

НИУ ВШЭ Санкт-Петербург
Санкт-Петербургская школа физико-математических и компьютерных
наук
Машинное обучение и анализ данных

Белова Юлия Дмитриевна

РАЗРАБОТКА МЕТОДОВ ДОЛГОСРОЧНОЙ ПАМЯТИ ДЛЯ
БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ ПРИ ПОМОЩИ МЕТОДОВ
ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ

Научный руководитель:

Кривцов Антон Мирославович,
доктор физико-математических
наук, профессор, департамент
информатики

Свидченко Олег Анатольевич,
руководитель направления
разработки центра экспертизы и
разработки ИИ в АЦТП

Рецензент:

Трушков Алексей Олегович,
ведущий инженер ключевых
проектов, Хуавей

Санкт-Петербург
2024

Оглавление

Введение	4
Глава 1. Обзор предметной области и формулировка цели исследования	6
1.1. Предварительные сведения из обработки естественного языка	6
1.2. Долгосрочная память больших языковых моделей	10
1.3. Требования к методу	12
1.4. Мотивация к использованию обучения с подкреплением . .	13
1.5. Цель исследования и поставленные задачи	14
Глава 2. Обзор существующих решений	15
2.1. Подходы к решению проблемы ограниченного контекста: краткий обзор	15
2.2. Рекуррентные трансформеры	16
2.3. Дополнительно	19
2.4. Выводы	21
Глава 3. Описание подхода	23
3.1. Общая схема предложенного подхода	23
3.2. Обозначения	24
3.3. Архитектура дообучаемой LLM	26
3.4. Архитектура Модели Памяти	27
3.5. Сравнение с существующими подходами к рекуррентной памяти	29
3.6. Выводы	30
Глава 4. Эксперименты	32
4.1. Датасет	32
4.2. Характеристики обучения	33
4.3. Результаты	38
4.4. Выводы	45

Заключение	46
Список литературы	47

Введение

Большие языковые модели (LLM) стали неотъемлемой частью современной жизни, находя применение в самых разных областях: от создания статей и перевода текстов до помощи в написании программного кода и автоматизации общения с клиентами. Люди настолько привыкли к качеству и удобству, предлагаемому этими моделями, что уже не могут представить свою жизнь без их помощи. Однако, несмотря на все их достижения, LLM сталкиваются с серьезной проблемой, ограничивающей их потенциал: сложность работы с длинными текстами.

Основная причина этой проблемы кроется в механизме внимания, используемом в архитектуре классического трансформера [1]. В основе большинства современных LLM лежит трансформер, который применяет механизм внимания для обработки входных данных. Этот механизм имеет квадратичную сложность, что приводит к квадратичному росту вычислительных затрат по мере увеличения длины текста, который мы хотим обработать с помощью данной языковой модели. В результате, LLM часто оказываются неспособными эффективно работать с длинными документами, что особенно критично в таких областях, как медицина, наука и диалоговые системы, где требуется анализировать большие объемы текстовой информации. Для генерации текстов, которые являются логически последовательными и связными, модель должна помнить предыдущий контекст.

Параллельно с этим, в последнее десятилетие себя хорошо зарекомендовал подход обучения с подкреплением (RL). Этот метод успешно применяется для обучения роботов, игры в компьютерные игры, а также для обучения чат-ботов. Несмотря на успехи RL, возможности его применения для формирования долгосрочной памяти в больших языковых моделях остаются недостаточно исследованными. Обучение с подкреплением обладает рядом важных преимуществ, которые могут быть полезными для решения задачи долгосрочной памяти. В данном исследовании представлен новый подход для формирования долгосрочной памяти на основе обучения с подкреплением.

Данное исследование состоит из четырех глав.

Глава 1 содержит краткий обзор предметной области, постановку целей и задач настоящего исследования.

В главе 2 представлен обзор существующих решений.

В главе 3 подробно описан предложенный подход к решению проблемы долгосрочной памяти.

Глава 4 содержит набор экспериментов и их результаты.

Глава 1

Обзор предметной области и формулировка цели исследования

В данном разделе представлен краткий обзор архитектуры трансформера, обозначены её ограничения, сформулирована задача обработки длинных текстов, а также определены цели и задачи исследования

1.1. Предварительные сведения из обработки естественного языка

1.1.1. Архитектура трансформера

Изначально для решения задач естественного языка, в том числе для решения задачи языкового моделирования на длинных текстах, использовались рекуррентные нейронные сети (Recurrent Neural Network, RNN) [2], такие как LSTM [3] и GRU [2]. Из-за того, что RNN обрабатывают элементы последовательно по одному, обучение и инференс такой модели оказываются достаточно долгими. Также RNN не могут эффективно улавливать долгосрочные зависимости, если тексты достаточно длинные. Поэтому была предложена архитектура, решающая эту и некоторые другие проблемы RNN, называемая трансформером.

Классический трансформер [1] представляет собой модель, состоящую из энкодера и декодера, каждый из которых представляет собой стек из L идентичных блоков. Блоки трансформера состоят из механизма многоголового самовнимания (multi-head self-attention mechanism), сетей прямого распространения (feed-forward network), слоев нормализации (layer norm) и остаточных связей (residual connection). Общая архитектура классического трансформера представлена на рисунке 1.1. Более подробно мы рассмотрим только механизм внимания, об остальных частях архитектуры можно почитать в [1].

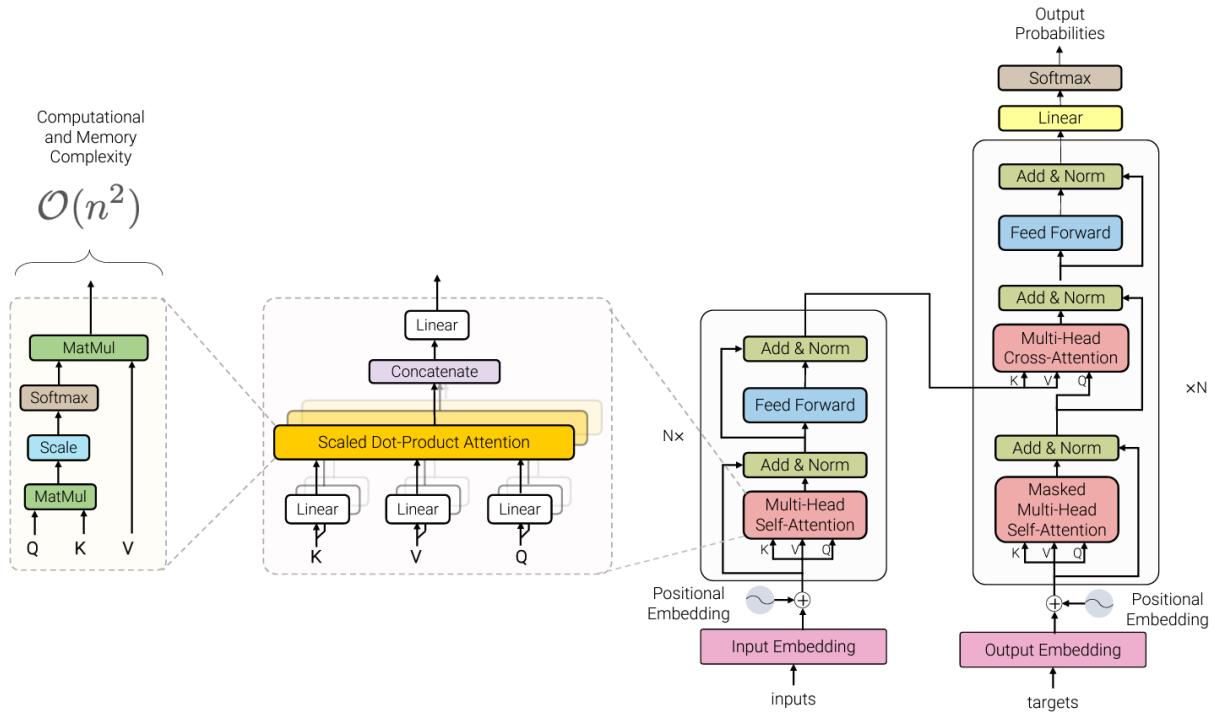


Рис. 1.1. Архитектура классического трансформера

Основная идея механизма многоголового внимания заключается в том, что каждый элемент входной последовательности учится собирать полезную для него информацию с остальных элементов последовательности. Операция внимания для отдельной головы определяется следующим образом:

$$A_h = \text{Softmax} \left(\frac{Q_h K_h^T}{\sqrt{d_k}} \right) V_h, \quad (1.1)$$

где X - матрица векторных представлений входной последовательности размером $\mathbb{R}^{N \times d}$, $Q_h = XW_q$, $K_h = XW_k$ и $V_h = XW_v$ являются линейными преобразованиями, применяемые к скрытым состояниям входной последовательности, W_q , W_k , $W_v \in \mathbb{R}^{d \times d_H}$ - матрицы весов для проекций запросов, ключей и значений. Эти матрицы проецируют входной тензор X в выходной тензор размерности d , а N_H - количество голов в механизме внимания. Выходы голов $A_1 \dots A_{N_H}$ конкатенируются и далее передаются в полно связную сеть (FFN).

Матрица внимания $A = Q^T$ получается путем скалярного произведения между каждым элементом в запросе (Q) и ключе (K), так токены учатся извлекать информацию друг от друга.

1.1.2. Ограничения трансформеров

Вычислительная сложность механизма внимания. Вычислительные затраты работы трансформера определяются некоторыми факторами. Во-первых, объем памяти и вычислительная сложность, необходимые для вычисления матрицы внимания, являются квадратичными по отношению к длине входной последовательности N . В частности, операция умножения матриц QK^T потребляет $N \times N$ памяти и времени. Это ограничивает использование моделей с механизмом внимания в приложениях, где требуется обработка длинных текстов. Ограничения по памяти чаще всего применимы к обучению (из-за обновлений градиента) и, как правило, в меньшей степени влияют на инференс. Существенная часть вычислительных затрат также связана с двумя слоями полносвязной сети (FFN) в каждом блоке трансформера, сложность FFN линейна по отношению к длине последовательности, но, как правило, всё равно высока (см. таблицу 1.1).

Модуль	Вычислительная сложность	Число параметров
Attention	$\mathcal{O}(N^2d)$	$4d^2$
FFN	$\mathcal{O}(Nd^2)$	$8d^2$

Таблица 1.1. Вычислительная сложность и количество параметров для механизма внимания и полносвязной сети

Ограничение на максимальную длину входа. Во время обучения модели трансформерного типа инженерам обычно требуется определить важный гиперпараметр - максимальную длину входной последовательности (L_{max}), которая обозначает верхний предел длины последовательности для любого обучающего примера в батче. Обычно его значение устанавливается на уровне одной, двух или четырех тысяч токенов, в зависимости от доступных вычислительных ресурсов, чтобы избежать проблемы недостатка памяти на GPU. Так, BERT [4] может обрабатывать только 512 токенов за раз. Во время инференса поставщики больших языковых моделей (LLM) должны либо ограничивать длину пользовательских запросов

сов, либо автоматически обрезать их, чтобы они соответствовали предопределенному L_{max} .

Стоит отметить, что по своей природе трансформер не требует таких ограничений, так как все обучаемые параметры зависят только от размерностей эмбеддингов, или скрытых состояний. Таким образом, теоретически, трансформер может обрабатывать последовательности любой длины. Однако, к сожалению, современные языковые модели демонстрируют заметное ухудшение производительности при обработке входных последовательностей, превышающих L_{max} , что часто приводит к повторяющимся и неправдоподобным выводам. Это происходит за счет того, что языковой модели сложно адаптироваться к новой длине входа, на которой она еще не обучалась.

Отсутствие явного механизма памяти. Трансформеры не имеют явного механизма памяти и опираются исключительно на текущие скрытые состояния, получаемые с помощью механизма внимания для текущей входной последовательности. Это означает, что после завершения одного запроса трансформер не сохраняет и не вспоминает предыдущие состояния или последовательности в последующих запросах. Таким образом, трансформер обладает лишь оперативной контекстной памятью во время каждого вызова, в отличие от встроенных механизмов памяти, например, как в модели LSTM. Такое отсутствие рекуррентного механизма дает вычислительные преимущества в отношении параллелизма, но создает трудности в задачах, где важно сохранять долгосрочную память, например, в чат-ботах, при обработке длинных биологических последовательностей и т.д.

1.1.3. Использование блоков трансформера

Способы использования модели типа трансформер зависят от целевого применения. Трансформер можно использовать в трех основных режимах:

1. Полная модель энкодер-декодер, как было представлено на рисунке 1.1. Такая архитектура обычно используется на задаче машинного

перевода.

2. Только энкодер. Выходы энкодера используются в качестве суммаризированного, или скрытого, представления входной последовательности. Обычно такая архитектура используется на задачах классификации. Примером такой модели служит BERT, где используется специальный токен $[CLS]$, добавляемый в качестве префикса к исходной последовательности.
3. Только декодер. Используется на задаче генерации текста и обучается с целью предсказания следующего токена. В такой архитектуре необходимо использовать каузальную маску, которая предотвращает подсматривание токенов в будущее.

В моем исследовании я сосредоточусь на моделях декодерного типа, поскольку такой режим использования трансформера наиболее подходит для решения задачи языкового моделирования на длинных последовательностях.

1.2. Долгосрочная память больших языковых моделей

1.2.1. Формальное определение языкового моделирования длинных последовательностей

Увеличение числа параметров в языковой модели и объема данных для её обучения обычно приводит к повышению точности предсказаний. Это объясняется тем, что более крупные модели с большим количеством параметров способны улавливать более сложные зависимости и паттерны в данных, а обучение на большем объеме данных способствует лучшей обобщающей способности модели. Однако, как уже упоминалось, вычислительная сложность механизма внимания в трансформерах ограничивает их масштабируемость.

Таким образом, встает вопрос: как эффективно (и в целом) обрабатывать длинные последовательности?

Для начала дадим формальное определение языковому моделированию длинных последовательностей. Пусть длинный текст представляет собой последовательность токенов $\mathcal{X} = (x_1, \dots, x_n)$, которая может содержать тысячи и более токенов, в отличие от коротких или обычных текстов, которые могут быть непосредственно обработаны трансформером. Отсюда следует, что необходимо использовать некоторую функцию предварительной обработки входной последовательности $g(\cdot)$ для преобразования ее в более короткую или разделения на сегменты. Далее применяется архитектура типа трансформер \mathcal{M} для захвата контекстуальной информации из входных данных \mathcal{X} и построения некоторого отображения f входной последовательности \mathcal{X} в ожидаемый выход \mathcal{Y} :

$$\mathcal{Y} = f(g(\mathcal{X}); \mathcal{M}) \quad (1.2)$$

В данном исследовании под \mathcal{Y} будет подразумеваться последовательность токенов, хотя в некоторых приложениях \mathcal{Y} может быть и меткой, например, на задаче классификации длинного текста. Для получения целевой последовательности модели \mathcal{M} необходимо уметь захватывать ключевую семантическую информацию длинного текста и строить логическую связь между различными его частями.

1.2.2. Способы предобработки длинных входных последовательностей

Существует несколько способов преобразования входной последовательности для возможности обрабатывать ее языковой моделью с ограничением на длину входа:

- Обрезание входной последовательности.** Самый простой вариант — обрезать длинный входной текст до последовательности в пределах предопределенной максимальной длины. Более формально: $g(\mathcal{X}) = (x_1, \dots, x_t), t \leq N$. Очевидно, такой подход имеет существенный недостаток — снижение производительности из-за потери информации, ведь в длинных текстах важная информация распределяется равномерно по всему тексту.

2. Разделение текста на сегменты, или фрагменты. В данном способе входной текст \mathcal{X} делится на серию фрагментов, где каждый фрагмент представляет собой последовательность токенов в пределах максимальной длины модели, и объединение этих фрагментов эквивалентно исходному тексту:

$$g(\mathcal{X}) = \{s_1, s_2 \dots s_l\}, \mathcal{X} = s_1 \cup s_2 \cup \dots \cup s_l \quad (1.3)$$

Обычно входная последовательность разделяется на непересекающиеся фрагменты с фиксированным шагом, например, равным максимальной длине входа модели. Но также существуют подходы, в которых входная последовательность разбивается на пересекающиеся фрагменты [5] или, если текст имеет определенную структуру, фрагменты соответствуют частям этой структуры. В отличие от обрезания текста, разделение текста на фрагменты позволяет использовать всю длинную последовательность, избегая потери важной информации. Тем не менее, поле восприятия (receptive field) моделей ограничено сегментом, что приводит к разрыву долгосрочных зависимостей, охватывающих несколько сегментов. Поэтому возникает потребность в определении такой функции \mathcal{F} , которая бы связывала эти сегменты и каким-то способом передавала между ними информацию.

В моем исследовании я сосредоточусь на подходе сегментирования входной последовательности и разработке функции \mathcal{F} , которая позволит эффективно передавать контекстную информацию между сегментами. Иными словами, требуется создать долгосрочную память в языковой модели, к которой она могла бы обращаться и которую могла бы заполнять по мере обработки сегментированной последовательности.

1.3. Требования к методу

В рамках данного исследования выдвигаются следующие требования к разрабатываемому методу долгосрочной памяти:

- **Интеграция с существующими моделями.** Метод должен быть совместим с существующими языковыми моделями и не требовать значительных затрат на дообучение.
- **Вычислительная эффективность.** Метод должен быть эффективным относительно количества вычислений, а также потребляемой памяти, чтобы оставаться практичным для реальных применений.
- **Гибкость по длине последовательностей.** Метод должен эффективно адаптироваться к различным длинам последовательностей, обеспечивая обработку как коротких, так и очень длинных текстов.

Метод долгосрочной памяти, удовлетворяющий всем выдвинутым выше требованиям, может быть легко встроен в существующие языковые модели. Соблюдение этих требований может обеспечить высокое качество предсказаний на длинных текстах, вычислительную эффективность и практическую применимость.

1.4. Мотивация к использованию обучения с подкреплением

На сегодняшний день обучение с подкреплением (Reinforcement learning, RL) показало значительные успехи во многих областях, в том числе в обработке естественного языка. Так, например, при обучении GPT-3 [6], разработанной OpenAI, используется обучение с подкреплением для улучшения качества генерируемого текста на основе фидбэка (Reinforcement Learning from Human Feedback, RLHF).

Ниже представлены основные мотивы для использования RL на задаче создания долгосрочной памяти у больших языковых моделей:

1. Долгосрочное планирование. Обучение с подкреплением позволяет моделям учитывать не только немедленные, но и будущие результаты своих действий. В нашей задаче это может помочь агенту формиро-

вать воспоминания и запоминать необходимый контекст, что важно для обработки длинных текстов.

2. Отсутствие необходимости вычисления градиентов по функции награды, получаемой в результате взаимодействия со средой, что может упростить и ускорить процесс обучения.
3. Способность алгоритмов RL к обобщению на сценарии длительного взаимодействия при обучении на эпизодах малой длительности.
4. Возможность переиспользования данных (буфера опыта) в рамках одной итерации обучения. Это может позволить более эффективно использовать собранные данные и ускорить процесс обучения.

1.5. Цель исследования и поставленные задачи

Цель настоящего исследования заключается в разработке метода долгосрочной памяти для больших языковых моделей с использованием обучения с подкреплением. Для достижения данной цели необходимо выполнить следующие задачи:

- Создать датасет, содержащий длинные тексты с возможностью выделения контекста и основного текста, что позволит эффективно обучать модели на последовательностях значительной длины.
- Разработать и формализовать подход к внедрению долгосрочной памяти в большие языковые модели, основанный на принципах обучения с подкреплением.
- Реализовать предложенный метод и провести его обучение и настройку гиперпараметров.
- Провести сравнительный анализ качества работы разработанного метода относительно выбранного бейзлайна, используя соответствующие метрики.

Глава 2

Обзор существующих решений

В данной главе я рассмотрю существующие решения для расширения контекста больших языковых моделей и формирования долгосрочной памяти для них. Поскольку в этой области было проведено множество исследований, я сосредоточусь только на тех, которые относятся к моей задаче — созданию памяти на основе рекуррентного подхода к сегментированным последовательностям. Кроме того, будут описаны подходы, не связанные напрямую с долгосрочной памятью больших языковых моделей, но использующиеся для решения поставленной задачи.

2.1. Подходы к решению проблемы ограниченного контекста: краткий обзор

Подходы к решению проблемы ограниченного контекста языковых моделей можно классифицировать по следующим категориям:

1. Эффективное внимание. Этот класс методов направлен на оптимизацию механизма внимания, чтобы уменьшить его квадратичную вычислительную сложность. Такие методы позволяют расширить длину контекста больших языковых моделей, увеличивая гиперпараметр L_{max} . Сюда относятся подходы разреженного внимания [7], [8], [9], иерархического внимания [10], аппроксимации механизма внимания [11], [12] и IO-aware attention [13].
2. Экстраполирующие позиционные кодировки. Данные методы улучшают экстраполяционные свойства существующих схем позиционного кодирования, что позволяет моделям эффективно работать с последовательностями, длина которых превышает ту, на которой они были обучены. О них можно подробнее узнать из следующих исследований: [14], [15].

3. В эту категорию входят рекуррентные подходы к памяти ([16], [17]), а также методы с использованием внешнего банка памяти и определенных критериев извлечения информации из него, такие как Retrieval-Augmented Generation (RAG): [18], [19].

Этот список не является исчерпывающим; существует множество других подходов, которые также заслуживают внимания, но не будут рассматриваться в данной работе.

2.2. Рекуррентные трансформеры

Рекуррентные трансформеры являются одним из основных подходов к решению проблемы ограниченной длины контекста языковых моделей. В отличие от подходов, в которых исследователи упрощают структуру механизма внимания, рекуррентный трансформер сохраняет его в полном объеме. Обычно длинная последовательность разбивается на множество фрагментов, как было описано в разделе 1.2.

В данной области проведено немало исследований. Я сосредоточусь на тех из них, которые оказали наибольшее влияние на разработку моего подхода и послужили основным источником вдохновения.

2.2.1. Обозначения

Пусть S - количество сегментов, на которые разбивается входная последовательность \mathcal{X} . Пусть M_{et} - память, в которую предполагается сохранять прыдущий контекст. Пусть каждый сегмент s_i имеет длину N , а модели состоят из l блоков трансформера. Оператор $[o]$ обозначает конкатенацию вдоль размерности ‘длина сегмента’. Использование символа $\hat{\cdot}$ будет означать отсутствие градиентов, или отделение от графа вычислений.

2.2.2. Transformer XL

Transformer-XL [16] впервые предлагает механизм рекуррентной памяти для трансформеров. Как показано в уравнении 2.1, скрытые состояния,

вычисленные для m предыдущих сегментов сохраняются для каждого слоя l_j модели. Вход слоя l_i состоит из последних m сохраненных скрытых состояний слоя l_{i-1} и выходных данных слоя l_{i-1} для текущего сегмента t .

$$\text{Mem}_{XL}(l, t, m) = \left[\hat{O}_{t-m}^{l-1} \circ \dots \circ \hat{O}_{t-1}^{l-1} \right] \quad (2.1)$$

Авторы статьи обучали модель с нуля, возможности применения подхода к уже предобученной модели не были исследованы. Размер памяти Mem составляет $\mathcal{O}(d_{model} \cdot N \cdot l \cdot m)$, а размер контекста - $\mathcal{O}(N \cdot l \cdot m)$ токенов.

2.2.3. Compressive Transformer

В модели Transformer-XL скрытые представления удаленных фрагментов s_{t-i} , где $i > m$, будут отброшены, что ограничивает длину контекста, который может быть эффективно использован. Чтобы решить эту проблему, Compressive Transformer [20] сжимает прошлые скрытые представления с использованием функции сжатия f_c с коэффициентом сжатия c :

$$\text{Mem}_{Comp}(l, t, m_1, m_2, c) = [\text{Mem}_{f_c} \circ \text{Mem}_{XL}(n, t, m_1)], \quad (2.2)$$

$$\text{где } \text{Mem}_{f_c} = \left[f_c(\hat{O}_{t-m_1-m_2}^{l-1}) \circ \dots \circ f_c(\hat{O}_{t-m_1-1}^{l-1}) \right] \quad (2.3)$$

Так, для m_1 сегмента хранятся все скрытые состояния, а к m_2 наиболее старым воспоминаниям применяется некоторая функция сжатия по размерности *длина*. Авторы статьи обучали модель с нуля, возможности применения подхода к уже предобученной модели не были исследованы. Размер потребляемой памяти составляет $\mathcal{O}(d_{model} \times N \times l \times (m_1 + cm_2))$, а длина контекста - $\mathcal{O}(N \times l \times (m_1 + cm_2))$.

2.2.4. Memorizing Transformer

В статье Memorizing Transformers [21] авторы предложили сохранять пары ключ-значение (K, V) из одного из последних слоев L , вместо сохранения скрытых состояний. Для механизма внимания используется не весь набор сохраненных пар, а лишь их часть, отобранная с помощью алгоритма

k -ближайших соседей, который выбирает k наиболее похожих на векторы запросов из текущего сегмента s_t :

$$(K_t^L, V_t^L) = [\text{retr}(Q_t^L, m, k) \circ (K_t^L, V_t^L)], \quad (2.4)$$

где

$$\text{retr}(Q_t^L, m, k) = \text{kNN} \left[Q_t^L, \{(K_{t-s}^L, V_{t-s}^L)\}_{s=1}^m \right] \quad (2.5)$$

В статье авторам удалось применить подход к предобученной большой языковой модели, дообучая всю модель полностью. Размер потребляемой памяти составляет $\mathcal{O}(d_{model} \times N \times M)$, где M - количество сохраняемых пар, а длина контекста - $\mathcal{O}(N \times M)$. Преимуществом данного подхода по сравнению с предыдущими является появление механизма эффективного извлечения информации из памяти (kNN), что позволяет находить наиболее релевантную и важную информацию в памяти.

2.2.5. Recurrent Memory Transformer

Recurrent Memory Transformer [17] внедряет в трансформер механизм памяти путем добавления специальных токенов памяти [mem], которые добавляются в начало и конец каждого сегмента, как показано в формуле. После обработки каждого сегмента токены записи принимаются в качестве токенов [mem] для следующего сегмента.

$$\tilde{O}_t^L = \text{Transformer}(\tilde{s}_t^0), \tilde{s}_t^0 = [s_t^{\text{mem}} \circ s_t^0 \circ s_t^{\text{mem}}], \quad (2.6)$$

где

$$s_t^{\text{mem}} = O_{t-1}^{\text{write}}, [O_{t-1}^{\text{read}} \circ O_{t-1}^L \circ O_{t-1}^{\text{write}}] = \tilde{O}_{t-1}^L \quad (2.7)$$

Такой подход требует дообучения всей модели, чтобы параметры каждого слоя адаптировались под работу с новыми токенами памяти, в статье авторы пробовали применять подход к уже предобученным моделям. Размер потребляемой памяти составляет $\mathcal{O}(d_{model} \times p \times l \times 2)$, а длина контекста - $\mathcal{O}(N \times S)$. В формуле для объема памяти все еще присутствует фактор l - количество слоев трансформера, так как для каждого слоя необходимо хранить скрытые представления [mem] токенов для обратного распространения ошибки на шаге оптимизации.

2.2.6. AutoCompressors

Трансформер AutoCompressors [22] сжимает векторные представления сегментов в набор векторов, называемых векторами суммаризации. Подход построен на основе RMT, но с некоторыми изменениями: векторы суммаризации накапливаются и добавляются в начало каждого последующего сегмента (см. рис. 2.1). Также было предложено варьировать длины сегментов, на которые разбивается входная последовательность.

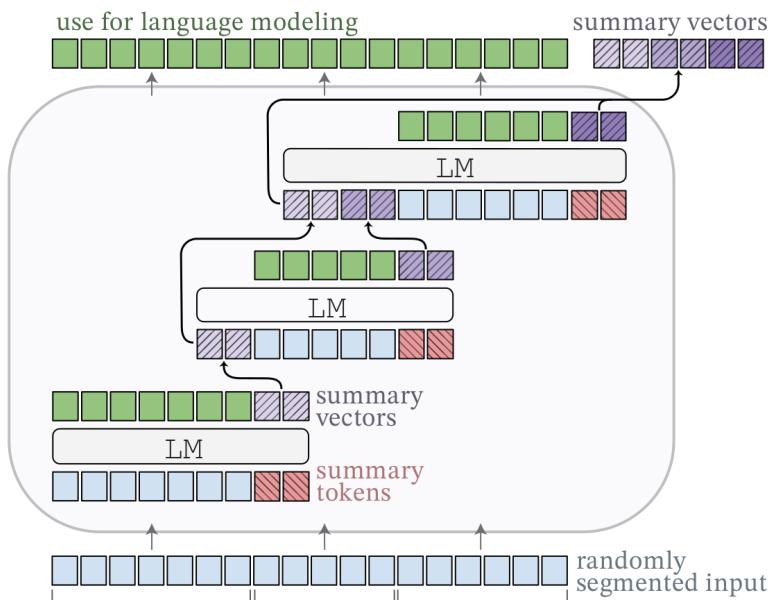


Рис. 2.1. Схема обработки длинного текста AutoCompressors

AutoCompressors применяется к предобученным моделям, размер потребляемой памяти составляет $\mathcal{O}(d_{model} \times p \times (m+1) \times l)$, где m - число шагов аккумулирования векторов суммаризации; длина контекста - $\mathcal{O}(N \times S)$.

2.3. Дополнительно

В данном параграфе приводятся описания подходов или моделей, которые я буду использовать для решения задачи.

2.3.1. REINFORCE

Алгоритм REINFORCE [23] является методом градиентной оптимизации, используемым в обучении с подкреплением. Основная идея заключается в обновлении параметров политики, чтобы максимизировать ожидаемое вознаграждение. Это достигается путем увеличения вероятности действий, приводящих к высоким вознаграждениям. Алгоритм использует метод Монте-Карло для оценки градиентов и обновления политики на основе этой оценки. Обновление параметров политики θ выполняется по следующей формуле:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R_t \quad (2.8)$$

где:

- α — скорость обучения,
- T — длина эпизода,
- $\pi_{\theta}(a_t | s_t)$ — вероятность выбора действия a_t в состоянии s_t при параметрах θ ,
- R_t — суммарная ожидаемая награда, полученная на шаге t .

2.3.2. LoRA

LoRA (Low-Rank Adaptation) — это метод адаптации больших языковых моделей, направленный на снижение количества обучаемых параметров при сохранении качества решения задачи. Метод был представлен в исследовании [24]. Основная идея метода заключается в следующем: вместо полной донастройки всех параметров модели, веса замораживаются, и вводятся дополнительные матрицы низкого ранга, которые обучаются (см. рисунок 2.2).

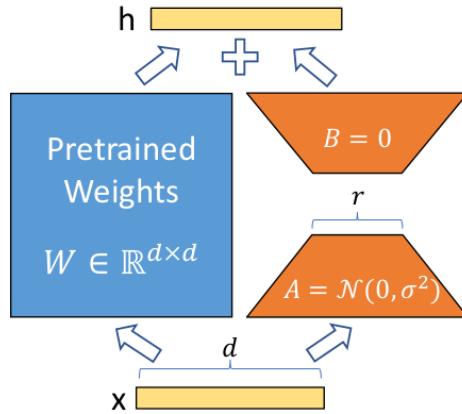


Рис. 2.2. Схема низкоранговой адаптации предобученной модели (LoRA)

2.3.3. Дивергенция Кульбака-Лейблера

Дивергенция Кульбака-Лейблера (Kullback–Leibler divergence, KLD) — это мера различия между двумя вероятностными распределениями. Для двух распределений P и Q KLD определяется как:

$$D_{KL}(P\|Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

KLD измеряет, насколько одно распределение Q отклоняется от другого распределения P .

2.4. Выводы

В данной главе я подробно рассмотрела подходы к созданию долгосрочной памяти для больших языковых моделей, объединяемые под названием ‘рекуррентные трансформеры’. Однако эти подходы требуют полного дообучения моделей (с нуля или на основе предобученных моделей), в которые интегрируется механизм внимания.

Я хотела бы предложить альтернативный метод, который мог бы быть внедрен при ограниченных ресурсах и не требовал бы значительных вычислительных затрат при обучении. Также остается малоисследованной возможность использования обучения с подкреплением для развития рекуррентного механизма памяти, что представляет собой перспективное на-

правление для моего исследования. Также важно найти подход, который не уступал бы рассмотренным методам по размеру памяти.

Глава 3

Описание подхода

3.1. Общая схема предложенного подхода

Пусть модель *LLM-LTM* строится на основе предобученной модели по типу *GPT3*. Для расширения контекста языковой модели предлагается подход, при котором агент, представленный в виде модели памяти *Memory Model*, обучается генерировать и сохранять новые воспоминания на основе верхнеуровневого представления текста, созданного большой языковой моделью *LLM-LTM*. Эти воспоминания затем сохраняются в матрице памяти *Memory*, которую языковая модель в следующих блоках может использовать как дополнительный контекст при генерации текста. Общая схема обучения представлена на рисунке 3.1. В следующих параграфах я более подробно опишу предложенные модели.

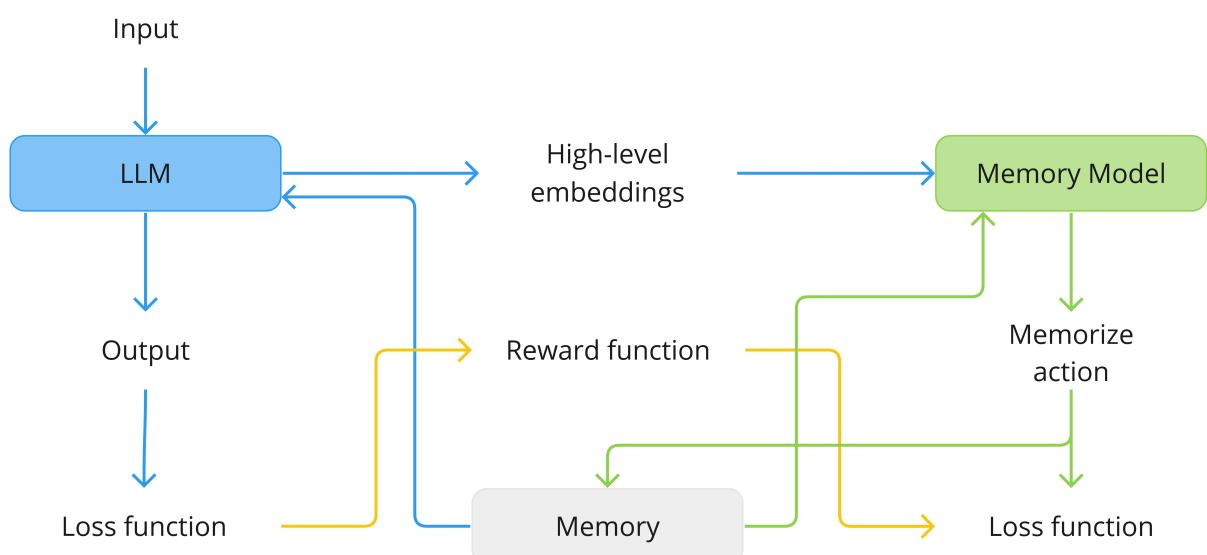


Рис. 3.1. Схема взаимодействия языковой модели *LLM-LTM* и модели памяти *Memory Model*

Значимым преимуществом такого подхода является то, что дообучаются лишь несколько последних слоев *LLM-LTM* и относительно неболь-

шая модель памяти, что потребует значительно меньшего времени и ресурсов, чем дообучение большой языковой модели полностью. Также предполагается, что такой подход позволит расширить горизонт памяти языковой модели, так как, интуитивно, такой подход эквивалентен суммаризации предшествующего контекста в несколько информативных векторов.

Общий пайплайн обучения представляет из себя итеративное поочередное обучение *LLM-LTM* и модели памяти:

- Сбор данных при помощи *LLM-LTM* и актуальной модели памяти.
- Дообучение *LLM-LTM*.
- Сбор данных при помощи *LLM-LTM* и актуальной модели памяти.
- Обучение модели памяти.

3.2. Обозначения

Введем некоторые обозначения, которые позволяют нам более формально определить предложенный подход.

Средой будем называть языковую модель *LLM-LTM*, а агентом — некую сущность, чьей политикой, или правилом, на основе которого принимается решение, будет Memory Model. Эпизод взаимодействия со средой определяется как последовательная обработка сегментированного текста X , где терминальным состоянием эпизода является завершение обработки длинного текста.

Сегмент текста определим как s_t , где $t \in [0, l]$. Пусть N - количество токенов в одном сегменте, а d - размерность скрытых состояний языковой модели.

Состоянием среды S_t будем называть совокупность двух сущностей: матрицы памяти $M_t \in \mathcal{R}^{m_1 \times m_2}$ и верхнеуровневых эмбеддингов E_t , получаемых из среды:

$$S_t = (M_t; E_t) \tag{3.1}$$

Память имеет m_1 векторов размерности m_2 , а верхнеуровневые эмбеддинги представляют собой матрицу размерности $N \times d$. За начальное состояние

S_0 мы будем принимать память, заполненную нулями, и эмбеддинги, полученные для сегмента под номером $t = 0$.

Агент, в ответ на состояние из среды, выдает действие $A_t = (n_i, a_i)$. Действие состоит из двух компонент:

- n_i - выбранная позиция в памяти,
- a_i - вектор нового воспоминания, сгенерированный из нормального распределения для выбранной позиции i .

Обновленная матрица памяти M_{t+1} определяется как

$$M_{t+1} = \begin{cases} M_t[j], & \text{если } j \neq n_i \\ a_i, & \text{если } j = n_i \end{cases} \quad (3.2)$$

В ответ на действие агент получает награду $R_t \in \mathbb{R}$ и новое состояние S_{t+1} . Новое состояние включает обновленную память и эмбеддинги для следующего сегмента:

$$S_{t+1} = (M_{t+1}; E_{t+1}) \quad (3.3)$$

Награда агента R_t определяется как минус значение функции потерь языковой модели на следующем шаге. То есть, чтобы оценить, насколько полезной оказалась сохраненная агентом информация, необходимо оценить качество генерации LLM на следующем сегменте и с обновленной памятью.

Языковая модель $LLM-LTM$ обучается с использованием функции потерь в виде кросс-энтропии. Кросс-энтропия измеряет различие между предсказанным распределением вероятностей и истинным распределением. Формула кросс-энтропии выглядит следующим образом:

$$\mathcal{L}(P_{pred}, P_{true}) = - \sum_{i=1}^V P_{true}(t_i) \log P_{pred}(t_i) \quad (3.4)$$

где:

1. V - размер словаря выбранного токенизатора.
2. $P_{true}(t_i)$ - истинная вероятность токена t_i .
3. $P_{pred}(t_i)$ - предсказанная вероятность токена t_i .

3.3. Архитектура дообучаемой LLM

Пусть $LLM_{original}$ - исходная предобученная языковая модель декодерного типа, состоящая из n блоков. Как мы уже отметили выше, агент будет взаимодействовать со средой, называемой $LLM-LTM$ (см. рисунок 3.2).

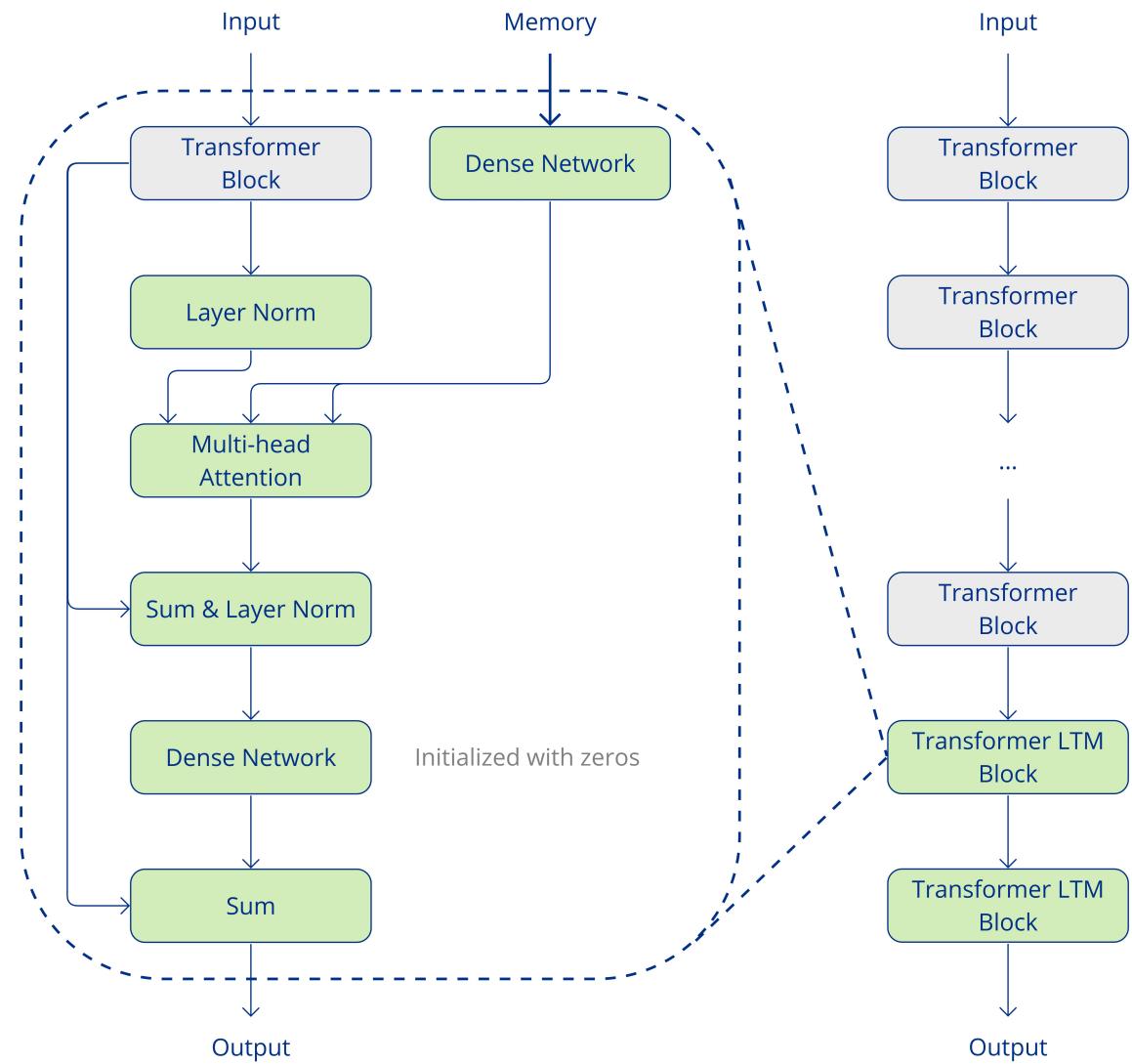


Рис. 3.2. Архитектура дообучаемой языковой модели $LLM-LTM$ с добавлением новых слоев внимания, отвечающих за взаимодействие с памятью

$LLM-LTM$ состоит из следующих компонентов:

- Зафиксированные первые k блоков исходной модели (на рисунке от-

мечены серым цветом):

$$LLM_{fixed} = \bigcup_{i \leq k} \text{Transformer Block} \quad (3.5)$$

Эти слои не изменяются в процессе обучения и используются для получения верхнеуровневого представления текста, формально это выходы k -го блока модели:

$$E_i = LLM_{fixed}(s_i), \quad (3.6)$$

где s_i - i -й сегмент, полученный в результате предобработки длинной последовательности, а E_i - верхнеуровневое представление этого сегмента.

- Дообучаемые блоки исходной модели с архитектурными изменениями (на рисунке отмечены зеленым цветом):

$$LLM_{LTM} = \bigcup \text{Transformer LTM Block} \quad (3.7)$$

В каждом из этих блоков матрица памяти M используется в механизме многоголового внимания (Multi-Head Attention) в качестве ключей и значений, а затем полученные скрытые состояния обрабатываются полносвязной сетью (Dense Network).

Важно отметить, что в начале обучения полносвязные слои в дообучаемых блоках LLM_{LTM} инициализируются нулями. Это обеспечивает тождественность сети оригинальной модели $LLM_{original}$ на начальном этапе обучения. Ожидается, что данный метод будет демонстрировать высокую сходимость, аналогично подходу ControlNet для Stable Diffusion.

3.4. Архитектура Модели Памяти

Архитектура *Memory Model*, используемой в качестве политики агента, представлена на рисунке 3.3.

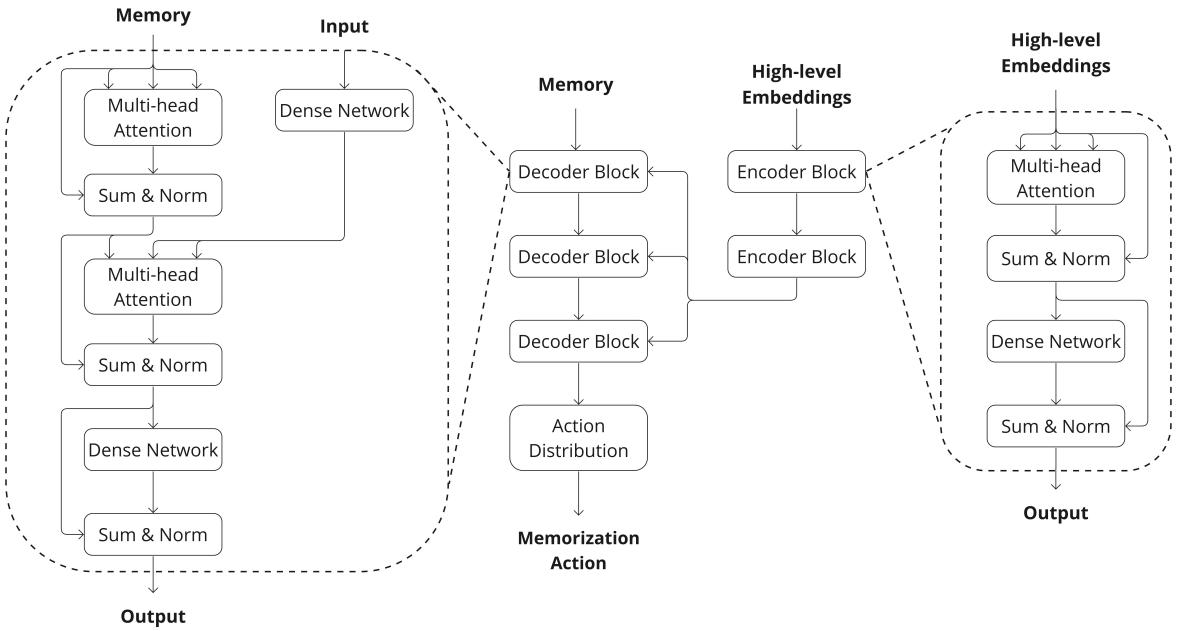


Рис. 3.3. Архитектура модели памяти

Модель памяти, аналогично классическому трансформеру, состоит из двух частей: энкодера и декодера. Задача энкодера — обрабатывать верхнеуровневые представления текста, полученные из LLM_{fixed} . Эти представления затем поступают в блоки декодера и участвуют в многоголовом внимании. Первый блок декодера принимает на вход матрицу памяти M и выходы последнего блока энкодера, преобразовывая их в размерность, соответствующую размерности векторов памяти. Далее он выполняет три ключевые операции:

1. Self-attention. Векторы памяти обмениваются информацией между собой, что позволяет им интегрировать контекст друг друга.
2. Операция многоголового кросс-внимания (multi-head cross-attention). Векторы памяти используют преобразованные выходы энкодера в качестве ключей и значений для кросс-внимания, что позволяет им интегрировать информацию из верхнеуровневых представлений текста.
3. Обработка полученной информации в полно связной сети (dense network).

Выходы последнего декодерного блока передаются в блок, называемый Action Distribution (см. рисунок 3.4).

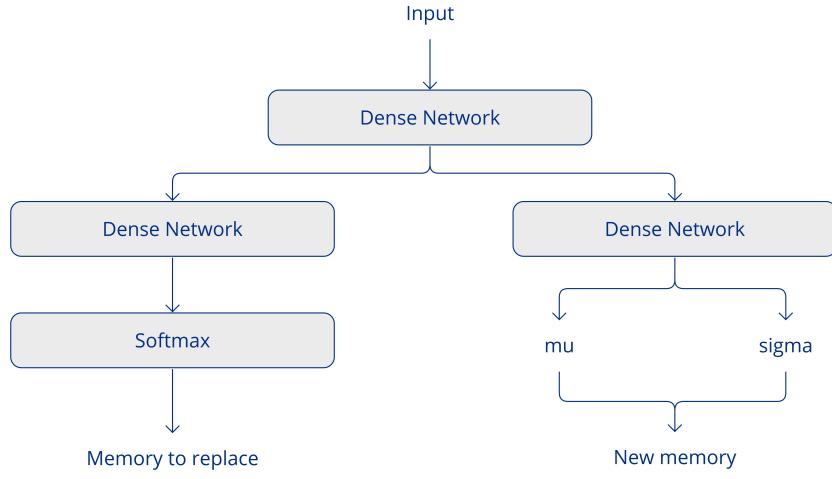


Рис. 3.4. Архитектура блока Action Distribution

В этот блок поступает матрица памяти, и выходами блока являются несколько распределений:

1. Распределение вероятностей выбора позиции в памяти $p(i)$:

$$\sum_{i=0}^{m_1-1} p_i = 1, \quad (3.8)$$

где p_i - вероятность выбора i -й позиции в памяти.

2. Параметры нормального распределения - среднее значение μ и стандартное отклонение σ для каждого элемента вектора памяти:

$$a_i \sim \mathcal{N}(\mu_i, \sigma_i^2) \quad (3.9)$$

3.5. Сравнение с существующими подходами к рекуррентной памяти

В рамках данного параграфа я хочу провести сравнение с существующими подходами, использующие механизмы рекуррентной памяти, по части используемой памяти и длины контекстного окна. Основываясь на сравнении, приведенном в недавнем исследовании [25], я добавлю в него характеристики нашего подхода:

Модель	Размер памяти	Длина контекста	Обращение к памяти	Ссылка
Transformer-XL	$(d_{key} + d_{value}) \times H \times N \times l$	$N \times l$	Скалярное внимание	[16]
Compressive Transformer	$d_{model} \times (c + N) \times l$	$(c \times r + N) \times l$	Скалярное внимание	[20]
Memorizing Transformers	$(d_{key} + d_{value}) \times H \times N \times S$	$N \times S$	kNN + Скалярное внимание	[21]
RMT	$d_{model} \times p \times l \times 2$	$N \times S$	Софт-промпт	[17]
AutoCompressors	$d_{model} \times p \times (m + 1) \times l$	$N \times S$	Софт-промпт	[22]
Мой подход	$k \times d_{mem}$	$N \times S$	LTM блоки	-

Таблица 3.1. Сравнение подходов, использующих рекуррентную память

Обозначения: N - длина входного сегмента, S - количество сегментов, l - количество слоев, H - количество голов внимания, c - размер памяти Compressive Transformer, r - коэффициент сжатия, p - количество токенов памяти, представленных в виде промпта и m - количество шагов накопления векторов суммаризации, k - количество векторов памяти, d_{mem} - размерность векторов памяти.

Из сравнения можно увидеть, что размер памяти в предложенном подходе никак не зависит от длины последовательности или размерности скрытых состояний модели. Размер памяти является регулируемым гиперпараметром, оптимальные значения которого необходимо исследовать. Предполагается, что объем памяти может быть меньше по сравнению с объемом памяти в существующих подходах (это не относится к памяти, необходимой для хранения моделей).

Обращение к памяти осуществляется с использованием блоков долгосрочной памяти LLM_{LTM} через механизм внимания. Как и в методах Memorizing Transformer, RMT и AutoCompressors, в моем подходе также используется более продвинутый механизм запоминания (агент), который определяет, что следует запомнить, а что можно забыть.

3.6. Выводы

В данной главе я подробно описала предлагаемый подход для формирования долгосрочной памяти больших языковых моделей. Предложенная архитектура имеет ряд преимуществ: возможность использования уже предобученной языковой модели, относительно простая интеграция новых

блоков в ее архитектуру, потенциальное сокращение размера памяти, потенциально эффективный механизм отбора воспоминаний, а также использование преимуществ обучения с подкреплением, которые обсуждались в параграфе 1.4.

Глава 4

Эксперименты

4.1. Датасет

Для обучения моделей, способных формировать и использовать долгосрочную память, был создан датасет на основе русскоязычных статей Википедии. Основная идея заключается в том, чтобы предоставить модели контекст из других статей, что позволяет ей лучше запоминать и использовать информацию для предсказания основного текста.

4.1.1. Структура статьи Википедии

Каждая статья Википедии x имеет определенную структуру: она состоит из определения некоторой сущности O , о которой рассказывается в статье и параграфов p_i , $i \in [1, n]$:

$$x = O \circ p_1 \circ \dots \circ p_n \quad (4.1)$$

В каждом параграфе упоминаются сущности из других статей Википедии, представленные ссылками l_j . Множество ссылок для параграфа p_i можно представить в виде l_1, \dots, l_k , где k - целое неотрицательное число.

4.1.2. Формирование примера в датасете

Для каждого параграфа и введения статьи семплируется одна ссылка $l_{rand(p_i)}$, после чего все ссылки перемешиваются и объединяются с основным текстом статьи. Это позволяет модели сперва увидеть контекст, запомнить его и использовать для улучшения своих предсказаний на основном тексте. Обобщая вышесказанное, структура статьи для модели выглядит следующим образом:

$$x_{dataset} = \text{permute} \& \text{concat} [l_{rand(O)}, l_{rand(p_1)}, \dots, l_{rand(p_n)}] \circ x \quad (4.2)$$

Схема формирования примера датасета представлена на рисунке 4.1.

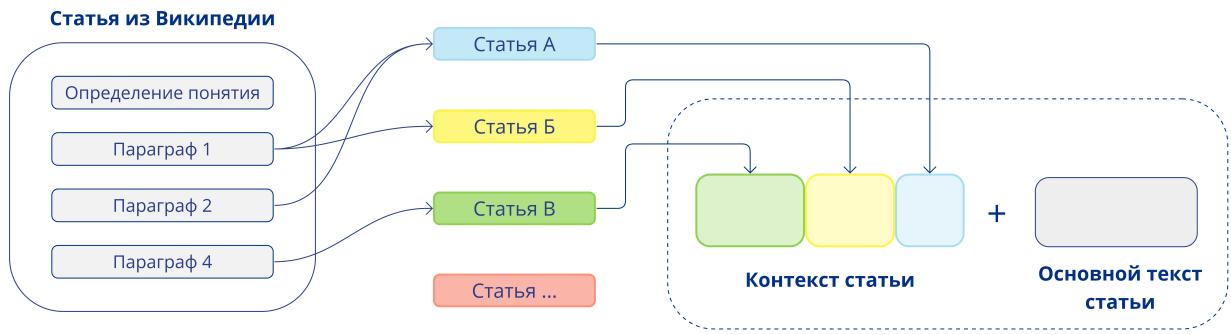


Рис. 4.1. Формирование примера в датасете с использованием статей Википедии

4.1.3. Характеристики датасета

Для парсинга и очистки статей использовался парсер Wikiextractor-V2 [26]. В датасет включались только те статьи, длина которых превышала определенное количество токенов и для которых было непустое множество ссылок. Статьи, не соответствующие этим требованиям, использовались только в качестве контекста.

Количество примеров:

1. Обучающая выборка - 250 128.
2. Валидационная выборка - 31 280.
3. Тестовая выборка - 31 512.

4.2. Характеристики обучения

4.2.1. Выбор предобученной модели

Для обучения была выбрана модель GPT-3 Small [27], которая включает 12 декодерных слоев и имеет размерность скрытых состояний 768. Такой выбор обусловлен ограничениями на вычислительные ресурсы.

4.2.2. Алгоритм обучения с подкреплением

Для обучения агента был применен алгоритм REINFORCE [23]. В алгоритм были внесены следующие модификации:

1. Ограничение дивергенции Кульбака-Лейблера с помощью клиппинга отношения вероятностей действий старой и новой политик:

$$z_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

Тогда функция потерь определяется следующим образом:

$$L^{\text{CLIP}}(\theta) = -\mathbb{E}_t [\min(z_t(\theta)R_t, \text{clip}(z_t(\theta), 1 - \epsilon, 1 + \epsilon)R_t)]$$

Здесь ϵ — это гиперпараметр, определяющий границы клиппинга, и R_t — суммарная ожидаемая награда агента.

2. Ранняя остановка при превышении заданного значения дивергенции Кульбака-Лейблера.
3. Регуляризация энтропии действий агента для поддержания исследования среды агентом. Регуляризация энтропии добавляется к общей функции потерь с отрицательным знаком и некоторым коэффициентом регуляризации, определяющим вес энтропии.
4. Клиппинг нормы градиентов.

Награда агента — средняя кросс-энтропия по трем случайным префиксам следующего сегмента, что позволяет более точно оценивать качество генерации на разных длинах.

4.2.3. Число обучаемых параметров

Модель *LLM-LTM* содержит 7 783 744 обучаемых параметров во float32. Модель памяти содержит 9 823 105 обучаемых параметров во float32, энкодер состоит из двух блоков, декодер - из трех.

4.2.4. Параметры обучения

Обучение проводилось на двух GPU с общим объемом памяти 22 ГБ. Основные параметры обучения:

1. Длина сегмента: 256 токенов.
2. Размер памяти: 10 векторов размерности 64.
3. Скорость обучения (learning rate) для обеих моделей - 3e-5.
4. Количество итераций обучения LLM в одном цикле - 15.
5. Количество итераций обучения агента в одном цикле - 15.

Для предотвращения деградации генерации коротких последовательностей сегменты для LLM обрезались по случайному префиксу. Для обучения агента использовались полные сегменты.

4.2.5. Метрики для валидации результатов

В качестве метрик для валидации полученных результатов я выбрала следующие:

1. Кросс-энтропия $H(p, q)$, где p - истинное распределение, а q - предсказанное.
2. Перплексия - экспонента средней кросс-энтропии. Более низкие значения перплексии указывают на лучшую производительность модели. Определяется следующим образом:

$$\text{Perplexity} = \exp(H(p, q))$$

3. Top-k accuracy. Эта метрика измеряет долю случаев, когда истинный класс находится среди k наиболее вероятных классов, предсказанных моделью. Если истинное следующее слово находится в k наиболее вероятных, предсказанных моделью, то это считается успешным предсказанием в рамках метрики Top-k accuracy.

$$\text{Top-k accuracy} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(y_i \in \text{Top-}k(q_i))$$

где:

- y_i — истинное следующее слово для примера i .
- Топ- k (q_i) — множество k наиболее вероятных слов, предсказанных моделью для примера i .
- $\mathbb{I}(\cdot)$ — индикаторная функция, которая возвращает 1, если условие выполняется, и 0 в противном случае.

4.2.6. Решение проблем с обучением

На начальных этапах я столкнулась с рядом проблем, связанных с необучаемостью моделей, и внедрила некоторые модификации в архитектуру моделей и процесс обучения для их решения.

1. Модель памяти агента демонстрировала расхождение, из-за чего в компонентах градиента появлялись бесконечные значения. Это происходило потому, что энтропия действий агента стремительно росла, даже несмотря на большой коэффициент перед ней в общей функции потерь. Для решения этой проблемы были приняты следующие меры:

- Введение адаптивного коэффициента перед энтропией.
- Ограничение параметров нормального распределения, а именно - стандартного отклонения. Для этого выходы блока Action Distribution - \log_{sigma} - преобразовывались следующим образом:

$$\sigma = \exp [2 \cdot \tanh(\log_{\text{sigma}}) - 2] \quad (4.3)$$

Эти модификации позволили добиться стабильного обучения модели памяти.

2. Тестирование агента на синтетической задаче и модификация архитектуры. Изначально в архитектуре модели памяти отсутствовали блоки self-attention, из-за чего векторы памяти не могли обмениваться информацией друг с другом. Это оказалось критически важным для синтетической задачи, направленной на генерацию таких позиций в

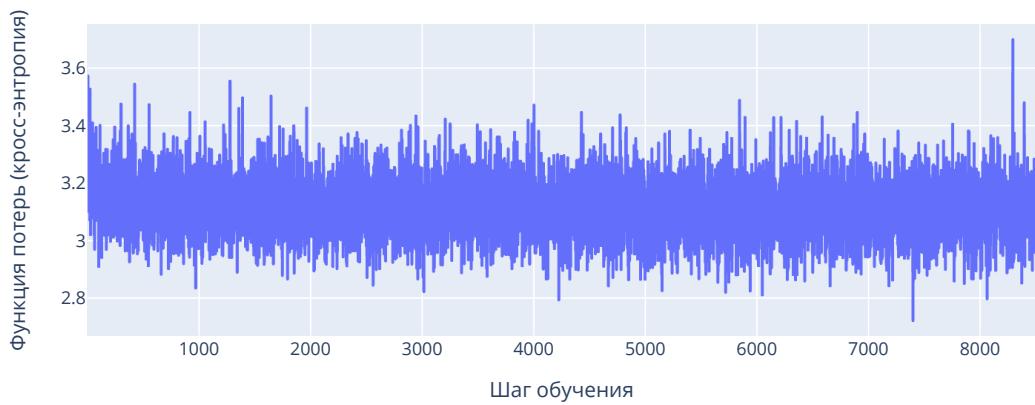
памяти, которые еще не заняты. Агенту необходимо было понимать, что и где уже хранится в памяти.

Используя только блок Action Distribution, состоящий из полносвязанных сетей, я обнаружила, что агент не справляется с этой простой задачей. Введение нескольких сверточных слоев, аналогичных механизмам внимания, позволило добиться роста награды и сходимости к оптимальному решению. Поэтому для решения основной задачи я решила добавить в архитектуру модели памяти механизм self-attention.

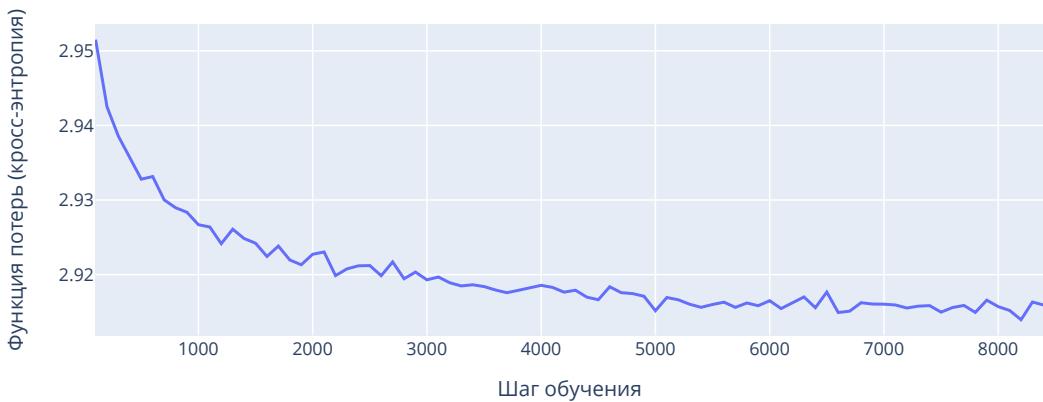
4.3. Результаты

4.3.1. Бейзлайн

В качестве бейзлайна для сравнения я выбрала GPT3, использовавшуюся в качестве основы для моей модели. Я дообучила эту модель с помощью LoRA на обучающей выборке до момента сходимости. График функции потерь (средней кросс-энтропии) представлен на рисунках 4.2, *a*, 4.2, *б*.



(*а*) Функция потерь бейзлайна на обучающей выборке

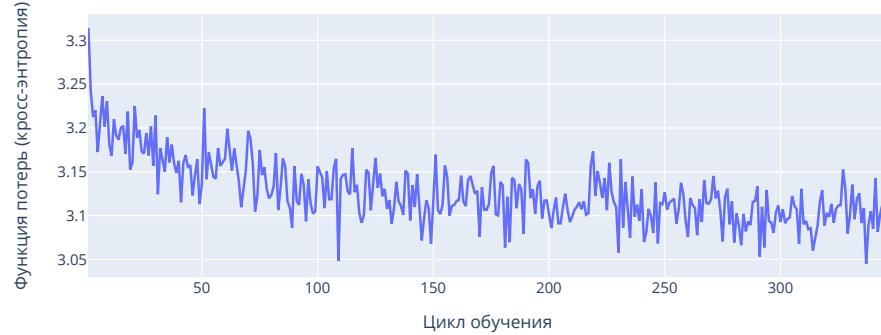


(*б*) Функция потерь бейзлайна на валидационной выборке

Рис. 4.2. Функции потерь бейзлайна

4.3.2. Обучение модели $LLM-LTM + Memory Model$

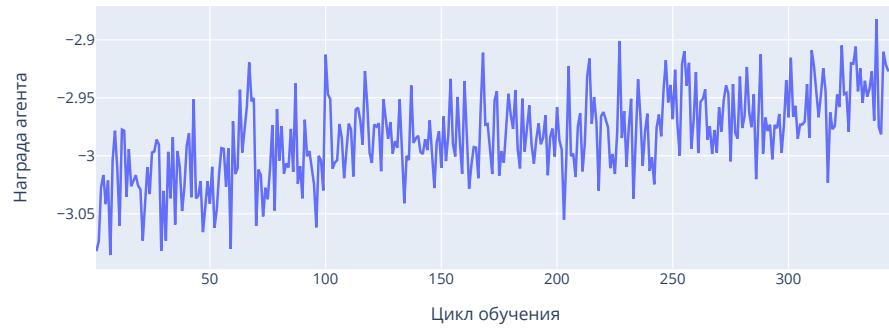
Ниже представлены результаты обучения модели $LLM-LTM + Memory Model$ с параметрами, указанными в предыдущем разделе.



(а) Функция потерь модели $LLM-LTM$ на обучающей выборке



(б) Функция потерь модели $LLM-LTM$ на валидационной выборке



(в) Награда агента

Рис. 4.3. Результаты обучения модели $LLM-LTM + Memory Model$

Как видно из графиков на рисунке , функция потерь $LLM-LTM$ умень-

шается, и начиная с определенного шага сходится к некоторому значению. Однако это значение больше, чем то, к которому сходится бейзлайн. Хотелось бы достичь лучшей производительности по сравнению с бейзлайном, так как бейзлайн не учитывает контекст из предыдущих сегментов, но при этом делает более качественные предсказания.

4.3.3. Гипотеза: агенту требуется предобучение

Гипотеза состоит в том, что агенту сложно одновременно учиться понимать верхнеуровневые представления текста, выбирать необходимую позицию в памяти и генерировать полезный вектор. Поэтому было принято решение предобучить агента на генерацию векторов, похожих на эмбеддинги, получаемые из LLM_{fixed} .

Для этого агент получает на вход матрицу эмбеддингов, пытаясь заполнить матрицу памяти похожими векторами. Поскольку вектор эмбеддинга и вектор памяти имеют разные размерности, необходимо создать матрицу преобразования A , которая будет случайно инициализирована, для преобразования одной размерности в другую.

Награда агента на шаге t определяется следующим образом:

$$R_t = -\Delta_{t-1, t} \text{ diff}_t \quad (4.4)$$

$$\text{diff}_t = \sum_i \min_j \|A \cdot \text{memory}_{t,i} - \text{emb}_j\|$$

На рисунке 4.4 представлен график средней награды агента за шаг.



Рис. 4.4. Средняя награда агента за шаг в процессе его предобучения

Изначально агент генерировал векторы, которые по расстоянию дальше от эмбеддингов, чем нулевые векторы и получал отрицательную награду, но далее достаточно быстро сошелся. Можно считать, что агент обучился генерировать векторы, близкие к эмбеддингам, так как он заполняет все пустые позиции в матрице: нулевой вектор находится дальше от векторов эмбеддингов, чем сгенерированные агентом. На рисунке 4.5 изображено сравнение модели, основанной на предобученном агенте, и модели из параграфа 4.3.2.

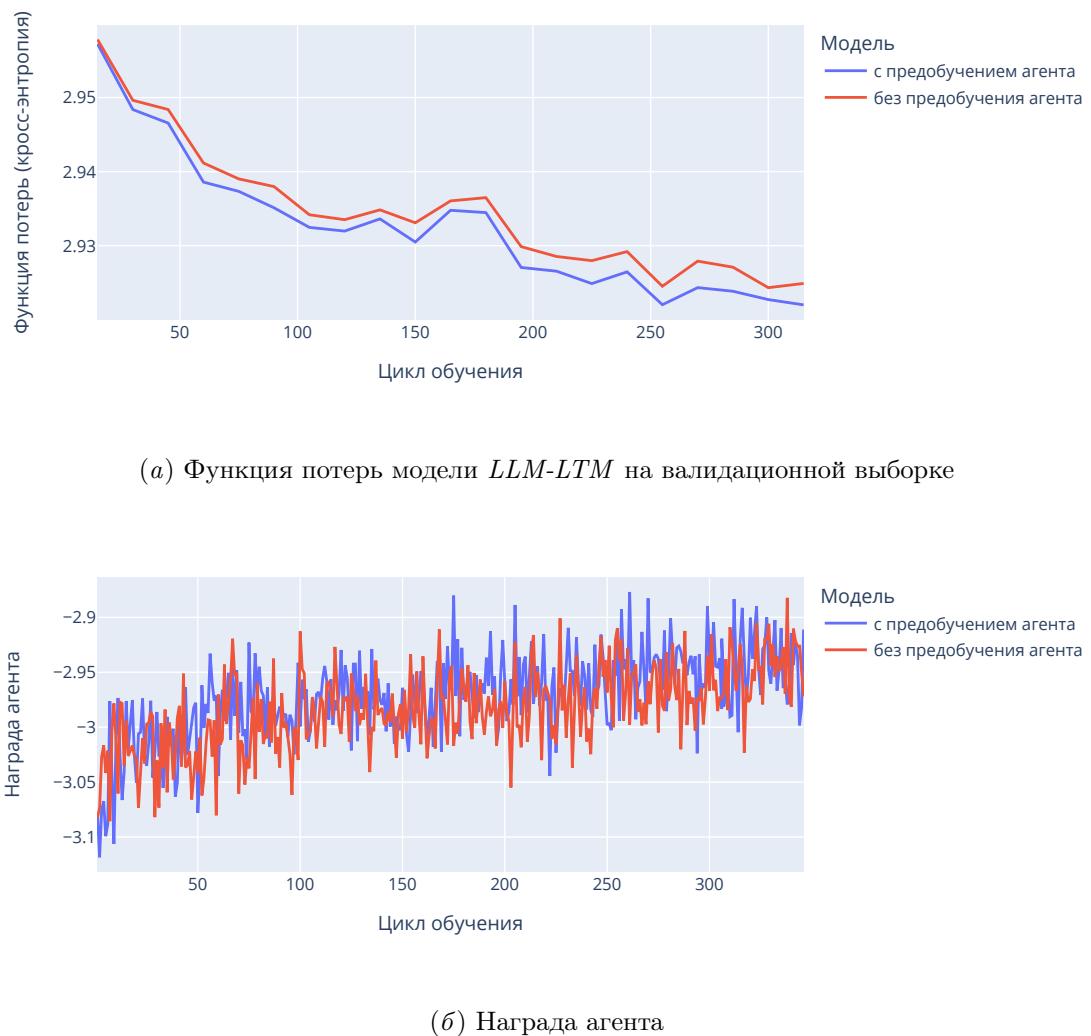


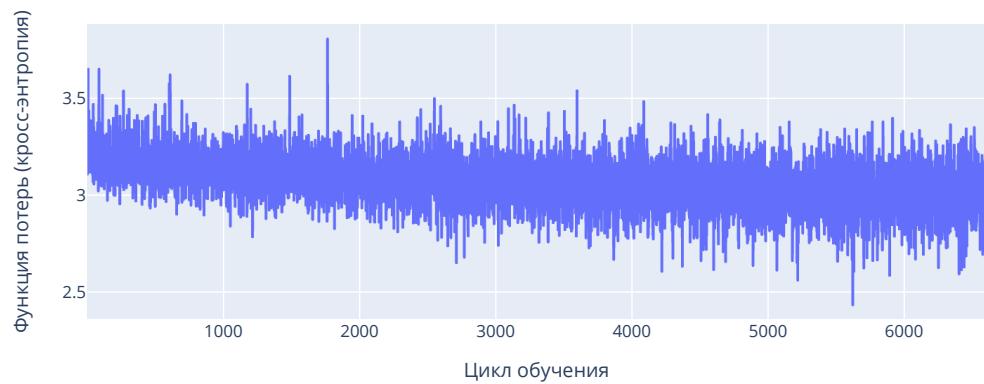
Рис. 4.5. Сравнение моделей: *LLM-LTM* с предобучением агента и без

Из рисунков видно, что на валидационном множестве значение функции потерь модели с предобучением агента чуть ниже, чем без предобуче-

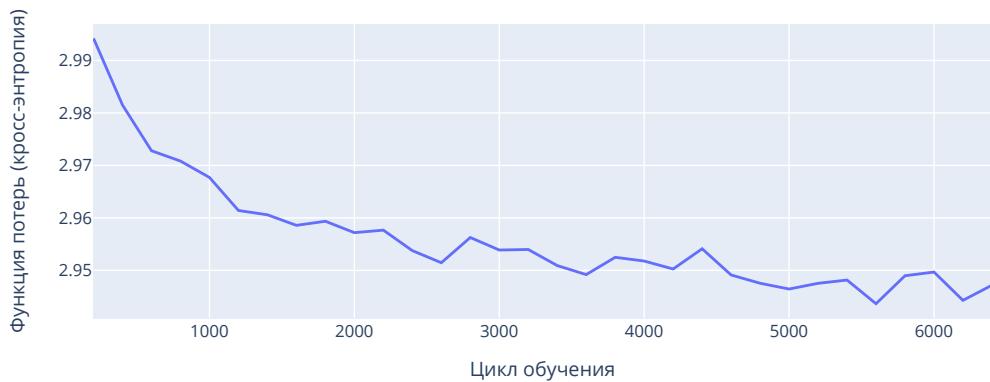
ния. Однако модель по-прежнему сходится к некоторому локальному оптимуму, уступая бейзлайну.

4.3.4. Гипотеза: предобучение *LTM-LLM*

Аналогично предыдущей гипотезе, я решила отдельно предобучить *LLM-LTM* модель, чтобы она учились смотреть на векторы в памяти. Для этого в матрицу памяти я помещала эмбеддинги с предыдущего сегмента для k последних слов, где k - количество векторов в памяти. На данной задаче также необходимо использовать матрицу преобразования для векторов памяти, чтобы они имели такую же размерность, как и эмбеддинги. Результаты обучения представлены ниже:



(a) Функция потерь модели *LLM-LTM* на обучающей выборке



(б) Функция потерь модели *LLM-LTM* на валидационной выборке

Рис. 4.6. Результаты предобучения *LLM-LTM*

Судя по тому, что значение функции потерь во время проверки на валидационном наборе данных, к которому сошлась данная модель, меньше чем то, что получается в общем пайплайне обучения, можно утверждать, что модель учится использовать память, но значения, генерируемые агентом, оказываются полезнее, чем векторные представления слов, полученных из предыдущего сегмента.

Далее я решила использовать эту модель в общем пайплайне обучения, дообучая её и обучая агента с нуля. Результаты сравнения функции потерь на валидационной выборке представлены на рисунке 4.7.

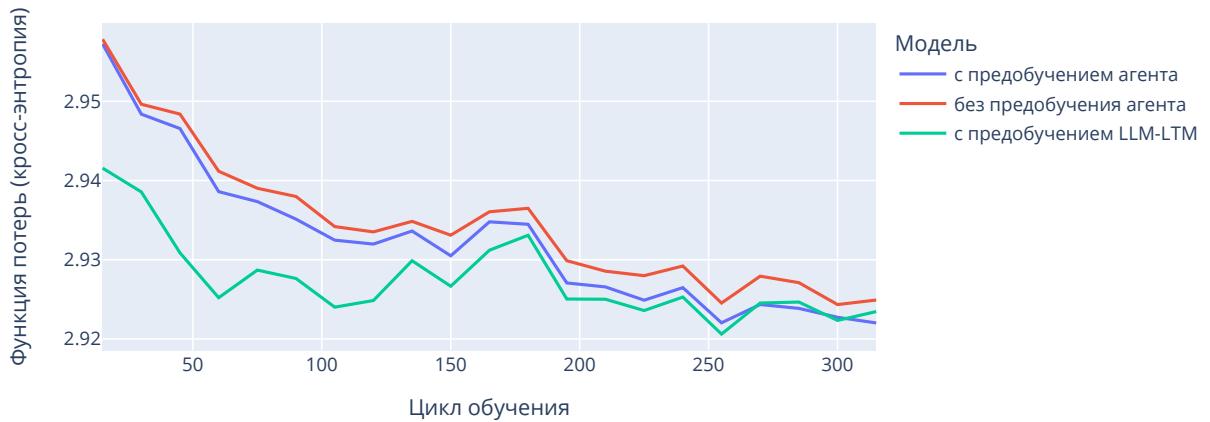


Рис. 4.7. Сравнение моделей: *LLM-LTM + Memory Model* без предобучения, *LLM-LTM + Memory Model* с предобучением *Memory Model* и *LLM-LTM + Memory Model* с предобучением *LLM-LTM*

По графикам видно, что все модели сходятся к одному и тому же значению функции потерь, несмотря на разные начальные точки процедуры оптимизации. Таким образом, добавление дополнительной памяти предложенным способом не позволяет повысить качество генерации, однако достигает сравнимого результата.

4.3.5. Дополнительные эксперименты

Я провела серию экспериментов, в которых перебрала ограниченное множество гиперпараметров и не добилась сходимости предложенного под-

хода:

1. Скорость обучения: $2e-5$, $3e-5$, $5e-5$. Значительной разницы в результатах обучения не обнаружено. При достаточно высокой скорости обучения $5e-5$ политика агента достаточно сильно менялась, что часто служило причиной ранней остановки обучения из-за превышения допустимого значения KLD (оно было также повышенено относительно общепринятых из-за большого количества составляющих действия агента).
2. Размерность векторов памяти: 64 и 256. Результаты обучения также идентичны (см. рис. 4.8).

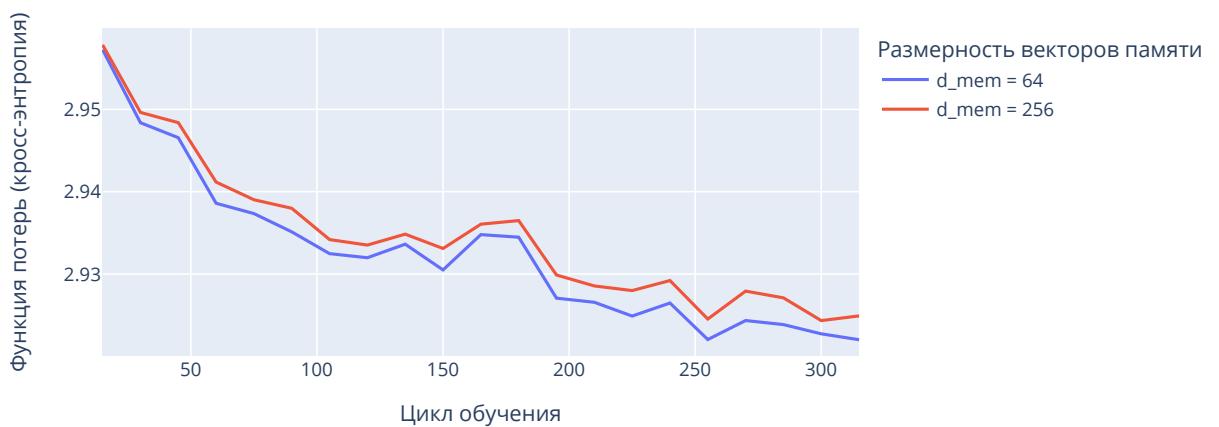


Рис. 4.8. Сравнение моделей *LLM-LTM + Memory Model* без предобучения с разными размерностями векторов памяти: 64 и 256

4.3.6. Сравнение метрик

Для сравнения обученных моделей и бейзлайна я использовала следующие метрики: кросс-энтропия, перплексия и Top-k accuracy. Подробное их описание представлено в параграфе 4.2.5.

Результаты на тестовой выборке практически соответствуют результатам, полученным на валидационной выборке. Модель с предобучением агента достигает результата, сопоставимого с бейзлайном.

Модель	Кросс-энтропия	Перплексия
Бейзлайн	2.9250 ± 0.0044	18.7243 ± 0.0810
Модель с предобучением агента	2.9294 ± 0.0088	18.8081 ± 0.0816
Модель без предобучения агента	2.9365 ± 0.0044	18.9401 ± 0.0818
Предобученная LLM-LTM	2.9577 ± 0.0044	19.3466 ± 0.0834

Таблица 4.1. Сравнение метрик на тестовой выборке

Модель	K=5	K=10	K=20	K=50	K=100
Бейзлайн	0.6542	0.7173	0.7730	0.8380	0.8805
Модель с предобучением агента	0.6539	0.7171	0.7728	0.8377	0.8802
Модель без предобучения агента	0.6531	0.7165	0.7721	0.8371	0.8795
Предобученная LLM-LTM	0.6509	0.7141	0.7701	0.8354	0.8781

Таблица 4.2. Top-k accuracy на тестовой выборке

4.4. Выводы

В данной главе описан подход к формированию датасета с русскоязычными текстами, определены основные параметры и характеристики процесса обучения, а также приведены результаты проведенных экспериментов. Анализ показал, что предложенный метод не привел к улучшению качества генерации текста и не превзошел результаты бейзлайна — классического трансформера декодерного типа без встроенного механизма памяти. Были выдвинуты и протестированы несколько гипотез для улучшения предсказательной способности модели, однако они не подтвердились. Возможно, выбранные гиперпараметры не были оптимальными для рассматриваемой задачи и требуют более тщательной настройки.

Заключение

В данной работе подробно рассматривается задача формирования долгосрочной памяти у больших языковых моделей и расширения их контекста. Проведен анализ существующих подходов к решению этой задачи, выявлены их сильные стороны и недостатки.

В рамках данного исследования были получены следующие результаты:

- Осуществлена сборка русскоязычного датасета на основе статей Википедии с возможностью формирования дополнительного контекста.
- Разработан и протестирован подход к созданию долгосрочной памяти больших языковых моделей с использованием обучения с подкреплением.
- Проведены эксперименты по улучшению оригинального подхода.
- Приведено сравнение результатов работы предложенного метода с предобученной языковой моделью, которая лежала в основе обучаемых моделей и выполняла роль бейзлайна.

Результаты исследования показали, что предложенный подход не продемонстрировал улучшений по сравнению с бейзлайном, который не использует память для генерации текста. Это может свидетельствовать о том, что предложенный метод не оказал значительного влияния на формирование памяти у модели. Это потенциально может стать одним из направлений дальнейшего исследования.

Список литературы

1. Attention Is All You Need. — 2023. — 1706.03762.
2. Hierarchical Attention Networks for Document Classification / Yang Zichao, Yang Diyi, Dyer Chris, He Xiaodong, Smola Alex, and Hovy Eduard // Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies / ed. by Knight Kevin, Nenkova Ani, Rambow Owen. — San Diego, California : Association for Computational Linguistics. — 2016. — June. — P. 1480–1489. — Access mode: <https://aclanthology.org/N16-1174>.
3. A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents / Cohan Arman, Dernoncourt Franck, Kim Doo Soon, Bui Trung, Kim Seokhwan, Chang Walter, and Goharian Nazli // Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers) / ed. by Walker Marilyn, Ji Heng, Stent Amanda. — New Orleans, Louisiana : Association for Computational Linguistics. — 2018. — June. — P. 615–621. — Access mode: <https://aclanthology.org/N18-2097>.
4. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. — 2019. — 1810.04805.
5. An Exploration of Hierarchical Attention Transformers for Efficient Long Document Classification. — 2022. — 2210.05529.
6. Training language models to follow instructions with human feedback. — 2022. — 2203.02155.
7. Beltagy Iz, Peters Matthew E., Cohan Arman. Longformer: The Long-Document Transformer. — 2020. — 2004.05150.
8. Big Bird: Transformers for Longer Sequences. — 2021. — 2007.14062.
9. LongT5: Efficient Text-To-Text Transformer for Long Sequences. — 2022. — 2112.07916.
10. BP-Transformer: Modelling Long-Range Context via Binary

- Partitioning. — 2019. — 1911.04070.
11. Linformer: Self-Attention with Linear Complexity. — 2020. — 2006.04768.
 12. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. — 2020. — 2006.16236.
 13. Dao Tri. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. — 2023. — 2307.08691.
 14. Press Ofir, Smith Noah A., Lewis Mike. Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. — 2022. — 2108.12409.
 15. Dissecting Transformer Length Extrapolation via the Lens of Receptive Field Analysis. — 2023. — 2212.10356.
 16. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context / Dai Zihang, Yang Zhilin, Yang Yiming, Carbonell Jaime G., Le Quoc Viet, and Salakhutdinov Ruslan // ACL (1) / ed. by Korhonen Anna, Traum David R., Màrquez Lluís. — Association for Computational Linguistics. — 2019. — P. 2978–2988.
 17. Bulatov Aydar, Kuratov Yuri, Burtsev Mikhail S. Recurrent Memory Transformer. — 2022. — 2207.06881.
 18. Improving language models by retrieving from trillions of tokens. — 2022. — 2112.04426.
 19. REALM: Retrieval-Augmented Language Model Pre-Training. — 2020. — 2002.08909.
 20. Compressive Transformers for Long-Range Sequence Modelling. — 2019. — 1911.05507.
 21. Memorizing Transformers. — 2022. — 2203.08913.
 22. Adapting Language Models to Compress Contexts. — 2023. — 2305.14788.
 23. Policy gradient methods for reinforcement learning with function approximation / Sutton R. S., Mcallester D., Singh S., and Mansour Y. // Advances in Neural Information Processing Systems 12. — MIT Press. — 2000. — Vol. 12. — P. 1057–1063.
 24. LoRA: Low-Rank Adaptation of Large Language Models. — 2021. — 2106.09685.

25. Munkhdalai Tsendsuren, Faruqui Manaal, Gopal Siddharth. Leave No Context Behind: Efficient Infinite Context Transformers with Infini-attention. — 2024. — 2404.07143.
26. BSC Language Technologies Unit. Wikiextractor V2. — <https://github.com/langtech-bsc/Wikiextractor-V2>. — 2023.
27. A Family of Pretrained Transformer Language Models for Russian. — 2023. — 2309.10931.