САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра технологии программирования

Шенбин Илья Игоревич

Магистерская диссертация

Обучение распределенных представлений слов на основе символов

Направление 02.04.02

Фундаментальные информатика и информационные технологии

Магистерская программа: Технологии баз данных

Научный руководитель,

кандидат физ.-мат. наук,

доцент

Добрынин В. Ю.

Санкт-Петербург

2017 г.

Содержание

1 Введение

В 2013 году Томашем Миколовым была представлена статья “Efficient Estimation of Word Representations in Vector Space” [1] в которой были описаны две модели для обучения так называемых распределённых представлений слов: continuous bag-of-words и continuous skip-gram, программная реализация которых широко известна как word2vec. Распределённое представление слова — это вектор из $\mathbb{R}^n$ (при n значимо меньшем, чем размер словаря обучающей выборки). Основная идея распределённых представлений заключается в том, что семантически близкие слова будут иметь схожие представления.

После публикации этой статьи word2vec стал крайне популярным в задачах автоматической обработки естественного языка и остаётся таким по сей день. Но идея, используемая в статье, не была совершенно новой, ещё в 2003 году в статье “A neural probabilistic language model” [2] была представлена похожая модель, однако она уступала как в качестве обучаемых представлений, так и в плане вычислительной эффективности. Так же стоит отметить модели тематического моделирования, например, latent dirichlet allocation [3], главной отличительной чертой которых являлось то, что при обучении в качестве контекста для слова эти модели используют весь документ целиком, а не только небольшое число ближайших соседей.

Большинство популярных методов для обучения представлений слов рассматривают слова атомарно, то есть никак не учитывают информацию ни о символах, образующих слово, ни о морфологии. Например, два слова, различающиеся только окончаниями, будут восприняты моделью как разные слова. Такой подход эффективен с вычислительной точки зрения. Однако в случае с морфологически богатыми языками (к которым, напри-

мер, относятся многие славянские и романские языки) могут появляться трудности: некоторые формы слов могут ни разу не встретиться даже в очень большой обучающей выборке. Ещё одной, но уже меньшей проблемой такого подхода является неустойчивость к опечаткам.

Самым простым способом решения проблемы с морфологией является приведение всех слов к начальной форме. С таким подходом имеется две проблемы: во-первых, потребуется использовать морфологический анализатор для нормирования слов, во-вторых, в некоторых задачах приведение к начальной форме недопустимо.

Другой подход, набирающий популярность в последние годы — нейронные сети, принимающие в качестве входа слова как последовательность букв. Недостатком такого подхода является необходимость в большем размере обучающей выборки (как правило, размеченной), а так же в больших вычислительных ресурсах.

Отдельно стоит отметить char2vec [4], являющийся обобщением word2vec и тесно связанный с моделями, предложенными в этой работе, но так же имеющий недостатки, описанные в параграфе выше.

2 Постановка задачи

Данная работа посвящена построению модели, которая будет обучаться на парах (слово, представление слова) и по поданному на вход слову в посимвольном виде она будет прогнозировать соответствующее представление. Говоря более формально, вход модели — последовательность произвольной длины (слово), состоящая из векторов, которые некоторым образом кодируют буквы, а выход модели — вектор из $\mathbb{R}^n$.

Обучающую выборку в поставленной задаче можно получить, обучив представления слов на некотором корпусе текстов. Обученные представления на различных корпусах также можно найти в открытом доступе, например, на PubMed и PMC[1] или на ряде русскоязычных[2].

Следует отметить, что в данной работе не подразумевается использование размеченных данных какого-либо другого рода, экспертных знаний о языке (например, для ручного создания признаков), так же как и инструментов, которые для обработки текстов используют экспертные знания о языке (например, морфологические анализаторы). Причина этому — желание построить модель, которая будет хорошо работать вне зависимости от того, для какого языка она используется.

Кроме вышеперечисленного, в данной работе будут предложены метрики для интерпретируемой оценки качества модели, а так же применение предложенного решения к задаче распознавания именованных сущностей.

[1] http://bio.nlplab.org/

[2] http://ling.go.mail.ru/ru/

3 Обзор литературы

В этом разделе описаны базовые методы, использованные в работе.

3.1 Нейронные сети прямого распространения

Последние пять лет исследований в области нейронных сетей прошли чрезвычайно успешно. Нейронные сети превзошли по качеству остальные модели машинного обучения во многих задачах (классификация изображений, распознавание речи, машинный перевод и в других).

Нейронная сеть прямого распространения [19] (FNN) является наиболее базовой из архитектур нейронных сетей. Она состоит из последовательно соединённых слоёв, среди которых выделяются входной слой (внизу на рисунке 1), выходной слой (вверху) и скрытые слои.

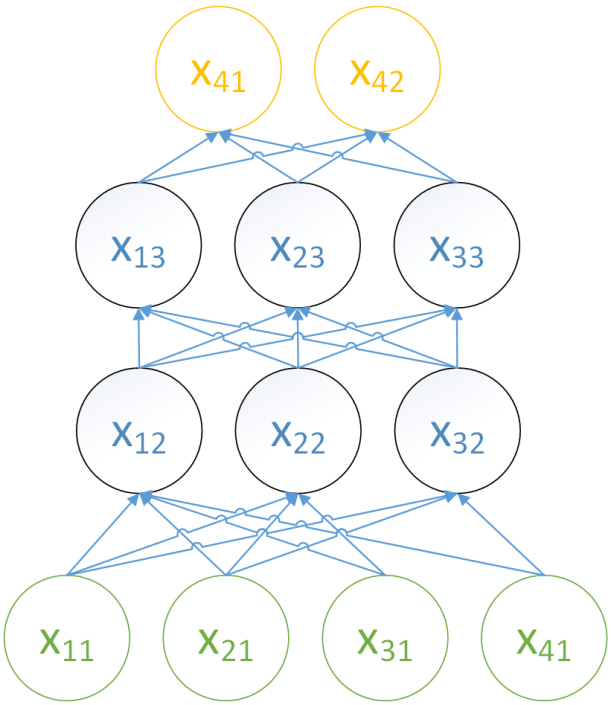

Рис. 1: Нейронная сеть прямого распространения

Каждый слой содержит в себе набор нейронов. Выход i-го нейрона j-го слоя ($j > 1$, если входной слой считать первым) определяется следующим образом:

$$x_{ij} = f_j\left(\sum_k w_{ijk} x_{k,j-1}\right)$$

где w_{ijk} – коэффициенты линейной комбинации, а $f_j(x)$ – функция активации, например, сигмоид:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

В дальнейшем будем использовать матричное представление, в соответствии с которым выход j-го слоя будет равен

$$\mathbf{x}_j = f_j(W_j \mathbf{x}_{j-1})$$

В качестве функции активации предлагается выбирать любую нелинейную функцию, но на практике из-за некоторых функций активаций (например, сигмоид), может быстро затухать градиент в глубоких сетях. Бороться с этим можно, например, используя функции вроде ReLU [22] или её модификаций.

Для обучения сети, то есть для поиска параметров W_j, будем сравнивать отклонение значений выходного слоя $\mathbf{x}_n$ на некотором входе $\mathbf{x}_1$ от ожидаемых значений $\mathbf{z}$. Для этого введём функцию потерь $E(\mathbf{x}_n, \mathbf{z})$ и минимизируем её по параметрам W_j для $j \in [2, n]$, а градиенты вычислим при помощи алгоритма обратного распространения [6], заключающийся в применении правила дифференцирования сложной функции.

Для поиска минимума обычно используется либо стохастический градиентный спуск [6], либо его модификации вроде RMSProp[7] или Adam[8],

которые позволяют регулировать темп обучения (learning rate) покоординатно.

3.2 Рекуррентные нейронные сети

При обучении сети прямого распространения неявно предполагается, что элементы обучающей выборки являются взаимно независимыми. Следовательно, такая модель не подходит для работы с данными, у которых есть, например, временная зависимость. В связи с этим были предложены сети Элмана [5], которые впоследствии стали известны как рекуррентные нейронные сети (RNN) [11].

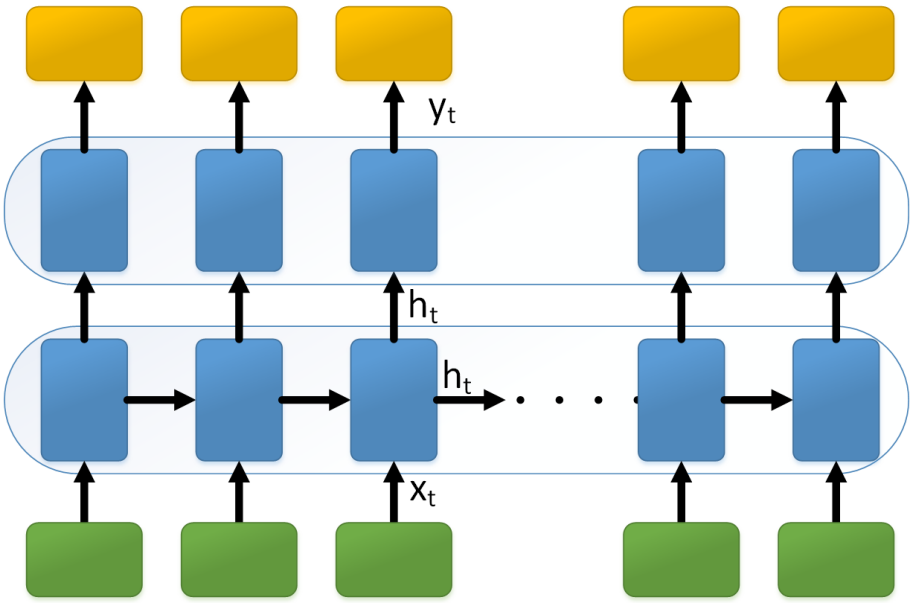

Рис. 2: Рекуррентная нейронная сеть

Будем рассматривать сеть с одним скрытым слоем. Отличие такой сети от сети прямого распространения заключается в добавлении к входам скрытого слоя выходов из скрытого слоя в предыдущий момент времени. Обозначив за $\mathbf{x}_t$, $\mathbf{h}_t$ и $\mathbf{y}_t$ выходы входного, скрытого и выходного слоёв

соответственно в момент времени t, определим рекуррентную нейронную сеть:

$$\mathbf{h}_t = f(U\mathbf{x}_t + W\mathbf{h}_{t-1})$$

$$\mathbf{y}_t = f(V\mathbf{h}_t)$$

3.3 Проблема уменьшающихся/взрывающихся градиентов

На практике при обучении стандартных рекуррентных нейронных сетей могут возникать проблемы, связанные с экспоненциальным затуханием или ростом градиента. Они были подробно изучены в [9].

Из-за первой проблемы такие сети не способны обучиться долговременным зависимостям. Для её решения предлагается использовать LSTM сети, которые описаны ниже.

Для решения второй проблемы в [9] предлагалось нормировать градиент, если его норма превосходит некоторую заранее заданную величину.

uRNN — ещё один подход, избавляющий от указанных проблем, идея которого заключается в том, что при любых параметрах матрица перехода будет унитарной. uRNN работает почти так же хорошо, как LSTM, но содержит меньше параметров [12].

3.4 Long short-term memory

LSTM юниты [10][11] основаны на системе гейтов, благодаря которым становится возможным хранить информацию в юните сколь угодно долго.

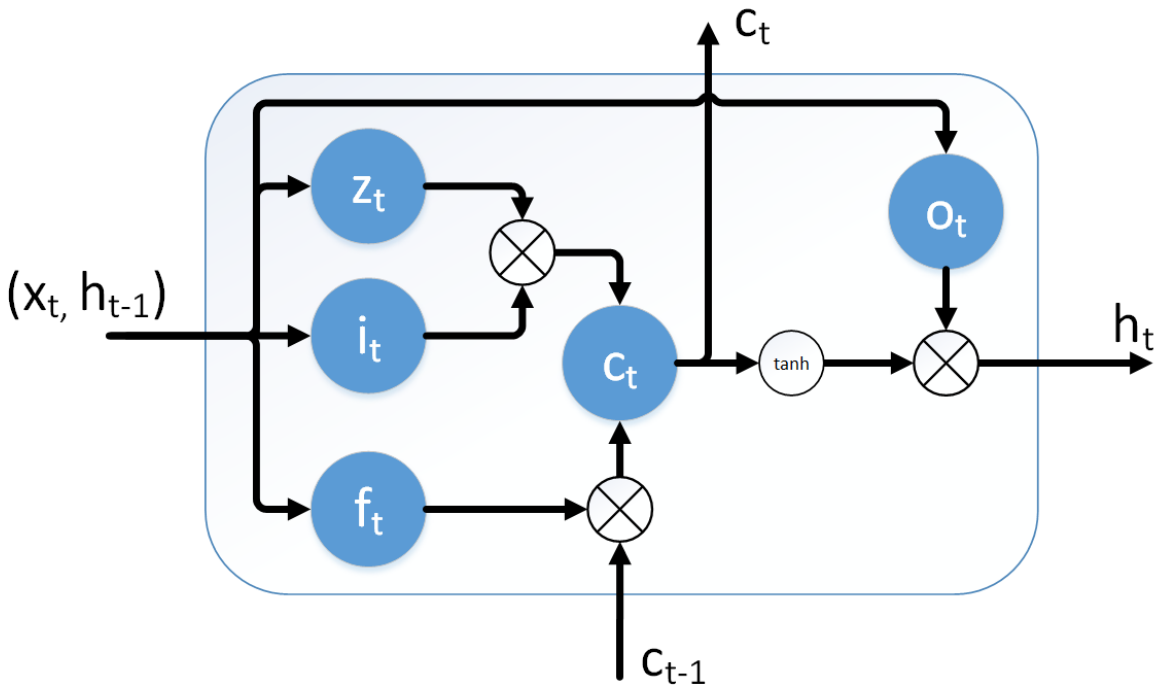

Рис. 3: Long short-term memory юнит

Рассмотрим формулы, описывающие LSTM юнит. Обозначения остаются те же, как в случае с рекуррентными нейронными сетями, но появляется ещё одно значение $\mathbf{c}_t$ – внутренняя память нейрона в момент t.

$$\mathbf{i}_t = \sigma(U_i \mathbf{x}_t + W_i \mathbf{h}_{t-1} + b_i)$$

$$\mathbf{f}_t = \sigma(U_f \mathbf{x}_t + W_f \mathbf{h}_{t-1} + b_f)$$

$$\mathbf{o}_t = \sigma(U_o \mathbf{x}_t + W_o \mathbf{h}_{t-1} + b_o)$$

$$\mathbf{z}_t = \tanh(U_z \mathbf{x}_t + W_z \mathbf{h}_{t-1} + b_z)$$

$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t + \mathbf{z}_t \odot \mathbf{i}_t$$

$$\mathbf{h}_t = \tanh(\mathbf{c}_t) \odot \mathbf{o}_t$$

где $\odot$ – поэлементное произведение.

Функцией активации для входного $\mathbf{i}_t$, выходного $\mathbf{o}_t$ гейтов и гейта забывания $\mathbf{f}_t$ является сигмоид, то есть область значений этих гейтов есть отрезок $[0; 1]$. Можно сказать, что эти гейты обучаются тому, насколько следует открыть вход, выход нейрона и насколько следует стереть внутреннюю память в зависимости от входа в текущий момент времени и состояния в предыдущий. $\mathbf{z}_t$ есть предварительное состояние выходного сигнала до того как он будет ослаблен входным гейтом и слит с внутренней памятью, а затем ослаблен выходным гейтом.

3.5 Свёрточные сети

Свёрточные сети впервые были предложены Яном Лекуном [13], на данный момент они широко известны по задачам компьютерного зрения. Однако они нашли применение и в анализе естественного языка [14]. В последнем случае используются не двухмерные, а одномерные свёрточные слои, именно они будут рассмотрены ниже.

Свёрточный слой точно так же, как и рекуррентный, может использоваться в случаях, когда размерность входа не является постоянной, но отличается тем, что в каждый момент времени выход со свёрточного слоя зависит только от соответствующей подпоследовательности входов фиксированной длины.

Пусть входом свёрточного слоя будет $C = [c_1, \ldots, c_l] \in \mathbb{R}^{d \times l}$ — последовательность длины l, состоящая из векторов размерности d. Ядром свёртки будет называться матрица $H \in \mathbb{R}^{d \times w}$, где w — ширина свёртки. Тогда выходом свёрточного слоя в момент времени i будет:

$$f_i = Tr(C_{*,i:i+w-1} H^\top) + b$$

где b — свободный член, а $C_{*,i:i+w-1}$ — подматрица C, содержащая только строки с i по $i + w - 1$.

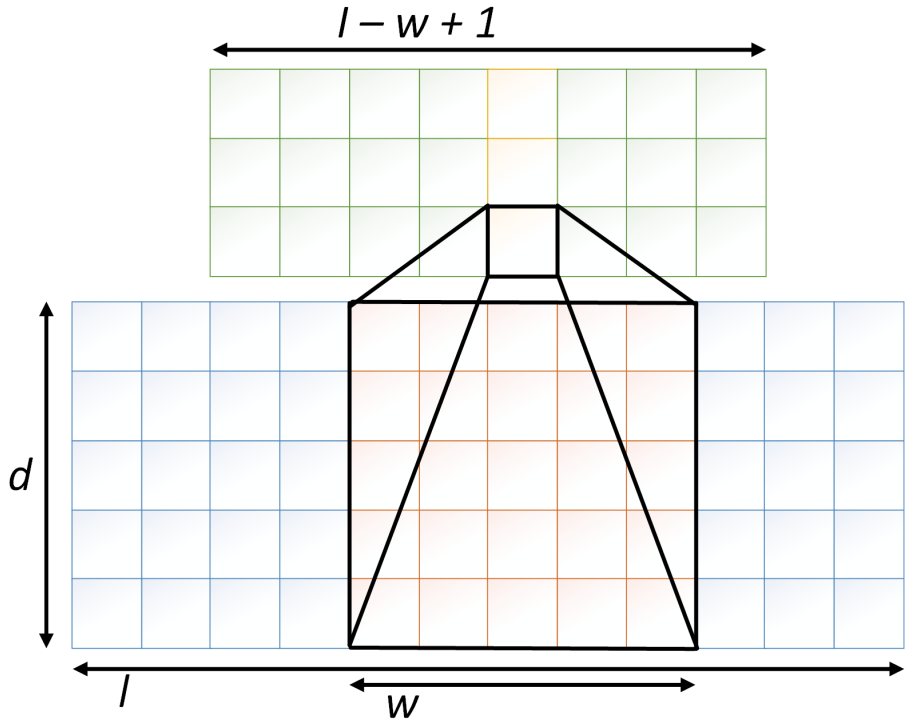

Рис. 4: Свёрточный слой

Можно заметить, что размерность выхода зависит от размерности входа и тоже будет переменной. В некоторых случаях требуется перейти к фиксированной размерности, для этого можно предлагается использовать операцию global maxpooling, которая возвращает максимальное значение, перебрав все моменты времени:

$$y = \max_i f_i$$

Обычно для каждого слоя обучается несколько ядер свёртки, возможно, с различной шириной. Тогда размерность выхода будет равна числу ядер свёрток (при условии использования global maxpooling).

3.6 Dropout

Для борьбы с переобучением нейронных сетей одним из самых популярных подходов является dropout [15]. Идея метода заключается в том, что в процессе обучения (и только обучения) на выходе некоторого слоя случайным образом обнуляются некоторые его компоненты.

Если x_i — i-ый выход слоя, после которого установлен dropout, то i-ым выходом dropout-слоя будет

$$y_i = r_i x_i$$

где

$$r_i \sim Bernoulli(p)$$

и где p — параметр слоя, вероятность обнуления. На этапе тестирования такая процедура не производится, но выходы слоя, за которым следует dropout домножаются на p что бы как на этапе тестирования, так и на этапе обучения средние значения выходов были одинаковыми.

Эксперименты показали, что такой подход при использовании на рекурентный связях отрицательно сказывается на качестве. Но в статье [16] было показано, что для этого случая требуется разыгрывать r_i единожды и применять его между всеми моментами времени.

3.7 word2vec

Рассмотрим метод для обучения распределённых представлений слов — continuous skip-gram [1][17], известный под названием своей программной

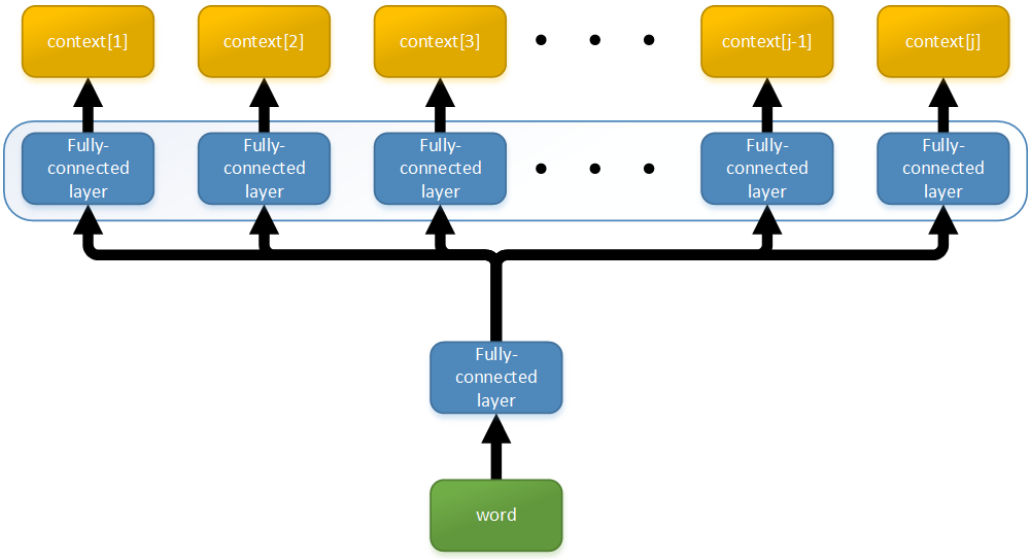

Рис. 5: Continuous skip-gram, представленная в виде нейронной сети

реализации — word2vec[3].

В качестве обучающей выборки используются пары $(w_i, c_i) \in W \times C$, где w_i — слово из текста в i-ой позиции, а c_i — контекст этого слова, то есть $s/2$ ближайших слов справа и столько же слева (s — размер контекста).

Теперь строим модель, которая будет по w_i прогнозировать c_i. Пусть $|V|$ — число различных слов в корпусе, а N — размерность скрытого слоя модели, она же — размерность представления. Тогда для каждого слова v введём векторы: $In(v) \in \mathbb{R}^N$ и $Out(v) \in \mathbb{R}^N$. Тогда можно записать вероятностную модель

$$p(c_{ik}|w_i) = \frac{\exp(In(w_i)^\top Out(c_{ik}))}{\sum_{w' \in W} \exp(In(w_i)^\top Out(w'))}$$

параметры которой можно найти, вычислив

$$\prod_{i=1}^{|V|} \prod_{k=1}^{s} p(c_{ik}|w_i) \to \max_{In,Out}$$

[3]https://code.google.com/archive/p/word2vec/

Такую модель можно рассматривать как нейронную сеть с одним скрытым слоем и без функций активации. Причём такая сеть будет схожа с сетью из [2], но со следующими отличиями: веса для прогнозирования слов из контекста в continuous skip-gram общие, то есть не зависят от позиции слова в контексте; не используется функция активации; для вычисления $p(c_{ik}|w_i)$ используется negative sampling, описанный в статье.

После обучения модели векторы $In(w)$ предлагается использовать как представления слов.

Continuous bag-of-words — очень похожая на continuous skip-gram модель, отличающаяся тем, что прогнозирует не контекст по слову, а наоборот, слово по контексту.

3.8 char2vec

В статье [4] предложена модель, которая решает задачи разделение слова на морфемы (без учителя) и отображение слова (на уровне символов) в его векторное представление.

Сначала берётся модель skip-gram, описанная выше, в которой каждому слову ассоциируется по два вектора: по входному и выходному.

Основная идея char2vec — заменить таблицу входных векторов на обучаемую функцию $f(w)$, которая будет принимать на вход посимвольное представление слова. После обучения такой модифицированной модели, функцию $f(w)$ можно будет использовать как отображение посимвольного представления произвольного слова на его представление.

В качестве функции $f(w)$ предлагается использовать нейронную сеть. Первым слоем предлагается поставить полносвязный слой, позволяющий обучить представления для символов, вторым — двунаправленную LSTM.

Во входную строку добавляются символы начала и конца, причем так, чтоб forward LSTM не читала символ конца, а backward — символ начала. Такое выравнивание сделано для более корректного разбиения на морфемы. Далее поставлен полносвязный слой для каждого момента времени (т.е. для каждой буквы) с общими параметрами между моментами времени. Далее — выходы с каждого момента времени суммируются с весами, полученными из механизма внимания (подобная схема рассмотрена в 4.6).

Числа, полученные на выходе механизма внимания используются в качестве нечетких разделителей морфем.

3.9 Sequence-to-sequence модели

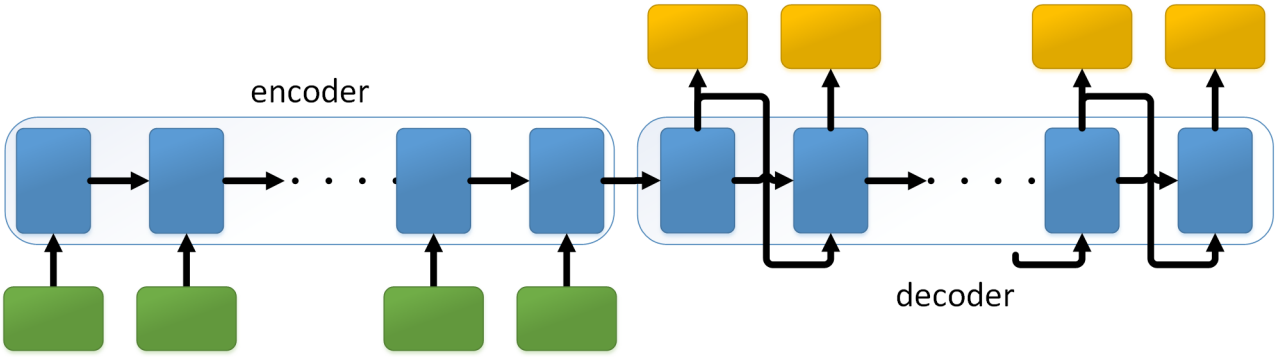

Рис. 6: Sequence to sequence модель

В статье [18] была представлена архитектура рекуррентной сети, у которой не только вход мог быть переменной длины, но и выход. Она состояла из двух частей: кодировщика и декодировщика. Кодировщик представлял из себя обычный рекуррентный слой, а декодировщик в первый момент времени принимал некоторый токен начала строки, в остальные моменты времени — выход с предыдущего, а последнее скрытое состояние из кодировщика назначалось начальным скрытым состоянием декодировщика.

При помощи такой модели можно обучать представления фиксированной длины для последовательностей, подавая на вход и выход одни и те же данные, значение на выходе кодировщика при этом и являлось представлением.

4 Основная часть

4.1 Общее описание модели

Для начала необходимо выбрать класс моделей, подходящий для сформулированной задачи. Подходы из так называемого “классического машинного обучения”, вроде метода опорных векторов, решающих деревьев и прочих, не подходят, поскольку требуют входные данные фиксированного размера и не предполагают, чтоб они имели структуру. Воспользоваться ими возможно, но это приведёт к необходимости ручного составления признаков.

Вероятностные графические модели, например, скрытые марковские модели, поддерживают работу с входными данными произвольной длины, так же имеются исследования, в которых они применялись для задачи восстановления регрессии, тем не менее, они не являются такими же гибкими, как ставшие последнее время крайне популярными свёрточные и рекуррентные нейронные сети.

В результате огромного числа исследований, свёрточные и, в особенности, рекуррентные нейронные сети было показано, что они отлично подходят для задач автоматической обработки языка, вне зависимости от того, в каком виде текстовые данные подаются на вход: в виде последовательности букв, слов или, например, морфем. В связи с этим предлагается использовать эти модели в данной работе.

Далее в этой части будут предложены функция потерь и метрики качества, ряд последовательно усложняющихся архитектур нейронных сетей, а так же подход, который добавляет в представление информацию о символах, который, как будет показано, окажется полезным в прикладных зада-

чах.

4.2 Функция потерь

Выход модели решаемой задачи — представление слова, то есть вектор из $\mathbb{R}^n$. Следовательно, с точки зрения машинного обучения, необходимо строить модель восстановления регрессии с многомерным выходом. Традиционно используемая функция потерь для одномерного случая — квадрат ошибки:

$$l_{error\_squared}(\hat{y}, y) = (\hat{y} - y)^2$$

где y — ожидаемое значение, $\hat{y}$ — прогноз, сделанный моделью. Эта функция потерь легко выводится, если подбор параметров модели осуществляется при помощи метода максимального правдоподобия при предположении о том, что к наблюдаемым выходам добавлен аддитивный гауссовский шум [19]. На случай многомерного выхода можно сделать следующее обобщение (в случае предположения, что вектор шума распределён нормально с нулевым вектором математического ожидания и единичной матрицой ковариации, умноженной на некоторую константу):

$$l_{euclidian}(\hat{y}, y) = \sum_i (\hat{y}_i - y_i)^2$$

что является квадратом евклидова расстояния.

Если перейти к матричной нотации, то можно получить следующий результат:

$$l_{euclidian}(\hat{y}, y) = \|\hat{y} - y\| = (\hat{y} - y)^\top (\hat{y} - y) = \|\hat{y}\|^2 + \|y\|^2 - 2\hat{y}^\top y =$$

$$= \|\hat{y}\|^2 + \|y\|^2 + 2\|\hat{y}\|\|y\| l_{cosine}(\hat{y}, y)$$

где $l_{cosine}(\hat{y}, y)$ — отрицательная косинусная мера:

$$l_{cosine}(\hat{y}, y) = -\frac{\hat{y}^\top y}{\|\hat{y}\|\|y\|}$$

Если предположить, что $\|\hat{y}\| = \|y\| = 1$, то l_{cosine} и $l_{euclidian}$ равны с точностью до мультипликативной константы. В общем случае $l_{euclidian}$ можно рассматривать как l_{cosine} с регуляризатором на длину вектора $\hat{y}$.

Так же следует отметить, что для вычисления схожести представлений слов обычно предлагается использовать косинусную меру. Например, в word2vec, одном из самых известных инструментов для обучения представлений слов используется именно косинусная мера. В связи с этим предлагается использовать l_{cosine} в качестве функции потерь. Как показали эксперименты, лучшие результаты (в соответствии с метрикой качества, о которой будет рассказано ниже) были получены при использовании следующей функции потерь:

$$l_{cosine\_reg}(\hat{y}, y) = l_{cosine}(\hat{y}, y) + c \times l_{euclidian}(\hat{y}, y)$$

где c — константа, подбираемая на валидационной части датасета.

4.3 Метрики качества

Оценивать качество обученной модели можно метрикой, которая будет использовать описанную выше функцию потерь. Однако к метрике нет таких требований, как к функции потерь: она не обязана быть ни гладкой, ни даже непрерывной. Более того, допустимо, что она будет менее вычислительно эффективной, так как используется реже и на меньших объёмах данных.

Отметим качества желаемой метрики:

- Интерепретируемость. Хочется иметь представление, насколько модель хорошая сама по себе, на не на каких-то конкретных прикладных задачах. Среднеквадратическая ошибка или среднеекосинусное расстояние не дают такого представления.

- Учёт значимости отдельных координат вектора y. Значимыми могут оказаться лишь некоторые координаты вектора представления слова, причём в векторах для разных слов значимые координаты тоже могут отличаться.

В связи с вышеуказанными причинами предлагается ввести метрику, значение которой вычисляется следующим образом: возьмём прогноз представления $\hat{y}^{(i)}$ предлагаемой модели для некоторого слова $w^{(i)}$ и найдём top n ближайших соседей к $\hat{y}^{(i)}$ среди всех векторов, на которых мы обучали и тестировали модель. Если среди этих top n ближайших соседей нашелся вектор, соответствующий слову $w^{(i)}$, до делаем $score^{(i)}$ равным единице, иначе – нулю. Усреднение $score^{(i)}$ для всех i из валидационной/тестовой выборки будет значением метрики качества для текущей модели. Запишем это формально:

$$score_{top_n}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{w^{(i)} \in \mathcal{D}} I[rank(w^{(i)}, nearest(\hat{y}^{(i)})) \leq n]$$

- $nearest(\hat{y})$ — упорядоченный по близости список соседей вектора $\hat{y}$;

- $rank(w, list)$ — индекс ячейки списка $list$, в которой находится слово w;

- $\mathcal{D}$ — валидационная или тестовая выборка.

Таким образом, метрика перестаёт быть чувствительной к тому, что в различных частях пространства плотность векторов может быть сильно разной. А, например, значение $score_{top_n} = 0.5$ можно интерпретировать следующим образом: для половины слов валидационной/тестовой выборки прогноз оказался *достаточно* хорошим (если n сильно меньше $\mathcal{D}$).

Можно избавиться от n и предложить альтернативную метрику:

$$score_{weighted}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{w^{(i)} \in \mathcal{D}} \frac{1}{rank(w^{(i)}, nearest(\hat{y}^{(i)}))}$$

Можно заметить схожесть с метрикой mean average precision, которая используется в ранжировании. Основное отличие заключается в том, что в рассматриваемой задаче для каждого “запроса” существует только один “релевантный документ” (возможно, допустимо было бы считать за правильный ответ так же, например, слово из “запроса”, но в иной морфологической форме, но такая модификация в данной работе не исследовалась).

4.4 Входные данные

Входными данными в решаемой задаче являются слова в виде последовательности букв. Каждую букву в слове можно рассматривать как категориальный признак, это значит, что использовать кодирование каждой буквы различными числами будет некорректно.

Поскольку число различных используемых символов совсем небольшое (30–50, в зависимости от тематики текстового корпуса), можно воспользоваться простым способом кодирования — one-hot-encoding. Сначала каждой букве требуется сопоставить свой индекс $i \in 1, \ldots, N$, где N — число различных букв. Теперь каждое слово можно будет описать вектором, состоящего из нулей, за исключением i-ой позиции, на которой будет стоять

единица. Таким образом, входные данные можно описывать матрицами из $[0,1]^{l \times N}$, где l — длина строки.

4.5 Базовая архитектура

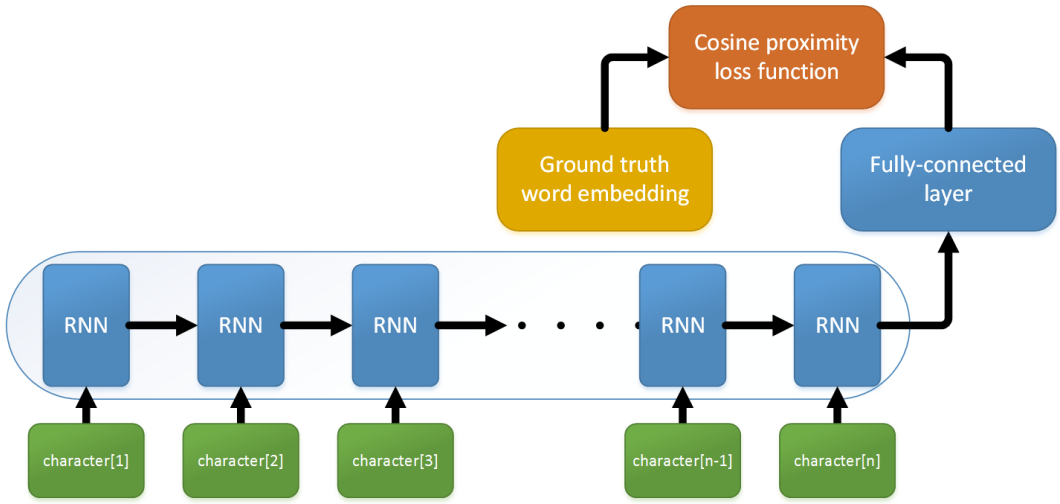

Рис. 7: Базовая архитектура с рекуррентным слоем

Поскольку входные данные являются структурированными, в качестве первого скрытого слоя нейронной сети будем использовать либо рекуррентный слой, либо свёрточный. Эксперименты будут проведены для каждого из этих случаев.

Использовался рекуррентный слой с LSTM юнитами, описанными в разделе 3.4. Как показали эксперименты, для достижения хорошего качества требуется использовать достаточно большую размерность рекуррентного слоя, порядка 1000–2000. При использовании большей размерности модель переобучалась, так как число параметров LSTM юнита квадратично зависит от размерности скрытого состояния.

Чтобы выходы рекуррентного слоя в каждый момент времени зависели не только от левой прочитанной часть слова, но и от правой, использо-

валась двунаправленная рекуррентная сеть [20]. При такой архитектуре входные данные подаются на два независимых рекуррентных слоя, которые читают строку слева направо и справа налево соответственно. Выходы этих слоёв в каждый момент времени конкатенируются.

Во избежание переобучения в рекуррентном слое использовался variational dropout [16], но с небольшой вероятностью обнуления компонент рекуррентных связей ($1 - p$ брался из отрезка $[0.05, 0.2]$).

За рекуррентным слоем следует полносвязный, поэтому следует преобразовать выход первого так, чтоб его размерность не зависела от размерности входа. Один из способов осуществить это — применить global maxpooling, описанный в разделе 3.5. Другой — передавать на вход полносвязному слою только скрытое состояние последнего момента времени.

Благодаря отказу от использования функции активации между слоями значительно повышалось качество прогноза модели, что интересно, аналогичная ситуация описана в оригинальной статье про word2vec [1] по сравнению с [2].

Как было упомянуто ранее, за рекуррентным слоем следует полносвязный, описываемый матрицей $W_{\text{output}} \in \mathbb{R}^{e \times r}$ и вектором смещения $b_{\text{output}} \in \mathbb{R}^e$, где e — размерность прогнозируемого представления, а r — размерность рекуррентного слоя.

Теперь рассмотрим случай со свёрточным слоем вместо рекуррентного. В нём использовались ядра свёртки разной ширины: от 2 до 5 включительно, в соответствии с тем, как это было описано в статье [21]. Число ядер линейно зависело от их ширины, а их суммарное число было порядка 10000. На этот раз функция активации используется, причём лучшие результаты были получены при помощи rectified linear unit [22].

4.6 Механизм внимания

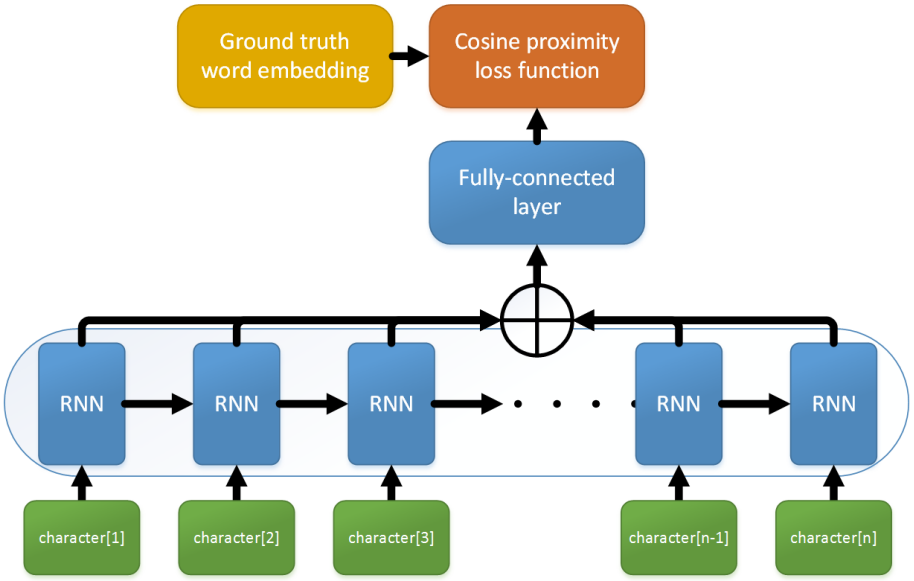

Рис. 8: Архитектура с механизмом внимания

Одной из самых популярных идей последних лет в глубоком обучении стал механизм внимания [23], позволяющий динамически считывать информацию из последовательностей переменной длины без необходимости заранее отображать её в вектор фиксированной размерности. Позже была предложена модификация — hard attention [24], но ради простоты в данной работе она не была использована.

Поскольку в решаемой задаче выход — это вектор, а не последовательность произвольной длины, будет использоваться частный случай механизма внимания (по крайней мере, в этом разделе), подобный тому, что описан в [4].

Пусть $h^{(t)}$ — выход свёрточного/рекуррентного слоя в момент времени t. Тогда вычислим для него вес:

$$a_t = \text{softmax}(w_{\text{att}_2}^\top \text{ReLU}(W_{\text{att}_1} h_t))$$

где $\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp x_j}$, $\text{ReLU}(x) = \max(0, x)$, $a_t \in \mathbb{R}$, $h_t \in \mathbb{R}^r$, $W_{\text{att}_1} \in \mathbb{R}^{p \times r}$, $w_{\text{att}_2} \in \mathbb{R}^p$.

Вектор a будем называть картой внимания. Выход механизма внимания вычисляется следующим образом:

$$h_{\text{att\_output}} = \sum_t a_t h_t$$

Выход модели аналогичен 4.5:

$$v_{att} = W_{\text{output}} h_{\text{att\_output}} + b_{\text{output}}$$

$h_{\text{att\_output}}$ — это выпуклая комбинация выходов свёрточного/рекуррентного слоя, поскольку $\sum_t a_t = 1$ и $\forall t : a_t > 0$.

Но если не использовать softmax для вычисления a_t, то есть заменить на линейную функцию активации (подобная идея предложена в [25], но с другой целью), то в нашем случае такая замена позволяет улучшить качество, а при использовани ReLU вместо softmax при прежнем качестве карта внимания становится разреженной. Минусом такой замены является то, что $h_{\text{att\_output}}$ перестаёт быть выпуклой комбинацией. Как следствие, на очень длинных или очень коротких входных последовательностях длина $h_{\text{att\_output}}$ может значимо отличаться. Но для текущей задачи это не является проблемой, так как в качестве функции потерь используется косинусное расстояние, а слои, зависящие от $h_{\text{att\_output}}$ — линейные.

Основная идея [25] заключается в том, что предлагается сделать карту внимания линейной функцией от h:

$$\hat{a}_t = w_{\text{att}_3}^\top h_t$$

$w_{\text{att}_3} \in \mathbb{R}^r$, и за счёт этого уменьшить количество вычислений:

$$\hat{h}_{\text{att\_output}} = \sum_t \hat{a}_t h_t = \sum_t (w_{\text{att}_3}^\top h_t) h_t = \left(\sum_t h_t h_t^\top \right) w_{\text{att}_3} = h h^\top w_{\text{att}_3}$$

где $h \in \mathbb{R}^{r \times l}$. Такой подход оказывается полезным в 4.6.1.

4.6.1 Индивидуальные карты внимания

Теперь модифицируем модель так, чтоб для каждой компоненты представления обучалась индивидуальная карта внимания. Сделать это можно так:

$$a'_{ti} = \text{ReLU}((W'_{\text{att}_2})_i^\top \text{ReLU}((W'_{\text{att}_1})_i h_t))$$

где $a'_{t*} \in \mathbb{R}^e$, $\forall i \in 1, \ldots, e : (W'_{\text{att}_1})_i \in \mathbb{R}^{p \times r}$, $(W'_{\text{att}_2})_i \in \mathbb{R}^p$.

Выход теперь будет вычисляться так:

$$(v_{att\_per\_coord})_i = (W_{\text{output}})_{i*} \sum_t a'_{ti} h_t + (b_{\text{output}})_i$$

Такая модель оказывается слишком сложной с вычислительной точки зрения, поэтому будем использовать общую W_{att_1} для всех карт внимания:

$$a''_{ti} = \text{ReLU}((W'_{\text{att}_2})_i^\top \text{ReLU}(W_{\text{att}_1} h_t))$$

Обучив эту модель, можно заметить, что многие карты внимания a'_{*i} похожи друг на друга, значит, стоит её упрощать, что будет сделано в следующем разделе.

4.6.2 Общие карты внимания

Предлагается обучить некоторое небольшое количество карт внимания и для каждой из них делать прогноз представления слова, а потом эти представления комбинировать в одно.

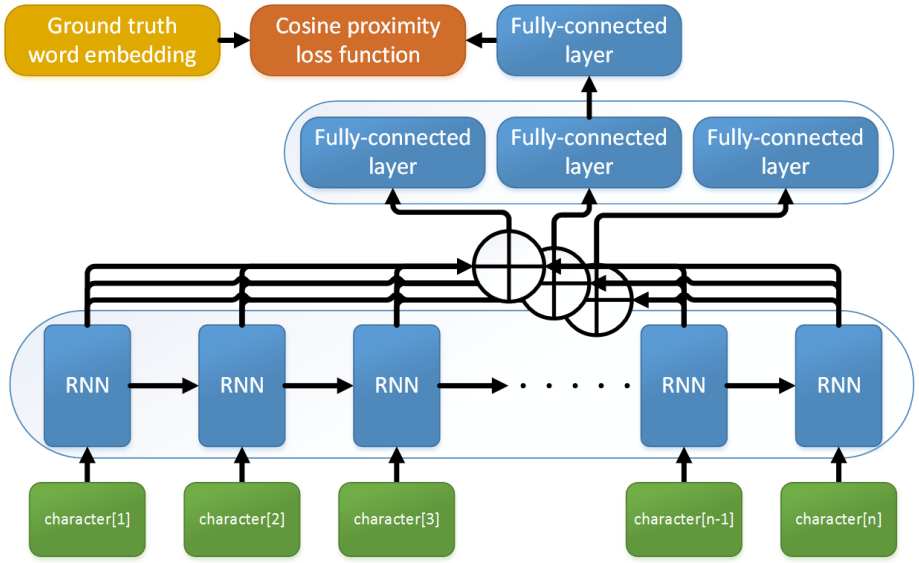

Рис. 9: Архитектура с несколькими картами внимания

Для каждой карты внимания

$$a'''_{tj} = \mathrm{ReLU}((W'_{\mathrm{att}_2})_j^\top \mathrm{ReLU}((W'_{\mathrm{att}_1})_j h_t))$$

$\forall j \in 1, \ldots, m$, где m — небольшое число, вычислим своё представление:

$$v_{att\_shared_j} = (W'_{\mathrm{output}})_j \sum_t a'''_{tj} h_t + (b'_{\mathrm{output}})_j$$

где $(W'_{\mathrm{output}})_j \in \mathbb{R}^{e \times r}$ и $(b'_{\mathrm{output}})_j \in \mathbb{R}^e$.

Теперь $v_{att\_shared_j}, j \in 1, \ldots, m$ можно объединить в один вектор несколькими способами, например, линейной комбинацией:

$$v_{att\_shared\_final} = \sum_j (w''_{output})_j v_{att\_shared_j}$$

где $w''_{output} \in \mathbb{R}^m$.

Или покоординатной линейной комбинацией:

$$(v'_{att\_shared\_final})_i = \sum_j (W''_{output})_{ij} (v_{att\_shared_j})_i$$

где $W''_{output} \in \mathbb{R}^{e \times m}$.

Попытка сделать веса в линейной комбинации зависимыми от $v_{att\_shared_j}$ не удалась, так как метод оптимизации при обучении такой модели расходился.

Интересно, что при объединении векторов $v_{att\_shared_j}$ линейной комбинацией, сами векторы $v_{att\_shared_j}$ в некоторых случаях становятся осмысленными, пример будет продемонстрирован в разделе 5.4,

В качестве возможного расширения этой модели можно использовать её для прогнозирования морфологии слова: если известно, что j-ая морфема начинается в позиции t_1 и заканчивается в t_2, то в векторе a'''_{*j} присвоим единицу элементам $a'''_{t_1 j}$ и $a'''_{t_2 j}$ и ноль всем остальным элементам a'''_{*j} (то есть каждая карта внимания будет ответственна только за одну морфему). Но такое исследование уже выходит за рамки данной работы.

4.7 Многослойная рекуррентная архитектура

Всё вышеописанные модели содержали только один рекуррентный слой. Связано это с тем, что добавление второго слоя во всех случаях (кроме последней модели) понижало качество. Поэтому опишем обновлённую архитектуру для модели из 4.6.2.

Пусть h_t — выход с (первого) рекуррентного слоя в момент t. Тогда, во-первых, встроим рекуррентный слой в механизм внимания. Тогда в t-ый момент времени он будет принимать h_t, а возвращать — h_t^{att}. Вычисление карт внимания теперь будет происходить так:

$$\hat{a_{tj}}''' = \mathrm{ReLU}((W'_{\mathrm{att}_2})_j^\top \mathrm{ReLU}(h_t^{att}))$$

Пока что произошло только одно изменение: между юнитами перво-

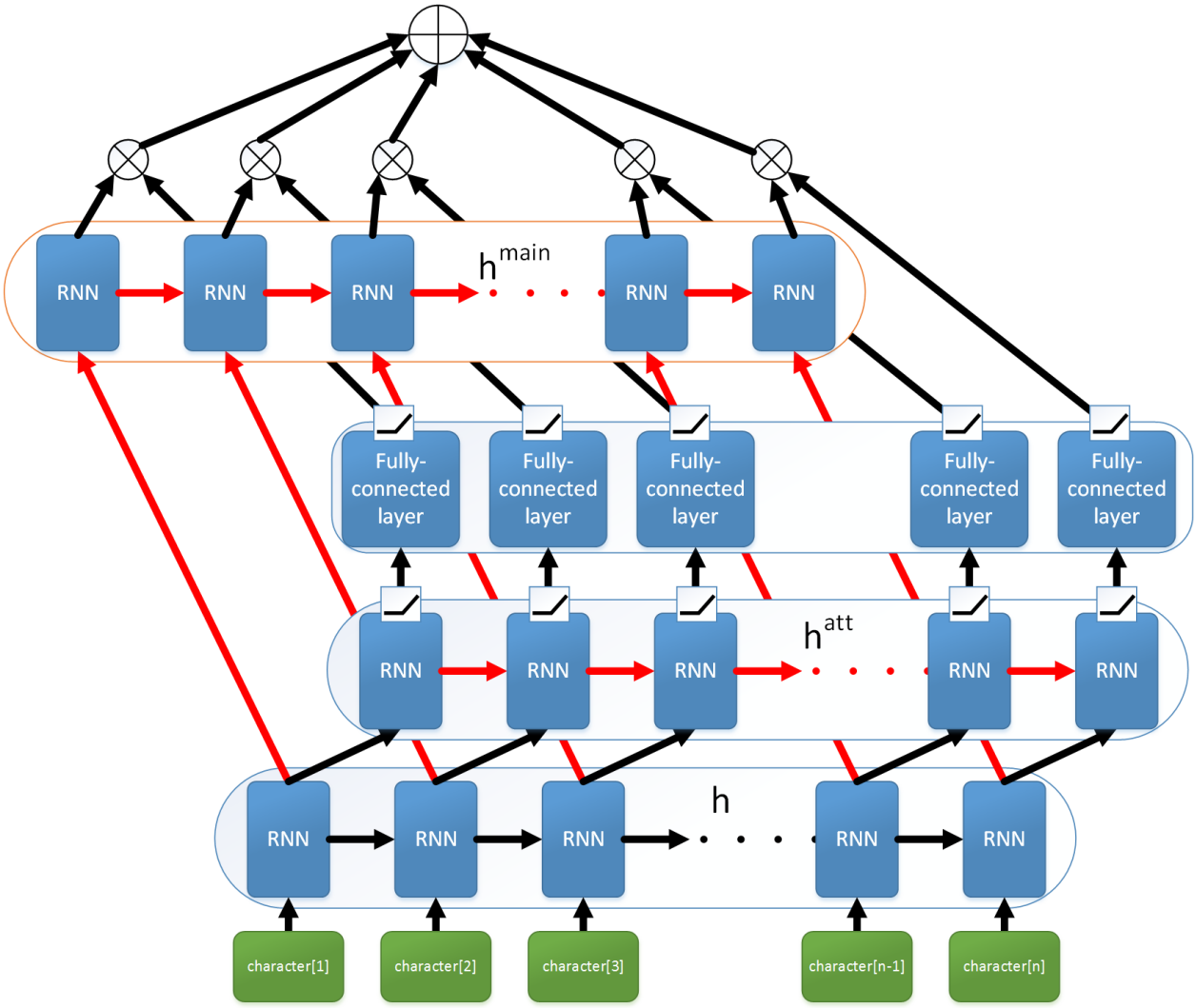

Рис. 10: Механизм внимания с встроенным рекуррентным слоем.

го слоя механизма внимания появились рекуррентные связи. Размерность этого рекуррентного слоя также равна p, как и размерность выхода первого слоя в механизме внимания.

Во-вторых, добавим ещё один рекуррентный слой такой же размерности, как первый. Так же, как и рекуррентный слой из механизма внимания, в момент времени t на входе будет h_t, а на выходе h_t^{main}.

Тогда формула для вычисления одного из представлений легко переписывается:

$$\hat{v}_{att\_shared_j} = (W'_{\text{output}})_j \sum_t \hat{a}'''_{tj} h_t^{main} + (b'_{\text{output}})_j$$

Красным в формулах и на рисунке 10 выделены изменения по сравнению с 4.6.2.

4.8 Расширение представления слова информацией о символах

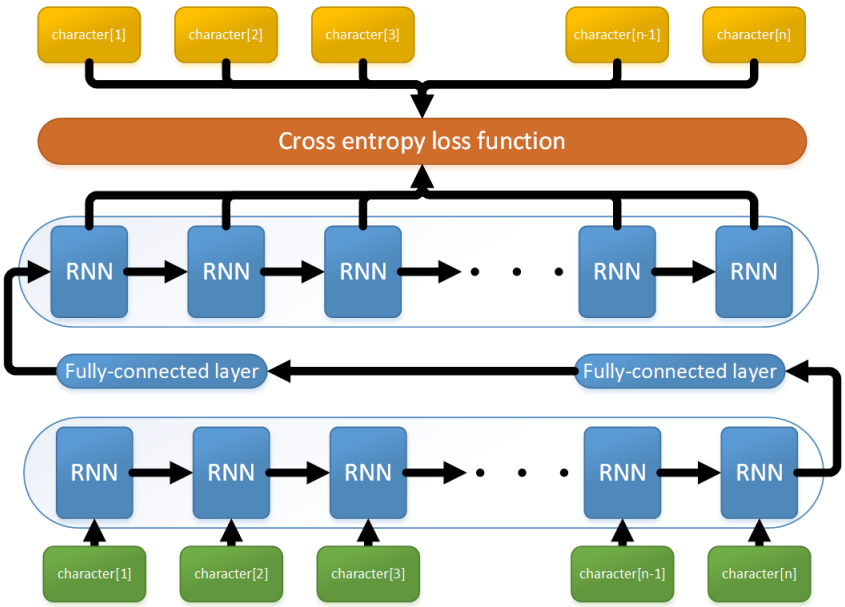

Рис. 11: Sequence-to-sequence модель для обучения представлений слов, содержащих информацию о символах

Проблемой остаётся то, что векторы, получаемые из предложенной модели всё равно не учитывают посимвольное представление текста. Поскольку в некоторых прикладных задачах, например, поиске именованных сущностей может оказаться, что наличие некоторой регулярной подпоследовательности в слове окажется хорошим признаком, только представление

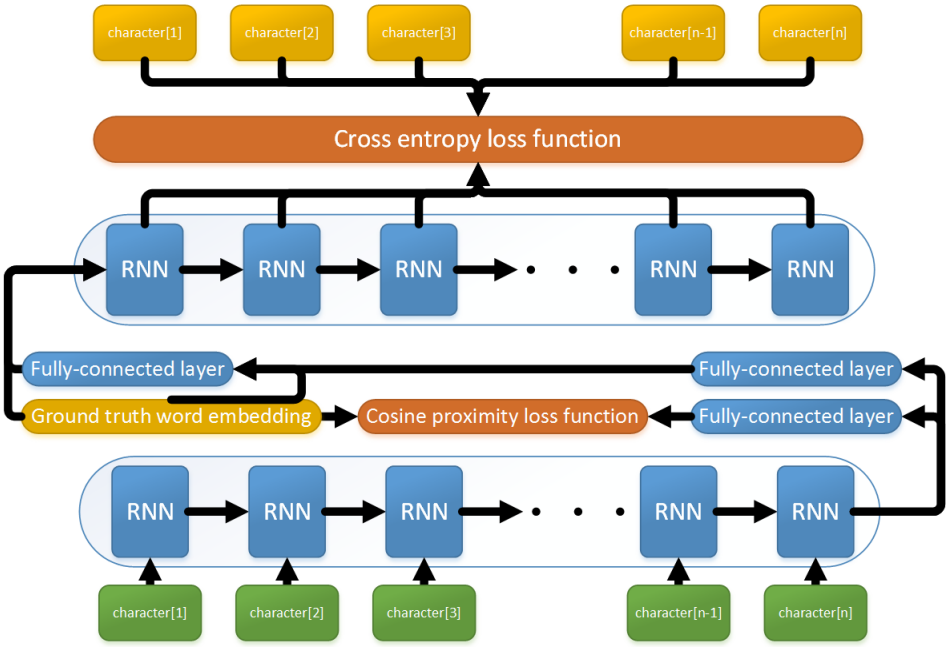

Рис. 12: Объединение архитектуры sequence-to-sequence с базовой моделью из 4.5

может не зависеть от наличия такой подпоследовательности и, соответственно, не содержать информацию о ней.

Поэтому предлагается обучить представление слова при помощи модели sequence-to-sequence у которой на входе и выходе будет одно и то же слово, а между кодировщиком и декодировщиком будет находиться вектор небольшой размерности, который будет содержать сжатую информацию о символах в слове. Предполагается, что в этот вектор будет кодировать некоторые регулярно встречающиеся подпоследовательности в словах.

Пусть $h_{encoder\_output} \in \mathbb{R}^r$ — выход кодировщика. Тогда обучаемым представлением будет вектор

$$c = W_{bottleneck\_in} h_{encoder\_output}$$

где $c \in \mathbb{R}^k, W_{bottleneck_in} \in \mathbb{R}^{k \times r}$.

А на вход декодировщику будет подан вектор $h_{decoder\_input} \in \mathbb{R}^r$:

$$h_{decoder\_input} = W_{bottleneck\_out} c$$

где $W_{bottleneck\_out} \in \mathbb{R}^{r \times k}$.

Чтобы не было необходимости сохранять в c информацию, которая уже имеется в обученном представлении v, предлагается объединить только что описанную модель с моделью из 4.5, что изображено на рисунке 12. Формула для вычисления значения входа декодировщика обновиться следующим образом:

$$h_{decoder\_input} = W'_{bottleneck\_out} concat(c, v)$$

где $W'_{bottleneck\_out} \in \mathbb{R}^{r \times (k+e)}$, а $concat(c, v)$ — операция конкатенации векторов.

5 Экспериментальная часть

В этой части будут продемонстрированы: сравнение моделей, описанных в работе, визуализации скрытых состояний сети и тестирование на задаче поиска именованных сущностей.

5.1 Подготовка данных

Эксперименты проводились на представлениях слов, которые были получены при помощи модели continuous bag-of-words, обученной на текстовом корпусе ok.ru, размерность представления слова — 300.

При обучении представлений обычно все слова в обучающей выборке нормализуются, то есть приводятся к начальной форме. Нормализация особо актуальна для морфологически богатых языков, так как в них для одного слова может существовать множество форм, некоторые из которых могут встречаться редко (представления для таких слов могут быть обучены не достаточно качественно) или не встречаться вообще. Поскольку целью этой работы является, в частности, научиться восстанавливать представления для форм слов, которые не встречались в обобщающей выборке, нормализация слов в корпусе явно усложнит эту задачу.

Всего имеется около 100000 пар (слово; представление слова). Обучающую выборку составим случайным образом из 80% этих данных, а валидационную, как и тестовую — из 10%. Все эти три выборки являются непересекающимися.

5.2 Сравнение различных архитектур

Протестируем базовую модель со свёрточным слоем с различным числом ядер:

Число ядер	$score_{top\_100}$	$score_{weighted}$
2800	0.24	0.06
7000	0.38	0.12
10500	0.42	0.14
14000	0.44	0.15
17500	0.41	0.13
21000	0.40	0.12

А так же с рекуррентным слоем с различными размерностями скрытого состояния:

Размерность скрытого состояния	$score_{top\_100}$	$score_{weighted}$
500	0.46	0.17
1000	0.50	0.20
1500	0.51	0.20
1750	0.51	0.21
2000	0.51	0.21
2500	0.44	0.15

Судя по проведённому эксперименту, использование рекуррентного слоя значимо улучшает качество, к тому же рекуррентный слой имеет выход меньшей размерности. С другой стороны, при использовании свёрточного слоя одна итерация занимала меньше времени и в целом метод сходился за меньшее число итераций. В дальнейшем все эксперименты будем проводить с использованием рекуррентного слоя размерности 2000.

Теперь подберём коэффициент c в функции потерь:

c	$score_{top\_100}$	$score_{weighted}$
0	0.49	0.19
1	0.49	0.19
2	0.49	0.19
4	0.50	0.20
8	0.51	0.21
16	0.49	0.19
32	0.45	0.16

Оказывается, не смотря на то, что косинусная мера никак не учитывает длины векторов, регуляризация позволяет повысить качество.

Сравним качество для всех предложенных архитектур:

c	$score_{top\_100}$	$score_{weighted}$
Базовая модель	0.51	0.20
Базовый механизм внимания	0.49	0.19
Механизм внимания с индивидуальными картами	0.47	0.18
Механизм внимания с общими картами и покоординатным объединением векторов	0.52	0.22
Механизм внимания с общими картами	0.53	0.23
Механизм внимания с общими картами и с двумя рекуррентными слоями	0.54	0.23

Последняя модель оказалась лучшей по качеству, но стоит отметить, что одна итерация такой модели требует в восемь раз больше времени, чем базовая. Поэтому предлагается для sequence-to-sequence взять за основу базовую модель.

Теперь подберём число карт внимания для механизм внимания с общими картами:

c	$score_{top\_100}$	$score_{weighted}$
1	0.49	0.19
2	0.51	0.21
3	0.52	0.22
4	0.53	0.23
5	0.54	0.23
6	0.53	0.23
7	0.51	0.22

Результаты несколько неожиданные, поскольку, как будет видно на визуализациях, обычно ненулевые веса имеют только от одной до трёх карт.

Для поиска параметров во всех моделях использовался метод adam с learning rate равным 0.001. Изменение метода оптимизации на RMSProp или стохастический градиентный спуск при различных значениях learning rate не влекло за собой существенных изменений качества.

5.3 Тестирование sequence-to-sequence модели

Размерность скрытого слоя как кодировщика, так и декодировщика будет равна 500. Сравним модели, в качестве метрики качества взяв кросс-энтропию. Сравниваться будут две версии модели, в одной из которых на

вход декодировщику помимо выхода с кодировщика подаётся распределённое представление соответствующего слова.

Размерность обучаемого вектора	cross entropy, обычный seq2seq	cross entropy, с распределённым представлением
100	0.05	0.05
50	0.12	0.11
20	0.43	0.25
15	0.53	0.32
10	0.64	0.37
5	0.97	0.89
0	—	2.06

5.4 Визуализации для архитектур с картами внимания

В разделе 4.6.2 было сказано, что модель с общими картами внимания с своём предпоследнем слое содержит интерпретируемые представления слов, продемонстрируем это на нескольких примерах. Для двух векторов в предпоследнего слоя найдём по пять ближайших соседей среди известных представлений:

Для слова “смешно”:

смешно	смешные
забавно	смешно
некрасиво	анекдоты
глупо	прикольно
умно	ржака

Для слова “творческий”:

творческий	творческий
художественный	творческие
музыкальный	творчества
креативный	творческих
хореограф	музыкальный

Для слова “развита”:

развить	развитии
отразить	развитие
освоить	развита
вписаться	развиты
проявила	генетика

Для слова “совершать”:

вносить	совершения
отделять	совершение
функционировать	совершать
расширять	совершении
обновлять	совершая

Как видно, в некоторых случаях такие векторы описывают информацию о корне слова, в некоторых — о смысле, с некоторых — о морфологии.

А на рисунке 13 изображены карты внимания из модели, описанной в 4.7, дня некоторых слов. Можно заметить, что максимальные значения достигаются в местах соединения различных морфем (за исключением последнего примера). Об аналогичном наблюдении было написано в [4].

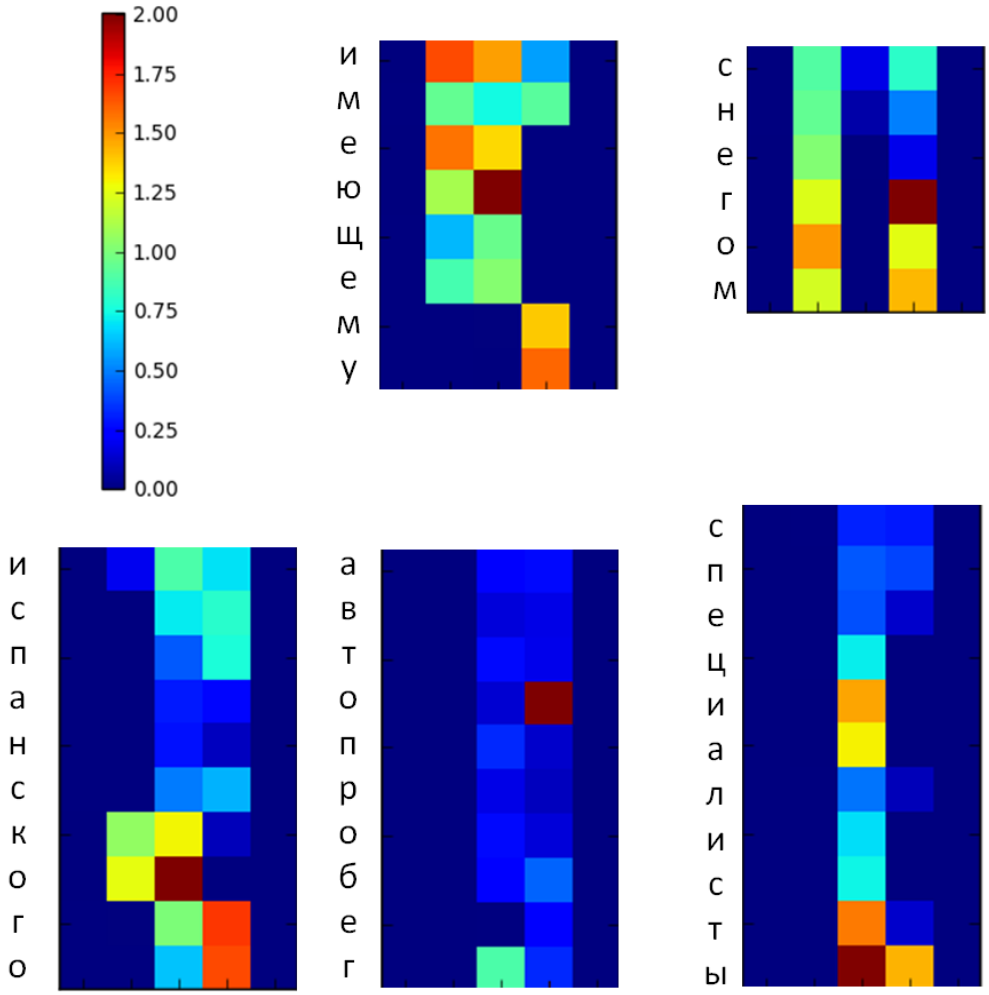

Рис. 13: Визуализации карт внимания

5.5 Применение к задаче распознавания именованных сущностей

Проблема со словами, которые встречаются крайне редко, имеется не только при работе с морфологически богатыми языками, но и при анализе статей по медицине и химии, поэтому задача поиска именованных сущностей в таких статьях является сложной задачей.

Предлагается протестировать нашу модель на такой задаче, а именно на датасете CHEMNDER [26]. Сначала обучимся на датасете с представ-

лениями от bio.nlplab.org, который в свою очередь был обучен на PubMed и PMC. Сделаем это на моделях из разделов 4.6.2 и 4.5:

c	$score_{top\_100}$	$score_{weighted}$
Базовая модель	0.015	0.002
Механизм внимания с общими картами	0.025	0.006

Кажется, что результаты довольно слабые, но их будет некорректным сравнивать с теми, что были получены на русскоязычном корпусе, поскольку используемые метрики не являются устойчивыми к размеру корпуса. Интересно, что на этот раз более сложная модель значимо оторвалась от базовой по качеству.

Попробуем увеличить выборку оставив только самые частотные слова. Информация о частотах униграм также предаставлена на bio.nlplab.org, воспользуемся ею и оставим только 100000 слов из 2 миллионов. Обучим модели на урезанной выборке:

c	$score_{top\_100}$	$score_{weighted}$
Базовая модель	0.07	0.0015
Механизм внимания с общими картами	0.11	0.029

Теперь воспользуемся моделью для поиска именованных сущностей, описанная в [27], реализация которой доступна в открытом доступе[4]. Размерность скрытого слоя сделаем равным 200 (как это было рекомендовано в [28]). Проведём ряд экспериментов на различных данных: будем обозначать расширение словаря нашей моделью (а именно моделью из 4.6.2) как

[4]https://github.com/abhyudaynj/LSTM-CRF-models

ext и дополнительно указывать размер датасета, на котором она обучалась (2 миллиона или 100 тысяч), если эта модель не используется, то неизвестным словам будет сопоставляться усреднённое представление; использование модели sequence-to-sequence будем обозначать как s2s и дополнительно указывать размерность вектора (10 или 100). Качество будет сравниваться при помощи метрики F1.

модель	F1
без расширений	0.611
ext(1M)	0.617
ext(100K)	0.620
s2s(10)	0.629
s2s(100)	0.621
ext(100K) + s2s(10)	0.641

Таким образом, мы показали, что представления слов, полученные из наших моделей позволяют несколько улучшить качество распознавания именованных сущностей на датасете CHEMDNER.

6 Заключение

В этой работе был представлен ряд моделей, позволяющих восстанавливать распределённые представления слов из их посимвольного представления. Была проведена их оценка и сравнение, а так же продемонстрировано успешное применение на прикладной задаче.

Планируется продолжить работу над этим проектом, а именно исследовать интерпретируемость прогнозов модели и протестировать её на других морфологически богатых языках.

Список литературы

[1] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[2] Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. Journal of machine learning research, 3(Feb), 1137-1155.

[3] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2002). Latent dirichlet allocation. Advances in neural information processing systems, 1, 601-608.

[4] Cao, K., & Rei, M. (2016). A Joint Model for Word Embedding and Word Morphology. arXiv preprint arXiv:1606.02601.

[5] Elman, J. L. (1990). Finding structure in time. Cognitive science, 14(2), 179-211.

[6] Yu, D., & Deng, L. (2014). Automatic speech recognition: A deep learning approach. Springer.

[7] Tieleman, T., & Hinton, G. (2012). RMSProp. COURSERA: Neural Networks.

[8] Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[9] Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. ICML (3), 28, 1310-1318.

[10] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[11] Graves, A. (2012). Supervised sequence labelling. In Supervised Sequence Labelling with Recurrent Neural Networks (pp. 5-13). Springer Berlin Heidelberg.

[12] Arjovsky, M., Shah, A., & Bengio, Y. (2016, June). Unitary evolution recurrent neural networks. In International Conference on Machine Learning (pp. 1120-1128).

[13] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4), 541-551.

[14] Zhang, X., & LeCun, Y. (2015). Text understanding from scratch. arXiv preprint arXiv:1502.01710.

[15] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929-1958.

[16] Gal, Y., & Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In Advances in Neural Information Processing Systems (pp. 1019-1027).

[17] Goldberg, Y., & Levy, O. (2014). word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722.

[18] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Advances in neural information processing systems (pp. 3104-3112).

[19] Bishop, C. M. (2006). Pattern recognition. Machine Learning, 128, 1-58.

[20] Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11), 2673-2681.

[21] Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2016, March). Character-aware neural language models. In Thirtieth AAAI Conference on Artificial Intelligence.

[22] Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th international conference on machine learning (ICML-10) (pp. 807-814).

[23] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

[24] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... & Bengio, Y. (2015, June). Show, attend and tell: Neural image caption generation with visual attention. In International Conference on Machine Learning (pp. 2048-2057).

[25] de Brébisson, A., & Vincent, P. (2016). A Cheap Linear Attention Mechanism with Fast Lookups and Fixed-Size Representations. arXiv preprint arXiv:1609.05866.

[26] Krallinger, M., Leitner, F., Rabal, O., Vazquez, M., Oyarzabal, J., & Valencia, A. (2015). CHEMDNER: The drugs and chemical names extraction challenge. Journal of cheminformatics, 7(1), S1.

[27] Jagannatha, A. N., & Yu, H. (2016, November). Structured prediction models for RNN based sequence labeling in clinical text. In Proceedings

of the Conference on Empirical Methods in Natural Language Processing. Conference on Empirical Methods in Natural Language Processing (Vol. 2016, p. 856). NIH Public Access.

[28] Rei, M., Crichton, G. K., & Pyysalo, S. (2016). Attending to Characters in Neural Sequence Labeling Models. arXiv preprint arXiv:1611.04361.