CS 546 – Advanced Topics in NLP

Dilek Hakkani-Tür

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

CONVAI

CDS Siebel School of Computing and Data Science

Topics for Today

Transformers

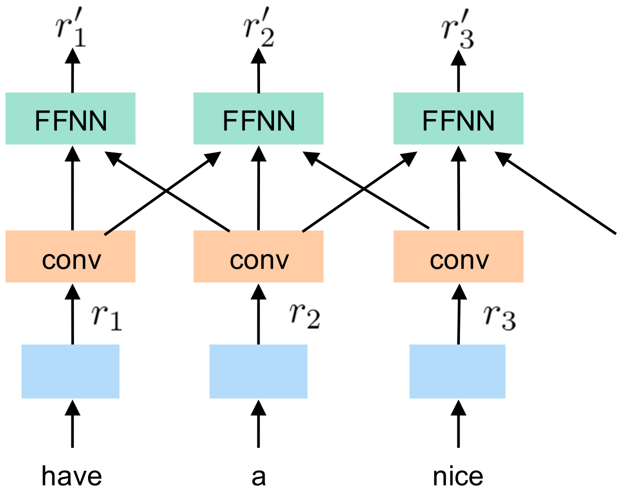
- The Transformer Model Architecture
- Self Attention
- Multi-head attention
- The encoder block
- The decoder block

Readings

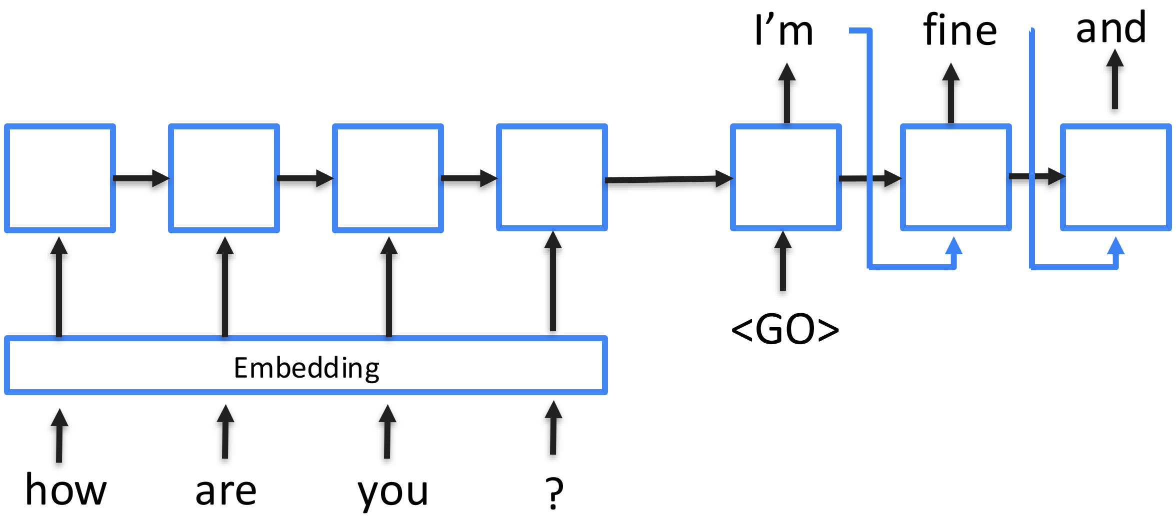
- Vaswani et al, NIPS 2017. Attention is all you need.
- Continuing Ch 11 of the Dive into Deep Learning book
- Blogs:
    - Jay Alammar, The illustrated transformer
    - PyTorch explanation by Sasha Rush: http://nlp.seas.harvard.edu/2018/04/03/attention.html

Convolutional Neural Networks

- Easy to parallelize at each layer.
- Exploit local dependencies
    - **Long-distance** dependencies require many layers



Recurrent Neural Networks



- Allow for modeling of long- and short-range dependencies (though not explicitly)
- Sequential computation is slow, and parallelization is not straightforward.
- Context window is fixed size and may not be able to store all the information => attention

Attention

- Encoder-decoder approach has been successful in NMT and other sequence-to-sequence problems.

- RNNs' attention mechanism is useful to handle long dependencies

- Attention allows us to access any state

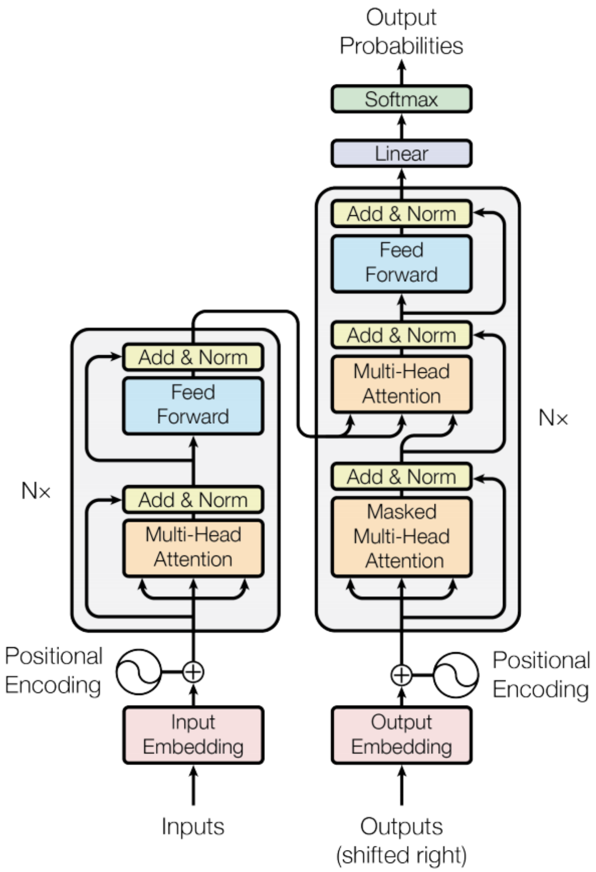
Can we use attention to replace recurrent architectures?

Topics for Today

Transformers

- The Transformer Model Architecture
- Self Attention
- Multi-head attention
- The encoder block
- The decoder block

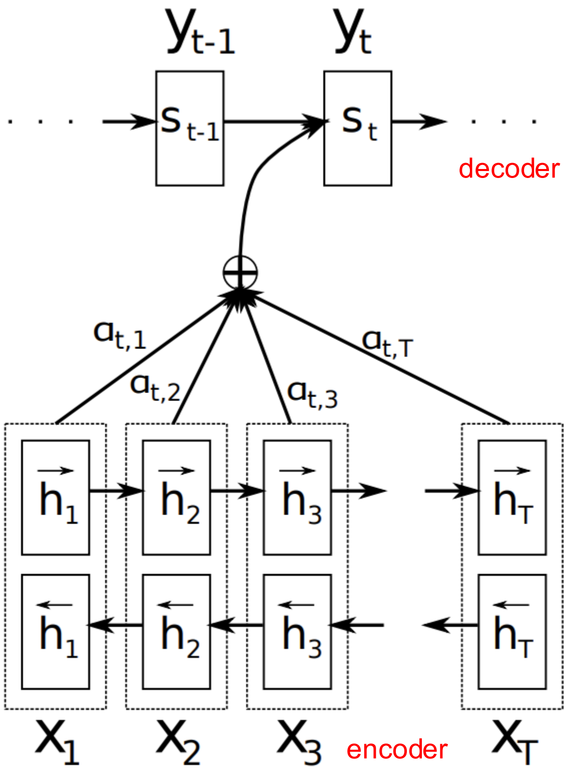
The Transformer Model Architecture

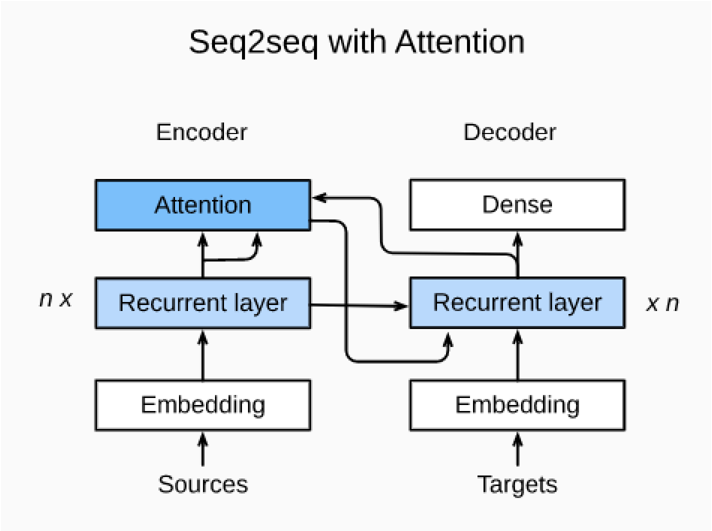


Core ideas:

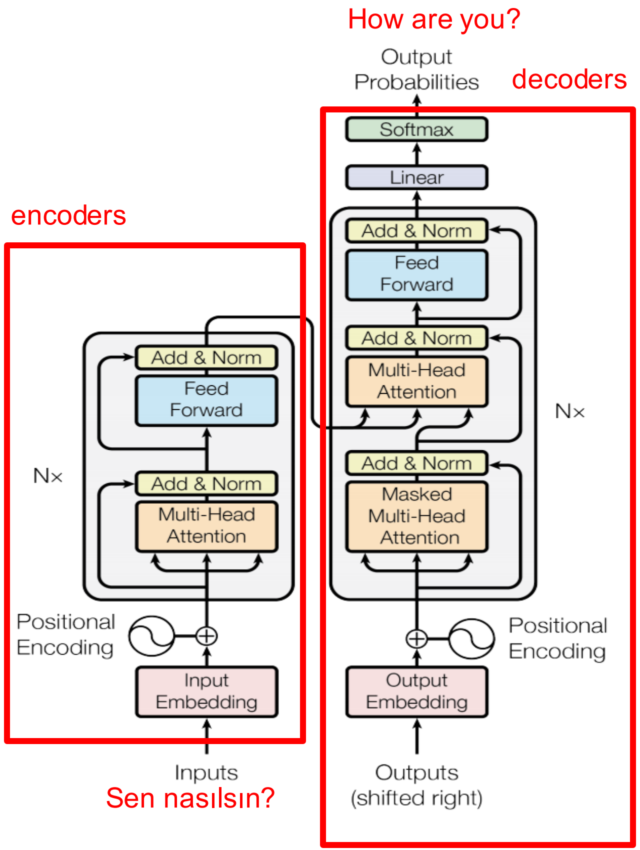
- **Self-Attention Mechanism**: Lets the model directly relate each element of a sequence to every other element, regardless of distance.
- **Parallelization**: Unlike RNNs, all tokens can be processed at once.

Encoder-Decoder RNN with Attention



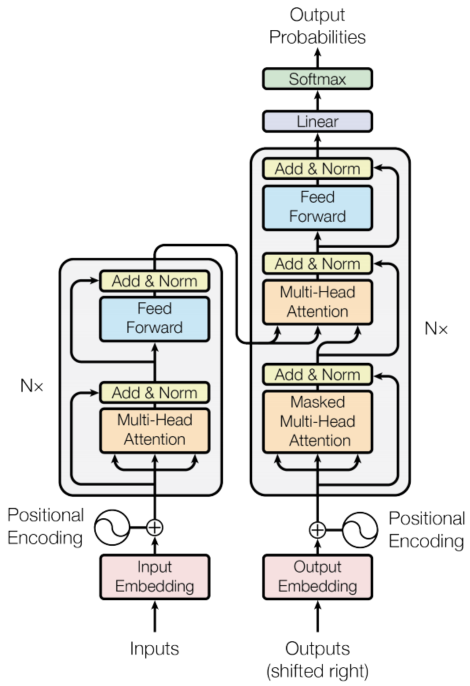


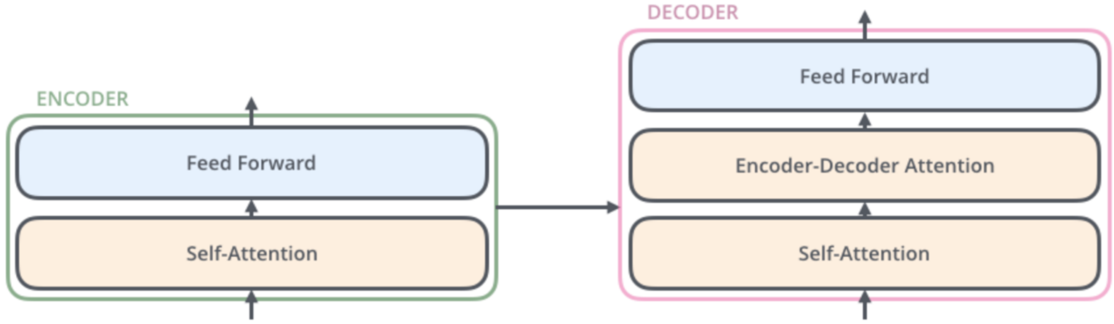
The Transformer Model Architecture



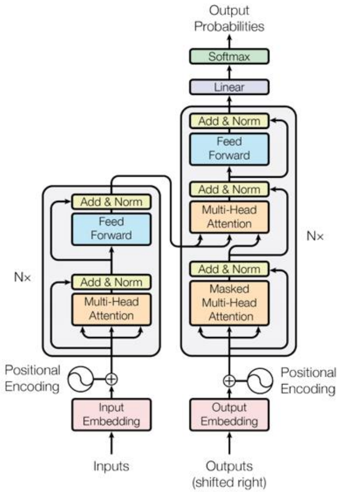
- Stack of encoder and decoder layers.
- Each have the same architecture.
- They do not share weights.

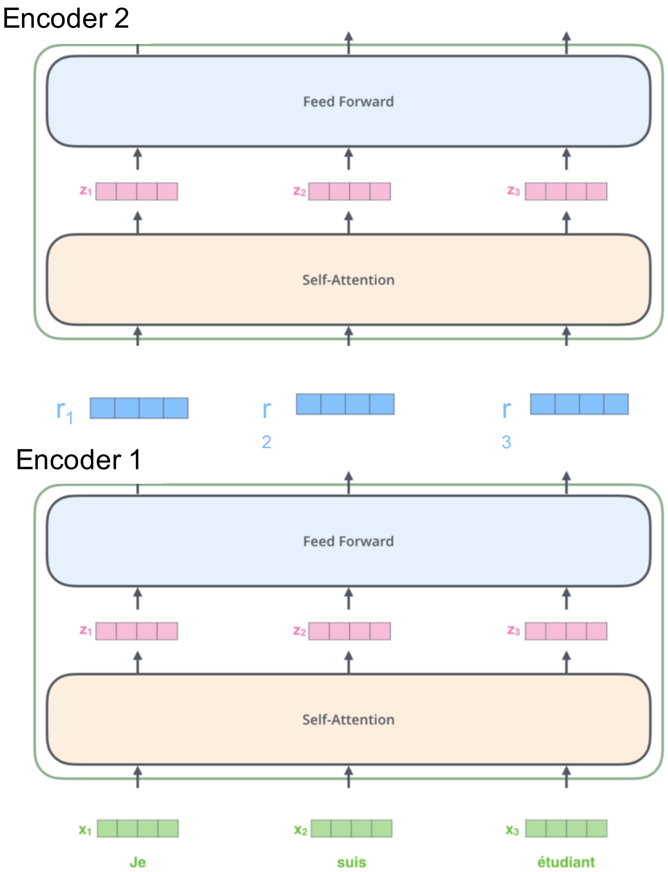
Simplified Encoder and Decoder Blocks





Stacking





The number of layers to stack is a hyper-parameter.

The original MT paper had 6 layers.

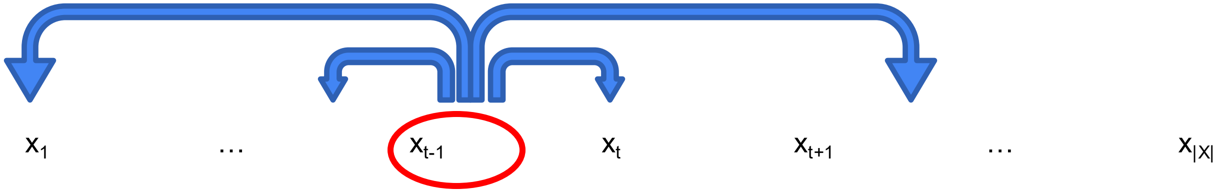
Topics for Today

Transformers

- The Transformer Model Architecture
- Self Attention
- Multi-head attention
- The encoder block
- The decoder block

Self Attention

- Introduced by the "Attention is all you need" paper.
- Instead of attending to input while decoding, self attention attends to each token in the input while encoding them.
- The aim is to capture dependencies between input tokens.



Self Attention (cont.)

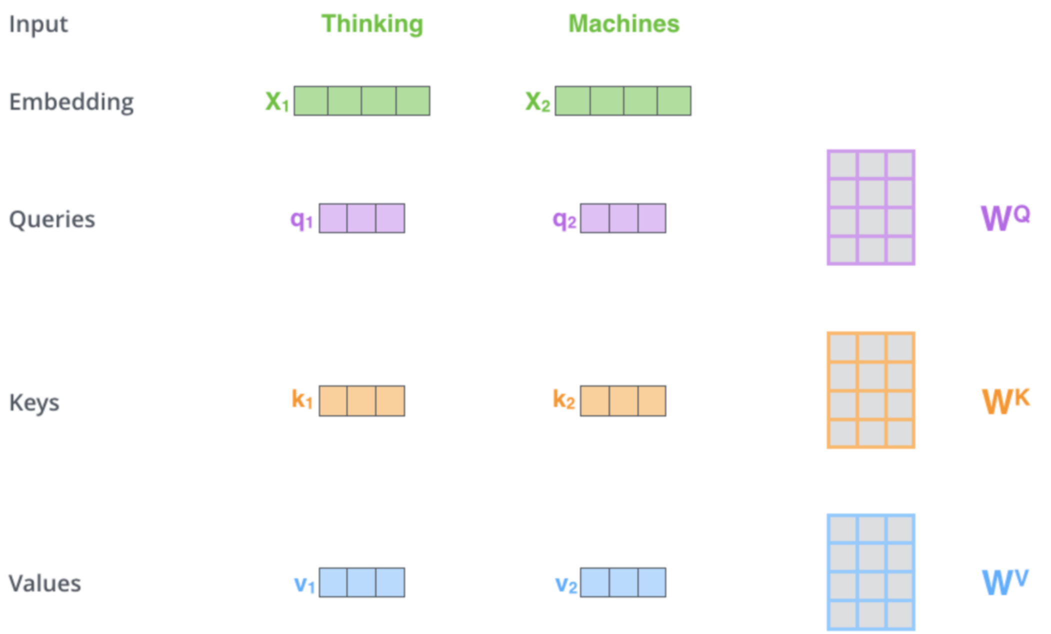
- Input: a query q and a set of key-value (k-v)
- Output: weighted sum of values

Inner product of
query and corresponding key

$$A(q, K, V) = \sum_i \boxed{\frac{\exp(q \cdot k_i)}{\sum_j \exp(q \cdot k_j)}} v_i$$

- Query q is a d_k-dim vector
- Key k is a d_k-dim vector
- Value v is a d_v-dim vector

Self Attention (cont.)



W^Q, W^K, and W^V are learned during training.

- Linear projections to obtain *q, k, v* for each token:

$$q_i = W^Q x_i$$
$$k_i = W^K x_i$$
$$v_i = W^V x_i$$

- They typically map from the embedding dimension to a smaller one.

- The separation aims to enable the model to **learn different projections** for “asking,” “indexing,” and “carrying content.”

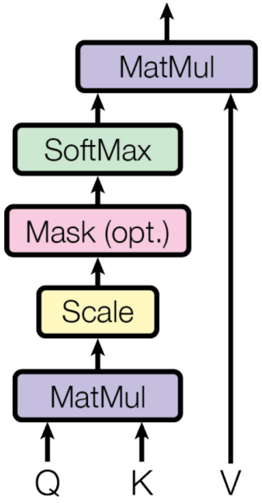
Scaled Dot Product Attention

- Problem: when d_k gets large, the variance of $q^T k$ increases
  → some values inside softmax get large
  → the softmax gets very peaked
  → hence its gradient gets smaller

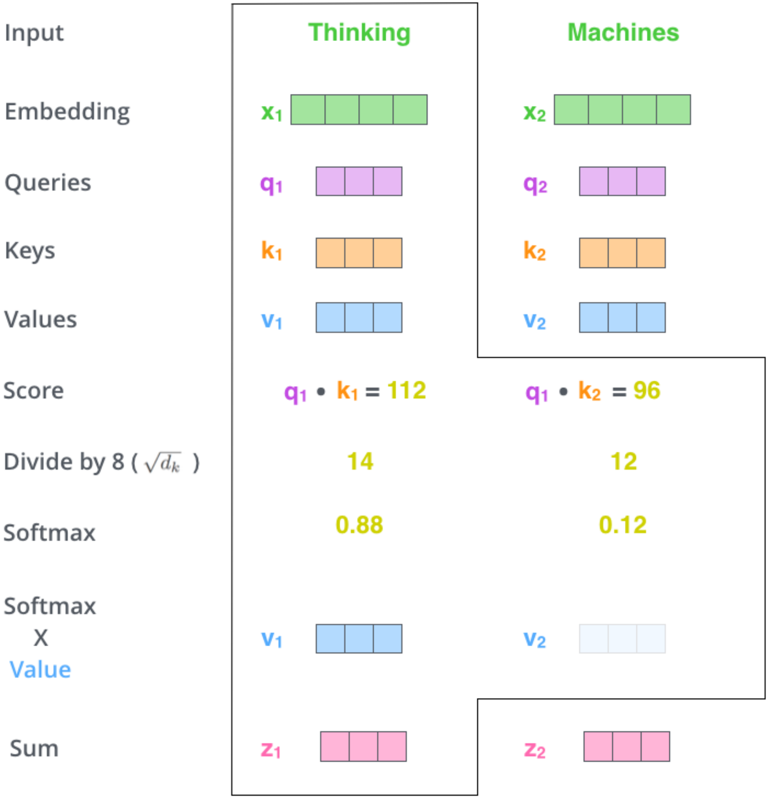
- Solution: scale by the length of query/key vectors

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

$$A(q, K, V) = \sum_i \frac{\exp(q \cdot k_i)}{\sum_j \exp(q \cdot k_j)} v_i$$



Example: Self-Attention Computation



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

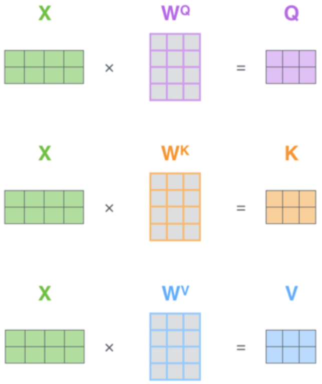
- Scaled dot product attention.
- d_k is the size of the q, k, v vectors (64 in this case).

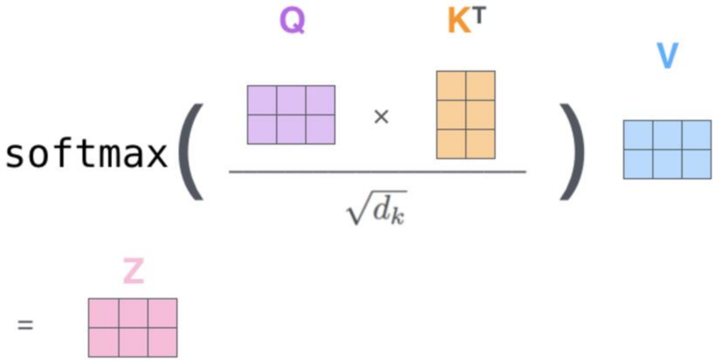
Dot-Product Attention with Matrices

- Input: *multiple* queries q and a set of key-value (k-v) pairs
- Output: a set of weighted sum of values

$$A(q, K, V) = \sum_i \frac{\exp(q \cdot k_i)}{\sum_j \exp(q \cdot k_j)} v_i$$

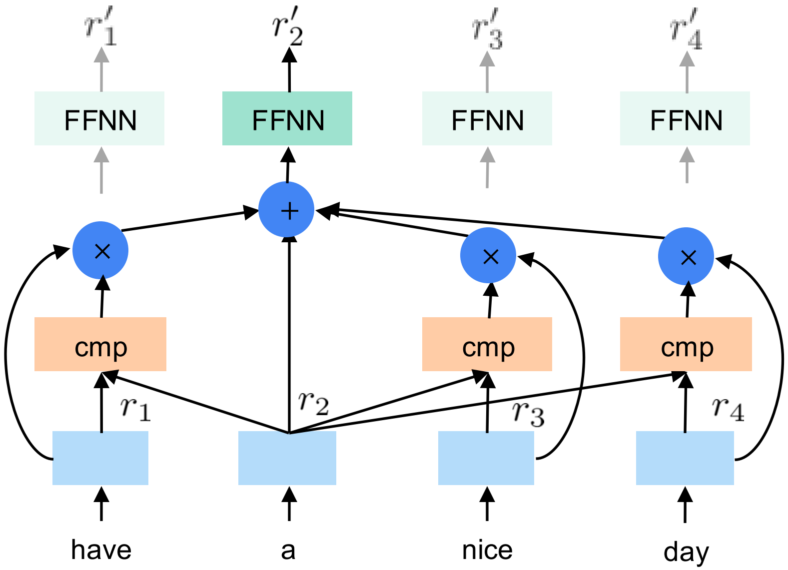
$$A(Q, K, V) = \mathrm{softmax}(QK^T)V$$



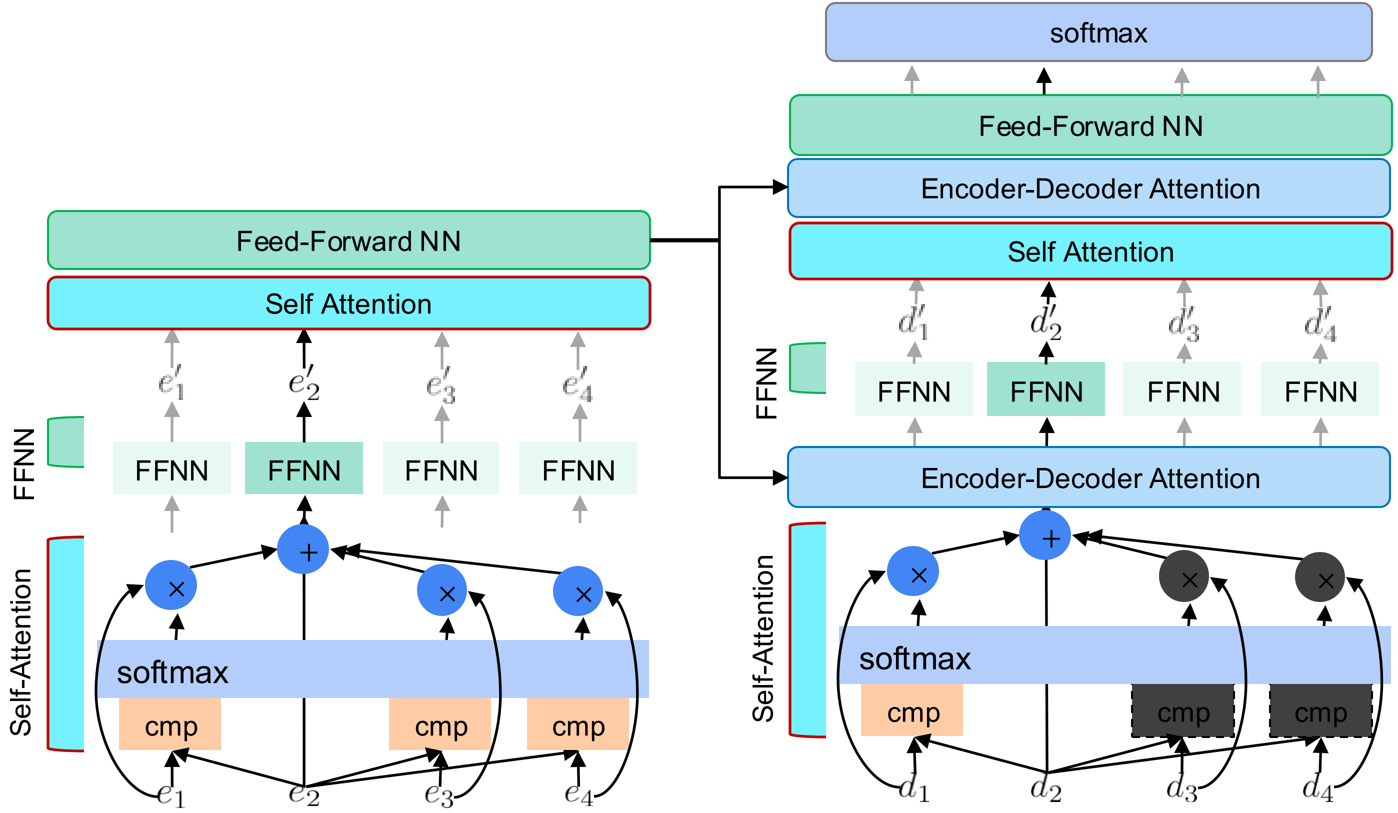


Self Attention – Parallel Computation

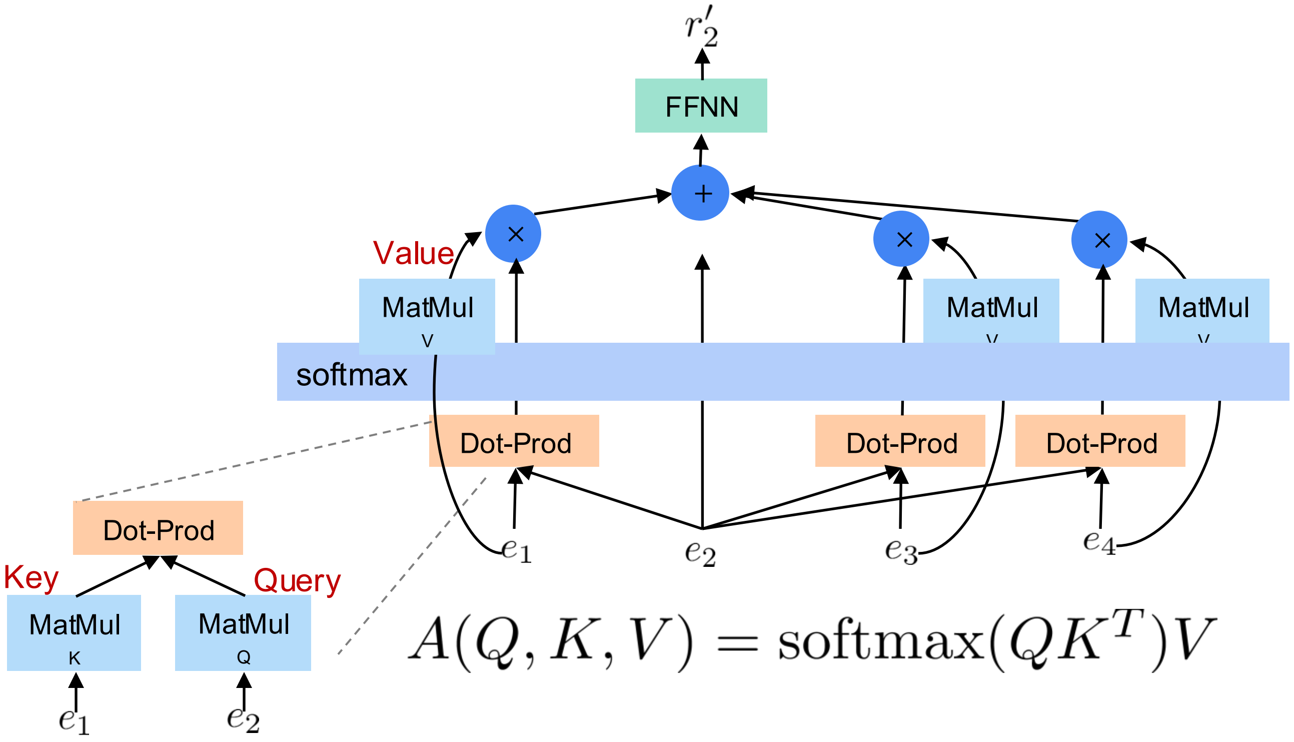
- Constant "path length" between two positions (just 1 attention hop)
- Every token can attend to every other in a single operation
- Easy to parallelize



The Transformer Model Architecture

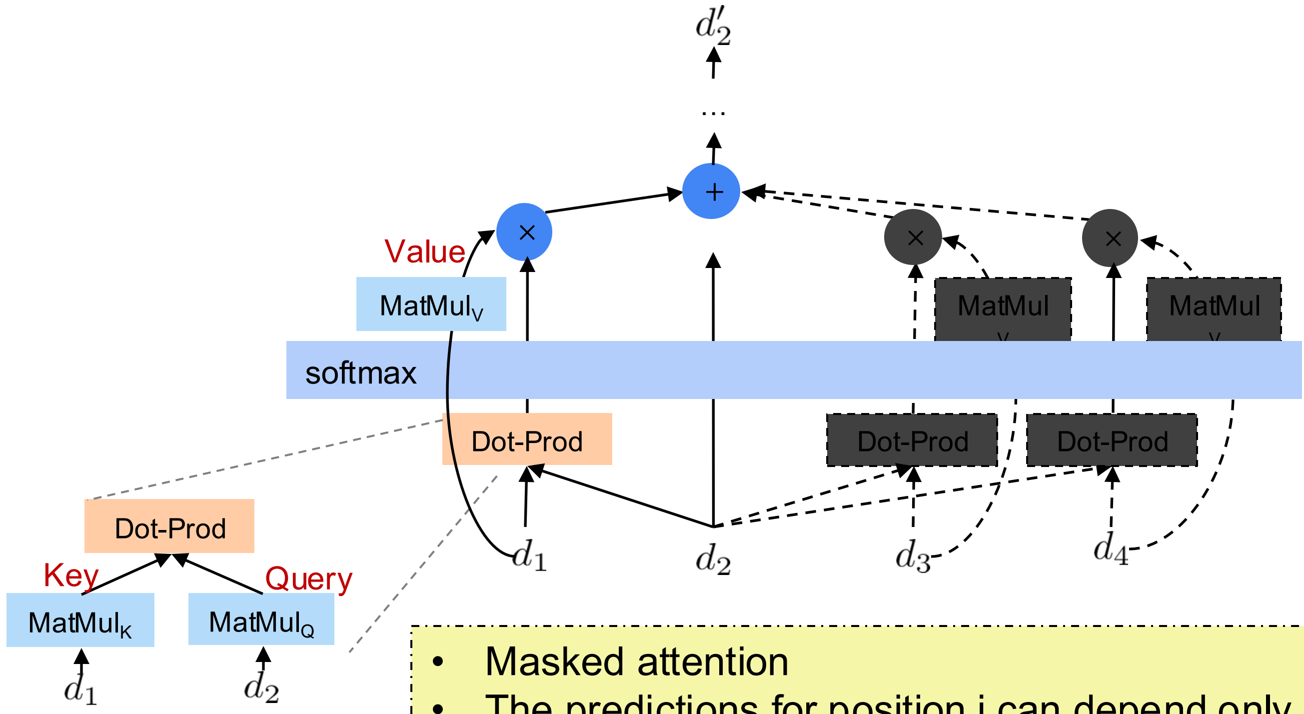


Encoder Self Attention



$$A(Q, K, V) = \mathrm{softmax}(QK^T)V$$

Decoder Self Attention



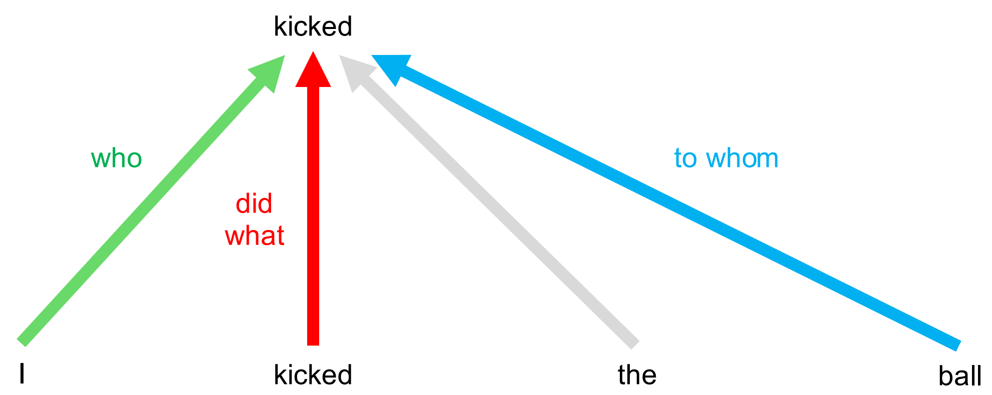
- Masked attention
- The predictions for position i can depend only on the known outputs at positions less than i.
- Implemented by setting all values in the input of the softmax which correspond to illegal connections to -∞.

Topics for Today

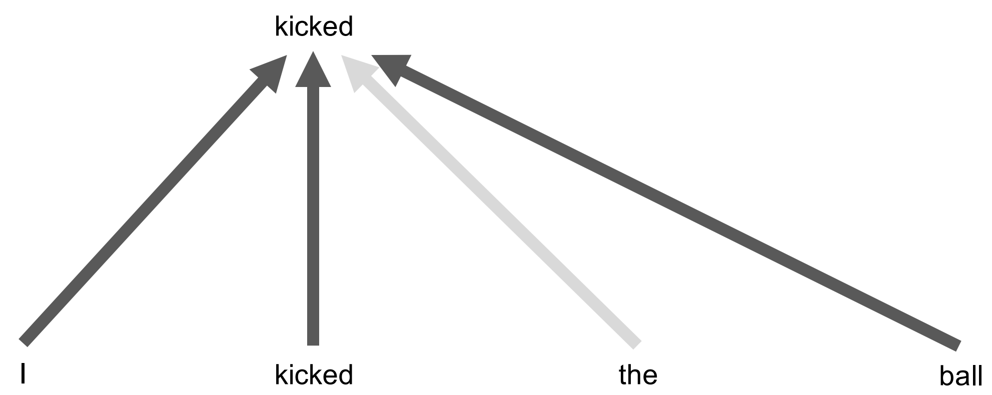
Transformers

- The Transformer Model Architecture
- Self Attention
- Multi-head attention
- The encoder block
- The decoder block

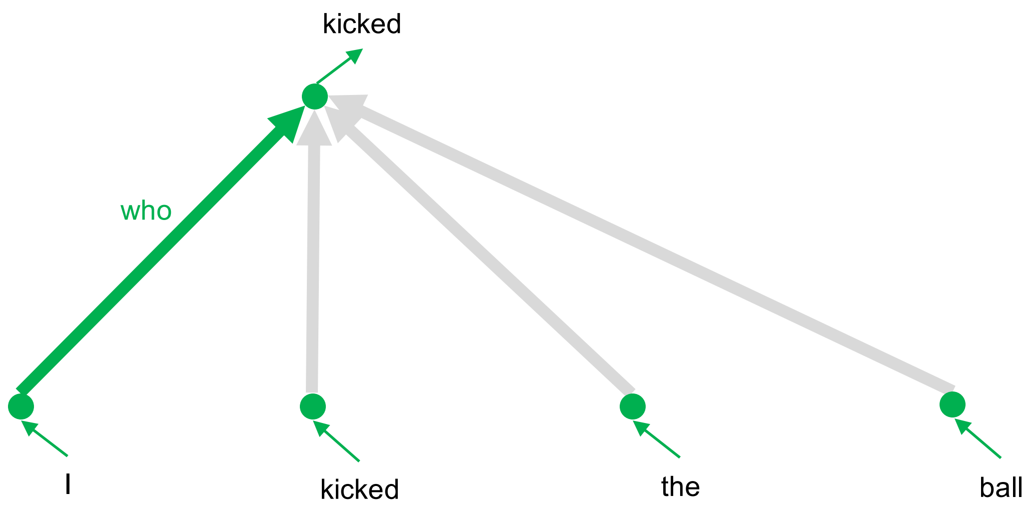
Convolutions



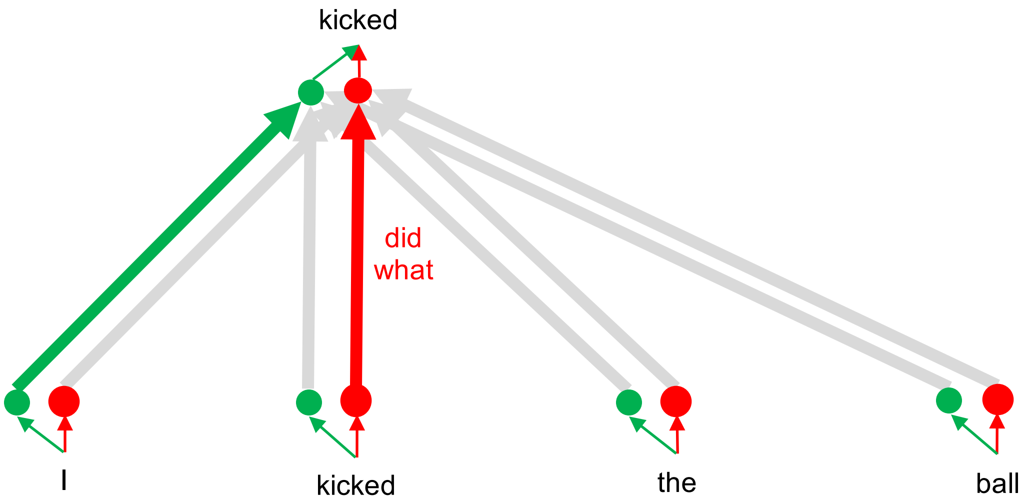
Self Attention



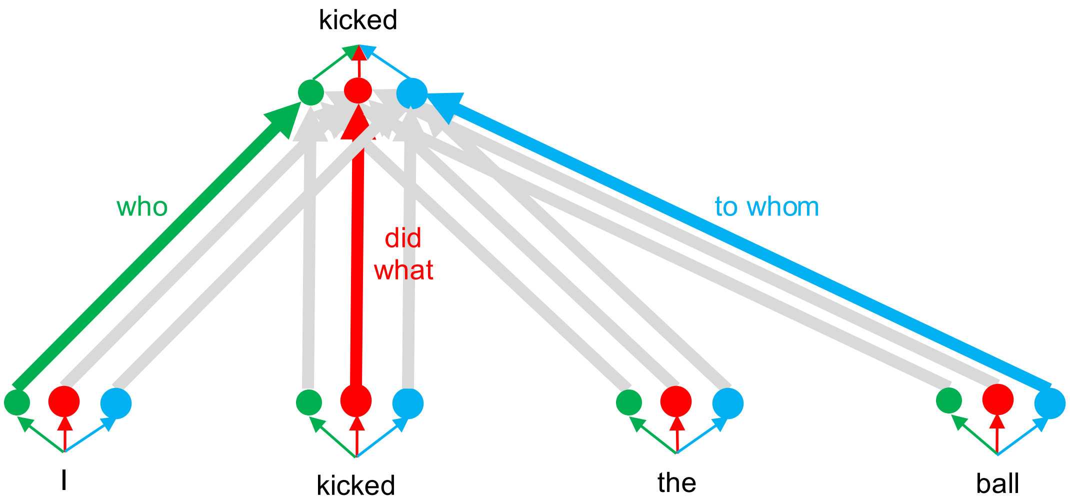
Attention Head: who



Attention Head: did what

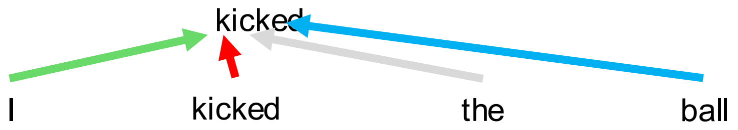


Multi-Head Attention

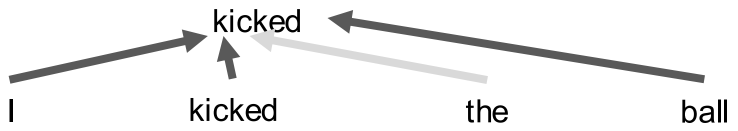


Comparison

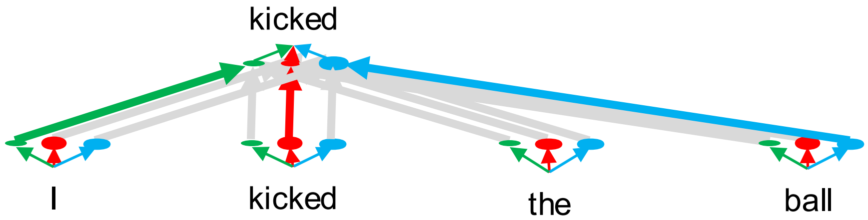
- Convolution: different linear transformations by relative positions



- Attention: a weighted average



- Multi-Head Attention: parallel attention layers with different linear transformations on input/output

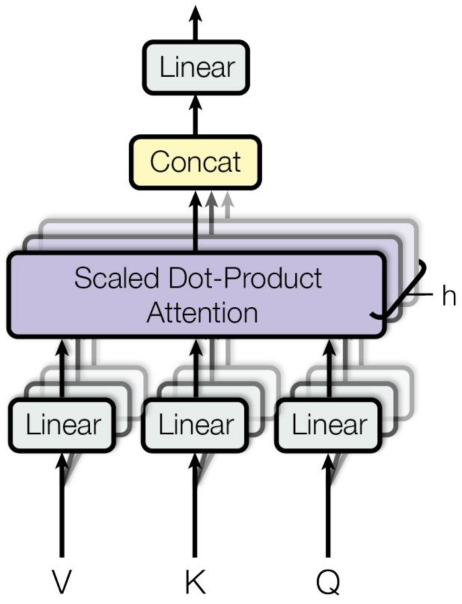


Multi-Head Attention

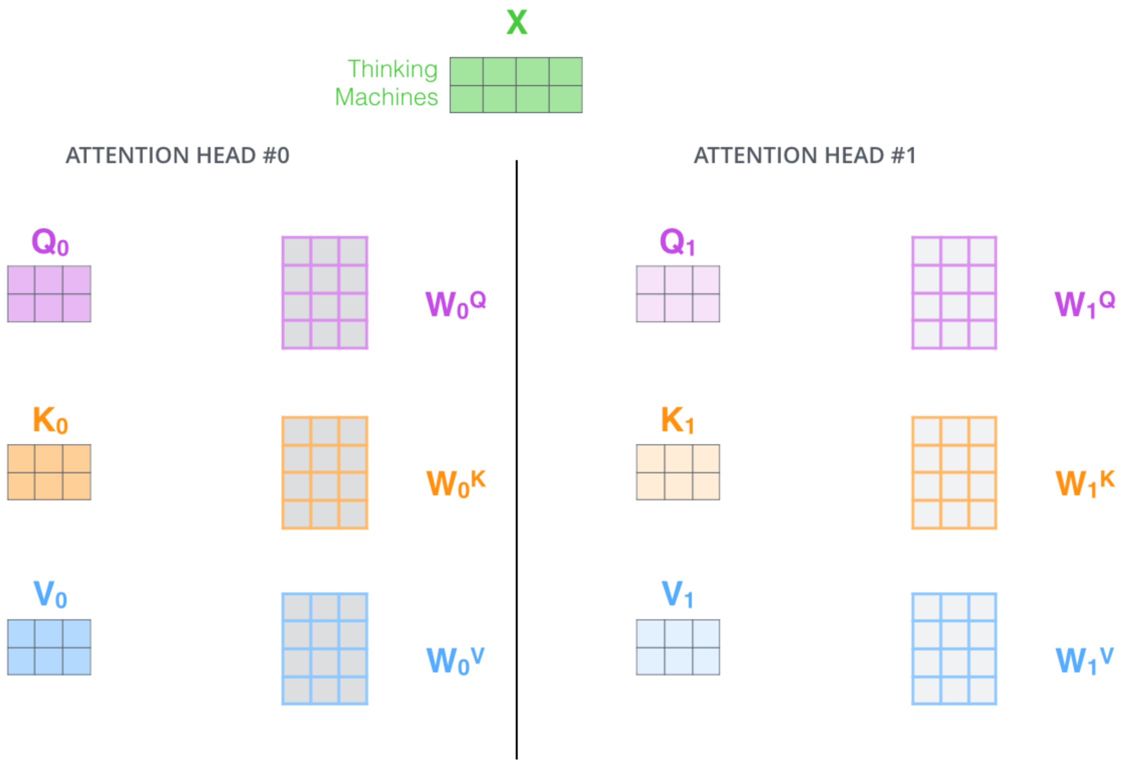
- Idea: allow words to interact with one another
- Model
  - Map V, K, Q to lower dimensional spaces
  - Apply attention, concatenate outputs
  - Linear transformation

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \cdots, \mathrm{head}_h)W^O$$

$$\mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

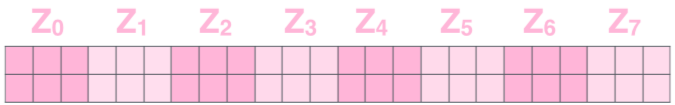


Multi-Head Attention (cont.)



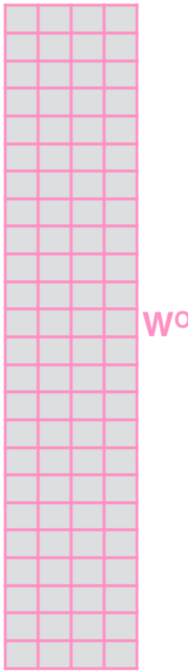
Multi-Head Attention (cont.)

1) Concatenate all the attention heads



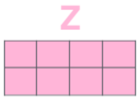
2) Multiply with a weight matrix W^O that was trained jointly with the model

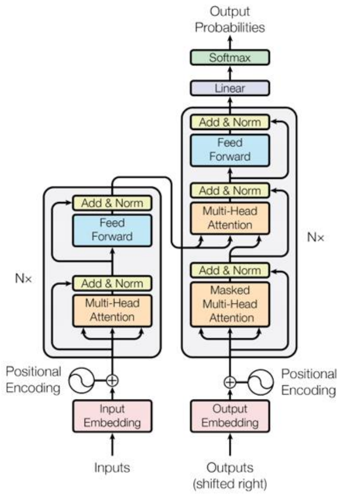
X



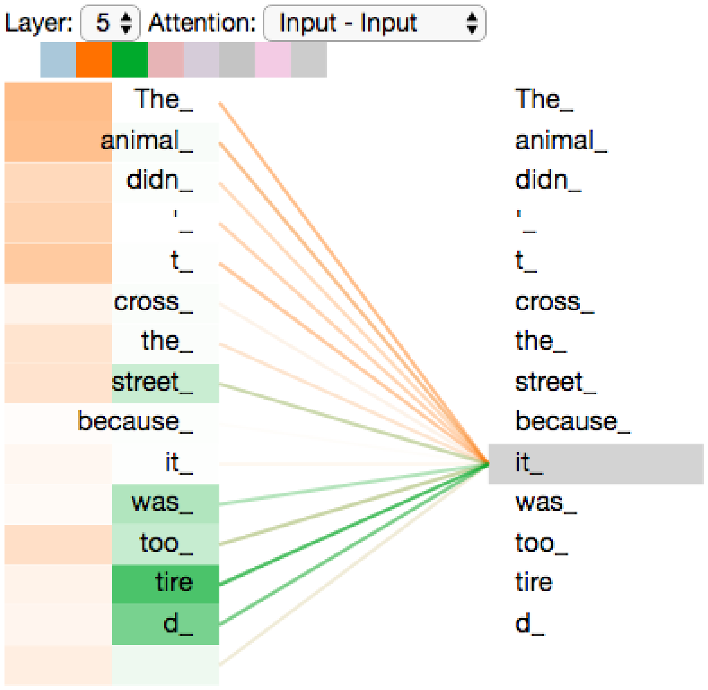
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=

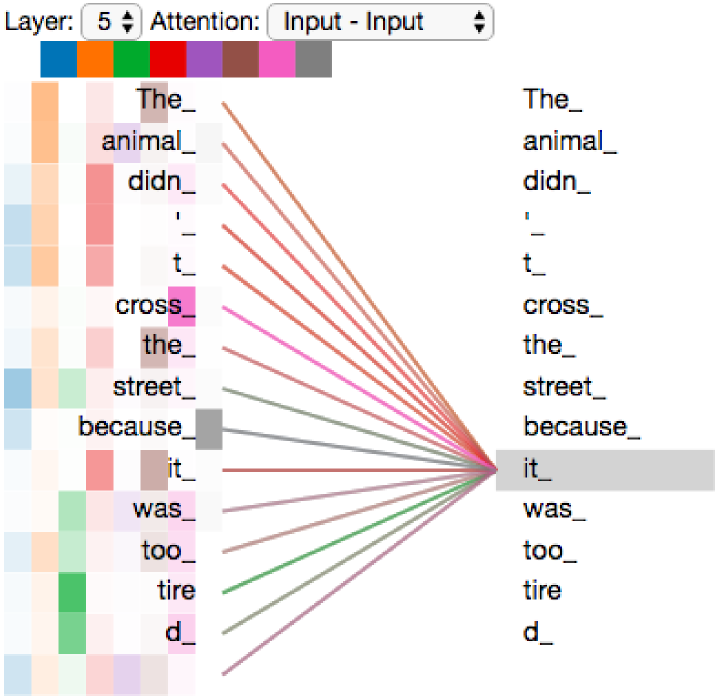




Visualization of Attention



2 attention heads



8 attention heads

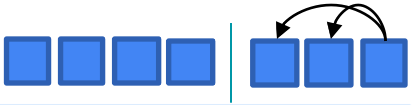
Multi-Head Attention – Details

encoder self attention

1. Multi-head Attention
2. Query=Key=Value

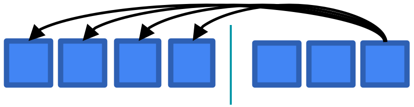
decoder self attention

1. Masked Multi-head Attention
2. Query=Key=Value



encoder-decoder attention

1. Multi-head Attention
2. Encoder Self attention=Key=Value
3. Decoder Self attention=Query



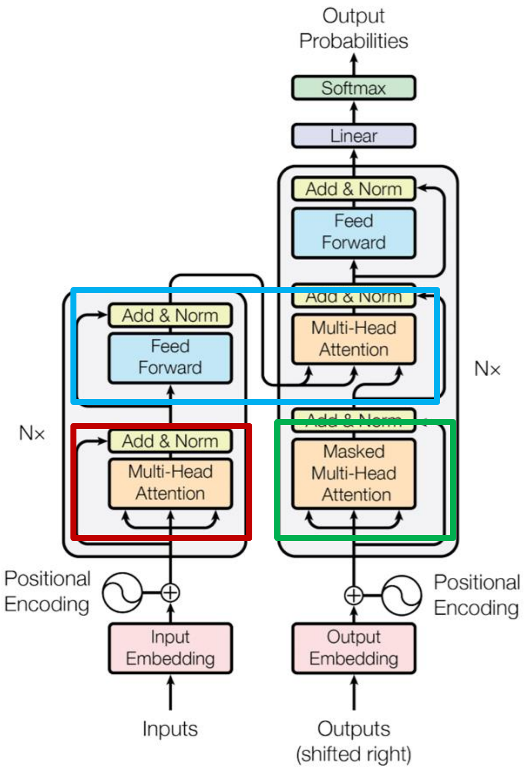


Figure from: https://bgg.medium.com/seq2seq-pay-attention-to-self-attention-part-2-cf81bf32c73d

Topics for Today

Transformers

- The Transformer Model Architecture
- Self Attention
- Multi-head attention
- The encoder block
- The decoder block

Encoder Input

- Problem: temporal information is missing
- Solution: **positional encoding** allows words at different locations to have different embeddings with fixed dimensions

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

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

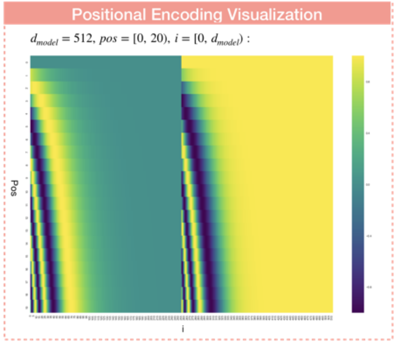
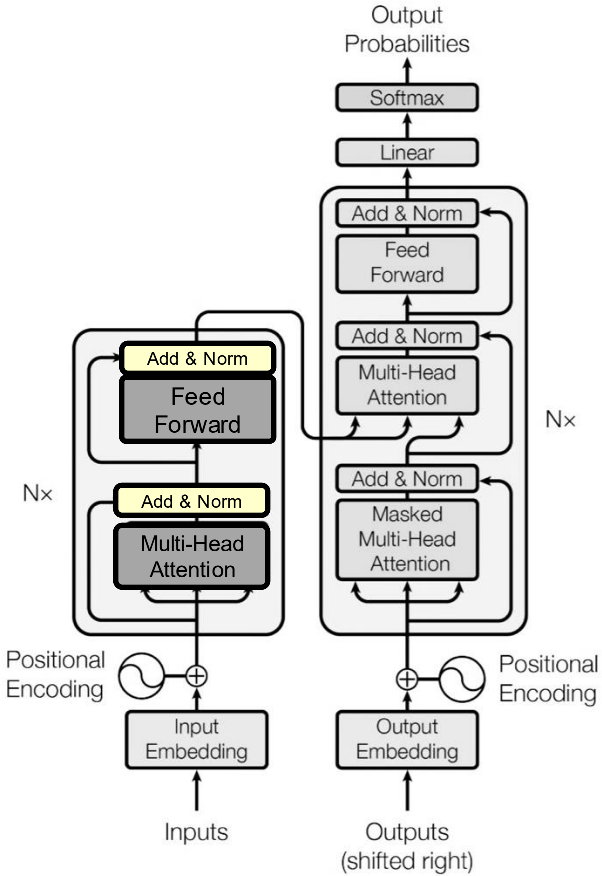
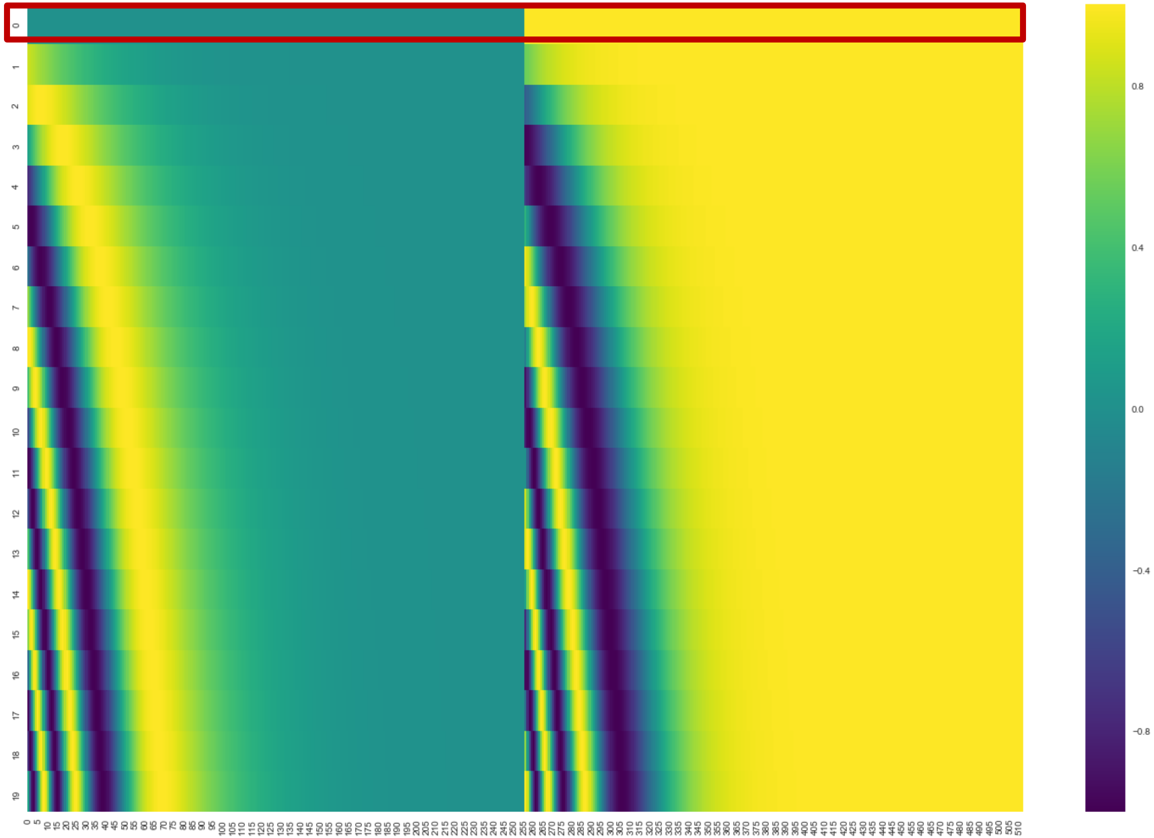


Figure from: https://bgg.medium.com/seq2seq-pay-attention-to-self-attention-part-2-cf81bf32c73d

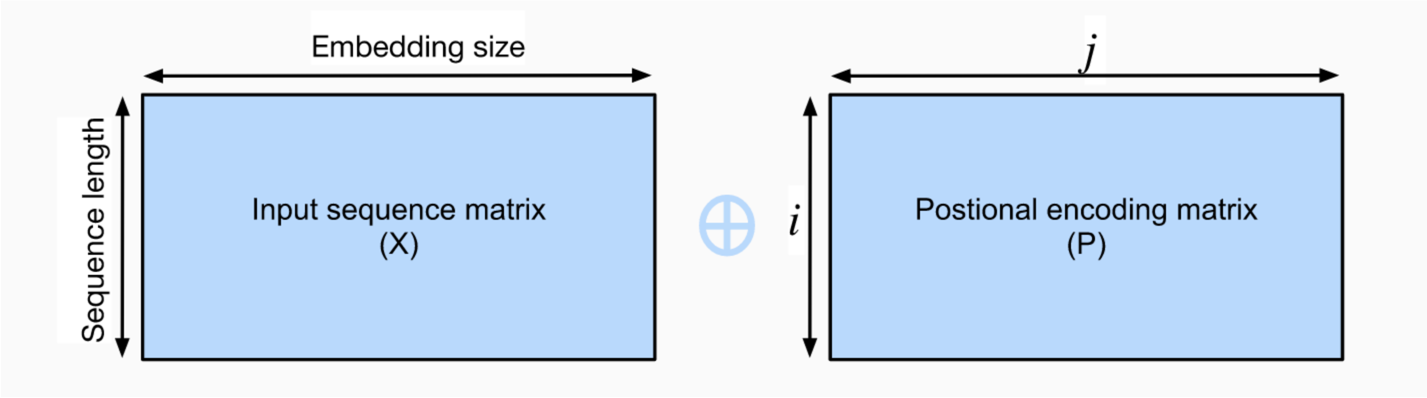


Encoder Input (cont.)

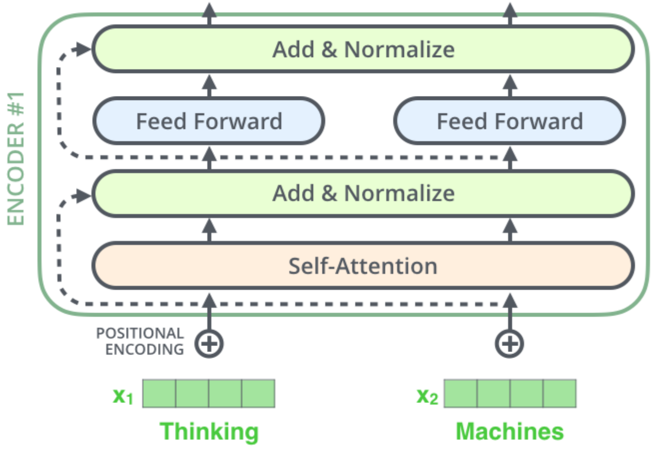


Encoder Input (cont.)

- Positional embeddings are concatenated to embeddings of each token.



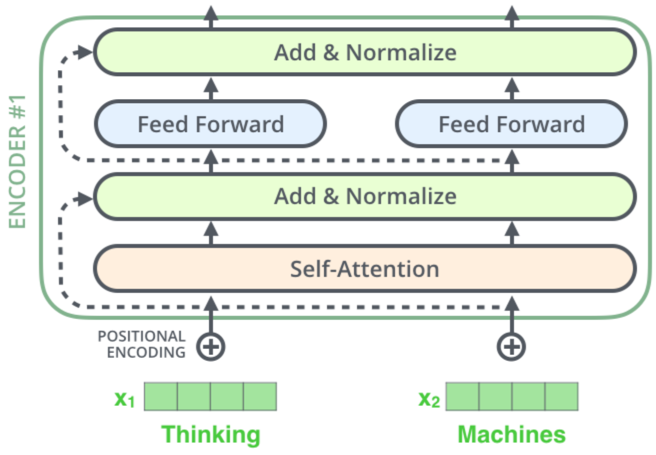
The Residuals and Layer Normalization



- Every **self-attention** and **feedforward** sublayer has:

$$\text{output} = \mathit{LayerNorm}(x + \mathit{Sublayer}(x))$$

The Residuals

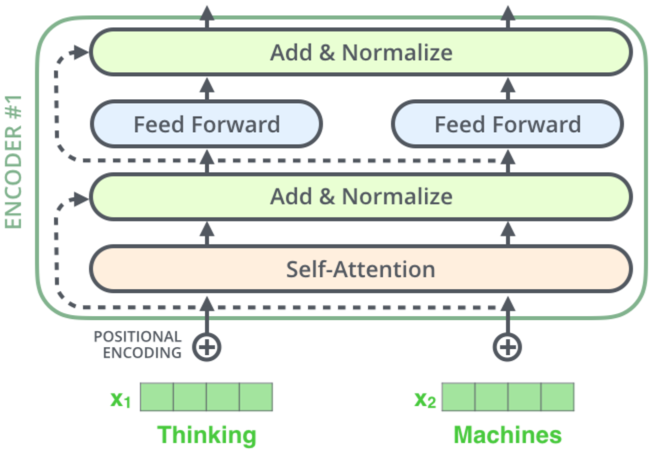


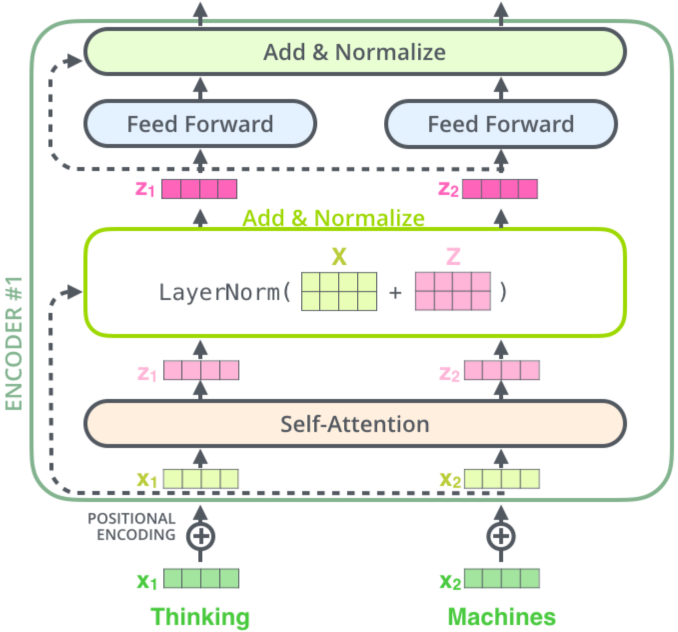
- As networks get deeper, gradients often vanish or explode.
- Even if they are okay, deep stacks of nonlinear layers are simply **hard to optimize** — they might fail to converge or converge very slowly.
- A residual block adds the input $x$back to the transformed output $F(x)$:

$$y = F(x) + x$$

- With residuals, the layer only needs to learn the **difference** from the identity mapping, and not the entire transformation.
- Useful for easier optimization, better gradient flow, stabilization and preserving information.

Layer Normalization

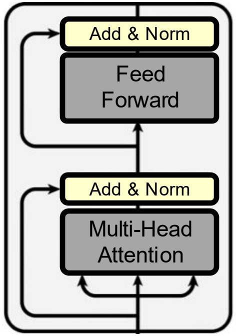




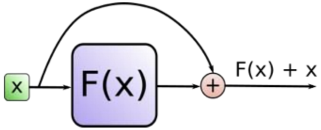
- Deep networks can suffer from **internal covariate shift**: the distribution of activations changes layer by layer, making training unstable.
- **Normalization** techniques stabilize this by keeping activations in a consistent range.

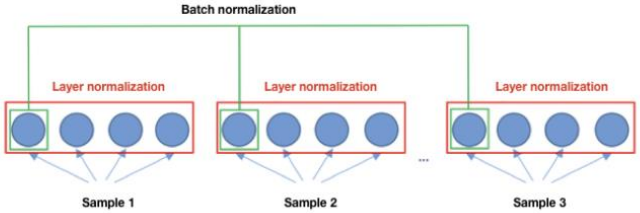
Transformer Encoder Block

- Each block has
  - multi-head attention
  - 2-layer feed-forward NN (w/ ReLU)
- Both parts contain
  - Residual connection & layer normalization (LayerNorm)
    - LayerNorm(x + sublayer(x))
    - Change input to have 0 mean and 1 variance **per layer & per training point**



$$H(x) = g(x) = x + F(x)$$





$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \qquad h_i = f\left(\frac{g_i}{\sigma_i}(a_i - \mu_i) + b_i\right)$$

Compute mean, variance and then normalize

[Ba et al. 2016. Layer Normalization. https://arxiv.org/pdf/1607.06450.pdf](https://arxiv.org/pdf/1607.06450.pdf)

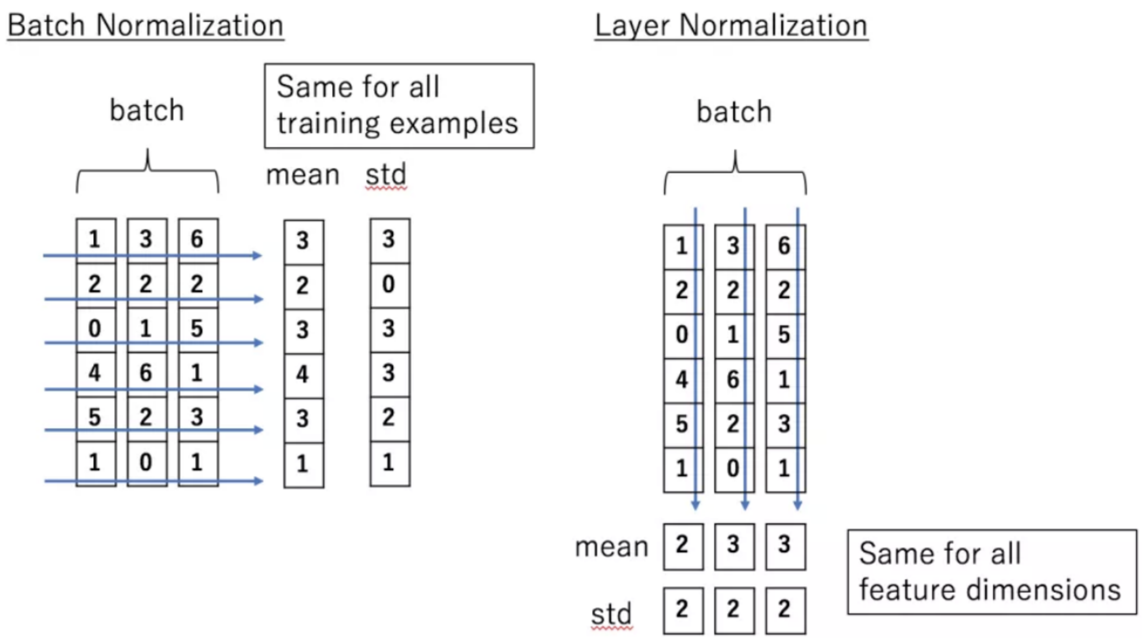
Batch vs. Layer Normalization

Batch normalization

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_{ij}$$
$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_{ij} - \mu_j)^2$$
$$\hat{x_{ij}} = \frac{x_{ij}-\mu_j}{\sqrt{\sigma_j^2+\epsilon}}$$

Layer normalization:

$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} x_{ij}$$
$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m}(x_{ij} - \mu_i)^2$$
$$\hat{x_{ij}} = \frac{x_{ij}-\mu_i}{\sqrt{\sigma_i^2+\epsilon}}$$



Unlike **BatchNorm**, which normalizes across a batch of examples, **LayerNorm normalizes across the features of a single example** — perfect for sequences and variable batch sizes.

- https://mlexplained.com/2018/01/13/weight-normalization-and-layer-normalization-explained-normalization-in-deep-learning-part-2/

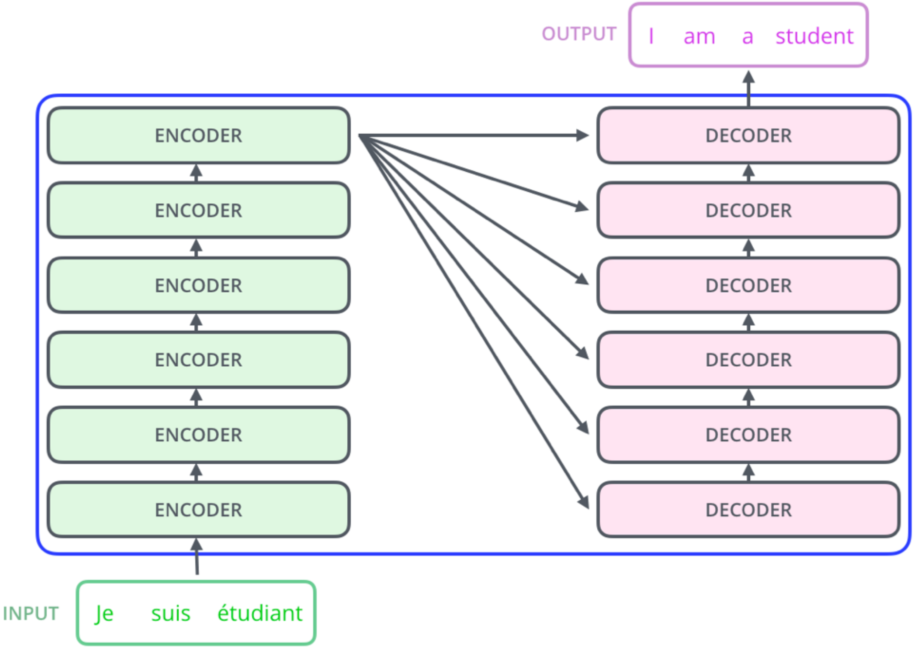
Topics for Today

Transformers

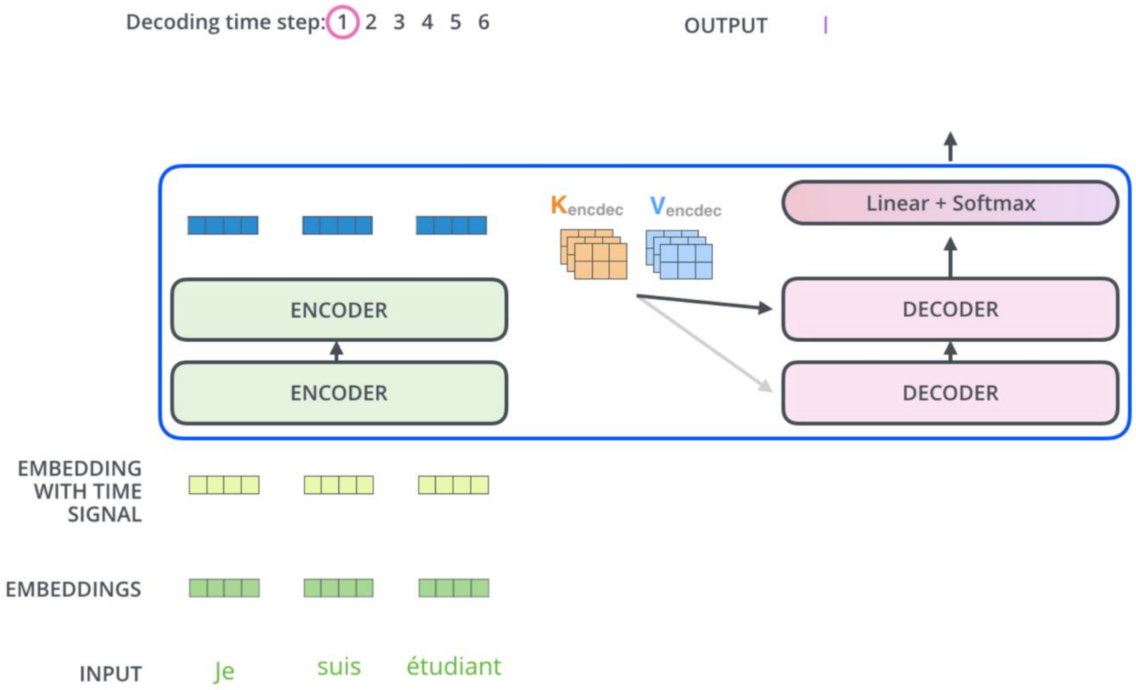
- The Transformer Model Architecture
- Self Attention
- Multi-head attention
- The encoder block
- The decoder block

The Decoder

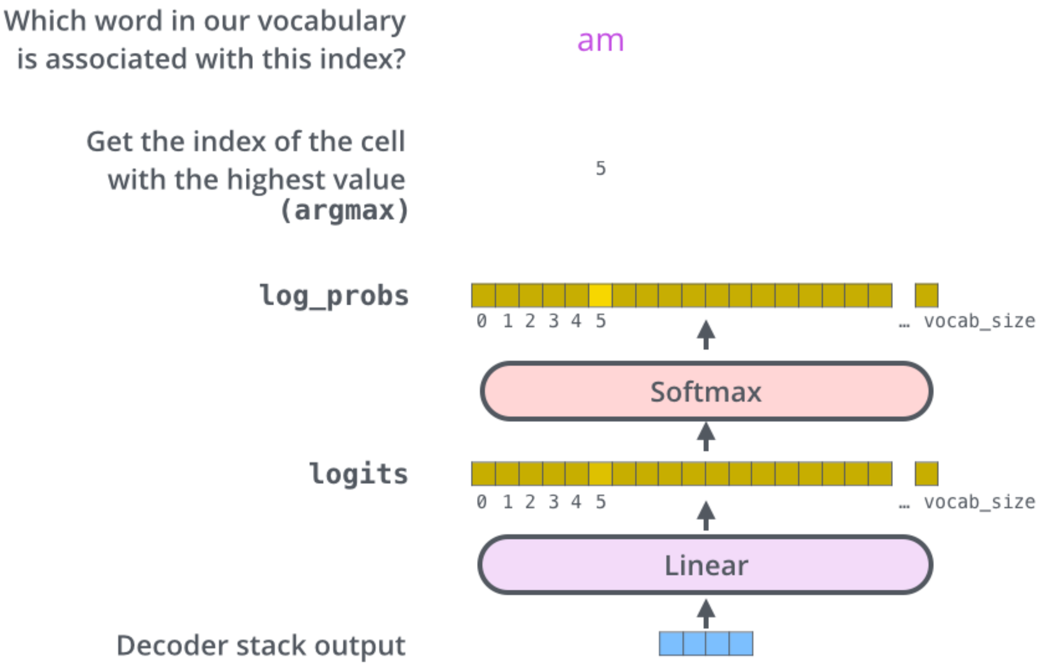
- The decoder block is a stack of decoders of the same number.



The Decoder (cont.)



The Decoder (cont.)

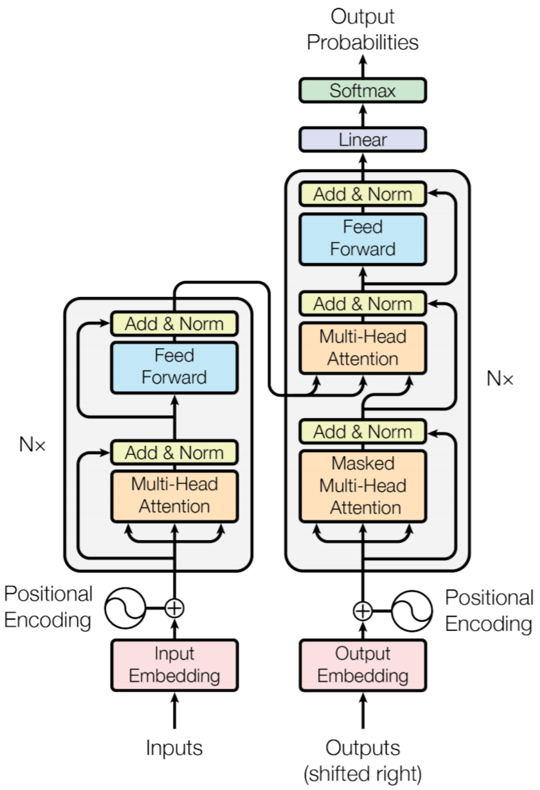


Variations:

- Beam search instead of greedy search
- Top-k sampling, nucleus sampling, etc.

Transformer Overview

- Non-recurrent encoder-decoder for MT
- PyTorch explanation by Sasha Rush
  - http://nlp.seas.harvard.edu/2018/04/03/attention.html



MT Experiments

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Parsing Experiments

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

Training Tips

- Byte-pair encodings (details next week)
- ADAM optimizer with learning rate changes
- Dropout during training at every layer just before adding residual
- Auto-regressive decoding
    - Use previous time step output as input
- Beam search and length penalties (beam size = 4, α=0.65)
    - Short utterances are favored in MT, so scores (log-probabilities) are divided by $length^{\alpha}$
- Label smoothing

Label Smoothing

- Regularization technique that aims to deal with the problem of over-confidence on outputs.
- Replaces one-hot encoded target label vector y_{hot} with a mixture of y_{hot} and the uniform distribution:

$$y_{ls} = (1 - \alpha) \cdot y_{hot} + \alpha \cdot \frac{1}{K}$$

K: the number of label classes

α: hyperparameter that determines the amount of smoothing.

If $\alpha = 0 \rightarrow$ the original one-hot encoded y_{hot}.

If $\alpha = 1 \rightarrow$ uniform distribution.

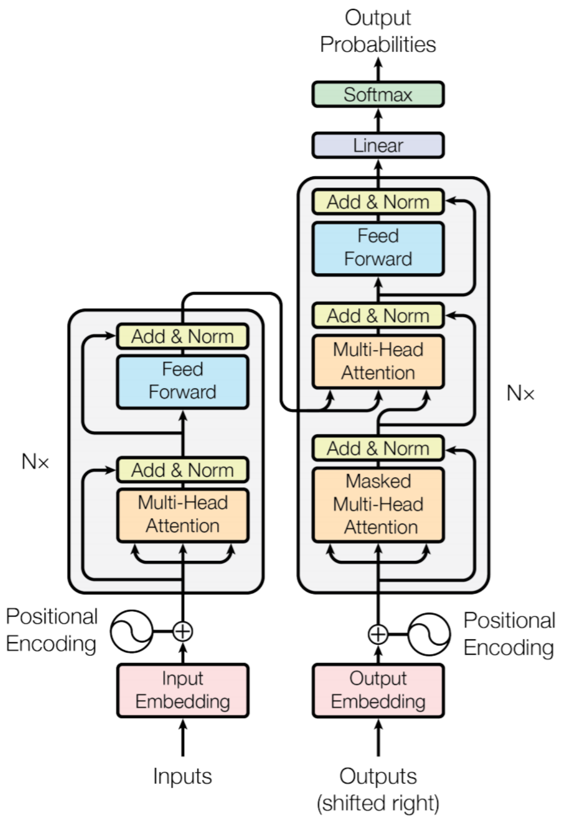
Size of the Transformer Network

- Directly correlated with a set of hyper-parameters.

- Number of:
  - Layers
  - Attention heads
  - Embeddings

Concluding Remarks

- **Non-recurrence** model is easy to parallelize
- **Multi-head attention** captures different aspects by interacting between words
- **Positional encodings** capture location information
- Each transformer block can be applied to diverse tasks



Concluding Remarks (cont.)

- Modern LLMs are using transformers, more recently the focus is on auto-regressive/decoder-based models.
- Several improvements since the original architecture, examples:
  - Variations on positional embeddings, such as, relative position embeddings in TransformerXL (Dai et al., 2019), KERPLE, kernelized relative position embeddings (Chi et al, 2022).
  - Variations on attention structure, such as, sparse attention to relax the quadratic computation complexity for long sequences (Child et al., 2019), flash attention (Dao et al., 2022) and extensions, gated attention/ mixture of experts (Lepikhin et al., 2020) and switch transformers (Fedus et al., 2022).
  - Parameter efficiency during training, such as, adapter layers (Houlsby et al., 2019) and Low-Rank Adaptation (Hu et al., 2021).
  - Changes in ordering of layer normalization, for example, original version: Post-LN versus Pre-LN (Xiong et al, 2020).

Topics for Next Week

Tuesday:

- Pre-training and Fine-tuning

Thursday

- Prompting & in-context learning

Homework 1

Due: Tuesday, October 14th, 2025

Goals: explore how pretrained language models can be adapted for NLP tasks

1. Establish a baseline
2. Fine-tune a model
3. Evaluate
4. Build good practices

Homework 1

Due: Tuesday, October 14th, 2025

Goals: explore how pretrained language models can be adapted for NLP tasks

You will learn: fine-tuning, evaluation, and transfer learning

1. The difference between frozen and fine-tuned representations
2. How to set up training and evaluation loops
3. How to critically assess model performance beyond accuracy

Homework 1

- **NLP Task:** Sentiment Classification on the IMDB movie reviews dataset

- Metrics such as recall, precision, and F1-scores for evaluation

Homework 1

CONVAI

Due: Tuesday, October 14th, 2025

What are you supposed to do?

1. Fill in the TODO's in the .ipynb notebook

```python
# TODO: count how many 0s and 1s are in train_labels
num_train_zeros = ...
num_train_ones  = ...

# TODO: count how many 0s and 1s are in test_labels
num_test_zeros  = ...
num_test_ones   = ...

print(f"training:\n\t# of 0s: {num_train_zeros}\n\t# of 1s: {num_train_ones}\n"
      f"\ntesting:\n\t# of 0s: {num_test_zeros}\n\t# of 1s: {num_test_ones}")
```

Python

2. Once you're satisfied with your answers, submit your notebook to Canvas with the following name: "hw1_<YOUR_NET_ID>.ipynb"
    a. Example: "hw1_sagnikm3.ipynb"

Homework 1

Due: Tuesday, October 14th, 2025

How will you be graded?

1. Visible test cases in the HW’s notebook
2. A few hidden test cases

```python
# ✅ Tests
assert num_train_zeros == 12500, "Expected exactly 12500 zeros in train set"
assert num_train_ones == 12500, "Expected exactly 12500 ones in train set"
assert num_test_zeros == 12500, "Expected exactly 12500 zeros in test set"
assert num_test_ones == 12500, "Expected exactly 12500 ones in test set"
print(“Label distribution tests passed!”)

# 🔍 Extra tests
assert (num_train_zeros + num_train_ones) == len(train_labels), "Mismatch in train label counts"
assert (num_test_zeros + num_test_ones) == len(test_labels), "Mismatch in test label counts"
print(“Extra label consistency tests passed!”)
```
Python

Homework 1

Due: Tuesday, October 14th, 2025

Bonus points!!

1. Upload your model to HuggingFace (instructions here: https://mediaspace.illinois.edu/media/t/1_cr5kp3vd)
2. Enter your model name and your email id on the google sheet here: https://docs.google.com/spreadsheets/d/1hC5i2Q6JcLvHAz2TOvAMmommita5XlpiypR4wMqMCYI/edit?usp=sharing

A	B	C	D
Student Name	Illinois email id	HF model path	
Ishika Agarwal	ishikaa2@illinois.edu	ishikaa2/adv-nlp-hw1	