Интернет-журнал «Науковедение» ISSN 2223-5167 https://naukovedenie.ru/

Том 9, №5 (2017) https://naukovedenie.ru/vol9-5.php

URL статьи: https://naukovedenie.ru/PDF/100TVN517.pdf

Статья опубликована 08.11.2017

Ссылка для цитирования этой статьи:
Закиров М.Э. Построение словарной морфологии на основе рекуррентного авто кодировщика // Интернет-журнал «НАУКОВЕДЕНИЕ» Том 9, №5 (2017) https://naukovedenie.ru/PDF/100TVN517.pdf (доступ свободный). Загл. с экрана. Яз. рус., англ.

УДК 62

Закиров Марат Энварович[1]
«Поисковый портал «Спутник», Россия, Москва
Разработчик направления
E-mail: marat61@gmail.com

Построение словарной морфологии на основе рекуррентного авто кодировщика

Аннотация. Целью данной работы было проверить, насколько хорошо простая архитектура рекуррентной нейросети способна запоминать последовательности символов применительно к морфологии русского языка. Описанная в данной работе модель, обучается преобразовывать слово в представление морфологического вектора конечного размера, из которого слово может быть декодировано обратно. В качестве потенциального применения модели: исправление опечаток, анализ морфологической близости, исправление опечаток, и способ хранения слов. В первой части будет описана модель простой нейросети, которая обрабатывая слово символ за символом переводит его представление во внутренние состояние рекуррентной сети, это часть сети называется кодировщиком. После чего другая часть занимается тем, что раскодирует это внутреннее представление сети в символьное представление, это часть модели называется декодировщик. Обучается модель как единое целое, на каждой итерации сначала отрабатывает кодировщик, а потом декодировщик, а применяется части могут как совместно, так и раздельно, в зависимости от поставленной задачи (к примеру, часто может возникать необходимость только в использовании кодировщика, для получения близких по морфологии слов, например, для поиска похожих слов). Во второй части данной работы, показаны результаты экспериментов и подводятся итоги.

Ключевые слова: нейросеть; рекуррентный авто кодировщик; морфология; обратное распространение ошибки сквозь время; морфологические синонимы; словарь; градиент; кодировщик; декодировщик

В настоящее время нейросети все глубже проникают в различные отрасли информационного поиска, не так недалек тот день, когда весь информационный поиск будет работать на основе некоторой обучающейся в реальном времени нейросети работающей непосредственно на уровне символов, а быть может даже на основе их шрифтового, графического отображения. Однако темой данной работы является создание и демонстрация работы простой нейросети производящей свертку символьного представления слова в

[1] 117463, г. Москва, ул. Голубинская, д. 32/2, кв. 711

некоторой вектор постоянной длины и его развертку в это же слово. Данная работа покажет какую информацию может нести подобное представление и какие могут быть возможные использования.

Архитектура

Как уже было сказано данная работа рассматривает архитектуру нейросети получившую название рекуррентного авто кодировщика. Расшифруем это название. Авто кодировщик – это вид нейросети, при котором модель стремится воспроизвести вход на выходе нейросети, поскольку информационная емкость нейросети конечная, а данных может быть очень много, такая нейросеть начинает обобщать данные, и на выходе кодировщика получается компактное представление входных данных, отсюда название авто кодировщик. Рекуррентный потому, что данная сеть, построенная на основе двух рекуррентных нейросетей, одна из которых производит по символьную свертку слова (кодировщик), а другая развертку (это декодер).

Работу энкодера описывает формула 1. В энкодер последовательно подаются коды символов x_t слова и предыдущее состояние h^e_{t-1} энкодера. Состояние энкодера описывается плотным вектором небольшой длины, а код символа представляет из себя кодировку, в которой все элементы вектора равны нулю кроме единственного, соответствующего коду некоторого символа. Новое состояние энкодера h^e_t формируется как результат применения нелинейности (в данном случае *tanh*) к векторной сумме, матричных произведений, формирующий таким образом обычный слой нейросети замкнутый итеративно на себя.

$$h^e_t = \sigma(W^e x_t + H^e h^e_{t-1}) \quad (1)$$

Работа декодера описывается сходной формулой за исключением того, что на вход ему каждый шаг подается в качестве входа вектор последнего состояния энкодера.

$$h^d_t = \sigma(W^d h^e_{last} + H^d h^d_{t-1}) \quad (2)$$

В дальнейшем состояние декодера на каждом шаге преобразуется (декодируется) в символ при помощи формулы 3, которая вычисляет вероятность символов алфавита (нужно добавить, что для корректной работы модели необходимо добавить к алфавиту символов специальный символ остановки).

$$p^i_t = \frac{e_i^{\sigma(Vh^d_t)}}{\sum_i^m e_i^{\sigma(Vh^d_t)}} \quad (3)$$

В данном случае к выходу декодера применяется еще один нелинейный слой, после чего применяется экспонента с делением на статсумму, так чтобы p^t_i можно было интерпретировать как вероятность *i* символа алфавита, полученного на шаге *t*.

Обучение модели

Машинное обучение состоит по сути из двух главных компонент, первая отвечает за интерпретацию модели, а вторая за стратегию оптимизации согласно заданной на первом этапе интерпретации.

Поскольку интерпретация уже задана, можно ответить на вопрос об оптимизации. Оптимизировать будем функционал, заданный формулой 4.

$$L_\Theta = -\sum_t^n \sum_i^m y_i^t log(p_i^t(\Theta)) \quad (4)$$

Данная формула описывает величину ошибки при неправильном декодировании одного слова, задача состоит в ее минимизации для всего набора словаря согласно формуле 5.

$$\Theta_{optimal} = argmin_\Theta(\sum_{w \in V} L_\Theta) \quad (5)$$

Полученный оптимальный набор параметров $\Theta_{optimal}$ и будет результатом обучения модели. В качестве стратегии обучения можно применить обычный стохастический градиентный спуск (который есть последовательное применение антиградиента к некоторому случайно выбираемому подмножеству обучающих примеров) на основе вычисленного градиента от параметров $\nabla\Theta$. Этот градиент вычисляется процессом, получившим название обратного распространения ошибки сквозь время и представляет из себя по сути обычный алгоритм обратного распространения примененный последовательно.

Выведем для примера формулу вычисления градиента для декодера. Ошибка декодера есть сумма ошибок предсказания символов некоторого слова $E = \sum_t E_t$ а потому градиент от весов так же есть сумма ошибок на каждом шаге $\frac{\partial E}{\partial \Theta} = \sum_t \frac{\partial E_t}{\partial \Theta}$, понимая это вычислим градиент ошибки на шаге *t* для матриц параметров V, *W*, *H* используя цепное правило, имя в виду, что $\otimes$ – п тензорное произведение векторов (или декартово произведение), $\odot$ поэлементное произведение векторов, иначе это матричное произведение, матрицы на вектор.

Для $\frac{\partial E_t}{\partial V}$ получается $\frac{\partial E_t}{\partial V} = \frac{\partial E_t}{\partial p_t}\frac{\partial p_t}{\partial V} = \frac{\partial E_t}{\partial p_t}\frac{\partial V h_t^d}{\partial V} = \frac{\partial E_t}{\partial p_t} \otimes h_t^d = (p_t - y_t) \otimes h_t^d$, что приводит нас к формуле 6

$$\frac{\partial E_t}{\partial V} = (p_t - y_t) \otimes h_t^d \quad (6)$$

Для $\frac{\partial E_t}{\partial H}$получим формулы 7 и 8, 8 – рекуррентная часть формулы

$$\frac{\partial E_t}{\partial H} = \frac{\partial E_t}{\partial h_{t+1}} \odot \frac{\partial h_{t+1}}{\partial H} \quad (7)$$

$$\frac{\partial h_{t+1}}{\partial H} = \frac{\partial h_{t+1}}{\partial (W x_t + H h_{t-1})} \otimes (h_t + H\frac{\partial h_t}{\partial H}) = (1 - (h_{t+1})^2) \otimes (h_t + H\frac{\partial h_t}{\partial H}) \quad (8)$$

Для $\frac{\partial E_t}{\partial W}$ получим аналогичные формулы по 9, 10

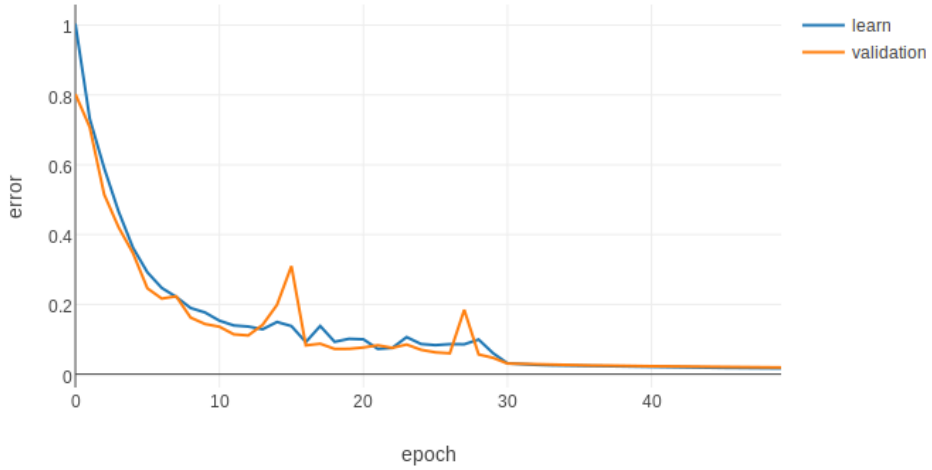
$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial h_{t+1}} \odot \frac{\partial h_{t+1}}{\partial W} \quad (9)$$

$$\frac{\partial h_{t+1}}{\partial W} = \frac{\partial h_{t+1}}{\partial (W x_t + H h_{t-1})} \otimes (x_t + H\frac{\partial h_t}{\partial W}) = (1 - (h_{t+1})^2) \otimes (x_t + H\frac{\partial h_t}{\partial W}) \quad (10)$$

Еще необходимо выражение для $\frac{\partial E_t}{\partial h_{t+1}}$

$$\frac{\partial E_t}{\partial h_{t+1}} = \frac{\partial E_t}{\partial p_t}\frac{\partial V h_{t+1}^d}{\partial h_{t+1}^d} = (p_t - y_t)V \quad (11)$$

Результаты экспериментов



Данная работа ставит своей целью, получение и изучение свойств закодированной словарной морфологии. Для начала обучим модель на словаре русских слов Зализняка это ~90000 слов. Поскольку наша модель должна извлекать словообразовательную информацию, интересно будет выделить случайный набор слов на валидацию (10 % всех слов) и посмотреть за тем как она уменьшается, скажем, процент от общего их числа. Вот так выглядит график уменьшения ошибки на обучающем множестве и на проверочном.

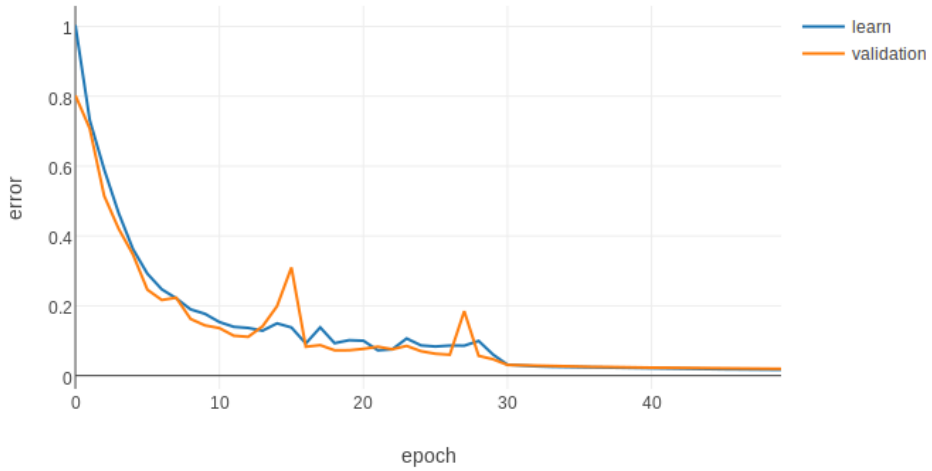


Рисунок 1. *Процесс обучения (рисунок автора)*

Обучение проходило в 50 итераций, после 30 из которых коэффициент обучения был снижен в 10 раз из-за того, что процесс перестал быть стабильным. После чего ошибка на валидации сильно снизилась и сам процесс обучения вошел в насыщение.

Естественный вопрос после проведенного обучения насколько хорошо модель усвоила морфологию? Для ответа на него, прогоним проверочное множество слов через процесс кодирования-декодирования.

Таблица 1

Примеры ошибочного декодирования (составлено автором)

Оригинал	Образ
самохвальствовать	сабовхальствовать
времяисчисление	времоисчисление
по-мальчишечьи	полмальчишечьи

Оригинал	Образ
чертополоховый	червополоховый
самозагружающийся	самизагружающийся
профилактический	просилактический
перекапитализация	перенапитализация
видоизменяемость	виноизменяемость
демуниципализация	дедениципализация
льдообразование	лодообразование
жульничество	жильничество
перекувыркиваться	перекскыркиваться

Опыт показал (таблица 1), что на проверочном множестве модель показала ошибку в 3,7 % всех слов, в которых есть хотя бы один неправильный символ. Некоторые ошибки выглядят так, как будто допущены первоклассником, затрудняющимся запомнить сложное слово. Как будто бы произошла интерференция, в результате которой сеть строит неуместные морфологические аналогии.

Естественен вопрос, способна ли сеть закодировать слова, морфологию которых её в принципе неоткуда было взять, произвольные слова. Для ответа на этот вопрос, прогоним слова, которых нет в словаре. Результат опыта в таблице 2. показывает, что в общем и целом сеть умеет кодировать произвольные слова.

Таблица 2

Кодировка произвольных слов (составлено автором)

Оригинал	Образ
Глокая	глокая
Куздра	куздра
быдланула	быдланула
Штеко	штеко
Бокренка	бокренка
Кагоцел	кагоцел
живой-журнал	живо--журнал
фсбшник	фсбшник
абвгдейка	абвгдейка
громозавр	громозавр

Как можно видеть – не плохой результат, действительно модели не хватило бы емкости запомнить слова, все, что она делает – сворачивает и разворачивает морфологию слова.

Дальнейшая работа

Данная модель показывает, что слово может быть эффективно закодировано семантическим вектором, конечной (не очень большой, до 100 элементов) размерности. Значит можно попытаться данным вектором воспользоваться как адресом для хранения статистики взаимной вероятности двух слов. И таким образом породить модель наподобие word2vec.

ЛИТЕРАТУРА

1. Backpropagation Through time: what is it and how to do it Paul J. Werbose Proceedings Of The Ieee, Vol. 78, No. Io, October 1990.
2. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean, Distributed Representations of Words and Phrases and their Compositionality Neural and Information Processing System (NIPS) (2013).
3. Yoav Goldberg, Omer Levy, word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-EmbeddingMethod. arXiv 2014.
4. Yoav Goldberg, Omer Levy, Neural Word Embedding as Implicit Matrix Factorization. NIPS 2014.
5. Jeffrey Pennington, Richard Socher, Christopher D. Manning, GloVe: Global Vectors for Word Representation. EMNLP, 2014.
6. Пантелеев А. В., Летова Т. А. Методы оптимизации в примерах и задачах. Высшая школа, 2005, 544 с.
7. Себер Дж. Линейный регрессионный анализ. М.: Мир, 1980. – 456 с.
8. Yoav Goldberg, A Primer on Neural Network Models for Natural Language Processing. Journal of Artificial Intelligence Research 57 (2016).
9. Minh-Thang Luong, Richard Socher, Christopher D. Manning, Better Word Representations with Recursive Neural Networks for Morphology. CoNLL-2013, 104, 2013.
10. Tomas Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink Subword Language Modeling With Neural Networks. ICASSP 2012.

Zakirov Marat Anvarovich
«Search portal «Sputnik», Russia, Moscow
E-mail: marat61@gmail.com

Building vocabulary morphology on the basis of recurrent auto encoder

Abstract. The purpose of this paper was to test how well a simple architecture of a recurrent neural network is capable of storing sequences of symbols with respect to the morphology of the Russian language. The model described in this paper is trained to transform a word into a representation of a morphological vector of finite size, from which the word can be decoded back. As a potential application of the model: correction of typos, analysis of morphological proximity, correction of typos, and way of storing words. In the first part, a simple neural network model will be described, which by processing the word symbol by symbol takes its representation to the internal state of the recurrent network, this part of the network is called the encoder. After that, the other part is engaged in decoding this internal representation of the network into a symbolic representation, this part of the model is called the decoder. The model is taught as a single unit, at each iteration, the encoder first works, and then the decoder, and the parts can be used either jointly or separately, depending on the task (for example, often it may be necessary only to use an encoder to get close to morphology of words, for example, to find similar words). In the second part of this work, the results of experiments are shown and the results are summed up.

Keywords: neural network; recurrent autoencoder; morphology; backpropagation through time; morphological synonyms; dictionary; gradient; coder; decoder