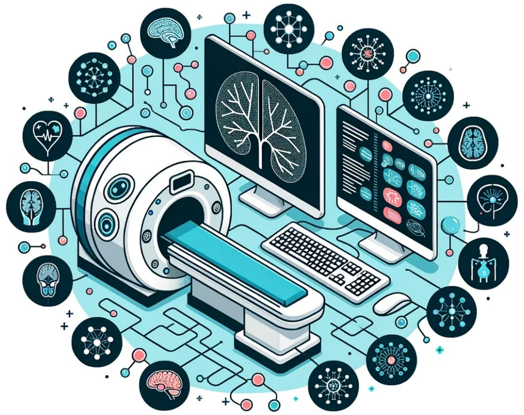
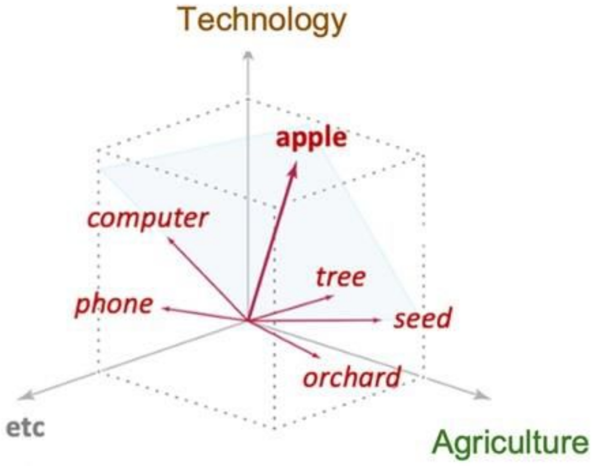


Frontiers in Generative AI for Medical Imaging and Healthcare

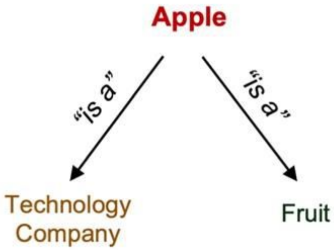
Session 2
Large Language Models (BioBERT, Med-PaLM, fine-tuning)
Sep 13 2025

Probabilistic Model of Language



Word Polysemy

1. I mainly use my **Apple** **iPhone** to make **phone** calls.
2. The **Apple** MacBook Pro is a **computer** with a powerful **processor**.
3. I use an **Apple** **computer** to write **emails** and create **documents**.
4. I picked a red **apple** from the **tree** in the backyard.
5. The planted **seeds** in the **orchard** produced several **apple** **trees**.
6. **Apples** are my favorite type of **fruit**.



Distributional Hypothesis of Word Meaning

Probabilistic Model of Language

Probability model:

1. $p(L) = 1$

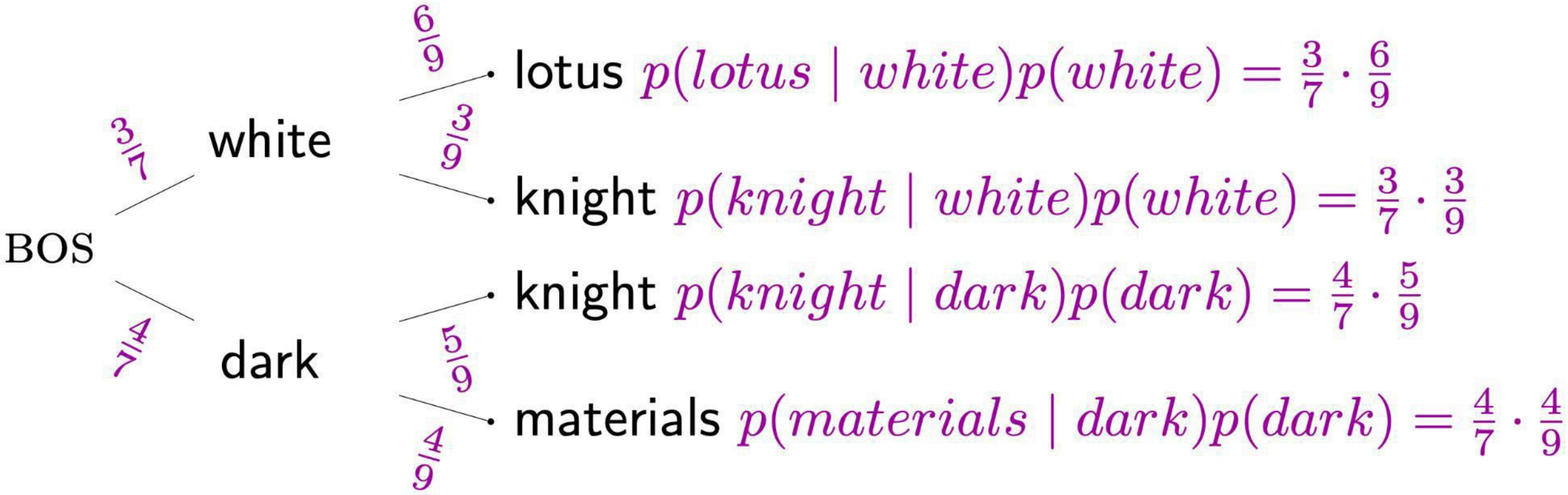
2. $p(\bigcup_{i=1}^{n} \mathcal{E}_i) = \sum_i^n p(\mathcal{E}_i)$ if $\mathcal{E}_1, \mathcal{E}_2, \ldots$ is a countable sequence of disjoint sets of $\mathcal{P}(L)$, the power set (=set of all subsets) of L.

3. (Conditional probability) $$p(\mathbf{x}) = p(x_0) \prod_{i=1}^{L} p(x_i | x_1, \ldots, x_{i-1})$$

$$\log p(\mathbf{x}) = \log p(x_0) \sum_{i=1}^{L} \log p(x_i | x_1, \ldots, x_{i-1})$$

Link

Probabilistic Model of Language



Probabilistic Model of Language

- Given a sequence of words, compute the **probability distribution of the next word:**

$$P(\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $x^{(t+1)}$ **can be any word in the vocabulary** $V = \{\boldsymbol{w}_1, ..., \boldsymbol{w}_{|V|}\}$

$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} \mid \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} \mid \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

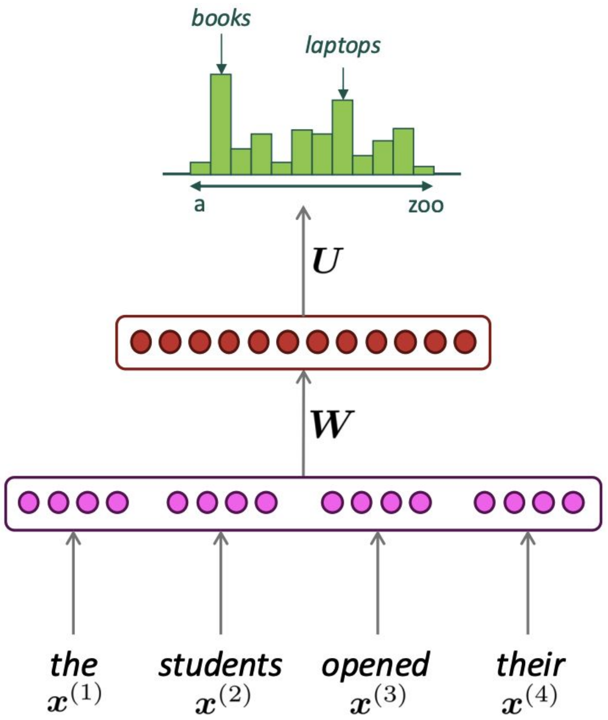
$$= \prod_{t=1}^{T} \underbrace{P(\boldsymbol{x}^{(t)} \mid \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})}_{}$$

This is what the LM provides

Neural Network Model of Language

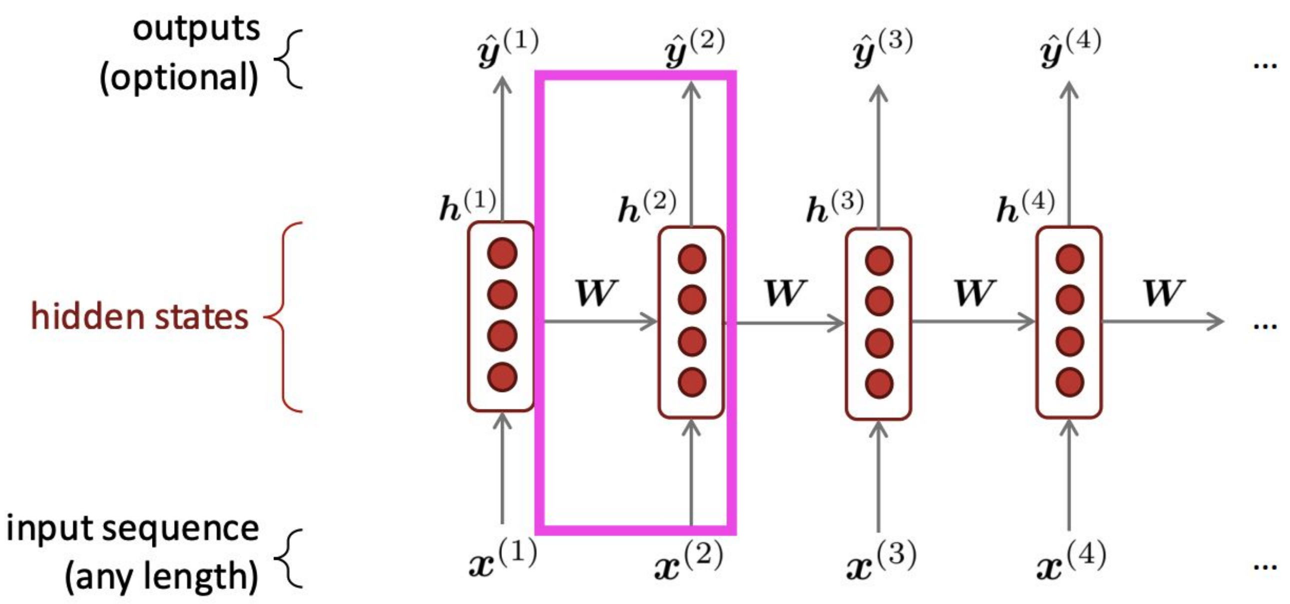
- A neural probabilistic language model (Y. Bengio, et al.)
- Fixed window is small
- No window is large enough

We need a neural architecture that can process *any length input*



Recurrent Neural Networks

- Apply the same weights **W** repeatedly
- Input can be of any length!



Recurrent Neural Networks

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

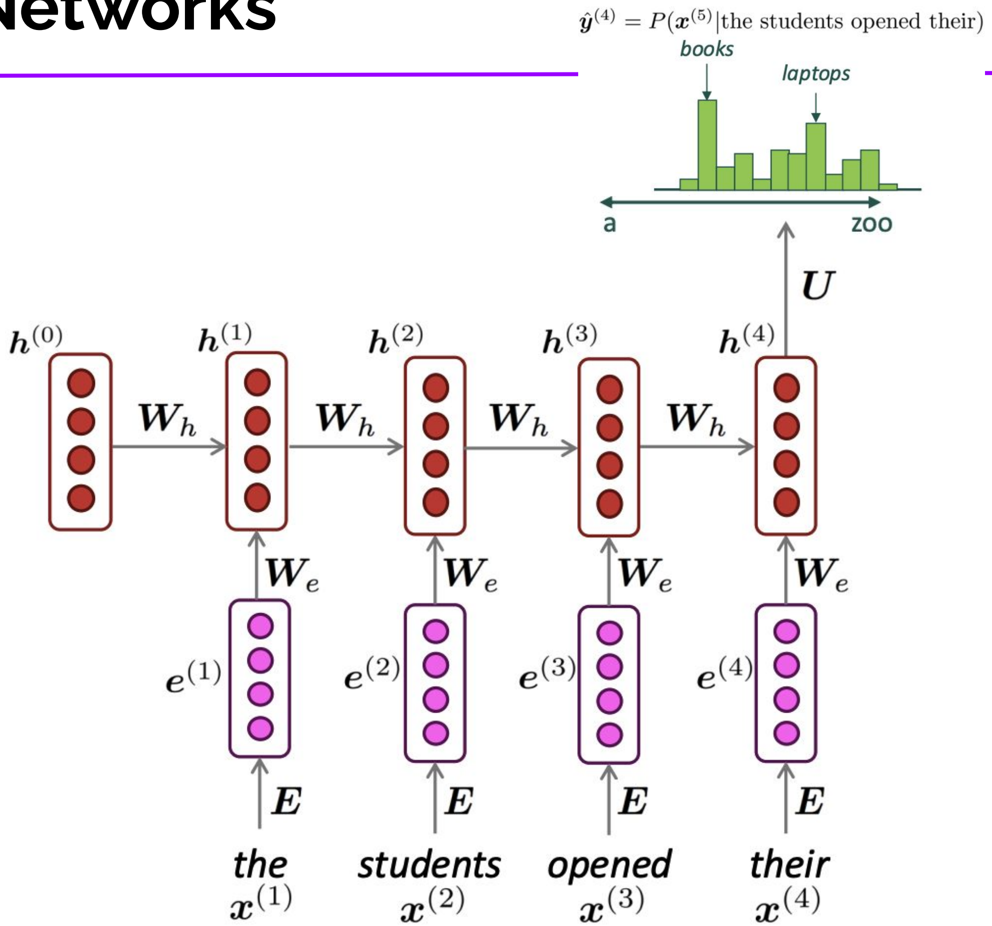
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

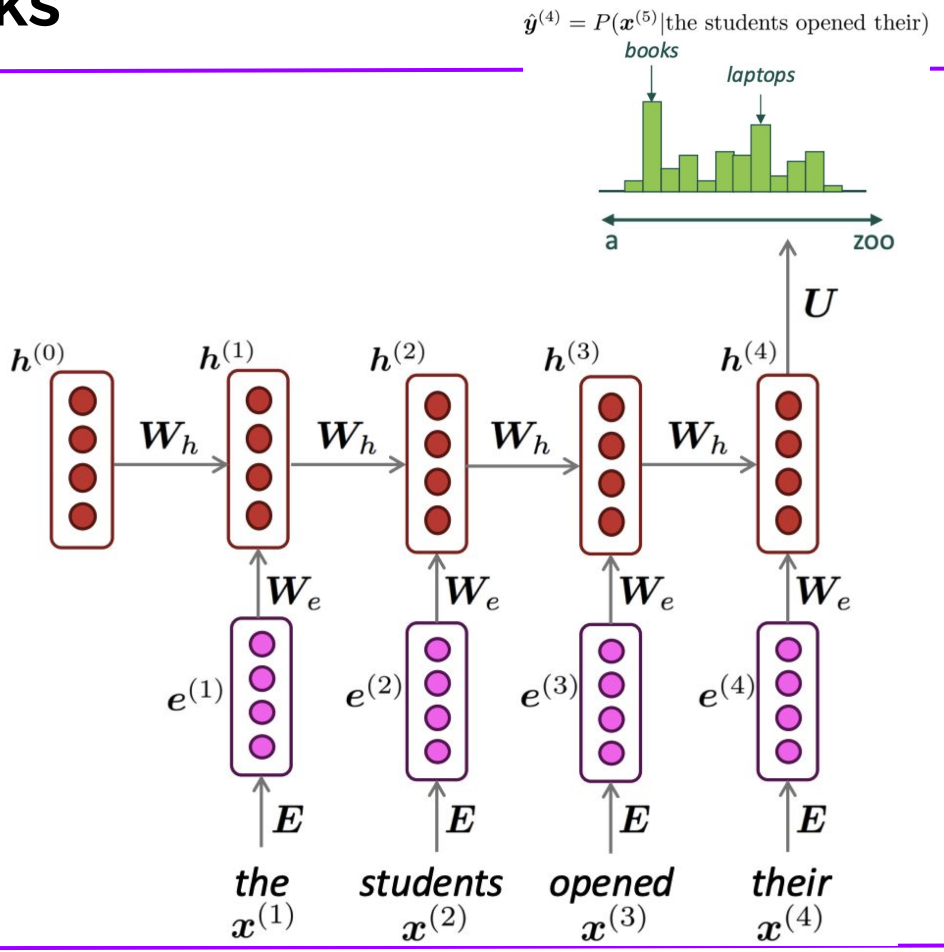
words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$



Recurrent Neural Networks

- Advantages
  - The process any length!
  - Can use information from previous steps
  - Model size does not increase for longer input context
  - Same weights applied on every timestep
- Disadvantages
  - Slow
  - Difficult to access information from many steps back



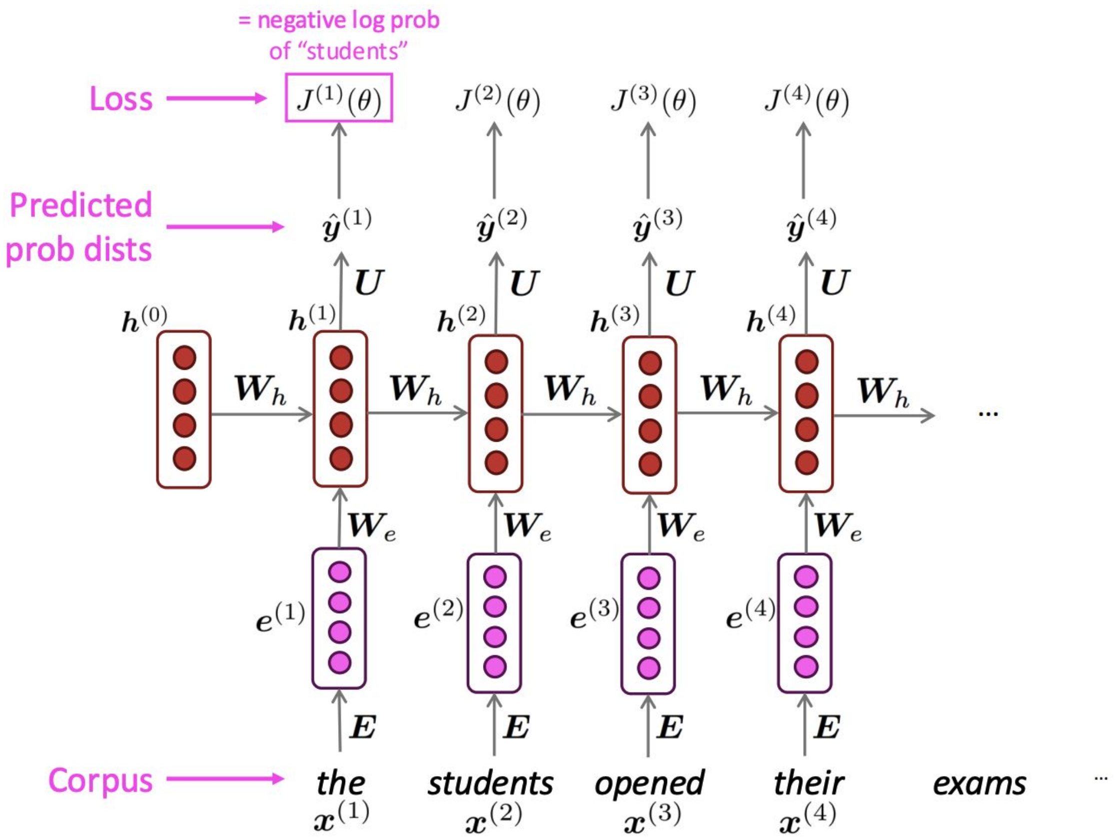
Recurrent Neural Networks

- Get a big corpus of text, i.e., sequence of $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$
- Feed into RNN, compute output distribution $\hat{\boldsymbol{y}}^{(t)}$
    - Predict probability dist of every word, given words so far
- Loss function is cross-entropy between predicted probability $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)}$

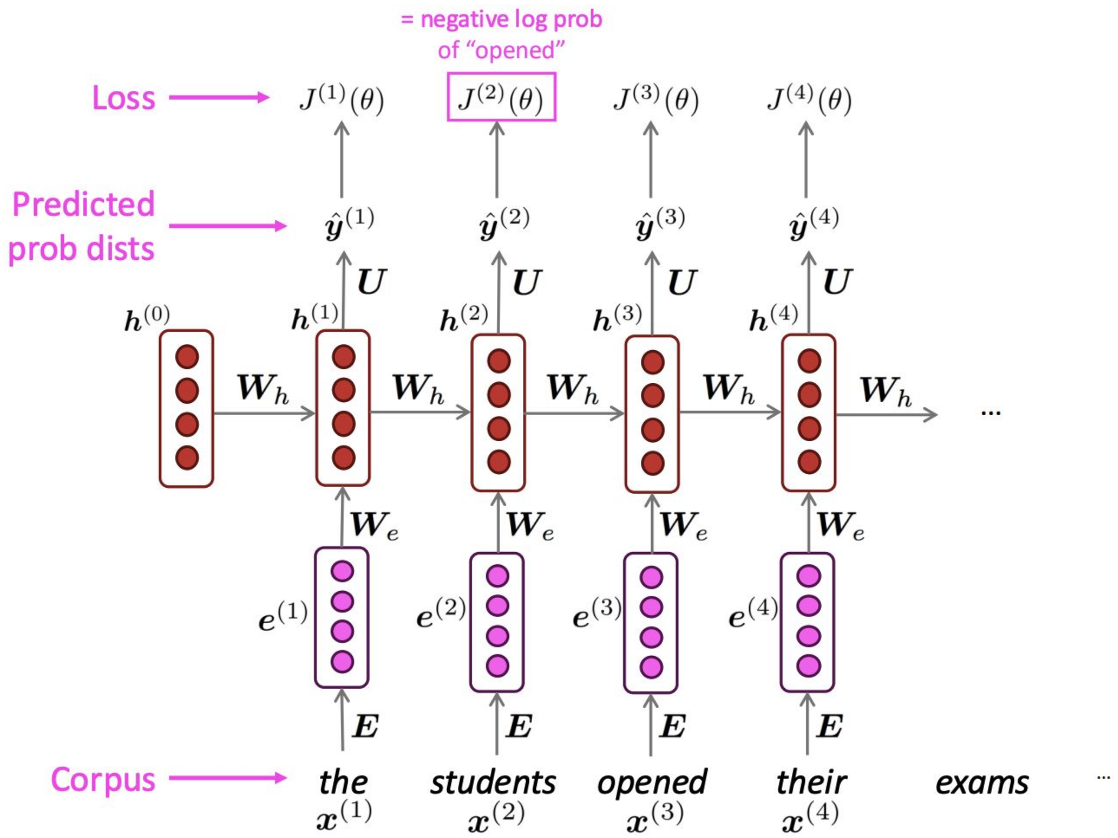
$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

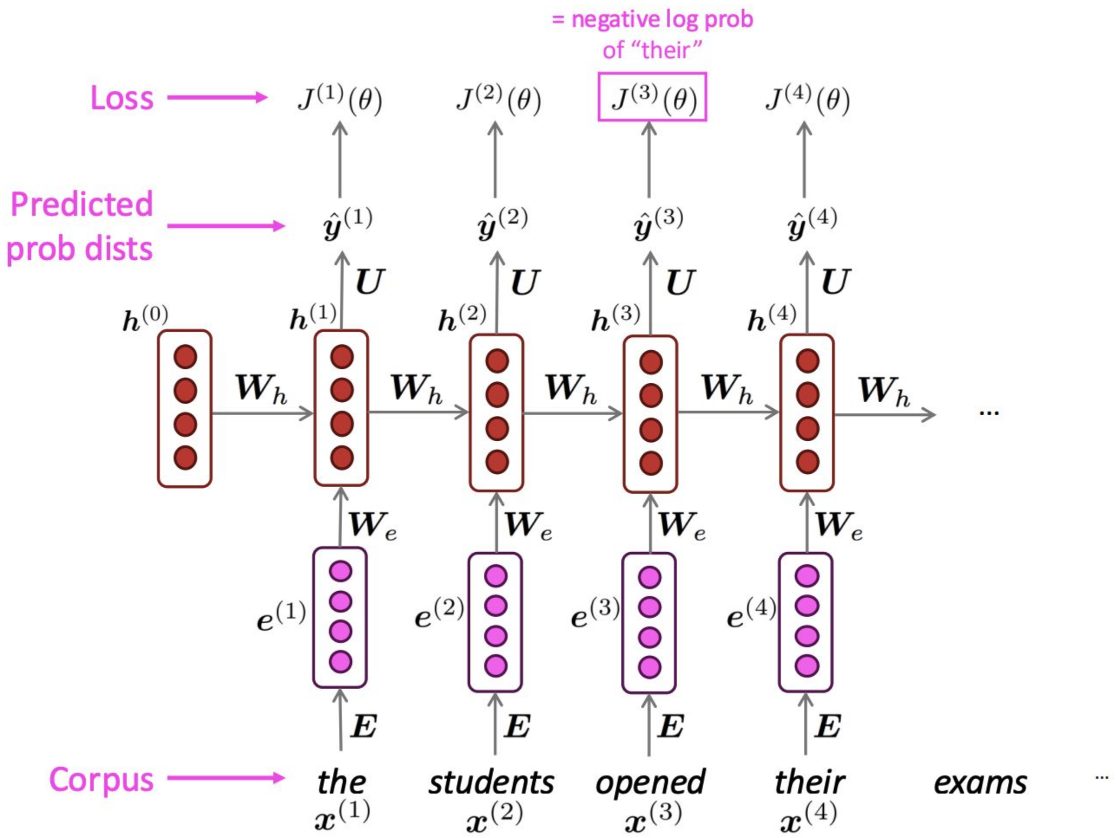
Recurrent Neural Networks



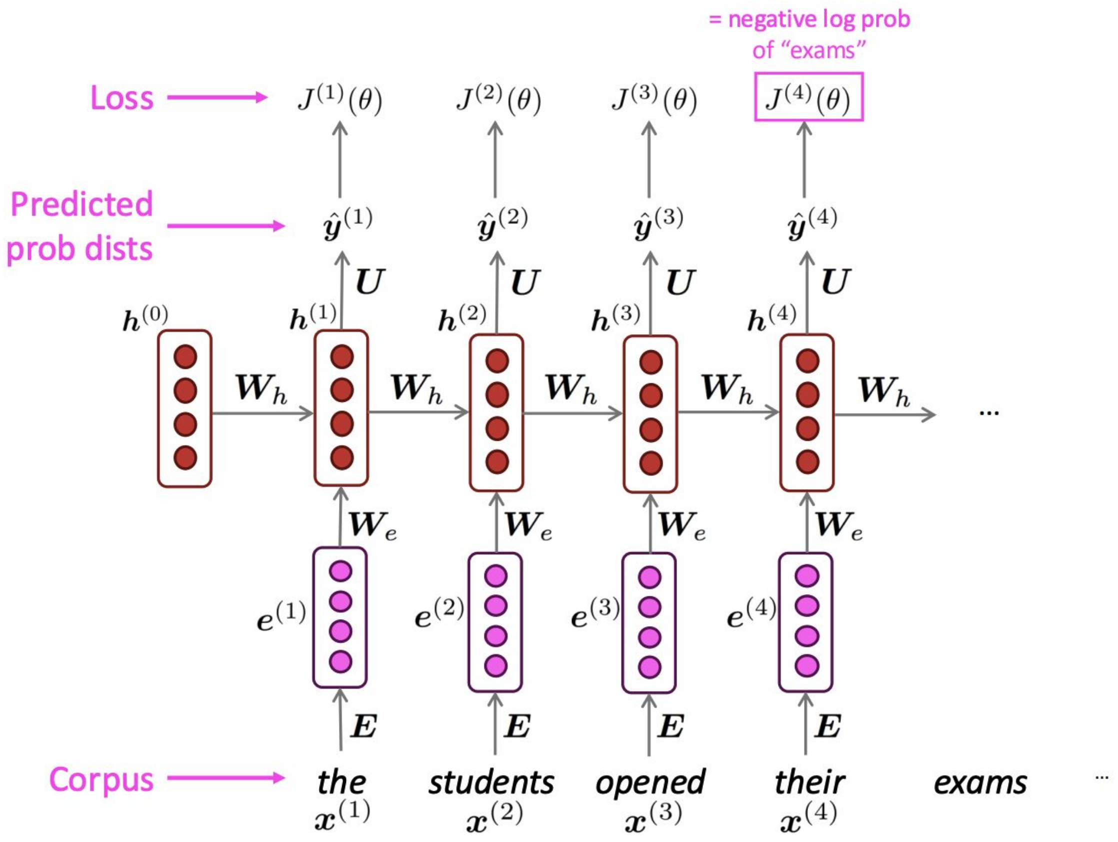
Recurrent Neural Networks



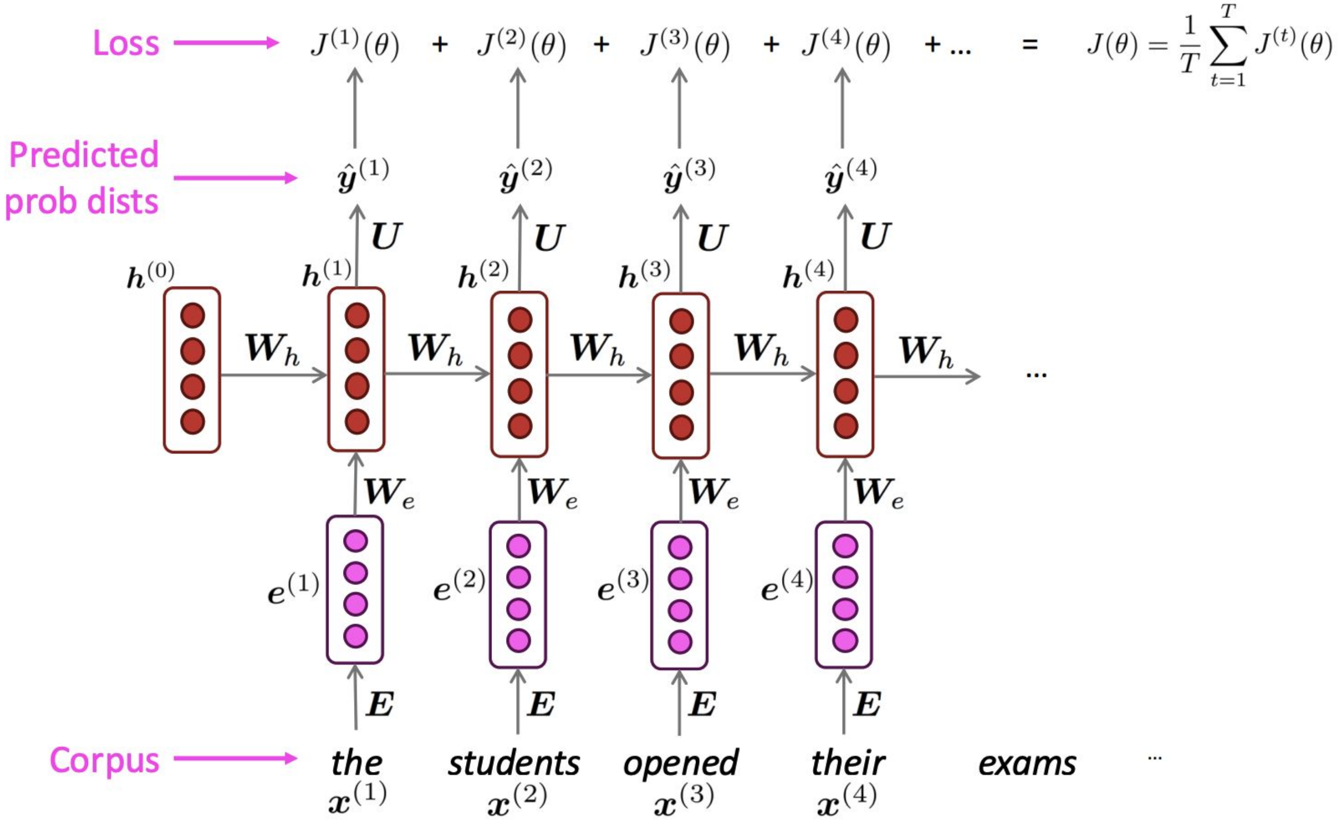
Recurrent Neural Networks



Recurrent Neural Networks

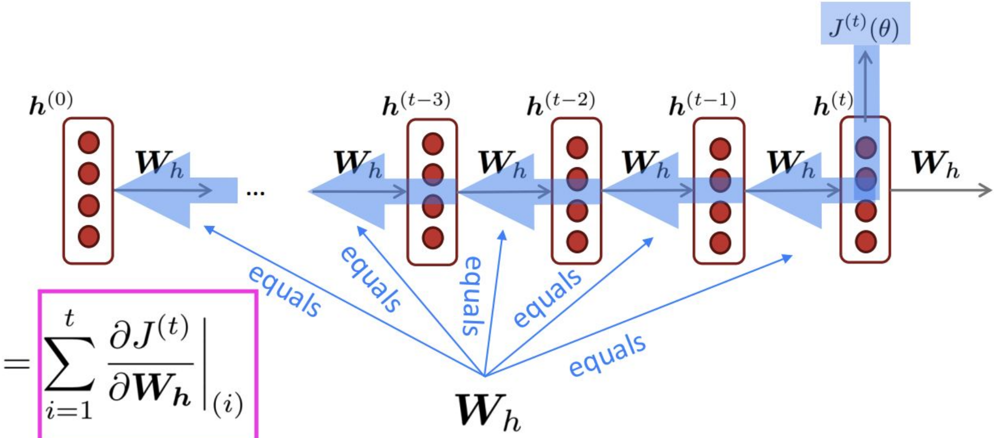


Recurrent Neural Networks



Recurrent Neural Networks

- Backpropagation through time



$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \boxed{\sum_{i=1}^{t} \left.\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\right|_{(i)}}$$

Question: How do we calculate this?

Answer: Backpropagate over timesteps $i = t, \ldots, 0$, summing gradients as you go. This algorithm is called **"backpropagation through time"** [Werbos, P.G., 1988, *Neural Networks* **1**, and others]

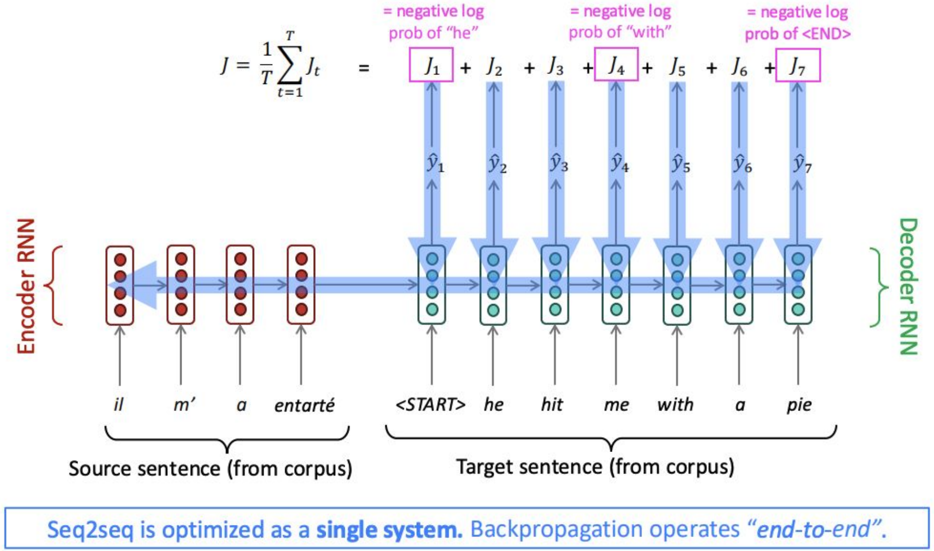
Apply the multivariable chain rule:

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \left.\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\right|_{(i)} \underbrace{\boxed{\frac{\partial \boldsymbol{W}_h|_{(i)}}{\partial \boldsymbol{W}_h}}}_{=1}$$

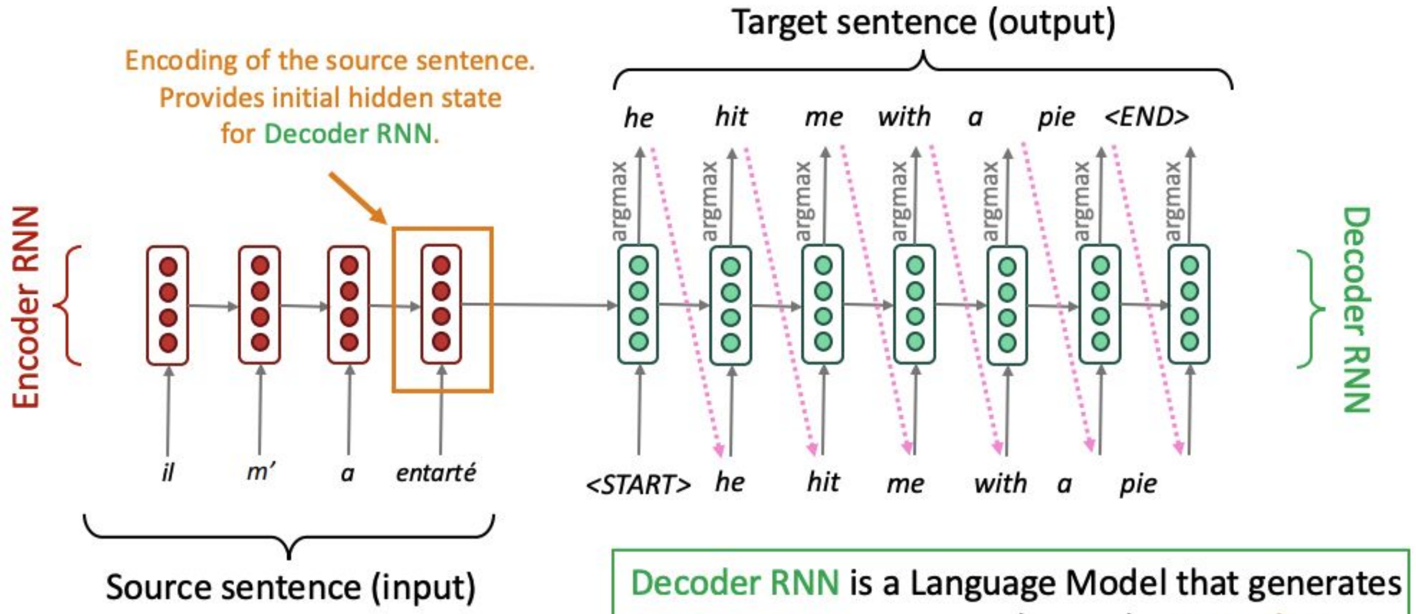
$$= \sum_{i=1}^{t} \left.\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\right|_{(i)}$$

Sequence 2 Sequence Modeling Using RNNs

- The general notion here is an encoder-decoder model
  - One neural network takes input and produces a neural representation
  - Another network produces output based on that neural representation
- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization
  - Dialogue
  - Code generation
  - Translation



Neural Machine Translation using RNNs

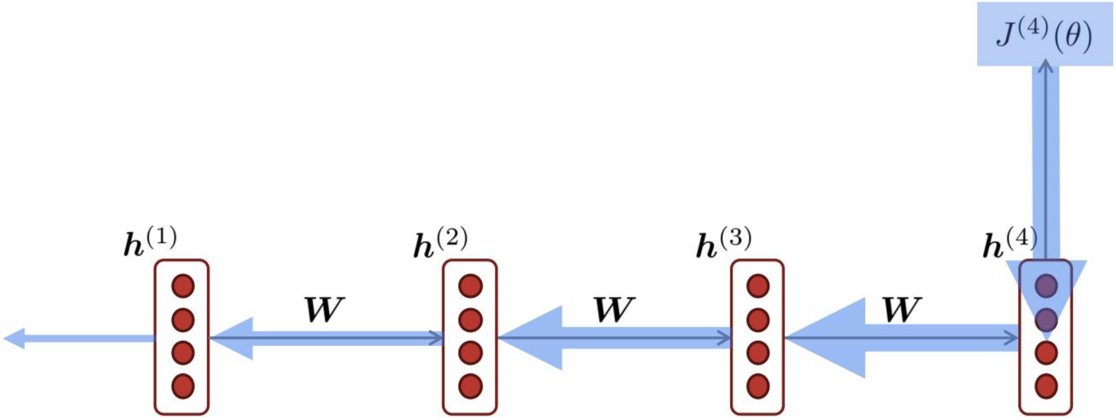


Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Note: This diagram shows **test time** behavior: decoder output is fed in ······➝ as next step's input

Encoder RNN produces an encoding of the source sentence.

Vanishing Gradients in RNNs

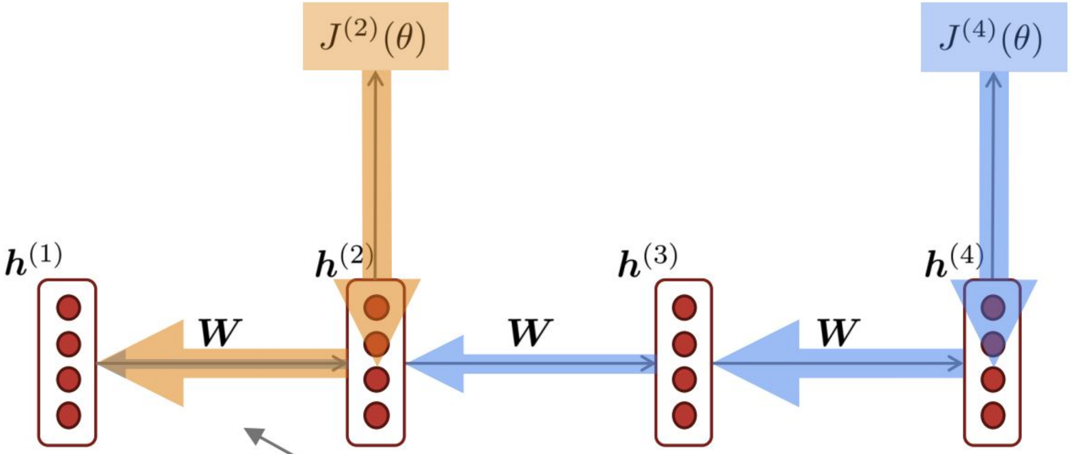


$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \boxed{\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

What happens if these are small?

Vanishing gradient problem:
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

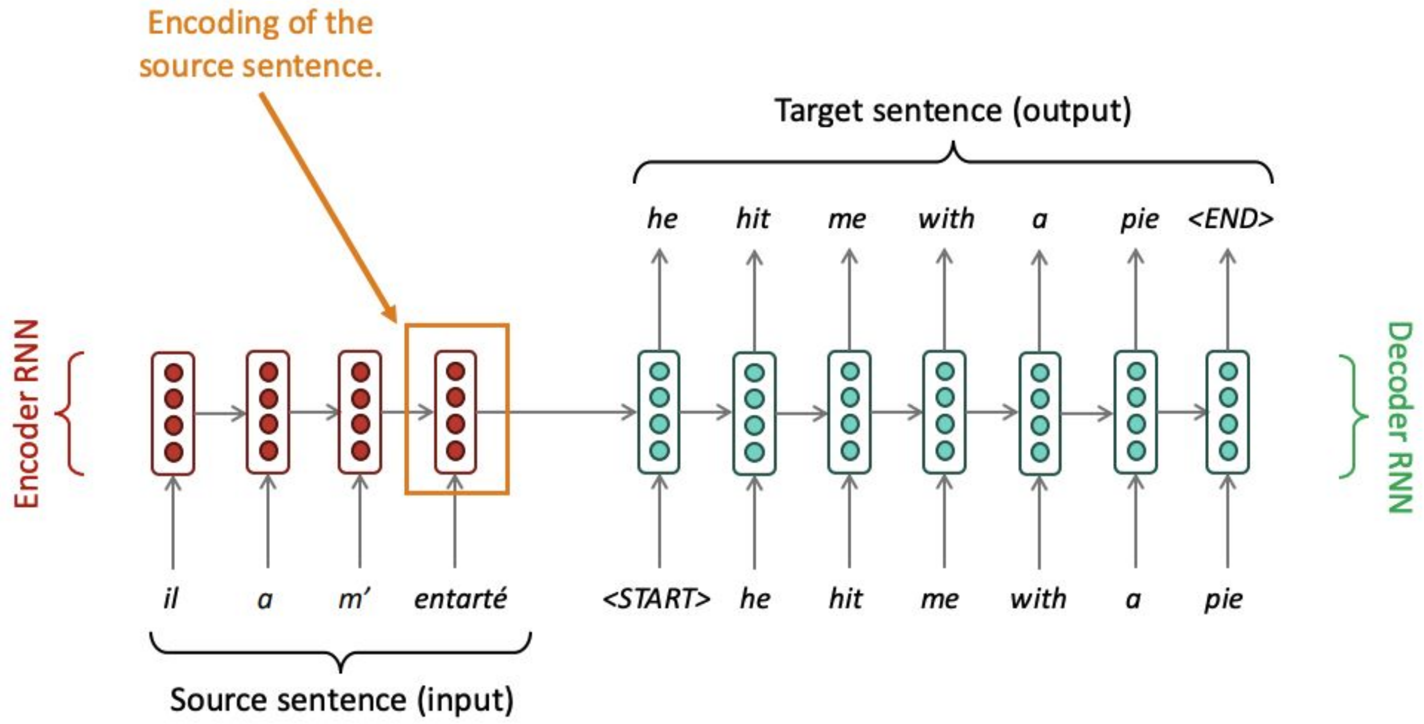
Vanishing Gradients in RNNs



Gradient signal from far away is lost because it's much smaller than gradient signal from close-by.

So, model weights are updated only with respect to near effects, not long-term effects.

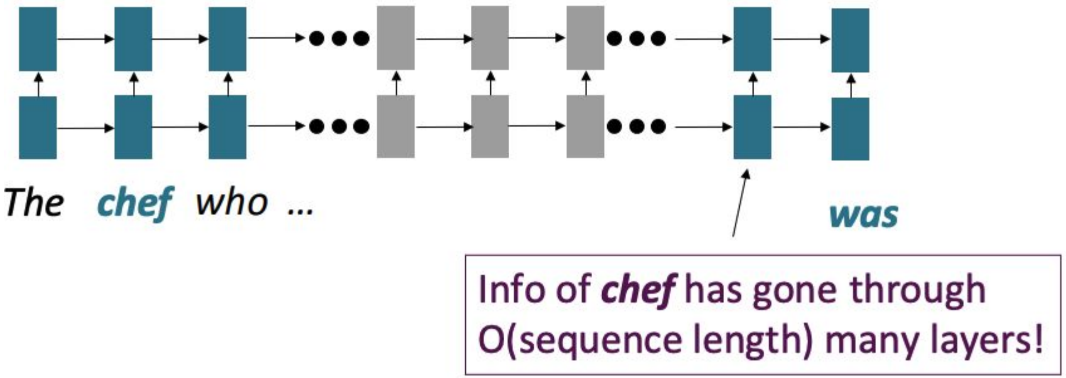
Bottleneck Problem in RNNs



Problems with this architecture?

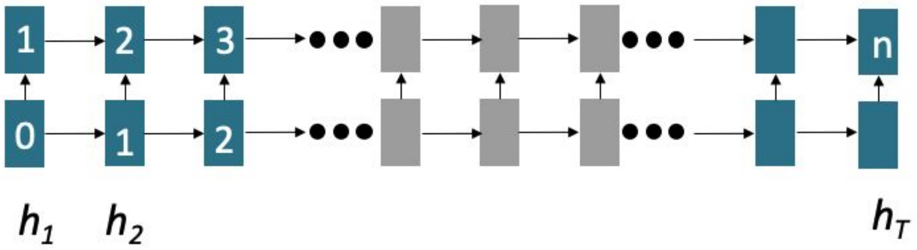
Lack of Parallelizability in RNNs

- O(sequence length) steps for distant word pairs to interact means:
  - Hard to learn long-distance dependencies (because gradient problems!)
  - Linear order of words is "baked in"; we already know linear order isn't the right way to think about sentences...



Lack of Parallelizability in RNNs

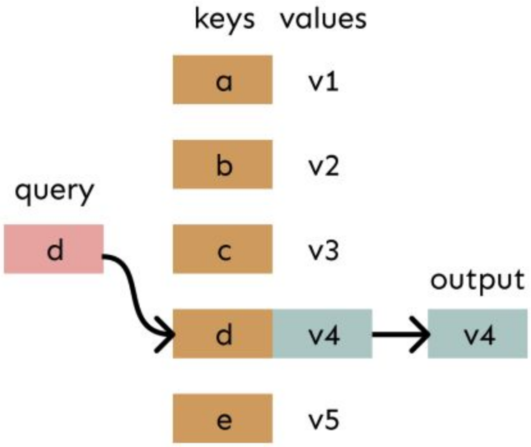
- Forward and backward passes have O(sequence length) non parallelizable operations
- GPUs can perform a bunch of independent operations at once!
- BUT! future RNN hidden states can't be computed in full before past RNN hidden states have been computed

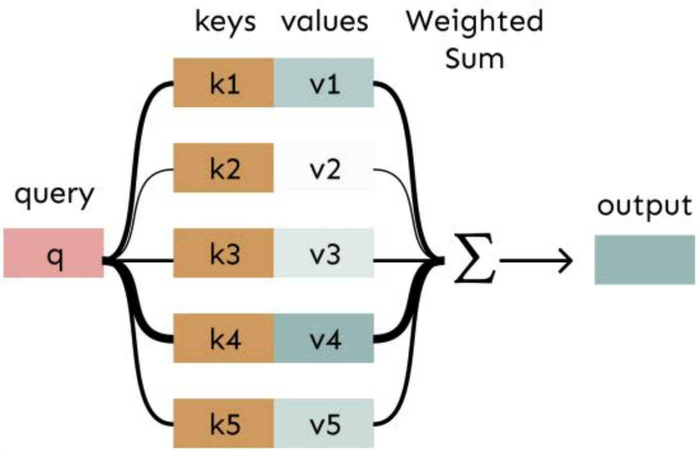


Numbers indicate min # of steps before a state can be computed

Attention is a solution!

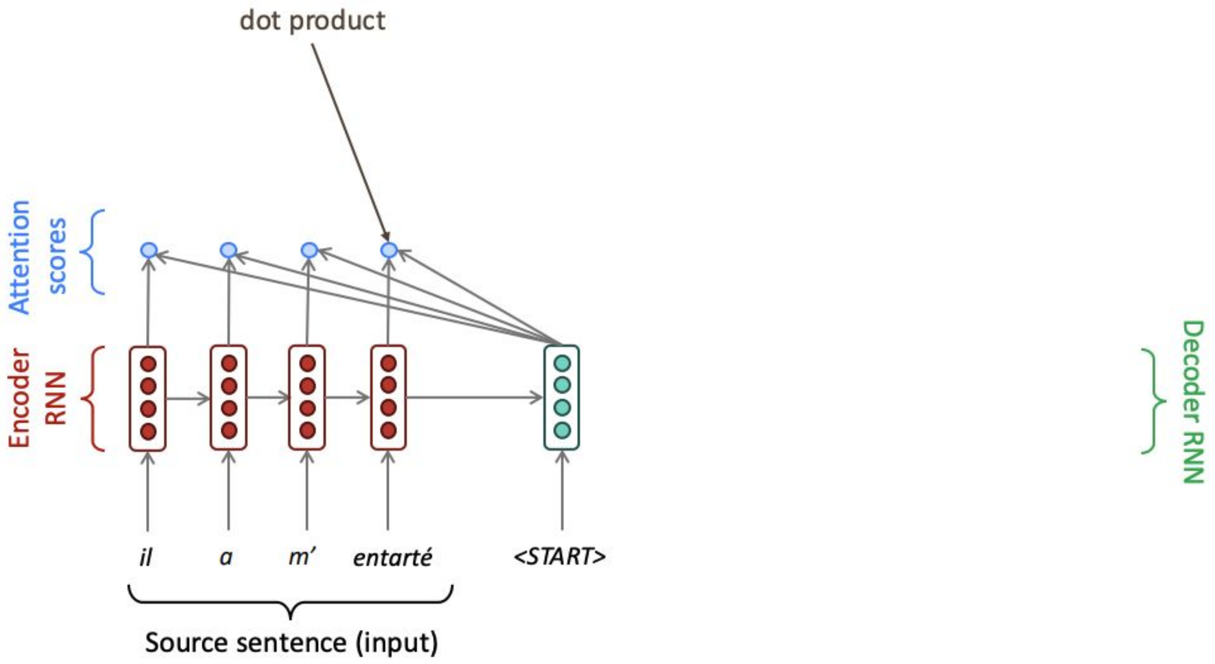
- Attention provides a solution to the bottleneck problem!
- Core idea: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence!
- In attention, the query matches all keys softly, to a weight between 0 and 1. The key's values are multiplied by the weights and summed!



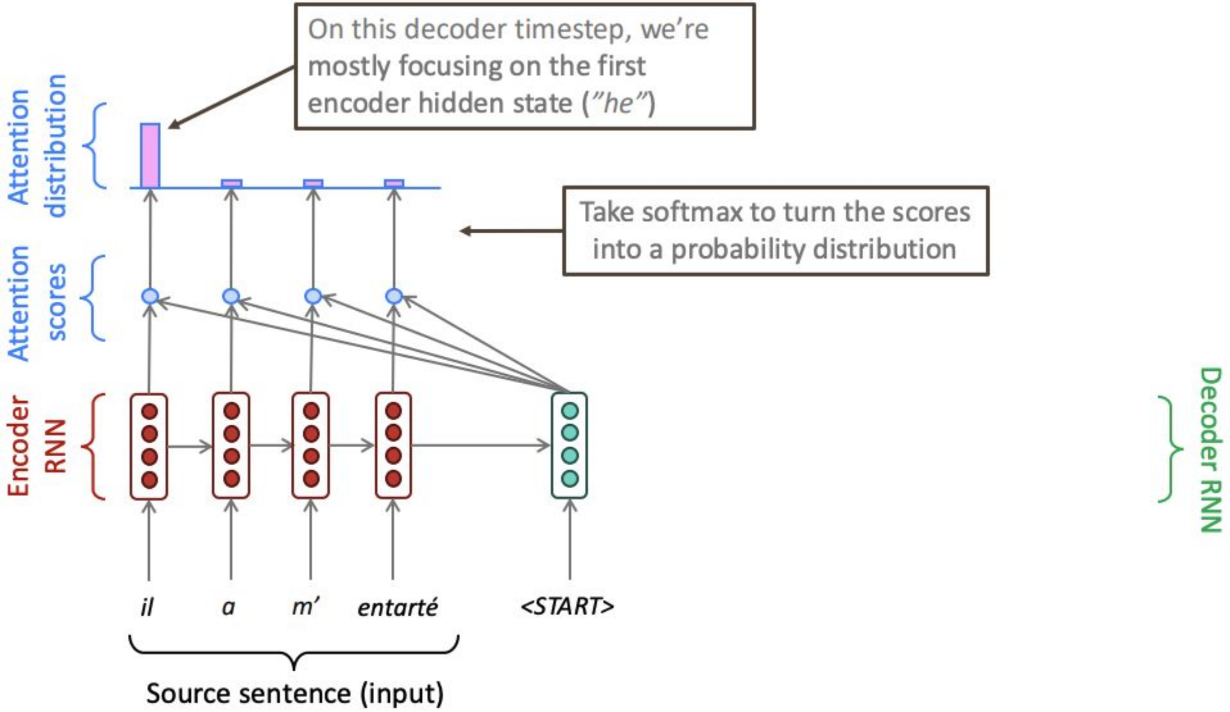


Attention in RNNs

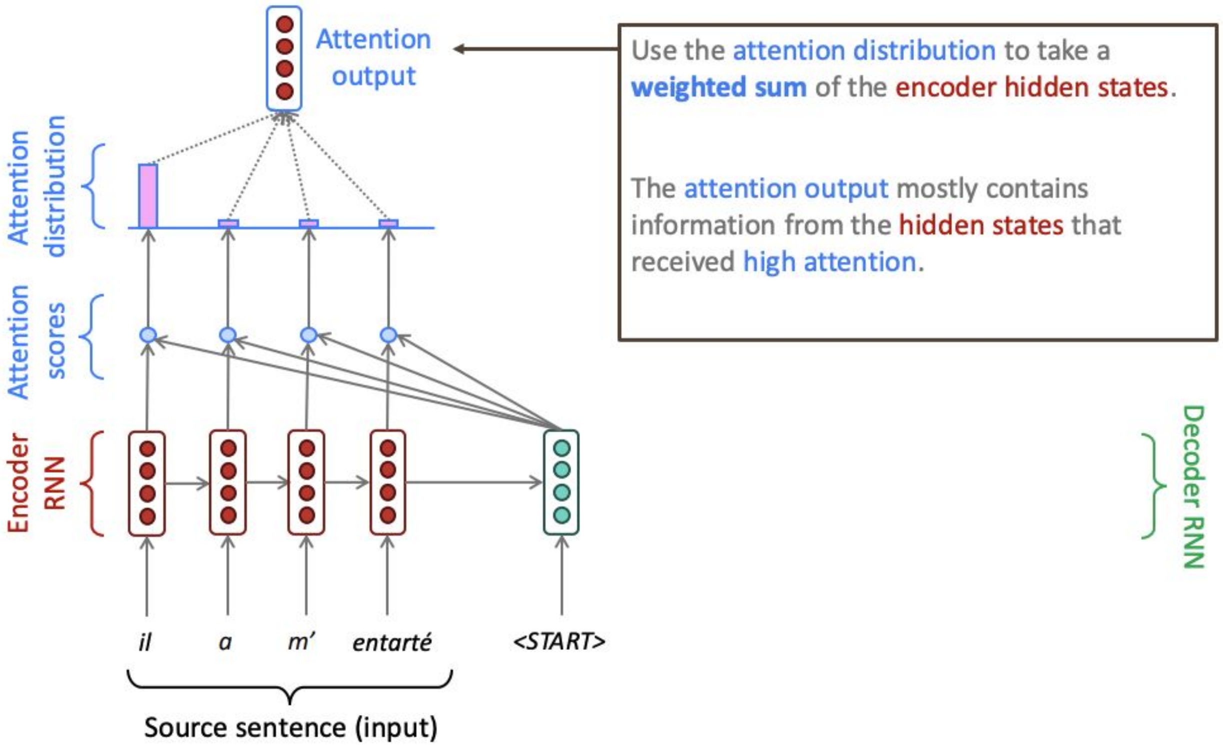
- On each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence.



Attention in RNNs



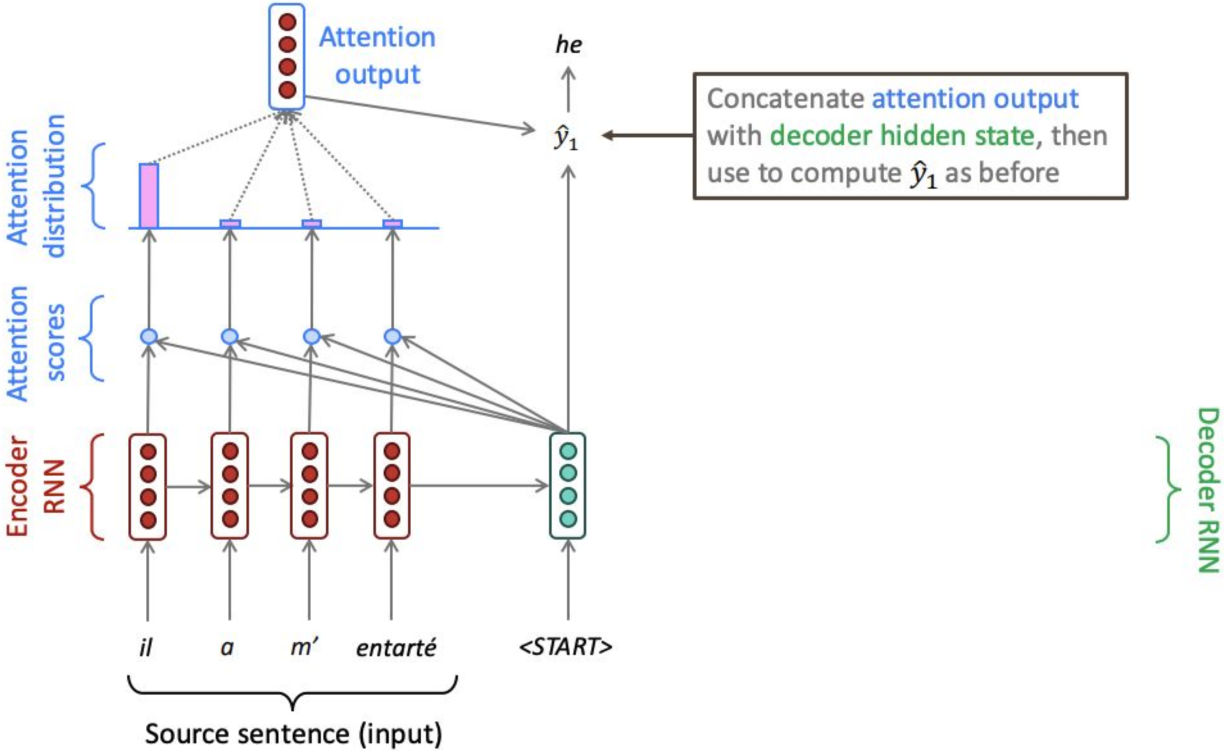
Attention in RNNs



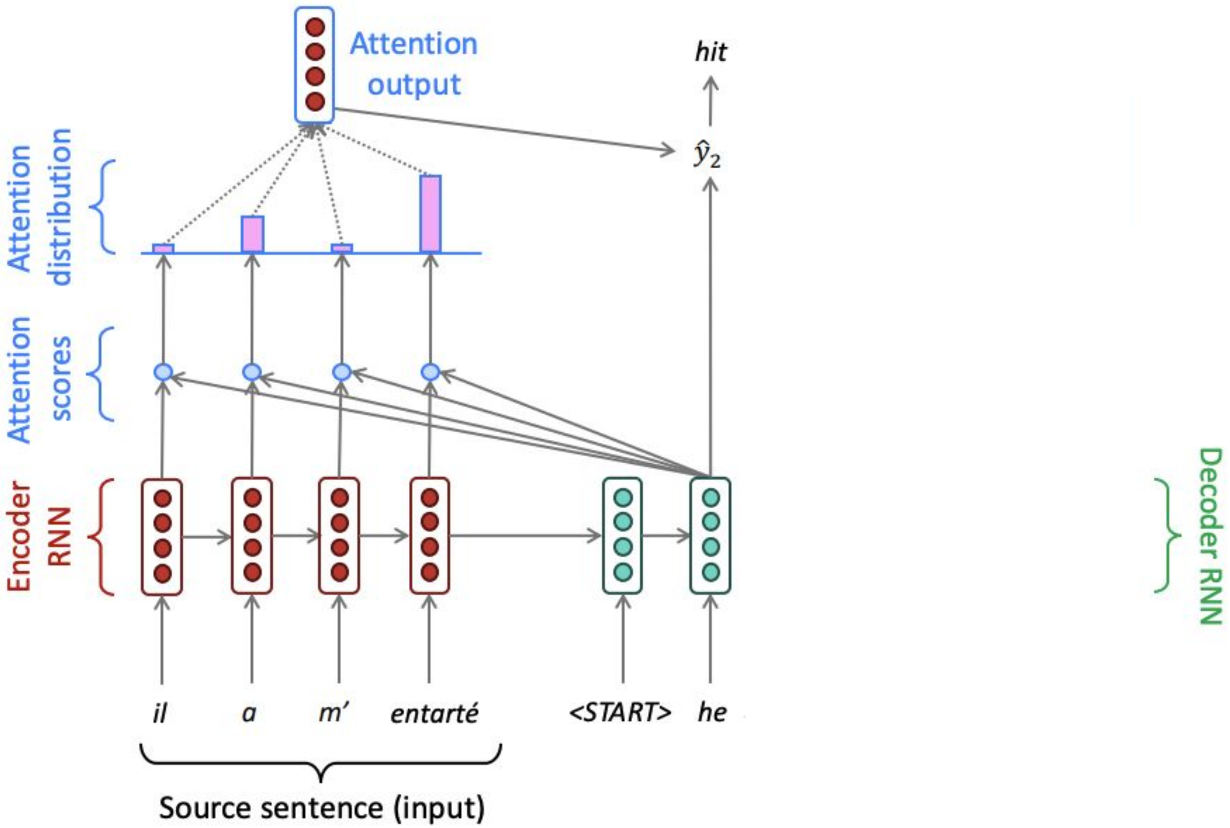
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information from the hidden states that received high attention.

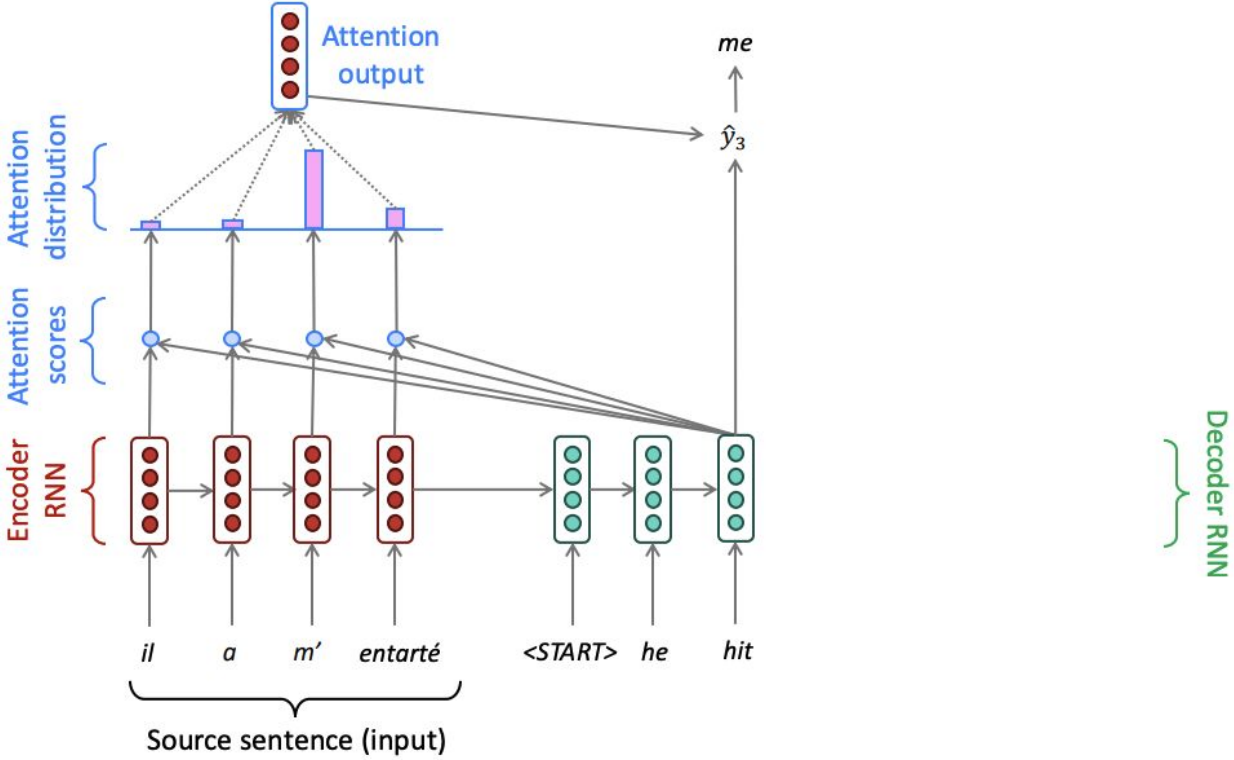
Attention in RNNs



Attention in RNNs



Attention in RNNs



Is Recurrent Necessary at All?

- Abstractly: Attention is a way to pass information from a sequence (x) to a neural network input. (h_t)
- This is also exactly what RNNs are used for – to pass information!
- Can we just get rid of the RNN entirely? Maybe attention is just a better way to pass information!
- The building block we need is self Attention!
- So far we saw cross-attention!