



Transformer Language Models

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

<https://self-supervised.cs.jhu.edu/sp2024/>

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!



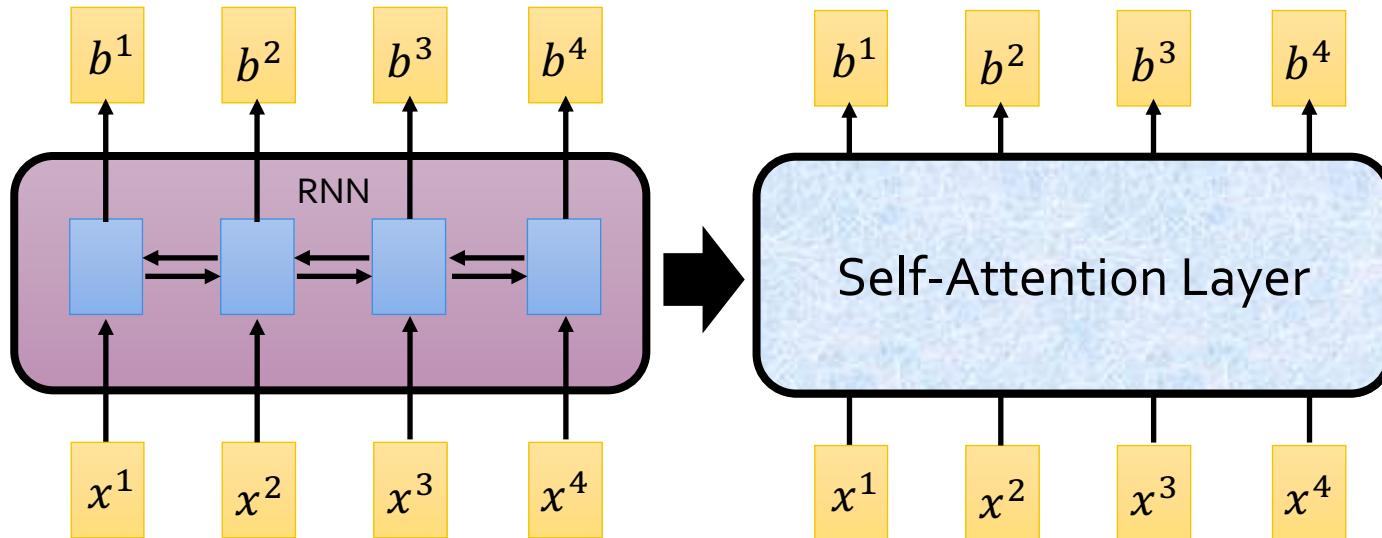
Chapter Plan

1. Self-Attention: how it works
2. Limitations of self-attention
3. Building language models with attention
4. Practical hacks and variants
5. Several recent notable recent language models
6. Various objective functions

Self-Attention

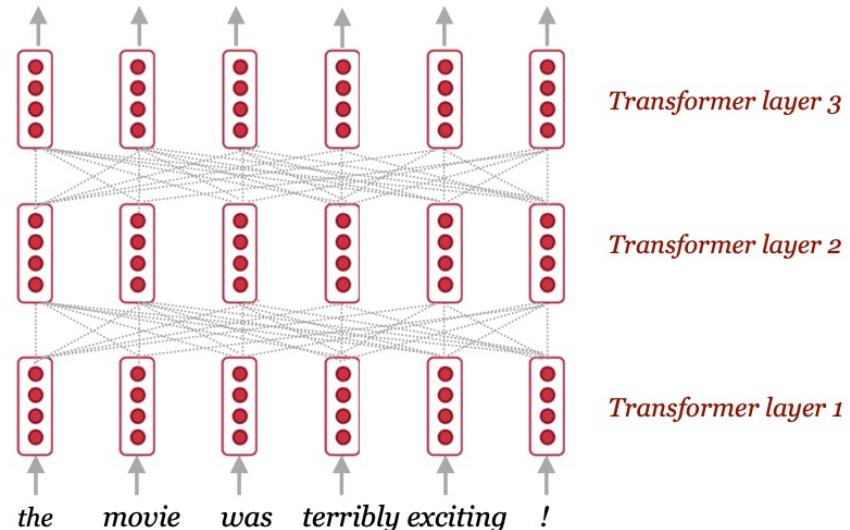
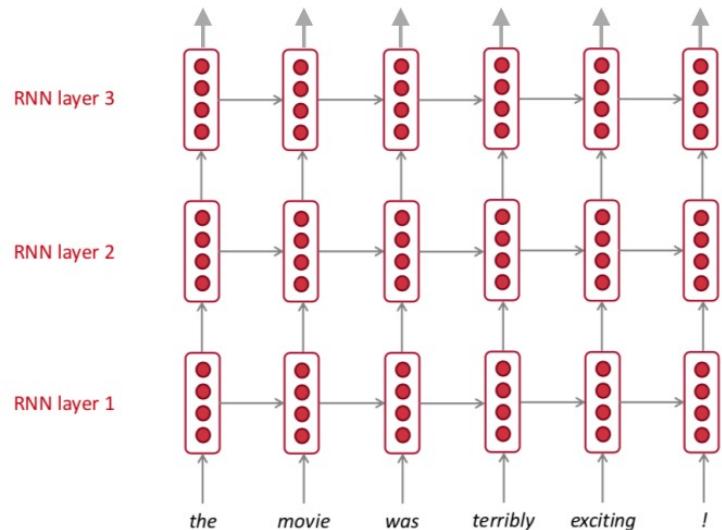
Self-Attention

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



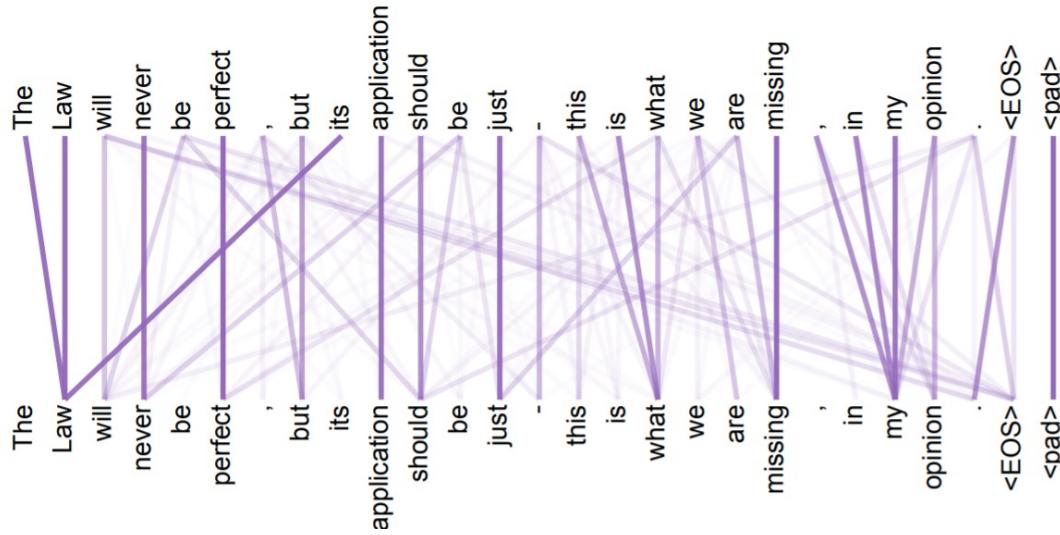
Idea: replace anything done by RNN with **self-attention**.

RNN vs Transformer



Attention

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



Defining Self-Attention

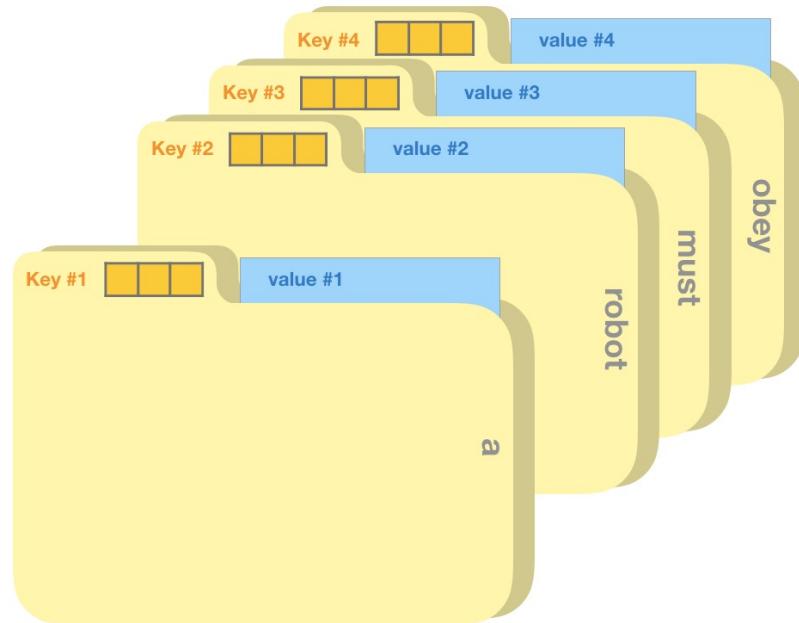
- Terminology:
 - **Query**: to match others
 - **Key**: to be matched
 - **Value**: information to be extracted

Defining Self-Attention

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

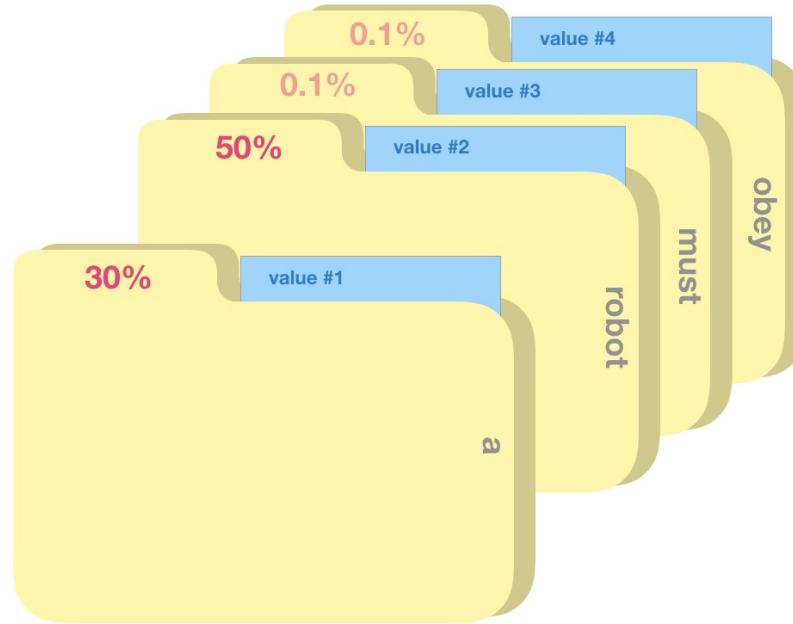
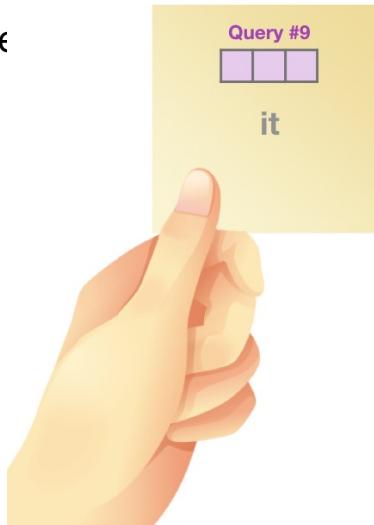


An analogy



Defining Self-Attention

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



q : query (to match others)

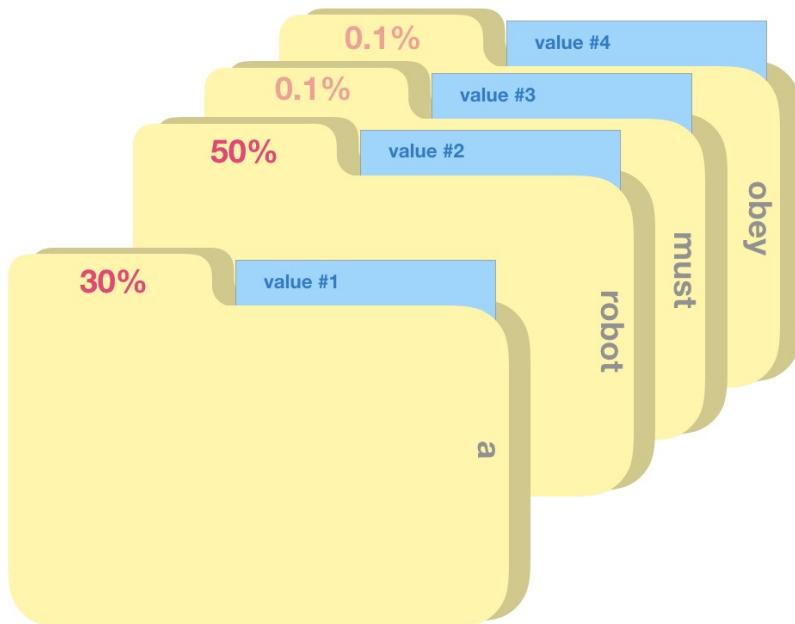
$$q_i = W^q x_i$$

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)

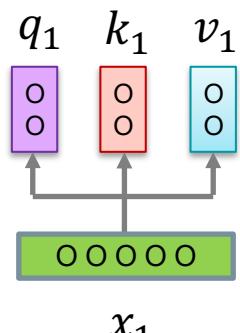
$$q_i = W^q x_i$$

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)

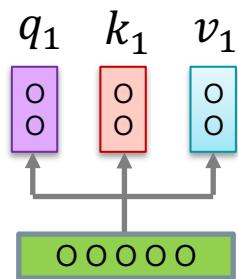
$$q_i = W^q x_i$$

k : key (to be matched)

$$k_i = W^k x_i$$

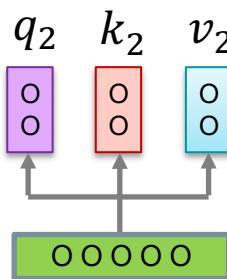
v : value (information to be extracted)

$$v_i = W^v x_i$$



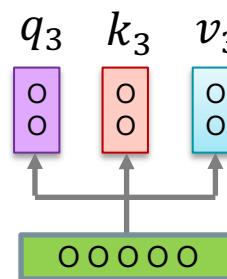
x_1

The



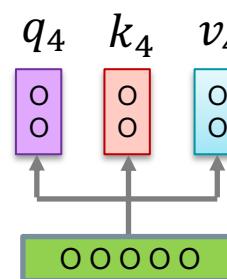
x_2

cat



x_3

sat



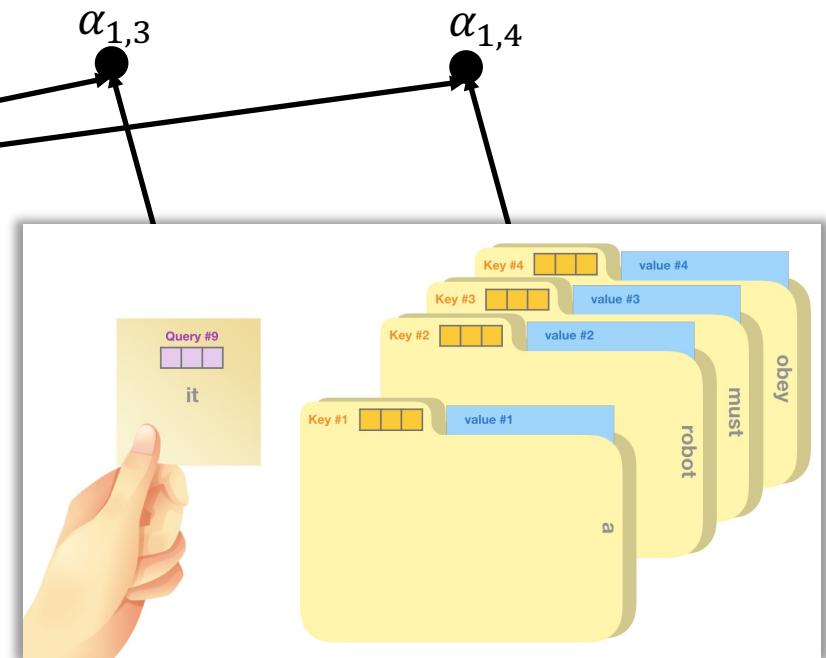
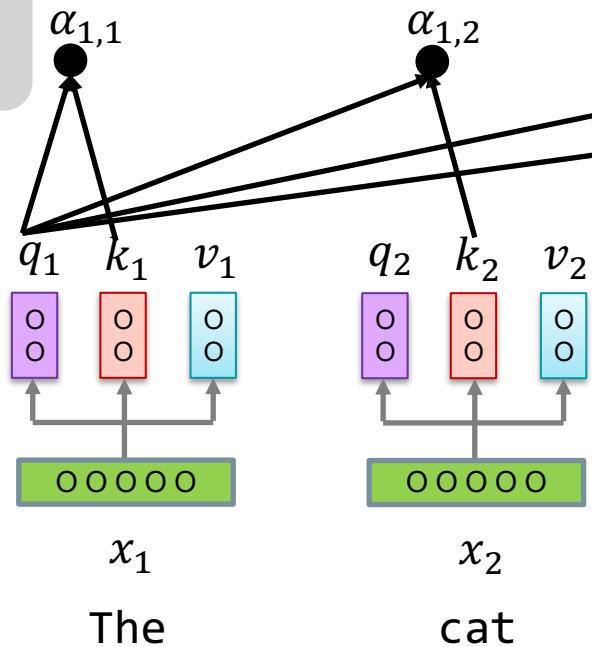
x_4

on

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

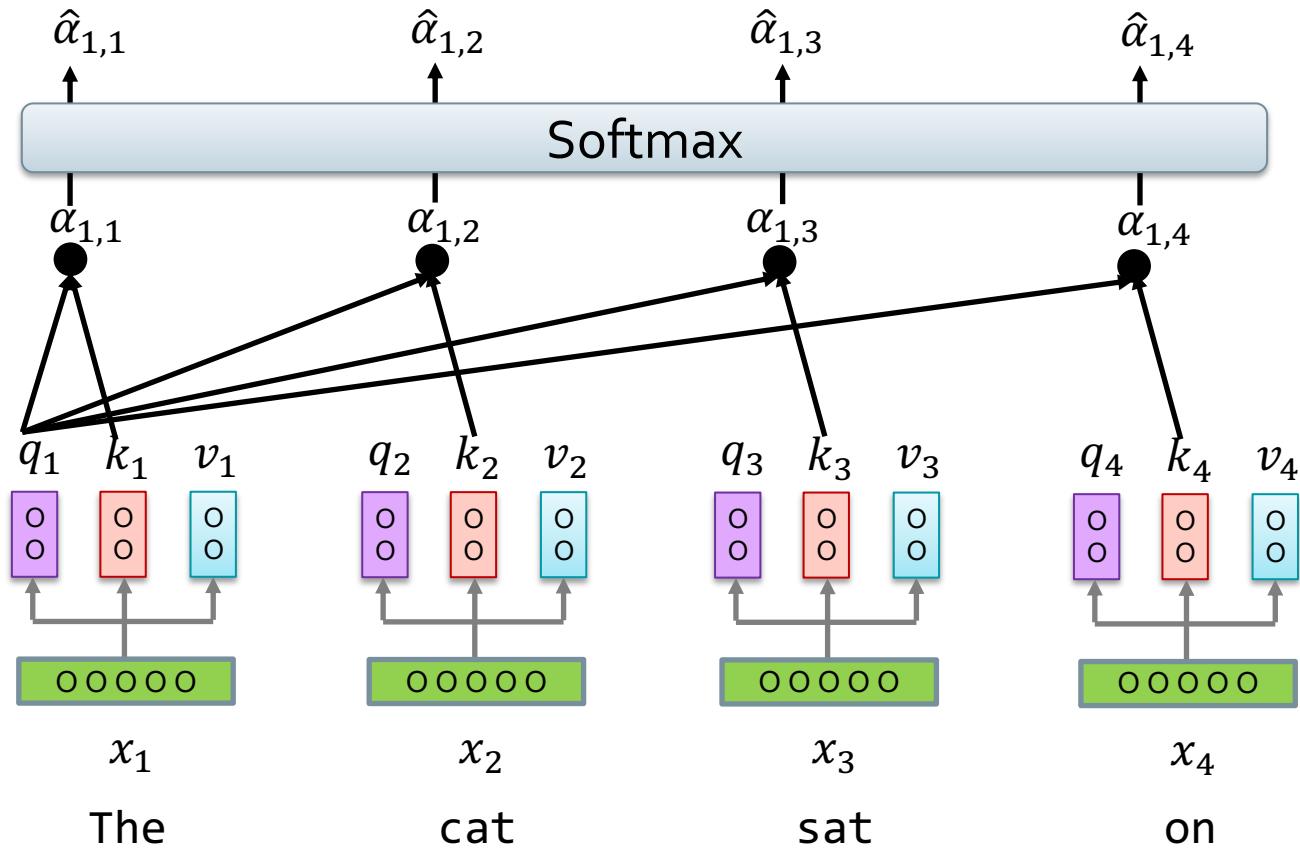
q: query (to match others)
k: key (to be matched)
v: value (information to be extracted)

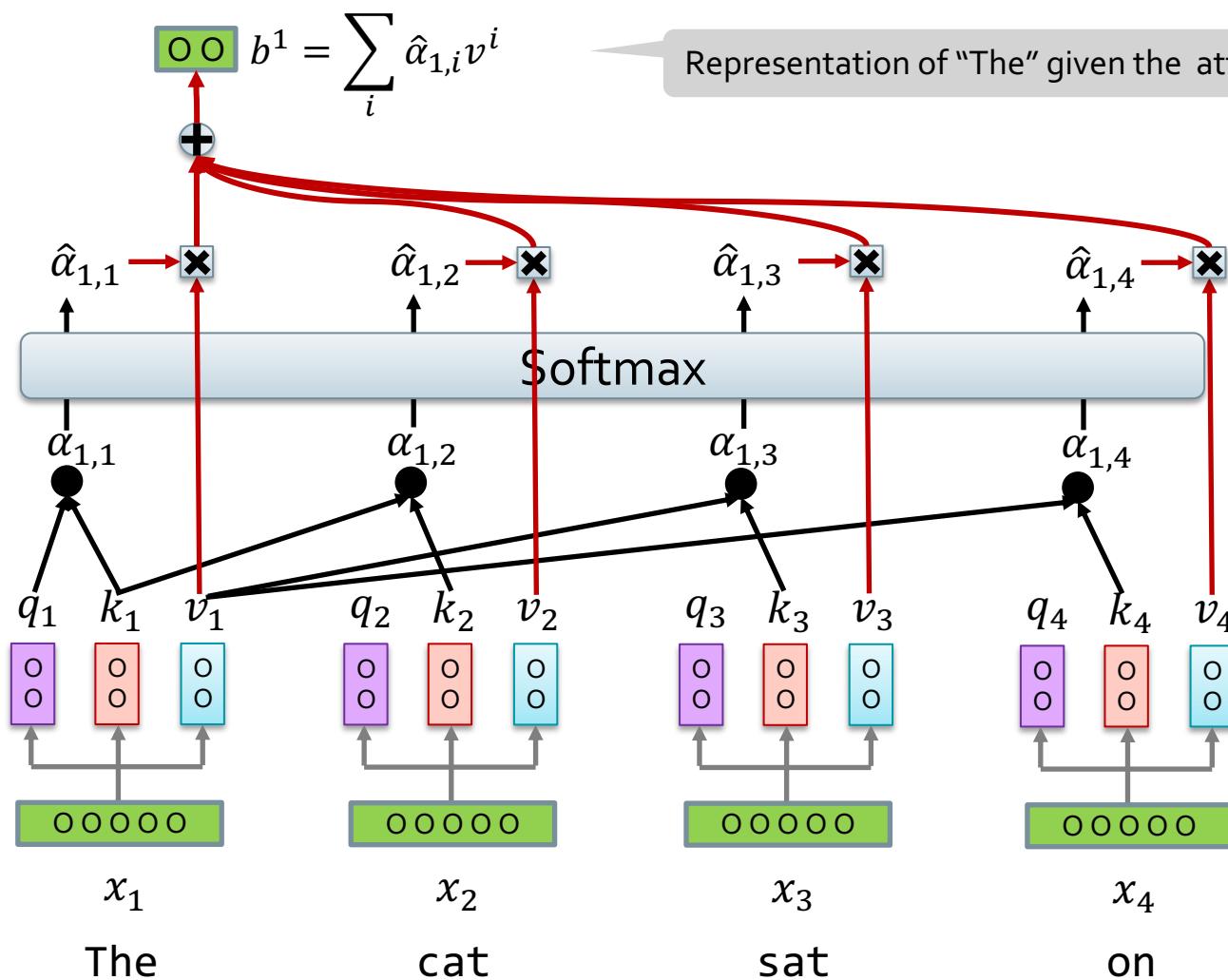
How much should
"The" attend to
other positions?



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

How much
should "The"
attend to other
positions?





Self-Attention

- Can write it in matrix form:
- Given input \mathbf{x} :

$$Q = \mathbf{W}^q \mathbf{x}$$

$$K = \mathbf{W}^k \mathbf{x}$$

$$V = \mathbf{W}^v \mathbf{x}$$

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



The most important formula in deep learning after 2018

Self-Attention

What is self-attention? Self-attention calculates a weighted average of feature representations with the weight proportional to a similarity score between pairs of representations. Formally, an input sequence of n tokens of dimensions d , $X \in \mathbf{R}^{n \times d}$, is projected using three matrices $W_Q \in \mathbf{R}^{d \times d_q}$, $W_K \in \mathbf{R}^{d \times d_k}$, and $W_V \in \mathbf{R}^{d \times d_v}$ to extract feature representations Q , K , and V , referred to as query, key, and value respectively with $d_k = d_q$. The outputs Q , K , V are computed as

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V. \quad (1)$$

So, self-attention can be written as,

$$S = D(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_q}}\right)V, \quad (2)$$

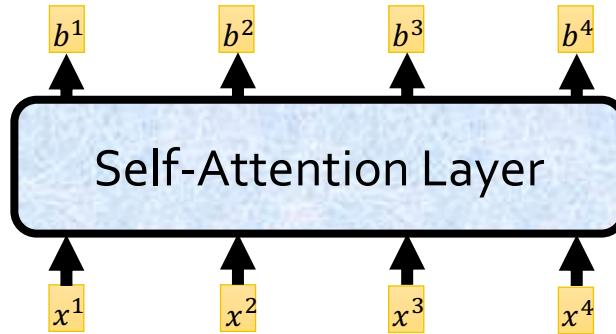
where softmax denotes a *row-wise* softmax normalization function. Thus, each element in S depends on all other elements in the same row.

9:08 PM · Feb 9, 2021 · Twitter Web App

553 Retweets 42 Quote Tweets 3,338 Likes

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.

Self-Attention in PyTorch

- TODO: define Einsum

Args:

```
X: a tensor with shape [b, n, d]
M: a tensor with shape [b, m, d]
mask: a tensor with shape [b, h, n, m]
P_q: a tensor with shape [h, d, k]
P_k: a tensor with shape [d, k]
P_v: a tensor with shape [d, v]
P_o: a tensor with shape [h, d, v]
```

Returns:

```
Y: a tensor with shape [b, n, d]
.....
Q = torch.einsum("bnd,hdk->bhnk", X, proj_q)
K = torch.einsum("bmd,dk->bmk", M, proj_k)
V = torch.einsum("bmd,dv->bmv", M, proj_v)
logits = torch.einsum("bhnk,bmk->bhnm", Q, K)
weights = torch.softmax(logits, dim=-1)
O = torch.einsum("bhnm,bmv->bhnv", weights, V)
Y = torch.einsum("bhnv,hdv->bnd", O, proj_o)
return Y
```

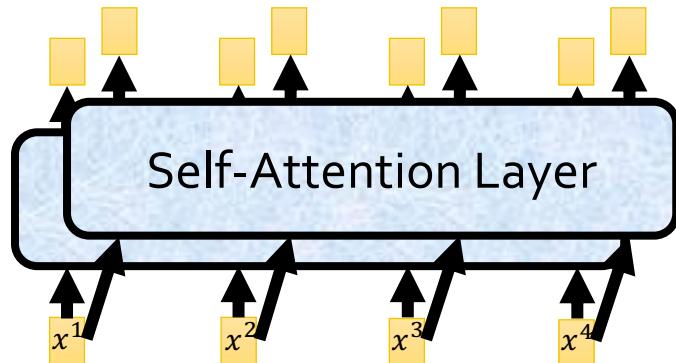
Properties of Self-Attention

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)
- Efficient implementations

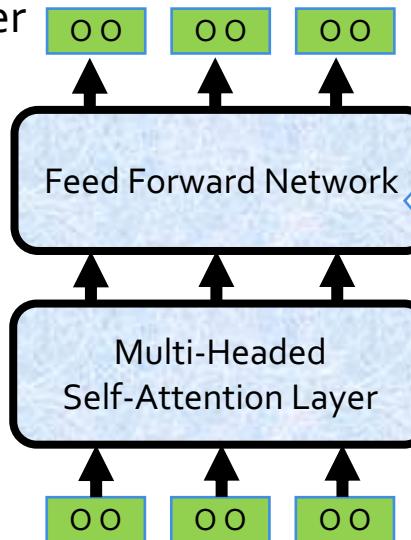
Multi-Headed Self-Attention

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

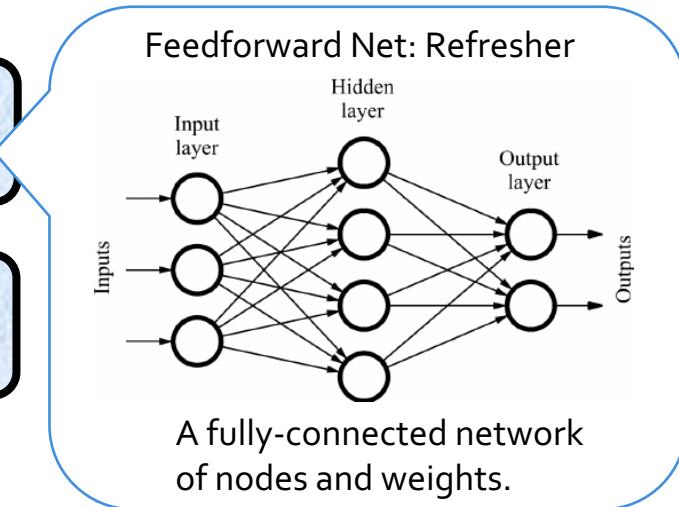


Combine with FFN

- 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.

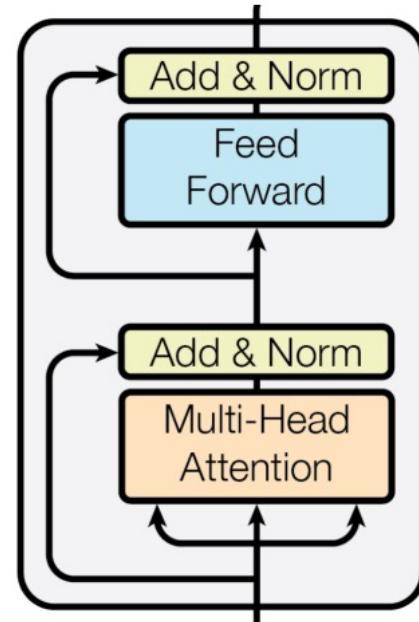


$$\text{FFN}(\mathbf{x}) = f(cW_1 + b_1)W_2 + b_2$$



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

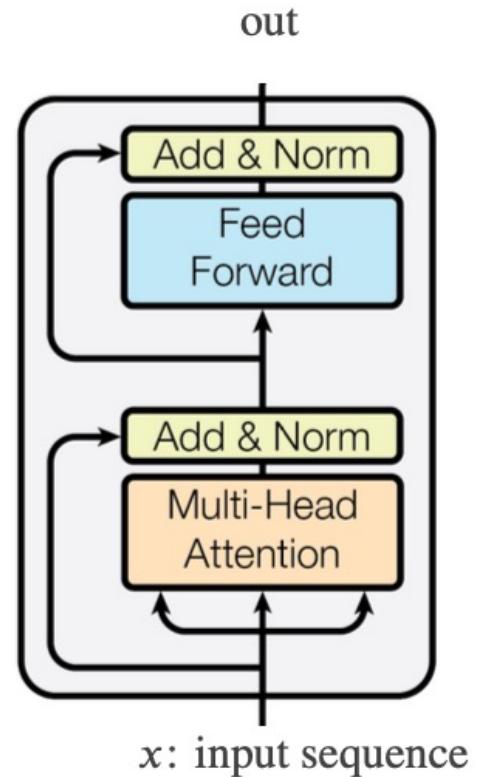


Putting it Together: Self-Attention Block

Given input \mathbf{x} :

$$\begin{aligned}\text{out} &= LN(\tilde{\mathbf{c}} + \mathbf{c}') \\ \tilde{\mathbf{c}} &= \text{FFN}(\mathbf{c}') = f(\mathbf{c}'W_1 + b_1)W_2 + b_2\end{aligned}$$

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



Summary: Self-Attention Block

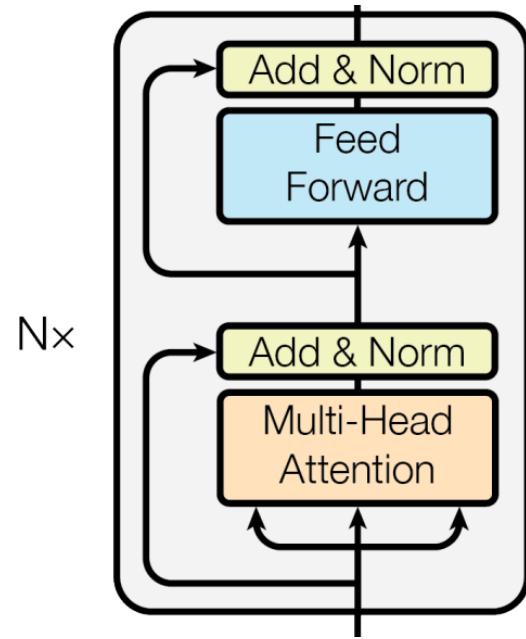
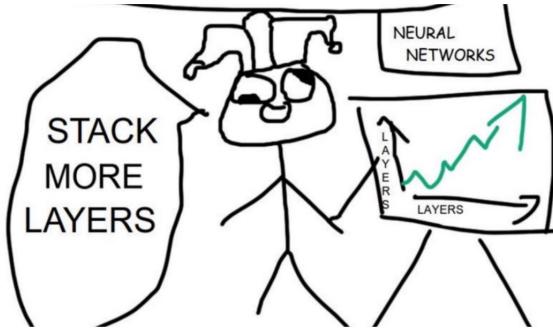
- **Self-Attention:** A critical building block of modern language models.
- TODO:
- **Next:** We will combine self-attention blocks to build various architectures known as Transformer.

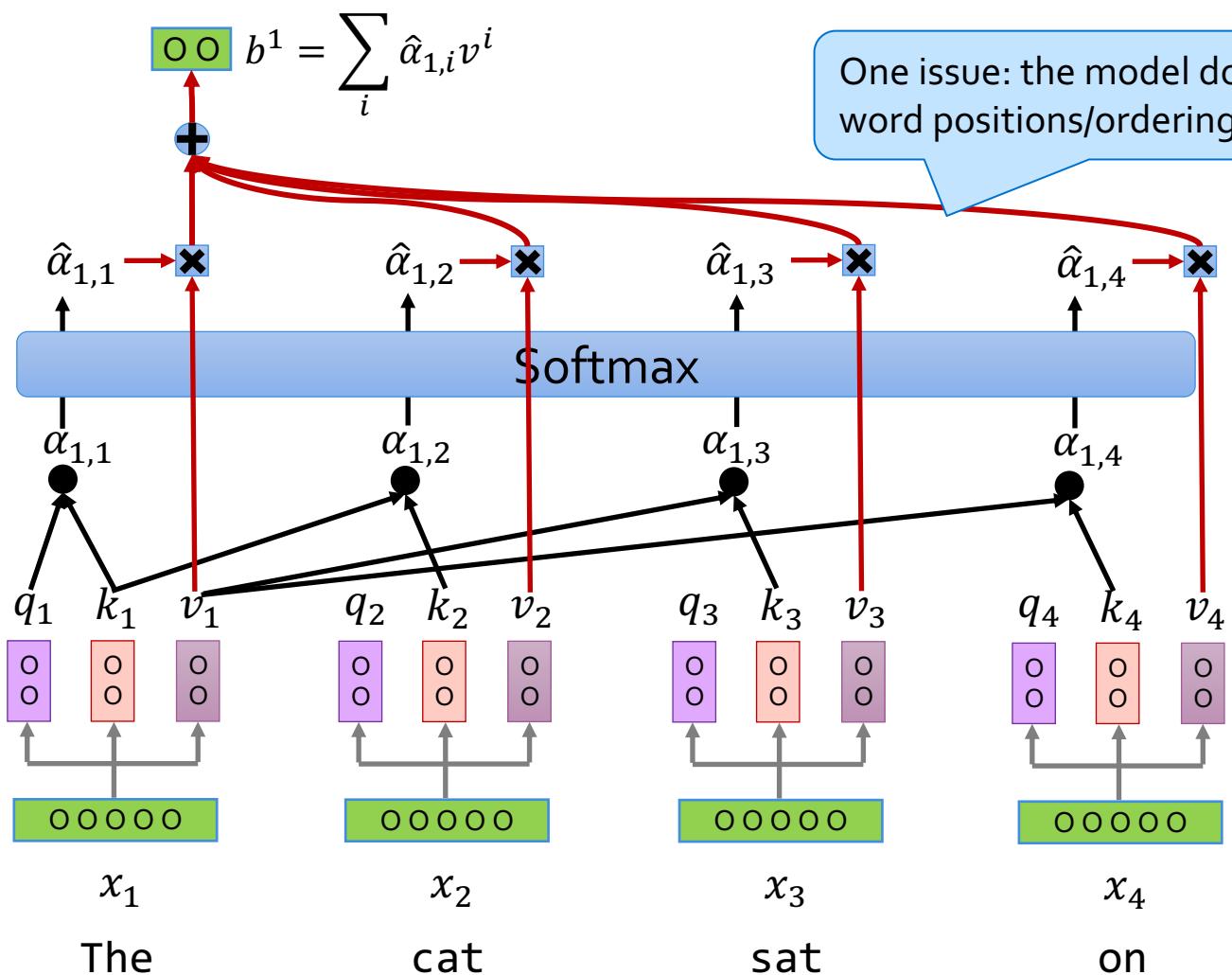


Transformer

How Do We Make it Deep?

- Stack more layers!

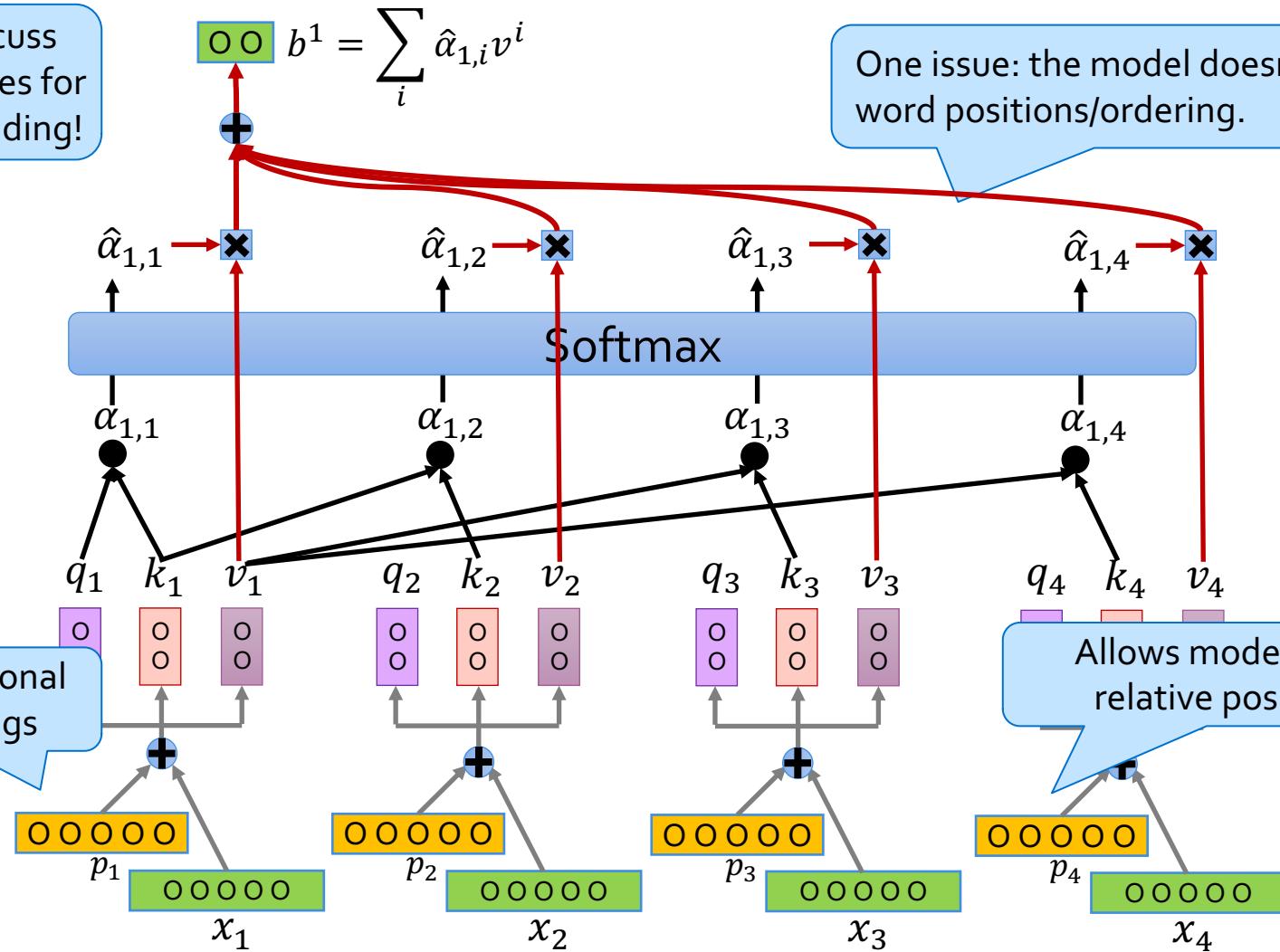




We will discuss various choices for these embedding!

$$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$$

One issue: the model doesn't know word positions/ordering.



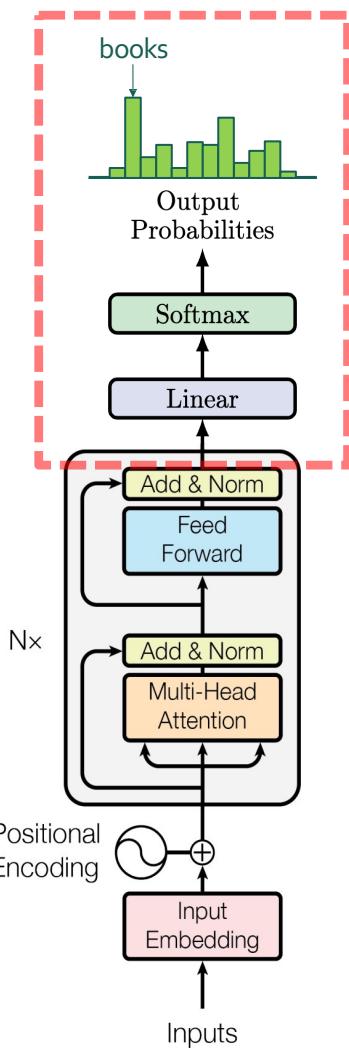
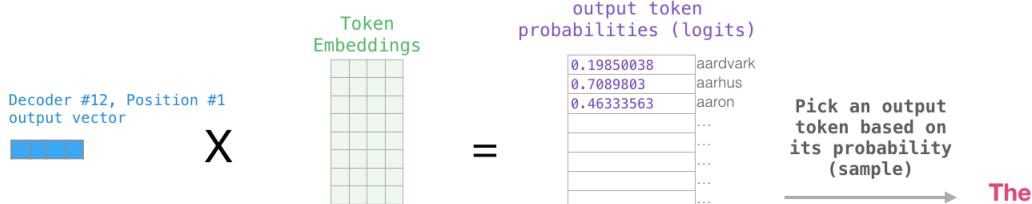
From Representations to Prediction

- 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)

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

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

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



The Transformer Stack in PyTorch

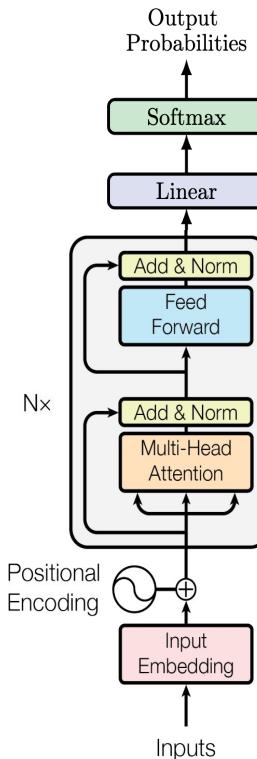


```
class Block(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.ln_1 = LayerNorm(config.n_embd, bias=config.bias)
        self.attn = CausalSelfAttention(config)
        self.ln_2 = LayerNorm(config.n_embd, bias=config.bias)
        self.mlp = MLP(config)

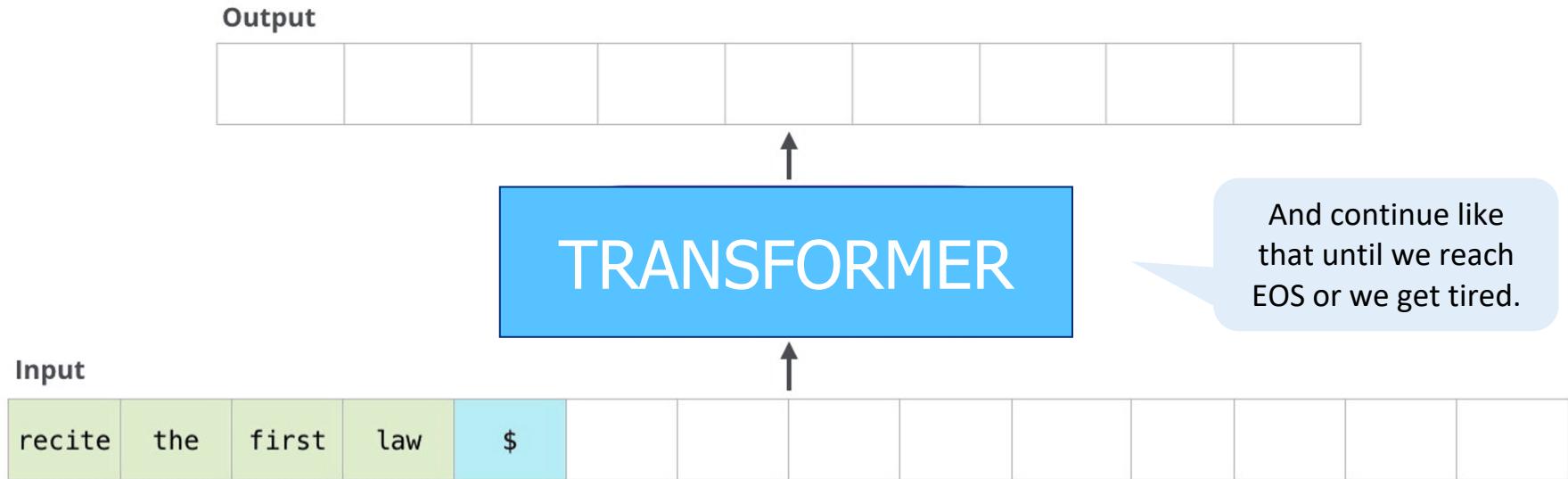
    def forward(self, x):
        x = x + self.attn(self.ln_1(x))
        x = x + self.mlp(self.ln_2(x))
        return x

    self.transformer = nn.ModuleDict(
        dict(
            wte=nn.Embedding(config.vocab_size, config.n_embd),
            wpe=nn.Embedding(config.block_size, config.n_embd),
            drop=nn.Dropout(config.dropout),
            h=nn.ModuleList([Block(config) for _ in range(config.n_layer)]),
            ln_f=LayerNorm(config.n_embd, bias=config.bias),
        )
    )

    self.lm_head = nn.Linear(config.n_embd, config.vocab_size, bias=False)
```



Transformer-based Language Modeling

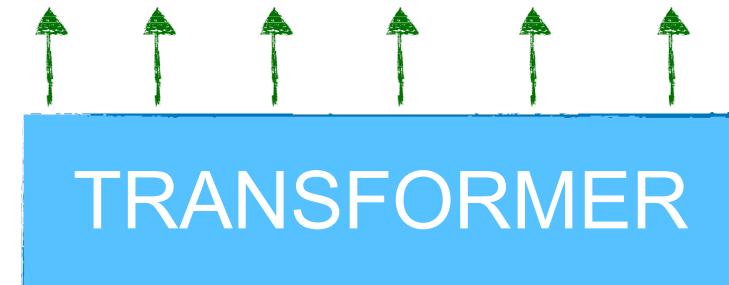


Training a Transformer Language Model

- **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

(gold output) $Y = \text{cat sat on the mat } </s>$

EOS special token

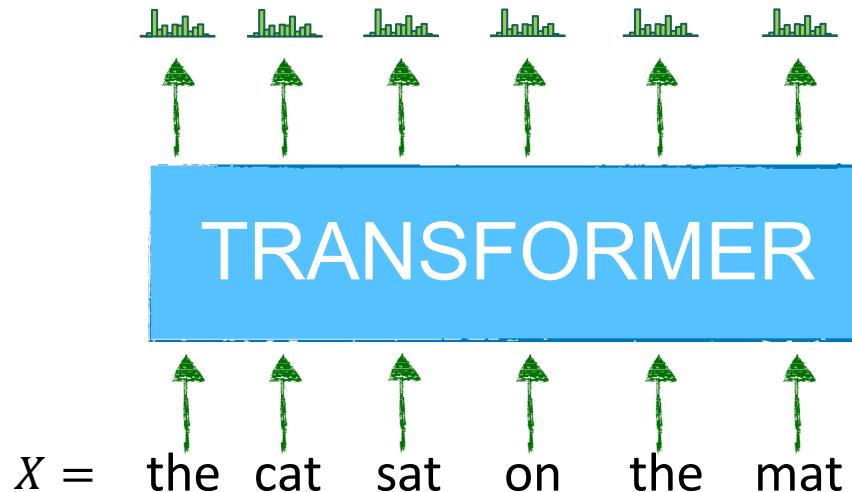


```
X = text[:, :-1]  
Y = text[:, 1:]
```

Training a Transformer Language Model

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

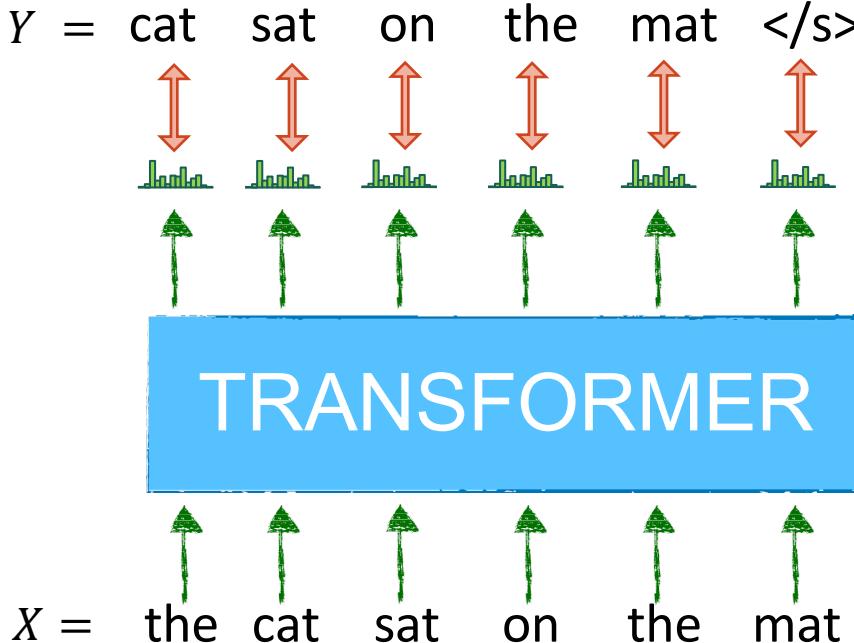
(gold output) $Y = \text{cat sat on the mat } </s>$



Training a Transformer Language Model

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

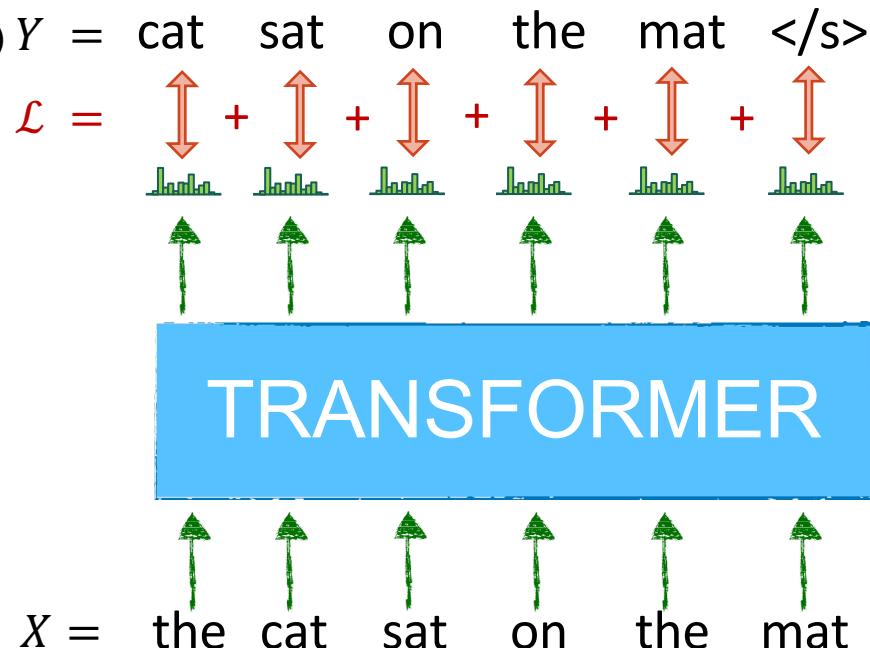
(gold output) $Y = \text{cat sat on the mat } </s>$



Training a Transformer Language Model

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

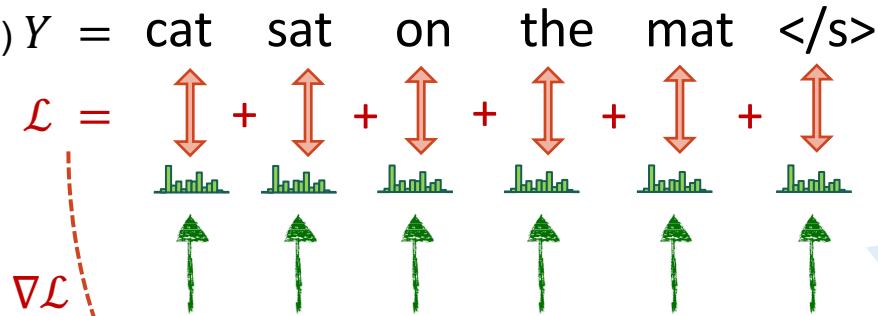
(gold output) $Y = \text{cat sat on the mat } </s>$



Training a Transformer Language Model

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

(gold output) $Y = \text{cat sat on the mat } </s>$



$\nabla \mathcal{L}$

TRANSFORMER

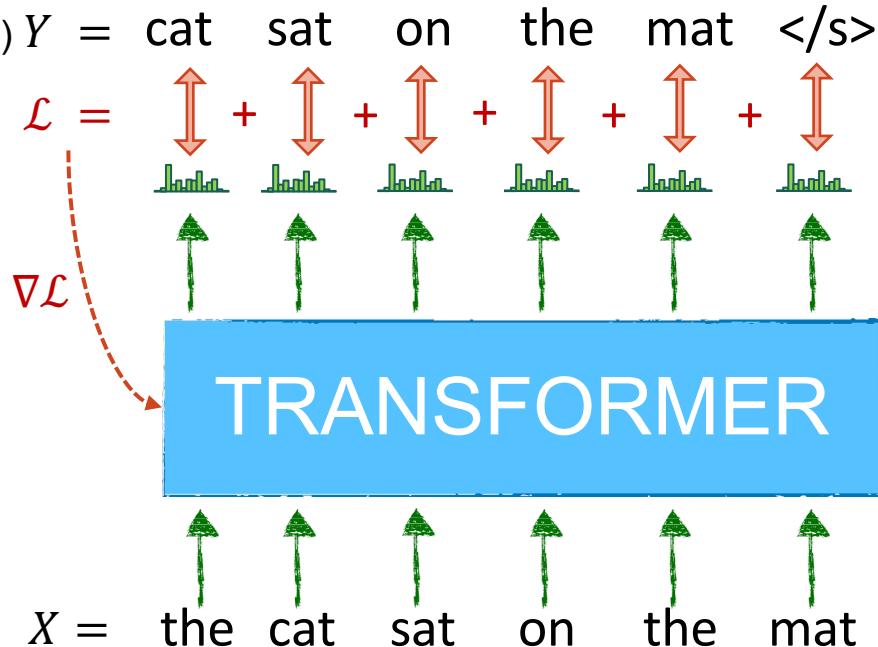
$X = \text{the cat sat on the mat}$

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

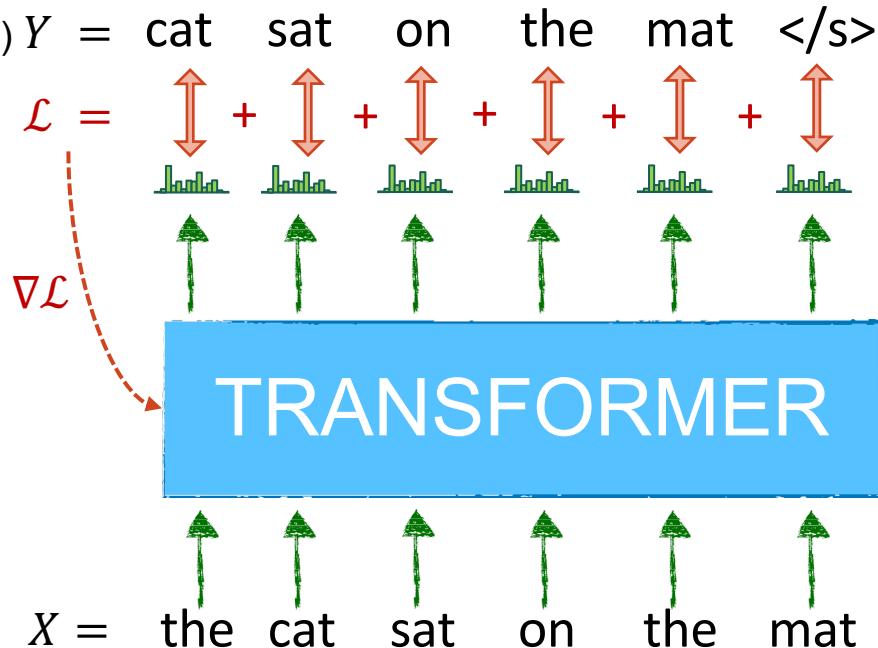
(gold output) $Y = \text{cat sat on the mat } </s>$



Training a Transformer Language Model

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

(gold output) $Y = \text{cat sat on the mat } </s>$



Attention mask

Attention raw scores

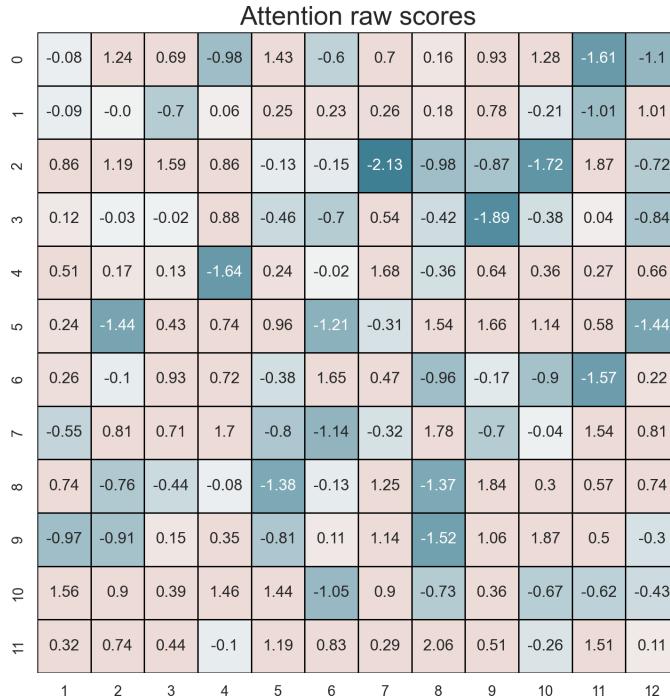
0	-0.08	1.24	0.69	-0.98	1.43	-0.6	0.7	0.16	0.93	1.28	-1.61	-1.1
1	-0.09	-0.0	-0.7	0.06	0.25	0.23	0.26	0.18	0.78	-0.21	-1.01	1.01
2	0.86	1.19	1.59	0.86	-0.13	-0.15	-2.13	-0.98	-0.87	-1.72	1.87	-0.72
3	0.12	-0.03	-0.02	0.88	-0.46	-0.7	0.54	-0.42	-1.89	-0.38	0.04	-0.84
4	0.51	0.17	0.13	-1.64	0.24	-0.02	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												
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



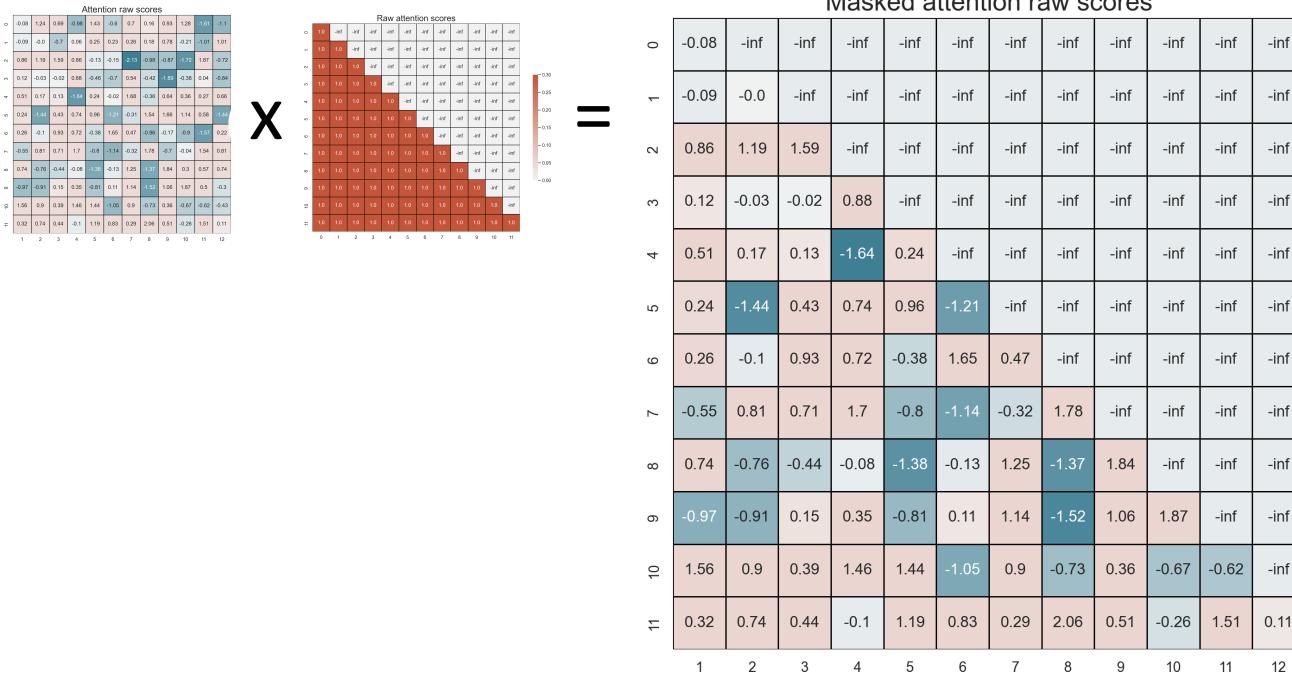
Attention mask



X

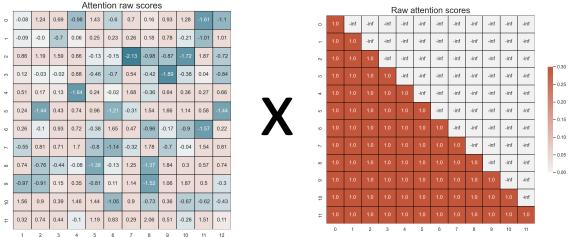


Attention mask



Attention mask

The effect is not pruning out some of the wirings in self-attention block.
Note matrix multiplication is quite fast in GPUs.



Masked attention raw scores											
0.08	-0.01	0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01
-0.09	-0.0	0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01
0.09	1.19	1.59	0.96	-0.12	-0.17	1.11	0.03	-0.07	1.72	1.07	-0.72
-0.12	-0.03	0.02	0.08	-0.46	-0.17	0.54	-0.42	1.81	-0.28	0.04	-0.86
0.51	0.17	0.15	0.24	0.24	-0.03	1.98	0.36	0.64	0.27	0.07	-0.27
0.28	-1.44	0.03	0.74	0.96	-1.2	-0.31	1.54	1.81	1.14	0.58	-1.64
0.26	-0.1	0.84	0.72	0.08	1.05	0.47	0.96	-0.17	0.5	1.01	0.22
-0.05	0.81	0.71	1.7	0.18	-0.22	1.78	-0.17	-0.54	1.54	0.81	-0.05
0.74	0.76	-0.04	0.08	-0.32	-0.12	1.25	0.37	1.84	0.3	0.57	0.74
0.37	0.91	0.14	0.35	1.46	-0.01	1.14	-1.32	1.08	1.87	0.2	-0.2
0.16	0.9	0.39	1.46	1.44	-1.05	0.9	0.73	0.36	-0.67	0.62	-0.42
0.02	0.74	0.44	0.1	1.19	0.09	0.29	2.06	0.93	-0.28	1.51	0.11

softmax



Implementation

```
def multiquery_attention_batch(X, M, proj_q, proj_k, proj_v, proj_o):
    """
    Args:
        X: a tensor with shape [b, n, d]
        M: a tensor with shape [b, m, d]
        mask: a tensor with shape [b, h, n, m]
        P_q: a tensor with shape [h, d, k]
        P_k: a tensor with shape [d, k]
        P_v: a tensor with shape [d, v]
        P_o: a tensor with shape [h, d, v]
    Returns:
        Y: a tensor with shape [b, n, d]
    """
    Q = torch.einsum("bnd,hdk->bhnk", X, proj_q)
    K = torch.einsum("bmd,dk->bmk", M, proj_k)
    V = torch.einsum("bmd,dv->bmv", M, proj_v)
    logits = torch.einsum("bhnk,bmk->bhnm", Q, K)
    weights = torch.softmax(logits, dim=-1)
    O = torch.einsum("bhnm,bmv->bhnv", weights, V)
    Y = torch.einsum("bhnv,hdv->bnd", O, proj_o)
    return Y
```

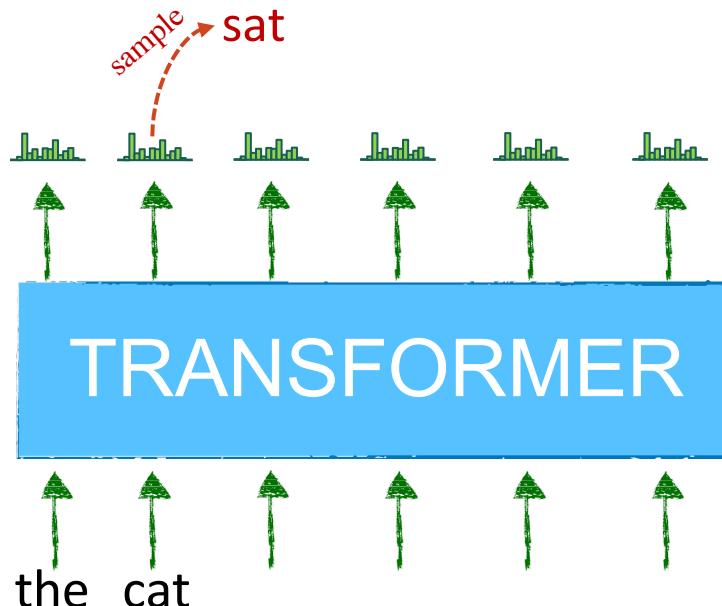
Implementation

```
def multiquery_attention_batch(X, M, proj_q, proj_k, proj_v):
    """
    Args:
        X: a tensor with shape [b, n, d]
        M: a tensor with shape [b, m, d]
        mask: a tensor with shape [b, h, n, m]
        P_q: a tensor with shape [h, d, k]
        P_k: a tensor with shape [d, k]
        P_v: a tensor with shape [d, v]
        P_o: a tensor with shape [h, d, v]
    Returns:
        Y: a tensor with shape [b, n, d]
    """
    Q = torch.einsum("bnd,hdk->bhnk", X, proj_q)
    K = torch.einsum("bmd,dk->bmk", M, proj_k)
    V = torch.einsum("bmd,dv->bmv", M, proj_v)
    logits = torch.einsum("bhnk,bmk->bhnm", Q, K)
    weights = torch.softmax(logits, dim=-1)
    O = torch.einsum("bhnm,bmv->bhnv", weights, V)
    Y = torch.einsum("bhnv,hdv->bnd", O, proj_o)
    return Y
```

```
def causal_multihead_attention_batch(X, M, proj_q, proj_k, proj_v, proj_o):
    """
    Args:
        X: a tensor with shape [b, n, d]
        M: a tensor with shape [b, m, d]
        P_q: a tensor with shape [h, d, k]
        P_k: a tensor with shape [h, d, k]
        P_v: a tensor with shape [h, d, v]
        P_o: a tensor with shape [h, d, v]
    Returns:
        Y: a tensor with shape [b, n, d]
    """
    q_len, v_len = X.shape[1], M.shape[1]
    # mask: a tensor with shape [b, h, n, m]
    mask = torch.tril(torch.ones(q_len, v_len)).view(
        1, 1, q_len, v_len)
    Q = torch.einsum("bnd,hdk->bhnk", X, proj_q)
    K = torch.einsum("bmd,dk->bmhmk", M, proj_k)
    V = torch.einsum("bmd,dv->bmhv", M, proj_v)
    logits = torch.einsum("bhnk,bmhmk->bhnm", Q, K)
    logits = logits.masked_fill(mask == 0, float("-inf"))
    weights = torch.softmax(logits, dim=-1)
    O = torch.einsum("bhnm,bhmhv->bhnv", weights, V)
    Y = torch.einsum("bhnv,hdv->bnd", O, proj_o)
    return Y
```

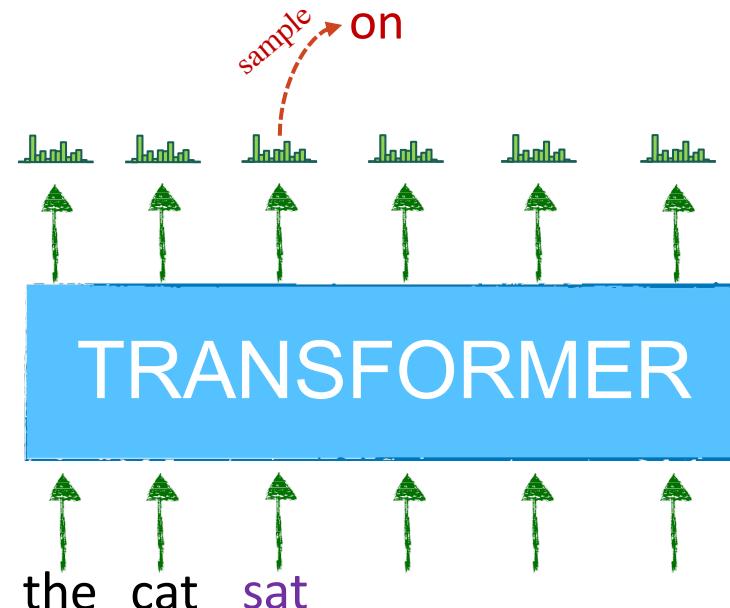
How to use the model to generate text?

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



How to use the model to generate text?

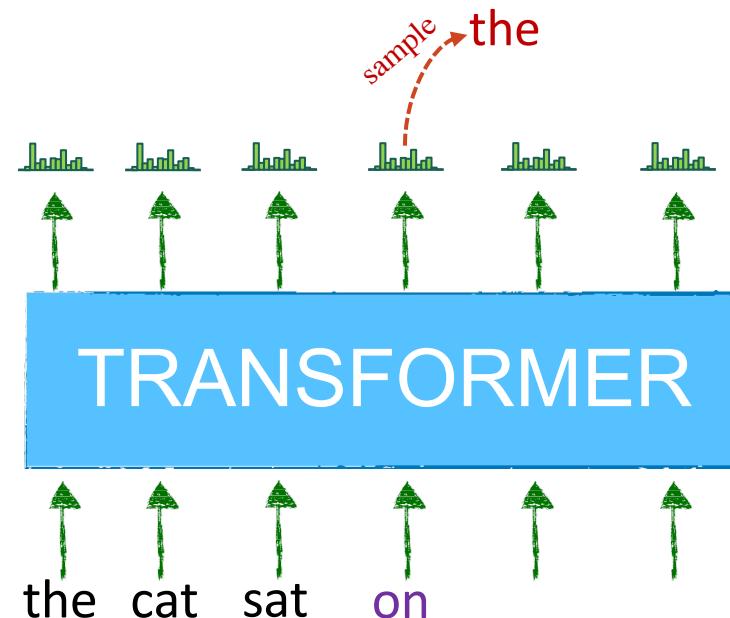
- Use the output of previous step as input to the next step repeatedly



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

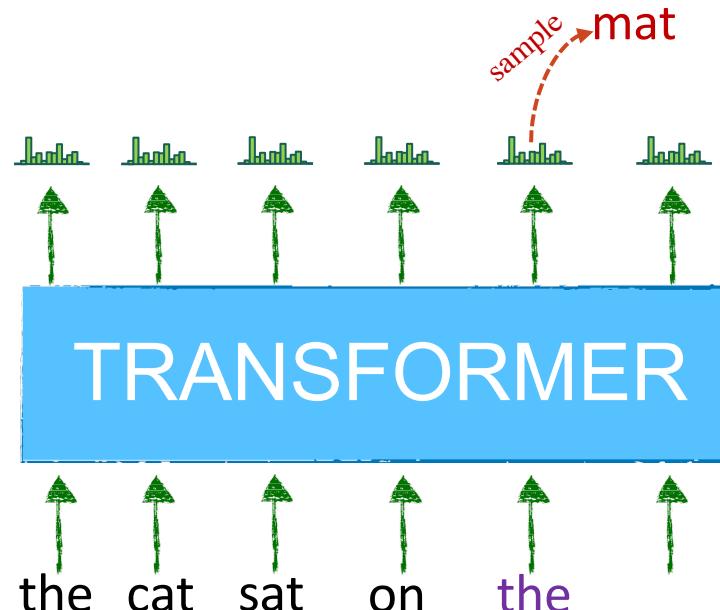


The probabilities get revised upon adding a new token to the input.

How to use the model to generate text?

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

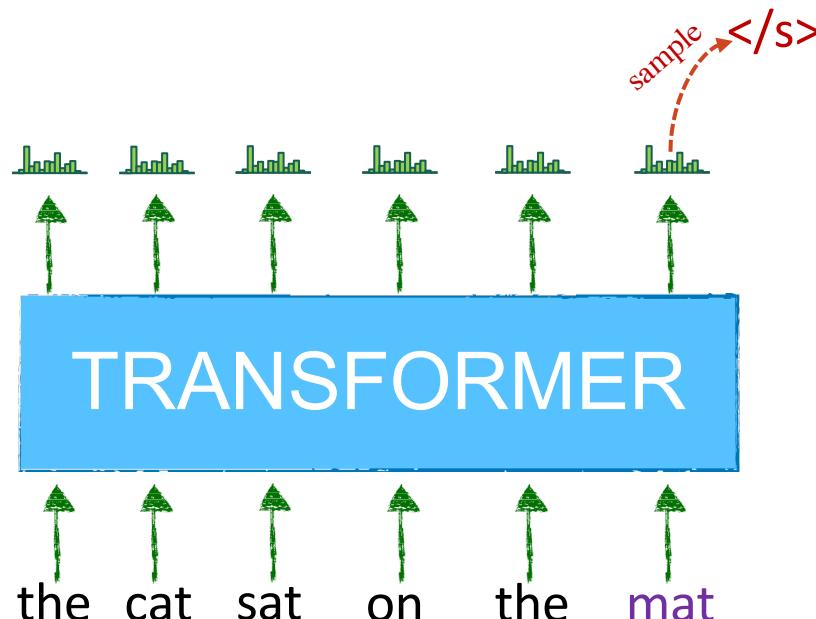
The probabilities get revised upon adding a new token to the input.



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.



Summary

- What we say was a very generic Transformer
- **Next:**
- **After that:**



Transformer: Efficiency considerations

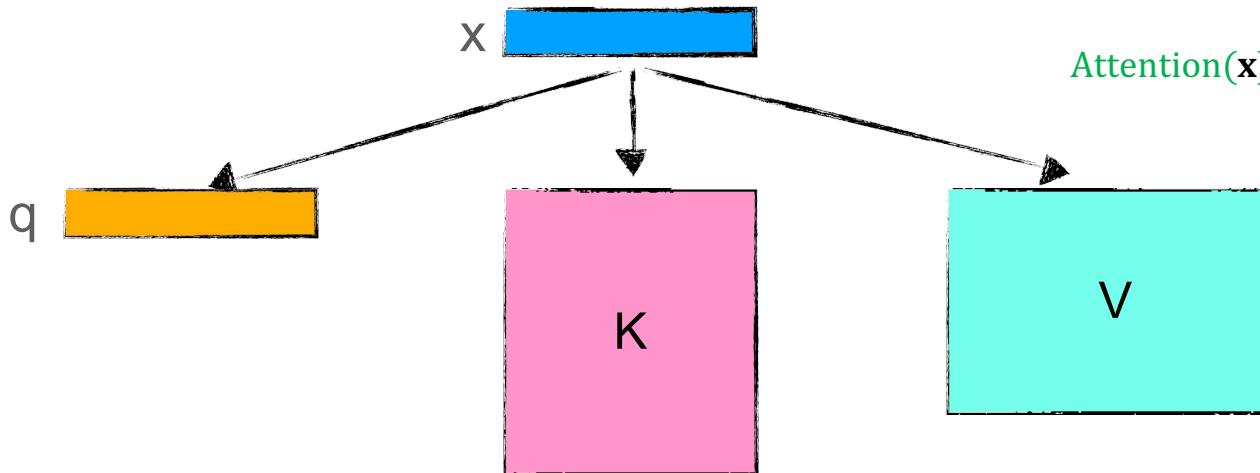
Making decoding more efficient

$$Q = W^q x$$

$$K = W^k x$$

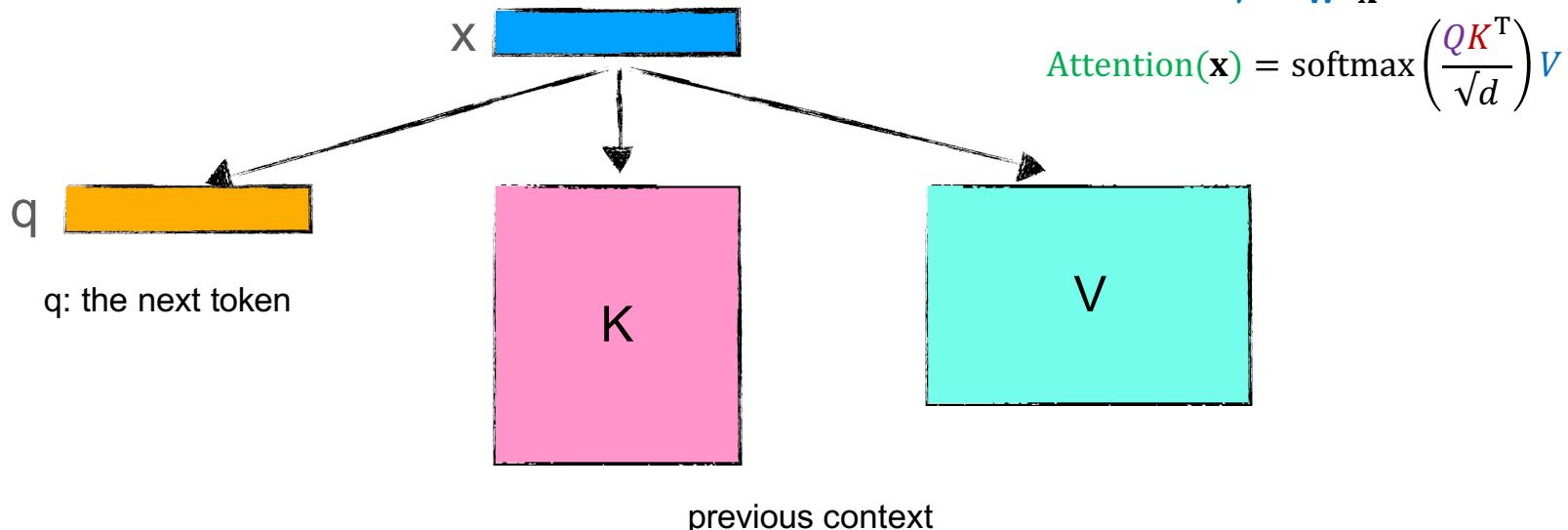
$$V = W^v x$$

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



[Slide credit: Arman Cohan]

Making decoding more efficient



[Slide credit: Arman Cohan]

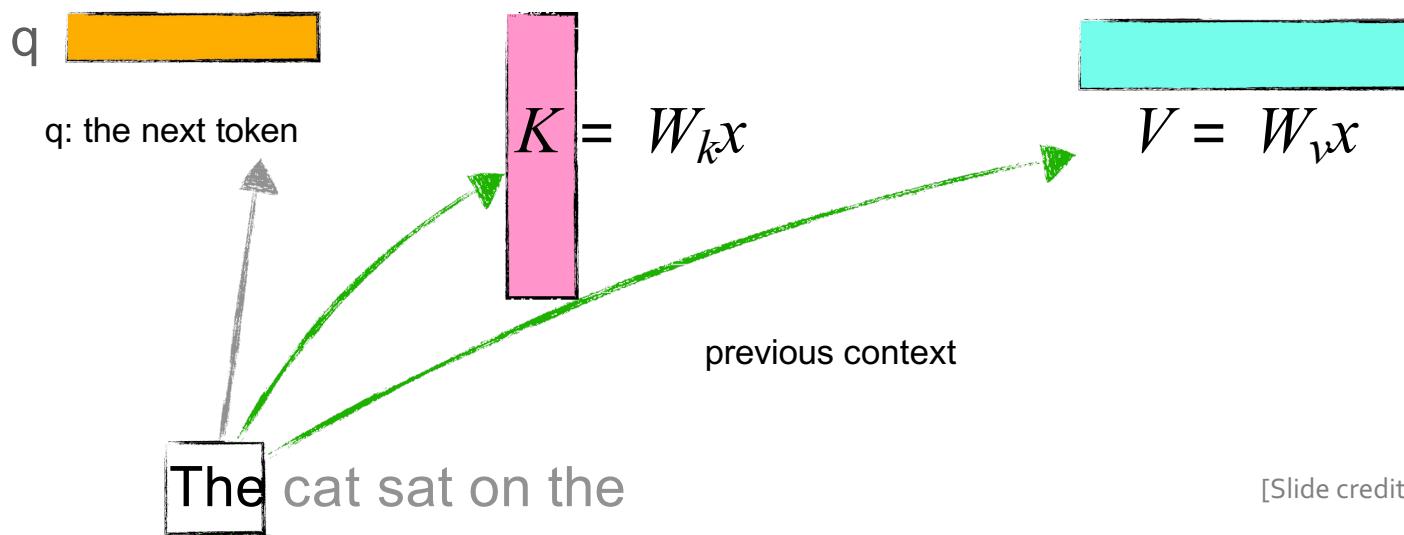
Making decoding more efficient

$$Q = W^q \mathbf{x}$$

$$K = W^k \mathbf{x}$$

$$V = W^v \mathbf{x}$$

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



[Slide credit: Arman Cohan]

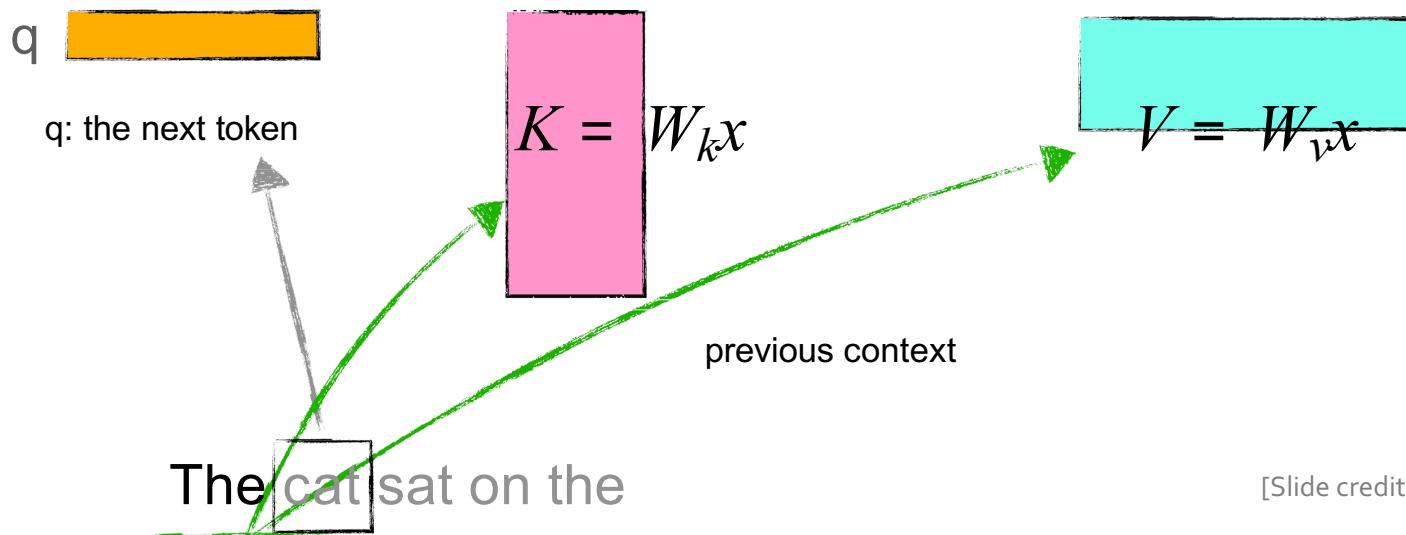
Making decoding more efficient

$$Q = W^q \mathbf{x}$$

$$K = W^k \mathbf{x}$$

$$V = W^v \mathbf{x}$$

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



[Slide credit: Arman Cohan]

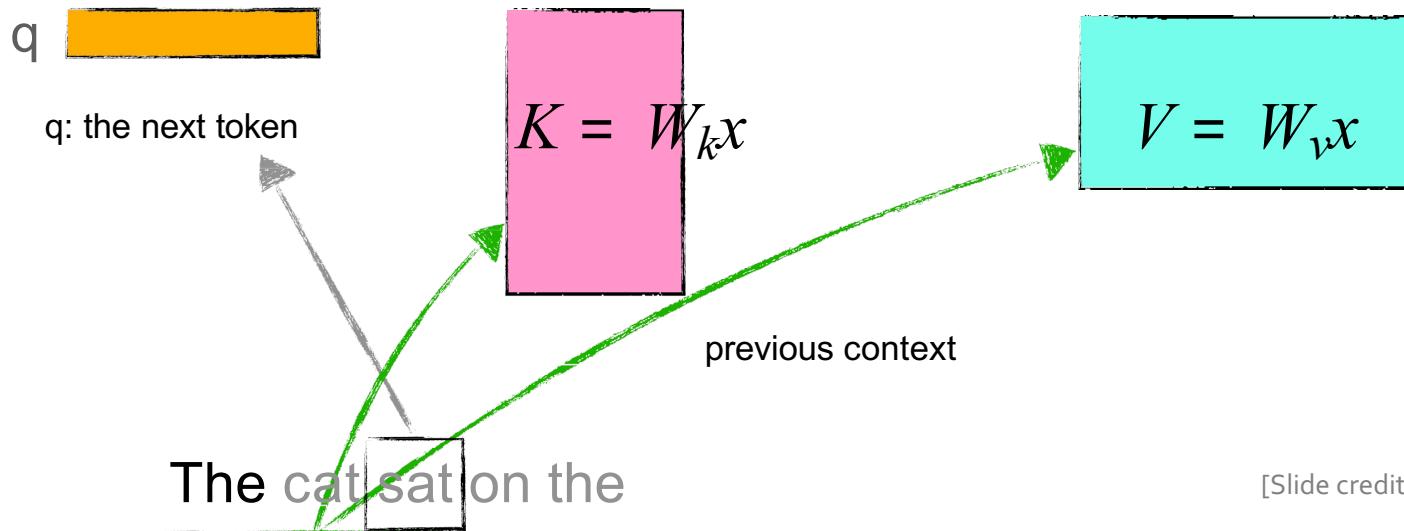
Making decoding more efficient

$$Q = W^q \mathbf{x}$$

$$K = W^k \mathbf{x}$$

$$V = W^v \mathbf{x}$$

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



[Slide credit: Arman Cohan]

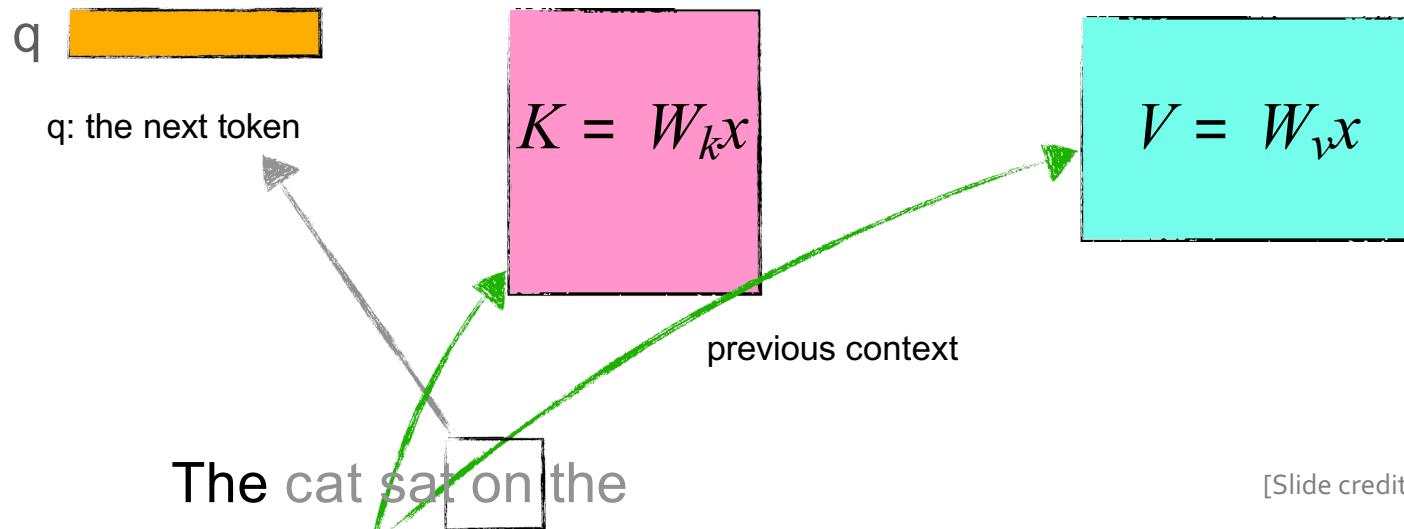
Making decoding more efficient

$$Q = \mathbf{W}^q \mathbf{x}$$

$$K = \mathbf{W}^k \mathbf{x}$$

$$V = \mathbf{W}^v \mathbf{x}$$

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



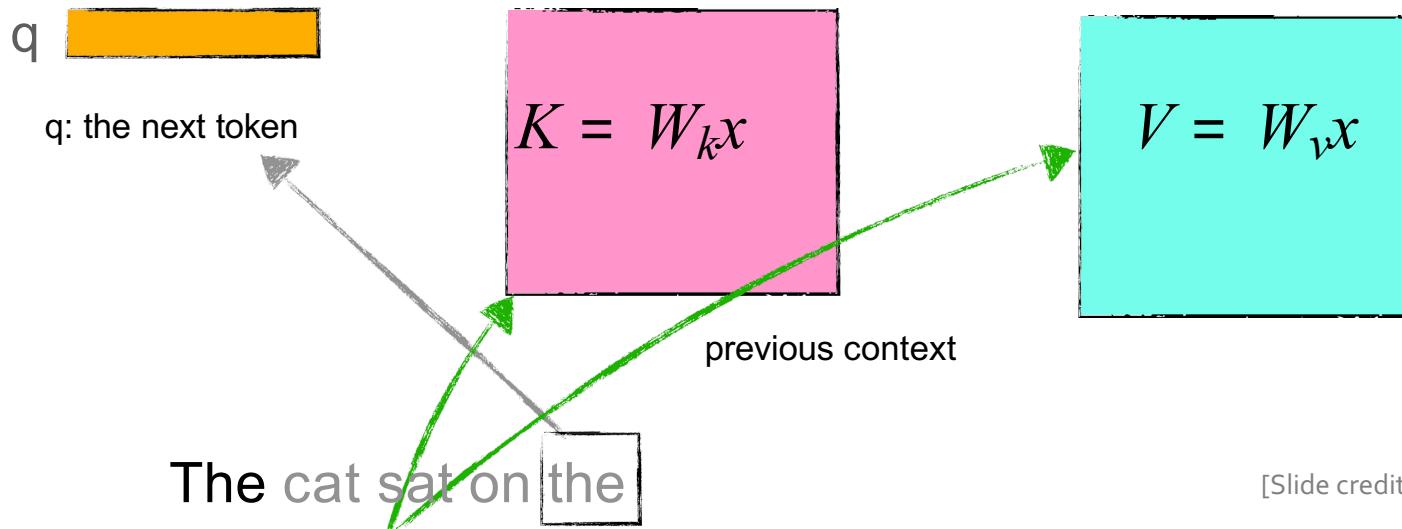
Making decoding more efficient

$$Q = W^q \mathbf{x}$$

$$K = W^k \mathbf{x}$$

$$V = W^v \mathbf{x}$$

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



[Slide credit: Arman Cohan]

Making decoding more efficient

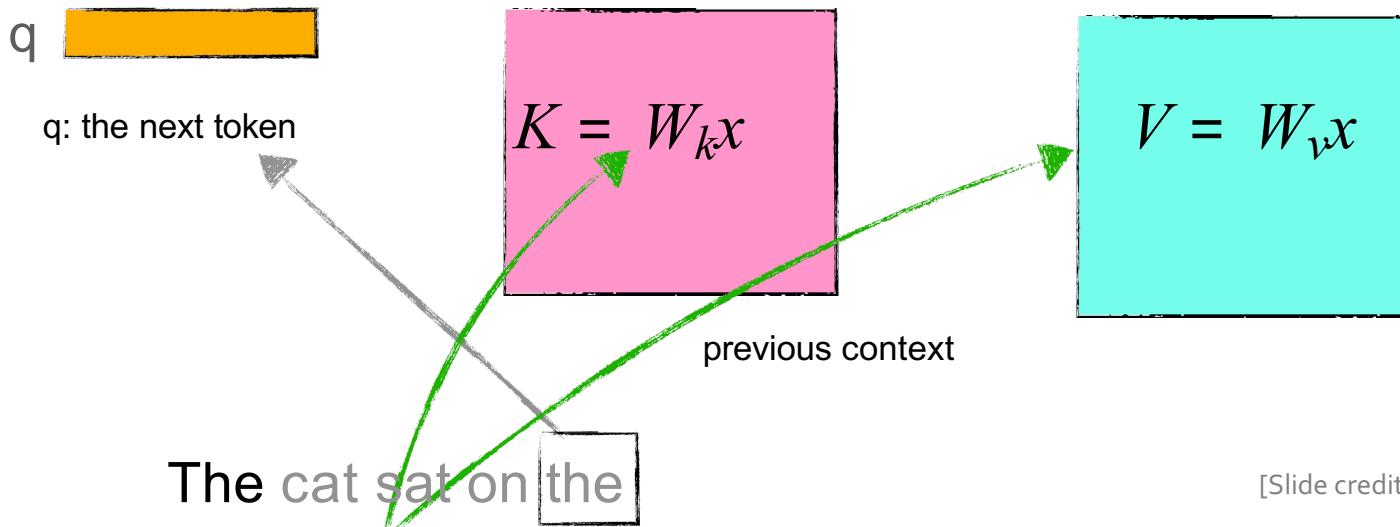
- We are computing the Keys and Values many times!
 - Let's reduce redundancy! 😱

$$Q = W^q \mathbf{x}$$

$$K = W^k \mathbf{x}$$

$$V = W^v \mathbf{x}$$

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



[Slide credit: Arman Cohan]

Making decoding more efficient

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

$$Q = \mathbf{W}^q \mathbf{x}$$

$$K = \mathbf{W}^k \mathbf{x}$$

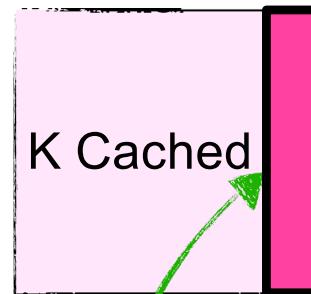
$$V = \mathbf{W}^v \mathbf{x}$$

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

$$k_{new} = W_k \mathbf{x}[:, : - 1]$$



q: the next token



previous context



$$v_{new} = W_v \mathbf{x}[:, : - 1]$$

The cat sat on the

[Slide credit: Arman Cohan]

Making decoding more efficient

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

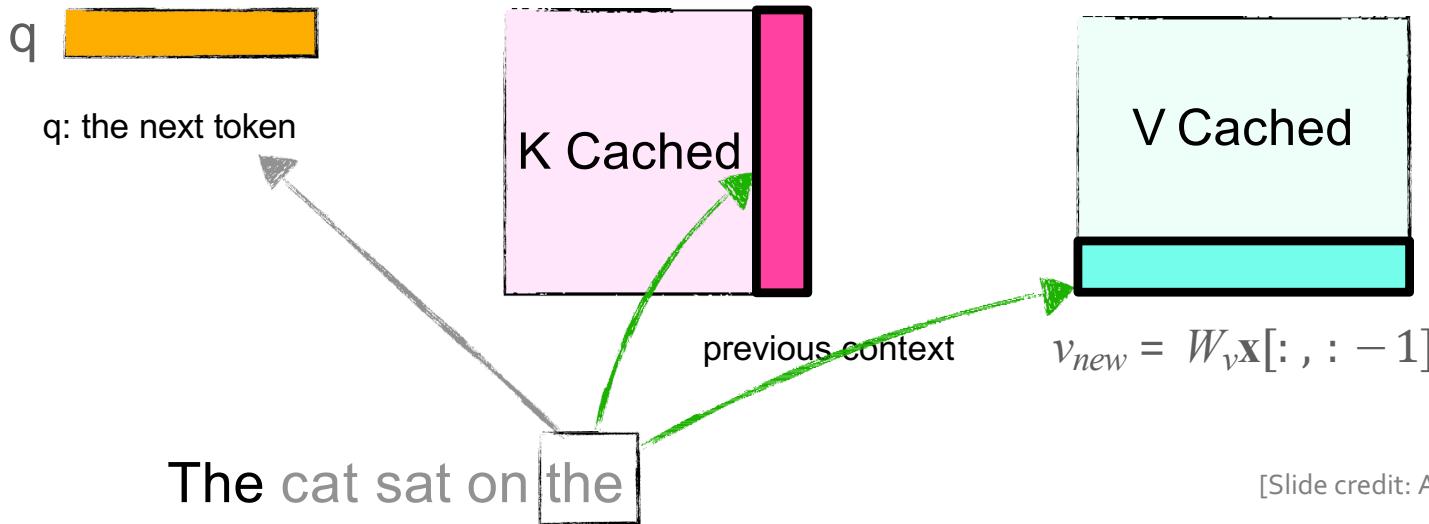
$$Q = W^q \mathbf{x}$$

$$K = W^k \mathbf{x}$$

$$V = W^v \mathbf{x}$$

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

$$k_{new} = W_k \mathbf{x}[:, : - 1]$$



[Slide credit: Arman Cohan]

MLP Layers

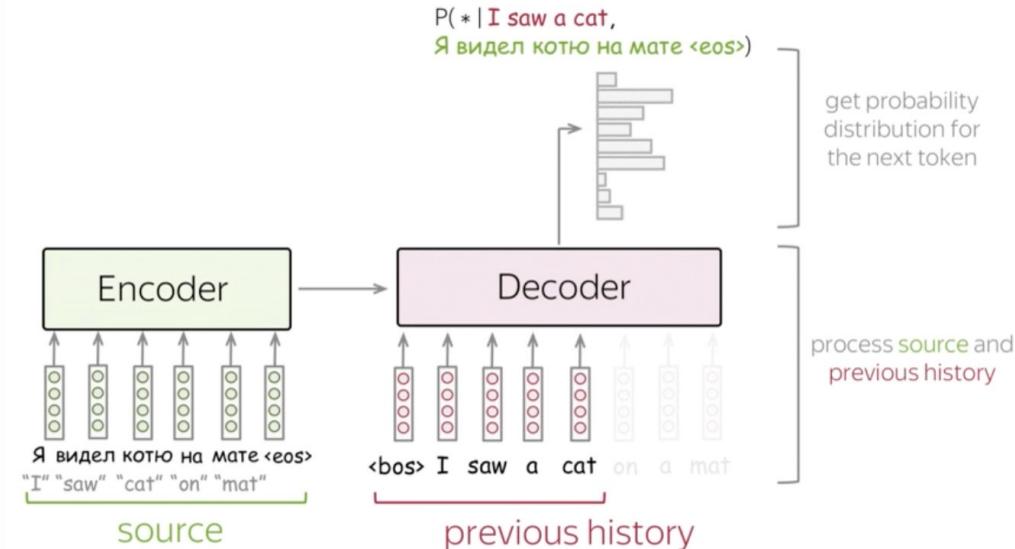
- Transformers are not just the attention layers!
- For LLMs, it has been shown that a lot of the information is encoded in the feed forward layers: “key-value memories”.
- TODO: Mor’s work.



Transformer Architectural Variants

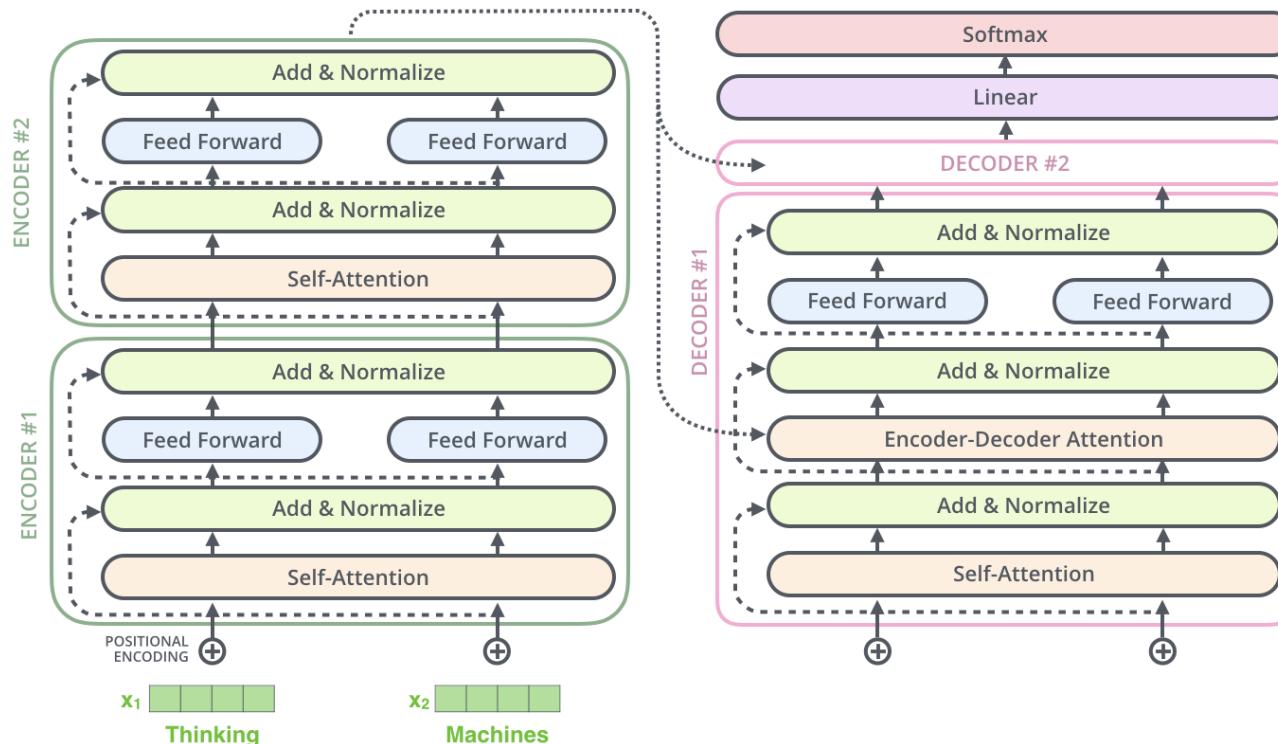
Encoder-decoder

- It is possible to have two stacks of transformer layers
- The encoder is as we've seen
- We can also add a decoder layer that is identical to the encoder but we give it the ability to also attend to the input



Transformer [Vaswani et al. 2017]

- An encoder-decoder architecture built with attention modules.

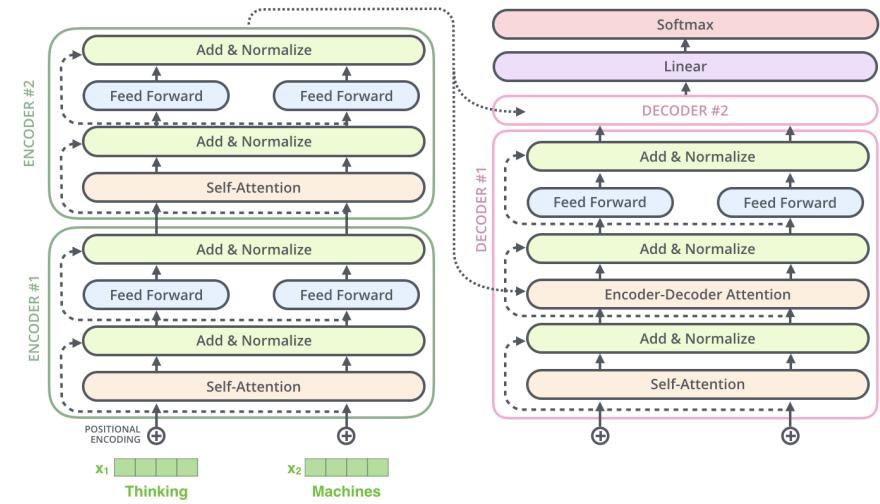


Transformer [Vaswani et al. 2017]

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

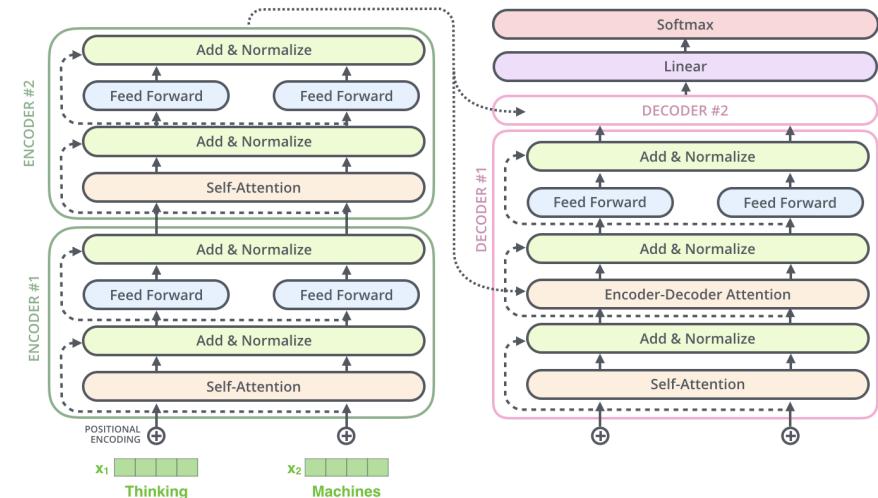
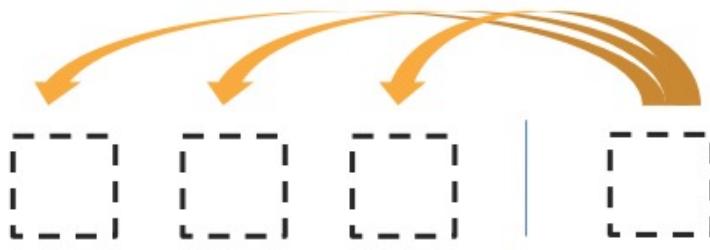


Encoder Self-Attention



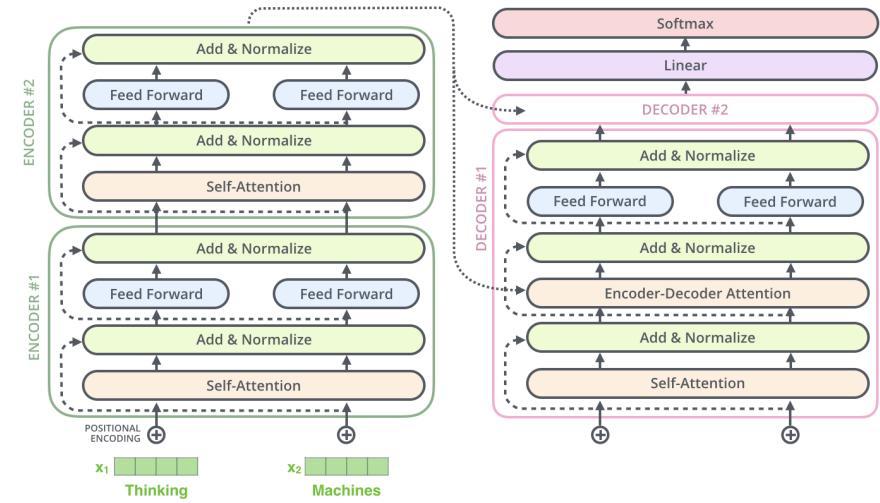
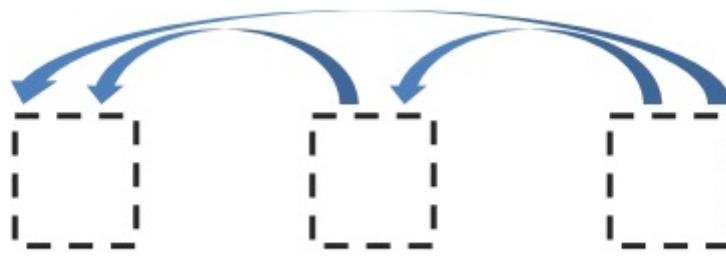
Transformer [Vaswani et al. 2017]

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



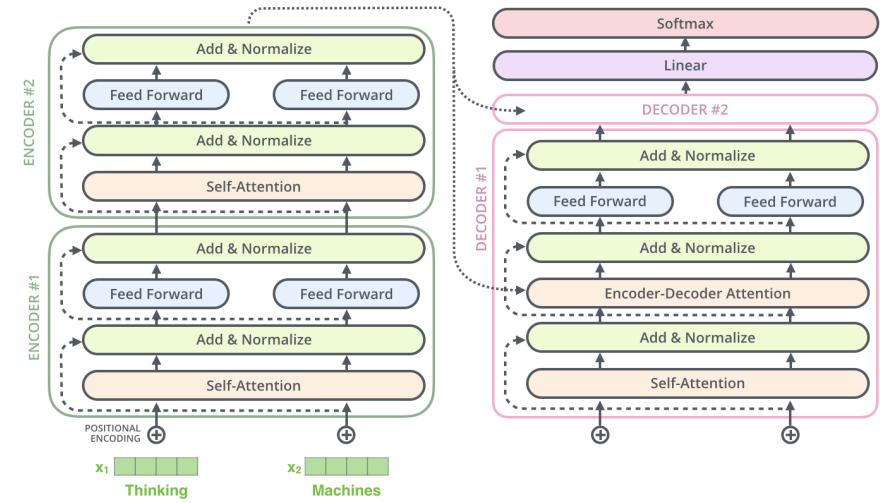
Transformer [Vaswani et al. 2017]

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



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.



Transformer encoder-decoder

Encoder-Decoder
Machine translation

$h = \text{Encoder}(\text{English})$
 $\text{French} = \text{Decoder}(h)$

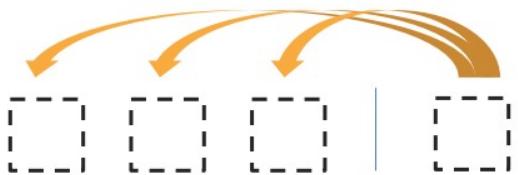
$N = \text{Enc. sequence length}$
 $M = \text{Dec. sequence length}$

$\text{Enc. Attention} = O(N^2)$
 $\text{Dec. Attention} = O(N*M + M^2)$

Transformer

[Vaswani et al. 2017]

- An **encoder-decoder** architecture
- 3 forms of attention



Encoder-Decoder Attention

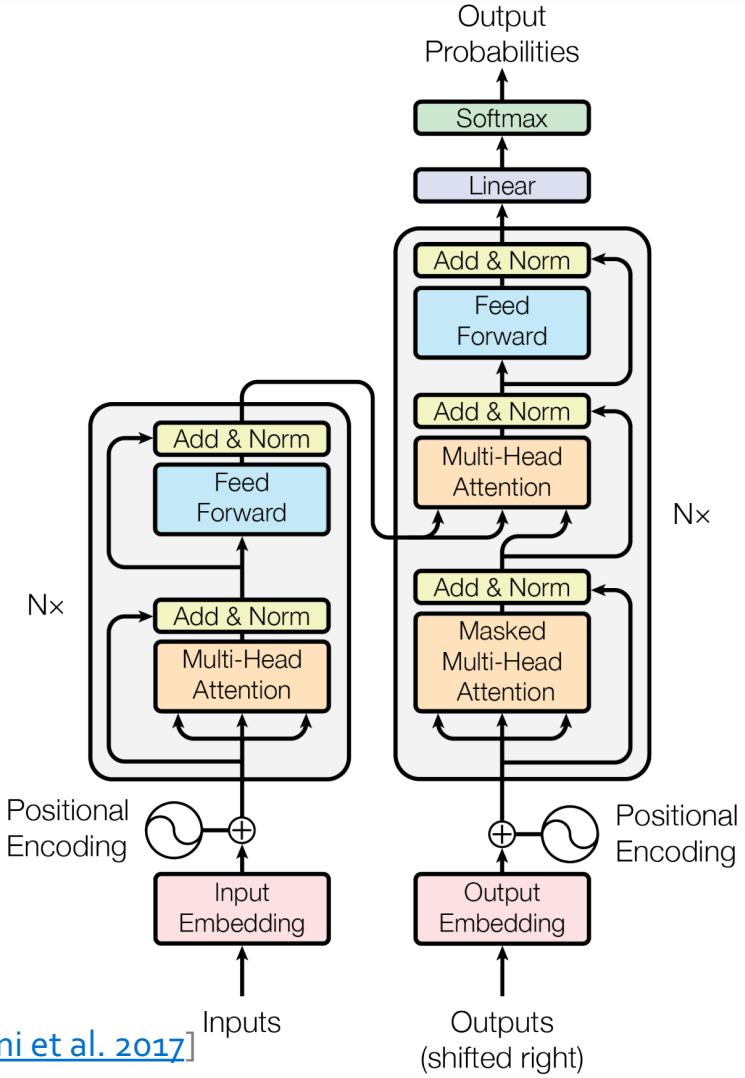


Encoder Self-Attention



Masked Decoder Self-Attention

[\[Attention Is All You Need, Vaswani et al. 2017\]](#)



Inputs

Outputs
(shifted right)

Positional
Encoding

Output
Probabilities

Nx

Positional
Encoding

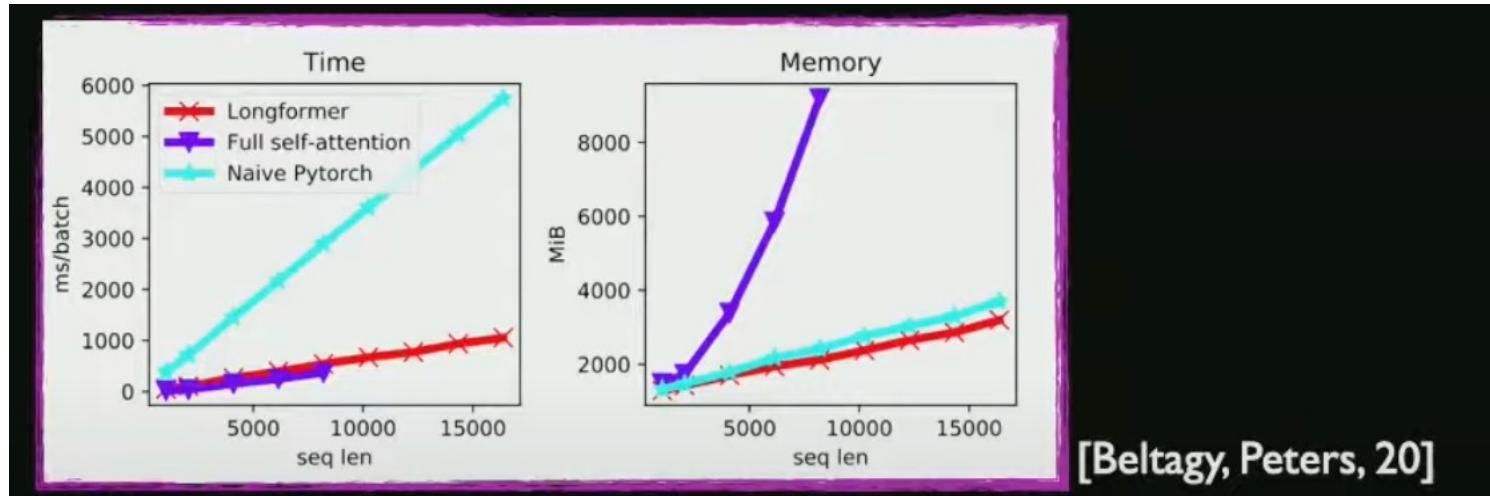
Transformers in practice

- Let to better predictive models of language!

Model	Layers	Heads	Perplexity
LSTMs (Grave et al., 2016)	-	-	40.8
QRNNs (Merity et al., 2018)	-	-	33.0
Transformer	16	16	19.8

Time/compute complexity

- Memory grows quadratically with sequence length.



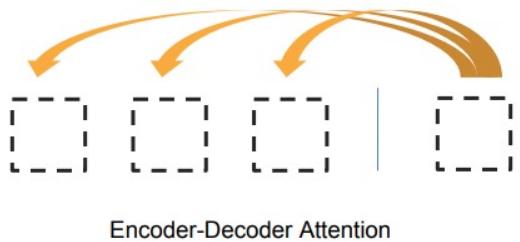
[Beltagy, Peters, 20]

Summary

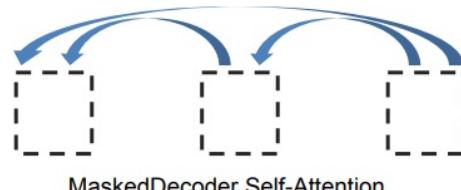
- Yaaay we know Transformers now! 😊
- More summary: TODO
- Next: extensions on Transformers.

Recap: Transformer [Vaswan

- An encoder-decoder architecture
- 3 forms of attention

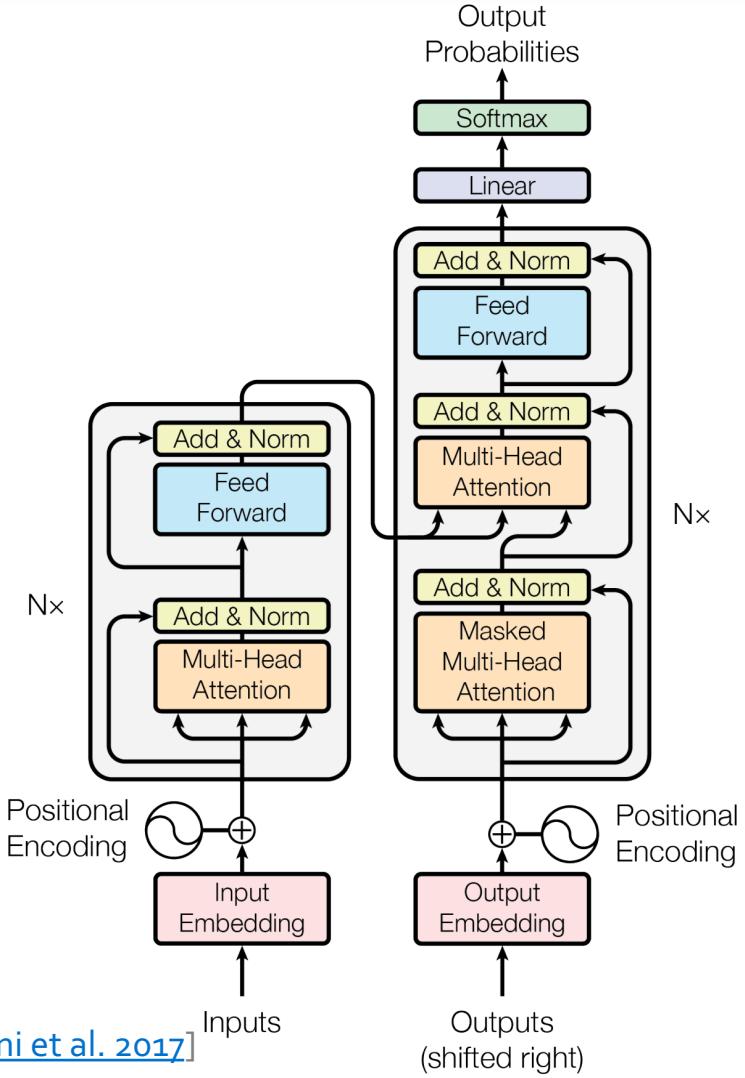


Encoder Self-Attention



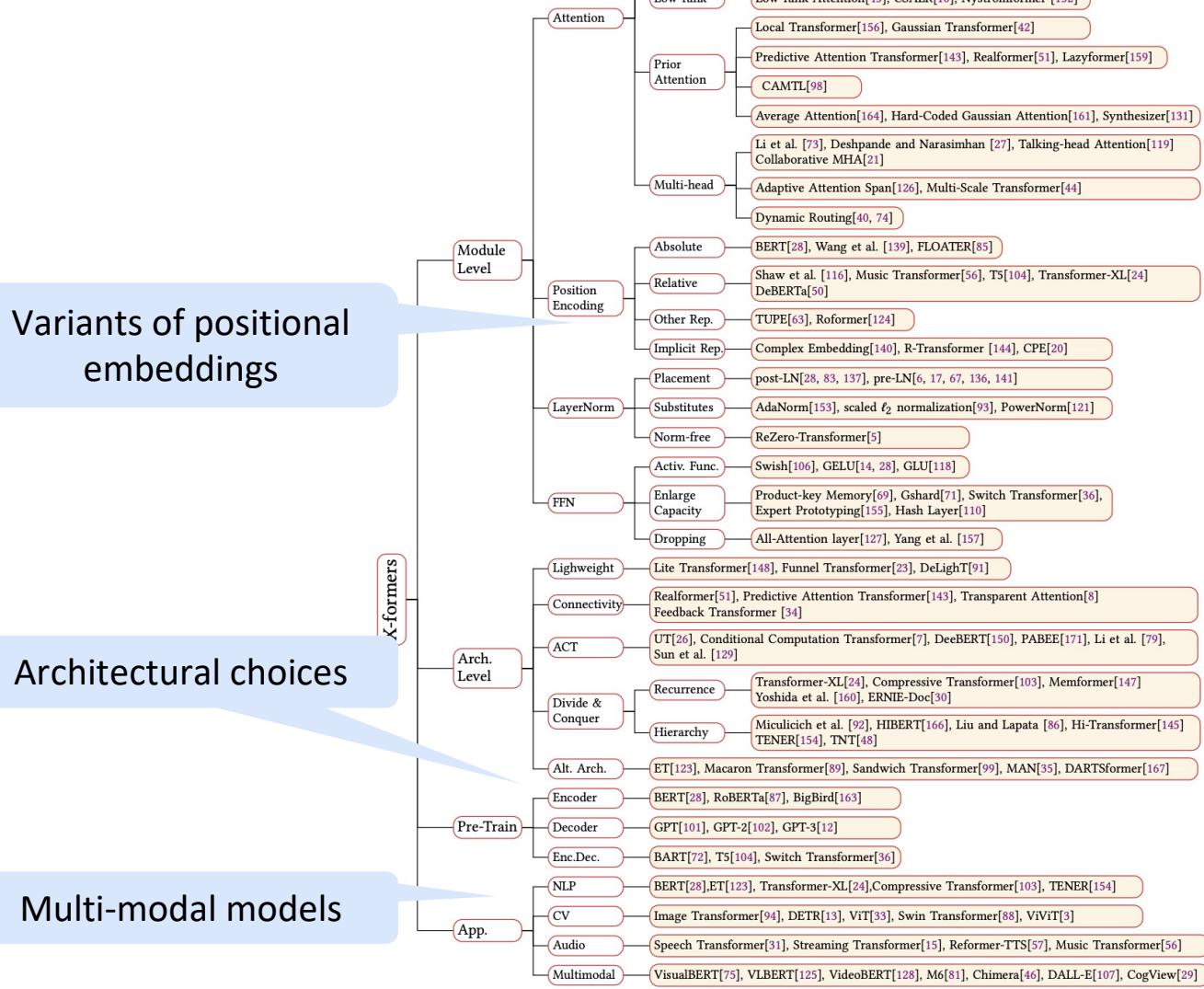
Masked Decoder Self-Attention

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



After Transformer ...





We will visit a few of
these branches ...

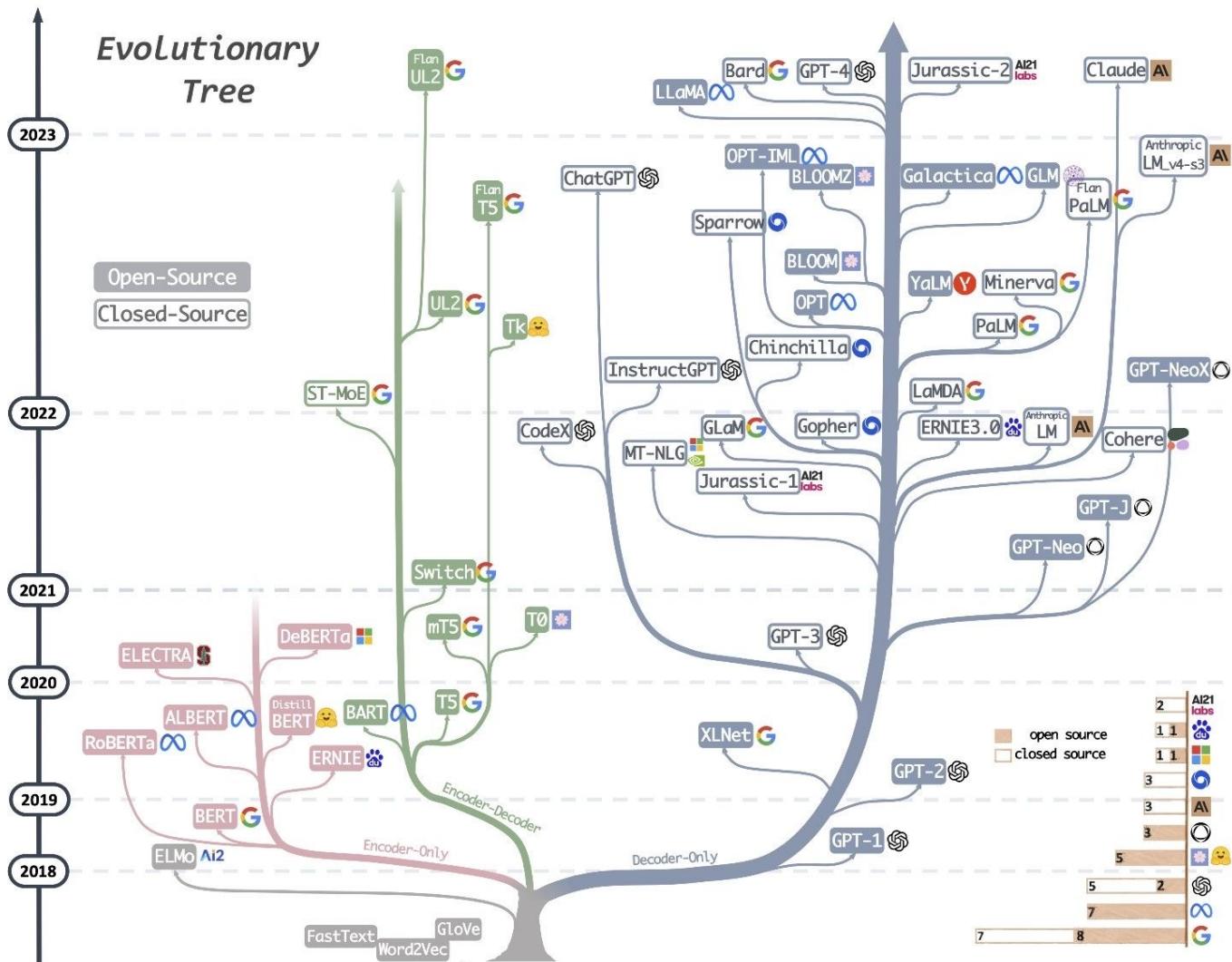
But there is a lot
that we do **not**
cover ...

Variants of positional embeddings

Architectural choices

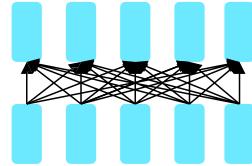
Multi-modal models

Evolutionary Tree



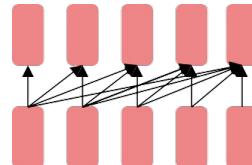
Impact of Transformers

- A building block for a variety of LMs



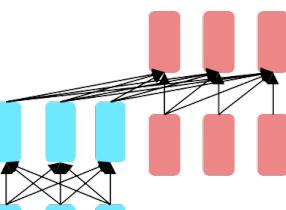
Encoders

- ❖ 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

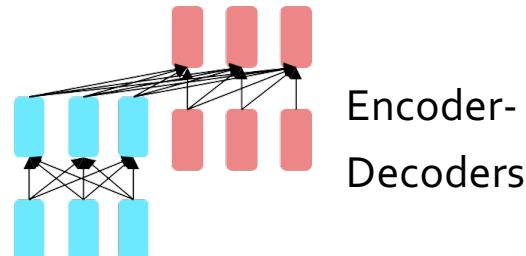


Encoder-
Decoders

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

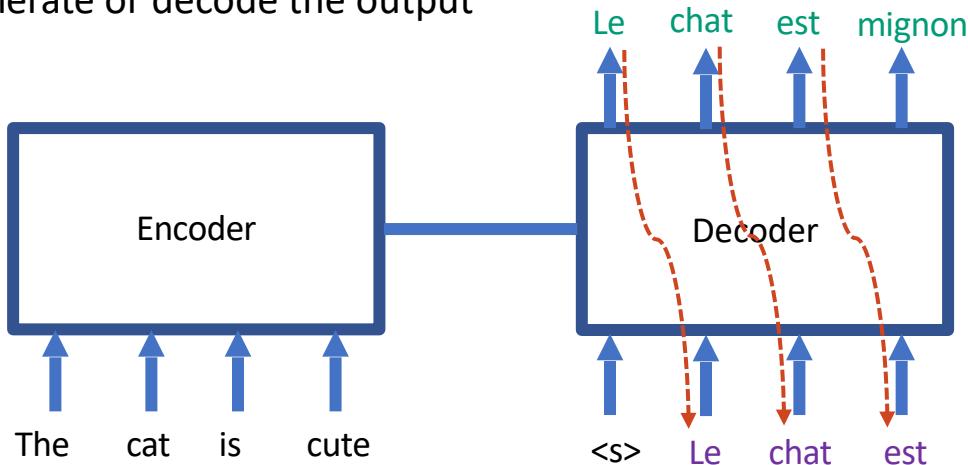
Summary

Encoder-Decoder Family of Transformers



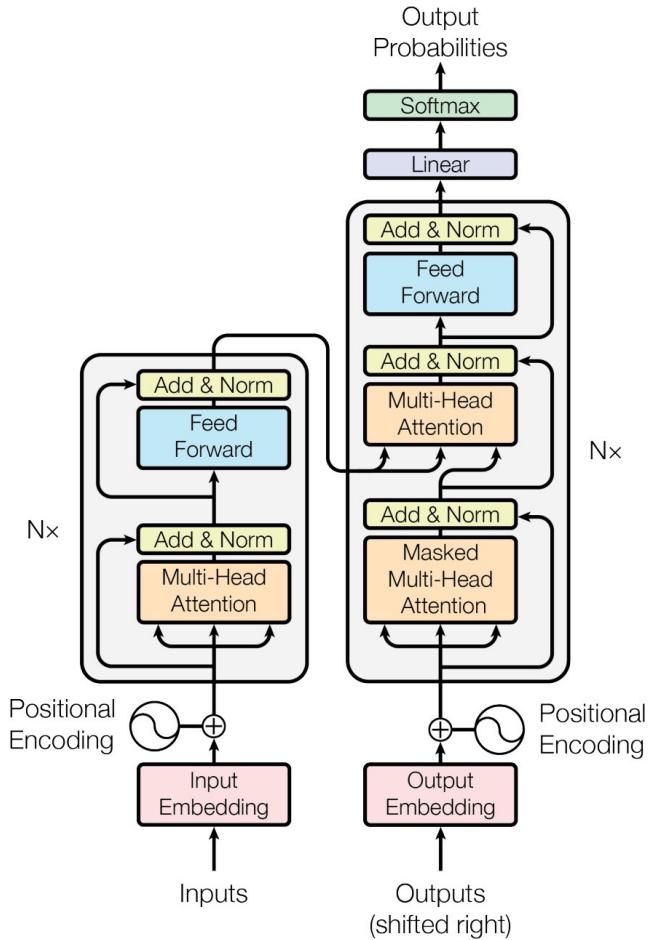
Encoder-decoder models

- Encoder = read or encode the input,
- Decoder = generate or decode the output



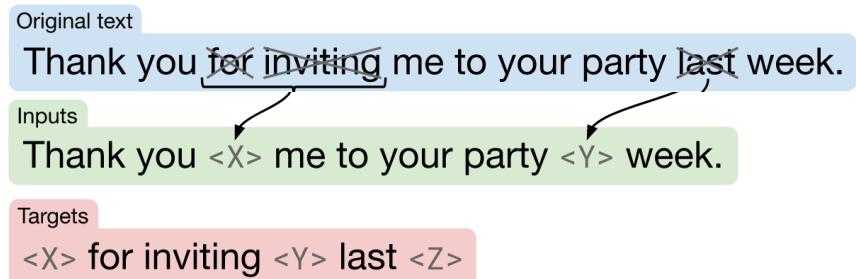
Encoder-decoder models

- 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 (2019)

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

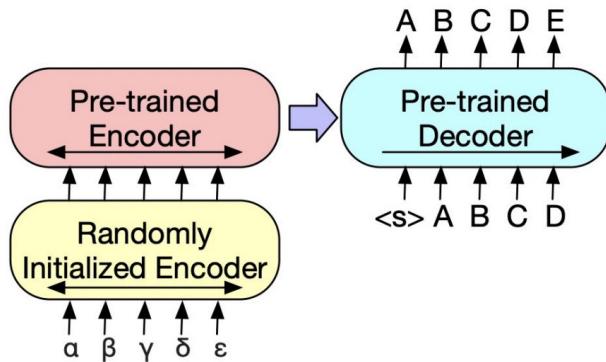


Model	Parameters	No. of layers	d_{model}	d_{ff}	d_{kv}	No. of heads
Small	60M	6	512	2048	64	8
Base	220M	12	768	3072	64	12
Large	770M	24	1024	4096	64	16
3B	3B	24	1024	16384	128	32
11B	11B	24	1024	65536	128	128

- 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>

BART (Lewis et al. 2020)

- Similar Architecture as T5.
 - Corrupt the input -> ask the model to reconstruct the original input
 - Outperformed existing methods on generative tasks (question answering, and summarization).



BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension

Mike Lewis*, Yinhan Liu*, Naman Goyal*, Marjan Ghazvininejad,
Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, Luke Zettlemoyer
Facebook AI

{mikelewis,yinhanliu,naman}@fb.com

BART

```
from transformers import BartTokenizer, BartForConditionalGeneration

tokenizer = BartTokenizer.from_pretrained("facebook/bart-large")
model = BartForConditionalGeneration.from_pretrained("facebook/bart-large")

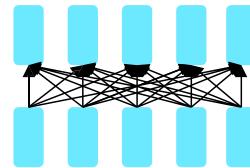
TXT = "The sun is <mask> ."
input_ids = tokenizer([TXT], return_tensors="pt")["input_ids"]
logits = model(input_ids).logits

masked_index = (input_ids[0] == tokenizer.mask_token_id).nonzero().item()
probs = logits[0, masked_index].softmax(dim=0)
values, predictions = probs.topk(5)

tokenizer.decode(predictions).split()
```

Result: ['located', 'at', 'approximately', 'also', 'about']

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: Pre-training of Deep Bidirectional Transformers for Language Understanding

BERT (2018)

- Transformer based network to learn representations of language
- Compared to ELMo (2017)
 - Bi-directional LSTM -> Self-attention
 - Massive data
 - Masked-LM objective

Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova

Google AI Language

{jacobdevlin, mingweichang, kentonl, kristout}@google.com

Abstract

We introduce a new language representation model called **BERT**, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

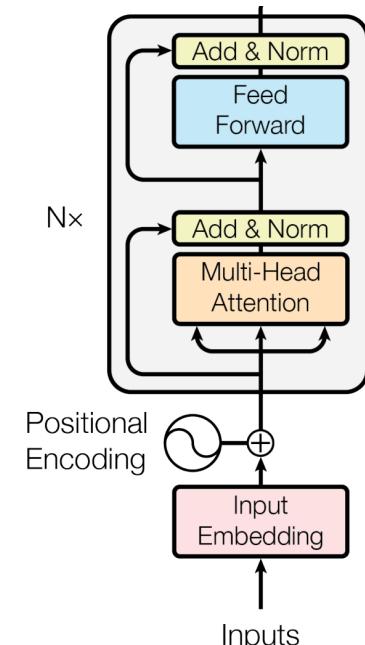
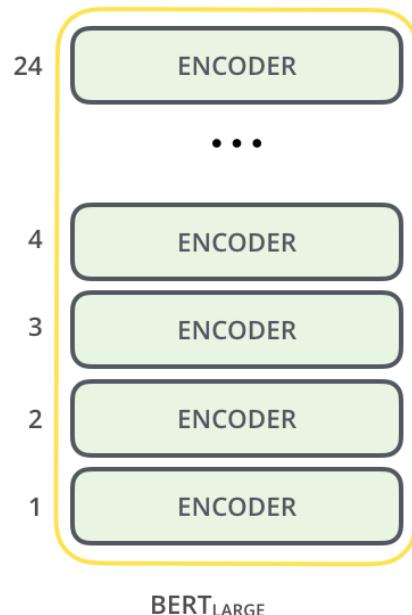
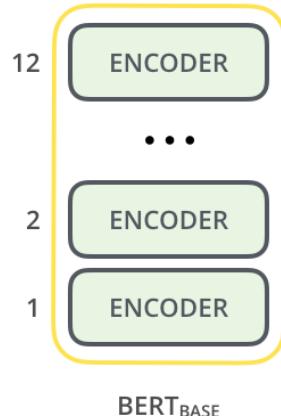
BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute im-

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning *all* pre-trained parameters. The two approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

We argue that current techniques restrict the power of the pre-trained representations, especially for the fine-tuning approaches. The major limitation is that standard language models are unidirectional, and this limits the choice of architectures that can be used during pre-training. For example, in OpenAI GPT, the authors use a left-to-

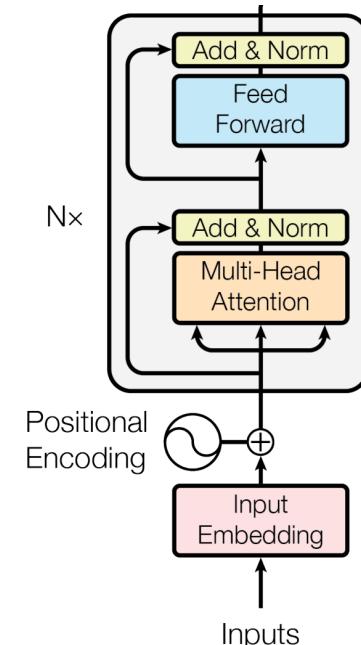
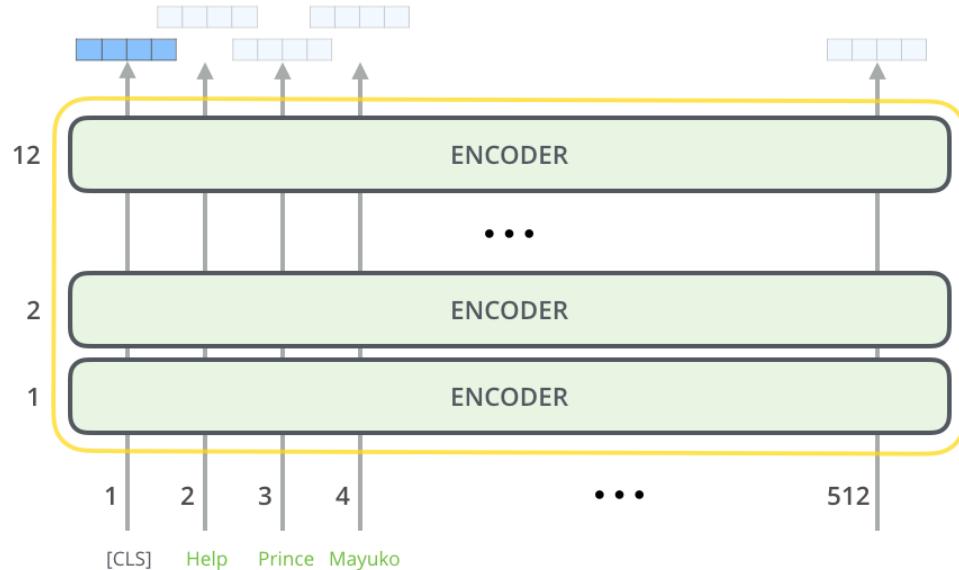
BERT: Architecture

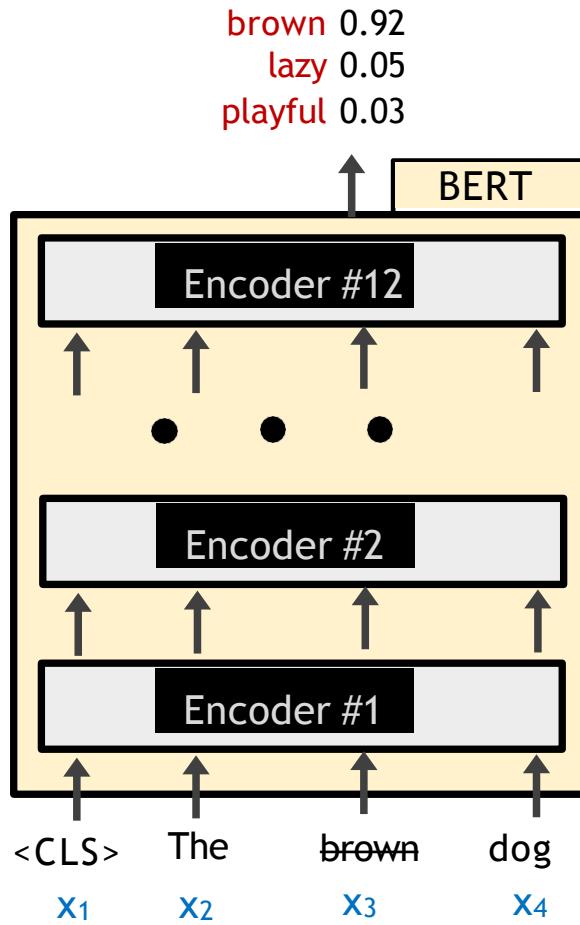
- Stacks of Transformer encoders



BERT: Architecture

- Model output dimension: 512





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.

Paris is the [MASK] of France.

Compute

Computation time on cpu: cached

capital	0.997
heart	0.001
center	0.000
centre	0.000
city	0.000

</> JSON Output

Maximize

Probing BERT Masked LM

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

Today is Tuesday, so tomorrow is [MASK].

Compute

Computation time on cpu: cached



</> JSON Output

Maximize

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

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

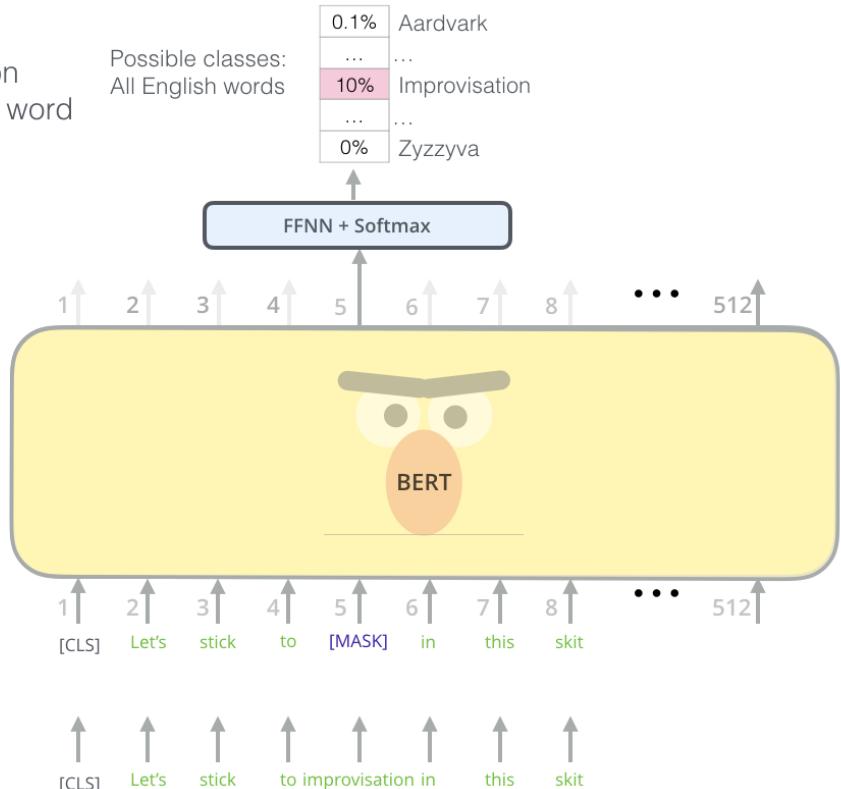
Use the output of the masked word's position to predict the masked word

Possible classes:
All English words

0.1%	Aardvark
...	...
10%	Improvisation
...	...
0%	Zyzyva

Randomly mask 15% of tokens

Input



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

store

Galon

the man went to the [MASK] to buy a [MASK] of milk

- 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.

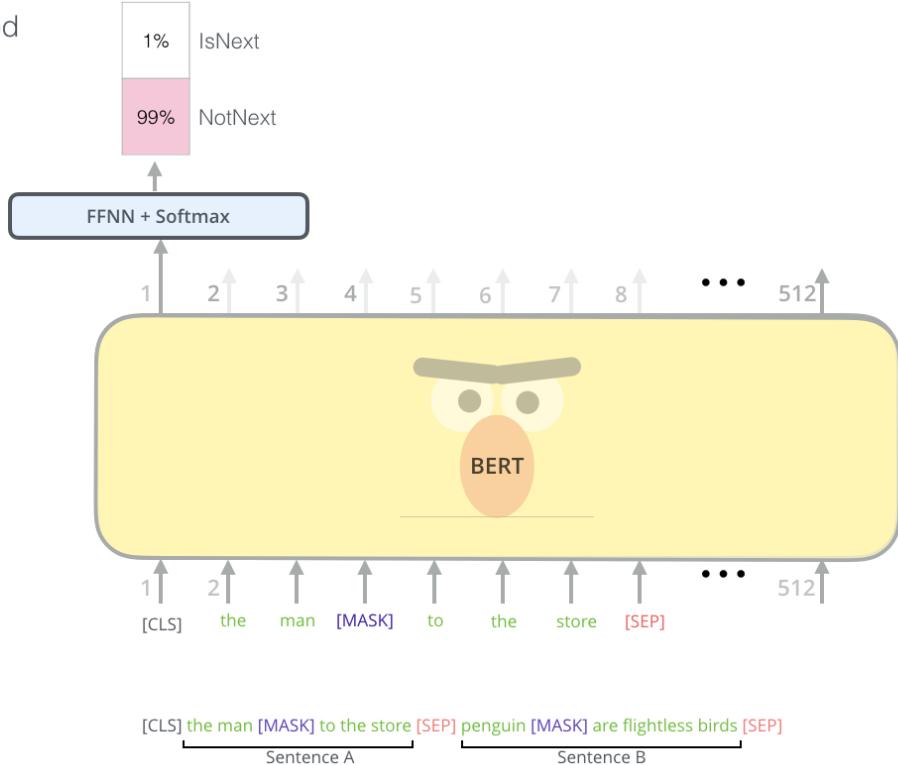
PMI Masking (Levine et al., 2021) <https://arxiv.org/pdf/2010.01825.pdf>

SpanBERT (Joshi et al., 2020) <https://arxiv.org/pdf/1907.10529.pdf>

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

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

Predict likelihood
that sentence B
belongs after
sentence A



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.

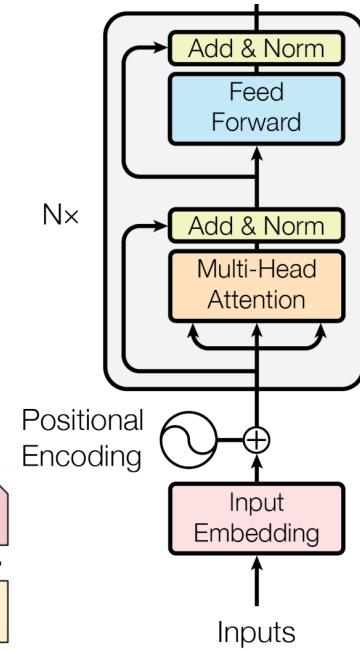
Sentence B = Penguins are flightless.

Label = NotNextSentence

BERT: Input Representation

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

Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Token Embeddings	E _[CLS]	E _{my}	E _{dog}	E _{is}	E _{cute}	E _[SEP]	E _{he}	E _{likes}	E _{play}	E _{##ing}	E _[SEP]
Segment Embeddings	+ E _A	+ E _B									
Position Embeddings	E ₀	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	E ₇	E ₈	E ₉	E ₁₀



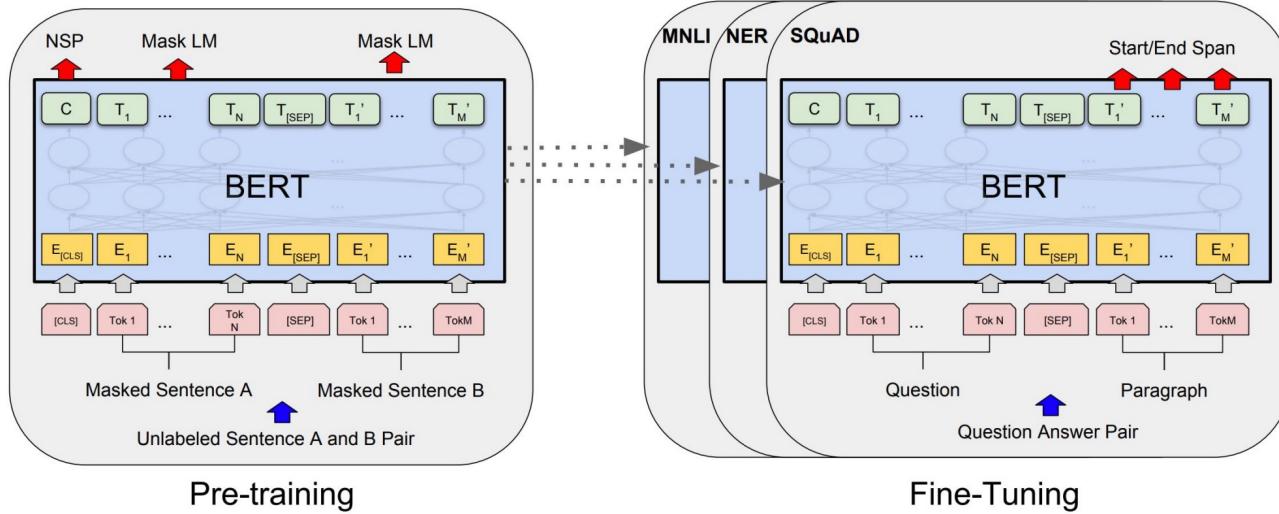
Training

- 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)

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



An Example Result: SWAG

Leaderboard

- Human Performance (88.00%)
- Running Best
- ◆ Submissions

Rank	Model	Test Score
1	BERT (Bidirectional Encoder Representations from Transfo... <i>Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova</i> 10/11/2018	86.28%
2	OpenAI Transformer Language Model <i>Original work by Alec Radford, Karthik Narasimhan, Tim Salimans, ...</i> 10/11/2018	77.97%
3	ESIM with ELMo <i>Zellers, Rowan and Bisk, Yonatan and Schwartz, Roy and Choi, Yejin</i> 08/30/2018	59.06%
4	ESIM with Glove <i>Zellers, Rowan and Bisk, Yonatan and Schwartz, Roy and Choi, Yejin</i> 08/29/2018	52.45%

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 o ([CLS])

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!

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.

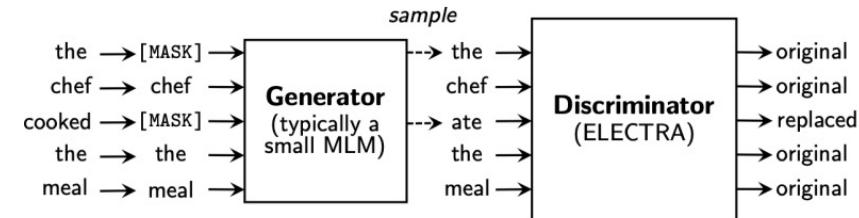
What Happened After BERT?

- 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
BERT _{LARGE}						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

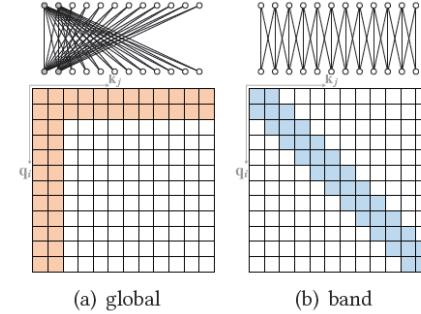
What Happened After BERT?

- 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

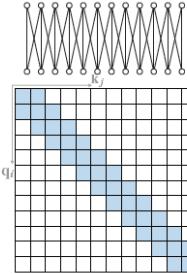


What Happened After BERT?

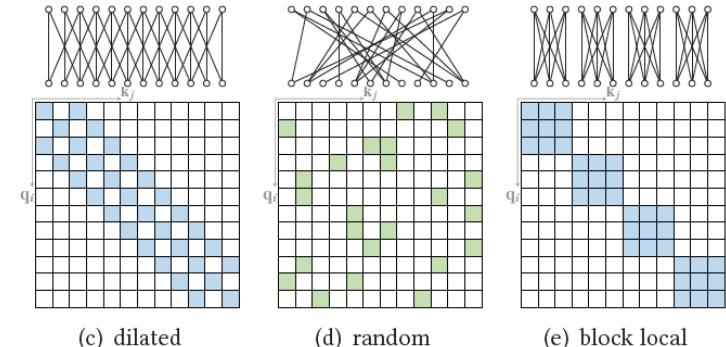
- Models that handle long contexts
 - Longformer, Big Bird, ...
- Multilingual BERT
 - Trained single model on 104 languages from Wikipedia.
- BERT extended to different domains
 - SciBERT, BioBERT, FinBERT, ClinicalBERT, ...
- Making BERT smaller to use
 - DistillBERT, TinyBERT, ...



(a) global



(b) band



(c) dilated

(d) random

(e) block local

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

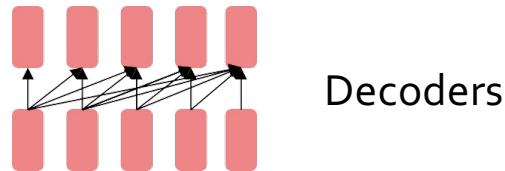
Sascha Rothe, Shashi Narayan, Aliaksei Severyn

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



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 Child¹ David Luan¹ 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

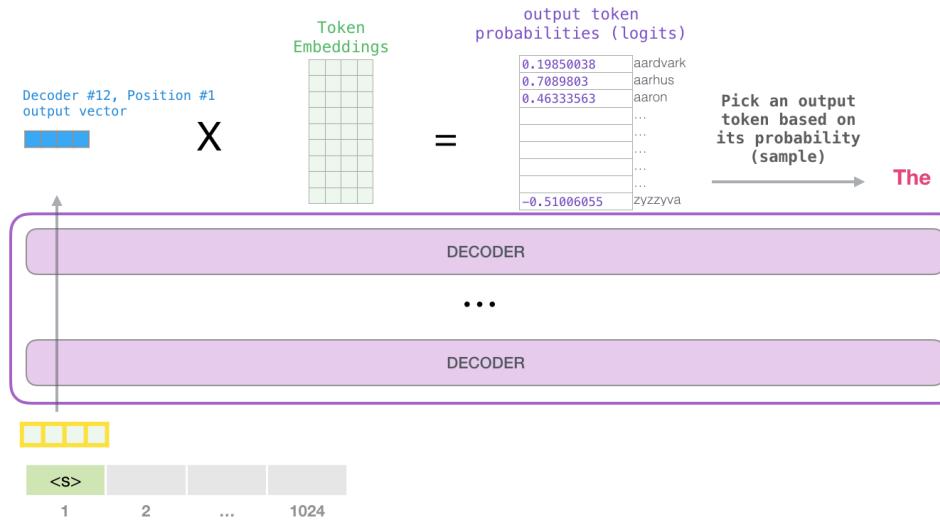


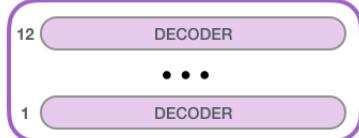
Image by <http://jalammar.github.io/illustrated-gpt2/>

GPT2: Model Sizes

Play with it here: <https://huggingface.co/gpt2>



GPT-2
SMALL

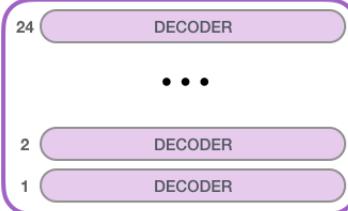


Model Dimensionality: 768

117M parameters



GPT-2
MEDIUM

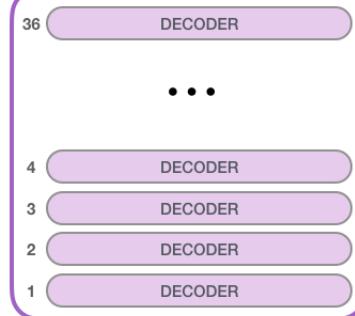


Model Dimensionality: 1024

345M



GPT-2
LARGE

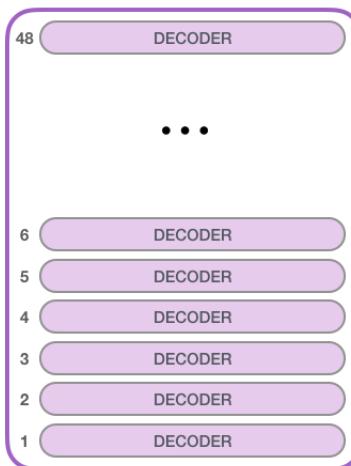


Model Dimensionality: 1280

762M



GPT-2
EXTRA
LARGE



Model Dimensionality: 1600

1542M

GPT-2 is identical to GPT-1, but:

- Has Layer normalization in between each sub-block (as we've already seen)
- Vocab extended to 50,257 tokens and context size increased from 512 to 1024
- Data: 8 million docs from the web (Common Crawl), minus Wikipedia

Language Models are Unsupervised Multitask Learners

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

GPT2: Some Results

Language Models are Unsupervised Multitask Learners

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW (PPL)
SOTA	99.8	56.25	85.7	82.3	39.14	46.54	0.99	1.08	18.3	21.8
117M	35.13	45.99	87.65	83.4	29.41	65.85	1.16	1.17	37.50	75.20
345M	15.60	55.48	92.35	87.1	22.76	47.33	1.01	1.06	26.37	55.72
762M	10.87	60.12	93.45	88.0	19.93	40.31	0.97	1.02	22.05	44.575
1542M	8.63	63.24	93.30	89.05	18.34	35.76	0.93	0.98	17.48	42.16

Table 3. Zero-shot results on many datasets. No training or fine-tuning was performed for any of these results. PTB and WikiText-2 results are from (Gong et al., 2018). Other language model results are from (Dai et al., 2019).

Article: Prehistoric man sketched an incredible array of prehistoric beasts on the rough limestone walls of a cave in modern day France 36,000 years ago.

Now, with the help of cutting-edge technology, those works of art in the Chauvet-Pont-d'Arc Cave have been reproduced to create the biggest replica cave in the world.

The manmade cavern named the Caverne du Pont-d'Arc has been built a few miles from the original site in Vallon-Pont-D'arc in Southern France and contains 1,000 painstakingly-reproduced drawings as well as around 450 bones and other features...

Cavemen and women sketched an incredible array of prehistoric beasts on the rough limestone walls of a cave 36,000 years ago and now a replica has been created (pictured)

...

GPT-2: The original site in Vallon-Pont-D'arc in Southern France is a Unesco World Heritage site and is the oldest known and the best preserved cave decorated by man. The replica cave was built a few miles from the original site in Vallon-Pont-D'Arc in Southern France. The cave contains images of 14 different species of animals including woolly rhinoceros, mammoths, and big cats.

Reference: Cave mimics famous Caverne du Pont-d'Arc in France, the oldest cave decorated by man and the best preserved. The replica contains all 1,000 paintings which include 425 such as a woolly rhinoceros and mammoths. Minute details were copied using 3D modelling and anamorphic techniques, often used to shoot widescreen images. The modern cave also includes replica paw prints of bears, bones and details preserved in the original cave.

Impact of GPT2

- Zero-shot learning (no use of task-specific supervision) increasingly become a reality.

NMT: “Translate to french,” <English text>, <French text>.

QA: “Answer the question,” <Document>, <Question>, <Answer>.

SUMM: <Document> “TL; DR:” <Summarization>

GPT-3: A Very Large Language Model (2020)

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



Size Comparisons

- **BERT-Base** model has 12 transformer blocks, 12 attention heads,
 - 110M parameters!
- **BERT-Large** model has 24 transformer blocks, 16 attention heads,
 - 340M parameters!
- **GPT-2** is trained on 40GB of text data (8M webpages)!
 - 1.5B parameters!
- **GPT-3** is an even bigger version of GPT-2, but isn't open-source
 - 175B parameters!

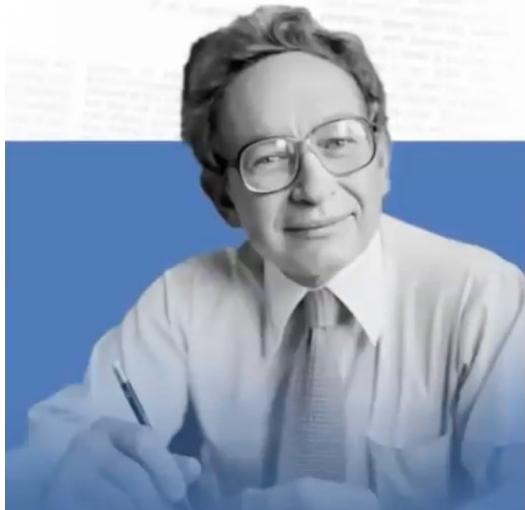
Impact of GPT3

- 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 open source efforts:
 - OPT, LLaMA by Meta, BLOOM by Huggingface, etc.

GPT4

- TBD
- Also acknowledge that RLHF stuff will be discussed later.

More is different



We expect to encounter fascinating and, I believe, very fundamental questions at each stage in fitting together less complicated pieces into the more complicated system and understanding the basically new types of behavior which can result.

— “More Is Different , SCIENCE”
4 August 1972
Philip W. Anderson

```
import openai
openai.api_key = ("sk-3kFAtzisBypel")
my_prompt = '''The sun is [MASK].  
Replace [MASK] with the most probable 5 words to replace, and give me their probabilities.'''
# Here set parameters as you like
response = openai.Completion.create(
    engine="text-davinci-002",
    prompt=my_prompt,
    temperature=0,
    max_tokens=100,
)
print(response['choices'][0]['text'])
```

Try it yourself!

Ada

Fastest

\$0.0004 / 1K tokens

Babbage

\$0.005 / 1K tokens

Curie

\$0.0020 / 1K tokens

Davinci

Most powerful

\$0.0200 / 1K tokens

Fine-tuned models

Create your own custom models by fine-tuning our base models with your training data. Once you fine-tune a model, you'll be billed only for the tokens you use in requests to that model.

[Learn more about fine-tuning ↗](#)

Model	Training	Usage
Ada	\$0.0004 / 1K tokens	\$0.0016 / 1K tokens
Babbage	\$0.0006 / 1K tokens	\$0.0024 / 1K tokens
Curie	\$0.0030 / 1K tokens	\$0.0120 / 1K tokens
Davinci	\$0.0300 / 1K tokens	\$0.1200 / 1K tokens

Other Available [Decoder] LMs

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

<https://huggingface.co/EleutherAI>

<https://6b.eleuther.ai/>

Meta/Facebook: OPT (Open Pre-trained Transformer), various sizes up to 175B

<https://huggingface.co/facebook/opt-125m>

BLOOM, 176B model

<https://huggingface.co/bigscience/bloom>

LLaMA, 65B

<https://github.com/facebookresearch/llama>

Revisiting Encoding Positional Information

- [Toward Length Extrapolatable Transformers -- Ta-Chung Chi \(CMU\) – YouTube](#)

Self-Attention

- Can write it in matrix form:
- Given input \mathbf{x} :

$$Q = \mathbf{W}^q \mathbf{x}$$

$$K = \mathbf{W}^k \mathbf{x}$$

$$V = \mathbf{W}^v \mathbf{x}$$

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



hardmaru
@hardmaru

...

The most important formula in deep learning after 2018

Self-Attention

What is self-attention? Self-attention calculates a weighted average of feature representations with the weight proportional to a similarity score between pairs of representations. Formally, an input sequence of n tokens of dimensions d , $X \in \mathbf{R}^{n \times d}$, is projected using three matrices $W_Q \in \mathbf{R}^{d \times d_q}$, $W_K \in \mathbf{R}^{d \times d_k}$, and $W_V \in \mathbf{R}^{d \times d_v}$ to extract feature representations Q , K , and V , referred to as query, key, and value respectively with $d_k = d_q$. The outputs Q , K , V are computed as

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V. \quad (1)$$

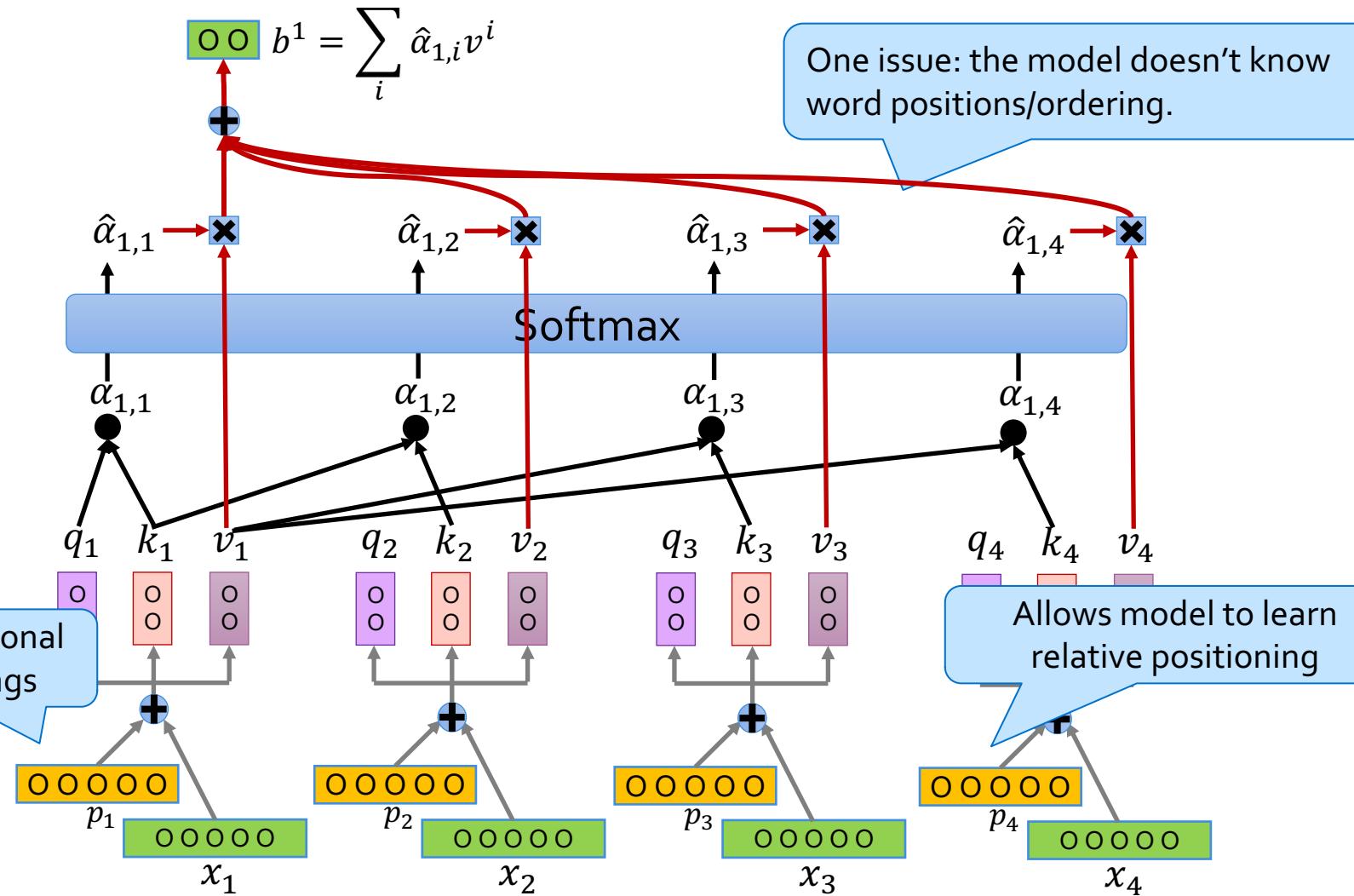
So, self-attention can be written as,

$$S = D(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_q}}\right)V, \quad (2)$$

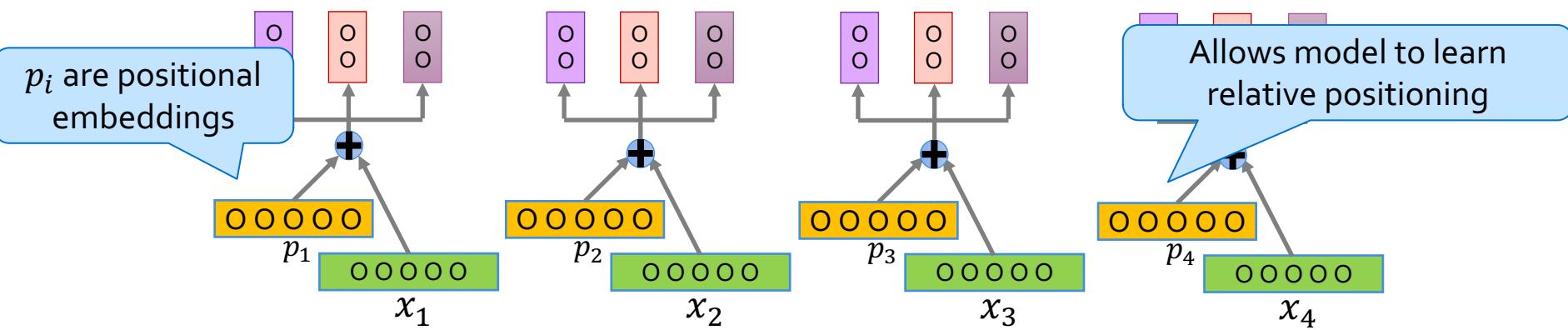
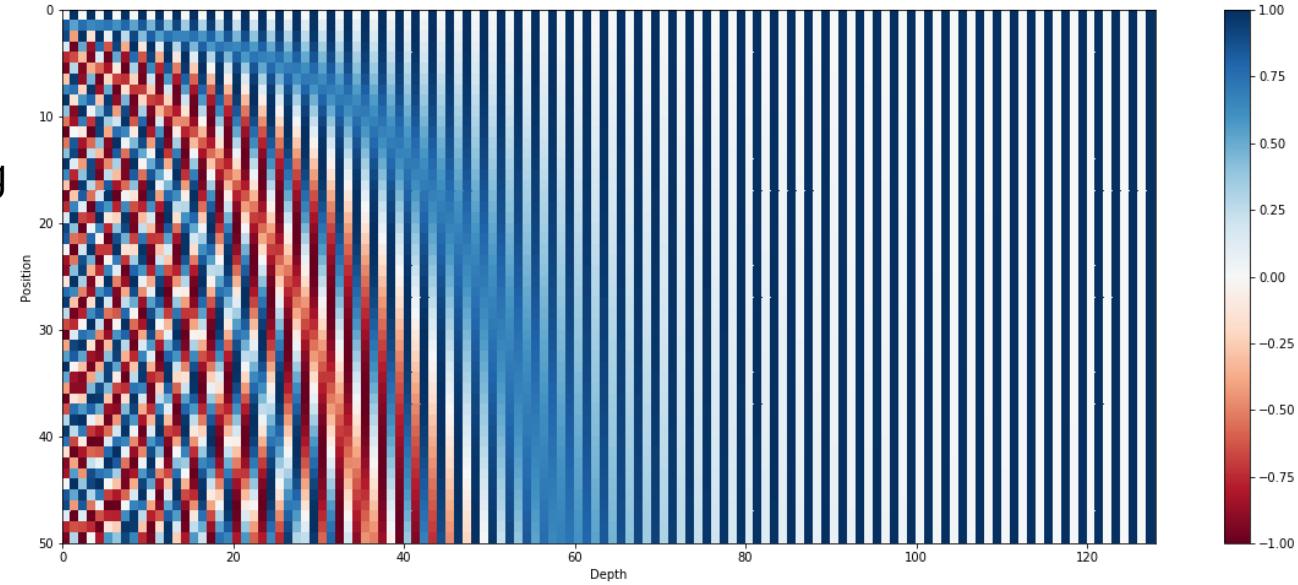
where softmax denotes a *row-wise* softmax normalization function. Thus, each element in S depends on all other elements in the same row.

9:08 PM · Feb 9, 2021 · Twitter Web App

553 Retweets 42 Quote Tweets 3,338 Likes



An approach:
Sine/Cosine encoding



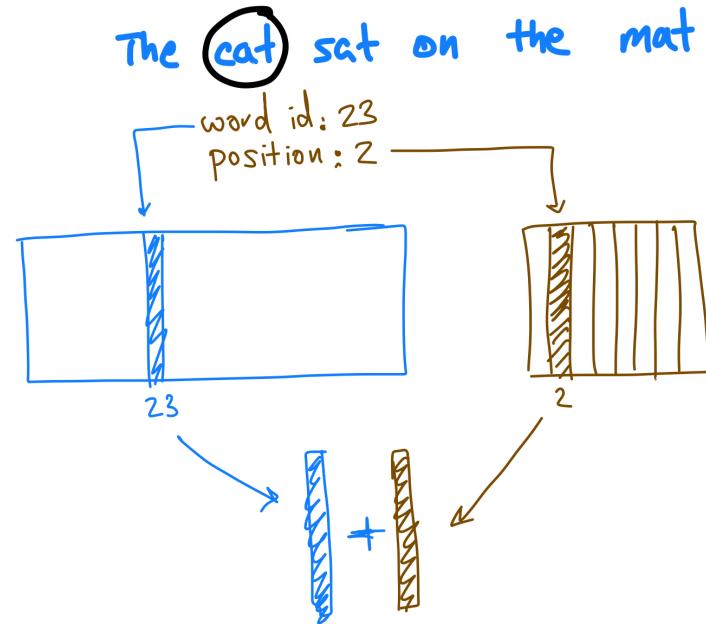
How to encode position information?

- Multi-head attention doesn't have a way to know whether an input token comes before or after another
 - Position is important in sequence modeling in NLP
- One way to introduce position information is add individual position encodings to the input for each position in the sequence

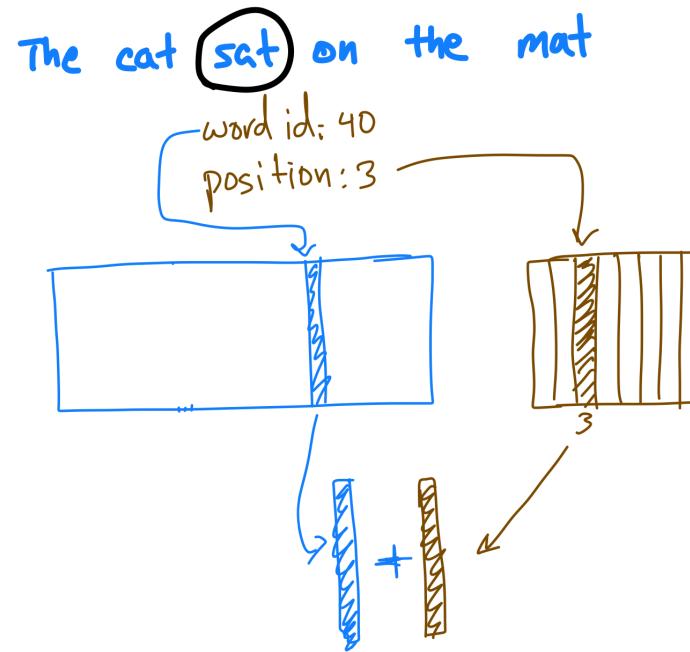
$$\blacksquare \quad x_j = x_j + pos_j$$

- Where pos_j is a position vector

How to encode position information?



How to encode position information?



Absolute position embeddings

- Learned positions embeddings:
 - Maximum length that can be presented is limited
- Functional position embeddings (e.g., sinusoidal):

$$\text{Embedding}[i, 2k] = \sin(\text{position} / (10000^{(2k / d_{\text{model}})}))$$

$$\text{Embedding}[i, 2k+1] = \cos(\text{position} / (10000^{(2k / d_{\text{model}})}))$$

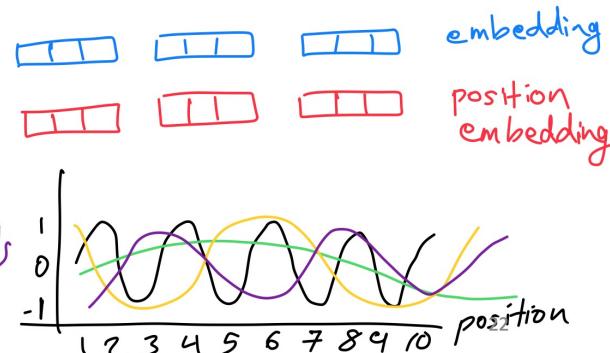
Absolute position embeddings

- Learned positions embeddings:
 - Maximum length that can be presented is limited
- Functional position embeddings (e.g., sinusoidal):

$$\text{Embedding}[i, 2k] = \sin(\text{position} / (10000^{(2k/d_model)}))$$

$$\text{Embedding}[i, 2k+1] = \cos(\text{position} / (10000^{(2k/d_model)}))$$

each color corresponds
to one dimension



Limitations of learned positions embeddings

- Fixed length of the embedding matrix
- Do not directly model the distance between tokens
- Not always meaningful because of “packing” often used during pre-training
 - Pack multiple examples in a single context

cats sleep

The cat sat on the mat

Birds fly high

The sun rises daily

Limitations of learned positions embeddings

- Fixed length of the embedding matrix
- Do not directly model the distance between tokens
- Not always meaningful because of “packing” often used during pre-training
 - Pack multiple examples in a single context

cats sleep

The cat sat on the mat

Birds fly high

The sun rises daily

context size = 9

Limitations of learned positions embeddings

- Fixed length of the embedding matrix
- Do not directly model the distance between tokens
- Not always meaningful because of “packing” often used during pre-training
 - Pack multiple examples in a single context

[cats sleep P P P P P P P]
[The cat sat on the mat P P P]
[Birds fly high P P P P P P]
[The sun rises daily P P P P P]

context size = 9

Limitations of learned positions embeddings

- Fixed length of the embedding matrix
- Do not directly model the distance between tokens
- Not always meaningful because of “packing” often used during pre-training
 - Pack multiple examples in a single context

context size = 9

[cats sleep P P P P P P P]

[The cat sat on the mat P P P]

[Birds fly high P P P P P]

[The sun rises daily P P P P P]

[The cat sat on the mat S cats sleep]

[The sun rises daily S Birds fly high P]

Limitations of learned positions embeddings

- Fixed length of the embedding matrix
- Do not directly model the distance between tokens
- Not always meaningful because of “packing” often used during pre-training
 - Pack multiple examples in a single context

context size = 9

[cats sleep P P P P P P P]
[The cat sat on the mat P P P]
[Birds fly high P P P P P]
[The sun rises daily P P P P P]

[The cat sat on the mat S cats sleep]
[The sun rises daily S Birds fly high P]

Special tokens

Relative position embedding

- Relative position embeddings produce different learned embedding according to the offset between query and key
- T5 (Raffel et al., 2019) uses a scalar that is added to the attention raw scores
- Each attention head uses a different position bias

r ₀	r ₁	r ₂	r ₃
r ₀	r ₁	r ₂	
	r ₀	r ₁	
			r ₀

The cat sat on

raw scores (logits)

$$\left\{ \begin{array}{l} q_0 k_0 \\ q_0 k_1 \\ q_0 k_2 \\ q_0 k_3 \end{array} \right. \longrightarrow \left\{ \begin{array}{l} q_0 k_0 + r_0 \\ q_0 k_1 + r_1 \\ q_0 k_2 + r_2 \\ q_0 k_3 + r_3 \end{array} \right. \longrightarrow \text{softmax}()$$

Rotary position embeddings

- **Goal:** find a positional encoding function $f(\mathbf{x}, \ell)$ for an input and a position ℓ such that for two inputs \mathbf{q} and \mathbf{k} and positions m and n is only sensitive to values \mathbf{q} and \mathbf{k} and their relative position.
- Recall the geometric definition of the dot product between Euclidean vectors

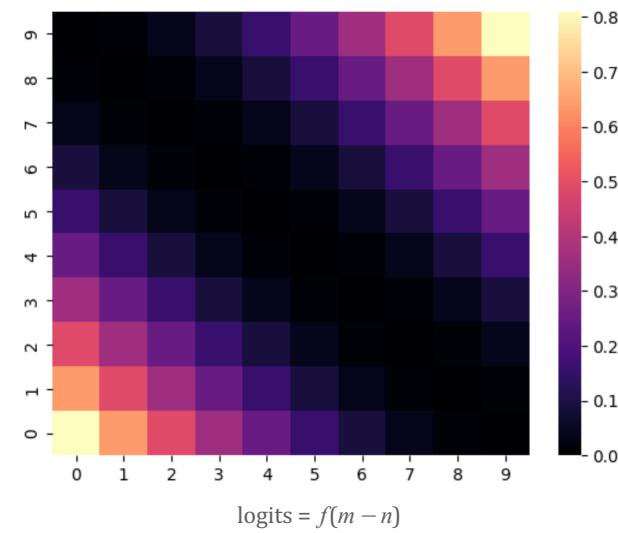
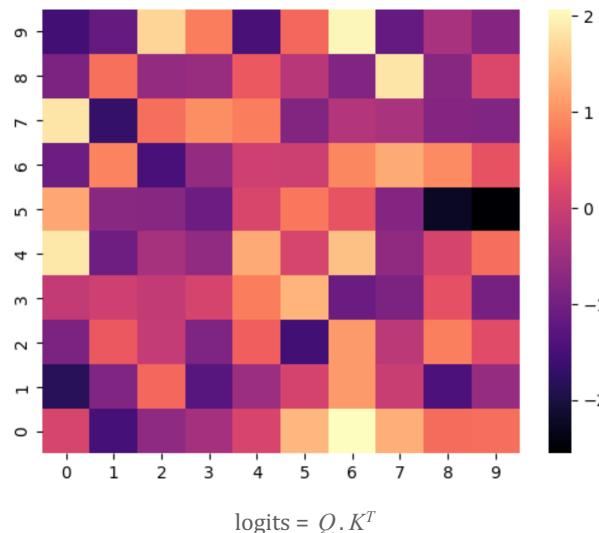
$$\mathbf{q} \cdot \mathbf{k} = \|\mathbf{q}\| \|\mathbf{k}\| \cos(\theta_{qk})$$

Rotary Positional Embeddings (RoPE)

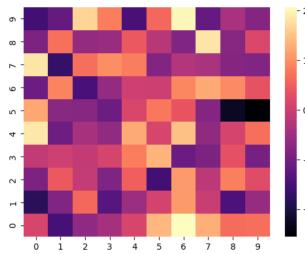
- The intuition behind RoPE is that we can represent the token embeddings as complex numbers and their positions as pure rotations that we apply to them.
- By exploiting the nature of rotations, the dot product used in self- attention will have the property we are looking for, preserving relative positional information while discarding absolute position.

Rotary Positional Embeddings (ROPE)

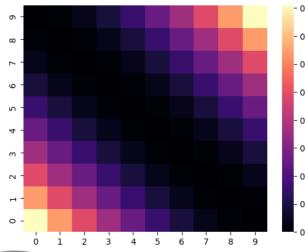
- Rotates the query and key vectors based on their positions in the sequence



Rotary Positional Embeddings (ROPE)



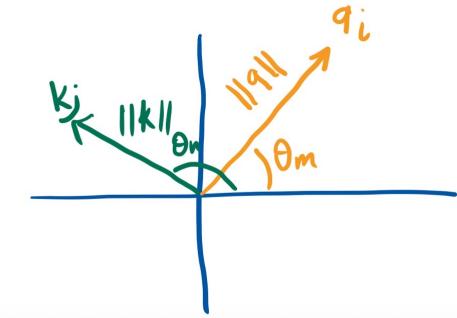
$$a_{ij} = q_i \cdot k_j$$



$$a_{ij} = \Theta(m - n)$$

$$\begin{array}{|c|c|} \hline q_i^{(o)} & q_i^{(u)} \\ \hline \end{array}$$

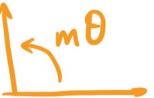
$$\begin{array}{|c|c|} \hline k_j^{(o)} & k_j^{(u)} \\ \hline \end{array}$$



Rotary Positional Embeddings (ROPE)

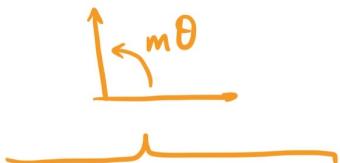
$$f_{\{q,k\}}(\mathbf{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W_{\{q,k\}}^{(11)} & W_{\{q,k\}}^{(12)} \\ W_{\{q,k\}}^{(21)} & W_{\{q,k\}}^{(22)} \end{pmatrix} \begin{pmatrix} \mathbf{x}_m^{(1)} \\ \mathbf{x}_m^{(2)} \end{pmatrix}$$

Rotary Positional Embeddings (ROPE)


$$f_{\{q,k\}}(\mathbf{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W_{\{q,k\}}^{(11)} & W_{\{q,k\}}^{(12)} \\ W_{\{q,k\}}^{(21)} & W_{\{q,k\}}^{(22)} \end{pmatrix} \underbrace{\begin{pmatrix} x_m^{(1)} \\ x_m^{(2)} \end{pmatrix}}_{\text{the vector we are rotating}}$$

a rotation matrix

Rotary Positional Embeddings (ROPE)


$$f_{\{q,k\}}(\mathbf{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \underbrace{\begin{pmatrix} W_{\{q,k\}}^{(11)} & W_{\{q,k\}}^{(12)} \\ W_{\{q,k\}}^{(21)} & W_{\{q,k\}}^{(22)} \end{pmatrix}}_{\text{linear projections for queries and keys}} \underbrace{\begin{pmatrix} \mathbf{x}_m^{(1)} \\ \mathbf{x}_m^{(2)} \end{pmatrix}}_{\text{the vector we are rotating}}$$

a rotation matrix

linear projections for queries and keys

the vector we are rotating

Rotary Positional Embeddings (ROPE)

- General form

$$f_{\{q,k\}}(\mathbf{x}_m, m) = \mathbf{R}_{\Theta,m}^d \mathbf{W}_{\{q,k\}} \mathbf{x}_m$$

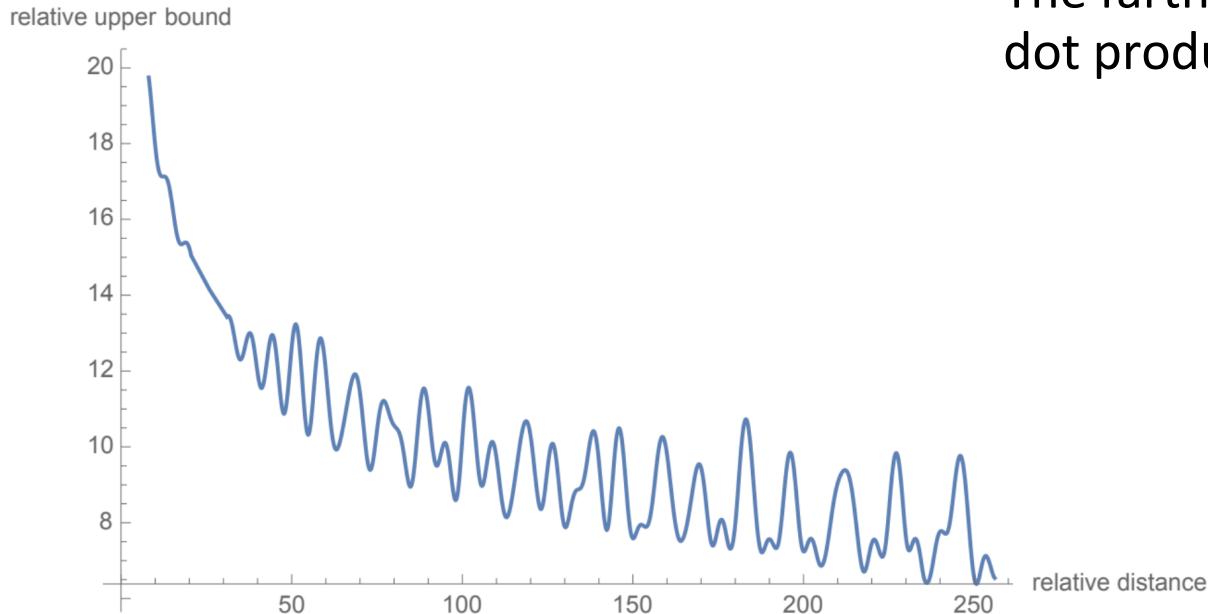
$$\mathbf{R}_{\Theta,m}^d = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

Rotary Positional Embeddings (ROPE)

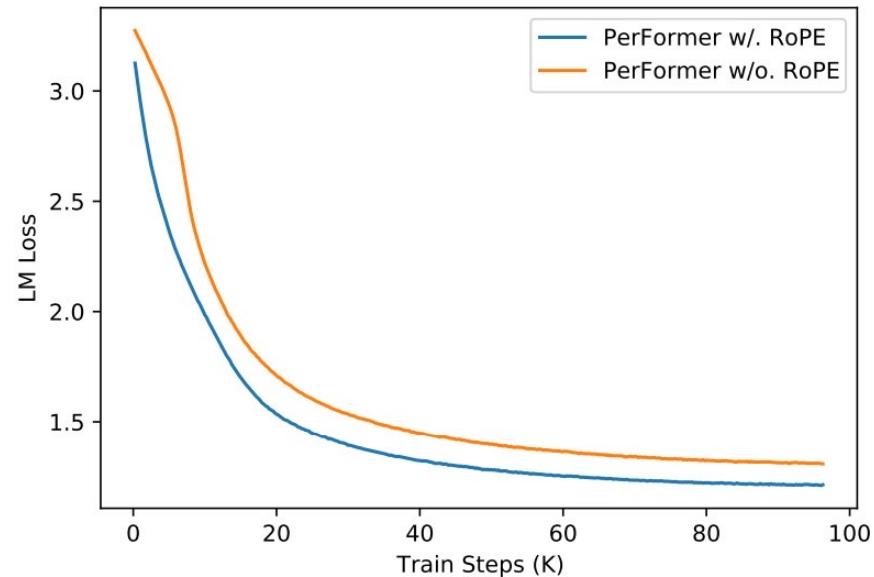
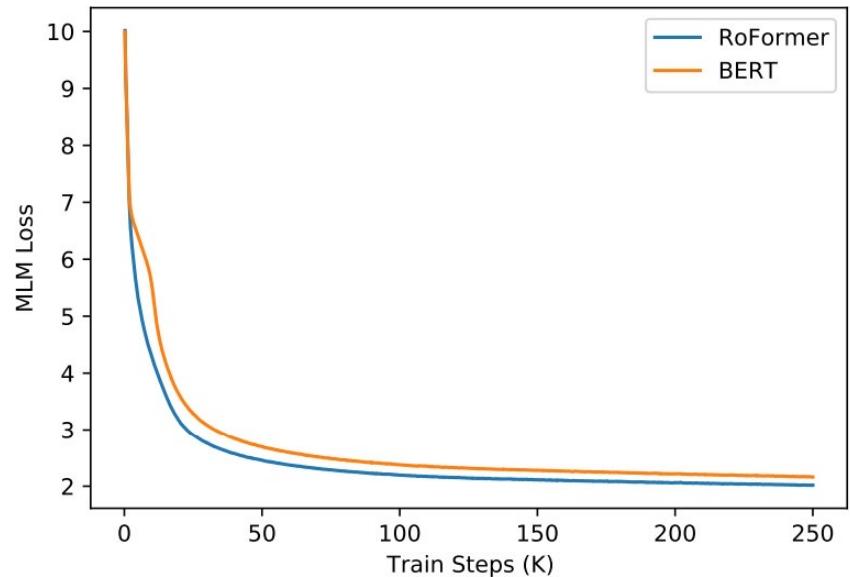
- The matrix multiplication is going to be very slow
- In practice we can use the following realization (taking advantage of the sparsity of R)

$$\mathbf{R}_{\Theta,m}^d \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix}$$

Rotary Positional Embeddings (ROPE)



The farther the tokens their dot product will be lower



What is a good property of a positional encoding?

- Should extrapolate to longer inputs

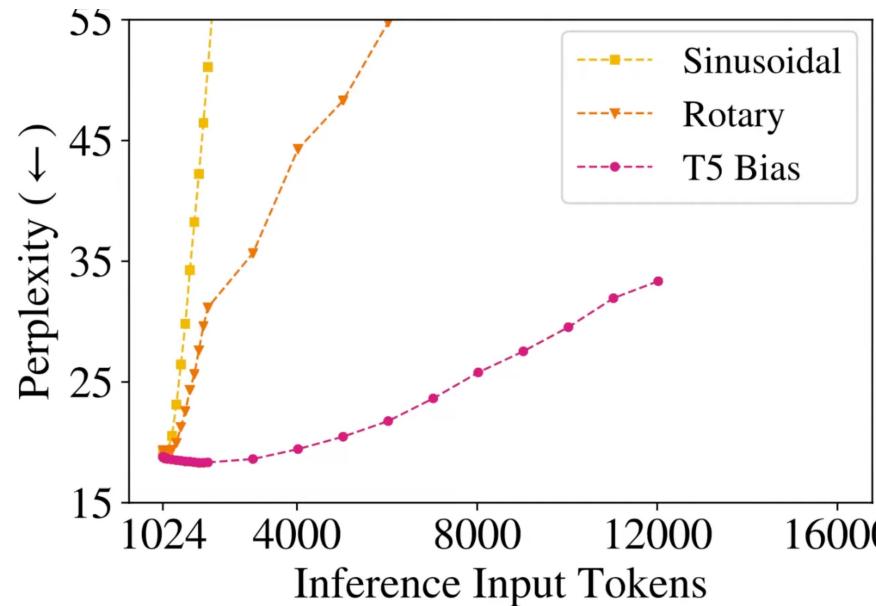


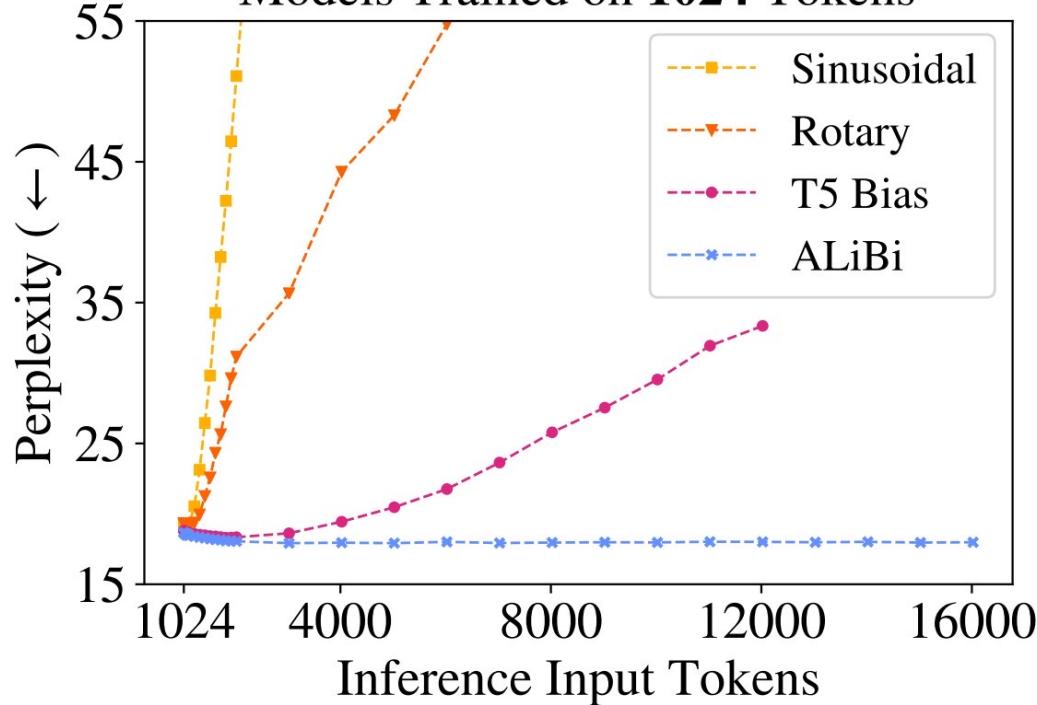
fig from Ofir Press (2023)

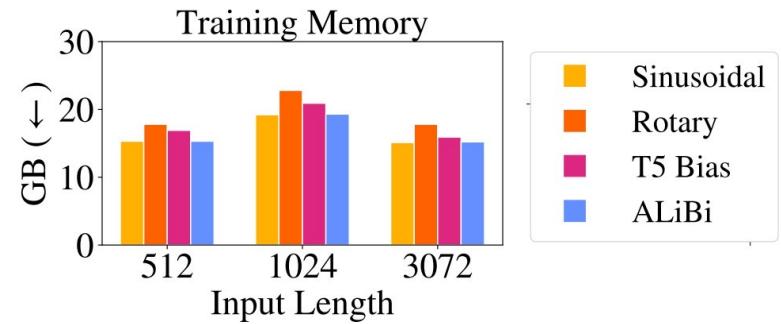
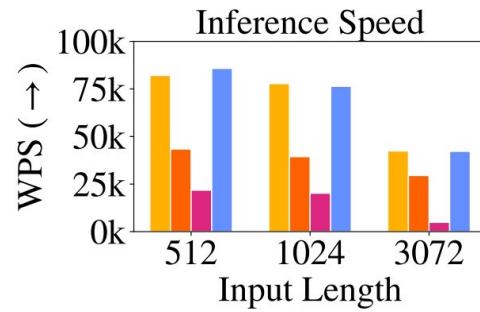
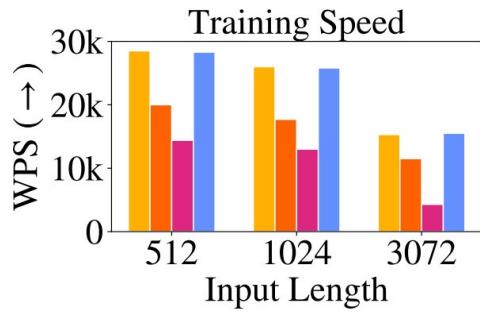
Attention with Linear Biases (ALiBi)

- Add a constant value to the attention logits
- Apply a head specific scalar m

$$\begin{matrix} q_1 \cdot k_1 \\ q_2 \cdot k_1 & q_2 \cdot k_2 \\ q_3 \cdot k_1 & q_3 \cdot k_2 & q_3 \cdot k_3 \\ q_4 \cdot k_1 & q_4 \cdot k_2 & q_4 \cdot k_3 & q_4 \cdot k_4 \\ q_5 \cdot k_1 & q_5 \cdot k_2 & q_5 \cdot k_3 & q_5 \cdot k_4 & q_5 \cdot k_5 \end{matrix} + \begin{matrix} 0 \\ -1 & 0 \\ -2 & -1 & 0 \\ -3 & -2 & -1 & 0 \\ -4 & -3 & -2 & -1 & 0 \end{matrix} \cdot m$$

Extrapolation for Models Trained on 1024 Tokens







- TODO:
 - LMs might work without explicit positional encoding
 - Unifying view of positional embeddings
 - Length generalization is different than keeping perplexity low.

Summary

Pre-Training Data

- TODO:
 - data cleaning
 - Deduplicating
 - Shuffling
 - Nearest neighbor sorting
 -

C4: The Data

- C4: Colossal Clean Crawled Corpus
 - Web-extracted text
 - English language only (langdetect)
 - 750GB
- Retain:
 - Sentences with terminal punctuation marks
 - Pages with at least 5 sentences, sentences with at least 3 words
 - Deduplicate three sentence spans
- Remove:
 - References to Javascript
 - “Lorem ipsum” text — placeholder text commonly used to demonstrate the visual form of a document

Letraset / Body Type

Lore ipsum dolor sit amet, consectetur adipisci
tempor incident ut labore et dolore magna aliqua
veniam, quis nostrund exercitation ullamcorpor s
commodo consequat. Duis autem vel eum irru-
esse molestiae consequat, vel illum dolore eu fugi
et iusto odio dignissim qui blandit praesent luptat
exceptur sint occaecat cupiditat non provident,
deserunt mollit anim id est laborum et dolor fuga
distinct. Nam liber tempor cum soluta nobis elige
quod maxim placeat facer possim omnis volupt

Lore ipsum dolor si amet, consectetur adipiscing
incident ut labore et dolore magna aliquam erat
nostrud exercitation ullamcorper suscipit laboris nis
duis autem vel eum irure dolor in reprehenderit i
dolore eu fugiat nulla pariatur. At vero eos et accusa
praesent luptatum delenit aigue duos dolor et mole
provident, simil tempor sunt in culpa qui officia de
fuga. Et harumd dereud facilis est er expedit disti
eligend optio congue nihil impedit doming id quod
assumenda est, omnis dolor repellend. Temporibus

Lore ipsum dolor si amet, consectetur adipiscing
incident ut labore et dolore magna aliquam erat
nostrud exercitation ullamcorper suscipit laboris nis
duis autem vel eum irure dolor in reprehenderit i
dolore eu fugiat nulla pariatur. At vero eos et accusa
praesent luptatum delenit aigue duos dolor et mole
provident, simil tempor sunt in culpa qui officia de
fuga. Et harumd dereud facilis est er expedit disti
eligend oprio congue nihil impedit doming id quod
assumenda est, omnis dolor repellend. Temporibus

LETRASET PATENTED PRINTED IN ENGLAND

PM 169

C4: The Data

C4: The Data

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 lemon is unknown, though lemons are thought to have first grown in Assam (a region in northeast India), northern Burma or China. A genomic study of the lemon indicated it was a hybrid between bitter orange (sour orange) and citron.

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.

Consectetur
adipiscing elit.
Curabitur in tempus quam. In mollis et ante at consectetur.

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);  
    }  
}
```

C4: The Data

- 750GB? What does that mean?

Data set	Size
★ C4	745GB
C4, unfiltered	6.1TB
RealNews-like	35GB
WebText-like	17GB
Wikipedia	16GB
Wikipedia + TBC	20GB

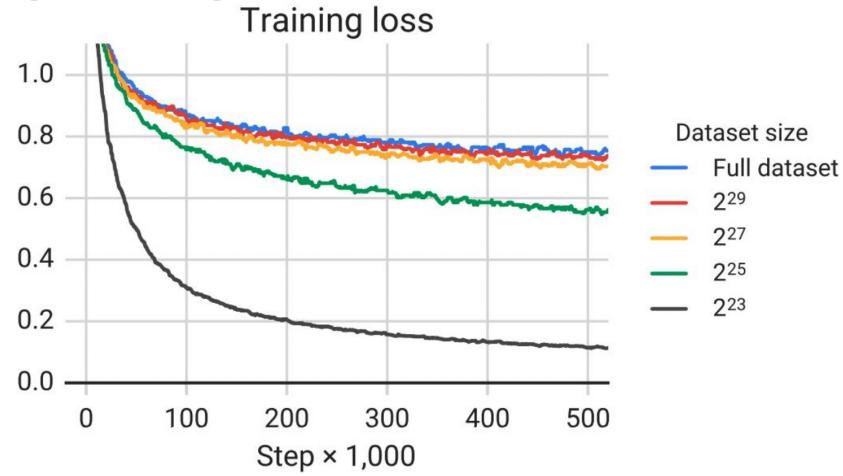
Pre-training Data: Experiment

- 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
RealNews-like	35GB	83.83	19.23	80.39	72.38	26.75	39.90	27.48
WebText-like	17GB	84.03	19.31	81.42	71.40	26.80	39.74	27.59
Wikipedia	16GB	81.85	19.31	81.29	68.01	26.94	39.69	27.67
Wikipedia + TBC	20GB	83.65	19.28	82.08	73.24	26.77	39.63	27.57

Pre-training Data: Experiment

- Effect of data repetitions
- Takeaways:
 - (Table) Performance **degrades** as the **information content shrinks**.
 - (Figure) The model **memorizes** the pre-training data, with **smaller/repeated datasets**.



Number of tokens	Repeats	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Full data set	0	83.28	19.24	80.88	71.36	26.98	39.82	27.65
2^{29}	64	82.87	19.19	80.97	72.03	26.83	39.74	27.63
2^{27}	256	82.62	19.20	79.78	69.97	27.02	39.71	27.33
2^{25}	1,024	79.55	18.57	76.27	64.76	26.38	39.56	26.80
2^{23}	4,096	76.34	18.33	70.92	59.29	26.37	38.84	25.81

LLM Name	Release Date	Pretraining		Checkpoints		Pretraining Dataset			Tokens (T)
		Code	Config	Model	Optim	Data Mix	Ordering	Available	
GPT-J [18]	May'21	✓	✓	✓	✓	✓	✓	✓	0.40
GPT-NeoX [19]	Apr'22	✓	✓	✓	✓	✓	✓	✓	0.40
OPT [20]	May'22	✓	✓	✓		✓			0.18
BLOOM [21]	Nov'22	✓	✓	✓	✓	✓	✓	✓	0.34
Pythia [16]	Feb'23	✓	✓	✓	✓	✓	✓	✓	0.30
LLaMA [3]	Feb'23		✓			✓			1.0
OpenLLaMA [11]	May'23	✓	✓	✓		✓		✓	1.0
INCITE [10]	May'23	✓	✓	✓		✓		✓	1.0
MPT [22]	May'23	✓	✓			✓			1.0
Falcon [23]	May'23		✓			✓			1.5
Llama 2 [4]	Jul'23		✓						2.0
Qwen [24]	Aug'23			✓					2.4
Mistral [6]	Sep'23								?
Yi [25]	Nov'23								?
AMBER	Dec'23	✓	✓	✓	*	✓	✓	✓	1.3
CRYSTALCODER	Dec'23	✓	✓	✓	✓	✓	✓	✓	1.4

Table 1: Summary of notable open-source LLMs. We note a trend of progressively less disclosure of important pretraining details over time: (1) availability of pretraining code, (2) disclosure of training configurations and hyperparameters, (3) intermediate checkpoints of model weights, (4) intermediate checkpoints of optimizer states, (5) disclosure of data mixture and sources, (6) reproducibility of pretraining data sequence, and (7) availability (or reconstruction scripts) of the pretraining data.

* Amber optimizer states are lost due to errors in early implementation.

- On data: <https://huyenchip.com/2023/05/02/rlhf.html>

Practical Considerations

Other Training Considerations

- Training objectives
- Learning rates
- Training divergence



Pre-training objectives



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.

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

Objectives: Experiments

- Performance of the i.i.d. corruption objective with different **corruption rates**
- Takeaway:
 - Little corruption rate may prevent effective learning.
 - Larger corruption rate leads to downstream performance degradation.
 - Larger corruption rate also leads to longer targets, slowing down training.

Corruption rate	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
10%	82.82	19.00	80.38	69.55	26.87	39.28	27.44
★ 15%	83.28	19.24	80.88	71.36	26.98	39.82	27.65
25%	83.00	19.54	80.96	70.48	27.04	39.83	27.47
50%	81.27	19.32	79.80	70.33	27.01	39.90	27.49

On Pre-training Objectives

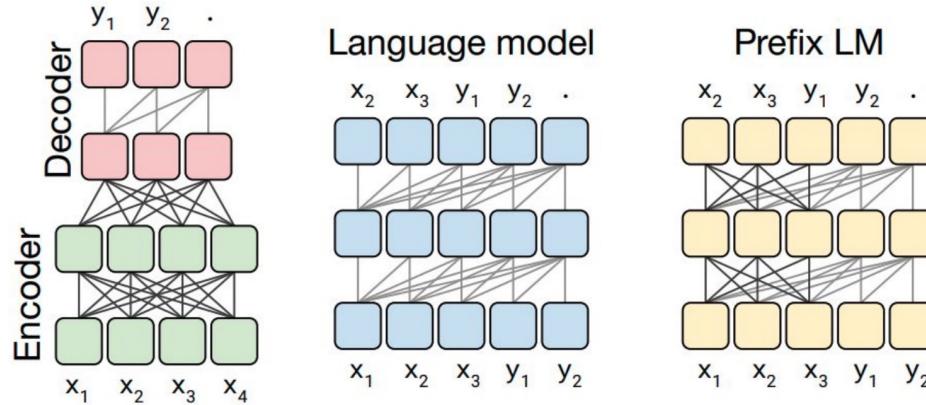
- TBD
- There is work on unifying various pre-training objectives.
 - Intuitively, different downstream tasks benefit from different objectives.



Architectural choices

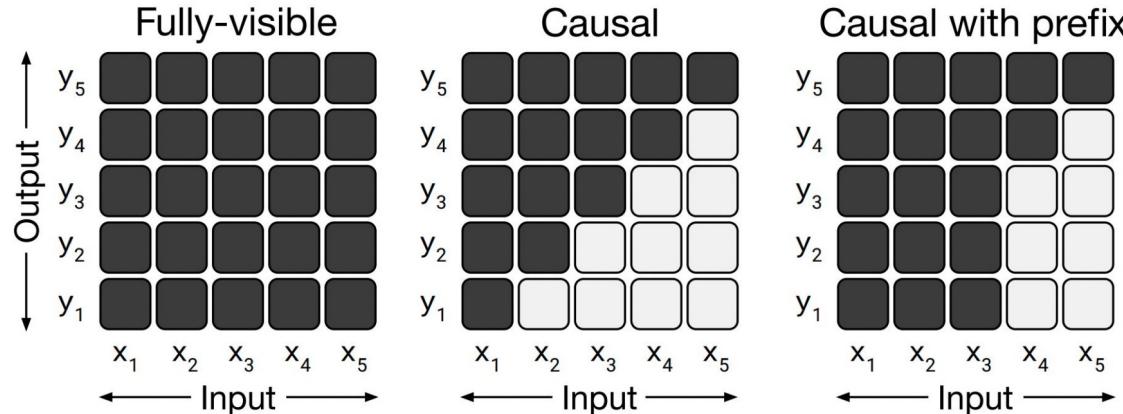


Architectures: Different Choices



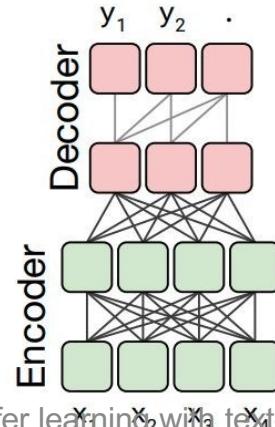
Architectures: Different Attention Masks

- **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

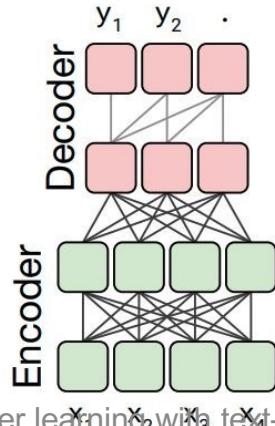
Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	83.28	19.24	80.88	71.36	26.98	39.82	27.65



Architectural Variants: Experiments

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

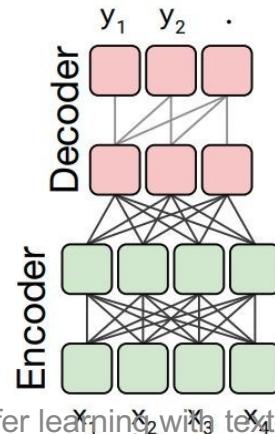
Input: Thank you for <X> me to your party
<Y>. Target: <X> inviting <Y> last week.



Architectural Variants: Experiments

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

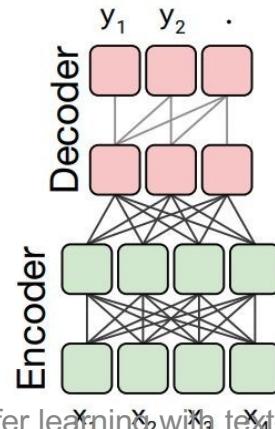
Number of parameters



Architectural Variants: Experiments

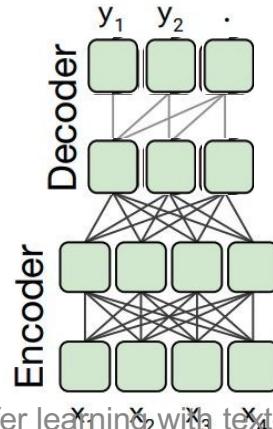
Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	83.28	19.24	80.88	71.36	26.98	39.82	27.65

Number of FLOPS



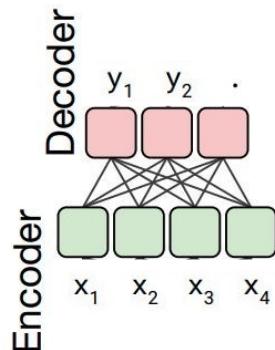
Architectural Variants: Experiments

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



Architectural Variants: Experiments

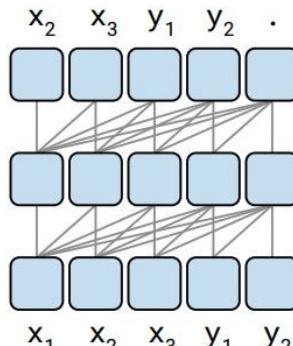
Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	83.28	19.24	80.88	71.36	26.98	39.82	27.65
Enc-dec, shared	Denoising	P	M	82.81	18.78	80.63	70.73	26.72	39.03	27.46
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95



Architectural Variants: Experiments

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

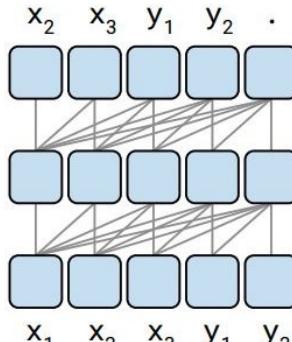


Architectural Variants: Experiments

Architecture	Objective	Params	Cost	GLUE	CNNNDM	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

Language model

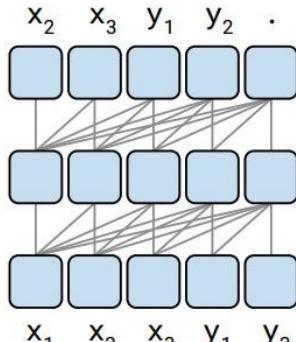


Architectural Variants: Experiments

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.

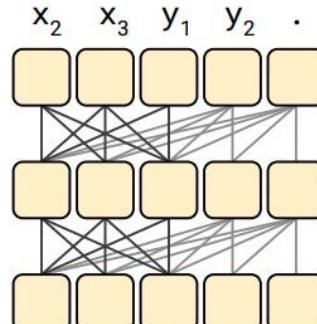
Language model



Architectural Variants: Experiments

Architecture	Objective	Params	Cost	GLUE	CNNNDM	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

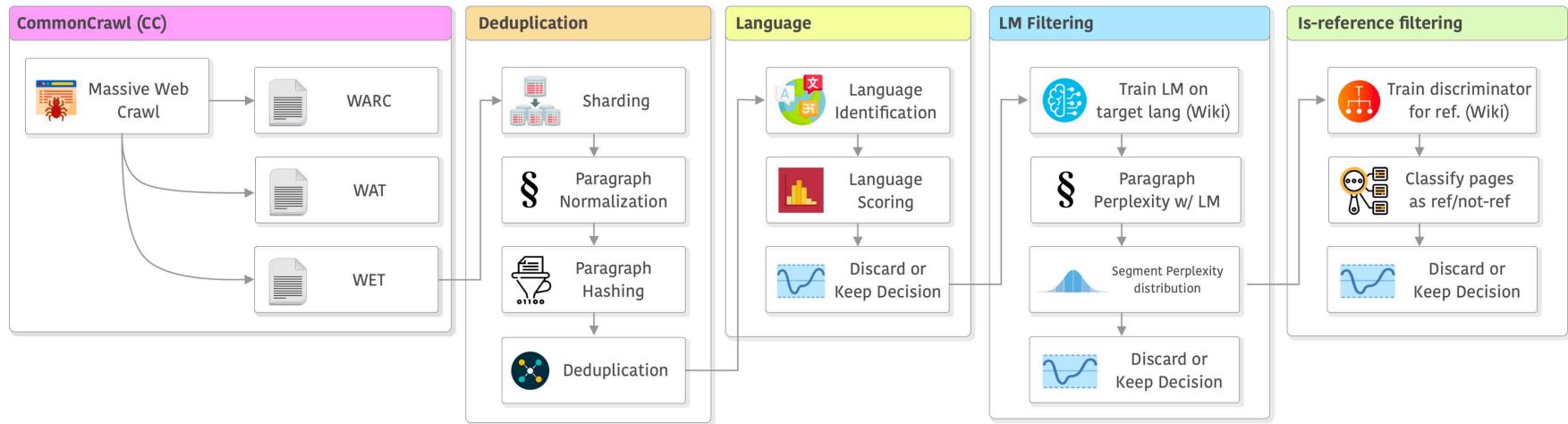


Architectural Variants: Experiments

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:
 - Halving the number of layers in encoder and decoder hurts the performance.
 - Performance of Encoder and Decoder with shared parameters is better than decoder only LM and prefix LM.

TODO: Elaborate the pipeline for dev of LMs



https://twitter.com/tarantulae/status/1650170087708454913?t=ncWWY0FI0tYC_dYLs76r5g&s=19

- Use the figures here to show the students how parameters are occupied in GPU memory during inference: <https://arxiv.org/pdf/2309.06180.pdf>
- Also discuss, how to select the batch size in GPUs.

Move it! What is scaling?

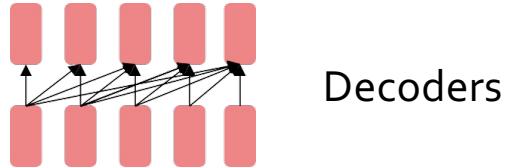
$$a \times \text{Model size } \text{ (robot icon)} \times \text{Training data } \text{ (book icon)} = \text{Training compute } \text{ (laptop icon)}$$

	Model size (# parameters)	Training data (# tokens)	Training compute (FLOPs)	Resources
 BERT-base (2018)	109M	250B	1.6e20	64 TPU v2 for 4 days (16 V100 GPU for 33 hrs)
 GPT-3 (2020)	175B	300B	3.1e23	~1,000x BERT-base
 PaLM (2022)	540B	780B	2.5e24	6k TPU v4 for 2 months

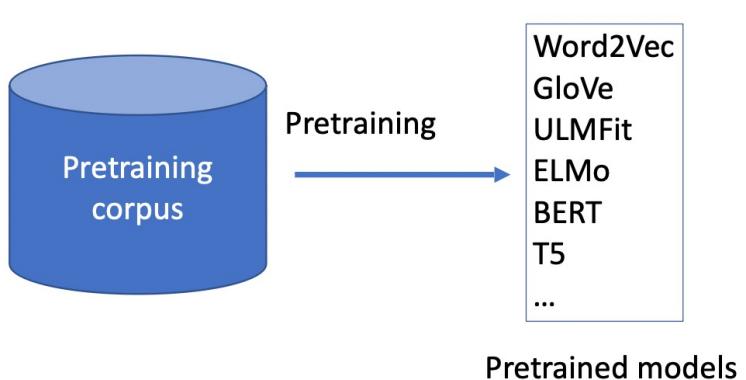


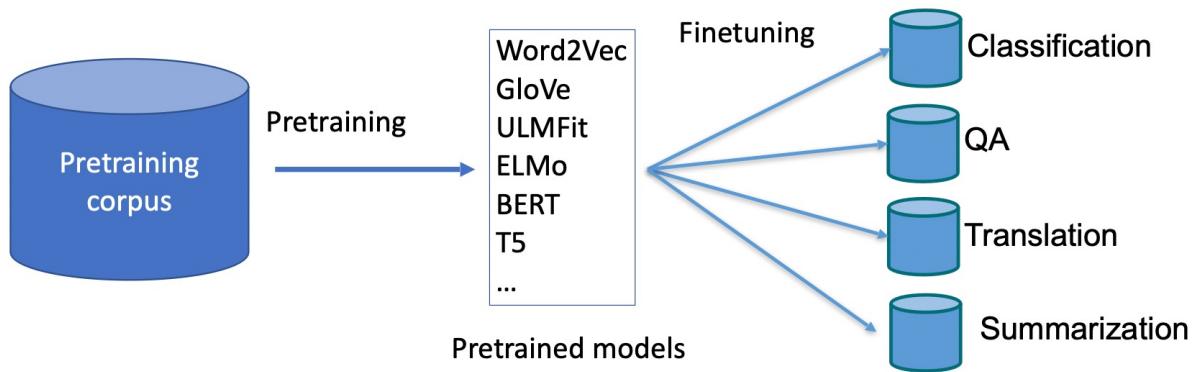
BERT: Pre-training of Deep Bidirectional Transformers
for Language Understanding (2018)
Language Models are Few-Shot Learners (2020)
PaLM: Scaling Language Modeling with Pathways (2022)

https://www.youtube.com/watch?v=Kp5R6GZh8O0&ab_channel=SimonsInstitute

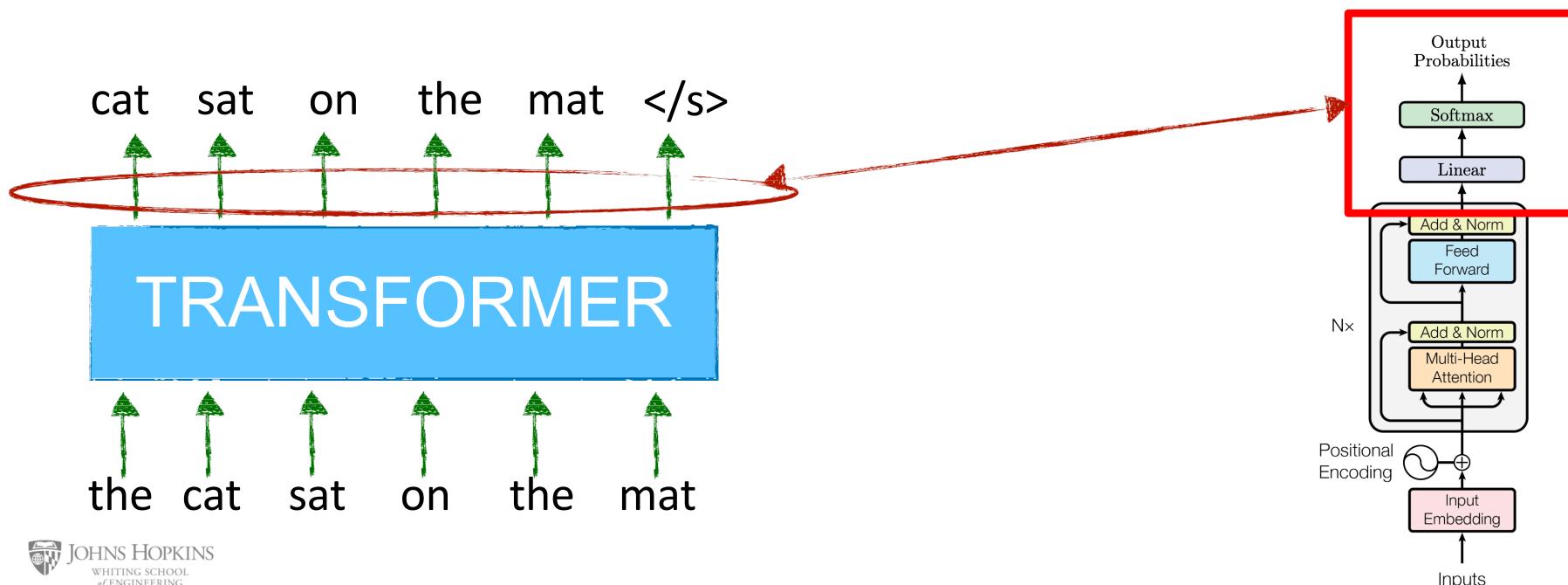


Adapting LLMs

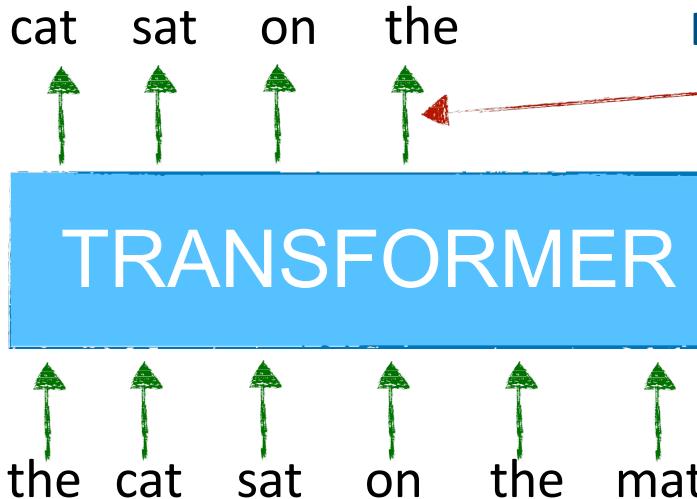




How to train models on language modeling?

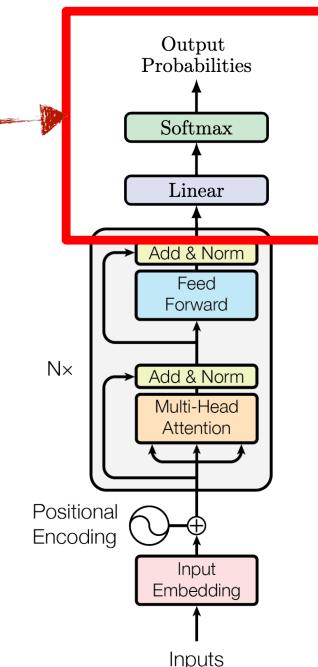


How to train models on language modeling?

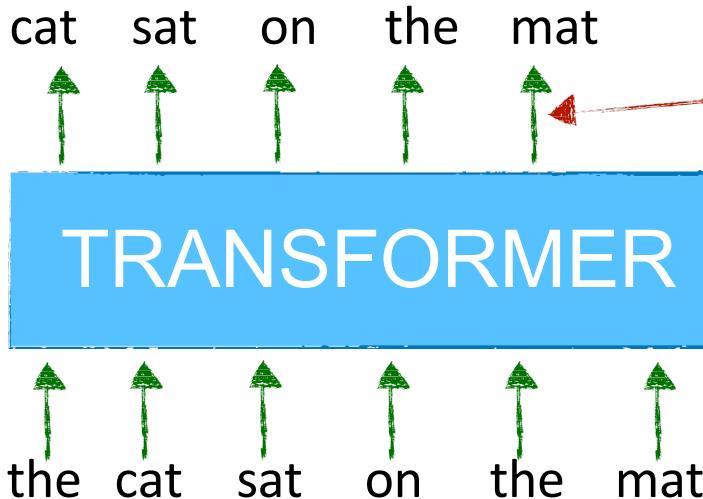


One mini optimization

Instead of passing all outputs to the final linear layer, just pass the last word

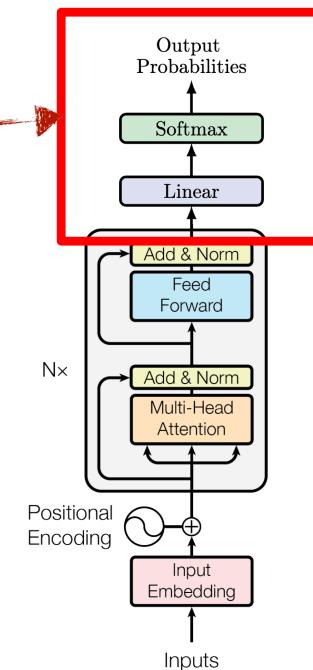


How to train models on language modeling?

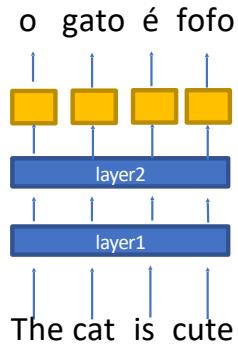


One mini optimization

Instead of passing all outputs to the final linear layer, just pass the last word

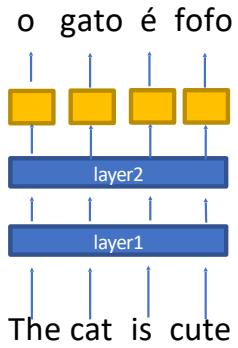


How to use a transformer for NLP tasks

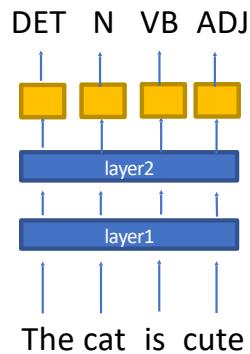


Translation

How to use a transformer for NLP tasks

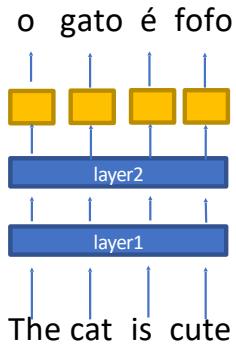


Translation

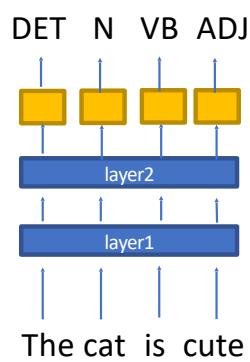


POS Tagging

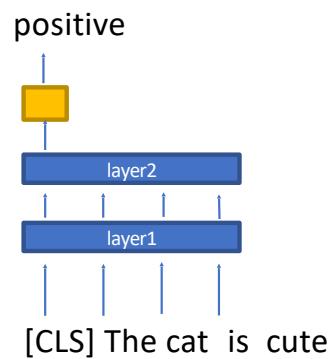
How to use a transformer for NLP tasks



Translation

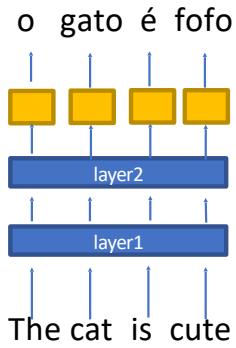


POS Tagging

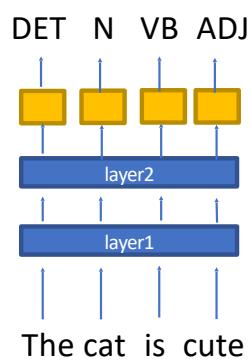


Text classification

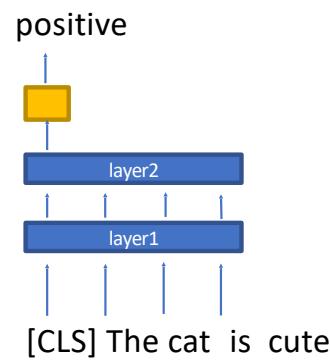
How to use a transformer for NLP tasks



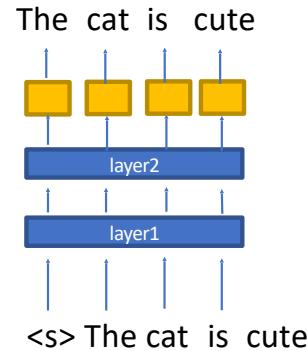
Translation



POS Tagging



Text classification



Language modeling

TODO

- Worth adding a short chapter on: how models resolve conflicts between their parameters and context?
 - [2310.15910.pdf \(arxiv.org\)](https://arxiv.org/pdf/2310.15910.pdf)
 - Orion has a paper
 - Eunsol has multiple papers too

- On Transformers: [lec15 \(princeton-nlp.github.io\)](https://lec15.princeton-nlp.github.io)