The Transformer Family Version 2.0

Марьин Геннадий

Transformers - how to make them better?

Notation:

$$X \in \mathbb{R}^{L \times d}$$

$$Q = XW^q \in \mathbb{R}^{L \times d_k} \quad K = XW^k \in \mathbb{R}^{L \times d_k} \quad V = XW^v \in \mathbb{R}^{L \times d_v}$$

$$W^k \in \mathbb{R}^{d \times d_k} \qquad W_i^k, W_i^q \in \mathbb{R}^{d \times \frac{d_k}{h}}$$

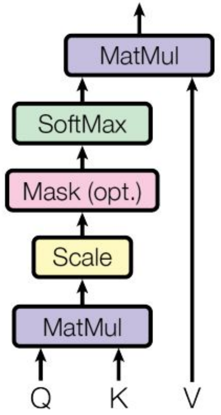
$$W^q \in \mathbb{R}^{d \times d_k} \qquad W_i^v \in \mathbb{R}^{d \times \frac{d_v}{h}}$$

$$W^v \in \mathbb{R}^{d \times d_v} \qquad W^0 \in \mathbb{R}^{d_v \times d}$$

Transformer Basics

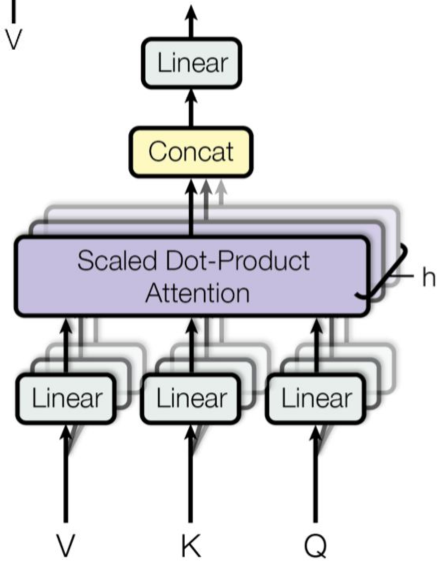
Self-Attention:

$$attn(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$



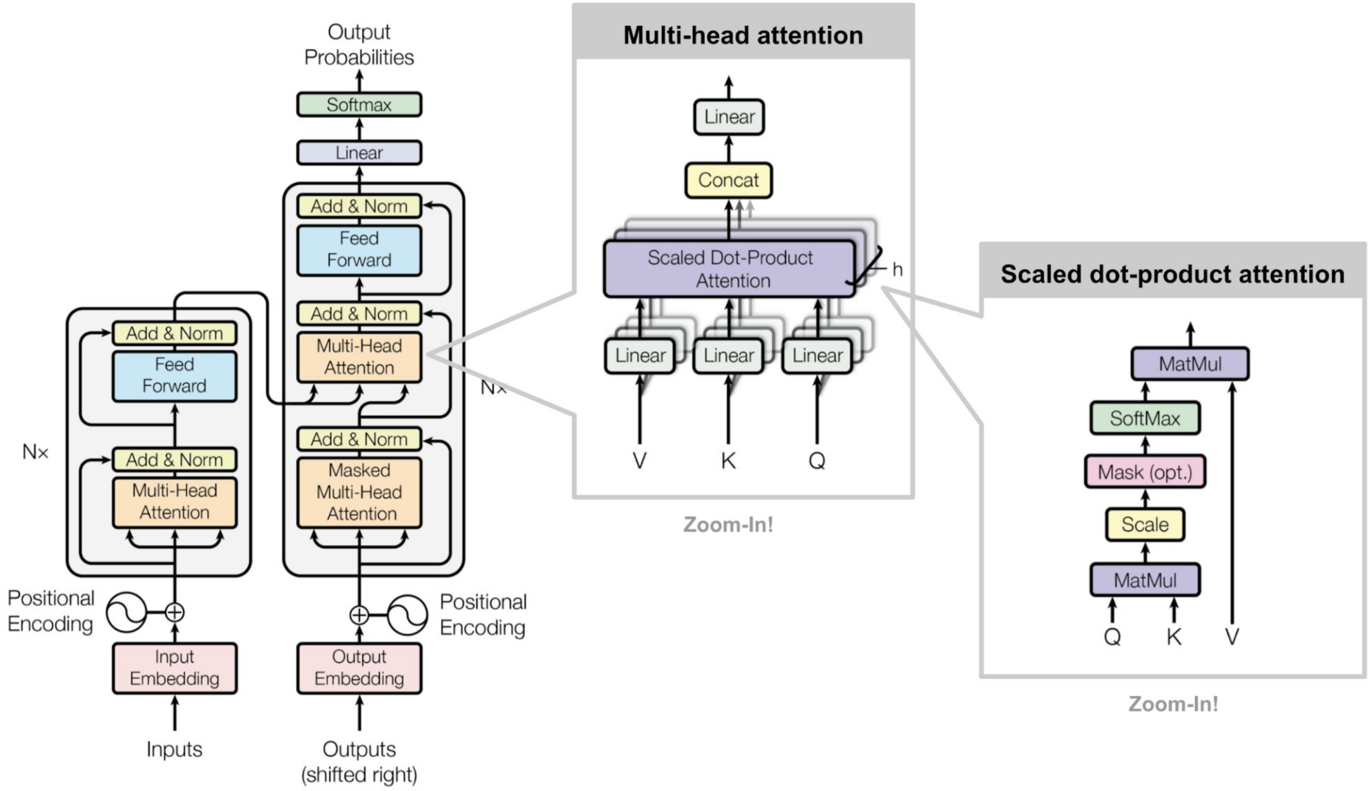
Multi-Head Self-Attention:

$$MultiHeadAttn(X_q, X_k, X_v) = [head_1, ..., head_h]W^0$$
$$head_i = attn(X_q W_i^q, X_k W_i^k, X_v W_i^v)$$



Transformer Basics

Encoder-Decoder Architecture:



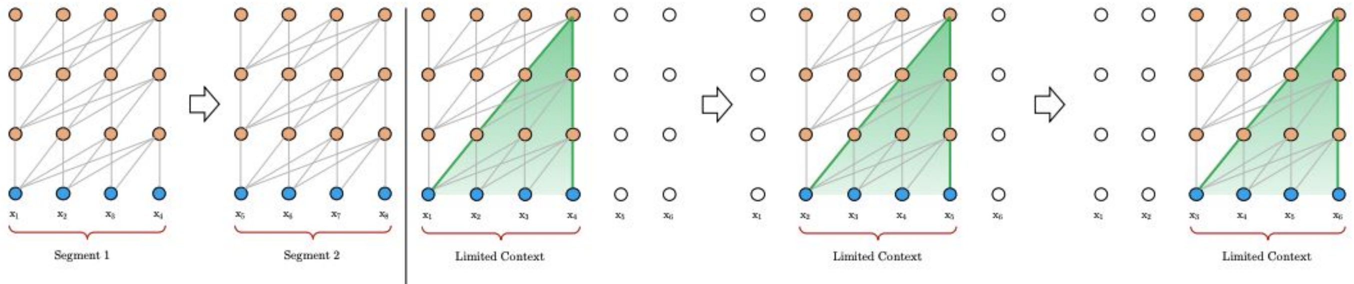
Longer Context:

Problem: the length of an input sequence for transformer models at inference time is upper-bounded by the context length used for training. Naively increasing context length leads to high consumption in both time $\mathcal{O}(L^2d)$ and memory $\mathcal{O}(L^2)$ and may not be supported due to hardware constraints

Context Memory

The vanilla Transformer has a fixed and limited attention span. The model can only attend to other elements in the same segments during each update step and no information can flow across separated fixed-length segments. This context segmentation causes several issues:

- The model cannot capture very long term dependencies.
- It is hard to predict the first few tokens in each segment given no or thin context.
- The evaluation is expensive. Whenever the segment is shifted to the right by one, the new segment is re-processed from scratch, although there are a lot of overlapped tokens.



(a) Train phase. (b) Evaluation phase.

Context Memory: Transformer XL

Idea: reuse hidden states between segments with an additional memory; the recurrent connection between segments is introduced into the model by continuously using the hidden states from the previous segments.

$$\tilde{h}^{n-1}_{\tau+1} = [StopGradient(h^{n-1}_{\tau}), h^{n-1}_{\tau+1}] \in \mathbb{R}^{2L\times d}$$

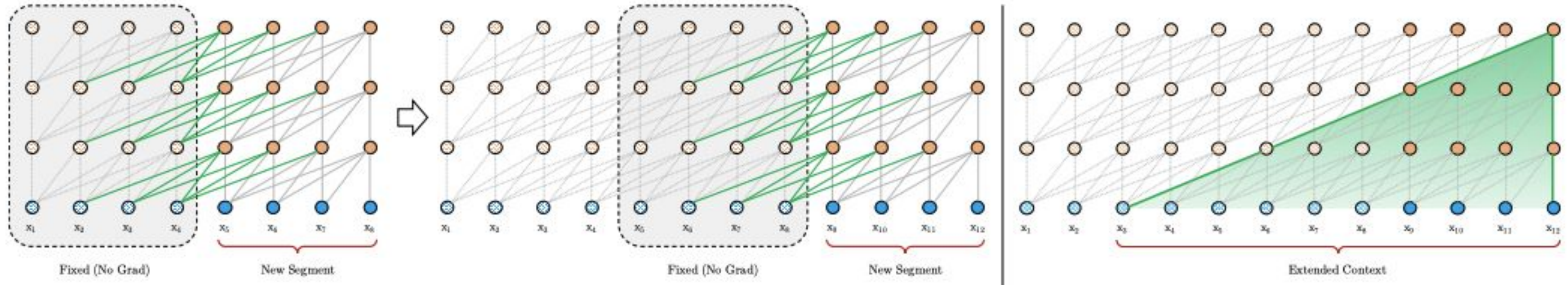
$$Q^{(n)}_{\tau+1} = h^{(n-1)}_{\tau+1} W^q$$

$$K^{(n)}_{\tau+1} = \tilde{h}^{(n-1)}_{\tau+1} W^k$$

$$V^{(n)}_{\tau+1} = \tilde{h}^{(n-1)}_{\tau+1} W^v$$

$$h^{(n)}_{\tau+1} = TransformerLayer(Q^{(n)}_{\tau+1}, K^{(n)}_{\tau+1}, V^{(n)}_{\tau+1})$$

$h^{(n)}_{\tau} \in \mathbb{R}^{L\times d}$ n-th layer hidden state sequence produced for the τ-th segment


x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
Fixed (No Grad)
New Segment
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}
Fixed (No Grad)
New Segment
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}
Extended Context

(a) Training phase. (b) Evaluation phase.

Note: it needs relative positional encoding! (absolute encoding doesn't work)

Context Memory: Compressive Transformer

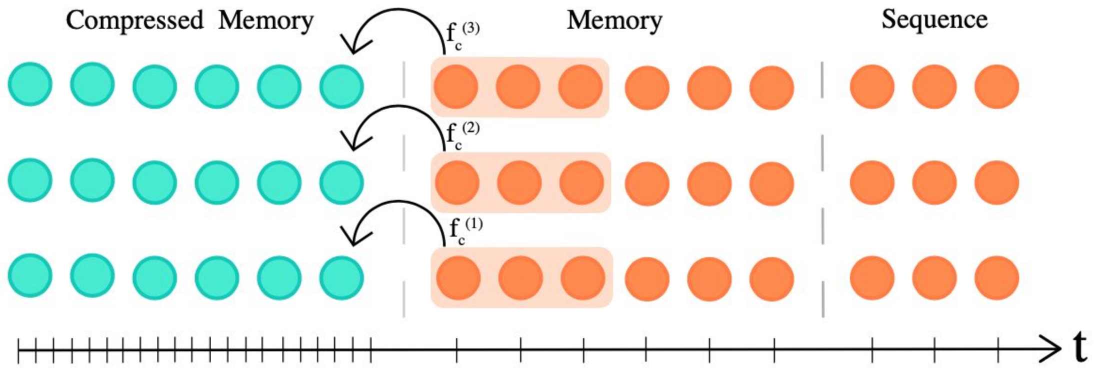
Idea: extends Transformer-XL by compressing past memories to support longer sequences. It explicitly adds memory slots of size m_m per layer for storing past activations of this layer to preserve long context. When some past activations become old enough, they are compressed and saved in an additional compressed memory of size m_{cm} per layer.

Both memory and compressed memory are FIFO queues.

Given the model context length L, the compression function of compression rate c

$$f_c : \mathbb{R}^{L\times d} \rightarrow \mathbb{R}^{[\frac{L}{c}]\times d}$$

compression function of compression rate c



Context Memory: Compressive Transformer

Auto-encoding loss (lossless compression objective) measures how well we can reconstruct the original memories from compressed memories:

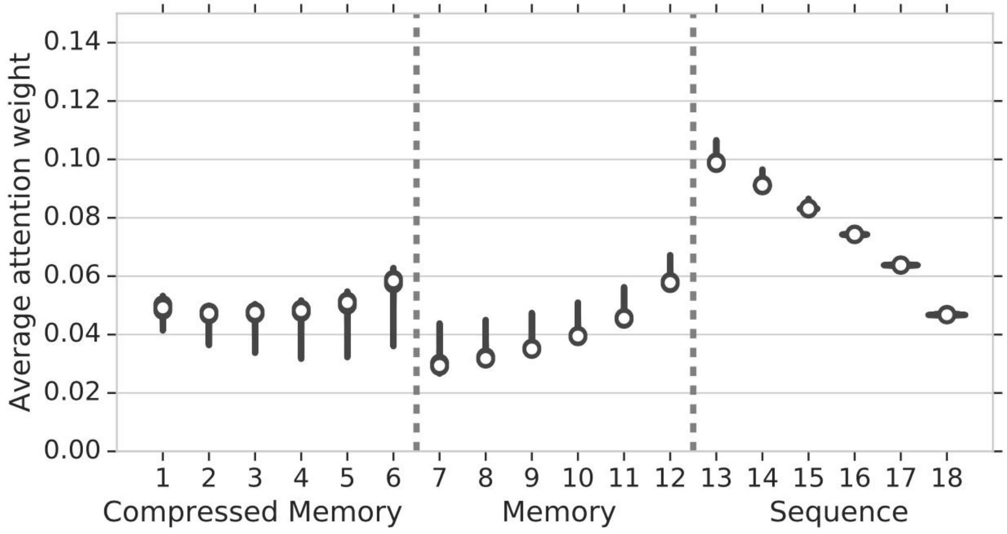
$$L_{ac} = \left\| old\_mem^{(i)} - g(new\_cm^{(i)}) \right\|_2, \quad g : \mathbb{R}^{[\frac{L}{c}]\times d} -> \mathbb{R}^{L\times d}$$ reverses the compression function

Attention-reconstruction loss (lossy objective) reconstructs content-based attention over memory vs compressed memory and minimize the difference:

$$L_{ar} = \left\| attn(h^{(i)}, old\_mem^{(i)}) - attn(h^{(i)}, new\_cm^{(i)}) \right\|_2$$

Context Memory: Summary

Transformer-XL with a memory of size m has a maximum temporal range of $m \times N$, where N is the number of layers in the model, and attention cost $\mathcal{O}(L^2 + Lm)$. In comparison, compressed transformer has a temporal range of $(m_m + m_{cm}) \times N$ and attention cost $\mathcal{O}(L^2 + L(m_m + m_{cm}))$. A larger compression rate c gives better tradeoff between temporal range length and attention cost.



Non-Differentiable External Memory: kNN-LM

Idea: enhances a pretrained LM with a separate kNN model by linearly interpolating the next token probabilities predicted by both models. The kNN model is built upon an external key-value store which can store any large pre-training dataset or OOD new dataset. This datastore is preprocessed to save a large number of pairs, (LM embedding representation of context, next token) and the nearest neighbor retrieval happens in the LM embedding space.

At inference time, the next token probability is a weighted sum of two predictions:

$$p(y \mid x) = \lambda p_{kNN}(y \mid x) + (1 - \lambda) p_{LM}(y \mid x)$$

$$p_{kNN}(y \mid x) \propto \sum_{(k_i, w_i) \in N} \mathbb{I}[y = w_i] exp(-d(k_i, f(x)))$$

where N contains a set of nearest neighbor data points retrieved by kNN; d() is a distance function, f - context representation (e.g. feed forward)

Non-Differentiable External Memory: kNN-LM

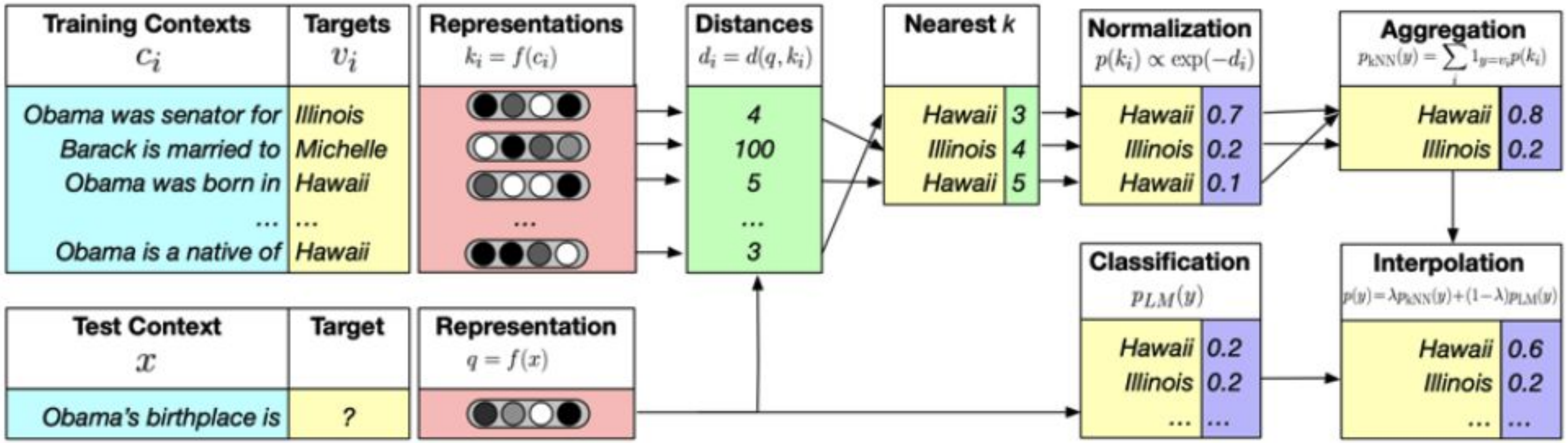

Training Contexts c_i
Targets v_i
Representations $k_i = f(c_i)$
Distances $d_i = d(q, k_i)$
Nearest k
Normalization $p(k_i) \propto \exp(-d_i)$
Aggregation $p_{kNN}(y) = \sum_i 1_{y=v_i} p(k_i)$
Obama was senator for | Illinois
Barack is married to | Michelle
Obama was born in | Hawaii
... | ...
Obama is a native of | Hawaii
4
100
5
...
3
Hawaii 3
Illinois 4
Hawaii 5
Hawaii 0.7
Illinois 0.2
Hawaii 0.1
Hawaii 0.8
Illinois 0.2
Test Context x
Target
Representation $q = f(x)$
Obama's birthplace is | ?
Classification $p_{LM}(y)$
Hawaii 0.2
Illinois 0.2
... ...
Interpolation $p(y) = \lambda p_{kNN}(y) + (1-\lambda) p_{LM}(y)$
Hawaii 0.6
Illinois 0.2
... ...

Figure 1: An illustration of kNN-LM. A datastore is constructed with an entry for each training set token, and an encoding of its leftward context. For inference, a test context is encoded, and the k most similar training contexts are retrieved from the datastore, along with the corresponding targets. A distribution over targets is computed based on the distance of the corresponding context from the test context. This distribution is then interpolated with the original model's output distribution.

Non-Differentiable External Memory: SPALM

Idea: incorporates both (1) Transformer-XL style memory for hidden states from external context as short-term memory and (2) kNN-LM style key-value store as long memory.

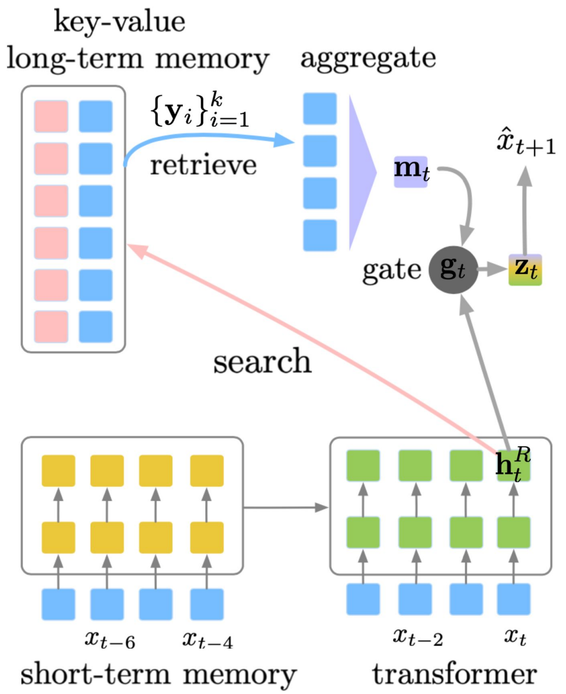
SPALM runs kNN search to fetch k tokens with most relevant context. For each token we can get the same embedding representation provided by a pretrained LM, denoted as y. The gating mechanism first aggregates the retrieved token embeddings with a simple attention layer using h^R_t (the hidden state for token x_t at layer R) as a query and then learns a gating parameter g_t to balance between local information h^R_t and long-term information m_t.

$$m_t = \sum_{i=1}^{k} \frac{exp(y_i^T h_t^R)}{\sum_{j=1}^{k} exp(y_j^T h_t^R)} \cdot y_i$$

$$g_t = \sigma(w_g^T h_t^R)$$

$$z_t = (1 - g_t) \odot m_t + g_t \odot h_t^R$$

$$p(x_{t+1} \mid x_{\le t}) = softmax(z_t; W)$$



Non-Differentiable External Memory: SPALM

Idea: incorporates both (1) Transformer-XL style memory for hidden states from external context as short-term memory and (2) kNN-LM style key-value store as long memory.

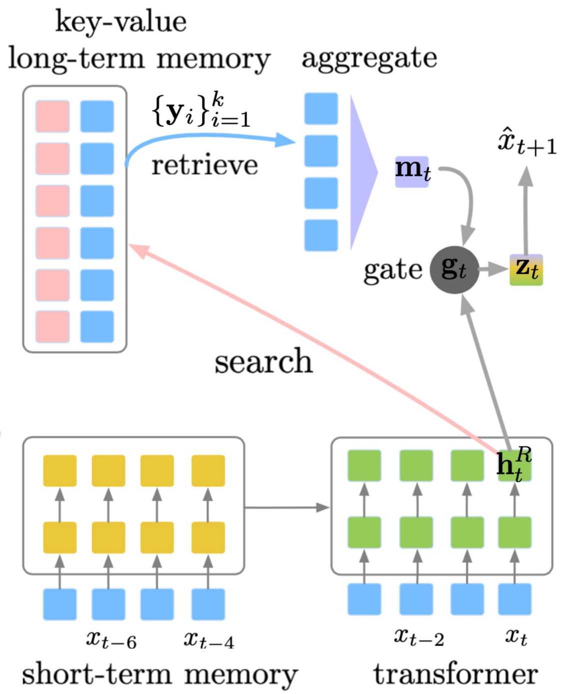
$$m_t = \sum_{i=1}^{k} \frac{exp(y_i^T h_t^R)}{\sum_{j=1}^{k} exp(y_j^T h_t^R)} \cdot y_i$$

$$g_t = \sigma(w_g^T h_t^R)$$

$$z_t = (1 - g_t) \odot m_t + g_t \odot h_t^R$$

$$p(x_{t+1} \mid x_{\leq t}) = softmax(z_t; W)$$

where w_g is a parameter vector to learn; σ() is sigmoid; W is the word embedding matrix shared between both input and output tokens. Different from kNN-LM, they didn't find the nearest neighbor distance to be helpful in the aggregation of retrieved tokens.

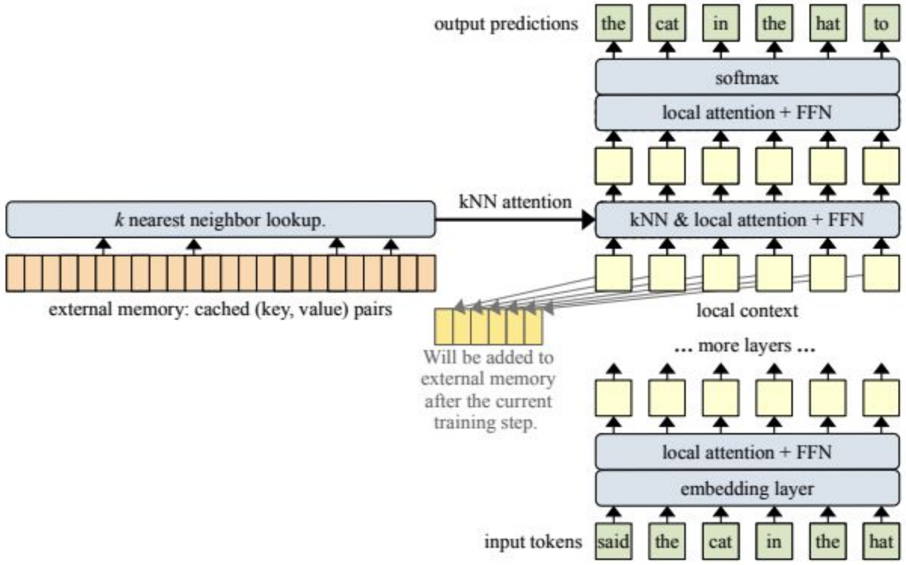


During training, the key representations in the long-term memory stay constant, produced by a pretrained LM, but the value encoder, a.k.a. the word embedding matrix, gets updated.

Non-Differentiable External Memory: Memorizing Transformer

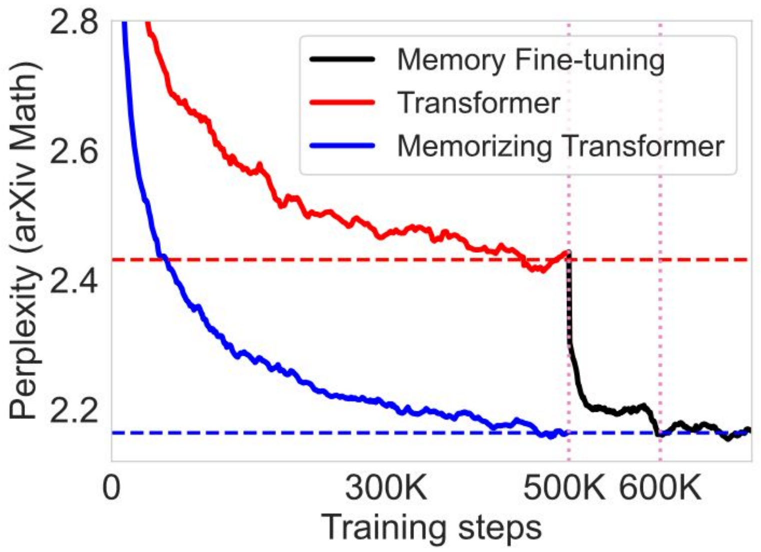
Idea: adds a kNN-augmented attention layer near the top stack of a decoder-only Transformer. This special layer maintains a Transformer-XL style FIFO cache of past key-value pairs.

The same Q,K,V values are used for both local attention and kNN mechanisms. The kNN lookup returns top-k (key, value) pairs for each query in the input sequence and then they are processed through the self-attention stack to compute a weighted average of retrieved values. **Two types of attention** are combined with a learnable per-head gating parameter. To prevent large distributional shifts in value magnitude, both keys and values in the cache are normalized.



Non-Differentiable External Memory: Memorizing Transformer

- It is observed in some experiments that training models with a small memory and then finetuned with a larger memory works better than training with a large memory from scratch.
- The smaller Memorizing Transformer with just 8k tokens in memory can match the perplexity of a larger vanilla Transformer with 5X more trainable parameters.
- Increasing the size of external memory provided consistent gains up to a size of 262K.
- A non-memory transformer can be finetuned to use memory.



Distance-Enhanced Attention Scores

Idea: in order to encourage the model to extrapolate over longer context than what the model is trained on, we can explicitly attach the positional information to every pair of attention score based on the *distance between key and query tokens*.

Distance-Enhanced Attention Scores: DA-Transformer

DA-Transformer: multiplies attention scores at each layer by a learnable bias that is formulated as a function of the distance between key and query. Different attention heads use different parameters to distinguish diverse preferences to short-term vs long-term context.

$$\mathbf{R}^{(i)} = \alpha_i \mathbf{R} \quad \text{where } R_{ij} = |i-j|$$

α_i - learnable parameter to weight relative distance

$$f(\mathbf{R}^{(i)}; \beta_i) = \frac{1+\exp(\beta_i)}{1+\exp(\beta_i - \mathbf{R}^{(i)})}$$

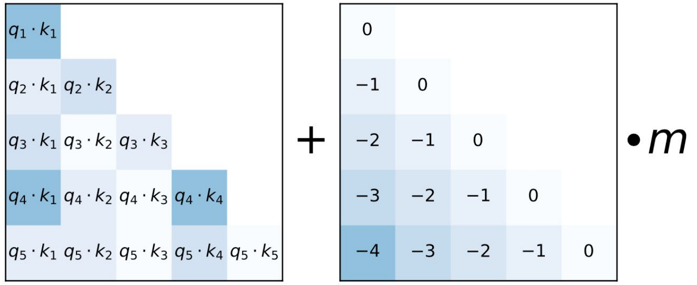
b_i - learnable parameter to control upper bound and ascending slope wrt distance

$$\text{attn}(\mathbf{Q}^{(i)}, \mathbf{K}^{(i)}, \mathbf{V}^{(i)}) = \text{row-softmax}\left(\frac{\text{ReLU}(\mathbf{Q}^{(i)}\mathbf{K}^{(i)\top}) f(\mathbf{R}^{(i)})}{\sqrt{d}}\right)\mathbf{V}^{(i)}$$

- Extra time complexity brought by $f(R^{(i)})$ is $\mathcal{O}(L^2)$ and it is small relative to the self attention time complexity $\mathcal{O}(L^2 d)$.
- The extra memory consumption is minimal, ~$\mathcal{O}(2h)$

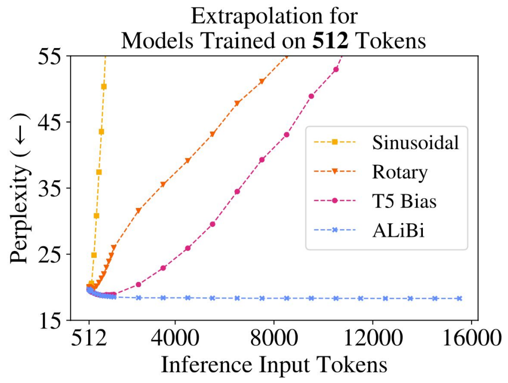
Distance-Enhanced Attention Scores: ALiBi

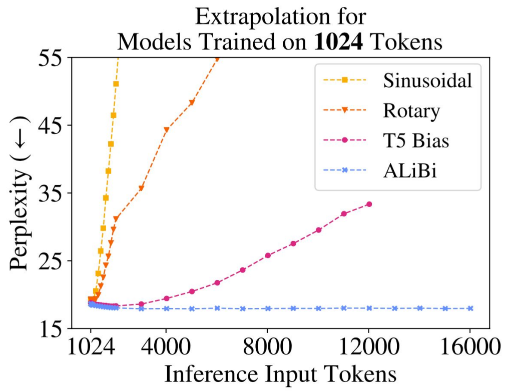
ALiBi: adds a constant bias term on query-key attention scores, proportional to pairwise distances. The bias introduces a strong recency preference and penalizes keys that are too far away. The penalties are increased at different rates within different heads.


$q_1 \cdot k_1$
$q_2 \cdot k_1$ $q_2 \cdot k_2$
$q_3 \cdot k_1$ $q_3 \cdot k_2$ $q_3 \cdot k_3$
$q_4 \cdot k_1$ $q_4 \cdot k_2$ $q_4 \cdot k_3$ $q_4 \cdot k_4$
$q_5 \cdot k_1$ $q_5 \cdot k_2$ $q_5 \cdot k_3$ $q_5 \cdot k_4$ $q_5 \cdot k_5$
+
0
−1 0
−2 −1 0
−3 −2 −1 0
−4 −3 −2 −1 0
$\bullet\, m$

$$\mathrm{softmax}(\mathbf{q}_i \mathbf{K}^\top + \alpha_i \cdot [0, -1, -2, \ldots, -(i-1)])$$

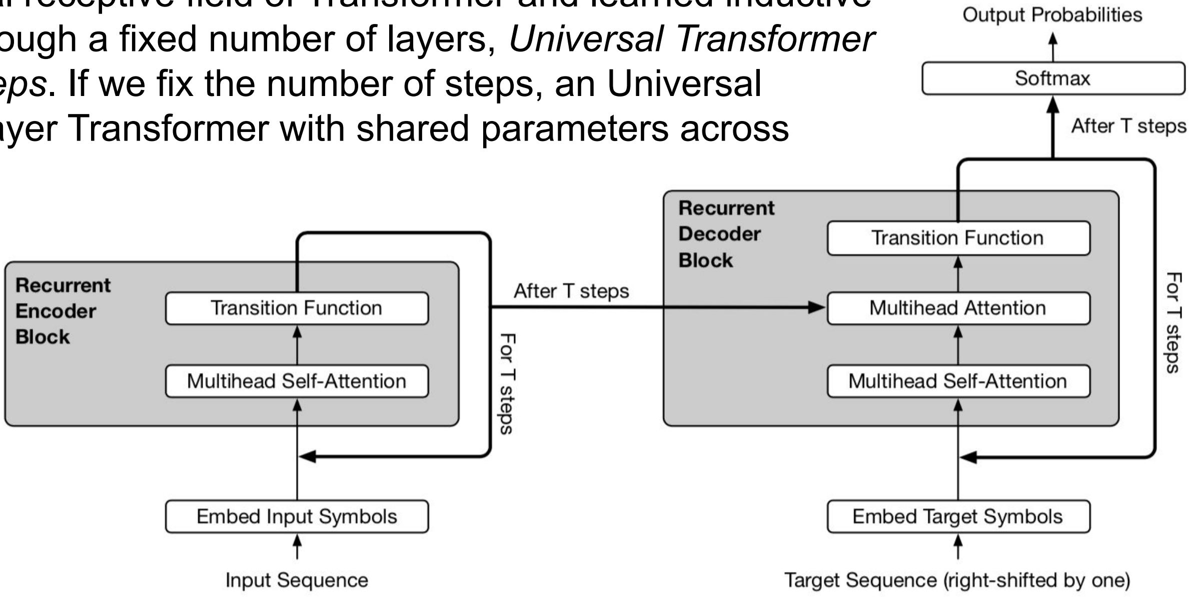
α_i (m) - head-specific weighting scalar;
differently from DA-transformer, α_i is not learned but fixed as a geometric sequence





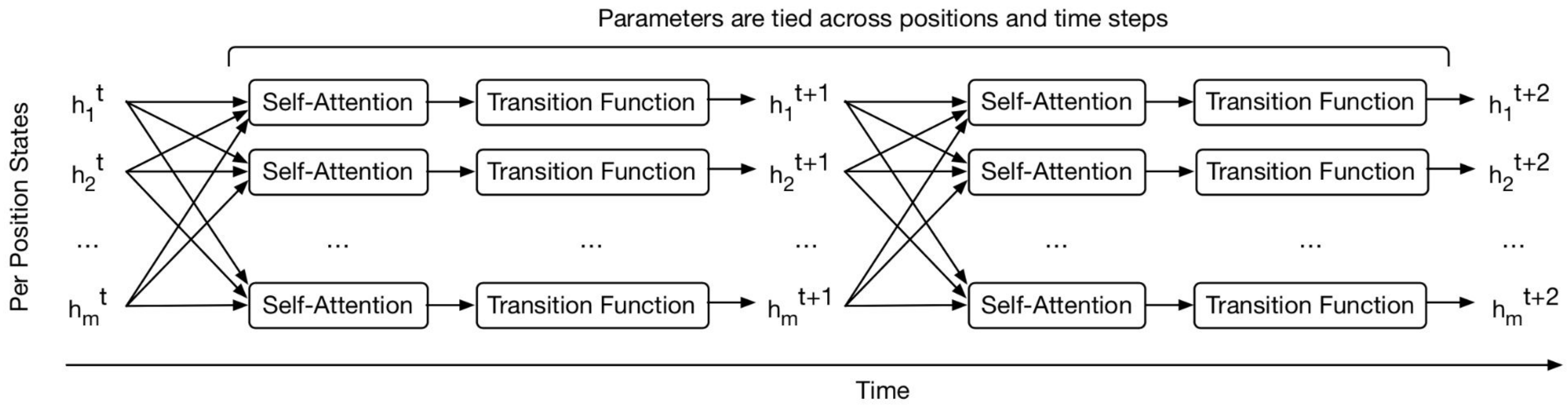
Make it Recurrent: Universal Transformer

Idea: combines self-attention in Transformer with the recurrent mechanism in RNN, aiming to benefit from both a long-term global receptive field of Transformer and learned inductive biases of RNN. Rather than going through a fixed number of layers, *Universal Transformer dynamically adjusts the number of steps*. If we fix the number of steps, an Universal Transformer is equivalent to a multi-layer Transformer with shared parameters across layers.



Make it Recurrent: Universal Transformer

On a high level, the universal transformer can be viewed as a recurrent function for learning the hidden state representation per token. **The recurrent function evolves in parallel across token positions and the information between positions is shared through self-attention.**

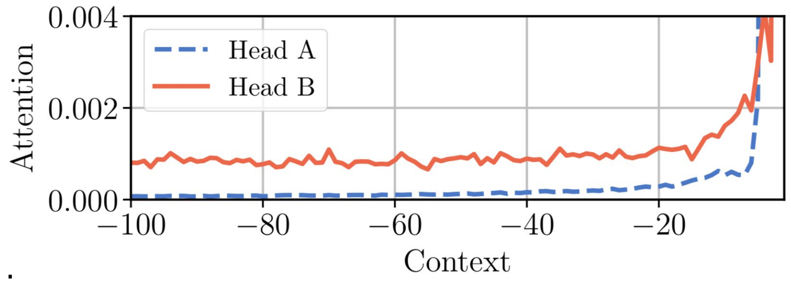


Adaptive modeling

Problem: Adaptive modeling refers to a mechanism that can adjust the amount of computation according to different inputs. For example, some tokens may only need local information and thus demand a shorter attention span; Or some tokens are relatively easier to predict and do not need to be processed through the entire attention stack.

Adaptive modeling: Adaptive Attention Span

Motivation: one key advantage of Transformer is the capability of capturing long-term dependencies. Depending on the context, the model may prefer to attend further sometime than others; or one attention head may had different attention pattern from the other. If the attention span could adapt its length flexibly and only attend further back when needed, it would help reduce both computation and memory cost to support longer maximum context size in the model.



Idea: proposed a self-attention mechanism that seeks an optimal attention span.

Results:

- Found a general tendency that lower layers do not require very long attention spans, while a few attention heads in higher layers may use exceptionally long spans.
- Adaptive attention span also helps greatly reduce the number of FLOPS, especially in a big model with many attention layers and a large context length.

Adaptive modeling: Depth-Adaptive Transformer

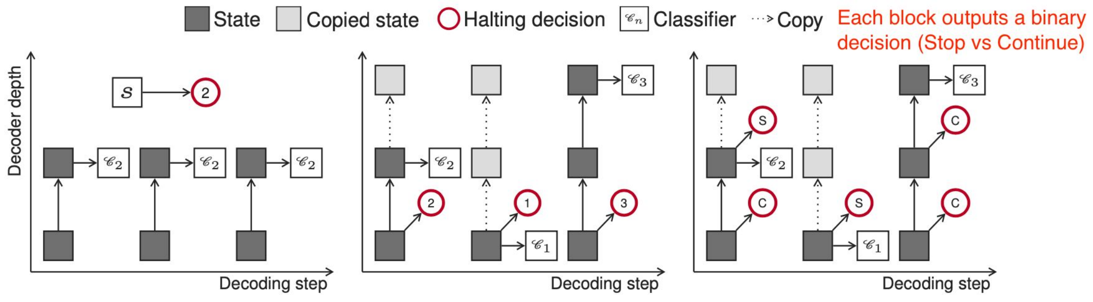
Motivation: at inference time, it is natural to assume that some tokens are easier to predict and thus do not require as much computation as others. Therefore we may only process its prediction through a limited number of layers to achieve a good balance between speed and performance. Learns to predict optimal numbers of layers needed for different input tokens.

Idea: attaches an output classifier to every layer to produce exit predictions based on activations of that layer. The classifier weight matrices can be different per layer or shared across layers. During training, the model sample different sequences of exits such that the model is optimized with hidden states of different layers. The learning objective incorporates likelihood probabilities predicted at different layers, n=1,...,N:

$$\mathrm{LL}_t^n = \log p(y_t|\mathbf{h}_{t-1}^n) \quad \mathrm{LL}^n = \sum_{t=1}^{|\mathbf{y}|} LL_t^n$$

Adaptive depth classifiers outputs a parametric distribution q_t. It is trained with cross entropy loss against an oracle distribution q_t^*.

Adaptive modeling: Depth-Adaptive Transformer



(a) Sequence-specific depth (b) Token-specific - Multinomial (c) Token-specific - Geometric-like

1. Sequence-specific depth classifier: All tokens of the same sequence share the same exit block. It depends on the average of the encoder representation of the sequence.
2. Token-specific depth classifier (multinomial): Each token is decoded with different exit block
3. Token-specific depth classifier (geometric-like): A binary exit prediction distribution is made per layer per token

Note: unlike dynamic computation in Universal Transformers, which applies the same set of layers iteratively, we apply different layers at every step to adjust both the amount of computation as well as the model capacity.

Efficient Attention

Problem: The computation and memory cost of the vanilla Transformer grows quadratically with sequence length and hence it is hard to be applied on very long sequences. Many efficiency improvements for Transformer architecture have something to do with the self-attention module - making it cheaper, smaller or faster to run.

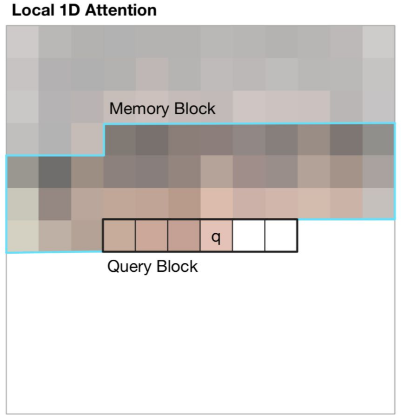
Efficient Attention: Sparse Attention Patterns - Fixed Local Context

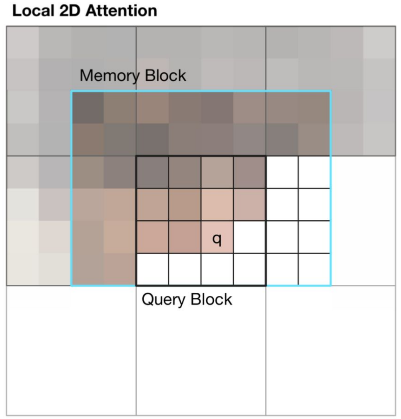
Fixed Local Context - a simple alternation to make self-attention less expensive is to **restrict the attention span of each token to local context only**, so that self-attention grows linearly with the sequence length.

The idea was introduced by **Image Transformer** (Parmer, et al 2018), which formulates image generation as sequence modeling using an encoder-decoder transformer architecture:

- The encoder generates a contextualized, per-pixel-channel representation of the source image;
- Then the decoder autoregressively generates an output image, one channel per pixel at each time step.

Let's label the representation of the current pixel to be generated as the query $\boldsymbol{q}$. Other positions whose representations will be used for computing $\boldsymbol{q}$ are key vector $\boldsymbol{k}_1, \boldsymbol{k}_2, \ldots$ and they together form a memory matrix $\boldsymbol{M}$. The scope of $\boldsymbol{M}$ defines the context window for pixel query $\boldsymbol{q}$.





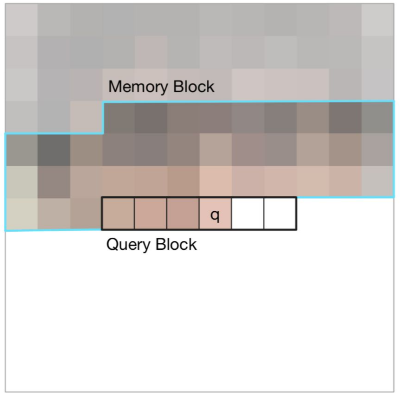
Efficient Attention: Sparse Attention Patterns - Fixed Local Context

Fixed Local Context - a simple alternation to make self-attention less expensive is to restrict the attention span of each token to **local** context only, so that self-attention grows linearly with the sequence length.

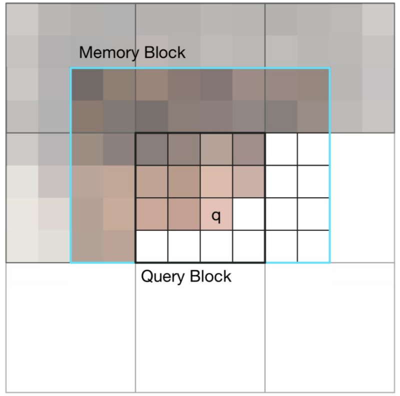
1. 1D Local Attention: The input image is flattened in the raster scanning order, that is, from left to right and top to bottom. The linearized image is then partitioned into non-overlapping query blocks. The context window consists of pixels in the same query block as q and a fixed number of additional pixels generated before this query block.

2. 2D Local Attention: The image is partitioned into multiple non-overlapping rectangular query blocks. The query pixel can attend to all others in the same memory blocks. To make sure the pixel at the top-left corner can also have a valid context window, the memory block is extended to the top, left and right by a fixed amount, respectively.

Local 1D Attention



Local 2D Attention



Efficient Attention: Sparse Attention Patterns - Strided Context

Sparse Transformer (Child et al., 2019) introduced factorized self-attention, through **sparse matrix factorization**, making it possible to train dense attention networks with hundreds of layers on sequence length up to 16,384, which would be infeasible on modern hardware otherwise.

Given a set of attention connectivity pattern $S=\{S_1,\dots,S_n\}$, where each S_i records a set of key positions that the i-th query vector attends to.

In **autoregressive models**, one attention span is defined as $S_i=\{j:j\leq i\}$ as it allows each token to attend to all the positions in the past.

In **factorized self-attention**, the set S_i is decomposed into a tree of dependencies, such that for every pair of (i,j) where $j\leq i$, there is a path connecting i back to j and i can attend to j either directly or indirectly.

Efficient Attention: Sparse Attention Patterns - Strided Context

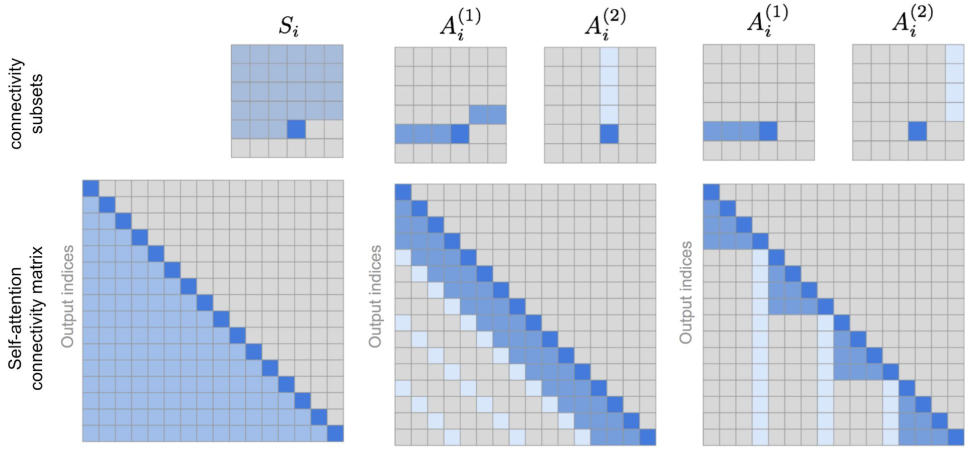
Sparse Transformer (Child et al., 2019) introduced factorized self-attention, through **sparse matrix factorization**, making it possible to train dense attention networks with hundreds of layers on sequence length up to 16,384, which would be infeasible on modern hardware otherwise.

1) Stride attention with stride $l \sim n^{1/2}$

$$A_i^{(1)} = \{t, t+1, \ldots, i\}, \text{ where } t = \max(0, i - \ell)$$
$$A_i^{(2)} = \{j : (i - j) \mod \ell = 0\}$$

2) Fixed attention

$$A_i^{(1)} = \{j : \lfloor \frac{j}{\ell} \rfloor = \lfloor \frac{i}{\ell} \rfloor\}$$
$$A_i^{(2)} = \{j : j \mod \ell \in \{\ell - c, \ldots, \ell - 1\}\}$$



(a) Transformer

(b) Sparse Transformer with strided attention.

(c) Sparse Transformer with fixed attention.

Efficient Attention: Sparse Attention Patterns - Strided Context

Sparse Transformer (Child et al., 2019) introduced factorized self-attention, through **sparse matrix factorization**, making it possible to train dense attention networks with hundreds of layers on sequence length up to 16,384, which would be infeasible on modern hardware otherwise.

There are three **ways to use sparse factorized** attention patterns in Transformer architecture:

1) One attention type per residual block and then interleave them, attn(X)=Attend(X, $A^{(n \bmod m)}$)W^0, where n is the index of the current residual block.
2) Set up a single head which attends to locations that all the factorized heads attend to, attn(X)=Attend(X, $\cup^{p}_{m=1} A^{(m)}$)W^0.
3) Use a multi-head attention mechanism, but different from vanilla Transformer, each head might adopt a pattern presented above, 1 or 2. This option often performs the best.

Note: total time/memory $\mathcal{O}(n^{3/2})$

$$\text{Attend}(\mathbf{X}, \mathcal{S}) = \Big(a(\mathbf{x}_i, S_i)\Big)_{i \in \{1,\dots,L\}} \qquad a(\mathbf{x}_i, S_i) = \text{softmax}\Big(\frac{(\mathbf{x}_i \mathbf{W}^q)(\mathbf{x}_j \mathbf{W}^k)^\top_{j \in S_i}}{\sqrt{d_k}}\Big)(\mathbf{x}_j \mathbf{W}^v)_{j \in S_i}$$

Efficient Attention: Sparse Attention Patterns - Strided Context

Blockwise Attention (Qiu et al. 2019) introduces a sparse block matrix to only allow each token to attend to a small set of other tokens. Each attention matrix of size $L \times L$ is partitioned into $n \times n$ smaller blocks of size $L/n \times L/n$ and a sparse block matrix $M \in \{0,1\}^{L \times L}$ is defined by a permutation of $1,\ldots,n$, which records the column index per row in the block matrix.

$$\text{attn}(\mathbf{Q},\mathbf{K},\mathbf{V},\mathbf{M}) = \text{softmax}\Big(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}} \odot \mathbf{M}\Big)\mathbf{V}$$

$$(\mathbf{A} \odot \mathbf{M})_{ij} = \begin{cases} A_{ij} & \text{if } M_{ij} = 1 \\ -\infty & \text{if } M_{ij} = 0 \end{cases}$$

$$\text{where } M_{ij} = \begin{cases} 1 & \text{if } \pi\big(\lfloor \frac{(i-1)n}{L} + 1 \rfloor\big) = \lfloor \frac{(j-1)n}{L} + 1 \rfloor \\ 0 & \text{otherwise} \end{cases}$$

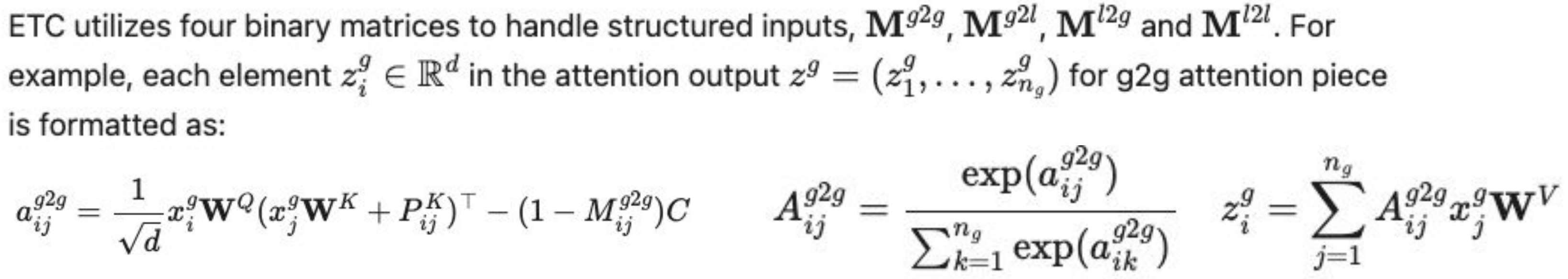
$$\text{Blockwise-attn}(\mathbf{Q},\mathbf{K},\mathbf{V},\mathbf{M}) = \begin{bmatrix} \text{softmax}\Big(\frac{\hat{\mathbf{q}}_1 \hat{\mathbf{k}}_{\pi(1)}^\top}{\sqrt{d}}\Big)\hat{\mathbf{v}}_{\pi(1)} \\ \vdots \\ \text{softmax}\Big(\frac{\hat{\mathbf{q}}_n \hat{\mathbf{k}}_{\pi(n)}^\top}{\sqrt{d}} \odot\Big)\hat{\mathbf{v}}_{\pi(n)} \end{bmatrix}$$

Note: reduce memory complexity from $\mathcal{O}(L^2)$ to $\mathcal{O}(L/n \times L/n \times n) = \mathcal{O}(L^2/n)$

Efficient Attention: Sparse Attention Patterns - Combination of Local and Global Context

Idea: ETC (Extended Transformer Construction; Ainslie et al. 2019), Longformer (Beltagy et al. 2020) and Big Bird (Zaheer et al. 2020) models combine both local and global context when building an attention matrix.

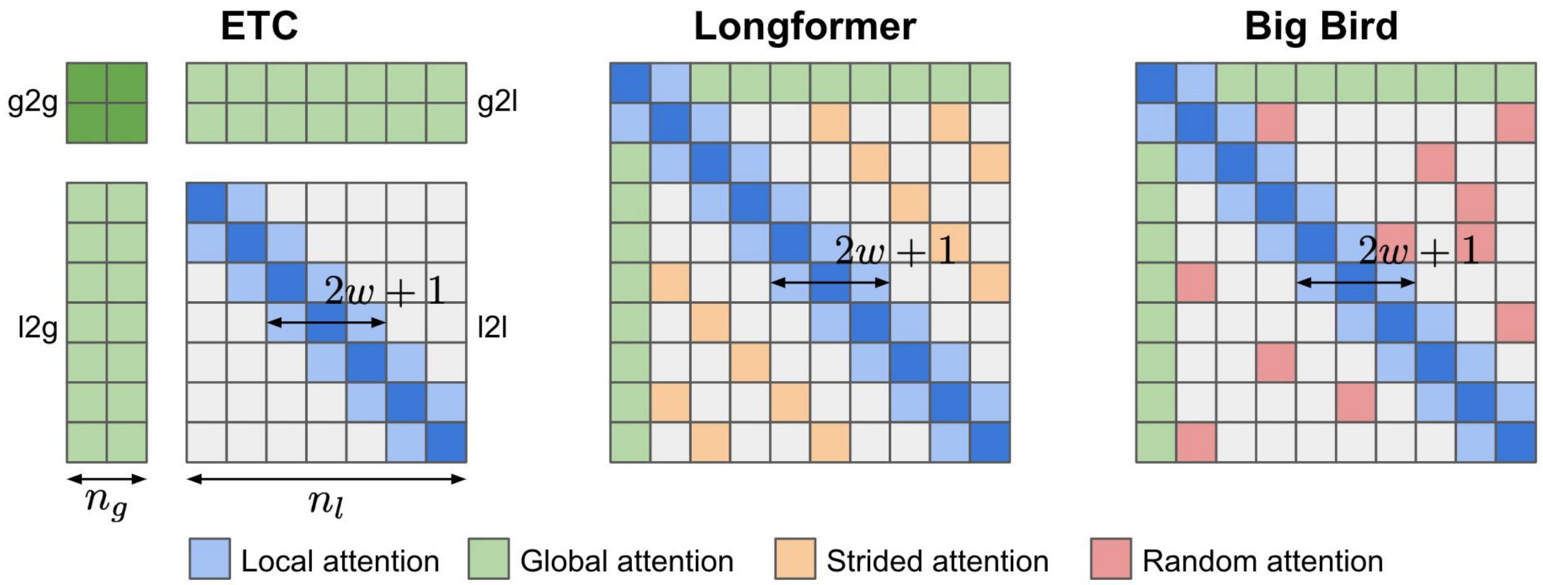
Global-Local Attention of ETC (Ainslie et al. 2019) takes two inputs, (1) the long input x^l of size n_l which is the regular input sequence and (2) the global input x^g of size n_g which contains a smaller number of auxiliary tokens, $n_g \ll n_l$. Attention is thus split into four components based on directional attention across these two inputs: **g2g**, **g2l**, **l2g** and **l2l**. Because the **l2l** attention piece can be very large, it is restricted to a fixed size attention span of radius w (i.e. local attention span) and the **l2l** matrix can be reshaped to $n_l \times (2w+1)$.



ETC utilizes four binary matrices to handle structured inputs, $\mathbf{M}^{g2g}$, $\mathbf{M}^{g2l}$, $\mathbf{M}^{l2g}$ and $\mathbf{M}^{l2l}$. For example, each element $z_i^g \in \mathbb{R}^d$ in the attention output $z^g = (z_1^g, \dots, z_{n_g}^g)$ for g2g attention piece is formatted as:

$$a_{ij}^{g2g} = \frac{1}{\sqrt{d}} x_i^g \mathbf{W}^Q (x_j^g \mathbf{W}^K + P_{ij}^K)^\top - (1 - M_{ij}^{g2g}) C \qquad A_{ij}^{g2g} = \frac{\exp(a_{ij}^{g2g})}{\sum_{k=1}^{n_g} \exp(a_{ik}^{g2g})} \qquad z_i^g = \sum_{j=1}^{n_g} A_{ij}^{g2g} x_j^g \mathbf{W}^V$$

Efficient Attention: Sparse Attention Patterns - Combination of Local and Global Context



Efficient Attention: Content-based Attention

The improvements proposed by **Reformer** (Kitaev, et al. 2020) aim to solve the following pain points in vanilla Transformer:

- Quadratic time and memory complexity within self-attention module.
- Memory in a model with N layers is N-times larger than in a single-layer model because we need to store activations for back-propagation.
- The intermediate FF layers are often quite large.

Reformer proposed two main changes:

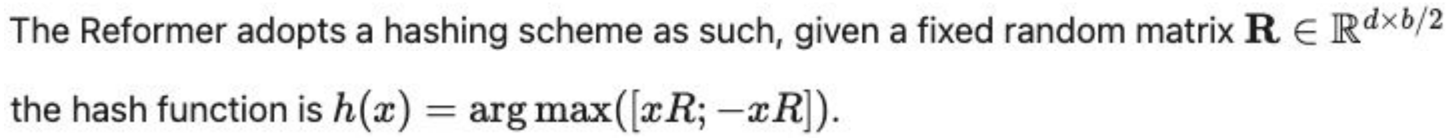
1. Replace the dot-product attention with locality-sensitive hashing (LSH) attention, reducing the complexity from $\mathcal{O}(L^2)$ to $\mathcal{O}(L \log L)$.
2. Replace the standard residual blocks with reversible residual layers, which allows storing activations only once during training instead of $\mathbf{N}$ times (i.e. proportional to the number of layers).

Efficient Attention: Content-based Attention

Locality-Sensitive Hashing Attention

In $QK^\top$ part of the attention formula, we are only interested in the largest elements as only large elements contribute a lot after softmax. For each query $q_i \in Q$, we are looking for row vectors in K closest to q_i. In order to find nearest neighbors quickly in high-dimensional space, Reformer incorporates Locality-Sensitive Hashing (LSH) into its attention mechanism.

In LSH attention, a query can only attend to positions in the same hashing bucket, $S_j = \{j : h(q_i) = h(k_j)\}$, where h - hash function



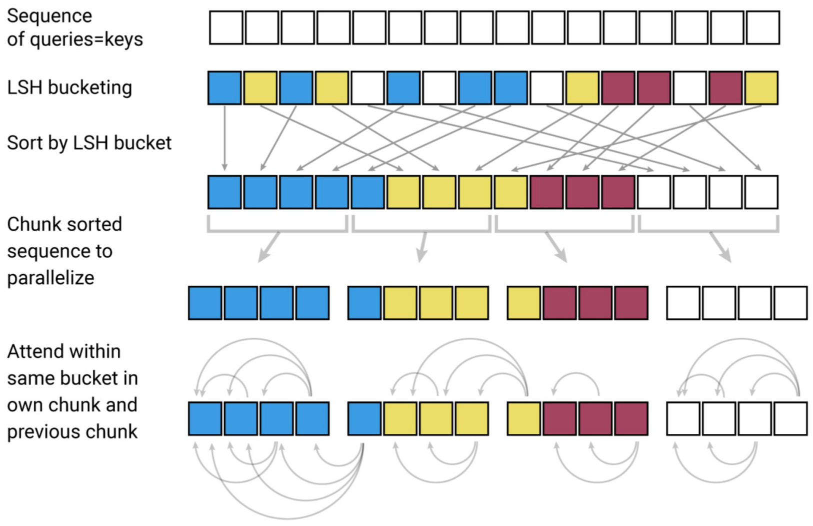
The Reformer adopts a hashing scheme as such, given a fixed random matrix $\mathbf{R} \in \mathbb{R}^{d \times b/2}$

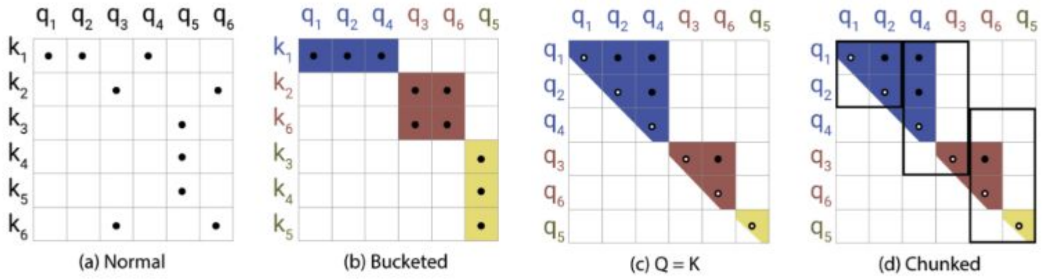
the hash function is $h(x) = \arg\max([xR; -xR])$.

Efficient Attention: Content-based Attention

Locality-Sensitive Hashing Attention

- (a) The attention matrix for full attention is often sparse.
- (b) Using LSH, we can sort the keys and queries to be aligned according to their hash buckets.
- (c) Set $\boldsymbol{Q}=\boldsymbol{K}$ (precisely $\boldsymbol{k}_j=\boldsymbol{q}_j/|\boldsymbol{q}_j|$), so that there are equal numbers of keys and queries in one bucket, easier for batching. Interestingly, this “shared-QK” config does not affect the performance of the Transformer.
- (d) Apply batching where chunks of $\boldsymbol{m}$ consecutive queries are grouped together.


Sequence of queries=keys
LSH bucketing
Sort by LSH bucket
Chunk sorted sequence to parallelize
Attend within same bucket in own chunk and previous chunk



Efficient Attention: Content-based Attention

Reversible Residual Network

The motivation for reversible residual network is to design the architecture in a way that activations at any given layer can be recovered from the activations at the following layer, using only the model parameters. Hence, we can save memory by recomputing the activation during backprop rather than storing all the activations.

Given a layer $x \mapsto y$, the normal residual layer does $y=x+F(x)$, but the reversible layer splits both input and output into pairs $(x_1,x_2) \mapsto (y_1,y_2)$, and then:

$$y_1 = x_1 + F(x_2),\ y_2 = x_2 + G(y_1) \qquad \text{reverse} \qquad x_2 = y_2 - G(y_1),\ x_1 = y_1 - F(x_2)$$

For transformer: $$Y_1 = X_1 + \mathrm{Attention}(X_2),\ Y_2 = X_2 + \mathrm{FeedForward}(Y_1)$$

Efficient Attention: Low-Rank Attention

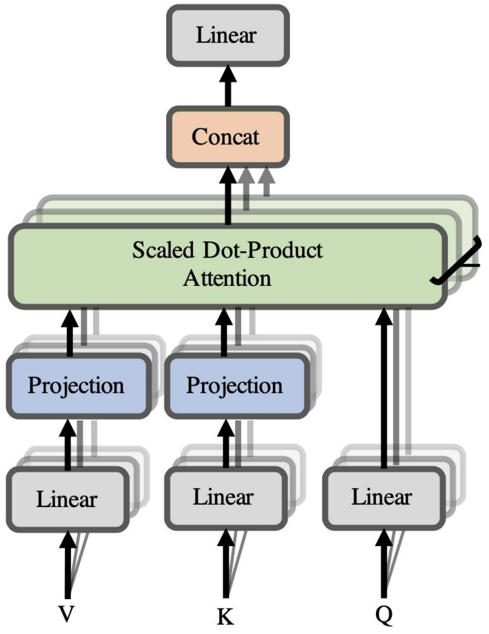
Linformer (Wang et al. 2020) approximates the full attention matrix with a low rank matrix, reducing the time & space complexity to be ***linear***.

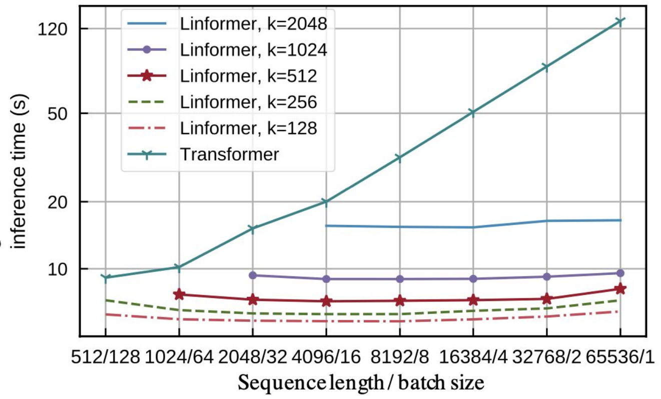
Instead of using expensive SVD to identify low rank decomposition, Linformer adds two linear projections $E_i, F_i \in \mathrm{R}^{L\times k}$ for key and value matrices, respectively, reducing their dimensions from $L\times d$ to $k\times d$. As long as $k \ll L$, the attention memory can be greatly reduced.

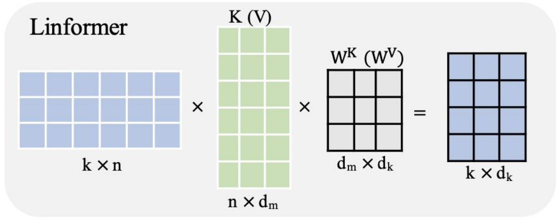
$$\begin{aligned}\overline{\text{head}}_i &= \text{attn}(\mathbf{X}_q\mathbf{W}_i^q, \mathbf{E}_i\mathbf{X}_k\mathbf{W}_i^k, \mathbf{F}_i\mathbf{X}_v\mathbf{W}_i^v) \\ &= \underbrace{\text{softmax}\left(\frac{\mathbf{X}_q\mathbf{W}_i^q(\mathbf{E}_i\mathbf{X}_k\mathbf{W}_i^k)^\top}{\sqrt{d}}\right)}_{\text{low rank attention matrix } \bar{A}\in\mathbb{R}^{k\times d}}\mathbf{F}_i\mathbf{X}_v\mathbf{W}_i^v\end{aligned}$$

Efficient Attention: Low-Rank Attention

Linformer (Wang et al. 2020) approximates the full attention matrix with a low rank matrix, reducing the time & space complexity to be *linear*.







Efficient Attention: Low-Rank Attention

Linformer (Wang et al. 2020) approximates the full attention matrix with a low rank matrix, reducing the time & space complexity to be *linear*.

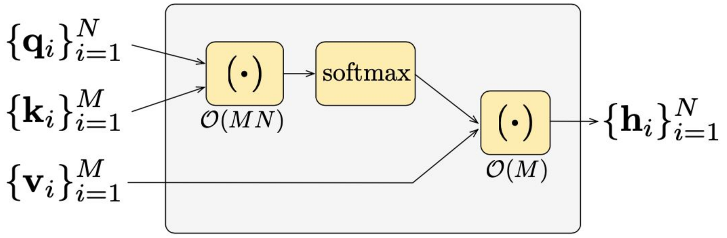
Additional techniques can be applied to further improve efficiency of Linformer:

- Parameter sharing between projection layers, such as head-wise, key-value and layer-wise (across all layers) sharing.
- Use different k at different layers, as heads in higher layers tend to have a more skewed distribution (lower rank) and thus we can use smaller k at higher layers.
- Use different types of projections; e.g. mean/max pooling, convolution layer with kernel and stride L/k.

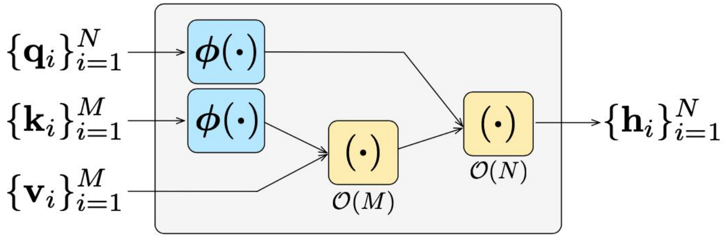
Efficient Attention: Low-Rank Attention

Other ideas:

1. **Random Feature Attention** (RFA; Peng et al. 2021) relies on random feature methods (Rahimi & Recht, 2007) to approximate softmax operation in self-attention with low rank feature maps in order to achieve *linear time and space complexity.*
2. **Performers** (Choromanski et al. 2021) also adopts random feature attention with improvements on the kernel construction to further reduce the kernel approximation error.


$\{\mathbf{q}_i\}_{i=1}^N$
$\{\mathbf{k}_i\}_{i=1}^M$
$\{\mathbf{v}_i\}_{i=1}^M$
$(\cdot)$
$\mathcal{O}(MN)$
softmax
$(\cdot)$
$\mathcal{O}(M)$
$\{\mathbf{h}_i\}_{i=1}^N$

(a) Softmax attention.


$\{\mathbf{q}_i\}_{i=1}^N$
$\{\mathbf{k}_i\}_{i=1}^M$
$\{\mathbf{v}_i\}_{i=1}^M$
$\phi(\cdot)$
$\phi(\cdot)$
$(\cdot)$
$\mathcal{O}(M)$
$(\cdot)$
$\mathcal{O}(N)$
$\{\mathbf{h}_i\}_{i=1}^N$

(b) Random feature attention.

Conclusion

We discovered different types of approaches such as using as “Longer Context”, “Adaptive modeling” and “Efficient Attention” to create a better (cheaper, faster or perform better) transformer.

Спасибо за внимание!

Источники:

https://lilianweng.github.io/posts/2023-01-27-the-transformer-family-v2/