Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»

На правах рукописи

Грачев Артем Михайлович

Методы сжатия рекуррентных нейронных сетей для задач обработки естественного языка

Диссертация на соискание учёной степени
кандидата компьютерных наук НИУ ВШЭ

Научный руководитель:
кандидат технических наук, доцент
Игнатов Дмитрий Игоревич

Москва — 2019

Оглавление

Стр.

Список сокращений и условных обозначений

Здесь собраны некоторые математические обозначения, а также список сокращений, которые использовались в тексте. Необходимо отметить, что область, в которой написана работа, развивается очень быстро и для некоторых терминов ещё нет устоявщихся переводов или же существует несколько их вариантов. Даже одно из основных понятий, “Deep learning”, имеет два перевода: “Глубокое обучение” и “Глубинное обучение”. В данной работе мы преподчитаем использовать первый из них. Некоторые другие термины будем употреблять в оригинале именно исходя из соображений отсутствия подходящих переводов на русский язык. Поэтому помимо некоторых математических обозначений мы приводим расшифровки аббревиатур и объяснения некоторых англоязычных терминов.

$\mathbb{R}$ – множество действительных чисел.

$\mathbb{R}^n$ – множество всех действительных векторов размера n.

$\mathbb{R}^{n \times m}$ — множество всех действительных матриц размера $n \times m$.

$\mathbb{R}^{n_1 \times \ldots, \times n_d}$ — множество всех действительных тензоров размера $n_1 \times \ldots, \times n_d$.

Везде, где не указано иное, мы будем использовать прописные буквы (например, $\mathcal{A}$) для обозначения тензоров размерности > 2, большие буквы (например, A) для обозначения матриц и маленькие буквы (например, a) для обозначения векторов.

RNN — Recurrent Neural Networks, рекуррентные нейронные сети.

LSTM – Long-short Term Memory, сети с так называемой длинно-короткой памятью, ячейки которых имеют 4 вентиля (gate), контролирующих входную, выходную информацию, а также процесс обновления вектора памяти.

GRU – Gated Recurrent Unit, разновидность рекуррентных нейронных сетей, которые имеют более простую структуру в сравнении с LSTM.

В рекуррентных нейронных сетях есть два измерения: время и количество слоев. Если не указано иное, то верхние индексы обозначают измерение времени, нижние индексы — измерение слоев.

Дропаут – Dropout, метод регуляризации нейронных сетей, при котором каждый вес с некоторой заранее фиксированной вероятностью может быть удалён в процессе расчёта нейронной сети.

VD – Variational Dropout, разновидность дропаута, при которой вероятность дропаута для каждого веса подбирается индвидуально и автоматически, с помощью байесовского вывода.

TT – Tensor Train, разложение в тензорный поезд.

SotA — State of the Art, наилучший на текущий момент подход или модель в какой-либо задаче (на каком-либо наборе данных.)

ARD — Automatic Relevance Determination, автоматическое определение значимости — класс методов, которые позволяют автоматически выделять значимые признаки среди их большого числа в задачах регрессии и классификации.

DSVI — Doubly Stochastic Variational Inference, двойной стохастический вариационный вывод — алгоритм для вариационного вывода, использующий стохастические приближения вместо полного расчёта градиента.

ELBO – Evidence Variational Lower Bound – вариационная нижняя граница.

Введение

В последнее десятилетие быстро развивается направление в машинном обучении, называемое глубоким обучением (deep learning) [1; 2], связанное с успешным обучением нейронных сетей и использованием сложных и **глубоких** архитектур. Подходы, связанные с глубоким обучением, вывели сразу несколько направлений в компьютерных науках на новый уровень. В первую очередь эти направления связаны со сложно формализуемыми задачами, такими как обработка изображений, понимание текста, распознавание речи. Во многих областях и конкретных задачах подходы, основанные на методах глубинного обучения, стали признанным индустриальным решением [3]. При этом появляются всё новые задачи [2; 4], а потенциал этих подходов далеко не исчерпан.

Успех нейронных сетей объясняется несколькими факторами. Во-первых, это теоретическая возможность аппроксимировать произвольные функции с помощью нейронных сетей [5; 6]. Во-вторых, это простота, с которой масштабируется процесс обучения. Алгоритм обучения сети с помощью обратного распространения ошибки позволяет легко считать градиент функции потерь по параметрам модели и делать это параллельно. В-третьих, рост вычислительных мощностей. Масштабируемость процессов обучения ничего бы не стоила, если бы не возможность масштабировать все вычисления. Появление высокомощных графических карт позволяет обучать нейронные сети дёшево и быстро и практически любому специалисту.

При этом, у нейронных сетей есть и недостатки. Например, несмотря на то, что с теоретической точки зрения нейронные сети являются универсальным аппроксиматором, на текущий момент не существует конструктивного способа построения нейронных сетей для каких-то конкретных задач. То есть нет методов, которые заранее позволяют сказать нейронная сеть какой топологии (сколько слоёв, какого размера каждый слой) потребуется для решения той или иной задачи. На практике специалистам приходится подбирать гиперпараметры, отвечающие за топологию сети, в процессе обучения.

Другой важный недостаток становится очевидным, когда появляется потребность использовать нейронные сети на каких-либо устройствах: на мобильных телефонах, телевизорах, пылесосах или даже холодильниках. Это могут быть задачи обработки звуковых сигналов, распознавания объектов, моделирование подсказок в мобильной клавиатуре или электронной почте. Обычно такие устройства имеют не очень мощные процессоры и не очень большое количество памяти. В то же время нейронные сети требовательны и к тому, и к другому.

Этот второй недостаток делает актуальным задачи сжатия нейронных сетей с целью их последующего размещения на различных устройствах[1][2][3].

В данном диссертационном исследовании мы рассматриваем проблему сжатия нейронных сетей в контексте класса задач, связанного с моделированием естественного языка, и работаем, в основном, с рекуррентными нейронными сетями. Это позволяет нам выделить некоторые характерные особенности этого класса задач и адаптировать наши методы именно для него. При этом большая часть описываемых методов применимы и в более широком контексте, то есть для произвольных нейронных сетей.

Базовая постановка задачи моделирования языка заключается в предсказании следующего слова по предыдущей цепочке слов или, другими словами, по левому контексту. Самые первые методы для решения этой задачи были основаны на прямом сохранении всех возможных вариантов. В момент предсказания выбирался наиболее вероятный вариант или делалось сэмплирование из сохраненного распределения. У такого подхода есть проблемы, связанные с учетом дальних зависимостей и необходимостью хранить все варианты. Так, например, все возможные цепочки длины 4 для некоторых словарей уже невозможно хранить в памяти компьютеров.

Рекуррентные нейронные сети отчасти способны моделировать дальние зависимости и не требуют прямого запоминания всех вариантов. Но они всё

[1] https://os.mbed.com/blog/entry/uTensor-and-Tensor-Flow-Announcement/
[2] https://towardsdatascience.com/why-machine-learning-on-the-edge-92fac32105e6
[3] https://petewarden.com/2017/05/08/running-tensorflow-graphs-on-microcontrollers/

ещё слишком большого размера для их повсеместного использования на устройствах, которые, как уже упоминалось выше, обычно очень требовательны к памяти и быстродействию. Особенностью построения языковых моделей является необходимость хранить словарь из нескольких тысяч слов (например, около 20 тысяч слов покрывают лишь 80% русского языка). Кроме того, часто необходимо решать задачу прогнозирования следующего слова или, выражаясь терминами машинного обучения, решать задачу классификации на несколько тысяч (или даже десятков тысяч) классов. Поэтому, несмотря на то, что нейронные сети занимают гораздо меньше места, чем простые статистические модели, задача сжатия является актуальной для их широкого применения.

Современные методы сжатия включают в себя как простые методы: например, прореживание матриц или квантизация представления чисел, так и более сложные, основанные на разложениях матриц или применении байесовских методов. Среди работ, основанных на прореживании матриц и квантизации, можно отметить [7;8]. Методы, основанные на разложениях матриц, делятся на несколько категорий. Во-первых, можно раскладывать матрицы в низкоранговые разложения [9], тем самым достигая некоторого сжатия. Во-вторых, можно использовать матрицы специального типа, например URNN [10] или матрицы Теплица [9]. Наконец, можно использовать разложения слоёв в Tensor Train [11]. Вариационный дропаут первоначально использовали для настройки индивидуальной степени дропаута для каждого нейрона [12]. В [13] авторы показали, что вариационный дропаут также может занулять некоторые нейроны, тем самым давая сжатие. Если накладывать априорное распределние на отдельные веса, то мы получаем разреженные матрицы, что, как и в случае с обычным разреживанием, не очень удобно, поэтому в последующей работе предлагается делать структурное сжатие, накладывая априорное распределение сразу на столбцы и строчки [14]. В данном диссертационном исследовании мы детально проанализировали текущие методы сжатия нейронных сетей и предложили ряд новых алгоритмов, решающих эту задачу более эффективно.

Целью данного диссертационного исследования является создание новых алгоритмов сжатия нейронных сетей для применения их в задаче обработки

естественного языка. Предлагаемые алгоритмы должны решать задачу сжатия более эффективно с точки зрения использования ресурсов клиентских устройств, например, мобильных телефонов.

Для достижения цели в рамках работы предполагается исследовать различные методы сжатия нейронных сетей и предложить на их основе нейросетевые архитектуры, имеющие сравнительно малый размер и эффективно решающие прикладные задачи. В качестве основных **задач** исследования можно выделить:

1. Эмпирический анализ алгоритмов сжатия нейронных сетей, основанных на техниках разреживания и квантизации, разложении матриц, байесовских методах.
2. Построение новых и модфикация существующих алгоритмов сжатия нейронных сетей с учётом специфики задачи обработки естественного языка и типичных архитектур рекуррентных нейронных сетей.
3. Адаптация разработанных алгоритмов, отвечающих заданным характеристикам, для применения их на различных устройствах, например, мобильных телефонах.

Сформулируем **основные результаты исследования и положения, выносимые на защиту**:

1. Построение алгоритмов сжатия скрытых слоев нейронных сетей для задачи моделирования языка с помощью методов матричных разложений.
2. Эффективное решение задачи классификации для большого числа классов (10–30 тыс.) с использованием методов матричных разложений для входного и выходного слоев нейронной сети.
3. Адаптация байесовских методов разреживания для рекуррентных нейронных сетей в задаче моделирования языка.
4. Портирование эффективной реализации рекуррентных нейронных сетей на мобильные устройства и лабораторное тестирование.

Научная новизна:

1. Впервые было применено TT-разложение для сжатия рекуррентных нейронных сетей в задаче моделирования языка. Также было подробно исследовано применение низкорангового матричного разложения для данной задачи и класса моделей на основе рекуррентных нейросетей.
2. Впервые был применён алгоритм автоматического определения значимых признаков с помощью двойного стохастического вариационного вывода (DSVI-ARD) в задаче моделирования языка.
3. Было выполнено портирование сжатых моделей на мобильные устройства. На тот момент (согласно публикациям на ведущих конференциях) это была первая (задокументированная в [15]) реализация нейронных сетей (тем более сжатых) на мобильном GPU.
4. Была предложена общая схема сжатия для рекуррентных нейронных сетей в задаче моделирования языка.

Научная и практическая значимость работы заключается в разработке новых методов сжатия рекуррентных нейронных сетей с учётом особенностей, возникающих в задаче моделирования языка, и переноса этих моделей на устройства с мобильными GPU.

Степень достоверности полученных результатов обеспечивается математически корректными моделями сжатия, а также большим количеством экспериментов с различными архитектурами и наборами данных. Результаты сопоставимы с работами других авторов.

Публикации и апробация работы.

Основные результаты по теме диссертации изложены в печатных изданиях [15–18].

Публикации повышенного уровня.

– Artem M. Grachev, Dmitry I. Ignatov and Andrey V. Savchenko. Neural Networks Compression for Language Modeling. — *Pattern Recognition and Machine Intelligence - 7th International Conference, PReMI 2017, Kolkata, India, December5-8, 2017, Proceedings. – 2017. – Pp. 351–35.*
– Elena Andreeva, Dmitry I. Ignatov, Artem M. Grachev and Andrey V. Savchenko. Extraction of Visual Features for Recommendation of Products

via Deep Learning. – *Analysis of Images, Social Networks and Texts – 7th International Conference, AIST 2018, Moscow, Russia, July 5-7, 2018, Revised Selected Papers. – 2018. – Pp. 201–210.*

– Artem M. Grachev, Dmitry I. Ignatov and Andrey V. Savchenko. Compression of Recurrent Neural Networks for Efficient Language Modeling. — *Applied Soft Computing – 2019. – Vol. 79. – Pp. 354 – 362.*

Прочие публикации.

– Maxim Kodryan*, Artem Grachev*, Dmitry Ignatov and Dmitry Vetrov. Efficient Language Modeling with Automatic Relevance Determination in Recurrent Neural Networks — *ACL 2019, Proceedings of the 4th Workshop on Representation Learning for NLP, August 2, 2019, Florence, Italy*

Работы [15–17] цитируются в системе научного цитирования **Scopus**. Статья [17] в журнале Applied Soft Computing [17] входит в Q1 **Scopus** и **Web of Science**. Работа [18] опубликована на воркшопе при конференции CORE A* и входит в **ACL Anthology**.

Апробация работы.

Результаты диссертационного исследования докладывались на:

– 7th International Conference on Pattern Recognition and Machine Intelligence (06.12.2017, Калькутта, Индия). Доклад «Neural network compression for language modeling».

– Аспирантский семинар факультета компьютерных наук НИУ ВШЭ (2017, Москва, Россия). Доклад «Neural network compression for language modeling» («Сжатие нейронных сетей для задачи языкового моделирования»)

– Конференция «Технологии машинного обучения» (октябрь 2018, Москва, Россия). Доклад «Neural Networks for mobile devices» («Нейронные сети для мобильных устройств»)

– Семинар «Автоматическая обработка и анализ текстов» НИУ ВШЭ (20.04.2019, Москва, Россия). Доклад «Методы сжатия рекуррентных нейронных сетей для задач обработки естественного языка».

– Семинар в компании Samsung R&D Russia (2017, Москва, Россия). Доклад «Методы сжатия рекуррентных нейронных сетей для задач обработки естественного языка».
– Семинар в компании Samsung R&D Russia (2019, Москва, Россия). Доклад «DSVI-ARD сжатие в нейронных сетях».

Личный вклад. Все результаты получены автором лично, кроме результатов, полученных совместно с Е. Андреевой в статье [16] и М. Кодряном в статье [18].

Объем и структура работы. Диссертация состоит из введения, четырёх глав и заключения. В первой главе мы подробно рассматриваем задачу моделирования языка и применение рекуррентных нейронных сетей для её решения. Обсуждаются методы обучения рекуррентных нейронных сетей. Во второй главе мы приводим обзор современных методов сжатия нейронных сетей, а также анализируются особенности задачи моделирования языка, связанные с высокоразмерными входными и выходными данными. Излагаются особенности простейших методов сжатия (прунинга и квантизации) для рекуррентных нейронных сетей. В третьей главе мы описываем различные методы, связанные с матричными разложениями. Мы проделали ряд экспериментов с низкоранговыми разложениями и с Tensor Train разложениями для сжатия рекуррентных нейронных сетей. Кроме того мы представили общую схему сжатия рекуррентных нейронных сетей для задачи моделирования языка с помощью всех описанных методов. Четвёртая глава посвящена применению байесовских методов для сжатия нейронных сетей. Мы адаптировали алгоритм автоматического подбора гиперпарметров для рекуррентных нейронных сетей.

Полный объём диссертации составляет 105 страниц с 17 рисунками и 9 таблицами. Список литературы содержит 84 наименования.

Глава 1. Рекуррентные нейронные сети для задачи моделирования языка

В этой главе рассмотрена задача моделирования языка и её характерные особенности. Мы обсудим применение этой задачи в реальной жизни и способы оценивания качества различных моделей для её решения. Большое внимание будет уделено решению этой задачи с помощью рекуррентных нейронных сетей, а также способу их обучения и различным практическим аспектам ускорения процесса обучения. В конце главы также приведены описания основных датасетов, которые использовались для экспериментов.

1.1 Задача моделирования языка

Рассмотрим задачу моделирования языка в общем виде. Пусть $\mathbb{V}$ – это словарь некоторого фиксированного языка. Требуется оценить вероятность предложения или последовательности слов $(w^1, \ldots, w^T)$ в данном корпусе.

$$\mathsf{P}\left(w^1, \ldots, w^T\right) = \mathsf{P}\left(w^1, \ldots, w^{T-1}\right) \mathsf{P}\left(w^T | w^1, \ldots, w_{T-1}\right) = \\ = \prod_{t=1}^{T} \mathsf{P}\left(w^t | w^1, \ldots, w^{t-1}\right) \quad (1.1)$$

С формальной точки зрения такую задачу можно считать из класса обучения без учителя. В то же время встречаются и другие определения, в которых она определяется как задача с частичным обучением или задача с самообучением (semi-supervision task, self-supervision task). Эта задача, несмотря на простую формулировку, имеет фундаментальное значение для многих практических приложений, таких как умная клавиатура в мобильном телефоне, автоответы на

письма и СМС, исправление опечаток в тексте. Кроме того, сейчас набирает популярность использование предобученных моделей языка (см. ULMFIT [19], ELMO [20], BERT [21]) в других теоретических и прикладных задачах, например, word-sense induction [22], задачи классификации текста [19] и машинный перевод [23]. На Рисунке 1.1 представлен пример практического приложения: подсказки в мобильной клавиатуре. Текст, который использовался в качестве тестового представлен ниже:

```
Albert Einstein was a German-born physicist who developed
the general theory of relativity. He is considered one of
the most influential physicists of the 20th century.
```

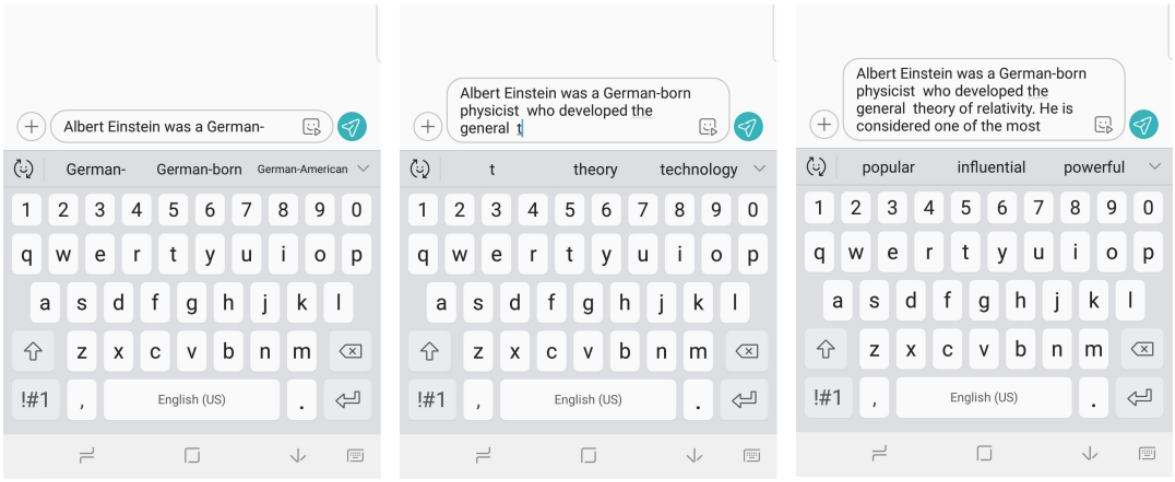


Рисунок 1.1 — Пример индустриального применения задачи моделирования языка. Мобильная клавиатура.

Для того чтобы использовать такую модель напрямую необходимо вычислять вероятность $\mathsf{P}\left(w^t|w^1,\ldots,w^{t-1}\right)$ из формулы 1.1. В общем случае, то есть для произвольных t эта вероятность невычислима. Поэтому обычно данную вероятность аппроксимируют с помощью $\mathsf{P}\left(w^t|w^{t-N},\ldots,w^{t-1}\right)$ — вероятности следующего слова при фиксированной длине предыдущего контекста N. Это приводит нас к N-граммным моделям [24; 25], которые представляют из себя частотные распределения всех цепочек длины N, встречающихся в обучающем тексте.

Опишем N-граммные модели более подробно. Пусть $C(w^1, \ldots, w^t)$ — это количество раз, которое последовательность $w^1, \ldots, w^t$ встречается в корпусе. Тогда мы естественным образом можем оценить вероятность встретить слово w^t при условии того, что мы встретили $w^1, \ldots, w^{t-1}$ по формуле:

$$\mathsf{P}\left(w^1, \ldots, w^t\right) = \frac{C(w^1, \ldots, w^t)}{C(w^1, \ldots, w^{t-1})} \tag{1.2}$$

Для того чтобы понять, откуда у N-граммных моделей возникают проблемы, по аналогии с [26] проанализируем эффективность N-граммных моделей для датасета PTB (Penn Treebank) [27]. Этот датасет имеет размер словаря $|\mathbb{V}| = 10^4$, размер корпуса порядка $Size = 10^6$ (более точно 929590). Для $n = 2$ мы встретим в лучшем случае не более $\frac{size}{|\mathbb{V}|^2} = 1\%$ всех цепочек, что, естественно, недостаточно для построения полноценной модели. С увеличением n ситуация будет только ухудшаться. Другая проблема, которая здесь возникает, связана с тем, что эти цепочки необходимо хранить в памяти компьютера. И, например, цепочки длины $n = 4$ будут требовать хранения по меньшей мере $|\mathbb{V}|^4 = 10^{16}$, а это, даже по самым скромным подсчётам, больше, чем может поместиться в память современных компьютеров.

Конечно же на практике при работе с N-граммными моделями используются дополнительные эвристики, позволяющие сгладить описанные выше проблемы. Поэтому эта группа методов долгое время была популярной в задаче моделирования языка, но в последнее время их почти полностью вытеснили нейронные сети [28; 29]. Можно сказать, что использование рекуррентных нейронных сетей (recurrent neural networks, RNN) стало новой вехой в развитии статистического моделирования языка. Они гораздо эффективнее учитывают долгосрочные зависимости в тексте и занимают гораздо меньше места. Хотя, конечно, использование их в современных устройствах всё ещё связано с некоторыми трудностями, так как хочется иметь более компактные и быстрые модели. Мы обсудим эти проблемы далее, но сначала опишем подробнее, как устроены рекуррентные нейронные сети и как они применяются для задачи моделирования языка.

.

1.2 Рекуррентные нейронные сети для задачи моделирования языка

Рассмотрим произвольную рекуррентную нейронную сеть с L скрытыми рекуррентными слоями и N шагами по времени. Нулевой слой такой сети (или слой эмбеддингов) является отображением $f : \mathbb{V} \to \mathbb{R}^k$ из индекса слова в словаре в k-мерный вектор эмбеддингов, являющийся численным представлением этого слова, и определяется матрицей $W_0 \in \mathbb{R}^{|\mathbb{V}| \times k}$. Обозначим выход этого слоя x_0^t. Далее пусть для $t \in \{1, \ldots, N\}$, $\ell \in \{1, \ldots, L\}$, x_ℓ^t будет выходным вектором для слоя ℓ в момент времени t. Теперь мы можем записать уравнение для каждого слоя в следующем виде:

$$z_\ell^t = W_\ell x_{\ell-1}^t + U_\ell x_\ell^{t-1} + b_l \tag{1.3}$$

$$x_\ell^t = \tanh(z_\ell^t), \tag{1.4}$$

где W_ℓ и U_ℓ — это матрицы весов для слоя ℓ и $\tanh$ — это гиперболический тангенс, который традиционно используется как функция активации в рекуррентных нейронных сетях. Выход такой нейронной сети в момент времени t будет задан следующим образом:

$$y^t = \operatorname{softmax}\left[W_{L+1} x_L^t + b_{L+1}\right]. \tag{1.5}$$

Функция софтмакс $\mathrm{softmax}(\cdot)$ является отображением из $\mathbb{R}^k \to \mathbb{R}^k$ и для произвольного вектора $z \in \mathbb{R}^k$ её можно определить следующим образом:

$$\mathrm{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)}, \tag{1.6}$$

где $i = 1, \ldots, k$. Эта функция является обощением логистической функции для многомерного случая. Можно видеть, что она переводит действительные числа в распределение вероятностей, так как компоненты вектора теперь неотрицательные, лежат в отрезке $[0, 1]$ и суммируются в единицу. Тогда интересующее нас распределение вероятностей слов по словарю из уравнения 1.1 можно выразить следующим образом:

$$\mathsf{P}\left(w_t | w_{t-N}, \ldots, w_{t-1}\right) = y^t. \tag{1.7}$$

В зависимости от типа конкретного приложения нам иногда требуется получать либо вырожденные распределения, где есть чёткий максимум, либо, наоборот, сглаженные. В функции $\mathrm{softmax}(\cdot)$ для такого рода можно использовать температуру. Заметим, что если все компоненты вектора-аргумента z для функции $\mathrm{softmax}(\cdot)$ умножить на один и тот же гиперпараметр τ, то ранжирование вероятностей в выходном софрмакс-векторе останется тем же, а вот значения по модулю изменятся и станут либо более вырожденными, либо более сглаженными.

Рассмотрим пример. Возьмём вектор $z = (1, 5, 6)$, выход софтмакса для него будет равен $y = (0.005, 0.268, 0.727)$. Если мы умножим на $\tau = 1/10$ все компоненты вектора z, то получим $z/10 = (1/10, 5/10, 6/10)$ и выходной вектор $y_{0.1} = (0.24, 0.36, 0.4)$. Видно, что выходное распределение сгладилось. Если же $\tau = 10$, то выходной вектор будет равен $y_{10} = (1.9 \cdot 10^{-22}, 4.5 \cdot 10^{-5}, 0.99995)$, то есть мы получим практически вырожденный вектор. Сэмплирование в соответствии с вероятностями, которые мы получаем из софтмакса при $\tau \longrightarrow +\infty$, эквивалентно взятию argmax по компонентам вектора.

Самое важное отличие рекуррентных нейронных сетей от обычных полносвязных заключается в том, каким образом мы можем моделировать прошлые моменты времени. В полносвязной сети нам для этого потребовалось бы подать на вход в сеть всю предыдущую историю, при этом у неё не было бы возможности как-то автоматически выучить порядок событий, то есть его бы потребовалось дополнительно размечать. Рекуррентная нейронная сеть выучивает некоторое представление истории автоматически. Наши точки в датасете теперь имеют ещё одно измерение – время t. Во время работы рекуррентной нейронной сети каждый слой получает информацию с предыдущего слоя (аналогично с полносвязной сетью) и с предыдущего момента времени. Именно этот смысл заложен в формуле 1.3.

В то время как N-грамные модели даже с не очень большими N из-за комбинаторного взрыва требуют большого количества памяти, рекуррентные нейронные сети могут гораздо эффективнее выучивать представления слов и последовательностей без непосредственного запоминания всех слов в контексте. Прежде чем перейти к вопросу обучения нейронных сетей рассмотрим, как измеряется качество языковых моделей.

1.3 Оценка качества языковых моделей

1.3.1 Перплексия

Для оценки качества языковых моделей обычно используется **перплексия**.

Определение 1.3.1. *Перплексия (perplexity) — это величина, которая измеряет насколько хорошо распределение, которое выдает вероятностная модель*

предсказывает цепочку слов:

$$PP(w) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{\mathsf{P}(w_i|w_1, \ldots, w_{i-1})}} = 2^{-\frac{1}{N}\sum \log \mathsf{P}(w_i|w_1,\ldots,w_{i-1})} \tag{1.8}$$

Из определения видно, что перплексия выражается через кросс-энтропию между моделью и тестовыми данными и может быть получена как экспонента от средней пословной энтропии некоторого набора тестовых данных. В [29] приводится ряд аргументов, почему для оценки качества языковых моделей следует использовать именно значение перплексии.

Один из таких аргументов состоит в том, что перплексия имеет некоторый физический смысл. Интуитивно она выражает меру запутанности языковой модели. Другими словами, можно сказать, что если перплексия языковой модели на некотором корпусе равна p, то это эквивалентно тому, что в среднем модель выбирает равномерно один из p вариантов слова на каждой итерации. Другим аргументом является тот факт, что для многих реальных приложений (таких как распознавание речи) есть строгая корелляция между переплексией используемой языковой модели и тем, насколько хорошо работает вся система в целом. Это показано, например, в [30].

В целом, кросс-энтропия является довольно популярным функционалом для такого рода задач, хотя у неё есть и недостатки. Одним из важных недостатков для применения в реальных приложениях является то, что измерение перплексии обычно подразумевает “идеальную” историю, то есть, что предыдущий контекст был распознан/смоделирован правильно. Это не всегда верно в некоторых приложениях, например, в автоматическом распознавании речи.

Тем не менее, мы будем предполагать, что для обучения RNN достаточно оптимизировать кросс-энтропию и на этом основывать наши дальнейшие выкладки там, где это необходимо.

1.3.2 Точность

Стандартной метрикой для задачи классификации является точность (accuracy). Её же можно использовать и для проверки качества языковых моделей. Для задачи классификации с произвольным количеством классов её можно определить следующим образом:

$$accuracy = \frac{Correct}{All}, \tag{1.9}$$

где $Correct$ – количество верно классифицированных объектов и All – количество всех объектов. К сожалению, ввиду того, что обычно в языковом моделировании требуется предсказать несколько тысяч, а то и десятков тысяч классов, эта метрика менее показательна при оценке языковых моделей. Большая часть слов приходится на вспомогательные слова: "the" , "do" , "is" , "are" (для английского языка). Мы будем использовать эту метрику в некоторых случаях как дополнительный показатель качества.

1.3.3 Дивергенция Кульбака-Лейблера

Еще одна важная функция потерь, которая будет использоваться в наших экспериментах — это KL-дивергенция. Она тесно связана с кросс-энтропией, и сейчас мы обсудим ее свойства.

Определение 1.3.2. *Пусть P и Q два вероятностных распределения, у которых соответственно существуют плотности p и q, определенные на одном носителе X. Тогда KL-дивергенция для этих двух распределений вычисляется по следующей формуле:*

$$KL(P\|Q) = \int_X p(x) \log \frac{p(x)}{q(x)}\, dx \tag{1.10}$$

Можно заметить, что если в определении разбить логарифм частного на разность логарифмов и, соответственно, разнести интегралы, то получим:

$$KL(P\|Q) = \int_X p(x) \log p(x)\ dx - \int_X p(x) \log q(x)\ dx \tag{1.11}$$

Из такой записи становится понятен смысл KL-дивергенции. Видно, что она разбивается на энтропию распределения P и кросс-энтропию P и Q со знаком минус. Другими словами, она показывает, насколько распределение P надо изменить, чтобы получить распределение Q. В этом смысле KL-дивергенция используется для того, чтобы измерять расстояние между распределениями. Но несмотря на то, что часто говорят о сходстве KL-дивергенции с расстоянием между распределениями, строго говоря, эта метрика расстоянием не является, так как не обладает симметричностью и для нее не выполняется неравенство треугольника.

Тем не менее, на практике это довольно удобный функционал и он широко используется в байесовских методах. Также можно заметить, что кросс-энтропия в случае one-hot распределения (которое используется в языковом моделировании) эквивалентна KL-дивергенции $F^t = KL(z_0^t\|y^t)$.

1.4 Методы оптимизации

На сегодняшний день в большинстве задач машинного обучения для оптимизации используются методы, основанные на расчёте градиента. Пусть $F(X, Y, \theta)$ – функционал ошибки, зависящий от всех описаний объекта X, всех истинных меток Y и от вектора параметров нашей модели θ. Алгоритм градиентного спуска заключается в том, чтобы на каждой итерации вычислять значение вектора градиента функционала, посчитанного по всей выборке, и сдвигать значение параметров на этот вектор. Такой алгоритм неприменим для

Algorithm 1 Алгоритм SGD

Входные данные: Обучающая выборка X, метки истинных классов Y, начальные значения параметров θ_0, скорость обучения γ
repeat
$i := i + 1$
случайно выбираем мини-батч (x^m, y^m) размера k из (X, Y).
$g_i := \nabla F(x^m, y^m, \theta_{i-1})$ вычисляем градиент на i-ой итерации
$\theta_i := \theta_{i-1} - \gamma \cdot g_i$
until не выполнен критерий сходимости

выборок большого размера, так как расчёт градиента для функционала, посчитанного на всей выборке будет занимать очень много времени.

Вместо этого используют стохастические варианты градиентного спуска (Stochastic Gradient Descent, SGD), где на каждом шаге мы аппроксимируем полный функционал ошибки его приближением, посчитанным на части выборки или даже на одном её элементе:

$$F(x^m, y^m, \theta) \approx F(X, Y, \theta), \tag{1.12}$$

где подразумевается, что $m \ll M$ – полного объёма выборки. Мы приводим формальное описание стохастического градиентного спуска в Алгоритме 1.

Помимо SGD на текущий момент существует множество его различных вариаций, которые стараются запоминать предыдущие шаги и использовать эту информацию для того, чтобы корректировать направление спуска и уменьшать дисперсию. Среди таких вариаций можно перечислить RMSprop [31], AdaGrad [32], AdaDelta [33], ADAM [34]. В наших исследованиях мы помимо SGD также использовали ADAM, поэтому также представляем его листинг в Алгоритме 2.

Как видим, основное отличие от SGD здесь состоит в том, что теперь при расчете финального выражения для градиента мы учитываем первый и второй моменты m и v. В статье [34] представлен анализ сходимости такого алгоритма.

Algorithm 2 Алгоритм ADAM [34]

Входные данные: Обучающая выборка X, метки истинных классов Y, начальные значения параметров θ_0, гиперпараметры, отвечающие за скорость обучения γ, β_1, β_1, регуляризятор ϵ

repeat

$i := i + 1$

случайно выбираем мини-батч (x^m, y^m) размера k из (X, Y).

$g_i := \nabla F(x^m, y^m, \theta_{i-1})$ вычисляем градиент на i-ой итерации

$m_i := \beta_1 \cdot m_{i-1} + (1 - \beta_1) \cdot g_i$ обновляем первый момент (смещеный)

$v_i := \beta_2 \cdot v_{i-1} + (1 - \beta_2) \cdot g_i^2$ обновляем второй момент (смещеный)

$\hat{m}_i := \frac{m_i}{1-\beta_1^i}$ учитываем смещение для первого момента

$\hat{v}_i := \frac{v_i}{1-\beta_2^i}$ учитываем смещение для второго момента

$\theta_i := \theta_{i-1} - \frac{\gamma \cdot \hat{m}_i}{\sqrt{\hat{v}_i} + \epsilon}$

until не выполнен критерий сходимости

1.5 Алгоритм обратного распространения ошибки через время

Рассмотрим теперь схему обучения рекуррентных нейронных сетей. На сегодняшний день самым популярным является алгоритм обратного распространения ошибки через время [35–37]. Это модификация алгоритма обратного распространения ошибки [38;39], который применяется для обычных полносвязных нейронных сетей. Как уже обсуждалось, в рекуррентных нейронных сетях сигнал проходит не только в следующий слой, но и в текущий слой, в следующий момент времени. Поэтому делается развертка сети ещё и по времени. Алгоритм распространения ошибки (как обычный, так и через время) основаны на методе стохастичего градиентного спуска, который был рассмотрен в предыдещем секции 1.4. Для его применения необходимо уметь вычислять градиенты весов модели. Рассмотрим как происходит расчёт градиентов в рекуррентной нейронной сети.

Обозначим за y_t^* истинный класс следующего слова в момент времени t и $y^* = (y_1^*, \ldots, y_T^*)$. Пусть у нас есть некоторый функционал ошибки $F_t(y_t, y_t^*, \theta)$

(например, кросс-энтропия) в момент времени t. Тогда суммарную ошибку мы определим как сумму по всем t:

$$F(y, y^*, \theta) = \sum_{t=1}^{N} F_t(y_t, y_t^*, \theta), \tag{1.13}$$

где $y = (y_1, \ldots, y_T)$ — вектор выходов нейронной сети, а вектор $\theta = (W_0, W_1, V_1, b_1 \ldots, W_L, V_L, b_L, W_{L+1})$ – вектор всех её параметров.

Посчитаем производную этого функционала по параметрам θ. Для удобства дальнейшего изложения аналогично с [37] введём понятие мгновенной частной производной:

Определение 1.5.1. *Мгновенная частная производная* $\frac{\partial^+ z_l^t}{\partial \theta}$ *– это частная производная* z_l^t *по* θ *в предположении, что* x_l^{t-1} *не зависит от* θ*.*

Так, например, для уравнения $z_\ell^t = W_\ell x_{\ell-1}^t + U_\ell x_\ell^{t-1} + b_l$ любая строка матрицы мгновенной частной производной $\partial^+(z_\ell^t) \;/\; \partial U$ равен x_ℓ^{t-1}. Используя эти обозначения, имеем:

$$\frac{\partial F}{\partial \theta} = \sum_{t=1}^{N} \frac{\partial F_t}{\partial \theta} \tag{1.14}$$

$$\frac{\partial F_t}{\partial \theta} = \sum_{k=1}^{N} \left(\frac{\partial F}{\partial z^t} \frac{\partial z^t}{\partial z^k} \frac{\partial^+ z^k}{\partial \theta} \right) \tag{1.15}$$

$$\frac{\partial z^t}{\partial z^k} = \prod_{i=t}^{k} \frac{\partial z^i}{\partial z^{i-1}} = \prod_{i=t}^{k} U_\ell^T \operatorname{diag}(x^{i-1}) \tag{1.16}$$

Проанализируем теперь поведение этих частных производных. Заметим, что все компоненты $\frac{\partial F_t}{\partial \theta}$ в уравнении 1.14 имеют схожий вид (с точностью до индекса) и, таким образом, поведение каждого из них определяет поведение всей суммы в целом.

В свою очередь, каждая компонента $\frac{\partial F_t}{\partial \theta}$ – это тоже сумма 1.15, компоненты которой показывают, как вносятся в градиент изменения по времени. Можно заметить, что $\frac{\partial F}{\partial z^t} \frac{\partial z^t}{z^k} \frac{\partial^+ z^k}{\partial \theta}$ отвечает за то, как сильно параметры θ на шаге k

Algorithm 3 Алгоритм обратного распространения ошибки через время

Входные данные: Текущие значения параметров θ, вычисленная функция $F(y_t, y_t^*, \theta)$ для$t \in \{1, \ldots, N\}$
Инициализация вектора градиентов параметров
$\nabla\ \theta\ = (\nabla W_0, \nabla W_1, \nabla V_1, \nabla b_1 \ldots, \nabla W_L, \nabla V_L, \nabla b_L, \nabla W_{L+1})$
$\nabla W_{L+1} := \frac{\partial F_t}{\partial W^{L+1}}$
for ℓ **from** L **downto** 1 **do**
  for t **from** T **downto** 1 **do**
    $g_{z_l^t} := g_{z_\ell^t} \frac{\partial^+ z^{t+1}}{z^t} + \frac{\partial F_t}{\partial z^t}$
    $\nabla W_\ell := \nabla W_\ell + g_{z_t^l} \frac{\partial^+ z^t}{\partial W_\ell}$
    $\nabla U_\ell := \nabla U_\ell + g_{z_t^l} \frac{\partial^+ z_l^t}{\partial U_\ell}$
    $\nabla b_\ell := \nabla b_\ell + g_{z_t^l} \frac{\partial^+ z_l^t}{\partial b_\ell}$
  end for
end for
$\nabla W_0 := \frac{\partial z^1}{z^0} \frac{\partial x^0}{\partial W_0}$
Выходные данные: вектор градиентов параметров
$\nabla\theta = (\nabla W_0, \nabla W_1, \nabla V_1, \nabla b_1 \ldots, \nabla W_L, \nabla V_L, \nabla b_L, \nabla W_{L+1})$

влияют на функционал качества на шаге $t > k$. То есть компонента $\frac{\partial z^t}{\partial z^k}$ как бы переносит ошибку по времени. Алгоритм 3 представляет собой подробное описание расчёта всех производных параметров нейронной сети (по всем слоям ℓ и по всем моментам времени t).

В [40] показано, что при таком расчёте возникает проблема *взрывающихся градиентов*, когда их норма сильно возрастает во время обучения. Вместе с тем существует противоположная проблема *затухающих градиентов*, когда компонента градиента, отвечающая за долгосрочные зависимости, стремится к нулю экспоненциально быстро с ростом t. Это делает невозможным выучивание каких-то долгосрочных зависимостей в данных. Более точно это явление можно сформулировать в виде следующего утверждения [37]:

Утверждение 1. *Пусть f — функция активации в рекуррентной нейронной сети и справедливо, что $|f'(z)|$ ограничена, $\|\operatorname{diag}(f'(z))\| \leq \gamma \in \mathbb{R}$ и λ_1 – это максимальное собственное значение матрицы U_ℓ. Тогда условие $\lambda_1 < \frac{1}{\gamma}$ является достаточным для возникновения проблемы затухающего градиента.*

Доказательство. В уравнении 1.16 норма каждого множителя в произведении может быть ограничена сверху следующим образом:

$$\left\|\frac{\partial z^{i+1}}{\partial z^i}\right\| \leq \|U^T\| \|\operatorname{diag}(x^i)\| = \|U^T\| \|\operatorname{diag}(f(z^i))\| < \frac{1}{\gamma}\gamma = 1 \tag{1.17}$$

Это верно $\forall i$. Отсюда следует, что $\forall i\ \exists \eta \in \mathbb{R}$ такая, что

$$\left\|\frac{\partial z^{i+1}}{\partial z^i}\right\| \leq \eta < 1. \tag{1.18}$$

Из этого вытекает

$$\left| \frac{\partial F_t}{\partial z_t} \left(\prod_{i=k}^{N-1} \frac{\partial z^{i+1}}{\partial z^i} \right) \right\| \leq \eta^{N-k} \left| \frac{\partial F_t}{\partial z_t} \right\|. \tag{1.19}$$

Так как $\eta < 1$, то отсюда следует, что влияние слагаемого, отвечающего за дальние зависимости, убывает экспоненциально с ростом $N - k$. □

Мы показали, что $\lambda_1 < \frac{1}{\gamma}$ является достаточным условием возникновения проблемы *затухающих градиентов*. Отсюда же сразу следует, что условие $\lambda > \frac{1}{\gamma}$ является необходимым условием возникновения проблемы *взрывающихся градиентов*. Более подробный анализ этой проблемы см. [40].

Проблемы затухающих и взрывающихся градиентов не позволяют RNN выучивать по-настоящему длинные зависимости [40–43].

1.6 Разновидности рекуррентных нейронных сетей

Для решения проблемы затухающего градиента, описанной в предыдущем разделе, в настоящее время широко используются различные вариации рекуррентных сетей. Наиболее популярные из них — это рекуррентные нейронная сеть с ячейкой LSTM (Long-short term memory, длинно-короткая память), предложенной в [41] и GRU (Gated Recurrent Unit), предложенной в [44]. Сначала

опишем один слой LSTM сети согласно слегка изменённой и более распространённой формуле, которую можно встретить, например в [45]:

$$i_\ell^t = \sigma\left[W_l^i x_{l-1}^t + U_l^i x_l^{t-1} + b_l^i\right] \quad \text{input gate} \quad (1.20)$$

$$f_\ell^t = \sigma\left[W_l^f x_{l-1}^t + U_l^f x_l^{t-1} + b_l^f\right] \quad \text{forget gate} \quad (1.21)$$

$$c_\ell^t = f_l^t \odot c_l^{t-1} + i_l^t \tanh\left[W_l^c x_{l-1}^t + U_l^c x_l^{t-1} + b_l^c\right] \quad \text{cell state} \quad (1.22)$$

$$o_\ell^t = \sigma\left[W_l^o x_{\ell-1}^t + U_l^o x_l^{t-1} + b_l^o\right] \quad \text{output gate} \quad (1.23)$$

$$x_\ell^t = o_\ell^t \cdot \tanh[c_l^t], \quad (1.24)$$

где $t \in \{1, \ldots, N\}$, $\ell \in \{1, \ldots, L\}$, c_ℓ^t — это вектор памяти для слоя ℓ и временного шага t, а $\odot$ обозначает операцию покомпонентного умножения. Выход сети задается той же формулой (1.5), что и для RNN.

В отличие от обычного рекуррентного слоя, который просто суммирует входной вектор и рекуррентный вектор с предыдущего слоя, каждая ячейка LSTM хранит вектор памяти c_ℓ^t. Соответственно, **output gate** из уравнения (1.23) отвечает за то, какой объём памяти будет выдан наружу в уравнении (1.24). Вектор памяти обновляется в уравнении (1.22), а коэффициенты, с которыми будут взвешиваться новый сигнал и память с прошлого шага, вычисляются в уравнениях (1.20-1.21).

Если обычная рекуррентная ячейка (1.3-1.3) перезаписывает своё содержимое в каждый момент времени, то LSTM ячейка способна решать, перезаписывать ли текущую память и в каком объёме. Интуитивно это означает, что если LSTM ячейка увидит какую-нибудь важную информацию во входной последовательности на раннем этапе, она сможет сохранить эту информацию на длинной дистанции и, таким образом, выявить долгосрочные зависимости. Пример LSTM нейрона изображен на Рис.1.2a.

Как мы уже упоминали, GRU ячейка была предложена в [44], но сейчас более распространёнными являются формулы, немного отличающиеся от

оригинальных и предложенные в статье [46]. В соответствии с ней и выпишем уравнения сети с такой ячейкой для одного слоя ℓ в момент времени t:

$$z_l^t = \sigma\left(W_l^z x_{l-1}^t + U_l^z x_l^{t-1}\right) \quad \text{update gate} \quad (1.25)$$

$$r_l^t = \sigma\left(W_l^r x_{l-1}^t + U_l^r x_{t-1}\right) \quad \text{reset gate} \quad (1.26)$$

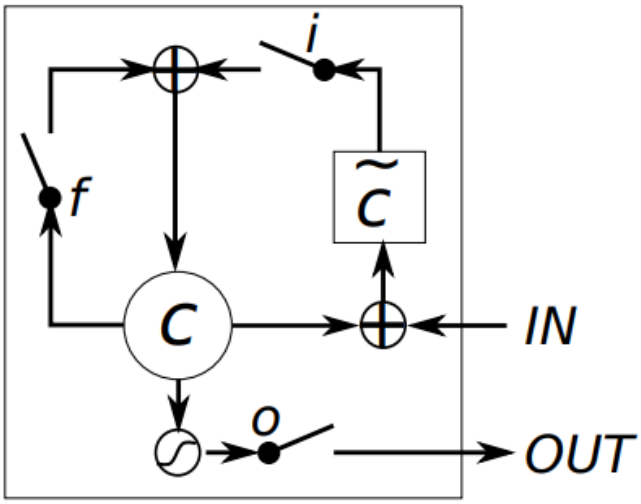
$$\tilde{x}_l^t = \tanh\left(W_l^h x_{l-1}^t + U_l^h\left(r_l^t \cdot x_l^{t-1}\right)\right) \quad \text{proposal output} \quad (1.27)$$

$$x_t^l = (1 - z_l^t) \odot x_{l-1}^t + z_l^t \odot \tilde{x}_l^t \quad \text{final output} \quad (1.28)$$

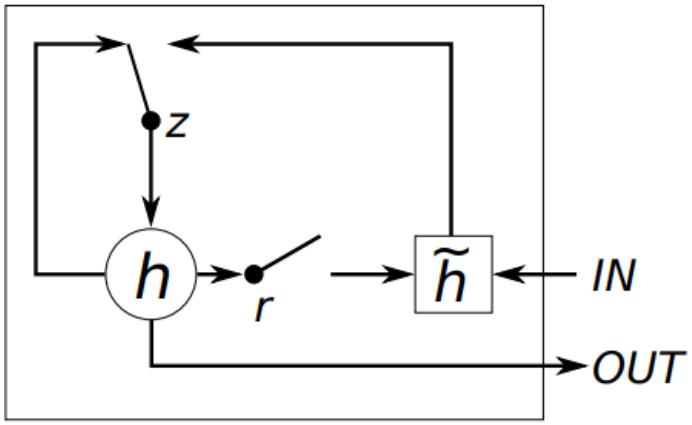
Выход ячейки GRU в уравнении (1.28), x_t^l – это результат линейной интерполяция между предыдущим выходом x_l^{t-1} и предложенным выходом $\tilde{x}_l^t$ на этом шаге. Соответственно, **update gate** z_l^t здесь выступает в роли взвешивающего коэффициента между ними. Предложенный выход 1.27 считается очень похожим на обычный рекуррентный слой (1.3-1.3), а **reset gate** очень похож на **update gate**. Схему GRU можно найти на Рис.1.2б.

Легко заметить, что ячейки LSTM и GRU похожи между собой. Наиболее явная особенность, которая их объединяет, это наличие дополнительной компоненты в их обновлении с момента времени t на $t+1$ и которая отсутствует в классической рекуррентной ячейке. Классическая рекуррентная ячейка всегда заменяет содержимое ячейки на новое значение, высчитанное на основе нового входного вектора и старого скрытого состояния. С другой стороны, LSTM и GRU способны и учитывать предыдущий контекст, и сохранять текущее состояние (1.22, 1.28).,

Эта дополнительная компонента имеет два преимущества. Во-первых, теперь каждая такая ячейка может помнить о существовании какого-либо важного или специального признака во входном потоке большое число шагов назад. Если **forget gate** LSTM или **update gate** решит, что какой-либо признак достаточно важен, то он не будет перезаписан, а будет сохранен как есть. Во-вторых, и возможно это даже более важно, в дополнение к этому ячейки сохраняют способность передавать информацию на большое число шагов. И это в итоге



а)



б)

Рисунок 1.2 — Схематичное представление из [47] для LSTM и GRU: (a) LSTM, (b) GRU.

позволяет применять алгоритм обратного распространения ошибки без такого масштабного затухания (или взрыва) градиентов, какое бывает у обычных рекуррентных нейронных сетей. Также эти две ячейки имеют и некоторые отличия. В LSTM, в отличие от GRU присутствует возможность контролировать выдачу содержимого памяти другим нейронам. То есть, не вся та память, которая присутствует внутри одного нейрона, видна следующим нейронам, и за это как раз отвечает **output gate**. А ячейка GRU отдаёт всё содержимое своей памяти. Другое различие — это положение **input gate** или отвечающего ему **reset gate**. Ячейка LSTM вычисляет новое содержимое памяти без какого-либо

отдельного контроля количества информации, которое пришло с предыдущего этапа. Кроме того, объём новой памяти, который будет добавлен, рассчитывается независимо от **forget gate**, что может привести к переизбытку или уменьшению важности какой-либо компоненты. С другой стороны, GRU контролирует информационный поток из предыдущего момента времени во время расчёта предложенного выхода, но не контролирует отдельно объём, который этот кандидат добавит (потому что этот контроль разделён с **update gate**).

В целом, вопрос о том, какая из двух ячеек работает лучше, достаточно дискуссионный [47]. Но зато есть несколько работ [46; 47], в которых эмпирически показывается, что обе ячейки LSTM и GRU работают лучше, чем RNN на задачах моделирования последовательностей.

1.7 Практические приёмы для обучения нейронных сетей

В данном разделе описаны различные приёмы, которые на практике улучшают процесс обучения нейронных сетей. На самом деле их значительно больше, но здесь собраны самые релевантные для обучения именно рекуррентных нейронных сетей.

Одно из наиболее проблемных мест в обучении рекуррентных нейронных сетей – это проблема взрывающихся градиентов. Простейшим, но очень эффективным решением этой проблемы является их ограничение по модулю (**gradient clipping**). Есть разные способы реализации этой эвристики [29; 37]. Например, можно ограничивать градиенты покомпонентно. Другой вариант заключается в том, чтобы зафиксировать некоторый порог C и далее если на очередной итерации норма вектора градиента $\|\nabla\theta\| > C$, то

$$\nabla\theta_c = C \cdot \frac{\nabla\theta}{\|\nabla\theta\|} \tag{1.29}$$

Так как все параметры градиента репараметризуются совместно, то второй способ является более обоснованным, так как он сохраняет общее направление градиента и просто уменьшает размер шага.

В качестве регуляризации в машинном обучении традиционно используются так называемые ℓ_2- и ℓ_1-регуляризации, идея которых заключается в том, чтобы добавить в функционал ошибки слагаемое, которое будет ограничивать соответственно либо ℓ_2, либо ℓ_1 норму весов. Для рекуррентных нейронных сетей этот способ не очень хорошо работает на практике; здесь используют другой распространенный метод, который называется дропаутом (**dropout**). Его суть заключается в том, что для каждого слоя задаётся некоторая вероятность p. На каждой итерации с этой вероятностью веса будут обнуляться и выпадать из процесса обучения на данном шаге.

Ещё одно затруднение, которое часто встречается при обучении нейронных сетей и в машинном обучении в целом – это подбор гиперпараметров. В нашем исследовании эта тема является ещё более актуальной, так как дополнительно приходится подбирать гиперпараметры, которые отвечают за уровень сжатия. В итоге мы имеем большое, многомерное пространство гиперпараметров. Делать полный перебор по такому пространству даже с очень грубой сеткой довольно затратно по вычислительным ресурсам.

К счастью, есть альтернатива полному перебору параметров, которая также легко программируется и гораздо более удобна в использовании. Это процедура случайного поиска [48], устроенную следующим образом. Мы предполагаем, что каждый гиперпараметр распределён по какому-либо закону, то есть он задаётся каким-либо распределением, например: Бернулли, равномерное, лог-равномерное. Далее на каждой итерации подбора гиперпараметров мы сэмплируем этот параметр из его распределения. Таким образом, мы не делаем полный перебор по всему пространству. Эффективность случайного подбора гиперпараметров достигается за счёт того, что обычно модель сильно зависит от небольшой группы гиперпараметров и более-менее равномерно зависит от всех остальных. Делая полный перебор, мы можем так и не попасть в зону “хороших” значений для этой важной группы гиперпараметров из-за слишком большой

размерности пространства. С другой стороны, делая сэмплирование каждый раз, то есть, меняя все гиперпараметры, мы очень вероятно попадем в зону “хороших” значений для важной группы гиперпараметров. Иллюстративный пример можно видеть на Рис. 1.3 [4]. Здесь предполагается, что в нашей модели два гиперпараметра (обычно их гораздо больше), но при этом наша модель сильно зависит от одного из них и слабо зависит от другого. Видно, что случайный поиск здесь с большой вероятностью попадёт в максимум по важному параметру, в то время как обычный перебор может этот максимум проскочить.

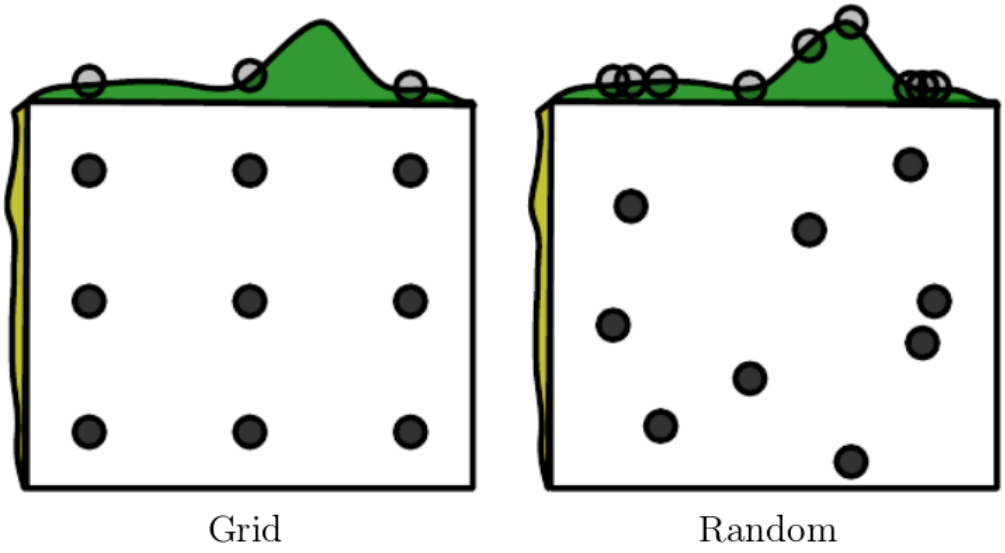


Рисунок 1.3 — Иллюстративный пример [4], поясняющий идею случайного поиска гиперпараметров.

1.8 Описание датасетов

Для тестирования наших методов мы использовали два датасета: PTB (Penn Treebank) [27] и Wikitext-2 [49]. Основные статистики по количеству слов представлены в таблице 1.1. Здесь OoV (out of vocabulary) означает процент слов, которые не попали в словарь и были заменены на специальные

Таблица 1.1

Статистика по датасетам PTB и Wikitext-2

	PTB			Wikitext-2		
	Train	**Valid**	**Test**	**Train**	**Valid**	**Test**
Размер словаря	10 000			33 278		
OoV	4.8%			2.6%		
Статьи	-	-	-	600	60	60
Слова	929 590	73 761	82 431	2 088 628	217 646	245 569

<unk> — неизвестные слова, которые моделируются как один класс. В таблице 1.2 представлены значения перплексии и времени работы для LSTM сетей, которые использовались как основные базовые модели.

Таблица 1.2

Значения перплексии и времени работы для базовых моделей

	Модель	**Размер, Mb**	**Кол-во параметров, M**	**Тестовая перплексия**	**Время работы, ms**
PTB Benchmarks	LSTM 200-200	18.6	4.64	117.659	9.63
	LSTM 650-650	79.1	19.7	82.07	16.13
	LSTM 1500-1500	264.1	66.02	78.29	45.47

Глава 2. Введение в методы сжатия. Особенности сжатия рекуррентных нейронных сетей для моделирования языка

2.1 Анализ числа параметров в нейронной сети

Подходы к моделированию языка, основанные на нейронных сетях, эффективны и широко используются [26; 29], но все еще требуют большого количества параметров. Причём в отличие от обычных полносвязных сетей, у рекуррентных слоёв на каждый слой приходится несколько матриц. Для обычных RNN (1.3–1.4) это две матрицы, для GRU (1.25–1.28) шесть, для LSTM (1.20–1.24) восемь. Дальнейшие расчёты будем проводить для LSTM. Для восьми матриц получаем $8 \times k \times k$ параметров, где k размер скрытого слоя. Кроме того, обычно в задаче моделирования языка мы хотим использовать слова (а не символы) в качестве фундаментальной единицы входа и выхода. Это требование приводит нас к большим размерам входного и выходного слоя. Входной слой или слой эмбеддингов переводит номер слова в словаре $\mathbb{V}$ в числовой вектор. Выходной слой — аффинное преобразование из скрытого представления в номер слова в словаре, для которого далее обычно используется функция софтмакс. Здесь надо отметить, что размер словаря $|\mathbb{V}|$ — величина порядка тысяч или даже десятка тысяч, поэтому эти две матрицы (входного и выходного слоя) будут иметь большой вклад. Складывая всё это, получаем, что количество параметров во всей сети с L слоями равно

$$n_{total} = 8Lk^2 + 2|\mathbb{V}|k. \tag{2.1}$$

2.1.1 Проблема входного и выходного слоя

Проанализируем, насколько велик вклад каждого слагаемого в (2.1). В датасете PTB (Penn Tree Bank) [50] размер словаря $|\mathbb{V}| = 10{,}000$. Рассмотрим LSTM сеть с двумя скрытыми слоями, $k = 650$ нейронов в каждом. Тогда получаем, что каждый LSTM слой имеет восемь матриц размером 650×650, то есть $650 \times 650 \times 8 \times 2 = 6.76$ млн параметров. Выходной слой такой сети имеет $650 \times 10000 = 6.5$ млн параметров. Похожие вычисления для LSTM сети с размерами скрытых слоев $k = 1500$ дают нам 36 млн параметров суммарно в скрытом слое и 15 млн параметров для выходного слоя. Таким образом, мы можем видеть, что выходной (софтмакс) слой может занимать до одной трети от всей сети. Заметим, что эмбеддинги сети занимают такой же размер, если не используются методы наподобие совместного использования входных и выходных весов ("tied softmax") [51], про которые мы поговорим ниже. Более подробные расчёты приведены в таблице 2.1, где подсчитано какое количество параметров занимают LSTM сети и их части. Данные модели использовались как основные базовые модели [52], на которых потом тестировались алгоритмы сжатия.

Проблема высокоразмерных выходов в задаче моделирования языка широко обсуждается в литературе. Например, в [53] рассматривается проблема редких или неизвестных слов. Дело в том, что если включать все редкие слова в словарь, то он получится очень большим, но на редких словах модель всё равно будет работать плохо, так как встретит их в обучающей выборке небольшое количество раз и выучит их плохое представление. Кроме того, это увеличивает размер нейронной сети. Если же выкидывать все редкие слова, то мы получим большой процент OoV слов, от чего сильно будет страдать качество модели.

Большая вычислительная сложность и размер выходного слоя обсуждаются в [54;55]. В [55] разрабатывается идея иерархического софтмакса. Авторы показывают, что возможно вычислять софтмакс со скоростью $O(\sqrt{|\mathbb{V}|})$. Bengio et al. [54] предлагают ускорять вычисление софтмакса с помощью сэмплирования подмножества словаря на каждой итерации в процессе вычисления выходов

сети. Мы также обращаемся к проблеме высокоразмерного выхода в софтмакс слое. Мы используем низкоранговое разложение и Tensor Train разложение для уменьшения размера матрицы софтмакса и ускорения ее расчета.

Таблица 2.1

Количество параметров в базовых моделях.

Датасет	**Размер словаря**	**Размер скрытого слоя**	**Кол-во парам. (млн) для двух слоёв**	**Кол-во парам. (млн) выходного слоя и ембеддингов**	**Кол-во парам. всей сети (млн)**
PTB	10000	200	0.64 (14%)	4 (86%)	4.64
PTB	10000	650	6.76 (34%)	13 (66%)	19.76
PTB	10000	1500	36 (55%)	30 (45%)	66
WikiText-2	33278	650	6.76 (14%)	43.26 (86%)	50.02
WikiText-2	33278	1500	36 (27%)	98.83 (73%)	135.83

2.1.2 Использование одной матрицы для входного и выходного слоя

В этом разделе мы опишем ещё один метод, который применяется для сокращения числа параметров и улучшения качества языковых моделей, основанных на нейронных сетях. Идея его заключается в том, чтобы переиспользовать матрицу эмбеддингов при расчёте выходного софтмакса (“weight sharing”, “tied weights”). Такой подход вдохновлён тем, что и матрица эмбеддингов, и выходная матрица представляют собой некоторое отображение – в одном случае из словаря в какое-то векторное представление, в другом случае наоборот. Кажется, что поскольку слова в предложении идут друг за другом, они действительно должны иметь как минимум похожие эмбеддинги.

Опишем эту идею чуть более формально по аналогии с [56]. Введём дополнительную функцию потерь в задачу моделирования языка

$$y^t = \text{softmax}\left(\frac{W_{L+1}x^t_{L+1}}{\tau}\right) \tag{2.2}$$

$$F^t_{aug} = KL(\tilde{y}^t \| y^t) \tag{2.3}$$

$$F^t_{tot} = F^t + \alpha F^t_{aug} \tag{2.4}$$

Здесь α это некоторый гиперпараметр, который определяет значимость дополнительной функции потерь, y^t — это распределение вероятностей по словарю, которое эквивалентно тому, которое встречается в обычной задаче моделирования языка, но здесь оно делится на дополнительный параметр температуры, который отвечает за то, насколько распределение будет сглажено или, наоборот, вытянуто (см. Главу 1).

Далее, мы вводим дополнительное распределение $\tilde{y}^t$, которое удовлетворяет $\mathsf{E}\tilde{y}^t = \mathsf{E}y^t$. Идея здесь заключается в том, чтобы минимизировать расстояние между предсказанным распределением y^t и более аккуратно оцененным изначальным распределением.

Пусть z^t_0 – входной вектор в момент времени t. Тогда для получения распределения $\tilde{y}$ предлагается использовать следующие соотношения:

$$u^t = W_0 z^t \tag{2.5}$$

$$\tilde{y} = \text{softmax}\left(W_0^T u^t\right), \tag{2.6}$$

где W_0^T обозначает транспонированную матрицу W_0 – матрицу эмбеддингов.

Покажем теперь почему оптимальна именно такая формула для расчёта $\tilde{y}$ Предположим, что мы обучаем некоторую языковую модель. При обучении используем $F_{tot} = F_{aug}$ и параметр температуры τ достаточно высок. Также мы считаем, что в процессе обучения мы достигаем нулевого значения функции потерь.

Обозначим w_i как i-ю колонку матрицы W_0. Используя разложение Тейлора мы можем представить $\exp(x) \approx 1 + x$ и получить следующее равнество

по столбцам:

$$\tilde{y}_i^t = \frac{\exp(\langle u_t, w_i \rangle)/\tau}{\sum_{j \in V} \langle u_t, w_j \rangle)/\tau} \approx \frac{1 + \langle u_t, w_i \rangle/\tau}{|V| + \sum_{j \in V} \langle u_t, w_j \rangle/\tau} \tag{2.7}$$

Можно упростить далее, предположив, что $\mathsf{E}_{j \in V} \langle u_t, w_j \rangle = 0$. Тогда получаем:

$$\tilde{y}_i^t \approx \frac{1 + \langle u_t, w_{0,i} \rangle/\tau}{|V|} \tag{2.8}$$

По аналогии с этим уравнением для y^t из 2.2 мы получаем

$$y_i^t \approx \frac{1 + \langle x_{L+1}, w_{L+1,i} \rangle/\tau}{|V|} \tag{2.9}$$

Далее, для производной F_{aug}^t имеем:

$$\frac{\partial F_{aug}^t}{\partial \left(W_{L+1} x_{L+1}^t \right)} = \frac{1}{\tau} \left(y^t - \tilde{y}^t \right). \tag{2.10}$$

Подставляя в это уравнение $\tilde{y}$ из 2.8 и y^t из 2.9 получаем:

$$\frac{\partial F_{aug}^t}{\partial \left(W_{L+1} x_{L+1}^t \right)_i} = \frac{1}{\tau^2 |V|} \left(W_{L+1} x_{L+1}^t - W_0^T u^t \right)_i \text{ при } \tau \to \inf, \tag{2.11}$$

Так как мы считаем, что при обучении достигается нулевое значение функции потерь, то мы получаем, что $W_{L+1} x_{L+1}^t = W_0^T u^t$ верно для каждого примера из обучающей выборки. Введем некоторую матрицу A, такую что $W_{L+1} = W_0^T A$ и перепишем это равенство в виде:

$$W_{L+1} x_{L+1}^t = W_0^T A x_{L+1}^t = W_0^T h \tag{2.12}$$

Другими словами, в данной постановке, если мы будем переиспользовать матрицу в выходном софтмакс слое, мы как бы заставляем сеть выучить некоторое преобразование, $h_t \to Au$, которое мы как раз и хотели бы получить.

Ограничения, введённые нами, далеко не всегда выполняются на практике. Более того, на самом деле этого сложно добиться. Но, по крайне мере,

данный раздел позволяет дать интуицию для понимания происходящего. Имея ввиду эту мотивацию, мы можем попытаться использовать матрицу эмбеддингов W_0 в качестве выходной матрицы W_{L+1}, то есть положить $W_0^T = W_{L+1}$. Во-первых, это сразу же даёт сокращение числа параметров, которые используются в входном и выходном слое в два раза. Во-вторых, на практике оказывается, что такая модель лучше обучается. Скорей всего это связано с тем, что мы выучиваем более качественные представления для редких слов, так как теперь их эмбеддинги как бы обучаются в два раза чаще.

2.2 Основные подходы к сжатию нейронных сетей

В целом, известно несколько подходов к сжатию нейронных сетей. Верхнеуровнево их можно разделить на две категории: методы, основанные на разреженных вычислениях, и методы, которые используют различные матричные разложения или свойства матриц весов.

Первая группа методов включает в себя прунинг и квантизацию и изначально широко применялась в компьютерном зрении. В одной из первых работ на эту тему [7] было показано, что прунинг делает возможным удаление большого числа весов без потери в качестве работы сети. Для таких популярных сетей, как LeNet, AlexNet, VGGNet есть результаты, что прунингом можно удалить 67% свёрточных слоёв и до 90% полносвязных слоёв. Более того, более высокого уровеня сжатия можно достичь, объединяя прунинг с вычислениями со смешанной точность (8-битные, 16-битные) [8].

Можно сказать, что методы вариационного дропаута – это разновидность прунинга, которая имеет под собой некоторую математическую базу. Изначально вариационный дропаут был введен [12], как метод, который позволяет автоматически настраивать вероятности дропаута разных весов в нейронной сети. В [13; 14] авторы адаптировали эту технику для сжатия нейронных сетей.

Применив специальную параметризацию (см. подробнее Гл. 4), можно позволить вероятности дропаута быть единице, что эквивалентно полному удалению такого нейрона из нейронной сети. Так как при таком подходе все параметры, в том числе вероятность дропаута, подбираются автоматически в процессе обучения, то такой подход к прунингу кажется более математически обоснованным. Однако он страдает от тех же недостатков, что и обычный прунинг: необходимость работать с разреженными матрицами. Поэтому в [14; 57] идеи вариационного дропаута получили развитие, и были разработаны методы, которые позволяют удалять сразу целые строки или столбцы матриц. Этот подход называется структурным байесовским прунингом.

Другая ветвь методов сжатия – это методы, основанные на подходах связанных с разложением матриц или с использованием различных матриц, обладающих специальными свойствами. Например, в статье [10] предлагается новый вид ячейки RNN, основанный на потенциально более компактных унитарных матрицах. Называется она Unitary Evolution Recurrent Neural Networks. Такая RNN ячейка показала себя лучше, например, в задаче сложения двух чисел и в задаче копирования последовательности. В [58] авторы применили так называемое FastFood преобразование для полносвязных и сверточных слоёв. Они достигли до 90% сжатия в терминах количества параметров, которое необходимо хранить на диске. Как уже упоминалось различные матричные разложения также можно отнести к этом классу методов сжатия. Это могут быть как обычные низкоранговые разложения, так и более сложные, например, разложение в тензорный поезд (Tensor Train, TT) [11; 59–61].

Однако, подходы, основанные на TT разложениях, не были изучены в моделировании естественного языка до сих пор. При этом именно здесь мы сталкиваемся с проблемой высокоразмерных входных и выходных данных и, следовательно, с очень широкими возможностями настраивать TT разложение. В целом, вторая группа методов, основанная на сжатии матриц, позволяет получить сокращение в размере моделей и все ещё иметь удобные неразреженные матрицы для использования в моделях.

Большинство описанных методов могут быть использованны для сжатия RNN. Методы, основанные на матричных разложениях, были в основном использованы в задаче распознавания речи [9; 60; 62]. Например, использование матриц вида Теплицевых в [9] дало около 40% сжатия для RNN в задаче голосового поиска. Сжатие слоя эмбеддингов рассматривалось в статье [63].

Одна из наиболее важных и сложных проблем, которая в том числе порождает большой размер языковых моделей – это высокоразмерные выходной и входной слой, которые вызваны большим размером словаря. Эта проблема широко обсуждается в литературе. Так, например, в статье [53] рассматриваются трудности, связанные с запоминанием редких или неизвестных слов. Большая вычислительная сложность и большой размер софтмакс слоя обсуждается в [54;55]. Morin et al [55] разработал подход, называемый иерархический софтмакс. Они показали, что можно использовать $O(\sqrt{|\mathbb{V}|})$ параметров для такого способа вычисления софтмакса. Bengio et al. [54] предлагают метод для увеличения скорости вычисления софтмакса с использованием сэмплирования некоторого подмножества слов из словаря на каждой итерации обучения.

Методы прунинга и вариационного дропаута были применены для сжатия RNN в задаче моделирования языка [57; 64]. Однако, результы в [57] получены на посимвольной языковой модели, что делает затруднительным их сравнение классическими результатами Zaremba et al [52]. Более того, по этой же причине здесь не рассматривается проблема высокоразмерных входных и выходных данных. Сравнение с результатами, полученными в [64], мы обсуждаем в секциях 3.6.3 и 4.5.2.

Таким образом, кажется, что до сих пор не было систематически изучено сжатие RNN для моделирования языка, которое обходило бы трудности, связанные с высокоразмерными входным и выходным слоями, а также сделало бы возможным достижение одновременно маленького размера нейронной сети по памяти, быстродействия и эффективности для таких моделей и дало бы возможность использовать их в мобильных приложениях, которые работают в реальном времени, без необходимости обмениваться информацией с сервером.

2.3 Прунинг и квантизация как базовые методы сжатия

В данной секции мы рассмотрим некоторые базовые методы, которые могут быть использованы для сжатия нейронных сетей. Они были успешно применены для обработки аудио [8] и изображений [65]. Однако они не очень хорошо изучены для задачи моделирования языка [15].

Прунинг это метод снижения числа параметров в нейронной сети путем удаления весов, которые приблизительно равны нулю. На Рис. 2.1 (вверху) изображено распределение значений весов софтмакс слоя для классической рекуррентной нейронной сети. Видно, что значения большинства весов по модулю близки к нулю. Можно предположить, что вклад, который эти веса вносят в финальное предсказание мал, и удалить эти параметры. Таким образом, мы можем перебирая различные уровни отсечения удалять из нейронной сети все веса, модуль которых не превосходит нужного порога. На Рис. 2.1 (внизу) представлено распределение весов этого же софмакс слоя после удаления 90% самых маленьких по модулю и дообучения оставшихся.

Квантизация – это метод уменьшения размера нейронной сети в оперативной памяти. Он заключается в том, что все множество весов сети разбивается на 256 интервалов. И каждому интервалу ставится в соответствие 8-битный integer. Таким образом, если изначально наше число хранилось в памяти как 32-битный float, то теперь оно хранится в 8-битном integer'е и занимает в 4 раза меньше места.

Прунинг и квантизация имеют общие недостатки связанные с тем, что они не поддерживают обучение нейронной сети с нуля. Более того, для применения их на практике приходится затрачивать дополнительные усилия. Так, например, чтобы получить ускорение нейронной сети после применения прунинга, необходимо эффективно поддерживать разреженные матричные вычисления на аппаратном уровне. В случае квантизации мы получаем модель, сохраненную в 8-битном представлении, но для вычислений мы всё ещё должны будем

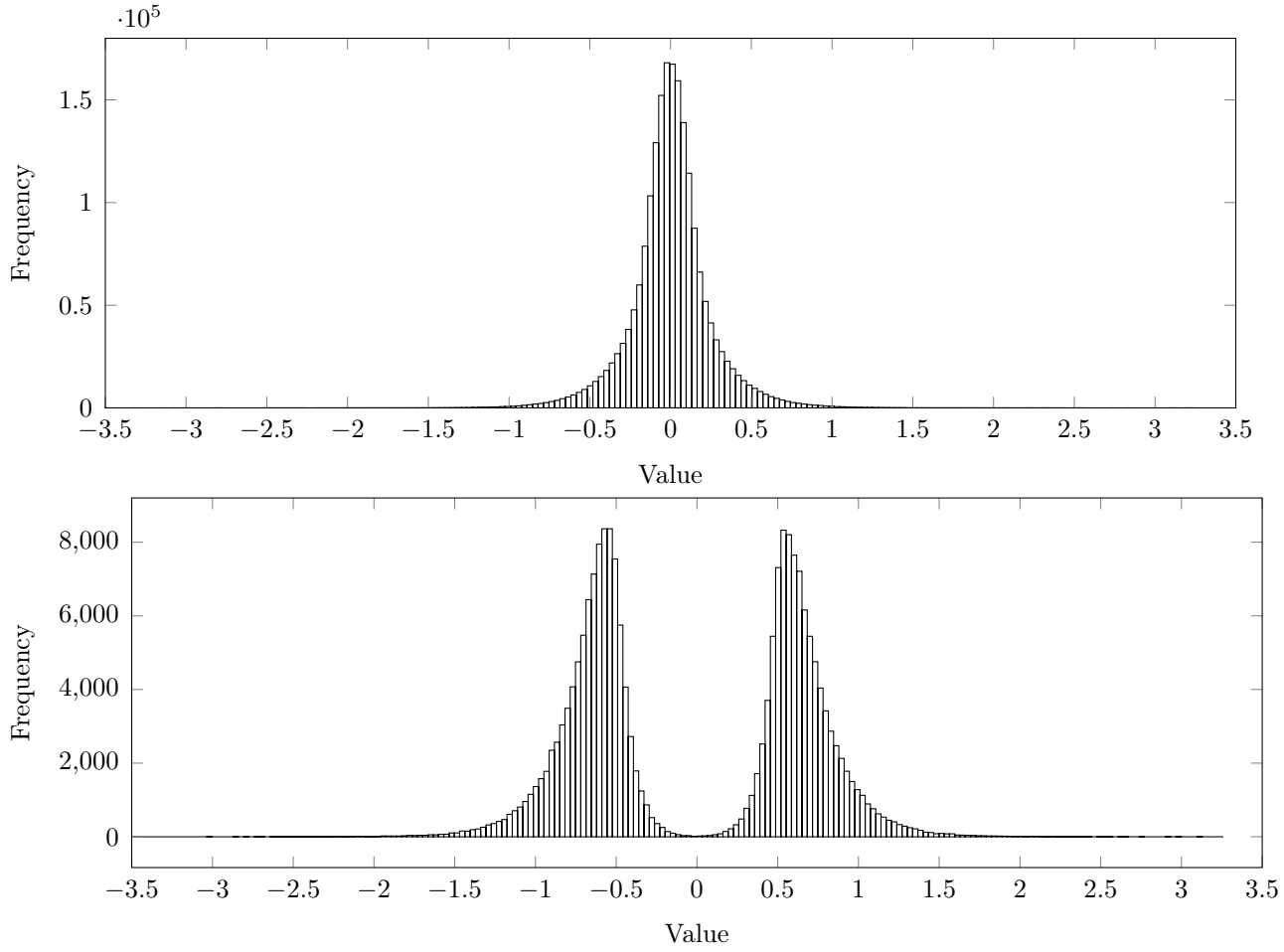


Рисунок 2.1 — Пример распределения весов до (вверху) и после применения прунинга (внизу)

использовать 32-битное представление, что означает, что мы должны использовать тот же объем памяти, что и в оригинальной модели.

Правда, необходимо отметить, что как в случае с прунингом, так и в случае с квантизацией возможно построение специальных процессоров, поддерживающих эффективные разреженные матричные вычисления и эффективные 16-битные и даже 8-битные вычисления, как, например, Tensor Processing Unit или TPU [66], но пока что такие процессоры не являются достаточно распространенными.

Глава 3. Матричные и тензорные разложения для сжатия нейронных сетей

3.1 Введение

В этой главе рассмотрим методы матричных разложений для задачи сжатия нейронных сетей и их возможные модификации в контексте задачи моделирования естественного языка. Операции умножения матриц хорошо оптимизированы на современных процессорах [1] и скорость их умножения напрямую зависит от их размера. То есть уменьшая размер матриц весов в нейронной сети, мы одновременно и снижаем количество параметров, и увеличиваем скорость вычислений. Мы рассматриваем различные варианты использования матричных разложений для сжатия нейронных сетей, а также адаптируем матричное низкоранговое и TT-разложение для использования в рекуррентных нейронных сетях с большими входными и выходными данными.

3.2 Обзор методов матричных разложений

В этом разделе мы приводим описания двух методов матричных разложений, с которыми мы не эксперементировали напрямую, но которые близки к нашим методам.

[1] Intel matrix multiplication

3.2.1 Адаптивное преобразование Fastfood

В [58] авторы описывают применение адаптивного преобразования FastFood к задаче сжатия больших полносвязных слоёв:

$$h_{\ell+1} = Wh_\ell, \tag{3.1}$$

здесь $W \in \mathbb{R}^{n\times m}$. Авторы предлагают вместо этой матрицы использовать следующую параметризацию:

$$h_{\ell+1} = (SHGPHB)\, h_\ell = \widehat{W} h_\ell \tag{3.2}$$

Идея здесь в том, что обычный полносвязный слой занимает $O(n \cdot m)$ места и требует $O(n \cdot m)$ вычислений во время работы, а такой FastFood слой занимает $O(n)$ места и требует $O(n \cdot \log m)$ вычислений. Далее поясним почему. Начнем с примера, когда $n = m$. Адаптивное преобразование FasFood имеет компоненты трёх типов:

- S, G, B — диагональные матрицы параметров. В оригинальном неадаптивном преобразовании FastFood они являются случайными матрицами. Они занимают $O(n)$ памяти и требуют столько же вычислений.
- $P \in \{0, 1\}^{n\times n}$ — это случайная матрица перестановок. Она может быть реализована просто как таблица, и она также требует $O(n)$ памяти и вычислений.
- H обозначает матрицу Волш-Адамара, которая определяется рекурсивно следующим образом:

$$H_2 := \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \text{ и } H_{2d} := \begin{pmatrix} H_d & H_d \\ H_d & -H_d \end{pmatrix} \tag{3.3}$$

Быстрое преобразование Адамара, которое является вариантом быстрого преобразования Фурье и позволяет вычислять матрицу произведение $H_n h_l$ за $O(n \log n)$ времени

Таким образом можно убедиться, что суммарно такой слой занимает $O(n)$ места и в нём производится $O(n \log n)$ вычислений. Если же $m > n$, то мы можем просто выполнить m/n преобразований FastFood и таким образом сделаем не более $O(m \log n)$ операций. Для полносвязного слоя требуется соответственно $O(n^2)$ и $O(n \cdot m)$ вычислений. Для этого преобразования в [58] предложен алгоритм обратного распространения ошибки, а также приводятся эксперименты, в которых показывается, что практически без потери качества можно уменьшить количество параметров в 10 раз.

3.2.2 Унитарная RNN

Унитарная RNN (unitary evolution RNN) была предложена в статье [10] как новый тип рекуррентной нейронной сети, использующий ортогональные матрицы и другие матрицы специального вида, и поэтому обладающий двумя важными свойствами:

- Нейронная сеть с использованием таких слоёв более эффективно выучивает долгосрочные зависимости за счёт борьбы с проблемой затухающих и взрывающихся градиентов.
- Для скрытого вектора размер n скрытый слой такой сети требуют $O(n)$ памяти и $O(n \log n)$ вычислений.

С точки зрения сжатия нас в большей степени интересует второе свойство. Рассмотрим, как такая сеть устроена. Ячейка такой сети использует в том числе комплексные числа, поэтому в рамках этого подраздела i везде означает комплексную единицу. Матрицы весов ячейки унитарной RNN задаются

по следующей формуле:

$$U = D_3 R_2 \mathcal{F}^{-1} D_2 P R_1 \mathcal{F} D_1 \tag{3.4}$$

Здесь

- D — это диагональная матрица $D_{j,j} = \exp(iw_j)$, с параметрами $w_j \in \mathbb{R}$
- R $= I - 2\frac{vv^*}{\|v\|^2}$ обозначает матрицу отражения с комплекснозначным вектором $v \in \mathbb{C}^n$
- P — случайная матрица перестановок
- $\mathcal{F}$ и $\mathcal{F}^{-1}$ обозначают прямое и обратное преобразование Фурье

Матрицы D, R и P занимают $O(n)$ места и требуют $O(n)$ вычислений при умножении. $\mathcal{F}$ и $\mathcal{F}^{-1}$ не требуют места и требуют $O(n \log n)$ вычислений с помощью быстрого преобразования Фурье (Fast Fourier Transform).

В статье [10] показано, что при меньшем количестве параметров данная архитектура в сравнении с LSTM добивается лучшего или такого же качества в ряде задач.

В обоих этих случаях (и в случае FastFood преобразования, и в случае URNN) авторами не была рассмотрена проблема моделирования языка. Хотя преобразование URNN было придумано специально для рекуррентных нейронных сетей.

Мы не обращаемся к этим методам в наших исследованиях, ввиду необходимости иметь реализации быстрого преобразования Фурье и быстрого преобразования Адамара не только в фреймворках глубинного обучения (даже в них это реализовано не всегда), но и на мобильном устройстве в случае тестирования в реальном приложении.

3.3 Методы матричного разложения для сжатия рекуррентных нейронных сетей

Матричное низкоранговое разложение очень популярно для сжатия нейронных сетей и уже не раз показало свою эффективность. Сама идея разложения происходит из разложения SVD (Singular Value Decomposition). Для любой матрицы $M \in \mathbb{R}^{n \times m}$ существует разложение

$$M = M^a \Sigma M^b, \tag{3.5}$$

где $M^a \in \mathbb{R}^{n \times n}$ и $M^b \in \mathbb{R}^{m \times m}$ – действительные унитарные матрицы и $\Sigma \in \mathbb{R}^{n \times m}$ – это прямоугольная матрица с неотрицательными элементами на диагонали, которые называются сингулярными числами.

Есть целый ряд результатов по примению такого типа разложений к нейронным сетям. Так в [67] показывается, что прямое обучение низкорангового разложения может привести к заметному ухудшению качества на некоторых задачах. Поэтому, например, в [68] проводят более сложную процедуру. Сначала обучается стандартная нейронная сеть, потом делается SVD для матриц весов, а потом эта конструкция дообучается. Мы будем учить нейронную сеть с заранее заданными низкоранговыми разложениями матриц весов.

Простейшее разложение для слоя RNN (1.3-1.4) может быть выполнено в следующей форме:

$$x_\ell^t = \sigma\left[W_\ell^a W_\ell^b x_{\ell-1}^t + U_\ell^a U_\ell^b x_\ell^{t-1} + b_\ell\right] \tag{3.6}$$

Здесь $U_\ell^a, W_\ell^a \in R^{k \times r}$ и $U_\ell^b, W_\ell^b \in R^{r \times k}$, где k – оригинальный размер скрытого слоя, а r — ранг разложения. В [9] при применении рекуррентной нейронной сети к задаче распознавания голоса авторы предложили использовать следующее ограничение: $W_l^b = U_{\ell-1}^b$. В таком случае уравнение для слоя RNN

можно переписать следующим образом:

$$x_l^t = \sigma\left[W_l^a m_{l-1}^t + U_l^a m_l^{t-1} + b_l\right] \tag{3.7}$$

$$m_l^t = U_l^b x_l^t \tag{3.8}$$

$$y_t = \text{softmax}\left[W_{L+1} m_L^t + b_{L+1}\right] \tag{3.9}$$

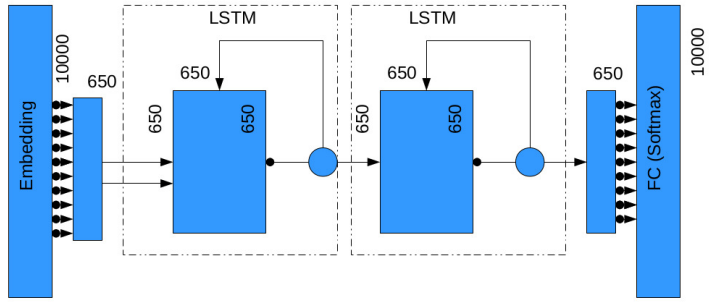
Сжатая LSTM сеть может быть описана теми же уравнениями 1.20-1.24, но размеры матриц будут другими. Здесь, аналогично случаю с RNN, мы требуем существование специальной матрицы W_l^p такой, что $W_l^{ib} = W_l^{fb} = W_l^{cb} = W_l^{ob} = U_{l-1}^{ib} = U_{l-1}^{fb} = U_{l-1}^{cb} = U_{l-1}^{ob} = W_l^p$ и выход нейронной сети будет рассчитывать следующим образом:

$$\hat{x}_l^l = W_l^p x_l^t. \tag{3.10}$$

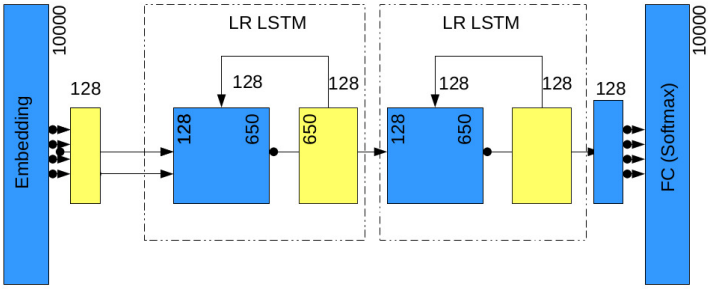
В случае с GRU разложение делается не так явно. Если мы просто уменьшим размеры матриц $W_l^z, U_l^z, W_l^f, U_l^f$ до $k \times r$, в уравнениях (1.27-1.28) у нас окажутся неправильные размерности. Поэтому мы уменьшаем эти матрицы до $r \times r$, уменьшаем W_l^h и U_l^h до $r \times k$ и вводим специальную проекционную матрицу $W_l^p \in \mathbb{R}^{k \times r}$ после уравнения 1.27 , так что уравнение 1.28 применяется также, но с заменой $\tilde{x}_t^l$ на

$$x_t^{lp} = W_l^p \tilde{x}_t^l. \tag{3.11}$$

Главное преимущество такого низкорангового разложения лежит в потенциально маленьких размерах $r \times k$ и $k \times r$ для матриц W_l^a / U_l^b и U_l^a, соотвественно в случае RNN. Они намного меньше, чем $k \times k$ оригинальных матриц W_l and V_l если $r \ll k$. С маленьким r мы получаем выигрыш и в размере сети, и в скорости вычисления. Похожие соображения справедливы для ячеек LSTM и GRU соответственно с 8 и 6 матрицами. Схематично такое разложение можно изобразить как на Рис. 3.1.



а)



б)

Рисунок 3.1 — Архитектуры нейронных сетей: (a) оригинальная сеть LSTM 650-650, (b) Модель, сжатая с помощью низкорангового разложения.

3.4 Алгоритм обратного распространения ошибки в случае матриц низкого ранга

Опишем, как изменится алгоритм обратного распространения ошибки через время в случае обучения рекуррентной нейронной сети с низкоранговым разложением (3.7-3.9). Теперь обучаемыми параметрами являются $\theta_{lr} = \left(W_0^a, W_0^b, W_1^a, U_1^a, U_1^b, b_1 \ldots, W_L^a, U_L^a, U_L^b, b_L, W_{L+1}^a, W_{L+1}^b\right)$, но суть алгоритма при этом меняется не сильно, только уравнений становится больше, так как матриц больше (см. Алгоритм 4). При этом обостряются все проблемы, связанные с затуханием или взрывом градиента, описанные в разделе 1.5. Поэтому возрастает важность применения всех приёмов регуляризации, которые были описаны в разделе 1.7.

Algorithm 4 Алгоритм обратного распространения ошибки через время в случае матриц низкого ранга

Входные данные: Текущие значения параметров $\theta_{\ell r}$, вычисленная функция $F(y_t, y_t^*, \theta)$ для $t \in \{1, \ldots, N\}$
Инициализация вектора градиентов параметров
$\nabla\ \theta\ = \big(\nabla W_0,\ \nabla W_1, \nabla U_1^a, \nabla U_1^b, \nabla b_1 \ldots, \nabla W_L, \nabla U_L^a, \nabla U_L^b, \nabla b_L, \nabla W_{L+1}\big)$
$\nabla W_{L+1}^a := \frac{\partial F_t}{\partial W_{L+1}^a}$
$\nabla W_{L+1}^b := \frac{\partial F_t}{\partial W_{L+1}^b}$
for ℓ **from** L **downto** 1 **do**
  for t **from** T **downto** 1 **do**
    $g_{z_l^t} := g_{z_\ell^t} \frac{\partial^+ z^{t+1}}{\partial z^t} + \frac{\nabla F_t}{\nabla z^t}$
    $\nabla W_\ell := \nabla W_\ell + g_{z_t^l} \frac{\partial^+ z^t}{\partial W_\ell}$
    $\nabla U_\ell^a := \nabla U_\ell^a + g_{z_t^l} \frac{\partial^+ z_l^t}{\partial U_\ell}$
    $\nabla b_\ell := \nabla b_\ell + g_{z_t^l} \frac{\partial^+ z_l^t}{\partial b_\ell}$
    $\nabla U_\ell^b := \nabla U_\ell^b + g_{z_t^l} \frac{\partial^+ z_l^t}{\partial U_\ell^b}$
  end for
end for
$\nabla W_0^b := \frac{\partial z^1}{z^0} \frac{\partial x^0}{\partial W_0^b}$
$\nabla W_0^a := \frac{\partial z^1}{z^0} \frac{\partial x^0}{\partial W_0^a}$
Выходные данные: вектор градиентов параметров
$\nabla\theta = \big(\nabla W_0, \nabla W_1, \nabla U_1^a, \nabla U_1^b, \nabla b_1 \ldots, \nabla W_L, \nabla U_L^a, \nabla U_L^b, \nabla b_L, \nabla W_{L+1}\big)$

3.5 Сжатие нейронной сети с помощью TT разложения

В этом разделе мы опишем ещё один метод для сжатия нейронных сетей, который, по сути, близок к низкоранговому матричному разложению. Это разложение матриц весов в так называемый тензорный поезд (Tensor Train, TT). Изначально оно было предложено как более эффективная форма представления тензора [69] и является одним из возможных обобщений SVD на тензоры размерности больше двух. Использование TT-разложения мотивировано возможностью конвертировать матрицы в соответствующие им тензоры, после чего раскладывать и сжимать уже их. За счёт того, что в тензоре мы имеем сразу несколько размерностей, мы получаем потенциально больший размер сжатия

Таблица 3.1

Матричное низкоранговое разложение и Tensor Train разложение для выходного слоя нейронной сети

	Модель	**Размер, Mb**	**Кол-во параметров всего, млн**	**Кол-во параметров выходного слоя, млн**	**Тестовая PP**
PTB Benchmarks	LSTM 200-200	18.6	4.64	2.0	117.659
	LSTM 650-650	79.1	19.7	6.5	82.07
	LSTM 1500-1500	264.1	66.02	15.0	78.29
LR for Softmax layer	LSTM 200-200	12.6	3.15	0.51	112.065
	LSTM 650-650	57.9	14.48	1.193	84.12
	LSTM 1500-1500	215.4	53.85	2.829	89.613
TT for Softmax layer	LSTM 200-200	11.8	2.95	0.304	116.588
	LSTM 600-600	51.12	12.8	1.03	88.551
	LSTM 1500-1500	215.8	53.95	2.92	85.63

и большую гибкость в сжатии. Кроме того, возможно, с помощью TT-разложения можно как-то учитывать важность нейронов и структуру матриц весов.

3.5.1 Описание TT-разложения

Начнем с определения

Определение 3.5.1. *Представление d-мерного тензора $\mathcal{A} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ в формате Tensor Train – это набор трёхмерных $G_1, \ldots, G_d$ таких, что*

$$\mathcal{A}(i_1, \ldots, i_d) = G_1(i_1) \ldots, G_d(i_d), \tag{3.12}$$

где $G_k \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$.

Здесь запись $\mathcal{A}(i_1, \ldots, i_d)$ обозначает элемент тензора d-мерного $\mathcal{A}$ в позиции $(i_1, \ldots, i_d)$. Числа $r_0, \ldots, r_d$ называются рангами разложения. Из уравнения 3.12 также следует, что $r_0 = r_d = 1$ (так как слева в равенстве стоит число).

Для того чтобы подробнее обсудить это представление, нам потребуется дополнительное определение.

Определение 3.5.2. ***Матрицей развёртки*** A_k *для тензора* $\mathcal{A}$ *будем называть матрицу, для которой верно следующее:*

$$\mathcal{A}(i_1, \ldots, i_d) = A_k(i_1, \ldots, i_k; i_{k+1}, \ldots, i_d) = A_k, \tag{3.13}$$

то есть первые k *индексов обозначают строку матрицы* A_k*, а оставшиеся* $d - k$ *столбцы.*

Другими словами, матрицы развертки — это матрицы, которые получаются в результате операции **reshape** для исходного тензора. Например, в языке программирования **Python 3.7** в библиотеке **numpy** такое преобразование можно сделать следующим образом: **A.reshape** $\left(\left(\prod_{i=1}^{k} n_i, \prod_{i=k+1}^{d} n_i, \right)\right)$ По аналогии с [69] сформулируем и докажем теорему о ранге такого разложения.

Теорема 1. *Пусть* r_k *– ранг матрицы развёртки* A_k *для тензора* $\mathcal{A}$*. Обозначим* $R = \max\limits_{i=1,\ldots,d} r_k$*. Тогда существует TT-разложение тензора* $\mathcal{A}$ *с рангом не превосходящим* R*.*

Доказательство. Рассмотрим матрицу развёртки A_1. Для неё существует представление:

$$A_1 = UV^T$$

или же, в индексной форме,

$$A_1(i_1; i_2, \ldots, i_d) = \sum_{\alpha_1=1}^{r_1} U(i_1, \alpha_1) V(\alpha_1, i_2, \ldots, i_d) \tag{3.14}$$

Далее из этого уравнения выразим матрицу V:

$$V = A_1^T U \left(U^T U\right)^{-1} = A_1^T W \tag{3.15}$$

или, в индексной форме,

$$V(\alpha_1, i_2, \ldots, i_d) = \sum_{i_1=1}^{n_1} A_1^T(i_1; i_2, \ldots, i_d) W(i_1, \alpha_1) \tag{3.16}$$

Теперь мы можем применить операцию **reshape** для матрицы V и превратить её в $(d-1)$-мерный тензор $\mathcal{V}(\alpha_1 i_2, i_3, \ldots, i_d)$, считая что $(\alpha_1 i_2)$ является одной большой размерностью.

Далее, рассмотрим матрицы развёртки $V_2 \ldots, V_d$ для тензора $\mathcal{V}$. Необходимо показать, что соотношение $V_k \leq R$ является верным. Для этого рассмотрим следующее представление изначального тензора $\mathcal{A}$.

$$\mathcal{A}(i_1, \ldots, i_d) = \sum_{j=1}^{r_k} F(i_1, \ldots, i_k, j) G(j, i_{k+1}, \ldots, i_d) \tag{3.17}$$

Используя это, из уравения 3.16 получаем

$$V_k = V(\alpha_1 i_2, \ldots, i_k; i_{k+1}, \ldots, i_d) = \tag{3.18}$$

$$= \sum_{i_1=1}^{n_1} \sum_{j=1}^{r_k} W(i_1, \alpha_1) F(i_1, \ldots, i_k, j) G(j, i_{k+1}, \ldots, i_d). \tag{3.19}$$

Поменяем местами суммирования и обозначим

$$H(\alpha_1 i_2, \ldots, i_k, j) = \sum_{i_1=1}^{n_1} F(i_1, \ldots, i_k, j) W(i_1, \alpha_1), \tag{3.20}$$

тогда получим

$$V(\alpha_1 i_2, \ldots, i_k; i_{k+1}, \ldots, i_d) = \sum_{j=1}^{r_k} H(\alpha_1 i_2, \ldots, i_k, j) G(j, i_{k+1}, \ldots, i_d). \tag{3.21}$$

Отсюда можно видеть, что индексы столбцов и строчек матрицы V_k разделены и $\operatorname{rank} V_k \leq R$. Этот процесс можно продолжить по индукции. Рассмотрим тензор $\mathcal{V}$ и отделим индекс (α_1, i_2) от остальных:

$$\mathcal{V}(\alpha_1 i_2, i_3, \ldots, i_d) = \sum_{\alpha_2=1}^{r_2} G_2(\alpha_1, i_2, \alpha_2)(V)'(\alpha_2 i_3, i_4, \ldots, i_d). \tag{3.22}$$

Таким образом, мы получили следующее ядро разложения, тензор $G_2(\alpha_1, i_2, \alpha_2)$. Первым ядром является матрица $W(i_1, \alpha_1)$, так как α_0 имеет единичную размерность. Продолжая описанный процесс, мы как раз получим TT-представление, для которого будут выполнены требования теоремы. □

Как мы видим, доказательство теоремы конструктивно и даёт нам алгоритм построения TT-разложения. В этой теореме ничего не говорится о том, какое матричное разложение используется на каждом шаге построения, так и о том, насколько хорошо такое TT-разложение приближает исходный тензор.

В [69] также показывается, что если использовать в качестве разложения на каждом шаге SVD, то норма разности оригинального тензора $\mathcal{A}$ и его TT-разложения оценивается через ошибку, которая возникает при каждом SVD разложении. Сам алгоритм построения TT-разложения с использованием SVD называется TT-SVD. Сформулируем эту теорему.

Теорема 2. *Пусть для матриц развёртки A_k справедливо следующее:*

$$A_k = R_k + E_k, \tag{3.23}$$

где $\operatorname{rank} R_k = r_k$, $\|E_k\|_F = \varepsilon_k$ *и* $\|\cdot\|_F$ *обозначает норму Фробениуса. Тогда алгоритм TT-SVD вычисляет TT-разложение для тензора* $\mathbb{A}$ *с TT-рангами* r_k *и справедливо следующее:*

$$\|\mathbb{A} - \mathrm{TT}(\mathbb{A})\|_F \leq \sqrt{\sum_{k=1}^{d-1} \varepsilon_k^2} \tag{3.24}$$

Доказательство см. в [69; 70]. Таким образом, TT-разложение позволяет для любого тензора находить аппроксимацию с произвольной точностью в смысле нормы Фробениуса. Верно и обратное. Произвольное TT-разложение задаёт некоторый тензор $\mathbb{A}$ и является его аппроксимацией с некоторой точностью. Этими свойствами мы будем пользоваться для сжатия рекуррентных нейронных сетей.

3.5.2 Применение TT для сжатия нейронной сети

Опишем, как такое разложение может быть применено к нейронным сетям. Рассмотрим, например, матрицу весов $W \in \mathbb{R}^{k \times k}$ RNN слоя (1.3). Далее мы можем выбрать такие числа $k_1, ..., k_d$, чтобы $k_1 \times \ldots \times k_d = k \times k$ и произвести преобразование матрицы W в тензор $\mathcal{W} \in \mathbb{R}^{k_1 \times \ldots \times k_d}$. Здесь d – размерность нового тензора, $k_1, \ldots, k_d$ – внутренние размеры каждого измерения. Далее мы можем выполнить TT-разложение тензора $\mathcal{W}$ и получить набор матриц $G_m[i_m] \in \mathbb{R}^{r_{m-1} \times r_m}, i_m = 1, \ldots, k_m$, $m = 1, \ldots, d$ и $r_0 = r_d = 1$ такой, что каждый элемент тензора может быть представлен как $\mathcal{W}(i_1, i_2, \ldots, i_d) = G_1[i_1] G_2[i_2] \ldots G_d[i_d]$. Как уже упоминалось, числа $r_0, \ldots r_m$ называются рангами Tensor Train разложения. Фактически каждый $G_m \in \mathbb{R}^{r_{m-1} \times k_m \times r_m}$ это трёхмерный тензор с измерением k_m, которое соответствует измерению в оригинальном тензоре, и двумя рангами r_{m-1} и r_m, которые в некотором смысле отвечают за внутреннее представление этого измерения внутри разложения. Также необходимо подчеркнуть, что даже если мы зафиксируем структуру тензора (количество измерений и их размерности), в который мы конвертируем матрицу весов, у нас все ещё будет богатый выбор рангов для TT-разложения в качестве гиперпараметров.

Обозначим эти две операции – конвертацию матрицы W в тензор $\mathcal{W}$ и её последующее разложение в Tensor Train формат – как одну операцию $\mathrm{TT}(W)$. Применяя её к обеим матрицам W и V в рекуррентном слое (1.3), мы получаем TT-RNN слой в следующей форме:

$$z_\ell^t = \tanh(\mathrm{TT}(W_l) x_{\ell-1}^t + \mathrm{TT}(U_l) x_\ell^{t-1} + b_\ell). \tag{3.25}$$

Похожим образом мы можем применить TT-разложение для каждой матрицы LSTM слоя (1.20–1.24) или для матрицы выходного слоя (1.5).

Сжатие с помощью такого разложения может быть достигнуто за счет соответствующего выбора внутренних рангов $r_1, \ldots, r_{d-1}$. Пусть $R = \max\limits_{m=0,\ldots,d} r_m$, $K = \max\limits_{m=0,\ldots,d} k_m$. Тогда количество параметров в TT-разложении $N_{\mathrm{TT}} =$

$\sum r_{m-1} k_m r_m \leq dR^2K$. Фактически, каждый множитель в этом произведении может быть на порядок меньше, чем оригинальное k.

3.6 Эксперименты по сжатию рекуррентных нейронных сетей с использованием низкорангового и TT-разложения

3.6.1 Описание экспериментов и детали реализации

При проведении экспериментов в качестве базовой архитектуры были использованы модели из [52]. По умолчанию там было рассмотрено три LSTM модели: Small, Medium, Large с размерами скрытого слоя соответственно 200, 650 и 1500. Мы также провели дополнительные эксперименты с другими размерами сетей и с другими типами ячеек, такими как RNN и GRU. Все эксперименты по матричным разложениям были сделаны на датасете PTB (Penn TreeBank) [50].

Мы сравнивали все модели в терминах двух метрик: перплексии и количества параметров. Как мы уже упоминали, перплексия языковой модели это удобная метрика для оценки её качества. В некотором смысле перплексия модели соответствует среднему количеству слов, из которого модель равномерно выбирает предсказание на каждом шаге. Также мы можем дополнительно оценить **среднюю точность предсказания слова**, то есть вероятность предсказать следующее слово в предложении, которое может быть оценено величиной, обратной перплексии.

В дополнение мы измерили среднее время работы модели (inference time), используя наши собственные реализации сжатых и несжатых RNN для мобильных GPU. Необходимо отметить, что наша реализация была одной из первых реализаций нейронных сетей на мобильных GPU. Тестирование было проведено на мобильном телефоне и сравнивало производительность сжатых моделей. Для

экспериментов был использован мобильный телефон Samsung S7 Edge с GPU процессором Mali-T880. Все расчеты проводились после фазы прогрева (100 предварительных умножений случайных матриц). Это нужно, чтобы потребление энергии GPU вышло на необходимый уровень Мы приводим результаты времени работы нейронной сети на одном сэмпле, усредненные за 1000 испытаний для каждой модели.

Мы реализовали все упомянутые выше методы сжатия. Прунинг и квантизация были протестированы на маленькой LSTM модели из [52]. Кроме того, мы тщательно изучили, как ведет себя низкоранговое разложение для LSTM различных размеров, а также для других ячеек GRU и обычных RNN. В экспериментах мы удерживали уровень сжатия в диапазоне 3-5 раз.

Из-за большого числа гиперпараметров (скорость обучения, расписание изменения скорости обучения, значение дропаута, размер разложения, размер внутреннего слоя и тд) их полный перебор требует огромных вычислительных мощностей. Чтобы избежать этого, но при этом подобрать более-менее оптимальные гиперпараметры, мы использовали случайный поиск (см. Гл. 1) по этому пространству.

Отдельно стоит сказать про подбор рангов в случае TT-разложения. В этом случае для заданной сети мы фиксируем количество тензоров в разложении и подбираем внутренние ранги, далее выбираем тот набор рангов, который даёт наилучшее сотношение перплексии и сжатия.

Опишем подробнее сжатие сети LSTM-medium с помощью разложения в Tensor Train. Вместо матриц 650×650 мы взяли матрицы 600×600, так как они дают большую вариативность в потенциальном разложении ($600 = 2 \cdot 3 \cdot 5 \cdot 2 \cdot 5 \cdot 2$ и $650 = 2 \cdot 5 \cdot 5 \cdot 13$). Далее мы зафиксировали количество ядер в разложении. Здесь мы взяли 4 ядра. После чего мы запустили перебор по рангам разложения и последующее обучение каждой конфигурации. Надо сказать, что это вычислительно затратная процедура, поэтому удается перебрать всего лишь порядка 100 различных наборов рангов.

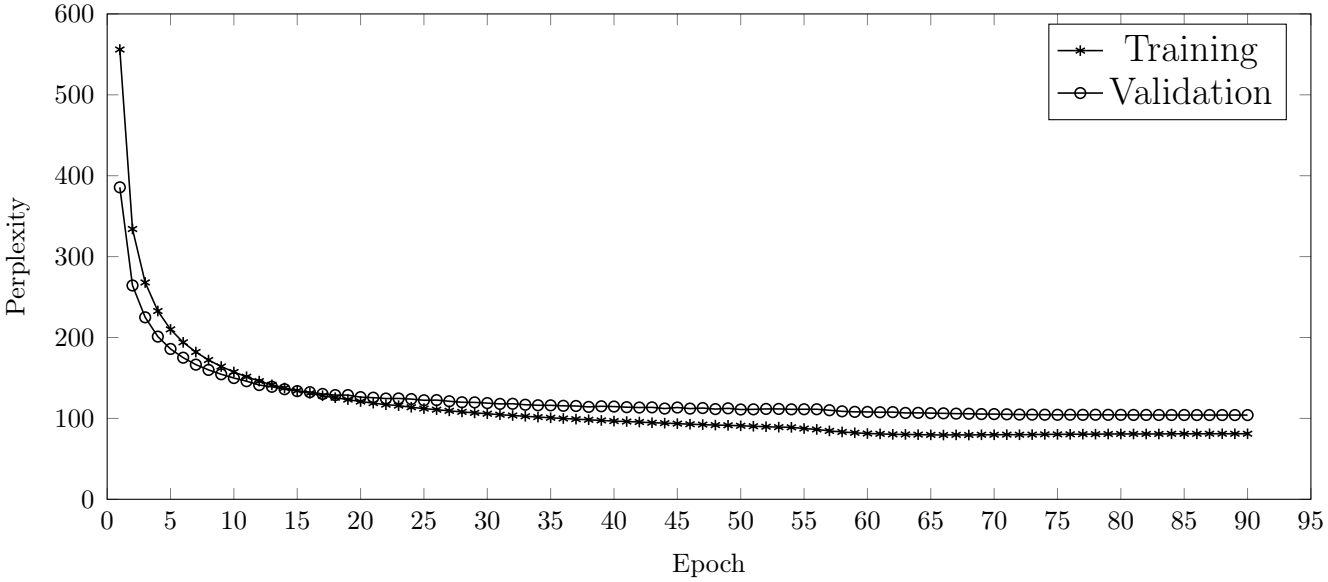


Рисунок 3.2 — Кривые обучения для **LR LSTM 500-500** модели

Для низкоранговых моделей саму процедуру обучения мы проводим в два этапа. На первом этапе мы используем оптимизатор Adam [34], а на втором переключаемся на обычный SGD. Характерный пример кривых функции потерь обучающей и валидационной выборки представлен на Рис. 3.2. Там показаны кривые для обучения модели **LR LSTM 500-500**. В автоматической процедуре настройки мы стараемся не допустить переобучения, используя описанные техники регуляризации, а также остановку обучения в момент, когда функция потерь на валидационной выборке начинает расти.

Таблица 3.2

Результаты SotA моделей для датасета PTB

Model	**Size, Mb**	**No. of param., M**	**Test perplexity**	**Avg. word prediction accuracy**
RNN-LDA+KN-5 cache [71]	36	9	92	0.0109
LSTM 650-650 [52]	79.1	19.7	82.7	0.0121
Variational LSTM (MC) [72]	80	20	78.6	0.0127
CharCNN [73]	76	19	78.9	0.0127
Variational RHN [74]	92	23	65.4	0.0153
AWD-LSTM [75]	88	22	55.97	0.0179
AWD-LSTM-MoS [76]	88	22	54.44	0.0184
TrellisNet-MoS [77]	136	34	54.19	0.0185
LSTM-SparseVD [64]	13.248	3.312	109.2	0.0092
LSTM-SparseVD-VOC [64]	6.688	1.672	120.2	0.0083

Таблица 3.3

Результаты сжатия на датасете PTB. Проведены эксперименты с различными типами рекуррентных ячеек и с различными разместностями. Также здесь приводятся результаты прунинга и квантизации

	Модель	**Размер, Mb**	**Кол-во парам., млн**	**Тестовая перплексия**	**Средняя точность предсказания слова**	**Время работы, ms**
PTB Baselines	LSTM 200-200	18.6	4.64	117.659	0.0085	9.63
	LSTM 300-300	29.8	7.45	91.95	0.0109	10.24
	LSTM 400-400	42.24	10.56	86.687	0.0115	12.4
	LSTM 500-500	56	14	84.778	0.0118	14.03
	LSTM 650-650	79.1	19.7	82.07	0.0122	16.13
	RNN 650-650	67.6	16.9	124.371	0.008	15.91
	GRU 650-650	72.28	18.07	92.86	0.0108	16.94
	LSTM 1500-1500	264.1	66.02	78.29	0.0128	45.47
Ours	LSTM 200-200 pruning output layer 90% w/o additional training	5.5	0.5	149.31	0.0067	9.56
	LSTM 200-200 pruning output layer 90% with additional training	5.5	0.5	121.123	0.0083	9.56
	LSTM 200-200 quantization (1 byte per number)	4.7	4.64	118.232	0.0085	9.61
	LR LSTM 200-200	3.712	0.928	136.115	0.0073	7.83
	LR LSTM 300-300	8.228	2.072	113.691	0.0088	8.39
	LR LSTM 400-400	13.12	3.28	106.623	0.0094	8.82
	LR LSTM 500-500	14.336	3.584	97.282	0.0103	8.95
	LR LSTM 650-650	16.8	4.2	92.885	0.0108	9.68
	LR RNN 650-650	35	8.75	134.111	0.0075	11.03
	LR GRU 650-650	12.76	3.19	111.06	0.009	8.74
	TT LSTM 600-600	50.4	12.6	168.639	0.0059	16.75
	LR LSTM 1500-1500	94.9	23.72	89.462	0.0112	15.70

3.6.2 Результаты прунинга и квантизации

Как можно видеть из Таблицы 3.3, прунинг и квантизация позволяют позволяют достичь хорошего уровня сжатия, иногда даже без потери качества. К сожалению, ни один из этих методов не подходит для уменьшения времени работы нейронной сети на мобильном телефоне. Более точно: разница во времени работы сжатой модели в сравнении с оригинальной несжатой моделью

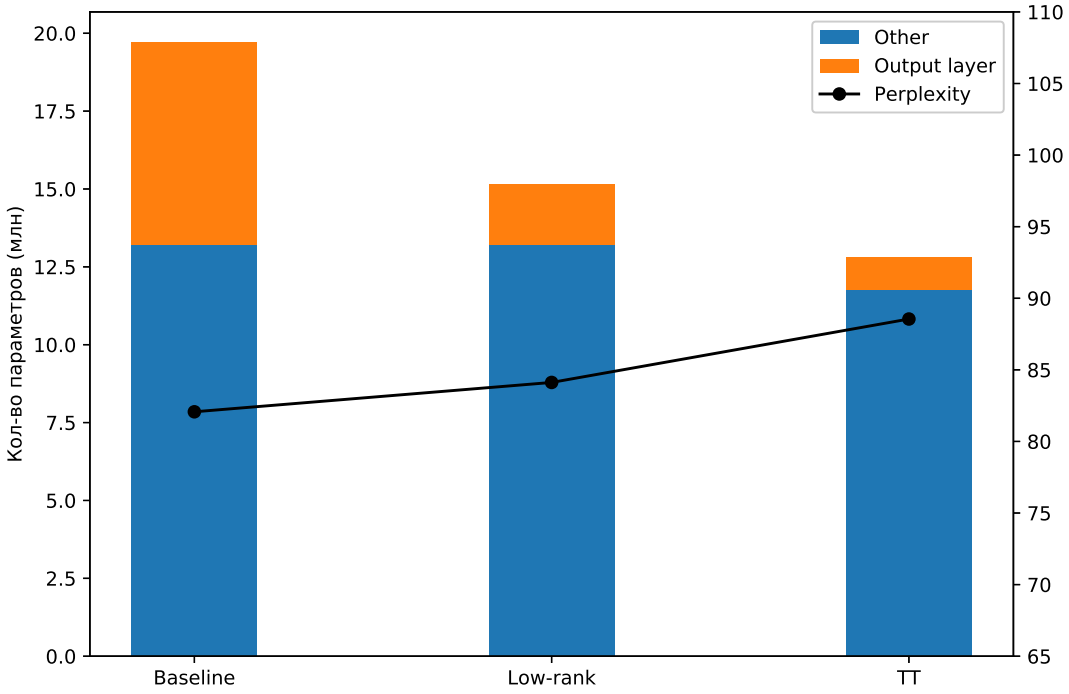


Рисунок 3.3 — Диаграмма сжатия сети, LSTM 650-650, только выходной слой

LSTM 200-200 не значима статистически при использовании теста МакНемара с уровнем статистической значимости 5%.

3.6.3 Результаты экспериментов

Для того, чтобы проанализировать сокращение параметров при сжатии, подробно опишем разложение модели **LSTM 650-650**. Мы начинаем с начальных размеров $W \in \mathbb{R}^{650\times650}$ $U \in \mathbb{R}^{650\times650}$ и $|\mathbb{V}| = 10{,}000$. Матрицы, отвечающие за эмбеддинг и за выходной слой, это соответственно $W_{emb} \in \mathbb{R}^{10{,}000\times650}$ и $W_{out} \in \mathbb{R}^{10{,}000\times650}$. Далее во время сжатия размеры каждой из матриц W и U уменьшаются до 650×128, а размеры матрицы эмбеддингов и выходной матрицы уменьшаются соответственно до $10{,}000\times128$ and $128\times10{,}000$. Здесь значение 128 выбирается как наиболее подходящая степень двойки для эффективной реализации на конечном устройстве. Были проведены эксперименты и с другими

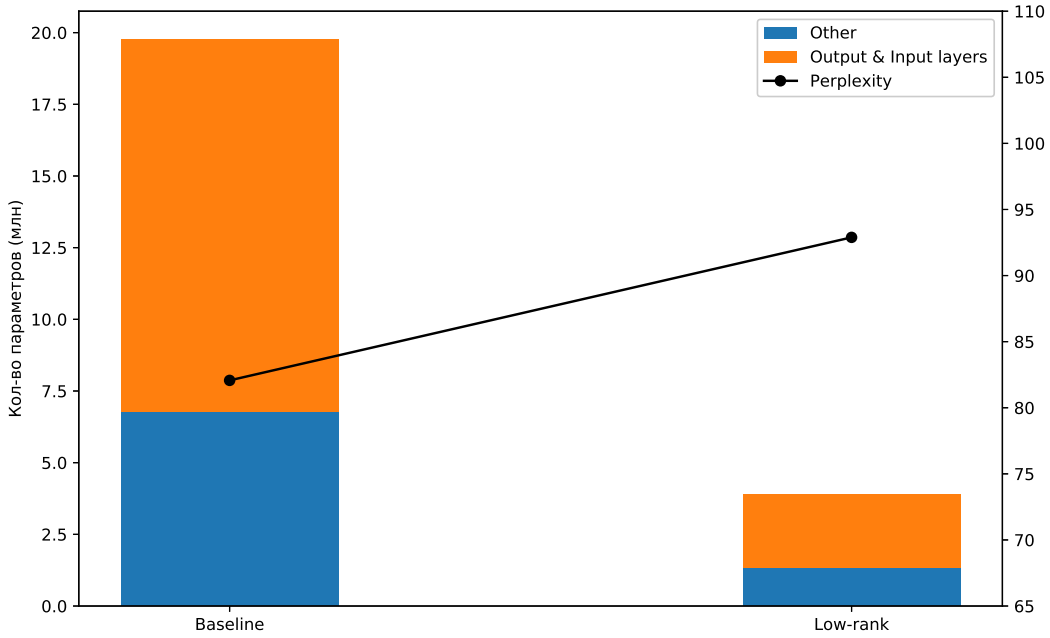


Рисунок 3.4 — Диаграмма сжатия сети, LSTM 650-650, вся сеть

конфигурациями, но вышеупомянутая оказалась наилучшей в терминах сжатия и перплексии. Схемы оригинальной и сжатой по LR представлены на Рис. 3.1

Все эти соображения подтверждены результатами экспериментов, представленными в Таблице 3.3 и Таблице 3.1. В Таблице 3.3 также приведены контрольные замеры скорости работы таких рекуррентных нейронных сетей на мобильном телефоне Samsung S7 Edge. Как видно из приведенных результатов, с помощью низкорангового разложения удалось достичь сжатия модели LSTM 650-650 почти в пять раз с некоторой потерей качества, но при этом с показателем перплексии всё ещё ниже, чем у модели LSTM 200-200.

Результаты применения TT-разложения также можно найти в Таблицах 3.1 и 3.3. При применении этого метода для выходного слоя с большим числом классов Tensor Train в основном даёт качество, сравнимое с применением низкорангового разложения.

Важная особенность низкоранговых разложений – это высокая эффективность операций на мобильных устройствах. Эксперименты демонстрируют, что наша модель **LR LSTM 650-650**, сжатая с помощью такого разложения, меньше по числу параметров и по памяти, чем обычная **LSTM 200-200**, в то время

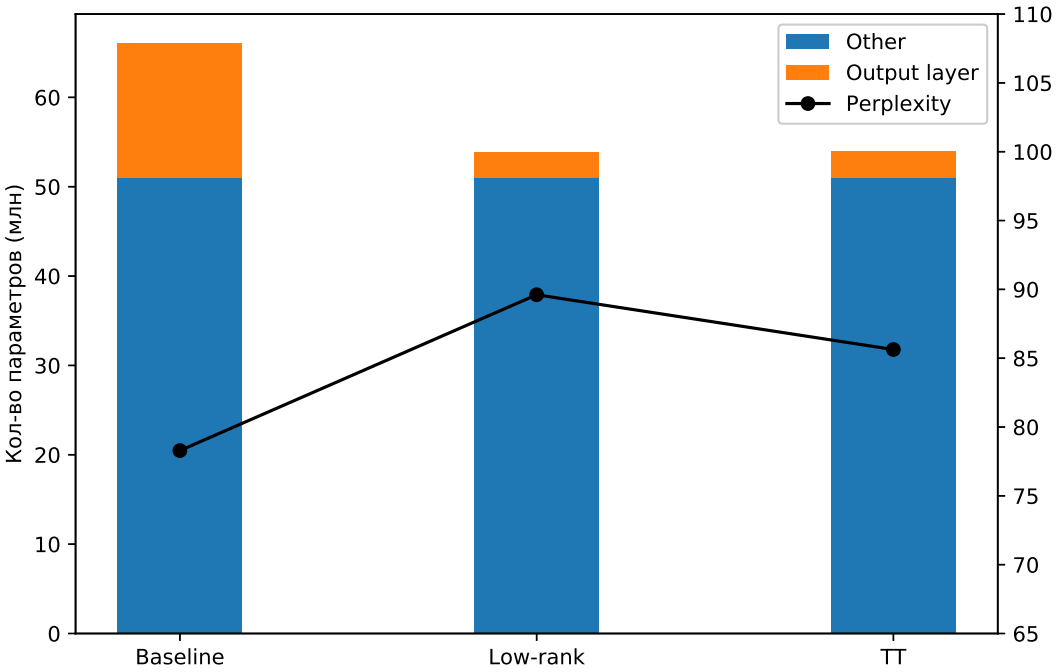


Рисунок 3.5 — Диаграмма сжатия сети, LSTM 1500-1500, только выходной слой

как перплексия в последней гораздо хуже и разница в перплексии статистически значима. Время работы нашей сжатой модели **LR LSTM 650-650** на мобильном телефоне приблизительно равно времени работы **LSTM 200-200**. Во всех случаях разница в размерах моделей и времени работы сжатых и оригинальных моделей статистически значима.

В Таблице 3.2 представлены state-of-the-art результаты для задачи моделирования языка. Отсюда можно видеть, что несмотря на высокое качество предсказаний этих методов, количество параметров в них в 3-6 раз выше, чем в наших сжатих моделях и этот результат также статистически значим. Более того, с практической точки зрения средняя точность предсказания следующего слова – это метрика, которую гораздо проще интерпретировать. Сравнивая именно её по Таблицам 3.2 и 3.3 и можно видеть, что точности наших сжатых моделей и state-of-the-art моделей очень близки друг к другу.

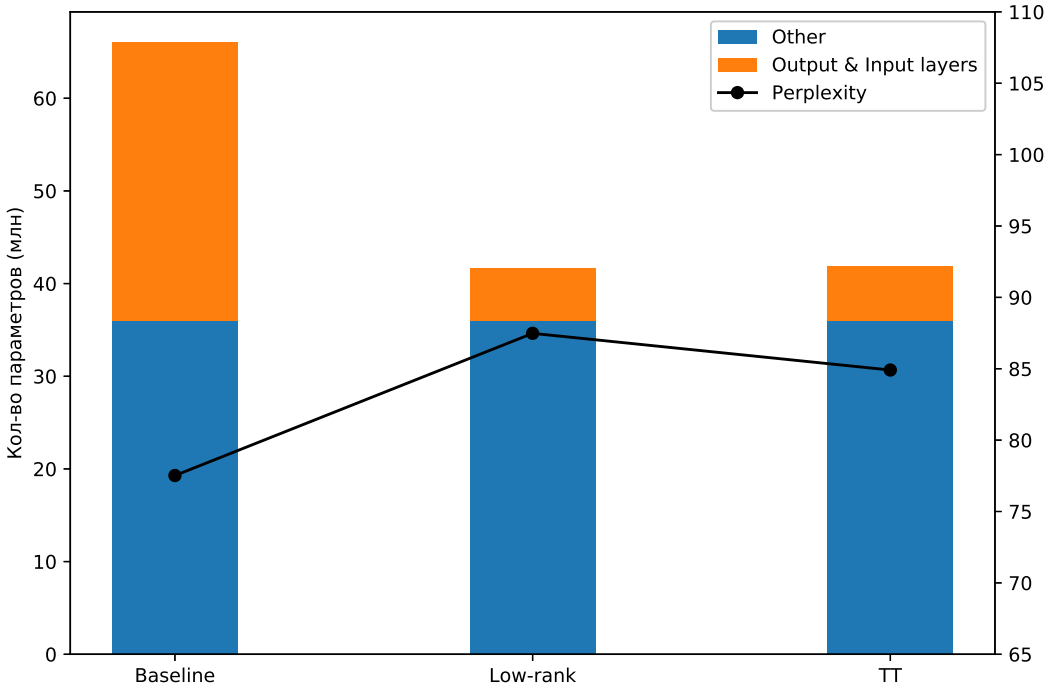


Рисунок 3.6 — Диаграмма сжатия сети, LSTM -1500 входной и выходной слой, “tied weigth”

3.6.4 Результаты разложения софтмакс слоя

В другой серии экспериментов мы анализировали эффективность применения низкорангового разложения и разложения в TT в применении к последнему (полносвязному) слою. Мы зафиксировали всю архитектуру нейронной сети и изменяли только этот слой. Для каждого эксперимента был выполнен случайный поиск среди следующих параметров: внутренние ранги разложения TT, скорость обучения, расписание изменения скорости обучения и значение вероятности дропаута. Лучшие результаты по скорости и потреблению памяти, которые удалось достичь только на разложении последнего высокоразмерного слоя, представлены в Таблице 3.1.

Разложение последнего слоя уменьшает общий размер модели в 1.2-1.5 раз. Несмотря на то, что перплексия в общем увеличивается, иногда удаётся достичь её уменьшения, как, например, в случае с моделью **LSTM 200-200**. В

этой задаче TT-разложение показало себя более успешно. Однако, необходимо заметить, что, в целом, слои нейронной сети, представленные в TT-формате, всё ещё достаточно нестабильны и сильно зависит от гиперпараметров. Таким образом, обычно требуется перебрать гораздо больше конфигураций для достижения приемлимых результатов.

Для более наглядного представления уровня сжатия см. Рис. 3.3 - 3.6. Здесь оранжевым цветом показано, сколько занимают входные и выходные слои (на Рис. 3.3 и Рис. 3.5 только входной слой), а синим размер всей остальной сети. Соответственно, можно визуально оценить уровень сжатия за счёт только уменьшения входного слоя сети.

3.7 Общая схема сжатия рекуррентных нейронных сетей

Суммируя все вышесказанное, мы предлагаем схему общего вида (см. Рис. 3.7) для сжатия нейронных сетей в задаче моделирования языка. В первую очередь здесь сжимаются рекуррентные слои рекуррентной нейронной сети. Это можно сделать с помощью прунинга, квантизации и матричных разложений. Далее сжимаются входной и выходной слой. Это можно сделать с помощью применения байесовских методов или матричных разложений (низкоранговое, TT), как было показано в соответствующих экспериментах. Полученная сжатая модель может быть использована напрямую на мобильном устройстве. Мы разработали оптимизированную версию низкоранговой модели для LSTM, которая эффективно использует GPU современных мобильных телефонов.

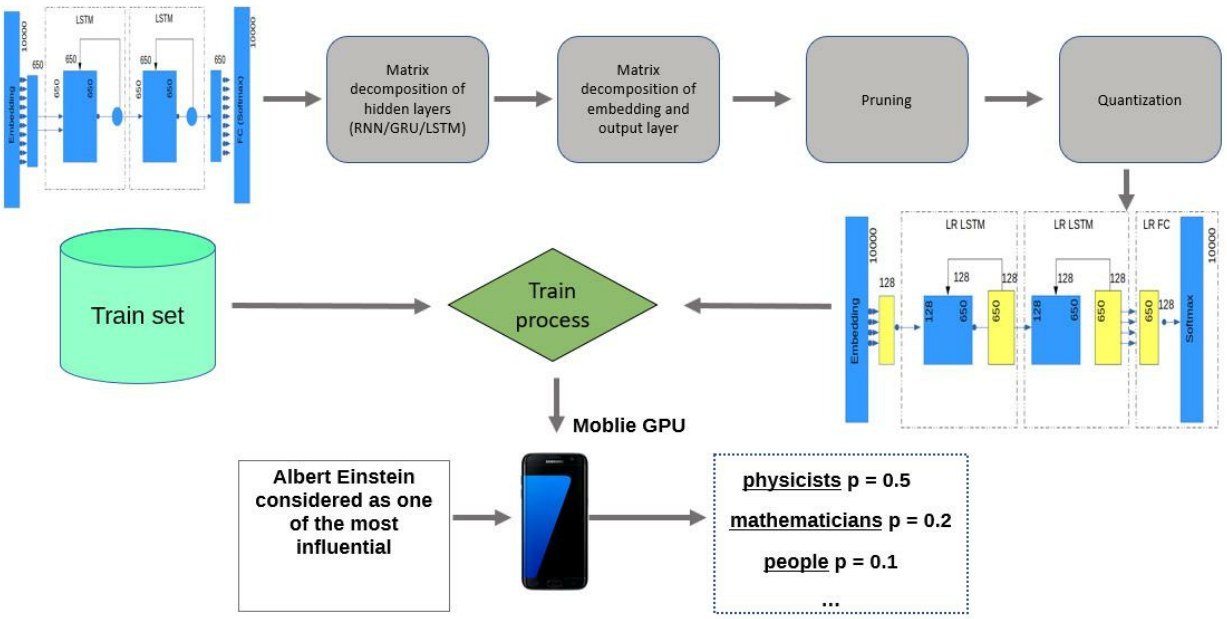


Рисунок 3.7 — Общая схема сжатия рекуррентных нейронных сетей для задачи моделирования языка

3.8 Заключение

В данной главе мы предложили несколько методов для сжатия RNN в задаче моделирования языка. Большое внимание было уделено важной проблеме высокоразмерных входных и выходных данных, которые являются критической особенностью моделирования языка из-за очень большого размера словаря. Различные методы, такие как прунинг, квантизация, низкоранговые матричные разложения, TT-разложение были экспериментально сравнены в терминах перплексии, размера модели и скорости работы на реальном мобильном устройстве. Были протестированы различные типы RNN (LSTM/GRU/RNN) и модели разных размеров (см. Таблицу 3.3).

Одним из результатов является формулировка общей схемы (Рис. 3.7) для сжатия такого типа моделей и адаптации их для переноса на мобильные устройства. Эта схема удобна для любой подобной сети с LSTM ячейками и высокоразмерными входными и выходными матрицами (Таблицы 3.1 и 3.3). На

момент публикации статей это была первая имплементация RNN (тем более сжатой) для мобильных GPU.

Было показано, что главный выигрыш, который мы получаем от сжатия рекуррентных нейронных сетей с помощью низкоранговых матричных разложений в сравнении с прунингом и квантизацией, заключается в том, что при таком подходе мы почти не теряем в скорости вычислений. Выигрыш в памяти почти напрямую переносится в выигрыш в скорости. С другой стороны, методы, которые работают с разреженными матрицами, позволяют получить большое сжатие по памяти, но долго работают при исполнении. Наши эксперименты на моделях для мобильного телефона подтверждают, что низкоранговое сжатие модели **LR LSTM 650-650** наиболее эффективно как по памяти, так и по вычислительной сложности.

Поскольку мы, в основном, изучали рекуррентные нейронные сети, некоторые SotA модели (Таблица 3.2), которые основаны либо на альтернативных модификациях рекуррентных нейронных сетей (например, Trellis network, RHN и AWD-LSTM-MOS), либо же полностью на других принципах (например, BERT [21] и модели с механизмами внимания [78]) не были протестированы в нашей схеме сжатия. Как правило, они обладают достаточно “хрупкими” архитектурами с большим количеством разных эвристик, для сжатия которых требуются дополнительные исследования. Мы предлагаем оставить это направление применения методов сжатия в качестве задач для будущих исследований. Несмотря на то что они решают похожие задачи, некоторые архитектуры гораздо сложнее, чем уже “привычными” LSTM и GRU.

Глава 4. Байесовские методы для сжатия нейронных сетей

4.1 Введение

Данная глава посвящена применению байесовских методов для сжатия нейронных сетей. Как уже было сказано, байесовские методы могут считаться математически более обоснованной формой прунинга. Сначала мы опишем общую концепцию байесовских методов, затем рассмотрим применение вариационного дропаута (Variational Dropout, VD) для задачи сжатия в языковом моделировании и наконец рассмотрим применение метода автоматического определения значимости.

Кроме того, мы впервые рассмотрим применение байесовских методов вместе с применением комбинирования весов входного-выходного слоя (weights sharing, tied weights).

4.2 Байесовский подход. Вариационная нижняя граница (ELBO)

В отличие от классических методов машинного обучения, где мы обычно ищем оценку максимального правдоподобия, в байесовском подходе мы, как правило, заинтересованы в получении полного вероятностного распределения на W – параметры модели. Пусть X, Y – обучающая выборка, $p(W)$ – априорное распределение, тогда более формально можно сказать, что мы заинтересованы в нахождении апостериорного распределения, заданного следующей формулой

(формула Байеса):

$$\mathsf{P}(W \mid X, Y) = \frac{\mathsf{P}(X, Y \mid W)\,\mathsf{P}(W)}{\mathsf{P}(X, Y)} \tag{4.1}$$

В идеальном случае, когда мы можем вычислять $p(W \mid X, Y)$, получение полного апостериорного распределения на параметры намного выгоднее, чем точечная оценка. С полным распределением на параметры мы в любой момент всё ещё можем получить их точечную оценку разных видов (среднее, мода, медиана), а также решать другие задачи, такие как обнаружение аномалий и противодействие соперничающим атакам [79] (англ. adversarial attacks). Эти задачи особенно важны в практических приложениях, когда не только хочется иметь какое-то решение, но и понимать уверенность модели в той или иной ситуации.

К сожалению, на сегодняшний день байесовские методы работают не так хорошо. Апостериорное распределение из формулы 4.1 в практических задачах не удаётся вычислять точно из-за вычислительной сложности интеграла в правой части $p(X, Y) = \int p(X, Y \mid W)p(W)\ dW$. И вместо этого широко применяются различные методы, которые позволяют аппроксимировать искомое распределение, при этом заранее сужая класс возможных апостерионых распределений.

Так, в методе вариационного байесовского вывода мы приближаем наше распределение $p(W \mid X, Y)$ с помощью некоторого параметрического класса распределений $q_\phi(W)$ и подбираем оптимальные параметры ϕ с помощью максимизации вариационной нижней границы (Variational Lower Bound или по-другому Evidence Lower Bound, ELBO):

$$\mathcal{L}_(\phi) = \mathsf{E}_{q_\phi(W)} \log p(Y|X, W) - KL(q_\phi \| p(W)) \longrightarrow \max_\phi \tag{4.2}$$

Большим недостатком такого класса методов является тот факт, что параметрический класс для $q_\phi(W)$ часто выбирается не очень широким и с предположениями, которые заведомо не выполняются на практике. (например,

веса внутри одного слоя считаются независимыми). Также большой проблемой остаётся сложность этих методов.

Тем не менее, при всех недостатках уже существуют практические приложения, в которых байесовские методы показывают себя как вполне конкурентоспособный класс методов, а иногда даже и опережающий конкурентов. Сжатие нейронных сетей как раз является таким приложением. В следующем разделе мы опишем метод вариационного дропаута для прореживания нейронных сетей согласно [13].

4.3 Вариационный дропаут для сжатия нейронных сетей

В [12] авторы рассматривали использование гауссовского шума $\xi \approx N(1, \alpha)$ в качестве дропаута в нейронных сетях. Это означает, что веса w_{ij} становятся случайными переменными:

$$w_{ij} = \theta_{ij}\xi_{ij} = \theta_{ij}(1 + \sqrt{\alpha}) \approx N(w_{ij} \mid \theta_{ij}, \alpha\theta_{ij}^2) \tag{4.3}$$

Здесь $\epsilon_{ij} \sim N(0, 1)$, а θ_{ij} – это изначальные параметры нейронной сети. Используя логнормальное распределение в качестве априорного распределения на веса:

$$p(|w_{ij}|) \propto \frac{1}{|w_{ij}|} \tag{4.4}$$

мы можем делать вариационный вывод для распределения $q(W|\theta, \alpha)$ имея в виду, что $\phi = (\theta, \alpha)$. Для слагаемого $KL(q(w_{ij})|\theta_{ij}, \alpha_{ij}\|p(w_{ij}))$ можно получить следующее выражение с точностью до константы C [12]:

$$-KL(q(w_{ij})|\theta_{ij}, \alpha_{ij}\|p(w_{ij})) = 0.5 \log \alpha_{ij} - \mathsf{E}_{\epsilon \sim N(1,\alpha_{ij})} \log |\epsilon| + C \tag{4.5}$$

В оригинальной статье [12] рассматривали $\alpha \leq 1$ и было предложено несколько различных аппроксимаций для выражения 4.5. В [13] авторы

предложили рассмотреть случай произвольных α и использовать другую аппроксимацию:

$$-KL(q(w_{i,j}|\theta_{ij},\alpha_{ij})\|p(w_{ij})) \approx k_1\sigma(k_2 + k_3 \log \alpha_{ij})) - 0.5\log(1+\alpha^{-1}) \quad (4.6)$$

Здесь $k_1 = 0.63576$, $k_2 = 1.87320$, $k_3 = 1.48695$ — специально подобранные константы. Для снижения дисперсии конечного функционала в [12] также предложили использовать специальную репараметризацию $\sigma_{ij}^2 = \alpha_{ij}\theta_{ij}^2$ и выполнять оптимизацию по параметрам (θ, σ). Теперь для $w_{i,j}$ справедливо следующее:

$$w_{ij} = \theta_{ij}(1 + \sqrt{\alpha_{ij}} \cdot \epsilon_{ij}) = \theta_{ij} + \sigma_{ij} \cdot \epsilon_{ij} \quad (4.7)$$

В экспериментах в [13] показано, что когда мы позволяем α принимать произвольные значения, часть весов получают довольно большие значения $\alpha_{ij} \longrightarrow +\infty$, что соответствует случаю, когда для этих весов вероятность дропаута $p = 1$. Это эквивалентно удалению этих весов из нейронной сети. Таким образом, можно, обучая модель, автоматически настраивать α_{ij} и проводить отсечение весов больше некоторого порога. На практике (см. [13]) удается достигать большой разреженности $(41x - 65x)$ для обычных и свёрточных нейронных сетей в задаче классификации изображений.

Для рекуррентных нейронных сетей этот метод адаптировали в [64]. Мы сравниваемся с этой работой в секции 4.5.2.

У метода вариационного дропаута также существуют некоторые проблемы. В частности возникает вопрос о закономерности использования лог-нормального априорного распределения [80].

4.4 Автоматическое определение значимости

В данной секции мы опишем ещё один способ примения байесовских методов для получения разреженных моделей. Данный метод хорошо подходит для задачи моделирования языка, потому что предполагает работу с высокоразмерными данными. Мы особенно выделяли эту проблему ранее и именно поэтому работаем с этим методом.

Начнем с описания алгоритма дважды стохастического вариационного вывода для автоматического определения значимости признаков (Doubly Stochastic Variational Inference Automatic Relevance Determination, DSVI-ARD), который изначально был предложен в [81]. Для этого опишем проблему многоклассовой классификации скозь призму байесовского подхода. Метод, изложенный далее, представляет нам возможность автоматически выбирать значимые признаки при решение задачи классификации. А использование двойного стохастичекого вывода для оптимизации позволяет масштабировать этот алгоритм на большие размерности.

Пусть у нас есть набор данных из N независимых объектов $(X, Y) = \{(x_n, y_n)\}_{n=1}^{N}$. Рассмотрим дискриминативную вероятностную модель

$$\begin{aligned} p(W, Y \mid X, \Lambda) &= p(W \mid \Lambda) p(Y \mid W, X) = \\ &= \left[\prod_{i=1}^{D} \prod_{j=1}^{K} \mathcal{N}\left(w_{ij} \mid 0, \lambda_{ij}\right) \right] \prod_{n=1}^{N} \operatorname{softmax}\left(W x_n\right)[y_n] = \\ &= \left[\prod_{i,\,j}^{D,\,K} \mathcal{N}\left(w_{ij} \mid 0, \lambda_{ij}\right) \right] \prod_{n=1}^{N} \operatorname{softmax}\left(W x_n\right)[y_n], \end{aligned} \tag{4.8}$$

здесь $W \in \mathbb{R}^{D \times K}$ — это матрица параметров модели, $x_n \in \mathbb{R}^D$ – это вектор признаков, который описывает текущий объект, $t \in \{1, \ldots, K\}$ – метка класса объекта и $\Lambda \in \mathbb{R}^{D \times K}$ — это матрица гиперпараметров, определяющих априорное распределение $p(W \mid \Lambda)$ для параметров W. Порядок, в котором умножаются случайные величины, не важен, так как мы на самом деле работаем с матрицей

весов и все веса в этом смысле равноправны. Поэтому для сокращения нотации мы ввели следующее обозначение:

$$\prod_{i,\,j}^{D,\,K} = \prod_{i=1}^{D}\prod_{j=1}^{K} = \prod_{j=1}^{K}\prod_{i=1}^{D}$$

И аналогично для суммы $\sum$.

Априорное распределение $p(W \mid \Lambda)$ — это поэлементное факторизованное нормальное распределение для каждого w_{ij} с нулевым средним и дисперсией λ_{ij}. Функция правдоподобия $p(y \mid W, \mathbf{x})$ (плотность распределения над всеми возможными классами объекта при условии их описания $\mathbf{x}$ и параметров W) это вектор из t элементов, полученный в результате применения софтмакс преобразования $\text{softmax}(\mathbf{a}) = \frac{1}{\sum_{i=1}^{K} e^{a_i}} \left(e^{a_1}, \ldots, e^{a_n}\right)^T$, $\mathbf{a} \in \mathbb{R}^K$ к скалярному произведению Wx.

Во всём последующем выводе мы преследуем две цели. Первая – это вывод апостериорного распределения для параметров модели при условии обучающей выборки:

$$p(W \mid X, Y, \Lambda) = \frac{p(W, Y \mid X, \Lambda)}{p(Y \mid X, \Lambda)}. \tag{4.9}$$

Вторая – это выбор оптимальной модели, то есть выбор оптимальных гиперпараметров. Вычисление апостериорного распределения напрямую невозможно. Как уже говорилось, интеграл $p(Y \mid X, \Lambda) = \int p(W, Y \mid X, \Lambda) dW$ невычислим. На сегодняшний день популярным является метод аппроксимации апостериорного распределения с помощью максимизации ELBO:

$$\begin{aligned} &\mathcal{L}(q, \Lambda) = \log p(Y \mid X, \Lambda) - KL\left(q(W) \| p(W \mid X, Y, \Lambda)\right) = \\ &= \mathbb{E}_{W \sim q(W)} \left[\log p(Y \mid W, X)\right] - KL\left(q(W) \| p(W \mid \Lambda)\right. \end{aligned} \tag{4.10}$$

В данной модели ELBO – это функция двух переменных: произвольного вариационного распределения над параметрами модели $q(W)$ и гиперпараметрами Λ. Заметим, что $\mathcal{L}(q, \Lambda) \leq \log p(Y \mid X, \Lambda), \forall q, \Lambda$ и $\mathcal{L}(q, \Lambda) = \log p(Y \mid X, \Lambda) \Leftrightarrow q(W) = p(W \mid X, Y, \Lambda)$, то есть максимизация ELBO по q для фиксированных Λ эквивалентна тому, что мы приближаем распределение q к

нужному нам апостериорному распределению $p(W \mid X, Y, \Lambda)$, таким образом решая первую проблему.

Максимизация обоснованности $p(Y \mid X, \Lambda)$ по гиперпараметрам Λ широко используется как байесовский выбор модели и также известна как эмпирический Байес (empirical Bayes [82]). Модель с наибольшим значением обоснованности рассматривается как "лучшая" в терминах подстройки под данные и сложности модели, что решает вторую упомянутую проблему. Мы предлагаем [82] для дальнейшего ознакомления с этой концепцией. Максимизация обоснованности может быть выполнена через максимизацию ELBO в следующем виде:

$$\begin{aligned} &\max_{\Lambda} \log p(Y \mid X, \Lambda) = \max_{\Lambda} \max_{q} \mathcal{L}(q, \Lambda) = \\ &= \max_{q, \Lambda} \mathcal{L}(q, \Lambda) = \max_{q} \max_{\Lambda} \mathcal{L}(q, \Lambda) \end{aligned} \tag{4.11}$$

Эта оптимизационная процедура решает проблему выбора модели и также наилучшим образом приближает q к апостериорному распределению. Рассмотрим подробнее функционал ELBO (4.10). Видно, что в нём только KL-слагаемое $KL\left(q(W) \| p(W \mid \Lambda)\right.$ зависит от Λ и следовательно:

$$\mathcal{L}(q, \Lambda) \longrightarrow \max_{\Lambda} \Longleftrightarrow KL\left(q(W) \| p(W \mid \Lambda)\right. \longrightarrow \min_{\Lambda}.$$

Зафиксируем класс произвольного вариационного распределения q. Пусть это будет факторизованное нормальное распределение, то есть:

$$q(W \mid \boldsymbol{\mu}, \boldsymbol{\sigma}) = \prod_{i,\,j}^{D,\,K} \mathcal{N}(w_{ij} \mid \mu_{ij}, \sigma_{ij}^2), \tag{4.12}$$

где $\boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^{D \times K}$ *вариационные параметры*. После этого распишем KL:

$$KL\left(q(W)\|p(W \mid \Lambda)\right) = KL\left(\prod_{i,\,j}^{D,\,K} \mathcal{N}(w_{ij} \mid \mu_{ij}, \sigma_{ij}^2) \middle\| \prod_{i,\,j}^{D,\,K} \mathcal{N}\left(w_{ij} \mid 0, \lambda_{ij}\right)\right) = \sum_{i,\,j}^{D,\,K} \left(\log \frac{\lambda_{ij}}{\sigma_{ij}} + \frac{\sigma_{ij}^2 + \mu_{ij}^2}{2\lambda_{i,j}^2}\right) \longrightarrow \min_{\lambda_{ij}} \tag{4.13}$$

Как можно видеть, в данном случае KL-дивергенция распадается на сумму, и каждое λ_{ij} входит только в одно слагаемое. Для нахождения точки минимума возьмём частную производную по λ_{ij}:

$$\begin{aligned} \frac{\partial KL\left(q(W)\|p(W \mid \Lambda)\right)}{\partial \lambda_{ij}} &= \left(\sum_{i,\,j}^{D,\,K} \left(\log \frac{\lambda_{ij}}{\sigma_{ij}} + \frac{\sigma_{ij}^2 + \mu_{ij}^2}{2\lambda_{i,j}^2}\right)\right)'_{\lambda_{ij}} = \\ &= \frac{1}{\lambda_{ij}} - \frac{\sigma_{ij}^2 + \mu_{ij}^2}{\lambda_{i,j}^3} \end{aligned} \tag{4.14}$$

Последнее равенство верно в силу того, что производная всех остальных слагаемых по λ_{ij} равна нулю. Полученное выражение приравниваем к нулю и получаем аналитическое решение для всех λ_{ij}:

$$\lambda_{ij}^* = \mu_{ij}^2 + \sigma_{ij}^2, \tag{4.15}$$

Это верно для всех $i = 1, \ldots, D$ и $j = 1, \ldots, K$. Теперь мы можем подставить полученное $\Lambda^* = \{\lambda_{ij}\}_{i,j}^{D,K}$ в уравнение для ELBO (4.10):

$$\begin{aligned} &\mathcal{L}\left(q\left(W \mid \boldsymbol{\mu}, \boldsymbol{\sigma}\right), \Lambda^*\right) = \mathbb{E}_{W \sim q(W|\boldsymbol{\mu},\boldsymbol{\sigma})}\left[\log p(Y \mid W, X)\right] + \\ &+ \frac{1}{2} \sum_{i=1}^{D} \sum_{j=1}^{K} \log \frac{\sigma_{ij}^2}{\mu_{ij}^2 + \sigma_{ij}^2} \longrightarrow \max_{\boldsymbol{\mu},\boldsymbol{\sigma}} \end{aligned} \tag{4.16}$$

Это финальное выражение для метода релевантных векторов. Мы можем видеть, что первое слагаемое, называемое data-term, отвечает за то, чтобы вариационные параметры хорошо описывали наблюдаемые данные, вынуждая

вариционные параметры консолидироваться в точке максимума правдоподобия. В то же время второе слагаемое (KL-слагаемое) обрезает ненужные веса, потому что, если $\mu_{ij}^2 + \sigma_{ij}^2 \longrightarrow 0$, то отвечающий им параметр w_{ij} не несёт практической пользы, и поэтому может быть удалён из модели.

Мы применили такую схему к нашей задаче следующим образом. В начале мы зафиксировали всю нейронную сеть и заново обучили только выходной слой с помощью описанного выше метода. То есть на вход этому алгоритму мы подавали выходы последнего рекуррентного слоя нейронной сети и предсказывали класс следующего слова. Далее мы соединили эту процедуру с обучением всей остальной нейронной сети. При этом качество осталось на таком же уровне или даже стало лучше. Подробнее про процедуру обучения и полученные результаты будет рассказано в следующей секции.

Таблица 4.1

Сравнение DSVI-ARD для **LSTM** из [52] на датасетах PTB и Wikitext.

Датасет	**Архитектура**	**Кол-во парам., млн (Все / Софтмакс)**	**Удалённые веса, % (Все / Софтмакс)**	**Перплексия**	**Точность, %**
PTB	*Original*	19.8 / 6.5	No compression	80.85	27.4%
	DSVI-ARD ***(ours)***	**13.4** / **0.14**	32.1% / 97.8%	91.84	27.2%
	LR for Softmax (Глава 3)	14.5 / 1.19	26.8 % / 81.7 %	84.12	N/A
	TT for Softmax (Глава 3)	14.3 / 1.03	27.8 % / 84.2 %	88.55	N/A
Wikitext2	*Original*	50.1 / 21.6	No compression	94.27	27.5%
	DSVI-ARD ***(ours)***	**28.9** / **0.43**	42.3% / 98.0%	103.54	**27.6%**

Таблица 4.2

Сравнение DSVI-ARD для **LSTM + tied weights** [56] на датасетах PTB и Wikitext.

Датасет	**Архитектура**	**Кол-во парам., млн (Все / Софтмакс)**	**Удалённые веса, % (Все / Софтмакс)**	**Перплексия**	**Точность, %**
PTB	*Original*	13.3 / 6.5	No compression	75.68	27.7%
	DSVI-ARD ***(ours)***	**7.4** / **0.66**	**44.0%** / **89.9%**	82.27	27.3%
Wikitext2	*Original*	28.4 / 21.6	No compression	86.62	27.9%
	DSVI-ARD ***(ours)***	**8.7** / **1.94**	**69.3%** / **91.0%**	87.36	**28.1%**

Таблица 4.3

Сравнение DSVI-ARD для **LSTM** с архитектурой из [64] на датасете PTB.

Датасет	**Архитектура**	**Кол-во парам., млн (Все / Софтмакс)**	**Удалённые веса, % (Все / Софтмакс)**	**Перплексия**	**Точность, %**
PTB	*Original*	5.64 / 2.56	No compression	129.3	N/A
	VD, [64]	3.2 / 0.12	43.3 % / 95.5 %	109.2	N/A
	DSVI-ARD (Ours)	**3.18** / **0.1**	**43.6 %** / **96.1 %**	**106.2**	25.9 %

4.5 Детали экспериментов и результаты

4.5.1 Обучение моделей и оценка качества

Все параметры модели со слоями DSVI-ARD могут быть разделены на вариационные параметры $\boldsymbol{\mu}, \boldsymbol{\sigma}$ и на все остальные параметры сети (включая векторы сдвига для слоёв DSVI-ARD). Вариационная оптимизация выполняется соответственно для вариациационных параметров. Делается это с помощью DSVI-ARD алгоритма 5, которому требуются значения градиентов функции логарифма правдоподобия модели и KL-дивергенции. Таким образом, обучение модели – это стандартная градиентная оптимизация, с отрицательным ELBO как функцией потерь.

Для более эффективного обучения мы применили технику отжига коэффициента перед KL частью [83]. Идея заключается в том, чтобы во время обучения на каждой итерации умножать KL-слагаемое в уравнении ELBO на некоторый вес. Этот вес (будем называть его KL-весом) постепенно увеличивается от нуля до единицы или какого-то другого фиксированного положительного значения в течение первых нескольких эпох. Эта процедура позволяет достичь лучшего качества модели, потому что в некотором смысле такая процедура может считаться предобучением модели, так как слагаемое data term является доминирующим в уравнении ELBO на этих шагах. А после такого предобучения продолжается обычная “честная” оптимизация полного ELBO (если

Algorithm 5 Двойной стохастический вариационный вывод

Входные данные: логарифм функции правдоподобия $\log p(y \mid W, \mathbf{x})$, обучающая выборка (X, Y) размера N, параметры скорости обучения $\{\rho_k\}$, размер мини-батча M

Инициализировать вариационные параметры $\boldsymbol{\mu}^{(0)}, \boldsymbol{\sigma}^{(0)}$, $k = 0$

repeat

$k = k + 1$

Сэмплировать мини-батч

$\{\boldsymbol{x}_m, y_m\}_{m=1}^{M} \subseteq (X, Y)$

for all объектов $(\boldsymbol{x}_m, y_m)$ в мини-батче **do**

Sample $\boldsymbol{\epsilon}_m \sim \mathcal{N}(\mathbf{0}, I), \boldsymbol{\epsilon}_m \in \mathbb{R}^{K \times D}$

$W_m \coloneqq \boldsymbol{\mu}^{(k-1)} + \boldsymbol{\sigma}^{(k-1)} \odot \boldsymbol{\epsilon}_m$

end for

$g_{\boldsymbol{\mu}}^{Data} \coloneqq \frac{N}{M} \sum_{m=1}^{M} \nabla_{\boldsymbol{\mu}} \log p(y_m \mid W_m, \boldsymbol{x}_m)$

$g_{\boldsymbol{\sigma}}^{Data} \coloneqq \frac{N}{M} \sum_{m=1}^{M} \nabla_{\boldsymbol{\sigma}} \log p(y_m \mid W_m, \boldsymbol{x}_m)$

$g_{\boldsymbol{\mu}}^{KL} \coloneqq \nabla_{\boldsymbol{\mu}} \left[\frac{1}{2} \sum_{i=1}^{K} \sum_{j=1}^{D} \log \frac{\sigma_{ij}^2}{\mu_{ij}^2 + \sigma_{ij}^2} \right] \Big|_{\substack{\boldsymbol{\mu}=\boldsymbol{\mu}^{(k-1)} \\ \boldsymbol{\sigma}=\boldsymbol{\sigma}^{(k-1)}}}$

$g_{\boldsymbol{\sigma}}^{KL} \coloneqq \nabla_{\boldsymbol{\sigma}} \left[\frac{1}{2} \sum_{i=1}^{K} \sum_{j=1}^{D} \log \frac{\sigma_{ij}^2}{\mu_{ij}^2 + \sigma_{ij}^2} \right] \Big|_{\substack{\boldsymbol{\mu}=\boldsymbol{\mu}^{(k-1)} \\ \boldsymbol{\sigma}=\boldsymbol{\sigma}^{(k-1)}}}$

$g_{\boldsymbol{\mu}} \coloneqq g_{\boldsymbol{\mu}}^{Data} + g_{\boldsymbol{\mu}}^{KL}$

$g_{\boldsymbol{\sigma}} \coloneqq g_{\boldsymbol{\sigma}}^{Data} + g_{\boldsymbol{\sigma}}^{KL}$

$\boldsymbol{\mu}^{(k)} = \boldsymbol{\mu}^{(k-1)} + \rho_k g_{\boldsymbol{\mu}}$

$\boldsymbol{\sigma}^{(k)} = \boldsymbol{\sigma}^{(k-1)} + \rho_k g_{\boldsymbol{\sigma}}$

until не выполнен критерий сходимости

KL-вес равен единице). Для экспериментов мы использовали простую стратегию линейного увеличения KL-веса. Скорость увеличения подбиралась как гиперпараметр.

Во время тестирования моделей DSVI-ARD на тестовой части выборки мы не сэмплируем веса, как мы это делаем во время обучения, а берем среднее $\boldsymbol{\mu}$ из апостериорного распределения на весах. Далее мы обнуляем веса, у которых логарифм априорной вероятности ниже фиксированного порога отсечения $\log \lambda_{thresh}$ (гиперпараметр, который мы подбираем на валидационной части выборки): $\log \lambda_{ij}^{*} = \log \left(\mu_{ij}^2 + \sigma_{ij}^2 \right) < \log \lambda_{thresh}$, что обеспечивает нам разреженность слоя, так как обнулённые веса в прямом смысле удаляются из нейронной сети.

Algorithm 6 Двойной стохастический вариационный вывод для зависимых данных

Входные данные: логарифм функции правдоподобия $\log p(y \mid W, \mathbf{x})$, обучающая выборка (X, Y) размера N, параметры скорости обучения $\{\rho_k\}$, размер мини-батча M

Инициализировать вариационные параметры $\boldsymbol{\mu}^{(0)}, \boldsymbol{\sigma}^{(0)}$, $k = 0$

repeat

$k = k + 1$

Сэмплировать мини-батч

$\{\boldsymbol{x}_m, y_m\}_{m=1}^{M} \subseteq (X, Y)$ не н. о. р. объектов

Сэмплируем $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I), \boldsymbol{\epsilon} \in \mathbb{R}^{K \times D}$

$W \coloneqq \boldsymbol{\mu}^{(k-1)} + \boldsymbol{\sigma}^{(k-1)} \odot \boldsymbol{\epsilon}$

$g_{\boldsymbol{\mu}}^{Data} \coloneqq \frac{N}{M} \sum_{m=1}^{M} \nabla_{\boldsymbol{\mu}} \log p(y_m \mid W, \boldsymbol{x}_m)$

$g_{\boldsymbol{\sigma}}^{Data} \coloneqq \frac{N}{M} \sum_{m=1}^{M} \nabla_{\boldsymbol{\sigma}} \log p(y_m \mid W, \boldsymbol{x}_m)$

$g_{\boldsymbol{\mu}}^{KL} \coloneqq \nabla_{\boldsymbol{\mu}} \left[\frac{1}{2} \sum_{i=1}^{K} \sum_{j=1}^{D} \log \frac{\sigma_{ij}^2}{\mu_{ij}^2 + \sigma_{ij}^2} \right] \Big|_{\substack{\boldsymbol{\mu}=\boldsymbol{\mu}^{(k-1)} \\ \boldsymbol{\sigma}=\boldsymbol{\sigma}^{(k-1)}}}$

$g_{\boldsymbol{\sigma}}^{KL} \coloneqq \nabla_{\boldsymbol{\sigma}} \left[\frac{1}{2} \sum_{i=1}^{K} \sum_{j=1}^{D} \log \frac{\sigma_{ij}^2}{\mu_{ij}^2 + \sigma_{ij}^2} \right] \Big|_{\substack{\boldsymbol{\mu}=\boldsymbol{\mu}^{(k-1)} \\ \boldsymbol{\sigma}=\boldsymbol{\sigma}^{(k-1)}}}$

$g_{\boldsymbol{\mu}} \coloneqq g_{\boldsymbol{\mu}}^{Data} + g_{\boldsymbol{\mu}}^{KL}$

$g_{\boldsymbol{\sigma}} \coloneqq g_{\boldsymbol{\sigma}}^{Data} + g_{\boldsymbol{\sigma}}^{KL}$

$\boldsymbol{\mu}^{(k)} = \boldsymbol{\mu}^{(k-1)} + \rho_k g_{\boldsymbol{\mu}}$

$\boldsymbol{\sigma}^{(k)} = \boldsymbol{\sigma}^{(k-1)} + \rho_k g_{\boldsymbol{\sigma}}$

until не выполнен критерий сходимости

Каждый эксперимент устроен следующим образом. Мы обучаем несколько моделей на обучающей выборке с различными гиперпараметрами (вероятность дропаута, скорость обучения, расписание изменения скорости обучения и так далее). После этого мы выбираем лучшую модель по кросс-энтропии на валидационной выборке. Мы не обнуляем веса на этой стадии, то есть устанавливается порог отсечения $\log \lambda_{thresh} = -\infty$. После выбора лучшей модели мы перебираем значения $\log \lambda_{thresh}$: от минимального, при котором мы не удаляем ни один вес, до максимального, при котором мы удаляем все веса и выбираем лучший (обозначим его $\log \lambda_{thresh}^{opt}$) в терминах перплексии на валидационной выборке. После этого мы оцениваем достигаемый уровень сжатия следующим образом:

$$cr = \frac{\sum_{i=1}^{K} \sum_{j=1}^{D} \mathbb{1}[\log \lambda_{ij}^{*} < \log \lambda_{thresh}^{opt}]}{total\_parameters}, \tag{4.17}$$

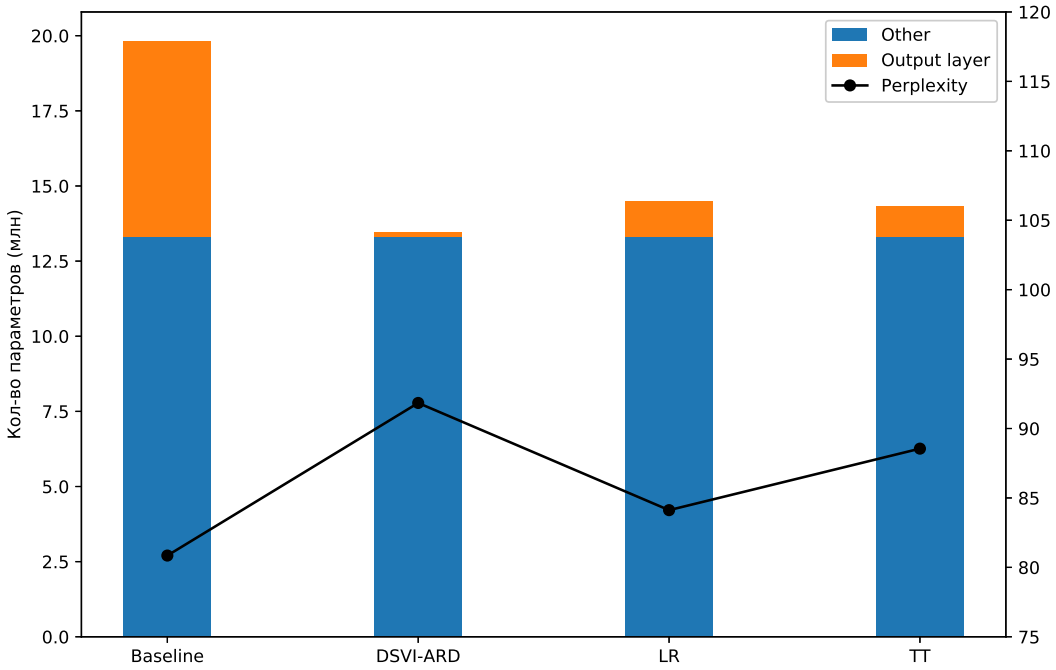


Рисунок 4.1 — Диаграмма сжатия сети с помощью алгоритма DSVI-ARD (LSTM -650-650, датасет PTB)

а также перплексию и точность на валидационной и тестовой выборке для той модели, которая у нас получается при выбранном пороге отсечения $\log \lambda^{opt}_{thresh}$.

4.5.2 Результаты экспериментов

Таблицы 4.1-4.3 содержат результаты экспериментов. Сравнение DSVI-ARD с другими подходами для сжатия полносвязных слоёв показывает, что наши модели могут достичь сравнимого качества перплексии при достижении гораздо более высокого уровня сжатия (например, в сравнении с матричным случаем) и даже превзойти некоторые модели, которые используют похожие Байесовские техники (см. сравнение с [64]).

Можно заметить, что метод совместного использования весов (weight tying) помогает достичь лучшего качества сжатия (с приблизительно 45% до 70% уменьшения всех весов в модели) ввиду того, что мы теперь автоматически

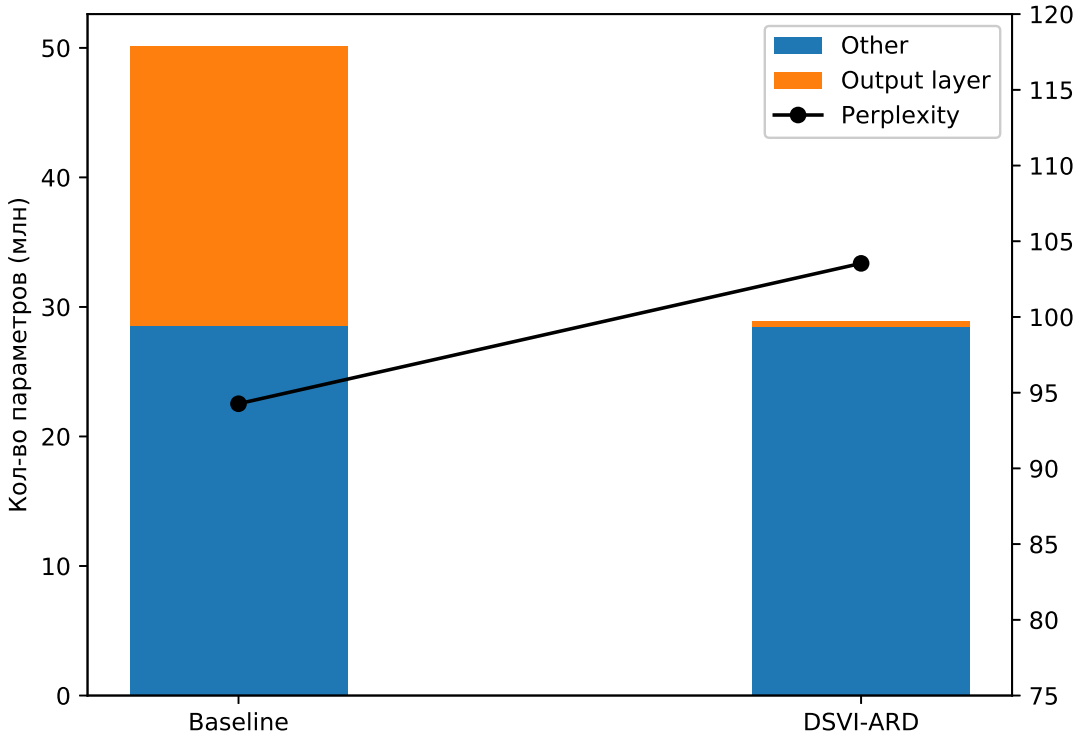


Рисунок 4.2 — Диаграмма сжатия сети с помощью алгоритма DSVI-ARD (LSTM -650-650, датасет WikiText-2)

сжимаем оба слоя нейронной сети. В то же время улучшается и перплексия, и точность[1] на обоих датасетах.

Есть исследования [84], которые показывают, что байесовские методы могут вести к излишней разреженности. В наших экспериментах также наблюдается некоторая потеря качества в терминах перплексии. Поэтому мы дополнительно изучили, как ведет себя точность на тестовой выборке. По этой метрике наши модели достигают такого же или даже лучшего качества в сравнении с оригинальными моделями. Это может говорить о том, что потеря качества в модели также может быть связана с неопределенностью распределения (с энтропией), а не с деградацией модели.

На Рис. 4.5 (вверху) изображен график кросс-энтропии и зависимость полученной разреженности при разных порогах отсечения для $\log \lambda_{thresh}$. Внизу

[1]Под точностью здесь понимается доля верно предсказанных слов на тестовой части выборки. Предсказание делается с помощью операции argmax по всему распределению на словаре, которое возникает после софтмакса

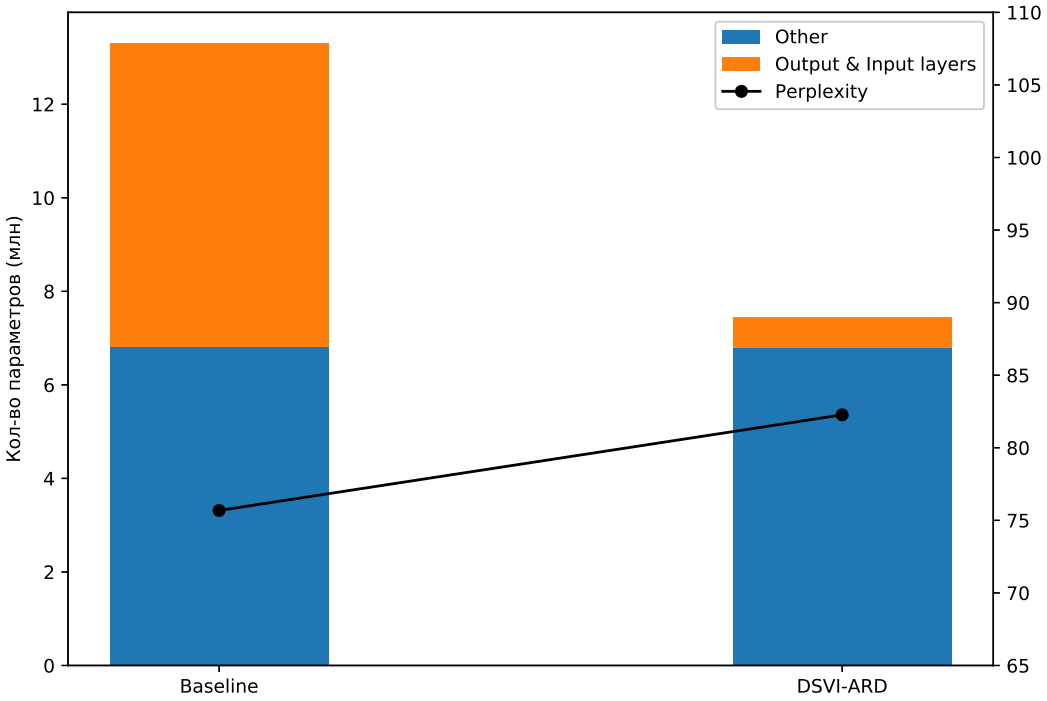


Рисунок 4.3 — Диаграмма сжатия сети с помощью алгоритма DSVI-ARD (LSTM -650-650, датасет PTB, tied weights)

приведена гистограмма распределения (в логарифмической шкале) логарифмов дисперсий $\log \lambda_{ij}^*$ априорного распределения полученного в результате обучения модели DSVI-ARD LSTM. Зеленая пунктирная линия показывает уровень отсечения, который был выбран и который соответствует лучшему значению перплексии на обучающей выборке. Для наглядности мы также приводим гистограмму распределения в стандартном масштабе на Рис. 4.6. Можно заметить, что согласно полученному распределению большинство весов в выходном слое считаются ненужными, имеющими маленькую априорную дисперсию (обратим внимание еще раз, что плотность изображена в логарифмической шкале) и не вносящими большого влияния на финальный результат и соответственно могут быть удалены без потери качества, а иногда даже с его увеличением. Мы предполагаем, что модель DSVI-ARD удаляет веса, которые препятствуют обощающей способности модели, и оставляет те, которые действительно необходимы для предсказания.

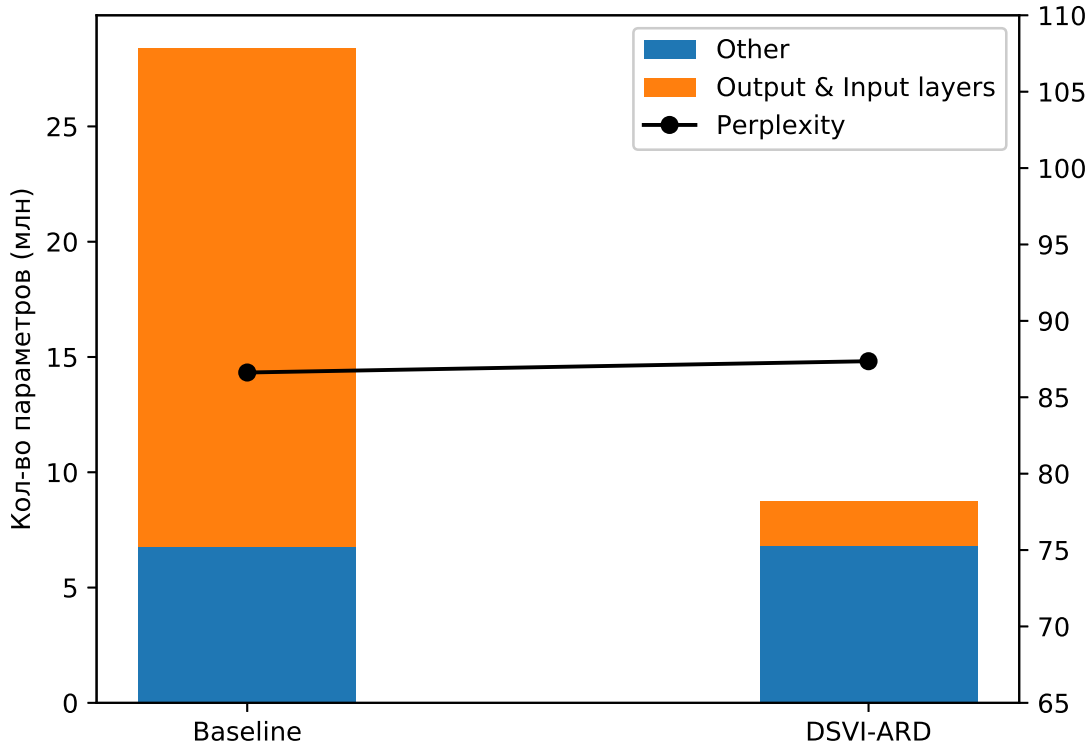


Рисунок 4.4 — Диаграмма сжатия сети с помощью алгоритма DSVI-ARD (LSTM -650-650, датасет WikiText-2, tied weights)

4.6 Выводы

В данной главе мы рассмотрели применение байесовских методов для сжатия рекуррентных нейронных сетей в задаче моделирования языка. Мы сконцентрировались на работе с входными и выходными слоями, которые в этой задаче имеют особое значение.

Из результатов экспериментов, представленых в таблицах 4.1-4.3, видно, что удаётся добиться сжатия выходного слоя до 50 раз в сравнении с изначальной моделью (удаляем 98% параметров). Это больше, чем с помощью методов матричных разложений, но при худшей перплексии. На рисунках 4.1–4.4 показаны гистограммы сжатия моделей для различных датасетов, а также для варианта с использованием общих весов для входного и выходного слоя и с разными весами для этих слоёв.

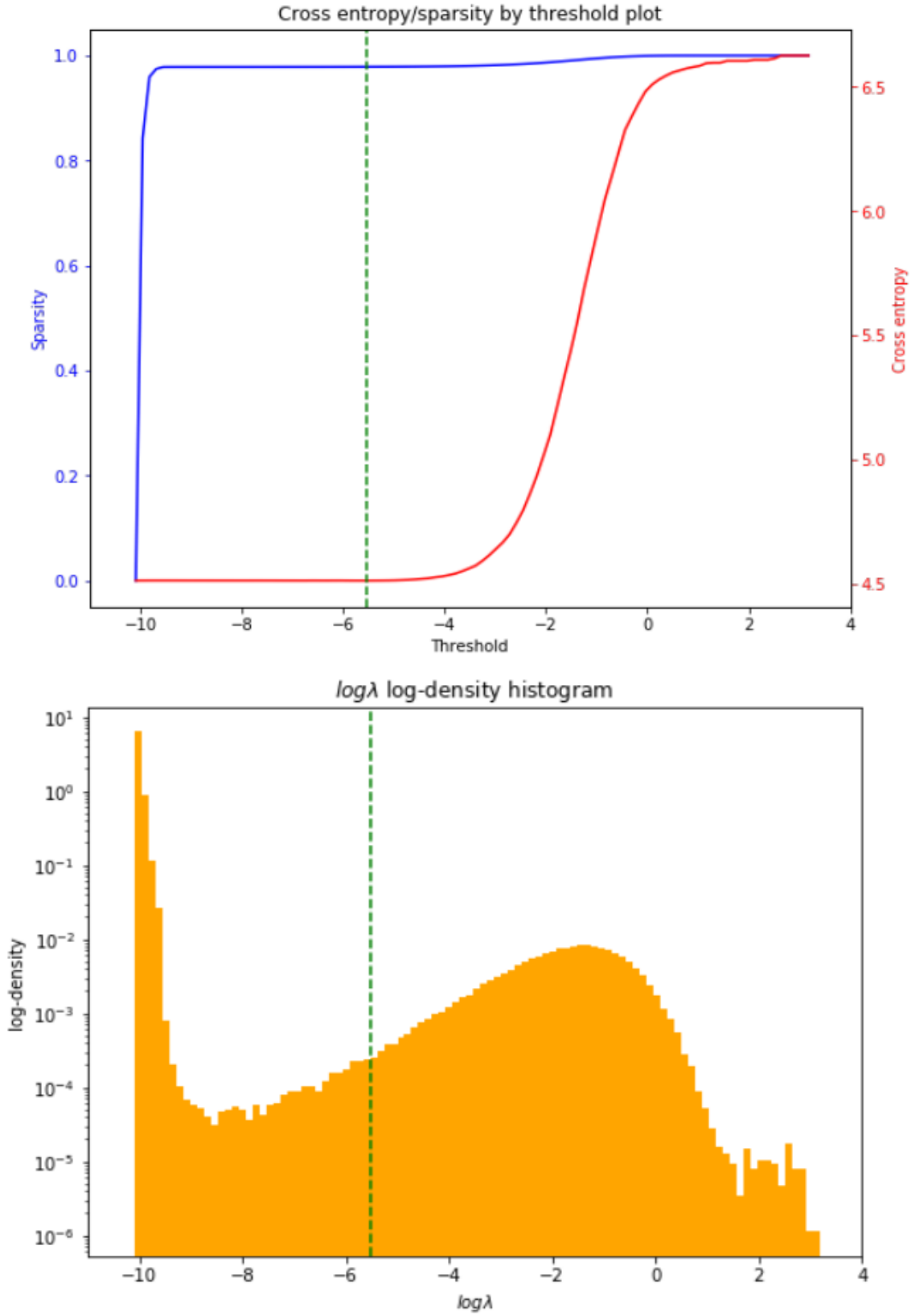


Рисунок 4.5 — Кросс-энтропия на валидационной выборке и соответствующий уровень разреженности (уровень сжатия) софтмакс слоя для различных возможных уровней отсечения $\log \lambda_{thresh}$ (**наверху**) и гистограмма распределения априорных логарифмов дисперсии $\log \lambda_{ij}$ (**снизу**), полученных для LSTM модели в DSVI-ARD софтмакс слое на датасете PTB. Уровень отсечения показан зелёным цветом (выбирается на валидационной части выборки). Мы показываем плотность в **логарифмическом масштабе** так как распределение очень вырожденное.

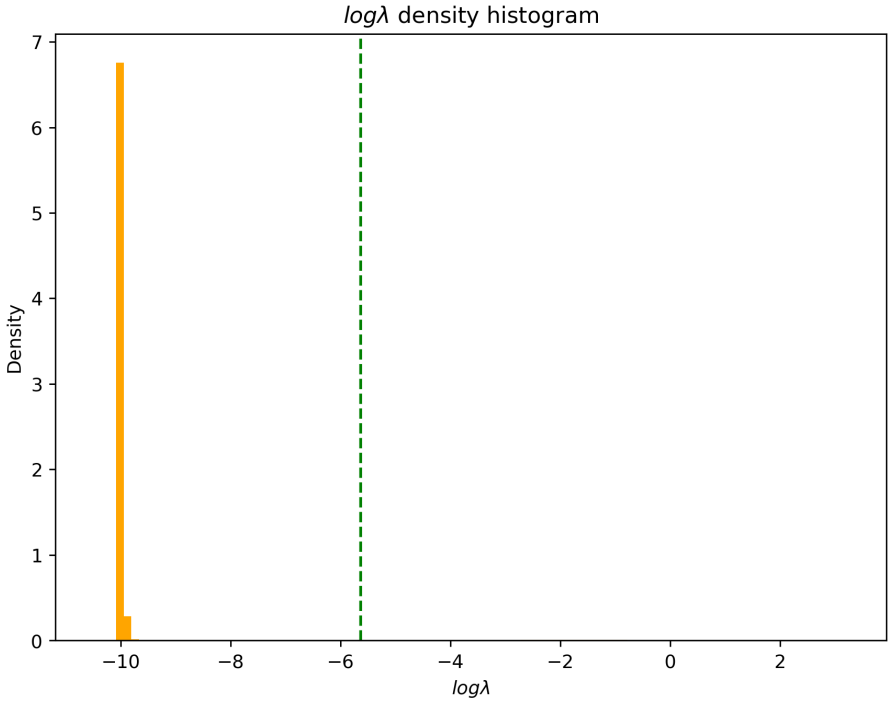


Рисунок 4.6 — Гистограмма распределения априорных логарифмов дисперсии $\log \lambda_{ij}$ (**снизу**) полученными для LSTM модели в DSVI-ARD софтмакс слое на датасете PTB. Уровень отсечения показан зелёным цветом (выбирается на валидационной части выборки). На этом рисунке мы изобразили плотность в стандартном масштабе для того, чтобы показать, насколько вырождено распределение и сравнить с Рис. 4.5.

Байесовские методы сжатия действительно позволяют достичь большего разреживания, чем, например, матричные методы сжатия, но остаётся открытым вопрос эффективной имплементации таких методов для конкретных устройств.

Решением этой проблемы могло бы быть использование структурного сжатия, как, например, в [14]. К сожалению, в наших экспериментах использование этого подхода дало сильное ухудшение в качестве моделей. Скорее всего, это связано с искуственным ограничением на класс априорных и апостериорных распределений. Заметим, что уже здесь, в обычном DSVI-ARD подходе мы ищем аппроксимацию апостериорного распределения в классе многомерного нормального распределения с независимыми компонентами. То есть, в нашу модель изначально встроено предположение о независимости весов, что, конечно же, скорее всего не выполняется на практике. Подбор таких распределений,

которые бы учли предположение о зависимости весов и получение для них моделей, которые вычислимы на практике, остаётся направлением для будущих исследований.

Заключение

Повсеместное распространение мобильных устройств, а также большое число задач, где нейронные сети показывают результаты лучше, чем другие модели, делают задачу сжатия нейронных сетей очень востребованной. В данном исследовании мы предложили несколько методов сжатия класса рекуррентных нейронных сетей в задаче моделирования языка. Мы изучили простейшие методы такие, как прунинг и квантизации. Далее мы рассмотрели матричные сжатия и предложили свои адаптации для задачи сжатия рекуррентных нейронных сетей. Наконец, завершающая глава была посвящена применению байесовских методов для этой задачи.

Большое внимание было уделено важной проблеме высокоразмерных входных и выходных данных, которые являются критической особенностью моделирования языка из-за очень большого размера словаря.

В экспериментальной части диссертации было показано, что главный выигрыш, который мы получаем от сжатия рекуррентных нейронных сетей с помощью низкоранговых матричных разложений в сравнении с прунингом, квантизацией, а также с байесовским прунингом, заключается в том, что при таком подходе мы почти не теряем в скорости вычислений. Выигрыш в памяти почти напрямую переносится в выигрыш в скорости работы. С другой стороны, байесовское сжатие, а также обычный прунинг работают с разреженными матрицами. Благодаря этим методам можно получить большое сжатие по памяти, но такие модели будут долго работать во время вычислений, если только не использовать специальные процессоры, адаптированные к работе с разреженными матрицами.

Важным результатом диссертации является формулировка общей схемы (Рис. 3.7) для сжатия языковых нейросетевых моделей и их адаптации для переноса на мобильные устройства.

Основные результаты работы заключаются в следующем:

1. Были разработаны алгоритмы сжатия скрытых слоев нейронных сетей для задачи моделирования языка с помощью методов матричных разложений.
2. Было предложено эффективное решение задачи классификации для большого числа классов (10–30 тыс.) с использованием методов матричных разложений для входного и выходного слоев нейронной сети.
3. Алгоритм DSVI-ARD был адаптирован для рекуррентных нейронных сетей в задаче моделирования языка.
4. Эффективные реализации рекуррентных нейронных сетей были портированы на мобильные устройства с использованием мобильных GPU и было проведено лабораторное тестирование в максимально приближенном к реальной жизни сценарии.

В качестве направлений будущих исследований можно выделить следующие. Во-первых, это работа по сжатию современных SotA моделей (Таблица 3.2). Как уже было упомянуто, многие новейшие архитектуры, например, модели на основе механизмов внимания, BERT, Trellis Networks имеют большие отличия в сравнении с уже широко использующимися рекуррентными моделями. Поэтому для их сжатия могут потребоваться существенные доработки имеющихся методов, хотя некоторые принципы могут быть перенесены сразу же (например, работа с высокоразмерными входными и выходными данными). Во-вторых, это дальнейшее развитие матричных и байесовских методов сжатия. В байесовских методах перспективной является разработка алгоритмов, работающих с более гибкими классами распределений. Другим интересным направлением кажется объединение байесовских и матричных методов для создания моделей, в которых применяется низкоранговое матричное, TT-разложение или какое-либо ещё, а параметры этих разложений автоматически подбираются с помощью байесовских методов.

Список литературы

1. *Schmidhuber Jurgen.* Deep Learning in Neural Networks: An Overview // *Neural Networks.* — 2015. — Vol. 61. — Pp. 85–117.

2. *Bengio Yoshua.* Learning deep architectures for AI // *Foundations and Trends in Machine Learning.* — 2009. — Vol. 2, no. 1. — Pp. 1–127. — Also published as a book. Now Publishers, 2009.

3. *Deng Li, Yu Dong.* Deep Learning: Methods and Application // *Foundations and Trends in Signal Processing.* — 2013. — Vol. 7. — Pp. 197–387.

4. *Goodfellow Ian, Bengio Yoshua, Courville Aaron.* Deep Learning. — MIT Press, 2016. — http://www.deeplearningbook.org.

5. *Cybenko George.* Approximation by superpositions of a sigmoidal function // *MCSS.* — 1989. — Vol. 2, no. 4. — Pp. 303–314. https://doi.org/10.1007/BF02551274.

6. *Hornik Kurt, Stinchcombe Maxwell B., White Halbert.* Multilayer feedforward networks are universal approximators // *Neural Networks.* — 1989. — Vol. 2, no. 5. — Pp. 359–366. https://doi.org/10.1016/0893-6080(89)90020-8.

7. Learning both Weights and Connections for Efficient Neural Network / Song Han, Jeff Pool, John Tran, William J. Dally // Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada. — 2015. — Pp. 1135–1143. http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.

8. *Han Song, Mao Huizi, Dally William J.* Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding // *CoRR.* — 2015. — Vol. abs/1510.00149. http://arxiv.org/abs/1510.00149.

9. *Lu Zhiyun, Sindhwani Vikas, Sainath Tara N.* Learning compact recurrent neural networks // 2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016. — 2016. — Pp. 5960–5964. https://doi.org/10.1109/ICASSP.2016.7472821.

10. *Arjovsky Martín, Shah Amar, Bengio Yoshua.* Unitary Evolution Recurrent Neural Networks // Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016. — 2016. — Pp. 1120–1128. http://jmlr.org/proceedings/papers/v48/arjovsky16.html.

11. Tensorizing Neural Networks / Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, Dmitry P. Vetrov // Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada. — 2015. — Pp. 442–450. http://papers.nips.cc/paper/5787-tensorizing-neural-networks.

12. *Kingma Diederik P, Salimans Tim, Welling Max.* Variational Dropout and the Local Reparameterization Trick // Advances in Neural Information Processing Systems 28 / Ed. by C. Cortes, N. D. Lawrence, D. D. Lee et al. — Curran Associates, Inc., 2015. — Pp. 2575–2583. http://papers.nips.cc/paper/5666-variational-dropout-and-the-local-reparameterization-trick.pdf.

13. *Molchanov Dmitry, Ashukha Arsenii, Vetrov Dmitry P.* Variational Dropout Sparsifies Deep Neural Networks // Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. — 2017. — Pp. 2498–2507. http://proceedings.mlr.press/v70/molchanov17a.html.

14. Structured Bayesian Pruning via Log-Normal Multiplicative Noise / Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, Dmitry P. Vetrov // Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA. — 2017. — Pp. 6778–6787. http://papers.nips.cc/paper/7254-structured-bayesian-pruning-via-log-normal-multiplicative-noise.

15. *Grachev Artem M., Ignatov Dmitry I., Savchenko Andrey V.* Neural Networks Compression for Language Modeling // Pattern Recognition and Machine Intelligence - 7th International Conference, PReMI 2017, Kolkata, India, December 5-8, 2017, Proceedings. — 2017. — Pp. 351–357. https://doi.org/10.1007/978-3-319-69900-4_44.

16. Extraction of Visual Features for Recommendation of Products via Deep Learning / Elena Andreeva, Dmitry I. Ignatov, Artem M. Grachev, Andrey V. Savchenko // Analysis of Images, Social Networks and Texts - 7th International Conference, AIST 2018, Moscow, Russia, July 5-7, 2018, Revised Selected Papers. — 2018. — Pp. 201–210. https://doi.org/10.1007/978-3-030-11027-7_20.

17. *Grachev Artem M., Ignatov Dmitry I., Savchenko Andrey V.* Compression of recurrent neural networks for efficient language modeling // *Applied Soft Computing*. — 2019. — Vol. 79. — Pp. 354 – 362. http://www.sciencedirect.com/science/article/pii/S1568494619301851.

18. Efficient Language Modeling with Automatic Relevance Determination in Recurrent Neural Networks / Maxim Kodryan, Artem Grachev, Dmitry Ignatov, Dmitry Vetrov // ACL 2019, Proceedings of the 4th Workshop on Representation Learning for NLP, August 2, 2019, Florence, Italy. — 2019.

19. *Howard Jeremy, Ruder Sebastian.* Universal Language Model Fine-tuning for Text Classification // Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers. — 2018. — Pp. 328–339. https://aclanthology.info/papers/P18-1031/p18-1031.

20. Deep Contextualized Word Representations / Matthew E. Peters, Mark Neumann, Mohit Iyyer et al. // Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA,

June 1-6, 2018, Volume 1 (Long Papers). — 2018. — Pp. 2227–2237. https://aclanthology.info/papers/N18-1202/n18-1202.

21. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding / Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova // Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers). — 2019. — Pp. 4171–4186. https://aclweb.org/anthology/papers/N/N19/N19-1423/.

22. Neural GRANNy at SemEval-2019 Task 2: A combined approach for better modeling of semantic relationships in semantic frame induction / Nikolay Arefyev, Boris Sheludko, Adis Davletov et al. // Proceedings of the 13th International Workshop on Semantic Evaluation. — Minneapolis, Minnesota, USA: Association for Computational Linguistics, 2019. — jun. — Pp. 31–38. https://www.aclweb.org/anthology/S19-2004.

23. Large Language Models in Machine Translation / Thorsten Brants, Ashok C. Popat, Peng Xu et al. // EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic. — 2007. — Pp. 858–867. http://www.aclweb.org/anthology/D07-1090.

24. *Kneser Reinhard, Ney Hermann.* Improved backing-off for M-gram language modeling // 1995 International Conference on Acoustics, Speech, and Signal Processing, ICASSP '95, Detroit, Michigan, USA, May 08-12, 1995. — 1995. — Pp. 181–184. https://doi.org/10.1109/ICASSP.1995.479394.

25. *Jelinek Frederick.* Statistical Methods for Speech Recognition. — MIT Press, 1997. https://mitpress.mit.edu/books/statistical-methods-speech-recognition.

26. *Кудинов Михаил.* Статистическое моделирование русского языка с помощью нейронных сетей: Ph.D. thesis. — 2016. http://www.frccsc.ru/diss-council/00207305/diss/list/kudinov_ms.

27. *Marcus Mitchell P., Santorini Beatrice, Marcinkiewicz Mary Ann.* Building a Large Annotated Corpus of English: The Penn Treebank // *Computational Linguistics.* — 1993. — Vol. 19, no. 2. — Pp. 313–330.

28. A Neural Probabilistic Language Model / Yoshua Bengio, Réjean Ducharme, Pascal Vincent, Christian Janvin // *Journal of Machine Learning Research.* — 2003. — Vol. 3. — Pp. 1137–1155. http://www.jmlr.org/papers/v3/bengio03a.html.

29. *Mikolov Tomáš.* Statistical Language Models Based on Neural Networks: Ph.D. thesis / Brno University of Technology. — 2012. — P. 129. http://www.fit.vutbr.cz/research/view_pub.php?id=10158.

30. *Goodman Joshua T.* A bit of progress in language modeling // *Computer Speech & Language.* — 2001. — Vol. 15, no. 4. — Pp. 403–434. https://doi.org/10.1006/csla.2001.0174.

31. *Tieleman T., Hinton G.* Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. — COURSERA: Neural Networks for Machine Learning. — 2012.

32. *Duchi John C., Hazan Elad, Singer Yoram.* Adaptive Subgradient Methods for Online Learning and Stochastic Optimization // *Journal of Machine Learning Research.* — 2011. — Vol. 12. — Pp. 2121–2159. http://dl.acm.org/citation.cfm?id=2021068.

33. *Zeiler Matthew D.* ADADELTA: An Adaptive Learning Rate Method // *CoRR.* — 2012. — Vol. abs/1212.5701. http://arxiv.org/abs/1212.5701.

34. *Kingma Diederik P., Ba Jimmy.* Adam: A Method for Stochastic Optimization // 3rd International Conference on Learning Representations, ICLR 2015,

San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. — 2015. http://arxiv.org/abs/1412.6980.

35. *Werbos P.* Backpropagation through time: what it does and how to do it // *Proceedings of the IEEE.* — 1990. — Vol. 78(10). — Pp. 1550–1560.

36. *Rumelhart David E., Hinton Geoffrey E., Williams Ronald J.* Neurocomputing: Foundations of Research / Ed. by James A. Anderson, Edward Rosenfeld. — Cambridge, MA, USA: MIT Press, 1988. — Pp. 696–699. http://dl.acm.org/citation.cfm?id=65669.104451.

37. *Pascanu Razvan, Mikolov Tomas, Bengio Yoshua.* On the difficulty of training recurrent neural networks // Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013. — 2013. — Pp. 1310–1318. http://jmlr.org/proceedings/papers/v28/pascanu13.html.

38. *Галушкин А. И.* Синтез многослойных систем распознавания образов. — М.: Энергия, 1974.

39. *Werbos Paul.* Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences: Ph.D. thesis / Harvard University. — 1974.

40. *Bengio Yoshua, Simard Patrice Y., Frasconi Paolo.* Learning long-term dependencies with gradient descent is difficult // *IEEE Trans. Neural Networks.* — 1994. — Vol. 5, no. 2. — Pp. 157–166. https://doi.org/10.1109/72.279181.

41. *Hochreiter Sepp, Schmidhuber Jürgen.* Long Short-Term Memory // *Neural Computation.* — 1997. — Vol. 9, no. 8. — Pp. 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

42. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies / Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Juergen Schmidhuber // *S. C. Kremer and J. F. Kolen, eds. A Field Guide to Dynamical Recurrent Neural Networks.* — 2001. ftp://ftp.idsia.ch/pub/juergen/gradientflow.pdf.

43. *Deng Hongli, Zhang Lei, Shu Xin.* Feature memory-based deep recurrent neural network for language modeling // *Appl. Soft Comput.* — 2018. — Vol. 68. — Pp. 432–446. https://doi.org/10.1016/j.asoc.2018.03.040.

44. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches / Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, Yoshua Bengio // Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014. — 2014. — Pp. 103–111. http://aclweb.org/anthology/W/W14/W14-4012.pdf.

45. *Graves Alex.* Generating Sequences With Recurrent Neural Networks // *CoRR.* — 2013. — Vol. abs/1308.0850. http://arxiv.org/abs/1308.0850.

46. *Bahdanau Dzmitry, Cho Kyunghyun, Bengio Yoshua.* Neural Machine Translation by Jointly Learning to Align and Translate // 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. — 2015. http://arxiv.org/abs/1409.0473.

47. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling / Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, Yoshua Bengio // *CoRR.* — 2014. — Vol. abs/1412.3555. http://arxiv.org/abs/1412.3555.

48. *Bergstra James, Bengio Yoshua.* Random Search for Hyper-Parameter Optimization // *Journal of Machine Learning Research.* — 2012. — Vol. 13. — Pp. 281–305. http://dl.acm.org/citation.cfm?id=2188395.

49. Pointer Sentinel Mixture Models / Stephen Merity, Caiming Xiong, James Bradbury, Richard Socher // 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. — 2017. https://openreview.net/forum?id=Byj72udxe.

50. Recurrent neural network based language model / Tomas Mikolov, Martin Karafiát, Lukás Burget et al. // INTERSPEECH 2010, 11th Annual

Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010. — 2010. — Pp. 1045–1048. http://www.isca-speech.org/archive/interspeech_2010/i10_1045.html.

51. *Press Ofir, Wolf Lior.* Using the Output Embedding to Improve Language Models // Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers. — 2017. — Pp. 157–163. https://aclanthology.info/papers/E17-2025/e17-2025.

52. *Zaremba Wojciech, Sutskever Ilya, Vinyals Oriol.* Recurrent Neural Network Regularization // *CoRR*. — 2014. — Vol. abs/1409.2329. http://arxiv.org/abs/1409.2329.

53. Pointing the Unknown Words / Çaglar Gülçehre, Sungjin Ahn, Ramesh Nallapati et al. // Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers. — 2016. http://aclweb.org/anthology/P/P16/P16-1014.pdf.

54. *Bengio Yoshua, Senecal Jean-Sébastien.* Quick Training of Probabilistic Neural Nets by Importance Sampling // Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics, AISTATS 2003, Key West, Florida, USA, January 3-6, 2003. — 2003. http://research.microsoft.com/en-us/um/cambridge/events/aistats2003/proceedings/164.pdf.

55. *Morin Frederic, Bengio Yoshua.* Hierarchical Probabilistic Neural Network Language Model // Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, AISTATS 2005, Bridgetown, Barbados, January 6-8, 2005. — 2005. http://www.gatsby.ucl.ac.uk/aistats/fullpapers/208.pdf.

56. *Inan Hakan, Khosravi Khashayar, Socher Richard.* Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling // *CoRR*. — 2016. — Vol. abs/1611.01462. http://arxiv.org/abs/1611.01462.

57. *Lobacheva Ekaterina, Chirkova Nadezhda, Vetrov Dmitry.* Bayesian Sparsification of Recurrent Neural Networks // *arXiv preprint arXiv:1708.00077.* — 2017.

58. Deep Fried Convnets / Zichao Yang, Marcin Moczulski, Misha Denil et al. // 2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015. — 2015. — Pp. 1476–1483. https://doi.org/10.1109/ICCV.2015.173.

59. Ultimate tensorization: compressing convolutional and FC layers alike / Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, Dmitry P. Vetrov // *CoRR.* — 2016. — Vol. abs/1611.03214. http://arxiv.org/abs/1611.03214.

60. *Tjandra Andros, Sakti Sakriani, Nakamura Satoshi.* Compressing recurrent neural network with tensor train // 2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017. — 2017. — Pp. 4451–4458. https://doi.org/10.1109/IJCNN.2017.7966420.

61. Long-term Forecasting using Tensor-Train RNNs / Rose Yu, Stephan Zheng, Anima Anandkumar, Yisong Yue // *CoRR.* — 2017. — Vol. abs/1711.00073. http://arxiv.org/abs/1711.00073.

62. On the compression of recurrent neural networks with an application to LVCSR acoustic modeling for embedded speech recognition / Rohit Prabhavalkar, Ouais Alsharif, Antoine Bruguier, Ian McGraw // 2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016. — 2016. — Pp. 5970–5974. https://doi.org/10.1109/ICASSP.2016.7472823.

63. Online Embedding Compression for Text Classification using Low Rank Matrix Factorization / Anish Acharya, Rahul Goel, Angeliki Metallinou, Inderjit S. Dhillon // *CoRR.* — 2018. — Vol. abs/1811.00641. http://arxiv.org/abs/1811.00641.

64. *Chirkova Nadezhda, Lobacheva Ekaterina, Vetrov Dmitry P.* Bayesian Compression for Natural Language Processing // Proceedings of the 2018 Conference on

Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018. — 2018. — Pp. 2910–2915. https://aclanthology.info/papers/D18-1319/d18-1319.

65. *Rassadin A. G., Savchenko A. V.* Deep neural networks performance optimization in image recognition // *Proceedings of the 3rd International Conference on Information Technologies and Nanotechnologies (ITNT)*. — 2017.

66. In-Datacenter Performance Analysis of a Tensor Processing Unit / Norman P. Jouppi, Cliff Young, Nishant Patil et al. // Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017. — 2017. — Pp. 1–12. https://doi.org/10.1145/3079856.3080246.

67. Predicting Parameters in Deep Learning / Misha Denil, Babak Shakibi, Laurent Dinh et al. // Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States. — 2013. — Pp. 2148–2156. http://papers.nips.cc/paper/5025-predicting-parameters-in-deep-learning.

68. *Xue Jian, Li Jinyu, Gong Yifan.* Restructuring of deep neural network acoustic models with singular value decomposition // INTERSPEECH 2013, 14th Annual Conference of the International Speech Communication Association, Lyon, France, August 25-29, 2013. — 2013. — Pp. 2365–2369. http://www.isca-speech.org/archive/interspeech_2013/i13_2365.html.

69. *Oseledets Ivan V.* Tensor-Train Decomposition // *SIAM J. Scientific Computing*. — 2011. — Vol. 33, no. 5. — Pp. 2295–2317. http://dx.doi.org/10.1137/090752286.

70. *Oseledets Ivan V., Savostyanov Dmitry V., Tyrtyshnikov Eugene E.* Tucker Dimensionality Reduction of Three-Dimensional Arrays in Linear Time // *SIAM*

J. Matrix Analysis Applications. — 2008. — Vol. 30, no. 3. — Pp. 939–956. https://doi.org/10.1137/060655894.

71. *Mikolov Tomas, Zweig Geoffrey*. Context dependent recurrent neural network language model // 2012 IEEE Spoken Language Technology Workshop (SLT), Miami, FL, USA, December 2-5, 2012. — 2012. — Pp. 234–239. https://doi.org/10.1109/SLT.2012.6424228.

72. *Gal Yarin, Ghahramani Zoubin*. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks // Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain. — 2016. — Pp. 1019–1027. http://papers.nips.cc/paper/6241-a-theoretically-grounded-application-of-dropout-in-recurrent-neural-networks.

73. Character-Aware Neural Language Models / Yoon Kim, Yacine Jernite, David Sontag, Alexander M. Rush // Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA. — 2016. — Pp. 2741–2749. http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12489.

74. Recurrent Highway Networks / Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, Jürgen Schmidhuber // Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. — 2017. — Pp. 4189–4198. http://proceedings.mlr.press/v70/zilly17a.html.

75. *Merity Stephen, Keskar Nitish Shirish, Socher Richard*. Regularizing and Optimizing LSTM Language Models // *CoRR*. — 2017. — Vol. abs/1708.02182. http://arxiv.org/abs/1708.02182.

76. Breaking the Softmax Bottleneck: A High-Rank RNN Language Model / Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, William W. Cohen // *CoRR*. — 2017. — Vol. abs/1711.03953. http://arxiv.org/abs/1711.03953.

77. *Bai Shaojie, Kolter J. Zico, Koltun Vladlen.* Trellis Networks for Sequence Modeling // *CoRR.* — 2018. — Vol. abs/1810.06682. http://arxiv.org/abs/1810.06682.

78. Attention is All you Need / Ashish Vaswani, Noam Shazeer, Niki Parmar et al. // Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA. — 2017. — Pp. 6000–6010. http://papers.nips.cc/paper/7181-attention-is-all-you-need.

79. Intriguing properties of neural networks / Christian Szegedy, Wojciech Zaremba, Ilya Sutskever et al. // 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings. — 2014. http://arxiv.org/abs/1312.6199.

80. *Hron Jiri, de G. Matthews Alexander G., Ghahramani Zoubin.* Variational Bayesian dropout: pitfalls and fixes // *CoRR.* — 2018. — Vol. abs/1807.01969. http://arxiv.org/abs/1807.01969.

81. *Titsias Michalis K., Lázaro-Gredilla Miguel.* Doubly Stochastic Variational Bayes for non-Conjugate Inference // Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014. — 2014. — Pp. 1971–1979. http://jmlr.org/proceedings/papers/v32/titsias14.html.

82. *Carlin Bradley P., Louis Thomas A.* BAYES AND EMPIRICAL BAYES METHODS FOR DATA ANALYSIS // *Statistics and Computing.* — 1997. — Vol. 7, no. 2. — Pp. 153–154. https://doi.org/10.1023/A:1018577817064.

83. How to Train Deep Variational Autoencoders and Probabilistic Ladder Networks / Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe et al. // *CoRR.* — 2016. — Vol. abs/1602.02282. http://arxiv.org/abs/1602.02282.

84. *Trippe Brian, Turner Richard.* Overpruning in variational bayesian neural networks // *arXiv preprint arXiv:1801.06230.* — 2018.

Благодарности

Автор глубоко благодарен своему научному руководителю Д. И. Игнатову за постоянную поддержку в работе, обсуждение идей, полезные советы, а также за предоставленную свободу в процессе выбора темы и направления исследования.

Автор благодарит соавторов совместных статей: А. В. Савченко, М. С. Кодряна, Д. П. Ветрова, Е. Е. Андрееву за творческую поддержку и помощь в разработке идей.

Автор благодарит своих коллег в Samsung R&D Russia: А. Ю. Невидомского, И. И. Пионтковскую, А. С. Чернявского, Н. В. Арефьева, А. В. Подольского, Д. А. Липина, М. С. Кудинова, Д. В. Полуботко, А. Ю. Казанцева за полезные обсуждения, советы по работе и помощь с реализацией моделей для мобильного GPU.

Автор благодарит коллектив преподавателей и сотрудников НИУ ВШЭ: С. О. Кузнецова, С. А. Объедкова, Л. И. Антропову, Т. С. Никуличева, И. А. Кононенко, Е. А. Антонову за ценные советы и помощь в организационных вопросах.

Автор благодарит Е. М. Грекову за постоянное внимание к работе и неоценимую поддержку.

Список рисунков

Список таблиц