Harder
Better
Faster
Stronger
Transformers

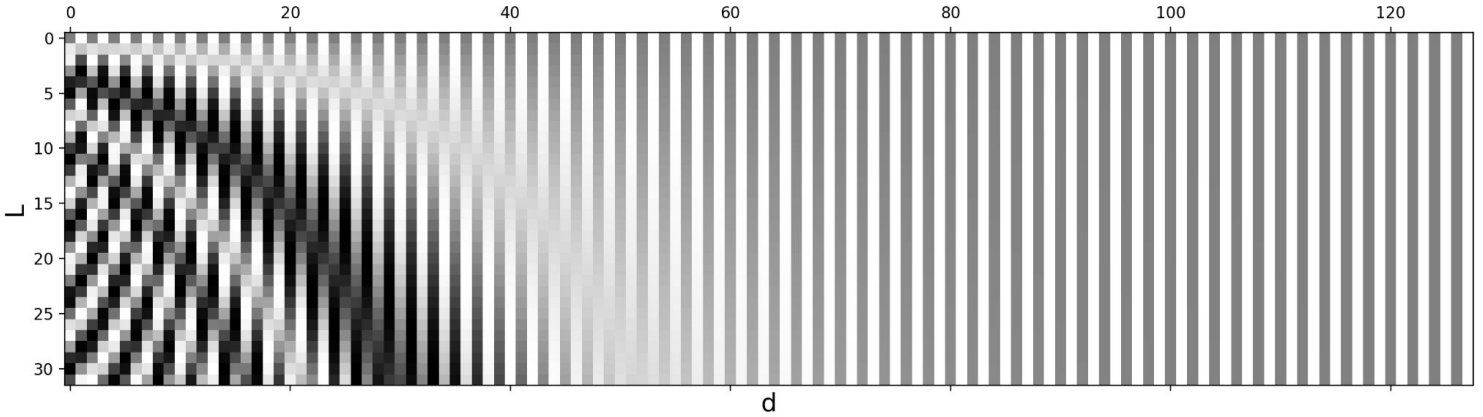
By Max Zakharchenko

Positional Encoding

Self-attention operation is permutation invariant, it is important to use **positional encoding**

Sinusoidal Positional Encoding (vanilla)

$$\mathrm{PE}(i,\delta) = \begin{cases} \sin\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if} \delta = 2\delta' \\ \cos\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if} \delta = 2\delta' + 1 \end{cases}$$



Learned Positional Encoding (vanilla)

Relative Position Encoding (Transformer-XL)

$$a_{ij} = \mathbf{q}_i \mathbf{k}_j^\top = (\mathbf{x}_i + \mathbf{p}_i)\mathbf{W}^q((\mathbf{x}_j + \mathbf{p}_j)\mathbf{W}^k)^\top$$

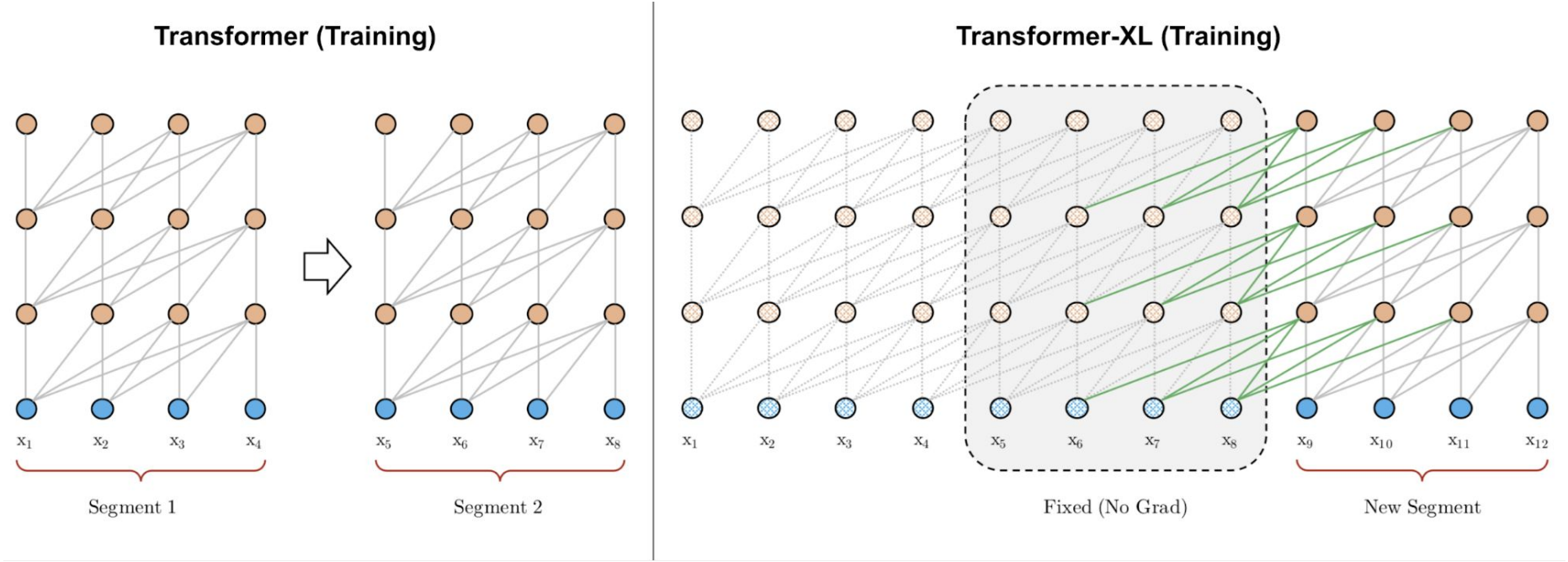
$$= \mathbf{x}_i \mathbf{W}^q \mathbf{W}^{k\top} \mathbf{x}_j^\top + \mathbf{x}_i \mathbf{W}^q \mathbf{W}^{k\top} \mathbf{p}_j^\top + \mathbf{p}_i \mathbf{W}^q \mathbf{W}^{k\top} \mathbf{x}_j^\top + \mathbf{p}_i \mathbf{W}^q \mathbf{W}^{k\top} \mathbf{p}_j^\top$$

$$a_{ij}^{\text{rel}} = \underbrace{\mathbf{x}_i \mathbf{W}^q \mathbf{W}_E^{k\top} \mathbf{x}_j^\top}_{\text{content-based addressing}} + \underbrace{\mathbf{x}_i \mathbf{W}^q \mathbf{W}_R^{k\top} \mathbf{r}_{i-j}^\top}_{\text{content-dependent positional bias}} + \underbrace{\mathbf{u} \mathbf{W}_E^{k\top} \mathbf{x}_j^\top}_{\text{global content bias}} + \underbrace{\mathbf{v} \mathbf{W}_R^{k\top} \mathbf{r}_{i-j}^\top}_{\text{global positional bias}}$$

Longer Context

- 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.

Transformer-XL



Transformer-XL

$$\mathbf{h}_{\tau}^{n} - \text{hidden state of } n\text{-th layer for } \tau\text{-th segment}$$

$$\widetilde{\mathbf{h}}_{\tau+1}^{(n-1)} = [\text{stop-gradient}(\mathbf{h}_{\tau}^{(n-1)}) \circ \mathbf{h}_{\tau+1}^{(n-1)}]$$

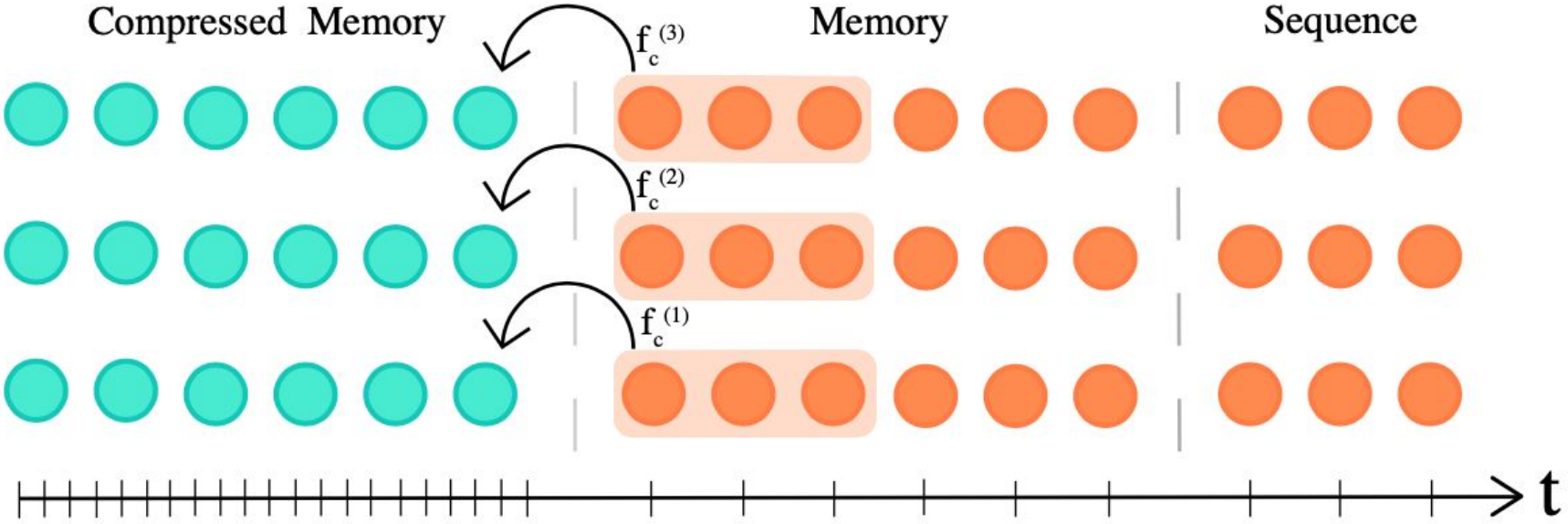
$$\mathbf{Q}_{\tau+1}^{(n)} = \mathbf{h}_{\tau+1}^{(n-1)}\mathbf{W}^{q}$$

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

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

$$\mathbf{h}_{\tau+1}^{(n)} = \text{transformer-layer}(\mathbf{Q}_{\tau+1}^{(n)}, \mathbf{K}_{\tau+1}^{(n)}, \mathbf{V}_{\tau+1}^{(n)})$$

Compressive Transformer



Compressive Transformer

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

- Max/mean pooling of kernel and stride size *c*
- 1D convolution with kernel and stride size *c*
- Dilated convolution
- Most used memories

Compressive Transformer

1. **Auto-encoding loss (lossless compression objective)**

$$\mathcal{L}_{ac} = \|\mathbf{old\_mem}^{(i)} - g(\mathbf{new\_cm}^{(i)})\|_2$$

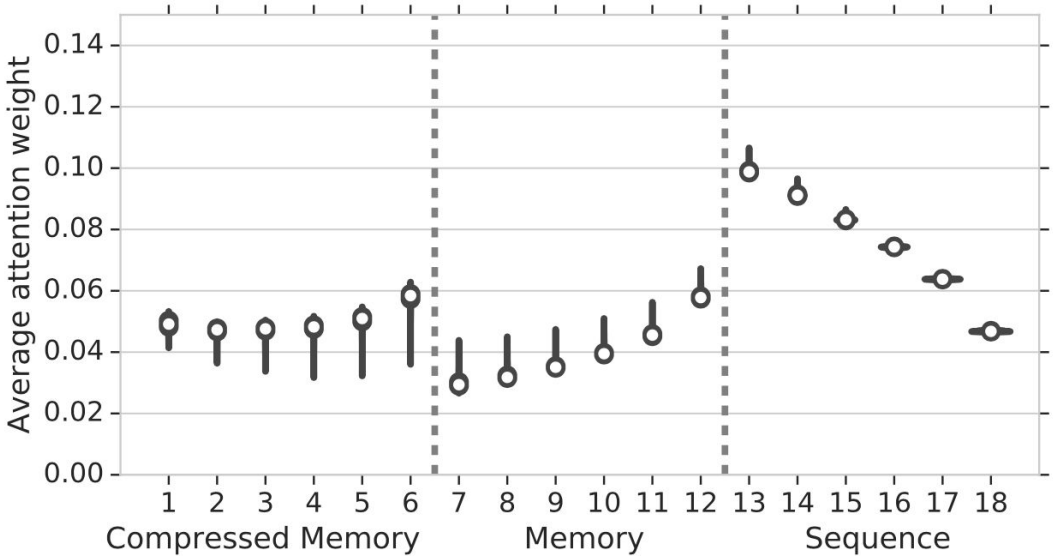
g is reverse of ***f*** and ***g*** is learned

2. **Attention-reconstruction loss (lossy objective)**

$$\mathcal{L}_{ar} = \|\text{attn}(\mathbf{h}^{(i)}, \mathbf{old\_mem}^{(i)}) - \text{attn}(\mathbf{h}^{(i)}, \mathbf{new\_cm}^{(i)})\|_2$$

Compressive Transformer

Attention weights, from oldest to newest, are stored in three locations: compressed memory → memory → causally masked sequence



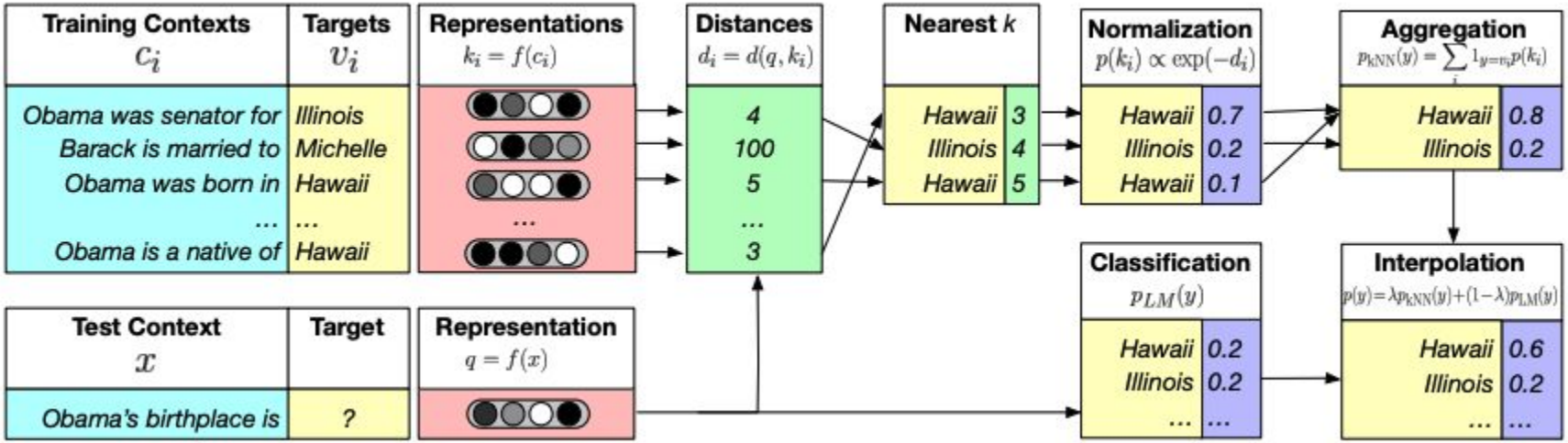
kNN-LM

LM solves two subproblems

- Mapping sentence prefixes to fixed-sized representations
- Using these representations to predict the next word in the text

kNN-LM is based on the hypothesis that the representation problem is easier

kNN-LM


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

kNN-LM

f - maps context to fixed-length vector representation computed by pre-trained LM (like output of self-attention layer). The kNN datastore is

$$(\mathcal{K}, \mathcal{V}) = \{(f(c_i), w_i) \mid (c_i, w_i) \in \mathcal{D}\}$$

At test time, given the context ***x*** model generates next word distribution

$$p_{\text{kNN}}(y|x) \propto \sum_{(k_i, v_i) \in \mathcal{N}} \mathbb{I}[y = v_i] e^{-\|k_i - f(x)\|_2}$$

In the end with hyperparameter ***λ***

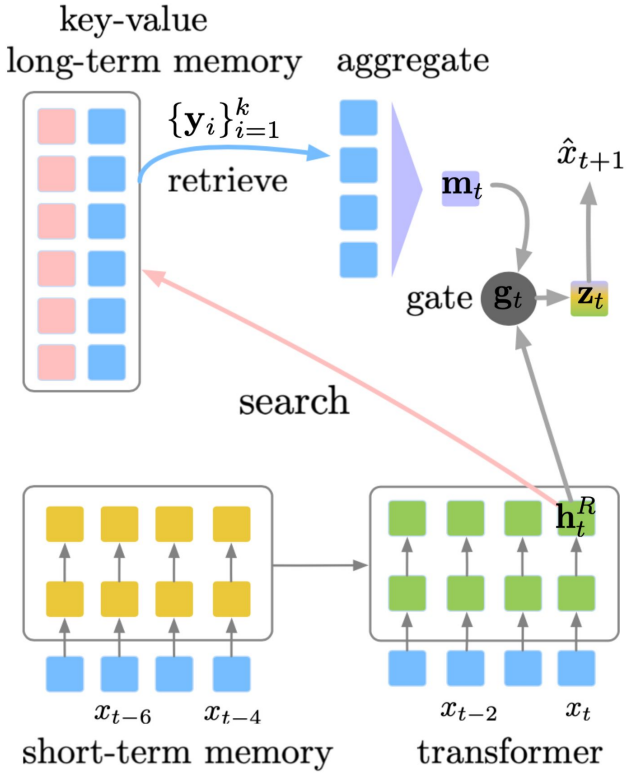
$$p(y|x) = \lambda p_{\text{kNN}}(y|x) + (1 - \lambda) p_{\text{LM}}(y|x)$$

SPALM

Idea to combine the best of all

- Transformer-XL style memory for hidden states (short-term memory)
- kNN style key-value store (long memory)

SPALM


key-value
long-term memory
aggregate
$\{\mathbf{y}_i\}_{i=1}^{k}$
retrieve
$\mathbf{m}_t$
$\hat{x}_{t+1}$
gate $\mathbf{g}_t$
$\mathbf{z}_t$
search
$\mathbf{h}_t^R$
x_{t-6}
x_{t-4}
x_{t-2}
x_t
short-term memory
transformer

SPALM

- Obtain output tokens from database $y_1, y_2 \ldots y_N$ which contexts' are close to current
- Map y_k with word embedding matrix to $\boldsymbol{y}_k$
- Do funny ($\mathbf{w}_g$ is a vector to learn)

$$\mathbf{m}_t = \sum_{i=1}^{k} \frac{\exp(\mathbf{y}_i^\top \mathbf{h}_t^R)}{\sum_{j=1}^{k} \exp(\mathbf{y}_j^\top \mathbf{h}_t^R)} \cdot \mathbf{y}_i$$

$$\mathbf{g}_t = \sigma(\mathbf{w}_g^\top \mathbf{h}_t^R)$$

$$\mathbf{z}_t = (1 - \mathbf{g}_t) \odot \mathbf{m}_t + \mathbf{g}_t \odot \mathbf{h}_t^R$$

$$p(x_{t+1} \mid \mathbf{x}_{\leq t}) = \mathrm{softmax}(\mathbf{z}_t; \mathbf{W})$$

Memorizing Transformer

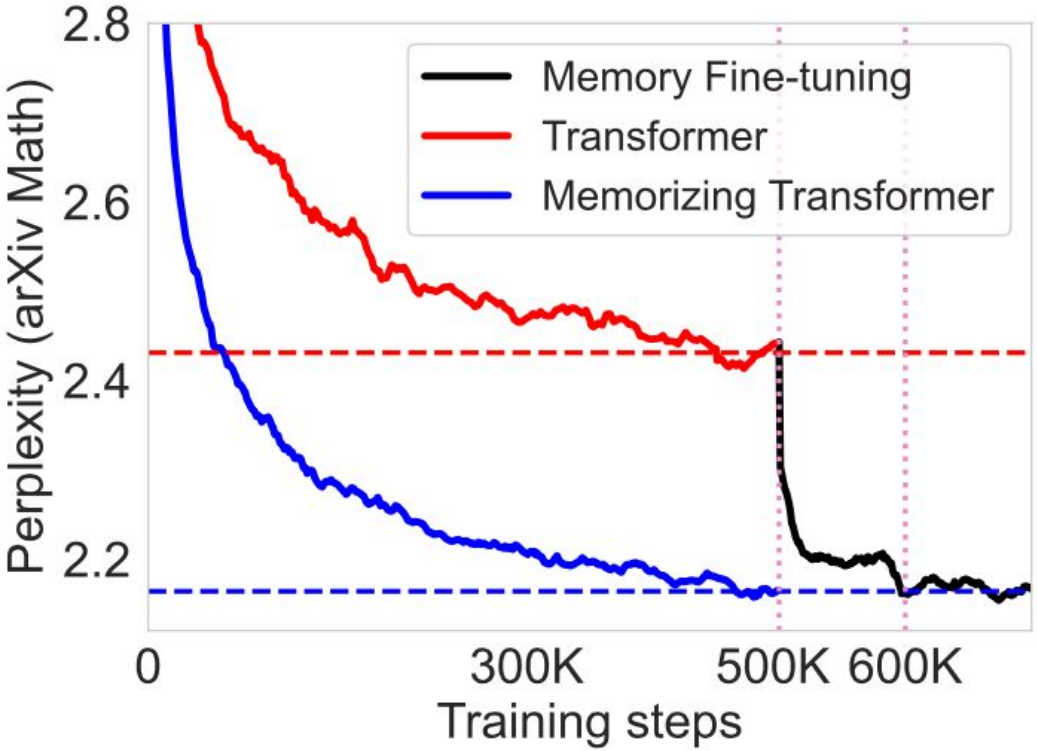
- Adds a kNN-augmented attention layer near the top of a **decoder-only** Transformer.
- Same QKV values are user for both local and kNN attention
- Two types of attention are combined with a learnable gating parameter

Memorizing Transformer

Fun facts

- It is better to start with a small cache size and then finetune with a bigger one
- The smaller Memorizing Transformer with just 8k cache size can match the score of larger vanilla Transformer with 5X more trainable parameters
- Increasing the size of external memory provides consistent gains up to 262K
- non-memory transformer can be finetuned to use memory

Memorizing Transformer



Distance-Enhanced Attention Scores

In order to encourage the model to extrapolate over longer context that what the model is trained on, we can explicitly attach the positional information to every pair of attention score

Distance-Enhanced Attention Scores

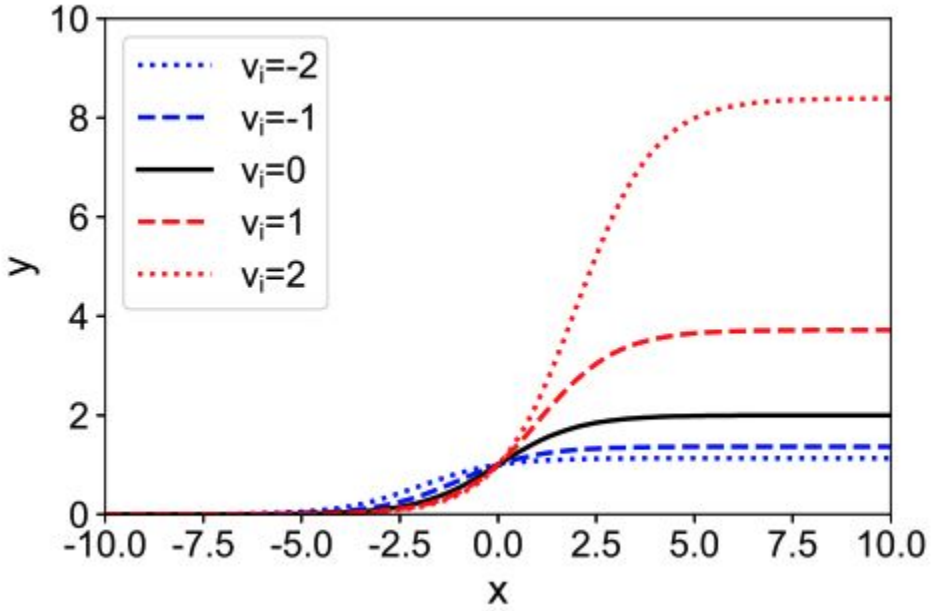
α and β are learnable

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

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

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

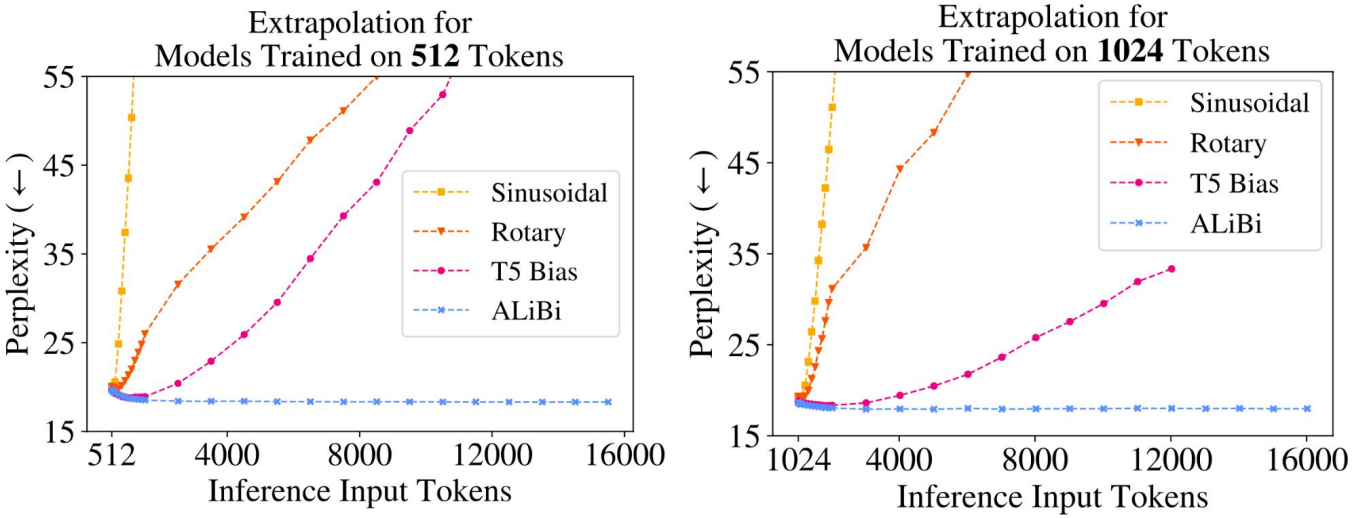
Distance-Enhanced Attention Scores



ALiBi

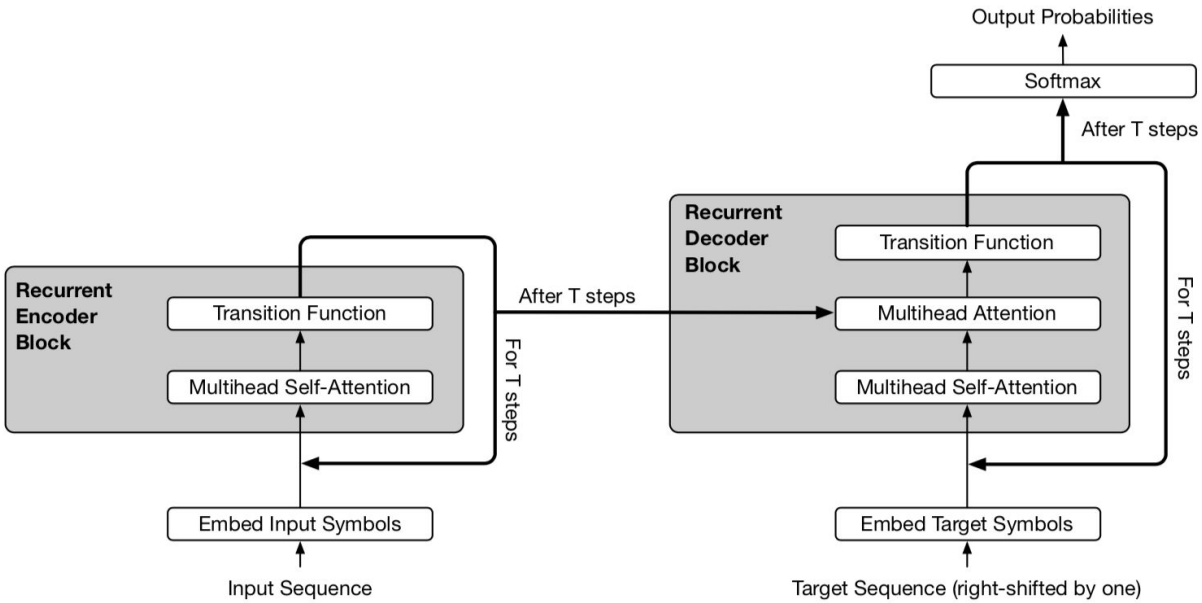
$$\text{softmax}(\mathbf{q}_i \mathbf{K}^\top + \alpha_i \cdot [0, -1, -2, \ldots, -(i-1)])$$

$$\text{For 8 heads } \alpha_i = \frac{1}{2}, \frac{1}{2^2}, \ldots, \frac{1}{2^8}$$



Universal Transformer

Idea to do attention + Point-wise dynamically amount of time



Universal Transformer

$$\mathbf{A}^t = \mathrm{LayerNorm}(\mathbf{h}^{t-1} + \mathrm{MultiHeadAttention}(\mathbf{h}^{t-1} + \mathbf{P}^t)$$
$$\mathbf{h}^t = \mathrm{LayerNorm}(\mathbf{A}^{t-1} + \mathrm{Transition}(\mathbf{A}^t))$$

Transition is either a depth-wise convolution or fully-connected that consists of two position-wise (applied to each row individually) affine transformation + ReLU

P^T definition

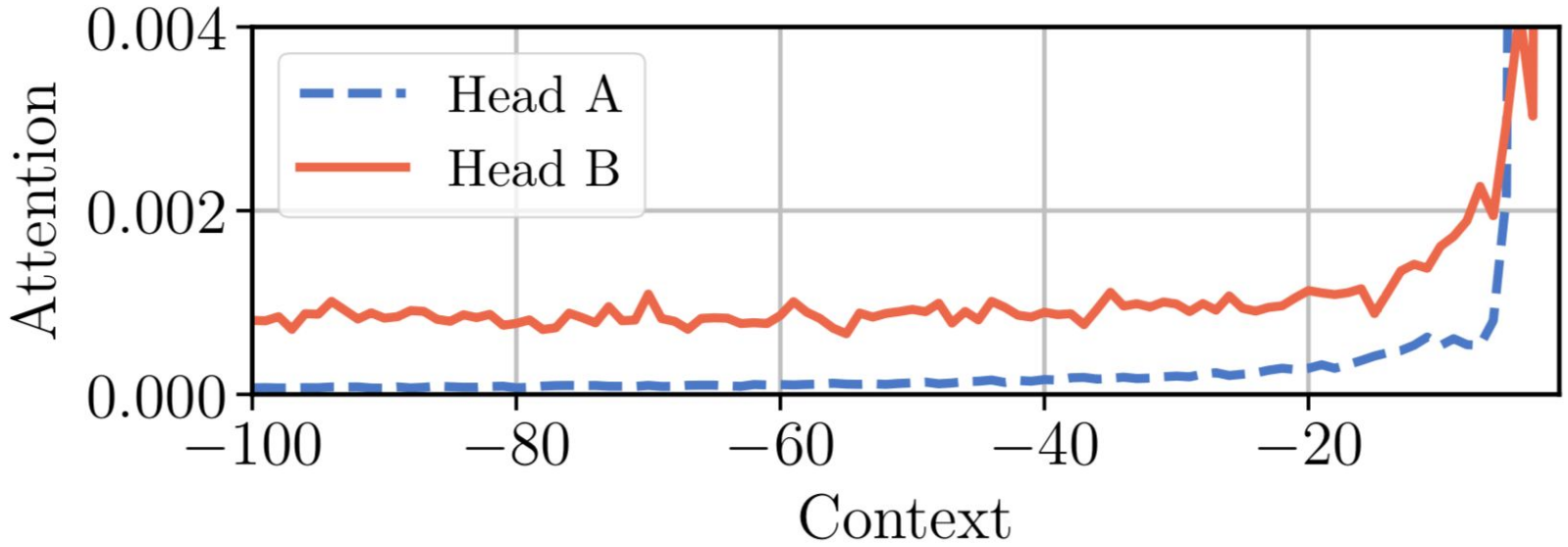
$$\mathrm{PE}(i,t,\delta) = \begin{cases} \sin(\frac{i}{10000^{2\delta'/d}}) \oplus \sin(\frac{t}{10000^{2\delta'/d}}) & \mathrm{if} \delta = 2\delta' \\ \cos(\frac{i}{10000^{2\delta'/d}}) \oplus \cos(\frac{t}{10000^{2\delta'/d}}) & \mathrm{if} \delta = 2\delta' + 1 \end{cases}$$

Adaptive modeling

Idea to adapt amount of computation according to different inputs

Adaptive Attention Span

Maybe different attentions heads might assing scores differently



Adaptive Attention Span

Fix the attention span to size *s*. Classic attention looks like this

$$e_{ij} = \mathbf{q}_i \mathbf{k}_j^\top$$

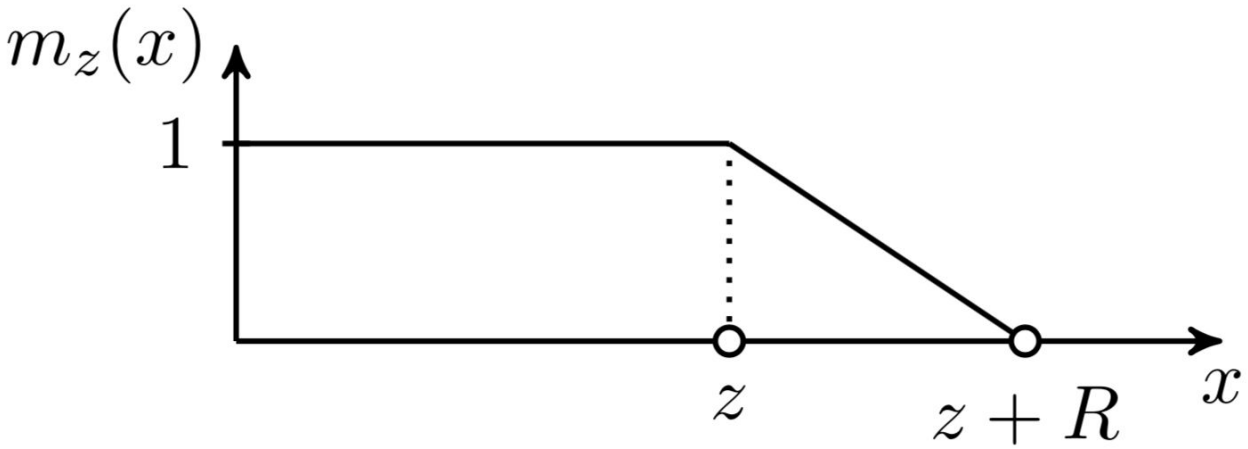
$$a_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{r=i-s}^{i-1} \exp(e_{ir})}$$

$$\mathbf{y}_i = \sum_{r=i-s}^{i-1} a_{ir} \mathbf{v}_r = \sum_{r=i-s}^{i-1} a_{ir} \mathbf{x}_r \mathbf{W}^v$$

Adaptive Attention Span

Define *soft mask function* m_z to control attention span with a learnable param *z*

$$m_z(x) = \text{clip}(\frac{1}{R}(R + z - x), 0, 1)$$



Adaptive Attention Span

$$a_{ij} = \frac{m_z(i-j)\exp(s_{ij})}{\sum_{r=i-s}^{i-1} m_z(i-r)\exp(s_{ir})}$$

m_z different per-head. Moreover, the loss function has a L1 penalty on Σ z

Depth-Adaptive Transformer

Idea to predict output after each decoder block and decide how many block we need

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

Depth-Adaptive Transformer

1. Sequence-specific depth classifier

All tokens of the same sequence share the same exit block

$$q(n|\mathbf{x}) = \mathrm{softmax}(\mathbf{W}_n \bar{\mathbf{x}} + b_n) \in \mathbb{R}^N$$

$$q^*_{\mathrm{lik}}(\mathbf{x}, \mathbf{y}) = \delta(\arg\max_n \mathrm{LL}^n - \lambda n)$$

$$\text{or } q^*_{\mathrm{corr}}(\mathbf{x}, \mathbf{y}) = \delta(\arg\max_n C^n - \lambda n) \text{ where } C^n = |\{t|y_t = \arg\max_y p(y|\mathbf{h}^n_{t-1})\}|$$

Depth-Adaptive Transformer

2. Token-specific depth classifier (multinomial)

Each token is decoded with different exit block, predicted on the first decoder hidden state

$$q_t(n|\mathbf{x}, \mathbf{y}_{<t}) = \mathrm{softmax}(\mathbf{W}_n \mathbf{h}_t^1 + b_n)$$

Depth-Adaptive Transformer

3. Token-specific depth classifier (geometric-like)

A binary exit prediction is made per layer per token

$$\mathcal{X}_t^n = \text{sigmoid}(\mathbf{w}_n^\top \mathbf{h}_t^n + b_n) \quad \forall n \in [1, \ldots, N-1]$$

$$q_t(n|\mathbf{x}, \mathbf{y}_{<t}) = \begin{cases} \mathcal{X}_t^n \prod_{n'<n}(1 - \mathcal{X}_t^{n'}) & \text{if } n < N \\ \prod_{n'<N}(1 - \mathcal{X}_t^{n'}) & \text{otherwise} \end{cases}$$

$$\kappa(t, t) = \exp(\frac{|t-t|^2}{\sigma})$$

$$q^*_{\text{lik}}(\mathbf{x}, \mathbf{y}) = \delta(\arg\max_n \widetilde{\text{LL}}_t^n - \lambda n) \text{ where } \widetilde{\text{LL}}_t^n = \sum_{t'=1}^{|\mathbf{y}|} \kappa(t, t') LL_{t'}^n$$

$$\text{or} q^*_{\text{cor}}(\mathbf{x}, \mathbf{y}) = \delta(\arg\max_n \tilde{C}_t^n - \lambda n) \text{ where } C_t^n = \mathbb{I}[y_t = \arg\max_y p(y|\mathbf{h}_{t-1}^n)],\ \tilde{C}_t^n = \sum_{t'=1}^{|\mathbf{y}|} \kappa(t, t') C_{t'}^n$$

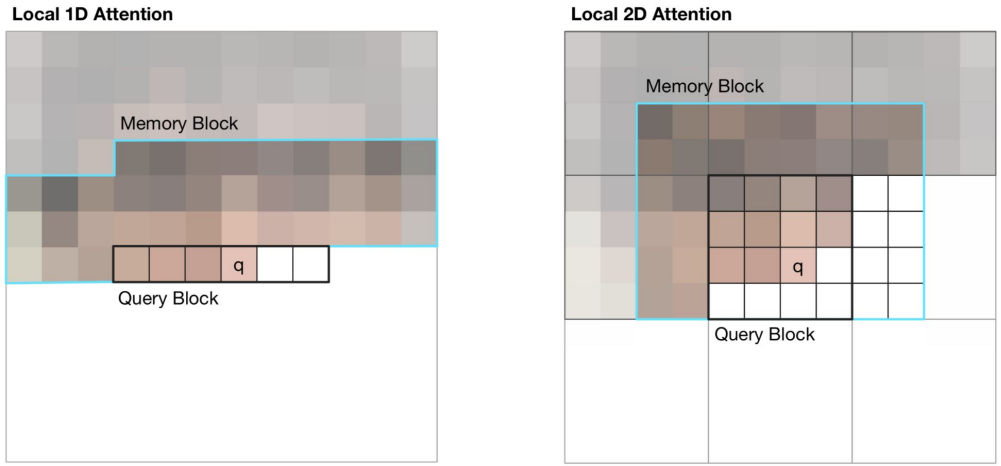
Efficient Attention

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.

Sparse Attention Patterns

Fixed Local Context

Restrict the attention span of each token to **local** context only, results in a linearly grows



Sparse Attention Patterns

Strided Context

$$S = \{S_1, \ldots, S_n\} - \text{attention sets}$$

$$\text{Attend}(\mathbf{X}, \mathcal{S}) = \Big(a(\mathbf{x}_i, S_i)\Big)_{i \in \{1,\ldots,L\}}$$

$$\text{where } a(\mathbf{x}_i, S_i) = \text{softmax}\Big(\frac{(\mathbf{x}_i \mathbf{W}^q)(\mathbf{x}_j \mathbf{W}^k)_{j \in S_i}^\top}{\sqrt{d_k}}\Big)(\mathbf{x}_j \mathbf{W}^v)_{j \in S_i}$$

In vanilla $S_i = \{j : j \leq i\}$

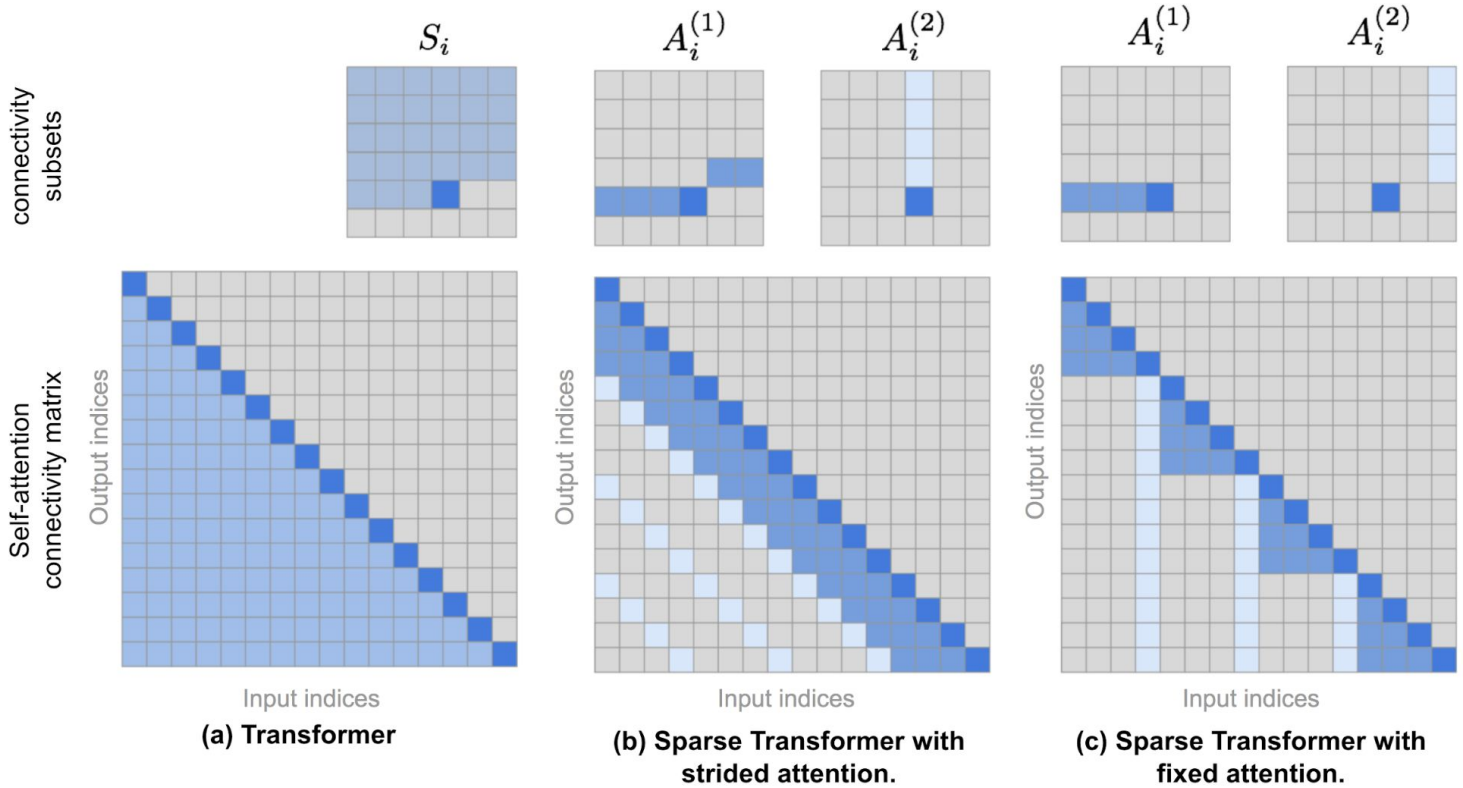
Sparse Attention Patterns

Strided Context

Idea to factorize self-attention by decomposing S_i into *tree* of dependencies

Precisely, the set S_i is divided into ***p*** *non-overlapping* subsets $A_i^{(m)}$. Therefore the path between the output position ***i*** and any ***j*** has max length of ***p+1***

Sparse Attention Patterns



Sparse Attention Patterns

Either single head with

$$\text{attn}(\mathbf{X}) = \text{Attend}(\mathbf{X}, A^{(n \mod p)})\mathbf{W}^o$$

$$\text{attn}(\mathbf{X}) = \text{Attend}(\mathbf{X}, \cup_{m=1}^{p} A^{(m)})\mathbf{W}^o$$

Or multi head which do one of the above

Sparse Attention Patterns

Sparse Transformer also proposed a set of changes so as to train the Transformer up to hundreds of layers, including gradient checkpointing, recomputing attention & FF layers during the backward pass, mixed precision training, efficient block-sparse implementation, etc

Blockwise Attention

Each attention matrix of size *L x L* is partitioned into *n x n* smaller blocks

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

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

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

Blockwise Attention

Actual implementation only stores QKV as block matrices

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

$\mathbf{q}_i \mathbf{k}_{\pi(i)}^\top \in \mathbb{R}^{\frac{N}{n} \times \frac{N}{n}}$ therefore memory complexity reduces from $\mathcal{O}(L^2)$ to $\mathcal{O}(L^2/n)$

Global-Local Attention of ETC

Idea to take two **local** input and **global** input, which is much smaller. Then compute 4 attention g2g, g2l, l2g, l2l. l2l can be very large, so it is restricted to fixed attention span

$$a_{ij}^{g2g} = \frac{1}{\sqrt{d}} x_i^g \mathbf{W}^Q (x_j^g \mathbf{W}^K + P_{ij}^K)^\top - (1 - M_{ij}^{g2g}) C$$

$$A_{ij}^{g2g} = \frac{\exp(a_{ij}^{g2g})}{\sum_{k=1}^{n_g} \exp(a_{ik}^{g2g})} \qquad z_i^g = \sum_{j=1}^{n_g} A_{ij}^{g2g} x_j^g \mathbf{W}^V$$

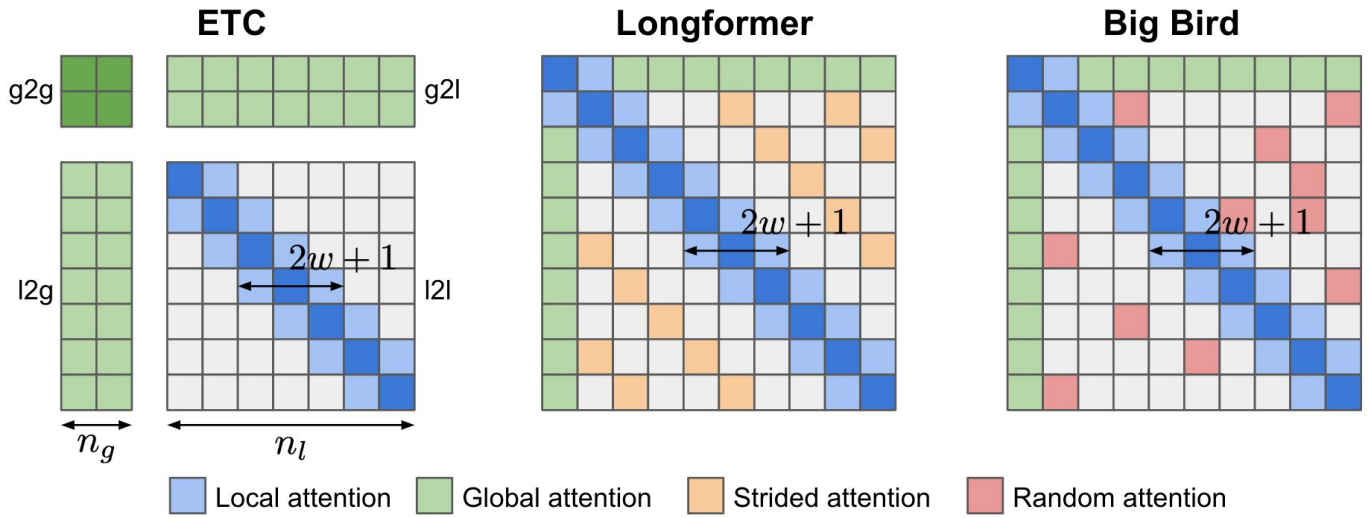
P_{ij}^K is a learnable relative encoding and
C is a very large constant ($C = 10000$ in the paper)

Global-Local Attention of ETC

Global input is constructed as follows: Assuming there are some segments within the long inputs (e.g. by sentence), each segment is attached with one auxiliary token to learn global inputs

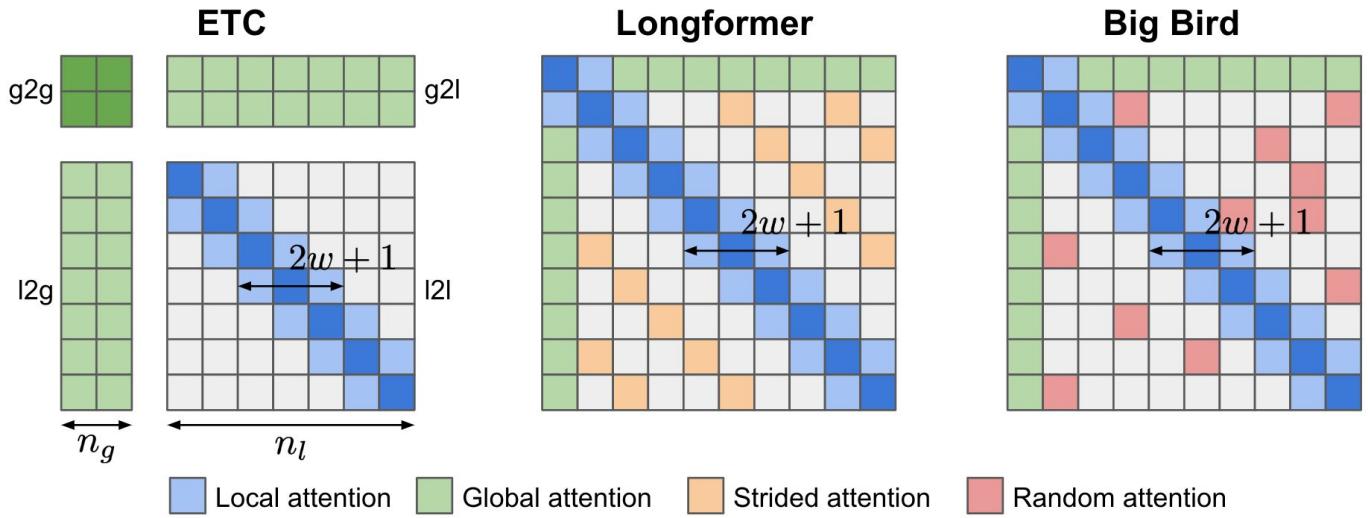
Global-Local Attention of Longformer

1. Local attention: like in ETC
2. GLobal attention of preselected tokens: like [CLS]
3. Dilated attention



Global-Local Attention of Big Bird

Big Bird replaces dilated attention with random tokens. Idea is attention can be viewed as graph and random graph has the property that information is able to rapidly flow between any pair of nodes



Content-based Attention

1. Quadratic time and memory complexity is a pain
2. Memory in a model with *N* layers is *N*-times larger
3. Feed Forward layers are often quite large

Reformer proposed two main changes

1. Funny dot-product (LSH) with less complexity
2. Replace the standard residual blocks with reversible residual layers, which allows storing activations only once during training instead of N

Content-based Attention

Funny dot-product a.k.a. Locality-Sensitive Hashing (LSH) comes from idea that we only care about largest elements in attention matrix

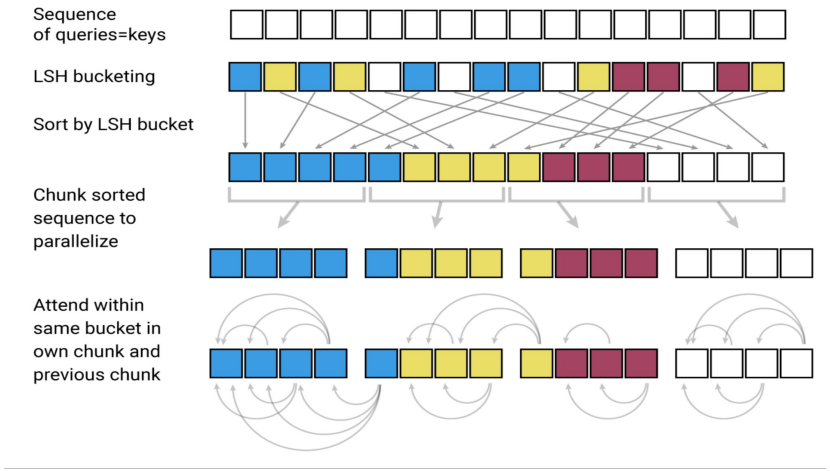
$$\text{A hashing scheme } x \to h(x) \text{ is } \textit{locality-sensitive} \text{ if it preserves the distance}$$

$$\text{Define } h(x) = \arg\max([xR; -xR]) \text{ where } [\cdot;\cdot] \text{ is concatenation}$$

Content-based Attention

The algo:

1. Get hash of each query
2. Sort keys and queries with respect to LSH value
3. Set $\mathbf{W}^{Q} = \mathbf{W}^{K}$. Shared QK does not affect performance
4. Apply batching



Content-based Attention

Reversible Residual Network.

Core idea that activations at any given layer can be recovered from the activations at the following layer with only model params

Vanilla residual does $y = x + F(x)$ Reversible residual does $(x_1, x_2) \rightarrow (y_1, y_2)$

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

For transformers let ***F*** be Attention and ***G*** be FeedForward

Low-Rank Attention

Linformer approximates the full attention with *low rank matrix*

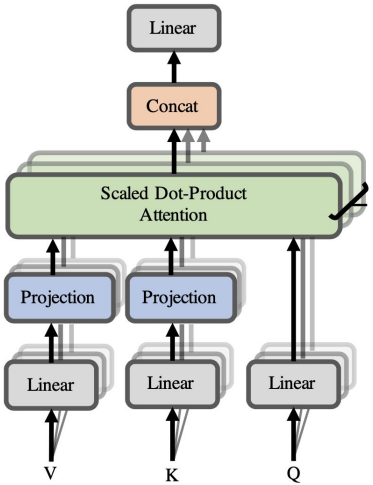
$$\overline{\text{head}_i} = \text{attn}(\mathbf{X}_q\mathbf{W}_i^q, \mathbf{E}_i\mathbf{X}_k\mathbf{W}_i^k, \mathbf{F}_i\mathbf{X}_v\mathbf{W}_i^v)$$

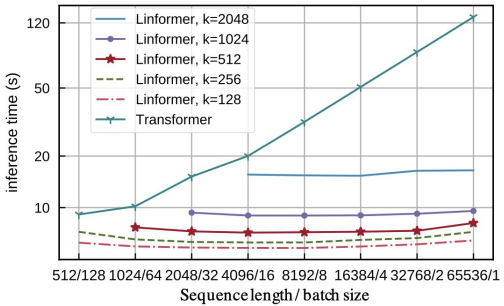
$$= \underbrace{\text{softmax}\left(\frac{\mathbf{X}_q\mathbf{W}_i^q(\mathbf{E}_i\mathbf{X}_k\mathbf{W}_i^k)^\top}{\sqrt{d}}\right)}_{\text{low rank attention matrix } \bar{A} \in \mathbb{R}^{k\times d}} \mathbf{F}_i\mathbf{X}_v\mathbf{W}_i^v$$

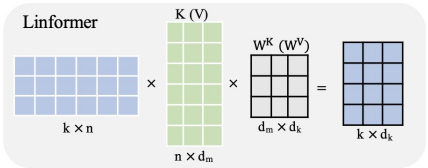
$$\mathbf{E}_i, \mathbf{F}_i \in \mathbb{R}^{L\times k}$$

Low-Rank Attention

- Share params between projection layers
- Different ***k*** for different layers (smaller ***k*** for higher layers)
- Different projections like mean/max pooling, convolutions







Low-Rank Attention

Performer uses funny approximation to optimize softmax

The main theorem behind RFA is from Rahimi & Recht, 2007:

> Let $\phi : \mathbb{R}^d \to \mathbb{R}^{2D}$ be a nonlinear transformation:
>
> $$\phi(\mathbf{x}) = \frac{1}{\sqrt{D}}[\sin(\mathbf{w}_1^\top \mathbf{x}), \dots, \sin(\mathbf{w}_D^\top \mathbf{x}), \cos(\mathbf{w}_1^\top \mathbf{x}), \dots, \cos(\mathbf{w}_D^\top \mathbf{x})]^\top$$
>
> When d-dimensional random vectors $\mathbf{w}_i$ are i.i.d. from $\mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_d)$,
>
> $$\mathbb{E}_{\mathbf{w}_i}[\phi(\mathbf{x}) \cdot \phi(\mathbf{y})] = \exp(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2})$$

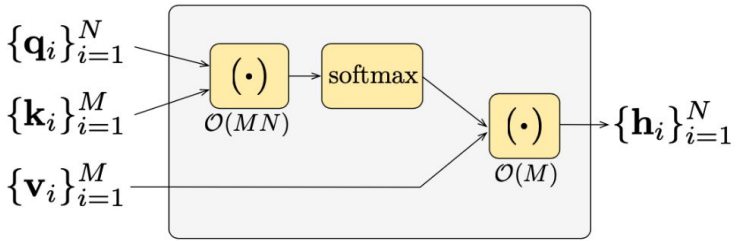
An unbiased estimation of $\exp(\mathbf{x} \cdot \mathbf{y})$ is:

$$\begin{aligned}
\exp(\mathbf{x} \cdot \mathbf{y} / \sigma^2) &= \exp(\frac{1}{2\sigma^2}(\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - \|\mathbf{x} - \mathbf{y}\|^2) \\
&= \exp(\frac{\|\mathbf{x}\|^2}{2\sigma^2}) \exp(\frac{\|\mathbf{y}\|^2}{2\sigma^2})(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}) \\
&\approx \exp(\frac{\|\mathbf{x}\|^2}{2\sigma^2}) \exp(\frac{\|\mathbf{y}\|^2}{2\sigma^2})\ \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) \\
&= \exp(\frac{1}{\sigma^2})\ \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) && \text{; unit vectors}
\end{aligned}$$

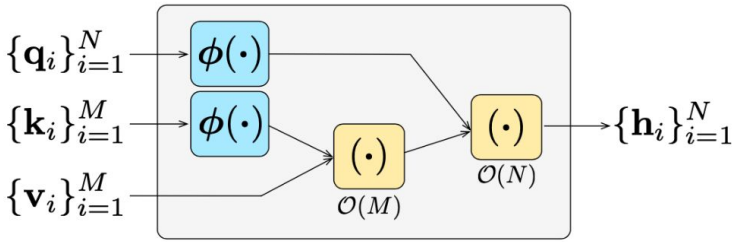
Low-Rank Attention

Then we can write the attention function as follows, where $\otimes$ is outer product operation and σ^2 is the temperature:

$$\mathrm{attn}(\mathbf{q}_t, \{\mathbf{k}_i\}, \{\mathbf{v}_i\}) = \sum_i \frac{\exp(\mathbf{q}_t \cdot \mathbf{k}_i/\sigma^2)}{\sum_j \exp(\mathbf{q}_t \cdot \mathbf{k}_j/\sigma^2)} \mathbf{v}_i^\top \approx \sum_i \frac{\phi(\mathbf{q}_t)\phi(\mathbf{k}_i)\mathbf{v}_i^\top}{\sum_j \phi(\mathbf{q}_t)\phi(\mathbf{k}_j)}$$
$$= \frac{\phi(\mathbf{q}_t)^\top \sum_i \phi(\mathbf{k}_i) \otimes \mathbf{v}_i}{\phi(\mathbf{q}_t)^\top \sum_j \phi(\mathbf{k}_j)} = \mathrm{RFA}(\mathbf{q}_t, \{\mathbf{k}_i\}, \{\mathbf{v}_i\})$$



(a) Softmax attention.



(b) Random feature attention.

Low-Rank Attention

Causal Attention RFA has token at time step t only attend to earlier keys and values $\{\mathbf{k}_i\}_{i\le t}, \{\mathbf{v}_i\}_{i\le t}$. Let us use a tuple of variables, $(\mathbf{S}_t \in \mathbb{R}^{2D\times d}, \mathbf{z} \in \mathbb{R}^{2D})$, to track the hidden state history at time step t, similar to RNNs:

$$\text{causal-RFA}(\mathbf{q}_t, \{\mathbf{k}_i\}_{i\le t}, \{\mathbf{v}_i\}_{i\le t}) = \frac{\phi(\mathbf{q}_t)^\top \mathbf{S}_t}{\phi(\mathbf{q}_t)\cdot \mathbf{z}_t}$$
$$\text{where } \mathbf{S}_t = \mathbf{S}_{t-1} + \phi(\mathbf{k}_t)\otimes \mathbf{v}_t, \quad \mathbf{z}_t = \mathbf{z}_{t-1} + \phi(\mathbf{k}_t)$$

where $2D$ is the size of $\phi(.)$ and D should be no less than the model size d for reasonable approximation.