Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
Институт космических и информационных технологий
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
(подпись)
«___»_____2019

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника
(код и наименование направления)

Бинарный классификатор на основе нейросети
тема

Руководитель	_____	доцент, к.т.н.	Н.Ю. Сиротинина
	подпись, дата	должность, ученая степень	инициалы, фамилия
Выпускник	_____		М.В. Силин
	подпись, дата		
Нормоконтролер	_____	доцент, к.т.н.	В.И. Иванов
	подпись, дата	должность, ученая степень	инициалы, фамилия

Красноярск 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодняшний момент искусственный интеллект прочно вошёл в нашу жизнь и помогает в решении большого числа задач. Одно из самых перспективных направлений искусственного интеллекта являются нейронные сети. Нейронные сети активно используются в бизнесе, особенно маркетинговой работе, применяются в сфере безопасности, развлечения и других областях. Исследованиями в этой области занимаются все самые передовые компании, например, такие как Microsoft и Google.

Искусственные нейронные сети построены по принципу биологических, т.е. нейронные сети являются примитивной моделью нервной модели живого организма. Подобно человеческому мозгу эти сети способны обучаться. Для нейронных сетей под обучением понимается процесс настройки архитектуры сети (структуры связей между нейронами) и весов синаптических связей для эффективного решения поставленной задачи. Обычно обучение осуществляется по некоторой выборке. По ходу обучения сеть начинает все лучше выполнять поставленные задачи.

Большинство задач, решаемых с применением нейронных сетей, может быть отнесено к следующим классам:

- задачи классификации, т.е. выбора одного из нескольких заранее определённых возможных вариантов решений;
- задачи аппроксимации функции многих переменных по нескольким известным точкам (которые и составляют обучающую выборку);
- предсказание/прогноз;
- оптимизация;
- управление.

Для того, чтобы использовать нейронные сети эффективно, нужно знать особенности их обучения и функционирования. Зачастую, пользователи нейронных сетей не владеют программированием и, как в следствии, им

сложно ориентироваться в существующих нейросетевых библиотеках. Поэтому, актуальным является задача создания программы, которая на простом примере позволит обучиться основам применения нейронных сетей. Разрабатываемая программа может использоваться как для решения несложных задач, так и для обучения.

Целью выпускной квалификационной работы является создание программы для обучения студентов основам нейронных сетей на примере задачи бинарной классификации.

Для достижения поставленной цели необходимо решить следующие задачи:

- разработка технического задания и требований к системе;
- обзор предметной области;
- аналитический обзор аналогов;
- проектирование и разработка программы.

Программа должна обладать следующим функционалом:

- выбор функции активации в каждом слое нейросети;
- выбор количества нейронов в каждом слое нейросети;
- выбор количества слоёв в нейросети;
- выбор количества эпох;
- сохранение графического представления архитектуры нейросети в файл;
- сохранение архитектуры нейросети в файл;
- построение графика точности и ошибок на каждой из эпох.

Программ должна иметь графический интерфейс и поддерживаться на операционных системах Windows актуальных версий.

1 Анализ предметной области

1.1 Основы нейронных сетей

1.1.1 Искусственный нейрон

Несмотря на существенные различия, отдельные типы нейронных сетей обладают несколькими общими чертами. Во-первых, основу каждой нейронных сетей составляют относительно простые, в большинстве случаев – однотипные, элементы (ячейки), имитирующие работу нейронов мозга [1]. Далее под нейроном будет подразумеваться искусственный нейрон, то есть ячейка нейронной сети. Каждый нейрон характеризуется своим текущим состоянием по аналогии с нервными клетками головного мозга, которые могут быть возбуждены или заторможены. Он обладает группой синапсов – однонаправленных входных связей, соединенных с выходами других нейронов, а также имеет аксон – выходную связь данного нейрона, с которой сигнал (возбуждения или торможения) поступает на синапсы следующих нейронов. Общий вид нейрона приведен на рисунке 1 [2].

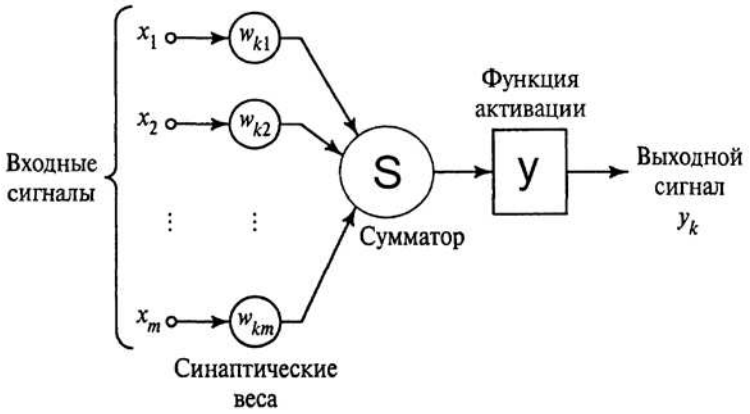

Рисунок 1 – Модель искусственного нейрона

Каждый синапс характеризуется величиной синаптической связи или ее весом, который по физическому смыслу эквивалентен электрической проводимости.

Текущее состояние нейрона определяется, как взвешенная сумма его входов:

$$S = \sum_{i=1}^{n} x_i * w_i \quad (1)$$

Выход нейрона есть функция его состояния:

$$y = f(s) \quad (2)$$

Нелинейная функция f называется активационной и может иметь различный вид, как показано на рисунке 2. Одной из наиболее распространенных, является нелинейная функция с насыщением, так называемая логистическая функция или сигмоида [2]:

$$f(x) = \frac{1}{1+e^{-\alpha x}} \quad (3)$$

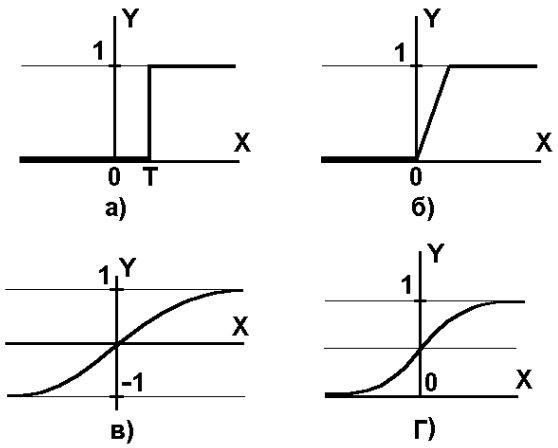

Рисунок 2 – Виды функции активации а) функция единичного скачка; б) линейный порог (гистерезис); в) сигмоида; г) логистическая функция

При уменьшении α сигмоида становится более пологой, в пределе вырождаясь в горизонтальную линию на уровне 0,5. При увеличении α сигмоида приближается по внешнему виду к функции единичного скачка с порогом T в точке $x = 0$. Из выражения для сигмоиды очевидно, что выходное значение нейрона лежит в диапазоне [0,1]. Одно из ценных свойств сигмоидной функции - простое выражение для ее производной, применение которого будет рассмотрено в дальнейшем.

$$f'(x) = \alpha \cdot f(x) \cdot (1 - f(x)) \quad (4)$$

Следует отметить, что сигмоидная функция дифференцируема на всей оси абсцисс, что используется в некоторых алгоритмах обучения. Кроме того, она обладает свойством усиливать слабые сигналы лучше, чем большие, и предотвращает насыщение от больших сигналов, так как они соответствуют областям аргументов, где сигмоида имеет пологий наклон.

В контексте искусственной нейронной сети процесс обучения может рассматриваться как настройка архитектуры сети и весов связей для эффективного выполнения специальной задачи. Обычно нейронная сеть должна настроить веса связей по имеющейся обучающей выборке. Функционирование сети улучшается по мере итеративной настройки весовых коэффициентов. Свойство сети обучаться на примерах делает их более привлекательными по сравнению с системами, которые следуют определенной системе правил функционирования, сформулированной экспертами.

Для конструирования процесса обучения, прежде всего, необходимо иметь модель внешней среды, в которой функционирует нейронная сеть – знать доступную для сети информацию. Эта модель определяет парадигму обучения. Во-вторых, необходимо понять, как модифицировать весовые параметры сети – какие правила обучения управляют процессом настройки. Алгоритм обучения

означает процедуру, в которой используются правила обучения для настройки весов.

Теория обучения рассматривает три фундаментальных свойства, связанных с обучением по примерам: емкость, сложность образцов и вычислительная сложность. Под емкостью понимается, сколько образцов может запомнить сеть, и какие функции и границы принятия решений могут быть на ней сформированы. Сложность образцов определяет число обучающих примеров, необходимых для достижения способности сети к обобщению. Слишком малое число примеров может вызвать «переобученность» сети, когда она хорошо функционирует на примерах обучающей выборки, но плохо на тестовых примерах, подчиненных тому же статистическому распределению [2].

1.1.2 Классификация нейронных сетей

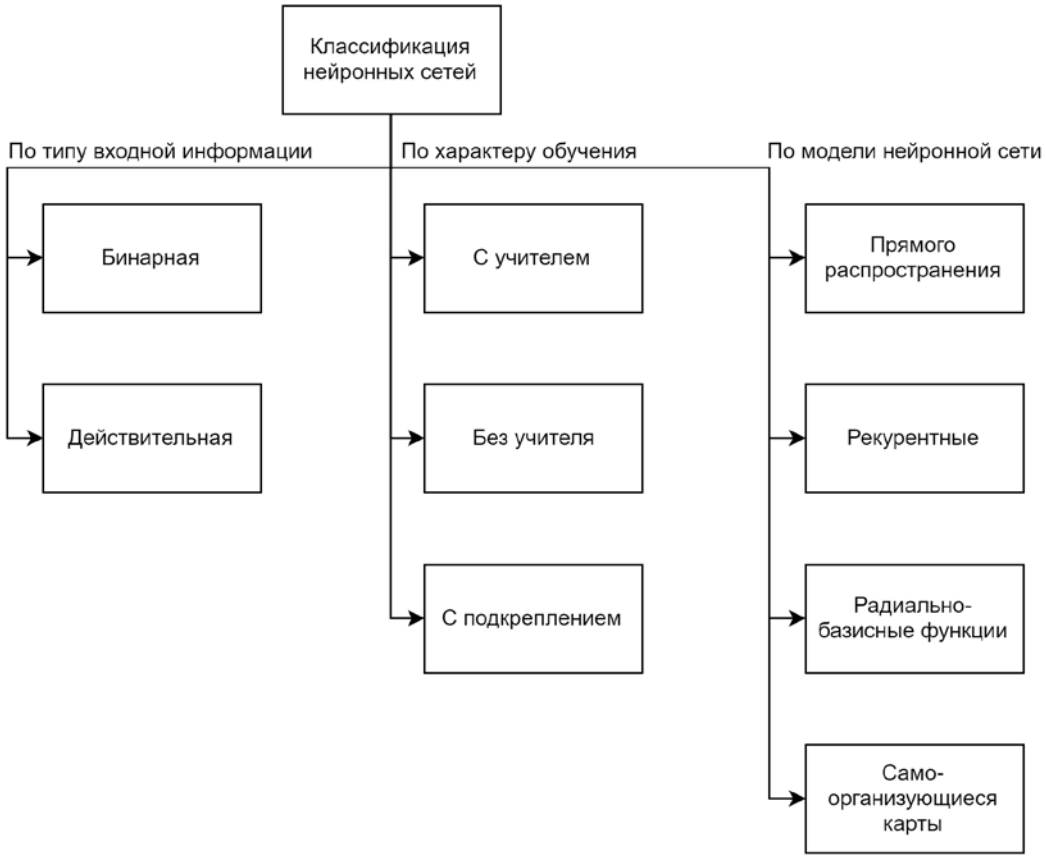

Рисунок 3 – Классификация нейронных сетей

Классификация по характеру обучения

Обучение с учителем: главная цель обучения с учителем – обучить модель на помеченных обучающих данных, что позволит вырабатывать прогнозы на не встречающихся ранее или будущих данных. Понятие «с учителем» относится к набору образцов, где желаемые выходные сигналы (метки) уже известны.

Обучение без учителя: при обучении с учителем правильный ответ нам известен заранее, когда мы обучаем модель, а при обучении без учителя мы имеем дело с непомеченными данными или данными с неизвестной структурой. Использование приёмов обучения без учителя даёт нам возможность установить структуру данных для извлечения значимой информации без управления со стороны известной переменной результата или функции награды.

Обучение с подкреплением: цель обучения с подкреплением заключается в разработке системы (агента), улучшающей свои характеристики на основе взаимодействия со средой [4].

Классификация по входным данным

Действительные входные данные представлены в виде действительных чисел, а двоичные представлены в виде нулей и единиц [3].

Классификация моделей нейронных сетей

Сети прямого распространения: все связи этой сети имеют строгое направление от входных нейронов к их выходам. Среди таких сетей хочется отметить: простейший персептрон, автором которого является Фрэнк Розенблатт и многослойный персептрон.

Нейронные сети Реккурентного типа: данные с выходных нейронов или из скрытого слоя передается частично обратно на входные нейроны.

Радиально базисные функции: это нейронная сеть, в основе которой является наличие скрытого слоя из радиальных элементов и выходного слоя из линейных элементов. Такие сети довольно компактны и обучаются достаточно быстро. Они были предложены в работах Broomhead and Lowe (1988) и Moody and Darkin (1989). Радиально базисная сеть пользуется следующими уникальными свойствами: один скрытый слой, нейроны только скрытого слоя имеют нелинейную функцию активации и синаптические веса скрытого и входного слоев являются единицей.

Сети Кохонена или Самоорганизующиеся карты: этот класс сетей обычно обучается без помощи учителя и часто применяется в задачах связанных с распознаванием изображений. Такие сети способны определять новые элементы во входных данных: если пройдя обучение сеть увидит набор данных, непохожий ни на один из знакомых образцов, то она классифицирует такой набор и не выявит его новизну. Сеть Кохонена имеет всего два слоя: выходной и входной, составленный из радиальных элементов [3].

1.1.3 Архитектура нейросети прямого распространения

Нейронные сети прямого распространения передают информацию от входа к выходу. Нейронные сети часто описываются в виде слоёного торта, где каждый слой состоит из входных, скрытых или выходных клеток. Клетки одного слоя не связаны между собой, а соседние слои обычно полностью связаны.

Нейросеть прямого распространения обычно обучается по методу обратного распространения ошибки.

Метод обратного распространения – это семейство методов «с учителем», используемых для эффективного обучения искусственных нейронных сетей по

принципу градиентного спуска. Главной особенностью обратного распространения является его итеративный метод вычисления подстроечных изменений весов для улучшения работы сети до тех пор, пока она не сможет выполнить задачу, для которой она обучается.

В ходе обучения обратное распространение обычно минимизирует целевую функцию (функцию ошибки) методом градиентного спуска, при этом в слоистой нейросети для вычисления составляющих градиент частных производных целевой функции выполняется передача сигналов в направлении обратном прямому, от выхода к входу.

Чтобы иметь возможность применять метод обратного распространения ошибки, функция активации нейронов должна быть дифференцируемой.

1.2 Задача бинарной классификации

Среди задач классификации особое место занимает задача бинарной классификации. Эта задача является достаточно простой для исследования основных особенностей обучения и использования нейросетей, и в то же время достаточно распространенной.

Любую задачу классификации на m классов можно представить в виде набора m задач бинарной классификации. Например, задачу распознавания изображения 33 букв алфавита можно разбить на 33 задачи следующего вида: предъявленная буква – это А или нет? Буква Б или нет? И т.д. При этом оказывается, что обучение и функционирование m бинарных классификаторов зачастую требует меньше затрат, чем одного классификатора на m классов.

Более того, задача классификации на множество классов в раде случаем вообще не корректна. Например, при построении системы медицинской диагностики некорректно предполагать задачу типа: выбрать для пациента из набора «ОРЗ», «Бронхит», «Пневмония». Во-первых, пациент может не страдать ни одним из перечисленных заболеваний. Введение класса «Здоров»

не исправит положение – вдруг у него некое четвертое заболевание. Во-вторых, перечисленные заболевания могут встречаться одновременно. В этому случае более правильным является создание набора бинарных классификаторов, по одному для каждого диагноза. Если на вопрос о диагнозе все классификаторы дадут отрицательный ответ, не исключено, что пациент страдает от другого заболевания. Если положительный ответ дан двумя и более классификаторами – врач может сам принять решение, что же лечить в первую очередь и как скорректировать лечение [5].

1.3 Анализ существующих аналогов

Цель анализа аналогов состоит в поиске программы, которая бы соответствовала следующим критериям:

Функционал программы:

- выбор функции активации в каждом слое нейросети;
- выбор количества нейронов в каждом слое нейросети;
- выбор количества слоёв в нейросети;
- выбор количества эпох;
- сохранение графического представления архитектуры нейросети в файл;
- сохранение архитектуры нейросети в файл;
- построение графика точности и ошибок на каждой из эпох.

Требования к программе:

- минимальное время ожидания отклика от программы;
- удобный, интуитивно понятный пользовательский интерфейс.

1.3.1 CLAB

CLAB – это прикладной пакет программ, представляющий собой программный имитатор нейросети. Он позволяет создавать нейросеть и обучать её определению принадлежности объекта к одному из двух классов по набору входных сигналов (например, по ответам на заданные вопросы) [5].

В состав пакета входят следующие программы:

- EDITOR – программа записи и редактирования задачника;
- NETGENDER – программа генерации, т.е. определения основных параметров и характеристик нейросети;
- TEACHER – программа-учитель, осуществляющая обучение нейросети по алгоритму обратного распространения ошибки. Этот вариант рассчитан на сопроцессор;
- TEACH-87 – то же, что и TEACHER, но без сопроцессора;
- TESTER – программа тестирования работы нейросети.

В итоге: данный прикладной пакет программ является устаревшим и его трудно найти на просторах интернета.

1.3.2 Онлайн-пример нейронной сети на сайте Primat.org

Данный ресурс позволяет познакомиться с онлайн визуализацией работы простой искусственной нейронной сети (рисунок 4).

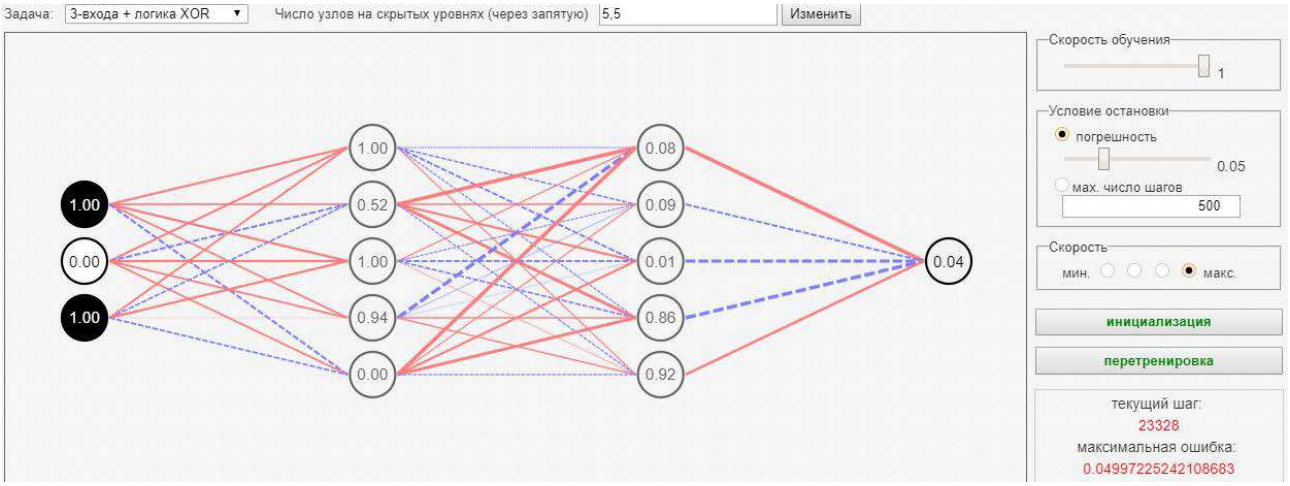

Рисунок 4 – Главная страница ресурса

Доступны три типа демонстраций: решение проблемы XOR, построение функции приближения и распознавание образов.

В программе можно изменяться следующие параметры:

- число узлов скрытых слоев;
- скорость обучения;
- условия остановки;
- скорость визуализации.

Недостатки программы:

- нет сохранения модели нейросети;
- нет выбора слоёв нейросети;
- нет выбора функции активации;
- нет выбора нейронов в каждом слое нейросети.

В итоге: данная программа имеет хорошую визуальную составляющую обучения нейросети, имеет приятный и простой интерфейс. Но большинстве своём данная программа не соответствует критериям, которые были заданы в пункте 1.3.

1.3.3 Нейронная сеть v.2.4.2

Программа имеет графический интерфейс и предназначена для создания нейронной сети с произвольной конфигурацией (рисунок 5).

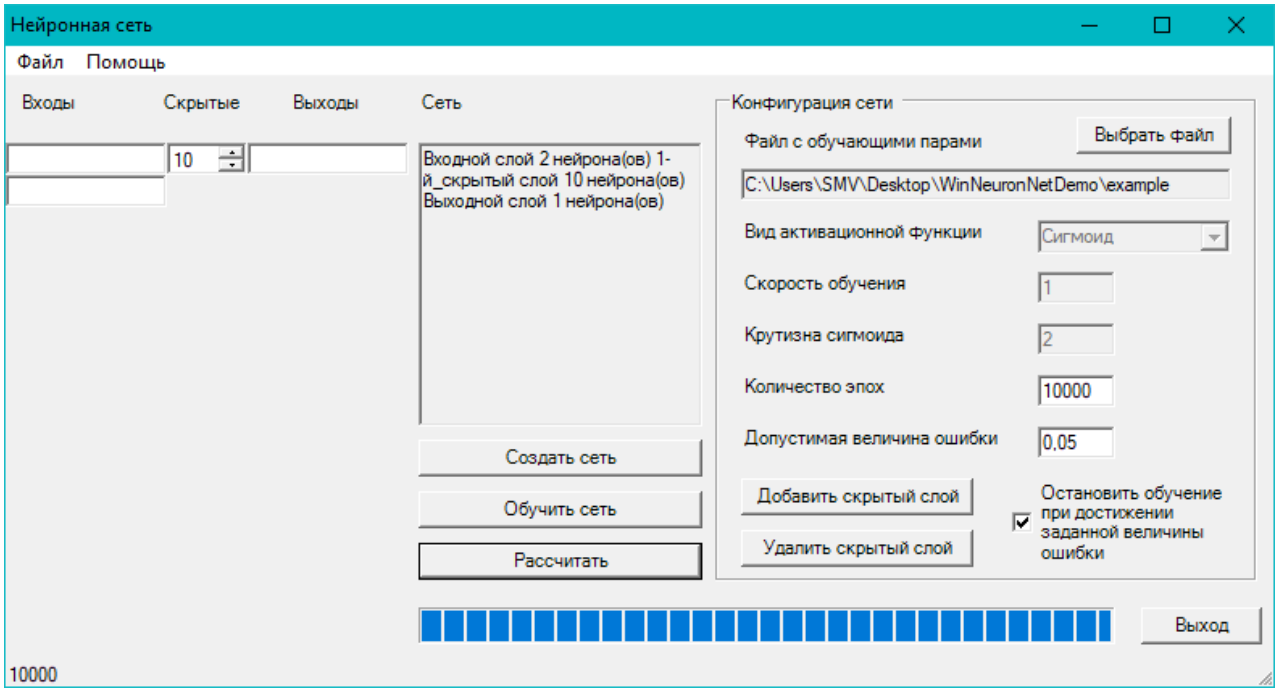

Рисунок 5 – Главное окно программы «Нейронная сеть v2.4.2»

Алгоритм обучения: с обратным распространением ошибки. При создании архитектуры нейросети, можно настроить следующие параметры:

- количество нейронов во входном слое;
- количество нейронов в выходном слое;
- количество скрытых слоев и количество нейронов в каждом из них;
- скорость обучение;
- крутизна сигмоиды;
- количество эпох;
- допустимая величина ошибки.

В качестве дополнительных функций, программа поддерживает сохранение конфигурации сети и автоматическое формирование количества нейронов во входном и выходным слоям из файла с обучающими парами.

В итоге: данная программа является DEMO версией, поэтому удостовериться в работоспособности функционала программы, который описан автором программы, не предоставляется возможным.

1.3.4 Заключение

Результаты сравнения аналогов приведены в таблице 1.

Таблица 1 – Результат сравнения аналогов

Функции, реализуемые системой	Название программы		
	Primat.org	Нейронная сеть v2.4.2	Разработанное приложение
Выбор количества нейронов в каждом слое	Нет	Да	Да
Выбор количества слоёв в нейросети	Нет	Да	Да
Выбор функции активации	Нет	Нет	Да
Выбор количества эпох	Да	Да	Да
Сохранение графического представления архитектуры нейросети в файл	Нет	Нет	Да
Сохранение архитектуры нейросети в файл	Нет	Да	Нет
Построение графика точности и ошибок на каждой из эпох	Нет	Нет	Да

Исходя из таблицы 1 можно сделать вывод о том, что программы, рассматриваемые в пункте 1.3, частично или полностью не соответствуют критериям, которые заданы в пункте 1.3. Вследствие чего, разработка программы, которая поможет разобрать особенности обучения и использования нейросетей студентам, оправдана.

2 Проектирование системы

2.1 Выбор средств разработки

Для решения поставленной задачи использовались следующие технологии и средства:

- язык программирования Python 3.7;
- нейросетевая библиотека Keras 2.2.4. Backend библиотеки Keras представлен нейросетевой библиотекой TensorFlow 1.13.1;
- рабочая среда разработки Anaconda 1.9.7;
- графическая библиотека Tkinter.

2.1.1 Язык программирования Python

Python – высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты – динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты [6].

2.1.2 Нейросетевая библиотека Keras

Keras – это библиотека для Python с открытым исходным кодом, которая позволяет легко создавать нейронные сети. Библиотека совместима с TensorFlow, Microsoft Cognitive Toolkit, Theano и MXNet. Tensorflow и Theano являются наиболее часто используемыми численными платформами на Python для разработки алгоритмов глубокого обучения, но они довольно сложны в использовании [7].

Keras, наоборот, предоставляет простой и удобный способ создания моделей глубокого обучения. Создатель, Франсуа Шолле, разработал ее для того, чтобы максимально ускорить и упростить процесс создания нейронных сетей. Он сосредоточил свое внимание на расширяемости, модульности, минимализме и поддержке Python. Keras можно использовать с GPU и CPU; она поддерживает как Python 2, так и Python 3.

Библиотека содержит многочисленные реализации широко применяемых строительных блоков нейронных сетей, таких как слои, целевые и передаточные функции, оптимизаторы, и множество инструментов для упрощения работы с изображениями и текстом.

2.1.3 Рабочая среда разработки Anaconda

Anaconda – это пакетный менеджер, менеджер окружения, а также Python-дистрибутив, содержащий в себе более 400 основных библиотек (с открытым исходным кодом) для научных и инженерных расчетов. Из предустановленных библиотек можно отметить numpy, scikit-learn, scipy, pandas, а также Jupyter Notebooks.

Основное преимущество заключается в том, что если потребуются дополнительные пакеты после установки, то можно использовать менеджер пакетов conda или pip для установки этих пакетов. Это очень удобно, так как не

нужно беспокоиться из-за совместимости одних пакетов с другими. Всю работу сделают менеджеры пакетов conda или pip. Также Conda упрощает переход между Python 2 и 3.

2.1.4 Графическая библиотека Tkinter

Кроссплатформенная графическая библиотека на основе средств Tk, написанная Стином Лумхольтом (Steen Lumholt) и Гвидо ван Россумом. Входит в стандартную библиотеку Python.

2.2 Проектирование

2.2.1 Функционал программы

Разрабатываемая программа должна соответствовать следующему функционалу:

выбор функции активации в каждом слое нейросети;

- выбор количества нейронов в каждом слое нейросети;
- выбор количества слоёв в нейросети;
- выбор количества эпох;
- сохранение графического представления архитектуры нейросети в файл;
- сохранение архитектуры нейросети в файл;
- построение графика точности и ошибок на каждой из эпох.

Так же программа должна иметь графических интерфейс, который упростит взаимодействие пользователя с программой.

2.2.2 Сценарий использования программы

При открытии программы, пользователь наблюдает графический интерфейс программы. У программы есть 2 режима работы: «Демонстрация» и «Конструктор». Режим «Демонстрация» установлен по умолчанию. После выбора режима работы программы (пользователь может оставить режим, который установлен по умолчанию, или же переключиться на другой режим), пользователю доступен функционал, описанный в пункте 2.2.1. Сценарий использования программы представлен в виде use-case диаграммы на рисунке 6.

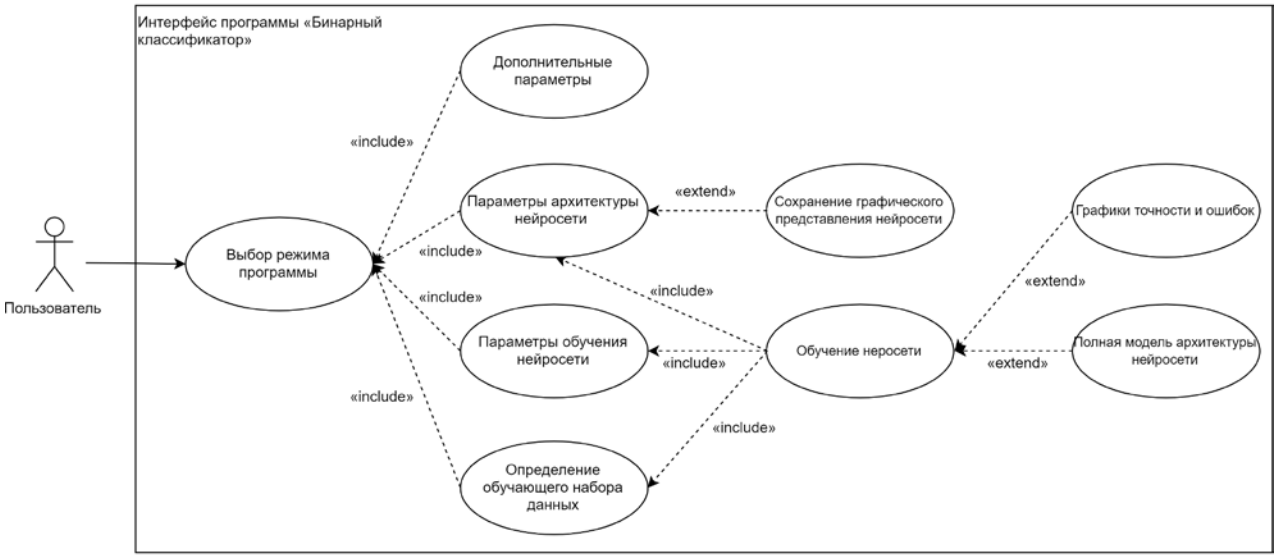

Рисунок 6 – Диаграмма прецедентов

2.2.3 Формирование и обучение нейросети

Формирование нейросети представляет собой выбор её основных параметров и характеристик, и формирование случайной карты весов синоптических связей.

После формирования нейросети, программа переходит к обучению нейросети на основе обучающих данных. При этом можно просмотреть

графики точности и ошибок в реальном времени. После обучения доступна следующая информация:

- графики точности и ошибок;
- графическое представление архитектуры, созданное средствами нейросетевой библиотеки Keras;
- полная модель нейросети.

2.2.4 Описание основных функций нейросетевой библиотеки Keras

Нейросетевая библиотека Keras имеет большое количество функций, которые отвечают за настройку архитектуры и обучение нейросети. В данной работе использовались [8]:

- Dense(x, activation = y). Dense – метод, который описывает слой нейросети, где x – количество нейронов в слое, y – функция активации для каждого нейросна слоя;
- compile(loss = x, optimizer = y, metrics = z]). Compile – метод, отвечающий за компиляцию нейросети. X – функция потерь, y – алгоритм оптимизации, z – функция оценки производительности модели;
- fit(X, Y, epochs = e). Fit – метод отвечает за обучения нейросети. X – входные данные, Y – метки классов, E – количество эпох обучения;
- save('Название_файла.h5'). Save – метод, отвечающий за сохранение полной модели нейросети;
- predict(data). Predict – метод, отвечающий за прогнозирование вывода последним слоем в рабочем режиме нейросетевой модели, data – данные, относительно которых происходит прогнозирование.

3 Разработка приложения

3.1 Реализация модели нейросети

В таблице 2 представлены функции, используемые в программе.

Таблица 2 – Описание функций приложения

Название функции	Описание
DEMODraw	Функция рисовки архитектуры нейросети
DEMONumberNeu	Функция выбора нейрона в сети, для дальнейшего редактирования его параметров
DEMOPredict	Функция предсказания
DEMOChoise	Функция выбора функции активации
DEMOTrain	Функция обучения нейросети
DEMOLoadData	Функция загрузки данных в программу

3.2 Входные данные

Входные данные представлены 3 файлами текстового (txt) формата: questionnaire (анкета), train (тренировочный набор данных) и test (тестовый набор данных). Данные файлы не имеют прямой связи между собой, например, если изменить данные в файле train, то данные в файлах questionnaire и test никак не буду изменены.

Файл questionnaire содержит список вопросов, которые буду заданы респонденту.

Файл train содержит тренировочный набор данных, который представлен в виде ответов на вопросы, которые были заданы респонденту. Так же в файле train содержится информация о принадлежности респондента к классу, который задаёт пользователь программы, например, класс респондента «учится» и «не учится». Данные для валидации отбираются случайным образом из файла train,

пользователь задаёт процент данных, которые в дальнейшем будут использоваться для валидации.

Файл test содержит набор данных, который будет использован для проверки точности предсказания нейросети. Набор данных, который содержит файл, не должен совпадать с набором данных файла train.

3.3 Интерфейс пользователя

При запуске приложения, запускается главное окно (рисунок 7).

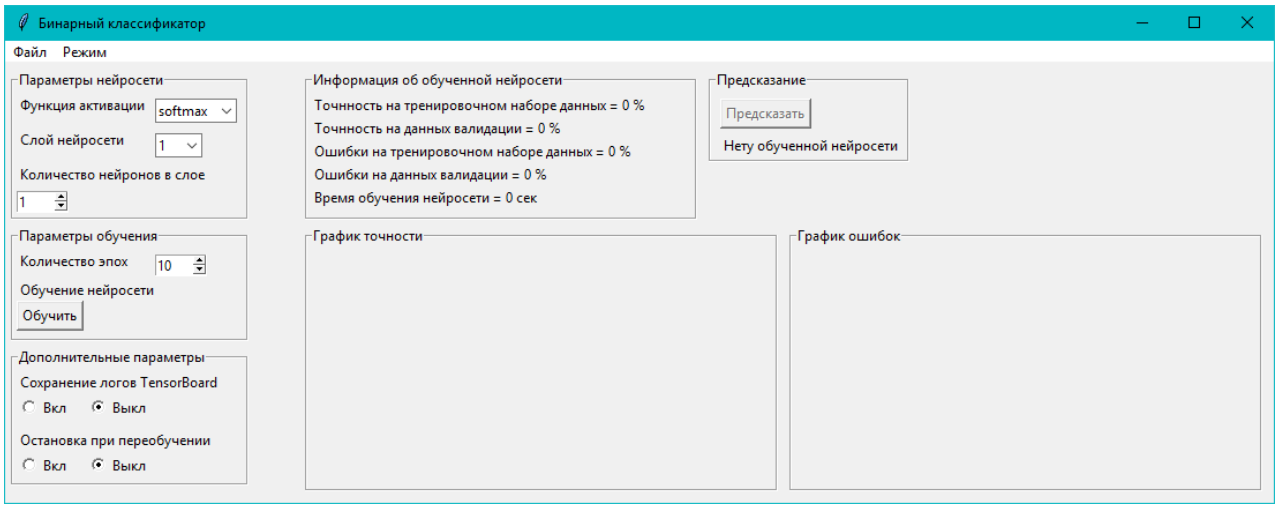

Рисунок 7 – Основное окно приложения

Пользователю предоставляется 2 режима работы: демонстрация и конструктор.

3.3.1 Режим работы «Демонстрация»

При открытии программы, данный режим выбран по умолчанию (рисунок 7). В данном режиме представлена нейросеть, состоящая из трёх слоёв. В каждом слое можно редактировать:

- функцию активации в каждом слое (Choice of Activation Function);
- количество нейронов в слое (Choise of Neural Network Layer).

Помимо настройки слоёв нейросети, в данном режиме можно настроить:

1. Параметры обучения:
   - количество эпох.
2. Дополнительные параметры:
   - сохранение логов TensorBoard;
   - остановка при переобучении.

До обучения нейросети, доступна возможность сохранить архитектуру нейросети.

После обучения нейросети, нам доступны графики точности и ошибок, общая информация о нейросети и возможность предсказания.

3.3.2 Режима работы «Конструктор»

В данный режим можно перейти, нажав кнопку «Режим», затем нажать кнопку «Конструктор» (рисунок 8).

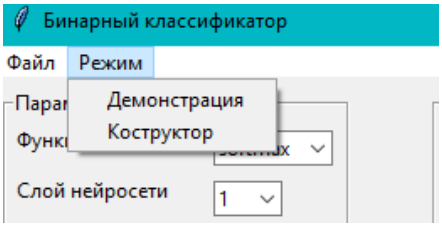

Рисунок 8 – Переход в режим «Конструктор»

Данный режим находится на стадии доработки. В режиме «Конструктор» можно:

- всё то, что включает в себя режим «Демонстрация»;
- выбор количества слоёв в нейросети;
- просмотр и редактирование данных непосредственно из программы.

3.3.3 Обучение нейросети на примере режима работы «Демонстрация»

После выбора режима работы программы и параметров нейросети, можно начать обучение.

Определимся с параметрами:

- функция активации на 1 и 2 слое - Relu, на 3 - Sigmoid;
- количество нейронов на 1 и 2 слое будет равно 20. На 3 слое по умолчанию 1;
- количество эпох 1000;
- включение сохранение логов TensorBoard.

Перед обучением, построим графическое представление архитектуры нейросети (рисунок 9).

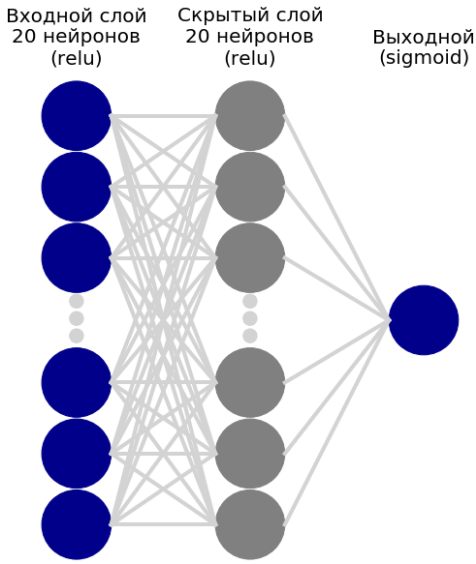

Рисунок 9 – Архитектура нейросети

Исходя из рисунка 9 можно сделать вывод, что наша будущая нейросеть полностью соответствует тем параметрам, которые мы задали.

Для того, чтобы наша нейросеть обучилась, выбираем количество эпох и нажимаем кнопку «Обучить». О завершении обучения оповестит программа (рисунок 10).

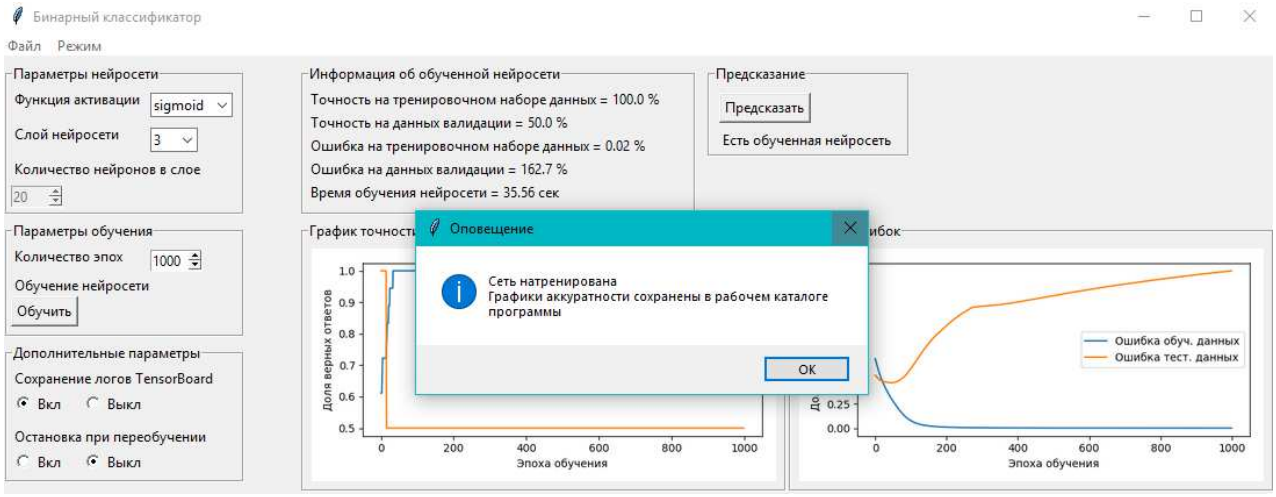

Рисунок 10 – Завершение обучения нейросети

После обучения, в главном окне программы, пользователь может наблюдать (рисунок 11):

- общую информацию о нейросети;
- графики точности и ошибок.

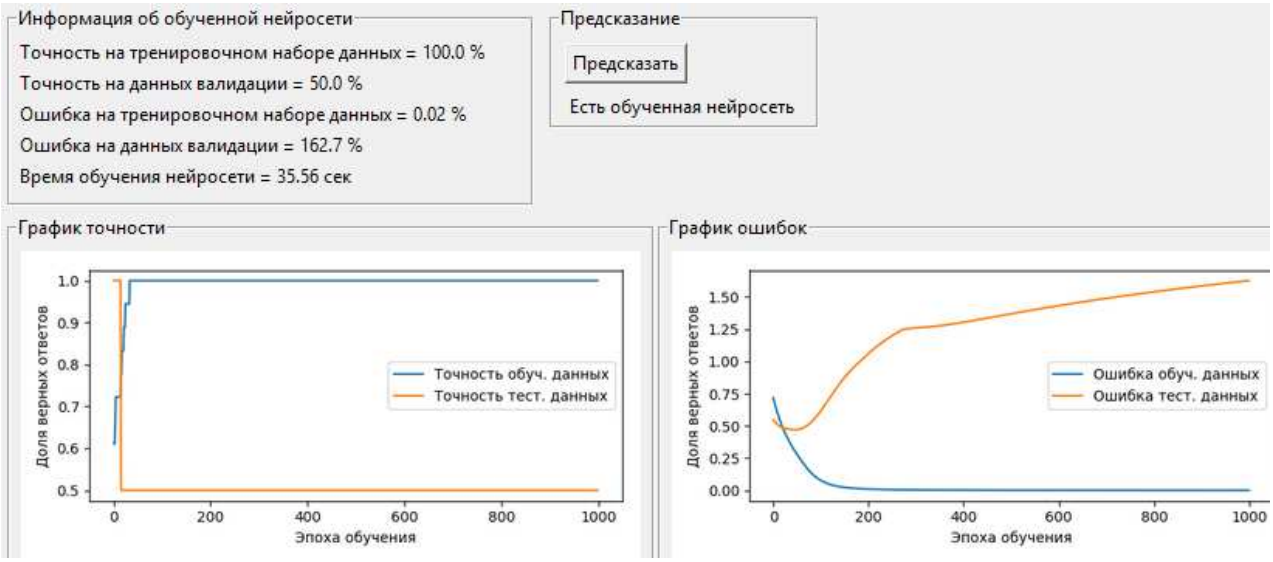

Рисунок 11 – Доступная информация об обученной нейросети

Так же после обучения пользователю доступны файлы, сохранённые в рабочем каталоге программы (рисунок 12):

- файл полной модели нейросети (ModelKeras.h5), который содержит значения весов, конфигурацию модели, и конфигурацию оптимизатора. Файл имеет формат .h5;

- файл (KerasModel.png), построенный средствами Keras, который содержит архитектуру нейросети. Файл имеет формат .png (рисунок 13);

- графики точности (Accurancy.png) и ошибок (Loss.png). Файлы имеют формат .png.

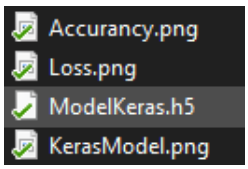

Рисунок 12 – Файлы, доступные после обучения

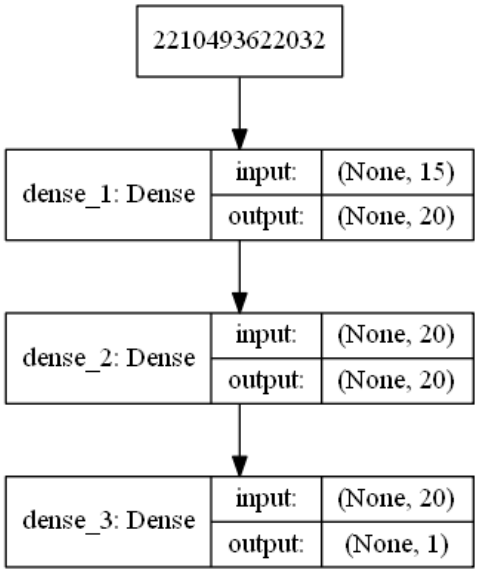

Рисунок 13 – Архитектура нейросети, созданная стандартными средствами библиотеки Keras

Так же не стоит забывать о том, что при обучении был использован дополнительный параметр, который включает логирование TensorBoard. Данный инструмент позволяет увидеть граф нейросети (рисунок 14) и графики точности и ошибок (рисунок 15).

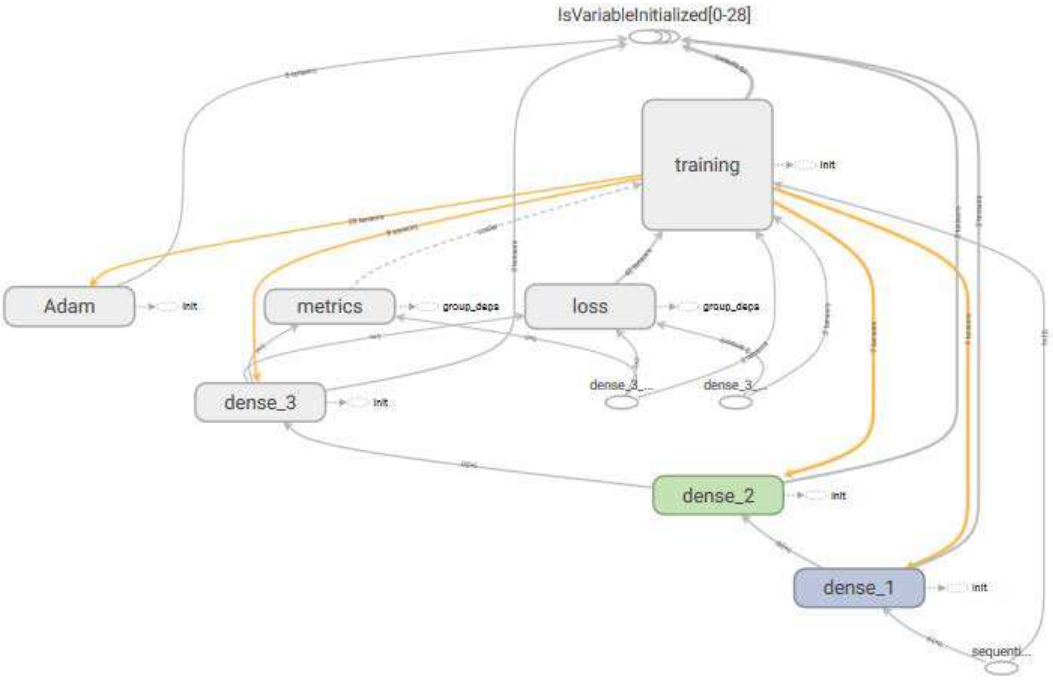

Рисунок 14 – Граф нейросети в TendorBoard

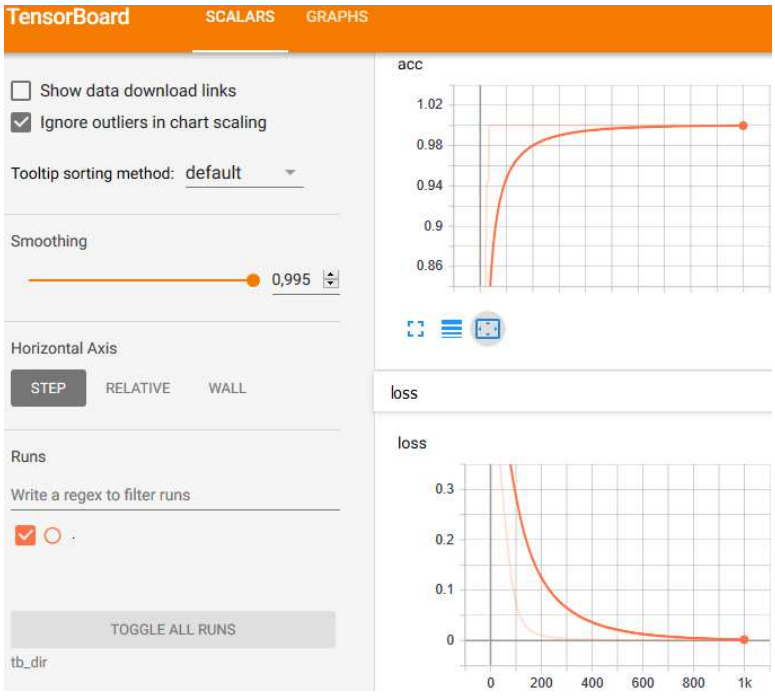

Рисунок 15 – Графики точности и ошибок в TensorBoard

3.3.4 Тестирование обученной нейросети

Тестирование обученной нейросети будет проведено на примере выполнения лабораторной работы.

3.3.4.1 Перечень типовых заданий для лабораторной работы

Построить систему для разделения респондентов по данным анкеты на следующие классы [5]:

- пол: мужчина – женщина;
- социальное положение: студент – не студент;
- семейное положение: семейный – не семейный;
- место проживание: общежитие – квартира;
- отношение к курению: курит – не курит;
- состояние здоровья: здоров – болен;
- успеваемость: успевающий – не успевающий;
- дети: есть дети – бездетный.

3.3.4.2 Анализ задания и определение набора входных данных

После выбора задания лабораторной работы осуществляется его анализ и определение набора входных данных для системы. Описание входных данных представляет собой анкету, вопросы которой допускают ответы «Да», «Нет», «Не знаю».

Рекомендации по составлению анкеты:

- число вопросов: 15 - 20;
- набор вопросов должен полно характеризовать рассматриваемую задачу классификации. При этом желательно избегать явно «лишних» вопросов, не относящихся у существу дела;

- вопросы должны быть чётко и внятно сформулированы;

- анкета не должна содержать вопросов, которые подразумевают в прямой или скрытой форме ответ на решаемую задачу классификации, например, в анкете для задачи определения семейного положения нежелателен вопрос «Носите ли вы иногда обручальное кольцо на безымянном пальце правой руки?»;

- анкета не должна содержать парадоксальных вопросов, например, вида «Перестали ли вы бить свою жену?»;

- анкета должна содержать по возможности большее число объективных вопросов, поскольку вопросы, содержащие словосочетания «Много ли», «Часто ли», «Сильно ли» обычно вносят путаницу. Например, вопрос «Много ли времени вы уделяете своей внешности?» явно неудачен, поскольку для разных людей понятие «Много» может означать существенно различные периоды времени.

Составленная анкета представляет собой описание набора входных данных нейросети, т.е. число входных сигналов и их имена [5].

После составленной анкеты, вопросы необходимо записать в файл в формате, который воспринимается программой «Бинарный классификатор». Этот файл должен иметь расширение .txt. Файл является текстовым и создаётся с использованием любого текстового редактора.

Каждая строка анкеты представлена вопросом, на который будет отвечать респондент

3.3.4.3 Формирование обучающей выборки

По сформированной анкете проводится опрос, ответы заносятся в таблицу с указанием к какому классу относится каждый пример (опрошенный человек).

Основные требования к обучающей выборке:

- выборка должна содержать достаточно много примеров, чтобы сколько-нибудь полно характеризовать решаемую задачу. Для выполнения лабораторной работы, обучающая выборка должна содержать число примеров вдвое больше числа вопросов;

- число примеров первого и второго класса должны быть примерно одинаковым;

- выборка не должна содержать два одинаковых набора ответов на вопросы, относящиеся к разным классам.

Помимо обучающей выборки, нужны данные валидации. Данные валидации случайным образом берутся из обучающий выборки. Процент данных валидации определяет пользователь.

После сбора данных пользователь должен создать файл-задачник, в который заносится обучающая выборка.

Составлять и редактировать задачник можно как с помощью любого текстового редактора, так и с помощью программы «Бинарный классификатор».

Задачники для лабораторной работы состоят из одной страницы. Каждая строка в задачнике – это номер респондента, а каждый столбец – номер вопроса. Последний столбец каждой строки хранит класс к которому принадлежит респондент.

3.3.4.4 Генерация нейросети

Генерация нейросети представляет собой выбор параметров, которые отвечают за построение нейросети. Генерация происходит непосредственно в графическом интерфейсе программы «Бинарный классификатор».

3.3.4.5 Обучение нейросети

После выбора параметров нейросети, создания обучающей выборки идёт обучение нейросети. За обучение нейросети отвечает метод «fit» нейросетевой библиотеки Keras. Более подробное описание метода fit представлено в пункте 2.2.4 выпускной квалификационной работы. Данный метод связан с кнопкой «Обучить», которая находится на графическом интерфейсе программы «Бинарный классификатор». После нажатия кнопки «Обучить» программа переходит к обучению нейросети на примере обучающей выборки. Обучение происходит в автоматическом режиме.

3.3.4.6 Тестирование нейросети

Тестирование нейросети осуществляется для проверки её работоспособности при классификации объектов, отличных от примеров обучающей выборки.

Для тестирования нейросети в приложении «Бинарный классификатор» есть кнопка «Предсказать». За тестирование нейросети отвечает метод «predict» нейросетевой библиотеки Keras. Более подробное описание метода predict представлено в пункте 2.2.4 выпускной квалификационной работы. После нажатия кнопки «Предсказать» программа запрашивает файл с тестовыми данными (рисунок 16). При тестировании конкретных примеров необходимо редактировать набор входных данных.

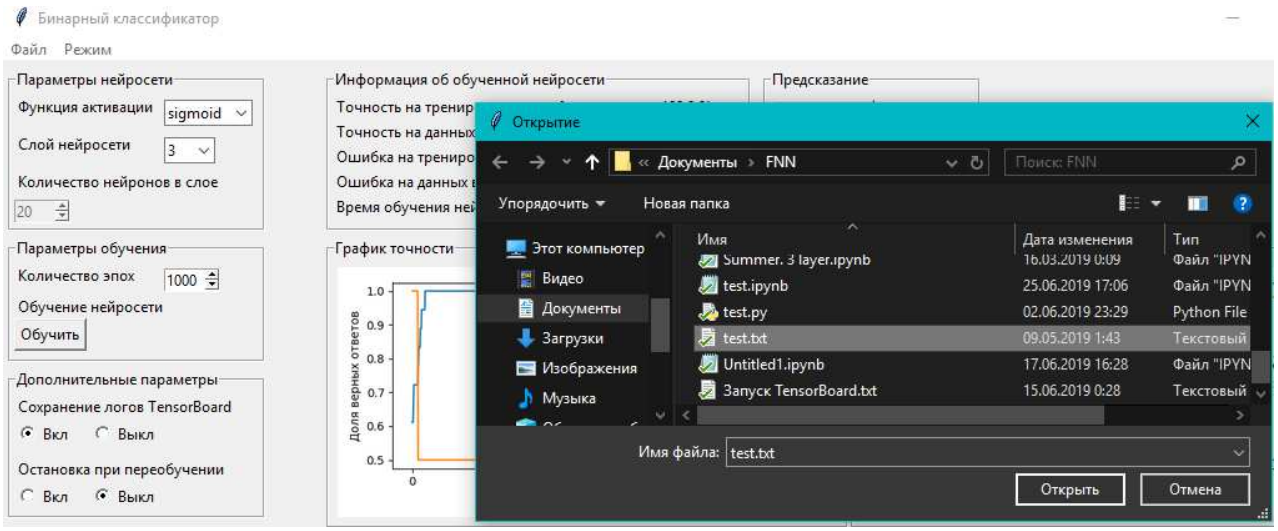

Рисунок 16 – Загрузка тестового набора данных

В файле (test.txt), представленном на рисунке 16, «загадан» респондент мужского пола, нейросеть должна определить принадлежность респондента к классу «мужчина». Чем выше точность принадлежности респондента к классу «мужчина», определяемая нейросетью, тем выше вероятность того, что нейросеть является полностью обученной.

После загрузки данных, нейросеть сделает предсказание относительно тестового набора данных (рисунок 17).

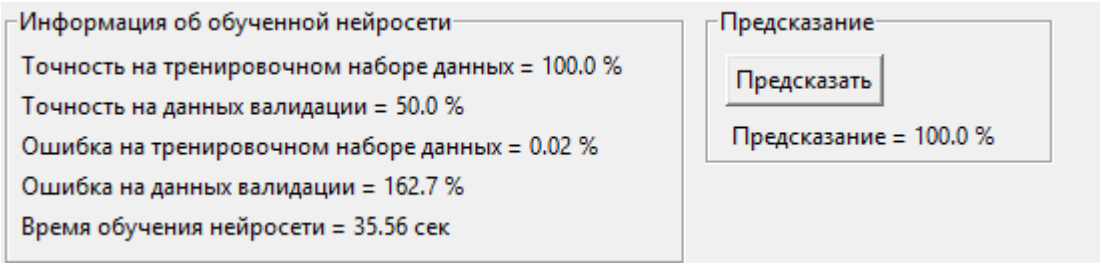

Рисунок 17 – Предсказание

Исходя из данных, представленных на рисунке 17, можно сделать вывод о том, что респондент с 100 % вероятностью является лицом мужского пола.

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы была реализована программа для обучения студентов основам нейронных сетей. Данная программа полностью соответствует заданию выпускной квалификационной работы:

Функции, реализуемые системой:

- выбор функции активации в каждом слое нейросети;
- выбор количества нейронов в каждом слое нейросети;
- выбор количество слоёв в нейросети;
- выбор количества эпох;
- сохранение графического представления архитектуры нейросети в файл;
- сохранение архитектуры нейросети в файл;
- построение графика точности и ошибок на каждой из эпох.

Требования к системе:

- минимальное время ожидания отклика от программы;
- удобный, интуитивно понятный пользовательский интерфейс.

Так же предполагается расширение функционала программы и дальнейшее ее использование в учебном процессе.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Короткий С. Нейронные сети: основные положения [Электронный ресурс] : НТУУ «КПИ». – Режим доступа: http://iit.ntu-kpi.kiev.ua/Neuro/ru.

2. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский. – Москва : Горячая линия – Телеком, 2004. – 452 с.

3. Классификация нейронных сетей [Электронный ресурс] : «Портал искусственного интеллекта». – Режим доступа: http://www.aiportal.ru/articles/ neural-networks/classification.html.

4. С. Рашка, В. Мирджалили. Python и машинное обучение: машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow, 2-е изд. : Пер. с англ. – Санкт-Петербург. : ООО “Диалектика”, 2019 – 656 с. : ил. – Парал. тит. англ.

5. С.Е. Гилев, А.Н. Горбань, Е.М. Миркес, Н.Ю. Сиротинина. Построение экспертных систем на базе нейросетевого бинарного классификатора – Красноярск : 1995. – 28 с.

6. Плещев В.В. Разработка сайтов и Web-программирование / В.В. Плещев. – Москва. : ФИЗМАТЛИТ, 2013. – 200 с.

7. Ф. Шоле. Глубокое обучение на Python. – Санкт-Петербург. : Питер, 2018. - 400 с.

8. Документация Keras [Элекстронный ресурс] : «Keras: The Python Deep Learning library». – Режим доступа: https://keras.io.

ПРИЛОЖЕНИЕ А

Фрагмент листинга кода программы

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras.callbacks import TensorBoard
from keras.utils import plot_model
import time
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from nnv import NNV
from tkinter import
from tkinter import ttk
from tkinter import messagebox as mb
from tkinter import filedialog as fd
from tkinter.ttk import Combobox
from PIL import ImageTk
from PIL import Image as Img
   #1 - Да. 0 - Не знаю. -1 - Нет
X = np.array([[-1., -1., -1., -1., -1.,  1., -1., -1., -1.,  1., -1., -1., -1.,  1., -1.],
              [-1., -1., -1.,  0., -1.,  1., -1., -1., -1.,  1., -1., -1.,  1., -1., -1.],
              [-1., -1.,  1.,  1., -1., -1.,  1., -1.,  1.,  1., -1., -1.,  1., -1., -1.],
              [-1., -1., -1., -1.,  1., -1.,  1., -1.,  1., -1., -1., -1.,  1., -1., -1.],
              [-1., -1., -1., -1.,  1.,  1., -1., -1.,  1.,  1., -1., -1.,  1., -1., -1.],
              [-1., -1., -1.,  1., -1., -1., -1., -1., -1.,  1.,  1., -1., -1., -1.,  0.],
              [-1., -1.,  1.,  1., -1.,  0., -1., -1.,  1.,  1., -1., -1.,  1., -1., -1.],
              [-1.,  1.,  1.,  0.,  1.,  0., -1., -1.,  1., -1., -1., -1., -1.,  1.,  0.],
              [ 1., -1., -1.,  0.,  1.,  1., -1., -1.,  1.,  1., -1., -1.,  1., -1., -1.],
              [-1., -1., -1.,  1., -1., -1., -1., -1.,  1., -1., -1.,  1.,  1., -1., -1.],
              [-1.,  1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,  1.],
              [ 1.,  1.,  1.,  1.,  1.,  1.,  1., -1., -1., -1.,  1.,  1.,  1.,  1., -1.],
              [ 1., -1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,  1., -1.,  1.,  1., -1.,  0.],
              [-1.,  1.,  1.,  1.,  1.,  1.,  1., -1., -1., -1.,  1.,  1., -1., -1., -1.],
              [ 1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1., -1.,  1.],
              [ 1.,  1., -1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1., -1.,  1.],
              [ 1.,  1., -1., -1.,  1.,  1.,  1.,  1.,  1.,  1.,  1., -1., -1.,  1., -1.],
              [ 1.,  1.,  1.,  0.,  1.,  0.,  1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.],
              [ 1.,  1.,  1.,  1.,  1.,  1., -1., -1.,  1., -1., -1., -1., -1.,  1., -1.],
              [ 1.,  1.,  1.,  1.,  1.,  1., -1.,  1., -1.,  1., -1.,  1., -1.,  1.,  1.]])
Y = np.array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
              0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
def DEMODraw():
   layersList = [
      {"title": f"Входной слой\n{NumbNeurons} нейронов\n({SwitchActFunc1})", "units":
NumbNeurons, "color": "darkBlue"},
```

```
            {"title": f"Скрытый слой\n{NumbNeurons1} нейронов\n({SwitchActFunc2})", "units": NumbNeurons1},
            {"title": f"Выходной\n({SwitchActFunc3})", "units": 1, "color": "darkBlue"},
        ]
        plt.rcParams["figure.figsize"] = (50, 10)
        NNV(layersList, max_num_nodes_visible = 6,
            font_size = 20).render(save_to_file = "FNN.png")
        mb.showinfo("Оповещение", "Рисунок архитектуры нейросети сохранён в рабочем каталоге")
    def DEMOTrain():
        S = int(Spin.get())
        global photo, image, photo1, image1, model
        model = None
        start_time = None #Начало измерения времени
        history = None
        end_time = None
        callbacks = None
        TBV = TensorBRVar.get()
        SV = StopVar.get()
        model = Sequential()
        model.add(Dense(NumbNeurons, activation = SwitchActFunc1))
        model.add(Dense(NumbNeurons1, activation = SwitchActFunc2))
        model.add(Dense(1, activation = SwitchActFunc3))
        model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
        if SV == 0 and TBV == 0:
            callbacks = [
                EarlyStopping(patience = 20, monitor = 'val_loss'),
                TensorBoard(log_dir = 'tb_dir')]
            start_time = time.monotonic()
            history = model.fit(X, Y, epochs = S, validation_split = 0.1, verbose = 1, callbacks = callbacks
            end_time = time.monotonic()

        elif TBV == 0:
            callbacks = [
                TensorBoard(log_dir = 'tb_dir')]
            start_time = time.monotonic()
            history = model.fit(X, Y, epochs = S, validation_split = 0.1, verbose = 1, callbacks = callbacks)
            end_time = time.monotonic()
        elif SV == 0:
            callbacks = [
                EarlyStopping(patience=2, monitor='val_loss')]
            start_time = time.monotonic()
            history = model.fit(X, Y, epochs = S, validation_split = 0.1, verbose = 1, callbacks = callbacks)
            end_time = time.monotonic()
        else:
            start_time = time.monotonic()
            history = model.fit(X, Y, epochs = S, validation_split = 0.1, verbose = 1)
            end_time = time.monotonic()
```

```
            if SV == 0:
                ES = callbacks[0].stopped_epoch
                ScoreTrainData = round(history.history['acc'][ES - 1] * 100, 2)
                ScoreValData = round(history.history['val_acc'][ES - 1] * 100, 2)
                LossTrainData = round(history.history['loss'][ES - 1] * 100, 2)
                LossValData = round(history.history['val_loss'][ES - 1] * 100, 2)
            else:
                ScoreTrainData = round(history.history['acc'][S - 1] * 100, 2)
                ScoreValData = round(history.history['val_acc'][S - 1] * 100, 2)
                LossTrainData = round(history.history['loss'][S - 1] * 100, 2)
                LossValData = round(history.history['val_loss'][S - 1] * 100, 2)
            TimeVar = round(end_time - start_time, 2
            print(callbacks)
            TimeL.config(text = f'Время обучения нейросети = {TimeVar} сек') #изменение лейбла TimeLabel
            InfoLabel1.config(text = f'Точность на тренировочном наборе данных = {ScoreTrainData} %')
            InfoLabel2.config(text = f'Точность на данных валидации = {ScoreValData} %')
            InfoLabel3.config(text = f'Ошибка на тренировочном наборе данных = {LossTrainData} %')
            InfoLabel4.config(text = f'Ошибка на данных валидации = {LossValData} %')
            PredButton.config(state = 'normal')
            PredictLabel.config(text = 'Есть обученная нейросеть')
            plot_model(model, to_file = 'KerasModel.png', show_shapes = True)
            model.save('ModelKeras.h5')
            fig, ax = plt.subplots(figsize=(6, 3))
            ax.plot(history.history['acc'],
                    label = 'Точность обуч. данных')
            ax.plot(history.history['val_acc'],
                    label = 'Точность тест. данных')
            ax.set_xlabel('Эпоха обучения')
            ax.set_ylabel('Доля верных ответов')
            ax.legend()
            fig.tight_layout()
            fig.savefig("Accurancy.png")
            image = Img.open('Accurancy.png')
            image = image.resize((400, 200), Img.ANTIALIAS)
            photo = ImageTk.PhotoImage(image)
            ImagesAccLF.config(padx = 1, pady = 5)
            ImagesAccL.config(image = photo)
            #построение графика ошибок
            fig1, ax1 = plt.subplots(figsize=(6, 3))
            ax1.plot(history.history['loss'],
                    label = 'Ошибка обуч. данных')
            ax1.plot(history.history['val_loss'],
                    label = 'Ошибка тест. данных')
            ax1.set_xlabel('Эпоха обучения')
            ax1.set_ylabel('Доля верных ответов')
            ax1.legend()
            fig1.tight_layout()
            fig1.savefig("Loss.png")
```

```
            image1 = Img.open('Loss.png')
            image1 = image1.resize((400, 200), Img.ANTIALIAS)
            photo1 = ImageTk.PhotoImage(image1)
            ImagesLossLF.config(padx = 1, pady = 5)
            ImagesLossL.config(image = photo1)
            mb.showinfo("Оповещение", "Сеть натренирована\nГрафики аккуратности сохранены в рабочем каталоге программы")
        def DEMOChoise(event):
            DEMONumberNeu()
            CF = ComboFunc.get()
            CL = int(ComboLayer.get())
            global SwitchActFunc1, SwitchActFunc2, SwitchActFunc3
            if CF == 'softmax':
                if CL == 1:
                    SwitchActFunc1 = 'softmax'
                elif CL == 2:
                    SwitchActFunc2 = 'softmax'
                else:
                    SwitchActFunc3 = 'softmax'
            elif CF == 'softplus':
                if CL == 1:
                    SwitchActFunc1 = 'softplus'
                elif CL == 2:
                    SwitchActFunc2 = 'softplus'
                else:
                    SwitchActFunc3 = 'softplus'
            elif CF == 'softsign':
                if CL == 1:
                    SwitchActFunc1 = 'softsign'
                elif CL == 2:
                    SwitchActFunc2 = 'softsign'
                else:
                    SwitchActFunc3 = 'softsign'
            elif CF == 'relu':
                if CL == 1:
                    SwitchActFunc1 = 'relu'
                elif CL == 2:
                    SwitchActFunc2 = 'relu'
                else:
                    SwitchActFunc3 = 'relu'
            elif CF == 'tanh':
                if CL == 1:
                    SwitchActFunc1 = 'tanh'
                elif CL == 2:
                    SwitchActFunc2 = 'tanh'
                else:
                    SwitchActFunc3 = 'tanh'
            elif CF == 'sigmoid':
                if CL == 1:
                    SwitchActFunc1 = 'sigmoid'
                elif CL == 2:
```

```
                SwitchActFunc2 = 'sigmoid'
            else:
                SwitchActFunc3 = 'sigmoid'
        def DEMOLoadData():
            file_name = fd.askopenfilename()
            datasets = np.loadtxt(file_name, delimiter=",")
            datasets = np.expand_dims(datasets, axis = 0)
        def DEMOPredict():
            mb.showinfo("Информация", "Выберите файл с данными для предсказания")
            file_name = fd.askopenfilename()
            datasets = np.loadtxt(file_name, delimiter=",")
            datasets = np.expand_dims(datasets, axis = 0)
            preds = model.predict(datasets)
            preds = round(preds[0][0] * 100, 2)
            PredictLabel.config(text = f"Предсказание = {preds} %")
        def DEMONumberNeu():
            global NumbNeurons, NumbNeurons1
            SN = int(SpinNumber.get())
            CL = int(ComboLayer.get())
            if CL == 1:
                NumbNeurons = SN
                SpinNumber.config(state = "normal")
            elif CL == 2:
                NumbNeurons1 = SN
                SpinNumber.config(state = "normal")
            elif CL == 3:
                SpinNumber.config(state = "disabled")
        root = Tk()
        root.title('Бинарный классификатор')
        root.minsize(width = 1100, height = 380)
        main_menu = Menu(root)
        draw_item = Menu(main_menu, tearoff = 0)
        draw_item.add_command(label = 'Нарисовать архитектуру нейросети', command =
DEMODraw)
        draw_item.add_command(label = 'Загрузить обучающий набор данных', command =
DEMOLoadData)
        constr_item = Menu(main_menu, tearoff = 0)
        constr_item.add_command(label = 'Демонстрация', command = DEMODraw)
        constr_item.add_command(label = 'Коструктор', command = DEMODraw)
        main_menu.add_cascade(label = 'Файл', menu = draw_item)
        main_menu.add_cascade(label = 'Режим', menu = constr_item)
        root.config(menu = main_menu)
        Param = LabelFrame(root, text = 'Параметры нейросети', padx = 98, pady = 45)
        Param.place(x = 5, y = 5)
        ParamL = Label(Param)
        ParamL.pack()
        ActFunc = Label(root, text = 'Функция активации')
        ActFunc.place(x = 10, y = 25)
        SwitchActFunc1 = 'softmax'
        SwitchActFunc2 = 'softmax'
        SwitchActFunc3 = 'softmax'
```

```
        NumbNeurons = 1
        NumbNeurons1 = 1
        model = None
        ComboFunc = Combobox(root, values =
                        ['softmax', 'softplus','softsign', 'relu', 'tanh', 'sigmoid'], width = 8)
        ComboFunc.current(0)
        ComboFunc.bind("<<ComboboxSelected>>", DEMOChoise)
        ComboFunc.place(x = 130, y = 30)
        LabelChoiseLayer = Label(root, text = 'Слой нейросети')
        LabelChoiseLayer.place(x = 10, y = 55)
        ComboLayer = Combobox(root, values = [1, 2, 3], width = 3)
        ComboLayer.current(0)
        ComboLayer.bind("<<ComboboxSelected>>", DEMOChoise)
        ComboLayer.place(x = 130, y = 60)
        LabelNumber = Label(root, text = 'Количество нейронов в слое')
        LabelNumber.place(x = 10, y = 85)
        SpinNumber = Spinbox(root, from_ = 1, to = 1000, width = 5, command =
DEMONumberNeu)
        SpinNumber.place(x = 10, y = 110)
        ParamTrain = LabelFrame(root, text = 'Параметры обучения', padx = 98, pady = 30)
        ParamTrain.place(x = 5, y = 140)
        ParamTrainL = Label(ParamTrain)
        ParamTrainL.pack()
        LabelChoiseLayer = Label(root, text = 'Количество эпох')
        LabelChoiseLayer.place(x = 10, y = 160)
        Spin = Spinbox(root, from_ = 10, to = 10000, width = 5)
        Spin.place(x = 130, y = 165)
        LabelLoadData = Label(root, text = 'Обучение нейросети')
        LabelLoadData.place(x = 10, y = 185)
        ButtonLoadData = Button(root, text = "Обучить", command = DEMOTrain)
        ButtonLoadData.place(x = 10, y = 205)
        ParamAdd = LabelFrame(root, text = 'Дополнительные параметры', padx = 98, pady = 40)
        ParamAdd.place(x = 5, y = 245)
        ParamAddL = Label(ParamAdd)
        ParamAddL.pack()
        TensorBLabel = Label(root, text = 'Сохранение логов TensorBoard')
        TensorBLabel.place(x = 10, y = 265)
        StopLabel = Label(root, text = 'Остановка при переобучении')
        StopLabel.place(x = 10, y = 315)
        TensorBRVar = IntVar()
        TensorBRVar.set(1)
        TensorBR1 = Radiobutton(text = 'Вкл', variable = TensorBRVar, value = 0)
        TensorBR2 = Radiobutton(text = 'Выкл', variable = TensorBRVar, value = 1)
        TensorBR1.place(x = 10, y = 285)
        TensorBR2.place(x = 70, y = 285)
        StopVar = IntVar()
        StopVar.set(1)
        StopR1 = Radiobutton(text = 'Вкл', variable = StopVar, value = 0)
        StopR2 = Radiobutton(text = 'Выкл', variable = StopVar, value = 1)
        StopR1.place(x = 10, y = 335)
        StopR2.place(x = 70, y = 335)
```

```
InfoLF = LabelFrame(root, text = 'Информация об обученной нейросети', padx = 165, pady = 45)
InfoLF.place(x = 260, y = 5)
InfoL = Label(InfoLF)
InfoL.pack()
InfoLabel1 = Label(root, text = 'Точнность на тренировочном наборе данных = 0 %')
InfoLabel1.place(x = 265, y = 25)
InfoLabel2 = Label(root, text = 'Точнность на данных валидации = 0 %')
InfoLabel2.place(x = 265, y = 45)
InfoLabel3 = Label(root, text = 'Ошибка на тренировочном наборе данных = 0 %')
InfoLabel3.place(x = 265, y = 65)
InfoLabel4 = Label(root, text = 'Ошибка на данных валидации = 0 %')
InfoLabel4.place(x = 265, y = 85)
TimeL = Label(root, text = 'Время обучения нейросети = 0 сек')
TimeL.place(x = 265, y = 105)
#дополнительная информация
AddInfoLF = LabelFrame(root, text = 'Предсказание', padx = 82, pady = 20)
AddInfoLF.place(x = 610, y = 5)
AddInfoL = Label(AddInfoLF)
AddInfoL.pack()
PredButton = Button(text = 'Предсказать', command = DEMOPredict, state = 'disabled')
PredButton.place(x = 620, y = 30)
PredictLabel = Label(text = "Нету обученной нейросети")
PredictLabel.place(x = 620, y = 60)
image = None
photo = None
image1 = None
photo1 = None
ImagesAccLF = LabelFrame(root, text = 'График точности', padx = 200, pady = 95)
ImagesAccLF.place(x = 260, y = 140)
ImagesAccL = Label(ImagesAccLF)
ImagesAccL.pack()
ImagesLossLF = LabelFrame(root, text = 'График ошибок', padx = 200, pady = 95)
ImagesLossLF.place(x = 680, y = 140)
ImagesLossL = Label(ImagesLossLF)
ImagesLossL.pack()
root.mainloop()
```

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт
Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В.Непомнящий
подпись инициалы, фамилия
« 28 » 06 2019 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника
код и наименование направления

Бинарный классификатор на основе нейросети
тема

Руководитель	20.06.19 подпись, дата	доцент, к.т.н. должность, ученая степень	Н.Ю.Сиротинина инициалы, фамилия
Выпускник	, 20.06.19 подпись, дата		М.В.Силин инициалы, фамилия
Нормоконтролер	27.06.19 подпись, дата	доцент, к.т.н. должность, ученая степень	В.И. Иванов инициалы, фамилия

Красноярск 2019