Федеральное государственное автономное образовательное учреждение

высшего образования

«Национальный исследовательский университет «Высшая школа экономики»

Факультет компьютерных наук

Образовательная программа Прикладная математика и информатика

бакалавриат

Образовательная программа 09.03.04 «Программная инженерия»

**О Т Ч Е Т**

по производственной практике

в департаменте анализа данных и искусственного интеллекта, НИУ ВШЭ

Выполнил студент гр. БПИ185

Малашенко Борис Тарасович

*(ФИО)*

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*(подпись)*

Руководитель практики от Организации

Старший преподаватель департамента анализа данных и искусственного интеллекта, м.н.с. МЛ ИССА, НИУ ВШЭ, Паринов Андрей Андреевич,

| *21.07.2021* | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| --- | --- | --- |
| *дата* | *оценка по 10 бальной шкале* | *Подпись* |

Руководитель практики от департамента программной инженерии

Ахметсафина Римма Закиевна, доцент департамента ПИ

| *21.07.2021* | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| --- | --- | --- |
| *дата* | *оценка по 10 бальной шкале* | *подпись* |

**Москва, 2021**

Содержание

[Введение](#_gjdgxs) **3**

[Цель](#_30j0zll) **3**

[**Задачи практики**](#_vtwoo2jf7rbi) **3**

[Календарный план-график](#_1fob9te) **3**

[Обзор технологий](#_3znysh7) **4**

[Тестирование](#_sunprhpm6vno) 4

[Рейтинг моделей](#_p0zz7lejd7b2) 4

[**Описание методов, моделей, алгоритмов**](#_aiviqtutqn3d) **5**

[Архитектура BERT](#_yjlmu9h7vr4k) 5

[Маски](#_kyv98x2vi4s0) 5

[Предсказание следующего предложения](#_hkxl8qduiix1) 6

[Описание гиперпараметров](#_wykq4if5yvjy) 6

[BertTokenizer](#_3cwljfifo5r1) 6

[Encoder](#_60wkuvkbs4in) 6

[DataLoader](#_8ay2wsp1wzqg) 6

[Optimizer](#_3f795bhdwx00) 6

[Scheduler](#_cliw1vi5tu64) 7

[Russian SuperGLUE](#_6er8d8i3enuo) 7

[**Используемые технологии**](#_xzll9iaa3syo) **9**

[Выбор модели](#_e59hcsryx88w) 10

[Результаты](#_2et92p0) **10**

[Тестовые данные](#_de64vt1y140s) 10

[Программная реализация](#_cou8rn77pse2) 11

[**Заключение**](#_pzk7bdk0qj0h) **14**

[Информационные источники](#_tyjcwt) **14**

# Введение

Ранее обучение моделей для обработки естественного языка (NLP) тренировало модели с нуля для каждой отдельной задачи. Но подобный подход может быть крайне дорогостоящим, так как требует сбора большого числа данных, разработки архитектуры, настройки гиперпараметров и оценки результатов.

В настоящий момент существует большое число заранее обученных на больших объемах данных и настроенных моделей, которые могут быть затем использованы для различных проблем NLP. Для высокой точности модель дообучается на данных для каждой индивидуальной задачи, объём которых может быть на порядки меньше необходимого для обучения аналогичной модели с нуля, а для выбора гиперпараметров достаточно обратиться к тестам GLUE или SuperGlue.

BERT - одна из таких моделей. В данной практической работе рассматриваются модели на основе BERT, обученные для работы с русским языком для проблемы классификации.

# Цель

Цель практики - реализация программного решения для задачи мультиклассовой классификации текста на основе нейронной сети BERT.

# Задачи практики

* изучение классификации на основе BERT, а также знакомство с алгоритмами кластеризации, алгоритмами создания ассоциативных правил и алгоритмом SOM;
* поиск и исследование моделей BERT с различными гиперпараметрами;
* изучение библиотек для работы с BERT в языке Python;
* выбор модели и гиперпараметров для задачи классификации;
* реализация на языке Python;
* тестирование.

# Календарный план-график

| № п/п | Сроки проведения | Выполненные работы | Отметка руководителя о выполнении (подпись) |
| --- | --- | --- | --- |
| 1 | 01.07.2021 | Инструктаж по ознакомлению с требованиями охраны труда, техники безопасности, пожарной безопасности, а также правилами внутреннего трудового распорядка |  |
| 2 | 05.07.2021 | Изучение алгоритмов классификации и кластеризации, алгоритмов создания ассоциативных правил, алгоритма SOM |  |
| 3 | 08.07.2021 | Изучение библиотек для реализации алгоритмов на языке Python |  |
| 4 | 15.07.2021 | Реализация алгоритмов на языке Python |  |
| 5 | 18.07.2021 | Тестирование алгоритмов |  |
| 6 | 21.07.2021 | Разработка цифрового ассистента студента |  |

# Обзор технологий

BERT [1] — это нейронная сеть от Google, показавшая с большим отрывом state-of-the-art результаты на целом ряде задач (на момент своего создания). С помощью BERT можно создавать программы с ИИ для обработки естественного языка: отвечать на вопросы, заданные в произвольной форме, создавать чат-ботов, автоматические переводчики, анализировать текст и так далее.

На основе BERT существует большое число модификаций. Из наиболее известных стоит упомянуть RoBERTa [2]. Это модель на основе BERT, созданная Facebook и обученная на гораздо большем объёме данных с небольшими модификациями.

Модели показывают различные результаты в зависимости от задачи и языка. Для текущей практики необходимо подобрать модель, показывающую наилучшие результаты при работе с русским языком. Далее будут рассмотрены критерии выбора такой модели.

## Тестирование

Для сравнения воспользуемся бенчмарком Russian SuperGLUE [3]. Он предлагает методологию тестирования, основанную на задачах, которые предлагаются для “Strong AI” - логика, здравый смысл, рассуждение. Все тесты основаны на методологиях GLUE [4] и SuperGLUE [5], но полностью составлены из русскоязычных данных, собранных из открытых источников, новостей и литературных журналов, затем вручную проверены и дополнены человеческой оценкой на Яндекс.Толока.

Подробнее бенчмарк рассматривается в разделе [Russian SuperGLUE](#_6er8d8i3enuo).

## Рейтинг моделей

Рассмотрим несколько основанных на BERT моделей, выполненных командами sberdevices и DeepPavlov.

| **Rank** | **Name** | **Team** | **Link** | **Score** |
| --- | --- | --- | --- | --- |
| 4 | [ruRoberta-large finetune](https://russiansuperglue.com/login/submit_info/1166) | sberdevices | [url](https://huggingface.co/sberbank-ai/ruRoberta-large) | 0.624 |
| 6 | [ruBert-large finetune](https://russiansuperglue.com/login/submit_info/1182) | sberdevices | [url](https://huggingface.co/sberbank-ai/ruBert-large) | 0.583 |
| 7 | [ruBert-base finetune](https://russiansuperglue.com/login/submit_info/1164) | sberdevices | [url](https://huggingface.co/sberbank-ai/ruBert-base) | 0.578 |
| 11 | [RuBERT plain](https://russiansuperglue.com/login/submit_info/176) | DeepPavlov | [url](https://huggingface.co/DeepPavlov/rubert-base-cased) | 0.521 |
| 12 | [SBERT\_Large\_mt\_ru\_finetuning](https://russiansuperglue.com/login/submit_info/944) | sberdevices | [url](https://huggingface.co/sberbank-ai/sbert_large_mt_nlu_ruhttps://huggingface.co/sberbank-ai/sbert_large_mt_nlu_ru) | 0.514 |

Ниже в таблице приведены результаты моделей по каждому тесту.

| **Rank** | **LiDiRus** | **RCB** | **PARus** | **MuSeRC** | **TERRa** | **RUSSE** | **RWSD** | **DaNetQA** | **RuCoS** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 0.339 | 0.357 / 0.518 | 0.508 | 0.83 / 0.561 | 0.801 | 0.715 | 0.571 | 0.82 | 0.73 / 0.716 |
| 6 | 0.235 | 0.356 / 0.5 | 0.492 | 0.76 / 0.427 | 0.704 | 0.682 | 0.669 | 0.773 | 0.68 / 0.658 |
| 7 | 0.224 | 0.333 / 0.509 | 0.476 | 0.742 / 0.399 | 0.703 | 0.706 | 0.669 | 0.712 | 0.74 / 0.716 |
| 11 | 0.191 | 0.367 / 0.463 | 0.574 | 0.711 / 0.324 | 0.642 | 0.726 | 0.669 | 0.639 | 0.32 / 0.314 |
| 12 | 0.218 | 0.351 / 0.486 | 0.498 | 0.642 / 0.319 | 0.637 | 0.657 | 0.675 | 0.697 | 0.35 / 0.347 |

# Описание методов, моделей, алгоритмов

## 

## Архитектура BERT

BERT использует Transformer, механизм внимания, который изучает контекстные отношения между словами (или подсловами) в тексте. Transformer включает в себя два отдельных механизма - кодировщик, который считывает вводимый текст, и декодер, который производит прогноз для задачи. Поскольку целью BERT является создание языковой модели, необходим только механизм кодировщика [6].

В отличие от однонаправленных моделей, которые считывают вводимый текст последовательно (слева направо или справа налево), кодировщик Transformer считывает сразу всю последовательность слов. Эта характеристика позволяет модели узнавать контекст слова на основе всего его окружения (с обеих сторон от слова).

При обучении языковых моделей возникает проблема определения цели прогнозирования. Многие модели предсказывают следующее слово в последовательности, используя однонаправленный подход, который по своей сути ограничивает контекстное обучение. Чтобы преодолеть эту проблему, BERT использует две стратегии обучения.

### Маски

Перед загрузкой последовательностей слов в BERT 15% слов в каждой последовательности заменяются токеном [MASK]. Затем модель пытается предсказать исходное значение замаскированных слов на основе контекста, обеспечиваемого другими, немаскированными словами в последовательности.

Функция потерь BERT принимает во внимание только предсказание замаскированных значений и игнорирует предсказание немаскированных слов. Как следствие, модель сходится медленнее, чем однонаправленные модели, что компенсируется повышенной осведомленностью о контексте.

### Предсказание следующего предложения

В процессе обучения BERT модель получает пары предложений в качестве входных данных и учится предсказывать, может ли быть второе предложение в паре логическим следствием первого предложения в исходном документе.

Чтобы модель могла различать два предложения при обучении, входные данные перед вводом в модель дополнительно обрабатываются. В начало первого предложения вставляется токен [CLS], а в конец каждого предложения вставляется токен [SEP].

## Описание гиперпараметров

### BertTokenizer

Токенизатор — инструмент для автоматического разделения текста на токены, т.е. на слова и другие цепочки символов, которые мы хотим считать минимальными линейными единицами текста.

Рассмотрим гиперпараметры токенизатора BERT:

* **vocab\_file** – File containing the vocabulary.
* **do\_lower\_case** – приведение строки к нижнему регистру;
* **do\_basic\_tokenize** – применение простейшей токенизации перед WordPiece;

### Encoder

* **add\_special\_tokens** – Следует ли кодировать последовательности специальными токенами, относящимися к их модели;
* **padding** – контроль отступов;
* **truncation** – контроль усечения.

### DataLoader

* **train/val size** – размер выборок для обучения и валидации;
* **batch\_size** – размер батча;
* **sampler** – способ разбиения выборки на батчи.

### Optimizer

Гиперпараметры для AdamW:

* **lr** – скорость обучения;
* **betas** – b1 и b2 параметры;
* **eps** – эпсилон для числовой стабильности;
* **weight\_decay** – применять снижение веса;
* **correct\_bias** – исправление предвзятости в Adam.

### Scheduler

* **num\_training\_steps** – общее число шагов для обучения;
* **num\_warmup\_steps** – общее число шагов для разминки.

## Russian SuperGLUE

* **Лингвистическая диагностика** (LiDiRus). Модель должна определить, является ли одно предложение логическим следствием другого.

| "sentence1": "Кошка сидела на коврике.",  "sentence2": "Кошка не сидела на коврике.",  "label": "not\_entailment",  "knowledge": "",  "lexical-semantics": "",  "logic": "Negation",  "predicate-argument-structure": "" |
| --- |

* **Набор данных преступлений** (RCB). Модель устанавливает, является ли гипотеза логическим выводом из предпосылки.

| "premise": "Сумма ущерба составила одну тысячу рублей. Уточняется, что на место происшествия выехала следственная группа, которая установила личность злоумышленника. Им оказался местный житель, ранее судимый за подобное правонарушение.",  "hypothesis": "Ранее местный житель совершал подобное правонарушение.",  "verb": "судить",  "negation": "no\_negation",  "label": "entailment",  "idx": 269 |
| --- |

* **Выбор правдоподобных альтернатив** (PARus). Модель по предпосылке выбирает наиболее вероятное суждение. Тест оценивает прогресс в области причинно-следственных рассуждений на основе здравого смысла.

| "premise": "Гости вечеринки прятались за диваном.",  "choice1": "Это была вечеринка-сюрприз.",  "choice2":"Это был день рождения.",  "question": "cause",  "label": 0,  "idx": 4 |
| --- |

* **Понимание чтения нескольких предложений** (MuSeRC). Это задача понимания прочитанного, в которой на вопросы можно ответить, только приняв во внимание информацию из нескольких предложений.

| "id": 397,  "text": some\_text,  "questions": [...],  "idx": 0 |
| --- |

* **Распознавание текстового ввода** (TERRA). Эта задача требует распознавания по двум фрагментам текста, вытекает ли (может ли быть выведено) значение одного текста из другого текста.

| "premise": "Автор поста написал в комментарии, что прорвалась канализация.",  "hypothesis": "Автор поста написал про канализацию.",  "label": "entailment",  "idx": 6062 |
| --- |

* **Соответствие контекстов** (RUSSE). В зависимости от контекста неоднозначное слово может иметь несколько, потенциально не связанных между собой значений. Обычные статические встраивания слов, такие как Word2vec и GloVe, не могут отразить эту динамическую семантическую природу. Контекстуализированные вложения слов - это попытка устранить это ограничение путем вычисления динамических представлений для слов, которые могут адаптироваться в зависимости от контекста.

| "idx" : 8,  "word" : "дорожка",  "sentence1" : "Бурые ковровые дорожки заглушали шаги",  "sentence2" : "Приятели решили выпить на дорожку в местном баре",  "start1" : 15,  "end1" : 23,  "start2" : 26,  "end2" : 34,  "label" : false,  "gold\_sense1" : 1,  "gold\_sense2" : 2 |
| --- |

* **Схема Винограда** (RWSD). Схема Винограда - это пара предложений, которые отличаются только одним или двумя словами и содержат неоднозначность, которая разрешается противоположными способами в двух предложениях и требует использования мировых знаний и рассуждений для ее разрешения.

| "sentence1": "Кубок не помещается в коричневый чемодан, потому что он слишком большой.",  "sentence2": "Кубок не помещается в коричневый чемодан, потому что он слишком маленький."  "label": 0,  "idx": 5 |
| --- |

* **Ответ на да/нет вопросы** (DaNetQA). Каждый пример представляет собой тройку (вопрос, отрывок, ответ). Настройка классификации текстовых пар аналогична существующим задачам вывода на естественном языке.

| "text": "В период с 1969 по 1972 год по программе «Аполлон» было выполнено 6 полётов с посадкой на Луне. Всего на Луне высаживались 12 астронавтов США. Список космонавтов Список космонавтов — участников орбитальных космических полётов Список астронавтов США — участников орбитальных космических полётов Список космонавтов СССР и России — участников космических полётов Список женщин-космонавтов Список космонавтов, посещавших МКС Энциклопедия астронавтики.",  "question": "Был ли человек на луне?",  "label": true,  "idx": 5 |
| --- |

* **Понимание прочитанного с рассуждением** (RuCoS). Это крупномасштабный набор данных о понимании прочитанного, который требует рассуждений на основе здравого смысла. RuCoS состоит из запросов, автоматически генерируемых из новостных статей. Ответ на каждый запрос - это отрывок текста из резюмирующего отрывка соответствующей новости. Цель RuCoS - оценить способность машины рассуждать на основе здравого смысла при понимании прочитанного.

| "source": "Lenta",  "passage": {  "text":  "... Через несколько дней после того, как Виктор Гаврилов бросил своих детей в аэропорту, он явился с повинной к следователям в городе Батайске Ростовской области. ...",  "entities": [  ...  {"start": 470, "end": 485, "text": "Виктор Гаврилов"},  ...  ]  },  "qas": [  {  "query": "26 января @placeholder бросил сыновей в возрасте пяти и семи лет в Шереметьево.",  "answers": [  {"start": 470, "end": 485, "text": "Виктор Гаврилов"}  ],  "idx": 0  }  ],  "idx": 0 |
| --- |

# Используемые технологии

## Выбор модели

В таблице ниже приведено сравнивается число и размер слоёв в моделях и параметры при обучении.

| **Модель** | **L** | **H** | **batch** | **Optimizer** | **Learning rate** | **Число эпох** |
| --- | --- | --- | --- | --- | --- | --- |
| [ruRoberta-large finetune](https://russiansuperglue.com/login/submit_info/1166) | 24 | 1024 | 16/32 | Adam | 1e-5/2e-5/3e-5 | 10 |
| [ruBert-large finetune](https://russiansuperglue.com/login/submit_info/1182) | 24 | 1024 | 32 | Adam | 2e-5/3e-5/4e-5/5e-5 | 3 |
| [ruBert-base finetune](https://russiansuperglue.com/login/submit_info/1164) | 12 | 768 | 32 | Adam | 2e-5/3e-5/4e-5/5e-5 | 3 |
| [RuBERT plain](https://russiansuperglue.com/login/submit_info/176) | 12 | 768 | 32 | Adam | 2e-5/3e-5/4e-5/5e-5 | 3 |
| [SBERT\_Large\_mt\_ru\_finetuning](https://russiansuperglue.com/login/submit_info/944) | 24 | 1024 | 96/72 | Adam | 2e-6 | 25 |

На основе результатов теста Russian SuperGLUE, а конкретно теста RuCoS, который наиболее похож на проблему классификации, так как требует от модели выбрать ‘категорию’ на основе большого текста, выберем модель [ruBert-base finetune](https://russiansuperglue.com/login/submit_info/1164). Кроме того эта модель меньше весит и быстрее в обучении, чем [ruBert-large finetune](https://russiansuperglue.com/login/submit_info/1182), но общие результаты тестов лишь немного хуже.

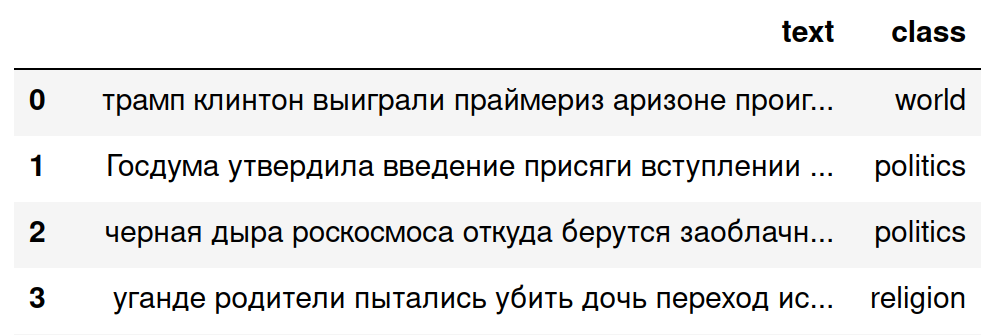
# Результаты

Рассмотрим применение выбранной модели в коде для задачи мультиклассовой классификации.

## Тестовые данные

Для тестирования будем использовать открытый набор данных **russian\_news\_dataset** [7]. Он состоит из заголовков, полных текстов статей и категорий. Загрузим датасет и оставим только заголовки (примерно 10 первых слов) и категории.

| df = pd.read\_csv("text\_series\_full.csv", delimiter=',')  df = df[['text', 'class']]  df['text'] = df['text'].apply(lambda x: ' '.join(x.split(' ')[:10]))  df = df.dropna() |
| --- |



## Программная реализация

Воспользуемся классом **LabelEncoder** из библиотеки **sklearn.preprocessing** и закодируем категории числовыми значениями.

| le = preprocessing.LabelEncoder()  df['class'] = le.fit\_transform(df['class']) |
| --- |

Для загрузки токенизатора воспользуемся библиотекой transformers. Чтобы получить доступ к последним моделям, библиотеку необходимо обновить до версии 4.8.2.

Импортируем класс **BertTokenizer** и воспользуемся методом **from\_pretrained**.

| tokenizer = BertTokenizer.from\_pretrained('sberbank-ai/ruBert-base') |
| --- |

Выделим в значения texts и labels заголовки и категории соответственно. Токенизируем texts методом **encode\_plus**.

| texts = df['text'].values  labels = df['class'].values  def encode(text):  encoded\_dict = tokenizer.encode\_plus(  text,  add\_special\_tokens = True,  pad\_to\_max\_length = True,  return\_tensors = 'pt',  truncation = True,  padding='max\_length'  )    return encoded\_dict['input\_ids']    input\_ids = list(map(lambda text: encode(text), tqdm(texts)))  input\_ids = torch.cat(input\_ids, dim=0)  labels = torch.tensor(labels) |
| --- |

Создадим выборки для обучения и валидации методом **train\_test\_split** из библиотеки **sklearn.model\_selection** с параметром **test\_size** равным 0.3.

| X\_train, X\_val, y\_train, y\_val = train\_test\_split(input\_ids, labels, test\_size=0.3, random\_state=42) |
| --- |

Установим значение **batch\_size** равным 32, создадим **DataLoader** для каждой выборки. **DataLoader** это класс библиотеки **torch.utils.data**, который будет по батчу передавать данные в модель. Для обучения будем передавать данные в случайном порядке при помощи класса **RandomSampler**, а для валидации последовательно при помощи класса **SequentialSampler**. В случае с обучением мы будем уверены, что модель не переобучится на отдельных частях датасета, а для валидации результат будет одинаков при любом порядке передачи данных.

| batch\_size = 32  train\_dataset = TensorDataset(X\_train, y\_train)  train\_dataloader = DataLoader(  train\_dataset,  sampler = RandomSampler(train\_dataset),  batch\_size = batch\_size  )  val\_dataset = TensorDataset(X\_val, y\_val)  validation\_dataloader = DataLoader(  val\_dataset,  sampler = SequentialSampler(val\_dataset),  batch\_size = batch\_size  ) |
| --- |

Загрузим модель при помощи класса **BertForSequenceClassification** из библиотеки **transformers**, указав число различных категорий в параметре **num\_labels**.

| model = BertForSequenceClassification.from\_pretrained(  "sberbank-ai/ruBert-base",  num\_labels = 7  ) |
| --- |

Определим оптимизатор. Для нашей задачи воспользуемся **AdamW** из библиотеки **transformers**. Укажем гиперпараметры **lr** и **eps**, в соответствии с документацией BERT.

| optimizer = AdamW(model.parameters(),  lr = 3e-5,  eps = 1e-8  ) |
| --- |

Укажем число эпох, шагов и планировщик **get\_linear\_schedule\_with\_warmup** из библиотеки **transformers**. Планировщик будет линейно уменьшать скорость обучения оптимизатора с установленного значения до нуля. В данном случае модель сходится достаточно быстро, что видно по итоговым графикам, но для другого набора данных параметр **num\_warmup\_steps** может быть увеличен.

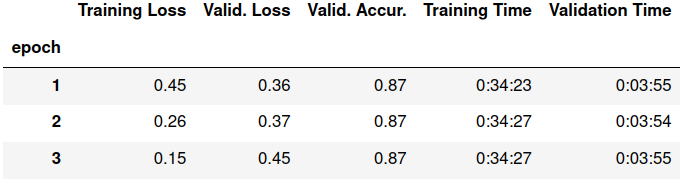
| epochs = 3  total\_steps = len(train\_dataloader) \* epochs  scheduler = get\_linear\_schedule\_with\_warmup(optimizer,  num\_warmup\_steps = 0,  num\_training\_steps = total\_steps) |
| --- |

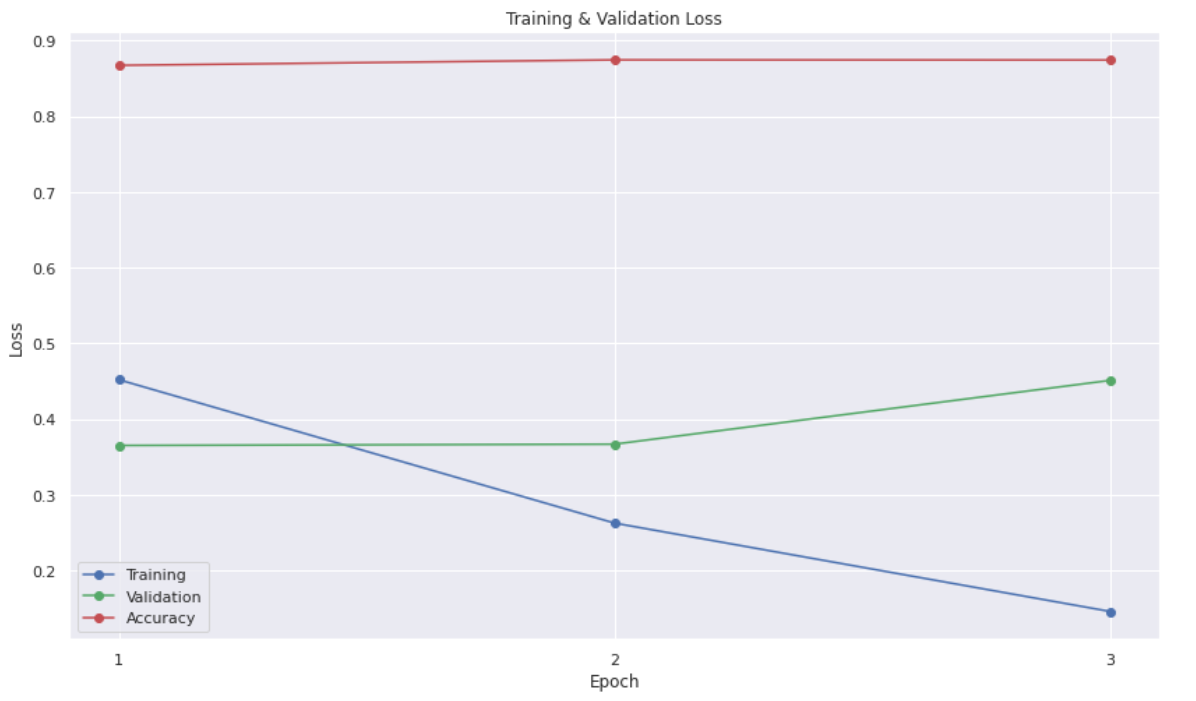
Для измерения точности модели во время обучения воспользуемся простой метрикой **flat\_accuracy**.

| def flat\_accuracy(preds, labels):  pred\_flat = np.argmax(preds, axis=1).flatten()  labels\_flat = labels.flatten()  return np.sum(pred\_flat == labels\_flat) / len(labels\_flat) |
| --- |

Запустим процесс обучения.

| for epoch\_i in range(0, epochs):  model.train()  for step, batch in enumerate(train\_dataloader):  optimizer.step()  scheduler.step()  model.eval() |
| --- |

Выведем итоговые результаты для потерь выборок обучения и валидации и точность на валидации для каждой эпохи.



# Заключение

В результате практики:

* рассмотрена модель обработки естественного языка BERT;
* изучены различные модели на основе BERT;
* проведено сравнение и выбор итоговой модели для конкретной задачи;
* выбранная модель реализована в коде на языке python;
* проведены тесты точности модели.

# Информационные источники

1. Devlin J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding //arXiv preprint arXiv:1810.04805. – 2018.
2. Liu Y. et al. Roberta: A robustly optimized bert pretraining approach //arXiv preprint arXiv:1907.11692. – 2019.
3. Shavrina T. et al. Russiansuperglue: A russian language understanding evaluation benchmark //arXiv preprint arXiv:2010.15925. – 2020.
4. Wang A. et al. GLUE: A multi-task benchmark and analysis platform for natural language understanding //arXiv preprint arXiv:1804.07461. – 2018.
5. Wang A. et al. Superglue: A stickier benchmark for general-purpose language understanding systems //arXiv preprint arXiv:1905.00537. – 2019.
6. Vaswani A. et al. Attention is all you need //Advances in neural information processing systems. – 2017. – С. 5998-6008.
7. <https://www.kaggle.com/serart/nti-news-text-seriescsv> [Электронный ресурс] доступ свободный.
8. <https://github.com/BlessedTatonka/BERT_classification> репозиторий проекта.