ecom.tech

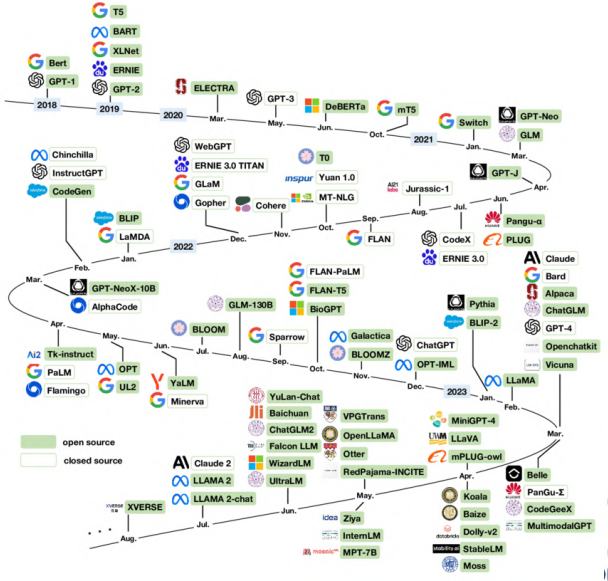
Эволюция Transformer: как меняется самая успешная архитектура в DL

Мурат Апишев, ecom.tech

AiConf 2024

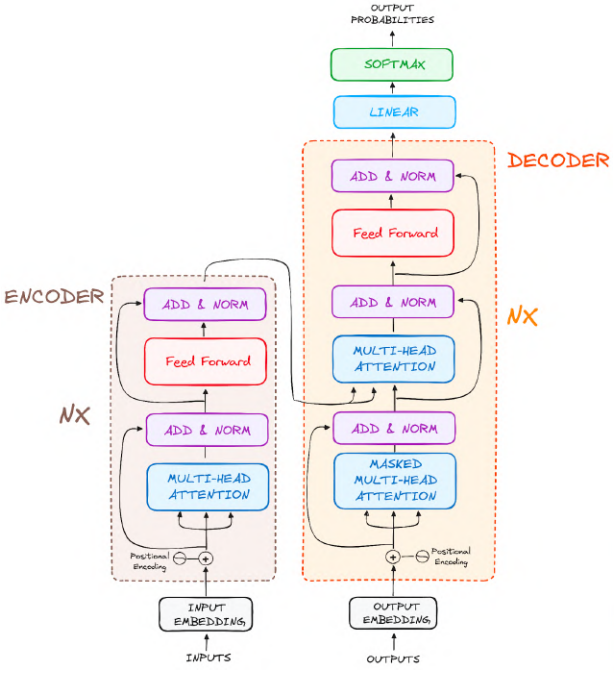
О чём будет рассказ

- LLM – основа современного NLP и AI
- Мультимодальные, мультидоменные, мультиязычные, инструктивные
- Сотни открытых и проприетарных моделей
- Доминирующая нейросетевая архитектура — Transformer
- С 2017 года предложено много архитектурных модификаций и внедрений
- **Цель доклада**: рассмотреть наиболее значимые и/или интересные из этих идей



О чём будет рассказ

- Токенизация
- Позиционное кодирование
- Нормализация и регуляризация
- Self-Attention
- Полносвязные слои
- State Space Models



Токенизация: базовые алгоритмы

- **Цель**: разбиение текст на структурные единицы для обработки моделью
- Варианты разбиения:
  - только по словам – большой словарь, проблема OOV
  - только по символам – низкое качество, очень длинный вход
  - по символьным N-грамма – компромиссный вариант
- Базовые алгоритмы:
  - BPE:
    - стартовые токены – все символы коллекции
    - склейка часто идущих подряд пар токенов до целевого размера словаря
  - WordPiece:
    - похож на BPE, но слияние не по частоте, а по правдоподобию
  - Unigram:
    - стартовые токены – все слова, символы и частые символьные N-граммы
    - удаление заданной доли токенов с минимальным ухудшением правдоподобия
- Претокенизация – разбиение текста на фрагменты перед их токенизацией

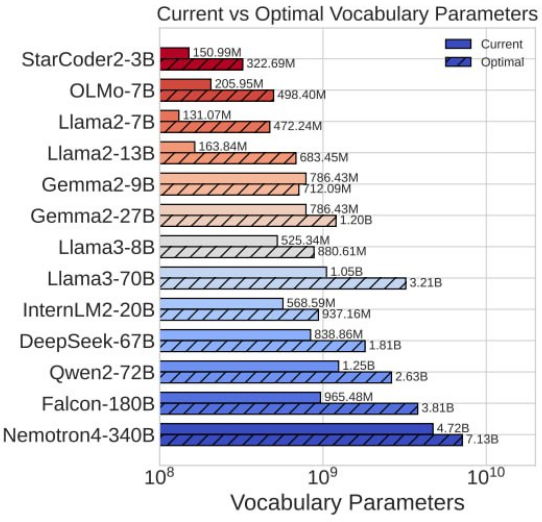
Токенизация: настройка

- Выделение цифр в отдельные токены и запрет их слияния
- Запрет на слияние алфавитных символов и цифр/пунктуации
- Byte-level BPE: стартовый словарь содержит 256 байтов, можно кодировать любые последовательности без UNK-токена
- Варьирование размера словаря:
  - больше – ниже фертильность, короче последовательности
  - меньше – меньше обучаемых параметров, больше данных на токен
- Выбираются компромиссные варианты $\sim 10^4$-10^5
- Вариант реализации алгоритма:
  - SentencePiece – построение словаря чисто по BPE
  - TikToken – предварительный отбор частотных слов в качестве токенов
- Входные и выходные эмбеддинги модели могут разделять веса (как в ванильном варианте) или иметь свои собственные (например, Qwen)

SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing, 2018
https://github.com/openai/tiktoken
Qwen Technical Report, 2023

Токенизация: настройка

- Размер словаря можно выбирать не только «на глаз»
- Строятся эмпирические оценки оптимального соотношения числа параметров и токенов в словаре (по аналогии с Chinchilla)
- Рост словаря от числа параметров сильно сублинейный
- У многих современных моделей словари слишком малы



Токенизация: разные типа претокенизации

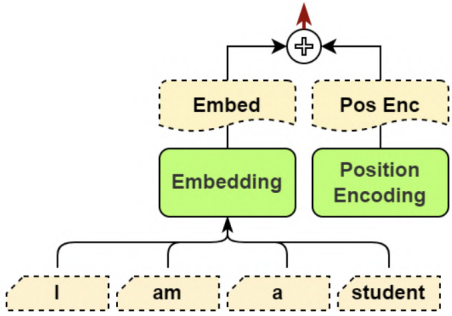
- Galactica использует разную токенизацию для разных типов данных
- Обычные слова токенизируются на обычные токены
- Специализированные типы данных (числа, аминокислоты, ДНК, формулы) токенизируются по-символьно
- для этого они:
  - предварительно помечаются спецтокенами
  - обрабатываются регулярным выражением: между каждым символом вставляется спецразделитель
- На претокенизации применяется последовательность обработчиков, первый – по спецразделителю
- Пример токенизации:
  - `QWERTY` $\rightarrow$ `QWE RTY`
  - `[AMINO_S]QWERTY[AMINO_E]` $\rightarrow$ `Q W E R T Y`

Токенизация: морфология

- Добавление морфологии для более осмысленной токенизации:
  `undesirable` → `un desirable` вместо `un des ira ble`
- FLOTA:
  - заводится словарь V слов и морфем
  - слово при токенизации ищется в V
  - если находится – оно становится токеном
  - если нет — пошагово уменьшается с обеих сторон, пока не найдётся в V
  - остаток слова проверяется до успеха или до лимита глубины рекурсии
  - всё, что не нашлось, токенизируется обученным BPE
- MorphPiece
  - заводится KV-таблица перевода слов в токены-морфемы
  - слово ищется в таблице, если есть, токены берутся из неё, нет — BPE
  - нетривиальный процесс токенизации с эвристическим алгоритмом
- Идеи интересные, но реализации сырые и слабо проверенные

Позиционное кодирование: абсолютный подход

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



Позиционное кодирование: относительный подход

- Вместо позиции токена кодируется расстояние между парой токенов, входной эмбеддинга заменяется модификацией подсчёта self-attention:

$$e_{ij} \propto \left(x_i W^Q\right)\left(x_j W^K + a_{ij}^K\right)^T, \qquad z_i = \sum_{j=1}^{n} \mathrm{sm}(e)_{ij}\left(x_{ij} W^V + a_{ij}^V\right)$$

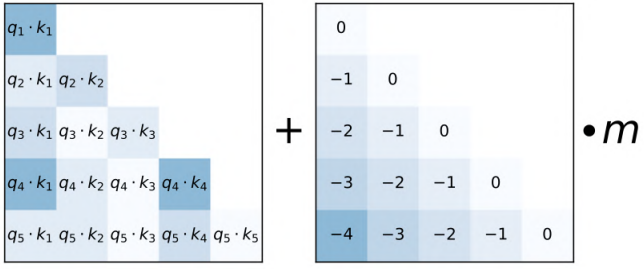
  e_{ij} – веса до нормировки, x_i, z_i – входной и выходной векторы токена

- Для всех голов внимания обучаются общие векторы $a_{ij}^K, a_{ij}^V \in \mathbb{R}^{d_z}$ на каждое расстояние $i - j$
- В общем случае формулу e_{ij} для абсолютного случая можно переписать:
  - представим x_i как сумму позиционной x_i^p и смысловой x_I^e информации
  - репараметризуем и избавимся от зависимости от x_i^p
  - новые параметры описывают относительную позиционную информацию
- Transformer-XL и DeBERTA используют разный вид и части репараметризованной формулы

Self-Attention with Relative Position Representations, 2018
Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context, 2019
DeBERTa: Decoding-enhanced BERT with Disentangled Attention, 2021

Позиционное кодирование: сдвиги

- Встроить позиционную информацию можно просто сдвигом e_{ij} перед нормировкой для получения весов α_{ij} внимания
- В T5 сдвиги – это 32 скаляра, распределённых в логарифмической шкале по 128 отступам (всё, что дальше – кодируется одним)
- Скаляры свои у каждой головы внимания, но общие для всех слоёв
- В AliBi скаляры для $i - j$ не обучаются, а считаются как $m(i - j)$, m – заданное на старте число, своё для каждой головы и общее для слоёв:



Позиционное кодирование: вращения

- RoPE – самый популярный метод позиционного кодирования, почти все современные LLM обучаются с ним или его модификациями
- Обощённая формула для e_{ij} (без учёта позиционных эмбеддингов):

$$e_{ij} \propto \langle f_q(x_i, i), f_k(x_j, j) \rangle$$

- Можно подобрать такие функции g, f_q, f_k, что будет верно равенство

$$\langle f_q(x_i, i), f_k(x_j, j) \rangle = g(x_i, x_j, i - j)$$

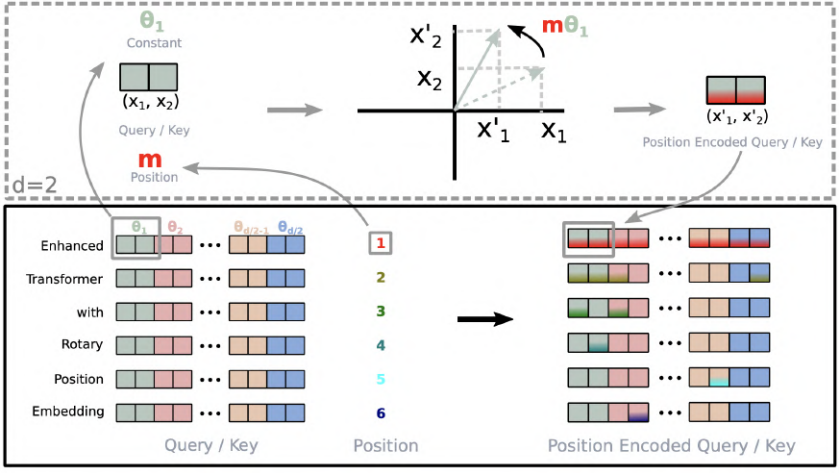
- Можно доказать, что для $d_z = 2$ подойдёт

$$f_q(x_i, i) = \begin{pmatrix} \cos i\theta & -\sin i\theta \\ \sin i\theta & \cos i\theta \end{pmatrix} \begin{pmatrix} W_{11}^Q & W_{12}^Q \\ W_{21}^Q & W_{22}^Q \end{pmatrix} \begin{pmatrix} x_i^1 \\ x_i^2 \end{pmatrix}$$

- Преобразование легко обобщается на любое чётное d_z – достаточно применить его к каждой паре координат x_i

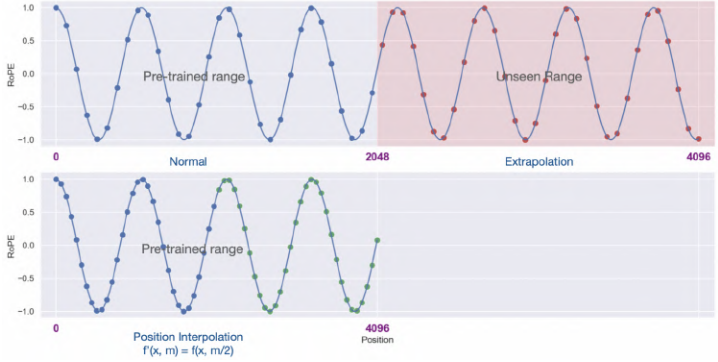
Позиционное кодирование: вращения

- Применение RoPE заключается в повороте вектора запроса/ключа на угол, зависящий от индекса его позиции
- Поворот обоих векторов на один угол, т.е. смещение позиций без изменения расстояния, сохранит значение скалярного произведения



Позиционное кодирование: вращения с интерполяциями

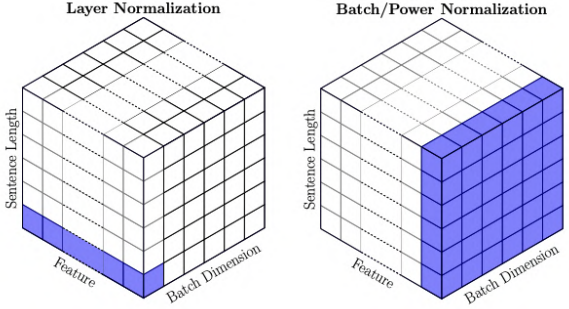
- RoPE, как и прочие методы, слабо адаптируется к росту длины контекста
- Идея PI RoPE и SuperHOT RoPE: обучить модель с RoPE и вложить увеличенный контекст в тот же диапазон с минимальным дообучением
- Модификации PI RoPE точнее настраивают сжатие диапазона для каждой пары признаков в RoPE, улучшают качество и не требуют дообучения



Extending Context Window of Large Language Models via Positional Interpolation, 2023
https://kaiokendev.github.io/tilextending-context-to-8k
YaRN: Efficient Context Window Extension of Large Language Models, 2023

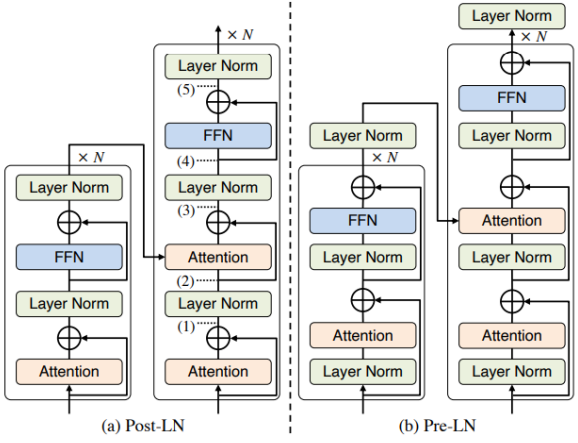
Нормализация: LayerNorm

- Нормализация повышает скорость и стабильность обучения
- В моделях CV распространена более старая Batch-нормализация, но в Transformer её использовать неудобно:
  - нужны большие батчи для сбора статистики
  - в NLP у последовательностей разная длина $\rightarrow$ флуктуации статистик
- Альтернатива – Layer-нормализация вектора каждого токена



Нормализация: порядок встраивания

- В оригинальном Transformer слои нормализации идут после MHSA и FFN с residual connections (Post-LN)
- Это вариант менее стабилен и больше подвержен влиянию LR и warmup, в современных моделях доминирует Pre-LN
- Вариации типа Sub-LN в Magneto (Pre-LN + дополнительный LayerNorm внутри блоков MHSA и FFN) пока не получили широкого распространения



On Layer Normalization in the Transformer Architecture, 2020
Magneto: A Foundation Transformer, 2022
https://teletype.in/@gedomada/fclTOHhytDy

Нормализация: RMSNorm

- LayerNorm борется с проблемой сдвига весов и входов путём центрирования и нормирования на отклонение активаций a
- Изначально считалось, что обе операции существенно важны
- RMSNorm – модификация LayerNorm, в которой нет центрирования:

$$\hat{a}_i = \frac{a_i - \cancel{\mu}}{\sigma}, \quad \cancel{\mu = \frac{1}{n}\sum_{i=1}^{n} a_i}, \quad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_i - \cancel{\mu})^2} = \mathrm{RMS}(a)$$

- Такой вариант сохраняет качество и быстрее считается
- RMSNorm стал стандартом для современных LLM (LLaMA, Qwen, Mistral)

Model	Time
Baseline	315±6.30s
BatchNorm-Everywhere	348±10.5s
BatchNorm-LSTM	345±11.2s
LayerNorm	392±5.70s
RMSNorm	333±5.20s (15.1%)
pRMSNorm	330±5.50s (15.8%)

Регуляризация: Dropout

- Базовый вариант применения с $p_{\text{dropout}} = 0.1$:
  - к выходу каждого блока MHSA и FFN до residual connection
  - к сумме эмбеддингов токенов и позиций
- Data Dropout – удаление целых токенов
- Layer Dropout – удаление целых слоёв
- Feature Dropout – обычное удаление нейронов, но в блоках MHSA и FFN
- DropHead – удаление целых голов MHSA
- DropAttention – удаление весов для последовательности в MHSA
- Curriculum Dropout – управление долей удаляемых нейронов:
  - на старте обучения лучше удалять меньше, чтобы не мешать переходу весов из случайной инициализации в осмысленные структуры
  - далее усиление dropout уменьшает взаимозависимость нейронов

UniDrop: A Simple yet Effective Technique to Improve Transformer without Extra Cost, 2021
Scheduled DropHead: A Regularization Method for Transformer Models, 2020
DropAttention: A Regularization Method for Fully-Connected Self-Attention Networks, 2019
Curriculum Dropout, 2017

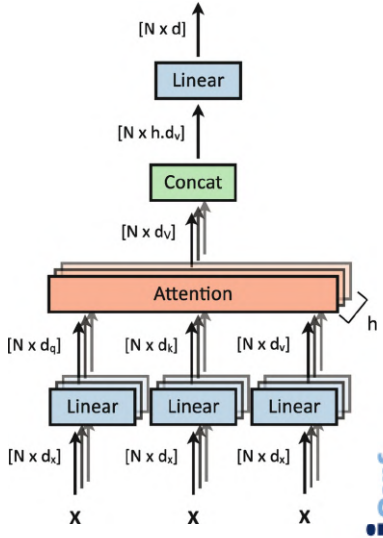
Регуляризация: Dropout

- DropDim:
  - вместо нейронов зануляется часть элементов выходного вектора с перенормировкой оставшихся
  - применяется к каждому блоку MHSA и FFN после residual connection
  - удалятся могут как отдельные случайные элементы, так и спаны
- LayerShuffle – развитие Layer Dropout, не просто удаление слоёв на обучении и инференсе, но и их перемешивание:
  - LayerShuffle – порядок слоёв случайно меняется для каждого батча
  - LayerShuffle-position – слой получает эмбеддинг своей текущей позиции
  - LayerShuffle-predict – предсказывает свою позицию по своему выходу
- Слои робастных моделей можно извлекать и перемешивать
- Работ по dropout-регуляризации много, но пока подходы остаются нишевыми за исключением идеи DropAttention

Механизм внимания: базовый MHSA

$$e_{ij} \propto \left(x_i W^Q\right)\left(x_j W^K\right)^T, \quad z_i = \sum_{j=1}^{n} \mathrm{sm}(e)_{ij}\left(x_{ij} W^V\right)$$

- Основные проблемы:
  - сложность подсчёта SA $O(n^2)$
  - потребление памяти при кэшировании векторов K и V
- В результате прямое увеличение длины контекста неэффективно
- Идеи модификаций MHSA:
  - разреженное и локальное внимание
  - рекуррентная обработка
  - понижение размерностей векторов
  - использование KNN-индексов
  - иерархическая обработка
  - уменьшение числа наборов K и V



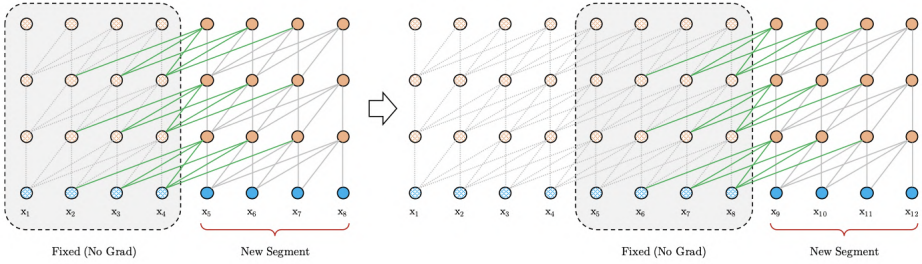
Механизм внимания: разреженность и локальность

- Матрица весов внимания существенно разреженная
- Если вычислять только нужные подмножества, можно существенно уменьшить вычисления с небольшим ухудшением качества
- Основные варианты подсчёта весов внимания для токена:
  - с непосредственными соседями (внутри групп или SWA)
  - с соседями с заданным Dilation Rate разреживанием (которое можно усиливать на верхних слоях)
  - со случайными токенами последовательности
  - с глобальными токенами (они со всеми, все с ними)
- Работы: Sparse Transformer, Longformer, Big Bird, LongT5, LongNet
- Те же идеи используют GPT-3, Qwen, Mistral, Qwen 2

Generating Long Sequences with Sparse Transformers, 2019
Longformer: The Long-Document Transformer, 2020
Big Bird: Transformers for Longer Sequences, 2020
LongT5: Efficient Text-To-Text Transformer for Long Sequences, 2022
LongNet: Scaling Transformers to 1,000,000,000 Tokens, 2023
Language Models are Few-Shot Learners, 2020
Qwen Technical Report, 2023
Mistral 7B, 2023
Qwen2 Technical Report, 2024

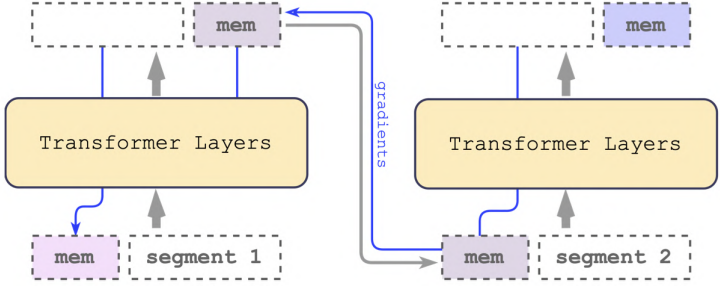
Механизм внимания: рекурентность по сегментам

- **Идея:** разделить длинную последовательность на сегменты, обрабатывать один за другим и передавать вперёд информацию
- Transformer-XL:
  - выходы self-attention текущего сегмента кэшируются во всех блоках
  - при обработке i-го сегмента из кэша берутся выходы $(i-1)$-го
  - выходы двух сегментов конкатенируются, по ним считаются K и V
  - Q – только по токенам текущего сегмента, градиенты тоже только по ним



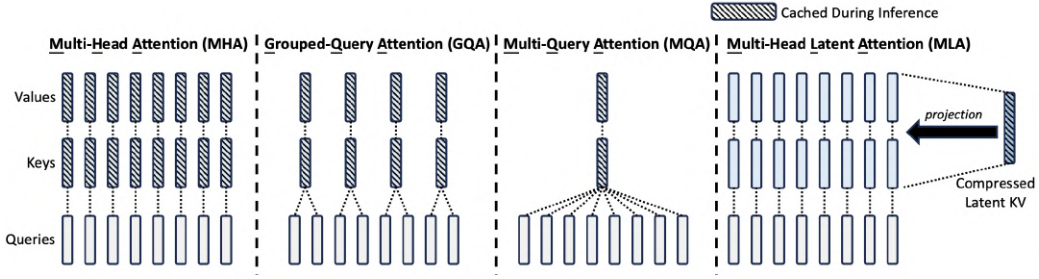
Механизм внимания: рекурентность по сегментам

- RMT:
  - в начало каждого сегмента добавляются M токенов – «векторов памяти»
  - если в основе кодировщик – выходные векторы i-го сегмента для этих токенов идут первыми на вход для $i + 1$-го сегмента
  - если декодировщик – векторы памяти есть ещё и в конце, первые несут информацию из $i - 1$-го сегмента, а векторы последних понесут в $i + 1$-й
  - механизм добавляется в модель без архитектурных изменений



Механизм внимания: число ключей и значений

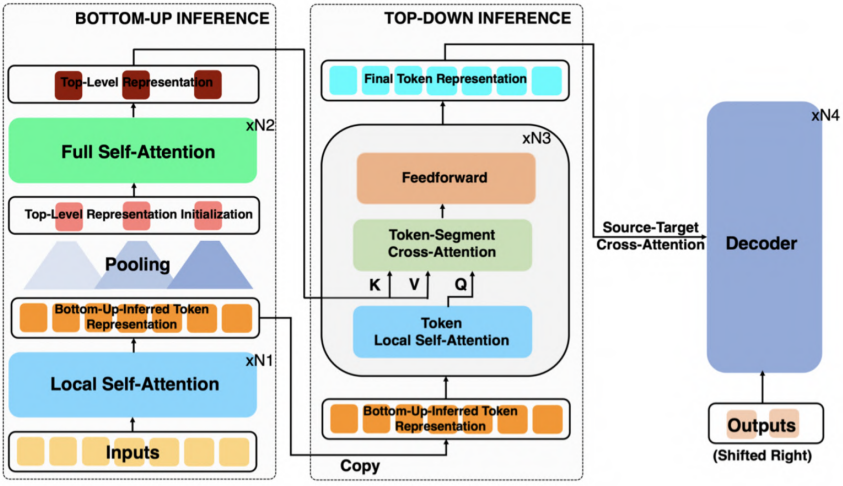
- Q должны быть свои для каждой головы внимания, K и V – нет, их можно разделять между головами для экономии вычислений и памяти
- MQA – один набор K и V на все головы
- GQA – один набор K и V на каждую группу голов
- MLA
  - на токен вычисляется и кэшируется один вектор большей размерности
  - он проектируется общей весовой матрицей в набор K/V для всех голов



Fast Transformer Decoding: One Write-Head is All You Need, 2019
GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints, 2023
DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model, 2024

Механизм внимания: иерархическая обработка

- Top Down Transformer: локальное внимание + полный Cross-SA



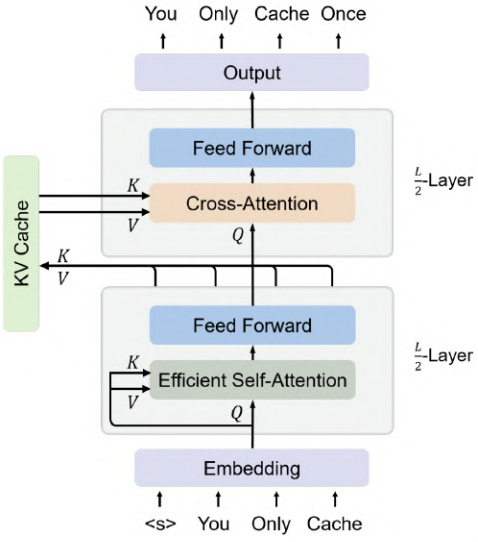
- SLED: локальное внимание с контекстом, Cross-SA в декодировщике

Механизм внимания: иерархическая обработка

- Кодировщик:
  - рассчитывает self-attention параллельно для всей последовательности
  - для генерации должен пересчитывать все заново для каждого токена
- Декодировщик:
  - рассчитывает self-attention эффективно каждого для токена
  - должен хранить большие кэши и долго обрабатывает префикс последовательности перед генерацией
- Полный Transformer:
  - обрабатывает префикс быстро
  - рассчитывает self-attention эффективно каждого для токена
  - при генерации не использует кодировщик и тоже хранит большие кэши
- **Идея:** архитектура типа «кодировщик-декодировщик», но с
  - авторегрессионной обработкой префикса
  - использованием кодировщика и для генерируемых токенов
  - уменьшением размеров кэшей в декодировщике

Механизм внимания: иерархическая обработка

- **YOCO**: декодировщик разбивается на два блока равного размера по слоям, оба обрабатывают весь вход
- Нижний – обычный декодировщик, но за счёт эффективного MHSA (SWA, gRetNet) работает быстро и с кэшем размера $O(1)$
- Его выходы – кэш K и V для верхнего блока слоёв
- В верхнем полный MHSA, но с Cross-SA, в котором на всех слоях общие K и V из кэша
- Префикс обрабатывается только нижним блоком – ускорение



Механизм внимания: сегменты и kNN-индексы

- Memorizing Transformer:
  - Transformer-XL, в последнем блоке добавляется kNN-Augmented Attention
  - блок памяти для K и V размера M, добавление в конец с вытеснением
  - для Q ищутся kNN, с ними считается MHSA (блок на голову), результаты складываются с обычными с обучаемым весом
- Unlimiformer:
  - полный Transformer, сегмент с контекстом обрабатывается локальным вниманием в кодировщике, кэшируются его выходы h_e
  - слои декодировщика используют kNN-Augmented Attention в Cross-SA
  - хранение h_e (общих для всех слоёв) вместо K и V экономит память

$$QK^T = (h_d W^Q)(h_e W^K)^T = (h_d W^Q)((W^K)^T h_e^T) = (h_d W^Q (W^K)^T) h_e^T$$

- Focused Transformer (LongLLaMA): идея схожа с Memorizing Transformer, упор на технику обучения модели работе с кэшем

Memorizing Transformers, 2022
Unlimiformer: Long-Range Transformers with Unlimited Length Input, 2023
Focused Transformer: Contrastive Training for Context Scaling, 2023

Механизм внимания: RetNet

- RetNet близок к SSM (будет далее), хотя так не позиционируется
- Рекуррентная формулировка для одномерных входа и выхода v_n и o_n:

$$s_n = As_{n-1} + K_n^T v_n, \qquad o_n = Q_n s_n = \sum_{m=1}^{n} Q_n A^{n-m} K_m^T v_m$$

  s_n – вектор состояния, A – матрица перехода, K_n, Q_n – векторы, получаемые умножением входа на обучаемые матрицы W_Q и W_K

- Диагонализация матрицы A и представление её в специальном виде позволяют свести рекурентное вычисление к параллельному
- Для входа x из набора векторов и матрицы параметров W_V для получения v_n

$$Q = (xW_Q) \odot \Theta, \qquad K = (xW_K) \odot \overline{\Theta}, \qquad V = xW_V$$

$$\text{Retention}(x) = (QK^T \odot D)V$$

  с поправкой на дополнительные матрицы параметров Θ и D (зависит от γ) получается похоже на MHSA (с xPos – разновидностью RoPE)

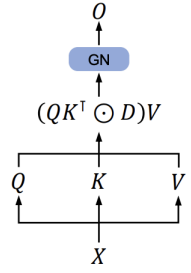
Retentive Network: A Successor to Transformer for Large Language Models, 2023
A Length-Extrapolatable Transformer, 2022

Механизм внимания: RetNet

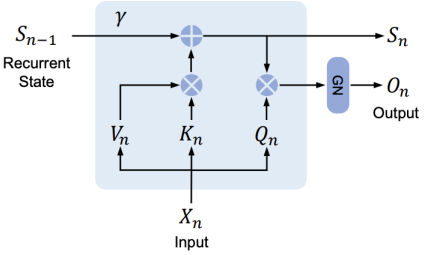
- Для инференса Retention сохраняется рекурентное представление:

$$S_n = \gamma S_{n-1} + K_n^T V_n, \qquad \text{Retention}(x_n) = Q_n S_n$$

Q, K, V, γ – те же, что и параллельном варианте



(a) Parallel representation.

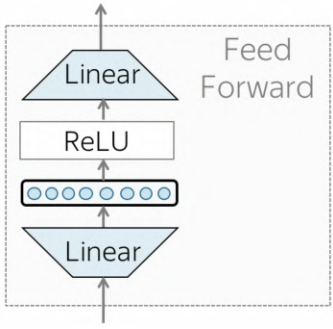


(b) Recurrent representation.

- Гибридное представление для обучения на длинном входе:
  - вход разбивается на части, каждая обрабатывается параллельно
  - между частями обработка рекуррентная (родственно Cross-SA)
- Блок RetNet – Multi-Head Retention + LayerNorm + FFN (как в Transformer)

Полносвязные слои: базовый FFN

- Базовый вариант (без bias): $\max(xW_1, 0)W_2$
- Блоки FFN занимают до 2/3 от общего числа параметров сети
- Self-attention определяет важность прочих токенов для текущего
- FFN обрабатывает эту информацию и содержит основные «знания» модели, включая фактологию



- FFN может рассматриваться как KV-хранилище, первый слой содержит ключи, второй – значения
- Есть попытки редактировать и встраивать факты через изменение весов FFN обученной модели

https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html
Transformer Feed-Forward Layers Are Key-Value Memories, 2021
Mass-Editing Memory in a Transformer, 2023
Locating and Editing Factual Associations in GPT, 2023

Полносвязные слои: функции активации

- FFN с ReLU – не единственный возможный вариант блока с полносвязными слоями
- Пробуют различные активации:
  - $\mathrm{GeLU} = x\Phi(x)$, $\Phi(x)$ – функция распределения $\mathcal{N}(0, 1)$
  - $\mathrm{Swish} = x\sigma(\beta x)$, $\sigma(x)$ – сигмоида, β – гиперпараметр (при $\beta = 1$ – SiLU)
  - $\mathrm{Mish} = x\tanh(\mathrm{softplus}(x))$, $\mathrm{softplus}(x) = \ln(1 + x)$
- Все функции – гладкие модификации ReLU без проблемы большого числа нулевых бесполезных нейронов и резкого перехода в нуле
- Несмотря на эксперименты в разных работах, нет однозначного мнения о том, какая из них лучше
- GeLU используется в FFN блоках BERT и GPT-2

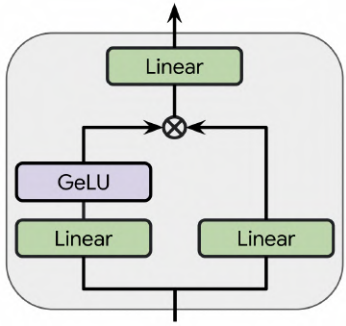
Полносвязные слои: GLU

- Gated Linear Units:

  $$\mathrm{GLU}(x) = \sigma(xW_1) \otimes xW_2,$$

  $\otimes$ – покомпонентное умножение

- В GLU тоже можно заменять активации, популярные варианты
  GeGLU (GeLU) и SwiGLU (Swish / SiLU)



- В блоке FFN можно заменить первый полносвязный слой и функцию активации на GLU
- Весовых матриц становится 3, для сохранения общего числа параметров внутренняя размерность блока уменьшается на треть
- Модификация показывает хорошие результаты и активно используется в известных моделях (LLaMA, Qwen, Mistral, Griffin)

Полносвязные слои: KAN

- Основная задача нейронной сети – приблизить на основе данных некоторую функцию в многомерном пространстве аргументов
- FFN работают благодаря универсальной теореме аппроксимации:

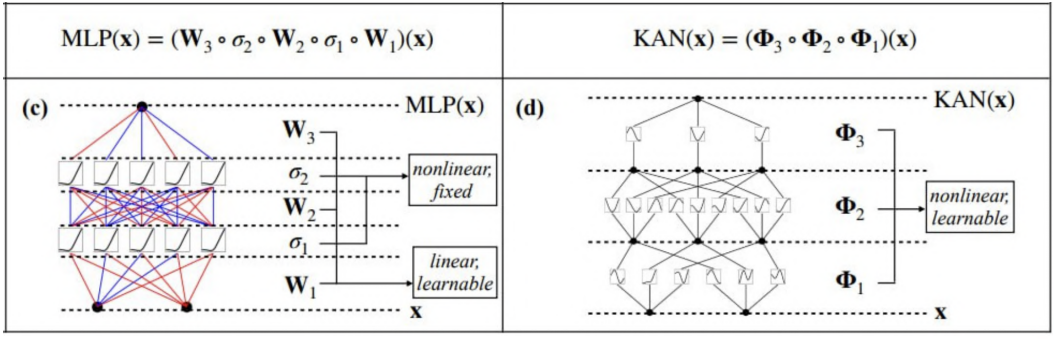
> Сеть прямого распространения с одним слоем с сигмоидальной активацией может приблизить любую функцию многих переменных с заданной точностью, при условии достаточного числа нейронов и оптимально подобранных весов

- FFN:
  - рёбра сети – обучаемые веса
  - узлы сети – фиксированные активации
- Альтернативный подход основан на теореме Колмогорова-Арнольда:

> Любая непрерывная функция многих переменных может быть представлена в виде суперпозиции непрерывных функций одной переменной с использованием сложения

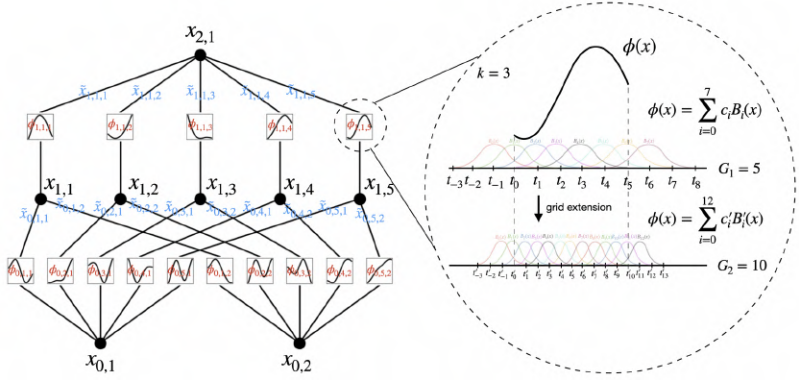
Полносвязные слои: KAN

- KAN:
  - рёбра сети – обучаемые активации
  - узлы сети – операция сложения
- Теорема описывает двухслойную KAN с заданной размерностью слоёв $n \to 2n+1 \to 1$
- Глубокий KAN строится инженерно путём стекинга слоёв



Полносвязные слои: KAN

- Каждая функция активации имеет вид $\phi(x) = w_b\, b(x) + w_s\, s(x)$, где
  - $b(x) = \mathrm{SiLU(x)}$ – по сути residual connection с активацией
  - $s(x) = \sum_i c_i B_i(x)$ – линейная комбинация сплайнов, веса c_i обучаемые
  - w_b, w_s – обучаемые веса слагаемых (в целом не очень нужные)
- Кубические B-сплайны локальны и хорошо комбинируются для аппроксимации сложных функций



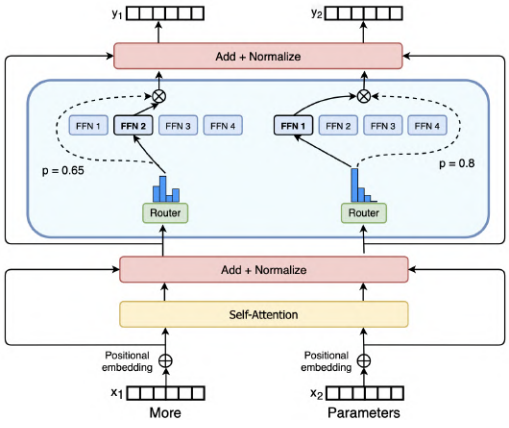
Полносвязные слои: KAN

- В сети с L слоями ширины N число параметров у
  - FFN – $O(N^2L)$
  - KAN со степенью сплайнов k на G интервалах – $O(N^2L(G+k))$
- FFN более эффективен с т.з. числа параметров и учится на порядок быстрее, но KAN для того же качества требует сильно меньшего N
- Меньше N + сплайны вместо весов = лучше интерпретируемость
- KAN выглядит перспективно и активно развивается:
  - работы по модификации сети (полиномы Чебышева, вейвлеты или коэффициенты Фурье вместо сплайнов)
  - работы по повышению производительности (efficient-kan)
- Но экспериментальных доказательств превосходства над FFN в общем случае недостаточно, часть результатов противоречивы

Chebyshev Polynomial-Based Kolmogorov-Arnold Networks, 2024
https://github.com/GistNoesis/FourierKAN
Wav-KAN: Wavelet Kolmogorov-Arnold Networks, 2024
https://github.com/Blealtan/efficient-kan

Полносвязные слои: Mixture-of-Experts

- Если учить оптимально, то больше весов ⇒ выше качество
- Но больше весов ⇒ медленнее инференс
- Решение – Mixture-of-Experts, «экспертами» становятся FFN-слои
- Вектор слова после self-attention идёт Router, тот направляет его к лучшим экспертам



- Параметров сильно больше, но при обработке каждого токена активируется небольшая часть
- Эксперт учится решать свои типы задач, имеет свои доменные знания

Полносвязные слои: проблемы MoE

- Нагрузка распределяется по экспертам неравномерно, из-за чего сеть вырождается, качество падает
- Изначально подход внедрялся в огромные приватные модели, что замедляло его популяризацию
- Вместо доменной информации эксперты улавливают родственные токены: имена, артикли, спецсимволы, определённые части речи

Punctuation	Layer 2	, , , , , , , , , - , , , , ,).)
	Layer 6	, , , , , : . : , & , & & ? & - , , ? , , , . <extra_id_27>
Conjunctions and articles	Layer 3	The the the the the the the the the The the the the the the The the the the
	Layer 6	a and and and and and and and or and a and . the the if ? a designed does been is not
Verbs	Layer 1	died falling identified fell closed left posted lost felt left said read miss place struggling falling signed died falling designed based disagree submitted develop

Полносвязные слои: развитие MoE, Mixtral

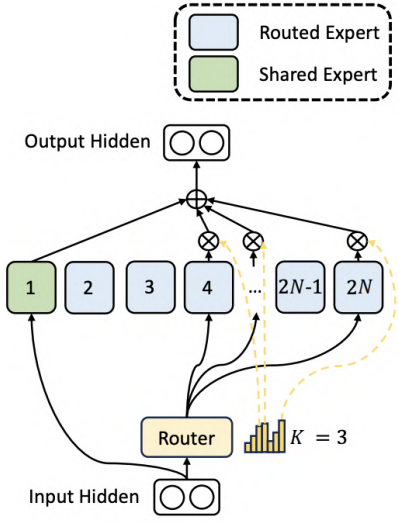
- Регуляризация loss для Router для балансировки распределения и добавление ограничений на ёмкость эксперта
- Популярная открытая сравнительно небольшая модель – Mixtral
  - 8 экспертов – блоков FFN со SwiGLU, 2 активных на токен
  - 47B параметров в целом, 13B активных на токен
  - эффективная реализация за счёт разреженного матричного умножения

```
Question: Solve -42*r + 27*c = -1167 and 130*r
Answer: 4

Question: Calculate -841880142.544 + 411127.
Answer: -841469015.544

Question: Let x(g) = 9*g + 1. Let q(c) = 2*c +
Answer: 54*a - 30
```

```
A model airplane flies slower when flying into th
wind and faster with wind at its back. When launch
right angles to the wind, a cross wind, its ground
compared with flying in still air is
(A) the same (B) greater (C) less (D) either grea
or less depending on wind speed
```

Полносвязные слои: развитие MoE, DeepSeekMoE

- DeepSeekMoE:
  - вектор токена разделяется на части, каждая идёт своему эксперту
  - экспертов становится больше, у каждого меньшая размерность
  - добавляются общие эксперты, в которые части токена попадают всегда
- При том же объёме вычислений и числе параметров качество и скорость инференса выше, чем у обычного MoE
- Эксперты более доменные и важные – удаление 2-3 ощутимо роняет перплексию
- В версии 2 добавляется Multi-Head Latent Attention для уменьшения кэшей



DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models, 2024
DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model, 2024

Полносвязные слои: развитие MoE, PEER

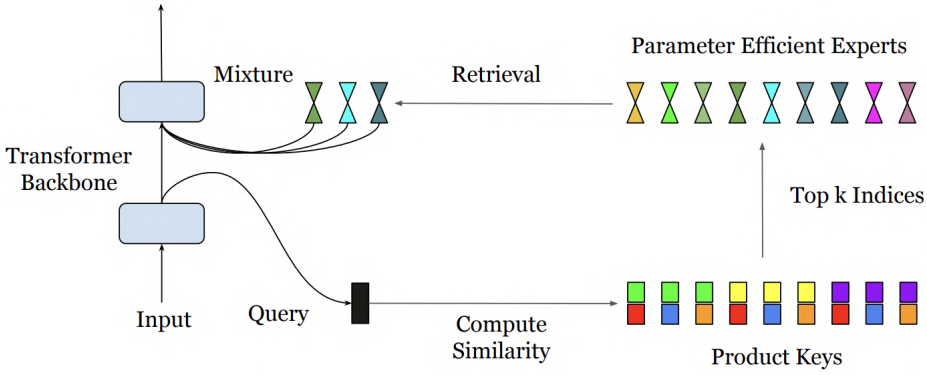
- Много небольших узких экспертов может быть лучше, чем мало больших и общих, предположения:
  - выше качество в экспериментах
  - лучше интерпретируемость
  - возможность обучения на потоках данных без забывания
- PEER – обучения MoE с большим числом экспертов из одного нейрона:
  - слой – набор из N экспертов

$$e_i(x) = \sigma(u_i^T x) v_i, \quad x, u_i, v_i \in \mathbb{R}^n, \quad i = 1, \dots N$$

    и их ключей $k_i \in \mathbb{R}^d,\ i = 1, \dots N$

  - вектор токена x проектируется в запрос $q(x) \in \mathbb{R}^d$
  - подсчёт как в self-attention, выбираются топ-K лучших экспертов
  - векторы $e_i(x)$ выбранных экспертов складываются с весами из softmax
  - как и в MHSA, можно по x генерировать $h > 1$ запросов $q(x)$

Полносвязные слои: развитие MoE, PEER



- Если $K = 1$, то h голов такого MoE равны по размеру одному эксперту обычного MoE с h нейронами
- Но вместо фиксированной матрицы получаются разные комбинации строк для разных токенов и запросных весов
- Слой может заменять FFN или встраиваться между обученными блоками

State Space Models: основы

- Основа Transformer – Self-Attention, его можно не оптимизировать, а заменить на новый архитектурный компонент
- Transformer: качественно, но неэффективно (запоминается весь вход)
- RNN: некачественно, но эффективно (запоминается лишь вектор состояния)
- SSM: решение для качественной и быстрой обработки длинных контекстов
- В обзоре Mamba-360 описывается таксономия из более 20 моделей в 4 группах
- Пусть $h(t)$ – функция состояния, $x(t)$ и $y(t)$ – одномерные входной и выходной сигналы, A, B, C, D – матрицы параметров, тогда SSM выглядит так:

$$h'(t) = Ah(t) + Bx(t), \qquad y(t) = Ch(t) + Dx(t)$$

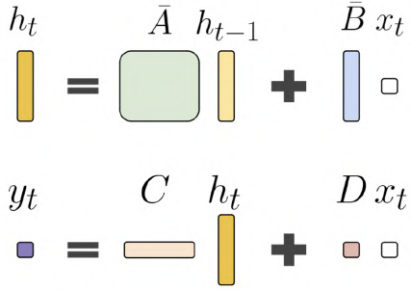
- Для применения в обработке токенов нужна дискретизация с шагом Δ, пусть $h_t = h(t)$, $h_{t+1} = h(t + \Delta)$, тогда

$$h_t = \overline{A}h_{t-1} + \overline{B}x_t, \qquad y_t = Ch_t + Dx_t$$

- $\overline{A}$ и $\overline{B}$ могут считаться по-разному в зависимости от дискретизации

State Space Models: основы

- Смысл наборов параметров:
  - $\overline{A}$ – что забыть из текущего состояния
  - $\overline{B}$ – что запомнить из текущего входа
  - C – как использовать состояние для предсказания
  - D – как использовать вход для предсказания



- Δ определяет размер шага – силу фокуса на текущий вход x_t
- D – вариант residual connection, легко считается, обычно опускается
- Наивная реализация обрабатывает элементы один за другим (как RNN)
- Для работы с N-мерным входом создаётся N копий базовой модели, обучаемых совместно (расширение состояния, чего нет в RNN)
- В отличие от обычных RNN и LSTM, в SSM нет нелинейных активаций
- Рекурентные формулы SSM можно переписать в виде дискретной свёртки

State Space Models: S4

- В SSM билинейная дискретизация даёт формулы

$$\overline{A} = (I - \Delta/2 \cdot A)^{-1}(I + \Delta/2 \cdot A), \qquad \overline{B} = (I - \Delta/2 \cdot A)^{-1}\Delta B$$

- На этапе обучения рекуррентная форма преобразуется в свёрточную

$$y = x * \overline{K}, \qquad \overline{K} = (C\overline{B}, C\overline{AB}, \dots, C\overline{A}^{K}\overline{B}, \dots)$$

- В «чистом» виде такая модель непригодна для работы:
  - матрица A произвольного вида приводит к затуханию/взрыву градиентов
  - наивное вычисление свертки $\overline{K}$ вычислительно ёмкое
- Проблема с A решается путём инициализации т.н. HiPPO матрицей
- В S4 предлагается метод Diagonal Plus Low-Rank (DPLR) для эффективного подсчёта свёртки
  - вводится репараметризация специального вида для A, все матрицы параметров в комплексном пространстве
  - вместо прямого вычисления свёртки – вычисление некоторого ядра DPLR

State Space Models: Mamba

- Параметры модели в S4 не зависят от входа (Linear Time Invariance):
  - рекурентную формулу можно выразить свёрткой и эффективно обрабатывать последовательность целиком
  - хуже улавливаются сложные зависимости, ниже качество
- Mamba основана на Selective SSM: параметры B, C и Δ становятся зависимыми от входа (моделирование линейными слоями)
- Качество работы растёт, но исчезает возможность вычисления свёрткой

Algorithm 1 SSM (S4)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
$\boldsymbol{C} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
$\Delta : (\mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter})$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{D}, \mathsf{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
return y

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_C(x)$
$\Delta : (\mathsf{B}, \mathsf{L}, \mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{B}, \mathsf{L}, \mathsf{D}, \mathsf{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
return y

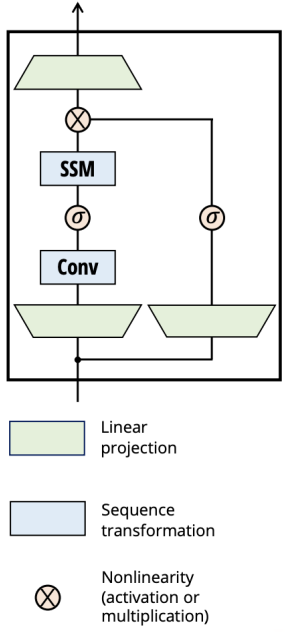
State Space Models: Mamba

- Для массива чисел массив его кумулятивных сумм вычисляется параллельно: reduce с сохранением промежуточных значений + обратный проход
- Эта процедура применима для любой последовательности с бинарным оператором, удовлетворяющим ряду условий
- Mamba SSM схожа с Linear RNN и вычисление последовательности её состояний можно произвести параллельно, правильно задав входы и оператор
- Как и FlashAttention для MHSA, SSM в Mamba можно оптимизировать вычислительно и по памяти:
  - параметры модели (Δ, A, B, C) сразу копируются из HBM в быструю память SRAM, где выполняются и дискретизация, и сканирование
  - оба шага вместе с умножением на C реализуются одним ядром (fusing)
  - длинный контекст разбивается на части, промежуточные значения после обработки предыдущих частей остаются в SRAM
  - нужные промежуточные состояния на backward-шаге пересчитываются

Prefix Sums and Their Applications, 1990
FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness, 2022

State Space Models: Mamba

- Верхнеуровнево Mamba похожа на Transformer:
  - на входе эмбеддинги токенов (в оригинале – без позиционной информации)
  - модель состоит из блоков Mamba
  - каждый блок – GLU со вставкой в линейный путь свёрточного слоя, активации и SSM
- Сравнения Mamba и Transformer (с Flash Attention 2) равного размера:
  - качество у Mamba во многих задачах выше
  - лучше масштабирование по длине входа
  - потребление памяти и скорость чуть хуже
- В экспериментах Mamba превосходит другие SSM
- Это самый перспективный кандидат для замены Transformer с MHSA



State Space Models: Mamba 2 и Hydra

- В Mamba 2 существенно развивается теория структурных SSM:
  - доказывается эквивалентность SSM и алгоритмов перемножения матриц специального вида
  - развивается теория Structured Masked Attention на основе Linear Attention
  - вводится общий теоретический фреймворк Structured State Space Duality
  - в его рамках показывается близость SSM и SMA
- На практике Mamba 2
  - учится быстрее первой версии
  - имеет выше качество, в т.ч. за счёт увеличения вектора состояния
- Помимо статьи авторы выпустили набор хороших постов
- SSM могут быть адаптированы для двунаправленной обработки последовательности и конкурировать с кодировщиками Transformer
- Яркий свежий пример – модель Hydra от создателей Mamba

Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention, 2020
Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality, 2024
https://goombalab.github.io/blog/2024/mamba2-part1-model
Hydra: Bidirectional State Space Models Through Generalized Matrix Mixers, 2024

Основные выводы

- При обучении LLM на основе Transformer выделились общепринятые практики:
  - Byte-level BPE
  - RoPE и его вариации
  - Pre-LayerNorm, RMSNorm
  - Attention Dropout
  - SwiGLU
  - Sparse Attention, SWA, GQA
  - MoE
- Из множества исследований больше всего выделяются модели типа SSM:
  - рекуррентная природа с возможностью эффективных вычислений соединяет лучшие черты Transformer и RNN
  - активное развитие теории и техник работ с GPU/TPU создаёт фундамент для возможного сдвига парадигмы
- Transformer активно применяется в мультимодальных моделях и в RAG-системах – темы для отдельных докладов

Спасибо за внимание!

Мурат Апишев, ecom.tech
Технический руководитель
поиска для «Мегамаркета»
mel-lain@yandex.ru

ecom.tech

AiConf 2024