СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире нейронные сети стали одним из самых актуальных направлений в области искусственного интеллекта. Они находят широкое применение в различных областях, таких как медицина, финансы, робототехника, компьютерное зрение и многие другие. Одним из популярных языков программирования для работы с нейронными сетями является C++.

C++ - это мощный и универсальный язык программирования, который широко используется при разработке программных приложений, системного программного обеспечения, драйверов устройств и встроенного микропрограммного обеспечения.

С тех пор этот язык эволюционировал и стал одним из самых популярных и широко используемых языков программирования в мире. Его популярность обусловлена его эффективностью, гибкостью и широким спектром применений, для которых он может быть использован. C++ известен своей высокой производительностью, поскольку позволяет выполнять низкоуровневые манипуляции с оборудованием и памятью, что делает его подходящим для разработки ресурсоемких приложений.

1 ПОСТАНОВКА ЗАДАЧИ

Исследовать принципы работы нейронных сетей и их реализацию на языке программирования C++. Реализовать класс нейронной сети. Класс для работы с матрицами, чтобы работать с весами нейронов. Разработать иерархию классов для использовония математических функций с использованием наследования. Реализовать методы для обучения, тестирования и другие методы, в зависимости от специфики задачи. Реализовать графический интерфейс для тестирования нейронной сети.

2 ОБЗОР ЛИТЕРАТУРЫ

2.1 Обзор методов и алгоритмов решения поставленной задачи

MNIST (Modified National Institute of Standards and Technology) - это набор изображений рукописных цифр, состоящий из 60 000 обучающих и 10 000 тестовых примеров. Предварительно база данных была обработана: изображение преобразовано числовой формат. 0 – черный пиксель, 1 – белый.

Многослойный перцептрон представляет собой нейронную сеть, состоящую из нескольких слоев нейронов, включая входной, скрытые и выходной слои.

Функция активации: для каждого нейрона применяется функция активации, например, сигмоидная функция, или функция ReLU, или гиперболический тангенс.

Прямое распространение: входные данные передаются через сеть, проходя через каждый слой и вычисляя выходные значения нейронов.

Обратное распространение ошибки: используется для обновления весов нейронов на основе разницы между предсказанными и ожидаемыми значениями.

Функция потерь: используется для измерения разницы между предсказанными и ожидаемыми значениями. В моем случае среднеквадратичная ошибка.

Для обновления весов нейронов используются алгоритмы оптимизации: градиентный спуск

Матрицы весов: веса нейронов могут быть представлены в виде матриц, где каждый элемент матрицы соответствует весу связи между двумя нейронами.

Матрицы весов сохранены в файлы для последующего использования или загрузки.

Инициализация весов: начальные значения весов нейронов инициализируются случайным образом.

Процесс обучения: прямое распространение, вычисление ошибки, обратное распространение и обновление весов повторяются до достижения определенного критерия остановки. В моем случае определенного числа эпох обучения.

После обучения сети производится оценка ее точности на тестовой выборке.

Более подробно о многослойном перцептроне и функциях активации можно узнать в источниках [3] и [4].

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описываются входные и выходные данные программы, диаграмма классов, а также приводится описание используемых классов и их методов.

3.1 Структура входных и выходных данных

Таблица 3.1 – файл с информацией нейронной сети *Config.txt*

Количество слоёв нейронной сети	Массив с количеством нейронов на каждом слое

Таблица 3.2 – файл с базой данных для обучения *NewBase.txt*

Нарисованная цифра	Массив пикселей

Таблица 3.3 – файл с базой данных для тестирования *NewBaseTEST.txt*

Нарисованная цифра	Массив пикселей

Таблица 3.4 – файл с матрицами весов *Weight.txt*

Матрица весов

Таблица 3.5 – файл с тестовой цифрой *test.txt*

Массив пикселей

Пример для таблиц 3.2 и 3.3:

0 – нарисованная цифра
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
массив пикселей

3.2 Разработка диаграммы классов

Диаграмма классов для курсового проекта приведена в Приложении А.

3.3 Описание классов

3.3.1 Класс нейронной сети

Класс `NetWork` представляет нейронную сеть и содержит поля и методы для ее работы.

Поля класса:

`int L` - количество слоев в нейронной сети.

`int* size` - массив, содержащий количество нейронов на каждом слое.

`ActivateFunction actFunc` - функция активации, которая определяет, каким образом будет преобразовываться входной сигнал нейрона.

`Matrix* weight` - матрица весов, хранящая значения весов между нейронами на разных слоях.

`double** bios` - матрица смещений, хранящая значения смещений для каждого нейрона на каждом слое.

`double** neuros_val` - матрица значений нейронов, хранящая значения активаций для каждого нейрона на каждом слое.

`double** neuros_err` - матрица ошибок нейронов, хранящая значения ошибок для каждого нейрона на каждом слое.

`double* neuros_bios_val` - массив значений смещений нейронов.

Методы класса:

Конструктор `NetWork(data_NetWork data)` принимает объект `data_NetWork`, и инициализирует нейронную сеть, и устанавливает значения полей класса.

Деструктор `~NetWork()` освобождает память, выделенную под поля класса: матрицы и массивы, хранящие значения весов, смещений, активаций и ошибок нейронов.

Метод `PrintConfig()` выводит информацию о сети на экран: количество слоев, количество нейронов на каждом слое

Метод `SetInput(double* values)` устанавливает значения входных нейронов на первом слое нейронной сети на основе переданных значений.

Метод `ForwarFeed()` прямого распространения сигнала по нейронной сети, вычисляет значения активаций для каждого нейрона на каждом слое нейронной сети и возвращает значение активации выходного нейрона..

Метод `SearchMaxIndex(double* values)` метод находит индекс элемента с максимальным значением в переданном массиве и возвращает его.

Метод `PrintValue(int L)` выводит значения активаций для каждого нейрона на указанном слое нейронной сети.

Метод `BackPropagation(double expert)` обратного распространения ошибки по нейронной сети, вычисляет значения ошибок для каждого нейрона на каждом слое нейронной сети на основе переданного ожидаемого значения.

Метод `WeightsUpdater(double lr)` обновляет значения весов между нейронами на разных слоях нейронной сети на основе переданного коэффициента обучения.

Метод `SaveWeights()` сохраняет значения весов между нейронами на разных слоях нейронной сети в файл для последующего использования.

Метод `void ReadWeights()` считывает значения весов между нейронами на разных слоях нейронной сети из файла для использования в нейронной сети.

Класс `NetWork` представляет основу для создания и использования нейронных сетей, позволяя инициализировать, обучать и использовать нейронную сеть для решения различных задач

3.3.2 Класс матрицы

Класс `Matrix` представляет собой матрицу и содержит поля и методы для ее работы с ней.

Поля класса:

`double** matrix` - двумерный массив, хранящий значения элементов матрицы. Это поле является указателем на двумерный массив типа double и используется для хранения значений элементов матрицы.

`int row` - количество строк в матрице. Это поле хранит информацию о количестве строк в матрице.

`int col` - количество столбцов в матрице. Это поле хранит информацию о количестве столбцов в матрице.

Методы класса:

Конструктор `Matrix()` выделяет место под массивы для знчений матрицы.

Деструктор `~Matrix()` освобождает память, выделенную под массив значений матрицы.

Метод `Init(int row, int col)` принимает количество строк и столбцов и выделяет память под массив значений матрицы.

Метод `Rand()` заполненяет матрицу случайными значениями в диапазоне от 0 до 1.

Метод `Multi(const Matrix& m, const double* b, int n, double* c)` умножает матрицы на вектор. Этот метод умножает матрицу на вектор и сохраняет результат в переданный массив.

Метод `Multi_T(const Matrix& m, const double* b, int n, double* c)` умножает транспонированную матрицу на вектор. Этот метод умножает транспонированную матрицу на вектор и сохраняет результат в переданный массив.

Метод `SumVector(double* a, const double* b, int n)` складывает два вектора и сохраняет результат в первом векторе.

Оператор `() (int i, int j)` позволяет получить доступ к элементам матрицы по индексам строки и столбца.

Оператор `<< (std::ostream& os, const Matrix& m)` позволяет вывести значения элементов матрицы в поток вывода.

Оператор `>> (std::istream& is, Matrix& m)` позволяет ввести значение элементов матрицы в поток ввода

3.3.3 Класс функции активации

Класс `ActivateFunction` представляет собой функцию активации для нейронной сети и содержит поле и методы для ее работы с ней.

Поля класса:

`activateFunc actFunc` - перечисление, определяющее выбранную функцию активации. Это поле хранит информацию о выбранной функции активации.

Методы класса:

Конструктор `activateFunction()` инициализирует объект класса `ActivateFunction`. И устанавливает функцию ReLU.

Метод `set()` устанавливает функцию активации. Этот метод позволяет пользователю выбрать функцию активации из сигмойды, ReLU и гиперболического тангенса, которая будет использоваться в нейронной сети.

Метод `use(double* value, int n)` применяет функцию активации к массиву значений и сохраняет результат в этом же массиве.

Метод `useDer(double* value, int n)` применяет производную функции активации к массиву значений и сохраняет результат в этом же массиве.

Перегруженный метод `useDer(double value)` применяет производную функции активации к значению сохраняет результат.

Также в классе `ActivateFunction` используются наследование от классов `Sigmoid`, `ReLU` и `Thx`, которые представляют собой различные функции активации.

3.3.4 Класс Сигмойды

Математическая формула:

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.1}$$

График функции приведен на рисунке 3.1.

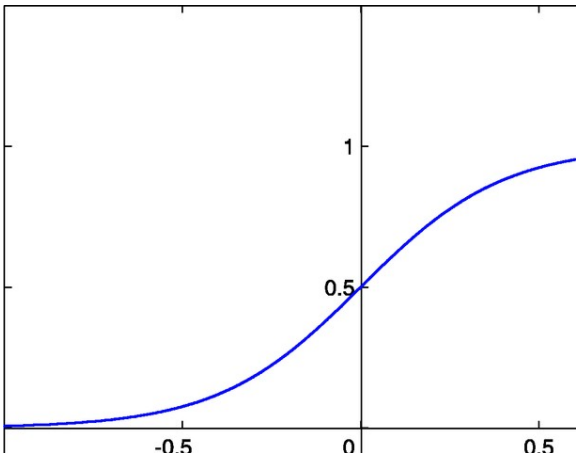

Рисунок 3.1– График сигмойды

Класс `Sigmoid` представляет собой функцию активации для нейронной сети и содержит методы для ее работы с ней.

Методы класса:

Метод `use(double* value, int n)` применяет функцию активации к массиву значений и сохраняет результат в этом же массиве.

Метод `useDer(double* value, int n)` применяет производную функции активации к массиву значений и сохраняет результат в этом же массиве.

Перегруженный метод `useDer(double value)` применяет производную функции активации к значению сохраняет результат.

3.3.5 Класс ReLU

Математическая формула:

$$f(x) = \begin{cases} x, \text{ если } x > 0 \\ 0{,}01x, \text{если } x \leq 0 \end{cases} \quad (3.2)$$

График функции приведен на рисунке 3.2.

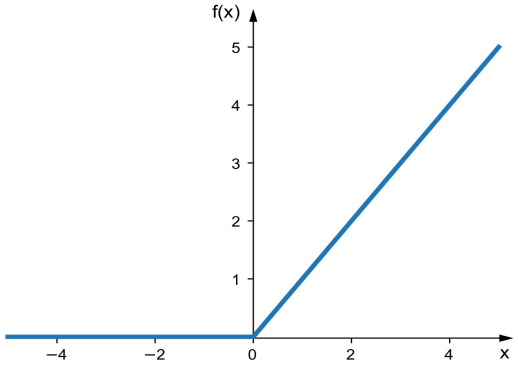

Рисунок 3.2– График ReLU

Класс `ReLU` представляет собой функцию активации для нейронной сети и содержит методы для ее работы с ней.

Методы класса:

Метод `use(double* value, int n)` применяет функцию активации к массиву значений и сохраняет результат в этом же массиве.

Метод `useDer(double* value, int n)` применяет производную функции активации к массиву значений и сохраняет результат в этом же массиве.

Перегруженный метод `useDer(double value)` применяет производную функции активации к значению сохраняет результат.

3.3.6 Класс Гиперболического тангенса

Математическая формула:

$$f(x) = th(x) = \frac{sh(x)}{ch(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} \quad (3.3)$$

График функции приведен на рисунке 3.3.

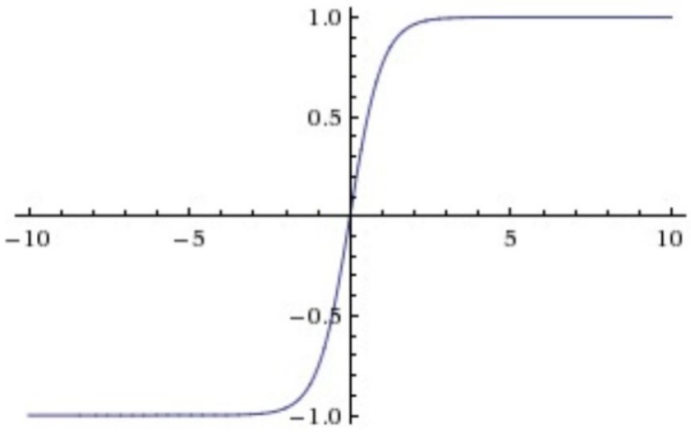

Рисунок 3.3– График гиперболического тангенса

Класс `Thx` представляет собой функцию активации для нейронной сети и содержит методы для ее работы с ней.

Методы класса:

Метод `use(double* value, int n)` применяет функцию активации к массиву значений и сохраняет результат в этом же массиве.

Метод `useDer(double* value, int n)` применяет производную функции активации к массиву значений и сохраняет результат в этом же массиве.

Перегруженный метод `useDer(double value)` применяет производную функции активации к значению сохраняет результат.

3.3.7 Класс окна для рисования

Класс `PaintScene` представляет собой виджет для рисования и содержит следующие поля и методы.

Поля класса:

`bool draw` - флаг, указывающий на то, находится ли пользователь в режиме рисования. Если значение этого поля равно `true`, то пользователь рисует на виджете, если `false` - то нет.

`QVector<QPointF> vv` - вектор точек, представляющих собой координаты точек, которые пользователь нарисовал на виджете.

`QImage pic` - изображение, на котором отображается результат рисования.

Методы класса:

Конструктор `PaintScene(QWidget *parent = nullptr)` инициализирует объект класса PaintScene и устанавливает родительский виджет.

Метод `paintEvent(QPaintEvent*)` отрисовывает изображение на виджете. Этот метод вызывается автоматически при необходимости перерисовки виджета и отрисовывает содержимое изображения на виджете.

Метод `mousePressEvent(QMouseEvent*)` обрабатывает событие нажатия кнопки мыши. Этот метод вызывается при нажатии кнопки мыши на виджете и сохраняет координаты точки в векторе vv, если пользователь находится в режиме рисования (если поле `draw` равно `true`).

Метод `void mouseMoveEvent(QMouseEvent*)` обрабатывает событие перемещения мыши. Этот метод вызывается при перемещении мыши по виджету и добавляет координаты точки в вектор vv, если пользователь находится в режиме рисования.

Метод `mouseReleaseEvent(QMouseEvent*)` обрабатывает событие отпускания кнопки мыши. Этот метод вызывается при отпускании кнопки мыши на виджете и завершает режим рисования (устанавливает поле `draw` в `false`).

Метод `clear()` очищает виджет: удаляет все точки из вектора `vv` и перерисовывает виджет, чтобы отобразить пустое изображение.

В классе `PaintScene` также используется наследование от класса `QWidget` и содержится объект `ui`, который представляет собой интерфейс виджета и позволяет управлять его элементами.

3.3.8 Класс главного окна

Класс `MainWindow` представляет собой главное окно приложения и содержит следующие поля.

Поля класса:

`PaintScene* PS` - указатель на объект класса `PaintScene`, который представляет виджет для рисования.

`NetWork NW` - объект класса `NetWork`, который представляет нейронную сеть.

`data_NetWork NW_config` - объект структры `data_NetWork`, который представляет информацию о нейронной сети.

`Ui::MainWindow *ui` - указатель на объект класса `Ui::MainWindow`, который представляет интерфейс главного окна приложения.

Методы класса:

Конструктор MainWindow(QWidget *parent = nullptr) инициализирует объект класса `MainWindow` и устанавливает родительское окно. В конструкторе также создается объект класса `PaintScene` и устанавливается в качестве виджета в главном окне.

Метод `ReadDataNetWork(std::string path)` читает информацию о нейронной сети из файла. Принимает путь к файлу в виде строки и возвращает объект структуры `data_NetWork`, содержащий информацию о нейронной сети.

Метод `ReadTest(double* input, int input_n)` читает текстовые данные. Принимает указатель на массив входных данных и их количество. Метод используется для чтения данных, которые будут поданы на вход сети нейронных элементов для распознавания.

Метод `StartGuess()` запускает процесс распознавания. Метод вызывается при нажатии на кнопку "Guess" в интерфейсе главного окна и запускает процесс распознавания с использованием нейронной сети.

Метод `clearLCD()` очищает виджет `PaintScene` и LCD-дисплей. Метод вызывается при нажатии на кнопку "Clean up".

В классе `MainWindow` также используется наследование от класса `QMainWindow` и содержится объект `ui`, который представляет собой интерфейс главного окна и позволяет управлять его элементами.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Разработка схем алгоритмов

`void Training(NetWork NW, const data_NetWork& NW_config)` — функция предназначена для обучения нейронной сети.

Алгоритм по шагам:

1. Начало.
2. Считываем массив пикселей и цифру в массив структур типа `data_info`.
3. Начинаем цикл, который прогоняет всю выборку цифр, повторим его 20 раз.
4. Обнуляем счетчик правильных ответов за один цикл.
5. Начинаем цикл, в котором рассматриваем каждый пример.
6. Загружаем в сеть пиксели и цифру.
7. С помощью метода `ForwarFeed()` предугадываем правильный ответ.
8. Если сеть не угадала, то перецти к пункту
9. Иначе к пункту 10
10. Применяем метод `BackPropogation()` и обновляем веса нейронной сети.
11. Увеличиваем счетчик правильных ответов.
12. Перейти к пункту 5.
13. Выводим информацию о результатах цикла в консоль.
14. Перейти к пункту 3.
15. Конец.

`void PaintScene::mouseReleaseEvent(QMouseEvent* pe)` - захватывает изображение сцены, масштабирует его до 28x28 пикселей и сохраняет его в файл в формате, который можно использовать для обучения нейронной сети распознаванию образов.

Алгоритм по шагам:

1. Устанавливаем переменную `draw` в значение `false`, чтобы остановить рисование.
2. Создаем объект `QPixmapдля` захвата изображения.
3. Захватываем изображение на вижете.
4. Создаем файл test.txt для преобразования изображения в массив пикселей.
5. Открываем файл test.txt.
6. Создаем объект `QTextStream` для записи данных в файл.
7. Преобразуем изображение в объект `QImage`.
8. Масштабируем изображение до размеров 28x28 пикселей.
9. Получаем ширину и высоту изображения.
10. Начинаем цикл, проходящий по всем столбцам из пикселей.
11. Начинаем цикл, проходящий по всем строкам из пикселей.

12. Если пиксель черный присваиваем 0, иначе 1.
13. Записываем в пиксель в файл.
14. Перейти к пункту 11.
15. Записываем символ новой строки
16. Перейти к пункту 10.
17. Закрываем файл.
18. Конец.

4.2 Разработка алгоритмов

Схема алгоритма метода `ForwarFeed()` приведена в приложении Б. – принимает входные данные и передает их через каждый слой сети, учитывая веса и смещения, чтобы получить выходные данные. Этот процесс называется прямым распространением и является основным шагом в работе нейронной сети. Он позволяет нейронной сети делать предсказания или классификацию входных данных и играет ключевую роль в ее обучении и функционировании.

Схема алгоритма метода `BackPropogation()` приведена в приложении В. – используется для обновления весов сети на основе ошибки, которая была выявлена в процессе прямого распространения. Этот метод вычисляет градиент функции потерь по отношению к весам и смещениям сети, чтобы определить, какие веса нужно изменить, чтобы уменьшить ошибку. Затем он применяет эти изменения к весам с помощью оптимизационного алгоритма, такого как градиентный спуск, чтобы улучшить производительность сети.

5 РЕЗУЛЬТАТЫ РАБОТЫ

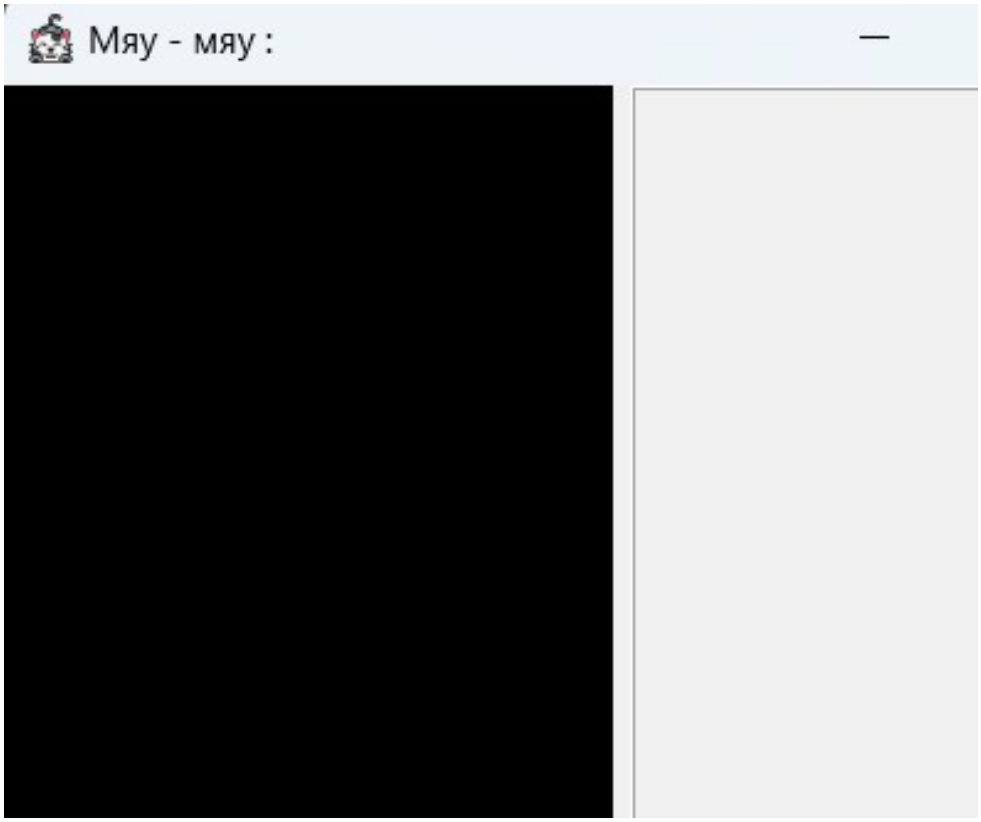

Рисунок 5.1 – Главное окно

На рисунке 5.1 представлено главное окно программы для тестирования нейронной сети. Слева виджет предоставляет возможность нарисовать цифру. Справа LCD-дисплей, который показывает ответ нейронной сети. Кнопка "Guess" запускает процесс распознавания цифры, нарисованной в правом окне. Кнопка "Clean up" очищает виджет и LCD-дисплей.

Рисунок 5.2 – Тестирование нейронной сети

На рисунке 5.2 представлен фрагмент работы программы, в котором нейронная сеть угадывает цифру “3”.

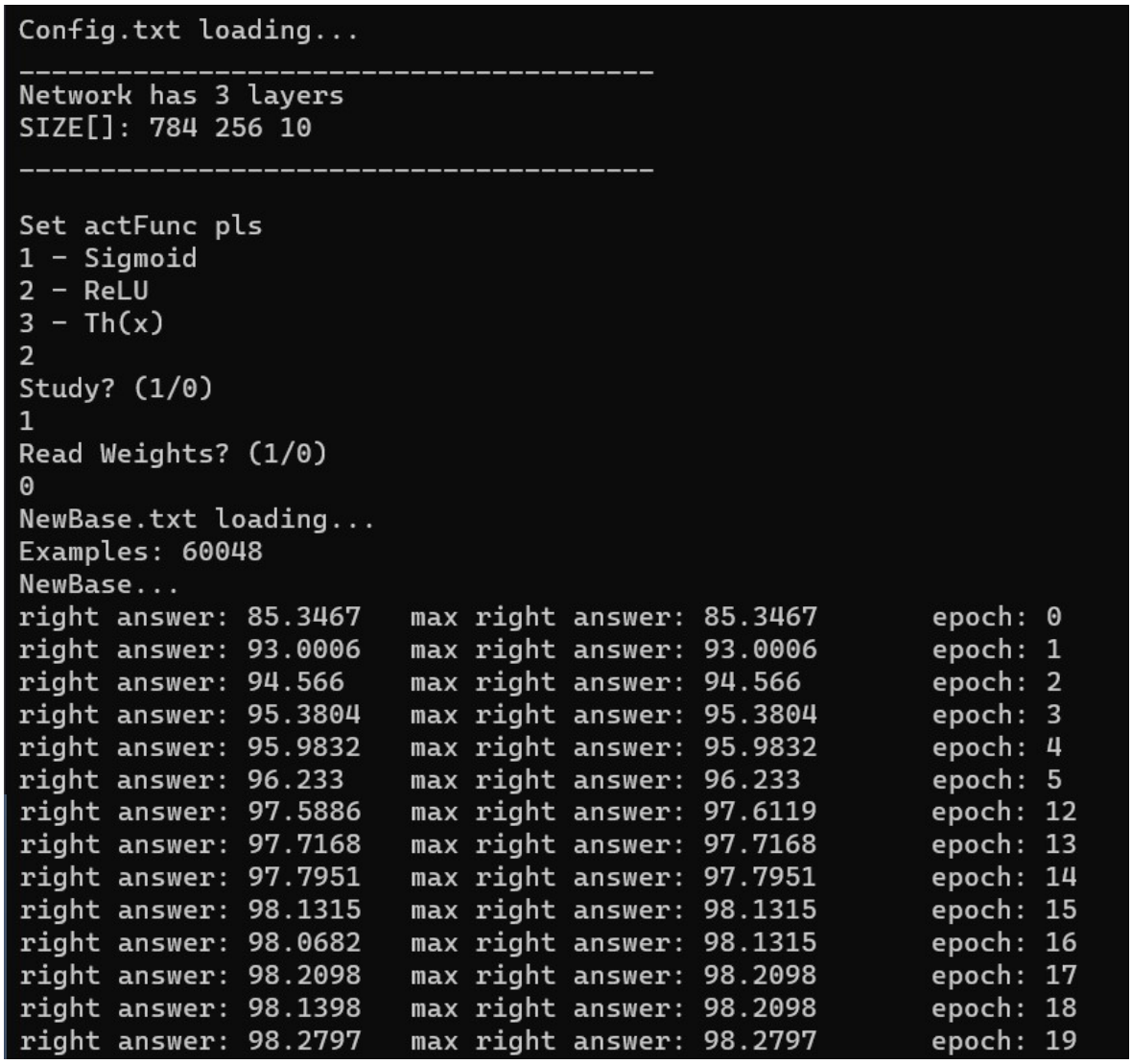

Рисунок 5.3 – Обучение нейронной сети на базе данных

На рисунке 5.3 представлен процесс обучении нейронной сети. нейронной сети. В качестве функции активации выбрана функции выбрана ReLU. Процесс обучения сопровождается статистикой обучения: коэффициент правильных ответов за эпоху, миксимальный коэффициент правильных ответов за все эпохи, номер эпохи, время затраченное на эпоху.

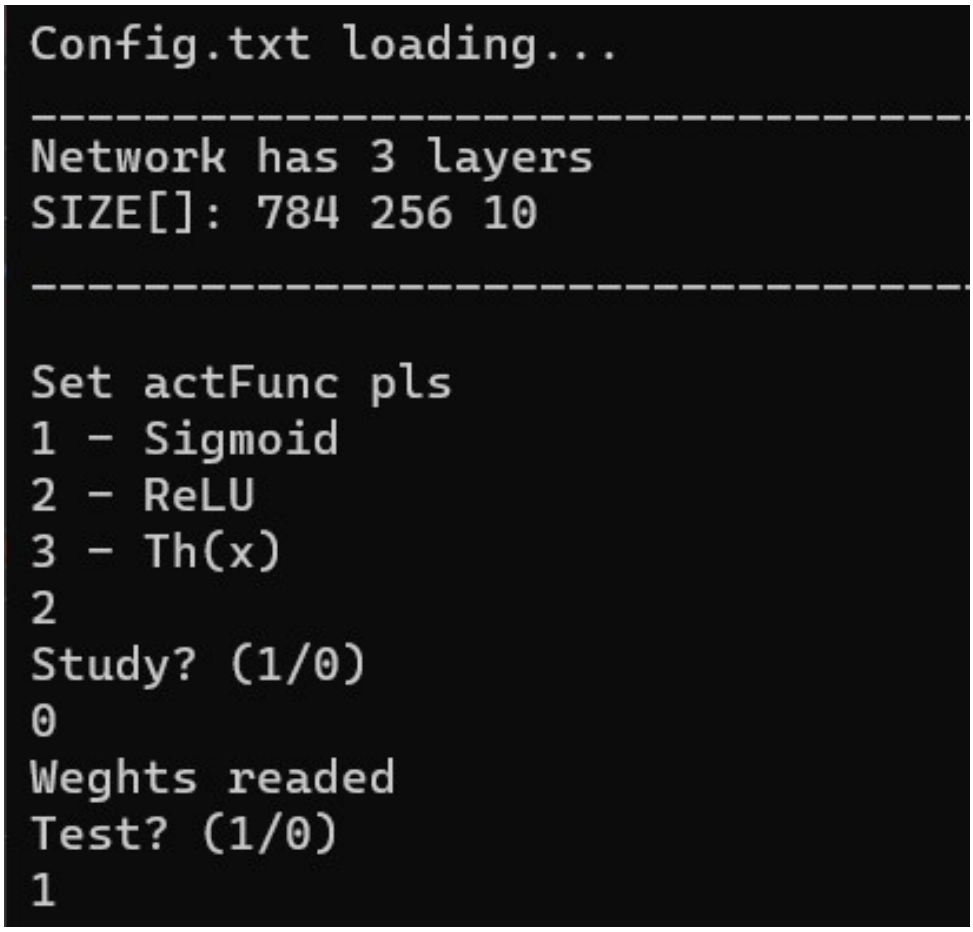

Рисунок 5.4– Тестирование нейронной сети на базе данных

На рисунке 5.4 представлен процесс тестирования нейронной сети. нейронной сети. В качестве функции активации выбрана функции выбрана ReLU. Были загружены веса из файла. И протестирована нейронная сеть. И выведена статистика по тестирования: коэффициент правильных ответов.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была создана и обучена нейронная сеть на базе набора данных MNIST. Для обучения сети были использованы методы прямого и обратного распространения ошибки, а веса нейронов были хранены в матрицах и файлах. Тип нейронной сети, который был выбран для этой работы – многослойный перцептрон.

Основная часть курсовой работы посвящена языку программирования C++, были усвоены основы объектно-ориентированного программирования, что позволило эффективно использовать возможности языка. Использование C++ позволило создать высокопроизводительную, эффективную нейронную сеть, обладающую широкими возможностями для работы с данными и обучения.

В результате обучения нейронная сеть, способная распознавать рукописные цифры из набора данных MNIST. Полученные результаты подтверждают эффективность методов обучения нейронных сетей на примере многослойного перцептрона. Графический интерфейс для тестирования нейронной сети был создан с использованием Qt.

Таким образом, данная работа не только позволила углубить знания в программирования, но и получить знания в области нейронных сетей. Полученные результаты могут быть использованы в различных областях, таких как компьютерное зрение, автоматизация процессов распознавания и классификации данных.

Для улучшения функционала нейронной сети. Её можно обучить не только на цифрах, но и на буквах. Также возможно расширить функционал нейронной сети для распознавания и классификации других типов данных, таких как изображения, звуковые сигналы или текст.

СПИСОК ЛИТЕРАТУРЫ

1. Объектно-ориентированное программирование на языке C++: учеб. пособие / Ю. А. Луцик, В. Н. Комличенко. – Минск : БГУИР, 2008.

2. Конструирование программ и языки программирования: метод. указания по курсовому проектированию для студ. спец. I-40 02 01 “Вычислительные машины, системы и сети” для всех форм обуч. / сост. А. В. Бушкевич, А. М. Ковальчук, И. В. Лукьянова. – Минск : БГУИР, 2009

3. Habr [Электронный ресурс]. -Электронные данные. -Режим доступа: https://habr.com/ru/companies/wunderfund/articles/314242/- Дата доступа: 27.11.2023

4. Wikipedia [Электронный ресурс]. -Электронные данные. -Режим доступа: https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%86%D0%B5%D0%BF%D1%82%D1%80%D0%BE%D0%BD - Дата доступа: 27.11.2023

5. Qt Documentation [Электронный ресурс]. -Электронные данные. -Режим доступа: https://doc.qt.io/ - Дата доступа: 27.11.2023

ПРИЛОЖЕНИЕ А
(обязательное)

(Диаграмма классов)

ПРИЛОЖЕНИЕ Б
(обязательное)

(Схема метода ForwarFeed())

ПРИЛОЖЕНИЕ В
(обязательное)

(Схема метода BackPropogation(double expert))

ПРИЛОЖЕНИЕ Г
(обязательное)

(Полный код программы)