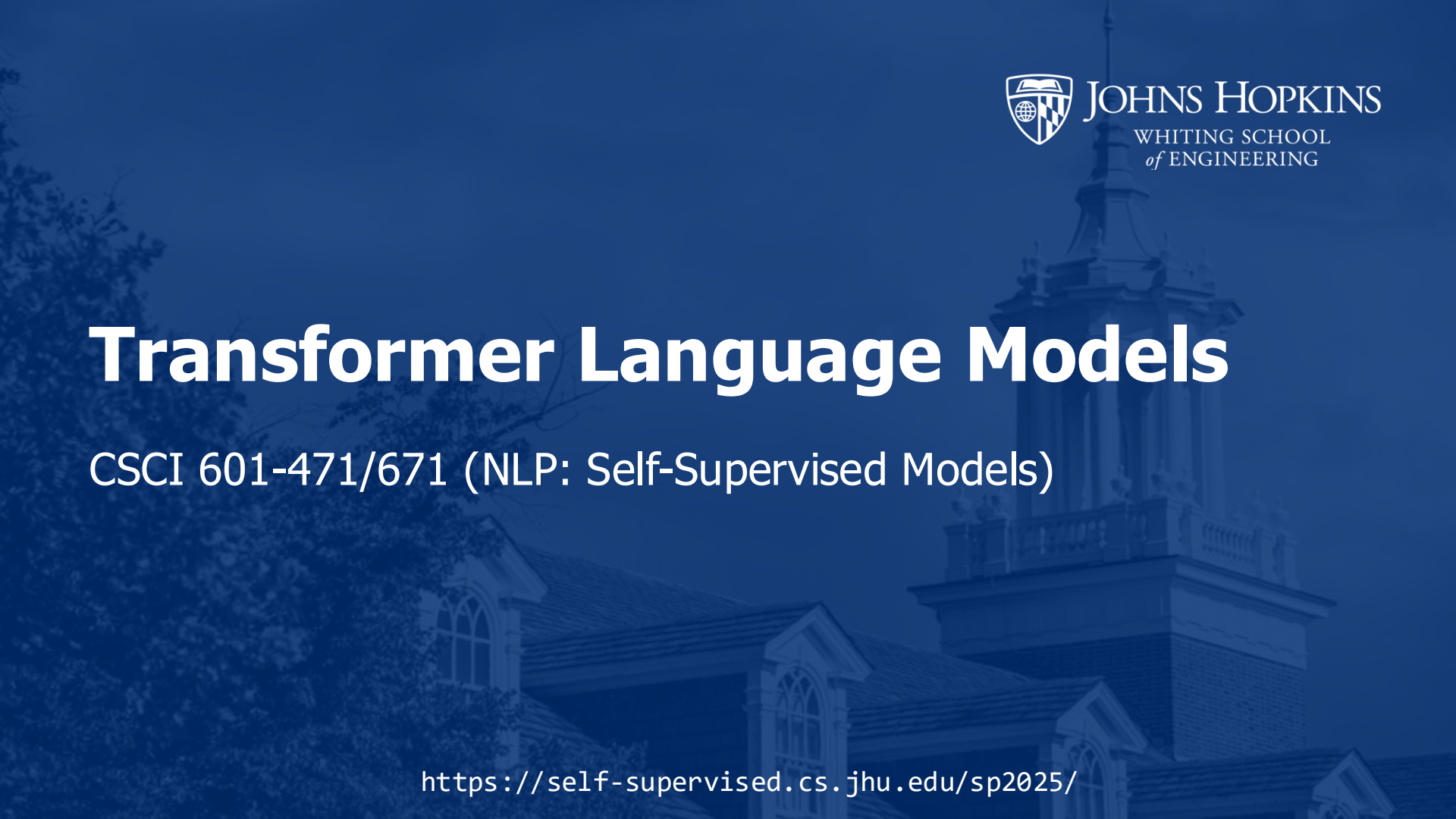


Transformer Language Models

CSCI 601-471/671 (NLP: Self-Supervised Models)

https://self-supervised.cs.jhu.edu/sp2025/

Language Models: A History

- Probabilistic n-gram models of text generation [Jelinek+ 1980's, …]
  - Applications: Speech Recognition, Machine Translation
- Word representation learning [Brown 1992, …]
  - Brown, LSA, Word2Vec, Glove …
- Statistical or shallow neural LMs (late 90's – mid 00's) [Bengio+ 2001, …]
- Pre-training deep neural language models (2017's onward):
  - Many models based on: Self-Attention

RNNs, Back to the Cons

- While RNNs in theory can represent long sequences, they quickly forget portions of the input.
- Vanishing/exploding gradients
- Difficult to parallelize
- The alternative solution we will see: Transformers!

Chapter Plan

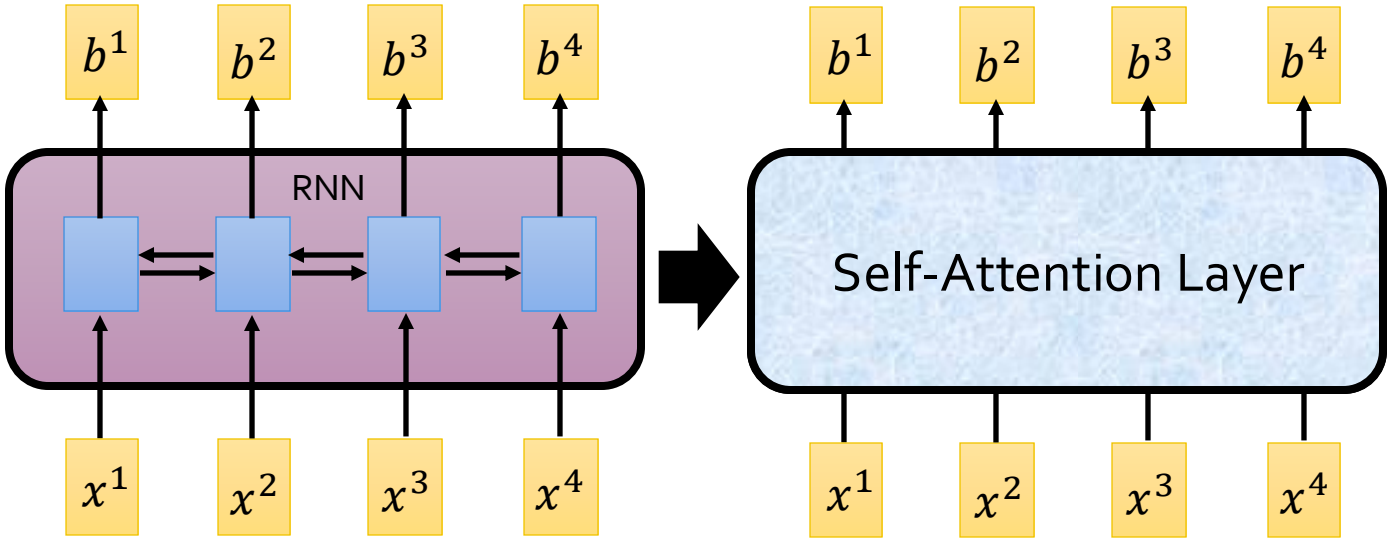
1. Self-Attention: how it works
2. Transformer architecture
3. Transformer-based families of Language Models
4. Practical hacks and variants
5. Various objective functions

Chapter goal----

Self-Attention

Self-Attention

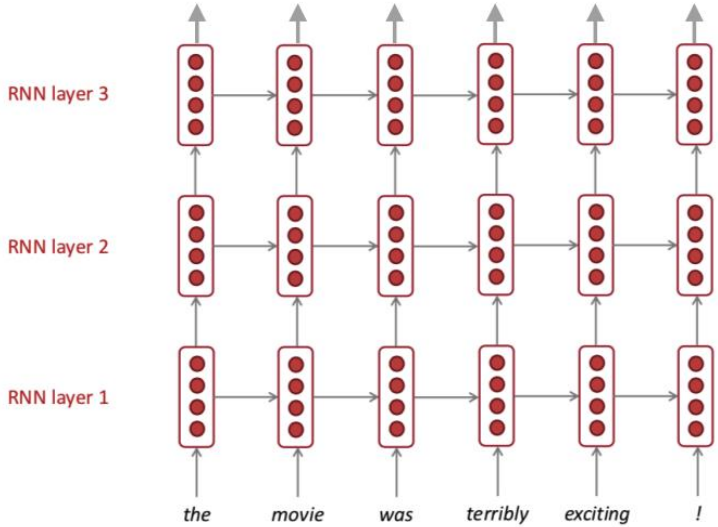
- b^i is obtained based on the whole input sequence.
- can be parallelly computed.

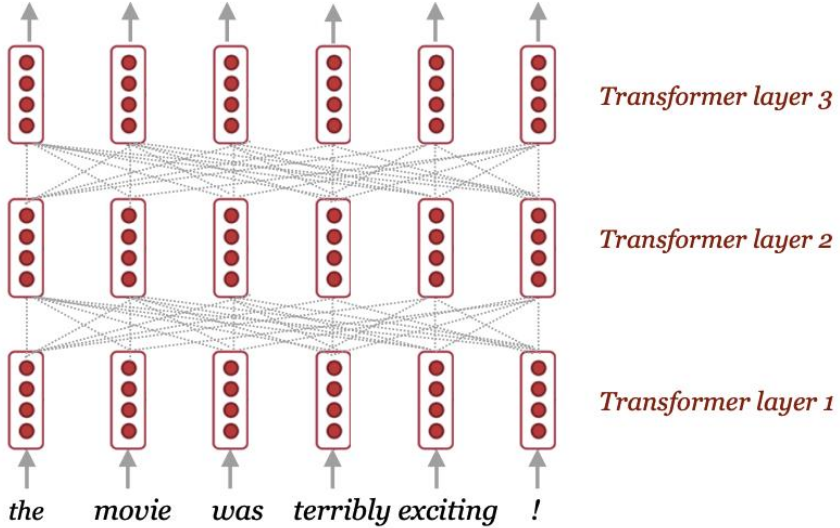


Idea: replace any thing done by RNN with self-attention.

"Neural machine translation by jointly learning to align and translate" Bahdanau etl. 2014;
"Attention is All You Need" Vaswani et al. 2017

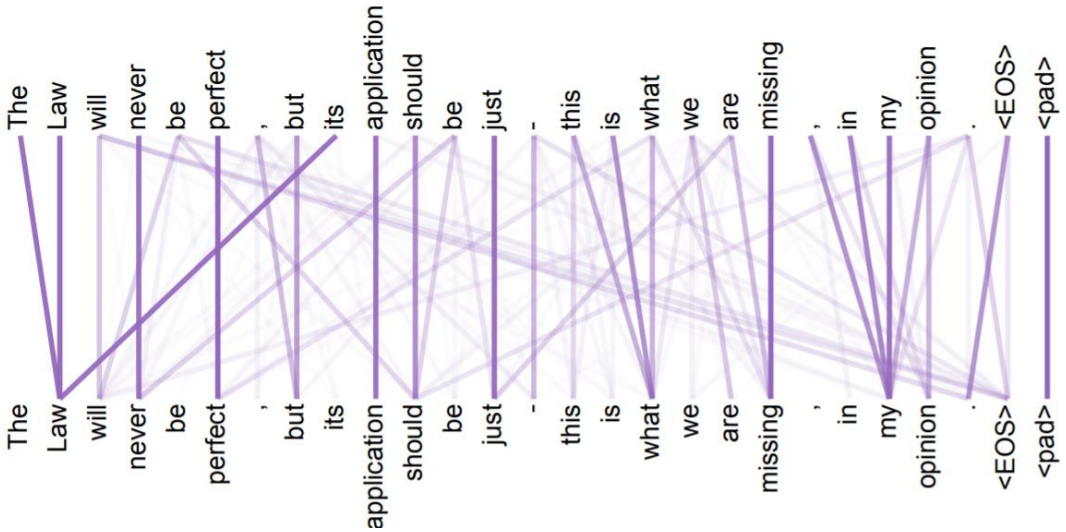
RNN vs Transformer





Attention

- Core idea: build a mechanism to focus ("attend") on a particular part of the context.



[Attention Is All You Need, Vaswani et al. 2017]

Defining Self-Attention

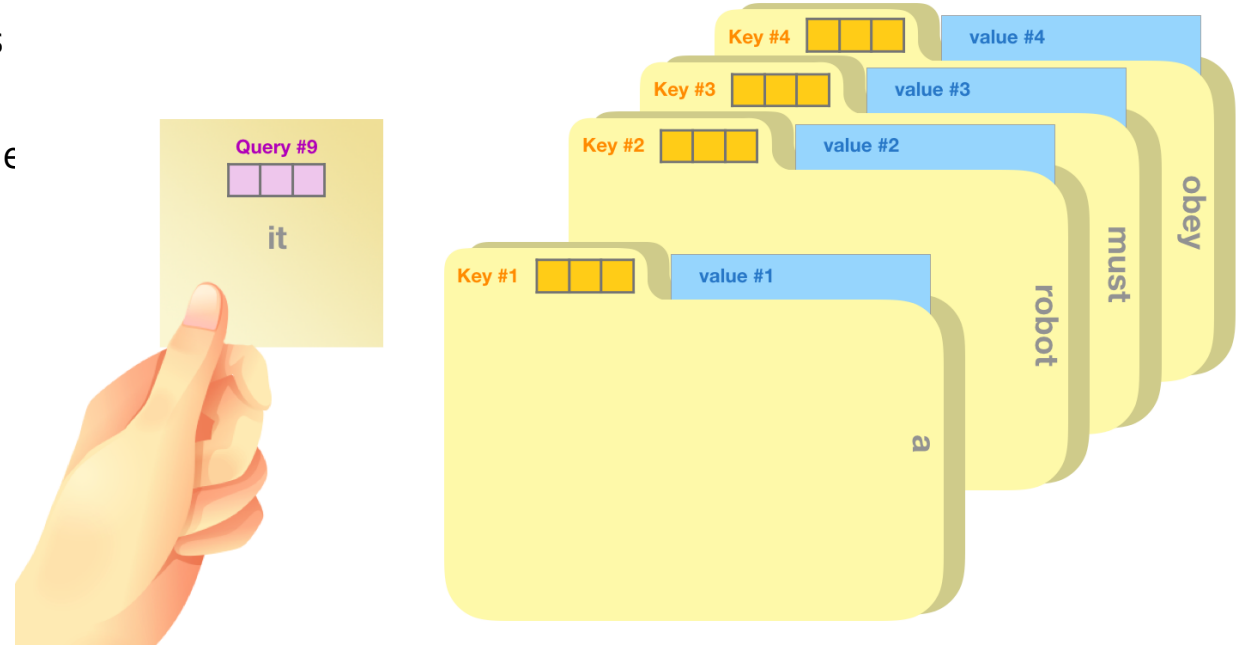
- Terminology:
  - Query: to match others
  - Key: to be matched
  - Value: information to be extracted

[Vaswani et al. 2017: https://arxiv.org/abs/1706.03762]

Defining Self-Attention

An analogy ....

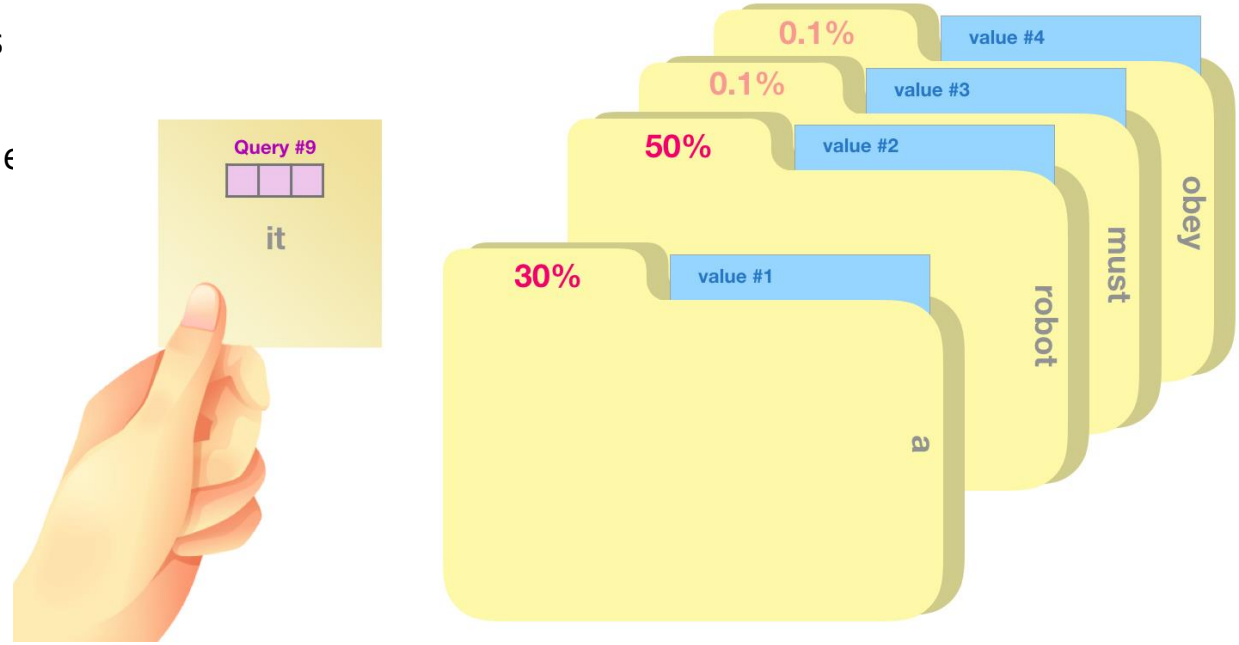
- Terminology:
  - Query: to match others
  - Key: to be matched
  - Value: information to be



[Vaswani et al. 2017: https://arxiv.org/abs/1706.03762]

Defining Self-Attention

- Terminology:
  - Query: to match others
  - Key: to be matched
  - Value: information to be



[Vaswani et al. 2017: https://arxiv.org/abs/1706.03762]

q: query (to match others)

$$q_i = W^q x_i$$

k: key (to be matched)

$$k_i = W^k x_i$$

v: value (information to be extracted)

$$v_i = W^v x_i$$

q: query (to match others)

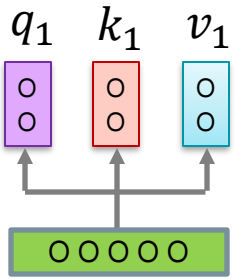
$$q_i = W^q x_i$$

k: key (to be matched)

$$k_i = W^k x_i$$

v: value (information to be extracted)

$$v_i = W^v x_i$$



The

q: query (to match others)

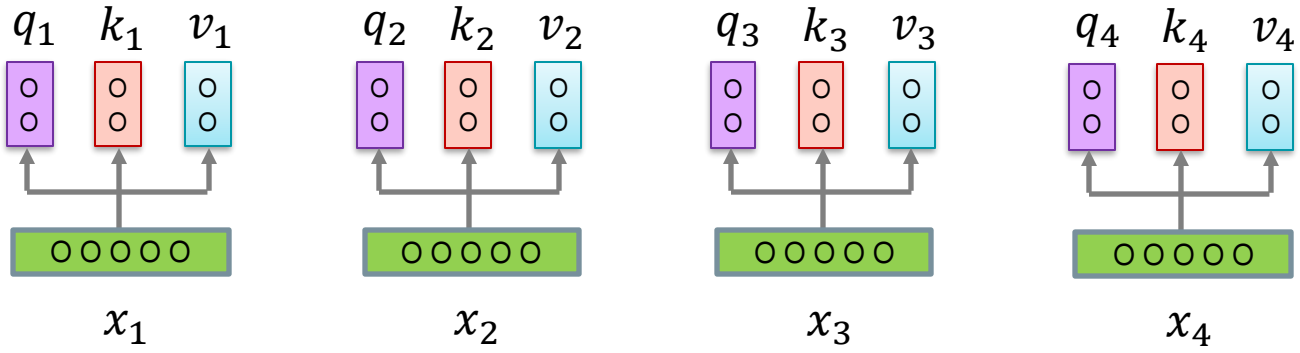
$$q_i = W^q x_i$$

k: key (to be matched)

$$k_i = W^k x_i$$

v: value (information to be extracted)

$$v_i = W^v x_i$$



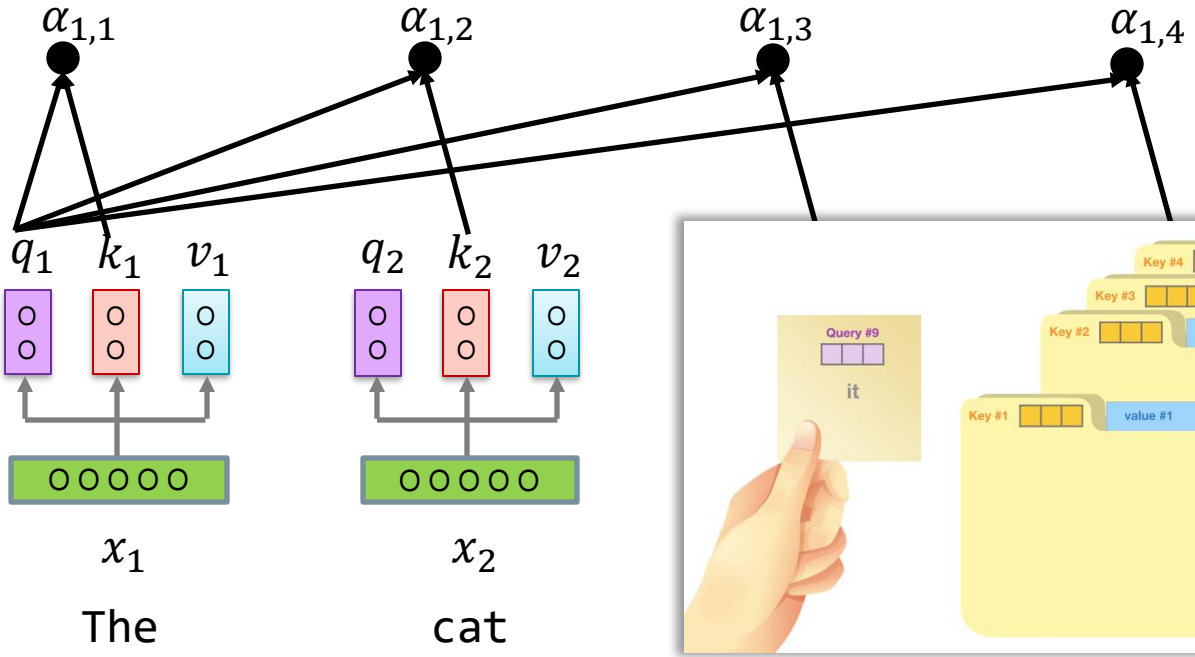
$$\alpha_{1,i} = \underbrace{q^1 \cdot k^i / \sqrt{d}}_{\text{Scaled dot product}}$$

q: query (to match others)

k: key (to be matched)

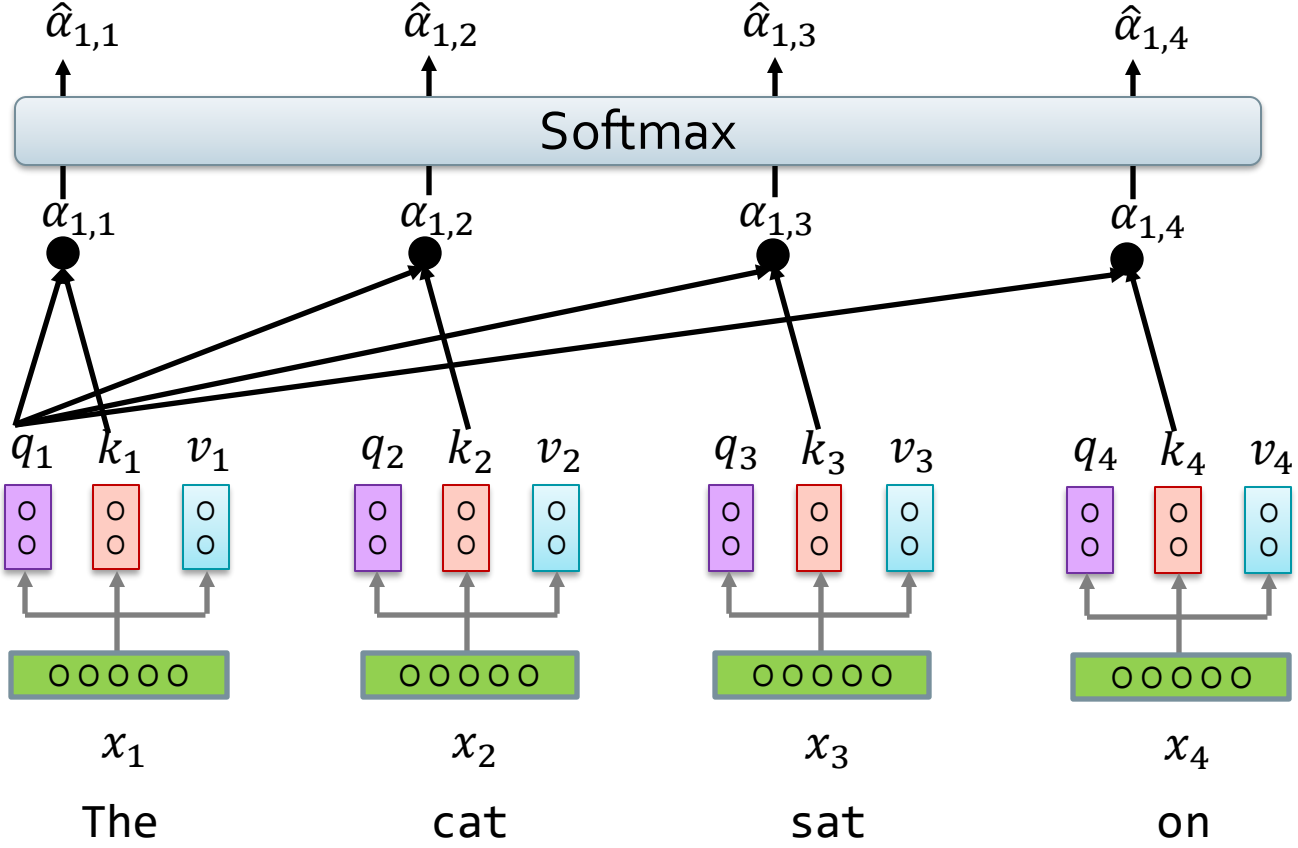
v: value (information to be extracted)

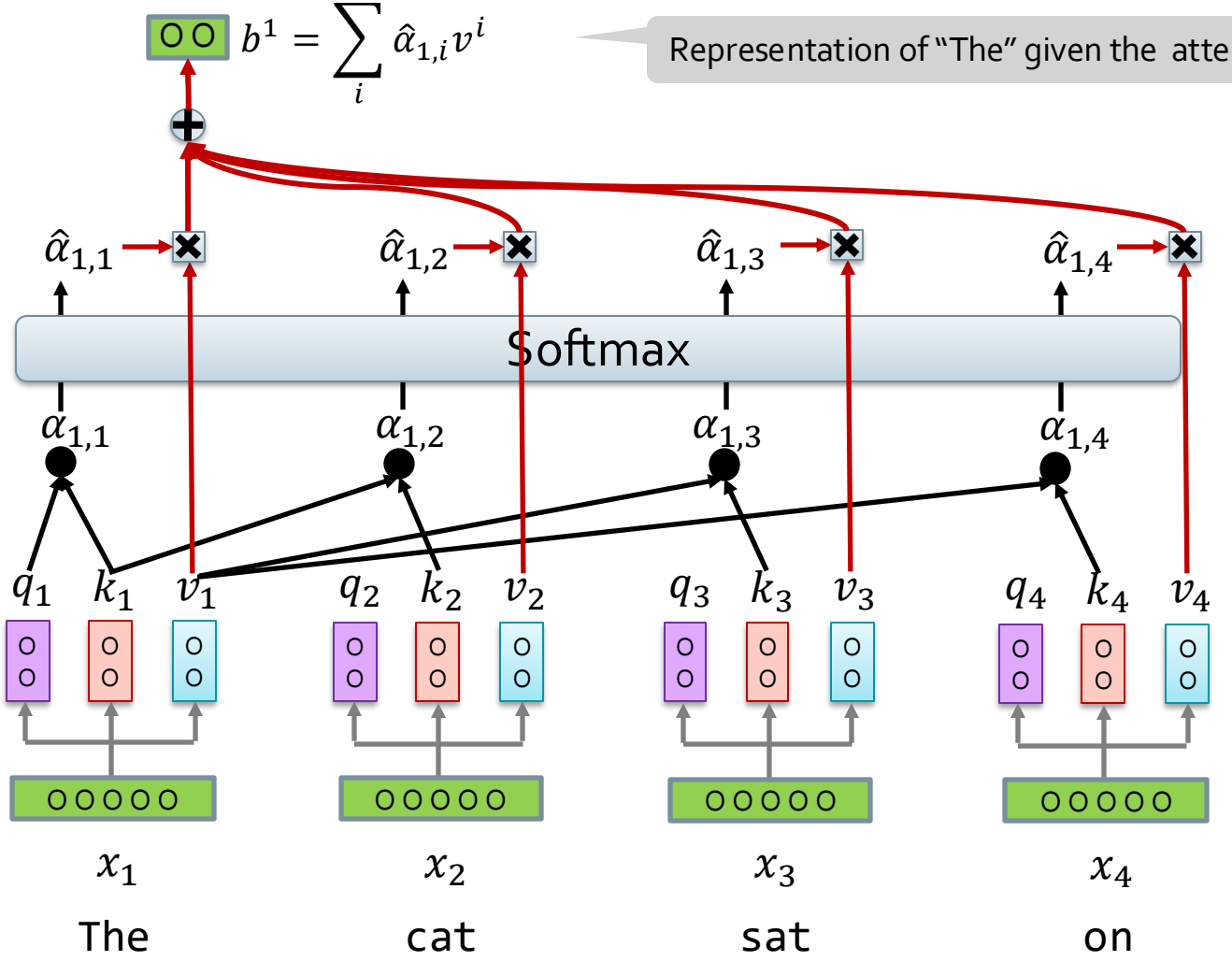
How much should "The" attend to other positions?



$$\sigma(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

How much should "The" attend to other positions?




$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$
Representation of "The" given the attention weights
$\hat{\alpha}_{1,1}$
$\hat{\alpha}_{1,2}$
$\hat{\alpha}_{1,3}$
$\hat{\alpha}_{1,4}$
Softmax
$\alpha_{1,1}$
$\alpha_{1,2}$
$\alpha_{1,3}$
$\alpha_{1,4}$
q_1 k_1 v_1
q_2 k_2 v_2
q_3 k_3 v_3
q_4 k_4 v_4
x_1 x_2 x_3 x_4
The cat sat on

Question

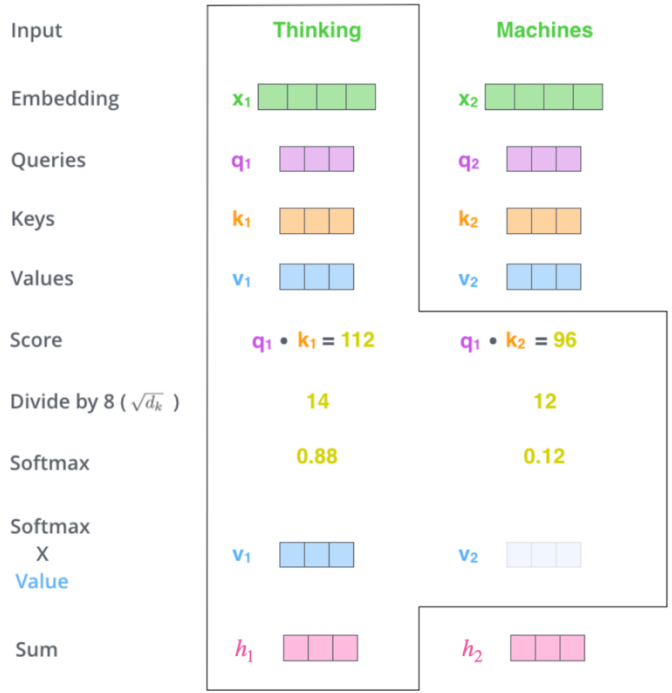
- What would be the output vector for the word "Thinking"?

(a) $0.5\mathbf{v}_1 + 0.5\mathbf{v}_2$

(b) $0.54\mathbf{v}_1 + 0.46\mathbf{v}_2$

(c) $0.88\mathbf{v}_1 + 0.12\mathbf{v}_2$

(d) $0.12\mathbf{v}_1 + 0.88\mathbf{v}_2$



[Slide credit: Danqi Chen]

Self-Attention: Matrix Notation

$X \in \mathbb{R}^{n \times d_1}$ (n = input length)

$Q = XW^Q \quad K = XW^K \quad V = XW^V$

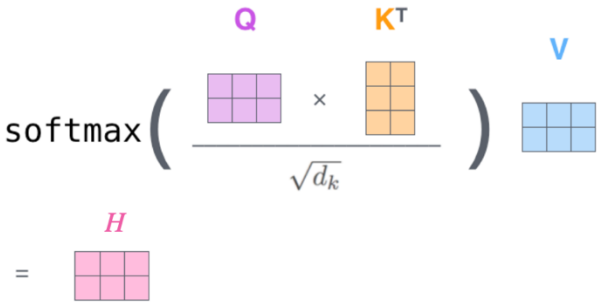
$W^Q \in \mathbb{R}^{d_1 \times d_q}, W^K \in \mathbb{R}^{d_1 \times d_k}, W^V \in \mathbb{R}^{d_1 \times d_v}$

$n \times d_q$ $d_k \times n$

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

$n \times$

Q: What is this softmax operation?



[Slide credit: Danqi Chen]

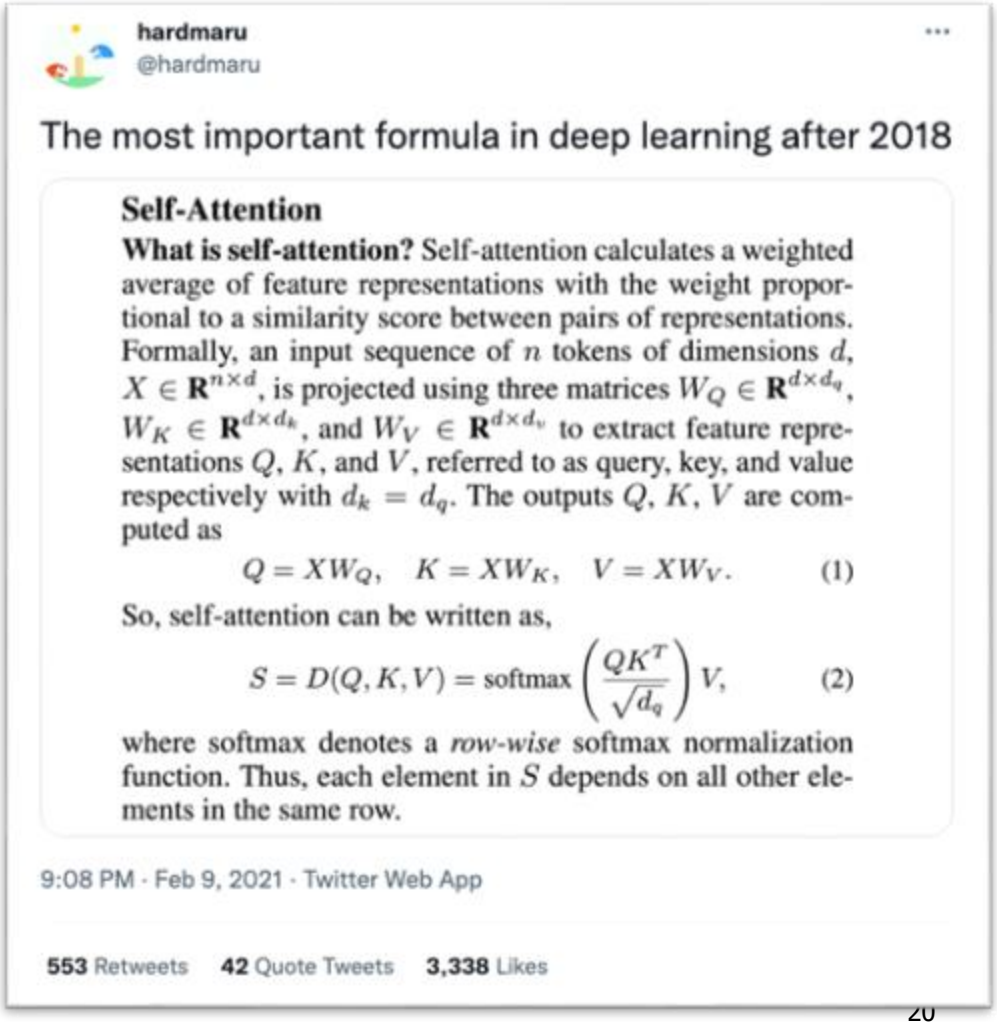
Self-Attention

- Can write it in matrix form:

- Given input $\mathbf{x}$:

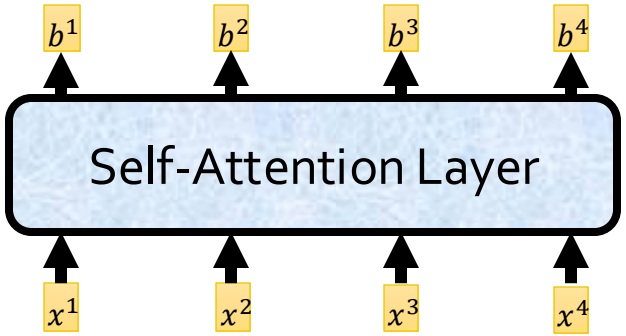
$$Q = \mathbf{W}^q\mathbf{x}$$
$$K = \mathbf{W}^k\mathbf{x}$$
$$V = \mathbf{W}^v\mathbf{x}$$

$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$



Self-Attention: Back to Big Picture

- Attention is a powerful mechanism to create context-aware representations
- A way to focus on select parts of the input



- Better at maintaining long-distance dependencies in the context.

[Attention Is All You Need, Vaswani et al. 2017]

21

Computational and Space Complexity

- The attention function:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- $\dim(QK^T) = N^2 \rightarrow O(N^2 d_k)$ time complexity to calculate QK.
- Attention matrix $\dim\left(\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\right) = N \times N$
  - Storing the attention matrix for each head $\rightarrow O(N^2 h)$.
- If $N \gg d_k, h$, the time and space complexity is $O(N^2)$.
  - Scalability, resource consumption, adoption, etc.

[Slide credit: CS886 at UWaterloo]

Computational and Space Complexity (2)

Layer Type	Complexity per Layer	Sequential Operations
Self-Attention	$O(n^2 \cdot d)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$

- n = sequence length, d = hidden dimension
- Quadratic complexity, but:
  - O(1) sequential operations (not linear like in RNN)

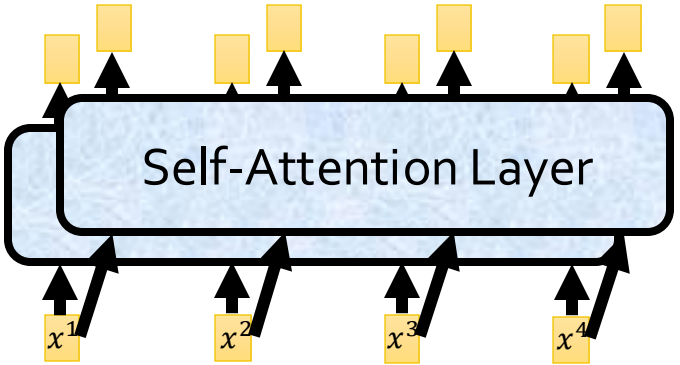
- Can be efficiently parallelized

[Attention Is All You Need, Vaswani et al. 2017]

Multi-Headed Self-Attention

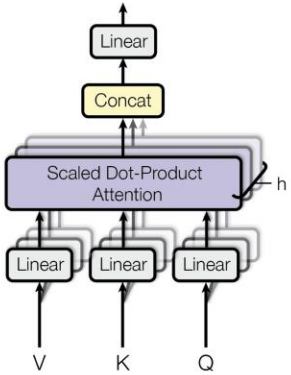
- Multiple parallel attention layers.
  - Each attention layer has its own parameters.
  - Concatenate the results and run them through a linear projection.

- Main idea: Allows model to jointly attend to information from different representation subspaces (like ensembling)



[Attention Is All You Need, Vaswani et al. 2017]

Multi-Headed Self-Attention



- Just concatenate all the heads and apply an output projection matrix.

$$\text{head}_i = \text{Attention}\left(\mathbf{W}_i^q \mathbf{x}, \mathbf{W}_i^k \mathbf{x}, \mathbf{W}_i^v \mathbf{x}\right)$$
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\boldsymbol{W}^O$$

- In practice, we use a reduced dimension for each head.
  - Denote: $d = \text{hidden dimension}$, $\text{m} = \text{number of heads}$

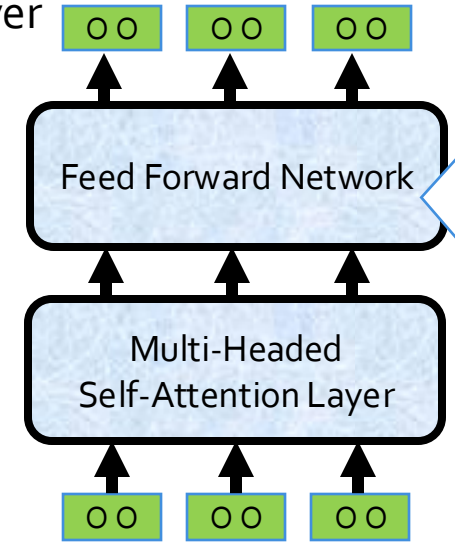
$$\mathbf{W}_i^q \in \mathbb{R}^{d \times \frac{d}{m}}, \quad \mathbf{W}_i^k \in \mathbb{R}^{d \times \frac{d}{m}}, \quad \mathbf{W}_i^v \in \mathbb{R}^{d \times \frac{d}{m}}, \quad \boldsymbol{W}^O \in \mathbb{R}^{d \times d}$$

- The total computational cost is similar to that of single-hear attention with full dimensionality.

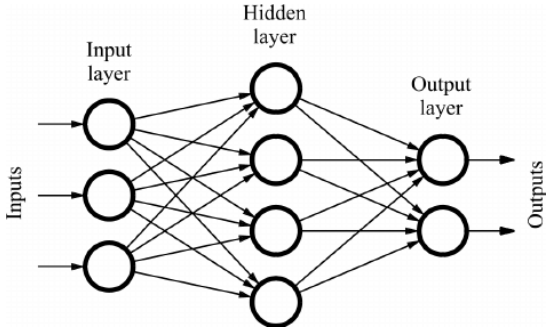
Combine with FFN

- Add a feed-forward network on top it to add more expressivity.
  - This allows the model to apply another transformation to the contextual representations (or "post-process" them).
  - Usually, the dimensionality of the hidden feedforward layer is 2-8 times larger than the input dimension.

$$\text{FFN}(\mathbf{x}) = f(cW_1 + b_1)W_2 + b_2$$



Feedforward Net: Refresher

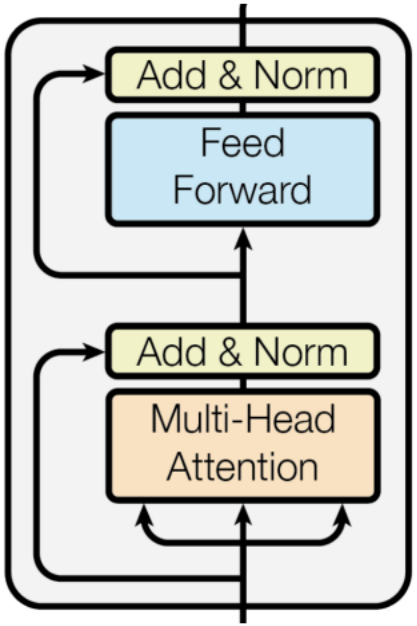


A fully-connected network of nodes and weights.

How Do We Prevent Vanishing Gradients?

- Residual connections let the model "skip" layers
  - These connections are particularly useful for training deep networks

- Use layer normalization to stabilize the network and allow for proper gradient flow



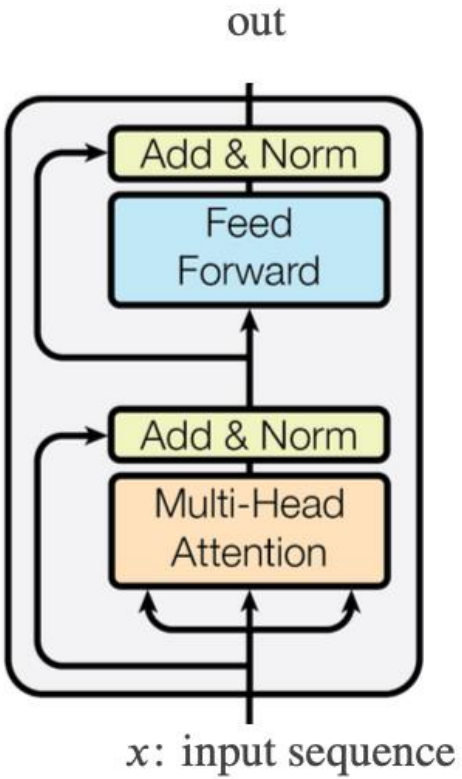
[Attention Is All You Need, Vaswani et al. 2017]

Putting it Together: Self-Attention Block

Given input $\mathbf{x}$:

$$\text{out} = LN(\tilde{\boldsymbol{c}} + \boldsymbol{c}')$$
$$\tilde{\boldsymbol{c}} = \text{FFN}(\boldsymbol{c}') = f(\boldsymbol{c}'W_1 + b_1)W_2 + b_2$$

$$\boldsymbol{c}' = LN(\boldsymbol{c} + \boldsymbol{x})$$
$$\boldsymbol{c} = \text{MultiHeadedAttention}(\boldsymbol{x}; \mathbf{W}^q, \mathbf{W}^k, \mathbf{W}^v)$$



[Attention Is All You Need, Vaswani et al. 2017]

Summary: Self-Attention Block

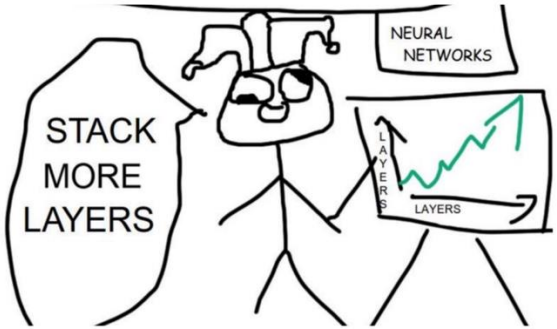
- **Self-Attention:** A critical building block of modern language models.
  - The idea is to compose meanings of words weighted according some similarity notion.

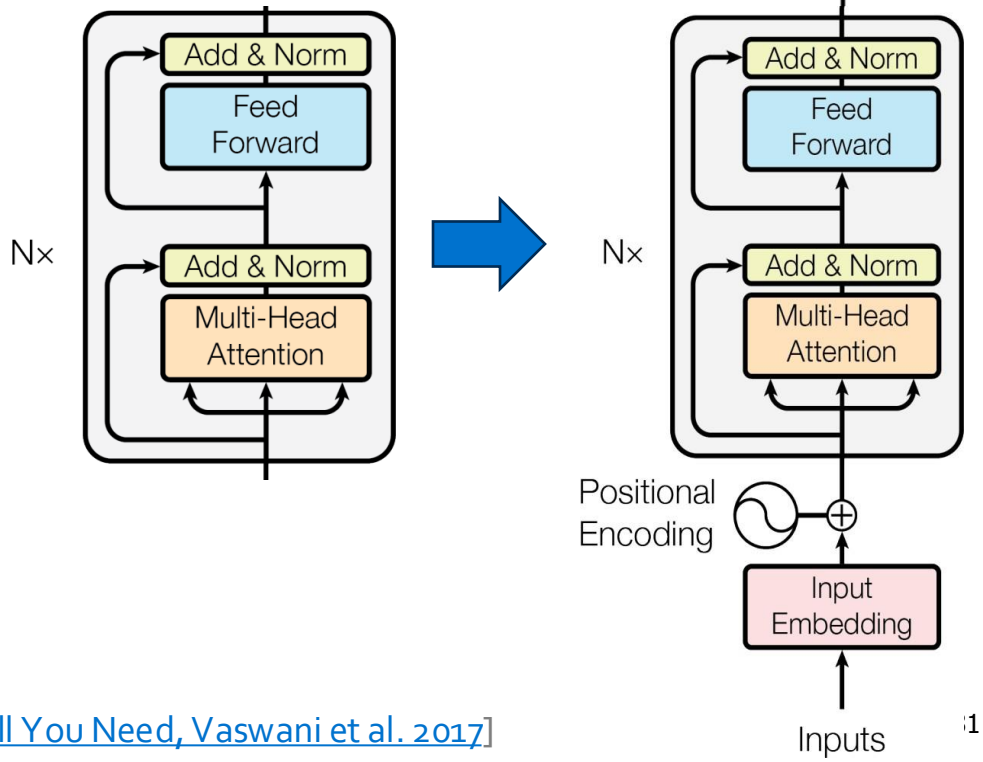
- **Next:** We will combine self-attention blocks to build various architectures known as Transformer.

Transformer

How Do We Make it Deep?

- Stack more layers!





[Attention Is All You Need, Vaswani et al. 2017]

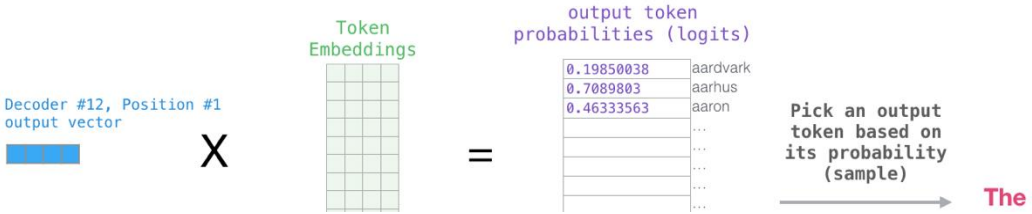
From Representations to Prediction

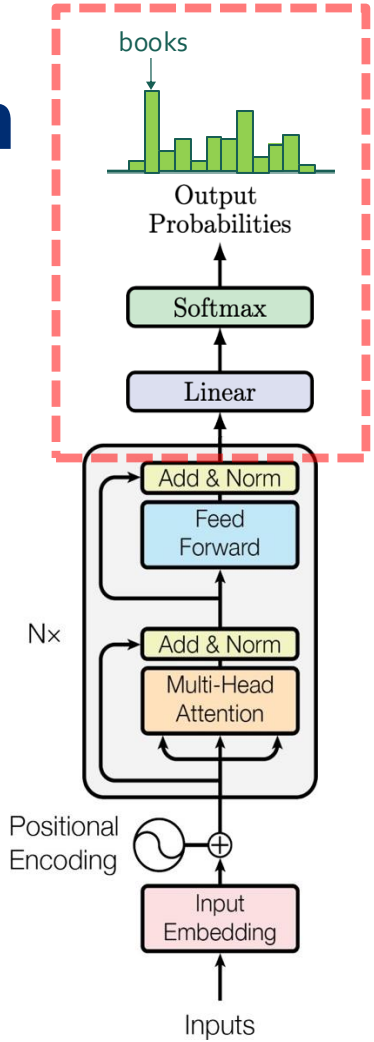
- To perform prediction, add a classification head on top of the final layer of the transformer.
- This can be per token (Language modeling)
- Or can be for the entire sequence (only one token)

$$\text{out} \in \mathbb{R}^{S \times d} \quad \text{(S: Sequence length)}$$

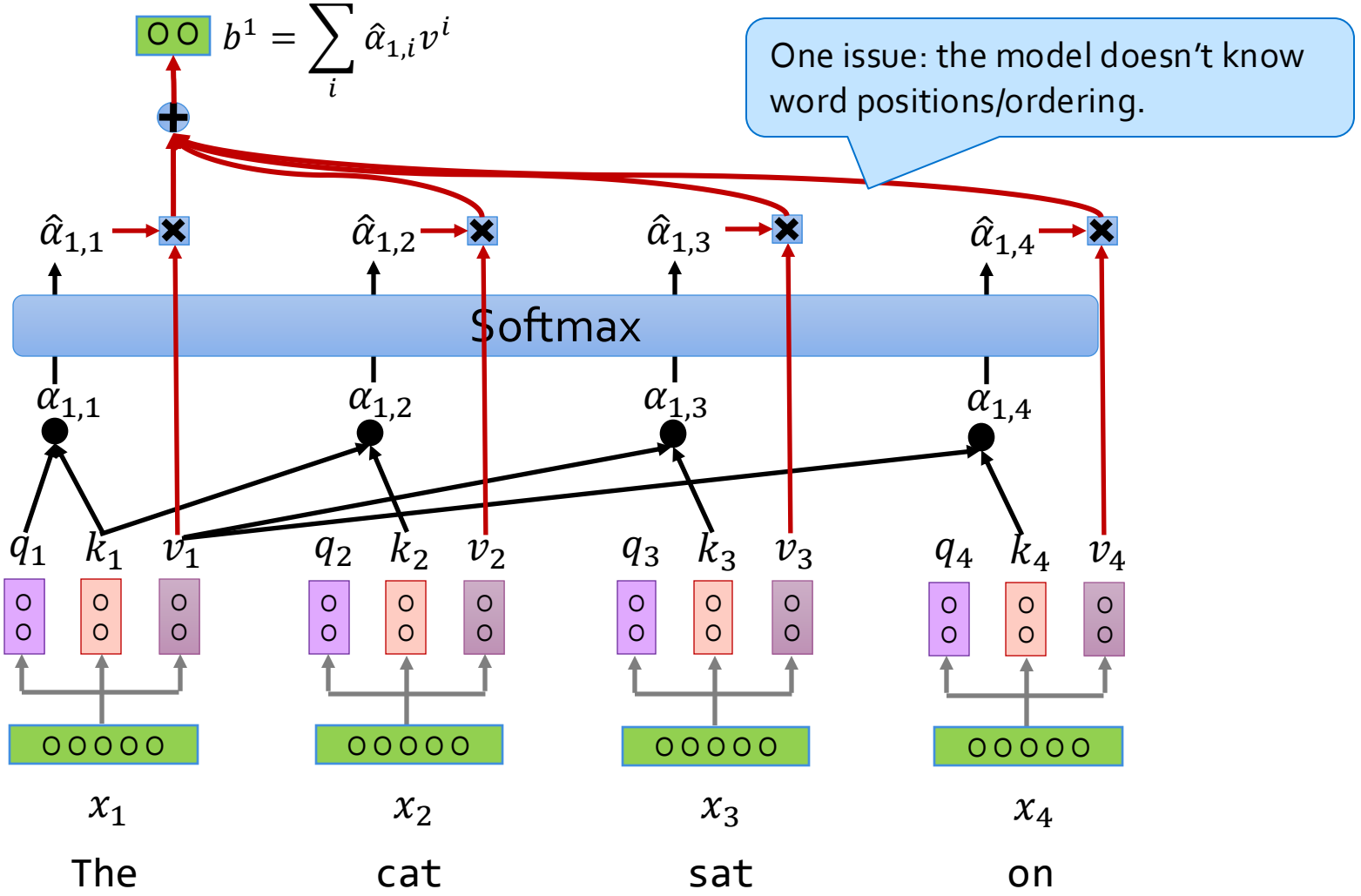
$$\text{logits} = \text{Linear}_{(\text{d, V})}(out) = f\left(out \,.\, W_V\right) \in \mathbb{R}^{S \times V}$$

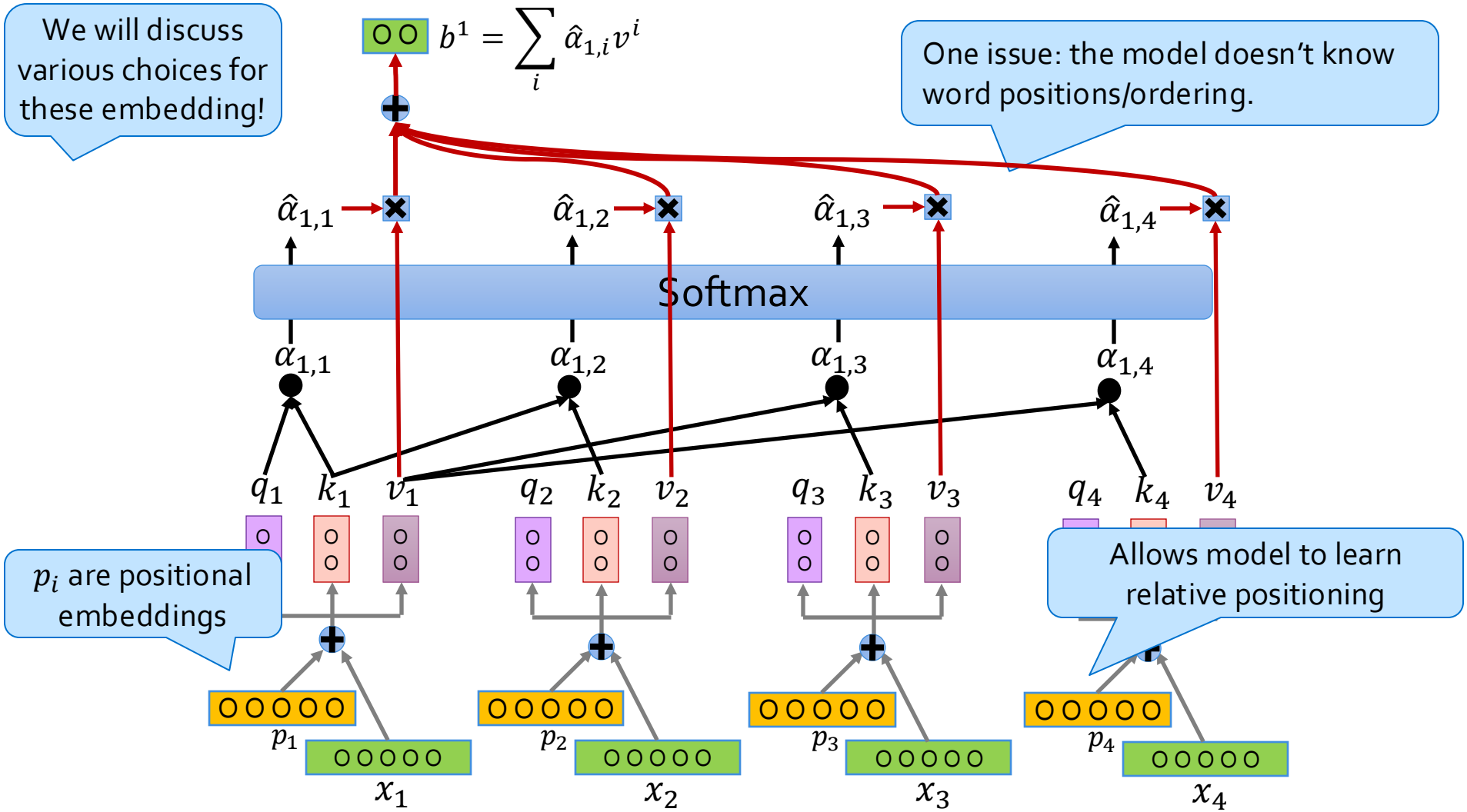
$$\text{probabilies} = \text{softmax}(\text{logits}) \in \mathbb{R}^{S \times V}$$



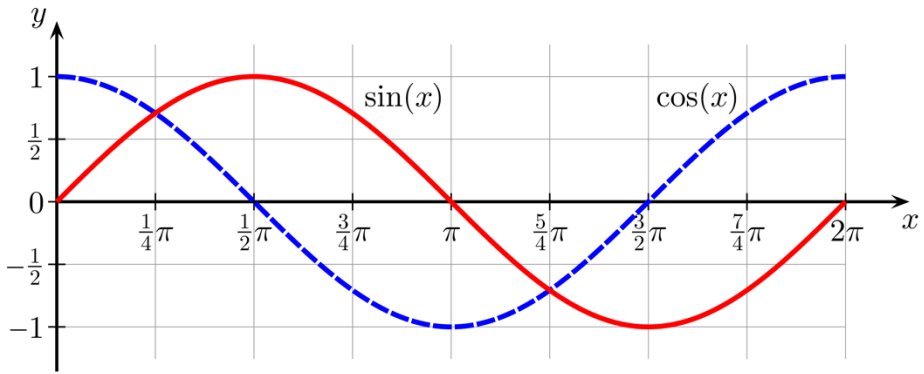


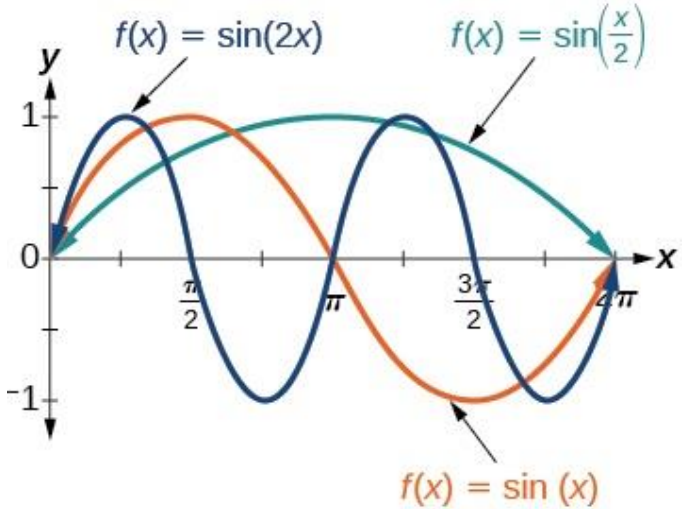
One last wrinkle though …


$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$
One issue: the model doesn't know word positions/ordering.
$\hat{\alpha}_{1,1}$
$\hat{\alpha}_{1,2}$
$\hat{\alpha}_{1,3}$
$\hat{\alpha}_{1,4}$
Softmax
$\alpha_{1,1}$
$\alpha_{1,2}$
$\alpha_{1,3}$
$\alpha_{1,4}$
q_1 k_1 v_1
q_2 k_2 v_2
q_3 k_3 v_3
q_4 k_4 v_4
x_1 x_2 x_3 x_4
The cat sat on


We will discuss various choices for these embedding!
$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$
One issue: the model doesn't know word positions/ordering.
$\hat{\alpha}_{1,1}$
$\hat{\alpha}_{1,2}$
$\hat{\alpha}_{1,3}$
$\hat{\alpha}_{1,4}$
Softmax
$\alpha_{1,1}$
$\alpha_{1,2}$
$\alpha_{1,3}$
$\alpha_{1,4}$
q_1 k_1 v_1
q_2 k_2 v_2
q_3 k_3 v_3
q_4 k_4 v_4
p_i are positional embeddings
Allows model to learn relative positioning
p_1
p_2
p_3
p_4
x_1
x_2
x_3
x_4

Math Recap: Sine and Cosine Functions





Absolute Positional Embeddings

- Let t be a desired position. Then the i-th element of the positional vector is:

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases} \qquad \omega_k = \frac{1}{10000^{2k/d}}$$

- Here d is the maximum dimension.
- This provides unique vectors for each position.

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Quiz

- Let t be a desired position:

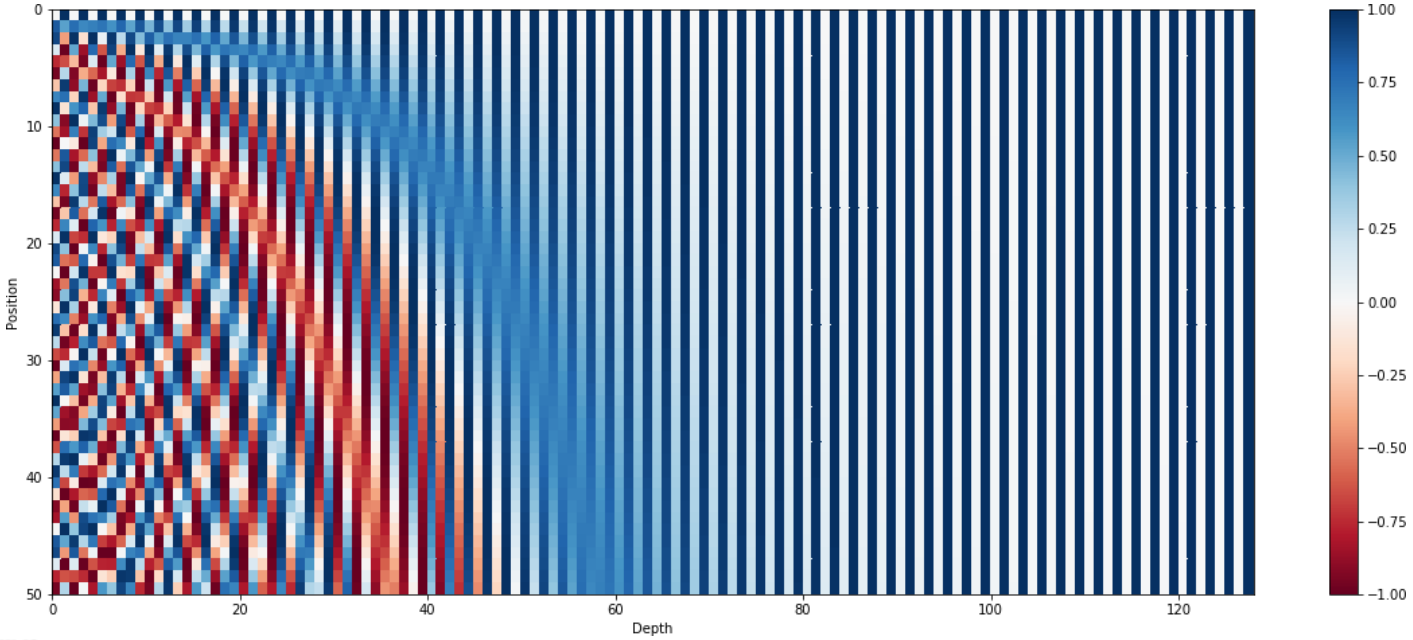
$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases} \qquad \omega_k = \frac{1}{10000^{2k/d}}$$

- **Q:** Are the frequencies increasing with dimension i ?
- **Answer:** The frequencies are decreasing along the vector dimension.

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Visualizing Absolute Positional Embeddings

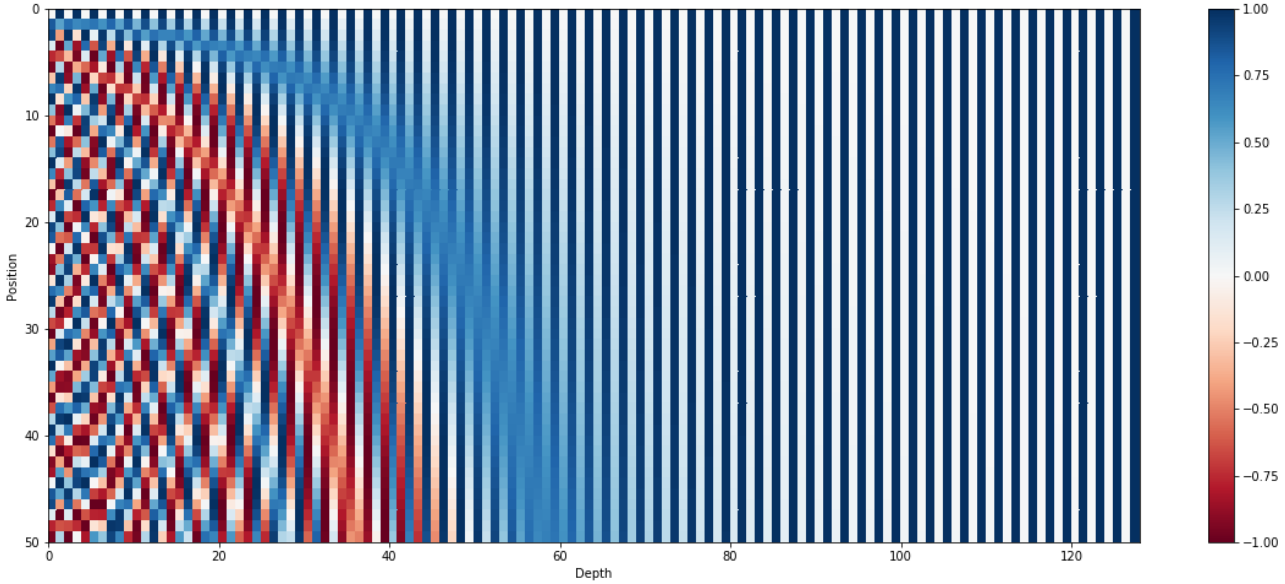
- Here positions range from 0-50, for an embedding dimension of 130.

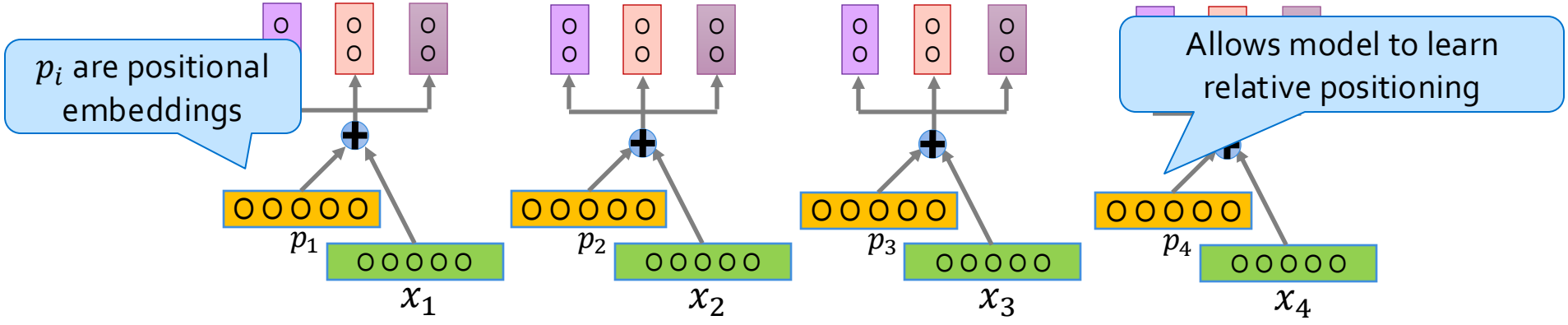


An approach:

Sine/Cosine encoding

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$





Transformer-based Language Modeling

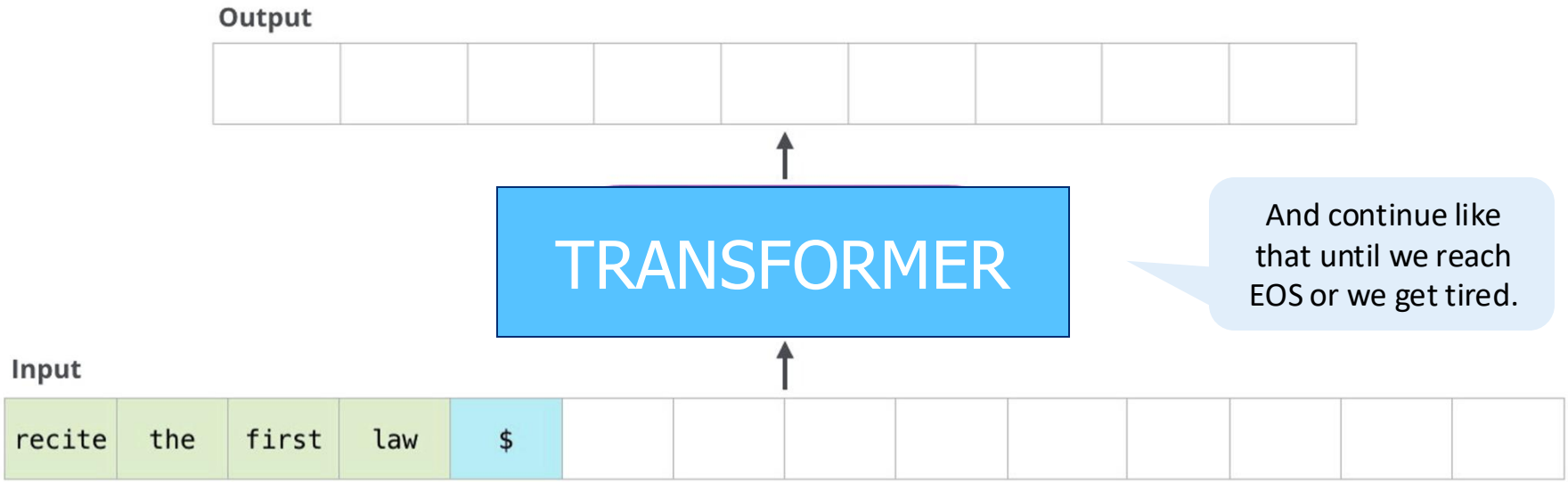
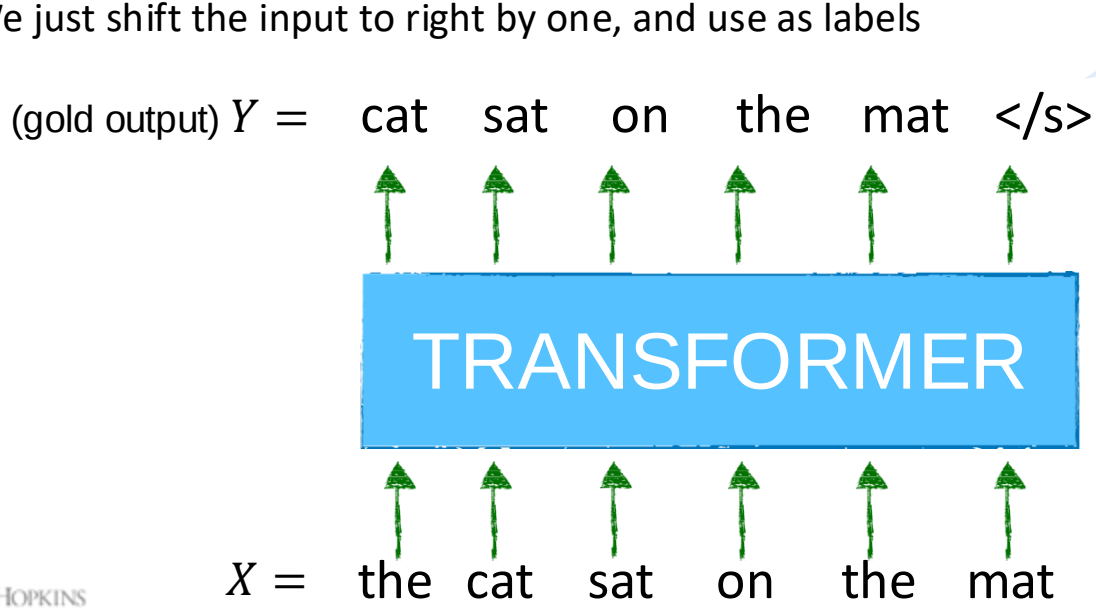


Image by http://jalammar.github.io/illustrated-gpt2/

Training a Transformer Language Model

- **Goal:** Train a Transformer for language modeling (i.e., predicting the next word).
- **Approach:** Train it so that each position is predictor of the next (right) token.
  - We just shift the input to right by one, and use as labels



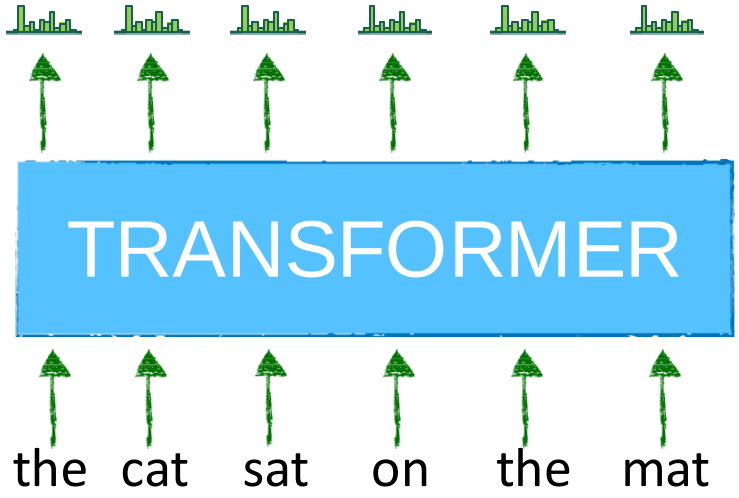

```
X = text[:, :-1]
Y = text[:, 1:]
```

[Slide credit: Arman Cohan]

Training a Transformer Language Model

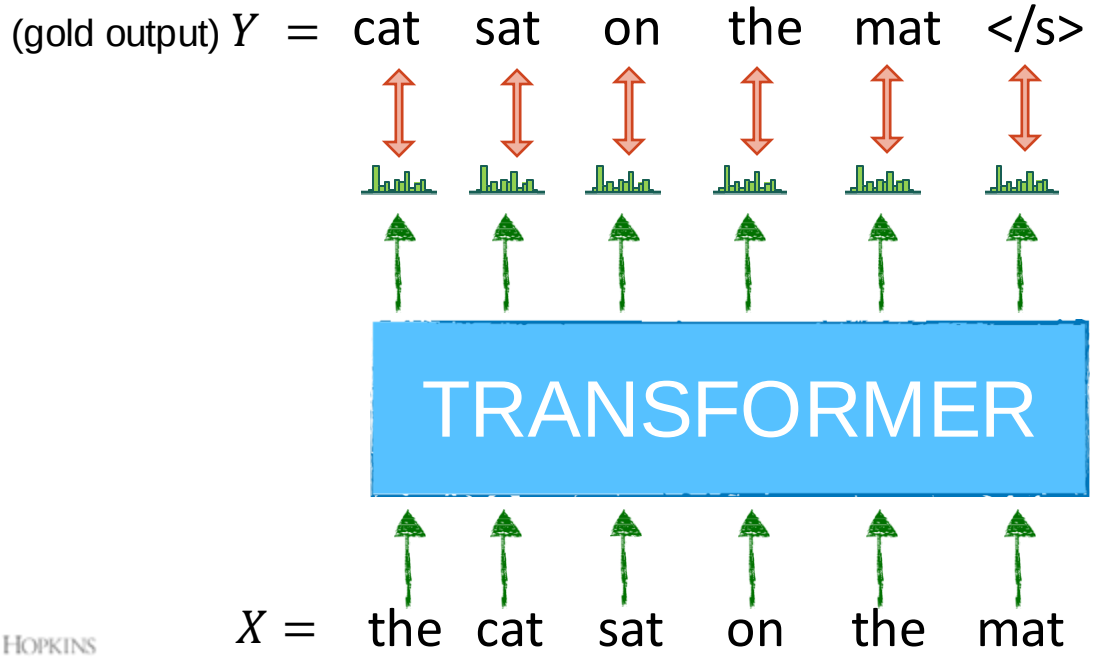
- For each position, compute their corresponding **distribution** over the whole vocab.

(gold output) Y = cat sat on the mat </s>



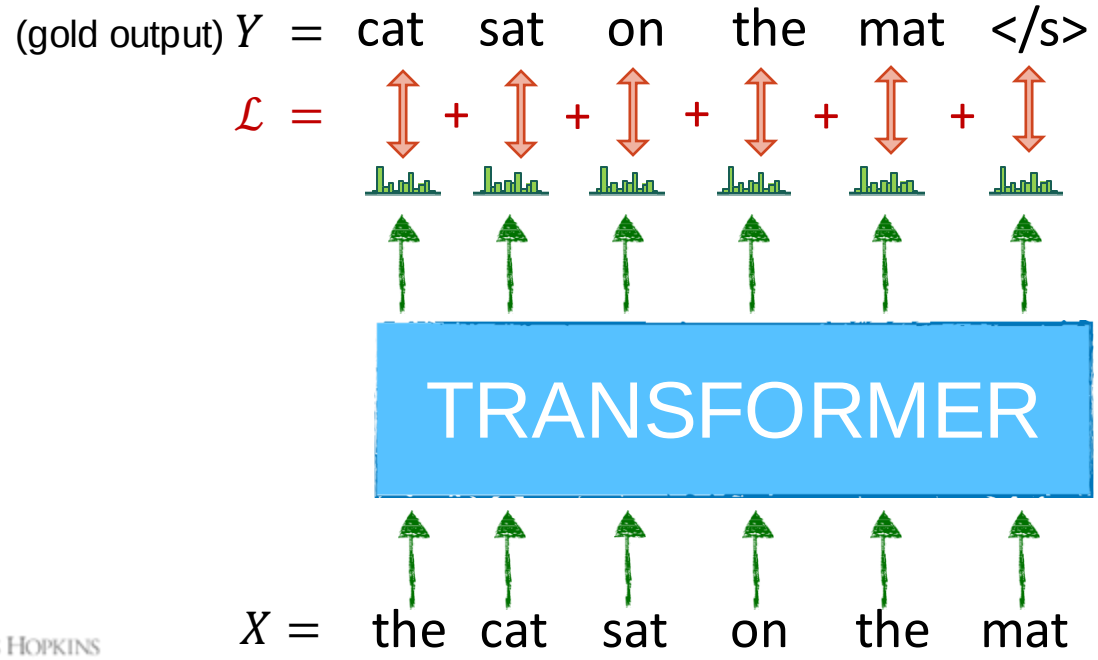
Training a Transformer Language Model

- For each position, compute the **loss** between the distribution and the gold output label.



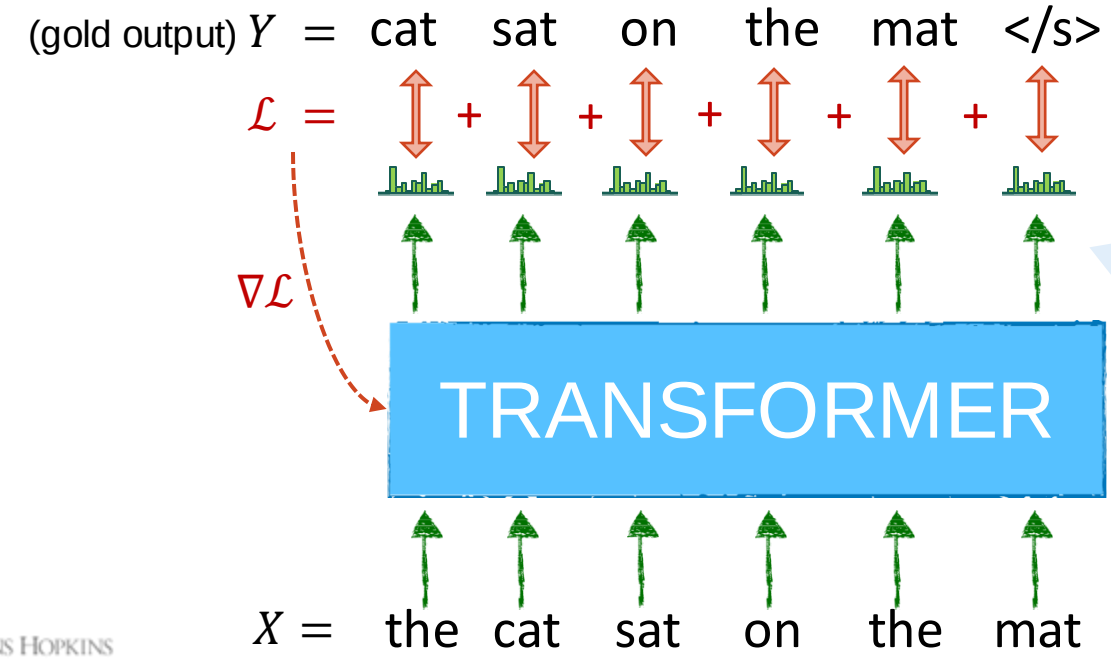
Training a Transformer Language Model

- Sum the position-wise loss values to a obtain a **global loss**.



Training a Transformer Language Model

- Using this loss, do **Backprop** and **update** the Transformer parameters.



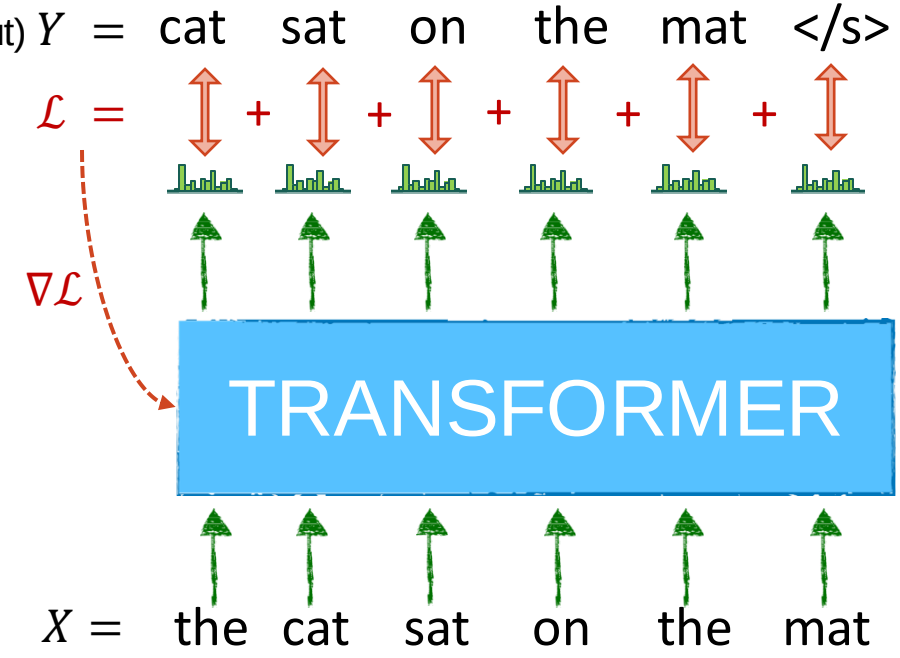
Well, this is not quite right 🤡

…

what is the problem with this?

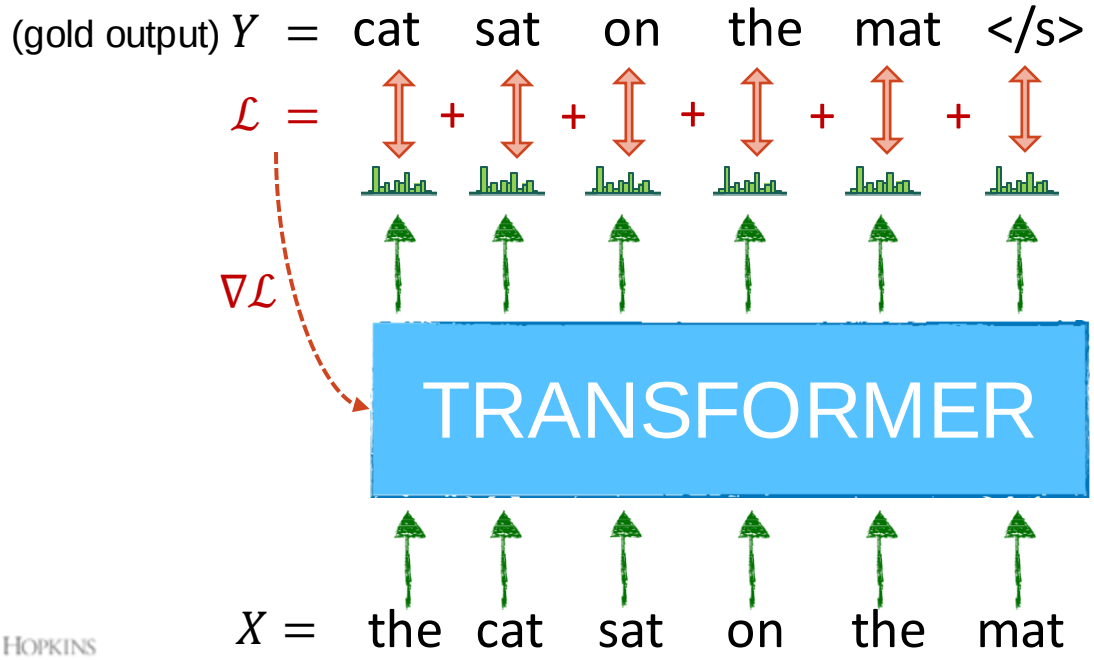
Training a Transformer Language Model

- The model would solve the task by copying the next token to output (data leakage).
  - Does not learn anything useful



Training a Transformer Language Model

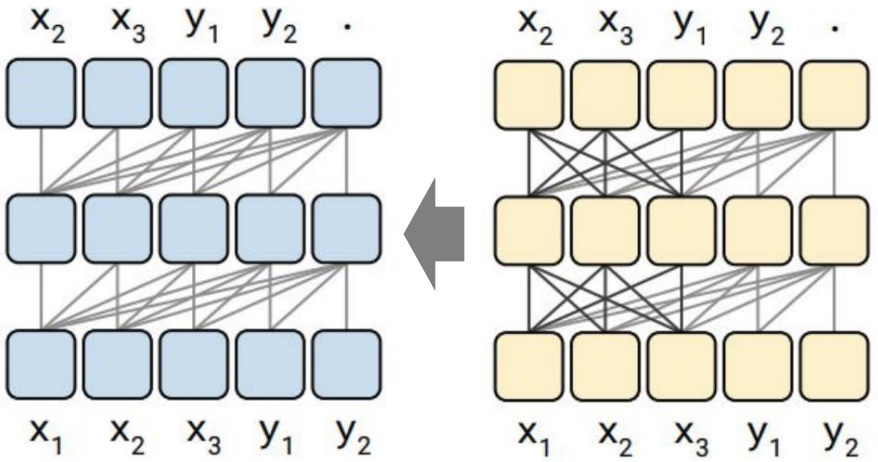
- We need to **prevent information leakage** from future tokens! How?



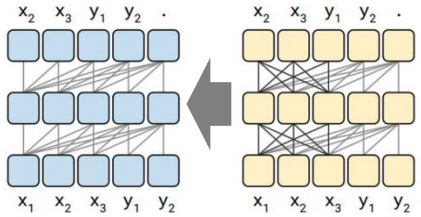
Attention mask

Attention raw scores

	1	2	3	4	5	6	7	8	9	10	11	12
0	-0.08	1.24	0.69	-0.98	1.43	-0.6	0.7	0.16	0.93	1.28	-1.61	-1.1
1	-0.09	-0.0	-0.7	0.06	0.25	0.23	0.26	0.18	0.78	-0.21	-1.01	1.01
2	0.86	1.19	1.59	0.86	-0.13	-0.15	-2.13	-0.98	-0.87	-1.72	1.87	-0.72
3	0.12	-0.03	-0.02	0.88	-0.46	-0.7	0.54	-0.42	-1.89	-0.38	0.04	-0.84
4	0.51	0.17	0.13	-1.64	0.24	-0.02	1.68	-0.36	0.64	0.36	0.27	0.66
5	0.24	-1.44	0.43	0.74	0.96	-1.21	-0.31	1.54	1.66	1.14	0.58	-1.44
6	0.26	-0.1	0.93	0.72	-0.38	1.65	0.47	-0.96	-0.17	-0.9	-1.57	0.22
7	-0.55	0.81	0.71	1.7	-0.8	-1.14	-0.32	1.78	-0.7	-0.04	1.54	0.81
8	0.74	-0.76	-0.44	-0.08	-1.38	-0.13	1.25	-1.37	1.84	0.3	0.57	0.74
9	-0.97	-0.91	0.15	0.35	-0.81	0.11	1.14	-1.52	1.06	1.87	0.5	-0.3
10	1.56	0.9	0.39	1.46	1.44	-1.05	0.9	-0.73	0.36	-0.67	-0.62	-0.43
11	0.32	0.74	0.44	-0.1	1.19	0.83	0.29	2.06	0.51	-0.26	1.51	0.11



Attention mask



Attention raw scores

	1	2	3	4	5	6	7	8	9	10	11	12
0	-0.08	1.24	0.69	-0.98	1.43	-0.6	0.7	0.16	0.93	1.28	-1.61	-1.1
1	-0.09	-0.0	-0.7	0.06	0.25	0.23	0.26	0.18	0.78	-0.21	-1.01	1.01
2	0.86	1.19	1.59	0.86	-0.13	-0.15	-2.13	-0.98	-0.87	-1.72	1.87	-0.72
3	0.12	-0.03	-0.02	0.88	-0.46	-0.7	0.54	-0.42	-1.89	-0.38	0.04	-0.84
4	0.51	0.17	0.13	-1.64	0.24	-0.02	1.68	-0.36	0.64	0.36	0.27	0.66
5	0.24	-1.44	0.43	0.74	0.96	-1.21	-0.31	1.54	1.66	1.14	0.58	-1.44
6	0.26	-0.1	0.93	0.72	-0.38	1.65	0.47	-0.96	-0.17	-0.9	-1.57	0.22
7	-0.55	0.81	0.71	1.7	-0.8	-1.14	-0.32	1.78	-0.7	-0.04	1.54	0.81
8	0.74	-0.76	-0.44	-0.08	-1.38	-0.13	1.25	-1.37	1.84	0.3	0.57	0.74
9	-0.97	-0.91	0.15	0.35	-0.81	0.11	1.14	-1.52	1.06	1.87	0.5	-0.3
10	1.56	0.9	0.39	1.46	1.44	-1.05	0.9	-0.73	0.36	-0.67	-0.62	-0.43
11	0.32	0.74	0.44	-0.1	1.19	0.83	0.29	2.06	0.51	-0.26	1.51	0.11

Attention mask

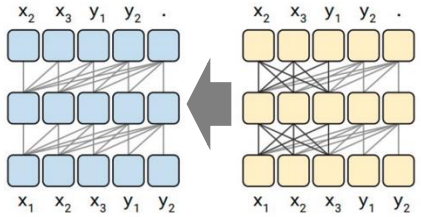
	0	1	2	3	4	5	6	7	8	9	10	11
0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
1	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
2	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
3	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
4	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf
5	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf
6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf
7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf
8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf
9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf
10	1.0	1.0	1.0	1.0								
11	1.0	1.0	1.0	1.0								

0.30 0.25 0.20 0.15 0.10 0.05 0.00

large negative numbers,
which leads to softmax$(-\infty) \approx 0$

Slide credit: Arman Cohan

Attention mask



Attention raw scores

	1	2	3	4	5	6	7	8	9	10	11
0	-0.08	1.24	0.69	-0.98	1.43	-0.6	0.7	0.16	0.93	1.28	-1.61	-1.1
1	-0.09	-0.0	-0.7	0.06	0.25	0.23	0.26	0.18	0.78	-0.21	-1.01	1.01
2	0.86	1.19	1.59	0.86	-0.13	-0.15	-2.13	-0.98	-0.87	-1.72	1.87	-0.72
3	0.12	-0.03	-0.02	0.88	-0.46	-0.7	0.54	-0.42	-1.89	-0.38	0.04	-0.84
4	0.51	0.17	0.13	-1.64	0.24	-0.02	1.68	-0.36	0.64	0.36	0.27	0.66
5	0.24	-1.44	0.43	0.74	0.96	-1.21	-0.31	1.54	1.66	1.14	0.58	-1.44
6	0.26	-0.1	0.93	0.72	-0.38	1.65	0.47	-0.96	-0.17	-0.9	-1.57	0.22
7	-0.55	0.81	0.71	1.7	-0.8	-1.14	-0.32	1.78	-0.7	-0.04	1.54	0.81
8	0.74	-0.76	-0.44	-0.08	-1.38	-0.13	1.25	-1.37	1.84	0.3	0.57	0.74
9	-0.97	-0.91	0.15	0.35	-0.81	0.11	1.14	-1.52	1.06	1.87	0.5	-0.3
10	1.56	0.9	0.39	1.46	1.44	-1.05	0.9	-0.73	0.36	-0.67	-0.62	-0.43
11	0.32	0.74	0.44	-0.1	1.19	0.83	0.29	2.06	0.51	-0.26	1.51	0.11

X

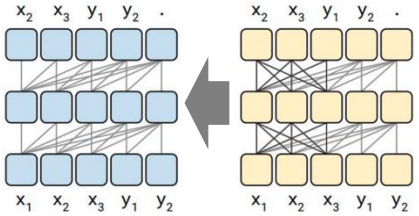
Attention mask

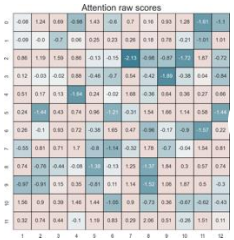
	0	1	2	3	4	5	6	7	8	9	10	11
0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
1	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
2	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
3	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
4	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf
5	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf
6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf
7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf
8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf
9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf
10	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf
11	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Note matrix multiplication is quite fast in GPUs.

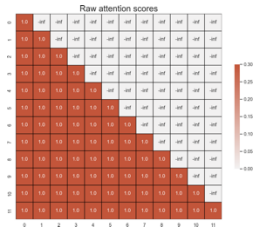
Slide credit: Arman Cohan

Attention mask





X



=

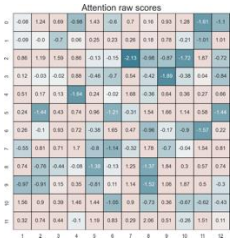
Masked attention raw scores

	1	2	3	4	5	6	7	8	9	10	11	12
0	-0.08	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
1	-0.09	-0.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
2	0.86	1.19	1.59	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
3	0.12	-0.03	-0.02	0.88	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
4	0.51	0.17	0.13	-1.64	0.24	-inf	-inf	-inf	-inf	-inf	-inf	-inf
5	0.24	-1.44	0.43	0.74	0.96	-1.21	-inf	-inf	-inf	-inf	-inf	-inf
6	0.26	-0.1	0.93	0.72	-0.38	1.65	0.47	-inf	-inf	-inf	-inf	-inf
7	-0.55	0.81	0.71	1.7	-0.8	-1.14	-0.32	1.78	-inf	-inf	-inf	-inf
8	0.74	-0.76	-0.44	-0.08	-1.38	-0.13	1.25	-1.37	1.84	-inf	-inf	-inf
9	-0.97	-0.91	0.15	0.35	-0.81	0.11	1.14	-1.52	1.06	1.87	-inf	-inf
10	1.56	0.9	0.39	1.46	1.44	-1.05	0.9	-0.73	0.36	-0.67	-0.62	-inf
11	0.32	0.74	0.44	-0.1	1.19	0.83	0.29	2.06	0.51	-0.26	1.51	0.11

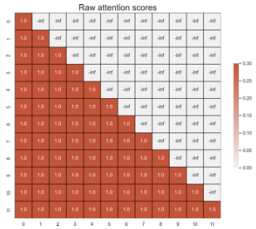
Slide credit: Arman Cohan

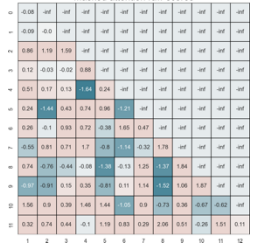
Attention mask

The effect is more than just pruning out some of the wirings in self-attention block.

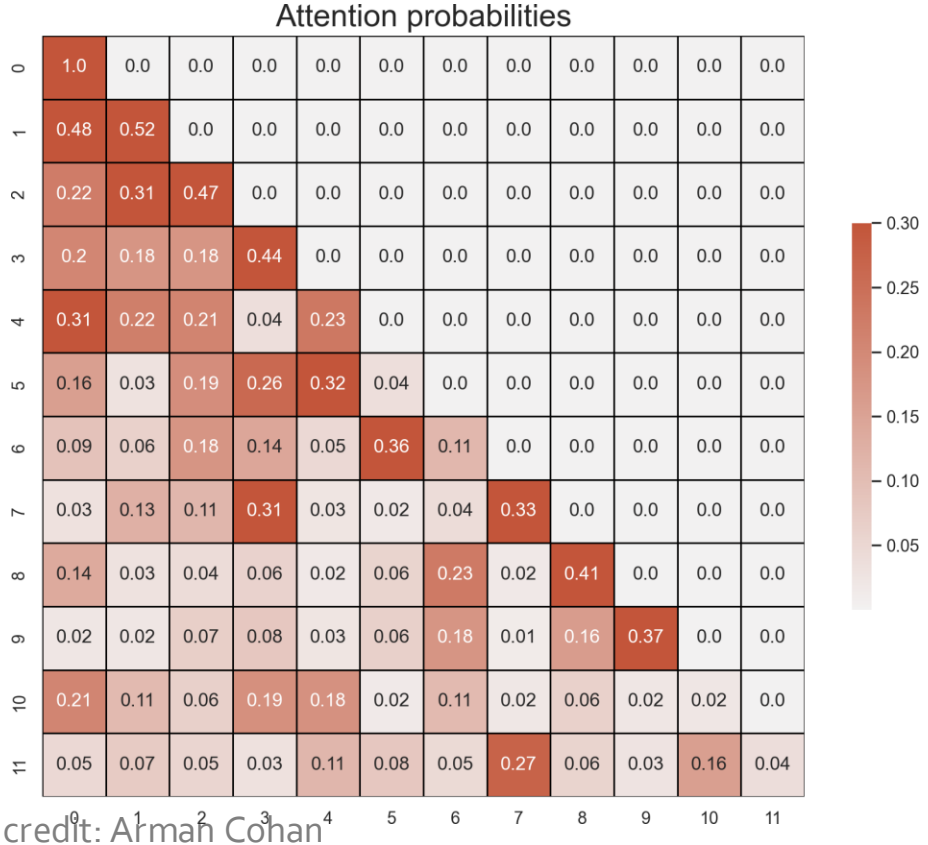


X





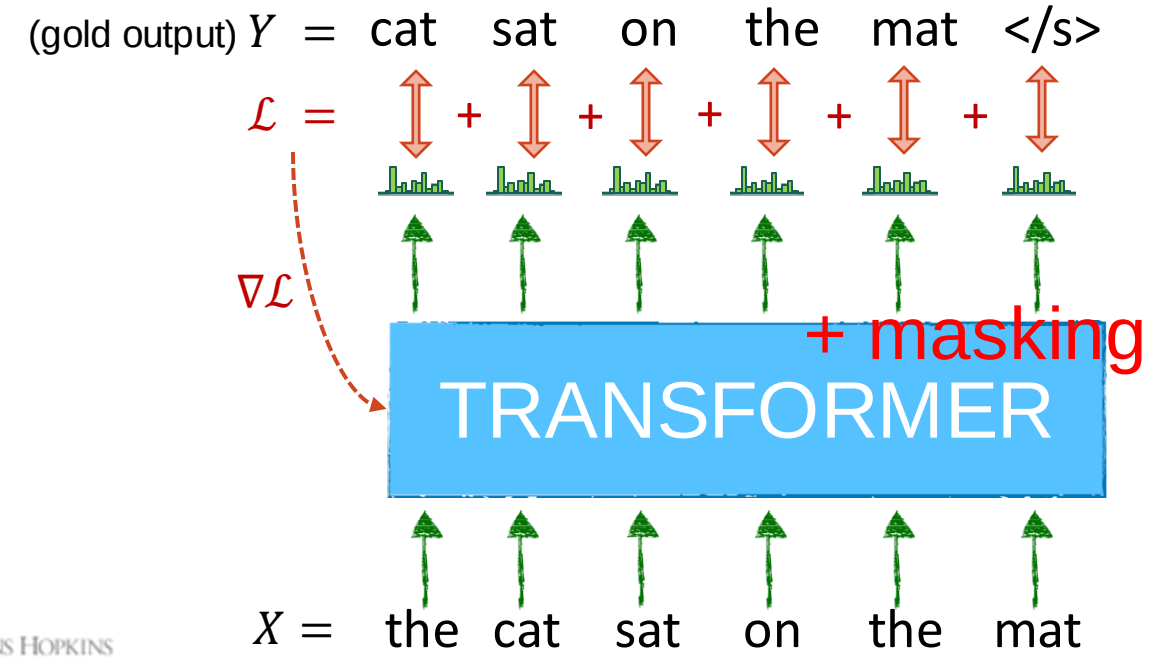
softmax



Slide credit: Arman Cohan

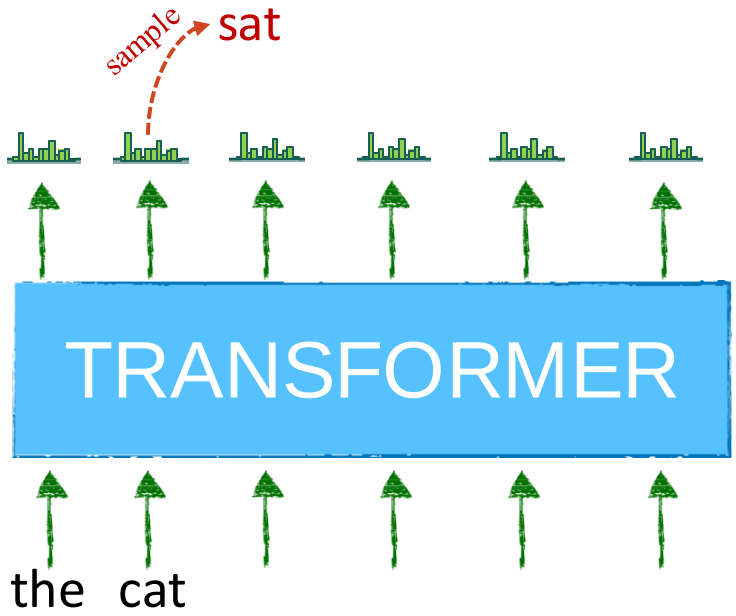
Training a Transformer Language Model

- We need to **prevent information leakage** from future tokens! How?



How to use the model to generate text?

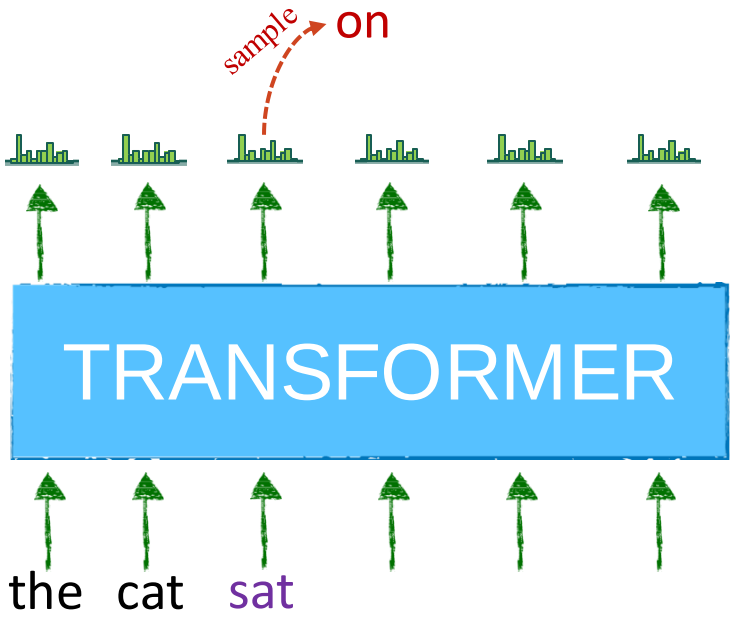
- Use the output of previous step as input to the next step repeatedly



How to use the model to generate text?

- Use the output of previous step as input to the next step repeatedly

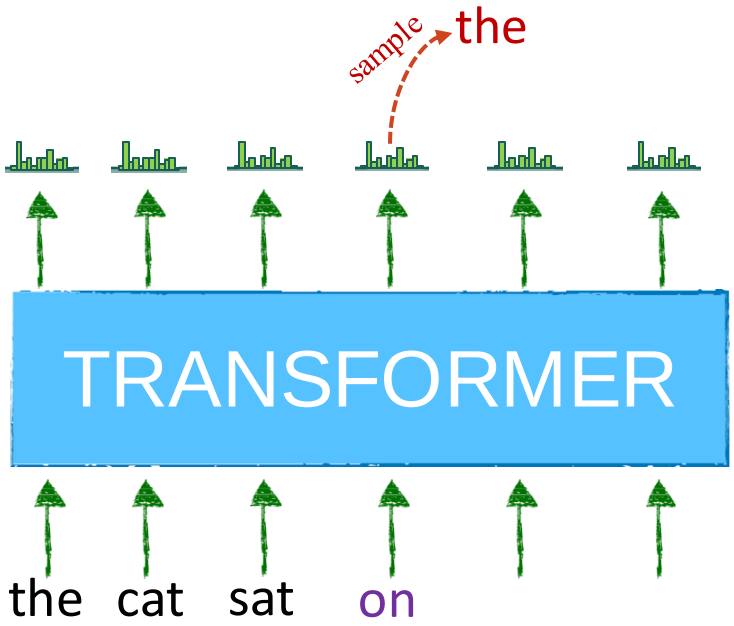
The probabilities get revised upon adding a new token to the input.



How to use the model to generate text?

- Use the output of previous step as input to the next step repeatedly

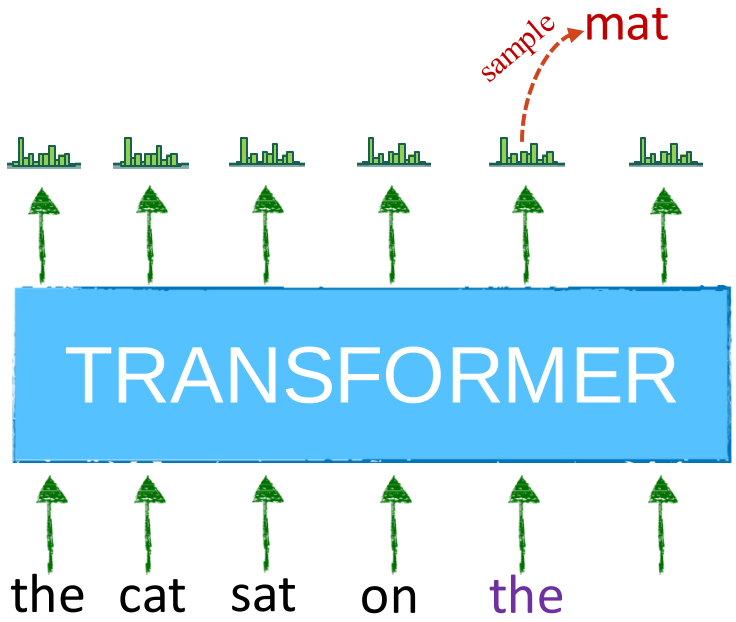
The probabilities get revised upon adding a new token to the input.



How to use the model to generate text?

- Use the output of previous step as input to the next step repeatedly

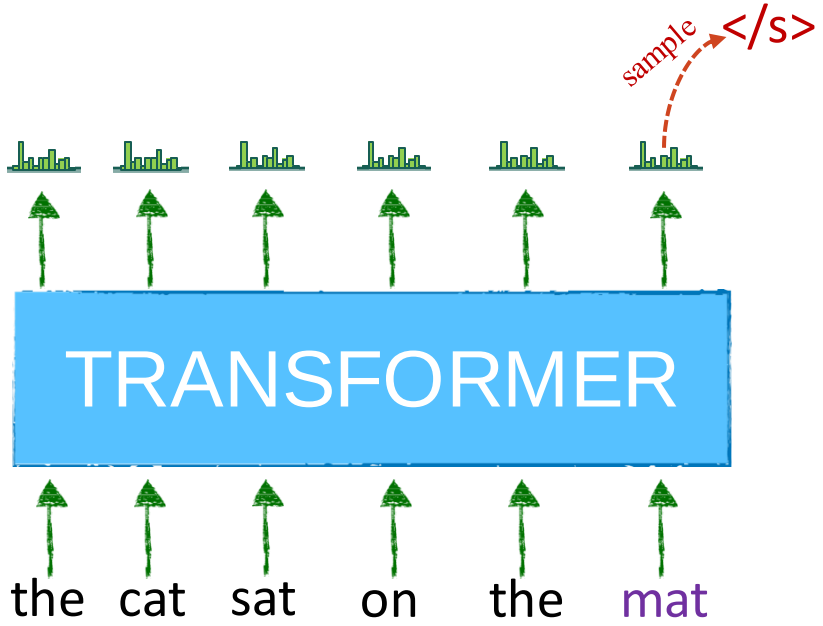
The probabilities get revised upon adding a new token to the input.



How to use the model to generate text?

- Use the output of previous step as input to the next step repeatedly

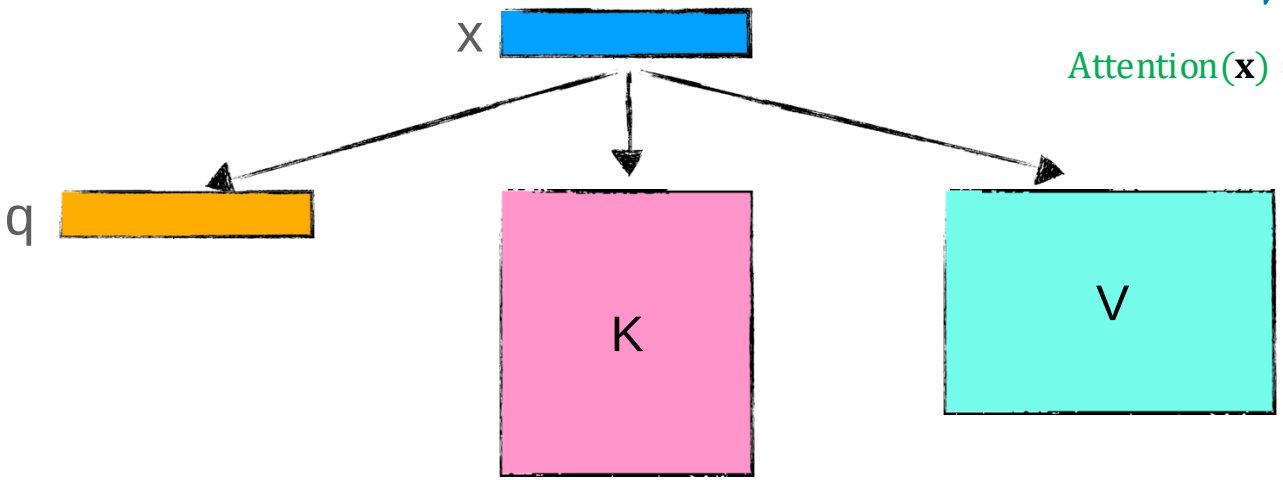
The probabilities get revised upon adding a new token to the input.



An important efficiency consideration about decoding!

Making decoding more efficient

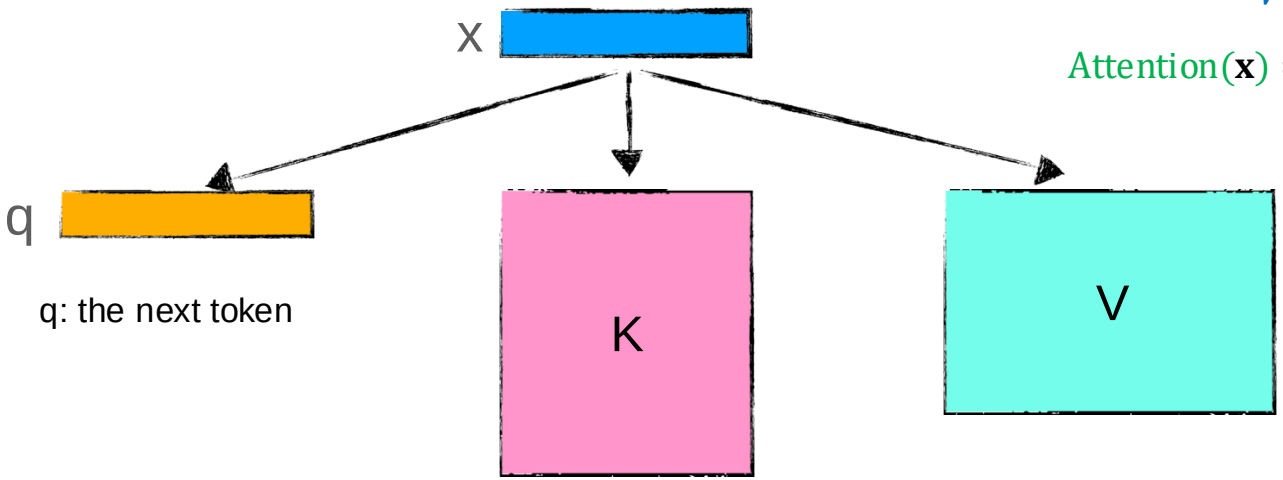
$$Q = \mathbf{W}^q \mathbf{x}$$
$$K = \mathbf{W}^k \mathbf{x}$$
$$V = \mathbf{W}^v \mathbf{x}$$
$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$



[Slide credit: Arman Cohan]

Making decoding more efficient

$$Q = \mathbf{W}^q \mathbf{x}$$
$$K = \mathbf{W}^k \mathbf{x}$$
$$V = \mathbf{W}^v \mathbf{x}$$
$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$



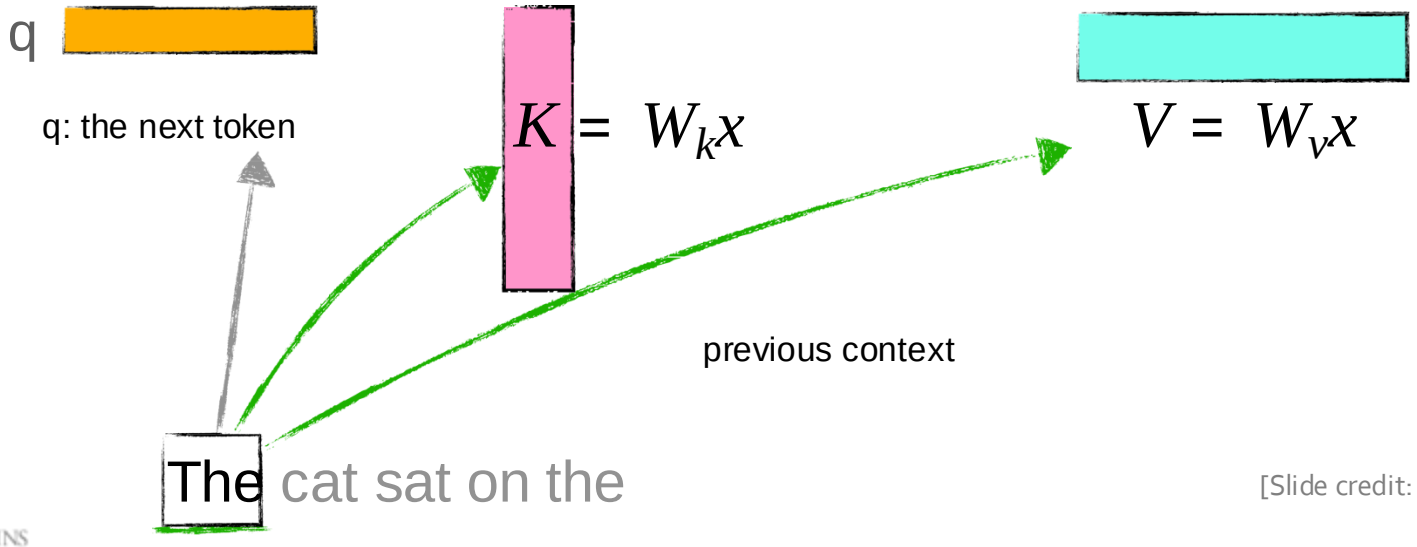
q: the next token

previous context

[Slide credit: Arman Cohan]

Making decoding more efficient

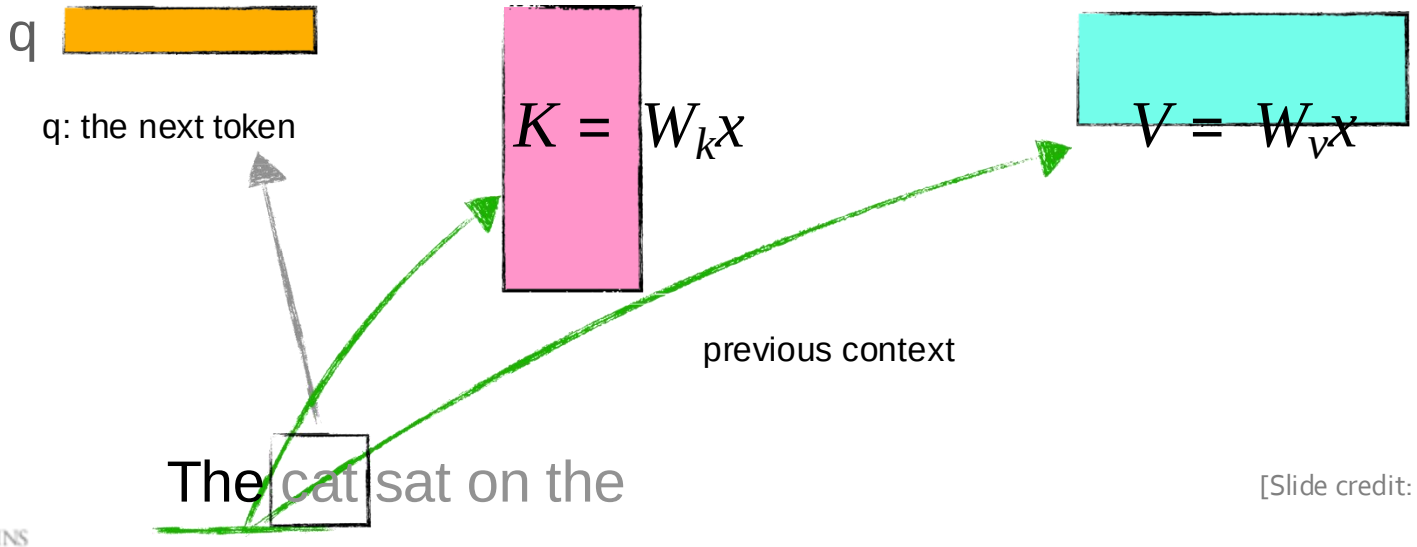
$$Q = \mathbf{W}^q \mathbf{x}$$
$$K = \mathbf{W}^k \mathbf{x}$$
$$V = \mathbf{W}^v \mathbf{x}$$
$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$



[Slide credit: Arman Cohan]

Making decoding more efficient

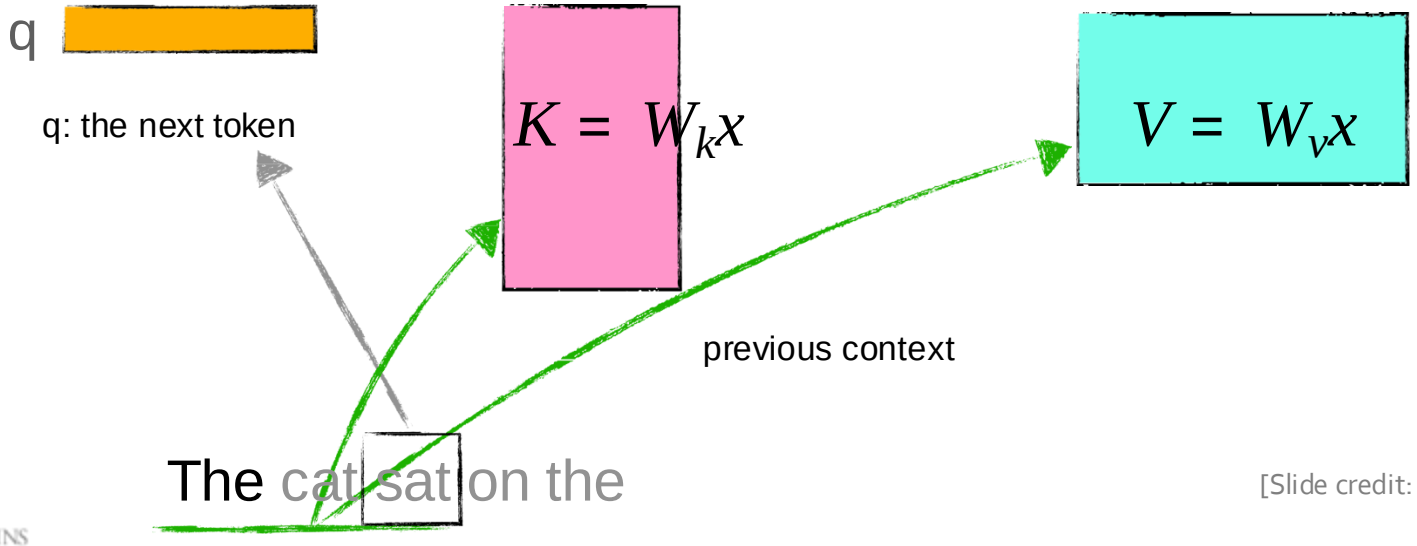
$$Q = \mathbf{W}^q \mathbf{x}$$
$$K = \mathbf{W}^k \mathbf{x}$$
$$V = \mathbf{W}^v \mathbf{x}$$
$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$



[Slide credit: Arman Cohan]

Making decoding more efficient

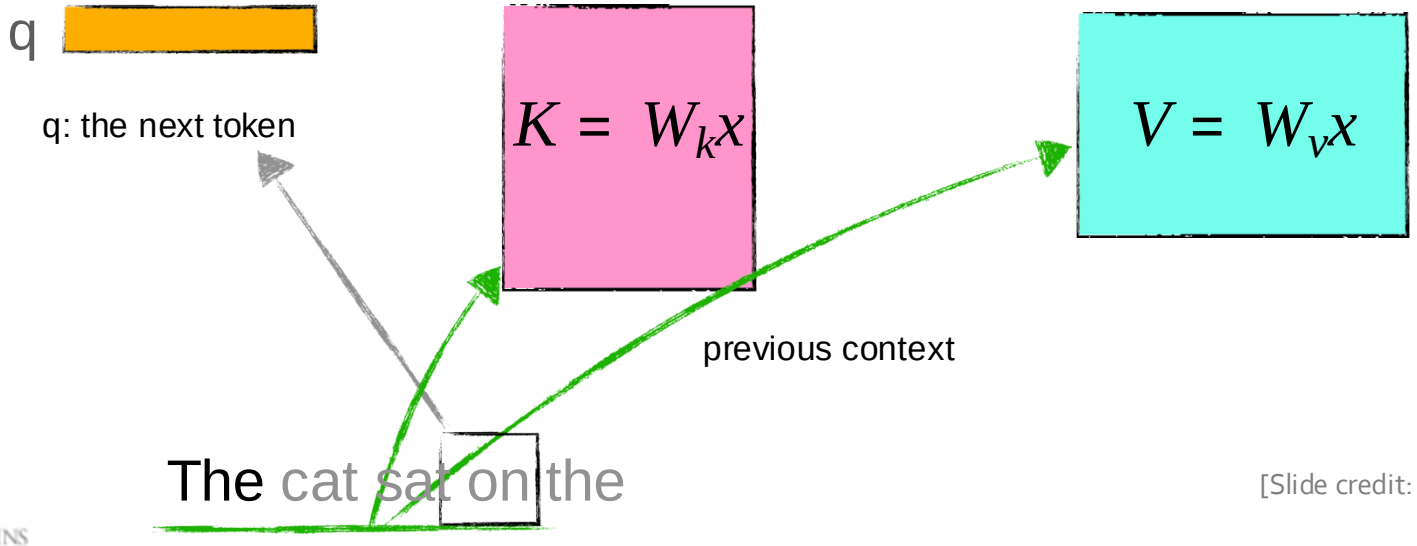
$$Q = \mathbf{W}^q \mathbf{x}$$
$$K = \mathbf{W}^k \mathbf{x}$$
$$V = \mathbf{W}^v \mathbf{x}$$
$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$



[Slide credit: Arman Cohan]

Making decoding more efficient

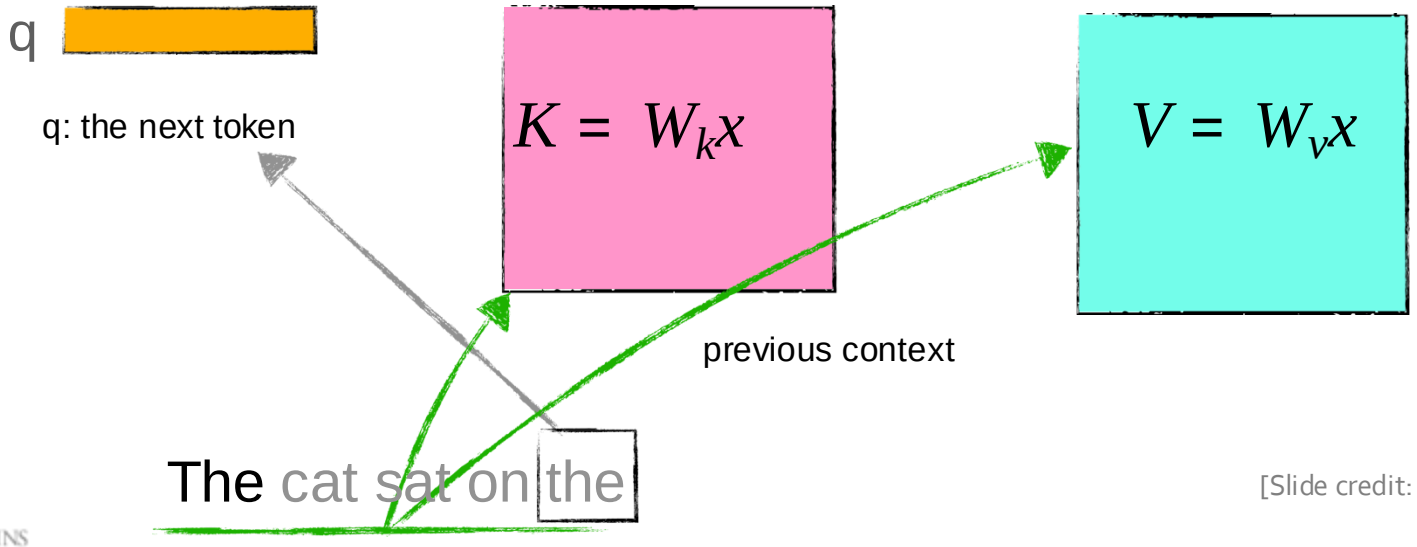
$$Q = \mathbf{W}^q \mathbf{x}$$
$$K = \mathbf{W}^k \mathbf{x}$$
$$V = \mathbf{W}^v \mathbf{x}$$
$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$



[Slide credit: Arman Cohan]

Making decoding more efficient

$$Q = \mathbf{W}^q \mathbf{x}$$
$$K = \mathbf{W}^k \mathbf{x}$$
$$V = \mathbf{W}^v \mathbf{x}$$
$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$



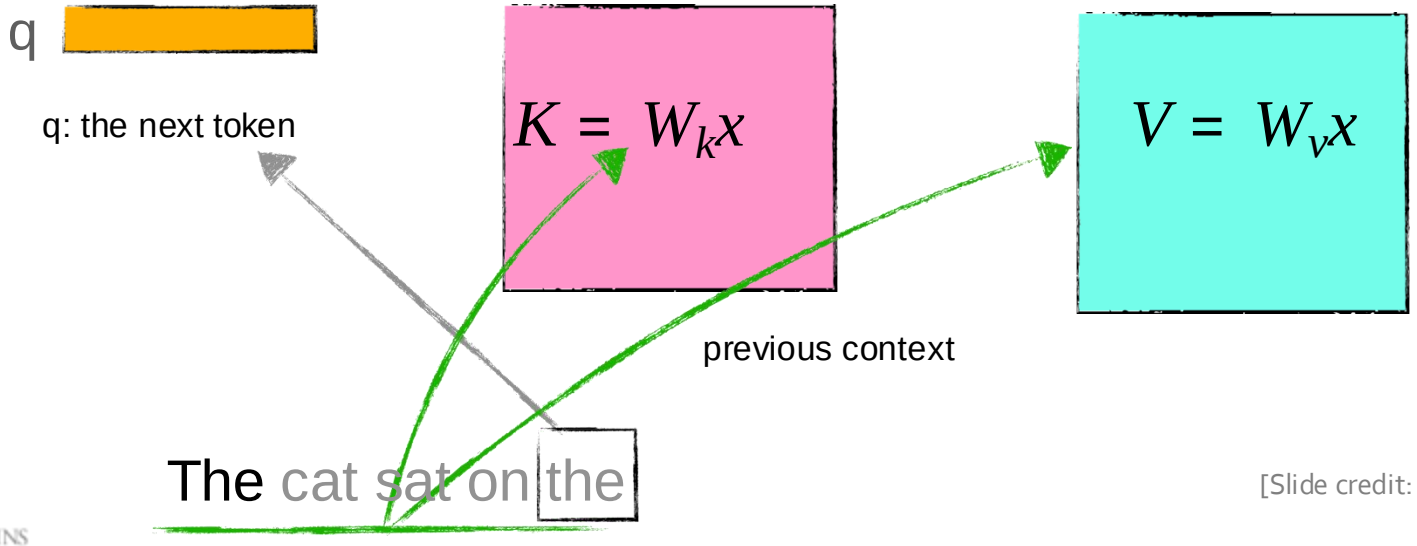
[Slide credit: Arman Cohan]

Making decoding more efficient

$$Q = \mathbf{W}^q \mathbf{x}$$
$$K = \mathbf{W}^k \mathbf{x}$$
$$V = \mathbf{W}^v \mathbf{x}$$

$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$

- We are computing the Keys and Values many times!
  - Let's reduce redundancy! 😖



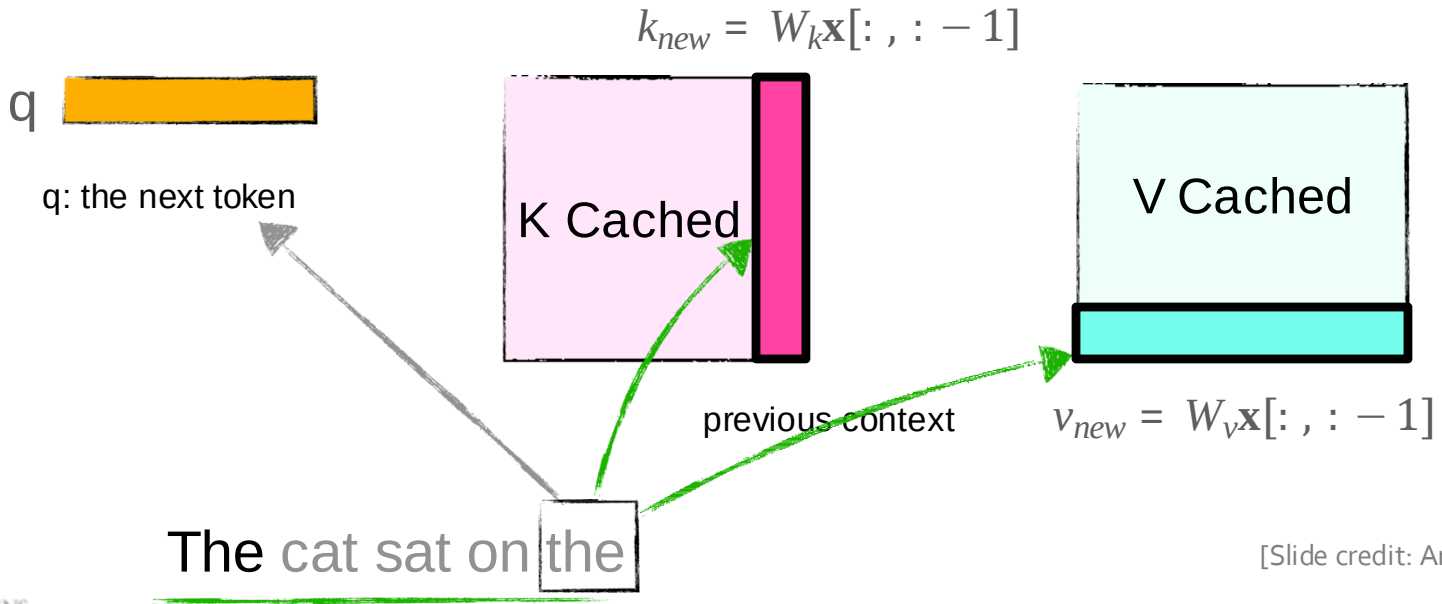
[Slide credit: Arman Cohan]

Making decoding more efficient

$$Q = \mathbf{W}^q\mathbf{x}$$
$$K = \mathbf{W}^k\mathbf{x}$$
$$V = \mathbf{W}^v\mathbf{x}$$

$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$

- We are computing the Keys and Values many times!
  - Let's reduce redundancy! 😵



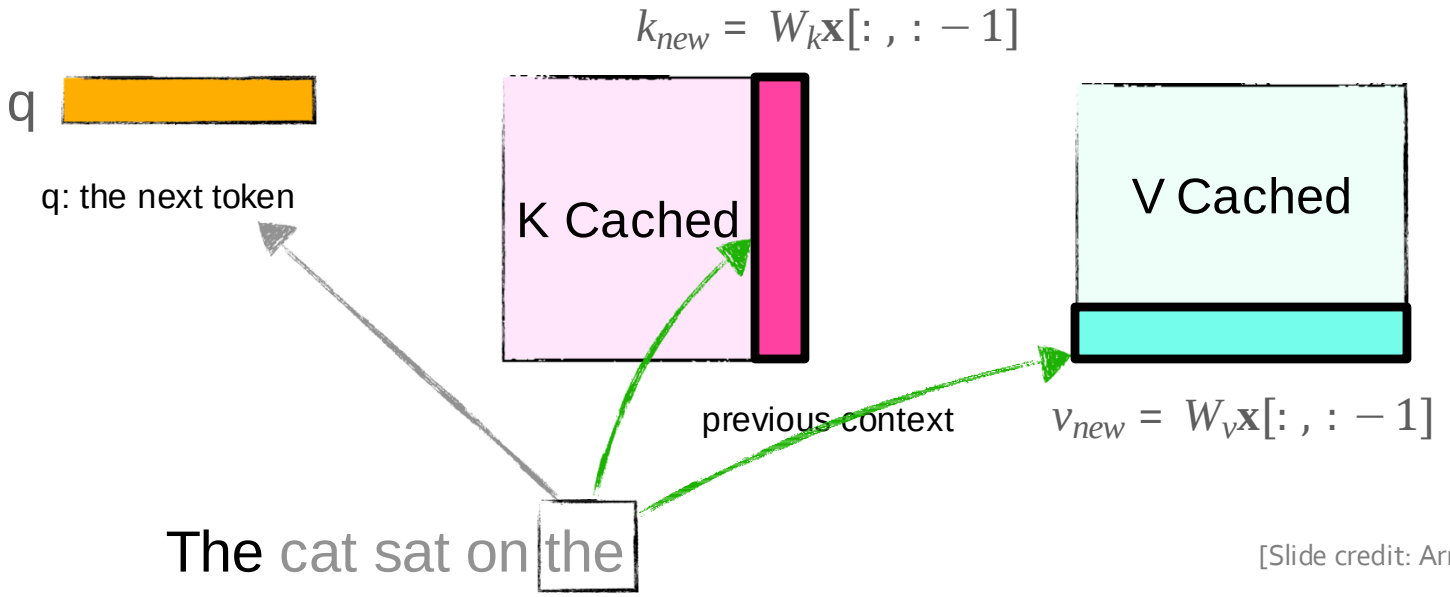
[Slide credit: Arman Cohan]

Making decoding more efficient

$$Q = \mathbf{W}^q\mathbf{x}$$
$$K = \mathbf{W}^k\mathbf{x}$$
$$V = \mathbf{W}^v\mathbf{x}$$

$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$

- **Question:** How much memory does this K, V cache require?



[Slide credit: Arman Cohan]

Summary

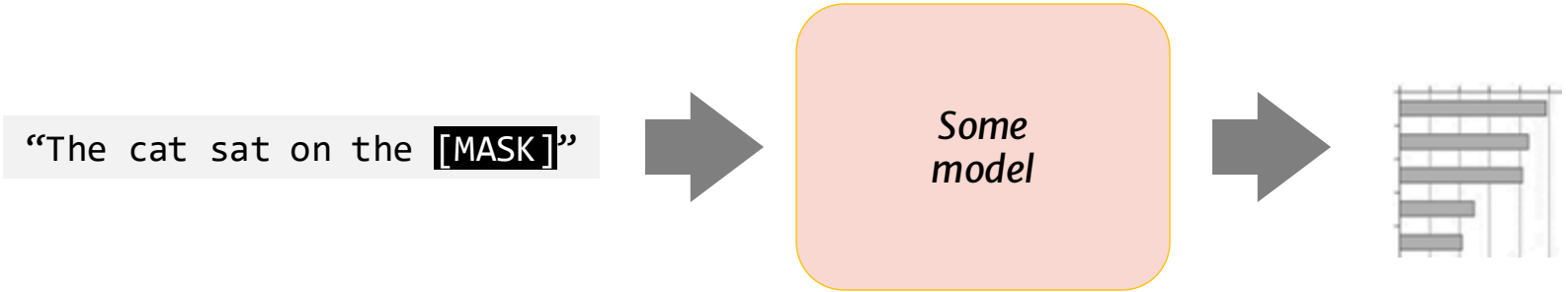
- This is a very generic Transformer!
- We will implement this in HW5 to build a simple Transformer Language Model!!

- **Next:**
  - Architectural variants
  - Efficiency issues.
  - …

Transformer Architectural Variants

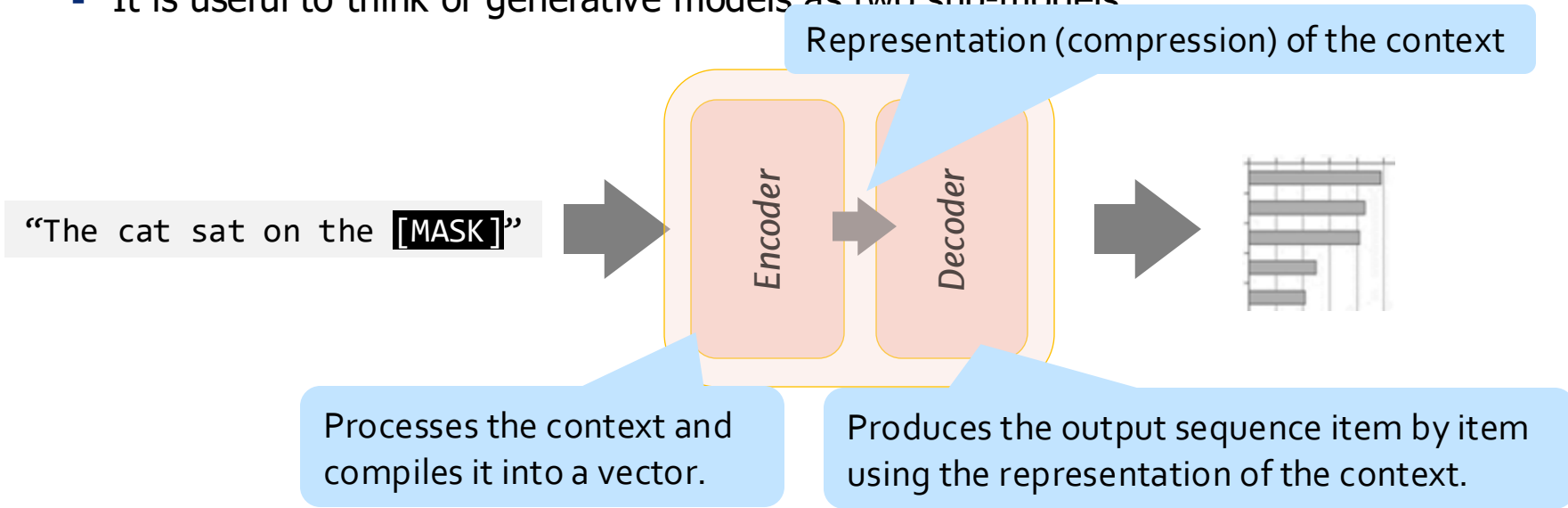
Encoder-Decoder Architectures

- It is useful to think of generative models as two sub-models.



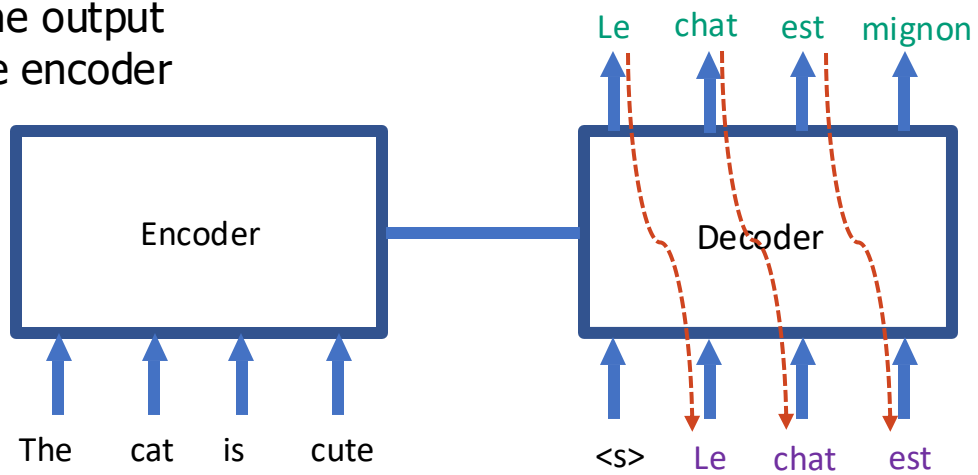
Encoder-Decoder Architectures

- It is useful to think of generative models as two sub-models

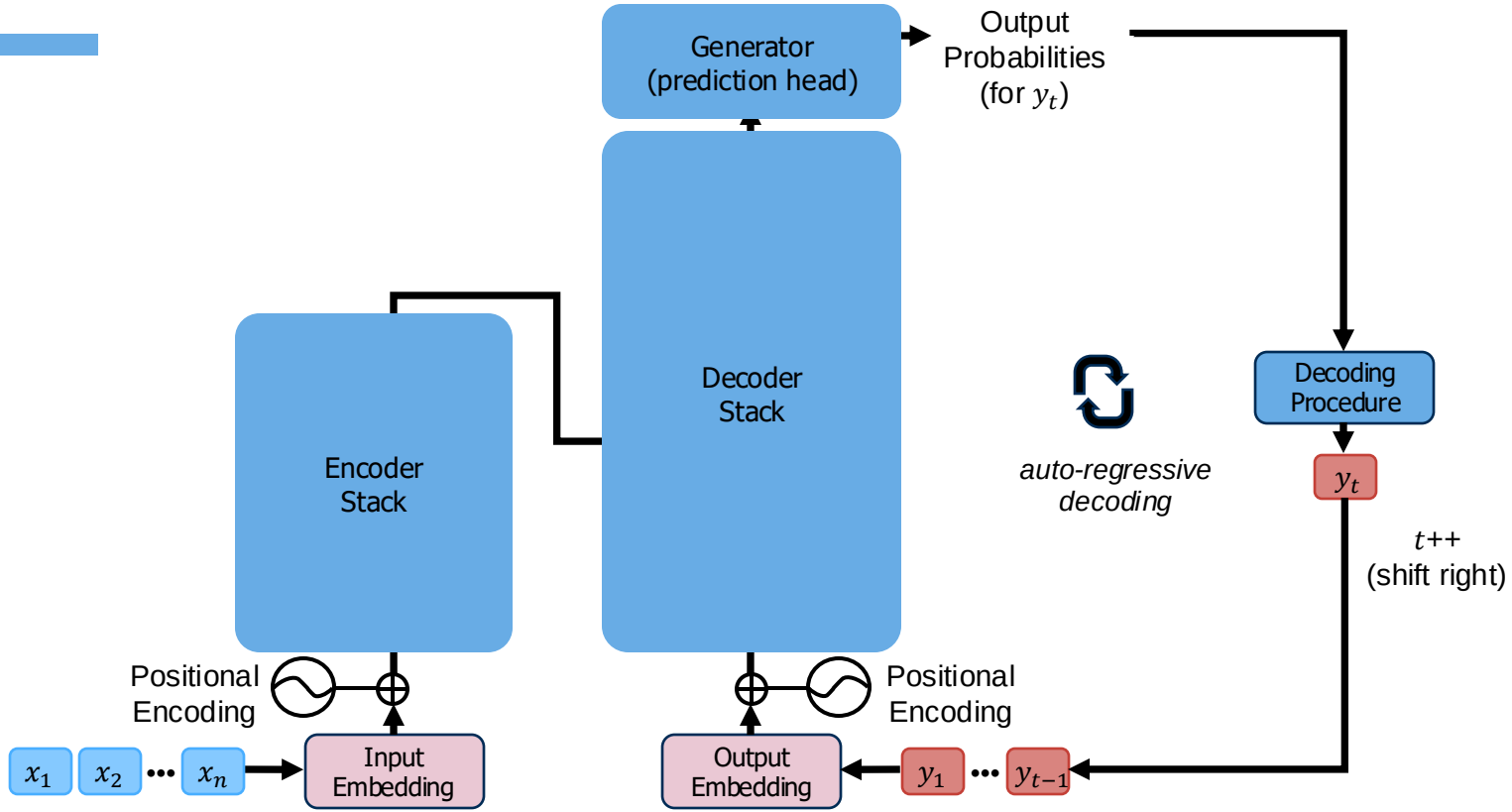


Encoder-decoder models

- Transformer is two blocks
- Encoder = read or encode the input,
  - Architecture is as we've seen
- Decoder = generate or decode the output
  - Architecture is identical to the encoder but we give it the ability to also attend to the input

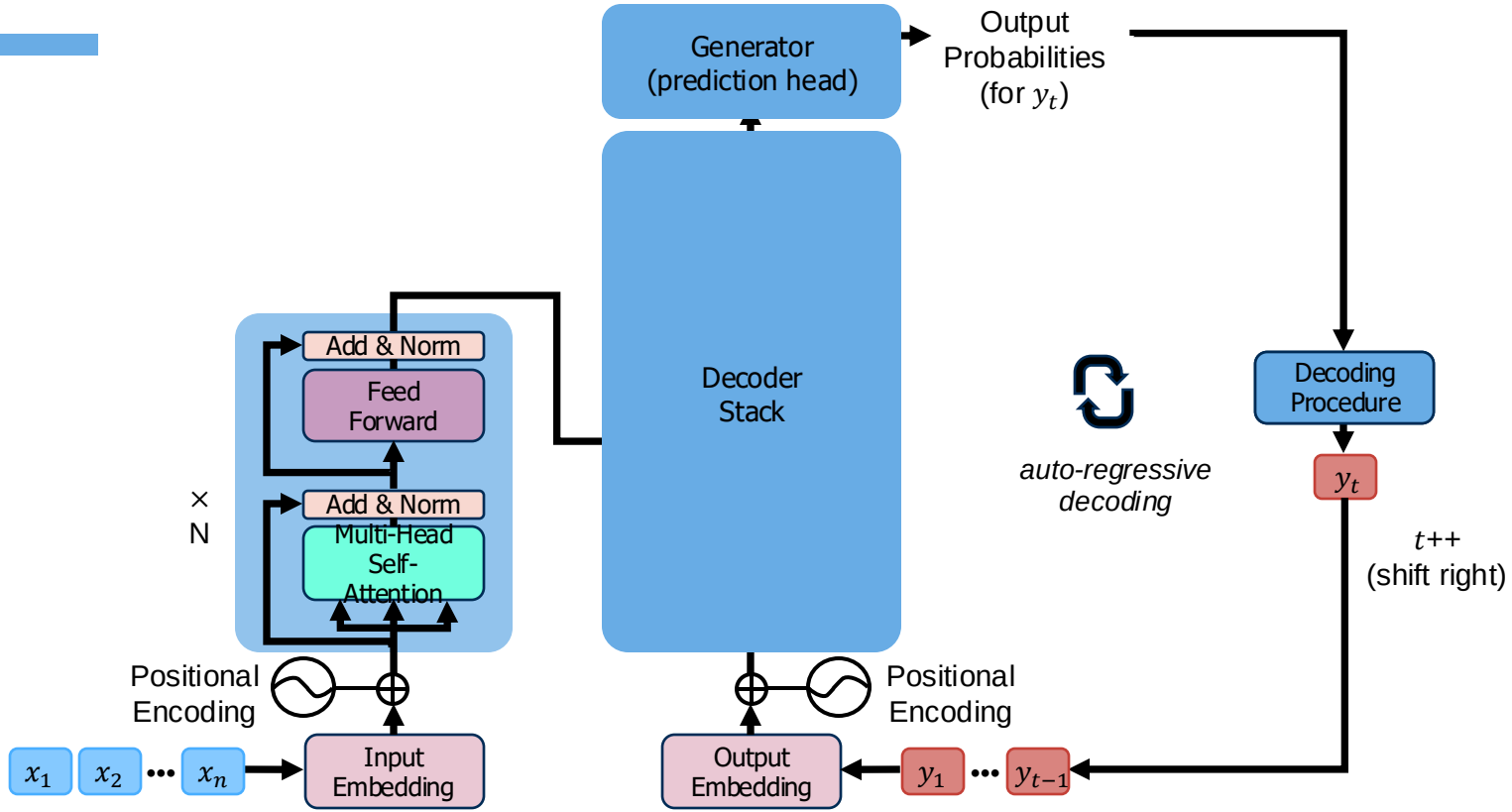


Transformer [Vaswani et al. 2017]



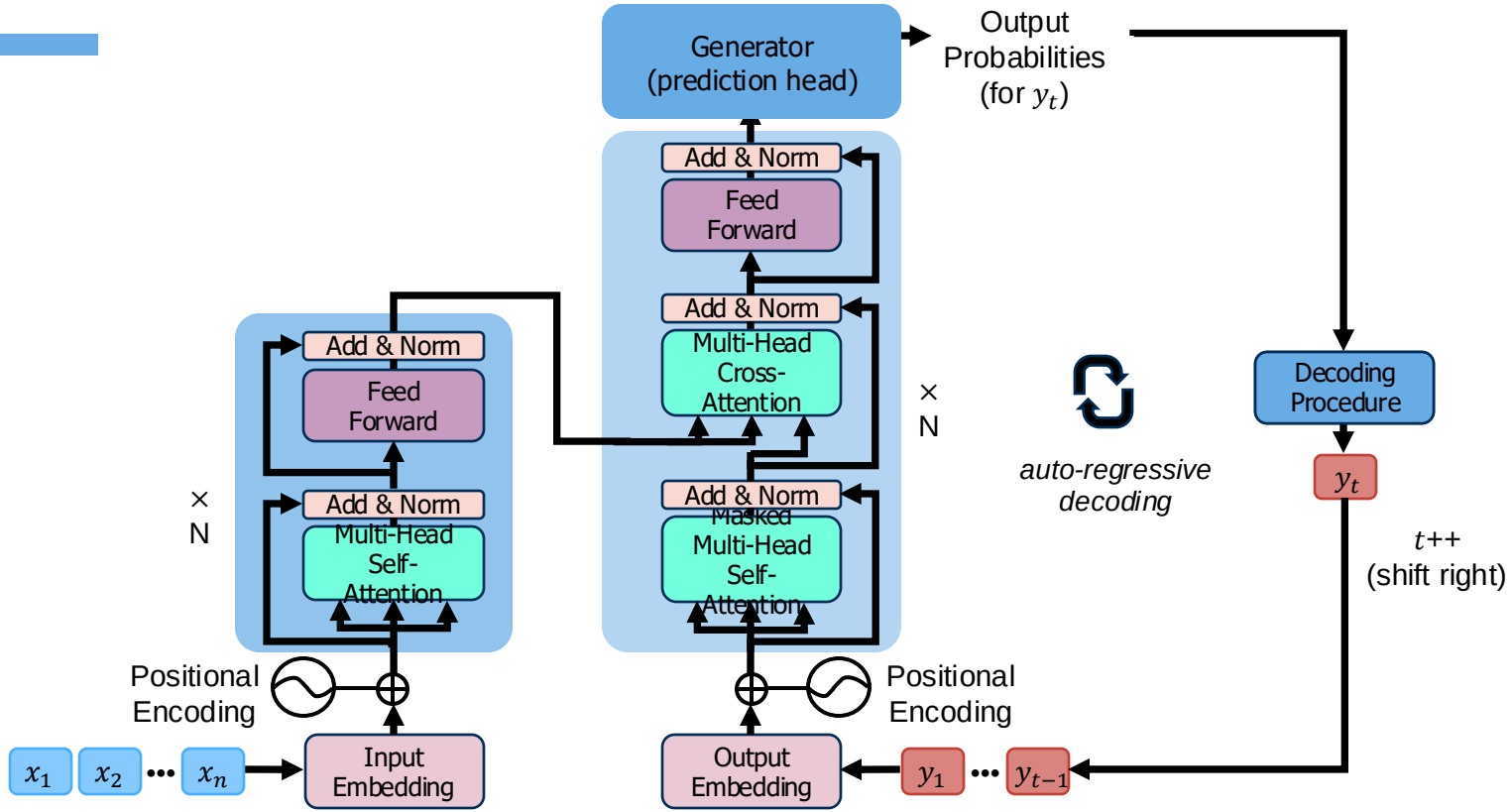
[Slide credit: CS886 at UWaterloo]

Transformer [Vaswani et al. 2017]



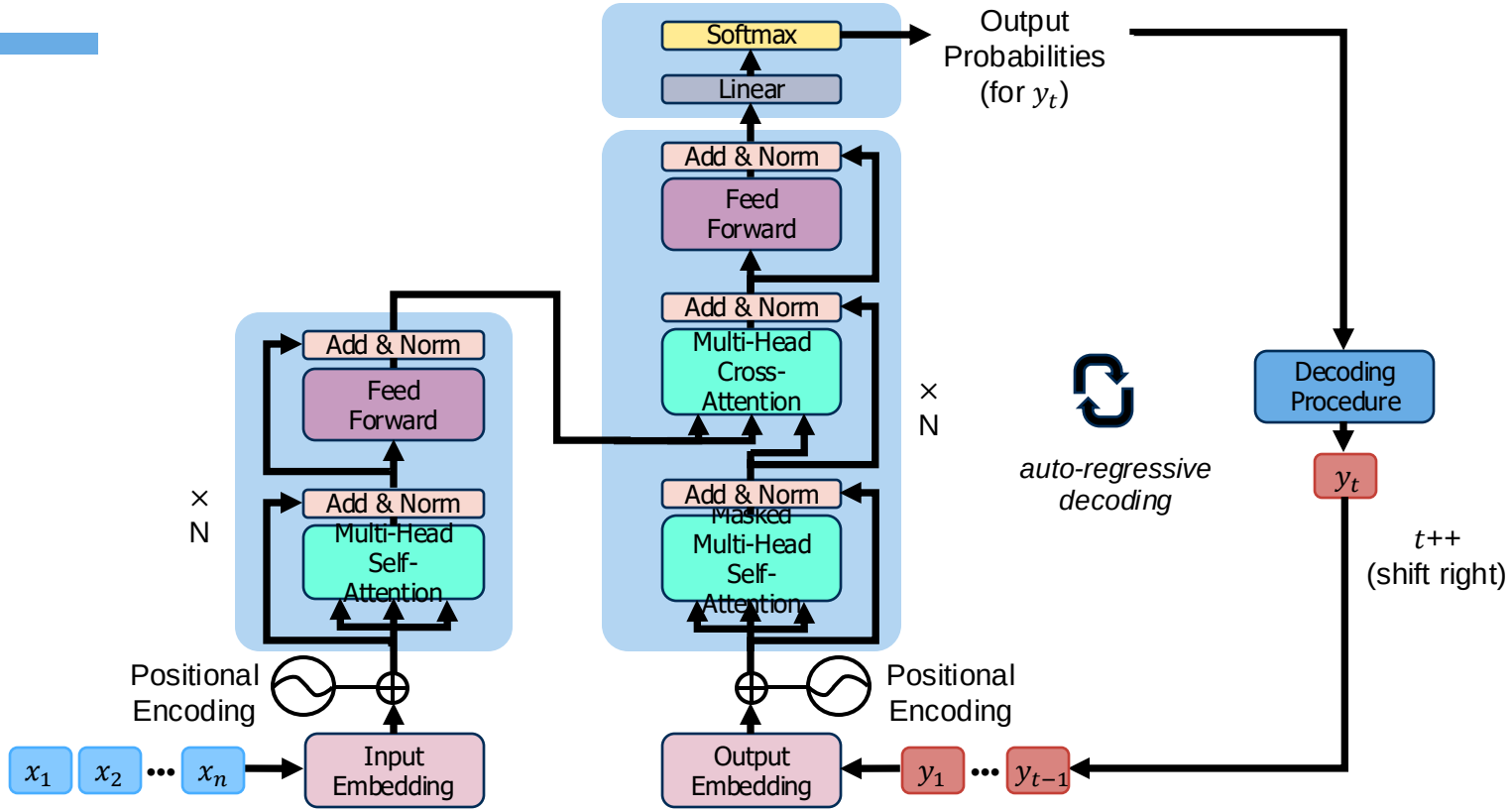
[Slide credit: CS886 at UWaterloo]

Transformer [Vaswani et al. 2017]



[Slide credit: CS886 at UWaterloo]

Transformer [Vaswani et al. 2017]

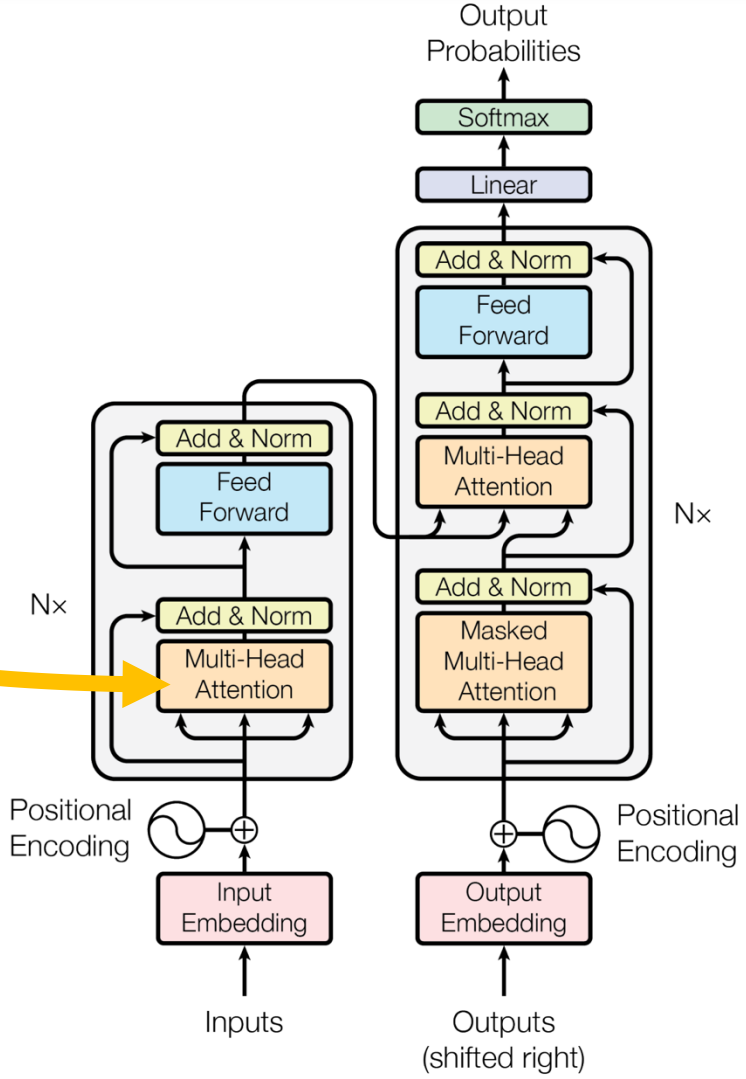


[Slide credit: CS886 at UWaterloo]

Transformer [Vaswani et al. 2017]

- Computation of **encoder** attends to both sides.

Encoder Self-Attention

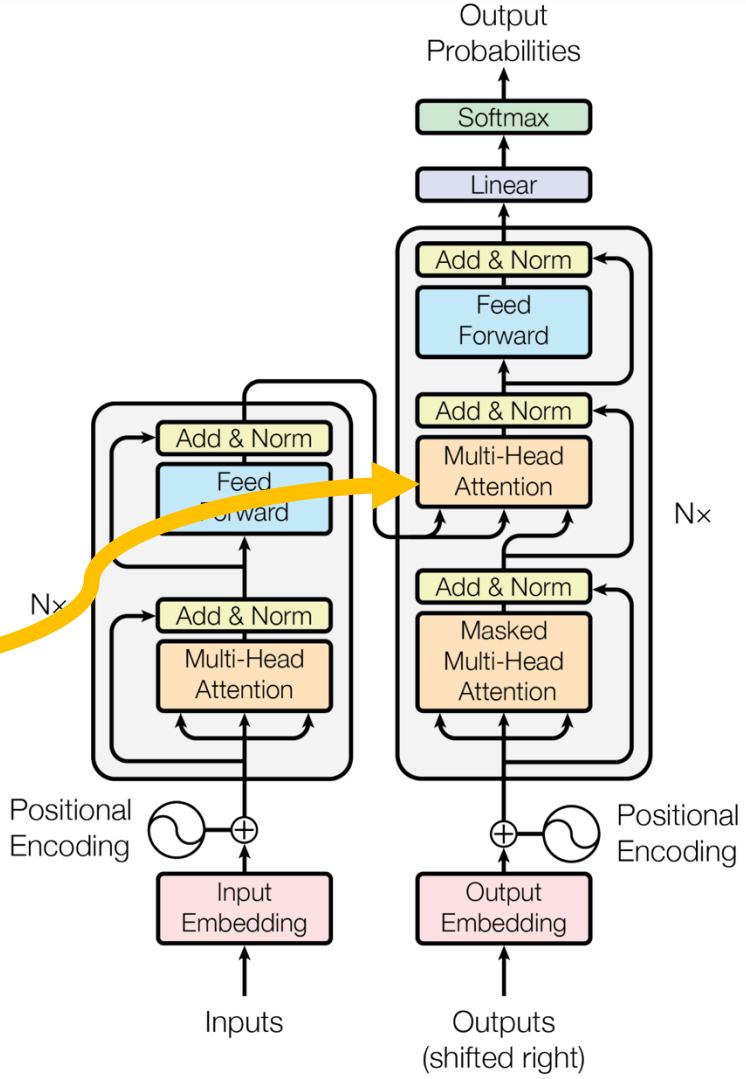


[Attention Is All You Need, Vaswani et al. 2017]

Transformer [Vaswani et al. 2017]

- At any step of **decoder**, it attends to previous computation of **encoder**

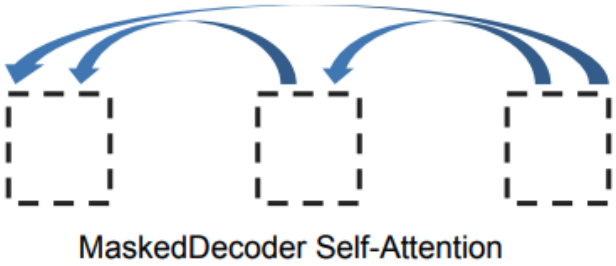
Encoder-Decoder Attention

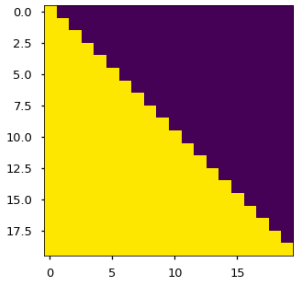


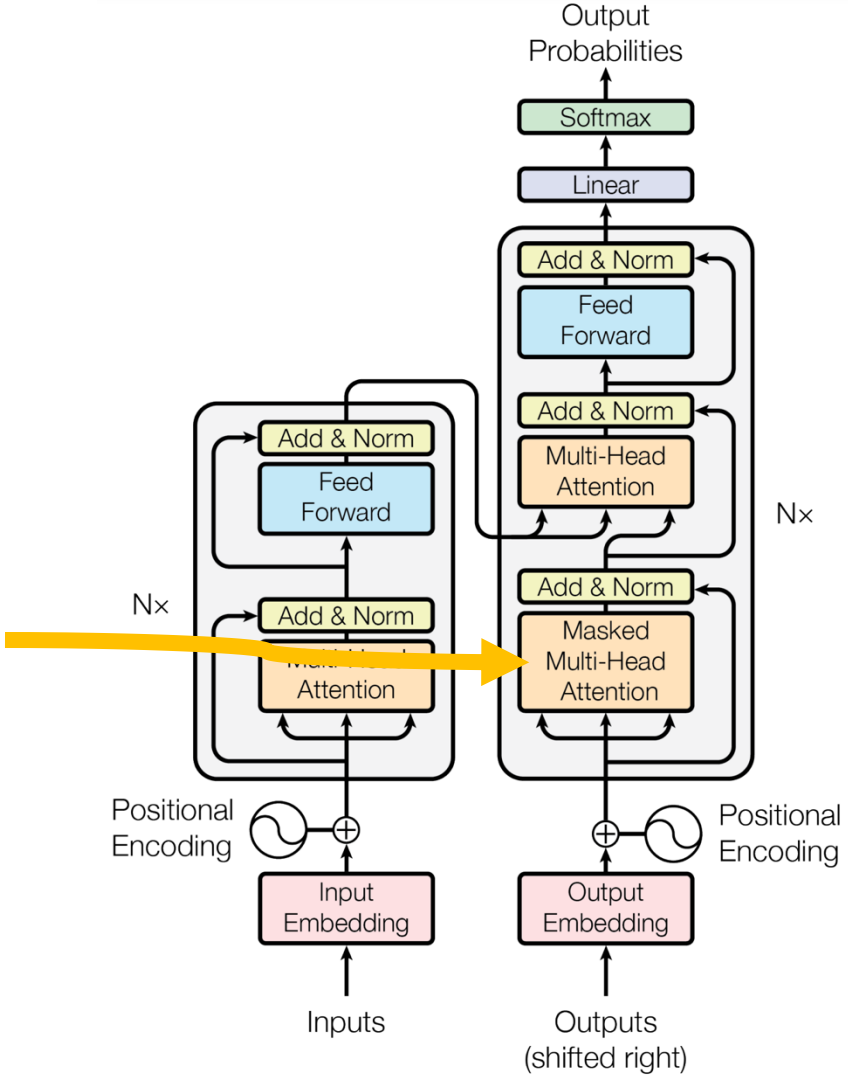
[Attention Is All You Need, Vaswani et al. 2017]

Transformer [Vaswani et al. 2017]

- At any step of **decoder**, it attends to previous computation of **encoder** as well as **decoder's** own generations





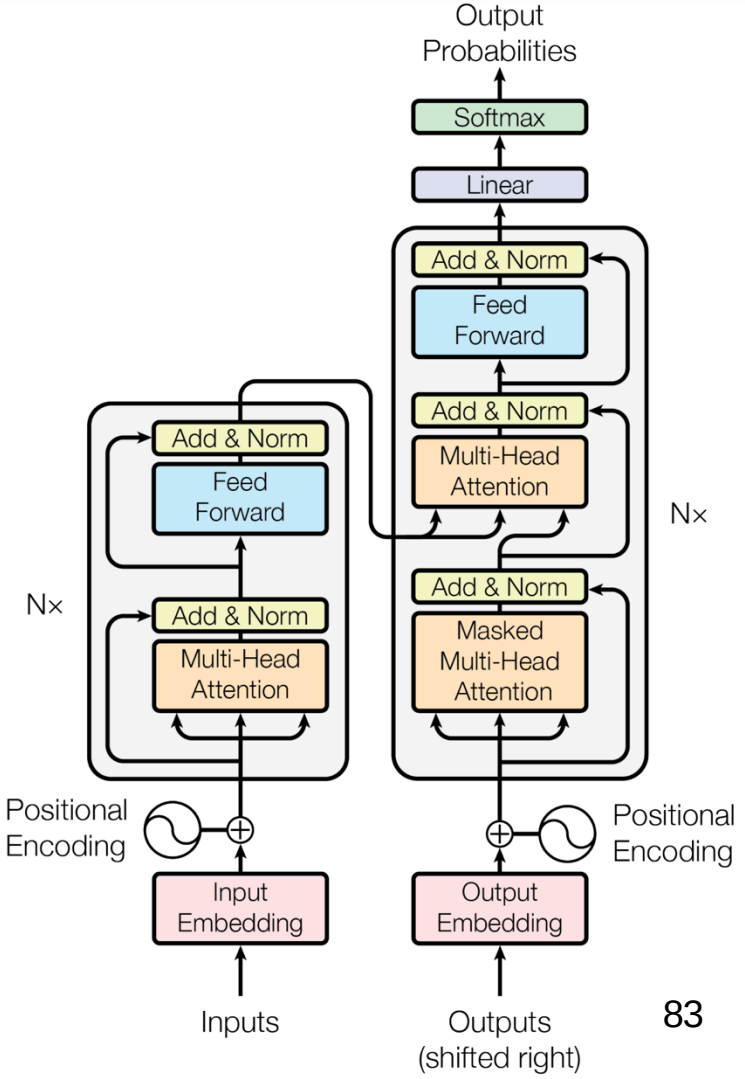


[Attention Is All You Need, Vaswani et al. 2017]

Transformer [Vaswani et al. 2017]

- At any step of **decoder**, it attends to previous computation of **encoder** as well as **decoder's** own generations

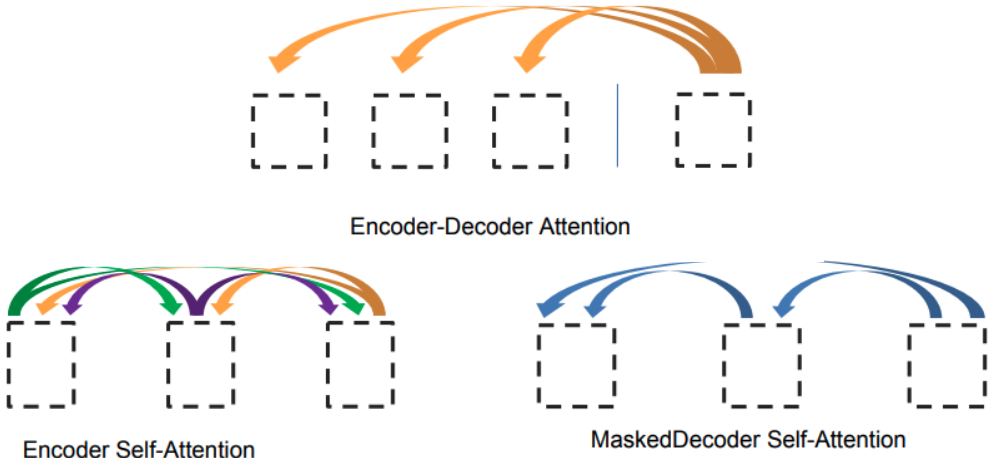
- At any step of decoder, **re-use** previous computation of encoder.
- Computation of decoder is **linear**, instead of quadratic.



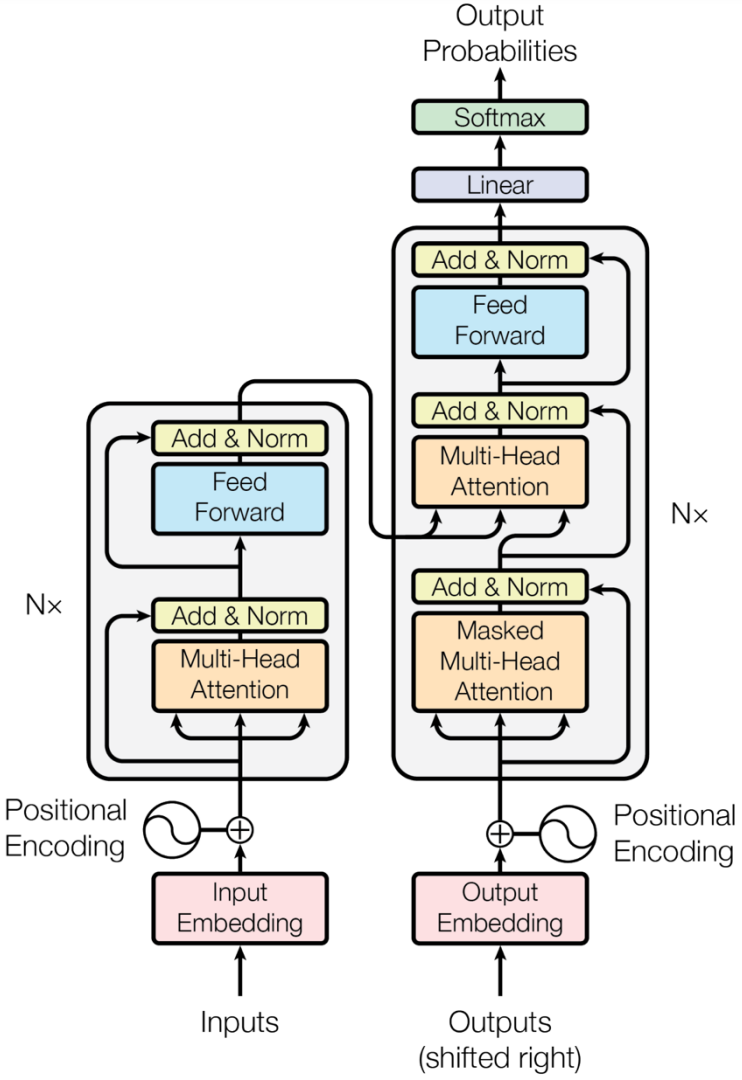
[Attention Is All You Need, Vaswani et al. 2017]

Recap: Transformer

- Yaaay we know Transformers now!
- An encoder-decoder architecture
- 3 forms of attention



[Attention Is All You Need, Vaswani et al. 2017]



Quiz: Enc-Dec Cost

- Source data (large!):
  - The references for a Wikipedia article.
  - Web search using article section titles, ~ 10 web pages per query.

- For a passage of length N and a summary of length M, the complexity of the attention is:
  - $O(N) + O(M)$
  - $O(N) + O(M) + O(NM)$
  - $O(N^2) + O(M^2) + O(NM)$
  - $O(N^2) + O(M^2)$

No, self attention is all-to-all and so quadratic.

[Slide: John Canny]

Quiz: Enc-Dec Cost

- Source data (large!):
  - The references for a Wikipedia article.
  - Web search using article section titles, ~ 10 web pages per query.

- For a passage of length N and a summary of length M, the complexity of the attention is:
  - $O(N) + O(M)$
  - $O(N) + O(M) + O(NM)$
  - $O(N^2) + O(M^2) + O(NM)$
  - $O(N^2) + O(M^2)$

No, self attention is all-to-all and so quadratic in M and N.

[Slide: John Canny]

Quiz: Enc-Dec Cost

- Source data (large!):
  - The references for a Wikipedia article.
  - Web search using article section titles, ~ 10 web pages per query.

- For a passage of length N and a summary of length M, the complexity of the attention is:
  - $O(N) + O(M)$
  - $O(N) + O(M) + O(NM)$
  - $O(N^2) + O(M^2) + O(NM)$
  - $O(N^2) + O(M^2)$

No, self attention is all-to-all and so quadratic in M and N.

[Slide: John Canny]

Quiz: Enc-Dec Cost

- Source data (large!):
  - The references for a Wikipedia article.
  - Web search using article section titles, ~ 10 web pages per query.

- For a passage of length N and a summary of length M, the complexity of the attention is:
  - $O(N) + O(M)$
  - $O(N) + O(M) + O(NM)$
  - $O(N^2) + O(M^2) + O(NM)$
  - $O(N^2) + O(M^2)$

No, cross attention is missing.

[Slide: John Canny]

Quiz: Enc-Dec Cost

- Source data (large!):
  - The references for a Wikipedia article.
  - Web search using article section titles, ~ 10 web pages per query.

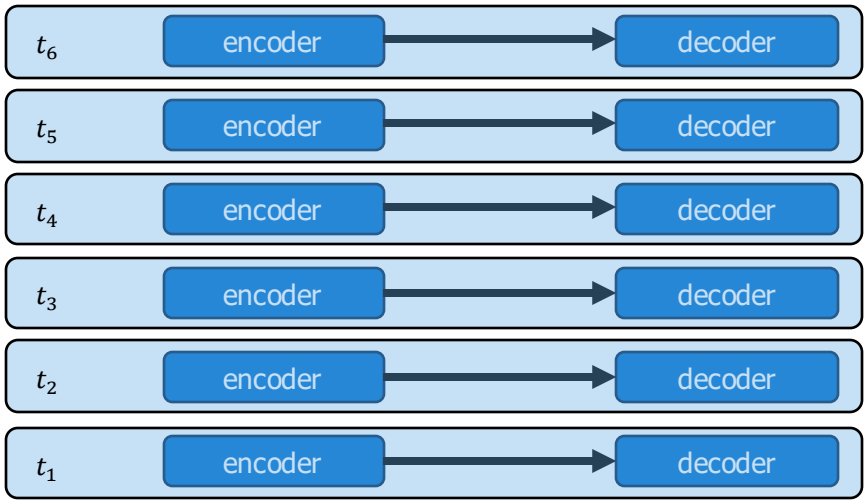
- For a passage of length N and a summary of length M, the complexity of the attention is:
  - $O(N) + O(M)$
  - $O(N) + O(M) + O(NM)$
  - $O(N^2) + O(M^2) + O(NM)$
  - $O(N^2) + O(M^2)$

Yes. The three terms are respectively the Encoder self-attention, Decoder self-attention, and Cross attention.

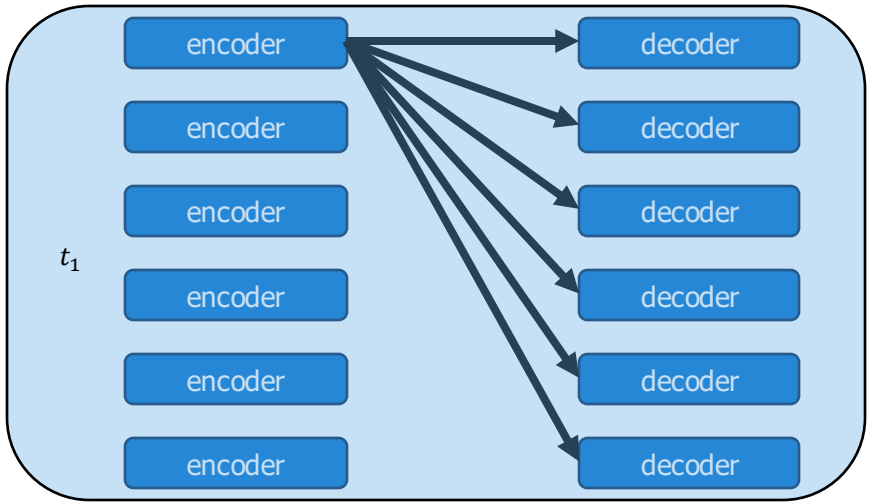
[Slide: John Canny]

Quiz: Enc-Dec Connections

- Which best represents encoder-decoder connections?



Incorrect



Correct

[Slide credit: CS886 at UWaterloo]

Writing our own Transformer

Clone Helper Function

- Create N copies of pytorch nn.Module
- The Transformer's structure contains a lot of design repetition (like VGG)
- Remember these clones shouldn't share parameters (for the most part)

```
def clones(module, N):
    "Produce N identical layers."
    return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
```


[Slide credit: CS886 at UWaterloo]

Create Embedding

- Create vector representation of sequence vocabulary
- nn.Embedding creates a lookup table to map sequence vocabulary to unique vectors

```
class Embeddings(nn.Module):
    def __init__(self, d_model, vocab):
        super(Embeddings, self).__init__()
        self.lut = nn.Embedding(vocab, d_model)
        self.d_model = d_model

    def forward(self, x):
        return self.lut(x) * math.sqrt(self.d_model)
```


[Slide credit: CS886 at UWaterloo]

Positional Encoding

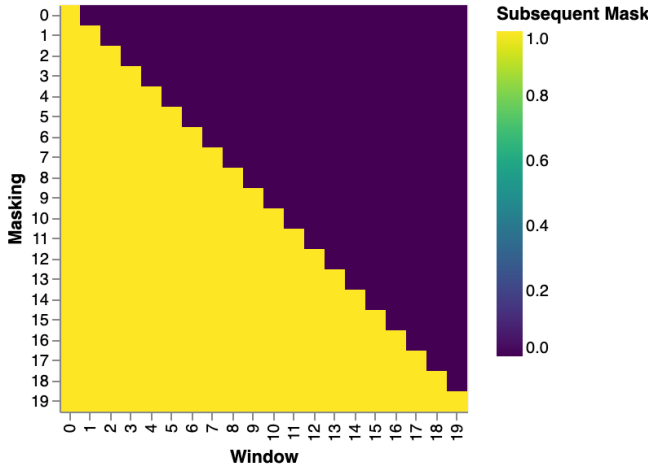
- Add information about an element's position in a sequence to its representation
- Element wise addition of sinusoidal encoding


```python
class PositionalEncoding(nn.Module):
    "Implement the PE function."

    def __init__(self, d_model, dropout, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        # Compute the positional encodings once in log space.
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len).unsqueeze(1)
        div_term = torch.exp(
            torch.arange(0, d_model, 2) * -(math.log(10000.0) / d_model)
        )
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)
        self.register_buffer("pe", pe)

    def forward(self, x):
        x = x + self.pe[:, : x.size(1)].requires_grad_(False)
        return self.dropout(x)
```


[Slide credit: CS886 at UWaterloo]

Attention block

```
def attention(query, key, value, mask=None, dropout=None):
    "Compute 'Scaled Dot Product Attention'"
    d_k = query.size(-1)
    scores = torch.matmul(query, key.transpose(-2, -1)) / math.sqrt(d_k)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, -1e9)
    p_attn = scores.softmax(dim=-1)
    if dropout is not None:
        p_attn = dropout(p_attn)
    return torch.matmul(p_attn, value), p_attn
```



$-1e9$ is a large negative number, which leads to softmax(-1e9) ≈ 0

Multi-Head Attention

```python
class MultiHeadedAttention(nn.Module):
    def __init__(self, h, d_model, dropout=0.1):
        "Take in model size and number of heads."
        super(MultiHeadedAttention, self).__init__()
        assert d_model % h == 0
        # We assume d_v always equals d_k
        self.d_k = d_model // h
        self.h = h
        self.linears = clones(nn.Linear(d_model, d_model), 4)
        self.attn = None
        self.dropout = nn.Dropout(p=dropout)
```

```python
def forward(self, query, key, value, mask=None):
    "Implements Figure 2"
    if mask is not None:
        # Same mask applied to all h heads.
        mask = mask.unsqueeze(1)
    nbatches = query.size(0)

    # 1) Do all the linear projections in batch from d_model => h x d_k
    query, key, value = [
        lin(x).view(nbatches, -1, self.h, self.d_k).transpose(1, 2)
        for lin, x in zip(self.linears, (query, key, value))
    ]

    # 2) Apply attention on all the projected vectors in batch.
    x, self.attn = attention(
        query, key, value, mask=mask, dropout=self.dropout
    )

    # 3) "Concat" using a view and apply a final linear.
    x = (
        x.transpose(1, 2)
        .contiguous()
        .view(nbatches, -1, self.h * self.d_k)
    )
    del query
    del key
    del value
    return self.linears[-1](x)
```


[Slide credit: CS886 at UWaterloo]

FeedForward Layer

```
class PositionwiseFeedForward(nn.Module):
    "Implements FFN equation."

    def __init__(self, d_model, d_ff, dropout=0.1):
        super(PositionwiseFeedForward, self).__init__()
        self.w_1 = nn.Linear(d_model, d_ff)
        self.w_2 = nn.Linear(d_ff, d_model)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        return self.w_2(self.dropout(self.w_1(x).relu()))
```


[Slide credit: CS886 at UWaterloo]

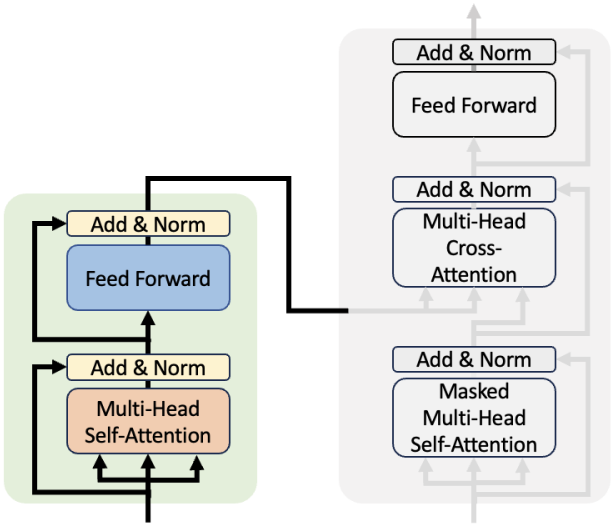
Sublayer Connections

```
class SublayerConnection(nn.Module):
    """
    A residual connection followed by a layer norm.
    Note for code simplicity the norm is first as opposed to last.
    """

    def __init__(self, size, dropout):
        super(SublayerConnection, self).__init__()
        self.norm = LayerNorm(size)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, sublayer):
        "Apply residual connection to any sublayer with the same size."
        return x + self.dropout(sublayer(self.norm(x)))
```


[Slide credit: CS886 at UWaterloo]

Encoder Layer

```
class EncoderLayer(nn.Module):
    "Encoder is made up of self-attn and feed forward (defined below)"

    def __init__(self, size, self_attn, feed_forward, dropout):
        super(EncoderLayer, self).__init__()
        self.self_attn = self_attn
        self.feed_forward = feed_forward
        self.sublayer = clones(SublayerConnection(size, dropout), 2)
        self.size = size

    def forward(self, x, mask):
        "Follow Figure 1 (left) for connections."
        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, mask))
        return self.sublayer[1](x, self.feed_forward)
```

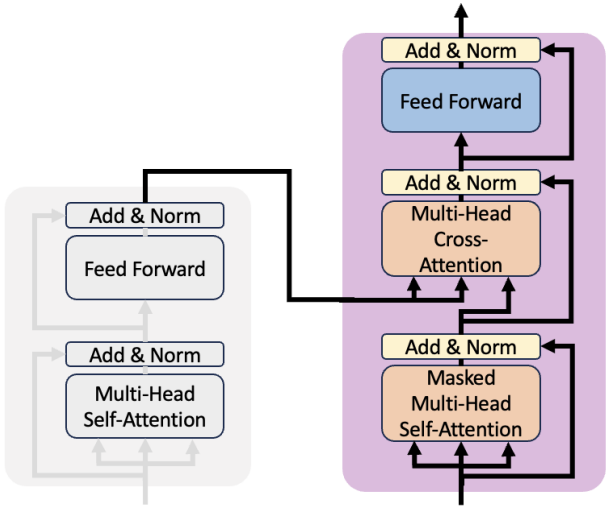


[Slide credit: CS886 at UWaterloo]

Decoder Layer

- Same as encoder layers other than:
  - the additional multi-head attention block to preform cross-attention with the output representation from the encoder

```
class DecoderLayer(nn.Module):
    "Decoder is made of self-attn, src-attn, and feed forward (defined below)"

    def __init__(self, size, self_attn, src_attn, feed_forward, dropout):
        super(DecoderLayer, self).__init__()
        self.size = size
        self.self_attn = self_attn
        self.src_attn = src_attn
        self.feed_forward = feed_forward
        self.sublayer = clones(SublayerConnection(size, dropout), 3)

    def forward(self, x, memory, src_mask, tgt_mask):
        "Follow Figure 1 (right) for connections."
        m = memory
        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, tgt_mask))
        x = self.sublayer[1](x, lambda x: self.src_attn(x, m, m, src_mask))
        return self.sublayer[2](x, self.feed_forward)
```

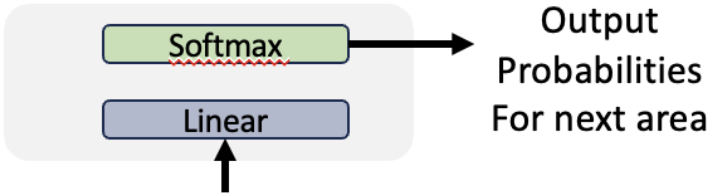


[Slide credit: CS886 at UWaterloo]

The Prediction Head

- A final linear mapping
- Apply softmax to convert logits to probabilities

```
class Generator(nn.Module):
    "Define standard linear + softmax generation step."

    def __init__(self, d_model, vocab):
        super(Generator, self).__init__()
        self.proj = nn.Linear(d_model, vocab)

    def forward(self, x):
        return log_softmax(self.proj(x), dim=-1)
```



[Slide credit: CS886 at UWaterloo]

Build each block

```
class Encoder(nn.Module):
    "Core encoder is a stack of N layers"

    def __init__(self, layer, N):
        super(Encoder, self).__init__()
        self.layers = clones(layer, N)
        self.norm = LayerNorm(layer.size)

    def forward(self, x, mask):
        "Pass the input (and mask) through each layer in turn."
        for layer in self.layers:
            x = layer(x, mask)
        return self.norm(x)
```

```
class Decoder(nn.Module):
    "Generic N layer decoder with masking."

    def __init__(self, layer, N):
        super(Decoder, self).__init__()
        self.layers = clones(layer, N)
        self.norm = LayerNorm(layer.size)

    def forward(self, x, memory, src_mask, tgt_mask):
        for layer in self.layers:
            x = layer(x, memory, src_mask, tgt_mask)
        return self.norm(x)
```


[Slide credit: CS886 at UWaterloo]

Putting it Together

```
class EncoderDecoder(nn.Module):
    """
    A standard Encoder-Decoder architecture. Base for this and many
    other models.
    """

    def __init__(self, encoder, decoder, src_embed, tgt_embed, generator):
        super(EncoderDecoder, self).__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.src_embed = src_embed
        self.tgt_embed = tgt_embed
        self.generator = generator

    def forward(self, src, tgt, src_mask, tgt_mask):
        "Take in and process masked src and target sequences."
        return self.decode(self.encode(src, src_mask), src_mask, tgt, tgt_mask)

    def encode(self, src, src_mask):
        return self.encoder(self.src_embed(src), src_mask)

    def decode(self, memory, src_mask, tgt, tgt_mask):
        return self.decoder(self.tgt_embed(tgt), memory, src_mask, tgt_mask)
```

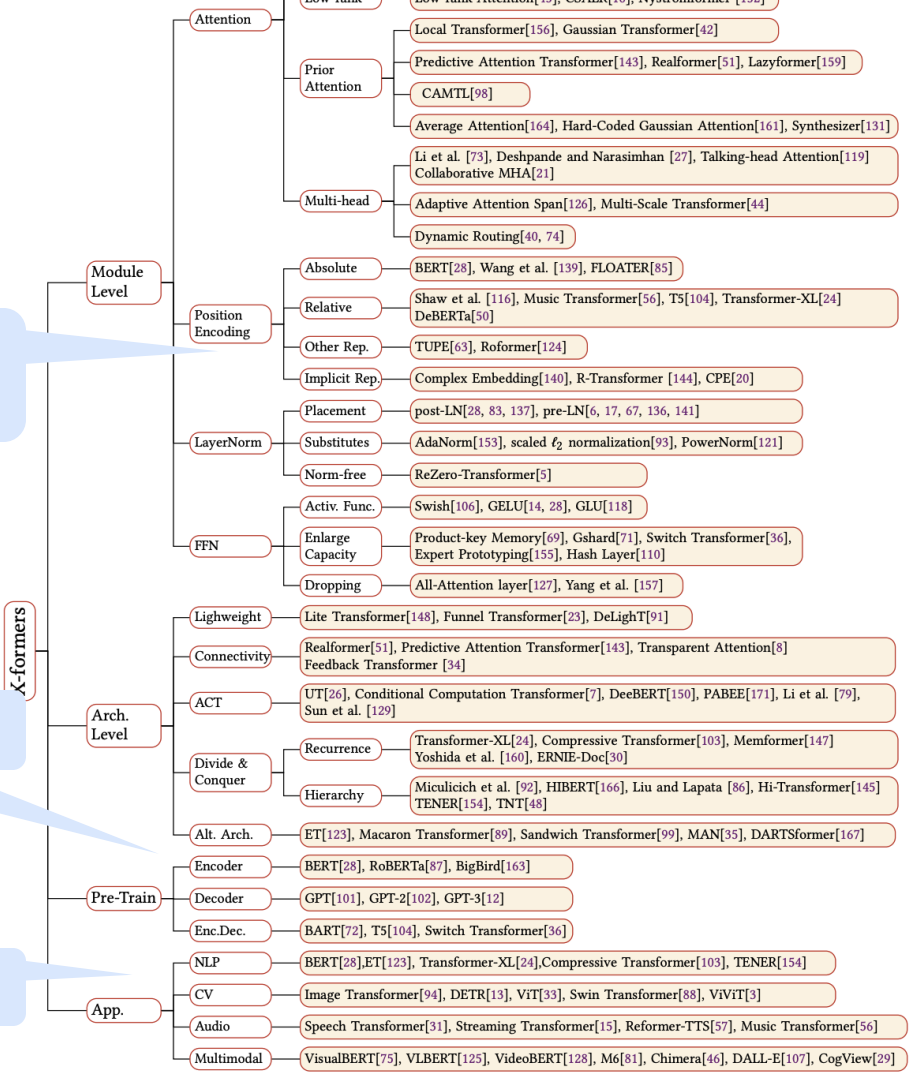
[Slide credit: CS886 at UWaterloo]

Initialize the model

```
def make_model(
    src_vocab, tgt_vocab, N=6, d_model=512, d_ff=2048, h=8, dropout=0.1
):
    "Helper: Construct a model from hyperparameters."
    c = copy.deepcopy
    attn = MultiHeadedAttention(h, d_model)
    ff = PositionwiseFeedForward(d_model, d_ff, dropout)
    position = PositionalEncoding(d_model, dropout)
    model = EncoderDecoder(
        Encoder(EncoderLayer(d_model, c(attn), c(ff), dropout), N),
        Decoder(DecoderLayer(d_model, c(attn), c(attn), c(ff), dropout), N),
        nn.Sequential(Embeddings(d_model, src_vocab), c(position)),
        nn.Sequential(Embeddings(d_model, tgt_vocab), c(position)),
        Generator(d_model, tgt_vocab),
    )

    # This was important from their code.
    # Initialize parameters with Glorot / fan_avg.
    for p in model.parameters():
        if p.dim() > 1:
            nn.init.xavier_uniform_(p)
    return model
```


[Slide credit: CS886 at UWaterloo]

After Transformer ...



Variants of positional embeddings

Architectural choices

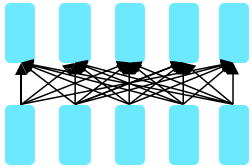
Multi-modal models

We will visit a few of these branches …

But there is a lot that we do **not** cover …

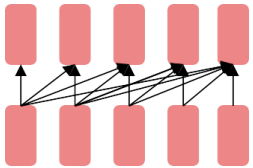
Impact of Transformers

- A building block for a variety of LMs



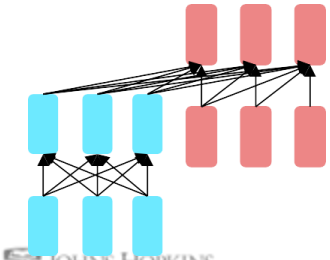
Encoders

- Examples: BERT, RoBERTa, SciBERT.
- Captures bidirectional context. Wait, how do we pretrain them?



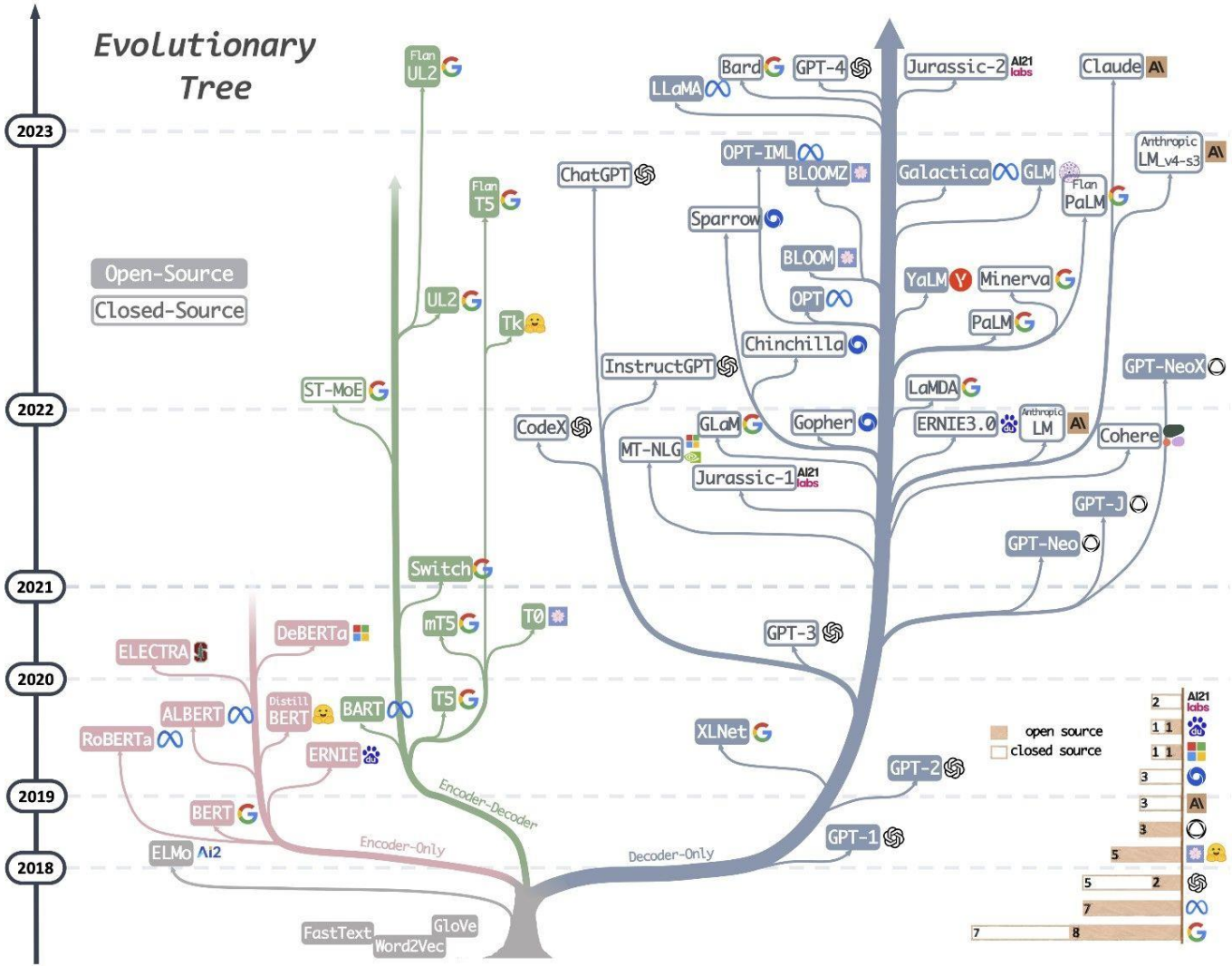
Decoders

- Examples: GPT-2, GPT-3, LaMDA
- Other name: causal or auto-regressive language model
- Nice to generate from; can't condition on future words



Encoder-Decoders

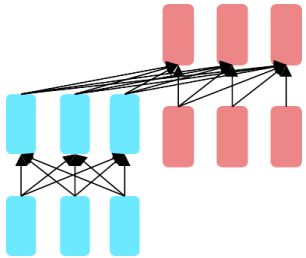
- Examples: Transformer, T5, Meena
- What's the best way to pretrain them?



Yang et al. Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond, 2023

Transformer Language Model Families

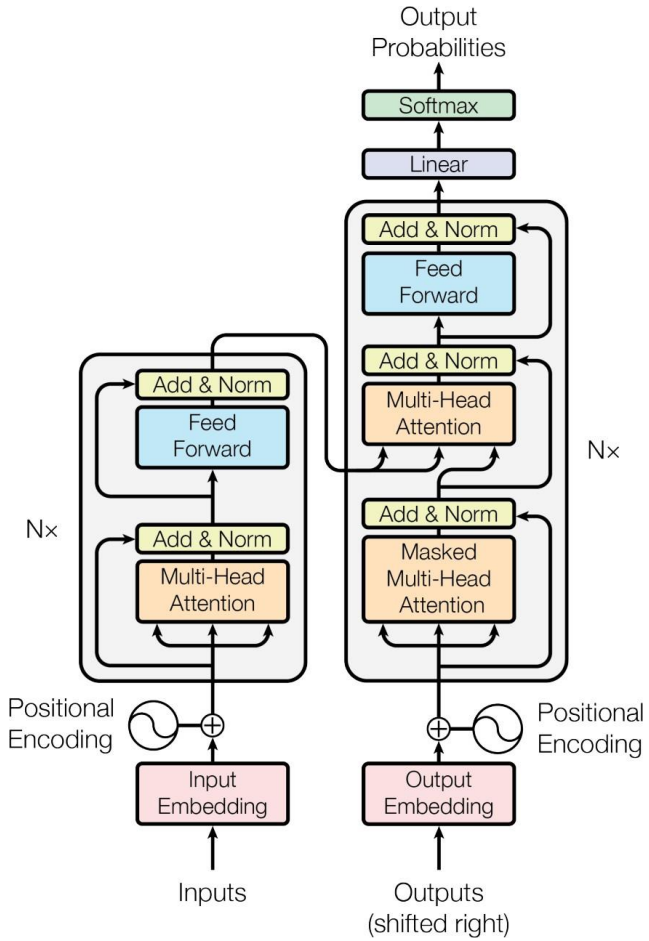
Encoder-Decoder Family of Transformers



Encoder-Decoders

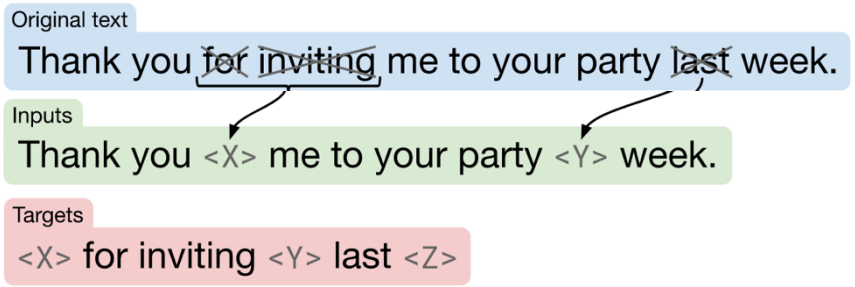
Encoder-decoder Models

- The original transformer architecture was encoder decoder
- Encoder-decoder models are flexible in both generation and classification tasks
- How can we pretrain an encoder-decoder model like BERT to be a good general language pretrained LM?



T5: Text-To-Text Transfer Transformer

- An encoder-decoder architecture
- Pre-training objective:
  corrupt and reconstruct objective



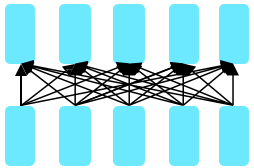
Model	Parameters	No. of layers	d_{model}	d_{ff}	d_{kv}	No. of heads
Small	60M	6	512	2048	64	8
Base	220M	12	768	3072	64	12
Large	770M	24	1024	4096	64	16
3B	3B	24	1024	16384	128	32
11B	11B	24	1024	65536	128	128

- The original paper is an excellent set of in-depth analysis of various parameters of model design. We discuss some of these results in other places.

`https://huggingface.co/t5-base`

Exploring the limits of transfer learning with text-to-text transfer transformers, 2020

Encoder-only Family of Transformers



BERT

Bidirectional Encoder Representations from Transformers

BERT

Bidirectional Encoder Representations from Transformers

Like Bidirectional LSTMs (ELMo), let's look in **both** directions

BERT

Bidirectional Encoder Representations from Transformers

Let's only use Transformer Encoders, no Decoders

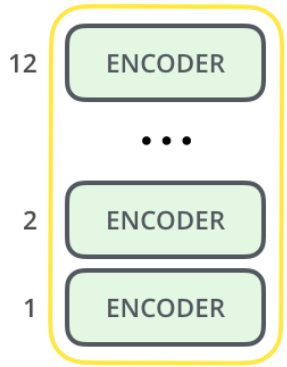
BERT

Bidirectional Encoder Representations from Transformers

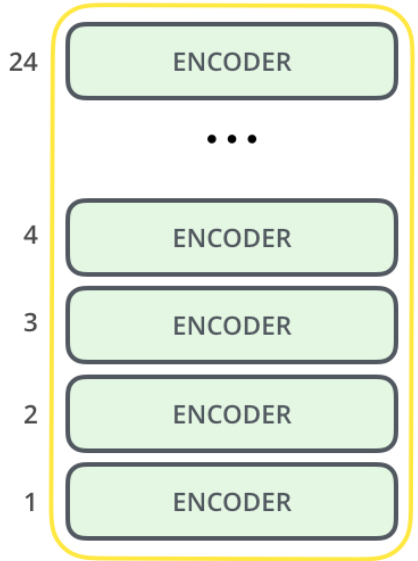
It's a language model that builds rich representations via self-supervised learning (pre-training)

BERT: Architecture

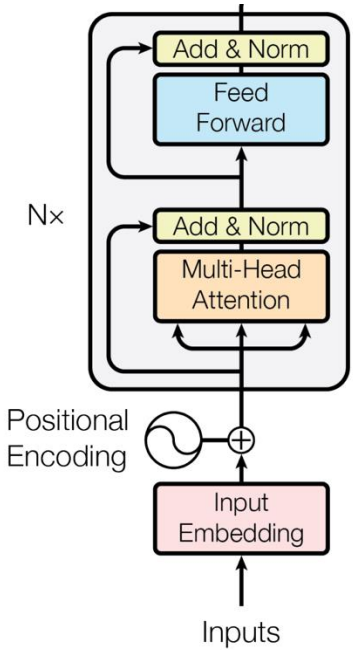
- Stacks of Transformer encoders



$BERT_{BASE}$



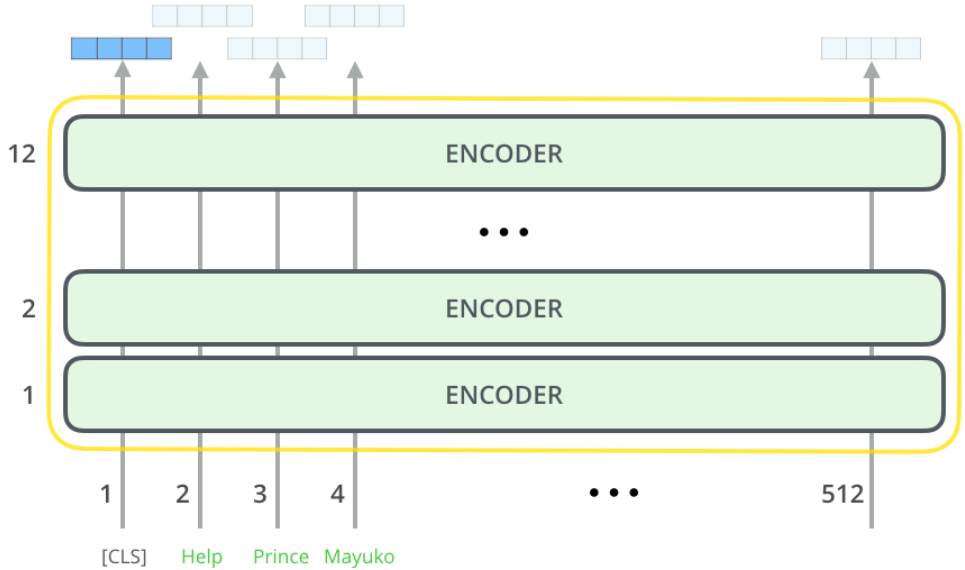
$BERT_{LARGE}$

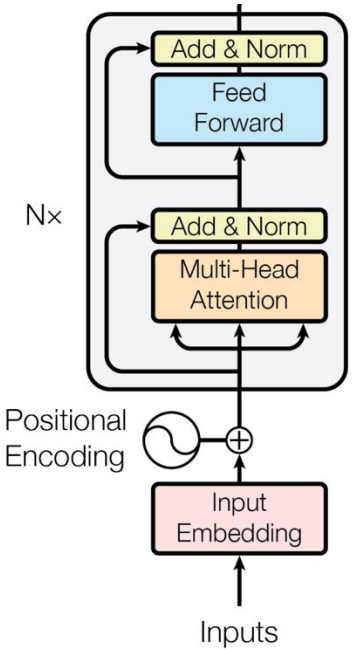


[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

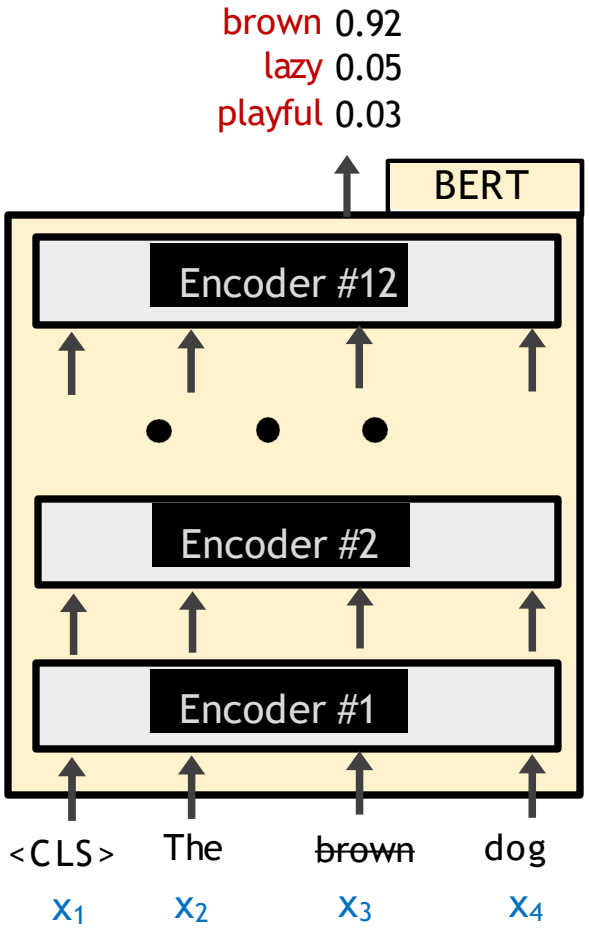
BERT: Architecture

- Model output dimension: 512





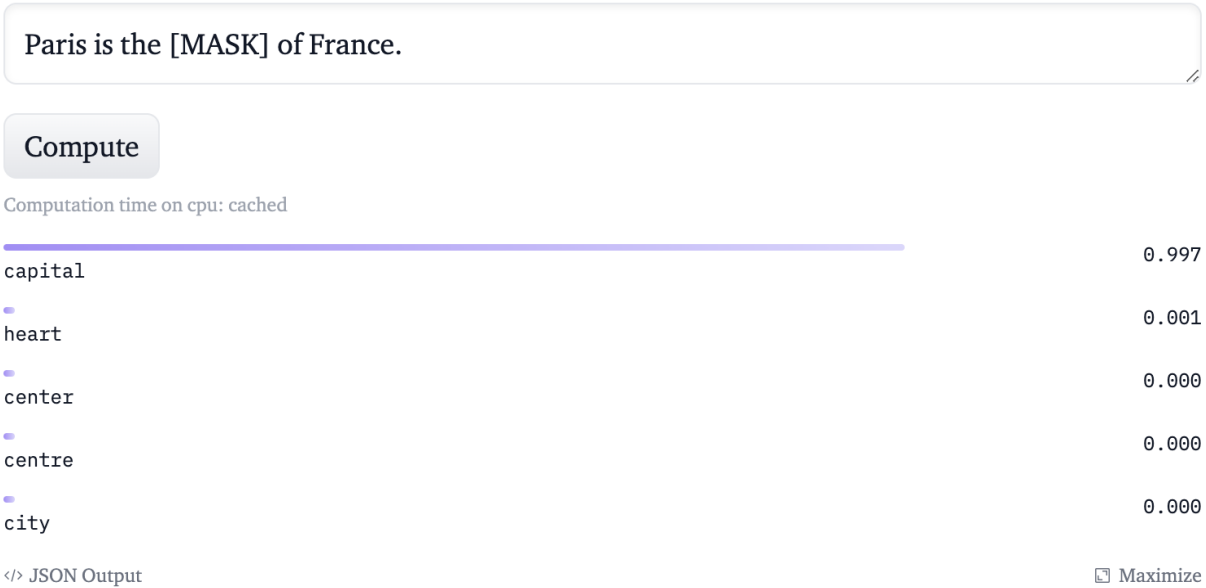
[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]



BERT is trained to uncover masked tokens.

Probing BERT Masked LM

- Masking words forces BERT to use context in both directions to predict the masked word.

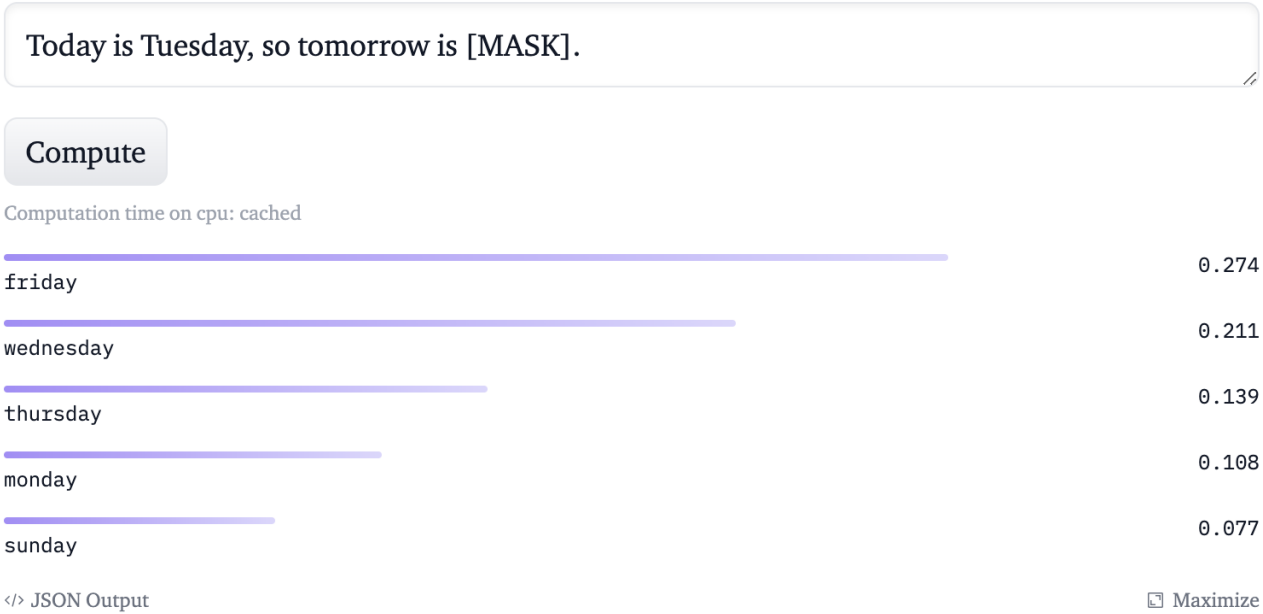


https://huggingface.co/bert-base-uncased

122

Probing BERT Masked LM

- Masking words forces BERT to use context in both directions to predict the masked word.

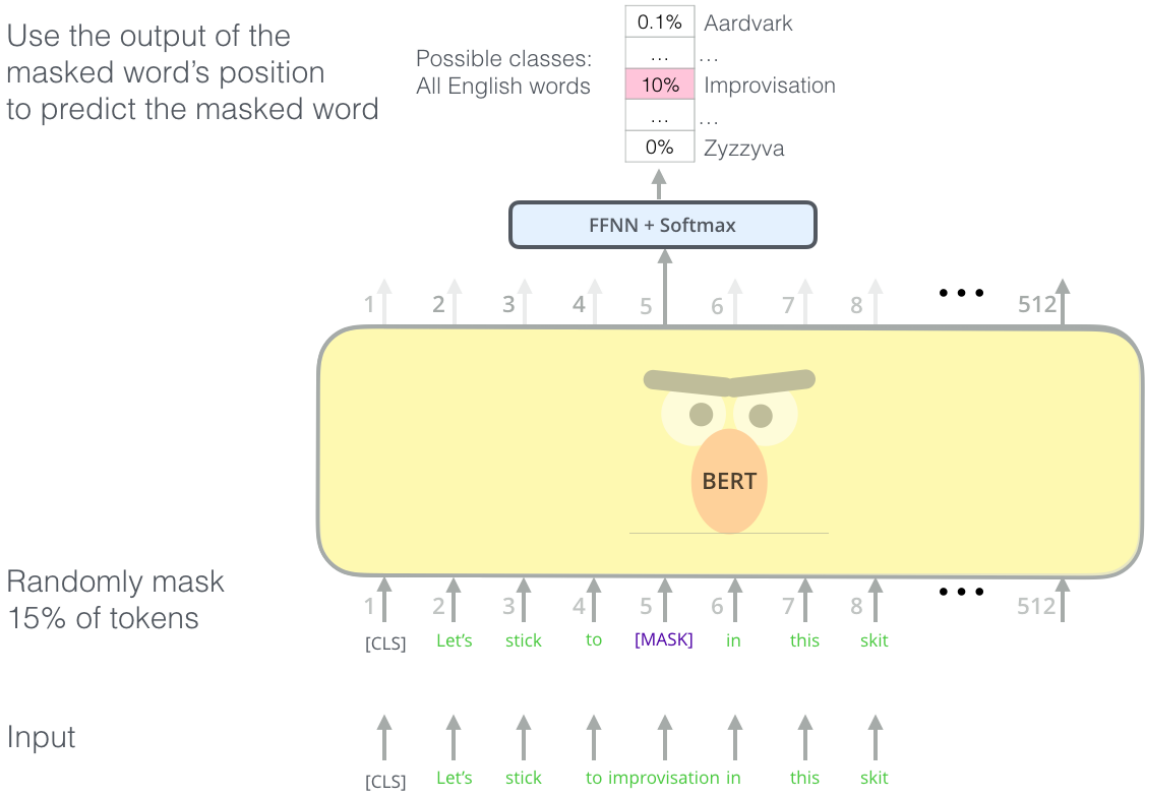


https://huggingface.co/bert-base-uncased

123

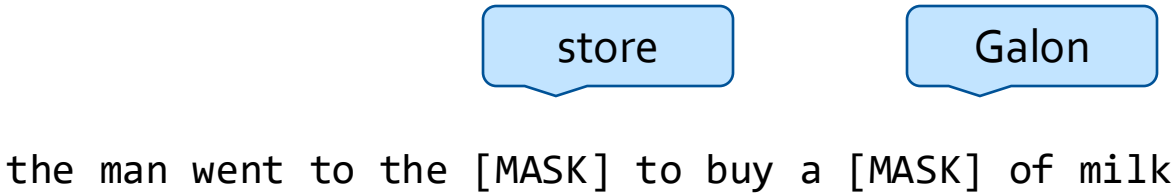
BERT: Pre-training Objective (1): Masked Tokens

- Randomly mask 15% of the tokens and train the model to predict them.



[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

BERT: Pre-training Objective (1): Masked Tokens



- Too little masking: Too expensive to train
- Too much masking: Underdefined
  - (not enough info for the model to recover the masked tokens)

Later work shows that more principled masking (instead of uniformly random) could benefit downstream task performance and result in faster training.

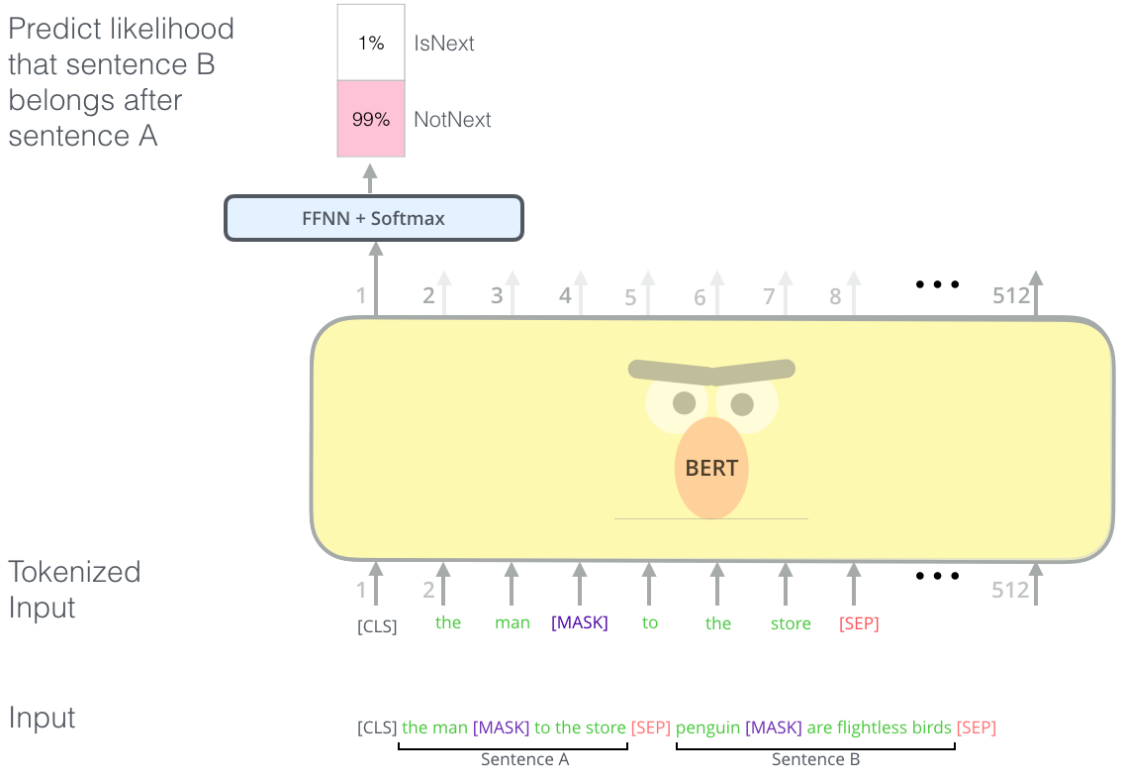
PMI Masking (Levine et al., 2021) https://arxiv.org/pdf/2010.01825.pdf
SpanBERT (Joshi et al., 2020) https://arxiv.org/pdf/1907.10529.pdf

[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

BERT: Pre-training Objective (2): Sentence Ordering

- Predict sentence ordering

- 50% correct ordering, and 50% random incorrect ones



[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

BERT Pre-training Objective (2): Sentence Ordering

- Learn relationships between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

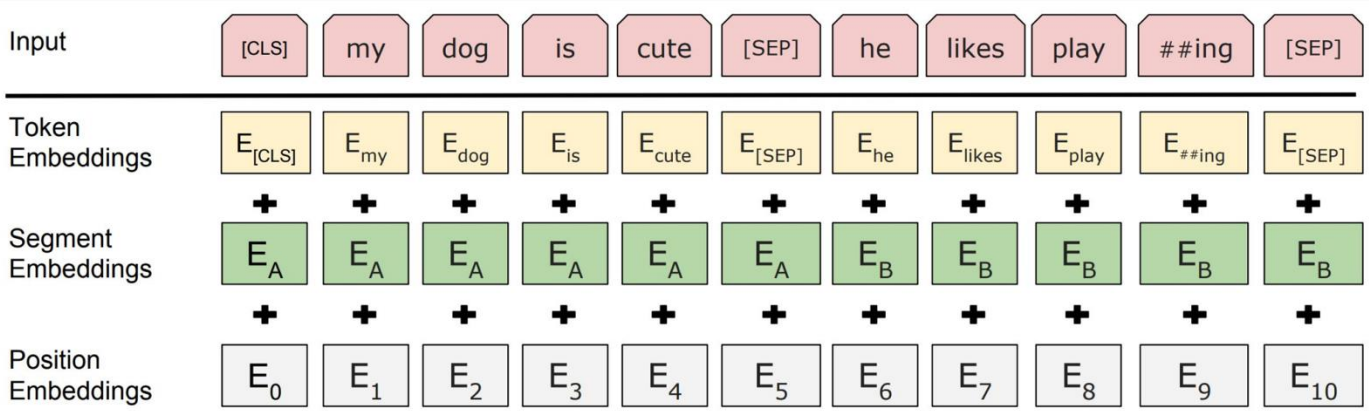
Sentence A = `The man went to the store.`
Sentence B = `He bought a gallon of milk.`
Label = `IsNextSentence`

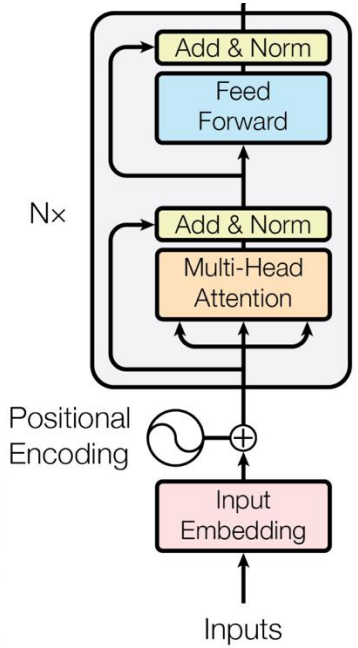
Sentence A = `The man went to the store.`
Sentence B = `Penguins are flightless.`
Label = `NotNextSentence`

[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

BERT: Input Representation

- Use 30,000 WordPiece vocabulary on input.
- Each token is sum of three embeddings
  - Addition to transformer encoder: sentence embedding





[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

Training

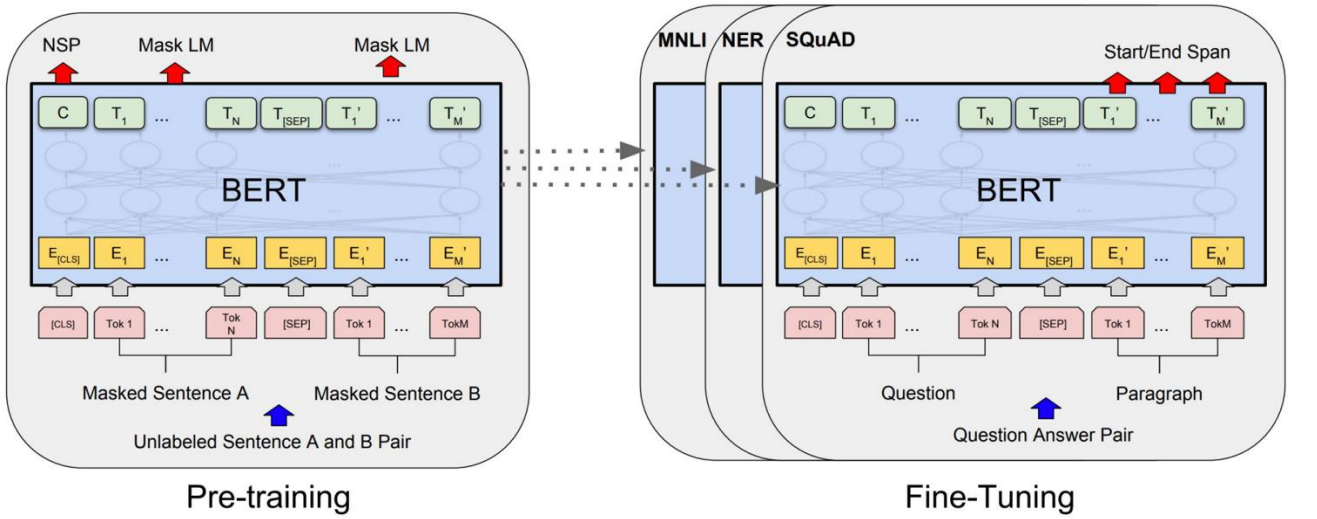
- Trains model on unlabeled data over different pre-training tasks (self-supervised learning)
- **Data:** Wikipedia (2.5B words) + BookCorpus (0.8B words)
- **Training Time:** 1M steps (~40 epochs)
- **Optimizer:** AdamW, 1e-4 learning rate, linear decay
- **BERT-Base:** 12-layer, 768-hidden, 12-head, sequence length of 512
- **BERT-Large:** 24-layer, 1024-hidden, 16-head, sequence length of 512
- Trained on 4x4 and 8x8 TPUs for 4 days (cost today using cloud TPU: $1.3K and $5K)

[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

Fine-tuning BERT

"Pretrain once, finetune many times."

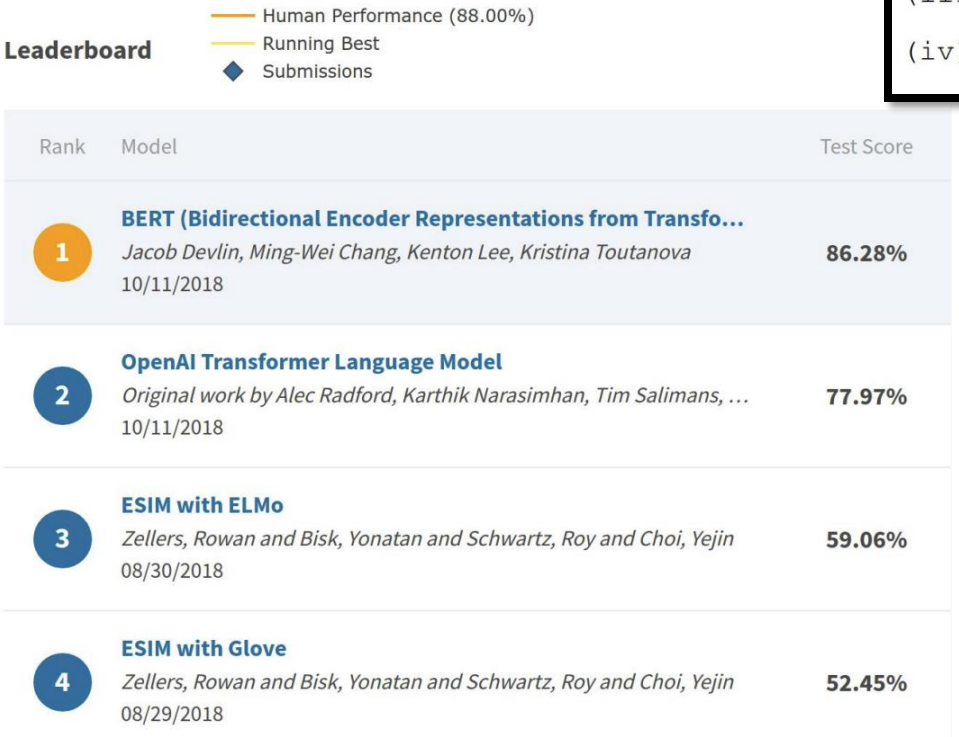
- **Idea:** Make pre-trained model **usable** in **downstream tasks**
- Initialized with pre-trained model parameters
- Fine-tune model parameters using labeled data from downstream tasks



[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

An Example Result: SWAG

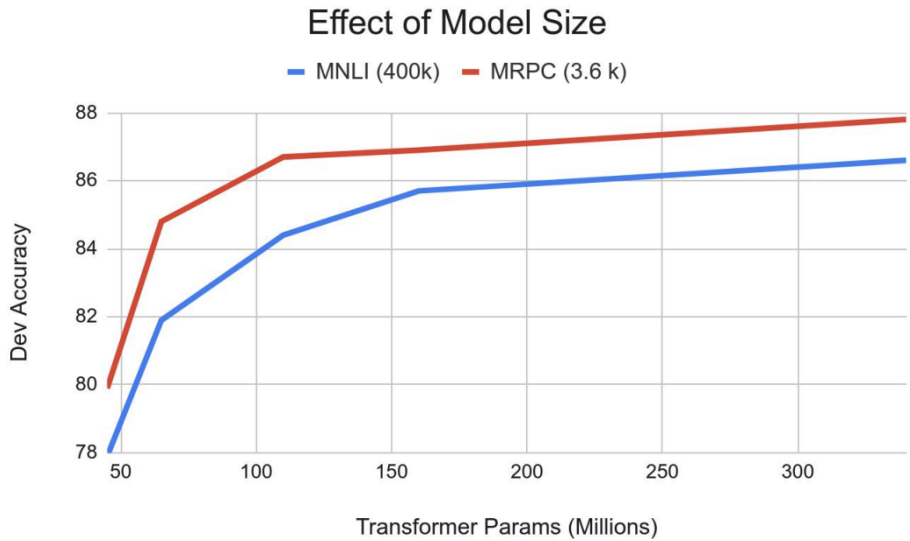
```
A girl is going across a set of monkey bars.  She
(i) jumps up across the monkey bars.
(ii) struggles onto the bars to grab her head.
(iii) gets to the end and stands on a wooden plank.
(iv) jumps up and does a back flip.
```



- Run each Premise + Ending through BERT.
- Produce logit for each pair on token 0 ([CLS])

[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

Effect of Model Size



- Big models help a lot
- Going from 110M -> 340M params helps even on datasets with 3,600 labeled examples
- Improvements have **not** plateaued!

[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

Impact of BERT

- In order to have state-of-the-art performance on different tasks, there is no need for coming up with a novel model architecture
  - End of task-specific model architecture engineering

- An early sign that larger scales and self-supervised learning (language modeling) are the key for future performance improvements

Why did no one think of this before?

- Why wasn't contextual pre-training popular before 2018 with ELMo?

- Good results on pre-training is >1,000x to 100,000 more expensive than supervised training.

What Happened After BERT?

- RoBERTa (Liu et al., 2019)
  - Exact same architecture as BERT
  - Drops the next sentence prediction loss!
  - Trained on 10x data (the original BERT was actually under-trained)
  - Much stronger performance than BERT (e.g., 94.6 vs 90.9 on SQuAD)

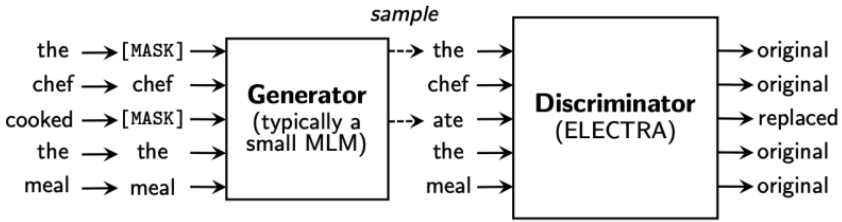
Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

What Happened After BERT?

- RoBERTa (Liu et al., 2019)
  - Exact same architecture as BERT
  - Drops the next sentence prediction loss!
  - Trained on 10x data (the original BERT was actually under-trained)
  - Much stronger performance than BERT (e.g., 94.6 vs 90.9 on SQuAD)

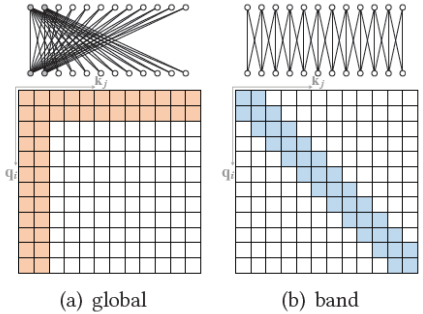
- ALBERT (Lan et al., 2020)
  - Increasing model sizes by sharing model parameters across layers
  - Less storage, much stronger performance but runs slower..

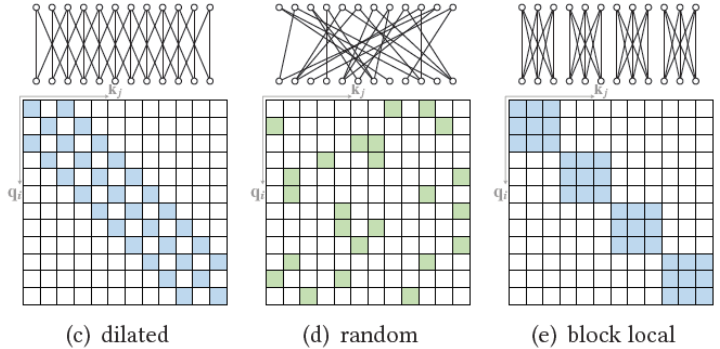
- ELECTRA (Clark et al., 2020)
  - Pre-training objective: replaced-token detection
  - Two models generator and discriminator (GAN-like)
  - It provides a more efficient training method



What Happened After BERT?

- Models that handle long contexts
  - Longformer, Big Bird, …
- Multilingual BERT
  - Trained single model on 104 languages from Wikipedia.
- BERT extended to different domains
  - SciBERT, BioBERT, FinBERT, ClinicalBERT, …
- Making BERT smaller to use
  - DistillBERT, TinyBERT, …





Text generation using BERT

- Does not support generation or sequence-to-sequence tasks
  - Summarization, Translation, Text simplification, etc

BERT has a Mouth, and It Must Speak:
BERT as a Markov Random Field Language Model

Alex Wang
New York University
alexwang@nyu.edu

Kyunghyun Cho
New York University
Facebook AI Research
CIFAR Azrieli Global Scholar
kyunghyun.cho@nyu.edu

Mask-Predict: Parallel Decoding of Conditional Masked Language Models

Marjan Ghazvininejad* **Omer Levy*** **Yinhan Liu*** **Luke Zettlemoyer**
Facebook AI Research
Seattle, WA

Exposing the Implicit Energy Networks behind Masked Language Models via Metropolis--Hastings

Kartik Goyal, Chris Dyer, Taylor Berg-Kirkpatrick

Leveraging Pre-trained Checkpoints for Sequence Generation Tasks

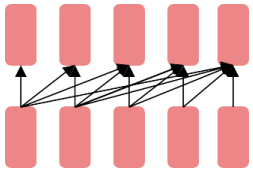
Sascha Rothe, Shashi Narayan, Aliaksei Severyn

src	Der Abzug der franzsischen Kampftruppen wurde am 20. November abgeschlossen .
$t = 0$	The departure of the French combat completed completed on 20 November .
$t = 1$	The departure of French combat troops was completed on 20 November .
$t = 2$	The withdrawal of French combat troops was completed on November 20th .

Summary Thus Far

- BERT and the family
- An encoder; Transformer-based networks trained on massive piles of data.
- Incredible for learning contextualized embeddings of words
- It's very useful to pre-train a large unsupervised/self-supervised LM then fine-tune on your particular task (replace the top layer, so that it can work)
- However, they were not designed to generate text.

Decoder-only Family of Transformers

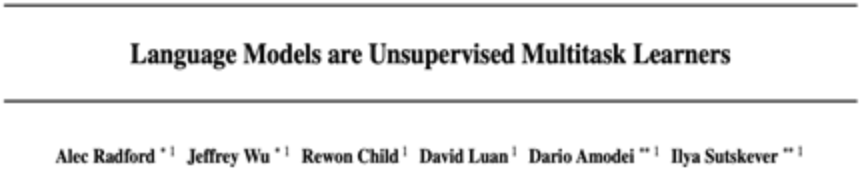


Decoders

GPT

Generative Pre-trained Transformer

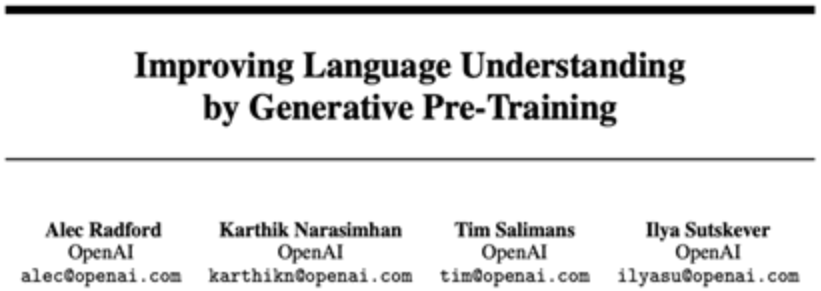
GPT-2: A Big Language Model (2019)



Language Models are Unsupervised Multitask Learners

Alec Radford [*1] Jeffrey Wu [*1] Rewon Child [1] David Luan [1] Dario Amodei [**1] Ilya Sutskever [**1]

GPT: An Auto-Regressive LM (2018)



Improving Language Understanding by Generative Pre-Training

Alec Radford
OpenAI
alec@openai.com

Karthik Narasimhan
OpenAI
karthikn@openai.com

Tim Salimans
OpenAI
tim@openai.com

Ilya Sutskever
OpenAI
ilyasu@openai.com

GPT-2

- GPT-2 uses only Transformer Decoders (no Encoders) to generate new sequences from scratch or from a starting sequence
- As it processes each subword, it masks the "future" words and conditions on and attends to the previous words

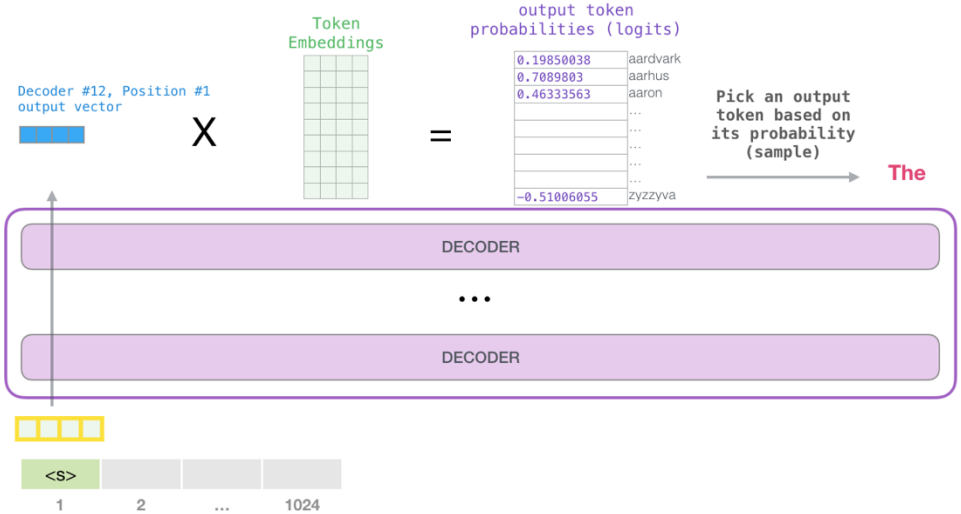


Image by http://jalammar.github.io/illustrated-gpt2/

GPT-3: Just Scale

- More layers & parameters
- Bigger dataset
- Longer training
- Larger embedding/hidden dimension
- Larger context window

[Slide credit: Sbhya Chhabria & Michael Tang]

Impact of GPT3

- Moving away from the fine-tuning paradigm
  - Zero/Few-shot learning and in-context learning
- Massive LM scale makes high zero/few-shot performance possible
- Start of closed source models
  - Not too many details about their model
  - No released code / model checkpoint
- Also revitalized ppen source efforts:
  - OPT, LLaMA by Meta, BLOOM by Huggingface, etc.

GPT4

Model	Usage
davinci-002	$0.0020 / 1K tokens

Model	Input	Output
gpt-4	$0.03 / 1K tokens	$0.06 / 1K tokens

- Transformer-based
  - The rest is …. mystery! ☺
  - If we're going based on costs, GPT4 is ~15-30 times costlier than GPT3. That should give you an idea how its likely size!

- Note, these language models involve more than just pre-training.
  - Pre-training provides the foundation based on which we build the model.
  - We will discuss the later stages (post hoc alignment) in a 2-3 weeks.

https://openai.com/pricing

Other Available [Decoder] LMs

EleutherAI: GPT-Neo (6.7B), GPT-J (6B), GPT-NeoX (20B)

https://huggingface.co/EleutherAI

https://6b.eleuther.ai/

LLaMA, 65B: https://github.com/facebookresearch/llama

Mistral and Mixtral:

https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2

https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

LMSys ChatArena

https://lmarena.ai/

Rank* (UB)	Model	Arena Score	95% CI	Votes	Organization	License	Knowledge Cutoff
1	ChatGPT-4o-latest (2024-08-08)	1317	+5/-5	20805	OpenAI	Proprietary	2023/10
2	Gemini-1.5-Pro-Exp-0801	1298	+4/-4	23232	Google	Proprietary	2023/11
2	Grok-2-08-13	1293	+7/-6	6686	xAI	Proprietary	2024/3
3	GPT-4o-2024-05-13	1286	+3/-3	80741	OpenAI	Proprietary	2023/10
5	GPT-4o-mini-2024-07-18	1275	+5/-4	21621	OpenAI	Proprietary	2023/10
5	Claude 3.5 Sonnet	1271	+3/-3	51097	Anthropic	Proprietary	2024/4
5	Grok-2-Mini-08-13	1268	+7/-7	7266	xAI	Proprietary	2024/3
6	Gemini Advanced App (2024-05-14)	1267	+4/-3	52136	Google	Proprietary	Online
6	Meta-Llama-3.1-405b-Instruct	1266	+4/-4	22312	Meta	Llama 3.1 Community	2023/12
7	GPT-4o-2024-08-06	1262	+5/-5	13703	OpenAI	Proprietary	2023/10
8	Gemini-1.5-Pro-001	1260	+3/-2	72623	Google	Proprietary	2023/11
10	Gemini-1.5-Pro-Preview-0409	1257	+3/-3	55604	Google	Proprietary	2023/11
10	GPT-4-Turbo-2024-04-09	1257	+2/-3	86648	OpenAI	Proprietary	2023/12
12	Mistral-Large-2407	1250	+5/-5	14793	Mistral	Mistral Research	2024/7
12	Athene-70b	1250	+5/-5	13655	NexusFlow	CC-BY-NC-4.0	2024/7
14	GPT-4-1106-preview	1251	+3/-3	93540	OpenAI	Proprietary	2023/4

Transformer Language Models in Practice

Pre-training Transformer LMs

- You have learned about the basics of pre-training Transformer language models.
- There is so much empirical knowledge/experiences that goes into training these models.
- Various empirical issues about:
  - Preparation/pre-processing data
  - Efficient training of models
  - …

Architectural variations

Another View of Architectural Variations

Name	Year	Norm	Parallel Layer	Pre-norm	Position embedding	Activations
Original transformer	2017	LayerNorm	Serial	☐	Sine	ReLU
GPT	2018	LayerNorm	Serial	☐	Absolute	GeLU
T5 (11B)	2019	RMSNorm	Serial	☑	Relative	ReLU
GPT2	2019	LayerNorm	Serial	☑	Sine	GeLU
T5 (XXL 11B) v1.1	2020	RMSNorm	Serial	☑	Relative	GeGLU
mT5	2020	RMSNorm	Serial	☑	Relative	GeGLU
GPT3 (175B)	2020	LayerNorm	Serial	☑	Sine	GeLU
GPTJ	2021	LayerNorm	Parallel	☑	RoPE	GeLU
LaMDA	2021			☑	Relative	GeGLU
Gopher (280B)	2021	RMSNorm	Serial	☑	Relative	ReLU
GPT-NeoX	2022	LayerNorm	Parallel	☑	RoPE	GeLU
BLOOM (175B)	2022	LayerNorm	Serial	☑	AliBi	GeLU
OPT (175B)	2022	LayerNorm	Serial	☑	Absolute	ReLU
PaLM (540B)	2022	RMSNorm	Parallel	☑	RoPE	SwiGLU
Chinchilla	2022	RMSNorm	Serial	☑	Relative	ReLU
Mistral (7B)	2023	RMSNorm	Serial	☑	RoPE	SwiGLU
LLaMA2 (70B)	2023	RMSNorm	Serial	☑	RoPE	SwiGLU
LLaMA (65B)	2023	RMSNorm	Serial	☑	RoPE	SwiGLU
Qwen (14B)	2024	RMSNorm	Serial	☑	RoPE	SwiGLU
DeepSeek (67B)	2024	RMSNorm	Serial	☑	RoPE	SwiGLU
Yi (34B)	2024	RMSNorm	Serial	☑	RoPE	SwiGLU

Low consensus (except pre-norm)

Most try to follow previous successful choices.

[Slide credit: Tatsu Hashimoto]

Grouped Query-Attention

- Used for training LLaMA 2.
- One key-value vector for each group of queries — an interpolation between "multi-head" attention and "multi-query" attention.

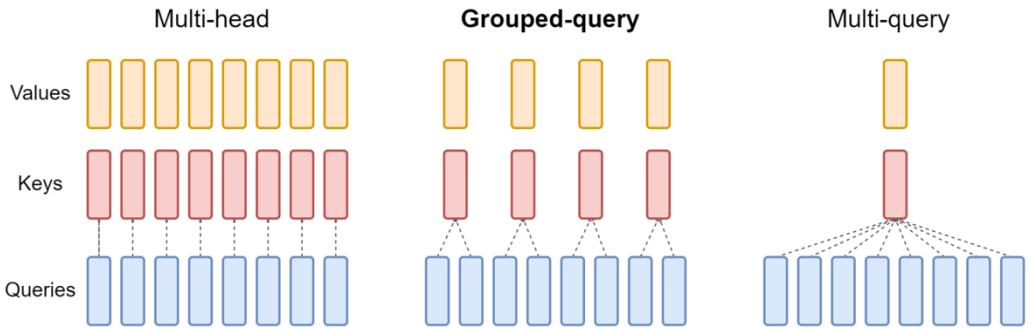


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

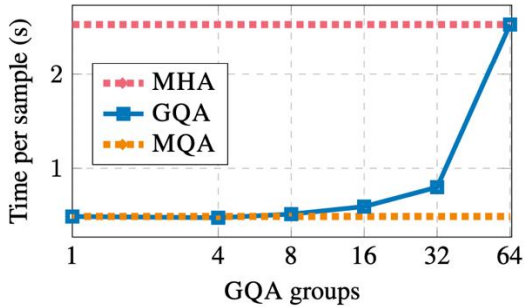


Figure 6: Time per sample for GQA-XXL as a function of the number of GQA groups with input length 2048 and output length 512. Going from 1 (MQA) to 8 groups adds modest inference overhead, with increasing cost to adding more groups.

- Improves inference scalability for our larger models

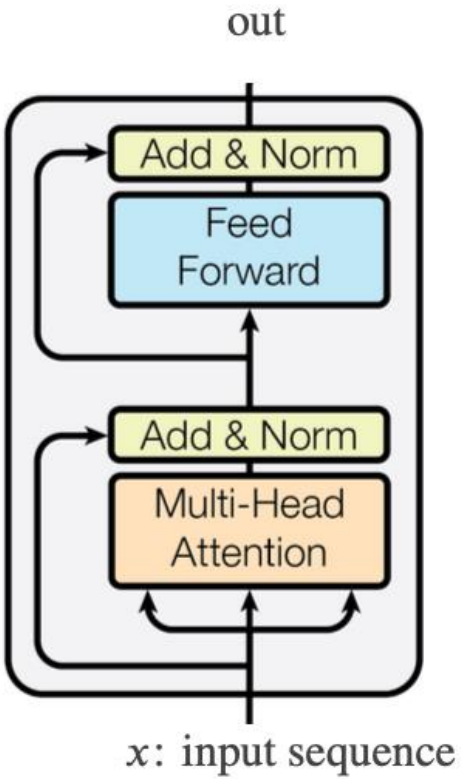
GQA: Training generalized multi-query transformer models from multi-head checkpoints, 2023

Quiz

- Which is the original implementation?
- Does the other implementation work?

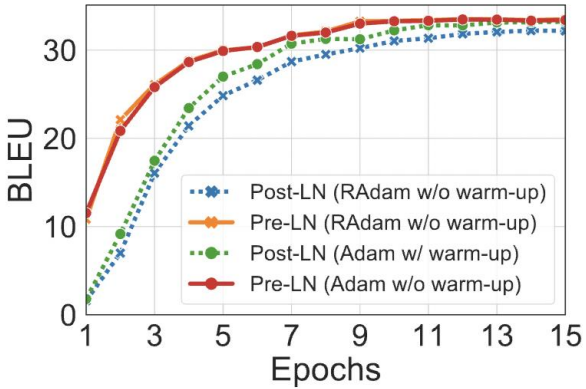
$$LayerNorm\left(x + Dropout\left(SubLayer(x)\right)\right)$$

$$x + Dropout\left(SubLayer\left(LayerNorm(x)\right)\right),$$



Pre-norm vs Post-norm

- Pre-norm (right) is set up so that LayerNorm does not affect the main residual path (in gray).
- In theory, both should work fine.
- In practice, however …



(b) BLEU (IWSLT)

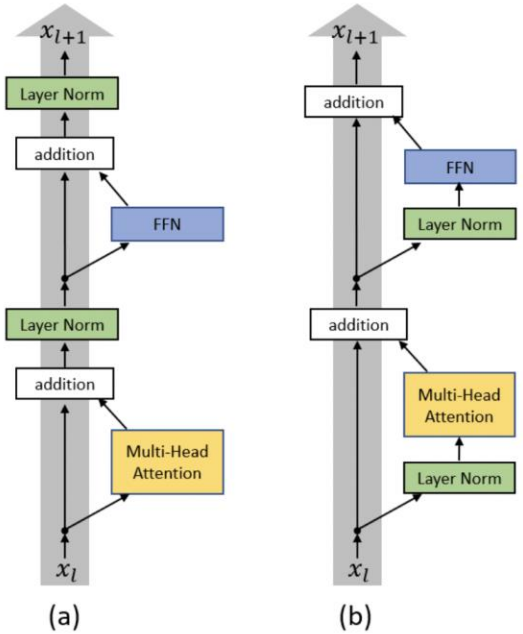


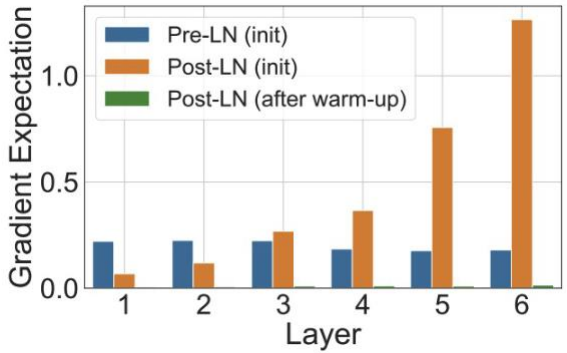
Figure 1. (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

[On Layer Normalization in the Transformer Architecture, 2020]

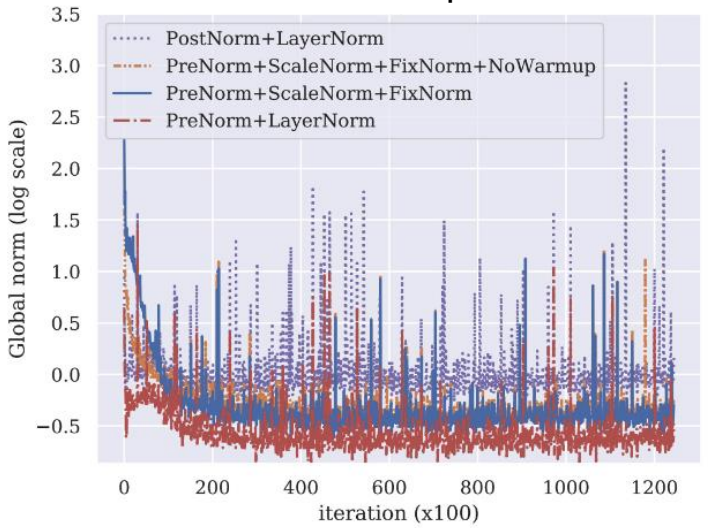
Pre-norm vs Post-norm — Explanation?

- Stability and larger LRs for large networks

Gradient attenuation



Gradient spikes



[Slide credit: Tatsu Hashimoto]

[Left from: On Layer Normalization in the Transformer Architecture, 2020]
[Right from: Transformers without Tears: Improving the Normalization of Self-Attention, 2019]

Layer Norm vs RMSNorm

- Original transformer: **LayerNorm**
  - Normalizes the mean and variance across d_{model}

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

Notable models:
GPT3/2/1, OPT, GPT-J, BLOOM

- Many modern LMs: **RMSNorm**
  - Does not subtract mean or add a bias term

$$y = \frac{x}{\sqrt{||x||_2^2 + \varepsilon}} * \gamma$$

Notable models:
LLaMA-family, PaLM, Chinchilla, T5

[Slide credit: Tatsu Hashimoto]

Why RMSNorm?

- **Modern explanation** – it's faster (and just as good).
  - **Fewer operations** (no mean calculation)
  - **Fewer parameters** (no bias term to store)

- Does this explanation make sense?
  - Matrix multiplies are the vast majority of FLOPs (and memory)
  - Non-matmul ops only make up 0.2% of our FLOPS
    - So perhaps it doesn't matter that GPUs compute non-matmul ops slower.

"Tensor Contraction" := matmuls

Table 1. Proportions for operator classes in PyTorch.

Operator class	% flop	% Runtime
△ Tensor contraction	99.80	61.0
□ Stat. normalization	0.17	25.5
○ Element-wise	0.03	13.5

[Slide credit: Tatsu Hashimoto]

[Data Movement Is All You Need: A Case Study on Optimizing Transformers, 2020]

Why RMSNorm?

- **RMSNorm** runtime (and surprisingly, perf) gains have been seen in papers

Model	Params	Ops	Step/s	Early loss	Final loss	SGLUE	XSum	WebQ	WMT EnDe
Vanilla Transformer	$223M$	$11.1T$	3.50	2.182 ± 0.005	1.838	71.66	17.78	23.02	26.62
RMS Norm	$223M$	$11.1T$	3.68	2.167 ± 0.008	**1.821**	**75.45**	**17.94**	**24.07**	**27.14**

[Slide credit: Tatsu Hashimoto]

[Do Transformer Modifications Transfer Across Implementations and Applications?, 2021]

The Bias Terms

- Most modern transformers don't have bias terms.
  - Original Transformer:

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

  - Most implementations (if they're not gated):

$$FFN(x) = \sigma(xW_1)W_2$$

- Reasons: memory (similar to RMSnorm) and optimization stability

[Slide credit: Tatsu Hashimoto]

Recap

- Basically everyone does pre-norm.
  - Intuition – keep the good parts of residual connections
  - Observations – nicer gradient propagation, fewer spike

- Most people do RMSnorm
  - In practice, works as well as LayerNorm
  - But, has fewer parameters to move around, which saves on wallclock time

- Bias term:
  - People more generally drop bias terms since the compute/param tradeoffs are not great.

[Slide credit: Tatsu Hashimoto]

Activations

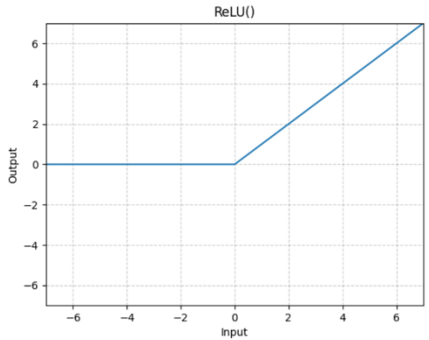
- No much consensus:

  ReLU, GeLU, Swish, ELU, GLU, GeGLU, ReGLU, SeLU, SwiGLU, LiGLU, …

Activations: ReLU vs GeLU

- **ReLU:**

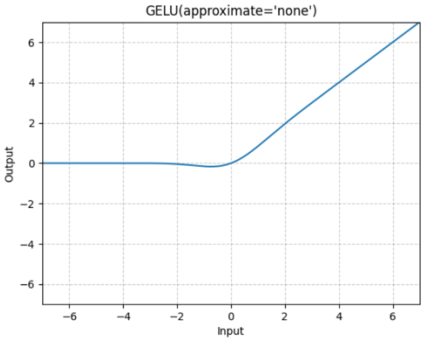
$$FF(x) = \max(0, xW_1)\, W_2$$



Notable models:
Original transformer, T5, Gopher, Chinchilla, OPT

- **GeLU:**

$$FF(x) = \mathrm{GELU}(xW_1)W_2$$
$$GELU(x) \coloneqq x\Phi(x)$$



Notable models:
GPT1/2/3, GPTJ, GPT-Neox, BLOOM

[Slide credit: Tatsu Hashimoto]

Activations: Gated Activations (*GLU)

- GLUs modify the 'first part' of a FF layer:

$$FF(x) = \max(0, xW_1)\, W_2$$

- Instead of a linear + ReLU, augment the above with an (entrywise) linear term:

$$\max(0, xW_1) \rightarrow \max(0, xW_1) \otimes (xV)$$

- This gives the gated variant (ReGLU) – note that we have an extra parameter (V)

$$\mathrm{FF}_{\mathrm{ReGLU}}(x) = (\max(0, xW_1) \otimes xV)\, W_2$$

[Slide credit: Tatsu Hashimoto]

Activations: Gated variants of standard FF layers

- **GeGLU**

$$\mathrm{FFN}_{\mathrm{GEGLU}}(x, W, V, W_2) = (\mathrm{GELU}(xW) \otimes xV)W_2$$

Notable models:
T5 v1.1, mT5, LaMDA

- **SwiGLU:** swish is $x * \mathrm{sigmoid}(x)$

$$\mathrm{FFN}_{\mathrm{SwiGLU}}(x, W, V, W_2) = (\mathrm{Swish}_1(xW) \otimes xV)W_2$$

Notable models:
LLaMa, PaLM

- Note: Gated models use smaller dimensions for the d_ff by 2/3

[Slide credit: Tatsu Hashimoto]

Do gated linear units work?

- Yes, fairly consistently so.

	Score Average	CoLA MCC	SST-2 Acc
FFN_{ReLU}	83.80	51.32	94.04
FFN_{GELU}	83.86	53.48	94.04
$\text{FFN}_{\text{Swish}}$	83.60	49.79	93.69
FFN_{GLU}	84.20	49.16	94.27
$\text{FFN}_{\text{GEGLU}}$	84.12	53.65	93.92
$\text{FFN}_{\text{Bilinear}}$	83.79	51.02	**94.38**
$\text{FFN}_{\text{SwiGLU}}$	84.36	51.59	93.92
$\text{FFN}_{\text{ReGLU}}$	**84.67**	**56.16**	**94.38**
[Raffel et al., 2019]	83.28	53.84	92.68
ibid. stddev.	0.235	1.111	0.569

[Slide credit: Tatsu Hashimoto]

[GLU Variants Improve Transformer, 2020]

Do gated linear units work?

- Yes, fairly consistently so.

Model	Params	Ops	Step/s	Early loss	Final loss	SGLUE	XSum	WebQ
Vanilla Transformer	$223M$	$11.1T$	3.50	2.182 ± 0.005	1.838	71.66	17.78	23.02
GeLU	$223M$	$11.1T$	3.58	2.179 ± 0.003	1.838	**75.79**	**17.86**	**25.13**
Swish	$223M$	$11.1T$	3.62	2.186 ± 0.003	1.847	**73.77**	17.74	**24.34**
ELU	$223M$	$11.1T$	3.56	2.270 ± 0.007	1.932	67.83	16.73	23.02
GLU	$223M$	$11.1T$	3.59	2.174 ± 0.003	**1.814**	**74.20**	**17.42**	24.34
GeGLU	$223M$	$11.1T$	3.55	2.130 ± 0.006	**1.792**	**75.96**	**18.27**	**24.87**
ReGLU	$223M$	$11.1T$	3.57	2.145 ± 0.004	**1.803**	**76.17**	**18.36**	**24.87**
SeLU	$223M$	$11.1T$	3.55	2.315 ± 0.004	1.948	68.76	16.76	22.75
SwiGLU	$223M$	$11.1T$	3.53	2.127 ± 0.003	**1.789**	**76.00**	**18.20**	**24.34**
LiGLU	$223M$	$11.1T$	3.59	2.149 ± 0.005	**1.798**	**75.34**	**17.97**	**24.34**
Sigmoid	$223M$	$11.1T$	3.63	2.291 ± 0.019	1.867	**74.31**	17.51	23.02
Softplus	$223M$	$11.1T$	3.47	2.207 ± 0.011	1.850	**72.45**	17.65	**24.34**

[Slide credit: Tatsu Hashimoto]

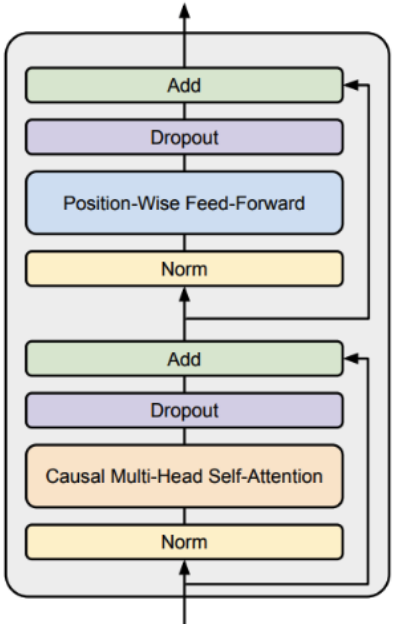
[Do Transformer Modifications Transfer Across Implementations and Applications?, 2021]

Recap: Gating, activations

- Many variations (ReLU, GeLU, *GLU) across models.
- *GLU isn’t necessary for a good model (see GPT3)
- But evidence points towards somewhat consistent gains from Swi/GeGLU

Serial vs Parallel Layer

- Normal transformer blocks are serial – they compute attention, then the MLP
- Could we parallelize the transformer block?



Parallel Layers

- A few models (GPTJ, PaLM, GPT-NeoX) do parallel layers. Originally in GPT-J

Parallel Layers – We use a "parallel" formulation in each Transformer block (Wang & Komatsuzaki, 2021), rather than the standard "serialized" formulation. Specifically, the standard formulation can be written as:

$$y = x + \text{MLP}(\text{LayerNorm}(x + \text{Attention}(\text{LayerNorm}(x)))$$

Whereas the parallel formulation can be written as:

$$y = x + \text{MLP}(\text{LayerNorm}(x)) + \text{Attention}(\text{LayerNorm}(x))$$

The parallel formulation results in roughly 15% faster training speed at large scales, since the MLP and Attention input matrix multiplications can be fused. Ablation experiments showed a small quality degradation at 8B scale but no quality degradation at 62B scale, so we extrapolated that the effect of parallel layers should be quality neutral at the 540B scale.

- If implemented right, LayerNorm can be shared, and matrix multiplies can be fused

Recap

- **Pre-vs-post norm:**
  - Everyone does pre-norm (except OPT350M), likely with good reason.
- **Layer vs RMSnorm:**
  - RMSnorm has clear compute wins, sometimes even performance.
- **Gating:**
  - GLUs seem generally better, though differences are small
- **Serial vs parallel layers:**
  - No extremely serious ablations, but has a compute win.

Positional Encodings

Data

C4: The Data

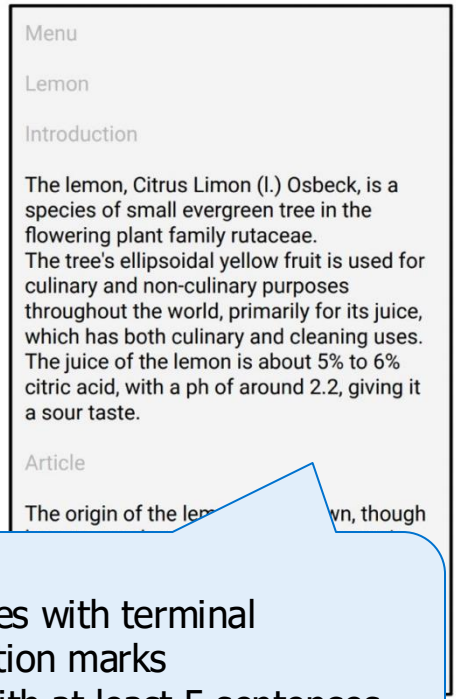
- C4: Colossal Clean Crawled Corpus
  - Web-extracted text
  - English language only
  - 750GB

Data set	Size
★ C4	745GB
C4, unfiltered	6.1TB

Play with the data: https://c4-search.apps.allenai.org/

C4: The Data

Remove any:
- References to Javascript
- "Lorem ipsum" text — placeholder text commonly used to demonstrate the visual form of a document



Menu

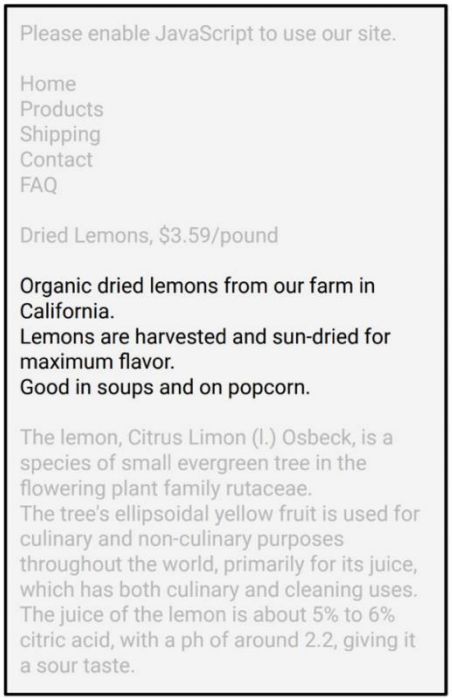
Lemon

Introduction

The lemon, Citrus Limon (l.) Osbeck, is a species of small evergreen tree in the flowering plant family rutaceae.
The tree's ellipsoidal yellow fruit is used for culinary and non-culinary purposes throughout the world, primarily for its juice, which has both culinary and cleaning uses. The juice of the lemon is about 5% to 6% citric acid, with a ph of around 2.2, giving it a sour taste.

Article

The origin of the lem wn, though



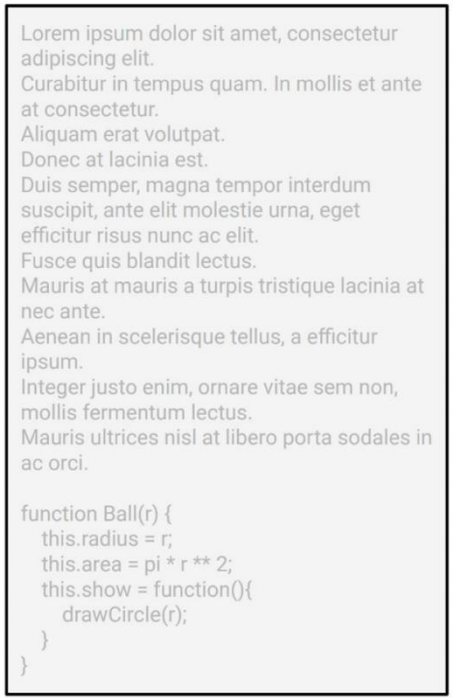
Please enable JavaScript to use our site.

Home
Products
Shipping
Contact
FAQ

Dried Lemons, $3.59/pound

Organic dried lemons from our farm in California.
Lemons are harvested and sun-dried for maximum flavor.
Good in soups and on popcorn.

The lemon, Citrus Limon (l.) Osbeck, is a species of small evergreen tree in the flowering plant family rutaceae.
The tree's ellipsoidal yellow fruit is used for culinary and non-culinary purposes throughout the world, primarily for its juice, which has both culinary and cleaning uses. The juice of the lemon is about 5% to 6% citric acid, with a ph of around 2.2, giving it a sour taste.



Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Curabitur in tempus quam. In mollis et ante at consectetur.
Aliquam erat volutpat.
Donec at lacinia est.
Duis semper, magna tempor interdum suscipit, ante elit molestie urna, eget efficitur risus nunc ac elit.
Fusce quis blandit lectus.
Mauris at mauris a turpis tristique lacinia at nec ante.
Aenean in scelerisque tellus, a efficitur ipsum.
Integer justo enim, ornare vitae sem non, mollis fermentum lectus.
Mauris ultrices nisl at libero porta sodales in ac orci.

```
function Ball(r) {
  this.radius = r;
  this.area = pi * r ** 2;
  this.show = function(){
    drawCircle(r);
  }
}
```

Retain:
- Sentences with terminal punctuation marks
- Pages with at least 5 sentences, sentences with at least 3 words

Slide adapted from Colin Raffel

Pre-training Data: Experiment

- Takeaway:
  - Clean and compact data is better than large, but noisy data.
  - Pre-training on in-domain data helps.

Data set	Size	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ C4	745GB	83.28	**19.24**	80.88	71.36	**26.98**	**39.82**	**27.65**
C4, unfiltered	6.1TB	81.46	19.14	78.78	68.04	26.55	39.34	27.21

Pre-training Data Duplicates

- There is a non-negligible number of duplicates in any pre-training data.

	% train examples with dup in train	% train examples with dup in valid	% valid with dup in train
C4	3.04%	1.59%	4.60%
RealNews	13.63%	1.25%	14.35%
LM1B	4.86%	0.07%	4.92%
Wiki40B	0.39%	0.26%	0.72%

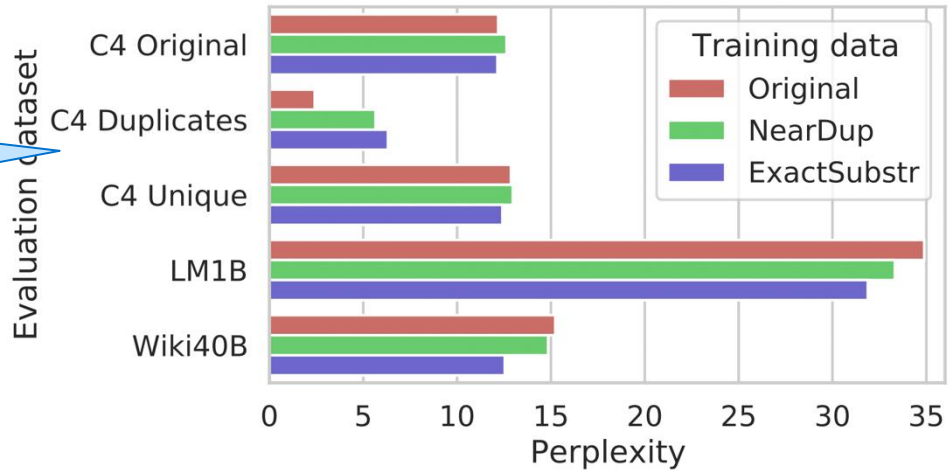
Dataset	Example	Near-Duplicate Example
Wiki-40B	\n_START_ARTICLE_\nHum Award for Most Impactful Character \n_START_SECTION_\nWinners and nominees\n_START_PARAGRAPH_\nIn the list below, winners are listed first in the colored row, followed by the other nominees. [...]	\n_START_ARTICLE_\nHum Award for Best Actor in a Negative Role \n_START_SECTION_\nWinners and nominees\n_START_PARAGRAPH_\nIn the list below, winners are listed first in the colored row, followed by the other nominees. [...]
LM1B	I left for California in 1979 and tracked Cleveland 's changes on trips back to visit my sisters .	I left for California in 1979 , and tracked Cleveland 's changes on trips back to visit my sisters .
C4	Affordable and convenient holiday flights take off from your departure country, "Canada". From May 2019 to October 2019, Condor flights to your dream destination will be roughly 6 a week! Book your Halifax (YHZ) - Basel (BSL) flight now, and look forward to your "Switzerland" destination!	Affordable and convenient holiday flights take off from your departure country, "USA". From April 2019 to October 2019, Condor flights to your dream destination will be roughly 7 a week! Book your Maui Kahului (OGG) - Dubrovnik (DBV) flight now, and look forward to your "Croatia" destination!

Deduplicating Data Improves LMs

- Models: GPT-2-like (1.5B param) models
- On there datasets:
  - C4 : the original training data
  - C4-NearDup: C4 excluding exact duplicates
  - C4-ExactSubs: C4 excluding near-duplicates

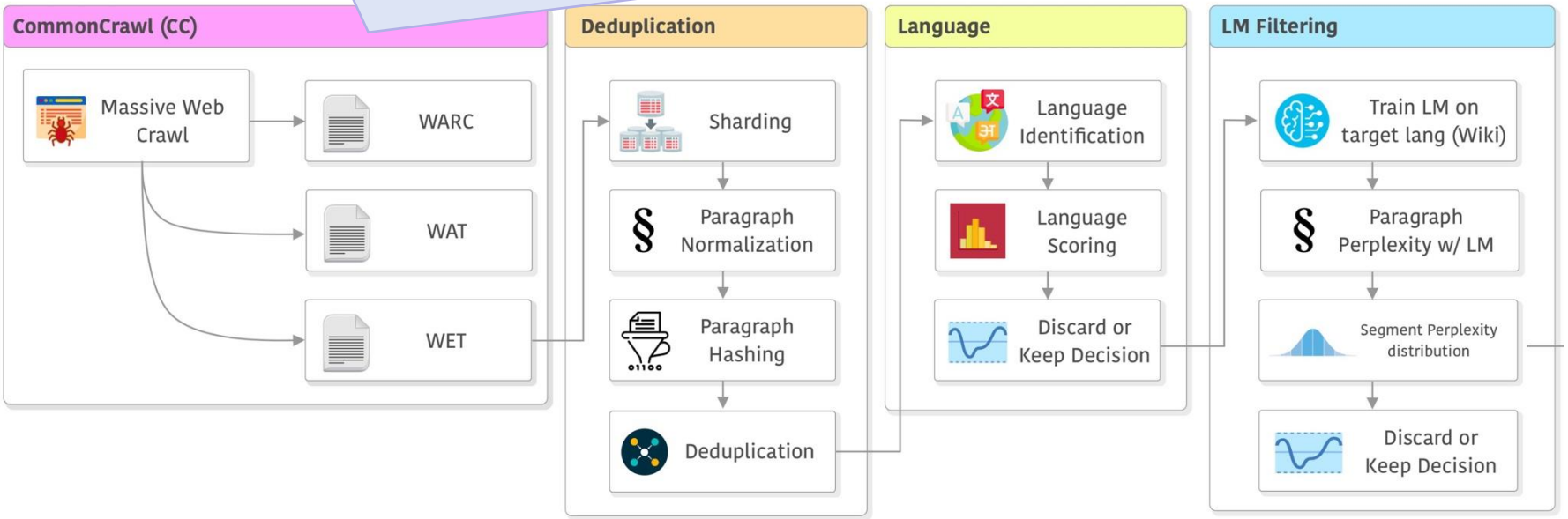
Except when evaluated on duplicate evaluation data!

Training on deduplicated data always leads to lower PPL!



LLaMA's Data Pipeline

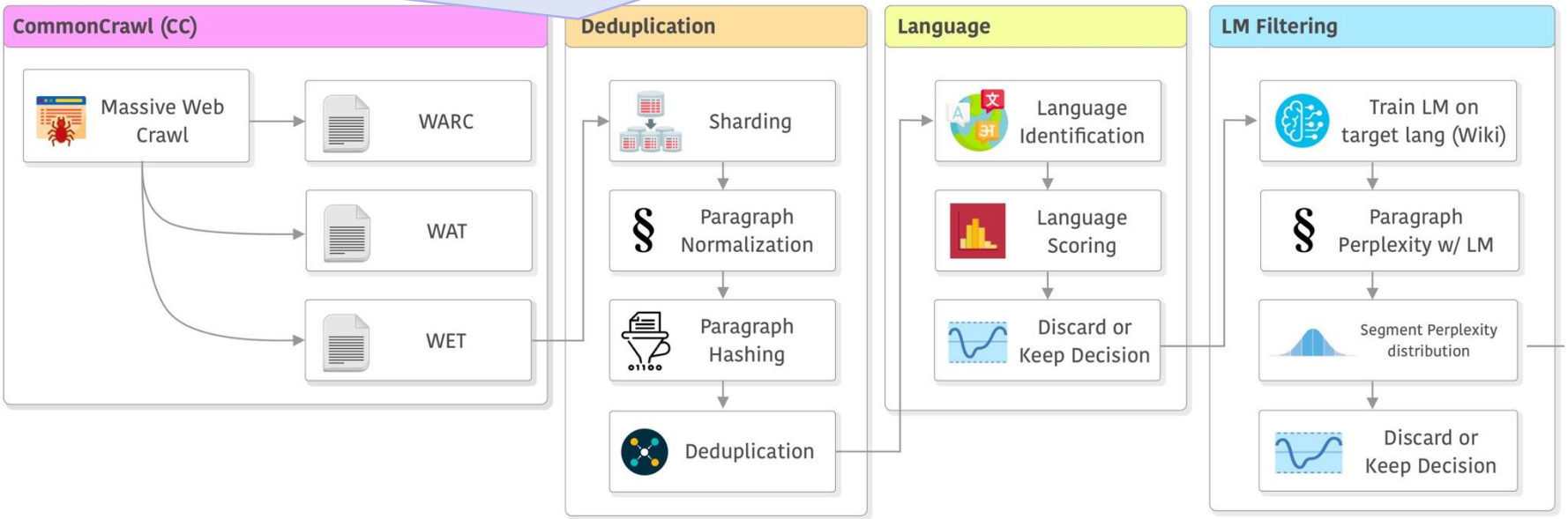
Starts with the massive crawled data by CommonCrawl.
The WET format that contains textual information.
WARC is raw, WAT is metadata, WET is text+some metadata.



https://twitter.com/tarantulae/status/1650170087708454913?t=ncWWY0FI0tYC_dYLs76r5g

LLaMA's Data Pipeline

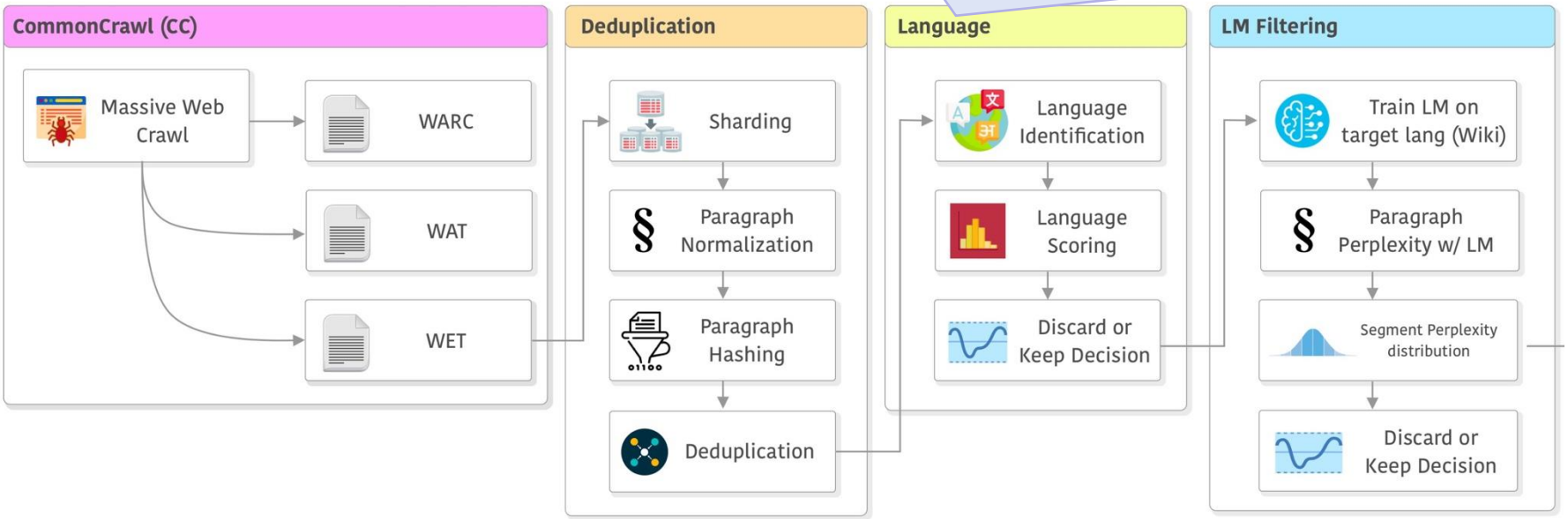
Shard WET content into shards of 5GB each (one CC snapshot can have 30TB). Then you normalize paragraphs (lowercasing, numbers as placeholders, etc), compute per-paragraph hashes and then duplicate them.



https://twitter.com/tarantulae/status/1650170087708454913?t=ncWWY0FI0tYC_dYLs76r5g

LLaMA's Data Pipeline

Perform language identification and decide whether to keep or discard languages.
The order of when you do this in the pipeline can impact the language discrimination quality.

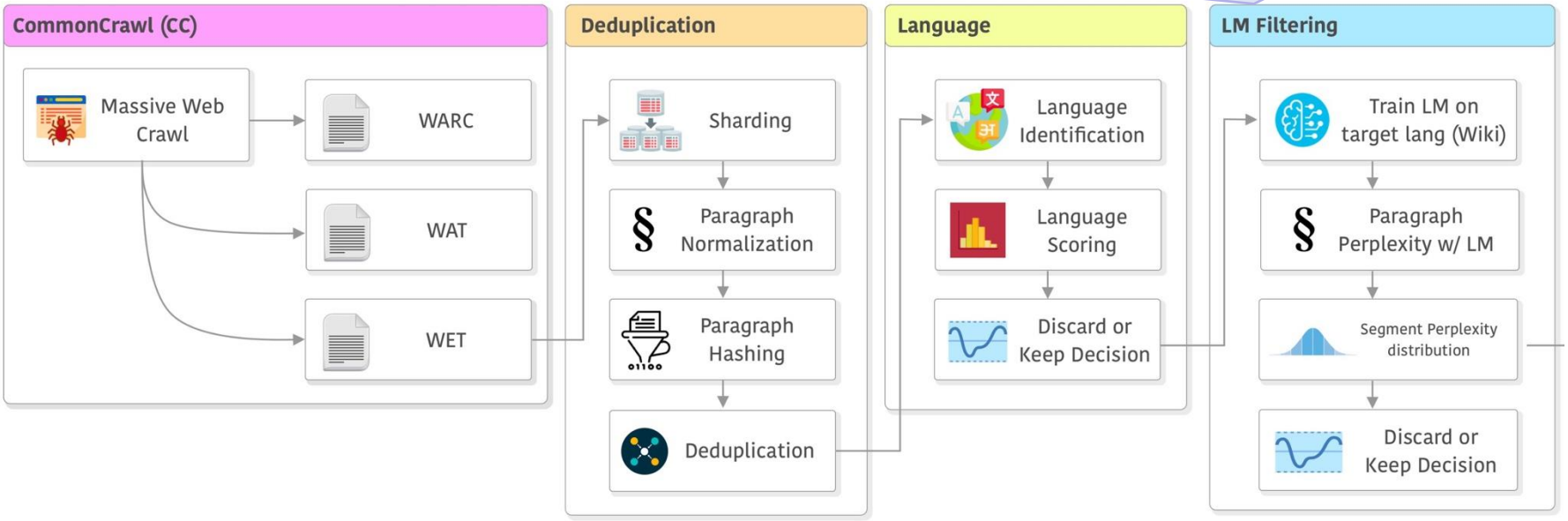


https://twitter.com/tarantulae/status/1650170087708454913?t=ncWWY0FI0tYC_dYLs76r5g

LLaMA's Data Pipeline

Do further quality filtering: Train a simple LM (n-gram) on target languages using Wikipedia, then compute per-paragraph perplexity on the rest of the data:

- Very high PPL: Very different than Wiki and likely low-quality → Drop
- Very low PPL: Very similar or near duplicates to Wiki → Drop



https://huggingface.co/spaces/HuggingFaceFW/blogpost-fineweb-v1

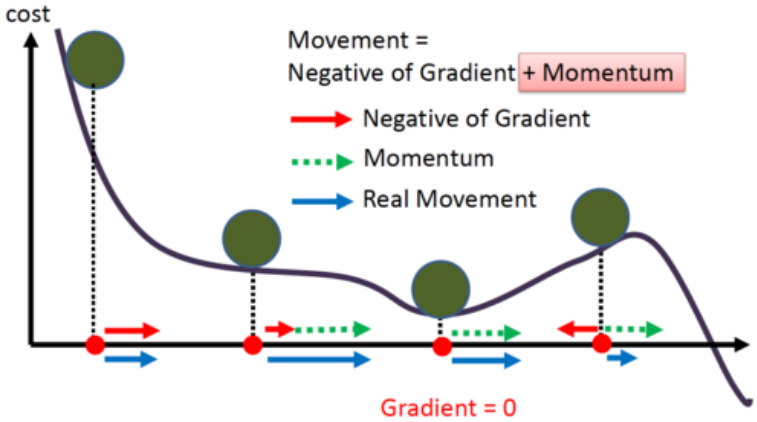
Robot.txt

- https://hal.science/hal-04824161/document
- Data licensing https://arxiv.org/pdf/2310.16787

Training time!

Optimizers

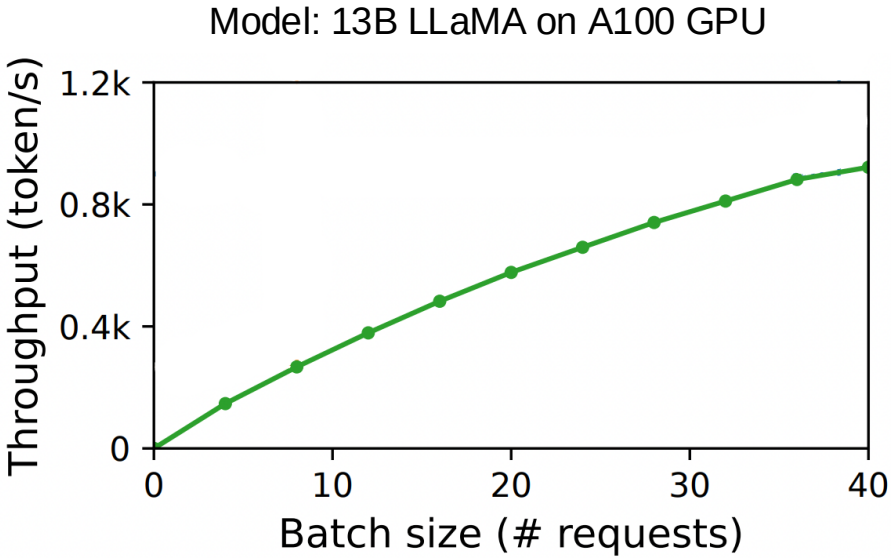
- Most modern models use "AdamW" optimizer (not vanilla Gradient Descent).
  - Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order "momentums".
  - "W" because it decouples "weight decay" from "learning rate". (Details out of scope for us. See the cited paper.)



https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html
https://pytorch.org/docs/stable/generated/torch.optim.Adam.html
[Decoupled Weight Decay Regularization, 2017]

Batching Data

- Previously we talked about the importance of batching data
- GPUs are faster at Tensor operations and hence, we want to do batch processing
- The lager batch of data, the faster they get processed.
- Alas, the speedup is often sub-linear (e.g., 2x larger batch leads to less than 2x speedup).

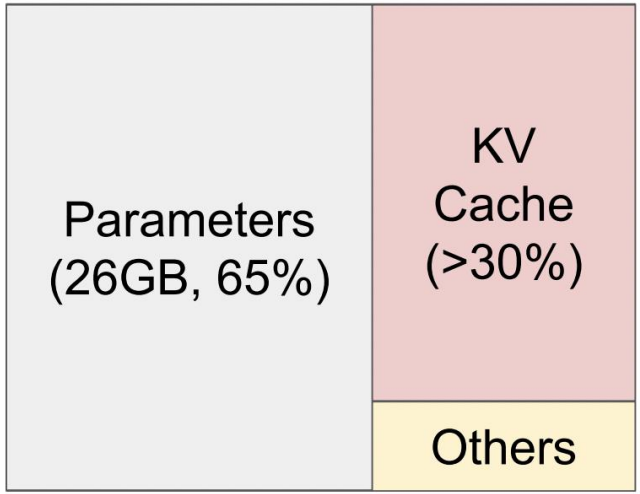


- TODO: There are some nuances about the gains of batch size when using GPUs: https://open.substack.com/pub/finbarrtimbers/p/how-does-batching-work-on-modern?r=2b77bo&utm_campaign=post&utm_medium=email

The Memory Usage

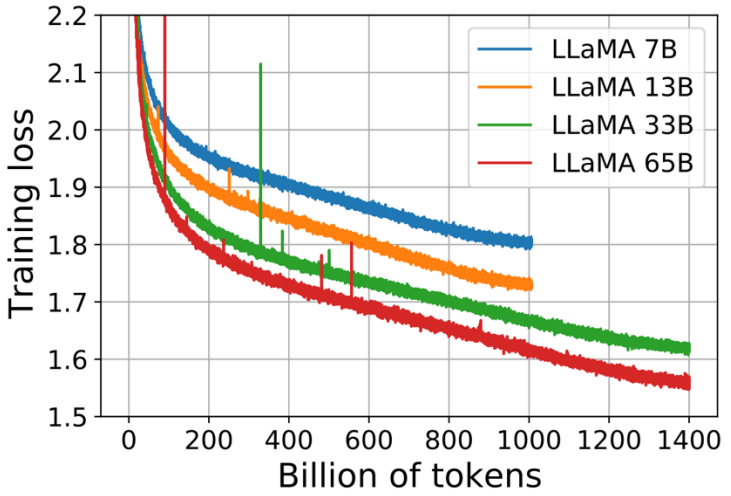
- Here is the memory usage of an NVIDIA A100 when serving (i.e., no training)
  - Model: 13B LLaMA
  - Batch size of 10

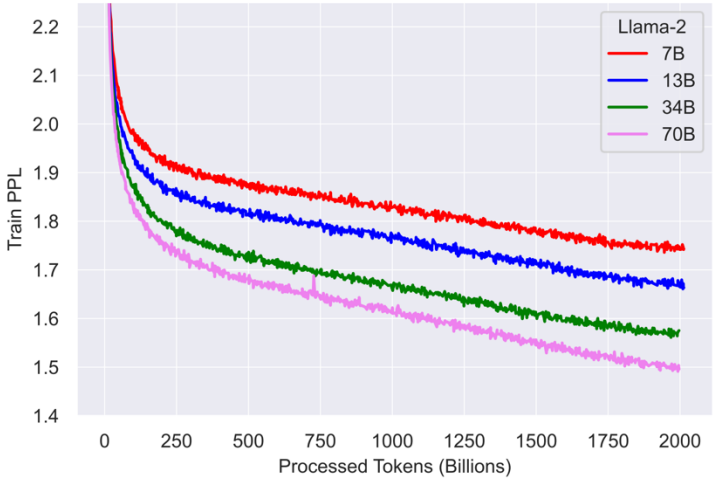
- ~65% of your GPU memory is the model parameters that never change
- ~32% of your memory are KV tensors that change for each input.
  - This KV cache will increase for larger batch sizes.
  - Managing this part of the memory is key for efficient training.



Convergence

- In practice, your model's loss should continue to go down with more training on more data.
- So, the real bottlenecks are:
  - (1) compute
  - (2) data
- Sometimes training diverges (spikes in the loss), at which point practitioners usually restart training from an earlier checkpoint.





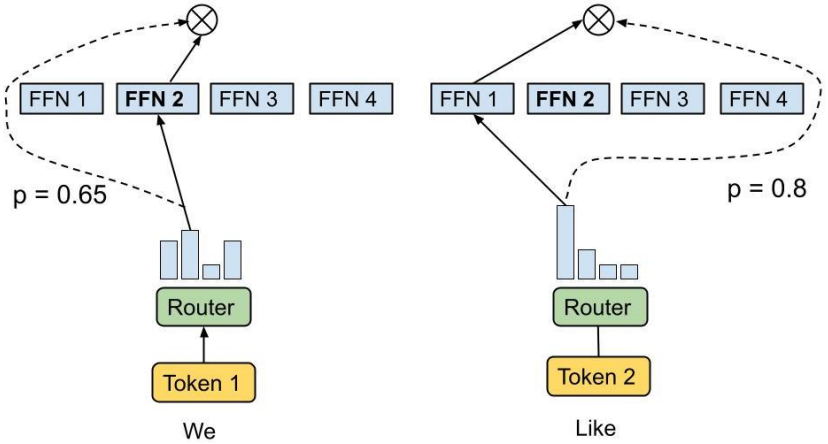
Summary

- There is many empirical knowledge that goes into engineering LMs.
- Here we covered a basic topics about data and architecture engineering.
- Various topics are forthcoming: scaling laws, efficient training, etc.

Mixture of Experts

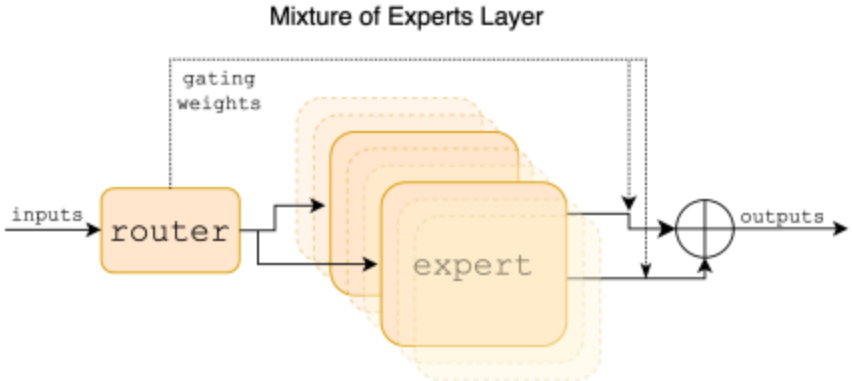
Mixture of Experts

- **Replace dense Feedforward layers with MoE layers!**



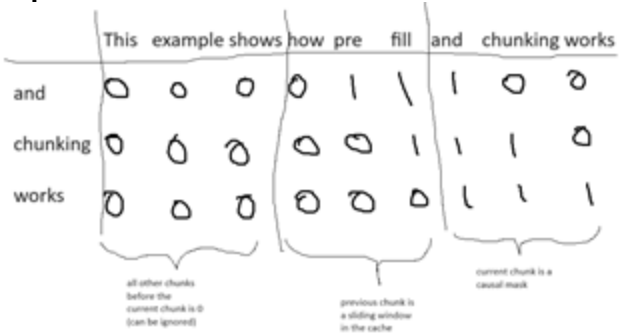
Mixtral of Experts (MoE) introduction

- Mixtral 8x7B is a sparse mixtral of experts (SMoE) model that can outperform Llama-2 70B and GPT3.5 on most benchmarks (math, code generation, multilingual tasks)
- Uses a subset of parameters for every token, can change the size to make inference faster or higher throughput



Mistral architecture

- Mistral 7B is an earlier version of Mixtral
- Similar architecture to Llama, but also with:
  - Sliding Window Attention (SWA) limits the amount of tokens each token can attend to by window size W, to save computation and memory
  - Rolling Buffer Cache was then used to limit cache size to W, and the keys and values at the i-th step are stored at the (i mod W)-th position of the cache
  - Pre-fill and chunking: can pre-fill cache with prompt, or split and pre-fill with each chunk if prompt is large; then, to compute the attention for each chunk, only that of the current chunk and the pr needed



Sparse Mixtral of Experts

- Mixtral has the same architecture and its parameters as Mistral, except the context length is 32k (quadrupled from Mistral), and feed-forward blocks are replaced by 8 MoE layers
- Experts are individual feed-forward networks
- Generally, MoE's output is the sum of the dot products of the expert $E(x)_i$ and its gating network $G(x)_i$, over each expert i
  - Mixtral uses the softmax of top k logits of linear layer: $G(x)=\text{Softmax}(\text{TopK}(x \cdot W_g))$, where TopK is identity for top K logits and -inf otherwise
  - Then, the total (sparse) parameter count can grow with n while the active parameter count (used for processing a token) only grows with k
  - For mixtral, E=SwiGLU and K=2

Mixtral results

- Mixtral was compared to Llama by evaluating them on several benchmarks similar to Llama-2's paper

Model	Active params	MMLU	Hellaswag	Winogander	PI QA	ARC-Easy	ARC-Challenge	Natural Questions	TriviaQA	Human Eval	MBPP	MATH	GSM8K
Llama-2	7B	44.4%	77.1%	69.5%	77.9%	68.7%	43.2%	17.5%	56.6%	11.6%	26.1%	3.9%	16.0%
Llama-2	13B	55.6%	80.7%	72.9%	80.8%	75.2%	48.8%	16.7%	64.0%	18.9%	35.4%	6.0%	34.3%
Llama-1	33B	56.8%	83.7%	76.2%	82.2%	79.6%	54.4%	24.1%	68.5%	25.0%	40.9%	8.4%	44.1%
Llama-2	70B	69.9%	85.4%	80.4%	82.6%	79.9%	56.5%	25.4%	73.0%	29.3%	49.8%	13.8%	69.6%
Mistral	7B	62.5%	81.0%	74.2%	82.2%	80.5%	54.9%	23.2%	62.5%	26.2%	50.2%	12.7%	50.0%
Mixtral	13B	70.6%	84.4%	77.2%	83.6%	83.1%	59.7%	30.6%	71.5%	40.2%	60.7%	28.4%	74.4%

Mixtral results

- Compared to Mistral, multilingual data was significantly upsampled, allowing it to perform well on multilingual benchmarks
- For each language, ARC-Challenge, Hellaswag, and MMLU are reported

Model	French	German	Spanish	Italian
Llama-1 33B	39.3% 68.1% 49.9%	41.1% 63.3% 48.7%	45.7% 69.8% 52.3%	42.9% 65.4% 49.0%
Llama-2 70B	49.9% 72.5% 64.3%	47.3% 68.7% 64.2%	50.5% 74.5% 66.0%	49.4% 70.9% 65.1%
Mixtral 8x7B	58.2% 77.4% 70.9%	54.3% 73.0% 71.5%	55.4% 77.6% 72.5%	52.8% 75.1% 70.9%

Mixtral results

- Long range performance
  - 100% retrieval accuracy of the passkey retrieval task which measures the ability of models to retrieve a passkey randomly inserted in a long prompt
  - Perplexity decreases as context length increases
- Bias benchmarks
  - Mixtral outperforms Llama-2 70B on Bias Benchmark for QA (BBQ) (56.0% vs 51.5%)
  - On Bias in Open-Ended Language Generation Dataset (BOLD), Mixtral has higher scores overall compared to Llama-2, which represents less social bias

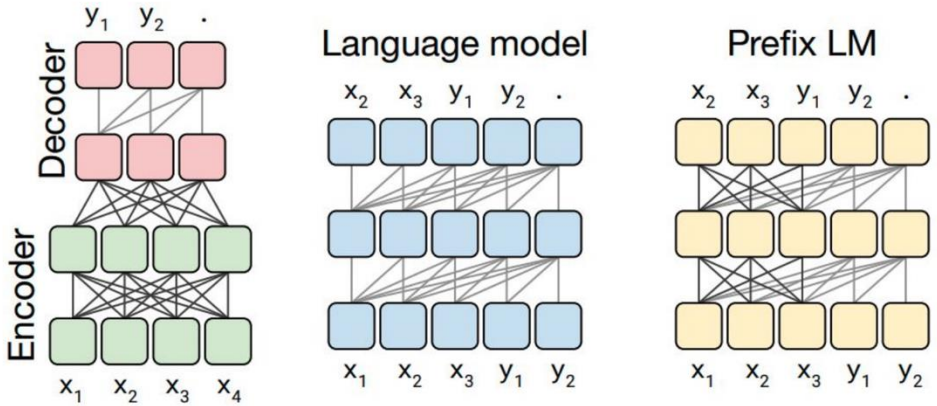
Summary

- TBD

Bonus Content on Transformer Language Models

Architectural choices

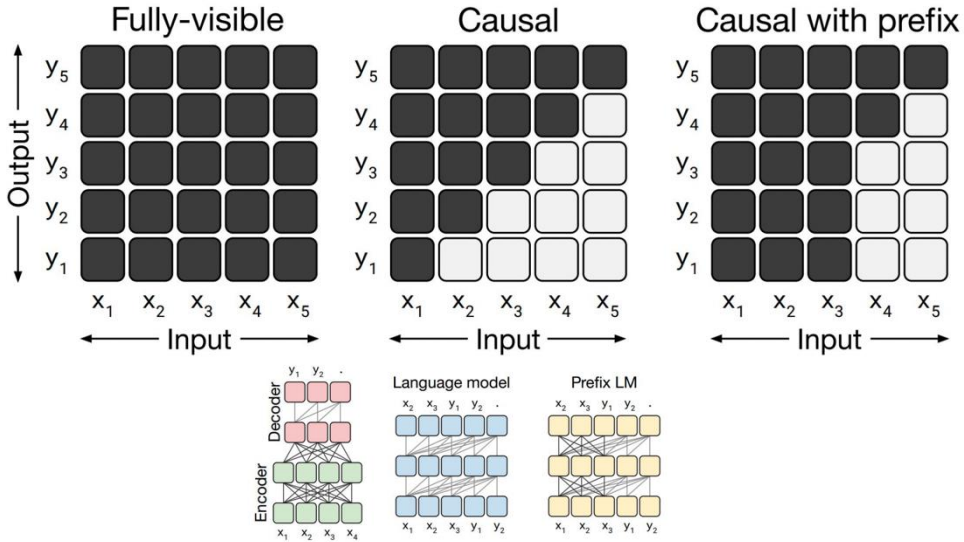
Architectures: Different Choices



Exploring the limits of transfer learning with text-to-text transfer transformers, 2020

Architectures: Different Attention Masks

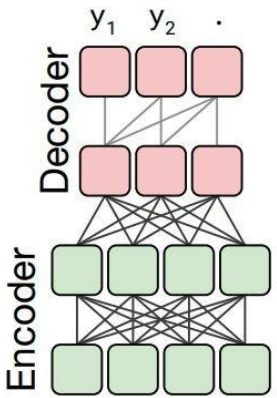
- **Fully visible** mask allows the self attention mechanism to attend to the full input.
- A **causal mask** doesn't allow output elements to look into the future.
- **Causal mask** with prefix allows to fully-visible masking on a portion of input.



Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**

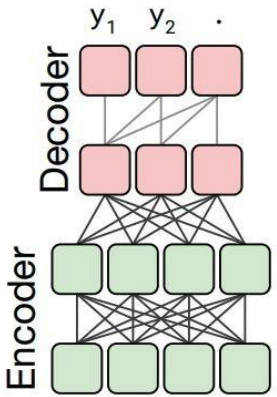


Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**

Input: Thank you for <X> me to your party <Y>. Target: <X> inviting <Y> last week.

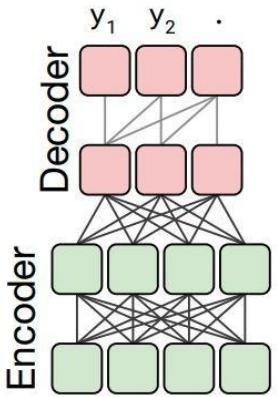


Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**

Number of parameters

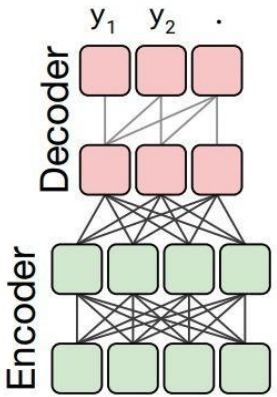


Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**

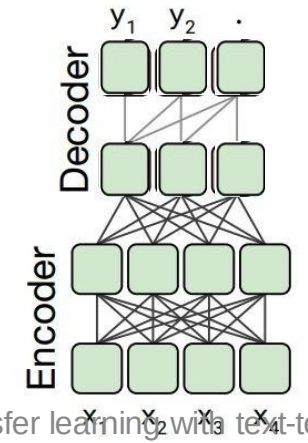
Number of FLOPS



Architectural Variants: Experiments

Evaluated for classification tasks.

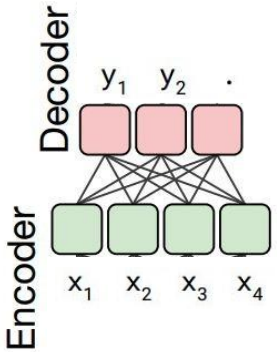
Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**



Architectural Variants: Experiments

Evaluated for classification tasks.

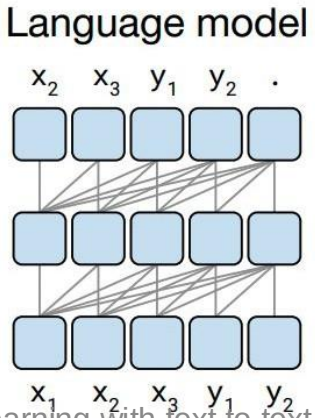
Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95



Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86

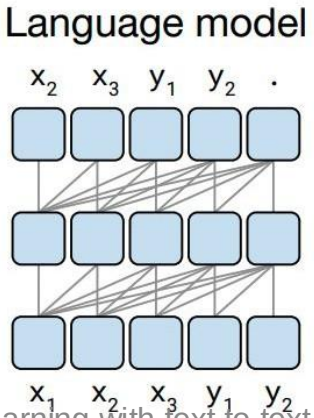


Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86

Language model is decoder-only

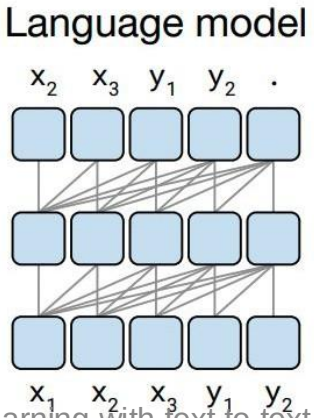


Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86

LM looks at both input and target, while encoder only looks at input sequence and decoder looks at output sequence.

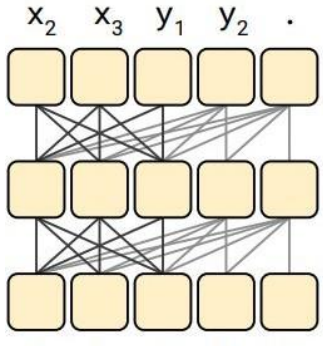


Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39

Prefix LM



Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39

- Takeaways:
  1. Halving the number of layers in encoder and decoder hurts the performance.

Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39

- Takeaways:
  1. Halving the number of layers in encoder and decoder hurts the performance.
  2. Performance of Enc-Dec with shared params is almost on-par with prefix LM.

Pre-training objectives

On Pre-training Objectives

- So far, the dominant objective we have seen is “next-token” prediction.
- In reality any “marginal” observations about language can be a source of supervision.

Objectives

- Prefix language modeling
  - **Input:** Thank you for inviting
  - **Output:** me to your party last week
- BERT-style denoising
  - **Input:** Thank you <M> <M> me to your party apple week
  - **Output:** Thank you for inviting me to your party last week
- Deshuffling
  - **Input:** party me for your to. last fun you inviting week Thanks.
  - **Output:** Thank you for inviting me to your party last week
- IID noise, replace spans
  - **Input:** Thank you <X> me to your party <X> week
  - **Output:** <X> for inviting <Y> last <Z>
- IID noise, drop tokens
  - **Input:** Thank you me to your party week .
  - **Output:** for inviting last

Objectives: Experiments

- All the variants perform similarly
- "Replace corrupted spans" and "Drop corrupted tokens" are more appealing because target sequences are shorter, speeding up training.

Assuming Enc-Dec architecture.
Evaluated for classification tasks.

Objective	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Prefix language modeling	80.69	18.94	77.99	65.27	**26.86**	39.73	**27.49**
Deshuffling	73.17	18.59	67.61	58.47	26.11	39.30	25.62
BERT-style (Devlin et al., 2018)	82.96	19.17	**80.65**	69.85	26.78	**40.03**	27.41
★ Replace corrupted spans	83.28	**19.24**	**80.88**	**71.36**	**26.98**	39.82	**27.65**
Drop corrupted tokens	**84.44**	**19.31**	**80.52**	68.67	**27.07**	39.76	**27.82**