Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
САЕ «Институт человека цифровой эпохи»
Автономная магистерская программа “Компьютерная лингвистика”

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ООП
д-р филол. наук, профессор
_____ З. И. Резанова
«17» 06 2019 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

ГЕНЕРАЦИЯ НОВОСТНЫХ ЗАГОЛОВКОВ ПРИ ПОМОЩИ DEEP SEQ2SEQ МОДЕЛИ

по основной образовательной программе подготовки магистра направление подготовки 45.04.03 - Фундаментальная и прикладная лингвистика

Здоровец Александр Игоревич

Руководитель ВКР
д-р. филол. наук, профессор
_____ З. И. Резанова
подпись
«17» 06 2019 г.

Консультант
науч. сотр. Лаборатории
когнитивных исследований языка
_____ Степаненко А.А.
«17» 06 2019 г.

Автор работы
студент группы № 13785
_____ А.И.Здоровец
подпись

Томск-2019

Оглавление.

Введение……... 4

Глава 1..…… 9

1.1. Искусственные нейронные сети: математическая модель биологической сети для решения задач…… 9

1.2. Ранние модели искусственных нейронных сетей: перцептрон Розенблатта…….. 10

1.3. Процесс обучения искусственных нейронных сетей…… 11

1.3.1. Forward- и Backpropagation……... 11

1.3.2. Градиентный спуск……... 12

1.3.3. Оптимизация…….. 14

1.3.4. Проблемы переобучения и недообучения…….. 19

Выводы по главе 1……... 22

Глава 2…….. 24

2.1. Искусственные нейронные сети в решении задач обработки естественного языка…… 24

2.2. Текстовые данные и проблема обработки последовательностей искусственными нейронными сетями……... 25

2.3. Рекуррентные нейронные сети как архитектура искусственных нейронных сетей для работы с последовательностями……... 27

2.4. Модификации блоков рекуррентных нейронных сетей: LSTM и GRU…….. 33

2.5. Перевод последовательности в последовательность при помощи архитектуры Seq2Seq…….. 37

2.6. Техники и приёмы для улучшения Seq2Seq моделей……... 41

2.7. Добавление “вертикальной” глубины в рекуррентные нейронные сети……... 48

2.8. Оценка качества работы рекуррентной нейронной сети, генерирующей тексты……. 50

Выводы по главе 2……... 51

Глава 3…….. 53

3.1. Практическое применение глубоких рекуррентных нейронных сетей для генерации новостных заголовков……. 53

3.2. Данные: корпус и словари……... 54

3.3. Архитектура нейронной сети, подбор гиперпараметров……. 55

3.4. Результаты работы…….. 61

Выводы по главе 3……... 65

Заключение…….. 67

Список использованных источников и литературы....…… 70

Приложение А……. 80

Введение

В данной работе решается задача автоматической генерации новостных заголовков на основе новостной статьи при помощи Seq2Seq модели (искусственной нейронной сети — ИНС).

Актуальность данной работы проистекает из информационной перегрузки вследствие информационного взрыва. Для успешной и эффективной работы с информацией человеку необходима помощь компьютера. Однако машина не может понимать текст, она понимает только нули и единицы. Научить её работать с текстом — как понимать, так и создавать — цель компьютерных лингвистов. Эта общая цель может быть решена последовательной разработкой всё более эффективных методов и разрешением частных задач, среди которых одной из самых актуальных является автоматическое понимание и генерация текстов, в том числе — автоматическая генерация заголовков и новостей. Разрешению первой из них посвящена данная работа.

Объектом является суммаризация и генерация текстов естественного языка.

Предметом — генерация текста глубокой рекуррентной нейронной сетью на основе автоматически выделенной языковой модели, обусловленной смысловым вектором другого текста.

Цель данной работы — создание базовой глубокой рекуррентной нейронной сети для генерации заголовков на русском языке из новостных статей.

В процессе достижения поставленной цели мы решали несколько **задач**:

1. Исследование возможных архитектур ИНС для достижения цели;
2. Анализ отличий русского языка от английского в контексте достижения цели;

3. Исследование и выбор возможных техник и приёмов для достижения цели;

4. Нахождение/создание корпуса для тренировки и проверки ИНС;

5. Создание «игрушечных» архитектур для настройки и проверки НС;

6. Подбор гиперпараметров ИНС;

7. Создание глубокой РНС, обучение и проверка.

Теоретической основой являются такие разделы компьютерной лингвистики и машинного обучения, как векторное представление слов (Миколов и др., “Distributed Representations of Words and Phrases and their Compositionality”), абстрактивная и экстрактивная суммаризация (Наллапати и др., “Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond”), статистический машинный перевод (Чо и др., “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation”), нейронный машинный перевод (Ву и др., “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation”), рекуррентные нейронные сети (Шерстинский, “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network”), оптимизация рекуррентных нейронных сетей (Бенжио и др., “Advances in Optimizing Recurrent Networks”), рекуррентные блоки LSTM (Хохрайтер и Шмидхубер, “Long Short-Term Memory”), рекуррентные блоки GRU и архитектура Encoder-Decoder (Чо и др., “On the Properties of Neural Machine Translation: Encoder–Decoder Approaches”), генерация заголовков нейросетью (Гаврилов, “Self-Attentive Model for Headline Generation”), глубокие рекуррентные нейросети (Суцкевер и др., “Sequence to Sequence Learning with Neural Networks”).

Нейросеть написана на языке программирования Python 3.6[1]. Основной библиотекой стала Keras[2] с Tensorflow на бэкэнде; также использовались библиотеки numpy, pandas, scikit-learn, seaborn и др. Нами была выбрана архитектура рекуррентной нейронной сети с LSTM-блоками. При анализе текстов использовались статистические методы, такие как частотный анализ, трансформация и построение описательных статистик при помощи библиотеки Scikit-Learn языка программирования Python.

В качестве материала использовался архив новостного ресурса Lenta.ru объёмом 739,346 текстов.

Новизна данной работы заключается в применении рекуррентных нейронных сетей для генерации заголовков к русскому языку, построении базового решения и в использовании глубоких архитектур рекуррентных нейронных сетей. Сложность любых языковых моделей возрастает с вариативностью языка, а русский язык намного более флективен, чем английский, на материале которого написано, пожалуй, подавляющее большинство работ. Кроме того, глубокая архитектура рекуррентных нейронных сетей применяется в литературе достаточно редко. В данной работе мы обращаем внимание на различные техники и приёмы, повышающие эффективность нейронных сетей, и предпринимаем попытку проанализировать, какие из них могут принести наибольшую пользу в контексте работы с русским языком.

Теоретическая значимость определяется тем, что полученные в работе данные позволяют проверить результативность использования разрабатываемых методов применительно к решению задач суммаризации и генерации текстов на русском языке, способствуя созданию базовой модели, подходящей для генерации новостных заголовков на русском языке.

---

[1] Kirk M. Thoughtful Machine Learning with Python: A Test-driven Approach. – " O'Reilly Media, Inc.", 2017.
[2] Шолле Ф. Глубокое обучение на Python //СПб.: Питер.– 2018.

Практическая значимость обуславливается тем, что в работе представлено применение рабочей модели рекуррентной нейронной сети для решения задачи автоматической генерации новостных заголовков из текста новости на русском языке, которая может служить основой для модификаций, повышающих её эффективность, базовым решением для сравнения новых моделей и может применяться для ряда смежных задач (например, для автоматического перевода, генерации лида и т.д.).

Структура работы обусловлена этими задачами и логикой исследования заявленной области. Исследование включает в себя введение, четыре главы и заключение.

Во введении ставится тема исследования, раскрывается научная ценность работы в контексте области, ставятся цели и задачи.

В первой главе рассматривается краткая история искусственных нейронных сетей, в частности одна из первых моделей — перцептрон Розенблатта. На его примере описываются общие алгоритмы их работы и обучения, а также различные методы оптимизации.

Во второй главе обсуждается текст как данные для ИНС и его специфика. В соответствии с этим фокус смещается на архитектуры, предназначенные для работы с подобного рода данными, в частности на рекуррентные НС и их более эффективные разновидности. Отдельное внимание уделяется модели Seq2Seq Encoder-Decoder, которая часто используется для понимания и генерации текстов машиной, и её улучшения.

В третьей главе описывается наша модель для генерации заголовков русскоязычных новостей. Представляются данные, используемые библиотеки и программные решения, а также результаты её работы.

В заключении подводятся краткие итоги нашего исследования, в частности о применимости некоторых из существующих подходов к суммаризации и генерации англоязычных текстов к русскому языку,

анализируются возможные проблемы и пути решения, задаётся траектория для дальнейших исследований.

В приложении А приводятся примеры сгенерированных заголовков.

Глава 1.

1.1. Искусственные нейронные сети: математическая модель биологической сети для решения задач.

В 1943 году нейрофизиолог Уоррен МакКаллок и математик Уолтер Питтс предположили, как работает нейронная сеть в головном мозге, и при помощи электрической цепи создали рабочий прототип. В 1949 году нейропсихолог Дональд Хебб укрепил понимание работы биологических нейронов: нейронные связи укрепляются при использовании. Примерно в то же время начали появляться первые прототипы искусственных нейронных сетей, однако они не получили развития: технологические ограничения времени не смогли удовлетворить надежды на появление искусственного интеллекта, вызвав разочарование, а научная фантастика посеяла страх перед мыслящими машинами.

Интерес к компьютерным нейросетям воскрес в 80-х годах прошлого века, однако только с развитием и распространением технологий они стали пригодны для массового применения и серьёзных исследований.

Биологические нейроны “выстреливают” электрическими импульсами, и при достаточной интенсивности сигнала другие нейроны могут активироваться, посылая сигнал дальше, или заканчивая цепочку активаций. Искусственные нейронные сети (ИНС) условно моделируют этот процесс при помощи математики: нейроны обрабатывают и передают входные значения (“сигналы”), связи между нейронами увеличивают (“усиливают”) или уменьшают (“ослабляют”) их, функция активации определяет, что делать с полученным значением дальше.

Необходимо понимать, что хотя ИНС были вдохновлены мозгом биологическим, нет доказательств, что человеческий разум производит те же операции. Технически машинное обучение при помощи последовательности математических операций находит некоторое представление исходных

данных, пригодное для решения некоторых задач. Алгоритмически это выражается в виде матричных операций, однако также ИНС можно представить в виде графа потока сигналов: сеть соединённых узлов, где связь между узлами j и k определяет функцию переход сигнала x_j в сигнал y_k; связи могут быть синаптическими (линейная функция умножения входного сигнала на соответствующий вес $x_j \times w_{jk} = y_k$) и активационными (нелинейная функция, например сигмоидная).

1.2. Ранние модели искусственных нейронных сетей: перцептрон Розенблатта.

Перцептрон Розенблатта — это первая алгоритмически описанная ИНС (или по крайней мере её прародитель). Он основан на вышеупомянутой модели Маккалока-Питтса. Его значимость заключается не только в том, что он вдохновил многих исследователей, но также в том, что принципы его работы аналогичны принципам работы более глубоких и крупных нейросетей прямого распространения (и других видов, однако в этой главе мы преимущественно рассматриваем сети прямого распространения). Данные подаются на входной слой, который передаёт их последующим слоям; главное отличие более сложных ИНС от перцептрона заключается в том, что в них добавляются скрытые слои (его наличие в перцептроне признаётся не всеми исследователями), которые и конструируют сложные функциональные отношения; наконец, данные, пройдя «фильтрацию» в функциях нейросети активируются, и выходной слой выдаёт ответ нейронной сети[3,4].

Перцептрон используется в качестве бинарного классификатора для двух линейно разделимых классов. Он состоит из входного слоя, функции комбинирования, функции активации и выходного слоя. Взвешенная сумма

[3] Haykin S.S. et al. Neural networks and learning machines. – Upper Saddle River : Pearson education, 2009. – Т. 3.
[4] Kirk M. Thoughtful Machine Learning with Python: A Test-driven Approach. – " O'Reilly Media, Inc.", 2017.

входных сигналов (значений признаков) и компонента смещения (bias) передаётся функции активации, выходной слой выдаёт одно из двух значений (-1 или +1, также возможны 0 или 1 в зависимости от используемой функции активации). Таким образом, перцептрон представляет собой формулу

$$y = \varphi(\sum_{i=1}^{m} w_i x_i + b), \quad (1)$$

где y — предсказание перцептрона, φ — функция активации, **x** — множество входных значений $\{x_1, x_2, ..., x_m\}$, **w** — множество весов $\{w_1, w_2, ..., w_m\}$, b — компонент смещения.

Искусственная нейронная сеть — это универсальный аппроксиматор[5]. Это значит, что с её помощью можно создать приближение $g(\cdot)$ для любой функции $f(\cdot)$ (предполагается, что наблюдаемые данные подчиняются некоторому закону, т.е. некоторой функции). Такое аппроксимирование происходит в результате обучения модели, которое состоит из двух этапов: прямого распространения сигналов (forward propagation) и обратного распространения ошибки (backpropagation); это верно для обучения нейронных сетей с учителем, которые и рассматриваются в данной работе.

1.3. Процесс обучения искусственных нейронных сетей.

1.3.1. Forward- и Backpropagation.

“Обучение модели” — несколько метафорическое описание процесса автоматического подбора параметров модели. Параметры можно рассматривать как степени свободы модели: чем их больше, тем более сложные решения она может принимать. Любая модель параметризуется двумя типами переменных: внутренними, которые модель настраивает сама, и внешними “гиперпараметрами”, которые настраиваются разработчиком.

[5] Nielsen M. A. Neural networks and deep learning. – San Francisco, CA, USA: : Determination press, 2015. – T. 25.

Например, в перцептроне гиперпараметром можно считать функцию активации, остальные же переменные являются внутренними параметрами модели. Учитывая, что входные значения признаков даны изначально, а выходные гипотезы являются результатом вычислений, нетрудно увидеть, что перцептрон может изменять веса **w** и компонент смещения b. Он делает это итеративно, чередуя направление распространения сигнала/ошибки.

Forward propagation — простой процесс. На вход модели подаются сигналы, которые она передаёт дальнейшим слоям, применяя вес связи между нейронами к двигающимся сигналам. Характер этой передачи зависит от разновидности нейронной сети и её характеристик. Например, в перцептроне все входные узлы (нейроны) и узел (нейрон) смещения соединены с единственным узлом единственного скрытого слоя, сигнал из которого передаётся функции активации для получения гипотезы в выходном слое.

В начале цикла обучения внутренние параметры инициализированы случайным образом (или приравниваются к 0, или выбираются по какому-либо алгоритму и т.д.). Непосредственно обучение происходит на этапе backpropagation. Backpropagation можно назвать “обучением на ошибках”: гипотеза модели $\hat{y}$ сравнивается с реальным значением y для данного набора признаков **x**, после чего определяется степень ошибки, в соответствии с которой нейросеть передаёт информацию о том, как нужно изменить параметры для её минимизации, в обратном прямому распространению сигналов порядке (от выходного слоя ко входному). Минимизация потерь (ошибок) осуществляется при помощи градиентного спуска.

1.3.2 Градиентный спуск.

Функция потерь J (также встречается C или L) или в общем смысле представляет собой несовпадение между гипотезой ИНС $\hat{y}$ и наблюдаемым значением y. Обучение нейронной сети — это задача оптимизации, где цель backpropagation — минимизировать функцию потерь, или, другими словами, найти точку минимума функции. Для этого необходимо найти такую точку, в которой производная функции равняется 0 (стоит отметить, что подобное условие не означает, что точка является глобальным минимумом; это может быть локальным минимумом, точкой максимума или точкой, подозрительной на экстремум). Нейронная сеть — это сложное геометрическое преобразование в многомерном пространстве, реализованное в виде последовательности простых шагов. Другими словами, ИНС зависит от многих переменных, вследствие чего необходимо взять производную функции по всем изменяющимся переменным (внутренним параметрам модели). Для оптимизации этого процесса в backpropagation используется градиентный спуск.

Градиент — это вектор, указывающий направление наибыстрейшего возрастания функции; соответственно, для нахождения наибыстрейшего убывания функции можно использовать градиент с обратным знаком. Этот вектор состоит из частных производных функции по всем переменным, что позволяет применить его для изменения этих переменных. Математически процесс пакетного градиентного спуска (Batch Gradient Descent) выглядит как

$$\theta = \theta + \Delta\theta = \theta - \eta \nabla_{\theta} J(\theta), \qquad (2)$$

где θ — параметры сети (веса), $J(\theta)$ — целевая функция (функция потерь), η — скорость обучения.

После изменения весов модель снова производит forward propagation, высчитывая результат, определяет значение функции потерь и настраивает переменные. Этот цикл повторяется заданное количество раз (эпох) или до

достижения некоторого порога ошибок (ранняя остановка). С каждым шагом (если в сети нет ошибок) значение функции потерь должно уменьшаться.

1.3.3. Оптимизация.

Внутренние параметры сеть обновляет сама, однако гиперпараметры необходимо задавать вручную. В некотором роде гиперпараметрами можно считать и архитектуру сети, и количество слоёв, и используемую функцию активации, однако в данном разделе мы считаем гиперпараметрами переменные, которые задаются разработчиком и влияют на внутренние параметры нейронной сети. Однако это не значит, что они не могут менятся в процессе работы модели. Тем не менее, на данный момент подбор хороших гиперпараметров — это, скорее, искусство, нежели наука, где первостепенную роль играет опыт и научная интуиция разработчика.

Выше мы описали пакетный, или полный, градиентный спуск (ГС). Он использует все доступные наблюдения для одного изменения параметров, что неэффективно как с точки зрения затрачиваемого времени, так и с точки зрения используемой памяти. На практике более распространены минипакетный ГС (Minibatch GD) и стохастический ГС (Stochastic GD). Стохастический ГС пересчитывает градиент и обновляет параметры на основе одного наблюдения из выборки, что ускоряет работу алгоритма, но вызывает флуктуации в функции ошибок. Минипакетный ГС — что-то среднее между этими двумя вариациями: он обновляет параметры после каждого пакета (или, скорее, минипакета, хотя часто эти термины используются взаимозаменяемо) данных, обычно 32-64 наблюдения. Обычно работа с одним минипакетом называется итерацией, работа со всеми минипакетами называется эпохой, т.е. цикл обучения сети состоит из эпох, которые в свою очередь состоят из итераций.

Часто градиент умножается на некоторый скаляр α или η (обычно в диапазоне $[0;1]$), который называется “скорость обучения”. Вследствие этого “шаг” градиентного спуска уменьшается; это увеличивает время сходимости модели, как и шанс нахождения экстремума, т.к. не даёт алгоритму обратного распространения ошибки “перепрыгнуть” подобную точку вследствие слишком большого изменения параметров. Слишком маленькая скорость обучения значительно увеличивает время обучения модели и повышает риск “застрять” в локальном минимуме, слишком большая может привести к тому, что функция не сойдётся совсем.

Производные — это мощный инструмент, однако не лишённый недостатков. Например, график целевой функции может иметь сложный ландшафт, где плато чередуется с областями резкого возрастания/убывания; очевидно, что скорость изменения весов на плато, где производная стремится к 0, будет крайне маленькой, а в области резкого изменения она будет чересчур большой. Можно было бы попробовать брать производные более высокого порядка или гессиан-функции, но большое количество внутренних параметров модели, обеспечивающих вычислительную мощь ИНС, в таком аспекте становится огромным недостатком. Для борьбы с этими и другими проблемами используются различные методы оптимизации. Рассмотрим некоторые из них.

Для решения проблемы “застревания” и для ускорения сходимости функции часто применяется инерция. В физике инерция — это свойство тела оставаться в состоянии покоя или движения при отсутствии воздействия внешних сил. Другими словами, физическое тело какое-то время продолжает движение после придания ему начального импульса. Подобный механизм можно применить к функции: если значение функции какое-то время изменяется в одном направлении, возможно, ей следует продолжать

изменяться в этом же направлении. На каждом шаге функции высчитывается инерция:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta), \quad (3)$$

$$\theta_{t+1} = \theta_t - v_t, \quad (4)$$

где v_t — инерция на данном шаге, v_{t-1} — инерция на предыдущем шаге, γ — коэффициент сохранения ($0 < \gamma < 1$).

При длительном “движении” функции в одном направлении инерция накапливается, ускоряя схождение, при резком же изменении направления инерция имеет противоположный знак и смягчает осцилляции. Это полезно, однако более эффективным решением было бы предугадывание дальнейшего изменения функции и ускорения/замедления на основе этого. Такой алгоритм существует: ускоренный градиент Нестерова (Nesterov Accelerated Gradient). Для изменения параметров будет использоваться инерция γv_{t-1} ; если не учитывать градиент, изменение параметров будет иметь вид

$$\theta_{t+1} = \theta_t - \gamma v_{t-1}, \quad (5)$$

что даёт грубую оценку следующих их значений. Подсчитав градиент от них, мы получим столь же грубое приближение будущих значений градиента:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta_t - \gamma v_{t-1}). \quad (6)$$

Существуют и иные методы оптимизации на основе инерции, отличающиеся в, по сути, мелочах. Интереснее рассмотреть алгоритмы с совершенно иной идеей, например, адаптивный градиент, или AdaGrad (Adaptive Gradient). У этого алгоритма есть много различных имплементаций, поэтому иногда его рассматривают не как отдельный алгоритм, а как семейство алгоритмов, что будет видно далее. В его основе лежит идея о том, что различные признаки обладают разной чувствительностью к изменению

градиента, например, в силу частоты встречаемости, поэтому к ним нужно применять различную скорость обучения. Для этого для каждого параметра вычисляется сумма квадратов его обновлений:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} \odot g_t \,, \tag{7}$$

где θ — параметр для обновления, η — начальная скорость обучения, ε — небольшое значение, чтобы избежать деления на 0, g_t — оценка градиента на шаге t, G_t — диагональная матрица сумм тензорных произведений градиентов до шага t:

$$G_t = G_t + g_t^2 \ . \tag{8}$$

Легко видеть, что частые изменения параметра ведут к быстрому увеличению этой суммы; соответственно, быстро будет расти и знаменатель, сильно уменьшая его скорость обучения.

Проблема данного алгоритма заключается в том, что знаменатель будет расти всегда (если, конечно, градиент не равен 0), поэтому уменьшение скорости обучения будет монотонным. Для борьбы с этим была создана модификация Adadelta, которая использует экспоненциально затухающее среднее $E[g^2]$ (decaying average) последних w градиентов, чтобы не хранить значения сумм квадратов. Тогда:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \ , \tag{9}$$

где γ — коэффициент затухания, аналогичный коэффициенту сохранения в методах с инерцией. С учётом этого выражение (7) становится:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} \odot g_t \,. \tag{10}$$

Однако это ещё не Adadelta, а RMSProp — Root Mean Squared Propagation:

$$\text{RMS}\,[g]_t = \sqrt{E[g^2]_t + \varepsilon}\,. \qquad (11)$$

Интересно, что этот метод никогда не был опубликован; его предложил профессор Джоффри Хинтон (Geoffrey Hinton) в рамках своих лекций. Он широко используется в машинном обучении, хотя во многом является лишь шагом Adadelta, который добавляет в числитель функции (НОМЕР) RMS относительно изменения параметров на прошлом шаге для совпадения размерности матрицы параметров θ и градиента $\Delta\theta$:

$$\theta_{t+1} = \theta_t - \frac{RMS[\theta]_{t-1}}{RMS[g]_t} \odot g_t\,. \qquad (12)$$

Adadelta сочетает в себе монотонное убывание функции потерь как в SGD, аккумулирование части прошлых градиентов в числителе как в инерции, индивидуальное обновление параметров в разных измерениях в знаменателе как в AdaGrad. Но это не единственный подобный метод оптимизации: большую известность также имеет Adam.

Adam — Adaptive Moment Estimation, адаптивное приближение инерции — хранит не только экспоненциально затухающее среднее квадратов градиентов v_t, как Adadelta и RSMProp, но также и затухающее среднее градиентов m_t, что похоже на инерцию:

$$m_t = \beta_2 m_{t-1} + (1-\beta_2) g_t\,, \qquad (13)$$

$$v_t = \beta_1 v_{t-1} + (1-\beta_1) g_t^2\,, \qquad (14)$$

где m_t — первый центральный момент (среднее) градиента, v_t — второй центральный момент (дисперсия) градиента, β_1 и β_2 — коэффициенты затухания в диапазоне [0;1]. При нулевом начальном значении векторов m_t и v_t и большом значении коэффициентов β_1 и β_2 их изменения накапливаются очень долго. Для этого на практике используется скорректированные значения $\widehat{m}_t$ и $\widehat{v}_t$:

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad (15)$$

$$\widehat{v}_t = \frac{v_t}{1-\beta_2^t}. \quad (16)$$

С учётом этих параметров, шаг обновления Adam выглядит следующим образом:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v}_t}+\varepsilon} \odot \widehat{m}_t. \quad (17)$$

1.3.4. Проблемы переобучения и недообучения.

Как было упомянуто выше, ИНС — универсальный аппроксиматор, способный смоделировать любую функцию. Подавая на вход сети данные, мы предполагаем, что за ними стоят некоторые законы, которые нейросеть сможет обнаружить.

С достаточно мощной моделью эта цель более чем достижима. Но встаёт вопрос о её применимости в реальном мире: нейросеть может выдавать идеальные результаты на представленных ей данных, но совсем не работать на других. Если модель не может генерализовать выученные законы на новые датасеты, это называется “переобучение” (Overfitting, оверфиттинг); если модель не может обнаружить их на известных данных, это называется “недообучение” (Underfitting, андерфиттинг).

Недообучение является менее серьёзной проблемой, чем переобучение, потому что по сути это не более чем индикатор “слабости” модели, её неспособности к выделению определённых паттернов, или недостаточности данных. Она решается расширением выборки или перестройкой архитектуры. Переобучение же свидетельствует о неспособности модели обобщать; в некотором роде можно сказать, что модель запомнила правильные ответы на подаваемые ей данные, а не научилась их находить. Также переобучение может быть вызвано контаминированностью имеющихся данных (например, большое количество

“шума” или необычные признаки, не встречающиеся за пределами этих данных). Размер модели (количество параметров в ней) также играет роль в появлении данных проблем: если рассматривать параметры как степени свободы, то слишком большое их число позволит модели рассматривать каждый случай как уникальный (грубо говоря, отдельная цепочка решений для каждого), а слишком маленькое не позволит принимать комплексные решения (например, она выделит меньше классов, чем есть в данных).

Наипервейший, самый базовый шаг в борьбе с оверфиттингом — разделение датасета на 3 подсета. Первый, тренировочный набор, используется во время тренировки модели, второй, валидационный (который иногда не используется), используется для настройки гиперпараметров, третий, тестовый, используется для проверки обученной модели на новых данных.

Существует множество приёмов и путей решения проблемы переобучения[6]. Для классических методов машинного обучения и неглубоких нейросетей, которые быстро тренируются, можно запустить модель несколько раз: случайная инициализация параметров может сильно повлиять на отправную точку процесса обучения, сильно влияя на него, и, имея выбор из моделей с разными начальными точками, можно выбрать лучшую. Часто используется ранняя остановка: при переобучении потеря на тренировочном наборе данных начинает убывать, а потеря на валидационном — возрастать (или скорость уменьшения обеих сильно различается), что служит сигналом для остановки обучения. Хороших результатов можно добиться применением регуляризации: она добавляет в функцию потерь терм, который накладывает на нейросеть “штраф” за слишком большие значения параметров, заставляя её уменьшать их как можно сильнее.

[6] Improve Shallow Neural Network Generalization and Avoid Overfitting // Mathworks. — URL: https://www.mathworks.com/help/deeplearning/ug/improve-neural-network-generalization-and-avoid-overfitting.html;jsessionid=825d3112487ac09f537fb633dbe5 (дата обращения 05.06.2019). — 2019.

Многие из приёмов борьбы с переобучением применимы и к глубоким нейросетям, которые характеризуются огромным количеством нейронов и слоёв (и, соответственно, параметров). Но в 2014 году был предложен многообещающий механизм, который набирает всё большую популярность: дропаут (Dropout, также “исключение” и “прореживание”)[7]. Дропаут — метод усреднения больших нейросетей. При применении дропаута каждый нейрон в сети может быть “исключён” с заданной вероятностью p; это означает полное обнуление всех его входящих и исходящих связей (т.е. он возвращает 0 вне зависимости от передаваемых ему значений). Исключённые нейроны не вносят вклад в обучение сети, поэтому во время backpropagation игнорируются. По сути это равносильно созданию другой архитектуры, которая имеет общие с изначальной веса. Например, если поставить рекомендованную авторами метода вероятность дропаута $p = 0.5$ (выключается примерно каждый второй нейрон), то фактически каждая итерация — это рандомное сэмплирование сети из 2^H вариантов архитектур, где H — количество нейронов в сети[8]. Эффективность дропаута заключается, во-первых, в том, что он напоминает ансамблевые методы, когда комбинируются результаты работы нескольких небольших сетей, часто показывают себя лучше, чем одна большая сеть; во-вторых, он борется с коадаптацией нейронов, уменьшая их специализацию и увеличивая их генерализационную силу. Коадаптация происходит потому, что при обучении сети с помощью производных по всем переменным для уменьшения функции потерь изменяются параметры нейронов, и некоторые из них могут подстроиться таким образом, чтобы “компенсировать” ошибки других. При исключении случайных нейронов оставшиеся должны

[7] Srivastava N. et al. Dropout: a simple way to prevent neural networks from overfitting //The Journal of Machine Learning Research. – 2014. – Т. 15. – №. 1. – С. 1929-1958.
[8] Stanford Seminar - Can the brain do back-propagation? [Электронный ресурс] // Stanford University. URL: https://www.youtube.com/watch?v=VIRCybGgHts (дата обращения 03.06.2019). — 2016.

исправлять свои ошибки сами, так как наличие “страхующих” нейронов не гарантировано, что повышает их универсальность и способность к обобщению.

Дропаут прореживает нейросеть, делая её “тоньше”. Соответственно, сигналы в ней также становятся несколько меньше, поэтому, чтобы избежать проблем во время теста, необходимо масштабировать активации с учётом вероятности дропаута p (т.е. умножить на неё). Также можно использовать обратный дропаут (Inverse Dropout), который масштабирует активации во время тренировки с учётом обратной вероятности дропаута $\frac{1}{p}$.

Выводы по главе 1.

Искусственные нейронные сети как грубая модель мыслительного процесса человека появились в середине прошлого века, однако только в последние несколько десятилетий технические мощности позволили эффективно применять их в самых разных отраслях, что подробнее будет продемонстрировано в дальнейшем. Первые прототипы были очень простыми, но даже они показывали хороший результат в задачах линейной классификации. Сейчас архитектуры усложнились, но в их основе лежат приблизительно те же самые принципы и алгоритмы, что и более полувека назад.

Машинное обучение — широкая область, и сейчас, когда говорят о нейронных сетях, чаще всего имеют в виду глубокие нейронные сети (Deep Neural Networks), противопоставляя их традиционным методам машинного обучения (например, регрессии, деревьям решений и пр.). Они глубокие, потому что включают в себя несколько скрытых слоёв с огромным количеством параметров, или, другими словами, степеней свободы.

Самый важный компонент нейросетей — процесс обучения, в частности почти магический метод обратного распространения ошибок и

градиентного спуска. Силой градиента и оптимизационных приёмов он позволяет ИНС подбирать огромное количество (зачастую порядка десятков или сотен миллионов) параметров, оптимальных для решения конкретной задачи, без прямого участия человека. Необходимо отметить, что ГС не единственный метод обучения, однако он, пожалуй, наиболее распространённый, и на нём основываются многие другие.

Впрочем, разработчик моделей играет огромную роль в обучении. Прежде всего, он должен предоставить данные и выбрать гиперпараметры, такие как архитектура, скорость обучения, оптимизатор и др. И в этих аспектах, в отличие от внутренних параметров моделей, нет “волшебных” формул для подбора наиболее подходящих; часто они выбираются на основе существующего эмпирического опыта, многих итераций с различными значениями и прочих “правил большого пальца”.

Глава 2.

2.1. Искусственные нейронные сети в решении задач обработки естественного языка.

Современные нейросети обладают огромным количеством внутренних параметров, которые требуют серьёзных вычислительных мощностей. Но даже с ними результаты часто не идеальны. Как было сказано ранее, ИНС — универсальный аппроксиматор, и в теории, имея неограниченные мощности, можно было бы создать приближение любых реальных явлений. На данный момент это не представляется возможным; но зная свои данные и примерно представляя, что необходимо смоделировать, можно использовать наиболее адекватную архитектуру, как бы направляя работу сети в нужное русло и снимая с неё бремя обработки нерелевантных данных.

Достаточно очевидно, что, будучи математическими моделями, ИНС были созданы для работы с цифрами. И действительно, в качестве основной структуры данных в современных системах машинного обучения используются обобщение матриц с произвольным количеством измерений — тензоры. Однако сейчас с помощью нейросетей выполняется автоматический перевод с одного языка на другой[9] (текст), улучшается фотография с камеры телефона[10] (изображение), создаются новые видеозаписи[11] (видео)... Другими словами, нейросети способны работать с практически любыми типами данных, если найти способ представить их в численном виде.

---

[9] Wu Y. et al. Google's neural machine translation system: Bridging the gap between human and machine translation //arXiv preprint arXiv:1609.08144. – 2016.

[10] Вычислительная фотография : Будущее фотографии — это код [Электронный ресурс] // Vas3k URL: https://vas3k.ru/blog/computational_photography/ (дата обращения 04.06.2019). — 2019

[11] Deepfake videos: Inside the Pentagon's race against disinformation // CNN. URL: https://edition.cnn.com/interactive/2019/01/business/pentagons-race-against-deepfakes/ (дата обращения 10.06.2019). — 2019.

2.2. Текстовые данные и проблема обработки последовательностей искусственными нейронными сетями.

Существует немало способов численного представления текстуальных единиц, в частности слов. Например, можно подсчитать слова и представлять тексты как набор слов. Этот метод так и называется: “мешок слов” (Bag of Words). Чуть интереснее было бы подсчитать слова не только относительно текста (Term Frequency, частотность слова), но также и относительно всей коллекции документов (Document Frequency, частотность слова по документам); в таком случае это называется TF-IDF (I означает “Inverse”, обратная).

Эти подходы довольно очевидны, но находят относительно широкое применение применение, например, в Наивном Байесовском классификаторе. Более интересный подход — перевод слов в векторное пространство, или word embeddings. Самое просто решение — унитарное кодирование (One-Hot Encoding), когда к слову приводится нулевой вектор с одной единицей, положение которой соответствует положению слова в словаре. Легко видеть, что такие векторы будут крайне огромны (их размерность равна мощности словаря) и малоинформативны (1 бит на весь вектор, который лишь указывает на слово в словаре). Обычно, когда говорят про word embeddings, имеют в виду плотные векторы, полученные в результате работы некоторого алгоритма, такого как Word2Vec[12], GloVe[13] и т.д. Они небольшие (обычно около 300 измерений) и способны самостоятельно обнаруживать некоторые дистрибутивные особенности слов (как в уже хрестоматийном примере).

Наивный Байесовский классификатор основан на теореме Байеса, а наивность его заключается в предположении о независимости элементов друг от друга. Однако текст — это не только набор слов, но это ещё и их

[12] Mikolov T. et al. Distributed representations of words and phrases and their compositionality //Advances in neural information processing systems. – 2013. – С. 3111-3119.
[13] Pennington J., Socher R., Manning C. Glove: Global vectors for word representation //Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). – 2014. – С. 1532-1543.

взаимоотношение и взаимодействие в локальном и глобальном контекстах. Иногда они не играют роли при решении некоторой задачи, либо же играют столь незначительную роль, что эффективность их учёта относительно сложности алгоритма и вычислений стремится к нулю. Впрочем, это не значит, что их можно всегда игнорировать: “Мать любить дочь” и “Дочь любит мать” имеют относительно противоположный смысл. Таким образом, текст часто имеет смысл рассматривать как последовательность.

Последовательность – это набор элементов, упорядоченных по времени. Относительно текста можно считать временным измерением порядок, например, слов или символов: человек обрабатывает (читает, генерирует) слева направо (в случае русского, английского и др.) по слову/символу в единицу времени. Это грубая аналогия обработки текстовых данных человеком, но именно так работают рекуррентные НС.

Также необходимо отметить техническую проблему варьируемой длины входных данных. Разные тексты (как последовательность элементов) имеют разную длину, однако у нейросетей входной слой фиксирован (т.к. фиксирована размерность матрицы весов). Рекуррентные нейронные сети обходят эту проблему, фокусируясь лишь на одном элементе, что позволяет им иметь фиксированный вход на каждом шаге и произвольное количество шагов.

Впрочем, хорошую эффективность также показывают и свёрточные нейросети (Convolutional NN). Они нацелены на распознавание образов и как бы сворачивают области (контексты) данных в некоторое представление[14]. В данной работе мы рассматриваем только РНС, т.к. несмотря на успехи СНС первые представляются нам более интересными с точки зрения

[14] Convolutional Neural Networks (CNNs / ConvNets) : CS231n Convolutional Neural Networks for Visual Recognition [Электронный ресурс] // Stanford University. — URL: http://cs231n.github.io/convolutional-networks (дата обращения 30.04.2019). — 2019.

моделирования процесса понимания и генерации языка (что, однако, не означает, что они априори лучше[15]).

Стоит также отметить техническую деталь: обычно временные ряды в машинном обучении представляются в виде трёхмерных (трёхранговых) тензоров, где первая ось (измерение) называется осью пакетов (хранит наблюдения), вторая — осью времени (хранит временные шаги соответствующих наблюдений), третья — осью признаков (хранит признаки соответствующих временных шагов).

2.3. Рекуррентные нейронные сети как архитектура искусственных нейронных сетей для работы с последовательностями.

Искусственные нейронные сети отлично справляются с нахождением паттернов в неструктурированных данных. Однако последовательности, к которым относятся и тексты, сложны структурно, что видно даже в тензорной репрезентации данных (двуранговые и трёхранговые тензоры). В таких данных элементы одного наблюдения зависят друг от друга, прошлые (и даже будущие) временные шаги определяют содержание текущего; поэтому нейронная сеть должна уметь моделировать подобную зависимость.

Модели, решающие задачу выработки некоторого правила для перевода входных данных **x** в выходные данные **y**, можно разделить на два типа: дискриминативные и генеративные (порождающие)[16]. Первые моделируют условное распределение $p(y|x)$, т.е. вероятность результата от входа, а вторые — совместное распределение $p(x,y)$, т.е. распределение как на входе, так и на выходе. Генеративные модели более общие, т.к. по теореме Байеса можно легко найти условную вероятность при известной совместной.

[15] The Bitter Lesson [Электронный ресурс] // Rich Sutton. URL: http://www.incompleteideas.net/IncIdeas/BitterLesson.html (дата обращения 05.06.2019). — 2019.
[16] Machine learning: Generative and discriminative models [Электронный ресурс] // Srihari S. Machine Learning Course. URL: http://www.cedar.buffalo.edu/~srihari/CSE574/Discriminative-Generative.pdf (дата обращения 05.06.2019). – 2010.

Дискриминативные модели хорошо показывают себя в задачах классификации, но, как очевидно из названия, порождающие модели больше подходят для создания новых данных. Цель генеративных моделей — максимизировать вероятность совпадения моделируемого и наблюдаемого распределений, а нейросеть используется как функциональный аппроксиматор[17].

В контексте работы с текстами мы хотим научить РНС предсказывать следующий токен (в данной работе — слово) на основании уже имеющейся последовательности. Другими словами, перед нами стоит задача создания языковой модели, которая могла бы определить вероятность существования текста длины T как совместную вероятность слов $w_1, w_2, ..., w_T$ [18]:

$$p(w_1, w_2, ..., w_T) = \prod_{t=1}^{T} p(w_t | w_1, ..., w_{t-1}) . \quad (18)$$

Идеальная модель могла бы генерировать новые тексты из этой вероятности, извлекая по одному слову на шаге t:

$$w_t \sim p(w_t | w_{t-1}, ..., w_1). \quad (19)$$

Однако легко видеть, что такая языковая модель могла бы существовать, если бы они имела доступ ко всем возможным текстам языка. Модель, построенная лишь на относительно небольшой выборке, изначально ограничена. Кроме того, выбирая лишь наиболее вероятное слово на каждом шаге, модель, очевидно, будет генерировать один и тот же текст. Тем не менее, даже ограниченная языковая модель может быть крайне мощной и полезной. Рекуррентные нейронные сети отлично подходят для их построения, так как моделируют зависимости во времени.

[17] Doersch C. Tutorial on variational autoencoders //arXiv preprint arXiv:1606.05908. – 2016.
[18] Dive into Deep Learning. [Электронный ресурс] // Zhang A. et al. — URL: http://www.d2l.ai (дата обращения 04.06.2019). — 2019.

Структура РНС похожа на структуру многослойного перцептрона, но скрытые нейроны (hidden units) временной задержки могут быть связаны[19]. Входные данные подаются на вход нейросети на первом шаге, обрабатываются ею, и она выдаёт выходные данные; тут в дело вступает рекуррентность: эти выходные данные подаются на вход на втором шаге, и так далее, пока не будет достигнут конец временной оси. В процессе этого она итеративно строит релевантное представление прошлой информации h_t (скрытое состояние) как функцию с обучаемыми параметрами θ, регулирующими запоминание/забывание, прошлого скрытого состояния h_{t-1} и текущих входных данных x_t[20]:

$$h_t = f(h_{t-1},\ x_t, \theta). \qquad (20)$$

Как было отмечено ранее, глубокие нейросети отличаются большим количеством слоёв и параметров. Из-за этого существуют споры о том, стоит ли считать РНС действительно глубокими: блок нейросети (cell) часто однослойный; глубина же появляется, когда мы “разворачиваем” её во времени и рассматриваем блоки на каждом шаге как слои (потенциально бесконечное число). Отсюда иногда проистекает путаница в наименовании: слоями РНС называются непосредственно скрытые слои, но также и развёрнутые во времени блоки. Это можно видеть на рисунке 1 из статьи “Training and analyzing deep recurrent neural networks”[21], где по горизонтали отложено время (Time), по вертикали — слои (n-layer RNN):

[19] Pascanu R., Mikolov T., Bengio Y. On the difficulty of training recurrent neural networks //International conference on machine learning. – 2013. – С. 1310-1318.
[20] Bengio Y., Boulanger-Lewandowski N., Pascanu R. Advances in optimizing recurrent networks //2013 IEEE International Conference on Acoustics, Speech and Signal Processing. – IEEE, 2013. – С. 8624-8628.
[21] Hermans M., Schrauwen B. Training and analysing deep recurrent neural networks //Advances in neural information processing systems. – 2013. – С. 190-198.

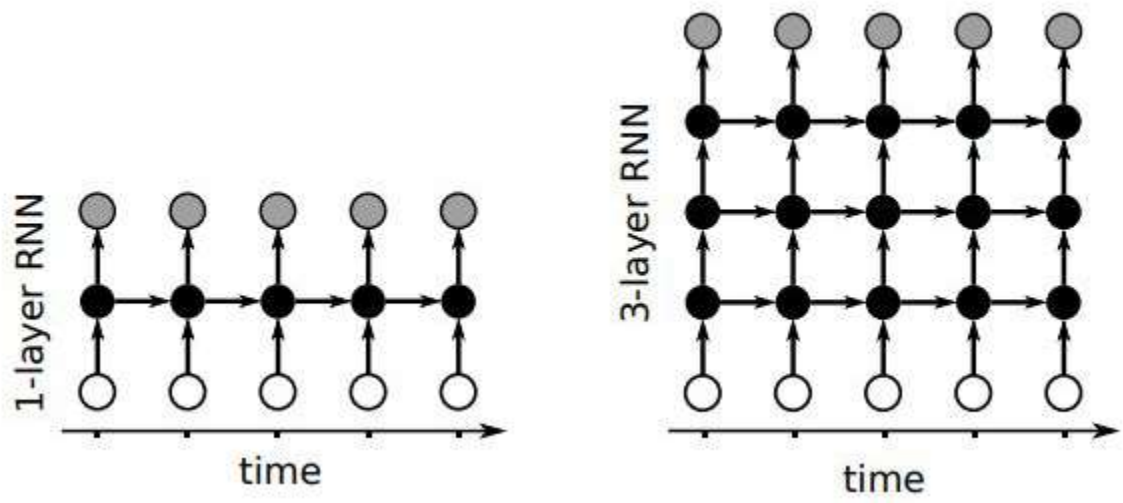

Рисунок 1 — Схема развёрнутых во времени неглубокой и глубокой РНС.

Мы вслед за К. Лопырёвым в статье “Generating Sequences With Recurrent Neural Networks”[22] считаем, что рекуррентные нейросети глубоки как во времени, так и в пространстве, т.к. каждый бит информации, проходя через вычислительный граф горизонтально или вертикально, последовательно взаимодействует с несколькими матрицами весов и нелинейными функциями; впрочем, в данной работе мы хотим добавить пространственной глубины с помощью дополнительных “вертикальных” слоёв.

Если обозначить входную последовательность как $\boldsymbol{x} = (x_1, ..., x_T)$, обычная РНС высчитывает скрытый вектор $\boldsymbol{h} = (h_1, ..., h_T)$ и выходную последовательность $\boldsymbol{y} = (y_1, ..., y_T)$, итерируя с $t = 1$ до T:

$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + b_h), \quad (21)$$

$$y_t = g(W_{hy}h_t + b_y), \quad (22)$$

где W — матрицы весов (индекс показывает связываемые слои) из параметров θ, b — вектор смещения (индекс показывает смещаемый слой), f — функция скрытого слоя, g — нелинейная функция активации. В

[22] Lopyrev K. Generating news headlines with recurrent neural networks //arXiv preprint arXiv:1512.01712. – 2015.

зависимости от разновидности РНС она разная, однако в классическом варианте это поэлементная сигмоидная функция. Выходной вектор y_t параметризует предикативную дистрибуцию $P(\widehat{x}_{t+1}|y_t)$, т.е. на основе этого вектора осуществляется предсказание о наиболее вероятном следующем входном элементе. Текстовые данные (например, слова) дискретны, их легко можно представить в виде унитарных векторов длины *K*, равной мощности словаря *V*; другими словами, на шаге *t* модели подаётся класс *k* из набора *K*, т.е. вектор x_t длины *K*, в котором на позиции *k* 1, на остальных — 0. В таком случае $P(\widehat{x}_{t+1}|y_t)$ — мультиномиальное распределение, которое можно параметризовать с помощью сигмоидальной функции активации:

$$P(\widehat{x}_{t+1} = k|y_t) = y_t^k = \frac{exp(\widehat{y}_t^k)}{\sum_{k'=1}^{K} exp(\widehat{y}_t^{k'})} \quad . \tag{23}$$

Для получения оценки правильности работы нейросети и для её обучения необходимо сравнить предсказание нейросети $\widehat{x}_{t+1}$ с реальными данными (например, с истинным содержимым следующего временного шага x_{t+1}) с помощью функции ошибок (например, перекрёстная энтропия (cross-entropy, также log loss) или расхождение Кульбака-Лейблера (KL Divergence), которую мы стремимся минимизировать: L = -lnPr(y|x). На практике обе эти функции часто эквивалентны, т.к. с точки зрения проблемы оптимизации функции потерь перекрёстная энтропия зачастую сводится к минимизации расхождения между моделируемым и предполагаемым истинным распределениями, что и есть KL Divergence[23].

Технически к исходному тексту добавляется дополнительный “пустой” (т.е. с нулевым вектором) токен начала последовательности (также часто добавляется токен конца последовательности, чтобы сеть могла остановить генерацию). Другими словами, РНС на первом шаге $t = 0$

[23] Maaten L., Hinton G. Visualizing data using t-SNE //Journal of machine learning research. – 2008. – Т. 9. – №. Nov. – С. 2579-2605

предсказывает первый значимый входной элемент последовательности x_1 на основании распределения вероятностей из выходного вектора y_0, полученного в результате первой итерации сети из нулевого вектора x_0; говоря простым языком, у сети нет априорной информации о генерируемой последовательности[24].

В силу особенностей строения РНС (вертикально и горизонтально), обычный метод обратного распространения ошибок не будет работать. Поэтому при работе с РНС часто применяется метод обратного распространения ошибок через время (Backpropagation Through Time, BPTT). На каждом шаге итерации у нас есть часть входных данных (содержимое текущего временного шага), часть истинных выходных данных (содержимое текущего временного шага) и копия нейросети (нейросеть с состоянием текущего временного шага). Распространив входные данные через сеть и сравнив предсказание $\widehat{y}_t$ с истиной y_t, мы получим ошибку для текущего шага l; нам необходимо учитывать ошибки, совершённые на каждом шаге, поэтому мы суммируем их:

$$L(x, y, \theta) = \sum_{t=1}^{T} l(y_t, \widehat{y}_t) \ . \tag{24}$$

Запишем производную этой функции по переменной θ в упрощённом виде(σ означает сигмоидную функцию, хотя может быть заменена на любую необходимую):

$$\partial_\theta L = \sum_{t=1}^{T} \partial_\theta l(y_t, \widehat{y}_t) = \sum_{t=1}^{T} \partial_{\widehat{y}_t} l(y_t, \widehat{y}_t) \left[\partial_\theta \sigma(h_t, \theta) + \partial_{h_t} \sigma(h_t, \theta) \partial_\theta h_t \right] \tag{25}$$

Но, как видно из уравнения (21), h_t — это тоже функция нескольких переменных. Соответственно,

[24] Graves A. Generating sequences with recurrent neural networks //arXiv preprint arXiv:1308.0850. – 2013.

$$\partial_\theta h_t = \partial_\theta f(x_t, h_{t-1}, \theta) + \partial_h f(x_t, h_{t-1}, \theta)\partial_\theta h_{t-1} = \quad (26)$$
$$= \sum_{i=t}^{1} \left[\prod_{j=t}^{i} \partial_h f(x_t, h_{t-1}, \theta) \right] \partial_\theta f(x_t, h_{t-1}, \theta).$$

Так как текущее скрытое состояние h_t рекурсивно зависит от предыдущего h_t, эта цепочка может быть очень длинной. Кроме вычислительных затрат это также может привести к проблемам взрыва и затухания градиента (exploding/vanishing gradient)[25]. Простыми словами, если значение производных в какой-либо момент очень маленькое или очень большое, то при умножении на него общий градиент становится значительно меньше/больше; чем больше термов в цепочке, тем больше вероятность появления таких экстремальных значений, следовательно, выше вероятность взрыва или затухания градиента. Для борьбы с подобными проблемами можно ограничить количество рекурсивных шагов алгоритма, но очевидно, что в таком случае нейросеть не способна устанавливать долговременные зависимости между элементами, только лишь кратковременные. Это в целом повышает её устойчивость к выбросам, однако очень часто — как, например, в случае с текстовыми данными — ключевые для генерации следующего элемента единицы были достаточно далеко в прошлом, вследствие чего были созданы модификации блоков РНС.

2.4. Модификации блоков рекуррентных нейронных сетей: LSTM и GRU.

В теории, достаточно большая РНС способна генерировать последовательности любой длины, но на практике эффективная длина достаточно ограничена, что приводит к нестабильности сгенерированных

[25] Pascanu R., Mikolov T., Bengio Y. Understanding the exploding gradient problem //CoRR, abs/1211.5063. – 2012. – Т. 2.

последовательностей (повторение одного и того же и т.д.). Одним из вариантов решения такой проблемы стало усиление компонента памяти.

LSTM — Long Short-Term Memory, или долгая краткосрочная память — архитектура РНС, появившаяся в 1997 году и способная соединять более 1000 временных шагов без потери краткосрочных взаимоотношений[26]. Проблему взрыва градиента можно решить клиппингом: если норма вектора превышает установленное пороговое значение (threshold), то градиент умножается на $\frac{threshold}{||g||}$, что фактически урезает его. Р. Паскану, Т. Миколов и Й. Беджио в статье “On the difficulty of training recurrent neural networks” предлагают находить это пороговое значение, которое вводит дополнительный гиперпараметр в модель, эмпирически через среднее значение нормы градиентов, однако на практике часто выбирается небольшое значение (например, от 1 до 5). Проблему же затухания градиента таким способом решить нельзя: норма градиента не обязательно будет маленькой, и скорее всего от этого будут страдать его компоненты, отвечающие за долгосрочные зависимости.

В блоке LSTM это решается репараметризацией РНС. В ней вводятся вентили (вентили), контролирующие поток информации, а также не используется функция активации внутри рекуррентных компонентов: вместо высчитывания текущего состояния h_t из предыдущего h_{t-1} с помощью матричных операций и нелинейности, она напрямую считает изменение Δh_t, которое прибавляется к h_{t-1}, чтобы получить h_t. Это видно на следующем примере: если количество временных шагов $T = 1000$, то $h_{1000} = \sum_{t=1}^{1000} \Delta h_t$; таким образом, при взятии градиента каждое изменение Δh_t внесёт значительный вклад.

[26] Hochreiter S., Schmidhuber J. Long short-term memory //Neural computation. – 1997. – Т. 9. – №. 8. – С. 1735-1780.

Стандартный LTSM-блок принимает на вход три переменные: текущие входные данные x_t, предыдущее скрытое состояние h_{t-1} и предыдущее состояние блока c_{t-1} (в LSTM два скрытых состояния: “медленное” состояние блока *c*, борющееся с проблемой затухания градиента, и “быстрое” *h*, позволяющее нейросети принимать краткосрочные сложные решения; грубо говоря, долговременная и рабочая памяти). После операций над ними он выдаёт текущее скрытое состояние h_t (из которого можно получить выход y_t и предсказание x_{t+1} — стоит помнить, что результат работы модели на текущем шаге используется для предсказания входных данных следующего шага) и текущее состояние блока c_t. Потоком данных управляют три вентиля, по сути представляющих собой маленькие нейронные сети. Первый, вентиль забывания, для каждого числа из прошлого состояния блока c_{t-1} на основе текущих входных данных x_t и прошлого скрытого состояния h_{t-1} с помощью сигмоидной функции активации σ выдаёт число от 0 до 1, контролируя забывание части прошлых данных:

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)\,. \tag{27}$$

Второй, входной вентиль, определяет используемую информацию с текущего шага. Он совмещает две функции, а именно *i*, которая выбирает данные для обновления, и *j*, которая предлагает кандидатов для добавления в долговременную память *c*:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)\,, \tag{28}$$

$$j_t = tanh(W_{xj}x_t + W_{hj}h_{t-1} + b_j)\,. \tag{29}$$

Наконец, третий, выходной вентиль, определяет, какую часть входных данных и скрытого состояния выдать:

$$o_t = tanh(W_{xo}x_t + W_{ho}h_{t-1} + b_o)\,. \tag{30}$$

После этого LSTM обновляет долговременную (состояние блока) и рабочую (скрытое состояние) памяти ($\odot$ обозначает произведение Адамара, т.е. поэлементное умножение):

$$c_t = c_{t-1} \odot f_t + i_t \odot j_t , \tag{31}$$

$$h_t = tanh(c_t) \odot o_t . \tag{32}$$

Вариантом LSTM можно считать представленные в 2014 году блоки GRU — Gated Recurrent Units, управляемый рекуррентный блок[27]. Его идея аналогична идее LSTM, основное же отличие заключается в меньшем количестве вентилей и состояний.

Первый вентиль, вентиль сброса, определяет, какую часть памяти (информации скрытого состояния) стоит забыть; при значениях, близких к 0, блок игнорирует прошлое скрытое состояние (как бы сбрасывает его) и использует только входные данные:

$$r_t = \sigma(W_{xr} x_t + W_{hr} h_{t-1} + b_r) . \tag{33}$$

Второй вентиль, вентиль обновления, по сути совмещает в себе входной вентиль и вентиль забывания, определяя, какую часть прошлой информации передать в будущее:

$$z_t = \sigma(W_{xz} x_t + W_{hz} h_{t-1} + b_z) . \tag{34}$$

Из этого определяется промежуточное состояние памяти $\bar{h}_t$ с оставшейся релевантной прошлой информацией из вентиля сброса, а потом — текущее состояние памяти h_t с релевантной новой информацией из вентиля обновления:

$$\bar{h}_t = tanh(W_{x\bar{h}} x_t + r_t \odot W_{h\bar{h}} h_{t-1} + b_{\bar{h}}) , \tag{35}$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \bar{h}_t . \tag{36}$$

[27] Cho K. et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation //arXiv preprint arXiv:1406.1078. – 2014.

GRU, будучи более молодой моделью, получает всё больше признания, однако LSTM не сдаёт позиций. Более простая структура блока GRU делает его более вычислительно эффективным, однако более сложная структура LSTM даёт больший контроль над потоком информации. На практике оба варианта показывают себя приблизительно одинаково, показывая незначительное преимущество в разных видах заданий. Так, например, в стате “An Empirical Exploration of Recurrent Network Architectures”[28] авторы провели ряд экспериментов, в результате которых GRU победила LSTM во всех заданиях, кроме языкового моделирования, а LSTM с некоторыми изменениями (такими как, например, дропаут) была даже лучше в некоторых заданиях. Из этого можно сделать вывод, что выбор архитектуры блоков РНС зависит от задания и, скорее всего, иных факторов, включая доступные данные, используемые техники и т.д., что подтверждается в статье “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”[29]. Основываясь на этом, в данной работе мы используем LSTM-блоки, потому что, как будет видно в следующем разделе, языковое моделирование играет огромную роль в выбранной нами архитектуре, переводящей последовательность элементов (слов) исходного текста в последовательность элементов целевого.

2.5. Перевод последовательности в последовательность при помощи архитектуры Seq2Seq.

Итак, рекуррентная нейронная сеть принимает на вход содержимое одного временного шага, например, слово, и пытается предсказать содержимое следующего, например следующее слово на основе построенной языковой модели. В контексте решаемой нами задачи генерации новостных

[28] Jozefowicz R., Zaremba W., Sutskever I. An empirical exploration of recurrent network architectures //International Conference on Machine Learning. – 2015. – С. 2342-2350.
[29] Chung J. et al. Empirical evaluation of gated recurrent neural networks on sequence modeling //arXiv preprint arXiv:1412.3555. – 2014.

заголовков это кажется не очень полезным: подав нейросети новость на вход, на выходе мы получим ту же новость, но смещённую на один временной шаг влево. Найти им применение в этой области можно в области автоматического машинного перевода, ведь в некотором роде генерация новостных заголовков похожа на перевод с одного языка (языка новостей) на другой (язык заголовков).

На заре становления машинного перевода сервисы пытались найти пословный аналог, иногда расширяясь до n-грамм. Всё изменилось с приходом статистического перевода, из которого родилась идея Encoder-Decoder модели[30]. Она предназначена для Seq2Seq (Sequence to Sequence, “последовательность-в-последовательность”) — техники для решения задач перевода последовательности в последовательность от Google 2017 года[31]. В данной работе мы рассматриваем Encoder-Decoder как частное решение Seq2Seq в задачах обработки естественного языка, поэтому используем оба этих термина взаимозаменяемо, имея в виду одно и то же и фактически обозначая Seq2Seq как архитектуру. Encoder-Decoder идеальна для работы с текстами, т.к. способна улавливать как семантическую, так и синтаксическую структуру фраз, или, если мы работаем со словами, слов, что позволяет ей генерировать осмысленные и грамматичные последовательности.

Как можно догадаться из названия, архитектура Encoder-Decoder состоит из двух элементов, каждый из которых является РНС (то есть небольшой языковой моделью): кодировщик (Encoder), который кодирует исходный текст произвольной длины в некоторый смысловой вектор фиксированной длины (впрочем, это необязательно[32]), и декодировщик

[30] Cho K. et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation //arXiv preprint arXiv:1406.1078. – 2014.
[31] Neural Machine Translation (seq2seq) Tutorial [Электронный ресурс] / Luong M.T., Brevdo E., Zhao R. — URL: https://github.com/tensorflow/nmt (дата обращения 25.05.2019). — 2017.
[32] Bahdanau D., Cho K., Bengio Y. Neural machine translation by jointly learning to align and translate //arXiv preprint arXiv:1409.0473. – 2014.

(Decoder), который на основе этого смыслового вектора генерирует целевой текст произвольной длины. Концептуально это можно выразить как две последовательные задачи: суммаризация (понимание) текста и генерация нового текста (пересказ). Говоря же языком вероятностей, Seq2Seq старается выучить условное распределение последовательности произвольной длины, зависящее от другой последовательности произвольной длины $p(y_1, ..., y_T | x_1, ..., x_T)$.

Энкодер поочерёдно обрабатывает символы входной последовательности **x**=$\{x_1, ..., x_T\}$, изменяя скрытое состояние используемой в нём РНС h (или несколько состояний в зависимости от используемого варианта блоков); состояние энкодера в конце последовательности (технически выраженной в виде особого токена вроде <END>) становится вектором смысла (кратким содержанием) всей входной последовательности, который принимает декодер, поочерёдно генерирующий символы выходной (целевой) последовательности **y**=$\{y_1, ..., y_T\}$ и вычисляющий скрытое состояние декодера на каждом временном шаге h_t, зависящее от предыдущего скрытого состояния h_{t-1}, предыдущего токена y_{t-1} и содержания исходного текста c:

$$h_t = f(h_{t-1}, y_{t-1}, c). \quad (37)$$

Тогда вероятность следующего символа будет параметризоваться как

$$P(y_t | y_{t-1}, ..., y_1, c) = g(h_t, y_{t-1}, c). \quad (38)$$

Можно заметить, что в формулах выше следующий токен зависит только от предыдущего, а не от всей последовательности. Это допущение, возможное благодаря свойству марковской модели первого порядка:

$$p(y_1, ..., y_T) = \prod_{t=1}^{T} p(y_t | y_{t-1}). \quad (39)$$

Таким образом, в силу итеративной природы РНС на каждом шаге необходимо обращать внимание только на предыдущий шаг, т.к. он хранит в себе все шаги до него. Это верно также и для скрытых состояний h. В свете этого рассмотрим вектор содержания исходного текста *c*.

Итак, смысловым вектором исходного текста *c* становится последнее скрытое состояние энкодера h_T (т.к. в LSTM два вектора состояния, то смысловой вектор состоит из двух векторов, включая и последнее состояние блока, но для простоты объяснения мы будем рассматривать классический вариант РНС). Также можно получить этот вектор, взяв взвешенную сумму состояний энкодера[33], но чаще берётся только последнее. Так как скрытое состояние декодера на каждом шаге h'_t зависит от предыдущего скрытого состояния h'_{t-1} и смыслового вектора *c*, фактически на первом шаге оно высчитывается на двух одинаковых векторах. Начиная со второго шага, они отличаются, но каждое следующее скрытое состояние хранит информацию о предыдущем, включая первое, аналогичное вектору содержания исходного текста. Поэтому на практике в библиотеках машинного обучения в языке Python для повышения эффективности вычислений смысловой вектор не передаётся каждому шагу как дополнительный параметр, только первому, а затем считается частью скрытого состояния[34,35]. Encoder-decoder архитектура появилась в 2014 году, и с того момента было придумано несколько её разновидностей, например, Трансформер (Transformer)[36], Pointer-Generator Network[37] и другие.

---

[33] J.W.G. Putra, H. Kobayashi, N. Shimizu. Experiment on Using Topic Sentence for Neural News Headline Generation //Proceedings of the 24th Annual Meeting of the Speech Processing Society of Japan (March 2018). — 2018.

[34] A ten-minute introduction to sequence-to-sequence learning in Keras [Электронный ресурс] // Keras Docs. URL: https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html (дата обращения 08.06.2019). — 2017.

[35] Translation with a Sequence to Sequence Network and Attention [Электронный ресурс] // Pytorch Docs. URL: https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html (дата обращения 08.06.2019). — 2017.

[36] Vaswani A. et al. Attention is all you need //Advances in neural information processing systems. – 2017. – С. 5998-6008.

[37] See A., Liu P. J., Manning C. D. Get to the point: Summarization with pointer-generator networks //arXiv preprint arXiv:1704.04368. – 2017.

Мы выбрали Seq2Seq как основу для нашей модели по нескольким причинам. Во-первых, результаты Трансформера лучше, но не на порядок, и возможно модификации Seq2Seq смогут покрыть эту разницу. Во-вторых, в литературе очень мало сведений о применимости Seq2Seq к русскому языку — как, впрочем, и других архитектур — поэтому имеет смысл начать с более простых моделей. В-третьих, Seq2Seq лежит в основе большинства модификаций, поэтому более глубокое понимание её может быть полезным при работе с ними. В-четвёртых, на данном этапе проекта мы ставим перед собой задачу создания базового решения, которое в дальнейшем могло бы расширяться и улучшаться.

Как было отмечено выше, многие разновидности Encoder-decoder моделей являются развитием Seq2Seq, или Seq2Seq с модификациями. В первой главе мы уже отмечали некоторые техники и приёмы для оптимизации нейронных сетей. РНС, будучи нейросетями, также используют их, однако у них есть свои особенности, ограничения и возможности. Более того, Seq2Seq архитектура, будучи разновидностью рекуррентной архитектуры, имеет схожести с РНС, но кроме этого обладает и собственными техниками и приёмами. В следующем разделе мы кратко рассмотрим некоторые из её общих и собственных механик, в частности механизм внимания, лучевой поиск и рекуррентный дропаут.

2.6. Техники и приёмы для улучшения Seq2Seq моделей.

Нейронная сеть принимает на вход данные, производит над ними операции и выдаёт выходные данные, или, другими словами, состоит из трёх частей: входного слоя, скрытого слоя (в глубоких сетях их несколько и они большие) и выходного слоя. Соответственно, повысить её эффективность можно на любом из этих компонентов.

Кроме хорошей предварительной обработки текстовых данных на входном слое можно попробовать сделать более эффективным кодирование токенов. Напомним, что для того, чтобы НС могла работать со словами, необходимо найти способ их численного представления. В контексте РНС это означает их отражение в векторное пространство: из слов корпуса составляется словарь, где каждому слову соответствует уникальный (или относительно уникальный) числовой вектор. В самом простом случае это унитарное кодирование, однако размерность каждого вектора будет огромной (равна мощности словаря), а информативность маленькой (т.к. только 1 значение отлично от 0). Намного эффективнее использовать плотные векторы, полученные, например, в результате работы алгоритмов Word2Vec или подобных: их размерность задаёт сам разработчик, обычно это значение равно 100 или 300, однако может доходить и до 1000, что всё равно на порядок меньше мощности словаря. Но даже такие вектора занимают место в оперативной памяти, что ограничивает их максимальное число. Эта проблема часто решается урезанием словаря: берётся n самых частотных единиц, остальные заменяются специальным токеном (например,<UNK>, от “unknown” — “неизвестный”). Большое количество неизвестных токенов препятствует “пониманию” текста машиной, что в свою очередь мешает процессу генерации нового текста на основе исходного. Стоит учитывать, что если мы работаем на уровне слов, то без лемматизации или стемминга (т.к. мы учим нейросеть генерировать грамматически верные предложения, то так и происходит) под единицами словаря мы имеем в виду словоформы; таким образом, для английского языка, где у слов не так много вариантов, это решение относительно эффективно, но для русского языка, в котором даже падежные формы одного и того же слова будут считаться разными единицами, мощность “урезанного” словаря будет огромна, что неэффективно. Аналогичная проблема наблюдается и на выходном слое

(часто для выходного слоя создаётся отдельный словарь по аналогии с задачами автоматического перевода, где невозможно использовать общий словарь), однако у неё есть свои небольшие нюансы: в частности, машине не обязательно выдавать специальный токен неизвестной единицы, т.к. она должна быть способна к перефразированию, но в то же время есть вероятность потери смысла вследствие отсутствия возможности к конкретизации (например, замена названия компании из оригинала простым словом “компания”).

Интересным вариантом решения проблемы слов вне словаря, особенно для флективных языков вроде русского, может служить алгоритм Byte-Pair Encoding (BPE, кодирование байтовых пар), в основе которого лежит идея создания частотных пар символов[38]. Он разбивает исходные тексты на отдельные символы (буквы и пр.), после чего считает их совместное появление; частые пары объединяются в одну n-грамму (в данном случае биграмму, хотя в ходе дальнейшей работы алгоритма они объединяются в ещё более крупные единицы). Обычно такие пары не пересекают границы слов, т.е. максимально возможное объединение символов будет равно слову, а не фразам и т.п., хотя на практике чаще всего выделяются различные морфемы. Интересным побочным эффектом такого подхода (желательным или нежелательным) может стать возможность декодера генерировать новые слова из подобных “байтов”.

Другим интересным решением подобной проблемы может стать механизм копирования (Copy Mechanism), который берёт единицу из исходного текста и копирует её в целевой[39]. В его основе лежит либо соотношение позиций слов, либо механизм внимания, имплементируемый на скрытых слоях.

[38] Sennrich R., Haddow B., Birch A. Neural machine translation of rare words with subword units //arXiv preprint arXiv:1508.07909. – 2015.
[39] Gu J. et al. Incorporating copying mechanism in sequence-to-sequence learning //arXiv preprint arXiv:1603.06393. – 2016.

Teacher Forcing (“Форсирование при обучении”) — техника, повышающая стабильность и правильность работы сети на этапе обучения. В теории выход рекуррентной сети опережает вход на 1 шаг, т.к. её задача — предсказывать следующий токен, и выход текущего временного шага y_t становится входом следующего x_{t+1}. В начале процесса параметры сети выбраны случайным образом, поэтому скорее всего уже на первом шаге нейросеть выдаст неверный ответ, в результате чего неверным станет и следующий, и так ошибка будет накапливаться, в итоге делая итоговую последовательность крайне далёкой от эталонной (истинной) из реальных данных. Для борьбы с этим на этапе обучения можно не использовать выход нейросети, а подкреплять её, передавая ей слова истинной последовательности вместо её собственных ответов[40]. Это должно уменьшить время, требуемое для обучения модели. Однако это создаёт т.н. Exposure Bias (“Привыкание к истине”): при использовании Teacher Forcing во время тренировки модель основывает свои предсказания на скрытом состоянии и истинных токенах, а во время тестирования модель вместо истинных токенов использует предсказанные ей самой, что может привести к накапливанию ошибок на этапе декодирования[41].

Последний небольшой приём, позволяющий улучшить результаты работы нейросети, заключается в инверсии входной последовательности так, что первое слово становится последним, последнее — первым: вместо картирования $\{a, b, c\} -> \{a', b', c'\}$ предлагается делать картирование $\{c, b, a\} -> \{c', b', a'\}$ [42]. Это может быть эффективным, т.к. при переводе (в парадигме которого появилась эта идея) с одного языка на другой при условии некоторой их схожести позиция отдельных единиц скорее всего

[40] Williams R. J., Zipser D. A learning algorithm for continually running fully recurrent neural networks //Neural computation. – 1989. – Т. 1. – №. 2. – С. 270-280.
[41] Shi T. et al. Neural Abstractive Text Summarization with Sequence-to-Sequence Models //arXiv preprint arXiv:1812.02303. – 2018.
[42] Sutskever I., Vinyals O., Le Q. V. Sequence to sequence learning with neural networks //Advances in neural information processing systems. – 2014. – С. 3104-3112.

будет совпадать — или по крайней мере будут присутствовать определённые структурные сходства. Таким образом, переворачивая входную последовательность мы сближаем слово первой позиции в ней (a) со словом первой позиции в выходной последовательности (a'), что позволяет нейросети точнее установить связи между ними, что повышает её эффективность в генерации первого токена целевой последовательности, что в свою очередь, учитывая рекуррентную природу инференции, может положительно повлиять на выбор следующих токенов (через уменьшение вероятности начать генерацию не с нужной единицы), улучшая полученный текст в целом.

На выходном слое декодер генерирует слова один за одним, выбирая наиболее вероятное. Действительно, если мы аппроксимируем зависимость следующего токена от всех предыдущих до только одного предыдущего ($p(w_1, w_2, w_3) = p(w_1)p(w_2|w_1)p(w_3|w_2)$) и предполагаем, что скрытое состояния хранит информацию о всех предыдущих шагах, то кажется логичным максимизировать вероятность одного следующего слова. Но это аппроксимация, и в силу комплексности естественного языка она не идеальна; на этапе тренировки она намного вычислительно эффективнее, чем просчитывать полную цепочку зависимостей рекурсивно, но на этапе инференции (генерации) имеет смысл выбрать качество над производительностью. Лучевой поиск создаёт k кандидатов цепочек токенов с наибольшей общей вероятностью, уменьшая k на 1 при достижении в одной из них символа конца последовательности; т.к. цепочки могут достигнуть конца на разных временных шагах, то необходимо нормализовать вероятности относительно длины[43]. Легко видеть, что при $k = 1$ лучевой поиск аналогичен обычному, “жадному” поиску.

[43] Freitag M., Al-Onaizan Y. Beam search strategies for neural machine translation //arXiv preprint arXiv:1702.01806. – 2017.

Механизм внимания — один из важнейших приёмов при построении рекуррентных моделей, т.к. он позволяет моделировать зависимости между элементами вне зависимости от расстояния между ними в последовательности[44]. Функцию внимания — которую можно рассматривать как дополнительный слой сети — можно объяснить как картирование запроса (query) и пары ключ-значение к выходной последовательности; последняя определяется как взвешенная сумма значений, веса которых получаются в результате функции над запросами и соответствующими ключами. В архитектуре Encoder-Decoder ключи и значения извлекаются из энкодера, а запросы — из предыдущего временного слоя декодера таким образом, что каждый запрос (скрытое состояние декодера на временном шаге h'_{t-1}) "направляется" к каждой паре ключ-значение (скрытые состояния энкодера $\{h_1, ..., h_t\}$). Самовнимание (Self-attention) извлекает все векторы из одного компонента (кодировщик или декодировщик) по аналогичной схеме, позволяя РНС отслеживать, что уже было сгенерировано.

Все элементы функции внимания являются векторами: **q** для запроса, **k** для ключа и **v** для значения; наборы всех векторов являются матрицами Q, K и V соответственно. Тогда внимание определяется как softmax (функция, преобразующая вектор в вектор такой же размерности, в котором все координаты исходного вектора переведены в диапазон [0;1] так, что они суммируются к 1) скалярного произведения матрицы запросов и ключей, нормализованных по квадратному корню размерности вектора ключей, скалярно умноженный на матрицу значений:

$$Attention(Q, K, V) = softmax(\frac{QK^t}{\sqrt{d_k}})V \ . \qquad (40)$$

[44] Vaswani A. et al. Attention is all you need //Advances in neural information processing systems. – 2017. – С. 5998-6008.

Конечно, это не единственный вариант функции внимания[45], однако он использует матричные операции, что позволяет ему быть крайне эффективным с вычислительной точки зрения; кроме того, глубокий анализ различных имплементаций механизма внимания выходит за рамки задач данной работы. Впрочем, стоит отметить многоглавое внимание (Multihead Attention), которое конкатенирует несколько функций внимания (каждое из которых называется головой (Head)), позволяя модели фактически обращать внимания на информацию в нескольких репрезентативных подпространствах.

Важность механизма внимания можно оценить по тому, что на основе него была создана новая архитектура для работы с последовательностями — Трансформер (Transformer), который по сути заменяет рекуррентные слои слоями внимания[46]. Кроме того, он является важной частью механизма копирования и архитектуры Pointer-Generator Network (PGN), который, в свою очередь, инкорпорирует копирование. PGN совмещает в себе экстрактивный (извлечение оригинальных строк) и абстрактивный (перефразирование) подходы к реферированию, фактически выбирая на каждом шаге один из двух режимов работы: генерация элемента целевого текста или указывание (pointing) на элемент оригинального. Такие подходы, основанные на использовании слов из текста-источника, позволяют решить проблему слов вне словаря, обогащая выходную последовательность семантически, но повышают вероятность нарушения локальной грамматической когезии.

В разделе 1.4.4. упоминался дропаут — техника для борьбы с переобучением модели через исключение отдельных нейронов из процесса обучения. Фактически при каждом прямом распространении сигналов архитектура сети меняется, заставляя нейроны быть универсальными. Но

[45] Luong M. T., Pham H., Manning C. D. Effective approaches to attention-based neural machine translation //arXiv preprint arXiv:1508.04025. – 2015.
[46] Vaswani A. et al. Attention is all you need //Advances in neural information processing systems. – 2017. – С. 5998-6008.

рекуррентные нейронные сети разворачиваются во времени, и на каждом временном шаге фактически создаётся копия нейросети. Применение дропаута нарушает симметричность сети во времени, что на практике часто не приносит позитивных результатов, и иногда даже наоборот мешает эффективной работе сети. Поэтому был создан рекуррентный дропаут (recurrent dropout)[47], который использует одну и ту же "маску прореживания" на каждом временном шаге.

Также к приёмам улучшения Seq2Seq можно отнести двунаправленный LSTM, о котором рассказывается в следующей главе.

2.7. Добавление "вертикальной" глубины в рекуррентные нейронные сети.

Как было отмечено в разделе 2.2, глубина РНС скорее "горизонтальная", происходящая из разворачивания сети во времени, нежели "вертикальная", архитектурная. Это может быть слабостью, если структура входных данных сложная, особенно иерархически, но они проходят только через 1 слой, который можно считать своего рода фильтром, или необходимо моделировать отношения между единицами на нескольких временных шкалах. Например, речь на самом низком уровне состоит из кратковременных (т.е. быстро начинаются и быстро заканчиваются) фонем, более долговременных слогов, слов, фраз и т.д.[48]. Кроме того, в статье "Google's neural machine translation system: Bridging the gap between human and machine translation" отмечается, что обычно глубокие LSTM превосходят неглубокие, и в своём исследовании они пришли к выводу, что системы нейронного машинного перевода (Neural Machine Translation) показывают лучшие результаты, если и кодировщик, и декодеровщик достаточно

[47] Gal Y., Ghahramani Z. A theoretically grounded application of dropout in recurrent neural networks //Advances in neural information processing systems. – 2016. – С. 1019-1027.
[48] Hermans M., Schrauwen B. Training and analysing deep recurrent neural networks //Advances in neural information processing systems. – 2013. – С. 190-198.

глубоки, чтобы улавливать “трудноуловимые нарушения нормы” (subtle irregularities) в текстах исходного и целевого языков.

В целом архитектура глубокой РНС (ГРНС) аналогична структуре обычной, только в ней добавлены слои. Единственное отличие заключается в том, что если на первый слой $l = 1$ подаётся содержимое входного слоя (т.е. векторизованное слово входной последовательности x_t), то на последующие подаётся скрытое состояние предыдущего h^{l-1}:

$$h_t^1 = f(x_t, h_{t-1}^1), \tag{41}$$

$$h_t^1 = f(h_t^{l-1}, h_{t-1}^1). \tag{42}$$

Результат работы сети (выходная единица) определяется на основе скрытого состояния последнего слоя. Впрочем, также предлагается использовать взвешенную сумму состояний всех скрытых слоёв, что в теории должно помочь правильному вычислению градиента при backpropagation и также может дать некоторую информацию о роли каждого слоя в процессе инференции.

ГРНС используются относительно редко, но часто используется приём, частично основанный на увеличении глубины: двунаправленные LSTM (Bi-Directional LSTM). В нём добавляется второй слой (LSTM блок), который обрабатывает входную последовательность инверсивно, т.е. в обратном порядке. Таким образом входная единица порождает два скрытых состояния, прямое и инверсивное. Это позволяет нейросети как бы заглядывать в будущее, определяя зависимости текущей единицы (слова) не только от предыдущих, но и от последующих. Впрочем, очевидно, что во время тренировки это могло бы сработать, т.к. мы знаем всю последовательность, но во время инференции у нас нет информации о будущем, только текущий токен и уже сгенерированные. Как было показано на примере дропаута и Teacher Forcing, создавать различия в этапах

тренировки и инференции крайне нежелательно, поэтому двунаправленный LSTM имеет относительно ограниченную применимость (восстановление пропущенных слов, извлечение именованных сущностей и т.д.) и вряд ли подходит для генерации новостных заголовков.

2.8. Оценка качества работы рекуррентной нейронной сети, генерирующей тексты.

Обычно для в качестве оценки моделей, генерирующих текст на основе другого текста, используются BLEU Score[49] или ROUGE Score[50]. Метрика BLEU (Bilingual Evaluation Understudy, замена билингвальной оценки) используется для оценки точности перевода текста; она предполагает, что чем ближе сгенерированный текст к оригинальному, тем он лучше, поэтому проверяет совпадением n-грамм (обычно несколько n, от 1 до 4) в обоих текстах. Метрика ROUGE (Recall-Oriented Understudy for Gisting Evaluation, ориентированная на полноту замена сущностной оценке) похожа на BLEU, т.к. тоже сравнивает совпадение n-грамм текстов. Разница между ними прежде всего концептуальная (т.к. существуют разные версии формул): BLEU проверяет, сколько n-грамм из сгенерированного текста присутствует в оригинальном, ROUGE проверяет, сколько n-грам из оригинального текста присутствует в сгенерированном. Метрики выдают результат в диапазоне от 0 до 1. Даже из такого краткого представления становятся очевидны проблемы этих метрик: предпочтение генерации коротких текстов (т.к. меньше вероятность расхождения n-граммного состава, особенно в случае с униграммами) и оценка формальной схожести текстов вместо смысловой (например, замена “краткий” на “короткий” будет расцениваться как ошибка). Несмотря на эти ограничения, эти метрики пользуются огромной популярностью.

---

[49] Papineni K. et al. BLEU: a method for automatic evaluation of machine translation //Proceedings of the 40th annual meeting on association for computational linguistics. – Association for Computational Linguistics, 2002. – С. 311-318.

[50] Lin C. Y. Rouge: A package for automatic evaluation of summaries //Text Summarization Branches Out. – 2004.

Выводы по главе 2

Рекуррентные нейронные сети — обобщение нейронных сетей для временных данных (последовательностей), а Seq2Seq Encoder-Decoder — один из вариантов рекуррентной архитектуры. РНС состоят из блоков, разворачивающихся во времени, и поэтому способных эффективно моделировать временные отношения между единицами, например, словами текстов. В задачах обработки естественного языка рекуррентные нейронные сети могут использоваться для распознавания речи, извлечения именованных сущностей, автоматического перевода и др.

Encoder-Decoder состоит из двух РНС, одна из которых отвечает за своего рода понимание исходного текста, а вторая — за генерацию нового.

В реальных задачах классические рекуррентные блоки зачастую не очень результативны, так как имеют ограниченную применимость в моделировании долговременных отношений, поэтому обычно используются различные улучшенные варианты, например, LSTM и GRU.

Глубина в РНС появляется при разворачивании блоков во времени, где каждая их инстанция (фактически копия) может рассматриваться как отдельный слой нейросети в классическом понимании. Однако можно усилить её, добавив дополнительные рекуррентные блоки на каждый временной шаг, что может позволить сети выделять зависимости в данных более гранулярно, на разных временных шкалах, или более иерархично, что может положительно сказаться на результатах её работы.

РНС обладают своим набором техник и приёмов, позволяющих их улучшить. Самыми проминентными являются механизм внимания и лучевой поиск, которые в последнее время считаются де-факто стандартным расширением Encoder-Decoder архитектуры. Без них не обходятся State of the Art (SOTA), т.е. лучшие решения, но базовые (baseline) решения их не требуют. Особенно аккуратно стоит использовать техники оптимизации

классических нейросетей, т.к. не все из них подходят, либо же требуют некоторых изменений.

Глава 3.

3.1. Практическое применение глубоких рекуррентных нейронных сетей для генерации новостных заголовков.

В данной главе мы используем информацию, полученную в ходе работы над предыдущими, чтобы имплементировать свою Seq2Seq Encoder-Decoder модель. В первом разделе рассказывается об используемых текстовых данных и их обработке, во втором — о нейросети с концептуальной и программной точек зрения, в третьем приводятся результаты её работы.

При построении модели мы ориентировались на теоретические основания и на результаты похожих статей, однако некоторые аспекты требовали нахождения компромиссов в силу практических причин. Например, часто глубокие нейросети требуют огромного количества эпох (т.е. времени на тренировку) — недели или даже месяцы[51] в зависимости от их размеров, количества данных и т.д. В силу ограниченности вычислительных ресурсов и времени нам пришлось сильно уменьшить время тренировки, что, конечно, может сказаться на результатах работы негативным образом.

Как упоминалось в разделе 2.4, Encoder-Decoder состоит из 2 РНС, первая из которых извлекает из исходного текста его смысл, а вторая генерирует новый текст по выученной ей языковой модели с учётом такого контекста. Соответственно можно сказать, что заголовок хороший, когда а) из новости верно извлечён смысл (кодировщик работает верно); б) заголовок полностью и недвусмысленно отражает смысл новости (декодировщик обуславливается смысловым вектором из кодировщика); в) сгенерированный текст семантически и грамматически связан (декодировщик построил верную языковую модель). Однако проверить правильность извлечения смысла из

[51] Lopyrev K. Generating news headlines with recurrent neural networks //arXiv preprint arXiv:1512.01712. – 2015.

новости не представляется возможным, т.к. неверное отражение новости может судить как о проблемах в пункте а), так и в пункте б), поэтому имеет смысл объединить их в один критерий “правильность отражения смысла”. Таким образом, хороший сгенерированный заголовок должен обладать межтекстовой (между своим текстом и текстом новости) смысловой когерентностью и внутритекстовой (между своими единицами) семантической и синтаксической когезией.

3.2. Данные: корпус и словари.

Для обучения модели мы использовали архив новостей издания Lenta.ru[52]. Он состоит из 739,346 пар новостных текстов и заголовков общим объёмом в примерно 1.8 гигабайт. В разделе 1.4.4. мы рассказывали, что обычно данные разбиваются на 3 выборки (набора). Чем меньше тренировочных данных, тем меньше в них вариантивности, и тем самым модель хуже генерализует; чем меньше тестовых данных, тем менее надёжна оценка мощности модели. Поэтому деление 50 на 50 не очень эффективно и не рекомендуется. Первое разделение — тренировочные и тестовые данные, и обычно это делается в соотношении 80/20% или 75/25%. Второе — тренировочные и валидационные, которые также разбиваются примерно в соотношении 80/20%. В данном исследовании мы решили не отходить от принятых практик и разбили набор данных на тренировочную и тестовую выборку в соотношении 80/20% (591,477 и 147,869 текстов соответственно), после чего разбили тренировочную на собственно тренировочную и валидационную в таком же соотношении (473,182 и 118,295).

[52] News dataset from Lenta.Ru Corpus of Russian news articles collected from Lenta.Ru [Электронный ресурс] // Dmitry Utkin at Kaggle. URL: https://www.kaggle.com/yutkin/corpus-of-russian-news-articles-from-lenta (дата обращения 14.02.2019). — 2019.

Кроме того, мы убрали знаки пунктуации (кроме тире, которые фактически являются частью слова) с помощью регулярных выражений и перевели их в нижний регистр.

3.3. Архитектура нейронной сети, подбор гиперпараметров.

РНС обрабатывают последовательности по одному элементу, и поэтому способны переводить последовательности произвольной длины в последовательности произвольной длины. На практике же эффективность вычислений нейронных сетей повышается, если перевести данные в матричный вид, но в случае с рекуррентными НС это приводит к необходимости как-то фиксировать длину данных. Для подачи текстов в НС их необходимо привести в тензорную форму с фиксированными измерениями. В общем случае это трёхмерный тензор с размерностью (размер минипакета, количество временных шагов, размерность векторного представления слов). На входе нейросети мы используем перевод слов в векторное пространство (Word2Vec) и можем задать размерность этих векторов самостоятельно, однако на выходе нам необходимо получить мультиномиальное распределение вероятностей, поэтому приходится использовать унитарные векторы с размерностью, равной мощности словаря. Таким образом, для определения формы тензоров нам необходимо определить желаемое число минипакетов, измерение векторного представления слов, мощность словаря и длины статей и заголовков. Для определения последних двух мы провели статистический анализ данных. После удаления выбросов (6632 статей и 3361 заголовков) данные получили распределение, близкое к Гауссовскому. Средняя длина статей составила 176.8 слов (максимальная — около 8000), среднеквадратичное отклонение — 64.3 слова; средняя длина заголовков составила 7.5 слов (максимальная — 17), отклонение — 1.75. По эмпирическому правилу “68-95-99.7”, для выборок с нормальным распределением диапазон значений плюс-минус одно

стандратное отклонение от среднего значения обычно покрывает примерно 68% наблюдаемых данных, плюс-минус два — 95% данных, плюс-минус три — 99.7% данных. Таким образом, мы можем установить максимальную длину статей равной 370, заголовков — 13. Несмотря на это, мы решили немного увеличить эти числа (до 500 и 20 соответственно), чтобы частично покрыть больше данных, т.к. 700 тысяч текстов — это немало, но действительно крупные нейросети тренируются на десятках миллионов, и для дальнейшего масштабирования модели потребуется добавлять в неё тексты в том числе и из других источников, средняя длина которых может отличаться.

На рисунках 2 и 4 представлены гистограммы распределения длин всех заголовков и новостей, на рисунках 3 и 5 — гистограммы распределения длин заголовков и новостей без статистических выбросов. По оси ординат отложены количество текстов, по оси абсцисс — длины, прямоугольники отражают интервалы.

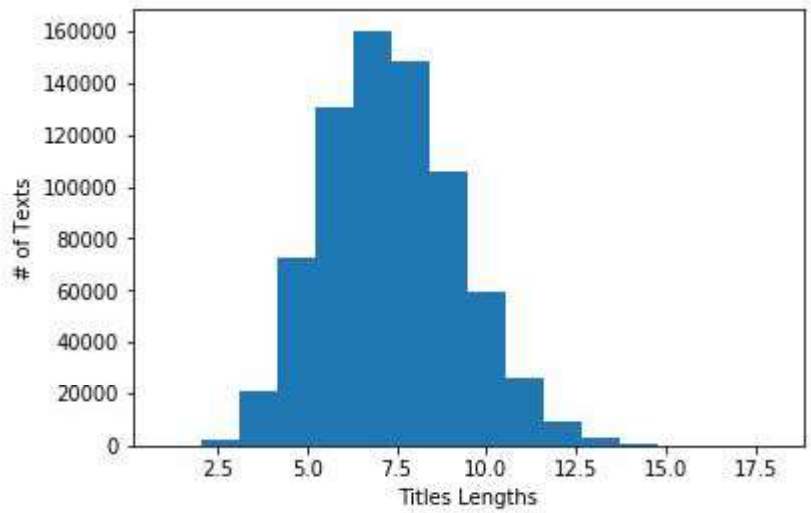

Рисунок 2 — Гистограмма распределения длин заголовков

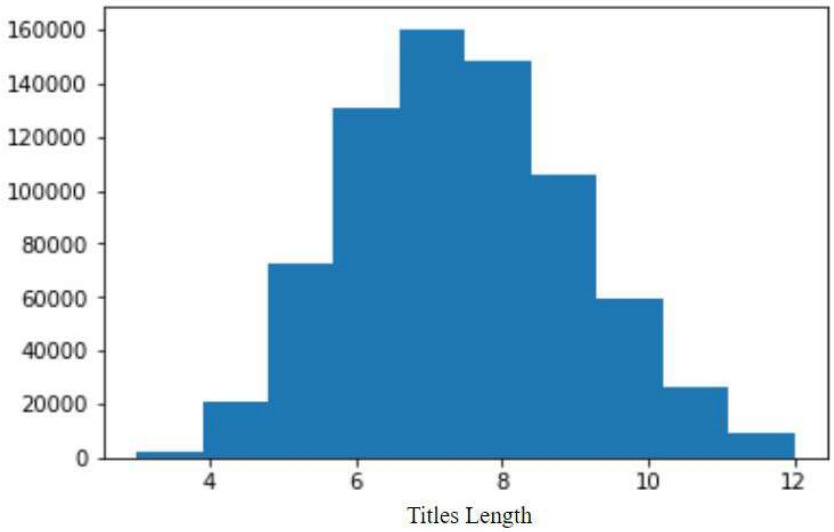

Рисунок 3 —Гистограмма распределения длин заголовков (без выбросов)

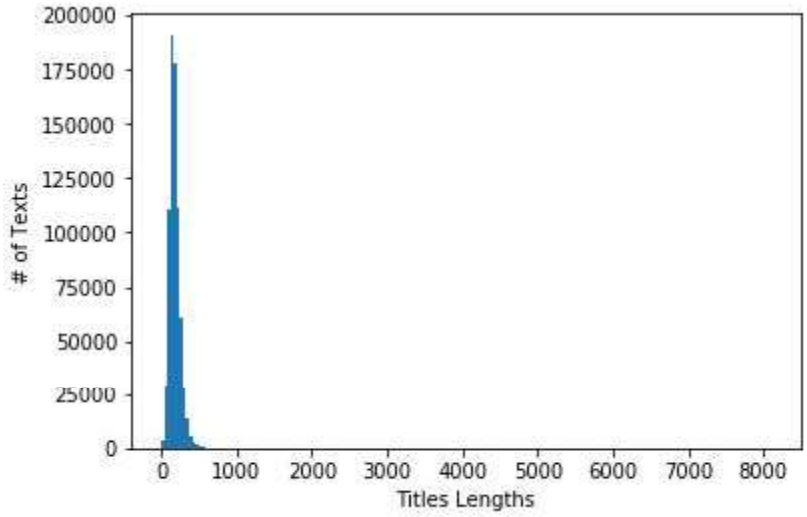

Рисунок 4 —Гистограмма распределения длин новостей

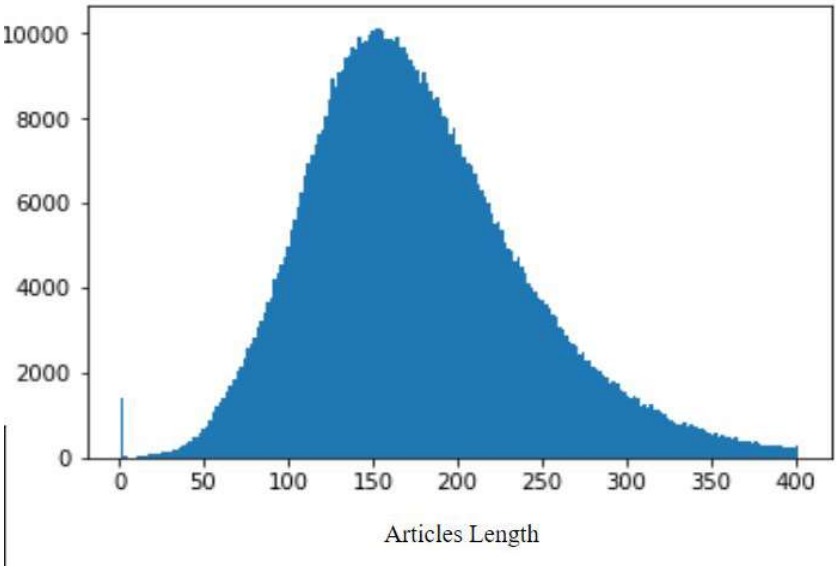

Рисунок 5 — Гистограмма распределения длин новостей (без выбросов)

Рассмотрим вторую ось тензора, временную ось. Именно её размерность равна максимальной длине последовательности, а каждая позиция вмещает в себя один временной шаг, или слово. Для каждого наблюдения в пакете она одинаково, но очевидно, что длины непосредственно текстов могут быть отличны: какие-то больше максимального значения, какие-то меньше. Поэтому для унификации мы используем два метода: урезание (сокращение текста до первых T единиц, т.е. слов) и пэддинг (добавление к тексту длины L n специальных токенов пэддинга PAD, чьё векторное представление равно 0 и не учитывается нейросетью, до максимальной длины T: $L + n = T$).

В новостных текстах было 1,347,755 уникальных слов, в заголовках — 221,946. Это слишком большое число для словаря, т.к. он займёт слишком много памяти. Для сравнения, в английском языке обычно используется 20-40 тысяч, но для русского этого будет слишком мало из-за обилия словоформ. Мы подсчитали количество слов, которые встречаются менее 50 раз в новостных текстах (1,251,805) и менее 5 раз в заголовках (159,158). Из этого было решено

использовать 100,000 слов для словаря новостей и 60,000 слов для словаря заголовков; при этом пересечение двух словарей составило примерно 85%.

На рисунках 6 и 7 представлены отношения частотности слов заголовков и новостей к их популярности (рангу). По оси ординат отложено количество появлений слова, по оси абсцисс — его ранг (индекс в словаре, отсортированном от самого частого к наименее частому).

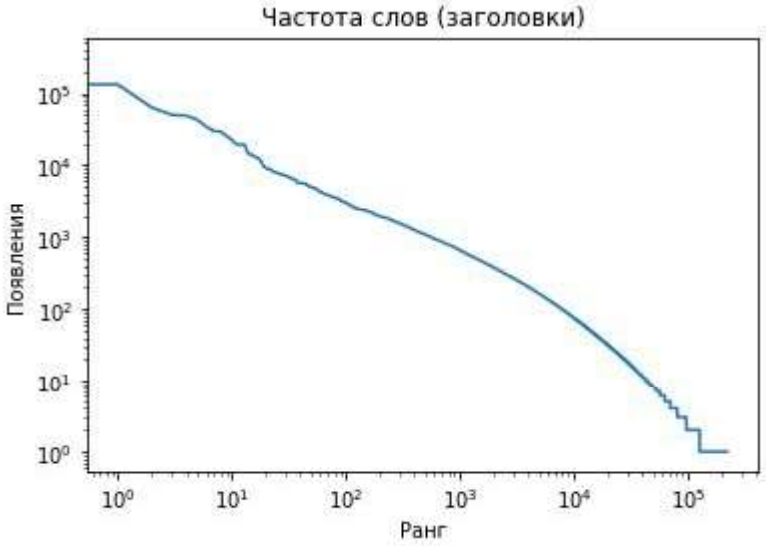

Рисунок 6 — График отношения частотности слов заголовков к их популярности (рангу)

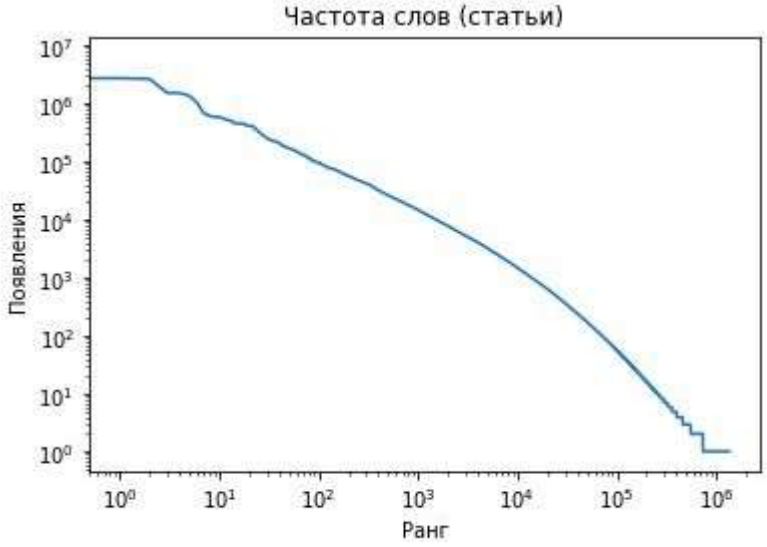

Рисунок 6 — График отношения частотности слов новостей к их популярности (рангу)

Для предварительной обработки текстов и дальнейшего обучения нами были написаны несколько вспомогательных функций и классов. К текстам заголовков добавлялись токены начала и конца последовательности, новости и заголовки приводились к максимальной длине (урезание/пэддинг). Каждая пара новость-заголовок превращалась в три вектора: входные данные кодировщика, в которых каждое слово новости заменялось на соответствующий ему индекс в словаре, а неизвестные слова заменялись специальным токеном неизвестного слова; входные данные декодировщика (т.к. использовался Teacher Forcing, иначе модель использовала собственные ответы в качестве входа), аналогичные входным данным кодировщика, но из заголовка; выходные данные декодировщика, которые представляли собой смещённые на 1 шаг влево ($t + 1$) входные данные декодировщика. Перевод слов в векторное пространство осуществлялось во время работы модели, об этом ниже.

Кодировщик состоит из входного слоя, слоя Embedding, двух LSTM-блоков (слоёв); у него нет выходного слоя, потому что данные с него не учитывались бы, т.к. задача энкодера состоит лишь в векторном представлении “смысла” входного текста). Декодировщик состоит из входного слоя, слоя Embedding, двух LSTM-блоков (слоёв), Dense (полносвязный) слоя и выходного слоя. Входной и выходной слои принимают и выдают данные, LSTM-блоки отвечают за основную работу модели, Embedding слой представляет собой отдельную нейросеть, тренирующуюся совместно с основной для векторизации слов, Dense слов моделирует распределение вероятностей над единицами словаря в результате своей работы, т.е. фактически выбирает слово для генерации. Так как мы используем два рекуррентных блока, у нас в два раза больше скрытых состояний (по паре (т.к. LSTM) с каждого), и состояния первого блока кодировщика мы передаём первому блок декодировщика, а второго — второму. Мы используем последние состояния кодировщика как

начальное состояние первого шага декодировщика и как вектор контекста, подавая на последующие шаги состояние предыдущего шага, а не последнего кодировщика. Размер LSTM-блоков (количество в нём нейронов) равно 128, размерность векторов слов — 100. Всего в модели 24,238,561 параметров. В качестве функции потерь выбрана перекрёстная энтропия (фактически расхождение между “истинным” распределением и моделируемым), оптимизатором стал RMSProp, также используется клиппинг градиента равный 1.

3.4. Результаты работы.

Для тренировки модель использовался графический процессор (GPU) Nvidia Tesla K80: 4992 ядра CUDA, 8.73/2.91 терафлопс (одинарная/двойная точность), 11.5 ГБ памяти GDDR5. Модель тренировалась 6 эпох (полных итераций forward-backpropagation по всем минипакетам) с темпом обучения 0.0033 (гипермараметр, меняющий значения градиента) и размером пакета равным 200 (по 200 наблюдений за итерацию), что заняло 193 минуты на эпоху, всего около 19.5 часов.

Де-факто стандартом оценки генерирующих текст моделей являются ROUGE Score и BLEU Score. Кроме уже упомянутых проблем с ними (формальное совпадение вместо смыслового, предпочтение коротких текстов и др.), существует ещё одна, которая, пожалуй, является даже более серьёзной: неконсистентность метрики. Во время тренировки для оценки модели используется перекрёстная энтропия в качестве функции потерь и точность в качестве метрики, а во время тестирования модели — BLEU или ROUGE, которые не подходят для тренировки, т.к. они не дифференцируемы. Проблема в том, что результат этих метрик не сопоставим напрямую, и модель, хорошо показывающая себя во время тренировки[53]. Вследствие этого мы решили

[53] Shi T. et al. Neural Abstractive Text Summarization with Sequence-to-Sequence Models //arXiv preprint arXiv:1812.02303. – 2018.

использовать для оценки качества модели метрики, используемые на этапе тренировки, а именно энтропию и точность, но применить их только к тренировочной и валидационной выборкам, а тестовые данные оценить вручную методом случайной выборки. Таким образом, к концу финальной эпохи потеря на тренировочной выборке составила 3.70, на валидационной – 4.07 (чем меньше, тем лучше). Точность на тренировочной составила 0.45, на валидационной – 0.43 (чем больше, тем лучше). На рисунке 8 представлен график потерь нейросети во время обучения. По оси ординат отложены значения функции потерь (loss), на оси абсцисс — эпохи; синяя линия (train) — динамиика потерь на тренировочной выборке, оранжевая (val) -валидационной.

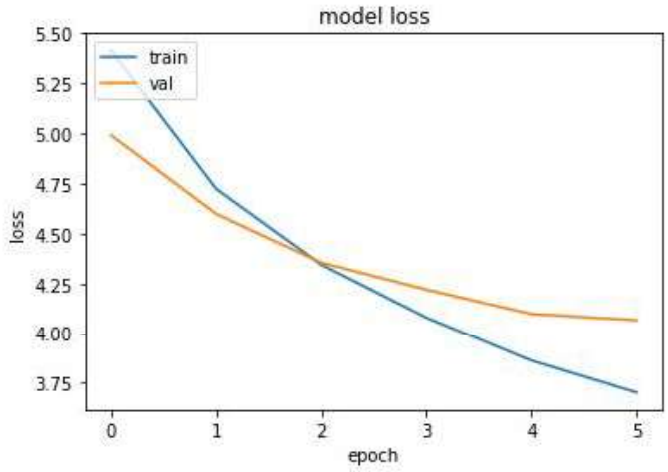

Рисунок 8 — График потерь нейросети во время обучения.

На рисунке 9 представлен график точности нейросети во время обучения. По оси ординат отложены значения основной метрики модели — точности (accuracy), по оси абсцисс — эпохи;синяя линия (train) - динамиика изменения точности на тренировочной выборке, оранжевая (val) — на валидационной.

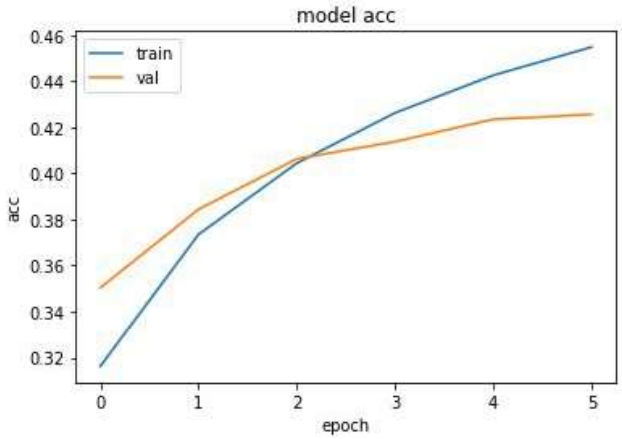

Рисунок 9 — График точности нейросети во время обучения.

В таблице 1 приведены некоторые из сгенерированных моделью заголовков (тексты новостей приведены в сокращённом виде в целях экономии места; также мы убрали спецсимвол конца последовательности, но оставили спецсимвол пэддинга PAD):

Новость	Оригинальный заголовок	Сгенерированный заголовокв
крупнейший производитель микропроцессоров корпорация Intel в среду объявила что приступает к разработке нового микропроцессора под названием pentium 4...	корпорация intel объявила о разработке процессора pentium 4	крупнейший в мире завод
китайские власти могут закрыть один из самых популярных телевизионных конкурсов талантов сообщает new york times...	китайские чиновники объявили конкурсы талантов угрозой для общества	китайцы не смогли договориться
футболист цска сергей игнашевич приписал два гола забитые им в матче чемпионата россии с амкаром своему одноклубнику вагнеру лав...	игнашевич приписал свои голы вагнеру лав	футболист забил PAD PAD гол

чистая прибыль евросети крупнейшего сотового ритейлера россии составила в первом полугодии 2010 года 19 миллиарда рублей сообщает reuters	евросеть перестала приносить убытки	прибыль PAD
палестинские власти разрешили полиции применять оружие для защиты демонстрантов и самообороны на западном берегу реки иордан и в секторе газа...	палестинской полиции разрешили стрелять в израильских солдат	палестинцы PAD PAD PAD PAD взорвали PAD израильский дом
в сша 30 мая был арестован мужчина который входит в десятку самых плодовитых мировых спаммеров сообщает associated press	в сша арестовали одного из самых плодовитых спаммеров	в сша арестован самый большой в мире
немецкие физики создали выключатель из одной молекулы статья исследователей появилась в журнале nature nanotechnology	создан переключатель из одной молекулы	физики создали новую систему
российская теннисистка екатерина макарова вышла в полуфинал турнира в сеуле об этом сообщает официальный сайт соревнований	российская теннисистка вышла в полуфинал турнира в сеуле	российская теннисистка вышла в полуфинал турнира

Таблица 1 — Примеры сгенерированных заголовков

В разделе 3.1. мы установили, что хороший сгенерированный заголовок должен обладать меж- и внутритекстовым единством. Приведённые выше заголовки были отобраны как лучшие, однако проблемы, присутствующие в них, характерны для всех. Как видно из данных примеров, внутритекстовая когезия нарушается редко: предложения связаны по смыслу и грамматически, и только в нескольких примерах из данных (“крупнейший в мире завод” и “в сша арестован самый большой в мире) в предложении нет явного предиката

(“прибыль PAD” может расцениваться как законченная мысль, что-то вроде “прибыль PAD [есть]”). Это говорит о том, что языковая модель декодировщика работает на удовлетворительном уровне. Интересно, что иногда модель выдаёт “пустой” символ PAD, однако он не разрывает ни смысловое, ни грамматическое единство. Тем не менее, нельзя не отметить, что такие предложения хоть и являются связными и даже относительно завершёнными, но они не несут смысла в плане отражения новостного текста, новой информации. Это свидетельствует о недостаточной межтекстовой когеренции. Иногда в сгенерированных заголовках просматривается общий смысл оригинальной новости без конкретики (“физики создали новую систему”), однако чаще нейросеть улавливает место действия, участников и/или инцидент, но не может выделить ключевой “инцидент”, лежащий в основе новости: “китайцы не смогли договориться”, конечно, отражает конфликт китайского народа и китайских властей, но не идёт дальше этого; “футболист забил PAD гол” соответствует действительности, но не подходит в качестве инфоповода.

Дополнительные примеры сгенерированных заголовков находятся в приложении А.

Выводы по главе 3.

В ходе данного исследования мы сделали baseline (базовую) модель для генерации заголовков на основе stacked (многоуровневой) Seq2Seq Encoder-Decoder архитектуры. Мы сознательно не использовали техники и приёмы для повышения её эффективности, т.к. это выходило за рамки наших целей, и в силу ограниченности времени нам пришлось пойти на некоторые компромиссы, например, уменьшить время тренировки (для сравнения, крупные модели тренируются неделями). Тем не менее, на тренировочных данных нейросеть научилась верно предсказывать примерно половину

следующих токенов на основе предыдущих, хотя, конечно, потери остались достаточно большими.

В приложении А можно видеть дополнительные примеры сгенерированных заголовков. Конечно, это отобранные вручную лучшие варианты из просмотренных случайных, потому что далеко не все получились хоть сколько-нибудь пригодными в качестве заголовка даже в самом широком смысле. При генерации нового текста на основе исходного необходимо обращать внимание на семантическое (смысловое) единство между текстами, семантическую и грамматическую когезию внутри целевого. Во всех просмотренных текстах от модели грамматика практически идеальна, но вот со смыслами, к сожалению, не всегда всё хорошо. Внутритекстовая семантическая когезия чаще присутствует, чем отсутствует, но очень часто сгенерированный текст заголовка не отображал истинный смысл статьи, хотя часто улавливал общую тематику.

Заключение.

В данной работе мы предприняли попытку создания базового решения для автоматической генерации новостных заголовков по тексту новости при помощи глубокой рекуррентной нейронной сети. Мы достигли этой цели, но решение не идеальное даже для базовой модели, и требует дальнейшей работы. Впрочем, как proof-of-concept оно вполне работает.

В ходе достижения поставленной цели мы изучили подходящие нам архитектуры РНС, сформулировали на основе анализа отличия в подходах к английскому и русскому языку и соответствующие модификации сетей, рассмотрели различные приёмы и техники для улучшения модели, собрали данные для тренировки, обработали их, создали небольшие мини-модели для проверки работоспособности, подобрали гиперпараметры и обучили многоуровневую Seq2Seq Encoder-Decoder модель. По результатам нашей теоретической и практической работы мы можем сделать вывод, что выбранная нами архитектура подходит для решения задачи автоматической генерации новостных заголовков из текста новости на русском языке, однако требует некоторых изменений.

Как было сказано в выводах к третьей главе, в нашей базовой модели присутствуют локальные семантическая и грамматическая когезия (хотя не без проблем), но главный провал заключается в слабой способности к “пониманию” источника и обращению к его содержанию. Как можно решить эти проблемы?

Межтекстовая связь возможно легко решается введением механизма внимания, который позволяет точнее генерировать каждое слово, обращаясь непосредственно к словам источника. Может быть, нейросеть не смогла полностью научится кодировать смысл, и ей нужно больше времени; однако скорее всего у неё уже началось переобучение, поэтому просто увеличивать количество эпох не имеет смысла, необходимо либо расширить выборку (в

больших данных это почти всегда плюс), либо увеличить сеть (например, добавить ещё слоёв), либо ввести дропаут, а лучше рекуррентный дропаут. Также проблема может заключаться в излишней генерализации, когда сеть не способна конкретизировать данные источника в силу ограниченности словаря, ведь для русского языка нужен действительно огромный словарь. Тут могут помощь механизм копирования, трюки для работы со словарём или Byte-Pair Encoding (или иной алгоритм векторизации слов, например, FastText, который также способен работать с подсловесными единицами).

Эти и другие методы могут улучшить работу сети, но они её очень сильно увеличат, что скажется на её потреблении памяти и требованиям к вычислительным ресурсам (включая время). Возможно, это повлечёт за собой иные компромиссы, как, например, использование предварительно обученных векторов слов (при наличии достаточно большого корпуса имеет смысл проводить собственное обучение).

Кроме того, можно попробовать совместить нейросеть и ручные правила для, например, выделения именованных сущностей. Часто такие комбинированные методы показывают себя лучше, чем по отдельности, хотя требуют дополнительных усилий, особенно в процессе написания правил.

Таким образом, перед нами есть огромное поле вариантов для дальнейшей работы. Нам представляется логичным обогатить данную базовую модель минимумом приёмов (внимание и лучевой поиск), а затем создавать новые, более мощные модели. Добавление новых рекуррентных блоков увеличивает время рассчёта одной эпохи, но даёт модели дополнительные мощности, фактически увеличивая максимальную эффективность, поэтому найти оптимальное количество блоков (максимизирующих соотношение время работы/качество) также представляется логичным продолжением данной работы.

Данное исследование выполнено в рамках проекта “World2News: создание веб-сервиса по автоматическому генерированию новостей” Лаборатории Когнитивных исследований языка ТГУ. Проект выполнен на средства Гранта ТГУ в рамках Программы государственной поддержки ведущих университетов Российской Федерации в целях повышения их конкурентоспособности среди ведущих мировых научно-образовательных центров.

Результаты исследования апробированы на конференциях:

1. VI (XX) Международная научно-практическая конференции молодых учёных «Актуальные проблемы лингвистики и литературоведения», 18–20 апреля 2019 г.;
2. 25-я Международная конференция по компьютерной лингвистике и интеллектуальным технологиям “Диалог-2019”.

Результаты исследования представлены в статьях:

1. Здоровец А.И. Генерация заголовков с помощью многоуровневой Seq2Seq модели / Актуальные проблемы лингвистики и литературоведения. Сборник материалов VI (XX) Международной конференции молодых ученых (18–20 апреля 2019 г.). Выпуск 20. Изд-во ТГУ, 2019. (статья принята к печати в 2019 году).
2. Shevchuk A.A., Zdorovets A.I. Text Summarization with Recurrent Neural Network for Headline Generation [Электронный ресурс] // Сборник материалов конференции “Диалог-2019” (28 мая – 02 мая 2019). Студенческая сессия. URL: http://www.dialog-21.ru/digest/2019/student/ (дата обращения 10.06.2019).

Список использованных источников и литературы.

1. Goodfellow I., Bengio Y., Courville A. Deep learning. – MIT press, 2016.
2. Nielsen M. A. Neural networks and deep learning. – San Francisco, CA, USA: : Determination press, 2015. – Т. 25.
3. Haykin S. S. et al. Neural networks and learning machines. – Upper Saddle River : Pearson education, 2009. – Т. 3.
4. Kirk M. Thoughtful Machine Learning with Python: A Test-driven Approach. – " O'Reilly Media, Inc.", 2017.
5. Шолле Ф. Глубокое обучение на Python //СПб.: Питер. – 2018.
6. Khodak M. et al. A la carte embedding: Cheap but effective induction of semantic feature vectors //arXiv preprint arXiv:1805.05388. – 2018.
7. Williams R. J., Zipser D. A learning algorithm for continually running fully recurrent neural networks //Neural computation. – 1989. – Т. 1. – №. 2. – С. 270-280.
8. Khatri C., Singh G., Parikh N. Abstractive and Extractive Text Summarization using Document Context Vector and Recurrent Neural Networks //arXiv preprint arXiv:1807.08000. – 2018.
9. Nallapati R. et al. Abstractive text summarization using sequence-to-sequence rnns and beyond //arXiv preprint arXiv:1602.06023. – 2016.
10. Zeiler M. D. ADADELTA: an adaptive learning rate method //arXiv preprint arXiv:1212.5701. – 2012.
11. Kingma D. P., Ba J. Adam: A method for stochastic optimization //arXiv preprint arXiv:1412.6980. – 2014.

12. Duchi J., Hazan E., Singer Y. Adaptive subgradient methods for online learning and stochastic optimization //Journal of Machine Learning Research. – 2011. – Т. 12. – №. Jul. – С. 2121-2159.
13. Bengio Y., Boulanger-Lewandowski N., Pascanu R. Advances in optimizing recurrent networks //2013 IEEE International Conference on Acoustics, Speech and Signal Processing. – IEEE, 2013. – С. 8624-8628.
14. Jozefowicz R., Zaremba W., Sutskever I. An empirical exploration of recurrent network architectures //International Conference on Machine Learning. – 2015. – С. 2342-2350.
15. Ruder S. An overview of gradient descent optimization algorithms //arXiv preprint arXiv:1609.04747. – 2016.
16. Gal Y., Ghahramani Z. A theoretically grounded application of dropout in recurrent neural networks //Advances in neural information processing systems. – 2016. – С. 1019-1027.
17. Olah C., Carter S. Attention and augmented recurrent neural networks //Distill. – 2016. – Т. 1. – №. 9. – С. e1.
18. Vaswani A. et al. Attention is all you need //Advances in neural information processing systems. – 2017. – С. 5998-6008.
19. Freitag M., Al-Onaizan Y. Beam search strategies for neural machine translation //arXiv preprint arXiv:1702.01806. – 2017.
20. Papineni K. et al. BLEU: a method for automatic evaluation of machine translation //Proceedings of the 40th annual meeting on association for computational linguistics. – Association for Computational Linguistics, 2002. – С. 311-318.
21. Ordóñez F., Roggen D. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition //Sensors. – 2016. – Т. 16. – №. 1. – С. 115.

22. Li P. et al. Deep recurrent generative decoder for abstractive text summarization //arXiv preprint arXiv:1708.00625. – 2017.
23. Mikolov T. et al. Distributed representations of words and phrases and their compositionality //Advances in neural information processing systems. – 2013. – С. 3111-3119.
24. Axelrod A., He X., Gao J. Domain adaptation via pseudo in-domain data selection //Proceedings of the conference on empirical methods in natural language processing. – Association for Computational Linguistics, 2011. – С. 355-362.
25. Srivastava N. et al. Dropout: a simple way to prevent neural networks from overfitting //The Journal of Machine Learning Research. – 2014. – Т. 15. – №. 1. – С. 1929-1958.
26. Luong M. T., Pham H., Manning C. D. Effective approaches to attention-based neural machine translation //arXiv preprint arXiv:1508.04025. – 2015.
27. Chung J. et al. Empirical evaluation of gated recurrent neural networks on sequence modeling //arXiv preprint arXiv:1412.3555. – 2014.
28. Sukhbaatar S. et al. End-to-end memory networks //Advances in neural information processing systems. – 2015. – С. 2440-2448.
29. J.W.G. Putra, H. Kobayashi, N. Shimizu. Experiment on Using Topic Sentence for Neural News Headline Generation //Proceedings of the 24th Annual Meeting of the Speech Processing Society of Japan (March 2018). — 2018.
30. Mikolov T. et al. Extensions of recurrent neural network language model //2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). – IEEE, 2011. – С. 5528-5531.

31. Chen Y. C., Bansal M. Fast abstractive summarization with reinforce-selected sentence rewriting //arXiv preprint arXiv:1805.11080. – 2018.
32. Sherstinsky A. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network //arXiv preprint arXiv:1808.03314. – 2018.
33. Lopyrev K. Generating news headlines with recurrent neural networks //arXiv preprint arXiv:1512.01712. – 2015.
34. Graves A. Generating sequences with recurrent neural networks //arXiv preprint arXiv:1308.0850. – 2013.
35. See A., Liu P. J., Manning C. D. Get to the point: Summarization with pointer-generator networks //arXiv preprint arXiv:1704.04368. – 2017.
36. Pennington J., Socher R., Manning C. Glove: Global vectors for word representation //Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). – 2014. – С. 1532-1543.
37. Wu Y. et al. Google's neural machine translation system: Bridging the gap between human and machine translation //arXiv preprint arXiv:1609.08144. – 2016.
38. Gu J. et al. Incorporating copying mechanism in sequence-to-sequence learning //arXiv preprint arXiv:1603.06393. – 2016.
39. Cho K. et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation //arXiv preprint arXiv:1406.1078. – 2014.
40. Martens J., Sutskever I. Learning recurrent neural networks with hessian-free optimization //Proceedings of the 28th International

Conference on Machine Learning (ICML-11). – 2011. – С. 1033-1040.

41. Hochreiter S., Schmidhuber J. Long short-term memory //Neural computation. – 1997. – Т. 9. – №. 8. – С. 1735-1780.
42. Tu Z. et al. Modeling coverage for neural machine translation //arXiv preprint arXiv:1601.04811. – 2016.
43. Chiu J. P. C., Nichols E. Named entity recognition with bidirectional LSTM-CNNs //Transactions of the Association for Computational Linguistics. – 2016. – Т. 4. – С. 357-370.
44. Shi T. et al. Neural Abstractive Text Summarization with Sequence-to-Sequence Models //arXiv preprint arXiv:1812.02303. – 2018.
45. Shen S. et al. Neural headline generation with sentence-wise optimization //arXiv preprint arXiv:1604.01904. – 2016.
46. Bahdanau D., Cho K., Bengio Y. Neural machine translation by jointly learning to align and translate //arXiv preprint arXiv:1409.0473. – 2014.
47. Sennrich R., Haddow B., Birch A. Neural machine translation of rare words with subword units //arXiv preprint arXiv:1508.07909. – 2015.
48. Pascanu R., Mikolov T., Bengio Y. On the difficulty of training recurrent neural networks //International conference on machine learning. – 2013. – С. 1310-1318.
49. Cho K. et al. On the properties of neural machine translation: Encoder-decoder approaches //arXiv preprint arXiv:1409.1259. – 2014.
50. Jean S. et al. On using very large target vocabulary for neural machine translation //arXiv preprint arXiv:1412.2007. – 2014.

51. Goldsmith J. Probability for linguists //Mathématiques et sciences humaines. Mathematics and social sciences. – 2007. – №. 180. – С. 73-98.
52. Bergstra J., Bengio Y. Random search for hyper-parameter optimization //Journal of Machine Learning Research. – 2012. – Т. 13. – №. Feb. – С. 281-305.
53. Lin C. Y. Rouge: A package for automatic evaluation of summaries //Text Summarization Branches Out. – 2004.
54. Gavrilov D., Kalaidin P., Malykh V. Self-Attentive Model for Headline Generation //arXiv preprint arXiv:1901.07786. – 2019.
55. Ranzato M. A. et al. Sequence level training with recurrent neural networks //arXiv preprint arXiv:1511.06732. – 2015.
56. Sutskever I., Vinyals O., Le Q. V. Sequence to sequence learning with neural networks //Advances in neural information processing systems. – 2014. – С. 3104-3112.
57. Graves A., Mohamed A., Hinton G. Speech recognition with deep recurrent neural networks //2013 IEEE international conference on acoustics, speech and signal processing. – IEEE, 2013. – С. 6645-6649.
58. Lambert J. Stacked RNNs for Encoder-Decoder Networks: Accurate Machine Understanding of Images //URL https://cs224d. stanford. edu/reports/Lambert. pdf.
59. Chen W., Grangier D., Auli M. Strategies for training large vocabulary neural language models //arXiv preprint arXiv:1512.04906. – 2015.
60. Hermans M., Schrauwen B. Training and analysing deep recurrent neural networks //Advances in neural information processing systems. – 2013. – С. 190-198.

61. Doersch C. Tutorial on variational autoencoders //arXiv preprint arXiv:1606.05908. – 2016.
62. Pascanu R., Mikolov T., Bengio Y. Understanding the exploding gradient problem //CoRR, abs/1211.5063. – 2012. – Т. 2.
63. Press O., Wolf L. Using the output embedding to improve language models //arXiv preprint arXiv:1608.05859. – 2016.
64. Maaten L., Hinton G. Visualizing data using t-SNE //Journal of machine learning research. – 2008. – Т. 9. – №. Nov. – С. 2579-2605.
65. A ten-minute introduction to sequence-to-sequence learning in Keras [Электронный ресурс] // Keras Docs. URL: https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html (дата обращения 08.06.2019). — 2017.
66. Deepfake videos: Inside the Pentagon's race against disinformation // CNN. URL: https://edition.cnn.com/interactive/2019/01/business/pentagons-race-against-deepfakes/ (дата обращения 10.06.2019). — 2019.
67. Dive into Deep Learning. [Электронный ресурс] // Zhang A. et al. — URL: http://www.d2l.ai (дата обращения 04.06.2019). — 2019.
68. Convolutional Neural Networks (CNNs / ConvNets) : CS231n Convolutional Neural Networks for Visual Recognition [Электронный ресурс] // Stanford University. — URL: http://cs231n.github.io/convolutional-networks (дата обращения 30.04.2019). — 2019.
69. Grid Search for model tuning [Электронный ресурс] // Rohan Joseph. — URL: https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e (дата обращения 06.05.2019). — 2018.

70. Improve Shallow Neural Network Generalization and Avoid Overfitting // Mathworks. — URL: https://www.mathworks.com/help/deeplearning/ug/improve-neural-network-generalization-and-avoid-overfitting.html;jsessionid=825d3112487ac09f537fb633dbe5 (дата обращения 05.06.2019). — 2019.
71. Machine learning: Generative and discriminative models [Электронный ресурс] // Srihari S. Machine Learning Course. URL: http://www.cedar.buffalo.edu/~srihari/CSE574/Discriminative-Generative.pdf (дата обращения 05.06.2019). – 2010.
72. Neural Machine Translation (seq2seq) Tutorial [Электронный ресурс] / Luong M.T., Brevdo E., Zhao R. — URL: https://github.com/tensorflow/nmt (дата обращения 25.05.2019). — 2017.
73. News dataset from Lenta.Ru Corpus of Russian news articles collected from Lenta.Ru [Электронный ресурс] // Dmitry Utkin at Kaggle. URL: https://www.kaggle.com/yutkin/corpus-of-russian-news-articles-from-lenta (дата обращения 14.02.2019). — 2019.
74. The Bitter Lesson [Электронный ресурс] // Rich Sutton. URL: http://www.incompleteideas.net/IncIdeas/BitterLesson.html (дата обращения 05.06.2019). — 2019.
75. Translation with a Sequence to Sequence Network and Attention [Электронный ресурс] // Pytorch Docs. URL: https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html (дата обращения 08.06.2019). — 2017.
76. Вычислительная фотография : Будущее фотографии — это код [Электронный ресурс] // Vas3k URL:

https://vas3k.ru/blog/computational_photography/ (дата обращения 04.06.2019). — 2019

77. A friendly introduction to Recurrent Neural Networks [Электронный ресурс] // Luis Serrano. URL: https://www.youtube.com/watch?v=UNmqTiOnRfg (дата обращения 14.02.2019). — 2017.
78. A Short Introduction to Entropy, Cross-Entropy and KL-Divergence [Электронный ресурс] // Aurélien Géron. URL: https://www.youtube.com/watch?v=ErfnhcEV1O8 (дата обращения 09.04.2019). — 2018.
79. Future (Present?) of Machine Translation [Электронный ресурс] // Microsoft Research. URL: https://www.youtube.com/watch?v=z4CNmiLF-YU (дата обращения 15.02.2019). — 2016.
80. Stanford CS224N : NLP with Deep Learning | Winter 2019 | Lecture 3 – Neural Networks [Электронный ресурс] // Stanford University. URL: https://www.youtube.com/watch?v=8CWyBNX6eDo&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=3 (дата обращения 20.02.2019). — 2019.
81. Stanford CS224N: NLP with Deep Learning | Winter 2019 | Lecture 8 – Translation, Seq2Seq, Attention [Электронный ресурс] // Stanford University. URL: https://www.youtube.com/watch?v=XXtpJxZBa2c (дата обращения 20.02.2019). — 2019.
82. Stanford CS224N: NLP with Deep Learning | Winter 2019 | Lecture 9 – Practical Tips for Projects [Электронный ресурс] // Stanford University. URL: https://www.youtube.com/watch?v=fyqm8fRDgl0 (дата обращения 21.02.2019). — 2019.

83. Stanford Seminar - Can the brain do back-propagation? [Электронный ресурс] // Stanford University. URL: https://www.youtube.com/watch?v=VIRCybGgHts (дата обращения 03.06.2019). — 2016.
84. Stanford Seminar - Information Theory of Deep Learning [Электронный ресурс] // Stanford University. URL: https://www.youtube.com/watch?v=XL07WEc2TRI (дата обращения 09.04.2019). — 2018.

Приложение А.

михаил михаил потерял исторический максимум
барак обама пообещал заменить его PAD PAD PAD PAD PAD PAD роналду
чемпион мира по футболу владимир путин завершил карьеру
украина осталась без россии
эксперты оценили шансы света
талибы захватили PAD ураган
минобороны не PAD PAD смогли
в сша террористы не смогли договориться
грузинские солдаты попали в интернет
на мкс произошла перестрелка
саакашвили решил стать президентом грузии
власти канады выпустит памятник сталину
израиль обвинил израиль PAD в подготовке покушения
в москве милиция разогнала митинг для военных войск
в италии похоронили PAD PAD PAD кладбище
сын саркози назвал себя лучшим фильмом для своей песни
организатор теракта 11 сентября PAD отложили
в ингушетии взорвали машину бывшего министра

charlie hebdo PAD опубликовал PAD следствие
криштиану роналду решил судиться из-за его смерти
новый год будет бороться с аль-каедой
британский посол отказался играть в фильме про гитлера
в PAD PAD москве обстреляли машину бизнесмена
белый дом белого дома
в санкт-петербурге произошел PAD пожар
бывший министр PAD обороны израиля PAD PAD получил восемь лет за шпионаж в пользу израиля
в казахстане поставят памятник

Exactus Like (/index.php)

(/index.php/ru/) (/index.php/en/)

Поиск заимствований в научных текстах$^{\beta}$

Введите текст:

...или загрузите файл:

Файл не выбран... **Выбрать файл...**

Укажите год публикации: 2019

Выберите коллекции

- **Все**
- **Рефераты** **Википедия** **Российские журналы**
- **Авторефераты** **Российские конференции** **Энциклопедии**
- **Иностранные конференции** **Иностранные журналы** **Англоязычная википедия**
- **PubMed**

Анализировать

Проверить по расширенному списку коллекций системы Руконтекст (http://text.rucont.ru/like)

Обработан файл:
ВКР Здоровец.pdf.

Год публикации: 2019.

Оценка оригинальности документа - 100.0%

Процент условно корректных заимствований - 0.0%

Процент некорректных заимствований - 0.0%

Просмотр заимствований в документе

Время выполнения: 47 с.

Заимствования отсутствуют

Дополнительно

Общеизвестные фрагменты

Значимые оригинальные фрагменты

Библиографические ссылки

Искать в Интернете