

АННОТАЦИЯ

Отчёт 60 с., 5 ч., 17 рис., 3 табл., 23 источника, 5 прил.

МАШИННОЕ ОБУЧЕНИЕ, НЕЙРОННЫЕ СЕТИ, ОБУЧЕНИЕ С УЧИТЕЛЕМ, ПОНИМАНИЕ ЕСТЕСТВЕННОГО ЯЗЫКА, КЛАССИФИКАЦИЯ ТЕКСТА, КЛАССИФИКАЦИЯ НАМЕРЕНИЙ, ТРАНСФОРМЕРЫ

В этой работе анализируется эффективность различных моделей архитектуры трансформер в классификации намерений и представлена разработка модуля понимания естественного языка на основе нейронных сетей архитектуры трансформер для системы создания диалоговых моделей. Модуль, разработанный в этом исследовании, сосредоточен на классификации намерений и составляет основную часть модуля понимания естественного языка для системы.

По теме работы было выступление на XVI Всероссийской научной конференции молодых ученых «НАУКА. ТЕХНОЛОГИИ. ИННОВАЦИИ» и была опубликована статья [14].

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1. ЗАДАЧА КЛАССИФИКАЦИИ ТЕКСТОВ	7
1.1. МЕТОДЫ КЛАССИФИКАЦИИ ТЕКСТОВ НА БАЗЕ МАШИННОГО ОБУЧЕНИЯ	7
1.2. НЕЙРОННЫЕ СЕТИ	8
1.3. ВЕКТОРНОЕ ПРЕДСТАВЛЕНИЕ ТЕКСТОВЫХ ДАННЫХ .	10
1.4. КЛАССИФИКАЦИЯ ТЕКСТА	11
1.5. ТРАНСФОРМЕРЫ	12
1.6. МОДЕЛИ	14
1.6.1. BERT	15
1.6.2. RoBERTa	15
1.6.3. DistilBERT	16
1.6.4. ALBERT	16
1.6.5. ELECTRA	17
1.6.6. OPT	17
1.7. МЕТРИКИ	18
2. ОПИСАНИЕ РАЗРАБОТАННЫХ ПРОГРАММ	20
2.1. ПРОГРАММА ДЛЯ ГЕНЕРАЦИИ ДАННЫХ	21
2.2. ПРОГРАММА ДЛЯ ПОИСКА ОПТИМАЛЬНЫХ ПАРАМЕТРОВ	21
2.3. ПРОГРАММА ДЛЯ СРАВНЕНИЯ МОДЕЛЕЙ	21
3. НАБОР ДАННЫХ	23
3.1. ВЫДЕЛЯЕМЫЕ ИНТЕНТЫ	23
3.1.1. Интенты для взаимодействия с игрой	23
3.1.2. Интенты для понимания фраз игрока в диалоге .	24
3.2. ГЕНЕРАЦИЯ ДАННЫХ	25
3.2.1. Метод маскирования	25
3.2.2. Модель перефразирования	26
3.3. ОБРАБОТКА И АНАЛИЗ ПОЛУЧЕННЫХ ДАННЫХ	27

4. ВЫБОР ОПТИМАЛЬНЫХ ПАРАМЕТРОВ ОБУЧЕНИЯ	30
4.1. ТЕХНОЛОГИЯ ИССЛЕДОВАНИЯ	30
4.2. РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ	30
4.2.1. BERT	30
4.2.2. RoBERTa	31
4.2.3. DistilBERT	32
4.2.4. ALBERT	33
4.2.5. ELECTRA	33
4.2.6. OPT	34
5. СРАВНЕНИЕ МОДЕЛЕЙ	35
5.1. ТЕХНОЛОГИЯ ИССЛЕДОВАНИЯ	35
5.2. РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ	35
5.3. ТЕСТИРОВАНИЕ И ПРИМЕРЫ РАБОТЫ ИТОГОВОЙ МОДЕЛИ	39
ЗАКЛЮЧЕНИЕ	41
СПИСОК ЛИТЕРАТУРЫ	42
ПРИЛОЖЕНИЕ А.ФРАГМЕНТ ПРОГРАММЫ ДЛЯ ГЕНЕРАЦИИ ДАННЫХ	44
ПРИЛОЖЕНИЕ Б.ТЕКСТ ПРОГРАММЫ ДЛЯ ПОИСКА ОПТИМАЛЬНЫХ ПАРАМЕТРОВ	49
ПРИЛОЖЕНИЕ В.ТЕКСТ ПРОГРАММЫ ДЛЯ СРАВНЕНИЯ МОДЕЛЕЙ	52
ПРИЛОЖЕНИЕ Г.ГРАФИКИ ПРИ ПОИСКЕ ОПТИМАЛЬНЫХ ПАРАМЕТРОВ	55
ПРИЛОЖЕНИЕ Д.ГРАФИКИ ПРИ ОБУЧЕНИИ МОДЕЛЕЙ ДЛЯ СРАВНЕНИЯ	59

ВВЕДЕНИЕ

Модуль понимания естественного языка (Natural Language Understanding или NLU) является важным компонентом многих приложений обработки естественного языка (Natural Language Processing или NLP), включая чат-ботов и виртуальных помощников. Его основная функция заключается в том, чтобы позволить системе понимать и интерпретировать ввод человеческого языка. Основной частью NLU модуля является модуль классификации намерений (интентов). Его функция заключается в том, чтобы точно определить основное намерение, стоящее за вводом пользователей на естественном языке. В данной работе такой модуль создаётся для системы создания чат-агентов для игр. Его функция в этой системе заключается в понимании цели запроса игрока и формировании более полного и корректного ответа на этот запрос, что позволяет улучшить опыт игрока за счёт более полного взаимодействия с игрой. В качестве основного для всей системы был выбран английский язык, что связано с большим количеством доступных моделей и данных.

Эффективность модуля классификации намерений зависит от двух основных факторов: модель и обучающий набор данных (датасет). Модели имеют различные архитектуру, размер и метод предобучения, что оказывает серьёзное влияние на их способность решать те или иные задачи. Кроме того, разные модели потребляют разное количество ресурсов (видеопамять, время и т.д.) при обучении и использовании. Основным и самым распространённым семейством архитектур в NLP является класс моделей трансформеры. В связи с большим количеством моделей, относящихся к этому семейству, было принято решение выбрать несколько наиболее популярных моделей и провести их сравнение при дообучении и решении задачи выделения намерений.

Основными параметрами датасета, влияющими на качество итоговой системы, являются его объём и разнообразие примеров (т.е. количество покрываемых ситуаций). Помимо всего прочего, для обучения модели выделения намерений, входящей в состав NLU модуля системы создания чат-агентов для игр, требуется специфический набор данных. Главное отличие такого набора в содержании специфических для игровой индустрии классов намерений, предполагающих более полное взаимодействие игрока и неигровых персонажей, из-за чего найти подходящие чистые данные в открытом доступе до-

статочно сложно. В связи с этим, для разработки системы классификации намерений требуется создание нового набора данных.

1. ЗАДАЧА КЛАССИФИКАЦИИ ТЕКСТОВ

1.1. МЕТОДЫ КЛАССИФИКАЦИИ ТЕКСТОВ НА БАЗЕ МАШИННОГО ОБУЧЕНИЯ

Машинное обучение – это набор методов, позволяющих на основе набора данных, называемого тренировочным, строить модели, позволяющие решать те или иные задачи без описания явного алгоритма их решения. Изменение параметров модели под решение конкретной задачи называется обучением. Существует три основных подхода к обучению:

1. Обучение с подкреплением (Reinforcement learning);
2. Обучение без учителя (Unsupervised learning);
3. Обучение с учителем (Supervised learning).

В задачах классификации используется подход «Обучение с учителем», что характеризуется наличием метки о принадлежности одному или нескольким классам для каждого примера в обучающем наборе. Применительно к классификации текстов, первым шагом до обучения модели является преобразование текстового входа в численное представление в виде вектора. Простейшим примером такого алгоритма является метод bag-of-words, в котором вектор представляет частоту каждого слова в заранее определённом словаре. Основными моделями для решения задачи классификации текста являются:

1. Наивный байесовский классификатор (Naive Bayes Classifier), основанный на теореме Байеса;
2. Метод опорных векторов (Support Vector Machine или SVM), заключающийся в поиске гиперплоскости в многомерном пространстве между точками, принадлежащими различным классам и являющимися отображением исходных данных в это пространство;
3. Искусственные нейронные сети (Artificial Neural Networks), основанные на имитации работы биологических нейронных сетей и позволяющие извлекать высокоуровневую информацию из входных данных.

На сегодняшний день наиболее эффективной моделью для решения данной задачи являются искусственные нейронные сети, а конкретно глубокие нейронные сети (Deep Neural Networks).

1.2. НЕЙРОННЫЕ СЕТИ

Искусственная нейронная сеть (чаще называемая просто нейронной сетью) – вычислительная система, основанная на имитации работы биологических нейронных сетей и обычно представляемая в виде графа. Простейшей нейронной сетью является однослойный перцептрон, представляющий собой один слой искусственных нейронов, называемый скрытым. Каждый искусственный нейрон умножает числовые входные данные на свой весовой коэффициент и применяет к результату смещение и активационную функцию. Математически это можно представить следующим образом:

$$y = f(Wx + b), \quad (1.1)$$

где:

- x – входные данные, вектор размерности n ,
- W – матрица весов внутреннего слоя, размерность $n \times m$,
- b – вектор смещений внутреннего слоя, размерность m ,
- y – выходные данные, вектор размерности m ,
- f – активационная функция.

Примерами активационных могут служить следующие функции:

1. Функция ReLU: $f(x) = \max(0, x)$;
2. Гиперболический тангенс: $f(x) = \text{th}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$;
3. Логистическая функция: $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$;
4. Функция GELU: $f(x) = \frac{x[1+\text{erf}(\frac{x}{\sqrt{2}})]}{2}$.

Глубокими нейронными сетями называют искусственные сети, имеющие больше одного скрытого слоя, т.е. выход каждого скрытого слоя подаётся на вход следующему скрытому слою. Примерами таких нейронных сетей могут служить рекуррентные нейронные сети (Recurrent Neural Networks или RNN), многослойные перцептроны (Multilayer Perceptron или MLP) или трансформеры.

Для оценки того, насколько предсказанный выход модели (y_i) отклоняется от требуемого (\hat{y}_i), вводится функция потерь. Таким образом обучение нейронной сети сводится к изменению параметров модели (весов W и смещений b) так, чтобы функция потерь принимала минимальное значение. Изначально параметры инициализируются небольшими случайными числами. В качестве функций потерь могут использоваться следующие функции (n – количество примеров в тренировочном наборе данных):

1. Средняя квадратичная ошибка (Mean Square Error или MSE):

$$L = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2; \quad (1.2)$$

2. Перекрёстная энтропия (Cross-Entropy Loss):

$$L = - \sum_{i=1}^n \hat{y}_i \log (y_i); \quad (1.3)$$

3. Функция потерь NLL (Negative Log-Likelihood):

$$L = - \sum_{i=1}^n (\hat{y}_i \log y_i + (1 - \hat{y}_i) \log(1 - y_i)). \quad (1.4)$$

Для минимизации функции потерь по обучающей выборке требуется вычислить градиент функции потерь по параметрам. Основным методом для вычисления является метод обратного распространения ошибки, позволяющий аналитически вычислить производную функции в точке. Метод основан на правиле вычисления производной сложной функции и автоматическом построении графа вычислений.

Обучение нейронной сети является итеративным процессом, каждая итерация которого является обработкой одного или нескольких примеров из обучающей выборки. Обработка всех примеров из обучающего набора называется эпохой. Каждая итерация состоит из четырёх этапов:

1. Вычисление предсказания модели по входным данным (forward pass);
2. Вычисление функции потерь;
3. Вычисление градиента функции потерь по параметрам в точке (backward pass);
4. Изменение параметров модели в заданном градиентом направлении.

1.3. ВЕКТОРНОЕ ПРЕДСТАВЛЕНИЕ ТЕКСТОВЫХ ДАННЫХ

Основной проблемой в применении нейронных сетей для обработки естественного языка является представление текстовых данных в векторном виде (embeddings). Простейшим подходом для её решения является метод one-hot encoding, который заключается в следующем:

1. Создаётся словарь, в котором содержатся все возможные слова;
2. Каждому слову во входной последовательности ставится в соответствие вектор размерности словаря, где в позиции, соответствующей этому слову в словаре, ставится единица, а во всех остальных ноль.

Главной проблемой такого метода является размер получившегося вектора. Каждое слово кодируется вектором размерности всего словаря, что делает обработку закодированных таким образом данных очень затратной.

Более удачным и широко используемым методом для этого является токенизация. Токенизация представляет собой разбиение входного текста на части (это могут быть символы, слова или части слов), называемыми токенами, и представление текста в виде последовательности номеров токенов в общем словаре. Таким образом, весь входной текст кодируется вектором, длина которого зависит только от длины входной последовательности.

Однако такой способ всё ещё не отражает смысла и связи слов или частей слов между собой, поэтому следующим этапом в кодировании текстовой

информации является обучение нейронной сети на основе входных данных в виде последовательности токенов для решения одной из задач языкового моделирования. Затем можно использовать получаемый из последнего скрытого слоя вектор в качестве закодированной информации высокого уровня и решать другие задачи, такие как классификация текста.

1.4. КЛАССИФИКАЦИЯ ТЕКСТА

Для решения задачи классификации текста необходим классификационный слой, отвечающий за создание выходных данных сети, которые представляют собой распределение вероятностей по классам. Он принимает в качестве входных данных значения, извлечённые предыдущими слоями, и применяет набор весов и смещений для получения набора оценок для каждого класса. При классификации текста задача может быть поставлена как задача с одним возможным классом для каждой последовательности (Single-label classification) и как задача с несколькими возможными классами для каждой последовательности (Multi-label classification).

В случае, если задача поставлена как Single-label classification, оценки, полученные моделью, преобразуются в вероятности с помощью функции softmax, которая гарантирует, что суммы вероятностей для всех классов равны 1. Затем класс с наибольшей вероятностью рассматривается как прогнозируемый класс для входных данных. Функция softmax определяется следующим образом:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}, \quad (1.5)$$

где:

- z – вектор из K действительных чисел,
- K – размерность выхода предыдущего слоя,
- i – индекс, $1 \leq i \leq K$,
- $\sigma(z)_i$ – i -я компонента вектора, полученного применением функции softmax к z .

В случае, если задача поставлена как Multi-label classification, оценки, полученные моделью, преобразуются в вероятности с помощью логистической

функции $\sigma(x)$. В этом случае каждое полученное число в выходном векторе принимает значения от 0 до 1 и интерпретируется как вероятность принадлежности входного текста соответствующему классу. В зависимости от поставленной границы (например, считаем что последовательность принадлежит классу в том случае, если вероятность больше 90%) входному тексту присваиваются соответствующие классы.

Классификация намерений является частным случаем задачи классификации текста. В этом случае задача так же может быть поставлена как задача классификации с одним возможным классом и как задача классификации с несколькими возможными классами. Для того, чтобы выделяемые интенты было проще обрабатывать при использовании, решено поставить задачу как Single-label classification.

1.5. ТРАНСФОРМЕРЫ

Архитектура нейронной сети Transformer — это тип модели глубокого обучения, в которой используются механизмы внимания для обработки последовательных данных переменной длины. Модель состоит из нескольких блоков кодировщика (encoder) и декодировщика (decoder), каждый из которых содержит модуль Multi-Head Attention и сеть прямого распространения. Механизм Multi-Head Attention базируется на механизме Attention. Attention вычисляется по следующей формуле:

$$\text{Attention} = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V, \quad (1.6)$$

где:

- $Q = XW_Q$ – вектор размерности d_k ,
- $K = XW_K$ – вектор размерности d_k ,
- $V = XW_V$ – вектор размерности d_v ,
- X – входной вектор размерности d_{model} ,
- W_Q – параметры модели, матрица размерности $d_{model} \times d_k$,
- W_K – параметры модели, матрица размерности $d_{model} \times d_k$,

- W_V – параметры модели, матрица размерности $d_{model} \times d_v$.

Multi-Head Attention является расширением механизма Attention и вычисляется по формуле

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h)W^O, \quad (1.7)$$

где каждый head_i вычисляется параллельно по следующей формуле:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V), \quad (1.8)$$

где:

- W_i^Q – матрицы параметров модели размерностью $d_{model} \times d_k$,
- W_i^K – матрицы параметров модели размерностью $d_{model} \times d_k$,
- W_i^V – матрицы параметров модели размерностью $d_{model} \times d_v$,
- W^O – матрица параметров модели размерностью $hd_v \times d_{model}$,
- $d_k = d_v = d_{model}/h$,
- d_{model} – размерности внутренних слоёв модели.

Multi-Head Attention позволяет модели взвешивать важность различных частей входной последовательности, в то время как сеть прямой связи применяет к входным данным нелинейные преобразования. Кроме того, архитектура модели позволяет обрабатывать несколько примеров параллельно за счёт упаковки входных векторов X в матрицу размерности $d_b \times d_{model}$, где d_b – количество примеров, обрабатываемых одновременно. Помимо этого, важным элементом является кодирование позиции токенов во входной последовательности, что также учитывается в моделях архитектуры Transformer. Выход кодировщика подается в декодировщик, который генерирует окончательную выходную последовательность на основе взвешенных комбинаций выходов этого кодировщика и собственного предыдущего выхода декодировщика (рисунок 1.1) [5].

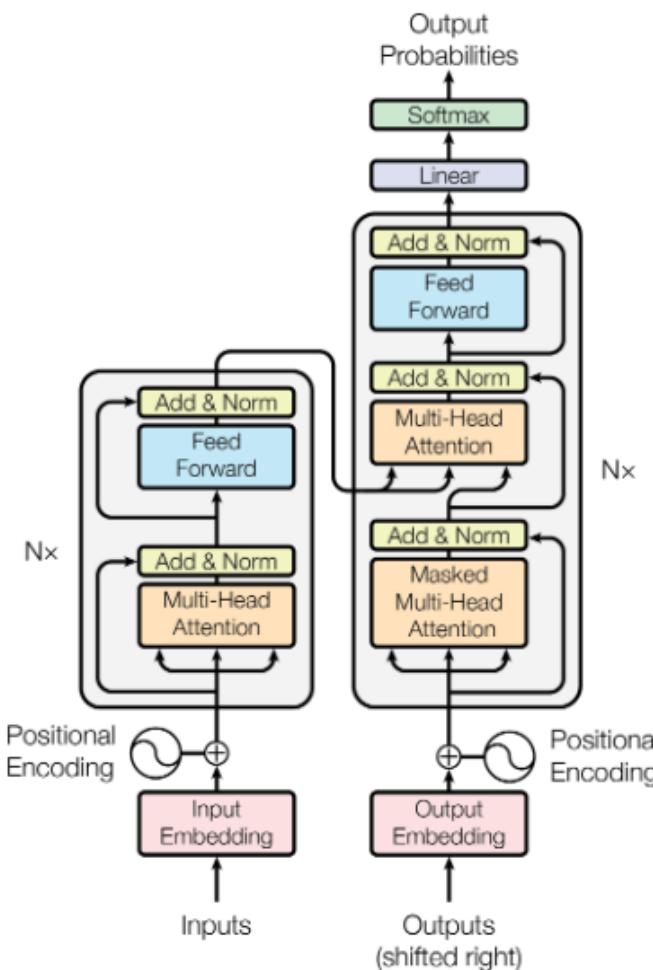


Рисунок 1.1 – Архитектура модели класса Transformer

В контексте классификации текста модели класса Transformer могут использоваться для кодирования входного текста в векторное представление фиксированной длины, которое затем может быть передано в классификатор для прогнозирования метки класса. Обычно это делается с помощью предварительно обученной модели класса Transformer (такой как BERT или GPT) для кодирования текста, а затем добавления классификационного слоя и точной настройки модели на размеченном наборе данных с использованием подхода «обучение с учителем» (Supervised learning).

1.6. МОДЕЛИ

Для исследования были выбраны следующие модели:

- BERT;
- RoBERTa;

- DistilBERT;
- ALBERT;
- ELECTRA;
- OPT.

Это связано с тем, что эти архитектуры являются наиболее популярными при обработке естественного языка, в частности при классификации текстов, что связано с их эффективностью в этих задачах, а также небольшими затратами на обучение моделей данных архитектур. Для каждой архитектуры была взята модель базового размера.

1.6.1. BERT

BERT (Bidirectional Encoder Representations from Transformers) является одной из самых популярных и используемых архитектур класса Transformer. BERT — это encoder-only (т.е. из transformer-блока encoder+decoder остаётся только encoder) двунаправленная модель, предобученная изначально на задачах размаскирования (Masked Language Modeling или MLM) и предсказания следующего предложения (Next Sentence Prediction или NSP). Задача MLM заключается в следующем: 15% токенов в исходной последовательности выбираются для возможной замены, 80% заменяются токеном [MASK], 10% остаются без изменений, оставшиеся 10% заменяются на случайный токен в словаре. В качестве целевого примера используется изначальная последовательность. Задача NSP ставится следующим образом: в качестве входной последовательности подаются два сегмента, разделённые специальным токеном [SEP], а модель предсказывает, являются ли они последовательными сегментами из одного текста или нет. Обе задачи при обучении BERT решаются одновременно [6].

1.6.2. RoBERTa

RoBERTa (Robustly optimized BERT approach) – версия модели BERT, для которой провели дополнительные исследования для улучшения качества итоговой модели. Результатами такого исследования стали следующие решения на этапе обучения модели:

- Шаблон маскирования формируется динамически перед формированием набора примеров для одного шага оптимизации, тогда как в оригинальной модели шаблон маскирования формируется один раз перед обучением;
- Вместо сегментов текста подавались целиком предложения;
- Из итоговой функции потерь исключили добавку для задачи NSP, при этом данные формируются тем же образом (предложения подаются параметрами);
- Увеличили размер набора для одного шага оптимизации;
- Увеличили количество данных для обучения.

Такой подход позволил получить более эффективную модель на выходе [7].

1.6.3. DistilBERT

DistilBERT (Distilled BERT) – версия BERT, к которой применили процесс дистилляции знаний на фазе обучения, что позволило уменьшить размер модели на 40%, сохранив при этом 97% эффективности изначальной модели. Процесс заключается в том, чтобы сделать модель с меньшим количеством весов за счёт уменьшения количества слоёв относительно оригинальной модели в 2 раза, сохраняя ту же архитектуру, и при обучении сделать добавку в функцию ошибки, которая заставляет модель выдавать те же ответы, что и оригинальная модель. Обучалась модель на тех же данных и на той же задаче, что и BERT [8].

1.6.4. ALBERT

ALBERT (A Lite BERT) – ещё одна версия BERT, в архитектуру которой внесли изменения, позволяющие сократить количество параметров модели при сохранении её эффективности, что сокращает затраты на её обучения и использование. Для достижения этого эффекта были внесены следующие архитектурные изменения:

1. Добавлен дополнительный слой перед скрытыми, для того чтобы отделить размер входной последовательности от размера скрытого слоя;

2. В разных слоях используются одни и те же веса, что позволяет уменьшить занимаемую моделью память при незначительном снижении качества модели.

Помимо этого задачу NSP сменили на задачу предсказания порядка предложений (Sentence Order Prediction или SOP), что также улучшило качество итоговой модели. Задача SOP отличается от NSP подбором примеров: если в качестве негативных примеров в NSP брались сегменты из разных текстов, то в SOP в качестве негативных примеров используются последовательные сегменты текста, которые поменяли местами [9].

1.6.5. ELECTRA

ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) – модель, схожая в своей архитектуре с BERT. В отличие от оригинальной модели, обучение ELECTRA происходило следующим образом: есть две модели (Generator и Discriminator), которые обучаются вместе. Generator получает на вход последовательности с маскированными токенами и пытается их восстановить, а Discriminator пытается определить, какие токены в последовательности являются сгенерированными моделью-генератором, а какие изначально находились в последовательности. После обучения Generator не используется, а в Discriminator меняется выходной слой под нужную задачу, после чего дообучается уже на новой задаче. Такой метод обучения позволил уменьшить размеры модели, а соответственно и затраты на её обучение, при этом улучшив её эффективность [10].

1.6.6. OPT

OPT (Open Pre-trained Transformers) - это decoder-only модель, которая по сути является аналогом GPT-3 (Generative Pre-trained Transformers), но отличается от неё меньшими затратами на обучение и тем, что является полностью доступной, в отличие от GPT-3. Модель обучена на большом количестве данных изначально для генерации текста [11].

1.7. МЕТРИКИ

Основными метриками для оценки моделей классификации являются accuracy (точность) и F_1 . Изначально обе метрики вводятся для оценки качества решения задачи бинарной классификации, однако в задаче с большим количеством классов можно также использовать эти метрики: точность в том же виде, а F_1 с некоторыми изменениями.

Accuracy показывает количество правильно данных моделью ответов и высчитывается по следующей формуле:

$$\text{Accuracy} = \frac{K}{N}, \quad (1.9)$$

где:

- K – количество правильно предсказанных примеров,
- N – общее количество примеров в выборке.

Однако такая метрика не является исчерпывающей, так как она не учитывает дисбаланс классов в выборке.

Более строгой и репрезентативной в данном случае является F_1 -метрика. Она учитывает не только количество правильных ответов, но и ошибки первого и второго рода. В задаче бинарной классификации F_1 высчитывается по следующей формуле:

$$F_1 = \frac{TP}{TP + \frac{FP+FN}{2}}, \quad (1.10)$$

где:

- TP – количество правильно предсказанных меток первого класса,
- FP – количество неправильно предсказанных меток первого класса или ошибок первого рода (примеру второго класса присваивается метка первого класса),
- FN – количество неправильно предсказанных меток второго класса или ошибок второго рода (примеру первого класса присваивается метка второго класса).

Для задачи с большим количеством классов F_1 расширяется двумя способами:

1. Macro F_1 ;

2. Micro F_1 .

Macro F_1 высчитывается как среднее F_1 по каждому классу, а в micro F_1 составляющие TP , FP , FN высчитываются как сумма соответствующих величин по каждому классу.

2. ОПИСАНИЕ РАЗРАБОТАННЫХ ПРОГРАММ

В качестве языка программирования для написания программ генерации данных и обучения моделей используется Python. Для генерации данных использовались стандартные средства языка, а для их сохранения использовалась библиотека Pandas. Для использования и обучения моделей, для проведения экспериментов использовались специализированные библиотеки: Transformers, Datasets, WandB, Evaluate.

Transformers используется для загрузки, хранения в облаке, использования и обучения моделей класса Transformer. Библиотека разработана HuggingFace и связана с открытым облачным хранилищем моделей HuggingFace.

Ещё одной библиотекой от HuggingFace является Datasets. Эта библиотека используется для загрузки, хранения в облаке и использования наборов данных и связанных с ними метаданных. Кроме того, она связана с библиотекой Transformers, что позволяет быстро и просто использовать наборы данных из Datasets для обучения моделей с помощью Transformers.

Библиотека Evaluate так же разработана HuggingFace и используется для высчитывания различных метрик при обучении и тестировании моделей.

Библиотека WandB разработана Weights&Biases. Используется для логирования процесса обучения, хранения моделей, наборов данных и визуализации метрик и других системных показателей во время обучения моделей. WandB также тесно связана с библиотекой Transformers. Кроме того, функционал библиотеки WandB позволяет дообучать несколько моделей с различными параметрами последовательно для проведения экспериментов, связанных с различными параметрами обучения моделей.

2.1. ПРОГРАММА ДЛЯ ГЕНЕРАЦИИ ДАННЫХ

В программе для создания набора данных были написана функция для создания примеров методом маскирования, которая принимает на вход список шаблонов, список сущностей для подстановки в шаблоны и маску, вместо

которой подставляются сущности. Также была написана функция для перефразирования примеров с помощью соответствующей модели, которая принимает на вход список примеров для перефразирования. Создание примеров вынесено в отдельные функции для каждого класса, после чего данные сохранялись в соответствующих файлах отдельно. Текст основных функций и пример генерации данных по одному из классов можно найти в приложении А.

2.2. ПРОГРАММА ДЛЯ ПОИСКА ОПТИМАЛЬНЫХ ПАРАМЕТРОВ

Для проведения исследования была разработана программа на языке Python с использованием описанных ранее библиотек. Программа загружает подготовленный набор данных и токенизирует его для дообучения модели. Далее с помощью функций класса из библиотеки WandB sweep обучается несколько моделей с различными параметрами обучения. Данные о процессе обучения и валидации во время обучения загружаются на сайт Weights&Biases, и по ним автоматически строятся графики для анализа. Текст программы для исследования можно найти в приложении Б.

2.3. ПРОГРАММА ДЛЯ СРАВНЕНИЯ МОДЕЛЕЙ

Для сравнения моделей была разработана программа для обучения и тестирования моделей, а также логирования данных во время обучения с использованием описанных ранее библиотек. Программа состоит из 4-х блоков:

1. Загрузка и предподготовка данных для обучения и тестирования;
2. Загрузка модели для обучения и тестирования;
3. Формирование параметров для обучения моделей;
4. Обучение и тестирование.

Данные о процессе обучения и валидации во время обучения загружаются на сайт Weights&Biases и по ним автоматически строятся графики для анализа. Текст программы для исследования можно найти в приложении В.

3. НАБОР ДАННЫХ

3.1. ВЫДЕЛЯЕМЫЕ ИНТЕНТЫ

Прежде чем приступать к созданию набора данных, необходимо определить список намерений (интентов), которые система должна различать. При этом важно учитывать уровень взаимодействия игрока и неигровых персонажей (далее NPC): игрок может ожидать от NPC не только словесного ответа, но и каких-то действий, будь то торговля, обмен, перемещение NPC и т.д. Помимо этого, если игрок хочет просто поговорить, для формирования более корректного ответа важно понимать, что именно игрок говорит NPC: это приветствие, вопрос, касающийся знаний NPC о чём-то, и т.п.

Таким образом интенты можно разделить на две основные группы:

1. Интенты для взаимодействия с игровой механикой;
2. Интенты для более полного понимания диалога игрока с NPC.

3.1.1. Интенты для взаимодействия с игрой

Перед составлением списка интентов, предполагающих взаимодействие с игрой, надо выделить универсальный для наибольшего количества игр набор действий, которые может совершить NPC в ответ на реплику игрока. Проведя некоторое исследование, было решено выделить следующие действия:

- Обмен или торговля (данные действия было решено не разделять, так как из реплики далеко не всегда может быть понятно, что именно хотел игрок);
- Нападение на других NPC;
- Перемещение NPC;
- Защита игрока, места или другого NPC;
- Передача устного сообщения;
- Следование за игроком или другим NPC;

- Стать напарником/помощником игрока;
- Передача предмета другому NPC;
- Выдача задания игроку;
- Завершение выполненного игроком задания;
- Реакция на угрозу от игрока (это может быть нападение NPC на игрока, паническое бегство или что-то ещё).

Каждому действию был присвоен соответствующий интент.

3.1.2. Интенты для понимания фраз игрока в диалоге

Для более корректной генерации ответа на фразу игрока были выделены следующие интенты:

- Приветствие (для ответа на приветствие со стороны игрока приветствием со стороны NPC);
- Вопрос, предполагающий наличие знаний NPC о других NPC или о мире игры в целом (для поиска контекста для модели генерации ответа в общей системе);
- Обычные фразы, не предполагающие наличие специальных знаний (например «Что делаешь?», «Как дела?» и т.п.);
- Шутка (ответом на шутку может быть как смех, так и непонимание со стороны NPC);
- Бессмысленный набор букв или слов (ответ NPC предполагает непонимание фразы);
- Прощание (для ответа на прощание прощанием и передачи информации игре о завершении диалога).

3.2. ГЕНЕРАЦИЯ ДАННЫХ

Для получения основного корпуса данных был использован метод маскирования, часто использующийся для аугментации данных в задаче распознавания имён сущностей (Name Entity Recognition или NER). При использовании такого метода примеры в наборе данных получаются несколько однообразные, поэтому в дополнение к данным, полученным с помощью маскирования, добавляются данные, полученные с помощью модели перефразирования.

Так как метод маскирования позволяет сгенерировать очень большое количество примеров, было принято решение получить как можно больше таких данных, затем перемешать примеры, часть из них перефразировать с помощью модели, затем выбрать из перефразированных и изначальных примеров случайным образом некоторый набор данных в пропорции 1 к 1 (иногда из-за особенностей некоторых интентов пропорции смещались в ту или иную сторону). Таким образом, в итоговом наборе будут как более качественные примеры, полученные методом маскирования, так и менее качественные, полученные с помощью модели перефразирования.

Примеры для классификации бессмысленных запросов генерировались отдельно следующим образом: был создан набор слов и последовательностей символов, а из них случайным образом создавалась последовательность от 1 до 25 слов.

Важно заметить, что примеры для интента «Шутка» были взяты из открытых источников. Это связано с тем, что изменение шутки методом маскирования или перефразированием может исказить её смысл настолько, что шутка перестанет быть шуткой.

Ещё одним интентом, для которого данные не генерировались, является интент, обозначающий фразы игрока, не предполагающие наличие специальных знаний. Примеры для этого класса были взяты также из открытых источников, так как генерация таких примеров сложнее, а в открытых источниках имеются нужные данные.

3.2.1. Метод маскирования

Метод маскирования заключается в следующем:

1. Выбирается несколько примеров для аугментации;
2. Выбранные сущности в них заменяются специальным набором символов, т.е. маской (например, [ITEM]);
3. Выбирается набор сущностей, которые могут находиться в примерах на месте маски;
4. В каждый пример вместо маски подставляются всевозможные варианты сущностей.

Для генерации изначального набора примеров и сущностей была использована инструкционная модель от OpenAssistant. Кроме того, большое количество сущностей было получено переводом простых слов (яблоко, хлеб, меч, брюки, человек, торговец и т.п.) на английский язык.

Важно учитывать тот факт, что если подставлять вместо маски любые сущности или наборы сущностей, можно получить большое количество неадекватных примеров, например, «Я хочу купить 2150 мечей» и т.п.

3.2.2. Модель перефразирования

В настоящее время существует множество сервисов для перефразирования текста, однако все или почти все из них либо являются платными, либо имеют очень ограниченный бесплатный тарифный план, что не позволяет использовать их для перефразирования большого количества примеров. В связи с этим было принято решение использовать предобученную модель с HuggingFace. В качестве такой модели была выбрана модель `chatgpt_paraphraser_on_T5_base`. Это базовая модель T5, дообученная на датасете перефразирования, сгенерированном с помощью ChatGPT. Такой выбор связан с тем, что остальные модели, представленные на HuggingFace для этой задачи, оказались слишком слабыми: либо не меняли предложение, либо слишком сильно искажали изначальный смысл.

После проведения нескольких экспериментов был выбран наиболее удачный набор параметров (таблица 3.1).

Таблица 3.1 – Параметры генерации для модели перефразирования

num_beams	5
num_beams_groups	5
num_return_sentences	5
repetition_penalty	10.0
diversity_penalty	3.0
no_repeat_ngram_size	2
temperature	0.7
max_length	512

3.3. ОБРАБОТКА И АНАЛИЗ ПОЛУЧЕННЫХ ДАННЫХ

В процессе генерации данных было обнаружено, что в некоторых ситуациях модель перефразирования текста может добавлять в примеры лишние символы, слова и целые фразы, что связано с датасетом, на котором она обучена. Например, модель может добавить ещё один или несколько дополнительных знаков препинания, кавычки, какую-то дополнительную информацию в скобках и т.п. Наличие таких ошибок в заметном количестве может оказаться на качестве итоговой модели, поэтому было решено избавиться от них.

Так как данные генерировались по классам, следующий этап предобработки заключается в объединении сгенерированных данных в единый набор и разделение на подвыборки для обучения, валидации и тестирования моделей. Для того, чтобы данные на разных подвыборках не сильно отличались друг от друга и в каждой подвыборке распределение по классам было одинаковое, разделение происходило следующим образом:

1. Данные по каждому классу загружались отдельно и перемешивались;
2. Затем данные по каждому классу разделялись на разные подвыборки (8:1:1 на тренировку, валидацию и тестирование);
3. После данные объединялись по подвыборкам и каждая из них перемешивалась.

После объединения был проведён анализ полученных данных. Общий размер набора данных примерно 2,2 миллиона токенов или 163,2 тысячи

примеров. Средняя длина одного примера 13 токенов. В среднем количество примеров по каждому классу составляет около 10000 примеров (рисунок 3.1). Однако количество примеров по некоторым классам меньше 10000 (от 4000 до 7000), что связано со сложностью создания примеров для этих классов.

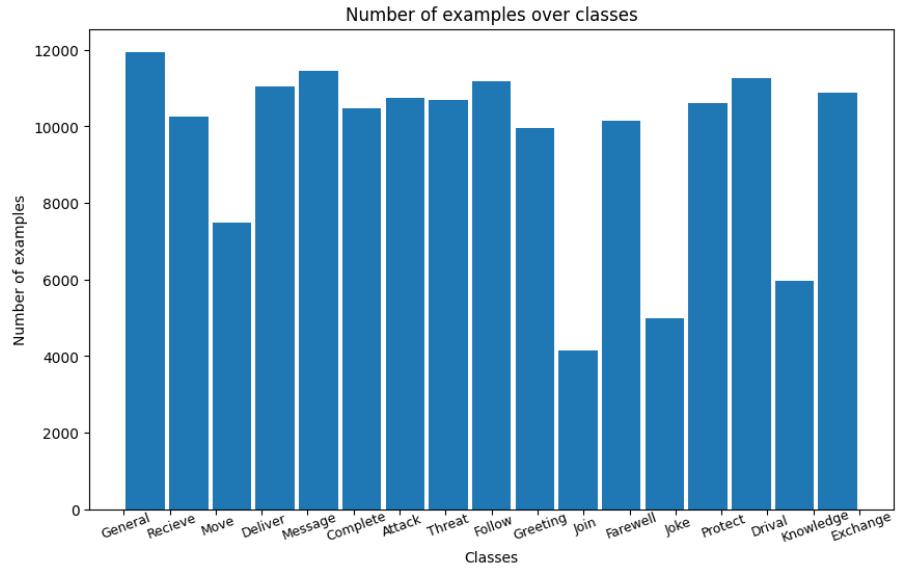


Рисунок 3.1 – Распределение примеров по классам

Кроме того, набор имеет следующее распределение длин примеров в токенах (рисунок 3.2).

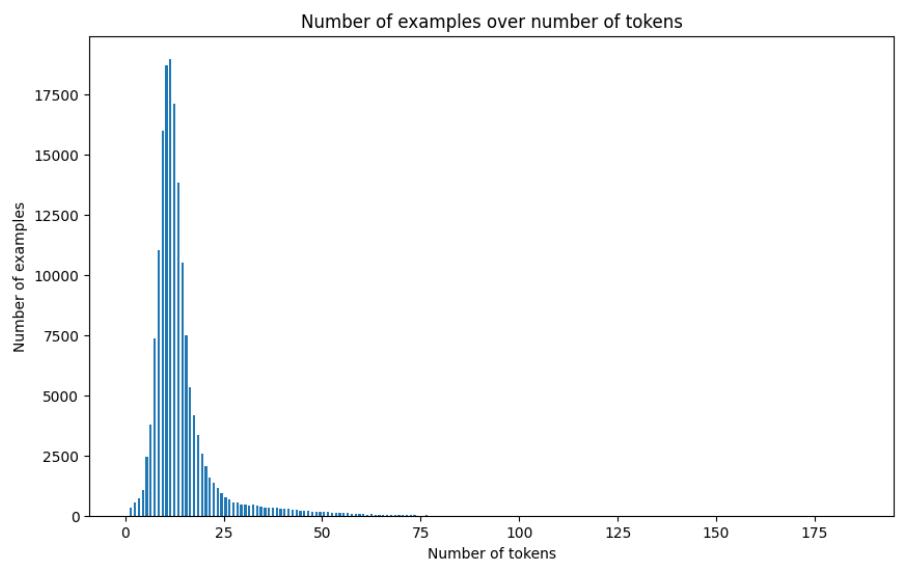


Рисунок 3.2 – Распределение примеров по количеству токенов

Далее приведено несколько примеров из полученного набора данных (таблица 3.2).

Таблица 3.2 – Примеры из полученного набора данных

Примеры	Классы примеров
Your task is to hand saber to Annah.	Deliver
Hello, dear friends!	Greeting
tomorrow one integer hundred claim Follow elves drop man	Drival
Make sure you choose wisely, because every move counts, Beithira Soulforger!	Threat
I'd like to sell tomato, steak.	Exchange
I would like to buy pike and falchion.	Exchange
Could you come after Kieran Fireblood?	Follow
I want you to beat that bandit.	Attack
Are you capable of killing Annah?	Attack
You are responsible for safeguarding an outpost.	Protect
Please, go to this wayfarer.	Move
What can you tell me about Caravan Crossroads?	Knowledge
We will never speak again, Adira Whitesnow.	Farewell
Please take care Brianna Moonlight.	Farewell
Join me.	Join
i don't know.	General
Why does lightning always strike trees? They are the path of leaf resistance.	Joke
Can i find mission for me in Goldgrasp?	Recieve quest
May I ask you to communicate with Hargrimm the Bleak.	Message
I've completed your job in Deeproot Village.	Complete quest

4. ВЫБОР ОПТИМАЛЬНЫХ ПАРАМЕТРОВ ОБУЧЕНИЯ

4.1. ТЕХНОЛОГИЯ ИССЛЕДОВАНИЯ

Чтобы сравнение моделей между собой было более корректным, важно подобрать оптимальные параметры для дообучения каждой модели. Основными параметрами, влияющими на дообучение моделей являются learning rate (температура обучения), learning rate scheduler (изменение learning rate в процессе обучения) и weight decay (сокращение весов или L_2 регуляризация). Каждая модель дообучается несколько раз с различными параметрами, после чего для каждой из них выбирается наиболее оптимальный набор параметров. В качестве основной метрики для выбора лучших параметров дообучения использовалась accuracy (точность), так как в процессе анализа графиков метрик во время обучения было замечено, что micro F_1 и macro F_1 не сильно отличаются от accuracy (не более 1%).

Все модели дообучались на 8 эпохах с ранней остановкой. Условие для ранней остановки: функция ошибки на валидационных данных не уменьшается в течение 3-х эпох.

4.2. РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ

Для анализа и выбора оптимальных параметров будем использовать графики точности на валидационных данных в процессе обучения. На каждом рисунке представлены графики точности на валидационных данных во время обучения по одной модели с разными параметрами. Остальные графики (функция ошибки при обучении, функция ошибки на валидационных данных, micro F_1 , macro F_1) можно найти в приложении Г.

4.2.1. BERT

По графику точности, полученному в результате серии экспериментов (рисунок 4.1), можно сделать вывод, что наиболее оптимальными для дообучения BERT являются следующие параметры:

- learning rate: 1,3e-4;
- learning rate scheduler: constant;
- weight decay: 7e-2.

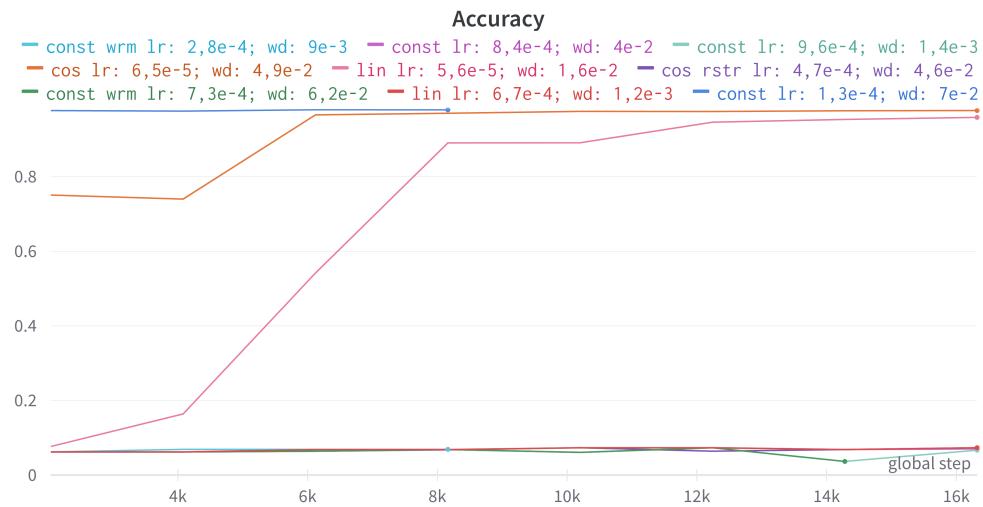


Рисунок 4.1 – График точности во время обучения BERT

4.2.2. RoBERTa

После дообучения серии моделей по графику точности во время обучения (рисунок 4.2) можно сделать вывод, что оптимальными для обучения RoBERTa на текущей задаче являются параметры:

- learning rate: 5e-5;
- learning rate scheduler: linear;
- wight decay: 0.

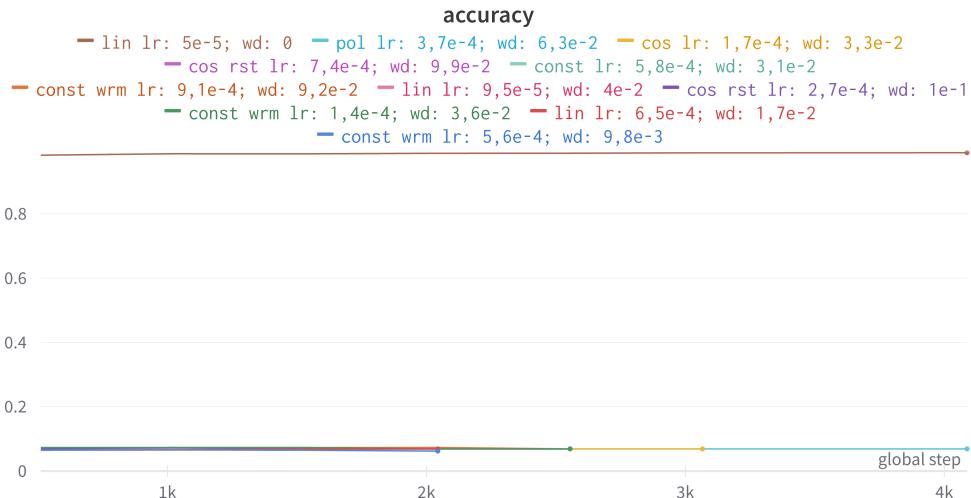


Рисунок 4.2 – График точности во время обучения RoBERTa

4.2.3. DistilBERT

По графику точности во время обучения моделей (рисунок 4.3) об оптимальных параметрах обучения DistilBERT можно сделать следующий вывод:

- learning rate: 1,8e-4;
- learning rate scheduler: linear;
- weight decay: 0.

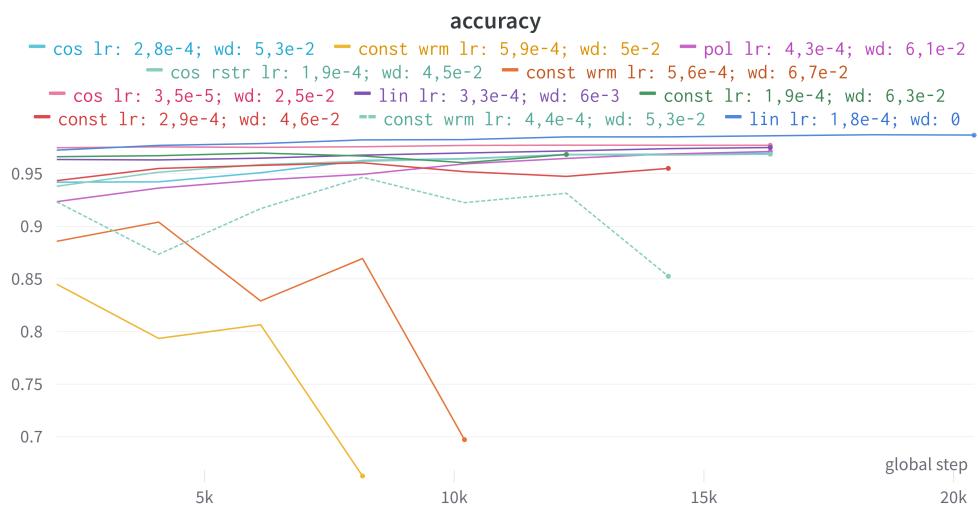


Рисунок 4.3 – График точности во время обучения DistilBERT

4.2.4. ALBERT

Проведя анализ графика точности во время обучения моделей (рисунок 4.4), оптимальные параметры дообучения ALBERT были определены следующим образом:

- learning rate: 5e-5;
- learning rate scheduler: linear;
- weight decay: 0.

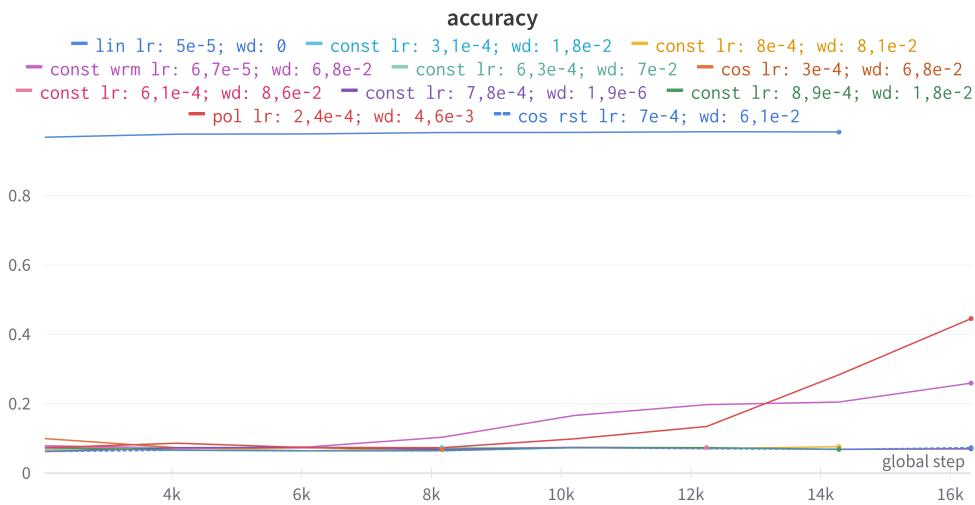


Рисунок 4.4 – График точности во время обучения ALBERT

4.2.5. ELECTRA

После проведения серии экспериментов по графику точности во время обучения моделей (рисунок 4.5) можно выбрать следующие оптимальные для дообучения ELECTRA параметры:

- learning rate: 6,6e-5;
- learning rate scheduler: cosine with restarts;
- weight decay: 4,8e-3.

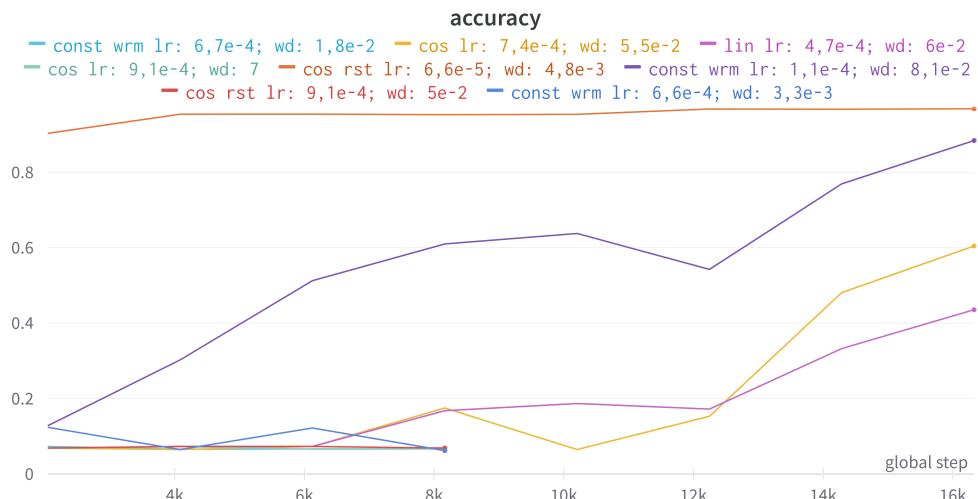


Рисунок 4.5 – График точности во время обучения ELECTRA

4.2.6. OPT

По графику точности во время обучения моделей (рисунок 4.6) об оптимальности параметров дообучения OPT можно сделать следующий вывод:

- learning rate: 5e-5;
- learning rate scheduler: linear;
- weight decay: 0.

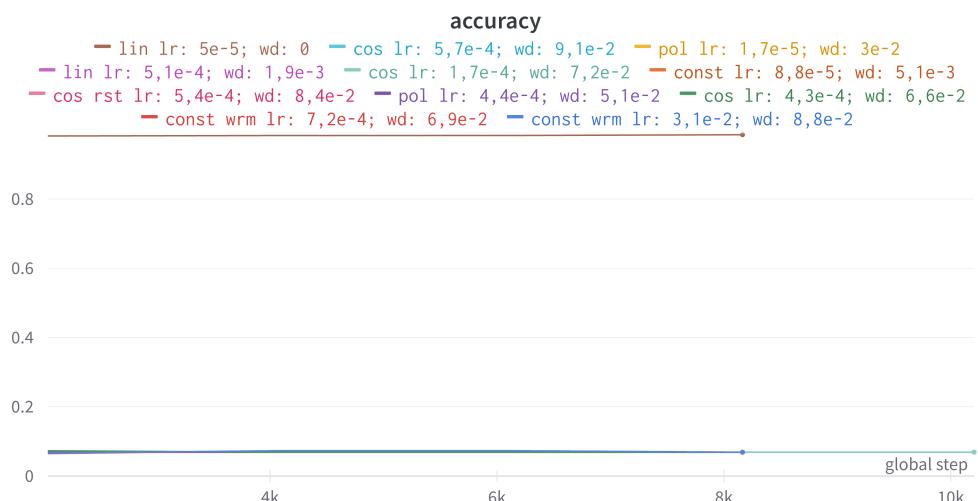


Рисунок 4.6 – График точности во время обучения OPT

5. СРАВНЕНИЕ МОДЕЛЕЙ

5.1. ТЕХНОЛОГИЯ ИССЛЕДОВАНИЯ

На текущем этапе каждая из моделей дообучалась с оптимальными параметрами, а затем тестировалась. В качестве основных метрик для сравнения использовались accuracy, micro F_1 и macro F_1 на валидационных данных во время обучения. Кроме того, для выбора наиболее эффективной модели важно учитывать также данные о времени обучения, количестве занимаемой памяти во время обучения и количестве операций с плавающей точкой, затраченных на обучение моделей, а также скорость во время тестирования.

Все модели дообучались на 8 эпохах с ранней остановкой. Условие для ранней остановки: функция ошибки на валидационных данных не уменьшается в течение 3-х эпох.

5.2. РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ

Проведя анализ графиков метрик во время обучения моделей (рисунок 5.1, рисунок 5.2, рисунок 5.3) можно сделать вывод, что наиболее точной моделью является RoBERTa, тогда как наименее точной моделью оказался BERT. Однако стоит отметить, что точность моделей отличается незначительно (около 1%).

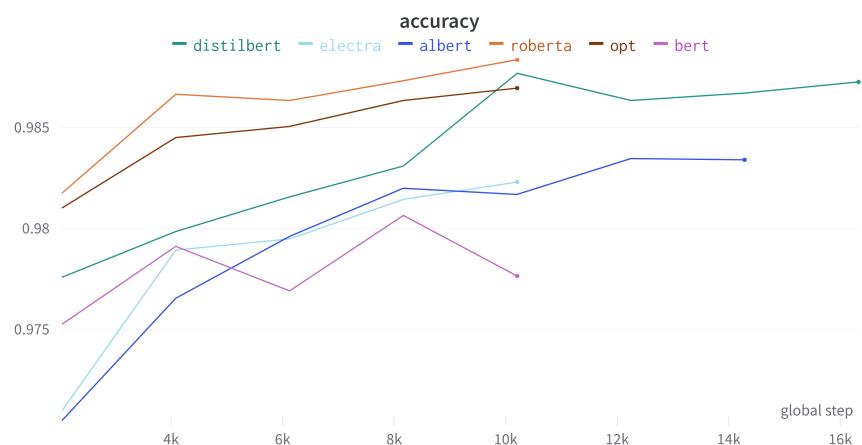


Рисунок 5.1 – График точности во время обучения моделей

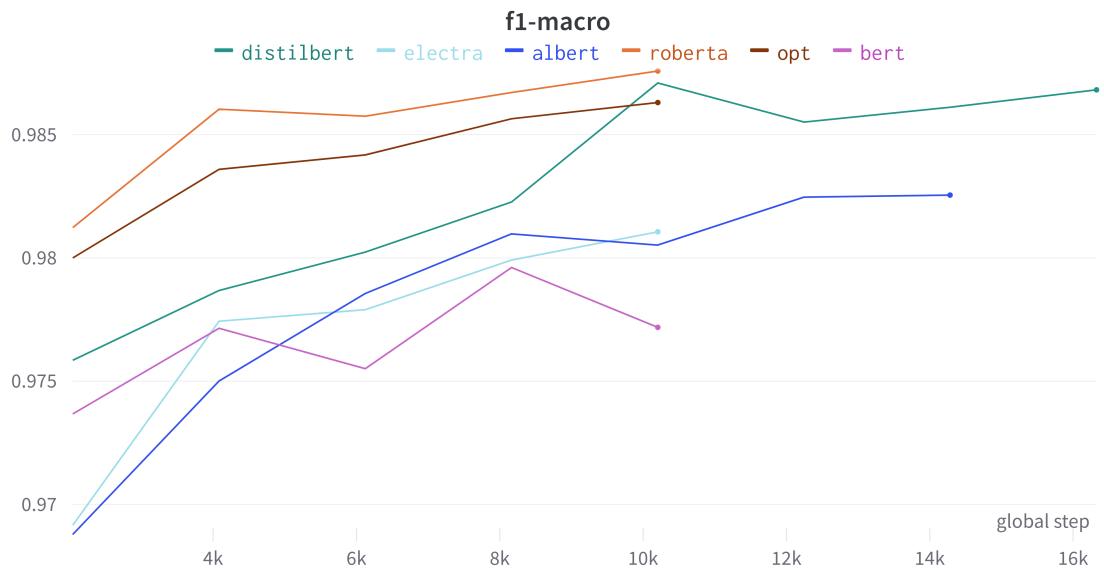


Рисунок 5.2 – График macro F_1 во время обучения моделей

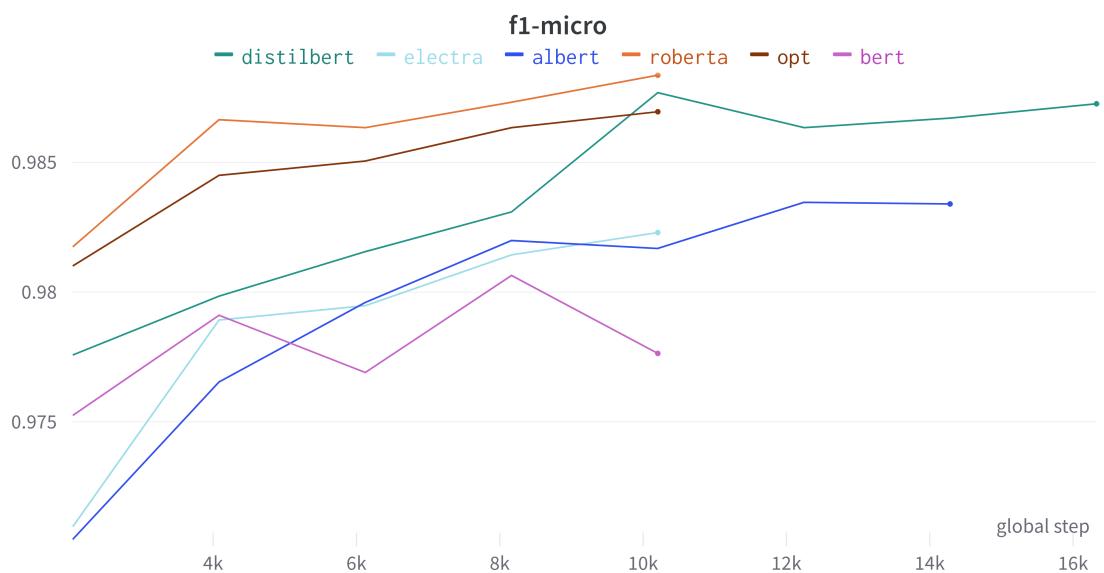


Рисунок 5.3 – График micro F_1 во время обучения моделей

По количеству занимаемой памяти и скорости обучения (рисунок 5.4, рисунок 5.5) RoBERTa также является лучшей моделью. По количеству операций с плавающей точкой, затраченных на обучение (рисунок 5.6), RoBERTa уступает лишь ALBERT и ELECTRA.

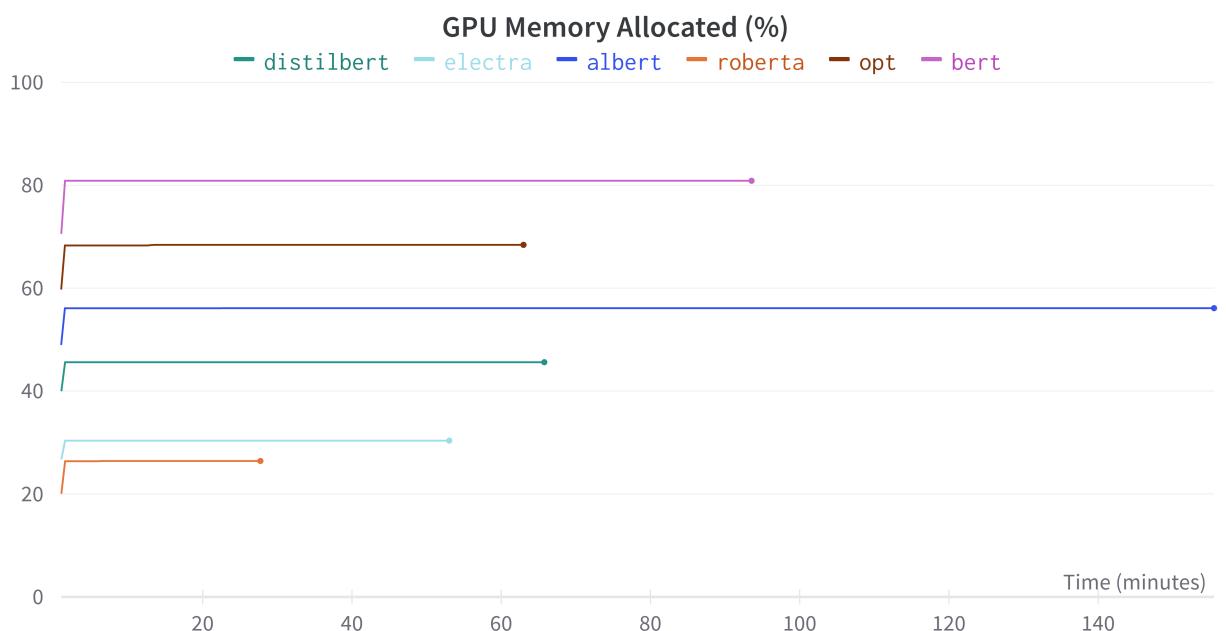


Рисунок 5.4 – Процент видеопамяти, занимаемый моделью при обучении

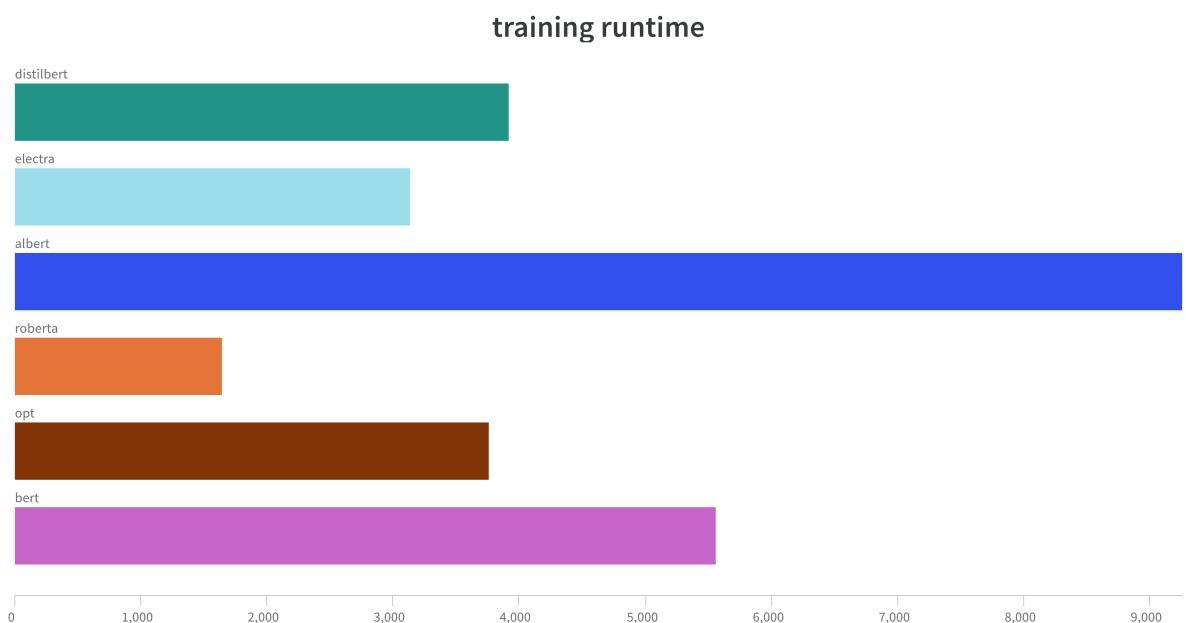


Рисунок 5.5 – Время, затраченное на обучение

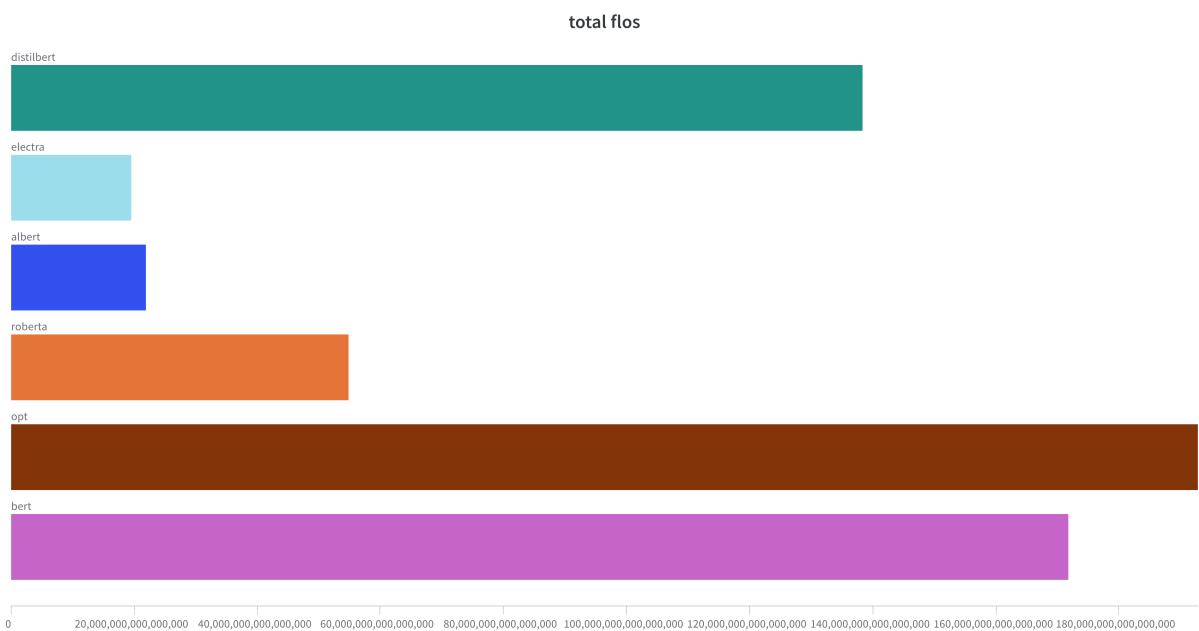


Рисунок 5.6 – Количество операций с плавающей точкой, затраченных на обучение

По скорости при использовании (рисунок 5.7, рисунок 5.8) RoBERTa также является наиболее оптимальной моделью.

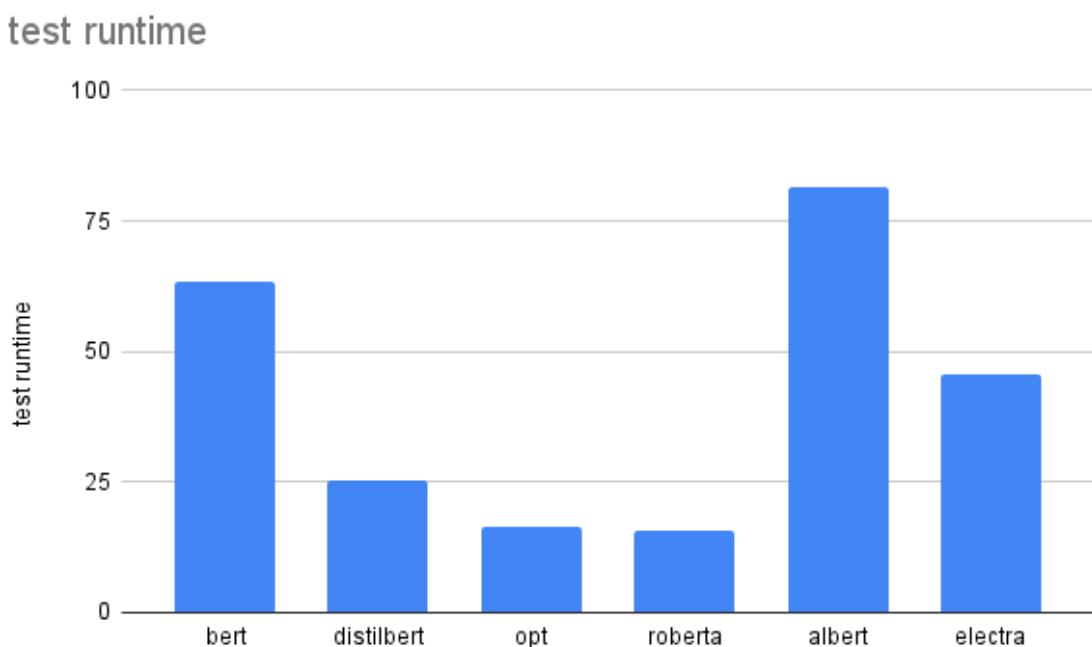


Рисунок 5.7 – Время, затраченное на тестирование

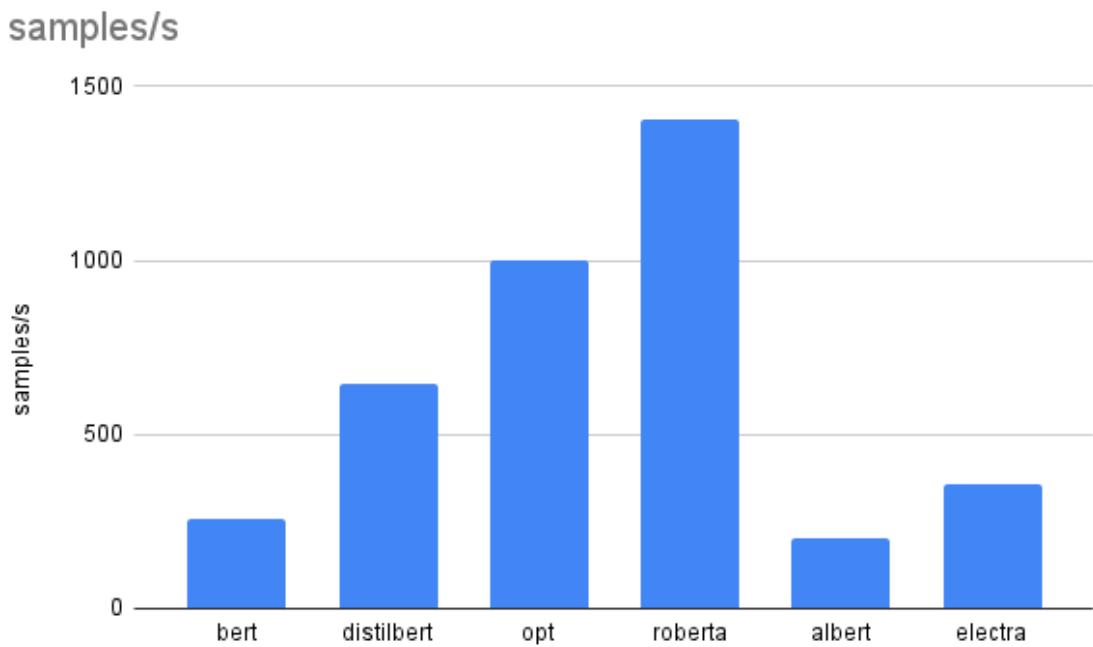


Рисунок 5.8 – Скорость при тестировании

График функции ошибки во время обучения и график функции ошибки на валидационных данных можно найти в приложении Д.

5.3. ТЕСТИРОВАНИЕ И ПРИМЕРЫ РАБОТЫ ИТОГОВОЙ МОДЕЛИ

Проанализировав метрики на тестовых данных (таблица 5.1) и примеры работы итоговой модели выделения интентов (таблица 5.2) можно сделать вывод, что модель неплохо справляется с поставленной задачей, однако наличие неправильно классифицированных интентов может говорить о неполноте данных и сложности в различении некоторых интентов. Например, вопросы «Как дела?» или «Что нового?» в английском языке могут использоваться в качестве приветствия. Отдельной проблемой является распознавание шуток. Это требует дальнейшей работы над обучающим набором данных для улучшения итогового результата.

Таблица 5.1 – Метрики на тестовых данных

Accuracy	98,54%
Micro F_1	98,54%
Macro F_1	98,45%

Таблица 5.2 – Примеры работы итоговой модели

Примеры	Предсказанные классы примеров
Hello, Jack!	Greeting
I want you to attack that people.	Attack
Can i buy something from you?	Exchange
1234	Drival
Is here job for me?	Receive quest
I'm going to hurt you!	Threat
Can you deliver this message to capital?	Message
What can you tell me about this lands?	Knowledge
Goodbye, my friend!	Farewell
fight can low be common	Drival
Follow me.	Follow
How are you today?	Greeting
What are you doing today?	Greeting
How are you today?	Greeting
What are you doing here?	Knowledge
Can you help me?	Join
Why is the chicken crossing the road?	Knowledge

ЗАКЛЮЧЕНИЕ

- Изучены методы генерации и аугментирования данных, методы представления текстовых данных в векторном виде, методы машинного обучения для классификации текста в лице нейронных сетей, архитектуры моделей, наиболее часто используемых для обработки естественного языка и классификации текста в частности, программные средства для обучения и тестирования этих моделей, для логирования данных во время обучения и их анализа;
- Разработаны программы для генерации данных и проведения исследований;
- Создан новый набор данных;
- Проведены исследования для поиска оптимальных параметров для каждой выбранной модели и сравнения моделей между собой;
- Получена основная часть NLU модуля, а именно модель для классификации намерений. Наиболее оптимальной для этой задачи архитектурой является RoBERTa;
- Не самое высокое качество обучающего набора данных, что связано со сложностью выделения некоторых интентов и несовершенством искусственно сгенерированных и аугментированных данных.

СПИСОК ЛИТЕРАТУРЫ

1. *Rish I. et al.* An empirical study of the naive Bayes classifier // IJCAI 2001 workshop on empirical methods in artificial intelligence. – 2001. – Т. 3. – №. 22. – С. 41-46.
2. *Cortes C., Vapnik V.* Support-vector networks // Machine learning. – 1995. – Т. 20. – С. 273-297.
3. *Hendrycks D., Gimpel K.* Gaussian error linear units (gelus) // arXiv preprint arXiv:1606.08415. – 2016 (дата обр. 13.06.2023).
4. *Hecht-Nielsen R.* Theory of the backpropagation neural network // Neural networks for perception. – Academic Press, 1992. – С. 65-93.
5. *Vaswani A. et al.* Attention is all you need // Advances in neural information processing systems. – 2017. – Т. 30.
6. *Devlin J. et al.* Bert: Pre-training of deep bidirectional transformers for language understanding // arXiv preprint arXiv:1810.04805. – 2018 (дата обр. 13.06.2023).
7. *Liu Y. et al.* Roberta: A robustly optimized bert pretraining approach // arXiv preprint arXiv:1907.11692. – 2019 (дата обр. 13.06.2023).
8. *Sanh V. et al.* DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter // arXiv preprint arXiv:1910.01108. – 2019 (дата обр. 13.06.2023).
9. *Lan Z. et al.* Albert: A lite bert for self-supervised learning of language representations // arXiv preprint arXiv:1909.11942. – 2019 (дата обр. 13.06.2023).
10. *Clark K. et al.* Electra: Pre-training text encoders as discriminators rather than generators // arXiv preprint arXiv:2003.10555. – 2020 (дата обр. 13.06.2023).
11. *Zhang S. et al.* Opt: Open pre-trained transformer language models // arXiv preprint arXiv:2205.01068. – 2022 (дата обр. 13.06.2023).

12. *Raffel C. et al.* Exploring the limits of transfer learning with a unified text-to-text transformer // The Journal of Machine Learning Research. – 2020. – Т. 21. – №. 1. – С. 5485-5551.
13. *Koepf A. et al.* OpenAssistant Conversations–Democratizing Large Language Model Alignment // arXiv preprint arXiv:2304.07327. – 2023 (дата обр. 13.06.2023).
14. *Бурлаков В. С., Ишутин М. А., Пятанин А. А.* Диалоговая система для разработчиков видеоигр // Наука. Технологии. Инновации : сб. науч. тр. : в 11 ч., Новосибирск, 05-08 дек. 2022 г. Ч. 2 – Изд-во НГТУ, 2022. – С. 101-104.
15. Документация Python [Электронный ресурс]. URL: <https://docs.python.org/3/> (дата обр. 13.06.2023).
16. Документация Pandas [Электронный ресурс]. URL: <https://pandas.pydata.org/docs/> (дата обр. 13.06.2023).
17. Документация Transformers [Электронный ресурс]. URL: <https://huggingface.co/docs/transformers/index> (дата обр. 13.06.2023).
18. Документация Datasets [Электронный ресурс]. URL: <https://huggingface.co/docs/datasets/index> (дата обр. 13.06.2023).
19. Документация Evaluate [Электронный ресурс]. URL: <https://huggingface.co/docs/evaluate/index> (дата обр. 13.06.2023).
20. Документация Weights&Biases [Электронный ресурс]. URL: <https://docs.wandb.ai/> (дата обр. 13.06.2023).
21. Документация tqdm [Электронный ресурс]. URL: <https://tqdm.github.io/> (дата обр. 13.06.2023).
22. Документация NumPy [Электронный ресурс]. URL: <https://numpy.org/doc/> (дата обр. 13.06.2023).
23. Документация Matplotlib [Электронный ресурс]. URL: <https://matplotlib.org/stable/index.html> (дата обр. 13.06.2023).

ПРИЛОЖЕНИЕ А. ФРАГМЕНТ ПРОГРАММЫ ДЛЯ ГЕНЕРАЦИИ ДАННЫХ

```
1 import pandas as pd
2 import json
3 import random
4 from tqdm import tqdm
5 from datasets import load_dataset
6 from transformers import (
7     AutoTokenizer,
8     AutoModelForSeq2SeqLM
9 )
10 import torch
11 import matplotlib.pyplot as plt
12
13 def augment_templates_with_entities(templates, entities, pattern):
14     new_examples = []
15     for template in tqdm(templates):
16         if pattern in template:
17             for entity in entities:
18                 example = template.replace(pattern, entity, 1)
19                 if pattern in example:
20                     for second_entity in entities:
21                         if second_entity != entity:
22                             second_example = example.replace(pattern,
23                                                 second_entity, 1)
24                             new_examples.append(second_example)
25                         else:
26                             new_examples.append(example)
27                     else:
28                         new_examples.append(template)
29
30     return new_examples
31
32     tokenizer =
33         AutoTokenizer.from_pretrained("humarin/chatgpt_paraphraser_on_T5_base")
34
35     model =
36         AutoModelForSeq2SeqLM.from_pretrained("humarin/chatgpt_paraphraser_on_T5_base")
37
38     def paraphrase(
39         phrase,
```

```

37     num_beams=5,
38     num_beam_groups=5,
39     num_return_sequences=5,
40     repetition_penalty=10.0,
41     diversity_penalty=3.0,
42     no_repeat_ngram_size=2,
43     temperature=0.7,
44     max_length=512
45 ):
46     input_ids = tokenizer(
47         f'paraphrase: {phrase}',
48         return_tensors="pt", padding="longest",
49         max_length=max_length,
50         truncation=True,
51     ).input_ids.to("cuda")
52     outputs = model.generate(
53         input_ids, temperature=temperature,
54         repetition_penalty=repetition_penalty,
55         num_return_sequences=num_return_sequences,
56         no_repeat_ngram_size=no_repeat_ngram_size,
57         num_beams=num_beams, num_beam_groups=num_beam_groups,
58         max_length=max_length, diversity_penalty=diversity_penalty
59     )
60     res = tokenizer.batch_decode(outputs, skip_special_tokens=True)
61
62     return res
63
64
65 def augment_with_paraphraser_for_two_iteration(examples):
66     new_set_of_examples = []
67     new_set_of_examples += examples
68     for example in tqdm(examples):
69         paraphrased_examples = paraphrase(example)
70         for paraphrased_example in paraphrased_examples:
71             if paraphrased_example not in new_set_of_examples:
72                 new_set_of_examples.append(paraphrased_example)
73                 new_paraphrased_examples = paraphrase(paraphrased_example,
74                                                     num_beams=10, num_beam_groups=2, num_return_sequences=10)
75                 for new_paraphrased_example in new_paraphrased_examples:
76                     if new_paraphrased_example not in new_set_of_examples:
77                         new_set_of_examples.append(new_paraphrased_example)
78
79     return new_set_of_examples
80
81 def augment_with_paraphraser_for_one_iteration(examples):
82     new_set_of_examples = []
83     new_set_of_examples += examples
84     for example in tqdm(examples):
85         paraphrased_examples = paraphrase(example)
86         for paraphrased_example in paraphrased_examples:

```

```

82         if paraphrased_example not in new_set_of_examples:
83             new_set_of_examples.append(paraphrased_example)
84     return new_set_of_examples
85
86 action_on_someone_templates = [
87     "I want you to [ACTION] [CREATURE].",
88     "Could you [ACTION] [CREATURE]?",
89     "Can you [ACTION] [CREATURE]?",
90     "Your job is to [ACTION] [CREATURE].",
91     "It's time for you to [ACTION] [CREATURE].",
92     "Your business is to [ACTION] [CREATURE].",
93     "Your task is to [ACTION] [CREATURE].",
94     "Your mission is to [ACTION] [CREATURE].",
95     "Your target is to [ACTION] [CREATURE]."
96 ]
97
98 action_to_someone_templates = [
99     "I want you to [ACTION] to [DESTINATION].",
100    "Could you [ACTION] to [DESTINATION]?",
101    "Can you [ACTION] to [DESTINATION]?",
102    "Your job is to [ACTION] to [DESTINATION].",
103    "It's time for you to [ACTION] to [DESTINATION].",
104    "Your business is to [ACTION] to [DESTINATION].",
105    "Your task is to [ACTION] to [DESTINATION].",
106    "Your mission is to [ACTION] to [DESTINATION].",
107    "Your target is to [ACTION] to [DESTINATION]."
108 ]
109
110 def generate_attack_examples():
111     attacking_verbs = [
112         "attack",
113         "kill",
114         "beat",
115         "eliminate",
116         "exterminate",
117         "liquidate",
118         "defeat",
119         "hit",
120         "stab",
121         "slay",
122         "assasinate",
123         "murder"
124     ]
125
126     attacking_templates =
127         augment_templates_with_entities(action_on_someone_templates,
128                                         attacking_verbs, "[ACTION]")

```

```
128     creatures_for_attacking = [
129         "this guy",
130         "someone",
131         "somebody",
132         "that bandit",
133         "that bandits",
134         "this bandit",
135         "those bandits",
136         "villager",
137         "villagers",
138         "wolves",
139         "pack of wolves",
140         "giant",
141         "that giant",
142         "troll",
143         "those trolls",
144         "that trolls",
145         "this orc",
146         "that orcs",
147         "those orcs",
148         "that elf",
149         "this elf",
150         "those elves",
151         "that elves",
152         "this gnome",
153         "that gnome",
154         "this dwarf",
155         "that dwarf",
156         "those dwarves",
157         "that dwarves",
158         "Barry Ice Spike",
159         "Dak'kon",
160         "Hargrimm the Bleak",
161         "Ku'atraa",
162         "Annah",
163         "Oringratum Battleborn",
164         "Zoltar Swiftrunner",
165         "Ivis Ironguard",
166         "Gontas Firehand",
167         "Delina Windrider",
168         "Thorn Grimson",
169         "Lyda Leafweaver",
170         "barbarians",
171         "wizard",
172         "warrior",
173         "hunters",
174         "Princess Eliconara of Aelion",
175         "Alaric Wisehammer",
```

```

176     "Melodius Chordstryke",
177     "Zephyr Windwhisperer",
178 ]
179
180 attacking_examples =
181     augment_templates_with_entities(attacking_templates,creatures_for_attacking
182
183 attacking_examples =
184     augment_with_paraphraser_for_one_iteration(attacking_examples)
185
186 random.shuffle(attacking_examples)
187
188 more_clear_attacking_examples =
189     random.sample(attacking_examples,k=(int)(0.5 *
190         len(attacking_examples)))
191
192 attack_labels = ["Attack" for i in
193     range(len(more_clear_attacking_examples))]
194
195 attack_dict = {"Examples": more_clear_attacking_examples, "labels": attack_labels}
196
197 attack_df = pd.DataFrame(attack_dict)
198
199 return attack_df
200
201
202 attack_df = generate_attack_examples()
203
204 attack_df.to_csv("attacking_examples.csv", index=False, sep=";")
```

Листинг А.1: Фрагмент программы для генерации данных

ПРИЛОЖЕНИЕ Б. ТЕКСТ ПРОГРАММЫ ДЛЯ ПОИСКА ОПТИМАЛЬНЫХ ПАРАМЕТРОВ

```
1 import wandb
2 import evaluate
3 import numpy as np
4 from datasets import Dataset, load_dataset
5 from transformers import (
6     AutoTokenizer,
7     AutoModelForSequenceClassification,
8     TrainingArguments,
9     Trainer,
10    DataCollatorWithPadding,
11    EarlyStoppingCallback
12 )
13 from tqdm import tqdm
14
15 tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
16
17 def preprocess_function(examples):
18     return tokenizer(examples['examples'], truncation=True,
19                      return_tensors='pt', padding='max_length')
20
21 dataset = load_dataset("neurae/dnd_style_intents",
22                        use_auth_token="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")
23
24 train_dataset = dataset["train"]
25 eval_dataset = dataset["eval"]
26
27 def compute_metrics(eval_preds):
28     metrics = dict()
29     accuracy = evaluate.load("accuracy")
30     precision = evaluate.load("precision")
31     recall = evaluate.load("recall")
32     f1 = evaluate.load("f1")
33
34     logits = eval_preds.predictions
35     labels = eval_preds.label_ids
```

```

36     preds = np.argmax(logits, axis=-1)
37     metrics = {
38         'accuracy': accuracy.compute(predictions=preds, references=labels),
39         'precision-micro': precision.compute(predictions=preds,
40             references=labels, average='micro'),
41         'precision-macro': precision.compute(predictions=preds,
42             references=labels, average='macro'),
43         'recall-micro': recall.compute(predictions=preds,
44             references=labels, average='micro'),
45         'recall-macro': recall.compute(predictions=preds,
46             references=labels, average='macro'),
47         'f1-micro': f1.compute(predictions=preds, references=labels,
48             average='micro'),
49         'f1-macro': f1.compute(predictions=preds, references=labels,
50             average='macro')
51     }
52     return metrics
53
54 model =
55     AutoModelForSequenceClassification.from_pretrained("bert-base-cased",
56     num_labels=17)
57
58 train_tokenized_dataset = train_dataset.map(preprocess_function,
59     batched=True)
60 eval_tokenized_dataset = eval_dataset.map(preprocess_function, batched=True)
61
62 train_tokenized_dataset.set_format("pt", columns=["input_ids",
63     "attention_mask", "labels"], output_all_columns=True)
64 eval_tokenized_dataset.set_format("pt", columns=["input_ids",
65     "attention_mask", "labels"], output_all_columns=True)
66
67 sweep_config = {
68     'method': 'bayes',
69     'name': 'sweep',
70     'metric': {
71         'goal': 'minimize',
72         'name': 'eval_loss'
73     },
74     'parameters': {
75         'lr': {'max': 1e-3, 'min': 1e-5},
76         'scheduler_type': { 'values': ['linear', 'cosine',
77             'cosine_with_restarts', 'polynomial', 'constant',
78             'constant_with_warmup']},
79         'weight_decay': {'min': 0.0, 'max': 0.1}
80     }
81 }
82 sweep_id = wandb.sweep(
83     sweep_config, project="bert-sweep")

```

```

71
72 def train_with_wandb(config=None):
73     with wandb.init(config=config):
74         config = wandb.config
75         training_args = TrainingArguments(
76             evaluation_strategy="epoch",
77             output_dir="models/bert",
78             overwrite_output_dir=True,
79             logging_strategy="steps",
80             logging_dir="models/bert/logs",
81             logging_steps=100,
82             learning_rate=config.lr,
83             per_device_train_batch_size=16,
84             per_device_eval_batch_size=16,
85             num_train_epochs=8,
86             weight_decay=config.weight_decay,
87             report_to="wandb",
88             load_best_model_at_end=True,
89             save_strategy="epoch",
90             metric_for_best_model="eval_loss",
91             greater_is_better=False,
92             torch_compile=True,
93             bf16=True,
94             optim="adafactor",
95             lr_scheduler_type=config.scheduler_type,
96             gradient_accumulation_steps=4
97         )
98
99         trainer = Trainer(
100             model=model,
101             args=training_args,
102             train_dataset=train_tokenized_dataset,
103             eval_dataset=eval_tokenized_dataset,
104             tokenizer=tokenizer,
105             data_collator=data_collator,
106             compute_metrics=compute_metrics,
107             callbacks=[EarlyStoppingCallback(early_stopping_patience=3)])
108
109         trainer.train()
110
111
112 wandb.agent(sweep_id, train_with_wandb, count=10)

```

Листинг Б.1: Текст программы для проведения серии экспериментов

ПРИЛОЖЕНИЕ В. ТЕКСТ ПРОГРАММЫ ДЛЯ СРАВНЕНИЯ МОДЕЛЕЙ

```
1 import wandb
2 import evaluate
3 import numpy as np
4 from datasets import Dataset, load_dataset
5 from transformers import (
6     AutoTokenizer,
7     AutoModelForSequenceClassification,
8     TrainingArguments,
9     Trainer,
10    DataCollatorWithPadding,
11    EarlyStoppingCallback,
12    AutoConfig
13 )
14
15 tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
16
17
18 def preprocess_function(examples):
19     return tokenizer(examples['text'], truncation=True,
20                      return_tensors='pt', padding="max_length")
21
22 dataset = load_dataset("neurae/dnd_style_intents",
23                        use_auth_token="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")
24
25 train_dataset = dataset["train"]
26 eval_dataset = dataset["eval"]
27 test_dataset = dataset["test"]
28
29 train_tokenized_dataset = train_dataset.map(preprocess_function,
30                                              batched=True)
31 eval_tokenized_dataset = eval_dataset.map(preprocess_function, batched=True)
32 test_tokenized_dataset = test_dataset.map(preprocess_function, batched=True)
33
34 train_tokenized_dataset.set_format("pt", columns=["input_ids",
35                                      "attention_mask", "labels"], output_all_columns=True)
36 eval_tokenized_dataset.set_format("pt", columns=["input_ids",
37                                      "attention_mask", "labels"], output_all_columns=True)
38 test_tokenized_dataset.set_format("pt", columns=["input_ids",
39                                      "attention_mask", "labels"], output_all_columns=True)
```

```

34
35     data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
36
37     def compute_metrics(eval_preds):
38         metrics = dict()
39         accuracy = evaluate.load("accuracy")
40         precision = evaluate.load("precision")
41         recall = evaluate.load("recall")
42         f1 = evaluate.load("f1")
43
44         logits = eval_preds.predictions
45         labels = eval_preds.label_ids
46         preds = np.argmax(logits, axis=-1)
47         metrics = {
48             'accuracy': accuracy.compute(predictions=preds, references=labels),
49             'precision-micro': precision.compute(predictions=preds,
50                 references=labels, average='micro'),
51             'precision-macro': precision.compute(predictions=preds,
52                 references=labels, average='macro'),
53             'recall-micro': recall.compute(predictions=preds,
54                 references=labels, average='micro'),
55             'recall-macro': recall.compute(predictions=preds,
56                 references=labels, average='macro'),
57             'f1-micro': f1.compute(predictions=preds, references=labels,
58                 average='micro'),
59             'f1-macro': f1.compute(predictions=preds, references=labels,
60                 average='macro')
61         }
62
63         return metrics
64
65     label2id = {
66         "Attack":0,
67         "Complete quest": 1,
68         "Deliver":2,
69         "Drival":3,
70         "Exchange": 4,
71         "Farewell": 5,
72         "Follow": 6,
73         "General": 7,
74         "Greeting": 8,
75         "Join": 9,
76         "Joke": 10,
77         "Knowledge":11,
78         "Move":12,
79         "Protect": 13,
80         "Recieve quest": 14,
81         "Message": 15,
82         "Threat": 16
83     }

```

```

76     }
77
78     id2label = {value:key for key, value in label2id.items()}
79
80     config = AutoConfig.from_pretrained("bert-base-cased", label2id=label2id,
81                                         id2label=id2label)
82
83     model = AutoModelForSequenceClassification.from_pretrained(
84         "bert-base-cased", config=config)
85
86
87     wandb.init(project="nlu embeddings", name="bert")
88
89     training_args = TrainingArguments(
90         evaluation_strategy='epoch',
91         output_dir='./models/bert',
92         overwrite_output_dir=True,
93         logging_strategy='steps',
94         logging_dir='./models/bert/logs',
95         logging_steps=100,
96         learning_rate=0.00013253246006813307,
97         per_device_train_batch_size=16,
98         per_device_eval_batch_size=16,
99         num_train_epochs=8,
100        weight_decay=0.0695551706838144,
101        report_to='wandb',
102        load_best_model_at_end=True,
103        save_strategy='epoch',
104        metric_for_best_model="eval_loss",
105        greater_is_better=False,
106        lr_scheduler_type="constant",
107        torch_compile=True,
108        bf16=True,
109        optim="adafactor",
110        gradient_accumulation_steps=4
111    )
112    trainer = Trainer(
113        model=model,
114        args=training_args,
115        train_dataset=train_tokenized_dataset,
116        eval_dataset=eval_tokenized_dataset,
117        tokenizer=tokenizer,
118        data_collator=data_collator,
119        compute_metrics=compute_metrics,
120        callbacks=[EarlyStoppingCallback(early_stopping_patience=3)]
121    )
122    trainer.train()
123    _, _, metrics = trainer.predict(test_dataset=test_tokenized_dataset)
124    print(metrics)

```

```
123 wandb.finish(exit_code=0)
```

Листинг B.1: Программа для обучения модели

ПРИЛОЖЕНИЕ Г. ГРАФИКИ ПРИ ПОИСКЕ ОПТИМАЛЬНЫХ ПАРАМЕТРОВ



Рисунок Г.1 – График функции ошибки во время обучения BERT

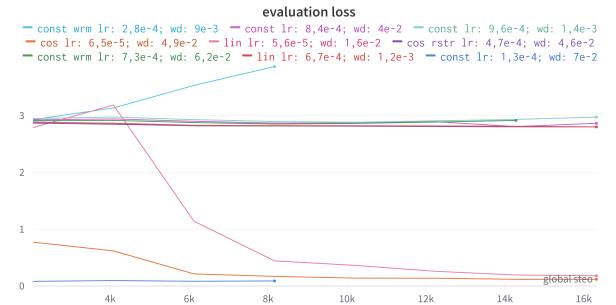


Рисунок Г.2 – График функции ошибки при валидации BERT

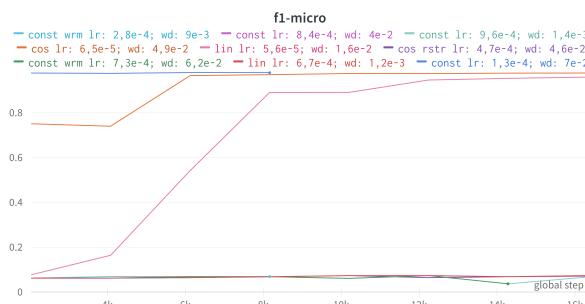


Рисунок Г.3 – Micro F_1 во время обучения BERT



Рисунок Г.4 – Macro F_1 во время обучения BERT

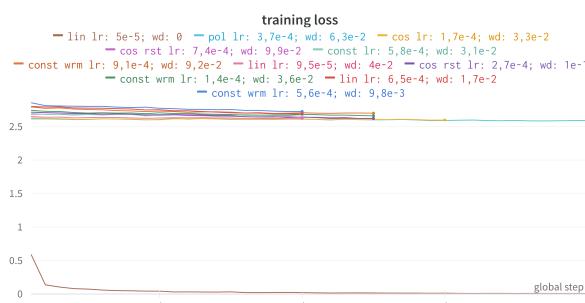


Рисунок Г.5 – График функции ошибки во время обучения RoBERTa



Рисунок Г.6 – График функции ошибки при валидации RoBERTa

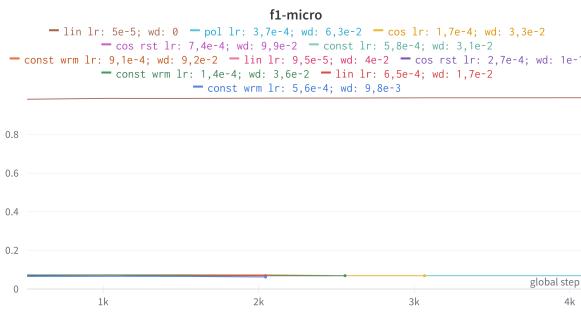


Рисунок Г.7 – Micro F_1 во время обучения RoBERTa

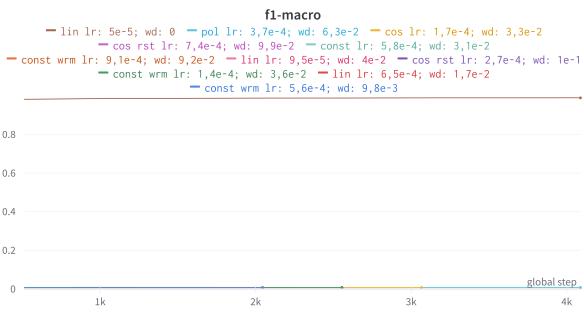


Рисунок Г.8 – Macro F_1 во время обучения RoBERTa



Рисунок Г.9 – График функции ошибки во время обучения DistilBERT

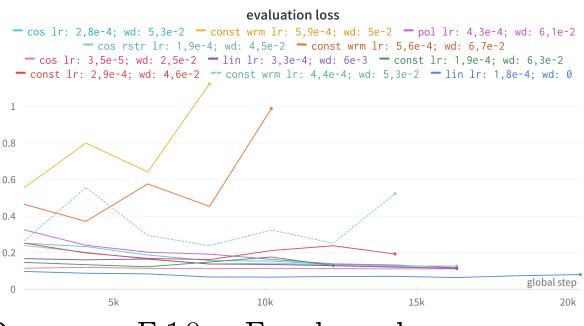


Рисунок Г.10 – График функции ошибки при валидации DistilBERT

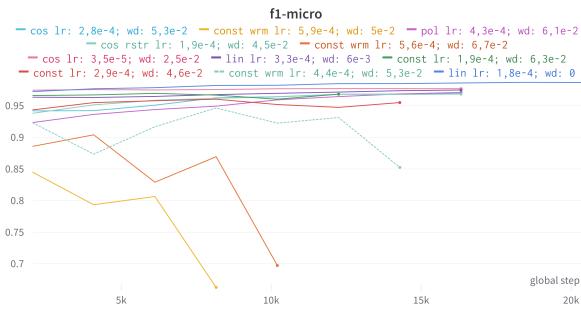


Рисунок Г.11 – Micro F_1 во время обучения DistilBERT

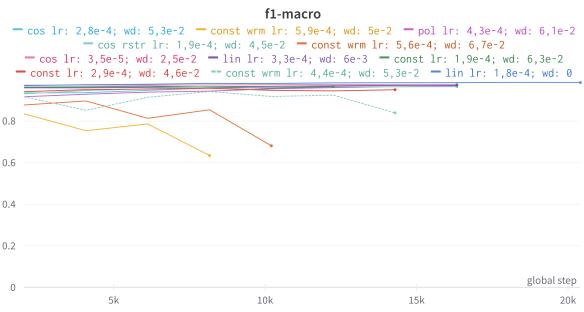


Рисунок Г.12 – Macro F_1 во время обучения DistilBERT



Рисунок Г.13 – График функции ошибки во время обучения ALBERT

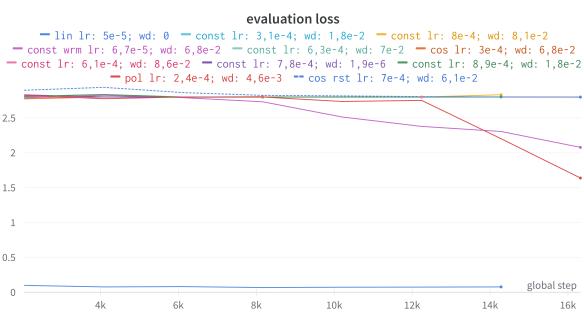


Рисунок Г.14 – График функции ошибки при валидации ALBERT

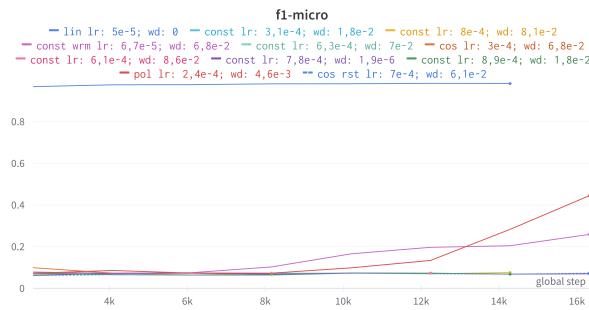


Рисунок Г.15 – Micro F_1 во время обучения ALBERT

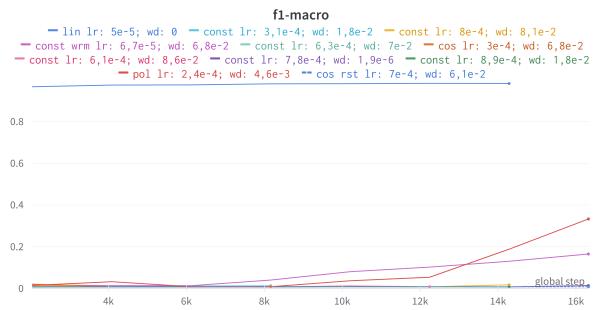


Рисунок Г.16 – Macro F_1 во время обучения ALBERT



Рисунок Г.17 – График функции ошибки во время обучения ELECTRA

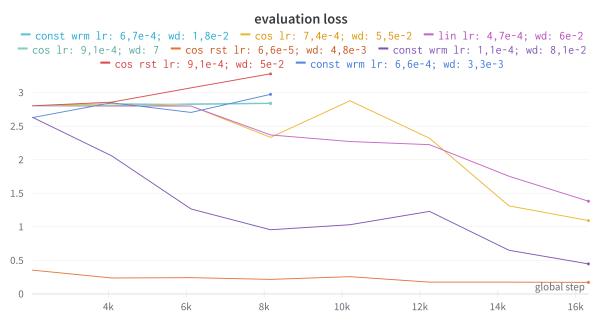


Рисунок Г.18 – График функции ошибки при валидации ELECTRA



Рисунок Г.19 – Micro F_1 во время обучения ELECTRA



Рисунок Г.20 – Macro F_1 во время обучения ELECTRA



Рисунок Г.21 – График функции ошибки во время обучения OPT

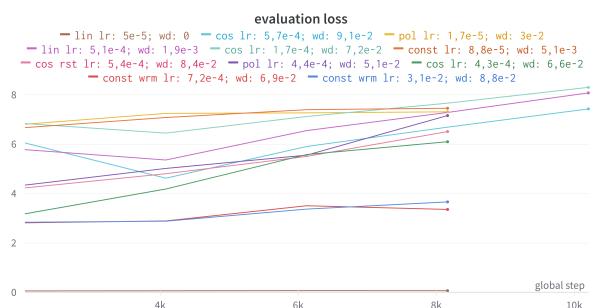


Рисунок Г.22 – График функции ошибки при валидации OPT

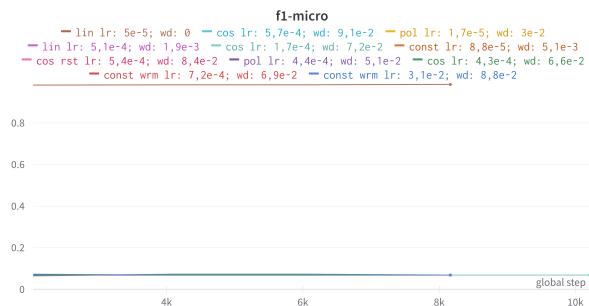


Рисунок Г.23 – Micro F_1 во время обучения ОРТ

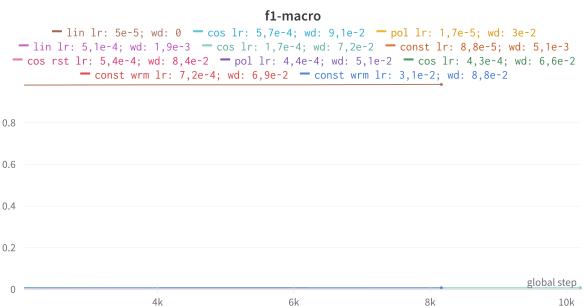


Рисунок Г.24 – Macro F_1 во время обучения ОРТ

ПРИЛОЖЕНИЕ Д. ГРАФИКИ ПРИ ОБУЧЕНИИ МОДЕЛЕЙ ДЛЯ СРАВНЕНИЯ

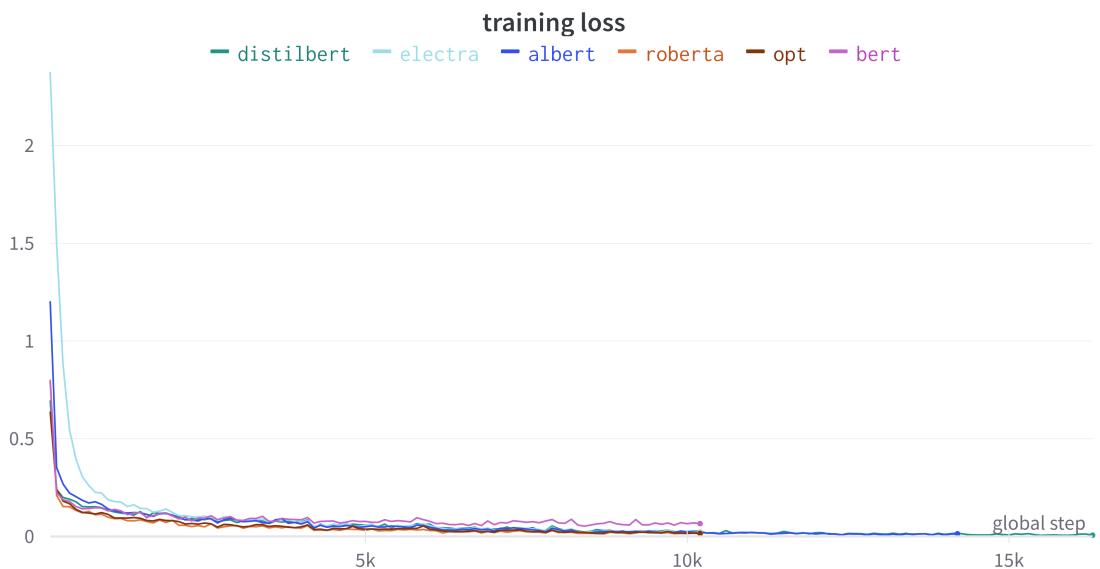


Рисунок Д.1 – График функции ошибки во время обучения моделей

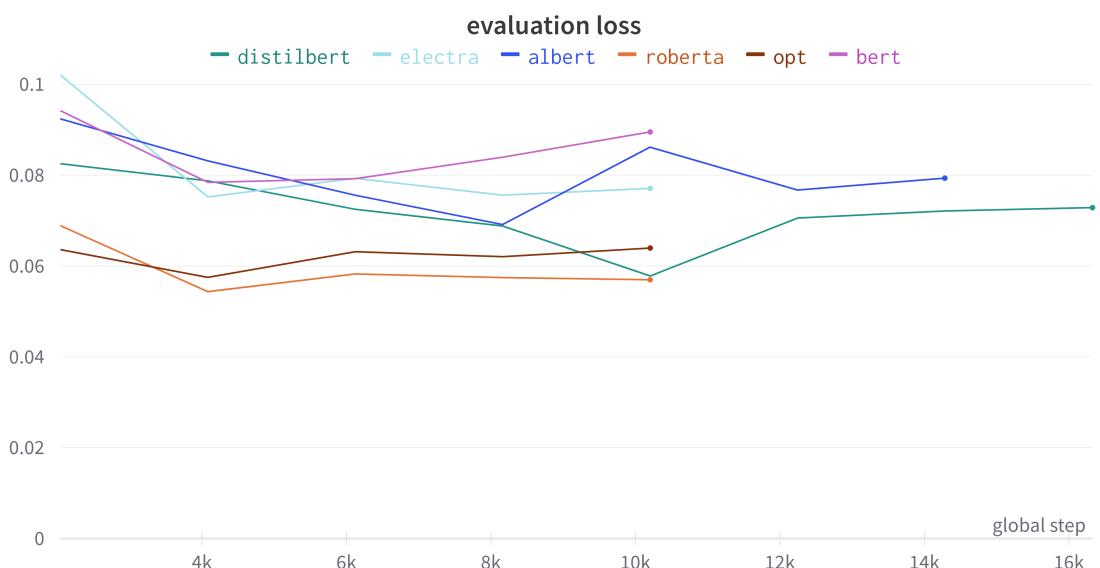


Рисунок Д.2 – График функции ошибки при валидации моделей