Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ТАМБОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра *Информационные системы и защита информации*

СОГЛАСОВАНО	УТВЕРЖДАЮ
Главный специалист предприятия	Заведующий кафедрой
_____	*В.В. Алексеев*
подпись инициалы, фамилия	подпись инициалы, фамилия
«____» _____ 20___г.	«____» _____ 20___г.

БАКАЛАВРСКАЯ РАБОТА

на тему:

Информационная система автоматизации вычислительных экспериментов с искусственными нейронными сетями, модернизированными посредством предикторов весовых коэффициентов

по направлению подготовки *09.03.02 «Информационные системы и технологии»*
шифр, наименование направления подготовки

Профиль *Прикладные информационные системы и технологии*
наименование профиля

Автор работы _____ *И.И. Хрущев* Группа *БИС41*
подпись, дата инициалы, фамилия

Обозначение работы *ТГТУ.09.03.02.01.021 БР*

Обозначение документа *ТГТУ.09.03.02.01.021 БР ДЭ*

Руководитель работы _____ *Д.В. Поляков*
подпись, дата инициалы, фамилия

Консультанты по разделам:

1_____ _____
подпись, дата инициалы, фамилия

2_____ _____
подпись, дата инициалы, фамилия

3_____ _____
подпись, дата инициалы, фамилия

Нормоконтролёр _____ *Н.Г. Шахов*
подпись, дата инициалы, фамилия

Тамбов 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время можно наблюдать быстрое развитие различных методов машинного обучения, и, в частности, искусственных нейронных сетей (ИНС), это направление является одним из наиболее перспективных, что подтверждается указом президента Российской Федерации «О развитии искусственного интеллекта в Российской Федерации», в рамках которого была утверждена национальная стратегия развития искусственного интеллекта на период до 2030 года.

Таким образом, построение и анализ искусственных нейронных сетей является актуальной задачей. Вместе с тем, средств для автоматизации вычислительных экспериментов с нейронными сетями немного, и они являются сложными в освоении. Из-за этого средние временные затраты, на построение нейронной сети с высокой скоростью сходимости, довольно велики. Кроме того, ряд потенциальных исследователей не обладает необходимыми компетенциями в области программирования, для реализации нейронных сетей путем написания кода.

Одним из подходов к повышению сходимости искусственных нейронных сетей является прогнозирование значений весовых коэффициентов. Если на этапе построения ИНС задать веса близкие к тем, что будут достигнуты в результате обучения, то время обучения сократится. Назовём программные модули для прогнозирования весовых коэффициентов предикторами.

Таким образом, целью работы является снижение трудовых и временных затрат на постановку вычислительных экспериментов по повышению сходимости искусственных нейронных сетей, модернизированных посредством предикторов весовых коэффициентов.

Для достижения поставленной цели необходимо решить следующие задачи:

– провести аналитический обзор существующих искусственных нейронных сетей, подходов к выбору начального значения весов и соответствующих инструментов разработки;

– разработать объектно-ориентированную архитектуру информационной системы автоматизации вычислительных экспериментов с искусственными нейронными сетями, модернизированными посредством предикторов весовых коэффициентов;

– реализовать спроектированную информационную систему;

– разработать элементы технической документации и произвести тестирование ИС;

– провести проверку гипотезы о влиянии сингулярных чисел на скорость обучения ИНС при их задании в качестве весовых коэффициентов.

Объектом исследования является информационная система автоматизации вычислительных экспериментов по повышению сходимости искусственных нейронных сетей, модернизированных посредством предикторов весовых коэффициентов.

Предметом исследования является автоматизация вычислительных экспериментов по повышению сходимости искусственных нейронных сетей, модернизированных посредством предикторов весовых коэффициентов.

1 Аналитический обзор средств автоматизации вычислительных экспериментов с нейронными сетями

1.1 Общие сведения об искусственных нейронных сетях

Искусственная нейронная сеть (также нейронная сеть) – это математическая модель, программно или аппаратно реализованная по принципу организации биологических нейронных сетей, данный термин охватывает совокупность популярных методов машинного обучения, имитирующих механизмы обучения, которые действуют в живых организмах, а точнее в их нервной системе [1]. Искусственная нейронная сеть по аналогии с биологической нейронной сетью состоит из нейронов и синапсов (связей) между ними (рисунок 1). Общий вид искусственной нейронной сети представлен на рисунке 2.

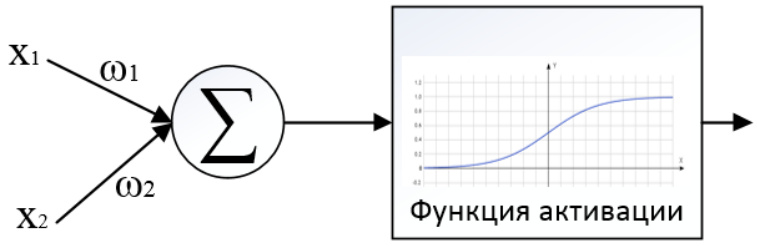


Рисунок 1 – Частный случай нейрона и связей

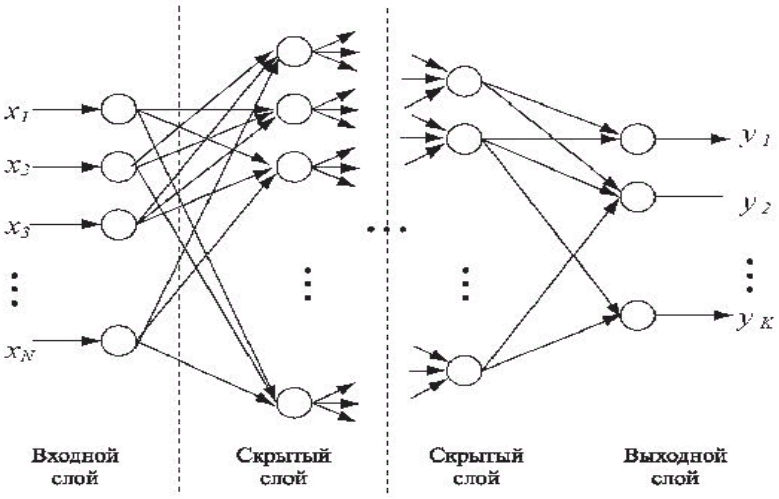


Рисунок 2 – Общий вид нейронной сети

Как видно из рисунка 2, у ИНС имеется множество входов и множество выходов, каждому элементу входного вектора *X* соответствует нейрон, множество таких нейронов образуют входной слой, а на выходе получается

вектор выходных значений *Y*, множество нейронов, которые предоставляют результирующее значение и представляют выходной слой. Кроме этого, каждый входной нейрон имеет связь с нейроном следующего, скрытого слоя. Принцип работы ИНС прост: на входной слой поступает вектор входных значений, затем эти значения поступают на следующий нейрон, и умножаются на вектор весов связей между нейронами [1]. После чего произведения весов и значений суммируются в нейроне, в который они поступают, и к этой сумме применяется заданная функция активации. Математически это можно записать следующим образом:

$$y = f\left(\sum_{i=1}^{k} x_i * w_i\right),$$

где y – выходное значение нейрона;

x_i – значение, поступающее на i-ый вход;

w_i – весовой коэффициент i-ой связи;

k – количество входов нейрона;

f – выбранная функция активации.

Несмотря на относительную простоту при первом приближении обучение ИНС очень сложный процесс, который зависит от множества факторов:

- архитектуры ИНС;
- функции активации;
- начальных значений весовых коэффициентов.

Под архитектурой ИНС понимают количество слоев и количество нейронов в этих слоях, а также связи между слоями сети [2]. В связи с этим существует большое число различных архитектур нейронных сетей, однако все это множество можно разделить на следующие группы:

- нейронные сети с прямой связью;
- рекуррентные нейронные сети;
- радиально-базисные функции;
- самоорганизующиеся карты.

Выбор функции активации – критическая часть процесса проектирования нейронной сети [3]. Функция активации вносит нелинейность в выход нейрона, что способствует выполнению нелинейных преобразований для ввода, благодаря чему нейрон способен выполнять более сложные задачи [1]. В настоящее время наиболее популярными являются следующие функции активации:

- тождественная;
- ступенчатая (знаковая);
- гиперболический тангенс;
- *ReLU* – полулинейный элемент;
- сигмоида;
- спрямленный гиперболический тангенс.

Внешний вид каждой функции активации представлен на рисунке 3.

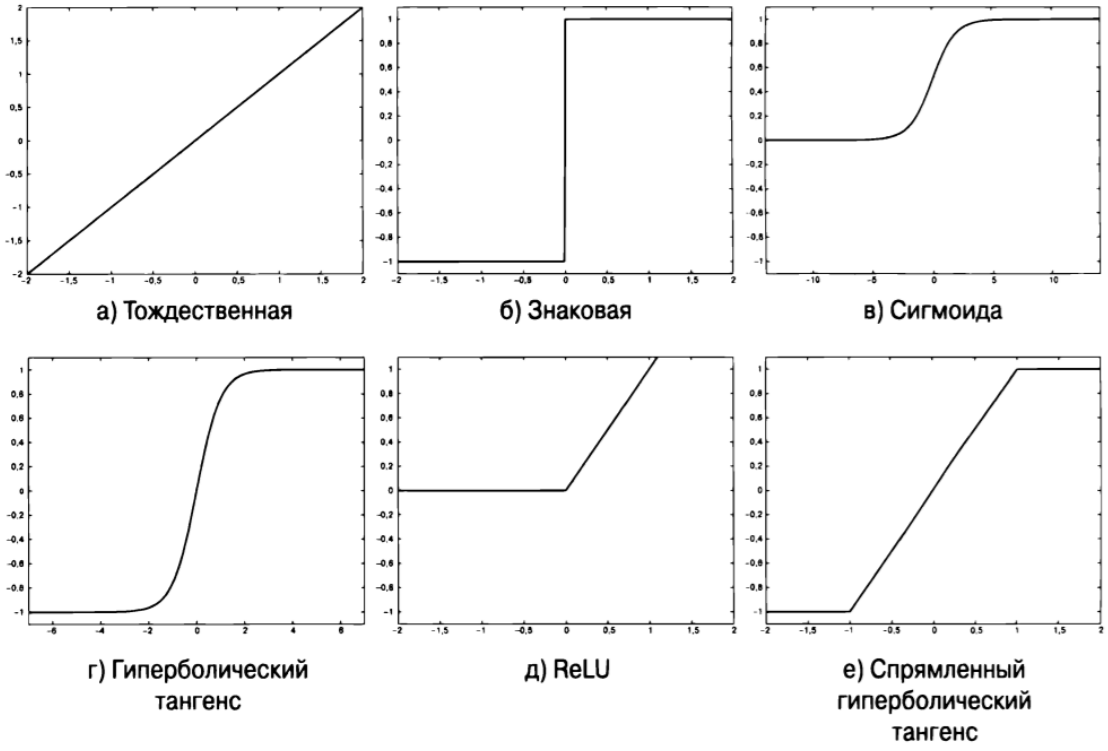


Рисунок 3 – Функции активации

Весовые коэффициенты нейронной сети это один из основных инструментов для повышения качества работы ИНС. Процесс обучения ИНС заключается в подборе таких значений весов, при котором результат всех математических операций приведет к получению нужного результата на выходе. Хорошими считаются значения весовых коэффициентов, достаточно близкие к

оптимальным. Это позволяет не только устранить задержки в точках локальных минимумов, но и значительно ускорить процесс обучения. К сожалению, не существует универсального метода подбора весов, который бы гарантировал нахождение наилучшего начального значения для любой решаемой задачи. По этой причине в большинстве практических реализаций чаще всего применяется случайный подбор весов с равномерным распределением значений в заданном интервале.

Оценить, насколько правильно подобрано сочетание данных параметров (архитектура, функция активации, начальное значение весов) – позволяет сходимость нейронной сети [4]. На рисунке 4 показаны графики обучения нейронной сети, которые сходятся и не сходятся. Кроме этого, сходимость позволяет определить переобучается ли ИНС. Количество эпох, за которое нейронная сеть сошлась, называют скоростью сходимости нейронной сети.

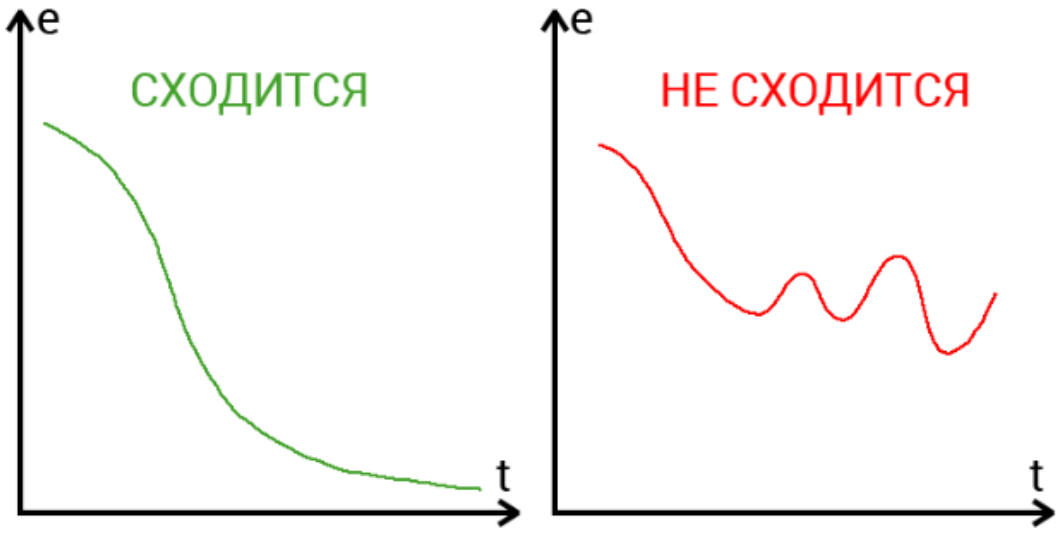


Рисунок 4 – Графики обучения

Процесс обучения ИНС достаточно длительный процесс, требующий больших вычислительных ресурсов, поэтому скорость сходимости нейронной сети является одним из показателей «хорошей» нейронной сети [4].

Суть вычислительных экспериментов с нейронными сетями заключается в подборе такого сочетания архитектуры ИНС, функций активации и способов инициализации весовых коэффициентов, которое минимизирует время обучения, необходимое для решения поставленных задачи с требуемой точностью.

На данный момент существует несколько пакетных решений для создания, обучения и эксплуатации искусственных нейронных сетей. В целях оценки, предоставляемой ими пользователям функциональности, а также выявления достоинств и недостатков, для их последующего учета при разработке собственной системы остановимся на некоторых из них.

1.2 Обзор пакета статистического анализа «*STATISTICA Automated Neural Networks*»

Statistica – это программный пакет для статистического анализа, разработанный компанией *StatSoft*, реализующий функции анализа данных, управления данными, добычи данных, визуализации данных с привлечением статистических методов. Существует четыре версии пакета:

– однопользовательская версия;
– сетевая версия;
– *enterprise* версия;
– веб-версия.

В зависимости от включенных функций существуют следующие комплекты поставки:

– *Base*;
– *Advanced*;
– *Quality Control* (контроль качества);
– *Automated Neural Networks*;
– *Data Miner*;
– *Text Miner*;
– *Process Optimization*;
– *Monitoring and Alerting Server* (*MAS*) (сервер мониторинга и предупреждений).

Нейронную сеть для различного рода экспериментов предоставляет пакет *STATISTICA Automated Neural Networks* (*SANN*). В данном пакете имеется возможность настроить параметры обучения:

– тип нейронной сети;

– функцию активации;

– минимальное и максимальное количество нейронов в скрытом слое;

– функцию ошибки и другие.

STATISTICA Automated Neural Networks является одним из самых передовых и самых эффективных нейросетевых продуктов на рынке. Он предлагает множество уникальных преимуществ и богатый функционал, включающий в себя:

– процессирование, включая выбор данных, кодирование номинальных значений, шкалирование, нормализацию, удаление пропущенных данных с интерпретацией для классификации, регрессию и задачи временных рядов;

– наличие инструмента автоматического нейросетевого поиска «Автоматизированная нейронная сеть»;

– самые современные, оптимизированные и мощные алгоритмы обучения сети;

– поддержка ансамблей нейросетей и нейросетевых архитектур практически неограниченного размера;

– выбор наиболее популярных сетевых архитектур;

– сохранение наилучших нейронных сетей;

– поддержка различного рода статистического анализа и построение прогнозирующих моделей, включая регрессию, классификацию, временные ряды с непрерывной и категориальной зависимой переменной, кластерный анализ для снижения размерности и визуализации;

– поддержка загрузки и анализа нескольких моделей;

– опциональная возможность генерации исходного кода на языках *C, C++, C#, Java, PMML* (*Predictive Model Markup Language*), который может быть легко интегрирован во внешнюю среду для создания собственных приложений.

Однако, такой богатый функционал данного ПО создает несколько проблем. Во-первых, это довольно высокая планка вхождения – для того, чтоб

полноценно использовать такой функционал, необходимо потратить много времени на то, чтобы разобраться в довольно запутанном интерфейсе. Во-вторых, наличие такого большого функционала не всегда необходимо, так как многие функции являются ненужными, для человека, занимающегося созданием нейронных сетей. Исходя из этих минусов можно сделать вывод, что *STATISTICA* является мощным инструментом, но имеет много лишнего функционала, за который тоже необходимо будет заплатить, так как пакет поставляется целиком.

1.3 Обзор среды «*MATLAB*»

MATLAB – это среда и язык технических расчетов, предназначенный для решения широкого спектра инженерных и научных задач любой сложности в любых отраслях.

Это одновременно:

- язык инженерных расчетов;
- графические приложения (приложения с графическим интерфейсом);
- средства разработки программного обеспечения;
- более сотни прикладных программ – профессиональных расширений системы и ее адаптации под решение определенных классов математических и научно-технических задач.

Ввиду своей популярности, и снова набирающей популярности нейронных сетей не обошлось без создания расширений предназначенных для построения нейронных сетей. В настоящее время наиболее известными расширениями являются:

- *Deep Learning Toolbox* (ранее *Deep Learning Toolbox*);
- *Neural Time Series*.

Рассматривать все эти расширения нецелесообразно, так как они являются схожими, из-за реализации в среде *Matlab*, поэтому рассмотрим лишь *Deep Learning Toolbox*.

Пакет для работы с нейронными сетями *Deep Learning Toolbox* представляет собой полноценную среду *MATLAB* для решения прикладных

задач, обеспечивая поддержку проектирования, обучения и моделирования множества известных сетевых парадигм, от базовых моделей персептрона до самых современных ассоциативных и самоорганизующихся сетей. Пакет может быть использован для исследования и применения нейронных сетей к таким задачам, как обработка сигналов, нелинейное управление и финансовое моделирование.

Основные возможности пакета:

– управляемые сетевые парадигмы: персептрон, линейные, обратного распространения, Левенберга, радиальный базис, Элмана, Хопфилда и самообучаемое квантование векторов;

– неуправляемые сети: Хэбб, Кохонен, конкурентные, карты признаков и самоорганизующиеся карты;

– конкурентные, предельные, линейные и сигмоидальные передаточные функции;

– неограниченное число элементов и взаимосвязей;

– настраиваемые на пользователя архитектуры и передаточные функции;

– модульная организация. Пакет использует согласованную, модульную реализацию, которая облегчает исследования и упрощает настройку на пользователя. Пакет не накладывает искусственные ограничения на размер сети или связность;

– архитектуры и обучающие правила. В пакет включены более 15 известных типов сетей и обучающих правил, позволяющих пользователю выбирать наиболее подходящую для конкретного приложения или исследовательской задачи парадигму. Для каждого типа архитектуры и обучающих правил имеются функции инициализации, обучения, адаптации, создания, моделирования и демонстрации сети;

– управляемые и неуправляемые сети. Для управляемых сетей можно выбрать прямую или рекуррентную архитектуру, используя множество обучающих правил и методов проектирования, таких как персептрон, обратное

распространение, обратное распространение Левенберга, сети с радиальным базисом и рекуррентные нейронные сети;

– для неуправляемых сетей можно выбрать ассоциативные или самоорганизующиеся сети, такие как конкурентные, карты свойств и самоорганизующиеся карты. Ассоциативные сети можно использовать как составляющие блоки для более сложных сетей с использованием обучающих правил Хебба, Кохонена, внутренних или внешних (*instar or outstar*) ассоциативных обучающих правил;

– *Deep Learning Toolbox* как инженерная среда. Пакет *Deep Learning Toolbox* предоставляет доступ к полному набору средств для исследования, проектирования и моделирования нейронных сетей. Средства анализа и моделирования *MATLAB* позволяют быстро оценивать поведение сети и ее качество в смысле окончательного результата проектирования.

Matlab и *Deep Learning Toolbox* являются очень мощными инструментами, однако есть проблемы связанные со спецификой языка программирования, а также при проектировании нейронных сетей многие моменты являются скрытыми от пользователя и не до конца понятно все ли работает как требовалось.

В качестве завершения следует отметить, что ни *Statistica* ни *Matlab* не позволяют создавать собственные функции активации, функции задания весов и производить точную настройку ИНС. Это делает невозможным производить различные эксперименты с нейронными сетями, что является очень актуальной тематикой в настоящее время, когда глубокое обучение становится одним из самых популярных направлений.

2 Разработка архитектуры информационной системы

2.1 Применение *SVD*-разложения для задания весовых коэффициентов

В машинном обучении все чаще находит применение *SVD*-разложение (*SVD*), например *SVD* применяется для шумоподавления в изображениях, изучения линейных обратных задач и полезно при анализе методов регуляризации. Метод широко используется в статистике, связанной с анализом главных компонент, а также в обработке сигналов и распознавании образов. *SVD* также используется в выходном модальном анализе, где не масштабируемая форма колебаний может быть определена из особых векторов. Еще одно использование является скрытой семантической индексацией в обработке текста на естественном языке [5].

Представление вещественной матрицы A размерности $m \times n$ виде:

$$A = U * S * V,$$

где S – диагональная матрица $m \times n$ с диагональю из невозрастающих сингулярных чисел s_1, s_2, …, s_k, а U и V – ортогональные $m \times m$ и $n \times n$ матрицы соответственно, называется сингулярным разложением матрицы A или *SVD*-разложением [6].

Особый интерес вызывает матрица S, так как элементы этой матрицы показывают, насколько важен каждый из атрибутов относительно других. Данный факт представляет интерес, так как начальный способ задания весовых коэффициентов ИНС сильно влияет на скорость обучения и результат.

Исследуемая гипотеза состоит в том, что если задать начальное значение весов первого скрытого слоя на основе нормированных диагональных элементов матрицы S (сингулярных чисел), то можно ускорить процесс обучения ИНС. Данная гипотеза основывается на идее, что в результате предложенной инициализации весовых коэффициентов первого скрытого слоя, нейросеть в начале обучения будет склонна делить выборку на априорно созданные кластеры.

Для решения задачи классификации с помощью ИНС, требуется набор данных для обучения, который состоит из массива входных данных X, размерности $m \times n$, и вектор-столбец выходных значений Y, размерности $m \times 1$, который содержит g классов. Для успешного применения *SVD*-разложения требуется применить классификацию и тем самым разделить исходный массив X на g массивов.

$$X = (X^1, X^2, \dots, X^g).$$

Классификация применяется для того, чтобы в каждом получившемся массиве $X^i, i = \overline{1,g}$ остались данные характеризующие i-ый класс, что позволит более точно определить важность каждого атрибута для этого класса. Дальнейший шаг заключается в выделении скрытых подклассов, для этого используется кластеризация, в качестве алгоритма кластеризации, будем использовать *K*-средних. Выбор алгоритма кластеризации эвристичен и направлен на проверку гипотезы. В дальнейшем целесообразно провести исследования с использованием альтернативных алгоритмов.

В результате кластеризации, массивы данных, соответствующие каждому классу, можно будет разделить ещё на k массивов:

$$X^1 = (X_1^1, X_2^1, \dots, X_k^1),$$
$$X^2 = (X_1^2, X_2^2, \dots, X_k^2),$$
$$X^g = (X_1^g, X_2^g, \dots, X_k^g).$$

Кластеризация позволяет определить зависимости между атрибутами массивов, путем выявления их кластерной структуры. Следующий этап заключается в применении *SVD*-разложения и нахождении сингулярных чисел для каждого массива X_j^i, где: i = [1, …, g]; j = [1, …, k]. В результате получим k матриц S_j^i для каждого класса и объединим их в одну матрицу:

$$S^1 = (S_1^1, S_2^1, \dots, S_k^1), S^2 = (S_1^2, S_2^2, \dots, S_k^2), S^g = (S_1^g, S_2^g, \dots, S_k^g),$$
$$S = (S^1, S^2, \dots, S^g).$$

В результате полученную матрицу *S* можно будет установить в качестве матрицы весовых коэффициентов, для ИНС имеющей *n* входов и *g*k* нейронов в первом скрытом слое. Однако для применения сингулярных чисел в ИНС следует нормализовать вектора S_j^i. Каждый элемент матрицы *S* будет показывать, насколько важен соответствующий входной нейрон, для прогнозирования результата.

Использование такого метода в качестве подбора начальных весовых коэффициентов, следует изучить, но средствами существующих программных продуктов реализовать эксперименты крайне затруднительно, из-за отсутствия возможности создания собственных инициализаторов весовых коэффициентов.

Поэтому было принято решение разработать информационную систему автоматизации вычислительных экспериментов по повышению сходимости ИНС, в которой было бы реализовано задание весовых коэффициентов, с помощью *SVD*-разложения. А также информационная система должна иметь гибкую архитектуру для дальнейшей модернизации, посредством дополнения различных инициализаторов и дополнительных средств машинного обучения.

2.2 Процедурная модель информационной системы

Для моделирования поведения информационной системы следует воспользоваться диаграммами вариантов использования и деятельности. Диаграмма вариантов использования предназначена для создания исходного концептуального представления системы, при этом модель не отражает то, как будет реализован набор представленных прецедентов. На рисунке 5 представлена диаграмма вариантов использования для информационной системы автоматизации вычислительных экспериментов.

В качестве актера на ней представлен «Пользователь», который может осуществлять взаимодействие с вариантами использования «Обучение НС» и «Установка параметров НС». Обучение нейронной сети является основным вариантом использования, и он включает в себя обязательную загрузку данных для обучения и визуализацию архитектуры ИНС. Кроме этого, вариант

использования «Обучение НС» расширяется с помощью вариантов использования: «Задание параметров НС», «Построение графика обучения», «Расчет метрик», «Сохранение обученной модели».

Рисунок 5 – Диаграмма вариантов использования

Вариант использования «Установка параметров НС» расширяется с помощью варианта использования «Загрузка параметров НС из файла».

Диаграмма деятельности позволяет описать логику различных процедур, бизнес-процессов, прецедентов. Основным направлением использования диаграмм деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения.

На рисунках 6 – 10 представлены диаграммы деятельности моделируемой системы.

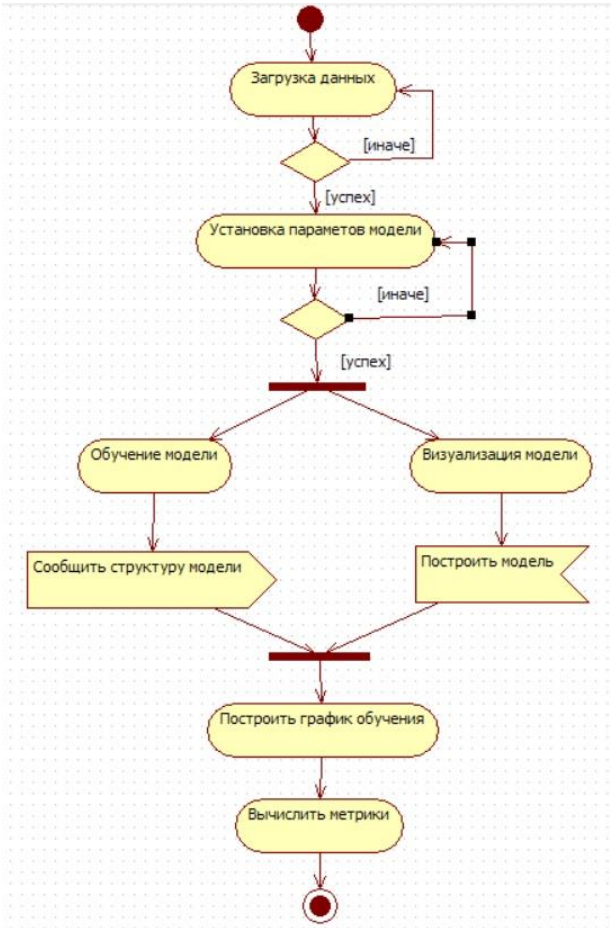


Рисунок 6 – Диаграмма деятельности для обучения НС

Рисунок 7 – Диаграмма деятельности для загрузки модели

Рисунок 8 – Диаграмма деятельности для загрузки данных

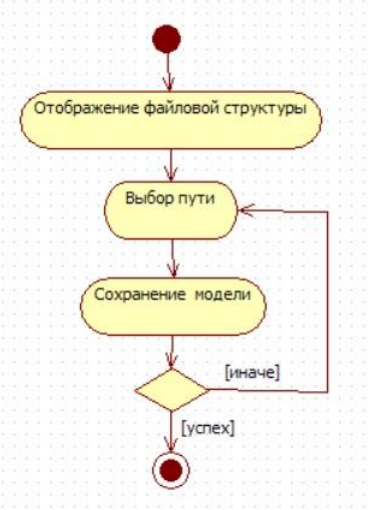


Рисунок 9 – Диаграмма деятельности для сохранения модели

Диаграмма, представленная на рисунке 6, имеет сложную деятельность «Обучение модели», которая требует дополнительной диаграммы, представленной на рисунке 10.

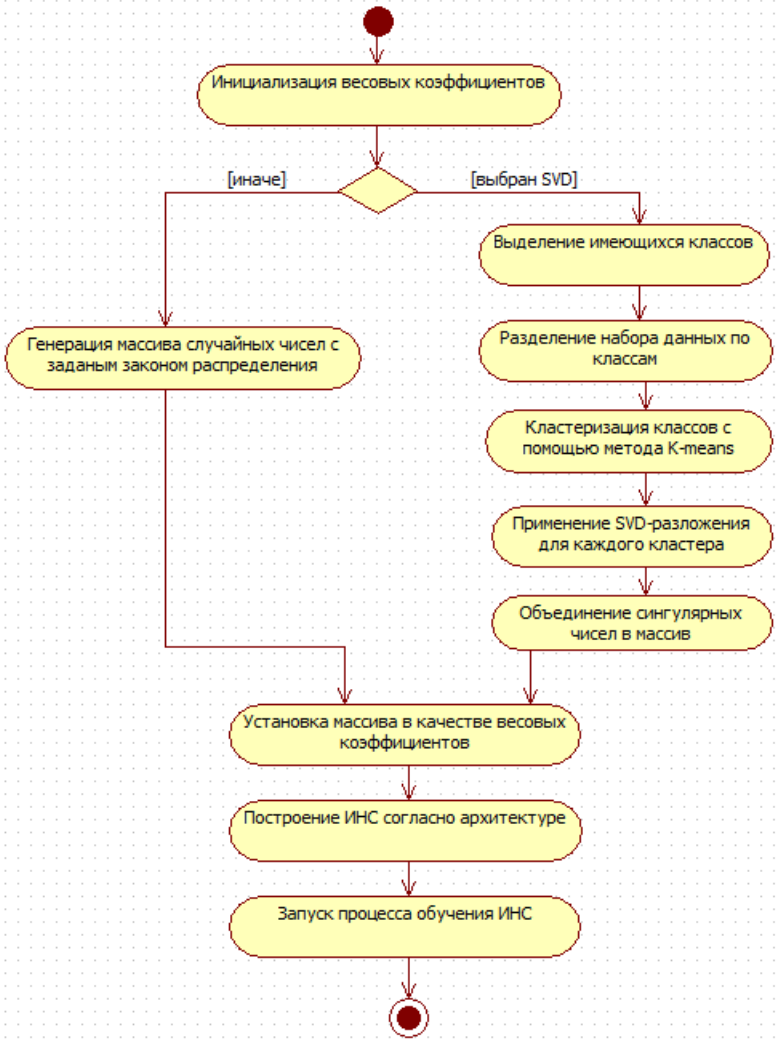


Рисунок 10 – Диаграмма деятельности процесса обучения

2.3 Моделирование объектно-ориентированной архитектуры информационной системы

Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений [7]. Диаграмму классов принято считать графическим представлением таких структурных взаимосвязей логической модели системы, которые не зависят от времени. На рисунке 11 представлена диаграмма классов для информационной системы автоматизации вычислительных экспериментов

по повышению сходимости нейронных сетей, модернизированных посредством предикторов весовых коэффициентов.

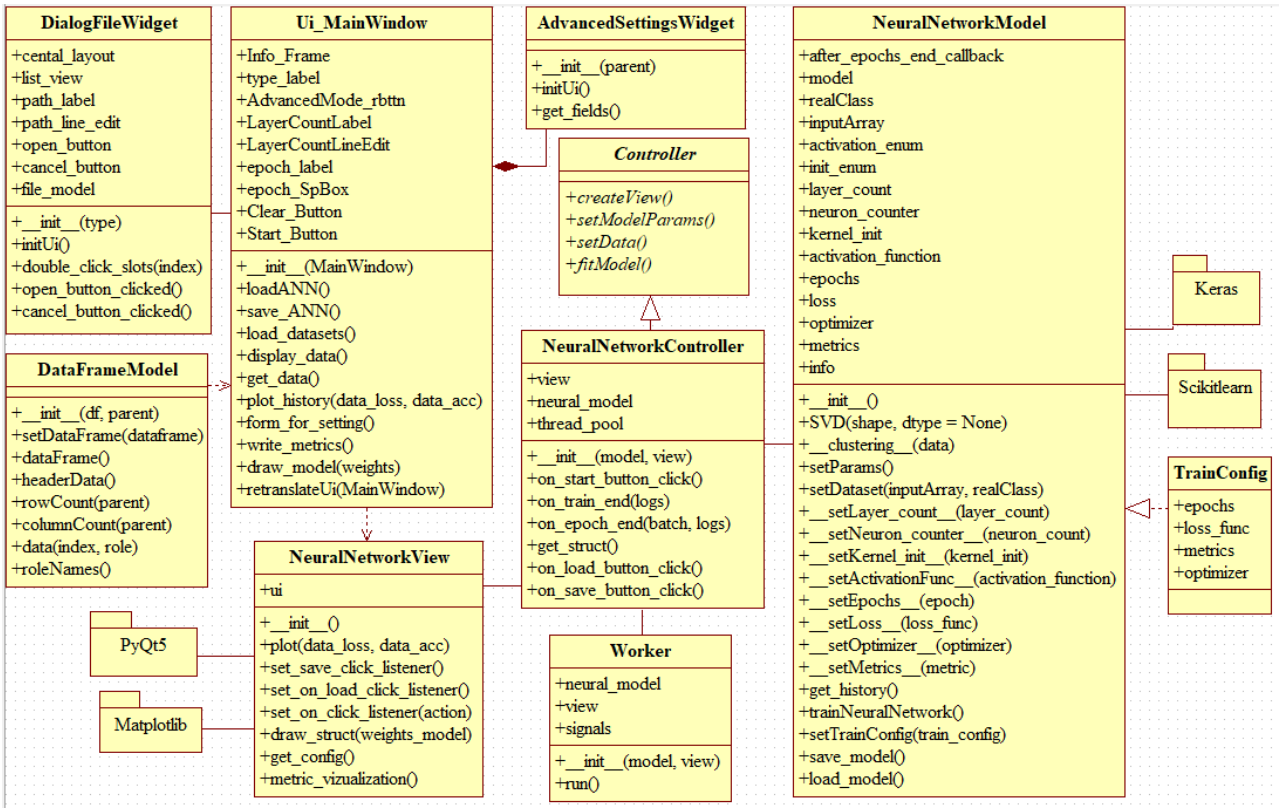


Рисунок 11 – Диаграмма классов

Как видно из диаграммы для разработки информационной системы будет применен паттерн проектирования *MVC*, который позволяет отделить графический интерфейс программы, от данных и бизнес-логики, используемых внутри приложения (модель). А для обеспечения обмена информацией между пользовательским интерфейсом и моделью используется контроллер. Применение такого паттерна позволит сделать архитектуру информационной системы более гибкой для дальнейшей модернизации [8].

Для лучшего понимания архитектуры и принципа работы приложения целесообразно сделать краткое описание классов, их атрибутов и методов. Результаты представлены в таблицах 1 – 8.

Таблица 1 – Описание класса *Controller*

Метод	Комментарий
createView()	Абстрактный метод для создания объекта графического интерфейса
setModelParams()	Абстрактный метод для задания параметров ИНС
setData()	Абстрактный метод для задания данных для обучения
fitModel()	Абстрактный метод для запуска процесса обучения

Класс *Controller* является абстрактным и родительским для класса *NeuralNetworkController*. Основное назначение данного класса – это создание интерфейса для создания сторонних классов, которые расширят функционал информационной системы.

Таблица 2 – Описание класса *NeuralNetworkController*

Атрибут	Комментарий
view	Ссылка на объект класса *NeuralNetworkView*
neural_model	Ссылка на объект класса *NeuralNetworkModel*
thread_pool	Объект для запуска обучения в фоновом потоке
Метод	Комментарий
__init__(*model*,*view*)	Конструктор класса
on_start_button_click()	Слушатель кнопки начала обучения
on_train_end(*logs*)	Функция обратного вызова по завершению обучения
on_epoch_end(*batch*,*logs*)	Функция обратного вызова по завершению эпохи обучения
get_struct()	Метод, запрашивающий структуру ИНС
on_load_button_click()	Слушатель кнопки загрузки ИНС
on_save_button_click()	Слушатель кнопки сохранения ИНС

Класс является наследником абстрактного класса *Controller*, что обязывает его переопределить методы родительского класса. Такой подход позволяет создавать новые контроллеры для разных задач, а не только связанных с ИНС.

Данный класс отслеживает нажатия кнопок на графическом интерфейсе с помощью различных слушателей (*listener*), и заставляет модель выполнять какие-либо действия.

Таблица 3 – Описание класса *NeuralNetworkView*

Атрибут	Комментарий
ui	Содержит указатель на графических интерфейс
Метод	Комментарий
__init__()	Конструктор класса
plot(*data_loss*,*data_acc*)	Метод для построения графика обучения
set_save_click_listener()	Метод для задания слушателя для кнопки сохранения ИНС
set_on_load_click_listener()	Метод для задания слушателя для кнопки загрузки модели
set_on_click_listener(*action*)	Метод для задания слушателя для кнопки запуска обучения
draw_struct(*weights_model*)	Метод для визуализации архитектуры ИНС
get_config()	Метод для получения конфигурации ИНС
metric_vizualization()	Метод для отображения метрик

NeuralNetworkView представляет класс для построения графического интерфейса, большинство его методов связано именно с визуализацией тех или иных данных которые поступают от контроллера. А также данный класс является прослушиваемым классом *NeuralNetworkController*.

Использование слушателей позволяет избавится от жесткой привязки к названиям методов, что делает возможным замену этого графического интерфейса на другой [9].

Таблица 4 – Описание класса *NeuralNetworkModel*

Атрибут	Комментарий
after_epochs_end_callback	Содержит указатель на функцию обратного вызова
model	Хранит данные о ИНС
realClass	Хранит вектор выходных значений
inputArray	Хранит массив входных значений
activation_enum	Список поддерживаемых функций активации
init_enum	Список инициализаторов весовых коэффициентов
layer_count	Хранит количество слоев в ИНС
neuron_counter	Хранит количество нейронов в каждом слое
kernel_init	Список используемых инициализаторов весовых коэффициентов
activation_function	Список используемых функций активаций
epochs	Количество эпох обучения

Окончание таблицы 4

Атрибут	Комментарий
loss	Функция, используемая для вычисления потерь
optimizer	Способ оптимизации ИНС
metrics	Вычисляемая метрика для оптимизации
info	Информация об обучении
Метод	Комментарий
__init__()	Конструктор класса
SVD(*shape*,*dtype*)	Метод, реализующий инициализацию весов с помощью *SVD*-разложения
__clustering__(*data*)	Реализует кластеризацию
setParams()	Устанавливает параметры ИНС
setDataset(*inputArray*,*realClass*)	Устанавливает данные для обучения
__setLayer_count__(*layer_count*)	Устанавливает количество слоев
__setNeuron_counter__(*neuron_count*)	Устанавливает количество нейронов в слоях
__setKernel_count__(*kernel_count*)	Устанавливает инициализаторы весов
__setActivationFunc__(*activation_function*)	Устанавливает функции активации
__setEpochs__(*epoch*)	Устанавливает количество эпох обучения
__setLoss__(*loss_func*)	Устанавливает функцию потерь
__setOptimizer__(*optimizer*)	Устанавливает способ оптимизации ИНС
__setMetrics__(*metric*)	Устанавливает метрику
get_history()	Возвращает информацию об обучении
trainNeuralNetwork()	Метод, запускающий процесс обучения
setTrainConfig(*train_config*)	Устанавливает конфигурацию для обучения
save_model()	Сохранение ИНС
load_model()	Загрузка ИНС

Класс *NeuralNetworkModel* представляет собой обертку над пакетом *Keras*, в котором расширен базовый функционал этого пакета. Данный класс реализует метод инициализации весовых коэффициентов с помощью *SVD*-разложения, а также позволяет конфигурировать и обучать модели искусственных нейронных сетей.

Таблица 5 – Описание класса *Ui_MainWindow*

Атрибут	Комментарий
Info_Frame	Указатель на фрейм с вкладками
type_label	Указатель на надпись
AdvancedMode_rbttn	Указатель на радио-кнопку расширенного режима
LayerCountLabel	Указатель на надпись
LayerCountLineEdit	Указатель на поле ввода количества слоев
epoch_label	Указатель на надпись
epoch_SpBox	Указатель на поле с количеством эпох обучения
Clear_Button	Указатель на кнопку отмены
Start_Button	Указатель на кнопку запуска обучения
Метод	Комментарий
__init__(*MainWindow*)	Конструктор класса
loadANN()	Вызывается при загрузке ИНС
save_ANN()	Вызывается при сохранении ИНС
load_datasets()	Вызывается при загрузке данных
display_data()	Метод для визуализации данных
get_data()	Метод для запроса данных
plot_history(*data_loss*,*data_acc*)	Метод для построения графика обучения
form_for_setting()	Метод для создания окна расширенных настроек ИНС
write_metrics()	Выводит значения вычисленных метрик
draw_model(*weights*)	Метод для визуализации архитектуры ИНС
retranslateUi(*MainWindow*)	Метод для инициализации полей интерфейса

Ui_MainWindow является классом графического интерфейса, который реализован посредствам библиотеки *PyQt*, в нем содержатся все элементы графического интерфейса и способы их отображения.

Таблица 6 – Описание класса *TrainConfig*

Атрибут	Комментарий
epochs	Количество эпох обучения
loss_func	Задаваемая функция потерь
optimizer	Задаваемый оптимизатор
metrics	Метрика для оптимизации

Класс *TrainConfig* представляет собой конфигурацию, задаваемую для процесса обучения.

Таблица 7 – Описание класса *DialogFileWidget*

Атрибут	Комментарий
central_layout	Указатель на главный слой
list_view	Указатель на списковое представление
path_label	Указатель на надпись
path_line_edit	Указатель на строку для задания пути к файлу
open_button	Указатель на кнопку открыть
cancel_button	Указатель на кнопку отмены
file_model	Задаваемы тип файловой модели
Метод	Комментарий
__init__(*type*)	Конструктор класса
initUi()	Метод для создания графического интерфейса
double_click_slots(*ibndex*)	Метод для навигации по файловой системе
open_button_clicked()	Метод для обработки нажатия кнопки
cancel_button_clicked()	Метод для обработки нажатия кнопки

Класс *DialogFileWidget* представляет собой диалоговое окно для загрузки и сохранений файла (данных и ИНС).

Таблица 8 – Описание класса *DataFrameModel*

Метод	Комментарий
__init__(*df*,*parent*)	Конструктор класса
setDataFrame()	Метод для создания датафрейма
dataFrame()	Конструктор датафрейма
headerData()	Метод для инициализации заголовков датафрейма
rowCount()	Счетчик строк
columnCount()	Счетчик колонок
data(*index*,*role*)	Метод для задания данных в датафрейм
roleNames()	Метод для установки заголовков

Данный класс является конвертором данных из текстового файла в датафрейм – объект, представляющий массив данных в удобной для визуализации и обучения ИНС форме.

Таким образом, спроектированная диаграмма классов с применением паттерна *MVC* отличается изолированностью графического интерфейса от данных. А также гибкостью и удобством дальнейшей модернизации, за счет наличия абстрактного класса *Controller*, который представляет интерфейс для реализации контроллера системы, не связанной с ИНС, но связанной с машинным обучением.

Для того чтобы показать последовательность вызовов функций, целесообразно построить диаграммы последовательностей. На рисунках 12 – 15 представлены диаграммы последовательностей для информационной системы автоматизации вычислительных экспериментов.

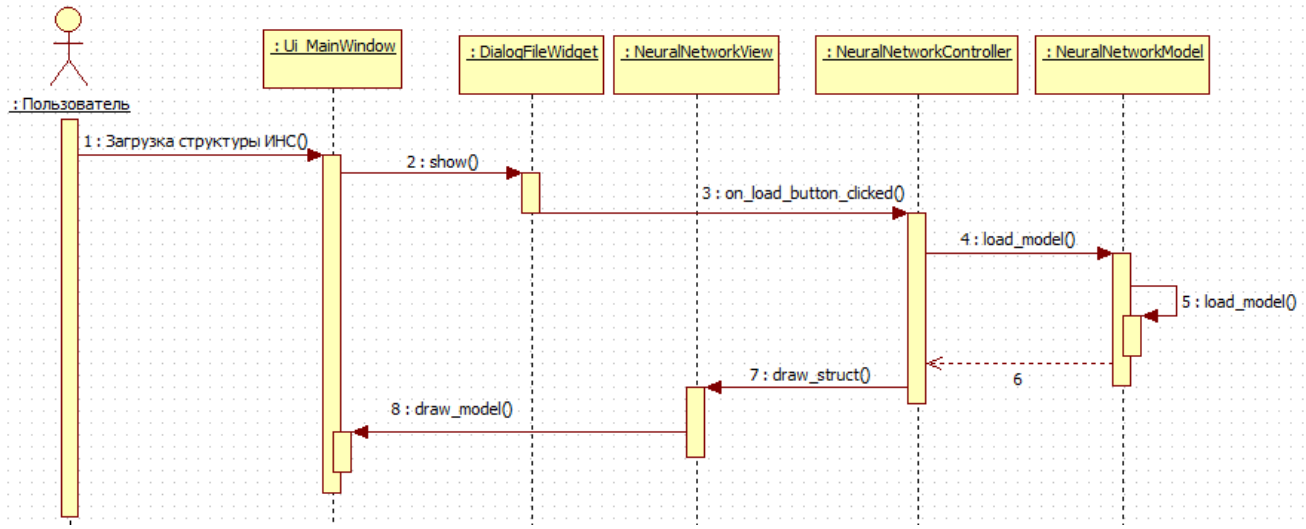


Рисунок 12 – Диаграмма последовательности для загрузки модели

Как видно из этой диаграммы, для загрузки модели из файла, пользователь взаимодействует с графическим интерфейсом, указывая путь и нажимая кнопку загрузки. Затем объект класса *Ui_MainWindow* вызывает метод *show* у объекта класса *DialogFileWidget*. Он, в свою очередь испускает сигнал о событии загрузки модели. Этот сигнал слушает объект класса *NeuralNetworkController*, и по принятию этого сигнала запускает метод *load_model*. После чего объект класса *NeuralNetworkModel* загружает файл с моделью и возвращает информацию о результате загрузки в контроллер, который вызывает метод *draw_struct* у объекта класса *NeuralNetworkView*. Это приводит к вызову метода *draw_model* который отвечает за визуализацию архитектуры ИНС.

Процесс загрузки данных для обучения схож с процессом загрузки модели, но в данном процессе участвуют объекты других классов. Пользователь взаимодействует с графическим интерфейсом, указывая путь хранения файла и подтверждая выбор нажатием кнопки. Это приводит к загрузке файла в объект класса *DataFrameModel* который обрабатывает файл для отображения в графическом интерфейсе и передает результат объекту *Ui_MainWindow*.

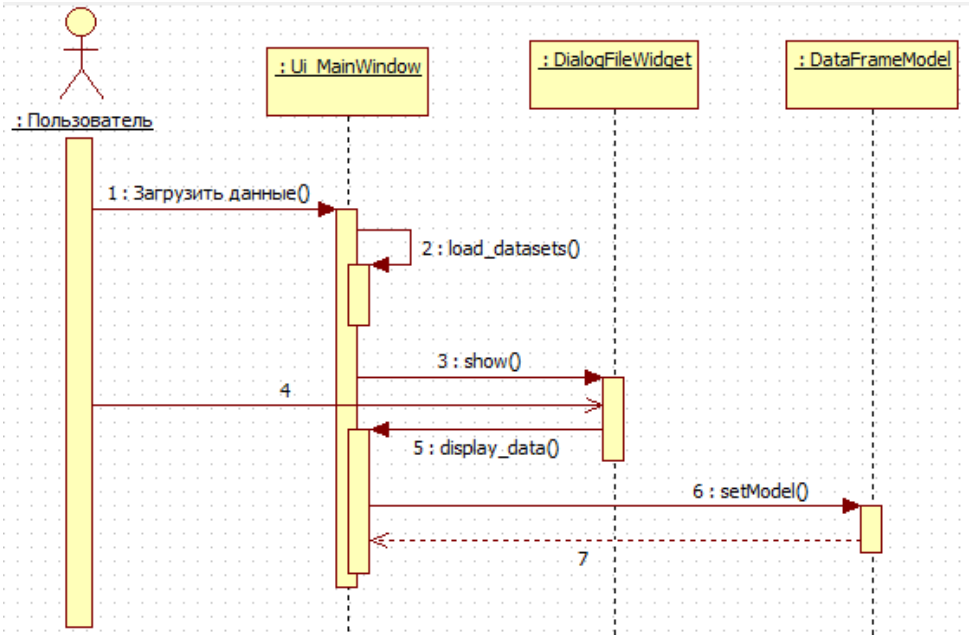


Рисунок 13 – Диаграмма последовательности для загрузки данных

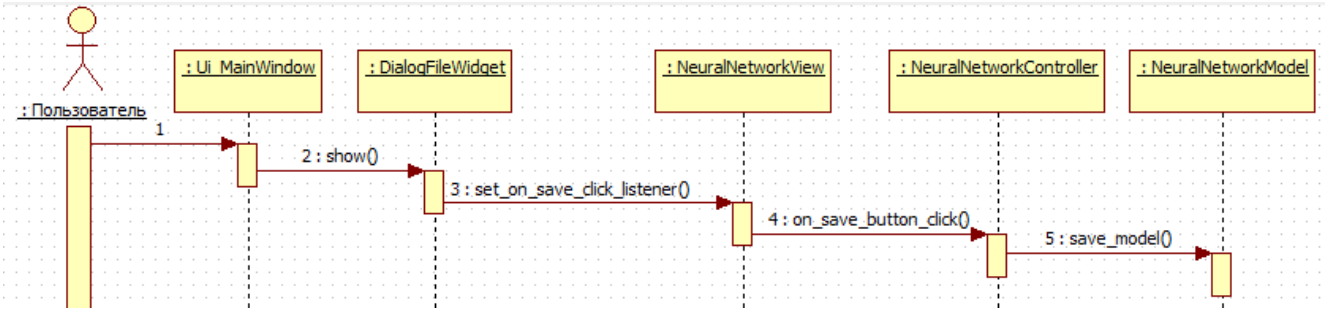


Рисунок 14 – Диаграмма последовательности для сохранения модели

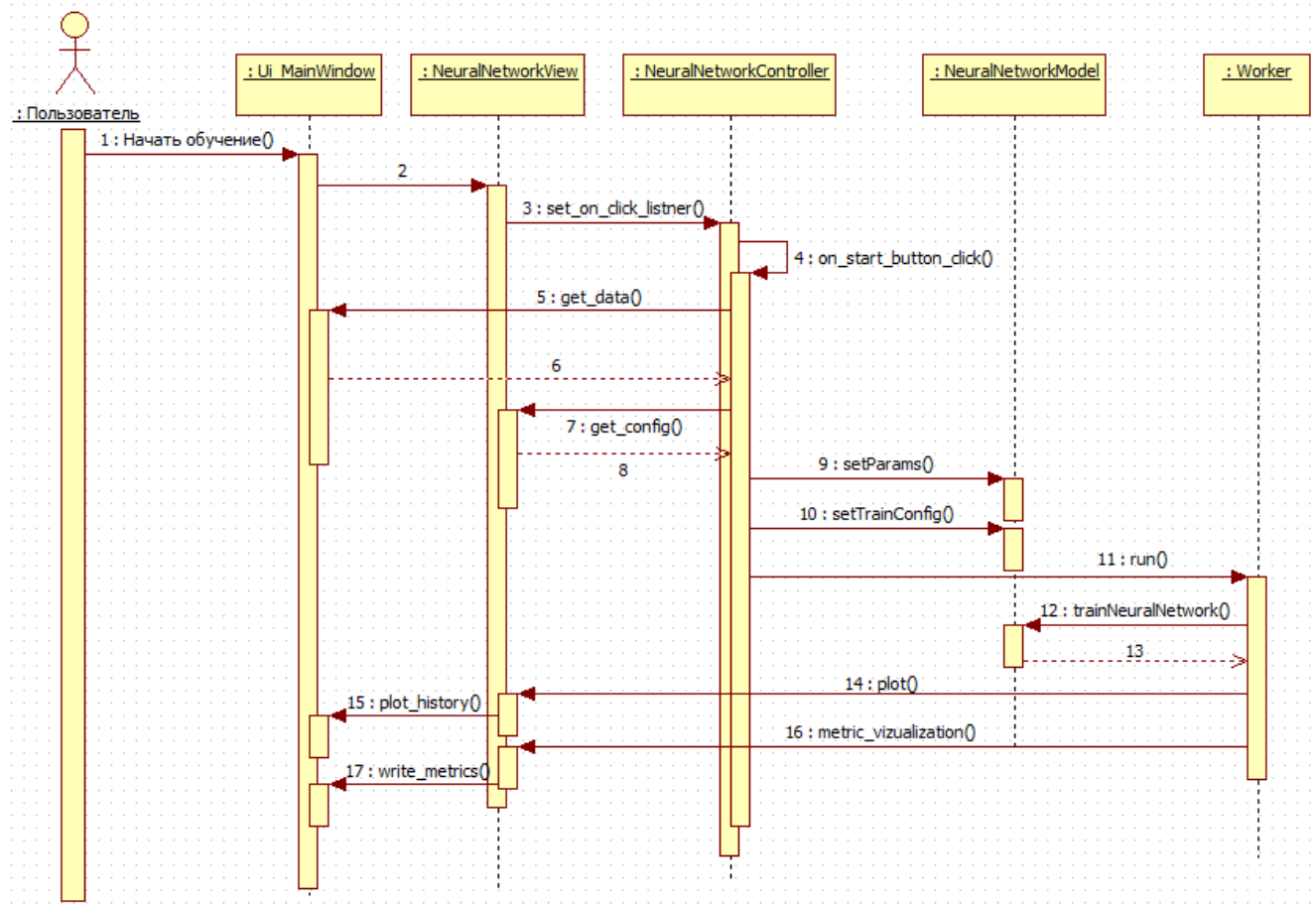


Рисунок 15 – Диаграмма последовательности для обучения НС

Процесс обучения сложнее, когда пользователь нажал кнопку для запуска обучения графический интерфейс испускает сигнал. Объект *NeuralNetworkController* получает этот сигнал и вызывает методы для получения данных для обучения и получения конфигурации ИНС, после чего он вызывает методы объекта класса *NeuralNetworkModel* для задания архитектуры и конфигурации, *setParams* и *setTrainConfig* соответственно. Далее контроллер вызывает метод *run* у объекта класса *Worker*, который запускает процесс обучения в фоновом режиме, что позволяет пользователю продолжать взаимодействовать с графическим интерфейсом [7]. По окончании обучения возвращается архитектура обученной ИНС с установившимися весами и достигнутыми в процессе обучения параметрами точности. Затем *Worker* вызывает методы для построения графика обучения и демонстрации метрик, *plot* и *metric_vizualization* соответственно. А объект *NeuralNetworkView* в свою очередь вызывает методы для визуализации графика обучения и метрик, класса *Ui_MainWindow*.

Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата, которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние [10]. Диаграммы состояний для наиболее важных классов представлены на рисунках 16 – 18.

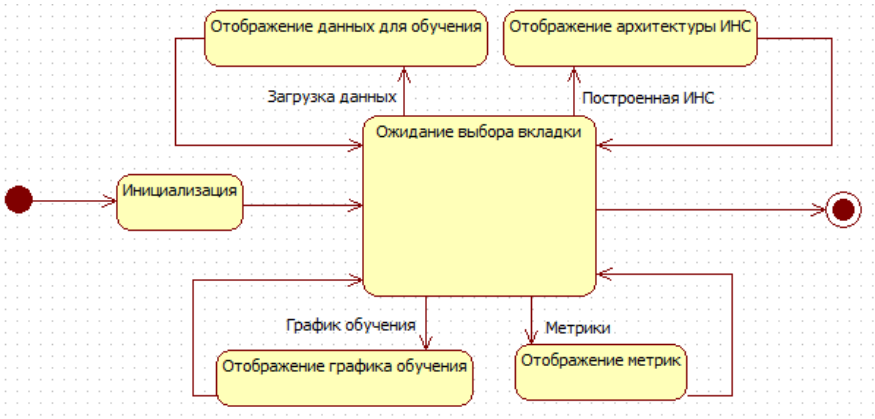


Рисунок 16 – Диаграмма состояний для класса *Ui_MainWindow*

Класс *Ui_MainWindow* после инициализации переходит в состояние ожидания действий пользователя, который может переходить по разным вкладкам, для того чтобы посмотреть данные для обучения, архитектуру ИНС, график обучения и полученные метрики.

После инициализации класс *NeuralNetworkModel* переходит в режим ожидания, из которого может перейти в состояния установки параметров ИНС, задания данных для обучения, сохранения ИНС и установки весовых коэффициентов при запуске обучения. После установки весовых коэффициентов происходит компиляция модели, когда строится ИНС исходя из установленных ранее параметров. После чего класс переходит в состояние обучения ИНС, а по завершении возвращается в режим ожидания.

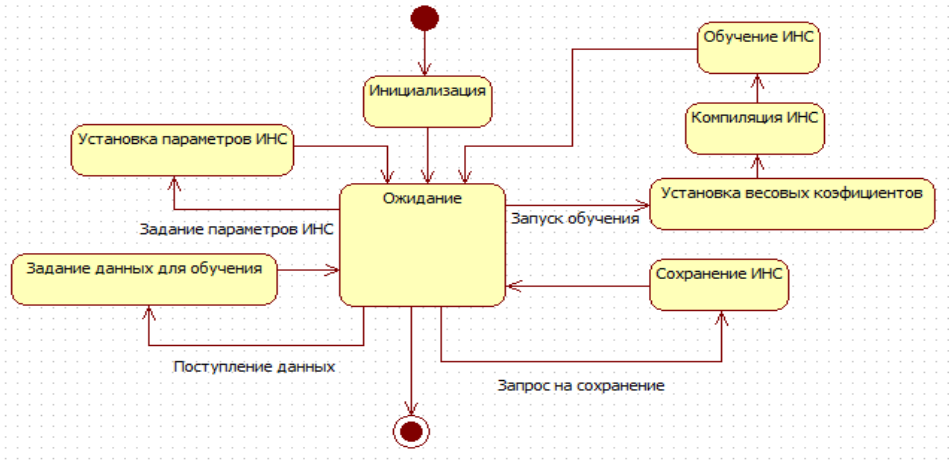


Рисунок 17 – Диаграмма состояний для класса *NeuralNetworkModel*

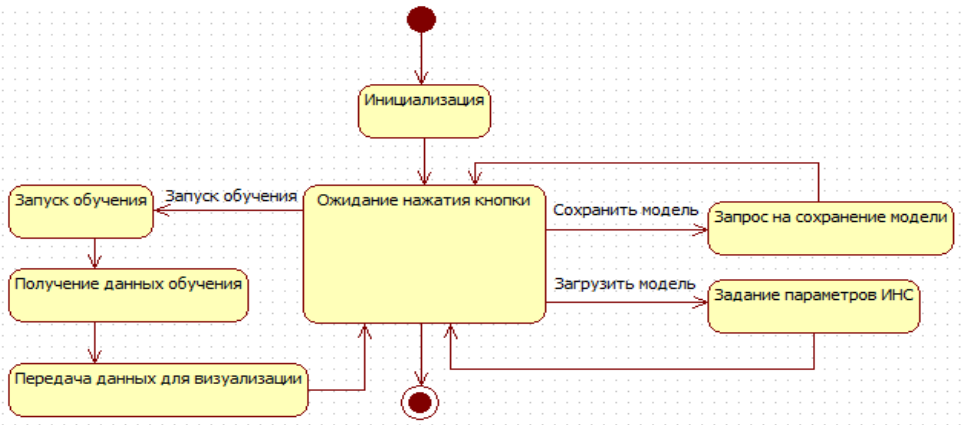


Рисунок 18 – Диаграмма состояний для класса *NeuralNetworkController*

Объект класса *NeuralNetworkController* после инициализации находится в состоянии ожидания нажатия кнопки. Особый интерес представляет состояние запуск обучения, в которое переходит класс *NeuralNetworkController* когда пользователь начинает обучение. Данное состояние заставляет модель начать процесс обучения, и после окончания обучения класс получает данные о том, как прошел процесс обучения, и сразу же передает их для визуализации, после чего возвращается в состояние ожидания.

Все приведенные ранее схемы отображают логику приложения то, как оно функционирует. Для того чтобы описать особенности физического представления системы используют диаграмму компонентов и диаграмму развертывания.

Диаграмма компонентов позволяет определить архитектуру разрабатываемой информационной системы и установить зависимости между

программными компонентами. Диаграмма компонентов информационной системы автоматизации вычислительных экспериментов представлена на рисунке 19.

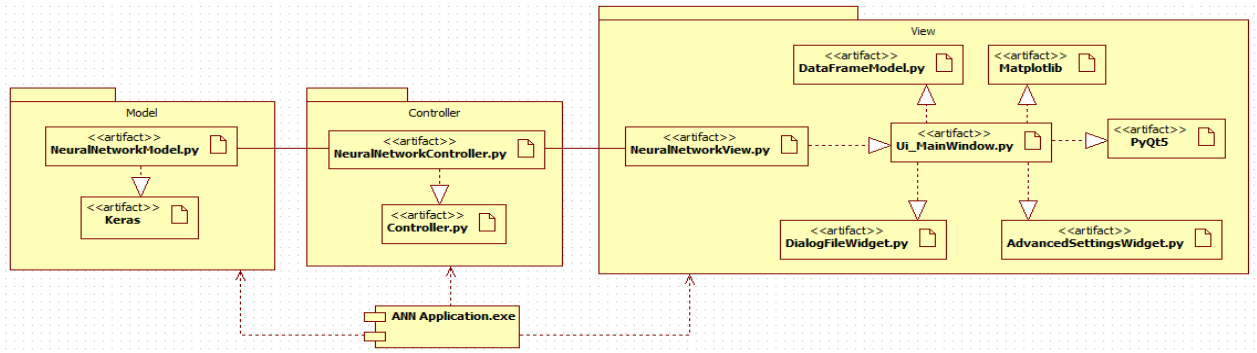


Рисунок 19 – Диаграмма компонентов информационной системы

Диаграмма развертывания предназначена для визуализации компонентов программы, существующих лишь на этапе ее исполнения. Диаграмма развертывания для проектируемой системы представлена на рисунке 20.

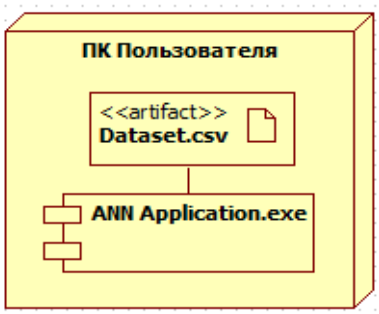


Рисунок 20 – Диаграмма развертывания информационной системы

3 Разработка документации и тестирование ИС

3.1 Элементы технической документации разработанной ИС

В соответствии со спроектированной объектно-ориентированной архитектурой была разработана ИС. Исходный код ИС представлен в приложении А. Рассмотрим функционал разработанной ИС более подробно.

После запуска приложения отображается главное окно информационной системы (рисунок 21), которое состоит из меню, виджета с вкладками, поля выбора режима настройки параметров нейронной сети, текстового поля для задания количества слоев, и поле задания количества эпох обучения, а также кнопки начала обучения и очистки всех полей.

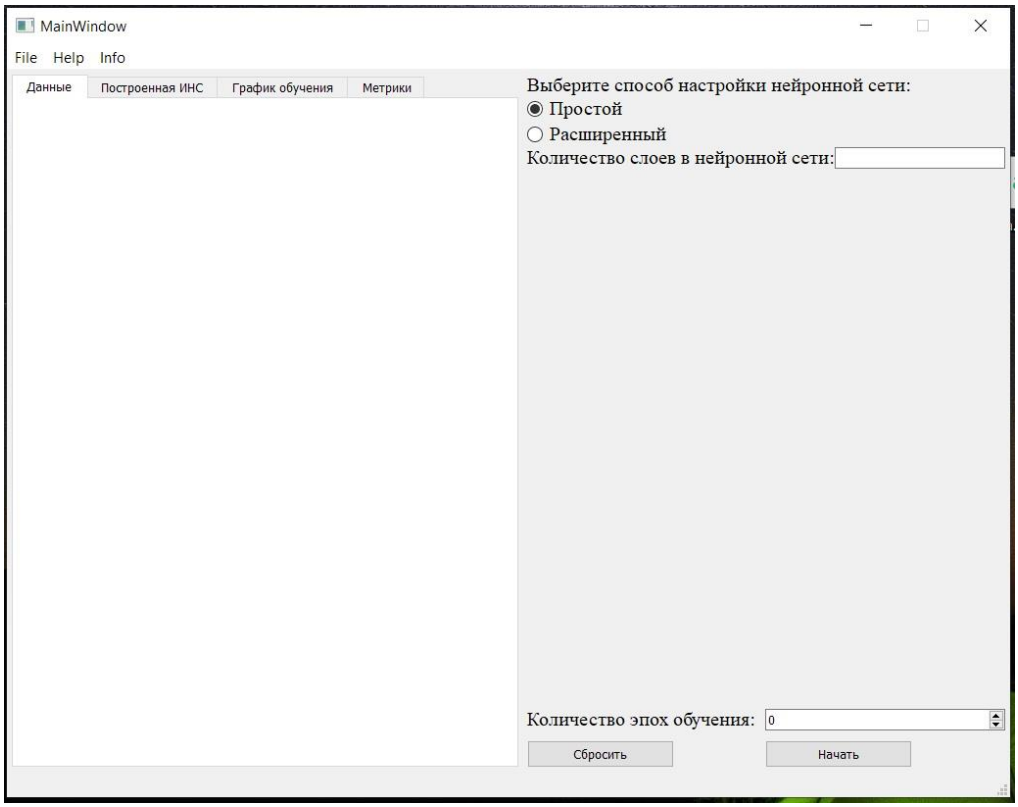


Рисунок 21 – Главное окно

Для того чтобы загрузить данные для обучения следует открыть меню «*File*» и выбрать «*Load dataset*» (рисунок 22), после чего появится диалоговое окно для выбора файла для загрузки (рисунок 23). К файлу с данными предъявляются следующие требования:

– формат файла *txt* или *csv*;

– первая строка является названиями столбцов;

– столбцы разделяются с помощью символа «;»;

– набор данных для обучения не должен содержать текстовые данные;

– столбец с выходными классами должен быть последним и называться «*predict*»

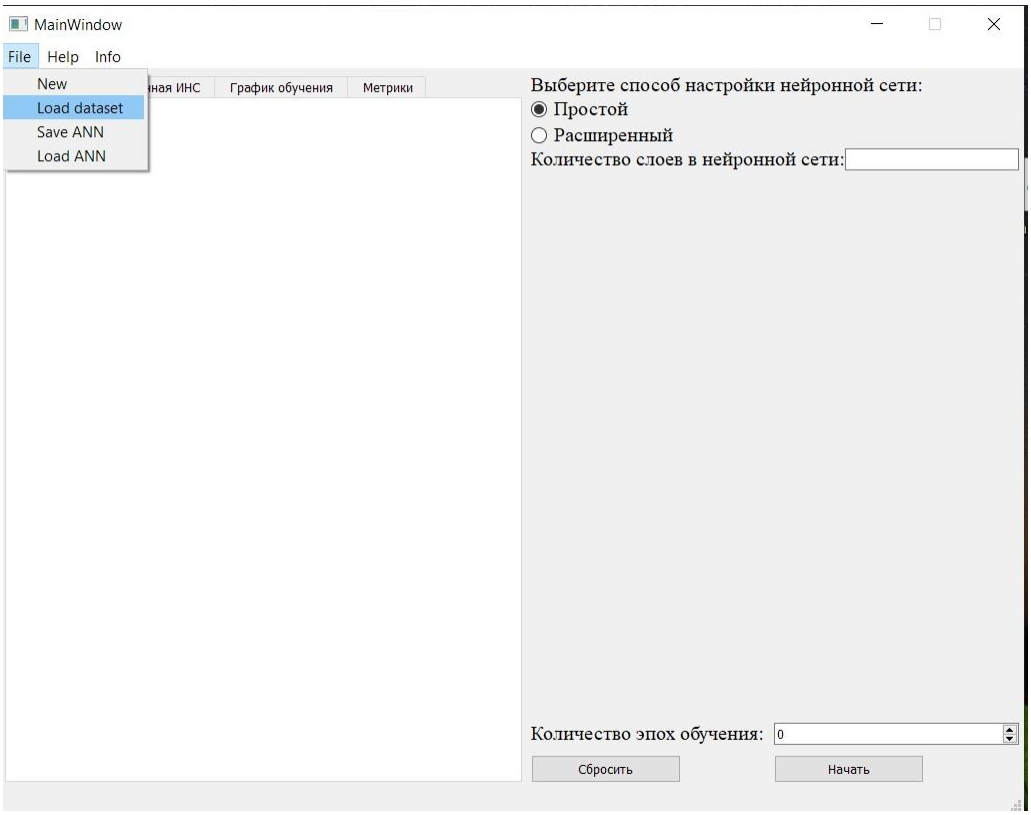


Рисунок 22 – Способ загрузки данных

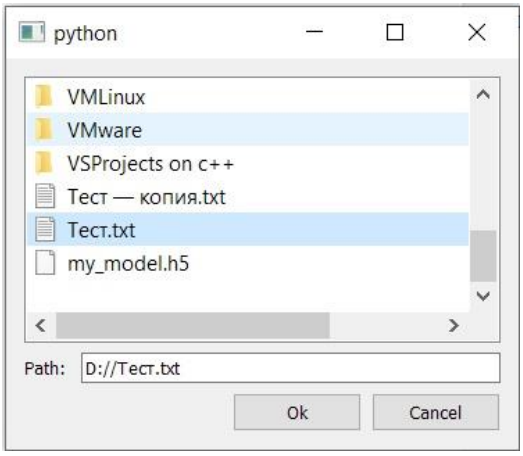


Рисунок 23 – Диалоговое окно загрузки данных

После выбора файла и нажатия кнопки подтверждения, диалоговое окно закрывается и начинается загрузка данных, затем, когда осуществится полная загрузка данных, они отображаются во вкладке «Данные» (рисунок 24).

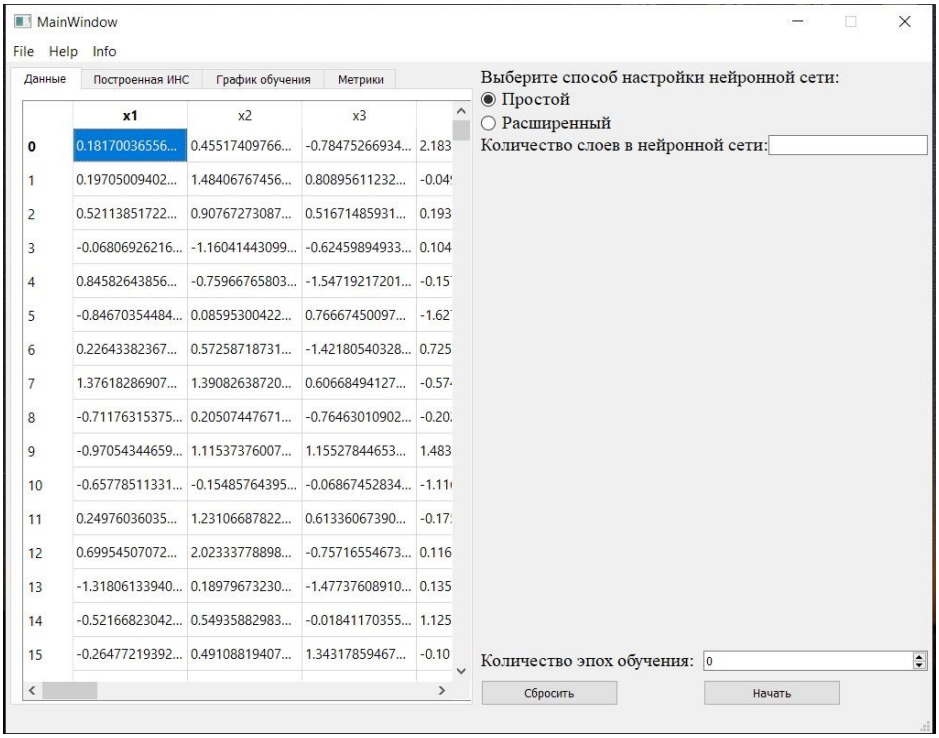


Рисунок 24 – Пример отображения данных

Имеется несколько вариантов задания архитектуры нейронной сети:

– загрузка заранее обученной модели из файла (рисунок 25);

– простой режим – требуется лишь установка количества слоев и эпох обучения, архитектура подберется автоматически;

– расширенный режим – в зависимости от установленного числа слоев появится список, для точной настройки каждого слоя (рисунок 26).

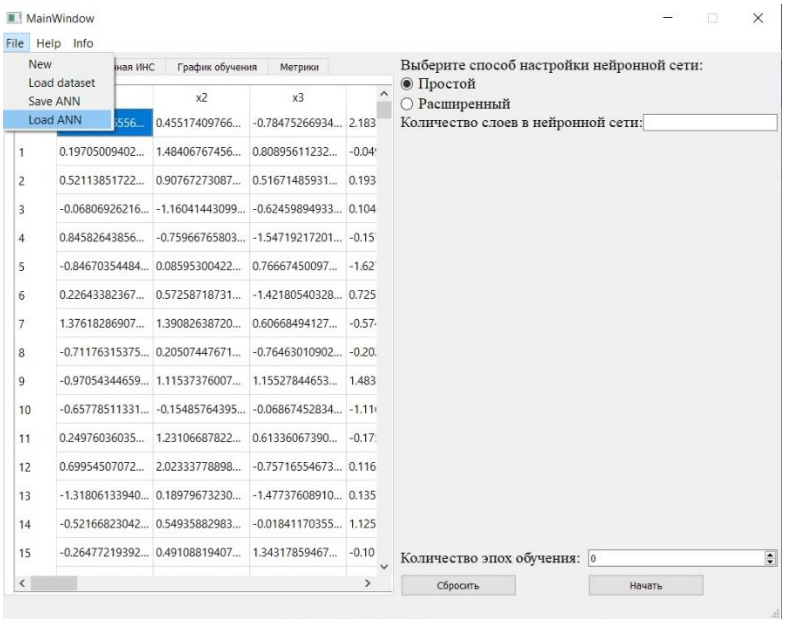


Рисунок 25 – Загрузка НС из файла

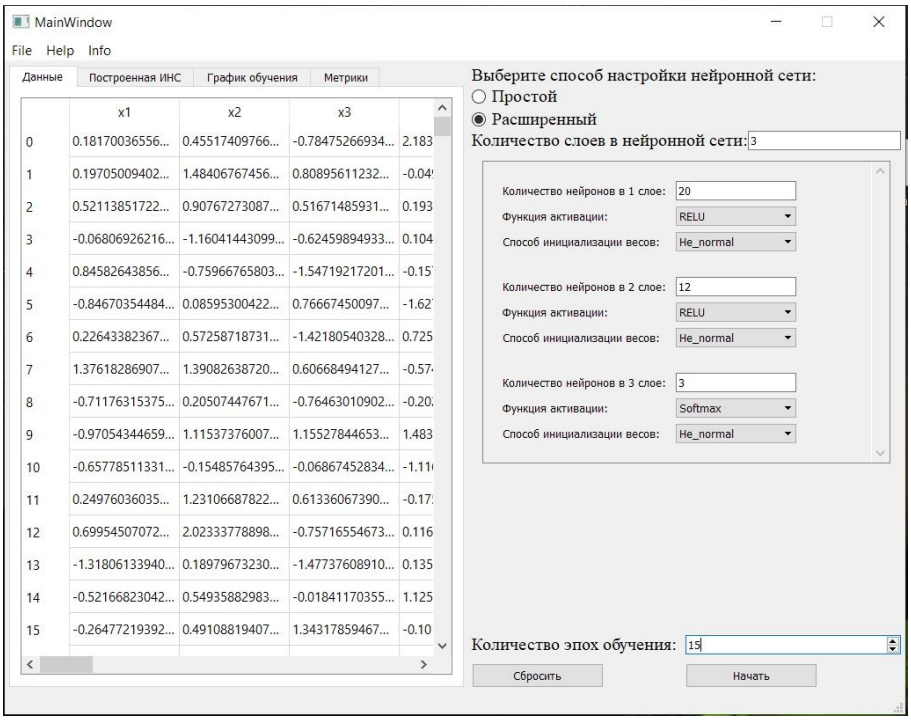


Рисунок 26 – Расширенный режим настройки НС

Расширенный режим позволяет установить количество нейронов в конкретном слое, выбрать функцию активации нейронов, а также установить метод задания весовых коэффициентов для связей в сети.

После задания параметров нейронной сети одним из ранее перечисленных методов можно начать процесс обучения нажав кнопку «Начать». Процесс обучения происходит в фоновом режиме, в этот момент пользователь может перейти во вкладку «Структура НС» и увидеть, какую архитектуру имеет нейронная сеть, визуально оценить влияние каждой связи и то, как они изменяются в процессе обучения (рисунок 27). Для большей наглядности линии связи разделяются на красные и синие, а также изменяются по толщине и прозрачности. Красные связи соответствуют положительным значениям весовых коэффициентов, а синие – отрицательным. Кроме этого, с увеличением абсолютного значения весового коэффициента увеличивается непрозрачность и толщина линии (связи). Таким образом значению веса, равному единице, будет соответствовать совсем непрозрачная толстая красная линия связи. А минус единице будет соответствовать совсем непрозрачная толстая синяя линия связи.

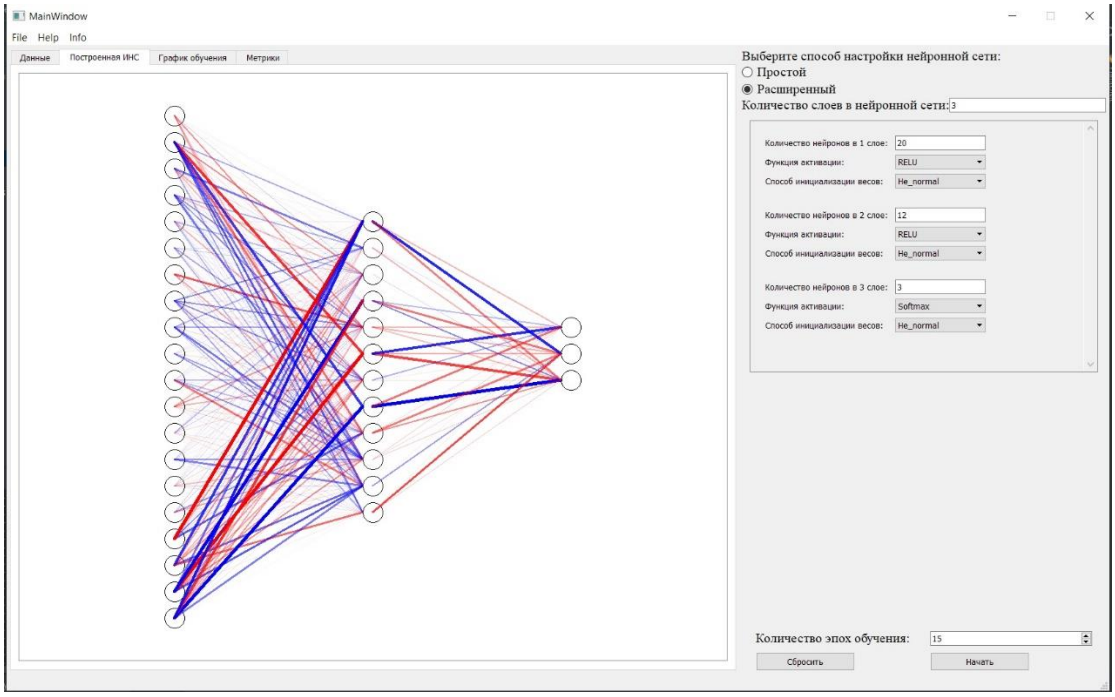


Рисунок 27 – Содержимое вкладки «Структура НС»

После окончания обучения архитектура нейронной сети останется доступной для анализа, кроме того, при открытии вкладки «График обучения» (рисунок 28), будет выведен график обучения, на котором видны значения ошибки и точности обученной модели, в зависимости от эпох.

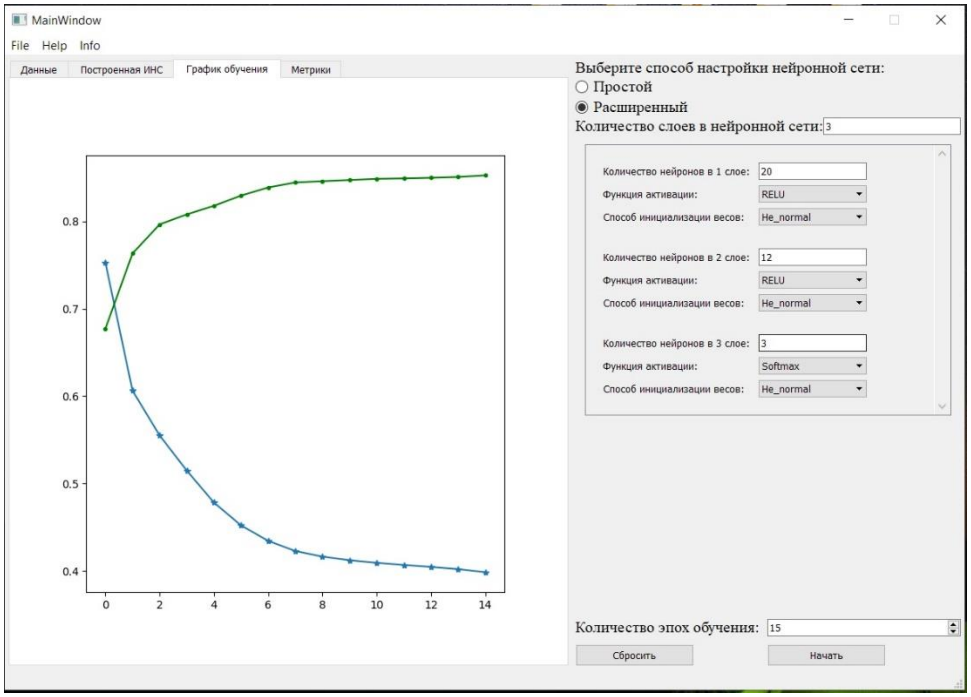


Рисунок 28 – Содержимое вкладки «График обучения»

Во вкладке «Метрики» можно найти значения вычисленных метрик, для оценки точности нейронной сети.

3.2 Результаты тестирования и проверки гипотезы посредством разработанной ИС

Воспользуемся разработанной информационной системой для проверки поставленной ранее гипотезы, и произведем тестирование ИС.

Исходя из разработанной документации, процесс проверки гипотезы следует начать с загрузки данных для обучения. Запустив приложение, следует выбрать *File>Load dataset* для загрузки данных. После чего, в появившемся диалоговом окне следует выбрать файл для загрузки и нажать кнопку «*Ok*». После этого начнется процесс загрузки и обработки данных для визуализации в приложении. Процесс загрузки данных для обучения представлен на рисунках 29 – 31.

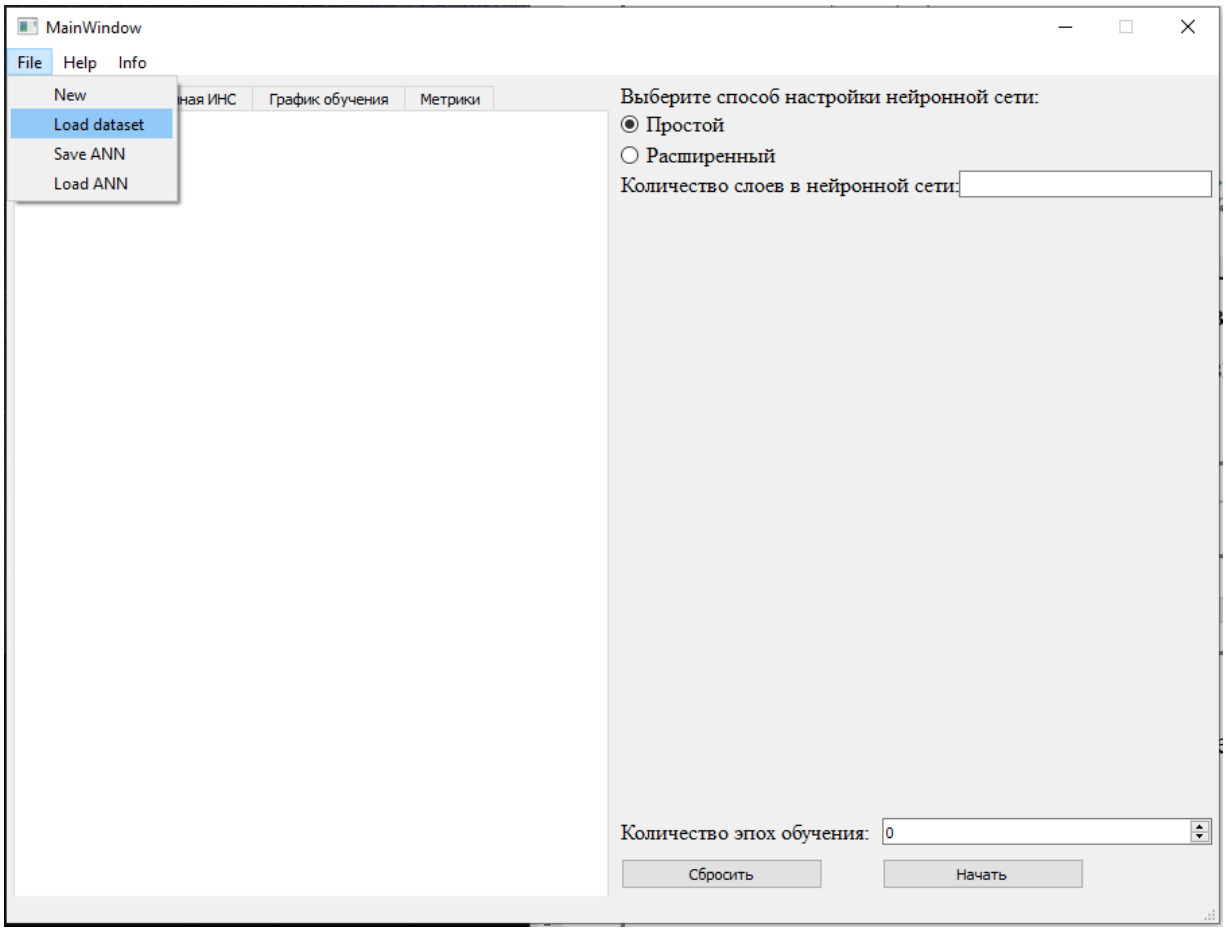


Рисунок 29 – Первый этап загрузки данных

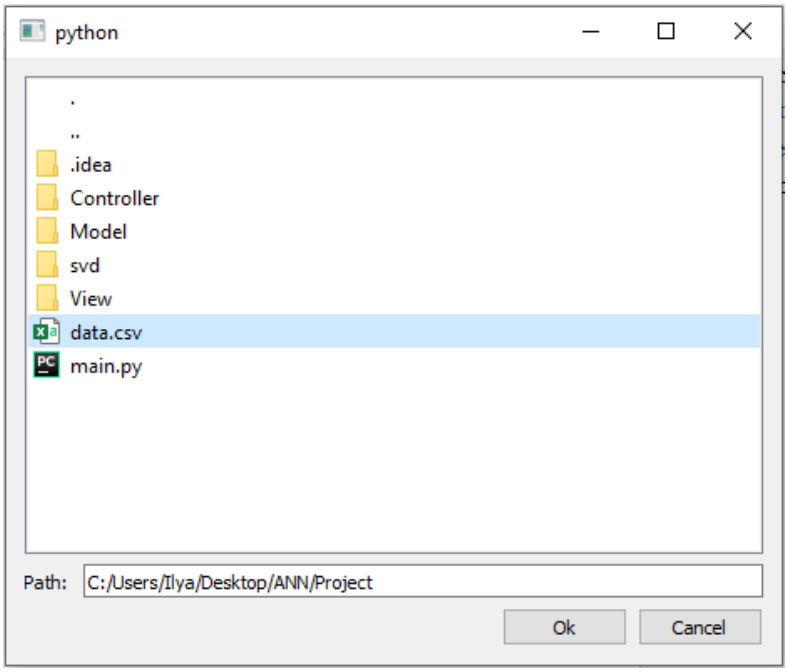


Рисунок 30 – Второй этап загрузки данных

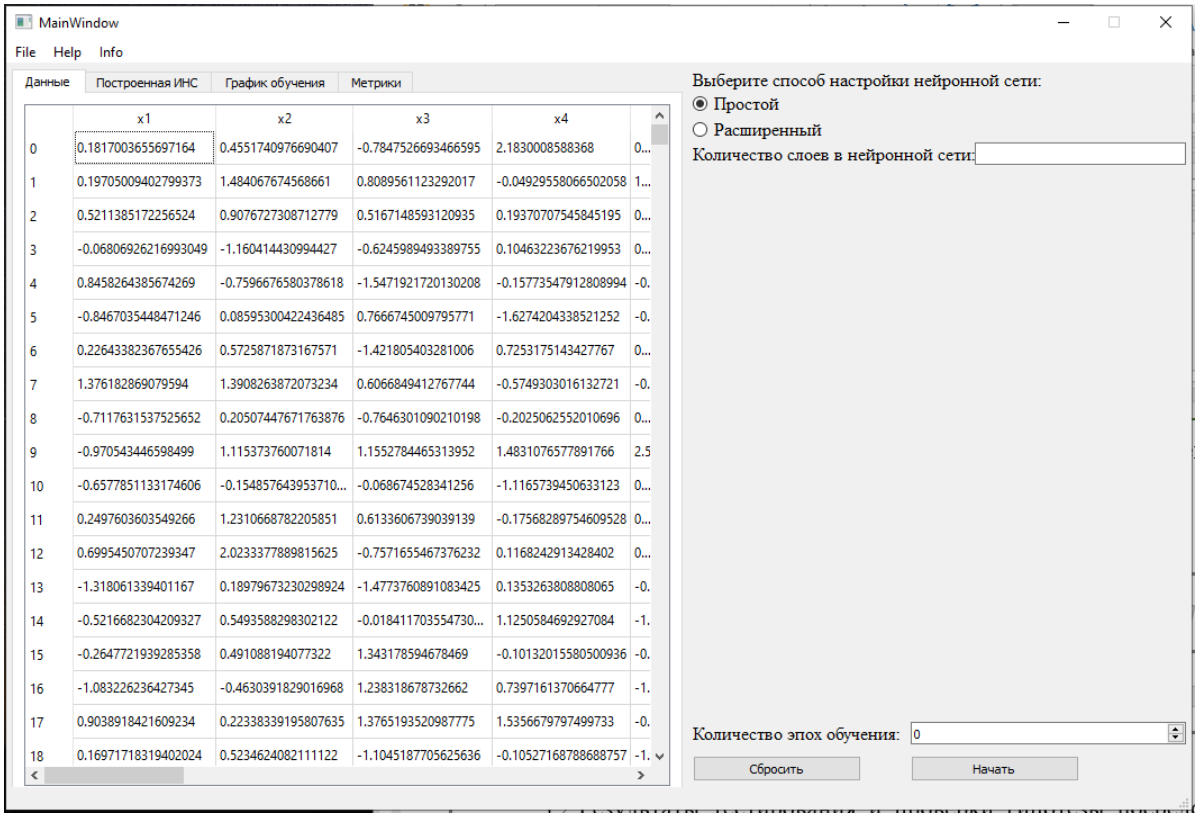


Рисунок 31 – Результат загрузки данных

Следующий шаг – это задание параметров ИНС, которая будет обучаться. Для этого следует проанализировать данные. Визуально оценив данные для обучения, можно сделать следующий вывод: данные представляет собой массив размерности 21×100000, в котором 20 столбцов это атрибуты классов, и один

столбец имеет индексы классов, к которому относится набор конкретных атрибутов, также видно, что набор данных имеет всего три выходных класса.

Основываясь на данном выводе, архитектура ИНС будет иметь первый (входной) слой с двадцатью нейронами (каждый нейрон будет соответствовать конкретному атрибуту). Так как решается задача классификации, то количество выходных нейронов будет соответствовать количеству классов в наборе данных.

Для задания параметров ИНС следует указать количество слоев в ИНС и выбрать расширенный режим, после чего появится возможность вручную указать все необходимые параметры (рисунок 32).

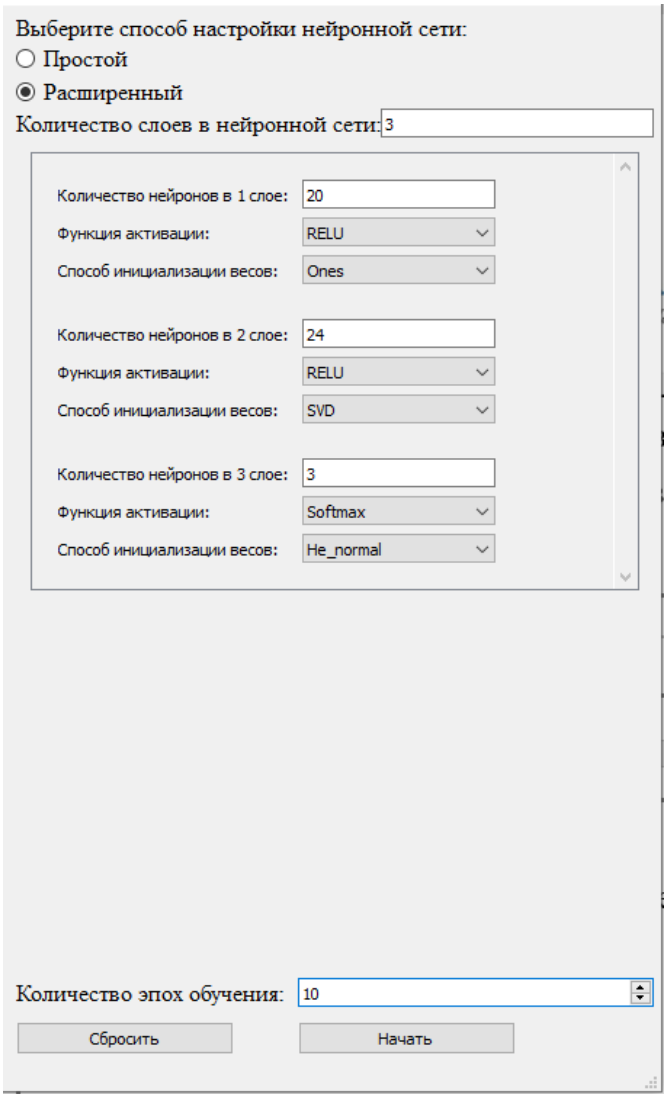


Рисунок 32 – Параметры обучения

После задания параметров можно начать процесс обучения. Как видно из рисунка 32 в качестве инициализатора весовых коэффициентов второго слоя указан *SVD*.

В процессе обучения сети можно наблюдать, как изменяются веса в ИНС. По окончании обучения архитектура ИНС и установившиеся весовые коэффициенты можно изучить, выбрав вкладку «Построенная ИНС» (рисунок 33). Как видно из построенной ИНС, метод инициализации весовых коэффициентов, основанный на *SVD* разложении, выделил три наиболее важных нейрона, об этом говорит количество положительных весов.

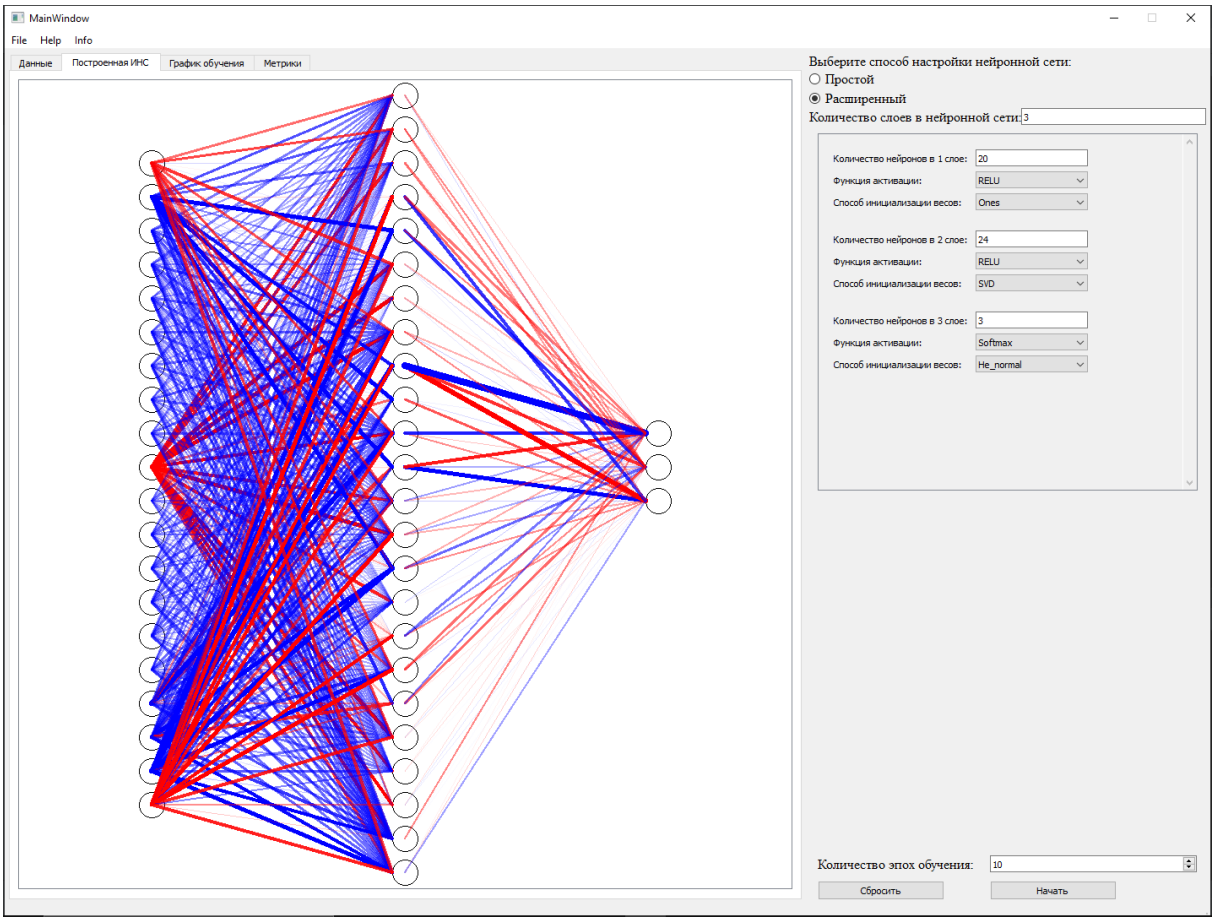


Рисунок 33 – Визуализация обученной ИНС

Также большой интерес представляет график обучения ИНС. Для его изучения перейдем во вкладку «График обучения». На рисунке 34 представлен график обучения, из которого видно, что сходимость и скорость сходимости построенной ИНС высоки, а ошибка в точности предсказания классов мала и имеет тенденцию к уменьшению, что позволяет говорить о дальнейшем обучении сети с целью уменьшения ошибки.

Последним шагом для оценки построенной ИНС является анализ метрики точности предсказания. Перейдем во вкладку «Метрики» (рисунок 35). Значение

метрики *Accuracy* 0,845 достаточно высокое и говорит о том, что ИНС обучена хорошо и способна классифицировать неизвестные для нее данные.

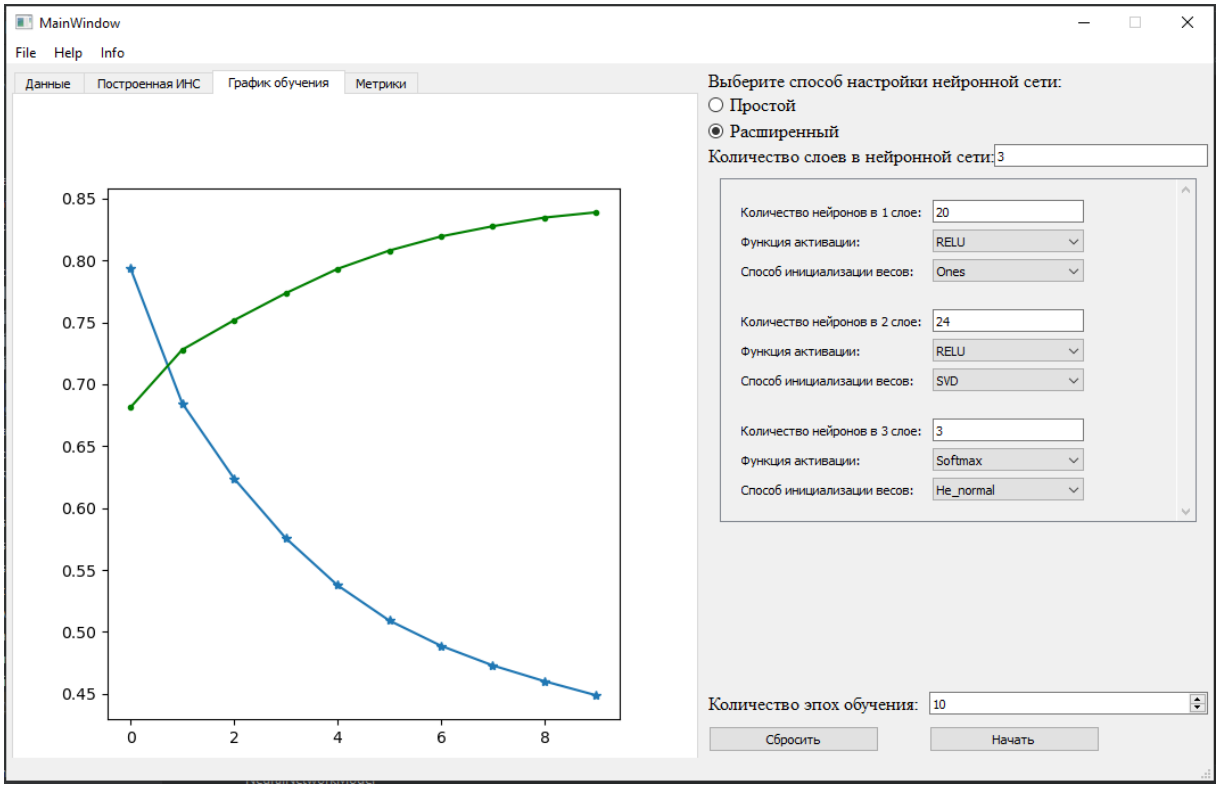


Рисунок 34 – График обучения

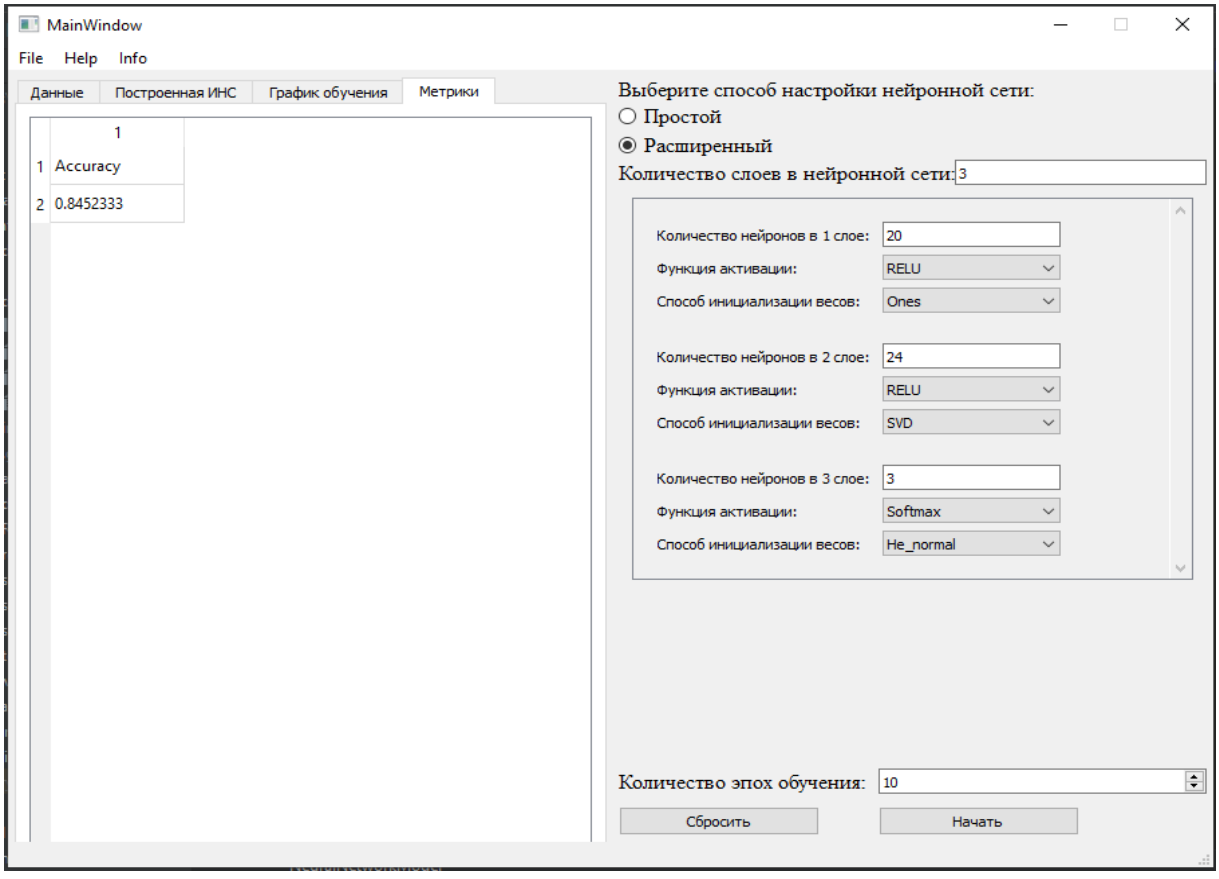


Рисунок 35 – Значение метрики

Сохраним обученную ИНС для возможного дальнейшего использования (рисунок 36).

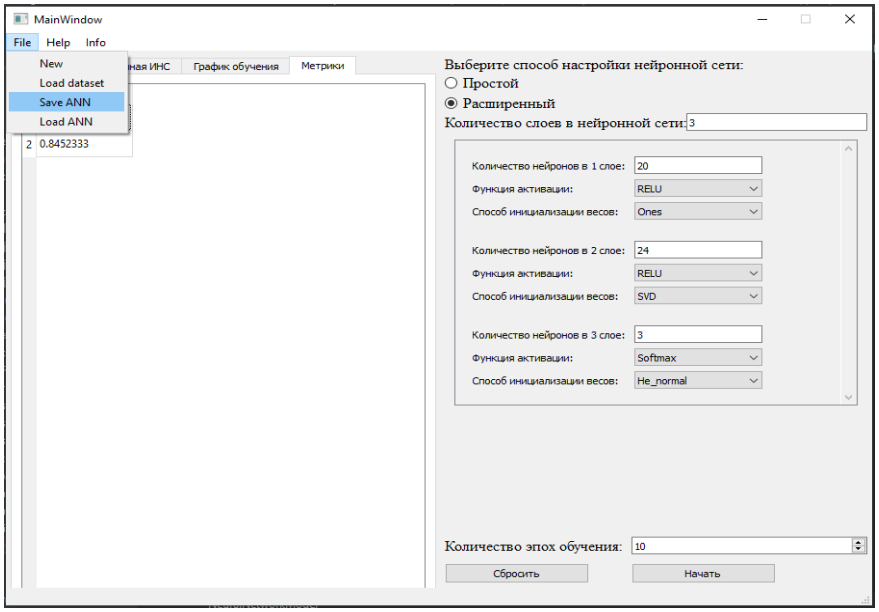


Рисунок 36 – Сохранение ИНС

Несмотря на хорошие результаты обучения ИНС, с методом *SVD* в качестве инициализатора весовых коэффициентов, следует построить ИНС с другими инициализаторами весовых коэффициентов, а также попробовать различные архитектуры ИНС. Процесс обучения аналогичной нейронной сети с единственным отличием в способе задания весов представлен на рисунках 37 – 40.

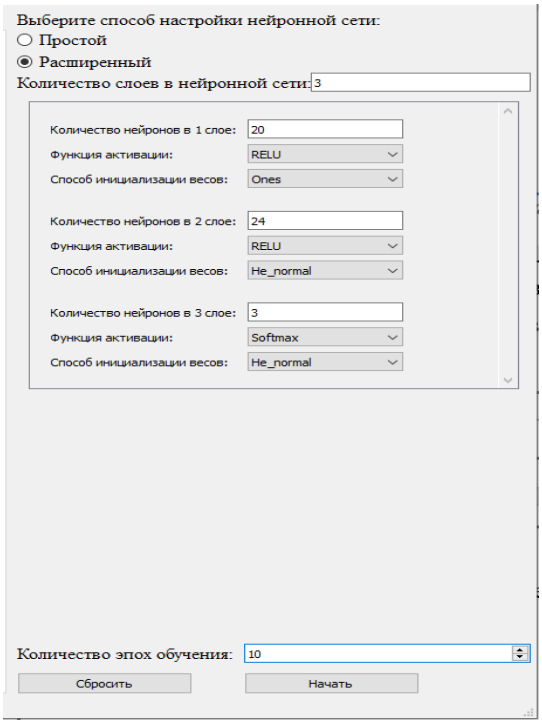


Рисунок 37 – Параметры обучения

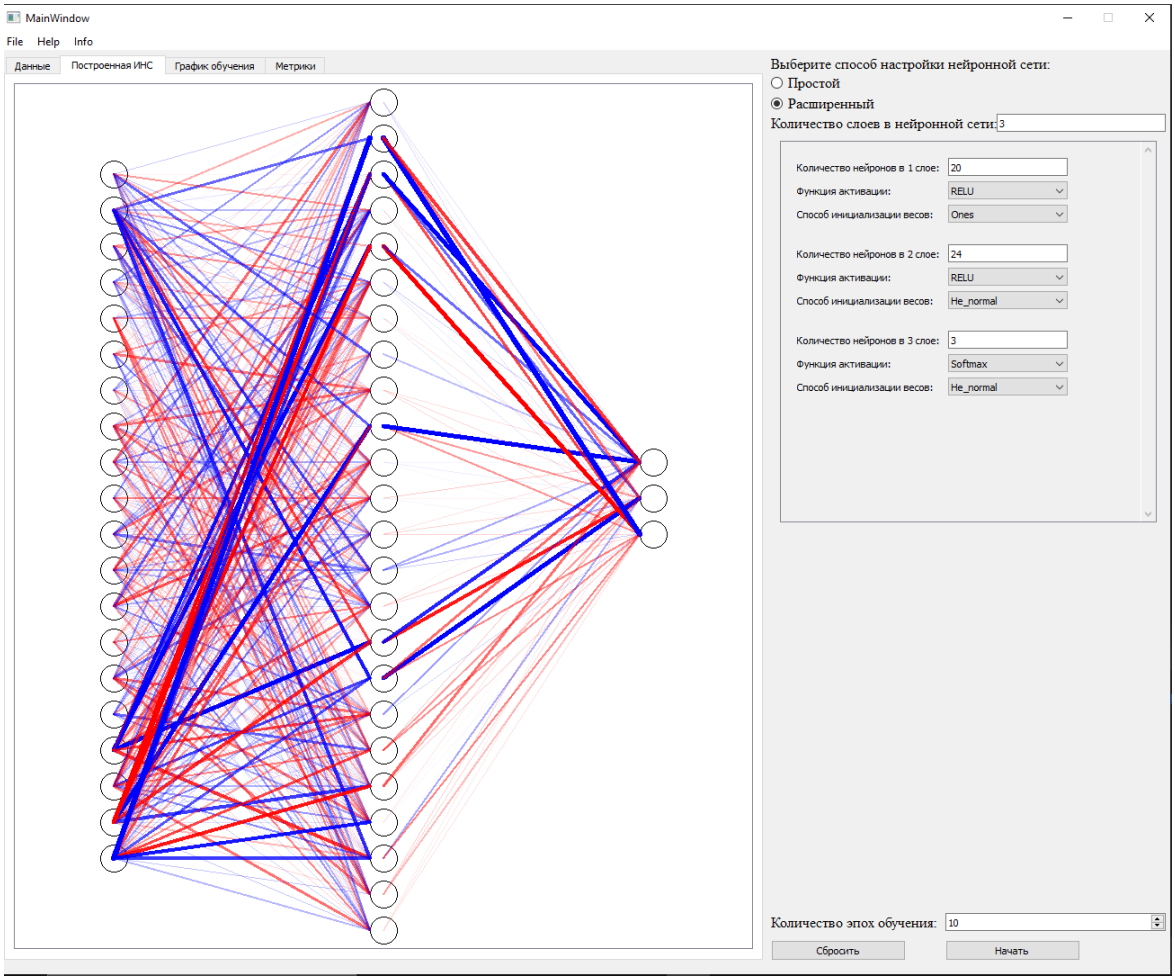


Рисунок 38 – Визуализация обученной ИНС

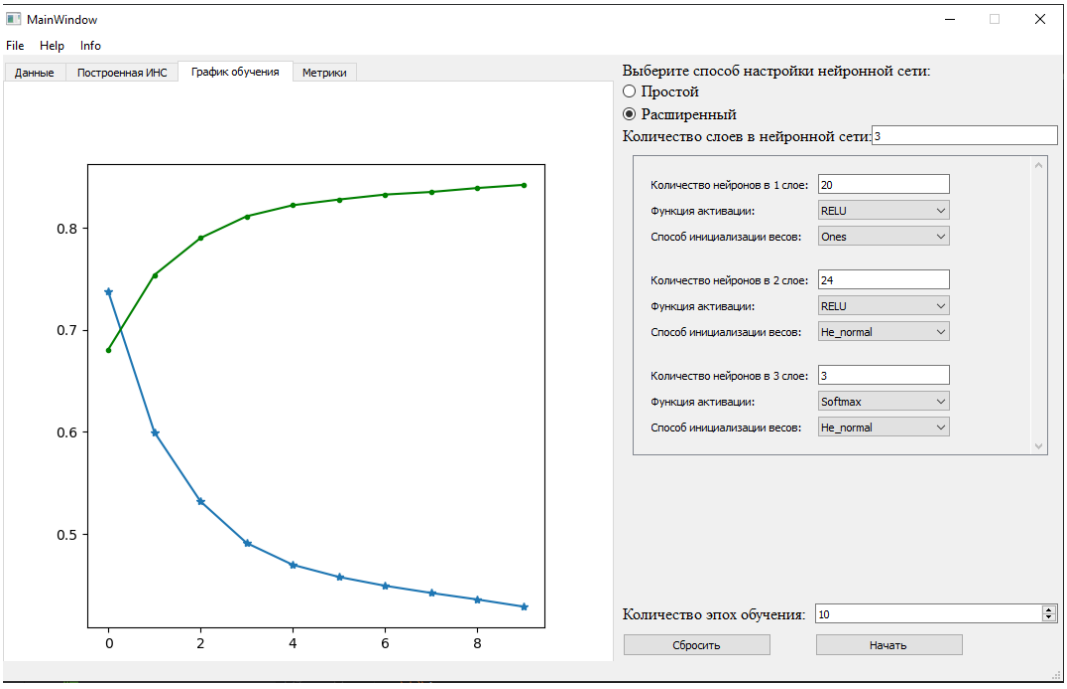


Рисунок 39 – График обучения

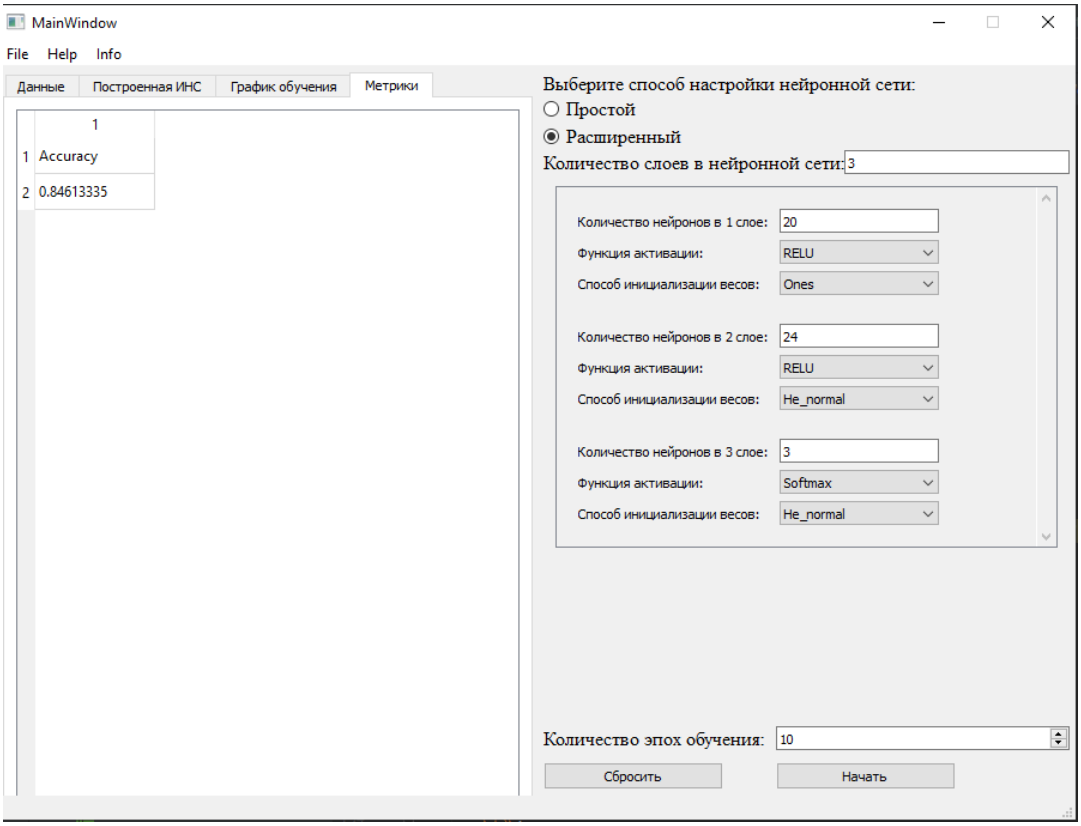


Рисунок 40 – Значение метрики

Как видно из рисунков 37 – 40 ИНС имеет отличие лишь в способе инициализации весов во втором слое, остальные параметры идентичны ИНС из предыдущего эксперимента. Этот факт дает возможность сравнить, какой способ инициализации весов лучше.

Сравним графики обучения, представленные на рисунках 34 и 39, из них видно, что график обучения при использовании *SVD* имеет больший наклон, это говорит о том, что возможно дальнейшее обучение сети для улучшения ее результатов. Кроме этого, скорость сходимости ИНС высока. А график обучения на рисунке 39 более пологий, что говорит о приближении к пределу значений точности, а дальнейшее обучение может вызвать проблему переобучения.

Значения метрики *Accuracy* 0,845 и 0,846 дают понять, что обе ИНС способны решать поставленную задачу с высокой точность. Однако исходя из анализов графиков обучения видно, что ИНС с *SVD* способна улучшить свои результаты.

Аналогичными способами проведем эксперименты с различными архитектурами и способами инициализации весовых коэффициентов, значения

метрики *Accuracy* запишем в таблицу. Результаты экспериментов представлены в таблице 9.

Таблица 9 – Результаты проведения экспериментов

Номер эксперимента	ИНС с одним скрытым слоем			ИНС с двумя скрытыми слоями		
	SVD	*He normal*	*Lecun normal*	*SVD*	*He normal*	*Lecun normal*
1	0,748	0,755	0,767	0,766	0,767	0,774
2	0,855	0,869	0,863	0,860	0,868	0,877
3	0,834	0,839	0,839	0,846	0,858	0,858
4	0,922	0,925	0,926	0,928	0,925	0,929
5	0,818	0,825	0,838	0,855	0,868	0,869
6	0,841	0,842	0,848	0,855	0,870	0,870
7	0,835	0,826	0,838	0,850	0,848	0,860
8	0,893	0,893	0,902	0,920	0,906	0,921
9	0,807	0,817	0,823	0,827	0,849	0,857
10	0,853	0,865	0,867	0,875	0,888	0,896

Следует отметить, что данные для обучения были сгенерированы посредством библиотеки *scikit-learn* с помощью метода *make_classification*, который генерирует кластеры точек с нормальным распределением с заданным количеством кластеров и вводит зависимости между этими данными и добавляет в них шум. Кроме этого, есть возможность указать количество наиболее информативных параметров. В генерируемом наборе данных имеется 20 атрибутов, в которых три наиболее информативны. Однако в рамках одного эксперимента данные не изменялись.

Исходя из результатов экспериментов видно, что наилучшие показатели имеет ИНС с двумя скрытыми слоями и способом инициализации весовых коэффициентов *Lecun normal*. Однако, инициализатор, основанный на *SVD* имеет показатели близкие к лучшим, а разница между средними показателями точности

всего 0,02. Графики обучения, демонстрирующие зависимость точности от количества эпох, полученные в ходе экспериментов, приведены в приложении Б.

Из графиков обучения видно, что ИНС, использующая *SVD* в качестве инициализатора весовых коэффициентов, с дополнительным скрытым слоем показывает результаты соизмеримые с уже зарекомендовавшими себя способами инициализации весов. Однако количество экспериментов слишком мало для конечного вывода о пригодности данного метода, но имеющиеся на данном этапе результаты говорят о том, что целесообразно провести большее число экспериментов.

Исходя из всего ранее сказанного можно сделать вывод: способ инициализации весовых коэффициентов, основанный на *SVD* можно использовать для обучения ИНС, так как обученная ИНС достаточно точно классифицирует данные, однако судить об увеличении скорости сходимости ИНС, на данной стадии экспериментов, нельзя.

ЗАКЛЮЧЕНИЕ

В процессе выполнения выпускной квалификационной работы были решены следующие задачи:

– проведен аналитический обзор существующих искусственных нейронных сетей, подходов к выбору начального значения весов и соответствующих инструментов разработки, в результате были выбраны наиболее популярные виды ИНС, основные подходы к выбору начального значения весов и средства разработки;

– разработана объектно-ориентированная архитектура ИС автоматизации вычислительных экспериментов с ИНС, модернизированных посредством предикторов весовых коэффициентов, отличающаяся относительной простотой понимания, гибкостью и наличием возможности дальнейшего расширения функционала, благодаря применению паттернов программирования *MVC*, *Template method*;

– реализована спроектированная информационная система, посредством языка *Python*, с использованием библиотек *Tensorflow*, *Keras*, *numpy*, *PyQt*;

– разработаны элементы технической документации и произведено тестирование ИС, продемонстрировавшее наглядность визуализации архитектуры ИНС, легкость создания и сравнения ИНС.

– произведена проверка гипотезы о влиянии сингулярных чисел на скорость обучения, в результате чего показано, что в ИНС с одним и двумя скрытыми слоями, использование предикторов на основе *SVD*-разложения, снижает точность в среднем на 0,02. Ограничением к результатам исследования является число экспериментов в количестве 10 и тестовый характер обучающей выборки.

В результате выполнения выпускной квалификационной работы было достигнуто снижение трудовых и временных затрат на постановку вычислительных экспериментов по повышению сходимости ИНС, модернизированных посредством предикторов весовых коэффициентов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Аггарвал, Ч. Нейронные сети и глубокое обучение. Учебный курс / Ч. Аггарвал. – М.: Диалектика, 2020. – 754 с.

2. Бринк, Х. Машинное обучение / Х. Бринк, Д. Ричардсон, Ф. Марк – СПб.: Питер, 2017. – 336 с.

3. Босуэлл, Д. Читаемый код, или Программирование как искусство / Д. Босуэлл, Т. Фаучер. – СПб.: Питер, 2012. – 208 с.

4. Гудфеллоу, Я. Глубокое обучение / Я. Гудфеллоу, И. Бенджио, А. Курвиль. – М.: ДМК, 2018. – 653 с.

5. *Ru.Qwe.Wiki* [Электронный ресурс] – Режим доступа: *https://ru.qwe.wiki/wiki/Singular_value_decomposition*. – Дата обращения: 10.05.2020.

6. Вержбицкий, В.М. Вычислительная линейная алгебра / В.М. Вержбицкий. – М.: Высшая школа, 2009. – 351 с.

7. Леоненков, А.В. Самоучитель *UML* / А. В. Леоненков. – 2-е изд – СПб.: БХВ, 2004. – 569 с.

8. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. – М.: Питер, 2018. – 366 с.

9. Эккель, Б. Философия программирования / Б. Эккель. − СПб.: Питер, 2015. – 1168 с.

10. Методы программирования: учебное пособие / Ю.Ю. Громов [и др.]. – Тамбов: Изд-во ФГБОУ ВПО «ТГТУ», 2012. – 144 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг исходного кода

main.py

```
import sys

from PyQt5.QtWidgets import QApplication
from View.NeuralNetworkView import NeuralNetworkView
from Controller.NeuralNetworkController import NeuralNetworkController, NeuralNetworkModel


def main():
    app = QApplication(sys.argv)
    # Инициализация контроллера с заданием модели и представления
    NeuralNetworkController(model=NeuralNetworkModel(), view=NeuralNetworkView())
    app.exec()


if __name__ == '__main__':
    sys.exit(main())
```

Controller.py

```
from abc import ABCMeta, abstractmethod, abstractproperty

from Model import config


class Controller():
    __metaclass__ = ABCMeta

    @abstractmethod
    def createView(self):
        "Создание View и Model под конкретный тип задачи"
        pass

    @abstractmethod
    def setModelParams(self,layer_count,neuron_counter,activation_function,kernel_init):
        pass

    @abstractmethod
    def setData(self,X,y):
        pass

    @abstractmethod
    def fitModel(self, config: config):
        print("Oppa")
```

```
        pass
```

NeuralNetworkController.py

```
from abc import ABC

from PyQt5.QtCore import QRunnable, pyqtSlot, QThreadPool, pyqtSignal, QObject
from sklearn.datasets import make_classification
from Model.config import TrainingConfig
from View.NeuralNetworkView import *
from Model.NeuralNetworkModel import *
from Controller.Controller import *
import re


class NeuralNetworkController(Controller, ABC, Callback):
    struct = None
    flag = False
    counter = 0

    def __init__(self, model: NeuralNetworkModel, view: NeuralNetworkView):
        super().__init__()
        self.view = view
        self.neural_model = model
        self.view.set_on_click_listener(lambda: self.on_start_button_click())
        self.view.set_save_click_listener(lambda: self.on_save_button_click())
        self.view.set_on_load_click_listener(lambda: self.on_load_button_click())
        self.view.set_on_validation_listener(lambda: self.on_validation())
        self.view.set_on_clear_button_click_listener(lambda: self.on_clear_button_listner())
        self.view.show()
        self.neural_model.after_epochs_end_callback = self
        self.thread_pool = QThreadPool()

    def on_start_button_click(self):
        dataset = self.view.ui.get_data()
        X = dataset.drop("predict", axis=1)
        X = X.to_numpy()
        y = dataset['predict'].to_numpy()
        try:
            self.neural_model.set_data_for_learning(inputArray=X, realClass=y)
            config = self.view.get_config()
            layer_count = int(self.view.ui.LayerCountLineEdit.text())
            neuron_counter = list()
            activation_func = list()
            kernel = list()
            for x, y, z in config:
                neuron_counter.append(int(x))
                activation_func.append(y)
                kernel.append(z)
            self.neural_model.set_ANN_params(layer_count=layer_count,
```

```
                                neuron_counter=neuron_counter,
                                activation_function=activation_func,
                                kernel_init=kernel)
            train_config = TrainingConfig(
                epochs=int(self.view.main_window.epoch_SpBox.text()),
                loss_func="sparse_categorical_crossentropy",
                metrics=["accuracy"],
                optimizer="sgd", )
            worker = Worker(self.neural_model, self.view)
            self.neural_model.set_train_config(train_config)
            self.thread_pool.start(worker)
        except ValueError as error:
            self.view.error_value_handler(error.args[0]+str(error.args[1])+error.args[2])

    def on_load_button_click(self):
        self.neural_model.load_model(path=
                                     str(self.view.main_window.ann_loader.path_line_edit.text()))
        self.view.main_window.ann_loader.close()
        weights = list()
        for layer in range(np.size(self.neural_model.model.layers)):
            weights.append(self.neural_model.model.get_layer(index=layer).get_weights())
        self.view.draw_ANN_architecture(weights_model=weights)

    def on_save_button_click(self):
        self.neural_model.save_model(path=
                                     str(self.view.main_window.save_wgt.path_line_edit.text()))
        self.view.main_window.save_wgt.close()

    def on_train_end(self, logs={}):
        weights = list()
        for layer in range(np.size(self.model.layers)):
            weights.append(self.model.get_layer(index=layer).get_weights())
        self.view.draw_ANN_architecture(weights_model=weights)

    def on_epoch_end(self, batch, logs={}):
        weights = list()
        if self.counter == 2:
            for layer in range(np.size(self.model.layers)):
                weights.append(self.model.get_layer(index=layer).get_weights())
            self.view.draw_ANN_architecture(weights_model=weights)
            self.counter = 0
            self.struct = [self.model.get_layer(index=0).get_weights(),
self.model.get_layer(index=1).get_weights()]
            self.flag = True
        else:
            self.counter = self.counter + 1

    def on_validation(self):
        match = re.search('^[1-9]+', self.view.main_window.LayerCountLineEdit.text())
```

```
        if match:
            pass
        else:
            self.view.error_value_handler("Неверно указано количество слоев ИНС.")

    def on_clear_button_listner(self):
        self.__init__(model=NeuralNetworkModel(), view=NeuralNetworkView())

class Worker(QRunnable):

    def __init__(self, model: NeuralNetworkModel, view: NeuralNetworkView):
        super().__init__()
        self.neural_model = model
        self.view = view
        self.setAutoDelete(True)
        self.signals = WorkerSignals()

    @pyqtSlot()
    def run(self):
        try:
            history = self.neural_model.train_ANN().history
            metrics = self.neural_model.get_metrics()
            self.view.plot(data_loss=history["loss"], data_acc=history["accuracy"])
            self.view.metrics_vizualization(metric=metrics)
        finally:
            self.signals.finished.emit()


class WorkerSignals(QObject):
    finished = pyqtSignal()
    error = pyqtSignal(tuple)
    result = pyqtSignal(object)
```

config.py

```
from dataclasses import dataclass


@dataclass
class TrainingConfig:
    epochs: int
    loss_func: str
    metrics: list
    optimizer: str
```

NeuralNetworkModel.py

```
import matplotlib as mpl
import matplotlib.pyplot as pltG
import numpy as np
import tensorflow as tf
```

```
from keras.callbacks import Callback
from tensorflow import keras
from sklearn.cluster import KMeans
from svd import *
from Model.WeightsCallback import *
from svd.svd import *


class NeuralNetworkModel:
    after_epochs_end_callback = Callback

    def __init__(self):
        self.model = None
        self.info = None
        self.activation_function_enum = {
            "RELU": keras.activations.relu,
            "Sigmoid": keras.activations.sigmoid,
            "ELU": keras.activations.elu,
            "Tangh": keras.activations.tanh,
            "Softmax": keras.activations.softmax,
            '': None
        }
        self.initializer_func_enum = {
            "Zeros": keras.initializers.Zeros,
            "Ones": keras.initializers.Ones,
            "RandomNormal": keras.initializers.RandomNormal,
            "RandomUniform": keras.initializers.RandomUniform,
            "TruncatedNormal": keras.initializers.TruncatedNormal,
            "Lecun_uniform": keras.initializers.lecun_uniform,
            "He_normal": 'he_normal',
            "Lecun_normal": 'lecun_normal',
            "He_uniform": keras.initializers.he_uniform,
            "SVD": self.SVD,
            '': None
        }

    def train_ANN(self):
        if self.model is None:
            self.__set_layer_params__()
        self.info = self.__learn_model__()
        return self.info

    def __set_layer_params__(self):
        self.model = keras.models.Sequential()
        i = 1
        self.model.add(keras.layers.Input(self.inputArray.shape[1]))
        while i != self.layer_count:
            self.model.add(keras.layers.Dense(self.neuron_in_layers[i],
                                              activation= self.activation_function[i],
```

```
                                        kernel_initializer=self.kernel_initializer_func[i]))
        i = i + 1

    def __learn_model__(self):
        self.model.compile(loss=self.loss, optimizer=self.optimizer, metrics=self.metrics)
        info = self.model.fit(self.inputArray, self.realClass, epochs=self.epochs,
                              validation_split=0.1, callbacks=[self.after_epochs_end_callback])
        return info

    def SVD(self, shape, dtype=None):
        class_array = list()
        real_class_value = self.__class_finder__()
        clustering_classes = list()
        self.__classification_algorithm__(class_array, real_class_value)
        for i in range(np.size(real_class_value)):
            clustering_classes.append(self.__clustering__(data=class_array[i]))
        output_massive = list()
        for i in range(np.shape(self.inputArray)[1]):
            massive = list()
            for j in range(np.size(real_class_value)):
                massive += clustering_classes[j][i]
            output_massive.append(massive)
        return output_massive

    def __class_finder__(self):
        real_class_value = list()
        real_class_value.append(self.realClass[0])
        for i in range(np.size(self.realClass)):
            for j in range(np.size(real_class_value)):
                if self.realClass[i] in real_class_value:
                    pass
                else:
                    real_class_value.append(self.realClass[i])
        sorted(real_class_value)
        return real_class_value

    def __classification_algorithm__(self, array_of_classes, real_class_value):
        for i in range(np.size(real_class_value)):
            array_of_classes.append(list())
            for j in range(np.size(self.realClass)):
                if self.realClass[j] == real_class_value[i]:
                    array_of_classes[i].append(self.inputArray[j])

    def __clustering__(self, data):
        mass1 = list()
        mass2 = list()
        mass3 = list()
        mass4 = list()
        mass5 = list()
```

```
        mass6 = list()
        mass7 = list()
        mass8 = list()
        cluster = KMeans(random_state=42)
        outClusters = cluster.fit(data)
        for x, i in zip(data, outClusters.labels_):
            if i == 0:
                mass1.append(x)
            elif i == 1:
                mass2.append(x)
            elif i == 2:
                mass3.append(x)
            elif i == 3:
                mass4.append(x)
            elif i == 4:
                mass5.append(x)
            elif i == 5:
                mass6.append(x)
            elif i == 6:
                mass7.append(x)
            elif i == 7:
                mass8.append(x)
        u, s, v = svd(mass1)
        weights1, u, v = svd(mass1)
        weights2, u, v = svd(mass2)
        weights3, u, v = svd(mass3)
        weights4, u, v = svd(mass4)
        weights5, u, v = svd(mass5)
        weights6, u, v = svd(mass6)
        weights7, u, v = svd(mass7)
        weights8, u, v = svd(mass8)

        weights1 = (weights1 / max(weights1) - 0.5) / 0.5
        weights2 = (weights2 / max(weights2) - 0.5) / 0.5
        weights3 = (weights3 / max(weights3) - 0.5) / 0.5
        weights4 = (weights4 / max(weights4) - 0.5) / 0.5
        weights5 = (weights5 / max(weights5) - 0.5) / 0.5
        weights6 = (weights6 / max(weights6) - 0.5) / 0.5
        weights7 = (weights7 / max(weights7) - 0.5) / 0.5
        weights8 = (weights8 / max(weights8) - 0.5) / 0.5
        out = list()
        for x1, x2, x3, x4, x5, x6, x7, x8 in zip(weights1, weights2, weights3, weights4, weights5,
weights6, weights7, weights8):
            out.append(
                [np.float(x1), np.float(x2), np.float(x3), np.float(x4), np.float(x5), np.float(x6), np.float(x7),
                 np.float(x8)])
        return out

    def set_data_for_learning(self, inputArray, realClass):
```

```
        self.inputArray = inputArray
        self.realClass = realClass


    def set_train_config(self, train_config):
        self.__setEpochs__(train_config.epochs)
        self.__setLoss__(train_config.loss_func)
        self.__setOptimizer__(train_config.optimizer)
        self.__setMetrics__(train_config.metrics)

    def __setEpochs__(self, epoch):
        self.epochs = epoch

    def __setLoss__(self, loss_func):
        try:
            if type(loss_func) == str:
                self.loss = loss_func
            else:
                raise ValueError(loss_func)
        except ValueError as error:
            print("Неверно указанна функция потерь:", error)

    def __setOptimizer__(self, optimizer):
        try:
            if type(optimizer) == str:
                self.optimizer = optimizer
            else:
                raise ValueError(optimizer)
        except ValueError as error:
            print("Неверная функция активации:", error)

    def __setMetrics__(self, metric):
        # TODO Реализовать метод
        self.metrics = metric

    def set_ANN_params(self, layer_count, neuron_counter=[1], activation_function=["sigmoid"],
                                    kernel_init=["random_uniform"]):
        self.__setLayer_count__(layer_count)
        self.__set_neuron_in_layers__(neuron_counter)
        self.__set_kernel_initializer__(kernel_init)
        self.__setActivationFunc__(activation_function)

    def __setLayer_count__(self, layer_count):
        if layer_count > 0:
            self.layer_count = layer_count
        else:
            raise ValueError("Не верное количество слоев.", layer_count, "Пропробуйте указать число
больше 0.")
```

```
    def __set_neuron_in_layers__(self, neuron_count_in_layers):
        counter = np.array(neuron_count_in_layers, dtype=int)
        if counter.size == self.layer_count:
            self.neuron_in_layers = counter
        else:
            raise ValueError("Неверно указано колличество слоев в сети:", neuron_count_in_layers,
                             ". Количество слоев должно совпадать с указынным.")

    def __set_kernel_initializer__(self, kernel_init):
        kernel = np.array(kernel_init, dtype=str)
        self.kernel_initializer_func = []
        for init in kernel:
            try:
                self.kernel_initializer_func.append(self.initializer_func_enum[init])
            except KeyError as e:
                raise ValueError('Undefined unit: {}'.format(e.args[0]))

    def __setActivationFunc__(self, activation_function):
        func_array = np.array(activation_function, dtype=str)
        self.activation_function = []
        for func in func_array:
            try:
                self.activation_function.append(self.activation_function_enum[func])
            except KeyError as e:
                raise ValueError('Неверно указана функция активации: {}'.format(e.args[0]))

    def get_history(self):
        return [self.info.history['loss'], self.info.history['accuracy']]

    def get_metrics(self):
        data = self.model.evaluate(x=self.inputArray, y=self.realClass)[1]
        print(data)
        return data

    def save_model(self, path=None):
        self.model.save(path)

    def load_model(self, path=None):
        self.model = keras.models.load_model(path)
```

WeightsCallback.py

```
from tensorflow import keras
from Model.NeuralNetworkModel import *

class WeightsCallback(keras.callbacks.Callback):
    counter = 0
    flag = False
```

```
    struct = None

    def on_train_end(self, logs={}):
        self.struct = [self.model.get_layer(index=0).get_weights(),
                       self.model.get_layer(index=1).get_weights()]

    def on_epoch_end(self, batch, logs={}):
        if self.counter == 5:
            self.counter = 0
            self.struct = [self.model.get_layer(index=0).get_weights(),
                           self.model.get_layer(index=1).get_weights()]
            self.flag = True
        else:
            self.counter = self.counter + 1

    def get_struct(self):
        self.flag = False
        return self.struct
```

AdvancedSettingsWidget.py

```
from PyQt5 import QtWidgets, QtCore
from PyQt5.QtWidgets import QWidget
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QGridLayout, QScrollArea
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure
import matplotlib.pyplot as plt
from PyQt5.QtGui import QPainter, QPixmap, QPen, QColor, QFont
from PyQt5.QtCore import Qt
from PyQt5 import uic


class AdvancedSettingsWidget(QWidget):

    def __init__(self, parent=None, i=0):
        _translate = QtCore.QCoreApplication.translate
        QWidget.__init__(self, parent=parent)
        self.gridLayout = QtWidgets.QGridLayout()
        self.neuron_counter_label = QtWidgets.QLabel()
        self.neuron_counter_line = QtWidgets.QLineEdit()
        self.activation_func_label = QtWidgets.QLabel()
        self.activation_func_cmb = QtWidgets.QComboBox()
        self.kernel_init_label = QtWidgets.QLabel()
        self.kernel_init_cmb = QtWidgets.QComboBox()
        self.i = i
        row, col = 0, 0
        self.gridLayout.addWidget(self.neuron_counter_label, row, col)
        col += 1
        self.gridLayout.addWidget(self.neuron_counter_line, row, col)
```

```
        row += 1
        col = 0
        self.gridLayout.addWidget(self.activation_func_label, row, col)
        col += 1
        self.gridLayout.addWidget(self.activation_func_cmb, row, col)
        row += 1
        col = 0
        self.gridLayout.addWidget(self.kernel_init_label, row, col)
        col += 1
        self.gridLayout.addWidget(self.kernel_init_cmb, row, col)
        row += 1
        col = 0
        self.setLayout(self.gridLayout)
        self.initUi()

    def initUi(self):
        _translate = QtCore.QCoreApplication.translate
        self.setWindowTitle(_translate("AdvancedWgt", "Конфигурация слоя"))
        self.neuron_counter_label.setText("Количество нейронов в " + str(self.i + 1) + " слое: ")
        self.activation_func_cmb.addItem("")
        self.activation_func_cmb.addItem("RELU")
        self.activation_func_cmb.addItem("Sigmoid")
        self.activation_func_cmb.addItem("ELU")
        self.activation_func_cmb.addItem("Tangh")
        self.activation_func_cmb.addItem("Softmax")
        self.activation_func_label.setText("Функция активации: ")
        self.kernel_init_label.setText("Способ инициализации весов: ")
        self.kernel_init_cmb.addItem("")
        self.kernel_init_cmb.addItem("Zeros")
        self.kernel_init_cmb.addItem("Ones")
        self.kernel_init_cmb.addItem("RandomNormal")
        self.kernel_init_cmb.addItem("RandomUniform")
        self.kernel_init_cmb.addItem("TruncatedNormal")
        self.kernel_init_cmb.addItem("Lecun_uniform")
        self.kernel_init_cmb.addItem("He_normal")
        self.kernel_init_cmb.addItem("Lecun_normal")
        self.kernel_init_cmb.addItem("He_uniform")
        self.kernel_init_cmb.addItem("SVD")

    def get_fields(self):
        return [self.neuron_counter_line.text(), self.activation_func_cmb.currentText(),
                self.kernel_init_cmb.currentText()]
```

CustomPandasModel.py

```
from PyQt5 import QtCore
import pandas as pd

class DataFrameModel(QtCore.QAbstractTableModel):
    DtypeRole = QtCore.Qt.UserRole + 1000
```

```
    ValueRole = QtCore.Qt.UserRole + 1001

    def __init__(self, df=pd.DataFrame(), parent=None):
        super(DataFrameModel, self).__init__(parent)
        self._dataframe = df

    def setDataFrame(self, dataframe):
        self.beginResetModel()
        self._dataframe = dataframe.copy()
        self.endResetModel()

    def dataFrame(self):
        return self._dataframe

    dataFrame = QtCore.pyqtProperty(pd.DataFrame, fget=dataFrame, fset=setDataFrame)

    @QtCore.pyqtSlot(int, QtCore.Qt.Orientation, result=str)
    def headerData(self, section: int, orientation: QtCore.Qt.Orientation, role: int =
QtCore.Qt.DisplayRole):
        if role == QtCore.Qt.DisplayRole:
            if orientation == QtCore.Qt.Horizontal:
                return self._dataframe.columns[section]
            else:
                return str(self._dataframe.index[section])
        return QtCore.QVariant()

    def rowCount(self, parent=QtCore.QModelIndex()):
        if parent.isValid():
            return 0
        return len(self._dataframe.index)

    def columnCount(self, parent=QtCore.QModelIndex()):
        if parent.isValid():
            return 0
        return self._dataframe.columns.size

    def data(self, index, role=QtCore.Qt.DisplayRole):
        if not index.isValid() or not (0 <= index.row() < self.rowCount()
                                       and 0 <= index.column() < self.columnCount()):
            return QtCore.QVariant()
        row = self._dataframe.index[index.row()]
        col = self._dataframe.columns[index.column()]
        dt = self._dataframe[col].dtype

        val = self._dataframe.iloc[row][col]
        if role == QtCore.Qt.DisplayRole:
            return str(val)
        elif role == DataFrameModel.ValueRole:
            return val
```

```
        if role == DataFrameModel.DtypeRole:
            return dt
        return QtCore.QVariant()

    def roleNames(self):
        roles = {
            QtCore.Qt.DisplayRole: b'display',
            DataFrameModel.DtypeRole: b'dtype',
            DataFrameModel.ValueRole: b'value'
        }
        return roles
```

DialogFileWidget.py

```
from PyQt5 import QtWidgets, QtCore
from PyQt5.QtWidgets import QWidget
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QGridLayout, QScrollArea
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure
import matplotlib.pyplot as plt
from PyQt5.QtGui import QPainter, QPixmap, QPen, QColor, QFont
from PyQt5.QtCore import Qt, QDir, QModelIndex
from PyQt5 import uic
import pandas as pd


class DialogFileWidget(QWidget):

    def __init__(self, type=None, parent=None):
        _translate = QtCore.QCoreApplication.translate
        QWidget.__init__(self, parent=parent)
        self.central_layout = QtWidgets.QVBoxLayout()
        self.list_view = QtWidgets.QListView()
        self.path_label = QtWidgets.QLabel("Path: ")
        self.path_line_edit = QtWidgets.QLineEdit()
        self.open_button = QtWidgets.QPushButton("Ok")
        self.cancel_button = QtWidgets.QPushButton("Cancel")
        self.central_layout.addWidget(self.list_view)
        self.file_model = QtWidgets.QFileSystemModel()
        self.file_model.setFilter(QDir.AllEntries)
        self.file_model.setRootPath("")
        self.file_data = None
        self.list_view.setModel(self.file_model)
        self.type = type

        horizontal_layout = QtWidgets.QHBoxLayout()
        horizontal_layout.addWidget(self.path_label)
        horizontal_layout.addWidget(self.path_line_edit)
        self.central_layout.addLayout(horizontal_layout)
```

```
        horizontal_layout = QtWidgets.QHBoxLayout()
        horizontal_layout.addSpacerItem(QtWidgets.QSpacerItem(150, 0,
                                        QtWidgets.QSizePolicy.Expanding,
                                        QtWidgets.QSizePolicy.Minimum))
        horizontal_layout.addWidget(self.open_button)
        horizontal_layout.addWidget(self.cancel_button)
        self.central_layout.addLayout(horizontal_layout)

        self.list_view.doubleClicked.connect(self.double_click_slots)
        self.open_button.clicked.connect(self.open_button_clicked)
        self.cancel_button.clicked.connect(self.cancel_button_clicked)

        self.setLayout(self.central_layout)
        self.initUi()

    def initUi(self):
        _translate = QtCore.QCoreApplication.translate

    def double_click_slots(self, index):
        file_info = self.file_model.fileInfo(index)
        dir = file_info.absoluteDir()
        if file_info.fileName() == "..":
            dir.cdUp()
            self.list_view.setRootIndex(self.file_model.index(dir.absolutePath()))
        elif file_info.fileName() == ".":
            self.list_view.setRootIndex(self.file_model.index(""))
            self.path_line_edit.setText(str(dir.absolutePath()))
        elif file_info.isDir():
            self.list_view.setRootIndex(index)
            self.path_line_edit.setText(str(dir.absolutePath()))
        elif file_info.isFile():
            self.path_line_edit.setText(str(dir.absolutePath())+"/"+file_info.fileName())

    def open_button_clicked(self):
        if self.type == "data_loader":
            self.file_data = pd.read_csv(self.path_line_edit.text(), delimiter=';')
            self.close()
        elif self.type == 'saver':
            pass
        elif self.type == 'ann_loader':
            pass

    def cancel_button_clicked(self):
        self.close()
```

MainWindow.py

```
import numpy as np
import pandas as pd
```

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QGridLayout, QScrollArea, QVBoxLayout, QErrorMessage
from PyQt5.QtWidgets import QDialog
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure
import matplotlib.pyplot as plt
from PyQt5.QtGui import QPainter, QPixmap, QPen, QColor, QFont
from PyQt5.QtCore import Qt
from PyQt5 import uic
from View.AdvancedSettingsWidget import *
from View.DialogFileWidget import *
from View.CustomPandasModel import *


class Ui_MainWindow():

    def __init__(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1024, 768)
        MainWindow.setMinimumSize(QtCore.QSize(900, 650))
        MainWindow.setMaximumSize(QtCore.QSize(1920, 1080))
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding,
                                           QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.centralwidget.sizePolicy().hasHeightForWidth())
        self.centralwidget.setSizePolicy(sizePolicy)
        self.centralwidget.setMinimumSize(QtCore.QSize(800, 600))
        self.centralwidget.setMaximumSize(QtCore.QSize(1920, 1080))
        self.centralwidget.setAutoFillBackground(True)
        self.centralwidget.setObjectName("centralwidget")
        self.horizontalLayout_2 = QtWidgets.QHBoxLayout(self.centralwidget)
        self.horizontalLayout_2.setContentsMargins(5, 5, 5, 5)
        self.horizontalLayout_2.setSpacing(7)
        self.horizontalLayout_2.setObjectName("horizontalLayout_2")
        self.Info_Frame = QtWidgets.QTabWidget(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.MinimumExpanding,
                                           QtWidgets.QSizePolicy.MinimumExpanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.Info_Frame.sizePolicy().hasHeightForWidth())
        self.Info_Frame.setSizePolicy(sizePolicy)
        self.Info_Frame.setMinimumSize(QtCore.QSize(300, 650))
        self.Info_Frame.setObjectName("Info_Frame")
        self.data_tab = QtWidgets.QWidget()
        self.data_tab.setObjectName("data_tab")
        self.Info_Frame.addTab(self.data_tab, "")
        self.struct_tab = QtWidgets.QWidget()
```

```
self.struct_tab.setObjectName("struct_tab")
self.Info_Frame.addTab(self.struct_tab, "")
self.learning_tab = QtWidgets.QWidget()
self.learning_tab.setObjectName("learning_tab")
self.Info_Frame.addTab(self.learning_tab, "")
self.metrics_tab = QtWidgets.QWidget()
self.metrics_tab.setObjectName("metrics_tab")
self.Info_Frame.addTab(self.metrics_tab, "")
self.horizontalLayout_2.addWidget(self.Info_Frame)
self.verticalLayout = QtWidgets.QVBoxLayout()
self.verticalLayout.setSizeConstraint(QtWidgets.QLayout.SetFixedSize)
self.verticalLayout.setSpacing(0)
self.verticalLayout.setObjectName("verticalLayout")
self.settings_type_lbl = QtWidgets.QLabel(self.centralwidget)
self.settings_type_lbl.setMinimumSize(QtCore.QSize(160, 0))
font = QtGui.QFont()
font.setFamily("Times New Roman")
font.setPointSize(12)
self.settings_type_lbl.setFont(font)
self.settings_type_lbl.setObjectName("settings_type_lbl")
self.verticalLayout.addWidget(self.settings_type_lbl)
self.simple_rbtn = QtWidgets.QRadioButton(self.centralwidget)
self.simple_rbtn.setEnabled(True)
font = QtGui.QFont()
font.setFamily("Times New Roman")
font.setPointSize(12)
self.simple_rbtn.setFont(font)
self.simple_rbtn.setObjectName("simple_rbtn")
self.verticalLayout.addWidget(self.simple_rbtn)
self.advanced_rbtn = QtWidgets.QRadioButton(self.centralwidget)
font = QtGui.QFont()
font.setFamily("Times New Roman")
font.setPointSize(12)
self.advanced_rbtn.setFont(font)
self.advanced_rbtn.setObjectName("advanced_rbtn")
self.verticalLayout.addWidget(self.advanced_rbtn)
self.LayerCounter = QtWidgets.QGridLayout()
self.LayerCounter.setObjectName("LayerCounter")
self.LayerCountLineEdit = QtWidgets.QLineEdit(self.centralwidget)
self.LayerCountLineEdit.setObjectName("LayerCountLineEdit")
self.LayerCounter.addWidget(self.LayerCountLineEdit, 1, 1, 1, 1)
self.LayerCountLabel = QtWidgets.QLabel(self.centralwidget)
font = QtGui.QFont()
font.setFamily("Times New Roman")
font.setPointSize(12)
self.LayerCountLabel.setFont(font)
self.LayerCountLabel.setObjectName("LayerCountLabel")
self.LayerCounter.addWidget(self.LayerCountLabel, 1, 0, 1, 1)
self.verticalLayout.addLayout(self.LayerCounter)
```

```
self.frame = QtWidgets.QFrame(self.centralwidget)
self.frame.setEnabled(True)
self.frame.setFrameShape(QtWidgets.QFrame.StyledPanel)
self.frame.setFrameShadow(QtWidgets.QFrame.Raised)
self.frame.setObjectName("frame")
self.verticalLayout_2 = QtWidgets.QVBoxLayout(self.frame)
self.verticalLayout_2.setObjectName("verticalLayout_2")
self.AdvancedLayout = QtWidgets.QVBoxLayout()
self.AdvancedLayout.setObjectName("AdvancedLayout")
self.verticalLayout_2.addLayout(self.AdvancedLayout)
spacerItem = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum,
                                   QtWidgets.QSizePolicy.Expanding)
self.verticalLayout_2.addItem(spacerItem)
self.verticalLayout.addWidget(self.frame)
self.gridLayout_2 = QtWidgets.QGridLayout()
self.gridLayout_2.setSizeConstraint(QtWidgets.QLayout.SetDefaultConstraint)
self.gridLayout_2.setSpacing(10)
self.gridLayout_2.setObjectName("gridLayout_2")
self.epoch_lbl = QtWidgets.QLabel(self.centralwidget)
font = QtGui.QFont()
font.setFamily("Times New Roman")
font.setPointSize(12)
self.epoch_lbl.setFont(font)
self.epoch_lbl.setObjectName("epoch_lbl")
self.gridLayout_2.addWidget(self.epoch_lbl, 0, 0, 1, 1)
self.Start_Button = QtWidgets.QPushButton(self.centralwidget)
self.Start_Button.setMaximumSize(QtCore.QSize(150, 16777215))
self.Start_Button.setObjectName("Start_Button")
self.gridLayout_2.addWidget(self.Start_Button, 1, 1, 1, 1)
self.epoch_SpBox = QtWidgets.QSpinBox(self.centralwidget)
self.epoch_SpBox.setMinimumSize(QtCore.QSize(245, 0))
self.epoch_SpBox.setMaximumSize(QtCore.QSize(100, 16777215))
self.epoch_SpBox.setObjectName("epoch_SpBox")
self.gridLayout_2.addWidget(self.epoch_SpBox, 0, 1, 1, 1)
self.Clear_Button = QtWidgets.QPushButton(self.centralwidget)
self.Clear_Button.setMaximumSize(QtCore.QSize(150, 16777215))
self.Clear_Button.setObjectName("Clear_Button")
self.gridLayout_2.addWidget(self.Clear_Button, 1, 0, 1, 1)
self.verticalLayout.addLayout(self.gridLayout_2)
self.horizontalLayout_2.addLayout(self.verticalLayout)
self.horizontalLayout_2.setStretch(0, 2)
self.horizontalLayout_2.setStretch(1, 1)
MainWindow.setCentralWidget(self.centralwidget)
self.menubar = QtWidgets.QMenuBar(MainWindow)
self.menubar.setGeometry(QtCore.QRect(0, 0, 1199, 26))
self.menubar.setObjectName("menubar")
self.menuFile = QtWidgets.QMenu(self.menubar)
self.menuFile.setObjectName("menuFile")
self.menuHelp = QtWidgets.QMenu(self.menubar)
```

```
self.menuHelp.setObjectName("menuHelp")
self.menuInfo = QtWidgets.QMenu(self.menubar)
self.menuInfo.setObjectName("menuInfo")
MainWindow.setMenuBar(self.menubar)
self.statusbar = QtWidgets.QStatusBar(MainWindow)
self.statusbar.setObjectName("statusbar")
MainWindow.setStatusBar(self.statusbar)
self.actionOpen = QtWidgets.QAction(MainWindow)
self.actionOpen.setObjectName("actionOpen")
self.actionNew = QtWidgets.QAction(MainWindow)
self.actionNew.setObjectName("actionNew")
self.action_2 = QtWidgets.QAction(MainWindow)
self.action_2.setObjectName("action_2")
self.actionSave_NN = QtWidgets.QAction(MainWindow)
self.actionSave_NN.setObjectName("actionSave_NN")
self.actionLoad_ANN = QtWidgets.QAction(MainWindow)
self.actionLoad_ANN.setObjectName("actionLoad_ANN")
self.menuFile.addAction(self.actionNew)
self.menuFile.addAction(self.actionOpen)
self.menuFile.addAction(self.actionSave_NN)
self.menuFile.addAction(self.actionLoad_ANN)
self.menuInfo.addSeparator()
self.menubar.addAction(self.menuFile.menuAction())
self.menubar.addAction(self.menuHelp.menuAction())
self.menubar.addAction(self.menuInfo.menuAction())
self.retranslateUi(MainWindow)
self.Info_Frame.setCurrentIndex(0)
self.simple_rbtn.setChecked(True)
self.pen_style = QPen()
self.pen_style.setStyle(Qt.SolidLine)
self.pen_style.setWidth(1)
self.pen_style.setCapStyle(Qt.RoundCap)
self.figure = plt.figure()
self.canvas = FigureCanvas(self.figure)
self.training_graph_layout = QtWidgets.QVBoxLayout()
self.training_graph_layout.addWidget(self.canvas)
self.learning_tab.setLayout(self.training_graph_layout)
self.scene = QtWidgets.QGraphicsScene()
self.graphic = QtWidgets.QGraphicsView(self.scene)
self.ANNlayout = QtWidgets.QVBoxLayout()
self.ANNlayout.addWidget(self.graphic)
self.struct_tab.setLayout(self.ANNlayout)
self.scroll_area = QScrollArea()
self.current_layers = "0"
self.save_wgt = DialogFileWidget('saver')
self.ann_loader = DialogFileWidget(type='ann_loader')
self.advanced_rbtn.toggled['bool'].connect(self.form_for_setting)
self.actionOpen.triggered.connect(self.load_datasets)
self.actionLoad_ANN.triggered.connect(self.loadANN)
```

```
        self.actionSave_NN.triggered.connect(self.save_ANN)
        self.metrics = QtWidgets.QTableWidget()
        self.metrics_layout = QVBoxLayout()
        self.metrics_layout.addWidget(self.metrics)
        self.metrics_tab.setLayout(self.metrics_layout)
        self.metrics.setEditTriggers(QtWidgets.QAbstractItemView.NoEditTriggers)
        self.table_view = QtWidgets.QTableView()
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def loadANN(self):
        self.ann_loader.show()

    def save_ANN(self):
        self.save_wgt.show()

    def dialog_window_error_value(self, error_message):
        dialog_window = QErrorMessage(self.centralwidget)
        dialog_window.setModal(True)
        dialog_window.showMessage(error_message)

    def load_datasets(self):
        self.file_widget = DialogFileWidget(type='data_loader')
        self.file_widget.show()
        self.file_widget.open_button.clicked.connect(self.display_data)

    def display_data(self):
        model = DataFrameModel(self.file_widget.file_data)
        layout = QtWidgets.QHBoxLayout()
        self.table_view.setModel(model)
        layout.addWidget(self.table_view)
        self.data_tab.setLayout(layout)

    def get_data(self):
        return self.file_widget.file_data

    def form_for_setting(self):
        if self.AdvancedLayout.isEmpty() or not (int(self.LayerCountLineEdit.text()) ==
int(self.current_layers)):
            self.current_layers = self.LayerCountLineEdit.text()
            self.layer_list = list()
            self.scroll_area.clearFocus()
            scroll_layout = QVBoxLayout()
            wgt1 = QWidget()
            for i in range(int(self.LayerCountLineEdit.text())):
                wgt = AdvancedSettingsWidget(self.centralwidget, i)
                self.layer_list.append(wgt)
                scroll_layout.addWidget(self.layer_list[i])
            wgt1.setLayout(scroll_layout)
            self.scroll_area.setWidget(wgt1)
```

```
        self.scroll_area.setVerticalScrollBarPolicy(Qt.ScrollBarAlwaysOn)
        self.AdvancedLayout.addWidget(self.scroll_area)

    def write_metrics(self, data):
        self.metrics.setRowCount(2)
        self.metrics.setColumnCount(1)
        newItem = QtWidgets.QTableWidgetItem("Accuracy")
        self.metrics.setItem(0, 0, newItem)
        newItem = QtWidgets.QTableWidgetItem(str(data))
        self.metrics.setItem(1, 0, newItem)

    def plot_history(self, data_loss, data_acc):
        self.figure.clear()
        ax = self.figure.add_subplot(111)
        ax.plot(data_loss, '*-', label="Loss")
        ax.plot(data_acc, '.-g', label="Accuracy")
        ax.set_xlabel('Epoch')
        ax.grid(True)
        ax.legend()
        self.canvas.draw()

    def draw_model(self, weights):
        self.scene = QtWidgets.QGraphicsScene()
        self.graphic.setScene(self.scene)
        line = QtCore.QLineF()
        neuron_in_layer = list()  # Массив количества нейронов в слоях
        neuron_in_layer.append(weights[0][0].shape[0])
        for x in weights:
            neuron_in_layer.append(x[0].shape[1])
            # возвращает количество нейронов во входном слое
        x_circle = 0
        y_circle = 0
        diameter = 30
        for i in range(np.size(neuron_in_layer)):
            for j in range(neuron_in_layer[i]):
                self.scene.addEllipse(x_circle, y_circle, diameter, diameter)
                if y_circle > 0:
                    y_circle = y_circle * -1
                elif y_circle <= 0:
                    y_circle = y_circle * -1
                    y_circle += 40
            x_circle += 300
            y_circle = 0
        line.setLine(15, 15, 300, 15)
        color_positive = QColor(255, 0, 0)
        color_negative = QColor(0, 0, 255)
        for i in range(np.size(neuron_in_layer) - 1):  # для количества слоев -1
            for j in range(neuron_in_layer[i]):  # для киоличества нейронов в слое
                for k in range(neuron_in_layer[i + 1]):
```

```
                if weights[i][0][j][k] > 0:
                    color_positive.setAlphaF(weights[i][0][j][k])
                    self.pen_style.setColor(color_positive)
                    self.pen_style.setWidth(int(weights[i][0][j][k] * (5 - 1) + 1))
                elif weights[i][0][j][k] <= 0:
                    self.pen_style.setWidth(1)
                    color_negative.setAlphaF(weights[i][0][j][k] * -1)
                    self.pen_style.setColor(color_negative)
                    self.pen_style.setWidth(int(weights[i][0][j][k] * -1 * (5 - 1) + 1))
                self.scene.addLine(line, pen=self.pen_style)
                if line.y2() > 15:
                    line.setLine(line.x1(), line.y1(), line.x2(), line.y2() * -1 + 30)
                elif line.y2() < 15:
                    line.setLine(line.x1(), line.y1(), line.x2(), line.y2() * -1 + 70)
                elif line.y2() == 15:
                    line.setLine(line.x1(), line.y1(), line.x2(), line.y2() + 40)
            line.setLine(line.x1(), line.y1(), line.x2(), 15)
            if line.y1() > 15:
                line.setLine(line.x1(), line.y1() * -1 + 30, line.x2(), line.y2())
            elif line.y1() < 15:
                line.setLine(line.x1(), line.y1() * -1 + 70, line.x2(), line.y2())
            elif line.y1() == 15:
                line.setLine(line.x1(), line.y1() + 40, line.x2(), line.y2())
        line.setLine(line.x1() + 300, 15, line.x2() + 300, line.y2())

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.Info_Frame.setTabText(self.Info_Frame.indexOf(self.data_tab), _translate("MainWindow",
"Данные"))
        self.Info_Frame.setTabText(self.Info_Frame.indexOf(self.struct_tab),
                                   _translate("MainWindow", "Построенная ИНС"))
        self.Info_Frame.setTabText(self.Info_Frame.indexOf(self.learning_tab),
                                   _translate("MainWindow", "График обучения"))
        self.Info_Frame.setTabText(self.Info_Frame.indexOf(self.metrics_tab),
_translate("MainWindow", "Метрики"))
        self.settings_type_lbl.setText(_translate("MainWindow", "Выберите способ настройки
нейронной сети: "))
        self.simple_rbtn.setText(_translate("MainWindow", "Простой"))
        self.advanced_rbtn.setText(_translate("MainWindow", "Расширенный"))
        self.LayerCountLabel.setText(_translate("MainWindow", "Количество слоев в нейронной
сети:"))
        self.epoch_lbl.setText(_translate("MainWindow", "Количество эпох обучения:"))
        self.Start_Button.setText(_translate("MainWindow", "Начать"))
        self.Clear_Button.setText(_translate("MainWindow", "Сбросить"))
        self.menuFile.setTitle(_translate("MainWindow", "File"))
        self.menuHelp.setTitle(_translate("MainWindow", "Help"))
        self.menuInfo.setTitle(_translate("MainWindow", "Info"))
        self.actionOpen.setText(_translate("MainWindow", "Load dataset"))
```

```
        self.actionNew.setText(_translate("MainWindow", "New"))
        self.action_2.setText(_translate("MainWindow", "Создал Хрущев Илья"))
        self.actionSave_NN.setText(_translate("MainWindow", "Save ANN"))
        self.actionLoad_ANN.setText(_translate("MainWindow", "Load ANN "))
```

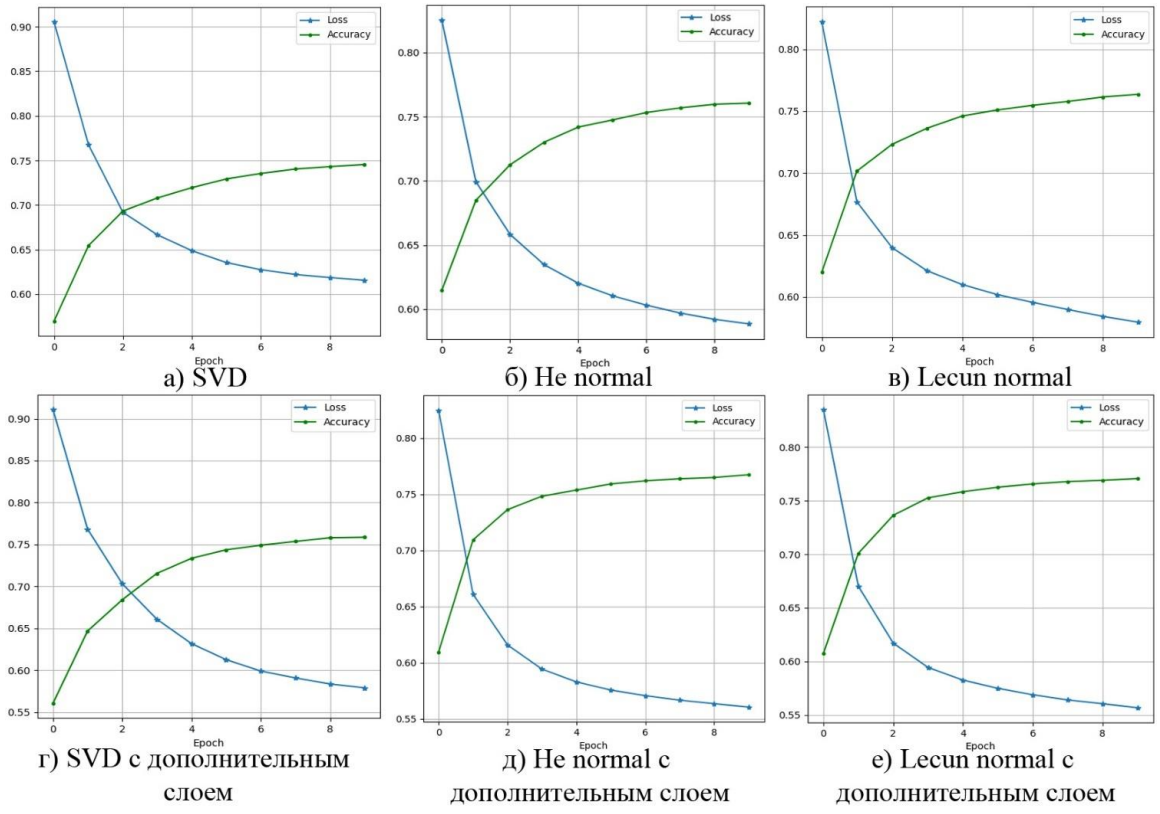
NeuralNetworkView.py

```
from View.MainWindow import *


class NeuralNetworkView(QtWidgets.QMainWindow):

    def __init__(self):
        super(NeuralNetworkView, self).__init__()
        self.main_window = Ui_MainWindow(self)

    def plot(self, data_loss, data_acc):
        self.main_window.plot_history(data_loss=data_loss, data_acc=data_acc)

    def draw_ANN_architecture(self, weights_model):
        self.main_window.draw_model(weights=weights_model)

    def metrics_vizualization(self, metric):
        self.main_window.write_metrics(data=metric)

    def set_on_click_listener(self, action):
        self.main_window.Start_Button.clicked.connect(action)

    def set_save_click_listener(self, action):
        self.main_window.save_wgt.open_button.clicked.connect(action)

    def set_on_load_click_listener(self, action):
        self.main_window.ann_loader.open_button.clicked.connect(action)

    def set_on_clear_button_click_listener(self, action):
        self.main_window.Clear_Button.clicked.connect(action)

    def set_on_validation_listener(self, action):
        self.main_window.epoch_SpBox.editingFinished.connect(action)
        self.main_window.LayerCountLineEdit.editingFinished.connect(action)

    def error_value_handler(self, message):
        self.main_window.dialog_window_error_value(message)
```

ПРИЛОЖЕНИЕ Б

(справочное)

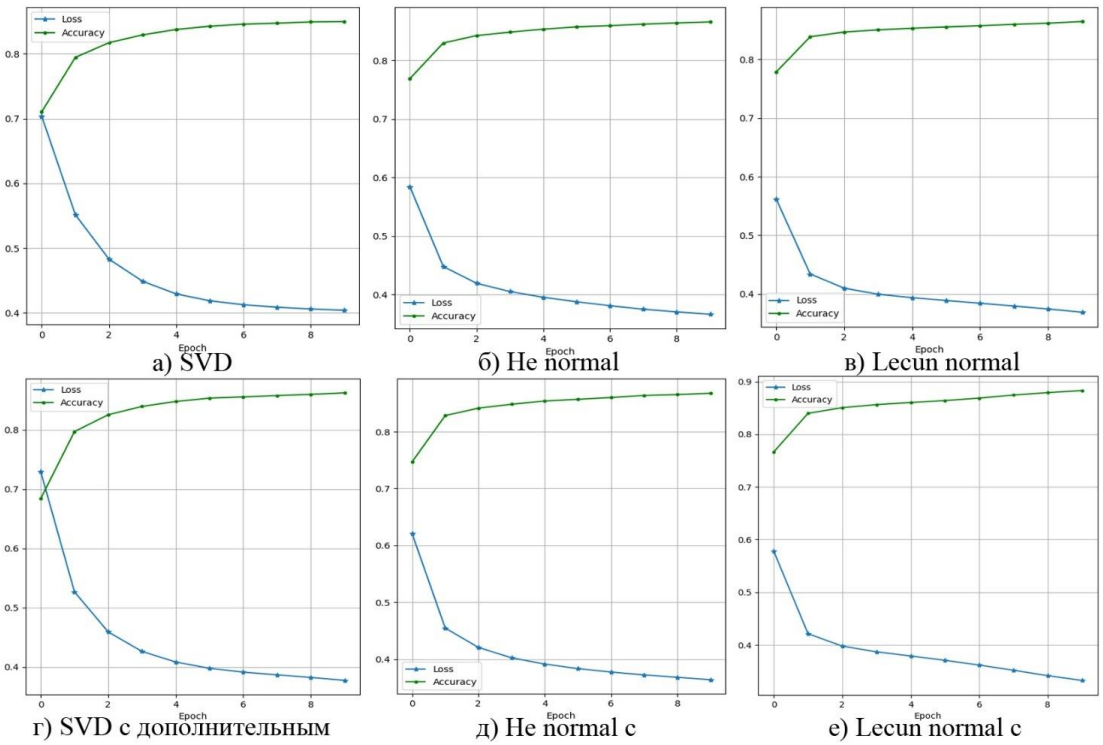
Графики обучения



а) SVD б) He normal в) Lecun normal

г) SVD с дополнительным слоем д) He normal с дополнительным слоем е) Lecun normal с дополнительным слоем

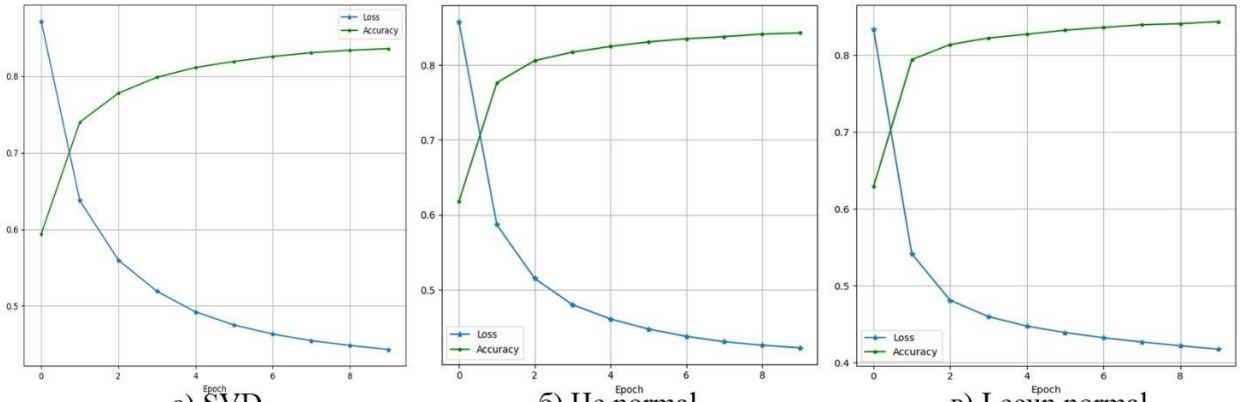
Рисунок Б.1 – Графики обучения, полученные в ходе первого эксперимента



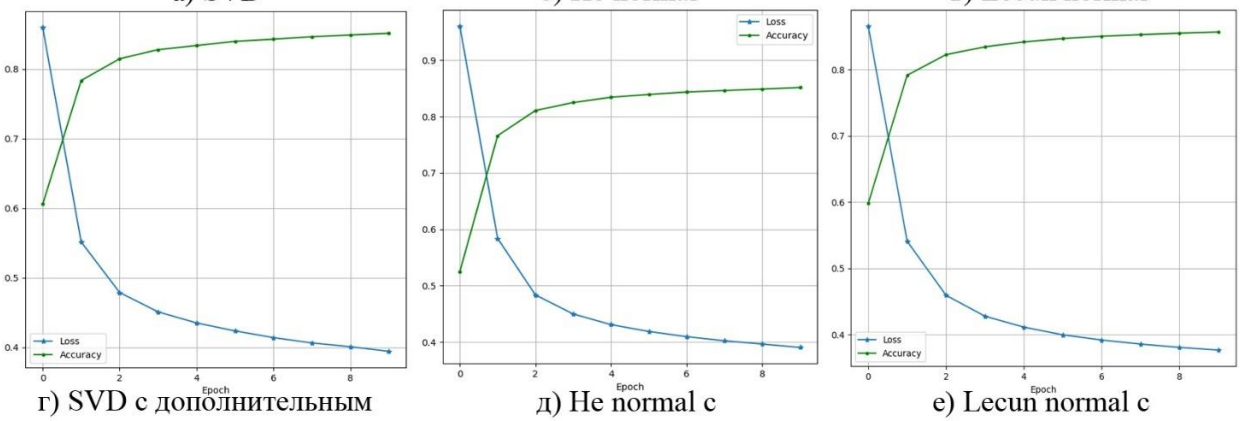
а) SVD б) He normal в) Lecun normal

г) SVD с дополнительным слоем д) He normal с дополнительным слоем е) Lecun normal с дополнительным слоем

Рисунок Б.2 – Графики обучения, полученные в ходе второго эксперимента

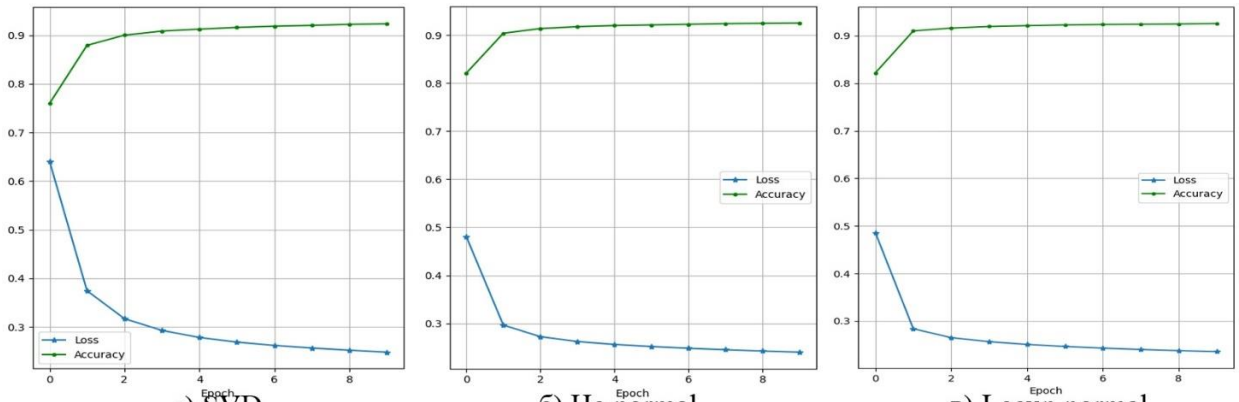


а) SVD б) He normal в) Lecun normal

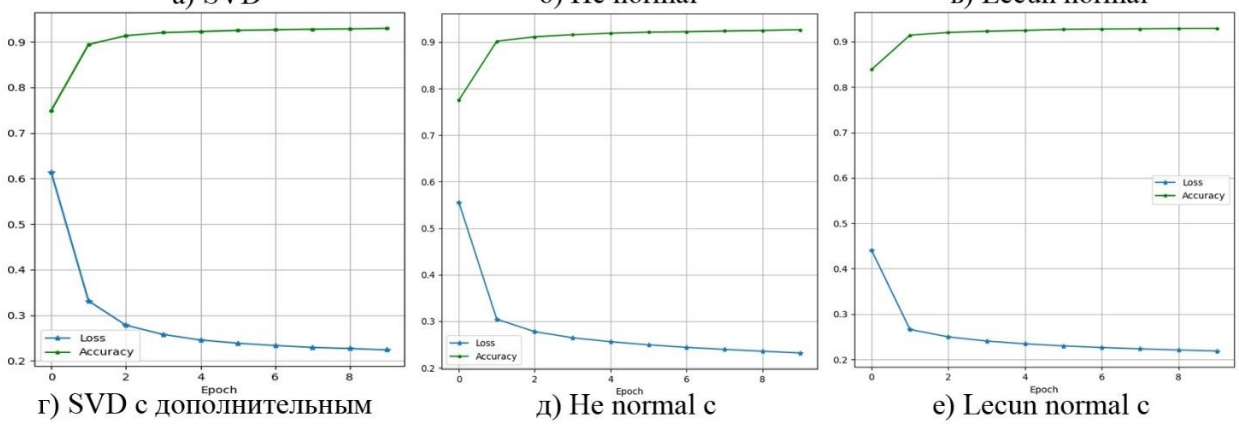


г) SVD с дополнительным слоем д) He normal с дополнительным слоем е) Lecun normal с дополнительным слоем

Рисунок Б.3 – Графики обучения, полученные в ходе третьего эксперимента

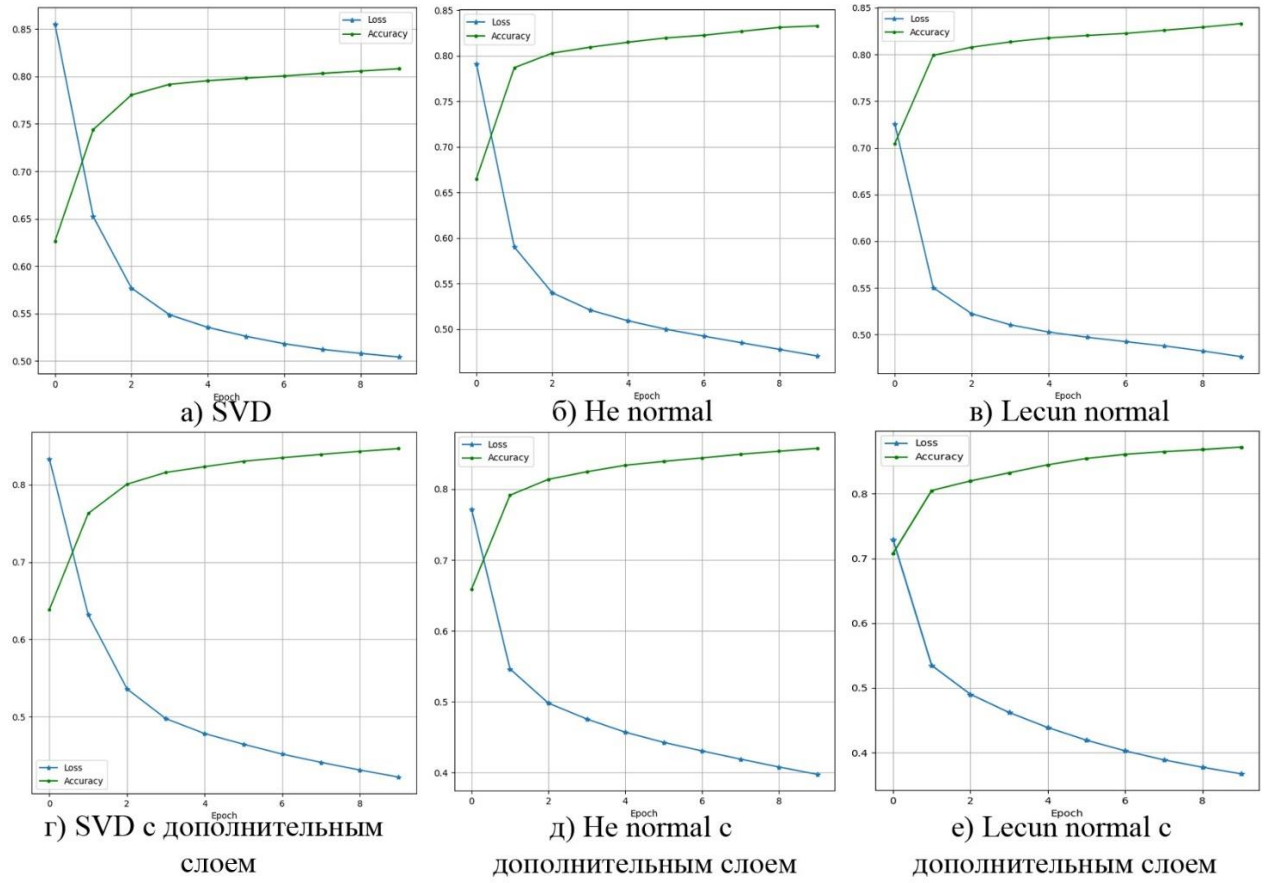


а) SVD б) He normal в) Lecun normal



г) SVD с дополнительным слоем д) He normal с дополнительным слоем е) Lecun normal с дополнительным слоем

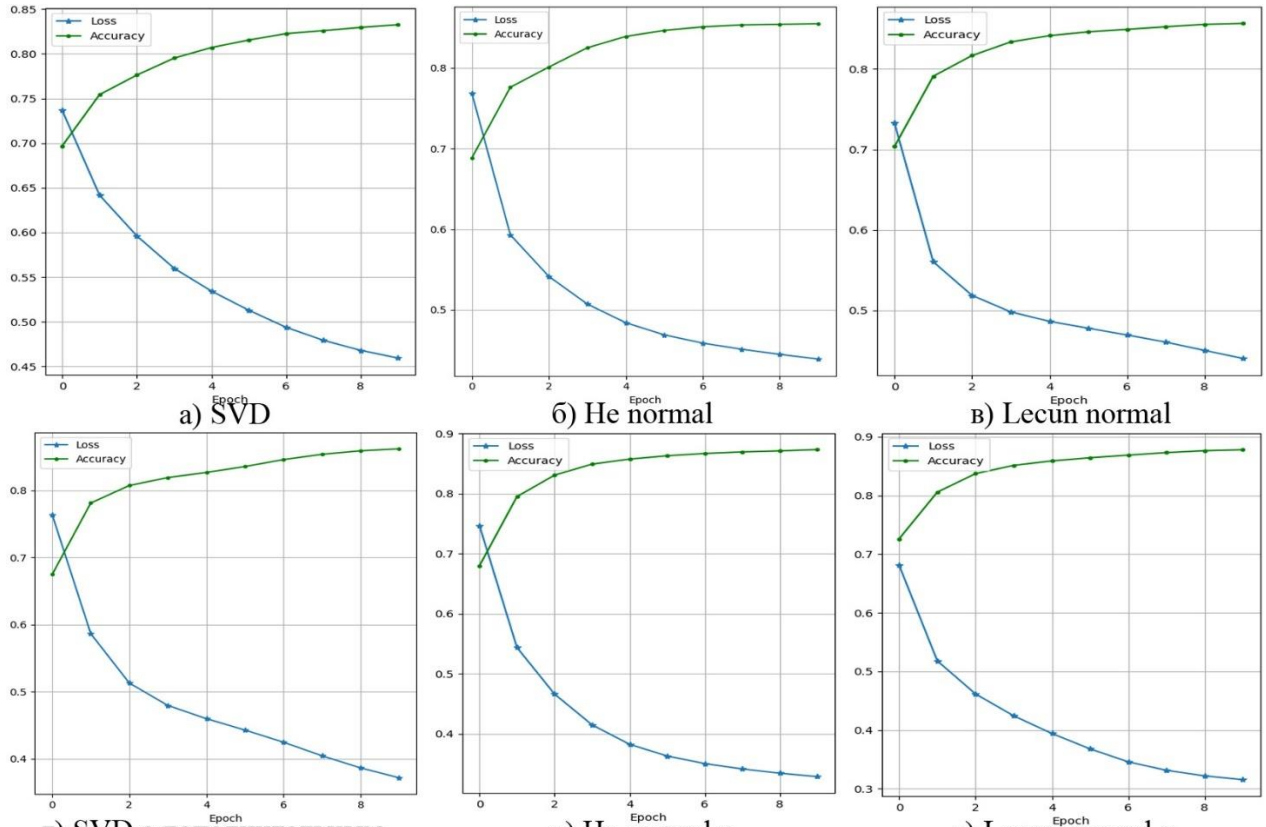
Рисунок Б.4 – Графики обучения, полученные в ходе четвертого эксперимента



а) SVD б) He normal в) Lecun normal

г) SVD с дополнительным слоем д) He normal с дополнительным слоем е) Lecun normal с дополнительным слоем

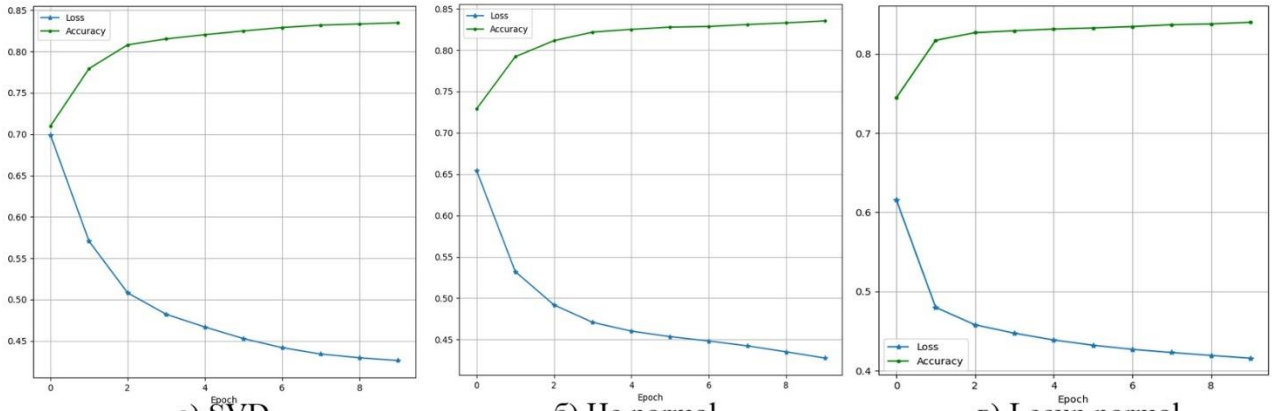
Рисунок Б.5 – Графики обучения, полученные в ходе пятого эксперимента



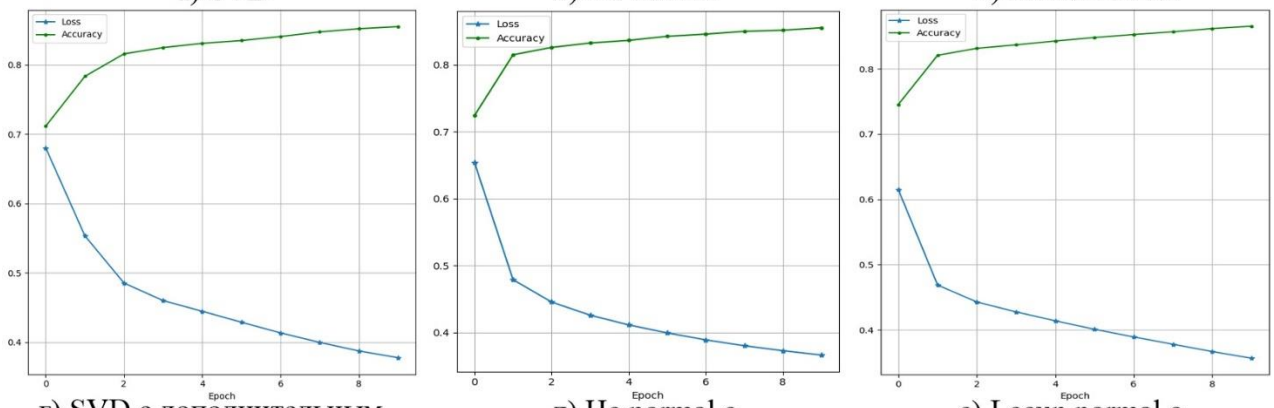
а) SVD б) He normal в) Lecun normal

г) SVD с дополнительным слоем д) He normal с дополнительным слоем е) Lecun normal с дополнительным слоем

Рисунок Б.6 – Графики обучения, полученные в ходе шестого эксперимента

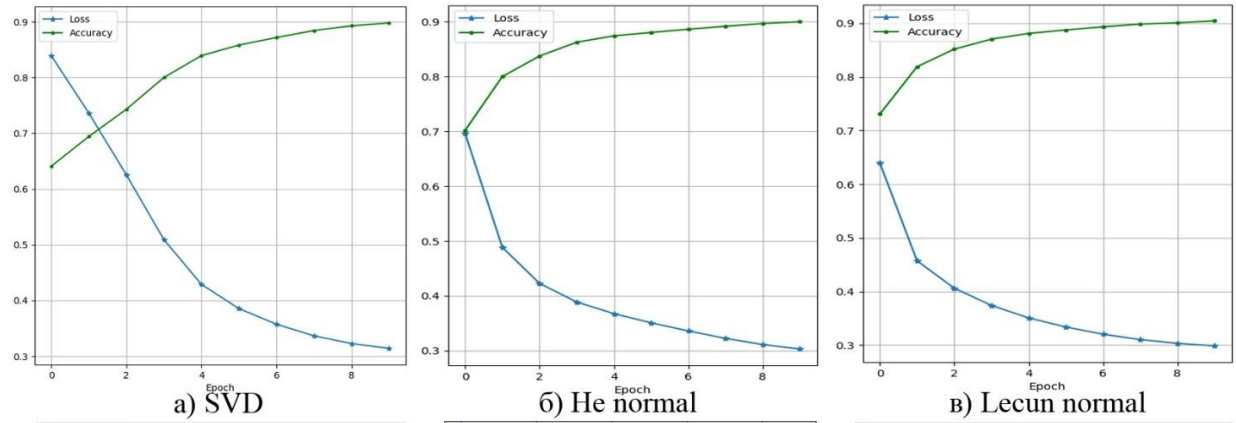


а) SVD б) He normal в) Lecun normal

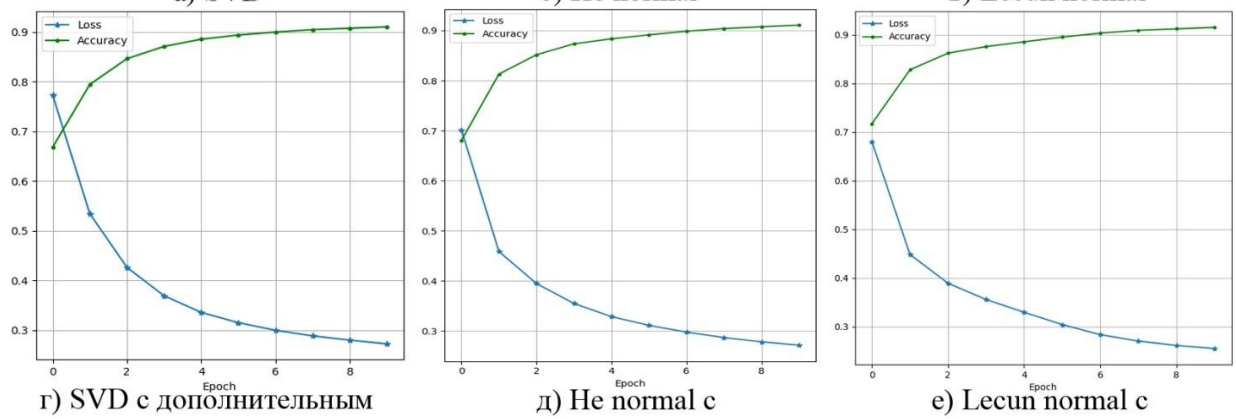


г) SVD с дополнительным слоем д) He normal с дополнительным слоем е) Lecun normal с дополнительным слоем

Рисунок Б.7 – Графики обучения, полученные в ходе седьмого эксперимента

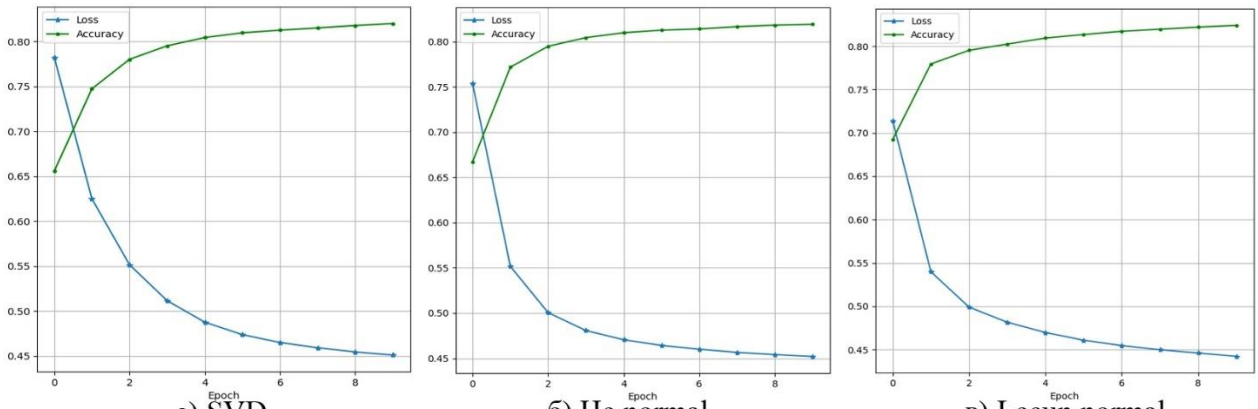


а) SVD б) He normal в) Lecun normal

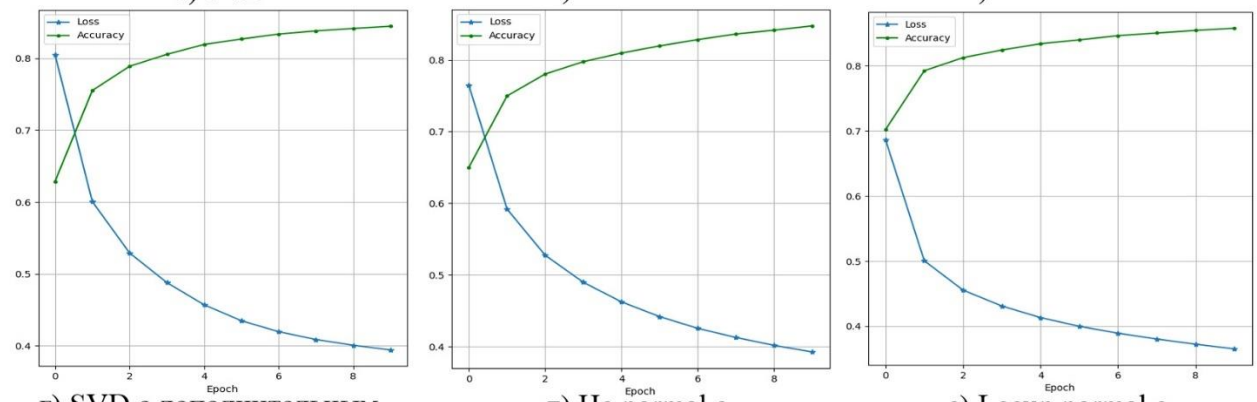


г) SVD с дополнительным слоем д) He normal с дополнительным слоем е) Lecun normal с дополнительным слоем

Рисунок Б.8 – Графики обучения, полученные в ходе восьмого эксперимента

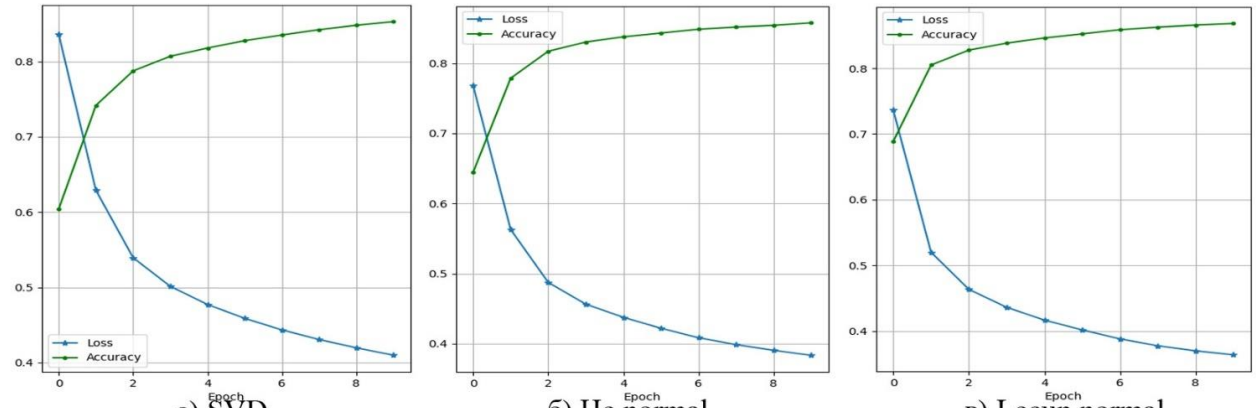


а) SVD б) He normal в) Lecun normal

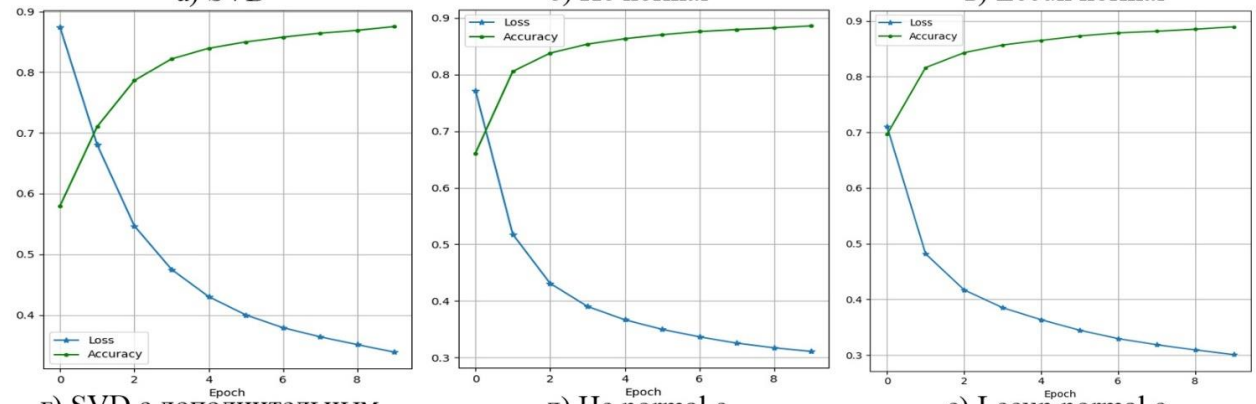


г) SVD с дополнительным слоем д) He normal с дополнительным слоем е) Lecun normal с дополнительным слоем

Рисунок Б.9 – Графики обучения, полученные в ходе девятого эксперимента



а) SVD б) He normal в) Lecun normal



г) SVD с дополнительным слоем д) He normal с дополнительным слоем е) Lecun normal с дополнительным слоем

Рисунок Б.10 – Графики обучения, полученные в ходе десятого эксперимента