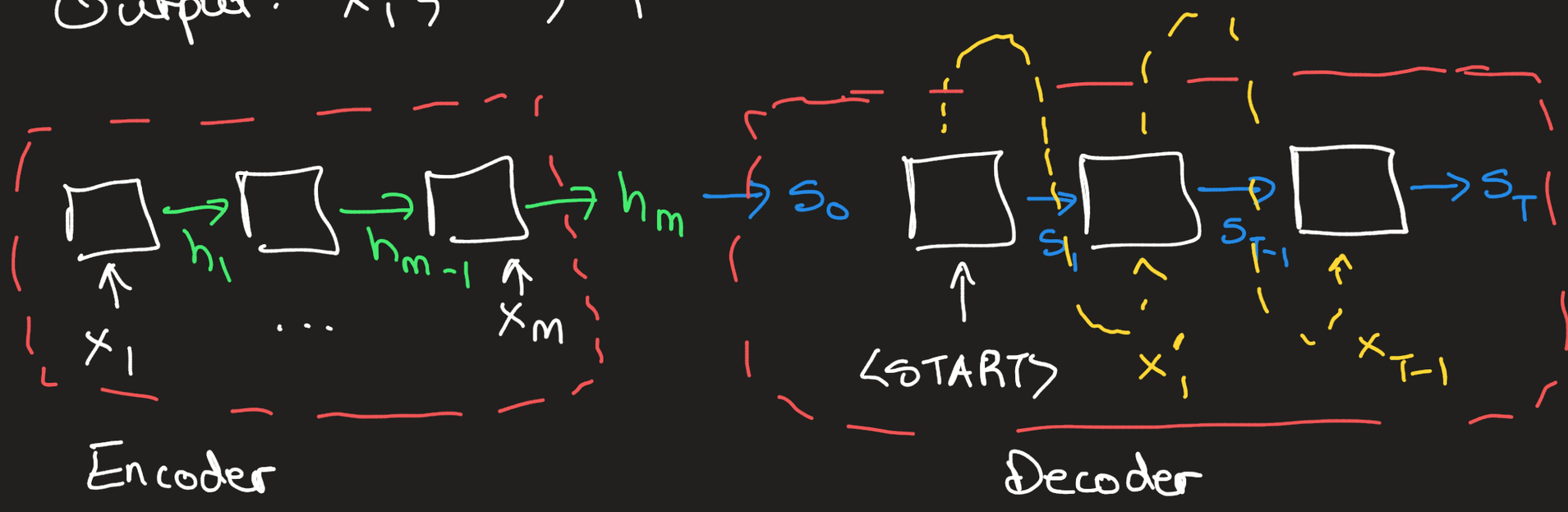
ML and Optimization Lec 23

- Seq2Seq model w/ attention
- Attention is all we need: seq2seq models w/o RNNs
- BERT: pretraining transformer models for NLP

Seq2Seq model

Input: $x_1, \cdots, x_m$

Output: $x'_1, \cdots, x_T$



Issue: regardless of length of input sequence, all context/historical dependence of the output sequence is contained in a d-dimensional vector h_m

Leads to poor performance (BLEU decreases w/ length of input sequence for Machine translation tasks)

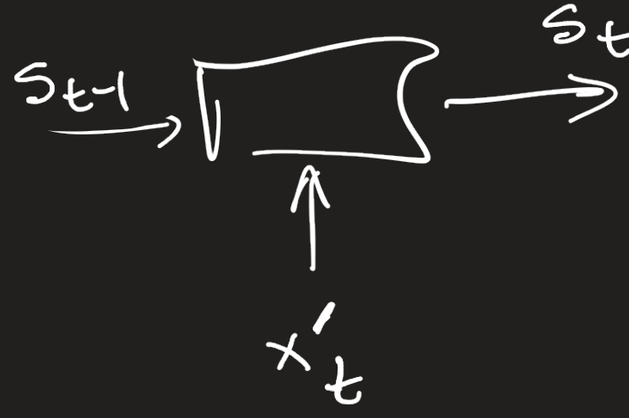
Seq2Seq with attention

- Introduced in Bahdanau et al. 2015, Neural Machine Translation by Jointly Learning to Align and Translate

- Allow each output token to attend to each individual hidden state in the input sequence. Captures idea of focusing on relevant portion of input sequence. Avoids the need to capture everything relevant to all outputs $x'_1, \ldots, x'_T$ in the one vector h_m

Seq2Seq w/ attention

Idea: replace our previous hidden state update for decoder



i.e. $s_t = g(s_{t-1}, x'_t)$

w/ first finding which historical hidden states are most relevant:

$$s_{t-1}, x'_t, [h_1, \cdots, h_m] \longmapsto \alpha_t \text{ a prob. dist over } [h_1, \cdots, h_m]$$

then form a context vector

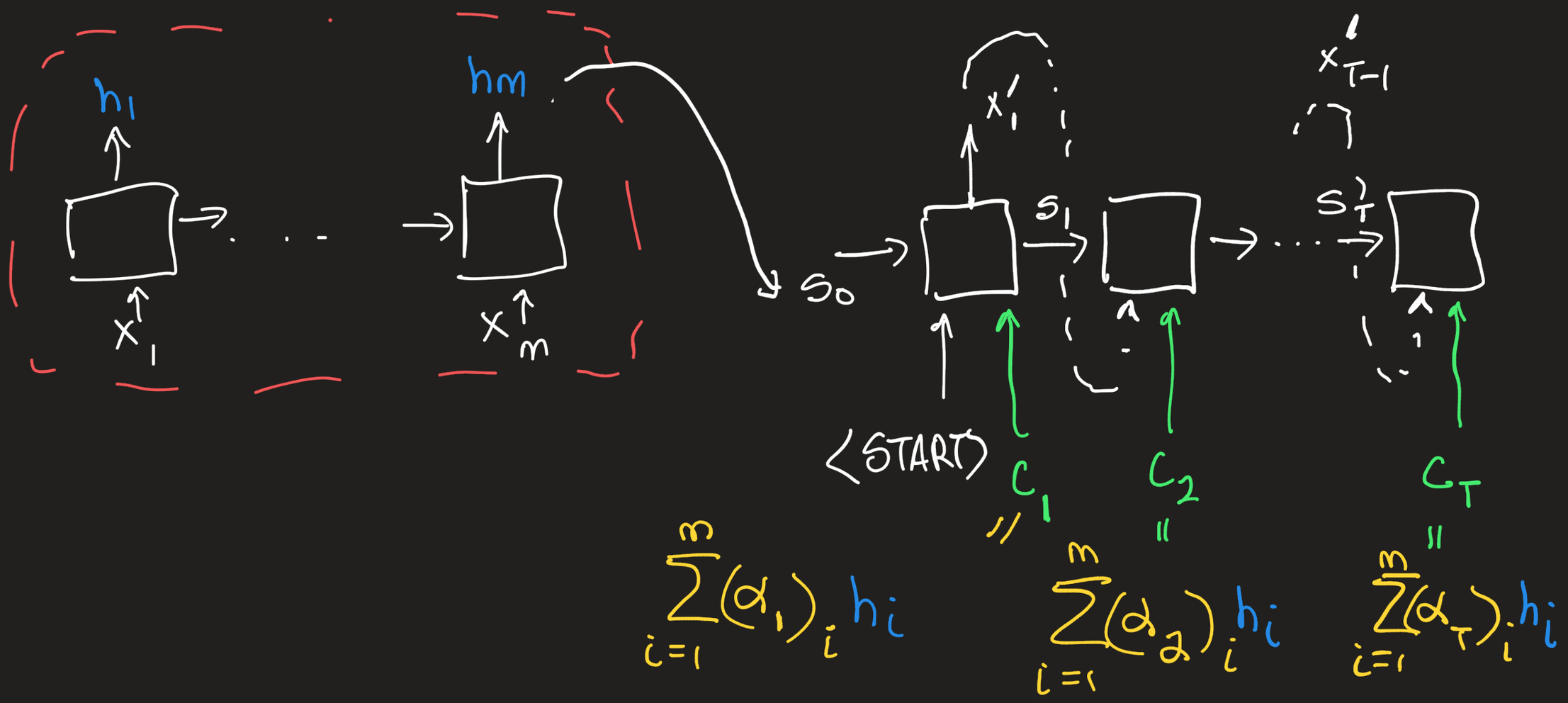
$$c_t = \sum_{i=1}^{m} (\alpha_t)_i h_i$$

and update with

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

we removed dependence on x'_t because in the decoder x'_t isn't much more than s_{t-1}

In the context of decoding, in the decoder, use attention over the hidden states of the encoder



Options for computing the attention vectors α_t

1) Option I (used in Bahdanau et al. 2015)

Take $\tilde{\alpha}_i = \underline{v^T} \tanh\left(\underline{W} \begin{bmatrix} h_i \\ s_{t-1} \end{bmatrix}\right)$ for $i = 1, \ldots, m$

parameters to learn

Then $\alpha_t = \text{softmax}(\tilde{\alpha}) \in \mathbb{R}^m$

2) (Dot-product attention) Idea: generate keys and queries as linear transforms of h_i and s_{t-1} respectively, take logits $\tilde{\alpha}_i$ to be their inner-products

$W_K h_i = k_i$

$W_Q s_{t-1} = q_{t-1}$

$\tilde{\alpha}_i = \langle k_i, q_{t-1} \rangle$

$\alpha_t = \text{softmax}(\tilde{\alpha})$

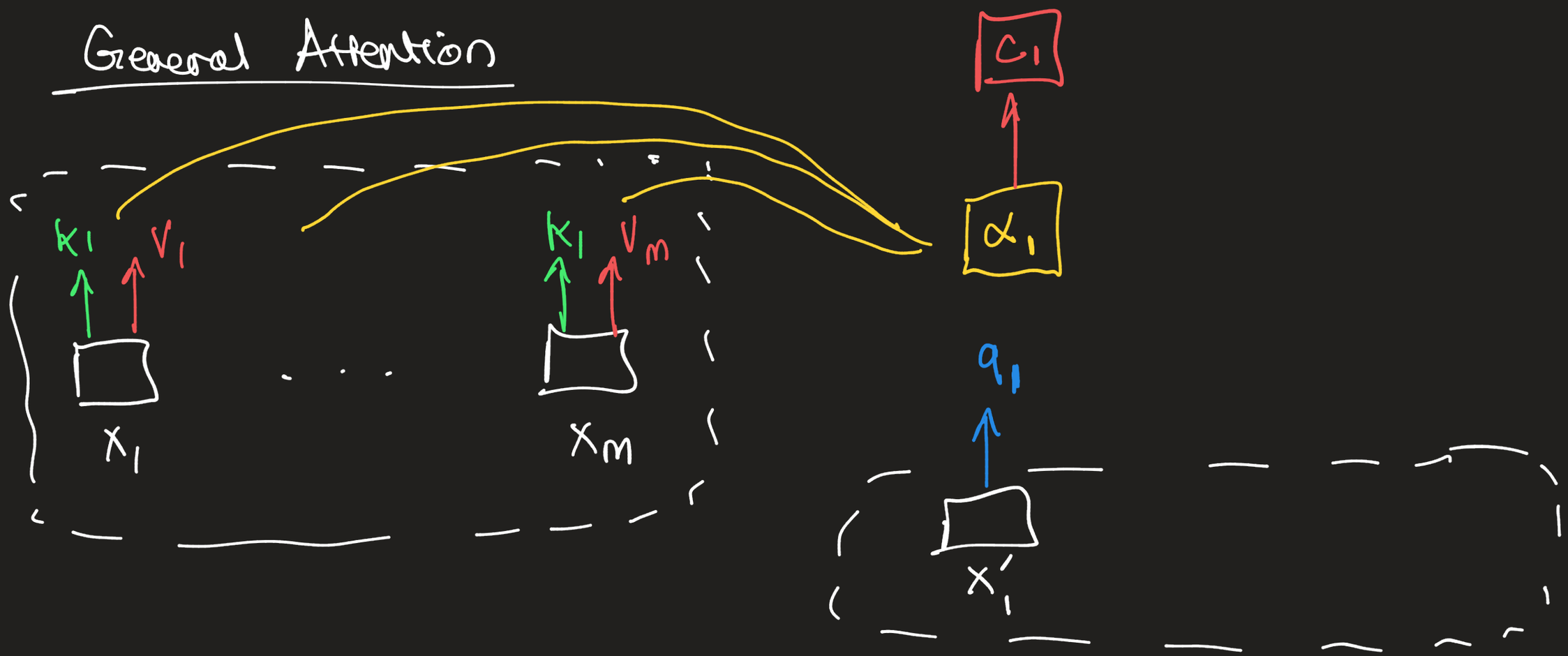
Complexity of using attention

1) Simple RNN encoder models : $O(n + T)$

2) RNN + attention models $O(n \cdot T)$

because for each output token, must compute $\alpha_t \in \mathbb{R}^n$ by comparing to all inputs

General Attention



Self-Attention (replace the hidden states learned in RNNs w/ contextual representations)

$$c_i = \sum_{j=1}^{m} (\alpha_i)_j \, v_j$$

where key $k_i = W_K x_i$

query $q_i = W_Q x_i$

value $v_i = W_V x_i$

and $(\tilde{\alpha}_i)_j = \langle q_i, k_j \rangle$ and $\alpha = \text{softmax}(\tilde{\alpha})$

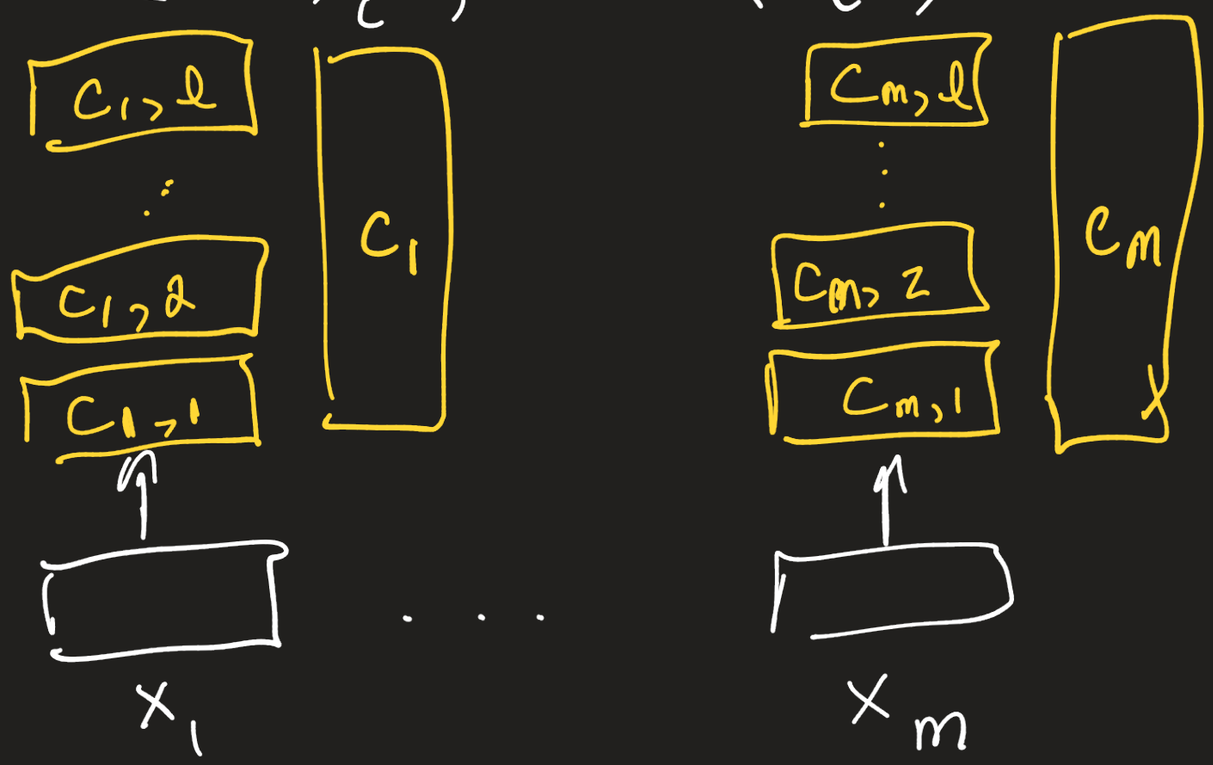
Transformers : Attention is all we need

- 2017, Vaswani et al., Attention is all we need
- Insight: attention alone suffices to contain all historical information we need, so can eliminate recursive structure.
  - Replace hidden representations in RNNs w/ contextual representations c_t computed using self-attention (for encoder)
  - Use combination of self-attention in decoder and general attention against the encoder contextual representations to get contextual representations to predict output tokens

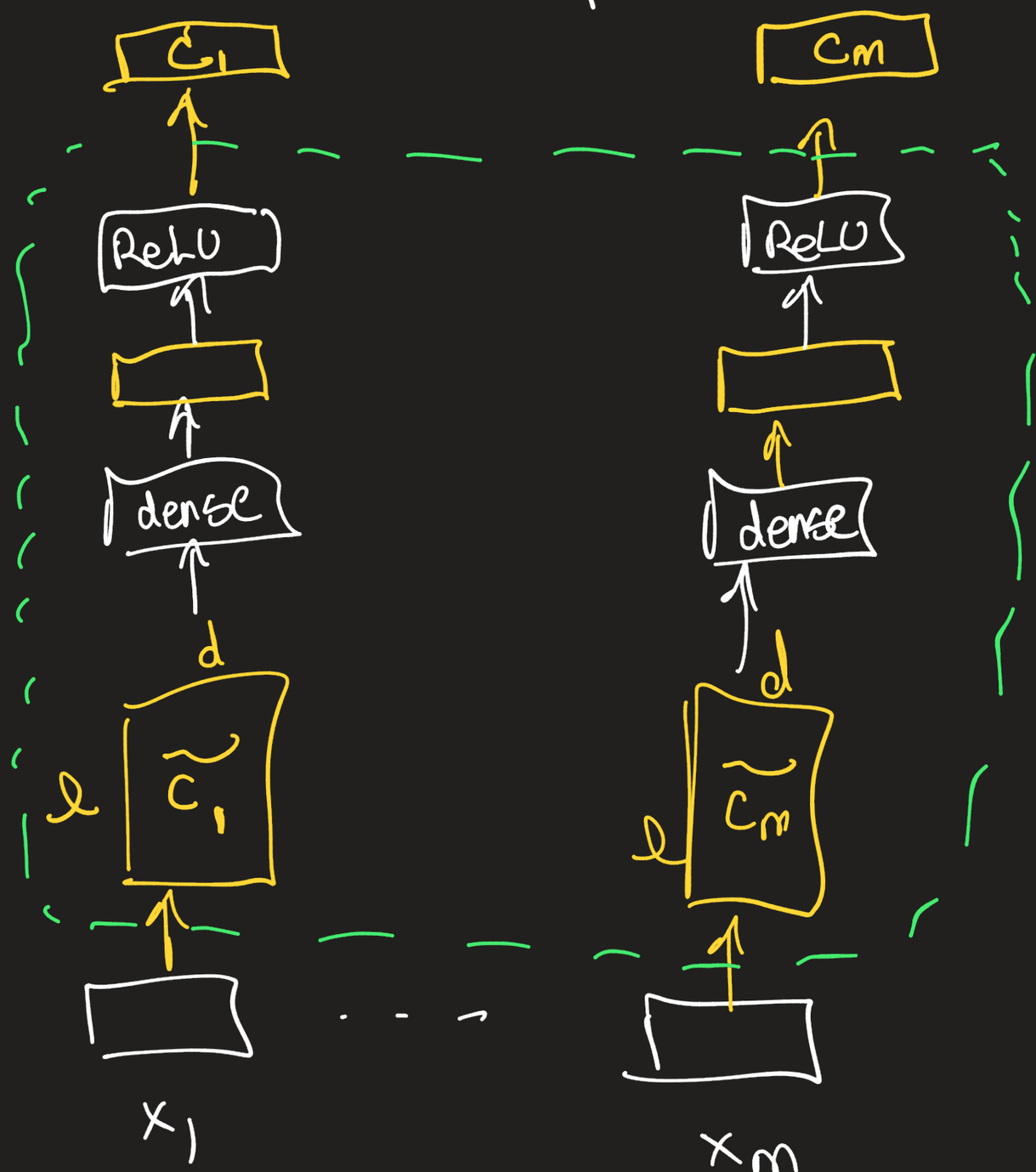
Multi-Head Self Attention

Just as with CNNs we benefit from learning multiple contextual representations:

$$(W_K)_i, (W_Q)_i, (W_V)_i \quad \text{for } i = 1, \dots, l$$



Contextual representations using multi-head self-attention

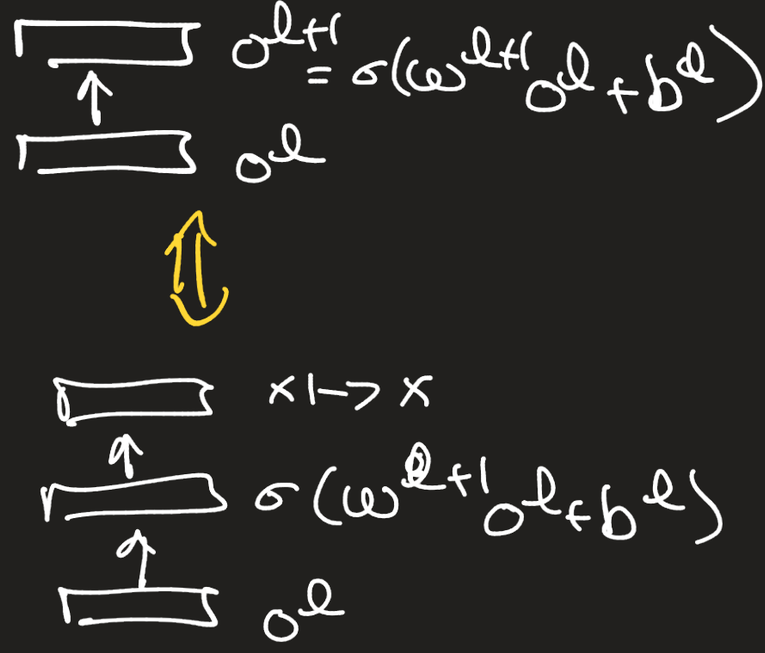


Self-Attention + Dense Layer

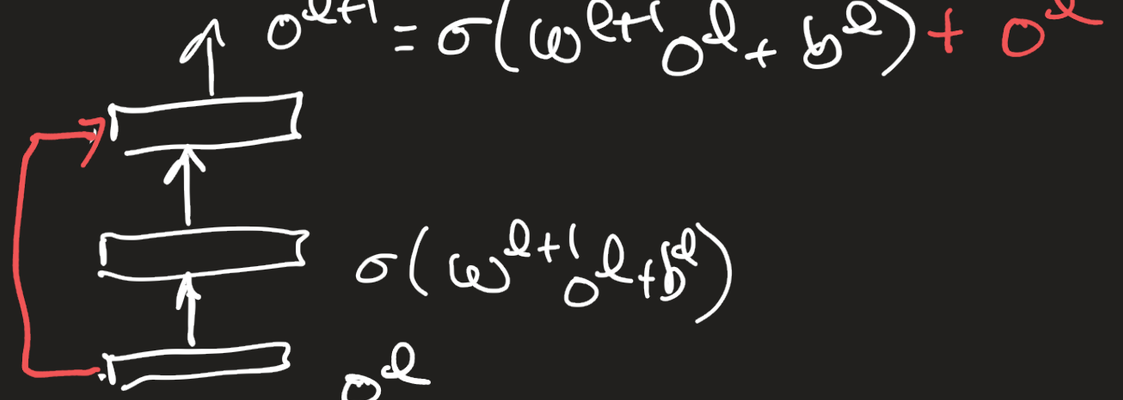
Skip-connections

Skip-connections are connections between layers l and layers higher than l+1, e.g. DenseNet, where all layers are connected.

An efficient & popular type of skip-connection is a residual connection,



Residual connection



Layer-normalization

Like batch normalization this helps stabilize training by ensuring the distribution of the outputs of layer l doesn't vary much over time, so layer $l+1$ doesn't have to waste optimization adapting.

$$\text{LayerNorm}(o) = \gamma \cdot \frac{o - \bar{o}}{\sigma} + \beta \,,$$

where $\gamma \in \mathbb{R}^{n_l}$ and $\beta \in \mathbb{R}^{n_l}$ are the parameters of the layer normalization layers and

$$\bar{o} = \frac{1}{n_l} \sum_{i=1}^{n_l} o_i \,, \qquad \sigma^2 = \frac{1}{n_l} \sum_{i=1}^{n_l} (o_i - \bar{o})^2$$

Batch normalization

operates on minibatches

↓ Batch normalization computes per neuron means and stdevs over the minibatch

then batch normalization of an o_i is given by

$$\gamma \cdot \left(\frac{o_i - \mu}{\sigma}\right) + \beta$$

Layer normalization

operates on a single output

$\bar{o}$ = mean of o

σ = stdv of o

then layer normalization of o is given by

$$\gamma \cdot \left(\frac{o - \bar{o}}{\sigma}\right) + \beta$$

Expressions for Attention

Self-attention:

then $\tilde{\alpha}_i^T = \begin{bmatrix} q_i^T k_1 & \cdots & q_i^T k_m \end{bmatrix}$

is the logits vector (row vector) for the i^{th} attention vector

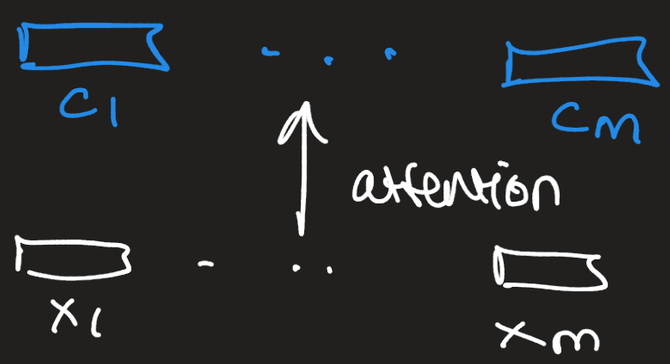
$\alpha_i^T = \text{softmax}(\tilde{\alpha}_i^T)$

is my attention vector for the i^{th} token is

then our contextual embedding

$$c_i^T = \sum_{j=1}^{m} (\alpha_i)_j \, v_j^T$$

let $X = \begin{bmatrix} x_1^T \\ \vdots \\ x_m^T \end{bmatrix} \in \mathbb{R}^{m \times d}$



and $C = \begin{bmatrix} c_1^T \\ \vdots \\ c_m^T \end{bmatrix} \in \mathbb{R}^{m \times d_v}$

Then

$$C = \mathrm{Aff}(X, X)$$

where the attention layer has parameters W_Q, W_K, W_V

To compute C, write

$$K = XW_K = \begin{bmatrix} x_1^T W_K \\ \vdots \\ x_m^T W_K \end{bmatrix}$$

$$Q = XW_Q = \begin{bmatrix} x_1^T W_Q \\ \vdots \\ x_m^T W_Q \end{bmatrix}$$

d_k-dim

$$V = XW_V = \begin{bmatrix} x_1^T W_V \\ \vdots \\ x_m^T W_V \end{bmatrix}$$

, then note that

$$C = \begin{bmatrix} c_1^T \\ \vdots \\ c_m^T \end{bmatrix} = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

row-wise

Similarly, a general attention layer takes two sequences $X \in \mathbb{R}^{m \times d}$ $X' \in \mathbb{R}^{T \times d}$

so a general attention layer computes

$$C = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where

$$K = X W_K$$

$$V = X W_V$$

$$Q = X' W_Q$$

Transformer Architecture (image upcoming)

Encoder:

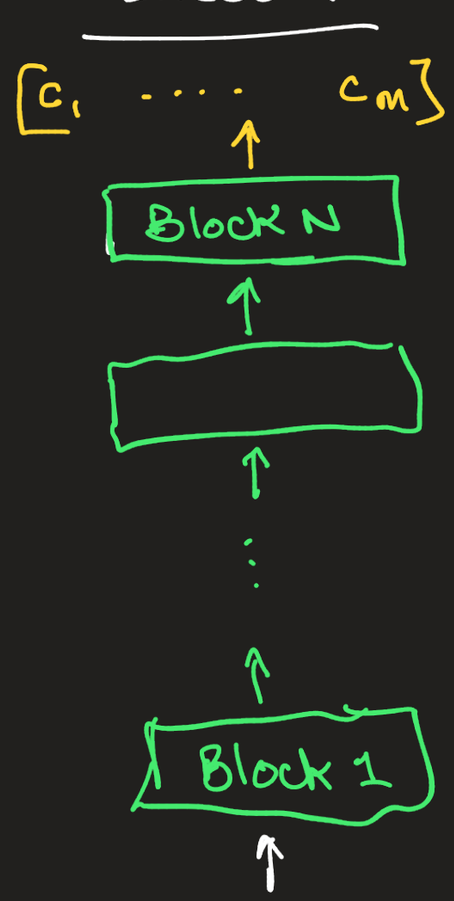
- use multiple layers of multi-head self-attention with residual connections > layer-normalization, and dense nonlinear transforms

Decoder

- use multiple layers of multi-head self-attention, multi-head general attention, with residual connections, layer normalizations, and dense nonlinear transforms.

NB: we use layer normalization in Transformer blocks for the same reasons as in RNNs: variable sequence lengths and potentially small batch sizes

Encoder:



where each embedding block has its own parameters and computes:

Given input $X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \in \mathbb{R}^{m \times d}$,

i) $\Psi = \text{LayerNorm}(X + \text{Mult-head Att}(X, X))$

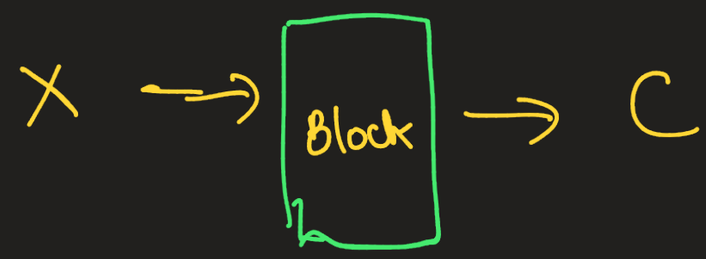
recall $\text{Mult-head Att}(X, X) =$

$[\text{Att}_1(X, X) \mid \cdots \mid \text{Att}_h(X, X)] W_0$

concatenates h different attention outputs together then takes linear combination to learn a final d-dim representation

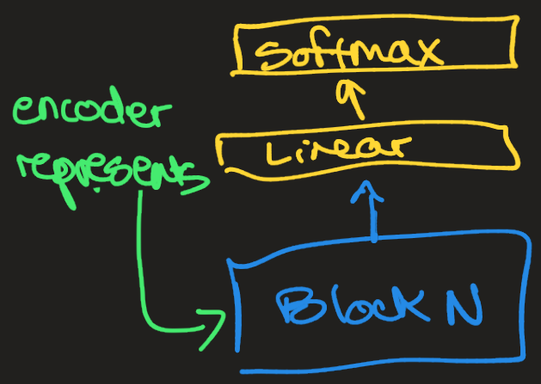
ii) $C = \text{LayerNorm}(\Psi + \text{FF}(\Psi))$

where FF applies the same feedforward dense + ReLU layer to each row of Ψ



Decoder

During training, use teacher forcing



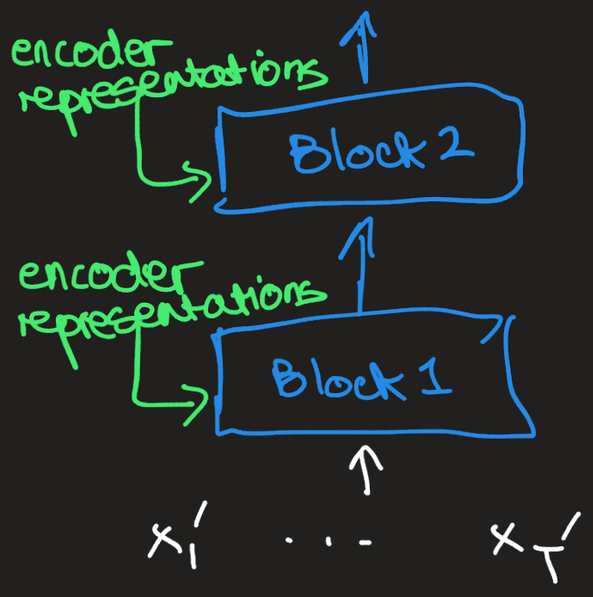
each block has its own parameters and computes, given inputs $X' = \begin{bmatrix} x_1' \\ \vdots \\ x_T' \end{bmatrix}$

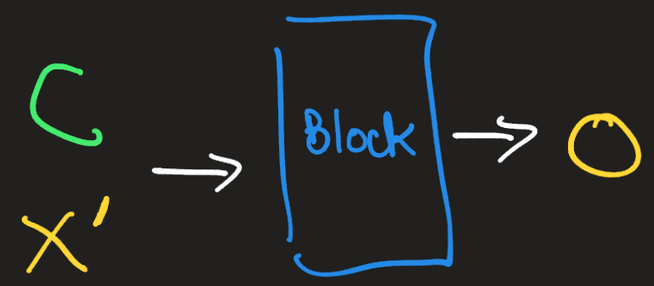
and the final representations from the encoder of the input sequence $C = \begin{bmatrix} c_1 \\ \vdots \\ c_m \end{bmatrix}$:

(i) $Y = \text{LayerNorm}(X' + \text{Multihead Att}(X', X'))$

(ii) $Z = \text{LayerNorm}(Y + \text{Multi-head Att}(C, Y))$

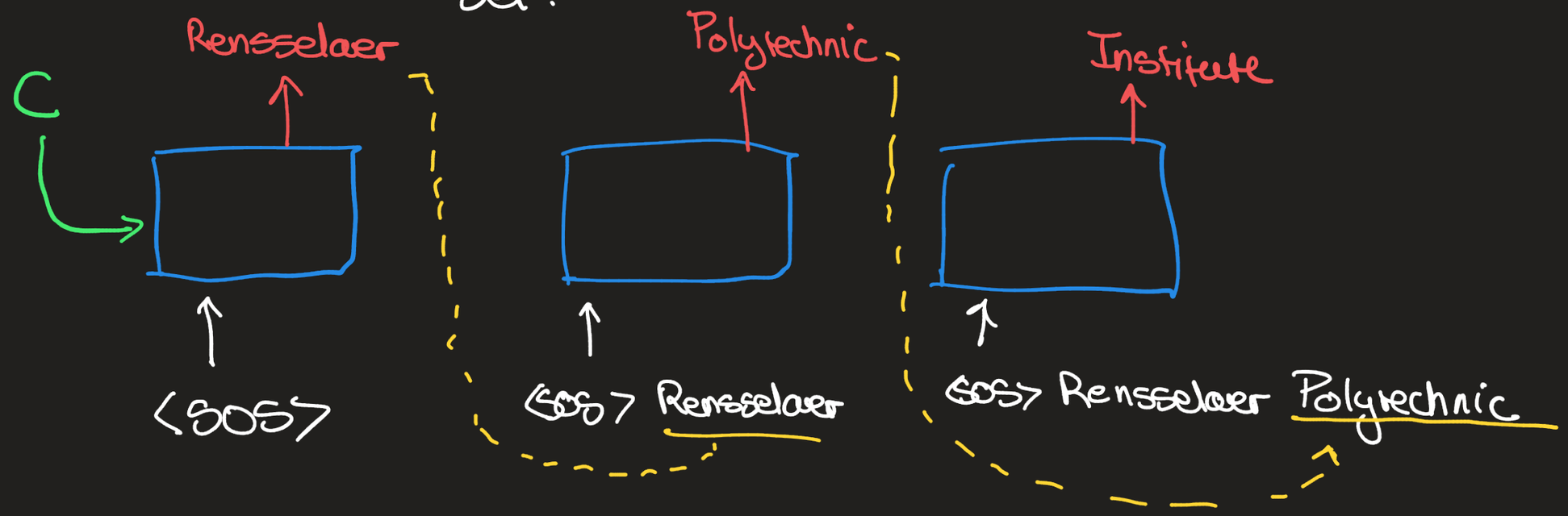
(iii) $O = \text{LayerNorm}(Z + FF(Z))$

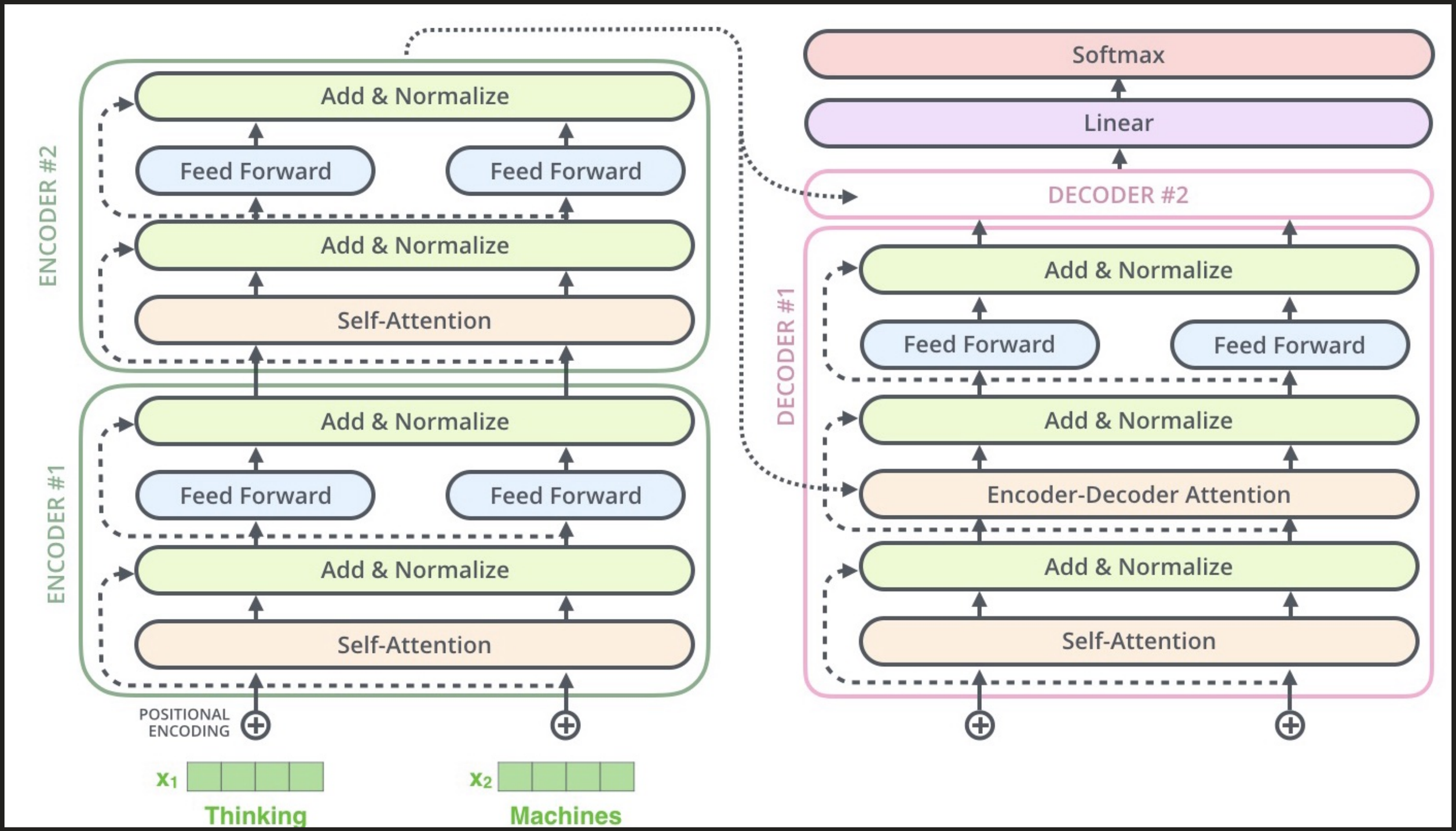




In the self-attention layers, the attention is masked: we set $(\alpha_i)_j = 0$ for $j > i$ to ensure we only predict using historical info.

Decoder During inference time, we autoregressively use the decoder to predict the next token, add it onto our input sequence, and pass it back through the decoder to get the next token. Repeat until we either get an <EOS> token, or we've reached the maximum sequence length we set.




ENCODER #2
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
ENCODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
POSITIONAL ENCODING
x1
Thinking
x2
Machines
Softmax
Linear
DECODER #2
DECODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Encoder-Decoder Attention
Add & Normalize
Self-Attention

Positional Encodings

Issue w/ Transformers as is: they don't properly learn positional relations (e.g. that "the cat ate its food" should have a diff contextual rep from "the food ate its cat")

Soln: replace $X \in \mathbb{R}^{m \times d}$ with $X + P$

where $P \in \mathbb{R}^{m \times d}$ encodes the positions 1 through m

$$P_{i,j} = \begin{cases} \sin(i \omega^{j/d}) & \text{if } j \text{ even} \\ \cos(i \omega^{(j-1)/d}) & \text{if } j \text{ odd} \end{cases}$$

(refer to preceeding diagram to see where positional embeddings are added)