

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ имени
М.В.ЛОМОНОСОВА»

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ
КАФЕДРА ТЕОРЕТИЧЕСКОЙ ИНФОРМАТИКИ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(ДИПЛОМНАЯ РАБОТА)
специалиста

**МОДЕЛЬ ТРАНСЛЯЦИИ С АКТИВАЦИЕЙ ДЛЯ
ИЗВЛЕЧЕНИЯ СУЩНОСТЕЙ ИЗ ТЕКСТА**

Выполнила студентка
603 группы
Бешева Инна Кахавна

подпись студента

Научный руководитель:
н.с., к. ф.-м. н.
Шокуров Антон
Вячеславович

подпись научного
руководителя

Москва

2019 г.

Содержание

1	Введение	2
2	Постановка задачи	3
3	Подготовка текстовых данных	4
3.1	Токенизация	4
3.2	Набивка	5
4	Векторное представление слов	7
4.1	Необучаемый слой Word2Vec	8
4.2	Обучаемый слой	9
4.3	Частично обучаемый слой	9
4.4	Особенности задачи	10
5	Рекуррентные нейронные сети	11
5.1	RNN	11
5.2	LSTM	12
5.3	Двунаправленная рекуррентная нейронная сеть	14
6	Модель трансляции	14
6.1	Кодировщик	15
6.2	Декодировщик	16
6.3	Лучевой поиск	19
7	Механизм внимания	20
7.1	Механизм внимания Льюнга	20
7.2	Механизм внимания Богданау	22
8	Архитектура нейронной сети	23
9	Метрика ROUGE	26
10	Результаты	27
11	Вывод	28
12	Список литературы	30

1 Введение

Text mining это частный случай интеллектуального анализа данных (data mining), в котором в качестве набора данных выступает неструктурированный набор текстовой информации. То есть, посредством методов интеллектуального анализа текстов возможна машинная обработка больших массивов текстовых данных.

Одним из методов интеллектуального анализа текстов является машинное обучение. В зависимости от типа решаемой задачи создается обучающая выборка. Например, для задачи классификации текстов объектом выступает последовательность слов из данного текста, а таргетом - номер класса, к которому текст относится; для задачи нахождения определенных паттернов в тексте объектом все так же остается последовательность слов текста, а таргетом выступает последовательность паттернов, соответствующих каждому слову из текста; для задачи составления пересказа текста объектом выступает последовательность слов полного текста, а таргетом - последовательность слов составленного человеком пересказа.

Во всех перечисленных задачах объектами выступают последовательности, длины которых могут отличаться. Поэтому в качестве моделей для решения этих задач используются рекуррентные нейронные сети, которые в отличие от стандартных простых алгоритмов машинного обучения позволяют обрабатывать последовательности произвольной длины, учитывают порядок слов в тексте и их взаимное расположение.

Особый интерес среди перечисленных задач представляет последняя - задача составления пересказа текста или задача извлечения сущностей из текста. В этой задаче не только объект является последовательностью произвольной длины, но и длина таргета произвольна и не зависит от длины входного объекта. В связи с этим в качестве модели для решения данной задачи используется рекуррентная нейронная сеть специфической архитектуры sequence-to-sequence (модель трансляции). Эта модель состоит из двух основных блоков: кодировщика (encoder) и декодировщика (decoder). Кодировщик, пропуская входную последовательность слов через слои рекуррентных клеток, кодирует входной объект (полный текст) в некий вектор. Затем декодировщик получает на вход вектор закодированного входного текста и с помощью последовательности

рекуррентных клеток выполняет его декодировку в выходную последовательность слов, которая представляет собой сгенерированный пересказ. Данный тип архитектуры нейронной сети имеет ряд тонкостей, которые будут рассмотрены в этой работе.

Также задача составления пересказа представляет особый исследовательский интерес среди прочих задач обработки естественного языка, так как для ее решения требуется разработать модель умеющую "понимать" текст и извлекать из него самую важную по смыслу информацию. Поэтому обученный для данной задачи кодировщик может использоваться для других задач анализа текстов для закодирования входного текста в вектор.

В рамках данной работы я детально рассмотрю архитектуру модели sequence-to-sequence и проведу сравнение нескольких ее вариаций, также опишу этапы подготовки текстовых данных, способы получения векторных представлений слов и вычисление метрики, используемой для оценки качества решения задачи.

2 Постановка задачи

Задачи составления пересказов делятся на два типа: основанные на извлечении (extraction-based) и основанные на абстракции (abstraction-based). Задача первого типа подразумевает извлечение из полного текста самых важных ключевых слов и предложений, отражающих суть текста, но в рамки этой задачи не входит генерация нового текста, в котором присутствуют слова и фразы неиспользованные в исходном тексте. Задача второго типа, наоборот, состоит в том, чтобы сгенерировать краткий текст емко излагающий смысл полного текста и представляет гораздо больший исследовательский интерес. В рамках данной работы будет решаться задача второго типа - основанная на абстракции.

Сформулируем задачу на математическом языке.

Пусть $W = \{w_i\}_{i=1}^N$ - словарь (множество слов данного языка). Пусть имеется набор полных текстов $\{a_i\}_{i=1}^n$, а так же набор пересказов (кратких изложений) данных текстов $\{b_i\}_{i=1}^n$. Каждый текст представляет из себя последовательность слов из словаря: $a_i = (w_{i_{11}}, \dots, w_{i_{1m}})$, $b_i =$

$(w_{i_{21}}, \dots, w_{i_{2l}})$, где m - длина полного текста a_i , $l < m$ - длина текста пересказа b_i , а $w_{i_{1k}} \in W$ - k -е слово в тексте a_i , $w_{i_{2k}} \in W$ - k -е слово в тексте b_i .

Требуется построить модель, которая сопоставляет входному тексту a_i сгенерированный текст $\hat{b}_i = (\hat{w}_{i_{21}}, \dots, \hat{w}_{i_{2p}})$, где $p < m$, $\hat{w}_{i_{2k}} \in W$, являющийся кратким пересказом исходного текста. "Похожесть" сгенерированного пересказа $\hat{b}_i$ и составленного человеком пересказа b_i измеряется при помощи набора метрик, описанного в главе "Метрика ROUGE".

В качестве данных для обучения и тестирования модели использовались текстовые данные набора English Gigaword. Количество объектов в обучающей выборке 10000, в тестовой - 1000. Средняя длина полного текста 28.6404 токенов, максимальная - 55 токенов. Средняя длина пересказа 7.5538 токена, максимальная - 24 токена.

3 Подготовка текстовых данных

Для решения задачи обработки текстовых данных методами машинного обучения данные требуют описания в дискретном числовом формате. Тексты преобразуются в последовательность чисел.

Для начала тексты были приведены к нижнему регистру, пунктуация была удалена. Нормализация текста (приведение всех слов текста к словарной форме) не проводилась, так как разные временные формы глаголов и числа для существительных важны для генерации текста.

3.1 Токенизация

Токенизация это процесс преобразования текста в последовательность определенных элементов - токенов. Токенами могут быть слова, фразы, символы. Я определила токен как "слово неразделенный пробелом набор букв или цифр. Каждому уникальному токenu сопоставляется число. К тому же количество токенов может быть ограничено

Рассмотрим подробнее процесс токенизации. Пусть имеется набор текстов $\{l_i\}_{i=1}^n$, $l_i = (w_{i_1}, \dots, w_{i_{m_i}})$ - текст длины m_i , w_{i_k} - k -е слово в тексте l_i . Пусть $W = \{w_k\}_{k=1}^N$ - множество всех слов (токенов), хотя бы едино-

жды встречавшихся в текстах l_i . Сопоставим каждому слову $w_k \in W$:

$$\nu(w_k) = \sum_{i=1}^n \sum_{j=1}^{m_i} \chi_{w_k}(w_{i_j}),$$

где $\chi_b(a)$ - характеристическая функция:

$$\chi_b(a) = \begin{cases} 1 & a = b \\ 0 & a \neq b \end{cases}$$

То есть $\nu(w_k)$ - натуральное число, обозначающее сколько раз слово (токен) w_k встретилось в текстах.

Упорядочим слова $\{w_k\}_{k=1}^N$ по $\nu(w_k)$ по убыванию, то есть первое слово - самое часто встречающееся в текстах, последнее - самое редкое. Если у каких-то слов $\nu(w_k)$ совпали, то упорядочим их в лексикографическом порядке. Получим упорядоченный по частоте использования слов в тексте словарь: $\hat{W} = \{\hat{w}_j\}_{j=1}^N$, где $\nu(\hat{w}_k) \geq \nu(\hat{w}_l) \forall k < l$.

Сопоставим каждому слову w_{j_k} - число k (токен), обозначающее его порядок в упорядоченном словаре, а каждому тексту $l_i = (w_{i_1}, \dots, w_{i_{m_i}}) = (\hat{w}_{j_1}, \dots, \hat{w}_{j_{m_i}})$ - вектор $(j_1, \dots, j_{m_i})$. Если количество токенов ограничено некоторым числом M , то значения $n_k > M$ будут удалены из числового вектора.

Таким образом, текстам были сопоставлены векторы - последовательности чисел. На практике для реализации данного подхода использовался класс `Tokenizer` библиотеки `keras`, а количество токенов было ограничено 20 000.

3.2 Набивка

Обучение нейронной сети происходит по технологии `mini-batch training`. Это значит, что обучающая выборка разбивается на партии (`batches`) некоторого фиксированного размера. Нейронная сеть совершает прямое распространение сигнала (`forward propagation`) или, иначе говоря, строит предсказание на некоторой партии, затем вычисляется оптимизируемая функция потерь на всей партии. Только после этого параметры

модели пересчитываются методом обратного распространения ошибки (backpropagation). То есть пересчет параметров происходит ни на каждом объекте из обучающей выборки, а на каждой партии. Такая технология уменьшает вычислительные затраты и делает обучение более робастным и быстрым.

Как говорилось ранее, рекуррентная нейронная сеть может обрабатывать последовательности произвольной длины, но при mini-batch обучении последовательности одной партии должны быть приведены к одной и той же длине, чтобы было возможно осуществление обратного распространения ошибки. Для этого используется техника набивки (padding) - заполнения отсутствующих токенов в более коротких последовательностях специальным дополняющим токеном (padding symbol) $\langle \text{PAD} \rangle$. Также архитектура рекуррентной нейронной сети Encoder-Decoder требует ввод следующих токенов: начальный символ $\langle \text{SOS} \rangle$ (start of sentence), конечный символ $\langle \text{EOS} \rangle$ (end of sentence), неизвестное слово $\langle \text{UNK} \rangle$ (unknown word). Данные токены должны быть внесены в словарь, например, на нулевое и последние места. Таким образом, если N - размерность словаря токенов $\hat{W}$, то при переводе последовательности токенов в числовую последовательность, токен $\langle \text{PAD} \rangle$ переходит в 0, токен $\langle \text{SOS} \rangle$ переходит в $N + 1$, токен $\langle \text{EOS} \rangle$ переходит в $N + 2$, токен $\langle \text{UNK} \rangle$ переходит в $N + 3$. Также выбирается максимальная длина текста, которой ограничиваются все тексты, что способствует избавлению от выбросов.

Пусть l - размер batch-а (количество объектов в партии), n_max - выбранное ограничение длины текста, $\{l_i\}_{i=1}^l = \{(\hat{w}_{i_1}, \dots, \hat{w}_{i_{m_i}})\}_{i=1}^l$ - тексты данного batch-а, представленные как набор токенов. Каждый текст l_i представляется числовой последовательностью $(i_1, \dots, m_i)$, а $m_max = \max_{i \in \{1, \dots, l\}} (m_i)$ - максимальная длина текста среди данного batch-а. $n = \min(n_max, m_max)$ - длина, к которой будут приводиться все последовательности данного batch-а. Определим функцию *pad_sentence* следующим образом:

pad_sentence($l_i, n, symbol$) :

```

if  $n \leq m_i$  then
  if  $symbol = \langle \text{SOS} \rangle$  then
     $pad\_l = (\langle \text{SOS} \rangle, \hat{w}_{i_1}, \dots, \hat{w}_{i_{n-1}})$ 
  else if  $symbol = \langle \text{EOS} \rangle$  then

```

```

    pad_l = ( $\hat{w}_{i_1}, \dots, \hat{w}_{i_{n-1}}, < EOS >$ )
else
    pad_l = ( $\hat{w}_{i_1}, \dots, \hat{w}_{i_n}$ )
end if
else
    if symbol = < SOS > then
        pad_l = ( $< SOS >, \hat{w}_{i_1}, \dots, \hat{w}_{i_{m_i}}, \underbrace{< PAD >, \dots, < PAD >}_{n - m_i - 1}$ )
    else if symbol = < EOS > then
        pad_l = ( $\hat{w}_{i_1}, \dots, \hat{w}_{i_{m_i}}, < EOS >, \underbrace{< PAD >, \dots, < PAD >}_{n - m_i - 1}$ )
    else
        pad_l = ( $\hat{w}_{i_1}, \dots, \hat{w}_{i_{m_i}}, \underbrace{< PAD >, \dots, < PAD >}_{n - m_i}$ )
    end if
end if
return pad_l

```

С помощью данной функции все тексты данного batch-a приводятся к общей длине n , при необходимости к ним добавляется начальный или конечный символ.

Обучающая выборка была разбита на партии размера 64. Так как 90 процентов полных текстов обучающей выборки имеют длину меньшую 34 токенов, ограничительная длина n для полных текстов принималась равной 35. Так как 90 процентов текстов пересказов обучающей выборки имеют длину меньшую 10 токенов, ограничительная длина n для пересказов - 10.

4 Векторное представление слов

Векторное представление слов - сопоставление словам (или, возможно, фразам) векторов из векторного пространства $\mathbb{R}^n$, где n (размерность векторного пространства, выбранная произвольно) значительно меньше количества слов в словаре.

Входным слоем нейронной сети является Embedding Layer - слой, сопоставляющий токену вектор векторного представления соответствующего

слова. Embedding Layer представляет из себя действие специальной матрицы (Embedding Matrix) на токен данного слова.

Embedding Matrix - матрица размерности $n \times N$, где n - размерность пространства векторных представлений слов, а N - количество слов в словаре. k -ый столбец матрицы - вектор $x_k = (x_{k_1}, \dots, x_{k_n})$ размерности n , соответствующий токenu k слова $w_{j_k} \in W, k \in \{1, \dots, N\}$. Действие матрицы на токен k - матричное умножение Embedding Matrix на единичный вектор $e_k = (0, \dots, 1, \dots, 0)$:

$$\begin{pmatrix} x_1^1 & x_1^2 & x_1^3 & \dots & x_1^k & \dots & x_1^n \\ x_2^1 & x_2^2 & x_2^3 & \dots & x_2^k & \dots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_n^1 & x_n^2 & x_n^3 & \dots & x_n^k & \dots & x_n^n \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} x_1^k \\ x_2^k \\ \vdots \\ x_n^k \end{pmatrix} = x_k$$

Но как получить векторное представление слов? Существует несколько подходов.

4.1 Необучаемый слой Word2Vec

Word2Vec 2013 разработанная Google технология построения векторного представления слов согласно их семантической близости. Алгоритм каждому слову сопоставляет вектор, причем близкие слова соответствуют близким векторам. Мерой близости слов выступает их контекстная близость: близкие слова встречаются в тексте рядом с одинаковыми словами. А расстояние между векторами измеряется при помощи косинусного сходства (чем ближе вектора, тем меньше угол между ними).

Обучая нейронную сеть, Word2Vec максимизирует косинусную меру близости между векторами слов, которые встречаются в похожих контекстах и минимизирует косинусную меру между словами которые не встречаются рядом. На выход Word2Vec передает координаты векторов, соответствующих данным словам. Подробнее об архитектуре нейронной сети, можно прочитать в моей предыдущей курсовой работе.

В открытом доступе можно найти уже обученные на больших массивах текстовой информации реализации Word2Vec, с их помощью мож-

но получить векторные представления слов различной размерности (например: 100, 200, 300). Если вдруг каких-то слов недостает в словарях обученных алгоритмов, их векторным представлением можно считать нулевой вектор, фиксированный случайный вектор или вектор, соответствующий самому редко встречающемуся слову из словаря.

Иногда более специфические результаты дает самостоятельное обучение Word2Vec на данном наборе текстов или на дополнительном наборе. Для того чтобы обучить Word2Vec и получить векторное представление некоторой размерности, можно воспользоваться библиотекой gensim.

Если в качестве векторного представления слов используются вектора, полученные с помощью Word2Vec, то входной слой нейронной сети (Embedding Layer) следует установить необучаемым (untrainable) и самостоятельно составить Embedding Matrix.

Преимущество данного подхода состоит в использовании больших сторонних корпусов текстов и получении векторного представления, отражающего смысловую близость слов.

4.2 Обучаемый слой

Другой подход состоит в том, чтобы обучить Embedding Layer. Для этого следует каким-либо образом задать начальную Embedding Matrix. Можно задать ее нулевой или инициализировать случайно, выбрав произвольно размерность векторов. Популярный подход состоит в том, чтобы инициализировать матрицу с помощью готовых векторных представлений (например, Word2Vec). При обучении нейронной сети элементы матрицы будут пересчитаны согласно задаче минимизации функции потерь на выходном слое.

Преимущество данного подхода состоит в создании векторного представления слов, отражающего специфику решаемой задачи.

4.3 Частично обучаемый слой

Комбинированный подход позволит совместить преимущества двух других подходов. Идея состоит в том, чтобы векторное представление слова

представляло из себя наполовину обучаемые вектор - комбинацию вектора Word2Vec или Glove и обучаемых компонент.

Таким образом, $x_k = (\hat{x}_1^k, \dots, \hat{x}_n^k, x_1^k, \dots, x_m^k)$, где $\hat{x}_1^k, \dots, \hat{x}_n^k$ - фиксированные (необучаемые) компоненты вектора, а $x_1^k, \dots, x_m^k$ - обучаемые компоненты вектора, то есть пересчитываемые по мере обучения нейронной сети. Необучаемые компоненты вектора представляют из себя компоненты вектора, полученного при помощи некоторой технологии (Word2Vec или Glove).

Входной слой представляет из себя Embedding Matrix - матрица размерности $(n + m) \times N$, так как размерность пространства векторных представлений слов $n + m$.

4.4 Особенности задачи

Рекуррентная нейронная сеть архитектуры sequence-to-sequence (модель трансляции) имеет большое количество параметров, поэтому ее обучение требует большого количества итераций и данных. Но отразить специфику данной задачи с помощью векторных представлений, а также сделать модель способной обучать векторные представления неизвестных слов, оказалось не менее важной задачей, чем ускорить процесс обучения. Поэтому было решено инициализировать обучаемый входной слой матрицей, состоящей из готовых векторных представлений. В качестве векторных представлений использовались предобученная Google модель Word2Vec. Модель была обучена на новостном наборе данных Google News. Словарь модели содержит 3 миллиона слов, размерность векторных представлений $n = 300$.

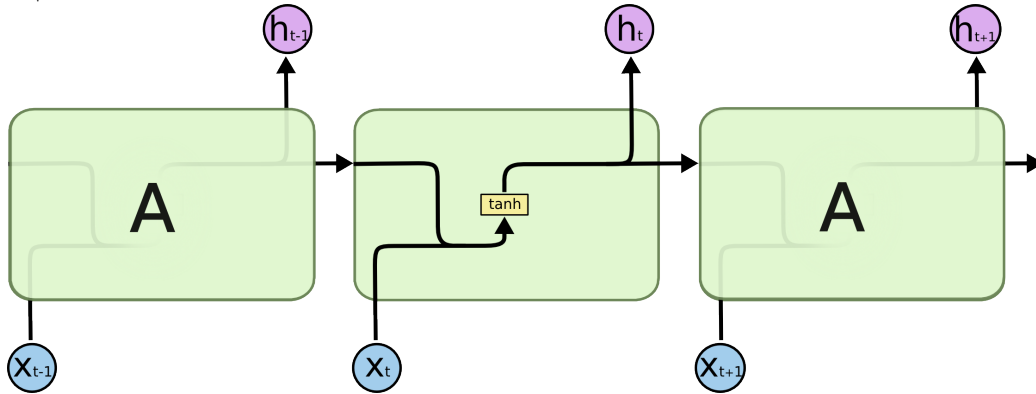
Векторные представления для конечного $\langle \text{EOS} \rangle$ и начального $\langle \text{SOS} \rangle$ символов и неизвестных слов $\langle \text{UNK} \rangle$ инициализировались как фиксированные случайные векторы $x_k = (x_1^k, \dots, x_n^k)$, $w_k \in \{ \langle \text{EOS} \rangle, \langle \text{SOS} \rangle, \langle \text{UNK} \rangle \}$, компоненты которых принадлежат нормальному распределению со средним 0 и дисперсией 1: $x_k \in \mathcal{N}(0, 1)$. В качестве векторного представления дополняющего символа $\langle \text{PAD} \rangle$ используется нулевой вектор $x_k = (0, \dots, 0)$, $w_k = \langle \text{PAD} \rangle$.

5 Рекуррентные нейронные сети

Рекуррентные нейронные сети это вид нейронных сетей, где связи между элементами образуют направленную последовательность. Поэтому в отличие от многослойных перцептронов и простых алгоритмов машинного обучения, рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины. Все рекуррентные сети имеют цепную структуру: повторяющуюся клетку. Клеткой может быть единственный нейрон или последовательность нескольких. Прежде чем описать архитектуру модели трансляции Sequence-to-sequence, рассмотрим два вида рекуррентных нейронных клеток: стандартную клетку RNN и ее улучшенную модификацию LSTM.

5.1 RNN

Recurrent Neural Network (RNN) имеет очень простую структуру: только один слой с функцией активации *tahn*. Пусть имеется последовательность входных данных: $\{x_t\}_{t=1}^T$. В нашем случае $x_t = (x_1^t, \dots, x_n^t)$ - векторное представление t -ого слова из текста. Для последовательности из T элементов имеем T клеток RNN. Выход t -ой клетки соединен с входом $t + 1$ -ой клетки.



Клетки последовательно обрабатывают вектора слов следующим образом:

$$h_t = \tanh(Ux_t + Wh_{t-1} + b)$$

Начальный вектор h_0 можно определить как нулевой и задать случайно. Функцией активации вместо *tahn* могут быть выбраны: сигмоида, ReLu, softmax или другая. Веса U и W одинаковы $\forall t$.

В процессе обучения нейронной сети обновление весов осуществляется методом обратного распространения ошибки. Пусть выходной слой имеет последовательность выходов $\{\hat{y}_t\}_{t=1}^T$, а $E(y, \hat{y}) = \sum_{t=1}^T E_t(y_t, \hat{y}_t)$ - оптимизируемая функция потерь (например, перекрёстная энтропия).

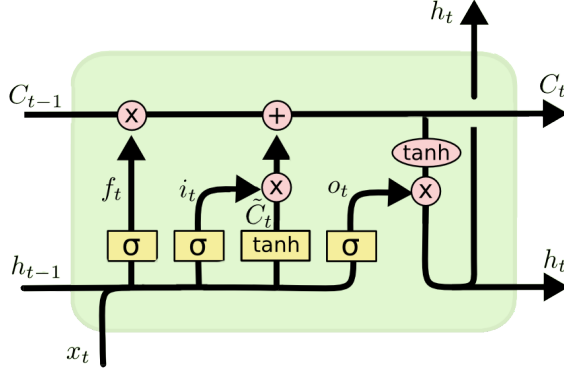
Для того, чтобы осуществить метод обратного распространения ошибки, нужно вычислить градиенты функции потерь в отношении весов W следующим образом:

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial W} = \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial W} = \sum_{k=1}^t \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W} = \sum_{k=1}^t \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \left(\prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$

Производные $\tanh$ и сигмолды по модулю ограничены единицей и их значения стремятся к нулю при увеличении модуля аргумента. Поэтому градиенты принимают значения близкие к нулю, а чем больше расстояние между t -ым объектом и предшествующим k -ым, тем меньшее влияние на обновление весов оказывает k -ый объект. Возможно и обратное, при использовании других функций активации: градиенты будут больше единицы по модулю и начнут неограниченно расти. В этом случае, дальние от t -ого объекты будут получать большее значение при пересчете весов. Эту проблему называют проблемой затухания/взрыва градиента (vanishing/exploding gradient problem). Таким образом, RNN не всегда удастся уловить долгосрочные или краткосрочные зависимости в последовательности объектов. К счастью, существует рекуррентная нейронная сеть специфической структуры, справляющаяся с этой проблемой.

5.2 LSTM

Long Short Term Memory networks - называемые “LSTMs” - это особые нейронные сети, способные находить долго- и краткосрочные зависимости. LSTM также имеют цепную структуру, но устройство повторяющейся клетки более сложное. Она состоит из четырех нейронов, соединенных специальным образом. Желтыми блоками обозначены нейроны, а розовыми кругами - покомпонентные линейные операции:



Рассмотрим структуру LSTM-клетки подробно. У клетки имеются две рекуррентные компоненты: выходной вектор h_t и вектор состояний C_t той же размерности. Отличительной особенностью является то, что LSTM-клетка не использует функцию активации внутри компонент C_t . Вектор состояний проходит напрямую через всю цепочку, участвуя лишь в нескольких линейных преобразованиях. Таким образом, хранимое значение не размывается во времени, и градиент или штраф не исчезает при использовании метода обратного распространения ошибки во времени при тренировке сети.

Тем не менее, LSTM может удалять информацию из рекуррентных компонент. Этот процесс регулируется структурами, называемыми фильтрами (gates): фильтр забывания f_t (forget gate, входной фильтр i_t (input gate) и выходной фильтр o_t (output gate). Вектора $f_t, i_t, o_t \in [0, 1]^m$, где m - размерность векторов h_t и C_t , определяются следующими формулами:

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

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

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

Новые значения вектора $C_t \in [-1, 1]^m$ строятся с помощью tanh-слоя:

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

Затем с помощью фильтров операций покомпонатного умножения $*$ вычисляются обновленные вектора h_t и C_t :

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

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

Начальные вектора h_0, C_0 можно также определить как нулевые или задать случайно.

5.3 Двухнаправленная рекуррентная нейронная сеть

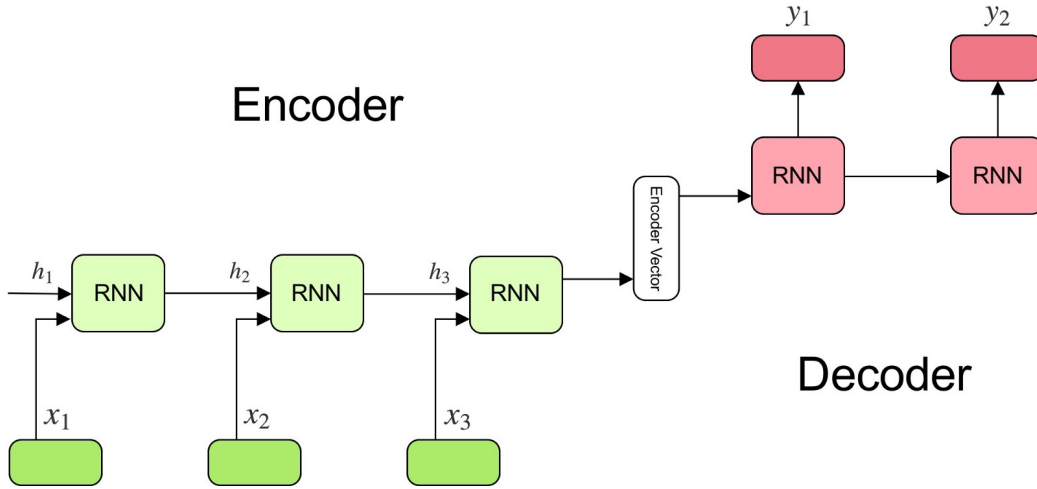
Нейронная сеть может состоять из нескольких рекуррентных нейронных слоев следующим образом: последовательность выходных векторов рекуррентного слоя $\{h_t\}_{t=1}^T$ подается на следующий слой в качестве входной последовательности $\{x_t\}_{t=1}^T$. Существуют и более интересные подходы комбинации рекуррентных слоев. Например, двухнаправленная рекуррентная нейронная сеть (Bidirectional Recurrent Neural Networks). BRNN это комбинация двух независимых рекуррентных слоев (RNN, LSTM, GRU или другой). В один слой последовательность входных векторов подается в прямом порядке $\{x_t\}_{t=1}^T$, а в другую - в обратном $\{x_{T-t+1}\}_{t=1}^T$. На выходе из прямого слоя имеем $\{h_t^{forward}\}_{t=1}^T$, а из обратного $\{h_t^{backward}\}_{t=1}^T$. Полученные вектора обычно конкатенируются в каждый временной шаг: $h_t = [h_t^{forward}, h_t^{backward}]$. Но могут использоваться и другие операции: среднее, сумма и так далее. Такой слой называется двухнаправленным.

В рамках работы над курсовой 5-ого курса я изучала различные виды рекуррентных клеток и комбинации рекуррентных слоев для решения задач классификации и разметки последовательности. LSTM-клетка показала наилучшее качество, поэтому решая задачу пересказа, я так же решила использовать LSTM-клетку. К тому же удачной практикой при решении задач было поставить BLSTM-слой следующим после Embedding слоя. В задаче пересказа BLSTM-слой будет использоваться в архитектуре Encoder-а. Тем самым, мы получим слой, "читающий" наш текст в прямом и в обратном порядке.

6 Модель трансляции

Задача сопоставления полному тексту его пересказа относится к задачам типа Sequence-to-sequence как и задача машинного перевода или диалоговые системы. В данных задачах рекуррентная нейронная сеть строения Encoder-Decoder показала себя эффективной моделью, дающей высокую точность. Рассмотрим архитектуру данной нейронной сети подробнее.

Данная архитектура состоит из двух моделей: кодировщика (Encoder) и декодировщика (Decoder). Задача кодировщика - считать входную последовательность векторов и закодировать ее в некоторый вектор фиксированной размерности. Задача декодировщика - раскодировать этот вектор в предсказанную последовательность векторов. Рассмотрим их устройство подробнее.



6.1 Кодировщик

Блок кодировщик (Encoder) следует за Embedding слоем, который сопоставляет токenu его векторное представление. Кодировщик представляет из себя последовательность рекуррентных слоев, которые в свою очередь состоят из рекуррентных клеток (RNN, LSTM или другие). Рекуррентные слои могут быть обычными или двунаправленным. Входными векторами на первый слой кодировщика является последовательность векторных представлений токенов $\{x_t\}_{t=1}^T$. Размерность выходных векторов, количество слоев и тип рекуррентной нейронной клетки являются гиперпараметрами модели. Пусть имеется l рекуррентных слоев, состоящих из RNN клеток, а $\{h_t^k\}_{t=1}^T$ - выходные вектора k -ого рекуррентного слоя, тогда t -ый выходной вектор на k -ом слое определяются как функция от выходного вектора предыдущего слоя и выходного вектора предыдущей клетки того же слоя:

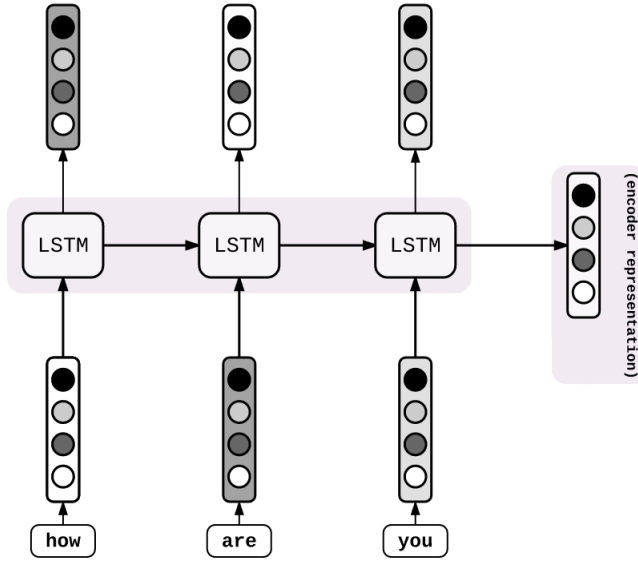
$$h_t^k = f(h_t^{k-1}, h_{t-1}^k), k \in \{2, \dots, l\}, t \in \{1, \dots, T\}$$

Вектором кодировки, который подается на вход декодировщику, является последний вектор в последовательности выходных векторов последнего слоя: h_T^l . Назовем его: $e = h_T^l$. Исходя из этого двунаправленный слой не имеет смысла ставить последним. В случае рекуррентных слоев использующих LSTM клетку выходные вектора h_t^k и вектора состояний C_t^k определяются следующим образом:

$$h_t^k = f_h(h_t^{k-1}, h_{t-1}^k, C_{t-1}^k), k \in \{2, \dots, l\}, t \in \{1, \dots, T\}$$

$$C_t^k = f_C(h_t^{k-1}, h_{t-1}^k, C_{t-1}^k), k \in \{2, \dots, l\}, t \in \{1, \dots, T\}$$

Вектора h_T^l и C_T^l передаются декодировщику, назовем их: $e_h = h_T^l, e_c = C_T^l$.



6.2 Декодировщик

Вектор или вектора (в случае LSTM) кодировки должны агрегировать в себе всю информацию входного текста, так как они должны быть раскодированы блоком Decoder. Декодировщик представляет из себя последовательность рекуррентных слоев, состоящих из рекуррентных клеток (RNN, LSTM или другие) того же типа, что и у кодировщика, и выходного слоя. Пусть имеется q рекуррентных слоев, состоящих из LSTM клеток с выходными векторами $\hat{h}_s^k$ и векторами состояний $\hat{C}_s^k$. Начальными

состояниями первой клетки первого слоя будут вектора кодировки:

$$\hat{h}_0^1 = e_h, \hat{C}_0^1 = e_C$$

. Входным вектором для первой клетки первого слоя будет векторное представление специального начального токена: $x_{i_0} = (x_1^{i_0}, \dots, x_n^{i_0})$, $w_{i_0} = < SOS >$. Тогда выходной вектор и вектор состояний первой клетки первого слоя будет вычисляться следующим образом:

$$\hat{h}_1^1 = f_h(x_{i_0}, e_h, e_C)$$

$$\hat{C}_1^1 = f_C(x_{i_0}, e_h, e_C)$$

Далее выходной вектор $\hat{h}_1^1$ передается второму слою и вычисляется $\hat{h}_1^2$ и так далее:

$$\hat{h}_1^k = f_h(\hat{h}_1^{k-1}, \hat{h}_0^k, \hat{C}_0^k), k \in \{2, \dots, q\}$$

Начальные состояния $\hat{h}_0^k, \hat{C}_0^k$ задаются нулевыми или случайными векторами. Выходной вектор последнего слоя $\hat{h}_1^q$ передается на вход выходному слою. Выходной слой представляется из себя N нейронов с функцией активации softmax, где N - размерность словаря токенов. Тем самым выходной слой должен отобразить выходной вектор последнего слоя в вектор вероятностей слов из словаря:

$$p_1 = softmax(V\hat{h}_1^q + b_p)$$

Затем используя вектор вероятностей $p_1 \in \mathbb{R}^N$ для того, чтобы получить предсказанное слово как w_{i_1} , где $i_1 = argmax(p_1)$.

Далее векторное представление x_{i_1} предсказанного слова w_{i_1} подается на вход второй рекуррентной клетке первого слоя, а именно:

$$\hat{h}_2^1 = f_h(x_{i_1}, \hat{h}_1^1, \hat{C}_1^1)$$

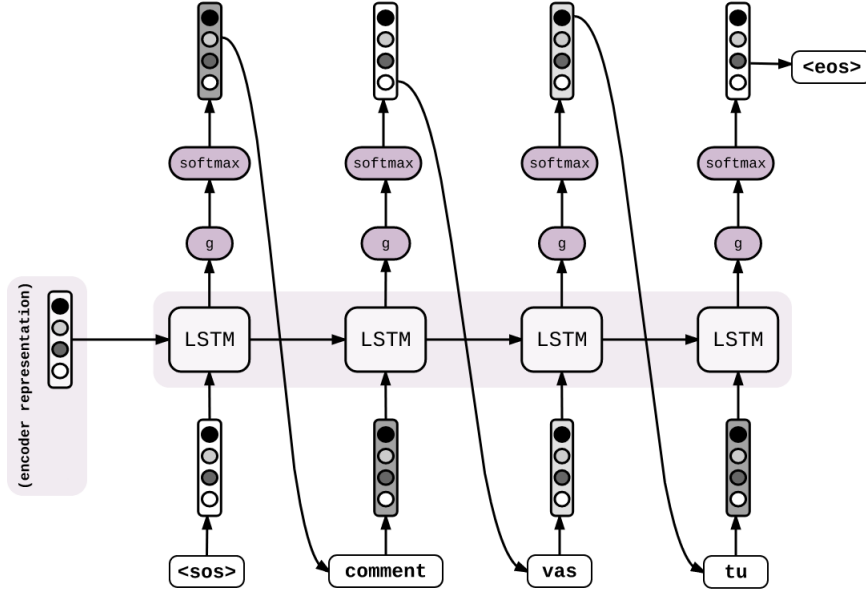
$$\hat{C}_2^1 = f_C(x_{i_1}, \hat{h}_1^1, \hat{C}_1^1)$$

После аналогичного процесса прохождения рекуррентных слоев и выходного слоя получаем предсказанное слово w_{i_2} и так далее. После предсказания s -ого слова w_{i_s} имеем:

$$\hat{h}_{s+1}^1 = f_h(x_{i_s}, \hat{h}_s^1, \hat{C}_s^1)$$

$$\begin{aligned}
\hat{C}_{s+1}^1 &= f_C(x_{i_s}, \hat{h}_s^1, \hat{C}_s^1) \\
\hat{h}_{s+1}^k &= f_h(\hat{h}_{s+1}^{k-1}, \hat{h}_s^k, \hat{C}_s^k), k \in \{2, \dots, q\} \\
\hat{C}_{s+1}^k &= f_C(\hat{h}_{s+1}^{k-1}, \hat{h}_s^k, \hat{C}_s^k), k \in \{2, \dots, q\} \\
p_{s+1} &= \text{softmax}(V\hat{h}_{s+1}^q + b_p) \\
i_{s+1} &= \text{argmax}(p_{s+1})
\end{aligned}$$

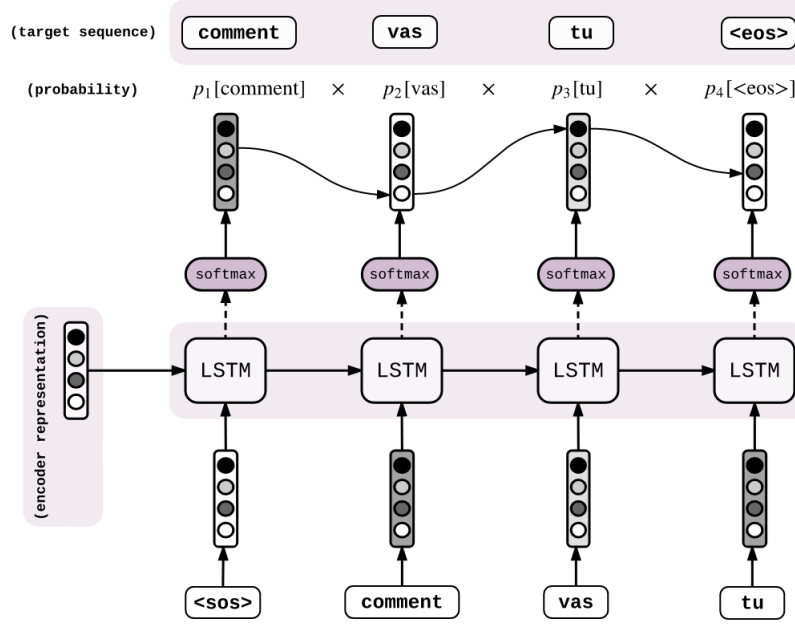
Процесс прекращается если предсказанное слово совпало со специальным конечным токеном: $w_{i_{s+1}} = \langle EOS \rangle$.



Использование s -ого предсказанного слова для предсказания $s + 1$ -ого приводит к быстрому накоплению ошибки, что делает процесс обучения модели долгим или даже невозможным. Поэтому при обучении модели в качестве входных векторов x_{i_s} подаются векторные представления слов таргета $\{w_{j_s}\}_{s=1}^S$. Тогда вероятность предсказать нужные токены на соответствующем шаге последовательности:

$$\mathbb{P}(w_{j_1}, \dots, w_{j_S}) = \prod_{s=1}^S p_s^{j_s}$$

где p_s^j - j -ая компонента предсказанного вектора вероятностей p_s .



Тогда максимизируя вероятность предсказания таргета, минимизируется следующая функция:

$$-\log \mathbb{P}(w_{j_1}, \dots, w_{j_S}) = -\log \prod_{s=1}^S p_s^{j_s} = -\sum_{s=1}^S \log p_s^{j_s}$$

Данная функция является перекрестной энтропией (cross entropy) между распределением таргета и предсказанными моделью вероятностями.

6.3 Лучевой поиск

При применении модели, предсказывая на каждом шаге слово с максимальной вероятностью, итоговая предсказанная последовательность слов может иметь не максимальную вероятность. Для решения этой проблемы был разработан метод лучевого поиска (Beam Search). Метод задается гиперпараметром K и состоит в том, чтобы на каждом шаге хранить K гипотез с максимальными вероятностями. Например, пусть на шаге s -ом множество гипотез имеет вид:

$$\mathcal{H}_s = \{(w_1^1, \dots, w_s^1), \dots, (w_1^K, \dots, w_s^K)\}$$

Рассмотрим все возможные $N * K$ гипотезы-кандидаты на $s + 1$ -ом шаге, образованные добавлением к гипотезам $\mathcal{H}_t$ всех слов из словаря $W = \{w_j\}_{j=1}^N$:

$$\mathcal{C}_{s+1} = \cup_{i=1}^K \{(w_1^i, \dots, w_s^i, w_1), \dots, (w_1^i, \dots, w_t^i, w_N)\}$$

Затем вычислим все $\mathbb{P}(w_1^i, \dots, w_t^i, w_j), i \in \{1, \dots, K\}, j \in \{1, \dots, N\}$, среди кандидатов выберем K с максимальными вероятностями и получим $\mathcal{H}_{s+1}$. Как только все гипотезы достигнут конечного токена $\langle \text{EOS} \rangle$, принимается гипотеза с максимальной вероятностью.

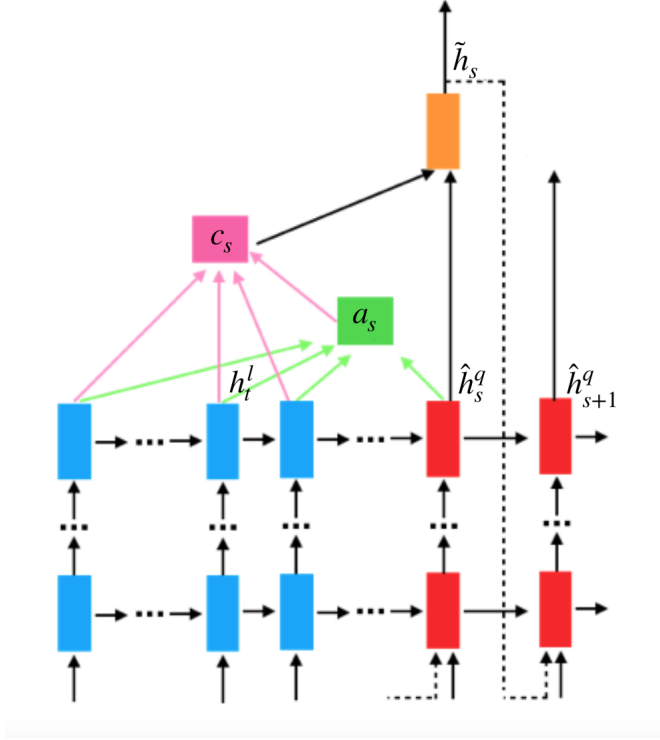
7 Механизм внимания

В описанном подходе вся входная последовательность слов была закодирована в один вектор (в случае LSTM два вектора). Но когда мы сами составляем пересказ текста (предсказываем каждое новое слово) мы по-разному акцентируемся на разных частях текста. Именно для того, что бы использовать разный контекст на каждом шаге предсказания Decoder-а был разработатан механизм внимания (Attention mechanism), позволяющий оценить важность каждого входного слова на данном шаге предсказания. Рассмотрим два классических механизма внимания: механизм внимания Льюнга (Luong) и механизм внимания Богданау (Bahdanau).

7.1 Механизм внимания Льюнга

Для того что применить данный механизм нужно сохранить в памяти не только последний выходной вектор последнего слоя Encoder-а, но и все выходные вектора последнего слоя $h_t^l, t \in \{1, \dots, T\}$. После вычисления s -ого выходного вектора последнего слоя вычиляется вектор attention-весов a_s и контекстный вектор c_s . Затем с их помощью вычисляется учитывающий внимание (attentional) выходной вектор $\tilde{h}_s$, который впоследствии используется для предсказания таргета:

$$\hat{h}_s^q \rightarrow a_s \rightarrow c_s \rightarrow \tilde{h}_s$$



Attention-веса на s -ом шаге представляют из себя набор из T векторов $a_s = \{a_s(t)\}_{t=1}^T$. Вектор $a_s(t)$ соответствует значимости t -ого входного слова Encoder-а для предсказания s -ого слова в Decoder-е, а именно:

$$a_s(t) = \text{align}(\hat{h}_s^q, h_t^l) = \frac{\exp(\text{score}(\hat{h}_s^q, h_t^l))}{\sum_{t'=1}^T \exp(\text{score}(\hat{h}_s^q, h_{t'}^l))}$$

Функция $\text{score}(\hat{h}_s^q, h_t^l)$ оценивает "схожесть" векторов $\hat{h}_s^q$ и h_t^l . Она может быть определена одним из трех следующих способов:

$$\text{score}(\hat{h}_s^q, h_t^l) = \begin{cases} \hat{h}_s^{q\top} h_t^l & \text{dot} \\ \hat{h}_s^{q\top} W_a h_t^l & \text{general} \\ v_a^\top \tanh(W_a [\hat{h}_s^q; h_t^l]) & \text{concat} \end{cases}$$

Далее с помощью attention-весов вычисляется контекстный вектор:

$$c_s = \sum_{t=1}^T a_s(t) h_t^l$$

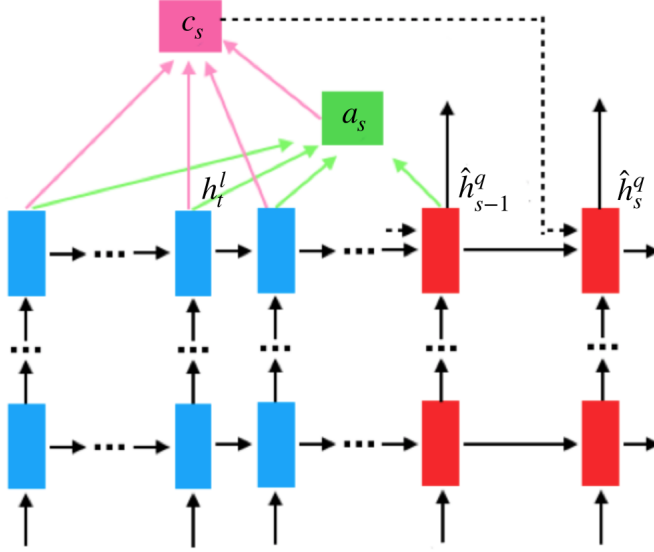
Вычисленный контекстный вектор используется для вычисления учитывающего внимание выходного вектора:

$$\tilde{h}_s = \text{tahn}(W_c[c_s; \hat{h}_s^q])$$

7.2 Механизм внимания Богданау

Для того что применить данный механизм нужно также сохранить в памяти не только последний выходной вектор последнего слоя Encoder-a, но и все выходные вектора последнего слоя $h_t^l, t \in \{1, \dots, T\}$. Вычисление s -ого выходного вектора последнего слоя будет осуществляться через $s-1$ -ый выходной вектор, вектор attention-весов a_s и контекстный вектор c_s :

$$\hat{h}_{s-1}^q \rightarrow a_s \rightarrow c_s \rightarrow \hat{h}_s^q$$



Attention-веса на s -ом шаге также представляют из себя набор из T векторов $a_s = \{a_s(t)\}_{t=1}^T$. Вектор $a_s(t)$ соответствует значимости t -ого входного слова Encoder-a для предсказания s -ого слова в Decoder-e. Но для их вычисления используется выходной вектор $s-1$ -ого шага $\hat{h}_{s-1}^q$ в отличии

от механизма Льюнга, а именно:

$$a_s(t) = \text{align}(\hat{h}_{s-1}^q, h_t^l) = \frac{\exp(\text{score}(\hat{h}_{s-1}^q, h_t^l))}{\sum_{t'=1}^T \exp(\text{score}(\hat{h}_{s-1}^q, h_{t'}^l))}$$

Функция $\text{score}(\hat{h}_{s-1}^q, h_t^l)$ оценивает "схожесть" векторов $\hat{h}_{s-1}^q$ и h_t^l определена аналогично.

Далее с помощью attention-весов вычисляется контекстный вектор:

$$c_s = \sum_{t=1}^T a_s(t) h_t^l$$

Вычисленный контекстный вектор используется для вычислений выходного вектора и вектора состояний (в случае LSTM):

$$\hat{h}_s^q = f_h(\hat{h}_s^{q-1}, \hat{h}_{s-1}^q, c_s)$$

$$\hat{C}_s^q = f_C(\hat{h}_s^{q-1}, \hat{h}_{s-1}^q, c_s)$$

8 Архитектура нейронной сети

При решении задачи составления пересказа особый интерес представляет исследование различных механизмов внимания. Поэтому я решила запрограммировать и обучить три различные модели: модель трансляции без механизма внимания, модель трансляции с механизмом внимания Льюнга и модель трансляции с механизмом внимания Богданау.

У всех трех моделей была следующая архитектура:

1. Слой векторных представлений

- (a) Обучаемый слой, инициализированный матрицей векторных представлений Word2Vec
- (b) Размерность векторных представлений 300

2. Encoder

- (a) Два двунаправленных рекуррентных нейронных слоя с операцией конкатенации
- (b) Используемая рекуррентная клетка: LSTM с выходными векторами размерностью 150

3. Decoder

- (a) Один рекуррентный нейронный слой
- (b) Используемая рекуррентная клетка: LSTM с выходными векторами размерностью 300
- (c) При предсказании использовался метод Beam Search

Ниже представлен программный код на языке Python, реализующий модель трансляции с механизмом внимания Богданау:

```
class Model(object):
    def __init__(self, id2word, article_max_len, summary_max_len, start_symbol_id, end_symbol_id,
                  args, forward_only=False):
        self.start_symbol_id = start_symbol_id
        self.end_symbol_id = end_symbol_id
        self.vocabulary_size = len(id2word)
        self.embedding_size = args.embedding_size
        self.num_hidden = args.num_hidden
        self.num_layers = args.num_layers
        self.learning_rate = args.learning_rate
        self.beam_width = args.beam_width
        if not forward_only:
            self.keep_prob = args.keep_prob
        else:
            self.keep_prob = 1.0
        self.cell = tf.nn.rnn_cell.BasicLSTMCell
        with tf.variable_scope("decoder/projection"):
            self.projection_layer = tf.layers.Dense(self.vocabulary_size, use_bias=False)

        self.batch_size = tf.placeholder(tf.int32, (), name="batch_size")
        self.X = tf.placeholder(tf.int32, [None, article_max_len])
        self.X_len = tf.placeholder(tf.int32, [None])
        self.decoder_input = tf.placeholder(tf.int32, [None, summary_max_len])
        self.decoder_len = tf.placeholder(tf.int32, [None])
        self.decoder_target = tf.placeholder(tf.int32, [None, summary_max_len])
        self.global_step = tf.Variable(0, trainable=False)

        with tf.name_scope("embedding"):
            if not forward_only and args.glove:
                init_embeddings = tf.constant(get_init_embedding(id2word), dtype=tf.float32)
            else:
                init_embeddings = tf.random_uniform([self.vocabulary_size, self.embedding_size], -1.0, 1.0)
            self.embeddings = tf.get_variable("embeddings", initializer=init_embeddings)
            self.encoder_emb_inp = tf.transpose(tf.nn.embedding_lookup(self.embeddings, self.X), perm=[1, 0, 2])
            self.decoder_emb_inp = tf.transpose(tf.nn.embedding_lookup(self.embeddings, self.decoder_input),
                                                perm=[1, 0, 2])

        with tf.name_scope("encoder"):
            fw_cells = [self.cell(self.num_hidden) for _ in range(self.num_layers)]
            bw_cells = [self.cell(self.num_hidden) for _ in range(self.num_layers)]
            fw_cells = [rnn.DropoutWrapper(cell) for cell in fw_cells]
            bw_cells = [rnn.DropoutWrapper(cell) for cell in bw_cells]

            encoder_outputs, encoder_state_fw, encoder_state_bw = tf.contrib.rnn.stack_bidirectional_dynamic_rnn(
                fw_cells, bw_cells, self.encoder_emb_inp,
                sequence_length=self.X_len, time_major=True, dtype=tf.float32)
            self.encoder_output = tf.concat(encoder_outputs, 2)
```

```

encoder_state_c = tf.concat((encoder_state_fw[0].c, encoder_state_bw[0].c), 1)
encoder_state_h = tf.concat((encoder_state_fw[0].h, encoder_state_bw[0].h), 1)
self.encoder_state = rnn.LSTMStateTuple(c=encoder_state_c, h=encoder_state_h)

with tf.name_scope("decoder"), tf.variable_scope("decoder") as decoder_scope:
    decoder_cell = self.cell(self.num_hidden * 2)

    if not forward_only:
        attention_states = tf.transpose(self.encoder_output, [1, 0, 2])
        attention_mechanism = tf.contrib.seq2seq.BahdanauAttention(
            self.num_hidden * 2, attention_states, memory_sequence_length=self.X_len, normalize=True)
        decoder_cell = tf.contrib.seq2seq.AttentionWrapper(decoder_cell, attention_mechanism,
            attention_layer_size=self.num_hidden * 2)
        initial_state = decoder_cell.zero_state(dtype=tf.float32, batch_size=self.batch_size)
        initial_state = initial_state.clone(cell_state=self.encoder_state)
        helper = tf.contrib.seq2seq.TrainingHelper(self.decoder_emb_inp, self.decoder_len,
            time_major=True)
        decoder = tf.contrib.seq2seq.BasicDecoder(decoder_cell, helper, initial_state)
        outputs, _, _ = tf.contrib.seq2seq.dynamic_decode(decoder, output_time_major=True,
            scope=decoder_scope)
        self.decoder_output = outputs.rnn_output
        self.logits = tf.transpose(
            self.projection_layer(self.decoder_output), perm=[1, 0, 2])
        self.logits_reshape = tf.concat(
            [self.logits, tf.zeros([self.batch_size, summary_max_len - tf.shape(self.logits)[1],
                self.vocabulary_size])], axis=1)
    else:
        tiled_encoder_output = tf.contrib.seq2seq.tile_batch(
            tf.transpose(self.encoder_output, perm=[1, 0, 2]), multiplier=self.beam_width)
        tiled_encoder_final_state = tf.contrib.seq2seq.tile_batch(self.encoder_state,
            multiplier=self.beam_width)
        tiled_seq_len = tf.contrib.seq2seq.tile_batch(self.X_len, multiplier=self.beam_width)
        attention_mechanism = tf.contrib.seq2seq.BahdanauAttention(
            self.num_hidden * 2, tiled_encoder_output, memory_sequence_length=tiled_seq_len,
            normalize=True)
        decoder_cell = tf.contrib.seq2seq.AttentionWrapper(decoder_cell, attention_mechanism,
            attention_layer_size=self.num_hidden * 2)
        initial_state = decoder_cell.zero_state(dtype=tf.float32,
            batch_size=self.batch_size * self.beam_width)
        initial_state = initial_state.clone(cell_state=tiled_encoder_final_state)
        decoder = tf.contrib.seq2seq.BeamSearchDecoder(
            cell=decoder_cell,
            embedding=self.embeddings,
            start_tokens=tf.fill([self.batch_size], tf.constant(start_symbol_id)),
            end_token=tf.constant(end_symbol_id),
            initial_state=initial_state,
            beam_width=self.beam_width,
            output_layer=self.projection_layer
        )
        outputs, _, _ = tf.contrib.seq2seq.dynamic_decode(
            decoder, output_time_major=True, maximum_iterations=summary_max_len,
            scope=decoder_scope)
        self.prediction = tf.transpose(outputs.predicted_ids, perm=[1, 2, 0])

with tf.name_scope("loss"):
    if not forward_only:
        crossent = tf.nn.sparse_softmax_cross_entropy_with_logits(
            logits=self.logits_reshape, labels=self.decoder_target)
        weights = tf.sequence_mask(self.decoder_len, summary_max_len, dtype=tf.float32)
        self.loss = tf.reduce_sum(crossent * weights / tf.to_float(self.batch_size))

    params = tf.trainable_variables()
    gradients = tf.gradients(self.loss, params)
    clipped_gradients, _ = tf.clip_by_global_norm(gradients, 5.0)
    optimizer = tf.train.AdamOptimizer(self.learning_rate)
    self.update = optimizer.apply_gradients(zip(clipped_gradients, params),
        global_step=self.global_step)

```

9 Метрика ROUGE

Для оценки качества работы модели трансляции для составления пересказа текста используется метрика *ROUGE*. Она сравнивает снерерированный моделью пересказ A с пересказом B , составленным человеком. В основе этой метрики лежат метрики точность и полнота (precision и recall). Для метрики *ROUGE* – N рассматриваются все N -граммы предсказанного и настоящего пересказов. Затем вычиляется количество k_N N -грамм предсказанного текста, которые присутствуют среди N -грамм настоящего пересказа. Полнота вычисляется как отношение k_N к длине n предсказанного пересказа:

$$ROUGE - N_{recall} = \frac{k}{n}$$

Точность вычисляется как отношение k_N к длине m исходного пересказа:

$$ROUGE - N_{precision} = \frac{k}{m}$$

С помощью $ROUGE - N_{recall}$ и $ROUGE - N_{precision}$ вычисляется $f1$ -мера $ROUGE - N_{f1}$:

$$ROUGE - N_{f1} = \frac{2ROUGE - N_{recall}ROUGE - N_{precision}}{ROUGE - N_{recall} + ROUGE - N_{precision}}$$

Чаще всего рассматриваются совпадение слов и биграмм, то есть вычисляются метрики $ROUGE - 1$ и $ROUGE - 2$.

Еще одним важным видом метрики *ROUGE* является метрика $ROUGE - L$. Для ее вычисления рассматриваются не N -граммы, а наибольшая общая подпоследовательность слов среди последовательностей слов в текстах предсказанного и исходных пересказов. Пусть $LCS(A, B)$ - длина (количество слов) наибольшей общей подпоследовательности (longest common subsequence), тогда $ROUGE - L_{recall}$ и $ROUGE - L_{precision}$ вычисляются как:

$$ROUGE - N_{recall} = \frac{LCS(A, B)}{n}$$

$$ROUGE - N_{precision} = \frac{LCS(A, B)}{m}$$

А $f1$ -мера $ROUGE - L_{f1}$, соответственно, как:

$$ROUGE - L_{f1} = \frac{2ROUGE - L_{recall}ROUGE - L_{precision}}{ROUGE - L_{recall} + ROUGE - L_{precision}}$$

10 Результаты

Для сравнения моделей вычленим три меры $ROUGE - 1$, $ROUGE - 2$ и $ROUGE - L$ для всех пар предсказанных и исходных текстов $\{(A_i, B_i)\}_{i=1}^m$ из тестового набора данных длиной m . Получим последовательность метрик $\{ROUGE - 1(A_i, B_i)\}_{i=1}^m$, $\{ROUGE - 2(A_i, B_i)\}_{i=1}^m$, $\{ROUGE - L(A_i, B_i)\}_{i=1}^m$. Итоговые метрики вычисляем как среднее по тестовому набору данных метрик $ROUGE$:

$$ROUGE - 1 = \frac{\sum_{i=1}^m ROUGE - 1(A_i, B_i)}{m}$$

$$ROUGE - 2 = \frac{\sum_{i=1}^m ROUGE - 2(A_i, B_i)}{m}$$

$$ROUGE - L = \frac{\sum_{i=1}^m ROUGE - L(A_i, B_i)}{m}$$

Ниже указаны полученные результаты для исследуемых моделей:

Модель	$ROUGE - 1$	$ROUGE - 2$	$ROUGE - L$
Без механизма внимания	15.38	5.69	14.49
С механизмом внимания Льюнга	18.90	7.34	17.69
С механизмом внимания Богданау	19.59	8.99	18.53

Все три модели были обучены на более чем ста итерациях. В таблицах указан лучший результат по совокупности метрик, достигнутый на тестовом наборе данных, для каждой модели.

Примеры предсказаний модели трансляции с механизмом внимания Богданау, достигшей лучшего результата:

1. Полный текст:

hong kong share prices finished the morning session sharply higher monday on a fresh inflow of funds and gains across the region as further wall street gains adding to the momentum, dealers said.

- (a) Исходный текст пересказа:

hong kong shares end morning higher on fund inflows

- (b) Предсказанный текст пересказа:
hong kong stock prices up percent

2. Полный текст:

the head of the swiss central bank, jean-pierre roth, has been appointed the new chief of the bank for international settlements, the bis said on monday .

- (a) Исходный текст пересказа:
swiss central banker named as new head of bis
- (b) Предсказанный текст пересказа:
swiss bank announces new million dollars

3. Полный текст:

a serb war crimes fugitive and his son were injured thursday in a gunfight with eu peacekeepers in eastern bosnia, local police said.

- (a) Исходный текст пересказа:
wanted bosnian serb son injured in gunfight with eu troops
- (b) Предсказанный текст пересказа:
serbs injured by peacekeeper

11 Вывод

Задача составления пересказа текста является одной из самых интересных и непростых среди задач анализа текстов. Поэтому для ее решения была разработана нетривиально устроенная архитектура нейронной сети - модель трансляции. Используя знания, полученные в рамках работы над курсовыми предыдущих лет, я выбрала для разработки модели трансляции самые удачные из известных мне подходы обработки текстовых данных, получения векторных представлений слов и комбинации различных видов рекуррентных нейронных клеток. После чего я запрограммировала базовую модель трансляции, обучила и протестировала ее на подмножестве набора данных English Gigaword.

Затем были изучены теоретически, а потом запрограммированы две модели с разными видами механизма внимания. Обе модели показали

результат на несколько процентов лучший, чем базовая модель. Это говорит о том, что при предсказании различных слов пересказа разный контекст имеет большую значимость. Лучший результат по совокупности метрик показала модель с механизмом внимания Богданау.

Хоть качество предсказания модели остается не очень высоким, были выявлены следующие тенденции. Во-первых, для решения данной задачи требуется сложная модель, обладающая большим множеством параметров. Так в ходе экспериментов было замечено, что увеличение количества рекуррентных слоев в кодировщике, увеличение размерностей выходных векторов и векторных представлений слов улучшало качество предсказания. Механизм внимания также добавляет в модель дополнительные параметры и значительно влияет на ее качество. Из этого следует, что, во-вторых, обучение модели требует большого набора текстовых данных. К тому же при визуальном анализе предсказаний было замечено, что модель неплохо научилась строить предсказания для текстов самых распространенных тематик: новости об изменениях на фондовом рынке, новости о количестве жертв в ходе чрезвычайного происшествия, новости об избрании лиц в органы власти. И, в-третьих, модель в большинстве случаев использовала в предсказанном пересказе слова, соответствующие основному действующему лицу или предмету текста (подлежащие - названия стран, предприятий, произошедшие явления), но испытывала проблемы с логикой построения и смыслом предложения. Это наводит на мысль о том, что данная модель может быть применена для задачи поиска ключевых слов в тексте.

12 Список литературы

1. Word2Vec Parameter Learning Explained, Xin Rong
<https://arxiv.org/abs/1411.2738>
2. Understanding LSTM Networks, Christopher Olah
<https://colah.github.io/posts/2015-08-Understanding-LSTMs>
3. Neural Machine Translation by Jointly Learning to Align and Translate, Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio
<https://arxiv.org/abs/1409.0473>
4. Effective Approaches to Attention-based Neural Machine Translation, Minh-Thang Luong, Hieu Pham, Christopher D. Manning
<https://arxiv.org/abs/1508.04025>
5. Sequence to Sequence Learning with Neural Networks, Ilya Sutskever, Oriol Vinyals, Quoc V. Le
<https://arxiv.org/abs/1409.3215>
6. Seq2Seq with Attention and Beam Search, Guillaume Genthial
<https://guillaumegenthial.github.io/sequence-to-sequence>