Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.М07-мм

Сравнительный анализ моделей NLP для автоматической корректуры грамматики

Карими Хурматулла

Отчёт по производственной практике
в форме «Решение»

Научный руководитель:
к. ф.-м. н.доцент Д. В. Луцив

Санкт-Петербург
2023

Оглавление

Введение

Нейронные языковые модели (NLM) превратились в мощные инструменты в задачах обработки естественного языка (NLP), таких как машинный перевод, реферирование текста, ответы на вопросы и исправление грамматики. Эти модели используют способность нейронных сетей изучать сложные шаблоны из больших объемов текстовых данных. Среди различных архитектур NLM, Длинная краткосрочная память (LSTM), Механизм внимания и T5 получили значительное внимание благодаря их превосходной производительности и универсальности. В этом отчете мы изучаем теоретические основы и детали реализации этих трех моделей, предоставляя информацию об их сильных сторонах и ограничениях.

Долгосрочная краткосрочная память (LSTM) - это рекуррентная нейронная сеть (RNN), специально разработанная для обработки последовательностных данных, таких как текст. LSTM преодолевает проблему исчезающего градиента RNN, позволяя ему изучать длительные отношения между словами в предложении. Это делает их подходящими для задач, таких как машинный перевод [10].

Механизмы внимания вводят новый подход к захвату длительных зависимостей в нейронных сетях. Они избирательно фокусируются на определенных частях входной последовательности в зависимости от их релевантности текущей задаче. Это позволяет моделям сосредоточиться на самой важной информации, улучшая производительность в задачах, таких как машинный перевод и реферирование текста. [3]

T5 - это трансформаторная языковая модель, которая представляет собой значительное достижение в области NLP. Трансформеры полностью не используют рекуррентные соединения, а вместо этого полагаются на механизмы самовнимания для захвата зависимостей между словами. Эта архитектура привела к значительно улучшенной производительности в широком диапазоне задач NLP. Таким образом, каждая из архитектур этих моделей является лучшей, поэтому мы собираемся сравнить каждую из них и окончательно решить, какая модель будет

более подходящей для исправления грамматики. [6]

1. Цель работы

Для анализа современных методов и выбора подходящего метода для реализования были поставлены следующие задачи. следующие задачи:

1. Изучить модели обработки естественного языка, которые могут анализировать предложения.
2. Выбрать модели обработки естественного языка, которые позволят, после обучения на правильных текстах, выбирать необходимые формы для вставки фрагментов в текст.
3. Провести эксперименты с этими моделями и выбрать наиболее подходящую.
4. Интегрировать инструмент с библиотекой LanguageTool для получения дополнительных возможностей исправления предложений.
5. Реализовать прототип инструмента, выполняющего макрозамену фраз в правильном падеже и числе.

2. Обзор

2.1. Обзор NLTK

Инструментарий естественного языка (NLTK) - это платформа, используемая для создания программ на Python, которые работают с данными человеческого языка для применения в статистической обработке естественного языка (NLP). Он содержит библиотеки обработки текста для токенизации, синтаксического анализа, классификации, выделения элементов, пометки и семантического обоснования. Он также включает графические демонстрации и примеры наборов данных, а также сопровождается кулинарной книгой и книгой, в которой объясняются принципы, лежащие в основе задач обработки языка, которые поддерживает NLTK [24]. Итак, как мы знаем, NLTK слишком велик, чтобы его объяснять, но мы сосредоточимся на очень специфической области нашей работы: создании структур и внедрении различных анализаторов для наших структур.

2.1.1. Рекурсия в синтаксической структуре

Грамматика называется рекурсивной, если категория, встречающаяся в левой части произведения, также появляется в правой части произведения, как показано рис 1. Производственный Nom -> Adj Nom (где Nom - категория номиналов) включает прямую рекурсию по номинации категории, тогда как косвенная рекурсия по S возникает в результате комбинации двух производств, а именно S -> NP VP и VP -> V S. [14]

2.1.2. Синтаксический анализ рекурсивного спуска (метод сверху вниз)

Это один из самых простых методов синтаксического анализа, который мы можем использовать для наших грамматических структур. Этот метод делит цели на подцели, и каждая подцель подразделяется

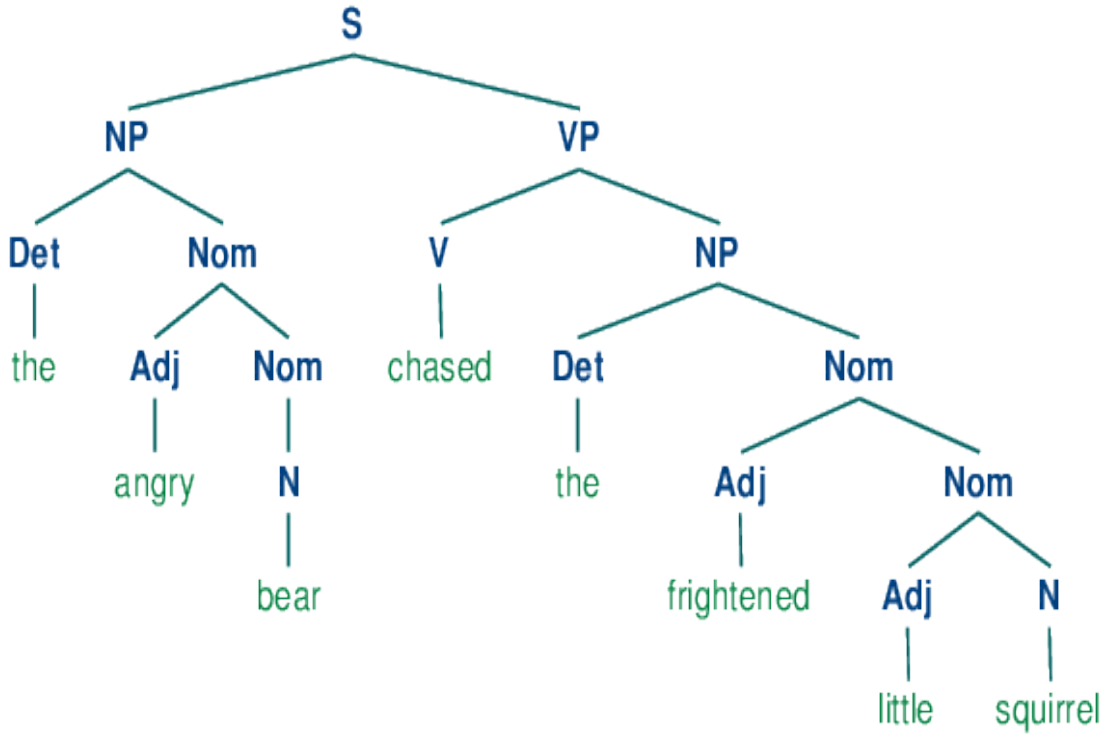

Рис. 1: Пример рекурсивности в синтаксической структуре)

на другую подцель до тех пор, пока структура ваших предложений не будет завершена. [15]

2.2. PyMorphy2

Прежде чем мы начнем обсуждать, что такое pymorphy2, давайте обсудим морфологический анализ.Морфологический анализ - это анализ внутренних структур слов, поэтому, если мы увидим, что русский и украинский языки имеют богатую морфологию, поэтому, используя морфологический анализ, мы определим состояние наших слов, что это состояние, например, существительное или глагол [12]. Pymorphy2 это морфологический анализатор, который анализирует русские тексты с помощью словаря opencorpora. Алгоритм Pymorphy выполняет морфологическую обработку на основе грамматической характеристики типа (слова, лемматизация), но если слово не существует в словаре, поэтому предиктор в pymorphy2 объединит два алгоритма: 1 - по префиксу 2 - по окончанию слов. В то же время, когда мы анализируем слова, мы столкнемся с несколькими состояниями слов и оценками, затем, выбрав

состояние слова с наивысшим баллом Мы можем исправить предложение. [19]. Морфологический анализатор pymorphy2 умеет:

1. приводить слово к нормальной форме (например, “люди -> человек”, или “гулял -> гулять”).

2. ставить слово в нужную форму. Например, ставить слово во множественное число, менять падеж слова и т.д.

3. возвращать грамматическую информацию о слове (число, род, падеж, часть речи и т.д.)[21]

2.3. Высокоэффективное повышение градиента Дерево решений (LightGBM)

Это один из наиболее распространенных алгоритмов искусственного интеллекта, который использовался в многоклассовой классификации, прогнозировании кликов. В настоящее время перед GBP стоит задача получения информации по каждой функции и их расчета, поскольку это отнимает много времени. Итак, для решения этой проблемы GBPT внедрила два метода: 1 - ГОСС (односторонняя выборка на основе градиента): Итак, для наших экземпляров данных нет предопределенных весов, и мы говорим, что разные экземпляры данных с разным градиентом играют разную роль в вычислениях, и в результате мы можем сказать, что больше информации может быть получено из экземпляров больших данных с большие градиенты. 2 - EFB (эксклюзивный пакет функций) - Обычно в реальных приложениях дополнительные функции создают эффективный подход без потерь. [9].

2.4. LemmInflect

LemmInflect использует словарный подход для лемматизации английских слов и преобразования их в формы, заданные предоставленным пользователем тегом Universal Dependencies или Penn Treebank. Библиотека работает со словами, не входящими в словарный запас

(OOV), применяя методы нейронных сетей для классификации словоформ и выбора соответствующих правил морфинга [13].

2.5. NLTK4Russian

Как было написано, проект направлен на создание лингвистического комплекса для анализирования корпусов русскоязычных текстов, основанного на различных методах и алгоритмах в рамках наиболее популярных инструментов современной компьютерной лингвистики (NLTK, Pattern, GenSim и т.д.). Таким образом, в этом проекте они имеют работал над двумя основными задачами:

1. разработка морфологического анализатора для русского языка на основе NLTK и Pymorphy

2. Создание парсера для русского языка на основе категориальной грамматики и парсера, встроенного в NLTK

В рамках проекта производится разработка синтаксического анализатора для русского языка. Модуль синтаксического анализа в NLTK позволяет исследователям самостоятельно создавать формальные грамматики различных типов для разных естественных языков и применять их в конкретных целях автоматической обработки текстов. [20].

2.6. Метод повторного использования документации семейств программных продуктов

Существуют существующие технологии разработки многоразовой технической документации с акцентом на возможность повторного использования, как указано в диссертации Романовского Константина Юрьевича. Существует два типа повторного использования: случайное и запланированное.

Случайное повторное использование предполагает, что разработчики подключают существующие компоненты, когда и если представится

такая возможность. Запланированное повторное использование предполагает систематическую разработку компонентов, подготовленных для повторного использования, и дальнейшее применение таких компонентов в процессе разработки.

В методе SPP есть два прогрессивных метода: упреждающий и гибкий. Мы сможем начать с нуля в proactive, разработав семейство и добавив в него компоненты. Однако в flexible это похоже на то, как мы начнем использовать продукт, а затем добавим повторно используемые компоненты, но выпуск программного обеспечения отнимает много времени [23].

3. Архитектуры и Модели

В данном разделе подробно рассматриваются архитектуры, которые будут задействованы в системе автоматической коррекции грамматики.

3.1. Рекуррентная нейронная сеть (RNN)

Это тип искусственной нейронной сети, которая используется для обработки последовательных данных. Эти алгоритмы глубокого обучения обычно используются, например, для языкового перевода, NLP и т.д.Давайте возьмем выражение, например “плохое самочувствие”, которое означает, что кто-то болен. На этом примере мы объясним работу RNN. Итак, для более глубокого понимания этой идиомы и придания ей смысла, нам нужно выразить идиому в определенном порядке. Используя рекуррентную нейронную сеть, нам нужно обозначить слова, чтобы предсказать следующее слово в последовательности. [11]. Ниже мы поговорим о типе рекуррентной нейронной сети (RNN): 1 - Многие К Одному RNN: это означает, что у нас есть много входов под именем (Tx), но(И) один выход у нас будет под именем (Ty). [17]

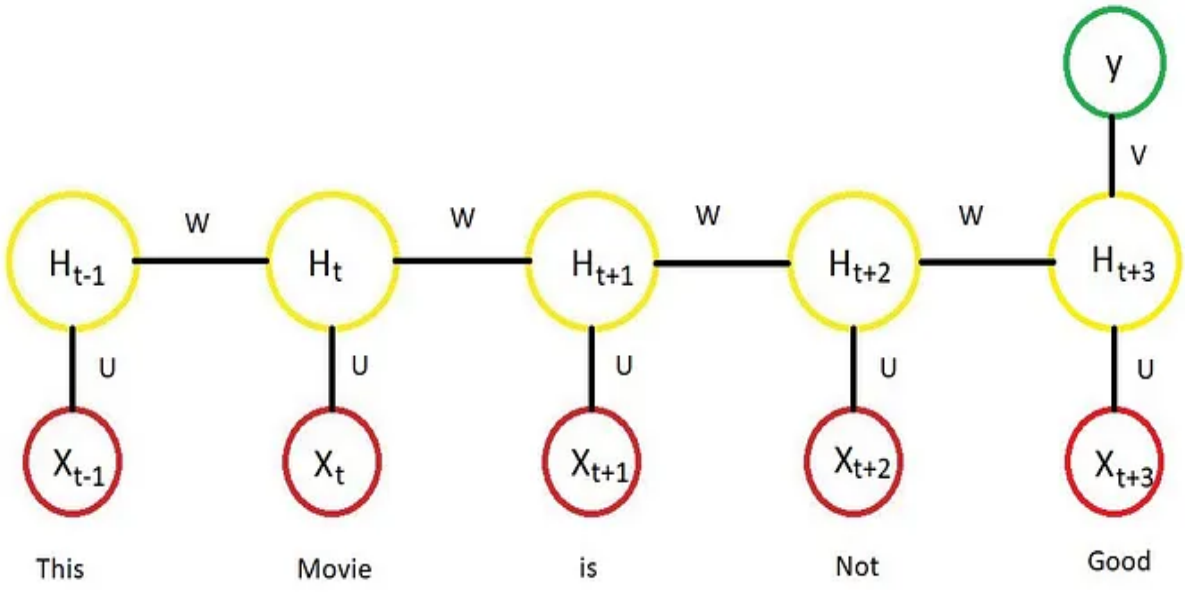

Рис. 2: Многие К Одному RNN

2 - Один Ко Многим: Эта архитектура означает, что RNN, которую

мы обучили ранее, будет генерировать множество выходных данных на основе одного входного сигнала. [17]

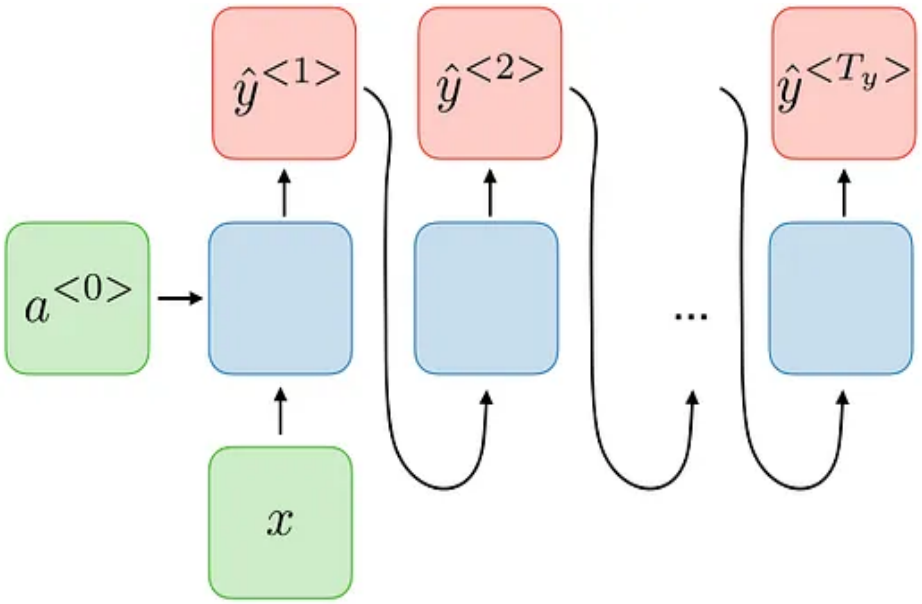

Рис. 3: Один Ко Многим RNN

3 - Многие Ко Многим: Эта архитектура означает, что у нас есть много входных данных, и на основе наших входных данных у нас есть много выходные данные. [17]

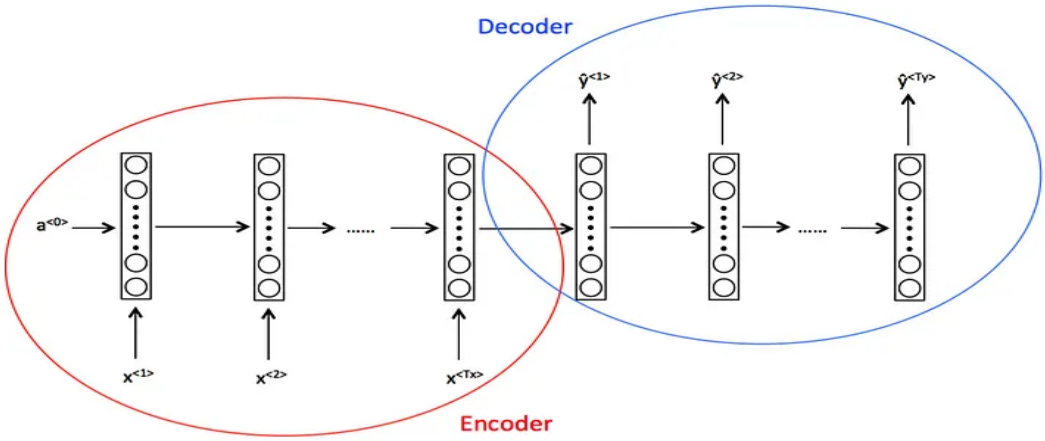

Рис. 4: Многие Ко Многим RNN

3.1.1. Преимущество RNN

3 основных преимущества RNN, которые делают RUN лучше других, заключаются в следующем:

1. Последовательная обработка: это означает, что она(RNN) будет обрабатывать данные последовательно.

2. Гибкость: RNN могут обрабатывать входы и выходы переменной длины, что делает их более гибкими по сравнению с другими моделями.

3. Интерпретируемое: Скрытое состояние RNN может быть интерпретировано как краткое изложение входной последовательности, что облегчает понимание того, как модель делает свои прогнозы.

3.1.2. Недостатки RNN

1. Трудно обрабатывать большую последовательность текста: это означает, что когда у нас будет больше контекстов, промежутков между каждым словом будет больше, и обрабатывать их будет сложно. Например, когда у нас есть предложение “Я вырос во Франции.... Я свободно говорю по-французски”, так что здесь мы, основываясь на прогнозе, будем думать, что следующим словом, вероятно, будет название языка. В случае, если сузить контекст, то пробелы уже становятся настолько большими, что невозможно научиться связывать информацию.

2. Забывание того, что произошло в начале: RNN, конечно, предназначен для сохранения информации с предыдущего шага и прогнозирования на текущем. И если мы знаем, что RNN хранит и распространяет информацию во времени через скрытое состояние, то оно будет обновляться на каждом временном шаге. Таким образом, в скрытом состоянии информация будет передаваться с прошлых шагов на текущие, и когда промежутки становятся

больше, этот процесс усложняется, что мы также можем назвать проблемой исчезающего градиента.

3. Трудно поддается обучению: Когда мы говорим о том, что RNN трудно поддаются обучению, это означает, что они не являются нейронными сетями прямой связи. Нейронная сеть с прямой связью подаст сигнал о перемещении только в одном направлении, и этот сигнал будет перемещаться от входных слоев к различным скрытым слоям, а затем перенаправляться на выход системы.

4. Трудно распараллелить.

3.2. Долгая краткосрочная память (LSTM)

Долгая краткосрочная память (LSTM) - это архитектура рекуррентной нейронной сети, состоящая из одной единицы - памяти (также известной как LSTM-единица). LSTM-единица состоит из четырех полносвязных нейронных сетей. Каждая из этих нейронных сетей состоит из входного слоя и выходного слоя. Во всех этих нейронных сетях входные нейроны соединены со всеми выходными нейронами. Таким образом, LSTM-единица имеет четыре полносвязных слоя. Три из четырех полносвязных нейросетей отвечают за выбор информации. Это ворота забывания, входных ворот и выхода. Эти три ворот используются для выполнения трех типичных операций управления памятью: удаление информации из памяти (ворота забывания), вставка новой информации в память (ворота входа) и использование информации, присутствующей в памяти (выходные ворота). Четвертая нейронная сеть, кандидат в память, используется для создания новой кандидатной информации для вставки в память. [10, 5]

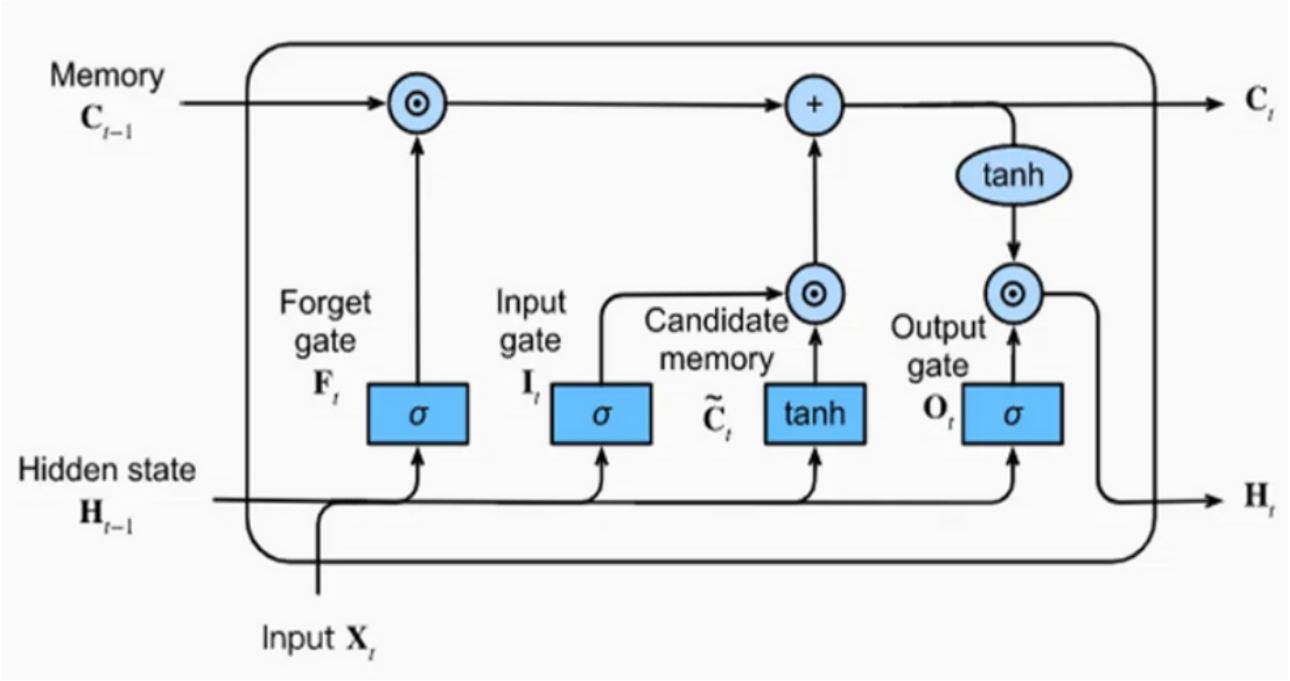

Рис. 5: LSTM структура

Архитектура LSTM основана на воротах, и LSTM имеет 3 таких ворот. Эти три шлюза (ворота забывания, входного шлюза и выхода) являются селекторами информации. Их задача - создавать векторы отбора. Вектор отбора - это вектор со значениями между нулем и единицей и близок к этим двум крайним значениям. Вектор отбора создается для того, чтобы быть умножен поэлементно на другой вектор того же размера. Это означает, что позиция, в которой вектор отбора имеет значение равное нулю, устраняет (в умножении поэлементно) информацию, включенную в той же позиции в другом векторе. Позиция, в которой вектор отбора имеет значение равное единице, оставляет без изменения (в умножении поэлементно) информацию, включенную в той же позиции в другом векторе. Все три шлюза являются нейронными сетями, которые используют сигмоидную функцию в качестве функции активации в выходном слое. Сигмоидная функция используется для получения в качестве выходного вектора, состоящего из значений между нулем и единицей и близкого к этим двум крайним значениям. [10, 5]

3.2.1. Ворота забывания

В любой момент времени t LSTM получает в качестве входного вектора (X_[t]) вектор входных данных. Он также получает векторы скрытого состояния (H_[t−1]) и состояния ячейки (C_[t−1]) , определенные в предыдущем моменте времени (t− 1). Первая активность единицы LSTM выполняется затвором забывания. Затвор забывания решает (на основе векторов X_[t] и H_[t−1]) какую информацию удалить из вектора состояния ячейки, поступающего из времени t−1. Результат этого решения представляет собой вектор селектора X. [10, 5] Вектор селекто-

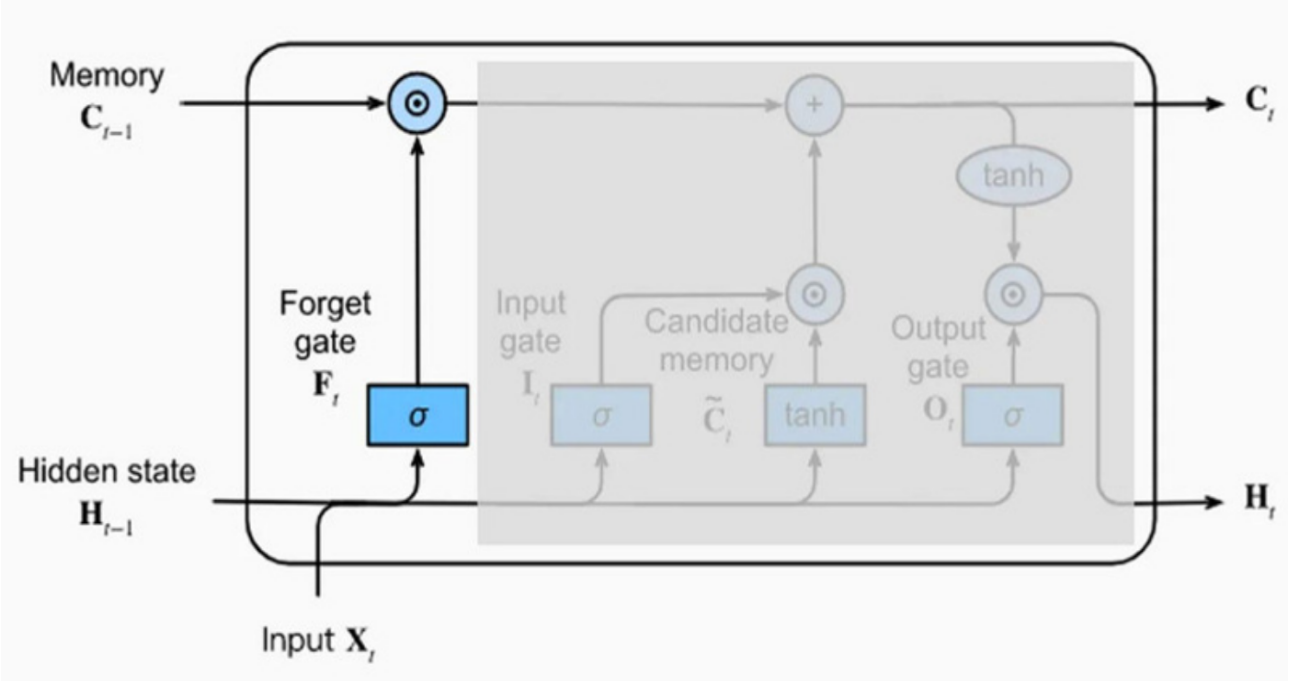

Рис. 6: Ворота забывания

ра умножается поэлементно с вектором состояния ячейки, полученным в качестве входного сигнала для LSTM-единицы. Это означает, что в позиции, где вектор селектора имеет значение, равное нулю, информация, включенная в той же позиции в состоянии ячейки, устраняется (в процессе умножения). Позиция, в которой вектор селектора имеет значение, равное единице, оставляет информацию, включенную в той же позиции в состоянии ячейки, неизменной (в процессе умножения). [10, 5]

3.2.2. Ворота Входного и Память кандидата

После удаления некоторой информации из входного состояния ячейки (C_[t−1]), мы можем вставить новую. Эта активность выполняется двумя нейросетями: кандидатной памятью и входным затвором. Две нейросети независимы друг от друга. Их входом являются векторы X_[t] и H_[t−1], соединенные вместе в один вектор. [10, 5] Кандидат-

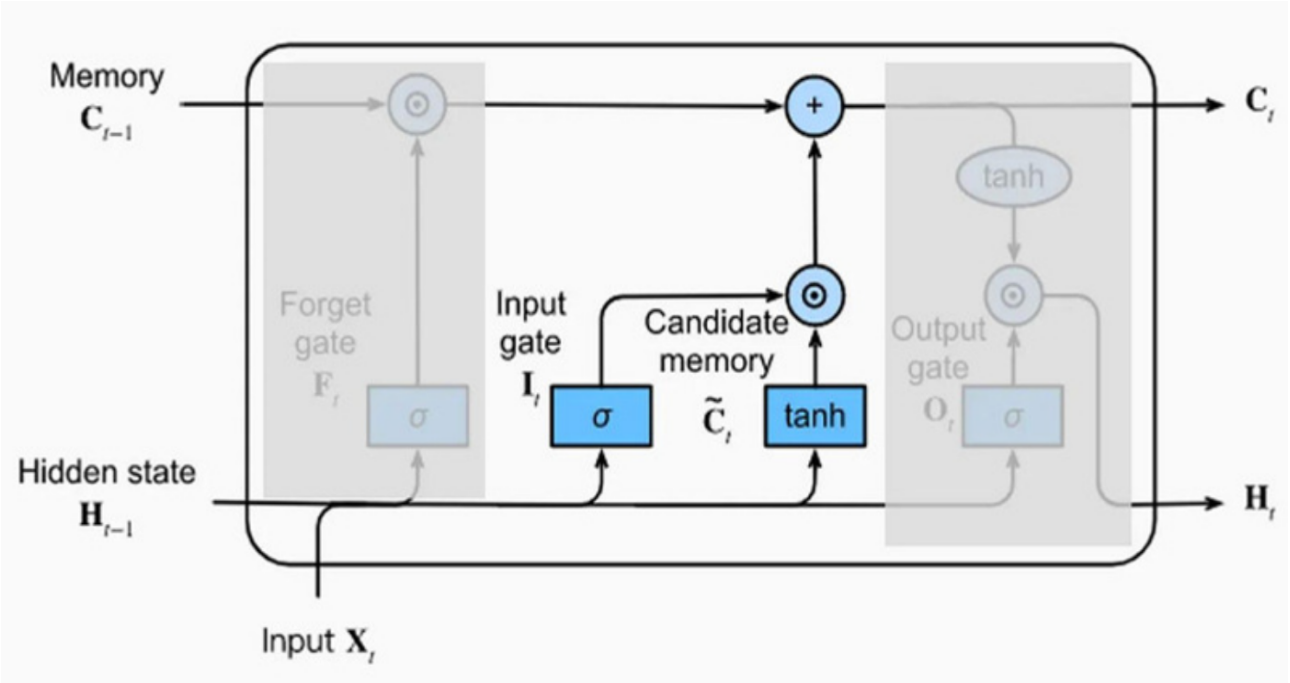

Рис. 7: Входной шлюз и Память кандидата

ная память отвечает за генерацию кандидатного вектора: вектора информации, который может быть добавлен к состоянию ячейки. Выходные нейроны кандидатной памяти используют гиперболическую тангенсную функцию. Свойства этой функции гарантируют, что все значения кандидатного вектора находятся между -1 и 1. Это используется для нормализации информации, которая будет добавлена к состоянию ячейки. Входной затвор отвечает за генерацию селектора, который будет умножен поэлементно с кандидатным вектором. Селекционный вектор и кандидатный вектор умножают друг на друга поэлементно. Это означает, что позиция, где селекторный вектор имеет значение, равное нулю, исключает (в процессе умножения) информацию, включенную в

той же позиции в кандидатном векторе. Позиция, в которой селекторный вектор имеет значение, равное единице, оставляет информацию, включенную в той же позиции в кандидатном векторе, неизменной (в процессе умножения). Результат умножения между кандидатным вектором и селективным вектором добавляется к вектору состояния ячейки. Это добавляет новую информацию к состоянию ячейки. Состояние ячейки, после того, как оно было обновлено операциями, которые мы видели, используется выходным затвором и передается в набор входных данных, используемых LSTM-единицей в следующем моменте времени (t + 1). [10, 5]

3.2.3. Ворота Выходного

Выходной затвор определяет значение скрытого состояния, выдаваемого LSTM (в момент времени t) и получаемого LSTM в следующем наборе входных данных (t + 1). Генерация выхода также работает с умножением селектора и кандидата. В данном случае, однако, кандидатный вектор не генерируется нейронной сетью, а получается просто путем применения гиперболической функции тангенса на вектор состояния ячейки. Этот шаг нормализует значения вектора состояния ячейки в пределах от -1 до 1. Таким образом, после умножения с селектором (значения которого находятся между нулем и единицей) мы получаем скрытое состояние со значениями между -1 и 1. Это позволяет контролировать стабильность сети в течение времени. [10, 5]

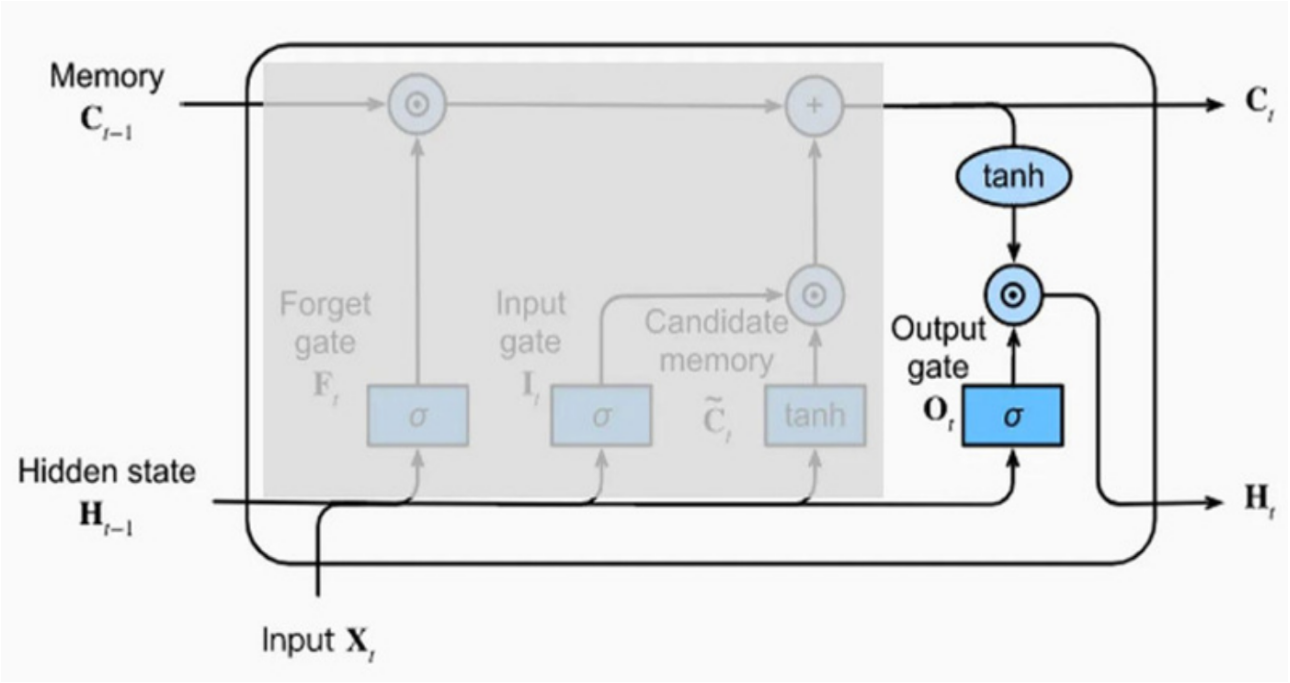

Рис. 8: Выходной шлюз

Вектор селектора генерируется из выходного затвора на основе значений X_[t] и H_[t−1], полученных в качестве входных данных. Выходной затвор использует функцию сигмоида в качестве функции активации выходных нейронов. Вектор селектора и кандидатный вектор умножают друг на друга, элемент за элементом. Это означает, что позиция, где вектор селектора имеет значение, равное нулю, исключает (при умножении) информацию, включенную в той же позиции в кандидатном векторе. Позиция, в которой вектор селектора имеет значение, равное единице, оставляет информацию, включенную в той же позиции в кандидатном векторе, неизменной (при умножении). [10, 5]

3.3. Механизм внимания (Bahdanau)

Механизм внимания Бахданау получил свое название от первого автора статьи, в которой он был опубликован. В этой архитектуре Багданау утверждал, что кодирование входного сигнала переменной длины в вектор с фиксированной длиной сжимает информацию исходного предложения, независимо от его длины, что приводит к быстрому ухудше-

нию производительности базовой модели кодировщика-декодера с увеличением длины входного предложения. Предлагаемый подход заменяет фиксированный вектор переменным вектором для улучшения производительности перевода базовой модели кодировщика-декодера. [4, 3]

3.3.1. Архитектура Bahdanau

Основные компоненты, используемые в архитектуре кодировщика-декодера Бахданау, следующие:

1. S_{t-1} - скрытое состояние декодера в предыдущем шаге времени, t-1.

2. C_t - вектор контекста в момент t. Он уникально генерируется на каждом шаге декодера для генерации целевого слова Y_t.

3. H_t - аннотация, которая захватывает информацию, содержащуюся в словах, образующих полное входное предложение, X_1, X_2, ... X_T с сильным фокусом вокруг i-го слова из T всех слов.$A_{t,i}$ - значение веса, присвоенное каждому аннотированию h_i в текущем момент времени t.

4. $E_{t,i}$ - значение оценки внимания, генерируемой моделью выравнивания a(.), которое оценивает, насколько хорошо S_{t-1} и H_i совпадают.

Эти компоненты находят свое применение на разных этапах архитектуры Бахданау, которая использует двунаправленную RNN в качестве кодировщика и RNN-декодера, с механизмом внимания между ними. [4, 3]

3.3.2. Кодировщик

Роль кодировщика состоит в том, чтобы генерировать аннотацию H_i для каждого слова X_i в предложении длиной T слов. Для этого Багданау и др. используют двунаправленную RNN, которая читает предложение в прямом направлении для получения скрытого состояния впе-

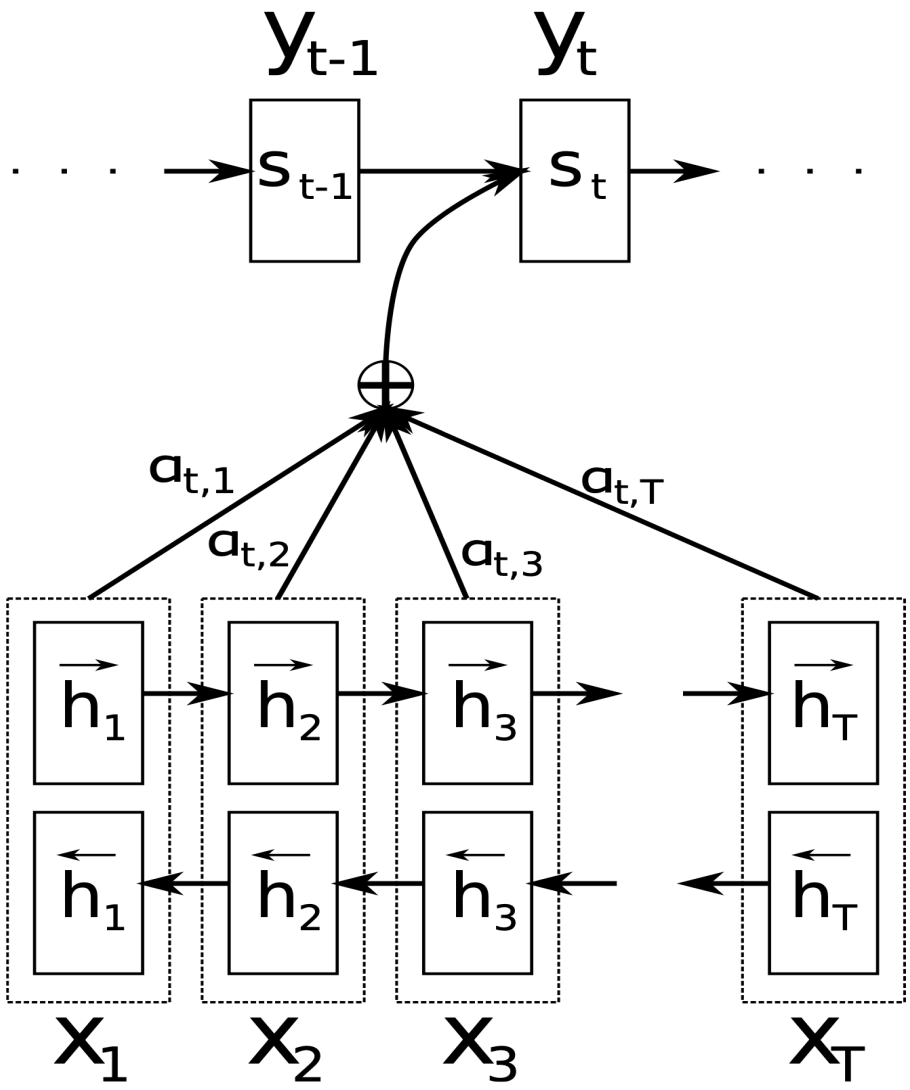

Рис. 9: Механизм внимания (Bahdanau)

ред $H^{->}_i$ и затем читает предложение в обратном направлении для получения скрытого состояния назад $H^{<-}_i$. Аннотация для конкретного слова X_i соединяет два состояния: [4, 3]

$$h_i = [\overrightarrow{h_i}^T ; \overleftarrow{h_i}^T]^T$$

3.3.3. Декодер

Роль декодера заключается в генерации целевых слов, сосредоточившись на наиболее релевантной информации, содержащейся в исходном предложении. Для этого он использует механизм внимания. Декодер принимает каждую аннотацию и передает ее модели выравнивания, a(.), вместе с предыдущим скрытым состоянием декодера S_{t-1} Это генерирует оценку внимания:

$$E_{t,i} = a(S_{t-1}, H_i)$$

Функция, реализованная в модели выравнивания здесь, объединяет S_{t-1} и H_i с помощью операции сложения. По этой причине механизм внимания, реализованный Багданау и др., называется аддитивным вниманием. Его можно реализовать двумя способами:

1. Посредством применения матрицы весов W к соединённым векторам S_{t-1} и H_i.

2. Посредством применения матриц весов W_1 и W_2 к векторам S_{t-1} и H_i отдельно.

$$a(s_{t-1}, h_i) = v^T \tanh(W [h_i ; s_{t-1}])$$

$$a(s_{t-1}, h_i) = v^T \tanh(W_1 h_i + W_2 s_{t-1})$$

Здесь v - вектор весов. Модель выравнивания параметризована в виде многослойной нейронной сети прямого распространения и совместно обучается с оставшимися компонентами системы. Затем к каждой оценке внимания применяется функция softmax, чтобы получить соответствующий весовой коэффициент:

$$E_{t,i} = \text{softmax}(E_{t,i})$$

Применение функции софтмакс в сущности нормализует значения аннотаций в диапазоне от 0 до 1; следовательно, полученные весовые

коэффициенты могут рассматриваться как вероятностные значения. Каждое вероятностное (или весовое) значение отражает, насколько важными являются h_i и S_{t-1} в генерации следующего состояния, S_t, и следующего выхода, Y_i.

За этим следует вычисление вектора контекста в виде взвешенной суммы аннотаций: [4, 3]

$$\mathbf{c}_t = \sum_{i=1}^{T} \alpha_{t,i} \mathbf{h}_i$$

3.4. Трансформеры (Transformers)

Трансформеры - это архитектура нейронной сети, которая отслеживает взаимосвязь последовательных данных и преобразует одну последовательность в другую с помощью кодера и декодера. Инновации, лежащие в основе трансформаторов, сводятся к трем основным концепциям:

1. Позиционное кодирование (Positional Encoding)
2. Внимание (Attention)
3. Внутреннее Внимание (Self Attention)

3.4.1. Позиционное кодирование

При позиционном кодировании мы сосредоточимся на двух основных моментах: 1 – кодировщик, 2- декодер.

1. Кодировщик (Encoder):Кодер состоит из стопки из N = 6 идентичных слоев. Каждый слой состоит из двух подслоев. Первый - это механизм саморегулирования с несколькими головками, а второй

- простая, ориентированная на местоположение, полностью подключенная сеть прямой связи. Мы используем остаточное соединение вокруг каждого из двух подслоев с последующей нормализацией слоя. То есть выход каждого подслоя - это норма уровня (x + Подслой(x)), где подслой(x) - это функция, реализуемая самим подслоем. Чтобы облегчить эти остаточные соединения, все подслои в модели, а также слои внедрения выдают выходные данные размерности $d_{model} = 512$

2. Декодер (Decoder): Декодер также состоит из стека из $N = 6$ идентичных слоев. В дополнение к двум подуровням в каждом слое кодера декодер вставляет третий подуровень, который выполняет многоголовочную обработку выходных данных стека кодера. Аналогично кодировщику, мы используем остаточные соединения вокруг каждого из подуровней с последующей нормализацией уровня. Мы также модифицируем подуровень self-attention в стеке декодера, чтобы предотвратить переключение позиций на последующие. Эта маскировка в сочетании с тем фактом, что вложения выходных данных смещены на одну позицию, гарантирует, что предсказания для позиции i могут зависеть только от известных выходных данных в позициях, меньших, чем i.

3.4.2. Внимание (Attention)

Раньше в RNN мы помещали всю информацию в одно скрытое состояние, и это была своего рода seq2seq. Но, во внимание (Attention), не вся информация будет сохранена в одном скрытом состоянии, но каждое слово, основанное на соответствующем, будет иметь свое собственное скрытое состояние, и это поможет декодеру просто расшифровать его. Итак, идея, лежащая в основе этой теории, заключается в том, что для того, чтобы найти соответствующую информацию в наших предложениях, мы должны использовать эту теорию для того, что называется вниманием. [7]

3.4.3. Внутреннее Внимание

Модуль Внутреннего внимания принимает n входных данных и возвращает n выходных данных. Что происходит в этом модуле? С точки зрения непрофессионала, механизм внутреннего внимания позволяет входным данным взаимодействовать друг с другом (“я”) и выяснять, на кого им следует обратить больше внимания (“attention”). Выходные данные представляют собой совокупность этих взаимодействий и показателей внимания. [7] Во внутреннем внимании у нас есть три основных момента:

1. Ключ
2. Запрос
3. Значение

Таким образом, вес для ключа, запроса и значения должен быть умножен. Поэтому мы должны делать это для каждого входного сигнала, который у нас есть. После того, как мы проделаем эту операцию, с помощью softmax мы узнаем эти максимальные значения входных данных.

3.4.4. Преимущества трансформаторов

1. Параллельная обработка: Как мы уже говорили о параллельной обработке, RNN не может распараллеливать процессы, но трансформаторы могут легко это делать.
2. Долгосрочные зависимости: Долгосрочная зависимость, как мы обсуждали для RNN, заключается в том, что когда расстояние становится больше, RNN не может запомнить предыдущую информацию на текущем шаге. Но, к счастью, эта проблема решается трансформерами.
3. Механизм внимания (Attention Mechanism): эта функция является одним из главных преимуществ Transformers, поскольку вся информация будет сохранена в одном скрытом состоянии, и каждое

слово будет иметь свое собственное скрытое состояние, основанное на их соответствующем. [1]

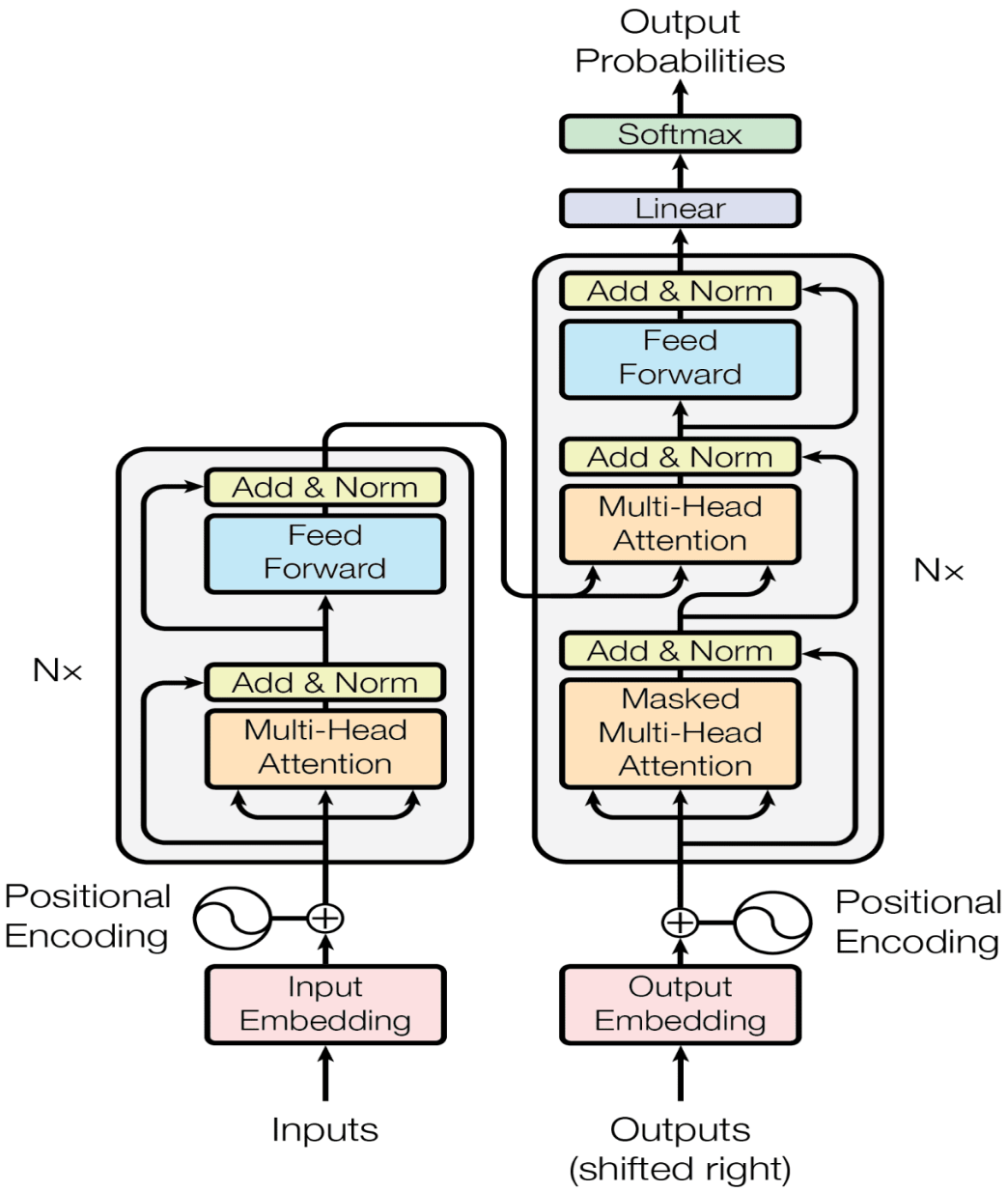

Рис. 10: Трансфомар структуры

3.5. Text-to-Text трансфер трансформер модель (T5)

Text-to-Text трансфер трансформер (T5) - это передовая модель языка, представленная Google AI в 2019 году. Это вариант модели

Transformer, которая представляет собой архитектуру нейронной сети, которая использовалась для различных задач обработки естественного языка (NLP), таких как машинный перевод, анализ настроения и извлечение текста.

T5 уникален тем, что он является моделью текста-в-текст, что означает, что он может быть обучен на широком спектре задач NLP, просто изменив форматы ввода и вывода обучаемых данных. Другими словами, T5 можно обучить на различных задачах, таких как классификация текста, вопрос-ответ и даже создание описания изображений, просто правильно форматируя данные. Архитектура T5 состоит из коди-

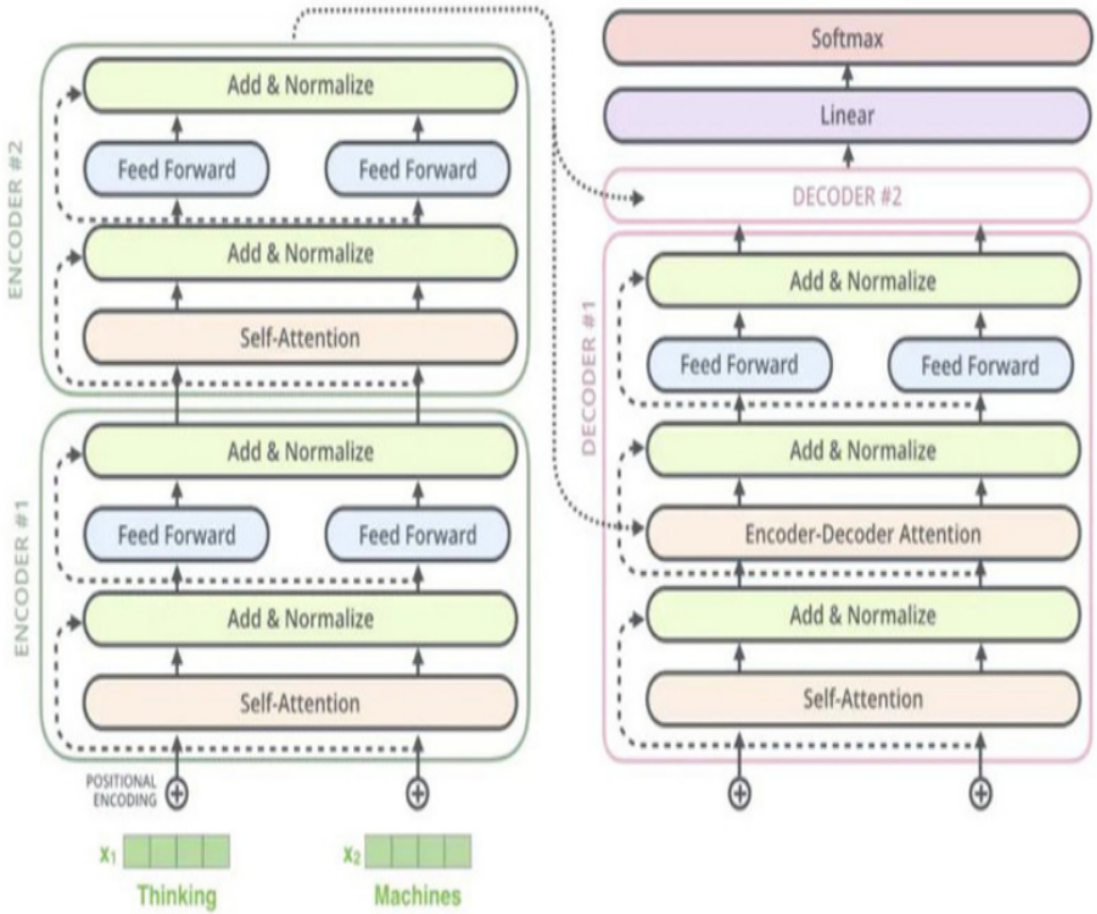

Рис. 11: T5 структуры

ровщика и декодера, как и модель Transformer. Кодировщик принимает входной текст и преобразует его в последовательность векторов,

в то время как декодер принимает эти векторы и генерирует выходной текст. Ключевое отличие T5 заключается в том, что кодировщик и декодер соединены таким образом, что позволяет модели научиться переносить информацию между разными типами входов и выходов. [6, 2]

3.6. Language Tool

LanguageTool - это многоязычный инструмент проверки орфографии и грамматики, который помогает авторам в написании текста, обнаруживая и исправляя опечатки и грамматические ошибки в различных форматах написания, включая блоги, веб-сайты, книги и другие. Точно так же несколько текстовых редакторов используют основанный на правилах подход к исправлению ошибок, включая LanguageTool. Таким образом, следующее обсуждение будет посвящено основанному на правилах модели, используемой LanguageTool. [8]

3.6.1. Подход на основе правил

Подход на основе правил в области искусственного интеллекта (ИИ) опирается на набор заранее определенных правил для определения следующего шага. Этот подход включает использование набора входов и набора правил для создания вывода. Система сначала определяет применимое правило на основе входных данных. Если правило совпадает с входными данными, система выполняет соответствующие шаги для получения вывода. В противном случае система может создать ответ по умолчанию или запросить у пользователя дополнительную информацию. [22]

3.6.2. Подход на основе правил компоненты

Основные компоненты системы правил:

1. База знаний (Knowledge base): Это хранилище правил, фактов и предметно-специфичной информации, используемой системой

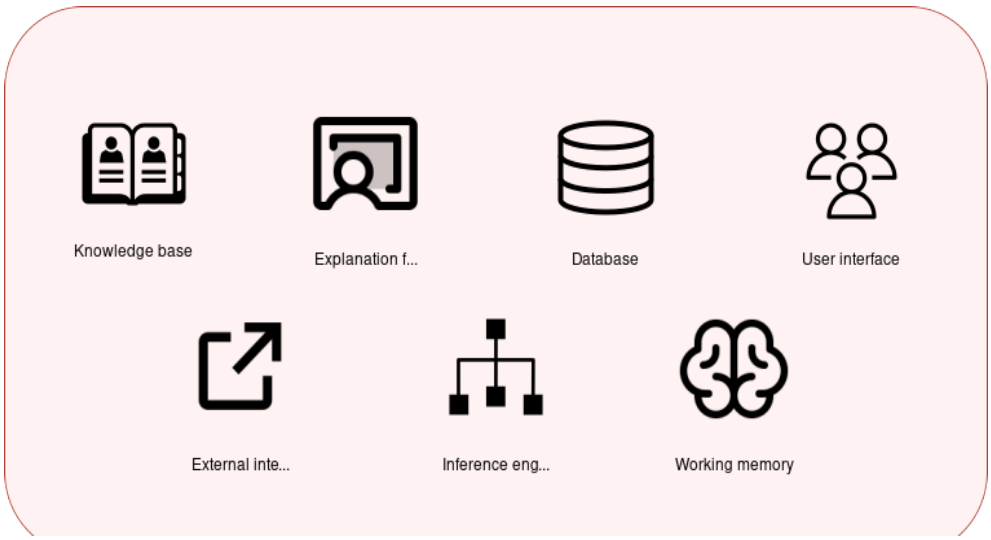

Рис. 12: Основные компоненты

правил для принятия решений, предоставляя необходимую информацию для логического анализа и сопоставления правил. [18]

2. Механизм объяснения (Explanation facilities): Он генерирует обоснования или объяснения решений системы, повышая прозрачность и помогая пользователям понять логику, лежащую в основе выводов системы, что повышает доверие и интерпретабельность. [18]

3. База данных (Database): Она хранит соответствующую информацию, используемую системой правил, например, входную информацию или исторические записи, предоставляя источник данных для процесса вывода и позволяя принимать решения, основанные на данных. [18]

4. Интерфейс пользователя (User interface): Он позволяет пользователям взаимодействовать с системой правил, предоставляя возможность вводить данные, изменять правила и получать результаты или рекомендации, что упрощает взаимодействие с пользователем и удобство использования системы. [18]

5. Внешний интерфейс (Externel interface): Он обеспечивает возможность связи и интеграции с внешними системами или службами,

позволяя обмениваться данными, взаимодействовать с другими программными компонентами или интегрироваться с внешними источниками для получения входов или доставки выходных данных. [18]

6. Механизм вывода (Inference engine): он обрабатывает правила и данные из базы знаний, применяя логическое рассуждение и сопоставление правил для определения подходящих действий или выводов на основе данных входов. [18]

7. Рабочая память (Working memory): она временно хранит текущее состояние системы во время процесса вывода, сохраняя входные данные, промежуточные результаты и полученные выводы, обеспечивая необходимую контекст для сопоставления правил и облегчая процесс принятия решений. [18]

Более того, учитывая поддержку LanguageTool корректоров правописания и грамматики более чем на 25 языках, а также его API для премиум-пользователей и существующую библиотеку, было решено интегрировать библиотеку с развернутой моделью в качестве дополнительной опции для пользователей, желающих исправить грамматические ошибки. Процесс интеграции был завершен, и пользователи теперь могут легко выбрать один из двух представленных вариантов. Кроме того, планируется, что развернутая модель будет генерировать API для пользователей, которые хотят использовать ее в качестве вспомогательной модели для исправления грамматических ошибок.

4. Эксперимент

Мы оцениваем предлагаемые подходы на задаче исправления грамматики. Мы используем набор данных Lang-8 Learner Corpora для наших моделей. Мы используем одинаковые процедуры обучения и одинаковый набор данных для всех наших архитектур.

4.1. Условия эксперимента

Чтобы убедиться в эффективности наших обученных моделей, мы внимательно рассмотрим их производительность с помощью специально предназначенного модуля. После завершения оценки мы можем быстро определить модель, которая демонстрирует превосходные характеристики по сравнению со своими аналогами. Данная оценка будет проводиться с помощью показателя GLEU, который обеспечивает всеобъемлющее измерение качества нашего сгенерированного текста.

4.1.1. Набор данных

Корпус Learner Lang-8 содержит тексты английских изучающих, извлеченные из Lang-8. В нём содержится 100 051 английский текст, написанный 29 012 активными пользователями.

После кодирования и декодирования токенизации мы получили 51 949 слов с помощью кодировщика токенизатора и 43 124 слова с помощью декодера токенизатора.

4.1.2. Модель

Мы обучим 2 модели и отточим 1 модель. Первые 2 модели архитектуры, которые мы собираемся обучить, это: 1 - LSTM, архитектура механизма внимания от Bahdanau.
Для обучения архитектуры LSTM мы учитывали следующие параметры:

1. Размерность встраивания: 100

2. Размерность входных данных для кодирования: 12
3. Размерность входных данных для декодирования: 13
4. Размер батча: 1024

Для обучения механизма внимания Bahdanau мы учитывали следующие параметры:

1. Размерность встраивания: 300
2. Длина входных данных для кодирования: 12
3. Размер LSTM: 192
4. Размер батча: 1024
5. Единицы внимания: 192

Для тонкой настройки нашей модели T5 мы учитывали следующие параметры:

1. Размер батча: 64
2. Скорость обучения: 2e-5
3. Количество эпох обучения: 1
4. Этап оценки: 2000

Для оптимизации упомянутых моделей мы использовали оптимизатор Adam, и обучение и тонкая настройка каждой из этих моделей занимали 8 часов.

4.2. Вопрос исследования

- RQ1: Какая из рассмотренных моделей (LSTM, механизм внимания Бахадана, T5) имеет хорошую производительность для корректора грамматики?

4.3. Метрики

Для оценки упомянутых архитектурных моделей мы будем применять показатель GLEU. Более подробно о нашей метрике мы расскажем ниже.
GLEU просто представляет собой минимум точности и полноты. Диапазон GLEU всегда находится между 0 (нет совпадений) и 1 (все совпадения), и при переключении выходных данных и целей он симметричный. [16]

4.4. Результат

В этой части мы представляем результат нашего эксперимента, отвечая на вопрос исследования:

- Какая из рассмотренных моделей (LSTM, механизм внимания Bahdanau, T5) имеет хорошую производительность для корректора грамматики?

Оценка GLEU	
Модель	Метрика
LSTM	0.217
Миханизм Внимания	0.319
T5	0.418

Таблица: Оценка GLEU

Итак, если мы посмотрим на таблицу выше, мы поймем, что архитектура трансформера модели T5 имеет лучшую производительность по сравнению с другими моделями.

Заключение

На данный момент выполнены следующие задачи:

1. Изучены другие различные модели NLP, которые могут анализировать предложения, такие как: LSTM, Механизм Внимания (Bahdanau), T5.

2. Выбрана модель T5 в качестве отличительной модели для выполнения макро-замены предложений.

3. Проведены эксперименты с 3 моделями: LSTM, Механизм Внимания (Bahdanau), T5.

4. Инструмент подключен к библиотеке Language tool для исправления предложений.

5. Завершен CI pipeline с использованием GitHub Actions.

6. Реализирован прототип для выполнения макро-замены предложений.

7. Исходный код доступен в: https://github.com/Hurmatullah/English-Grammar-Corrector.git

Список литературы

[1] Aidan N, Ashish, Gomez, Illia, Jakob, Jones, Kaiser, Llion, Lukasz, Niki, Noam, Parmar, Polosukhin, 'Shazeer', Uszkoreit, Vaswani. Attention is all you need. — 2017. — Vol. 30.

[2] Aoudi Yousra. T5: A State-of-the-Art Text-to-Text Transfer Transformer for Natural Language Processing. — Access url - https://www.my-ai-platform.com/post/t5-a-state-of-the-art-text-to-text-transfer-transformer-for-natural-language-processing, Online accessed - 2023.

[3] Bahdanau Dzmitry, Cho Kyunghyun, Bengio Yoshua. Neural machine translation by jointly learning to align and translate // arXiv preprint arXiv:1409.0473. — 2014.

[4] Brownlee Jason. The Bahdanau Attention Mechanism. — Access url - https://machinelearningmastery.com/the-bahdanau-attention-mechanism/, Online accessed - 2023.

[5] Calzone Ottavio. An Intuitive Explanation of LSTM. — Access url - https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c: :text=LSTMOnline accessed - 2022.

[6] Exploring the limits of transfer learning with a unified text-to-text transformer / Colin Raffel, Noam Shazeer, Adam Roberts et al. // The Journal of Machine Learning Research. — 2020. — Vol. 21, no. 1. — P. 5485–5551.

[7] Giacaglia Giuliano. How Transformers Work. — Access url - https://towardsdatascience.com/transformers-141e32e69591, Online accessed - 2019.

[8] Gina. LanguageTool. — Access url - https://languagetool.org/insights/post/artificial-intelligence/, Online accessed - 2023.

[9] Guolin Ke, Qi Meng, Qiwei Ye, Taifeng Wang, Thomas Finley, Wei Chen, Weidong Ma. LightGBM: A Highly Efficient Gradient Boosting Decision Tree." Advances in Neural Information Processing Systems 30 (NIPS 2017) - 2015, pp - 3149-3157.

[10] Hochreiter Sepp, Schmidhuber Jürgen. Long Short-term Memory // Neural computation. — 1997. — 12. — Vol. 9. — P. 1735–80.

[11] IBM. Что такое рекуррентные нейронные сети? — Access url - https://www.ibm.com/topics/recurrent-neural-networks, Online accessed - 2022.

[12] Korobov Mikhail. Morphological analyzer and generator for Russian and Ukrainian languages. — 2015. — P. 320–332.

[13] LemmInflect. LemmInflect. — https://lemminflect.readthedocs.io/en/latest//, Online accessed - 2022.

[14] NLTK. Рекурсия в синтаксической структуре. — Access url - https://www.nltk.org/book/ch08.html, Online accessed - 2022.

[15] NLTK. Синтаксический анализ рекурсивного спуска. — Access url - https://www.nltk.org/book/ch08.html, Online accessed - 2022.

[16] NLTK. GLEU Score Module. — Access url - https://www.nltk.org/api/nltk.translate.gleu$_score.html, Onlineaccessed$ — 2023.

[17] Nigam Vibhor. From Basics to using RNN and LSTM. — Access url - https://medium.com/analytics-vidhya/natural-language-processing-from-basics-to-using-rnn-and-lstm-ef6779e4ae66, Online accessed - 2022.

[18] Noor Ammara. What are rule-based systems in AI? — Access url -https://www.educative.io/answers/what-are-rule-based-systems-in-ai, Online accessed - 2023.

[19] P. V. Panicheva, A. P. Mirzagitova, E. B. Protopopova, O. A. Mitrofanova. Разработка лингвистического комплекса для морфологического анализа русскоязычных корпусов текстов на основе Pymorphy и NLTK, Труды международной конференции “Корпусная лингвистика”, СПб - 2015, С - 361-373. — 2015.

[20] P. V. Panicheva, E. B. Enikeeva Protopopova, A. P. Mirzagitova, A. D. Moskvina, O. A. Mitrofanova. Проект NLTK4Russian. — Access url - http://mathling.phil.spbu.ru/node/160, Online accessed - 2022.

[21] Pymorphy2. Морфологический анализатор pymorphy2. — Access url - https://pymorphy2.readthedocs.io/en/stable/, Online accessed - 2022.

[22] Shah Eshika. Rule-based Systems in AI. — Access url - https://www.scaler.com/topics/artificial-intelligence-tutorial/rule-based-system-in-ai/, Online accessed - 2023.

[23] К.Ю.Романовский. Метод повторного использования документации семейств программных продуктов / К.Ю.Романовский ; СПбГУ. — 2010.

[24] Технопедия. Что означает Natural Language Toolkit (NLTK)? — Access url - https://www.techopedia.com/definition/30343/natural-language-toolkit-nltk, Online accessed - 2022.