ML & Opt Lecture 21

Sequence-to-Sequence Modeling
- architecture using RNNs
- Training & Inference
- Teacher forcing
- Perplexity - performance metric
- Tricks for better performance
- Attention

Sequence-to-Sequence Modeling (2014 used for very good machine translation, popularized deep LSTM seq2seq model)

"Sutskever et al.
Sequence to sequence learning
with neural networks"

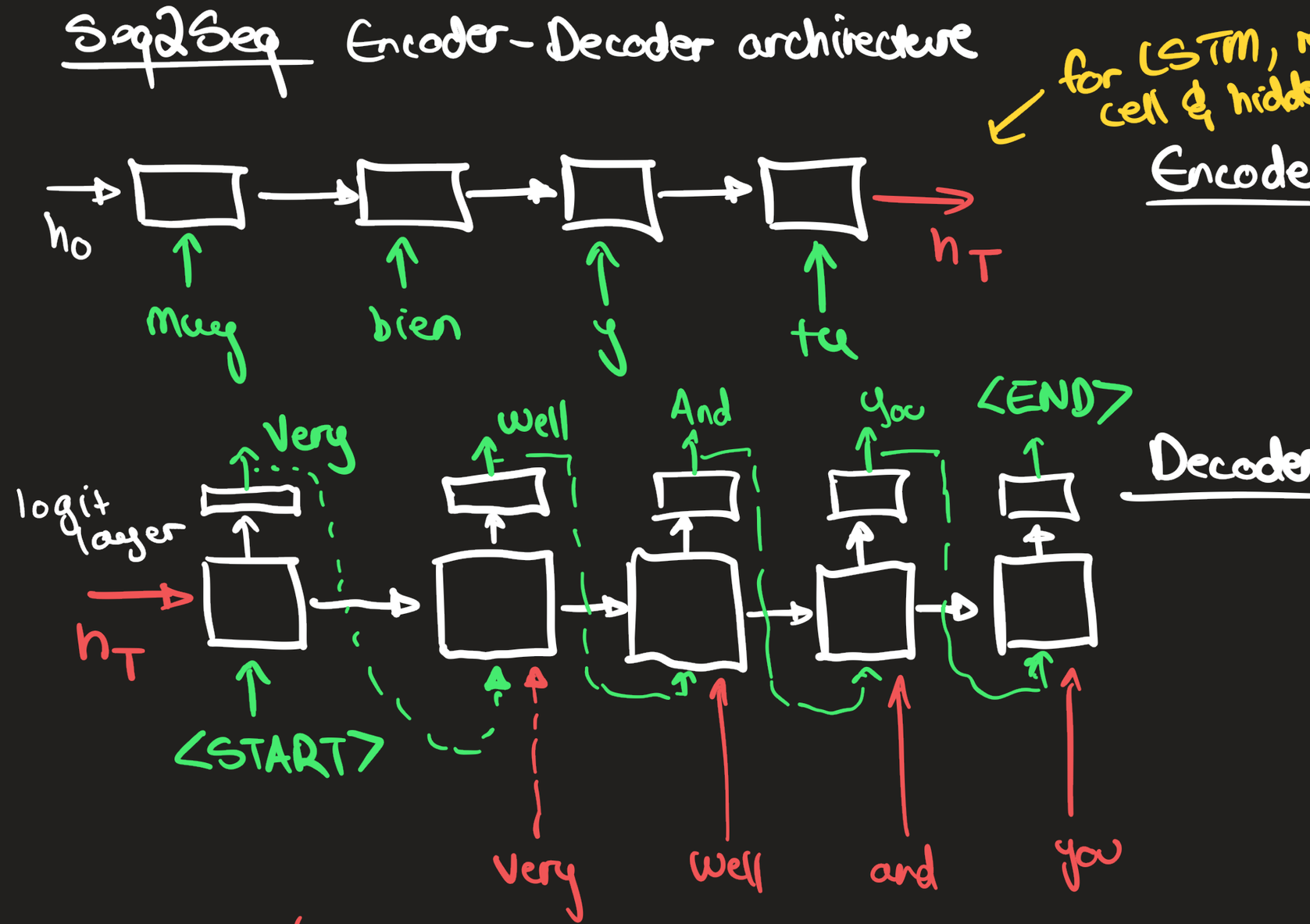
Goal: given input sequence of tokens $(x_1, \cdots, x_T)$
predict matching output sequence of tokens $(y_1, \cdots, y_{T'})$

Canonical example: Machine Translation

Issue: $T \neq T'$ in general so the many-to-many design paradigm for RNNs may not fit

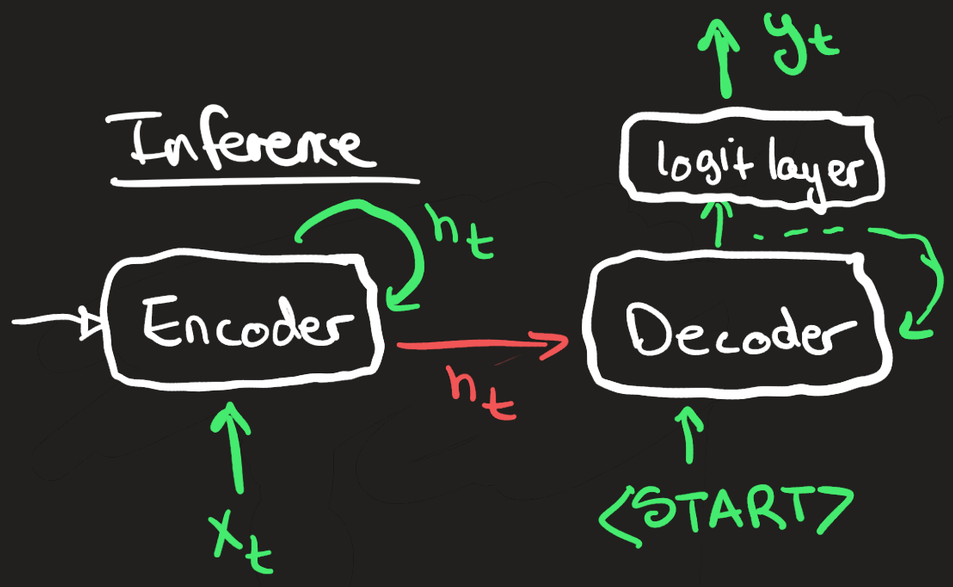
Soln: encoder-decoder architecture

Seq2Seq Encoder-Decoder architecture



Training

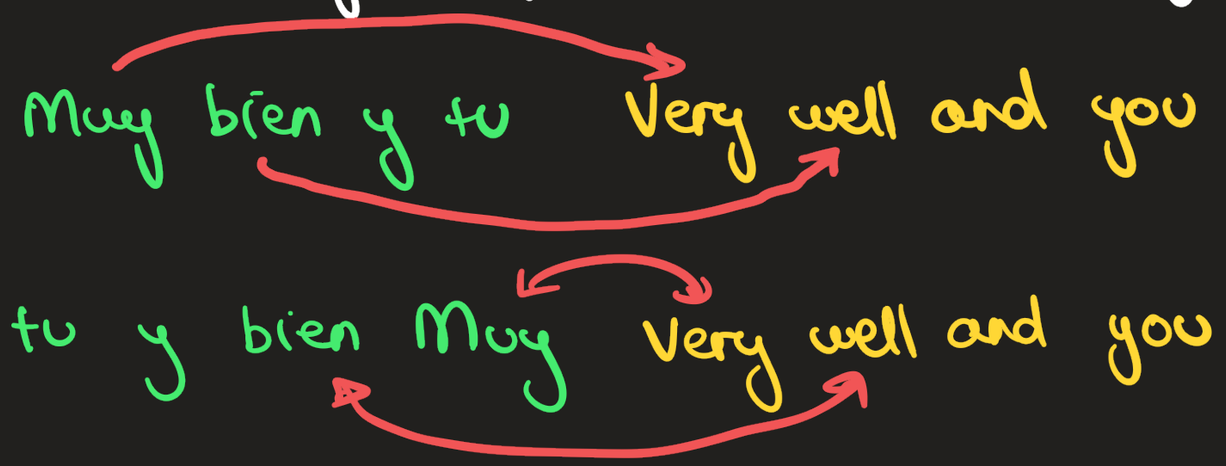
- train the encoder & decoder simultaneously (using BPTT) so that given the input sequence $(x_1, \dots, x_T)$ and output sequence $(y_1, \dots, y_{T'})$
  - encoder learns an informative h_T for decoding
  - decoder learns how to, given h_T and a special <START> token, generate the correct output followed by an <END> token.
- Since the output $\hat{y}_t$ depends on $\hat{y}_{t-1}$ we can have very slow training if $\hat{y}_t$ is inaccurate at the start of the sequence, because the errors compound. To mitigate, during training we can use teacher forcing: use y_{t-1} (the ground truth) as the input to predict $\hat{y}_t$



- Given input sequence $(x_1, \cdots, x_T)$ use encoder to obtain h_T
- initialize the decoder w/ hidden state h_T and feed it the <START> token
- at each time, feed $\hat{y}_{t-1}$ into the decoder to predict the next token and take $\hat{y}_t$ to be the most likely token

Practical tips

- use teacher forcing (sometimes)
- use deep LSTMs in the encoder and decoder
- reverse the order of the source sequence to introduce shorter length dependencies w/ the target sequence



- use "beam decoding" in practice (see Sutskever et al.)

Decoding (at inference time)

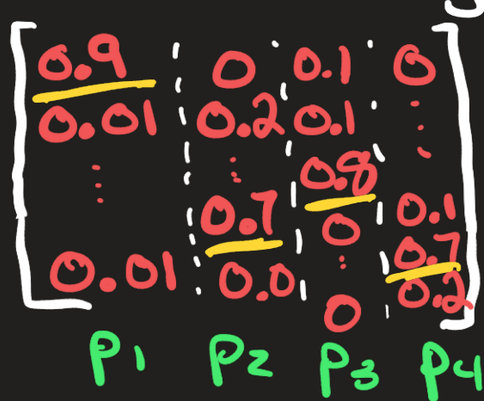
Given an input sequence $x_1, \dots, x_T$, a seq2seq model produces an output $p_1, \dots, p_{T'}$ where each $p_i \in \mathbb{R}^{|V|}$ is a probability vector over the output token vocabulary. Need to decode $p_1, \dots, p_{T'}$ into a sequence of tokens $y_1, \dots, y_{T'}$.

E.g. seq2seq(Muy, bien, y, tu) =

	p_1	p_2	p_3	p_4	
	0.9	0	0.1	0	very
	0.01	0.2	0.1		Not
			0.8		And
		0.7	0	0.1	well
	0.01	0.0		0.7	you
			0	0.2	<UNK>

How to convert to tokens?

Option 1 Greedy Decoding: take $y_i = \underset{j \in |V|}{\text{argmax}} (p_i)_j$



Greedy ⟶

Very, you, and, you

(determine T' by running until <EOS> becomes very likely, or reach too many tokens)

Option 2 Full decoding

Notice that

$$\mathbb{P}(y_1, \dots, y_{T'}) = \mathbb{P}(y_1)\,\mathbb{P}(y_2 \mid y_1) \cdots \mathbb{P}(y_{T'} \mid y_1, \dots, y_{T'-1})$$

$$= (p_1)_{y_1} (p_2)_{y_2} \cdots (p_{T'})_{y_{T'}}$$

so take

$$y_1, \dots, y_{T'} = \underset{j_1, \dots, j_{T'}}{\operatorname{argmax}} \prod_{i=1}^{T'} (p_i)_{j_i}$$

This requires searching over $|V|^{T'}$ sequences.

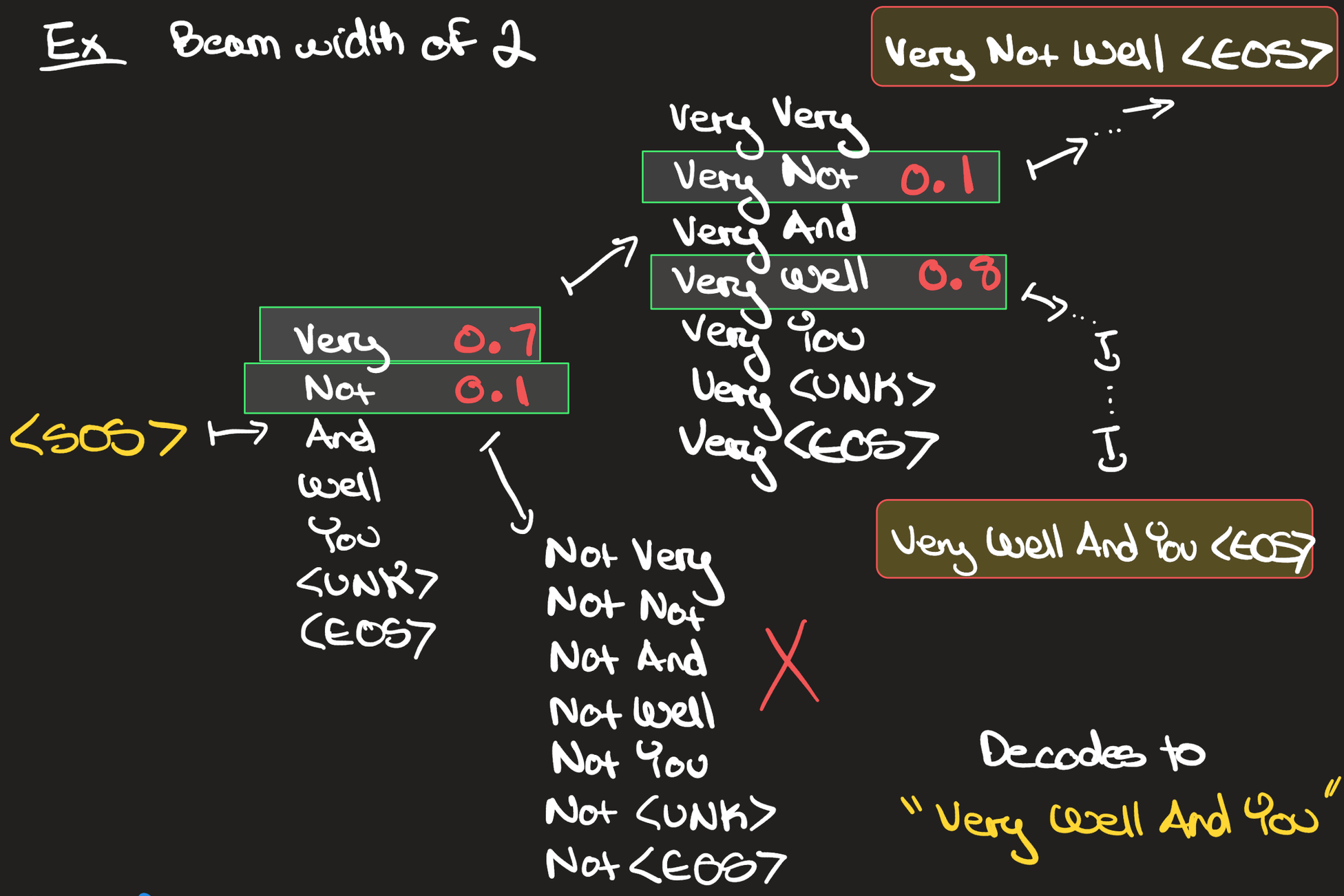
Consider $|V| = 20K$ and $T' = 50$. Even for moderately sized problems, this is intractable!

Option 3 Beam decoding

Maintain B "beams", lists of most likely partial prefixes for the output sequence.

1) At each time step, for each beam, consider adding a single token. This gives $B|V|$ new potential prefixes. Replace the original beams with the B most probable of these prefixes.
2) If any beam contains an <EOS>, place it in the candidate set C, and reduce the beam width, $B \leftarrow B-1$. Go to (1) if $B \neq 0$
3) Take the output to be the most probable sequence in C

Ex Beam width of 2



Decodes to
"Very Well And You"

$C = \{$ Very Well And You (0.41) , Very Not Well (0.01) $\}$

Measuring Performance (Perplexity)

Consider the cross-entropy loss on an input sequence

$$\ell(f(x_1,\dots,x_T),(y_1,\dots,y_{T'})) = -\frac{1}{T'}\sum_{i=1}^{T'}\log((p_i)_{y_i})$$

average NLL over the words in the sequence

This is small and somewhat difficult to interpret.

Instead, consider its exponentiation

$$e^{\ell(\dots)} = e^{-\frac{1}{T'}\sum_{i=1}^{T'}\log((p_i)_{y_i})} = \sqrt[T']{\frac{1}{\prod_{i=1}^{T'}\mathbb{P}(y_i \mid y_1,\dots,y_{i-1})}}$$

inverse geometric mean of conditional prob of the words in the sequence

This is called the "perplexity" on this input example

Interpretation/Use of Perplexity

- Perplexity is a measure of how confident a model is in its predictions
- High perplexity = low confidence
  Low perplexity = high confidence
- perplexity ≥ 1
- Models w/ larger vocabs but the same perplexity are better than models w/ smaller vocabs
- low perplexity means the outputs are closer to deterministic

Seq2Seq models w/ attention

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

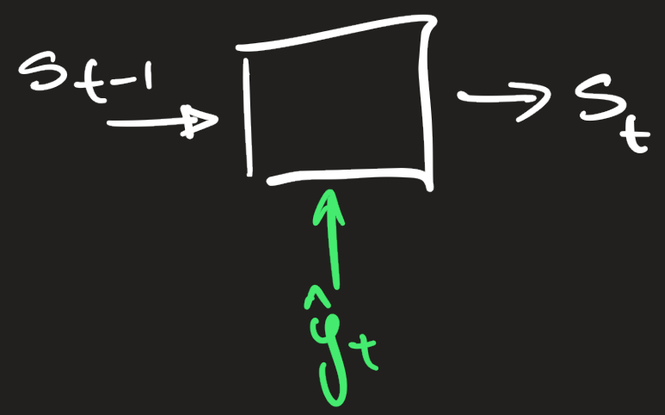
Leads to poor performance for long sequences.

A remedy 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 the idea of focusing on the relevant portion of input sequences. Avoids the need to capture everything relevant to all outputs $\hat{y}_1, \ldots, \hat{y}_{T'}$ in one vector h_T

Seq2Seq w/ Attention

replace our previous hidden state update for the decoder



i.e. $s_t = g(s_{t-1}, \hat{y}_t)$

w/ first finding which historical hidden states from the input are most relevant

$s_{t-1}, [h_1, \dots, h_T] \longmapsto \alpha_t \in \mathbb{R}^T$ a probability distribution over $[h_1, \dots, h_T]$

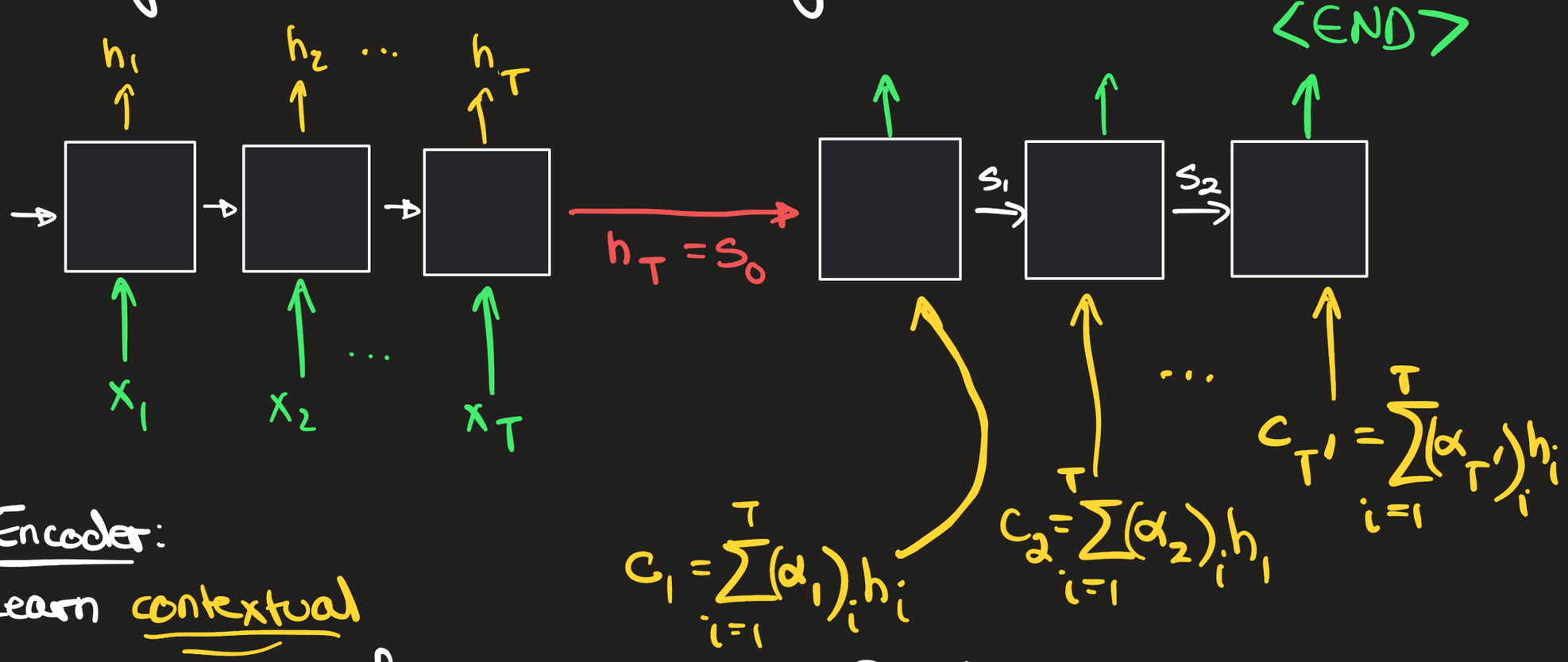
then form a context vector

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

and update with

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

Diagram of our Decoder using Attention



Encoder:

Learn contextual representations of the input tokens

Decoder

Determine which input tokens are relevant to generating the next output (compute α_t attention vector) and combine all input representations h_i to form context vectors c_t

Options for computing attention vectors α_t

- **Option 1** (used in Bahdanau et al. 2015)

Take $\tilde{\alpha}_i = \underbrace{v^T}_{\text{parameters to be learned}} \tanh\left(\underbrace{W}_{\text{parameters to be learned}} \begin{bmatrix} h_i \\ s_{t-1} \end{bmatrix}\right)$ for $i = 1, \dots, T$

$\alpha_t = \text{softmax}(\tilde{\alpha})$ "MLP attention"?

- **Option 2** (Dot product attention)

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

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

$W_Q s_{t-1} = q_{t-1}$ $\quad \alpha_t = \text{softmax}(\tilde{\alpha})$

Complexity of using attention

- Clear we have introduced parameters

Quadratic dependence on the sequence lengths?

Effort/time to use a RNN seq2seq model:

1) Simple RNN models: $O(T+T')$

2) RNN attention models: $O(TT')$

because for each output token, must compute $\alpha_t \in \mathbb{R}^T$ by comparing to all inputs, then form a weighted average of all inputs.