|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство образования и науки Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления (ИУ)

КАФЕДРА Системы обработки информации и управления (ИУ5)

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***

***НА ТЕМУ:***

***Анализ текстов на основе векторного представления графов знаний***

Студент ИУ5-43М **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** А.В. Иванников

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель ВКР **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Ю.Е. Гапанюк

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Нормоконтролер **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2020 г.*

**РЕФЕРАТ**

Эта работа посвящена исследованию методов синтаксического анализа предложения на русском языке. В работе определяются подходы к решению задачи построения синтаксического дерева предложения, а также синтезируются алгоритм и модель преобразования предложения в синтаксическое дерево.

Актуальность работы обусловлена развитием предметной области анализа текстов в последние несколько лет. Использование метаграфов для анализа текста поможет создать интеллектуальную систему понимания текста, а формирование синтаксического графа является одной из частей построения такого метаграфа.

В исследовательской части данной работы рассматриваются понятия синтаксического анализа и построению алгоритма преобразования предложения в синтаксический граф. Также производится обзор несколько методов решения задачи и выбор наиболее современного и подходящего – разработка архитектуры нейронной сети. В последние год машинное обучение оказало существенное влияние на развитие науки в области анализа изображений и текстов, поэтому такой подход является предпочтительным, что подтверждено результатами экспериментов.

В практической части работы проводятся эксперименты по преобразованию предложения на русском языке в синтаксическое дерево и оценка результатов преобразования. Разработанный автором алгоритм оказался успешным в решении задачи, показав наивысший результат по метрикам, применяемым к подобным системам.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_Toc44844414)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 8](#_Toc44844415)

[1.1. Вычислительные мощности для обработки метаграфов 8](#_Toc44844416)

[1.2. Метаграфовая модель данных 9](#_Toc44844417)

[1.3. Основны метаграфовой модели 9](#_Toc44844418)

[1.4. Обзор алгоритмов текстового анализа 13](#_Toc44844419)

[2 РАЗРАБОТКА МОДЕЛИ СИНТАКСИЧЕСКОГО АНАЛИЗАТОРА 19](#_Toc44844420)

[2.1. Особенности технологий синтаксического анализа 19](#_Toc44844421)

[2.2. Пример синтаксического анализатора 21](#_Toc44844422)

[2.3. Машинное обучение и нейросетевые алгоритмы 33](#_Toc44844423)

[2.4. Основные методы машинного обучения для NLP 37](#_Toc44844424)

[2.5. Алгоритм BERT для задач анализа текста 45](#_Toc44844425)

[2.6. Адаптация модели BERT под задачу 49](#_Toc44844426)

[2.7. Формулировка задачи 50](#_Toc44844427)

[2.8. Создание модели синтаксического анализа 51](#_Toc44844428)

[3 ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ 54](#_Toc44844429)

[3.1. Исходные данные и обучающая выборка 54](#_Toc44844430)

[3.1.1. Наборы данных в нейросетевых алгоритмах 54](#_Toc44844431)

[3.1.2. Национальный корпус русского языка 57](#_Toc44844432)

[3.1.3. SynTagRus – синтаксический корпус 59](#_Toc44844433)

[3.2. Обучение нейронной сети 61](#_Toc44844434)

[3.3. Формирование синтаксического дерева 70](#_Toc44844435)

[3.4. Результаты 73](#_Toc44844436)

[ЗАКЛЮЧЕНИЕ 74](#_Toc44844437)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 75](#_Toc44844438)

[ПРИЛОЖЕНИЕ A 83](#_Toc44844439)

[ПРИЛОЖЕНИЕ Б 90](#_Toc44844469)

# ВВЕДЕНИЕ

Задача анализа текста существую многие годы, а в последнее время, с развитием искусственного интеллекта, чат-ботов, а также широкого распространения социальных сетей и мессенджеров, анализ текста и понимание естественного языка стали одними из важнейших направлений научных и прикладных исследований в области искусственного интеллекта и машинного обучения. Существует множество подходов к решению этой задаче, но пока ни один из них не применяется для решений широкого круга задач анализа текст. Одним из возможных прорывных решений в этой области является использование графовых моделей для структурирования текста. Графы хорошо изучены, обладают сильной структурой и взаимосвязью внутренних компонент, а самое главное – существуют естественным образом, например, социальный граф, диффузионный график в социальных сетях, граф цитирования в областях исследований, граф интересов пользователей в области электронной торговли, граф знаний и т. д. В последние десятилетия работа с графами получила новый виток развития – так как с появлением мощных вычислительных машин и процессоров, реальным стало решений задачи анализа графа.

Анализ графов может принести пользу многим приложениям, например, путем классификации узлов, группировки узлов, определения местоположения и рекомендации узлов, предсказания ссылок и т. д. Анализируя граф на основе взаимодействия с пользователями в социальной сети, мы можем классифицировать пользователей. Исследуйте сообщества, рекомендуйте друзей и прогнозируйте взаимодействие между двумя пользователями. Чтобы привести данные в графически структурированную форму, необходимо, чтобы данные были представлены диаграммами в контексте задач, которые необходимо решить. Важно отметить, что каждый объект может быть сведен к компонентам и может быть описан с некоторой точностью с использованием структур диаграмм. Однако это описание должно быть непосредственно связано с задачей. Этот подход более продуктивен для задачи с различными сетями, где исходные данные имеют структуру, описанную диаграммами. упомянутые социальные сети, веб-графика, биологические сети, сеть транспортных маршрутов и так далее. Также важно отметить, что эти сети очень большие и их сложно структурировать. Поэтому вопрос применения обобщенных структурных диаграмм для этого типа объекта является более чем актуальным, а также вопрос представления элементов диаграммы в малой форме и для использования в качестве входных данных для подходящих алгоритмов для машинного обучения и глубокого обучение. Обоснование имеет несколько преимуществ: уменьшение размера исходных данных без потери информации, уменьшение сложности интерпретации промежуточных и конечных результатов, повышение уровня абстракции рассматриваемого объекта и возможность новой постановки задачи с возможной вероятностью. другие методы для использования.

Модель метаграфа имеет значительный уровень абстракции, который, как упоминалось выше, актуален в сложных сетях. Используя предлагаемую автором методику встраивания (эмбеддинга) сложной метаграфовой модели, можно представить иерархичную и максимально общую модель метаграфа в виде набора низкоразмерных векторов, которые, в свою очередь зачастую служат видом входных данных в множестве структур, нейросетях и других алгоритмах машинного и глубокого обучения. Особенно актуально это для задач работы с текстом, так как любой текст сначала необходимо представить в виде вектора или даже массива векторов слов.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Вычислительные мощности для обработки метаграфов

Современные научные исследования и все связанные с ними успехи в области машинного обучения частично связаны со стремительным развитием процессорных технологий. Вычислительная мощность процессоров возросла в несколько раз, а параллельное вычисление позволило по-настоящему раскрыть потенциал нейронных сетей. В настоящее время существует несколько потенциальных вариантов развития в этой области.

1. Увеличение “ядерности” процессоров – многоядерные и многопроцессорные архитектуры уже давно применяются, но в скором времени, из-за нехватки мест на интегральных микросхема, количество ядер будет возрастать кратно с каждым новым поколением процессоров. Хотя совсем недавно ведущие мировые производители анонсировали производство 12 нанометровых микросхем [1], пока эта технология остается на уровне научных исследований и не применяется на практике;
2. Параллельные вычисления. Видеопроцессоры или графические процессоры (GPU) еще несколько лет назад применялись для игровой индустрии, однако особенности их архитектуры позволяют проводят огромное количество параллельных вычислений [2], что в свою очередь необходимо для быстрого обучения и работы нейросетей. Компани NVIDIA – мировой лидер производства графических процессоров собирается выпустить на рынок новую линейку своего продукта, что приведет к еще более стремительному развитию скорости работы алгоритмов машинного обучения.

Для обработки большого массива текстов необходимо применение параллельных вычислений, так как сформированный метаграф может иметь огромное количество вершин и связей

## Метаграфовая модель данных

Граф является одним из базовых и простейших моделей представления данных и связей. Графы используются повсеместно, однако зачастую, ограничения графовой модели не позволяют отобразить полный контекст модели. Примером такой модели является текст. Как расположить вместе синтаксическую и смысловую связь между словами в предложении, а также смысловую и контекстную между предложениями в тексте, а далее между текстами [3].

Решением этой проблемы может являться сложная сеть, в этом случае вершина графа в свою очередь может являться графом, фрагментом, состоящим из вершин и связей, но выступающий как единое целое для графа верхнего уровня

Идея метаграфовой модели была впервые предложена на кафедре «Системы обработки информации и управления» МГТУ им. Н.Э. Баумана [4]. Эта модель может быть применена для описательной характеристики сложных сетей, для формализации семантико-прагматичных свойств информационных систем, а также с помощью модели описывается архитектура гибридных информационных систем.

Также были рассмотрены варианты операций над метаграфами, их описание и моделирование.

## Основны метаграфовой модели

Множество идей и теоретических аспектов для метаграфовой модели заложили работы ученых А. Базу и Р. Блэннинга, которые представили необходимую базы для дальнейшего исследования в данном направлении.

В данном разделе кратко рассмотрим формализованную модель метаграфа, ее математическое описание, основные свойства и характеристики [5].

Рассмотрим формализованное представление метаграфовой модели:

(1)

где *V* – множество вершин метаграфа; – метаграф; *E* – множество ребер метаграф; *MV* – множество метавершин метаграфа.

Вершина метаграфа это множество атрибутов:

(2)

где – вершина метаграфа; – атрибут.

Множеством атрибутов, исходной и конечной вершиной и признаком направленности называется ребро метаграфа:

(3)

где – исходная вершина (метавершина) ребра; – ребро метаграфа; – признак направленности ребра ( – направленное ребро, – ненаправленное ребр; – конечная вершина (метавершина) ребра; ­– атрибут.

Фрагмент метаграфа:

, (4)

где – фрагмент метаграфа; – элемент, принадлежащий объединению множеств вершин, метавершин и ребер метаграфа.

Исходя из описания, фрагмент метагарфа может вкоючать в себя вершины, множество вершин, ребра без ограничения.

Метавершина метаграфа:

(5)

где – метавершина метаграфа, принадлежащая множеству вершин *MV*; – атрибут, – фрагмент метаграфа.

Обобщив написанное выше, метавершина – это вложенный граф, который может включать в себя вершины и ребра, но предстает как единое целое для графа верхнего уровня.

Как раз это свойство метавершины – наличие в себе других вершин, атрибутов и ребер является важной и главной особенностью метаграфа. Это соответствует принципу эмерджентности, то есть приданию понятию нового качества, несводимости понятия к сумме его составных частей [6]. Фактически, понятие метавершины и определяет поянтие метаграфа – графа, состоящего из метавершин.

Пример описания небольшого метаграфа показан на рисунке 1.1.



Рис. 1.1– Пример описания метаграфа

Данный метаграф содержит вершины, метавершины и ребра. На рис. 1 показаны четыре метавершины: mv1, mv2, mv3 и mv4. Метавершина mv1 включает вершины v1, v2, v3 и связывающие их ребра e1, e2, e3. Метавершина mv2 включает вершины v4, v5 и связывающее их ребро e6. Ребра e4, e5 являются примерами ребер, соединяющих вершины v2-v4 и v3-v5, включенные в различные метавершины mv1 и mv2. Ребро e7 является примером ребра, соединяющего метавершины mv1 и mv2. Ребро e8 является примером ребра, соединяющего вершину v2 и метавершину mv2. Метавершина mv3 включает метавершины mv2 и mv4, вершины v2, v3 и ребро e2 из метавершины mv1 а также ребра e4, e5, e8. Метавершина mv4 включает не соединенные ребрами вершины v6 и v7.

В качестве активного элемента метаграфовой модели используется метаграфовые агенты. Определим метаграфовой агент следующим образом:

(6)

где , , на основе которого выполняются правила агента, т.е. рабочий метаграф. стартовое условие выполнения агента (фрагмент метаграфа, который используется для стартовой проверки правил, или стартовое правило).

Структура правила метаграфового агента:

. (7)

Метаграфовые правила можно разделит на замкнутые и разомкнутые правила. Замкнутые правила не меняют фрагмент метаграф, в отличие от разомкнутых, их можно сравнить как входной и выходной фрагмент метаграфа.

## Обзор алгоритмов текстового анализа

Текстовый анализ — это обширная и тяжелая задача. Более конкретно можно рассмотреть две темы – смысловой и синтаксический анализ. В этой работе далее будет рассмотрен синтаксический анализ текста. В свою очередь смысловой анализ подразумевает понимание связи между понятиями в тексте, концепциями предложений и параграфов, а также создание общего контектса [6].

Истина в тексте связывается с субъектом, задающем конструктивную и кондициональную субъектность смысла. В тексте выявляются системы смысла: претекст, номинальное содержание, реальное, деконструктивное, эпохальное, этническое, последействие текста, собственное намерение текста и эти смыслы специфицируются в модели понимания и стратегий сообщения и приема информации конкретными субъектами социальной структуры.

Текст имеет аспекты семантики, синтаксиса, прагматики (бытие, мышление, коммуникация), создавая гармонизуемую модель множественности смысла. Замечу, что это неполное определение текста [7]. Текст имеет расово-этнический и языковый претекст априорного характера, который предшествует семантике, синтаксису и прагматике, номинальное, реальное, деконструктивное, эпохальное, этническое (русское) содержание, а также имеет ближние, дальние, средние последствия своего бытования в обществе. Если иметь ввиду синтаксис, прагматику и семантику в целом, то истина в предложении определяется личными местоимениями и глагольными основами, но в языках, где слабы личные местоимения нет дифференциации на субъект и объект, предикат, а есть целостное действие как локально - общее и диффузное общее. Текст - в этническом языке и психотипе, психогенетическом типе и КПЛ - целостность смысла, значения, значимостей, когезия, опорные и ключевые слова, членимость, завершенность в некотором жанре в аспектах семантики, синтаксиса, прагматики, сюжета и фабулы, композиции, интриги действия. Текст подлежит пониманию, которое основывается на нормативности (нормы жанра, предметные и коммуникативные).

В современной науке зачастую применяют корпусную лингвистику [8]. Основная идея этого подхода – собрать большой набор данных – документов, словарей – корпус, чтобы затем, с помощью методов машинного обучения и статистического анализа построить систему, способную решить некоторую задачу. Хотя этот подход и оказался прорывным для многих сфер, первой из которых можно выделить так называемых чат-ботов – программы, способные выполнить простые текстовые команды пользователя, отвечать ему и поддерживать простой разговор, все же невозможно представить себе анализ текста и его смыслов, опираясь лишь на массив документов. Как невозможно понять идею великих произведений рассматривая лишь отдельные слова, из которых оно состоит, так и невозможно проанализировать любой текст, пользуясь лишь обобщенной моделью, синтезированной из огромного объема текстов.

Итак, обработка текста с помощью методов машинного обучения выглядит следующим образом:

1. Текст приводится к каноничной форме. Термин “каноничная форма” подразумевает исправление орфографических ошибок и опечаток, стандартизация – рассматриваем слова в основной форме, без учета многообразия приставок, суффиксов окончаний [9]. Все слова должны быть найдены в словаре, а, следовательно, находится только в основной форме. Для исправления ошибок можно использовать два подхода. Первый – фонетический матчинг. Суть метода состоит в записи слова в форме, котором оно пишется и в форме, в котором оно слышится, тогда ошибка, совершенная человеком (от того, что он записал слово “как слышится”) будет нивелирована. Второй подход – редакционное расстояние или расстояние левенштейна [10]. Этот подход подразумевает поиск в словаре максимально похожих слов, то есть слов, получить которые из исходного можно выполнив наименьшее количество преобразований. Например, для слов “превозмочь” и “привозмоч” это расстояние будет равно 2 (необходимо поменять букву “и” на “е” и добавить мягкий знак в конце.
2. Стемминг – нахождение основы слова. Как было объяснено ранее, мы можем работать только с основной формой слова, а значит все возможные варианты изменения слова необходимо преобразовать к простейшему. Например, “преобразившийся” состоит из приставки (префикса) “пре”, суффиксов “и”, “вш”, окончания “ий”, постфикса “ся” и корня “образ”. Основой в данном случае может являться как корень “образ”, так и сложный составной корень “преобраз”. Основная проблема подхода: правила для стеммера устанавливают лингвисты, и это достаточно тяжёлый труд. Перед подключением нового языка нужны лингвистические исследования. Есть разновидности подхода. Мы можем или просто делать поиск по словарю, или строить supervised-модели без учителя, опять же — вероятностные модели на основе скрытых цепей Маркова [11], или обучать нейросети, которые приведут слова в редуцированную форму. Стемминг используется достаточно давно. В Google — с начала 2000-х. Самый распространённый, наверное, инструмент — реализация в пакете Apache Lucene. Но у стемминга есть недостаток. Когда мы урезаем слово до основы, мы лишаемся части информации. Потому что у нас остаётся лишь корень, и мы можем потерять данные о том, было ли это прилагательное или существительное. И иногда это важно для постановки дальнейших задач. Вторая концепция, альтернатива стемминга — лемматизация. Она пытается привести слово не к основе или корню, а к базовой, словарной форме — т. е. лемме. К примеру, глагол — к инфинитиву. Существует множество реализаций, и тема очень хорошо проработана именно для user generated текстов, пользовательский зашумлённых текстов [12]. Однако приведение в каноническую форму по-прежнему остаётся сложной и до конца не решённой задачей.
3. Векторизация. Так как компьютеры пока не научились работать напрямую с человеческим языком, а только лишь с информацией [13], представленной в виде чисел (байтов), то для работы алгоритмов машинного обучения необходимо перевести слова в векторы – отображение слова в векторном пространстве [14]. Мы формируем для документа вектор в пространстве, размерность которого равна размеру нашего словаря. Для каждого слова выделена своя размерность, и для документа мы записываем признак, насколько часто это слово в нём использовалось. Получаем вектор. Есть много подходов к тому, как его выяснить. Доминирует так называемый TF-IDF. Частоту слова (term frequency, TF) определяют по-разному. Это может быть счётчик вхождения слова. Или флаг, видели мы слово либо нет. Или что-то чуть более хитрое, например, логарифмически сглаженное количество упоминаний слова. И вот что самое интересное. Определив TF в документе, мы перемножаем её с обратной частотой документа (inverse document frequency, IDF). IDF обычно вычисляют как логарифм от числа документов в корпусе, разделённый на количество документов, где это слово представлено. Вот пример. У нас встретилось слово, которое употреблялось во всех-всех-всех документах корпуса. Очевидно, логарифм даст нам ноль. Такое слово мы никуда не добавим: оно не несёт никакой информации, оно есть во всех документах. В чём преимущество подхода мешка слов? Его просто реализовать. Но он теряет часть информации, в том числе сведения о порядке слов. И сейчас продолжают ломать много копий на тему того, насколько важен порядок слов. У нас есть один известный пример — магистр Йода. Он расставляет слова в предложении хаотично. Речь Йоды необычна, но мы её свободно понимаем: т. е. человеческий мозг достаточно легко восстанавливает информацию даже при потерянном порядке. Но иногда эта информация значима. Например, при анализе эмоциональной окраски очень важно, к чему относилось, условно говоря, слово «хороший» или «нет». Тогда наряду с мешком слов поможет мешок N-грамм: мы добавляем в словарь не только слова, но и словосочетания. Мы не будем вносить все словосочетания, потому что это приведёт к комбинаторному взрыву, но часто используемые статистически значимые пары или пары, соответствующие именованным сущностям, можно добавить, и это повысит качество работы итоговой модели [15]. Другой пример ситуации, когда «мешок слов» может терять или искажать информацию — слова синонимы или слова со несколькими различными смыслами (например, замок). Отчасти эти ситуации позволяют обработать методы построения "векторных представлений слов", например, знаменитый word2vec или более модные skip-gramm.
4. Дедубликация. Нам не нужны повторения одних и тех же слов, так как мы уже взяли от них лишь основы. У нас есть векторы в векторном пространстве, мы можем определить их близость [16], взять косинус, можем другие метрики близости, но обычно используют именно косинус. Объединим в общую группу документы, где косинус близок к единице. Казалось бы, всё просто, понятно, но есть одно но: у нас 2 миллиарда документов. Если мы умножим 2 миллиарда на 2 миллиарда, то никогда не закончим считать косинусы. Нужна оптимизация, которая позволит быстро выбрать кандидатов для расчета косинуса, избавившись от полного перебора. И здесь поможет локально-чувствительный хеш. Стандартные хеш-функции равномерно размазывают данные по пространству хэшей. Локально-чувствительный хеш похожие объекты поместит в пространстве объектов близко. С какой-то вероятностью он вообще может дать им один и тот же хеш. Есть много техник подсчёта локально-чувствительного хеша под разные метрики похожести. Если речь идёт о косинусе, то часто используется метод случайных проекций. Мы выбираем случайный базис из случайных векторов. Считаем косинус нашего документа с одним из векторов базиса. Если он больше нуля, то ставим единичку. Меньше нуля или равен ему — ставим ноль. Дальше сравниваем со вторым вектором базиса, получаем ещё один нолик или единичку. Сколько у нас векторов в базисе — столько мы в итоге получаем битов, и это наш хеш. В чём преимущество, почему это вообще работает? Если два документа близки по косинусу друг с другом, то с высокой вероятностью они окажутся по одну и ту же сторону от вектора базиса. Поэтому у похожих документов хеш с высокой вероятностью окажется один. Тем не менее выбросы будут. Чтобы их исправить, мы просто повторим процедуру. На практике мы обычно используем два прогона. На первом вычисляем 24-битный хеш и удаляем много почти идентичных документов. Дальше считаем ещё один хеш на другом базисе, но уже в 16 бит и удаляем дубликаты. После этого копий не остаётся — либо же их настолько мало, что они не могут значимо отразиться на качестве работы моделей [17].

Таким образом, главной проблемой обработки текстов, на сегодняшний день является сложность понимания контекста, то есть нахождение одновременно синтаксической и семантической взаимосвязи в тексте. Составить синтаксический и семантический граф можно современными методами машинного обучения, а затем необходимо объединить эти графы в метаграфовую модель для дальнейшего исследования.

# РАЗРАБОТКА МОДЕЛИ СИНТАКСИЧЕСКОГО АНАЛИЗАТОРА

## Особенности технологий синтаксического анализа

Синтаксический анализ или синтаксический анализ является третьей фазой NLP. Цель этого этапа - нарисовать точное значение, или вы можете сказать значение словаря из текста. Синтаксический анализ проверяет текст на предмет осмысленности по сравнению с правилами формальной грамматики. Например, предложение типа "горячее мороженое" будет отвергнуто семантическим анализатором. [18]

В этом смысле синтаксический анализ или синтаксический разбор можно определить, как процесс анализа строк символов на естественном языке в соответствии с правилами формальной грамматики. Происхождение слова "parsing" происходит от латинского слова "pars", что означает "part". Используется для реализации задачи парсинга. Его можно определить, как программный компонент, предназначенный для приема входных данных (текста) и предоставления структурного представления входных данных после проверки правильности синтаксиса по формальной грамматике [19]. Он также строит структуру данных, как правило, в виде дерева разбора или абстрактного синтаксического дерева или другой иерархической структуры.

Общая структура парсера представлена на рисунке 2.1.

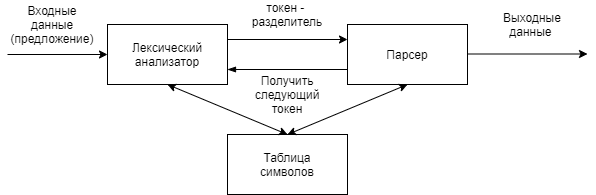
****

Рис. 2.2– Обобщенная структура парсера

Основные роли анализа включают в себя:

* Сообщение о любой синтаксической ошибке.
* Восстановление после часто встречающейся ошибки, чтобы можно было продолжить обработку оставшейся части программы.
* Создание дерева разбора.
* Создание таблицы символов.
* Создание промежуточных представлений (IR).

Деривация делит разделение на следующие два типа:

* Парсинг сверху вниз
* Анализ снизу-вверх

Парсинг сверху вниз

При таком разборе синтаксический анализатор начинает строить дерево разбора от стартового символа, а затем пытается преобразовать стартовый символ во входной [20]. Наиболее распространенная форма разбора сверху вниз использует рекурсивную процедуру для обработки входных данных. Основным недостатком рекурсивного разбора спуска является обратная траектория.

Анализ снизу-вверх

При таком синтаксическом анализе синтаксический анализатор начинает с входного символа и пытается построить дерево синтаксического анализатора до стартового символа.

Концепция Деривации

Для получения входной строки нам нужна последовательность правил производства. Деривация - это набор правил производства [21]. Во время синтаксического анализа нам необходимо решить, какой термин будет заменен, а также принять решение о правиле производства, с помощью которого будет заменен термин. Также выделяют два вида деривации. В крайнем левом производном отсканирована и заменена слева на справа отправленная форма входа. Форма отправки в данном случае называется форма отправки слева. В крайнем левом производном отсканирована и заменена справа налево отправленная форма входа. Форма отправки в данном случае называется форма отправки справа.

Концепция Дерева парсинга [22]. Его можно определить, как графическое изображение производной. Корнем дерева разбора служит начальный символ деривации. В каждом дереве разбора узлы листа являются терминалами, а внутренние узлы - не терминалами. Свойство дерева парсинга заключается в том, что при обходе по порядку получается исходная входная строка.

## Пример синтаксического анализатора

Синтаксический анализатор должен распознать структуру предложения, а именно синтаксические зависимости слов. В результате должно быть либо построено синтаксическое дерево, либо выявлены составляющие. Обычно грамматика строится так, чтобы на выходе получалось синтаксическое дерево, позволяющее выполнять разнообразные трансформации лексического содержания с пересогласованием зависимых слов, а также легко выделять семантику, в частности - применять алгоритм взвешивания альтернативных вариантов построения дерева, о котором речь пойдет ниже [23].

Предложение может допускать несколько альтернативных вариантов связывания слов. В этом случае анализатор пытается применять некоторые эвристики и базу знаний, но может в конце концов вернуть несколько вариантов синтаксического дерева [24]. К такому поведению особенно склонен восходящий парсер, так как он сильно ограничен в средствах по пресечению экспоненциального роста числа вариантов. Наличие нескольких итоговых вариантов разбора может означать не только недостаточный набор правил в языковой модели, но и присущую данному предложению неоднозначность, устраняемую только учетом более широкого контекста.

Анализируемые предложения могут иметь разную сложность, включать неизвестные слова или отступления от нормативного синтаксиса. Чтобы эффективно справляться с разными задачами, синтаксический анализатор применяет несколько разных алгоритмов, включая структурный нисходящий анализ и восходящий анализ, а также применяет семантический анализ для уточнения результатов в случае неоднозначностей [25].

Структурный нисходящий анализ - самый точный. Восходящий анализ работает очень быстро и способен разобрать даже очень длинные предложения, игнорируя непонятные фрагменты. Достоинства и недостатки каждого вида анализа мы рассмотрим ниже.

Восходящий и нисходящий алгоритмы воплощают "классический" подход к созданию грамматики естественного яхыка. Этот подход заключается в создании набора правил, которые работают со словами и грамматическими категориями слов. Правила обычно пишутся на специальном предметно-ориентированном языке (DSL), чтобы упростить изменение грамматики без перекомпиляции кода парсера и дать в руки разработчику грамматики эффективный набор средств, обычно имеющих мало общего с обычными императивными языками программирования. Выразительные средства этого языка будут описаны далее с необходимой детальностью. Вместе с исходными текстами учебного парсера русских предложений, данное описание позволит Вам создавать собственные анализаторы.

Правила описывают порядок распознавания различных синтаксических конструкций естественного языка. Исходный текст правил читается компилятором словаря и преобразуется в сильно оптимизированное внутреннее представление, фактически в набор команд для виртуальной машины.

Вообще говоря, грамматический движок предоставляет два алгоритмически различных подхода к построению анализатора: итерационный алгоритм на основе правил переписывания и очень мощный структурный нисходящий парсер. Правила переписывания детально описываются в отдельном параграфе далее. Несмотря на некоторое различие в математической основе, существует способ применять в обоих типах анализа общий набор правил, таким образом реализуя сложный процесс синтаксического анализа с динамическим переключением направления.

В терминах теории автоматов, набор правил реализует распознающую грамматику для предложений на естественном языке. Результатом работы анализатора является не только подтверждение того, что входное предложение с определенной достоверностью принадлежит к распознаваемому набору синтаксических конструкций и идиоматических оборотов [26], но и набор альтернативных синтаксических деревьев с весами в случае неоднозначности анализа.

Далее приводятся характерные особенности структурного нисходящего парсера (далее СНП), работающего в режиме “анализ через синтез”. СНП даст 100% верный разбор предложения на составляющие, если используемая грамматика адекватно описывает такие предложения. Процесс разбора структурным парсером очень похож на математически строгое доказательство. СНП найдет все возможные варианты разбора предложения, если грамматика содержит неоднозначности. СНП способен делать ослабляющие допущения и оценивать выполненный анализ с учетом таких допущений [27]. СНП может учитывать контекст любой ширины, вплоть до всего предложения. Другими словами, учитываются не только непосредственные соседи, но также связанные синтаксически части предложения, находящиеся на любом удалении от слова.

СНП не имеет внутреннего состояния и выполняет разбор за 1 заход. Нет естественного способа остановить выполнения разбора в промежуточной стадии и получить частичные результаты. Эта особенность имеет также важное положительное следствие. Отсутствие состояния позволяет распараллеливать выполнение анализа [28].

СНП естественным образом реализует поиск в глубину. Это может приводить к тому, что из-за отсутствия хороших эвристик очень много времени будет уделено проверке заведомо проигрышным вариантам анализа, а выигрышный путь в пространстве состояний так и не будет найден до исчерпания лимита времени.

Следствием отсутствия состояния является трудность отладки [29]. Отладчик парсера предоставляет некоторые ограниченные возможности, включая задание точек останова в некоторых местах правил и просмотр локального состояния вывода. Однако наличие множества параллельных процессов вывода делает интерактивную отладку крайне трудной в реализации и использовании. В реальности самым удобным способом отладки является просмотр пошаговой трассировки для полученных результатов.

Если структура предложения не выводится правилами грамматики в точном виде, то СНП вообще ничего не выведет, если не принимать специальные меры. Под такими специальными мерами подразумевается неполный анализ [30].

СНП реализует в большей степени математическую концепцию, чем языковую. В сочетании с декларативным и функциональным стилем задания правил вывода это может поднять порог вхождения для незнакомого с математической лингвистикой разработчика, привыкшего к императивному стилю программирования. СНП работает с предложением в ленивом стиле. Начав с левой границы, он постепенно выводит синтаксическую структуру, запрашивая у лексера новые и новые слова. Более того, СНП очень хорошо элегантно работают с графом токенизации, который получается в ходе лексического анализа.

Достоинства и недостатки итерационного анализа

Так как данный алгоритм анализа в целом основан на использовании таких же правил, как рассмотренный ранее нисходящий анализ через синтез, то он в определенной степени наследует некоторые его особенности. В частности, можно упомянуть необходимость ручного описания грамматики с большим количеством абстрактных категорий и некоторая сложность с пониманием процесса сопоставления правил.

Далее перечисляются только собственные особенности данного алгоритма.

Недетерминированность процесса. Входная цепочка токенов может породить несколько альтернативных новых цепочек после применения правила переписывания. Выбор оптимального результата выполняется с помощью механизма задания баллов для правил переписывания, либо с использованием общих с алгоритмов анализ-через-синтез приемов - ассоциации слов, tree scorer.

Итерации, состояние. Автомат имеет явное состояние. Процесс применения правил переписывания имеет заранее известные точки, в которых его можно прервать и получить текущие результаты связывания. Благодаря этому можно реализовать управление анализом по таймауту - привязывания деталей будет продолжаться до тех пор, пока автомат не перейдет в конечное состояние (успех - построен синтаксический граф с одним корнем) либо пока не будет исчерпан заданный лимит времени. Более того, механизм оценки достоверности переписывания позволяет создавать планировщик итераций, который меняет приоритет того или иного варианта анализа динамически.

Простота отладки. Процесс переписывания можно прервать в момент применения очередного правила и увидеть весь текущий набор результатов.

Работа с длинными предложениями. Можно задать такие правила, что автомат будет работать в минимальном контексте - например, ограничить его правилами для пар слов [31]. К примеру, связывать частицу НЕ с наречиями, прилагательными, деепричастиями и глаголами и т.д. Поэтому автомат может выполнять постепенное построение синтаксического дерева для очень длинных предложений, не пытаясь сразу охватить слишком большие фрагменты текста.

Правила переписывания используют весь набор возможностей для задания опорных точек контекста. Это позволяет задавать правила, которые сопоставляются с контекстом заранее неизвестной длины. Кроме того, доступны такие средства, как

1) теоретико-множественные операторы для проверки вхождения токена в именованное множество,

2) проверка с помощью регулярных выражений,

3) использование пользовательских функций, написанных на встроенном языке,

4) проверка согласования фрагментов сопоставляемого контекста.

Основной недостаток итерационного анализа - меньшая эффективность в равных условиях с анализом-через-синтез. В силу некоторых алгоритмических особенностей второго варианта компилятор словаря может выполнить весьма агрессивную оптимизацию, которая ускоряет процесс синтеза с проверкой. Более того, анализ-через-синтез реализует очень эффективную оптимизацию процесса во время работы [32]. Со своей стороны, итерационный анализ очень ограничен в возможных оптимизациях. Эффективность анализа в нем приносится в жертву итеративности и возможности постепенно улучшать результат, динамически отслеживая прогресс. Итерационный анализ можно начинать не просто с исходной цепочки токенов, а попытаться "прикинуть" общую структуру предложения с помощью набора специальных правил. Например, можно найти в предложении местоимение я и согласующийся с ним глагол, предположить из связи как подлежащего и сказуемого и далее уже запускать итерации, опираясь на эти ключевые токены.

Распознающая или порождающая грамматика

Архитектурная особенность структурного парсера заключается в таком нюансе.

С одной стороны, задача структурного парсера - определить синтаксическую структуру предложения. Иначе говоря, он распознаёт синтаксически корректные (или почти корректные - об этом далее) предложения. Результатом распознавания является не только сам факт признания корректности предложения с точки зрения реализованной грамматики [33]. Парсер также выдает информацию о синтаксических взаимоотношениях слов и извлекает некоторые факты.

С другой стороны, алгоритм распознавания грамматики спроектирован так, что он порождает по набору заданных правил возможные фразы и сопоставляет их с имеющейся фразой, отвергая недопустимые выводы и уточняя подтверждающиеся. Порождение начинается (если отбросить незначительные в данном случае детали) с некоторого начального нетерминального символа, обозначающего целое предложение, и постепенно расщепляется на множество вариантов, каждый из которых проверяется отдельно [34]. Например, разный порядок следования S-V-O и O-V-S для русского языка приводит к тому, что некоторые предложения подходят под разные шаблоны: мать любит дочь.

В итоге, синтаксический анализ предложения выполняется через синтез синтаксической структуры. Если синтез неоднозначен, то на выходе парсера появляются альтернативные результаты [35]. К примеру, без подключения дополнительного семантического анализа невозможно выбрать порядок связывания для предложения: “руки с мылом мой”. Возможные два варианта в скобочной записи выглядят так: мой(руки(с(мылом))) и мой(руки,с(мылом)). Оба они корректны с грамматической точки зрения.

Правила вывода можно рассматривать как контекстно-зависимые правила грамматики непосредственных составляющих с некоторыми специальными особенностями [36]. Визуально исходный текст правил выглядит как задание способа включения структурных элементов друг в друга, например, группы прилагательного в группу существительного. Ряд выразительных средств предназначен для удобного задания "нетерминальных символов".

Если рассматривать нисходящий анализ через синтез, то описываемая такими правилами грамматика является неукорачивающей, то есть порождаемые правилами цепочки нетерминалов удлиняются до тех пор, пока все нетерминальные символы не заменяются терминальными, соответствующими токенам в определенном пути в графе токенизации [37]. Процесс порождения цепочки теряется сразу после окончания синтаксического анализа, поэтому с точки зрения пользователя анализатор только распознает фразы.

Для итерационного алгоритма процесс выглядит принципиально иным. Прежде всего, если отбросить этап начального определения примерной структуры через sparse-правила, алгоритм начинает работу с исходной цепочки токенов. Применяя правила, анализатор стремится укоротить эту цепочку, в идеальном случае сведя ее к единственному корневому токену. Остальные токены прикрепляются к этому корню и друг к другу, формирую синтаксическое дерево. Правила обязательно укорачивают исходную цепочку токенов и не могут добавлять новые токены.

Лексикализация нетерминалов в правилах нисходящего разбора

Нисходящий парсер позволяет обходится очень небольшим количеством нетерминалов в правилах. Каким образом это небольшое количество позволяет учесть богатую морфологию русского языка, в частности разнообразные правила согласования по падежам, числам, родам, учитывать одушевленность, перечислимость, падежную валентность и все прочие элементы морфологической модели русского языка?

Нисходящий парсер содержит очень гибкий инструмент для лексикализации нетерминалов и расширения эффективного пространства признаков [38]. Говоря упрощенно, каждый нетерминал после своего сопоставления кроме собственно своего имени динамически расширяется целым набором дополнительных признаков, включая главное слово в фразе, второе существительное в предложной фразе, инфинитив в аналитических глагольных временах, а также морфологические признаки - род, число, падеж и так далее.

Весь этот набор доступен в вышестоящих правилах наряду с именем самого нетерминала. На уровне синтаксиса правил лексикализация выполняется использованием двух вещей.

Во-первых, каждая реализация нетерминала может экспортировать определенный, заранее объявленный набор лексем и морфологических признаков. Именно этот набор расширяет нетерминальный символ.

Во-вторых, в месте “вызова” нетерминального символа можно указать секцию уточнения, в которой задать проверки на вхождение или невхождение главного слова в именованные множества, заданные значения морфологических атрибутов или согласование морфологических атрибутов.

Оба этих механизма подробно описаны далее, когда объясняется синтаксис правил парсера.

Общий принцип работы нисходящего структурного парсера

Слово нисходящий в определении структурного парсера подчеркивает фундаментальную особенность данного алгоритма. Он выполняет разбор предложения с помощью выдвижения альтернативных гипотез и их доказательства или опровержения. Разбор начинается с выдвижения гипотез самого общего вида по поводу структуры предложения, например - вопрос, утверждение, вежливая побудительная конструкция. Для каждой из выдвинутых гипотез определяется набор гипотез, чье доказательство необходимо для их подтверждения или опровержения. Процесс доказательства гипотез продолжается до тех пор, пока не будет найдена необходимая комбинация токенов. Алгоритм нисходящего вывода позволяет применять альтернативные гипотезы (реализация недетерминированного автомата). Это крайне важно для текстов на естественном языке, в которых грамматическая трактовка токенов зависит от контекста заранее неизвестной ширины. В ходе доказательства гипотез может выполняться рекурсивное применение. Например, при разборе группы существительного с правым причастным оборотом может возникнуть необходимость разобрать входящие в состав оборота другие, более мелкие группы существительного, в которых также может быть причастный оборот, и так далее. Такие конструкции характерны для письменной речи, насыщенной сложными оборотами.

Нисходящий анализатор выполняет одновременно несколько важнейших операций над исходным текстом. Он распознает грамматические признаки слов (part of speech tagging в широком понимании), разрешает неоднозначности, строит синтаксическое дерево, возможно с альтернативными вариантами, взвешиваемыми на основе разных весовых алгоритмов.

Для эффективного применения большого набора правил компилятор словаря выполняет агрессивную глубокую оптимизацию исходного набора правил. Побочным результатом такой оптимизации является техническая трудность пошаговой отладки правил, так как понятия шага в привычном императивном понимании здесь нет. Как будет показано далее, наиболее близкой парадигмой является функциональное программирование. Посмотрим, как парсер разбирает предложение: *Мы спали.*

Анализ всегда начинается с начального нетерминального символа Предложение. Допустим, у нас есть два правила, которые выводят из этого начального символа две новых цепочки:

Предложение := Подлежащее Сказуемое

Предложение := Сказуемое Подлежащее

Оба варианта допускаются в русском синтаксисе.

Применение двух первых правил:

Было:

*Предложение*

Стало:

*Вариант 1. Подлежащее Сказуемое*

*Вариант 2. Сказуемое Подлежащее*

Символы Сказуемое и Подлежащее - это тоже нетерминалы, то есть они не соответствуют никаким словам, а обозначают некие абстрактные синтаксические сущности, которые еще предстоит раскрыть.

Добавляем два новых правила:

*Подлежащее := мы; Сказуемое := спали*

В этих правилах появляются нормальные слова, иначе говоря терминальные символы нашей грамматики. После их применения к двум текущим вариантам имеем:

Вариант 1. *мы Сказуемое*

Вариант 2. *спали Подлежащее*

Мы пока заменили только первый элемент в каждом варианте. Так как результатом этих замен являются терминальные слова, которые дальше уже не могут заменяться другими символами, то парсер приступает к первой проверке гипотез. Вариант №1 успешно проходит проверку, так как в предложении в текущей позиции стоит как раз нужное слово. Вариант №2 проваливает проверку, и поэтому парсер далее не пытается с ним работать. Это очень важный момент, так как мы еще не делали замену вторых частей в обоих вариантах и не пытались взять из предложения следующие слов. Отбросив второй вариант как явно недопустимый, мы экономим вычислительные ресурсы.

Парсер продолжает работать с первым вариантом. Он применяет имеющееся правило к нетерминалу Сказуемое и получает:

Вариант 1: *мы спали.*

После замены опять выполняется проверка, которая успешно подтверждает гипотезу для первого варианта. Гипотеза эта, как нетрудно догадаться звучит в нашем примере так: предложение имеет структуру Подлежащее+Сказуемое.

Так как нетерминалов более не осталось и текущие терминальные символы сопоставлены со словами в исходном предложении, то парсер прекращает анализ, сообщая о найденной структуре.

Более сложные примеры работы парсера в целом идут именно таким путем - поиск замен для нетерминальных правил, сопоставление полученных терминалов со словами в предложении, отсечение некорректных гипотез.

Этот этот пример также раскрывает релевантность терминов "нисходящий анализ", "анализ через синтез" и "грамматика непосредственных составляющих", которые постоянно будут встречаться при описании парсера.

Насколько глубоко могут идти замены одних нетерминалов цепочками других? Парсер не накладывает никаких явных ограничений. Более того, парсер допускает правую рекурсию в правилах замены. Например, причастных оборот может быть атрибутом существительного:

*Ветер, качающий деревья*

Элементами причастного оборота могут быть прямые дополнения - существительные. И у этих существительных также могут быть причастные обороты в роли их атрибутов:

*Ветер, качающий деревья, посаженные вдоль аллеи.*

Письменный язык, особенно художественный, вплне толерантен к таким усложненным синтаксическим конструкциям. Кроме причастных и деепричастных оборотов, существует масса других речевых конструкций, в которых однотипные синтаксические компоненты вложены друг в друга как матрешки. Парсер предоставляет естественный способ описания правил вывода для них - через упоминание в правой части правила заменяемого терминала или другого нетерминала, для которого в свою очередь есть правило замены на исходный нетерминал.

Итак, дерево разбора предложения будет выглядеть следующим образом:

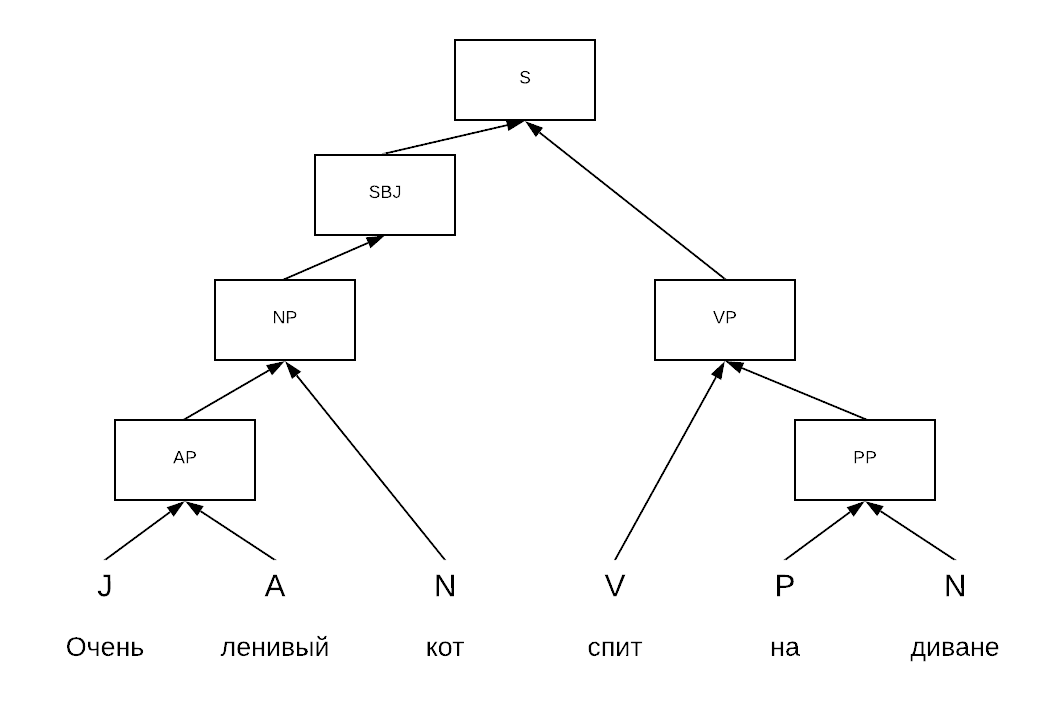


Рис. 2.2– визуализация дерева разбора предложения

## Машинное обучение и нейросетевые алгоритмы

За последние 10 лет лучшие системы искусственного интеллекта - такие как распознаватели речи на смартфонах или новейший автоматический переводчик Google - стали результатом методики, получившей название "глубокое обучение".

Глубокое изучение - это фактически новое название подхода к искусственному интеллекту, называемого нейронными сетями, который уже более 70 лет входит и выходит из моды. Впервые нейронные сети были предложены в 1944 году Уорреном МакКаллоу и Уолтером Питтсом, двумя исследователями из Чикагского университета, которые перешли в Массачусетский технологический институт в 1952 году в качестве членов-учредителей того, что иногда называют первым когнитивным научным отделом.

Нейронные сети были основной областью исследований в неврологии и компьютерной науки до 1969 года, когда, в соответствии с компьютерной наукой, они были убиты MIT математиков Марвина Минского и Сеймура Паперта, который год спустя станет содиректорами новой MIT Лаборатории искусственного интеллекта [39].

Техника затем пережила возрождение в 1980-х годах, в первом десятилетии нового века снова затмение, а во втором вернулась как гангстеры, подпитываемая во многом за счет увеличения вычислительной мощности графических чипов.

Очевидно, что существует пять или шесть основных штаммов вирусов гриппа, и, по-видимому, каждый из них возвращается с периодом около 25 лет. Люди заражаются, и у них развивается иммунная реакция, поэтому они не заразятся в течение следующих 25 лет. И тогда появляется новое поколение, готовое заразиться тем же штаммом вируса. В науке люди влюбляются в идею, возбуждаются от нее, забивают ее до смерти, а затем вакцинируются - им это надоедает. Поэтому идеи должны иметь такую же периодичность!"

Нейронные сети - это средство машинного обучения, в котором компьютер учится выполнять определенную задачу, анализируя учебные примеры. Обычно примеры заранее маркируются вручную. Например, в систему распознавания объектов могут быть поданы тысячи помеченных изображений машин, домов, кофейных чашек и т.д., и на изображениях будут найдены визуальные шаблоны, которые последовательно соотносятся с конкретными этикетками.

Нейронная сеть, свободно смоделированная на человеческом мозге, состоит из тысяч или даже миллионов простых обрабатывающих узлов, которые плотно взаимосвязаны. Большинство современных нейросетей организованы в слои узлов, и они "движутся вперёд", то есть данные движутся через них только в одном направлении. Отдельный узел может быть соединен с несколькими узлами в слое под ним, откуда он получает данные, и несколькими узлами в слое над ним, куда он отправляет данные [40].

Каждому входящему соединению узел присваивается номер, известный как "вес". Когда сеть активна, узел получает над каждым из своих соединений отдельный элемент данных - разное число - и умножает его на связанный с ним вес. Затем узел складывает полученные произведения вместе, получив единое число. Если это число ниже порогового значения, узел не передает данные следующему слою. Если число превышает пороговое значение, узел "выстреливает", что в современных нейронных сетях обычно означает отправку числа - суммы взвешенных входов - по всем своим исходящим соединениям.

При обучении нейросети все ее веса и пороги изначально устанавливаются на случайные значения. Обучающие данные подаются на нижний слой - входной слой - и проходят через последующие слои, умножаются и суммируются сложными способами, пока, наконец, не поступают, радикально преобразованные, на выходной слой. Во время обучения веса и пороговые значения постоянно корректируются до тех пор, пока тренировочные данные с одними и теми же метками не дадут одинаковые результаты.

Нейронные сети, описанные МакКаллоу и Питтсом в 1944 году, имели пороговые значения и веса, но они не были расслоены по слоям, и исследователи не уточняли механизм обучения. МакКаллоу и Питтс показали, что нейронная сеть в принципе может вычислить любую функцию, которую может выполнять цифровой компьютер. В результате было больше неврологии, чем компьютерной науки: Смысл состоял в том, чтобы предположить, что человеческий мозг можно считать вычислительным устройством.

Нейронные сети продолжают быть ценным инструментом для нейронаучных исследований. Например, конкретные схемы сети или правила для регулировки веса и пороговых значений воспроизвели наблюдаемые особенности человеческой нейроанатомии и познания, что свидетельствует о том, что они захватывают что-то о том, как мозг обрабатывает информацию.

Перцептроны были активной областью исследований как в психологии, так и в зарождающейся дисциплине информатики до 1959 года, когда Минский и Пайперт опубликовали книгу под названием "Перцептроны", которая показала, что выполнение некоторых довольно распространенных вычислений на Перцептронах будет непрактично трудоемким.

Конечно, все эти ограничения как бы исчезают, если взять машину, которая немного сложнее - например, два слоя [41]. Но в то время книга оказала охлаждающее воздействие на исследования нейросетей.

Не много лет назад люди все еще пользовались аналоговыми компьютерами. В то время было совсем не ясно, что программирование - это путь. Я думаю, что они немного перестарались, но, как обычно, это не черно-белое. Если вы думаете об этом как о соревновании между аналоговыми и цифровыми вычислениями, они боролись за то, что в то время было правильным.

Однако к 1980-м годам исследователи разработали алгоритмы модификации весов и пороговых значений нейросетей, которые были достаточно эффективны для сетей с более чем одним слоем, устранив многие из ограничений, выявленных Минским и Папертом. Поле получило ренессанс.

Но в интеллектуальном плане в нейронных сетях есть что-то неудовлетворительное. Достаточное обучение может привести к пересмотру настроек сети до такой степени, что она сможет с пользой классифицировать данные, но что означают эти настройки? На какие особенности изображения смотрит распознаватель объектов, и как он объединяет их в отличительные визуальные признаки автомобилей, домов и кофейных чашек? Глядя на вес индивидуальных соединений, вы не найдете ответа на этот вопрос [42].

В последние годы ученые в области вычислительной техники начали придумывать гениальные методы для вывода аналитических стратегий, принятых в нейронных сетях. Но в 1980-х годах стратегии сетей были неразборчивы. Поэтому на рубеже веков нейронные сети были вытеснены вспомогательными векторными машинами - альтернативным подходом к машинному обучению, основанным на очень чистой и элегантной математике.

Недавнее возрождение нейронных сетей - революция в области глубокого обучения - произошло благодаря компьютерной игровой индустрии. Сложные образы и быстрый темп современных видеоигр требуют аппаратного обеспечения, способного их поддерживать, в результате чего появился графический процессор (GPU), который упаковывает тысячи относительно простых процессорных ядер на одном чипе. Исследователям не потребовалось много времени, чтобы понять, что архитектура GPU удивительно похожа на архитектуру нейронной сети.

Современные GPU позволили одноуровневым сетям 1960-х годов, а двух- и трехуровневым сетям 1980-х годов расцвести в 10-, 15-, даже 50-уровневые сети сегодняшнего дня. Вот что означает "глубина" в "глубоком обучении" - глубина слоев сети. И в настоящее время "глубокое" обучение отвечает за наиболее эффективные системы практически во всех областях искусственного интеллекта.

Непрозрачность сетей до сих пор тревожит теоретиков, но на этом фронте тоже есть прогресс. Помимо руководства Центром мозгов, умов и механизмов (ЦМРМ), Poggio возглавляет исследовательскую программу центра в области теоретических основ интеллекта. Недавно Poggio и его коллеги по CBMM выпустили трехкомпонентное теоретическое исследование нейронных сетей.

Первая часть, которая была опубликована в прошлом месяце в Международном журнале по автоматизации и вычислениям, посвящена диапазону вычислений, которые могут выполняться в сетях глубокого обучения и когда глубокие сети имеют преимущества перед более мелкими. Части вторая и третья, которые были выпущены в виде технических отчетов по МДМ, посвящены проблемам глобальной оптимизации, или гарантирования того, что сеть нашла настройки, которые наилучшим образом соответствуют ее учебным данным, и переоснащению, или случаям, когда сеть настолько приспособляется к специфике своих учебных данных, что не в состоянии обобщить их для других случаев из тех же категорий [43].

Остается еще много теоретических вопросов, на которые необходимо ответить, но работа исследователей в области МДМ могла бы помочь гарантировать, что нейронные сети, наконец, разорвут цикл поколений, который приносил им пользу в течение семи десятилетий.

## Основные методы машинного обучения для NLP

За два последних века человечество успешно справилось с автоматизацией многих задач используя механические и электрические приборы. Вторая половина ХХ века обратила внимание человека к автоматизации обработки естественного языка. Теперь человеку нужна помощь не только с механической работой, но и с интеллектуальными задачами [44]. Задачей для машин в современных условиях ставится способность читать неподготовленный текст, проверять его на ошибки, выполнять задачи, поставленные в тексте, и даже понимать текст настолько хорошо, чтобы дать в ответ, основываясь на значении текста. Актуальность обуславливает наличие проблем обработки естественного языка, которую нельзя назвать простой. Трудности возникают по ряду объективных причин, среди которых существование сотен естественных языков, имеющих свои синтаксические правила. Кроме того, в рамках одного языка существуют слова, которые могут иметь смешанное содержание в зависимости от контекста употребления. Даже на уровне отдельных символов встречаются определенные трудности.

В процессе обработки естественного языка всегда следует учитывать кодирование, используемое в конкретном документе. Текст может храниться в разных кодировках: ASCII, Unicode, UTF-8, UTF-16, Latin-1 и др. Особые виды обработки могут понадобиться для знаков пунктуации и для чисел. Также иногда приходится отдельно обрабатывать использование знаков, которые отражают эмоции (комбинации символов или специальные символы), гиперссылок, повторяющихся знаков препинания (... или ---), расширений файлов и имен пользователей, содержащих точки [3].

Под распределением текста на фрагменты или элементы обычно имеется в виду представление текста в виде последовательности слов. В этом случае слова обозначаются термином «лексический элемент», «лексема», или просто «токен» (token), а процесс разделения текста - «токенизация» (tokenization). Этот процесс не представляет трудности в языках, использующих символы-пробелы для разделения слов, но в языках, подобных китайскому, это сделать гораздо труднее, поскольку иероглифы могут обозначать как склады, так и целые слова. Также и в английском языке с процессом токенизации могут возникнуть определенные трудности, ведь существует большое количество альтернативных вариантов, когда одно слово может писаться слитно, раздельно или через дефис.

Слова объединяются в словосочетания и предложения, следовательно, определение границ предложений тоже может быть связано с определенными трудностями, хотя на первый взгляд кажется, что достаточно только найти точку, обозначающую конец предложения. Но точки могут встречаться и внутри предложений, например, после сокращенных слов [3]. При грамматическом разборе все еще возникают серьезные проблемы с точностью. Во-первых, многое здесь зависит от качества морфологической (части речи) разметки (part-of-speech tagging), которая должна быть очень высокой (97-98%), однако в длинных предложениях очень часто можно встретить неправильно распознанную определенную часть речи, что приводит к дальнейшим ошибкам разбора. Во-вторых, современный автоматический синтаксический разбор дает точность примерно 90-93%, а это, в свою очередь, означает, что в длинном предложении практически всегда будут ошибки разбора. Например, при точности разбора 90%, вероятность разбора предложения длиной 10 слов без единой ошибки составит всего 35%.

Часто правильный синтаксический разбор включает также понимание семантики предложения, но, например, в английском языке это нередко вызывает трудности. Так, в предложении «Не saw a man with a hammer» может быть два варианта синтаксического разбора в зависимости от того, считаем ли мы, что человека увидели с помощью молотка или увидели человека с молотком. Конечно, если нужно получить максимально точный синтаксический разбор, то имеет смысл оставлять несколько наиболее вероятных вариантов, а затем определять правильный по совокупности различных факторов, в том числе семантических [1].

Иногда приходится определять связи между словами. Например, установление кореферентности (coreference resolution) определяет связи между конкретными словами, обозначающими один и тот же объект, то есть имеющими один и тот же референт в одном или в нескольких предложениях. Например, в предложениях «The city is large but beautiful. It fills the entire valley» слово «it» кореферирует, то есть референционно тождественному слову «city».

Явления кореферентности обусловлены фундаментальными закономерностями организации текста. Поскольку текст имеет линейное строение, а ситуация, которую он описывает, как правило, нелинейная, в тексте почти неизбежно должны содержаться повторные упоминания элементов ситуации, которая описывается. При каждом новом упоминании того же объекта проводится новая номинация этого объекта, основанная на том, что уже было сказано об этом объекте, и на тех знаниях, которые в тексте не вербализированы (экстралингвистические знания говорящего о контексте предметной области).

Хотя проблема кореферентности в лингвистике достаточно подробно исследована, воплощение этих теоретических знаний на практике на сегодняшний день является достаточно сложным [45]. Если слово может иметь несколько смысловых значений, для определения его смысла в данном конкретном случае может потребоваться выполнение операции решения лексической многозначности (word sense disambiguation, WSD). Это связано с определенными трудностями. Например, в предложении «John went back home» слово «home» может означать «housing that someone is living in» или «the country or state or city where someone lives».

Одной из самых открытых проблем при обработке текстов естественного языка является неоднозначность (многозначность) его единиц, что сказывается на всех уровнях и выражается в явлениях полисемии, омонимии и синонимии. Говоря о неоднозначности, можно отметить такие ее виды, как: лексическая, синтаксическая или структурная (например, проблема присоединения - attachment ambiguity), семантическая (когда одно и то же предложение можно по-разному понимать в разных контекстах, хотя лексическая или структурная многозначность отсутствует), прагматическая неоднозначность (когда одно предложение можно по-разному понимать в контексте, в котором оно существует) [2].

Современные системы решения лексической многозначности имеют точность в диапазоне 60-70% и, чаще всего, представлены как самостоятельные методы. Решение проблемы снятия неоднозначности требует интеграции нескольких источников информации и методов. Несмотря на все перечисленные трудности, технология обработки естественного языка в большинстве случаев способна достаточно успешно справиться со своими задачами, поэтому очень полезна во многих отраслях.

Примерно в половине случаев имеет место любая форма омонимии, и набор морфологических признаков оказывается недостаточным для ее решения. Уменьшить неоднозначность можно с помощью синтаксического и семантического анализа с использованием статистических методов, которые и позволяют отбросить крайне маловероятны варианты. Естественный язык хоть и является по своей природе символическим, обработать его с помощью символических, основанных на логике, правил и объективных моделей достаточно сложно. Естественный язык является крайне неоднозначной и переменчивой, поэтому для ее обработки необходимо применять статистические алгоритмы, поэтому доминантными подходами современной обработки языка являются подходы, основанные на статистическом машинном обучении (statistical machine learning) [6].

Машинное обучение (machine learning) исследует изучение и построение алгоритмов, которые могут продуцироваться на основе данных, и выполнять предсказательный анализ на них [7]. Такие алгоритмы действуют путем построения модели из образцового тренировочного набора входных наблюдений, чтобы создавать управляемые данными прогнозы или принимать решения, выраженные как выходы (результаты), вместо того, чтобы строго придерживаться статических программных инструкций.

Большинство определений машинного обучения отождествляют с наукой о задействовании компьютеров к обучению выполнению действий - подобно как это делают люди для улучшения их обучения в течение долгого времени в самостоятельном режиме, наполнения данными и информацией в виде наблюдений и реального взаимодействия с окружающими. В современных условиях машинное обучение, в основном, развивается в направлении глубокого обучения (deep learning) - это такой тип обучения, в котором модель учится решать задачи классификации непосредственно с изображениями, текстом или звуком. Глубокое обучение, как правило, осуществляется с помощью архитектуры глубокой искусственной нейронной сети [46].

Основной характеристикой, определяющей преимущество глубокого обучения, является точность. Передовой инструментарий и новейшие методы резко улучшили алгоритмы глубокого обучения. Эти технологии достигли точки, где они могут превзойти людей в классификации изображений, выигрывать против лучших игроков соответствующей отрасли в мире, включать контролируемого голосом помощника от Microsoft Cortana, Amazon Echo или Google Home и пр. Ниже перечислены три технологические возможности, которые позволяют достичь необходимой степени точности глубокого обучения [47]:

1. Легкий доступ к массивам наборов помеченных данных, таких как ImageNet, PASCAL VoC, доступных и удобных для обучения на многих различных типах объектов.

2. Увеличение вычислительных мощностей - графические процессоры (GPU) высокой производительности ускоряют подготовку огромного количества данных, необходимых для глубокого обучения, что позволяет достичь сокращения учебного времени от недель до часов.

3. Предварительно тренированные (Pretrained) модели, построенные экспертами, такие как AlexNet, могут перетренироваться для выполнения новых задач по распознаванию естественного языка (и других даных) и используют технологию, названную «передача обучения». Хотя AlexNet учился на 1,3 млн. образах с высоким разрешением для распознания 1000 различных объектов, но точную передачу обучения достиг с гораздо меньшим набором данных.

Алгоритмы машинного обучения является убедительно привлекательными для внедрения в различные сферы социально-экономического хозяйствования. Насчитывается большое количество проектов на основе технологий машинного обучения. Как отмечает Д. Фаджела, в условиях ведения бизнеса перед стартапами пока возникает вопрос - начинать бизнес-проект с начальным привлечением технологии машинного обучения или начинать его использовать на более поздней стадии развития бизнеса [9]. Следует при этом иметь в виду, что рентабельность инвестиций в машинное обучение требует кропотливой работы по настройке и корректировке.

Доктор университета Оклахомы Данко Николс в своей статье [8] отмечает, что наиболее распространенными ошибками, которые делают предприятия при использовании машинного обучения является мнение о том, что решения систем и принципов машинного обучения для конкретной сферы - одноразовый процесс: менеджеры присылают данные специалистам по данным (data scientists), которые готовят готовые модели. На самом деле поиск хорошего решения - это итерационный процесс, который включает в себя исследования, пробы и ошибки, экспериментирование, консультирование бизнес-специалистов и т.д. Утверждение о том, что машинное обучение никогда не сможет стать товаром, и его успех сильно зависит от знаний, навыков и самоотверженности тех людей, которые это делают [8] - выглядит достаточно убедительным.

Вывод. Под обработкой естественных языков понимают создание систем с признаками искусственного интеллекта, которые определенным образом обрабатывают речевую информацию с целью выполнения определенных задач. К таким задачам относятся: чат-боты или формирование ответов на вопросы пользователя; определение характера эмоциональной окраски высказываний; машинный перевод с одного языка на другой; распознавание языков; проверка правописания; определение частей речи в предложении и их аннотирование; рерайт текстовой информации для создания веб-контента. Машинное обучение в рамках обработки естественных языков представляет собой актуальную сферу научного знания, которая интенсивно развивается и имеет очень большие перспективы. В наиболее узком смысле под машинным обучением понимают класс методов искусственного интеллекта, характерной чертой которых является не прямое решение поставленной задачи, а применение для этого специально обученной математической модели. Такая модель учится за счет решения большого количества задач в нужной области [48].

Смысл применения машинного обучения для обработки естественных языков заключается в том, что глубинные нейронные сети выполняют работу, на осуществление которой в течение приемлемого промежутка времени нужно было бы применять десятки или даже сотни команд профессиональных лингвистов. Традиционные нейронные сети не имеют возможности принимать текущие решения на основе своих предыдущих суждений. Большое количество задач, решаемых при машинной обработке естественных языков, требует поэтапного анализа данных с учетом предыдущих результатов. Нейронная сеть должна «читать» предложение слово за словом, «осмысливая» его значение исходя из контекста.

Одним из самых перспективных современных технологий машинного обучения является применение глубинных нейронных сетей, в основе которых лежит применение глубокого обучения [49]. Глубокое обучение - это набор алгоритмов машинного обучения, которые позволяют создавать модели с высоким уровнем абстракции в исходных данных, используя архитектуры нейронных сетей, содержащих нелинейные преобразования сигнала. Характерной особенностью алгоритмов глубоко обучения является прохождение входящей информации через гораздо большее количество слоев, чем при традиционном (поверхностном) обучении. Для нейронных сетей с рекуррентной архитектурой путь, которым проходит информационный сигнал от входа к выходу, является теоретически неограниченным (практически он может ограничиваться возможностями примененного программного обеспечения). Благодаря использованию глубинных нейронных сетей решаются задачи компьютерного видения, обработки естественных языков, распознавание музыки, прогнозирования развития событий, интеллектуальной фильтрации данных, построения чат-ботов и др. Большое количество привычных и удобных сервисов компании Google, которые популярные сейчас во всем мире и во всех сферах применения, было бы совершенно невозможным без применения глубинных нейронных сетей. Следовательно, обработка естественных языков посредством применения машинного (глубокого) обучения является перспективным направлением в теории и практики наук современного развивающегося информационного общества [50].

NLP – natural language processing или обработка естественных языков – в последнее время зачастую ассоциирует с машинным обучением и нейронными сетями. Хотя в предыдущих разделах было описана общая структура методов обработки естественных языков, такие как синтаксический и семантический анализ, которые могут существовать и отдельно от машинного обучения, в этой работе основное исследования будет опираться на последние достижения современной науки в области ml и nlp. Несколько лет назад, благодаря компании Google был совершен прорыв в этой области. Они создали архитектуру BERT –

Bidirectional Encoder Representations from Transformers – двунаправленный кодировщик представлений. Эта архитектура позволяет решать множество задач разных классов, кроме того, на ее основе созданы надстройки. Одна из них deeppavlov – модель, созданная в России, позволяющее совершать множество операций над текстом, в том числе на русском языке. [51]

## Алгоритм BERT для задач анализа текста

Использование модели BERT как основы для решения задачи синтаксического анализа предложения на русском языке имеет множество преимуществ перед написанием и проектировкой “собственной” архитектуры:

а) Модель BERT показывают высокую точность и достоверность при решении множества классов задач по анализу текста

б) Архитектура отличается высокой скоростью, а предобученные модели позволяют выполнять задачи даже на вычислительных машинах со слабыми характеристиками, в том числе на домашних ПК и ноутбуках.

Исходя из этих преимуществ, будет разумно выбрать архитектуру BERT в качестве основания модели синтаксического анализатор. Архитектура BERT для решения этого класса задач представлена на рисунке 2.3.

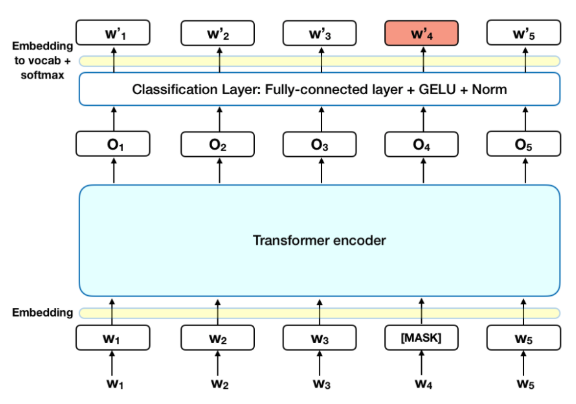


Рис. 2.3 – Архитектура BERT

Можно отметить, что наиболее близкой задачей для синтаксического анализа – является задача классификации, поэтому такая архитектура и была выбрана.

BERT использует трансформатор, механизм внимания, который изучает контекстные отношения между словами (или подсловами) в тексте [52]. В своем ванильном виде трансформатор включает в себя два отдельных механизма - кодировщик, который считывает вводимый текст, и декодер, который выдает предсказание для задачи. Поскольку целью BERT является создание языковой модели, необходим только механизм кодирования. Подробная работа Трансформатора описана в статье Google [53].

В отличие от моделей направленности, которые считывают последовательно вводимый текст (слева направо или справа налево), кодировщик трансформатора считывает всю последовательность слов сразу. Поэтому он считается двунаправленным, хотя было бы точнее сказать, что он ненаправленный. Эта характеристика позволяет моделировать контекст слова исходя из всего его окружения (слева и справа от слова) [54].

Приведенная ниже диаграмма представляет собой высокоуровневое описание кодировщика трансформатора. Входной сигнал представляет собой последовательность токенов, которые сначала встраиваются в векторы, а затем обрабатываются в нейросети. Выходом является последовательность векторов размера H, в которой каждый вектор соответствует входному токену с одинаковым индексом [55].

При обучении языковых моделей возникает задача определения цели прогнозирования. Многие модели предсказывают следующее слово в последовательности (например, "Ребенок вернулся домой из \_\_\_") - направленный подход, который по своей сути ограничивает контекстное обучение. Для преодоления этой проблемы BERT использует две стратегии обучения:

Первая – это “маскированная” языковая модель (Masked LM). Перед подачей последовательностей слов в BERT 15% слов в каждой последовательности заменяются на символ [MASK]. Затем модель пытается предсказать исходное значение маскированных слов, исходя из контекста, предоставленного другими, не маскированными словами в последовательности. С технической точки зрения, прогнозирование выходных слов требует:

* Добавление классификационного слоя поверх выхода кодирующего устройства.
* Умножение выходных векторов на встраиваемую матрицу, преобразование их в словарное измерение.
* Вычисление вероятности каждого слова в словаре с помощью функции софтмакса.

Пример функционирования стратегии приведен на рисунке 2.4.

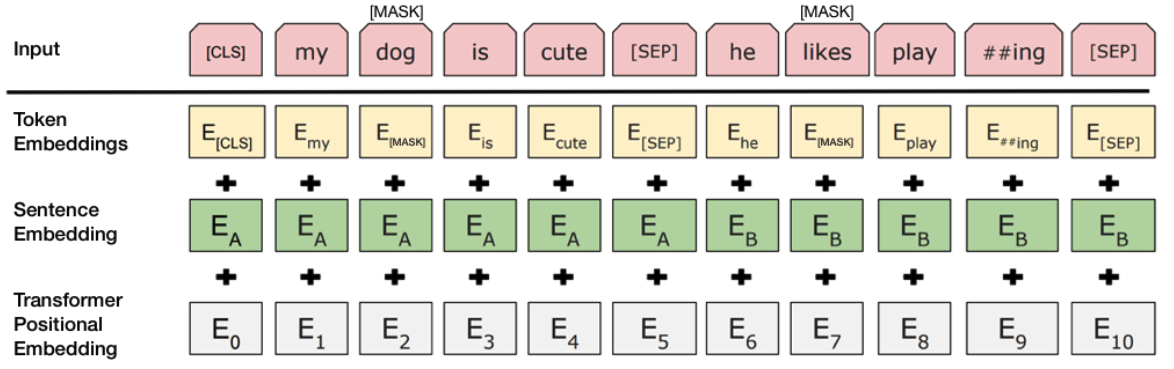


Рис. 2.4 – Визуализация алгоритма работы Masked LM

Вторая стратегия – предсказание следующего предложения (Next Sentence Predction). В процессе обучения BERT модель получает пары предложений в качестве входных и учится предсказывать, является ли второе предложение в паре последующим предложением в оригинальном документе. Во время обучения 50% входных данных - это пара, в которой второе предложение в паре является последующим предложением в оригинальном документе, в то время как в остальных 50% случайное предложение из корпуса выбирается как второе предложение. Предполагается, что случайное предложение будет отключено от первого предложения [56]. Для того, чтобы модель могла различать два предложения в процессе обучения, ввод данных обрабатывается следующим образом, прежде чем они будут введены в модель:

* В начале первого предложения вставляется маркер [CLS], а в конце каждого предложения - маркер [SEP].
* К каждому предложению добавляется предложение с указанием Приказа А или Приказа Б. Встраивание в предложение аналогично встраиванию токена со словарным запасом 2.
* К каждой маске прибавляется вложение позиции с указанием ее позиции в последовательности.

Чтобы предсказать, действительно ли второе предложение связано с первым, выполняются следующие шаги:

* Вся входная последовательность проходит через модель трансформатора.
* Вывод токена [CLS] преобразуется в вектор формы 2×1 с помощью простого классификационного слоя (выученные матрицы весов и смещений).
* Вычисление вероятности IsNextSequence (является ли следующее предложение подходящим) с помощью функции софтмакса.
* При обучении модели BERT, Masked LM и Next Sentence Prediction обучаются вместе с целью минимизации комбинированной функции потерь двух стратегий.

## Адаптация модели BERT под задачу

Кодировщик BERT выдает последовательность скрытых состояний. Допустим, для задач классификации нам нужен только один вектор для предсказаний, поэтому последовательность должна быть вычтена в один вектор. Для этого есть два способа. Первый - это максимальное или среднее объединение, а второй - использование силы внимания. Однако, самый простой подход - это принять соответствующее скрытое состояние первой лексемы [57].

Итак, для настройки модели BERT выберем стратегию. Так как наш класс задач является относительно узким – необходимо выполнять лишь синтаксический парсинг предложения – настроим модель на узкий класс задач. Модель перестанет быть гибкой, зато должны улучшится показатели в нашей области.

Различные слои нейронной сети могут захватить различные уровни синтаксической и семантической информации. Чтобы адаптировать BERT к поставленной задаче, необходимо учитывать несколько факторов:

1) Первый фактор — это предварительная обработка длинного текста, так как максимальная длина последовательности BERT составляет 512.

2) Вторым Фактором является выбор слоев. Основная модель BERT состоит из встраиваемого слоя, 12-слойного кодировщика и слоя пулинга. Нам нужно выбрать наиболее эффективный слой для задачи классификации текста.

3) Третий фактор - проблема переобучения. Необходимо выбрать лучшую функцию при наименьшем времени обучения. Интуитивно можно предположить, что нижний слой модели BERT может содержать более общую информацию. Мы можем провести более тонкую настройку варьируя скорость обучения.

## Формулировка задачи

Разделим параметры на множество , где L – количество слоев

(8)

где

Устанавливая базовую скорость обучения будем рассчитывать скорость обучения для каждого последующего по формуле

(9)

При , на всех слоях скорость обучения будет одинакова, при нижние слои будут иметь меньшую скорость, а при нижние слои будут обучатся быстрее верхних. Можно также отметить, что при значении параметра обучение будет соответствовать модели стохастического градиентного спуска (SGD).

Кроме параметров обучения, необходимо провести, так называемое, предобучение. Модель BERT предобучена на большом массиве данных, состоящем из множества различных тем. Чтобы модель подходила под нашу область, необходимо провести предварительное дообучение, которое может состоять из следующих задач [58]:

1. Дополнительное обучение внутри конкретной задачи на небольшом массиве данных.
2. Внутриобластное дообучение. В данном случае, модель обучается для более широкого круга смежных задач, однако все еще остается в одной области.
3. Межобластное дообучение. Используя этот подход, дообучение проводится по нескольким близким областям, включая область, к которой относится задача.

Обученную модель можно будет использовать для решения задачи синтаксического анализа.

## Создание модели синтаксического анализа

При использовании обычной модели BERT количество специфичных свойств модели составит примерно 3%, для достижения желаемого уровня специфичности в 100% (узкий класс задач) необходимо провести настройку модели [59].

Будем использовать метод бутылочного горлышка. Суть метода состоит в применении большого числа зафиксированных параметров основной модели и небольшого числа варьируемых параметров добавочной модели. Также в этой работе использован адаптерный подход. Параметры, соответствующие задачи, подключаются “внешне”, никак не связаны с основной моделью, а также могут быть отключены.

Как уже было упомянуто раньше, архитектура BERT содержит всего 12 слоев кодировщиков и около 100 миллионов параметров. Адаптер будет содержать один слой кодировщика и около 10000 параметров.

Алгоритм работы модели будет следующим.

1. Вход модели. На вход подаются предложения, а также слой-маска – синтаксический разбор предложения, учитывая токены частей предложения
2. Встраивание разделителей. Токены – в этом случае – части предложения, приводятся в соответствие со словами и уровнями, начиная от основания (головы) предложения.
3. Уровень нормализации. Все данные приводятся к единому виду, что и позволяет проводить вычисления над ними.
4. Слой кодировщика. Основной процесс алгоритм, будет объяснен далее.
5. Второй уровень нормализации.
6. Слой связи, представляющий собой два полносвязных слоя нейронной сети
7. Третий слой нормализации
8. Слой Вывода.

Итак, работа алгоритма выглядит следующим образом.

1. На вход подается предложение на русском языке.
2. Каждое слова предложения переводим в вектор
3. Разбиваем на 8 голов (оснований) и перемножаем с матрицами весов.
4. Получаем результат и проводим уменьшение размерности
5. Проводим слияние матриц, затем умножаем на матрицу весов и получаем выходной слой

Пункты 3-5 повторяются 12 раз

Полученный выходной слой преобразовывается в дерево синтаксического разбора предложения.

Визуализация работы алгоритма представлена на рисунках 2.5-2.6.



Рисунок 2.5. Описание работы алгоритма

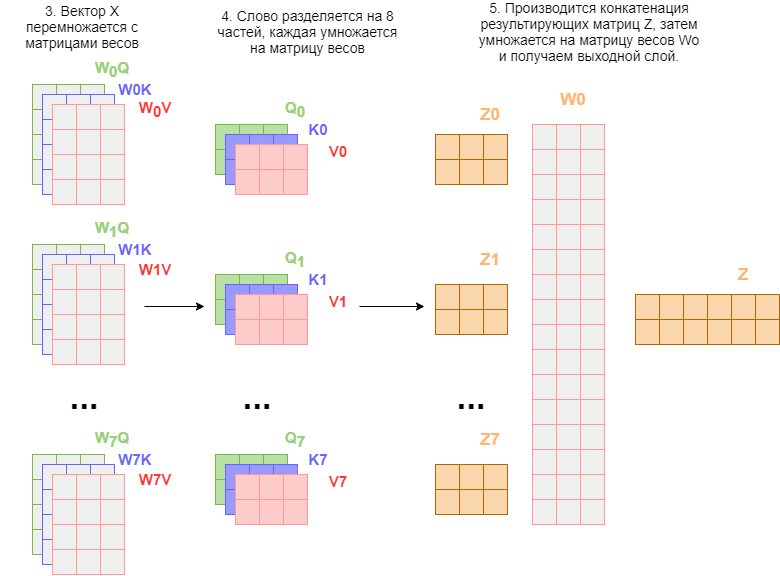


Рисунок 2.6. Описание работы алгоритма

# ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ

## Исходные данные и обучающая выборка

## Наборы данных в нейросетевых алгоритмах

При оценке возможности решения задачи с помощью алгоритмов машинного обучения, а в особенности, нейронных сетей, в первую очередь обращают внимание на наличие достаточно большой и разнообразной обучающей выборки [60].

Для разных задач выборка может представлять различный набор данных:

* Набор картинок
* Набор слов, букв, предложений, текстов
* Диалоги, сообщения в социальных сетях, “твиты” и другие

Первичный набор исходных данных принято называть генеральной совокупностью. Процесс формирования выборок из генеральной совокупности называется порождение данных. Выборка – это конечное подмножество элементов генеральной совокупности, изучив которое можно понять поведение исходного множества. Например, генеральная совокупность состоит из 150 тысяч посетителей сайта, а в выборку попали 250 из них [61].

Вероятностная модель порождения данных предполагает, что выборка из генеральной совокупности формируется случайным образом. Если все ее элементы одинаково случайно и независимо друг от друга распределены по исходному множеству (генеральной совокупности), выборка называется простой. Простая выборка является математической моделью серии независимых опытов и, как правило, используется для машинного обучения. При этом для каждого этапа Machine Learning необходим свой набор данных:

* для непосредственного обучения модели нужна обучающая выборка (training sample), по которой производится настройка (оптимизация параметров) алгоритма;
* для оценки качества модели используется тестовая (контрольная) выборка (test sample), которая, в идеальном случае, не должна зависеть от обучающей;
* для выбора наилучшей модели машинного обучения понадобится проверочная (валидационная) выборка (validation sample), которая также не должна пересекаться с обучающей.

Методы формирования обучающих и оценочных выборок зависят от класса задачи, решаемой с помощью машинного обучения:

* для задач классификации данные следует разделить так, чтобы в полученных наборах численное соотношение объектов разных классов было таким же, как в исходной генеральной совокупности;
* для задач регрессионного анализа необходимо одинаковое распределение целевой переменной в полученных наборах, которые будут использоваться для обучения и контроля качества.

При соблюдении этих условий объемы обучающей и оценочных выборок могут существенно различаться. Например, размер валидационного датасета может составлять всего 10% генеральной совокупности. Главное в формировании выборок – ни в коем случае не объединять обучающий датасет и с оценочными (тестовым и валидационным), поскольку это грозит переобучением модели Machine Learning. В этом случае модель получит высокие оценки качества в процессе тренировки, но не покажет такого результата на реальных данных [62].

После того, как выборка сформирована, наступают следующие процессы CRISP-DM: очистка данных и работа с признаками: генерация, трансформация, нормализация и отбрасывание лишних переменных, чтобы исключить мультиколлинеарность факторов и понизить размерность модели Machine Learning [63].

Датасет – набор данных – может быть размеченным или неразмеченным. Разные типы датасетов определяют различные методы обучения. При обучении с учителем необходим размеченный датасет – для задач классификации — это будет означать, что для каждой единицы данных этого набора заранее известен результат (класс, к которому эта единица принадлежит) [64]. При обучении без учителя результат заранее неизвестен и алгоритм должен сам распределить набор данных по классам. Отдельным типом данных в наборе является целевой признак – по сути это и есть разметка датасета. Целевой признак определяет насколько совокупность других параметров удовлетворяет желаемому решению [65].

В зависимости от варианта задачи классификации, целевой признак может выглядеть по-разному:

* один столбец с двоичными значениями (1/0, TRUE/FALSE и пр.): двухклассовая классификация (binary classification), когда каждый объект принадлежит только одному классу;
* несколько столбцов с двоичными значениями: задача классификации с пересекающимися классами (multi-label classification), когда один объект может принадлежать нескольким классам;
* один столбец с действительными значениями: регрессионный анализ, когда прогнозируется одна величина;
* несколько столбцов с действительными значениями: задача множественной регрессии, когда прогнозируется несколько величин.

Для задач анализа и обработки текста зачастую применяются так называемые корпусы языка. Существуют корпусы для английского, русского, испанского, французского и многих других языков. Корпус – это набор текстов, каждый из которых может быть, как размеченным, так и нет. Также, размеченными могут быть и слова или предложения в этом корпусе. Целевые функции также могут быть разными – синтаксический, грамматический, орфографический, морфологический или смысловой разбор текста представляют собой различную разметку. Для синтаксического анализ нам может потребоваться два корпуса SynTagRus и RusCorpora.

## Национальный корпус русского языка

В Национальный корпус русского языка включены прежде всего прозаические оригинальные тексты, представляющие русский литературный язык (с начала XVIII века), но также и (в меньшем объёме) переводные сочинения (параллельно с оригиналом), поэтические тексты, а также тексты, представляющие нелитературные формы современного русского языка: разговорную (записи устной речи, публичной и непубличной), диалектную [66].

Основной корпус — тексты, представляющие русский литературный язык, — можно подразделить на два главных массива, имеющих свои особенности: это современные письменные тексты (середина XX — начало XXI века) и ранние тексты (середина XVIII — середина XX века). По умолчанию поиск по этим массивам ведётся одновременно, задать хронологический диапазон (и иные параметры) можно на странице установки пользовательского подкорпуса.

Все тексты, входящие в основной корпус, проходят процедуру метаразметки и морфологической разметки. Морфологическая разметка осуществляется с помощью специальных программ автоматического морфологического анализа. В небольшой части основного корпуса (объемом 6 млн словоупотреблений; в дальнейшем эта цифра будет увеличена) произведено ручное снятие омонимии и дополнительная коррекция результатов работы программы автоматического морфологического анализа. Эта часть образует так называемый корпус со снятой омонимией, который может служить удобным полигоном для тестирования различных программ поиска, морфологического анализа и автоматической обработки текстов, а также для исследований современной русской морфологии, требующих повышенной точности поиска [67].

Представительный корпус современных текстов с морфологической разметкой является основным и самым объёмным из подкорпусов. В этот корпус входят различные типы текстов, представляющие современный русский литературный (письменный) язык:

* современная художественная проза разных жанров и направлений
* современная драматургия
* мемуарно-биографическая литература
* журнальная публицистика и литературная критика
* газетная публицистика и новости
* научные, научно-популярные и учебные тексты
* религиозные и религиозно-философские тексты
* производственно-технические тексты
* официально-деловые и юридические тексты

Тексты представлены в определенной пропорции, отражающей их долю в общем массиве современных текстов. Так, доля художественных текстов (включая драматургию и мемуары) составляет не более 40% [68].

Глубоко аннотированный (синтаксический) корпус

Данный фрагмент Национального корпуса русского языка содержит тексты, снабженные морфосинтаксической разметкой [69]. Это значит, что помимо морфологической информации, приписанной каждому слову текста, для каждого предложения задана его синтаксическая структура [70].

Синтаксическая структура предложения, используемая в глубоко аннотированном корпусе (ГАК), представляет собой дерево зависимостей, в узлах которого стоят слова предложения, а ветви помечены именами синтаксических отношений. Такое представление о синтаксической структуре предложения восходит к лингвистической модели «Смысл ⇔ Текст» И. А. Мельчука и А. К. Жолковского. Окончательный перечень синтаксических отношений, используемых в ГАК, а также целый ряд конкретных лингвистических решений, связанных с представлением синтаксической структуры предложения, был выработан в Лаборатории компьютерной лингвистики Института проблем передачи информации РАН. Силами коллектива этой Лаборатории и составлен ГАК. В отличие от морфологически размеченного фрагмента Национального корпуса русского языка, ГАК целиком состоит из структур со снятой морфологической и синтаксической омонимией.

## SynTagRus – синтаксический корпус

Отмеченные корпуса в первую очередь предназначены для обеспечения основы лингвистических исследований во всех случаях поля словарного запаса и грамматики (в том числе изменения, происходящие в языке) на протяжении всей своей истории). Существуют две существенно отличающиеся друг от друга области таких исследований. С одной стороны, существуют традиционные лингвистические исследования, для которых необходим массовый материал текстов: такой спрос гораздо легче удовлетворить, если в наличии есть хорошие и глубоко помеченные справочники [71]. С одной стороны, существуют традиционные лингвистические исследования, для которых необходим массовый материал текстов: такой спрос гораздо легче удовлетворить, если в наличии есть хорошие и глубоко помеченные справочники. С другой стороны, современная вычислительная лингвистика сама по себе становится страстным и заинтересованным пользователем таких справочников, как этот все чаще используются в качестве учебных комплектов в машинном обучении. В результате этого обучение, компьютерные программы расширяют свои возможности по извлечению сложных типов данных, которые содержатся в учебных наборах текстов, из новых текстов. Вообще говоря, чем глубже уровень аннотации тела, тем более продвинутые виды информации можно было бы узнать из тела [72].

Русский банк зависимостей "СинТагРус" в настоящее время содержит более 52 000 предложений (около 770 000 слов), относящихся к текстам различных жанров (современная фантастика, научно-популярные, газетные, журнальные и журнальные статьи за период с 1960 по 2013 год, тексты онлайн-новости и т.д.) и неуклонно растет [73]. Это неотъемлемая, но полностью автономная часть Российского национального корпуса разработан в рамках общенационального исследовательского проекта

Поскольку русский язык, как и другие славянские языки, имеет относительно свободный порядок слов, SynTagRus принял схему аннотирования на основе зависимости во многих отношениях параллельно с Пражским банков слов [74].

На сегодняшний день СИНТАГРУС - единственный российский корпус, поставляющий комплексные морфологические и синтаксические аннотации. Последняя представлена в виде полного дерева зависимостей, предусмотренного для каждого предложения. В дереве зависимостей узлы представляют собой слова, аннотированные символами части речи и морфологическими особенностями, в то время как дуги помечены синтаксической зависимостью от каждой фразы [75].

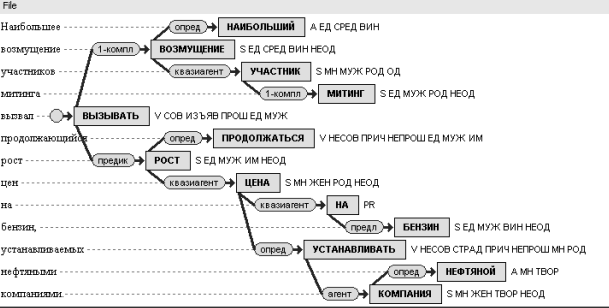


Рисунок 3.1 – Пример дерева зависимостей в SynTagRus

Типы зависимостей, используемые на рис. 1, включают в себя:

* Предик – подлежащее, отображает связь между сказуемым и остальными зависимыми частями.
* 1-компл (первое дополнение), которое обозначает отношение между предикатным словом (подлежащим), как основание и ее непосредственное дополнение как зависимое;
* агент – 2-ое дополнение, которое вводит связь между предикатным словом (словесным существительным или глаголом в пассивном залоге) в качестве головы и ее агента в инструментальном корпусе в качестве зависимого;
* квазиагент – обстоятельство или дополнение, которое соотносит любое предикатное существительное как голову со словом, реализует свою первую синтаксическую валентность как зависимую, если такое слово не может быть квалифицированного как агент существительного;
* опред – определение, которое соединяет основу существительного с прилагательным/частичной зависимостью, если последний служит дополнительным модификатором существительного;
* предл (предлог), который учитывает отношение между предлогом как головой и существительным как зависимым модификатором к существительному;

## Обучение нейронной сети

Чтобы соответствовать предварительному обучению, мы должны отформатировать входную последовательность модели в определенном формате. Для этого необходимо сначала маркировать, а затем правильно нумеровать тексты. Сложность здесь заключается в том, что каждая предварительно подготовленная модель, которую мы будем дорабатывать, требует точно такого же специфического препроцесса - токенирования и нумерации - что и препроцесс, используемый в предтренировочной части. К счастью, класс токенов от трансформаторов предоставляет корректные инструменты препроцессинга, соответствующие каждой предварительно подготовленной модели [76].

В библиотеке fastai препроцессинг данных выполняется автоматически при создании DataBunch. Как вы увидите в реализации DataBunch, токенайзер и нумератор передаются в аргументе процессора в следующем формате:



Рисунок 3.2 – Инициализация процессора для токенизации

Сначала проанализируем, как можно интегрировать токенайзер трансформаторов в функцию TokenizeProcessor.

Объект TokenizeProcessor принимает в качестве аргумента токенайзера объект Tokenizer. Объект BaseTokenizer реализует функцию tokenizer(t:str) → List[str], которая принимает текстовый t и возвращает список своих токенов.

Поэтому мы можем просто создать новый класс TransformersBaseTokenizer, унаследованный от BaseTokenizer, и перезаписать новую функцию токенайзера.



Рисунок 3.3 – Функция токенизации трансформера

В fastai объект NumericalizeProcessor принимает в качестве аргумента vocab объект Vocab. Из этого анализа я предлагаю два способа адаптировать нумератор fastai:

Можно получить список токенов и создать объект словаря. Создать новый класс TransformersVocab, который наследует от Vocab и перезаписать функции нумерации и текстурирования. Даже если первое решение кажется более простым, трансформеры не предоставляют для всех моделей простой способ восстановить список лексем. Поэтому я реализовал второе решение, которое работает для каждого типа моделей. Оно заключается в использовании функций convert\_tokens\_to\_ids и convert\_ids\_to\_tokens соответственно в нумерации и токенизации

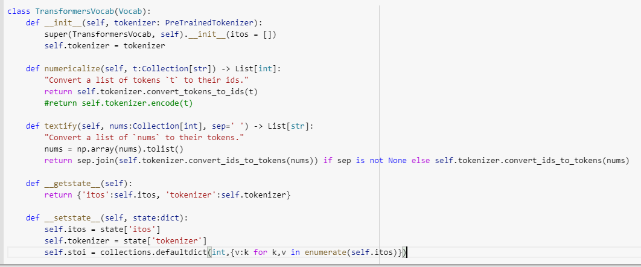


Рисунок 3.4 – Функция преобразования словаря

Для создания DataBunch необходимо обратить внимание на установку аргумента процессора в наш новый пользовательский процессор transformmer\_processor и правильное управление подкладкой.

Как указано в документации HuggingFace, BERT, RoBERTa, XLM и DistilBERT являются моделями с абсолютной встраиваемостью позиций, поэтому обычно рекомендуется расположить входы справа, а не слева. Что касается XLNET, то это модель с относительными встраиваниями позиций, поэтому можно разместить входы справа или слева [77].

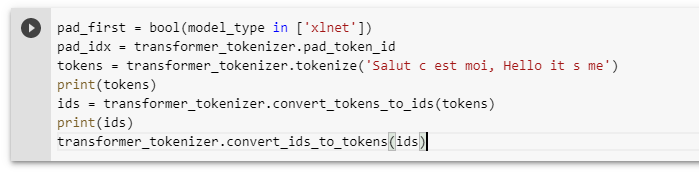


Рисунок 3.5 – Токенизация и конвертация данных

Как уже упоминалось, каждый метод форвардной модели всегда выводит кортеж с различными элементами в зависимости от модели и параметров конфигурации. В нашем случае нас интересуют только цифры. Одним из способов доступа к ним является создание пользовательской модели.

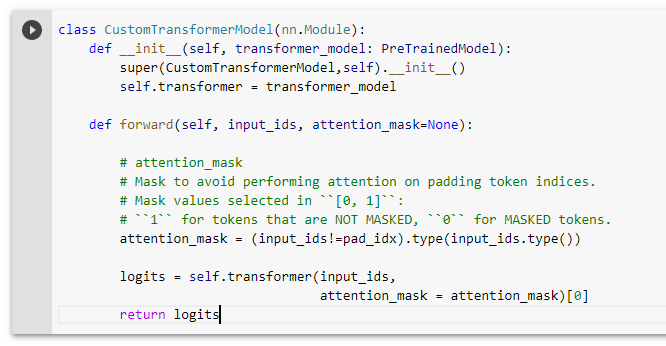


Рисунок 3.5 – Модель трансформера

Чтобы сделать трансформатор адаптированными к многоклассовой классификации, перед загрузкой предварительно подготовленной модели нам необходимо точно определить количество этикеток. Для этого можно изменить экземпляр конфигурации или модифицировать аргументы.

Рассмотрим метрики для подтверждения результатов обучения. В pytorch-трансформаторах компания HuggingFace внедрила два специальных оптимизатора - BertAdam и OpenAIAdam, которые были заменены на один AdamW-оптимизатор. Этот оптимизатор соответствует оптимизатору Pytorch Adam Api, поэтому становится просто интегрировать его в fastai. Стоит отметить, что для воспроизведения специфического поведения BertAdam необходимо установить correct\_bias = False.

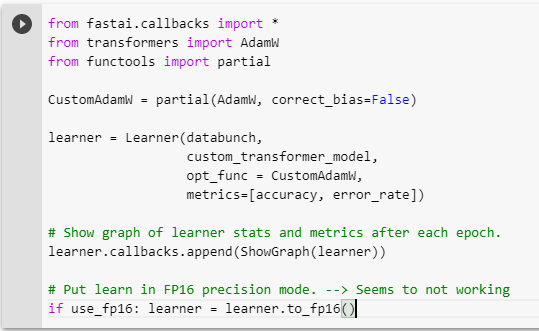


Рисунок 3.6 – Метрики модели

Для использования обучения дискриминирующих слоев и постепенного размораживания, fastai предоставляет один инструмент, который позволяет "разделить" модель структуры на группы. Инструкция по выполнению этого "разделения" описана в документации по fastai [78].

К сожалению, архитектуры моделей слишком отличаются друг от друга, чтобы создать уникальную общую функцию, которая может "разделить" все типы моделей удобным способом. Таким образом, вы должны будете реализовать пользовательское "разделение" для каждой архитектуры модели. Например, если мы используем модель RobBERTa и наблюдаем за его архитектурой, делая print(learningner.model) [79]. Нет необходимости приводить весь вывод модели, поэтому лишь отметим, что можно разделить модели на три класса:

* 1 эмбеддинг
* 12 преобразователей (трансформаторов)
* 1 классификатор

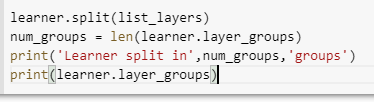


Рисунок 3.7 – Разделение модели по группам

Теперь мы, наконец, можем использовать все встроенные быстрые функции для обучения нашей модели. Как и в методе ULMFiT, мы будем использовать наклонные треугольные скорости обучения, дискриминируем скорость обучения и постепенно размораживаем модель.

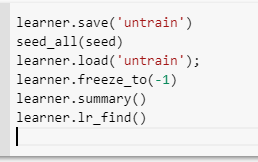


Рисунок 3.8 – Выборка слоев для обучения

Обучим модель, введя простую команду



Рисунок 3.9 – Команда для обучения модели

Получаем следующий график изменения функции потерь, в зависимости от скорости обучения:

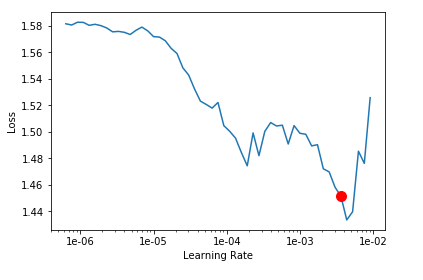


Рисунок 3.10 – График функции потерь

Мы выберем значение немного раньше минимума, где потери все равно улучшатся. Здесь 2x10^-3 кажется хорошим значением. Далее мы будем использовать цикл fit\_one\_cycle с выбранной скоростью обучения в качестве максимальной. Проведем обучение модели.

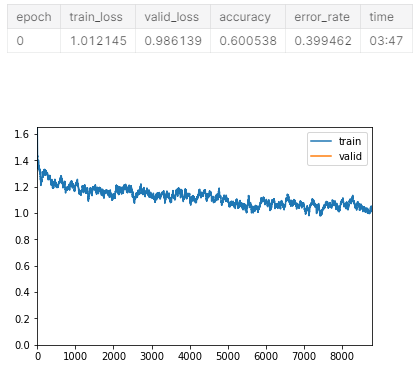
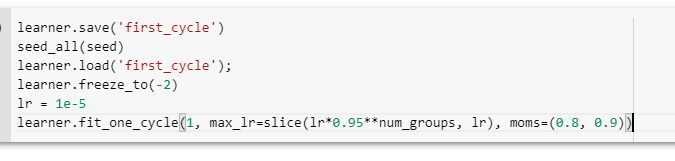


Рисунок 3.11 – Статистика обучения

Сохраняем полученные после первой тренировки веса, затем размораживаем слои второй группы и проводим еще одно обучение модели.



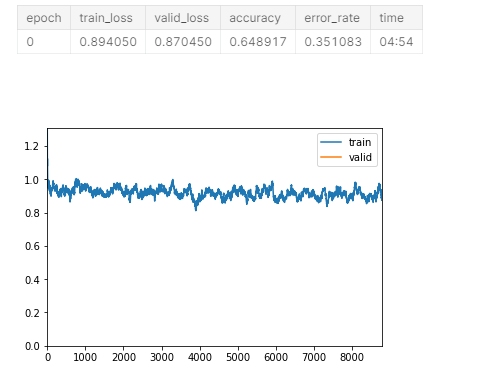


Рисунок 3.12 – Статистика обучения второй группы слоев

Теперь будем обучать все слои архитектуры:

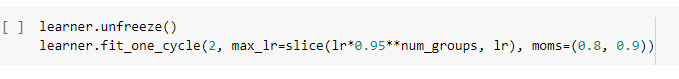


Рисунок 3.13 – Команда для размораживания всех слоев нейронной сети

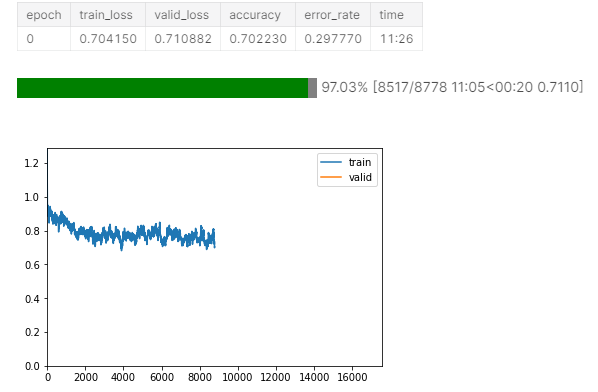


Рисунок 3.14 – Статистика обучения полной архитектуры

Архитектура готова, обучение нейронной сети произведено и веса получены. Остается только получить выходные данные и на их основе построить синтаксическое дерево.

Пример выходных данных после работы алгоритма:

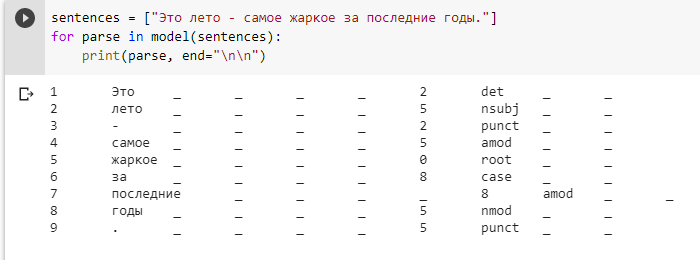


Рисунок 3.14 – Результат работы синтаксического анализатора

На выходе – словарь, состоящий из слов предложения и соответствующим им частям предложения, а также номеру в графе зависимостей.

## Формирование синтаксического дерева

Полученные после работы алгоритма результаты необходимо визуализировать, чтобы получить желаемое синтаксическое дерево. Такое дерево будет наглядным и понятным для пользователя, а также позволит привести результаты анализа к графовому виду для их дальнейшего преобразования в метаграф.

Для визуализации модели, полученные данные преобразовываются к структуре дерева библиотеки nltk [80]:



Рисунок 3.15 – Алгоритм работы преобразователя

В итоге получаем файл в формате json:



Рисунок 3.16 – Пример json-файла синтаксического дерева

Для визуализации данных, а в особенности древовидных структур особенно удобным и часто используемым инструментом является библиотека d3 для языка json [81]. Для формирования и создания визуального отображения результатов работы алгоритма использовалась именно эта библиотека.

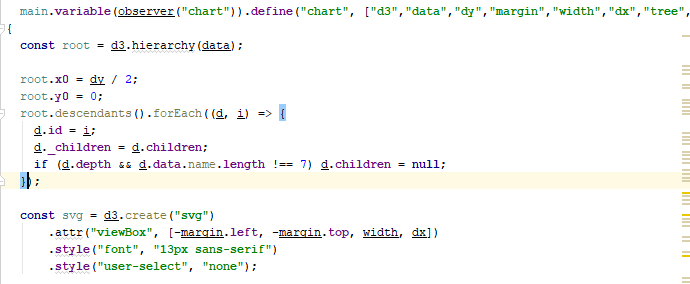
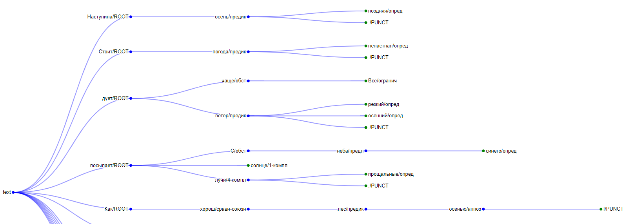


Рисунок 3.17 – Отрисовка дерева

Для примера возьмем текст известного русского писателя и поэта А.С. Пушкина:

*Наступила поздняя осень. Стоит ненастная погода. Всё чаще дует резкий осенний ветер. С синего неба посылает солнце прощальные лучи. Как хорош лес осенью! Воздух свеж. Жёлтые, оранжевые, багровые листья тихо падают с чёрных деревьев и медленно опускаются на холодную землю. Ничего не нарушает тишину леса, потому что они улетели на юг. В аллеях сада стало пусто, бесшумно. А ведь здесь летом было так радостно, чудесно! Из окрестных парков слетались сюда птицы на праздничный концерт. Осенью лес похож на терем расписной. Хорошо устроиться под белоствольной берёзкой и долго рассматривать золотые краски леса, лиловый краешек неба. Тихо, уютно.*



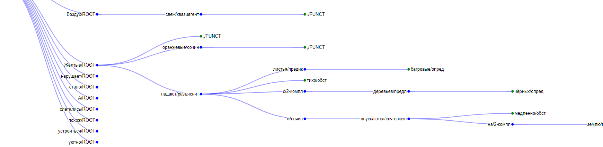


Рисунок 3.18 – Результат работы алгоритма после визуализации данных

## Результаты

При оценке результатов работы алгоритмов машинного обучения используются различные метрики. Для классификации это может быть достоверность, точность, правильность распределения результатов. Для задач синтаксического анализа и парсинга зачастую используются две метрики оценки:

* UAS – unlabeled attachment score – неразмеченная оценка вложений
* LAS – labeled attachment score – размеченная оценка

Первая оценка представляет собой процент правильного расположения основания дерева (сказуемого). Вторая метрика дает положительную оценку только при правильно угаданных частях предложения и всех связей в предложении.

На основании набора данных Universal Dependencies 2.3, который представляет собой набор размеченных слов на русском языке, были проведены исследований трех известных моделей, созданных для английского языка или мультиязычные, и модели, созданной в результате выполнения этого исследования.

Таблица 3.1 – Сравнение лучших результатов алгоритмов вложения

|  |  |  |
| --- | --- | --- |
| Название модели | UAS | LAS |
| [UD Pipe 2.3](http://ufal.mff.cuni.cz/udpipe) (Straka et al., 2017) | 90.3 | 89.0 |
| [UD Pipe Future](https://github.com/CoNLL-UD-2018/UDPipe-Future) (Straka, 2018) | 93.0 | 91.5 |
| UDify (multilingual BERT) (Kondratyuk, 2018 | 94.8 | 93.1 |
| **Собственная модель** | **95.2** | **93.7** |

Созданная модель оказалась лучше известных и используемых до этого для задач синтаксического анализа предложений русского языка.

# ЗАКЛЮЧЕНИЕ

В процессе исследования была поставлена задача синтезировать алгоритм и построить модель преобразования предложения на русском языке в синтаксический граф. Были проведены исследования вариантов и методов построения синтаксического графа, рассмотрены методики современного анализа текста, в том числе с помощью новейших технологий, таких как, машинное обучение и нейронные сети.

В результате исследования была определена математическая модель задачи, найден алгоритм решения, затем, на основе архитектуры BERT была построена модель нейронной сети, позволяющая преобразовать предложение на русском языке в синтаксический граф. Построенная модель оказалась лучше аналогов по метрикам, используемых для задач построения синтаксического дерева, а значит, задача, поставленная в исследовании, была выполнена.

Полученное синтаксическое дерево в дальнейшем можно использовать как одну из графовых структур метаграфа при задачах анализа текста. Для формирования полного метаграфа необходимого определить еще семантическую, контекстную и концептуальную связь внутри предложения и между предложениями в тексте.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Пескова О. В. Алгоритмы классификации полнотекстовых документов // Автоматическая обработка текстов на естественном языке и компьютерная лингвистика. — М.: МИЭМ (Московский государственный институт электроники и математики), 2011. — С. 170—212. — ISBN 978–5–94506–294–8.
2. Баранов Я.В., Радченко И.А., Миронов А.Ю. Использование средств обработки естественного языка для улучшения произношения на иностранном языке // Информатизация образования и науки. 2018. № 3 (39). С. 98-105.
3. Боярский К.К. Введение в компьютерную лингвистику. Учебное пособие. СПб: НИУ ИТМО, 2013. 72 с.
4. Анохин К.В. Когнитом: гиперсетевая модель мозга // Нейроинформатика-2015. XVII Всероссийская научно-техническая конференция. Сборник научных трудов. Ч. 1. М.: НИЯУ МИФИ. 2015. С. 18.
5. Стельмах М.А., Миснянкин В.Г., Кунац А.Ю., Костина А.В. Использование промежуточных языков представления для упрощения процесса перевода естественного языка в запросы к базе данных // Наука настоящего и будущего. 2017. Т. 1. С. 114-116.
6. Chernenkiy V.M., Terekhov V.I., Gapanyuk Yu.E. Predstavleniye slozhnikh setey na osnove metagrafov [Metagraph representation of complex networks]. Trudi XVIII vserossiyskoy konferencii “Neuroinformatics-2016” [Proc. XVIII all-russian conference “Neuroinformatics-2016”], Moscow, 2016, pp. 173-178.
7. Юргель В.Ю. Сложности моделирования естественного языка // Вестник науки и образования. 2019. № 23-1 (77). С. 12-14.
8. Samokhvalov E.N., Revunkov G.I. Gapanyuk Yu.E. Ispolzovaniye metagraphov dlya opisaniya semantiki i pragmatiki informatsionnykh sistem [Metagraphs for information systems semantics and pragmatics definition]. Vestnik MGTU im. N.E. Baumana, seriya “Priborostroeniye” [Herald of the Bauman Moscow State Technical University, “Instrument Engineering”], 2015, no. 1, pp. 83-99.
9. Survey of Text Mining I: Clustering, Classification, and Retrieval / Ed. by M. W. Berry. — 2004. — Springer, 2003. — 261 p. — [ISBN 0387955631](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/0387955631).
10. Черненький В.М., Терехов В.И., Гапанюк Ю.Е. Структура гибридной интеллектуальной информационной системы на основе метаграфов. Нейрокомпьютеры: разработка, применение. 2016. Выпуск №9. С. 3-14.
11. Dou Z., Wang X., Shi Sh., Tu Zh. Exploiting deep representations for natural language processing // Neurocomputing. 2020. Vol. 38 (621). P. 1-7.
12. Гапанюк Ю.Е., Ревунков Г.И., Федоренко Ю.С. Предикатное описание метаграфовой модели данных. Информационно-измерительные и управляющие системы. 2016. Выпуск № 12. С. 122-131.
13. Basu A., Blanning R. Metagraphs and their applications. Springer, 2007. 174 p.
14. Черненький В.М., Гапанюк Ю.Е., Ревунков Г.И., Терехов В.И., Каганов Ю.Т. Метаграфовый подход для описания гибридных интеллектуальных информационных систем. Прикладная информатика. 2017. № 3 (69). Том 12. С. 57–79.
15. Черненький В.М., Терехов В.И., Гапанюк Ю.Е. Представление сложных сетей на основе метаграфов // Нейроинформатика-2016. XVIII Всероссийская научно-техническая конференция. Сборник научных трудов. Ч. 1. М.: НИЯУ МИФИ, 2016. C. 173-178
16. Самохвалов Э.Н., Ревунков Г.И., Гапанюк Ю.Е. Использование метаграфов для описания семантики и прагматики информационных систем. Вестник МГТУ им. Н.Э. Баумана. Сер. «Приборостроение». 2015. Выпуск №1. С. 83-99.
17. Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In NIPS.
18. Giménez M., Palanca J., Botti V. Semantic-based padding in convolutional neural networks for improving the performance in natural language processing. A case of study in sentiment analysis // Neurocomputing. 2020. Vol. 78. P. 315-323.
19. Cai, Hongyun & W. Zheng, Vincent & Chen-Chuan Chang, Kevin. (2017). A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. IEEE Transactions on Knowledge and Data Engineering. 10.1109/TKDE.2018.2807452.
20. Kingma, D. and Ba, J. Adam: A method for stochastic optimization. ICLR, 2014.
21. J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, “Graphx: Graph processing in a distributed dataflow framework,” in OSDI, 2014, pp. 599–613.
22. Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, “Distributed graphlab: A framework for machine learning and data mining in the cloud,” Proc. VLDB Endow., vol. 5, no. 8, pp. 716–727, 2012.
23. P. Kumar and H. H. Huang, “G-store: High-performance graph store for trillion-edge processing,” in SC, 2016, pp. 71:1–71:12.
24. X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, “Community preserving network embedding,” in AAAI, 2017, pp. 203–209.
25. Hamilton L.M., Lahne J. Fast and automated sensory analysis: Using natural language processing for descriptive lexicon development // Food Quality and Preference. 2020. Vol. 83.
26. Nikolic D. The Human Side of AI // School of Finance and Management (DBIS). Frankfurt: Goethe-University, 2017.
27. Zhang, B.; Titov, I.; and Sennrich, R. 2019. Improving deep transformer with depth-scaled initialization and merged attention. arXiv preprint arXiv:1908.11365.
28. Faggella D. What is Machine Learning? // Emerj. URL: <https://emerj.com/ai-glossary-terms/what-is-machine-learning/> (дата обращения: 03.06.2020).
29. D. Wang, P. Cui, and W. Zhu, “Structural deep network embed- ding,” in KDD, 2016, pp. 1225–1234. [21] S. Cao, W. Lu, and Q. Xu, “Grarep: Learning graph represen- tations with global structural information,” in CIKM, 2015, pp. 891–900.
30. F. Tian, B. Gao, Q. Cui, E. Chen, and T. Liu, “Learning deep representations for graph clustering,” in AAAI, 2014, pp. 1293– 1299. [23] S. Cao, W. Lu, and Q. Xu, “Deep neural networks for learning graph representations,” in AAAI, 2016, pp. 1145–1152.
31. Основы Natural Language Processing для текста. [Электронный ресурс] // Хабрахабр: <https://habr.com/ru/company/Voximplant/blog/446738/>
32. Neil Houlsby, Andrei Giurgiu, Stanisław Jastrzebski “Parameter-Efficient Transfer Learning for” in WWW, 2013, pp. 37–48.
33. Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of lstms to learn syntaxsensitive dependencies. TACL.
34. Urvashi Khandelwal, He He, Peng Qi, and Daniel Jurafsky. 2018. Sharp nearby, fuzzy far away: How neural language models use context. In ACL.
35. Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. 2011. Stanford’s multi-pass sieve coreference resolution system at the conll-2011 shared task. In CoNLL.
36. David Marecek and Rudolf Rosa. 2018. Extracting syntactic trees from transformer encoder selfattentions. In BlackboxNLP@EMNLP.
37. M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, “Asymmetric transitivity preserving graph embedding,” in KDD, 2016, pp. 1105–1114.
38. Almeida, T. A., Hidalgo, J. M. G., and Yamakami, A. Contributions to the Study of SMS Spam Filtering: New Collection and Results. In Proceedings of the 11th ACM Symposium on Document Engineering. ACM, 2011.
39. Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. A neural probabilistic language model. Journal of Machine Learning Research, 2003.
40. Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. Enriching word vectors with subword information. ACL, 2017.
41. Brown, P. F., deSouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. Class-based n-gram models of natural language. Computational Linguistics, 1992.
42. Caruana, R. Multitask learning. Machine Learning, 1997.
43. Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., John, R. S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., et al. Universal sentence encoder. arXiv preprint arXiv:1803.11175, 2018.
44. Yang, B.; Tu, Z.; Wong, D. F.; Meng, F.; Chao, L. S.; and Zhang, T. 2018. Modeling localness for self-attention networks. arXiv preprint arXiv:1810.10182.
45. Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., St. John, R., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Strope, B., and Kurzweil, R. Universal sentence encoder for english. In EMNLP, 2019.
46. Huh, M., Agrawal, P., and Efros, A. A. What makes imagenet good for transfer learning? arXiv preprint arXiv:1608.08614, 2016.
47. A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, “Distributed large-scale natural graph factorization,” in WWW, 2013, pp. 37–48.
48. C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, “Network representation learning with rich text information,” in IJCAI, 2015, pp. 2111–2117.
49. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. PNAS, 2017.
50. C. Tu, W. Zhang, Z. Liu, and M. Sun, “Max-margin deepwalk: Discriminative learning of network representation,” in IJCAI, 2016, pp. 3889–3895.
51. Perez, E., Strub, F., de Vries, H., Dumoulin, V., and Courville, A. C. Film: Visual reasoning with a general conditioning layer. AAAI, 2018.
52. Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’14, pages 701–710, New York, NY, USA, 2014. ACM.
53. Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. Skip-thought vectors. In NIPS. 2015.
54. Xing Shi, Inkit Padhi, and Kevin Knight. 2016. Does string-based neural mt learn source syntax? In EMNLP.
55. Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.
56. Rosenfeld, A. and Tsotsos, J. K. Incremental learning through deep adaptation. IEEE transactions on pattern analysis and machine intelligence, 2018.
57. Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.
58. Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; and Bowman, S. R. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. arXiv preprint arXiv:1804.07461.
59. Sam Joshua Wiseman, Alexander Matthew Rush, Stuart Merrill Shieber, and Jason Weston. 2015. Learning anaphoricity and antecedent ranking features for coreference resolution. In ACL.
60. Long, J., Shelhamer, E., and Darrell, T. Fully convolutional networks for semantic segmentation. In CVPR, 2015.
61. Aditya Grover, Jure Leskovec. node2vec, Scalable feature learning for networks. Phoenix, Arizona: International Conference on Knowledge Discovery and Data Mining, 2016. – 4 p.
62. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In NIPS.
63. X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, “Community preserving network embedding,” in AAAI, 2017, pp. 203–209.
64. Kelly W. Zhang and Samuel R. Bowman. 2018. Language modeling teaches you more syntax than translation does: Lessons learned through auxiliary task analysis. In BlackboxNLP@EMNLP.
65. Pennington, J., Socher, R., and Manning, C. Glove: Global vectors for word representation. In EMNLP, 2014
66. F. Nie, W. Zhu, and X. Li, “Unsupervised large graph embedding,” in AAAI, 2017, pp. 2422–2428.
67. C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao, “Scalable graph embedding for asymmetric proximity,” in AAAI, 2017, pp. 2942– 2948.
68. Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. ICLR, 2014.
69. NLP — BERT & Transformer / BERT для решений задач анализа текста [Электронный ресурс] // medium.com: информ.-справочный портал. URL: <https://medium.com/@jonathan_hui/nlp-bert-transformer-7f0ac397f524>
70. Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In ICLR, 2017.
71. K. Georgiev and P. Nakov. A non-iid framework for collaborative filtering with restricted boltzmann machines. In ICML-13, pages 1148–1156, 2013.
72. Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R Bowman, Dipanjan Das, et al. 2018. What do you learn from context? probing for sentence structure in contextualized word representations. In ICLR.
73. F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu. Learning deep representations for graph clustering. In Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, pages 1293–1299, 2014.
74. Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. ICLR, 2018.
75. S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang. Heterogeneous network embedding via deep architectures. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 119–128. ACM, 2015.
76. Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In ICLR, 2017.
77. Документация deeppavlov [Электронный ресурс] <http://docs.deeppavlov.ai/en/master/>
78. Синтаксический анализ в NLTK. [Электронный ресурс] // Хабрахабр: <https://habr.com/ru/post/340574/>
79. API Reference — scikit-learn 0.21.2 documentation [Электронный ресурс] // URL: http://academim.org/art/pan1\_2.html (дата обращения: 17.06.2019).
80. Сандерс Дж., Кэндрот Э. Технология CUDA в примерах: введение в программирование графических процессоров: Пер. С англ. Силинкина А.А., научный редактор Борисов А.В. М.: ДМК Пресс, 2018. – 232 с.: ил. ISBN 978-5-97060-581-3
81. Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., Hughes, M., and Dean, J. Google’s multilingual neural machine translation system: Enabling zero-shot translation. ACL, 2017.

# ПРИЛОЖЕНИЕ A

Московский государственный технический университет имени Н. Э. Баумана

Кафедра «Системы обработки информации и управления»

|  |  |  |  |
| --- | --- | --- | --- |
| Утверждаю: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Согласовано: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | "\_\_\_"\_\_\_\_\_\_\_\_\_\_2020 г. |  | "\_\_\_"\_\_\_\_\_\_\_\_\_\_\_2020 г. |

**АНАЛИЗ ТЕКСТОВ НА ОСНОВЕ ВЕКТОРНОГО**

**ПРЕДСТАВЛЕНИЯ ГРАФОВ ЗНАНИЙ**

Техническое задание

(вид документа)

Листы А4

(вид носителя)

8

(количество листов)

|  |  |
| --- | --- |
| Исполнитель: | студент группы ИУ5-43 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Иванников А.В. | |
| "\_\_\_\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. | |

Москва - 2020

Содержание

1. Наименование 3

2. Основание для разработки 3

3. Исполнитель 3

4. Назначение и цель разработки 3

5. Содержание работы 4

5.1. Задачи, подлежащие решению 4

5.2. Требования к анализу существующих подходов 4

5.3. Требования к разработке модели 5

5.4. Требования к анализу результатов моделирования 5

5.5. Требования к входным и выходным данным 5

5.5.1. Требования к входным данным 6

5.5.2. Требования к выходным данным 5

5.6. Требования к надежности 5

5.7. Требования к составу технических средств 5

6. Этапы разработки 6

6.1. Сроки выполнения отдельных этапов работ 6

7. Техническая документация, предъявляемая по окончании работы 6

8. Порядок приема работы 7

9. Дополнительные условия 7

**1. Наименование**

Полное название «Анализ текстов на основе векторного представления графов знаний»

**2. Основание для разработки**

Основанием для разработки является задание на выпускную квалификационную работу магистра кафедры «Системы обработки информации и управления» Московского государственного технического университета им. Н. Э. Баумана в 2020 году.

**3. Исполнитель**

Студент МГТУ им. Н. Э. Баумана группы ИУ5-43 Иванников А.В.

**4. Назначение и цель разработки**

Квалификационная работа «Анализ текстов на основе векторного представления графов знаний» посвящена исследованию вариантов, механизмов и подходов реализации синтаксического анализа текста на русском языке с использованием методов машинного обучения.

Цель разработки – исследование и создание модели преобразования текста на русском языке в синтаксический граф.

**5. Содержание работы**

***5.1. Задачи, подлежащие решению***

1. Анализ существующих подходов к решению задачи;
2. Разработка модели преобразования текста в синтаксический граф;
3. Анализ результатов моделирования;
4. Разработка документации.

***5.2. Требования к анализу существующих подходов***

При проведении анализа должны быть рассмотрены передовые техники работы с текстом, включая методы машинного обучения, в том числе – нейронные сети, интеллектуальные системы, а также проведено исследование существующих публикаций на русском и иностранных языках.

После обзора существующих методик будет приведено направление исследования в области анализа текста. Затем необходимо сформулировать цели и задачи, решаемые в ходе научно-исследовательской работы.

***5.3. Требования к разработке модели преобразования текста в граф***

Представить общие теоретические положения о текстовом анализе, графовом представлении данных, описать методы выполнения сформулированных задач. Описать и создать модель, выполнить программную реализацию модели. Описание представить в виде алгоритма, с пояснением каждого шага. Разработанную имитационную модель необходимо описать и протестировать. Разработать методы для оценки эффективности модели, провести оценку.

***5.4.*** ***Требования к анализу результатов моделирования***

Провести эксперимент по преобразованию текста в граф, описать экспериментальную установку, исходные данные для эксперимента. Привести результаты эксперименты и обоснованные выводы. С помощью результатов

***5.5. Требования к входным и выходным данным***

***5.5.1. Требования к входным данным***

Входные данные – текст на русском языке, соответствующий всем правилам русского языка.

***5.5.2. Требования к выходным данным***

Выходные данные – синтаксический граф, основанный на входных данных.

***5.6. Требования к надежности***

Специальные требования к обеспечению надежности не предъявляются.

***5.7. Требования к составу технических средств***

Для корректной работы программного модуля пользователям достаточны компьютеры со следующими минимальными характеристиками:

* Процессор с частотой не менее 1 ГГц;
* Размер оперативной памяти не менее 1 Гб;
* Видеоадаптер и монитор, способные обеспечить графический режим не менее 1024 x 768 точек с 16 битной цветопередачей;
* Система устройства хранения данных не менее 1 ГБ;
* Наличие манипулятора «мышь» или другого указывающего устройства;
* Наличие клавиатуры;
* Обеспечение доступа в сеть Интернет.

Минимальные системные требования для серверной части проекта:

* Процессор с частотой не менее 2 ГГц;
* Размер оперативной памяти не менее 4 Гб;
* Система устройства хранения данных не менее 32 ГБ;
* Обеспечение доступа в сеть Интернет.

**6. Этапы разработки**

***6.1. Сроки выполнения отдельных этапов работ***

График приведен в соответствии с учебными неделями МГТУ им. Н.Э. Баумана 2 курса 4 семестра.

Таблица 1. Этапы разработки

|  |  |  |
| --- | --- | --- |
| №  п/п | Наименование этапа и содержание работ | Сроки исполнения |
| 1. | Разработка и утверждение ТЗ | Февраль 2020 г. |
| 2. | Исследование предметной области | Февраль 2020 г. |
| 3. | Анализ существующих подходов к решению задачи | Март 2020 г. |
| 4. | Разработка модели преобразования текста в синтаксический граф | Март 2020 г. |
| 5. | Анализ результатов моделирования | Март - май 2020 г. |
| 6. | Проведение экспериментов | Май 2020 г. |
| 7. | Оформление документации | Май - июнь 2020 г. |
| 8. | Защита работы | Июнь 2020 г. |

**7. Техническая документация, предъявляемая по окончании работы**

1. Техническое задание.

2. Расчётно-пояснительная записка.

3. Программа и методика испытаний.

4. Руководство пользователя.

5. Текст программных модулей.

**8. Порядок приема работы**

Приём и контроль осуществляется в соответствие с методикой испытаний

**9. Дополнительные условия**

Данное техническое задание может уточняться в установленном порядке.

# ПРИЛОЖЕНИЕ Б

Графическая часть

