ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Компьютерные науки и анализ данных»

Отчет о программном проекте

на тему: Нейросети с нуля

Выполнил:
Студент группы БКНАД221 _____ Артамонов Никита Николаевич
Подпись И.О.Фамилия

26.05.2024
Дата

Принял:
Руководитель проекта Дмитрий Витальевич Трушин
Имя, Отчество, Фамилия

доцент, к.ф.-м.н.
Должность, ученое звание

ФКН НИУ ВШЭ
Место работы (Компания или подразделение НИУ ВШЭ)

Дата проверки _____ 2024 _____ _____
Оценка (по 10-ти бальной шкале) Подпись

Москва 2024

Содержание

Аннотация **3**

1 Введение **4**
1.1 Описание предметной области . . . 4
1.2 Постановка задачи . . . 4
1.3 Результаты . . . 4

2 Обзор литературы **5**

3 Требования **6**
3.1 Функциональные требования . . . 6
3.2 Нефункциональные требования . . . 7

4 Математические основы программы **8**

5 Структура программы **10**
5.1 GlobalUsings.h . . . 10
5.2 ActivationFunction . . . 10
5.3 Оптимизаторы . . . 11
5.3.1 MiniBatchGD . . . 11
5.3.2 MomentumOptimizer . . . 11
5.3.3 AdamWOptimizer . . . 12
5.4 Layer . . . 12
5.5 LossFunction . . . 13
5.6 Network . . . 13
5.7 View . . . 14
5.8 Data . . . 14
5.9 Except . . . 14
5.10 Диаграмма потока данных . . . 15

6 Тестирование **16**
6.1 Тестирование на MNIST . . . 16
6.2 Тестирование на MNIST-Fashion . . . 17

Список литературы **18**

Аннотация

В данной работе описывается имплементация всех необходимых компонентов для работы и обучения нейросети с произвольным количеством слоев. Обучения нейросети демонстрируется на примере коллекций mnist [14] и mnsit-fashion [22]. Обучение нейросети даёт возможность решать задачи регрессии и классификации. В работе рассматривается имплементация нейросети с различными функциями активации, различными функциями потерь и оптимизаторами.

Ключевые слова

Нейросеть, слой нейросети, скрытый слой нейросети, матрица весов, вектор сдвига, функция активации, функция потерь, градиентный спуск, mini-batch, оптимизатор, mnist, mnist-fashion

1 Введение

1.1 Описание предметной области

Пусть дано неизвестное нам отображение $\varphi : \mathbb{R}^n \to \mathbb{R}^m$, где $n,\ m \in \mathbb{N}$. Нейросети используется для предсказания образа отображения по входному вектору. Для этого предварительного проводится обучение нейросети, при котором используется обучающая выборка - уже известные пары праобраз-образ отображения φ. Задача обучения заключается в подборе таких параметров, при использовании которых, воздействующее на входной вектор отображение выдаёт значение, которое достаточно хорошо приближает истинное значение. Ключевая роль в подборе параметров отводится методу градиентного спуска.

1.2 Постановка задачи

Целью данной работы является изучение теории нейронных сетей и имплементация необходимых структур для работы и обучения нейросетей, что позволит решать задачи регрессии и классификации.

Поставлены следующие **задачи**:

- Изучить структуры нейронных сетей. Данный пункт включает формирование понимания устройства отдельно взятого слоя нейронной сети и его компонент, включая параметры, входной и выходной векторы, функции активации, а также функции потерь.
- Изучить метод градиентного спуска и его вариации, такие как:
  - Градиентный спуск по батчу (англ. Batch gradient descent)
  - Стохастический градиентный спуск (англ. Stochastic gradient descent)
  - Градиентный спуск по мини-батчу (англ. Mini-batch gradient descent)
- Изучить алгоритмы оптимизации градиентного спуска
- Имплементировать необходимые структуры для работы и обучения нейросети:
  - Слой нейросети - структура, хранящая обучаемые параметры, выполняющая линейное и нелинейное преобразование для прохода вперёд, а также методы для осуществления градиентного спуска на данном слое
  - Класс Network, связующий слои воедино. Данная компонента отвечает за организацию обучения и тестирования нейросети в целом
  - Функции активации
  - Функции потерь
  - Методы оптимизации
- Обучить нейросеть для задачи классификация по распознаванию чисел по базе данных mnist и предметов одежды по базе данных mnist-fashion.

1.3 Результаты

По итогам выполнения работы получены следующие **результаты**:

- Изучена структура нейросетей
- Изучены разновидности градиентного спуска
- Ипмлементированы классы для работы и обучения нейросети, поддерживающие произвольное количество скрытых слоев, различные оптимизационные методы, а также возможность использования пользовательских функций активации и функций потерь помимо уже встроенных
- Имплементирован алгоритм градиентного спуска по мини-батчу
- Имплементирован метод оптимизации Momentum (в том числе NAG - Nesterov Accelerated Gradient)
- Имплементирован метод оптимизации AdamW

- Имплементированы различные функции активации и потерь для прохода вперед и их производные для обратного распространения ошибки
- Имплементированы методы, позволяющие создавать нейросеть на основе информации из файла
- Имплементированы методы, позволяющие сохранять обученную модель (параметры, состояние оптимизатора) в файл
- Обучена нейросеть для распознавания чисел по базе данных mnist с уровнем точности предсказания более 96% по прошествии 5-и эпох
- Обучена нейросеть для распознавания предметов одежды по базе данных mnist-fashion с уровнем точности более 85% по прошествии 5-и эпох

Имплементация программы доступна в репозитории на GitHub

2 Обзор литературы

Были изучены материалы, представленные в книге [17]. В данном источнике представлена основная информация о принципах устройства и работы нейронных сетей, объяснен вывод ряда формул, используемых при градиентном спуске.

Также присутствуют информация о выборе функций активации и функции потерь, гиперпараметров. Предложены некоторые оптимизации и методы борьбы с переобучением, такие как, например, weight decay.

Представленная в книге примерная имплементация нейросети на языке программирования Python, однако, не применяет матрично-векторное дифференцирование для градиентного спуска, которое используется в данной работе, не позволяет использовать пользовательские функции активации и функции потерь.

Рассмотрены различные варианты градиентного спуска и алгоритмы его оптимизации, представленные в обзорной статье [19]. Данная статья содержит описание сути рассматриваемых методов и преимущества их использования.

3 Требования

3.1 Функциональные требования

Программа должны иметь следующие компоненты:

- **Класс Network** - позволяет создать нейросеть с заданным количеством слоев и их размерностями, установить функцию активации для каждого слоя, задать оптимизатор. Класс предоставляет пользователю методы для обучения, тестирования и использования нейросети. Также предусматривается возможность сохранения нейросети в файл и создание нейросети по информации из файла
- **Класс View** - вспомогательный класс, предоставляющий возможность перемешивать данные во время обучения
- **Класс LossFunction** - отвечает за применение функции потерь и подсчета ее производной. Также класс предоставляет пользователю ряд встроенных функций потерь: **MSE, CrossEntropy**
- **Клас Layer** задает полносвязный слой нейросети. Содержит методы, необходимые для осуществления прохода вперед (англ. Forward Pass) на данном слое, расчета градиента во время обратного распространения ошибки. Делегирует обновление параметров оптимизатору.
- **Класс ActivationFunction** - отвечает за применение функции активации и подсчета ее производной. Класс предоставляет пользователю следующие функции потерь: **Sigmoid, ReLu, LeakyReLu, Tanh, SoftMax**
- **Оптимизаторы** - классы, определяющие механизм обновления параметров по ходу градиентного спуска. Также предоставляют возможность сохранить состояние оптимизатора в файл и прочитать из него.

  Пользователю доступные следующие классы оптимизаторов:

  - **MiniBatchGD** - градиентный спуск по мини-батчу без дополнительных оптимизаций
  - **MomentumOptimizer** - предоставляет методы оптимизации Momentum и NAG
  - **AdamWOptimizer** - предоставляет метод оптимизации AdamW
- **Пространство имен Data** - предоставляет пользователю методы для получения готовых для обработки нейросетью данных коллекций mnist и mnist-fashion. Также доступны методы для приведения входных векторов и целевых к формату, принимаемому нейросетью
- **Пространство имен Except** - отвечает за перехват исключений и выведения соответствующего сообщения в стандартный поток ошибок
- **Заголовочный файл GlobalUsings.h** - задает псевдонимы, используемые другими компонентами программы

3.2 Нефункциональные требования

Для корректной работы к соблюдению необходимы следующие требования:

- **Язык программирования:** C++17 [1]
- **Компилятор:** GCC версии начиная с 11 [7], Clang версии начиная с 11 [4]
- **Система сборки:** CMake версии начиная с 3.22 [5], Ninja версии начиная с 1.11 [18]
- **Система поддержки версий:** git версии начиная с 2.32 [8], GitHub [9]
- **Сторонние библиотеки:**
  - **Eigen** версии 3.4.0 [12], подключен как git submodule
  - **EigenRand** версии 0.5.0 [6], подключен как git submodule
  - **GoogleTest** версии 1.14.0 [11], подключен как git submodule
  - **mnist reader** [20], подключен как git submodule
- **Стиль кода:** Google C++ Style Guide [10], поддержка осуществляется через .clang-format
- **Инструмент форматирования кода:** ClangFormat [2], осуществляет форматирование в соответствии с заданным стилем кода, конфигурационный файл - .clang-format
- **Инструмент статичного анализа кода:** Clang-Tidy [3], конфигурационный файл - .clang-tidy
- **Обработка ошибок:**
  - Макросы **assert** используются для обработки ошибок, возникших по вине разработчика в контексте имплементации программы
  - Исключение **std::invalid_argument** генерируется в случае, если в конструктор класса Layer была передана невалидная функция активации

4 Математические основы программы

Пуст имеем неизвестное отображения $\boldsymbol{T} : \mathbb{R}^{\boldsymbol{n}} \to \mathbb{R}^{\boldsymbol{m}}$, которое приблизим параметризованным отображением $\boldsymbol{F}(\boldsymbol{\theta}) : \mathbb{R}^{\boldsymbol{n}} \to \mathbb{R}^{\boldsymbol{m}}$ подбором параметров $\boldsymbol{\theta}$. Отображение $\boldsymbol{F}(\boldsymbol{\theta})$ будет являться **полносвязной нейросетью**.

Введем ряд определений:

Определение 1. Набор $\boldsymbol{X} = (\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_n})$ назовем **входными данными**

Определение 2. Набор $\boldsymbol{Y} = (\boldsymbol{y_1}, \boldsymbol{y_2}, \cdots, \boldsymbol{y_n})$ назовем **истинными значениями** отображения $\boldsymbol{T}$ по аргументам $\boldsymbol{X} = (\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_n})$ соответственно

Определение 3. Набор $\hat{\boldsymbol{Y}} = (\hat{\boldsymbol{y}}_1, \hat{\boldsymbol{y}}_2, \cdots, \hat{\boldsymbol{y}}_n)$, где $\hat{\boldsymbol{Y}} = \boldsymbol{F}(\boldsymbol{X}, \boldsymbol{\theta})$, назовем **предсказаниями** нейросети для соответствующих входных данных $\boldsymbol{X} = (\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_n})$

Нейросеть состоит из некоторого произвольного количества слоев. Первый слой нейросети принимает вектор входных данных. Иные слои принимают на вход вектор, полученный в результате обработки предыдущим слоем.

Каждый слой представляет собой состоящее из двух этапов отображение вида $\mathbb{R}^{\boldsymbol{k}} \to \mathbb{R}^{\boldsymbol{l}}$, где $\boldsymbol{k}$ - размерность входного вектора, а $\boldsymbol{l}$ - результирующего. Этапы:

1. **Линейная компонента**

   Каждый слой выполняет над полученным вектором $\boldsymbol{x}$ линейное преобразование вида:

   $$\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{1}$$

   где $\boldsymbol{W} \in \boldsymbol{Mat}_{\boldsymbol{l} \times \boldsymbol{k}}$ - матрица весов, $\boldsymbol{b} \in \mathbb{R}^{\boldsymbol{l}}$ - вектор сдвига, которые первоначально инициализируются случайным образом

2. **Нелинейная функция активации $\boldsymbol{\sigma} : \mathbb{R}^{\boldsymbol{l}} \to \mathbb{R}^{\boldsymbol{l}}$**

   Функция $\boldsymbol{\sigma}$ применяется к вектору $\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ поэлементно

Таким образом, один слой нейросети можно описать следующей последовательностью преобразований:

$$\boldsymbol{x} \to \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \to \boldsymbol{\sigma}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{2}$$

Введем еще несколько определений:

Определение 4. Функция потерь $\boldsymbol{\psi} : \mathbb{R}^{\boldsymbol{m}} \times \mathbb{R}^{\boldsymbol{m}} \to \mathbb{R}$ - функция, отражающая отличие истинных значений отображения $\boldsymbol{T}$ от значений, предсказанных нейросетью

Определение 5. Функция ошибки:

$$\boldsymbol{\mathcal{L}}(\boldsymbol{\theta}) = \frac{\boldsymbol{1}}{\boldsymbol{N}} \sum_{\boldsymbol{i}=\boldsymbol{1}}^{\boldsymbol{N}} \boldsymbol{\psi}(\boldsymbol{y_i}, \hat{\boldsymbol{y}}_{\boldsymbol{i}}) \tag{3}$$

функция, отражающее усредненное по всем входным данным отклонение истинных значений отображения $\boldsymbol{T}$ от предсказанных

Ключевой задачей обучения нейросети является минимизация функции $\boldsymbol{\mathcal{L}}(\boldsymbol{\theta})$ по параметрам $\boldsymbol{\theta}$. Для данной цели используется метод **градиентного спуска**

Определение 6. Пусть $\boldsymbol{f} : \mathbb{R}^{\boldsymbol{n}} \to \mathbb{R}$ - некоторая функция. Вектор $\boldsymbol{\nabla f} = (\frac{\boldsymbol{\partial f}}{\boldsymbol{\partial x_1}}, \frac{\boldsymbol{\partial f}}{\boldsymbol{\partial x_2}}, \cdots, \frac{\boldsymbol{\partial f}}{\boldsymbol{\partial x_n}})$ называет **градиентом**. Вектор $\boldsymbol{-\nabla f}$ **антиградиентом**

Антиградиент показывает направление наискорейшего убывания функции. С целью минимизации функции $\boldsymbol{\mathcal{L}}(\boldsymbol{\theta})$ для i-го слоя будем обновлять параметры следующим образом:

$$\boldsymbol{W}'^{(\boldsymbol{i})} = \boldsymbol{W}^{(\boldsymbol{i})} - \boldsymbol{\alpha} \frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial W}^{(\boldsymbol{i})}}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}, \boldsymbol{\theta}) \tag{4}$$

$$\boldsymbol{b}'^{(\boldsymbol{i})} = \boldsymbol{b}^{(\boldsymbol{i})} - \boldsymbol{\alpha} \frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial b}^{(\boldsymbol{i})}}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}, \boldsymbol{\theta}) \tag{5}$$

где $\boldsymbol{W}^{(\boldsymbol{i})}$ **и** $\boldsymbol{b}^{(\boldsymbol{i})}$ **- параметры i-го слоя,** $\boldsymbol{\alpha}$ **- коэффициент скорости обучения**

Для вычисления необходимых поправок используем **метод обратного распространения ошибки**:

Пусть даны: $\boldsymbol{x_i}$ - вектор входных данных, $\hat{\boldsymbol{y}}_{\boldsymbol{i}} = \boldsymbol{F}(\boldsymbol{x_i})$ - предсказание нейросети, $\boldsymbol{y_i}$ - истинное значение, и нейросеть имеет $\boldsymbol{l}$ слоев. Необходимо выполнить следующие шаги:

1. **Инициализация**

   На l-м (последнем) слое нейросети вычислим вектор-строку $u_i^{(l)}$ по формуле:

$$u_i^{(l)} = \frac{\partial \psi}{\partial y}(y_i, \hat{y}_i) \tag{6}$$

2. **Вычисление поправок для параметров**

   Для вычисления необходимых поправок для матрицы весов и вектора сдвига используются формулы:

$$\frac{\partial \psi}{\partial W^{(l)}} = \sigma'(W^{(l)} x_i^{(l)} + b^{(l)})(u_i^{(l)})^T (x_i^{(l)})^T \tag{7}$$

$$\frac{\partial \psi}{\partial b^{(l)}} = \sigma'(W^{(l)} x_i^{(l)} + b^{(l)})(u_i^{(l)})^T \tag{8}$$

3. **Вычисление вектора, распространяющего ошибку на следующий слой**

   Вектор $u_i^{(l-1)}$, передаваемый на $(l-1)$-й слой вычисляется по формуле:

$$u_i^{(l-1)} = u_i^{(l)} \sigma'(W^{(l)} x_i^{(l)} + b^{(l)}) W^{(l)} \tag{9}$$

   Вектор $u_i^{(l-1)}$ выступает в качестве инициализирующего для $(l-1)$-го слой

4. **Повтор вышеперечисленных шагов для каждого слоя нейросети**

Подробнее с описанными выше методами можно ознакомиться в источнике [17]. Вывод формул 1-9 приведен в [13].

В данной работе имплементирован **метод градиентного спуска по мини-батчу**

Определение 7. Пусть $X = (x_1, x_2, \cdots, x_n)$ - входные данные. Тогда подмножество $B_i = (x_j, x_{j+1}, \cdots, x_{j+k})$, такое что $X = \cup_{i=1}^{N} B_i$, где $B_i \cap B_j = \emptyset, \forall i \neq j$ назовем **мини-батчом** размера k, где N - количество мини-батчей

Использование градиентного спуска по мини-батчу способно обеспечить более стабильную сходимость [19]. При его использовании формулы 4-5 модифицируются следующим образом:

$$W'^{(i)} = W^{(i)} - \frac{\alpha}{N} \sum_{i=1}^{N} \frac{\partial \psi}{\partial W^{(i)}}(y, \hat{y}, \theta) \tag{10}$$

$$b'^{(i)} = b^{(i)} - \frac{\alpha}{N} \sum_{k=1}^{N} \frac{\partial \psi}{\partial b^{(i)}}(y, \hat{y}, \theta) \tag{11}$$

В случае комбинации функции активации SoftMax на последнем слое и функции CrossEntropy в качестве функции потерь, множитель $z_i^{(l)} = \sigma'(W^{(l)} x_i^{(l)} + b^{(l)})(u_i^{(l)})^T$ возможно вычислить [16] по формуле:

$$\frac{\partial \psi}{\partial z_i^{(l)}} = \hat{y}_i - y_i \tag{12}$$

Также в работе имплементированы методы оптимизации AdamW [15], Momentum и NAG [21]. Обзорная информация о данных методах доступна в источнике [19].

5 Структура программы

Ниже представлена краткая диаграмма имплементированных классов пространств имен:

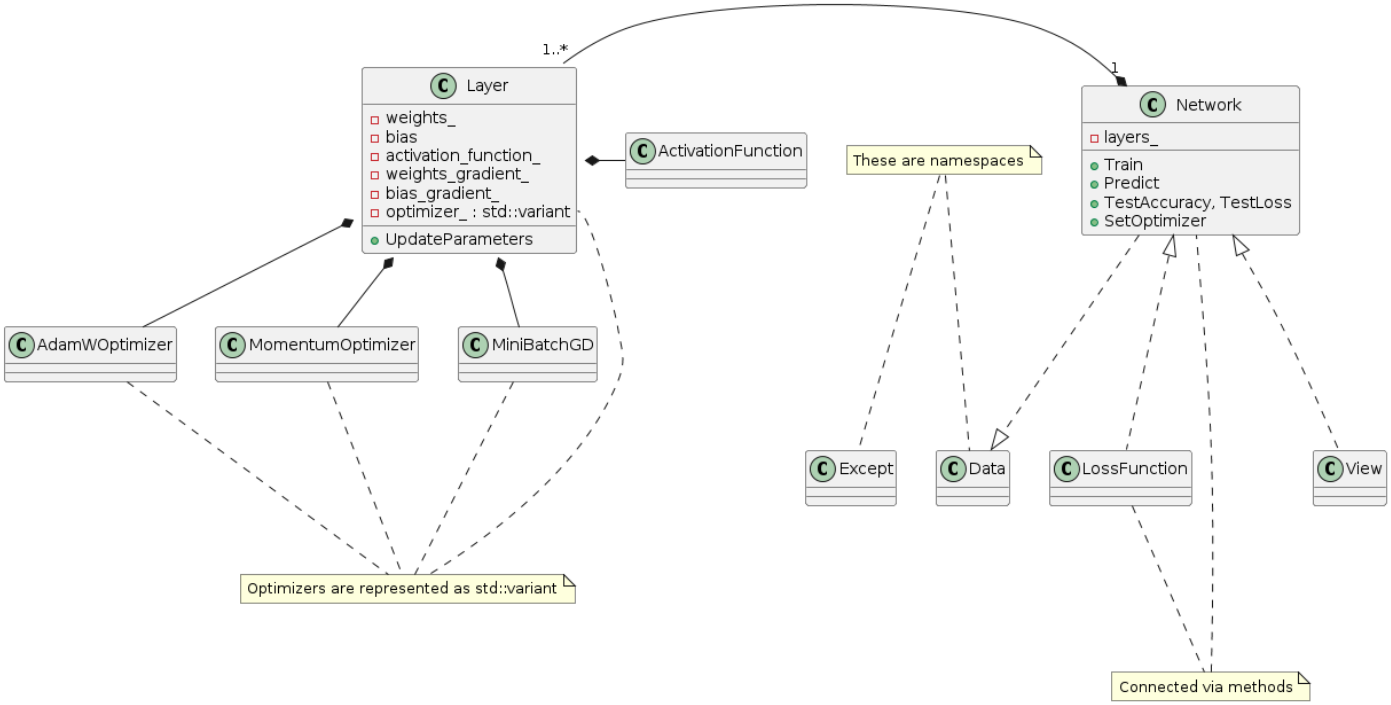

Рис. 1: Диаграмма классов

Далее приводится более подробное описание публичного интерфейса каждой компоненты:

5.1 GlobalUsings.h

В данном заголовочном файле определены псевдонимы для типов, часто используемых различными компонентами программы:

- `using Vector = Eigen::VectorXd;`
- `using RowVector = Eigen::RowVectorXd;`
- `using Matrix = Eigen::MatrixXd;`
- `using Index = Eigen::Index;`
- `using Vectors = std::vector<Vector>;`
- `struct Parameters` - содержит два поля - `Matrix`, `Vector`

5.2 ActivationFunction

Класс обеспечивает работу с функциями активации. В заголовочном файле ActivationFunction.h также содержится **enum class ActivationFunctionType**, содержащий перечь встроенных функций активации.

Конструкторы класса:

- `ActivationFunction() = default` - конструктор по умолчанию
- `ActivationFunction(const VecToVecFunc&, const VecToMatFunc&, ActivationFunctionType)` - конструктор, принимающий объекты `std::function`: для функции активации (принимает вектор, возвращает вектор), для производной функции активации (принимает вектор, возвращает матрицу)

- `ActivationFunction(VecToVecFunc&&, VecToMatFunc&&, ActivationFunctionType` - перегруз конструктора выше

Класс предоставляет следующие публичные **методы**:

- `Vector Activate(const Vector&) const` - применяет функцию активации к переданному вектору
- `Matrix ComputeJacobianMatrix(const Vector&) const` - вычисляет матрицу Якоби по переданному вектору
- `explicit operator bool() const` - проверяет, является ли экземпляр класса пустым
- `std::string GetType() const` - возвращает название функции активации в виде `std::string`

Также класс содержит ряд статичных методов, возвращающих готовые объекты `ActivatiobFunction` - встроенные функции активации, а именно: `Sigmoid, ReLu, LeakyReLu, Tanh, Softmax`.

Кроме того, пользователю имеет возможность создавать собственные функции активации

Статичный метод `ActivationFunction GetFunction(const std::string&)` предоставляет опцию получить функцию активации по ее названию. Данный метод используется в случае создания нейросети по информации из файла

5.3 Оптимизаторы

В работе имплементированы следующие классы оптимизаторов: `MiniBatchGD`, `MomentumOptimizer`, `AdamWOptimizer`

5.3.1 MiniBatchGD

Класс позволяет осуществить метод градиентного спуска по мини-батчу

Конструкторы класса:

- `MiniBatchGD() = default` - конструктор по умолчанию
- `MiniBatchGD(double, double)` - конструктор от параметров коэффициент скорости обучения и коэффициент сокращения весов
- `explicit MiniBatchGD(std::istream&)` - конструктор, принимающий входной поток - позволяет создать оптимизатор по информации, прочитанной из файла

Методы класса:

- `Parameters UpdateParameters(const Matrix&, const Vector&, const Matrix&, const Vector&, int) const` - высчитывает и возвращает обновленные параметры обучения после выполнения шага обучения (происходит по завер- шению обработку одного мини-батча) в соответствии с механизмом данного метода оптимизации
- `friend std::ostream& operator<<(std::ostream&, const MiniBatchGD&)` - передает состояние оптимизатора в поток - обеспечивает возможность сохранить состояние в файл, чтобы позже иметь возможность продолжить обучение

5.3.2 MomentumOptimizer

Класс предоставляет возможность использования метода оптимизации **Momentum** [21]. Также доступен **enum class Nesterov**, позволяющий задать метод оптимизации NAG.

Конструкторы класса:

- `MomentumOptimizer() = default` - конструктор по умолчанию
- `MomentumOptimizer(double, double, double, Nesterov = Nesterov::Disable)` - конструктор от параметров оптимизатора, а также сущности `enum class Nesterov`, позволяющий использовать метод NAG (по умолчанию отключен)
- `explicit MomentumOptimizer(std::istream&)` - конструктор, принимающий входной поток

Методы класса:

- `void Initialize(Index, Index)` - инициализирует оптимизатор в соответствии с размерностями слоя, за обновление параметров которого он отвечает
- `Parameters UpdateParameters(const Matrix&, const Vector&, const Matrix&, const Vector&, int) const` - высчитывает и возвращает обновленные параметры обучения после выполнения шага обучения в соответствии с механизмом данного метода оптимизации
- `friend std::ostream& operator<<(std::ostream&, const MomentumOptimizer&)` - передает состояние оптимизатора в поток

5.3.3 AdamWOptimizer

Класс предоставляет возможность использования метода оптимизации **AdamW** [15].

Конструкторы класса:

- `AdamWOptimizer() = default` - конструктор по умолчанию
- `AdamWOptimizer(double, double, double, double, double)` - конструктор от параметров оптимизатора
- `explicit AdamWOptimizer(std::istream&)` - конструктор, принимающий входной поток

Методы класса:

- `void Initialize(Index, Index)` - инициализирует оптимизатор в соответствии с размерностями слоя, за обновление параметров которого он отвечает
- `Parameters UpdateParameters(const Matrix&, const Vector&, const Matrix&, const Vector&, int) const` - высчитывает и возвращает обновленные параметры обучения после выполнения шага обучения в соответствии с механизмом данного метода оптимизации
- `friend std::ostream& operator<<(std::ostream&, const AdamWOptimizer&)` - передает состояние оптимизатора в поток

5.4 Layer

Класс, позволяющий создание полносвязного слоя нейросети заданной размерности. Параметры матрица весов и вектор сдвига инициализируется случайными числами из нормального стандартного распределения.

Конструкторы класса:

- `Layer() = default` - конструктор по умолчанию
- `Layer(Index, Index, const ActivationFunction&)` - создает слой с переданными размерностями и соответствующей функцией активации
- `Layer(Index, Index, ActivationFunction&&)` - перегруз конструктора выше
- `explicit Layer(std::istream&, const ActivationFunction& = ActivationFunction())` - создает слой на основе информации, переданный через поток, в случае необходимости передается также функция активации (например, если пользователь имеет собственную функцию)

Методы класса:

- `Vector PushForward(const Vector&)` - принимает входной вектор, преобразует его согласно формуле 2
- `RowVector PropagateBack(const RowVector&)` - во время обратного распространения ошибки высчитывает по вектору с предыдущего слоя необходимые поправки для матрицы весов и вектор сдвига и возвращает вектор градиента для передачи на следующий слой (см. раздел 4, формулы 7-9)
- `RowVector PropagateBackSoftMaxCE(const Vector&)` - производит на последнем слое те же действия, что и `PropagateBack`, но для случая комбинации функции активации `SoftMax` на последнем слое и `CrossEntropy` в качестве функции потерь

- `void UpdateParameters(int)` - принимает размер мини-батча, делегирует расчет обновленных параметров оптимизатору, обновляет параметры и обнуляет градиенты (см. раздел 4, формулы 10-11)
- `template<typename OptimizerType> void SetOptimizer(const OptimizerType& optimizer)` - устанавливает оптимизатор (любой из доступных за исключением `MiniBatch`
- `void SetOptimizer(const MiniBatchGD&)` - вызывается для установки оптимизатора `MiniBatch`. Перегруз необходим, т.к. для данного оптимизатора не требуется инициализация
- `friend std::ostream& operator<<(std::ostream&, const Layer&)` - передает в поток состояние слоя

5.5 LossFunction

Класс обеспечивает работу с функциями потерь.

Конструкторы класса:

- `LossFunction(const VecVecToDoubleFunc&, const VecVecToRowVecFunc&)` - конструктор, принимающий объекты `std::function`: для функции потерь (принимает два вектора, возвращает `double`), для производной функции потерь (принимает два вектора, возвращает вектор-строку)
- `LossFunction(VecVecToDoubleFunc&&, VecVecToRowVecFunc&&)` - перегруз для конструктора выше

Методы класса:

- `double ComputeLoss(const Vector&, const Vector&) const` - высчитывает ошибку по вектору истинных значений и вектору, полученному в качестве предсказания нейросети
- `RowVector ComputeInitialGradient(const Vector&, const Vector&) const` - высчитывает первоначальный градиент (формула 6)
- `explicit operator bool() const` - проверяет, является ли экземпляр класса пустым

Также пользователю предоставляется ряд статичны методом, возвращающих готовый функции потерь, а именно: `MSE, CrossEntropy`

Кроме того, пользователь имеет возможность создавать собственные функции потерь.

5.6 Network

Класс позволяет создать полносвязную нейросеть с заданным количеством слоев и размерностями. Также заголовочный файл Network.h дает доступ к `enum class EarlyStopping`, что дает возможность задать раннюю остановку обучения, `enum class Task` - установить задачу, `struct ClassificationMetrics` - получать в форме структуры метрики в случае задачи классификации

Конструкторы класса:

- `Network() = default` - конструктор по умолчанию
- `Network(std::initializer_list<Index>, std::initializer_list<ActivationFunction>)` - создает нейросеть согласно переданным размерностям и функциям активации
- `explicit Network(std::istream&)` - создает нейросеть по данным, полученным из потока
- `Network(std::istream&, std::initializer_list<ActivationFunction>)` - создает нейросеть по данным, полученным из потока, при этом функции активации задает согласно переданным в списке инициализации (используется в случае пользовательских функций активации)

Методы класса:

- `void Train(const Vectors&, const Vectors&, const Vectors&, const Vectors&, int, int, const LossFunction&, Task = Task::Unspecified, EarlyStopping = EarlyStopping::Disable, double = 0.0)` - метод, производящий обучения нейросети и валидацию. Принимает необходимые данные, количество эпох, размер мини-батча. В случае необходимости, существует возможность задать раннюю остановку обучения (по умолчанию опция отключена)

- `void Train(const Vectors&, const Vectors&, int, int, const LossFunction&, Task = Task::Unspecified)` - перегруз методы выше для случая отсутствия валидационных данных
- `Vector Predict(const Vector&)` - высчитывает предсказания нейросети
- `double TestLoss(const Vectors&, const Vectors&, const LossFunction&)` - тестирует нейросеть на тестовых данных, возвращая среднюю ошибку
- `ClassificationMetrics TestAccuracy(const Vectors&, const Vectors&, const LossFunction&)` - тестирует нейросеть на тестовых данных, возвращая среднюю ошибку и количество верно сделанных предсказаний (актуально для задачи классификации)
- `friend std::ostream& operator<<(std::ostream&, const Network&)` - передает состояние нейросети в поток

5.7 View

Вспомогательный класс, обеспечивающие перемешивание данных во время обучения

Доступны следующие статичные **методы**:

- `std::vector<int> GenerateViewVector(int)` - по заданному размеру данных генерирует `std::vector` индексов, по которым будет происходить обращения к данным
- `void ShuffleViewVector(std::vector<int>&)` - производит перемешивания переданного `std::vector`

5.8 Data

Пространство имен, позволяющее получить данные коллекций mnist и mnist-fashion в формате, принимаемом нейросетью

Заголовочный файл Data.h открывает также открывает доступ к `enum class MnistType`, с помощью которого задается необходимая коллекция данных и к `struct Dataset` - тип, который будет содержать набор данных - для обучения, валидации и тестирования.

Методы:

- `Dataset GetMnistData(MnistType, Index, Index, Index)` - возвращает тренировочные, валидационные и тестовые данные коллекций mnist или mnist-fashion заданного размера и типа
- `Vector GenerateOneHotVector(unsigned char, Index)` - на основании истинного значения и количества возможных классов генерирует вектор, который по индексу истинного значение имеет значение 1, по остальным - 0
- `Vectors GenerateTargets(const std::vector<unsigned char>&, Index)` на основании переданных истинных значений создает контейнер std::vector, состоящий из полученных методом `GenerateOneHotVector` векторов
- `Vectors GenerateInputVectors(const std::vector<std::vector<unsigned char>>&)` - приводит входные данные к принимаемому нейросетью формату

5.9 Except

Пространство имен, отвечающее за перехват возникшего исключения и выведение соответствующей информации в стандартный поток ошибок. Содержит один метод - `void React()`.

5.10 Диаграмма потока данных

Ниже представлена краткая диаграмма потока данных во время обучения нейросети

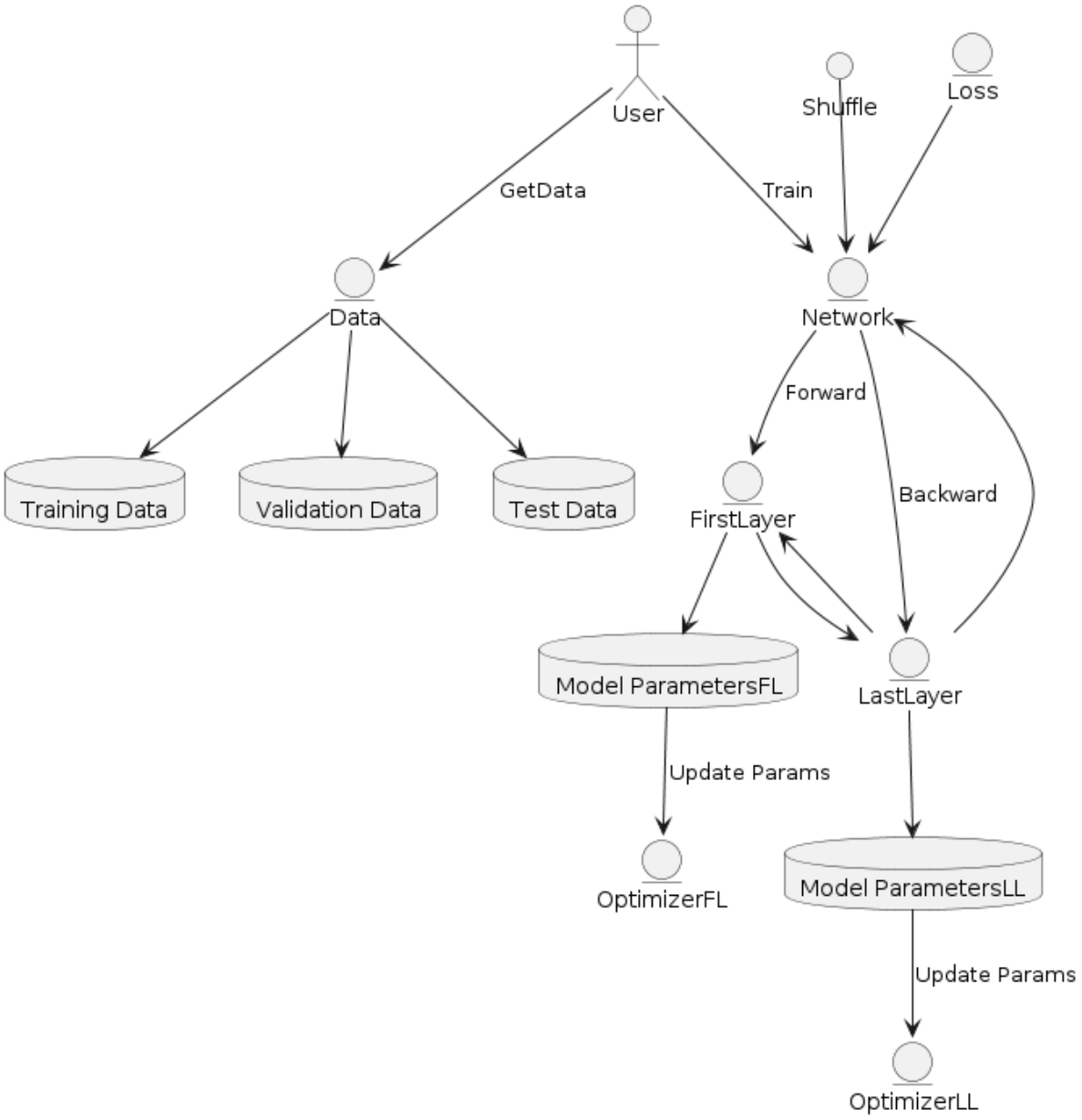

Рис. 2: Диаграмма потока данных

Пользователь получает готовые к обучению данные с помощью методов пространства имен `Data`. Затем передает их классу `Network` с помощью метода `Train`. Далее данные передаются первому слою нейросети и начинается проталкивание вперед (англ. Forward Pass) до достижения последнего слоя. Затем полученный результат (предсказание) возвращается в класс `Network`, где с помощью методов класса `LossFunction` высчитывается первоначальный градиент и далее запускается обратное распространение ошибки. По завершению одного мини-батча класс `Layer` делегирует расчет обновленных параметров оптимизатору.

6 Тестирование

Для тестирования корректности работы отдельных модулей программы были имплементированы юнит-тесты (англ. unit-tests) при помощи фреймворка `GoogleTest` [11]

Также с целью демонстрации корректности обучения и возможностей программы было проведено обучение нейросети на коллекциях данных mnist для решения задачи распознавания рукописных чисел и mnist-fashion для распознавания предметов одежды

6.1 Тестирование на MNIST

Для обучения нейросети по коллекции mnist была использована следующая архитектура:

```
    int batch_size = 4;
    double learning_rate = 0.01;
    double weights_decay = 0.0001;
    double beta1 = 0.9;
    double beta2 = 0.999;
    double epsilon = 1e-8;
    int epochs = 5;

    auto network =
      Network({784, 100, 25, 10},
              {ActivationFunction::LeakyReLu(), ActivationFunction::LeakyReLu(),
               ActivationFunction::SoftMax()});

    network.SetOptimizer(
      AdamWOptimizer(learning_rate, weights_decay, beta1, beta2, epsilon));

    network.Train(train_inputs, train_targets, batch_size, epochs,
                  LossFunction::CrossEntropyLoss(),
                  Task::SoftMaxCEClassification);
```

Результаты из стандартного потока вывода по прошествии 5 эпох представлены ниже:

```
/Users/nikitaartamonov/CLionProjects/neural-network/cmake-build-release/neural-network
Task: Handwritten digits recognition
Dataset: MNIST

5 epochs completed
Loss on test data: 0.11481
Accuracy on test data: 0.9682
Correct predictions: 9682 out of 10000
----------------------------------------

Process finished with exit code 0
```

Рис. 3: Результаты тестирования на MNIST

Видим, что на тестовых данных после 5 эпох удалось добиться точности предсказаний 96.82%

6.2 Тестирование на MNIST-Fashion

Для обучения нейросети по коллекции mnist-fashion была использована аналогичная архитектура, но с измененной размерностью одного из скрытых слоев:

```
    int batch_size = 4;
    double learning_rate = 0.01;
    double weights_decay = 0.0001;
    double beta1 = 0.9;
    double beta2 = 0.999;
    double epsilon = 1e-8;
    int epochs = 5;

    auto network =
      Network({784, 100, 30, 10},
              {ActivationFunction::LeakyReLu(), ActivationFunction::LeakyReLu(),
               ActivationFunction::SoftMax()});

    network.SetOptimizer(
      AdamWOptimizer(learning_rate, weights_decay, beta1, beta2, epsilon));

    network.Train(train_inputs, train_targets, batch_size, epochs,
                  LossFunction::CrossEntropyLoss(),
                  Task::SoftMaxCEClassification);
```

Результаты по прошествии 5 эпох представлены ниже:

```
/Users/nikitaartamonov/CLionProjects/neural-network/cmake-build-release/neural-network
Task: Fashion products recognition
Dataset: MNIST-Fashion

5 epochs completed
Loss on test data: 0.425217
Accuracy on test data: 0.8567
Correct predictions: 8567 out of 10000
---------------------------------------

Process finished with exit code 0
```

Рис. 4: Результаты тестирования на MNIST-Fashion

На тестовых данных после 5 эпох удалось добиться точности 85.67%

Список литературы

[1] *C++17 Standard. ISO/IEC 14882:2017*. URL: https://www.iso.org/standard/68564.html (дата обр. 10.01.2024).

[2] *Clang-Format: a tool to format C/C++/Java/JavaScript/Objective-C/Protobuf/C# code*. URL: https://clang.llvm.org/docs/ClangFormat.html (дата обр. 10.01.2024).

[3] *Clang-Tidy: clang-based C++ "linter" tool*. URL: https://clang.llvm.org/extra/clang-tidy/ (дата обр. 10.01.2024).

[4] *Clang: a C language family frontend for LLVM*. URL: https://clang.llvm.org/ (дата обр. 10.01.2024).

[5] *CMake: an open source, cross-platform family of tools designed to build, test, and package software*. URL: https://cmake.org/ (дата обр. 10.01.2024).

[6] *EigenRand: a header-only library for Eigen, providing vectorized random number engines and vectorized random distribution generators*. URL: https://github.com/bab2min/EigenRand (дата обр. 10.01.2024).

[7] *GCC, the GNU Compiler Collection*. URL: https://gcc.gnu.org/ (дата обр. 10.01.2024).

[8] *Git: a free and open source distributed version control system*. URL: https://git-scm.com/ (дата обр. 10.01.2024).

[9] *GitHub: a complete developer platform to build, scale, and deliver software*. URL: https://github.com/ (дата обр. 10.01.2024).

[10] *Google C++ Style Guide*. URL: https://google.github.io/styleguide/cppguide.html (дата обр. 10.01.2024).

[11] *GoogleTest: Google Testing and Mocking Framework*. URL: https://github.com/google/googletest (дата обр. 10.01.2024).

[12] Gaël Guennebaud, Benoît Jacob и др. *Eigen v3*. 2010. URL: http://eigen.tuxfamily.org (дата обр. 10.01.2024).

[13] Herman Kamper. *Yet another introduction to backpropagation*. URL: https://www.kamperh.com/notes/kamper_backprop17.pdf (дата обр. 20.12.2023).

[14] Yann LeCun, Corinna Cortes и CJ Burges. "MNIST handwritten digit database". В: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010). URL: http://yann.lecun.com/exdb/mnist.

[15] Ilya Loshchilov и Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].

[16] Shivam Mehta. *Deriving categorical cross entropy and softmax*. URL: https://shivammehta25.github.io/posts/deriving-categorical-cross-entropy-and-softmax/ (дата обр. 10.01.2024).

[17] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[18] *Ninja: a small build system with a focus on speed*. URL: https://ninja-build.org/ (дата обр. 10.01.2024).

[19] Sebastian Ruder. "An overview of gradient descent optimization algorithms". В: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[20] *Simple C++ reader for MNIST dataset*. URL: https://github.com/ArthurSonzogni/mnist-fashion (дата обр. 10.01.2024).

[21] Ilya Sutskever, James Martens, George E. Dahl и Geoffrey E. Hinton. "On the importance of initialization and momentum in deep learning". В: *International Conference on Machine Learning*. 2013. URL: https://api.semanticscholar.org/CorpusID:10940950.

[22] Han Xiao, Kashif Rasul и Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 28 авг. 2017. arXiv: cs.LG/1708.07747 [cs.LG]. URL: https://arxiv.org/abs/1708.07747 (дата обр. 30.03.2024).