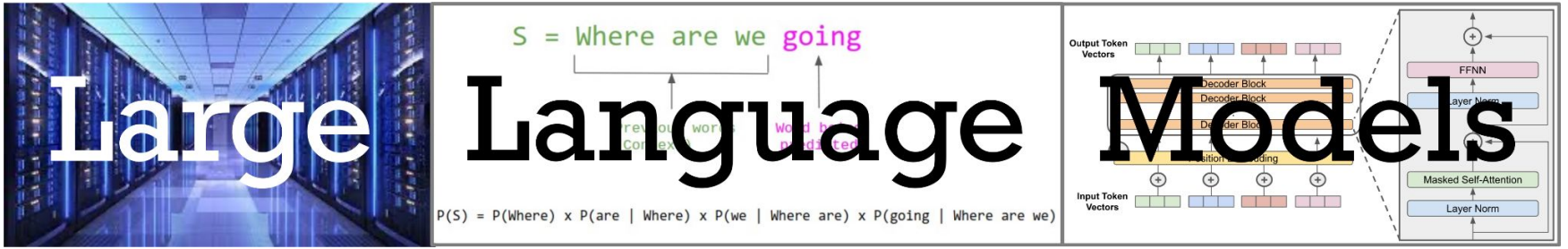


CIS 7000 - Fall 2024

The Pre-Transformer Era

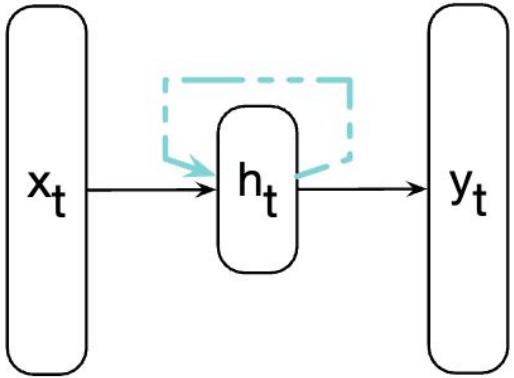
Professor Mayur Naik

Slides adapted in part from Stanford's CS224N: Natural Language Processing with Deep Learning (Spring'24) and Chapter 8 of Jurafsky/Martin's book "Speech and Language Processing" (3rd ed).

Today’s Agenda

- Recurrent Neural Networks (RNNs)
- Variants and Applications of RNNs
- Limitations of RNNs
- The Seq2Seq Architecture
- The Attention Mechanism

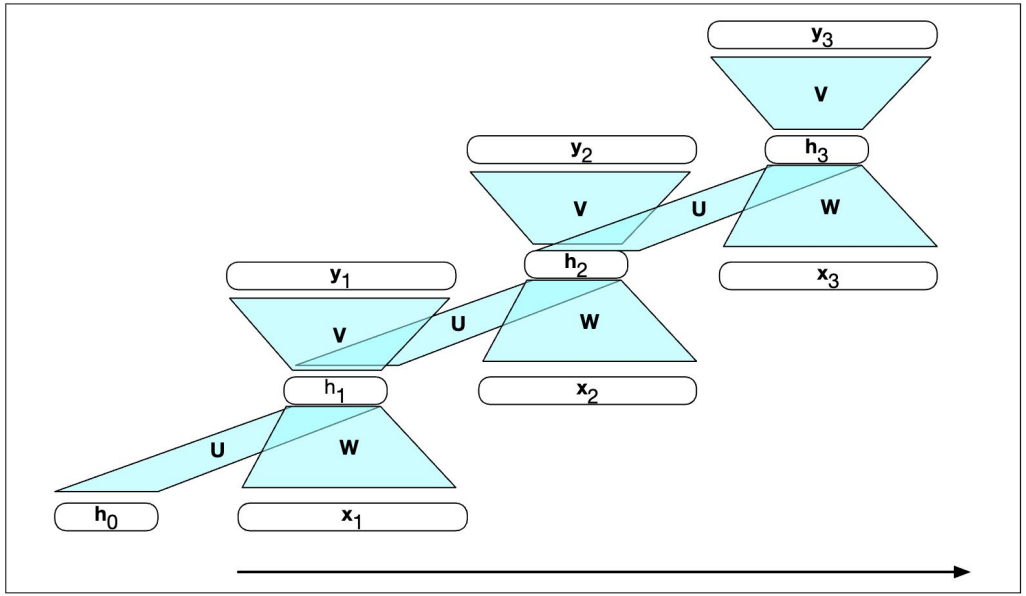
Recurrent Neural Networks



$$\mathbf{h}_t = \sigma\left(\mathbf{U}\,\mathbf{h}_{t-1} + \mathbf{W}\,\mathbf{x}_t\right)$$

$$\mathbf{y}_t = \text{softmax}\left(\mathbf{V}\,\mathbf{h}_t\right)$$

Recurrent Neural Networks: Forward Pass



$\mathbf{h}_0 = 0$

for **t** = 1 to length(**x**) do

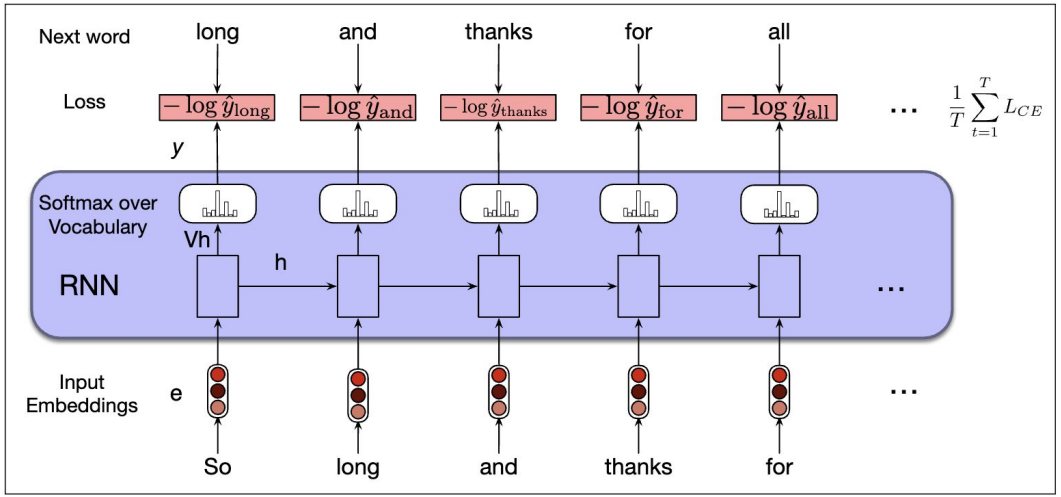
$\mathbf{h}_t = \sigma(\mathbf{U}\,\mathbf{h}_{t-1} + \mathbf{W}\,\mathbf{x}_t)$

$\mathbf{y}_t = \text{softmax}(\mathbf{V}\,\mathbf{h}_t)$

return **y**

The matrices **U**, **V** and **W** are shared across time, while new values for **h** and **y** are calculated with each time step.

Recurrent Neural Networks: Training

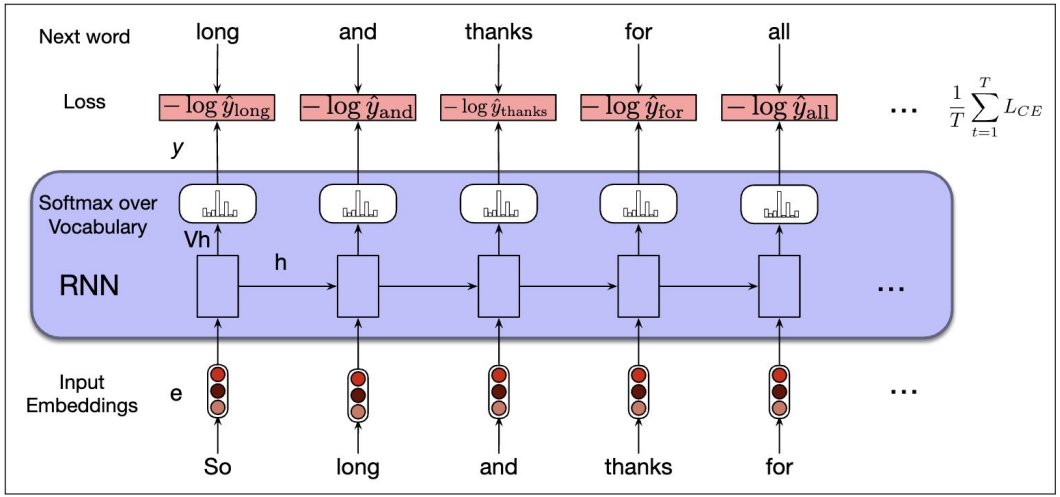


At each word position **t** of the input, the model takes word $\mathbf{w}_t$ together with $\mathbf{h}_{t-1}$, encoding information from the preceding $\mathbf{w}_{1:t-1}$, and uses them to compute a probability distribution over possible next words so as to compute the model's loss for the next word $\mathbf{w}_{t+1}$.

Then we move to the next word, ignore what the model predicted for the next word and instead use the correct word $\mathbf{w}_{t+1}$ along with the prior history encoded to estimate the probability of word $\mathbf{w}_{t+2}$.

This idea that we always provide the model the correct history sequence to predict the next word (rather than feeding the model its best case from the previous time step) is called **teacher forcing**.

Recurrent Neural Networks: Backpropagation

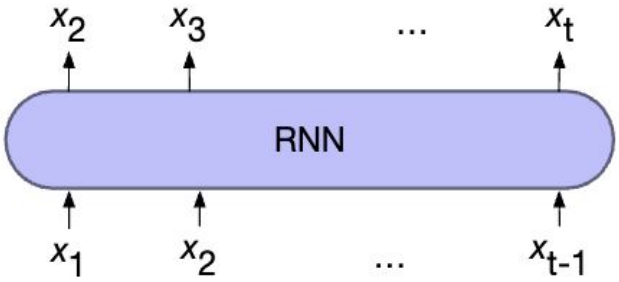


Backpropagate over timesteps **t** = **T**, … ,**1**, summing gradients as you go. This algorithm is called “backpropagation through time”.

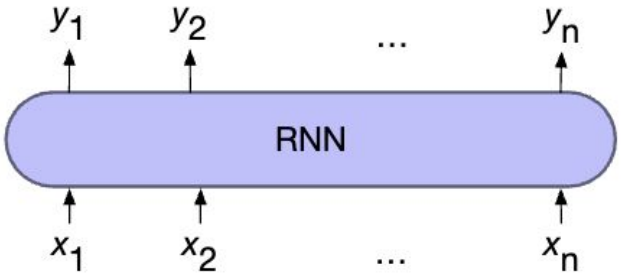
Computing loss and gradients over entire corpus at once is too expensive (memory-wise). In practice, consider (a batch of) sentences at a time.

In practice, often “truncated” after ~20 timesteps for training efficiency reasons.

RNN Architectures for Different NLP Tasks

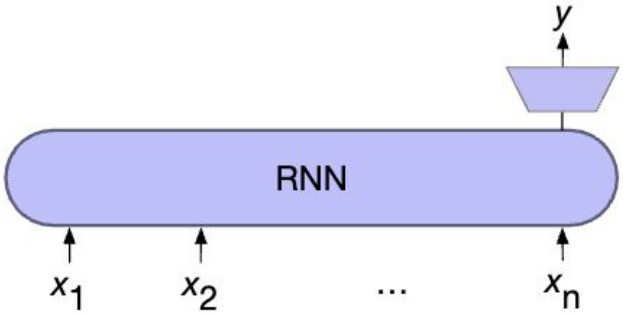


Language Modeling



Sequence Labeling

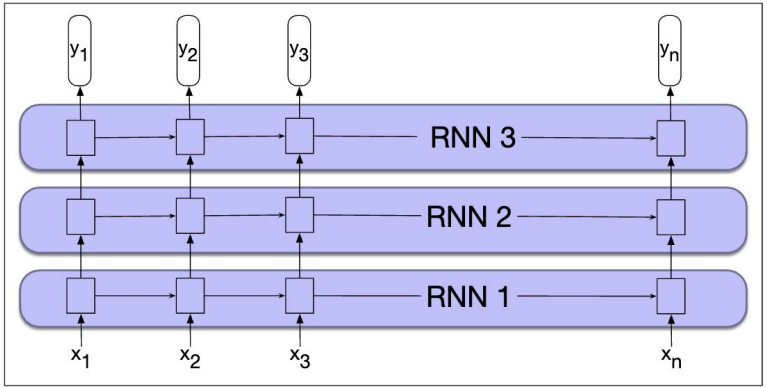
Applications:
Part-of-speech tagging,
Named entity recognition



Sequence Classification

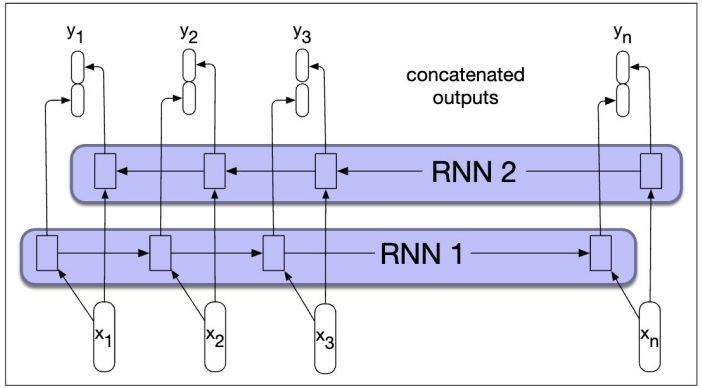
Applications:
sentiment analysis,
spam detection

Variations of RNNs



Stacking

The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output. The resulting network induces representations at differing levels of abstraction across layers.



Bidirectional

Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

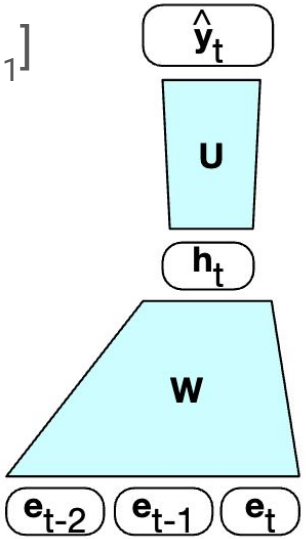
Feedforward vs. Recurrent Neural Network as Language Model

$\mathbf{e} = [\mathbf{E}\, x_{t-3}; \mathbf{E}\, x_{t-2}; \mathbf{E}\, x_{t-1}]$

$\mathbf{h} = \sigma(\mathbf{W}\, \mathbf{e})$

$\mathbf{z} = \mathbf{U}\, \mathbf{h}$

$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$

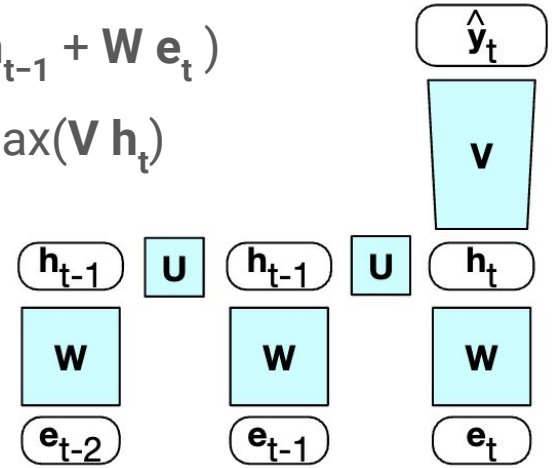


Feedforward Neural Network

$\mathbf{e}_t = \mathbf{E}\, x_t$

$\mathbf{h}_t = \sigma(\mathbf{U}\, \mathbf{h}_{t-1} + \mathbf{W}\, \mathbf{e}_t)$

$\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{V}\, \mathbf{h}_t)$



Recurrent Neural Network

Pros/Cons of Recurrent Neural Network as Language Model

Pros:

- Can process any length input!
- Computation for step **t** can (in theory) use information from many steps back.
- Model size (**W**) doesn't increase for longer input context.
- Each $\mathbf{x}_i$ is multiplied by same weights in **W**, so there is symmetry in how inputs are processed.

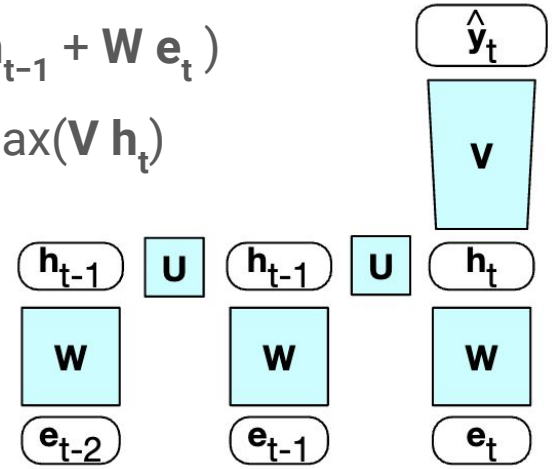
Cons:

- Recurrent computation is slow (cannot be parallelized).
- In practice, it is difficult to access information from many steps back.

$$\mathbf{e}_t = \mathbf{E}\,\mathbf{x}_t$$

$$\mathbf{h}_t = \sigma\left(\mathbf{U}\,\mathbf{h}_{t-1} + \mathbf{W}\,\mathbf{e}_t\right)$$

$$\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{V}\,\mathbf{h}_t)$$



Vanishing Gradient Problem!
RNN extensions such as LSTMs (or other varieties like GRUs) are commonly used.

Effect of Vanishing Gradient on RNN-Based Language Model

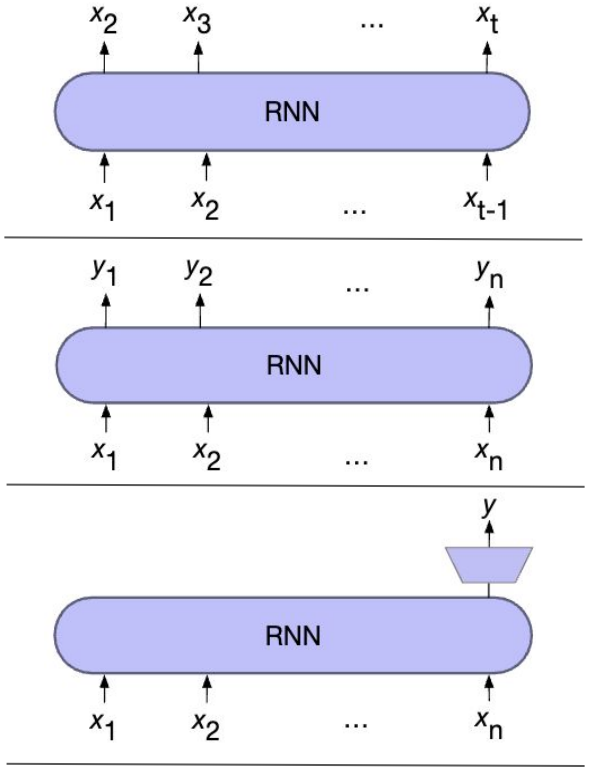
LM task: *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____*

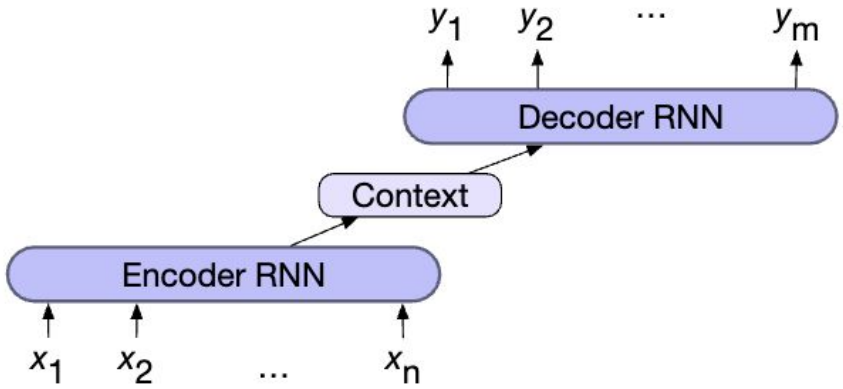
To learn from this training example, the RNN-LM needs to model the dependency between *"tickets"* on the 7^{th} step and the target word *"tickets"* at the end.

But if the gradient is small, the model can't learn this dependency. So, the model is unable to predict similar long-distance dependencies at test time.

More on vanishing/exploding gradient problems in training RNNs: R. Pascanu et al. On the difficulty of training recurrent neural networks. ICML 2013.

Encoder-Decoder or Seq2Seq Architecture



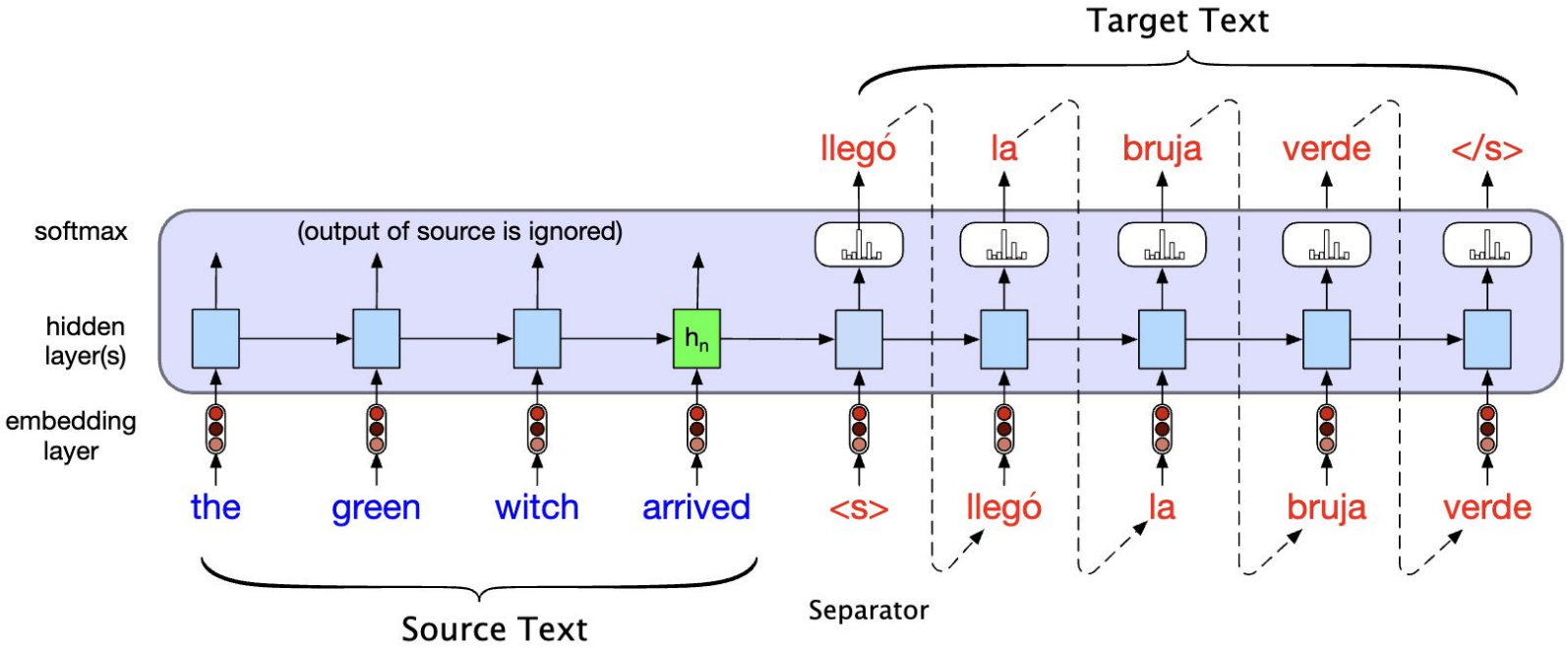


Taking an input sequence and translating it to an output sequence that is of a different length than the input, and doesn't align with it in a word-to-word way.

Many applications: machine translation, summarization, question-answering, dialogue, ...

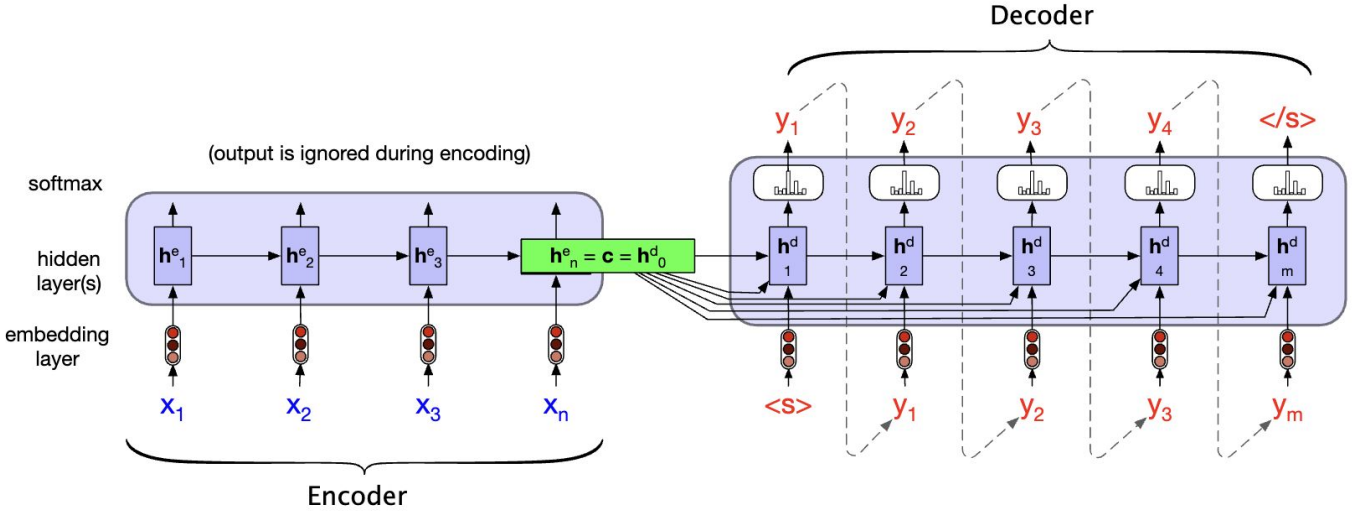
I. Sutskever et al. Sequence to Sequence Learning with Neural Networks. NeurIPS 2014.

Illustrative Example: Machine Translation



Encoder-Decoder RNN, More Formally

The final hidden state of the encoder RNN, $\mathbf{h}_n^e$, serves as the context for the decoder in its role as $\mathbf{h}_0^d$ in the decoder RNN, and is also made available to each decoder hidden state.



$$\mathbf{c} = \mathbf{h}_n^e$$

$$\mathbf{h}_0^d = \mathbf{c}$$

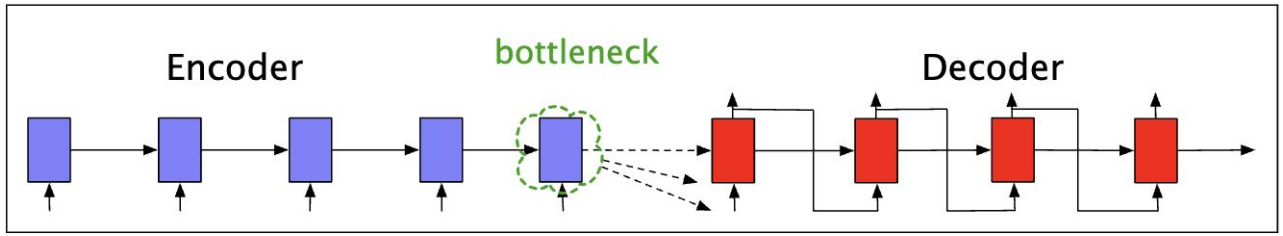
$$\mathbf{h}_i^d = g(\hat{\mathbf{y}}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c})$$

$$\hat{\mathbf{y}}_i = \text{softmax}(\mathbf{h}_i^d)$$

where **g** is a stand-in for some flavor of RNN.

The Bottleneck Problem

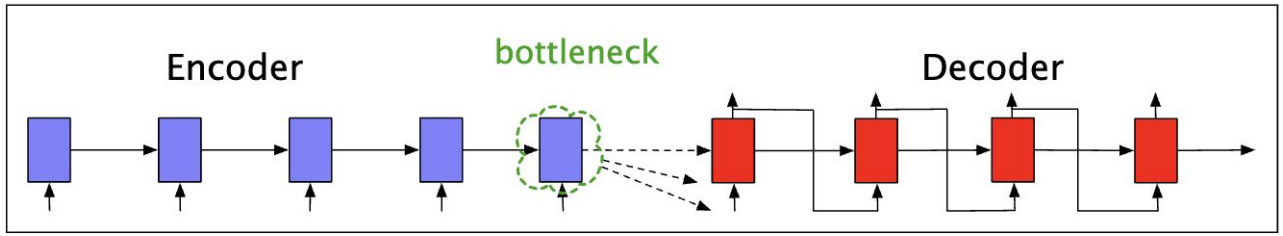
Requiring the context **c** to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck!



D. Bahdanau et al. Neural Machine Translation by Jointly Learning to Align and Translate. ICLR 2015.

Attention Mechanism: A Solution to the Bottleneck Problem

Requiring the context **c** to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck!



$$\mathbf{c} = \mathbf{h}_n^e$$

$$\mathbf{h}_0^d = \mathbf{c}$$

$$\mathbf{h}_i^d = g(\hat{\mathbf{y}}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

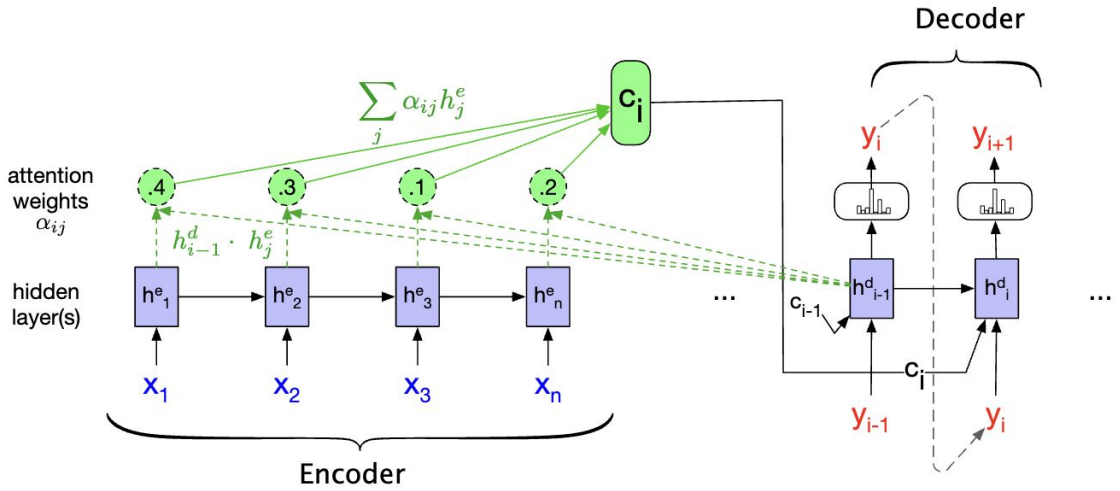
$$\hat{\mathbf{y}}_i = \text{softmax}(\mathbf{h}_i^d)$$

Instead of a static context vector **c**, dynamically generated a context vector $\mathbf{c}_i$ for each decoding step **i** that takes all encoder hidden states into account in its derivation.

$$\mathbf{c}_i = f(\mathbf{h}_1^e, \ldots, \mathbf{h}_n^e)$$

D. Bahdanau et al. Neural Machine Translation by Jointly Learning to Align and Translate. ICLR 2015.

Dot-Product Attention



Finally compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states.

First step in computing $\mathbf{c}_i$: How much to focus on each encoder state $\mathbf{h}_j^e$, or how relevant it is to the decoder state captured in $\mathbf{h}_{i-1}^d$.

$$\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{i-1}^d \cdot \mathbf{h}_j^e$$

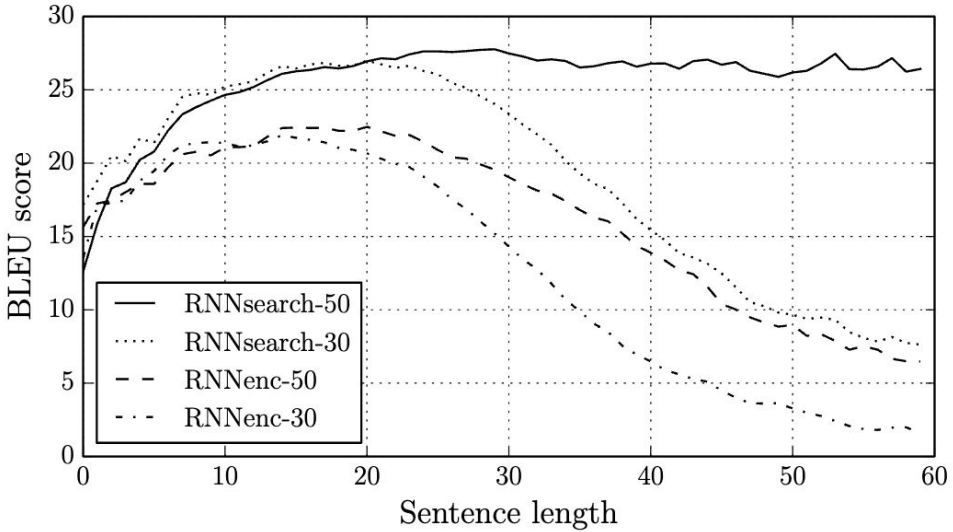
Then normalize the weights.

$$\boldsymbol{\alpha}_{ij} = \text{softmax}(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e))$$

$$\mathbf{c}_i = \sum_j \boldsymbol{\alpha}_{ij} \mathbf{h}_j^e$$

More sophisticated scoring function: $\mathbf{h}_{i-1}^d \mathbf{W} \mathbf{h}_j^e$ using learned weights $\mathbf{W}$.

Experimental Results With vs. Without Attention Mechanism



Performance of RNN encoder-decoder models trained on sentences of length upto 30 or 50:

RNNsearch - with attention
RNNenc - without attention

D. Bahdanau et al. Neural Machine Translation by Jointly Learning to Align and Translate. ICLR 2015.

Up Next …

- **Sept 11** Lecture: The Transformer Architecture: Part I (Impact of Transformers; From Recurrence to Attention; Transformer block).