|  |
| --- |
| **Отчет об исследовании** |
| **Reformer: the efficient transformer** |
|  |
| **Пронченко Павел Леонидович** |
| **25.07.2020** |

**Reformer model. Актуальность исследования.**

Модель Transformer - это все более популярная архитектура нейронных сетей в области исследований обработки естественного языка (NLP), где большие трансформеры могут достигать самых современных результатов при выполнении многих задач. Компромисс заключается в чрезмерном потреблении и стоимости вычислений трансформеров, особенно для обучающих моделей на длинных последовательностях.

В недавней статье, опубликованной исследователями Google и UC Berkeley и принятой престижной International Conference on Learning Representations (ICLR 2020), предлагается новая трансформерная модель под названием Реформер (Reformer) [1], которая достигает впечатляющей производительности даже при работе только на одном графическом процессоре.

Архитектура Reformer предназначена для эффективной обработки очень длинных последовательностей данных (например, до 1 миллиона слов в языковой обработке). Выполнение Reformer требует значительно меньшего потребления памяти и достигает впечатляющей производительности. Ожидается, что модель Reformer окажет значительное влияние на область, выйдя за рамки языковых приложений (например, создание музыки, речи, изображений и видео).

Класс задач в NLP (например, машинный перевод, генерация текста, ответы на вопросы) может быть сформулирован как задача последовательного (sequence-to-sequence) обучения. Нейронные сети с длительной кратковременной памятью (LSTM), позже оснащенные механизмом внимания, были выдающейся архитектурой, используемой для построения моделей прогнозирования для таких задач — например, в системе машинного перевода Google. Однако изначально последовательный характер рекуррентности в LSTMs был самым большим препятствием в распараллеливании вычислений над последовательностью данных (с точки зрения скорости и затухающих градиентов), и в результате эти архитектуры не могли использовать преимущества контекста в длинных последовательностях.

Более поздняя модель Transformer - представленная в статье “Attention is all you need” [2] - достигла SOTA результатов в ряде задач, избавившись от реккурентности и вместо этого введя механизм multi-head self-attention. Главной идей Transformer была его возможность параллельных вычислений, которая позволяла обрабатывать длинные последовательности (с контекстными окнами из тысяч слов), приводя к более совершенным моделям, таким как замечательная открытая языковая модель GPT2 AI с меньшим временем обучения. Библиотека Transformers Huggingface [4] с более чем *32+* предварительно обученными моделями на *100+* языках и совместимостью между TensorFlow и PyTorch — это фантастическая работа с открытым исходным кодом для создания современных систем NLP. Transformer использовался и для приложений, выходящих за рамки текста, таких как генерация музыки и изображений.

**Что не хватает в Transformer?**

Прежде чем разобрать подробно модель Reformer, давайте рассмотрим, что является сложным в модели Transformer. Хотя модели Transformer дают отличные результаты при использовании на все более длинных последовательностях, например, тексты длины 11 тыс. токенов в (Liu et al., 2018) - многие из таких больших моделей могут быть обучены только на больших промышленных вычислительных платформах и даже не могут быть дообучены (finetune) на одном графическом процессоре даже для одного шага обучения из-за их требований к памяти. Например, полная модель GPT-2 состоит примерно из 1,5 млрд. параметров. Количество параметров в самой большой конфигурации, описанной в (Shazeer et al., 2018) [7] превышает 0,5 млрд. на слой, в то время как количество слоев доходит до 64 в (Al-Rfou et al., 2018). [7]

Изображение выглядит как снимок экрана

Автоматически созданное описание

*Рис. 1. Проблемы в архитектуре Transformer*

Вы можете заметить, что на диаграмме есть некоторые очки (👓) с 3-мя различными цветами. Каждые из этих уникальных очков (👓) представляет собой часть модели Transformer, которую авторы Reformer рассматривали как источник проблем с вычислениями и памятью:

*Проблема 1 (красные 👓 ): вычисление Multi-head self-attention (механизм внимания)*

Вычисление Attention на последовательностях длины *L* равно *O(L2)* (как время, так и память). Представьте себе, что произойдет, если вычислить Attention на последовательности длиной 64k (64 тыс. тоекнов).

*Проблема 2 (черные 👓 ): большое количество слоев*

Модель с N слоями потребляет в N раз больше памяти, чем однослойная модель, так как активации в каждом слое должны храниться для обратного распространения градиента.

*Проблема 3 (зеленые 👓 ): глубина полносвязных слоев*

Глубина промежуточных полносвязных слоев часто намного больше глубины активаций механизма внимания.

Модель Reformer обращается к вышеупомянутым трем основным источникам потребления памяти в Transformer и улучшает их таким образом, что модель Reformer может обрабатывать контекстные окна до 1 миллиона слов, все на одном ускорителе и используя только 16 ГБ памяти.

В двух словах, модель Reformer сочетает в себе два метода для решения проблем распределения внимания и памяти: локально-чувствительное хэширование (locality-sensitive-hashing - LSH), чтобы уменьшить сложность участия в длинных последовательностях, и обратимые остаточные слои (reversible residual layers), чтобы более эффективно использовать доступную память.

Ниже мы рассмотрим более подробно.

1. **Locality sensitive hashing (LSH) Attention – механизм внимания основанного на локально-чувствительном хешировании.**

*Внимание и ближайшие соседи.*

Внимание (Attention) в глубоком обучении - это механизм, который позволяет нейронной сети фокусироваться на различных частях контекста, основываясь на их соответствию текущему временному шагу. В модели Transformer существует 3 типа механизма внимания, как показано ниже:

Изображение выглядит как рисунок

Автоматически созданное описание

*Рис. 2. Три типа механизма внимания (Attention) в архитектуре Transformer*

Стандартный механизм внимания (Attention), используемый в Transformer – это нормализованное скалярное произведение, которое можно выразить такой формулой:

Изображение выглядит как нож

Автоматически созданное описание

Где Q, K, V соответственно Запрос (Q - Query), Ключ (K – Key), Значение (V – Value) механизма внимания. Q, K, V – это последовательности длинны L и размерности d (также d - это и глубина механизма внимания, величина на квадратный корень из которой происходит нормирование матричного произведения QKT).

Из приведенного выше уравнения и рисунка ниже видно, что вычислительные затраты и расход памяти на умножение QKᵀ (с размерностью [L, L]) - составляют сложность *O(L2)*, что является основным узким местом Transformer.

Изображение выглядит как внутренний, часы

Автоматически созданное описание

*Рис 3. (слева): основное вычисление в Dot-product attention, (справа) пример токена (‘it’), обращающего внимание на подмножество других токенов в последовательности (‘the’, ‘animal’, ‘street’, ‘it’,‘.’)*

Но нужно ли вычислять и хранить в памяти полную матрицу QKᵀ? Ответ - нет, так как нас интересует только softmax(QKᵀ), в котором, как известно, преобладают самые большие по значению элементы в типично разреженной матрице. Следовательно, как вы можете видеть в приведенном выше примере, для каждого запроса q нам нужно обратить внимание только на ключи k, которые находятся ближе всего к q. например, если K имеет длину 64k (64 тыс.), то для каждого q мы могли бы рассмотреть только небольшое подмножество 32 или 64 ближайших ключей. Таким образом, механизм внимания находит ключи ближайших соседей запроса, но неэффективно. Разве это не напоминает вам об поиске ближайших соседей?

Первая инновация в модели Reformer заключается в замене dot-product attention (внимания, основанного на матричном произведении) на локально-чувствительное хеширование(LSH), чтобы изменить сложность с *O(L2)* на O *(L log L).*

*LSH для поиска ближайших соседей*

LSH - это хорошо известный алгоритм для эффективного и приближенного способа поиска ближайших соседей в многомерных наборах данных. Основная идея LSH состоит в том, чтобы выбрать хэш-функции таким образом, чтобы для двух точек " p " и "q", если " q "близко к "p", то с достаточно хорошей вероятностью мы имеем hash(q) == hash(p).

Самый простой способ добиться этого - продолжать нарезать пространство случайными гиперплоскостями и добавлять sign(pᵀH) к хэш-коду каждой точки. Давайте рассмотрим пример ниже:

Изображение выглядит как текст, карта

Автоматически созданное описание

Рис 4. Упрощенная схема LSH для поиска ближайших соседей

Как только мы найдем хэш — коды нужной длины, мы разделим точки На «корзины» (buckets) на основе их хэш-кодов-в приведенном выше примере ‘a’ и ‘b’ принадлежат одной и той же «корзине», так как хэш(a) == хэш(b). Теперь пространство поиска для поиска ближайших соседей каждой точки резко сокращается из всего набора данных до «корзины», к которой она принадлежит.

Угловой LSH: вариант простого алгоритма LSH, называемый угловым LSH, проецирует точки на единичную сферу, которая была разделена на предопределенные области, каждая из которых имеет отдельный код. Затем ряд случайных вращений точек определяет «корзину» (bucket), к которому принадлежат точки. Давайте проиллюстрируем это на упрощенном 2D примере, взятом из статьи Reformer:

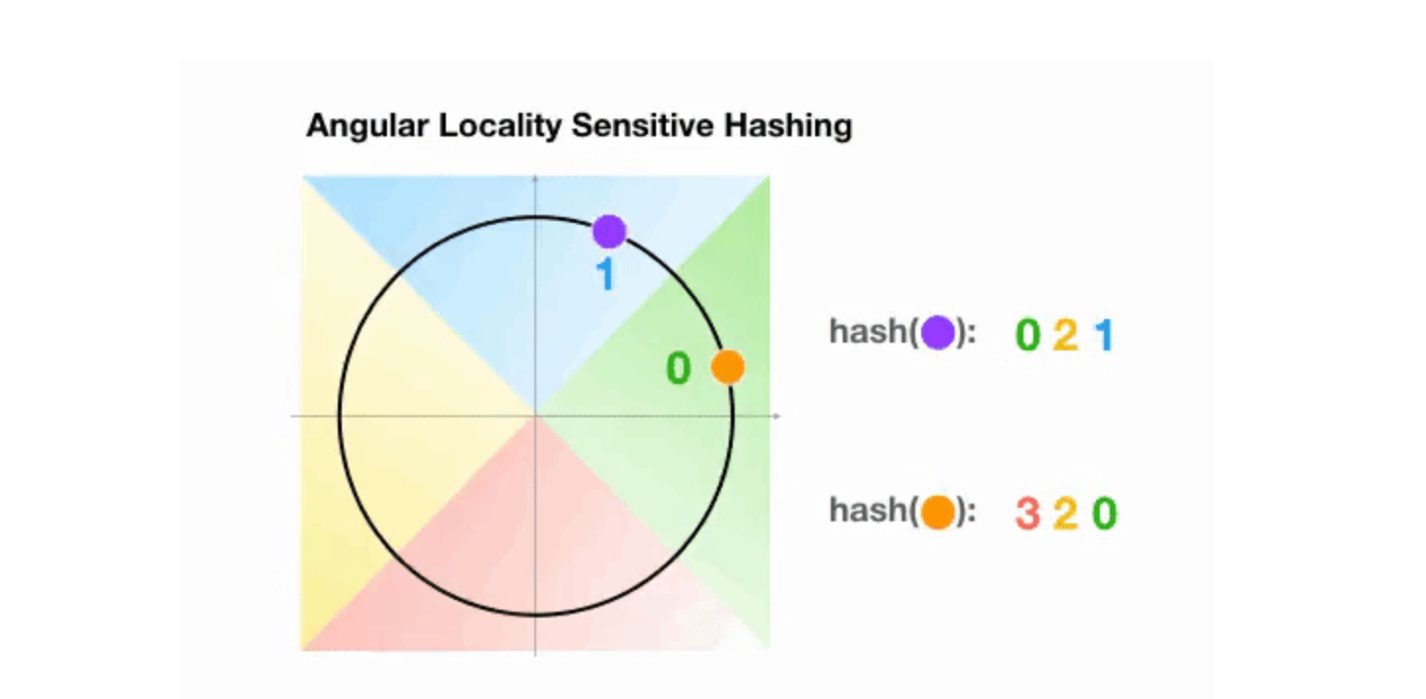


Рис. 5. Иллюстрация углового LSH

Здесь мы имеем две точки, которые проецируются на единичную окружность и произвольно поворачиваются 3 раза под разными углами. Мы можем заметить, что они вряд ли будут попадать в одну и ту же хэш-корзину. Однако в следующем примере мы видим, что две точки, которые находятся довольно близко друг к другу, в конечном итоге будут попадать в одну и ту же хэш-корзину после 3 случайных вращений:

Изображение выглядит как снимок экрана

Автоматически созданное описание

Рис. 6. Иллюстрация хэшей близких точек

*LSH Attention*

Основная идея, лежащая в основе внимания LSH, заключается в следующем. Возвращаясь к приведенной выше стандартной формуле внимания, вместо того чтобы вычислять внимание по всем векторам в матрицах Q и K, мы делаем следующее:

1. Найдем LSH-хэши матриц Q и K.
2. Вычислим стандартное внимание только для векторов k и q в пределах одних и тех же хэш-корзин (buckets).

Muti-round LSH attention: повторить описанную выше процедуру несколько раз, чтобы увеличить вероятность того, что похожие предметы не попадут в разные хэш-корзины (buckets).

Приведенные ниже рисунки иллюстрируют упрощенную версию алгоритма LSH Attention.

Шаг 1: Разбиение векторов на хэш-корзины

Изображение выглядит как рисунок

Автоматически созданное описание

Шаг 2: Сортировка по хэш-корзинам

Изображение выглядит как рисунок

Автоматически созданное описание

Шаг 3: Разбиваем отсортированные хэш-корзины на одинаковые по размеру части для лучшей паралелизации вычислений:

Изображение выглядит как рисунок

Автоматически созданное описание

Шаг 4-5-6-7 и тд: Вычисляем Attention внутри одной и той же части (собственная часть и предыдущая).

Изображение выглядит как рисунок

Автоматически созданное описание

*2. Reversible Transformer и chunking.*

Теперь мы готовы решить вторую и третью проблемы в Transformer, то есть большое количество (N) слоев энкодера и декодера и глубину слоев полносвязной сети.

*Reversible residual Network (RevNet)*

Обращая пристальное внимание на блоки кодера и декодера на рис. 2, мы понимаем, что каждый слой внимания и полносвязный слой обернуты в residual блок (подобный тому, что мы видим на рис. 7 (слева)). Residual Networks (ResNets), представленные в этой статье, являются мощным компонентом, используемым в NN— архитектурах для решения проблемы исчезающего градиента в глубоких сетях (со многими слоями). Однако потребление памяти в ResNets является узким местом, так как необходимо хранить активации в каждом слое в памяти, чтобы вычислить градиенты во время обратного распространения. Размер памяти пропорционален количеству элементов в нейронной сети.

Для решения этой проблемы используются reversible residual network (RevNet) (обратимые остаточные сети), которые состоят из ряда обратимых блоков. В Revnet активации каждого слоя могут быть восстановлены точно из последующих активаций слоя, что позволяет нам выполнять обратное распространение без сохранения активаций в памяти. Рис. 7. иллюстрирует ResNet блоки и RevNet блоки. Обратите внимание, как мы можем вычислить входы блока (X₁, X₂ ) из его выходов (Y₁, Y₂).

Изображение выглядит как текст

Автоматически созданное описание

Рис. 7. ResNet блок (слева) и RevNet блок (справа) в нейронных сетях.

*Reversible Transformer (Reformer)*

Возвращаясь к нашей второй проблеме, речь шла о требованиях к памяти N-слойной сети Transformer - с потенциально довольно большими N.

Reformer применяет идею RevNet к Transformer, объединяя слои внимания и полносвязный внутри блока RevNet. на рис. 7, теперь F становится слоем внимания, а G-полносвязным слоем:

*Y₁ = X₁ + Attention(X₂),*

*Y₂= X₂+ FeedForward(Y₁)*

Теперь использование reversible residual слоев вместо стандартных residual позволяет хранить активации только один раз в процессе обучения вместо N раз.

*Chunking (деление на части).*

Последняя часть повышения эффективности в Reformer связана с 3-й проблемой, то есть с промежуточными векторами большой размерности полносвязных слоев, которые могут достигать размерности 4k (4 тыс.) и выше.

Из-за того, что вычисления в полносвязных слоях независимы по позициям в последовательности, вычисления для прямого и обратного проходов, а также обратные вычисления могут быть разделены на части. Например, для прямого прохода у нас будет:



**Результаты**

Авторы провели эксперименты по двум задачам: задаче генерации изображений imagenet64 (с последовательностями длиной 12k) и текстовой задаче enwik8 (с последовательностями длиной 64k), а также оценили влияние Reformer и LSH на память, точность и скорость работы. [1]

Reformer соответствует бейзлайну Transformer: результаты показали, что Reformer экономит память без ущерба для точности:

Изображение выглядит как снимок экрана

Автоматически созданное описание

Рис. 8.  Сравнение с бейзлайном – стандартный Transformer и Reformer

Внимание LSH соответствует бейзлайну: поскольку внимание LSH является приближением полного внимания, его точность повышается по мере увеличения хэш-значения. Когда хэш-значение равно 8, внимание LSH почти эквивалентно полному вниманию:

Изображение выглядит как карта, текст

Автоматически созданное описание

Рис. 9. Сравнение полного механизма Attention и LSH-Attention с разным количеством хэш-корзин.

В статье продемонстрировали, что вычисление полного внимания замедляется по мере увеличения длины последовательности, в то время как скорость внимания LSH остается стабильной, и она работает на последовательностях длиной ~100K с обычной скоростью на графических процессорах с 8GB памяти:

Изображение выглядит как снимок экрана, текст, карта

Автоматически созданное описание

Рис. 10. Сравнение зависимость времени работы разных механизмов внимания при разной длине последовательности.

*Вывод: модель Reformer работала аналогично модели Transformer, но показала более высокую эффективность использования памяти и более высокую скорость вычислений на длинных последовательностях.*

**Эксперименты и результаты**

* Была обучена языковая модель Reformer на датасете enwiki8 на символьных токенах
* Размер последовательности – 4096
* Время обучения – около 10 часов на одном GPU c 12 Gb памяти

В результате на инференс была сгенерирована следующая последовательность:

Изображение выглядит как снимок экрана, держит, телефон

Автоматически созданное описание

Литература

[1] [Nikita Kitaev](https://arxiv.org/search/cs?searchtype=author&query=Kitaev%2C+N), [Łukasz Kaiser](https://arxiv.org/search/cs?searchtype=author&query=Kaiser%2C+%C5%81), [Anselm Levskaya](https://arxiv.org/search/cs?searchtype=author&query=Levskaya%2C+A).

“Reformer: The Efficient Transformer” <https://arxiv.org/abs/2001.04451>

[2] [Ashish Vaswani](https://arxiv.org/search/cs?searchtype=author&query=Vaswani%2C+A), [Noam Shazeer](https://arxiv.org/search/cs?searchtype=author&query=Shazeer%2C+N), [Niki Parmar](https://arxiv.org/search/cs?searchtype=author&query=Parmar%2C+N), [Jakob Uszkoreit](https://arxiv.org/search/cs?searchtype=author&query=Uszkoreit%2C+J), [Llion Jones](https://arxiv.org/search/cs?searchtype=author&query=Jones%2C+L), [Aidan N. Gomez](https://arxiv.org/search/cs?searchtype=author&query=Gomez%2C+A+N), [Lukasz Kaiser](https://arxiv.org/search/cs?searchtype=author&query=Kaiser%2C+L), [Illia Polosukhin](https://arxiv.org/search/cs?searchtype=author&query=Polosukhin%2C+I) “Attention Is All You Need” <https://arxiv.org/abs/1706.03762>

[3] <https://github.com/lucidrains/reformer-pytorch>

[4] <https://huggingface.co/transformers/model_doc/reformer.html>

[5] <https://towardsdatascience.com/illustrating-the-reformer-393575ac6ba0>

[6] <https://github.com/google/trax/tree/master/trax/models/reformer>

[7] <https://openai.com/blog/better-language-models/>

[8] <https://github.com/ppronchenko/study/blob/master/Diplom-Deep-Learning/ReformerLM_project_4096seq.ipynb>