Анализ современных подходов сжатия LLM и возможность локального развертывания

*Abstract*

В данной работе предлагается рассмотреть несколько различных “state-of-the-art” подходов сжатия больших языковых моделей и сравнить их между собой. В качестве критерия оценивания учитываются как теоретически значимые показатели, так и практические: проверка качества моделей после квантизации, размер конечной модели и скорость инференса. Для работы были выбраны 2 базовых модели – RedPajama-Chat-3B и LlaMA-2-7B. Модели с бо́льшим числом параметров не рассматривались, поскольку каждый эксперимент производился вручную на взятых в открытом доступе мощностях.

1. **Intro**

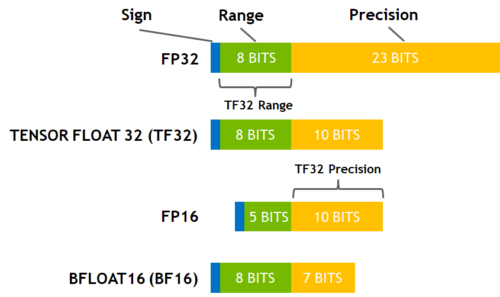
На сегодняшний день нейронные сети прогрессируют с каждым месяцем, поэтому возможности каждого человека сделать что-то оригинальное, нажав пару кнопок, увеличиваются. Однако, прогрессируют они не только в возможностях, но и в своем размере: по исследованиям Hugging Face и Carnegie Mellon University в недавней работе «Power Hungry Processing» указано, что генерация одной картинки сопоставима по потреблению энергии с целым зарядом смартфона. Однако, в данной работе речь пойдет о текстовых моделях, а точнее о современных подходах их сжатия. Более того, будет показано, что данные подходы позволяют развернуть собственные LLM без использования мощных вычислительных GPU.

1. **Practical part**

В качестве рассматриваемых моделей было рассмотрено множество кандидатов с различных ресурсов, и в конечном итоге были выбраны RedPajama-3B и LlaMA-2-7B. Далее будут рассматриваться методы сжатия на их примере.

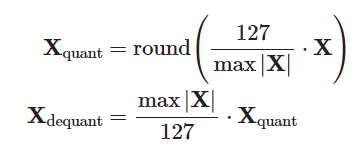
1. RedPajama-Chat-3B

Данная модель имеет, как видно из названия, 3 миллиарда параметров, и она запускалась с применением трех различных типов данных: float-16, brain-float16 и int8.

Для начала стоит обозначить различие между fp16 и bf16, поскольку обе единицы типов занимают по 16 бит: 

Как видно из рисунка, brain-float-16, в отличии от float-16, не сокращает свою экспоненциальную при переходе от 32-битного типа (как было 8 бит, так и осталось), поэтому такой тип данных может принимать значения в более широком диапазоне, однако с пониженной точностью вычислений из-за сокращенной мантиссы. Такой тип данных давно нашел себе применение в инференсе нейронных сетей на различных процессорах с современной архитектурой. Чтобы применить данные типы данных, не требуется больших усилий: нужно просто поменять внутренние операторы и ядра фреймворка для более быстрой работы с другим типом данных, и далее в нужных местах нейронной сети кастовать тип данных.

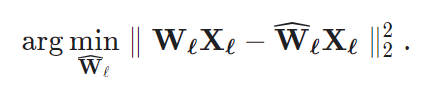
Далее, был рассмотрен более интересный случай: реальная квантизация из fp32 в int8. В данном пайплайне большую часть работы делает графовый компилятор, который заменяет точность вычислений во всех возможных слоях нейронной сети для ее ускорения. В данной нейронной сети используется метод Абсолютной Максимальной (AbsMax) квантизации:



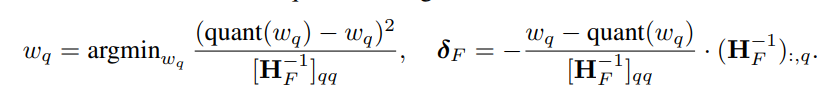
1. Llama-2-7b

После получения результатов первой модели, стало понятно, что можно взять что-то побольше и было принято решение исследовать Llama-2. Более того, здесь были разобраны более сложные способы квантизации.

**Первым** методом квантизации языковых моделей является GPT-Q. Данный метод является итеративным, в котором для каждого слоя минимизируется функция разности оригинальных и квантизированных весов.

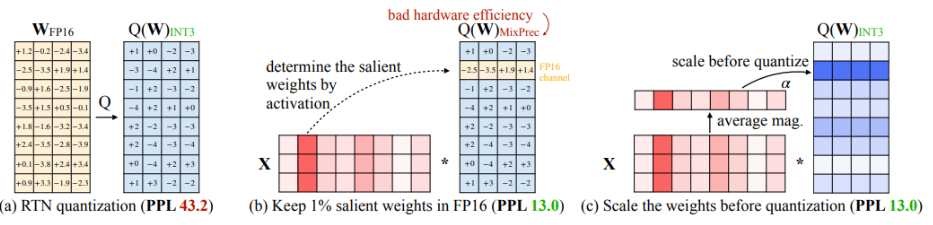


За основу метода берется Optimal Brain Quantization, в котором каждый слой квантизируется построчно. При квантизации весов, необработанная часть также изменяется, чтобы минимизировать общую ошибку.



GPTQ алгоритм решает ряд проблем, связанных с недостатками OBQ. Матрица Гессиана, используемая в формуле обновления весов, может быть вырожденной с ростом размера матрицы весов. Чтобы данная проблема не возникала, в методе применяется разложение Холецкого, которое позволяет найти обратную матрицу. Вторым недостатком является время обработки, так как поиск обратной матрицы на каждой итерации занимает много времени. Поэтому в GPTQ матрица Гессиана обновляется частично (относительно неквантизированных весов). В результате GPTQ является высокоточным и эффективным алгоритмом квантизации.

**Второй** рассмотренный метод квантизации модели к точности int4, это Activation-aware Weight Quantization, или сокращенно AWQ. AWQ использует идею, что не все ответы нейронной сети имеют одинаковый вес. Квантизация в AWQ происходит на этапе обучения модели. Она включает в себя два основных шага:

1. Взвешивание ответов: на каждом слое нейронной сети определяются веса ответов, которые представляют важность каждого ответа для итоговой оценки модели.
2. квантизация: после определения весов ответы квантуются, сохраняя только наиболее значимые биты. Количество сохраняемых битов определяется весами ответов. В результате AWQ -квантизации получается модель с меньшим количеством параметров.

Одним из главных преимуществ AWQ является его адаптивность. Поскольку веса ответов определяются на этапе обучения, квантизация может быть адаптирована к конкретной модели и данным, обеспечивая оптимальное сжатие и ускорение для каждого случая. Поэтому AWQ требует больше вычислительных ресурсов на этапе обучения по сравнению с другими методами квантизации.

1. **Results**
2. Dataset

Для экспериментов был выбран датасет WikiText. Набор данных для языкового моделирования WikiText представляет собой коллекцию из более чем 100 миллионов лексем, извлеченных из набора проверенных статей Good и Featured в Википедии. Поскольку вычисления сжатых моделей производились в том числе на CPU, то была выбрана тестовая выборка (4358 текстов) для валидации поднятых локально моделей.

1. Metric

Изначально планировалось рассмотреть оценивать качество моделей с помощью perplexity, однако затем выяснилось, что данная метрика работает хорошо лишь при обучении модели, т.к. она отражает, насколько хорошо она знает датасет, или другими словами, насколько ожидаемый контекст в рамках фиксированного датасета выдает модель. Разумеется, такая оценка не подошла, поэтому для валидации моделей была выбрана метрика BLEU. BLEU оценивает качество перевода или генерации текста на основе нескольких параметров:

- точное совпадение: количество слов в генерированном тексте, которые точно совпадают с эталонным текстом.

- частично совпадающие п-граммы: количество n-грамм (последовательностей из n слов) в генерированном и эталонном тексте, которые совпадают, но не обязательно в точной позиции.

- веса для п-грамм: BLEU придает больший вес более длинным n-граммам, так как они являются более точными индикаторами качества перевода.

- брекетинг и сглаживание: BLEU использует метод брекетинга и сглаживания для учета частичных совпадений и исключения перебора, когда одно и то же совпадение учитывается несколько раз.

Оценка BLEU вычисляется как среднее геометрическое значение точности и частично совпадающих n-грамм, умноженное на коэффициент веса. Чем выше значение BLEU, тем лучше качество перевода или генерации текста.

1. Hardware specifics

Поскольку одной из целей данного проекта была развертывание локальных языковых моделей без использования дополнительных ресурсов, то для сравнения скорости работы нейронных сетей использовались бесплатные мощности Google Colab. В базовой версии предоставляется 2 ядра Intel Xeon, 16 Gb RAM и Tesla T4 GPU 15Gb.

**Note!** Стоит учитывать, что некоторые вычисления таких типов данных оказались неподдерживаемыми на CPU или GPU, поэтому не оставалось выбора, кроме как запускать на том, где поддерживалось.

Для всех моделей был установлен размер вывода (длина последовательности) 64 токена.

По итогам всех исследований, мы поместили полученные данные в общую таблицу для более удобного сравнения. Критерии оценивания: размер модели, скорость инференса и качество сгенерированного текста (BLEU-метрика).

|  |  |  |  |
| --- | --- | --- | --- |
| **Model type** | **Model size, Gb** | **Inference time, sec** | **BLEU-metric** |
| RedPajama3B fp16 | 5.7 | 8.8 sec on GPU | 0.48 |
| RedPajama3B bf16 | 7 | 76 sec. on CPU | 0.48 |
| RedPajama3B int8 | 4.3 | 28 sec. on CPU | 0.48 |
| Llama-2-7B, int4 GPT-Q | 5.4 | 15 sec on GPU | 0.57 |
| Llama-2-7B, int4 AWQ | 12.5 | 11 sec. on GPU | 0.57 |

Из данной таблицы видны несколько достаточно интересных результатов, которые являются следствиями подходов, описанных ранее. Следует выделить, что AWQ квантизация показывает значительный прирост в производительности, однако задействует сильно больше памяти, что может быть важнее в некоторых системах. Без теоретической базы результаты могли бы казаться неочевидными.

Так же видно, что int8 маленькой RedPajama достаточно быстрый и имеет схожий результат, что и модели с 16-битной точностью, что сильно выделяет, при этом затраты по памяти так же значительно снижаются в 1.6 раз.

1. **Total**

В данной работе были рассмотрены несколько современных типов квантизаций больших языковых моделей и запущены для сравнительного анализа. Главным результатом, по авторскому мнению, является возможность запустить локально большую языковую модель, которая будет выдавать достаточно связный, грамматически и теоретически правильный текст, не задействовав при этом графических процессоров (часть результатов получена на CPU). Качество метрик при этом не ухудшается по сравнению с исходными моделями, запускаемыми на GPU, а значит, такое направление как «сокращение затрат на LLM» должно развиваться для возможности запуска еще более объемных и сложных моделей на менее мощных девайсах.

1. **References**
2. Power Hungry Processing, <https://arxiv.org/pdf/2311.16863.pdf>
3. Wikitext Dataset <https://huggingface.co/datasets/wikitext>
4. intro in quantization: <https://mlabonne.github.io/blog/posts/Introduction_to_Weight_Quantization.html>
5. LLM.int8: <https://arxiv.org/pdf/2208.07339.pdf>
6. GPT-Q: <https://arxiv.org/pdf/2210.17323.pdf>
7. Activation-aware Weight Quantization: <https://arxiv.org/pdf/2306.0097>
8. Quantization docs <https://huggingface.co/docs/transformers/main_classes/quantization>