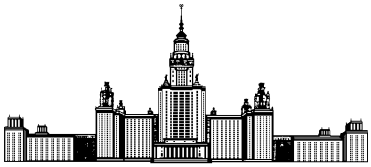
Московский государственный университет имени М. В. Ломоносова



Факультет Вычислительной Математики и Кибернетики
Кафедра Математических Методов Прогнозирования

ЭКЗАМЕНАЦИОННАЯ РАБОТА СТУДЕНТА 317 ГРУППЫ

ПО КУРСУ: DEEP LEARNING 2021

«Генерация текста (NLG)»

Выполнил:
студент 3 курса 317 группы
Елистратов Семен Юрьевич

Научный руководитель:
к.ф-м.н., доцент
Китов Виктор Владимирович

Москва, 2021

Содержание

Аннотация

В работе рассматриваются рассматриваются основные подходы для решения задач представления слов и саммаризации. Проведен обзор современных архитектур нейронных сетей, предназначенных для их решения, а также описаны метрики для оценки качества моделей в задаче саммаризации.

1. Введение

Генерация текста (Natural Language Generation - NLG) - одна из подзадач обработки естественного языка (NLP). Она заключается в генерации некоторого текста на естественном языке при некоторых заданных условиях. С развитием машинного обучения оказалось, что глубокие нейронные сети хорошо справляются с данной задачей. В последнее время это одна из самых бурно развивающихся тем в глубоком обучении, которая таит в себе множество перспектив.

В данной статье рассмотрены основные задачи генерации текстов: представление слов и саммаризация, и описаны модели решающие данные задачи.

2. Представление слов

Обработка естественного языка довольно сложная задача, так как, во-первых, входные данные (текст) не имеют фиксированной длины, а, во-вторых, модели машинного обучения работают с некоторым векторным представлением. При преобразовании текста в понятный для модели формат необходимо выделить как можно больше информации из входных данных. Для этого текст разбивается на токены, и каждый токен имеет некоторое векторное представление. В качестве токенов можно просто взять слова, однако тогда модель не будет учитывать смысл морфем, например, приставок под, над и др. Слова можно представлять посимвольно, но каждый символ не несет в себе определенной смысловой нагрузки, и существуют слова схожие по написанию, но различные по значению. Кроме того, модель также должна обобщаться на слова, которые не были представлены в обучающей выборке (Out of vocabulary - OOV).

Далее будут подробнее рассмотрены три основных подхода к представлению слов: токенизация на подслова, посимвольные методы и гибридные подходы.

2.1.Токенизация на подслова

Данный подход основан на разбиении слов на подслова. Например слово «подлодка» будет разбито на два токена «под» + «лодка». Таким образом алгоритм будет учитывать смысл различных морфем (приставок, суффиксов, окончаний).

2.1.1. Byte Pair Encoding (BPE)

Данный метод был впервые предложен в статье [1] для сжатия данных. Он заключается в том, что итеративно находятся самые популярные последовательные пары байтов и заменяются некоторым новым неиспользуемым байтом. Данный алгоритм можно использовать и для задачи токенизации текста [2]. Изначально словарь токенов инициализируется алфавитом символов и специальным символов конца «·», и каждое слово представляется просто как последовательность символов. Далее итеративно находятся наиболее популярные последовательные пары символов («А»,

«B») и заменяются на новый символ («AB»), который добавляется в словарь. Алгоритм останавливается, когда размер словаря превосходит значение некоторого порога (что эквивалентно заданию числа итераций, так как на каждой итерации в словарь добавляется один новый символ). Кроме того, новые сгенерированные токены являются интерпретируемыми цепочками символов (морфемами), и алгоритм на их основе может генерировать новые слова.

Также, при генерации новых токенов алгоритм помечает их специальными символами в зависимости от их положения в слове. Сеть GPT-2 использует данный подход на более низком уровне - на уровне байтов, а также отличает символы разного типа (буквы, символы пунктуации).

Algorithm 1: BPE

initialization vocabulary $\mathcal{V}$;
$t \leftarrow \mathcal{V}$ size threeshold;
while $|\mathcal{V}| < t$ **do**
  $\forall$ consecutive pairs x_i, x_j count frequency a_{ij};
  $\mathcal{V}$ add $x_i x_j$ with $\max a_{ij}$
end

Данный метод обладает следующими недостатками: во-первых, для каждого слова однозначно определяются его токены, что усиливает переобучение, во-вторых, для редких слов токены будут менее осмысленными. Один из методов решающих эту проблему - BPE Dropout [3]. На каждой итерации составляется множество всевозможных объединений, и с заданной вероятностью p удаляются некоторые из них. Таким образом, алгоритм становится более случайным и имеет лучшую обобщающую способность. При $p = 0$ метод превращается в классический BPE, а при $p = 1$ будет удалять все объединения, что приведет к чисто посимвольному подходу.

Algorithm 2: BPE-dropout

initialization vocabulary $\mathcal{V}$;
$t \leftarrow \mathcal{V}$ size threeshold;
while $|\mathcal{V}| < t$ **do**
  optimize $p(x)$ with EM;
  $\forall u_i \in merges$ count frequency a_i;
  $\mathcal{V}$ add u_i with $\max a_i$;
end

2.1.2. WordPiece

Данный подход [4] похож на BPE, но при объединении токенов максимизируется правдоподобие языковой модели, а не частота. Алгоритм можно описать так: для большого корпуса документов составляется словарь из unicode символов, настраивается языковая модель на обучающей выборке и подбираются пороги на правдоподобие и размер словаря. Далее итеративно выбирается новый токен, получаемый объединением существующих, добавляется в обучающую выборку, и вычисляется ее

правдоподобие с помощью языковой модели. И так на каждом шаге выбирается токен с максимальным правдоподобием, пока размер словаря или значение правдоподобия не превысят соответственные пороги. Данный подход использовался в модели BERT, однако в ее модификации RoBERT было показано, что он не превосходит по качеству BPE. Правдоподобие вычисляется по формуле:

$$\log p(x, y) - \log p(x) - \log p(y) = \log \frac{\log p(x)}{\log p(x) \log p(y)}. \tag{1}$$

2.1.3. Unigram Language Model

Метод был предложен в статье [5] и основан на предположении, что все подслова встречаются независимо, и вероятность цепочки подслов $\mathbf{x} = (x_1, ..., x_M)$ можно представить в виде произведения вероятностей появления каждого подслова:

$$P(\mathbf{x}) = \prod_{i=1}^{M} p(x_i), \forall i x_i \in \mathcal{V}, \sum_{x \in \mathcal{V}} p(x) = 1. \tag{2}$$

Тогда наиболее вероятную сегментацию можно выбрать, как:

$$\mathbf{x}^* = \arg \max_{\mathbf{x} \in \mathcal{S}(X)} P(\mathbf{x}), \tag{3}$$

где X - предложение полученное на вход, $\mathcal{S}(X)$ - множество объединений - кандидатов для сегментации. Для заданного словаря вероятности $p(x_i)$ вычисляются с помощью EM-алгоритма. Так как словарь подслов изначально неизвестен, можно взять достаточно большой словарь сгенерированный с помощью BPE. Алгоритм можно описать так:

Algorithm 3: Unigram language model

initialization big vocabulary $\mathcal{V}$;
$t \leftarrow \mathcal{V}$ size threeshold;
while $|\mathcal{V}| < t$ **do**
    $merges \leftarrow$ all possible merges;
    $\forall x_i \in \mathcal{V}$ compute $loss_i$;
    sort x_i by $loss_i$; keep top-ν (0.8) subwords by $loss_i$;
end

$loss_i$ - насколько изменится правдоподобие при удалении слова x_i из словаря. Кроме того, алгоритм всегда оставляет подслова, состоящие из одного символа, чтобы иметь возможность работать со словами, не попавшими в обучающую выборку (OOV).

Существует библиотека SentencePair [6], в которой реализованы предложенные методы. Кроме того, она учитывает кол-во пробелов между словами и другие специальные символы, что позволяет извлекать больше информации из текста.

2.2. Посимвольные методы

2.2.1. Compositional Character Model

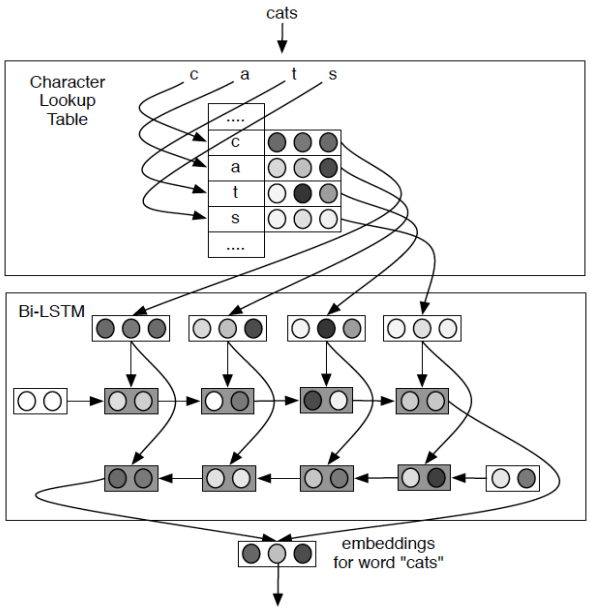


Рис. 1: Compositional Character Model [7]. Сеть состоит из двунаправленной LSTM, которая получает на вход последовательность векторов - представлений символов.

Другой способ представления слов - посимвольный. Общая идея таких методов - использовать специальную сеть, которая получает на вход слово, как цепочку символов, и выдает для него некоторое векторное представление. Один из методов - Compositional Character Model [7]. В нем каждый символ кодируется one-hot вектором, получаемым из алфавита корпуса, который включает в себя как буквы различных регистров, так и знаки пунктуации и цифры. Далее цепочка этих векторов подается на вход двунаправленной LSTM модели, которая преобразует эту последовательность в вектор, описывающий данное слово. Тогда полученное представление слова после примененния полносвязного слоя можно записать как:

$$e_w = W_1 h^f_{last} + W_2 h^b_{first} + b, \tag{4}$$

где W_1, W_2, b - параметры полносвязного слоя, h^f_{last}, h^b_{first} последнее скрытое состояние при прямом проходе и первое скрытое состояние при обратном проходе соответственно. Так как вычисление e_w затратно, то модель хранит значения для наиболее часто встречающихся слов и производит полные вычисления только для редких слов.

2.2.2. Character-Aware NLM

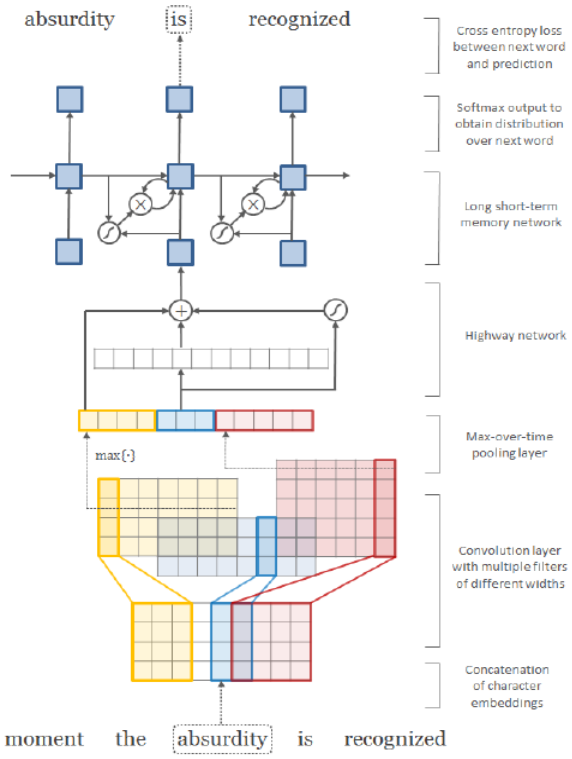


Рис. 2: Character-Aware NLM [8]. Модель состоит из сверточной сети, генерирующей векторное представление слов, Highway network [26] и RNN.

Другая модель - Character-Aware NLM [8]. Каждый символ кодируется некоторым вектором длины d, причем использование более коротких векторов, чем one-hot, дает лучший результат. Пусть слово k состоит из l символов. Тогда k можно представить в виде матрицы $C^k \in \mathbb{R}^{d \times l}$. Далее к данной матрице применяется свертка $H \in \mathbb{R}^{d \times w}$ шириной w, сдвиг и нелинейность. Тогда i-й элемент полученной карты признаков можно представить как:

$$f^k[i] = tanh(\langle C^k[*, i : i + w - 1], H \rangle + b). \tag{5}$$

Далее применяется операция максимизации относительно времени:

$$y^k = \max_i f^k[i]. \tag{6}$$

Модель применяет $h \in [100, 1000]$ сверток с различной шириной и конкатенирует полученные результаты, генерируя некоторое представление слова k: $\mathbf{y} = [y_1^k, ..., y_h^k]$. Далее это представление поступает на вход Highway сети [26]:

$$\mathbf{z} = \mathbf{t} \odot g(W_H \mathbf{y} + b_H) + (1 - \mathbf{t}) \odot \mathbf{y}, \tag{7}$$

где $\mathbf{t} = \sigma(W_T \mathbf{y} + b_T)$, а $g(\cdot)$ - нелинейная функция. Далее $\mathbf{z}$ поступает на вход RNN, которая генерирует следующее слово.

2.3. Гибридный подход

Гибридный подход заключается в том, что в обычном режиме модель работает на уровне слов, а для редких или тех слов, которые не попали в словарь (OOV), используется посимвольное представление. Пример модели: Hybrid NMT [9].

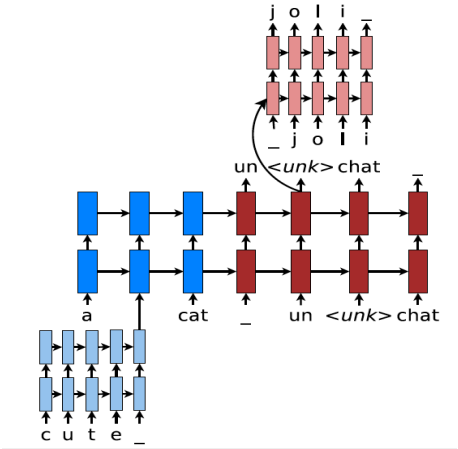


Рис. 3: Hybrid NMT [9]. Сеть имеет архитектуру: LSTM кодировщик-декодировщик. Модель работает на уровне слов, а для OOV - на уровне символов.

Благодаря данному подходу модель имеет хорошую обобщающую способность и может «понимать» слова, не попавшие в обучающую выборку.

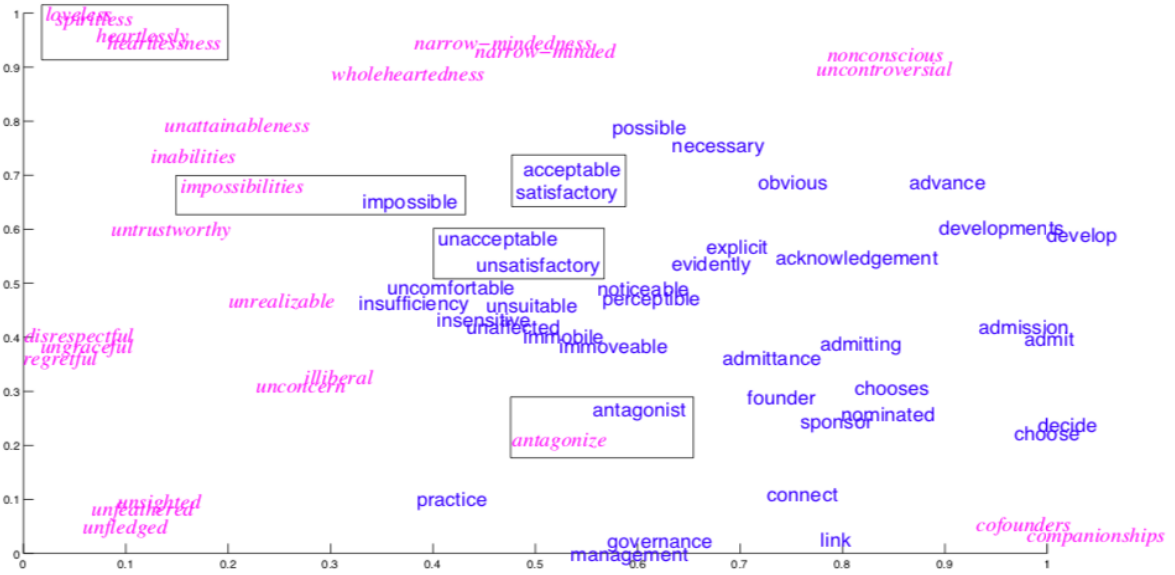


Рис. 4: Визуализация представлений слов с помощью Barnes-Hut-SNE [9]. Синим цветом выделены часто встречающиеся слова, фиолетовым - редкие. Таким образом, видно, что посимвольное представление правильно определяет смысл и семантику слов (похожие слова, даже для различных подходов к представлению, расположены близко друг к другу: impossibilities - impossible, antagonize - antagonist).

Идея модели Char2Vec [10] заключается в попытке научить модель понимать морфологические признаки слов, явно добавляя морфологию в посимвольные модели. Для этого сначала обучается модель skip-gram с отрицательным сэмплированием (word2vec), которая для словаря V обучает две матрицы векторных представлений $W, C \in \mathbb{N} \times \mathbb{V}$ для целевого и контекстного векторов (N - размер векторного представления). Каждый раз, когда слово w встречается вместе со словом c таблицы, обновляются по формуле:

$$\log \sigma(f(w) \cdot c) + \sum_{i=1}^{k} \mathbb{E}_{\tilde{c}_i \sim P(w)}[\log \sigma(-f(w) \cdot \tilde{c}_i)], \tag{8}$$

где $P(w)$ - некоторое шумовое распределение, используемое для отрицательного сэмплирования, $f(w)$ обучаемая функция, получающая на вход цепочку символов и выдающая векторное представление слова. $f(w)$ вычисляется с помощью двунаправленной LSTM с механизмом внимания и линейным слоем после конкатенации. Таким образом, благодаря тому, что модель учитывает контекст, она лучше понимает смысл, семантику и морфологию слов.

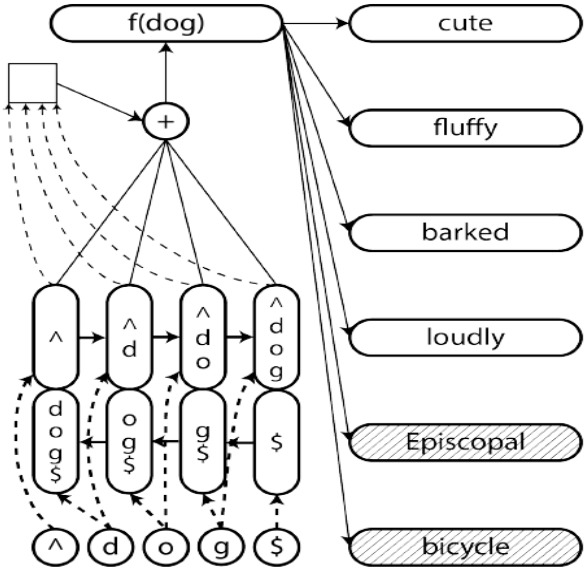


Рис. 5: Char2Vec [10]. Двунаправленная LSTM преобразует слово как цепочку символов в некоторое векторное представление, которое потом с помощью word2vec учитывает морфологические особенности слова.

3. Саммаризация текста

Задача саммаризации заключается в создании краткого содержания текста, то есть в выделении основных мыслей. Существует два типа саммаризации:

- Экстрактивная саммаризация: выделение важных предложений непосредственно в тексте.
- Абстрактивная саммаризация: генерация нового текста, описывающего основную мысль исходного.

Вторая задача значительно сложнее первой, так как имеет гораздо меньше ограничений.

Абстрактивную саммаризацию можно рассматривать как задачу машинного перевода. Модель получает на вход некоторый текст и на выходе также выдает некоторый другой текст. Поэтому можно использовать классическую модель для машинного перевода: seq2seq.

3.1. seq2seq + attention

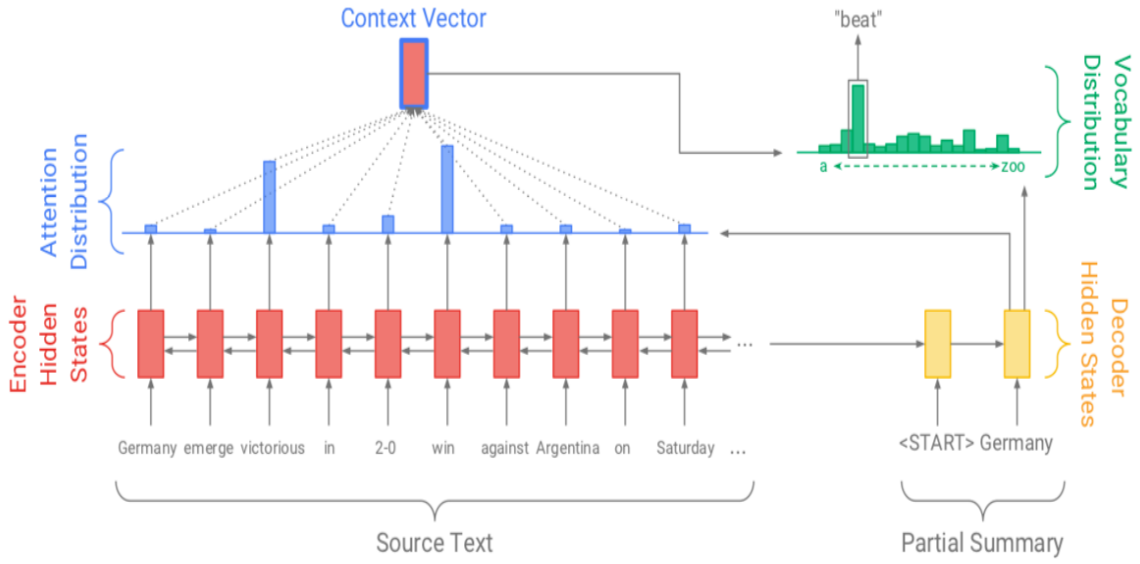


Рис. 6: Модель для машинного перевода seq2seq + attention [11].

Данная модель имеет архитектуру кодировщик-декодировщик. В качестве кодировщика берется двунаправленная LSTM, которая получает на вход текст и выдает некоторое представление слова. Декодировщик имеет архитектуру однонаправленной LSTM и получает на вход векторное представление предыдущего слова, а также контекстный вектор внимания. Механизм внимания позволяет определить распределение вероятностей в входной последовательности, и на основе этого распределения

вычисляется взвешенная сумма скрытых состояний энкодера, которая конкатенируется вместе с состоянием декодера и пропускается через два линейных слоя.

Эта модель, предназначенная для машинного перевода, достаточно неплохо справляется с саммаризацией, однако есть некоторые характерные особенности данной задачи, которые могут значительно повысить качество. Во-первых, модель не должна выдавать ложную информацию, например, при саммаризации предложения: «Германия обыграла Аргентину 2:0», модель не должна выдавать неправильное значение счета. Для этого можно использовать специальный механизм копирования, который заключается в том, что с некоторой вероятностью слова будут сгенерированы, а не копированы из входного текста, причем эта вероятность зависит от скрытых состояний модели:

$$p_{gen} = \sigma(w_{h^*}^T h_t^* + w_s^T s_t + w_x^T x_t + b_{ptr}), \tag{9}$$

где $w_{h^*}, w_s, w_x, b_{ptr}$ - обучаемые параметры. Тогда функция распределения для расширенного словаря:

$$P(w) = p_{gen} P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} a_i^t, \tag{10}$$

Кроме того, модель может зациклиться в смысле того, что будет описывать только один какой-то факт, следовательно необходимо правильное выделение контента (иерархическое внимание).

3.2. Pointer-generator network

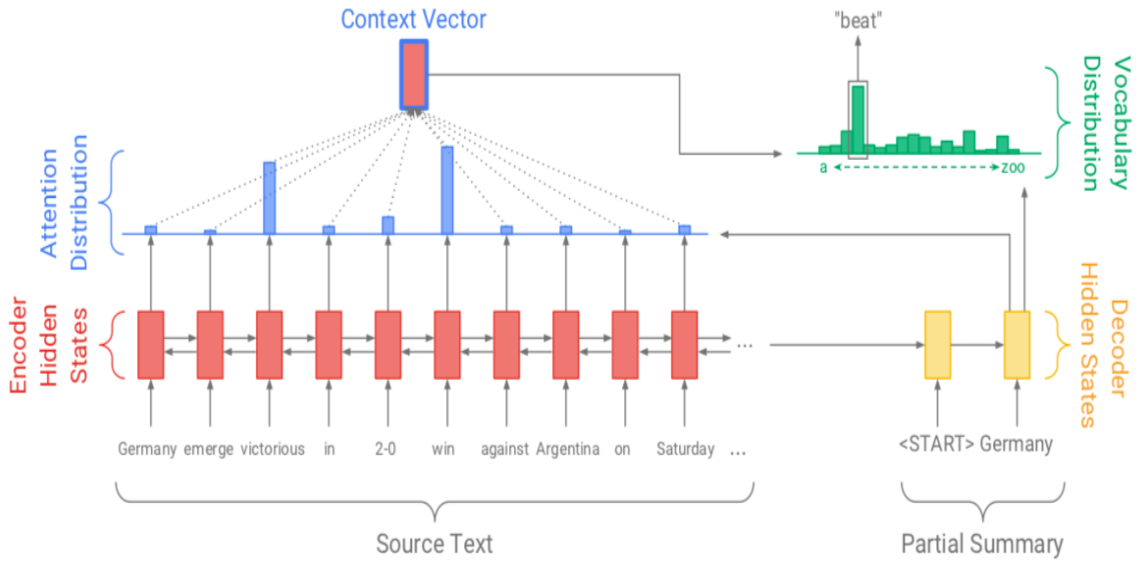


Рис. 7: Pointer-generator network [11] - модификация модели seq2seq+attention. В ней реализован механизм копирования и механизм покрытия.

В данной модели [11] реализованы две модификации предыдущего алгоритма: копирование и покрытие. Механизм копирования описан в прошлом пункте, рассмотрим концепцию покрытия.

Покрытие позволяет избежать такой ошибки модели, как повторение слов и повторение описаний фактов и позволяет разнообразить получаемый текст. Данный подход заключается в том, что вычисляется некоторый вектор покрытия:

$$c^t = \sum_{t'=0}^{t-1} a^{t'}, \tag{11}$$

равный сумме векторов внимания относительно времени, который влияет на вычисление вектора внимания в дальнейшем:

$$e_i^t = \nu^T \tanh(W_h h_i + W_s s_t + w_c c_i^t + b_{attn}). \tag{12}$$

Таким образом, вектор внимания строится так, чтобы меньше обращать внимания на уже просмотренные области. Кроме того, вводится дополнительная функция потерь:

$$covloss_t = \sum_i \min(a_i^t, c_i^t). \tag{13}$$

Благодаря добавлению покрытия модель выделяет главные мысли равномерно со всего текста. А механизм копирования позволяет точнее передавать основные факты.

3.3 Bottom-up summarization

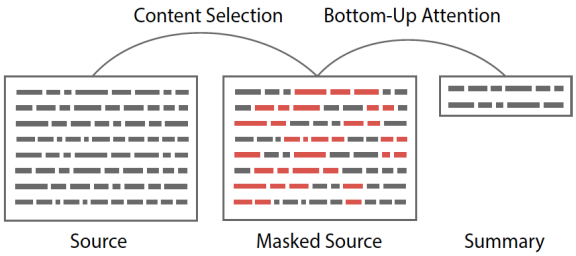


Рис. 8: Bottom-up summarization [12]. Данный подход сначала выделяет контент, который будет использоваться моделью в дальнейшем.

Саммаризация обычно проводится для достаточно больших по объему текстов, и механизм внимания может неправильно выбирать контент для саммаризации. Существует достаточно простой подход для решения этой проблемы. Модель Bottom-up summarization [12] состоит из двух стадий: на первой стадии алгоритм проходит по всему тексту и помечает тегами слова, которые не будут учитываться в дальнейшем, а на второй стадии уже работает механизм внимания и кодировщик-декодировщик. Первая стадия решается с помощью двунаправленной LSTM, а также двух моделей для представления слов ELMo [27] и GLoVE [28].

3.4. SummaRuNNer

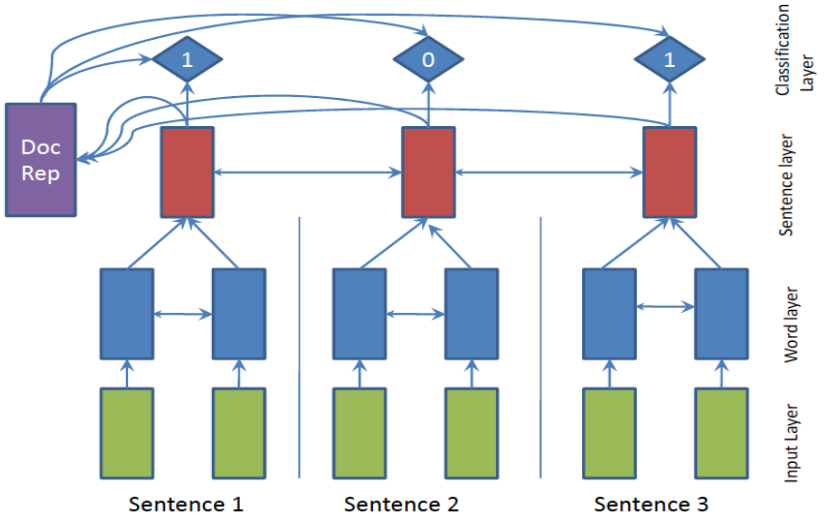


Рис. 9: SummaRuNNer [13]. RNN модель для экстрактивной саммаризации.

Это одна из первых RNN моделей для экстрактивной саммаризации [13]. Она состоит из двух двунаправленных GRU-RNN, работающих на уровнях слов и предложений, а также имеет линейный слой, который получает на вход скрытые состояния сети с уровня предложений и агрегирует информацию обо всем документе:

$$\mathbf{d} = \tanh(W_d \frac{1}{N_d} \sum_{j=1}^{N_d} [h_j^f, h_j^b] + b), \tag{14}$$

где h^f, h^b скрытые состояние слоя GRU-RNN, работающего с предложениями, при прямом и обратном проходе соответственно. Далее эта информация вместе с данными с уровня предложений поступает на слой для классификации (оставлять предложение или нет), который вычисляет вероятность как:

$$P(y_j = 1 | \mathbf{h}_j, \mathbf{s}_j, \mathbf{d}) = \sigma(W_c \mathbf{h}_j + \mathbf{h}_j^T W_s \mathbf{d} - \mathbf{h}_j^T W_r tanh(\mathbf{s}_j) + W_{ap} \mathbf{p}_j^a + W_{rp} \mathbf{p}_j^r + b), \tag{15}$$

где каждое слагаемое отвечает, соответственно, за: информацию о контенте в j-м предложении, значимость предложения по отношению к документу, избыточность предложения по отношению к текущему состоянию документа (то есть насколько данное предложения ввело новой информации) и за абсолютное и относительное положение предложения относительно документа.

3.5. TCONVS2S

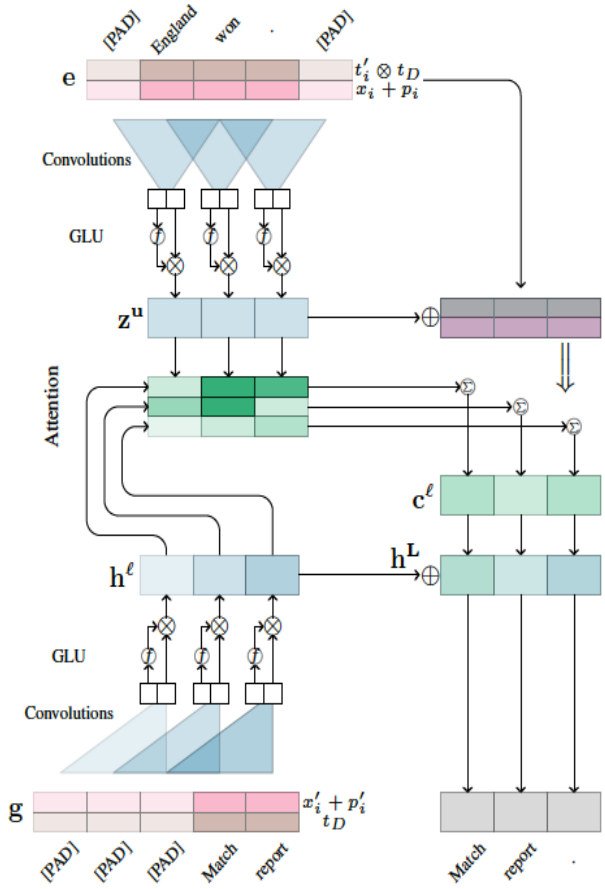


Рис. 10: TCONVS2S [14]. Сверточная архитектура для саммаризации.

Цель данной модели [14] - понять текст и ответить на вопрос: «о чем текст?». Архитектура данной сети изначально предназначалась для задачи машинного перевода. Модель состоит из сверточного кодировщика (расположен вверху), который получает на вход текст, и с помощью двусторонних сверток преобразует его в некоторую карту признаков. Далее карта признаков проходит через GLU, которая разбивает результат свертки на две части, применяет к одной из них сигмоиду, перемножает между собой, а также добавляет прокидывающую связть (residual). Декодировщик (расположен внизу) имеет односторонние свертки. Данные с кодировщика и декодировщика проходят через GLU и идут в слой внимания, который оценивает похожесть этих данных и обучает веса.

3.6. BertSUM

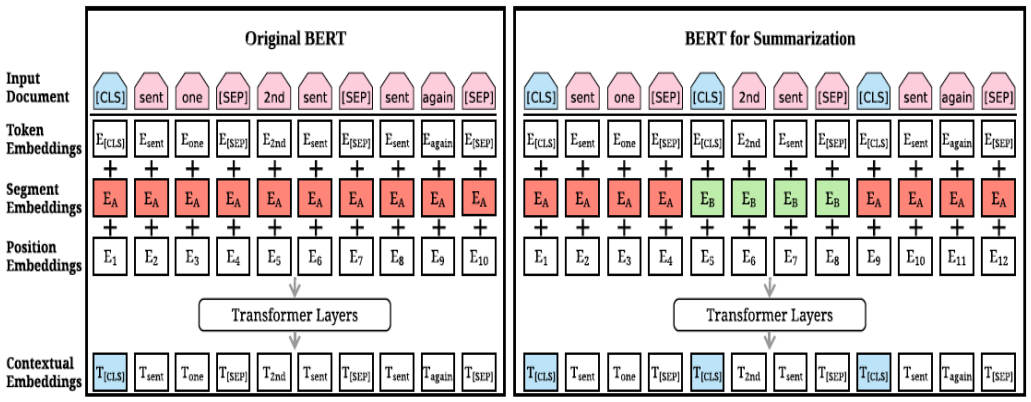


Рис. 11: Сравнение Bert и BertSum [15]. Прямой проход сверху-вниз. Основное отличие: BertSum работает с большим текстом, состоящим из множества предложений, поэтому каждое предложение необходимо разделять. Кроме того, BertSum помечает каждое предложение на основе четности его позиции в тексте для иерархического анализа текста на разных уровнях (соседние предложения, все предложения).

Данная модель [15] является модификацией популярной архитектуры BERT, являющейся трансформером. Основное отличие от классического BERT в том, что BertSUM получает на вход весь текст, который разбивается специальными токенами $[CLS]$ на предложения. Кроме того, модель разбивает все предложения на четные и нечетные в завимисости от их порядка в предложении, чтобы механизм внимания мог работать на разных уровнях: с соседними предложениями и для всего текста. Данная архитектура, с некоторыми модификациями, может использоваться как для экстрактивной, так и для абстрактивной саммаризации.

Для экстрактивной саммаризации необходимо добавить слой классификации, который вычисляет вероятность для каждого предложения как:

$$\hat{y}_i = \sigma(W_o h_i^L + b_o), \tag{16}$$

где W_o, b_o - обучаемые параметры линейного слоя, h_i^L - вектор из последнего слоя трансформера.

Для абстрактивной саммаризации в качестве декодировщика можно взять 6-слойный трансформер.

3.8. Text matching

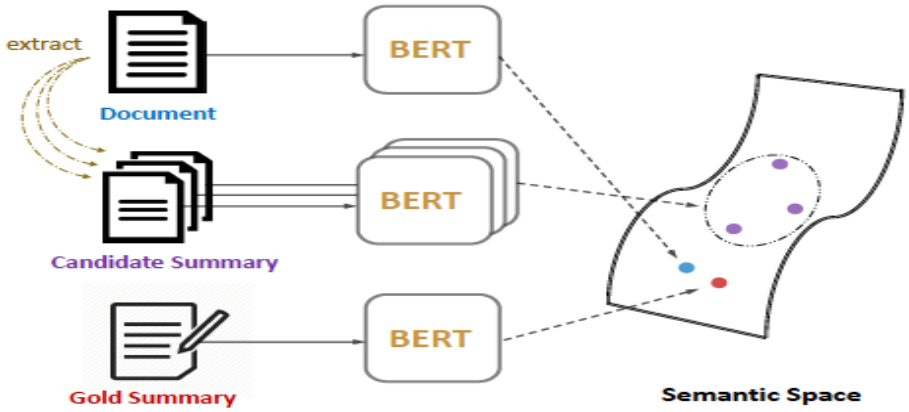


Рис. 12: Ключевая идея статьи «Extractive Summarization as Text Matching» [16]. Кандидаты на включение в саммаризацию и исходный документ отображаются в некоторое пространство, в котором более близкие по косинусной метрике объекты к документу, являются лучшими кандидатами для саммаризации.

Еще один интересный подход для экстрактивной саммаризации был предложен в статье [16]. Данный метод относится к так называемым двух-стадийным методам: на первой стадии алгоритм определяет список предложений-кандидатов для саммаризации, а на второй выбирает лучшие из них. Ключевая идея данной статьи - построить отображение исходного документа и предложенных саммаризаций в некоторое семантическое пространство, в котором простая метрика определяет качество саммаризации. В качестве модели используется сиамская BERT, которая состоит из двух классических предобученных BERT со связанными весами и слоем вычисляющим косинусную метрику выходов сетей. Пусть r_D и r_C - векторные представления полученные сетями для документа и кандидата соответственно. Тогда функция близости между этими векторами: $f(D, C) = cosine(r_D, r_D)$. Функцию потерь можно записать как:

$$\mathcal{L}_1 = \max(0, f(D, C) - f(D, C^*) + \gamma_1);$$

$$\mathcal{L}_2 = \max(0, f(D, C_i) - f(D, C_j) + (j - i)\gamma_2), (i < j);$$

$$\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2,$$

где C^* - эталон, γ_1, γ_2 - гиперпараметры модели. Порядок между кандидатами для второй функции потерь задается с помощью сортировки по $ROUGE$ метрике. То есть первая функция потерь отвечает за то, чтобы модель делала хорошие предсказания, а вторая, чтобы не было сильного разброса между кандидатами в полученном пространстве.

3.9. Reinforcement models

Ключевая идея модели описанной в статье [17] заключается в минимизации метрики $ROUGE_L$ [22]:

$$ROUGE_L = \frac{(1+\beta^2)R_{LCS}P_{LCS}}{R_{LCS}+\beta^2 P_L CS}, \qquad R_{LCS} = \frac{LCS}{|y|}, P_{LCS} = \frac{LCS}{|a|}. \tag{17}$$

LCS - наиболее длинная подпоследовательность ответа в эталоне. P_{LCS}, R_{LCS}, соответственно аналог точности и полноты. Данная метрика не является дифференцируемой, а значит не может быть оптимизирована с помощью метода градиентного спуска. Поэтому ее оптимизация проводится с помощью обучения с подкреплением (reinforcement learning). Обучение с подкреплением - один из подразделов машинного обучения, в котором агент взаимодействует со средой, пытаясь максимизировать некоторый долговременный выигрыш. Архитектура модели - кодировщик-декодировщик на каждом шаге выбирает некоторые слова для саммаризации, вычисляет метрику и в зависимости от этого понимает: правильное ли было применено решение или нет.

Модель DRESS [18] имеет возможность упрощать предложения: копировать, заменять, удалять, переставлять слова. Для каждого предложения на каждом шаге модель делает эти операции, пока не породит токен EOS. Затем она получает награду в зависимости от качества работы. Награда рассчитывается как:

$$r(\hat{Y}) = \lambda^S r^S + \lambda^R r^R + \lambda^F r^F, \tag{18}$$

где r^S, r^R, r^F отвечает за простоту, релевантность и беглость соответственно. r^S вычисляется как взвешенная сумма метрики SARI [25] (будет описана позднее). r^R вычисляется как косинусное состояние между векторами, полученными с помощью LSTM, которая получила на вход исходный текст и предсказанный. Беглость вычисляется как экспоненциал от перплексии предсказанного текста.

Для генерации изображений одним из интересных и мощных подходов является генеративно-состязательные сети (GAN). GAN состоит из генератора и дискриминатора, задача генератора «обмануть» дискриминатор, а задача дискриминатора определить какие данные эталонные, а какие сгенерированные. Данный подход можно использовать и в задаче саммаризации. В статье [19] описана генеративно-состязательная сеть для саммаризации, использующая обучение с подкреплением. Генератор состоит из кодировщика - двунаправленной LSTM и декодировщика - однонаправленной LSTM, кроме того реализован механизм внимания. Генератор выбирает токены, которые будут исползованы в саммаризации. Дискриминатор имеет сверточную архитектуру и обрабатывает пары входных данных, с целью определить какие из них сгенерированы. Оптимизация проводится с помощью метода обучения с подкреплением - Policy gradient. Он заключается в том, что в качестве функции потерь берется матожидание награды и по ней проводится оптимизация с помощью градиентного спуска. Шаг обучения дискриминатора происходит после того, как генератор на текущем этапе начинает давать более качественную саммаризацию.

TABLE I
COMPARISON OF VARIOUS MODELS ON THE CNN/DAILY MAIL DATASET USING FULL-LENGTH F1 VARIANTS OF ROUGE. (+: WITH; -: WITHOUT)

	ROUGE-1	ROUGE-2	ROUGE-L
Lead-3 ([7])	39.2	15.7	35.5
SummaRuNNer ([7])	39.6	16.2	35.3
words-lvt2k-temp-att ([9])	35.46	13.30	32.65
RL +intra-attention ([8])	**41.16**	15.75	39.08
ML+RL + intra-attention ([8])	39.87	15.82	36.90
GAN ([5])	39.92	**17.65**	36.71
Proposed method + attention	37.87	15.71	**39.20**
Proposed method - attention	34.87	14.71	33.20

TABLE II
COMPARISON OF HUMAN JUDGMENT. (+: WITH; -: WITHOUT)

	Grammatically	Relevance	Readability
ML+RL + intra-attention ([8])	7.1	7.71	**7.21**
Proposed method + attention	**7.5**	**7.9**	6.1

Рис. 13: Сравнение качества работы GAN [19] с другими методами.

3.9. GSum

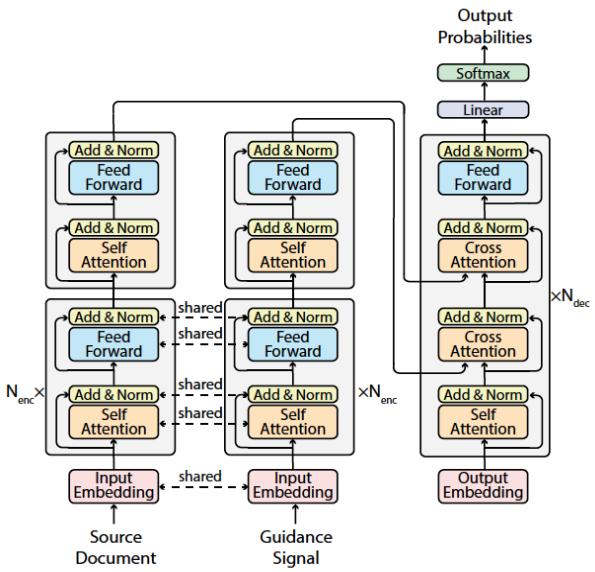


Рис. 14: Фреймворк GSum [20].

GSum [20] - фреймворк для построение моделей для управляемой абстрактивной саммаризации. Многие модели довольно хорошо решают данную задачу, однако они тяжело настраиваются и могут работать нестабильно. Данный фреймворк значительно упрощают работу с моделями абстрактивной саммаризации.

Архитектура модели основана на трансформерах (BERT или BART). Модель может получать на вход помимо исходного документа дополнительный сигнал (guidance), то есть некоторый текст, влияющий на работу модели. Это может быть, например, эталонный пример саммаризации написанный человеком. Кроме того, мо-

дель для ускорения обучения может сама генерировать пример другой «человеческой» саммаризации с помощью исходного текста и эталонного ответа. Примерами сигналов могут быть выделенные в тексте предложения, ключевые слова или некоторые зависимости. Для того, чтобы обрабатывать эти сигналы вместе с входными данными, модель имеет два кодировщика: один для документа, второй для сигнала, которые имеют общие параметры на нижних слоях сети. Декодировщик получает выход с двух кодировщиков и с помощью механизма внимания учитывает сигнал и выдает результат саммаризации.

4. Метрики качества для задачи саммаризации

Далее рассмотрим метрики, которые позволяют оценивать качество для задача саммаризации.

4.1. BLEU

Данная метрика [21] определяется как:

$$BLEU = (P_1 P_2 P_3 P_4)^{\frac{1}{4}}, \tag{19}$$

где P_n - точность вхождения n-грамм (иногда число n-грамм) в эталонный перевод. Существует модификация GLEU (Google BLEU), где вместо точности берется минимум между точностью и полнотой:

$$GLEU = (M_1 M_2 M_3 M_4)^{\frac{1}{4}}, M_n = \min(P_n, R_n). \tag{20}$$

4.2. ROUGE

ROUGE [22] - семейство метрик для оценки саммаризации. Например, $ROUGE_n$ определяется как отношение числа общих n-грамм, к числу n-грамм в правильном ответе. Другая метрика:

$$\begin{aligned} ROUGE_L &= \frac{(1+\beta^2) R_{LCS} P_{LCS}}{R_{LCS} + \beta^2 P_L CS}, \\ R_{LCS} &= \frac{LCS}{|y|}, P_{LCS} = \frac{LCS}{|a|}. \end{aligned} \tag{21}$$

LCS - наиболее длинная подпоследовательность ответа в эталоне. P_{LCS}, R_{LCS} соответственно аналог точности и полноты. Существует множество других модификаций данных метрик: взвешеный $ROUGE_w$, $ROUGE_s$ использующий биграммы и др.

4.3. METEOR

Построим соответствия слов в эталонном ответе и предсказанным моделью. Назовем чанком максимальную последовательность слов переходящих в эту же после-

довательность. Тогда данная метрика [23] может вычисляться как:

$$METEOR = \frac{10RP}{R+9P}(1 - 0.5(\frac{\#chunks}{\#owords})^3), \tag{22}$$

где $\#chunks$ - число чанков, $\#owords$ - число общих слов, P, R - точность и полнота.

4.4. TER

Метрика [24], основанная на редакторском расстоянии. Вычисляется число вставок, удалений, замен слов и переноса отрезков и делится на число слов в эталоне. Если правильных ответов несколько, то в знаменателе берется среднее от числа слов в ответах.

4.5. SARI

Данная метрика [25] основана на идее о том, что необходимо кроме эталонного ответа учитывать также и исходный текст. В качестве ее значения берется взвешенная сумма:

$$SARI = d_1F_{add} + d_2F_{keep} + d_3F_{del}. \tag{23}$$

То есть вычисляется число вставок, удалений и сохранений слов и нормируется на пересечение эталонного ответа и входного текста.

5. Заключение

В данной статье проведен обзор основных задач генерации текста: представление слов и саммаризация. Рассмотрены основные методы представления слов: токенизация на подслова, посимвольные и гибридные подходы, описаны их достоинства и недостатки. Исследованы архитектуры и принципы моделей, предназначенных для решения задач саммаризации: как экстрактивной, так и абстрактивной. Кроме того, рассмотрены основные метрики для оценки качества моделей саммаризации.

Из данной статьи можно сделать вывод, что различные архитектуры имеют свои достоинства и недостатки, и нельзя выделить какие-либо из них как эталон. Для решения данных задач используются сверточные сети, рекуррентные сети, трансформеры, генеративно-состязательные сети, обучение с подкреплением. Обработка и генерация естественного языка - довольно молодая область науки, имеющая обширную почву для исследований.

Список литературы

[1] Gage, Philip. "A new algorithm for data compression."C Users Journal 12.2 (1994): 23-38.

[2] Sennrich, Rico, Barry Haddow, and Alexandra Birch. "Neural machine translation of rare words with subword units."arXiv preprint arXiv:1508.07909 (2015).

[3] Provilkov, Ivan, Dmitrii Emelianenko, and Elena Voita. "Bpe-dropout: Simple and effective subword regularization."arXiv preprint arXiv:1910.13267 (2019).

[4] Schuster, Mike, and Kaisuke Nakajima. "Japanese and korean voice search."2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2012.

[5] Kudo, Taku. "Subword regularization: Improving neural network translation models with multiple subword candidates."arXiv preprint arXiv:1804.10959 (2018).

[6] Kudo, Taku, and John Richardson. "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing."arXiv preprint arXiv:1808.06226 (2018).

[7] Ling, Wang, et al. "Finding function in form: Compositional character models for open vocabulary word representation."arXiv preprint arXiv:1508.02096 (2015).

[8] Kim, Yoon, et al. "Character-aware neural language models."Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.

[9] Luong, Minh-Thang, and Christopher D. Manning. "Achieving open vocabulary neural machine translation with hybrid word-character models."arXiv preprint arXiv:1604.00788 (2016).

[10] Cao, Kris, and Marek Rei. "A joint model for word embedding and word morphology."arXiv preprint arXiv:1606.02601 (2016).

[11] See, Abigail, Peter J. Liu, and Christopher D. Manning. "Get to the point: Summarization with pointer-generator networks."arXiv preprint arXiv:1704.04368 (2017).

[12] Gehrmann, Sebastian, Yuntian Deng, and Alexander M. Rush. "Bottom-up abstractive summarization."arXiv preprint arXiv:1808.10792 (2018).

[13] Nallapati, Ramesh, Feifei Zhai, and Bowen Zhou. "Summarunner: A recurrent neural network based sequence model for extractive summarization of documents."Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 31. No. 1. 2017.

[14] Narayan, Shashi, Shay B. Cohen, and Mirella Lapata. "Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization."arXiv preprint arXiv:1808.08745 (2018).

[15] Liu, Yang, and Mirella Lapata. "Text summarization with pretrained encoders."arXiv preprint arXiv:1908.08345 (2019).

[16] Zhong, Ming, et al. "Extractive summarization as text matching."arXiv preprint arXiv:2004.08795 (2020).

[17] Paulus, Romain, Caiming Xiong, and Richard Socher. "A deep reinforced model for abstractive summarization."arXiv preprint arXiv:1705.04304 (2017).

[18] Zhang, Xingxing, and Mirella Lapata. "Sentence simplification with deep reinforcement learning."arXiv preprint arXiv:1703.10931 (2017).

[19] Rekabdar, Banafsheh, Christos Mousas, and Bidyut Gupta. "Generative adversarial network with policy gradient for text summarization."2019 IEEE 13th international conference on semantic computing (ICSC). IEEE, 2019.

[20] Dou, Zi-Yi, et al. "Gsum: A general framework for guided neural abstractive summarization."arXiv preprint arXiv:2010.08014 (2020).

[21] Papineni, Kishore, et al. "Bleu: a method for automatic evaluation of machine translation."Proceedings of the 40th annual meeting of the Association for Computational Linguistics. 2002.

[22] Lin, Chin-Yew. "Rouge: A package for automatic evaluation of summaries."Text summarization branches out. 2004.

[23] Banerjee, Satanjeev, and Alon Lavie. "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments."Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization. 2005.

[24] Snover, Matthew, et al. "A study of translation edit rate with targeted human annotation."Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers. 2006.

[25] Xu, Wei, et al. "Optimizing statistical machine translation for text simplification."Transactions of the Association for Computational Linguistics 4 (2016): 401-415.

[26] Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber. "Highway networks."arXiv preprint arXiv:1505.00387 (2015).

[27] Peters, Matthew E., et al. "Deep contextualized word representations."arXiv preprint arXiv:1802.05365 (2018).

[28] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation."Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.