Как учить большие языковые модели

FPMI ML MEETUP

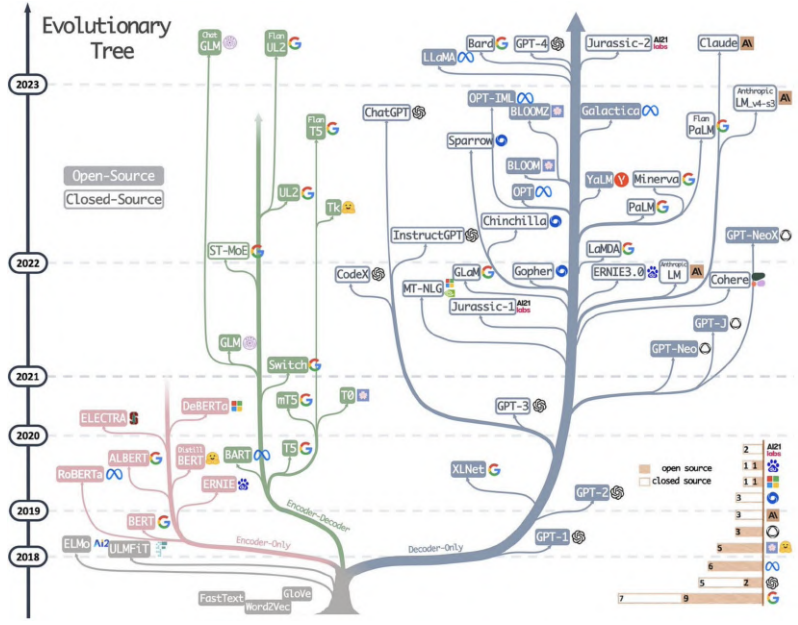
30 января, 2024

Мурат Апишев
Lead Data Scientist, SberDevices
mel-lain@yandex.ru

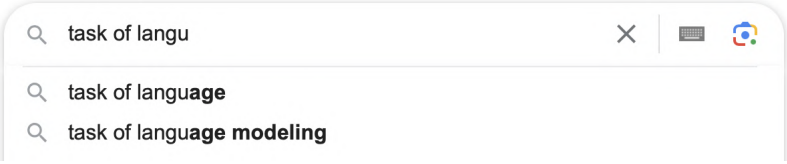
Сейчас «LLM» ≈ «AI»

Современные AI-сервисы:

- **общаются** с людьми на любые темы на естественном языке
- **решают** школьные и университетские задачи по разным дисциплинам
- **понимают** и **генерируют** тексты, изображения, аудио



Языковое моделирование



- У задачи моделирования языка есть две эквивалентные постановки:
  - предсказать совместную вероятность произвольной последовательности слов из n
  - предсказать вероятность следующего слова после произвольной последовательности из $n-1$ слов
- LLM учатся на корпусах текстов задаче предсказания следующего слова по предшествующему контексту
- Иногда контекст оказывается не только из прошлого, а из будущего (Fill-in-the-Middle), например, для задач code infilling

Промптинг LLM

- Современные большие языковые модели общего назначения:
  - мультиязычные
  - мультидоменные
  - инструктивные
- Благодаря этому они могут решать разные задачи, получая их на естественном языке (prompt), как исполнитель-человек
- Поставить задачу можно разными способами, стандартные подходы:
  - **Zero-shot**:

    «Вопрос: у Васи 4 яблока, у Пети 8 груш, сколько у них съел половину своих фруктов, сколько всего фруктов осталось? Ответ: »
  - **Few-shot**:

    «У Лены было 3 конфеты в одной руке, и 5 в другой, 2 она отдала Маше. Сколько конфет всего осталось у Лены? Ответ: 6
    Вопрос: у Васи 4 яблока, у Пети 8 груш, сколько у них съел половину своих фруктов, сколько всего фруктов осталось? Ответ: »

Промптинг LLM

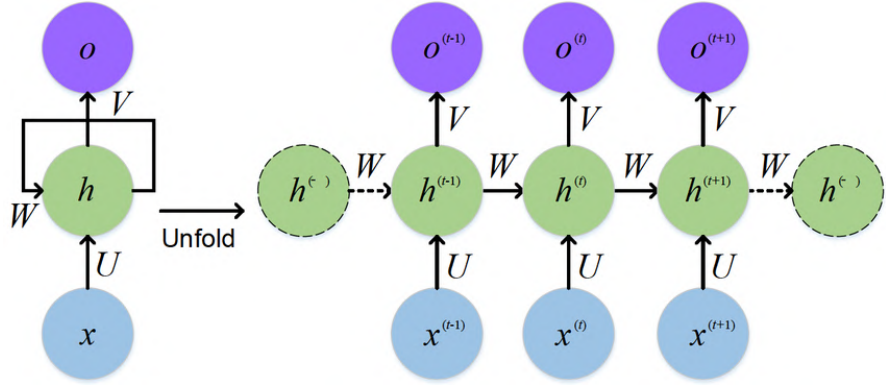
- Поставить задачу можно разными способами, стандартные подходы:
  - **Chain-of-Thought** (может быть и с zero-shot, и с few-shot):

    «У Лены было 3 конфеты в одной руке, и 5 в другой, 2 она отдала Маше. Сколько конфет всего осталось у Лены? Давай порассуждаем: всего у Маши $3 + 5 = 8$ конфет. 8 - 2 равно 6. Ответ: 6
    Вопрос: у Васи 4 яблока, у Пети 8 груш, сколько у них съел половину своих фруктов, сколько всего фруктов осталось? Давай порассуждаем: »

- В этом случае модель сперва сгенерирует промежуточные рассуждения, а затем, на их основании, ответ
- Модель будет хорошо решать задачи с разными типами промптов, если она этому обучалась
- Т.е. чтобы модель умела рассуждать последовательно, она должна видеть в данных много примеров таких рассуждений

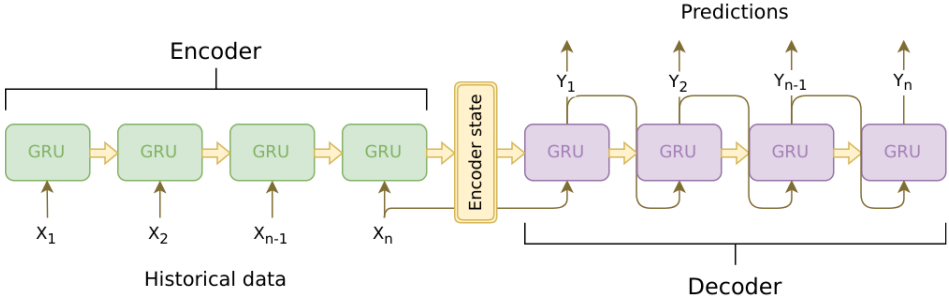
Архитектура

- До LLM: рекурентные нейросети (RNN), а именно LSTM, GRU, MGU
- Обрабатывают слово за словом, передавая обновляемый вектор состояния (и иногда дополнительный вектор) с информацией о последовательности
- Модель состоит из нескольких обучаемых весовых матриц



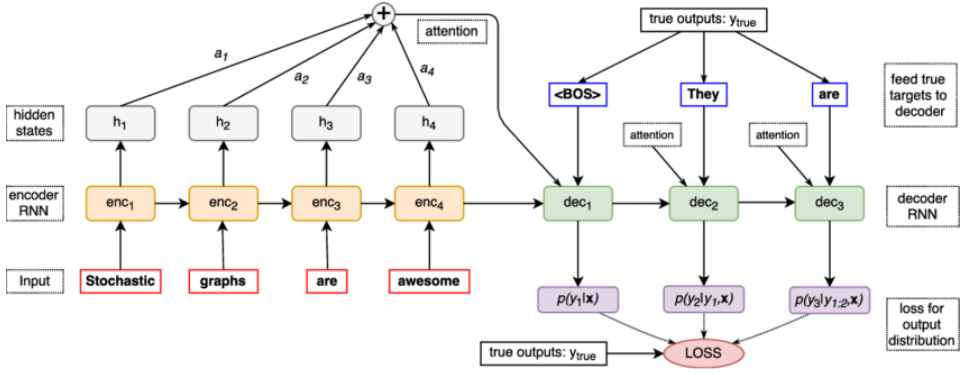
Архитектура

- RNN можно использовать для задач sequence-to-sequence (например, перевод или суммаризация)
- Нейросеть-кодировщик генерирует вектор состояния по входу
- Нейросеть-декодировщик генерирует по нему выход
- Работает не очень хорошо — на длинных последовательностях вектор теряет инфомрацию о словах в начале



Архитектура

- Для борьбы с забыванием добавляется механизм внимания
- Декодировщик при генерации очередного слова определяет важность каждого из слов входа и использует взвешенную сумму выходных векторов кодировщика как дополнительную информацию
- Качество сильно выросло, это был стандарт в области

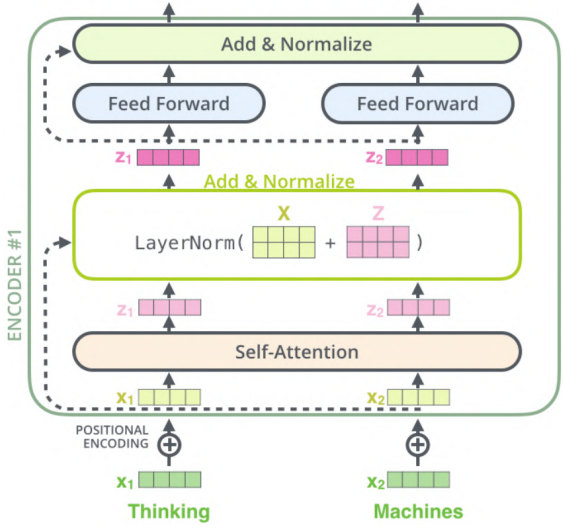


Архитектура

- Можно формировать векторы слов с учётом того, какие слова ещё есть в последовательности — получим Self-Attention
- Общая схема:
  - каждому слову входа сопоставляется вектор (на входе — эмбеддинг)
  - с помощью трёх обучаемых весовых матриц он переводится в три новых вектора: запросы, ключи и значения
  - новым представлением для слова будет взвешенная сумма всех векторов значений, нужно только определить, какие прочие слова для целевого слова важны
  - для этого запрос слова умножается скалярно на все ключи
  - после нормирования Softmax эти значения становятся весами суммы
- Можно считать несколько Self-Attention с разными весами конкатенировать результаты — Multi-Head Attention
- Нужно только добавить линейный слой проекции в исходную размерность

Архитектура

- Добавляются два полносвязных слоя, Layer-нормализация и residual connection и получается блок Transformer:

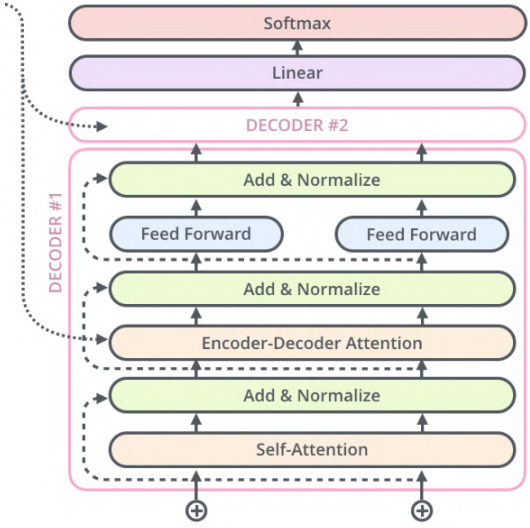


Архитектура

- Если поставить друг на друга несколько таких блоков, получится кодировщик Transformer
- Векторы слов многократно проходят через self-attention, на выходе получаются качественные контекстнозависимые представления
- Их уже можно использовать для решения разных задач напрямую или после дообучения небольшой модели-голов
- На основе архитектуры кодировщика обучены модели типа BERT
- Но оригинальный Transformer — sequence-to-sequence
- Нужен ещё и декодировщик, который на основе выходов кодировщика будет генерировать выходной текст

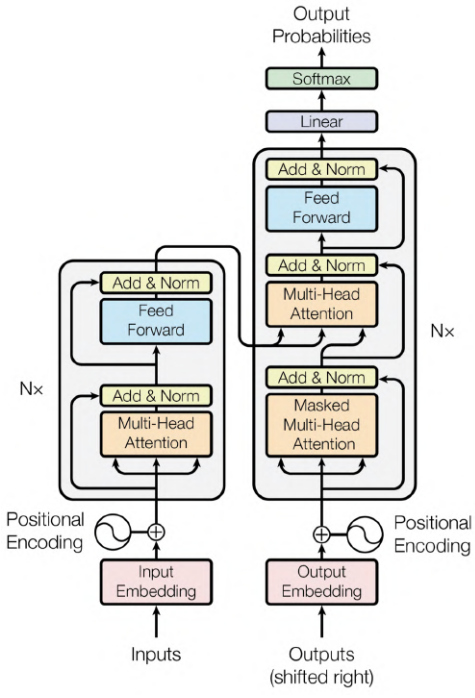
Архитектура

- Блок декодировщика похож на блок кодировщика, но есть отличия:
  - Masked Self-Attention для авторегрессионности
  - Cross-Attention между векторами сгенерированных и входных слов
  - запросы получаются из сгенерированных, ключи и значения — из входных
  - на выходе всего декодироващика softmax для генерации (как и в RNN)



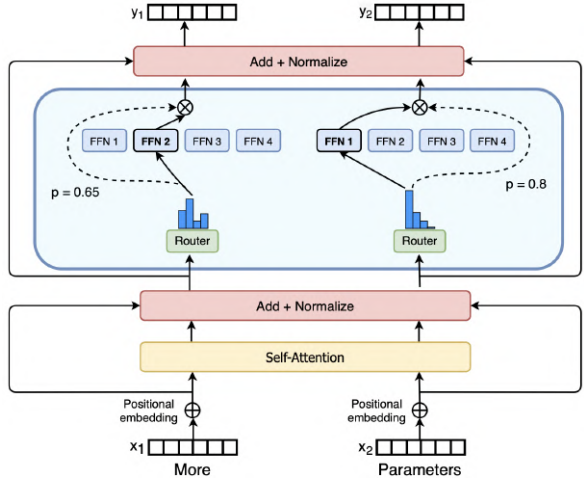
Архитектура

- Кодировщик с декодировщиком образуют полный Transformer (модели типа T5)
- Декодировщик часто используют отдельно (модели типа GPT)
- Чаще всего LLM обучаются на основе декодировщика Transformer (GPT, LLaMA, Qwen, Mistral, ...)
- За последние годы предложен ряд успешных архитектурных модификаций (pre-LayerNorm, RMSNorm), но суть сохранилась



Архитектура

- Если учить оптимально, то больше весов ⇒ выше качество
- Но больше весов ⇒ медленнее инференс
- Решение — Mixture-of-Experts, «эксперты» FF-слои
- Вектор слова после self-attention идёт Router, тот отправляет в его к лучшим экспертам
- Параметров много, но при обработке активируется только малая часть
- Каждый эксперт учится решать свои типы задач



Токенизация

- На самом деле Transformer не работает со словами:
  - требуется очень большой словарь
  - сложность учёта морфологии
  - проблема OOV слов
- Модели на символах тоже непопулярны — слишком длинная последовательность и низкое качество
- Вместо этого используются токены — подслова, т.е. символьные N-граммы (среди которых могут быть и частотные слова целиком)
- Разбиение на токены производит токенизатор, он обучается статистически по текстам выбранным алгоритмом:
  - BPE
  - Unigram
  - Wordpiece

Токенизация

- Токенизатор разбивает текст на токены и сопоставляет каждому его номер

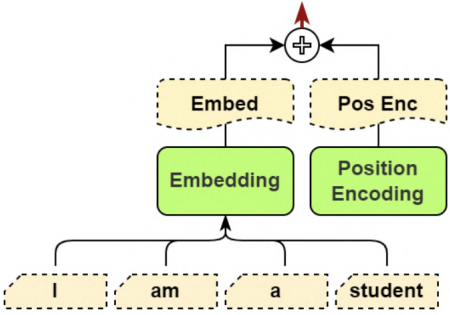
```
Two annoying things about OpenAI's tokenizer playground: (1) it's
 capped at 50k characters, and (2) it doesn't support GPT-4 or GPT-3.5
...

So, I built my own version w/ Transformers.js! It can tokenize the
 entire "Great Gatsby" (269k chars) in 200ms! ���
```

- Для обработки символов, не встречавшихся в обучающих данных токенизатора, используется Byte Fallback — в словарь добавляются сразу все возможных 256 байтов
- Есть эксперименты по использованию чисто byte-level токенизаторов
- Ещё пробуют строить словарь на основе морфологии
- Рецепта создания идеального токенизатора пока нет

Позиционное кодирование

- Без дополнительной информации о позициях токенов любая модель на основе Transformer работает плохо
- В оригинальной реализации для позиционного кодирования
  - каждой позиции i токена сопоставляется вектор, содержащий различные значения синусов и косинусов от i
  - этот вектор добавляется к вектору эмбеддинга токена на позиции i перед отправкой в модель
- Способ простой и рабочий, но есть проблемы:
  - низкое качество кодирования $\Rightarrow$ хуже результаты
  - низкое качество экстраполяции $\Rightarrow$ нет обобщения на больший контекст

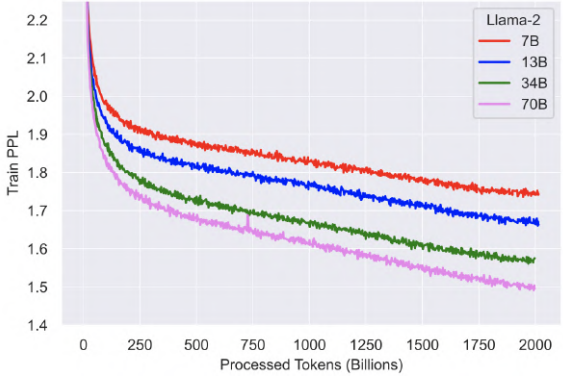


Позиционное кодирование

- Все более новые подходы — относительные, вместо позиции токена кодируется расстояние между парой токенов
- Вместо входного эмбеддинга модифицируется подсчёт self-attention, условно модификации можно разделить на три вида:
  - репараметризация формулы подсчёта внимания
    - Transformer-XL, 2019
    - DeBERTa, 2021
  - обычная формула с добавлением обучаемого сдвига
    - T5, 2020
    - AliBi, 2022
  - ротационное кодирование (RoPE, 2021) и его вариации
    - xPos, 2023
    - Positional Interpolation RoPE, 2023
    - YaRN, 2023
- Пробовали и вообще обходиться без кодирования позиций в декодерах (NoPE, 2023), но работает не очень

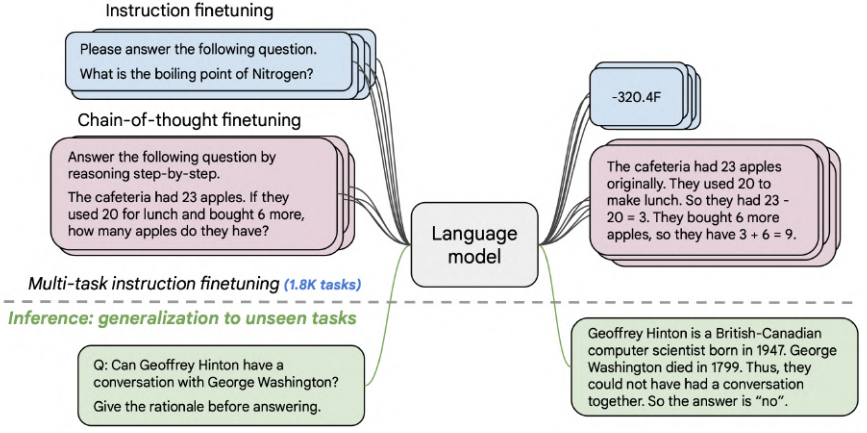
Этапы обучения

- Первая стадия обучения LLM — предобучение (pre-train)
- Модель учится предсказывать следующий токен по контексту слева
- Если учить с Teacher Forcing — контекст берётся из обучения, если без — из того, что сгенерировала в процессе сама модель (комбинируют)
- На этом этапе приобретает основные знания о языке и мире



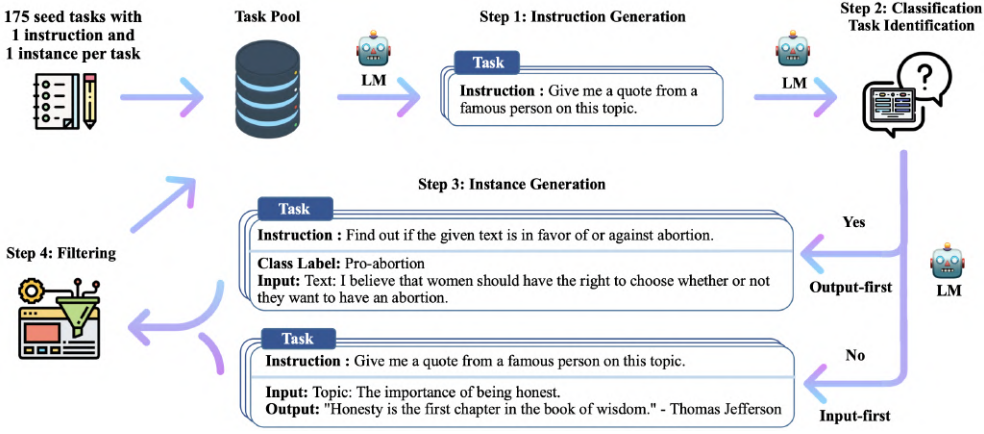
Этапы обучения

- Вторая стадия — Instruction Tuning (SFT)
- Модель учится понимать и исполнять запросы людей на естественном языке и вести диалоги
- Например: pre-train — LLaMA, instruct-tuned — Alpaca или Vicuna



Этапы обучения

- По сути модель так же обучается предсказывать следующий токен по прошлым, но на данных специального вида
- Объем данных сильно меньше, но требования к качеству высокие
- Сбор инструктивных датасетов сложен и дорог, пробуют Self-Instruct:

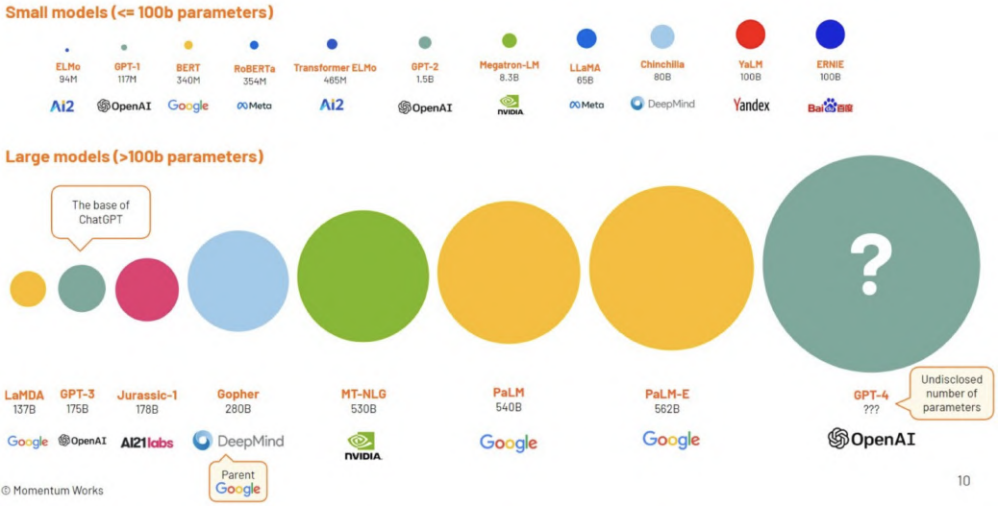


Этапы обучения

- Третий, опциональный, шаг — выравнивание (alignment)
- Диалоговая модель дообучается для генерации более корректных, полезных и безопасных ответов
- Популярная техника, использованная в Instruct GPT — RLHF:
  - обученная LLM генерирует на тестовом наборе инструкций ответы
  - ответы размечаются асессорами, на их ответах учится сильная reward-модель, она оценивает по тексту его качество
  - заводится две копии модели (A) и (B), учится (A)
  - обе модели генерируют ответы на каждый промпт, ответ (A) оценивается reward-моделью
  - веса (A) обновляются так, чтобы максимизировать reward и не давать ответы, очень далёкие от исходной (B)
  - расстояние определяется по KL-дивергенции между выходными распределениями моделей
  - обновление весов идёт по заданному алгоритме (PPO или A2C)

Баланс между параметрами и данными

- В общем случае модель чем больше, тем лучше, но только если обучена на достаточном объёме данных достаточное время

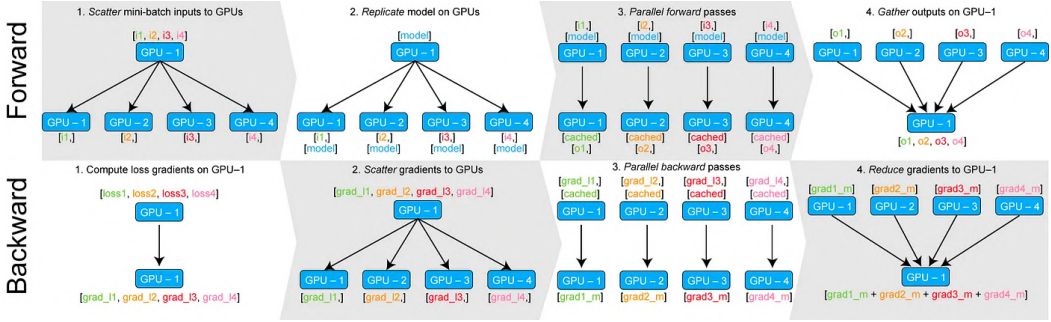


Масштабирование обучения

- При обучении память GPU в основном тратится на
  - веса модели
  - состояние оптимизатора
  - активации
  - градиенты
- Объем памяти GPU сильно ограничен: A100 имеет 80Гб
- У современных моделей даже веса могут не влезать в такой объём
- При обучении с помощью стандартного AdamW состояние оптимизатора требует x2 от размера модели
- Для обучения больших моделей с адекватной скоростью требуется обрабатывать большой объём данных одновременно
- Это возможно только при использовании параллельных вычислений на множестве GPU, этот процесс можно организовать по-разному

Масштабирование обучения

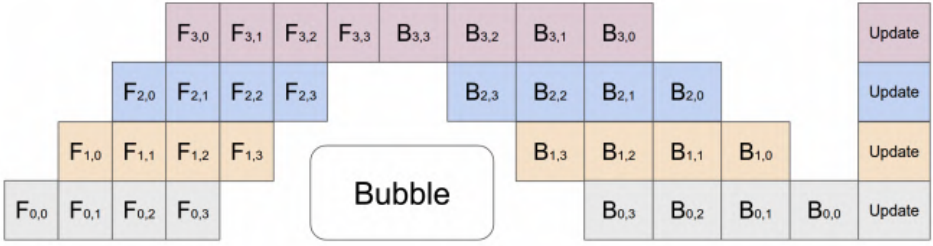
- Если модель и состояние оптимизатора не занимают всю память GPU, то очевидный способ параллелизма — по данным (Data Parallelism)
- Каждая GPU имеет свою копию модели и обрабатывает часть батча



- Можно добавить Gradient Accumulation: разделять батч по всем GPU не целиком, а частями для экономии памяти
- Градиенты агрегируются до обработки всего батча, после чего запускается обновление параметров модели

Масштабирование обучения

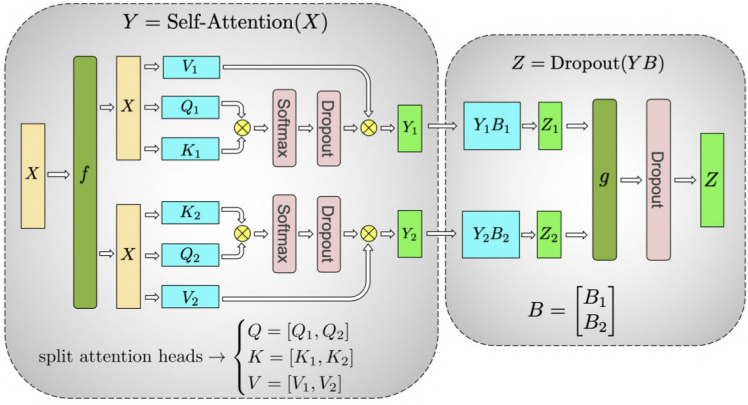
- Если памяти одной GPU не хватает, модель можно разрезать и разложить на несколько устройств
- Pipeline Parallelism — группы слоёв раскладываются по своим GPU
- Для уменьшения простоя батч нарезается на части, и более глубокие слои начинают работать раньше



- По сравнению с другими подходами требует сильно большего переписывания кода

Масштабирование обучения

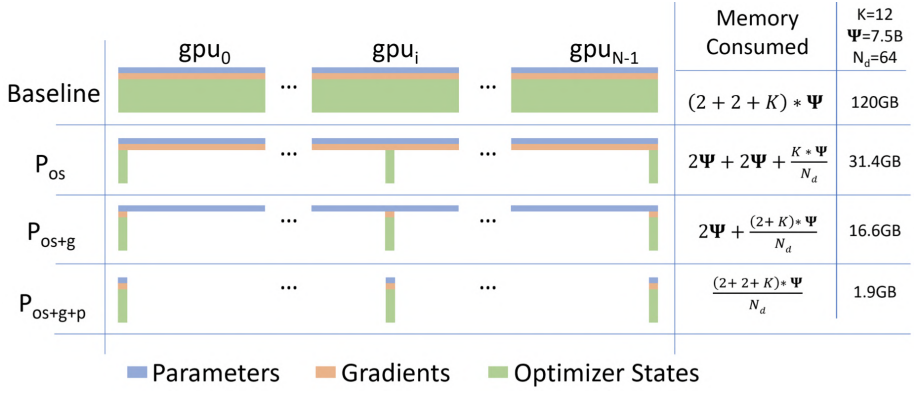
- Ещё вариант — Tensor Parallelism: по GPU раскладываются части тензора



- Self-attention параллелится естественно за счёт разных голов
- При TP сетевые коммуникации более интенсивные, чем при DP или PP
  $\Rightarrow$ модель лучше раскладывать на одном узле DGX или в сети InfiniBand

Масштабирование обучения

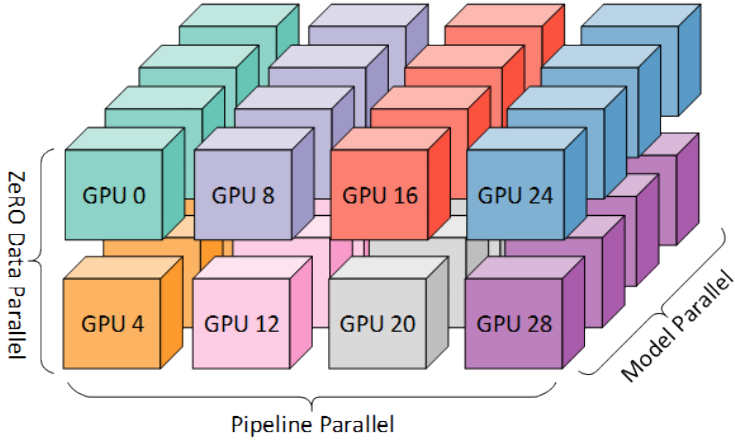
- В Data Parallelism можно избежать хранения избыточной информации с помощью ZeRO, 2019, стандартная реализация — DeepSpeed
- На каждом этапе (stage) потребление падает, сетевые коммуникации растут x1.5 только на этапе 3



- ZeRO умеет выгружать данные в RAM, что тоже экономит память GPU

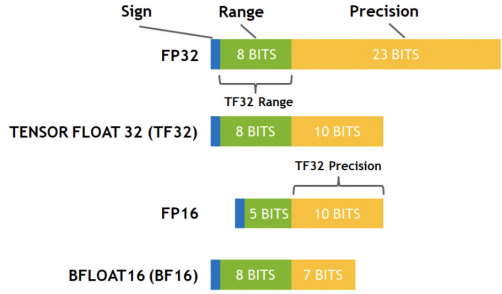
Масштабирование обучения

- Все техники могут применять как сами по себе, так и в комбинациях
- DP+PP+TP дают 3D-параллелизм, часто комбинируется с ZeRO stage 1 (stage 2/3 тоже можно, но сложнее + растут сетевые коммуникации)



Техники оптимизации

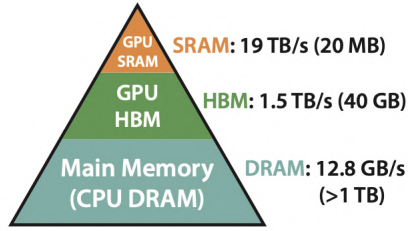
- Обучать модели в fp32 неэффективно
- Используют Mixed Precision:
  - две копии весов, в fp32 и fp16 / bf16 (если GPU Ampere)
  - активации считаются в fp16 / bf16
  - агрегации и нормализации в fp32
  - градиенты и состояние оптимизатора в fp32



- Затраты памяти нивелируются большим батчем, а обучение ускоряется
- fp16 требует масштабирования loss для стабильности (умножение на коэффициент и обратно), bf16 — нет, он в целом более стабилен
- GPU Ampere могут заменять fp32 на tf32 — более эффективный и экономичный формат, можно комбинировать это с Mixed Precision

Техники оптимизации

- Утилизация GPU была очень неоптимальной — много времени уходило не на вычисления, а на пересылку данных между HBM и SRAM
- Подсчёт softmax в self-attention генерирует промежуточные матрицы, которые занимают место и перемещаются

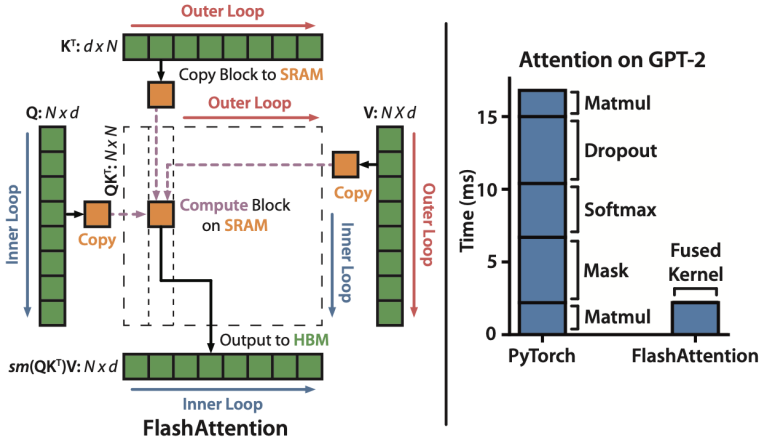


Memory Hierarchy with Bandwidth & Memory Size

- Можно ввести дополнительные переменные и считать softmax блочно
- Не нужно хранить промежуточные матрицы, передача данных между HBM и SRAM становится экономичнее
- Нужные для backward-шага промежуточные значения можно эффективно пересчитывать вместо хранения на forward

Техники оптимизации

- Дополнительное ускорение получено за счёт Fusing — выполнения набора операций одним CUDA ядром



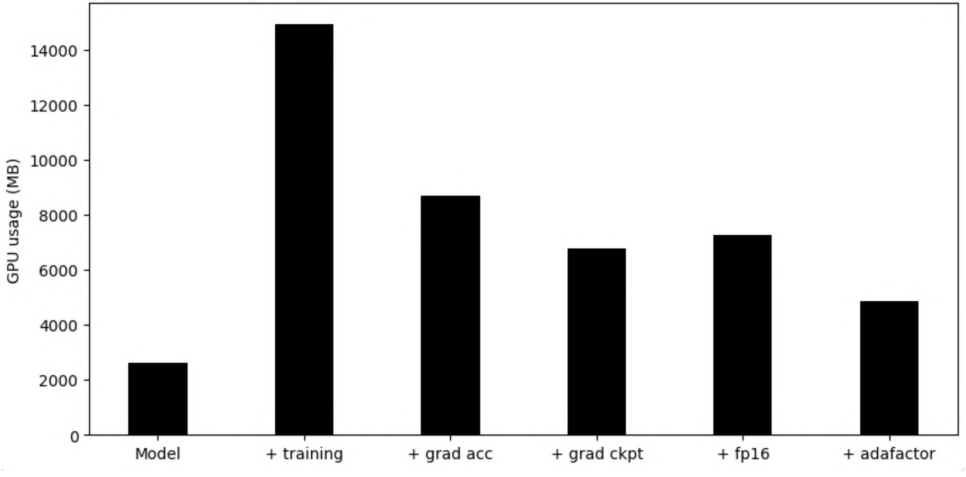
- Flash Attention 2, 2023 даёт ещё больший выигрыш по скорости за счёт вычислительных оптимизаций на GPU

Техники оптимизации

- Важная техника для оптимизации памяти на полносвязных слоях — Gradient (Activation) Checkpointing
- Выходы каждого линейного слоя на forward-шаге нужны для вычисления градиентов на этом слое на обратном шаге, поэтому они сохраняются
- Это приводит к линейному по числу слоёв росту потребления памяти
- Можно ничего не хранить и вычислять для каждого слоя активации с нуля (т.е. от начала сети до этого слоя)
- Это экономит память, но объем вычислений на forward из линейного по числу слоёв становится квадратичным
- **Решение:** сохранять активации части слоёв на некотором расстоянии друг от друга (checkpoint)
- Вычисление активаций слоя будет идти от последнего чекпойнта
- В среднем потребление памяти падает с $O(n)$ до $O(\log n)$ за счёт замедления примерно на 20%

Техники оптимизации

- Можно экспериментировать с оптимизаторами, например, Adafactor более экономичный по памяти, чем AdamW
- Большинство оптимизаций хорошо комбинируются друг с другом:



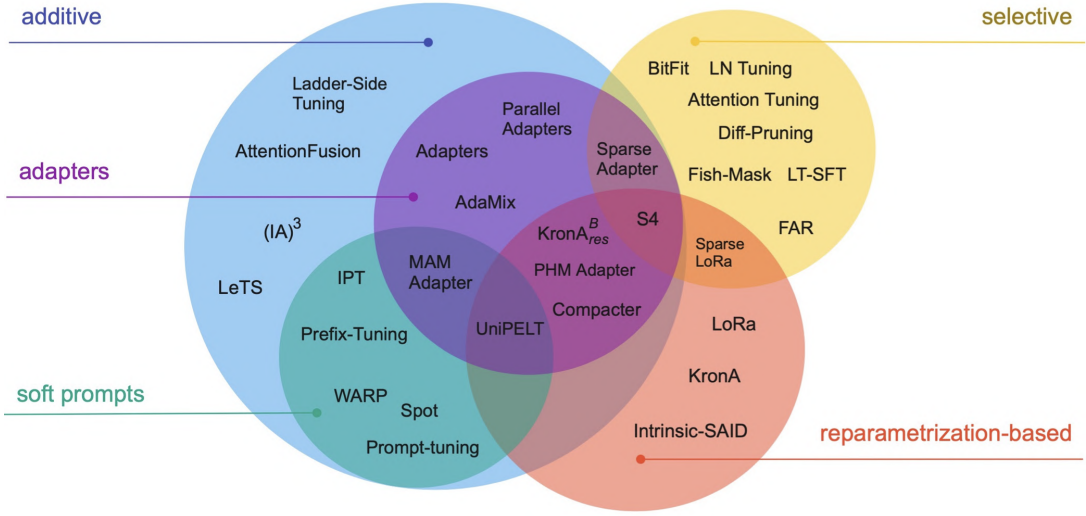
https://huggingface.co/docs/transformers/v4.20.1/en/perf_train_gpu_one

Ускорение дообучения

- Стандартный подход в использовании LLM — Transfer Learning
- Большая и умная модель адаптируется под частные задачи с помощью дообучения на небольшом наборе данных
- **Проблема:** дообучение LLM целиком может требовать больших ресурсов и времени
- **Возможное решение:** учить не всю модель, а только отдельные слои
- **Проблема:** задачи могут быть многочисленными и разнообразными — не хочется на каждую учить, хранить и хостить целую модель
- Альтернатива полному или частичному дообучению — адаптеры
- Модель остаётся неизменной, к ней как-то добавляются немного новых параметров (или используется малая часть исходных весов)
- При дообучении эти параметры настраиваются корректировать работу модели для повышения качества на целевой задаче

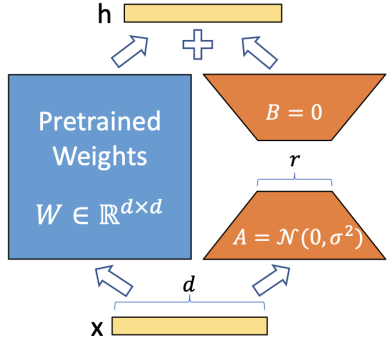
Ускорение дообучения

- Адаптеров придумали очень много:



Ускорение дообучения

- Одним из наиболее популярных методов остаётся LoRA:
  - веса модели полностью замораживаются, выбираются целевые линейные веса
  - для каждой матрицы весов заводится пара новых матриц — её низкоранговое разложение
  - при работе эти матрицы перемножаются и результат складывает с основными замороженными весами
  - хороший рецепт: добавлять адаптер на все матрицы весов запросов и значений в self-attention

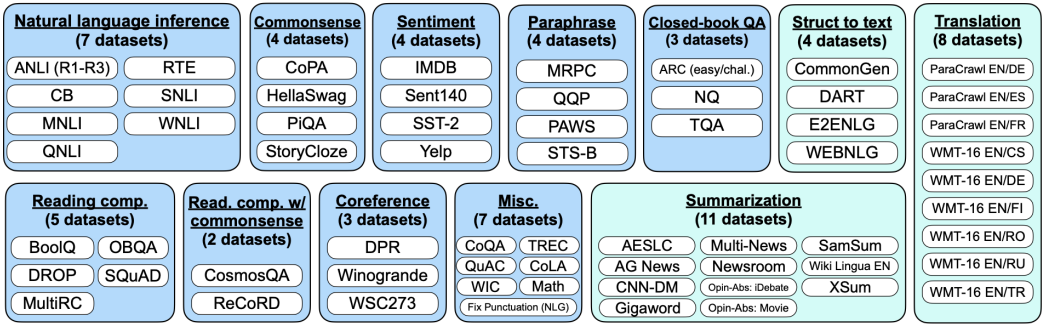


Обработка длинного контекста

- Для обхода квадратичной сложности self-attention можно:
  - понижать сложность вычислений за счёт понижения размерностей матриц
  - вычислять внимание по частям последовательности
  - обрабатывать последовательность иерархически
  - вместо увеличения длины последовательности передавать контекст рекурентно
  - вместо увеличения длины последовательности сохранять контекст в kNN-индексе
- Примеры работ:
  - Longformer: The Long-Document Transformer, 2020
  - Linformer: Self-Attention with Linear Complexity, 2020
  - Efficient Long-Text Understanding with Short-Text Models, 2022
  - Scaling Transformer to 1M tokens and beyond with RMT, 2023

Оценка качества LLM

- Способности моделей проверяются путём решения разных текстовых или мультимодальных задач на разных наборах данных
- Схема различных NLP-задач и соответствующих данных (синие — с короткими ответами, бирюзовые — с длинными):



Оценка качества LLM

- Из отдельных наборов данных формируют коллекции — бенчмарки
- Бенчмарков много для разных задач, длины контекста, доменов, языков и модальностей, примеры популярных:
  - MMLU, 2020 (тексты, английский)
  - HumanEval, 2021 (программный код, английский)
  - MMBench, 2023 (тексты и изображения, английский)
  - LongBench, 2023 (длинные тексты, английский)
  - MERA, 2023 (тексты, русский)
- Проверка коротких ответов автоматическая, с длинными сложнее — автометрики слабые, проверяют люди или более сильные LLM (GPT-4):

  «Ты выступаешь в роли асессора. Тебе покажут правильный пересказ текста и пересказ, сгенерированный моделью, твоя задача оценить по шкале от 1 до 10 качество генерации пересказа . . . »

Спасибо за внимание!

Мурат Апишев
Lead Data Scientist, SberDevices
mel-lain@yandex.ru