

Федеральное государственное бюджетное образовательное учреждение
высшего образования «Московский государственный университет имени
М.В.Ломоносова»

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра Математической теории интеллектуальных систем

КУРСОВАЯ РАБОТА

TODO

Выполнил:

студент 431 группы

Зенин В. О.

Научный руководитель:

к.ф.-м.н., н.с Половников В. С.

Москва - 2023

Оглавление

Введение	4
1. Методы прогнозирования временных рядов	5
1.1. AR(p)	5
1.2. MA(q)	5
1.3. Exponential Smoothing	5
1.4. Seasonal Decomposition	5
1.5. Decision Trees	6
1.6. Neural Networks	6
Основная часть	7
1. Формальная постановка задачи	7
2. Данные	7
2.1. Предобработка	7
2.2. Формирование обучающего, валидационного и тестового множества	7
3. Методы	8
3.1. LSTM	8
3.1.1. Bidirectional LSTM	9
3.2. Transformer	10
3.2.1. Self-attention	10
3.2.2. Multi-Head Attention	11
3.2.3. Fully-Connected Layer	12
3.2.4. Positional Encoding	12
3.2.5. Encoder-only Transformer	12
4. Эксперимент	13
4.1. Параметры	13
4.1.1. LSTM	14
4.1.2. Transformer	14
4.1.3. Подбор параметров	14
4.2. Результаты	15
Список литературы	16

Введение

В настоящее время прогнозирование временных рядов является одной из наиболее актуальных задач в области анализа данных. Это связано с тем, что временные ряды могут отражать различные экономические, социальные и политические процессы, которые необходимо учитывать при принятии решений в различных сферах жизни. Существует множество методов прогнозирования временных рядов, каждый из которых имеет свои преимущества и недостатки.

Существует много временных рядов, связанных с финансами, например, цены различного рода активов. Также можно найти описание паттернов движения цены, полученные путём анализа исторических данных биржевых котировок. Многие трейдеры используют их как основание для своих стратегий. Правила, образующиеся в результате найденных закономерностей, достаточно примитивны, как и сами паттерны. Если предположить, что кем-то найдена выгодная стратегия, то подобную способны найти и многие другие участники рынка, сводя на нет любую потенциальную выгоду. Вызывает интерес: способны ли нейронные сети находить паттерны и, тем самым, определять приносящие доход стратегии торговли, скрытые от большинства трейдеров.

Информация о классических финансовых инструментах во многом скрыта и хранится на биржах. Финансовые транзакции также скрыты за межбанковским обменом и не поддаются анализу. Однако существуют набирающие популярность криптофинансовые активы, информация о которых, по своей природе, намного более открыта и может быть использована для анализа движения цены.

Цель данной работы – исследование доступной публично информации о криптовалютах, построение нескольких архитектур нейронных сетей для анализа исторических данных и построение прогноза изменения будущей цены актива на примере Bitcoin

1. Методы прогнозирования временных рядов

1.1. $AR(p)$

Одним из методов прогнозирования временных рядов является использование AR (авторегрессионных) моделей [1]. AR модели позволяют учитывать прошлые значения временного ряда для предсказания его будущих значений. Авторегрессионный процесс задается следующим образом:

$$X_t = \beta_0 + \sum_{i=1}^p \beta_i X_{t-i} + \varepsilon_t,$$

где X_t – будущее значение, которое необходимо предсказать, β_i – параметры модели, p – порядок модели, независимые одинаково распределенные случайные величины $\varepsilon_t \sim N(0, 1)$.

1.2. $MA(q)$

Также широко распространён метод скользящего среднего (Moving Average, MA). Этот метод основан на усреднении значений временного ряда за определенный период времени. Модель скользящего среднего q -го порядка определяется как

$$X_t = \sum_{i=0}^q \beta_i \varepsilon_{t-i}$$

1.3. *Exponential Smoothing*

Для усреднения временного ряда кроме скользящего среднего можно использовать экспоненциальное сглаживание. Таким образом получается следующая модель [2]:

$$S_t = \begin{cases} C_t, & t = 1 \\ C_{t-1} + \alpha \cdot (C_t - S_{t-1}) & t > 1 \end{cases},$$

где S_t – сглаженный ряд, C_t – исходный ряд, α – коэффициент сглаживания, $\alpha > 0$ и обычно не превосходит 1 или 2.

1.4. *Seasonal Decomposition*

Для рядов с выраженной сезонностью существует метод сезонной декомпозиции – он позволяет выделить сезонные компоненты из общего ряда, что даёт возможность прогнозировать поведение временного ряда в зависимости от сезонных факторов. В данном методе временной ряд делится на две составляющие: сезонную и трендовую. Сезонная составляющая представляет собой повторяющиеся колебания, связанные с сезонными факторами (например, сезонность продаж в розничной торговле). Трендовая составляющая отражает общую тенденцию развития ряда [3], [4].

Описанные ранее классические методы могут применяться как по-отдельности, так и вместе. Из последнего вытекают комбинированные модели ARMA, ARIMA, SARIMA. В качестве

параметров необходимо задать факторы, которые будет использовать модель, например: значения временного ряда из прошлого, размер окна скользящего среднего, сдвиг для сезонности (необходимо чтобы заданные факторы находились в одном и том же сезонном промежутке). Коэффициенты при заданных факторах вычисляются по методу наименьших квадратов [5].

1.5. *Decision Trees*

Хорошо зарекомендовали себя методы на основе деревьев решений. Одним из преимуществ деревьев решений является их способность обрабатывать большие объемы данных и находить сложные зависимости между признаками и целевым значением. Они также могут быть легко интерпретированы и объяснены, что делает их полезными для задач прогнозирования временных рядов. Например, если имеются данные о продажах товаров в магазине за последние несколько лет, то можно использовать деревья решений для прогнозирования будущих продаж. Мы можем определить признаки, такие как цена товара, сезонность, количество конкурентов в районе и т.д., и использовать их для создания дерева решений. Каждый узел дерева будет принимать решение на основе значения признака, и на выходе получится прогноз будущих продаж.

Для алгоритмов на основе деревьев решений часто используется градиентный бустинг. Данный подход может быть описан следующим образом: построенное дерево имеет некоторую ошибку в своих предсказаниях, при наличии дифференцируемой функции ошибки можно определить поправочные значения, уменьшающие ошибку (градиент) и затем построить новое дерево, цель которого предсказать поправочные значения. Повторяя данную процедуру множество раз строится последовательность деревьев, в которой каждое новое дерево уточняет результат всех предыдущих [6].

Деревья решений не имеют представления о временной зависимости между наблюдениями. Чтобы сообщить им эту информацию необходимо закодировать время в признаках. Например, год, месяц, день недели, информация был ли день выходным или праздником – всё это может быть частью признаков, по которым будет строиться прогноз. Однако целевой признак, например такой как цена актива или величина продаж не должны включаться. В этом основное отличие от классических моделей, которые предсказывают целевую переменную основываясь на её же значениях в исторических данных.

1.6. *Neural Networks*

Также существуют различные подходы на основе нейронных сетей, которые могут использоваться для работы с изменяющимися во времени данными. В процессе своего обучения они способны извлекать сложные нелинейные зависимости из данных и генерировать своё предсказание, основываясь на этом.

В данной работе основное внимание сосредоточено на архитектурах LSTM и Transformer.

Основная часть

1. Формальная постановка задачи

Пусть дан состоящий из T наблюдений временной ряд $X = \{x_t : x_t \in \mathbb{R}^{n+1}\}_{t=1}^T$. Выделим из x_t вектор признаков \vec{x}_t и целевое значение y_t , таким образом, что $\forall t \in \{1, \dots, T\} \exists (\vec{x}_t, y_t)$, где $\vec{x}_t \in \mathbb{R}^n$, $y_t \in \mathbb{R}$. Сформируем пары $(X_{[m;t]}, Y_t)$, где $X_{[m;t]} = \{\vec{x}_{t-m}\}_{j=t-m}^{t-1}$ – подпоследовательность фиксированной длины m , Y_t – значение целевого признака. Для задачи регрессии можно положить $Y_t = y_t$, для задачи классификации $Y_t \in \{0, \dots, K\}$, где K – количество классов, на которые можно разбить y_t . Требуется построить и оценить качество модели, принимающей на вход последовательность векторов $X_{[m;t]}$ и возвращающей значение Y_t .

2. Данные

В данной работе использованы дневные наблюдения о состоянии блокчейн сети Bitcoin с 10 мая 2020 года по 8 мая 2023 года, полученные с blockchain.com. Некоторые базовые признаки также вычислены заранее поставщиком данных. Их описания собраны в таблице 1.¹

2.1. Предобработка

При работе с ценой актива часто используется логарифм цены,

$$\ln \left(\frac{x_t}{x_{t-1}} \right)$$

позволяющий перейти от абсолютных значений к относительным. Смысл данного преобразования заключается в том, что успешная стратегия приносит доход в результате изменения цен, умноженных на вложенный капитал и именно доход имеет ключевое значение.

Входные данные для нейронных сетей следует скалировать. Однако некоторые признаки в наших данных имеют количественную природу, что выражается в почти линейном росте. Например, абсолютное значение добытых на момент времени t монет BTC. Большой смысл имеет изменение в добыче, так как оно потенциально способно дать сигнал о будущих движениях цены. Поэтому в нашем случае подобное преобразование уместно применить ко всем признакам.

До логарифмирования имелось 1094 векторов, размерности 27 каждый. В результате преобразования наблюдение за первый день вырождается и остается 1093 вектора значений.

2.2. Формирование обучающего, валидационного и тестового множества

Для обучения использовались значения до 15 июня 2022 года. Для валидации – с 15 июня 2022 года по 20 января 2023 года. Для теста – с 20 января 2023 года по 8 мая 2023 года. Данные временные диапазоны выбраны чтобы обеспечить соотношение 70:20:10.

¹Признаки, отмеченные (*) имеют не более 3 пропущенных значений, которые восстановлены линейной интерполяцией.

Целевой признак – market-price.

Сформируем из данных пары $(X_{[m;t]}, Y_t)$. Y_t – значение целевого признака. Пусть y_t – логарифм цены, выделенный из исходного набора данных и преобразованный в разность логарифмов подряд идущих значений. Тогда становится удобно построить на его основе Y_t для задачи бинарной классификации:

$$Y_t = \begin{cases} 1, y_t > 0 \\ 0, y_t \leq 0 \end{cases}$$

$X_{[m;t]}$ подаётся на вход модели, предсказание которой сравнивается с Y_t используя разумную для решаемой задачи функцию потерь. В нашем случае это бинарная кросс-энтропия.

3. Методы

3.1. LSTM

LSTM (Long Short-Term Memory) – это тип рекуррентной нейронной сети, который используется для обработки последовательных данных, таких как текст, речь и временные ряды. LSTM-сети состоят из ячеек памяти, схема которой представлена на рисунке 1, которые хранят информацию о предыдущих значениях входных данных. Каждая ячейка имеет несколько состояний, которые изменяются в зависимости от входных данных и предыдущих состояний. Принцип работы LSTM состоит в том, чтобы сохранять информацию о предыдущем состоянии ячейки и использовать эту информацию для принятия решения о текущем состоянии ячейки. Это позволяет LSTM-сетям обрабатывать длинные последовательности данных и учитывать контекст [8].

Входной блок принимает на вход данные из предыдущей ячейки и передает их в блок памяти и выходной блок. Блок памяти хранит информацию о предыдущих данных и может использовать эту информацию для прогнозирования следующего значения. Выходной блок вычисляет прогнозное значение на основе информации из блока памяти и входного блока.

Формулы для вычисления значений в LSTM ячейке могут быть следующими:

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ \tilde{C}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\ C_t &= f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \\ h_t &= o_t \cdot \tanh(C_t) \end{aligned}$$

где i_t – вес входного сигнала, f_t – вес забытого сигнала, o_t – вес выходного сигнала, c_t – значение ячейки памяти, h_t – прогнозируемое значение. W – матрица весов ячейки, b вектор сдвига. W в формулах являются различными частями этой матрицы, конкатенация векторов $[\cdot, \cdot]$ позволяет оптимизировать вычисления.

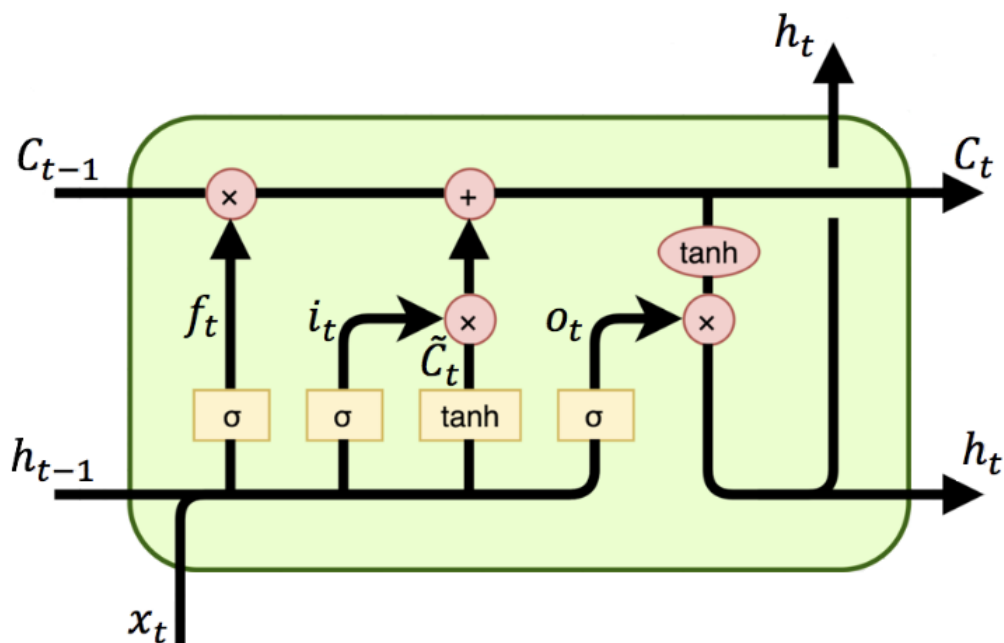


Рис. 1: Схема LSTM-ячейки

3.1.1 Bidirectional LSTM

Bidirectional LSTM - это разновидность рекуррентной нейронной сети, которая также используется для обработки последовательностей данных. Принцип работы этой сети основан на использовании двух LSTM-слоев: прямого и обратного. Прямой LSTM слой обрабатывает последовательность с начала до конца, а обратный LSTM слой - в обратном порядке. Затем результаты от обоих слоев объединяются, чтобы получить более точную оценку для каждого элемента последовательности [9].

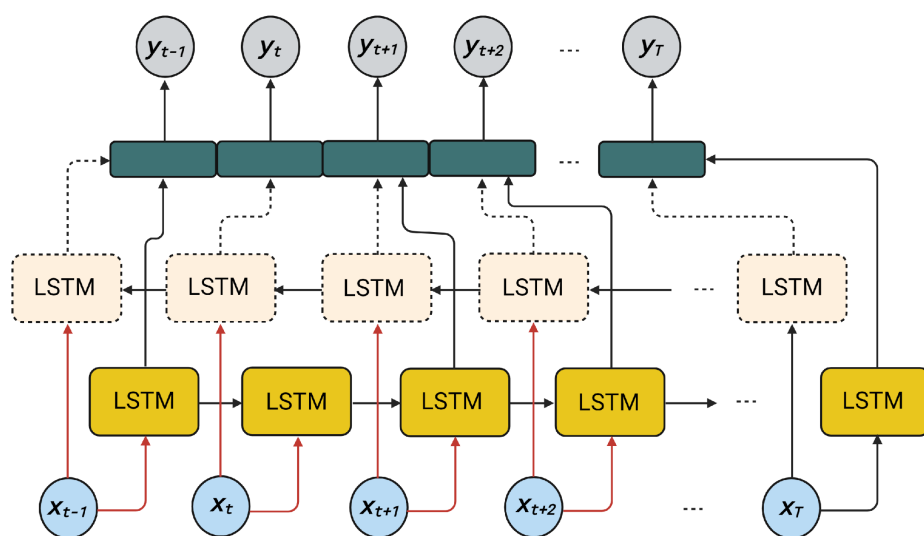


Рис. 2: Схема двунаправленной рекуррентной сети на основе ячейки LSTM

3.2. Transformer

3.2.1 Self-attention

Модель извлекает из входных данных информацию при помощи механизма внутреннего внимания (self-attention) и использует ее для формирования выходных данных.

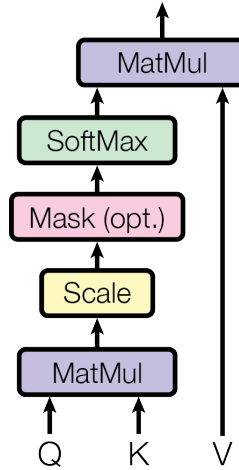


Рис. 3: Scaled Dot-Product Attention [10]

Пусть в качестве функции внимания используется скалярное произведение

$$\vec{q} \cdot \vec{k} = \sum_{i=1}^{d_k} q_i k_i$$

Обозначим за $Q = K = V$ матрицы, в строках которых записаны векторы входной последовательности, каждый вектор имеет размерность d_k . Тогда принцип работы self-attention можно описать следующим образом:

1. Создание матрицы внимания: Матрица внимания представляет собой квадратную матрицу размерности N , где N - количество элементов входной последовательности. Для её создания необходимо вычислить скалярное произведение между каждым элементом входного вектора и всеми остальными элементами

$$QK^T$$

2. Нормализация: Сумма всех значений в каждом столбце матрицы внимания должна быть равна 1, чтобы сохранить сумму всех значений в данных неизменной. Это достигается путем нормализации матрицы внимания с использованием softmax функции. Также перед применением softmax имеет смысл поделить все значения матрицы на $\sqrt{d_k}$. Это помогает компенсировать негативный эффект от проклятия размерности.

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

3. Умножение на веса: Каждый элемент матрицы представляет собой вес, в некоторой мере отражающий схожесть между векторами входной последовательности. Теперь необходимо вычислить взвешенную сумму

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$$

3.2.2 Multi-Head Attention

Возможности данной операции расширяются использованием механизма множественного внимания, который, в свою очередь, позволяет модели применить внутреннее внимание несколько раз на разные части входной последовательности, тем самым получая возможность извлечь больше полезной информации. Реализуется это следующим образом:

1. Входные данные проецируются в векторы меньшей размерности независимыми линейными слоями.
2. Каждый полученный вектор проходит через свой self-attention. Данная часть называется ‘головой’ (head).
3. Результирующие векторы объединяются путём конкатенации.

В общем случае размерности векторов после проекции должны быть равными, то есть начальная размерность вектора признаков должна быть кратна количеству голов h .

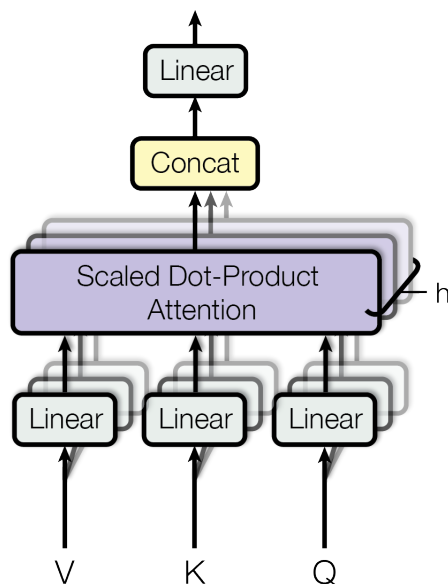


Рис. 4: Multi-Head attention [10]

3.2.3 Fully-Connected Layer

Полученный после прохождения блока множественного внимания вектор поступает на полносвязный слой размерности d_{hid} с функцией активации $ReLU$. После чего процедура может повторяться многократно, образуя слои энкодера. Результирующий вектор может быть использован для решения различных задач.

3.2.4 Positional Encoding

Описанная архитектура никак не использует информацию о последовательности подаваемых на вход векторов. В таких задачах как обработка текста или временного ряда необходимо сообщить информацию о позиционной или временной зависимости в данных. Это можно сделать используя следующее преобразование

$$PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos, 2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

d_{model} – размерность входных данных, с которой работает трансформер. Часто чтобы обеспечить её достаточно отобразить линейным слоем настоящие векторы и получить их эмбединги соответствующей размерности. Тогда компоненты $PE_{(pos)}$ прибавляются к компонентам входных векторов x_{pos} .

3.2.5 Encoder-only Transformer

Encoder-only Transformer (ЕОТ) - это разновидность Transformer, который использует только блок энкодера для генерации выходных данных. Поскольку модель не имеет декодера, ее задача заключается только в преобразовании входных данных в выходные, а не в авторегрессионном генерировании выходной последовательности.

4. Эксперимент

Решая задачу бинарной классификации в данной задаче удобно использовать “жесткий” подход. Пусть выход модели это двумерный вектор, значения которого преобразованы при помощи softmax:

Тогда каждая компонента говорит о некоторой доле вероятности принадлежности к классу, равному индексу этой компоненты. Данное представление более универсально и позволяет использовать обычную кросс-энтропию, вместо бинарной и не подбирать пороговое значение, которое будет отделять позитивный класс (1, цена пойдёт вверх) от негативного (0, цена пойдёт вниз).

Для LSTM не составляет труда получить такое предсказание: можно взять выход последней ячейки, сравнить с ground truth и сделать шаг в изменении весов. Transformer же вернёт последовательность векторов, равную по длине входной последовательности. Для приведения такого выхода к нужному вектору использовался следующий подход:

1. Значения векторов последовательности усредняются, в результате чего получается один вектор размерности `d_model`.
2. При помощи линейного слоя он отображается в вектор размерности 2
3. К нему применяется softmax и вычисляется кросс-энтропия

4.1. Параметры

В качестве гиперпараметров, не связанных с архитектурными особенностями моделей были взяты:

- **batch_size**: количество элементов в пакете, который используется для обучения нейронной сети. При обучении с использованием градиентного спуска каждый пакет используется для вычисления градиента ошибки и обновления весов сети. Чем больше размер пакета, тем больше данных используется для вычисления градиента, что может ускорить процесс обучения. Однако, слишком большой размер пакета может привести к переобучению сети и ухудшению ее производительности. Поэтому, выбор оптимального размера пакета является важной задачей при обучении нейронных сетей.
- **max_epoch**: количество проходов всех обучающих данных через модель. Каждая эпоха состоит из обучения на всех данных, а затем проверки производительности модели на тестовом наборе. Количество эпох обычно определяется размером набора данных: чем больше данных, тем больше эпох может потребоваться для обучения.
- **sequence_length**: количество элементов в последовательности, которая используется для обучения нейронной сети. Длина последовательности может влиять на производительность нейронной сети, так как она определяет количество информации, которую сеть может использовать для обучения и прогнозирования. Чем длиннее последовательность, тем больше информации можно использовать для обучения. Однако, слишком

длинная последовательность также может привести к переобучению, что может ухудшить производительность модели.

4.1.1 LSTM

- **num_layers**: количество блоков памяти в каждом слое модели LSTM, а также количество слоев в целом. Чем больше число слоев, тем больше информации может быть сохранено в блоках памяти и использовано при прогнозировании. Однако увеличение числа слоев может привести к увеличению сложности модели и времени обучения.
- **d_hid**: размерность скрытого слоя, т.е. размерность вектора, который представляет состояние ячейки LSTM. Размерность скрытого слоя оказывает значительное влияние на производительность и качество модели. Слишком маленький размер скрытого слоя может привести к недостаточной емкости для хранения информации о предыдущих состояниях ячейки, что может привести к потере информации при обучении модели. Слишком большой размер скрытого слоя также может привести к переобучению модели и снижению ее производительности.
- **dropout**: доля случайно отбрасываемых нейронов в соответствующем методе регуляризации, который используется для предотвращения переобучения.

Поскольку используется двунаправленная сеть, то **d_hid** на самом деле умножается на 2: одна половина используется для прямого прохода, другая для обратного.

4.1.2 Transformer

- **num_layers**: количество слоёв, содержащих блок множественного внимания и полносвязный слой с функцией активации ReLU.
- **d_hid**: размерность скрытого слоя, следующего после множественного внимания.
- **d_model**: размерность входных данных, с которой работает Transformer. Она может быть не равна количеству признаков в используемом датасете, поэтому требует линейного преобразования.
- **n_head**: число голов блока множественного внимания.
- **dropout**: используется после скрытого слоя.

4.1.3 Подбор параметров

LSTM из-за своих архитектурных особенностей позволяет выбрать любые параметры независимо. Transformer же требует, чтобы, например, **n_head** делило **d_model**. Поэтому для перебора были выбраны различные степени двойки, чтобы обеспечить делимость для любого набора выбранных параметров. Диапазоны и финальные параметры, на которых выполнялось тестирование, описаны в таблице 3 для LSTM и в таблице 4 для Transformer.

Параметры перебирались при помощи фреймворка *optuna* в заданных диапазонах. В качестве метрики для сравнения моделей с различными параметрами использовалась точность (Accuracy). Заведомо неудачные наборы параметров отсекались при помощи MedianPruner: данный метод прерывает обучение, если модель с выбранными параметрами на данном шаге хуже чем медианное значение ранее обученных моделей на этом же шаге.

Для LSTM было взято 400 попыток перебора, для Transformer сначала 400, а затем ещё 200 для более легкой архитектуры.

Наиболее перспективные результаты для LSTM получались при использовании малых размерностей скрытого слоя и большего числа слоёв LSTM, из чего был сделан вывод что стоит выбрать параметры, указанные в таблице 3, но увеличить продолжительность обучения до 500 эпох, поскольку значение функции ошибки на валидации указывало на устойчивость к переобучению.

Transformer давал плохие результаты на первых 400 попытках, говорящие о явном переобучении. Стоило рассмотреть архитектуру с малым числом параметров и повторить перебор. В результате перспективными были определены параметры, указанные как итоговые в таблице 4. Также значения функции ошибки на валидации указывали на то, что удалось добиться устойчивости к переобучению и отказаться от метода раннего останова.

С лучшими параметрами на валидации модели были обучены на всех данных, кроме тестовых.

4.2. Результаты

Значения на тестовом множестве для моделей перечислены в таблице 3.

Список литературы

- [1] James D. Hamilton. Time Series Analysis and Forecasting. Princeton University Press, 1994.
- [2] Everette S. Gardner. Exponential smoothing: The state of the art. 01 Jan 1985-Journal of Forecasting (John Wiley & Sons, Ltd.)-Vol. 4, Iss: 1, pp 1-28
- [3] Lovell, Michael C. Seasonal Adjustment of Economic Time Series and Multiple Regression Analysis. Journal of the American Statistical Association, vol. 58, no. 304, 1963, pp. 993-1010.
- [4] Robert Alan Yaffee, Monnie McGee. Introduction to Time Series Analysis and Forecasting: With Applications of SAS and SPSS. Academic press, 2000
- [5] Hyndman, R.J., & Athanasopoulos, G. (2021) Forecasting: principles and practice, 3rd edition, OTexts: Melbourne, Australia.
- [6] Panarese, A.; Settanni, G.; Vitti, V.; Galiano, A. Developing and Preliminary Testing of a Machine Learning-Based Platform for Sales Forecasting Using a Gradient Boosting Approach. Appl. Sci. 2022, 12, 11054.
- [7] U Thissen, R van Brakel, A.P de Weijer, W.J Melssen, L.M.C Buydens. Using support vector machines for time series prediction. Chemometrics and Intelligent Laboratory Systems, Vol. 69, Iss: 1-2, 2003, pp 35-49.
- [8] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. Neural Comput. 9, 8 (November 15, 1997), 1735-1780.
- [9] Schuster, Mike & Paliwal, Kuldeep. (1997). Bidirectional recurrent neural networks. Signal Processing, IEEE Transactions on. 45. 2673 - 2681.
- [10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In NeurIPS, 2017.

Таблица 1: Признаки из блокчейн сети Bitcoin

Признак	Описание
total-bitcoins (*)	Количество добытых монет
market-price	Средняя цена в USD на крупнейших обменниках
trade-volume	Объем обменянных BTC (USD)
blocks-size	Размер сети блокчейна (MB)
avg-block-size	Средний размер блока (MB)
n-transactions-total	Количество транзакций
n-transactions-per-block	Среднее число транзакций на блок
n-payments-per-block	Среднее число наград за валидированный блок
median-confirmation-time	Медианное время, за которое обработанная транзакция добавляется к сети
avg-confirmation-time	Среднее время, за которое обработанная транзакция добавляется к сети
hash-rate	Мощность сети
difficulty	Относительная мера сложности сети – насколько трудно валидировать очередной блок
transaction-fees	Выплаченные BTC за валидацию блоков
transaction-fees-usd	Выплаченные USD за валидацию блоков
fees-usd-per-transaction	Средняя выплата в USD за валидированную транзакцию
cost-per-transaction	Общий доход майнеров, разделённый на количество транзакций
n-unique-addresses (*)	Количество уникальных адресов, используемых в сети
n-transactions	Количество подтверждённых транзакций за день
n-payments	Количество подтверждённых выплат за день
output-volume	Объем транзакций за день
mempool-count	Количество неподтверждённых транзакций
mempool-growth	Рост хранилища неподтверждённых транзакций
mempool-size	Размер хранилища неподтверждённых транзакций
utxo-count (*)	Количество монет, доступных для использования в качестве оплаты за транзакции
n-transactions-excluding-popular	Количество транзакций, за исключением 100 самых популярных адресов
estimated-transaction-volume (*)	Оценочная стоимость транзакций (BTC)
estimated-transaction-volume-usd (*)	Оценочная стоимость транзакций (USD)

Таблица 2: Диапазоны перебора параметров

Таблица 3: LSTM			Таблица 4: Transformer		
Параметр	Диапазон	Итог	Параметр	Диапазон	Итог
batch_size	[16, 128]	32	batch_size	$2^{[4,7]}$	
max_epoch	[5, 2000]	500	max_epoch	[5, 4000]	
sequence_length	[5, 96]	40	sequence_length	[5, 96]	
num_layers	[2, 12]	6	num_layers	$2^{[2,6]}$	
d_hid	[16, 256]	28	dropout	[0.2, 0.5]	
dropout	[0.2, 0.5]	0.27	d_hid	$2^{[1,10]}$	
			d_model	$2^{[3,9]}$	
			n_head	$2^{[0,3]}$	

Таблица 3: Качество на тестовом множестве

Метрика	LSTM	Transformer
Accuracy		
Precision		
Recall		
F1-Score		