**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА ТЕХНОЛОГИЙ ПРОГРАММИРОВАНИЯ

Направление: 09.03.03 – «Прикладная информатика»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

**Разработка алгоритма обработки естественного языка при помощи машинного обучения с использованием нейросетей BERT**

**Работа завершена:**

Студент 4 курса

группы 09-852

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. \_\_\_\_\_\_\_\_\_\_\_\_ Саидмуродов С.С.

**Работа допущена к защите:**

Научный руководитель

старший преподаватель

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. \_\_\_\_\_\_\_\_\_\_\_\_ Прокопьев Н.А.

Заведующий кафедрой

канд. экон. наук, доцент

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. \_\_\_\_\_\_\_\_\_\_\_\_ Вахитов Г.З.

Казань – 2022

Оглавление

[Оглавление 2](#_Toc105000592)

[Введение 3](#_Toc105000593)

[1. Формирование требований 6](#_Toc105000594)

[1.1. Изучение предметной области машинного обучения и нейронных сетей 6](#_Toc105000595)

[1.2. Анализ подходов к обработке естественного языка 7](#_Toc105000596)

[1.2.1. Традиционные языковые модели 7](#_Toc105000597)

[1.2.2. Нейросетевые архитектуры 11](#_Toc105000598)

[1.2.3. Модели на основе архитектуры Трансформер 14](#_Toc105000599)

[1.3. Техническое задание 16](#_Toc105000600)

[2. Проектирование платформы для обучения нейросетевой модели 19](#_Toc105000601)

[2.1. Используемые модели нейронных сетей 19](#_Toc105000602)

[2.2. Формализация алгоритмов основного анализа текста 25](#_Toc105000603)

[2.3. Структура платформы для обучения нейросетевой модели 29](#_Toc105000604)

[3. Экспериментальная реализация алгоритма анализа текста 31](#_Toc105000605)

[3.1. Подготовка данных для обучения 31](#_Toc105000606)

[3.2. Реализация платформы для обучения нейросетевой модели 33](#_Toc105000607)

[3.3. Результаты машинного обучения 33](#_Toc105000608)

[4. Тестирование по результатам обучения модели для основного анализа текста 34](#_Toc105000609)

[4.1. Экспериментальная оценка результатов обучения 34](#_Toc105000610)

[4.2. Целевой оценочный тест реализации алгоритма 34](#_Toc105000611)

[Заключение 35](#_Toc105000612)

[Список использованных источников 36](#_Toc105000613)

Введение

Для понимания текста людьми и программами искусственного интеллекта (ИИ) часто требуется правильное использование заглавных букв и знаков препинания. Простые предложения, состоящие из нескольких слов, и люди, и ИИ могут читать и обрабатывать, даже если использование заглавных букв или знаков препинания отсутствует или неправильно. Например, в случае голосовых команд обработка обычно выполняется только для распознанных слов нижнего регистра. Однако, если объем текста, подлежащего анализу, увеличивается, например, до целых абзацев или страниц, то даже для человека быстрое понимание его смысла становится сложной задачей. Это было изучено Джонсом и др. (2003), которые проанализировали влияние заглавных букв и пунктуации на читаемость расшифровки речи в текст.

Ранние работы рассматривали пунктуацию только как подсказки с точки зрения читателя к возможным просодическим характеристикам и паузам текста (Markwardt, 1942). Нанберг (1990) утверждает, что пунктуация играет гораздо большую роль. Кроме того, знаки препинания классифицируются как разграничивающие, разделяющие и устраняющие неоднозначность. Некоторые метки, такие как запятая, могут принадлежать к нескольким категориям, поскольку они могут выполнять несколько ролей. Джонс (1994) доказывает, что «для более длинных предложений реального языка грамматика, использующая знаки препинания, значительно превосходит аналогичную грамматику, которая их игнорирует». Основываясь на этом и других подобных выводах, современные языковые модели считают пунктуацию частью своего словарного запаса. Сюда входят новейшие модели, такие как BERT, ELMo, OpenAI GPT-2 и GPT-3.

Алгоритмы обработки естественного языка (NLP – Natural language processing), такие как распознавание именованных сущностей (NER – Named entitiy recognition), идентификация части речи, анализ зависимостей, машинный перевод (MT – Machine translation), используют заглавные буквы в качестве признаков обрабатываемого в данный момент слова, в то время как пунктуация используется. как признаки для соседних слов. Например, Stanford Named Entity Recognizer рассматривает признаки на основе формы слова. Это означает построение представления слова на основе типа символов, встречающихся в слове. Было предложено несколько алгоритмов представления формы слова, но общая идея состоит в том, чтобы закодировать прописную букву определенным символом, скажем, «X», строчную букву «x» и цифру «d». В этом случае слово типа «McDonald» станет «ХхХхххх». Работа любых таких алгоритмов возможна только в том случае, если слова правильно представлены в виде прописных и строчных букв.

Особое внимание следует уделить системам автоматического распознавания речи (ASR – Automatiс speech recognition). Первичный вывод таких систем обычно состоит из необработанного текста с использованием одного и того же регистра (нижнего, либо верхнего регистра) и без знаков препинания. В таких ситуациях перед применением дальнейших алгоритмов NLP требуется дополнительная предварительная обработка, чтобы восстановить правильный регистр букв и пунктуацию. Их иногда называют «богатыми транскрипциями». Одна из первых инициатив, касающихся автоматической расширенной транскрипции разговорной речи, началась в 2002 году в контексте программы DARPA «Эффективное, доступное повторное использование речи в текст» (EARS – Effective, Affordable, Reusable Speech-to-text), целью которой было улучшение уровня развития алгоритмов обработки языка. С этой целью NIST (National institute of Standards and Technology) выпустил серию обширных наборов данных для оценки транскрипции, чтобы помочь в оценке таких систем.

Несмотря на то, что большой объем данных, требующих восстановления заглавных букв и пунктуации, поступает из систем ASR, необходимо также учитывать и другие источники. Миллер и др. (2000) идентифицируют другие источники шума в виде текста, полученного с помощью оптического распознавания символов (OCR), или в некоторых газетных статьях. В этих случаях отсутствие надлежащей буквы или пунктуации затрагивает не весь текст, а его части. В случае OCR некоторые знаки препинания могут быть не распознаны, в то время как в случае некоторых статей первое предложение или абзац могут быть написаны только заглавными буквами. Кроме того, в случае коротких текстовых сообщений (SMS), чатов, твитов или других действий в микроблогах люди также могут игнорировать правильный регистр и пунктуацию.

Одна из трудностей, при создании человеко-компьютерных интерфейсов с использованием естественного языка, с которыми приходится сталкиваться, связана с непоследовательным использованием пользователем пунктуации и использования заглавных букв. В этом контексте многие подходы пытаются скрыть проблему, удаляя все знаки препинания и заглавные буквы как из данных обучения, так и из входных данных, полученных во время работы. Кроме того, Coniam (2014) также проанализировал вывод чат-ботов с точки зрения человека, использующего эти программы для изучения английского как второго языка. Он смог определить проблемы с заглавными буквами и пунктуацией даже в произведенном тексте. Тем не менее, он утверждает, что для коротких предложений, создаваемых чат-ботами, «переход на английский язык за счет все более широкого распространения текстовых сообщений делает спорным вопрос о том, можно ли считать эти проблемы важными в наши дни».

# Формирование требований

## Изучение предметной области машинного обучения и нейронных сетей

Итак, разберемся что же такое машинное обучение. В Википедии можно найти следующее определение: «Машинное обучение — класс методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение за счёт применения решений множества сходных задач». Так же можно найти более современное определение машинного обучения, данное Томом Митчеллом: «A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E». Различают два основных типа обучения:

* Дедуктивное обучение, или обучение с учителем, предполагает формализацию знаний экспертов и их перенос в компьютер в виде базы знаний
* Обучение по прецедентам, или индуктивное обучение (обучение без учителя), основано на выявлении эмпирических закономерностей в собранных данных.

В первом случае важна правильно составленная и размеченная обучающая выборка данных. Для этого необходимо выделить важные признаки данных, и заранее определить правильный ожидаемый результат для каждого набора данных. Дедуктивное обучение принято относить к области экспертных систем, поэтому иногда под машинным обучением понимают дедуктивное обучение.

Второму случаю присуще обучение на основе неразмеченных данных, и целью алгоритмов является определение признаков данных и их приоритетов. Многие методы, применяемые в обучении без учителя тесно связаны с извлечением информации и анализом данных.

Так как машинное обучение является достаточно крупной областью науки, далее более детально будут рассмотрены только подходы к обработке естественного языка, подробнее с машинным обучением можно ознакомится в приведенных источниках.

## Анализ подходов к обработке естественного языка

В данной работе рассмотрим подходы, основанные на машинном обучении. Для обработки естественного языка с помощью алгоритмов машинного обучения необходимо использование языковых модель. Языковая модель – это вероятностное распределение на множестве словарных последовательностей. Другими словами, языковые модели могут предсказывать последовательность слов исходя из заданного контекста. Далее будут коротко описаны методы решения задач обработки естественного языка с применением таких технологий как BoW, TF-IDF, Word2Vec, GloVe, fastText, GRU, LSTM, BERT и ELMO.

### Традиционные языковые модели

#### BoW

BoW(от англ. Bag of words – мешок слов) является наиболее простым и часто встречающимся векторным представлением. BoW работает следующим образом: на вход подается анализируемый текст, для каждого предложения строится вектор с длиной n-элементов, где n – это количество уникальных токенов в тексте. Каждый элемент вектора хранит в себе частоту вхождения того или иного токена в предложении. Токеном может являться как слово, так и n-грамма.



Рисунок 1. Пример векторного представления текста по методике BoW, при этом исполуется 3-граммная модель

Представления BoW часто используются в методах классификации документов, где частота каждого слова, двухсловного или трехсловного слова является полезной функцией для обучения классификаторов. Одна из проблем с представлениями мешка слов заключается в том, что они не кодируют никакой информации относительно значения данного слова. В BoW вхождения слов имеют одинаковый вес независимо от того, как часто и в каком контексте они встречаются. Однако в большинстве задач NLP одни слова более важны, чем другие.

#### TF-IDF

TF-IDF (от англ. TF — term frequency, IDF — inverse document frequency) — статистическая мера, которая используется для оценки значимости слова в контексте документа, являющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции. Мера TF-IDF находится путем перемножения частоты слова в документе и обратной частоты слова в корпусе документов:

Для нахождения частоты слова используют следующую формулу:

где – число вхождений слова в документ, а знаменатель – общее число слов в документе. Обратная частота документа – инверсия частоты, с которой некоторое слово встречается в документах коллекции находится следующей формулой:

где – число документов, – число документов из коллекции , в которых встречается слово (когда )

#### Word2Vec

Word2Vec– совокупность моделей на основе нейронных сетей, используемых для получения векторных представлений слов. Используется для изучения семантики естественных языков. Word2Vec разработан исследовательской группой Google под руководством чешского ученого Томаса Миколова в 2013 году. Word2Vec включается в себя две архитектуры моделей: Continuous bag-of-words (CBOW) для предсказания слова наиболее подходящего для данного контекста (при этом порядок слов не важен как и в BoW) и Continuous skip-gram используется для получения набора схожих слов со входящим словом

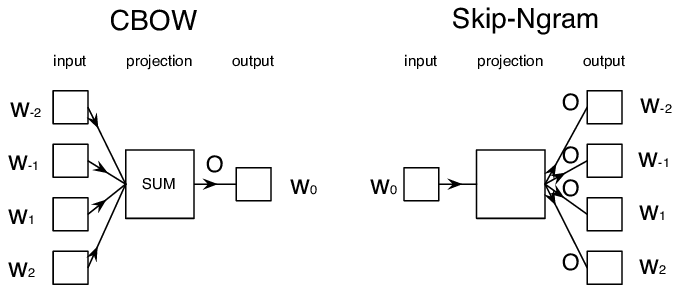


Рисунок 2. Наглядное сравнение архитектур Word2Vec

#### GloVe

GloVe (от англ. Global Vectors – глобальные векторы) – еще одна языковая модель, основанная на индуктивном обучении. В этой модели слова отображаются в векторном пространстве, где расстояние между словами определяет их семантическое сходство. GloVe создавался как проект с открытым исходным кодом в Стэнфорде в 2014 году. Эту модель можно использовать для поиска взаимосвязей между такими словами, такими как синонимы, отношения между компанией и продуктом, почтовые индексы и города и т. д. Хотя векторы GloVe быстрее обучаются, ни GloVe, ни Word2Vec, как показала практика, не дают однозначно лучших результатов, при тестировании на одних и тех же наборах данных.

#### fastText

fastText – это библиотека обучения моделей векторных представлений и классификации текста разработанная командой исследователей Facebook в 2018 году. fastText создан на основе Word2Vec путем обучения векторных представлений для каждого слова и n-грамм, найденных в каждом слове. Значения представлений усредняются в один вектор на каждом шаге обучения. Хотя такой подход добавляет много дополнительных вычислений к обучению, это позволяет векторным представлениям слов кодировать информацию о подсловах. Было показано, что векторы FastText более точны, чем векторы Word2Vec, по ряду различных показателей. На сегодняшний день Facebook предоставляет предварительно обученные модели для 294 языков.

### Нейросетевые архитектуры

В дополнение к лучшему представлению векторов слов появление нейронных сетей привело к достижениям в архитектуре машинного обучения, которые сделали возможными решения таких проблем как нахождение референционального тождества, анализ тональности текста, распознавание именованных сущностей, классификация намерений и так далее. В этом разделе коротко опишем существующие нейросетевые архитектуры.

#### Нейронные сети с прямой связью

Появление линейных сетей с прямой связью, также известных как многослойные персептроны (Multi-layer perceptrons - MLP) в NLP, открыло возможности для нелинейного моделирования. Эта разработка помогает с НЛП, потому что бывают случаи, когда пространство встраивания может быть нелинейным.



Рисунок 3. Схема архитектуры MLP

Однако это развитие само по себе не привело к существенной революции в NLP, поскольку MLP не могут моделировать порядок слов. В то же время как MLP открывают двери для незначительных улучшений в таких задачах, как классификация языков, где решения могут приниматься путем моделирования независимых частот символов, но для более сложных или неоднозначных задач автономные MLP не подходят.

#### Сверточные нейронные сети

Сверточные нейронные сети (Convolutional neural networks – CNN) пришли в NLP из компьютерного зрения. В NLP вместо свертки по пикселям сверточные фильтры применяются к группам векторов слов. В NLP CNN могут моделировать порядок слов, именно благодаря этой особенности они значительно улучшили решение задач классификации.

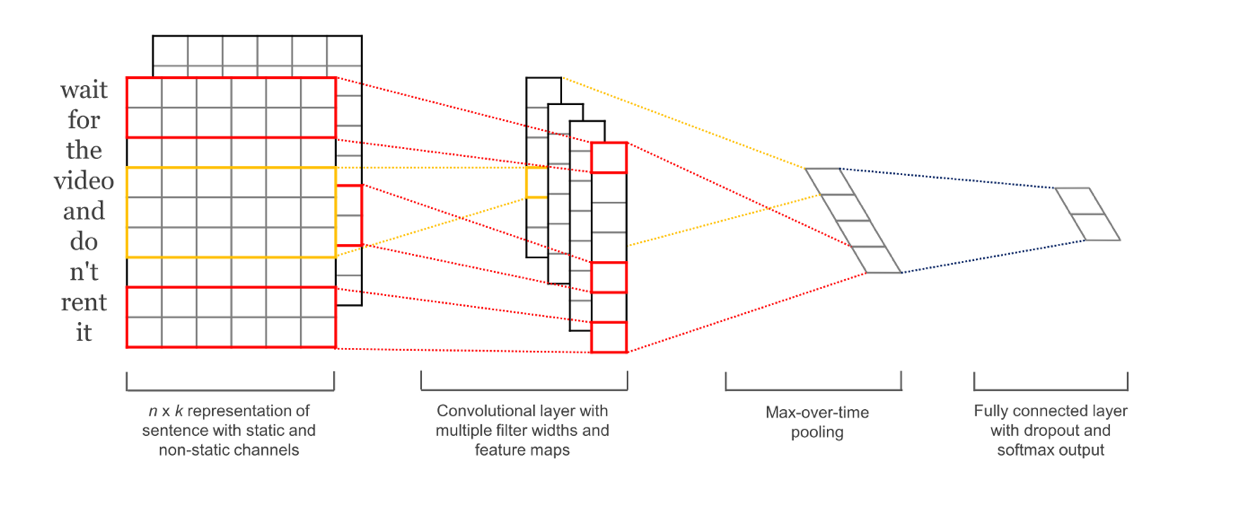


Рисунок 4. Применение сверточной нейронной сети для классификации предложений

#### Рекуррентные нейросети

Опираясь на локальное упорядочение, обеспечиваемое CNN, рекуррентные нейронные сети (RNN) и их варианты с закрытыми ячейками, такие как ячейки долговременной памяти (Long Short Term Memory Cells – LSTM) и закрытые рекуррентные единицы (Gated Recurrent Units – GRU), предоставляют механизмы для моделирования последовательного упорядочения и средних зависимостей в тексте, таких как влияние слова в начале предложения на конец предложения.

Несмотря на то, что стандартные архитектуры RNN привели к невероятным прорывам в области обработки естественного языка, они сталкиваются с множеством проблем. Хотя теоретически они могут фиксировать долгосрочные зависимости, они, как правило, испытывают трудности при моделировании более длинных последовательностей, что все еще остается нерешенной проблемой. Одной из причин неоптимальной производительности стандартных моделей кодера-декодера RNN для задач на последовательности, таких как NER или перевод, является то, что они равномерно взвешивают влияние каждого входного вектора на каждый выходной вектор, когда в действительности конкретные слова во входной последовательности могут иметь большее значение на разных временных шагах. Механизмы внимания обеспечивают средства взвешивания контекстуального воздействия каждого входного вектора на каждое выходное предсказание RNN. Эти механизмы отвечают за большую часть текущего или почти современного состояния обработки естественного языка. Механизм копирования — это дополнительный уровень, применяемый во время декодирования, который решает, лучше ли генерировать следующее слово из исходного предложения или из общего словаря встраивания.

#### ELMo

ELMo (от англ. Embeddings from Language Model – Векторные представления из языковой модели) - это модель, которая генерирует вложения для слова на основе контекста, в котором оно появляется, таким образом генерируя несколько разные вложения для каждого его появления.

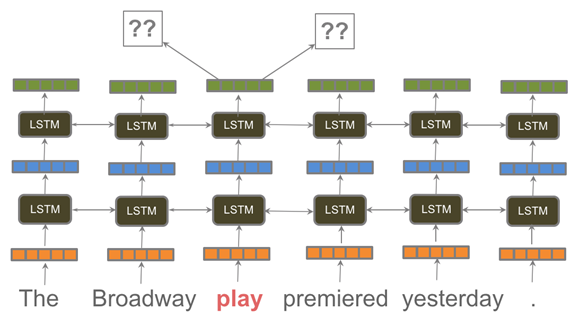


Рисунок 5. Пример работы ELMo

Например, слово «play» в предложении на Рисунке 5, будет закодировано ELMo как глагол «играть» и как существительное «спектакль», то есть получим два разных представления. В стандартных вложениях слов, таких как GloVe, FastText или Word2Vec, каждый экземпляр игры слов будет иметь одинаковое представление. На основе ELMo была разработана архитектура BERT.

### Модели на основе архитектуры Трансформер

По аналогии с рекуррентными нейронными сетями (РНС) трансформеры предназначены для обработки последовательностей, таких как текст на естественном языке, и решения таких задач как машинный перевод и автоматическое реферирование. В отличие от РНС, трансформеры не требуют обработки последовательностей по порядку. Например, если входные данные — это текст, то трансформеру не требуется обрабатывать конец текста после обработки его начала. Благодаря этому трансформеры распараллеливаются легче чем РНС и могут быть быстрее обучены. Так же в отличии от классических языковых моделей, модели на основе Трансформер обучают контекстно-зависимые векторные представления, в частности BERT учитывает двусторонний контекст, что позволяет ему лучше воспринимать многозначные слова.

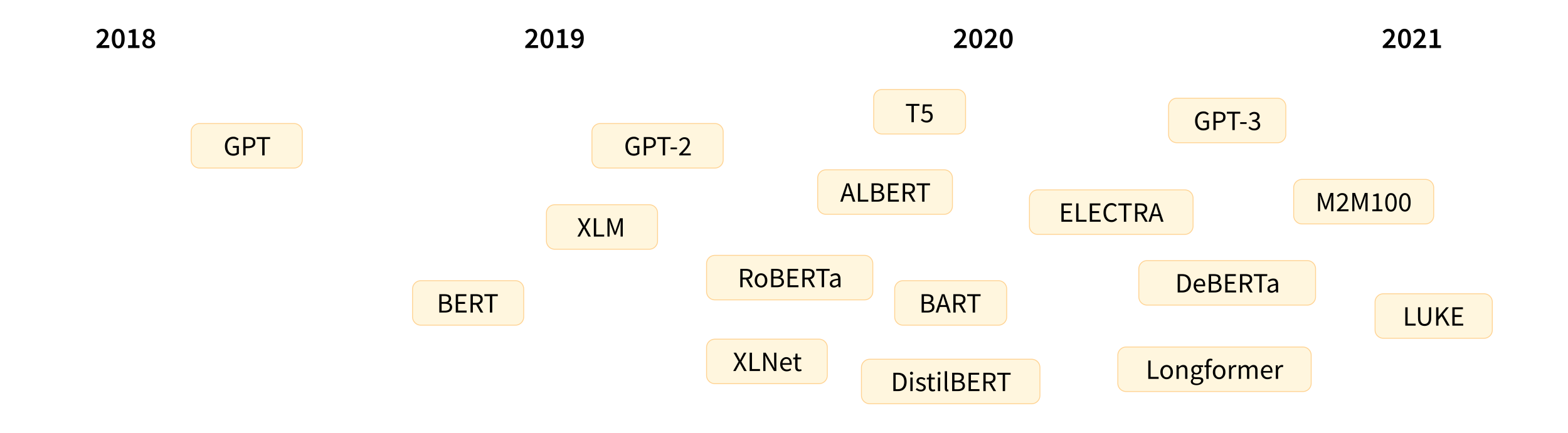


Рисунок 6. График появления новейших моделей на основе архитектуры Трансформер

Архитектура Трансформер была представлена ​​в июне 2017 года. Основное внимание в первоначальном исследовании уделялось задачам перевода. За этим последовало введение нескольких влиятельных моделей, в том числе:

1. GPT, первая предварительно обученная модель Трансформер, использовалась для тонкой настройки различных задач NLP и дала самые современные результаты.
2. BERT, еще одна большая предварительно обученная модель, предназначенная для создания более качественных сводок предложений
3. DistilBERT, оптимизированная версия BERT, которая на 60 % быстрее, на 40 % меньше памяти и при этом сохраняет 97 % производительности BERT.
4. BART и T5, две большие предварительно обученные модели, использующие ту же архитектуру, что и исходная модель Трансформер
5. GPT-3, более крупная версия GPT-2, способная хорошо выполнять различные задачи без необходимости точной настройки (так называемое обучение с нулевым выстрелом).

Этот список далеко не исчерпывающий, и он предназначен только для того, чтобы выделить несколько значимых моделей Трансформеров. В целом их можно разделить на три категории:

1. GPT-подобные (также называемые авторегрессивными моделями Трансформер, auto-regressive)
2. BERT-подобные (также называемые моделями Трансформер с автоматическим кодированием, auto-encoding)
3. Подобные BART/T5 (также называемые моделями преобразования последовательности в последовательность, sequence-to-sequence)

## Техническое задание

Ранее было сказано, что пунктуация и капитализация в тексте являются важными для понимания и дальнейшей обработки. Целью данной работы будет разработка алгоритма для восстановления пунктуации и капитализации для татарского и русского языков.

Разрабатываема система должна удовлетворять следующим принципам проектирования информационных систем:

1. Принцип системности - Подход к исследуемому объекту как единому целому. Это позволяет выявить многообразие типов связей между структурными элементами, обеспечивающими целостность системы. Системный подход предполагает проведение двухаспектного анализа (макро- и микроподходов). При макроанализе система и ее элемент рассматривается как часть системы более высокого порядка. При микроанализе изучается структура объекта, анализируются ее составляющие элементы с точки зрения их функциональных характеристик, проявляющихся через связи с другими элементами и внешней средой.
2. Принцип совместимости, который заключается в обеспечении взаимодействия ИС различных видов, назначений, уровней в процессе функционирования.
3. Принцип стандартизации и унификации, который заключается в необходимости применения типовых, унифицированных и стандартизованных элементов функционирования ИС. Это прежде всего относится к составляющим информационного, технического, программного и других обеспечительных подсистем. Этот принцип позволяет сократить временные, трудовые и стоимостные затраты на создание ИС при максимально возможном использовании накопленного опыта в формировании проектных решений и внедрении автоматизации проектировочных работ, обеспечивает многоаспектную взаимодействие ИС.
4. Принцип декомпозиции, который основан на разделении системы на части и выделении отдельных комплексов работ, создает условия для более эффективного анализа существующего состояния управленческой деятельности, изучение особенностей решения функциональных задач для дальнейшего моделирования конкретных аспектов управленческой деятельности и перевода их на автоматизированную технологию.

Здесь же попробуем декомпозировать разрабатываемую систему, ее можно поделить на несколько модулей:

1. Модуль чтения данных, который необходим для загрузки, предварительной обработки, разметки, а также разделения данных на обучающие и тестовые выборки.
2. Модуль токенизатор, будет переводить текст в токены
3. Модуль обучения нейросети, в которой будут определения архитектуры нейросети и процесса ее обучения.
4. Модуль тестирования, необходимый для получения объективной математической оценки работы обученной модели.

Для обучения и использования модели необходимо вычислительное устройство (как клиентское, так и серверное). С учетом используемых библиотек и технологий, к нему предъявляются следующие требования:

1. ЦПУ с поддержкой пакета инструкций AVX (процессоры Intel®: линейки Core i3, i5, i7 и i9 второго поколения и новее либо процессор линейки Xeon; процессоры AMD®: семейства Jaguar и новее)
2. Графический процессор Nvidia® с поддержкой технологии CUDA (семейства видеокарт RTX, Quadro, TITAN, Tesla)
3. Операционная система: желательно Ubuntu 18.04 и новее (желательно использовать LTS сборки), macOS 10.12 Sierra и новее, Windows 10 и новее
4. Оперативная память 8 ГБ и больше
5. Минимальное место в накопителе данных: 15 ГБ и больше (при условии обучения модели)

# Проектирование платформы для обучения нейросетевой модели

## Используемые модели нейронных сетей

Как было сказано ранее, языковые модели на основе архитектуры Трансформер обучаются с учетом двустороннего контекста, что повышает их чувствительность к многозначным словам. Именно поэтому данные модели будут использоваться в данной работе. Все упомянутые выше модели Трансформер (GPT, BERT, BART, T5 и т. д.) были обучены как языковые модели. Это означает, что они были обучены работе с большими объемами необработанного текста под самоконтролем. Самоконтролируемое обучение — это тип обучения, при котором цель автоматически вычисляется на основе входных данных модели. Это означает, что данные не нуждаются в разметке. Этот тип модели развивает статистическое понимание языка, на котором он был обучен, но он менее полезен для конкретных практических задач. Из-за этого общая предварительно обученная модель затем проходит процесс, так называемого трансферного обучения. В ходе этого процесса модель настраивается контролируемым образом, то есть с использованием размеченных наборов данных, для данной задачи. Пример задачи — угадать следующее слово в предложении, прочитав n предыдущих слов. Это называется каузальным языковым моделированием, потому что результат зависит от прошлых и настоящих входных данных, но не от будущих.

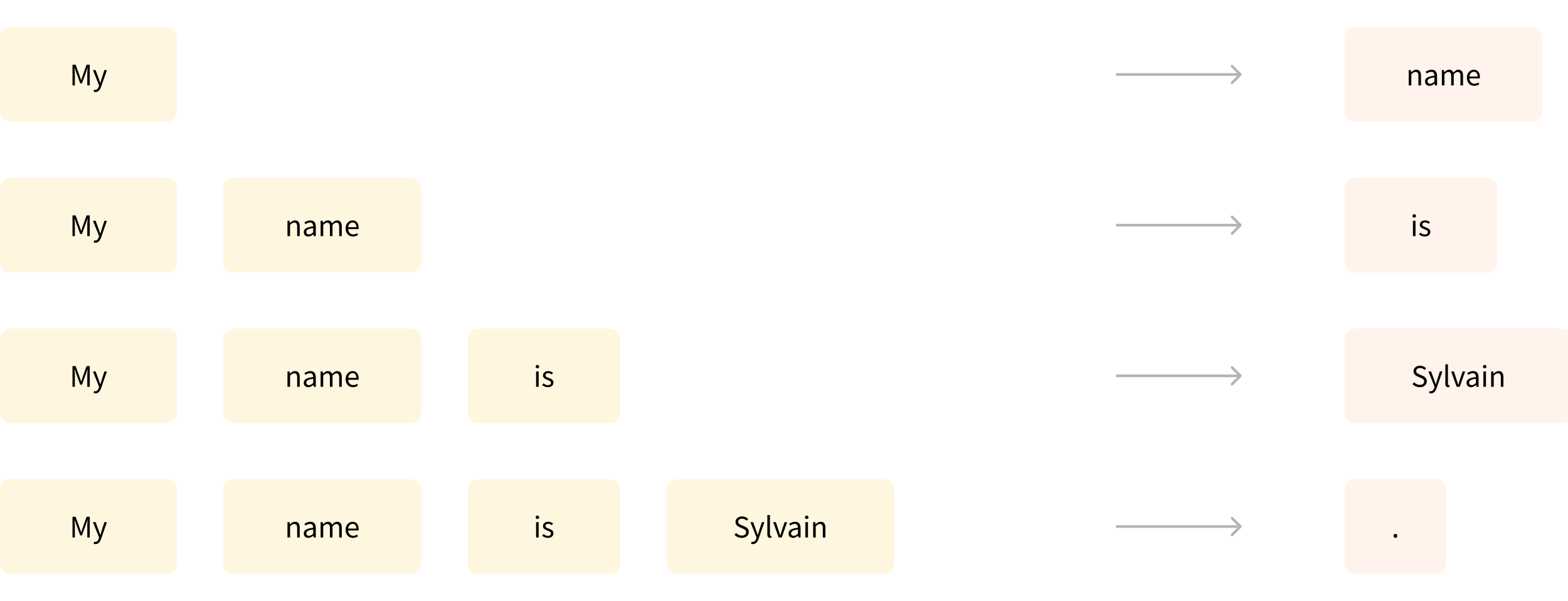


Рисунок 7. Пример предсказания следующего слова

Еще один пример – моделирование замаскированного слова, предсказания замаскированного слова в предложении.

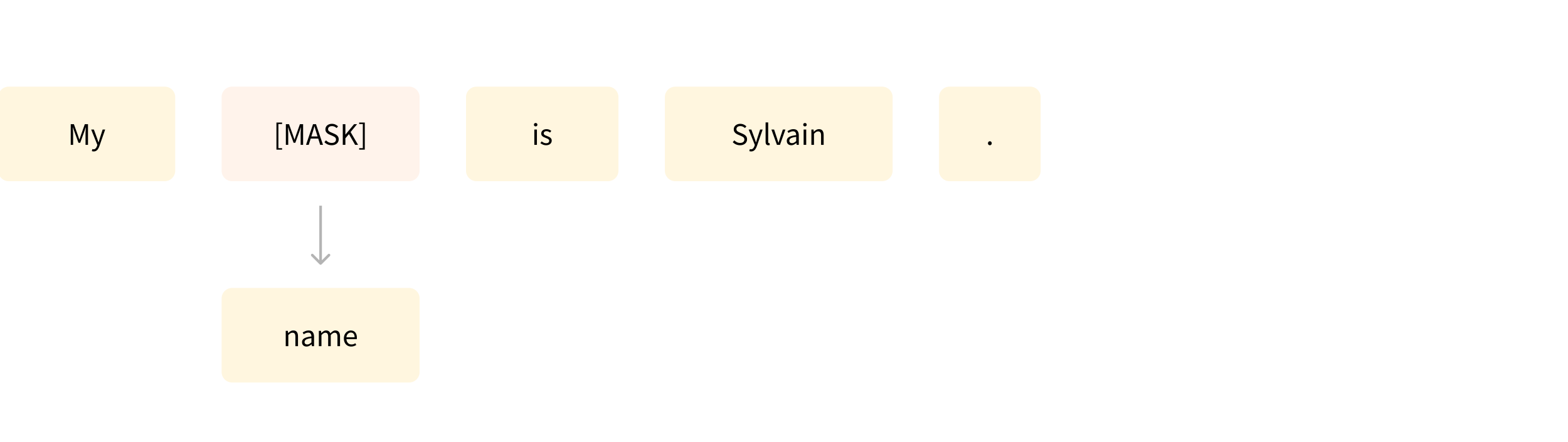


Рисунок 8. Пример предсказания замаскированного слова

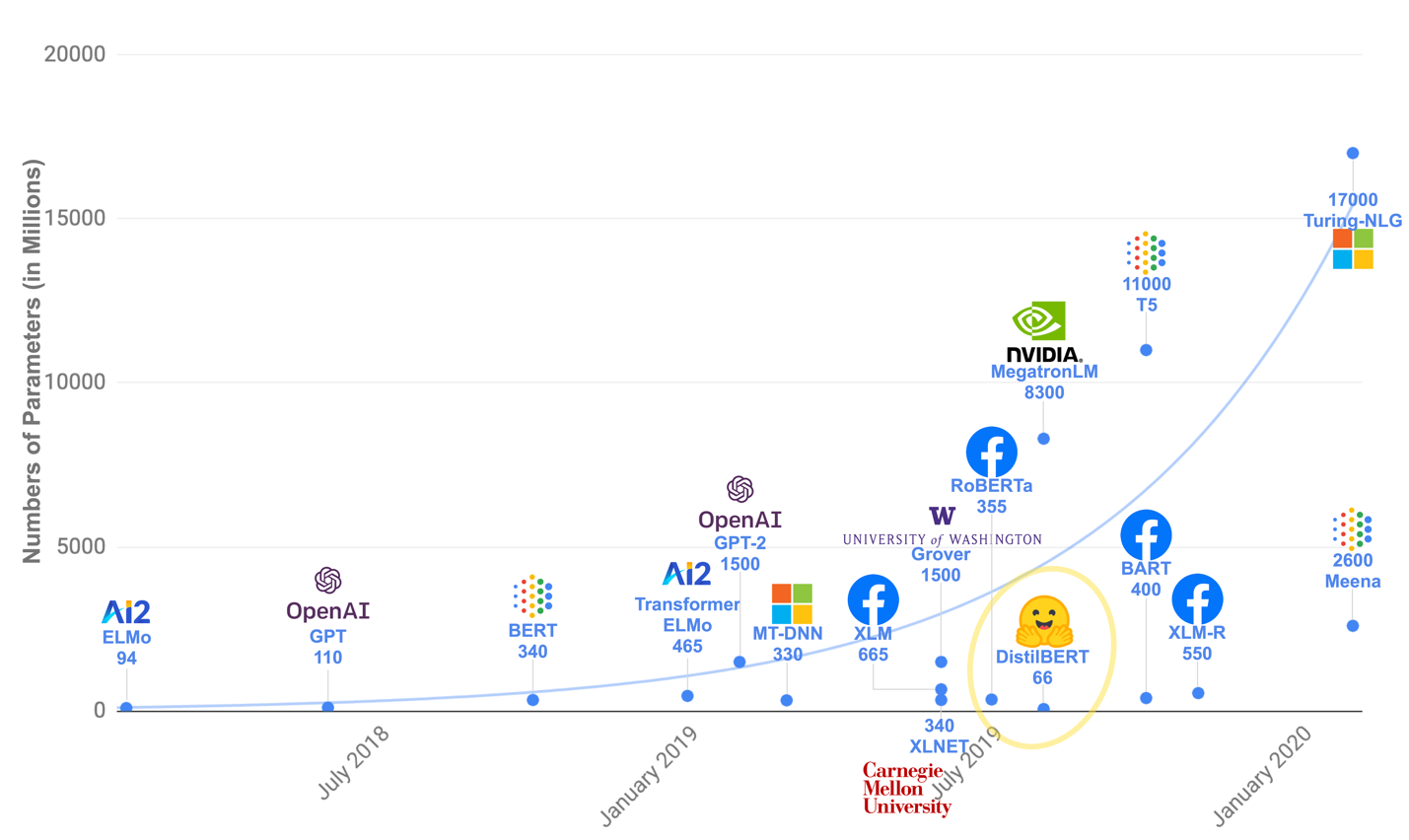
За исключением нескольких экзотических моделей (таких как DistilBERT), общая стратегия повышения производительности обработки заключается в увеличении размеров моделей, а также объема данных, на которых они предварительно обучаются.

Рисунок 9. Сравнение размера моделей по количеству параметров

При обучении модели с нуля, необходимо огромное количество времени, данных, ресурсов, оборудования, для достижения высоких результатов. Для решения данной проблемы архитектура Трансформер реализует механизм так называемого трансферного обучения, то есть для решения задачи машинного обучения можно воспользоваться предобученной моделью представлений и в дальнейшем ее оптимизировать и настроить для новой задачи.

Таким образом решение задачи сводится к двум этапам: предварительному обучению и тонкой настройке. Предварительное обучение — это обучение модели с нуля: веса инициализируются случайным образом, и обучение начинается без каких-либо предварительных знаний. Оно обычно выполняется на очень больших объемах данных. Поэтому для этого требуется очень большой массив данных, а обучение может занять до нескольких недель.

Тонкая настройка, с другой стороны, — это обучение, выполняемое после предварительного обучения модели. Чтобы выполнить точную настройку, необходимо предварительно обученную языковую модель дополнительно обучить с набором данных, специфичным для вашей задачи.

Данный подход дает некоторые преимущества:

1. Предварительно обученная модель уже была обучена на наборе данных, который имеет некоторое сходство с набором данных тонкой настройки. Таким образом, процесс тонкой настройки может использовать знания, полученные исходной моделью во время предварительного обучения (например, в задачах NLP предварительно обученная модель будет иметь какое-то статистическое понимание языка, который вы используете для своей задачи).
2. Поскольку предварительно обученная модель уже была обучена на большом количестве данных, для точной настройки требуется гораздо меньше данных, чтобы получить достойные результаты.
3. По той же причине количество времени и ресурсов, необходимых для получения хороших результатов, намного меньше.

Для точной настройки потребуется лишь ограниченный объем данных: знания, полученные предварительно обученной моделью, «передаются» (transferred), отсюда и термин трансферное обучение.

Рассмотрим детальнее как устроена архитектура Трансформер. Модель в основном состоит из двух блоков:

1. Кодировщик (Encoder): кодировщик получает входные данные и строит их представление (свои функции). Это означает, что модель оптимизирована для получения понимания из входных данных.
2. Декодер (Decoder): декодер использует представление (функции) кодировщика вместе с другими входными данными для создания целевой последовательности. Это означает, что модель оптимизирована для синтеза выходных данных.

Каждая из этих частей может использоваться самостоятельно, в зависимости от поставленной задачи:

1. Модели только для кодировщика: хороши для задач, требующих понимания входных данных, таких как классификация предложений и распознавание именованных объектов.
2. Модели только для декодера: хороши для генеративных задач, таких как генерация текста.
3. Модели кодер-декодер или модели последовательностей: подходят для генеративных задач, требующих ввода, таких как перевод или суммирование.

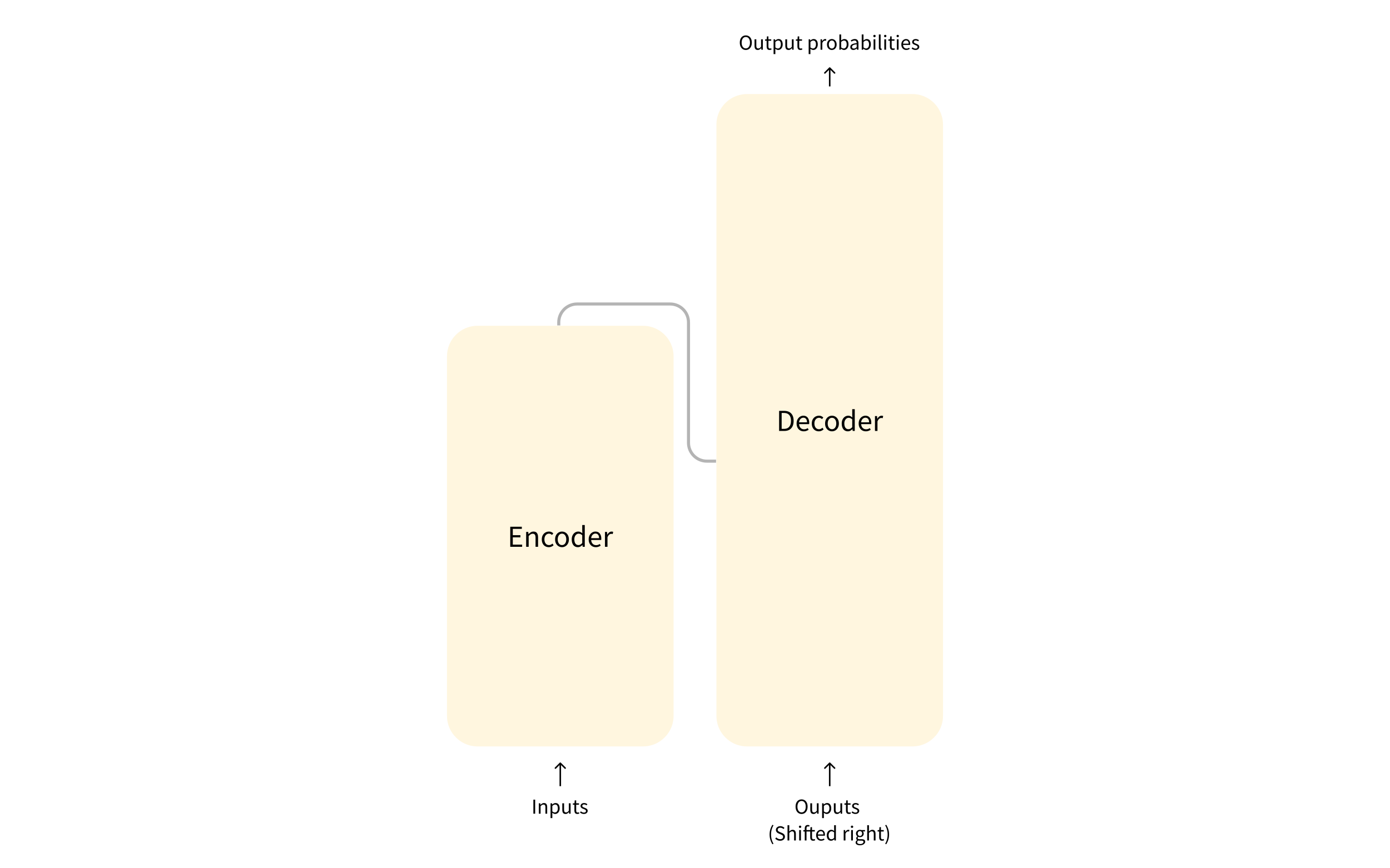


Рисунок 10. Схематическое изображение архитектуры Трансформер

Ключевой особенностью моделей Трансформер является то, что они построены со специальными слоями, называемыми слоями внимания. На самом деле, название статьи, посвященной этой архитектуре, было «Внимание — это все, что Вам нужно» (Attention Is All You Need). Данные слои указывают модели уделять особое внимание определенным словам в переданном вами предложении (и более или менее игнорировать другие) при работе с векторным представлением каждого слова.

Для ясности выше сказанного, рассмотрим задачу перевода текста с английского на французский. Учитывая ввод «You like this book», модель перевода должна будет обратить внимание на первое слово «You», чтобы получить правильный перевод слова «like», потому что во французском глагол «нравится» спрягается по-разному в зависимости от предмета. Остальная часть предложения, однако, бесполезна для перевода этого слова. В том же духе, при переводе «this» модели также необходимо будет обратить внимание на слово «book», потому что «this» переводится по-разному в зависимости от того, является ли связанное существительное мужского или женского рода. Опять же, другие слова в предложении не будут иметь значения для перевода «this». С более сложными предложениями (и более сложными грамматическими правилами) модель должна была бы уделять особое внимание словам, которые могут появиться дальше в предложении, чтобы правильно перевести каждое слово.

Та же концепция применима к любой задаче, связанной с естественным языком: слово само по себе имеет значение, но на это значение сильно влияет контекст, которым может быть любое другое слово (или слова) до или после данного слова.

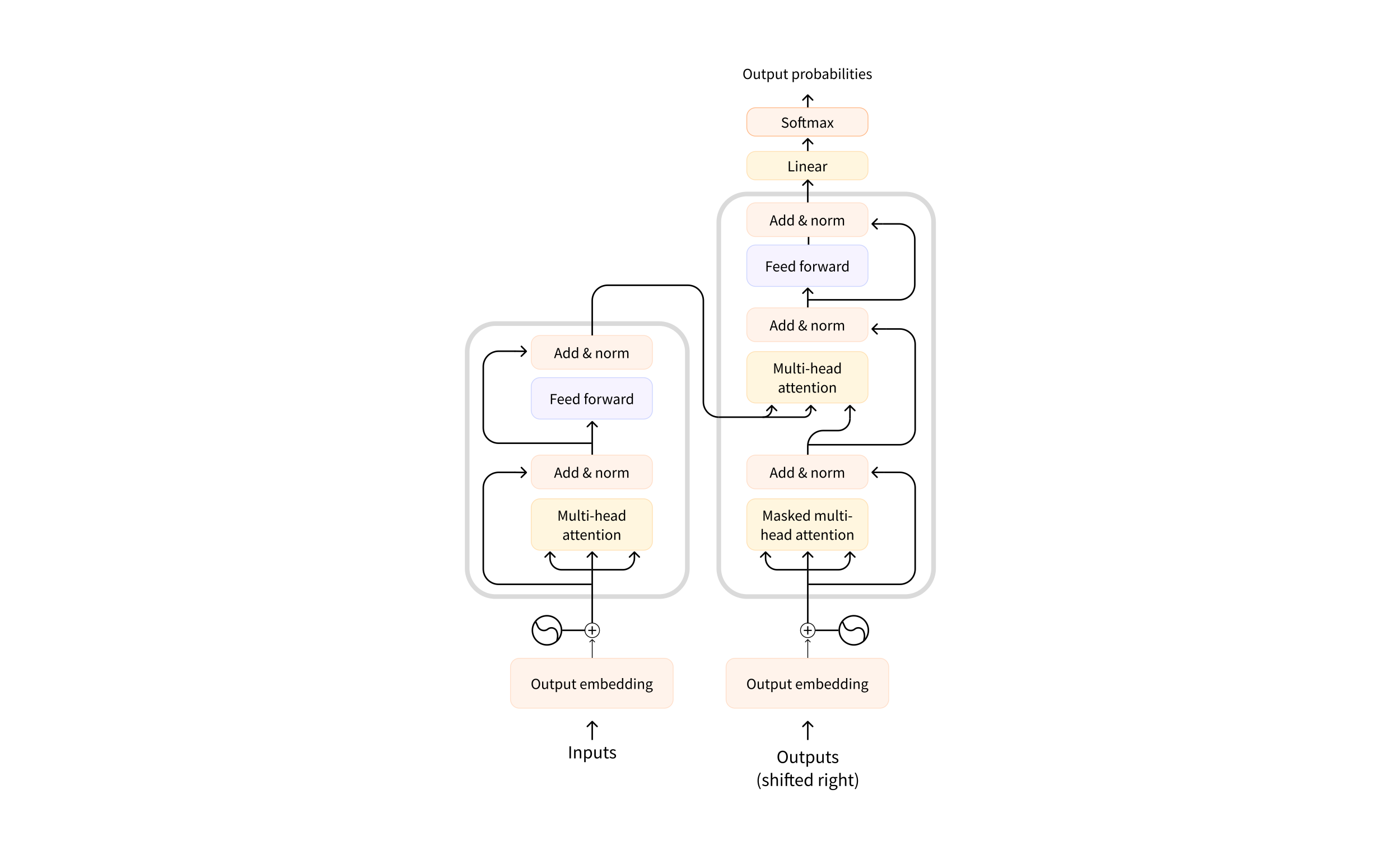


Рисунок 11. Детальная схема архитектуры Трансформер

Архитектура Трансформер изначально была разработана для перевода. Во время обучения кодировщик получает на вход данные (предложения) на определенном языке, а декодер получает те же предложения на желаемом целевом языке. В кодировщике слои внимания могут использовать все слова в предложении (поскольку, как мы только что видели, перевод данного слова может зависеть от того, что в предложении находится после и перед ним). Декодер, однако, работает последовательно и может обращать внимание только на слова в предложении, которые он уже перевел (то есть только на слова перед генерируемым в данный момент, словом). Например, когда мы предсказали первые три слова предложения, мы передаем их декодеру, который затем использует все входные данные кодировщика, чтобы попытаться предсказать четвертое слово. Чтобы ускорить процесс во время обучения (когда модель имеет доступ к целевым предложениям), декодер получает целевое предложение целиком, но ему не разрешается использовать следующие слова. Например, при попытке предсказать четвертое слово слой внимания будет иметь доступ только к словам в позициях с 1 по 3.

## Формализация алгоритмов основного анализа текста

Как и другие нейронные сети, модели Трансформер не могут обрабатывать текст на естественном языке напрямую, поэтому первым шагом в обработке текста является преобразование входных текстовых данных в числа, понятные модели. Для этого используется так называемый токенизатор, который отвечает за:

1. Разделение входных данных на слова, подслова или символы (например, знаки пунктуации), которые называются токенами
2. Сопоставление каждого токена с целым числом
3. Добавление дополнительных входных данных, которые могут быть полезны для модели

Ранее говорилось о существовании нескольких способов токенизации текста, давайте рассмотрим их подробнее.

Первый тип токенизатора, который приходит на ум, основан на разделении текста на слова. Как правило, его очень легко настроить и использовать, используя всего несколько правил, и он часто дает достойные результаты. Например, на изображении ниже цель состоит в том, чтобы разбить необработанный текст на слова и найти числовое представление для каждого из них.

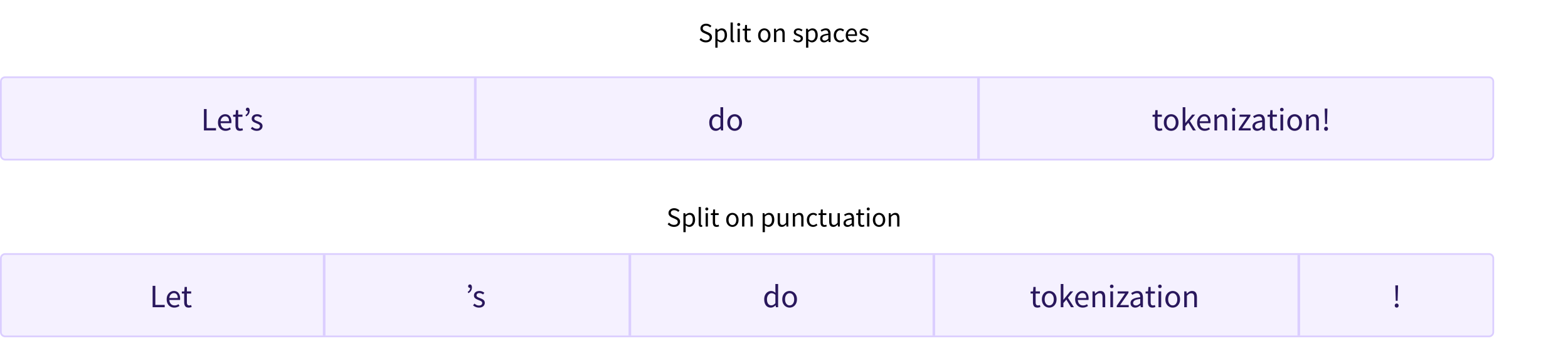


Рисунок 12. Токенизация по словам

Существуют разные способы разделения текста. Например, мы могли бы использовать пробелы, чтобы разбить текст на слова, применив функцию Python split().

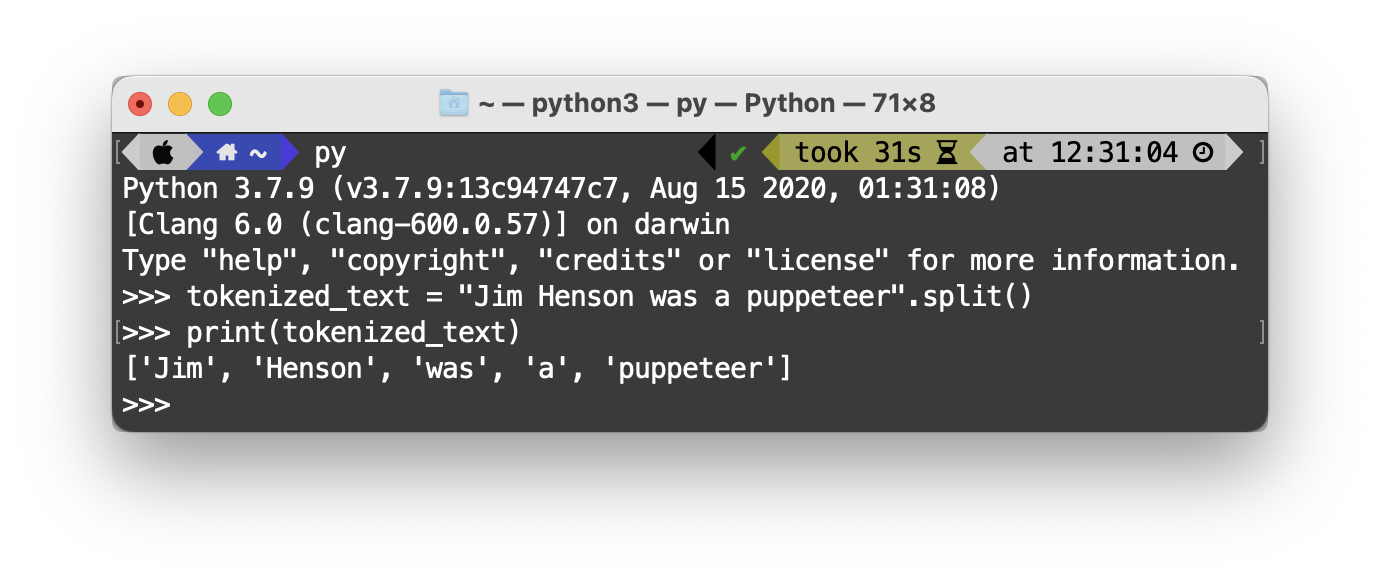


Рисунок 13. Пример использования функции split()

Существуют также варианты токенизаторов слов, которые имеют дополнительные правила пунктуации. С таким токенизатором мы можем получить несколько довольно больших «словарей», где словарь определяется общим количеством независимых токенов, которые есть в нашем корпусе. Каждому слову присваивается идентификатор, начиная с 0 и заканчивая размером словаря. Модель использует эти идентификаторы для идентификации каждого слова. Если мы хотим полностью покрыть язык токенизатором на основе слов, нам понадобится идентификатор для каждого слова в языке, что создаст огромное количество токенов. Например, в английском языке более 500 000 слов, поэтому для построения карты от каждого слова к входному идентификатору нам нужно отслеживать такое количество идентификаторов. Кроме того, такие слова, как «dog» (собака), представлены иначе, чем такие слова, как «dogs» (собаки), и модель изначально не сможет узнать, что «собака» и «собаки» похожи: она идентифицирует эти два слова как несвязанные. Наконец, нам нужен специальный токен для представления слов, которых нет в нашем словаре, часто представленный как «[UNK]» или «». Цель при создании словаря состоит в том, чтобы сделать это таким образом, чтобы токенизатор размечал как можно меньше слов в неизвестный токен. Один из способов уменьшить количество неизвестных токенов — перейти на один уровень глубже, используя токенизатор на основе символов.

Токенизаторы на основе символов разбивают текст на символы, а не на слова. Это дает два основных преимущества:

1. Словарный запас намного меньше.
2. Внесловарных (неизвестных) токенов намного меньше, так как каждое слово может быть построено из конечного числа символов.

Но и здесь возникают некоторые вопросы относительно пробелов и пунктуации.

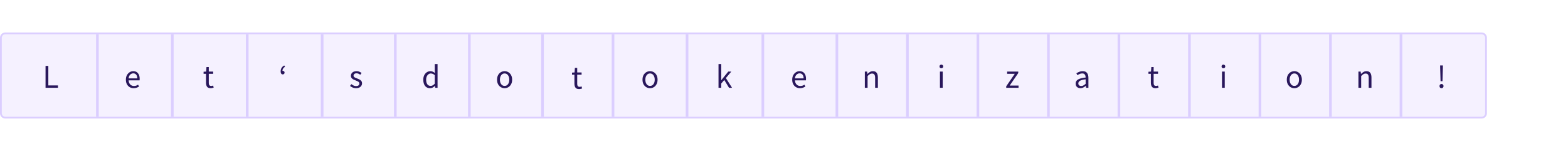


Рисунок 14. Пример токенизации на основе символов

Этот подход тоже не идеален. Поскольку представление теперь основано на символах, а не на словах, можно утверждать, что интуитивно оно менее значимо: каждый символ сам по себе мало что значит, тогда как со словами это не так. Однако это снова зависит от языка; в китайском, например, каждый иероглиф несет больше информации, чем буква в русском языке. Еще одна вещь, которую следует учитывать, это то, что мы получим очень большое количество токенов, которые будут обрабатываться нашей моделью: в то время как слово будет только одним токеном при использовании токенизатора на основе слов, оно может легко превратиться в несколько токенов при преобразовании в символы. Чтобы получить лучшее из обоих способов, мы можем использовать третий метод, который объединяет два подхода: токенизацию по подсловам.

Токенизация по подсловам основана на том принципе, что слова могут быть образованы морфемами, которые встречаются в нескольких словах. Например, на Рисунке 15 фраза «Let’s do tokenization!» разделена на токены по подсловам, и как можно заметить слово «token» отделено от суффикса, так как может участвовать в образовании других слов, так же суффикс «ization» отделен, так как участвует в образовании большого количества слов. Это позволяет нам иметь относительно хороший охват с небольшими словарями и почти полным отсутствием неизвестных токенов.

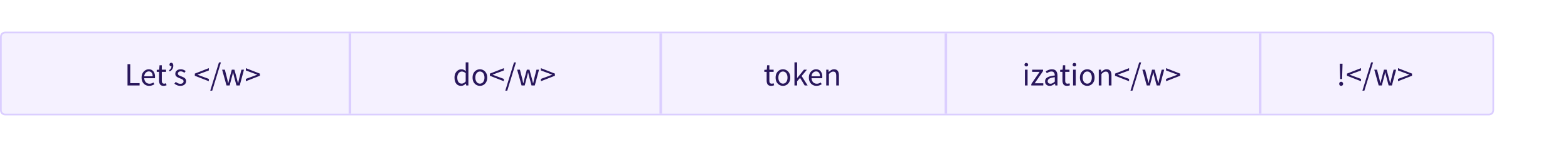


Рисунок 15. Токенизация по подсловам

Этот подход особенно полезен в агглютинативных языках, таких как турецкий или русский, где вы можете образовывать (почти) произвольно длинные сложные слова, соединяя подслова.

Неудивительно, что существует гораздо больше методик токенизации. Вот некоторые из них:

1. BPE на уровне байтов, используется в GPT-2
2. WordPiece, используется в BERT
3. SentencePiece или Unigram, используется в нескольких многоязычных моделях.

## Структура платформы для обучения нейросетевой модели

Итак, ранее были описаны существующие подходы к обработке естественного языка, языковые модели, архитектура Трансформер. В данном разделе коротко опишем структуру платформы, на базе которой будет проходить обучение модели.

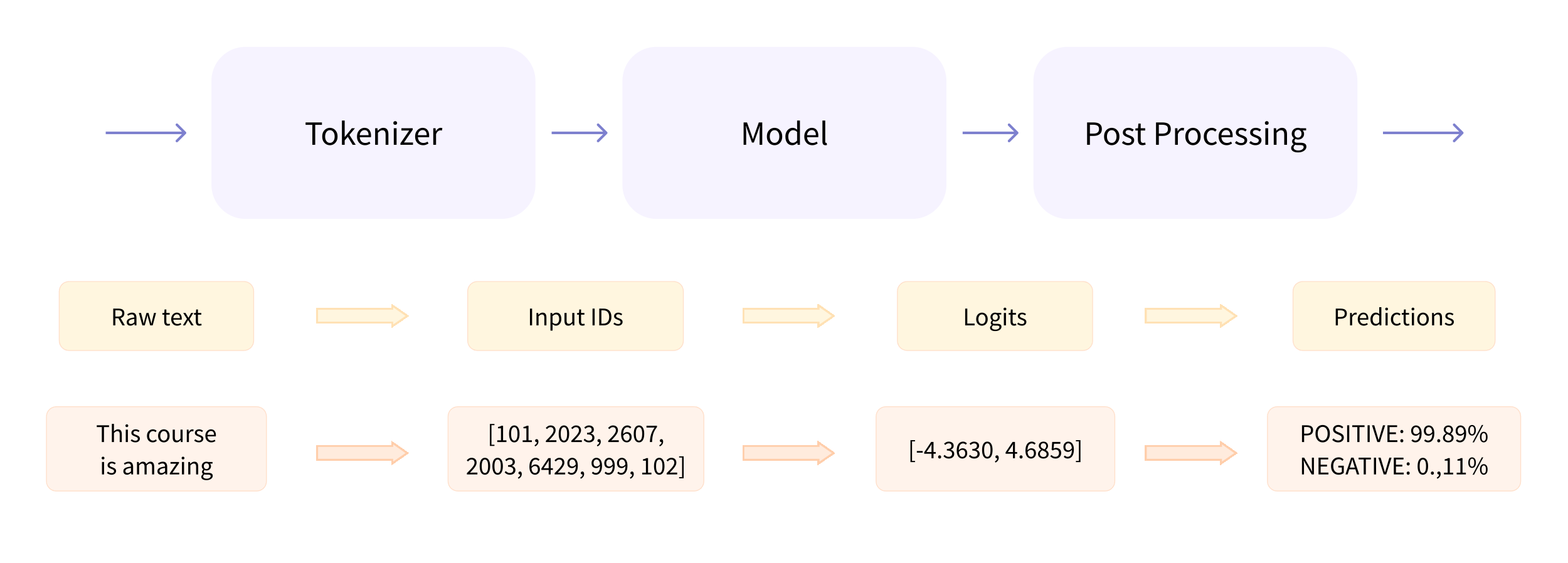


Рисунок 16. Конвейер для обработки естественного языка

В качестве основного языка разработки был выбран Python 3.7 ввиду наличия огромного количества библиотек и фреймворков для машинного обучения, построения нейронных сетей и обработки естественного языка, далее опишем данные библиотеки и фреймворки:

1. PyTorch – библиотека для глубокого машинного обучения. Отличительная черта данной библиотеки поддержка технологии вычислений CUDA, которая позволят ускорить процесс обучения с помощью видеокарт Nvidea.
2. scikit-learn – библиотека, предоставляющая удобные инструменты для машинного обучения и анализа данных.
3. Transformers – библиотека для Python которая предоставляет тысячи предобученных моделей, для решения задач в области обработки естественного языка, изображений, а так же последовательностей, таких как текст, видео и аудио. Предобученные модели представляют из себя обыкновенные модели PyTorch или TensorFlow.
4. Tokenizers – библиотека, включающая в себя реализации самых распространённых на сегодняшний день токенизаторов. Используется в библиотеке Transformers.
5. Pandas – высокоуровневая библиотека для обработки и анализа данных. Имеет тесную интеграцию с другими библиотеками для обработки данных.
6. Matplotlib – продвинутая библиотека визуализации данных. Использовалась для исследования данных и результатов обучения модели.

# Экспериментальная реализация алгоритма анализа текста

## Подготовка данных для обучения

Данные для обучения модели играют важнейшую роль. При этом, как и большинство программ, нейронные сети не воспринимают «сырые» необработанные данные. Для выполнения данной работы, Институтом прикладной семиотики были предоставлены текстовые данные, в основном состоящие из публичных и официальных документов на татарском языке. Данные были в формате Word документов. Для дальнейшей обработки было решено создать скрипт для извлечения текста, листинг кода приводится в Приложении.

Так как исходные данные представляют собой текст на естественном языке, необходимо их разметить, чтобы явно указать нейросети какой результат обработки мы от нее ждем. Так как решение задачи было сведено к Token Classification, необходимо разметить токены, по принадлежности к классам пунктуаций. На рисунке 17, код функции для разметки токенов.

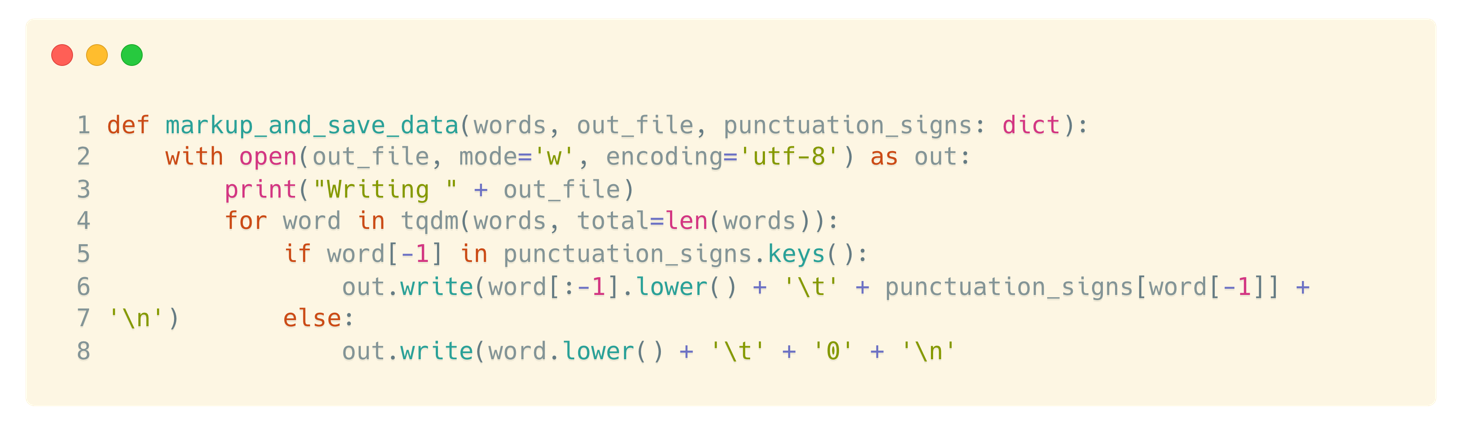


Рисунок 17. Функция для разметки текста

Результат работы данной функции – последовательность слов, которым сопоставляются классы знаков препинания. Для теста текст был размечен по классам 0:пробел, COMMA:запятая и PERIOD:точка.

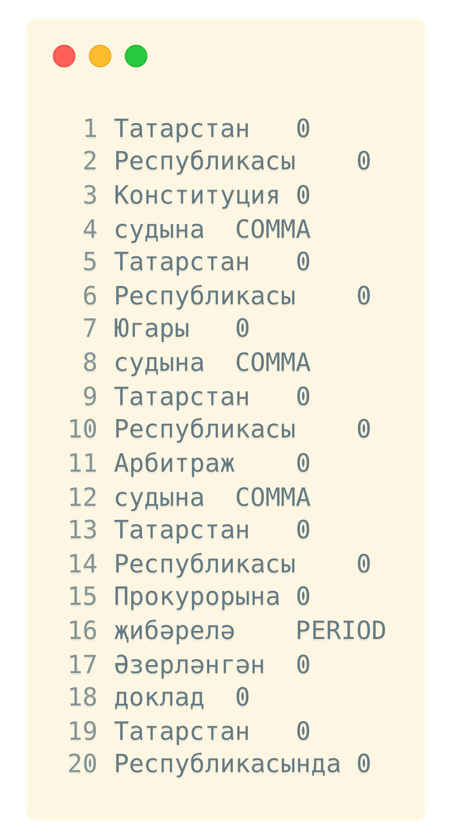


Рисунок 18. Результат разметки

Данные хотя и стали машиночитаемыми, но все же этого недостаточно для использования их в нейросети. В дальнейшем необходимо провести токенизацию и перевод классов в массивы распределения вероятности принадлежности к конкретному классу:

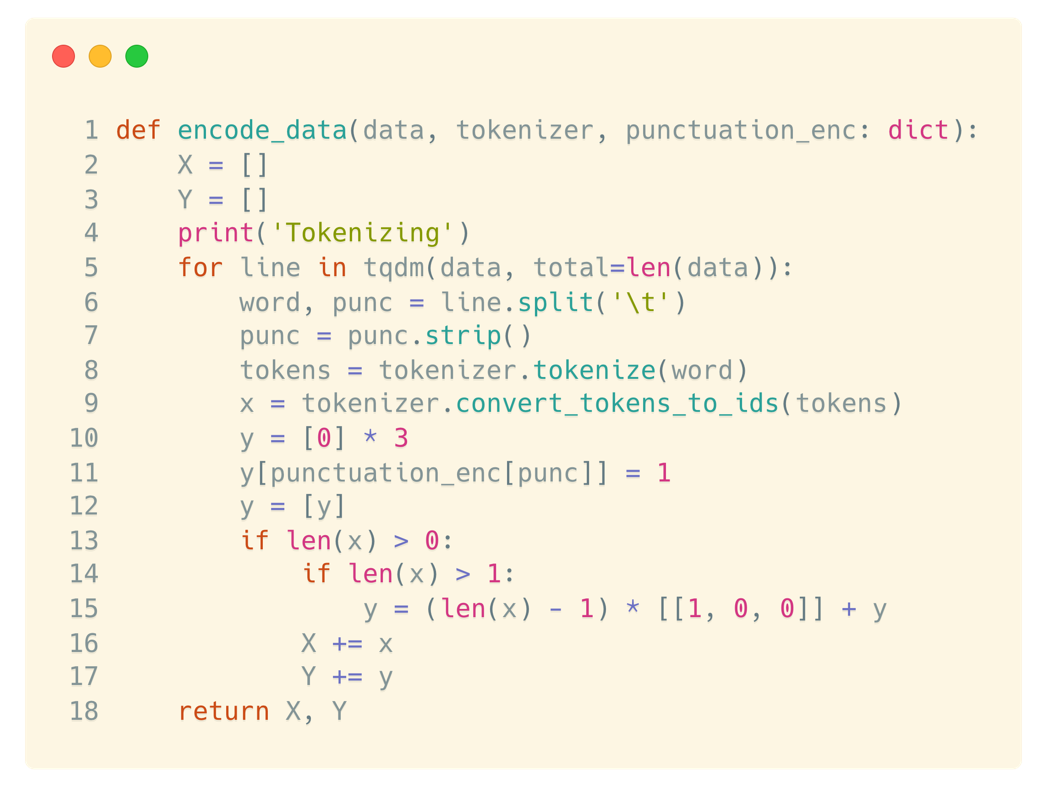


Рисунок 19. Функция для токенизации текстовых данных

В результате токенизации получаются последовательности чисел, символизирующих идентификаторы токенов. BERT по умолчанию использует алгоритм токенизации BPE (Byte pair encoding – от англ. по байтовая кодировка). Данный алгоритм сильно похож на кодирование Хаффмана. С помощью встроенной в библиотеку Tokenizers функцию decode() можно получить из токенов обратно текстовое представление данных.

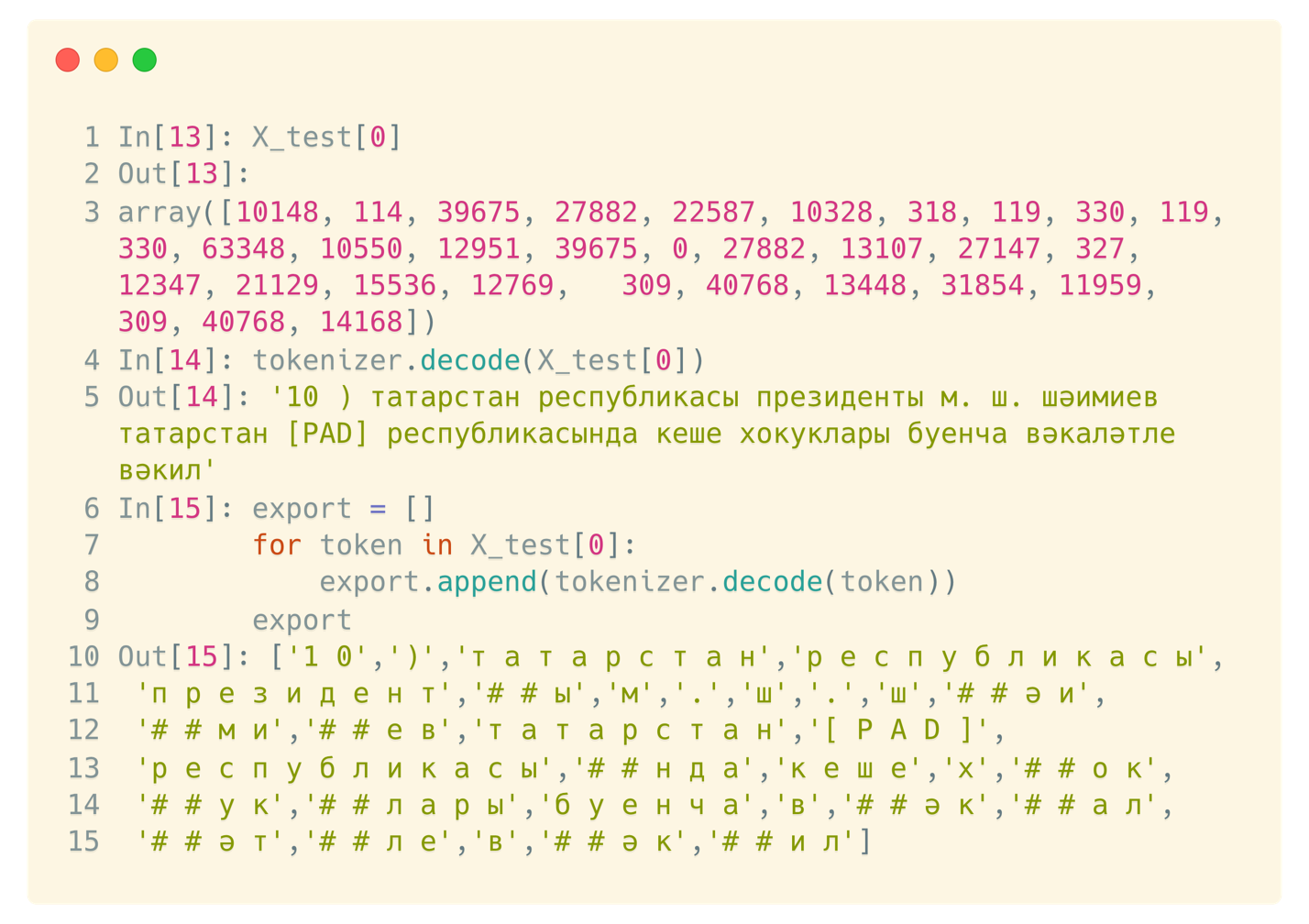


Рисунок 20. Пример токенизации предложения

На данном этапе предварительной обработки данных, данные готовы для подачи на вход в нейросеть.

## Реализация платформы для обучения нейросетевой модели

Как было сказано ранее, в данной работе будет использоваться предобученная модель BERT. Но проблема в том, что BERT обучается для генерации пропущенного токена и синтеза следующего предложения, то есть напрямую нашу задачу он не решает. Для выхода из ситуации было решено отбросить последний слой BERT и выход последнего скрытого слоя передается как вход в добавочный слой, используемый для классификации токенов. Так как в BERT состоит из 12 слоев по 768 ядер Трансформеров, то для модификации модели нужно добавить слой перцептронов со входом в 768 параметров и выходом равным количеству знаков препинания.

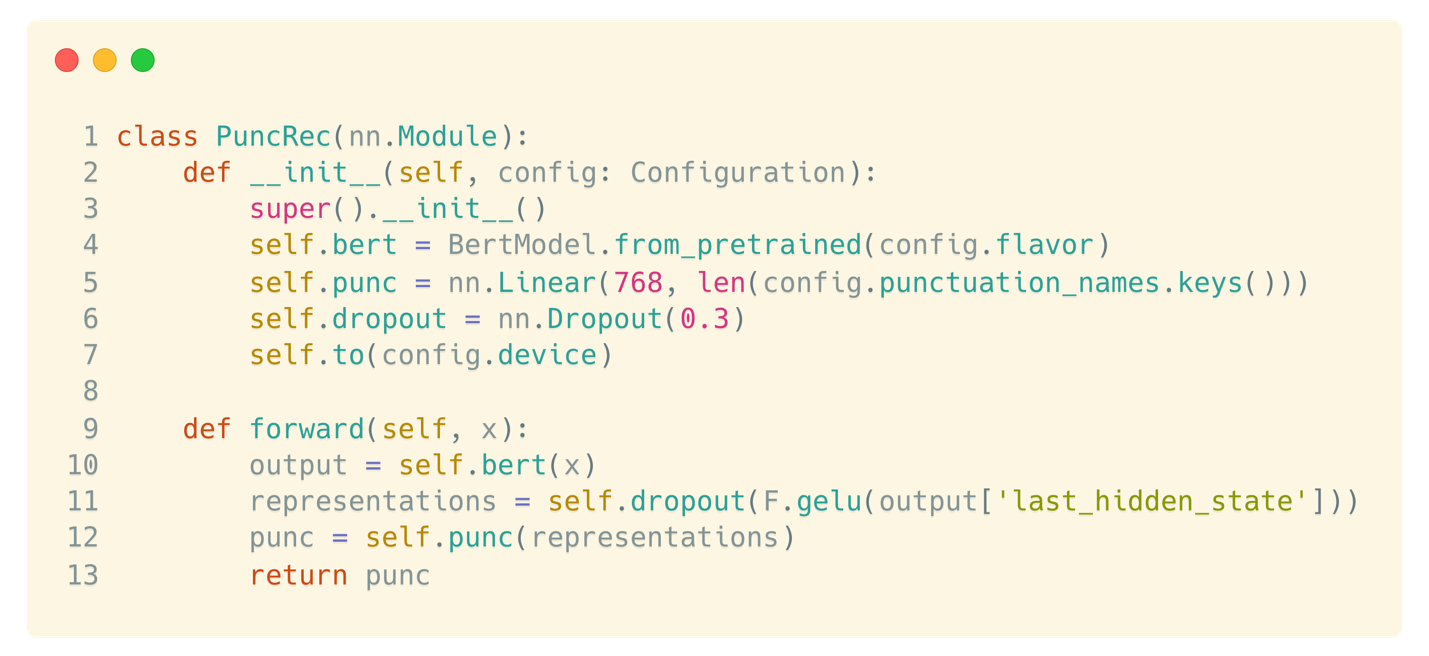


Рисунок 21. Архитектура модели восстановления знаков препинания

Таким образом получится модель, принимающая на вход токены текста, и выдающая на выходе совместное вероятностное распределение центрального токена и классов пунктуации. Для обучения модели используется классический цикл действий: подсчет функции ошибки, вычисление частной производной и обновление параметров модели. При этом каждые несколько таких циклов проводится тестирование модели на наборе данных, не включенном в обучающую выборку, это необходимо для предотвращения переобучения модели и сохранении степени генерализации модели на высоком уровне.

В функцию тестирования встроен функционал сохранения модели, это необходимо для хранения модели с наиболее лучшими показателями на данном этапе обучения, что в свою очередь полезно в случае непредвиденной остановки процесса обучения. В таких случаях можно загрузить обученную модель и продолжить обучение с цикла, когда произошла остановки.

С полным текстом функции обучения и тестирования модели можно ознакомится в приложении.

## Результаты машинного обучения

Итак, можно приступать к обучению модели. Для отслеживания прогресса обучения было настроено логирование, суть которого заключается в записывании в csv-файл метрик, вычисленных в конце каждой итерации.



Рисунок 22. Пример логов полученных при обучении

Ввиду трудности восприятия csv-таблиц была добавлена функция visualize\_results, которая после окончания обучения визуализирует данные, полученные при обучении. Визуализация заключается в отрисовке диаграммы с общей осью абсцисс, где в качестве параметра используются итерации, а по оси ординат располагаются функции ошибки и F-мера. Функция сохраняет диаграмму в виде графического файла в директорию сохранения модели.



Рисунок 23. Визуализация метрик, полученных при обучении

# Тестирование по результатам обучения модели для основного анализа текста

Рыба

## Экспериментальная оценка результатов обучения

Рыба

## Целевой оценочный тест реализации алгоритма

Заключение

Рыба

# Список использованных источников

**Jer Thomas Wolf and Lysandre Debut and Victor Sanh and Julien Chaumond and Clement Delangue and Anthony Moi and Pierric Cistac and Tim Rault and Rémi Louf and Morgan Funtowicz and Joe Davison and Sam Shleifer and Patrick von Platen and Clara Ma and Yacine** Transformers: State-of-the-Art Natural Language Processing [Книга]. - [б.м.] : Association for Computational Linguistics, 2020. - стр. 38-45.

**Mitchell Tom M.** Machine Learning [Книга]. - [б.м.] : McGraw Hill, 1997. - стр. 2.

**Păiş Vasile и Tufis Dan** Capitalization and Punctuation Restoration: a Survey [В Интернете] // ResearchGate. - ResearchGate, 21 ноябрь 2021 г.. - 15 январь 2022 г.. - https://www.researchgate.net/publication/356456267\_Capitalization\_and\_Punctuation\_Restoration\_a\_Survey.

Машинное обучение [В Интернете] // Википедия. Свободная энциклопедия.. - 23 январь 2022 г.. - 13 февраль 2022 г.. - https://ru.wikipedia.org/wiki/Машинное\_обучение.