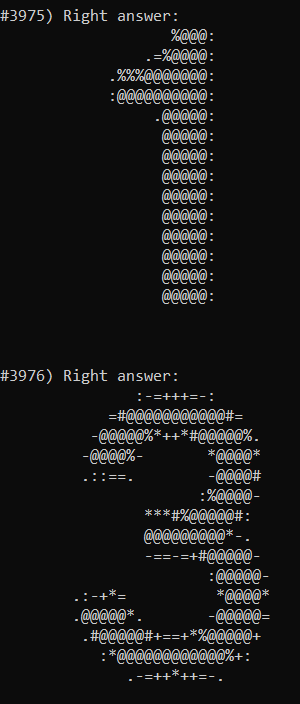


Рисунок 5.5. – Вывод изображения угаданных чисел

Вывод: результат соответствует ожиданиям. Тест 5 успешно пройден.

Тест 5. Повторное обучение нейронной сети (см. рисунок 5.6).

Ожидаемый результат: при повторном запуске обучения нейронной сети должен быть повышен процент правильных ответов.

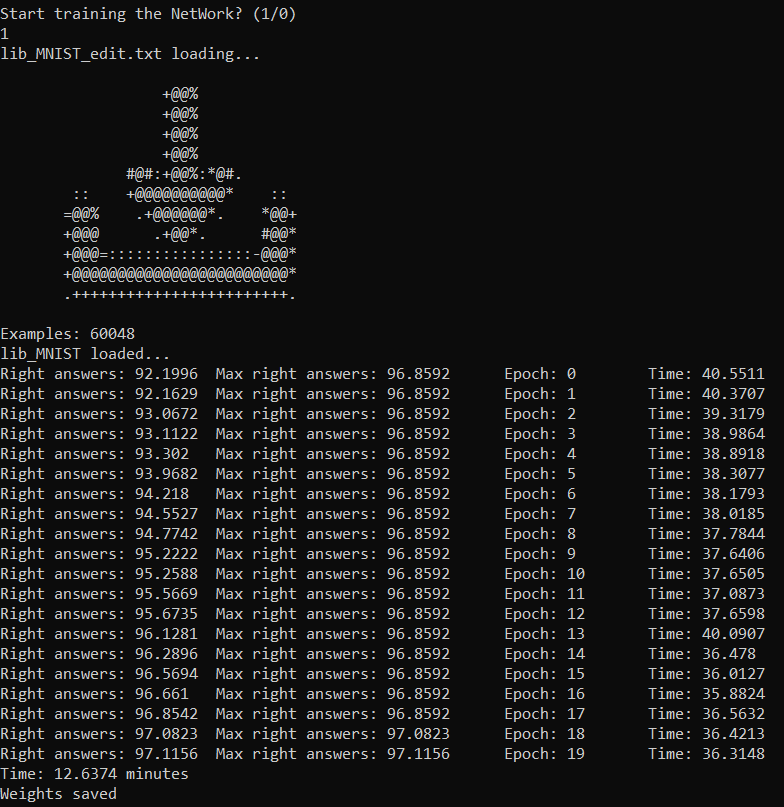


Рисунок 5.6. – Повторное обучение нейронной сети

Вывод: результат соответствует ожиданиям. Тест 6 успешно пройден.

# ЗАКЛЮЧЕНИЕ

В данной курсовой работе была разработана нейронная сеть для распознавания рукописных цифр на языке программирования C++. Основная цель проекта заключалась в создании эффективной и производительной модели, способной классифицировать изображения цифр с высокой точностью. Работа включала в себя проектирование и реализацию всех необходимых компонентов нейронной сети, включая алгоритмы прямого распространения, обратного распространения ошибки и обновления весов, а также поддержку различных функций активации.

Основные результаты

1. Разработка архитектуры нейронной сети:

- создана модульная структура, позволяющая гибко изменять количество слоев и нейронов в каждом слое;

- реализована возможность задания различных функций активации (сигмоида, ReLU, гиперболический тангенс) и их производных для эффективного обучения.

1. Алгоритмы и математическое обеспечение:

- реализованы алгоритмы матричного умножения и транспонирования матриц, необходимые для вычисления взвешенных сумм на каждом слое;

- созданы функции для выполнения прямого распространения сигнала (forward feed) и обратного распространения ошибки (backpropagation), что обеспечивает корректное обучение сети;

- разработан алгоритм обновления весов и смещений с использованием метода градиентного спуска.

1. Обработка данных:

- разработан механизм загрузки и предварительной обработки данных MNIST, включающий нормализацию входных значений и преобразование меток классов;

- реализованы функции для визуализации входных изображений в текстовом формате для отладки и тестирования.

1. Тестирование и результаты:

- проведено обучение сети на тренировочном наборе данных MNIST. Достигнута высокая точность классификации на тестовых данных;

- процессе тестирования были выявлены и устранены ошибки, что позволило повысить надежность и стабильность работы модели.

Разработанная нейронная сеть продемонстрировала высокую точность в задаче распознавания рукописных цифр, что подтверждает правильность выбранных методов и алгоритмов. Использование языка C++ позволило добиться высокой производительности и оптимизации вычислений, что особенно важно при работе с большими объемами данных.

Результаты данной курсовой работы могут быть полезны для дальнейшего изучения и применения нейронных сетей в различных задачах классификации и распознавания образов. Разработанная архитектура и реализованные алгоритмы могут быть расширены и адаптированы для решения других задач машинного обучения.

Таким образом, данная курсовая работа заложила прочный фундамент для дальнейшего изучения и разработки нейронных сетей на языке C++, продемонстрировав их потенциал и широкие возможности применения.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение / пер. с анг. А. А. Слинкина. – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.

2. Тарик Рашид. Создаём нейронную сеть. :Пер. с англ. СПб.: ООО «Альфа-книга», 2017. – 272с.

3. Лафоре Р. Объектно-ориентированное программирование в С++. Классика Computer Science. 4-е изд. – СПб.: Питер, 2011. – 928 с.

4. Роберт Мартин. Чистый код. Создание, анализ и рефакторинг / Роберт Мартин – СПБ.: БХВ - Санкт-Петербург. - 2020. — 425с.

5. Руководство по языку программирования C++ // METANIT.COM  
[Электронный ресурс]. － 2024. － Режим доступа: https://metanit.com/cpp/tutorial/.－ Дата доступа: 07.03.2024.

6. Справочник по языку C++ // Microsoft Learn  
[Электронный ресурс]. － 2024. － Режим доступа:  
https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=msvc-170. －  
Дата доступа: 15.04.2024.  
 7. Методы оптимизации нейронных сетей // habr.com [Электронный ресурс]. – 2024. – Режим доступа: <https://habr.com/ru/articles/318970/> － Дата доступа: 17.04.2024.

8. Обзор градиентных методов в задачах математической оптимизации // habr.com [Электронный ресурс]. – 2024. – Режим доступа: <https://habr.com/ru/articles/413853/> - Дата доступа: 17.04.2024.

9. WPF и C++SS | Полное руководство – Metanit. [Электронный ресурс] – 2021. – Режим доступа: <https://metanit.com/sharp/wpf/> - Дата доступа: 15.11.2024.

10. Обучаем нейросеть распознавать цифры на выборке от MNIST. Реализация алгоритма обратного распространения на C# // habr.com [Электронный ресурс]. – 2024. – Режим доступа: <https://habr.com/ru/articles/708928/> - Дата доступа: 21.04.2024.