ML & Opt Lecture 22

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)

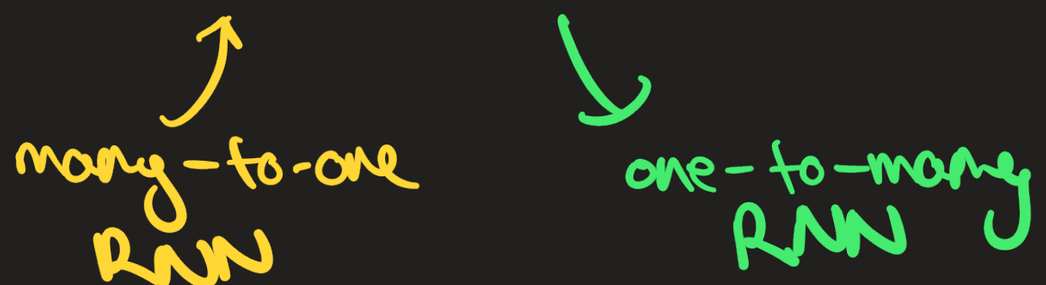
"Sutskever et al.
Sequence to sequence learning with neural networks"

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

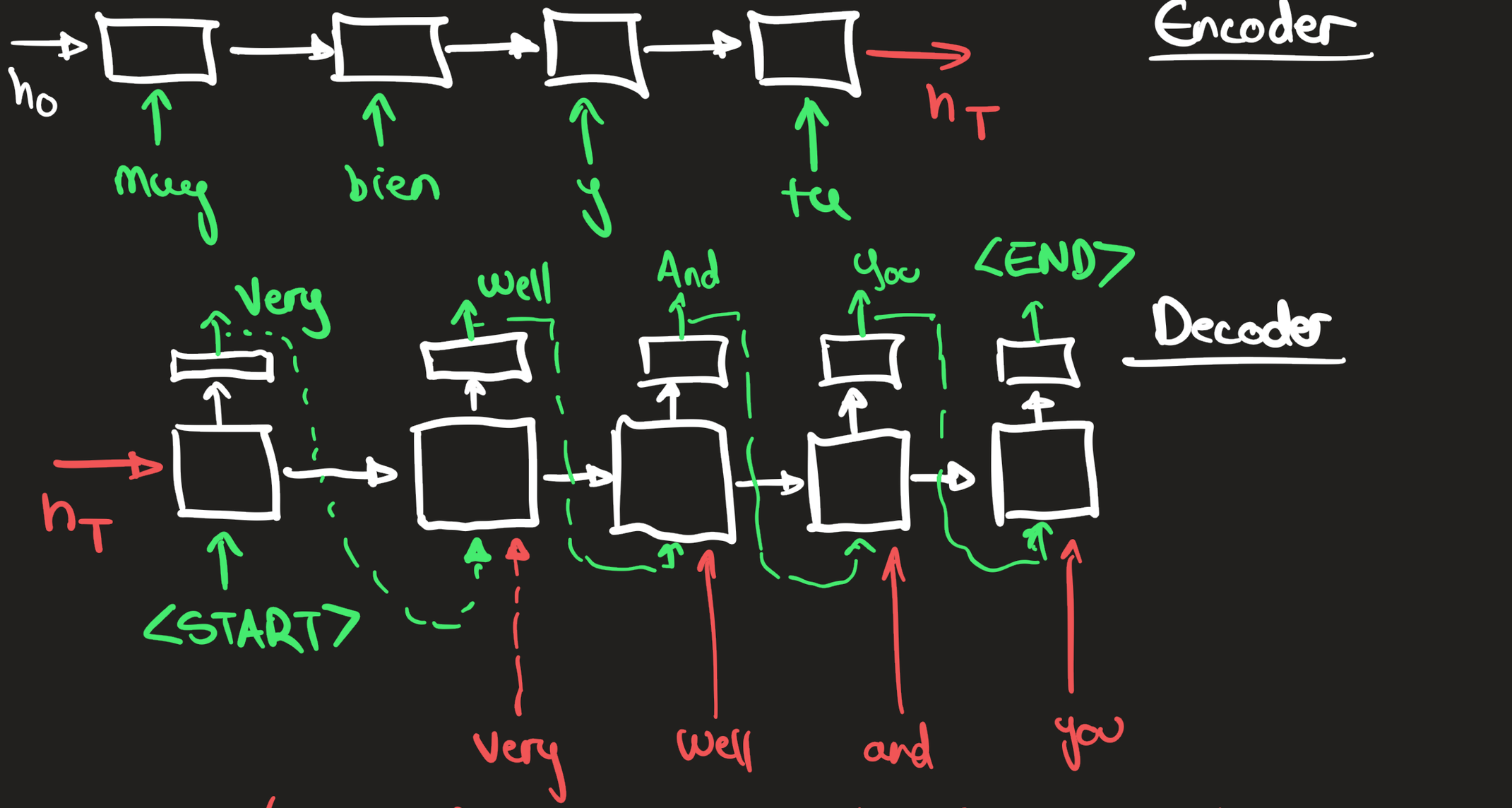
Canonical example: Machine Translation

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

Soln: encoder-decoder architecture



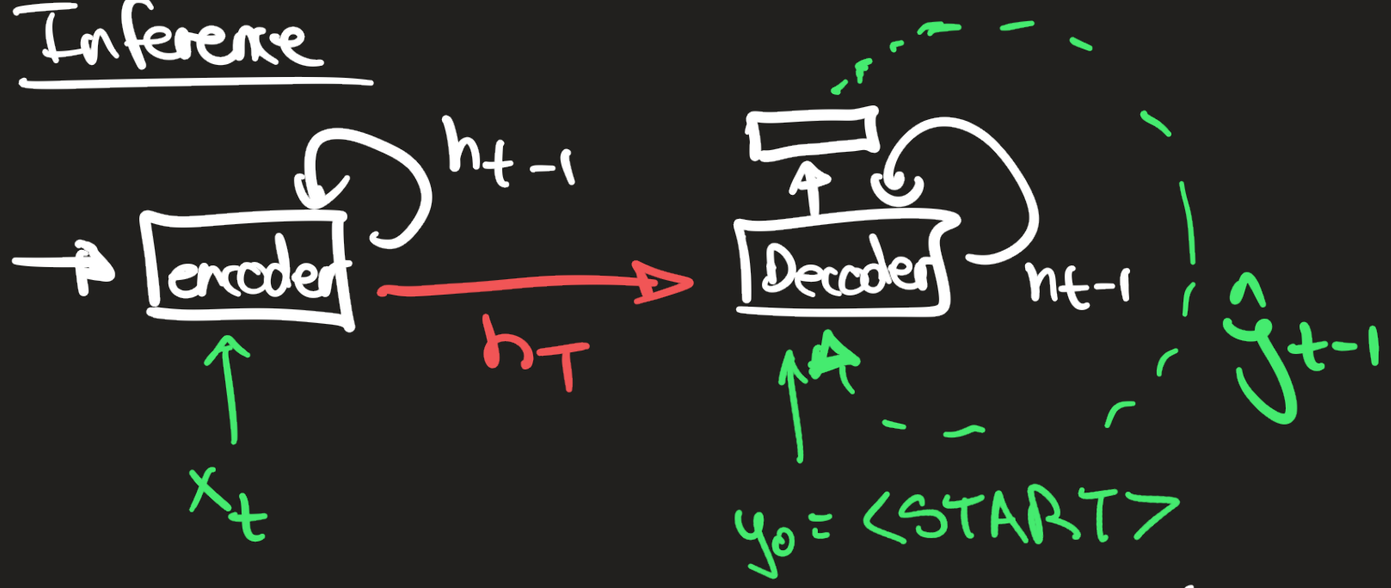
Seq2Seq Encoder-Decoder architecture



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 $\hat{y}_t$

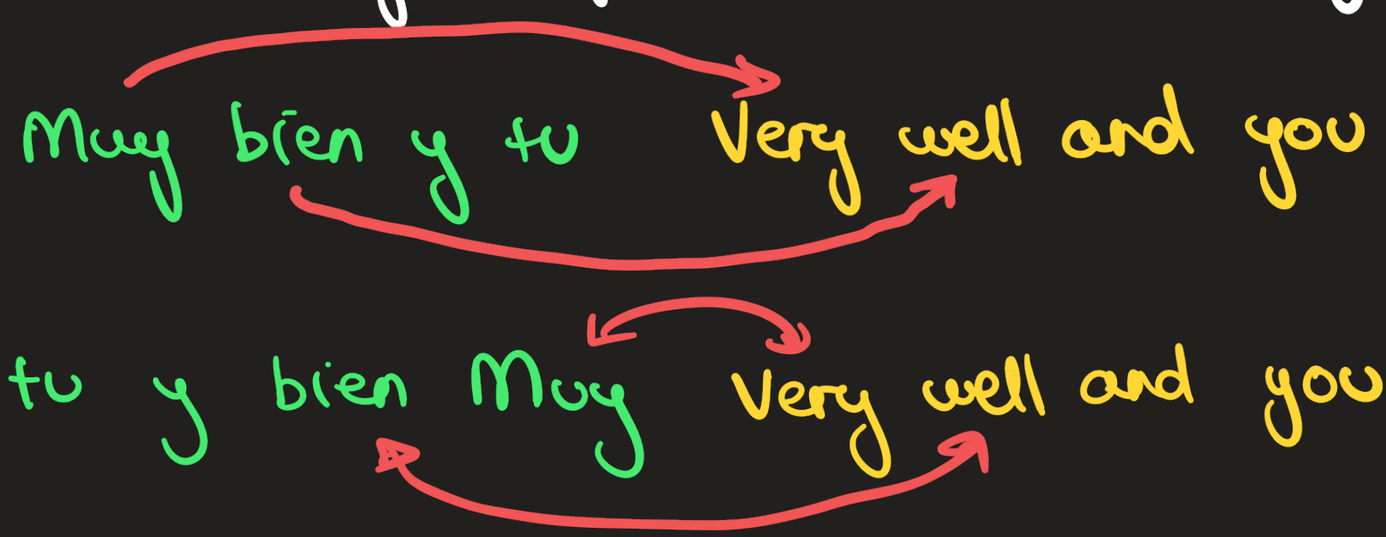
Inference



- Given input sequence $(x_1, \dots, x_T)$ 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.)

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

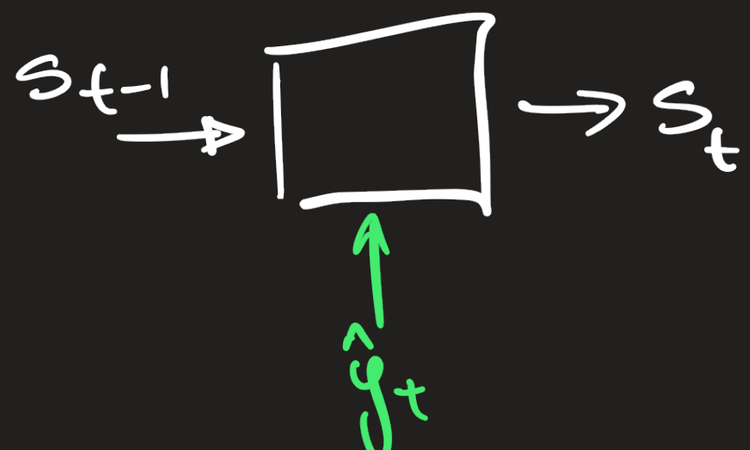
Leads to poor performance for long sequences.

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

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

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

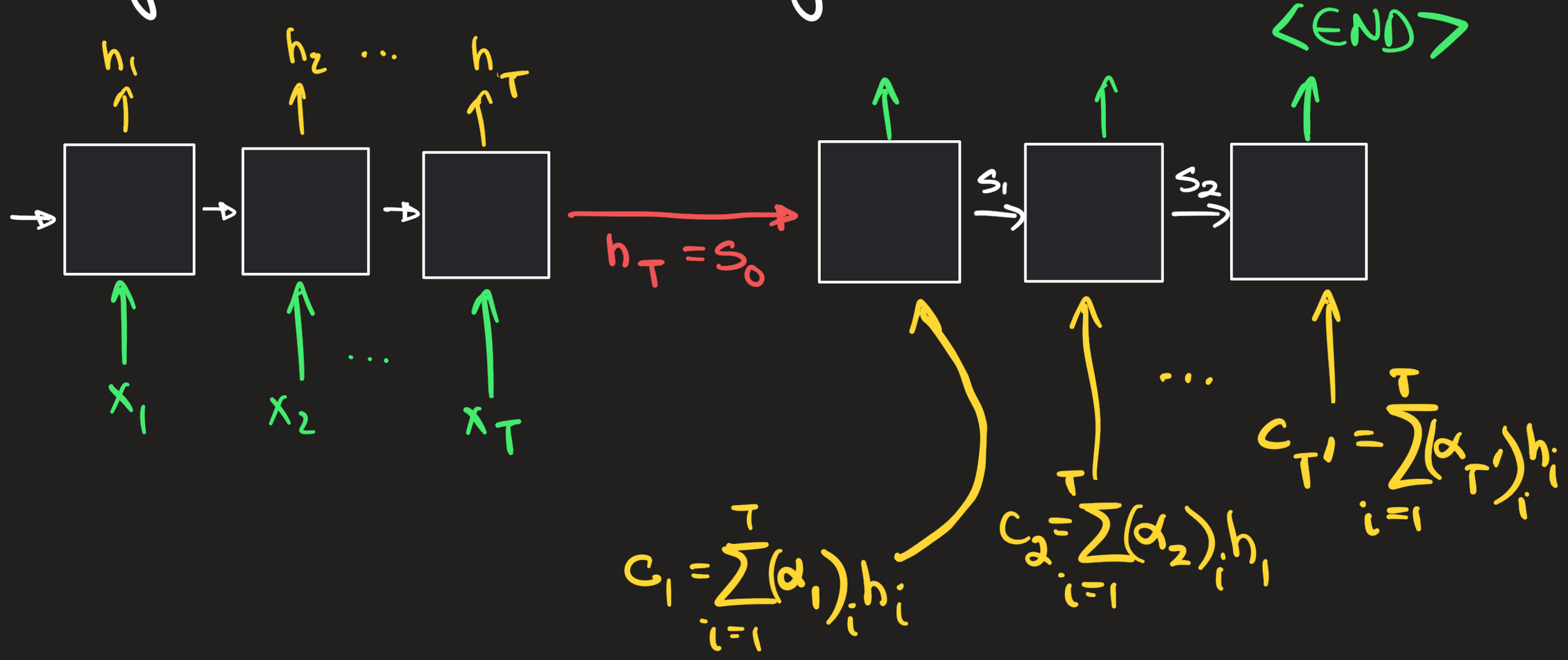
then form a context vector

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

and update with

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

Diagram of our Decoder using Attention



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.