CSE 493s/599s
Lecture 3. Self-attention and Transformers

Sewoong Oh

Lecture notes

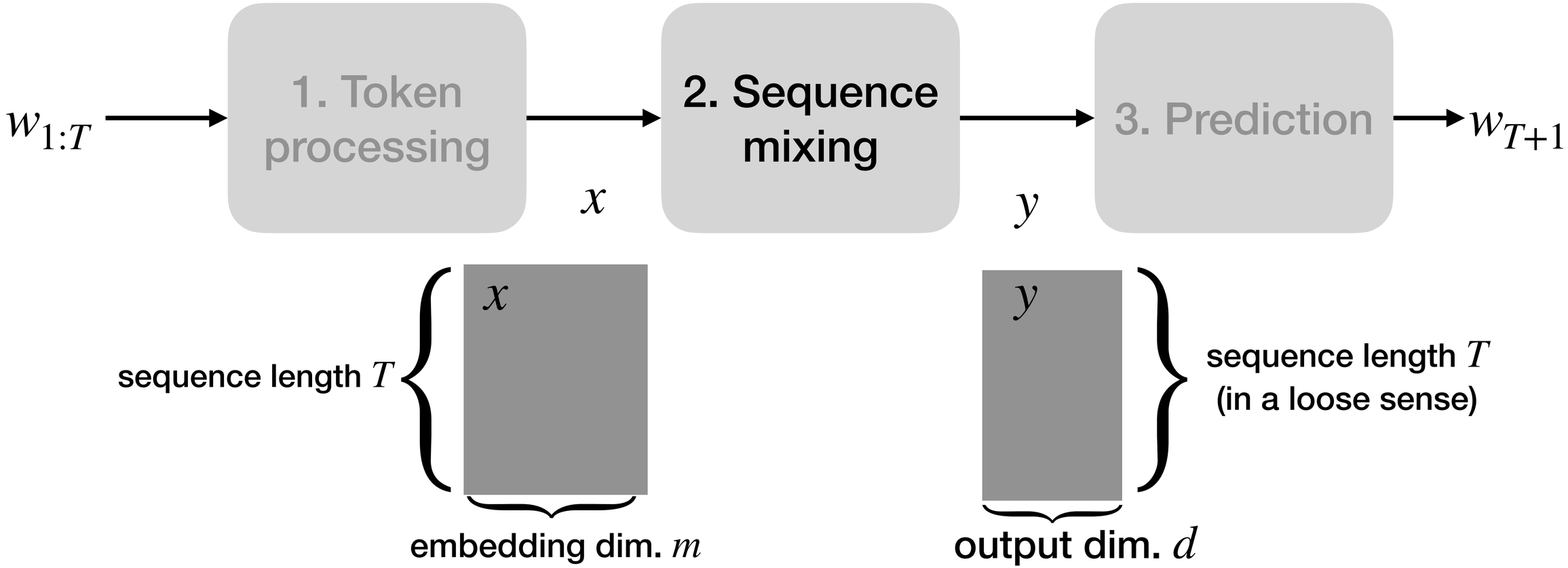
- These lecture notes are based on other courses in LLMs, including
  - CSE493S/599S at UW by Ludwig Schmidt: https://mlfoundations.github.io/advancedml-sp23/
  - EE-628 at EPFL by Volkan Cevher: https://www.epfl.ch/labs/lions/teaching/ee-628-training-large-language-models/ee-628-slides-2025/
  - ECE381V Generative Models at UT Austin by Sujay Sanghavi
  - and various papers and blogs cited at the end of the slide deck

Outline

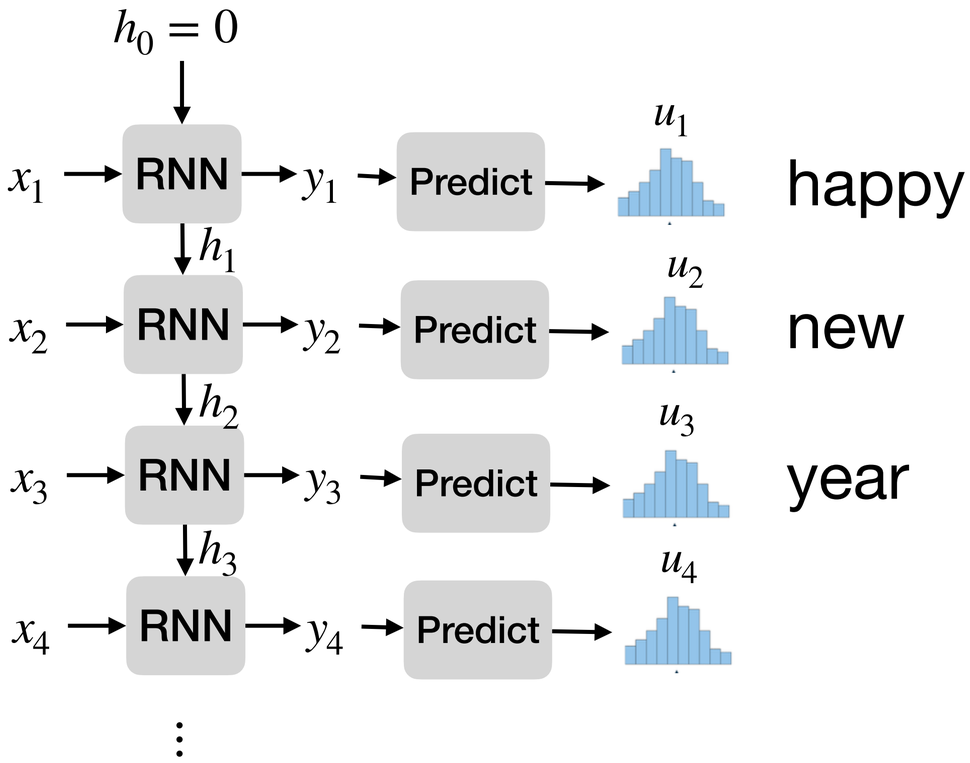
- Language models
- General LLM framework
  - Token processing
  - Sequence mixing
  - Prediction
- Example architectures

Sequence mixing

- Sequence mixing is one of the most well studied part of an LM that captures the dependencies across tokens.



Why do RNNs struggle?

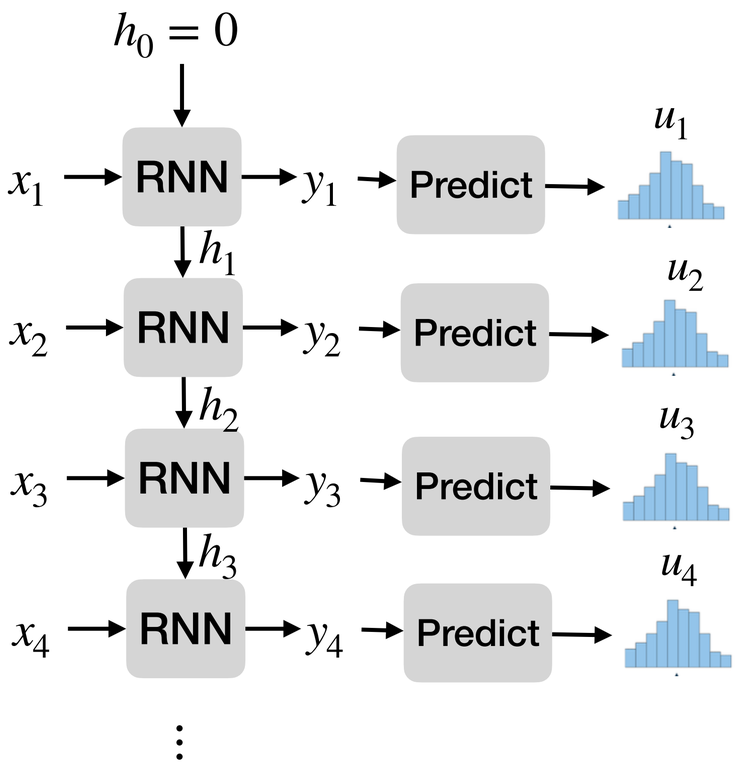


happy

new

year

Why RNNs struggle



- long range dependence gets diluted as the hidden state transitions multiple times
  - if there is a long range dependence between x_1 and x_{1000}, then this information needs to be carried over 1000 hidden states
- long sequences have vanishing or exploding gradients

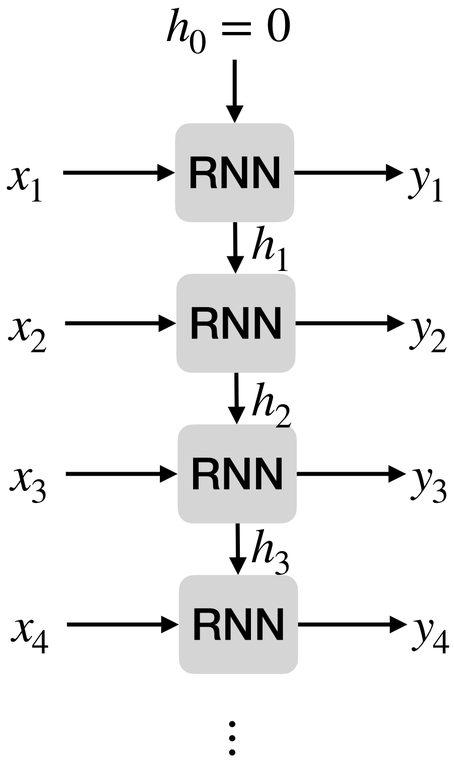
  [Pascanu et al. 2013, Hochreiter et al. 1997],

  $\nabla_{W_{\text{block 1}}} \text{X-entropy}(u_{100}, x_{101}) = J_W h_1 \times J_{h_1} h_2 \cdots \nabla_{h_{99}} \text{X-entropy}(u_{100}, x_{101})$
- mode collapse (i.e., generating repetitive outputs),

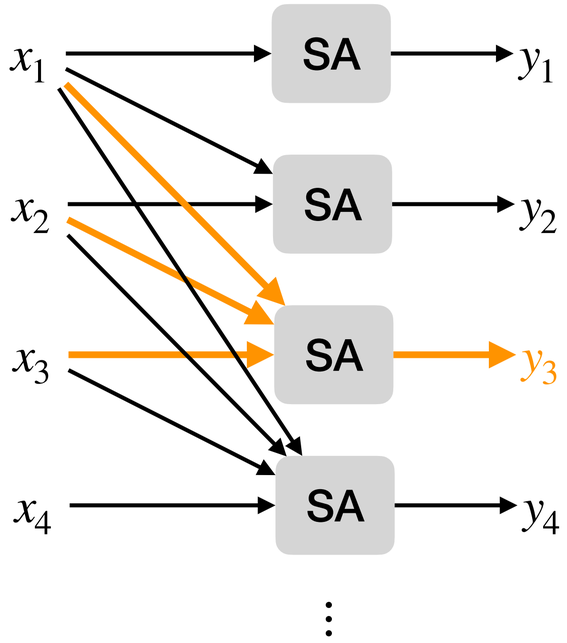
- struggle with highly variable input sizes due to limited memory in the hidden state, acting as a bottleneck.
  - the dimension is fixed a priori, e.g., $h \in \mathbb{R}^{200}$

- **Self-attention** [Vaswani et al. 2017] as a sequence mixer addresses the shortcomings of **RNNs** with different trade-offs for inference cost.

RNNs are sequential

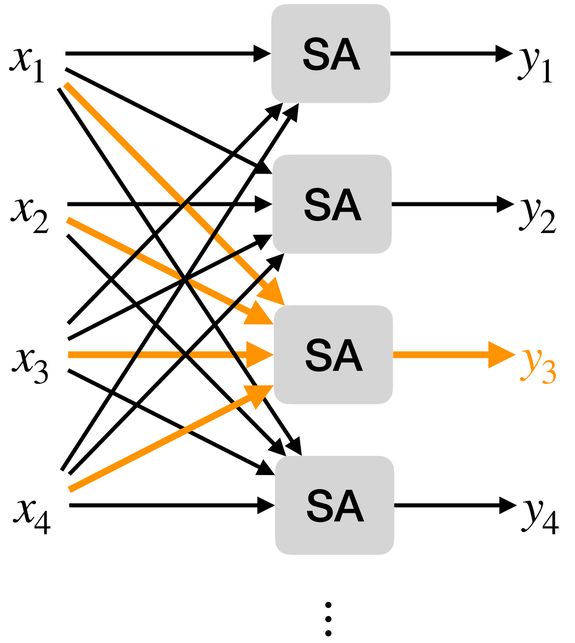


Causal structure



Self-attention is parallel

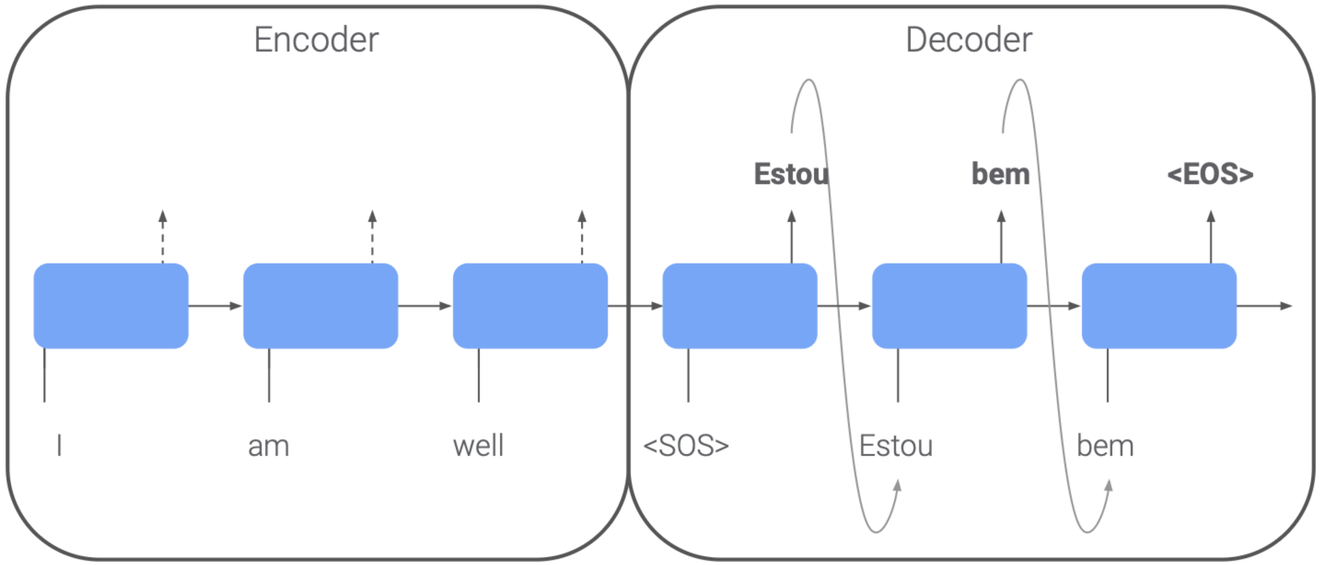
Non-causal structure



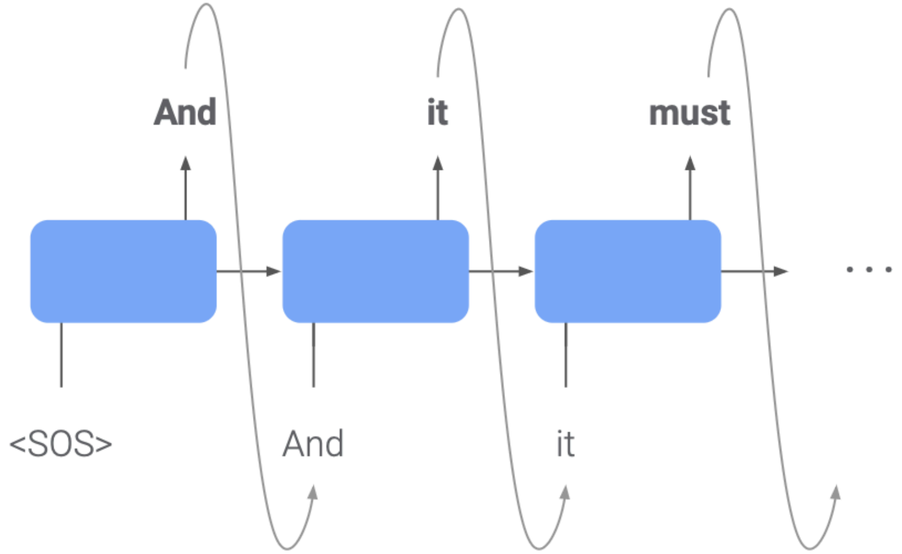
- Notice how we focus on a version of self-attention blocks that has a **causal structure**, i.e., the output only depends on the previous words. Both causal and non-causal are used depending on the application.

- **Causal vs. non-causal** dependencies determined by the user who chooses the architecture based on the application of interest.
- Original transformer paper is on **machine translation**, where
  - "**encoder**" does not require causality, as its goal is to produce a vector representation of the entire sentence.
  - "**decoder**" requires causality, as it generates a sequence autoregressively.

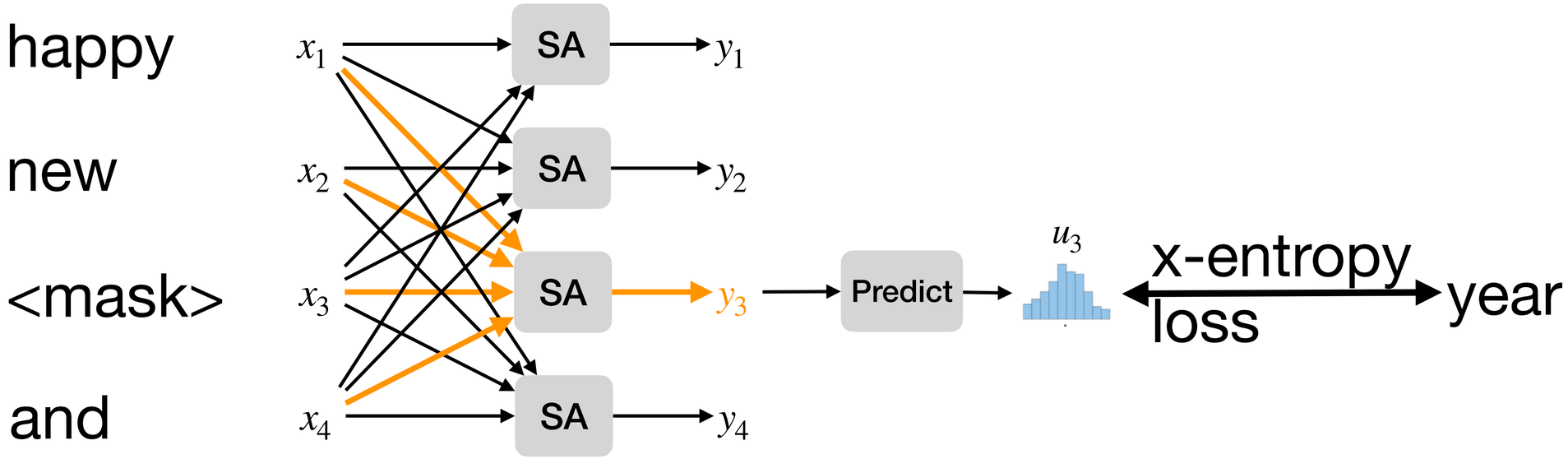
Machine translation:
encoder-decoder model
non-causal encoder, causal decoder



Next word prediction:
decoder-only model
causal decoder



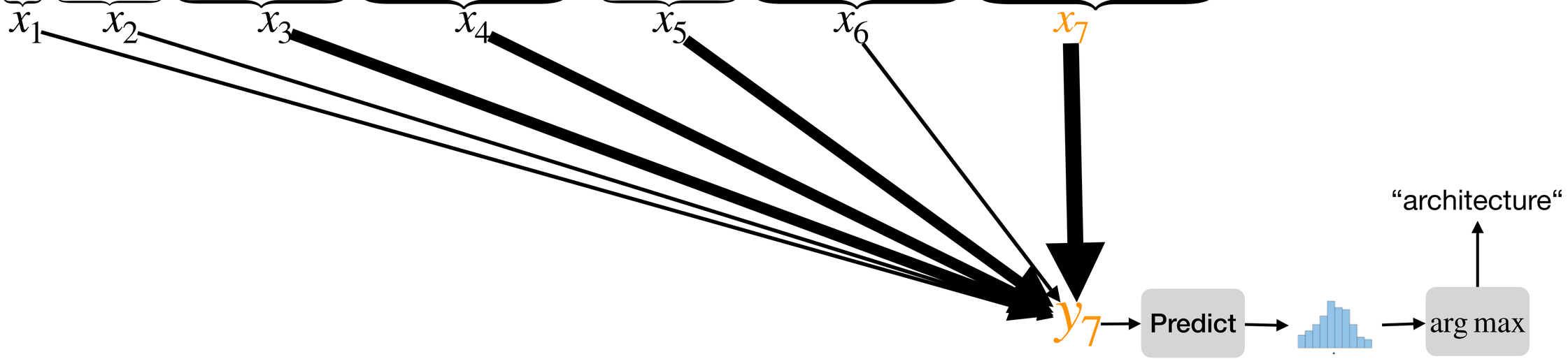
- Earlier language models, like BERT and RoBERTa, use what is called “**masked**” **language model**, where in training a random word is masked in a sentence and the model predicts it. Obviously, this is **not causal** i.e. a word in position i can attend to words both in the past and in the future. So non-causal self-attention was used.



- Modern LLMs, like GPT and Llama, are trained on **next-word prediction** tasks, which is obviously **causal**, since non-causal next-word prediction is a trivial. You just output the next token. As such, we focus in our lectures on **causal structured self-attention mechanisms**.

- The goal of a sequence mixer is to generate the representation of a word in a sequence that captures the true meaning of the word in the context the word is used in. For example, **“transformer”** can mean many things, but in this context:

“To solve machine translation, Google introduced **transformer** architecture …”



- Self-attention achieves this by a **weighted combination** of the vector embeddings of the other words, with more weight on a word that might be more relevant to the word of interest, say, “transformer”, i.e.,

$$y_{\text{transformer}} = c \cdot f(x_{\text{transformer}}) + \sum_{i<7} \alpha_{t;i} \cdot f(x_i) = \sum_{i=1}^{t} \underbrace{\alpha_{t;i}}_{\text{pairwise relevance}} \cdot f(x_i)$$

- Self-attention achieves this by a **weighted combination** of the vector embeddings of the other words,

$$y_t = \sum_{i=1}^{t} \underbrace{\alpha_{t;i}}_{\text{pairwise relevance}} \cdot f(x_i)$$

- Note that the relevance/weight $\alpha_{t;i}$ is **asymmetric**, since x_t is the target and x_i is the context.
- The main question in designing a new architecture is, how do we parametrize $\alpha_{t;i}$'s and $f(x_i)$'s?
  - Answer: **self-attention** and **transformers**
- Drawing inspiration from Word2vec, here is a **first attempt**:
  - Let $\alpha_{t,i} = x_t^T x_i = \langle x_t, x_i \rangle$, and $f(x_i) = x_i$

$$y_t = \sum_{i=1}^{t} \underbrace{\langle x_t, x_i \rangle}_{\alpha_{t;i}} \cdot \underbrace{x_i}_{f(x_i)}$$

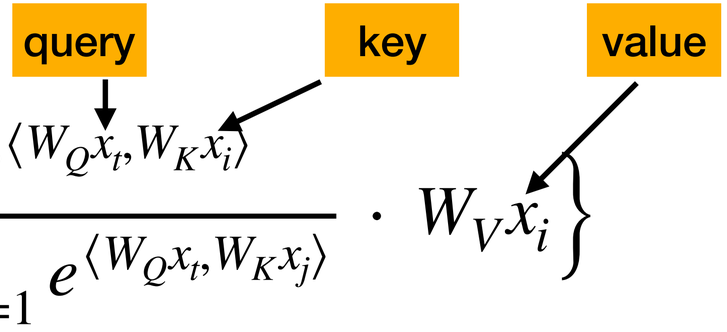
- Issue: $\alpha_{t;j}$ have uncontrollable scale, and the output y_t's could have arbitrary magnitude

- Drawing inspiration from Word2vec, here is a **second attempt**: use **SoftMax**
  - Let $\alpha_{t,i} = \text{Softmax}\big([\langle x_t, x_1 \rangle, \langle x_t, x_2 \rangle, \cdots, \langle x_t, x_t \rangle] \big)_i$, and $f(x_i) = x_i$, where

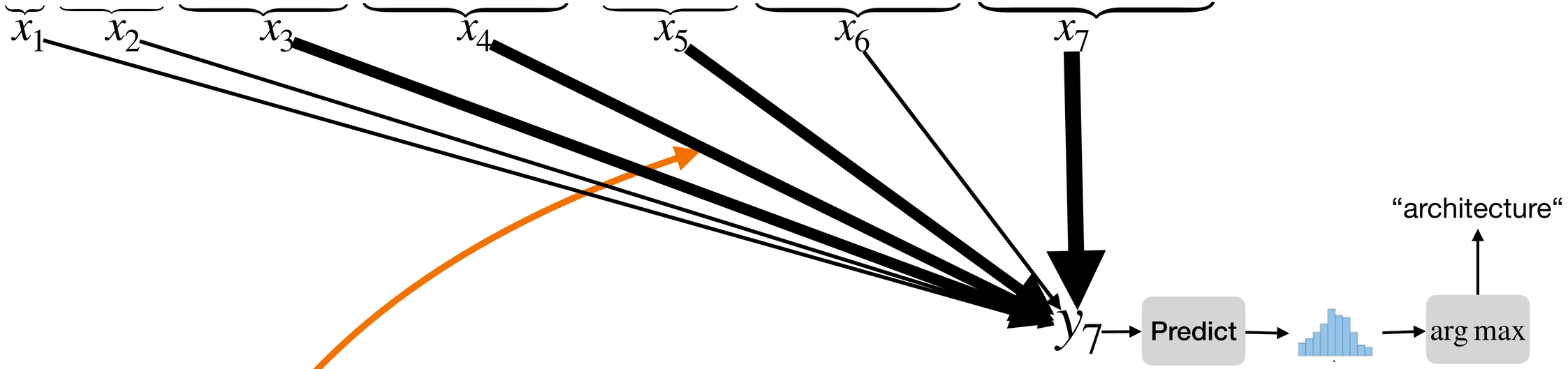
$$\text{Softmax}([a_1, a_2, \cdots, a_t]) = \frac{1}{\sum_{j=1}^{t} e^{a_j}} [e^{a_1}, e^{a_2}, \cdots, e^{a_t}]$$

$$y_t = \sum_{i=1}^{t} \Bigg\{ \underbrace{\frac{e^{\langle x_t, x_i \rangle}}{\sum_{j=1}^{t} e^{\langle x_t, x_j \rangle}}}_{\alpha_{t;i}} \cdot \underbrace{x_i}_{f(x_i)} \Bigg\}$$

- This use of Softmax ensures that $\alpha_{t;i}$'s are non-negative and sum to one.
- **Self-attention** includes linear transform of the token embedding x_j's with learnable parameters (W_Q, W_K, W_V) to give the freedom to learn the right representation of the tokens, depending on its role as **query**, **key**, or **value**

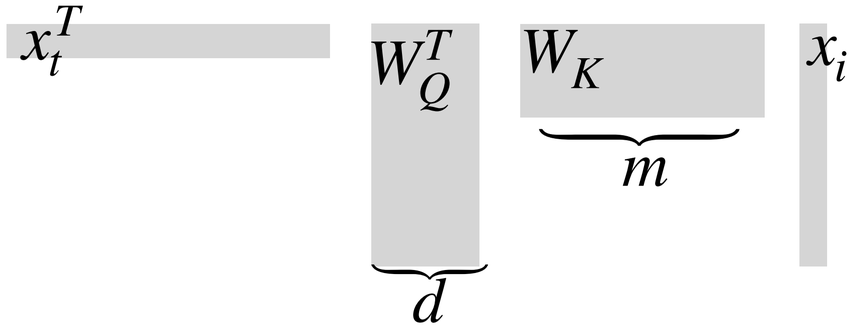


- “To solve machine translation, Google introduced **transformer** architecture …”



- To measure the similarity/relevance **key-query-value self-attention mechanism** uses **queries, keys, and values** with **learnable** parameters $W_Q, W_K, W_V \in \mathbb{R}^{d \times m}$. For modern language models, like GPT-4, the embedding dimension is $m = 12288$ and the query/key dimension is $d = 128$, such that the **query vector** for the target word is $q_t = W_Q x_t$, the **key vector** is $k_i = W_K x_i$, and the relevance is then

$$\text{Rel}(x_t; x_i) = \langle q_t, k_i \rangle = x_t^T W_Q^T W_K x_i$$



- $\mathrm{Rel}(x_t; x_i) = \langle q_t, k_i \rangle$ can be computed, for example, as a matrix, forcing upper-right triangle region to be $-\infty$, meaning no relevance, for **causality**. Output of self-attention can only depend on the past.

Query \ Key	To (x_1)	solve (x_2)	machine (x_3)	translation (x_4)	Google (x_5)	introduced (x_6)	transformer (x_7)
To	<q1,k1>	$-\infty$					
solve	<q2,k1>	<q2,k2>					
machine	<q3,k1>	<q3,k2>	<q3,k3>				
translation	<q4,k1>	<q4,k2>	<q4,k3>	<q4,k4>			
Google	<q5,k1>	<q5,k2>	<q5,k3>	<q5,k4>	<q5,k5>		
introduced	<q6,k1>	<q6,k2>	<q6,k3>	<q6,k4>	<q6,k5>	<q6,k6>	
transformer	<q7,k1>	<q7,k2>	<q7,k3>	<q7,k4>	<q7,k5>	<q7,k6>	<q7,k7>
…							

- $\langle q_{\text{transformer}}, k_{\text{Google}} \rangle$ might have a large positive value indicating that these terms are highly related, whereas $\langle q_{\text{transformer}}, k_{\text{to}} \rangle$ might be nearly zero or even negative.

- For each row, we take the Softmax of the similarity to ensure that the weights are non-negative and sum to one, calling the previous Query-Key matrix Rel,

$$\alpha_{t;i} = \text{Softmax}(t\text{-th row of } \mathrm{Rel})_i = \mathrm{Softmax}(\{\langle q_t, k_j \rangle\}_{j=1}^{t})_i = \frac{e^{\langle q_t, k_i \rangle}}{\sum_{j=1}^{t} e^{\langle q_t, k_j \rangle}}$$

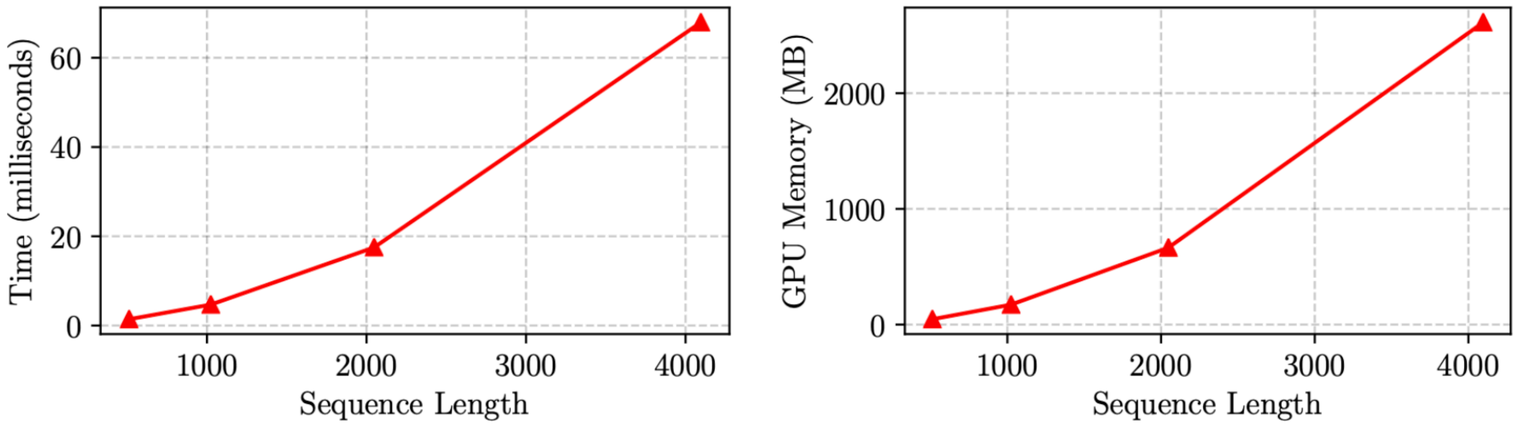
- The self-attention vector is the weighted sum of the **value** defined by a learnable parameter $W_V \in \mathbb{R}^{d \times m}$ such that $v_i = W_V x_i$, and

$$y_t = \sum_{i=1}^{t} \frac{e^{\langle q_t, k_i \rangle}}{\sum_{j=1}^{t} e^{\langle q_t, k_j \rangle}} v_i \ , \qquad \begin{bmatrix} y_1^T \\ \vdots \\ y_t^T \end{bmatrix} = \mathrm{Softmax}\Big(\underbrace{\Big(Q^T \quad K \Big) \odot M}_{\text{Query-Key matrix Rel}} \Big) V^T$$

- All y_t's can be compactly represented as $Y = \mathrm{Softmax}((Q^T K) \odot M) V^T$, where $Q = \{q_i = W_Q x_i\}_{i=1}^{T} \in \mathbb{R}^{d \times T}$, $K = \{k_i = W_K x_i\}_{i=1}^{T} \in \mathbb{R}^{d \times T}$, $V = \{v_i = W_V x_i\}_{i=1}^{T} \in \mathbb{R}^{d \times T}$, and $M_{t,i} = \begin{cases} 1, & t \geq i \\ -\infty, & t < i \end{cases}$ is the causal mask.

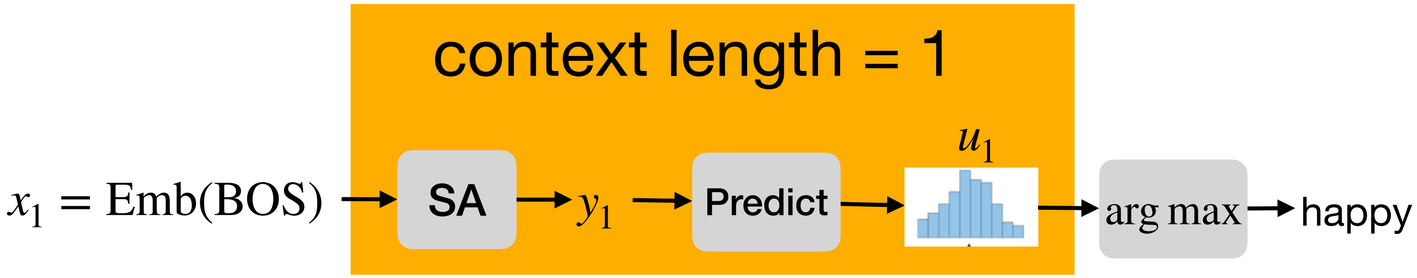
Takeaway

- The **causal mask** is necessary in training, in order to prevent “cheating” in the next word prediction loss.
- Attention with masking is called **masked attention** or **causal attention**.
- **Benefit of self-attention**: By allowing any word to attend to any other word in the sequence, this resolves long-distance dependence problem of RNNs.
- Self-attention has learnable parameters: $W_Q, W_K, W_V \in \mathbb{R}^{d \times m}$.
- **Cost of self-attention**: Note that the (masked) attention matrix is $T \times T$ dimension, and computation and memory scales like $O(T^2)$.



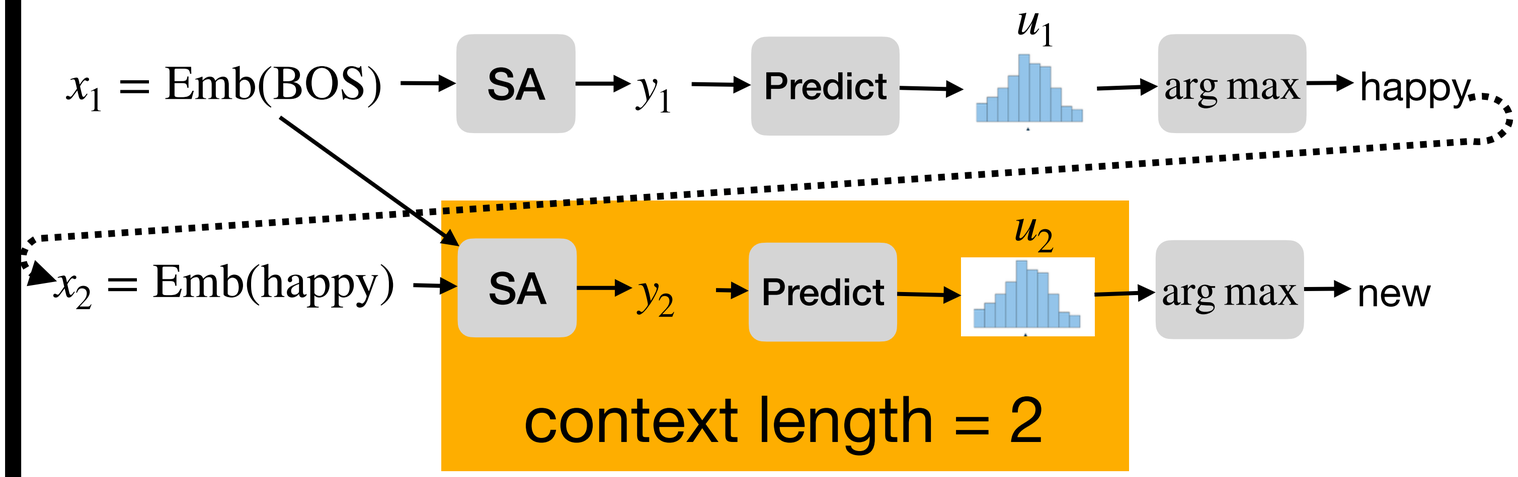
- Auto-regressive inference with self-attention

- Set x_1 as embedding of $\langle \text{BOS} \rangle$, t=1
- **While True:**
  - $q_t \leftarrow W_Q x_t, k_t \leftarrow W_K x_t, v_t \leftarrow W_V x_t$
  - compute score $s \leftarrow [q_t^T k_1, \cdots, q_t^T k_t,]^T$
  - $y_t \leftarrow [v_1, \cdots, v_t]\text{Softmax}(s)$
  - $x_{t+1} \leftarrow \text{Emb}(\arg\max_w u_t[w])$
  - If x_{t+1} is the embedding of $\langle \text{EOS} \rangle$: break
  - $t \leftarrow t + 1$
- **Output**: $[x_1, \cdots, x_{t+1}]$



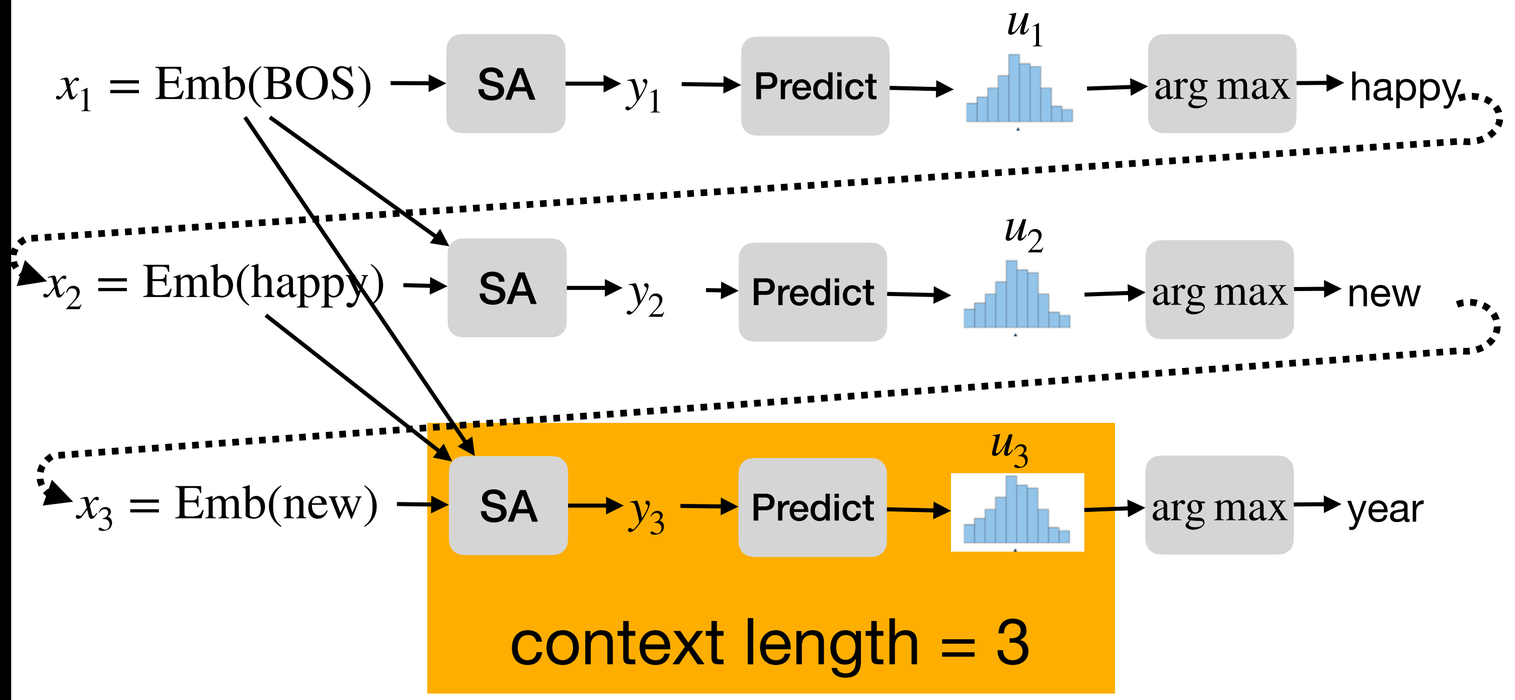
- Auto-regressive inference with self-attention

- Set x_1 as embedding of $\langle \text{BOS} \rangle$, t=1
- **While True:**
  - $q_t \leftarrow W_Q x_t, k_t \leftarrow W_K x_t, v_t \leftarrow W_V x_t$
  - compute score $s \leftarrow [q_t^T k_1, \cdots, q_t^T k_t,]^T$
  - $y_t \leftarrow [v_1, \cdots, v_t]\text{Softmax}(s)$
  - $x_{t+1} \leftarrow \text{Emb}(\arg\max_w u_t[w])$
  - If x_{t+1} is the embedding of $\langle \text{EOS} \rangle$: break
  - $t \leftarrow t + 1$
- **Output**: $[x_1, \cdots, x_{t+1}]$



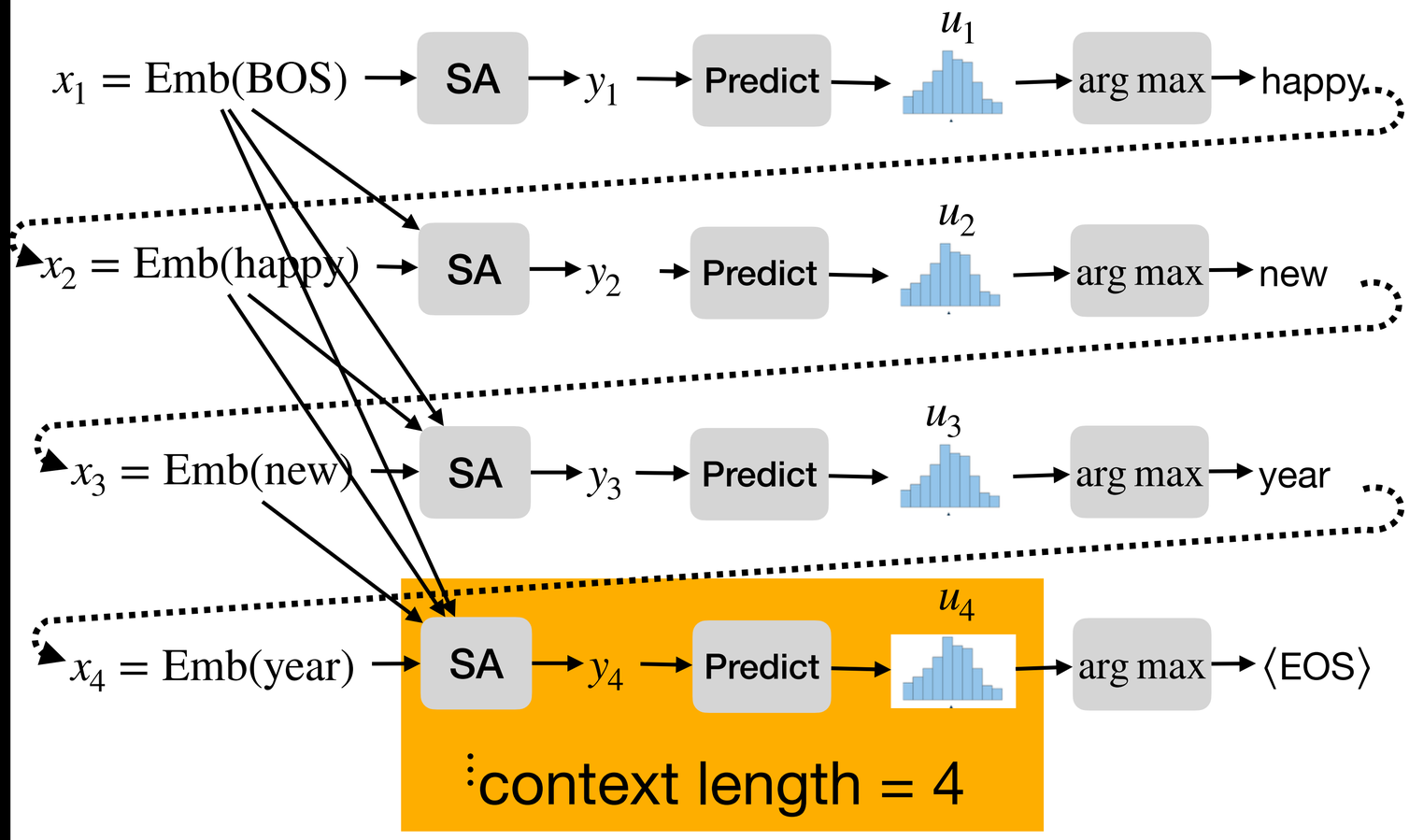
- Auto-regressive inference with self-attention

- Set x_1 as embedding of $\langle \text{BOS} \rangle$, t=1
- **While True:**
  - $q_t \leftarrow W_Q x_t, k_t \leftarrow W_K x_t, v_t \leftarrow W_V x_t$
  - compute score $s \leftarrow [q_t^T k_1, \cdots, q_t^T k_t,]^T$
  - $y_t \leftarrow [v_1, \cdots, v_t]\text{Softmax}(s)$
  - $x_{t+1} \leftarrow \text{Emb}(\arg\max_w u_t[w])$
  - If x_{t+1} is the embedding of $\langle \text{EOS} \rangle$: break
  - $t \leftarrow t + 1$
- **Output**: $[x_1, \cdots, x_{t+1}]$



- Auto-regressive inference with self-attention

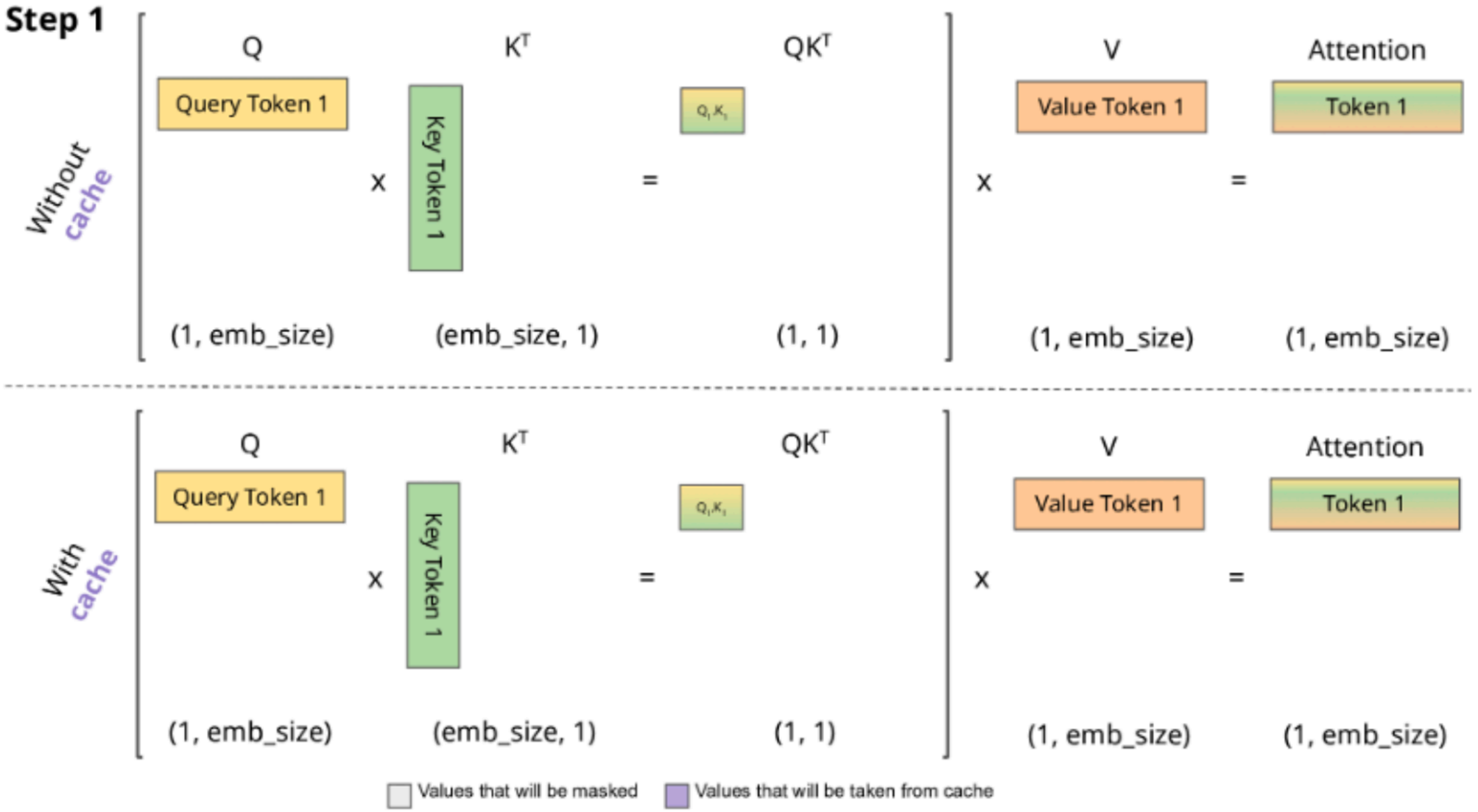
- Set x_1 as embedding of $\langle \text{BOS} \rangle$, t=1
- **While True:**
  - $q_t \leftarrow W_Q x_t, k_t \leftarrow W_K x_t, v_t \leftarrow W_V x_t$
  - compute score $s \leftarrow [q_t^T k_1, \cdots, q_t^T k_t,]^T$
  - $y_t \leftarrow [v_1, \cdots, v_t]\text{Softmax}(s)$
  - $x_{t+1} \leftarrow \text{Emb}(\arg\max_w u_t[w])$
  - If x_{t+1} is the embedding of $\langle \text{EOS} \rangle$: break
  - $t \leftarrow t + 1$
- **Output**: $[x_1, \cdots, x_{t+1}]$



- Note that this is still **auto-regressive** and **non-parallelizable** and hence slow.

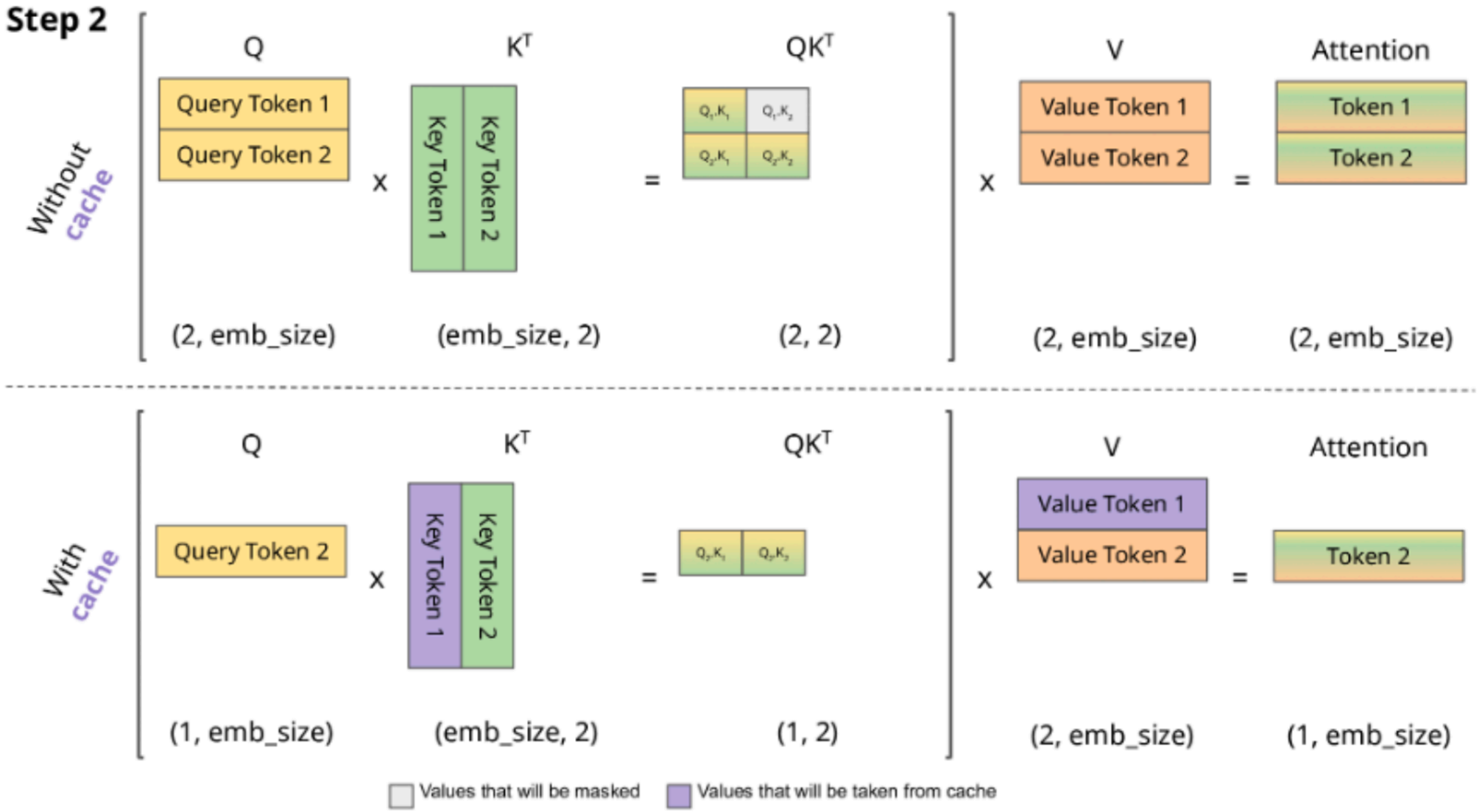
- How to speed up the computation of the attention heads is a very active research direction.
- **Definition** [KV cache]
  - **KV Cache** (Key-Value Cache) stores. computed keys (K) and values (V) from previous time steps to avoid recompilation in self-attention during autoregressive inference.
  - **KV Cache** does not change the model's output; it only avoids recomputing old keys and values during generation.
- How does KV Cache work?
  - $t = 1$: Compute and store k_1 and v_1.
  - $t = 2$: Retrieve k_1, v_1 and compute k_2, v_2 and append
  - for general t : retrieve all K, V and compute only key and value for the new word
- Naive implementation of self-attention recomputes K, V every step: $O(T^2 d)$.
- KV Cache reduces complexity to $O(Td)$ per step for faster **inference** with lower memory overhead. The idea is simple, and it is more about how to implement it in tensor operations. Training still takes $O(T^2 d)$ time, since it is not autoregressive.

- KV Cache



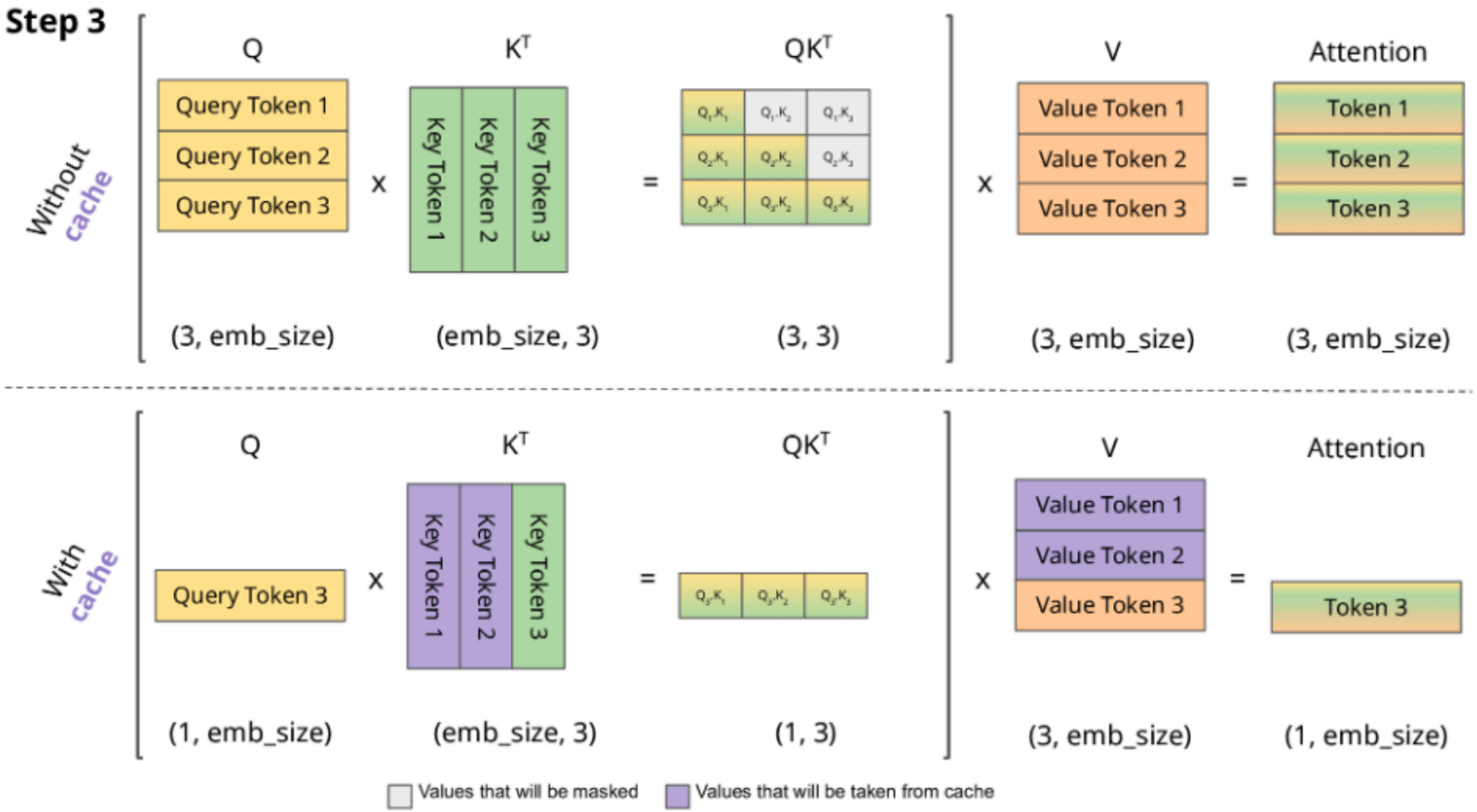
From https://medium.com/@joaolages/kv- caching- explained- 276520203249

- KV Cache



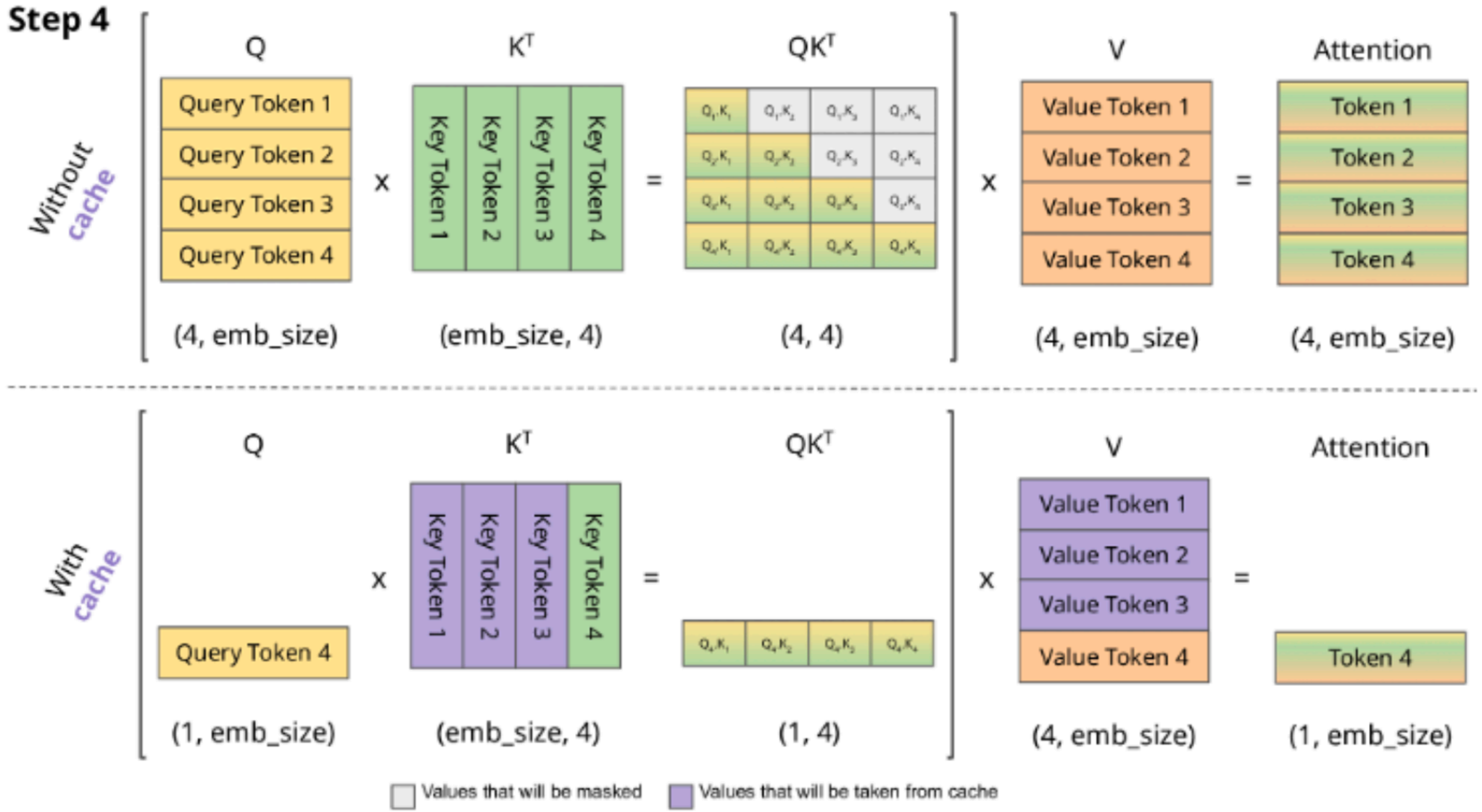
From https://medium.com/@joaolages/kv- caching- explained- 276520203249

- KV Cache



From https://medium.com/@joaolages/kv- caching- explained- 276520203249

- KV Cache

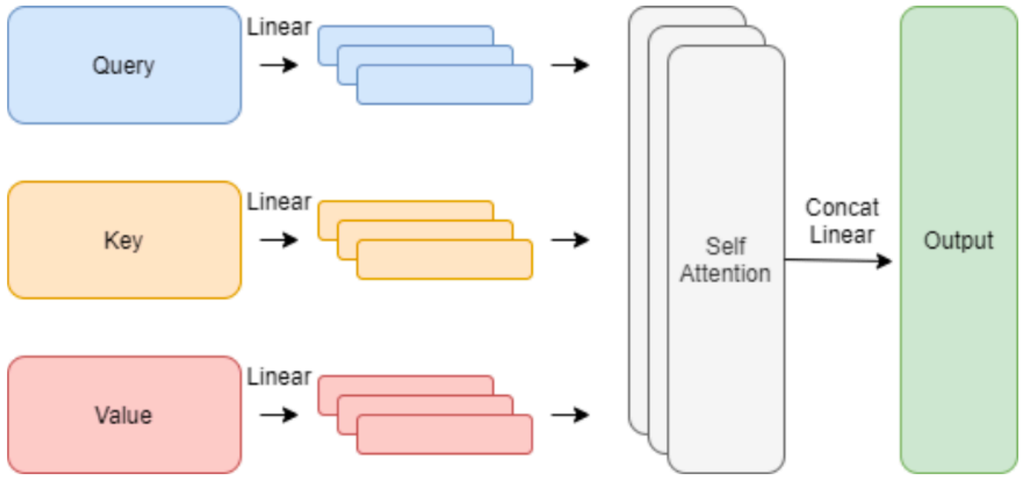


From https://medium.com/@joaolages/kv- caching- explained- 276520203249

- Another idea to speed up by parallelization is **multi-head attention.**
- **Definition** [Multi-head Attention]
  - Instead of having one attention, we can have h attention heads in parallel such that

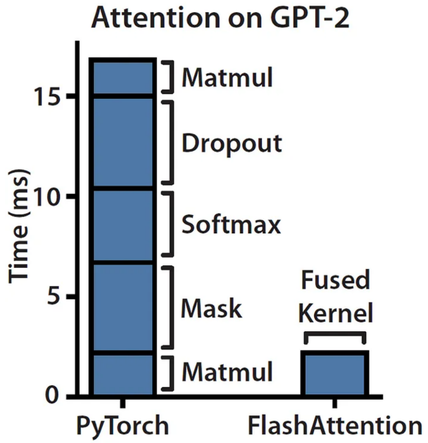
$$\text{MultiHead}(x) = \text{Concat}(\text{head}_1, \cdots, \text{head}_h),$$
$$\text{where head}_i = \text{Attention}\left(W_Q^{(i)}x, W_K^{(i)}x, W_V^{(i)}x\right).$$

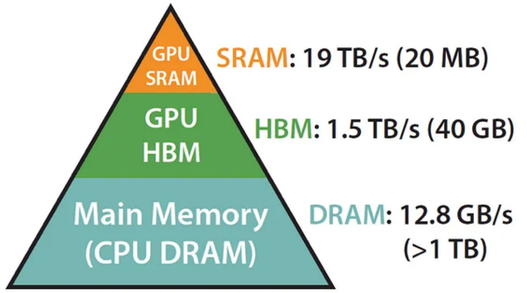
- Benefit 1: This division of heads allows parallelization.
- Benefit 2: Each head handles different aspect, e.g., subject-verb agreement, syntax, semantic relations, etc, enhancing the model's ability to capture diverse dependencies.



Advanced system optimization

- Matrix multiplications are **compute-bound**, i.e., run-time is dominated by computation, and element-wise operations are **memory-bound**, i.e., dominated by memory access
- Run-time of a transformer layer is dominated by **Dropout**, **Softmax**, **Mask** which are memory-bound.
- But memory is hierarchical. So the idea of **Flash attention** is to keep intermediate steps in SRAM (faster memory access) and only write the final result back to HBM (slower memory access).
- Use **tiling** to process in small blocks instead of full sequence.
- It is up to 2~4x faster with less memory footprint: $O(TCd)$ complexity where C is the size of the block.





Memory Hierarchy with Bandwidth & Memory Size

$$\text{Softmax}\left(\left(\underbrace{Q^T \quad K}_{\text{compute-bound}}\right) \odot M\right) V^T$$

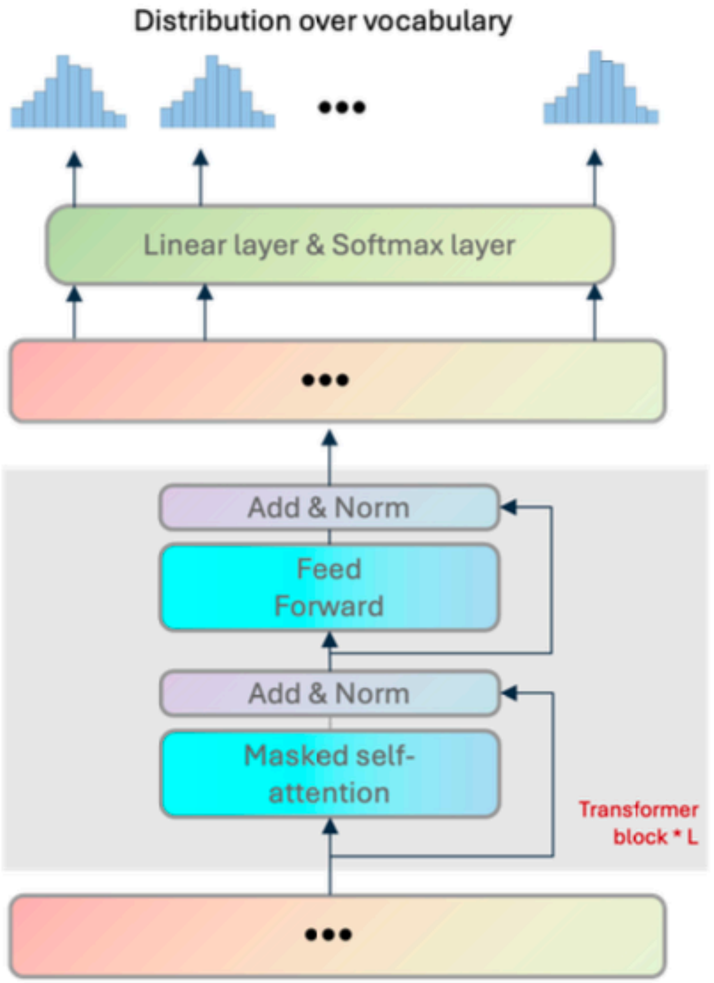
memory-bound

memory-bound

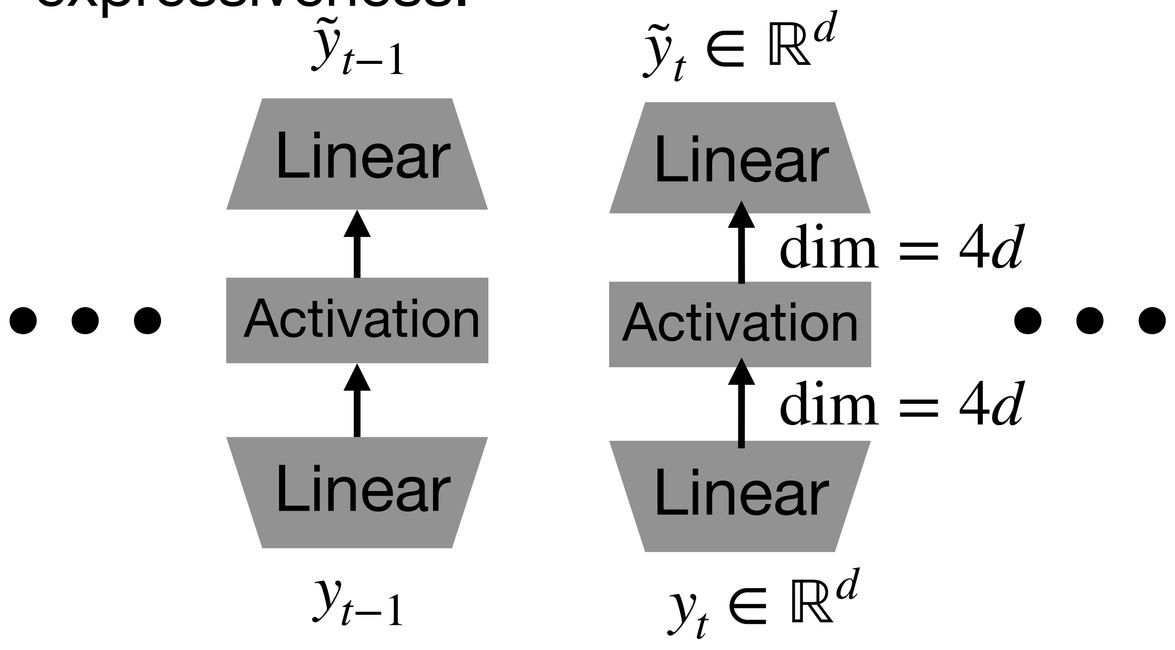
compute-bound

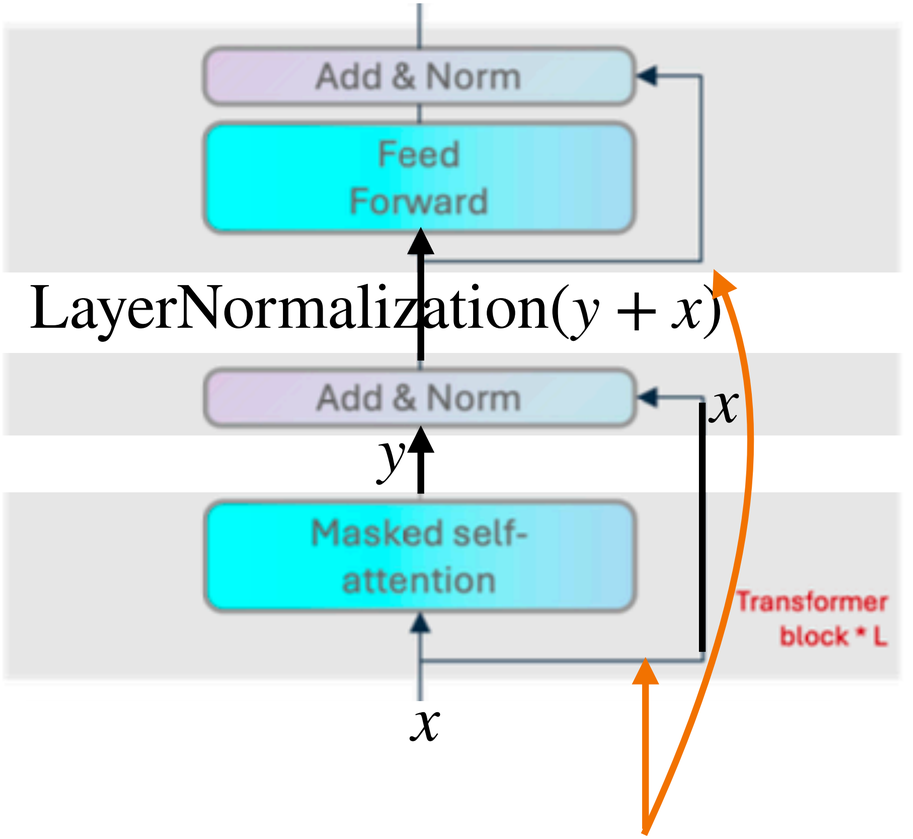
courtesy of https://gordicaleksa.medium.com/eli5-flash-attention-5c44017022ad

- Modern language models are made up of layers of **transformer blocks**.
- A transformer block is
  [Input ⇒multi-head self-attention ⇒ layer normalization ⇒ feedforward layer ⇒ layer normalization]
- GPT-2 has 12 heads and 12~48 layers.
- Original transformer is proposed with encoder and decoder for neural machine translation, but we only learned the transformer decoder which is sufficient as an LM.
- Let's look at each sub-block in detail.



- The role of **skip connection** is to prevent vanishing gradients (as we saw in the case of RNNs) by allowing gradients to flow directly from top layers to bottom layers.
    - It is motivated by **ResNet**'s success in image classification.
- The role of **Feed Forward Network** (applied to each token separately but shared weights) is to introduce additional non-linearity and improve expressiveness.



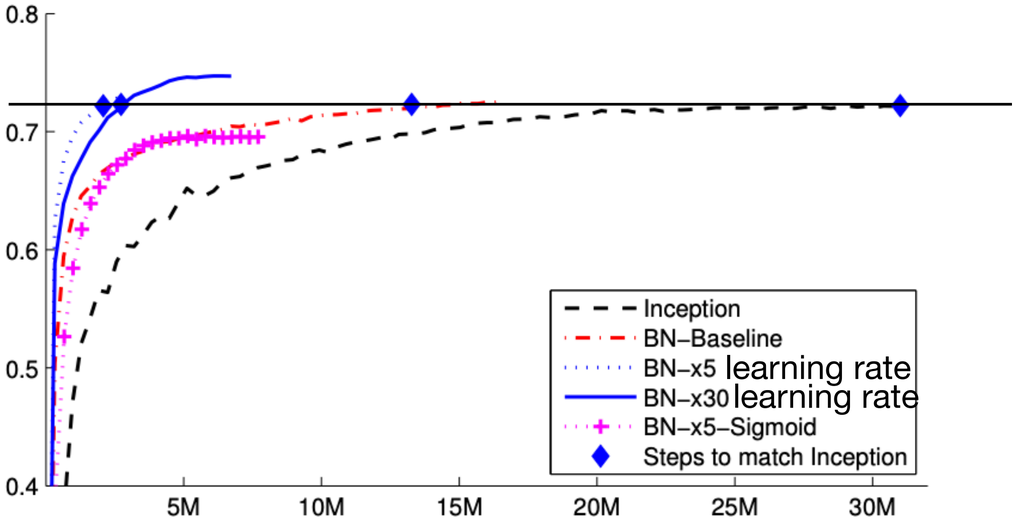


Skip connections

- To stabilize training, **Batch Normalization** and **Layer Normalization** are proposed.
- Batch normalization is initially proposed in [Ioffe et al. 2015] to mitigate a phenomena known as **internal covariate shift** by standardizing the input to each layer on the mini-batch that is being processed.
- **Covariate shift** is when the input distribution changes. **Internal covariate shift** is when the input distribution to a layer in a deep neural network changes, due to the sampling of a mini-batch.
- To mitigate such distribution changes, **batch normalization** standardizes the input to a layer: each input coordinate is subtracted its mini-batch mean and divided by the mini-batch standard deviation.
- This stabilizes the training, allowing larger step sizes and converging faster. At inference, moving average is used for the internal statistics.

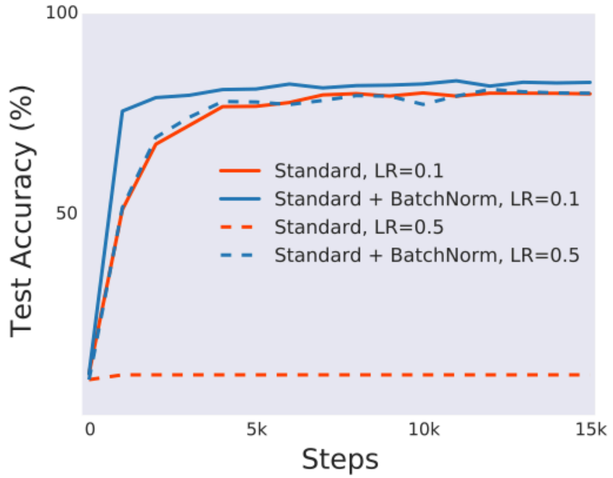
$$\mu_j = \frac{1}{B}\sum_{i=1}^{B} x_{i,j}, \sigma_j^2 = \frac{1}{B}\sum_{i=1}^{B}(x_{i,j} - \mu_j)^2$$

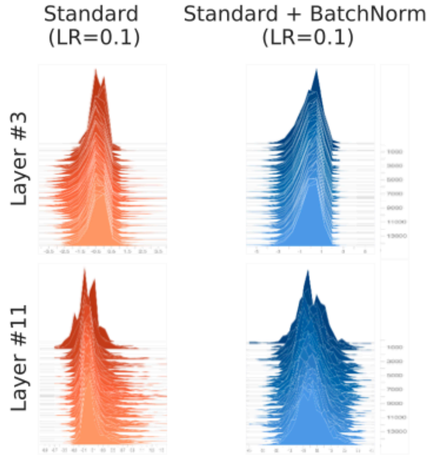
$$\tilde{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}, \text{ for sample } i, \text{ coordinate } j$$



- One caveat is that the output of batch normalization is coordinate-wise scaled and shifted by learnable parameters, which we omit in the explanation.

- Later research [Santurkar et al. 2018] has discovered that the success of batch normalization has little to do with internal covariate shifts.





- Instead, batch normalization makes the **landscape smooth**, making it easier for gradient based optimization methods to take larger steps.

- However, for LLM training, each sample is processed separately, and hence batch size is oftentimes one. Further, sequence lengths are variable, so there is no predefined notion of a coordinate-wise statistics.

- **Layer normalization** [Ba et al. 2016] is similar but works with a single input sequence of arbitrary lengths.

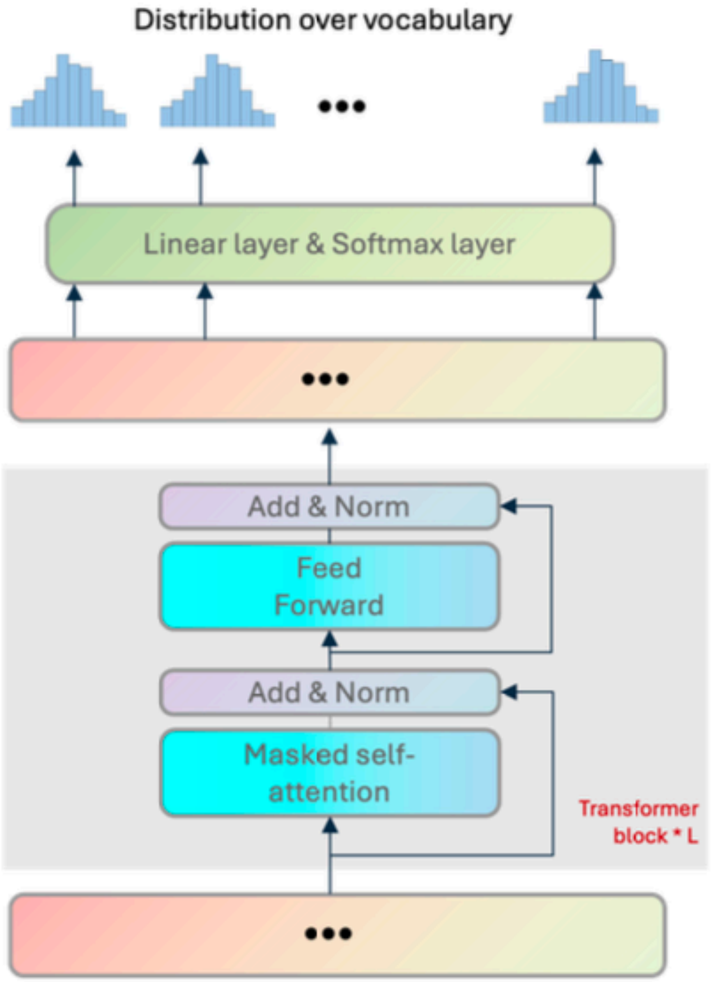
$$\mu_t = \frac{1}{d}\sum_{j=1}^{d} x_{t,j}, \sigma_t^2 = \frac{1}{d}\sum_{j=1}^{d}(x_{t,j} - \mu_t)^2$$

$$\tilde{x}_{t,i} = \frac{x_{t,i} - \mu_t}{\sqrt{\sigma_t^2 + \epsilon}}, \text{ for position } t, \text{ coordinate } i$$

- This is widely used in LLMs' transformer layers to stabilize the training dynamics.

- Forward pass in pre-training on single sentence:

  - Initial loss $L = 0$, and let $y_0 = x$
  - For each layer $\ell = 1, \ldots, \mathscr{L}$
    - $Q_\ell \leftarrow W_{Q,\ell} y_{\ell-1}$, $K_\ell \leftarrow W_{K,\ell} y_{\ell-1}$, $V_\ell \leftarrow W_{V,\ell} y_{\ell-1}$
    - $S_\ell \leftarrow \text{Mask}(Q_\ell^T K_\ell)$
    - $y_\ell \leftarrow \text{Row-wise-Softmax}(S_\ell) V_\ell^T$
    - $y_\ell \leftarrow \text{Layernorm}(y_\ell + y_{\ell-1})$
    - $y_{\text{skip}} \leftarrow y_\ell$
    - $y_\ell \leftarrow W_{F2,\ell}\, \sigma(W_{F1,\ell} y_\ell)$
    - $y_\ell \leftarrow \text{Layernorm}(y_\ell + y_{\text{skip}})$
  - $u = [u_1, \ldots, u_T] \leftarrow \text{Row-wise-Softmax}(W_O y_{\mathscr{L}})$
  - $L += \left(\sum_{t=1}^{T} \sum_{i=1}^{|\mathscr{V}|} -\hat{u}_t^{(i)} \log(u_t^{[i]}) \right)$

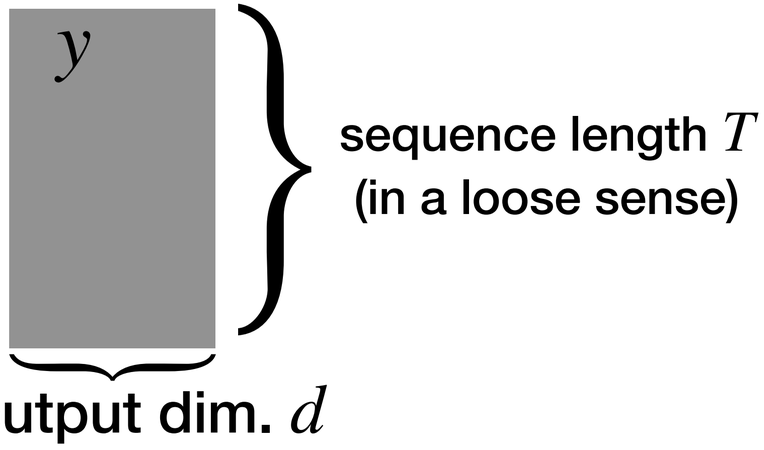


Outline

- Language models
- General LLM framework
  - Token processing
  - Sequence mixing
  - Prediction
- Example architectures

Prediction

- Next token prediction involve outputting a distribution over the vocab and sampling from the distribution.



- The last transformer block outputs $y_{\mathscr{L}} \in \mathbb{R}^{T \times d}$, where $\mathscr{L} \in \mathbb{Z}_+$ is the last layer, T is the context length, and d is the dimension of the hidden representation.

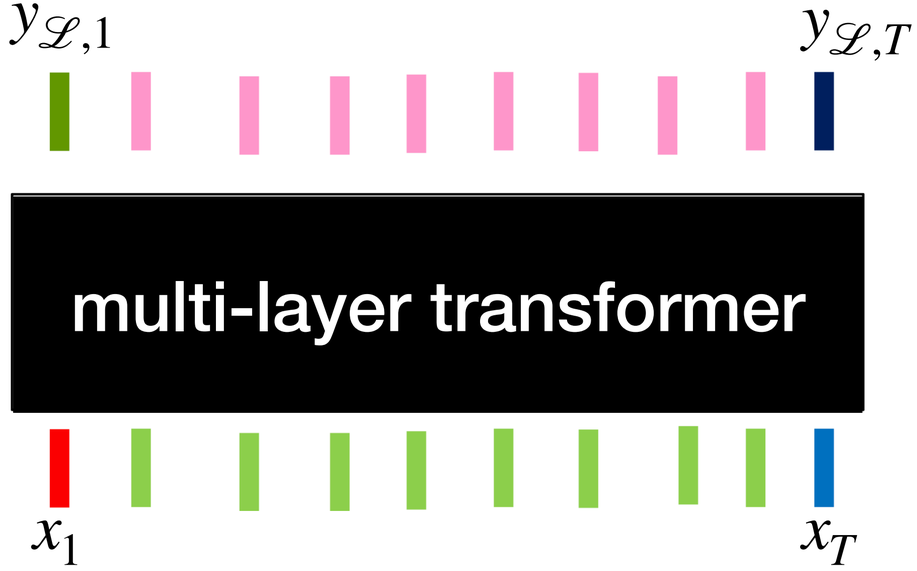
- The **prediction layer** takes the output representation of the last word, $y_{\mathscr{L},T} \in \mathbb{R}^d$, to predict the next token, with a learnable parameter $W_O \in \mathbb{R}^{|\mathscr{V}| \times d}$, which may or may not be sharing weights with the input token embedding matrix.

$$u_T \leftarrow \text{Softmax}(W_O y_{\mathscr{L},T})$$

$$u_{T,i} = \mathbb{P}(i\text{-th token}) = \frac{e^{W_{O,i} y_{\mathscr{L},t}}}{\sum_{j=1}^{|\mathscr{V}|} e^{W_{O,j} y_{\mathscr{L},t}}}$$

$$u_T = \text{Softmax}\left(W_O \; y_{\mathscr{L},T} \right)$$

$|\mathscr{V}|$ $\quad$ $|\mathscr{V}|$ $\quad$ d $\quad$ d

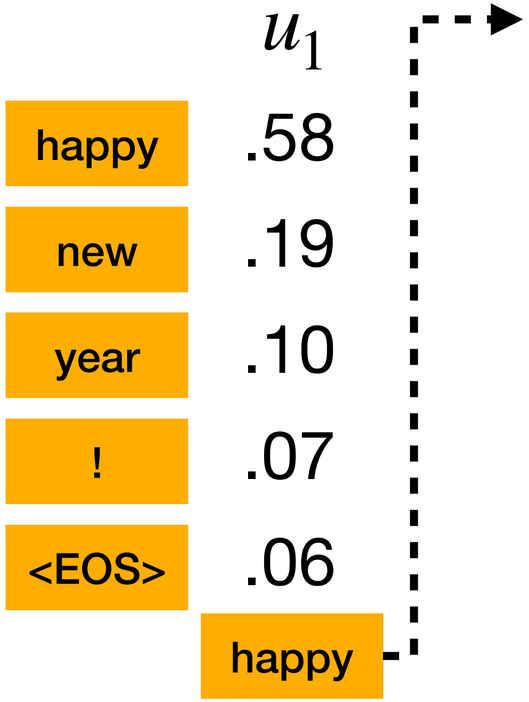


- Given the (conditional) token distribution $\{u_t\}_{t=1}^{T}$, there are many ways to sample tokens, auto-regressively.

- Auto-regressive sampling uses the chain rule to break the distribution on the sentence into:

$$\mathbb{P}(S) = \mathbb{P}(w_{1:T}) = \mathbb{P}(w_1)\mathbb{P}(w_2 \mid w_1)\mathbb{P}(w_3 \mid w_1, w_2)\cdots\mathbb{P}(w_T \mid w_{1:T-1})$$

- **Random sampling** from this conditional distribution generates token-by-token from the distribution u_t, each time until <EOS>.

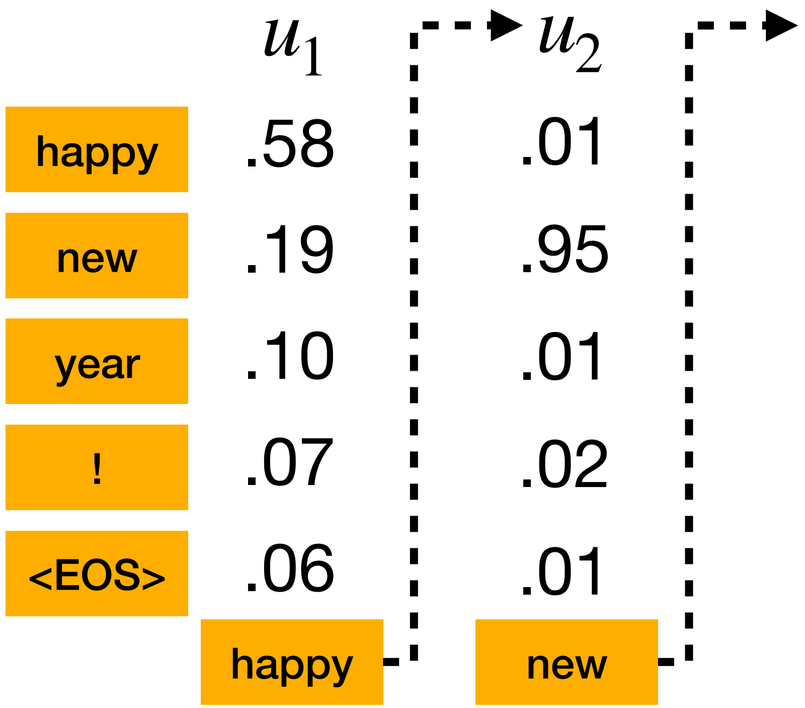


- Given the (conditional) token distribution $\{u_t\}_{t=1}^{T}$, there are many ways to sample tokens, auto-regressively.

- Auto-regressive sampling uses the chain rule to break the distribution on the sentence into:

$$\mathbb{P}(S) = \mathbb{P}(w_{1:T}) = \mathbb{P}(w_1)\mathbb{P}(w_2 \mid w_1)\mathbb{P}(w_3 \mid w_1, w_2)\cdots\mathbb{P}(w_T \mid w_{1:T-1})$$

- **Random sampling** from this conditional distribution generates token-by-token from the distribution u_t, each time until <EOS>.

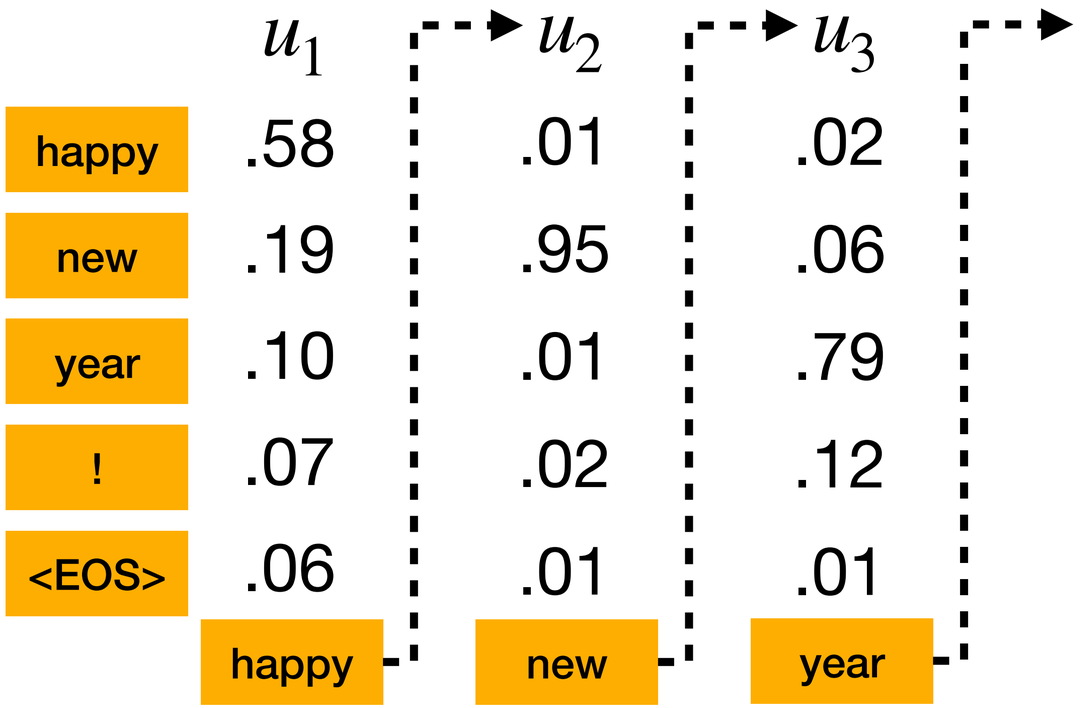


- Given the (conditional) token distribution $\{u_t\}_{t=1}^{T}$, there are many ways to sample tokens, auto-regressively.

- Auto-regressive sampling uses the chain rule to break the distribution on the sentence into:

$$\mathbb{P}(S) = \mathbb{P}(w_{1:T}) = \mathbb{P}(w_1)\mathbb{P}(w_2 \mid w_1)\mathbb{P}(w_3 \mid w_1, w_2)\cdots\mathbb{P}(w_T \mid w_{1:T-1})$$

- **Random sampling** from this conditional distribution generates token-by-token from the distribution u_t, each time until <EOS>.

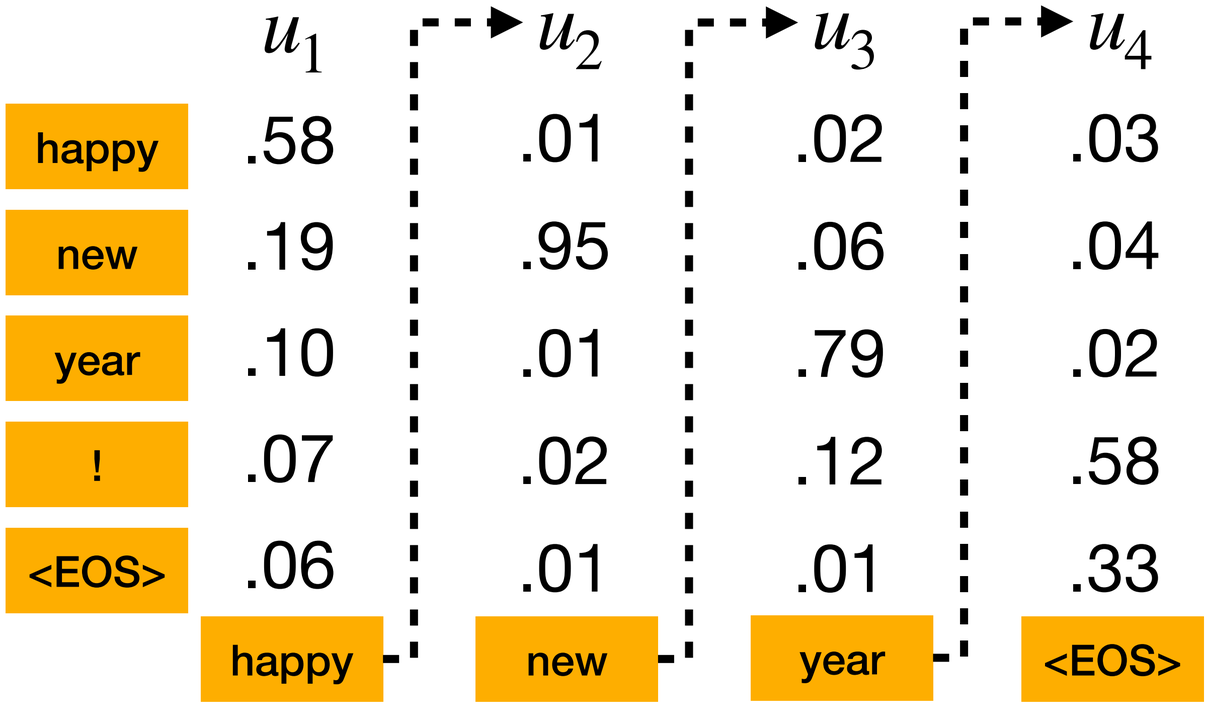


- Given the (conditional) token distribution $\{u_t\}_{t=1}^{T}$, there are many ways to sample tokens, auto-regressively.

- Auto-regressive sampling uses the chain rule to break the distribution on the sentence into:

$$\mathbb{P}(S) = \mathbb{P}(w_{1:T}) = \mathbb{P}(w_1)\mathbb{P}(w_2 \mid w_1)\mathbb{P}(w_3 \mid w_1, w_2)\cdots\mathbb{P}(w_T \mid w_{1:T-1})$$

- **Random sampling** from this conditional distribution generates token-by-token from the distribution u_t, each time until <EOS>.



- What could go wrong?

- Given the (conditional) token distribution $\{u_t\}_{t=1}^{T}$, there are many ways to sample tokens, auto-regressively.

- Auto-regressive sampling uses the chain rule to break the distribution on the sentence into:

$$\mathbb{P}(S) = \mathbb{P}(w_{1:T}) = \mathbb{P}(w_1)\mathbb{P}(w_2 \mid w_1)\mathbb{P}(w_3 \mid w_1, w_2)\cdots\mathbb{P}(w_T \mid w_{1:T-1})$$

- **Random sampling** from this conditional distribution generates token-by-token from the distribution u_t, each time until <EOS>.

    - **Benefit**: if the LM predicts an accurate and **calibrated *conditional distribution***, then random sampling auto-regressively provides an exact sample from the ***joint distribution*** on the sentence.

    - **Weaknesses**:
        - can generate out-of-distribution samples, resulting in **hallucination** and non-grammatical sentences.
        - LMs distributions are **not well calibrated** since they are trained as classifiers with cross-entropy.

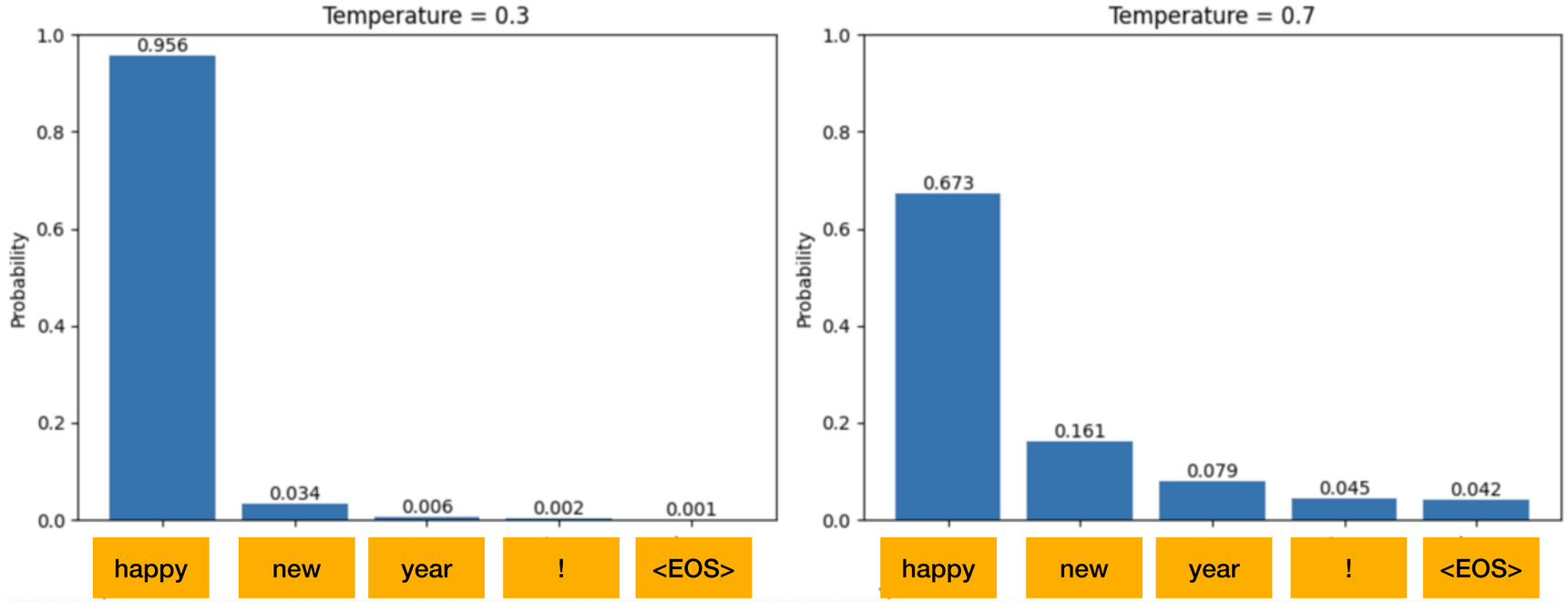
- On the other extreme is **Greedy sampling,** which samples the highest probability token, deterministically.

- **Random sampling** generates token-by-token from the distribution u_t.
- **Greedy sampling** generates the highest probability token, deterministically.
- In practice, one balances the two ends by **Sampling with temperature** T, usually between 0 and 1, which samples from an adjusted Softmax($W_O y_{\mathscr{L},t}$):

$$u_{t,i} = \mathbb{P}(i\text{-th token}) = \frac{e^{\frac{W_{O,i} y_{\mathscr{L},t}}{T}}}{\sum_{i-1}^{|\mathscr{V}|} e^{\frac{W_{O,j} y_{\mathscr{L},t}}{T}}},$$

where

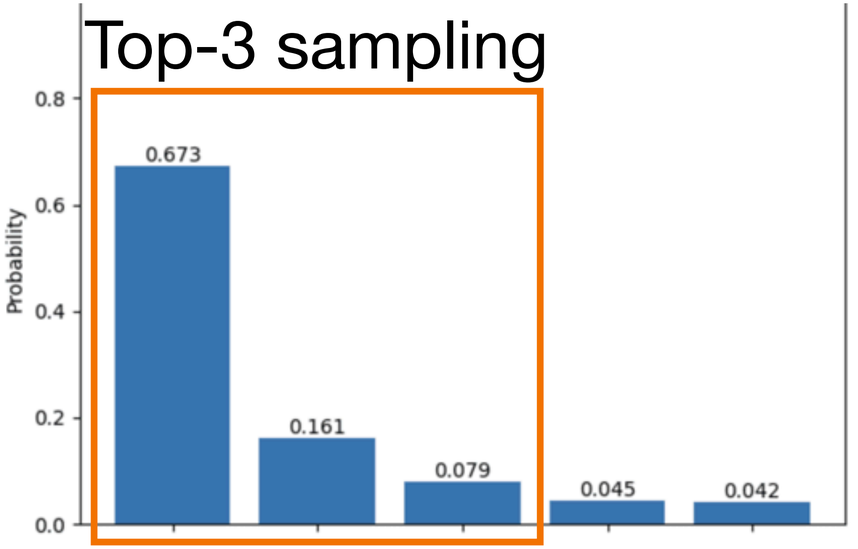
- $T = 1$ recovers the original **random sampling**,
- T=0 recovers the **greedy sampling,**
- 0<T<1 balances the two.
- T is tuned like a hyper-parameter at inference time



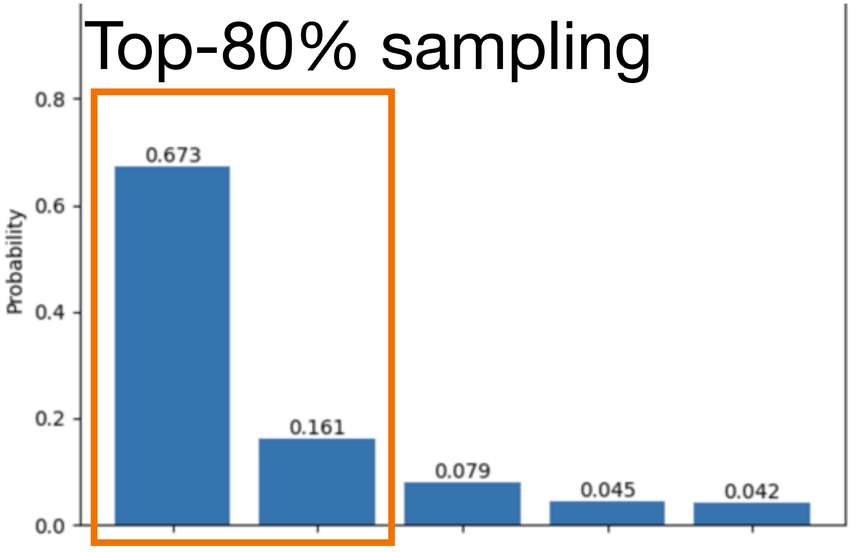
Next-word probabilities under different temperatures

- **Random sampling** generates token-by-token from the distribution u_t.
- **Greedy sampling** generates the highest probability token, deterministically.
- In practice, one balances the two ends by **Sampling with temperature** T

- **Top-k sampling** samples from u_t
  with temperature, but only among the top-k tokens.



- **Nucleus sampling** (also known as **top-p sampling**)
  [Holtzman et al. 2020] samples from u_t
  but only among the smallest set whose
  cumulative probability exceeds $p \in (0,1)$



- **Greedy sampling** is attempting to solve the following **joint** maximization:

$$\max_{w_{1:T}} \mathbb{P}(w_{1:T}) = \max_{w_{1:T}} \mathbb{P}(w_1)\mathbb{P}(w_2 \mid w_1)\cdots\mathbb{P}(w_T \mid w_{1:T-1})$$

  which is computationally hard especially for long T generation

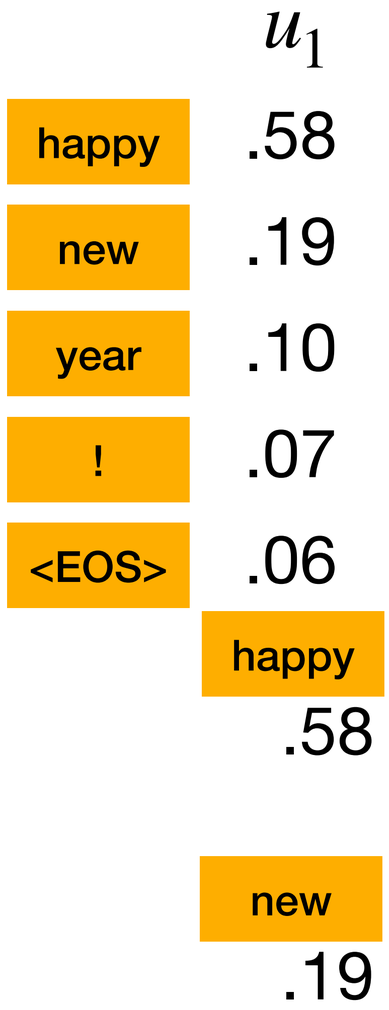
- Instead, **Greedy sampling** solves the following approximation:

$$\begin{aligned}
&\approx \max_{w_{1:T}} \mathbb{P}(w_1)\mathbb{P}(w_2 \mid w_1^*)\cdots\mathbb{P}(w_T \mid w_{1:T-1}^*), \\
&\approx \max_{w_1} \mathbb{P}(w_1) \cdot \max_{w_2} \mathbb{P}(w_2 \mid w_1^*) \cdot \cdots \cdot \max_{w_T} \mathbb{P}(w_T \mid w_{1:T-1}^*)
\end{aligned}$$

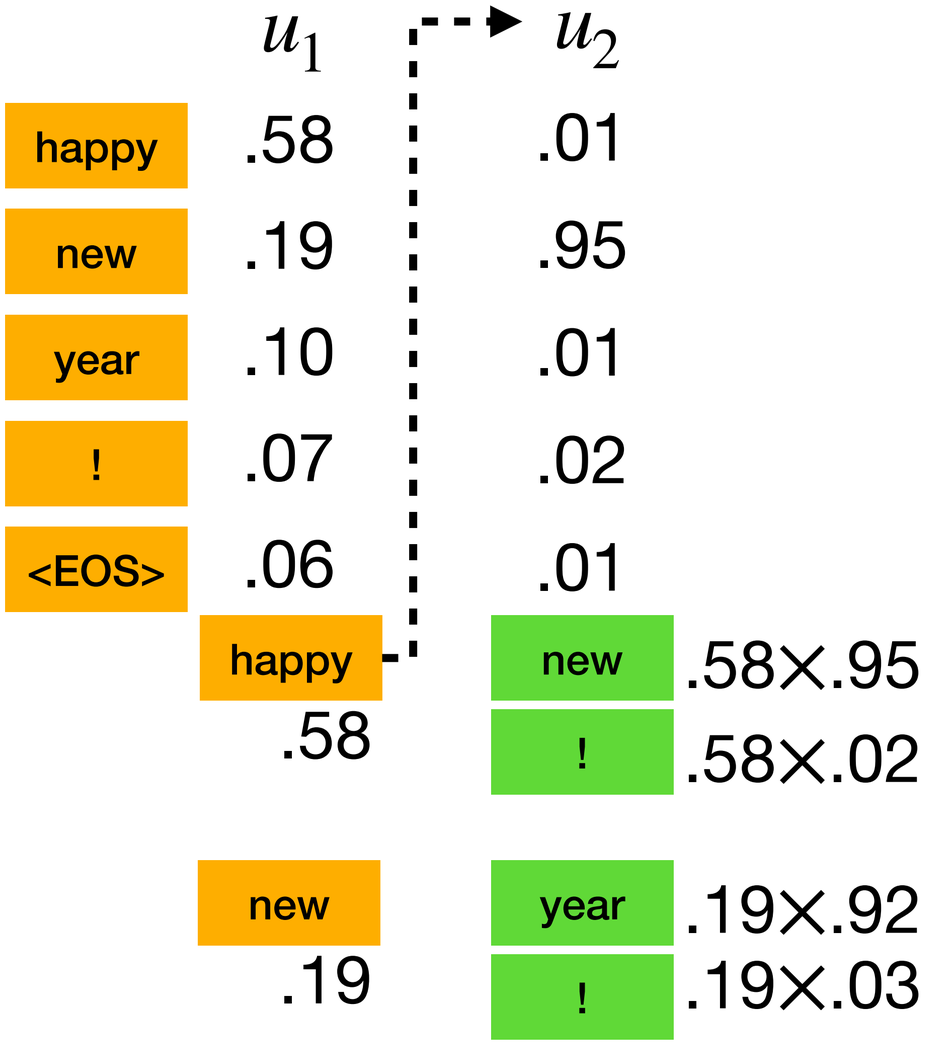
  where w_t^*'s are the solution of the previous maximization.

- We can try to better approximate the most likely sequence directly (solving the LHS of above), using **Beam search,** which produces high quality but low-diversity sequences.
- (deterministic) Beam search with beam width B proceeds as follows.
  - B most likely candidates are selected for the first token.
  - For each token at position 1, k most likely next tokens are selected.
  - Out of kB candidates for a sequence of length 2, the most likely B candidates are chosen.
  - This repeats until B candidates end on <EOS>.

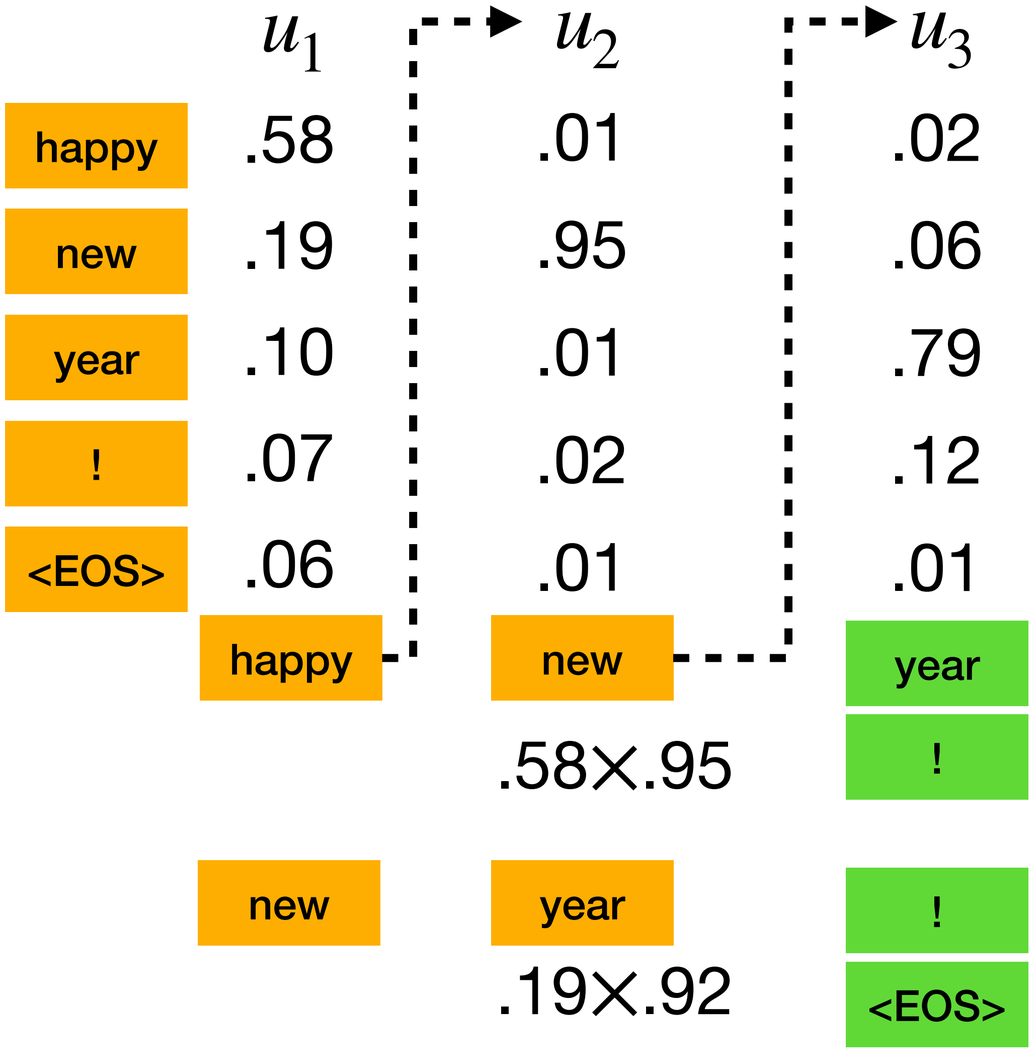
- **Beam search** produces high quality but low-diversity sequences.
- $B = 2, k = 2$



- **Beam search** produces high quality but low-diversity sequences.
- $B = 2, k = 2$



- **Beam search** produces high quality but low-diversity sequences.
- $B = 2, k = 2$



- What could go wrong?

In practice, **Nucleus sampling** strikes the right balance and usually performs the best.

Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Beam Search, *b*=32:
"The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the Universidad Nacional Autónoma de México (UNAM) and the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de ..."

Pure Sampling:
They were cattle called Bolivian Cavalleros; they live in a remote desert uninterrupted by town, and they speak huge, beautiful, paradisiacal Bolivian linguistic thing. They say, 'Lunch, marge.' They don't tell what the lunch is," director Professor Chuperas Omwell told Sky News. "They've only been talking to scientists, like we're being interviewed by TV reporters. We don't even stick around to be interviewed by TV reporters. Maybe that's how they figured out that they're cosplaying as the Bolivian Cavalleros."

Figure 1: Even with substantial human context and the powerful GPT-2 Large language model, Beam Search (size 32) leads to degenerate repetition (highlighted in blue) while pure sampling leads to incoherent gibberish (highlighted in red). When $b \geq 64$, both GPT-2 Large and XL (774M and 1542M parameters, respectively) prefer to stop generating immediately after the given context.

Sources

- Other courses in LLMs that the lecture slides are based on
  - CSE493S/599S at UW by Ludwig Schmidt: https://mlfoundations.github.io/advancedml-sp23/
  - EE-628 at EPFL by Volkan Cevher: https://www.epfl.ch/labs/lions/teaching/ee-628-training-large-language-models/ee-628-slides-2025/
- Useful blog posts
  - https://azizbelaweid.substack.com/p/complete-summary-of-absolute-relative
  - https://blog.dust.tt/speculative-sampling-llms-writing-a-lot-faster-using-other-llms/
  - https://gordicaleksa.medium.com/eli5-flash-attention-5c44017022ad
- Dan Jurafsky and James H. Martin. Speech and Language Processing (3rd ed. draft). draft, third edition, 2023.
- Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. “Efficient estimation of word representations in vector space”, In International Conference on Learning Representations, 2013.
- Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation”, Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.
- Ofir Press, Noah A. Smith1,3 Mike Lewis2 , “Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation”, In International Conference on Learning Representations, 2022
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, “Attention Is All You Need”, In Neural Information Processing Systems, 2017
- Beitong Zhou, Cheng Cheng, Guijun Ma, and Yong Zhang. “Remaining useful life prediction of lithium-ion battery based on attention mechanism with positional encoding”, In IOP Conference Series: Materials Science and Engineering, 2020.
- Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks." In International Conference on Machine Learning, 2013

- Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory.” In Neural Computation, 9(8):1735–1780, 11 1997.
- Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. “Learning phrase representations using rnn encoder-decoder for statistical machine translation”, In ACL 2014
- Andrey Andreyevich Markov. “Essai d’une recherche statistique sur le texte du roman. ‘Eugene Onegin’ illustrant la liaison des epreuve en chain”. In: Izvistia Imperatorskoi Akademii Nauk (Bulletin de l’Académie Impériale des Sciences de St.-Pétersbourg). 6th ser, 7:153–162, 1913.
- Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, Yejin Choi, “The Curious Case of Neural Text Degeneration”, In *International Conference on Learning Representations*, 2020
- Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre and John Jumper, “Accelerating Large Language Model Decoding with Speculative Sampling”In, *ACL-findings*, 2024
- Sergey Ioffe, Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", In *International Conference on Machine Learning*, 2015
- Shibani Santurkar∗ MIT shibani@mit.edu Dimitris Tsipras∗ MIT tsipras@mit.edu Andrew Ilyas∗ MIT ailyas@mit.edu Aleksander Madry, “How Does Batch Normalization Help Optimization?”, In *Advances in Neural Information Processing Systems* 31 (NeurIPS 2018)
- Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton, “Layer Normalization “, In 2016