МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э.
Баумана
(национальный исследовательский университет)»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по курсу

«13377. Архитектор данных (Data Architect Pro)»

Слушатель Устинов Д.С.

Москва, 2025

Содержание

Введение

В настоящее время на мировом финансовом рынке представлено большое количество инструментов, доступных для широкого ряда инвесторов, из которых участники рынка составляют инвестиционные портфели.

В последние несколько лет наблюдаются значительные колебания цен разнообразных активов. Падение и рост стоимости акций компаний, изменения основных биржевых индексов, процентных ставок и курсов валют.

Рост волатильности оказывает существенное влияние на работу финансовых институтов. В этих условиях их требования к управлению финансовым риском постоянно возрастает. В связи с этим растет и актуальность методов и моделей управления риском.

Модели с использованием машинного обучения становятся наиболее популярными в связи с развитием вычислительных технологий. Искусственные нейронные сети позволяют производить эффективные вычисления на больших объемах данных. Нейронные сети позволяют кардинально уменьшить время вычислений в сравнении с консервативными финансовыми методам, поэтому разработка нейросетевых моделей в финансах является крайне актуальной задачей.

Аналитическая часть

1. Постановка задачи

Цель данной работы заключается в исследовании возможности применения нейронных сетей для решения задачи управления риском портфеля ценных бумаг с помощью дельта хеджирования опционами.

2. Описание используемых методов.

В рамках выполнения работы планируется последовательно реализовать следующие шаги:

1) Теоретический обзор определения цены опционов и коэффициента Хеджирования;
2) Провести сбор данных. Для этого планируется использовать API MOEX (Московская биржа), для получения исторических данных о цене акций. Процесс сбора состоит из двух частей:
   - Получение всех исторических данных через API;
   - Ежедневное получение новых данных.
3) Создание нейронной сети определенной архитектуры, метода ее обучения и тестирования;
4) Реализация модели машинного обучения для определения коэффициента хеджирования с использованием языка программирования Python3;
5) Сохранение полученной модели;
6) Проведение вычислительного эксперимента для базового портфеля ценных бумаг, визуализация результатов и оценка проведенных исследований.

3. Разведочный анализ данных

3.1. Теоретическая часть

Рассматривая задачу управления рыночным риском портфеля ценных бумаг с помощью дельта хеджирования опционами, необходимо определить, что такое опционы.

3.1.1. Опционы

Опцион — это производный финансовый инструмент, который дает его владельцу право, но не обязанность купить или продать (в зависимости от типа опциона) соответствующий базовый актив по определенной фиксированной цене, которая устанавливается в момент покупки опционного контракта. Поскольку владение опционным контрактом выгодно благодаря возможности купить (или продать) ниже (или выше) рыночной цены базовый актив, опционный контракт имеет цену (премию).

Существует два основных типа опционов:

1) Опцион колл, который дает владельцу право, но не обязанность купить базовый актив по определенной цене, называемой ценой исполнения;
2) Опцион пут, который дает владельцу право, но не обязанность продать базовый актив по определенной цене исполнения.

Опционы могут быть исполнены в определенную дату, называемую датой погашения или истечения срока действия, или раньше, в зависимости от их характеристик. В зависимости от времени исполнения опционы бывают европейские или американские. Основное различие заключается между ними заключается в том, что европейские имеют фиксированную дату погашения, в то время как американские опционы могут быть исполнены в любое время до наступления срока погашения, что обычно подкрепляется более высокой премией. Американские опционы

также имеют более высокие риски для продавцов опционов из-за досрочного исполнения, которое может быть осуществлено держателем опциона.

На основании фактической стоимости базового актива опцион может быть классифицирован как "в деньгах", "при деньгах" или "вне денег", что зависит от относительной цены базового актива к цене исполнения. Опцион считается находящимся «в деньгах» (ITM), если разница между ценой базового актива и ценой исполнения положительна в случае опционов колл и отрицательна в случае опционов пут, что означает, что опцион имеет внутреннюю стоимость, которая будет получена в виде прибыли, если опцион будет немедленно исполнен.

Опцион называется «при деньгах» (ATM), если цена базового актива точно соответствует цене исполнения опциона и, следовательно, является точкой безубыточности между отсутствием внутренней стоимости и положительной внутренней стоимостью опциона.

Опцион считается «вне денег» (OTM), если цена исполнения превышает цену базового актива в случае опционов колл и не превышает цену базового актива в случае опционов пут. Такой опцион не имеет внутренней стоимости, и поскольку премия опциона не является отрицательной, прибыль при исполнении опциона будет отрицательной.

Практически торговля производными опционами похожа на торговлю фьючерсами в том смысле, что всегда есть один покупатель (длинный контракт) и один продавец контракта (короткий контракт). Продавец обязан выполнить право покупателя всякий раз, когда покупатель исполняет опцион, что является преимуществом для держателя опциона в случае американских опционов (большинство опционов на акции), который может выбрать оптимальное время для исполнения опциона.

Информация о цене опциона необходима, так как с ее помощью определяется коэффициент хеджирования. Теоретическая цена опционов в данной работе будет определяться с помощью модели Блэка-Шоулза.

3.1.2. Модель Блэка-Шоулза

Поскольку опционы широко используются в финансовом секторе, возникает большая потребность в методике оценки их стоимости. Задача ценообразования опционов часто решается путем применения формулы Блэка-Шоулза.

Модель Блэка-Шоулза, опубликованная учеными Фишером Блэком и Майроном Шоулзом в 1973 году, являлась новаторской для своего времени, сейчас воспринимается как классическая и наиболее распространенная модель для оценки премии опционов европейского типа. При разработке модели Блэк и Шоулз сделали несколько предположений:

1) Краткосрочный процент и дисперсионная ставка доходности акции известны и постоянны;
2) По акции не выплачиваются дивиденды;
3) Опцион является европейским и может быть исполнен только в момент погашения;
4) При покупке или продаже опциона нет транзакционных издержек;
5) Отсутствует возможность арбитража;
6) Модель Блэка-Шоулза является параболическим

дифференциальным уравнением и имеет следующий вид:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + r\frac{\partial V}{\partial S} - rV = 0,$$

Где V - цена опциона в зависимости от цены базового актива;

σ – стандартное отклонение доходности акций;

S - цена базового актива;

r - постоянно начисляемая процентная ставка;

t - время.

Производная, полученная при решении этого уравнения, зависит от используемых граничных условий, которые, в случае европейского опциона колл, имеют вид:

$$f = \max(S - X, 0)$$

Как объясняется в Hull (2003), портфель, используемый при выводе дифференциального уравнения, не является постоянно безрисковым. Поэтому необходимо минимизировать риск на рынке опционов путем реализации стратегии дельта-хеджирования, целью которой является снижение риска, связанного с движением цен на базовый актив.

Существует два способа вывести формулу Блэка-Шоулза-Мертона:

1) Путем решения дифференциального уравнения с учетом граничного условия;
2) Путем использования подхода риск-нейтральной оценки.

Первый метод приводит к следующей формуле Блэка-Шоулза, полученной для европейского колл опциона:

$$C(S, T) = N(d_1)S - N(d_2)Xe^{-rT},$$

Где C - цена опциона колл;

S – цена базового актива;

T - время до погашения;

N(d) - кумулятивная функция стандартного нормального распределения;

X - цена исполнения опциона;

r - безрисковая процентная ставка.

$$d_1 = \frac{\ln \frac{S}{X} + \left(r + \frac{\sigma^2}{2}\right) T}{\sigma\sqrt{T}},$$
$$d_2 = d_1 - \sigma\sqrt{T}.$$

Где σ – оценка волатильности базового актива.

Эта модель широко используется из-за своей простоты, однако на практике существуют некоторые допущения, которыми модель ограничена. Во-первых: модель недооценивает возможность экстремальных движений рынка и поэтому пренебрегает возможностью больших изменений. Во-вторых: она предполагает мгновенное и свободное исполнение торгов, что на практике также нарушается и порождает риск ликвидности. В-третьих: модель также предполагает, что процесс является стационарным, что также приводит к риску волатильности. И последнее, но не менее важное, это предположение о непрерывном времени и непрерывной торговле, которое обычно не выполняется.

3.1.3. Греческие символы опционов и дельта хеджирование

Задача ценообразования опционов может быть проанализирована также с помощью анализа так называемых греков. В математических финансах их часто описывают как чувствительность цены опционов при изменении некоторого параметра, от которого зависит цена.

Эти чувствительности используются, в частности, в управлении рисками. Их как в модели Блэка-Шоулза, так и в моделях, описываемых искусственными нейронными сетями, легко получить прямым вычислением или с помощью численных производных. Наиболее важной греческим символов опционов является дельта, определяемая для колл опционов как:

$$\Delta = \frac{\partial c}{\partial S},$$

Где c – стоимость опциона;

S – стоимость базового актива.

Дельта измеряет скорость изменения цены опциона в зависимости от стоимости базового актива. Коэффициент дельта для опционов колл лежит в интервале [0, 1].

Хеджирование — это инвестиции, которые осуществляются с целью снижения риска неблагоприятного изменения цен на актив. Как правило, хеджирование заключается в создании компенсирующей или противоположной позиции в соответствующей ценной бумаге.

Наиболее распространённым способом хеджирования являются деривативы. В данной работе рассматривается хеджирование опционами. Эффективность хеджирования деривата выражается в дельте, которую называют коэффициентом хеджирования – соотношение изменчивости доходности хеджируемого портфеля и изменчивости доходности хеджирующего инструмента. Основной целью дельта хеджирования является установление дельта нейтральности портфеля ценных бумаг.

В связи со сказанным выше, рассматривая задачу управления рыночным риском портфеля ценных бумаг при помощи машинного обучения, основная задача данной работы сводиться к возможности определения нейросетевой моделью коэффициента хеджирования для заданного портфеля и сравнение точности оценки нейронной сети с теоретически полученными значениями.

3.2. Машинное обучение в финансах

Искусственные нейронные сети — это объекты обработки данных, которые обладают свойствами и функциями обработки информации, аналогичными биологическим нейронным сетям, т.е. мозгу человека или животного. Использование структуры, подобной мозгу, в вычислении различных задач всегда было востребовано благодаря ее адаптивности и свойствам решения проблем.

3.2.1. Нейронные сети

Мозг состоит примерно из 100 миллиардов нервных клеток, называемых нейронами. Эти клетки отвечают за сбор, обработку и передачу электрических сигналов. Способность мозга обрабатывать информацию обеспечивается сетями таких нейронов.

Нейрон состоит из следующих основных частей в соответствии с рисунком 1:

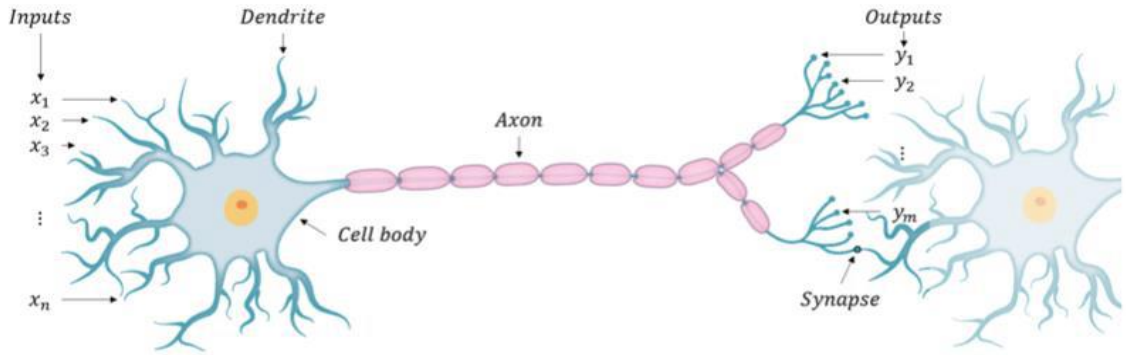


Рисунок 1 - Биологический нейрон

- Тело клетки: содержит ядро клетки и осуществляет биохимические преобразования, необходимые для жизни нейрона.

• Дендриты: эта корне-подобная структура отходит от тела клетки, образуя древовидную структуру. Каждый дендрит соединяется с другими нейронами, чтобы обеспечить вход для клетки.

• Аксон: это длинная, тонкая, трубчатая структура, отвечающая за передачу сигнала, генерируемого нейроном.

• Синапс: Соединение между нейронами образует сложное пространственное расположение.

Когда аксон достигает конечного пункта назначения, он начинает разветвляться на очень сложные и специализированные структуры, называемые синапсами. Эти структуры отвечают за связь между клетками для передачи сигнала от одного нейрона к другому. Таким образом, активность нейрона начинается, когда дендриты получают входящий сигнал через синапсы других нейронов. Клеточное тело со временем обрабатывает входные сигналы и преобразует обработанное значение, чтобы сформировать выходной сигнал. Наконец, этот сигнал посылается другим нейронам через аксон и синапсы.

Мозг принято считать "черным ящиком", многие функции которого до конца не изучены. Он получает входные сигналы и возвращает выходной ответ, который в качестве побочного действия укрепляет связи между определенными нейронами, которые часто использовались. Мозг обычно принимает решение о достижении определенных целей наиболее вероятным способом выполнения поставленной задачи, что ошибочно дает представление о том, что мозг работает как единое целое.

Нейропсихология и нейробиология свидетельствуют о том, что некоторые части мозга не зависят друг от друга, а некоторые специфические функции мозга могут выполняться даже без связи с другими частями. Нейробиологи давно считают, что отдельные части мозга выделяются в отдельные модули, взаимно сотрудничающие друг с другом, а достижения нейробиологии подтверждают эти мысли и указывают на то, что мозг человека и животных состоит из различных

модулей с различным физиологическим составом, которые отвечают за различные задачи.

Это является веским аргументом в пользу того, что имеет смысл применять искусственные нейронные сети для решения проблем с размытыми формами результатов без какой-либо конкретной легко просматриваемой взаимосвязи, поскольку именно таким естественным образом мозг обрабатывает информацию.

Первые архитектуры нейронных сетей были разработаны МакКаллохом и Питтсом (1943), где авторы представили модель того, как реальные биологические нейроны связаны и работают вместе в мозге млекопитающих. Более того, Хебб (1949) предложил теорию о том, как биологические нейроны в мозге адаптируются в процессе обучения.

Простейшая форма нейронной сети состоит из одного слоя с одним нейроном и называется персептрон, представленный Розенблаттом (1958), который вдохновлен биологическим нейроном. Структура персептрона представлена на рисунке 2.

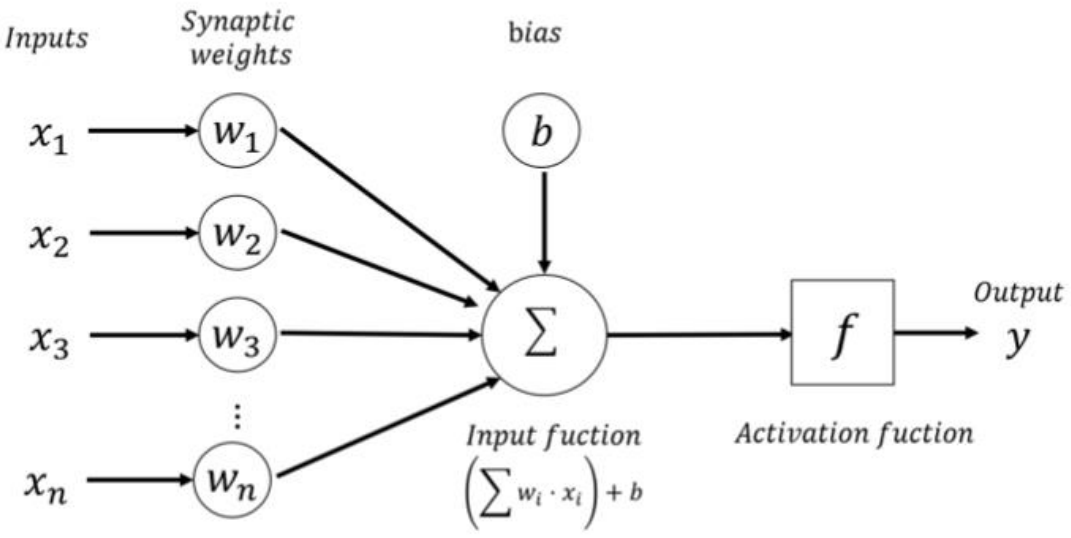


Рисунок 2 - Персептрон

Где:

- Входы x1, x2, x3, xn представляют собой независимые переменные из одного наблюдения;
- Синаптические веса w1, w2, w3, wn представляют силу каждого входа.

Эти веса умножаются на соответствующий вход. Функция входа агрегирует информацию входов, наиболее распространенным является суммирование. Последним параметром является смещение, b, которое позволяет добавить регулируемое смещение.

Функция активации генерирует выходной сигнал y на основе результатов, полученных от входной функции.

На основе перцептрона Ивахненко (1970) разработал более сложную модель, названную многослойным перцептроном.

Многослойный персептрон состоит из нескольких слоев нейронов и также известен как сеть с прямой связью, и считается полностью связанным, если все нейроны связаны друг с другом в соответствии с рисунком 3:

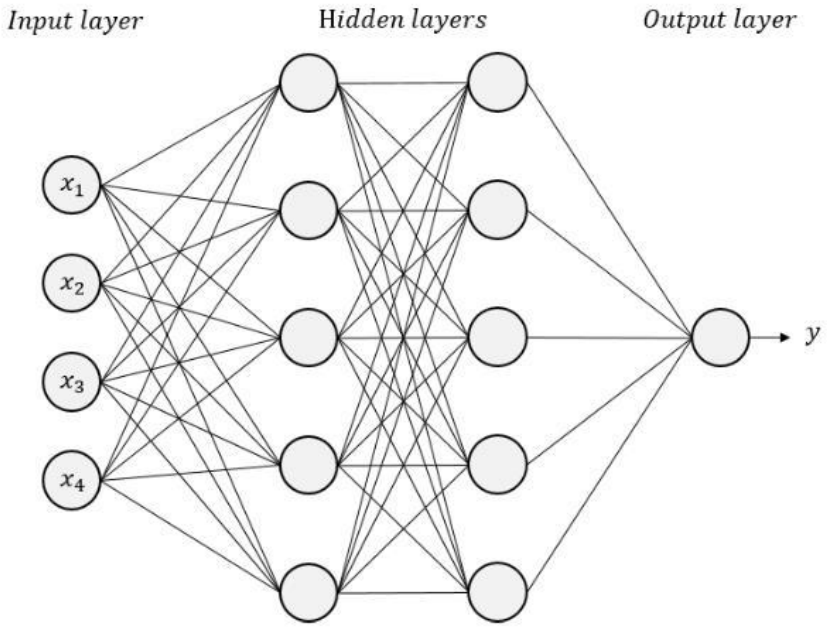


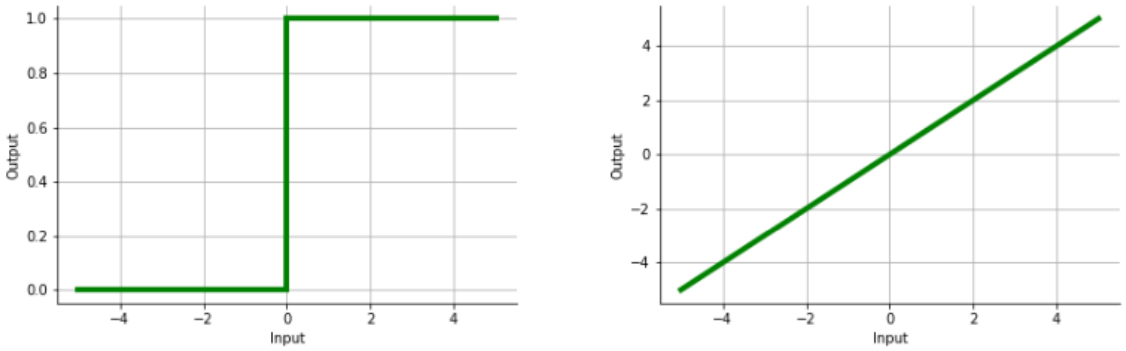
Рисунок 3 - Многослойный персептрон

Слои делятся на:

- Первый слой, который также известен как входной слой. Он состоит из входного вектора. Все элементы входного слоя подключены к каждому нейрону первого скрытого слоя. Более того, вход каждого нейрона в первом скрытом слое представляет собой взвешенную сумму всех входов плюс член смещения;
- Последний слой состоит из выходного слоя. Выходы всех нейронов последнего скрытого слоя соединены с нейронами выходного слоя;
- Слои между ними — это скрытые слои. Между входным и выходным слоями в многослойном персептроне может быть произвольное количество скрытых слоев и нейронов.

Функция активации является важнейшей частью нейронов, поскольку она отвечает за управление выходом нейрона. Функция активации принимает на вход результаты входной функции и преобразует их в выход нейрона.

Первоначальная функция активации, используемая в персептроне, была двоичная ступенчатая функция (Рисунок 4а), которая основана на пороге. Это означает, что нейрон посылает сигнал на следующий слой в зависимости от того, выше или ниже определенного порога входное значение. Недостатком этой функции активации является то, что выход ограничен только двумя возможными значениями, поэтому ее нельзя использовать для решения задач классификации с несколькими категориями.



(a) Binary Step Function (b) Linear Function

Рисунок 4 - Простейшие функции активации

Простейшей функцией активации является линейная функция (Рисунок 4 b), где входы линейно преобразуются для вычисления выхода. Модель многослойного персептрона, использующая линейные функции активации, эквивалентна модели линейной регрессии. Поэтому такая модель будет обладать ограниченной мощностью и ограниченной способностью обрабатывать сложные данные.

С одной стороны, линейная функция активации лучше, чем ступенчатая функция, поскольку позволяет использовать несколько выходов. С другой стороны, линейная функция активации имеет две проблемы:

- Невозможность использования алгоритма обратного распространения ошибки для обучения модели. В действительности, производная линейной функции постоянна, поэтому невозможно понять, какие веса во входных нейронах могут обеспечить лучшее предсказание;
- Линейная комбинация линейных функций все равно остается линейной функцией. При построении многослойной сети, в которой используется линейная функция активации, выход сети будет линейной функцией входа. Это означает, что многослойная сеть будет эквивалентна сети с одним слоем.

В более современных моделях нейронных сетей, как правило, используются нелинейные функции активации, поскольку они позволяют нейронной сети генерировать лучшие и более сложные отображения между входами и выходами сети. Это связано с тем, что нелинейная функция решает проблему линейной функции активации. Поэтому:

- Можно применять метод обратного распространения ошибки, поскольку производная функция нелинейных активаций связана со входами;
- Допускается использование нескольких скрытых слоев нейронов, что приводит к тому, что модель может обучаться сложным наборам данных с высоким уровнем точности.

Нелинейные функции активации:

1) ReLU (Линейный выпрямитель).

Эта функция, впервые представленная Hahnloser (2000), является линейной для значений больше нуля, а для отрицательных значений выход всегда равен нулю. Это означает, что функция активации ReLU обладает некоторыми свойствами линейной функции активации при обучении нейронной сети методом обратного распространения, но не совсем похожа на линейную функцию активации, поскольку производная не является постоянной (Рисунок 5).

$$\mathrm{ReLU(x)} = \begin{cases} x, x > 0 \\ 0, \mathrm{x} \leq 0 \end{cases}$$

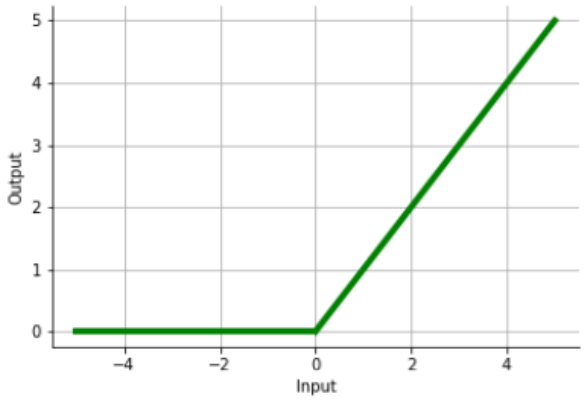


Рисунок 5 - Функция активации ReLU

С другой стороны, недостатком метода ReLu является случай, когда вход приближается к нулю. Сеть не может выполнять обратное распространение из-за нулевого градиента, и, следовательно, не может обучаться. Эта проблема также известна как проблема умирающего ReLu.

2) Leaky ReLU (Линейный выпрямитель с утечкой).

Функция Leaky ReLU похожа на функцию ReLU с той разницей, что в отрицательной части имеет небольшой положительный наклон, который помогает процессу обучения, поскольку решает вопрос, касающийся проблемы умирающего ReLU (Рисунок 6).

$$\text{Leaky ReLU(x)} = \begin{cases} x, x > 0 \\ ax, \text{x} \leq 0 \end{cases}$$

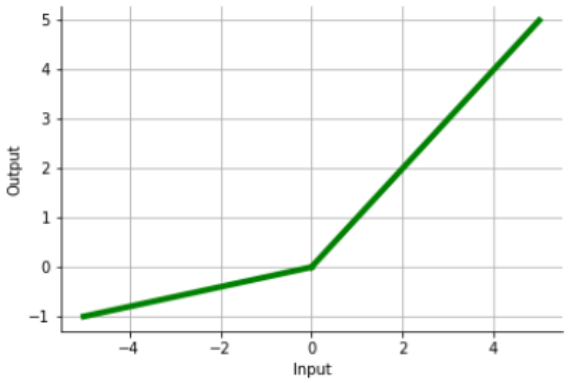


Рисунок 6 - Функция активации Leaky ReLU

Как следует из уравнения, любое значение x сопоставляется с тем же значением y, если значение x больше 0. Но, если значение x меньше 0, мы имеем коэффициент альфа, который равен 0,2. Это означает, что если входное значение x равно 5, то выходное значение, которому оно соответствует, равно 1.

3) ELU (Экспоненциальный линейный выпрямитель).

Функция активации ELU аналогична функции активации Leaky ReLu с той разницей, что отрицательная часть, вместо того чтобы быть линейной функцией с положительным наклоном, является экспоненциальной функцией (Рисунок 7).

$$\text{Leaky ReLU(x)} = \begin{cases} x, x > 0 \\ \text{a}(e^x - 1), \text{x} \leq 0 \end{cases}$$

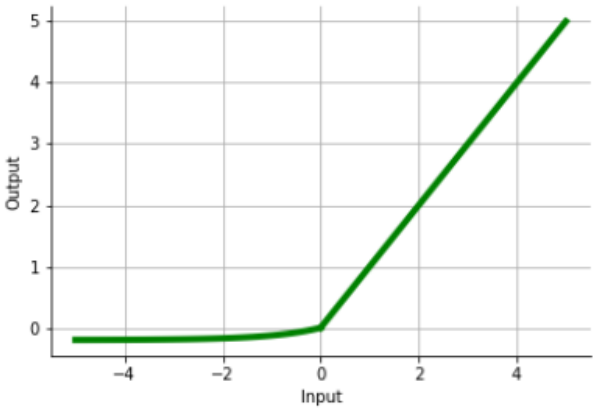


Рисунок 7 - Функция активации ELU

Поэтому в данном случае, если входное значение меньше 0, производная не является постоянной, модель лучше обучается. ELU требует больших вычислительных затрат.

Алгоритм обратного распространения ошибки был предложен Румельхартом. Обучение нейронной сети зависит от настройки всех

параметров модели с целью извлечения ценной информации из входных данных. Простейшим методом будет использование метода Брут Форса (перебора) для поиска оптимальных весов для сети. Однако для более сложных моделей нейронных сетей такой подход будет вычислительно затратным из-за большого количества параметров, которые необходимо регулировать. В настоящее время большинство нейронных сетей используют алгоритм обратного распространения ошибки для фазы обучения из-за его эффективности.

Рабочий процесс алгоритма можно описать следующим образом:

1) Первый шаг заключается в построении нейронной сети. Это означает определение количества слоев, количества нейронов, функции активации для каждого из слоев, а также входных и выходных параметров (Рисунок 8).

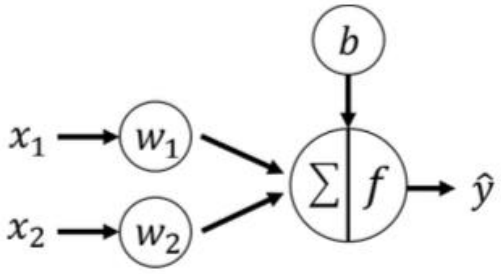


Рисунок 8 - Построение нейронной сети

2) После определения всех параметров нейронной сети необходимо инициализировать веса и погрешности. Этот шаг очень важен, поскольку неправильная инициализация может негативно повлиять на производительность модели. Чтобы избежать этого, значения весов и смещений задаются случайным образом. Обычно эти значения находятся в диапазоне от 0 до 1 (Рисунок 9).

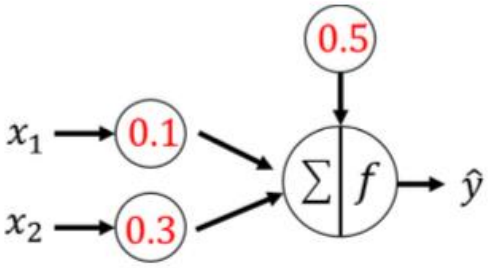


Рисунок 9 - Инициализация параметров нейронной сети

3) После инициализации весов можно начинать процесс обучения. Этот процесс начинается с выбора входных данных, которые затем подаются в сеть и проходят через все нейроны и их соединения (Рисунок 10).

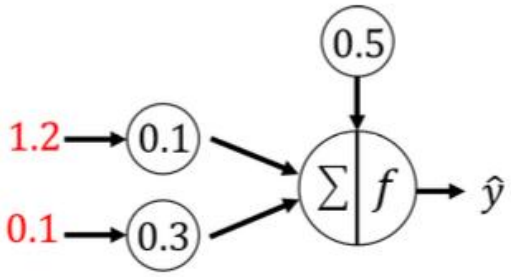


Рисунок 10 - Подача данных для прямого прохода

4) В результате прохождения данных через нейроны модель выдаст предсказанное значение y (Рисунок 11).

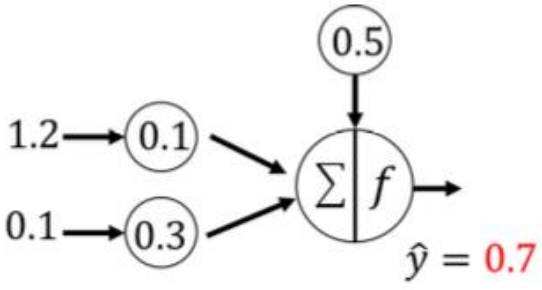


Рисунок 11 - Получение выходного значения

5) Во время фазы обучения для каждого набора входов доступен ожидаемый выход y. Это позволяет использовать функцию ошибки для вычисления разницы между прогнозируемым и фактическим значением. Эта ошибка также известна как "потеря" и используется для настройки параметров модели (Рисунок 12).

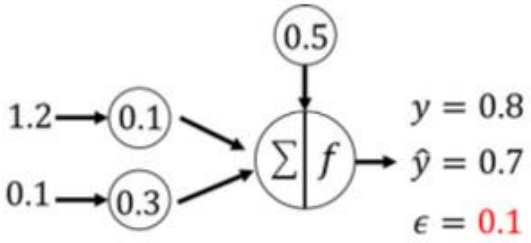


Рисунок 12 - Определение ошибки между предсказанным и ожидаемым значением

6) Для того чтобы отрегулировать веса и смещения сети, сначала необходимо определить вес (влияние) каждого параметра в ошибку. Для этого используется оптимизатор, который вычисляет частичную производную выхода по отношению к параметру каждого нейрона. Затем производная используется для расчета необходимой корректировки. Среди распространенных оптимизаторов - стохастический градиентный спуск и ADAM (Рисунок 13).

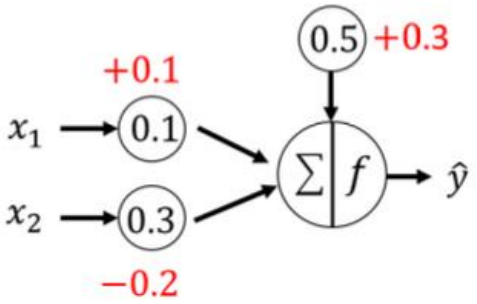


Рисунок 13 - Расчет значений для регулировки параметров модели посредством оптимизатора

7) На последнем этапе обновляются веса и смещения, и происходит следующий прямой проход. Хотя параметры могут обновляться при каждом новом вводе, это обычно нецелесообразно, поскольку при большом наборе данных это потребует больших вычислительных затрат. Общепринятой практикой является группировка образцов в партии и обновление весов только после обработки партии. Размер партий, а также количество эпох (количество раз, когда обрабатывается набор данных) являются важными гипер-параметрами, которые необходимо настроить для обеспечения наилучшей производительности модели (Рисунок 14).

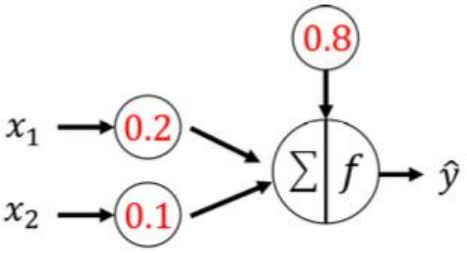


Рисунок 14 - Обновленные веса и погрешности нейронной модели

Как было описано выше, функции потерь применяются для обучения нейронных моделей. Функции потерь являются методом оценки

того, насколько хорошо конкретный алгоритм моделирует заданные данные. Если прогнозы слишком сильно отклоняются от реальных результатов, функция потерь выдаст очень большое число. Постепенно оптимизатор с помощью функции потерь учится уменьшать ошибку в предсказании модели.

В целом, функции потерь можно разделить на две основные категории в зависимости от типа задачи обучения - потери при регрессии и потери при классификации. В данной работе при предсказании цены опционов и определения дельта коэффициента необходимо использовать регрессионные функции потерь.

Наиболее распространёнными функциями являются:

• Средняя абсолютная ошибка (MAE):

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

где N — число примеров обучающей выборки;

y_i — целевое значение i-го примера;

$\hat{y}i$ — предсказанное моделью значение.

• Среднеквадратичная ошибка (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

где n — число примеров обучающей выборки.

• Коэффициент детерминации (R^2):

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y})^2}{\sum_i (y_i - \bar{y})^2}$$

В процессе обучения нейронной сети главная задача заключается в подборе наиболее удачных параметров модели, при которых функция потерь будет определять наименьшее значение. Для этого в нейросетевых моделях используются алгоритмы оптимизации.

Градиентный спуск — это самый распространенный алгоритм, используемый для минимизации целевой функции путем итеративного движения в направлении наиболее крутого спуска, определяемого отрицательной величиной градиента (Рисунок 15).

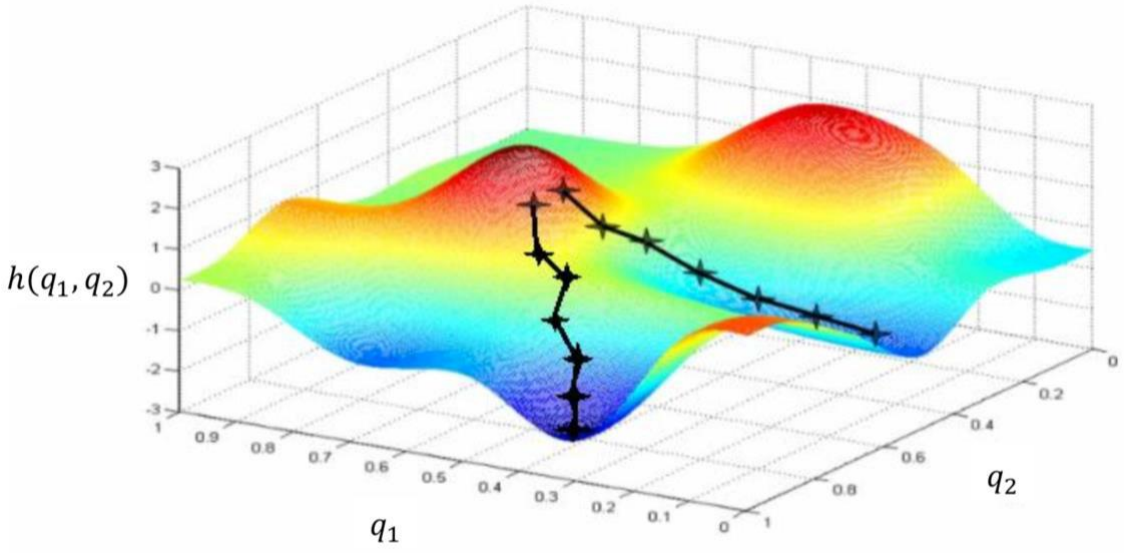


Рисунок 15 - Градиентный спуск

Существует три популярных типа градиентного спуска, отличия между которыми заключаются в количестве данных, которые используются:

1) Пакетный градиентный спуск (Batch Gradient Degree), также называемый ванильным градиентным спуском, рассчитывает ошибку для каждого примера в наборе обучающих данных, но только после того, как все обучающие примеры были оценены, модель обновляется. Весь этот процесс похож на цикл и называется эпохой обучения.

Некоторые преимущества пакетного градиентного спуска заключаются в его вычислительной эффективности, стабильном градиенте ошибки и устойчивой сходимости. К недостаткам можно отнести то, что стабильный градиент ошибки иногда может привести к состоянию сходимости, которое не является лучшим, которого может достичь модель. Также требуется, чтобы весь набор обучающих данных находился в памяти и был доступен алгоритму.

2) Стохастический градиентный спуск (Stochastic Gradient Descent), который, в отличии от предыдущего типа, обновляет параметры модели для каждого обучающего примера по очереди. В зависимости от задачи, стохастический градиентный спуск может оказаться быстрее, чем пакетный градиентный спуск. Одно из преимуществ заключается в том, что частые обновления позволяют иметь довольно подробную динамику изменений.

Однако частые обновления требуют больших вычислительных затрат, чем пакетный метод градиентного спуска. Кроме того, частота таких обновлений может привести к зашумлению градиентов, что может вызвать скачки коэффициента ошибок вместо его медленного уменьшения.

3) Мини-пакетный градиентный спуск (Mini-Batch Gradient Descent)

представляет собой комбинацию концепций стохастического и пакетного градиентного спуска. Он просто разбивает набор обучающих данных на небольшие партии и выполняет обновление для каждой из них. Это создает баланс между устойчивостью стохастического градиентного спуска и эффективностью пакетного градиентного спуска.

Обычные размеры мини-партий варьируются от 50 до 256, но, как и в любой другой технике машинного обучения, здесь нет четкого правила, поскольку значение варьируется для разных задач.

Существует несколько других популярных алгоритмов. Один из них – Adam, который используется в данной работе, является алгоритмом адаптивной оптимизации скорости обучения, разработанным специально для обучения глубоких нейронных сетей. Kingma (2014) Adam можно рассматривать как комбинацию алгоритма RMSprop и стохастического градиентного спуска с моментом. Он использует квадрат градиентов для масштабирования скорости обучения, как RMSprop, и применяет преимущества момента, используя скользящее среднее градиента вместо самого градиента, как стохастический градиентный спуск с моментом.

Adam является адаптивным алгоритмом в связи с тем, что он использует оценки первого и второго моментов градиента для адаптации скорости обучения каждого веса нейронной сети. Первым и вторым моментом является ожидаемое значение переменной в первой и второй степени.

Adam имеет следующие преимущества:

- Простота реализации;
- Вычислительная эффективность;
- Небольшие требования к памяти;
- Инвариантен к диагональному изменению градиентов;
- Хорошо подходит для задач с большим количеством данных и/или параметров;
- Подходит для нестационарных задач;

- Подходит для задач с очень шумными и/или разреженными Градиентами;
- Гипер-параметры имеют интуитивно понятную интерпретацию и обычно требуют незначительной настройки.

Однако в некоторых областях задач, например классификации изображений, Adam не сходится к оптимальному решению из-за своей специфики реализации, и лучшие результаты по-прежнему достигаются при использовании стохастического градиентного спуска с моментом.

3.2.2. Инструментальные средства

При выполнении данной работы для проведения вычислений и разработки и обучении нейронной сети в качестве инструментального средства был выбран язык Python 3, открытые библиотеки Pandas и NumPy, а также фреймворк TensorFlow.

Python — это интерпретируемый, интерактивный, объектно-ориентированный и высокоуровневый язык программирования общего назначения. Он был создан Гвидо ван Россумом в 1985-1990 годах. Как и Perl, исходный код Python доступен под лицензией GNU General Public License (GPL).

Python 3 является третьей версией языка Python, которая появилась в 2008 году.

Важные характеристики Python:

- Поддерживает функциональные и структурированные методы программирования, а также ООП;
- Его можно использовать как язык сценариев или компилировать в байт-код для создания больших приложений;
- Предоставляет динамические типы данных очень высокого уровня и поддерживает динамическую проверку типов;
- Поддерживается автоматическая сборка неиспользуемой памяти;

- Легко интегрируется с C, C++, COM, ActiveX, CORBA и Java.

Эти характеристики делают Python наиболее подходящим для проектов в области машинного обучения и нейронных сетей. А развитие доступных библиотек для работы с данными и машинным обучением делает этот язык очень удобным для создания нейронных моделей.

NumPy — это фундаментальный пакет для научных вычислений на языке Python. Он предоставляет объекты многомерного массива, различные производные объекты (такие как массивы с масками и матрицы), а также набор процедур для быстрых операций над
массивами, включая математические, логические, манипуляции с формами, сортировку, выборку, ввод-вывод, дискретные преобразования Фурье, основы линейной алгебры, основные статистические операции, случайное моделирование и многое другое.

В основе пакета NumPy лежит объект ndarray (Рисунок 16). Он инкапсулирует n- мерные массивы однородных типов данных, причем многие операции выполняются в скомпилированном коде для повышения производительности.

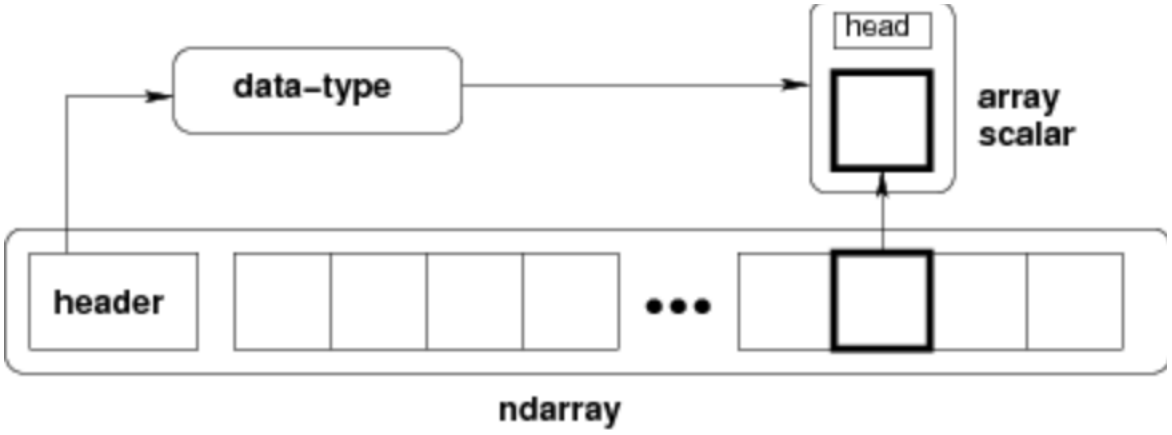


Рисунок 16 - Структура объекта ndarray

Существует несколько важных отличий между массивами NumPy и стандартными последовательностями Python:

- Массивы NumPy имеют фиксированный размер при создании, в отличие от списков Python (которые могут динамически увеличиваться). Изменение размера массива ndarray приведет к созданию нового массива и удалению исходного;
- Все элементы в массиве NumPy должны быть одного типа данных, и поэтому они будут иметь одинаковый размер в памяти. Исключение: можно иметь массивы объектов (Python, включая NumPy), что позволяет создавать массивы с элементами разного размера;
- Массивы NumPy облегчают сложные математические и другие виды операций над большим количеством данных. Как правило, такие операции выполняются более эффективно и с меньшим количеством кода, чем это возможно при использовании встроенных последовательностей Python.

Согласно оценкам, объекты массива NumPy в 50 раз быстрее традиционных списков Python. Это достигается за счет того, что объекты ndarray хранятся в одном постоянном месте в памяти, поэтому процессы могут обращаться к ним и манипулировать ими очень эффективно. Кроме этого, NumPy оптимизирован для работы с новейшими архитектурами процессоров.

Pandas — это пакет Python с открытым исходным кодом, который наиболее широко используется для решения задач науки анализа данных и машинного обучения. Он построен на основе Numpy, который обеспечивает поддержку многомерных массивов. Будучи одним из самых популярных пакетов для работы с данными, Pandas хорошо сочетается со многими другими модулями для работы с данными в экосистеме Python и обычно входит в любой дистрибутив Python.

Pandas хорошо подходит для работы с различными типами данных:

- Табличные данные с разнородными столбцами, как SQL или электронной таблице Excel;

- Упорядоченные и неупорядоченные (не обязательно с фиксированной частотой) данные временных рядов;
- Произвольные матричные данные (однородно типизированные или неоднородные) с метками строк и столбцов;
- Любые другие формы наборов наблюдательных/статистических данных.

Pandas упрощает выполнение многих трудоемких и повторяющихся задач, связанных с работой с данными, включая:

- Очистка данных;
- Заполнение данных;
- Нормализация данных;
- Слияния и объединения;
- Визуализация данных;
- Статистический анализ;
- Проверка данных;
- Загрузка и сохранение данных.

Две основные структуры данных (Рисунок 17) в Pandas, Series (одномерная) и DataFrame (двумерная), позволяют обрабатывать подавляющее большинство типичных случаев использования в финансах, статистике и других областях.

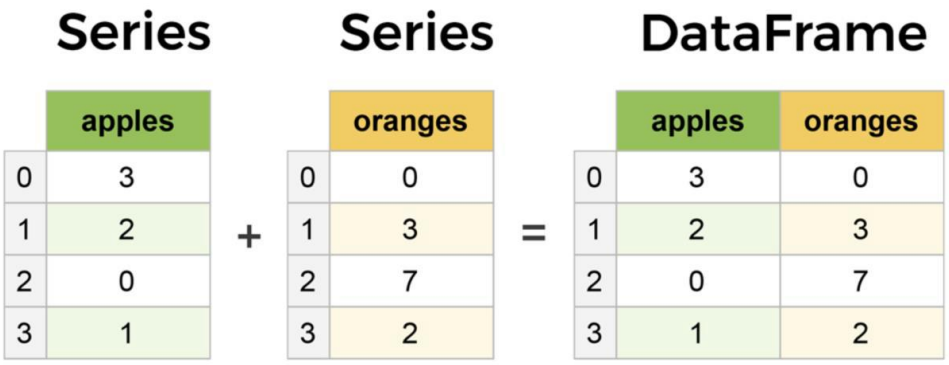


Рисунок 17 - Структура объектов Series и DataFrame

В процессе обучения моделей с нейронными сетями возникает необходимость в обработке большого количества данных. Сбор и подготовка датасета для обработки нейронной сети является важнейшей частью работы. Pandas и NumPy, благодаря своим преимуществам, позволяют эффективно провести работу с данными.

TensorFlow — это универсальная платформа с открытым исходным кодом для машинного обучения. Она имеет обширную, гибкую экосистему инструментов, библиотек и ресурсов сообщества, которая позволяет исследователям использовать передовые достижения в области машинного обучения, а разработчикам - легко создавать и внедрять приложения с его поддержкой.

TensorFlow был изначально разработан исследователями и инженерами, работающими в команде Google Brain в рамках организации Google Machine Intelligence Research для проведения исследований в области машинного обучения и глубоких нейронных сетей. Система является достаточно общей, чтобы быть применимой в широком спектре других областей.

TensorFlow предоставляет стабильные API на языках Python и C++, а также обратную совместимость API для других языков.

Приложения TensorFlow могут работать как на обычных процессорах CPU, так и на более производительных графических процессорах (GPU), а также на собственных тензорных процессорах (TPU) Google, которые представляют собой специализированные устройства, специально разработанные для ускорения работы TensorFlow.

Базовые концепты TensorFlow:

- Каждое вычисление в TensorFlow описывает направленный граф, состоящий из узлов и ребер, где узлы — это операции/функции, а ребра - входные и выходные переливы и эти функции:

- Входы/выходы в TensorFlow называются тензорами. Тензор — это не что иное, как многомерный массив, для которого тип базового элемента задается во время построения графа (Рисунок 18).

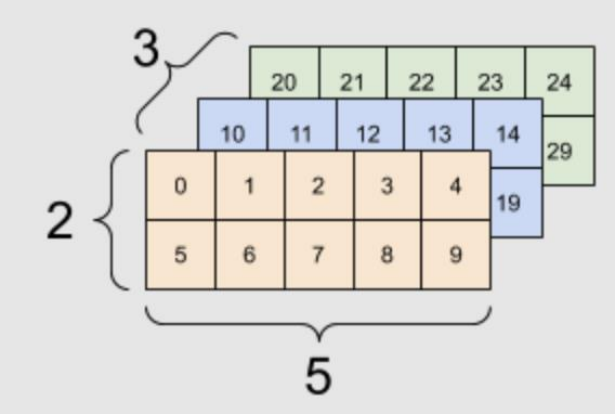


Рисунок 18 - Схема тензора.

- Клиентская программа, использующая TensorFlow, генерирует графы потоков данных с помощью одобряемых языков программирования (C или Python);
- Операция — это функция, которая имеет имя и представляет собой абстрактное вычисление. Операция может иметь атрибуты, которые должны быть сгенерированы и переданы во время построения графа;
- Одним из частых применений атрибутов является создание полиморфных операций;
- Ядро — это реализация операции на конкретном устройстве;
- Клиентская программа взаимодействует с притоком системы TensorFlow путем создания сессий. Интерфейс сессии имеет расширенные стили для генерации графа вычислений и метод усиления run(), который вычисляет выход каждого узла графа вычислений, предоставляя необходимые входные данные;
- В большинстве задач машинного обучения вычислительные графы применяются многократно, и большинство обычных тензоров не

выживают после не присоединённого выполнения, поэтому в TensorFlow есть переменные;

- Переменная — это особый вид операции, возвращающая обработчик к переменному тензору, который может сохраняться во время многократного выполнения графа;
- Обучаемые параметры, такие как веса, смещения, размещаются в тензорах, хранящихся в переменных;

Высокоуровневая архитектура TensorFlow может быть описана следующим образом (Рисунок 19):

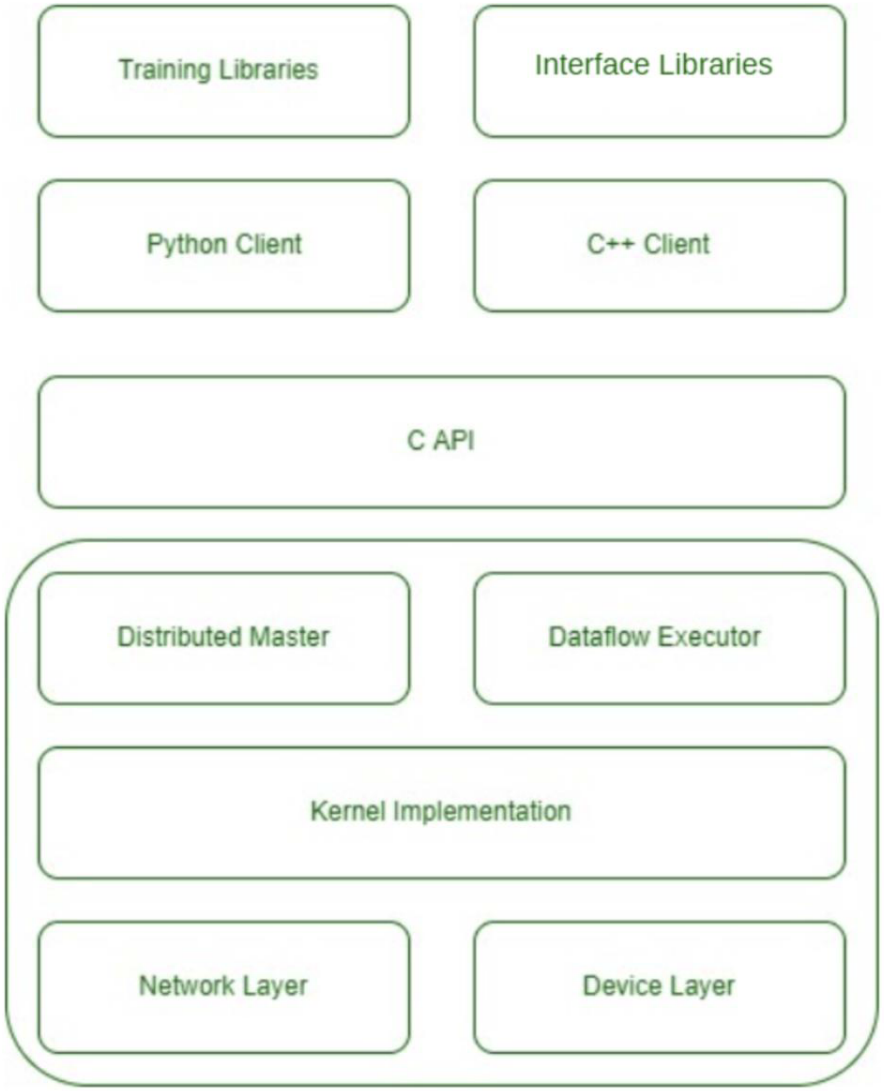


Рисунок 19 - Высокоуровневая схема архитектуры TensorFlow.

1) Первый уровень TensorFlow состоит из уровня устройства и сетевого уровня. Уровень устройств содержит реализацию для связи с различными устройствами, такими как GPU, CPU, TPU в операционной системе, на которой будет работать TensorFlow. В то время как сетевой уровень содержит реализацию для связи с различными машинами, использующими различные сетевые протоколы в распределяемой обучаемой среде;
2) Второй уровень TensorFlow содержит реализации ядра для приложений, в основном используемых в машинном обучении;
3) Третий уровень TensorFlow состоит из распределенного мастера и исполнителей потока данных. Распределенный мастер имеет возможность распределять рабочие нагрузки на различные устройства в системе. В то время как исполнитель потока данных оптимально выполняет граф потока данных;
4) Четвертый уровень предоставляет все функциональные возможности в виде API, который реализован на языке Си. Язык С выбран потому, что он быстрый, надежный и может работать на любой операционной системе;
5) Пятый слой обеспечивает поддержку клиентов на языках Python и C++;
6) И последний слой TensorFlow содержит библиотеки обучения и вывода, реализованные на языках python и C++.

Практическая часть

В этом разделе рассмотрена эмпирическая часть работы, которая включает в себя подготовку датасета, реализацию нейронной сети для вычисления цены опционов и определения коэффициента хеджирования. Реализация программы выполнена с использованием фреймворка TensorFlow и языка программирования Python 3.

1. Предобработка данных

Для проведения вычислительного эксперимента был составлен портфель ценных бумаг, состоящий из одного базового актива – акции АО «Астра».

Согласно теоретической модели Блэка-Шоулза, для определения премии колл опциона необходимо наличие нескольких параметров:

- Текущей цены на базовый актив;
- Цены исполнения;
- Времени погашения;
- Безрисковой процентной ставки;
- Оценки волатильности актива.

При подготовке датасета для обучения модели, данные параметры были определены образом, представленным ниже.

Цена исполнения была определена как средняя цена стоимости базового актива за прошедший год. Для этого были загружены данные закрывающих цен акций АО «Астра» и найдена медиана этого набора значений.

Текущей ценой базового актива было установлено значение произведения случайно величины в диапазоне от 0.8 до 1.2 и цены исполнения. Это позволило получить текущие цены базового актива в диапазоне от 80 до 120 процентов от цены экспирации, что позволяет имитировать данные, близкие к реальным данным рынка.

Для срока погашения опциона выбраны значения в диапазоне от 0.01 до 1, что соответствует времени исполнения от 1 дня до 1 года. Такие данные времени позволяют наиболее полно оценить предсказываемые значения нейронной сети для
премии опционов при наиболее коротком, среднем и длинном сроках истечения опционов.

В качестве процентной ставки была установлена текущая на 19 июня 2025 года процентная ставка Центрального Банка России в размере 20 процентов.

Для оценки волатильности базового актива был использован параметр исторической волатильности акции. Чтобы определить историческую волатильность, были использованы данные цены базового актива за прошедший год и рассчитано стандартное отклонение дневных изменений цен акций. Для расчета исторической волатильности была использована формула:

$$\sigma_{HV} = \sigma_{daily}\sqrt{252}$$

Где σ_{HV} – историческая волатильность за год;

σ_{daily} – дневная волатильность;

252 – число торговых дней в году.

Все полученные параметры были объединены и масштабированы в датасет, состоящий из двадцати тысяч наблюдений.

2. Разработка и обучение модели.

Для создания модели из фреймворка TensorFlow была импортирована библиотека Keras. Это API для создания моделей глубокого обучения, работающий поверх TensorFlow. В качестве основы для

нейронной сети была выбрана последовательная модель keras.Sequential. Такая модель подходит для стандартного стека слоев, где каждый слой имеет ровно один входной тензор и один выходной тензор.

В качестве входного слоя, получающего данные из датасета был выбран слой предварительной обработки, который нормализует получаемые признаки (tf.keras.layers.Normalization). Это позволяет передавать данные в модель без первоначальной нормализации и избегать связанных с этим ошибок.

Для нейронной сети установлены параметры, которые указаны в Таблице 1.

Таблица 1 - Параметры нейронной сети

Параметр	Значение
Количество скрытых слоев	4
Количество нейронов в одном слое	64
Функции активации скрытого слоя	f1 – Leaky ReLu f2 – ELU f3 – ReLu f4 – ELU

Выходной слой состоит из 1 нейрона без активации, который сообщает предсказанное значение – премию опциона.

Итоговая схема архитектуры модели представлена на рисунке 20:

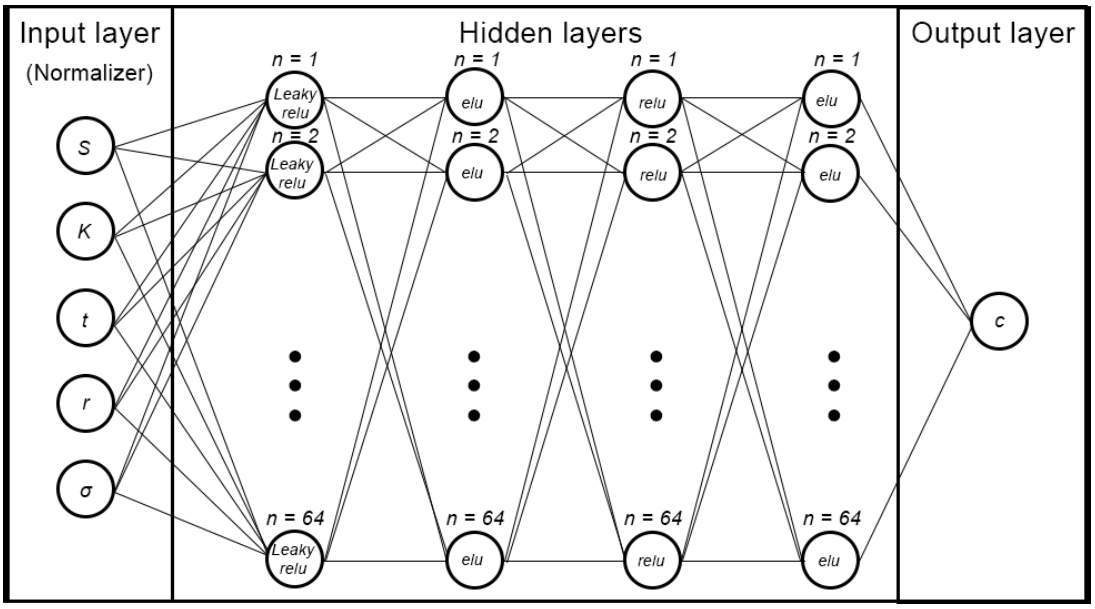


Рисунок 20 - Архитектура построенной модели

Обучение модели осуществляется с помощью метода обратного распространения ошибки с использованием градиентного спуска. Для обучения по полученному датасету определяется теоретическая цена опциона согласно формуле Блэка-Шоулза. Полученные данные разбиваются на тренировочную и тестовую выборку в соотношении 80 к 20. Данные нормализуются во входном слое.

В процессе обучения нейронной сети используется оптимизатор Adam. Параметры обучения нейронной сети представлены в Таблице 2.

Таблица 2 - Параметры обучения

Параметр	Значение
Скорость обучения	0.001
Оптимизатор	Adam
Коэффициент 1-го момента	0.9
Коэффициент 2-го момента	0.999
Константа точности	1×10^{-7}

В качестве целевой функции для оптимизатора используется MSE:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

где n — число примеров обучающей выборки;

y_i — целевое значение i-го примера;

$\hat{y}i$ — предсказанное моделью значение.

После обучения нейронная сеть тренируется на тестовой выборке данных из датасета.

На основе собранного датасета и теоретически полученной цены опциона определяется дельта коэффициент согласно формуле:

$$\sigma = N(d_1),$$

$$d_1 = \frac{\ln\frac{S}{X} + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}},$$

Где σ – оценка волатильности базового актива;

$N(d_1)$ - кумулятивная функция стандартного нормального распределения;

S - цена базового актива;

X - цена исполнения опциона;

r - безрисковая процентная ставка;

T - время до погашения.

Этот коэффициент определяется также как частная производная стоимости опциона по цене базового актива:

$$\Delta = \frac{\partial c}{\partial s},$$

Где c – стоимость опциона;

S – стоимость базового актива.

Расчет частных производных содержится во встроенном функционале TensorFlow. Для этого необходимо использовать класс GradientTape(). С его помощью, передав параметры датасета в качестве переменных TensorFlow в нейронную сеть, можно определить градиент предсказанных значений цен опционов по цене базового актива. Полученный градиент и есть частная производная, то есть искомый дельта коэффициент.

3. Результаты работы модели

Данный раздел описывает полученные результаты и визуализирует их.

Во время обучения модели отслеживалась ошибка функции потерь на каждой эпохе обучения. График представлен на рисунке 21.

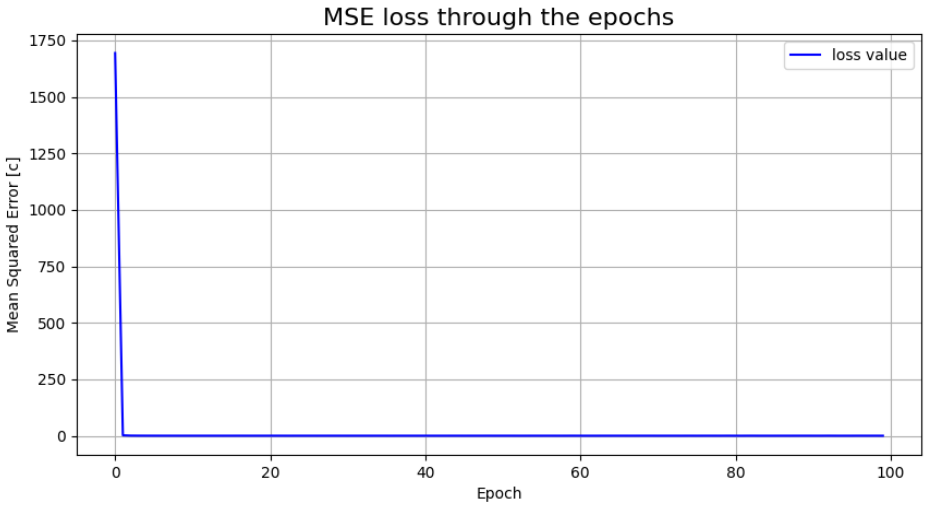


Рисунок 21 - График ошибки функции потерь

Чтобы оценить качество построенной модели, были проведены сравнения цен опционов, предсказанных нейронной сетью и аналитически с помощью формулы Блэка-Шоулза. Для оценки использовались три метрики – MSE, MAE, R2. Итоговые результаты показателей:

$MSE = 4{,}27 \times 10^{-3}$

$MAE = 6{,}54 \times 10^{-2}$

$R^2 = 9{,}99 \times 10^{-1}$

Гистограмма распределения ошибок изображена на рисунке 22.

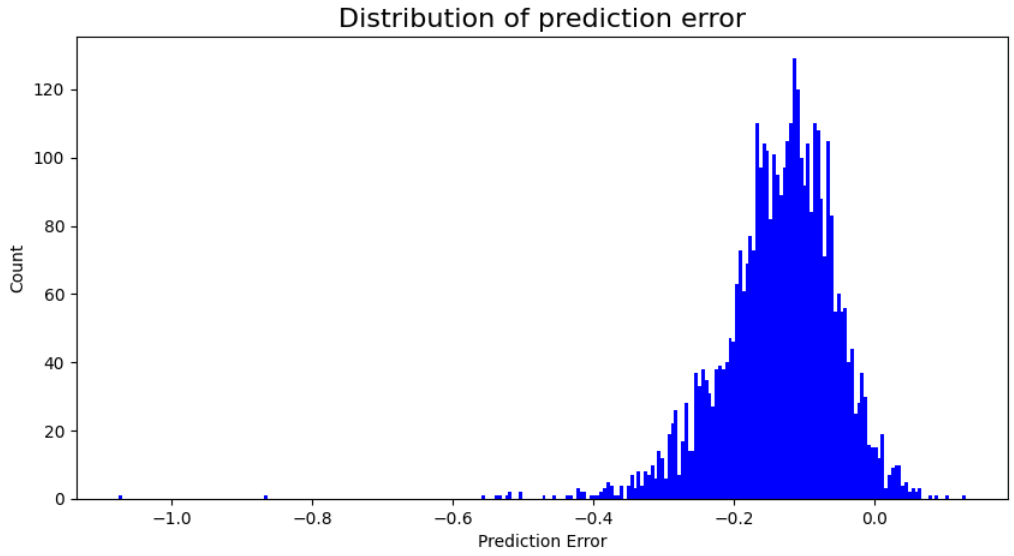


Рисунок 22 - Распределение ошибки предсказанных значений

График показывает нормальное распределение значений ошибок, что символизирует правильность полученных данных.

Графическое сравнение аналитического решения и решения, предсказанного нейронной сетью представлено на рисунке 23.

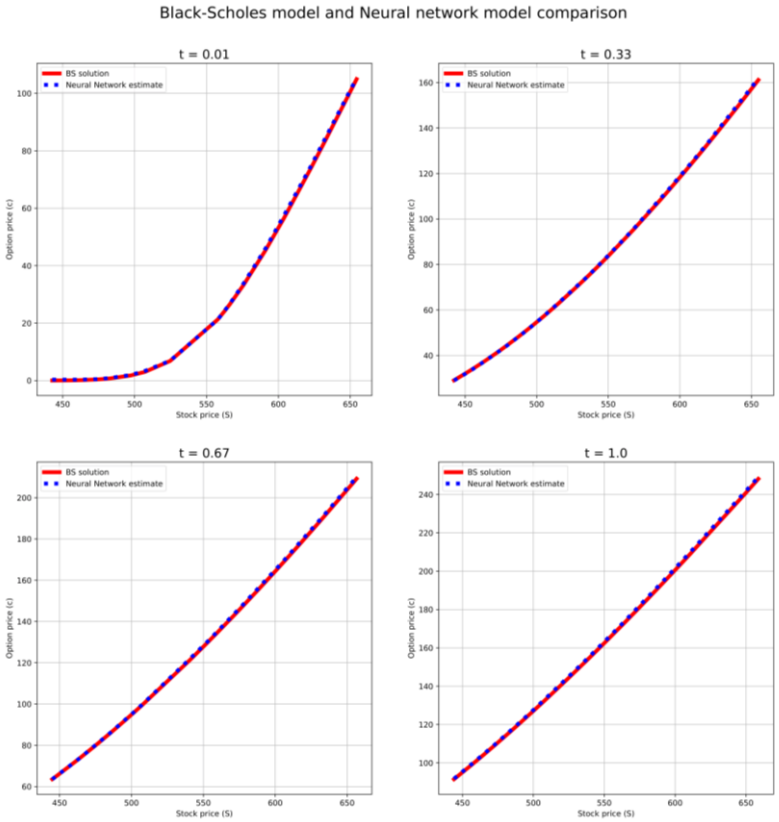


Рисунок 23 - Сравнение аналитического решения с решением модели

Трехмерные визуализации поверхности теоретической цены опционов и цены полученной с помощью нейронной сети показаны на рисунке 24.

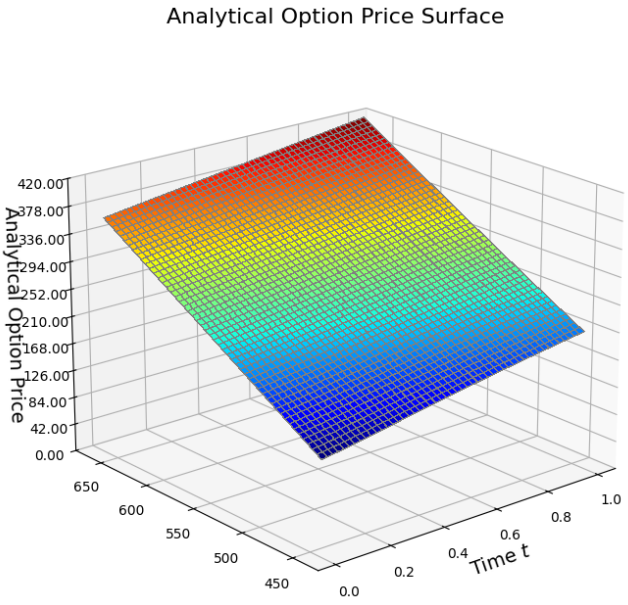


Рисунок 24 - Поверхность аналитической цены опционов

Как можно заметить, исходя из данных визуализаций, полученная модель с использованием нейронной сети демонстрирует достаточно хорошее качество полученных результатов, представленных на рисунке 25.

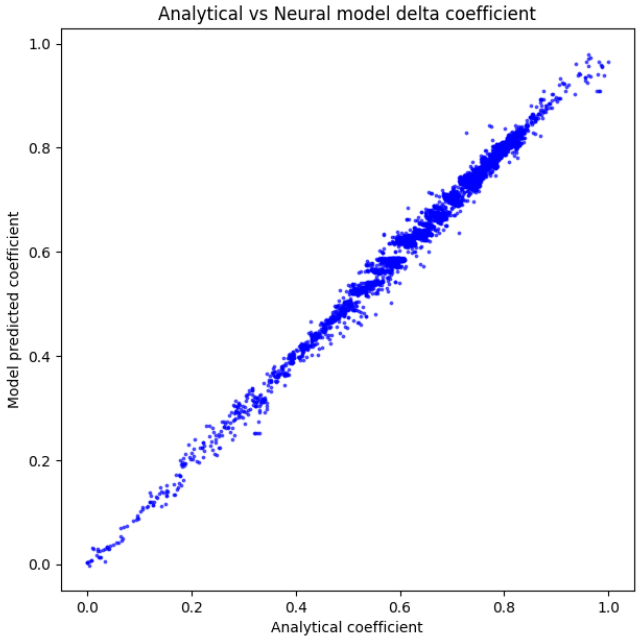


Рисунок 25 - График сравнения полученных дельта коэффициентов

На данном графике совершенно отчетливо прослеживается линейная регрессия. Это говорит о положительном результате применения модели с использованием машинного обучения для определения дельта коэффициента, что, в свою очередь, символизирует о возможности применения моделей машинного обучения для управления рыночным риском портфелей ценных бумаг.

4. Ссылка на репозиторий

Ссылка на репозиторий с исходным кодом, разработанного приложения:

https://github.com/yCTuHbl4/Data-Architect-Pro

Заключение

Во время проведения исследования была проделана большая работа по изучению и применению различных методов машинного обучения для решения задачи управления рисками портфеля ценных бумаг.

В заключении подведу итог выполненной работы:

1) В работе были изучены теоретические аспекты задач ценообразования опционов и определения дельта коэффициента, прямым образом связанного с риском актива;
2) В работе была представлена архитектура нейросетевой модели и ее параметры. Были описаны процессы обучения, тестирования и оценки нейронной сети;
3) Была разработана программа для оценки премии опционов и определения дельта коэффициента опционов на языке программирования Python с использованием открытого фреймворка машинного обучения TensorFlow и библиотеки Keras;
4) Были выполнены вычислительные эксперименты по определению коэффициента риска портфеля, состоящего из одного базового актива, и оценка качества проведенного эксперимента.

Методы, подходы и результаты, описанные в данной работе, могут служить основой и дорабатываться в дальнейших работах и исследованиях по ценообразованию опционов и определению дельта коэффициентов портфелей ценных бумаг.

Список источников

1) Hull John. Options futures and other derivatives. Pearson Education India, 2003;
2) Black, Fischer; Myron Scholes. The Pricing of Options and Corporate Liabilities / Journal of Political Economy: журнал. – 1973. – Вып. 81, № 3. – С. 637–654. – doi:10.1086/260062;
3) Haug Espen Gaarder. The complete guide to option pricing formulas. McGraw- Hill Companies – 2007;
4) С. Шорохов, Управление портфелями финансовых активов. – М.: РУДН, 2013;
5) McCulloch Warren, Pitts Walter. A logical calculus of the ideas immanent in nervous activity / The bulletin of mathematical biophysics: журнал. – 1973. Вып. 5, № 3. – С. 115–133;
6) Hebb Donald. The organization of behavior: a neuropsychological theory. J. Wiley; Chapman & Hall, 1949;
7) Rosenblatt Frank. The perceptron: a probabilistic model for information storage and organization in the brain / Psychological review: журнал. – 1958. – Вып. 65, № 6. – С. 386;
8) Ivakhnenko. Heuristic self-organization in problems of engineering cybernetics / Automatica: журнал. – 1970. – Вып. 6, №2. – С. 207–219;
9) Hahnloser Richard. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit / Nature: журнал. – 2000. – Вып. 405, №6789. – С. 947– 951;
10) Rumelhart David, Hinton Geoffrey, Williams Ronald. Learning representations by back-propagating errors / Nature: – журнал. – 1986. – Вып. 323, №6088. – С. 533–536;

11) “Python3 Documentation” [Электронный ресурс]: Режим доступа https://docs.python.org/3/ (Дата обращения 18.06.2025);
12) “NumPy Documentation” [Электронный ресурс]: Режим доступа https://numpy.org/doc/stable/ (Дата обращения 18.06.2025);
13) “Pandas documentation” [Электронный ресурс]: Режим доступа https://pandas.pydata.org/docs/ (Дата обращения 18.06.2025);
14) Cade Metz. “Google Just Open Sourced TensorFlow, Its Artificial Intelligence Engine” [Электронный ресурс]: Режим доступа https://www.wired.com/2015/11/google-open-sources-its-artificial-intelligence-engine/ (Дата обращения 18.06.2025)