Курсовая работа защищена с оценкой _

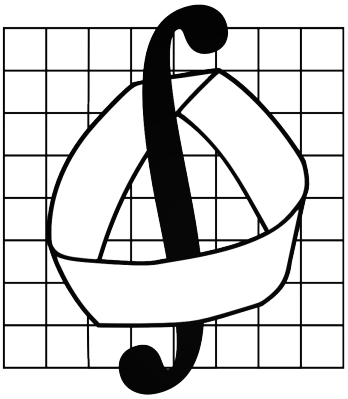
Ученый секретарь кафедры Теории вероятностей

доцент Илларионов Е. А.

Московский государственный университет имени М.В. Ломоносова

Механико-математический факультет

Кафедра Теории вероятностей



КУРСОВАЯ РАБОТА

Прогнозирование финансовых временных рядов с помощью нейронных сетей с памятью

Выполнена студентом 409 группы

Зюзиным Владиславом Андреевичем

Научный руководитель:

д.ф.-м.н. М.И. Кумсков

Москва, 2024

Аннотация

Важной задачей в финансовой области является прогнозирование временных рядов, таких, например, как цены на акции, курсы валют, стоимости деривативов и др. Существует множество методов, с помощью которых мы можем это сделать. Однако в последнее время наибольшую популярность приобрели модели на основе нейронных сетей. В данной работе рассмотрен особый вид нейронных сетей – нейронные сети с памятью: LSTM и Transformer. Эти модели являются достаточно эффективными по точности прогнозов, так как способны выявлять нелинейные закономерности в данных, но их основной недостаток – значительные временные затраты при обучении. Основная цель исследования заключается в использовании методов оптимизации вычислений, рассмотрению вопросов, связанных с возможностью параллельных вычислений. В результате работы будет получен вывод об эффективности предложенных подходов.

Содержание

Введение

Прогнозирование финансовых временных рядов является одной из ключевых задач в области финансов. Исходя из возможных сценариев поведения тренда, принимаются решения об инвестировании в тот или иной вид активов. В связи с этим возникает естественный вопрос об эффективных методов прогнозирования, которые могут учитывать сложные взаимосвязи между переменными, с наименьшими временными затратами на вычисления.

Одним из перспективных подходов к прогнозированию финансовых временных рядов является использование нейронных сетей с памятью, таких как LSTM (Long Short-Term Memory) и Transformer. Эти сети обладают способностью запоминать предыдущие состояния и учитывать долгосрочные зависимости между переменными, что делает их особенно подходящими для анализа временных рядов. Однако время на обучение на основе большого массива данных достаточно велико. Поэтому возникает вопрос о параллельных вычислениях в процессе построения модели.

Целью данной курсовой работы является изучение возможностей применения методов параллельных вычислений при обучении нейронных сетей LSTM и Transformer для прогнозирования финансовых временных рядов, а также проведение сравнительного анализа эффективности этих моделей.

В ходе работы будут рассмотрены архитектуры LSTM и Transformer, проведены эксперименты по обучению и тестированию моделей на реальных финансовых данных, оценены их временные характеристики.

В главе 1 ставится основная задача исследования, даётся определение ключевому понятию, такому как временной ряд, приводятся важные

свойства финансовых временных рядов, описываются классические методы прогнозирования. Особенное внимание уделяется ARIMA – базового метода прогнозирования финансовых временных рядов, ее модификации SARIMA, а также моделям на основе цепей Маркова и вейвлет-анализу. Кратко вводится терминология, связанная с моделями на основе нейронных сетей, приводится пример самой простой NN – нейронной сети с прямой связью.

В главе 2 подробно описывается архитектура LSTM, а также приводятся методы оптимизации вычислений в ней: потоковые матричные операции, слияние точечных операций и пр.

Глава 3 посвящена моделям нейронных сетей Transformer. В ней приводится архитектура Transformer с поэтапным алгоритмом вычислений в ней, что позволяет выявить этапы, на которых вычисления могут производится параллельно. Далее даются краткие описания методов оптимизаций, связанных с GPU.

Глава 4 представляет собой краткий обзор языка Python, библиотек, необходимых для визуализации данных, построения нейронных сетей, например, Tensorflow, а также способам параллельных вычислений в облачной среде Google Colab.

В главе 5 приведены результаты вычислительных экспериментов, сравнивается время на прямое обучение моделей и с оптимизациями. Делаются выводы о качестве прогнозирования.

Таким образом, в конце приходим к выводу о значительной эффективности применения данных методов в силу их точности и возможности быстрых параллельных вычислений.

Глава 1. Постановка задачи. Основные определения

1.1 Постановка задачи

Рассмотрим временной ряд u_t, например, могут быть значения цен на акции определенной компании в момент t – тик, $0 \leq t \leq n$. Причем мы знаем его значения до k-го момента ($0 \leq k \leq n$). Основной целью этой курсовой работы будет построение прогноза для следующих моментов времени при использовании нейронных сетей, таких как LSTM и Transformer. Фокус, на котором будет сконцентрировано основное внимание, составляют методы оптимизации на основе параллельных вычислений в этих моделях. Чтобы эти модели были эффективны в применении, т.е. достаточно быстро строилась модель, прогноз был довольно точным, следует уделять внимании и скорости вычислений.

В рамках данной работы ставятся следующие подзадачи:

- Дать определение ключевым понятиям исследования. Выделить основные особенности финансовых временных рядов;
- Провести краткий обзор классических методов при прогнозировании, а также моделей нейронных сетей, которые могут быть использованы для построения сценария изменения цен на акции;
- Подробно описать механизм работы сетей с долгой краткосрочной памятью (далее – LSTM). Основе его выделить свойства модели, которые помогут при оптимизации вычислений;

- Также как и в предыдущем пункте следует провести анализ модели Transformer;
- Изучить особенности облачных и параллельных вычислений в Google Colab (язык Python);
- Выбрать датасеты, провести на них вычислительные эксперименты;
- Сделать выводы о качестве построенных моделей, их эффективности по времени.

Этим вопросам и будет посвящена данная работа.

1.2 Ключевые определения. Методы анализа временных рядов

Начнем с базовых понятий нашего исследования.

Определение. *Временной ряд* – это набор наблюдений u_t, каждое из которых фиксируется в определенный момент времени $t \in \mathcal{T}$ [5]. *Временной ряд с дискретным временем* – это такой ряд, в котором набор $\mathcal{T}$ моментов времени, в которые производятся наблюдения, является дискретным набором, как в случае, например, когда наблюдения производятся через фиксированные интервалы времени. *Непрерывные временные ряды* получаются, когда наблюдения фиксируются непрерывно в течение некоторого интервала времени.

Временными рядами могут быть любые данные индексированные временем: цены на акции, индексы, солнечная активность в течении трехсот дет, почасовой уровень воды в реке за день – они встречаются прак-

тически повсюду: в инженерном деле, физике, социологии, экономике и других науках.

Финансовые временные ряды – основной объект данной работы. В чем их особенность?

- В большинстве случаев рыночные цены *нестационарны*, т.е. математическое ожидание и дисперсия непостоянны. Вследствие этого при использовании моделей прогнозирования, предполагающих условия стационарности процесса, к примеру, модель $ARIMA$, необходимо привести их к такому виду, в частности, взятием разности.

- Финансовые временные ряды *персистентны*: последующие данные сильно зависят от предыдущих. Показатель Херста один из индикаторов персистентности ряда; если этот индекс принадлежит интервалу $[0,5,1]$, то ряд является таковым. Их можно считать «процессами с долговременной памятью».

- Довольно часто эти ряды имеют большой объем: торги ведутся ежесекундно, а в сети можно получить данные даже с 1970-х годов. Поэтому эффективно к ним применять методы машинного обучения, что активно и делается, как при анализе, так и при алгоритмической торговле.

- Можно считать, что финансовые временные ряды *недифференцируемы*, но при этом непрерывны. Взглянув на кривую биржевых котировок, можно увидеть, что она не является гладкой, т.е. невозможно провести к ней касательную и посчитать производную в любой точке, поэтому к ним метод Фурье не применим, но сгладив ис-

ходный ряд, например, скользящим средним, можем использовать и этот инструмент анализа.

Существует два принципиально разных класса методов анализа временных рядов: методы *частотной* (в частности, спектральный и вейвлет анализ) и *временной* области (автокорреляция). Более того, методы могут быть *параметрическими*, это означает, что мы предполагаем определенную структуру, лежащую в основе стационарного случайного процесса и описанную с использованием небольшого числа параметров, основная задача в этом случае оценка этих параметров, и *непараметрическими* – явно оценивается ковариация или спектр процесса, не предполагая, что процесс имеет какую-либо конкретную структуру.

Следует также выделить основные компоненты временных рядов, которые характеризуют закономерности и поведение данных с течением времени. Анализ этих компонент помогает понять динамику временных рядов и создать более точные модели. *Тренд* показывает общее направление численного изменения (могут выявить общий рост или спад) данных: увеличиваются ли они, уменьшаются или остаются неизменными в течение длительного периода времени. *Сезонность* – явление, которое повторяется регулярно, например, ежегодные всплески розничной торговли во время сезона отпусков. Сезонные компоненты демонстрируют колебания, фиксированные по времени, направлению и величине. *Циклы* демонстрируют колебания, которые не имеют фиксированного периода, например, экономический подъем или рецессия. *Шум* охватывает остаточную изменчивость данных, которую другие компоненты не могут объяснить. Шум включает непредсказуемые, неустойчивые отклонения после учета тенденций, сезонности и циклов.

Возникает вопрос: какие техники применимы к анализу финансовых временных рядов?

- **Скользящее среднее** (или англ. **Moving Average**). Помогает сглаживать долгосрочные тренды, достаточно устраняет шум, а главное – определяет общее направление движения временного ряда.

- **Экспоненциальное сглаживание**. Данный подход эффективен для одномерных данных с систематическим трендом или сезонной составляющей, т.к. позволяет проводить более динамичные корректировки.

- **Авторегрессия**. Самый простой и широко используемый метод: на основе прошлых наблюдений строится уравнение регрессии для прогнозирования будущих значений.

- **Декомпозиция**. Временной ряд разбивается на его основные компоненты — тренд, сезонность и остатки — для улучшения понимания и точности прогноза.

- **Кластеризация временных рядов**. Неконтролируемый метод категоризации точек данных на основе сходства, помогающий идентифицировать архетипы или тенденции в последовательных данных. Для этого подходят и модели нейронных сетей, например, LSTM, о которой будет упомянуто ниже.

- **Вейвлет-анализ**. Эффективен для анализа нестационарных данных временных рядов. Помогает выявлять закономерности в различных масштабах или разрешениях.

Интересное направление дальнейшого развития методов анализа рядов – гибридные модели, которые стратегически сочетают в себе множество методов, таких как ARIMA, экспоненциальное сглаживание, сети LSTM с глубоким обучением, чтобы извлечь выгоду из преимуществ каждого метода, а также применение более продвинутых моделей нейронных сетей: сверточных или Transformer.

1.3 Классические методы прогнозирования временных рядов

Более подробно опишем некоторые из классических методов, упомянутых в предыдущем разделе.

1.3.1 Модель ARIMA

ARIMA (AutoRegressive Integrated Moving Average) – это математическая модель для анализа временных рядов, объединяющая в себе интегрированную авторегрессию, скользящее среднее и возможность учета дополнительных внешних факторов [5].

Авторегрессия порядка p ($AR(p)$) имеет следующий вид:

$$u_t = \sum_{i=1}^{p} u_{t-i} \cdot \alpha_i + C + \varepsilon_t, \tag{1}$$

где ε_t – ошибка модели, α_i – неизвестные параметры, C – константа, α_i и C могут быть найдены из метода наименьших квадратов или метода максимального правдоподобия.

Другая компонента этой модели – *модель скользящего среднего* $MA(q)$, q – порядок. Она описывается уравнением:

$$u_t = \mu + \sum_{i=1}^{q} \beta_i \varepsilon_{t-i}, \tag{2}$$

здесь μ – математическое ожидание процесса, $\varepsilon_i \sim \mathcal{N}\left(0, \sigma^2\right)$ – гауссовский белый шум.

Для достижения большей гибкости в подгонке модели часто целесообразно объединить в одной модели авторегрессию (1) и скользящее среднее (2), она называется $ARMA(p, q)$:

$$u_t = C + \sum_{i=1}^{q} \beta_i \varepsilon_{t-i} + \sum_{i=1}^{p} u_{t-i} \cdot \alpha_i, \tag{3}$$

где C находится из МНК.

Однако для моделей $AR(p)$ необходимо предполагать условие стационарности ряда. Поэтому модель $ARMA(p, q)$ необходимо расширить для нестационарных рядов. Если в качестве входных данных используются не сами значения временного ряда, а их разность d-того порядка (ряд становится стационарным), то модель носит $ARIMA(p, d, q)$. Модель $ARIMA(p, d, q)$ описывается уравнением

$$\Delta^d u_t = C + \sum_{i=1}^{p} \alpha_i \Delta^d u_{t-i} + \sum_{i=1}^{q} \beta_i \varepsilon_{t-i}. \tag{4}$$

Здесь $\Delta^d u_t$ – последовательное взятие d раз разностей первого порядка (сначала от временного ряда, затем от полученных разностей первого порядка, затем от второго порядка и т. д.).

К преимуществам данной модели можно отнести высокое качество интерпретируемости полученных результатов, что необходимо для анализа поведения временного ряда, вывода закономерностей, а также небольшие вычислительные затраты на построение модели.

1.3.2 SARIMA

Предыдущая модель может быть применена к несезонными данными. Однако модели ARIMA также способны моделировать широкий спектр сезонных данных. Сезонная модель ARIMA формируется путем включения дополнительных сезонных терминов в модели ARIMA, такая модель называется SARIMA [32]. Записывается следующим образом:

$$ARIMA \qquad \underbrace{(p,d,q)}_{non-seasonal\ part} \qquad \underbrace{(P,D,Q)_m}_{seasonal\ part},$$

где m – количество наблюдений. Сезонная часть модели состоит из компонент, которые аналогичны несезонным компонентам модели, но включают обратные сдвиги сезонного периода.

1.3.3 Модели на основе цепей Маркова

Очень часто возникают ситуации, когда одна компонента временного ряда может переходить в другие и обратно. Вследствие чего для прогнозирования временных рядов можно использовать модели основанные на цепях Маркова. Такие модели способны уловить такие закономерности путем оценки матрицы вероятности перехода, которая позволяет рассмотреть все взаимосвязанные компоненты временного ряда одновременно.

Формализуем механизм прогнозирования временных рядов на основе цепей Маркова. Мы предполагаем, что будущее значение зависит лишь от текущего, а не от прошлых состояний. Пусть у нас есть состояния $s_1, s_2, ..., s_n$ временного ряда. Запишем вероятности перехода из одного состояния s_i в s_j:

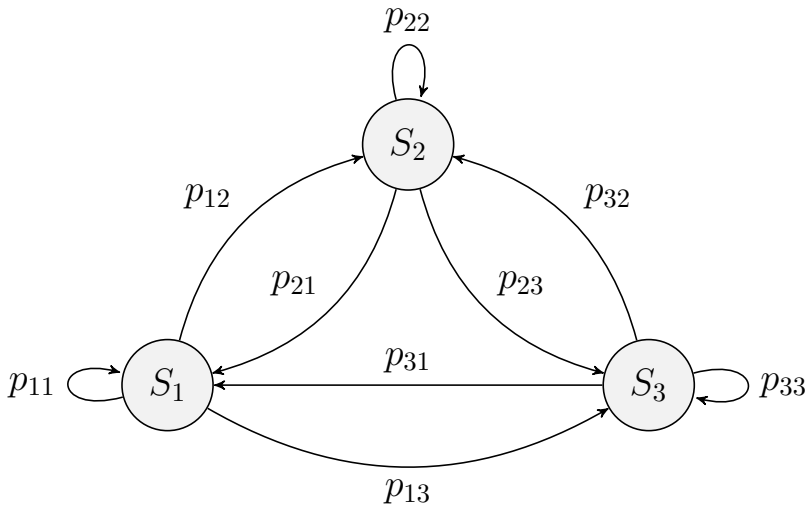
$$p_{ij} = \mathsf{P}(X_{t+1} = s_j | X_t = s_i).$$

Таким образом, получим матрицу переходных вероятностей:

$$\begin{pmatrix} p_{11} & p_{12} & p_{13} & \dots & p_{1n} \\ p_{21} & p_{22} & p_{23} & \dots & p_{2n} \\ p_{31} & p_{32} & p_{33} & \dots & p_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & p_{n3} & \dots & p_{nn} \end{pmatrix}$$

Приведем пример. Пусть у нас всего три состояния: s_1 – цены на акции пошли вниз, s_2 – цены не изменились, s_3 – цены выросли. На Рис. 1 изображена цепь Маркова для этого случая.

Рис. 1: Пример марковской цепи из трех состояний



Выбор количества состояний отдельный вопрос, который может быть решен с помощью алгоритма Витерби [26].

В целом, модель прогнозирования с помощью цепей Маркова гибка в плане учета различных структур. Однако эти модели имеют ряд существенных недостатков. Цепи Маркова требуют стационарности и марковости данных, что в большинстве случаев неверно для финансовых временных рядов. Также они чувствительны к отклонениям и шумам в

данных. Часто эти причины делают данный метод не эффективным в нашей задаче.

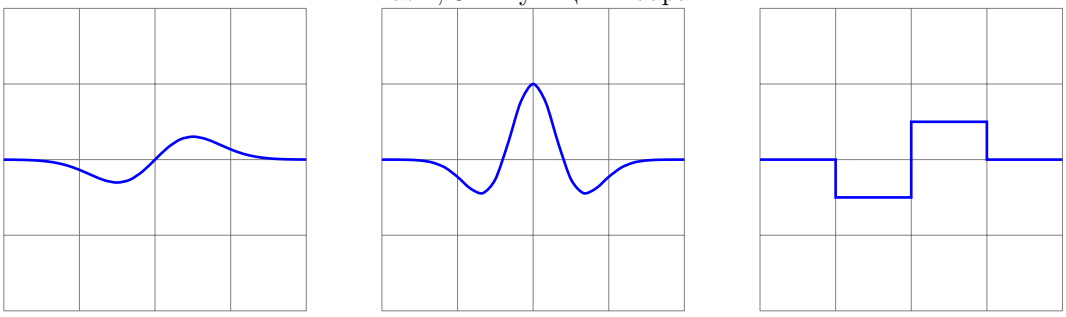
1.3.4 Вейвлет-анализ

Другим серьезным инструментом анализа временных рядов является вейвлет-анализ. Это сравнительно новый подход, его название происходит от англ. "wavelet" , что означает "маленькая волна".

Кратко опишем данный метод. Вещественная функция $\psi(\cdot)$, определенная на всем $\mathbb{R}$, называется *вейвлетом*, если

- она нормирована (в норме L_2): $\int_{-\infty}^{\infty} \psi^2(u)du = 1$,
- $\int_{-\infty}^{\infty} \psi(u)du = 0$ (технически нужно «условие допустимости»).

Рис. 2: Примеры вейвлетов: 1 – производная плотности нормального распределения, 2 – "Mexican hat" , 3 – Функция Хаара



Условия из определения действительно дают "маленькие волны" , т.к. из первого условия получаем, что $\forall\ \varepsilon > 0$, конечного T:

$$\int_{-\infty}^{-T} \psi^2(u)du + \int_{T}^{\infty} \psi^2(u)du < \varepsilon$$

Отсюда "значимая" часть $\psi(\cdot)$ содержится в $[-T, T]$, причем длина отрезка $2T$ может быть большой. Из второго свойства $\psi(\cdot)$ сбалансирована

относительно горизонтальной оси, что соответствует интуитивному понятию вейвлета. На Рис. 2 изображены примеры вейвлетов.

Их основное свойство – фиксация изменений локальных средних значений, что важно для количественной оценки. Пусть $x(\cdot)$ будет наш временной ряд (или "сигнал") – вещественная функция, t – время. Рассмотрим «среднее значение» $x(\cdot)$ на $[a, b]$:

$$\frac{1}{b-a}\int_a^b x(t)dt.$$

Уместно ввести локальное среднее, т.е. зависящее от точки и длины интервала:

$$A(\lambda, t) = \frac{1}{\lambda}\int_{t-\frac{\lambda}{2}}^{t+\frac{\lambda}{2}} x(u)du,$$

где λ — это длина интервала или «масштаб», t — середина интервала. Непрерывное вейвлет-преобразование определяется следующим образом:

$$W(\tau, t) = \frac{1}{\sqrt{\tau}}\int_{-\infty}^{\infty} x(u)\psi\left(\frac{u-t}{\tau}\right)du,$$

где $\psi(\cdot)$ – вейвлет-функция. В некотором смысле это взвешенное среднее, которое связано с масштабом и временем. Можно написать и обратное вейвлет-преобразование:

$$x(t) = \int_0^{\infty}\left[\frac{1}{C\tau^2}\int_{-\infty}^{\infty}\frac{1}{\sqrt{\tau}}W(\tau, u)\psi\left(\frac{t-u}{\tau}\right)du\right]d\tau,$$

где C – некоторая константа, зависящая лишь от вида вейвлет-функции. Поэтому есть взаимная однозначность между CWT и исходным временным рядом.

На практике в основном имеют дело с дискретными временными рядами, поэтому можно определить и дискретное вейвлет-преобразование (DWT). Легко заметить, что $W(\tau, t)$ непрерывна по второму аргументу,

для DWT интеграл заменяется суммированием, получаем W_{τ_i,t_j}:

$$W_{\tau_i,t_j} = \frac{1}{n(\tau_i,t_j)} \sum_{k=0}^{N-1} x_k \psi \left(\frac{u_k - t_j}{\tau_i} \right),$$

где

$$n(\tau_i,t_j) = \sum_{k=0}^{N-1} e^{-\frac{1}{B}\left(\frac{u_k-t_j}{\tau_i}\right)^2},$$

а константа B зависит от вида вейвлета: например, $B = 2$ для MHAT.

Основное преимущество данного метода – возможность работы с нестационарными временными рядами и обнаружения аномалий. Вейвлет-преобразования позволяет обнаруживать как краткосрочные колебания, так и долгосрочные тенденции. В отличие от преобразования Фурье, оно фиксирует локальную информацию о частоте в зависимости от времени, есть возможность исследовать изменение спектральных свойств ряда со временем. Алгоритм быстрого вейвлет-преобразования работает быстрее, нежели аналогичный алгоритм для преобразования Фурье (FFT). Суть же вейвлет-анализа заключается в поиске компактных структур в исходном ряде: он предоставляет нам информацию о спектральных характеристиках искомых структур, их локализации в пространстве параметров.

Алгоритм быстрого вейвлет-преобразования можно найти в работе Meyer Y., Salinger D. [18]. Более подробно ознакомиться с вейвлет-анализом: В.В. Витязев "Вейвлет-анализ временных рядов"[27].

1.4 Модели машинного обучения. Нейронные сети

Перейдем непосредственно к описанию основного метода нашего исследования, а именно, к прогнозированию временных рядов с помощью

моделей машинного обучения (ML). Эти модели нелинейны, в отличие от классических моделей, но при этом удобны для понимания.

Первый шаг в описании методов ML и построении нейронных сетей заключается в определении целевой функции.

Определение. *Целевая функция* – вещественная или целочисленная функция нескольких переменных, подлежащая оптимизации в целях решения некоторой оптимизационной задачи. В большинстве задач ML к ее минимизации.

Приведем некоторые примеры целевых функций:

$$l_2 = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{N_t}\sum_{i=1}^{N_t}\left(\sigma_{i,t} - f(x_{i,t-1})\right)^2, \tag{5}$$

$$l_1 = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{N_t}\sum_{i=1}^{N_t}|\sigma_{i,t} - f(x_{i,t-1})|, \tag{6}$$

$$l_H = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{N_t}\sum_{i=1}^{N_t}H\left(\sigma_{i,t} - f(x_{i,t-1}), \delta\right), \tag{7}$$

где

$$H(\epsilon, \delta) = \begin{cases} \epsilon^2, & \text{если } |\epsilon| \leqslant \delta \\ 2\delta|\epsilon| - \delta^2, & \text{иначе} \end{cases}$$

и $\sigma_{i,t}, f(x_{i,t})$ есть наш прогноз (с погрешностью) и наша искомая функция. Таким образом, нам требуется найти такую функцию, что он минимизирует наш функционал. Очень часто эта задача имеет вид задачи выпуклого программирования.

1.4.1 Методы оптимизации целевой функции

Практически всегда целевая функция является выпуклой, а значит если x_0 – локальный экстремум функции, то он является и глобальным.

Для поиска минимума или максимума целевой функции используют итеративные алгоритмы. Приведем примеры некоторых из них.

1. Градиентный спуск (англ. Gradient descent (GD)). Это самый простой итерационный метод. Основной его принцип основан на факте, что градиент – это направление наискорейшего локального возрастания функции. Если x_0 – начальная точка градиентного спуска. Следующая точка выбирается так:

$$x_{k+1} = x_k - \alpha \nabla f(x_k),$$

где α – это размер шага (learning rate). Возникает два естественных вопроса: как вычислять градиент (функция может быть не дифференцируемой) и как выбрать α? Метод наискорейшего спуска помогает ответить на второй вопрос:

$$\alpha_k = \arg\min_{\alpha \geq 0} f(x_k - \alpha \nabla f(x_k)).$$

2. Стохастический градиентный спуск (англ. Stochastic gradient descent (SGD)). Рассмотрим функционалы вида:

$$f(x) = \sum_{i=1}^{N} \mathcal{L}(x, y_i),$$

где N – количество элементов в выборке. Важный факт, лежащий в основе этого метода, состоит в том, что усреднение – это по сути взятие математического ожидания: $f(x) = \mathbb{E}[\mathcal{L}(x, \xi)]$, где ξ равномерно распределена по обучающей выборке. Вычислим градиент для таких функционалов: $\nabla f(x) = \mathbb{E}\nabla\mathcal{L}(x, \xi)$. Такое математическое ожидание вычислить напрямую невозможно, поэтому найдем его несмещенную Монте-Карло оценку или *стохастический градиент*:

$$\tilde{\nabla} f(x) = \frac{1}{B} \sum_{i=1}^{B} \nabla\mathcal{L}(x, \xi_i),$$

B – размерном батча (нашей выборки). Трудность заключается в том, что необходимо генерировать батчи на каждом шаге, однако это может быть решено с помощью перемешивания нашей выборки. Общий алгоритм SGD:

- Случайным образом перемешивается набор данных размера в количестве B;
- Выбирается скорость обучения α;
- Определяем начальные значения параметров θ – точка с которой мы стартуем;
- Обновляем параметры на основе x^i, y^j:

$$\theta_{i+1} = \theta_i - \alpha \times \nabla_\theta J\left(\theta, x^i, y^j\right),$$

  $J\left(\theta, x^i, y^j\right)$ – оценка целевой функция в предположении, что она дифференцируема по параметру (иначе аппроксимируем ее);
- Повторяем первые четыре шага до достижения локального минимума.

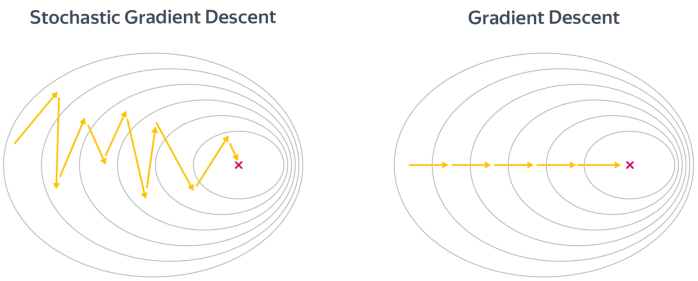


Рис. 3: Сравнение GD и SGD

Существуют и другие модификации методов поиска минимума, например, Adam, фактически являющийся наилучшим методом стохастической оптимизации, RMSProp, AdamW, Adagrad и другие. Подробнее о них можно ознакомиться в [8], [9], [13], [31].

1.4.2 Простейшие модели нейронных сетей

Более сложной альтернативой методам ML, также учитывающей нелинейности, служит класс моделей искусственных нейронных сетей. Нелинейность, присущая моделям нейронных сетей, связана с представлениями, которые выражаются через другие, более простые представления. Несмотря на свои преимущества, модели нейронных сетей трудны для интерпретации.

Начнем с описания самой простой модели нейронной сети – сети с прямой связью (FF). В таких моделях связь между входным слоем предикторов и переменной отклика создается одним или несколькими скрытыми слоями, которые взаимодействуют друг с другом, нелинейно преобразуя предикторы. Это нелинейное преобразование основано на поэлементном применении функции активации, например, Rectified Linear Unit (ReLU):

$$\mathcal{A}_{ReLU}(x) = \max(0, x). \tag{8}$$

Результатом функции ReLU является l-й скрытый слой, примененный к $(l-1)$-му слою:

$$x^{(l)} = \mathcal{A}_{ReLU}\left(\beta_0^{(l-1)} + \sum_{u=1}^{U^{(l-1)}} \beta_u^{(l-1)} x_u^{(l-1)}\right), \tag{9}$$

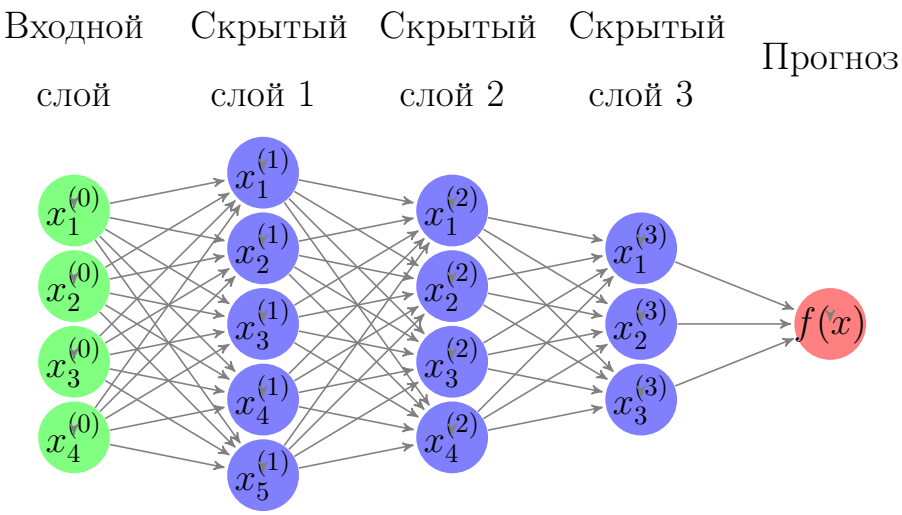
где $x^{(l)} = \left(x_1^{(l)}, ..., x_{U^l}^{(l)}\right) \in \mathbb{R}^{U^{(l)}}$ – выходной вектор для слоя l, $l = 1, ..., L$, $x_u^{(l)}$ – выходной сигнал узла u в l-ом слое, $U^{(l)}$ – число узлов в слое l. При

$l = 0$ вектор $x^{(0)} = (x_1^{(0)}, ..., x_U^{(0)})$ является входным слоем предикторов. На последнем этапе получаем окончательный прогноз:

$$f\left(x, \beta, U^{(l)}, L\right) = \beta_0^{(l)} + \sum_{u=1}^{U^{(L)}} \beta_u^{(L)} x_u^{(L)}. \tag{10}$$

На Схеме 1 представлен пример сети с входным слоем (зеленый), тремя скрытыми слоями (синий) и слоем полученных прогнозов (красный).

Схема 1: Нейронная сеть с прямой связью



Построение прогноза состоит в нахождении весов $\beta_u^{(l)}$, которые определяют взаимосвязи между входными и выходными данными. Для сетей с большим числом слоев количество таких весов может быть достаточно большим, что делает оценку трудоемкой с точки зрения вычислений. Число параметров в l-ом скрытом слое равно $\left(1 + U^{(l)}\right) \times U^{(l)}$ и $1 + U^{(L)}$ параметром в выходном слое.

1.5 Выводы

В данной главе была дана постановка задачи исследования: было определено, что такое финансовые временные ряды, их основные свойства – персистентность, нестационарность и другие. Фокус, на котором

было заострено внимание, заключался в описании классических методов, с их достоинствами и недостатками, среди которых были выделены модели ARIMA, ее модификация SARIMA, модели на основе марковских цепей, более подробно был рассмотрен вейвлет-анализ. Стоит отметить, что основное их преимущество высокая интерпретируемость и низкая вычислительная сложность, однако они уступают по качеству прогнозов моделям на основе нейронных сетей. Были введены основные понятия, связанные с нейронными сетями: целевая функция и методы ее оптимизации, приведен пример простейшей нейронной сети. В следующих главах они будут использоваться как базовый инструмент исследования.

Глава 2. Сети с долгой краткосрочной памятью (LSTM - сети)

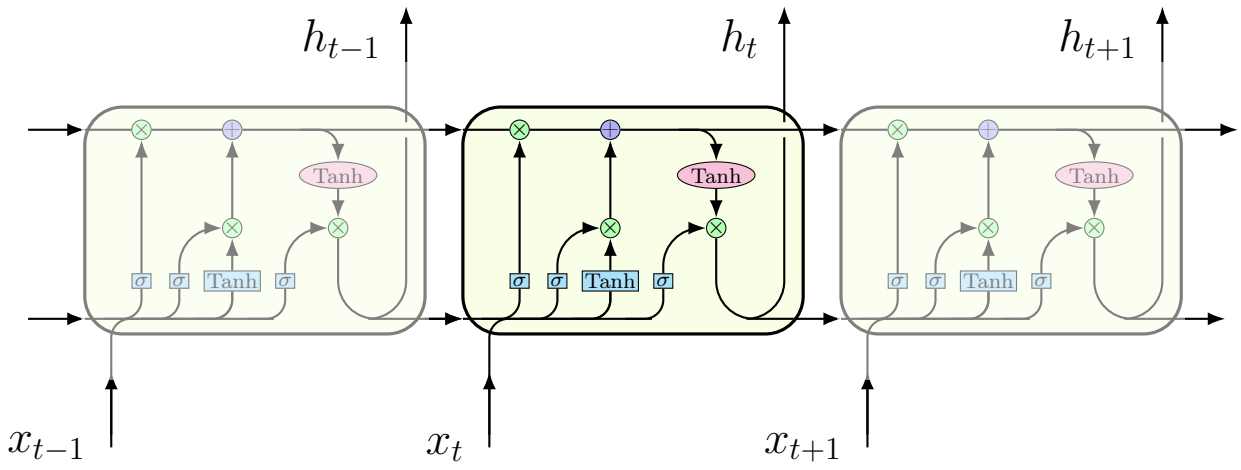
Нейронные сети с долгой краткосрочной памятью (или на англ. Long Short Term Memory – LSTM) – особый вид рекуррентный нейронных сетей, способный выявлять долгосрочные зависимости во временных рядах, что необходимо при прогнозировании финансовых котировок. Впервые они встречаются в работе Sepp Hochreiter и Jurgen Schmidhuber "LONG SHORT-TERM MEMORY" (1997 г.), в которой анализируется возможность обучения с хранением информации в течение длительных интервалов времени, однако это занимает очень много времени, в основном из-за затухающего обратного распространения ошибок. Данная модель нейронных сетей получила большое применение в различных задачах анализа данных: от финансовой индустрии до прогнозирования солнечной активности и биоинформатики.

2.1 Ключевые обозначения. Механизм работы

Как и все рекуррентные нейронные сети LSTM есть последовательность повторяющихся юнитов сети, однако они имеет другую структуру: вместо одного слоя нейронной сети их четыре, взаимодействующих особым образом. Рассмотрим схему 1. Характерной особенностью этой сети является последовательное включение в цепь клеток, причем на каждом шаге у нас на вход поступает прошлое явное и скрытое состояние ячейки, а также новые данные. Как можно видеть из этой схемы, мы не только вычисляем текущее состояние на каждом шаге, но и опреде-

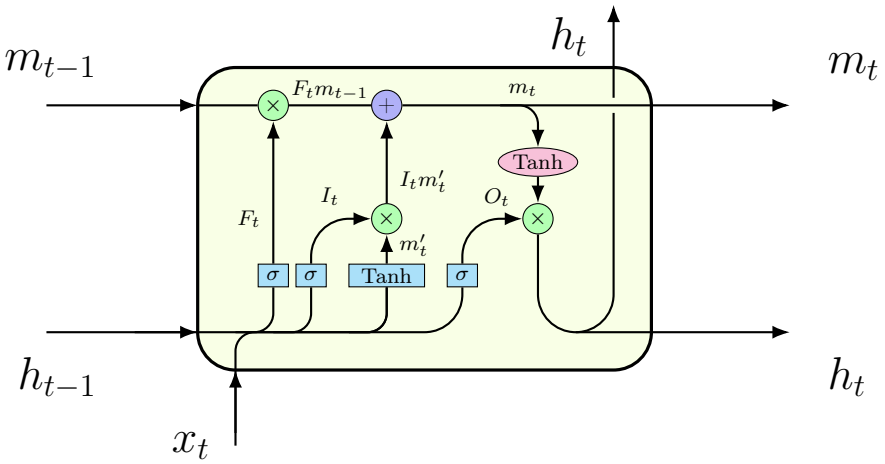
ляем скрытое, которое помогает избавиться от проблемы долгосрочной зависимости.

Схема 1: Модель Long Short Term Memory NN



Более подробно остановимся на схеме 2 и её обозначениях. Для начала определим операции, которые присутствуют в модуле LSTM. На приведенной выше диаграмме каждая строка несет целый вектор от выхода одного узла ко входам других.

Схема 2: Клетка LSTM-сети



Синие прямоугольники – обучаемые слои нейронной сети. $\otimes$ – поточечное умножение, $\oplus$ – поточечное сложение векторов, $Tanh$ в розовом

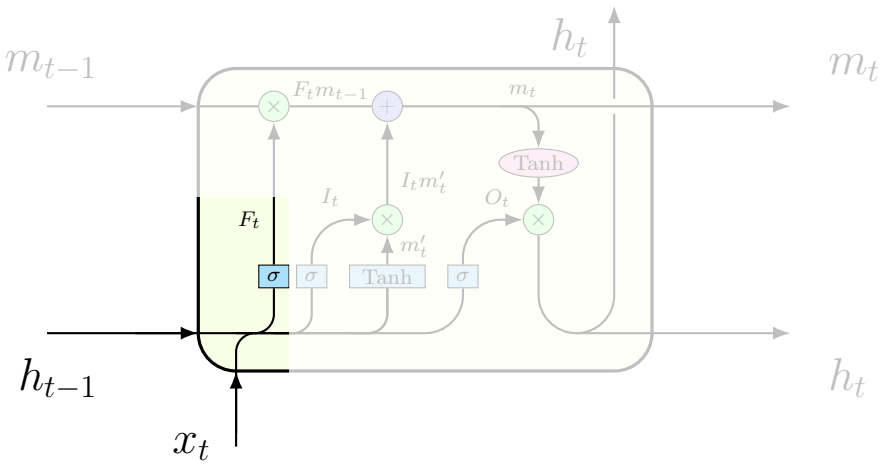
эллипсе есть поточечное применение функции гиперболического тангенса. Разветвление строки означает копирование ее содержимого и перемещение копий в разные места.

Формализуем механизм работы LSTM-сети. В каждый момент времени на долговременную память может влиять долговременная память, перенесенная из предыдущего времени – m_{t-1}, и текущая информация – m'_t. Первый шаг — решить, какую информацию мы собираемся выбросить из состояния ячейки. Это решение принимает сигмовидный слой, называемый «**вентелем забвения**» (англ. "**forget gate layer**"). На схеме 3 изображен данный этап работы нейронной сети. На этом шаге мы получаем:

$$F_t = \mathcal{A}_{log}\left(W_f \cdot [h_{t-1}, x_t] + b_f\right),$$

где $\mathcal{A}_{log}$ – логистическая функция активации (или сигмоида), h_{t-1} – вектор прогнозов из предыдущего периода, x_t – вектор входных данных текущего периода, $[h_{t-1}, x_t]$ – объединенный вектор, W_f, b_f – матрица весов и вектора смещений соответственно, которые как раз и получаются в процессе обучения.

Схема 3: Forget Gate Layer



Второй шаг — решить, какую информацию мы собираемся хранить в состоянии ячейки. Это состоит из двух частей: сигмоидного слоя, называемого «**слоем входных вентилей**» (англ. "**Input gate layer**"), основная задача которого определить, какие значения мы будем обновлять:

$$I_t = \mathcal{A}_{log}\left(W_i \cdot [h_{t-1}, x_t] + b_i\right),$$

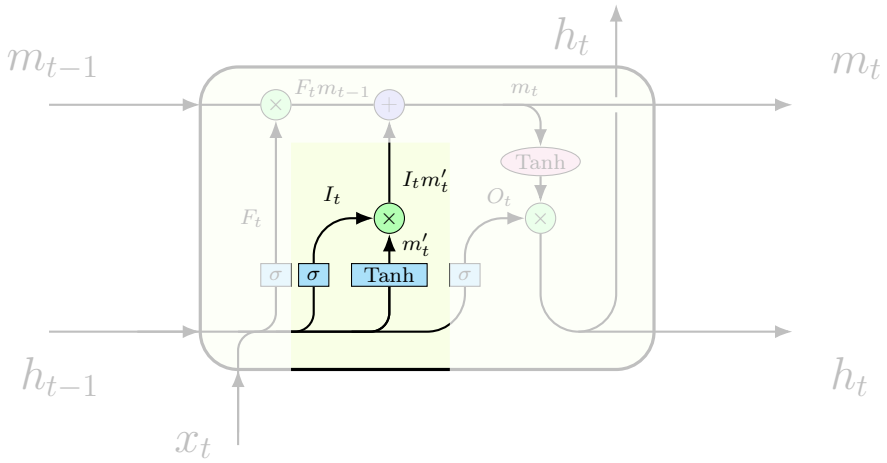
где W_i, b_i – матрица весов и вектора смещений соответственно, применяемые на данном этапе, и слоя гиперболического тангенса, где создается вектор новых "кандидатов" , которых можно добавить в скрытое состояние:

$$m'_t = \mathcal{A}_{tanh}\left(W_c \cdot [h_{t-1}, x_t] + b_c\right),$$

W_c, b_c определяются аналогично. Затем объединим два этих слоя с помощью поточечного умножения:

$$\hat{m}_t = I_t \otimes m'_t. \tag{11}$$

Схема 4: Input Gate

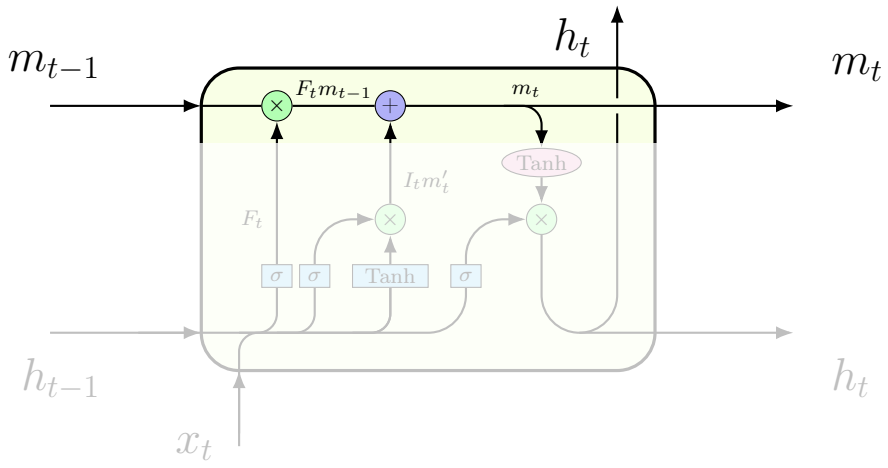


Третий шаг – добавляем наши полученные ранее значения в вектор скрытого состояния, т.е.

$$m_t = F_t \otimes m_{t-1} + I_t \otimes m'_t. \tag{12}$$

На схеме 5 изображен данный этап обучения нейронной сети.

Схема 5: Получение скрытого состояния ячейки



Последний шаг – получение открытого состояния ячейки (см. схему 6), т.е. что мы хотим вывести после обработки. Вначале с помощью сигмоиды получаем части состояния ячейки мы собираемся выводить:

$$O_t = \mathcal{A}_{log}\left(W_o \cdot [h_{t-1}, x_t] + b_o\right),$$

W_o, b_o – матрица весов и вектора смещений соответственно на данном этапе. Затем передаем скрытое состояние ячейки предварительно обработав его через функцию гиперболического тангенса, чтобы значения принадлежали $[-1, 1]$:

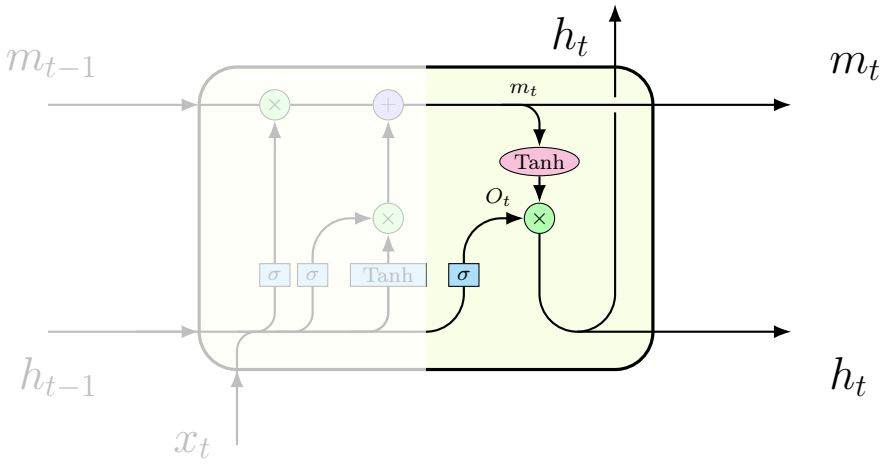
$$\hat{h}_t = \mathcal{A}_{tanh}(m_t),$$

где $\mathcal{A}_{tanh}$ – функция активации гиперболического тангенса. Последняя операция – поточечное умножение двух векторов (умножаем его на вы-

ходной сигнал сигмоидного вентиля для вывода только тех частей, которых решили):

$$h_t = O_t \otimes \mathcal{A}_{tanh}(m_t). \tag{13}$$

Схема 6: Output Gate



Следует отметить, что архитектура LSTM влечет за собой оценку большого числа параметров: матриц W_f, W_o, W_i, W_c, и векторов смещений b_f, b_o, b_i, b_c. Количество параметров для слоя l сети LSTM можно рассчитать как $4 \times U^{(l)} \times \left(1 + U^{(l)} + U^{(l-1)}\right)$, где $U^{(l)}$ обозначает количество единиц в слое l. При добавлении слоев значительно увеличивается количество параметров, что значительно снижаем вычислителью производельность нейронной сети данного типа. Следующий раздел посвящен оптимизации вычислений параметров модели LSTM-сети.

2.2 Методы оптимизации вычислений в модели LSTM

В среднем сеть LSTM имеет примерно в 5 раз больше параметров по сравнению с сетью прямого распространения, что существенно снижает

скорость обучения модели: для прогнозирования финансовых временных рядов это существенный показатель – мы хотим достаточно быстро получать точные модели. В связи с этим возникает вопрос, как можно усовершенствовать наши вычисления.

Существует значительный стимул для улучшения производительности и масштабируемости этих сетей. Хотя графические процессоры (далее GPUs) стали предпочтительным оборудованием для обучения и развертывания рекуррентных моделей, используемые реализации часто используют только базовые оптимизации для этих архитектур. В данном разделе будет показано, как с помощью параллелизма между операциями можно добиться ускорения обучения модели. Основные принципы, которые будут описаны далее, были впервые изложены в [10].

2.2.1 Базовые оптимизации

Существует достаточно много способов оптимизации одного шага в обучении нейронной сети. Для этого рассмотрим этапы построения модели рекуррентной LSTM: Input Gate, Output Gate и другие, указанные в предыдущем разделе. В качестве отправной точки необходимо исследовать реализацию, в которой каждая отдельная операция (т. е. умножение матриц, сигмоида, поточечное сложение и т. д.) является отдельным ядром. Хотя GPU выполняет операции внутри каждого ядра параллельно, ядра выполняются последовательно. Производительность прямого прохода в такой реализации низкая.

Довольно часто используется оптимизация, основной смысл которой слияние мелких матричных операций с одинаковыми входными данными, в одну более крупную матричную операцию. При прямом проходе стандартная формулировка LSTM приводит к восьми умножениям мат-

риц: четыре операции на рекуррентном входе, четыре операции на входные данные предыдущего слоя. В этих четырех группах операций входы являются общими, а веса W_i нет. Поэтому мы можем четыре матричных умножения представить в виде одного с матрицей в четыре раза большей исходных: большие матричные операции более параллельны, их проще реализовать на GPUs. Такая оптимизация возможна лишь для RNN, в которых более одного гейта (т.е. более одного матричного умножения), например, LSTM.

2.2.2 Потоковые матричные операции

Одновременное умножение матриц на GPU является одним из простых способов сделать обучение более эффективным за счет параллельных вычислений. Используя поточные вычисления, можем считать, что матрицы умножения независимы: происходит удвоение производительности до 2 раз при умножении небольших матриц. При умножении больших матриц потоковая передача остается полезной, т.к. сводит к минимуму "эффект хвоста" (недостаточное использование графического процессора при трате значительной часть времени выполнения). Если количество блоков, запущенных в GPU, достаточно только для того, чтобы заполнить его потоковый мультипроцессор несколько раз, можно считать, что они проходят через GPU волнами. Все блоки в первой волне заканчиваются примерно одинаково время, и когда они заканчивают, начинается вторая волна. Это продолжается до тех пор, пока больше не останется работы. Если количество волн невелико, на последней волне часто будет меньше работы, чем на других, создавая "хвост" низкой производительности. За счет увеличения параллелизма этот хвост может

быть перекрыт другой операцией, что снижает снижение производительности.

2.2.3 Слияние точечных операций

Для точечных операций свойственен параллелизм, однако зачастую он бывает неэффективен. Существует две причины этого явления. Первая – запуск ядра GPU довольно дорогая операция, а вторая – неэффективность перемещения выходные данных одной точечной операции в основную память GPU. Так как поточечные операции независимы, их можно объединить в одно крупное ядро.

2.2.4 Единичный слой

Один повторяющийся слой содержит множество ячеек, повторяющиеся входные данные каждой из которых зависят от выходных данных предыдущей. Входные данные с предыдущего уровня могут не иметь такой зависимости, и часто возможно объединить входные данные для нескольких временных шагов, что приводит к большему и более эффективному умножению матриц. Выбор количества шагов для объединения не является тривиальным: большее количество шагов приводит к более эффективному умножению матрицы, но меньшее количество шагов сокращает время.

2.2.5 Очередность операций

Для рекуррентных нейронных сетей становится все более распространенным использование нескольких повторяющихся слоев, “сложенных” таким образом, что каждая повторяющаяся ячейка передает свои выход-

ные данные непосредственно в повторяющуюся ячейку следующего слоя. В этой ситуации можно использовать параллелизм между повторяющимися слоями: завершение повторяющейся ячейки устраняет зависимость не только от следующей итерации текущего слоя, но и от текущей итерации следующего слоя. Это позволяет параллельно вычислять несколько уровней, значительно увеличивая объем работы графического процессора в любой момент времени.

Поскольку запуск работы на графическом процессоре занимает небольшое, но не незначительное количество времени, важно учитывать порядок, в котором ядра запускаются на графическом процессоре. Например, если ресурсы графического процессора доступны, почти всегда предпочтительнее запускать ядро с разрешенными всеми его зависимостями, а не ядро, которое может некоторое время ждать, пока его зависимости будут очищены. Таким образом, можно обеспечить как можно больший параллелизм.

2.2.6 Обновление весов

Упомянутые выше оптимизации могут быть использованы лишь к шагам распространения. В момент завершения распространения градиента обновление веса становится эффективным. Для обновления каждой матрицы без зависимостей можно использовать одно умножение большой матрицы, и это обычно обеспечивает производительность, близкую к пиковой. Обновление весов смещения обходится очень дешево по сравнению с обновлением матриц.

2.3 Выводы

LSTM (Long Short-Term Memory) модели имеют широкое применение в различных областях, включая прогнозирование временных рядов. Этот тип нейронных сетей оказался очень эффективными при работе с финансовыми временными рядами, где часто приходится иметь дело с нестационарными и сложными данными. Использование LSTM моделей в финансах включает в себя как предсказание цен на акции, валютных курсов, индексов и других финансовых показателей, как и построения тренда ряда, а это уже задача классификации: в этом случае эксперимент сводится к тому, что нейронная сеть пытается определить тенденцию временного ряда на один следующий таймфрейм.

Эти модели способны уловить сложные зависимости между различными факторами, влияющими на финансовый рынок, и использовать эту информацию для составления прогнозов. Одним из преимуществ LSTM моделей является их способность “запоминать” и использовать прошлую информацию для предсказания будущих значений временного ряда. Это особенно важно для финансовых временных рядов, где прошлые данные могут дать ключ к пониманию текущих и будущих трендов. Значительный недостаток LSTM заключается в неинтерпритируемости построенной модели, в отличие от других методов машинного обучения, таких как деревья решений. При этом для получения точных прогнозов может потребоваться значительное количество данных и время на обучение модели.

В данной главе был сделан упор на механизм вычислений при построении модели, а также методам их оптимизации: очередности операций, слиянии матричных и поточечных операций и других.

Глава 3. Сети Transformer

Transformer – это одна из самых передовых архитектур в области глубокого обучения, созданная компанией Google в 2017 году [20]. С помощью сетей Transformer удается достигать высоких результатов в обработке естественного языка, и они успешно применяются в сфере машинного перевода, генерировании текста и задачах, связанных с обработкой последовательностей. В этой главе мы рассмотрим основные принципы работы сетей Transformer, а также рассмотрим их применение в финансовых задачах. Общая архитектура представлена на Рис. 4

3.1 Архитектура сетей Transformer

3.1.1 Механизм Attention

Механизм масштабируемого скалярного произведения «Attention» представляет собой поиск по ключу-значению на основе матрицы запроса Q, матрицы ключей K и матрицы значений V, описанных как:

$$Attention(Q, K, V, \alpha) = \alpha\left(\frac{QK^T}{\sqrt{d_q}}\right)V, \tag{14}$$

где $Q \in \mathbb{R}^{N\times d_q}$, $K \in \mathbb{R}^{M\times d_q}$, $V \in \mathbb{R}^{M\times d_v}$ и $\alpha(\cdot)$ – функция активации, применяемая построчно. d_q и d_v представляют размерности вектора запроса и значения соответственно. $softmax(\cdot)$ – функция активации, используемая в Transformers, определяется как:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}.$$

Интуитивно $softmax(\cdot)$ преобразует вектор x в вектор вероятностей, для которого сумма элементов равна 1.

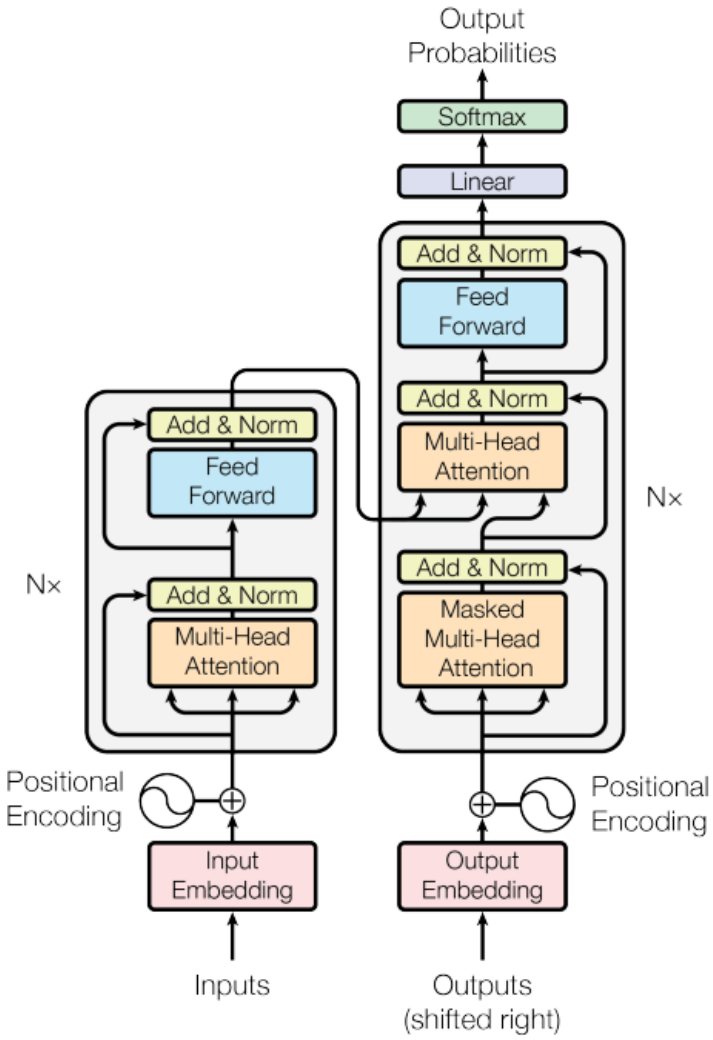


Рис. 4: Общая архитектура сети Transformer

Transformer, по сравнению с RNN, легко распараллеливаются и могут передавать несколько последовательных временных рядов в пакете для кодирования и декодирования. Однако предоставление последовательных временных рядов может помешать обучению Transformer, поскольку он может использовать временные ряды в пакете, чтобы увидеть «будущее». Маскированное внимание не позволяет Transformer видеть будущие значения:

$$MaskedAttention(Q, K, V, \alpha, M) = \alpha \left(mask \left(\frac{QK^T}{\sqrt{d_q}}, M \right) \right) V, \qquad (15)$$

где M_{nm} равен 0 при маскировке и 1 в противном случае. В случае $M_{nm} = 0$, положим $\left(QK^T\right)_{nm} = -\infty$. Тогда в $softmax(\cdot)$ значение $\exp\left(\frac{QK^T}{\sqrt{d_q}}\right) = 0$. Следовательно, значение v_m не будет учитываться в выводе Attention для запроса q_n.

Transformer объединяют несколько Attention в многоголовочный модуль Attention, где каждая головка захватывает разные части последовательности, что позволяет выполнять более точное кодирование временных рядов.

3.1.2 Кодирование положения

Кодер состоит из входного слоя, слоя позиционного кодирования и стека из четырех идентичных слоев кодировщика. Входной слой отображает входные данные временных рядов в вектор размерности d через полносвязную сеть. Этот шаг необходим для модели, чтобы использовать механизм внимания с несколькими головками. Позиционное кодирование с функциями синуса и косинуса используется для кодирования последовательной информации в данных временного ряда путем поэлементного сложения входного вектора с вектором позиционного кодирования. Результирующий вектор подается на четыре слоя кодировщика. Каждый уровень кодировщика состоит из двух подуровней: подуровня с самостоятельным вниманием и подуровня с полной прямой связью. За каждым подуровнем следует слой нормализации. Кодер создает вектор размерности d для подачи на декодер.

Стоит отметить, что масштабированное скалярное произведение Attention эквивалентно перестановке:

$$Attention(PQ, K, V, \alpha) = \alpha\left(\frac{PQK^T}{\sqrt{d_q}}\right)V = \\ = P\alpha\left(\frac{QK^T}{\sqrt{d_q}}\right)V = PAttention(Q, K, V, \alpha). \quad (16)$$

Поскольку информация о порядке имеет значение при прогнозировании временных рядов, когда мы вводим последовательность запросов $q_1, q_2, ..., q_N$, мы строим модифицированную версию матрицы запроса Q:

$$\tilde{Q} = \left(\tilde{q_1}, ..., \tilde{q_N}\right)^T, \; \tilde{q_n} = f\left(q_n, PE(n)\right), \quad (17)$$

где $f(\cdot)$ – суммирование и конкатенация, $PE(n)$ – позиционное кодирование ввода с индексом n, которые можно узнать или вычислить.

3.1.3 Нормализация слоя

Нормализация слоя используется для стабилизации обучения нейронной сети. Пусть $y = \{y_1, \ldots, y_n\} = Wx + b$ будет выходом слоя прямой связи перед прохождением функции активации. Нормализация слоя определяется как:

$$\frac{y - \hat{y}}{\sigma} \to y, \; \hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i, \; \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2}. \quad (18)$$

В Transformers нормализация уровня применяется с остаточным соединением:

$$Add\&Norm(x) = LayerNorm(x + Sublayer(x)), \quad (19)$$

где $Sublayer(x)$ может быть либо многоголовочным Attention, либо точечной сетью прямой связи.

3.1.4 Точечная сеть прямой связи

Transformer используют точечную сеть на уровне прямой связи. Выходные данные слоя Attention с несколькими заголовками после слоя Add&Norm представляют собой матрицу $N \times d_{out}$, которая содержит результаты Attention для N запросов. Точечная сеть прямой связи обрабатывает эту матрицу построчно.

3.2 Современные архитектуры Transformer

3.2.1 Autoformer

Autoformer использует новый тип Attention, основанный на автокорреляции в кодировщике и декодере. Кодер устраняет долгосрочный тренд путем разложения ряда с помощью быстрого преобразования Фурье и фокусируется на моделировании сезонных закономерностей. Прошлые сезонные данные и информация о тенденциях от кодировщика затем используются декодерем для предсказания будущих значений. По сравнению с полным модулем Attention Transformer автокорреляционный модуль Attention позволяет снизить сложность $O(L \ln(L))$ по сравнению с $O(L^2)$ для временного ряда длиной L. Повышение производительности связано с запросом только первых записей журнала $\ln(L)$ на основе периодичности серии.

3.2.2 FEDformer

FEDformer использует декомпозицию сезонного тренда для прогнозирования будущих значений. FEDformer использует либо быстрое преобразование Фурье, либо вейвлет-преобразование для преобразования

временного ряда в частотную область. Затем он случайным образом запрашивает модуль Attention в частотной области для высокочастотных и низкочастотных компонентов, что снижает сложность $O(L^2)$ до $O(L)$ для временных рядов длиной L. Целью использования частотной области является достижение оптимального сочетания высокочастотных компонентов, на которые больше влияет локальный шум, и низкочастотных компонентов, на которые больше влияет общая тенденция временного ряда.

3.2.3 HFformer

HFformer – новая архитектура Transformer, объединяющая в себе две предыдущие модели.

Кодер HFformer основан на энкодере Transformer с функциями активации, которые были изменены, чтобы быть пиковыми:

$$SpikingActivation(x) = \begin{cases} x, & \text{если } x \leqslant threshold \\ 0, & \text{иначе} \end{cases} \tag{20}$$

где порог возникает во время обучения модели.

Линейный декодер содержит два линейных уровня, связанных функцией активации PReLU. Декодер должен выводить одно значение для каждого горизонта прогноза, когда мы прогнозируем логарифмическую доходность. Переход от авторегрессионного декодера к линейному декодеру помогает уменьшить распространение ошибок и сократить время обучения.

Позиционное кодирование было удалено из модели. В результате производительность модели увеличилась, а время обучения сократилось за счет уменьшения количества обучаемых параметров. Позиционное коди-

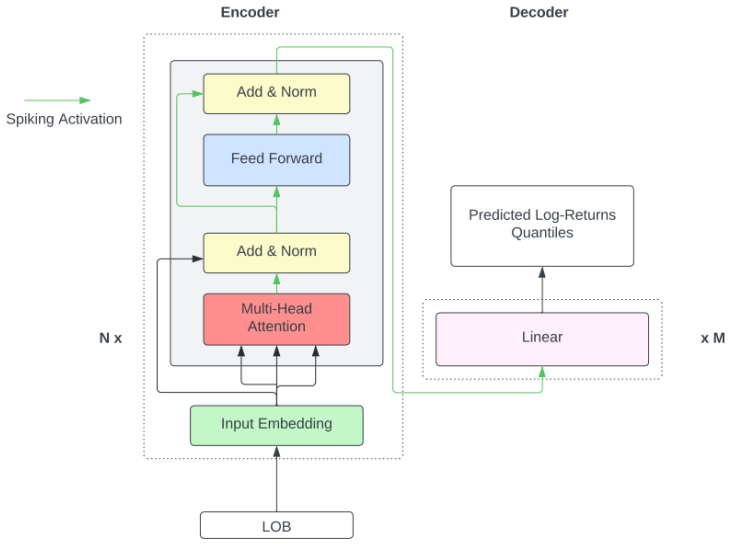


Рис. 5: Архитектура HFformer

рование было введено для задач NLP. Он используется как метод усиления механизма само-внимания, который инвариантен к порядку последовательности. Однако, поскольку нам больше не нужен авторегрессионный декодер, поскольку мы используем линейный декодер, позиционное кодирование становится менее актуальным. Кроме того, количество информации предоставляемый энкодеру Transformer, увеличивается с размером положения, что может служить формой позиционное кодирование [13].

Квантиль потерь определяется следующим образом:

$$L\left(\hat{y}_i^p, y_i\right) = \max\left\{q\left(\hat{y}_i^p - y_i\right), (q-1)\left(\hat{y}_i^p - y_i\right)\right\}, \tag{21}$$

где $\hat{y}_i^p$ – выход модели для заданного квантиля p, y_i – истинное значение. HFformer работает с потерями MSE и MAE. Однако квантиль потерь позволяет прогнозировать интервал вокруг прогнозируемого значения.

3.3 Методы оптимизации вычислений в моделе Transformer

Одной из важных причин, по которой были созданы данные модели, было продолжительное время обучения и распространения её предшественников – RNN и CNN (свёрточных нейронных сетей). Это было сделано за счет упрощения операций и уменьшения их числа, а также возможности использования методов параллельных вычислений. Как было отмечено в подразделе, посвященном описанию архитектуры сети Transformer, каждая головка multi-head работает параллельно (без связей между собой), поэтому можно проводить отдельно вычисления на GPU, а результат объединить в конце операции. Также существуют и другие подходы к модификации этой нейронной сети. Разберем некоторые из них.

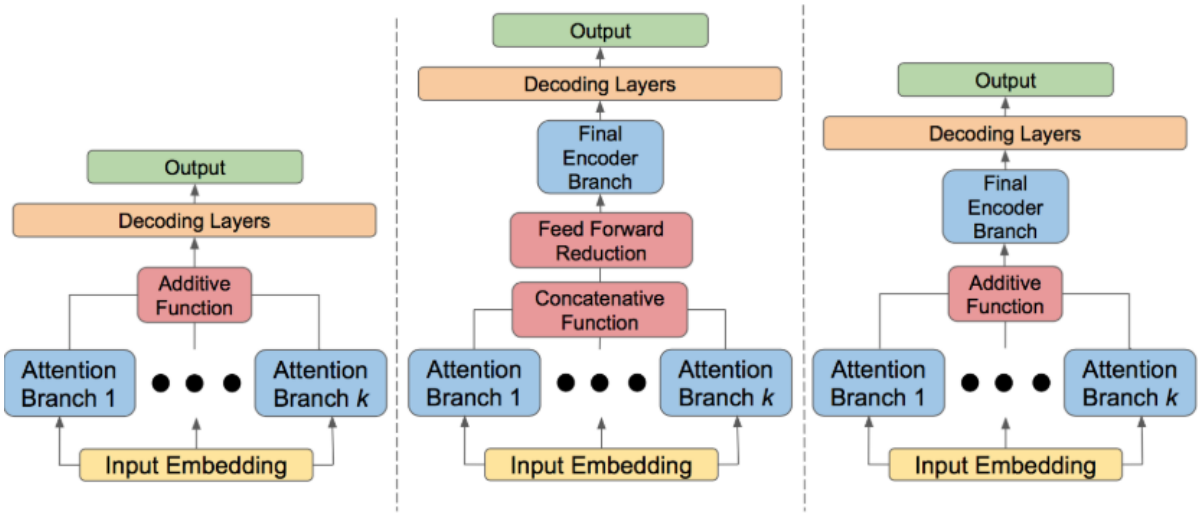


Рис. 6: 1 – Additive Parallel Attention (APA); 2 – Attended Concatenated Parallel Attention (ACPA); 3 – Attended Additive Parallel Attention (AAPA)

Аддитивный параллельный Attention (APA). Можно заменить весь стек (multi-head attention, add&norm, feedforward, add&norm), повторяющийся N раз в оригинальной архитектуре Transformer (см. левую часть Рис. 4), а вместо этого использовать у несколько подсетей Attention, ра-

ботающих параллельно. Выходные уровни этих сетей содержат вложения для применения механизма Attention к входным данным. Далее добавляются значения на выходных уровнях стеков.

Управляемый объединенный параллельный Attention (ACPA). Этот подход аналогичен APA, но значения на выходных уровнях подсетей Attention объединяются, а не добавляются.

Управляемый аддитивный параллельный Attention (AAPA). Эта улучшенная модель построена аналогично APA. Однако она удаляет один из параллельных стеков и использует его в качестве окончательного механизма последовательного Attention к дополнительным результатам.

На Рис. 3.3 наглядно представлены три данных подхода. В рамках данной работы будет выполнено лишь распараллеливание multi-head.

3.4 Применение к финансовым задачам

Нейронные сети Transformer относятся к классу NLP и в основном используются при генерации текста, машинном переводе. Однако интересным стало их приложение к прогнозированию финансовых временных рядов. В последнее время стало популярным их использование в высокочастотной торговле (HFT) (см. статьи ниже).

В чем же их преимущество по сравнению с другими моделями нейронных сетей в нашей задаче? Во-первых, эта архитектура позволяет моделировать длинные зависимости между элементами последовательности, что особенно важно для финансовых временных рядов, которые могут содержать множество скрытых факторов, влияющих на их поведение. Во-вторых, механизм само-внимания позволяет моделировать переменные зависимости между элементами последовательности, что так-

же важно для финансовых временных рядов, которые могут содержать нелинейные зависимости.

Основная проблема состоит в том, как закодировать временной ряд, исходя лишь из исторических цен. Например, это можно сделать с помощью японских свечей.

3.5 Выводы

В данной главе были рассмотрены основные аспекты связанные с прогнозированием финансовых временных с помощью нейронных сетей Transformer. Дано подробное описание архитектуры данной модели с предложениями по оптимизации вычислений в ней, а также приведены методы модификации обучения на основе параллельных вычислений.

В целом, можно отметить следующие преимущества нейронной сети Transformer перед другими моделями машинного. Transformer предназначен для обработки последовательностей, таких как текст на естественном языке, что делает его полезным для задач машинного перевода и генерации текста. Еще одним достоинством является отсутствие зависимости от порядка обработки: в отличие от рекуррентных нейронных сетей (RNN), Transformer не требует обработки последовательностей по порядку, что упрощает его использование и ускоряет обучение. Использование механизма Attention обеспечивает точное представление информации и улучшает качество результатов. И, конечно, наиболее важным является параллелизация процессов обучения: благодаря своей архитектуре, Transformer легко распараллеливается, что позволяет обучать модель быстрее на больших объёмах данных.

Глава 4. Облачные вычисления. Параллельные вычисления

4.1 Преимущества языка Python для моделей нейронных сетей

Язык программирования Python является одним из наиболее используемых для работы с нейронными сетями благодаря своей простоте, эффективности и наличию множества библиотек. Можно выделить несколько явных его достоинств для получения результатов исследования. Во-первых, Python имеет простой и понятный синтаксис, который позволяет быстро работать с нейронными сетями и машинным обучением, является интерпретируемым языком. Во-вторых, этот язык программирования один из самых популярных, что обеспечивает доступ к большому количеству документации, самих программ. В-третьих, для Python реализовано множество библиотек для работы с нейронными сетями, таких как TensorFlow, PyTorch, Keras и другие, о которых пойдет речь далее, что облегчает процесс разработки и обучения моделей. Стоит отметить, Python может работать медленнее, чем некоторые другие языки программирования, но благодаря использованию GPUs и многоядерных процессоров, производительность может быть довольно высокой.

4.2 Библиотеки для получения данных, их обработки и графического представления

Важным отличительным свойством языка Python является наличие библиотек для получения данных, их обработки и графического представления. Перечислим лишь необходимые нам для работы:

- *Pandas* – простой и мощный инструмент для работы с данными, позволяет манипулировать данными, выполнять различные статистические операции, визуализировать данные;
- *Numpy* – библиотека для работы с многомерными массивами, позволяет выполнять различные математические операции с массивами;
- *Matplotlib* – библиотека для визуализации данных, позволяет создавать различные типы графиков и диаграмм;
- *Seaborn* – библиотека для визуализации данных в статистическом анализе;
- *Plotly* – библиотека для создания интерактивных визуализаций данных;
- *Scikit-learn* – библиотека для машинного обучения и анализа данных.

Для выгрузки финансовых данных из различных источников можно использовать библиотеку pandas datareader. Эта библиотека позволяет

получать данные (цены на акции, индексы, валюты и т.д.) с различных финансовых рынков. Кроме того, можно использовать библиотеку yfinance, которая берет данные с источника Yahoo Finance.

4.3 Библиотеки, реализующие нейронные сети

Для построения моделей нейронных сетей нам потребуются библиотеки, реализующие их. Следует отметить самые важные и наиболее используемые из них:

- *TensorFlow* – это популярная библиотека для работы с глубокими нейронными сетями. Она предоставляет инструменты для создания, обучения и тестирования нейронных сетей;
- *PyTorch* – еще одна популярная библиотека для работы с нейронными сетями, имеющая более простой и гибкий интерфейс, чем TensorFlow;
- *Keras* – это библиотека, которая работает поверх TensorFlow или PyTorch. Она предоставляет набор готовых слоев для создания нейронных сетей.

В данной работе отдается предподчтение библиотеке TensorFlow. На базе неё могут быть использованы технологии для оптимизации производительности и ускорения обучения моделей, поскольку эта библиотека поддерживает работу с графическими процессорами (GPU). TensorFlow является одним из лидеров среди библиотек для машинного обучения.

4.4 Методы параллельных вычислений в Python на основе Google Colab

Google Colab — это облачная среда, удобная для машинного обучения и анализа данных, которая предоставляет доступ к ресурсам Google, таким как GPU и TPU, для ускорения вычислений. Для начала определим, какие виды параллельных вычислений предлагает Python:

- **Многопоточность (Threading)**. Многопоточность позволяет выполнять несколько потоков одновременно в рамках одного процесса. В Python для создания и управления потоками используется стандартная библиотека threading.

- **Многопроцессорность (Multiprocessing)**. Многопроцессорность позволяет использовать несколько процессов для параллельного выполнения кода, что обеспечивает более эффективное использование многоядерных процессоров. В Python для этого метода библиотека multiprocessing.

Среда выполнения GPU более гибкая и хорошо программируема при нерегулярных вычислений, таких как небольшие батчи и вычисления без использования MatMul. Среда выполнения TPU оптимизирована для больших батчей, имеет высокую производительность обучения.

Чтобы создать среду выполнения с поддержкой GPU/TPU, необходимо изменить тип среды выполнения, выбрать GPU или TPU в раскрывающемся меню «Аппаратный ускоритель». Пример приведен ниже: `https://colab.research.google.com/notebooks/gpu.ipynb`

4.5 Выводы

Построение моделей DL — это вычислительно затратный процесс, для обучения моделей необходимо одновременно выполнить множество операций. При решении этой проблемы приходить на помощь функция изменения среды выполнения в Google Colab – выбор GPU и TPU в зависимости от задачи и требований к производительности кода. Также были обсуждены основные библиотеки для выгрузки и визуализации данных. Их описание потребуется в следующей главе при их активном использовании.

Глава 5. Вычислительные эксперименты

В данном разделе приведены результаты вычислительных экспериментов по прогнозированию цен на акции с помощью параллельного механизма multi-head в моделе Transformer.

5.1 Датасеты

Как и в прошлой работе возьмем акции трех различных компаний, крупнейшего в мире производителей и поставщика программного обеспечения IBM, инновационной компании Tesla – производителя электромобилей и решений для хранения электроэнергии, а также нефтяной компании ExxonMobil. Будем проводить наши исследования на ценах с минутным тиком с 07.05.2024 по 14.05.2024 – всего 1948 значений, и на более крупном массиве дневных цен с 29.06.2010 по 14.05.2024 (3493).

5.2 Полученные результаты

Чтобы легче было сравнивать полученные метрики качества для моделей, будем проводить все вычислительные эксперименты на 30 эпохах с 2 батчами. Отдельно проводится исследование на однодневных и минутных тиках. На Рис. 5.2 и Рис. 5.2 представлены графики полученных прогнозов для IBM. В Табл. 1 показаны основные метрики качества полученной модели при вычислениях на разном количестве GPUs для массива данных цен на акции для IBM.

Для акций Tesla построенные прогнозы изображены на Рис. 5.2 и Рис. 5.2. Полученные метрики качества отражены в Табл. 2.

Таблица 1: Метрики качества построенной модели для компании IBM

Временной тик	**Время (с)**	**gpu**	**MSE**	**MAE**	**RMSE**	**MAPE**
1 минута	897.1	1	0.0055	0.0523	0.0742	0.0003
1 день	1714.4	1	5.8031	1.8161	2.409	0.0127
1 минута	888.4	2	0.0027	0.0298	0.0369	0.0002
1 день	1697.2	2	3.0294	0.921	1.549	0.0060
1 минута	878.9	4	0.0057	0.0504	0.0763	0.0003
1 день	1686.3	4	5.926	1.743	2.321	0.0124

Таблица 2: Метрики качества построенной модели для компании Tesla

Временной тик	**Время (с)**	**gpu**	**MSE**	**MAE**	**RMSE**	**MAPE**
1 минута	905.8	1	0.0325	0.1278	0.1801	0.0007
1 день	1210.4	1	119.75	8.1401	10.9431	0.0340
1 минута	896.5	2	0.0396	0.155	0.1832	0.0007
1 день	1192.1	2	116.4473	7.3199	9.4355	0.0313
1 минута	854.7	4	0.0356	0.1957	0.1813	0.0007
1 день	1135.2	4	118.5072	8.5046	11.1091	0.0365

Таблица 3: Метрики качества построенной модели для компании Exxon Mobil

Временной тик	**Время (с)**	**gpu**	**MSE**	**MAE**	**RMSE**	**MAPE**
1 минута	818.6	1	0.0028	0.038	0.0528	0.0003
1 день	1705	1	3.2043	1.3852	1.7901	0.0148
1 минута	809.3	2	0.0056	0.0643	0.0987	0.0005
1 день	1697.1	2	6.2241	2.7594	3.239	0.0254
1 минута	783.2	4	0.0024	0.029	0.0503	0.0003
1 день	1689.9	4	3.1932	1.2952	1.6938	0.0146

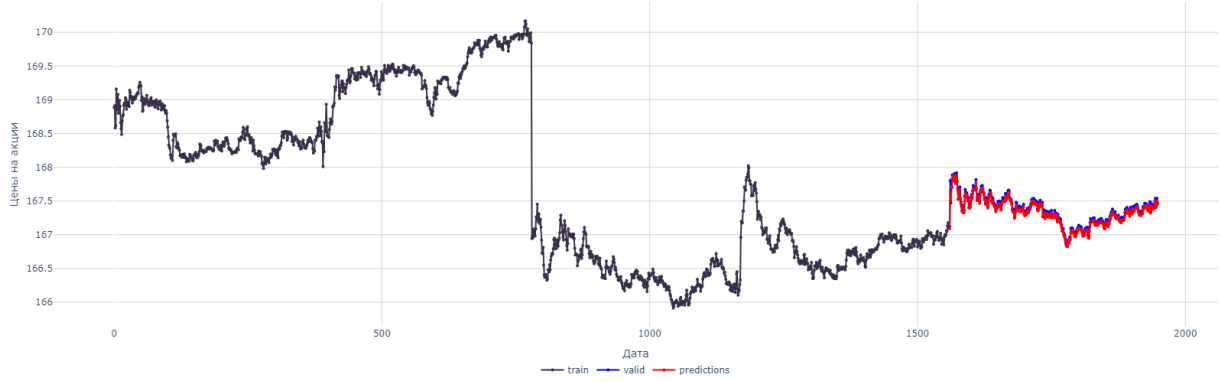


Рис. 7: Построенный прогноз на минутных данных для IBM

Рис. 8: Построенный прогноз на дневных данных для IBM

Рис. 9: Построенный прогноз на минутных данных для Tesla

Рис. 5.2 и Рис. 5.2 показывают полученный прогноз для компании Exxon Mobil. В Табл. 3 показаны основные метрики качества полученной модели.

Из данных таблиц можно сделать следующие выводы. Во-первых, как и предполагалось, уменьшается время на обучение на модели при

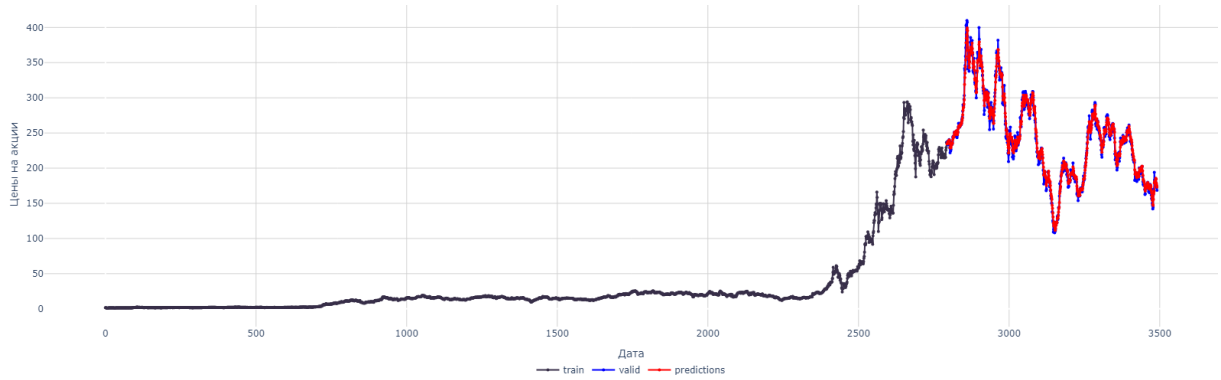


Рис. 10: Построенный прогноз на дневных данных для Tesla

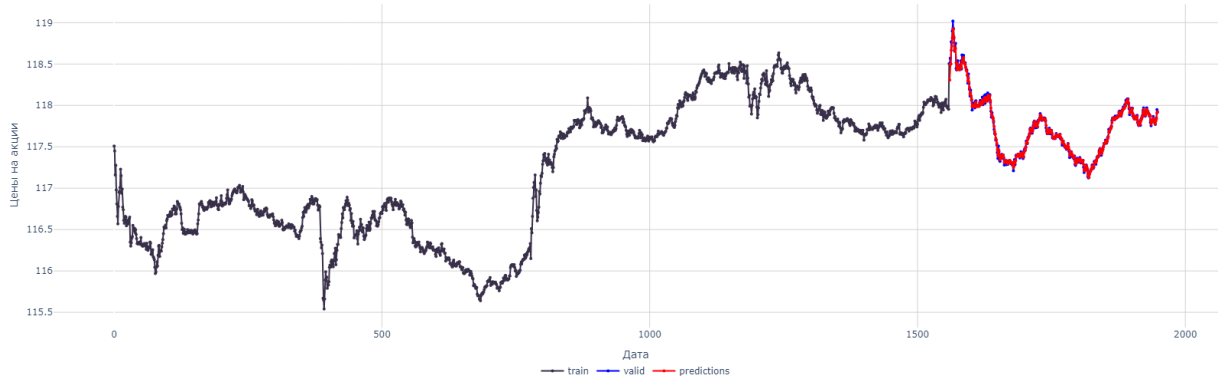


Рис. 11: Построенный прогноз на минутных данных для Exxon Mobil

Рис. 12: Построенный прогноз на дневных данных для Exxon Mobil

использовании GPU, однако эти изменения практически ничтожны по сравнению с вычислением на одном графическом процессоре. Возможно, это произошло из-за того, что модели обучались на небольшом датасете (сравнительно небольшом), с небольшим набором параметров, соответственно и с относительно "малым" количеством матричных умножений

в многоголовочном механизме Attention. Как я думаю, следовало проводить исследование на более крупном массиве данных, с большим количеством эпох, например, более 300, однако вычислительные мощности пользовательской версии Google Colab не позволяют это сделать. С другой стороны, даже несмотря на это, качество прогнозов, исходя из оценки метрик MSE, MAE достаточно велико (можно сравнить с классическими и LSTM моделями в прошлой работе). Опять же это можно объяснить наличием такого слоя в модели Transformer, как Attention.

Снижение качества модели для дневных цен кроется в большем разбросе данных, по сравнению с дневными, где практически изменения даже на 1 доллар уже значительны. У модели для компании Tesla, в целом, достаточно плохие показатели метрик. Это может быть связано с наличием большого скачка в начале тестовой выборки. Также плохой прогноз для Tesla можно объяснить достаточно бурным развитием компании за последние 3 года.

Таким образом, наши изначальные предположения о модели Transformer выполняются, правда не без оговорок. Стоит отметить, что качество прогнозирования никак не зависит от специализации компании. Результаты показали, что данный метод может быть эффективными для прогнозирования временных рядов, по сравнению с классическими моделями, и выбор того, какой из них использовать, зависит от особенностей данных и целей прогнозирования. Ссылка на GitHub: `https://github.com/vladislavzyuzin2002/Course_work`

Заключение

В данной курсовой работе центральное место занимал вопрос прогнозирования финансовых временных рядов: от определения и основных свойств временных рядов до методов их прогнозирования. В начале были упомянуты классические методы, такие как ARIMA – наиболее популярная в силу высокой интерпретируемости модель, ее модификация SARIMA, модели на основе цепей Маркова, Вейвлет-анализ. Все эти модели уступают по качеству методам, основанным на нейронных сетях. Это происходит в силу того, что нейронные сети способны находить нелинейные зависимости в данных, однако при этом возрастает вычислительная сложность. Главы 2 и 3 были посвящены рассмотрению моделей LSTM и Transformer, в них было дано подробное описание их структуры с поэтапным механизмом обработки данных в этих моделях. Основная особенность сетей Transformer состоит в том, что они используют механизм внимания, способный улавливать более сложные закономерности по сравнению с сетями LSTM. Как было замечено, нейронные сети требуют больших вычислительных мощностей. Поэтому отдельно были рассмотрены методы их оптимизации с помощью параллельных вычислений на графических процессорах.

Вычислительные эксперименты проводились на относительно больших датасетах (1948 и 3493), состоящих из исторических цен с минутными и дневными временными интервалами. Были взяты цены на акции различных по своей специализации компаний: IBM – информационные технологии, Tesla – современное автомобилестроение, Exxon Mobile – добыча полезных ископаемых. Исследование проводилось на 30 эпохах с двумя батчами. Было выявлено, что модели на основе Transformer да-

ют более точные модели прогнозирования, независимо от области специализации компании. При различном количестве GPU время меняется (GPU были использованы при вычислениях в слое multi-head механизма Attention: многоголовочная архитектура Attention позволяет каждой головке Attention обрабатываться параллельно), но не существенно, что можно объяснить небольшим набором параметров. Если бы вычисления проводились на более больших моделях, то эти изменения были бы велики.

В дальнейшем можно продолжить исследование по следующим направлениям:

- Вычисления в модели Transformer можно распределить на разных GPU и на других этапах, однако, как это сделать, более тонкий вопрос;

- Можно прогнозировать не цены, а тренд цен на акции: сети Transformer относятся к NLP, т.е. связаны с обработкой языка, более совершенные модели могут улавливать эмоции в тексте: это может быть полезно, когда хотим узнать пойдет тренд вниз или вверх (вопрос состоит лишь как закодировать временной ряд);

Резюмируя выше сказанное, результаты исследования свидетельствуют о высокой эффективности использования нейронных сетей для прогнозирования финансовых временных рядов с помощью LSTM и Transformer. Развитие области применения сетей Transformer при прогнозировании финансовых временных рядов является перспективным направлением, в том числе и из-за возможности параллельных вычислений в них.

Список литературы

[1] Ahmet Murat Ozbayoglu, Mehmet Ugur Gudelek, and Omer Berat Sezer, Deep learning for financial applications: A survey. working paper, 2019.

[2] Andreas C. Muller Sarah Guido, “Introduction to Machine Learning with Python”, 2016

[3] Ba JL, Kiros JR, Hinton GE. Layer normalization. arXiv preprint arXiv:160706450. 2016. pages 6

[4] Chaovalit, P., Gangopadhyay, A., Karabatis, G., and Chen, Z. 2011. Discrete wavelet transform-based time series analysis and mining. ACM Comput. Surv. 43, 2, Article 6 (January 2011), 37 pages. http://doi.acm.org/10.1145/1883612.1883613

[5] C. Chatfield, Analysis of Time Series An Introduction, 1995

[6] R. Cheng and Q. Li, "Modeling the momentum spillover effect for stock prediction via attribute-driven graph attention networks", Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 1, pp. 55–62, May 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16077

[7] Damir Filipovic and Amir Khalilzadeh, Machine Learning for Predicting Stock Return Volatility, Swiss Finance Institute Research Paper Series N°21-95, December 23, 2021

[8] Diederik P. Kingma, Jimmy Lei Ba; ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. https://arxiv.org/pdf/1412.6980.pdf

[9] John Duchi, Elad Hazan, Yoram Singer; Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, 12(61):21212159, 2011.

[10] Jeremy Appleyard, Tomas Kociský, Phil Blunsom; Optimizing Performance of Recurrent Neural Networks on GPUs, 2016, 1604.01946, arXiv, pages 9

[11] Fischer T, Krauss C. Deep learning with long short-term memory networks for financial market predictions. European Journal of Operational Research. 2018;270(2):654-69. pages 2

[12] Friedman, J., 2001. Greedy function approximation: A gradient boosting machine. Annals of Statistics 5, 1189–1232.

[13] Irie K, Zeyer A, Schluter R, Ney H. Language modeling with deep transformers. arXiv preprint:190504226. 2019. pages 8

[14] Kolm, P.N., Ritter, G., 2019. Dynamic replication and hedging: A reinforcement learning approach. The Journal of Financial Data Science 1, 159–171.

[15] Kwon, D., Ko, K., Vannucci, M., Reddy, A.N., Kim, S.: Wavelet methods for the detection of anomalies and their application to network traffic analysis. Quality and Reliability Engineering International 22(8), 953–969 (2006)

[16] Nicolas Vasilache, Jeff Johnson, Michaël Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A GPU performance evaluation. CoRR, abs/1412.7580, 2014

[17] Sidra Mehtab, Jaydip Sen and Abhishek Dutta, Stock Price Prediction Using Machine Learning and LSTM-Based Deep Learning Models, https://arxiv.org/ftp/arxiv/papers/2009/2009.10819.pdf,2021

[18] Meyer, Y., Salinger, D.: Wavelets and Operators:, Cambridge Studies in Advanced Mathematics, vol. 1. Cambridge University Press (1995)

[19] Sezer OB, Gudelek MU, Ozbayoglu AM. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. Applied soft computing. 2020;90:106181. pages 2

[20] Jan Schmitz, Stock predictions with state-of-the-art Transformer and Time Embeddings, Jul 6, 2020 https://towardsdatascience.com/stock-predictions-with-state-of-the-art-transformer-and-time-embeddi

[21] Tsay RS. Analysis of financial time series. John Wiley; 2005. pages 1

[22] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. Advances in neural information processing systems. 2017;30. pages 1, 5

[23] Qin Y, Song D, Chen H, Cheng W, Jiang G, Cottrell G. A dual-stage attention-based recurrent neural network for time series prediction. arXiv preprint arXiv:170402971. 2017. pages 1, 2

[24] Xu J, Wang J, Long M, et al. Autoformer: Decomposition transformers with auto-correlation for long term series forecasting. Advances in Neural Information Processing Systems. 2021;34. pages 2, 3, 7, 17

[25] Zhou T, Ma Z, Wen Q, Wang X, Sun L, Jin R. FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting. arXiv preprint arXiv:220112740. 2022. pages 2, 3

[26] Витерби А. Д., Омура Дж. К. Принципы цифровой связи и кодирования /Пер. с англ. под ред. К. Ш. Зигангирова. — М.: Радио и связь, 1982. — 536 с.

[27] Витязев В.В., Вейвлет-анализ временных рядов: Учеб. пособие. — СПб.: Изд-во С.-Петерб. ун-та, 2001. - 58 с.

[28] Нисон С. Японские свечи: Графический анализ финансовых рынков / Стив Нисон; Пер. с англ. — М.: Альпина Паблишер, 2016.

[29] https://habr.com/ru/articles/341240/

[30] https://github.com/YiSiouFeng/Python/blob/main/ARIMA%20Models%20in%20Python.md

[31] https://education.yandex.ru/handbook/ml/article/shodimost-sgd

[32] https://www.geeksforgeeks.org/sarima-seasonal-autoregressive-integrated-moving-average/