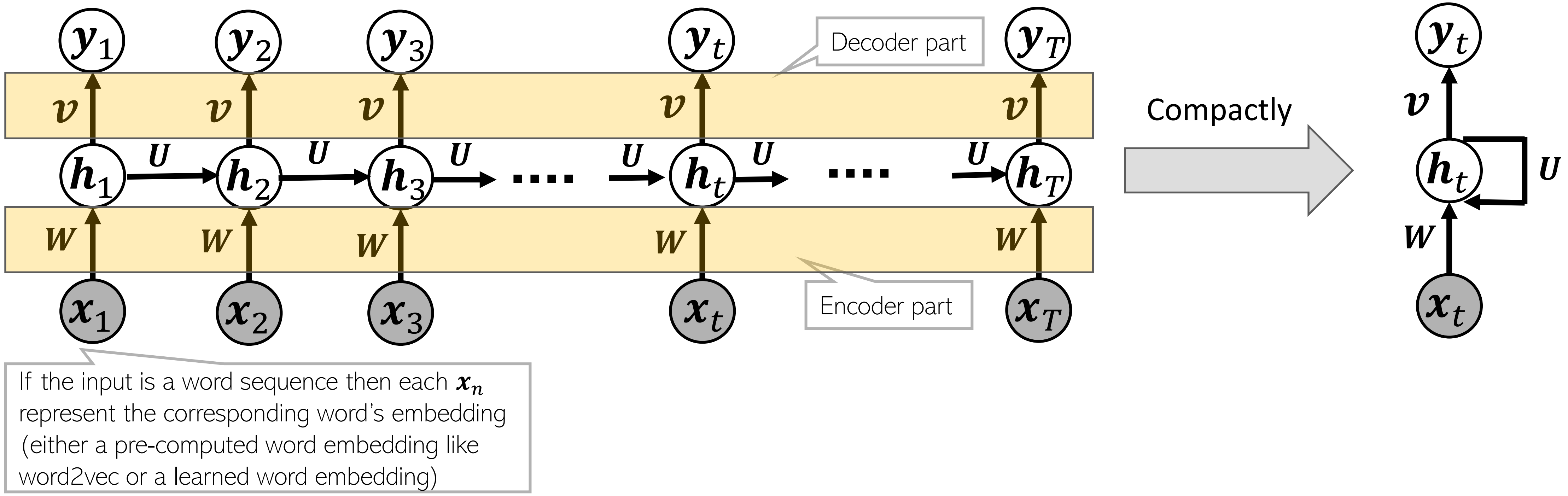
Deep Neural Networks:
Attention Mechanism and Transformers

CS771: Introduction to Machine Learning

Piyush Rai

Recap: RNNs

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

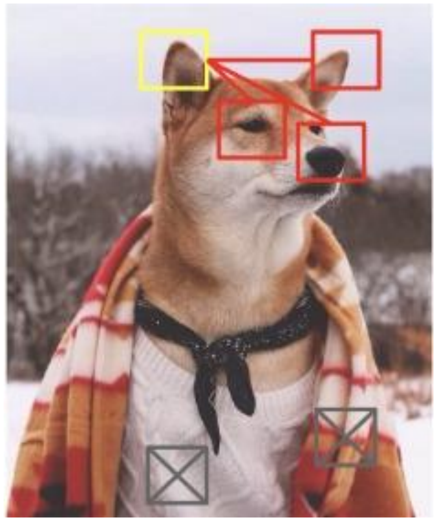


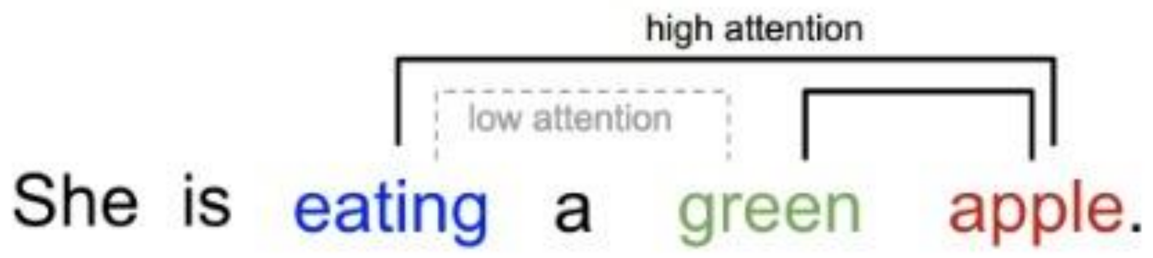
If the input is a word sequence then each $\boldsymbol{x}_n$ represent the corresponding word's embedding (either a pre-computed word embedding like word2vec or a learned word embedding)

- 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

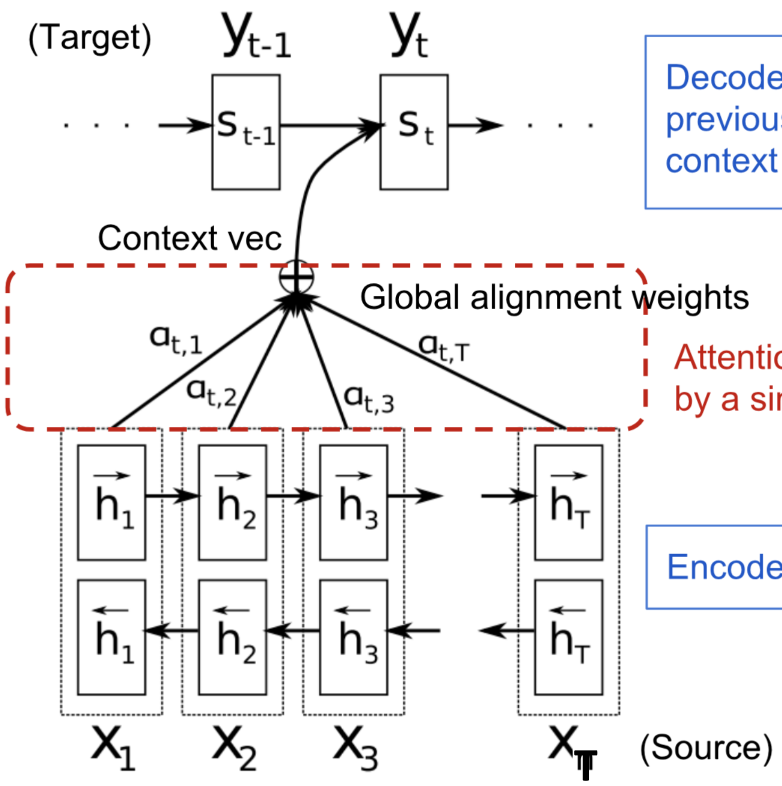
- RNNs have also been augmented with attention to help remember the distant past

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

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

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

$$s_t = f(s_{t-1}, c_t)$$



Decoder: RNN with input from previous state + dynamic context vector.

Decoder's output y_t at step t assumed to depend on the previous state s_{t-1} in the decoder and an input context vector c_t

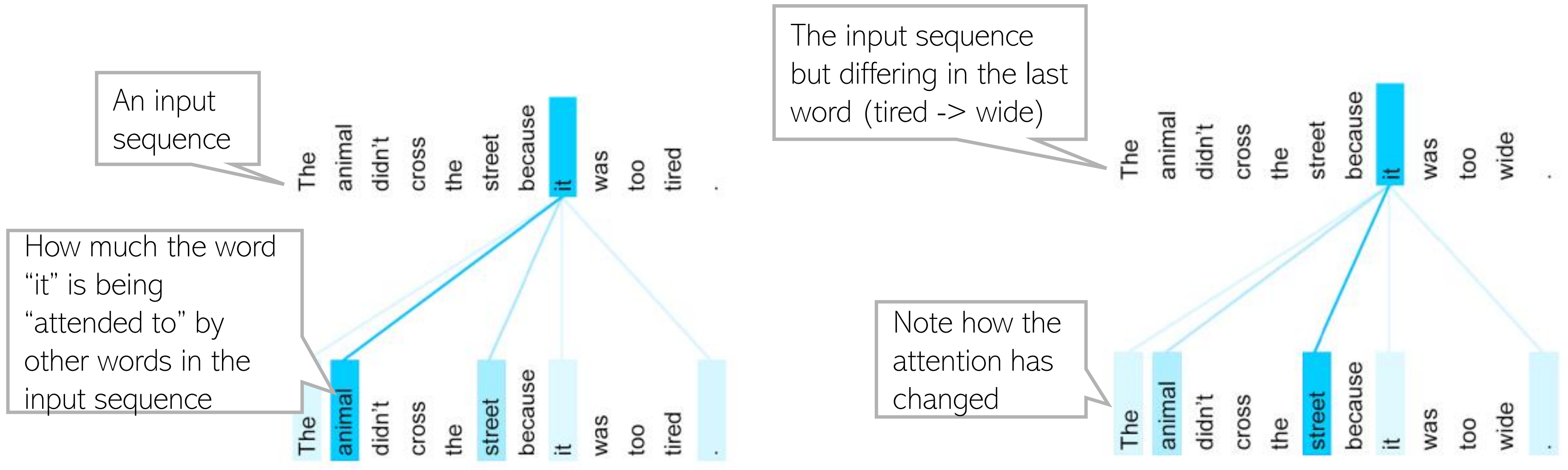
Attention layer: parameterized by a simple feed-forward network

Additive Attention

Encoder: bidirectional RNN

Self-Attention

- With self-attention, each token $\boldsymbol{x}_n$ can "attend to" all other tokens when the encoder computes its embedding $\boldsymbol{h}_n$



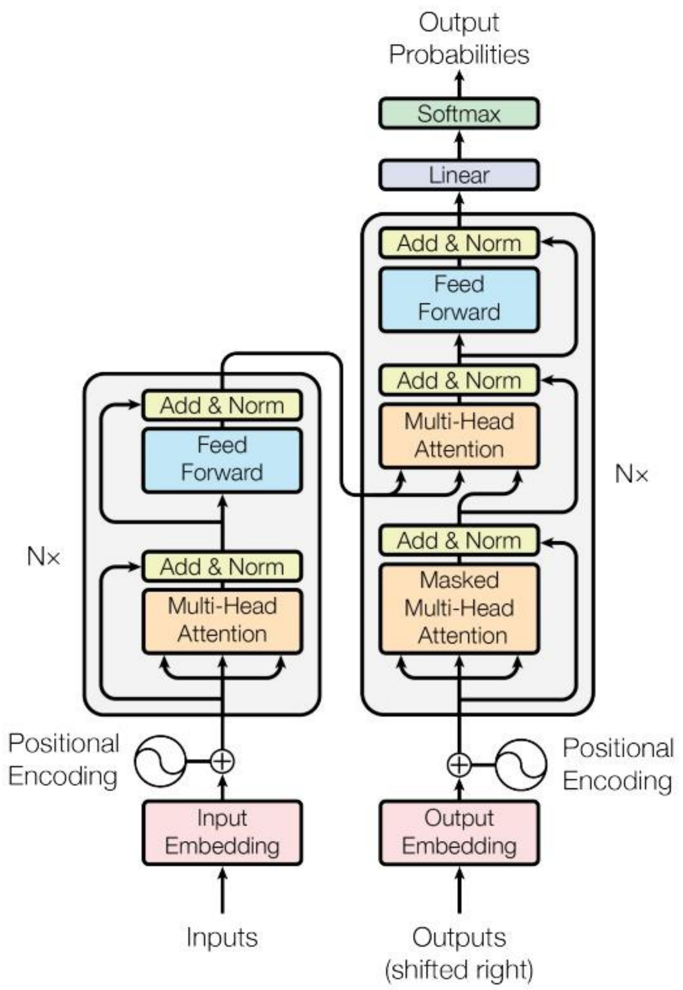
An input sequence
The animal didn't cross the street because it was too tired .
How much the word "it" is being "attended to" by other words in the input sequence
The animal didn't cross the street because it was too tired .
The input sequence but differing in the last word (tired -> wide)
The animal didn't cross the street because it was too wide .
Note how the attention has changed
The animal didn't cross the street because it was too wide .

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

Example credit: https://blog.research.google/2017/08/transformer-novel-neural-network.html

Transformers

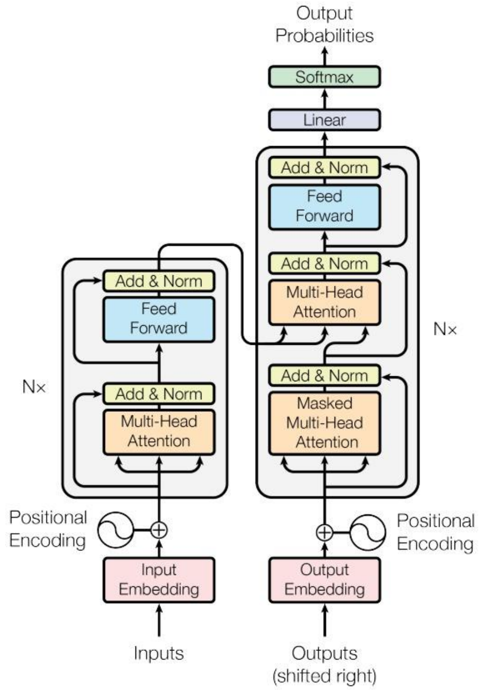
- Transformers also use the idea of attention
  - "self-attention" over all input tokens
  - "self-attention" over each output token and previous tokens
  - "cross-attention" between output tokens and input tokens
- Transformer also compute embeddings of all tokens in parallel
- Transformers are based 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)

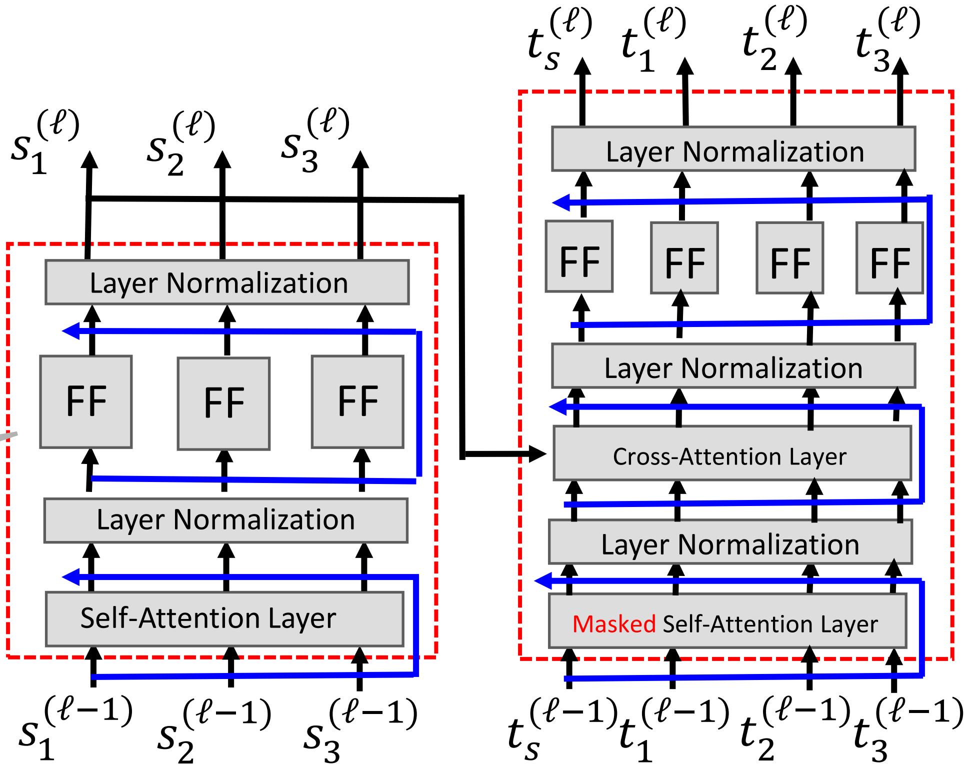
All Pieces..

- In the encoder and decoder, the blue arrows denote skip/residual connections
- s_1, s_2, s_3 denote the input ("source") token embeddings (assuming input sequence length is 3)
- t_s, t_1, t_2, t_3 denote the output ("target") token embeddings (assuming output sequence length is also 3 and t_s is a special start-of-sentence (SOS) token)
- Superscript ℓ denotes the layer ℓ. At layer 0, we have original token embeddings (with added positional encodings)
- In the output layer of decoder, at each step, we predict the most likely output token for that step ($\hat{t}_e$ denotes the special end-of-sentence (EOS) token)
- In the decoder's input layer, tokens in the output sequence are shown as shifted right by one position (because, in the output layer, the next token prediction depends on the previously predicted token in the output sequence)
- The feedforward (FF) and linear layers are applied position-wise (for each token separately)



Overall Transformer Architecture

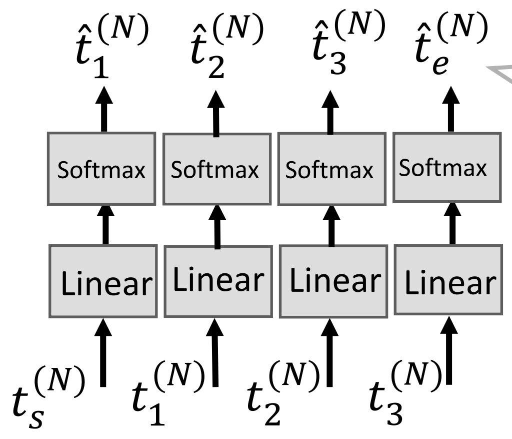
Each FF (feed-forward) in encoder and decoder blocks is usually a linear layer + ReLU nonlinearity + another linear layer



An Encoder Block (N such blocks)

A Decoder Block connected with the corresponding encoder block (N such blocks)

$$\hat{t}_m^{(N)} = \text{argmax}_{i=1,\ldots,V} \text{ softmax}(\boldsymbol{W} t_m^{(N)})$$



Note the one position (towards right) shift between decoder's input vs output

With weight matrix W of size $V \times D$ where V is vocab size and D is the dimensionality of the last decoder block embeddings $t_m^{(N)}$

Decoder's output layer

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 $\boldsymbol{K} = \{\boldsymbol{k}_1, \boldsymbol{k}_2, \dots, \boldsymbol{k}_N\}$ (one per token)
  - N value vectors $\boldsymbol{V} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_N\}$ associated with the key vectors

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

Linear projection by $D \times d$ matrix

Learnable

Row n is $\boldsymbol{q}_n$

$N \times d$ matrix of "queries"

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

$D \times d$ matrix. Learnable

Assuming same size d as query

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

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

$D \times$K matrix. Learnable

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

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

- 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)

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

- 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

$N \times v$

$N \times v$

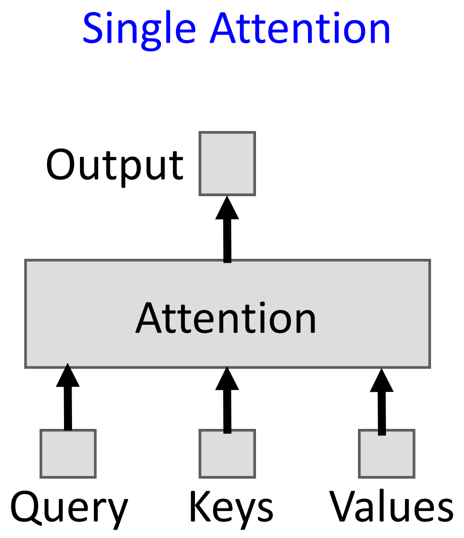
$$\boldsymbol{H} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V}$$

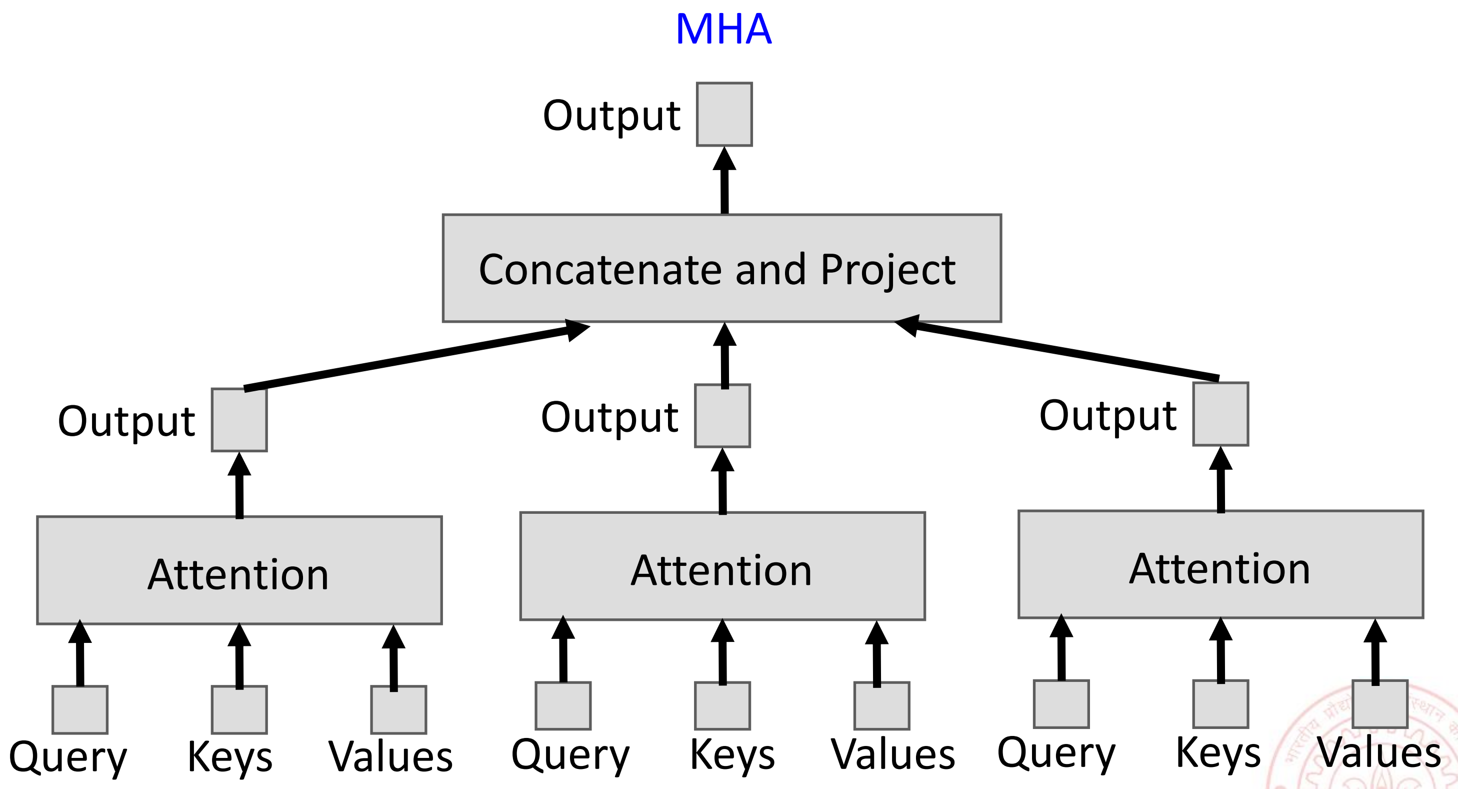
Dividing by $\sqrt{d}$ ensures variance of the dot product is 1

"Scaled" dot-product attention

Multi-head Attention (MHA)

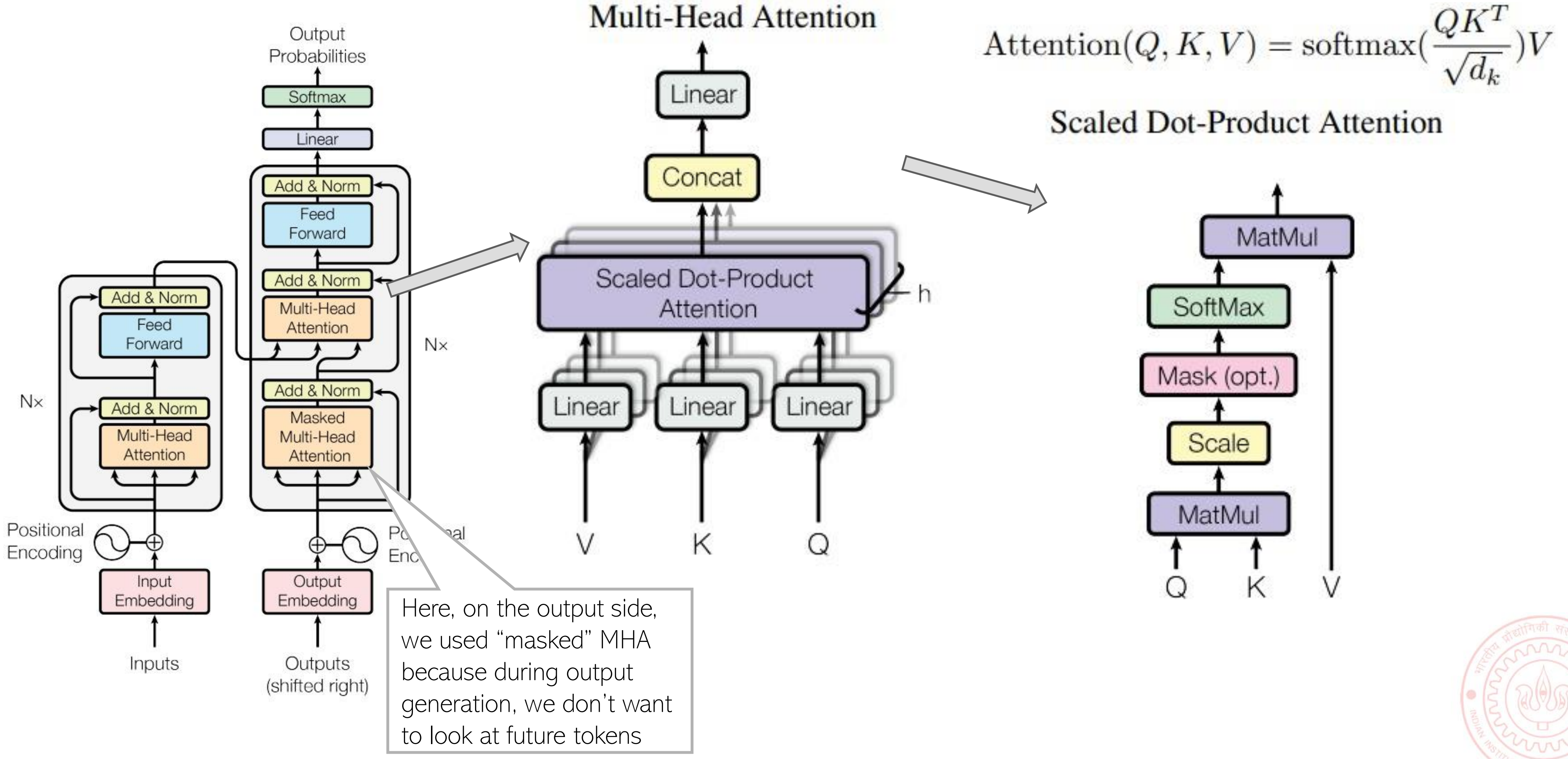
- A single attention function can capture only one notion of similarity
- Transformers therefore use multi-head attention (MHA)





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

(Masked) Multi-head Attention (MHA)



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

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

Pic source: "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

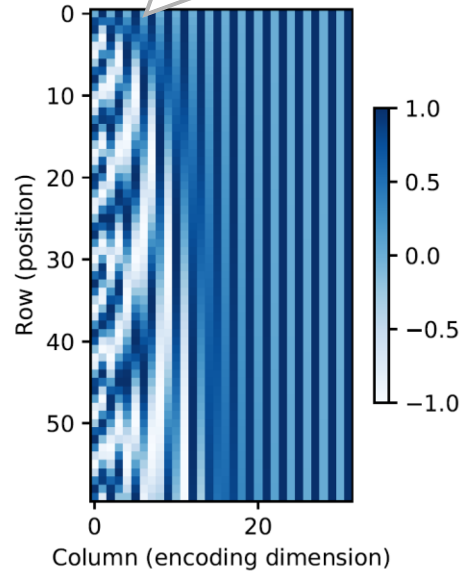
Here C denotes the maximum possible length of a sequence

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

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

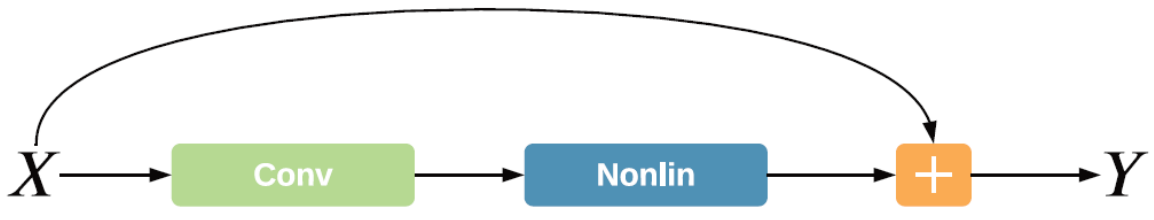
$$\hat{\boldsymbol{x}}_i = \boldsymbol{x}_i + \boldsymbol{p}_i$$

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

Pic credit: PML-1 (Murphy, 2023)

Residual Connections

- Transformers contain a very large number of layers
- In general, just stacking lots of layer doesn't necessarily help a deep learning model
  - Vanishing/exploding gradient may make learning difficult
- Skip connections or "residual connections" help if we want very deep networks
  - This idea was popularized by "Residual Networks"* (ResNets) which can have very large number of layers
- Basic idea: Don't force a layer to learn everything about a mapping
  - Let the layer just learn the "residual" and add the original input



The x to y mapping we want to learn using this layer

If the residual's and input's dimension are different then we apply a projection on the inputs to match the dimensions

$$Y = f(X;W) = g\big(\mathrm{conv}(X)\big) + X$$

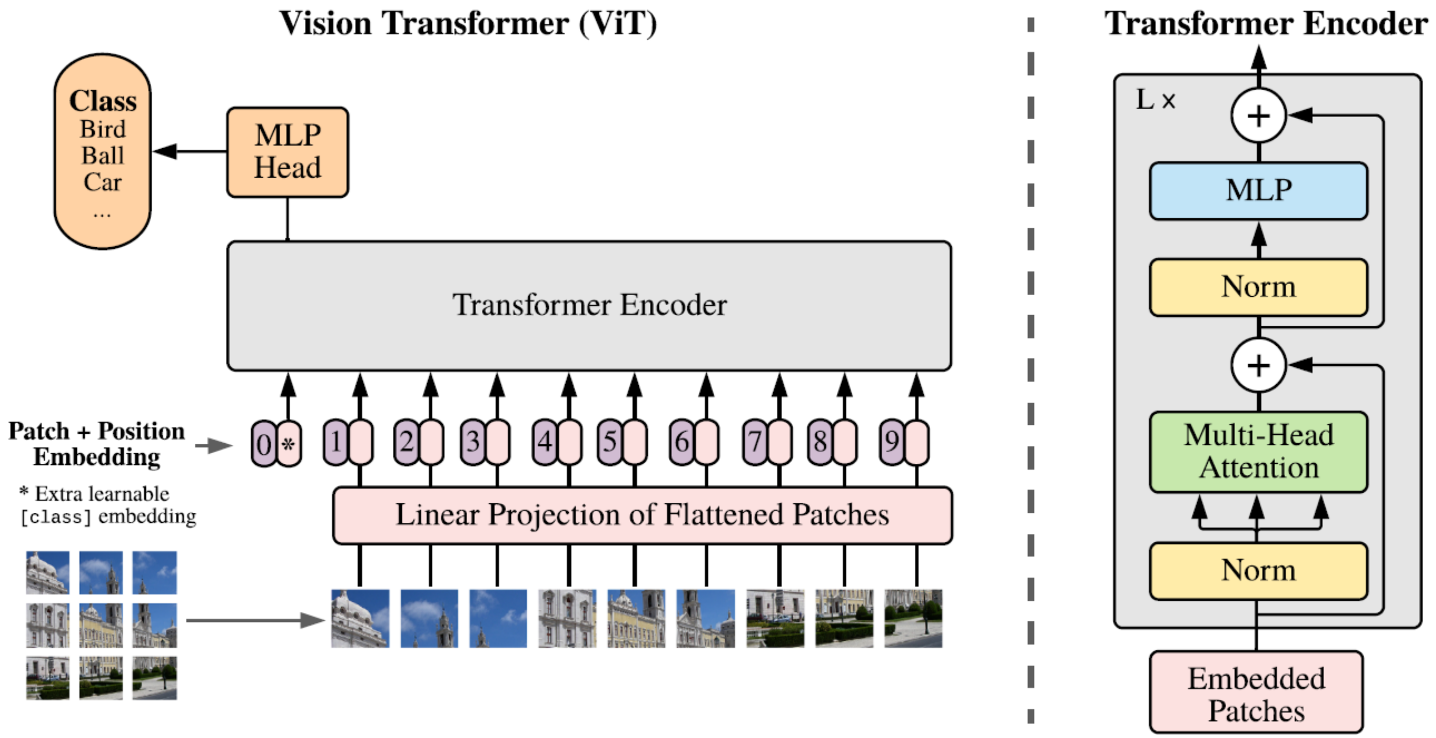
"Residual" of the X to Y mapping

*Deep Residual Learning for Image Recognition (He et al, 2015)

Pic credit: PML-1 (Murphy, 2023)

Transformers have many variants

- BERT (Bidirectional Encoder Representations from Transformers)
- GPT (Generative Pretrained Transformer)
- T5 (Text-to-Text Transfer Transformer)
- Vision Transformers (ViT)


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0 *
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches
Transformer Encoder
L ×
+
MLP
Norm
+
Multi-Head Attention
Norm
Embedded Patches