курс «Глубокое обучение»

Трансформеры++

Александр Дьяконов

16 мая 2022 года

План – варианты трансформера

- **Архитектура** (ex: кодировщик, декодировщик, кодировщик+декодировщик)
- **Параметры** (число слоёв, головок, распределение внимания и т.п.)
- **Входы** (символы / BPE / слова / пиксели) **есть отдельная лекция**
- **Позициональное кодирование** **дальше**
- **Размерность** (1D, 2D и т.д.)
- **Рекуррентность / способ применения** (по сегментам, по глубине и т.п.)
- **Память** (уменьшение)
- **Механизм внимания** (исходный, sparse, локальный, линейный и т.п.)

Дальше примеры

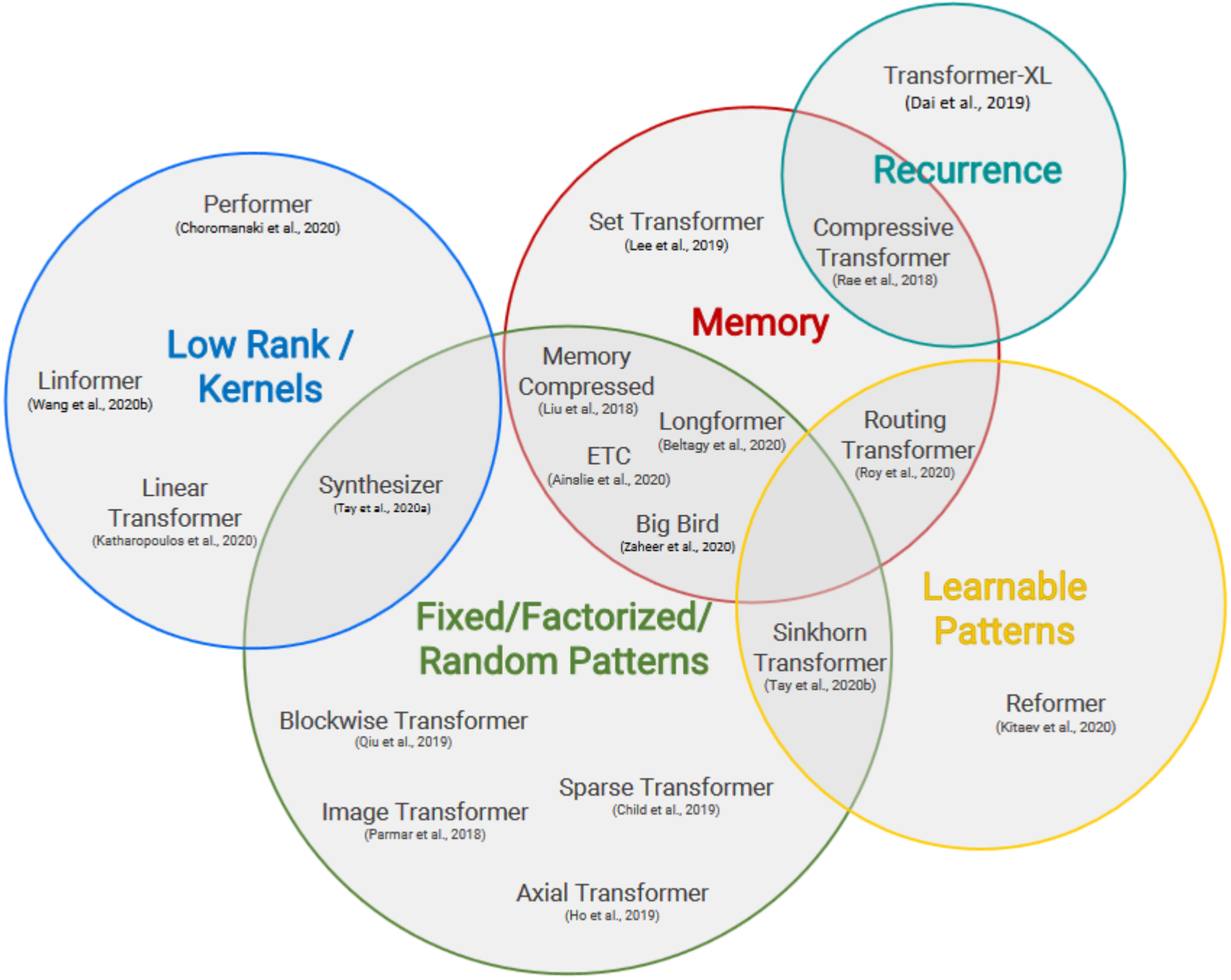


Figure 2: Taxonomy of Efficient Transformer Architectures.

Проблема трансформера

Есть задачи с большим входом

(саммаризация, ответ на вопрос по тексту, м.б. перевод)

а сложность внимания O (N^2)

решения – дальше

Приёмы, о которых поговорим

Распределение внимания

Fixed Patterns (FP) – внимание только в какой-то узкой зоне

Blockwise Patterns – делим на блоки, внимание внутри блока

Strided Patterns – разреженное внимание – через определённые интервалы

Compressed Patterns – пулинг для уменьшения длины последовательности

Learnable Patterns (LP) – учимся распределению

+ их комбинации!

Память

глобальная / локальная

Уменьшение числа параметров

Low-Rank Methods

+ кернализация (Kernels)

Позиционное кодирование

Absolute Positional Encoding / Sinusoidal Position Encoding
(Vaswani et al, 2017, https://arxiv.org/abs/1706.03762)
обычно используют – sin/cos

$$\alpha_{ij}^{Abs} = \frac{1}{\sqrt{d}}((w_i + p_i)W^{Q,1})((w_j + p_j)W^{K,1})^T$$

Learned Position Encoding (Gehring et al, 2017, https://arxiv.org/abs/1705.03122)
обучаем код позиции, но тут нет экстраполяции на большие номера

Relative Position Representations (Shaw et al, 2018, https://arxiv.org/abs/1803.02155)
считается лучшей

$$\alpha_{ij}^{Rel} = \frac{1}{\sqrt{d}}(x_i^l W^{Q,l})(x_j^l W^{K,l} + a_{j-i}^l)^T$$

Позиционное кодирование: Relative Position Representations

Сначала идея обучать a_{j-i} в каждом слое

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani «Self-attention with relative position representations» // https://arxiv.org/abs/1803.02155

$$\alpha_{ij}^{Rel} = \frac{1}{\sqrt{d}}(x_i^l W^{Q,l})(x_j^l W^{K,l} + a_{j-i}^l)^T$$

тут на самом деле учимся кодировать клиппированную разницу позиций

clip(j-i, –128, +128)

Model	Position Information	EN-DE BLEU	EN-FR BLEU
Transformer (base)	Absolute Position Representations	26.5	38.2
Transformer (base)	Relative Position Representations	**26.8**	**38.7**
Transformer (big)	Absolute Position Representations	27.9	41.2
Transformer (big)	Relative Position Representations	**29.2**	**41.5**

Table 1: Experimental results for WMT 2014 English-to-German (EN-DE) and English-to-French (EN-FR) translation tasks, using newstest2014 test set.

k	EN-DE BLEU
0	12.5
1	25.5
2	25.8
4	25.9
16	25.8
64	25.9
256	25.8

Table 2: Experimental results for varying the clipping distance, k.

Позиционное кодирование: Relative Position Representations

потом упрощение идеи: вынесено представление + общие параметры по слоям

Colin Raffel «Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer» // https://arxiv.org/abs/1910.10683

$$\alpha_{ij}^{T5} = \frac{1}{\sqrt{d}}(x_i^l W^{Q,l})(x_j^l W^{K,l})^T + b_{j-i}$$

формально нет в статье

Позиционное кодирование: Learned Position Encoding

в лекции по DeBERTa

Pengcheng He, et al. «DeBERTa: Decoding-enhanced BERT with Disentangled Attention» // https://arxiv.org/abs/2006.03654

Выучивается ли позиция?

Yu-An WangYun-Nung Chen «What Do Position Embeddings Learn? An Empirical Study of Pre-Trained Language Model Positional Encoding» // https://arxiv.org/pdf/2010.04903.pdf

Поскольку PE: N → X

попробуем посмотреть, можем ли мы реализовать простую (линейную) обратную функцию? более сложные функции переобучались

т.е. по вложению предсказать номер (абсолютную позицию) / разницу (относительную)

вторая таблица – ошибка предсказания правильный порядок позиций или нет

Type	PE	MAE
Learned	BERT	34.14
	RoBERTa	6.06
	GPT-2	1.03
Pre-Defined	sinusoid	0.0

Table 1: Mean absolute error of the reversed mapping function learned by linear regression.

Type	PE	Error Rate
Learned	BERT	19.72%
	RoBERTa	7.23%
	GPT-2	1.56%
Pre-Defined	sinusoid	5.08%

Table 2: Error rate of the relative position regression.

Выучивается ли позиция?

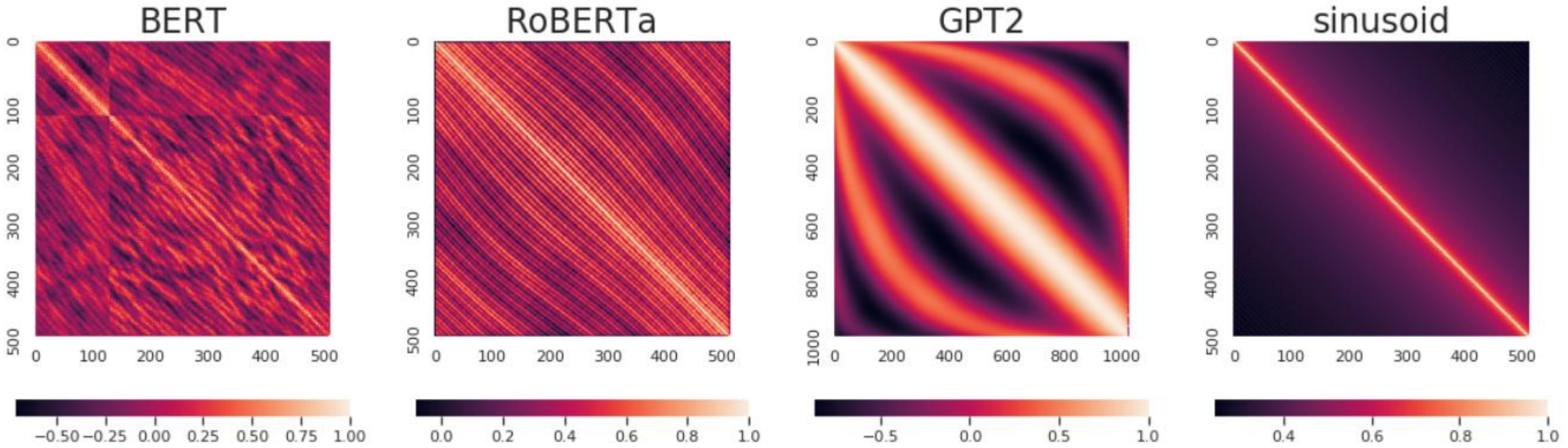


Figure 1: Visualization of position-wise cosine similarity of different position embeddings. Lighter in the figures denotes the higher similarity.

у Берта есть скачок, т.е. сначала обучение на предложениях длины ≤128

Выучивается ли позиция?

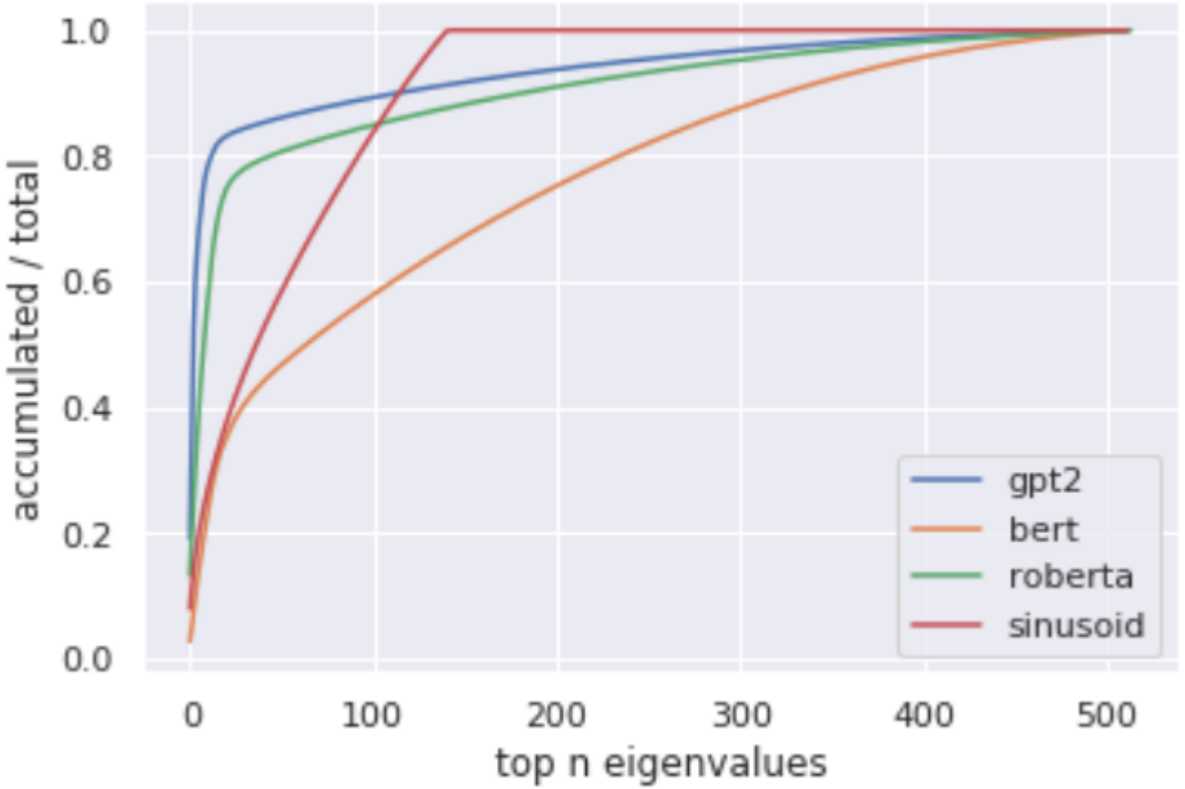


Figure 2: Accumulated top eigenvalues of position embeddings.

Посмотрели на SVD-разложение

Интересные выводы: трансформер-кодировщик лучше выучивает относительную позицию, декодировщик – абсолютную, многое зависит от задачи

Transformer with Untied Positional Encoding (TUPE)

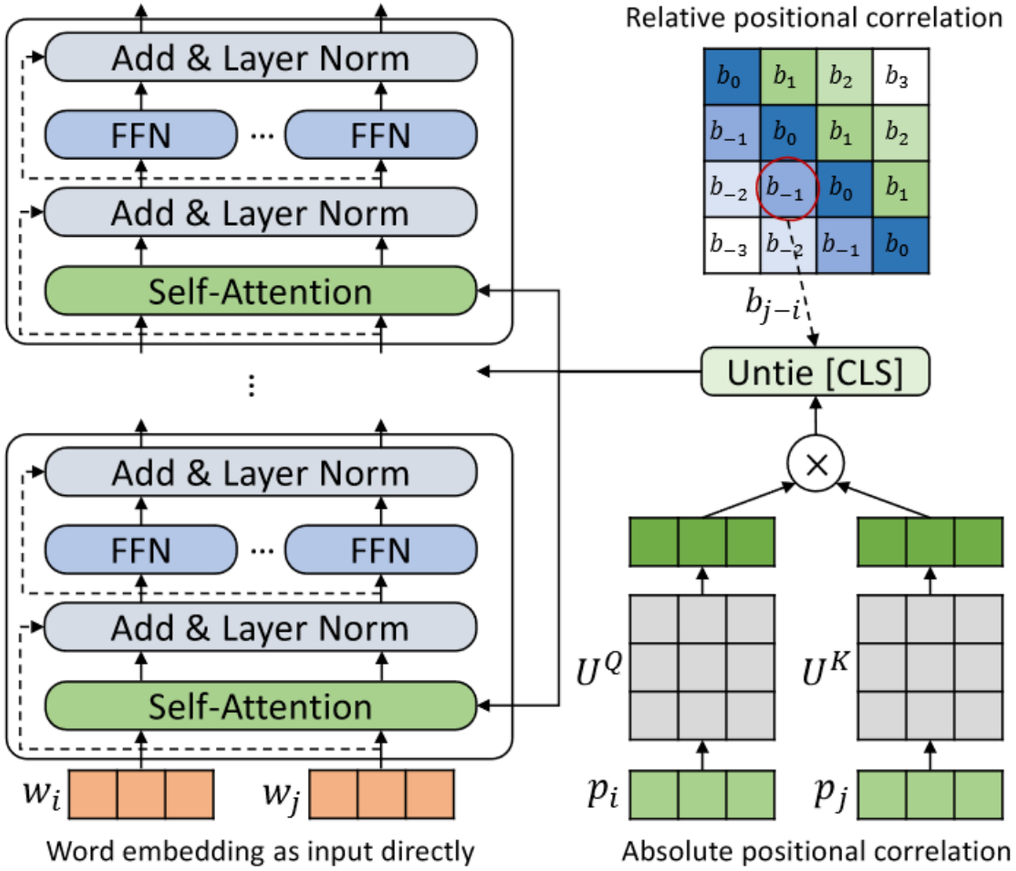


Figure 1: The architecture of TUPE.

Guolin Ke, Di He & Tie-Yan Liu «Rethinking positional encoding in language pre-training»

https://arxiv.org/pdf/2006.15595.pdf

TUPE: анализ близости

$$\alpha_{ij}^{Abs} = \frac{((w_i + p_i)W^{Q,1})((w_j + p_j)W^{K,1})^T}{\sqrt{d}}$$
$$= \frac{(w_i W^{Q,1})(w_j W^{K,1})^T}{\sqrt{d}} + \frac{(w_i W^{Q,1})(p_j W^{K,1})^T}{\sqrt{d}}$$
$$+ \frac{(p_i W^{Q,1})(w_j W^{K,1})^T}{\sqrt{d}} + \frac{(p_i W^{Q,1})(p_j W^{K,1})^T}{\sqrt{d}} \quad (6)$$

Если раскрыть скобки – 4 вида схожести:

word-to-word, word-to-position, position-to-word, position-to-position

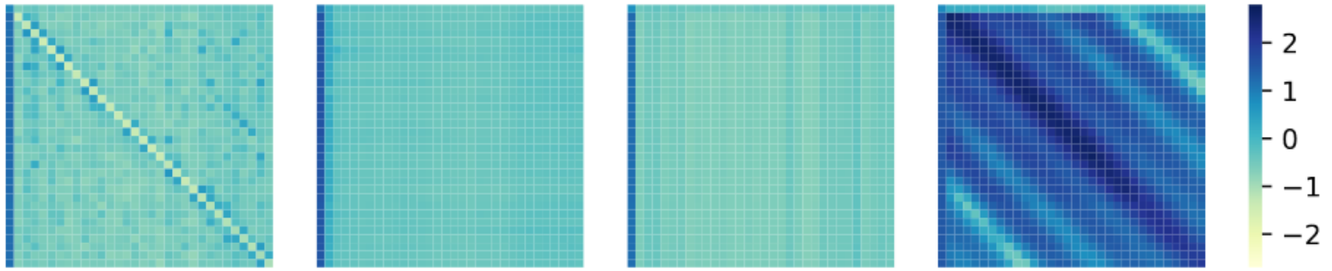


Figure 2: Visualizations of the four correlations (Eq. (6)) on a pretrained BERT model for a sampled batch of sentences. From left to right: word-to-word, word-to-position, position-to-word, and position-to-position correlation matrices. In each matrix, the (i-th, j-th) element is the correlation between i-th word/position and j-th word/position. We can find that the correlations between a word and a position are not strong as the values in the second and third matrices look uniform.

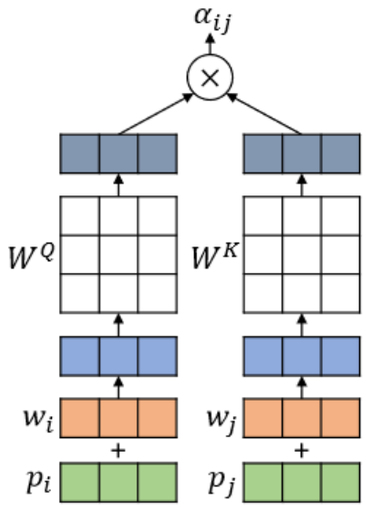
Показали, что нет большой корреляции между словом и его позицией

TUPE: основные модификации

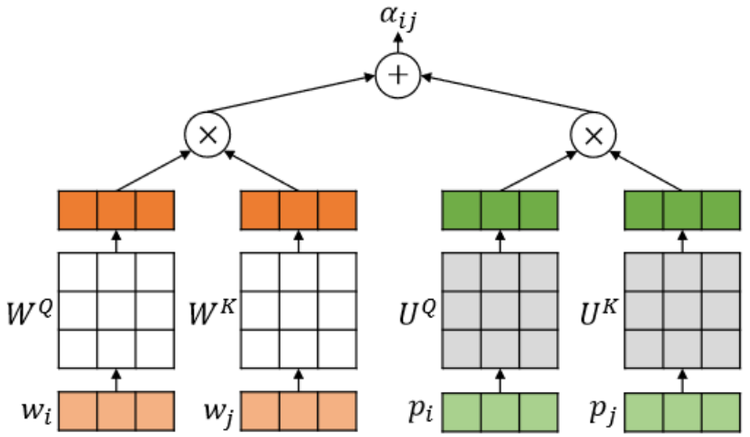
Убрать 2 слагаемых из той формулы, а оставшиеся изменить:

$$\alpha_{ij} = \frac{1}{\sqrt{2d}}(x_i^l W^{Q,l})(x_j^l W^{K,l})^T + \frac{1}{\sqrt{2d}}(p_i U^Q)(p_j U^K)^T + b_{j-i}.$$

– отношения между словами и позициями



(a) Absolute positional encoding.



(b) Untied absolute positional encoding.

Figure 3: Instead of adding the absolute positional embedding to the word embedding in the input (left), we compute the positional correlation and word correlation separately with different projection matrices, and add them together in the self-attention module (right).

TUPE: основные модификации

Особое кодирование для [CLS] –

у него отдельная роль и нелогично кодировать как другие токены

$$\text{reset}_\theta(v,i,j) = \begin{cases} v_{ij} & i \neq 1, j \neq 1, (\text{no } \texttt{[CLS]}) \\ \theta_1 & i = 1, (\text{from } \texttt{[CLS]} \text{ to others}) \\ \theta_2 & i \neq 1, j = 1, (\text{from others to } \texttt{[CLS]}) \end{cases}$$

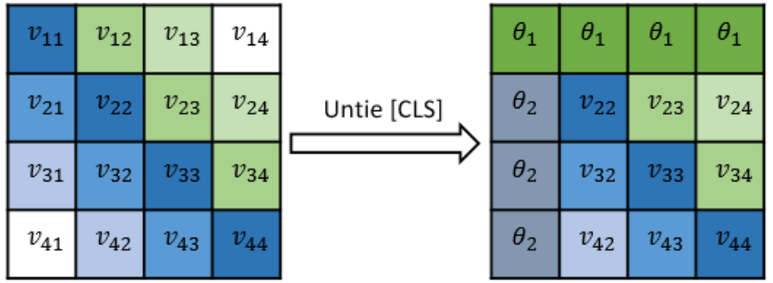


Figure 4: Illustration of untying [CLS]. v_{ij} denotes the positional correlation of pair (i, j). The first row and first column are set to the same values respectively.

Transformer-XL

Модификация LM на обычном трансформере (декодере)

Последовательность делится на фрагменты одинаковой длины, каждый фрагмент обрабатывается в свой момент времени, каждый слой может смотреть и в предыдущий фрагмент → можно выучивать более длинные зависимости.

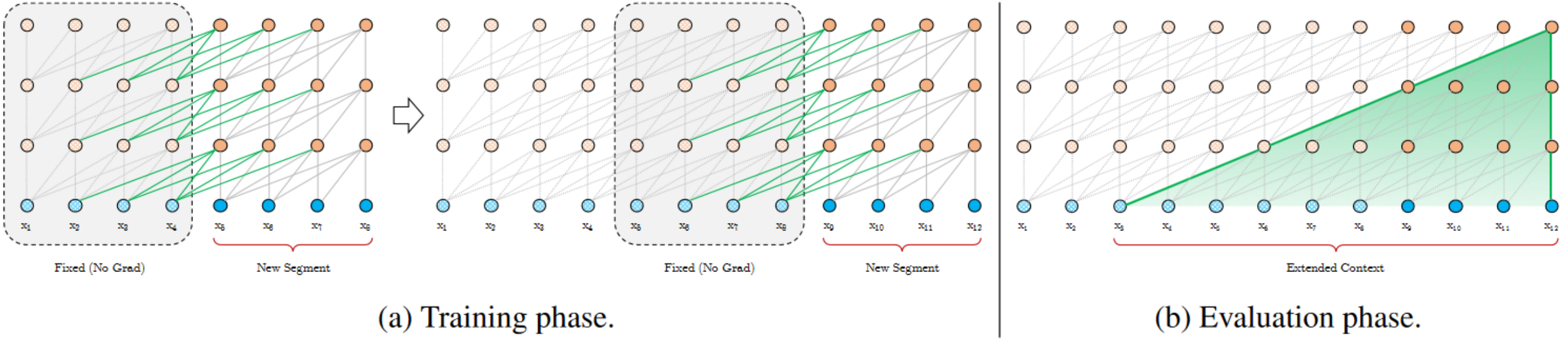


Figure 2: Illustration of the Transformer-XL model with a segment length 4.

Dai et al «Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context» https://arxiv.org/abs/1901.02860

Transformer-XL

тут как раз Relative Position Representations, т.к. из-за большой длины последовательности можем встречать незнакомые коды позиций

Раньше

$$\mathbf{A}^{\text{abs}}_{i,j} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(b)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(c)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(d)}.$$

Сейчас

$$\mathbf{A}^{\text{rel}}_{i,j} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(b)} + \underbrace{u^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(c)} + \underbrace{v^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(d)}.$$

Красные – обучаемые параметры

Синие – относительные представления

Compressive Transformer (продолжение идеи Transformer-XL)

На рис. идём по сегментам по 3, что прошли – складываем в память, что совсем старое «сохраняем» специальной функцией в долгой памяти

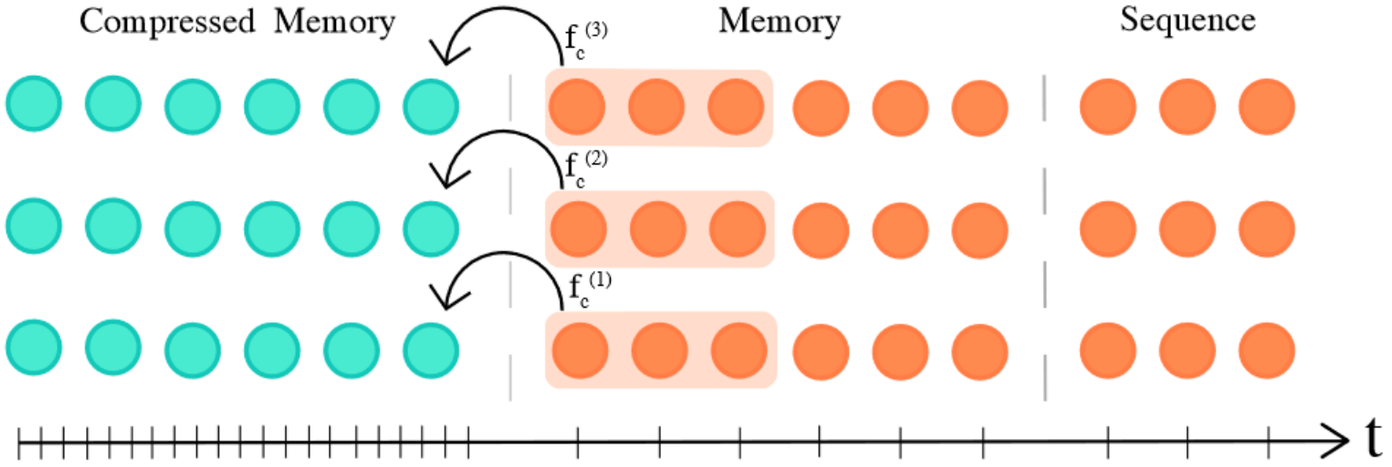


Figure 1: The Compressive Transformer keeps a fine-grained memory of past activations, which are then compressed into coarser *compressed* memories. The above model has three layers, a sequence length $n_s = 3$, memory size $n_m = 6$, compressed memory size $n_{cm} = 6$. The highlighted memories are compacted, with a compression function f_c per layer, to a single compressed memory — instead of being discarded at the next sequence. In this example, the rate of compression $c = 3$.

Jack W. Rae «Compressive Transformers for Long-Range Sequence Modelling»

https://arxiv.org/abs/1911.05507

Compressive Transformer

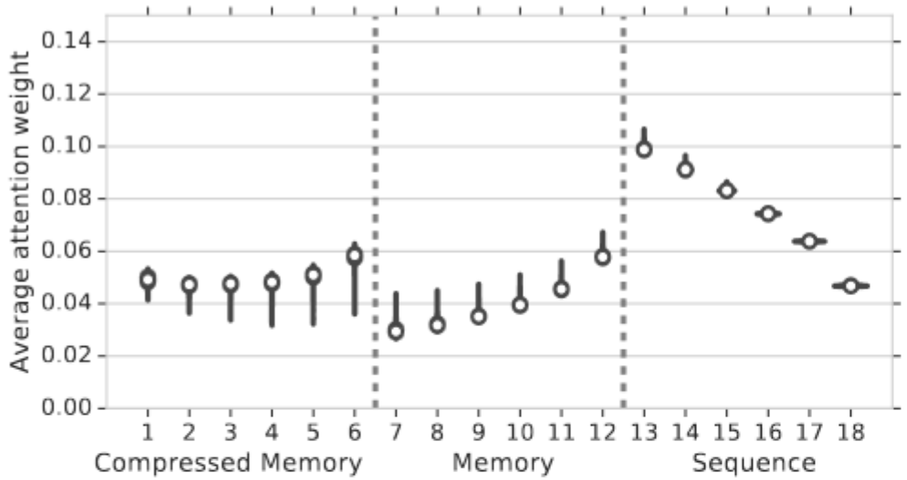


Figure 2: **Attention weight on Enwik8**. Average attention weight from the sequence over the compressed memory (oldest), memory, and sequence (newest) respectively. The sequence self-attention is causally masked, so more attention is placed on earlier elements in the sequence. There is an increase in attention at the transition from memory to compressed memory.

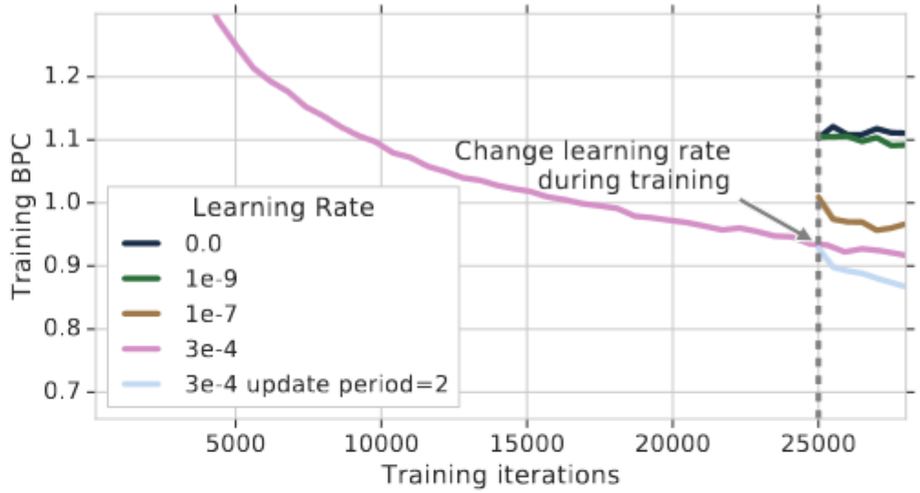


Figure 3: **Learning rate analysis**. Reducing the learning rate (e.g. to zero) during training (on Enwik8) harms training performance. Reducing the frequency of optimisation updates (effectively increasing the batch size) is preferable.

Universal Transformer

Для вычислительной универсальности (полнота по Тьюрингу)

Обычный трансформер не очень хорошо выполнял простые задачи типа «копировать последовательность», но здорово работал в машинном переводе

Число операций не зависит от входа

ОТ плох когда длины последовательностей отличаются в обучении и тесте

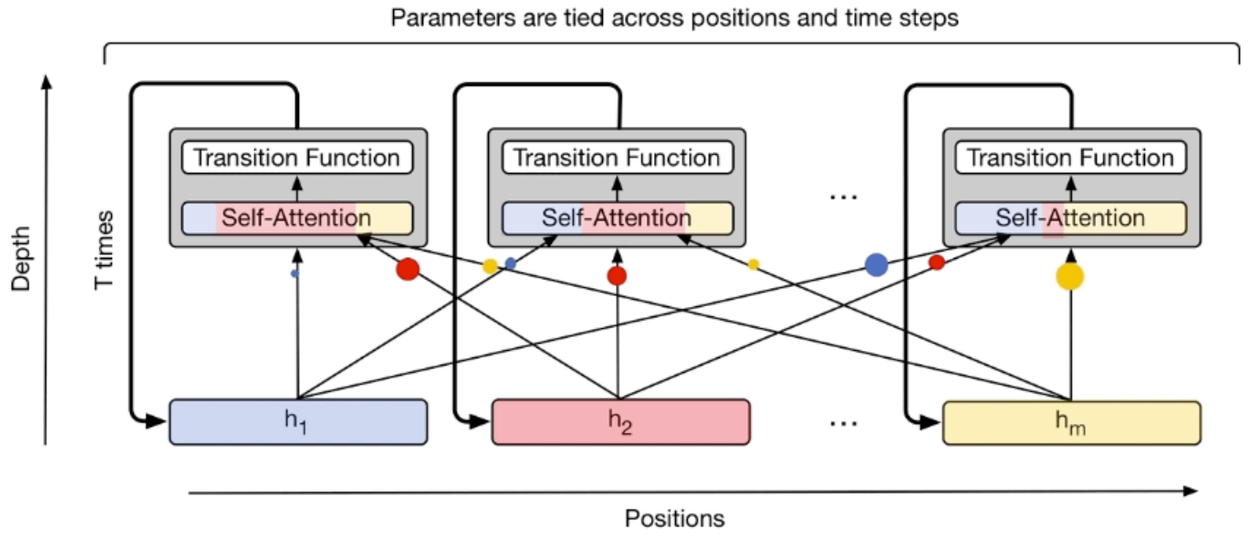
Moving Beyond Translation with the Universal Transformer

https://ai.googleblog.com/2018/08/moving-beyond-translation-with.html

https://mostafadehghani.com/2019/05/05/universal-transformers/

«Universal Transformers» // https://arxiv.org/abs/1807.03819

Universal Transformer: стекинг блоков трансформера

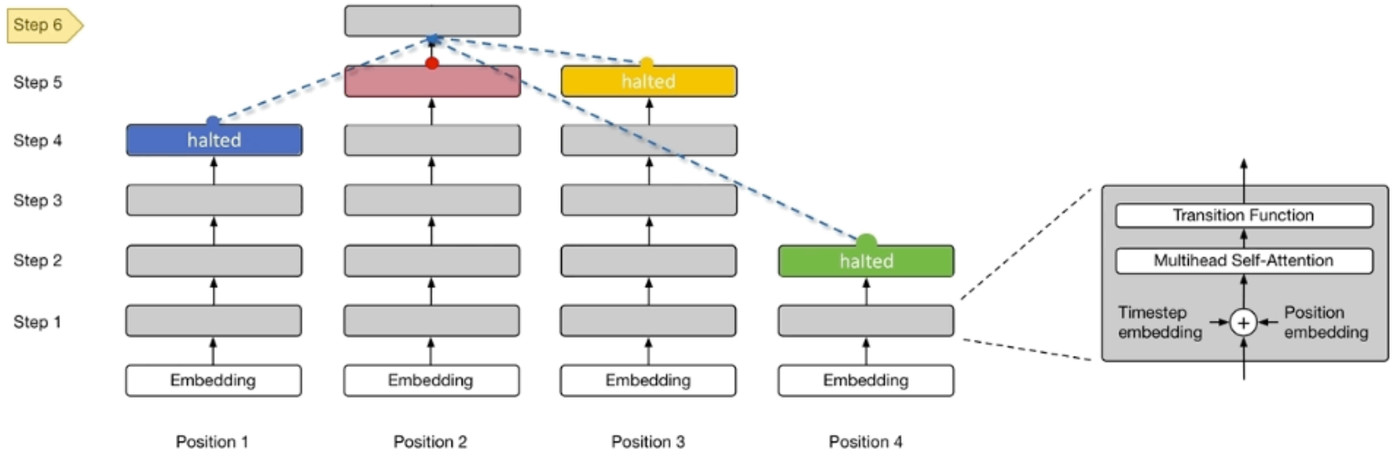


Сохранили параллельный трансформер,
но сколько слоёв (сколько раз его рекуррентно применять) не фиксируем
(может определяться им)

«рекуррентен в глубину»

Пример: I arrived at the bank after crossing the river (bank – требует большего контекста)

Universal Transformer: стекинг блоков трансформера



ACT (adaptive computation time) – механизм определения глубины

ранее предложен для RNN

Universal Transformer: стекинг блоков трансформера

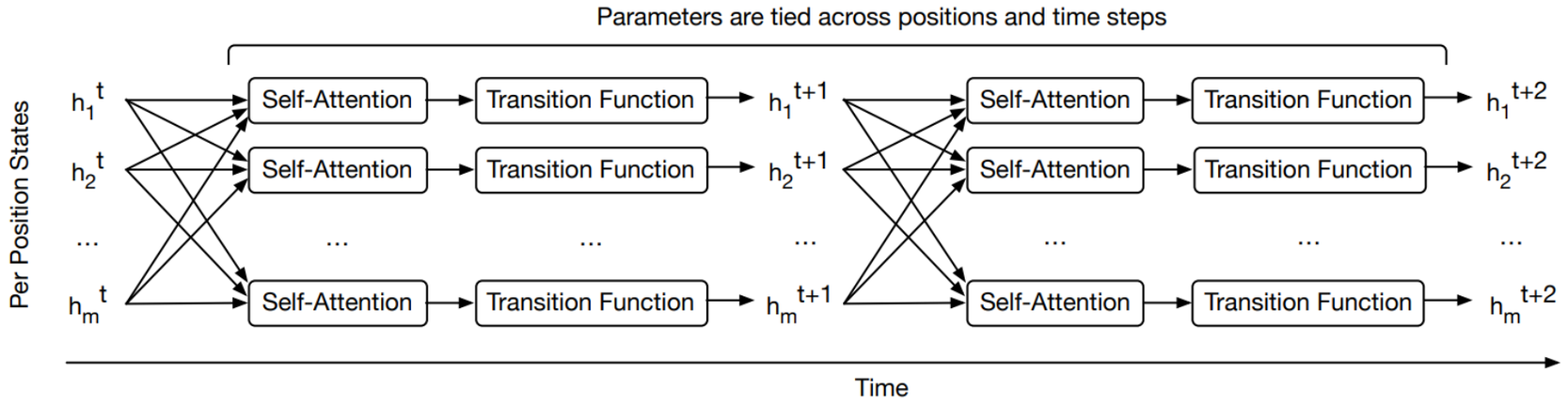


Figure 1: The Universal Transformer repeatedly refines a series of vector representations for each position of the sequence in parallel, by combining information from different positions using self-attention and applying a recurrent transition function. We show this process over two recurrent time-steps. Arrows denote dependencies between operations. Initially, h^0 is initialized with the embedding for each symbol in the sequence. h_i^t represents the representation for input symbol $1 \leq i \leq m$ at recurrent time-step t.

Universal Transformer: стекинг блоков трансформера

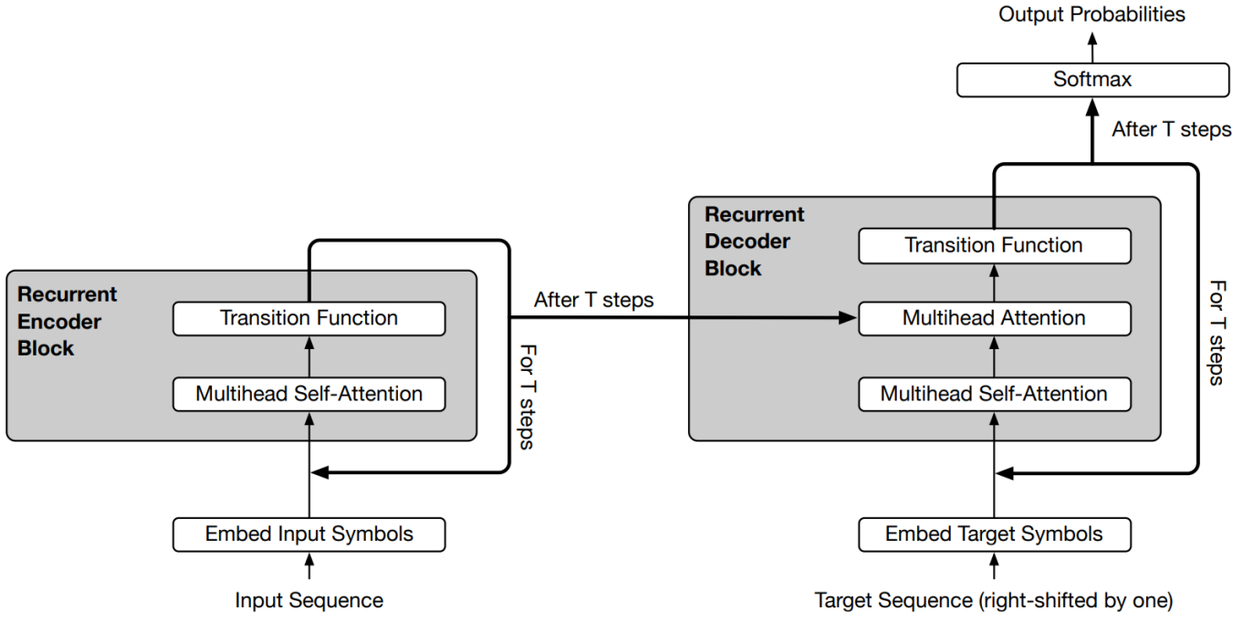


Figure 2: The recurrent blocks of the Universal Transformer encoder and decoder. This diagram omits position and time-step encodings as well as dropout, residual connections and layer normalization. A complete version can be found in the appendix. The Adaptive Universal Transformer dynamically determines the number of steps T for each position using ACT.

Adaptive Attention Span

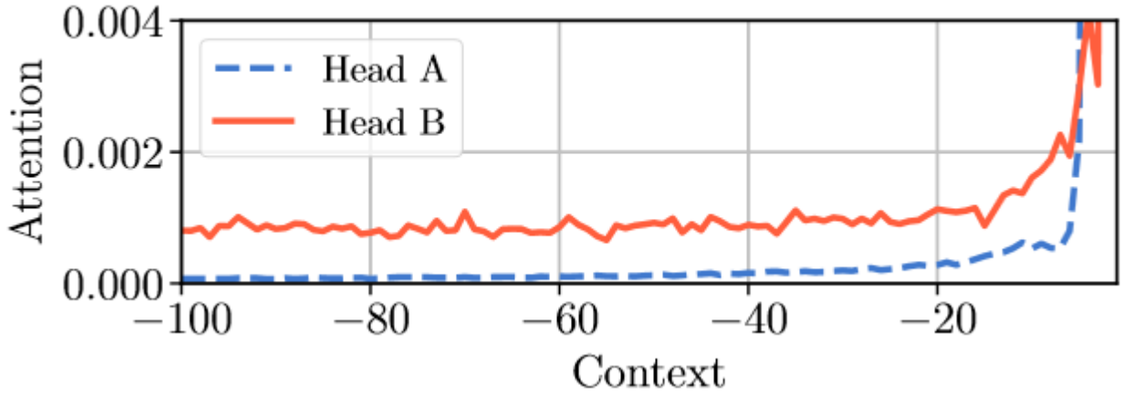


Figure 1: Attention patterns of two different heads of a standard Transformer. The two patterns are qualitatively different: Head A utilizes recent steps, while Head B has uniform attention over the context.

у разных головок внимание распределено по-разному…

будем определять нужный размер контекста

Learnable – обучается, но фиксируется после обучения

ACT-like – изменяется динамически в зависимости от входа

«Adaptive Attention Span in Transformers» // https://arxiv.org/abs/1905.07799

Adaptive Attention Span

Функция маски для внимания:

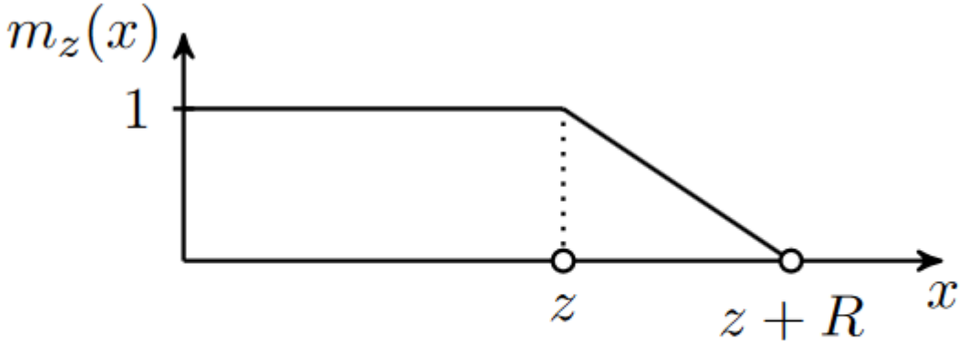


Figure 2: The soft mask as a function of the distance.

$$a_{tr} = \frac{m_z(t-r)\exp(s_{tr})}{\sum_{q=t-S}^{t-1} m_z(t-q)\exp(s_{tq})}.$$

We add a ℓ_1 penalization on the parameters z_i for each attention head i of the model to the loss function:

$$L = -\log P(w_1,\ldots,w_T) + \frac{\lambda}{M}\sum_i z_i,$$

Регуляризация на параметр z

Также Dynamic attention span – z результат действия маленькой сети, но регуляризация остаётся

Adaptive Attention Span

Model	#layers	Avg. span	#Params	#FLOPS	dev	test
Small models						
T12 (Al-Rfou et al., 2019)	12	512	44M	22G	-	1.18
Adaptive-Span ($S = 8192$)	12	314	38M	42M	1.05	**1.11**
Large models						
T64 (Al-Rfou et al., 2019)	64	512	235M	120G	1.06	1.13
T-XL (Dai et al., 2019)	24	3800	277M	438M	-	1.08
Adaptive-Span ($S = 8192$)	24	245	209M	179M	1.01	**1.07**

Table 1: Character level language modeling on `text8`. We report bpc for the dev and test sets, as well as, the number of parameters, the average attention spans and total number of FLOPS (an estimate of the number of FLOPS necessary for computing one step prediction).

Adaptive Attention Span

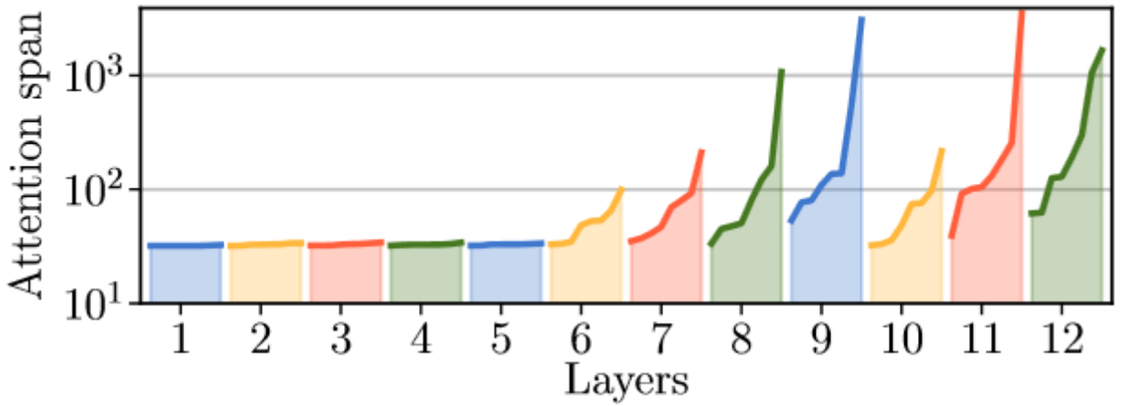


Figure 4: Adaptive spans (in log-scale) of every attention heads in a 12-layer model with span limit $S = 4096$. Few attention heads require long attention spans.

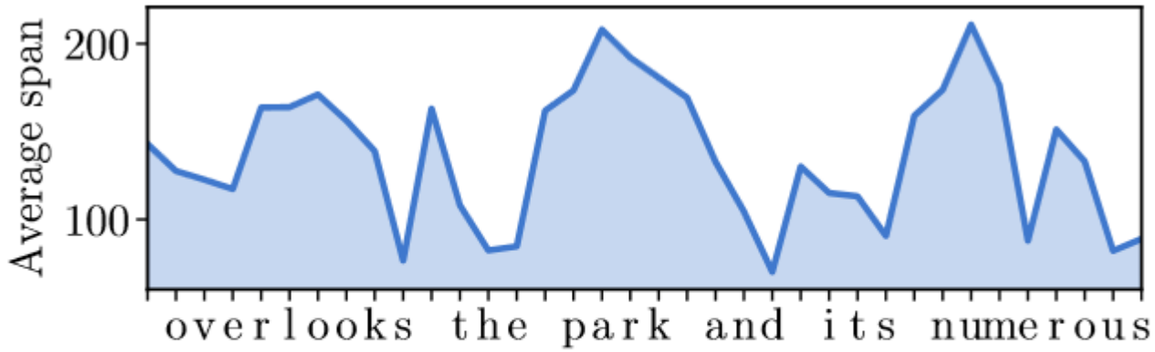


Figure 5: Example of average dynamic attention span as a function of the input sequence. The span is averaged over the layers and heads.

Малому число головок нужно большое внимание

Expire-Span Transformer (продолжение идеи Adaptive Attention Span – те же авторы)

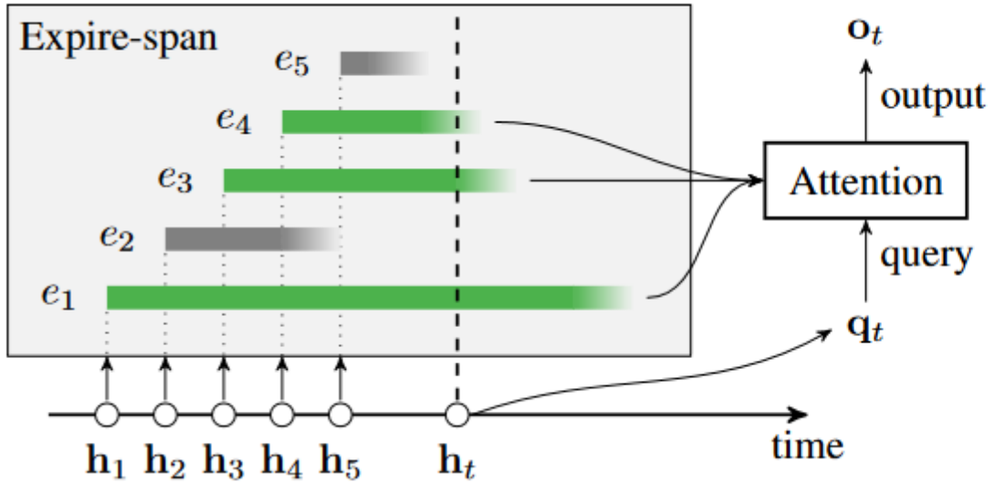


Figure 1. **Expire-Span**. For every memory $\mathbf{h}_i$, we compute an EXPIRE-SPAN e_i that determines how long it should stay in memory. Here, memories $\mathbf{h}_2$, $\mathbf{h}_5$ are already expired at time t, so the query $\mathbf{q}_t$ can only access $\{\mathbf{h}_1, \mathbf{h}_3, \mathbf{h}_4\}$ in self-attention.

метод «Expire-Span»

Для каждого токена по состоянию считаем время «expiration value»,

сколько его помнить: $e_i = L\sigma(\mathbf{w}^\top \mathbf{h}_i + b)$

каждый такт это время уменьшается на 1 (пока не станет ≤0)

внимание считаем по токенам, которые не забылись

Sainbayar Sukhbaatar, et al. «Not All Memories are Created Equal: Learning to Forget by Expiring» // https://arxiv.org/abs/2105.06548

Expire-Span Transformer

Внимание считается с маскированием

$$a'_{ti} = \frac{m_{ti} a_{ti}}{\sum_j m_{tj} a_{tj}}, \quad \mathbf{o}_t = \sum_i a'_{ti} \mathbf{v}_i$$

Но маска не бинарная, чтобы проникал градиент и время памяти могло увеличиться

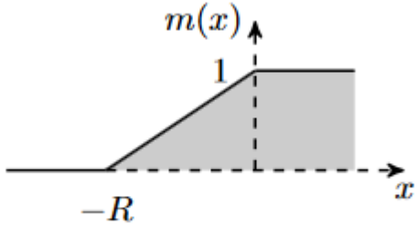


Figure 2. Soft Mask

В ошибку + L1-слагаемое для уменьшения среднего времени памяти:

$$L_{\text{total}} = L_{\text{task}} + \alpha \sum_i e_i / T$$

Эта техника применяется в каждом слое независимо

В каждом слое у каждого токена своё время памяти

Expire-Span Transformer

Есть тонкости эффективной реализации (см. статью)

Пример дополнительной регуляризации:
для каждого батча выбирается случайная длина $l \in$ [0, L]
обнуляем $a_{ij}=0$ при $j-i > l$
(даём понять, что нет гарантии, что в далёких токенах содержится нужная информация)

Expire-Span Transformer: reinforcement learning tasks

Figure 3. **Corridor Task (left)**- Agents must memorize the color of an object and walk through the door of the corresponding color at the end of a long corridor. **Portal Task (middle)**- An agent must trial-and-error to memorize the sequence of doors. **Instruction Task (right)**- A model must recognize instructions, memorize them, and execute when at the correct location.

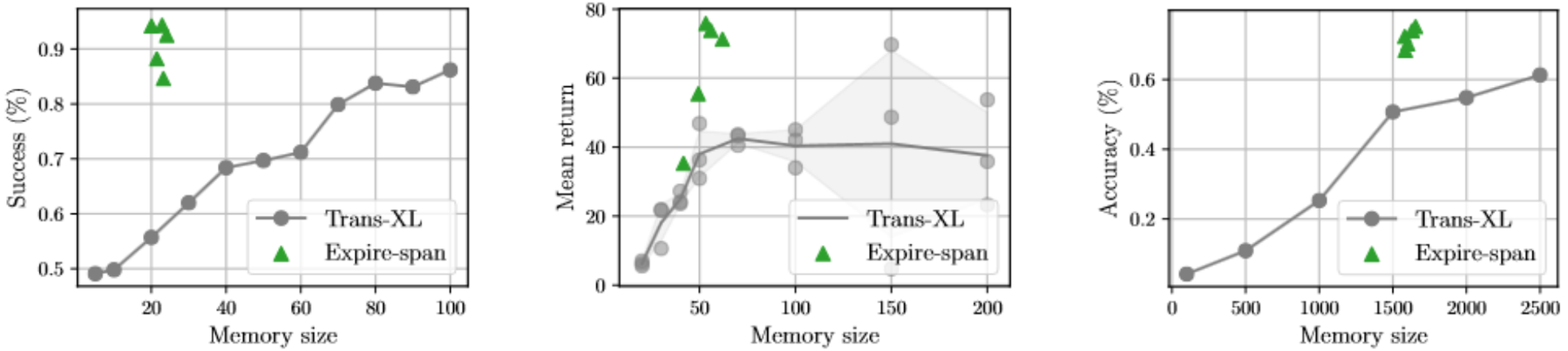


Figure 4. We plot performance as a function of memory size for three tasks. Training scores are shown. Ideal models can achieve strong performance with small memories by identifying which information is important to remember. **Corridor Task (left)** — We train 10 baseline models with different memory sizes, and five EXPIRE-SPAN models with different seeds. **Portal Task (middle)**- We train models with different memory sizes and random seeds. **Instruction Task (right)** — We train 6 baseline models with different memory sizes, and five EXPIRE-SPAN models with different seeds.

Expire-Span Transformer

Model	Maximum span	Accuracy (%)
Transformer-XL	2k	26.7
EXPIRE-SPAN	16k	29.4
EXPIRE-SPAN	128k	**52.1**

Table 1. **Copy Task.** We report accuracy on the test set.

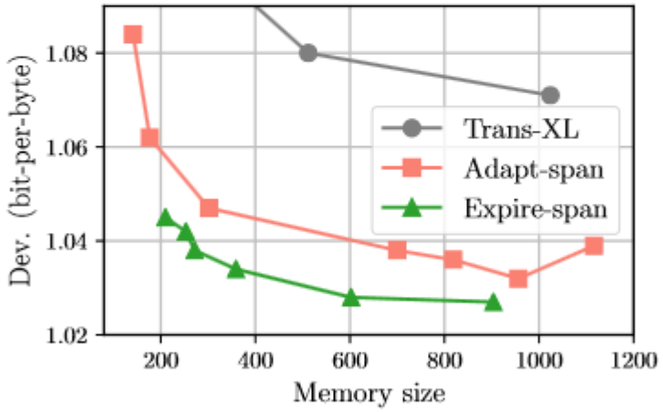


Figure 5. **Performance as a Function of Memory Size on Enwik8.** Lower bpb and smaller memory size is better.

Model	Params	Test
Small models		
Trans-XL 12L (Dai et al., 2019)	41M	1.06
Adapt-Span 12L (Sukhbaatar et al., 2019a)	39M	1.02
Our Trans-XL 12L baseline	38M	1.06
EXPIRE-SPAN 12L	38M	**0.99**
Trans-XL 24L (Dai et al., 2019)	277M	0.99
Sparse Trans. (Child et al., 2019)	95M	0.99
Adapt-Span 24L (Sukhbaatar et al., 2019a)	209M	0.98
All-Attention (Sukhbaatar et al., 2019b)	114M	0.98
Compressive Trans. (Rae et al., 2020)	277M	0.97
Routing Trans. (Roy et al., 2020)	-	0.99
Feedback Trans. (Fan et al., 2020b)	77M	0.96
EXPIRE-SPAN 24L	208M	**0.95**

Table 2. **Enwik8 Results.** We report bit-per-byte (bpb) on test and the number of parameters.

attention span доходит до 128K токенов!!!

Expire-Span Transformer

Figure 8. **Expiration in EXPIRE-SPAN on Enwik8.** In **(a)**, the model strongly memorizes two areas, "Egypt" and "Alexander". In **(b)**, if we replace "Egypt" with "somewhere", then it's forgotten fast. In **(c)**, we insert "Humpty Dumpty" and the model retains these rare words in memory.

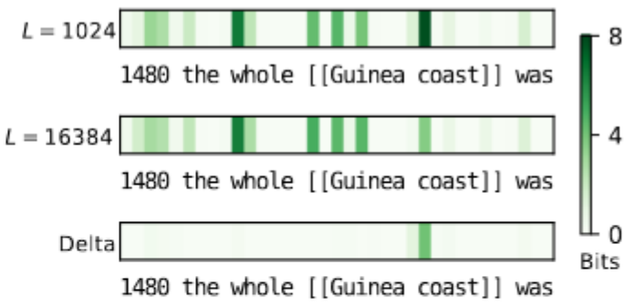


Figure 9. **Accuracy Needs Memory.** As the maximum span is artificially decreased at inference time from 16k to only 1k, the prediction is less accurate.

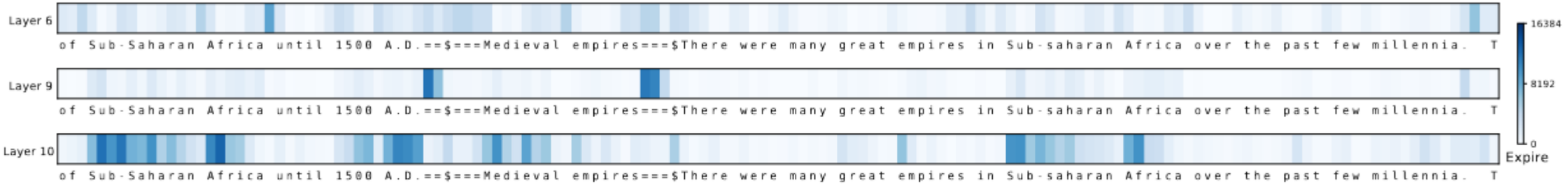


Figure 11. **Per-Layer EXPIRE-SPAN values on Enwik8.** We visualize the expire-spans of different layers: layer 6 gives long span to spaces, layer 9 memorizes special tokens like newlines and section titles, and layer 10 retains named entities in memory.

разные слои смотрят разное (именованные сущности, пробелы, заголовки)

Memory Transformer

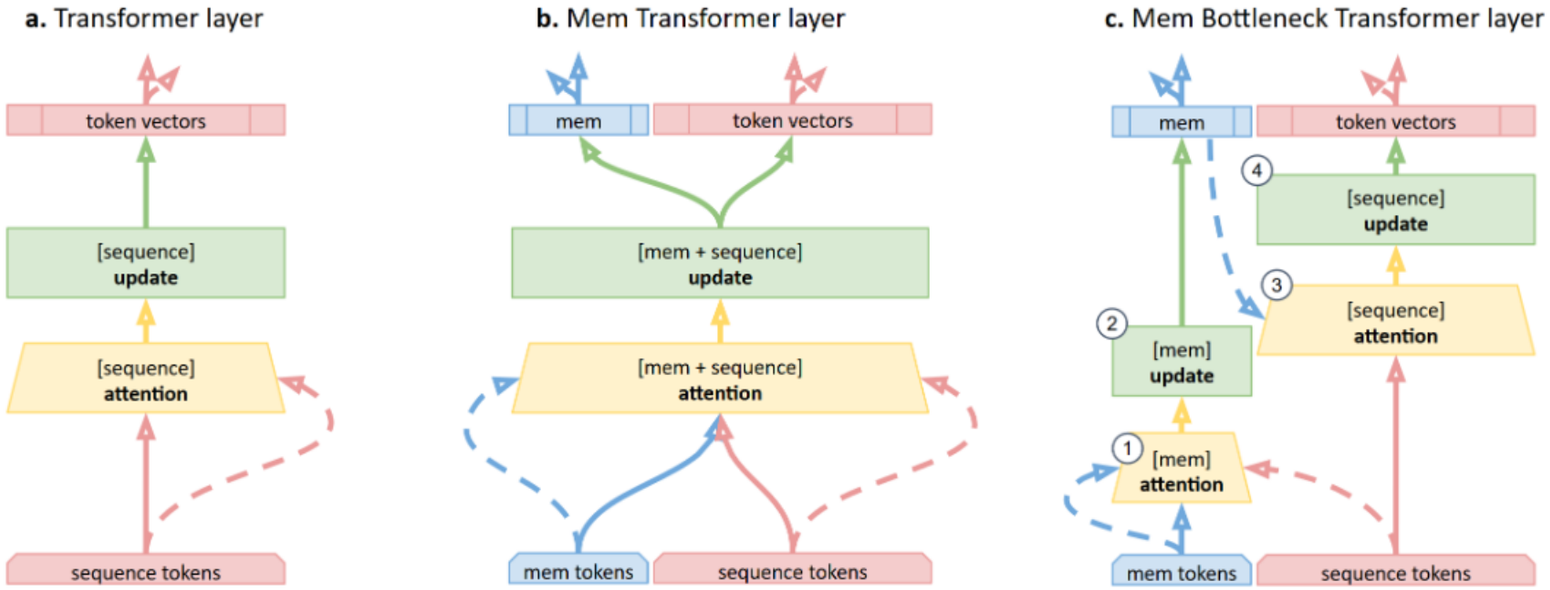


Figure 1: **Memory modifications of Transformer architecture.** (a) *Transformer layer*. For every element of a sequence (solid arrow), self-attention produces aggregate representation from all other elements (dashed arrow). Then this aggregate and the element representations are combined and updated with a fully-connected feed-forward network layer. (b) *Memory Transformer* (MemTransformer) prepends input sequence with dedicated `[mem]` tokens. This extended sequence is processed with a standard Transformer layer without any distinction between `[mem]` and other elements of the input. (c) *Memory Bottleneck Transformer* (MemBottleneck Transformer) uses `[mem]` tokens but separates memory and input attention streams. At the first step, representations of `[mem]` tokens are updated (2) with the attention span (1) covering both memory and input segments of the sequence. Then representations of input elements are updated (4) with memory attention (3) only. Thus information flow is distributed to representations of elements only through the memory.

Mikhail S. Burtsev, et al. «Memory Transformer» https://arxiv.org/abs/2006.11527

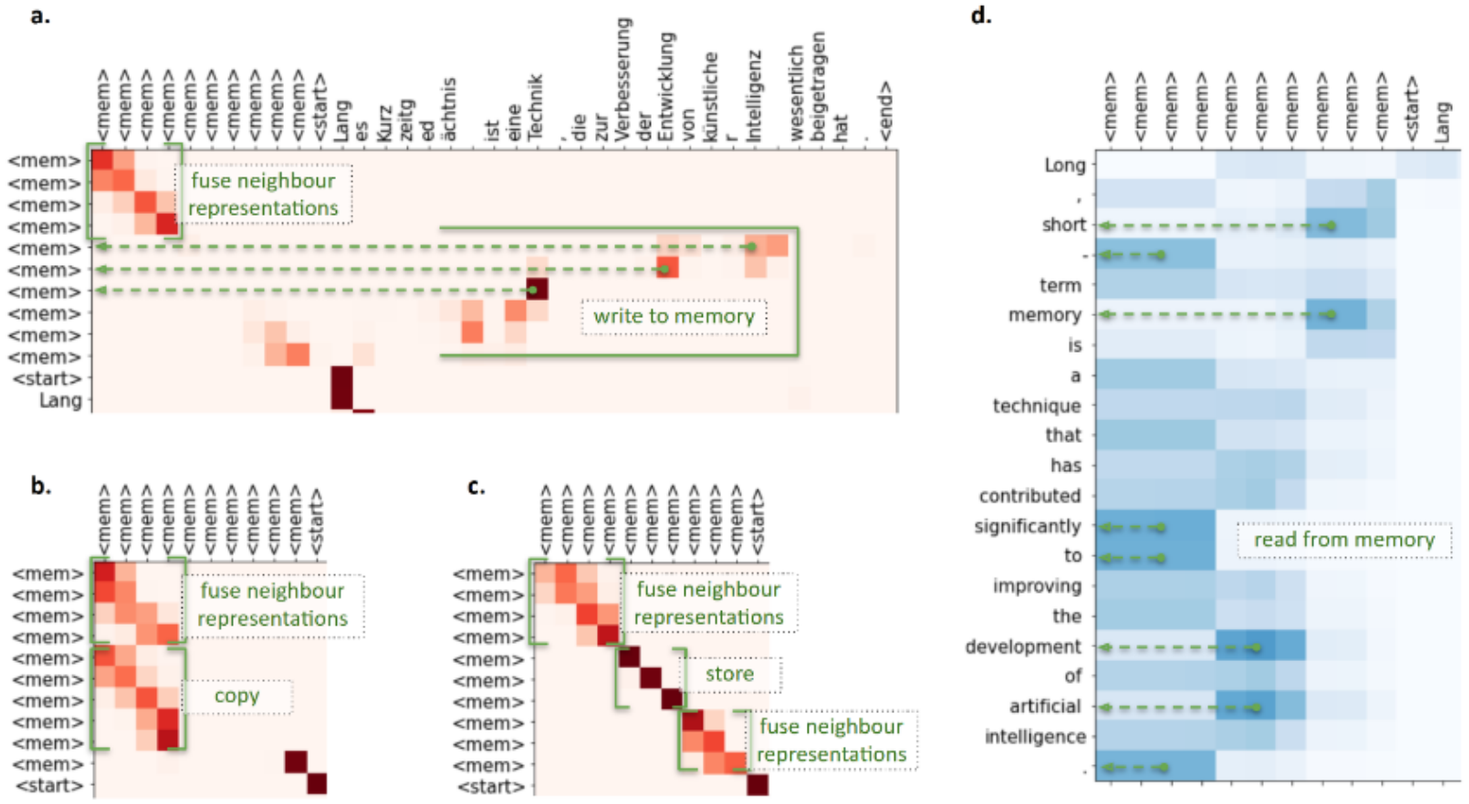


Figure 2: **Operations with memory learned by MemTransformer 10.** (a) The pattern of self-attention in the 3^{rd} encoder layer. Here, `[mem]` tokens in the central segment of memory (on the left) attend to the vector representations of tokens `Technik`, `Entwicklung`, `Intelligenz` (and some others). This attention values are consistent with the *writing* of selected token vectors to the `[mem]` tokens. Activity in the left top corner that involves first four tokens might indicate *fusion* of neighbour vectors by pairwise summation of `[mem]` tokens. (b) In the next 4^{th} layer of the same encoder similar fusion operation with the same `[mem]`'s is repeated. A parallel diagonal activity just below the fusion pattern can be attributed to *copy* operation. (c) Another attention head in the same encoder layer demonstrates combination of fusion and *store* operations. Sharp self-attention of three tokens in the middle results in adding vectors to themselves. (d) Attention pattern in decoder layer 4 over the output of 6^{th} encoder layer suggest that vectors of `[mem]` tokens are selectively *read* and added to the output token representations during decoding.

Memory Transformer

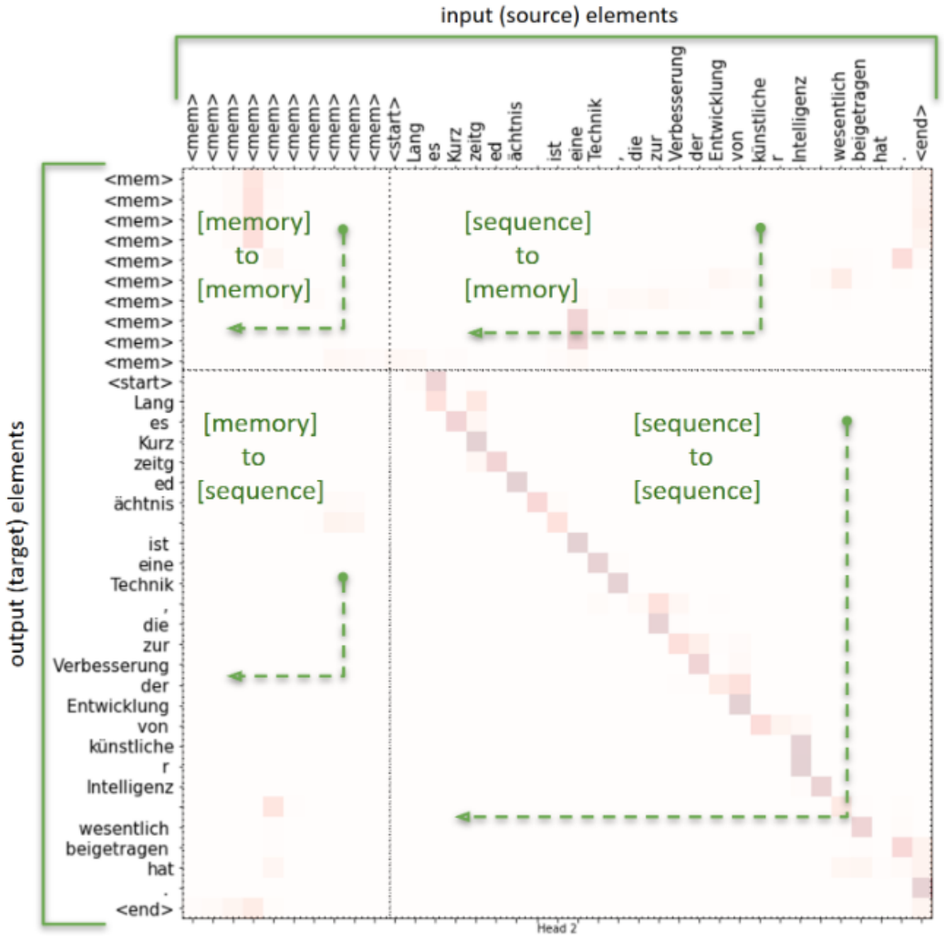


Figure 5: **How to read Memory Transformer attention map.** Attention values indicate how elements of input sequence (on the top) contribute to the update of representation for specific output element (on the left). Attention map for memory augmented transformer can be split into four blocks: (1) *update* - [sequence] to [sequence]; (2) *write* - [sequence] to [memory]; (3) *read* - [memory] to [sequence]; (4) *process* - [memory] to [memory].

Star-Transformer

Разреженное внимание – соединение с соседями и выделенной вершиной

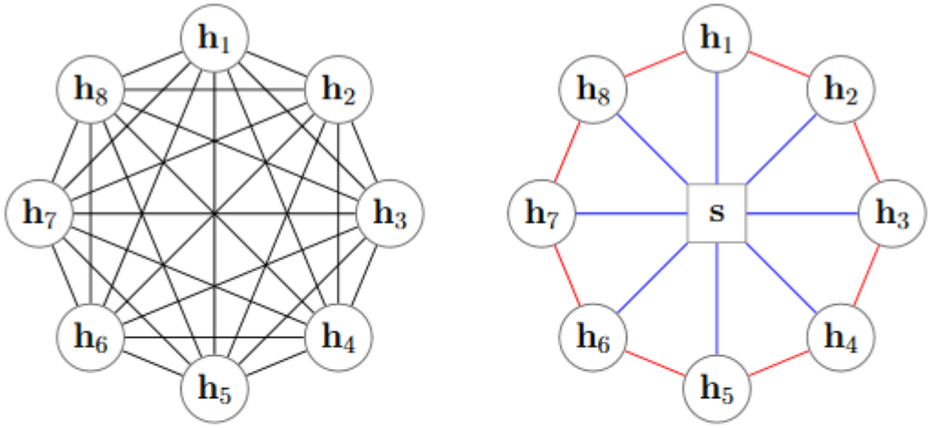


Figure 1: Left: Connections of one layer in Transformer, circle nodes indicate the hidden states of input tokens. Right: Connections of one layer in Star-Transformer, the square node is the virtual relay node. Red edges and blue edges are ring and radical connections, respectively.

Qipeng Guo et al. «Star-Transformer» // https://arxiv.org/pdf/1902.09113.pdf

Extended Transformer Construction (ETC)

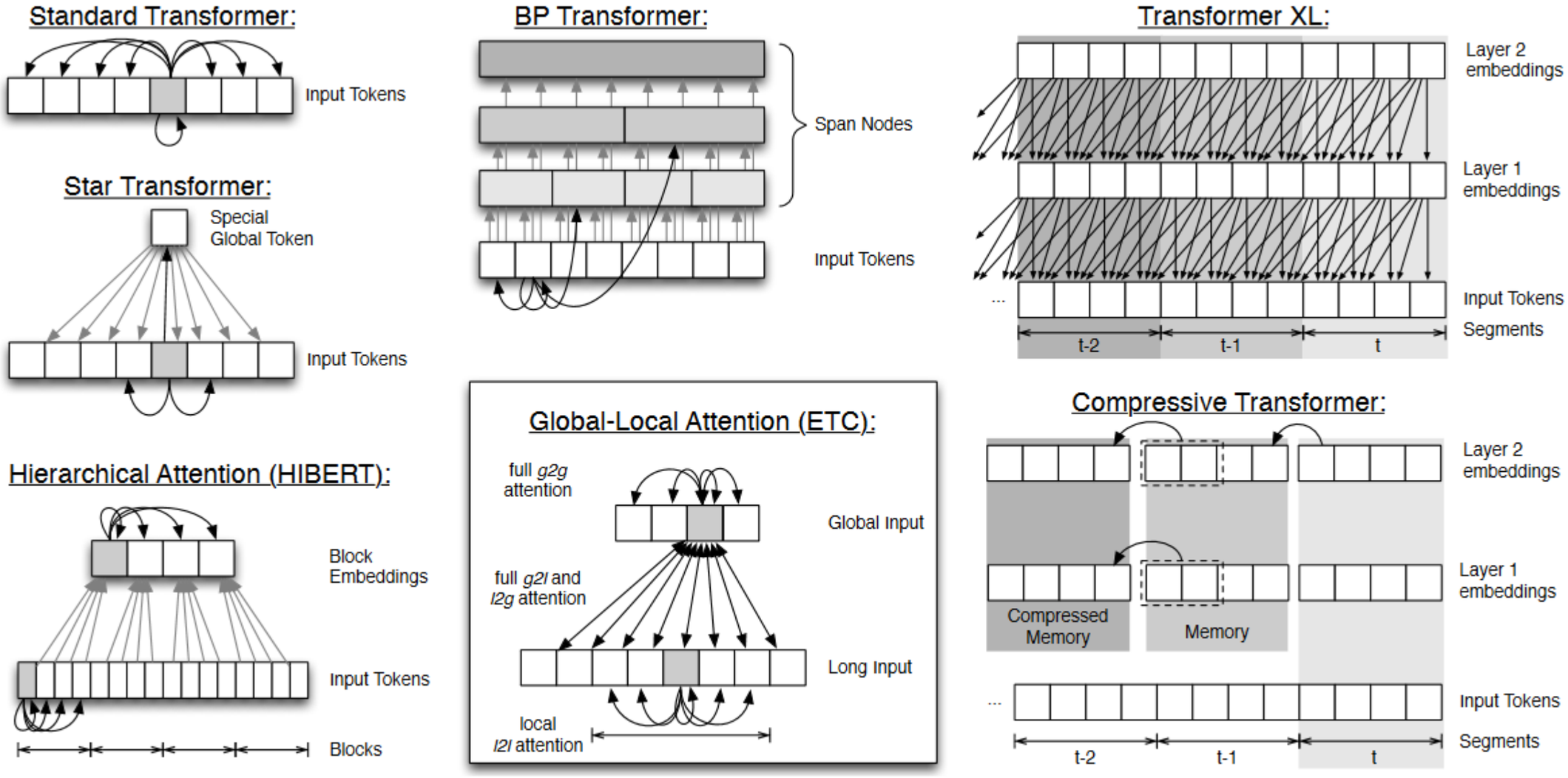


Figure 1: An illustration of mechanisms to scale attention to long inputs, including our proposed model, ETC.

Joshua Ainslie et al «ETC: Encoding Long and Structured Data in Transformers» // https://arxiv.org/abs/2004.08483

Extended Transformer Construction (ETC)

relative position encodings

Global-Local Attention

Вход делится на 2 части: global / local

Возможны 4 вида внимания g2g, g2l, l2g, l2l

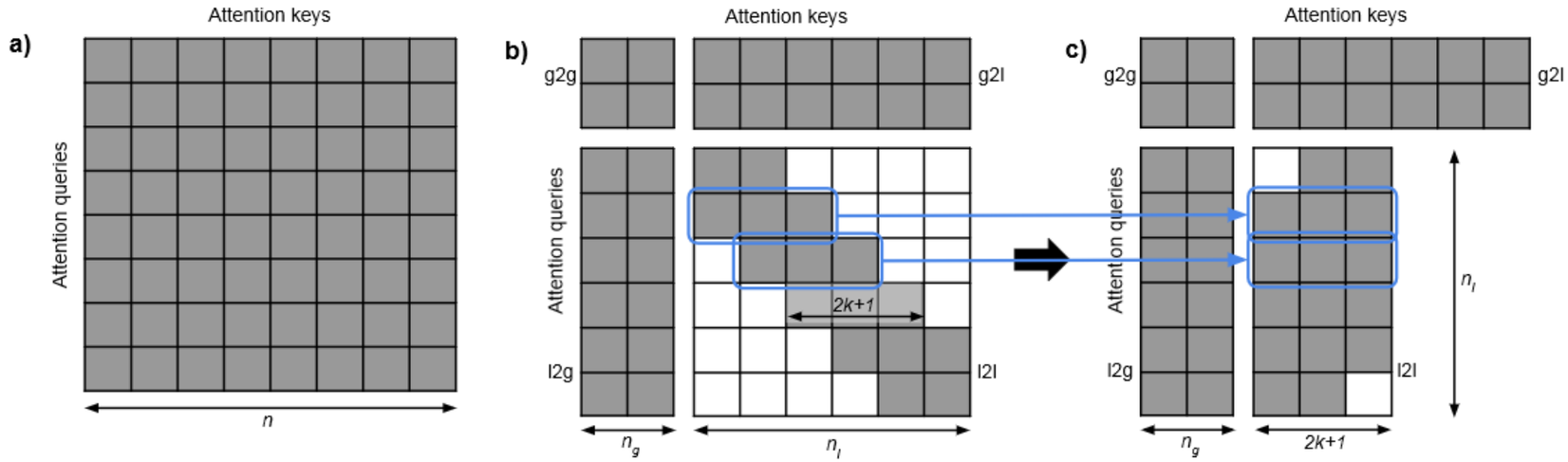


Figure 2: Sparsity diagram showing which attention queries (rows) can attend to which attention keys (columns) a) for standard Transformer attention with input size n; b) for global-local attention with input sizes n_g, n_l, and radius k; c) how l2l attention piece is reshaped into a much smaller attention matrix, limited by local radius.

не годится для авторегрессии, т.к. проблема с маскированием

Extended Transformer Construction (ETC)

Можно использовать информацию о структуре

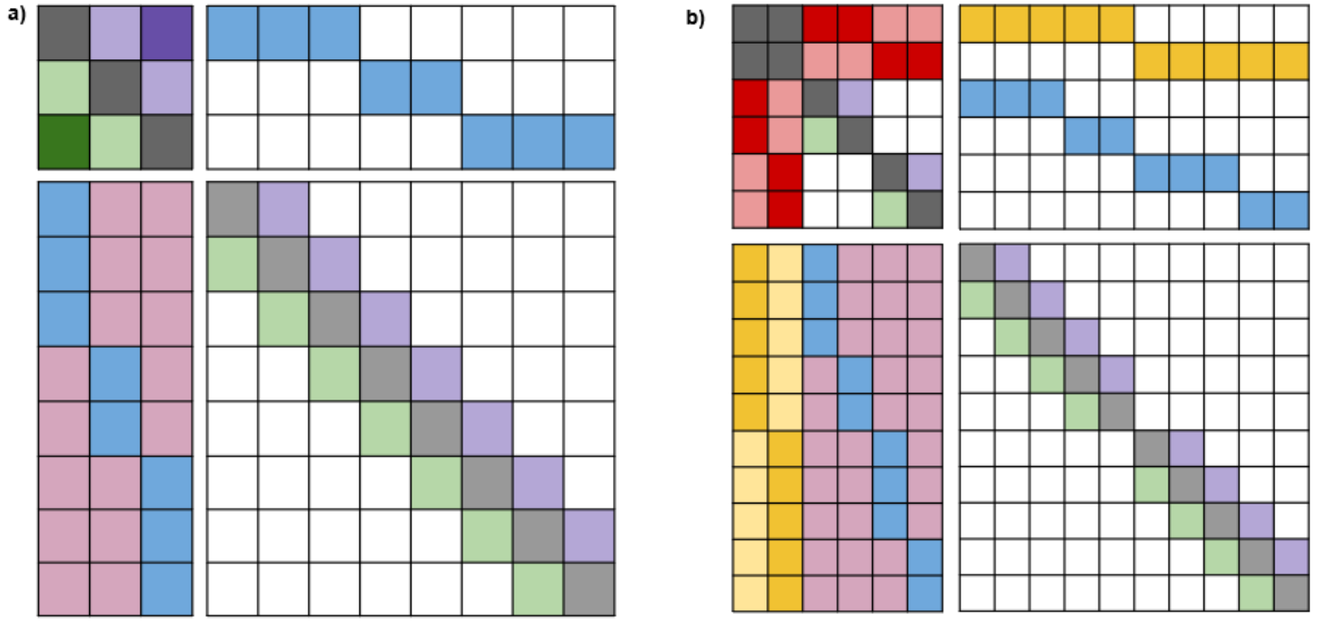


Figure 3: Some example attention patterns for (a) handling long inputs, (b) handling structured inputs. White background means attention is masked via M, and the different colors indicate different relative position labels.

Longformer: The Long-Document Transformer

– вариант Sparse Transformer

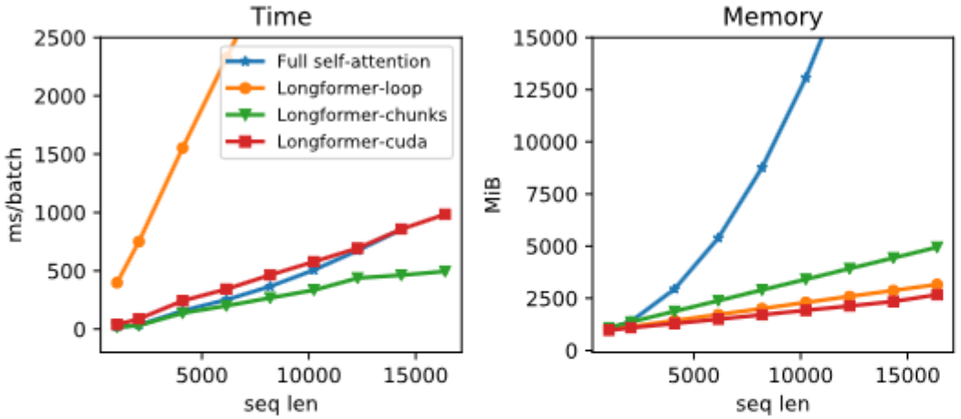


Figure 1: Runtime and memory of full self-attention and different implementations of Longformer's self-attention; `Longformer-loop` is non-vectorized, `Longformer-chunk` is vectorized, and `Longformer-cuda` is a custom cuda kernel implementations. Longformer's memory usage scales linearly with the sequence length, unlike the full self-attention mechanism that runs out of memory for long sequences on current GPUs. Different implementations vary in speed, with the vectorized `Longformer-chunk` being the fastest. More details are in section 3.2.

Beltagy et al., «Longformer: The Long-Document Transformer»
https://arxiv.org/abs/2004.05150

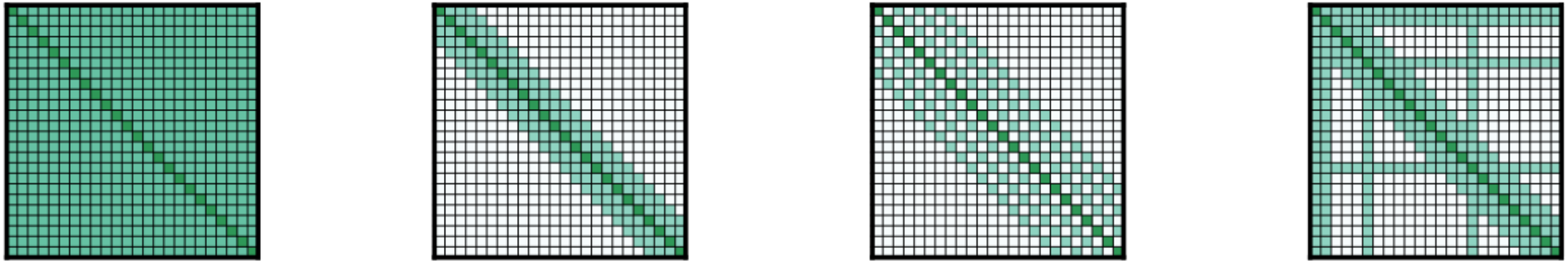
Longformer: The Long-Document Transformer

1) local sliding window attention – смотрим на соседей

2) dilated sliding window – разреженно смотрим

3) global attention for pre-selected positions – всегда смотрим на спец-токены

(CLS в задаче классификации, токены вопроса в задаче QA)



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

ещё CPC loss для предобучения

Longformer: The Long-Document Transformer

	QA			Coref.	Classification	
Model	WikiHop	TriviaQA	HotpotQA	OntoNotes	IMDB	Hyperpartisan
RoBERTa-base	72.4	74.3	63.5	78.4	95.3	87.4
Longformer-base	**75.0**	**75.2**	**64.4**	**78.6**	**95.7**	**94.8**

Table 7: Summary of finetuning results on QA, coreference resolution, and document classification. Results are on the development sets comparing our Longformer-base with RoBERTa-base. TriviaQA, Hyperpartisan metrics are F1, WikiHop and IMDB use accuracy, HotpotQA is joint F1, OntoNotes is average F1.

	R-1	R-2	R-L
Discourse-aware (2018)	35.80	11.05	31.80
Extr-Abst-TLM (2020)	41.62	14.69	38.03
Dancer (2020)	42.70	16.54	38.44
Pegasus (2020)	44.21	16.95	38.83
LED-large (seqlen: 4,096) (ours)	44.40	17.94	39.76
BigBird (seqlen: 4,096) (2020)	**46.63**	19.02	41.77
LED-large (seqlen: 16,384) (ours)	**46.63**	**19.62**	**41.83**

Table 11: Summarization results of Longformer-Encoder-Decoder (LED) on the arXiv dataset. Metrics from left to right are ROUGE-1, ROUGE-2 and ROUGE-L.

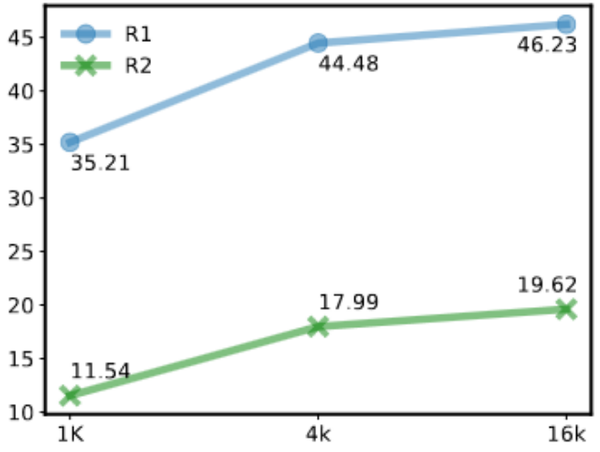


Figure 3: ROUGE-1 and ROUGE-2 of LED when varying the input size (arXiv validation set).

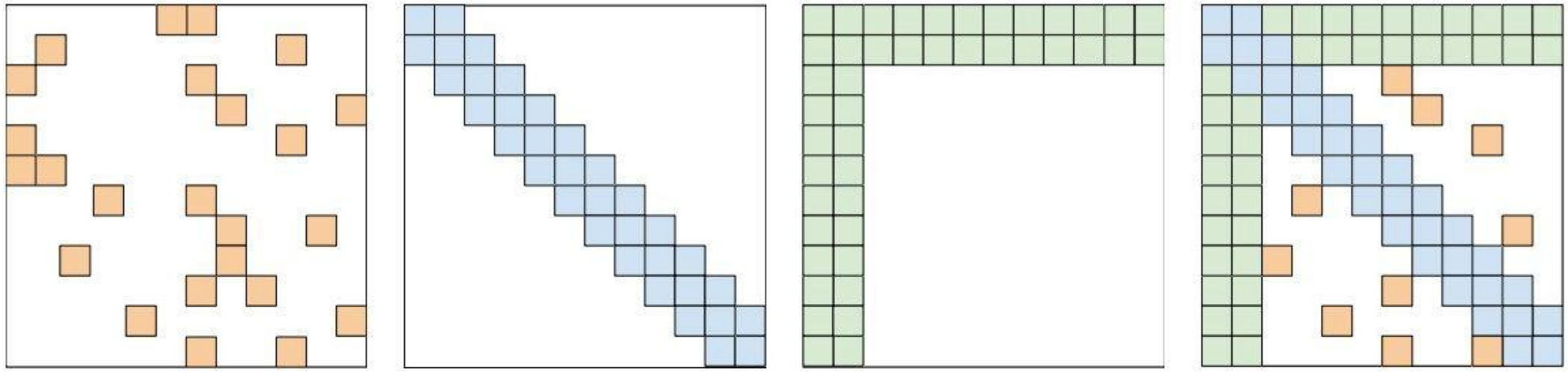
BigBird – продолжение ETC

паттерны внимания:

(1) глобальные: global tokens

(2) локальные: fixed patterns (local sliding windows)

(3) случайные: random attention (queries attend to random keys)



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Figure 1: Building blocks of the attention mechanism used in BIGBIRD. White color indicates absence of attention. (a) random attention with $r = 2$, (b) sliding window attention with $w = 3$ (c) global attention with $g = 2$. (d) the combined BIGBIRD model.

«Big Bird: Transformers for Longer Sequences» https://arxiv.org/abs/2007.14062

BigBird

Обычное внимание – полный граф,
его можно аппроксимировать случайным
нужны локальные связи + короткий путь между любой парой вершин

Обосновывается полнота по Тьюрингу

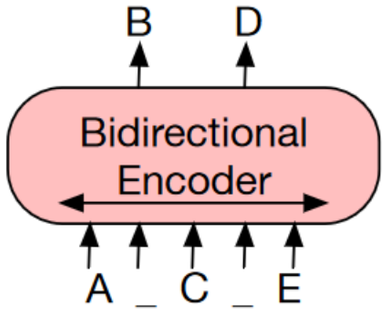
Механизм внимания линеен по числу токенов

BigBird

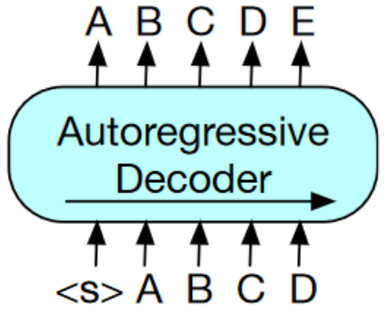
Model	HotpotQA			NaturalQ		TriviaQA		WikiHop
	Ans	Sup	Joint	LA	SA	Full	Verified	MCQ
HGN [26]	**82.2**	88.5	**74.2**	-	-	-	-	-
GSAN	81.6	88.7	73.9	-	-	-	-	-
ReflectionNet [32]	-	-	-	77.1	**64.1**	-	-	-
RikiNet-v2 [61]	-	-	-	76.1	61.3	-	-	-
Fusion-in-Decoder [39]	-	-	-	-	-	84.4	90.3	-
SpanBERT [42]	-	-	-	-	-	79.1	86.6	-
MRC-GCN [87]	-	-	-	-	-	-	-	78.3
MultiHop [14]	-	-	-	-	-	-	-	76.5
Longformer [8]	81.2	88.3	73.2	-	-	77.3	85.3	81.9
BIGBIRD-ETC	81.2	**89.1**	73.6	**77.8**	57.9	**84.5**	**92.4**	**82.3**

Table 3: Fine-tuning results on **Test** set for QA tasks. The Test results (F1 for HotpotQA, Natural Questions, TriviaQA, and Accuracy for WikiHop) have been picked from their respective leaderboard. For each task the top-3 leaders were picked not including BIGBIRD-etc. **For Natural Questions Long Answer (LA), TriviaQA, and WikiHop, BIGBIRD-ETC is the new state-of-the-art**. On HotpotQA we are third in the leaderboard by F1 and second by Exact Match (EM).

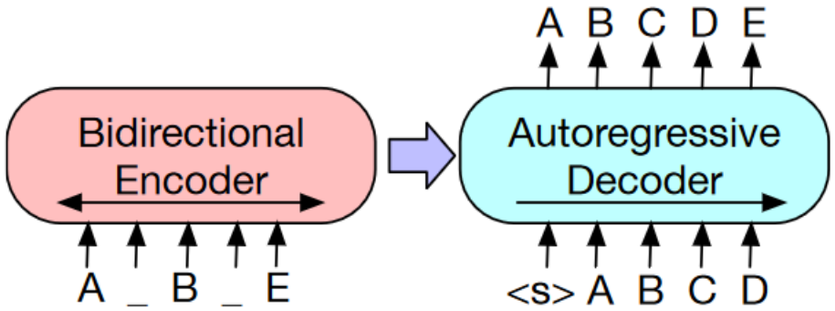
BART: шумоустраняющий seq2seq-автокодировщик на базе seq2seq-трансформера



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.



(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.



(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

Figure 1: A schematic comparison of BART with BERT (Devlin et al., 2019) and GPT (Radford et al., 2018).

BART

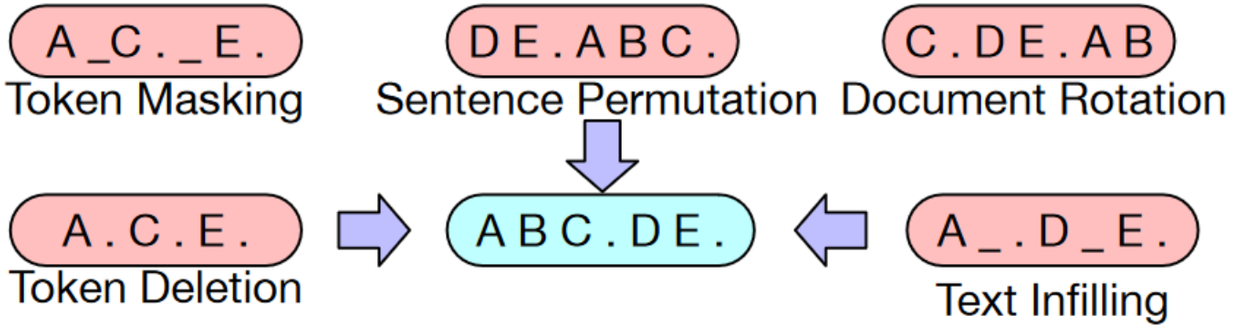


Figure 2: Transformations for noising the input that we experiment with. These transformations can be composed.

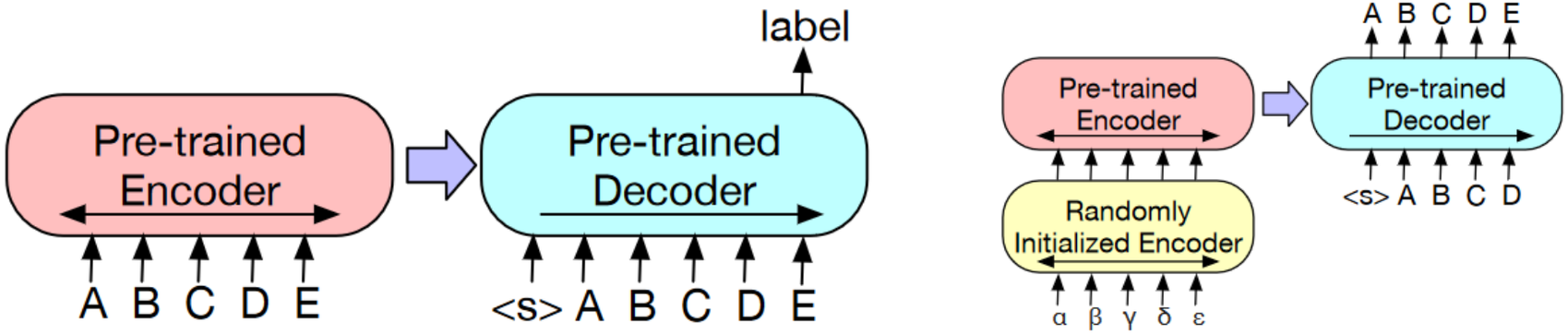
Разные варианты зашумления

Rotation – выбирается токен и сдвиг: текст с него начинался

Mike Lewis, et al « BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension» https://arxiv.org/abs/1910.13461

BART

Использование при донастройке на задачи



(a) To use BART for classification problems, the same input is fed into the encoder and decoder, and the representation from the final output is used.

(b) For machine translation, we learn a small additional encoder that replaces the word embeddings in BART. The new encoder can use a disjoint vocabulary.

Figure 3: Fine tuning BART for classification and translation.

В задаче 1 используют состояние последнего токена «end»

В задаче 2 – предлагаем end2end модель для обучения, в которой сначала переводим в английский шумоподавляющий автокодировщик

	SQuAD 1.1 EM/F1	SQuAD 2.0 EM/F1	MNLI m/mm	SST Acc	QQP Acc	QNLI Acc	STS-B Acc	RTE Acc	MRPC Acc	CoLA Mcc
BERT	84.1/90.9	79.0/81.8	86.6/-	93.2	91.3	92.3	90.0	70.4	88.0	60.6
UniLM	-/-	80.5/83.4	87.0/85.9	94.5	-	92.7	-	70.9	-	61.1
XLNet	**89.0**/94.5	86.1/88.8	89.8/-	95.6	91.8	93.9	91.8	83.8	89.2	63.6
RoBERTa	88.9/**94.6**	**86.5/89.4**	**90.2/90.2**	96.4	92.2	94.7	**92.4**	86.6	**90.9**	**68.0**
BART	88.8/**94.6**	86.1/89.2	89.9/90.1	**96.6**	**92.5**	**94.9**	91.2	**87.0**	90.4	62.8

Table 2: Results for large models on SQuAD and GLUE tasks. BART performs comparably to RoBERTa and XLNet, suggesting that BART's uni-directional decoder layers do not reduce performance on discriminative tasks.

	CNN/DailyMail			XSum		
	R1	R2	RL	R1	R2	RL
Lead-3	40.42	17.62	36.67	16.30	1.60	11.95
PTGEN (See et al., 2017)	36.44	15.66	33.42	29.70	9.21	23.24
PTGEN+COV (See et al., 2017)	39.53	17.28	36.38	28.10	8.02	21.72
UniLM	43.33	20.21	40.51	-	-	-
BERTSUMABS (Liu & Lapata, 2019)	41.72	19.39	38.76	38.76	16.33	31.15
BERTSUMEXTABS (Liu & Lapata, 2019)	42.13	19.60	39.18	38.81	16.50	31.27
BART	**44.16**	**21.28**	**40.90**	**45.14**	**22.27**	**37.25**

Table 3: Results on two standard summarization datasets. BART outperforms previous work on summarization on two tasks and all metrics, with gains of roughly 6 points on the more abstractive dataset.

BART

This is the first time anyone has been recorded to run a full marathon of 42.195 kilometers (approximately 26 miles) under this pursued landmark time. It was not, however, an officially sanctioned world record, as it was not an ”open race” of the IAAF. His time was 1 hour 59 minutes 40.2 seconds. Kipchoge ran in Vienna, Austria. It was an event specifically designed to help Kipchoge break the two hour barrier.	Kenyan runner Eliud Kipchoge has run a marathon in less than two hours.
PG&E stated it scheduled the blackouts in response to forecasts for high winds amid dry conditions. The aim is to reduce the risk of wildfires. Nearly 800 thousand customers were scheduled to be affected by the shutoffs which were expected to last through at least midday tomorrow.	Power has been turned off to millions of customers in California as part of a power shutoff plan.

Table 7: Example summaries from the XSum-tuned BART model on WikiNews articles. For clarity, only relevant excerpts of the source are shown. Summaries combine information from across the article and prior knowledge.

Sparse Transformer

– сеть для предсказания последовательности (текст, картинка, звук…)

Механизм самовнимания требует $O(N^2)$ операций, тут получили $O(N \cdot \mathrm{sqrt}(N))$

Есть стандартный приём – не хранить матрицу весов внимания, а пересчитывать:

DATA TYPE	STORED	RECOMPUTED
1024 text tokens (several paragraphs)	1.0 GB	16 MB
32x32x3 pixels (CIFAR-10 image)	9.6 GB	151 MB
64x64x3 pixels (Imagenet 64 image)	154 GB	2.4 GB
24,000 samples (~2 seconds of 12 kHz audio)	590 GB	9.2GB

Attention memory usage for a deep Transformer (64 layers and 4 heads) when matrices are stored in memory or recomputed during the backward pass. For reference, standard GPUs used for deep learning typically have memory of 12-32 GB.

Sparse Transformer

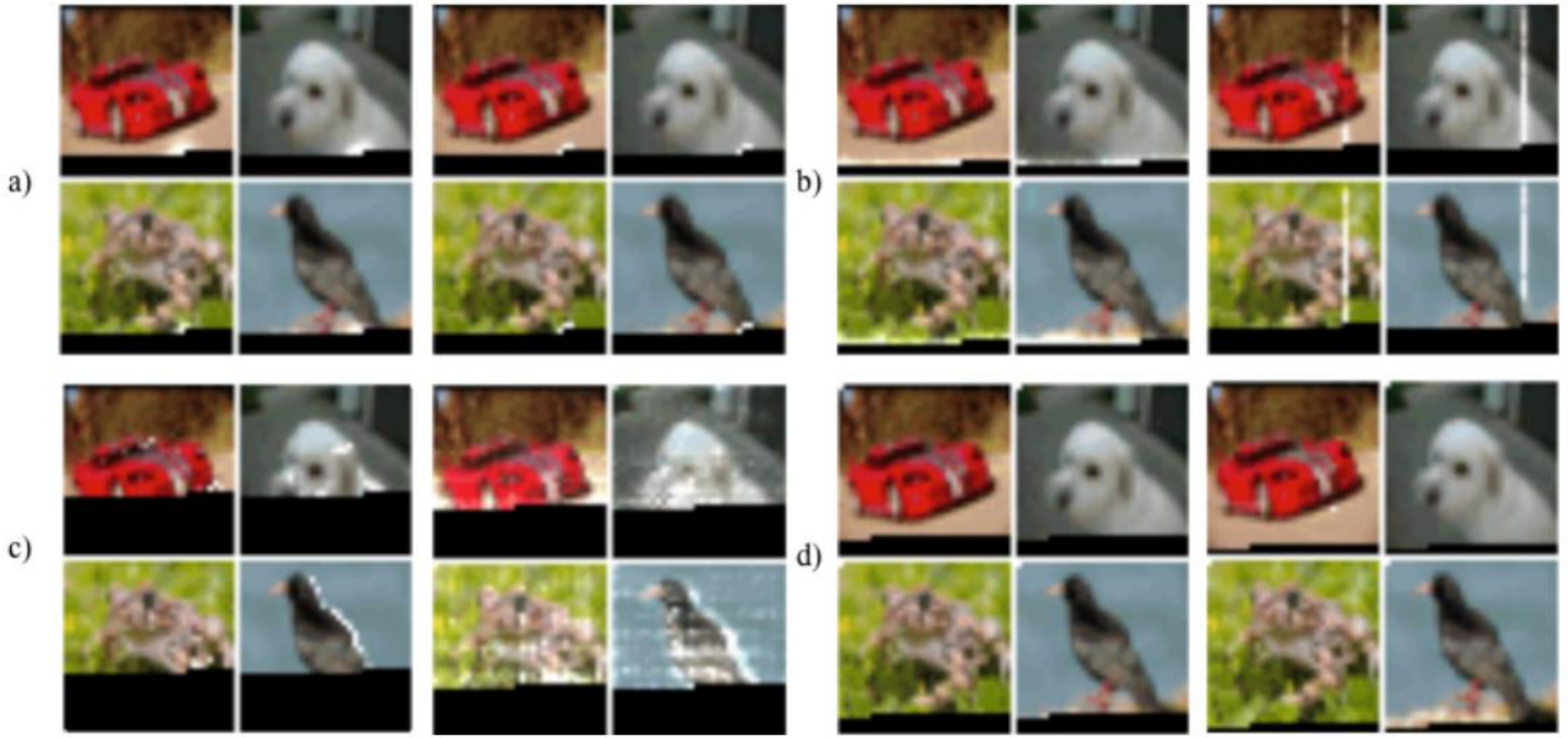


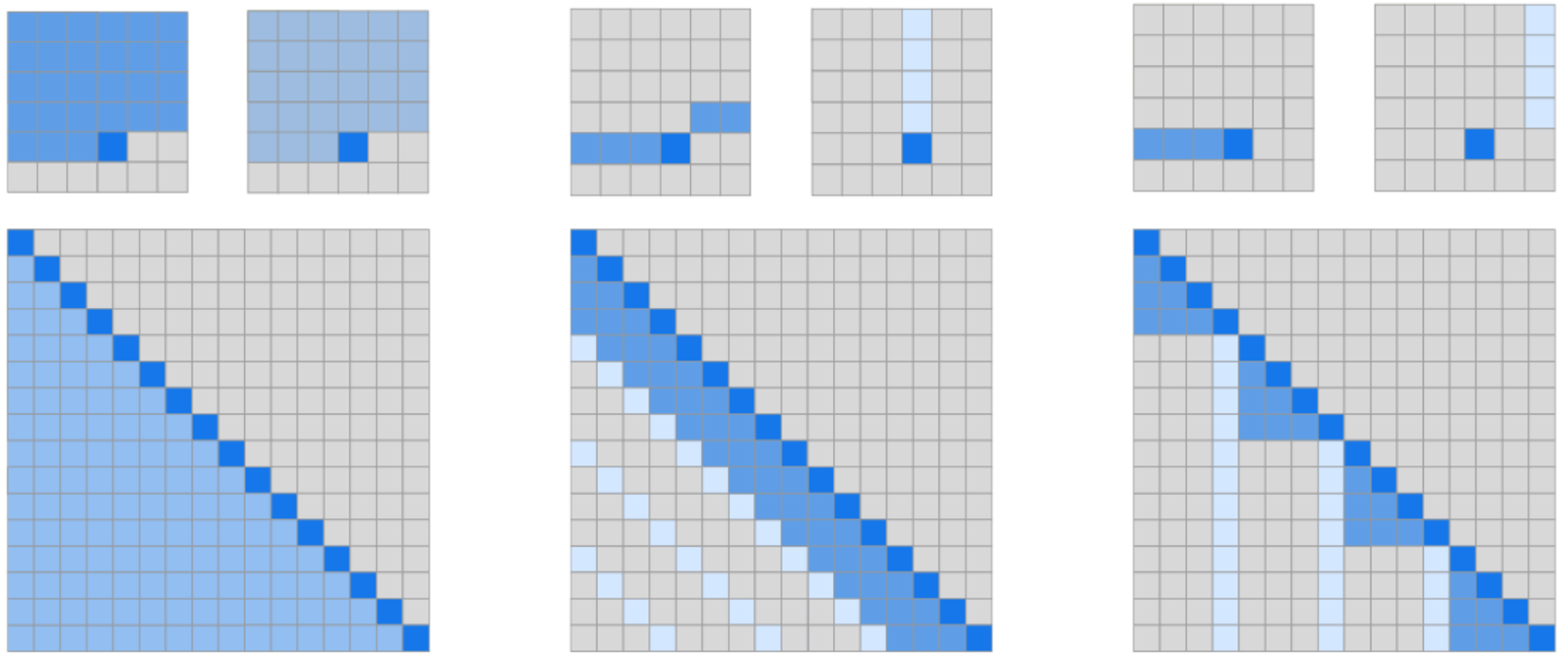
Figure 2. Learned attention patterns from a 128-layer network on CIFAR-10 trained with full attention. White highlights denote attention weights for a head while generating a given pixel, and black denotes the autoregressive mask. Layers are able to learn a variety of specialized sparse structures, which may explain their ability to adapt to different domains. a) Many early layers in the network learn locally connected patterns, which resemble convolution. b) In layers 19 and 20, the network learned to split the attention across a row attention and column attention, effectively factorizing the global attention calculation. c) Several attention layers showed global, data-dependent access patterns. d) Typical layers in layers 64-128 exhibited high sparsity, with positions activating rarely and only for specific input patterns.

На самом деле, не всё внимание нужно

Эксперименты с изображениями (белым подсвечена нужная часть при генерации)

Sparse Transformer

2D-факторизация матрицы внимания



(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

Figure 3. Two 2d factorized attention schemes we evaluated in comparison to the full attention of a standard Transformer (a). The top row indicates, for an example 6x6 image, which positions two attention heads receive as input when computing a given output. The bottom row shows the connectivity matrix (not to scale) between all such outputs (rows) and inputs (columns). Sparsity in the connectivity matrix can lead to significantly faster computation. In (b) and (c), full connectivity between elements is preserved when the two heads are computed sequentially. We tested whether such factorizations could match in performance the rich connectivity patterns of Figure 2.

Rewon Child, et al «Generating Long Sequences with Sparse Transformers»

https://arxiv.org/abs/1904.10509

https://openai.com/blog/sparse-transformer/

Reformer: The Efficient Transformer

Улучшения трансформера: LSH + reversible residual layers

1) dot-product attention → locality-sensitive hashing

сложность: $O(L^2) \to O(L \log L)$

результат внимания определяется токенами с большими весами, поэтому ищем ближайших соседей с помощью LSH и только их используем для вычисления внимания

теперь можно обрабатывать большие последовательности

$$\mathbf{Q} = \mathbf{K}$$

LSH bucketing токены раскладываются по бакетам

Sort и внутри упорядочиваются

Chunk делятся на группы (для распараллеливания вычислений)

Attend токены видят токены своего бакета $S_i = \{j : h(\mathbf{q}_i) = h(\mathbf{k}_j)\}$ и предыдущих

Nikita Kitaev et al. «Reformer: the efficient transformer» https://arxiv.org/abs/2001.04451

Reformer: The Efficient Transformer

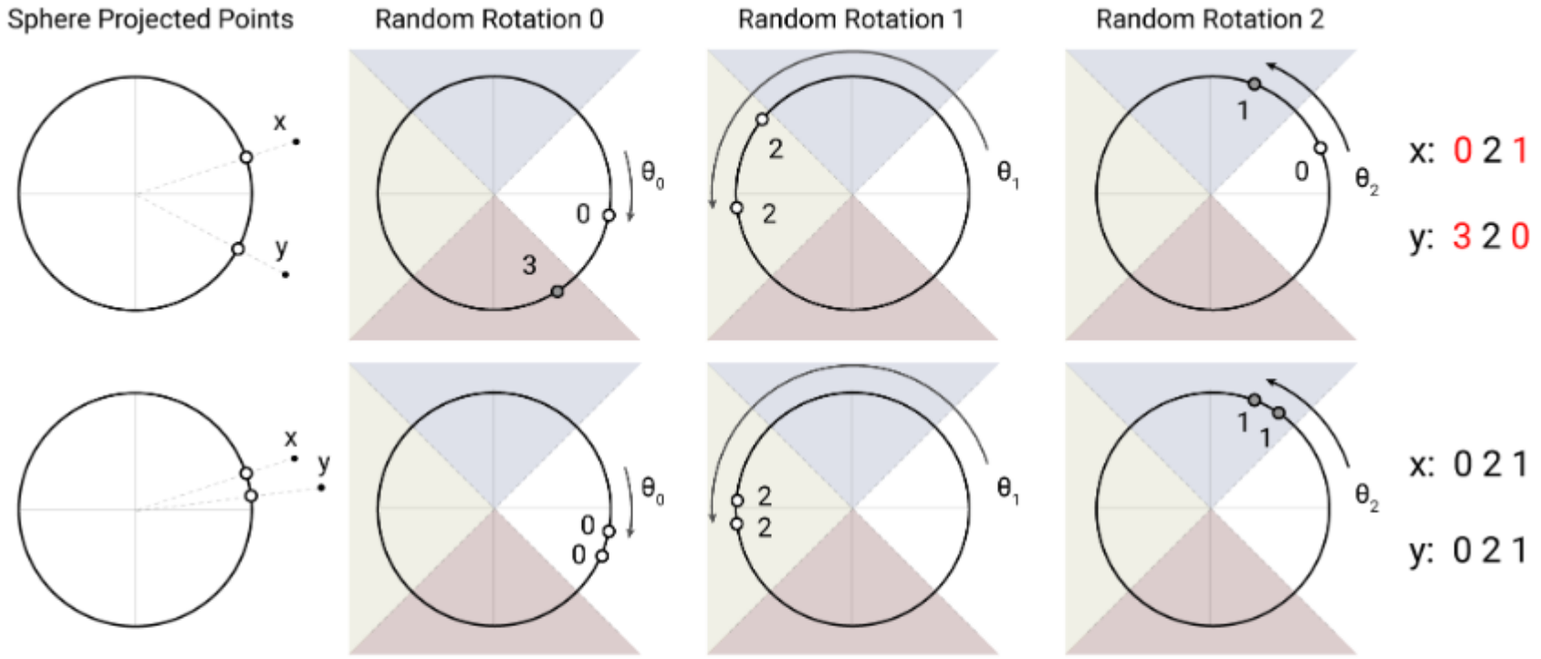


Figure 1: An angular locality sensitive hash uses random rotations of spherically projected points to establish buckets by an argmax over signed axes projections. In this highly simplified 2D depiction, two points x and y are unlikely to share the same hash buckets (above) for the three different angular hashes unless their spherical projections are close to one another (below).

Хеш-функция

$$h(x) = \arg\max([xR; -xR]) \quad \mathbf{R} \in \mathbb{R}^{d \times b/2}$$

Reformer: The Efficient Transformer

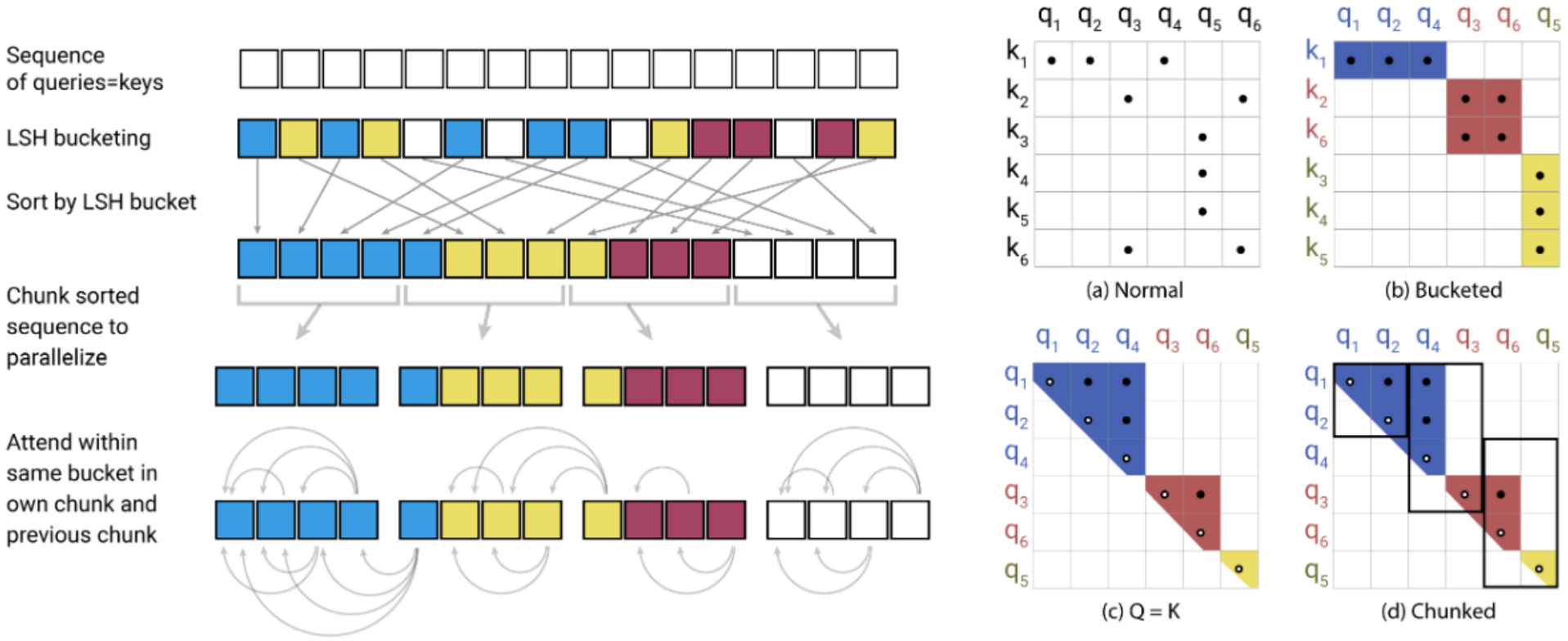


Figure 2: Simplified depiction of LSH Attention showing the hash-bucketing, sorting, and chunking steps and the resulting causal attentions. (a-d) Attention matrices for these varieties of attention.

Reformer: The Efficient Transformer

2) residual layers → reversible residual layers (идея из RevNets)

храним активации один раз, на не L (число слоёв)

(при обратном проходе они опять вычисляются)

$$x \mapsto y = x + F(x)$$

$$(x_1, x_2) \mapsto (y_1 = x_1 + F(x_2),\ y_2 = x_2 + G(y_1))$$

теперь F – attention, G – feedforward

Table 3: Memory and time complexity of Transformer variants. We write d_{model} and d_{ff} for model depth and assume $d_{ff} \geq d_{model}$; b stands for batch size, l for length, n_l for the number of layers. We assume $n_c = l/32$ so $4l/n_c = 128$ and we write $c = 128^2$.

Model Type	Memory Complexity	Time Complexity
Transformer	$\max(bld_{ff}, bn_h l^2)n_l$	$(bld_{ff} + bn_h l^2)n_l$
Reversible Transformer	$\max(bld_{ff}, bn_h l^2)$	$(bn_h l d_{ff} + bn_h l^2)n_l$
Chunked Reversible Transformer	$\max(bld_{model}, bn_h l^2)$	$(bn_h l d_{ff} + bn_h l^2)n_l$
LSH Transformer	$\max(bld_{ff}, bn_h l n_r c)n_l$	$(bld_{ff} + bn_h n_r lc)n_l$
Reformer	$\max(bld_{model}, bn_h l n_r c)$	$(bld_{ff} + bn_h n_r lc)n_l$

Reformer: The Efficient Transformer

Задача дополнения изображений

теперь даже можем делать такие задачи с кучей объектов внимания – пикселей

Routing Transformer

В пространстве Q и K происходи кластеризация (есть центры кластеров μ)

онлайновый кластеризатор k-means

During training, we update each cluster centroid μ by an exponentially moving average of all the keys and queries assigned to it:

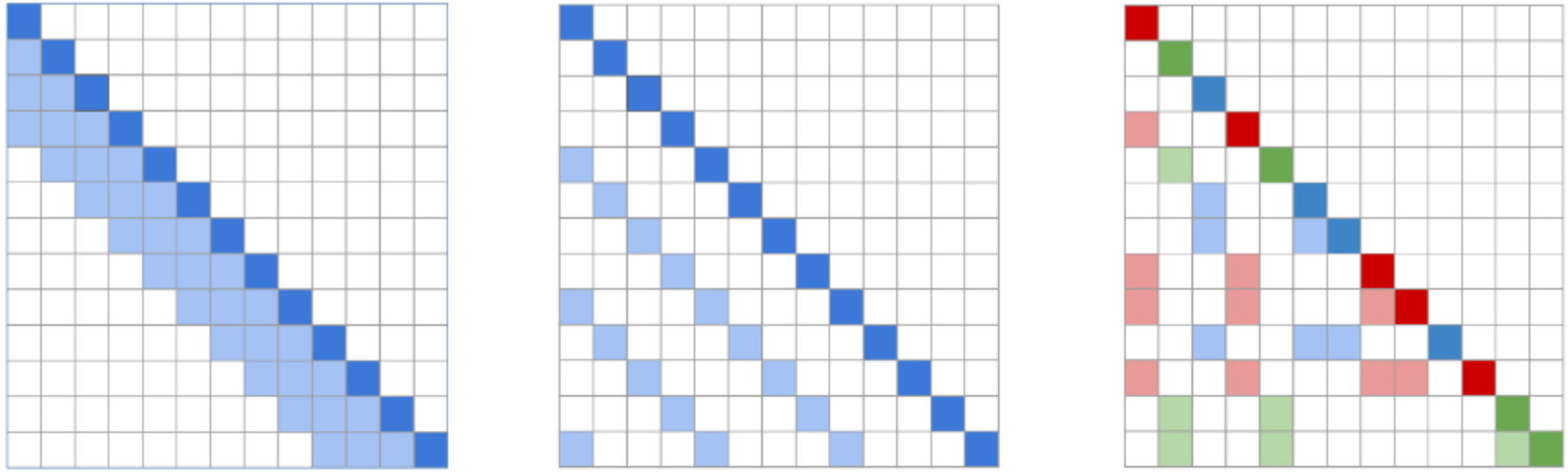
$$\mu \leftarrow \lambda\mu + \frac{(1-\lambda)}{2}\sum_{i:\mu(Q_i)=\mu} Q_i + \frac{(1-\lambda)}{2}\sum_{j:\mu(K_j)=\mu} K_j$$

Routing Strategy – внимание только по токенам аналогичного кластера

$$X_i' = \sum_{\substack{j:K_j\in\mu(Q_i),\\ j<i}} A_{ij}V_j$$

Roy et al. «Efficient Content-Based Sparse Attention with RoutingTransformers» https://arxiv.org/pdf/2003.05997.pdf

Routing Transformer



(a) Local attention (b) Strided attention (c) Routing attention

Figure 1: Figures showing 2-D attention schemes for the Routing Transformer compared to local attention and strided attention of (Child et al., 2019). The rows represent the outputs while the columns represent the inputs. For local and strided attention, the colored squares represent the elements every output row attends to. For attention routed as in Section 4.1, the different colors represent cluster memberships for the output token.

Sinkhorn Transformers

пересортировка ключей и значений
внимание по блокам

Sinkhorn balancing operator (Sinkhorn, 1964; Adams andZemel, 2011)
превращает матрицу в перестановочную

Tay et al., «Sparse Sinkhorn Attention» https://arxiv.org/pdf/2002.11296.pdf

Linear Transformer

сложность линейная, а не квадратичная

ядра (kernels)

Идея – вспомним выражение для self-attention

$$V_i' = \frac{\sum_{j=1}^{p} \text{sim}(Q_i, K_j) V_j}{\sum_{j=1}^{p} \text{sim}(Q_i, K_j)}$$

$$\text{sim}(q, k) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$$

Katharopoulos et al., «Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention» // https://arxiv.org/abs/2006.16236

Linear Transformer

пусть sim – ядро, переписываем в виде

$$V_i' = \frac{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j) V_j}{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j)}$$

$$V_i' = \frac{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)}$$

$$\left(\phi(Q)\,\phi(K)^T\right) V = \phi(Q)\left(\phi(K)^T V\right)$$

Можно вычислить $\sum_{j=1}^{N} \phi(K_j) V_j^T$ **и** $\sum_{j=1}^{N} \phi(K_j)$

один раз – и использовать для каждого query

тогда сложность не N^2, а N

Linear Transformer

Трансформер становится похожин на RNN

$$
\begin{aligned}
s_0 &= 0, \\
z_0 &= 0, \\
s_i &= s_{i-1} + \phi\left(x_i W_K\right)\left(x_i W_V\right)^T, \\
z_i &= z_{i-1} + \phi\left(x_i W_K\right), \\
y_i &= f_l\left(\frac{\phi\left(x_i W_Q\right)^T s_i}{\phi\left(x_i W_Q\right)^T z_i} + x_i\right).
\end{aligned}
$$

2 hidden states:
attention memory s + normalizer memory z

Linear Transformer

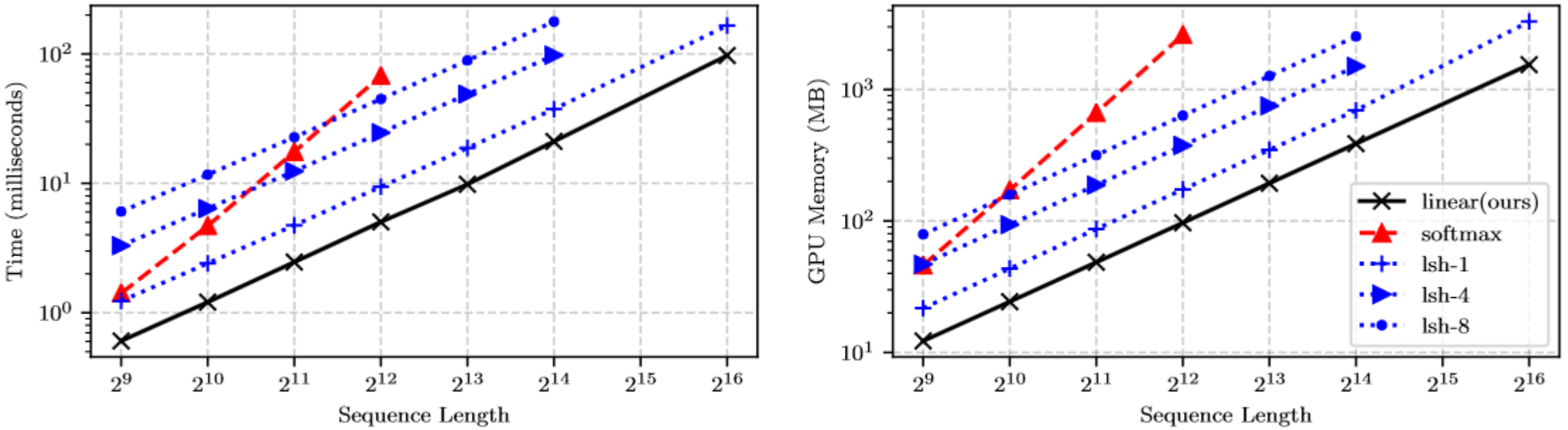


Figure 1: Comparison of the computational requirements for a forward/backward pass for Reformer (lsh-X), softmax attention and linear attention. Linear and Reformer models scale linearly with the sequence length unlike softmax which scales with the square of the sequence length both in memory and time. Full details of the experiment can be found in § 4.1.

Linformer

~ Low-rank approximation

Model Architecture	Complexity per Layer	Sequential Operation
Recurrent	$O(n)$	$O(n)$
Transformer, (Vaswani et al., 2017)	$O(n^2)$	$O(1)$
Sparse Tansformer, (Child et al., 2019)	$O(n\sqrt{n})$	$O(1)$
Reformer, (Kitaev et al., 2020)	$O(n \log(n))$	$O(\log(n))$
Linformer	$O(n)$	$O(1)$

Table 1: Per-layer time complexity and minimum number of sequential operations as a function of sequence length (n) for various architectures.

вместо квадратичной линейная сложность

Sinong Wang « Linformer: Self-Attention with Linear Complexity» //
https://arxiv.org/abs/2006.04768

Linformer

Матрица $$P = \underbrace{\text{softmax}\left[\frac{QW_i^Q(KW_i^K)^T}{\sqrt{d}}\right]}_{A} = \exp(A)\cdot D_A^{-1}$$ **малого ранга!**

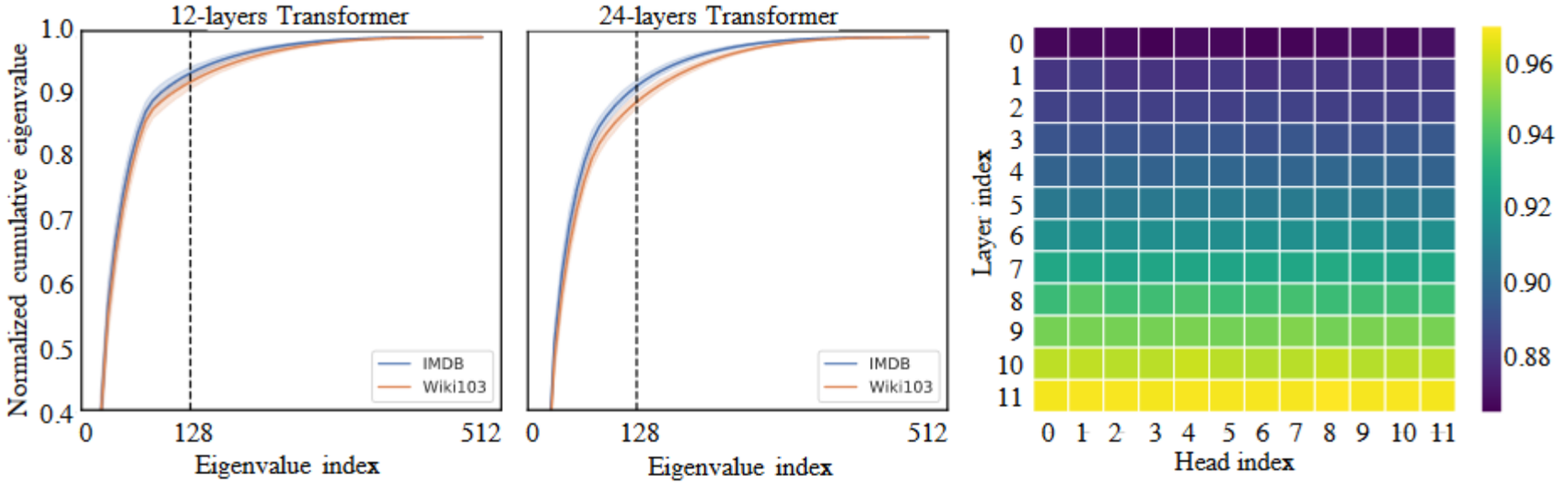
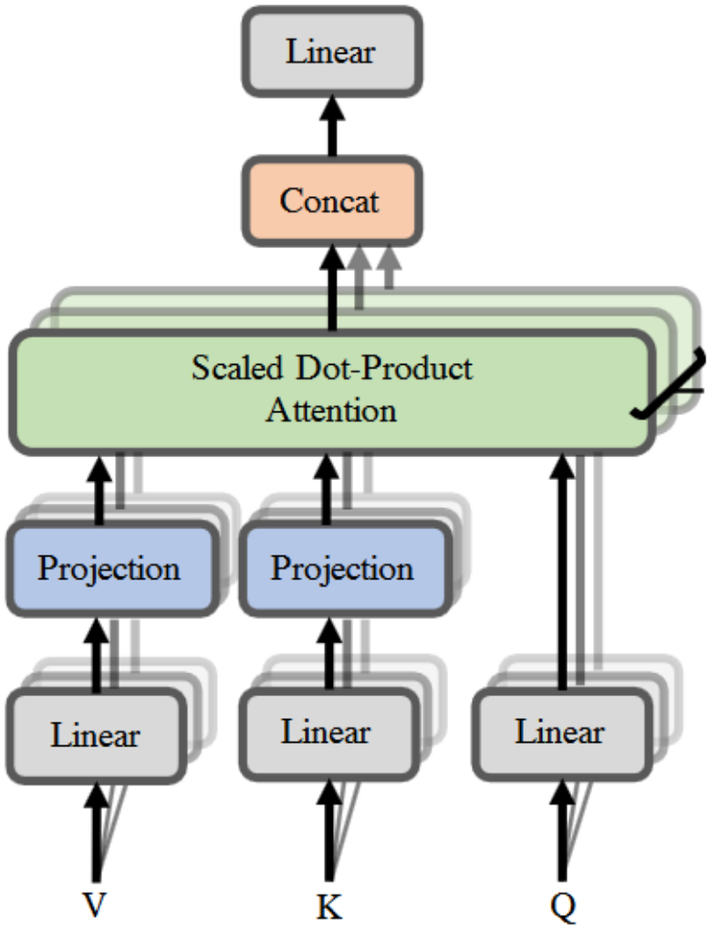
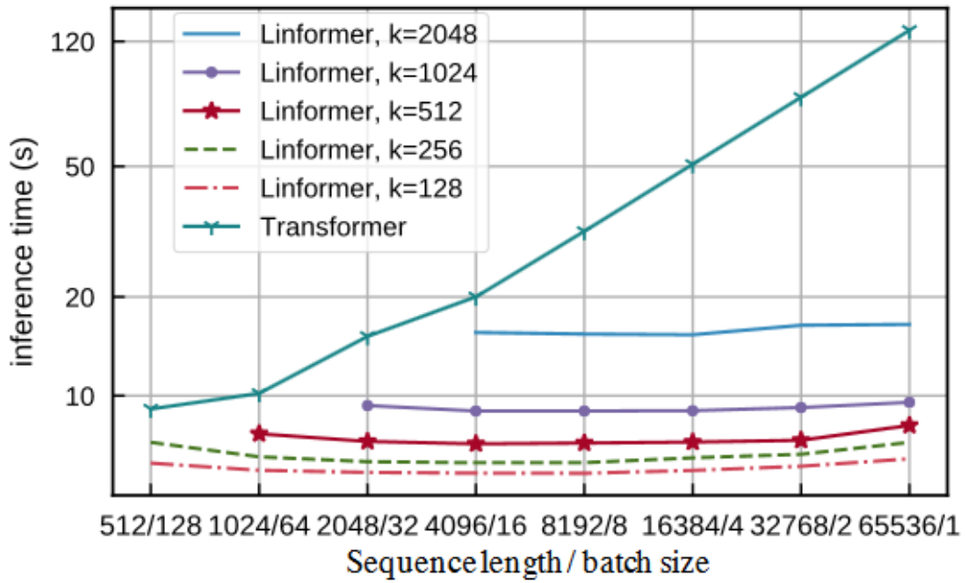


Figure 1: Left two figures are spectrum analysis of the self-attention matrix in pretrained transformer model (Liu et al., 2019) with $n = 512$. The Y-axis is the normalized cumulative singular value of context mapping matrix P, and the X-axis the index of largest eigenvalue. The results are based on both RoBERTa-base and large model in two public datasets: Wiki103 and IMDB. The right figure plots the heatmap of normalized cumulative eigenvalue at the 128-th largest eigenvalue across different layers and heads in Wiki103 data.

Linformer





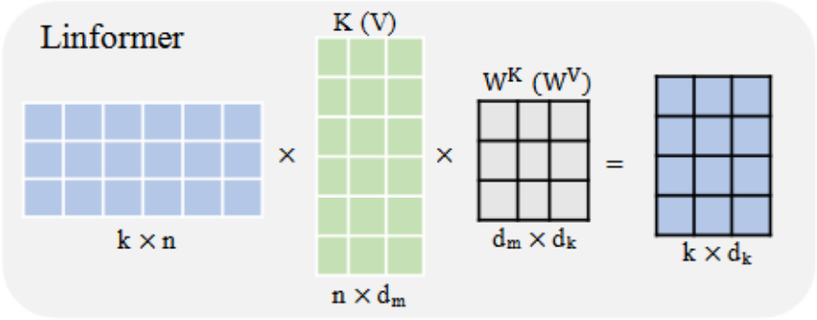


Figure 2: Left and bottom-right show architecture and example of our proposed multihead linear self-attention. Top right shows inference time vs. sequence length for various Linformer models.

проекция V и K из n×d в k×d'

Performer

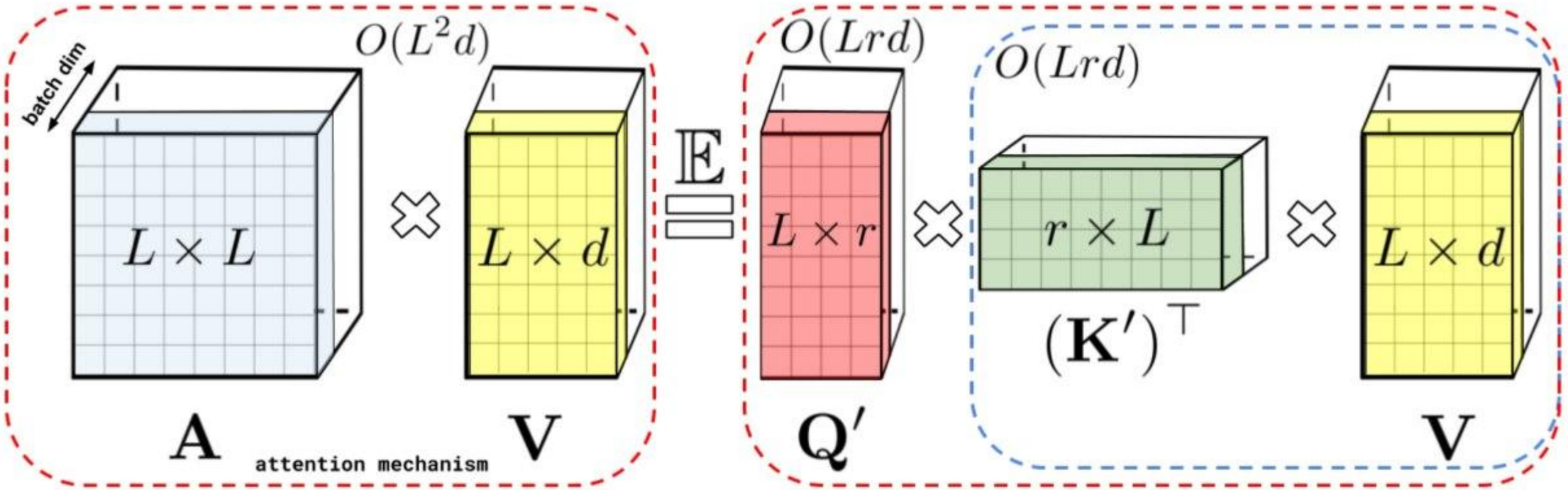


Figure 1: Approximation of the regular attention mechanism $\mathbf{AV}$ (before $\mathbf{D}^{-1}$-renormalization) via (random) feature maps. Dashed-blocks indicate order of computation with corresponding time complexities attached.

~ Low-rank approximation – в итоге линейное масштабирование

если умножать в указанном пунктиром порядке,

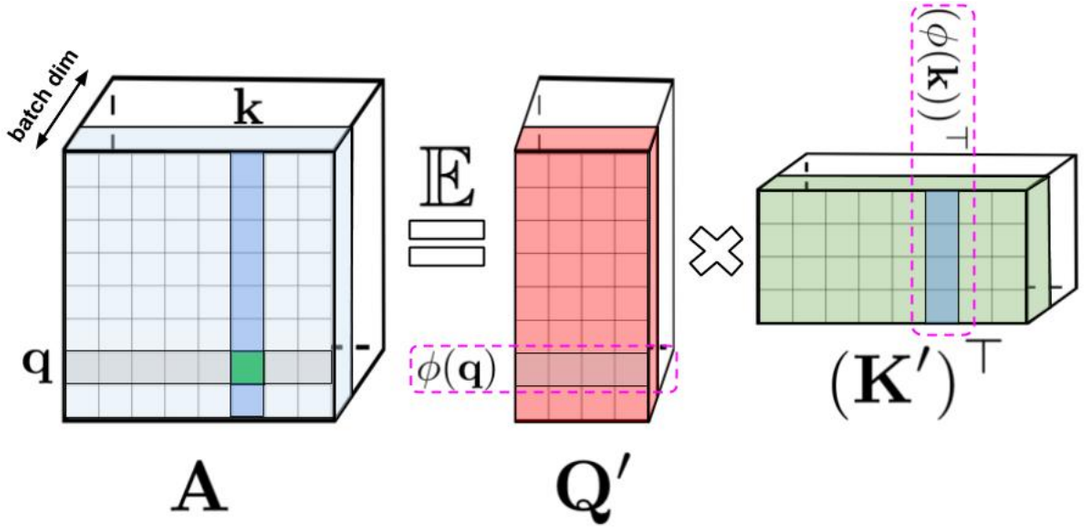
то линейный механизм внимания, явно не конструируя A

Choromanski et al., 2020 «Rethinking attention with performers» // https://arxiv.org/pdf/2009.14794.pdf

Performer: Fast Attention via Orthogonal Random Features (FAVOR++)

Идея похожа на linear transformer – используем ядра, строится ядро для softmax

Это описание только FA (Fast Attention) идеи…



FAVOR+ works for attention blocks using matrices $\mathbf{A} \in \mathbb{R}^{L\times L}$ of the form $\mathbf{A}(i,j) = \mathrm{K}(\mathbf{q}_i^\top, \mathbf{k}_j^\top)$, with $\mathbf{q}_i/\mathbf{k}_j$ standing for the i^{th}/j^{th} query/key row-vector in $\mathbf{Q}/\mathbf{K}$ and kernel $\mathrm{K} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$ defined for the (usually randomized) mapping: $\phi : \mathbb{R}^d \to \mathbb{R}_+^r$ (for some $r > 0$) as:

$$\mathrm{K}(\mathbf{x}, \mathbf{y}) = \mathbb{E}[\phi(\mathbf{x})^\top \phi(\mathbf{y})]. \tag{3}$$

We call $\phi(\mathbf{u})$ a *random feature map* for $\mathbf{u} \in \mathbb{R}^d$. For $\mathbf{Q}', \mathbf{K}' \in \mathbb{R}^{L\times r}$ with rows given as $\phi(\mathbf{q}_i^\top)^\top$ and $\phi(\mathbf{k}_i^\top)^\top$ respectively, Equation 3 leads directly to the efficient attention mechanism of the form:

$$\widehat{\mathrm{Att}}_{\leftrightarrow}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \widehat{\mathbf{D}}^{-1}(\mathbf{Q}'((\mathbf{K}')^\top \mathbf{V})), \qquad \widehat{\mathbf{D}} = \mathrm{diag}(\mathbf{Q}'((\mathbf{K}')^\top \mathbf{1}_L)). \tag{4}$$

Here $\widehat{\mathrm{Att}}_{\leftrightarrow}$ stands for the approximate attention and brackets indicate the order of computations. It is easy to see that such a mechanism is characterized by space complexity $O(Lr + Ld + rd)$ and time complexity $O(Lrd)$ as opposed to $O(L^2 + Ld)$ and $O(L^2 d)$ of the regular attention (see also Fig. 1).

есть более ранняя работа

https://papers.nips.cc/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf

Performer: Fast Attention via Orthogonal Random Features (FAVOR++)

The above computation is still quadratic in complexity. Hence, the Performer leverages approximation tricks to avoid storing and computing the $N \times N$ attention matrix. It leverages *orthogonal random features* (ORF) for doing so. The final attention output Y of the Performer is described as follows:

$$Y = \hat{D}^{-1}(Q'((K')^\top V)) \tag{6}$$

where $\hat{D} = \text{diag}(Q'((K')^\top 1_N))$, $Q' = D_Q \phi(Q^\top)^\top$, and $K' = D_K \phi(K^\top)^\top$. Note that $D_Q = g(Q_i^\top), D_K = h(K_i^\top)$. The function $\phi(x)$ is defined as:

$$\phi(X) = \frac{c}{\sqrt{M}} f(Wx + b)^\top \tag{7}$$

where $c > 0$ is a constant, $W \in \mathbb{R}^{M \times d}$ is a random feature matrix, and M is the dimensionality of this matrix that controls the number of random features. We are able to see that we do not explicitly compute $A = QK^\top$ and hence avoid paying the N^2 cost.

Performer

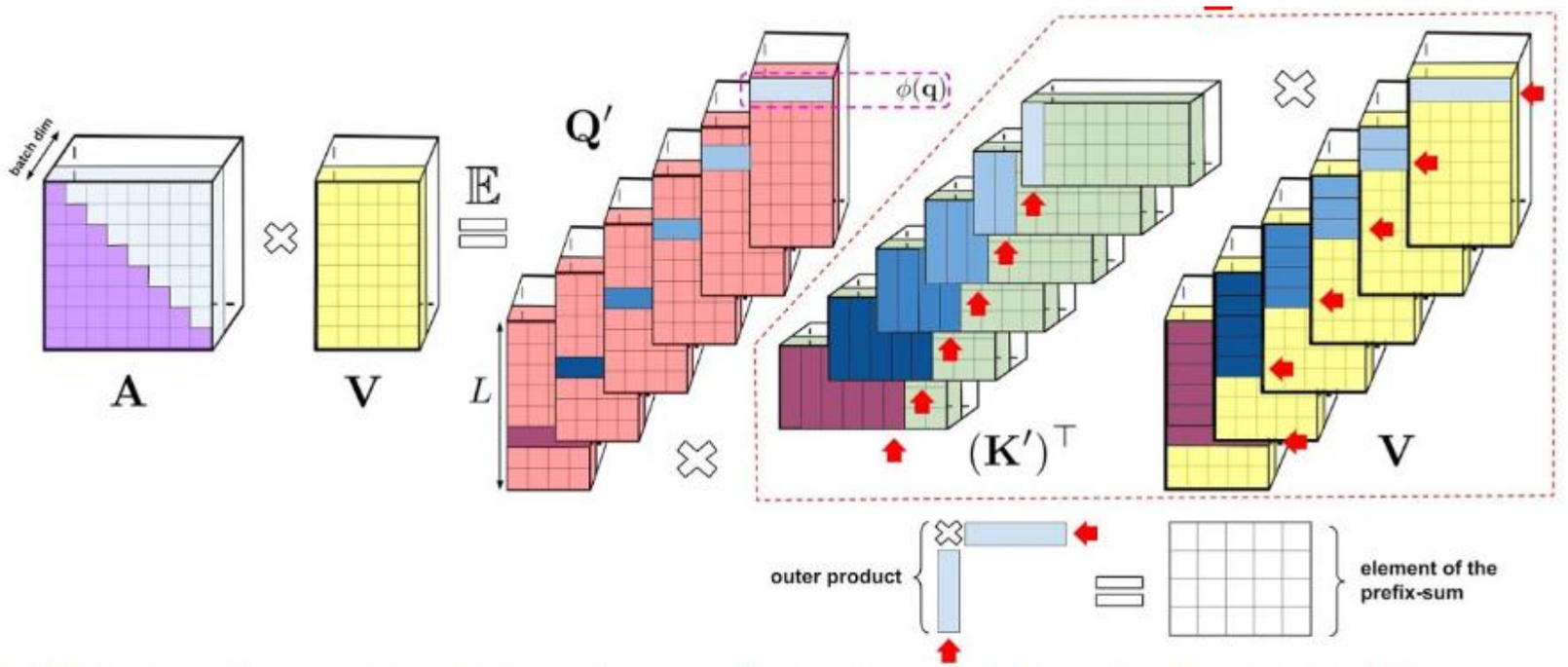


Figure 8: Visual representation of the prefix-sum algorithm for unidirectional attention. For clarity, we omit attention normalization in this visualization. The algorithm keeps the prefix-sum which is a matrix obtained by summing the outer products of random features corresponding to keys with value-vectors. At each given iteration of the prefix-sum algorithm, a random feature vector corresponding to a query is multiplied by the most recent prefix-sum (obtained by summing all outer-products corresponding to preceding tokens) to obtain a new row of the matrix $\mathbf{AV}$ which is output by the attention mechanism.

чуть сложнее если хотим с масками

Performer

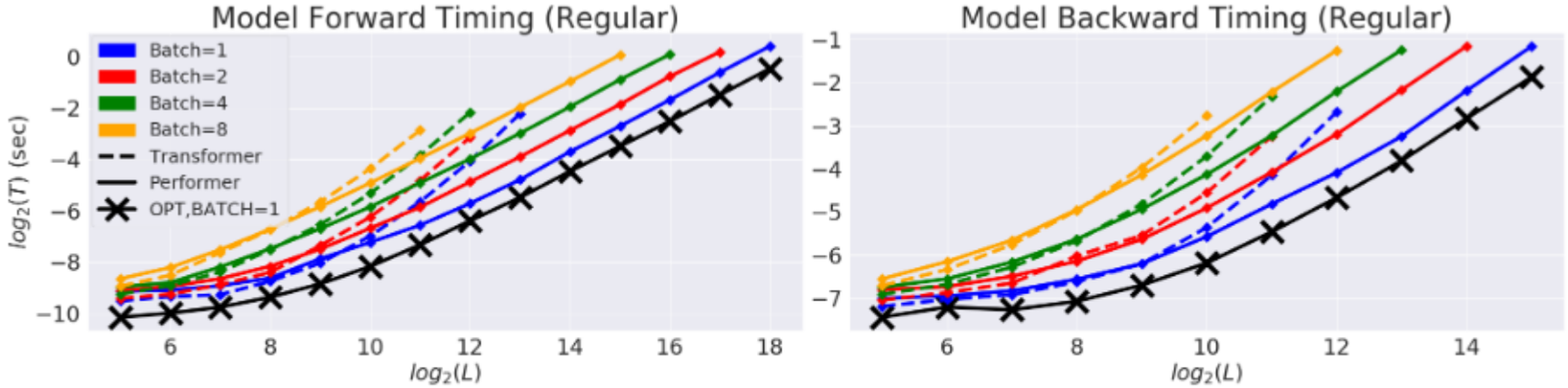


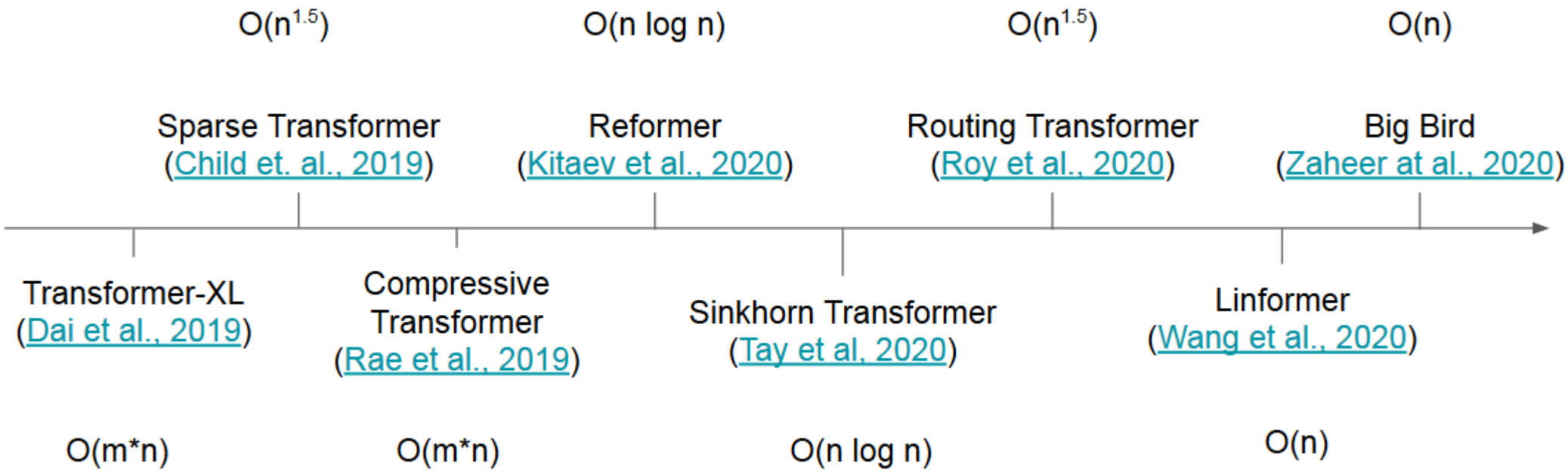
Figure 3: Comparison of Transformer and Performer in terms of forward and backward pass speed and maximum L allowed. "X" (OPT) denotes the maximum possible speedup achievable, when attention simply returns the $\mathbf{V}$-matrix. Plots shown up to when a model produces an out of memory error on a V100 GPU with 16GB. Vocabulary size used was 256. Best in color.

работают так быстро как будто совсем без внимания

Черная линия – максимально возможное сжатие памяти и ускорение при использовании «фиктивного» блока внимания

Применения трансформера

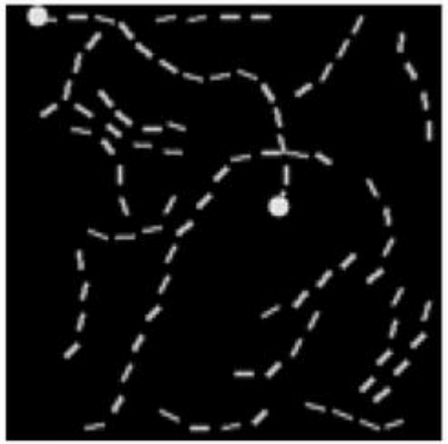
2017, Vaswani et al.	**Transformer**	**Machine translation**
2018, Devlin et al.	**BERT**	**NLU**
2019, Lewis et al.	**BART**	**NLG**
2020, Dosovitskiy, et al.	**Vision Transformer**	**Image Classification**
2020, DeepMind	**AlphaFold 2**	**Protein folding**



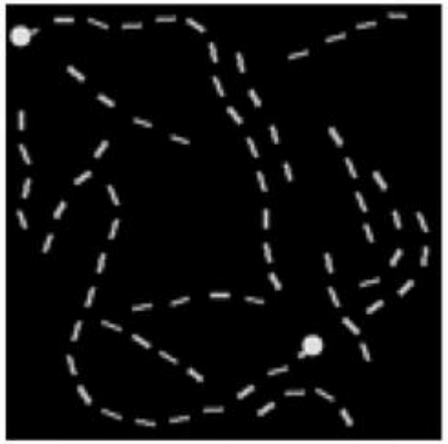
Сравнение на Long Range Arena

примеры задач

соединены ли точки



(a) A positive example.



(b) A negative example.

ответ выражения

INPUT: [MAX 4 3 [MIN 2 3] 1 0 [MEDIAN 1 5 8 9, 2]]

OUTPUT: 5

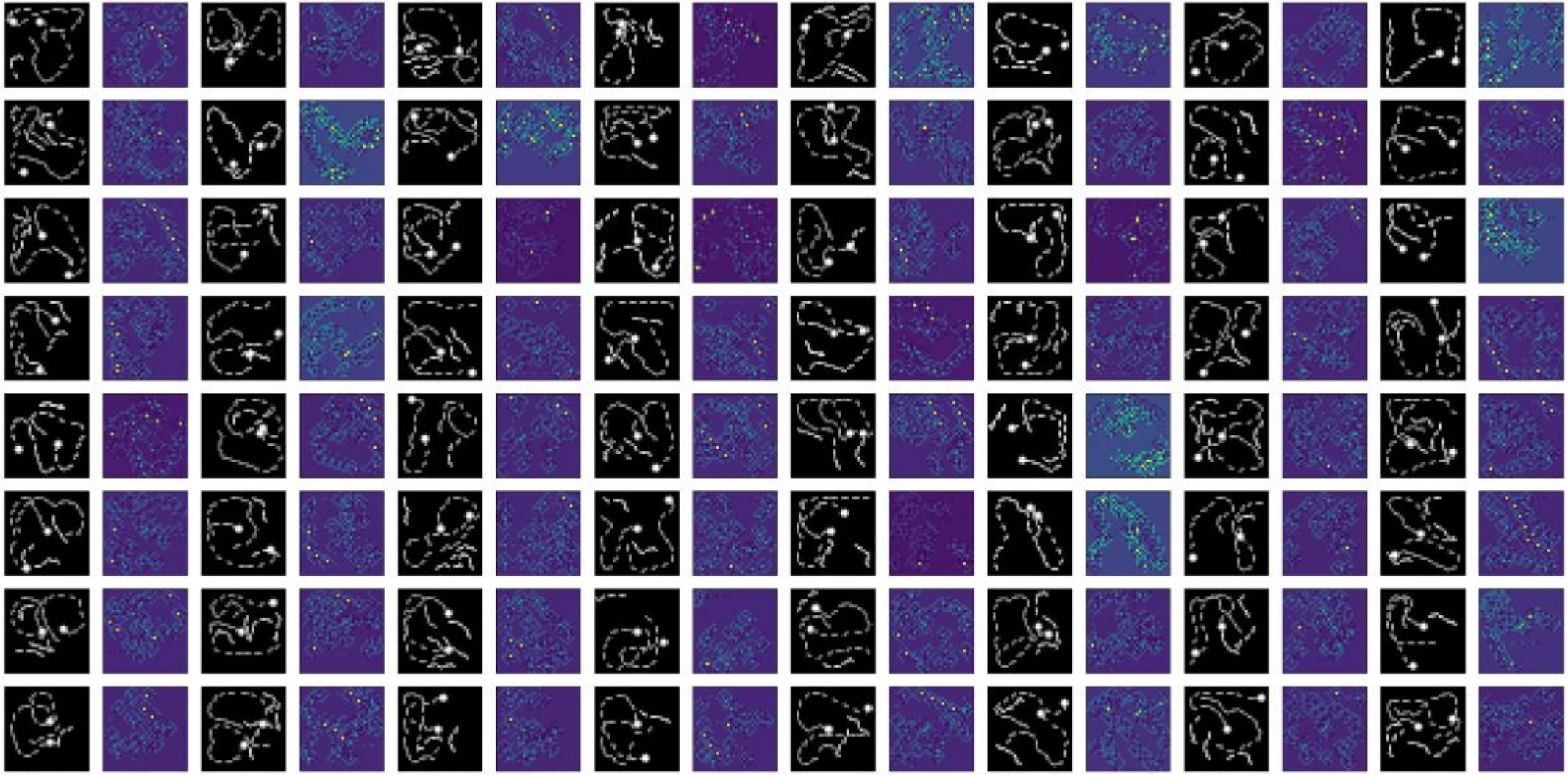


Figure 5: Attention map for different examples from the Pathfinder task. Each map presents the attention distribution, given the CLS token at the final layer as the query, averaged across all heads in a vanilla Transformer model. Note that for visualization, we use attention-rollout (Abnar & Zuidema, 2020) for more precise input attribution.

Сравнение на Long Range Arena

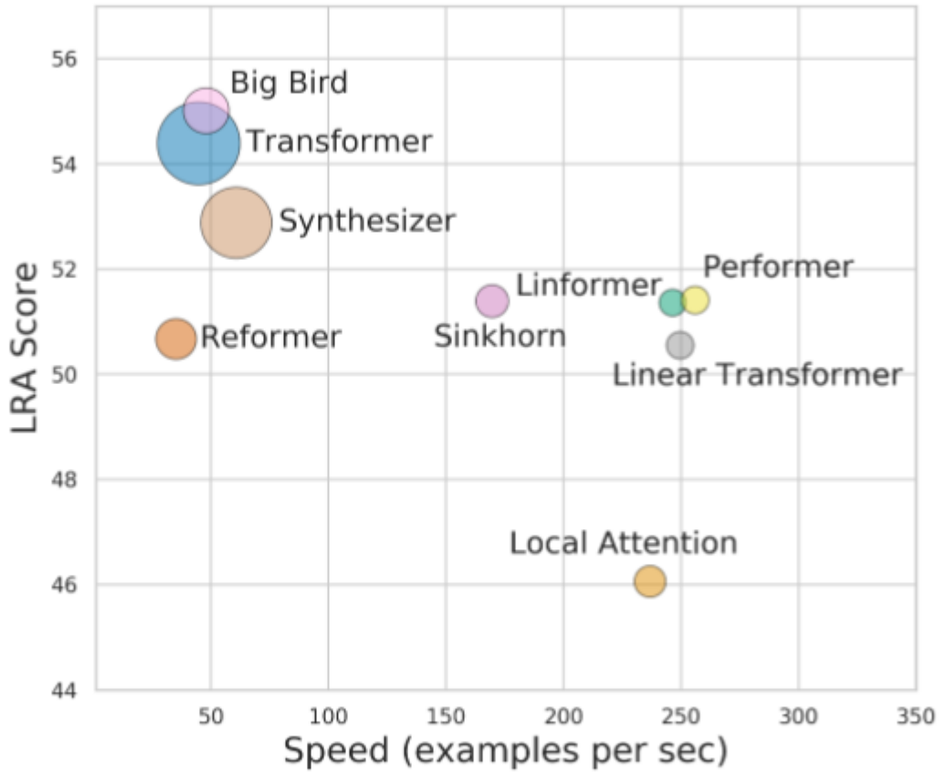


Figure 3: Performance (y axis), speed (x axis), and memory footprint (size of the circles)

Размеры трансформеров

Year	Model	# of Parameters	Dataset Size
2019	BERT [39]	3.4E+08	16GB
2019	DistilBERT [113]	6.60E+07	16GB
2019	ALBERT [70]	2.23E+08	16GB
2019	XLNet (Large) [150]	3.40E+08	126GB
2020	ERNIE-GEN (Large) [145]	3.40E+08	16GB
2019	RoBERTa (Large) [74]	3.55E+08	161GB
2019	MegatronLM [122]	8.30E+09	174GB
2020	T5-11B [107]	1.10E+10	745GB
2020	T-NLG [112]	1.70E+10	174GB
2020	GPT-3 [25]	1.75E+11	570GB
2020	GShard [73]	6.00E+11	–
2021	Switch-C [43]	1.57E+12	745GB

Table 1: Overview of recent large language models

http://faculty.washington.edu/ebender/papers/Stochastic_Parrots.pdf

Размеры трансформеров: из доклада Г. Сапунова (в 2021 году)

(English) GPT-Neo (2.7B), GPT-J (6B),
Jurassic-1 (7.5B/178B)
(Russian) ruGPT-3 (13B)
(Chinese) CPM-2 (11B/198B* - MoE),
M6 (10B/100B), Wu Dao 2.0 (1.75T*),
PangGu-α (2.6B/13B/207B)
(Korean) HyperCLOVA (204B)
(Code) OpenAI Codex (12B),
Google’s (up to 137B)
ByT5 (up to 12.9B)
XLM-R XL/XXL (3.5B/10.7B)
DeBERTa (1.5B)
Switch Transformer (1.6T*)
ERNIE 3.0 (10B)
DALL·E (12B)
Vision MoE (14.7B*)

Степенные законы

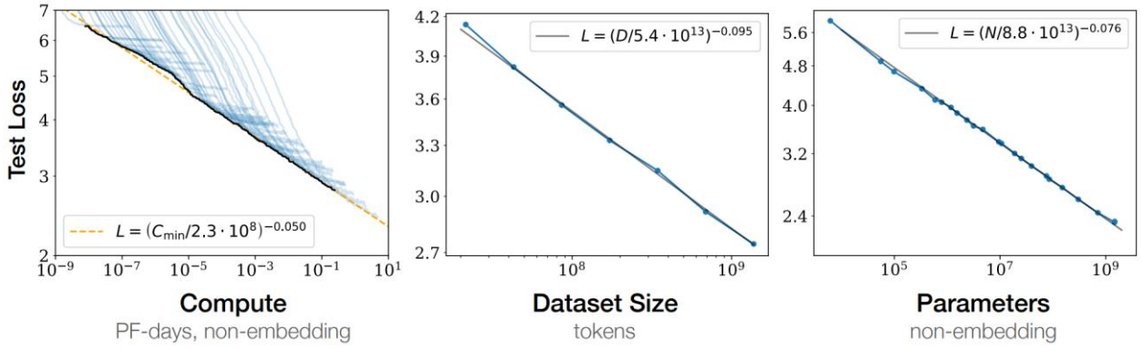


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

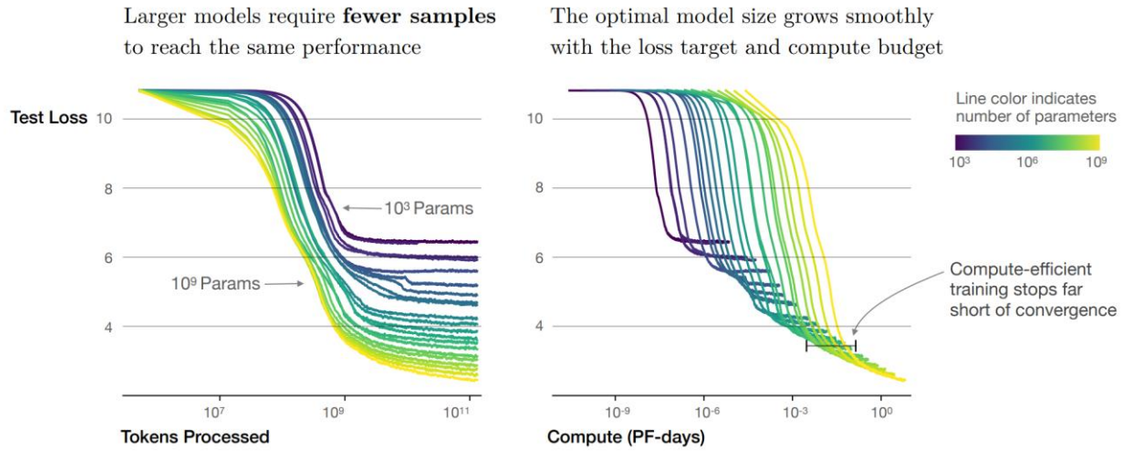


Figure 2 We show a series of language model training runs, with models ranging in size from 10^3 to 10^9 parameters (excluding embeddings).

Большим трансформерам и выборка нужна меньше для достижения некоторого качества (что неинтуитивно) «Scaling Laws for Neural Language Models» https://arxiv.org/abs/2001.08361

Стоимость обучения

Just how much does it cost to train a model? Two correct answers are “depends” and “a lot”. More quantitatively, here are current ballpark list-price costs of training differently sized BERT [4] models on the Wikipedia and Book corpora (15 GB). For each setting we report two numbers - the cost of one training run, and a typical fully-loaded cost (see discussion of "hidden costs" below) with hyper-parameter tuning and multiple runs per setting (here we look at a somewhat modest upper bound of two configurations and ten runs per configuration).[4]

- $2.5k - $50k (110 million parameter model)
- $10k - $200k (340 million parameter model)
- $80k - $1.6m (1.5 billion parameter model)

«The Cost of Training NLP Models: A Concise Overview» // https://arxiv.org/abs/2004.08900

Выделение CO2

Consumption	CO_2e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

«Energy and Policy Considerations for Deep Learning in NLP» //
https://arxiv.org/abs/1906.02243

Scaling Transformer/Terraformer

«Sparse is Enough in Scaling Transformers» // https://arxiv.org/abs/2111.12763

XLNet: Generalized Autoregressive Pretraining for Language Understanding

авторегрессионная языковая модель с перестановками

есть память

относительное позициональное кодирование

двухпотоковый механизм self-attention

один анализирует номера позиций, второй векторы представления

512 TPU 3 дня

Zhilin Yang «XLNet: Generalized Autoregressive Pretraining for Language Understanding» https://arxiv.org/pdf/1906.08237.pdf

XLNet: Generalized Autoregressive Pretraining for Language Understanding

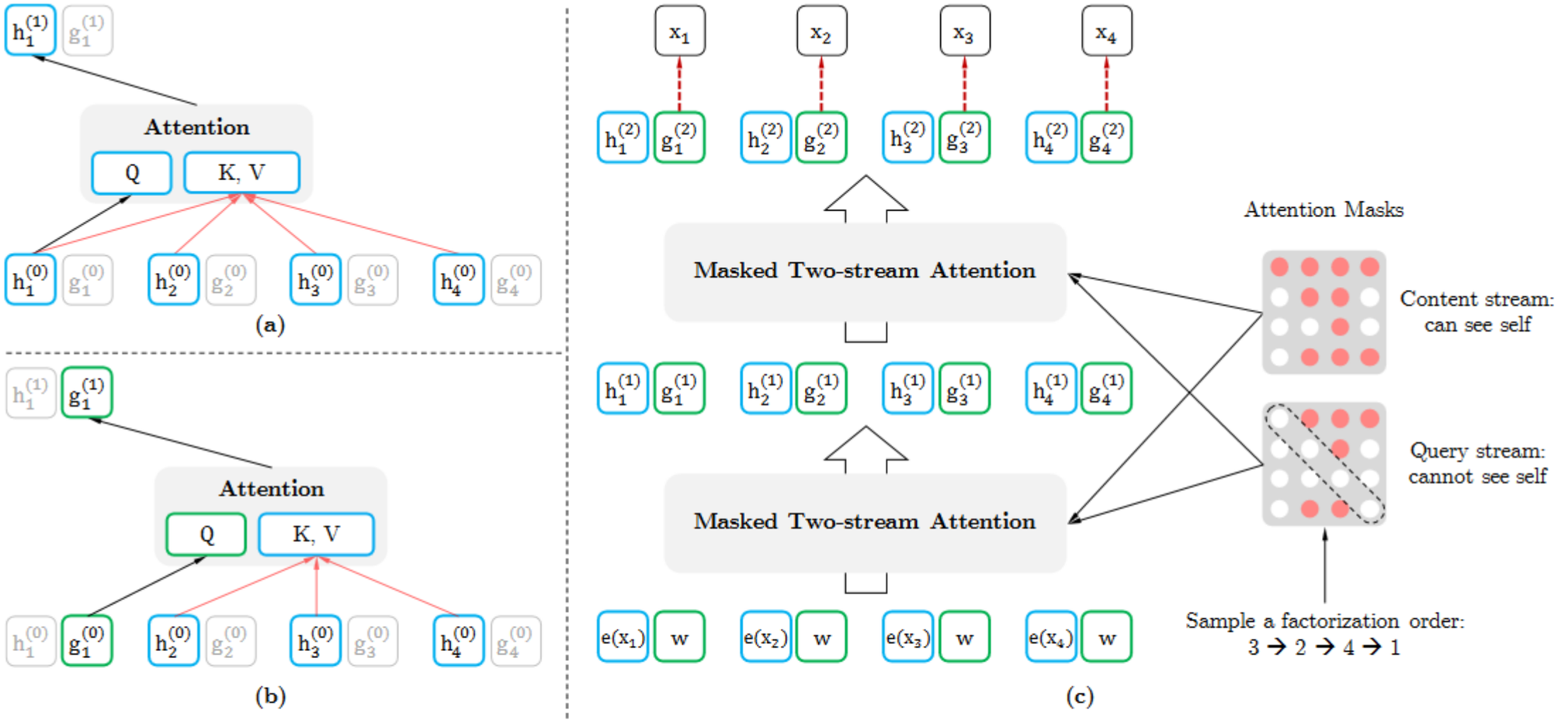


Figure 1: (a): Content stream attention, which is the same as the standard self-attention. (b): Query stream attention, which does not have access information about the content x_{z_t}. (c): Overview of the permutation language modeling training with two-stream attention.

XLNet: Generalized Autoregressive Pretraining for Language Understanding

	BERT	RoBERTa	DistilBERT	XLNet
Size (millions)	**Base**: 110 **Large**: 340	**Base**: 110 **Large**: 340	**Base:** 66	**Base**: ~110 **Large**: ~340
Training Time	**Base**: 8 x V100 x 12 days* **Large:** 64 TPU Chips x 4 days (or 280 x V100 x 1 days*)	**Large:** 1024 x V100 x 1 day; 4-5 times more than BERT.	**Base**: 8 x V100 x 3.5 days; 4 times less than BERT.	**Large:** 512 TPU Chips x 2.5 days; 5 times more than BERT.
Performance	Outperforms state-of-the-art in Oct 2018	2-20% improvement over BERT	3% degradation from BERT	2-15% improvement over BERT
Data	16 GB BERT data (Books Corpus + Wikipedia). 3.3 Billion words.	160 GB (16 GB BERT data + 144 GB additional)	16 GB BERT data. 3.3 Billion words.	**Base**: 16 GB BERT data **Large**: 113 GB (16 GB BERT data + 97 GB additional). 33 Billion words.
Method	BERT (Bidirectional Transformer with MLM and NSP)	BERT without NSP**	BERT Distillation	Bidirectional Transformer with Permutation based modeling

Ссылки

Лекция базировалась на

Григорий Сапунов, лекция «Transformer Zoo (a deeper dive)»
https://www.youtube.com/watch?v=KZ9NXYcXVBY
Трансформеры в 2021 году
https://www.youtube.com/watch?v=8dN6ZVnDArk

хороший обзор
Yi Tay et al «Efficient Transformers: A Survey» // https://arxiv.org/pdf/2009.06732.pdf

см. сравнение
Yi Tay, et al. «Long Range Arena: A Benchmark for Efficient Transformers» //
https://arxiv.org/abs/2011.04006

Synthesizer models (Tay et al., 2020a)
использовал доклад Vladislav Lialin «Applying Transformer Models to Long Texts: Challenges and Solutions»