Санкт-Петербургский государственный университет

Математико-механический факультет

Группа 20Б.13-мм

Аушев Ислам Амарханович

Методы глубокого обучения в распознавании тематик статей

Отчёт по курсовой работе

Научный руководитель:
к.ф-м.н.
В.И. Гориховский

Санкт-Петербург
2023

Оглавление

1. Введение

Глубокое обучение - это подход в машинном обучении, который позволяет нейронным сетям обучаться на больших объемах данных. В отличие от традиционных методов машинного обучения, где признаки для классификации или регрессии задаются заранее и имеют четкую табличную структуру, в глубоком обучении модель может сама научиться извлекать нужные признаки из данных. Это достигается за счет использования многослойных нейронных сетей, которые могут обрабатывать сложные структуры и зависимости в данных. Сила нейронных сетей имеет свои побочные эффекты - зачастую, человеку тяжело проинтерпретировать работу многослойной нейронной сети, поэтому решения, которые принимает сеть, могут быть не до конца объяснимы человеком. Несмотря на это, нейронные сети показывают одни из лучших результатах во многих областях науки и промышленности.

Глубокое обучение применяется в различных областях, таких как компьютерное зрение(CV), рекомендательные системы(RecSys), обработка естественного языка(Natural Language Processing, NLP) и других. В рамках данной работы будет рассматриваться задача из NLP.

Natural Language Processing - область компьютерных наук на стыке машинного обучения и математической лингвистики. Она включает в себя такие задачи синтаксический и семантический анализ текста компьютером, машинный перевод, генерация текста и многое другое. Именно нейронные сети занимают лидирующие позиции в решении задач из области обработки естественного языка.

Главным отличием NLP от компьютерного зрения является модальность данных с которыми приходится работать. Основным отличием текстов от картинок является то, что слова в предложениях имеют четкий порядок, поэтому текст в некоторым смысле похож на временной ряд, в котором новые слова приходят последовательно, а замена порядка слов(или же знаков пунктуации) может полностью поменять весь смысл предложения(казнить нельзя помиловать). Это объясняет то, почему подходы основанные на сверточных сетях не показали хоро-

шего результата в задачах, связанных с текстами.

Все это привело к созданию новых архитектур нейронных сетей — рекуррентных нейронных сетей(RNN) и трансформеров, которые повсеместно используются в задачах NLP.

Также стоит обратить внимание, что в отличии от картинок тексты не так легко преобразовать в что-то понятное компьютеру. Если картинка в общем виде представляла собой многомерный тензор, то для текстов мы также хотим научиться строить векторные представления, которые потом будем подавать на вход нейронным сетям. Задача о построении качественных векторных представлений(эмбеддингов) для текстов крайне важна для дальнейшего обучения и валидации моделей.

В рамках данной работы будет рассмотрена задача о распознавании тематик научных статей по ее названию и описанию(abstract). Тематика статьи будет определяться согласно таксономии arxiv.org. Особенностью задачи является обширный набор возможных слов, встречаемых в статьях, а также двусмысленность терминов, ведь один и тот же термин может встречаться в разных областях науки. На примере этой задачи будут испробованы современные методы глубокого обучения.

2. Постановка задачи

В данной работе будет проведен обзор существующих методов глубокого обучения для решения задач Natural Language Processing, а также, основываясь на этих методах, будет предложено решение задачи о распознавании тематики статьи.

Цели работы:

1. Обзор существующих методов построения эмбеддингов текстов
2. Обзор существующих методов решения задач NLP, обзор архитектур RNN и трансформеров.
3. Анализ и выбор подходящей архитектуры для решения задачи распознавания тематик статей
4. Сбор данных для обучения
5. Построение и обучение соответствующей модели
6. Проверка эффективности работы модели

3. Обзор

Здесь будут рассмотрены основные базовые методы решения задач Natural Language Processing

3.1. Embeddings

Начнем обзор с методов построения эмбеддингов текстов. Под эмбеддингами понимается некоторое векторное представление исходного текста. А именно, хотим построить функцию f

$$f : V \to \mathbb{R}^d$$

где V - словарь всевозможных слов в задаче, d - размерность эмбеддингов (гипер-параметр). При этом, мы хотим, чтобы прослеживалась функциональная близость у семантически близких слов, то есть: слова близки по смыслу, тогда и только тогда, когда их векторые представления близки(например, в смысле L_2 нормы в $\mathbb{R}^d$).

3.1.1. One-hot

Самый примитивный подход, пусть у нас есть словарь всех слов $V = \{w_i\}$, каждому слову сопоставляем вектор $\{0, 1\}^{|V|}$, где 1 ставится только на позиции соответствующей индексу данного слова в словаре.

Текст размера K будет представлен матрицей $0, 1^{K \cdot |V|}$

У такого подхода есть много очевидных проблем, во-первых, мы получили очень большой размер представлений, ведь всего в слов в словаре может быть десятки, а то и сотни тысяч. Во-вторых, все представления получились ортогональными друг-другу, что мешает интерпретируемости расстояний между эмбеддингами.

3.1.2. Bag Of Words

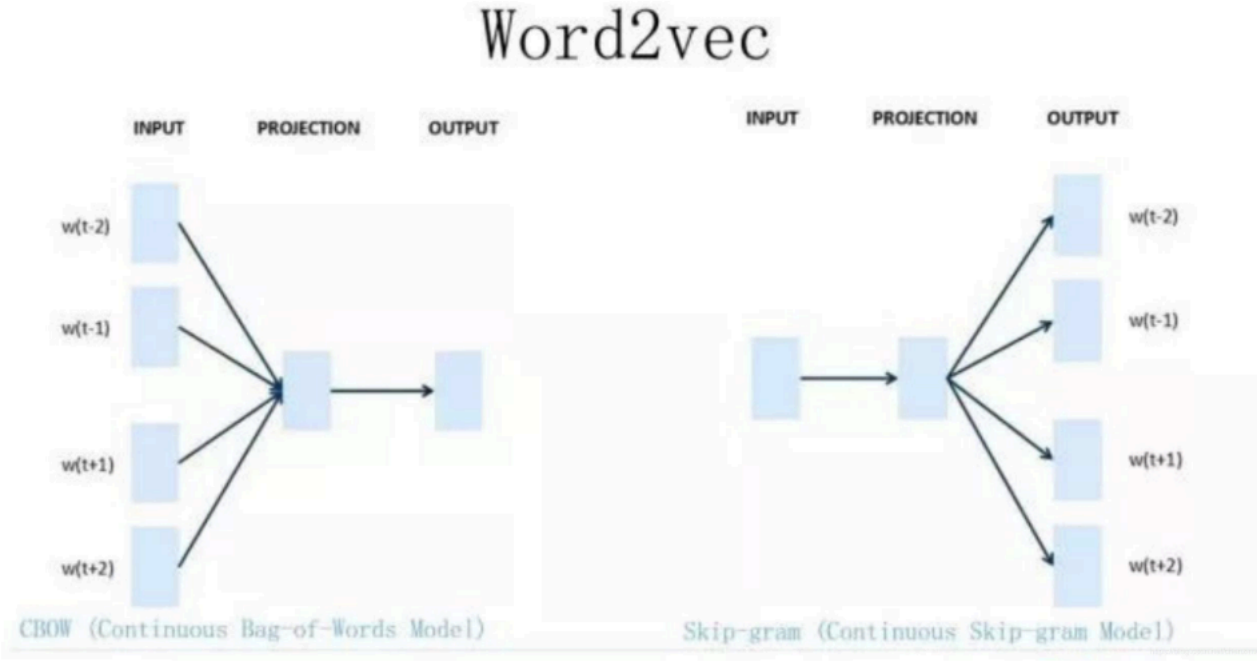
Вариант построения эмбеддингов для текстов — сумма one-hot векторов слов. Такой подход сокращает размерность эмбеддингов текстов

до $\mathbb{N}^{|V|}$, при этом предложения с похожим набором слов становятся похожи друг на друга. Минусы все такие же, слишком большие размерности представлений, при этом теряется весь порядок слов в предложениях.

3.1.3. Word2Vec

Word2Vec — метод построение эмбеддингов с помощью отдельных нейронных сетей. Идея Word2Vec заключается в в том, чтобы обучаться предсказывать связь между словом и несколькими словами в его ближайшем контексте(фиксированного размера окна).

Различают два вида Word2Vec, Continious Bag of Words — предсказываем слово по контексту из окна, и Skip-gram — предсказываем контекст по слову. Схематически это можно изобразить так



Word2Vec являются уже достаточно хорошими эмбеддингами, в том смысле, похожие по контекстам слова оказываются похожими в пространстве эмбеддингов. Особенностью Word2Vec эмбеддингов является возможность совершения векторных операций над эмбеддингами (сложение/вычитание эмбеддингов) без семантических потерь. Например, если рассмотреть эмбеддинги слов король, мужчина, женщина и коро-

лева, то будет приближенно верно следующее тождество

$$f(\text{король}) - f(\text{мужчина}) + f(\text{женщина}) = f(\text{королева})$$

Минусом Word2Vec является тот факт, что этот подход плохо справляется со словами, которых изначально не было в словаре. Если слова не было в словаре, то его эмбеддинги никак не обучались, поэтому никакой информации у Word2Vec для такого слова нет.

3.1.4. FastText

Модернизация Word2Vec, только здесь обучаем эмбеддинги не над словами, а над n-граммами слов. В такой интерпретации эмбеддингом слова будет сумма эмбеддингов его n-грамм.

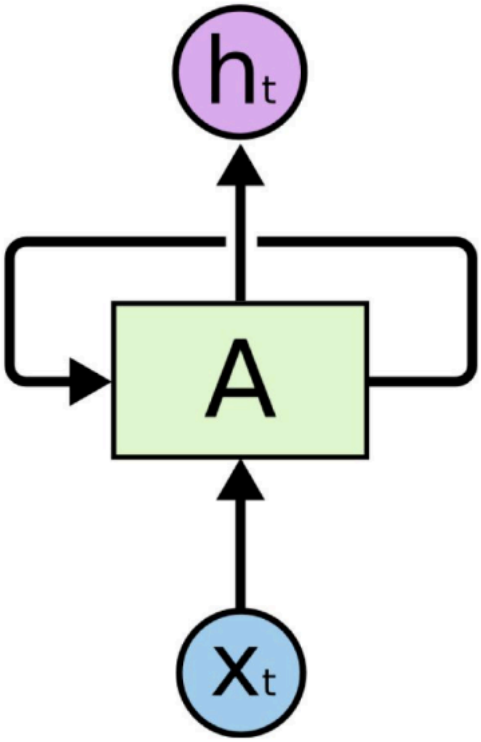
Такой метод борется с еще одним недостатком Word2Vec, для однокоренных или составных с общими частями слов эмбеддинги обучаются отдельно, никак не используя связь между этими словами. Подход с n-граммами пытается это исправить.

3.2. RNN

RNN(рекуррентная нейронная сеть) — вид нейронной сети, который используется для анализа последовательностей данных, таких как тексты, звуковые сигналы или временные ряды. Основная идея RNN заключается в том, что она имеет память, которая позволяет ей сохранять информацию о предыдущих состояниях и использовать ее для принятия решений в текущем состоянии. Это позволяет RNN обрабатывать данные, которые имеют зависимости во времени, и использовать эту информацию для прогнозирования будущих значений.

За открытием RNN стоит следующая интуиция : так как последовательности слов могут быть очень разной длины, и связанная информация внутри нее может быть расположена как угодно, то хочется иметь подход, который не зависит от размера некоторого окна, как это было в сверточных нейронных сетях, но при этом учитывающий связь вдоль времени.

Схематично работу RNN можно выразить так:



Здесь x_t - очередное слово из текста или предложения, слово подается на вход сети и преобразуется в выход h_t, слой A обладает памятью, которая будет обновляться и меняться при последовательной подаче в этот слой слов. В оригинальной версии RNN выполняется следующая формула для h_t:

$$h_t = tanh(h_{t-1}^\top W_{hh} + x_t^\top W_{xh})$$

где W_{hh} и W_{xh} - обучаемые веса. h_t еще называют внутренними состояниями сети. Добавив на выходе еще матрицу W_{hy}, получаем последовательность $\{y_i\}$ на выходе последовательности $\{x_i\}$ из RNN. Далее из выходной последовательности $\{y_i\}$ можно собирать агрегат, как предсказание брать последний выход y_{last}, или среднее/сумму/максимум по всем выходам.

Выпишем BackPropagation для RNN, получатся следующие формулы:

$$\frac{\partial h_t}{\partial W_{hh}} = \sum_{k=0}^{t} \frac{\partial h_t}{\partial h_k} \cdot \frac{\partial h_k}{\partial W_{hh}}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} tanh' \cdot W_{hh}^{t-k}$$

Из формул для BackPropagation видна основная проблема архитектур RNN — взрывы и затухания градиентов(если норма W_{hh} отлична от единицы). Бороться с взрывами градиентов можно с помощью gradient clipping, когда нормы градиентов обрезаются по какому-то заранее заданному порогу. Против затуханий можно использовать идею residual connections из resnet, тогда формула для h_t немного изменится

$$h_t = \tanh(h_{t-1}^\top W_{hh} + x_t^\top W_{xh}) + h_{t-1}$$

такой метод помогает бороться с затуханием градиентов, но он лишает нейронную сеть гибкости, потому что с каждым шагом RNN не сможет значительно изменять h_t, и все обучение будет сильно зависеть от начальных слов в предложениях.

Еще одним минусом RNN является то, что вычисления на RNN нельзя полноценно распараллелить на GPU, потому что данные имеют четкий порядок и слова должны друг за другом подаваться в нейронную сеть, поэтому рекуррентным нейронным сетям свойственно долго учиться.

3.3. LSTM

Архитектура LSTM пытается исправить проблемы, возникшие у RNN, а именно : возьмем RNN с residual connections и попробуем улучшить ее так, чтобы внутреннее состояние h_t можно было сильно менять с течением времени, при этом не сталкиваясь с проблемой взрывающихся и затухающих градиентов.

Работа LSTM основывается на попытке сделать память в слое более гибкой, достигается это с помощью так называемых гейтов(gates) памяти.

Введем дополнительное внутреннее состояние, отвечающее за память слоя, назовем его C_t, это состояние будем изменять на каждом шаге. Введем несколько гейтов, отвечающих за разные функции слоя

— очистка памяти, добавление в память, вывод на output.

Gate — функция с сигмоидой на выходе, каждый гейт принимает на вход текущий элемент последовательности x_t и выход предыдущей итерации h_{t-1}, каждый гейт отвечает за некоторое действие которое мы будем совершать с памятью.

На каждом шаге LSTM мы считаем прибавку к памяти от текущего шага, обозначим ее через $\hat{C}_t$

$$\hat{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

W_c и b_c — обучаемые параметры.

Определим гейты LSTM:

1. Forget gate — забыть ли содержимое памяти C_{t-1} на новом шаге?

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

   $[x_1, x_2]$ — конкатенация векторов

2. Input gate — добавить ли посчитанную на этой итерации прибавку к памяти $\hat{C}_t$?

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

3. Output gate — вывести ли содержимое памяти C_t наружу в h_t?

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

Общая формула для памяти C_t выглядит так

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t$$

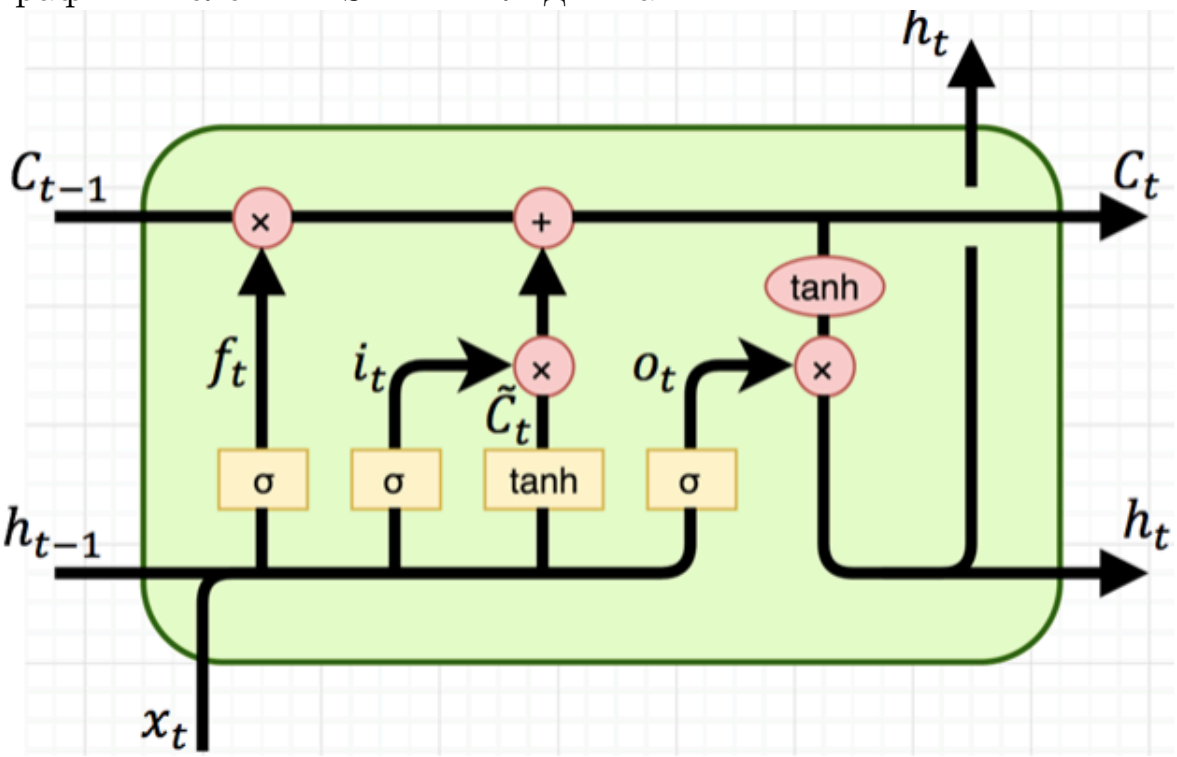
суть этой формулы легко проинтерпретировать, на каждом шаге мы получаем новую прибавку к памяти $\hat{C}_t$, теперь мы хотим понять, как много из этой памяти мы хотим использовать для изменения всей текущей памяти C_t, для этого мы умножаем скалярно i_t на $\hat{C}_t$. Здесь важно, что компоненты i_t принадлежат отрезку $[0, 1]$, поэтому, чем больше модуль компонент, тем больше информации мы хотим извлечь из $\hat{C}_t$ на

данном шаге. Далее мы умножаем скалярно f_t и C_{t-1}, это умножение показывает, сколько из текущей нашей памяти мы хотим забыть. Результирующая память складывается из двух этих компонент.

Когда мы посчитали память C_t на текущем шаге, мы можем высчитать выход на данном шаге с помощью последнего гейта

$$h_t = o_t * \tanh(C_t)$$

Граф вычислений LSTM выглядит так



Не выписывая общую формулу градиентов для LSTM, можно объяснить неформально объяснить тот факт, что у LSTM редко происходят затухания градиентов. Все сводится к тому, что градиент $\frac{\partial h_t}{\partial h_{t-1}}$ будет вычисляться через градиенты $\frac{\partial C_{t-1}}{\partial h_{t-1}}$ и $\frac{\partial \hat{C}_t}{\partial h_{t-1}}$, которые представляются как сумма градиентов по соответствующим гейтам. Сумма градиентов, в свою очередь, редко обращается в ноль.

3.4. GRU

Этот вид нейронных сетей появился сразу после LSTM, концептуально GRU и LSTM отличаются несильно. Часто говорят, что GRU это ослабленная версия LSTM.

Основные отличия GRU от LSTM:

1. GRU имеет только одно внутреннее состояние h_t(в LSTM мы хранили h_t и C_t).

2. Input и Output гейты объединены в один Update gate.

Гейты в GRU:

1. Reset gate — какие части h_{t-1} должны быть использованы для подсчета прибавки к памяти

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

2. Update gate — какие части h_{t-1} должны измениться

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

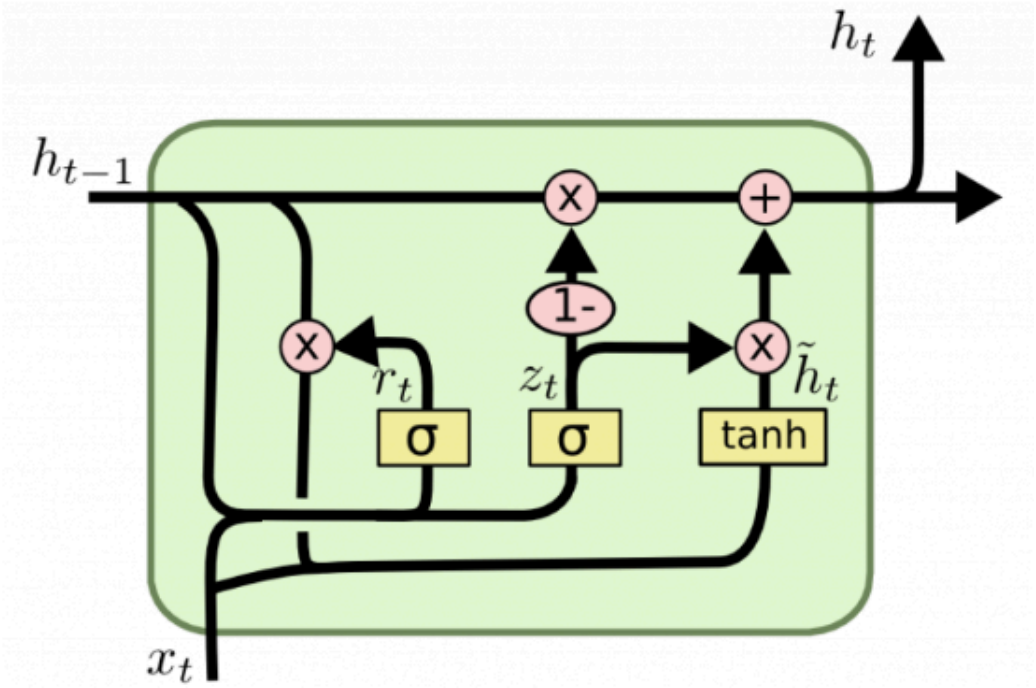
Прибавка к памяти вычисляется по формуле

$$\hat{h}_t = \tanh(W_h \cdot [r_t * h_{t-1}, x_t])$$

Обновление памяти

$$h_t = (1 - z_t) * h_{t-1} + z_t * \hat{h}_t$$

Граф вычислений GRU

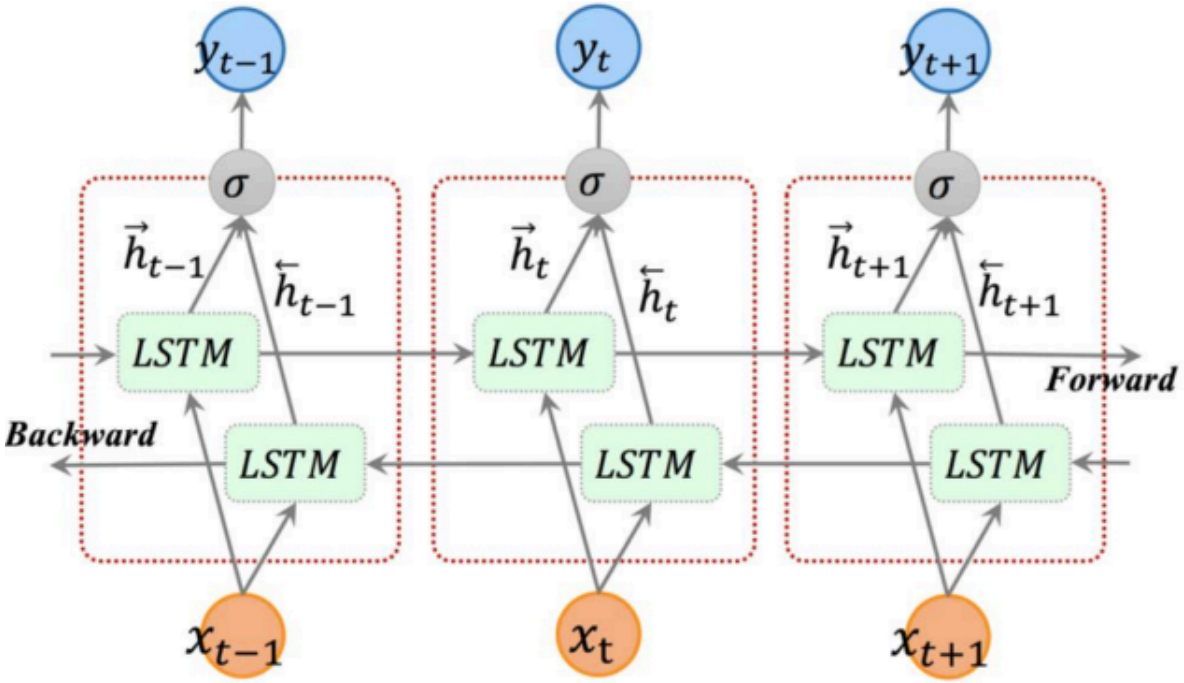


3.5. Bidirectional RNN

Работа всех нейронных сетей рассмотренных ранее построена на последовательной обработке входных слов. При этом мы считаем, что это последовательность имеет четкое направление, то есть, если мы обрабатывает какое-то предложение в тексте, то все слова в предложении проходят через нейронную сеть в строгом порядке слева-направо. Однако, иногда ключевая информация в предложениях может находиться позже анализируемой сейчас, из-за этого некоторый смысл нейронная сеть может не улавливать.

Чтобы исправить этот эффект часто используют bidirectional подход в RNN. Создаются два отдельных рекуррентных слоя, первый слой анализирует предложения в порядке слева-направо, второй слой в порядке справа-налево, далее выходы этих двух слоев как-то агрегируются(зачастую, их просто конкатенируют). Такой подход может быть использован также с LSTM или GRU.

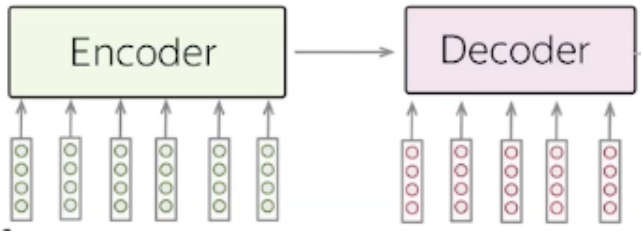
Схематично bidirectional сети выглядят следующим образом



3.6. Трансформеры

3.6.1. Механизм внимания

Основные минусы RNN нейронных сетей можно проследить на примере задачи машинного перевода, основной pipeline в этой задаче имеет следующий вид: используются две отдельные рекуррентные сети, первую будем называть encoder, другую decoder. На вход нейронной сети encoder поступает сначало предложение на языке X, каждое слово этого предложения encoder обрабатывает и получает на выходе некоторый результирующий вектор h_{last}. Получив h_{last}, теперь сеть decoder начинает один за другим предсказывать следующее слово, тем самым получая перевод исходного предложения с языка X на язык Y.



Здесь зеленые векторы — эмбеддинги слов в тексте на языке X. Красные — предсказываемые эмбеддинги слов на языке Y.

Данный подход неплохо себя показывал на практике, однако, у него была одна проблема, такой подход плохо работал с текстами большой длины. Объясняется это тем, что когда мы формируем эмбеддинги слов, мы задаем фиксированную размерность этим эмбеддингам, которая в дальнейшем никак не изменяется. После работы сети encoder мы также всегда получаем один вектор фиксированной размерности, независящей от длины самого текста. Это приводит к тому, что рекурентной нейронной сети становится сложно ужать всю информацию об исходном тексте в один вектор, какая-то часть информации неизбежно затирается с очередным словом в тексте.

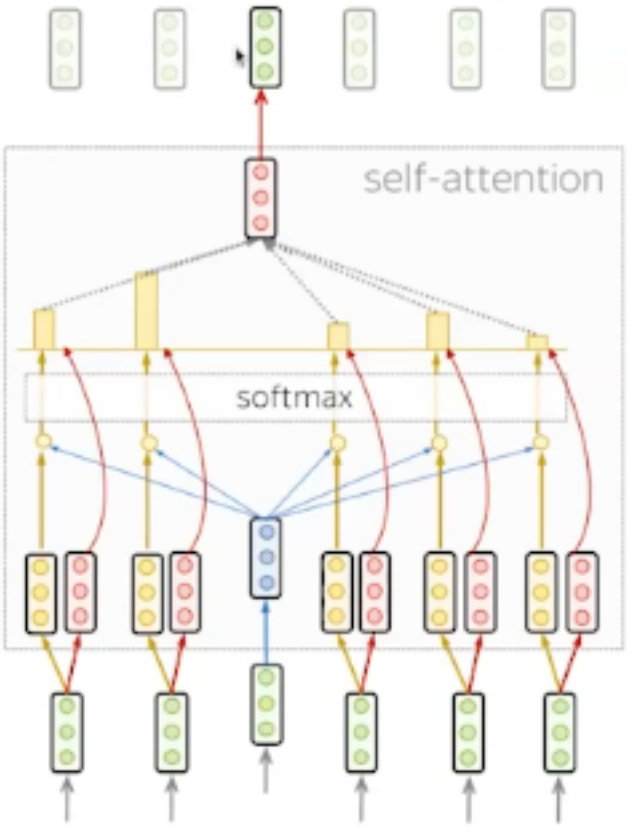
Пытаясь исправить эту проблему, был придуман так называемый механизм внимания(Attention[1]). Его интуиция заключается в следующем: вместо того, чтобы подавать в decoder результирующий выход сети encoder, попробуем научить decoder смотреть в любые промежуточные состояния encoder и извлекать из них информацию.

Опишем работу механизма внимания для задачи машинного перевода. Пусть в момент t RNN decoder имеет внутреннее состояние d_{t-1}, по этому внутреннему состоянию мы хотим получить следующее состояние d_t. Мы смотрим во все состояния $\{e_l\}$ сети encoder и считаем попарно некоторую величину score(d_t, e_l). Как именно здесь задавать функцию score не так важно, важно лишь правильно интерпретировать ее выходы, а именно, если $\text{score}(d_t, e_s) \geqslant \text{score}(d_t, e_p)$, то decoder больше предпочитает посмотреть в состояние s, чем состояние p. Далее здесь обычно берут в качестве результата взвешенное среднее всех внутренних состояний сети encoder(веса получается, например, с помощью softmax), полученный вектор подают на вход RNN decoder.

3.6.2. Трансформеры

Трансформер — вид нейронных сетей, используемых в задачах обработки текстов, звуков, временных рядов. Ключом архитектуры трансформеров являетсся механизм внимания. В общем случае трансформер состоит из двух частей : кодировщика encoder и декодировщика decoder. Оба слоя encoder и decoder используют механизм внимания для

обработки последовательности эмбеддингов(трансформеры не предполагают использование RNN). Делают они это с помощью self-attention. Изобразить работу self-attention можно следующим образом



Всего слой self-attention состоит из трех полносвязных слоев задаваемых матрицами весов W_B, W_Y, W_R, для каждого эмбеддинга входной последовательности мы считаем его выход со слоем W_B(синий вектор), для каждого другого эмбеддинга мы прогоняем их через полносвязные сети W_Y и W_R, далее берем скалярное произведение синего с желтым, получаем набор чисел, эти числа пропускаем через softmax и получаем набор весов, далее рассматриваем взвешенную сумму красных векторов и получаем результирующий вектор для текущего эмбеддинга. Теперь для каждого эмбеддинга мы получили новый вектор, который хранит в себе информацию о связях слова со словами в других текстах. То есть, заменили работу RNN с помощью механизма внимания.

3.6.3. BERT

BERT — трансформер encoder(не имеет слоя decoder), обученный на задаче заполнения пропусков в предложениях. Однако, как позже

выяснилось, механизм self-attention позволяет настолько хорошо улавливать связи между словами внутри предложения, что BERT стал использоваться как feature extractor во многих других задачах. Делается это с помощью fine tuning и transfer learning. Суть этих методов очень простая, давайте возьмем предобученный BERT в качестве способа построения эмбеддингов, а дальше навесим на него свой классификатор(можно взять обычные линейные слои) для нашей конкретной задачи. В случае transfer learning мы полностью замораживаем веса самого BERT и обучаем лишь классификатор, в fine tuning мы также дообучаем BERT вместе с классификатором. Понятно, что использование этих методов зависит от конкретной задачи и объема предоставленных данных для обучения. Если данных много — используем fine tuning, если мало — transfer learning.

4. Метод

В обзоре было рассмотрено большое количество методов решения задач NLP. Наша задача формулируется следующим образом: дана статья, необходимо по ее заголовку и описанию предсказать тематику статьи согласно таксономии arxiv.org. Попробуем для нашей задачи transfer learning на предобученном BERT. Для работы будем использовать библиотеку transformers, в которой собраны все основные модели трансформеров.

4.1. Данные

Для построение модели будут использоваться данные предоставленные платформой Kaggle[**?**].

4.1.1. Описание данных

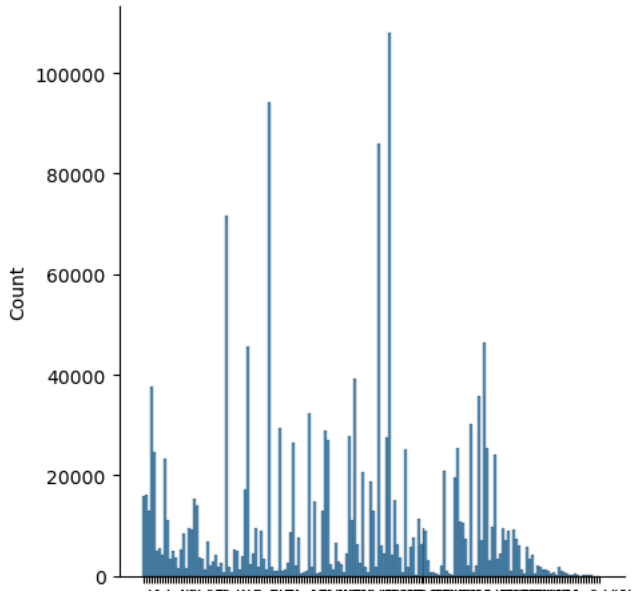
В датасете описаны следующие признаки :

1. Название статьи
2. Описание статьи
3. Авторы
4. Словарь тегов(тематик) статьи
5. Год публикации

4.1.2. Предобработка данных

Сначала удалим все ненужные признаки, оставив лишь название статьи, ее описание, а также словарь тегов. Теперь извлечем из словаря тегов нужные нам тематики. Обратим внимание, что у каждой статьи на arxiv есть несколько тем, но самая главная выделена жирным шрифтом, можно было бы попробовать решать задачу мультиклассификации, но на данном этапе это лишь сильно усложнит задачу,

поэтому из всех тегов оставим лишь главный тег, каждый тег закодируем отдельным числом(номер класса) — это будут наши таргеты. Посмотрим на распределение тегов



Видим, что некоторые теги встречаются очень часто, учтем это при разбиении данных на обучающую и валидационную выборку, чтобы предотвратить переобучение.

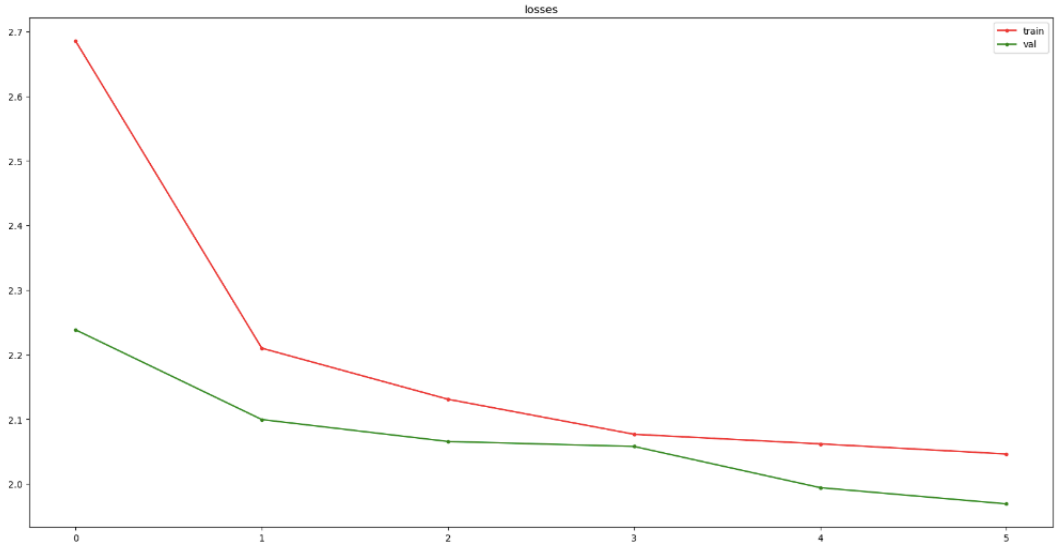
Теперь надо подготовить заголовки статей и их описания для векторизации. Приведем все тексты к нижнему регистру и удалим служебные символы(знаки препинания и прочее). Строить вручную эмбеддинги для слов не будем, воспользуемся предобученными эмбеддингами из BERT.

4.2. Обучение модели

Используем метод transfer learning, навесим на BERT два полносвязных слоя, первый размера (768, 1024), второй (1024, 152), 152 — столько итоговых классов у нас получилось. В качестве оптимизатора возьмем Adam, а в качестве лосса NLLLoss.

5. Результаты

Обучение проходило в 6 эпох, график обучения получился следующим



Метрика accuracy на тесте показала результат 0.39. Из графика видно, что последние эпохи наша нейронная сеть уже почти не обучалась. Отсюда можно сделать вывод, что для улучшения результата нужно видоизменять архитектуру или же использовать fine tuning для дообучения самого BERT. Данных в нашей конкретной задаче изначально было немного, поэтому было принятно решения обучаться именно с помощью transfer learning. Однако, если собрать побольше данных, можно попробовать и fine tuning. Сам по себе результат 0.39 не так плох для прямолинейного решения с дообучением BERT, ведь всего таргетов 152, при этом наблюдается сильный дизбаланс классов.

6. Заключение

В ходе работы было сделано :

1. Проведён обзор существующих инструментов для работы с задачами из области Natural Language Processing
2. Собраны и предобработаны данные для задачи распознавания тематик статьи.
3. С помощью transfer learning на предобученном BERT получена точность 0.39 на тестовых данных.

В будущем планируется еще глубже изучить архитектуру трансформеров и попробовать с помощью них улучшить результат в данной задаче.

Код с предобработкой данных:
https://www.kaggle.com/code/enoki1/notebook86e15be177
Код с обучением модели:
https://github.com/enoki1/some-scripts/blob/main/bert_transfer_learning.ipynb

Список литературы

[1] https://arxiv.org/abs/1706.03762

[2] https://arxiv.org/abs/1808.03314

[3] https://arxiv.org/abs/1912.05911

[4] https://arxiv.org/pdf/1909.09586.pdf

[5] https://arxiv.org/pdf/1701.05923.pdf

[6] https://arxiv.org/pdf/2107.02248.pdf

[7] https://arxiv.org/abs/2302.07730

[8] https://arxiv.org/abs/1810.04805

[9] https://arxiv.org/abs/2002.06305