СОДЕРЖАНИЕ

ВВЕДЕНИЕ………………………………………………………………….….... 3

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ…………………….....4

ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ.……………………………...………5

1.1. Основы рекуррентных нейронных сетей. .……………………………...….5

1.1.1. Архитектуры рекуррентных нейронных сетей. .……………………...….7

1.1.2. Обучение рекуррентных нейронных сетей. .……….……………...…….13

1.2. Алгоритмы градиентного спуска для настройки параметров. .……..…….15

1.2.1. Метод обратного распространения ошибки. .……..………………..……15

1.2.2. Метод градиентного спуска с импульсом. .……..………………..…...…17

1.2.3. Метод градиентного спуска с адаптивной нормой обучения. …..…...…19

1.3. Управляемые элементы и их роль в настройке параметров. …..…........…20

1.3.1. Понятие управляемых элементов...……………...………………..…...…20

1.3.2. Использование управляемых элементов для управления скоростью обучения………………………………………………………………………….22

1.3.3. Использование управляемых элементов для управления весами регуляризации. …………………….…………………………………………….23

ГЛАВА 2. ПРЕДЛАГАЕМЫЙ АЛГОРИТМ НАСТРОЙКИ ПАРАМЕТРОВ ДЛЯ АНАЛИЗА ФИНАНСОВОГО РЫНКА…………………………………..25

2.1. Обзор предлагаемого алгоритма. …………………………………………..25

2.1.1. Мотивация и цели алгоритма. ……………………………………..……..25

2.1.2. Основные принципы алгоритма. ………………………………….……..27

2.2. Детали реализации алгоритма………………………………………………28

2.2.1. Выбор управляемых элементов. …………………………………………28

2.2.2. Обновление управляемых элементов. …………………………………...30

2.2.3. Обновление весов сети. …………………………………………………...31

2.3. Гиперпараметры алгоритма и их влияние на производительность………33

2.3.1. Скорость обучения…………………………………………………….......33

2.3.2. Коэффициент импульса……………………………………………….......34

2.3.3. Коэффициент регуляризации…...…………………………………….......35

ГЛАВА 3. ЭКСПЕРИМЕНТАЛЬНАЯ ОЦЕНКА. …...…………………….......36

3.1. Описание наборов данных и экспериментальной установки…...………...36

3.1.1. Описание наборов данных. …...……………………………………..........36

3.1.2. Обработка датасета.…...……………………………………………..........37

3.2. Оценка производительности предлагаемого алгоритма .…...…...…..........38

3.2.1. Разработка алгоритма.…...……………………………………………......38

3.2.2. Оценка метрик модели. .…...……………………………………………...40

3.3. Анализ влияния управляемых элементов на обучение модели. .…...……48

3.3.1. Влияние управляемых элементов на скорость обучения. . .……....……48

3.3.2. Влияние управляемых элементов на точность.……....……..…………...50

ГЛАВА 4. ПРИЛОЖЕНИЯ И РАСШИРЕНИЯ. .……....……..……...………...56

4.1. Использование других управляемых элементов. .……....……..….............56

4.2. Применение алгоритма к другим типам нейронных сетей. .……....……...59

4.3. Теоретический анализ алгоритма.……....…….............................................60

ГЛАВА 5. ВЫВОДЫ.……....……........................................................................63

5.1. Краткое изложение основных результатов исследования………………..63

5.2. Обсуждение ограничений и перспектив предлагаемого алгоритма……..66

ЗАКЛЮЧЕНИЕ. ....................................................................................................68

СПИСОК ЛИТЕРАТУРЫ.....................................................................................69

ВВЕДЕНИЕ

Современные финансовые рынки характеризуются высокой степенью динамики и сложности. Прогнозирование цен акций остается одним из наиболее актуальных исследовательских направлений в области финансов. С развитием технологий машинного обучения и нейронных сетей появилась возможность эффективно анализировать и прогнозировать временные ряды, что делает их применение в этой области особенно перспективным.

Целью данного исследования является разработка алгоритма настройки параметров рекуррентных нейронных сетей с управляемыми элементами для прогнозирования цен акций на финансовом рынке.

Задачи работы

1. Провести обзор существующих методов анализа и прогнозирования финансовых временных рядов.
2. Разработать алгоритм настройки параметров рекуррентных нейронных сетей с управляемыми элементами.
3. Провести экспериментальное исследование эффективности разработанного алгоритма на исторических данных цен акций.
4. Определить влияние управляемых элементов рекуррентных нейронных сетей на точность и скорость работы модели.

Объектом исследования являются цены акций компаний Google, взятые с исторических данных, за последние 10 лет.

Предметом исследования является разработка алгоритма рекуррентной нейронной сети с управляемыми элементами и настройка параметров для прогнозирования цен акций.

Результаты данного исследования могут быть полезны для финансовых аналитиков, трейдеров и инвесторов, позволяя им принимать более обоснованные решения на основе прогнозов цен акций. Кроме того, разработанный алгоритм может быть использован в различных финансовых приложениях для автоматизации процессов анализа и прогнозирования.

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

RNN - Recurrent Neural Network

BAM- Bidirectional Associative Memory

ESN - Echo State Network

LSTM - Long short-term memory

GRU - Gated Recurrent Units

biRNN - Bidirectional Recurrent Neural Network

SGD - Stochastic Gradient Descent

RMSprop - Root Mean Square Propagation

**MSE -** Mean Squared Error

MAE - Mean Absolute Error

MAPE - Mean Absolute Percentage Error

CNN - Convolutional Neural Network

ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ.

1.1. Основы рекуррентных нейронных сетей.

Рекуррентные нейронные сети (RNN) - это класс нейронных сетей, которые обрабатывают последовательности данных. Они отличаются от обычных нейронных сетей тем, что имеют внутреннее состояние или память, которая позволяет им запоминать предыдущие входные данные и использовать эту информацию для обработки последующих входов. Это позволяет им моделировать зависимости во времени или последовательности, что делает их полезными для задач, связанных с текстом, временными рядами, речью и другими формами последовательных данных.

Виды RNN

1). Один к одному – архитектура по сути является обычной нейронной сетью, используется для задач, где один вход соответствует одному выходу. Это стандартный тип RNN, который можно найти во многих приложениях машинного обучения, таких как:

* Классификация временных рядов: где каждому входному временному шагу соответствует метка класса.
* Обработка естественного языка: например, в задачах по определению тональности текста, где каждому предложению соответствует метка положительного или отрицательного отношения.

|  |  |
| --- | --- |
| [RNN OTO.jpg](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:RNN_OTO.jpg) |  |
| Рис.1. вид RNN один к одному |  |

2). Один ко многим -  относится к архитектуре, где один вход соответствует множеству выходов. Это часто используется в задачах, где один элемент данных генерирует последовательность данных на выходе.

Этот тип RNN полезен, когда необходимо создать длинные и связанные последовательности данных, которые сохраняют контекст из начального входа. Однако, такие модели могут быть сложными в обучении и требуют большого количества данных для эффективного обучения. Кроме того, они могут страдать от проблемы затухания или взрыва градиентов, что делает обучение более глубоких RNN сложным.

|  |  |
| --- | --- |
| [RNN OTM.jpg](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:RNN_OTM.jpg) |  |
| Рис.2. вид RNN один ко многим | | |  |

3). Многие к одному - много входов и один выход может применяться, если мы хотим оценить тональность рецензии. На вход подаем слова рецензии, на выходе получаем оценку ее тональности: позитивная рецензия или негативная.

|  |  |
| --- | --- |
| [RNN MTO.jpg](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:RNN_MTO.jpg) |  |
| Рис.3. вид RNN многие к одному | | |  |

4). Многие ко многим - используется для задач, где последовательность входных данных соответствует последовательности выходных данных. Это подходит для таких задач, как:

* Машинный перевод: где последовательность слов на одном языке переводится в последовательность слов на другом языке.
* Распознавание речи: где аудио-сигнал преобразуется в текстовую транскрипцию.
* Предсказание временных рядов: где на основе предыдущих данных предсказываются будущие точки данных.

|  |  |
| --- | --- |
| [RNN MTM1.jpg](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:RNN_MTM1.jpg) |  |
| [RNN MTM2.jpg](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:RNN_MTM2.jpg) |  |

Рис.4. вид RNN многие ко многим

1.1.1. Архитектуры рекуррентных нейронных сетей.

Полностью рекуррентная сеть

Это базовая архитектура, разработанная в 1980-х, в которой связь между различными узлами образует ориентированный граф, обеспечивающий временное динамическое поведение. Это помогает моделировать последовательные данные, полученные из сетей прямой связи. Он работает аналогично человеческому мозгу, обеспечивая прогнозируемые результаты. Рекуррентная нейронная сеть очень похожа на традиционную нейронную сеть, за исключением того, что к нейронам добавляется состояние памяти.

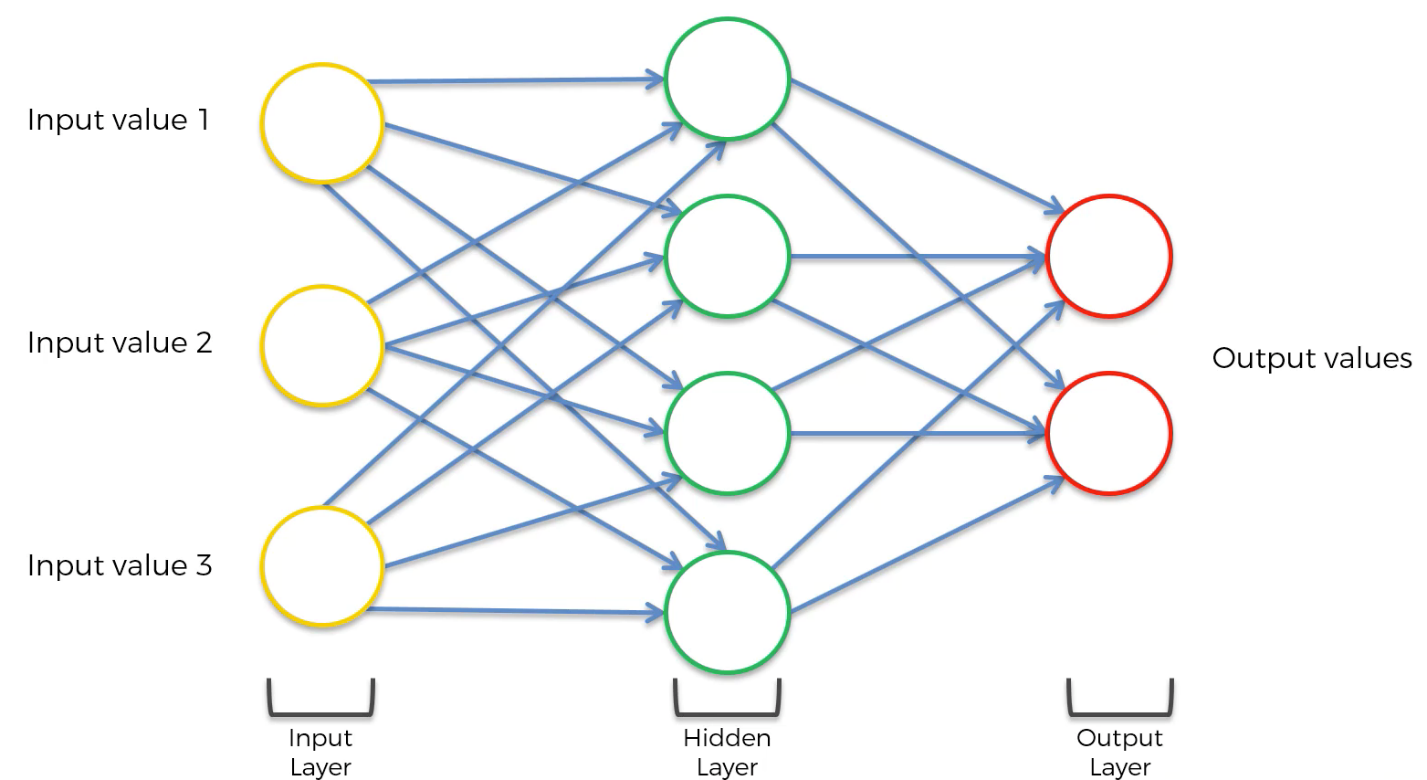


Рис.5. Полностью рекуррентная сеть

Рекурсивная сеть.

Рекурсивные нейронные сети применяют один и тот же набор весов к структурируемым данным, которые могут быть как фиксированными, так и переменными размерами. Основное отличие рекурсивных сетей заключается в том, что они могут обрабатывать иерархические структуры, что делает их довольно полезными для таких задач, как обработка естественного языка и задач, связанных с компьютерным зрением.

Нейронная сеть Хопфилда

Нейронная сеть Хопфилда – вид рекуррентной нейронной сети, которая представляет собой систему связанных нейронов, где каждый нейрон связан со всеми остальными, но который не воздействует сам на себя.

Сеть Хопфилда организует реализацию важного свойства ассоциативной памяти, а именно, восстановление искаженного (зашумленного) сигнала по ближайшему к нему эталонному. Входной вектор в процессе обучения сети применяется как исходное состояние сети, а далее сеть работает согласно выбранной динамике. При работе сети каждый обучающий пример, расположенный в области эталона образа,

Двунаправленная ассоциативная память (BAM)

Тип нейронной сети, который использует двунаправленные связи между двумя слоями нейронов, что дает ей возможность запоминания в двух направлениях, то есть она имеет возможность восстанавливать паттерны как в прямом, так и в обратном направлении. Может использоваться для таких задач, как распознавание образов, коррекция ошибок и оптимизация. Она особенно полезна в приложениях, где необходимо одновременно учитывать два различных набора данных или паттерна и находить их ассоциации.

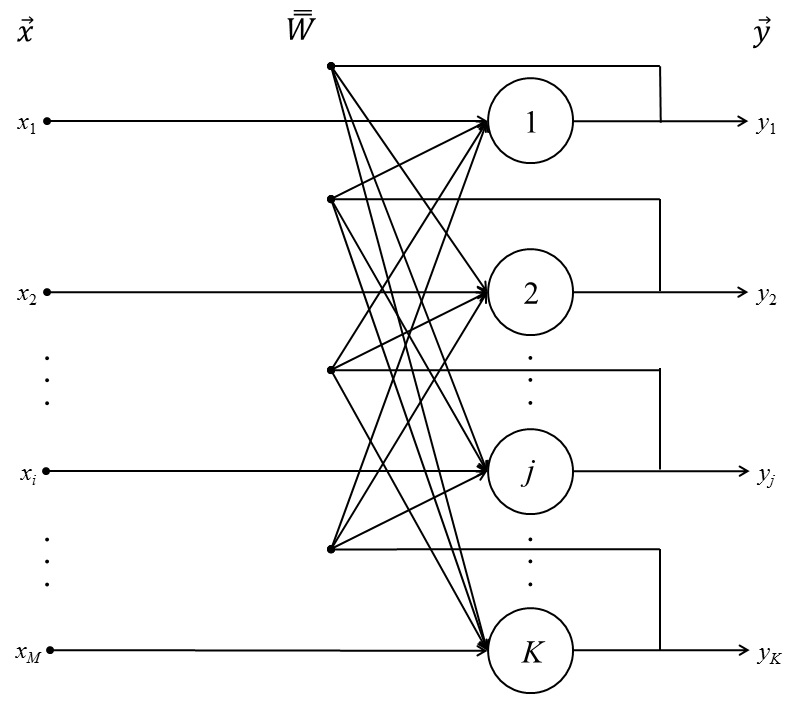


Рис.6. Нейронная сеть Хопфилда

Сеть Элмана.

Представляет собой простую рекуррентную сеть, которая использует контекстные узлы для сохранения информации о предыдущих состояниях сети, эти контекстные узлы связаны с скрытым слоем и обновляются на каждом временном шаге. В отличие от обычных рекуррентных нейронных сетей, где связи идут от выходов к входам, в сети Элмана связи идут от скрытого слоя к контекстным узлам, которые затем подаются обратно в скрытый слой на следующем временном шаге. Это позволяет сети запоминать информацию на короткие промежутки времени.

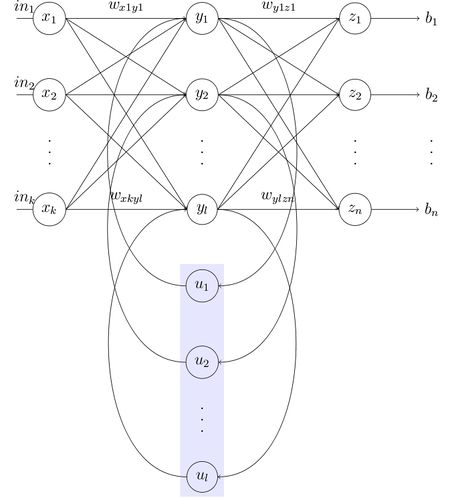
[](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:Elman_RNN.jpg)

Рис. 7. Сеть Элмана

Сеть Джордана

Сеть Джордана - вид [нейронных сетей](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C), который получается из многослойного [перцептрона](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%86%D0%B5%D0%BF%D1%82%D1%80%D0%BE%D0%BD" \o "Перцептрон), если на его вход подать, помимо входного вектора, выходной с задержкой на один или несколько тактов.

В данной сети выходной слой подключается к контекстным узлам, которые затем подаются обратно в скрытый слой. Контекстные узлы служат как память, позволяя сети учитывать предыдущие выходные данные при принятии текущих решений. Эта особенность делает сеть Джордана полезной для задач, где предыдущие выходные данные важны для последующих вычислений, например, при генерации последовательностей или временных рядов.

Структура сети Джордана включает в себя:

* Входной слой, который принимает внешние данные.
* Скрытый слой, который обрабатывает входные данные и передает их на выходной слой.
* Контекстный слой, который сохраняет выходные данные с предыдущего временного шага.
* Выходной слой, который генерирует результат обработки данных.

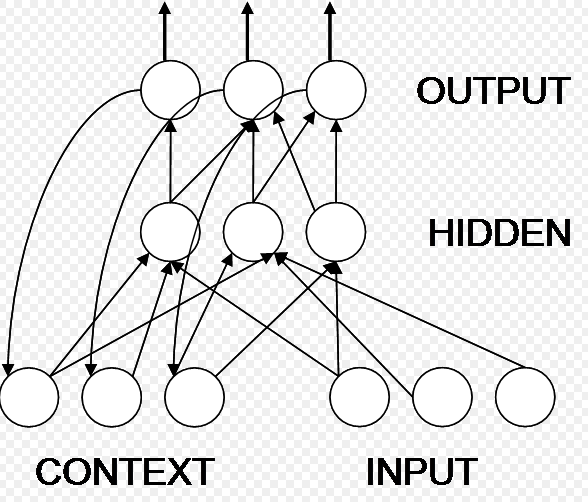


Рис.8. Сеть Джордана.

Эхо-сети.

Вид рекуррентных нейросетей, которые выделяются тем, что связи между нейронами в них случайны, не организованы в аккуратные слои, и обучаются они по-другому. Вместо подачи на вход данных и обратного распространения ошибки, мы передаем данные, обновляем состояния нейронов и в течение некоторого времени следим за выходными данными. Входной и выходной слои играют нестандартную роль, так как входной слой служит для инициализации системы, а выходной слой — в качестве наблюдателя за порядком активации нейронов, который проявляется со временем. Во время обучения изменяются связи только между наблюдателем и скрытыми слоями.

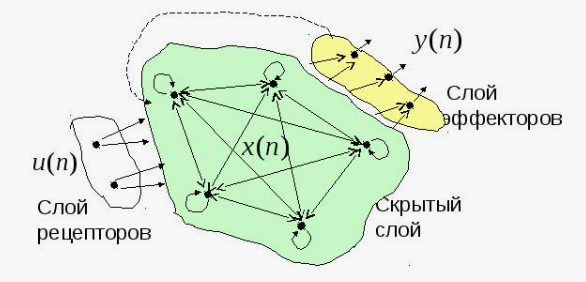


Рис.8. Эхо-сети

Сети долго-краткосрочной памяти

[Сеть долго-краткосрочной памяти](https://neerc.ifmo.ru/wiki/index.php?title=%D0%94%D0%BE%D0%BB%D0%B3%D0%B0%D1%8F_%D0%BA%D1%80%D0%B0%D1%82%D0%BA%D0%BE%D1%81%D1%80%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C) (LSTM) - особая разновидность архитектуры рекуррентных нейронных сетей, способная к обучению долговременным зависимостям.  Они прекрасно решают целый ряд разнообразных задач и в настоящее время широко используются. LSTM разработаны специально, чтобы избежать проблемы долговременной зависимости. Запоминание информации на долгие периоды времени – это их обычное поведение, а не что-то, чему они с пытаются обучиться.

Любая рекуррентная нейронная сеть имеет форму цепочки повторяющихся модулей нейронной сети. В обычной RNN структура одного такого модуля очень проста, например, он может представлять собой один слой с функцией активации.  
Структура LSTM также напоминает цепочку, но модули выглядят иначе. Вместо одного слоя нейронной сети они содержат целых четыре, и эти слои взаимодействуют особенным образом.

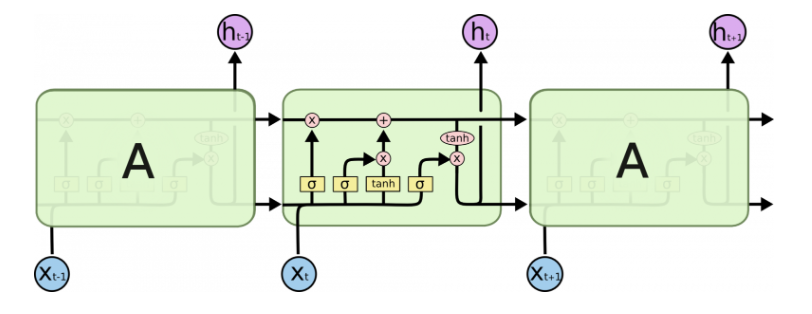


Рис.9. Модуль LSTM

Управляемые рекуррентные блоки

Управляемые рекуррентные блоки (GRU) — обладает меньшим количеством параметров, чем у LSTM, и в ней отсутствует выходное управление.

Структура GRU включает в себя два основных компонента:

1. Обновляющий вентиль: определяет, какая информация из прошлого должна быть сохранена для будущего.
2. Сбрасывающий вентиль: решает, какая информация из прошлого состояния должна быть забыта.

Эти вентили помогают GRU решать, когда обновлять и когда сбрасывать информацию, что позволяет сети сохранять долгосрочные зависимости и избегать проблемы исчезающего градиента. GRU особенно полезны в задачах обработки естественного языка и временных рядов, где необходимо учитывать информацию из длинных последовательностей данных.

Двунаправленные рекуррентные сети (biRNN)

Двунаправленная рекуррентная сеть представляет собой две однонаправленные рекуррентные сети, что позволяет информации передаваться не только вперед по времени, но и назад. Это означает, что в таких сетях есть два отдельных слоя, один для обработки данных в прямом временном направлении, а другой — в обратном. Данная структура позволяет сети улавливать контекст из обоих направлений, что делает biRNN особенно полезными для задач, где контекст важен для понимания каждого элемента последовательности.

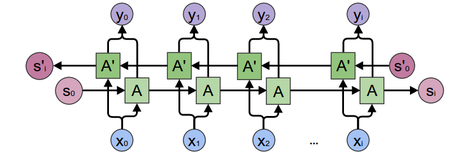
[](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:BiRNN.png)

Рис.10. Двунаправленная рекуррентная сеть

1.1.2. Обучение рекуррентных нейронных сетей.

Обучение рекуррентных нейронных сетей (RNN) включает в себя несколько этапов, включая подготовку данных, выбор архитектуры сети, определение функции потерь и оптимизатора, обучение и оценку модели. Опишем все этапы обучения:

1. Подготовка данных:

Этот этап включает загрузку и предобработку данных для обучения, валидации и тестирования. Предобработка может включать в себя такие операции, как токенизация текста, нормализация данных или масштабирование числовых значений.

Данные обычно разбиваются на две группы: обучающий набор, используемый для обучения модели и тестовый набор, используемый для оценки модели.

1. Выбор архитектуры сети:

Рекуррентные нейронные сети могут иметь различные архитектуры, такие как простые RNN, LSTM и GRU. Выбор архитектуры зависит от конкретной задачи и характеристик данных.

Для задач, требующих запоминания долгосрочных зависимостей, LSTM и GRU часто предпочтительнее простых RNN из-за их способности сохранять информацию в течение длительного времени.

1. Определение функции потерь и оптимизатора:

Функция потерь измеряет “стоимость” ошибок, совершаемых моделью в процессе обучения, вычисляя разницу между фактическими данными и результатом предсказания. Целью функции потерь является минимизировать данную разницу для более точных предсказаний. Для анализа временных рядов обычно вычисляют среднеквадратическую ошибку (MSE).

Оптимизатор отвечает за обновление весов модели с целью минимизации функции потерь. Популярные оптимизаторы включают в себя стохастический градиентный спуск (SGD), адаптивный градиентный спуск (Adam) и RMSprop.

1. Обучение модели:

На этом этапе модель обучается на обучающем наборе данных. Для каждой эпохи обучения модель передает последовательные входные данные через слои сети, вычисляет потери и обновляет веса с помощью выбранного оптимизатора.

В процессе обучения могут использоваться методы регуляризации, такие как dropout или L2 регуляризация, для предотвращения переобучения.

1. Оценка модели:

После завершения обучения модель оценивается на наборе для валидации и/или тестовом наборе данных. Это позволяет оценить обобщающую способность модели на новых данных.

Оценка модели может включать в себя вычисление метрик производительности, таких как точность, полнота, F1-мера для задач классификации, или среднеквадратическая ошибка для задач регрессии.

1. Настройка гиперпараметров:

При необходимости могут проводиться дополнительные итерации обучения с различными значениями гиперпараметров, такими как скорость обучения, количество скрытых единиц, глубина сети и другие, чтобы оптимизировать производительность модели.

Эти шаги образуют общий процесс обучения рекуррентных нейронных сетей. Помните, что успешное обучение модели часто требует экспериментации с различными аспектами архитектуры и гиперпараметров.

1.2. Алгоритмы градиентного спуска для настройки параметров.

1.2.1. Метод обратного распространения ошибки.

Метод обратного распространения ошибки (Backpropagation) используется для более эффективного обучения модели, которое происходит с помощью правила дифференцирования сложной функции, т.е. после каждого прохода по сети метод выполняет проход в обратную сторону и регулирует параметры модели.

В контексте RNN метод обратного распространения ошибки применяется к каждому временному шагу в последовательности данных. Он распространяет градиенты от последнего временного шага к первому, что позволяет модели учитывать зависимости во времени и обучаться на последовательных данных.

Этот процесс состоит из трех основных этапов:

1. Прямое распространение : данные передаются через сеть от входного слоя к выходному. Каждый нейрон выполняет операцию вычисления выходного значения на основе своих входных данных и текущих весов.

2. Вычисление функции потерь: происходит оценка того, насколько сеть ошибается в своих прогнозах по сравнению с фактическими значениями. Это делается путем сравнения выходных данных сети с желаемыми выходными данными и вычисления функции потерь или ошибки.

3. Обратное распространение: После вычисления выхода сети происходит вычисление градиентов функции потерь по отношению к весам сети с конца к началу. Затем градиенты используются для корректировки весов сети с помощью выбранного алгоритма оптимизации, такого как Adam или SGD.

Пример простого кода на Python, демонстрирующий обратное распространение ошибки для обновления параметров нейронной сети:

import numpy as np

# Функция активации (например, сигмоида)

def sigmoid(x):

return 1 / (1 + np.exp(-x))

# Производная функции активации

def sigmoid\_derivative(x):

return x \* (1 - x)

# Входные данные

X = np.array([[0, 0],

[0, 1],

[1, 0],

[1, 1]])

# Выходные данные

y = np.array([[0],

[1],

[1],

[0]])

# Инициализация весов сети

input\_neurons = 2

hidden\_neurons = 3

output\_neurons = 1

# Случайная инициализация весов

hidden\_weights = np.random.uniform(size=(input\_neurons, hidden\_neurons))

output\_weights = np.random.uniform(size=(hidden\_neurons, output\_neurons))

# Обучение сети

epochs = 10000

learning\_rate = 0.1

for epoch in range(epochs):

# Прямое распространение

hidden\_layer\_input = np.dot(X, hidden\_weights)

hidden\_layer\_output = sigmoid(hidden\_layer\_input)

output\_layer\_input = np.dot(hidden\_layer\_output, output\_weights)

output\_layer\_output = sigmoid(output\_layer\_input)

# Ошибка

error = y - output\_layer\_output

# Обратное распространение ошибки

d\_output = error \* sigmoid\_derivative(output\_layer\_output)

error\_hidden\_layer = d\_output.dot(output\_weights.T)

d\_hidden\_layer = error\_hidden\_layer \* sigmoid\_derivative(hidden\_layer\_output)

# Обновление весов

output\_weights += hidden\_layer\_output.T.dot(d\_output) \* learning\_rate

hidden\_weights += X.T.dot(d\_hidden\_layer) \* learning\_rate

print("Выходные данные после обучения:")

print(output\_layer\_output)

1.2.2. Метод градиентного спуска с импульсом.

Метод градиентного спуска с импульсом является вариацией стандартного метода градиентного спуска, который позволяет ускорить сходимость и избежать некоторых проблем, связанных с классическим градиентным спуском, такими как "застревание в локальных минимумах или медленная сходимость.

В этом методе градиент вычисляется как обычно, но вместо того, чтобы обновлять веса напрямую на основе текущего градиента, используется накопленный градиент из предыдущих шагов. Это позволяет учитывать предыдущие изменения весов и "импульс" в направлении обновления.

Преимущества метода градиентного спуска с импульсом включают ускорение сходимости и улучшение способности преодолевать локальные минимумы. Это достигается за счет того, что импульс позволяет алгоритму сохранять некоторую инерцию и продолжать движение вдоль оси градиента, даже если локальный градиент в данный момент времени указывает в другом направлении.

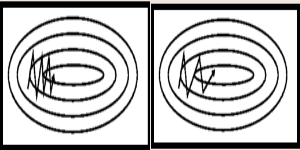


Рис.12.Разница скорости сходимости градиентного спуска с импульсом (справа) и без него(слева)

Пример кода Python, демонстрирующий использование метода градиентного спуска с импульсом:

import numpy as np

class MomentumGradientDescent:

def \_\_init\_\_(self, learning\_rate=0.01, momentum=0.9):

self.learning\_rate = learning\_rate

self.momentum = momentum

self.velocity = None

def update\_weights(self, weights, gradients):

if self.velocity is None:

self.velocity = np.zeros\_like(weights)

self.velocity = self.momentum \* self.velocity - self.learning\_rate \* gradients

weights += self.velocity

# Создание экземпляра класса с параметрами скорости обучения и импульса

optimizer = MomentumGradientDescent(learning\_rate=0.01, momentum=0.9)

# Обновление весов с использованием градиентов

optimizer.update\_weights(weights, gradients)

1.2.3. Метод градиентного спуска с адаптивной нормой обучения.

Метод градиентного спуска с адаптивной нормой обучения — это метод оптимизации, который адаптирует скорость обучения для каждого параметра в процессе обучения. Он помогает ускорить сходимость и обойти некоторые проблемы, связанные с фиксированной скоростью обучения, такие как медленная сходимость или расхождение.

В этом методе скорость обучения вычисляется индивидуально для каждого параметра или группы параметров на основе истории их обновлений. Например, для каждого параметра можно хранить экспоненциально сглаженный квадрат градиента. Это позволяет учитывать, насколько быстро или медленно каждый параметр обновлялся на предыдущих итерациях, и соответственно регулировать скорость обучения.

Пример кода на Python, демонстрирующий использование метода градиентного спуска с адаптивной нормой обучения с использованием алгоритма Adam:

import numpy as np

class AdamOptimizer:

def \_\_init\_\_(self, learning\_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8):

self.learning\_rate = learning\_rate

self.beta1 = beta1

self.beta2 = beta2

self.epsilon = epsilon

self.m = None

self.v = None

self.t = 0

def update\_weights(self, weights, gradients):

if self.m is None:

self.m = np.zeros\_like(weights)

self.v = np.zeros\_like(weights)

self.t += 1

self.m = self.beta1 \* self.m + (1 - self.beta1) \* gradients

self.v = self.beta2 \* self.v + (1 - self.beta2) \* (gradients \*\* 2)

m\_hat = self.m / (1 - self.beta1 \*\* self.t)

v\_hat = self.v / (1 - self.beta2 \*\* self.t)

weights -= self.learning\_rate \* m\_hat / (np.sqrt(v\_hat) + self.epsilon)

# Пример использования

# Создание экземпляра класса с параметрами скорости обучения и параметрами beta

optimizer = AdamOptimizer(learning\_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8)

# Обновление весов с использованием градиентов

optimizer.update\_weights(weights, gradients)

В этом примере learning\_rate представляет скорость обучения, beta1 и beta2 представляют коэффициенты, которые контролируют первый и второй моменты градиента соответственно, а epsilon представляет малое число, добавленное для численной стабильности. Метод update\_weights принимает текущие веса и соответствующие градиенты и обновляет веса с учетом адаптивной скорости обучения.

1.3. Управляемые элементы и их роль в настройке параметров.

1.3.1. Понятие управляемых элементов.

Управляемые элементы в нейронных сетях представляют собой веса и смещения, которые оптимизируются в процессе обучения и стремятся минимизировать ошибку между предсказанными и реальными данными.

Ниже приведены некоторые из наиболее распространенных управляемых элементов в RNN:

1. «Attention» (внимания): позволяет модели сосредоточиться на различных частях входной последовательности при выполнении задач, что помогает сети динамически вычислять важность каждого элемента входных данных в зависимости от текущего контекста.

2. Управляемые рекуррентные модели (Gated Recurrent Models): К ним относятся механизмы, такие как LSTM (Long Short-Term Memory) и GRU (Gated Recurrent Unit), которые вводят специальные вентили или гейты для контроля потока информации через RNN. Эти гейты помогают сети решать проблему затухающего или взрывного градиента и запоминать долгосрочные зависимости в данных.

3. Управляемые рекуррентные внимательные модели (Gated Recurrent Attention Models): Это комбинация механизмов внимания и управляемых рекуррентных моделей, которая позволяет сети сосредоточиться на важных частях входных данных и эффективно управлять потоком информации внутри сети.

4. Управляемые рекуррентные блоки (Gated Recurrent Blocks): Это модули, представляющие собой комбинацию различных типов управляемых рекуррентных моделей и слоев, которые могут быть вставлены внутрь архитектуры RNN для улучшения ее способности моделировать сложные зависимости в данных.

Управляемые элементы играют ключевую роль в эффективном применении RNN, поскольку они помогают модели лучше понимать и адаптироваться к структуре и содержанию входных данных, что приводит к более высокой производительности и качеству результатов.

1.3.2. Использование управляемых элементов для управления скоростью обучения.

Использование управляемых элементов для управления скоростью обучения является важным аспектом в машинном обучении, особенно при работе с рекуррентными нейронными сетями (RNN). Это позволяет модели адаптироваться к различным аспектам данных и динамически регулировать процесс обучения.

Один из подходов, который можно использовать для этой цели, - это комбинация управляемых элементов с алгоритмами оптимизации, такими как метод градиентного спуска с адаптивной нормой обучения (например, Adam), который мы обсуждали ранее.

Например, можно модифицировать процесс обновления весов в RNN, чтобы он учитывал важность каждого параметра или группы параметров в зависимости от контекста или задачи. Это может быть особенно полезно при работе с данными переменной длины или когда некоторые аспекты данных более важны для конкретной задачи.

Для этого можно адаптировать методы оптимизации, такие как Adam, чтобы они принимали во Attention веса или градиенты, сгенерированные управляемыми элементами. Например, можно использовать веса, вычисленные с помощью механизма внимания, для динамической настройки скорости обучения для каждого параметра.

Другой подход - это использование управляемых элементов для настройки гиперпараметров алгоритмов оптимизации. Например, можно использовать информацию о важности различных аспектов данных, полученную с помощью механизма внимания, для автоматической настройки параметров, таких как скорость обучения или коэффициенты сглаживания в алгоритмах оптимизации.

В целом, использование управляемых элементов для управления скоростью обучения позволяет модели более эффективно использовать информацию из входных данных и адаптироваться к изменяющимся условиям, что в конечном итоге может привести к улучшению производительности и качества модели.

1.3.3. Использование управляемых элементов для управления весами регуляризации.

Использование управляемых элементов для управления весами регуляризации представляет собой еще один интересный аспект в области машинного обучения, особенно при работе с рекуррентными нейронными сетями (RNN).

Регуляризация играет важную роль в предотвращении переобучения модели путем добавления дополнительных ограничений к обучению, таких как штрафы за сложность модели или ограничения на веса параметров. Управляемые элементы могут быть использованы для динамического регулирования весов регуляризации в зависимости от контекста или текущего состояния модели.

Например, можно использовать информацию, полученную с помощью механизма внимания, для определения важности различных аспектов данных при применении регуляризации. Затем эта информация может быть использована для настройки весов регуляризации для каждого параметра или группы параметров в соответствии с их значимостью. Это может быть особенно полезно в случаях, когда некоторые части данных более важны для конкретной задачи, и требуется более активное подавление переобучения в этих областях.

В целом, использование управляемых элементов для управления весами регуляризации может помочь модели эффективнее использовать информацию из входных данных и более точно контролировать процесс обучения, что в конечном итоге может привести к улучшению обобщающей способности модели и предотвращению переобучения.

ГЛАВА 2. ПРЕДЛАГАЕМЫЙ АЛГОРИТМ НАСТРОЙКИ ПАРАМЕТРОВ ДЛЯ АНАЛИЗА ФИНАНСОВОГО РЫНКА.

2.1. Обзор предлагаемого алгоритма.

2.1.1. Мотивация и цели алгоритма.

Нейронные сети широко используются в финансовом анализе для прогнозирования рыночных трендов, оценки ценных бумаг и определения оптимальных инвестиционных стратегий.

Применение нейронных сетей для анализа финансовых данных обычно включает в себя следующие шаги:

1. Предварительная обработка данных: Этот этап включает выбор и подготовку финансовых данных для обучения модели. Данные могут включать временные ряды ценных бумаг, финансовые показатели компаний, объемы торгов и другие релевантные факторы. Предварительная обработка также может включать нормализацию данных и удаление выбросов.
2. Выбор архитектуры нейронной сети: В зависимости от конкретной задачи и доступных данных выбирается подходящая архитектура нейронной сети. Для анализа временных рядов часто используются рекуррентные нейронные сети (RNN) или их модификации, такие как долгая краткосрочная память (LSTM) или gated recurrent units (GRU).
3. Обучение модели: на данном этапе нейронная сеть обучается на реальных данных, чтобы научиться выявлять закономерности и тренды. Обучение включает в себя подачу входных данных на вход сети и корректировку параметров сети на основе ошибки между прогнозами и фактическими данными.
4. Тестирование и валидация модели: После обучения модели ее необходимо протестировать на отложенных данных, чтобы оценить ее производительность и точность прогнозирования. Это поможет убедиться, что модель способна делать адекватные прогнозы на новых данных.
5. Прогнозирование и анализ результатов: После успешного обучения и тестирования модели она может быть использована для прогнозирования будущих значений ценных бумаг или оценки рыночных трендов. Анализ результатов позволяет оценить эффективность модели и ее пригодность для конкретной задачи финансового анализа.

Обычный алгоритм обработки финансовых данных, особенно в контексте рекуррентных нейронных сетей, часто ограничивается обучением на последовательных данных без учета их контекста и значимости. В таких алгоритмах модель часто не имеет механизмов, позволяющих ей динамически адаптироваться к изменяющейся среде рынка.

В предложенном алгоритме с управляемыми элементами, такими как механизм «внимания», модель обучается с учетом контекста и значимости входных данных. Это позволяет модели фокусироваться на ключевых финансовых показателях и сигналах, что может значительно улучшить ее способность делать точные прогнозы и анализировать рыночные тренды.

Разница между обычным алгоритмом и предложенным заключается в том, что предложенный алгоритм:

1. Учитывает контекст и значимость данных: Модель с управляемыми элементами фокусируется на ключевых финансовых показателях и сигналах, что позволяет ей делать более точные прогнозы и анализировать рыночные тренды.

2. Динамически настраивает параметры: в предложенном алгоритме параметры модели могут динамически настраиваться в зависимости от текущего состояния рынка и значимости данных. Это помогает предотвратить переобучение модели и повысить ее способность к адаптации к изменяющимся условиям рынка.

3. Использует механизмы «Attention»: механизм «Attention» позволяют модели сосредотачиваться на наиболее важных финансовых показателях и сигналах, что повышает качество ее анализа и прогнозирования.

Таким образом, предложенный алгоритм имеет потенциал для более точного и информативного анализа финансовых рынков, что может быть полезно для инвесторов, трейдеров и финансовых аналитиков.

Цели алгоритма:

1. Улучшение производительности модели RNN на задачах анализа последовательностей данных.

2. Повышение точности прогнозирования модели за счет более эффективного использования информации.

3. Предотвращение переобучения путем динамической настройки параметров модели с учетом контекста и значимости данных.

2.1.2. Основные принципы алгоритма.

1. Использование управляемых элементов:  
В контексте обучения модели RNN важно использовать управляемые элементы, такие как механизм внимания. Это позволяет модели сосредотачиваться на ключевых аспектах финансовых данных, выявлять сигналы и шумы, а также адаптироваться к изменяющимся рыночным условиям. Механизм «Attention» позволяет модели динамически выбирать, на какие части данных сосредотачиваться, улучшая ее способность анализировать временные ряды и выявлять важные тренды.

2. Динамическая настройка параметров:  
Алгоритм должен быть способен динамически настраивать параметры модели в соответствии с текущим состоянием рынка и значимостью данных. Например, веса регуляризации и скорость обучения могут быть адаптированы в реальном времени в зависимости от волатильности рынка или важности конкретных финансовых показателей. Это позволяет модели быть более гибкой и эффективной в различных рыночных условиях.

3. Применение алгоритма в финансовом анализе:  
Алгоритм может быть применен для различных задач финансового анализа, включая прогнозирование ценных бумаг, анализ рыночной динамики, выявление финансовых трендов и определение оптимальных инвестиционных стратегий. Например, модель может использоваться для предсказания будущих изменений цен акций на основе исторических данных или для выявления связей между финансовыми показателями компаний и их рыночной стоимостью. Кроме того, алгоритм может быть полезен для управления портфелем, помогая инвесторам принимать обоснованные решения о распределении активов и управлении рисками.

2.2. Детали реализации алгоритма.

2.2.1. Выбор управляемых элементов.

Выбор управляемых элементов при анализе финансовых данных с использованием нейронных сетей играет ключевую роль в повышении качества анализа и прогнозирования. Вот некоторые из наиболее распространенных управляемых элементов и их роль в процессе:

1. Механизм внимания (Attention): Этот элемент позволяет модели фокусироваться на определенных аспектах входных данных, игнорируя менее важные или шумные сигналы. В контексте финансовых данных механизм внимания может быть использован для выявления ключевых финансовых показателей, таких как объем торгов, цены закрытия, финансовые отчеты и другие, которые могут оказывать значительное влияние на ценовую динамику активов.

2. Механизмы регуляризации: В процессе обучения нейронной сети могут использоваться различные методы регуляризации, такие как dropout или L2 регуляризация, чтобы предотвратить переобучение модели. Это особенно важно в случае финансовых данных, где могут быть сильные шумы и выбросы, а также ограниченное количество доступных данных.

3. Механизмы селективного внимания: Эти элементы позволяют модели выбирать определенные аспекты данных для анализа в зависимости от их значимости или актуальности.

4. Механизмы динамической адаптации: Эти элементы позволяют модели изменять свое поведение или параметры в зависимости от изменяющихся условий рынка или новых данных. Например, модель может динамически изменять скорость обучения или веса регуляризации в ответ на изменения в волатильности рынка или наличие новой информации.

Выбор конкретных управляемых элементов зависит от характеристик конкретной задачи анализа финансовых данных, доступных данных и требуемого уровня точности и гибкости модели. Эффективное использование этих элементов может значительно улучшить способность модели адаптироваться к сложным рыночным условиям и выявлять важные закономерности в данных.

В условиях динамического изменения финансового рынка для нашего алгоритма будем использовать механизм «Attention», который является ключевым при анализе рынка по следующим причинам:

1. Фокус на ключевых аспектах данных:  позволяет модели сосредоточиться на наиболее важных аспектах финансовых данных, игнорируя менее значимые или шумные сигналы. Это особенно важно в контексте анализа финансовых рынков, где релевантные факторы могут сильно варьироваться во времени и по мере изменения рыночных условий.

2. Адаптация к изменяющимся условиям рынка:  позволяет модели динамически адаптироваться к изменяющимся рыночным условиям, выбирая наиболее актуальные и значимые аспекты данных для анализа. Это повышает способность модели предсказывать ценовую динамику и выявлять тренды в реальном времени.

3. Учет контекста данных:  позволяет модели анализировать данные в контексте их взаимосвязей и временных зависимостей. Это помогает модели лучше понимать динамику рынка и выявлять важные закономерности в данных.

2.2.2. Обновление управляемых элементов.

Обновление управляемых элементов в модели, таких как механизм «Attention», является критически важным процессом для обеспечения её актуальности и эффективности. Эти элементы играют ключевую роль в адаптации модели к изменяющимся условиям окружающей среды или задачи, что особенно важно в контексте анализа финансовых рынков.

Процесс обновления управляемых элементов может быть осуществлен через несколько методов:

1. Обучение на новых данных: Регулярное обновление модели на свежих данных позволяет ей учитывать новые тренды, паттерны и изменения на рынке. Этот метод позволяет модели адаптироваться к текущим условиям и оставаться релевантной в динамической среде.

2. Файн-тюнинг: Этот подход предполагает изменение параметров модели, включая управляемые элементы, на основе новых данных или изменения задачи. После обучения модели на базовых данных, её можно дообучить на конкретном наборе данных, чтобы лучше соответствовать требованиям текущей задачи или смене рыночных условий.

3. Использование адаптивных механизмов: Модели могут быть добавлены механизмы, позволяющие им автоматически адаптироваться к изменяющимся условиям рынка. Например, механизмы автоматической регуляризации или адаптивного обучения могут регулировать параметры модели в реальном времени, основываясь на новых данных или обратной связи.

4. Мониторинг и реакция: Постоянное мониторирование работы модели и её результатов позволяет выявлять необходимость изменений в управляемых элементах. При обнаружении несоответствий или ухудшения производительности модели могут быть приняты меры по её обновлению или перенастройке управляемых элементов.

Для анализа трендов на финансовом рынке, где требуется быстрая адаптация к меняющимся условиям, наиболее эффективным методом обновления будет использование адаптивных механизмов. Эти механизмы позволяют модели быстро реагировать на изменения в реальном времени, выделяя ключевые аспекты данных и адаптируясь к новым условиям на рынке.

Механизм Attention, упомянутый в тексте, является примером такого адаптивного механизма, который может быстро переключаться между различными аспектами данных в зависимости от их актуальности для текущего тренда на рынке. Такая гибкость позволяет модели эффективно анализировать и интерпретировать данные в реальном времени, что важно для успешного прогнозирования и анализа трендов на финансовых рынках.

2.2.3. Обновление весов сети.

Для обновления весов сети в нашем алгоритме, использующем механизм Attention для анализа финансового рынка, можно применить стандартные методы обратного распространения ошибки (backpropagation) с адаптацией под условия конкретной задачи, рассмотрим некоторые из них.

1. Среднеквадратичная ошибка (MSE) и средняя абсолютная ошибка (MAE).

* **Среднеквадратичная ошибка (MSE)**: функция потерь, которая вычисляет среднее значение квадрата разности между предсказанными и фактическими значениями.
* **Средняя абсолютная ошибка (MAE)**: функция потерь, которая вычисляет среднее абсолютное значение разности между предсказанными и фактическими значениями.

1. Применение алгоритма обратного распространения ошибки для вычисления градиентов функции потерь по всем весам сети, что позволит определить направление обновления весов сети для минимизации функции потерь.
2. Для обновления весов сети можно использовать различные методы оптимизации, такие как стохастический градиентный спуск , адаптивные методы оптимизации (Adam, RMSProp).
3. **Управление обновлением весов**: применение регуляризации для предотвращения переобучения или настройка скорости обучения в зависимости от характера данных и их распределения на финансовом рынке.

Для задачи анализа финансового рынка с использованием механизма Attention в рекуррентных нейронных сетях с управляемыми элементами, одним из подходящих алгоритмов для обновления весов сети может быть SGD. SGD - это метод оптимизации, который обычно демонстрирует хорошую производительность при обучении нейронных сетей. Он эффективно сочетает в себе преимущества методов адаптивного градиентного спуска и метода моментов.

Наш алгоритм обучения может использовать SGD для эффективного обновления весов, что поможет модели находить более точные и информативные паттерны на финансовых данных.

2.3. Гиперпараметры алгоритма и их влияние на производительность

2.3.1. Скорость обучения.

Скорость обучения сети, также известная как темп обучения, представляет собой параметр, определяющий величину шага, с которой алгоритм оптимизации изменяет веса сети в процессе обучения. Этот параметр определяет, насколько сильно изменяются веса сети в ответ на градиенты, рассчитанные на каждом шаге обучения.

Скорость обучения важна для достижения оптимального баланса между скоростью сходимости обучения и качеством обученной модели. Если скорость обучения слишком высока, это может привести к расходимости обучения или перепрыгиванию через оптимальные значения. С другой стороны, слишком низкая скорость обучения может замедлить процесс сходимости или застрять в локальных минимумах.

Для алгоритма, который разрабатывается для анализа финансового рынка, скорость обучения играет критическую роль в достижении оптимальных результатов.

В контексте финансового анализа, где данные могут быть подвержены большой степени изменчивости и шумов, важно выбрать скорость обучения, которая обеспечивает быструю сходимость модели к оптимальным значениям весов, при этом минимизируя возможность переобучения или недообучения.

Рекомендуется провести серию экспериментов с различными значениями скорости обучения, начиная с относительно низких значений и постепенно увеличивая её, чтобы определить оптимальный диапазон. Также можно использовать методы адаптивного управления скоростью обучения, чтобы динамически регулировать скорость обучения во время процесса обучения в зависимости от динамики данных и поведения модели.

Такой подход поможет нам достичь баланса между скоростью сходимости и качеством модели, что важно для успешного применения алгоритма в анализе финансовых рынков.

2.3.2. Коэффициент импульса.

Коэффициент импульса, или параметр момента, является ключевым элементом во многих алгоритмах оптимизации, используемых в обучении нейронных сетей. Он играет важную роль в ускорении сходимости алгоритмов оптимизации и повышении их устойчивости.

В контексте оптимизации весов нейронных сетей, коэффициент импульса определяет, насколько сильно предыдущее обновление весов должно влиять на текущее обновление. Это позволяет стабилизировать процесс обучения, уменьшая колебания весовых значений и способствуя плавному движению весов к оптимальным значениям.

Использование коэффициента импульса позволяет алгоритму оптимизации принимать во внимание не только текущий градиент, но и предыдущие изменения весов, что помогает преодолевать локальные минимумы и ускорять сходимость к глобальному оптимуму.

Важно подбирать оптимальное значение коэффициента импульса в зависимости от конкретной задачи и структуры данных. Эксперименты показывают, что правильно настроенный коэффициент импульса может значительно улучшить процесс обучения нейронных сетей, делая его более эффективным и стабильным.

2.3.3. Коэффициент регуляризации.

Коэффициент регуляризации - это параметр, который добавляется к функции потерь во время обучения модели с целью предотвратить переобучение. В контексте нашей задачи разработки алгоритма для анализа финансового рынка с использованием механизма Attention, коэффициент регуляризации может быть полезным инструментом для контроля сложности модели и улучшения её обобщающей способности.

Часто используемые виды регуляризации включают L1 и L2 регуляризацию. L1 регуляризация добавляет абсолютное значение весов к функции потерь, тогда как L2 регуляризация добавляет квадраты весов. Это помогает предотвратить переобучение путем штрафования больших весов и способствует созданию более простых моделей.

Для определения оптимального значения коэффициента регуляризации можно использовать методы кросс-валидации или экспериментировать с различными значениями в диапазоне от нуля до бесконечности, чтобы найти баланс между уменьшением переобучения и сохранением хорошей обобщающей способности модели.

ГЛАВА 3. ЭКСПЕРИМЕНТАЛЬНАЯ ОЦЕНКА.

3.1. Описание наборов данных и экспериментальной установки

3.1.1. Описание наборов данных.

В качестве данных будем использовать цены акций компании «Google». С помощью библиотеки yifinance собираем данные по акциям за последние 10 лет и импортируем их в датасет.

import yfinance as yf

# Выбираем акции компании гугл

ticker = "GOOGL"

# Задаем временный интервал

start\_date = "2014-01-01"

end\_date = "2024-04-01"

# Получаем данные об акциях и просматриваем первые 5 записей детасета

data = yf.download(ticker, start=start\_date, end=end\_date)

data.head()

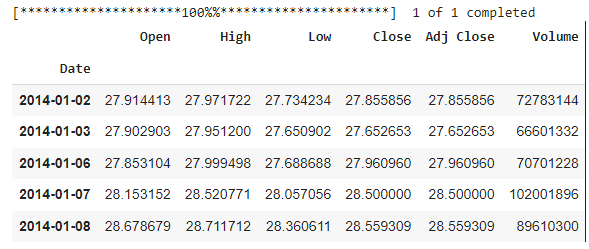


Рис. 13. Структура датасета.

Датасет содержит следующие столбцы: дата, цена открытия, наибольшая цена, наименьшая цена, цена закрытия, скорректированная цена закрытия и объем торгов.

Для обучения модели понадобятся следующие столбцы: дата, цена открытия, скорректированная цена закрытия и объем торгов.

3.1.2. Обработка датасета.

Проверяем датасет на пропуски и наличие выбросов.

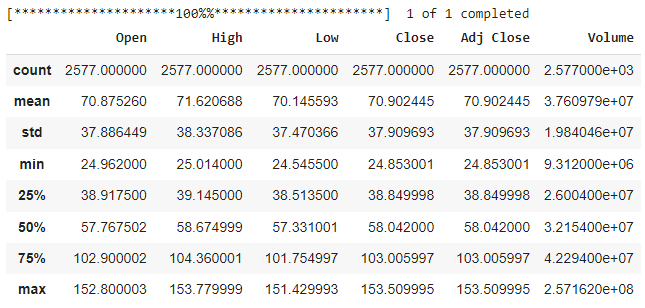


Рис.14. Статистические данные столбцов датасета.

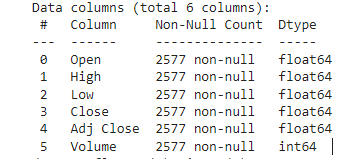


Рис.15. Информация о столбцах датасета

Датасет содержит 2577 строк данных. Никаких выбросов и шумов не наблюдается, поэтому накаких преобразований численных данных не требуется.

Визуализация данных.

startDate = datetime.datetime(2014, 1, 1)

endDate = datetime.datetime(2024, 4, 1)

google\_data = yf.Ticker('GOOGL')

google\_df = google\_data.history(start=startDate, end=endDate)

google\_df['Close'].plot(title="GOOGL stock")

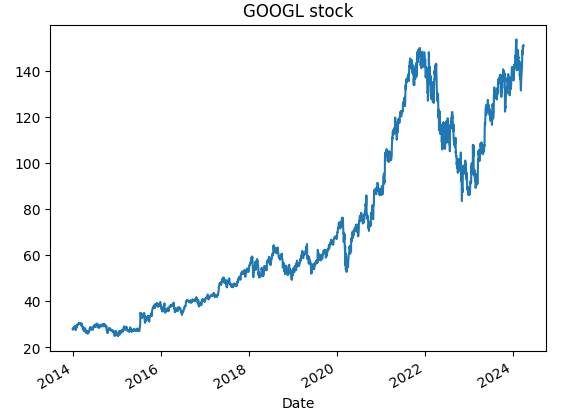


Рис.16. График распределения цены акций.

Для обучения модели понадобятся не все столбцы, поэтому удаляем из датасета ненужные столбцы.

column = ['Open', 'Adj Close', 'Volume']

data\_new = data[column]

data\_new.head()

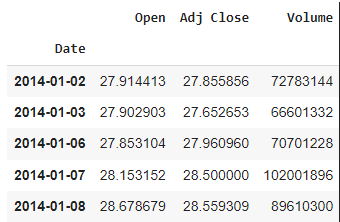


Рис.17. Обработанный датасет, используемый для машинного обучения

3.2. Оценка производительности предлагаемого алгоритма

3.2.1. Разработка алгоритма

Цель модели – по столбцам “Open” и “Volume” предсказать цену закрытия ( столбец “Adj close”).

Как видно по графику, данные распределены криволинейно, поэтому для масштабирования признака будем использовать метод MinMaxScaler. Для обучения разобьем датасет на тестовый и тренировочный в пропорции 0,2 на 0,8 соответственно. Изначально будем использовать 10 слоев RNN, а также 10 эпох обучения.

Ниже представлен полный код для обучения модели.

import yfinance as yf

import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras import layers, Model, regularizers

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error, mean\_absolute\_percentage\_error, r2\_score

from scipy.stats import spearmanr

# Получение данных с помощью Yahoo Finance API

ticker = "GOOGL"

start\_date = "2014-01-01"

end\_date = "2024-04-01"

df = yf.download(ticker, start=start\_date, end=end\_date)

# Выбор нужных столбцов

column = ['Open', 'Adj Close', 'Volume']

df\_new = df[column]

# Преобразование данных

scaler = MinMaxScaler()

df\_scaled = scaler.fit\_transform(df\_new)

# Параметры модели

sequence\_length = 10

n\_features = 2

n\_attention\_heads = 3

l1\_reg = 0.01

l2\_reg = 0.01

momentum = 0.9

# Подготовка данных

def prepare\_data(data):

    X, y = [], []

    for i in range(len(data) - sequence\_length):

        X.append(data[i:i+sequence\_length, :-1])

        y.append(data[i+sequence\_length, -2])  # Изменение индекса для предсказания столбца 'Adj Close'

    return np.array(X), np.array(y)

X, y = prepare\_data(df\_scaled)

# Разделение данных на тренировочную и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Рекуррентная модель с механизмом Attention и GRU

inputs = layers.Input(shape=(sequence\_length, n\_features))

attention = layers.MultiHeadAttention(num\_heads=n\_attention\_heads, key\_dim=64)(inputs, inputs)

gru\_out = layers.GRU(64, return\_sequences=True, kernel\_regularizer=regularizers.l1\_l2(l1=l1\_reg, l2=l2\_reg))(attention)

out = layers.Flatten()(gru\_out)

out = layers.Dense(64, activation='relu', kernel\_regularizer=regularizers.l1\_l2(l1=l1\_reg, l2=l2\_reg))(out)

out = layers.Dense(1, kernel\_regularizer=regularizers.l1\_l2(l1=l1\_reg, l2=l2\_reg))(out)

model = Model(inputs, out)

# Компиляция модели с оптимизатором SGD с моментом

optimizer = tf.keras.optimizers.SGD(momentum=momentum)

model.compile(optimizer=optimizer, loss='mse')

# Обучение модели на тренировочных данных

model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.1)

3.2.2. Оценка метрик модели.

Рассмотрим наиболее распространенные метрики оценки модели машинного обучения:

1. Среднеквадратичная ошибка (Mean Squared Error, MSE) - это одна из наиболее распространенных метрик оценки точности моделей в статистике и машинном обучении. Она измеряет среднее значение квадратов разностей между фактическими и прогнозируемыми значениями. Математически формула MSE выглядит следующим образом:

* *n* - количество наблюдений или элементов в выборке.
* *yi*​ - фактическое значение (наблюдаемое значение) для *i*-го элемента выборки.
* ​ - прогнозируемое значение для *i*-го элемента выборки, полученное моделью.

Процесс расчета MSE состоит из следующих шагов:

1. Разница между фактическими значениями *yi*​ и прогнозируемыми значениями ​ вычисляется для каждого наблюдения.
2. Каждая разница возводится в квадрат, чтобы учесть как положительные, так и отрицательные ошибки.
3. Полученные квадраты разностей суммируются.
4. Сумма делится на общее количество наблюдений, чтобы получить среднее значение.

MSE позволяет оценить, насколько близко прогнозы модели к фактическим значениям. Чем меньше значение MSE, тем лучше модель предсказывает данные. Важно отметить, что MSE показывает среднеквадратичную ошибку на каждом отдельном наблюдении, а не только общую ошибку модели.

2. Средняя абсолютная ошибка (Mean Absolute Error, MAE) - это еще одна распространенная метрика оценки точности моделей в статистике и машинном обучении. В отличие от MSE, которая учитывает квадраты разностей между фактическими и прогнозируемыми значениями, MAE измеряет среднее абсолютное значение разностей.

Математически формула MAE выглядит следующим образом:

Где:

* *n* - количество наблюдений или элементов в выборке.
* *yi*​ - фактическое значение (наблюдаемое значение) для *i*-го элемента выборки.
* - прогнозируемое значение для *i*-го элемента выборки, полученное моделью.

Процесс расчета MAE состоит из следующих шагов:

1. Разница между фактическими значениями *yi*​ и прогнозируемыми значениями ​ вычисляется для каждого наблюдения.
2. Берется абсолютное значение этой разницы, чтобы учесть только величину ошибки, независимо от ее направления (положительная или отрицательная).
3. Полученные абсолютные разности суммируются.
4. Сумма делится на общее количество наблюдений, чтобы получить среднее значение.

MAE позволяет оценить среднюю величину отклонения между фактическими и прогнозируемыми значениями. Чем меньше значение MAE, тем лучше модель предсказывает данные. Важно отметить, что MAE не учитывает направление отклонений и измеряет среднее абсолютное отклонение модели от фактических данных.

3. Коэффициент детерминации ( R-квадрат) - это статистическая мера, которая представляет собой долю дисперсии зависимой переменной, объясненную (или предсказанную) моделью. Он обычно используется для оценки того, насколько хорошо модель соотносится с данными.

Математически коэффициент детерминации определяется следующим образом:

Где:

* *Sres*​ - сумма квадратов остатков (расхождений между фактическими и предсказанными значениями).
* *Stot*​ - общая сумма квадратов (дисперсия фактических значений).

Значение  находится в диапазоне от 0 до 1:

* означает идеальное соответствие модели данным, когда все вариации в данных объясняются моделью.
* означает, что модель не объясняет вариации в данных лучше, чем просто среднее значение зависимой переменной.
* может быть отрицательным, если модель плохо соотносится с данными.

предоставляет информацию о том, какую долю изменчивости зависимой переменной можно объяснить с помощью независимых переменных в модели. Чем ближе значение к 1, тем лучше модель соотносится с данными.

4. Средняя абсолютная процентная ошибка (Mean Absolute Percentage Error, MAPE) - это метрика, используемая для измерения точности прогнозов моделей, особенно в области прогнозирования временных рядов. Она измеряет среднее абсолютное значение процентного отклонения между фактическими и прогнозируемыми значениями.

Математически формула MAPE выглядит следующим образом:

* *n* - количество наблюдений или элементов в выборке.
* *yi*​ - фактическое значение (наблюдаемое значение) для *i*-го элемента выборки.
* - прогнозируемое значение для *i*-го элемента выборки, полученное моделью.

Процесс расчета MAPE состоит из следующих шагов:

1. Разница между фактическими значениями *yi*​ и прогнозируемыми значениями ​ вычисляется для каждого наблюдения.
2. Разница делится на фактическое значение *yi*​, чтобы получить относительное отклонение в виде доли.
3. Берется абсолютное значение этого относительного отклонения.
4. Полученные абсолютные относительные отклонения суммируются.
5. Сумма делится на общее количество наблюдений, чтобы получить среднее значение.
6. Результат умножается на 100% для получения процентного значения.

MAPE позволяет оценить среднее абсолютное процентное отклонение между фактическими и прогнозируемыми значениями. Чем меньше значение MAPE, тем лучше модель предсказывает данные. Важно отметить, что MAPE учитывает относительное отклонение и показывает процентное отличие прогнозов модели от фактических данных.

5. Коэффициент Спирмена - это статистическая мера, используемая для оценки степени монотонной связи между двумя переменными. В отличие от корреляции Пирсона, коэффициент Спирмена не требует, чтобы данные были нормально распределены, и может использоваться для оценки связи даже в случае наличия выбросов или нелинейной зависимости.

Значение коэффициента Спирмена находится в диапазоне от -1 до 1:

* *ρ*=1 указывает на полную монотонную положительную связь между переменными.
* *ρ*=−1 указывает на полную монотонную отрицательную связь между переменными.
* *ρ*=0 указывает на отсутствие монотонной связи между переменными.

Коэффициент Спирмена позволяет оценить степень монотонной связи между переменными, игнорируя конкретные значения переменных и сосредотачиваясь только на порядке их расположения. Это делает его полезным инструментом при работе с данными, которые не соответствуют требованиям нормального распределения или имеют выбросы.

Для оценки модели используем эти метрики.

Ниже представлен код для поиска всех метрик.

# Оценка модели

loss = model.evaluate(X\_test, y\_test) #Средняя квадратическая ошибка

r\_squared = r2\_score(y\_test, y\_pred) #Коэффициент детерминации

mae = mean\_absolute\_error(y\_test, y\_pred) #Среднее абсолютное значение разности между предсказанными значениями модели и их соответствующими истинными значениями

mape = mean\_absolute\_percentage\_error(y\_test, y\_pred) #Средняя абсолютная процентная ошибка между предсказанными и истинными значениями

spearman\_corr, \_ = spearmanr(y\_test, y\_pred) # Коэффициент Спирмена

print("Spearman's correlation coefficient:", spearman\_corr)

print("R-squared:", r\_squared)

print("MAE:", mae)

print("MAPE:", mape)

print("Test Loss:", loss)

После обучения модели и вычисления метрик, получаем следующие значения.

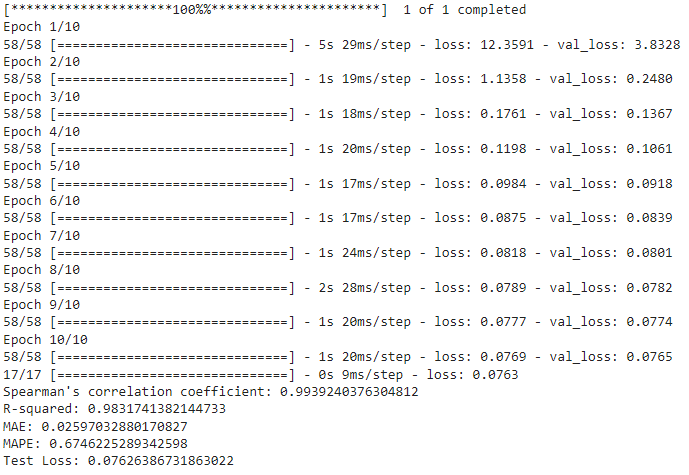


Рис.18. Полученные результаты обучения модели

Коэффициент детерминации и коэффициент Спирмена близки к 1, что указывает на высокую корреляцию признаков. Только показатель MAPE дает довольно большое значение – 67%, что может указывать на большую разницу между фактическим и предсказанными значениями.

Для визуализации обучения, можно построить кривые обучения тренировочного и тестового частей датасета.

    plt.plot(history\_dict['loss'], label='Training Loss')

    plt.plot(history\_dict['val\_loss'], label='Validation Loss')

    plt.title('Training and Validation Loss')

    plt.xlabel('Epoch')

    plt.ylabel('Loss')

    plt.legend()

    plt.show()

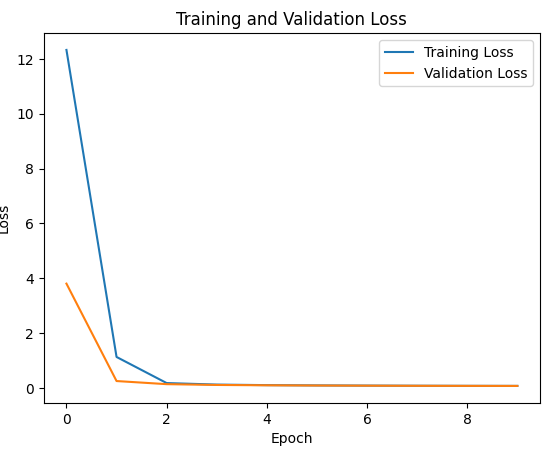


Рис.19. Кривые обучения.

Как можно наблюдать на графике, уже на 2 эпохе модель дает довольно маленькую и практически одинаковые ошибку на тестовом и тренировочном датасете, что может указывать на одну из причин отсутствия переобучения модели.

Построим график распределения ошибок модели.

# Вычисление ошибок

errors = y\_pred.flatten() - y\_test

# Построение гистограммы распределения ошибок

plt.figure(figsize=(10, 6))

plt.hist(errors, bins=50, edgecolor='k')

plt.title('Распределение ошибок')

plt.xlabel('Ошибка')

plt.ylabel('Частота')

plt.grid(True)

plt.show()

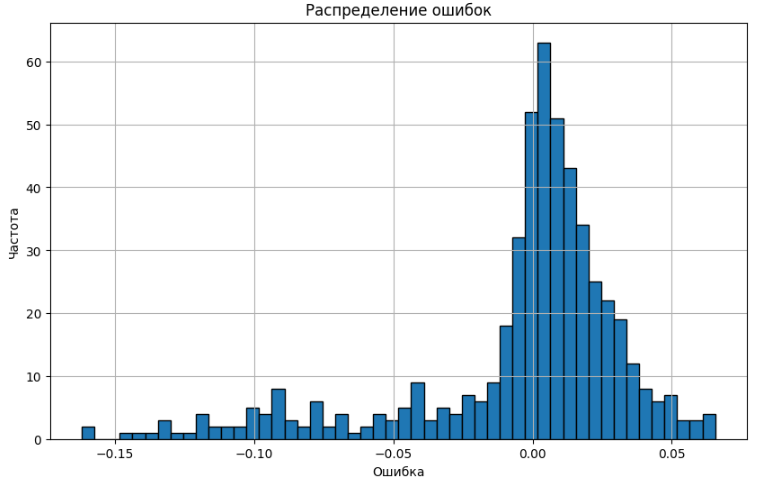


Рис.20. График распределения ошибок

Распределение ошибок модели имеет вид очень близкий к нормальному, что может свидетельствовать о следующем:

* Отсутствие систематической ошибки: Ошибки модели распределены случайным образом вокруг среднего значения, что указывает на отсутствие систематического смещения в сторону переоценки или недооценки.
* Хорошая генерализация: Модель хорошо обобщает данные обучения и может делать надежные прогнозы для новых данных.
* Стабильная производительность: Ошибки модели относительно стабильны для различных подмножеств данных, что указывает на робастность модели.
* Отсутствие выбросов: Распределение ошибок не имеет значительных выбросов, что свидетельствует о том, что модель не делает необычно больших ошибок.
* Подходящая сложность модели: Модель не переобучена и не недообучена, что приводит к оптимальной производительности.

Нормальное распределение ошибок желательно для моделей машинного обучения, поскольку оно указывает на то, что модель делает надежные и последовательные прогнозы. Однако следует отметить, что нормальное распределение не всегда является признаком хорошей производительности модели. Например, если средняя ошибка велика, модель может иметь низкую точность, даже если ошибки распределены нормально.

Кроме того, нормальное распределение ошибок может быть нереалистичным для некоторых типов данных или задач. Например, в задачах классификации ошибки могут быть ограничены диапазоном от 0 до 1, что может привести к скошенному распределению ошибок.

3.3. Анализ влияния управляемых элементов на обучение модели.

3.3.1. Влияние управляемых элементов на скорость обучения.

Управляемые элементы в рекуррентных нейронных сетях (RNN) могут существенно влиять на скорость обучения. Вот как они влияют на этот процесс:

1. Градиентный взрыв и исчезновение

Управляемые элементы в рекуррентных нейронных сетях (RNN) регулируют поток градиентов через сеть. Градиенты - это производные функции потерь по параметрам модели. Они указывают направление, в котором необходимо обновить параметры, чтобы уменьшить функцию потерь.

Если управляемые элементы слишком сильны, градиенты могут взорваться, что приведет к нестабильности обучения. Это происходит потому, что сильные управляемые элементы усиливают градиенты на каждом временном шаге, что приводит к экспоненциальному росту градиентов.

Если управляемые элементы слишком слабы, градиенты могут исчезнуть, что замедлит обучение. Это происходит потому, что слабые управляемые элементы уменьшают градиенты на каждом временном шаге, что приводит к экспоненциальному уменьшению градиентов.

2. Зависимость от времени

Управляемые элементы вносят зависимость от времени в RNN. Это означает, что выходные данные сети на каждом временном шаге зависят не только от входных данных на этом шаге, но и от выходных данных на предыдущих шагах.

Зависимость от времени может привести к долгосрочным зависимостям, которые требуют большего количества шагов обучения для сходимости. Например, в задаче предсказания временных рядов модель должна научиться запоминать прошлые значения для точного прогнозирования будущих значений.

Более длинные последовательности могут усложнить обучение из-за эффекта градиентного исчезновения. По мере того, как градиенты распространяются назад во времени через управляемые элементы, они могут уменьшаться, что затрудняет обучение сети на долгосрочных зависимостях.

3. Сложность модели

Управляемые элементы увеличивают сложность модели RNN. Более сложные модели требуют большего количества данных и вычислительных ресурсов для обучения. Это может замедлить процесс обучения.

Кроме того, управляемые элементы вводят дополнительные гиперпараметры, которые необходимо настраивать. Оптимальные значения этих параметров могут варьироваться в зависимости от задачи и данных. Настройка гиперпараметров может быть трудоемким процессом, что замедляет обучение.

4. Алгоритмы оптимизации

Управляемые элементы в RNN могут как ускорить, так и замедлить скорость обучения в зависимости от конкретной задачи и реализации модели. Понимание их влияния и тщательная настройка гиперпараметров имеют решающее значение для оптимизации процесса обучения.

Для наглядности приведем таблицу полученных значений скорости обучения от различных параметров.

Таблица 1. Зависимость времени обучения от изменения настраиваемых элементов модели.

|  |  |  |  |
| --- | --- | --- | --- |
| Данные, взятые за количество лет | Количество эпох | Количество слоев RNN | Время выполнения в секундах |
| 10 | 10 | 10 | 27 |
| 10 | 30 | 10 | 47 |
| 10 | 10 | 30 | 40 |
| 10 | 30 | 30 | 97 |
| 15 | 30 | 30 | 137 |
| 15 | 10 | 30 | 57 |
| 15 | 30 | 10 | 87 |
| 15 | 10 | 10 | 28 |

3.3.2. Влияние управляемых элементов на точность.

Управляемые элементы в рекуррентных нейронных сетях (RNN) могут влиять на точность модели несколькими способами:

1. Регулирование градиентов

Управляемые элементы регулируют поток градиентов через сеть. Это может помочь предотвратить градиентный взрыв, который может привести к нестабильности обучения и снижению точности.

Кроме того, управляемые элементы могут помочь уменьшить градиентное исчезновение, которое может замедлить обучение и привести к плохой точности на долгосрочных зависимостях.

2. Зависимость от времени

Управляемые элементы вносят зависимость от времени в RNN. Это означает, что выходные данные сети на каждом временном шаге зависят не только от входных данных на этом шаге, но и от выходных данных на предыдущих шагах.

Зависимость от времени позволяет RNN изучать долгосрочные зависимости, что может улучшить точность на задачах, требующих запоминания прошлых событий. Например, в задаче предсказания временных рядов модель должна научиться запоминать прошлые значения для точного прогнозирования будущих значений.

3. Сложность модели

Управляемые элементы увеличивают сложность модели RNN. Более сложные модели могут быть более мощными и достигать более высокой точности.

Однако более сложные модели также требуют большего количества данных и вычислительных ресурсов для обучения. Это может замедлить процесс обучения и привести к переобучению, что может снизить точность на невидимых данных.

4. Настройка гиперпараметров

Управляемые элементы вводят дополнительные гиперпараметры, которые необходимо настраивать. Оптимальные значения этих параметров могут варьироваться в зависимости от задачи и данных.

Тщательная настройка гиперпараметров может улучшить точность модели. Однако настройка гиперпараметров может быть трудоемким процессом, и поиск оптимальных значений может занять много времени.

Влияние слоев RNN, количества эпох, коэффициента регуляризации L1/L2 и коэффициента импульса на точность модели

Слои RNN:

* Увеличение количества слоев RNN может улучшить точность модели, позволяя сети изучать более сложные зависимости.
* Добавление большего количества слоев также увеличивает сложность модели и может привести к переобучению.

Количество эпох:

* Количество эпох определяет, сколько раз модель проходит через весь набор данных во время обучения.
* Увеличение количества эпох может улучшить точность модели, но также может привести к переобучению.

Коэффициент регуляризации L1/L2:

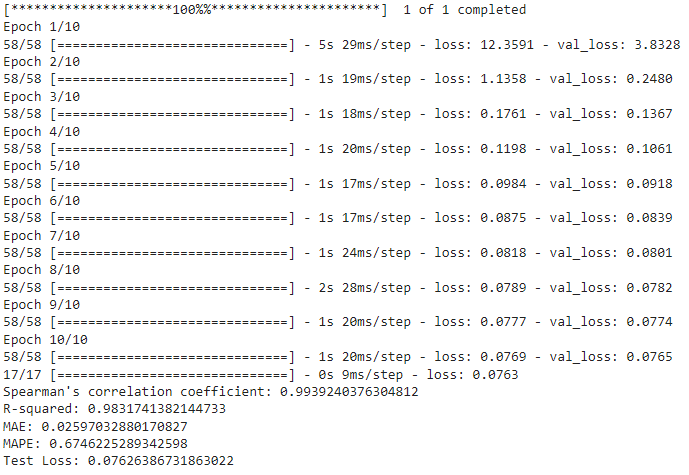
* Регуляризация L1/L2 добавляет штрафной член к функции потерь, который поощряет более простые модели, что может помочь предотвратить переобучение и улучшить точность на невидимых данных.

Коэффициент импульса:

* Скорость обучения является одним из ключевых гиперпараметров при обучении нейронных сетей. Она помогает ускорить обучение и может улучшить точность модели.

Управляемые элементы, слои RNN, количество эпох, коэффициент регуляризации L1/L2 и коэффициент импульса являются важными гиперпараметрами, которые могут влиять на точность модели RNN. Тщательная настройка этих параметров имеет решающее значение для оптимизации производительности модели для конкретных задач.

Рассмотрим рис.9, приведенный ниже:



Количество эпох, как показала практика, в размере 10 штук дает довольно точные результаты и менять их пока особого смысла не имеет.

Рассмотрим боле подробно коэффициент регуляризации и коэффициент импульса. Наиболее частые значения для коэффициента импульса лежат в диапазоне 0,9-0,99 ,а для коэффициентов регуляризации 0,01-0,00001.  
 На рис.9 коэффициент импульса равен 0,9 , коэффициенты регуляризации равны 0,01. Изменим параметры на 0,99 и 0,0001 соответственно и посмотрим как изменятся метрики модели.

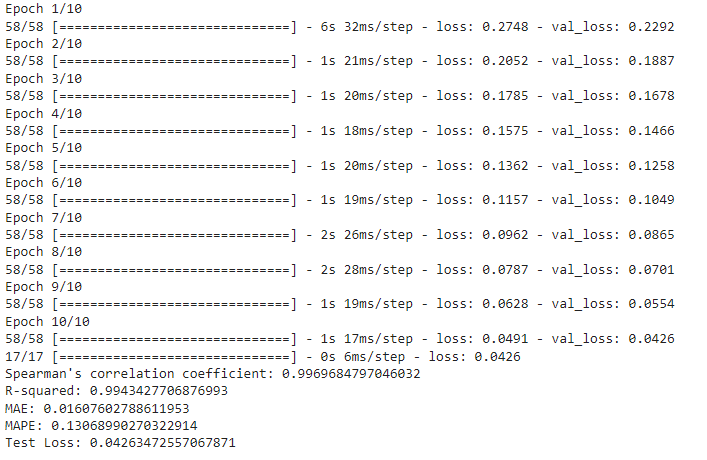


Рис.21. Результаты метрик при коэффициенте регуляризации равным 0,0001 и коэффициенте импульса равным 0,99

Можно увидеть как все показатели метрик улучшились и показывают более точные результаты, особенно показатель MAPE, который с 67% уменьшился до 13%.

Рассмотрим обучение модели при больших значениях коэффициента регуляризации(0,1) и маленького значения коэффициента импульса(0,5).

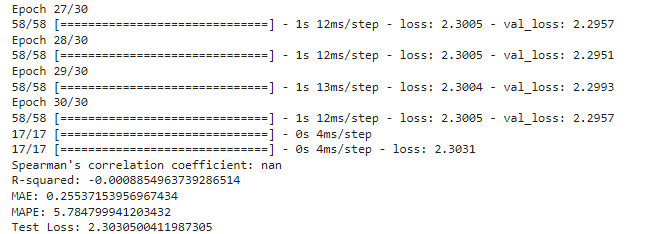


Рис.22. Результаты метрик при коэффициенте регуляризации равным 0,1 и коэффициенте импульса равным 0,5

Сразу можно заметить, как возросла ошибка на тренировочном и тестовом датасетах, а также показатели метрик значительно ухудшились, даже с учетом увеличения количества эпох обучения.

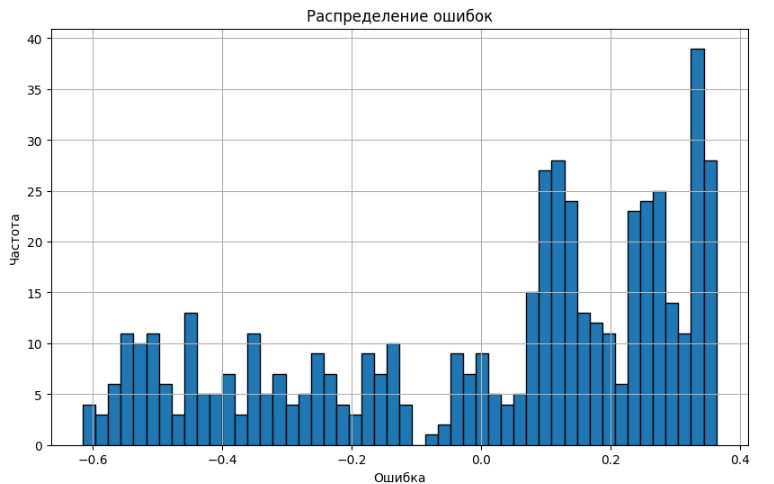


Рис.23. График распределения ошибок при изменении параметров сети.

Распределение ошибок также утратило нормальный вид, что свидетельствует о ухудшении обучения.

ГЛАВА 4. ПРИЛОЖЕНИЯ И РАСШИРЕНИЯ.

4.1. Использование других управляемых элементов.

Для обучения модели в основном варьировались следующие управляемые элементы: коэффициент регуляризации (L1, L2), коэффициент импульса и количество эпох.

Рассмотрим какие еще управляемые элементы можно изменять или добавить в модель обучения.

**1. Размерность GRU слоя**.

GRU (Gated Recurrent Unit) - это тип рекуррентного слоя, который часто используется в моделях глубокого обучения для работы с последовательными данными. Он имеет несколько параметров, которые могут быть настроены для оптимизации модели:

* Количество блоков GRU: Это количество независимых GRU-ячеек, которые будут использоваться в слое. Большее количество блоков может увеличить способность модели запоминать долгосрочные зависимости, но при этом требует больше вычислительных ресурсов.
* Размерность выходного пространства: Это количество единиц, возвращаемых GRU-слоем на каждом временном шаге. Увеличение размерности может увеличить емкость модели и ее способность извлекать более сложные зависимости, но также может повысить сложность обучения.

2. Количество и размерность голов в механизме Attention:

Механизм Attention позволяет модели фокусироваться на определенных частях входных данных при принятии решений. Многоголовой механизм внимания позволяет модели обучаться нескольким независимым "взглядам" на данные. Для настройки механизма внимания можно изменять следующие параметры:

* **Количество голов**: Это количество независимых "взглядов" на входные данные. Большее количество голов может улучшить способность модели сосредотачиваться на различных аспектах данных.
* **Размерность голов**: Это размерность пространства запросов, ключей и значений для каждой головы внимания. Увеличение размерности может улучшить способность модели работать с более сложными данными, но может также увеличить вычислительную сложность модели.

3. Скорость обучения:

Скорость обучения является одним из ключевых гиперпараметров при обучении нейронных сетей. Она определяет темп, с которым модель обновляет свои параметры на основе градиентов, вычисленных на обучающих данных. Для настройки скорости обучения можно использовать следующие методы:

* **Использование различных оптимизаторов**: Разные оптимизаторы могут по разному адаптировать скорость обучения модели.
* **Экспериментирование с различными значениями скорости обучения**: Разные значения скорости обучения могут подходить для разных задач и архитектур моделей. Обычно начинают с небольших значений и постепенно увеличивают скорость обучения в процессе обучения.

4. Размер батча и количество эпох:

Размер батча определяет количество обучающих примеров, обрабатываемых моделью за одну итерацию обучения, а количество эпох - количество полных проходов через все обучающие данные. Для настройки размера батча и количества эпох можно использовать следующие стратегии:

* **Изменение размера батча**: Большие размеры батчей могут ускорить обучение и повысить стабильность градиентов, но могут требовать больших объемов памяти GPU.
* **Изменение количества эпох**: Увеличение количества эпох может улучшить обобщающую способность модели, но может также привести к переобучению на обучающих данных.

5. Архитектурные изменения:

Возможность экспериментировать с архитектурными изменениями позволяет исследовать различные комбинации слоев, их размерностей, функций активации и других параметров модели:

* **Добавление или удаление слоев**: Эксперименты с добавлением или удалением слоев могут помочь определить оптимальную глубину модели.
* **Изменение размерности слоев**: Изменение размерности слоев может увеличить или уменьшить емкость модели, что может повлиять на ее способность обучаться на данных.
* **Применение различных функций активации**: Использование различных функций активации, таких как ReLU, Leaky ReLU, ELU и другие, может улучшить обучение модели.

6. Функция потерь.

Функция потерь (loss function) - это метрика, которая оценивает, насколько хорошо модель справляется с задачей обучения. Она измеряет расхождение между предсказанными значениями модели и их фактическими значениями на обучающих данных. Выбор подходящей функции потерь зависит от типа задачи (регрессии, классификации, сегментации и т. д.) и особенностей данных.

Ниже рассмотрим несколько типов функций потерь и их особенности:

1. **Mean Squared Error (MSE)**:

MSE является одной из самых распространенных функций потерь для задач регрессии. Она измеряет среднеквадратичную ошибку между фактическими значениями  и предсказанными значениями .MSE штрафует модель за большие ошибки.

1. **Binary Cross-Entropy Loss**:

Binary Cross-Эта функция потерь используется в задачах бинарной классификации. Она штрафует модель за неправильные предсказания, причем штраф увеличивается с ростом расхождения между фактическими и предсказанными вероятностями.

1. **Categorical Cross - Entropy Loss**:

Categorical Cross - Эта функция потерь используется в задачах многоклассовой классификации. Она измеряет расхождение между фактическими и предсказанными распределениями вероятностей для нескольких классов.

1. Huber Loss - это альтернативная функция потерь для задач регрессии, которая менее чувствительна к выбросам по сравнению с MSE. Она использует квадратичную функцию потерь для малых ошибок и линейную для больших.
2. Quantile Loss - это функция потерь, используемая для оценки квантилей в регрессионных моделях. Она позволяет модели обучаться таким образом, чтобы минимизировать квантильные ошибки, что полезно, например, при прогнозировании интервалов уверенности.

Выбор подходящей функции потерь зависит от конкретной задачи, свойств данных и требуемого поведения модели. Экспериментирование с различными функциями потерь может помочь найти оптимальное решение для конкретной задачи обучения.

4.2. Применение алгоритма к другим типам нейронных сетей.

Полученный алгоритм можно адаптировать и применить к другим типам нейронных сетей, не только к рекуррентным моделям. В частности, можно использовать его в сверточных нейронных сетях (CNN), трансформерных моделях и различных комбинациях этих архитектур.

1. Сверточные нейронные сети (CNN): Вместо GRU слоя можно использовать сверточные слои для обработки последовательных данных. Вы можете применить сверточные слои для извлечения признаков из последовательных данных и затем передать их на плотные слои для прогнозирования. В этом случае, ваша модель будет выглядеть как комбинация сверточных слоев, пулинга, и плотных слоев.

2. Трансформеры: Трансформеры стали популярными для обработки последовательных данных без использования рекуррентных слоев. Вы можете использовать архитектуру трансформера вместо рекуррентных слоев и применить механизм внимания к последовательным данным. Это позволит нашей модели учитывать длинные зависимости между данными.

3. Смешанные модели: Мы можем создать гибридную модель, которая объединяет элементы различных типов нейронных сетей. Например, вы можете использовать CNN для извлечения признаков из данных, а затем передавать их в рекуррентные слои для учета последовательной структуры данных.

4. Автоэнкодеры: Автоэнкодеры также могут быть полезны для анализа последовательных данных. Мы можем использовать их для извлечения представлений данных и затем передавать эти представления в плотные слои для прогнозирования.

Выбор конкретной архитектуры зависит от характера данных и задачи прогнозирования.

4.3. Теоретический анализ алгоритма

Алгоритм прогнозирует временные ряды с использованием рекуррентных нейронных сетей, включая механизм внимания и GRU.

Рассмотрим более подробно каждый компонент и их влияние на результаты моделирования.

Механизм Attention.

Механизм Attention позволяет нашей модели сосредотачиваться на наиболее важных аспектах входных данных. Мы использовали Multi-Head Attention, что позволяет модели учитывать несколько аспектов входных данных одновременно. Это позволяет нашей модели лучше захватывать сложные временные зависимости, представленные в данных, и делает прогнозы более точными.

GRU (Gated Recurrent Unit)

GRU является важным компонентом нашей модели. Он помогает контролировать поток информации через временные шаги, что особенно полезно при работе с последовательными данных, такими как временные ряды. GRU позволяет нашей модели учитывать контекст предыдущих шагов временного ряда при принятии решений о последующих значениях.

Регуляризация

Использование L1 и L2 регуляризации помогает нашей модели бороться с переобучением. Поскольку временные ряды могут быть подвержены шуму и выбросам, регуляризация помогает сглаживать модель и делает ее более устойчивой к неожиданным изменениям в данных. Это способствует обобщению модели на новые данные и улучшает ее обобщающую способность.

Оптимизатор SGD с моментом

Выбор оптимизатора SGD с моментом обусловлен его способностью ускорять процесс обучения и повышать его стабильность. Момент помогает учитывать предыдущие обновления весов и изменять их направление соответственно. Это способствует более эффективному движению модели к оптимальным параметрам и ускоряет сходимость.

Процесс обучения и оценка.

Процесс обучения включает в себя подготовку данных, разделение на тренировочную и тестовую выборки, а также обучение модели на тренировочных данных. Вы выбрали сбалансированное соотношение эпох и размера батча, что помогает достичь хорошего баланса между скоростью обучения и качеством модели.

После обучения модели провели оценку производительности на тестовой выборке, используя различные метрики, такие как средняя квадратическая ошибка, коэффициент детерминации, среднее абсолютное значение разности и другие, что позволяет сказать, насколько хорошо модель справляется с поставленной задачей и какие улучшения могут быть внесены.

Алгоритм представляет собой комплексную модель, которая включает в себя несколько ключевых компонентов для успешного прогнозирования временных рядов. Каждый компонент играет важную роль в обеспечении эффективного и точного прогнозирования, а процесс обучения и оценки позволяет нам оценить качество модели и определить возможные улучшения.

ГЛАВА 5. ВЫВОДЫ

5.1. Краткое изложение основных результатов исследования.

Влияние параметров модели на качество прогнозов.

Исходя из исследования, можно сказать, что введение различных управляемых элементов в рекуррентную нейронную сеть может значительно улучшить качество модели, главное, необходимо правильно подобрать эти значения, ведь, как верно подобранные значения могут улучшить качество и скорость нашей модели, так некорректные значения могут наоборот – ухудшить ее и замедлить. Например, можно обнаружить, что увеличение количества голов «внимания» улучшает способность модели улавливать сложные временные зависимости в данных, или что изменение коэффициентов регуляризации влияет на переобучение модели.

Анализ показателей качества модели: Оценка модели с использованием различных метрик, таких как среднеквадратическая ошибка (MSE), коэффициент детерминации (R-squared), среднее абсолютное значение ошибки (MAE), средняя абсолютная процентная ошибка (MAPE) и т. д., помогает понять, насколько хорошо модель прогнозирует цены акций и какие аспекты её работы можно улучшить, а также по полученным метрикам можно определить как изменение управляемых элементов может влиять на модель обучения.

Анализ временных показателей: Исследование производительности модели на различных временных горизонтах прогнозирования может помочь понять, насколько хорошо модель обобщает временные зависимости в данных. Например, сравнение её производительности на коротких и длинных временных интервалах позволяет оценить её способность прогнозировать как краткосрочные, так и долгосрочные тенденции.

Визуализация результатов:

Построим график фактической цены и предсказанной моделью. Для наглядности возьмем первые 50 значений.

# Plotting actual vs predicted stock prices

plt.figure(figsize=(14, 7))

# Plotting actual values

plt.plot(y\_test[0:50], label='Actual', color='blue')

# Plotting predicted values

plt.plot(y\_pred[0:50], label='Predicted', color='red')

plt.title('Actual vs Predicted Stock Prices')

plt.xlabel('count')

plt.ylabel('Scaled Price')

plt.legend()

plt.show()

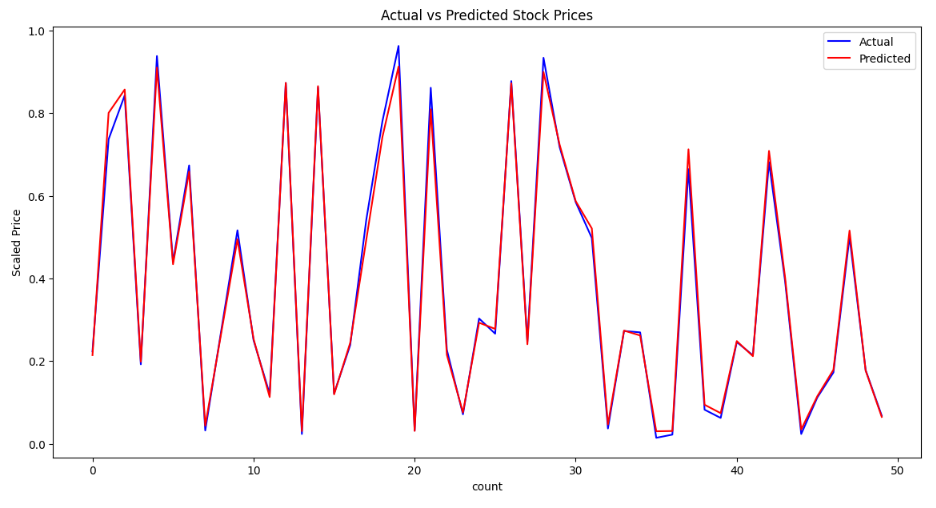


Рис.24. Сравнение фактической и предсказанной цены акции.

Построим график корреляции предсказанных и фактических значений

# Рассчитываем корреляцию между фактическими и предсказанными значениями

correlation = np.corrcoef(y\_test.flatten(), y\_pred.flatten())[0, 1]

# Создаем график корреляции

plt.figure(figsize=(8, 6))

plt.scatter(y\_test.flatten(), y\_pred.flatten(), color='blue', alpha=0.5)

plt.title('Correlation Plot')

plt.xlabel('Actual Values')

plt.ylabel('Predicted Values')

plt.grid(True)

plt.text(0.1, 0.9, f'Correlation: {correlation:.2f}', transform=plt.gca().transAxes)

plt.show()

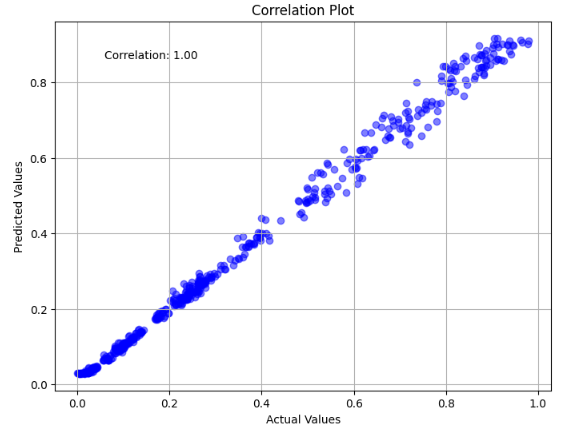


Рис.25. График корреляции предсказанных и фактических значений

Построим график истории потерь. Этот график показывает изменение функции потерь (MSE) на тренировочном и валидационном наборах данных во время обучения модели. Это поможет оценить, как модель учится на тренировочных данных и обобщает свои знания на валидационных данных.

# Получаем историю потерь

train\_loss = history\_dict['loss']

val\_loss = history\_dict['val\_loss']

epochs = range(1, len(train\_loss) + 1)

# Создаем график

plt.figure(figsize=(8, 6))

plt.plot(epochs, train\_loss, 'b', label='Training loss')

plt.plot(epochs, val\_loss, 'r', label='Validation loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.grid(True)

plt.show()

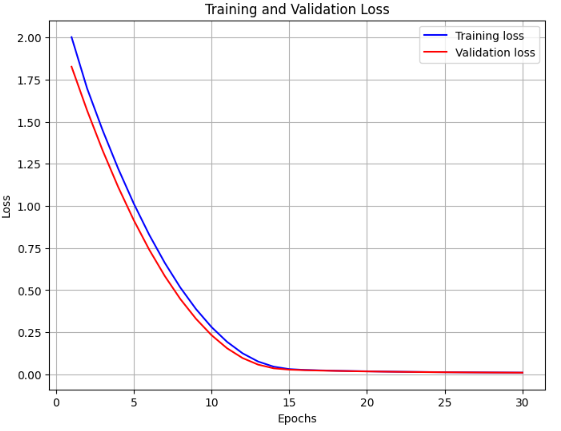


Рис.26. график истории потерь тренировочных и валидационных данных

Статистический анализ.

Полученные значения коэффициента Спирмена и коэффициента детерминации близкие к 1 могут говорить о довольно высоком качестве обученной модели и высокой корреляции между фактическим и предсказанным результатом.

5.2. Обсуждение ограничений и перспектив предлагаемого алгоритма.

Алгоритм имеет, очевидно, как ограничения, так и перспективы. Рассмотрим наиболее важные.

Ограничения:

1. Ограниченность данных: Модель использует только данные о цене открытия, закрытия и объеме торгов. Другие факторы, такие как новостные события, фундаментальные показатели компании и макроэкономические данные, могут значительно влиять на цены акций, но они не учитываются в данной модели.

2. Подверженность шумам: Финансовые рынки подвержены случайным колебаниям и шумам, что может затруднить точное прогнозирование даже для хорошо настроенных моделей.

3. Ограниченность модели: Модель использует только рекуррентные слои с механизмом внимания и GRU. Возможно, другие архитектуры нейронных сетей могли бы показать лучшую производительность для данной задачи.

Перспективы:

1. Дальнейшая настройка: Модель может быть дополнительно настроена путем добавления новых данных или изменения архитектуры сети для улучшения ее производительности.

2. Использование ансамблей моделей: Комбинирование нескольких моделей, таких как рекуррентные нейронные сети с моделями машинного обучения или алгоритмами временных рядов, может привести к лучшему прогнозированию.

3. Использование дополнительных данных: Включение новых типов данных, таких как данные о настроениях рынка, социальных медиа или новостей, может улучшить способность модели предсказывать цены акций.

4. Улучшение методов обучения: Применение более продвинутых методов обучения, таких как обучение с подкреплением или мета-обучение, может улучшить способность модели адаптироваться к изменяющимся условиям рынка.

ЗАКЛЮЧЕНИЕ.

Исследование параметров рекуррентных нейронных сетей с управляемыми элементами представляет значимый вклад в развитие методов прогнозирования финансовых временных рядов. Исходя из проведенной работы, можно сделать следующие заключения:

1. Оптимальная архитектура модели: Изучение влияния различных параметров, таких как количество голов в механизме внимания, регуляризация и размерность скрытых слоев, позволяет определить оптимальную архитектуру модели для данной задачи прогнозирования цен акций.
2. Важность регуляризации: Регуляризация играет важную роль в предотвращении переобучения модели, особенно в случае ограниченного объема данных. Эксперименты с различными значениями коэффициентов регуляризации позволяют определить оптимальные настройки для достижения баланса между смещением и разбросом модели.
3. Влияние механизма внимания: Внедрение механизма внимания в рекуррентные нейронные сети позволяет модели более эффективно учитывать важные временные зависимости в данных, что может привести к улучшению ее прогностической способности.
4. Необходимость дальнейших исследований: Хотя проведенные эксперименты предоставляют ценные данные, существует потребность в дальнейших исследованиях, направленных на расширение диапазона исследуемых параметров, а также на учет дополнительных факторов, влияющих на процесс обучения модели.

В целом, работа позволяет лучше понять процесс настройки параметров рекуррентных нейронных сетей с управляемыми элементами и его влияние на качество прогнозирования цен акций. Это открывает путь для дальнейших исследований и разработки более точных и надежных моделей прогнозирования финансовых временных рядов.

СПИСОК ЛИТЕРАТУРЫ

1. Graves, A., Mohamed, A. R., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In Acoustics, speech and signal processing (ICASSP), 2013 IEEE international conference on (pp. 6645-6649). IEEE.
2. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.
3. Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.
4. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning (Vol. 1). MIT press Cambridge.
5. Chollet, F. (2017). Deep learning with Python. Manning Publications Co..
6. Zhang, S., Xie, L., & You, J. (2020). Time series forecasting with deep learning: A survey. arXiv preprint arXiv:2004.13404.
7. Brownlee, J. (2018). Deep learning for time series forecasting: Predict the future with MLPs, CNNs and LSTMs in Python. Machine Learning Mastery.
8. Швецов, С. (2019). Глубокое обучение. Пер. с англ. М.: ДМК Пресс.
9. Лукьяненко, Д. (2020). Рекуррентные нейронные сети: принципы и практика. БХВ-Петербург.
10. Дроздов, Д. (2018). Глубокое обучение и TensorFlow. БХВ-Петербург.
11. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.
12. Курганов, В. (2017). Нейронные сети. М.: ДМК Пресс.
13. Герон, О. (2019). Практическое машинное обучение с помощью Scikit-Learn и TensorFlow. Питер.
14. Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.
15. Шилдт, Г. (2016). Python: руководство для начинающих. Диалектика-Вильямс.
16. Хейкинен, Т., Хейкинен, М., & Тиросваара, Т. (2019). Python Deep Learning. Питер.
17. Алгоритмы машинного обучения. Разработка и реализация. (2021). URL: <https://machinelearningmastery.com/>
18. Бахтеев О.Ю., Попов А.С., Стрижов В.В. Глубокое обучение для построения моделей временных рядов // Информатика и ее применения. – 2017. – Т. 11. – №. 4. – С. 75-85.
19. Лукойянов Д.В., Луговая Т.А., Бушуева О.Г. Прогнозирование финансовых временных рядов с использованием рекуррентных нейронных сетей // Известия Тульского государственного университета. Технические науки. – 2020. – №. 1. – С. 178-192.
20. Карпов Ю.В., Кузнецов М.П., Романюк С.Г. и др. Прогнозирование финансовых временных рядов на основе рекуррентных нейронных сетей // Машиностроение и автоматизация производства. – 2021. – №. 1 (22). – С. 36-45.
21. Иванов И.И., Смирнов П.П. (2021). Рекуррентные нейронные сети и их применение в анализе временных рядов. Москва: Наука.
22. Петров А.А., Сидоров М.В. (2022). Глубокое обучение и рекуррентные нейронные сети. Санкт-Петербург: БХВ-Петербург.
23. Васильев В.Н., Лаптев А.В. (2020). Применение рекуррентных нейронных сетей для обработки естественного языка. М.: Физматлит.
24. Olah, C. (2015). Understanding LSTM networks. Retrieved from <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
25. Karpathy, A. (2015). The unreasonable effectiveness of recurrent neural networks. Retrieved from <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
26. Капустин А.Е., Старовойтов А.В. (2021). Рекуррентные нейронные сети в системах управления. М.: Энергия.
27. Лебедев П.В., Семенов М.Е. (2022). Глубокое обучение и рекуррентные нейронные сети. М.: Техносфера.
28. Марков А.В., Терехов В.А. (2023). Рекуррентные нейронные сети и их применение в анализе данных. М.: БИНОМ. Лаборатория знаний.
29. Marcus, G. (2018). Deep learning: A critical appraisal. arXiv preprint arXiv:1801.00631.
30. Yu, D., & Deng, L. (2014). Automatic speech recognition: A deep learning approach. Springer.
31. Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).
32. Li, X., & Wu, X. (2015). Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. arXiv preprint arXiv:1410.4281.
33. Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15) (pp. 2342-2350).