CS60010: Deep Learning

Spring 2023

Sudeshna Sarkar

Transformer- Part 1

Sudeshna Sarkar

14-15 Mar 2023

Positional encoding

Naïve positional encoding: just append t to the input

$$\bar{x}_t = \begin{bmatrix} x_t \\ t \end{bmatrix}$$

But **absolute** position is less important than **relative** position

I walk my dog every day

every single day I walk my dog

The fact that "my dog" is right after "I walk" is the important part, not its absolute position

we want to represent **position** in a way that tokens with similar **relative** position have similar **positional encoding**

Positional Encoding Layer in Transformers

- Suppose you have an input sequence of length L.
- The positional encoding of kth object is given by sine and cosine functions of varying frequencies:

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

$\boldsymbol{k}$: Position of an object in the input sequence

$\boldsymbol{d}$: Dimension of the output embedding space

$\boldsymbol{P(k, j)}$**:** Position function for mapping a position k in the input sequence to index (k, j) of the positional matrix

$\boldsymbol{n}$: User-defined scalar, set to 10,000 by the authors of Attention Is All You Need.

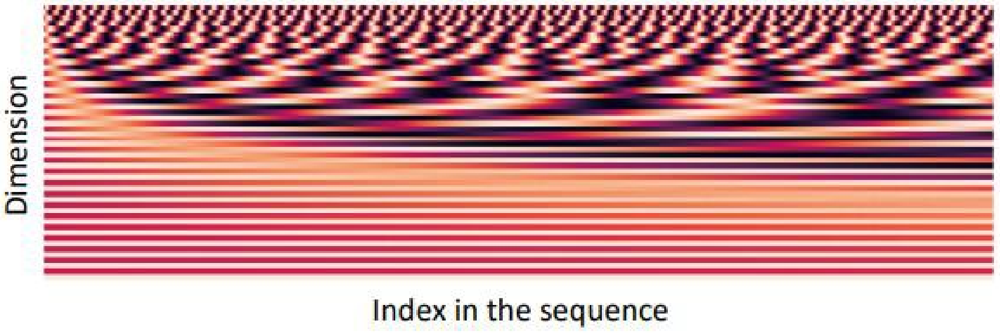
$\boldsymbol{i}$: Used for mapping to column indices

Positional encoding

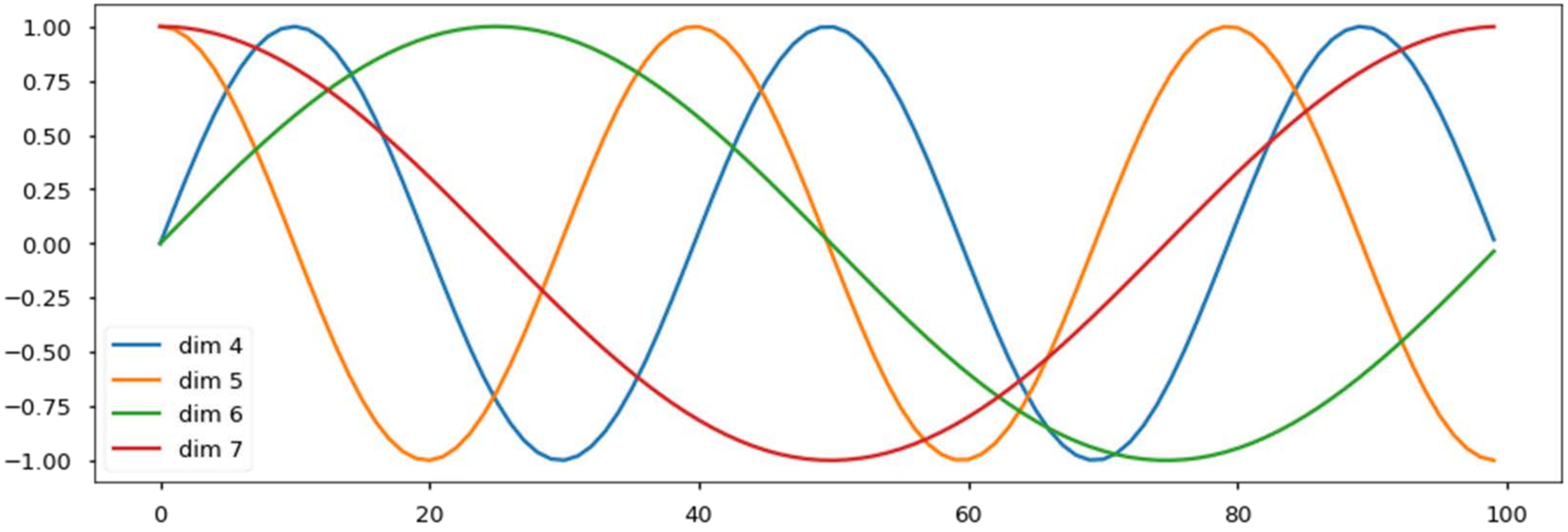
$$p_t = \begin{bmatrix} \sin(t/10000^{2*1/d}) \\ \cos(t/10000^{2*1/d}) \\ \sin(t/10000^{2*2/d}) \\ \cos(t/10000^{2*2/d}) \\ \dots \\ \sin(t/10000^{2*\frac{d}{2}/d}) \\ \cos(t/10000^{2*\frac{d}{2}/d}) \end{bmatrix}$$

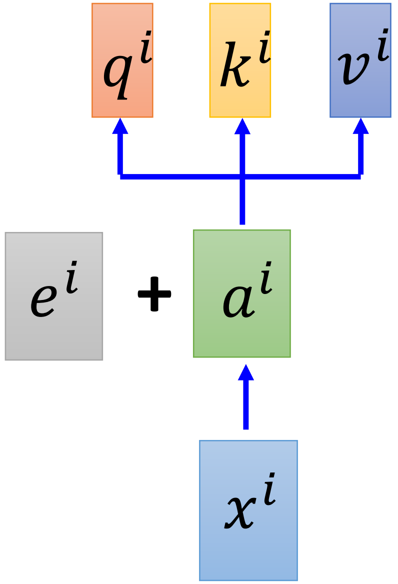
dimensionality of positional encoding

“even-odd” indicator

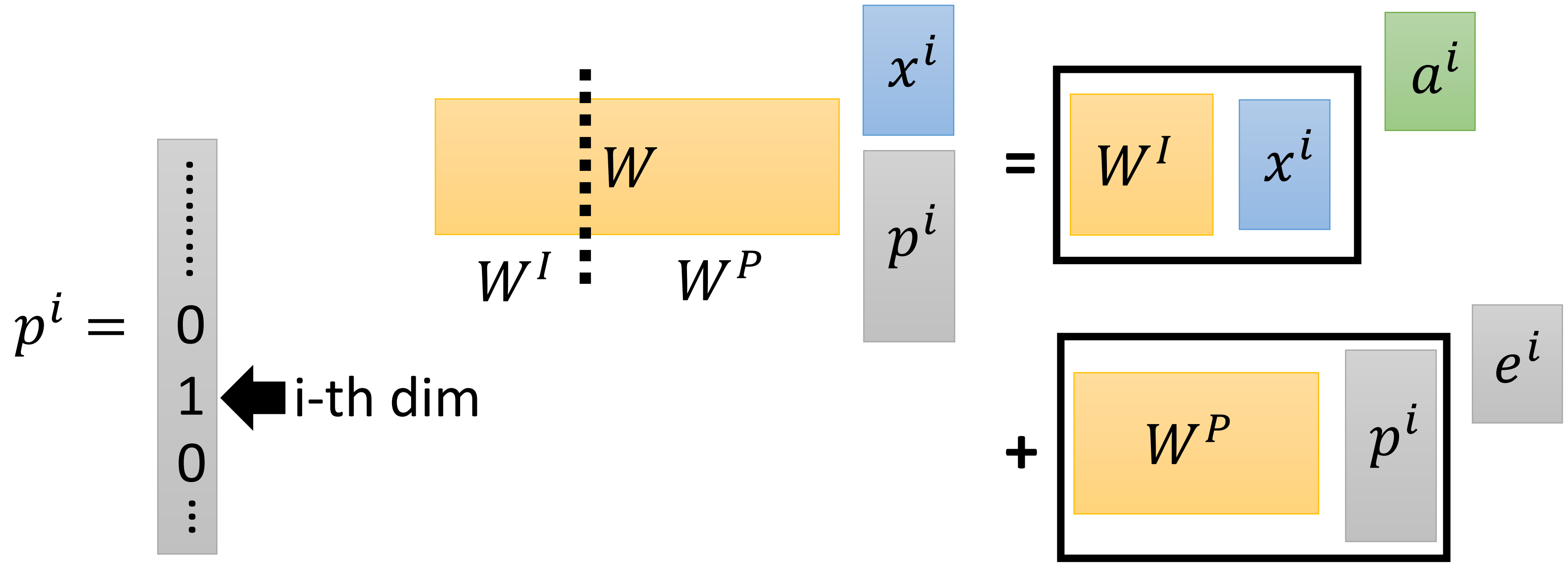


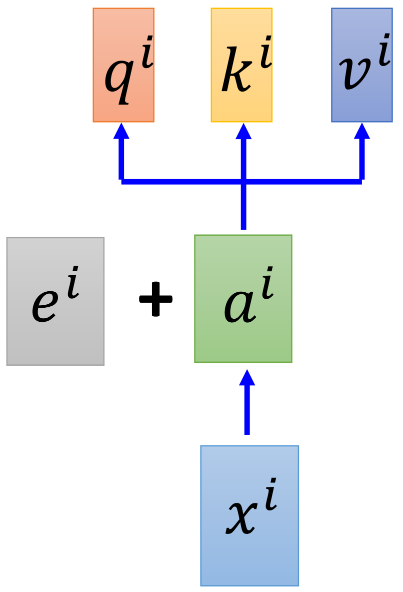
“first-half vs. second-half” indicator





Positional Encoding

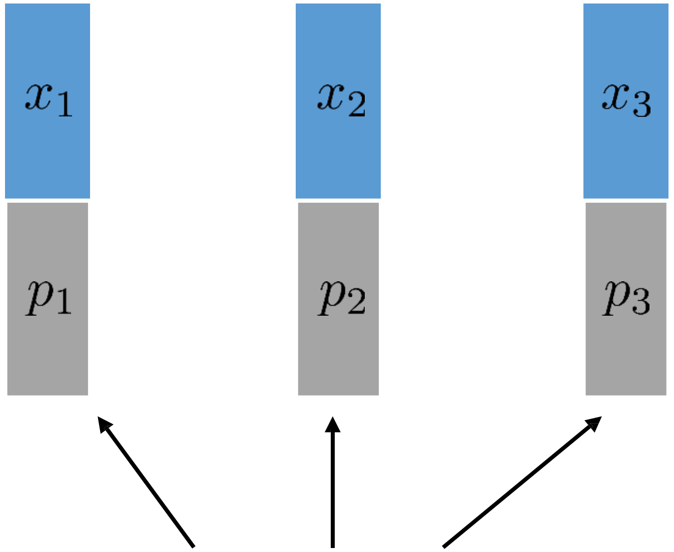




- Each position has a unique positional vector e^i (not learned from data)
- each x^i appends a one-hot vector p^i orr add them

Positional encoding: learned

Another idea: just learn a positional encoding



Different for every input sequence

The same learned values for every sequence

but different for different time steps

dimensionality

max sequence length

How many values do we need to learn?

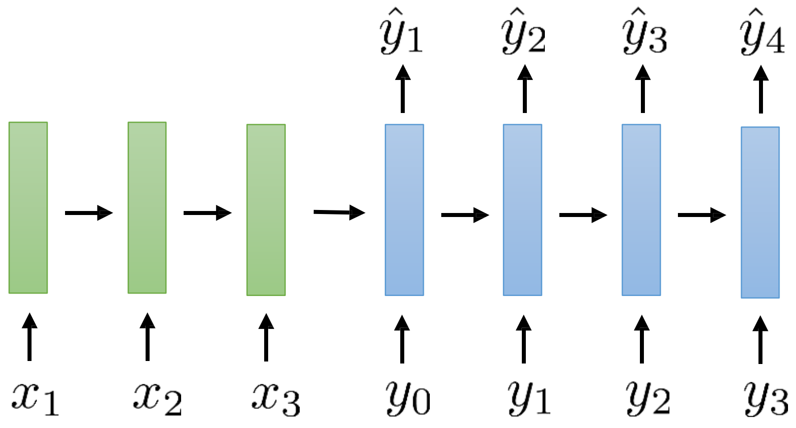
$$P = [p_1, p_2, \ldots, p_T] \in R^{d \times T}$$

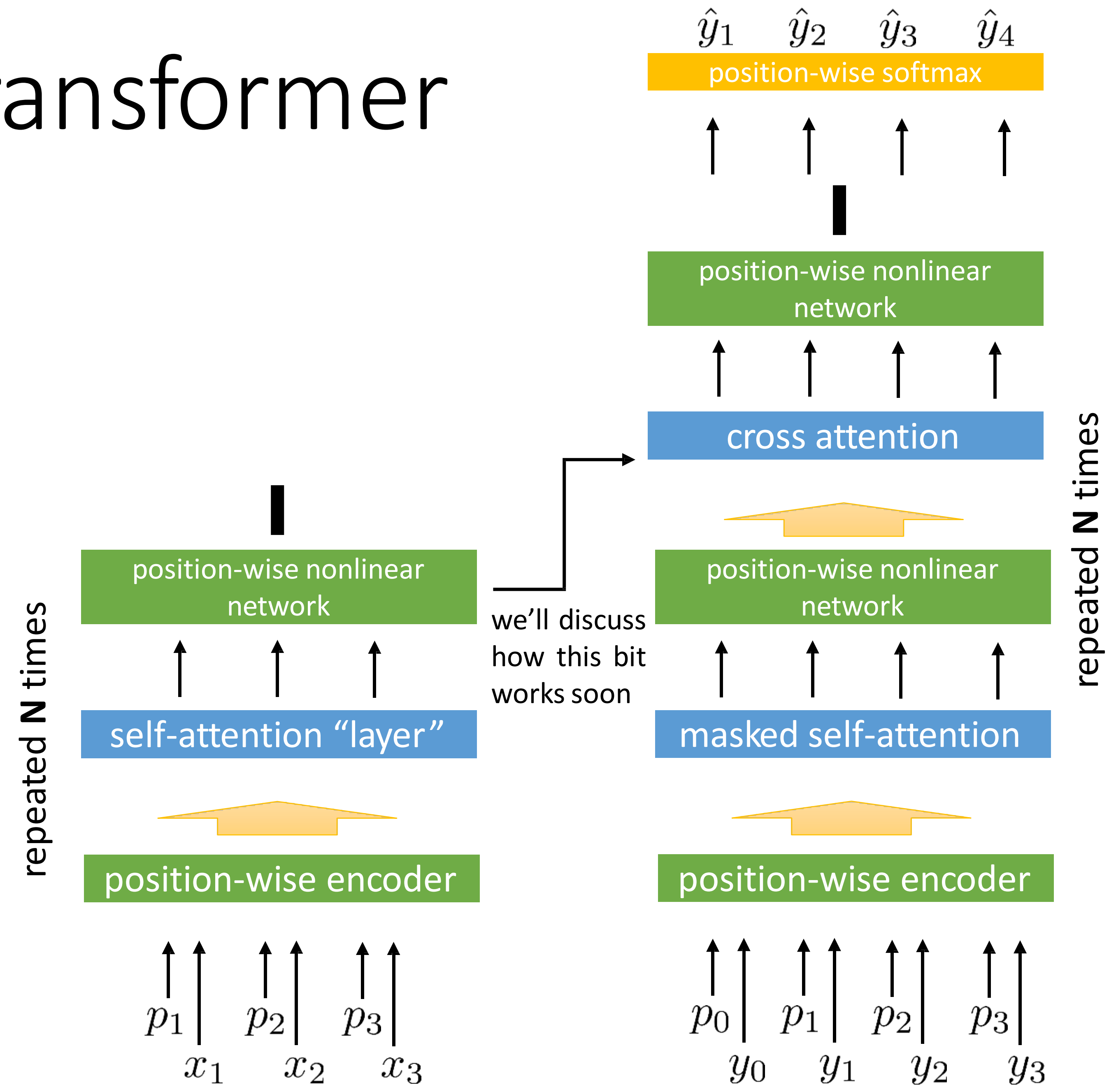
+ more flexible (and perhaps more optimal) than sin/cos encoding

+ a bit more complex, need to pick a max sequence length (and can't generalize beyond it)

The “classic” transformer

As compared to a sequence to sequence RNN model





The Final Linear and Softmax Layer

- A Softmax Layer to output word

- Let's assume that our model knows 10,000 unique English words (our model's "output vocabulary") that it's learned from its training dataset.

- The softmax layer then turns those scores into probabilities (all positive, all add up to 1.0). The cell with the highest probability is chosen, and the word associated with it is produced as the output for this time step.

Combining encoder and decoder values

“Cross-attention”

Much more like the **standard** attention from the previous lecture

$$\text{query: } q_l^\ell = W_q^\ell s_l^\ell$$

output of position-wise nonlinear network at (decoder) layer ℓ, step l

$$\text{key: } k_t^\ell = W_k^\ell h_t^\ell$$

output of position-wise nonlinear network at (encoder) layer ℓ, step t

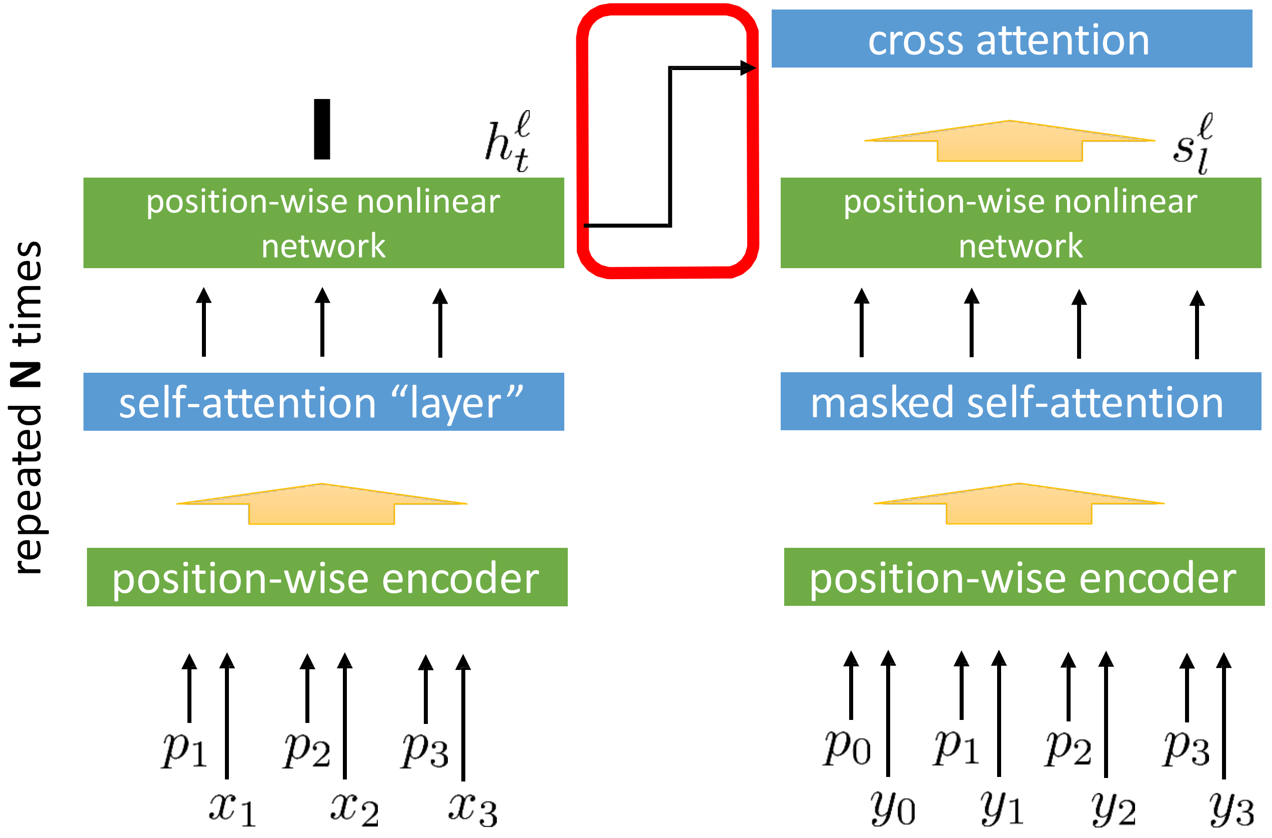
$$\text{value: } v_t^\ell = W_k^\ell h_t^\ell$$

$$e_{l,t}^\ell = q_l^\ell \cdot k_t^\ell$$

$$\alpha_{l,t}^\ell = \frac{\exp(e_{l,t}^\ell)}{\sum_{t'} \exp(e_{l,t'}^\ell)}$$

$$c_l^\ell = \sum_t \alpha_{l,t}^\ell v_t^\ell$$

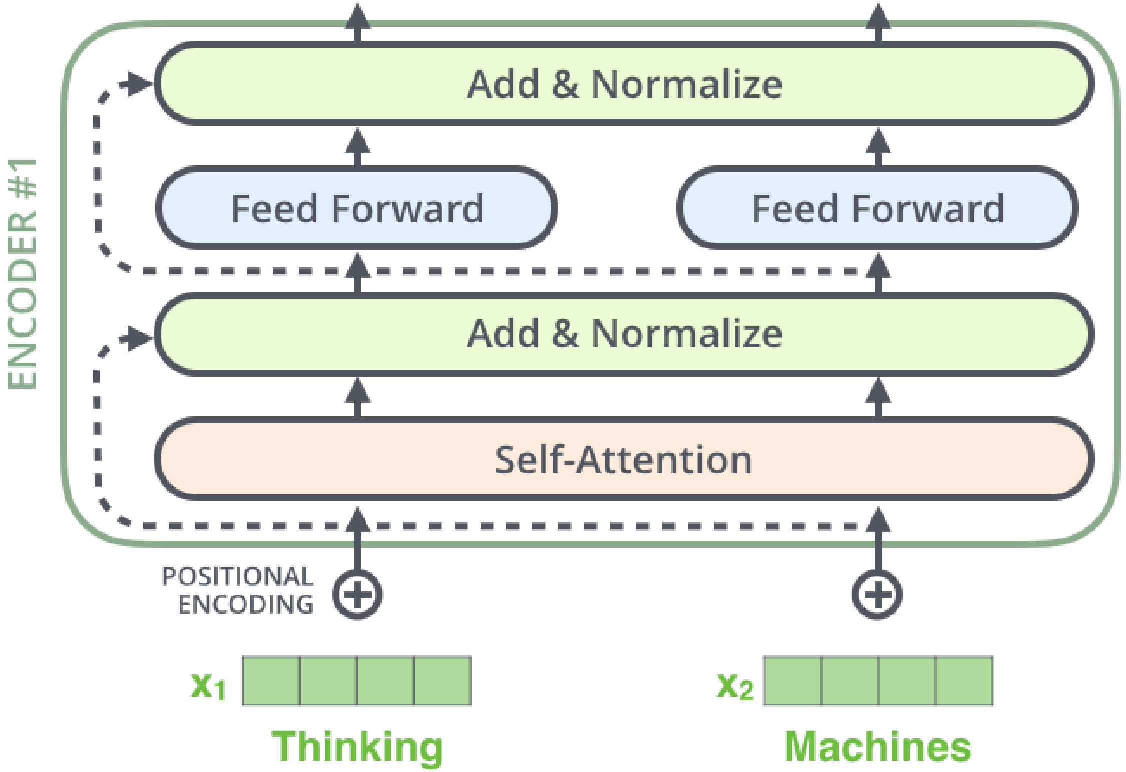
cross attention output

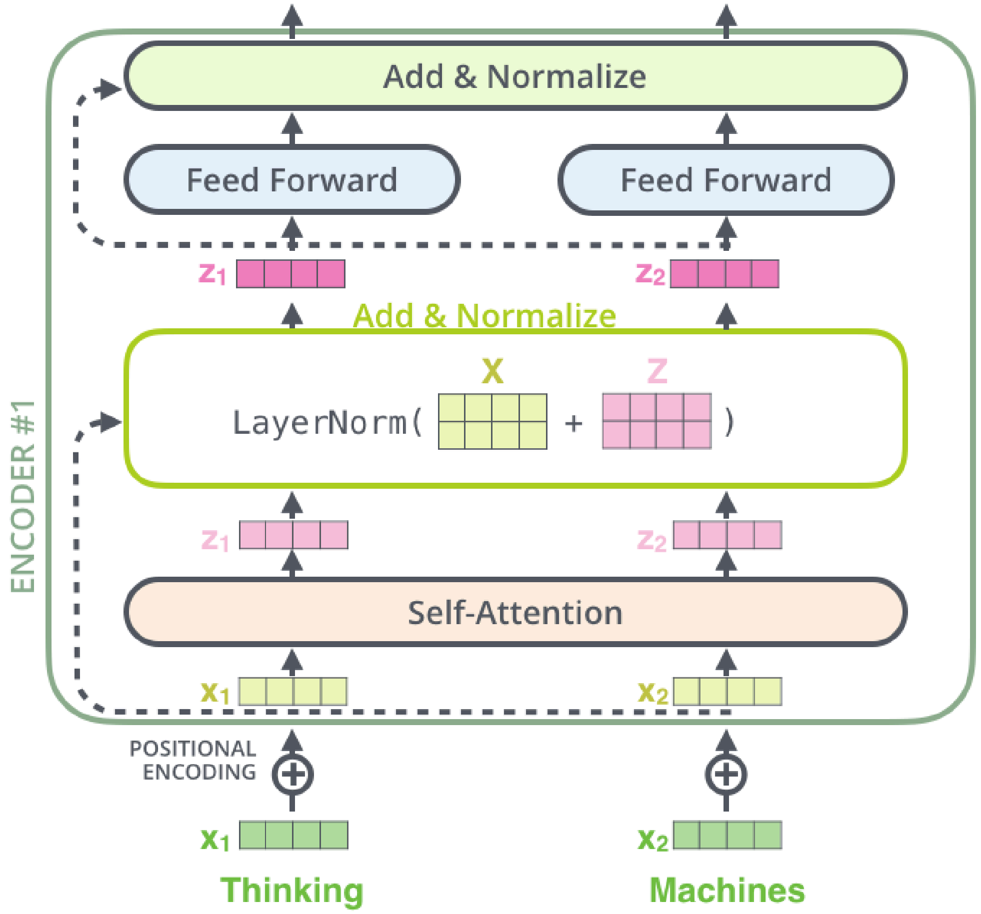


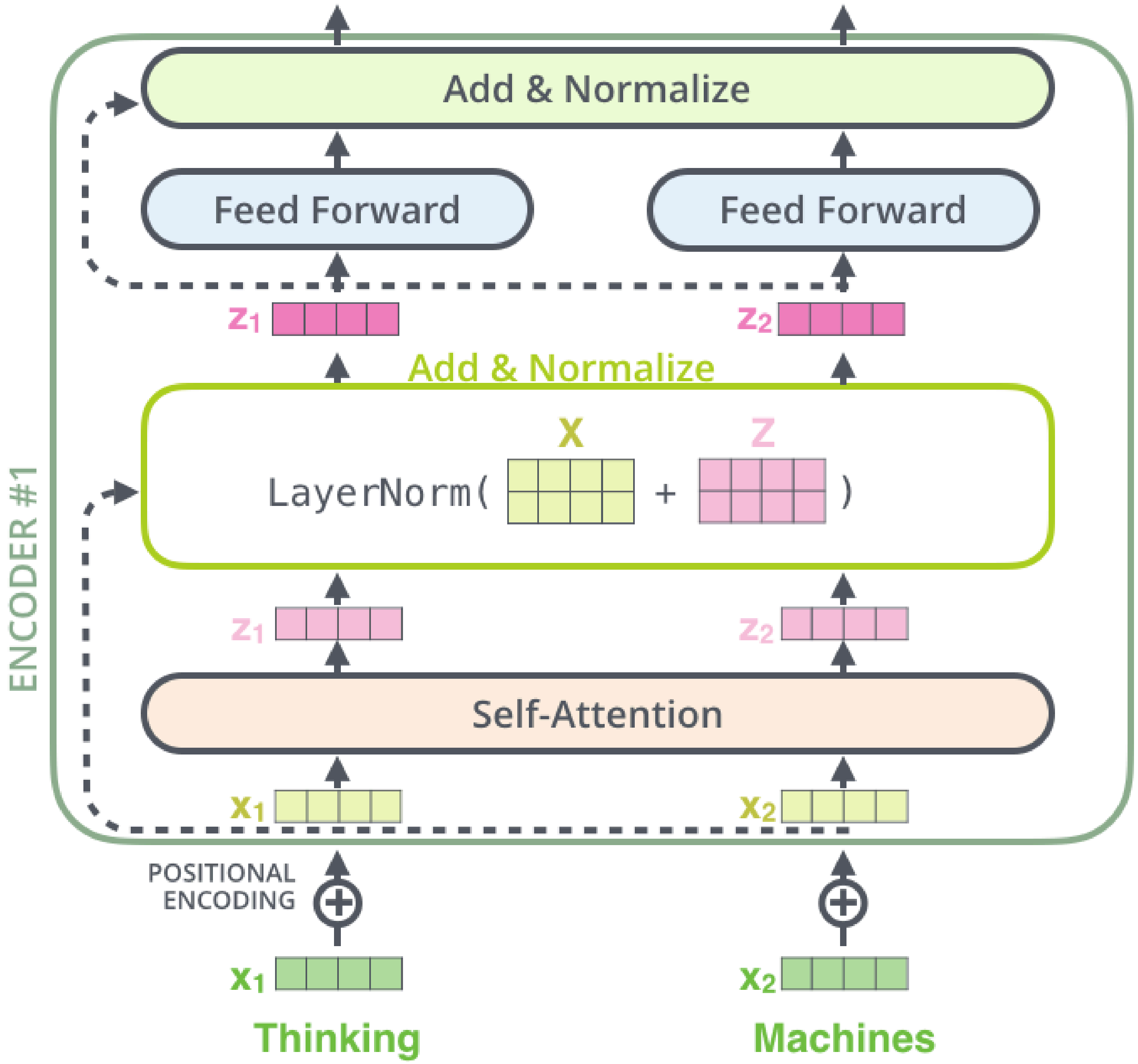
in reality, cross-attention is **also** multi-headed!

The Residuals

each sub-layer (self-attention, ffnn) in each encoder has a residual connection around it, and is followed by a layer-normalization step.







Add & Normalize
Feed Forward
Feed Forward
z1
z2
Add & Normalize
X
Z
LayerNorm(+)
z1
z2
Self-Attention
x1
x2
ENCODER #1
POSITIONAL ENCODING
x1
x2
Thinking
Machines

Layer normalization

Main idea: batch normalization is very helpful, but hard to use with sequence models Sequences are different lengths, makes normalizing across the batch hard Sequences can be very long, so we sometimes have small batches

Simple solution: “layer normalization” – like batch norm, but not across the batch

Batch norm

d-dimensional vectors for each sample in batch

$$a_1, a_2, \ldots, a_B$$

d-dim

$$\mu = \frac{1}{B}\sum_{i=1}^{B} a_i \qquad \sigma = \sqrt{\frac{1}{B}\sum_{i=1}^{B}(a_i - \mu)^2}$$

$$\bar{a}_i = \frac{a_i - \mu}{\sigma}\gamma + \beta$$

Layer norm

Different dimensions of a

$$a$$

$$\boxed{\mu = \frac{1}{d}\sum_{i=1}^{d} a_j} \qquad \boxed{\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(a_j - \mu)^2}}$$

1-dim

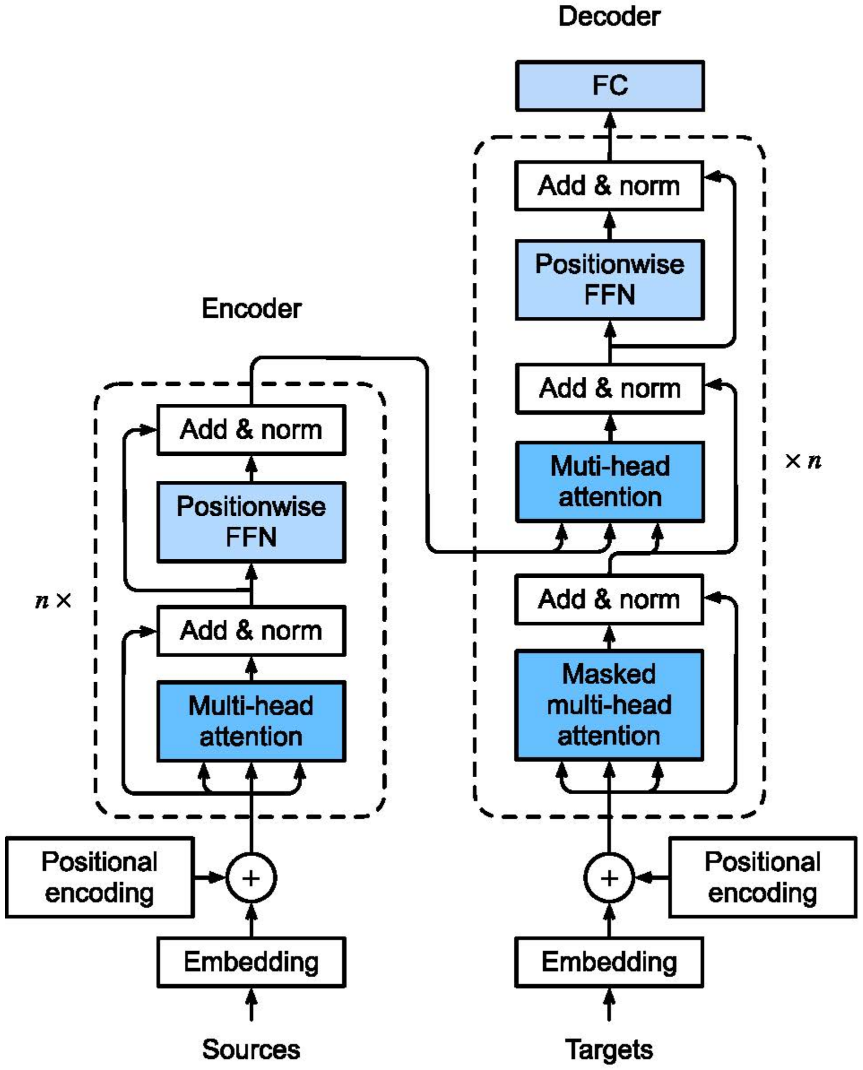
$$\boxed{\bar{a} = \frac{a - \mu}{\sigma}\gamma + \beta}$$

The Transformer Architecture

Composed of an encoder and a decoder.

- Encoder:
  - a stack of multiple identical blocks
  - each block has two sublayers

1. a multi-head self-attention pooling (queries, keys, and values are all from the outputs of the previous encoder layer)
2. a positionwise feed-forward network
3. A residual connection is employed around both sublayers followed by layer normalization



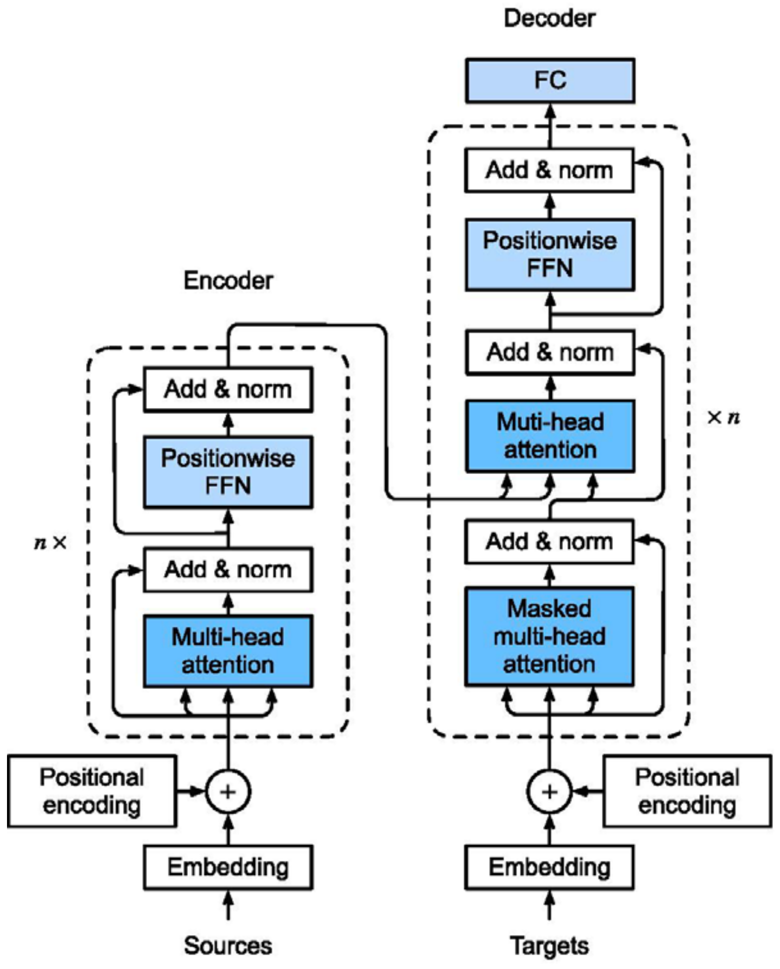
The Transformer Architecture

Decoder:

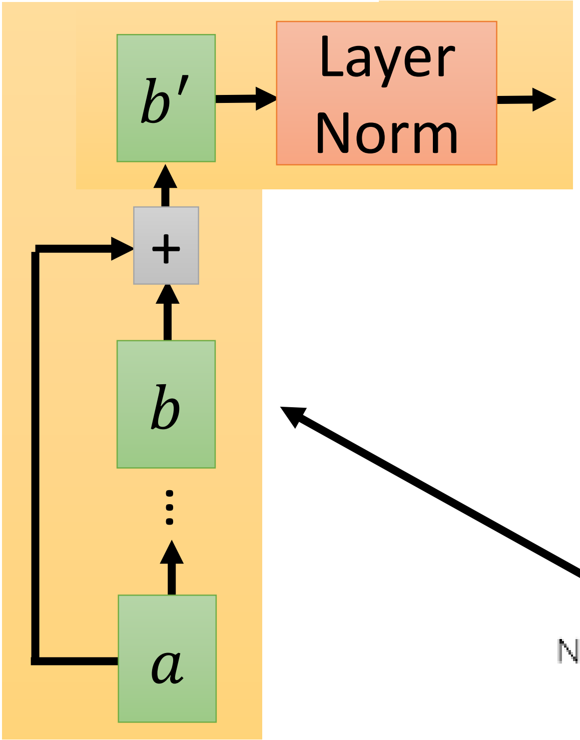
- a stack of multiple identical blocks
- each block has three sublayers.

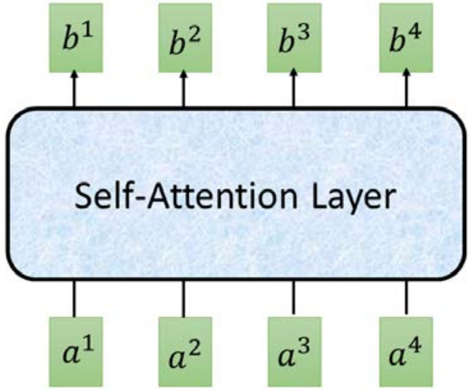
1. a multi-head self-attention pooling -- each position in the decoder is allowed to only attend to all positions in the decoder up to that position
2. Encoder-decoder attention: queries are from the outputs of the previous decoder layer, and the keys and values are from the Transformer encoder outputs
3. A positionwise feed-forward network

- A residual connection is employed around both sublayers followed by layer normalization



Transformer



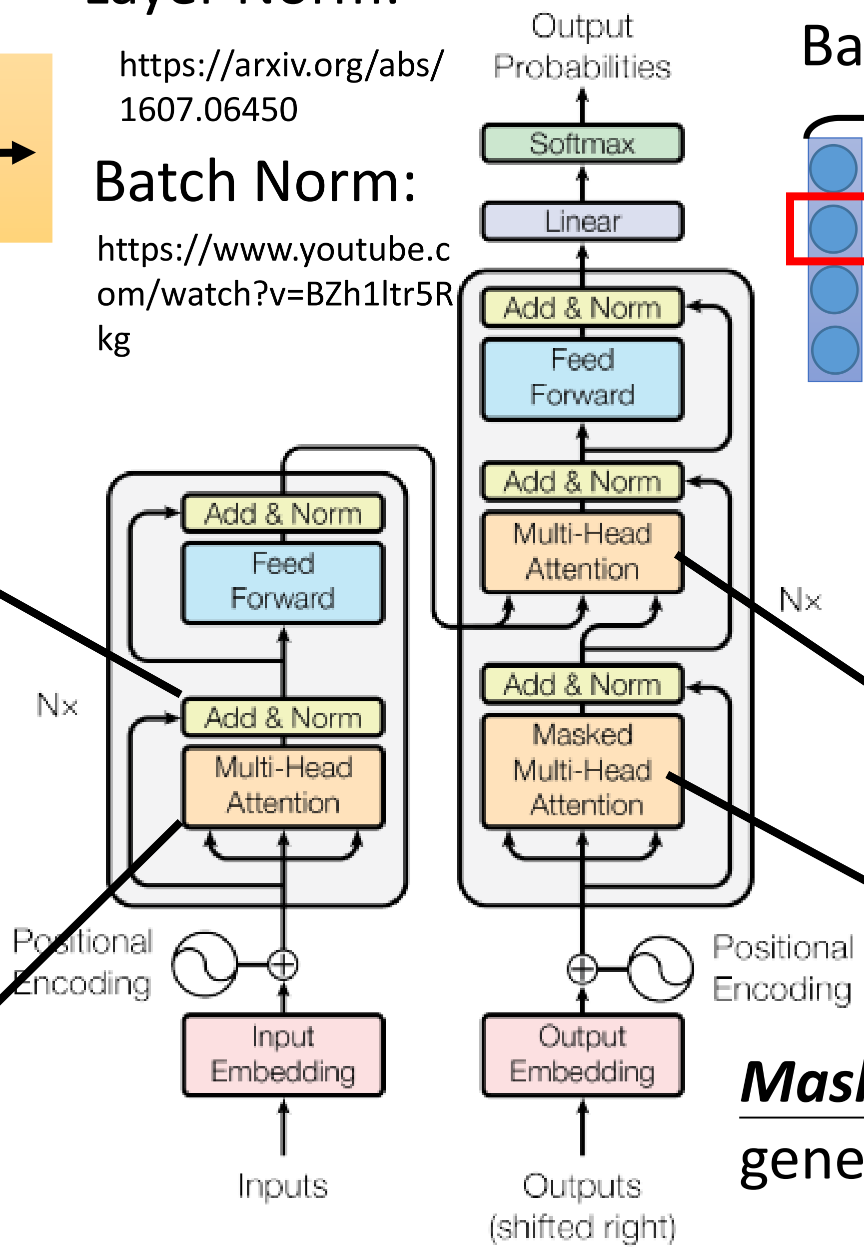


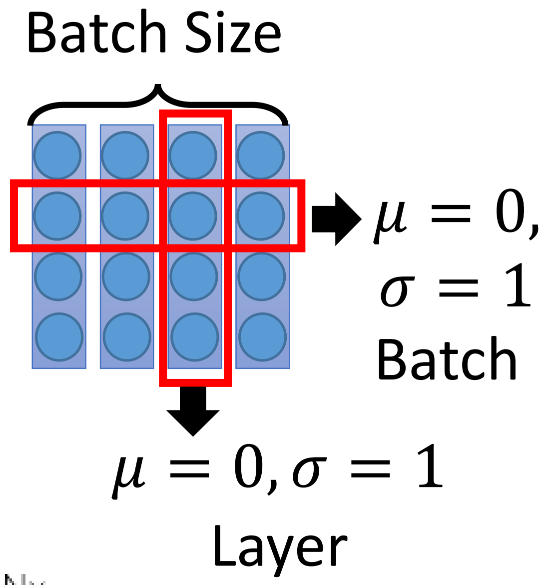
Layer Norm:

https://arxiv.org/abs/1607.06450

Batch Norm:

https://www.youtube.com/watch?v=BZh1ltr5Rkg





attend on the input sequence

Masked: attend on the generated sequence

Putting it all together

Decoder decodes one position at a time with masked attention

The Transformer

multi-head attention keys and values $k_{t,1}^{\ell}, \ldots, k_{t,m}^{\ell}$ and $v_{t,1}^{\ell}, \ldots, v_{t,m}^{\ell}$

6 layers, each with d = 512

$\bar{h}_t^{\ell} = \mathrm{LayerNorm}(\bar{a}_t^{\ell} + h_t^{\ell})$ passed to next layer $\ell + 1$

$h_t^{\ell} = W_2^{\ell}\mathrm{ReLU}(W_1^{\ell}\bar{a}_t^{\ell} + b_1^{\ell}) + b_2^{\ell}$

2-layer neural net at each position

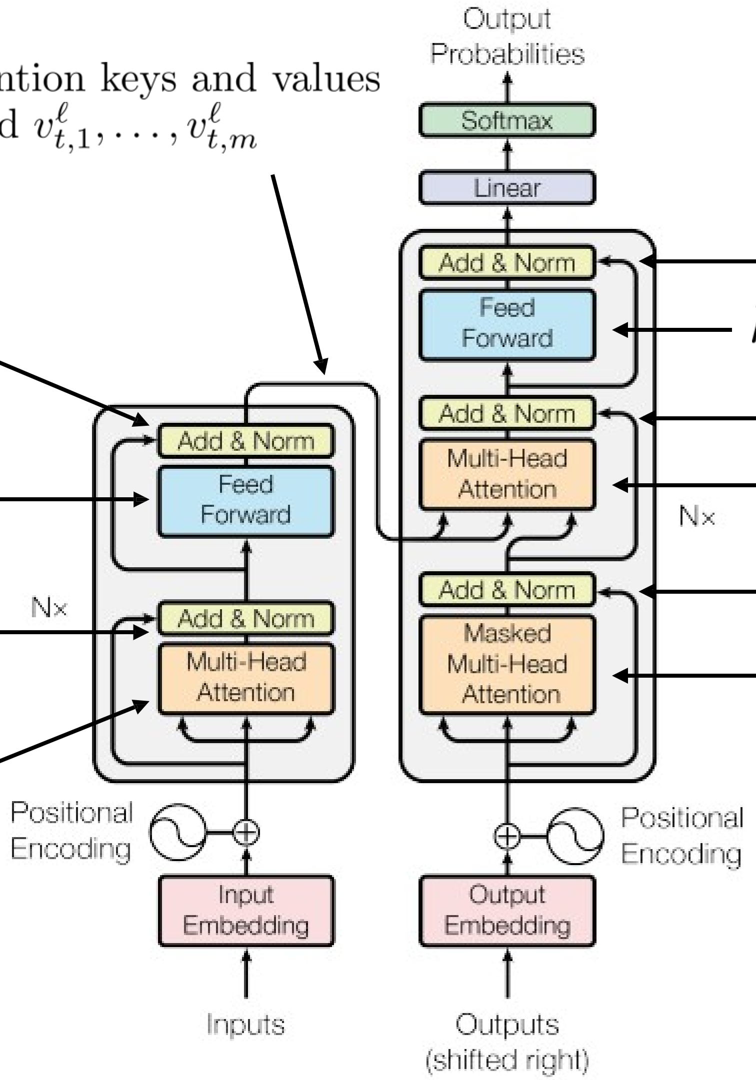
$\bar{a}_t^{\ell} = \mathrm{LayerNorm}(\bar{h}_t^{\ell-1} + a_t^{\ell})$

essentially a residual connection with LN

input: $\bar{h}_t^{\ell-1}$

output: a_t^{ℓ}

concatenates attention from all heads



residual connection with LN

$h_t^{\ell} = W_2^{\ell}\mathrm{ReLU}(W_1^{\ell}\bar{a}_t^{\ell} + b_1^{\ell}) + b_2^{\ell}$

residual connection with LN

multi-head cross attention

residual connection with LN

same as encoder only **masked**

Vaswani et al. **Attention Is All You Need.** 2017.

Why transformers?

Downsides:

- **Attention computations are technically $O(n^2)$**
- **Somewhat more complex to implement (positional encodings, etc.)**

Benefits:

+ **Much better long-range connections**
+ **Much easier to parallelize**
+ **In practice, can make it much deeper (more layers) than RNN**

The benefits seem to **vastly** outweigh the downsides, and transformers work **much** better than RNNs (and LSTMs) in many cases

Arguably one of the most important sequence modeling improvements of the past decade