Indian Institute of Technology Kharagpur
Department of Computer Science & Engineering

CS60075
Natural Language Processing
Autumn 2020

Module 8:

Transformers – Part1

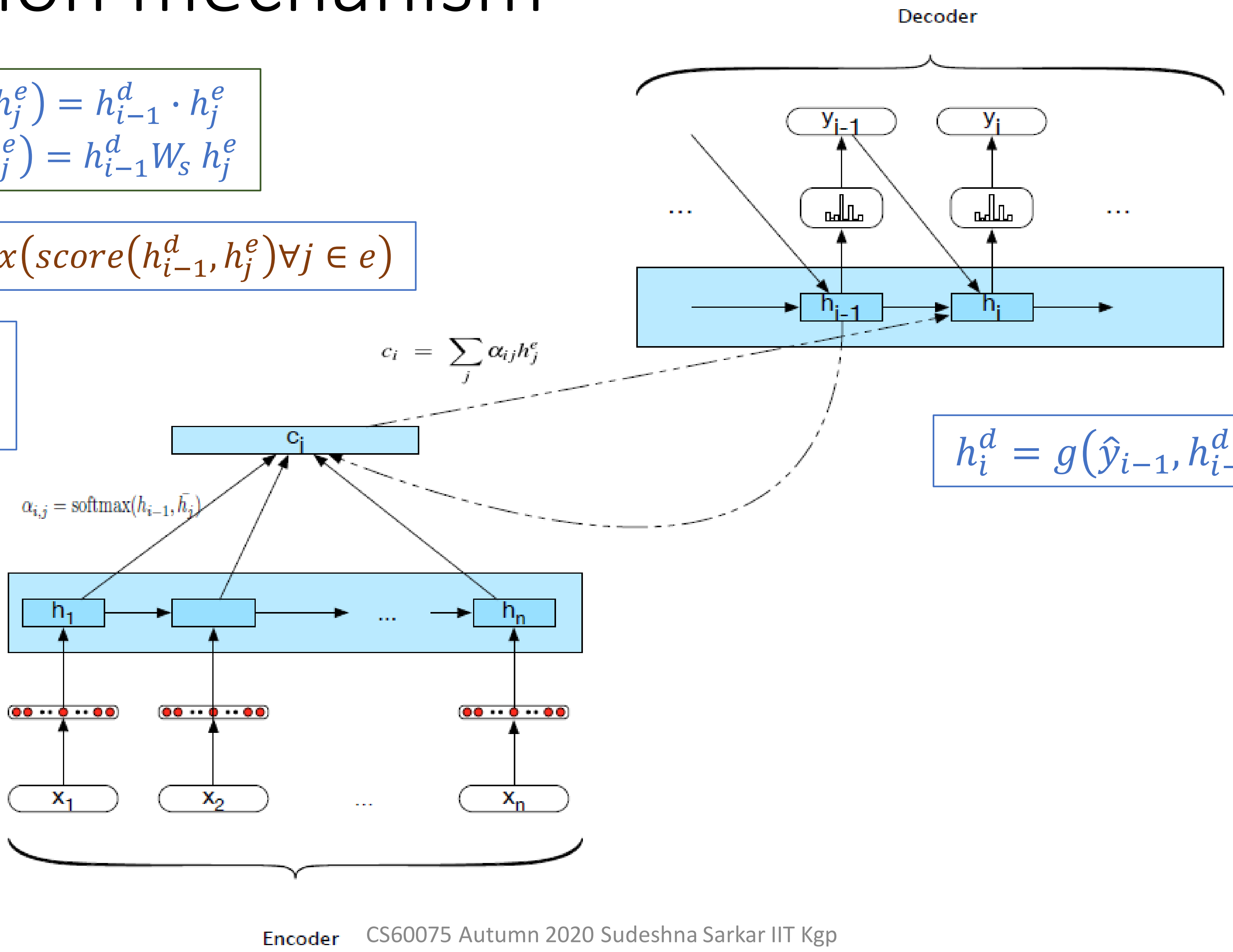
29 October 2020

Attention mechanism

$$score1(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e$$
$$score2(h_{i-1}^d, h_j^e) = h_{i-1}^d W_s\, h_j^e$$

$$\alpha_{i,j} = softmax(score(h_{i-1}^d, h_j^e) \forall j \in e)$$

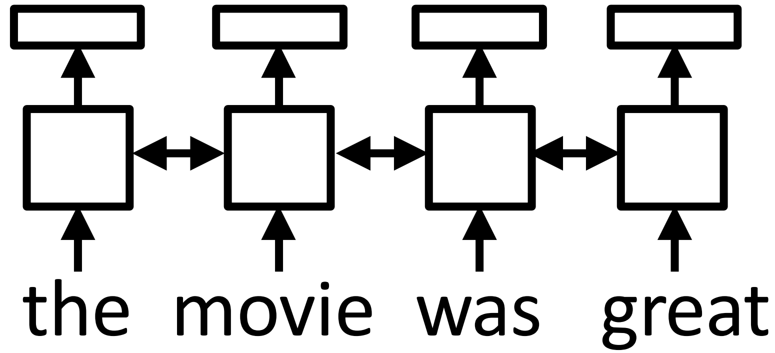
$$c_i = \sum_j \alpha_{i,j} h_j^e$$



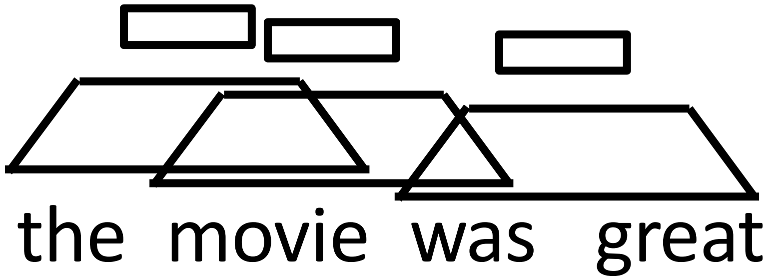
$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$$

Encoders

- RNN: map each token vector a new context-aware token using a autoregressive sequential process



- Attention can be an alternative method to generate context-dependent embeddings



LSTM/CNN Context

- What context do we want token embeddings to take into account?

The ballerina is very excited that she will dance in the show.

- What words need to be used as context here?
  - Pronouns context should be the antecedents (i.e., what they refer to)
  - Ambiguous words should consider local context
  - Words should look at syntactic parents/children
- Problem: RNNs (i.e., LSTMs) and CNNs fail to do this

LSTM/CNN Context

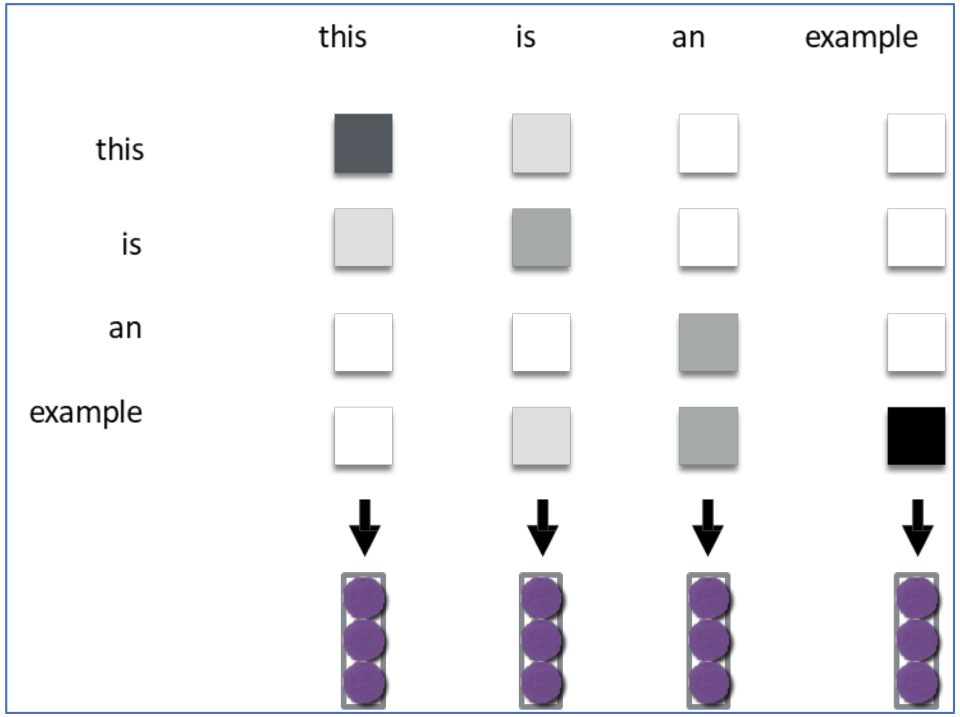
Want:

The ballerina is very excited that she will dance in the show.

- LSTMs/CNNs: tend to be local
- To appropriately contextualize, need to pass information over long distances for each word

Self-attention

- Each word is a query to form attention over all tokens
- This generates a context-dependent representation of each token: a weighted sum of all tokens
- The attention weights dynamically mix how much is taken from each token
- Can run this process iteratively, at each step computing self-attention on the output of the previous level



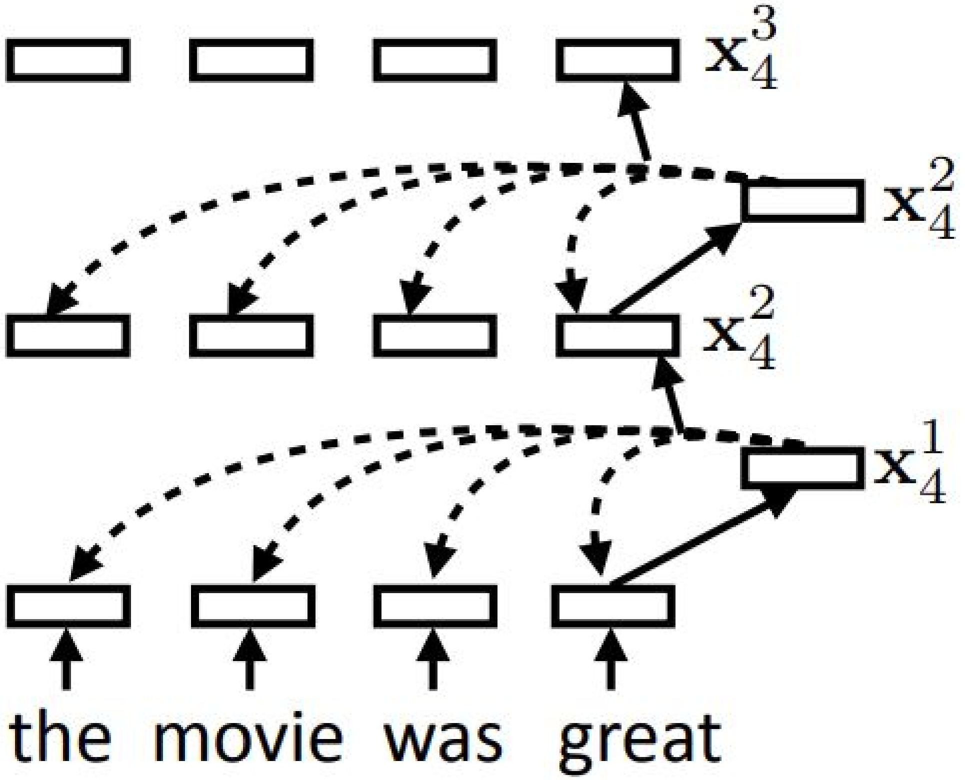
Self-Attention

Information flows from within the same subnetwork (either encoder or decoder).

- Convolution applies fixed transform weights.
- Self-attention applies variable weights.

Self-attention

- Each word is a query to form attention over all tokens
- This generates a context-dependent representation of each token: a weighted sum of all tokens
- The attention weights dynamically mix how much is taken from each token
- Can run this process iteratively, at each step computing self-attention on the output of the previous level



Self-attention

k : level number

X : input vectors

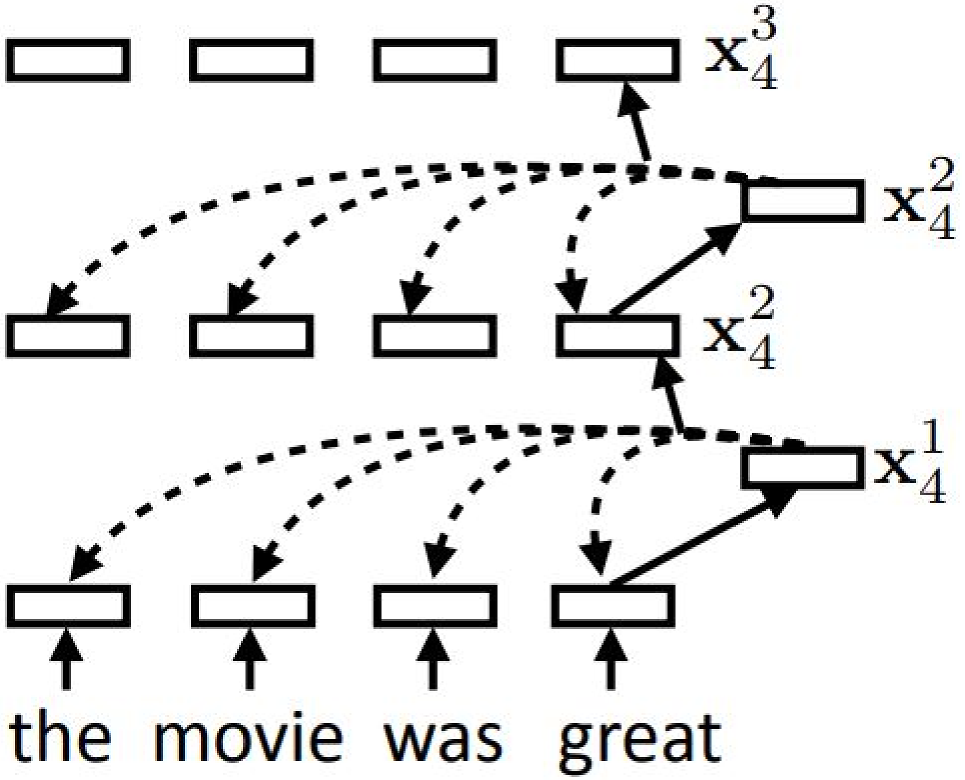
$X = \mathbf{x}_1, \dots, \mathbf{x}_n$

$\mathbf{x}_i^1 = \mathbf{x}_i$

$\bar{\alpha}_{i,j}^k = \mathbf{x}_i^{k-1} \cdot \mathbf{x}_j^{k-1}$

$\alpha_i^k = softmax(\bar{\alpha}_{i,1}^k, \dots, \bar{\alpha}_{i,n}^k)$

$$x_i^k = \sum_{i=1}^{n} \alpha_{i,j}^k \mathbf{x}_j^{k-1}$$



Multiple Attention Heads

- Multiple attention heads can learn to attend in different ways
- Requires additional parameters to compute different attention values and transform vectors

Multiple Attention Heads

k : level number

L: number of heads

$X = \mathbf{x}_1, \dots, \mathbf{x}_n$

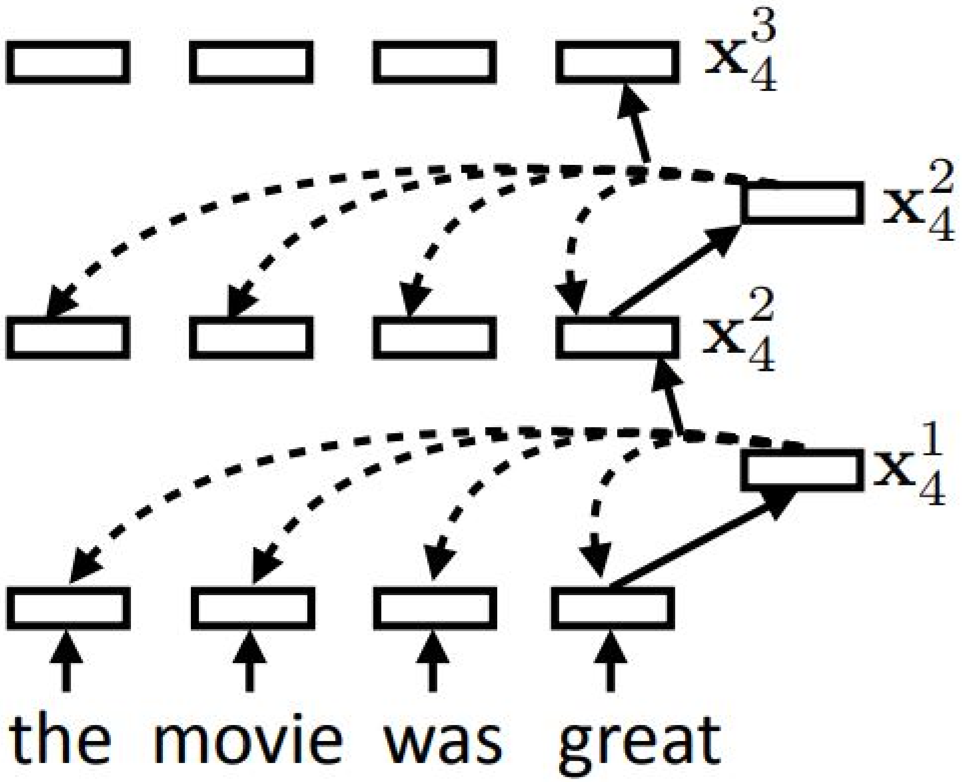
$\mathbf{x}_i^1 = \mathbf{x}_i$

$$\bar{\alpha}_{i,j}^{k,l} = \mathbf{x}_i^{k-1} \mathbf{W}^{k,l} \, \mathbf{x}_j^{k-1}$$

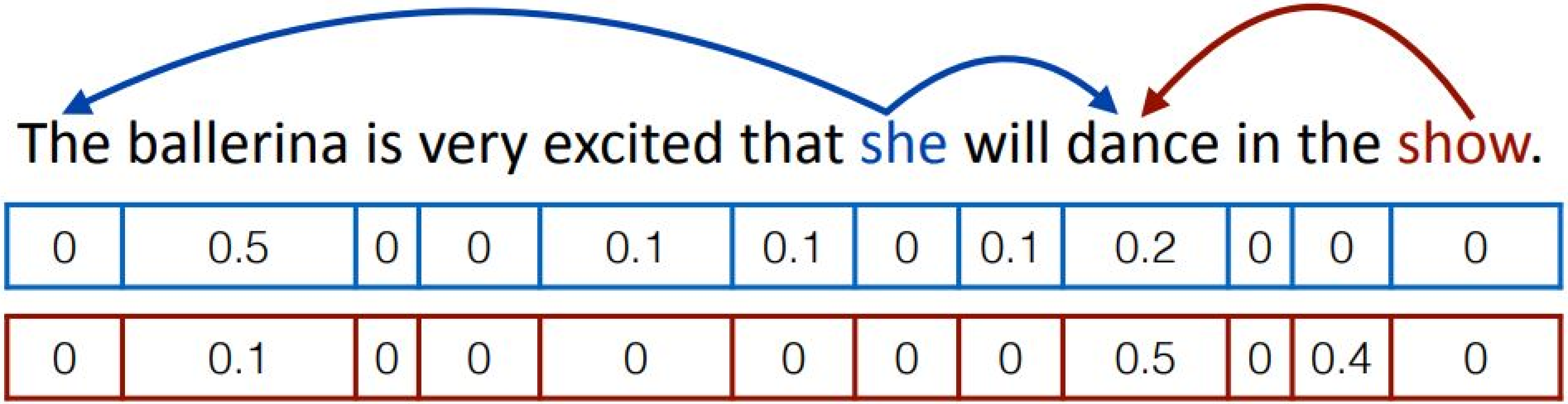
$$\alpha_i^{k,l} = \mathrm{softmax}\left(\bar{\alpha}_{i,1}^{k,l}, \dots, \bar{\alpha}_{i,n}^{k,l}\right)$$

$$x_i^{k,l} = \sum_{i=1}^{n} \alpha_{i,j}^{k,l} \mathbf{x}_j^{k-1}$$

$$\mathbf{x}_i^k = V^k\left[\mathbf{x}_i^{k,1}; \dots; \mathbf{x}_i^{k,L}\right]$$



What Can Self-attention do?



- Attend to nearby related terms
- But just the same to far semantically related terms

Details

- This is the basic building block of an architecture called Transformers
- There are many details to get it to work, see Vaswani et al. 2017, later work, and available implementations
- Significant improvements for many tasks, including machine translation (Vaswani et al. 2017) and context-dependent pre-trained embeddings (BERT; Devlin et al. 2018)

Links

- Annotated Transformer, Illustrated Transformer

- Contextualized Repr

- BERT, **The Illustrated BERT, ELMo, and co.**, Chen2019

Attention-only Translation Models

Problems with recurrent networks:

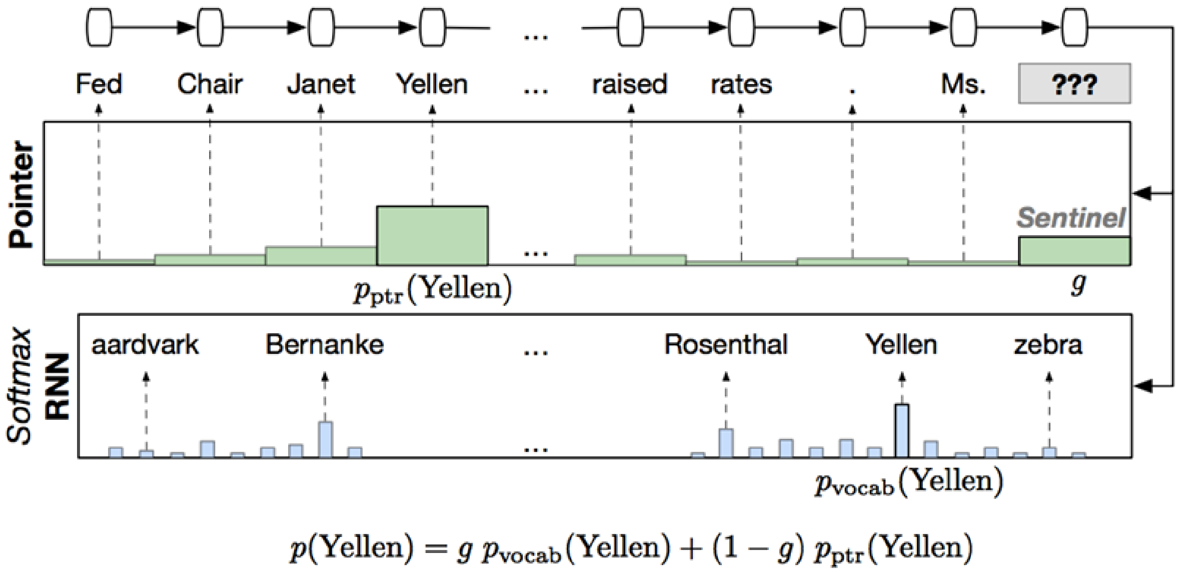
- Sequential training and inference: time grows in proportion to sentence length. Hard to parallelize.
- Long-range dependencies have to be remembered across many single time steps.
- Tricky to learn hierarchical structures ("car", "blue car", "into the blue car"...)

Alternative:

- Convolution – but has other limitations.

Attending to Previously Generated Things

- In language modeling, attend to the previous words (Merity et al. 2016)



- In translation, attend to either input or previous output (Vaswani et al. 2017)

Attention in Transformer Networks

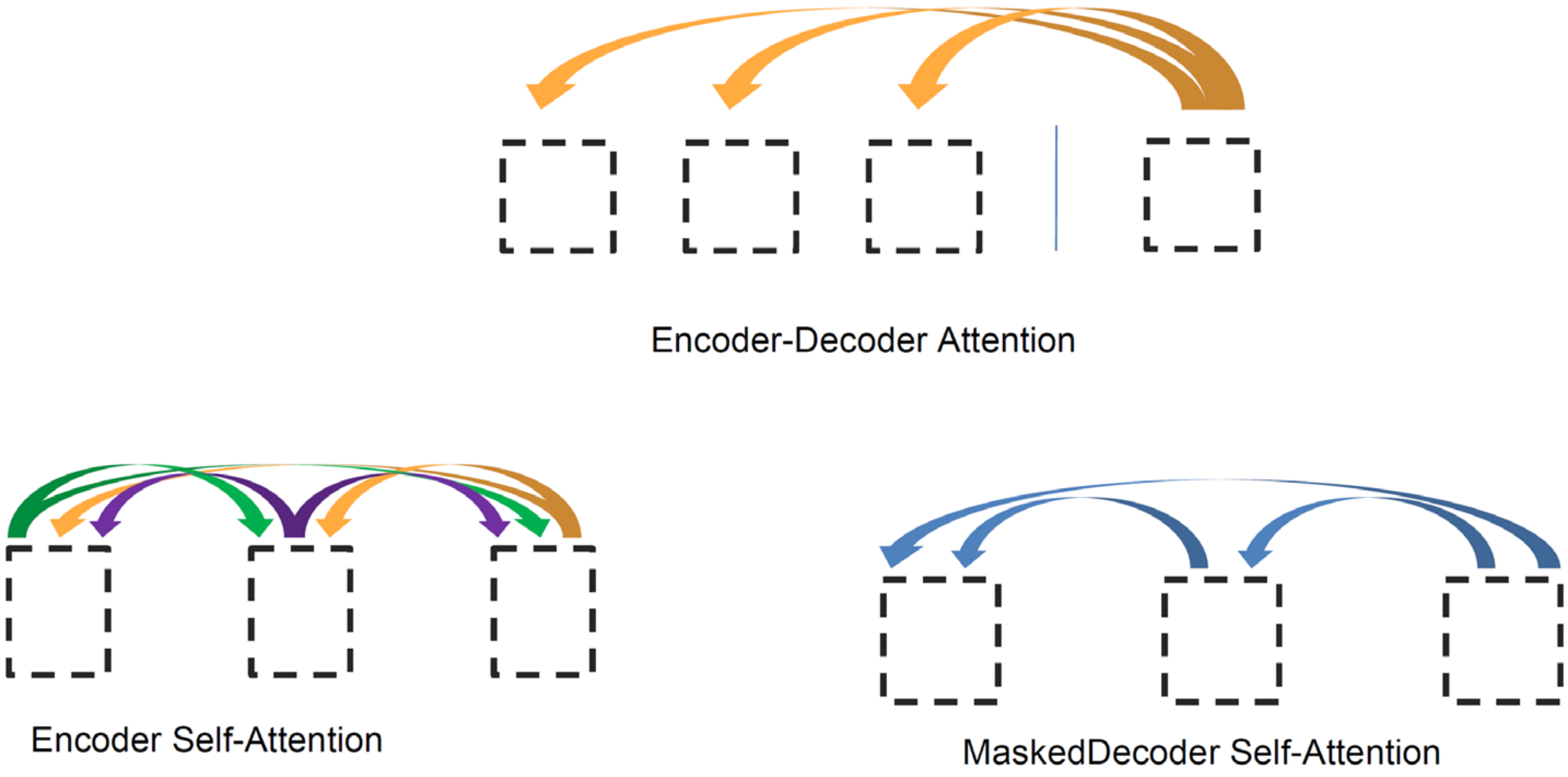
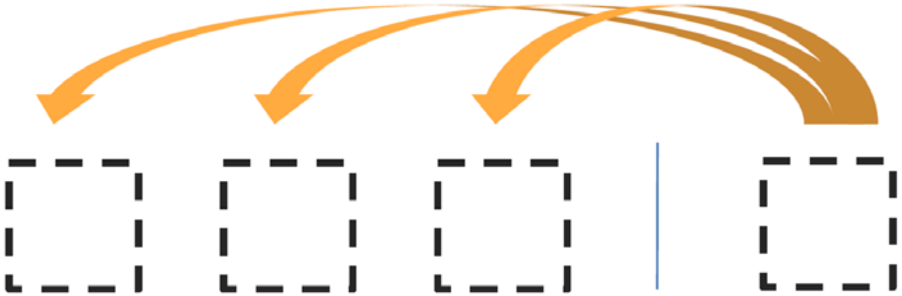


image from Lukas Kaiser, Stanford NLP seminar

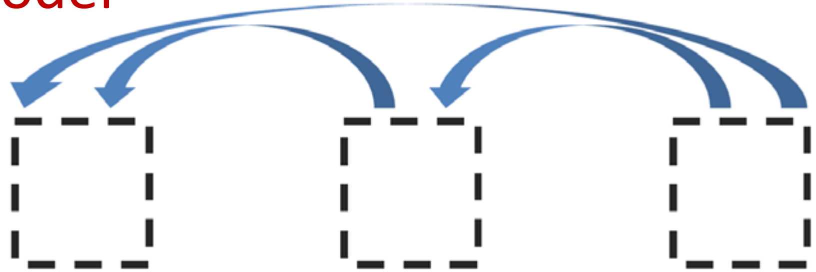
Attention in Transformer Networks



Encoder-Decoder Attention

Replaces word recurrence in encoder and decoder

Encoder Self-Attention



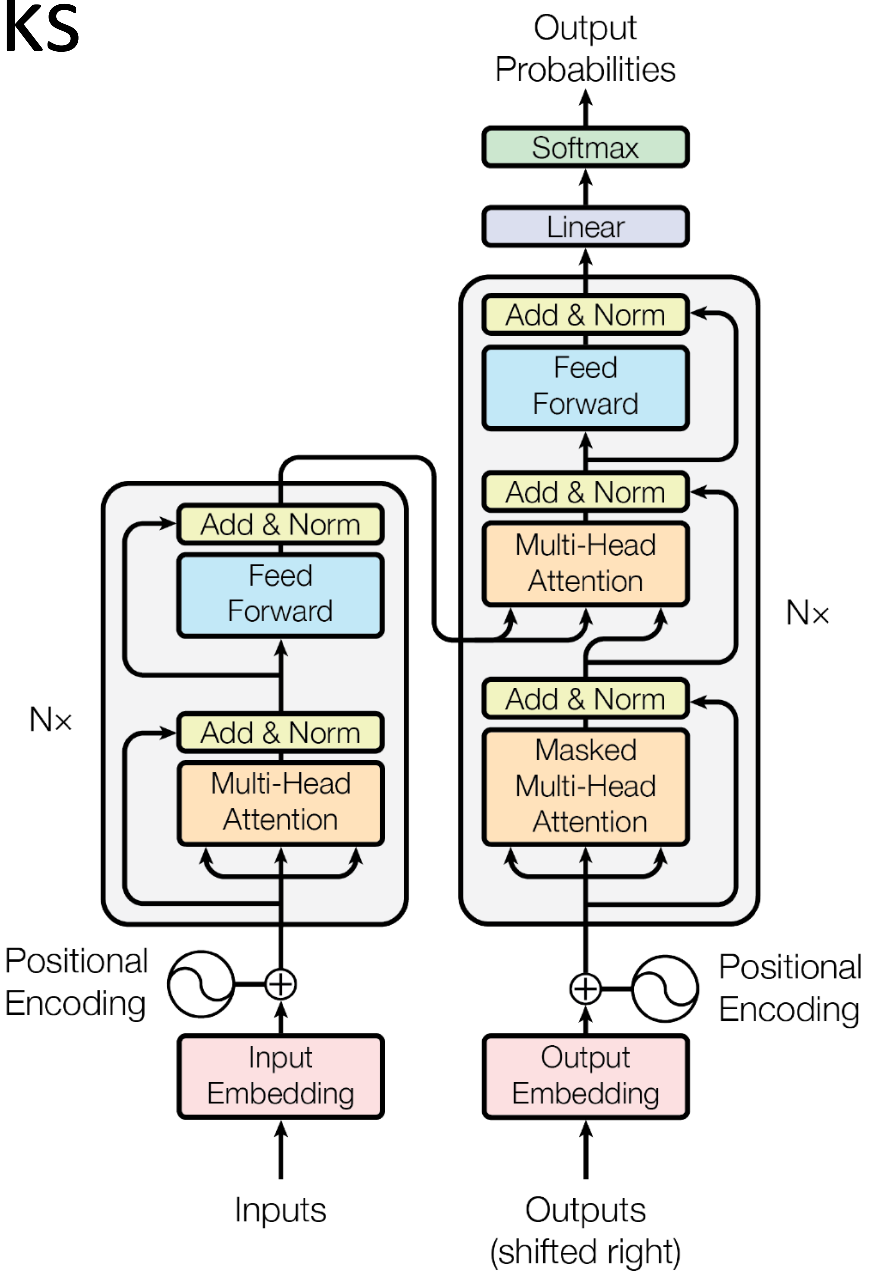
MaskedDecoder Self-Attention

Masking limits attention to earlier units: y_i depends only on y_j for $j < i$.

image from Lukas Kaiser, Stanford NLP seminar

The Transformer Attention Tricks

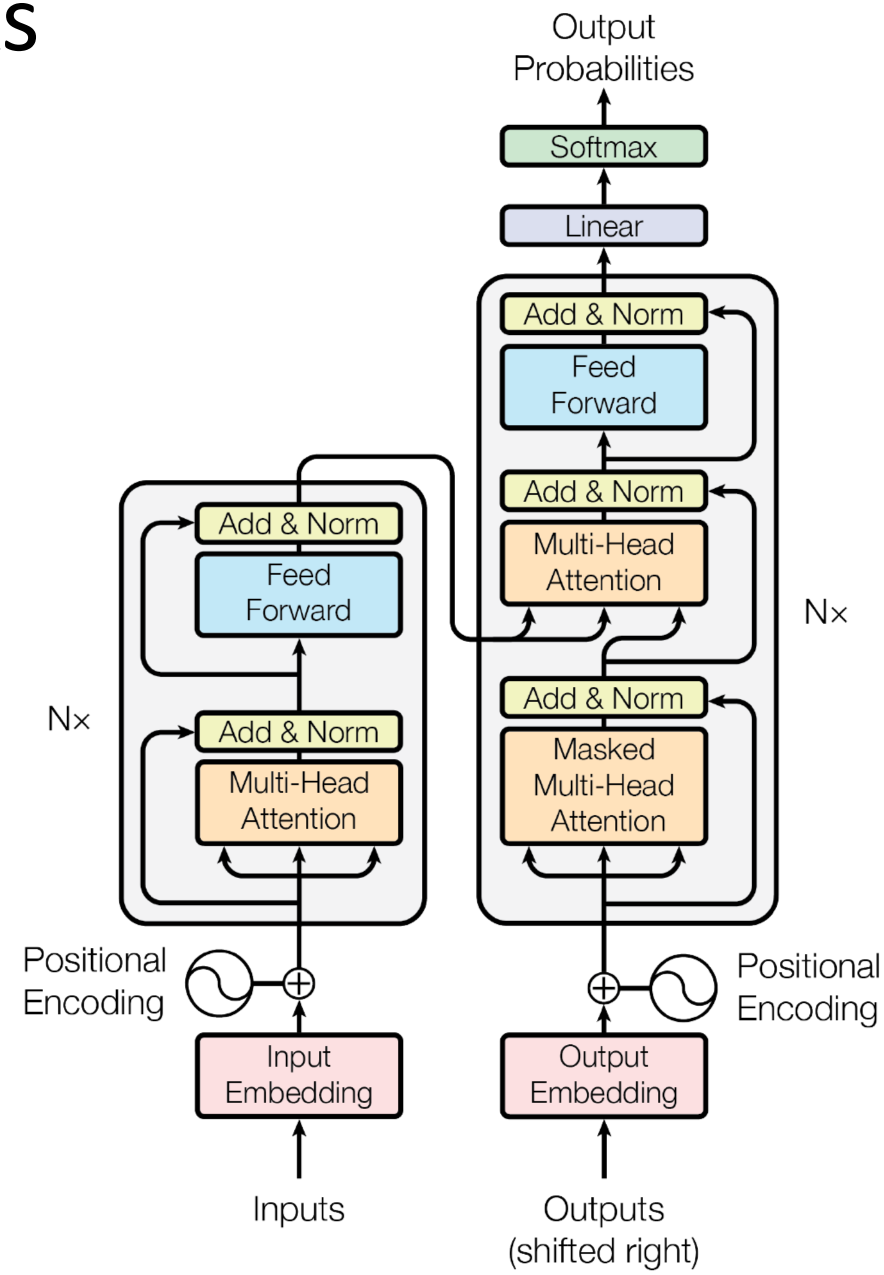
- **Self Attention:** Each layer combines words with others
- **Multi-headed Attention:** 8 attention heads function independently
- **Normalized Dot-product Attention:** Remove bias in dot product when using large networks
- **Positional Encodings:** Make sure that even if we don’t have RNN, can still distinguish positions



The Transformer Training Tricks

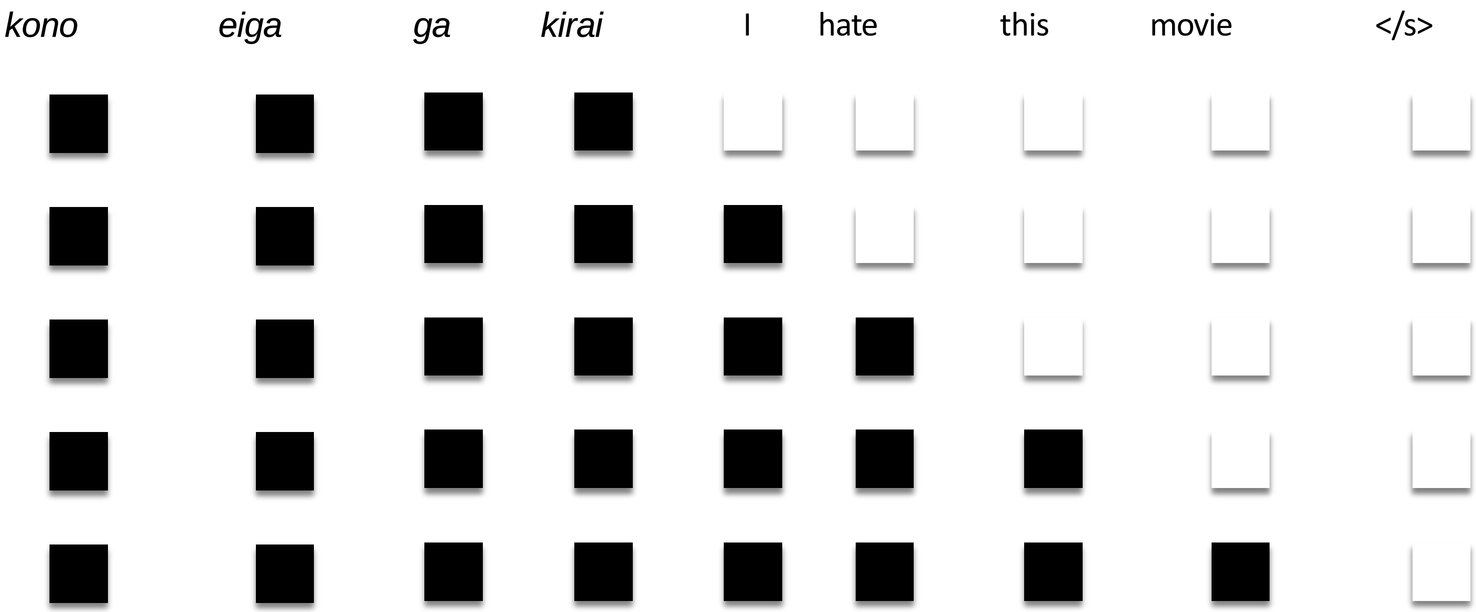
- **Layer Normalization:** Help ensure that layers remain in reasonable range
- **Specialized Training Schedule:** Adjust default learning rate of the Adam optimizer
- **Label Smoothing:** Insert some uncertainty in the training process

  Masking for Efficient Training



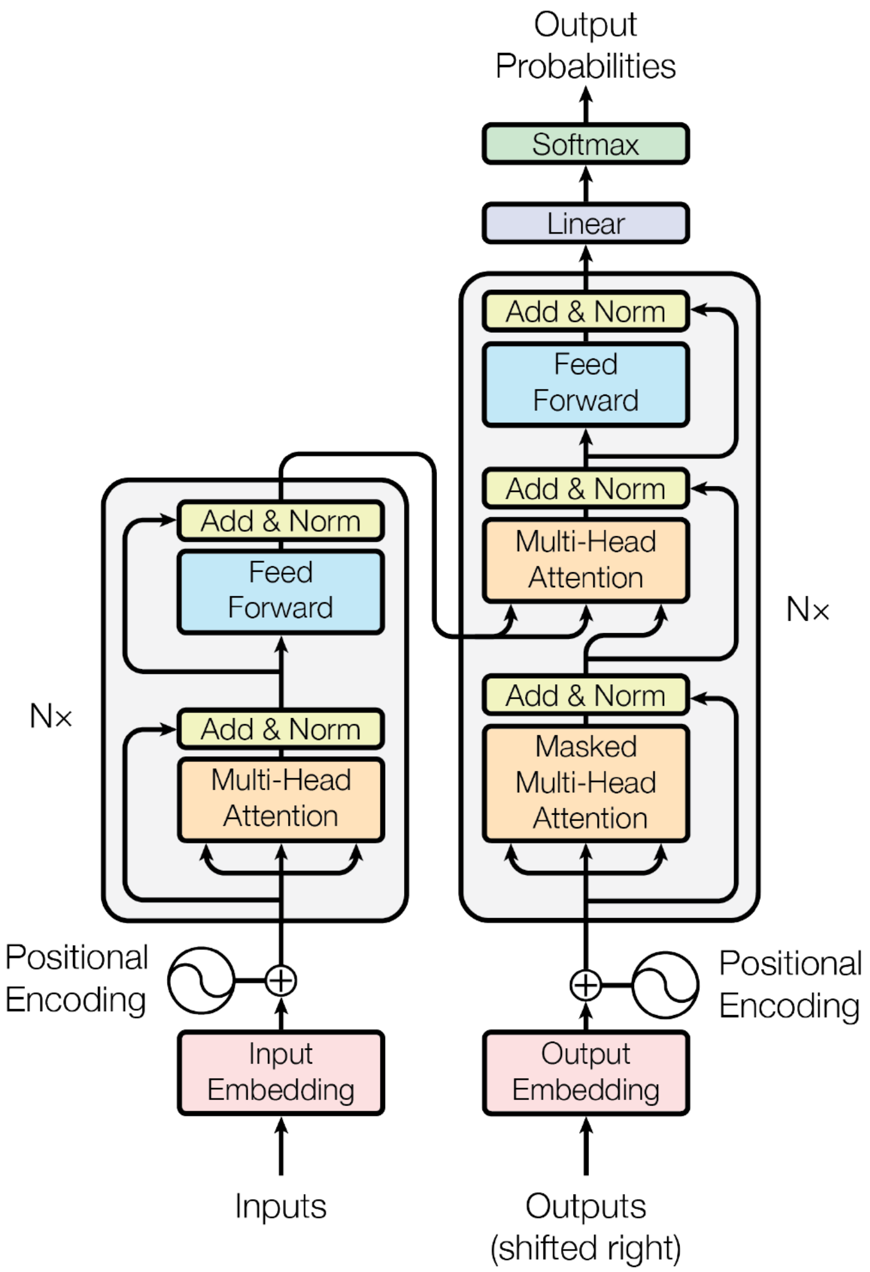
Masking for Training

- We want to perform training in as few operations as possible using big matrix multiplies
- We can do so by “masking” the results for the output



The Transformer

- In experiments, stacked with N=6.
- Output words fed back as input, shifted right. Can use beam search as before.
- Inputs and outputs are embedded in vector spaces of fixed dimension.
- Positional encoding: when words are combined through attention, their location is lost. Positional encoding adds it back.



Attention Implementation

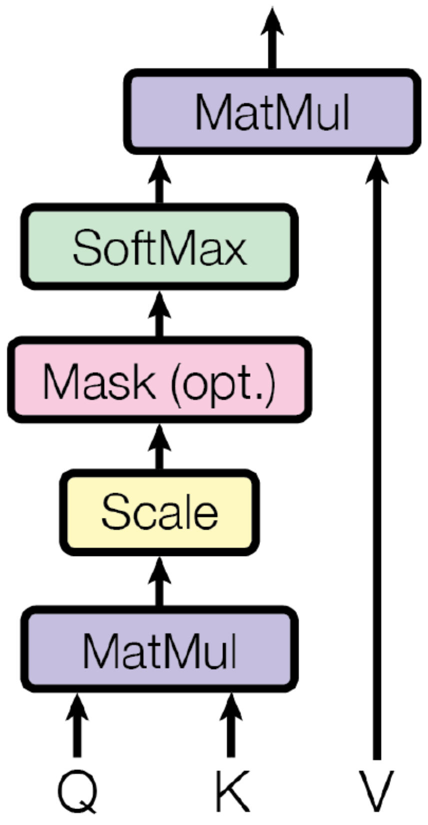
- Attention is modeled as a key-value store:
  Q = query vector
  K = key
  V = value

Encoder-decoder layer: the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. (Similar to Bahdanau).

Self-attention layer: all of the keys, values and queries come from the output of the previous layer in the encoder.

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

Scaled Dot-Product Attention



Multi-Headed Attention