Федеральное государственное автономное образовательное учреждение
высшего образования

**«Национальный исследовательский университет
«Высшая школа экономики»**

Факультет компьютерных наук

01.03.02 ОП «Прикладная математика и информатика»

Отчёт о прохождении практики

Студент: Рубачёв Иван Викторович

Группа: БПМИ161

Вид практики: Учебная

Руководитель: Научный Сотрудник, Ст. Преп., _____

Лобачева Екатерина

Куратор: Младший Научный Сотрудник, _____

Надежда Чиркова

Москва, 2018

Содержание

Введение

Большая часть успехов в распознавании речи, анализе текстов и изображений отчасти достигнута благодаря увеличению объемов данных и усложнению моделей (рекуррентных нейросетей в частности). При таком подходе возникают проблемы связанные с использованием полученных моделей с большим числом параметров в условиях ограничения ресурсов (например, на мобильных устройствах). Прунинг – один из методов решения данной проблемы.

Целью данной практики было изучение статьи [10] на тему прунинга рекуррентных нейронных сетей и дальнейшая реализация метода, описанного в ней алгоритма. Перед началом работы над темой практики необходимо было изучить основы рекуррентных нейронных сетей, выполнив лабораторную работу. Более детальная информация о методе прунинга из статьи, реализации базовой модели, реализации прунинга и результатах представлена в основной части отчета.

Основная часть

Обзор статьи «Exploring Sparsity in Recurrent Neural Networks» [10]

Существуют различные подходы к прунингу нейронных сетей (Optimal Brain Damage [2], Optimal Brain Surgeon [5], прунинг весов ниже порогового значения [4]).

Авторы статьи, выбранной для изучения в рамках данной практики предлагают алгоритм прунинга рекуррентных нейроннных сетей. Преимущества предлагаемого метода:

- Вычислительная простота
- Отсутствие необходимости в дообучении модели

Метод заключается в обращении в 0 весов, абсолютное значение которых ниже ε. Пороговое значение монотонно увеличивается в соответствии со следующими формулами:

$$\varepsilon = \begin{cases} \theta \cdot \texttt{diff}/\texttt{freq}, & \texttt{current\_itr} < \texttt{ramp\_itr} \\ \left(\theta \cdot \texttt{diff} + \phi \cdot (\texttt{current\_itr} - \texttt{ramp\_itr} + 1)\right)/\texttt{freq}, & \texttt{current\_itr} \geqslant \texttt{ramp\_itr} \end{cases} \tag{1}$$

Где $\texttt{diff} = \texttt{ramp\_itr} - \texttt{start\_itr} + 1$, а `start_itr`, `ramp_itr`, `end_itr` – итерациии начала прунинга, увеличения скорости прунинга и конца прунинга соответственно. `current_itr` – текущая итерация, `freq` – частота прунинга, ϕ и θ – коэффициенты, определяющие интенсивность удаления весов.

θ определяется по формуле:

$$\theta = \frac{2 \cdot q \cdot \texttt{freq}}{2 \cdot (\texttt{ramp\_itr} - \texttt{start\_itr}) + 3 \cdot (\texttt{end\_itr} - \texttt{ramp\_itr})} \tag{2}$$

Параметры `freq`, `ramp_itr`, `start_itr`, `end_itr` и ϕ подбираются отдельно для каждого слоя сети. Параметр q – девяностый перцентиль абсолютных значений весов обученной без применения прунинга модели. Алгоритм прунинга раз в `freq` итераций убирает из модели веса, абсолютное значение которых меньше ε, и добавляет веса, значения которых больше ε (Вес может стать больше порогового значения при обновлении весов в шаге градиентного спуска, т.к. градиенты для данных весов на этом шаге не изменяются). Таким образом удаление весов в данном алгоритме мягкое – веса могут вернуться в модель

Описание модели анализа тональности текста

Данные и их предобработка

В качестве набора данных для экспериментов был выбран датасет IMDB с рецензиями на фильмы [9]. В датасете собраны положительные и отрицательные отзывы (по эмоциональной окраске). Размеры тренировочной и тестовой выборок 25000 пар. Целевая переменная – два класса (0 – негативная рецензия, 1 – положительная рецензия). На начальной стадии проекта для обработки данных использовался модуль `torchtext`, затем он был заменен решением, написанным самостоятельно. Вся предобработка производится в модуле `utils`. Процесс можно описать пошагово:

- Загрузка текстов. Все тексты и значения целевой переменной сохраняются в python массивах. В случае первой загрузки данных сохраняются как тестовые, так и тренировочные данные.

- Токенизация. На данном этапе из текстов удаляются или заменяются редко встречающиеся служебные символы (например `<br />` заменяется на `"\n"`). Слова написанные в верхнем регистре заменяются на слова в нижнем регистре, при этом перед такими словами добавляется дополнительный токен `t_up`. Также перед повторяющимися символами и словами добавлены специальные токены (с указанием числа повторов). После этого применяется токенизатор из модуля `spacy`. Этот этап обработки производится в параллельном режиме (поэтому работает быстрее, чем `torchtext`)

- Нумерализация. Далее, полученным на предыдущем шаге токенам присваиваются номеара. При этом остаются только 60000 наиболее популярных токенов, из которых также отсеиваются те, которые встречаются реже 3 раз. Полученные данные сохраняются на диск.

Помимо предобработки также написан нестандартный сэмплер (`torch.utils.data.Sampler`), который сортирует тексты в данных таким образом, что в батч попадают тексты приблизительно одной длины. Независимо сортируются срезы размера ($\texttt{batchSize} \times 50$), затем сортируются срезы размера `batchSize`.

При обучении тексты в батче дополняются слева специальным символом (единица после нумерализации) до длины максимального из них.

Архитектура модели

Базовая модель представляет собой однослойную рекуррентную нейронную сеть с архитектурой long-short term memory (LSTM) [6], на вход которой подаются векторные представления токенов (word embedding), на выходе находится один линейный слой. Для регуляризации используется рекуррентный дропаут [3] с вероятностью $p = 0.65$ (одна и та же маска применяется на каждом шаге RNN) после word embedding, обычный дропаут [13] с вероятностью $p = 0.5$ на входе линейного слоя, также используется дропаут в word embedding [3] (зануляются векторы для отдельных слов с вероятностью $p = 0.1$).

Все дальнейшие эксперименты производились с моделью со следующими параметрами:

Layer	Input dim	Output dim
Word Embedding	60002	300
LSTM	300	128
Linear Layer	128	1

Общее число параметров в модели: 18,220,889.

Результаты

После обучения 8 эпох, были получены следующие результаты:

Epoch	Time (s)	Train Loss	Test Loss	Train Acc	Test Acc
1	108.65	35.958	27.779	0.68	0.78
2	111.31	24.467	24.578	0.84	0.84
3	98.24	19.873	20.623	0.87	0.87
4	109.19	17.795	22.941	0.89	0.87
5	106.14	16.155	20.806	0.9	0.87
6	105.55	14.864	21.017	0.91	0.87
7	103.66	13.645	22.307	0.92	0.87
8	104.53	11.989	21.826	0.93	0.86

Здесь точность – доля правильо классифицированных текстов, а функция потерь – бинарная перекрестная энтропия ($\ell(x, y) = \sum l_n, \quad l_n = -w_n \left[t_n \cdot \log \sigma(x_n) + (1 - t_n) \cdot \log(1 - \sigma(x_n))\right]$). Полученные качество соотвестствует модели такого типа из работы [12].

Реализация метода прунинга

Описание алгортма

Данный метод заключается в поддержании набора масок для весов модели. Изначально маски инициализируются единицами. После каждого шага оптимизации веса перемножаются с масками, таким образом 1 в маске означает, что соответствующий вес используется в модели, а 0 – вес не используется. Также с регулярным интервалом маски обновляются, выставлением в 0/1 элементов, соответствующих весам, абсолютное значение которых меньше/больше порогового значения, подсчитанного по формуле 1.

Реализация в pytorch

Вспомогательные классы, добавляющие поддержку прунинга находятся в модуле `pruner.py`. Класс `ModelPruner` инициализирует для каждого параметра модели класс `WeightPruner`, который затем обновляет и применяет маску к своему параметру в соотвестствии с параметрами, которые указаны в конфигурационном файле. Маски применялись ко всем параметрам модели (в том числе матрице слоя представлений и линейному слою), кроме сдвигов (bias).

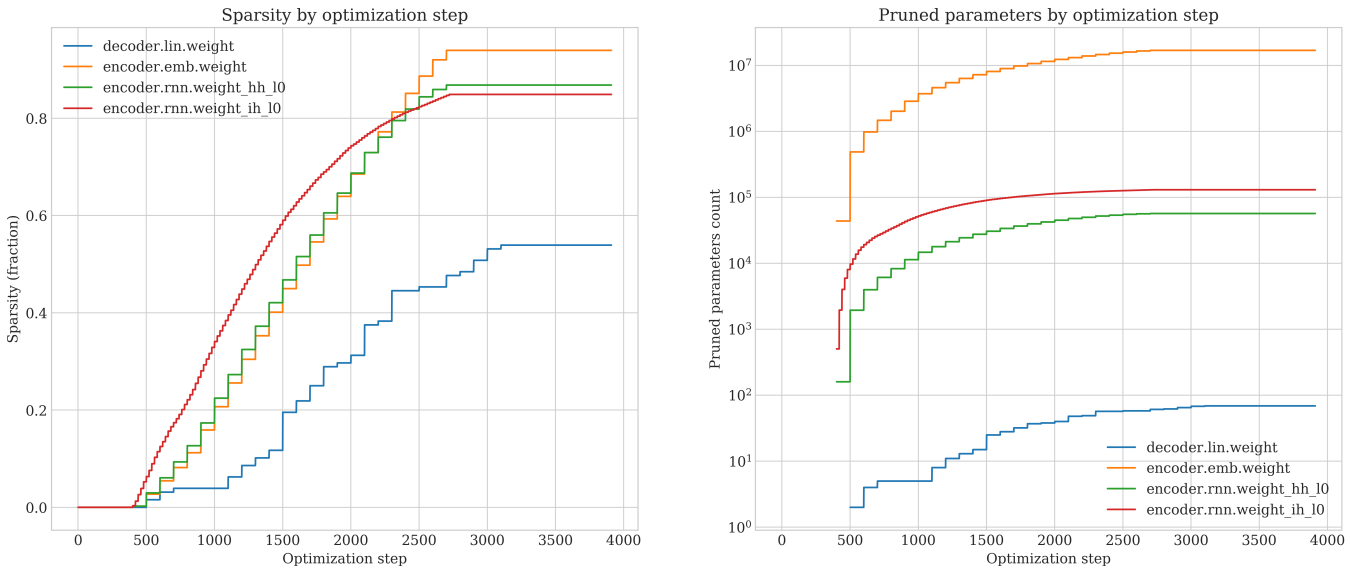
Результаты

При применении прунинга, были получены следующие результаты:

Epoch	Time (s)	Train Loss	Test Loss	Train Acc	Test Acc	Sparsity
1	110.37	35.547	25.801	0.69	0.83	0.0
2	100.53	24.139	23.167	0.84	0.81	0.08
3	103.67	19.789	22.054	0.87	0.84	0.26
4	101.6	16.809	20.582	0.89	0.87	0.45
5	103.79	15.053	20.901	0.91	0.87	0.64
6	107.55	14.253	21.799	0.91	0.87	0.81
7	112.47	14.276	24.024	0.92	0.87	0.94
8	104.8	11.653	22.526	0.93	0.84	0.94
9	100.44	11.072	24.969	0.93	0.86	0.94
10	113.35	11.093	22.774	0.93	0.85	0.94

В данном случае применение прунинга не ухудшило качество на тестовой выборке (Если взять модель после 7 эпохи).

Ниже приведены графики числа удаленных весов для различных параметров модели:



Графики построены для модели с конфигурацией описанной в файле base.yaml. Слева изображена зависимость разреженности весов от итерации обучения для различных параметров модели при прунинге. На графике справа изображена зависимость числа удаленных параметров от итерации обучения.

Заключение

В результате выполнения практики, был реализован и протестирован алгоритм прунинга рекуррентных нейроннных сетей. Код с инструкциями к запуску выложен в открытый доступ: https://github.com/puhsu/pruning Данная практика оказалась очень полезной с образовательной точки зрения. В процессе решения задачи были изучены основные принципы работы рекуррентных нейронных сетей (а также работы с текстовыми данными), библиотека `pytorch`, методы «сжатия» нейронных сетей. Также во время выполнения задания были изучены много источников (статей, opensource проектов, видеолекций). Например, во время изучения основ RNN были найдены блоги [11, 8], при реализации модели были использованы идеи из статьи [7]. К сожалению, далеко не все идеи были опробованы. В качестве следующих шагов по данному проекту было бы интересно провести следующие эксперименты: удалять веса из языковой модели во время обучения на тех же данных (не исключая рецензий без целевой переменной), и затем дообучить полученную модель для задачи классификации. Также стоит попробовать использовать библиотеку [14], в которой есть реализации различных алгоритмов прунинга.

Список используемых источников

[1] Torch Contributors. *PyTorch documentation*. 2018. URL: https://pytorch.org/docs/stable/index.html.

[2] Yann Le Cun, John S. Denker и Sara A. Solla. "Advances in Neural Information Processing Systems 2". В: под ред. David S. Touretzky. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990. Гл. Optimal Brain Damage, с. 598—605. ISBN: 1-55860-100-7. URL: http://dl.acm.org/citation.cfm?id=109230.109298.

[3] Yarin Gal и Zoubin Ghahramani. *A Theoretically Grounded Application of Dropout in Recurrent Neural Networks*. 2015. eprint: arXiv:1512.05287.

[4] Song Han и др. "Learning both Weights and Connections for Efficient Neural Networks". В: *CoRR* abs/1506.02626 (2015). arXiv: 1506.02626. URL: http://arxiv.org/abs/1506.02626.

[5] B. Hassibi, D.G. Stork и G.J. Wolff. "Optimal Brain Surgeon and general network pruning". В: *IEEE International Conference on Neural Networks*. IEEE. DOI: 10.1109/icnn.1993.298572. URL: https://doi.org/10.1109/icnn.1993.298572.

[6] Sepp Hochreiter и Jürgen Schmidhuber. "Long Short-Term Memory". В: *Neural Comput.* 9.8 (нояб. 1997), с. 1735—1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[7] Jeremy Howard и Sebastian Ruder. "Fine-tuned Language Models for Text Classification". В: *CoRR* abs/1801.06146 (2018). arXiv: 1801.06146. URL: http://arxiv.org/abs/1801.06146.

[8] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks*. 2015. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/ (дата обр. 21.05.2015).

[9] Andrew L. Maas и др. "Learning Word Vectors for Sentiment Analysis". В: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, июнь 2011, с. 142—150. URL: http://www.aclweb.org/anthology/P11-1015.

[10] Sharan Narang и др. "Exploring Sparsity in Recurrent Neural Networks". В: *CoRR* abs/1704.05119 (2017). arXiv: 1704.05119. URL: http://arxiv.org/abs/1704.05119.

[11] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (дата обр. 27.08.2015).

[12] *Sentiment Analysis with Deeply Learned Distributed Representations of Variable Length Texts*. 2015. URL: https://cs224d.stanford.edu/reports/HongJames.pdf.

[13] Nitish Srivastava и др. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". В: *Journal of Machine Learning Research* 15 (2014), с. 1929—1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[14] Neta Zmora, Guy Jacob и Gal Novik. *Neural Network Distiller*. Июнь 2018. DOI: 10.5281/zenodo.1297430. URL: https://doi.org/10.5281/zenodo.1297430.