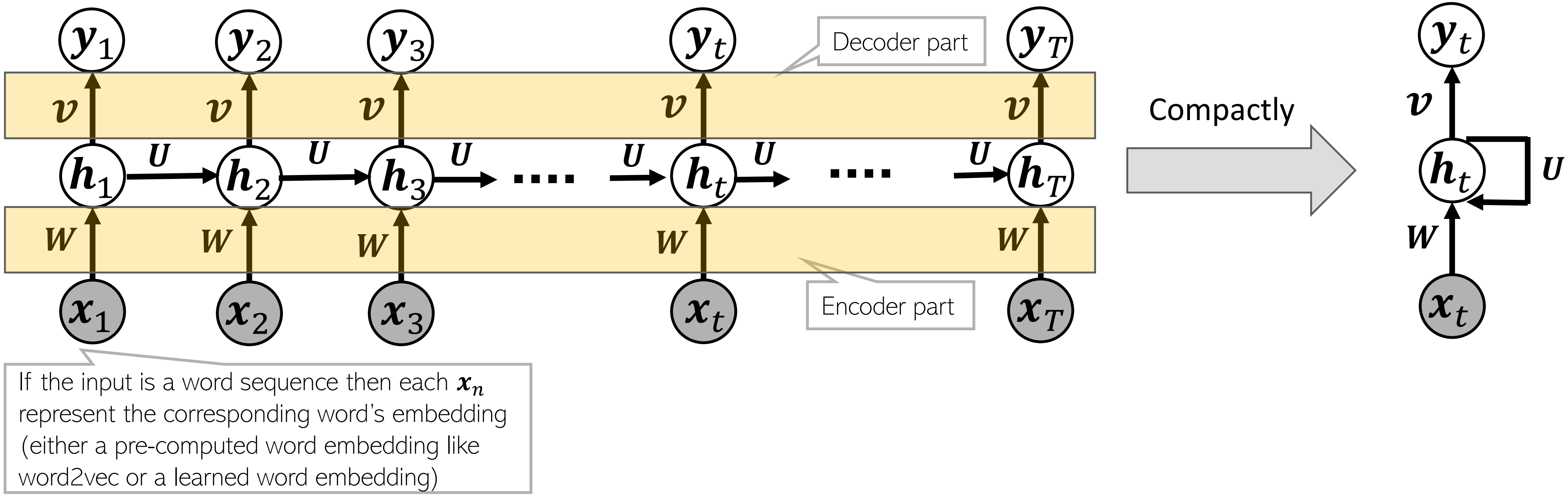
Deep Neural Networks:
Attention Mechanism and Transformers

CS771: Introduction to Machine Learning

Piyush Rai

Recap: RNNs

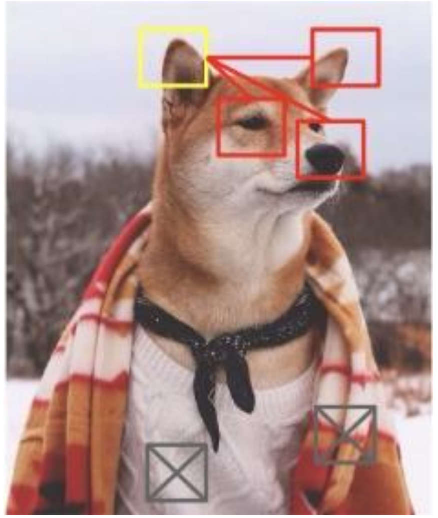
- RNNs are used when each input or output or both are sequences of tokens

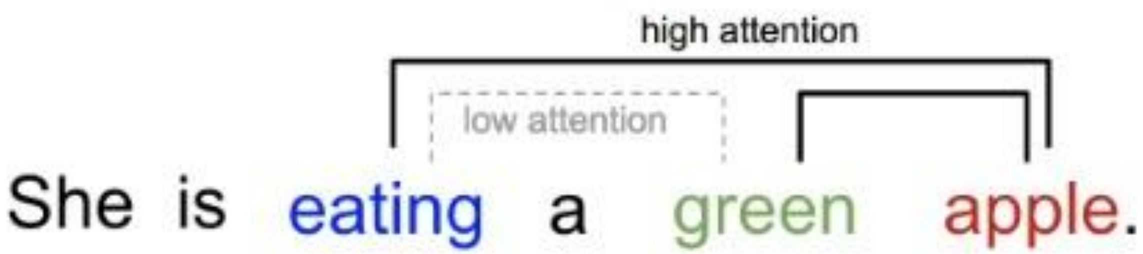


- Hidden state $\boldsymbol{h}_t$ is supposed to remember everything up to time $t-1$. However, in practice, RNNs have difficulties remembering the distant past
  - Variants such as LSTM, GRU, etc mitigate this issue to some extent
- Slow processing is another major issue (e.g., can't compute $\boldsymbol{h}_t$ before computing $\boldsymbol{h}_{t-1}$)

Attention

- We use attention to “focus” on some part of interest in an input
  - Other nearby relevant parts help us focus
  - Other irrelevant parts do not contribute in the process





- In sequence modeling problems, we can use attention between input and output tokens (between encoder and decoder parts), as well as among the inputs only (only within the encoder part)

Pic credit: https://lilianweng.github.io/posts/2018-06-24-attention/

RNN with Attention

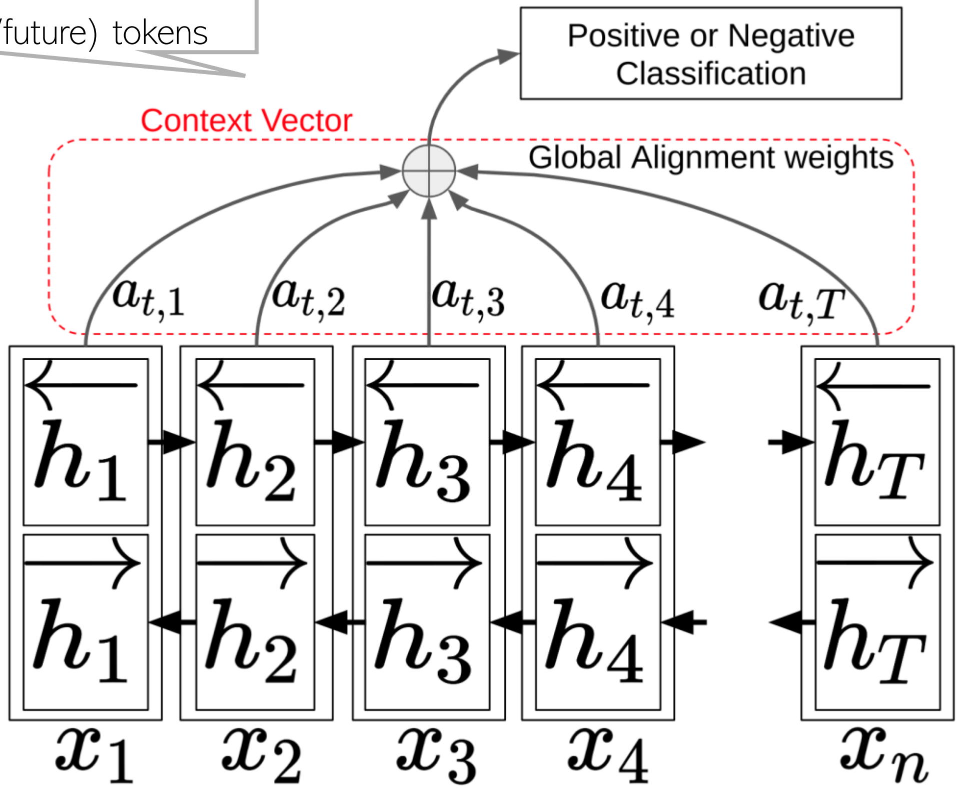
- RNNs have also been augmented with attention to help remember the distant past

- Attention mechanism for a bi-directional RNN encoder-decoder model

Its computation depends the embeddings of all (past/future) tokens

$$\boldsymbol{h}_i = \begin{bmatrix} \overrightarrow{\boldsymbol{h}}_i \\ \overleftarrow{\boldsymbol{h}}_i \end{bmatrix}$$

$$\boldsymbol{c}_t = \sum_{i=1}^{T} \alpha_{t,i} \boldsymbol{h}_i$$



Output: Classification

Attention (Additive) layer: parameterized by a simple feed-forward network

Encoder: bidirectional RNN

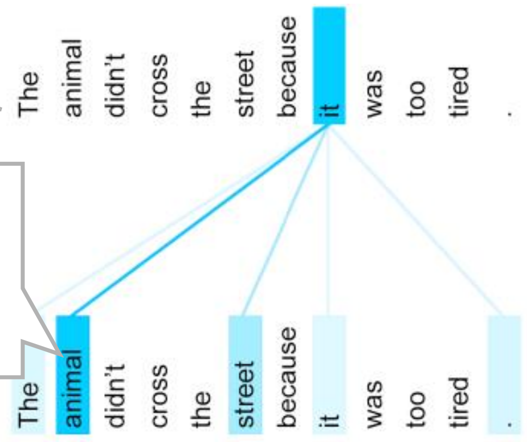
Pic source: https://matthewmcateer.me/blog/getting-started-with-attention-for-classification/

Self-Attention

- With self-attention, each token $\boldsymbol{x}_n$ can "attend to" all other tokens of the same sequence when computing this token's embedding $\boldsymbol{h}_n$

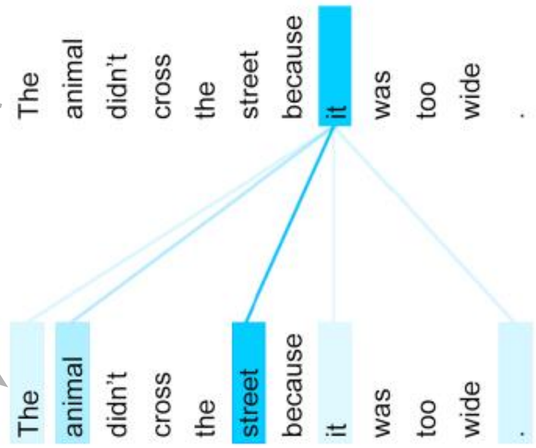
An input sequence

How much the word "it" is being "attended to" by other words in the input sequence



The input sequence but differing in the last word (tired -> wide)

Note how the attention has changed



- Attention helps capture the context better and in a much more "global" manner
  - "Global": Long ranges captures and in both directions (previous and ahead)

Self-Attention

Provided in form of a D-dim embedding (e.g., word2vec)

- For an N length sequence, the attention scores for each token $\boldsymbol{x}_n$ are computed using
  - A query vector $\boldsymbol{q}_n$ associated with that token
  - N key vectors $\boldsymbol{K} = \{\boldsymbol{k}_1, \boldsymbol{k}_2, \ldots, \boldsymbol{k}_N\}$ (one per token)
  - N value vectors $\boldsymbol{V} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_N\}$ associated with the key vectors

$N \times D$ matrix of original embeddings from the input layer

Linear projection by $D \times d$ matrix

Row n is $\boldsymbol{q}_n$

$N \times d$ matrix of "queries"

Learnable

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q$$

$D \times d$ matrix. Learnable

Assuming same size d as query

$N \times d$ matrix of N "keys"

$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K$$

- One way to compute the attention score is

How much token i attends to token n

$$\alpha_{n,i} = \frac{\exp(\boldsymbol{q}_n^\top \boldsymbol{k}_i)}{\sum_{j=1}^N \exp(\boldsymbol{q}_n^\top \boldsymbol{k}_j)}$$

Dot-product attention (query and key assumed d dimensional)

$N \times K$ matrix of "value" vectors of the N keys

D ×K matrix. Learnable

$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V$$

$\boldsymbol{Q}$ and $\boldsymbol{K}$ are assumed $N \times d$

$N \times v$

- Given attention scores, encoder's hidden state for $\boldsymbol{x}_n$ is

Attention-weighted sum of the value vectors of all the tokens in the sequence

$$\boldsymbol{h}_n = \sum_{i=1}^N \alpha_{n,i}\, \boldsymbol{v}_i$$

Thus the encoding of x_n depends on all the tokens in the sequence

$N \times v$

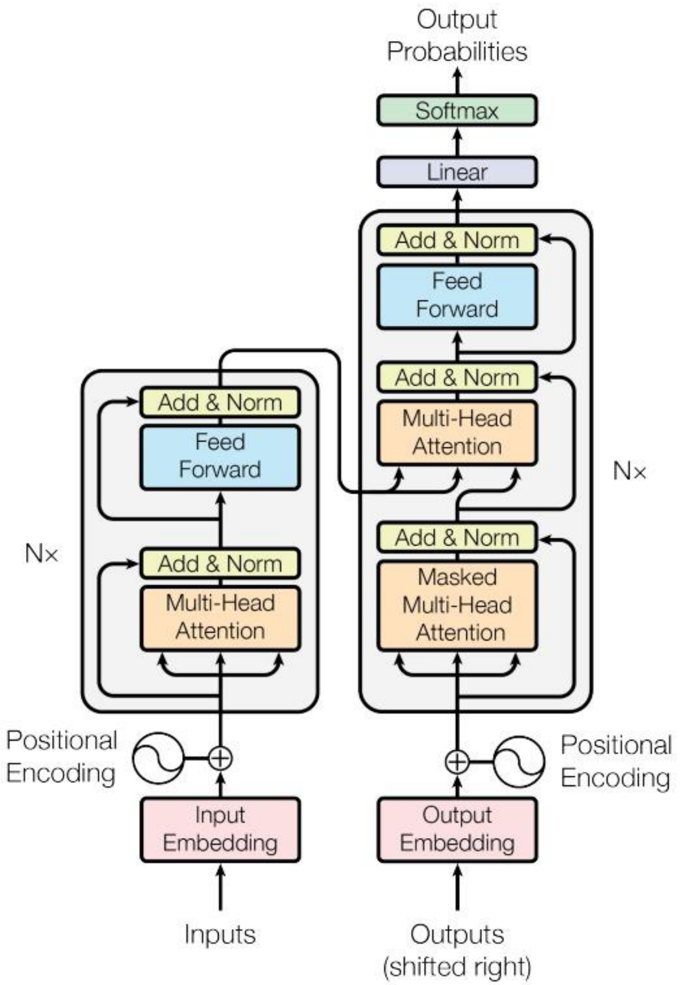
$$\boldsymbol{H} = \mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V}$$

Dividing by $\sqrt{d}$ ensures variance of the dot product is 1

"Scaled" dot-product attention

Transformers

- Transformers also use the idea of attention
  - "self-attention" over all input tokens
  - "self-attention" over each output token and previous tokens
  - "cross-attention" between output tokens and input tokens
- Transformer also compute embeddings of all tokens in parallel
- Transformers are based on the following key ideas*
  - "Self-attention" and "cross-attention" for computing the hidden states
  - Positional encoding
  - Residual connections
- Attention helps capture the context better and in a much more "global" manner in sequence data



"Attention is all you need" (Vaswani et al, 2017)

Positional Encoding

- Transformers also need a "positional encoding" for each token of the input since they don't process the tokens sequentially (unlike RNNs)
- Let $\boldsymbol{p}_i \in \mathbb{R}^d$ be the positional encoding for location $\boldsymbol{i}$. One way to define it is

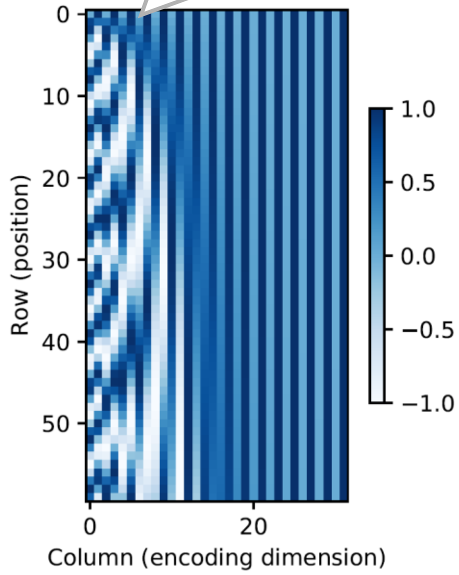
Here C denotes the maximum possible length of a sequence

$$p_{i,2j} = \sin\left(\frac{i}{C^{2j/d}}\right), \quad p_{i,2j+1} = \cos\left(\frac{i}{C^{2j/d}}\right)$$

Positional encoding vector for location i assuming $d = 4$

$$\boldsymbol{p}_i = [\sin(\frac{i}{C^{0/4}}), \cos(\frac{i}{C^{0/4}}), \sin(\frac{i}{C^{2/4}}), \cos(\frac{i}{C^{2/4}})]$$

Note the smooth transition as the position index changes



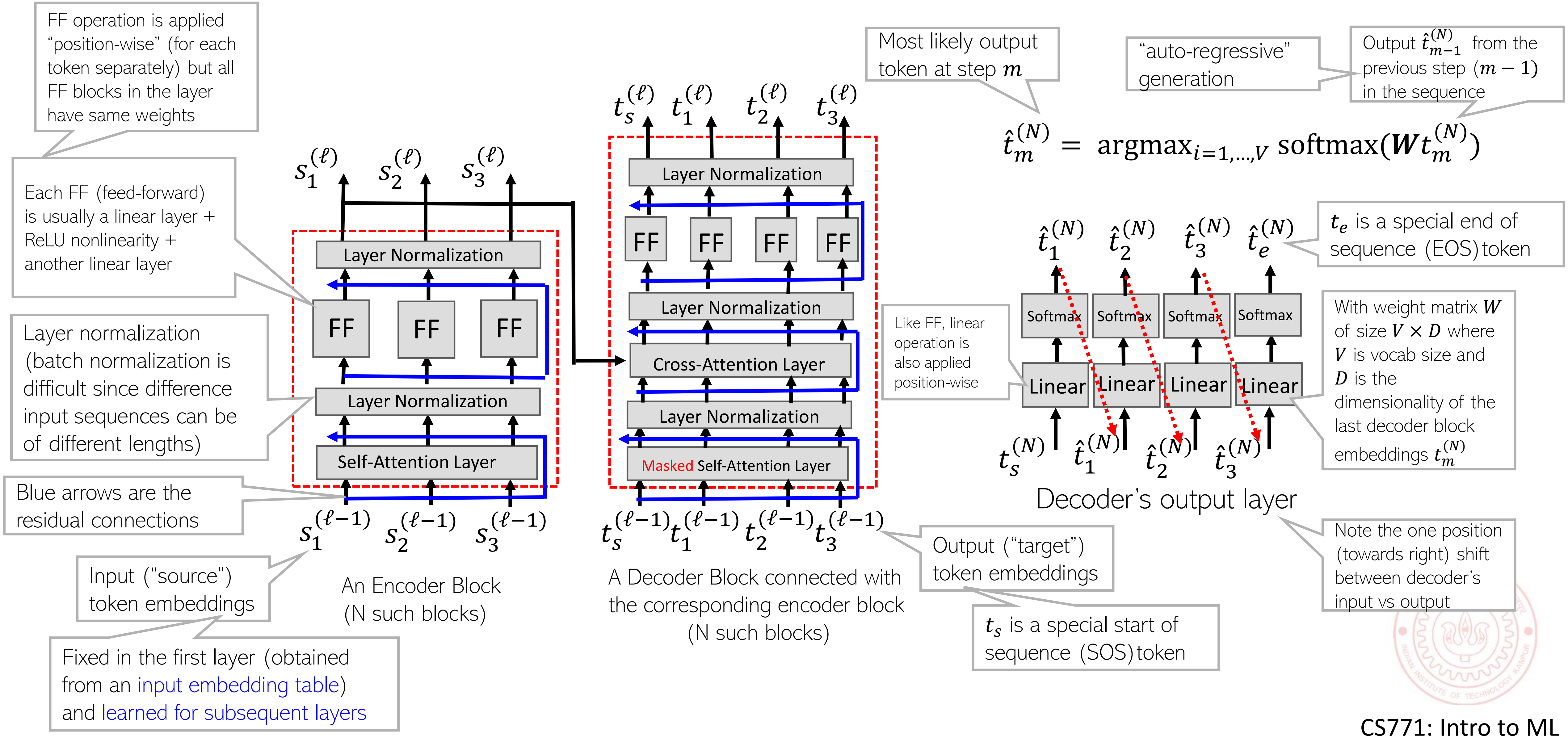
- Given the positional encoding, we add them to the token embedding

$$\hat{\boldsymbol{x}}_i = \boldsymbol{x}_i + \boldsymbol{p}_i$$

- The above positional encoding is pre-defined but can also be learned

Pic credit: PML-1 (Murphy, 2023)

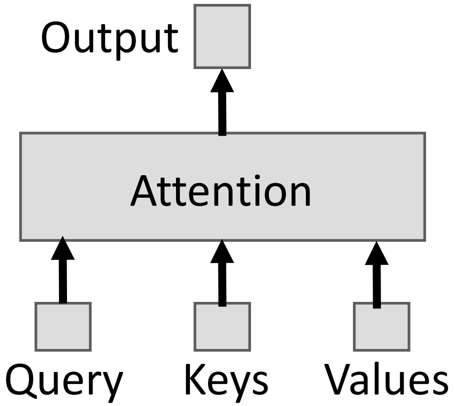
Zooming into the encoder and the decoder..



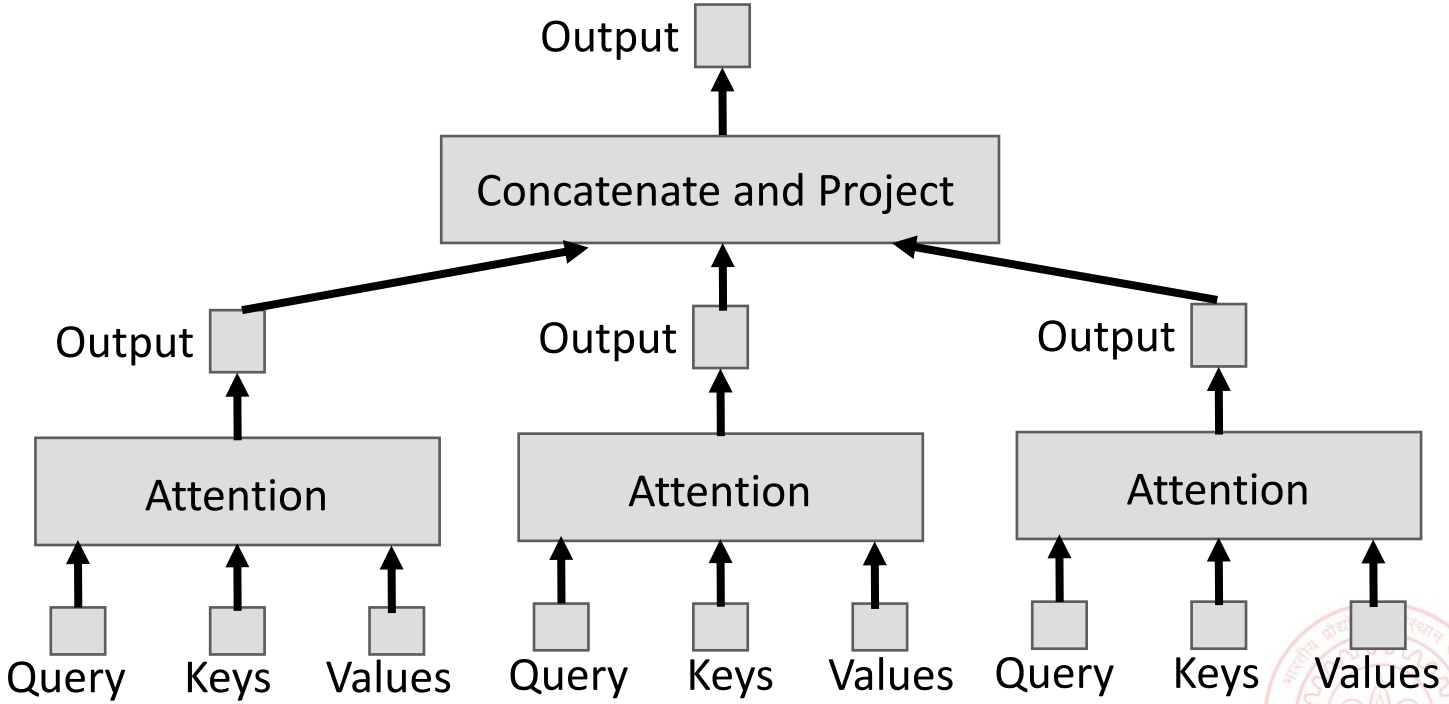
Multi-head Attention (MHA)

- A single attention function can capture only one notion of similarity
- Transformers therefore use multi-head attention (MHA)

Single Attention

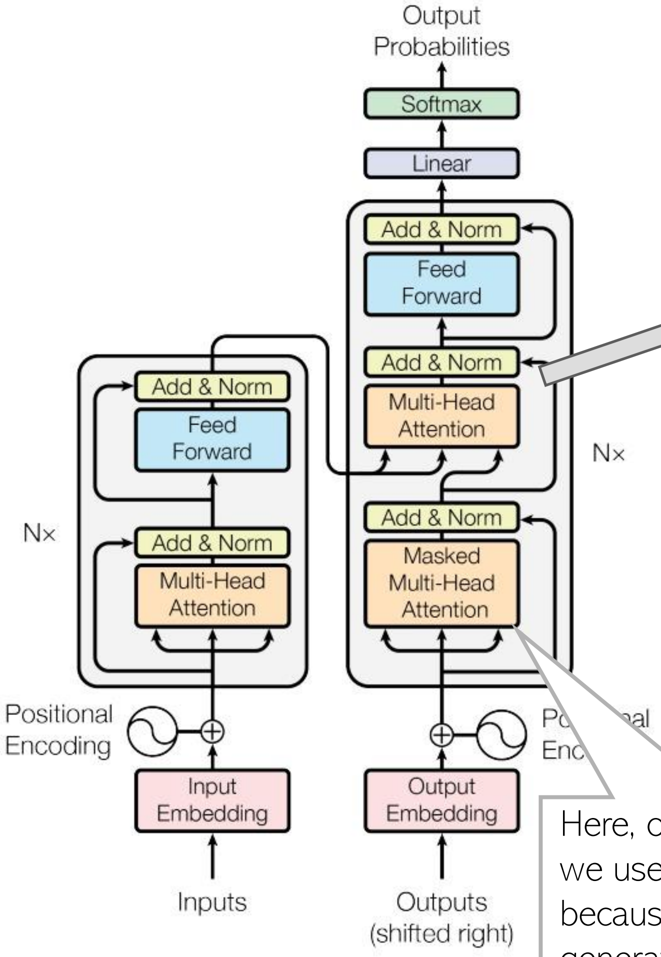


MHA

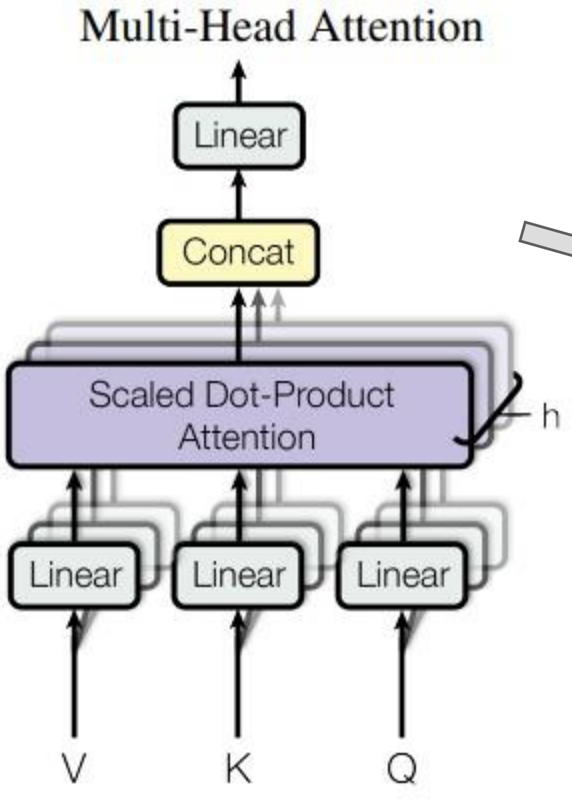


Pic credit: https://huggingface.co/learn/nlp-course/chapter1/

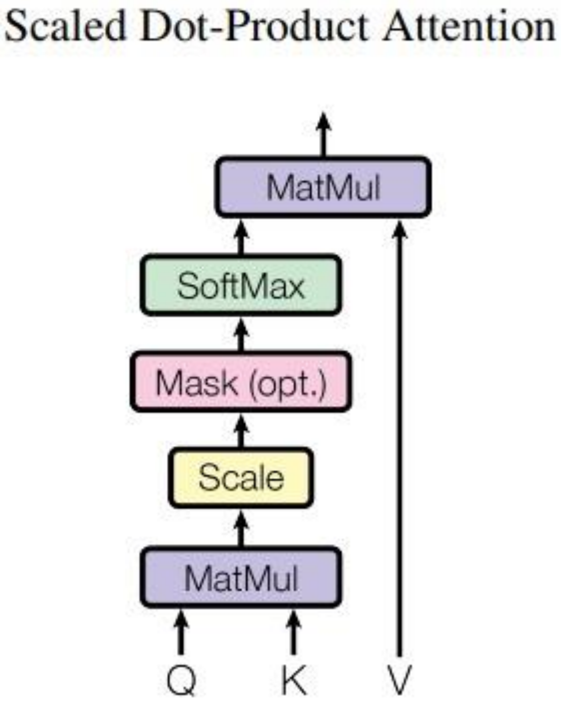
(Masked) Multi-head Attention (MHA)



Here, on the output side, we used "masked" MHA because during output generation, we don't want to look at future tokens



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



Pic source: "Attention is all you need" (Vaswani et al, 2017)

Computing Attention Efficiently

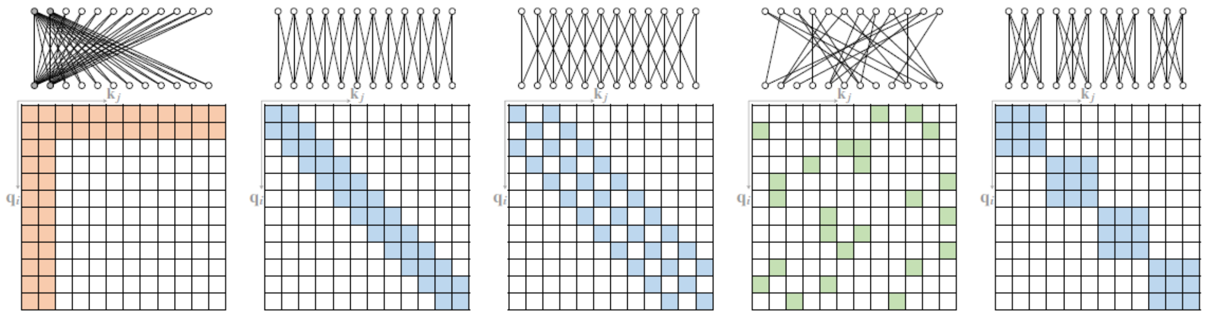
- The standard attention mechanism is inefficient for large sequences

$O(T^2)$ storage and computation cost for a T length sequence

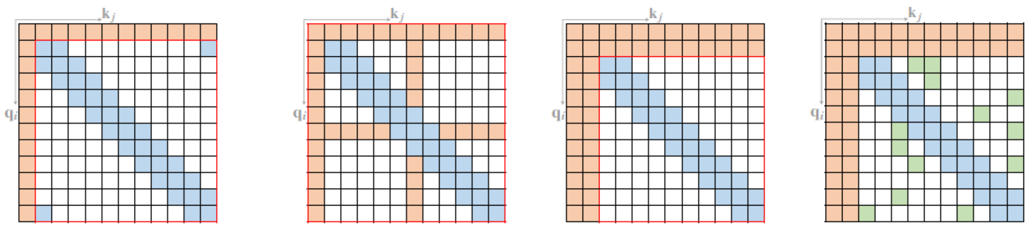
$$\boldsymbol{H} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V}$$

- Many ways to make it more efficient, e.g.,

Sparse Attention



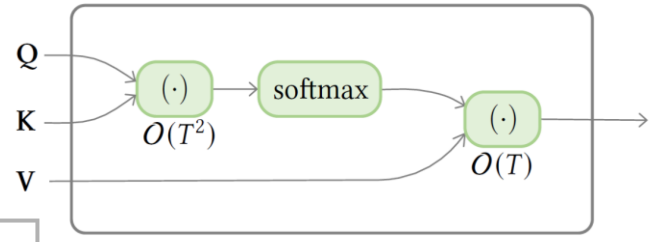
(a) global (b) band (c) dilated (d) random (e) block local



(a) Star-Transformer (b) Longformer (c) ETC (d) BigBird

Linearized Attention

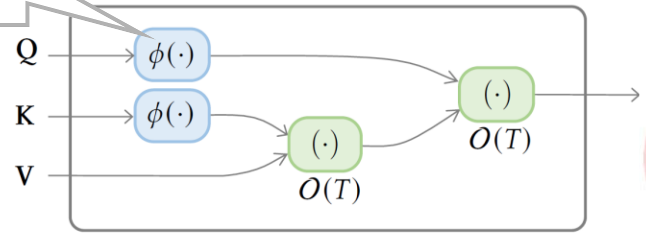
$$\exp(\boldsymbol{Q}\boldsymbol{K}^\top) \approx \phi(\boldsymbol{Q})^\top \phi(\boldsymbol{K})$$



(a) standard self-attention

A nonlinear projection based on kernels

E.g., kernel random features projection

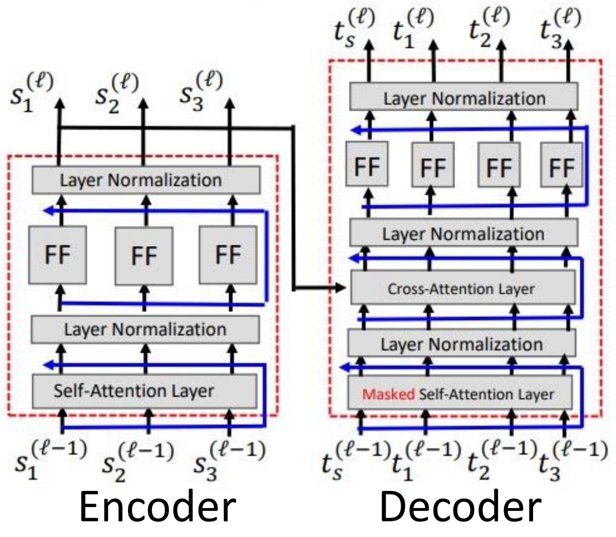


(b) linearized self-attention

Pic source: A Survey of Transformers (Lin et al, 2021)

Popular Transformer Variants: BERT and GPT

- The standard transformer architecture is an encoder-decoder model
- Some models use just the encoder <u>or</u> the decoder of the transformer
- BERT (Bidirectional Encoder Representations from Transformers)
  - Basic BERT can be learned to encoder token sequences
- GPT (Generative Pretrained Transformer)
  - Basic GPT can be used to <u>generate</u> token sequences similar to its training data

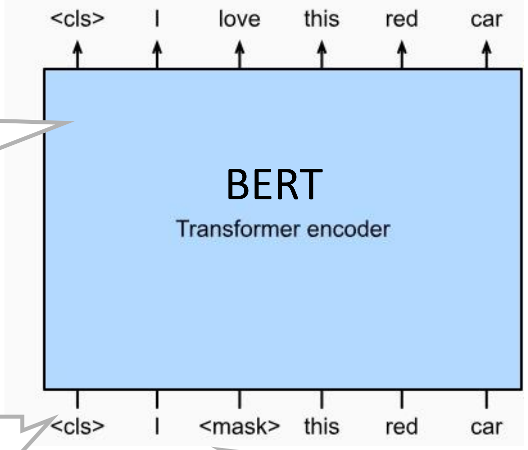


This encoder can be used for other tasks by fine-tuning

A transformer which contains only the encoder

Trained unsupervisedly using a missing token prediction objective

This is just start of sentence token

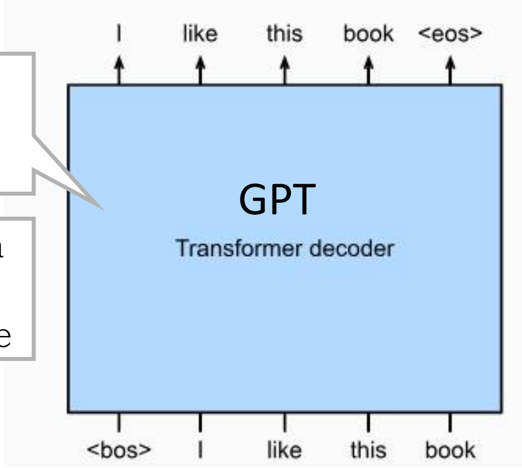


Missing token which BERT tries to predict

Also, no cross-attention since there is no encoder

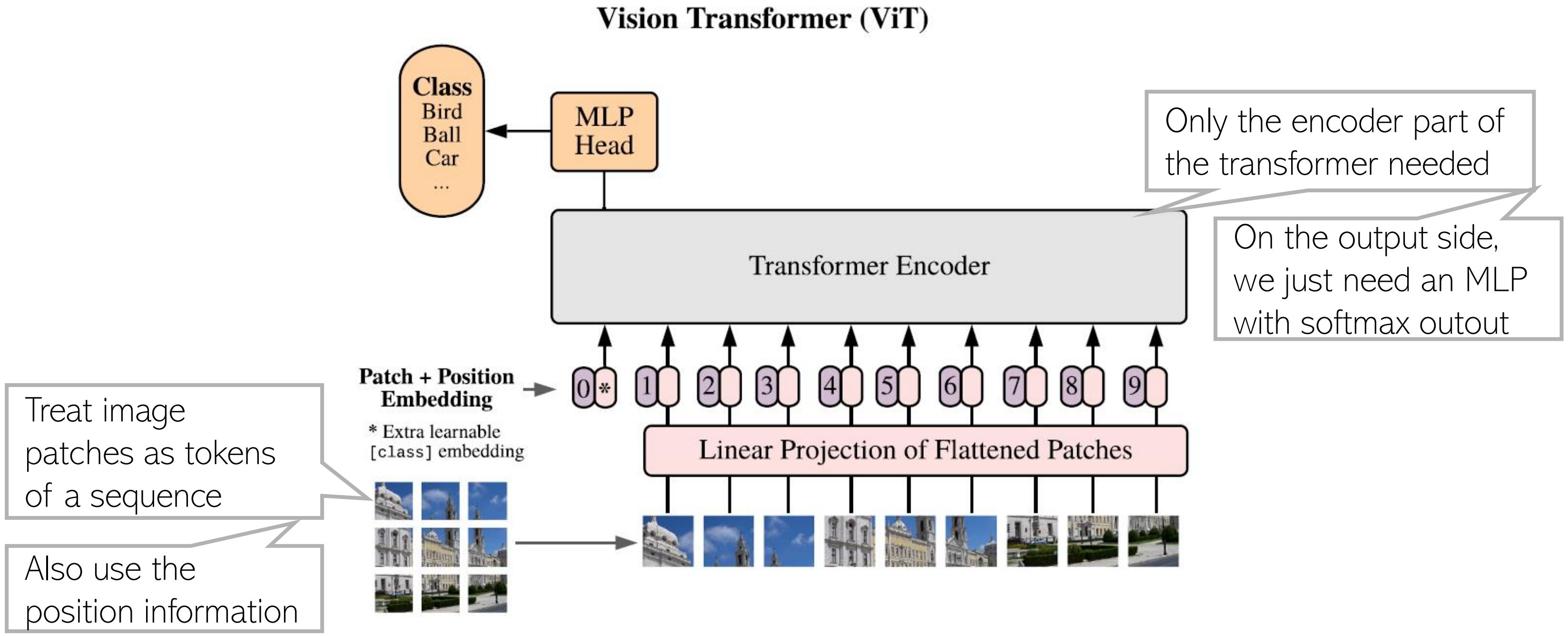
A transformer which contains only the decoder

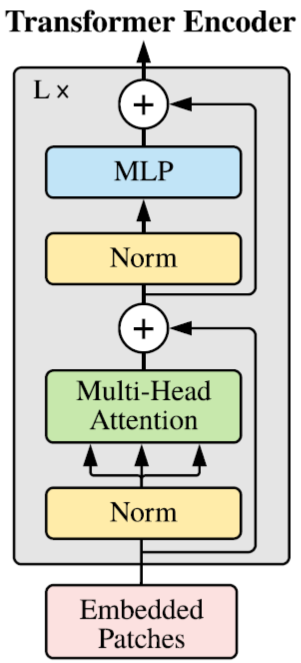
Pre-trained using a next token prediction objective



Transformers for Images: ViT

- Transformers can be used for images as well[#]. For image classification, it looks like this





- Early work showed ViT can outperform CNNs given very large amount of training data
- However, recent work* has shown that good old CNNs still rule! ViT and CNN perform comparably at scale, i.e., when both given large amount of compute and training data

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (Dosovitskiy et al, 2020)

*ConvNets Match Vision Transformers at Scale (Smith et al, 2023)