Transformers

Jim Glass / MIT 6.806-6.864 / Spring 2021

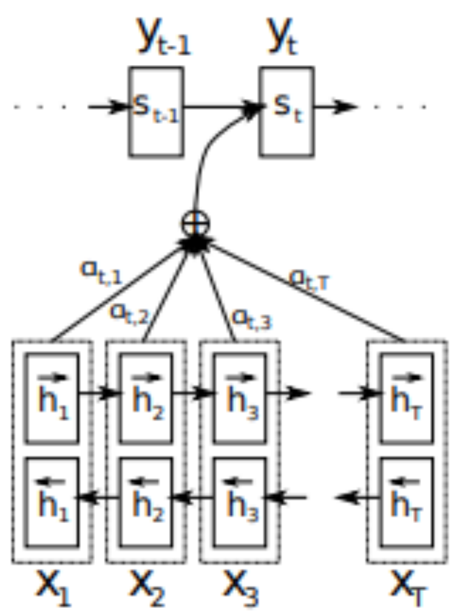
1

Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

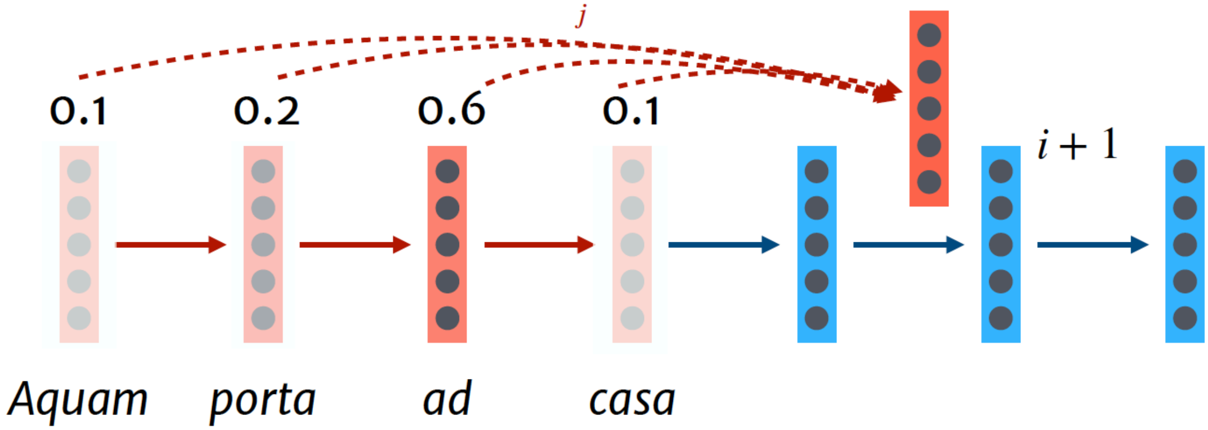
KyungHyun Cho **Yoshua Bengio***
Université de Montréal



2

Attention mechanisms

1. When predicting output i, assign a weight α_{ij} to each encoder state h_j
2. Compute a pooled input $c_i = \sum_j \alpha_{ij} h_j$

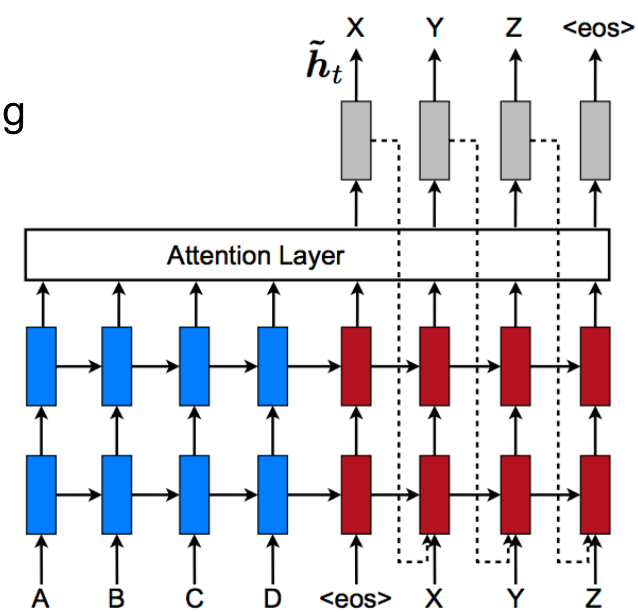


[Jacob's Attention lecture on Tuesday]

3

Challenges with RNNs

- The sequential nature of RNN models makes training challenging
  - Precludes parallelization
  - Unwieldy for long sequences
  - Limits batch sizes
- The best RNNs use attention to handle distant dependencies

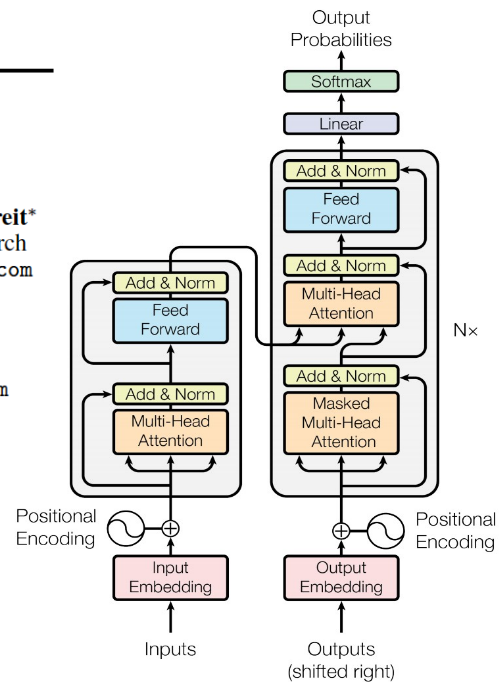


[Luong et al., "Attention-based NMT" arXiv:1508.04025, 2015]

4

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com



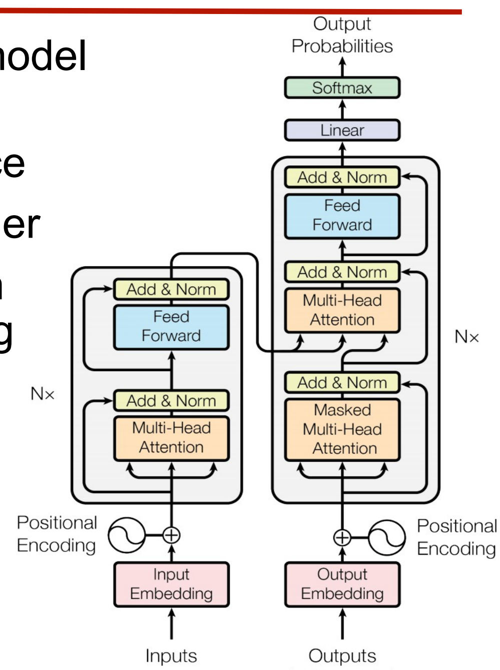
If attention gives access to all words, do we need recurrence?

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

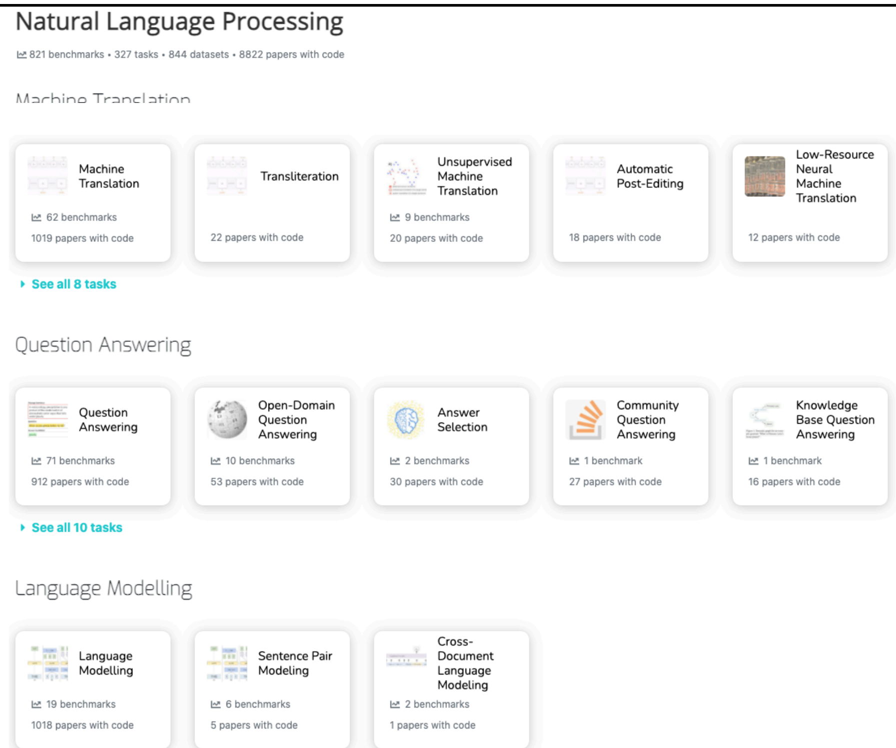
5

Transformers: Attention Is All You Need

- Non-recurrent seq2seq (encoder-decoder) model
- Multi-layered attention model enables lateral information transfer across an input sequence
- Loss function is cross-entropy error of decoder
- Original paper demonstrated good results on machine translation and constituency parsing
- Transformers are the basis for BERT etc. (which we will see next week)



6

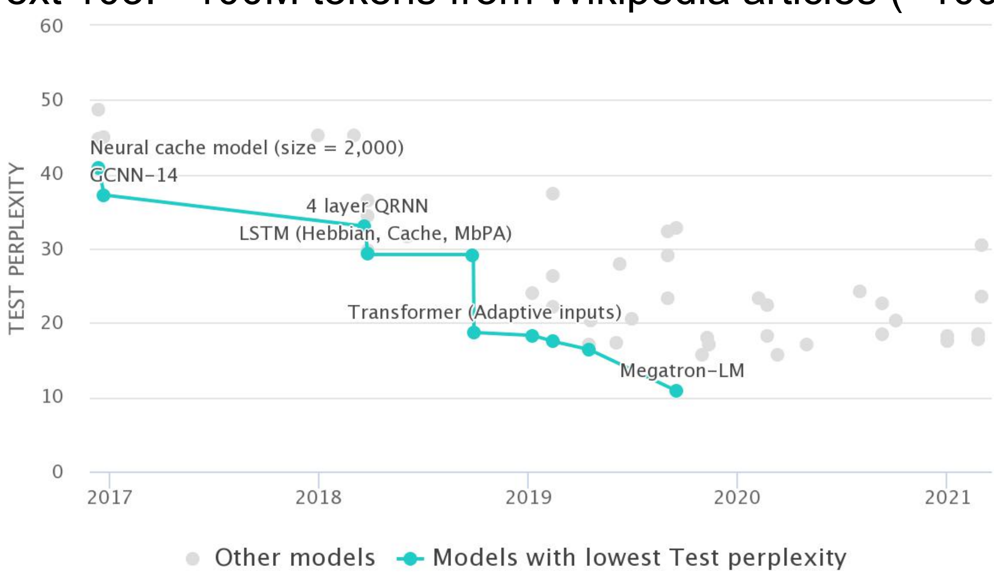


https://paperswithcode.com/area/natural-language-processing

7

Language Modeling

- WikiText 103: >100M tokens from Wikipedia articles (>100x PTB)

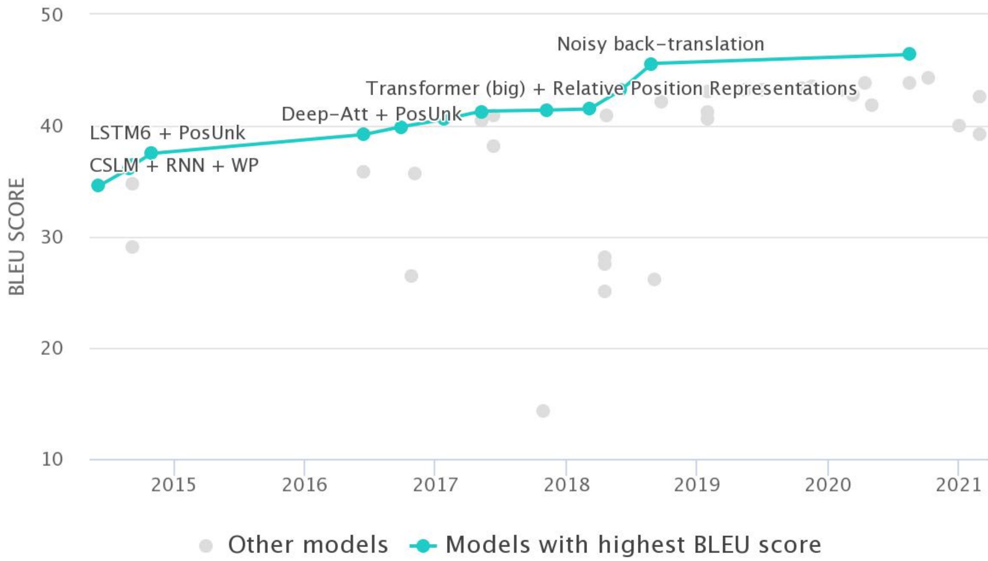


https://paperswithcode.com/sota/language-modelling-on-wikitext-103

8

Machine Translation

- WMT2014 English-French (~36M sentence pairs)



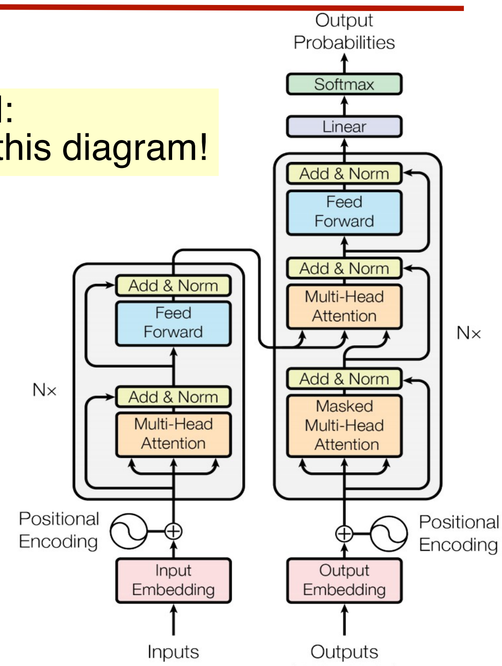
https://paperswithcode.com/sota/machine-translation-on-wmt2014-english-french

9

Today's Transformer Story

- Encoder-decoder model
- Attention
  - Self-attention
  - Multi-head attention
- Other transformer topics
  - Positional encoding
  - Residual connections and normalization
  - Decoder masking
- Training issues
  - Byte-pair encoding (BPE)

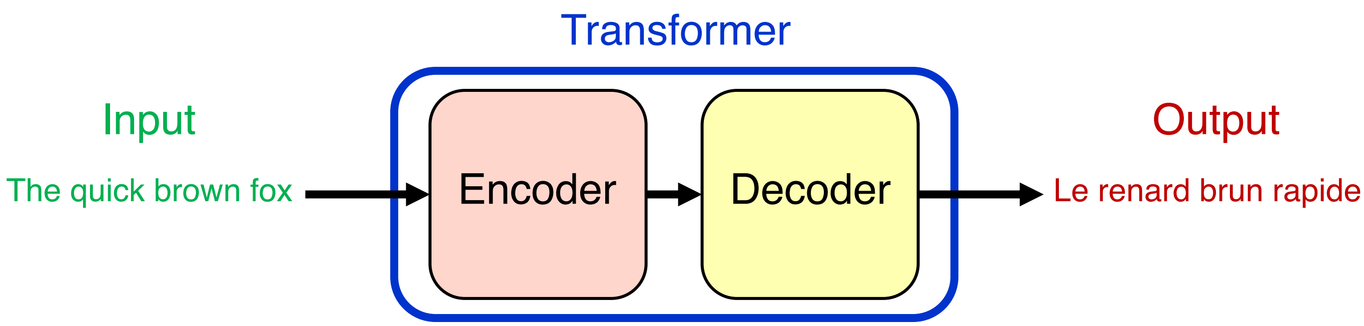
Today's goal:
understand this diagram!



10

Transformer as Encoder-Decoder

- The transformer is a non-recurrent sequence-to-sequence model
  - Uses an encoder to learn a latent representation of the input
  - Uses a decoder to generate an output from the latent representation

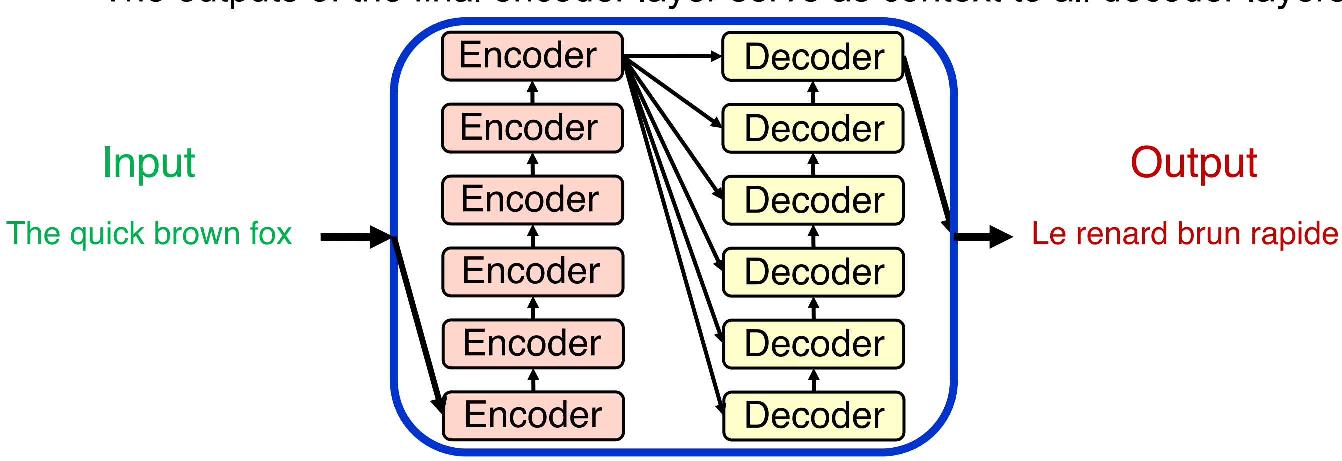


  - *Self-attention* is used to enable the model to observe entire sequences

11

Transformer Encoders and Decoders

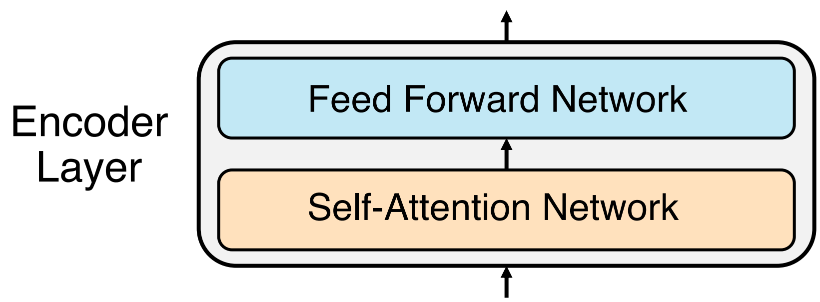
- The transformer consists of multiple (6) encoder and decoder layers
  - All encoder layers have identical structure but learn separate parameters
  - All decoder layers have identical structure but learn separate parameters
  - The outputs of the final encoder layer serve as context to all decoder layers



12

Basic Encoder Structure

- An encoder layer has a self-attention and a feed-forward sub-layer
  - The number of outputs is the same as the number of inputs (e.g., words)
  - Self-attention lets encoder look at entire input to encode a specific word
  - The same feed-forward network is applied to each word in turn

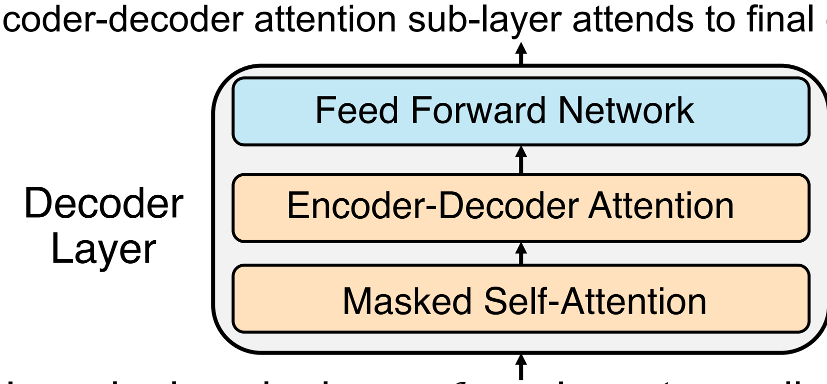


- Each encoder layer produces a latent representation for each word that is influenced by the surrounding context of the input sequence

13

Basic Decoder Structure

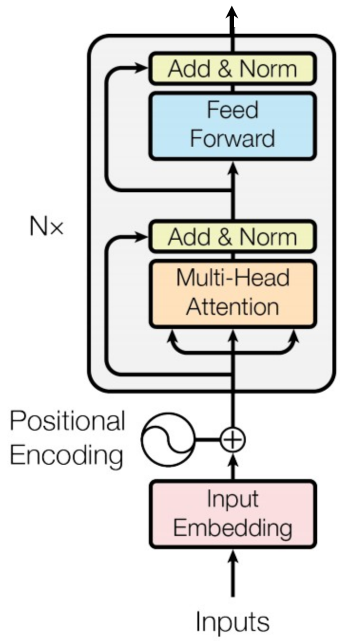
- The role of the decoder is to generate an output one word at a time
  - The output is generated incrementally (as in seq2seq language generation)
- A decoder layer has a self-attention, attention, and a FF sub-layer
  - Self-attention sub-layer attends to previously generated decoder output
  - Encoder-decoder attention sub-layer attends to final output of encoder



- Final layer in decoder is a *softmax* layer to predict next word

14

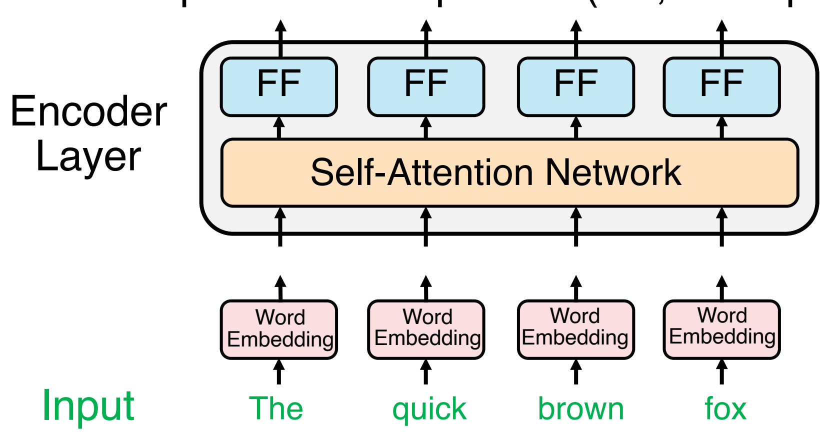
Transformer Encoder



15

Encoder Details

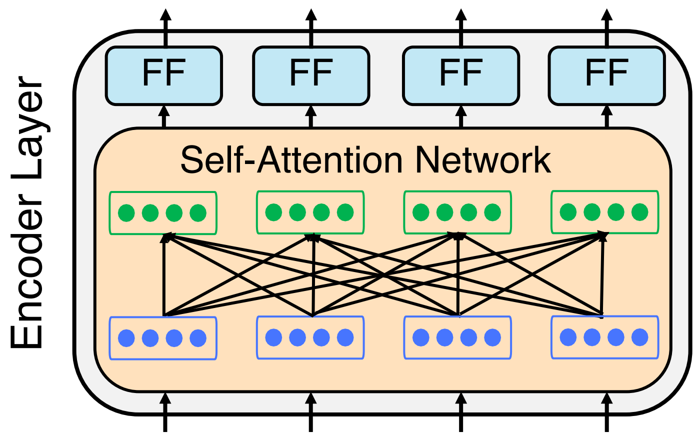
- Input words are represented by a learned embedding vector
- Each word in each position flows through its own path in encoder
- Dependencies between paths are captured by self-attention sub-layer
- FF layer can be implemented in parallel (i.e., no dependencies)

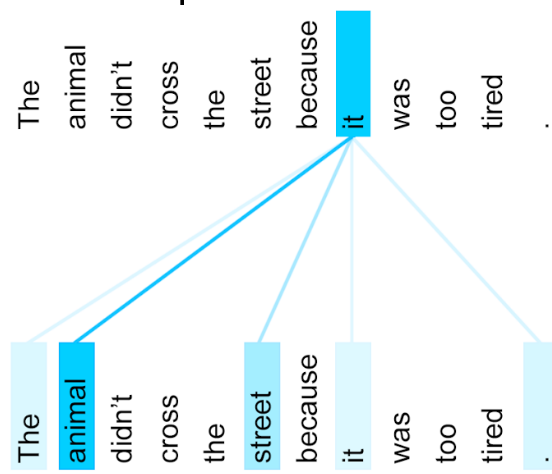


16

Self-Attention

- Self-attention is the concept of incorporating information from other words in the input sequence to encode a specific word
- Self-attention enables learning contextualized representations

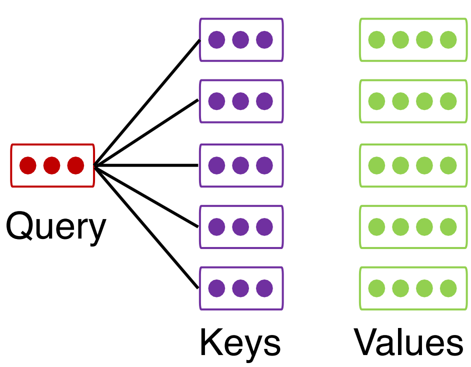




https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Attention Abstraction

- *Query*, *Key*, *Value* vectors are useful abstractions for attention
  - A set of candidate retrieval items represented in terms of keys and values
  - An incoming query is matched against all keys (via dot-product or MLP)
  - Individual query-key distances are normalized (e.g., via *softmax*)
  - Result is attention weighted sum of value vectors (i.e., a linear combination)

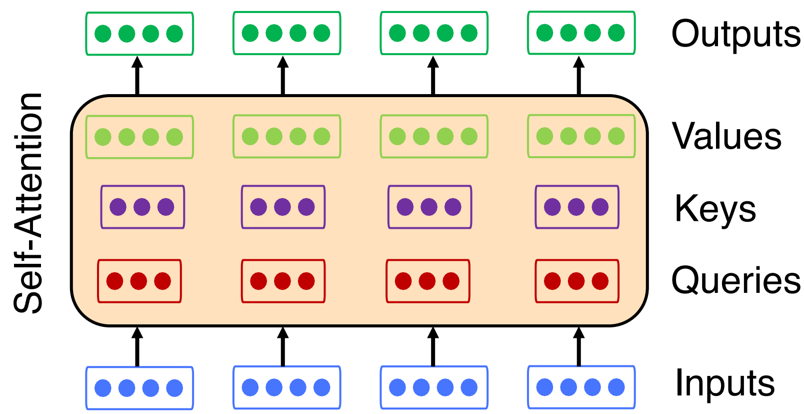


$$A(q, \boldsymbol{K}, \boldsymbol{V}) = \sum_i \frac{e^{\boldsymbol{q}\cdot\boldsymbol{k}_i}}{\sum_j e^{\boldsymbol{q}\cdot\boldsymbol{k}_j}} \boldsymbol{v}_i$$

Softmax values are attention weights! (i.e., α's)

Self-Attention

- For self-attention, the retrieval items are all terms in the sequence
  - Create query, key, value vectors for each input representation
- Vectors are created from input via three learned linear transforms
  - The three transformations provide a means to focus on different subspaces



$\boldsymbol{v}_i = \boldsymbol{W}_v \, \boldsymbol{x}_i$

$\boldsymbol{k}_i = \boldsymbol{W}_k \, \boldsymbol{x}_i$

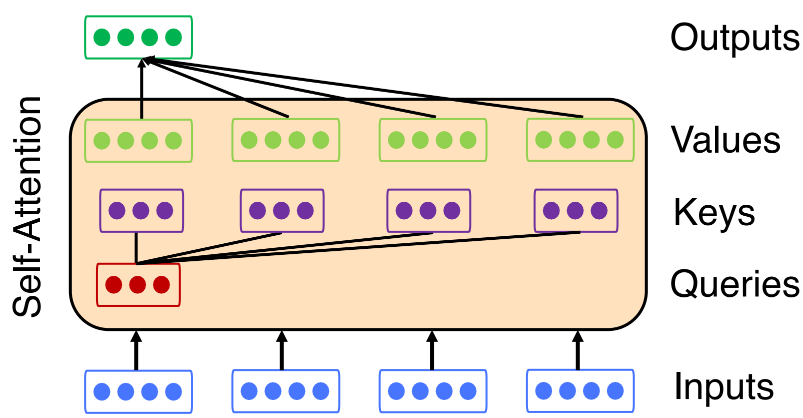
$\boldsymbol{q}_i = \boldsymbol{W}_q \, \boldsymbol{x}_i$

$\boldsymbol{x}_i$

19

Transformer Self-Attention Details

- Scaled dot-product works better for large dimensionality
  - i.e., dot product of two univariate variables has variance of dimension d
- Attention vectors are smaller than embedding size (64 vs 512)
  - When combined with multi-headed attention (8), dimensionality is similar



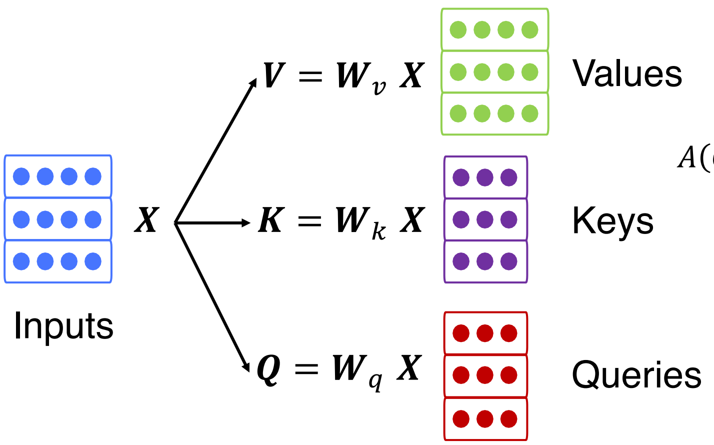
$$A(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \sum_i \frac{e^{\boldsymbol{q}\cdot\boldsymbol{k}_i}}{\sum_j e^{\boldsymbol{q}\cdot\boldsymbol{k}_j}} \boldsymbol{v}_i$$

Softmax values are attention weights! (i.e., α's)

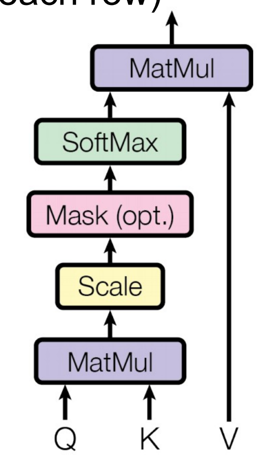
20

Matrix Formulation of Self-Attention

- Matrix formulations of attention can be implemented efficiently
  - Input matrix represents entire sequence of input embeddings
  - Matrix multiplications compute all queries, keys, and values for sequence
  - Sequence outputs are weighted sum of vectors (*softmax* on each row)



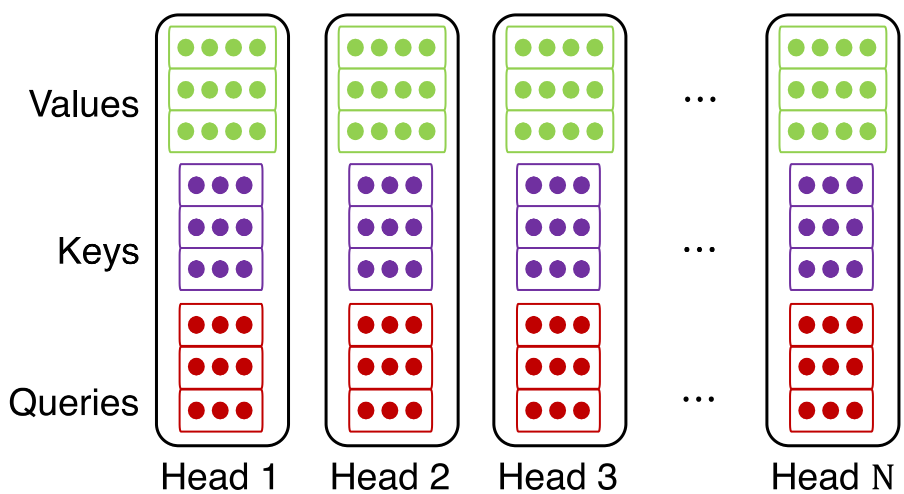
$$A(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



21

Multi-Headed Attention

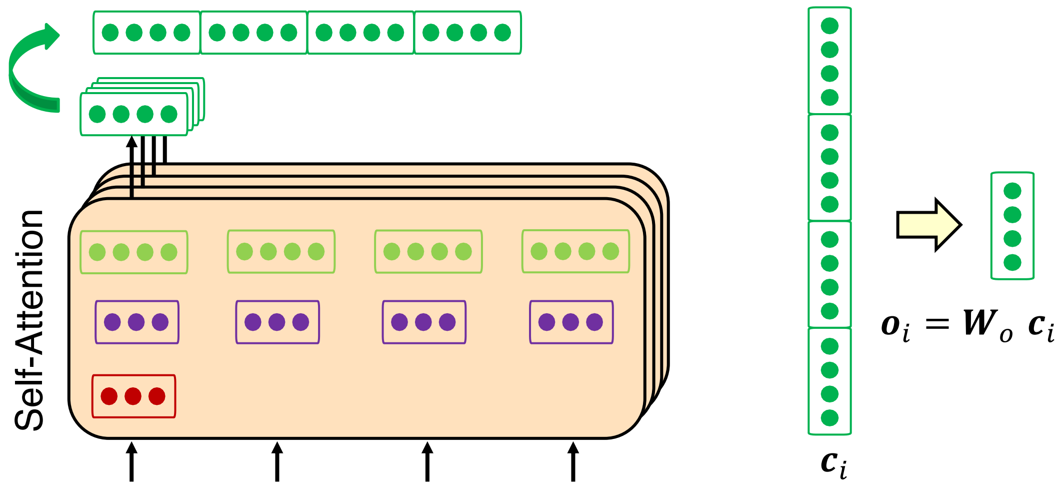
- Multi-headed attention enables multiple representation subspaces
  - Each representation space can attend to different concepts, positions etc.
  - Requires learning query, key, value transforms for each space (e.g., 8)



22

Transformer Multi-Headed Attention

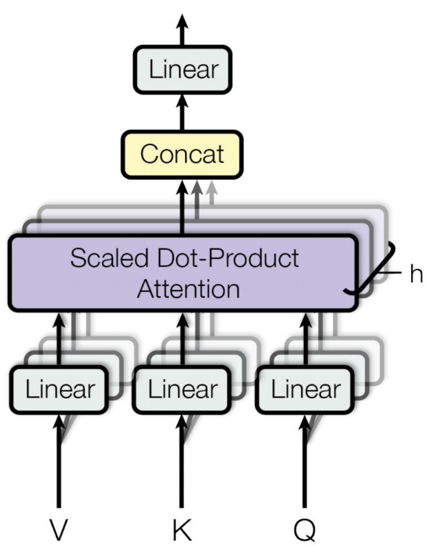
- Multi-headed attention produces new embeddings for each head
- Output vectors are concatenated to create super embedding vector
- A final transformation is learned to map super vector to final vector



23

Transformer Multi-Head Attention

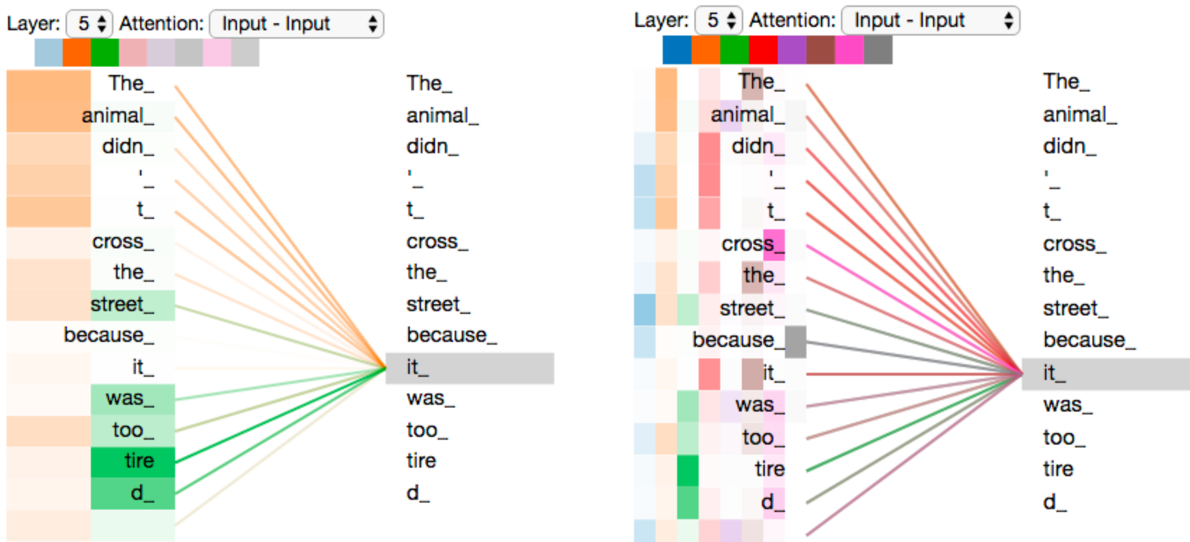
- Attention module is used in every encoder and decoder layer
  - Decoder uses attention on all inputs and all previously generated outputs



24

Attention Visualization

- Visualizing attention weights can yield insight to learned behavior

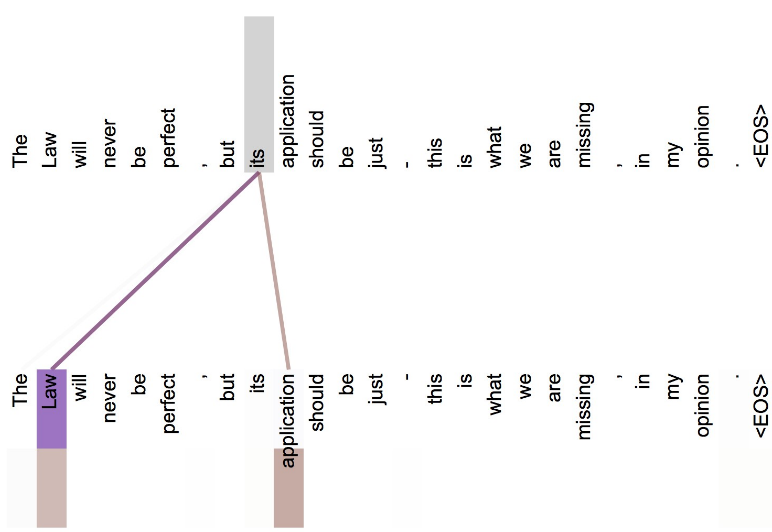


https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb

25

Attention Visualization

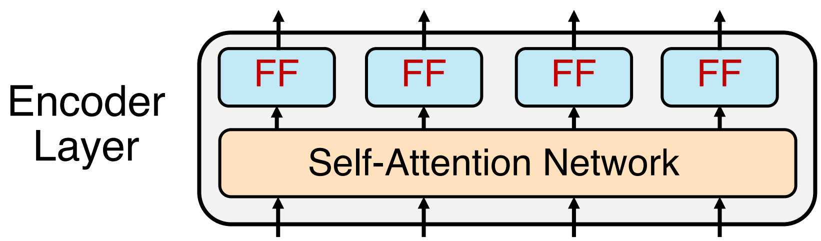
- Attention weights can learn syntactic and semantic relationships
  - e.g., implicit anaphora resolution for attentions for ‘its’ for heads 5 & 6



26

Feed Forward Network

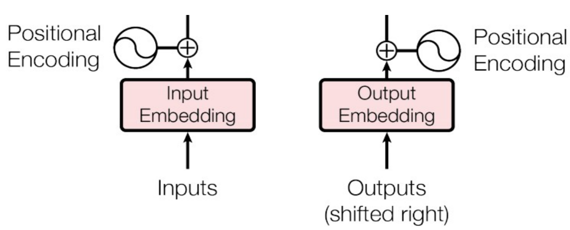
- Each encoder (and decoder) layer has a feed-forward network
  - Two linear transformations with *ReLU* activation in hidden layer
  - The FF network is applied in parallel to each position in sequence
  - Different FF network parameters are learned for each layer



27

Positional Encoding of Word Embeddings

- Attention does not account for word order like recurrent networks
- Positional encoding adds a "position" vector to each embedding
  - The same word in different locations will have different encodings



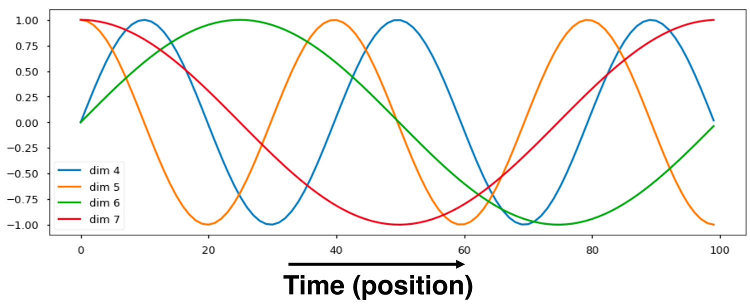
- Position encodings can be a pre-defined function, or be learned
  - The transformer model uses a *sinusoidal* positional encoding
- Position vectors enable the model to learn the position of a word in a sequence, or the relative distances between words
  - There is a linear relationship between position vectors

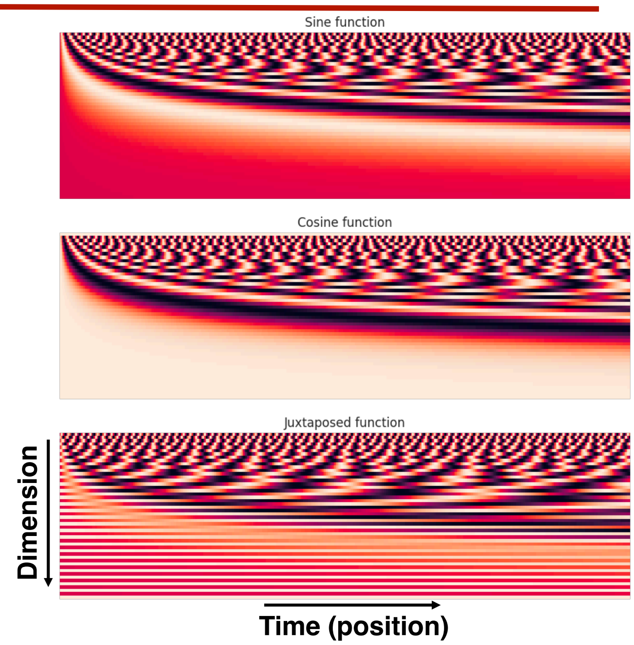
28

Sinusoidal Positional Encoding

- Sinusoidal positional encoding
  - Alternates between sine and cosine
  - Wavelength increases with dimension

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$





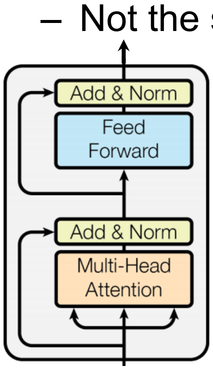
[Rush, Annotated Transformer]

http://vandergoten.ai/2018-09-18-attention-is-all-you-need/

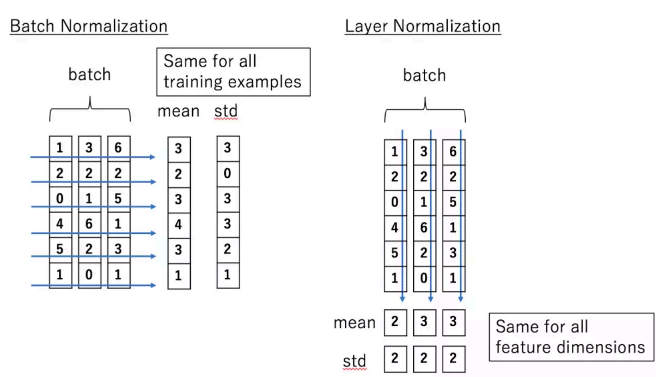
29

Residual Connections and Layer Normalization

- Transformer stages use residual connections and layer normalization
  - Performed after all stages in the encoder and decoder (i.e., 2 and 3 times)
- Residual connections directly *add* stage input and output vectors
  - Residual connections help vanishing gradient issue in deep networks
- Layer normalization scales outputs to be zero mean, unity variance
  - Not the same as batch normalization



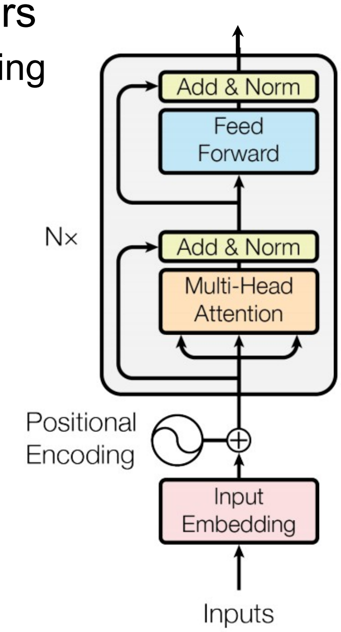
$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

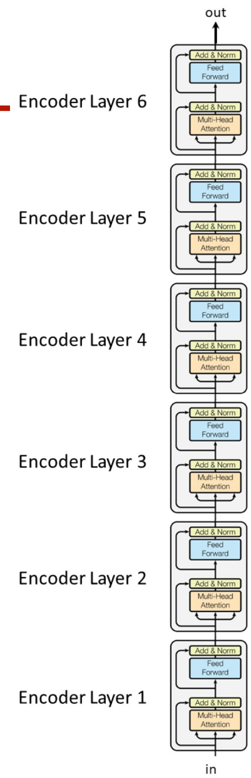


30

Transformer Encoder Summary

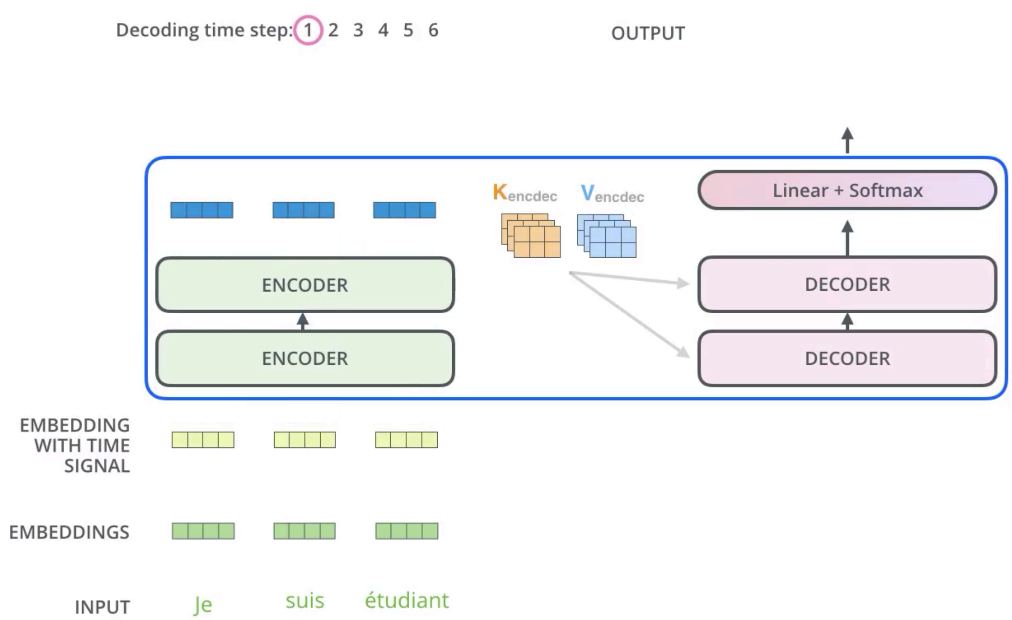
- Input represented as embedding vectors
  - Positional vector added to each embedding
- Encoder layers repeated 6 times
  - Entire sequence pass through in parallel
- Each encoder layer contains:
  - Multi-head attention (e.g., 8 heads)
  - Position-wise feed forward network
  - Residual connections at each sub-layer
  - Layer normalization after each sub-layer





31

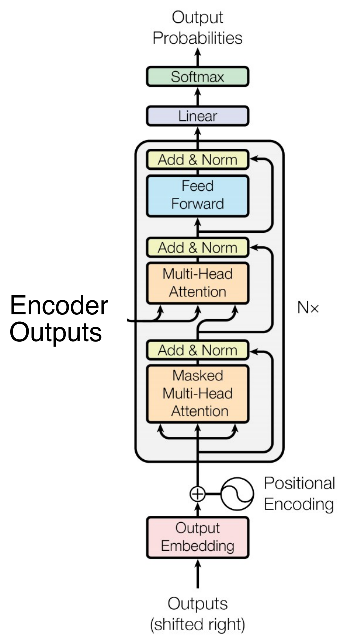
Illustration of Encoder Pass



[Alammar, The Illustrated Transformer]

32

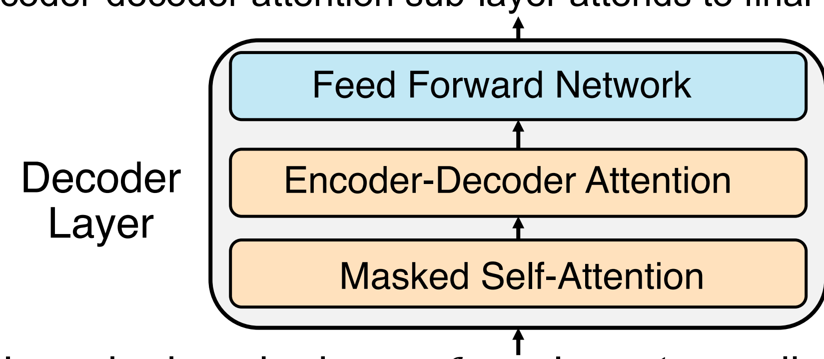
Transformer Decoder



33

Basic Decoder Structure

- The role of the decoder is to generate an output one word at a time
  - The output is generated incrementally (as in seq2seq language generation)
- A decoder layer has a self-attention, attention, and a FF sub-layer
  - Self-attention sub-layer attends to previously generated decoder output
  - Encoder-decoder attention sub-layer attends to final output of encoder

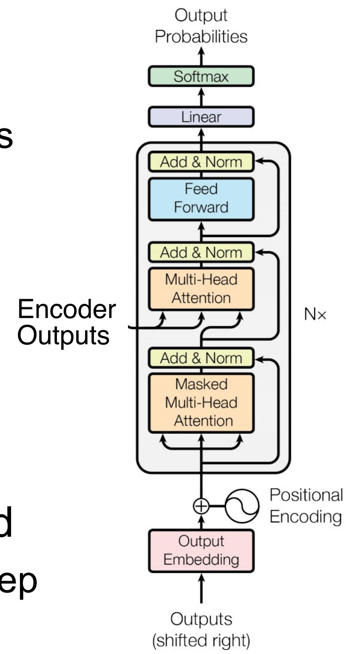


- Final layer in decoder is a *softmax* layer to predict next word

34

Transformer Decoder Summary

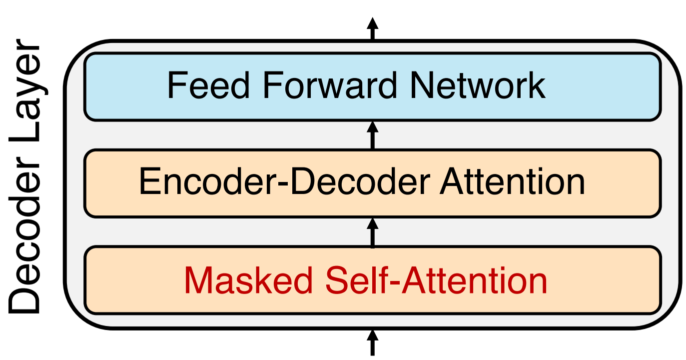
- Input represented as embedding vectors
  - Encoder & decoder use same embedding transform
  - Positional encoding vectors added to word embeddings
- Decoder layers repeated 6 times, and contain:
  - <u>*Masked*</u> multi-head attention (e.g., 8 heads)
  - <u>*Encoder-decoder multi-head attention*</u> (new)
  - Position-wise feed forward network
  - Residual connections at each sub-layer
  - Layer normalization after each sub-layer
- Final output/*softmax* layer used to predict next word
  - Current step output is fed back into decoder for next step

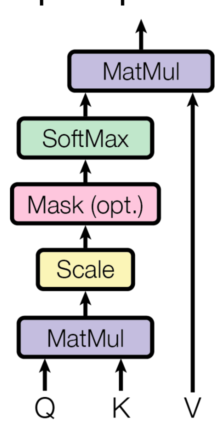


35

Masked Self-Attention

- The decoder layers all deploy a self-attention sub-layer (as encoder)
  - Query, key, and value vectors produced for each decoder layer input
- Decoder self-attention sub-layer is allowed to attend to *past* outputs
  - Accomplished by masking future outputs prior to attention *softmax*



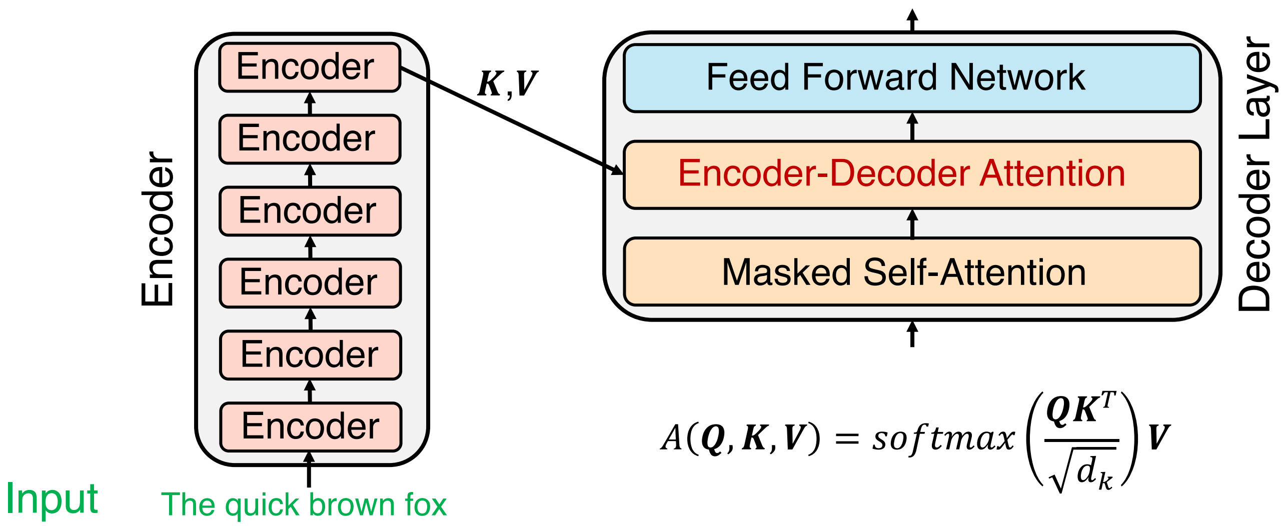


$$A(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}$$

36

Encoder-Decoder Attention

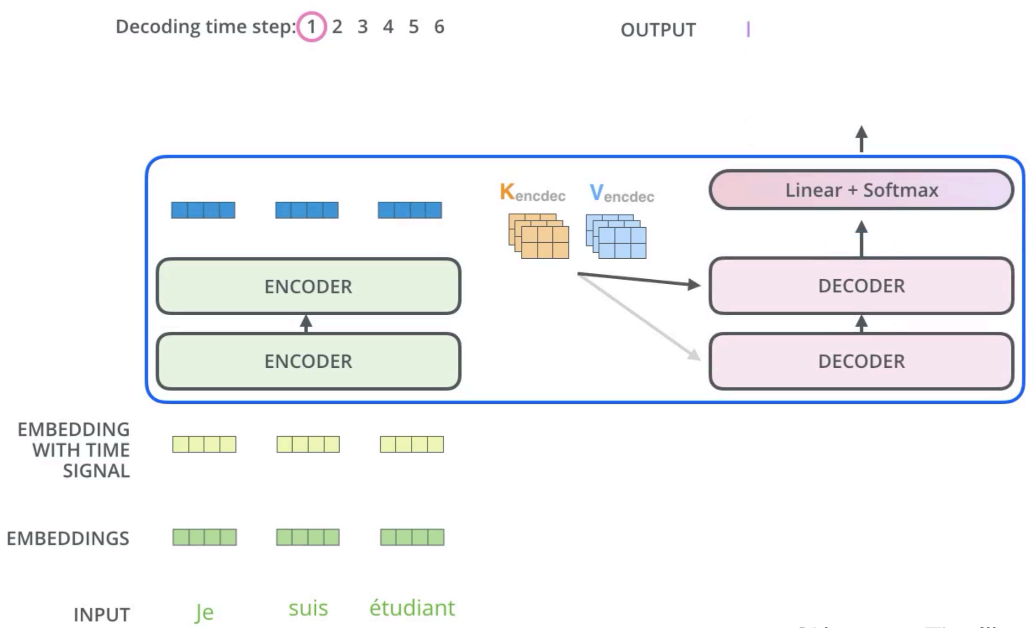
- Output of final encoder represented as key, value attention vectors
- Used by decoder layers in an “encoder attention” sub-layer



$$A(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}$$

37

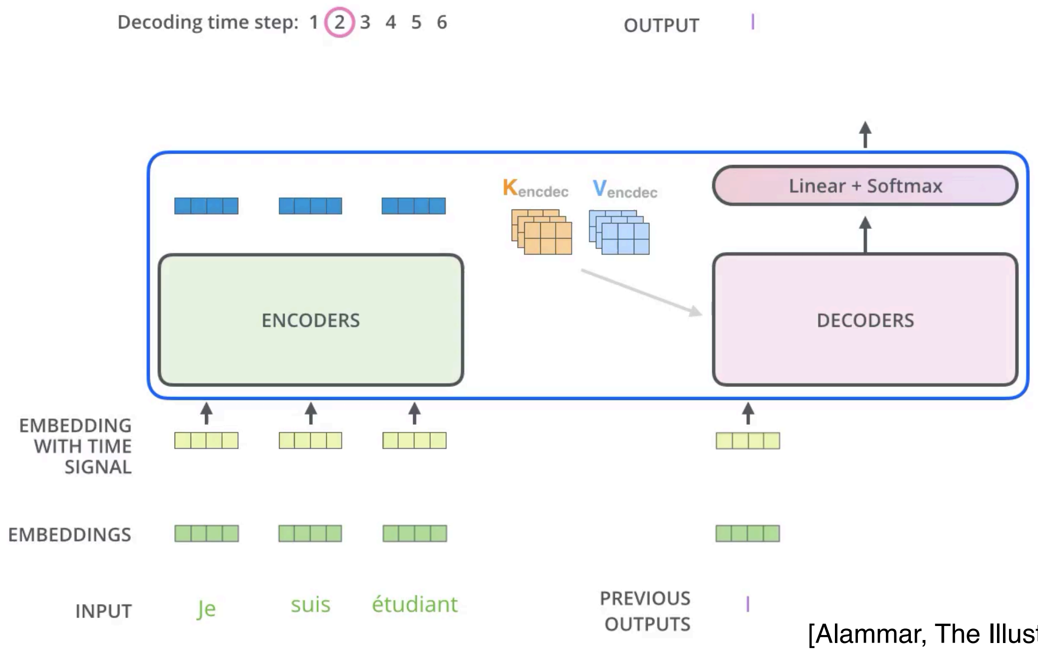
Illustration of First Decode Pass



[Alammar, The Illustrated Transformer]

38

Illustration of Decoder



[Alammar, The Illustrated Transformer]

39

Other Transformer Details

- Cross-entropy loss function is used for training

$$L(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T} L_t \qquad L_t = -\log p(w_t | w_1, \cdots, w_{t-1})$$

- During training, regularization techniques are deployed
  - Dropout is applied to all layer outputs (pre normalization) and input vectors
  - Label smoothing is applied to outputs (i.e., one-hot vector is smoothed)
- To handle large vocabularies, sub-word units are often used, e.g.,
  - Byte-pair encodings (BPE), word-piece models, …

40

Byte Pair Encoding (BPE)

- BPE is a compression technique that iteratively replaces the most frequent pair of bytes in a sequence with a single unused byte
- NLP has applied BPE to characters or character sequences
  - Addresses out-of-vocabulary (OOV) word issue (i.e., unseen words)
  - Reduces memory and computation (e.g., Transformer used ~37K BPEs)
- Initializes a symbol vocabulary with characters, and represents each word as a sequence of characters (plus end-of-word symbol)
- Iteratively counts all symbol pairs (no cross word counts), and adds most frequent pair into vocabulary, and updates dictionary entries
- Final vocabulary is initial vocabulary plus new BPE symbols
- Alternatives include Huffman encoding, word-piece models etc.

References

- Readings:
  - Jurafsky & Martin, "Speech and Language Processing," 2020 (Transformers; Sec. 9.4)

- On-line resources:
  - Rush, "The Annotated Transformer," https://nlp.seas.harvard.edu/2018/04/03/attention.html
  - Google Tensor2Tensor Colab, https://github.com/tensorflow/tensor2tensor
  - Alammar, "The Illustrated Transformer," http://jalammar.github.io/illustrated-transformer/