МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Фёдоров Евгений Андреевич

Машинное обучение для отслеживания знаний с целью повышения качества обучения

Магистерская диссертация

специальность 1-31 80 09 "Прикладная математика и информатика"

Научный руководитель:
Орлович Юрий
Леонидович
доцент кафедры БМИ,
канд. физ.-мат. наук

Допущен к защите

«___» _____ 2021 г.

Заведующий кафедрой дискретной математики и алгоритмики
Котов Владимир Михайлович
доктор физико-математических наук, профессор

Минск, 2021

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

AUC (area under ROC curve) - метрика, показывающая качество классификатора.

GRU (Gated recurrent units) - разновидность рекуррентных сетей с механизмом вентилей.

LSTM (Long short-term memory) - разновидность рекуррентных сетей, умеющая запоминать как давние, так и недавние события.

NLP (Natural Language Processing) - направление искусственного интеллекта по обработке естественного языка.

валидационная выборка (validation) - часть данных, предназначенная для сравнения между собой обученных моделей.

градиентный бустинг (gradient boosting) - алгоритм машинного обучения для регрессионных и классификационных задач, который создает модель прогнозирования в виде ансамбля слабых моделей прогнозирования, обычно деревьев решений.

момент времени (timestamp) - отметка времени какого-либо события.

трансформер (transformer) - разновидность архитектуры глубоких нейронных сетей, предназначенная для работы с последовательностями данных.

тренировочная выборка (train) - часть данных, предназначенная для обучения

функция потерь (loss) - функция потерь, которую минимизирует алгоритм с целью получения лучшей модели машинного обучения.

эмбеддинг (embedding) - векторное представление какого-либо объекта: слова, вопроса, типа, правильного ответа. Может с нуля обучаться вместе с остальными слоями, а можем быть заранее предобученно на больших объемах данных, как Word2Vec.

ВВЕДЕНИЕ

Темой данной магистерской является отслеживания знаний с целью повышения качества обучения. Это процесс, в течении которого происходит отслеживания приобретенных навыков учащимся. При помощи анализа истории ответов учеников, можно сделать вывод, в каких темах они сильны, а в каких нет и на этой основе пытаться предсказать ответит ли он на новый вопрос.

Представим учителя, который пытается понять, насколько ученик силен, и дает соответствующие его уровню задания. Это очень важный, хоть и очевидный факт, так как от простых заданий ученик быстрее начинает скучать, от сложных расстраивается из невозможности их решить. Это все может привести к потере мотивации у обучаемого, и, следовательно, к ухудшению образования.

В 2018 году 260 миллионов детей не ходило в школу, другими словами, у них не было учителя, который способен объяснить им материал, выдавать задания, соответствующие их уровню знаний. Также во время пандемии много учеников перешло на домашнее обучение, при котором не так просто получить доступ к учителю.

Данную технологию можно будет использовать в качестве персонального контроля за учениками. Отслеживание их сильных сторон, рекомендации изучения теоретического материала в слабых темах, рекомендации заданий, которые соответствуют уровню знаний учащихся.

Глава 1
ОБЗОР СУЩЕСТВУЮЩИХ ПОДХОДОВ

В целом, большинство моделей, начиная с байесовских, в начале девяностых показали себя достаточно хорошо и помогали развиваться педагогике. Взрыв данных в последнее время, безусловно, пошел на пользу глубоким сетевым моделям и большинство работ за последние 2 года были посвящены глубоким нейронным сетям. Ситуация с короновирусом подтолкнула еще больше студентов к онлайн-образованию, что привело к дальнейшему взрыву данных. Почти все эти методы используют механизм внимания, который хорошо себя зарекомендовал в задачах обработки естественного языка.

1.1 Байесовское отслеживание знаний

Байесовское отслеживание знаний[1] моделирует состояние знаний как набор параметров, каждый из которых описывает конкретный навык у ученика. Таким образом знание ученика представлено в виде списка параметров. Скрытые марковские модели используются для их обновления по мере того, как студент отвечает на упражнения. У каждого навыка есть 4 параметра вероятности:

- $P(L_0)$ - вероятность того, что ученик знает навык, где индекс является порядковым номером вопроса с конкретным навыком;
- $P(T)$ - вероятность того, что ученик выучит навык после правильного ответа на вопрос;
- $P(S)$ - вероятность того, что ученик допустит ошибку имея навык;
- $P(G)$ - вероятность того, что ученик угадает ответ без навыка;

Далее определяются формулы перехода латентного состояния ученика после очередного ответа на вопрос, u - это студент, k - это навык:

- В самом начале для студента с конкретным навыком устанавливается вероятность решить задачу по умолчанию, так как мы ничего не знаем про конкретного студента.

  $P(L_1)_u^k = P(L_0)^k$

- В зависимости от того, правильно или не правильно ученик ответил на вопрос, будет обновлена условная вероятность.

$$P(L_t|obs = correct)_u^k = \frac{p(L_t)_u^k \cdot (1-p(S)^k)}{p(L_t)_u^k \cdot (1-p(S)^k) + (1-p(L_t)_u^k) \cdot p(G)^k}$$

$$P(L_t|obs = wrong)_u^k = \frac{P(L_t)_u^k \cdot P(S)^k}{P(L_t)_u^k \cdot P(S)^k + (1-P(L_t)_u^k) \cdot (1-P(G)^k)}$$

- Условная вероятность используется для обновления навыка, с которым был связан вопрос.

$$P(L_{t+1})_u^k = P(L_t|obs)_u^k + (1 - P(L_t|obs)_u^k) \cdot P(T)^k$$

- формула, которая предсказывает вероятность ответа на вопрос, с заданным навыком k.

$$P(C_{t+1})_u^k = P(L_{t+1})_u^k \cdot (1 - P(S)^k) + (1 - P(L_{t+1})_u^k) \cdot P(G)^k$$

В результате, используя заданные формулы, можно отслеживать параметры каждого навыка, а также пытаться предсказать, как ученик будет отвечать на вопросы в будущем. Самым большим недостатком этого подхода является то, что никак не учитывается память ученика, которая на самом деле не совершенная, и материал, выученный пол года назад, может полностью стереться из нее.

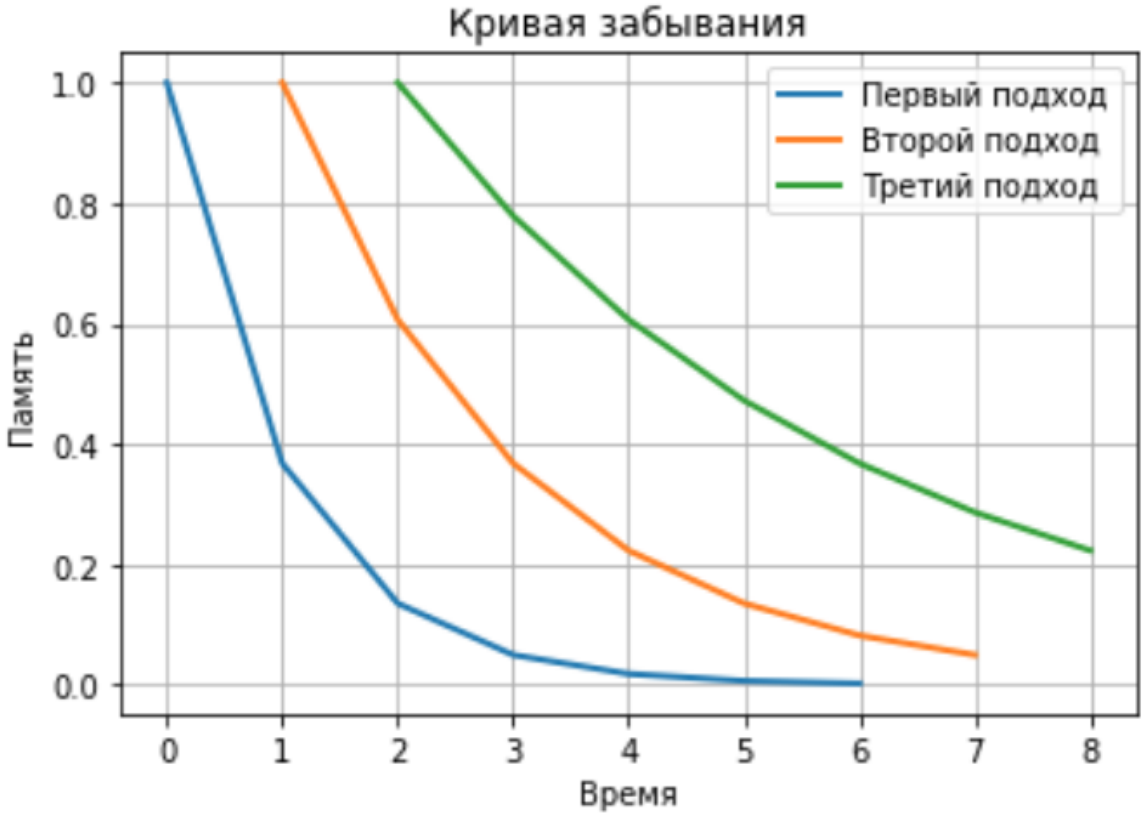

Рисунок 1.1 — Кривые забывания

На рисунке 1.1 изображена кривая забывания[2]. В частном случае она может зависеть от нескольких факторов, но в общем случае это кривая, которая со временем стремится к 0, а с каждым новым повторением, стремление к 0 замедляется. Кривые слева направо говорят о количестве повторений, синяя кривая означает первое изучение в момент времени 0, крайняя правая(зеленая) 3ее повторение на момент времени 3. Невооруженным глазом видно, что после 3ого повторения материал в памяти усваивается гораздо лучше, чем после первого. Данный факт стоит учитывать в модели отслеживания знаний. Поэтому далее будут рассмотрены модели машинного обучения, которые в каком-то виде учитывают особенности человеческой памяти.

1.2 Методы машинного обучение

1.2.1 Классическое машинное обучение

Предсказание правильности ответа на вопрос является задачей бинарной классификации, которую хорошо умеют решать с помощью различных алгоритмов машинного обучения. В нашем случае для каждой пары вопрос - студент нужно предсказать вероятность ответа на вопрос. Любая модель машинного обучения предсказывает целевую переменную на основе набора

признаков. В задаче отслеживания знаний список признаков не ограничивается каким-то номером вопроса и студента, их могут дополнить различные признаки основанные на статистиках временных рядов, разница моментов времени, векторы, являющиеся средними в кластерах, и многие другие. В разделе экспериментов будет описан наиболее успешный набор признаков. Модели, которые способны решать данную проблему, выписаны в списке ниже:

- Логистическая регрессия
- Дерево решений
- Случайный лес
- Градиентный бустинг
- Нейронная сеть
- Наивный байесовский классификатор
- Метод опорных векторов
- Метод ближайших соседей

1.2.2 Рекуррентные модели

В отличие от байесовской модели отслеживания знаний, которая предполагала, что состояние знаний на любом шаге зависит только от состояния предыдущего шага, рекуррентные нейронные сети способны эффективно улавливать сложность и разнообразие данных с течением времени и успешно предсказывать следующую партию вопросов. Кроме того, благодаря глубоким сетям, знания предметной области больше не стали обязательным требованием для моделирования (хотя это все еще огромное дополнительное преимущество). Рекуррентные нейронные сети являются лучшими в некоторых задач временных рядов - например, преобразования речи в текст, перевода и генерация подписей к изображениям, так как в этих областях доступно большое количество данных для обучения. Эти результаты говорят о том, что с ростом данных в задаче отслеживания знаний можно пытаться добиться большего успеха в улучшении качества моделей, если сформулировать задачу как новое приложение нейронных сетей. Рекуррентные сети способный сами улавливать взаимосвязи между вопросами и студентами. В качестве рекуррентной модели используют LSTM или GRU[3]. Эти модели хороши тем, что усваивают зависимости как на далеких, так и на коротких временных интервалах.

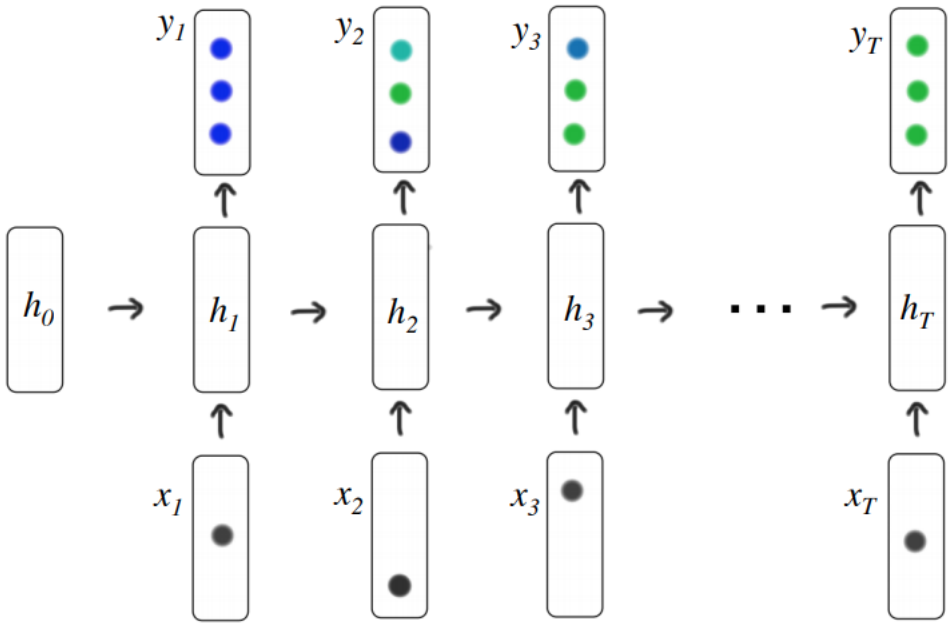

Рисунок 1.2 — Развертка рекуррентной сети

На Рисунке 1.2 изображена развертка рекуррентной нейронной сети. В качестве x_i на вход приходит набор признаков для пары конкретный студент и очередной вопрос, который ему задали. На выходе в качестве y_i может быть предсказанное распределение ответа на вопрос, вероятность правильного ответа, а также вероятность усвоения навыка. h_i - это внутреннее состояние нейронной сети, которое меняется в процессе обучения нейронной сети с целью минимизации функции потерь.

1.2.3 Отслеживание знаний при помощи трансформеров

До 2018 года в статьях, которые используют нейронные сети, вопросы кодировались с помощью One Hot Encoding, то есть каждый вопрос представлял из себя вектор 0, где по индексу id вопроса стояла 1. В последние два года из-за увеличения объема данных появилась возможность вместо One Hot обучать embeddings, также часто похожие вопросы, например по типу, представлялись одним вопросом перед получением эмбеддинга. Это делали для упрощения модели, которая генерирует вектор представления. Логично предположить, что чем больше данных, тем больше осмысленных эмбеддингов может выучить модель.

В статье[4] приводится пример Self Attentive Knowledge Tracing(SAKT) модели, которая основана на механизме внимания.

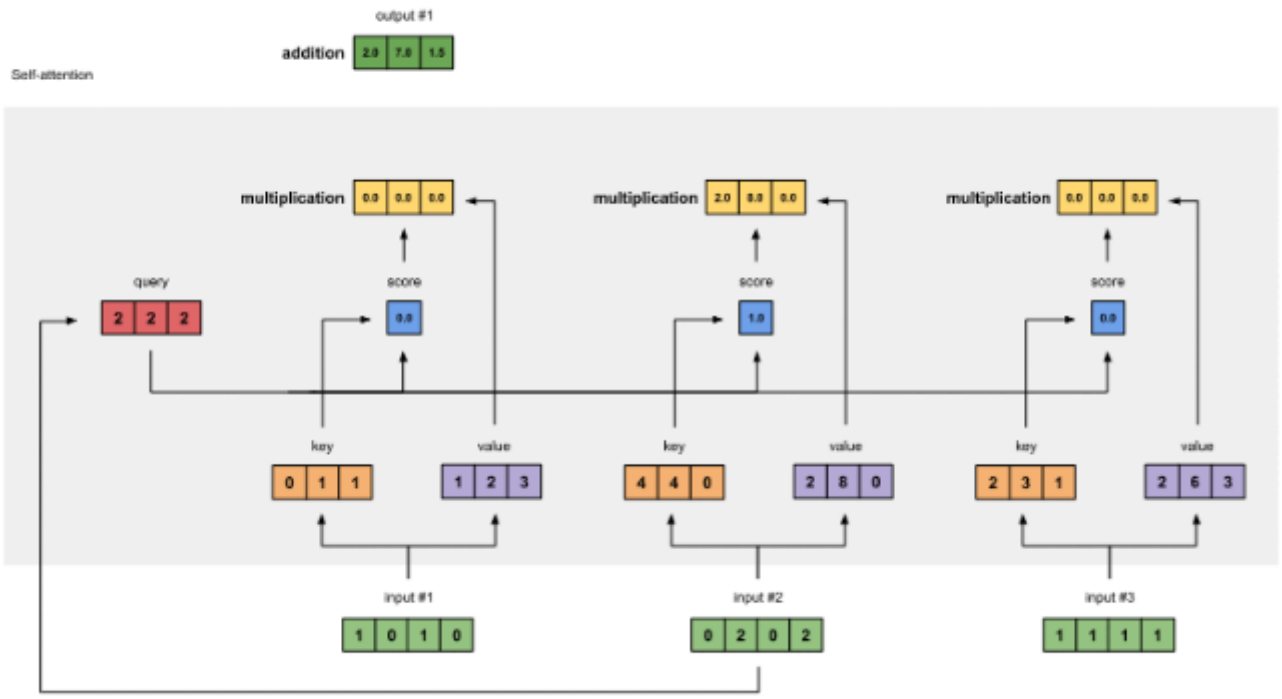

Рисунок 1.3 — Пример механизма self attention

Механизм Self Attention проще всего описать на основе рисунка 1.3. На вход поступает 3 вектора, это чаще всего эмбеддинги слов предложения. Из каждого эмбеддинга с помощью линейного слоя получается три вектора: query, key, value. Чтобы получить выходящий вектор для каждого слова, нужно query этого слова скалярно умножить на вектора value всех остальных, что и показано на рисунке. Далее используя полученные числа как веса, следует взвешенно просуммировать вектора value. Например у второго слова query * key равен 1, это значит, что мы учтем его value с максимальным весом, и это отразиться в результирующем векторе для первого слова.

Самая простая реализация модели включает в себя запрос, который состоит из векторов эмбеддингов предыдущих вопросов, включая текущий, для которого стоит предсказать правильность. В качестве ключа(key) и значения(value) используется взаимодействие, под взаимодействием можно понимать правильность ответа на предыдущие вопросы. Для упражнений и для взаимодействий используется свой слой для генерации эмбеддингов. Механизм внимание позволяет найти различные взаимосвязи между предыдущими вопросами и текущим, что улучшает качество модели. Помимо механизма внимание используется кодирование временной информации на основе принципа position-encodong в трансформерах, только вместо индекса в функции используется время ответа на вопросы. Также механизм self-attention применяются с несколькими головами, другими словами в модели присутствует

n параллельных self-attention слоев. Множество голов в модели помогают, так как пытаются найти взаимосвязи между вопросами в различных подпространствах. В задачах НЛП данный механизм позволяет улавливать различные зависимости между словами, например одна голова улавливает грамматическую основу, другая связывает предлоги и существительные. В контексте задачи отслеживания знаний, головы могут обращать внимание на типы вопросов, либо на разные интервалы времени. Данная модель сильно превосходит рекуррентные модели по качеству.

Глава 2
ДАННЫЕ

2.1 Описание данных

Данные были взяты из каггл соревнования и представляют собой вопросы и лекции в рамках Test of English for International Communication(TOEIC). Тест английского языка для международного общения - это международный стандартизированный тест на знание английского языка для лиц, не являющихся носителями языка. Он специально разработан для измерения повседневных навыков владения английским языком людей, работающих в международной среде. Существуют различные формы экзамена TOEIC, в нашем случае это TOEIC Listening Reading, который состоит из двух одинаково градуированных тестов оценки понимания деятельности на общую сумму возможных 990 баллов. В датасете 100 миллионов взаимодействий студента с вопросами или лекциями. Данные взаимодействия описаны тремя таблицами:

- train
- questions
- lectures

Опишем каждую таблицу подробнее.

2.1.1 Таблица train

Это основная таблица, которая связывает студента с вопросом или лекцией, с которыми он взаимодействует, а также дополнительная информация об этом взаимодействии. Список колонок и краткое их описание:

- **row_id**: (int64) порядковый номер строки в этой таблице.
- **timestamp**: (int64) время в миллисекундах между текущим взаимодействием и первым событием связанным с данным ученика.
- **user_id**: (int32) идентификационный код ученика.
- **content_id**: (int16) идентификационный код вопроса или лекции.
- **content_type_id**: (int8) 0 если это вопрос, 1 если лекция.

- **task_container_id**: (int16) идентификационный код группы вопросов, в которой задан текущий вопрос.
- **user_answer**: (int8) вариант ответа, который выбрал ученик. Всего вариантов может быть 4. В случае если взаимодействие - это лекция, а не вопрос, то данное поле равно -1.
- **answered_correctly**: (int8) правильно ли ученик ответил на вопрос. В случае, если взаимодействие - это лекция, а не вопрос, то данное поле равно -1. Данное поле нужно будет предсказывать.
- **prior_question_elapsed_time**: (float32) среднее время в миллисекундах, которое ученику потребовалось, чтобы ответить на все вопросы в предыдущей группе, игнорируя любые лекции между ними. Равен 0 для первой группы вопросов, заданной ученику.
- **prior_question_had_explanation**: (bool) смотрел ли ученик объяснение и правильный ответ на предыдущий вопрос.

2.1.2 Таблица questions

Таблица описывающая вопрос. Количество вопросов в отличии от количества учеников заранее зафиксировано, также задания повторно предлагаются различным ученикам. Список колонок:

- **question_id**: внешний ключ(foreign key) для связывание данной информации с таблицей train.
- **bundle_id**: код по которому вопросы объединяются в группы.
- **correct_answer**: правильный ответ на вопрос.
- **part**: один из семи типов вопросов TOEIC. Ниже будет краткое описание каждого
- **tags**: один или несколько кодов с тэгами для данного вопроса. Описание тэгов не представлено, но самих кодов достаточно для того, чтобы сделать кластеризацию.

Как видно данных про вопрос немного, но достаточно, чтобы кластеризовать вопросы по типу с помощью part, а также конкретно по смыслу через тэги.

Описание типов вопросов:

- **part 1**: выбрать наиболее подходящие утверждение на основе данной фотографии.

- **part 2**: прослушайте 3 ответа на вопрос, написанный на экране, выбрать лучший ответ.
- **part 3**: прослушайте диалог, выберите лучший ответ на вопрос, выведенный на экране.
- **part 4**: прослушайте монолог, выберите лучший ответ на вопрос, выведенный на экране.
- **part 5**: выбрать лучше продолжение предложения.
- **part 6**: выбрать лучшее продолжение текста.
- **part 7**: прочитайте несколько текстов, выберите правильные ответы на вопросы.

2.1.3 Таблица lections

Данная таблица аналогична таблице про вопросы, но она про лекции, перейдем к ее описанию:

- **lecture_id**: внешний ключ(foreign key) для связывание данной информации с таблицей train.
- **type_of**: краткое описание основной цели лекции.
- **part**: один из семи типов, который был в таблице с вопросами.
- **tag**: один тэг в качестве числа, который представляет лекцию, его значение не дано, также как и в вопросах.

2.2 Исследование данных

Перед тем как проводить эксперименты с моделями, стоит проанализировать данные.

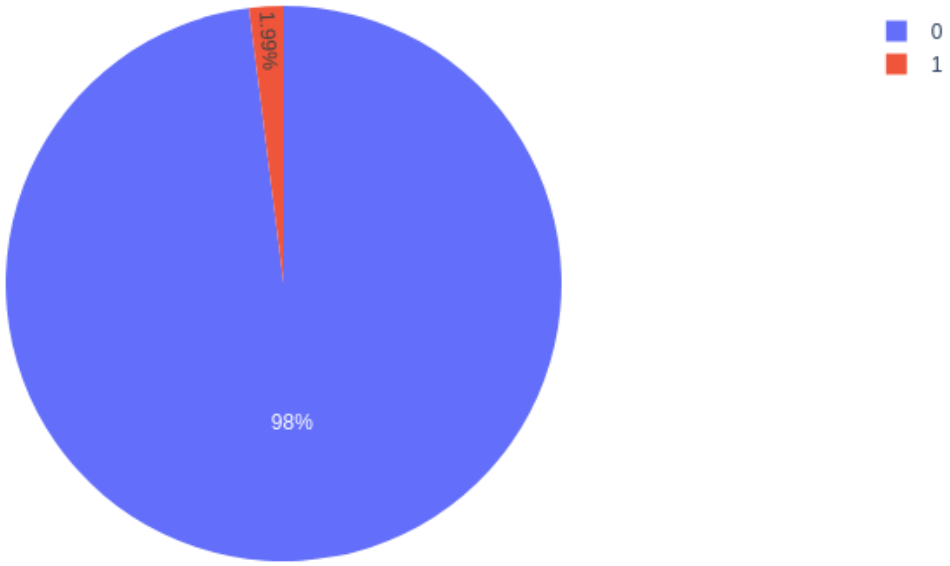

Рисунок 2.1 — Соотношение вопросов к лекциям

На рисунке 2.1 представлено отношение вопрос к лекциям. Вопросов 98 процентов, когда лекций всего 2. Нам нужно предсказывать правильный ответ на вопрос, следовательно преобладание вопросов в датасете будет скорее полезным.

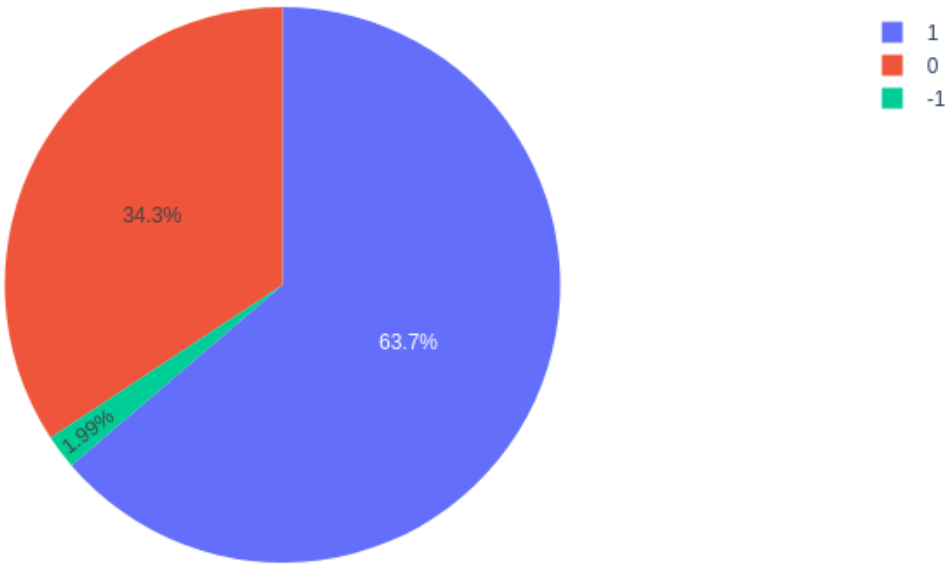

Рисунок 2.2 — Процент правильных ответов

На рисунке 2.1 можно заметить, что правильных ответов в 2 раза больше, чем не правильных. Лекция помечена как -1. Далее попробуем изобразить распределение вариантов ответов учеников.

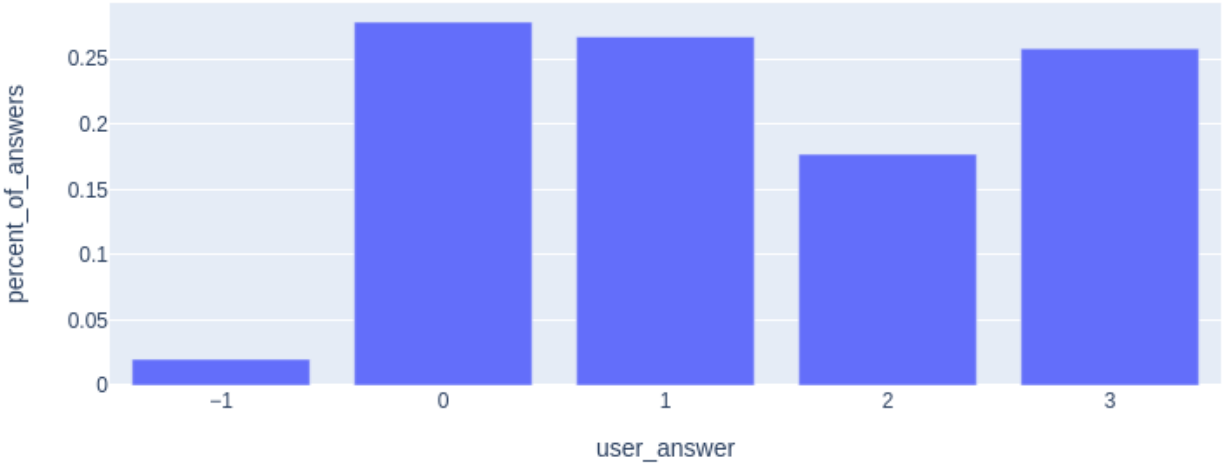

Рисунок 2.3 — Распределение вариантов ответов пользователей

Видно, что по какой-то причине вариант ответа 2 не такой популярный, можно будет попробовать сделать признак, основанный на этом знании.

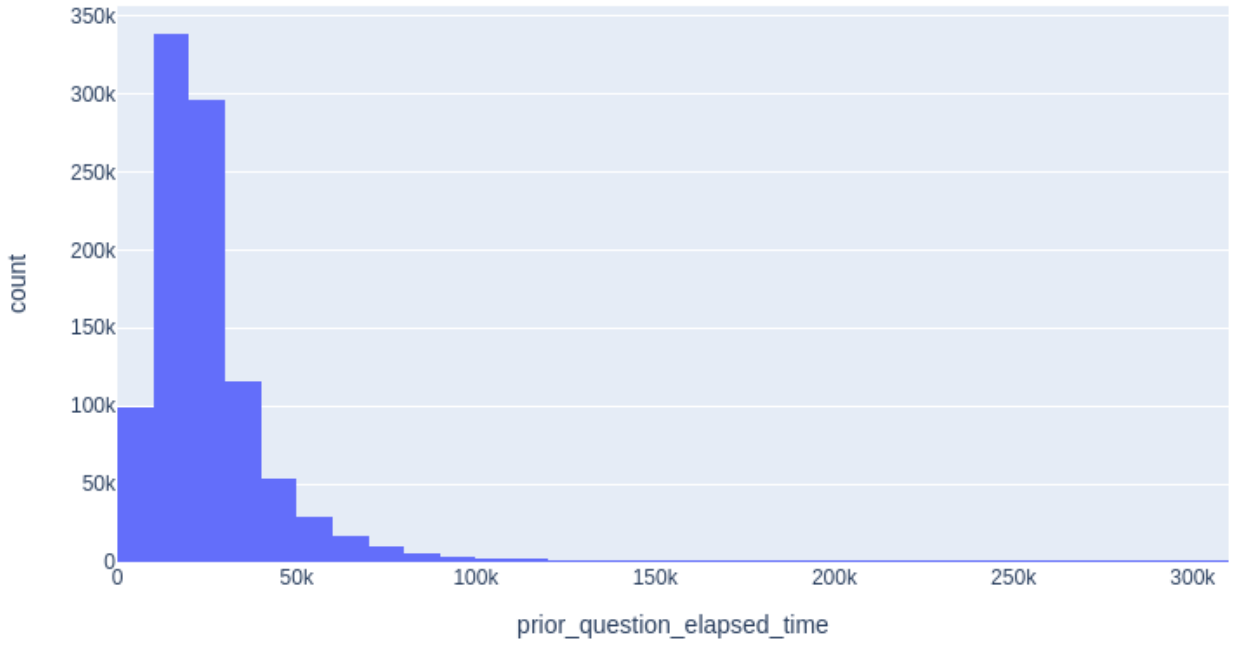

Рисунок 2.4 — Распределение среднего времени ответа на вопрос

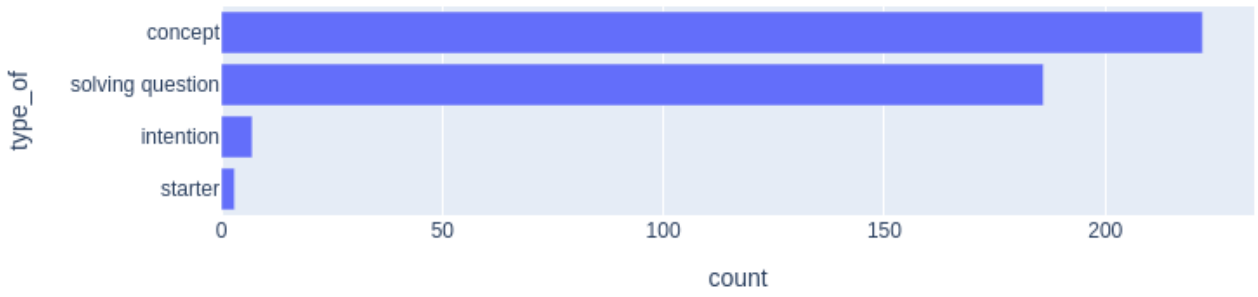

Рисунок 2.5 — Количество типов лекций

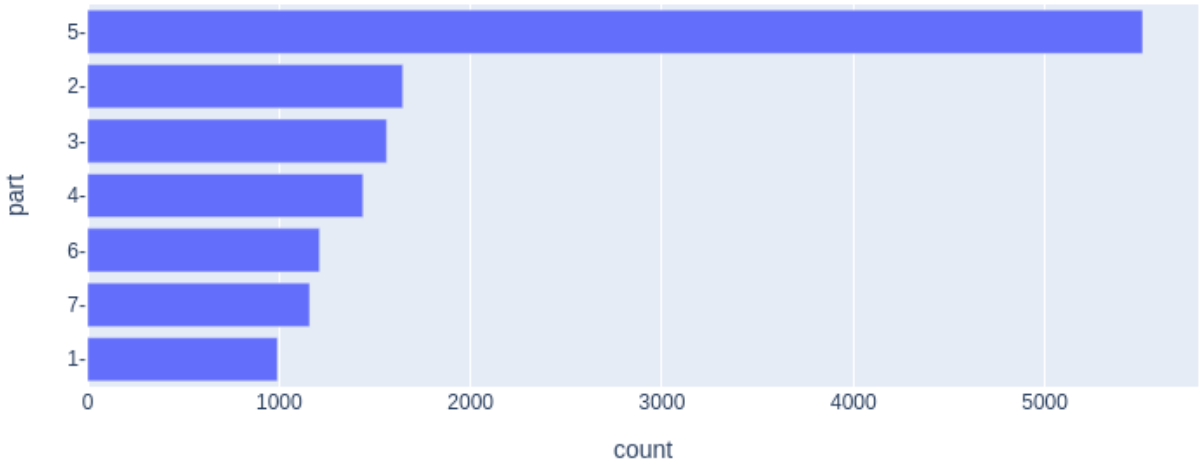

Рисунок 2.6 — Распределение темы вопросов

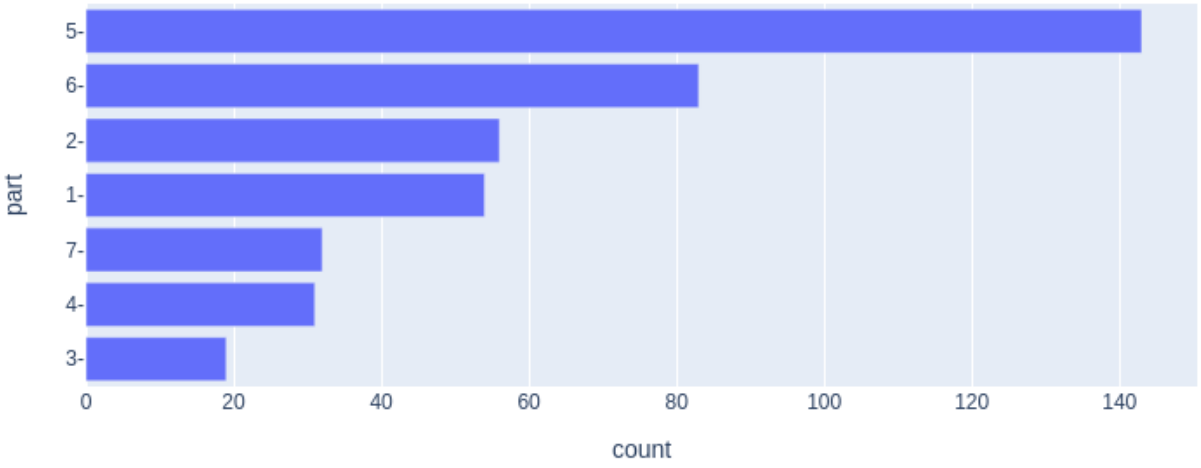

Рисунок 2.7 — Распределение тем у лекций

Сравнивая распределение тем лекций и вопросов, можно заметить что и там и там выделяется тема номер 5, а именно выбор лучшего продолжения предложения. Остальные темы распределены относительно равномерно. Хорошим знаком является то, что и в лекции и в вопросах одновременно преобладает 5 тема.

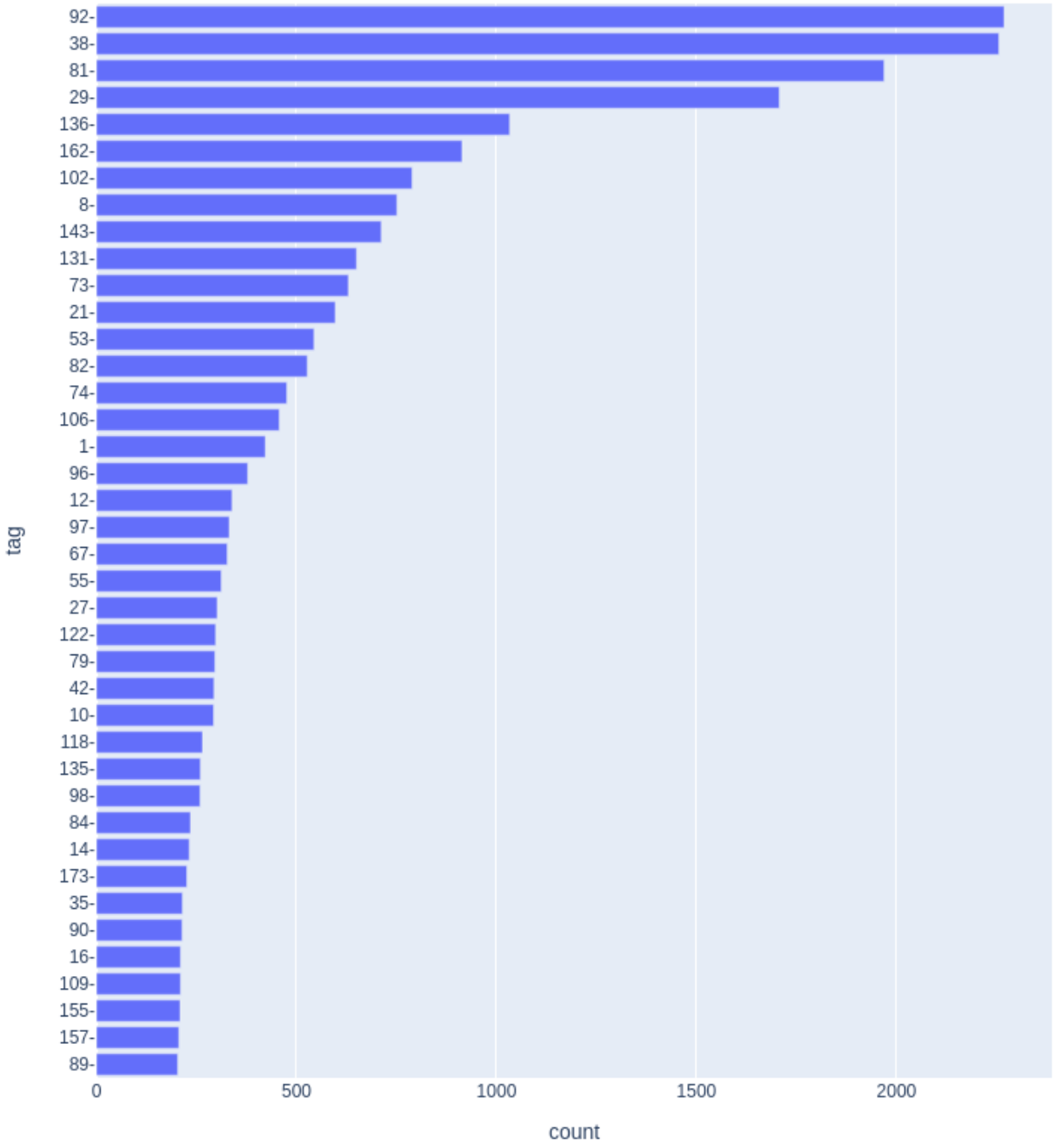

Рисунок 2.8 — Самые популярные тэги

Количество различных тэгов сильно больше, также видно что какие-то тэги выделяются на фоне остальных. Эти тэги входят в большую часть вопросов чем остальные, и скорее всего означают что-нибудь обобщенное, на-

пример, тип вопроса. Так как в датасете даны только идентификаторы тэгов, понять точный смысл не получится. Остается использовать числа в алгоритмах кластеризации или эмбеддингах.

2.3 Метрика

Для данной задачи в качестве основной метрики была выбора метрика AUC, на основе которой далее будут сравниваться все алгоритмы.

Глава 3
ГРАДИЕНТНЫЙ БУСТИНГ

3.1 Разбиение данных

Для обучения градиентного бустинга в первую очередь нужно выделить train и test из выборки. Для этого сгруппируем данные каждого ученика в группы, и разобьем эти группы на тренировочную и тестовую выборки в соотношение 95 : 5.

3.2 Признаки временных рядов

В каждой строчке датасета есть момент времени, по которому можно отсортировать последовательность взаимодействий для конкретного студента. Также из него можно извлечь большое количество признаков, например, способности учеников, основанные на истории. Данные из истории можно агрегировать, и на основе них строить статистические признаки:

- Последнее значение
- Количество
- Среднее значение
- Скользящее среднее
- Медиана
- Среднеквадратичное отклонение
- Коэффициент асимметрии

Реализовав класс, который умеет онлайн обновлять данные статистики, можно начать генерировать признаки на основе истории. Список признаков, по которым считаются статистики по времени:

- Время до предыдущего, до пред предыдущего вопроса.
- Правильный ответ на вопрос для каждого ученика. Данный признак показывает способность ученика.
- Правильный ответ на вопрос для каждого вопроса. Данный признак показывает сложность вопроса.

- Правильный ответ на вопрос для каждого типа вопроса(part в датасете).
- Правильный ответ на вопрос для каждого тэга вопроса(tags в датасете).
- Количество просмотренных лекций, количество лекций просмотренных по каждому тэгу.
- Смотрел ли ученик объяснение.(prior_question_had_explanation в датасете)
- Агрегация правильных ответов по часам в течении дня. Например ученик может лучше решать задания вечером, а не утром.
- Агрегация распределение ответов ученика(1 из 4). Возможно у него есть предпочтения к какому-то варианту ответа, когда он не знает правильный.

Данный набор признаков можно получить, итерируясь по каждой строчке и обновляя различные счетчики. Например, если в датасете у нас есть 5 вопросов, заданных одному ученику, то с точки зрения таблицы - это 5 строк с данными отсортированным по времени. Для вычисления всех признаков можно начать итерировать по этим вопросам, и во время итерации на i'ом вопросе сначала использовать все накопленные признаки с 0..i-1, а далее обновлять все счетчики с помощью i'ого вопроса и правильного ответа на этот вопрос. Тогда данные из будущего не смогу протечь в наши признаками, что очень плохо, так как в реальности у нас нет данных из будущего и модель может переобучиться на них.

В качестве модели был выбран градиентный бустинг с реализацией в библиотеке catboost. Параметры у модели были подобраны на основе поиска по сетке. Лучшие параметры:

- loss_function = Logloss;
- max_depth = 6;
- early_stopping_rounds = 100;
- metric_period = 20;
- iterations = 5000.

График обчуения приведен ниже, прерывистая линия - это AUC на трейне, сплошная - AUC на тесте.

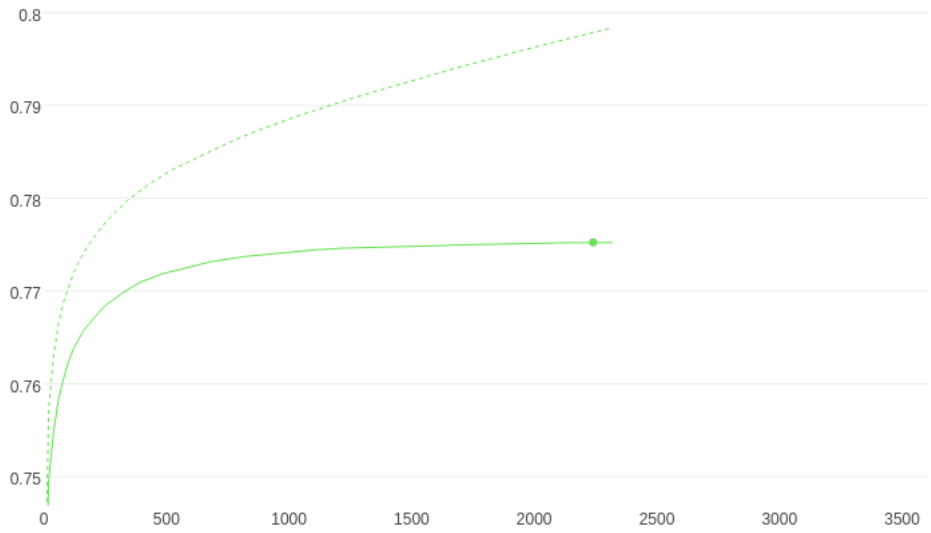

Рисунок 3.1 — Кривая обучения

Модель останавливает обучение на 2300 итерации, так как несколько десятков итераций подряд метрика на тесте начинает ухудшаться. Это прямой сигнал того, что начинается переобучение. Результирующий AUC равен 0.775

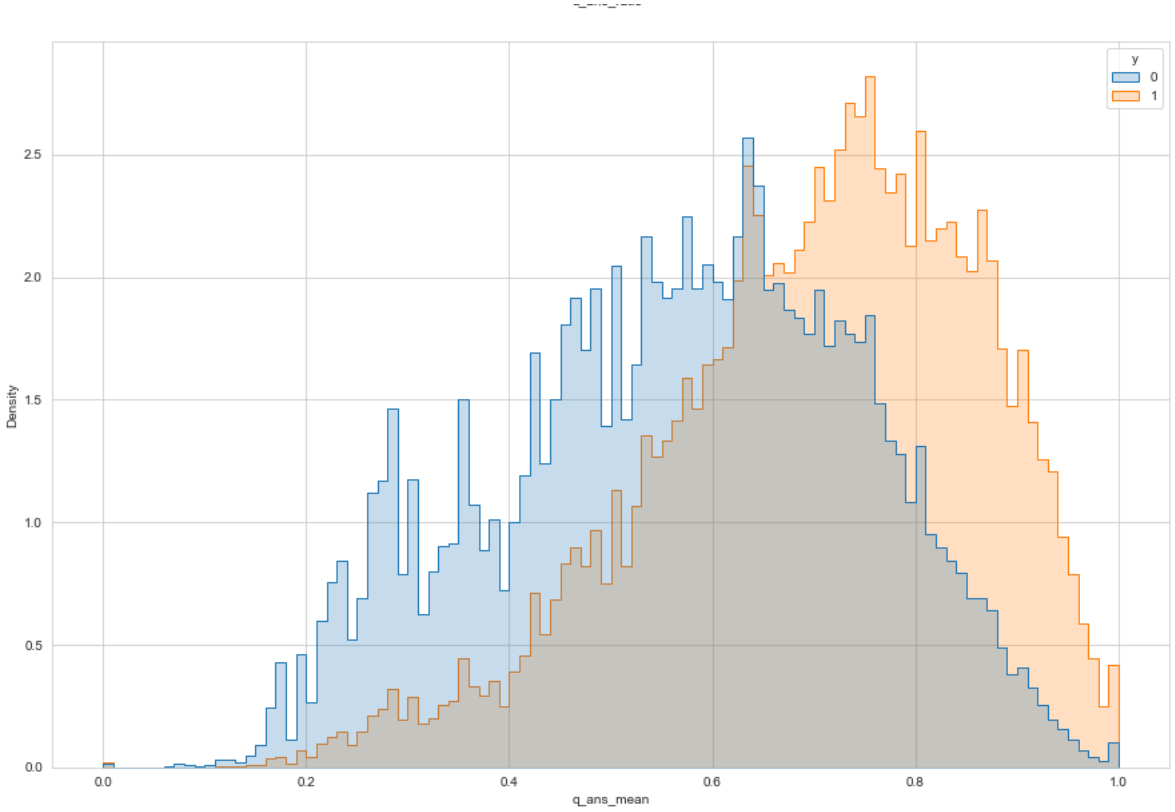

Рисунок 3.2 — Распределение признака среднее количество правильных ответов на заданный вопрос

На рисунке 3.2 можно заметить распределение признака среднее количество правильных ответов на заданный вопрос в зависимости от правильности ответа, который кодируется цветом. С помощью метода get_feature_importance была получена важность признаков в датасете, и данный признак является самым важным в нем.

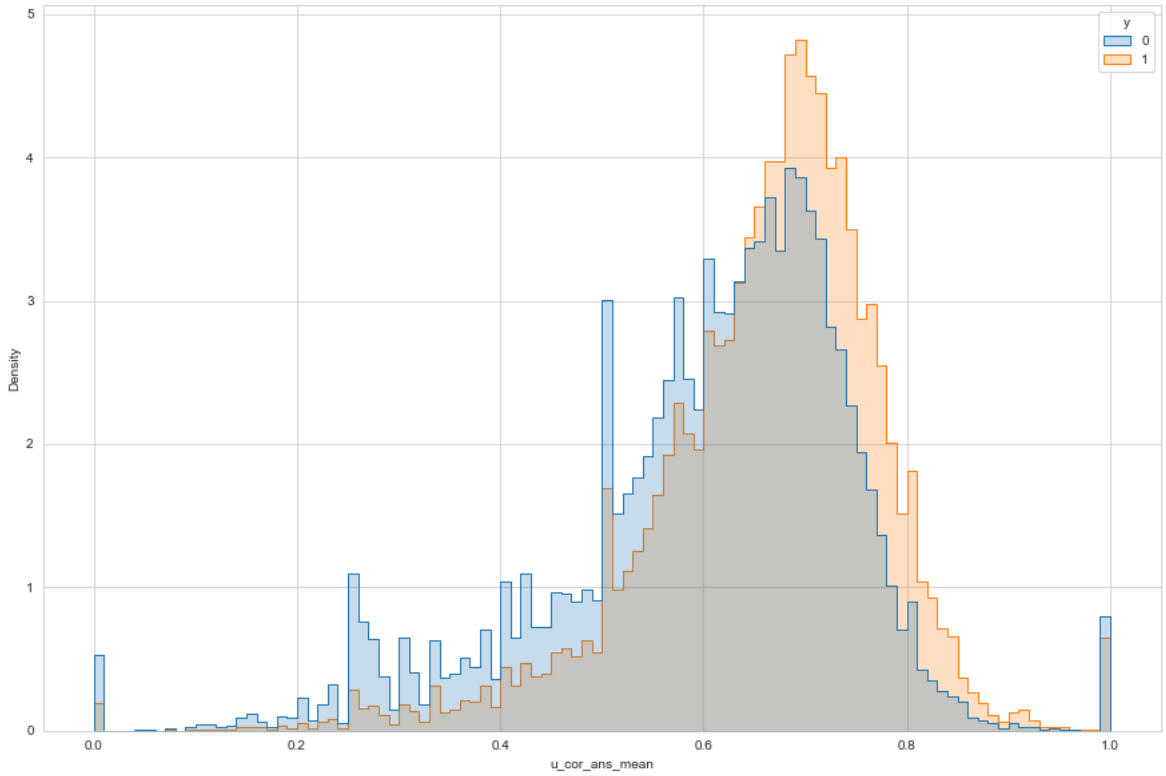

Рисунок 3.3 — Распределение признака среднее количество правильных ответов у ученика

Признак с рисунка 3.3 имеет второе место в списке важных признаков.

3.3 Эло рейтинг

Рейтинговая система Elo[5] - это метод расчета относительного уровня мастерства игроков в играх с нулевой суммой, например шахматы. Он так назван в честь создателя Арпада Эло. Данный рейтинг можно применять к задаче ответа на вопрос, где в качестве игроков выступают ученики и вопросы. Ученик побеждает в игре, если он отвечает на вопрос, проигрывает, если не отвечает. Ничьи в данной игре не бывает. Опишем подробнее, что из себя представляет рейтинг Эло.

Рейтинг Эло игрока представлен числом, которое может изменяться в зависимости от исхода сыгранных рейтинговых игр. После каждой игры выигравший игрок забирает очки у проигравшего. Разница между рейтингами победителя и проигравшего определяет общее количество очков, набранных или проигранных после игры. Если игрок с высоким рейтингом выигрывает у игрока с низким, то у игрока с низким рейтингом будет взято только малая часть рейтинговых очков. Однако, если игрок с более низким рейтингом выигрывает, то он за это получит много очков. Игрок с более низким рейтингом

также получает некоторое количество очков от игрока с более высоким рейтингом в случае ничьей. Это означает, что данная рейтинговая система само корректируется. Ниже будут приведены формулы, на основе которых вычисляется Эло рейтинг. Перед партией вычисляется ожидаемое количество очков, которое получит игрок A в партии с игроком B:

$$E_A = \frac{1}{1 + 10^{\frac{R_B - R_A}{400}}}$$

где:

- E_A - ожидаемое количество очков, которое наберет игрок A в партии с игроком B;
- R_A - текущий эло рейтинг игрока A;
- R_B - текущий эло рейтинг игрока B.

Аналогичная величина вычисляется для игрока B:

$$E_B = \frac{1}{1 + 10^{\frac{R_A - R_B}{400}}}$$

Другими словами E_A является вероятностью победы игрока A в предстоящей партии. После проведенной игры происходит обновление рейтинга по формуле:

$$R_{Anew} = R_A + K * (S_A - E_A)$$

где:

- R_{Anew} - новый эло рейтинг игрока A;
- K - это коэффициент, который можно подбирать на основе силы игроков, например, в шахматах он равен 10 для самых сильных игроков, 20 для средних, и 40 для новичков.
- S_A - 1 - если игрок A победил, 0 - если проиграл.

Аналогичная формула для игрока B:

$$R_{Bnew} = R_B + K * (S_B - E_B)$$

Теперь, используя данные формулы, их можно применить к нашей задаче. Все пользователи и вопросы по умолчанию получают рейтинг 1000. Далее каждый ответ на вопрос с точки зрения ЭЛО - это одна сыгранная игра. В датасет были добавлены следующие признаки:

- R_A;

- R_B;
- $R_A - R_B$;
- R_A / R_B;
- E_A;

Также как и с признаками временных рядов, эло можно считать не только для всех вопросов, но и отдельно для вопросов каждого типа(part) и каждого тэга. Данный рейтинг по сути будет означать, насколько ученик силен в какой-то из групп вопросов.

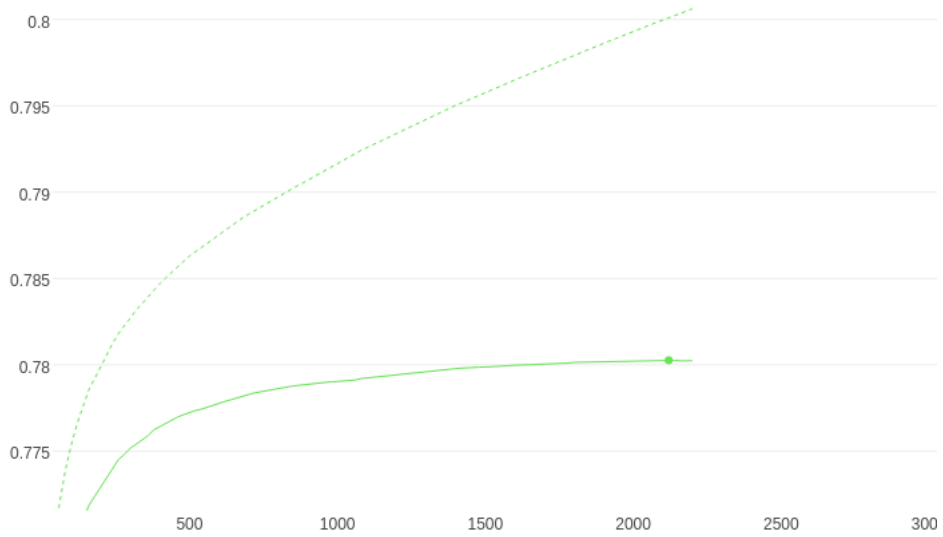

Рисунок 3.4 — Кривая обучения для датасета с признаками эло

После добавление признаков, основанных на Elo, качество модели значительно улучшается и AUC сходится к 0.78.

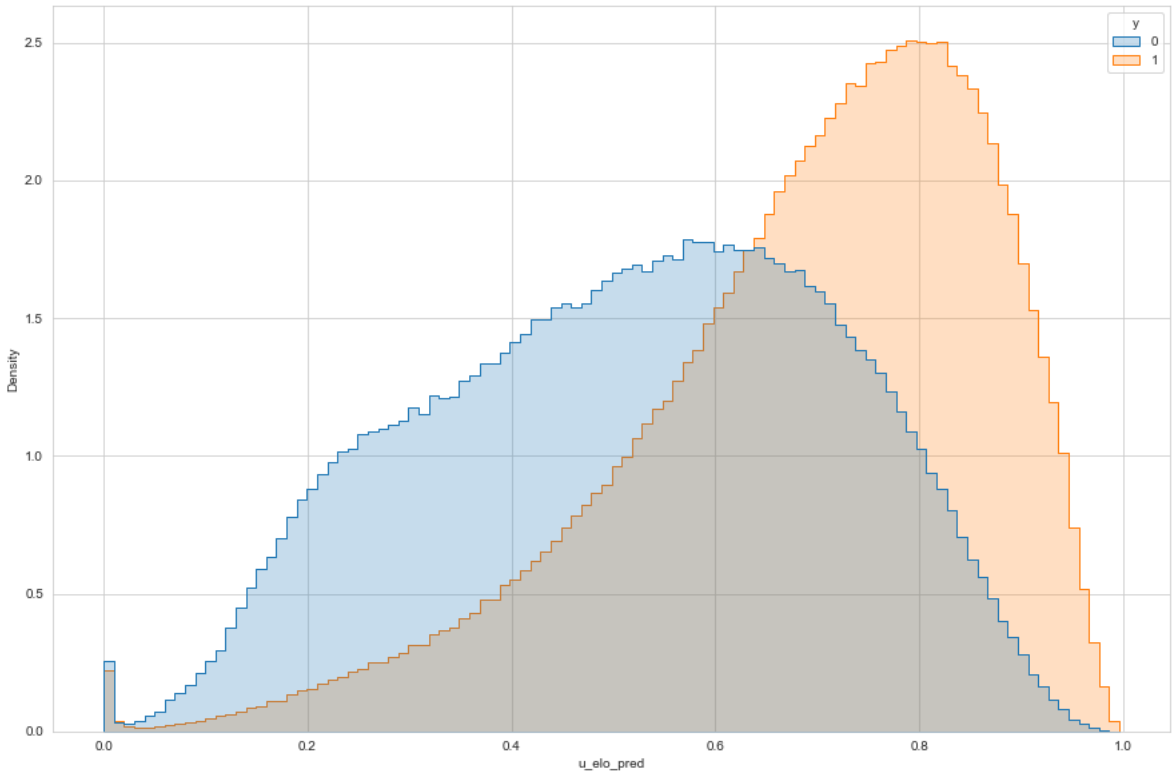

Рисунок 3.5 — Распределение предсказания эло

Глава 4
ТРАНСФОРМЕРЫ

4.1 Внутреннее внимание в нейронном отслеживании знаний

В данном разделе будут описаны эксперименты с нейронными сетями, основанными на теории из раздела Deep Knowledge Traci jng. Архитектуры нейронный сетей будут реализована с помощью библиотеки Pytorch на языке программирования Python. В качестве начального решения была реализована модель SAINT+[6]. Далее будут описаны улучшения для модели SAINT+, в конце данной главы будет приведена таблица с результатами экспериментов.

Внутреннее внимание в нейронном отслеживании знаний (Self-Attentive Neural Knowledge Tracing, далее SAINT) - нейронная сеть для отслеживания знаний, основанная на архитектуре transformer, а также содержит encoder и decoder.

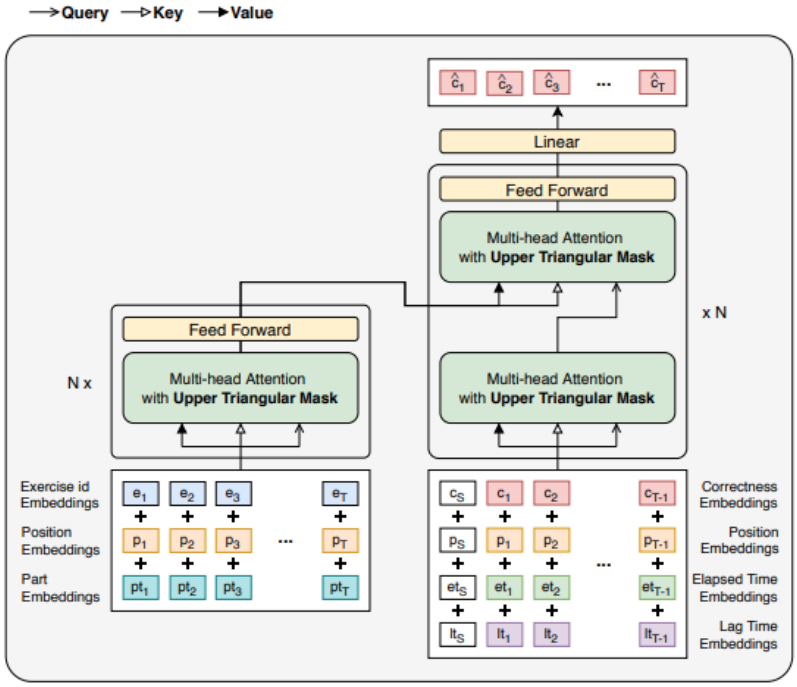

Рисунок 4.1 — Архитектура нейронной сети SAINT+

Объяснение модели можно разделить на две части: формирование входных последовательностей и архитектура сети.

Вход данной модели разбит на две составляющие: последовательность вопросов и последовательность правильных ответов. Encoder получает на вход последовательность размером T с эмбеддингами вопросов $E_{inp} = [E_1, E_2, ..., E_T]$ и выдает на выходе последовательность $O^{enc} = [O_1^{enc}, O_2^{enc}, ..., O_T^{enc}]$. Она в свою очередь подается на вход decoder, который дополнительно на вход получает последовательность эмбеддингов правильных ответов $D_{inp} = [S, D_1, ..., D_{T-1}]$. Данная последовательность также размером T, но она сдвинута право на 1, так что последний элемент теперь с индексом T - 1, а на месте первого теперь S - начальный токен последовательности. В ситуации, когда ученик еще ни разу не отвечал на вопросы, его входы для encoder и decoder будут выглядеть соответственно так: $E_{inp} = [S, S..., S, E_1]$ и $D_{inp} = [S, S, ..., S]$. Размер последовательности T - это параметр модели, чем больше это число, тем больше истории будет захватывать модель. В результате предсказывается вероятность ответа учеником на вопрос T.

Каждый элемент входной последовательности для encoder блока состоит из набора векторов одинаковой длинны:

- Эмбеддинг вопроса;
- Эмбеддинг позиции;
- Эмбеддинг типа вопроса;

Эмбеддинги суммируются в один общий вектор и получается один элемент последовательности E_i. Аналогично производится вычисление эмбеддингов для decoder блока:

- Эмбеддинг вопроса;
- Эмбеддинг того, правильно ли ответил ученик на вопрос;
- Эмбеддинг позиции;
- Эмбеддинг разницы времени между вопросами(lag time);

В результате суммы получается D_i. Разберем архитектуру модели, для этого опишем несколько слоев. Эмбеддинги подаются на вход multihead attention слою, в качестве query, value, key, которые получаются в результате умножения эмбеддингов на матрицы W^q, W^v, W^k соответственно.

$$Multihead(Q, V, K) = Concat(head_1, head_2, ..., head_h) * W_0$$

где h - это количество копий(голов) self attention слоя, W_0 - матрица весов, а head - это self attention слой.

$$head_i = Softamax(Mask(\frac{Q_i * K_i^T}{\sqrt{d}})) * V_i$$

где d - размерность векторов query, key и value, Mask - верхняя треугольная матрица, в которой все элементы выше диагонали заполнены $-\inf$, а ниже - единицей. Это сделано для того, чтобы первый элемент последовательности не мог рассматривать будущие элементы, так как на момент ответа на этот вопрос, будущие вопросы еще не были заданы.

Далее к multihead attention слою применяется feed forward network(FFN):

$$FFN(x) = ReLU(xW_1 + b_1) * W_2 + b_2$$

В результате блок энкодера выглядит следующим образом:

- $M^{enc} = LayerNorm(E + Multihead(E, E, E))$
- $O^{enc} = LayerNorm(M^{enc} + FFN(M^{enc}))$

здесь также используется LayerNorm[7] и skip connection[8], про которые можно подробнее узнать в литературе. Декодер можно описать следующим образом:

- $M_1^{dec} = LayerNorm(D + Multihead(D, D, D))$
- $M_2^{dec} = LayerNorm(M_1^{dec} + Multihead(M_1^{dec}, O^{enc}, O^{enc}))$
- $O^{dec} = LayerNorm(M_2^{dec} + FFN(M_2^{dec}))$

Выход O^{dec} передается в полносвязный слой, после применения которого предсказывается вероятность правильного ответа на каждый вопрос из последовательности, в том числе и для предыдущих.

В качестве лосса используется стандартная бинарная перекрестная энтропия для каждого выхода последовательности.

Все эксперименты будут проведены на 20% датасета или 20 миллионах обучающих примеров. В валидации будет 5 миллионов примеров никак не пересекающихся с трейном.

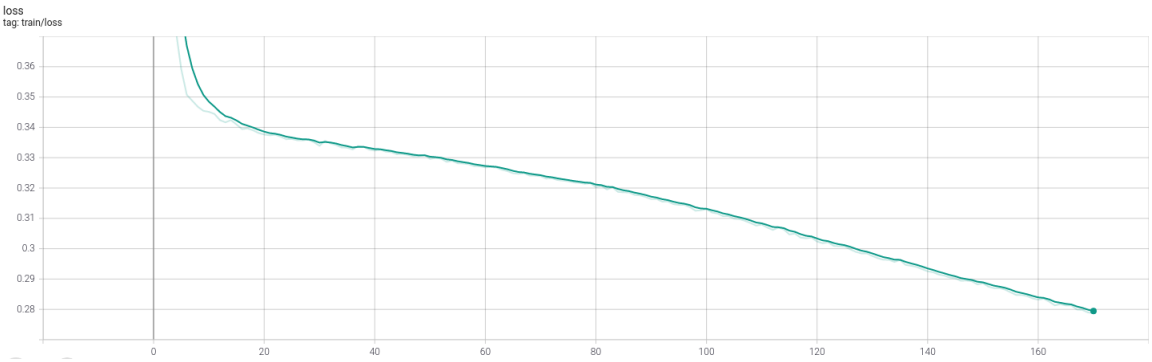

Рисунок 4.2 — Кривая изменения лосса модели SAINT+ на train данных

На графике выше изображена кривая изменения лосса на тренировочных данных. Лосс монотонно снижается, что и ожидается от нейронной сети во время обучения. Для того, чтобы понять когда стоит остановить обучения и предотвратить переобучение, следует смотреть изменение лосса на валидационных данных.

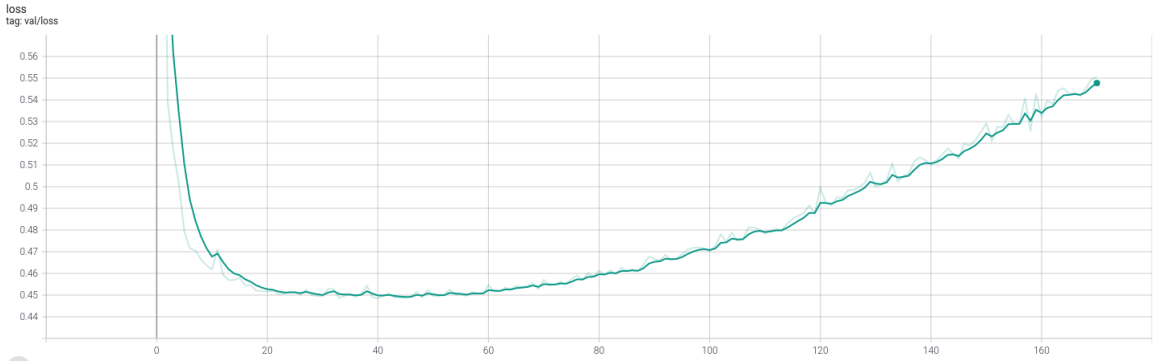

Рисунок 4.3 — Кривая изменения лосса модели SAINT+ на validation данных

По данному графику видно, что спустя 20 эпох валидационный лосс выходит на плато. После 60ой эпохи происходит явное ухудшение лосса, так как начинается переобучение модели. Между 20ой и 60ой эпохой находится плато, и так как оно расположено на довольно продолжительном участке в 40 эпох, между 20 и 60, сходу не ясно, в какую из эпох была получена модель с лучшими параметрами. Для того, чтобы выбрать лучшую модель следует использовать более чувствительную метрику - AUC.

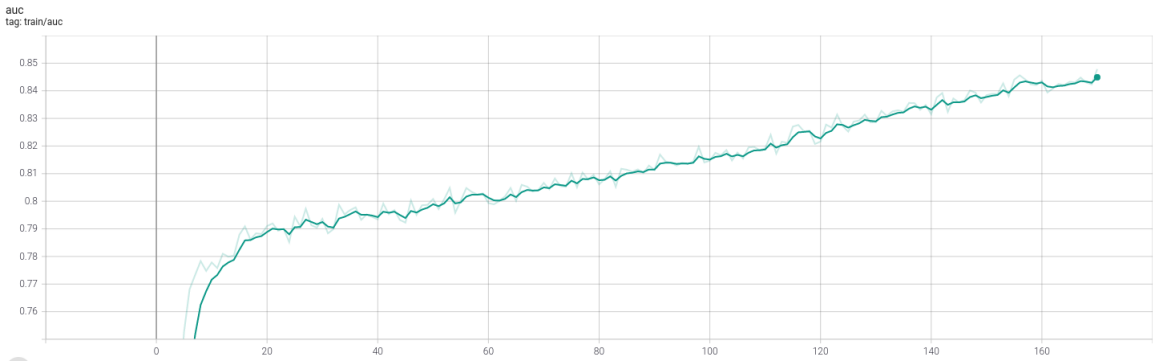

Рисунок 4.4 — Кривая изменения AUC модели SAINT+ на train данных

Кривая изменения AUC на трейне ожидаемо возрастает. Идеальный классификатор имеет AUC = 1.

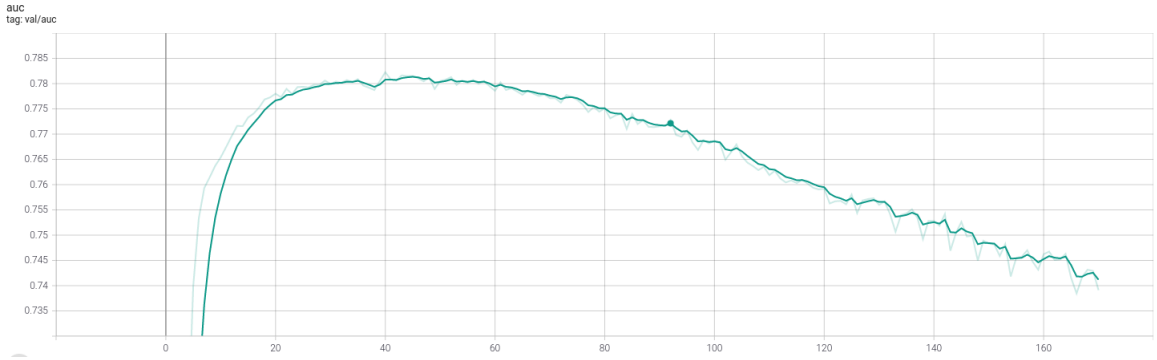

Рисунок 4.5 — Кривая изменения AUC модели SAINT+ на validation данных

Как и с кривой изменения лосса, на изменения AUC стоит смотреть на валидации. На данном графике видно, что лучшее значение AUC достигается на 45 эпохе и оно равно 0.7815.

Метрика AUC на валидации доказала, что является чувствительнее лосса, и поэтому в будущем будет использоваться для сравнения между собой различных версий модели.

4.2 Подготовка данных к обучению

Для обучения нейронной сети с self-attetntion архитектурой следует правильно подготовить данные.

Пусть T - размер окна, которое будет передано в self-attention. У каждого студента уже переменное число вопросов, на которые он отвечал. Поэтому стоит разбить последовательность вопросов у каждого студента на интервалы размером T. Например, последовательность вопросов $[q_1, q_2, ..., q_T, q_{T+1}, ..., q_{2T}, q2T + 1, ..]$ легко разбивается на интервалы [1, T], [T, 2T], [2T, 3T], ..., каждому из которых подставляется порядковый индекс. В случае если последовательность меньше, то она заполняется до T при помощи специального токена.

Далее применяем данный алгоритм для каждого студента и продолжаем индексировать данные интервалы:

1. Студент №1, интервал [1, T], индекс 1
2. Студент №1, интервал [T, 2T], индекс 2
3. Студент №2, интервал [1, T], индекс 3
4. Студент №3, интервал [1, T], индекс 4
5. и так далее

В библиотеке pytorch существует класс DataLoader, который работает с последовательностями и генерирует батчи нужного размера для входа в нейронную сеть. Так как мы на предыдущем шаге уже подготовили и проиндексировали данные, то не составит труда их обернуть в DataLoader.

Для лучшей сходимости модели стоит добавить аугментацию - изменение входных данных на новой эпохе обучения. В нашем несложно изменить входные последовательности, генерируя для каждой последовательности случайное число из равномерного распределения на интервале [-T / 2, T / 2] и сдвигая последовательность на это число.

Очевидно, что в каких-то ситуация, чтобы не выйти за пределы массива, нужно аккуратно формировать последовательность и потом ее сдвигать. Для простоты объяснения данный факт был опущен, но реализован в программе.

4.3 Внутреннее внимание с учетом времени

В отличии от задач нлп, в отслеживании знаний важным фактором является время, так как знания часто забываются, если их не закреплять повторениями. Отталкиваясь от этого свойства знаний, следует по максимуму использовать время в архитектуре сети. Для улучшения качества нашей модели следует использовать время в сердце нашей архитектуры - self-attention слое.

Напомним, что в self-attention на вход приходит последовательность векторов, обычно эти вектора в себе содержат как-то обработанный до этого набор факторов: эмбеддинги вопросов, positional encoding позиции. Далее каждый вектор с помощью трех линейных слоев превращается в key, query, value. После чего вычисляется выходящих вектор для каждого входа в последовательности по следующей формуле:

$$head_i = Softamax(Mask(\frac{Q_i * K_i^T}{\sqrt{d}})) * V_i$$

где d - размерность входящих векторов.

В этой формуле интересно подробнее рассмотреть веса внимания, с которыми взвешенно складываются value. Ниже приведена формула вычисления веса для фиксированного элемента i со всеми остальными элементами j для последовательности размером T:

$$a(i,j) = \frac{\exp s(i,j)}{\sum_{j'=0}^{T} \exp s(i,j')}$$

где $s(i,j) = \frac{q_i * k_j^T}{\sqrt{d}}$

Нетрудно заметить, что в $s(i,j)$ можно начать учитывать время следующим образом:

$$s(i,j) = \frac{\exp -d(i,j) * q_i * k_j^T}{\sqrt{d}}$$

где $d(i,j) = |t_i - t_j|$ - разность времени взятая по модулю.

В случае когда i и j совпадает $\exp -d(i,j) = 1$, чем больше разница между i-ым и j-ым временем, тем меньше $\exp -d(i,j)$. Таким образом веса далеких вопросов будут стремится к 0 и они никак не будут влиять на текущий.

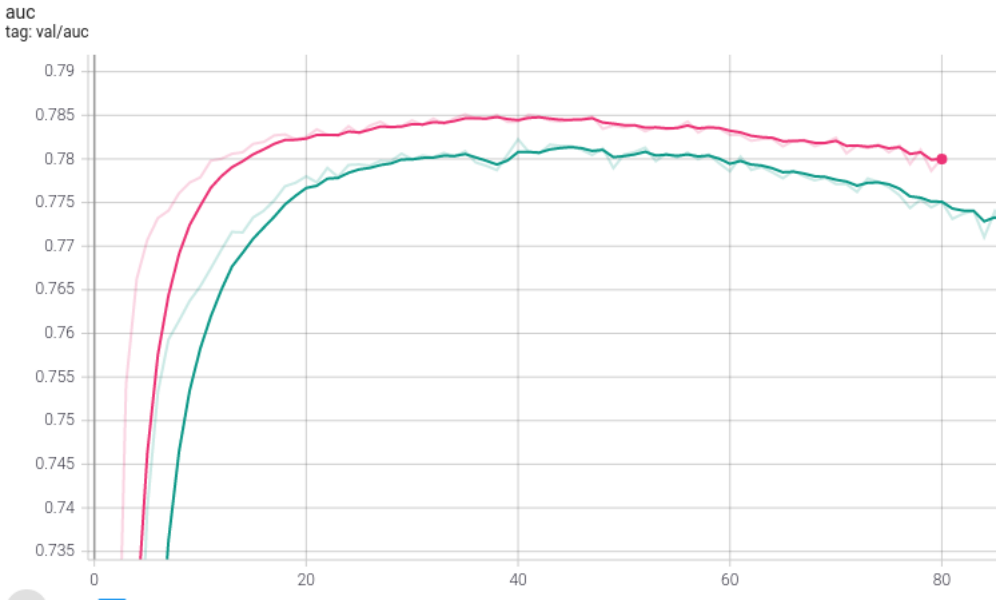

Рисунок 4.6 — Сравнение AUC. Зеленая: SAINT. Розовая: SAINT + Self-attention with time

Эксперименты показывают, что данный подход значимо увеличивает качество модели. Лучшее качество теперь AUC = 0.785 и достигается на 38ой эпохе.

4.4 Предварительная нормализация слоя

В классической transformer архитектуре используется следующий порядок вызова функций:

- $M = LayerNorm(E + Multihead(E,E,E)))$
- $O = LayerNorm(M + FFN(M))$

Основным недостатком данного подхода является то, что изначально вектора входных последовательностей могут быть плохо инициализированы, что может привести к нестабильному градиентному спуску, который легко может

улететь в бесконечность. Для того, чтобы это избежать предлагается оборачивать входы, а не выходы в функцию LayerNorm, которая нормализует вектора.

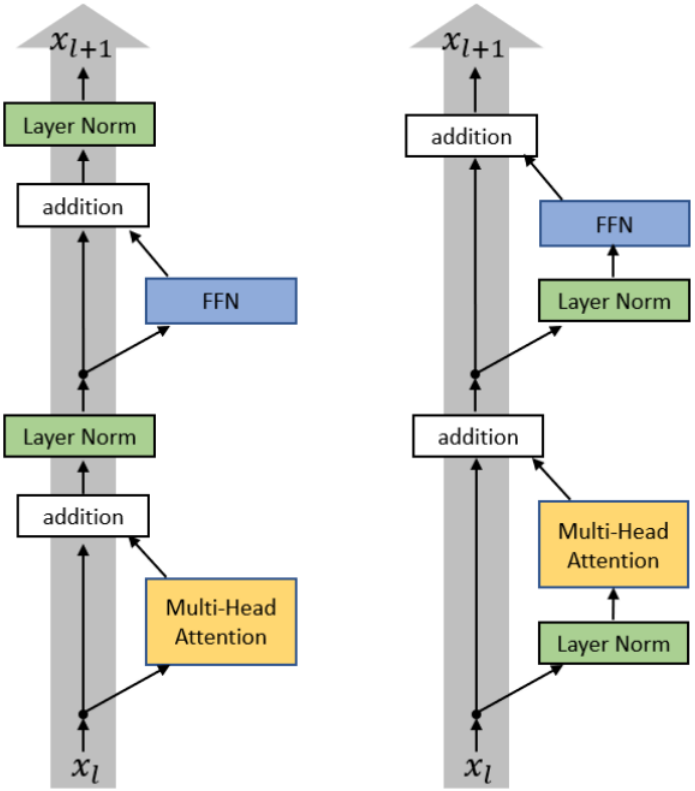

Рисунок 4.7 — Слева Post LN, справа Pre LN

По сути количество слоев не меняется, меняется только порядок, а именно порядок вызова слоя нормализации. В формулах это выглядит следующим образом:

- $M = E + Multihead(LayerNorm(E, E, E)))$
- $O = M + FFN(LayerNorm(M))$

Данный подход называется Pre Layer Norm Transformer[9], в отличии от предыдущего Post Layer Norm. Нормализация улучшает сходимость, что в свою очередь положительно сказывается на качестве модели.

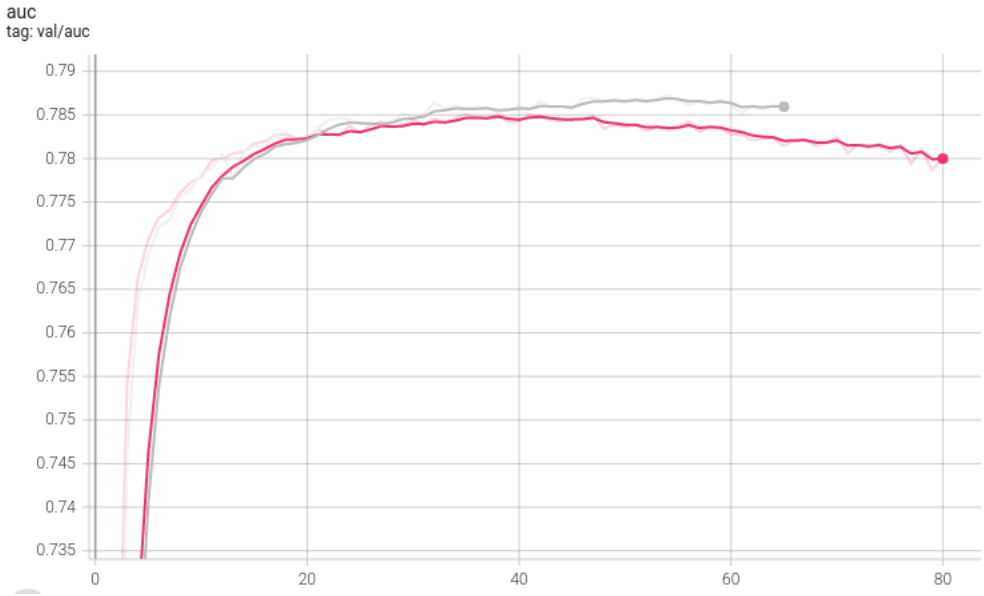

Рисунок 4.8 — Сравнение AUC. Розовая: SAINT + Self-attention with time. Серая: SAINT + Self-attention with time + Pre Layer Norm

Лучшее качество AUC = 0.7872, достигается на 55 эпохе.

4.5 Позиционное кодирование

Позиционное кодирование (Positional encoding) используется в нейронных сетях с архитектурой трансформеров. Трансформеры первоначально были придуманы для того, чтобы хорошо работать с задачами НЛП и должны были заменить рекуррентные сети. Рекуррентный слой шаг за шагом обрабатывает слова, и таким образом понимает порядок слов в предложении, что очень важно для понимания грамматики языка. В трансформерах слова обрабатываются одновременно и независимо, то есть там теряется порядок слов в предложении. Существует несколько способов передать информацию о порядке в модель. Все они завязаны на добавлении каких-либо чисел к существующим эмбеддингам слов.

Способы и их недостатки:

- Каждому слову назначать число из интервала [0, 1], таким образом первому слову будет соответствовать - 0, а последнему - 1. Недостатком данного подхода являются разные разности между последовательными числами для разных длин предложений, таким образом модели будет сложно вычислить порядок.

- Добавлять индекс слова, например, первому слову 1, второму 2, и так далее. В данном подходе недостатком является то, что модель на те-

стовых данных может получить на вход предложения длиннее, чем на трейне, и ей придется иметь дело с невиданными ранее числами.

- Для каждой позиции генерировать d-мерный вектор, содержащий информацию о конкретной позиции в предложении, где d - это размер эмбеддинга слова или в нашем случае вопроса. Данный вектор будет заполняться по следующим формулам:

$$PE(pos, 2i) = sin(\frac{pos}{10000^{2i/d}})$$

$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{2i/d}})$$

где pos - позиция слова в предложении или порядок заданного вопроса, i - индекс значения в векторе positional encoding. Последний подход не обладает недостатками из предыдущих пунктов. Легко показать, что расстояние между двумя временными шагами всегда одинаково, и это число не зависит от длинны последовательности. Каждая позиция имеет уникальный вектор и данная функция хорошо обобщается на большие длинны последовательностей, которые модель до этого не видела. Это возможно благодаря тому, что косинус и синус - ограниченные функции.

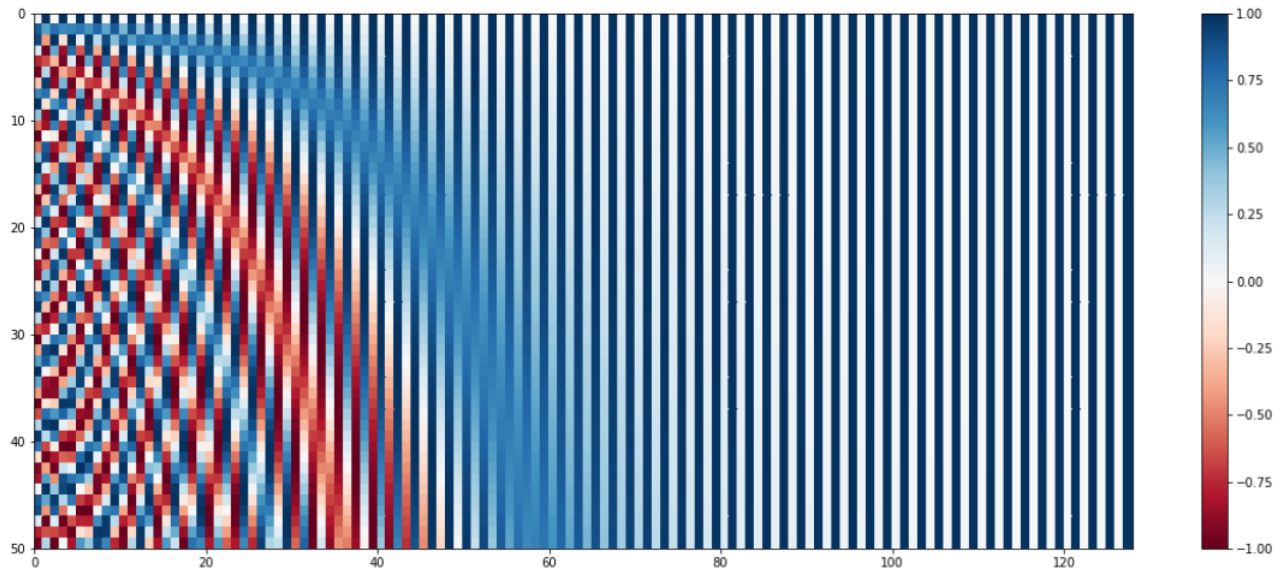

Рисунок 4.9 — Positional Encoding для пятидесяти позиций

На изображении 2.15 показаны закодированные 50 индексов(ось y) в вектора размером 128(ось x). Данное кодирование чем-то напоминает битовое кодирование чисел, только вместо того, чтобы для кодирования каждого элемента использовать 1 бит, используется вещественное число, что позволяет

более сжато кодировать информацию в нейронной сети, в силу того, что сети работают с вещественными числами.

Нетрудно заметить, что, помимо порядка, positional encoding способен кодировать непрерывные величины, которые никак не ограничены сверху. В задаче отслеживания данных такими данными является время, в частности разности времени между заданными вопросами для пользователя. Для этого нужно вместо позиции передавать значение дельт времени в функции, описанные выше.

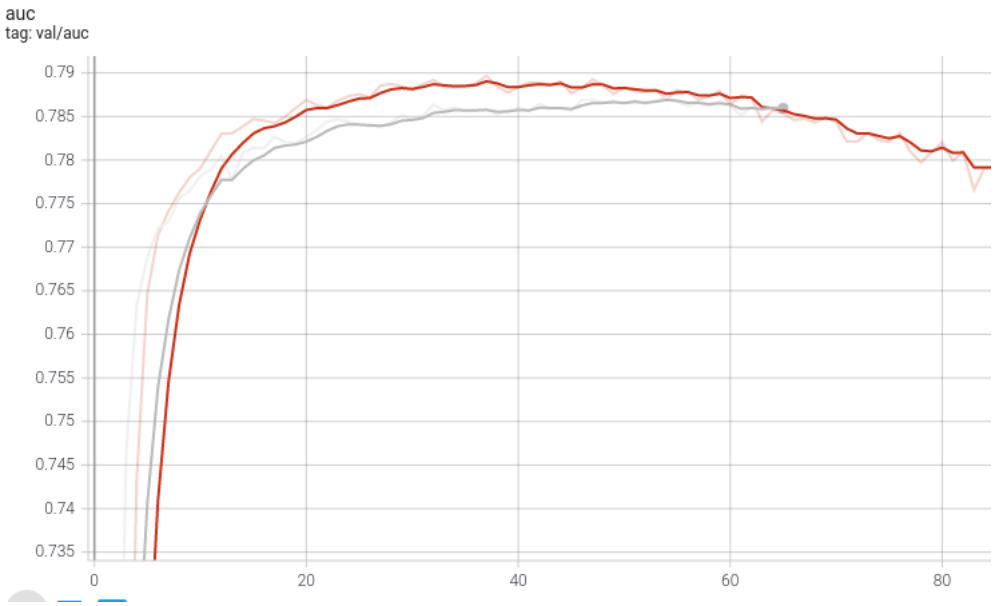

Рисунок 4.10 — Сравнение AUC. Серая: SAINT + Self-attention with time + Pre Layer Norm. Коричневая: SAINT + Self-attention with time + Pre Layer Norm + Time Positional Encoding

4.6 Дополнительные признаки

Можно вспомнить эксперименты с моделью градиентного бустинга и добавить следующие признаки в модель(название признака, тип, способ представление признака вектором):

- Среднее количество ответов на вопрос, double, positional encoding.
- Вероятность ответа на вопрос, double, positional encoding.
- Время до предпоследнего ответа, double, positional encoding.
- Тип вопроса(part), int, embedding.
- Просмотрено ли объяснение вопроса, int, embedding.

- Правильный ответ на вопрос, int, embedding.

Все признаки превращаются в вектора размерности d, далее складываются с векторами старых признаков и результирующий вектор размера d представляет один элемент последовательности.

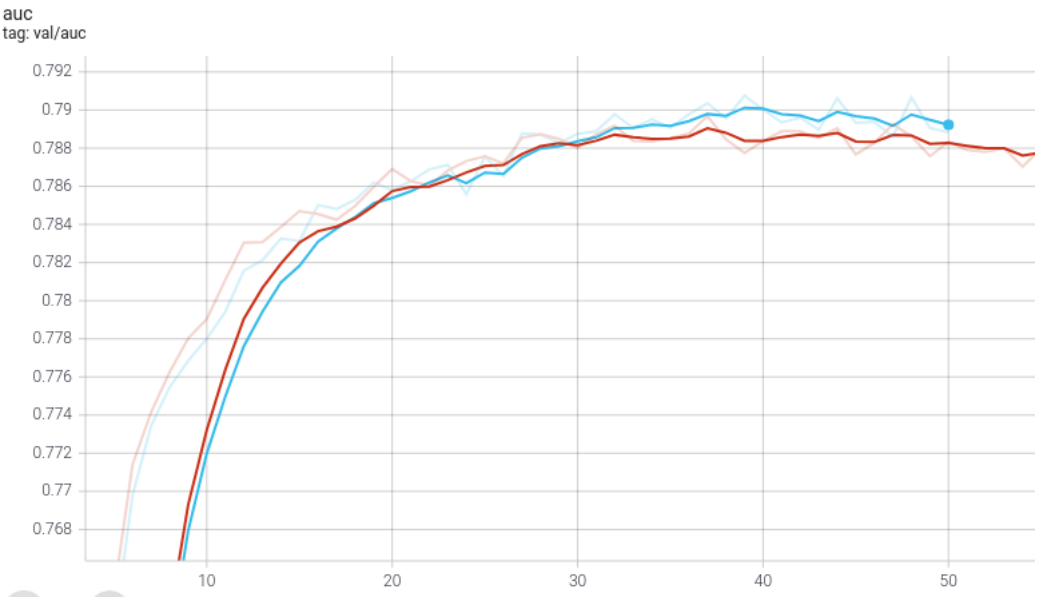

Рисунок 4.11 — Сравнение AUC. Коричневая: все предыдущие улучшения. Синяя: все предыдущие улучшения + дополнительные признаки

Добавление признаков также улучшает качество и теперь лучшее значение AUC = 0.7908 на сороковой эпохе.

4.7 Обучение на всем датасете

Как говорилось в начале данной главы, для ускорение экспериментов обучение предыдущих моделей производилось на датасете размером 20 миллионов, что равно 20% от всего датасета.

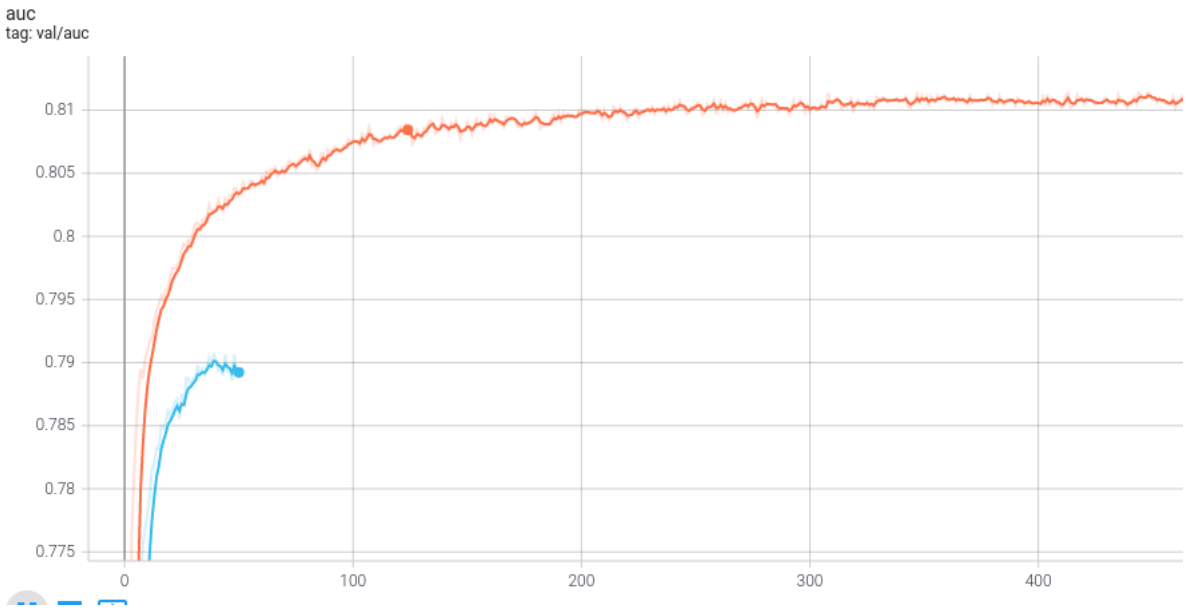

Рисунок 4.12 — Сравнение AUC. Синяя: лучшая модель, 20% датасета. Оранжевая: лучшая модель, 95% датасета

Лучшее значение AUC = 0.811, достигается на 400 эпохе. Модель обучалась 16 часов.

ЗАКЛЮЧЕНИЕ

Область отслеживания знаний в данное время находится в активной фазе развития. Это происходит из-за большого всплеска данных, которые в свою очередь позволяют успешно обучать нейронные сети с множеством параметров. В данной работе был проведен обзор материалов по различным методам машинного обучения, по предметной области, а именно по отслеживанию знаний. Также был произведен анализ исходных данных и подготовка их к применению методов машинного обучения. На основе анализа были реализованы признаки, позволяющие максимально улучшить качество модели, в том числе ЭЛО рейтинг.

Помимо классических моделей машинного обучения, была реализована модель трансформера - нейронная сеть, которая хорошо себя показывает в задачах НЛП. Были произведены улучшения нейронной сети, основываясь на специфике задачи отслеживания данных, ключевым фактором которой является время.

В результате проведения экспериментов были найдены улучшения архитектуры основанные на времени и особенностях датасета, что позволило получить приемлемое качество модели. Данную технологию можно будет использовать в качестве персонального контроля за учениками. Отслеживание их сильных сторон, рекомендации изучения теоретического материала в слабых темах, рекомендации заданий, которые соответствуют уровню знаний учащихся.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Knowledge tracing: Modeling the acquisition of procedural knowledge. / Corbett, A. T.; and Anderson, J. R. // ArXiv Prepr. ArXiv:2101.11335. – 1994.

2. Evaluating Forgetting Curves / Geoffrey R. Loftus // Journal of Experimental Psycholog - 1985.

3. Deep Learning / Ian Goodfellow, Yoshua Bengio, Aaron Courville // MIT Press – 2016. – 600 с;

4. A Self-Attentive model for Knowledge Tracing. / Shalini Pandey, George Karypis // ArXiv Prepr. ArXiv:1907.06837. – 2019.

5. Understanding and Pushing the Limits of the Elo Rating Algorithm. / JLeszek Szczecinski, Aymen Djebbi // arXiv preprint arXiv:1910.06081 - 2019.

6. SAINT+: Integrating Temporal Features for EdNet Correctness Prediction. / Dongmin Shin, Yugeun Shim, Hangyeol Yu, Seewoo Lee, Byungsoo Kim, Youngduck Choi // ArXiv Prepr. ArXiv:2010.12042. – 2020.

7. Layer normalization. / Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. // ArXiv Prepr. ArXiv:1607.06450 - 2016.

8. Deep residual learning for image recognition. /Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. // ArXiv Prepr. ArXiv:1512.03385 770–778. - 2016.

9. Глубокое обучение / Е. Архангельская, А. Кадурин, С. Николенко // Питер, 2020. – 480 с;

10. Foundations of Machine Learning / Mehryar Mohri, Afshin Rostamizadeh // MIT Press – 2018. – 750 с;

11. Bert: Pre-training of deep bidirectional transformers for language understanding. / Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. // arXiv preprint arXiv:1810.04805. - 2018