

**Факультет Информационных технологий**

**Кафедра   информационных систем**

**Направление подготовки/Специальность Прикладная информатика**

**КУРСОВОЙ ПРОЕКТ**

|  |  |
| --- | --- |
| **по дисциплине** | **Машинное обучение и анализ данных** |
|  |  |
| **Задание / Вариант №** | **10. Проектирование нейронной сети для анализа котировок валют** |
|  |  |
| **Тема** | **Обучение нейронной сети для решения прикладной задачи анализа данных** |
|  |  |
| **Выполнена обучающимся группы** | **н.ИЗДтв 23.1/Б3-21** |
| **ФИО обучающегося** | **Панфилов Константин Олегович** |
|  |  |
| **Преподаватель** | **Зайцев Сергей Александрович** |

г. Москва – 2023 г.

**Оглавление**

[**Введение** 3](#_Toc128163970)

[**1. Анализ предметной области** 4](#_Toc128163971)

[**1.1. Искусственные нейронные сети** 4](#_Toc128163972)

[**1.2. Основные проблемы, решаемые искусственными нейронными сетями** 9](#_Toc128163973)

[**1.3. Применение нейронных сетей с прямой связью для предсказания поведения стоимости портфеля акций** 11](#_Toc128163974)

[**2. Проектирование** 13](#_Toc128163975)

[**2.1. Функциональные требования** 13](#_Toc128163976)

[**2.2. Нефункциональные требования** 14](#_Toc128163977)

[**2.3 Разработка архитектуры сети** 14](#_Toc128163978)

[**2.4. Формирование обучающей выборки** 15](#_Toc128163979)

[**3. Реализация** 17](#_Toc128163980)

[**3.1. Используемые технологии разработки** 17](#_Toc128163981)

[**3.2. Топология нейронной сети** 18](#_Toc128163982)

[**3.3 Функция стоимости** 23](#_Toc128163983)

[**3.4 Оптимизатор** 24](#_Toc128163984)

[**3.5 Настройка нейросети** 24](#_Toc128163985)

[**3.6 Окно результата работы программы** 26](#_Toc128163986)

[**4. Тестирование** 26](#_Toc128163987)

[**4.1. Функциональное тестирование** 27](#_Toc128163988)

[**Заключение** 28](#_Toc128163989)

[**Список литературы** 29](#_Toc128163990)

# **Введение**

Актуальность работы определяется возросшим интересом к изучению нейронных сетей, которые используются для решения многих задач искусственного интеллекта.

В современном мире интерес к качественному прогнозированию финансовых рынков становится все более острым. Это связано с быстрым развитием высоких технологий и, соответственно, с появлением новых инструментов анализа данных. Однако технический анализ, к которому привыкло большинство участников рынка, неэффективен. Прогнозы, основанные на экспоненциальных скользящих средних, осцилляторах и других индикаторах, не дают ощутимого результата, потому что. экономика часто иррациональна, потому что ею движут иррациональные мотивы людей.

В последние годы у финансовых аналитиков стал проявляться большой интерес к так называемым искусственным нейронным сетям — это математические модели, а также их программные или аппаратные реализации, построенные по принципу организации и функционирования биологических нейронных сетей - сетей нервных клеток живого организма. Эта концепция возникла при изучении процессов, происходящих в мозге во время мышления, и при попытке смоделировать эти процессы. Впоследствии эти модели стали использоваться в практических целях, как правило, в задачах прогнозирования. Нейронные сети не программируются в обычном смысле этого слова, они обучаются. Способность к обучению - одно из главных преимуществ нейронных сетей перед традиционными алгоритмами. Технически, обучение заключается в нахождении коэффициентов связей между нейронами. В процессе обучения нейронная сеть способна выявлять сложные взаимосвязи между входными и выходными данными, а также выполнять обобщение. Способность нейронной сети к прогнозированию напрямую вытекает из ее способности обобщать и выделять скрытые зависимости между входными и выходными данными. После обучения сеть способна предсказать будущее значение определенной последовательности на основе нескольких предыдущих значений и/или некоторых существующих в данный момент факторов.

Целью этой работы является проектирование и разработка нейронной сети для прогнозирования поведения стоимости портфеля акций.

Для достижения этой цели необходимо выполнить следующие задачи:

1. проанализировать предметную область;

2. изучите теоретические основы прогнозирования поведения портфеля акций S&P 500 с использованием искусственных нейронных сетей;

3. спроектируйте архитектуру нейронной сети;

4. внедрите приложение для прогнозирования поведения портфеля акций индекса S&P 500;

5. сформируйте обучающую выборку;

6. протестируйте систему.

# **1. Анализ предметной области**

## **1.1. Искусственные нейронные сети**

Искусственные нейронные сети (ИНС) – набор моделей биологических нейронных сетей. Они представляют собой сеть элементов - искусственных нейронов, соединенных между собой синаптическими связями. Сеть обрабатывает входную информацию и, в процессе изменения своего состояния с течением времени, генерирует набор выходных сигналов [3].

Нейронные сети возникли в результате исследований в области искусственного интеллекта, а именно из попыток воспроизвести способность биологических нервных систем обучаться и исправлять ошибки путем моделирования низкоуровневой структуры мозга.

Впервые математическая модель искусственного нейрона была предложена У. Маккалохом и У. Питтсом. На практике сеть была реализована Фрэнком Розенблаттом в 1958 году как компьютерная программа, а позже, как электронное устройство - персептрон [9].

Изначально нейрон мог оперировать только сигналами логического нуля и логической единицы, поскольку он был построен на основе биологического прототипа, который может находиться только в двух состояниях - возбужденном или невозбужденном. Рассмотрим структуру биологического и искусственного нейрона и взаимосвязь между ними.

*Биологический нейрон*

Мозг состоит из очень большого количества (приблизительно 10 000 000 000) нейронов, соединенных многочисленными соединениями. Нейроны - это особые клетки, способные передавать электрохимические сигналы. Тело клетки содержит множество ветвящихся отростков двух типов - дендритов и аксонов. Дендриты служат входными каналами для нервных импульсов от других нейронов. Эти импульсы поступают в тело клетки размером от 3 до 100 микрон, вызывая ее специфическое возбуждение, которое распространяется по аксону. Аксоны клетки соединены с дендритами других клеток через синапсы. При активации нейрон посылает электрохимический сигнал вниз по своему аксону. Через синапсы этот сигнал достигает других нейронов, которые могут быть активированы. Нейрон активируется, когда общий уровень сигналов, поступивших в его ядро от дендритов, превышает определенный уровень (порог активации) [4].

На рисунке 1 показана структура биологического нейрона



Рис. 1. Биологический нейрон

*Искусственный нейрон*

Искусственный нейрон имитирует свойства биологического нейрона. На вход искусственного нейрона поступает определенный набор сигналов, каждый из которых является выходом другого нейрона. Каждый входной сигнал умножается на соответствующий вес, аналогичный синаптической силе, и все продукты суммируются для определения уровня активации нейрона [15].

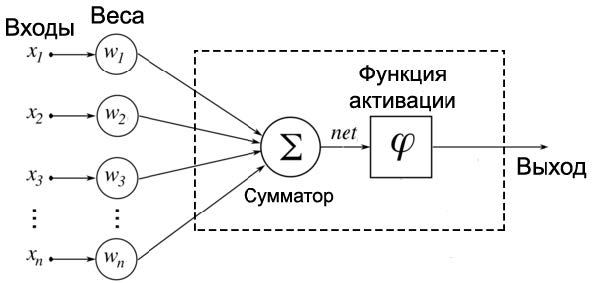
Модель искусственного нейрона показана на рисунке 2.

Рис. 2. Модель искусственного нейрона

Входные сигналы x умножаются на веса w, затем суммируются. Полученное число подается в функцию активации φ, которая вычисляет выходное значение.

Функция активации вычисляет выходной сигнал нейрона в соответствии с уровнем активности [6]. На рисунке 3 показаны графики некоторых функций активации.

Рис. 3. Функции активации: а) сигмоидальная; б) пороговая; в) линейная

*Структура нейронных сетей*

Хотя один нейрон способен выполнять простейшие процедуры распознавания, мощь нейронных вычислений исходит от соединений нейронов в сетях. Простейшая сеть состоит из группы нейронов, образующих слой, как показано в правой части рисунка 4. Обратите внимание, что вершины (круги слева) служат только для распределения входных сигналов. Они не выполняют никаких вычислений и поэтому не будут считаться слоем. По этой причине они обозначены кружками, чтобы отличать их от вычислительных нейронов, обозначенных квадратами. Каждый элемент из набора входов X связан с отдельным весом с каждым искусственным нейроном. И каждый нейрон выдает взвешенную сумму входных данных в сеть [11]. В искусственных и биологических сетях многие соединения могут отсутствовать, все соединения показаны для наглядности. Также могут существовать соединения между выходами и входами элементов в слое. Удобно рассматривать веса как элементы матрицы W. Матрица содержит n строк и m столбцов, где n - количество входных данных, а m - количество нейронов. Например, w23 - это вес, который связывает второй входной сигнал с третьим нейроном. Таким образом, вычисление выходного вектора Y, компонентами которого являются выходы yi нейронов, сводится к умножению матрицы Y = XW.

Рис. 4. Простейшая однослойная нейронная сеть

Более крупные и сложные нейронные сети обычно обладают большими вычислительными возможностями. Хотя были созданы сети всех мыслимых конфигураций, слоистая организация нейронов повторяет слоистые структуры определенных частей мозга [14].

Оказалось, что такие многослойные сети обладают большими возможностями, чем однослойные сети, и в последние годы были разработаны различные алгоритмы для их обучения.

Многослойные сети могут быть сформированы каскадами слоев. Выходные данные одного слоя являются входными данными для следующего слоя. Такая сеть показана на рисунке 5.

Рис. 5. Многослойная нейронная сеть

Многослойные сети не могут привести к увеличению вычислительной мощности по сравнению с однослойной сетью только в том случае, если функция активации между слоями не является линейной. Вычисление выходного сигнала слоя состоит в умножении входного вектора на первую весовую матрицу, а затем умножении (если нет нелинейной функции активации) результирующего вектора на вторую весовую матрицу. Поскольку умножение матриц является ассоциативным, двухслойная линейная сеть эквивалентна одному слою с весовой матрицей, равной произведению двух весовых матриц. Следовательно, любая многослойная линейная сеть может быть заменена эквивалентной однослойной сетью.

**1.2. Основные проблемы, решаемые искусственными нейронными сетями**

*Классификация образов.*

Задача состоит в том, чтобы указать, принадлежит ли входное изображение, представленное вектором признаков, к одному или нескольким предопределенным классам [12]. Известные области применения включают распознавание букв, распознавание речи, классификацию сигналов электрокардиограммы, классификацию клеток крови и рейтинговые задачи.

*Кластеризация/категоризация.*

При решении задачи кластеризации, которая также известна как неконтролируемая классификация шаблонов, нет обучающей выборки с выборками классов. Алгоритм кластеризации основан на сходстве изображений и помещает похожие изображения в один кластер. Кластеризация использовалась для извлечения знаний, сжатия данных и изучения свойств данных.

*Аппроксимация функций.*

Предположим, у нас есть обучающая выборка ((X1, Y2), (X2, Y2), ..., (XN, YN)), которая генерируется неизвестной функцией, искаженной шумом. Проблема аппроксимации состоит в том, чтобы найти оценку для этой функции.

*Предсказание/прогноз.*

Пусть N дискретных выборок {y(t1), y(t2), ..., y(tn)} заданы в последовательные моменты времени t1, t2, ..., tn. Проблема состоит в том, чтобы предсказать значение y(tn+1) в момент времени tn+1. Прогнозы оказывают значительное влияние на принятие решений в бизнесе, науке и технике.

*Оптимизация.*

## Многочисленные задачи в математике, статистике, инженерии, естественных науках, медицине и экономике можно считать задачами оптимизации. Задача оптимизации состоит в том, чтобы найти решение, удовлетворяющее системе ограничений и максимизирующее или минимизирующее целевую функцию.

## Как нейронная сеть решает все эти, часто не формализованные или трудно формализуемые задачи? Как известно, для решения подобных задач традиционно используются два основных подхода. Первый, основанный на правилах, характерен для экспертных систем. Она основана на описании предметной области в виде набора правил (аксиом) "если..., то..." и правил вывода. Желаемое знание представлено в данном случае теоремой, истинность которой доказывается путем построения цепочки умозаключений. Однако при таком подходе необходимо заранее знать весь набор шаблонов, описывающих предметную область. При использовании другого подхода, основанного на примерах (case-based), необходимо только иметь достаточное количество примеров для настройки адаптивной системы с заданной степенью уверенности. Нейронные сети являются классическим примером такого подхода.

## **1.3. Применение нейронных сетей с прямой связью для предсказания поведения стоимости портфеля акций**

Нейронная сеть с прямой связью — это искусственная нейронная сеть, в которой соединения между узлами не образуют цикла [13]. Такая сеть отличается от рекуррентной нейронной сети.

Нейронная сеть с прямой связью была первым и простейшим типом искусственной нейронной сети [14]. В этой сети информация передается только в одном направлении вперед от входных узлов, через скрытые узлы (если таковые имеются) и к выходным узлам. В сети нет циклов или петель обратной связи [15].

*Многослойный персептрон*

Этот класс сетей состоит из нескольких уровней вычислительных блоков, обычно соединенных между собой прямым соединением. Каждый нейрон в одном слое имеет направленные связи с нейронами в следующем слое. Во многих приложениях устройства в этих сетях используют сигмоидальную функцию в качестве функции активации.

Универсальная теорема аппроксимации для нейронных сетей утверждает, что каждая непрерывная функция, отображающая интервалы действительных чисел на некоторый выходной интервал действительных чисел, может быть произвольно аппроксимирована многослойным персептроном только с одним скрытым слоем. Этот результат справедлив для широкого спектра функций активации, например, для сигмовидных функций.

Многослойные сети используют множество методов обучения, самым популярным из которых является обратное распространение. Здесь выходные значения сравниваются с правильным ответом, чтобы вычислить значение некоторой предопределенной функции ошибки. Затем ошибка возвращается по сети различными способами. Используя эту информацию, алгоритм корректирует вес каждого соединения, чтобы уменьшить значение функции ошибки на некоторую небольшую величину. После повторения этого процесса в течение достаточно большого числа циклов обучения сеть обычно сходится к некоторому состоянию, в котором ошибка вычисления невелика. В этом случае мы можем сказать, что сеть усвоила определенную целевую функцию. Чтобы правильно настроить веса, используется общий метод нелинейной оптимизации, называемый градиентным спуском. Чтобы сделать это, сеть вычисляет производную функции ошибки относительно весов сети и изменяет веса таким образом, чтобы ошибка уменьшалась (таким образом, снижаясь по поверхности функции ошибки). По этой причине обратное распространение может использоваться только в сетях с дифференцируемыми функциями активации.

В целом, проблема обучения сети эффективной работе даже на выборках, которые не использовались в качестве обучения, является довольно тонкой проблемой, требующей дополнительных методов. Это особенно важно для случаев, когда доступно лишь очень ограниченное количество обучающих выборок [16]. Опасность заключается в том, что сеть накладывает данные обучения и не может зафиксировать истинный статистический процесс, генерирующий данные. Теория вычислительного обучения связана с обучением классификаторов на ограниченном объеме данных. В контексте нейронных сетей простая эвристика, называемая ранней остановкой, часто гарантирует, что сеть будет хорошо обобщаться на примеры за пределами обучающего набора.

Другими типичными проблемами алгоритма обратного распространения являются скорость сходимости и возможность достижения локального минимума функции ошибки. Сегодня существуют практические методы, которые делают обратное распространение в многослойных персептронах предпочтительным инструментом для многих задач машинного обучения.

Вы также можете использовать серию независимых нейронных сетей, модерируемых каким-либо посредником, аналогичное поведение происходит и в мозге. Эти нейроны могут работать отдельно и справляться с большой задачей, и результаты могут быть окончательно объединены [17].

# **2. Проектирование**

## **2.1. Функциональные требования**

Функциональные требования определяют функциональность программного обеспечения, т.е. описывают, какие возможности должна предоставлять разрабатываемая система. Функциональные требования включают бизнес-требования и требования пользователей. Были определены следующие функциональные требования:

1) разработанная система должна прогнозировать с приемлемой точностью;

2) разработанная система должна сохранить обученную модель для ее дальнейшего использования:

3) разработанная система должна прогнозировать фондовый индекс на минуту вперед.

## **2.2. Нефункциональные требования**

Нефункциональные требования описывают свойства и ограничения, налагаемые на информационную систему. Нефункциональные требования определяют бизнес-правила, системные требования и так далее. Были выбраны следующие нефункциональные требования:

1. система должна быть реализована на языке программирования Python.

## **2.3 Разработка архитектуры сети**

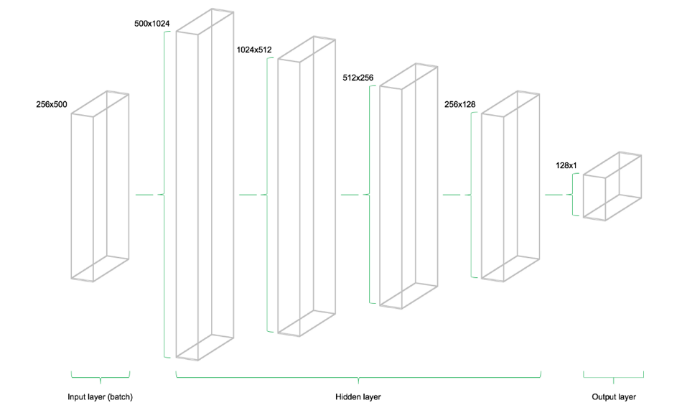
Изображение на рисунке 6 иллюстрирует сетевую архитектуру. Модель состоит из трех основных блоков. Входной уровень, скрытые уровни и выходной уровень. Эта структура называется сетью прямой связи. В упреждающей сети фрагменты данных перемещаются по структуре строго слева направо. Например, в случае рекуррентных нейронных сетей данные могут перемещаться внутри сети в обоих направлениях.

Рисунок 6. Архитектура упреждающей сети

Модель состоит из четырех скрытых уровней. Первый содержит 1024 нейрона, что примерно в два раза больше объема входных данных. Последующие скрытые уровни в два раза меньше предыдущего уровня - они объединяют 512, 256 и 128 нейронов. Уменьшение количества нейронов на каждом уровне приводит к сжатию информации, которую сеть обработала на предыдущих уровнях.

## **2.4. Формирование обучающей выборки**

Экспортируем биржевые данные в csv-файл. Датасет содержит n = 41266 минут данных, охватывающих торги 500 акциями в период с апреля по август 2017, также в него входит информация по цене индекса S&P 500.

На рисунке 7 представлен временной ряд индекса S&P500, построенный с помощью pyplot.plot(data['SP500']):

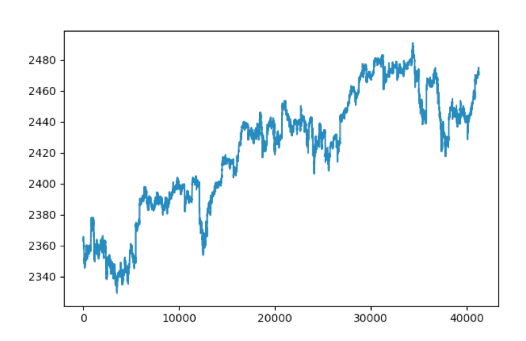


Рисунок 7. временной ряд индекса S&P500

|  |
| --- |
| # Импорт данных  data = pd.read\_csv('data\_stocks.csv')  # Сброс переменной date  data = data.drop(['DATE'], 1)  # Размерность датасета  n = data.shape[0]  p = data.shape[1]  # Формирование данных в numpy-массив  data = data.values |

Листинг 1. Подготовка данных и перевод их в numpy-массив

Поскольку цель состоит в том, чтобы "предсказать" значение индекса в будущем, его значение сдвигается на 60 секунд (одну минуту).

Набор данных был разделен на две части - одна часть для тестирования, а вторая для обучения. Обучающие данные составили 80% от общего объема данных и охватывали период с апреля примерно по конец июля 2017 года, тестовые данные закончились в августе 2017 года (около 20% от общего объема данных).

|  |
| --- |
| # Данные для тестирования и обучения  train\_start = 0  train\_end = int(np.floor(0.8\*n))  test\_start = train\_end  test\_end = n  data\_train = data[np.arange(train\_start, train\_end), :]  data\_test = data[np.arange(test\_start, test\_end), :] |

Листинг 2. Подготовка данных для тестирования и обучения

Существует множество подходов к перекрестной проверке временных рядов, от генерации прогнозов с переоборудованием модели или без него до более сложных концепций, таких как повторная выборка временных рядов начальной загрузки. В последнем случае данные разбиваются на повторяющиеся выборки, начиная с начала сезонной декомпозиции временного ряда - это позволяет моделировать выборки, которые следуют той же сезонной схеме, что и исходный временной ряд, но не полностью копируют его значения.

# **3. Реализация**

## **3.1. Используемые технологии разработки**

Программная система была реализована на языке программирования Python. Для реализации использовались следующие технологии:

1.) TensorFlow – открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронных сетей. Применяется как для исследований, так и для разработки собственных продуктов Google. Основное API для работы с библиотекой реализовано для Python.

2.) Pandas — программная библиотека на языке Python для обработки и анализа данных. Работа pandas с данными строится поверх библиотеки NumPy, являющейся инструментом более низкого уровня. Предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временны́ми рядами.

3.) Scikit-learn - один из наиболее широко используемых пакетов Python для Data Science и Machine Learning. Он позволяет выполнять множество операций и предоставляет множество алгоритмов.

4.) Matplotlib — библиотека на языке программирования Python для визуализации данных двумерной и трёхмерной графикой.

## **3.2. Топология нейронной сети**

Для решения задачи была разработана нейронная сеть, имеющая

описанную выше топологию. Исходный код построения модели сети

приведен в листинге.

|  |
| --- |
| # Инициализация графа  net = tf.compat.v1.Session()  # Параметры архитектуры модели  n\_stocks = 500  n\_neurons\_1 = 1024  n\_neurons\_2 = 512  n\_neurons\_3 = 256  n\_neurons\_4 = 128  n\_target = 1  # Плейсхолдер  X = tf.compat.v1.placeholder(dtype=tf.float32, shape=[None, n\_stocks])  Y = tf.compat.v1.placeholder(dtype=tf.float32, shape=[None])  # Инициализаторы  sigma = 1  weight\_initializer = tf.keras.initializers.VarianceScaling (mode="fan\_avg", distribution="uniform", scale=sigma)  bias\_initializer = tf.zeros\_initializer()  # Уровень 1: Переменные для скрытых весов и смещений  W\_hidden\_1 = tf.Variable(weight\_initializer([n\_stocks, n\_neurons\_1]))  bias\_hidden\_1 = tf.Variable(bias\_initializer([n\_neurons\_1]))  # Уровень 2: Переменные для скрытых весов и смещений  W\_hidden\_2 = tf.Variable(weight\_initializer([n\_neurons\_1, n\_neurons\_2]))  bias\_hidden\_2 = tf.Variable(bias\_initializer([n\_neurons\_2]))  # Уровень 3: Переменные для скрытых весов и смещений  W\_hidden\_3 = tf.Variable(weight\_initializer([n\_neurons\_2, n\_neurons\_3]))  bias\_hidden\_3 = tf.Variable(bias\_initializer([n\_neurons\_3]))  # Уровень 4: Переменные для скрытых весов и смещений  W\_hidden\_4 = tf.Variable(weight\_initializer([n\_neurons\_3, n\_neurons\_4]))  bias\_hidden\_4 = tf.Variable(bias\_initializer([n\_neurons\_4]))  # Уровень выходных данных: Переменные для скрытых весов и смещений  W\_out = tf.Variable(weight\_initializer([n\_neurons\_4, n\_target]))  bias\_out = tf.Variable(bias\_initializer([n\_target]))  # Скрытый уровень  hidden\_1 = tf.nn.leaky\_relu(tf.nn.bias\_add(tf.linalg.matmul (X, W\_hidden\_1), bias\_hidden\_1))  hidden\_2 = tf.nn.leaky\_relu(tf.nn.bias\_add(tf.linalg.matmul(hidden\_1, W\_hidden\_2), bias\_hidden\_2))  hidden\_3 = tf.nn.leaky\_relu(tf.nn.bias\_add(tf.linalg.matmul(hidden\_2, W\_hidden\_3), bias\_hidden\_3))  hidden\_4 = tf.nn.leaky\_relu(tf.nn.bias\_add(tf.linalg.matmul(hidden\_3, W\_hidden\_4), bias\_hidden\_4))  # Выходной уровень (должен быть транспонирован)  out = tf.transpose(tf.nn.bias\_add (tf.linalg.matmul (hidden\_4, W\_out), bias\_out)) |

Листинг 3. Подготовка данных для тестирования и обучения

tf.compat.v1.Session - объект инкапсулирует среду в которой выполняются объекты, а также определены тензоры.

Плейсхолдеры — объекты параметризующие граф и отмечающие места для подстановки внешних значений потом.

Инициализаторы используются для инициализации переменных перед началом обучения. Так как нейронные сети обучаются с помощью численных техник оптимизации, начальная точка оптимизации — это один из важнейших факторов на пути поиска оптимального решения. В TensorFlow существуют различные инициализаторы, каждый из которых использует собственный подход.

tf.keras.initializers.VarianceScaling - инициализатор, способный к адаптации его масштаба к форме тензоров весов.

Таблица 1. Аргументы функции initializers.VarianceScaling

|  |  |
| --- | --- |
| Аргументы | |
| scale | Масштабный коэффициент (положительное число с плавающей запятой). |
| mode | Cреднее число чисел входных и выходных единиц |
| distribution | Использование случайного распределение.  С distribution="uniform", образцы отстоят от равномерного распределения [-limit, limit], где limit = |
| seed | Целое число Python. Инициализатор будет всегда производить тот же случайный тензор для данной формы и типа данных. |

tf.zeros\_initializer - инициализатор, генерирующий тензоры, инициализированные к нулю. Инициализатор позволяет предварительно указывать стратегию инициализации, закодированную в объекте инициализатора, не зная форму и тип инициализируемой переменной.

tf.linalg.matmul - умножает матрицу А на матрицу В и создает новую матрицу А\* В.

tf.nn.bias\_add - добавляет смещение к значению.

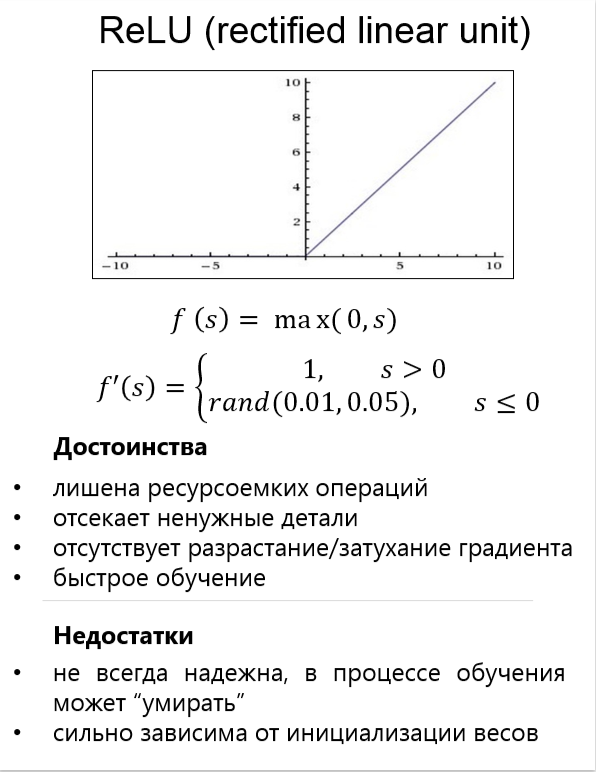


Рисунок 8. Описание функции активации ReLU

tf.nn.leaky\_relu - функция активации Leaky ReLU или LReLU - это тип функции активации, которая похожа на ReLU, но решает проблему "умирающих" нейронов. Чтобы решить проблему нулевого градиента для отрицательного значения, Leaky ReLu дает чрезвычайно малую линейную составляющую Х отрицательным входам.

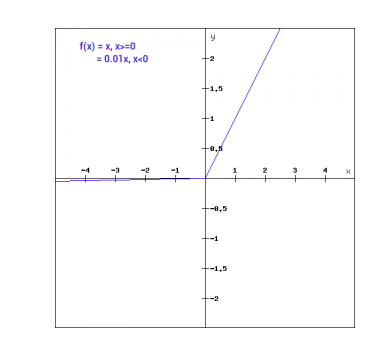


Рисунок 9. Графическое описание Leaky ReLu

## **3.3 Функция стоимости**

Функция стоимости сети используется для генерации оценки отклонения между прогнозами сети и реальными результатами наблюдений в ходе обучения. Для решения проблем с регрессией используют функцию средней квадратичной ошибки (mean squared error, MSE). Данная функция вычисляет среднее квадратичное отклонение между предсказаниями и целями, но вообще для подсчета отклонений может быть использована любая дифференцируемая функция.

|  |
| --- |
| # Функция стоимости  mse = tf.math.reduce\_mean (tf.math.squared\_difference(out, Y)) |

Листинг 4. Код функции стоимости

MSE отображает конкретные сущности, которые полезны для решения общей проблемы оптимизации.

tf.math.squared\_difference - нахождение квадрата разности между двумя тензорами.

tf.math.reduce\_mean - вычисляет средние значения из элементов через размеры тензора.

## **3.4 Оптимизатор**

Оптимизатор выполняет необходимые вычисления, требующиеся для адаптации весов и переменных отклонений нейросети в ходе обучения. Данные вычисления выполняют подсчет градиентов, обозначающие направление необходимого изменения отклонений и весов для минимизации функции стоимости.

|  |
| --- |
| # Оптимизатор  opt = tf.compat.v1.train.AdamOptimizer().minimize(mse) |

Листинг 5. Код оптимизатора

tf.compat.v1.train.AdamOptimizer - это оптимизатор, реализующий алгоритм Адама в TensorFlow. Адам, или адаптивная оценка момента — это алгоритм оптимизации, находящий лучшую глобальную точку, и вводит вторичную коррекцию градиента. Он сочетает в себе идеи RMSProp и оптимизатора импульса. Вместо того чтобы адаптировать скорость обучения параметров на основе среднего первого момента (среднего значения), как в RMSProp, Adam (аббревиатура “Adaptive Moment Estimation”) также использует среднее значение вторых моментов градиентов. В частности, алгоритм вычисляет экспоненциальное скользящее среднее градиента и квадратичный градиент.

## **3.5 Настройка нейросети**

После определения плейсхолдеров, переменных, инициализаторов, функций стоимости и оптимизаторов, обучают модель. Для этого применяется подход мини-партий (minibatch training). В ходе обучения из набора данных для обучения отбираются случайные семплы данных размера n = batch\_size и загружаются в нейросеть. Набор данных для обучения делится на n / batch\_size кусков, которые затем последовательно отправляются в сеть. В этот момент в игру вступают плейсхолдеры X и Y. Они хранят входные и целевые данные и отправляют их в нейросеть.

Семплированные данные X проходят по сети до достижения выходного уровня. Затем TensorFlow сравнивает сгенерированные моделью прогнозы с реально наблюдаемыми целями Y в текущем «прогоне». После этого TensorFlow выполняет этап оптимизации и обновляет параметры сети, после обновления весов и отклонений, процесс повторяется снова для нового куска данных. Процедура повторяется до того момента, пока все «нарезанные» куски данных не будут отправлены в нейросеть. Полный цикл такой обработки называется «эпохой».

Обучение сети останавливается по достижению максимального числа эпох.

|  |
| --- |
| # Количество эпох и размер сэмпла данных  epochs = 10  batch\_size = 256  mse\_train = []  mse\_test = []  for e in range(epochs):  # Перемешивание данных для обучения  shuffle\_indices = np.random.permutation(np.arange(len(y\_train)))  X\_train = X\_train[shuffle\_indices]  y\_train = y\_train[shuffle\_indices] |

Листинг 6. Код настройки нейросети

Функция random.permutation() возвращает случайную перестановку элементов массива или случайную последовательность заданной длинны из его элементов.

## **3.6 Окно результата работы программы**

В ходе обучения оценивались предсказания, сгенерированные сетью на тестовом наборе, затем осуществлялась визуализация.

|  |
| --- |
| # Запустить оптимизатор пакетов  net.run(opt, feed\_dict={X: batch\_x, Y: batch\_y})  # Показать прогресс  if np.mod(i, 5) == 0:  # Среднеквадратичная ошибка обучающей и тестовой выборки  mse\_train.append(net.run(mse, feed\_dict={X: X\_train, Y: y\_train}))  mse\_test.append(net.run(mse, feed\_dict={X: X\_test, Y: y\_test}))  print('MSE Train: ', mse\_train[-1])  print('MSE Test: ', mse\_test[-1])  # Предсказания  pred = net.run(out, feed\_dict={X: X\_test})  line2.set\_ydata(pred)  plt.title('Epoch ' + str(e) + ', Batch ' + str(i))  file\_name = 'epoch\_' + str(e) + '\_batch\_' + str(i) + '.jpg'  #plt.savefig(file\_name)  plt.pause(0.01)  # Вывести финальную фукнцию MSE после обучения  mse\_final = net.run(mse, feed\_dict={X: X\_test, Y: y\_test})  print(mse\_final) |

Листинг 7. Вывод результатов работы

# **4. Тестирование**

Тестирование – это процесс, который заключается в проверке соответствия программного продукта заявленным характеристикам и требованиям, требованиям эксплуатации в различных окружениях, с различными нагрузками, требованиями по безопасности и удобству использования [5].

В ходе обучения оценивались предсказания, сгенерированные сетью на тестовом наборе, затем осуществлялась визуализация.

## **4.1. Функциональное тестирование**

Функциональное тестирование - это тестирование программного обеспечения с целью проверки выполнимости функциональных требований, то есть способности программного обеспечения при определенных условиях решать задачи, необходимые пользователям. Функциональные требования определяют, что именно делает программное обеспечение, какие задачи оно решает [2].

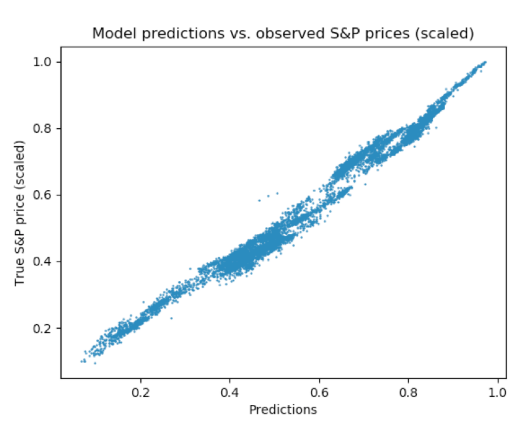
Рисунок 10 представляет собой точечный график между прогнозируемыми и фактическими ценами S&P в том виде, в каком он был подготовлен. Для этого обучающая выборка была программно разделена в соотношении 4 к 1 для каждого класса изображений. На 80% из них модель была обучена, на 20% она была протестирована. Для обучения было решено установить 10 эпох, поскольку дальнейшее увеличение количества эпох не принесло заметного увеличения точности распознавания.

Рисунок 10. График рассеяния между предсказанными и реальными ценами S&P

Нейронная сеть адаптируется к базовой форме временного ряда и ищет наилучшие шаблоны данных. Через 10 эпох мы получаем результаты, которые очень близки к тестовым данным. Конечное значение функции MSE равно 0,00142 (цели масштабированы). Средняя абсолютная процентная ошибка прогноза по тестовому набору составляет 5,31%.

# **Заключение**

В ходе разработки системы для анализа котировок валют были получены следующие результаты:

* произведен анализ предметной области;
* изучены теоретические основы анализа котировок валют с помощью искусственных нейронных сетей;
* спроектирована архитектура нейронной сети;
* реализовано приложение для анализа котировок валют;
* сформирована обучающая выборка;
* проведено тестирование системы.

# **Список литературы**

1. Арлоу Д., Нейштадт А. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-е изд. – Пер. с англ. – СПб.: Символ-Плюс, 2007. – 624 с.

2. Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. – СПб.: Питер, 2004.–544 с.

3. Заенцев И. Нейронные сети: основные модели. Учебное пособие к курсу «Нейронные сети» для студентов 5 курса магистратуры Воронежского Государственного университета. – Воронеж: Воронежский Государственный университет, 1999. – 76 с.

4. Каллан Р.Основные концепции нейронных сетей. – СПб.: Вильямс, 2003. – 288 c.

– Криспин Л., Грегори Д. Гибкое тестирование. Практическое руководство для тестировщиков ПО и гибких команд. – СПб.: Вильямс, 2010. 365 с.

5. Круг П.Г. Нейронные сети и нейрокомпьютеры: Учебное пособие по курсу «Микропроцессоры». – М.: Национальный исследовательский университет «МЭИ», 2002. – 176 с.

6. Осовский С. Нейронные сети для обработки информации. – М.: Финансы и статистика, 2002. — 344 с.: ил. — Пер. с польского И.Д. Рудинского.

7. Розенблатт Ф. Принципы нейродинамики. Персептроны и теория механизмов мозга. – М.: Мир, 1965. – 169 с.

8. Уоссермен Ф. Нейрокомпьютерная техника: Теория и практика. М.: Мир, 1992. – 184 с.

9. Флах П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных. – М.: ДМК Пресс, 2015. 400 с.

10. Фролов А.А., Муравьев И.П. Нейронные модели ассоциативной памяти. – М.: Наука, 2004. – 160 с.

11. Хайкин С. Нейронные сети: полный курс. – М.: Вильямс, 2006. 1104с.

12. Tensorflow Documentation [Электронный ресурс] URL: https://www.tensorflow.org/ (дата обращения 24.02.2023).