Introduction to Language Models

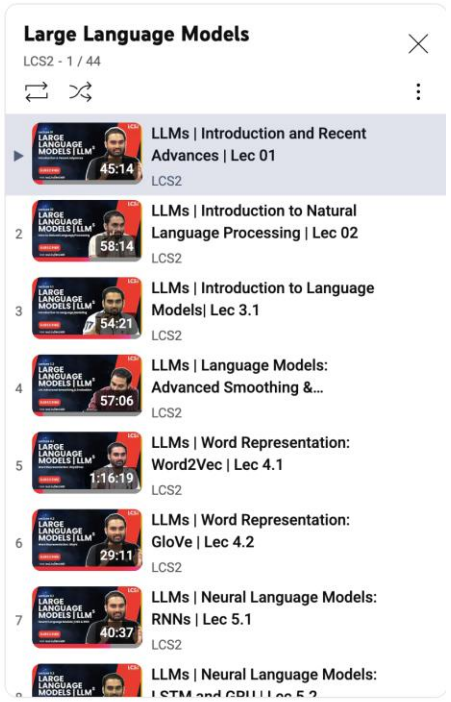
Tanmoy Chakraborty
Associate Professor, IIT Delhi
https://tanmoychak.com/

LCS
LABORATORY FOR
COMPUTATIONAL SOCIAL SYSTEMS

REMINDER

Last year's (2024) offering

LLMs | Introduction and Recent Advances | Lec 01

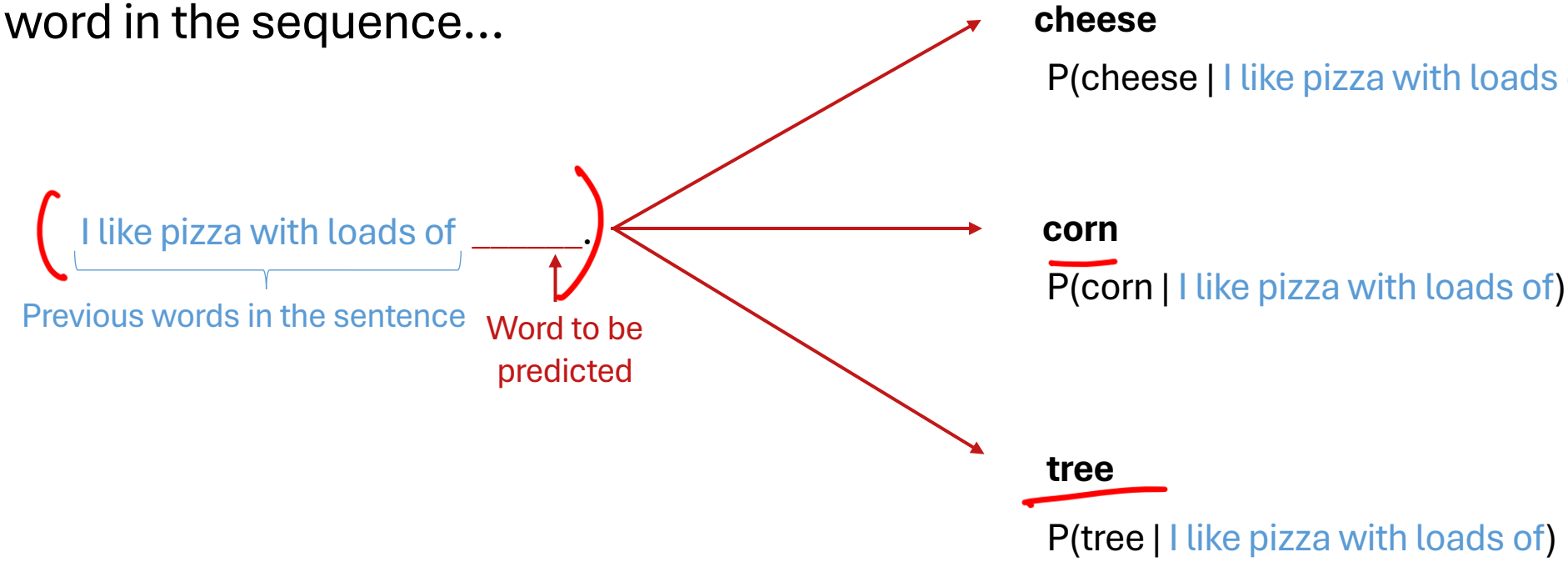


https://lcs2.in/llm2401

You are advised to study the **first 10 lectures (till Lec 6.1)** of the previous year's course playlist before the **next class on August 4.** Otherwise, you will not be able to follow. Here's the link to the playlist:

Next Word Prediction

Guess the next word in the sequence...



P(cheese| I like pizza with loads of) > P(corn| I like pizza with loads of) >> P(tree| I like pizza with loads of)

Probabilistic Language Models

- **Goal:** Calculate the probability of a sentence or sequence consisting of *n* words

$$P(W) = P(w_1, w_2, w_3, \dots, w_n)$$

or

- **Related Task:** Calculate the probability of the next word conditioned on the preceding words

$$P(w_6 \mid w_1, w_2, w_3, w_4, w_5)$$

A model that calculates either of these is referred to as a **Language Model (LM).**

Estimate Conditional Probabilities

$$P(\text{begun} \mid \text{The monsoon season has}) = \frac{\text{Count (The monsoon season has begun)}}{\text{Count (The monsoon season has)}}$$

- **Problem:** Enough data is not available to get an accurate estimate of the above quantities.
- **Solution:** Markov Assumption

Markov Assumption

Every next state depends only the previous k states

- Chain Rule:

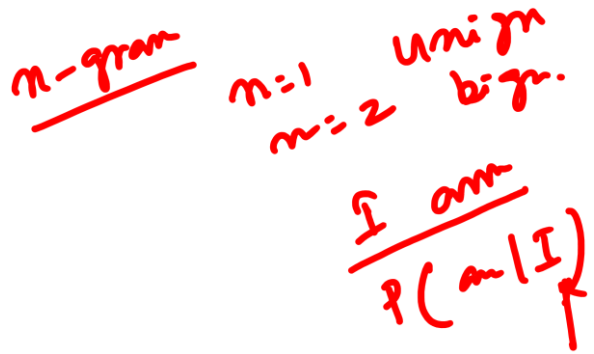
$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i | w_1 w_2 \ldots w_{i-1})$$

- Applying Markov Assumption we condition on only the preceding k words:

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i | w_{i-k} \ldots w_{i-1})$$

- Probabilistic Language Models exploit the **Chain Rule of Probability** and **Markov Assumption** to build a probability distribution over sequences of words.

N-gram Language Models



- Let's consider the following conditional probability:

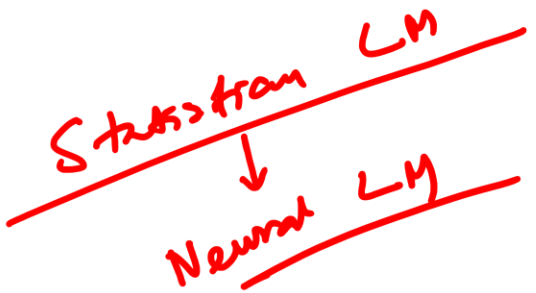
  P(begun | the monsoon season has)

- An **N-gram model** considers only the preceding **N −1 words**.
  - Unigram: P(begun)
  - Bigram: P(begun | the)
  - Trigram: P(begun | the monsoon)

Relation between Markov model and Language Model:

An N-gram Language Model ≡ (N −1) order Markov Model

Limitation of N-gram Language Models

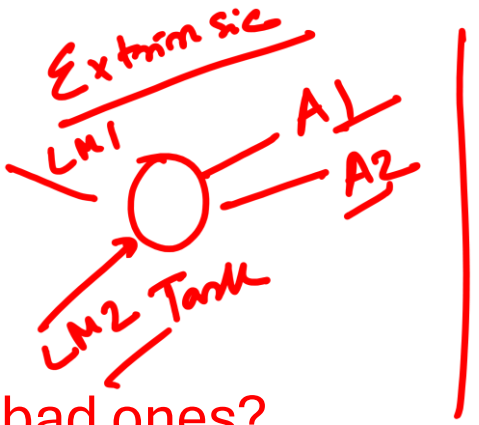


- An insufficient model of language since they are **not effective in capturing long-range dependencies present in language**.
  - Example:

The **project**, which he had been working on for months, was finally **approved** by the committee.

The above example highlights the long-distance dependency between “project” and “approved”, where the context provided by earlier words affects the interpretation of later parts of the sentence.

Evaluation of a Language Model



- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences than “ungrammatical” or “rarely observed” sentences

- Terminologies:
  - We optimize the parameters of our model based on data from a **training set**.
  - We assess the model's performance on unseen **test data** that is disjoint from the training data.
  - An evaluation metric provides a measure of the performance of our model on the test set.

Extrinsic Evaluation

- Measure the effectiveness of a language model by **testing their performance on different downstream NLP tasks**, such as machine translation, text classification, speech recognition.

- Let us consider two different language models: A and B
  - Select a suitable evaluation metric to assess the performance of the language models based on the chosen task.
  - Obtain the evaluation scores for A and B
  - Compare the evaluation scores for A and B

Intrinsic Evaluation: Perplexity

Thus, for the sentence W, perplexity is:

$$PP(W) = P(w_1 w_2 \ldots wn)^{-\frac{1}{n}}$$

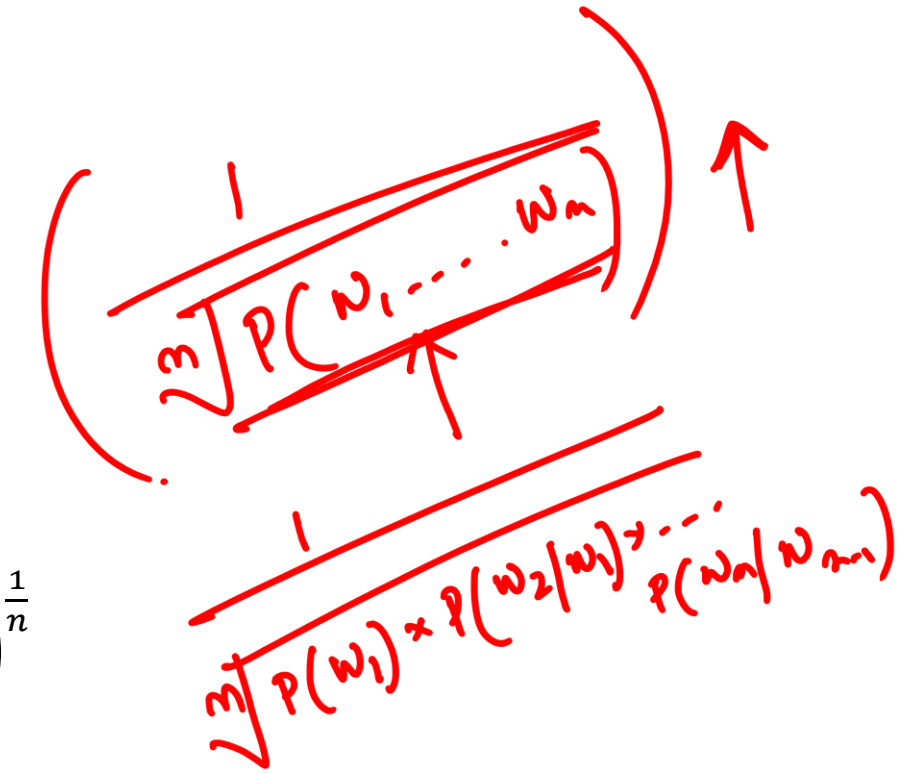
Applying Chain Rule:

$$PP(W) = \left(\prod \frac{1}{P(w_i | w_1 w_2 \ldots w_{i-1})}\right)^{\frac{1}{n}}$$

Applying Markov Assumption (n = 2), i.e. for bigram LM:

$$PP(W) = \left(\prod \frac{1}{P(w_i | w_{i-1})}\right)^{\frac{1}{n}}$$

Minimizing perplexity is the same as maximizing probability.

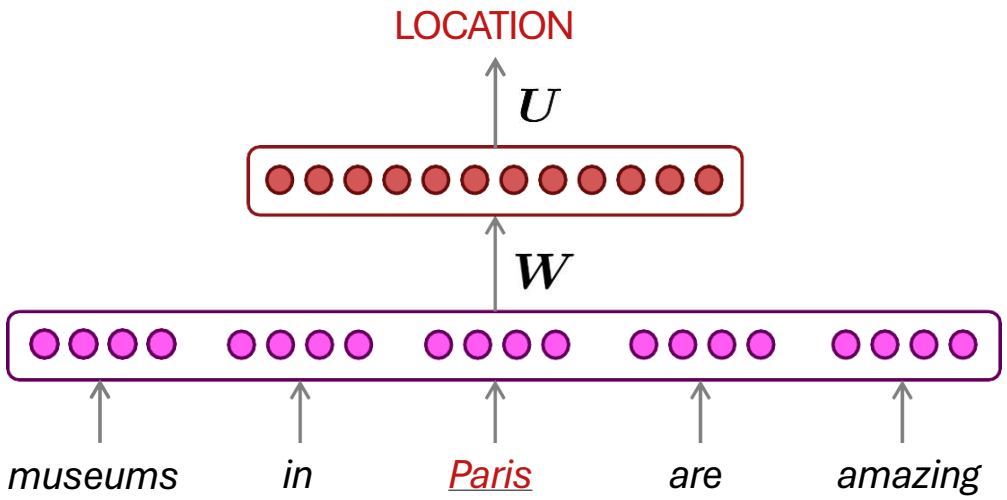


Neural Language Models

How to Build a *Neural* Language Model?

- Recall the Language Modeling task:
  - **Input:** sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$
  - **Output:** probability distribution of the next word $\boldsymbol{P}\left(\boldsymbol{x}^{(t+1)} \middle| \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)}\right)$
- How about a window-based neural model?

Example: NER Task



A Fixed-window Neural Language Model

output distribution

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

hidden layer

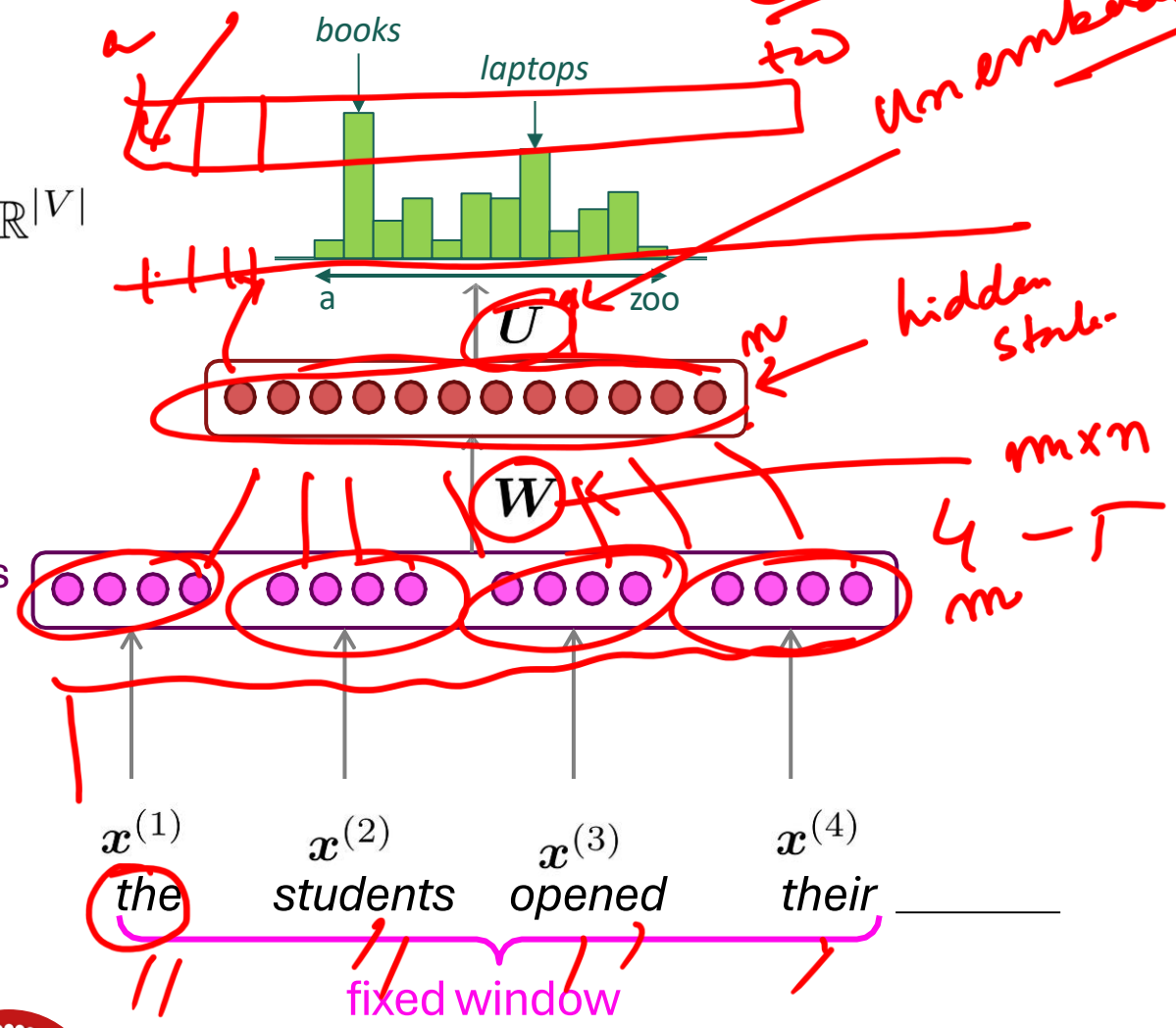
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



~~as the proctor started the clock~~

discard

A Fixed-window Neural Language Model

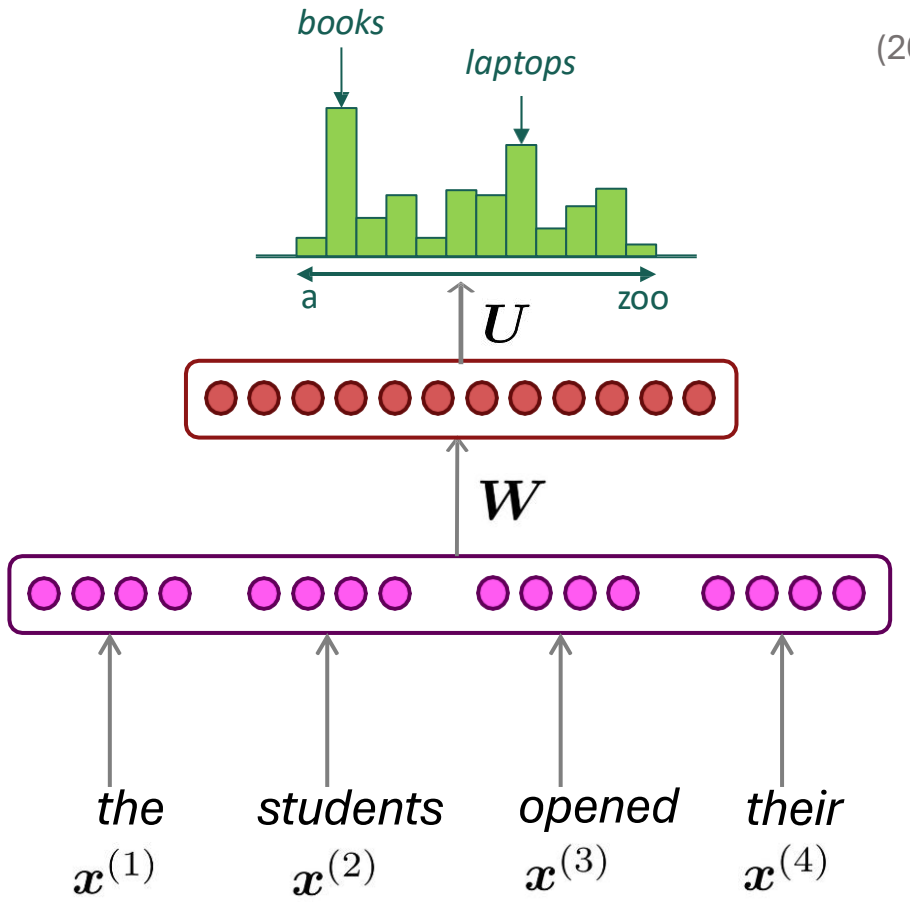
Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

Improvements over *n*-gram LM:

- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining problems:

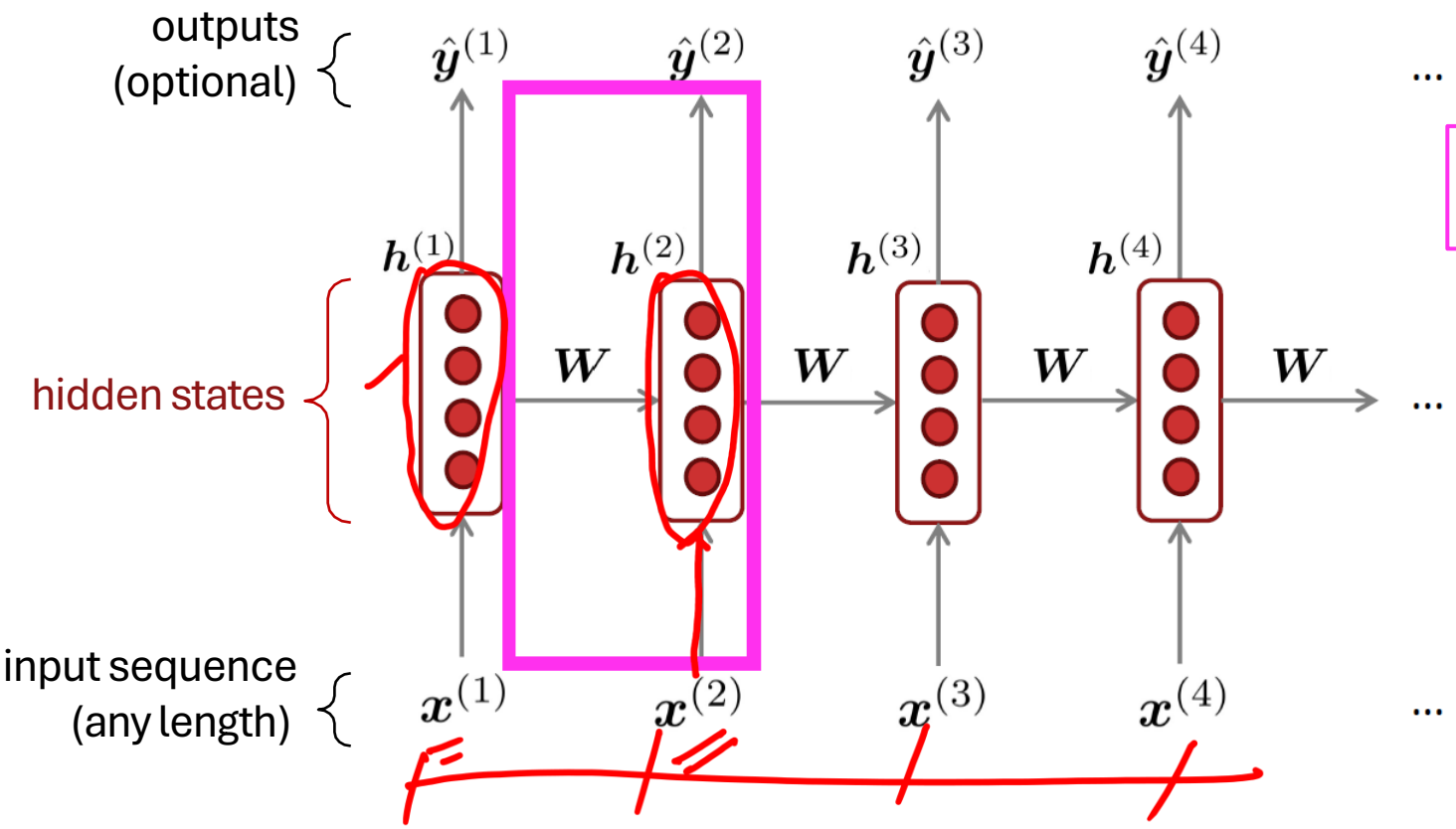
- Fixed window is too small
- Enlarging window enlarges W



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

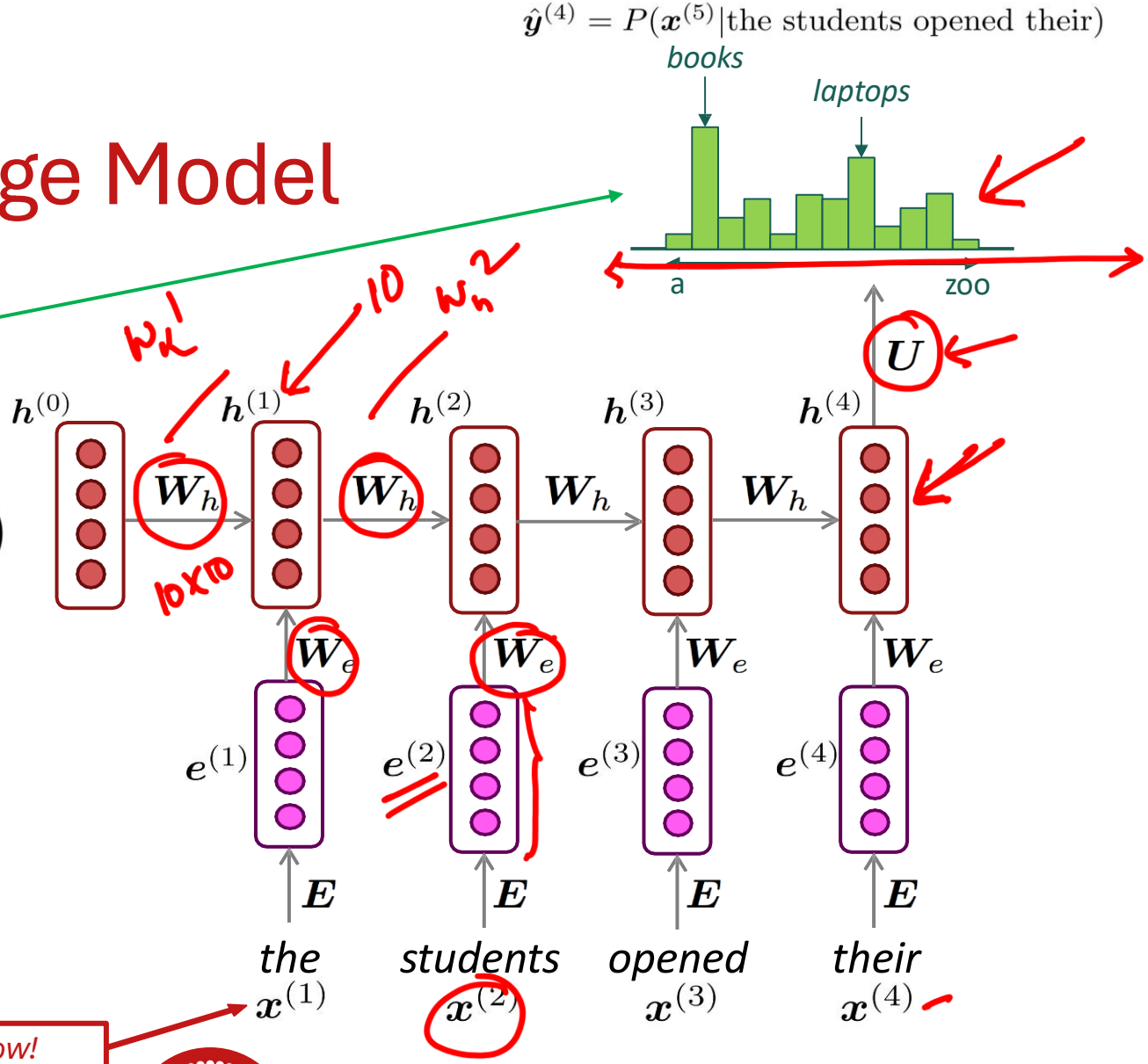
Recurrent Neural Networks (RNN)

$h^2 = f(x^2, h^1)$



Core idea: Apply the same weights *W repeatedly*

A Simple RNN Language Model



output distribution

$$\hat{y}^{(t)} = \text{softmax}\left(Uh^{(t)} + b_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_e e^{(t)} + b_1\right)$$

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

word embeddings

$$e^{(t)} = Ex^{(t)}$$

words / one-hot vectors

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

***Note**: this input sequence could be much longer now!*

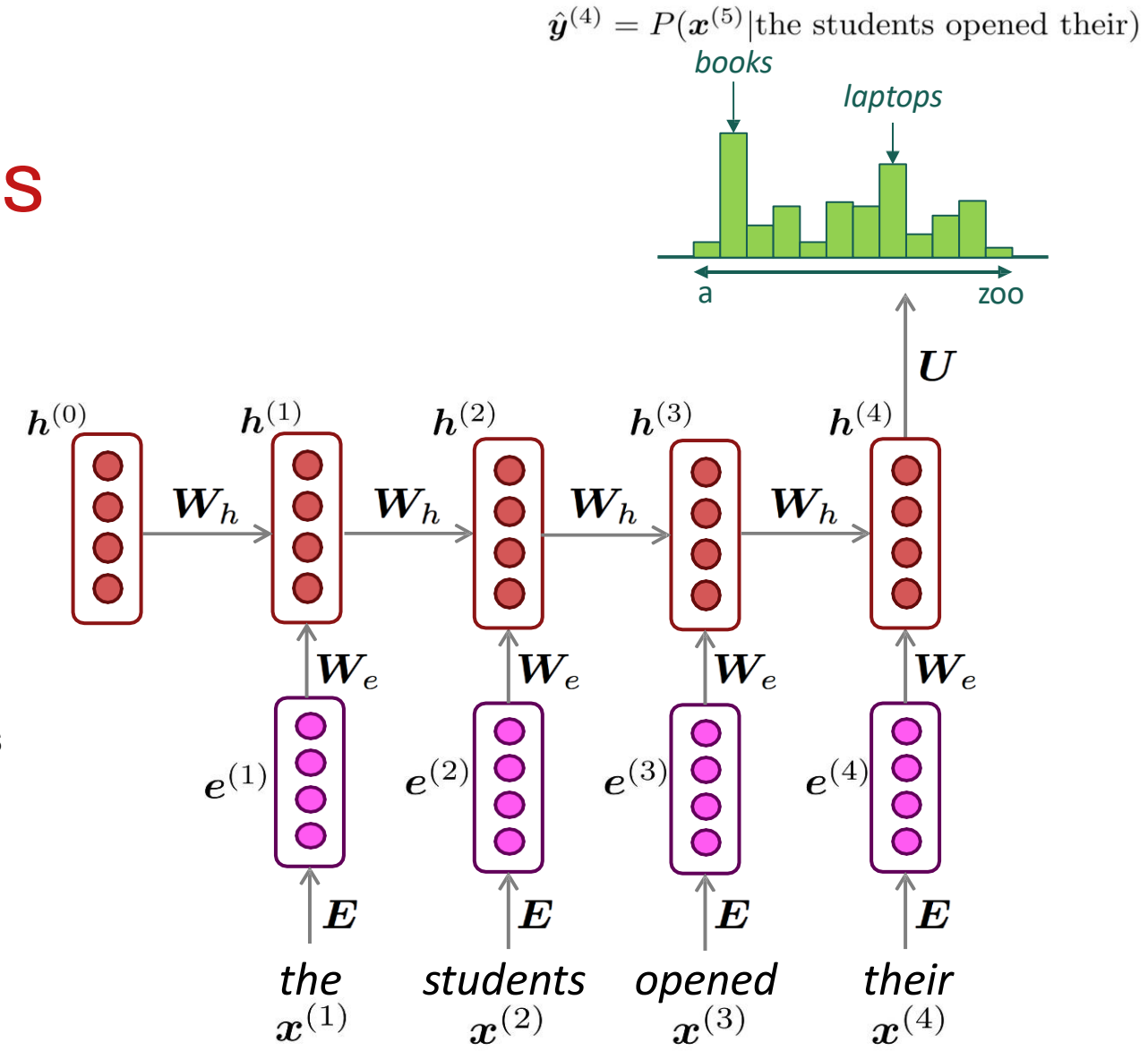
RNN Language Models

RNN Advantages:

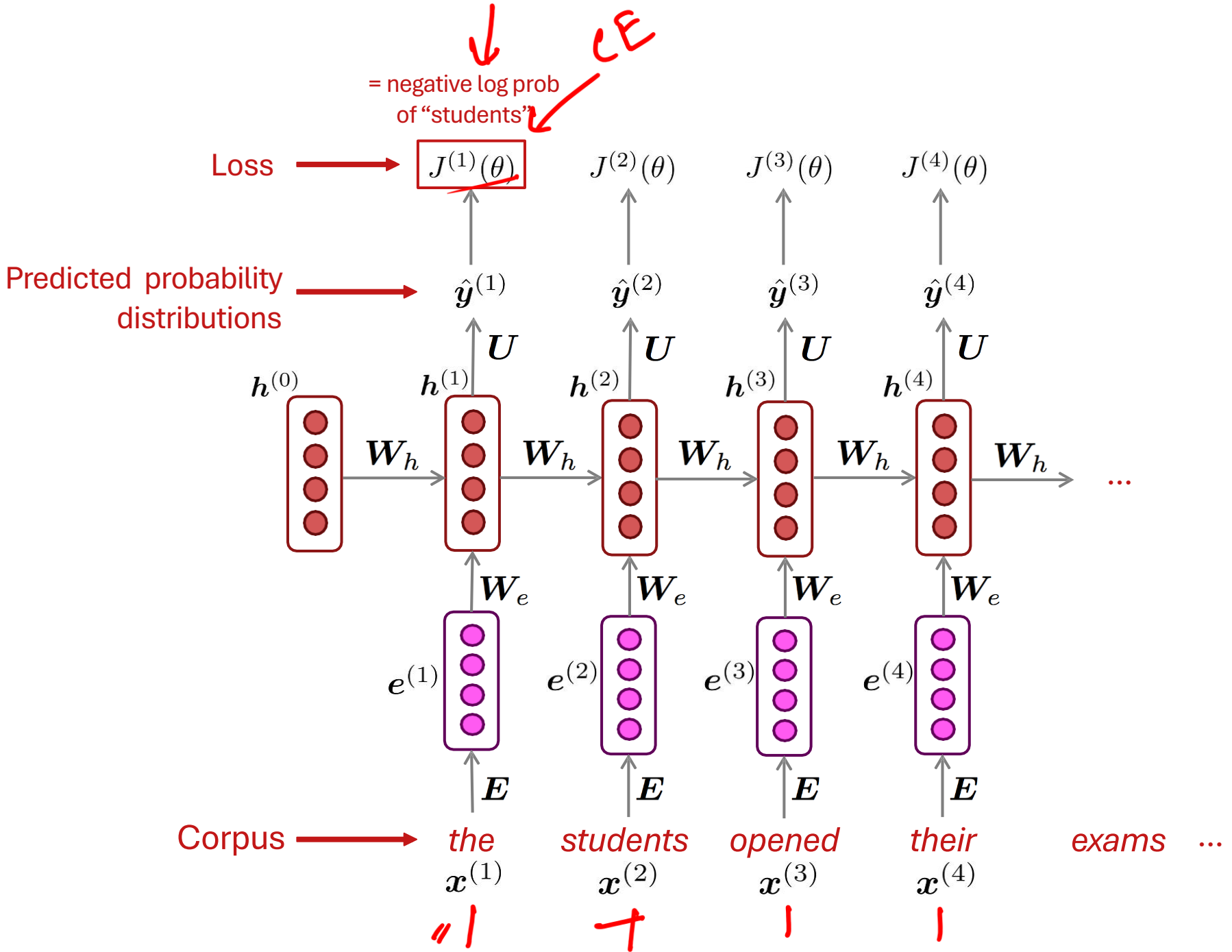
- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input context
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

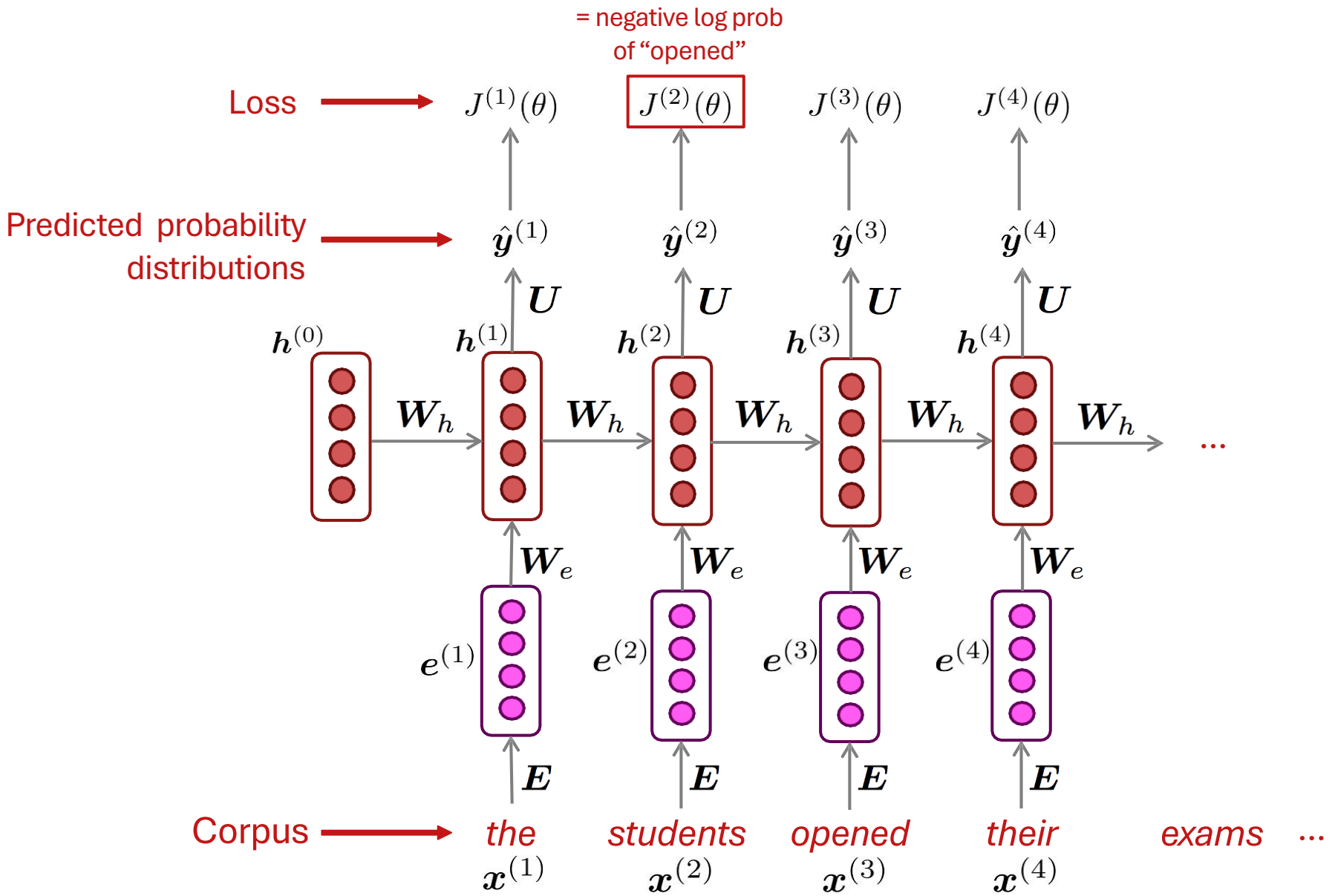
RNN Disadvantages:

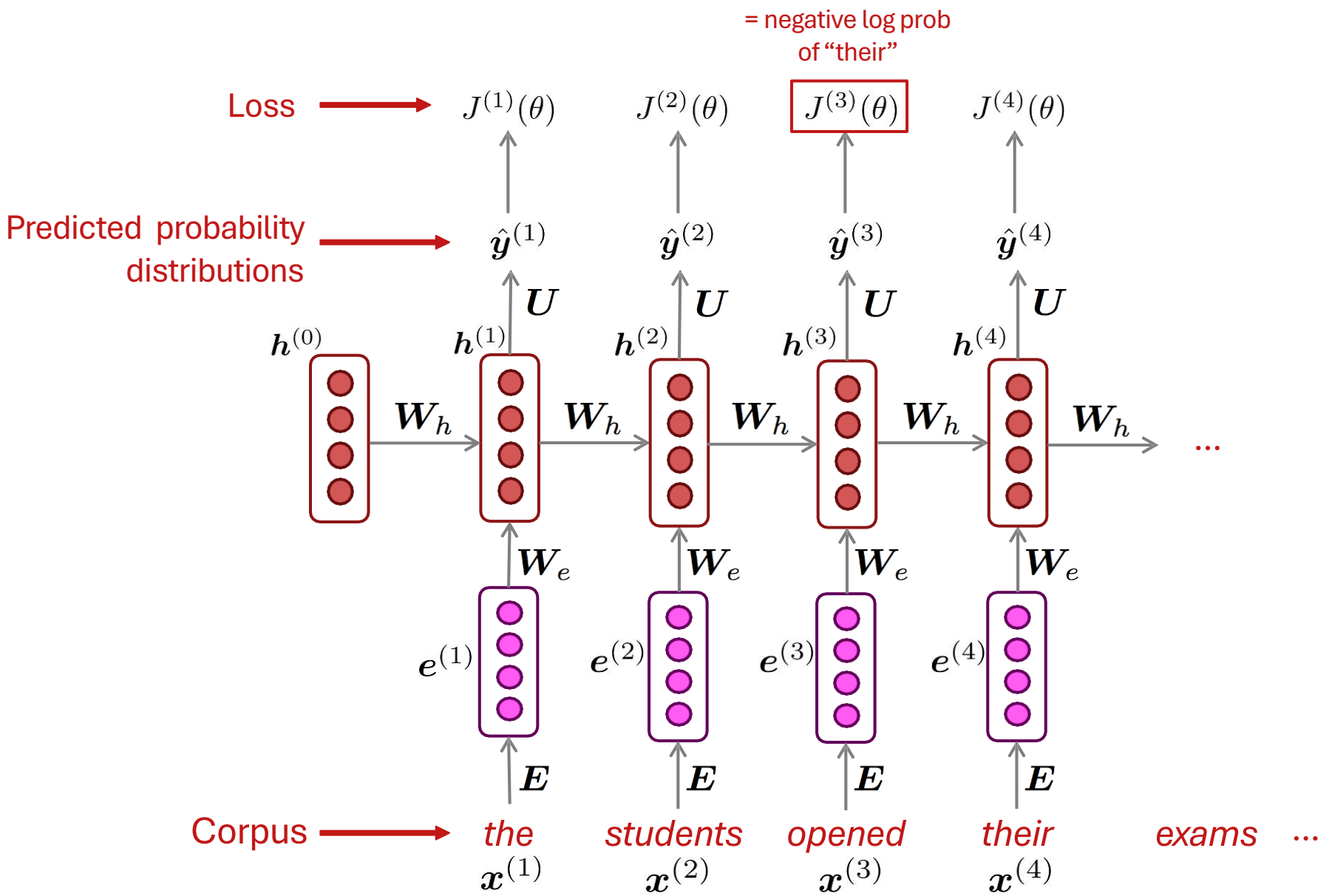
- Recurrent computation is slow
- In practice, difficult to access information from many steps back

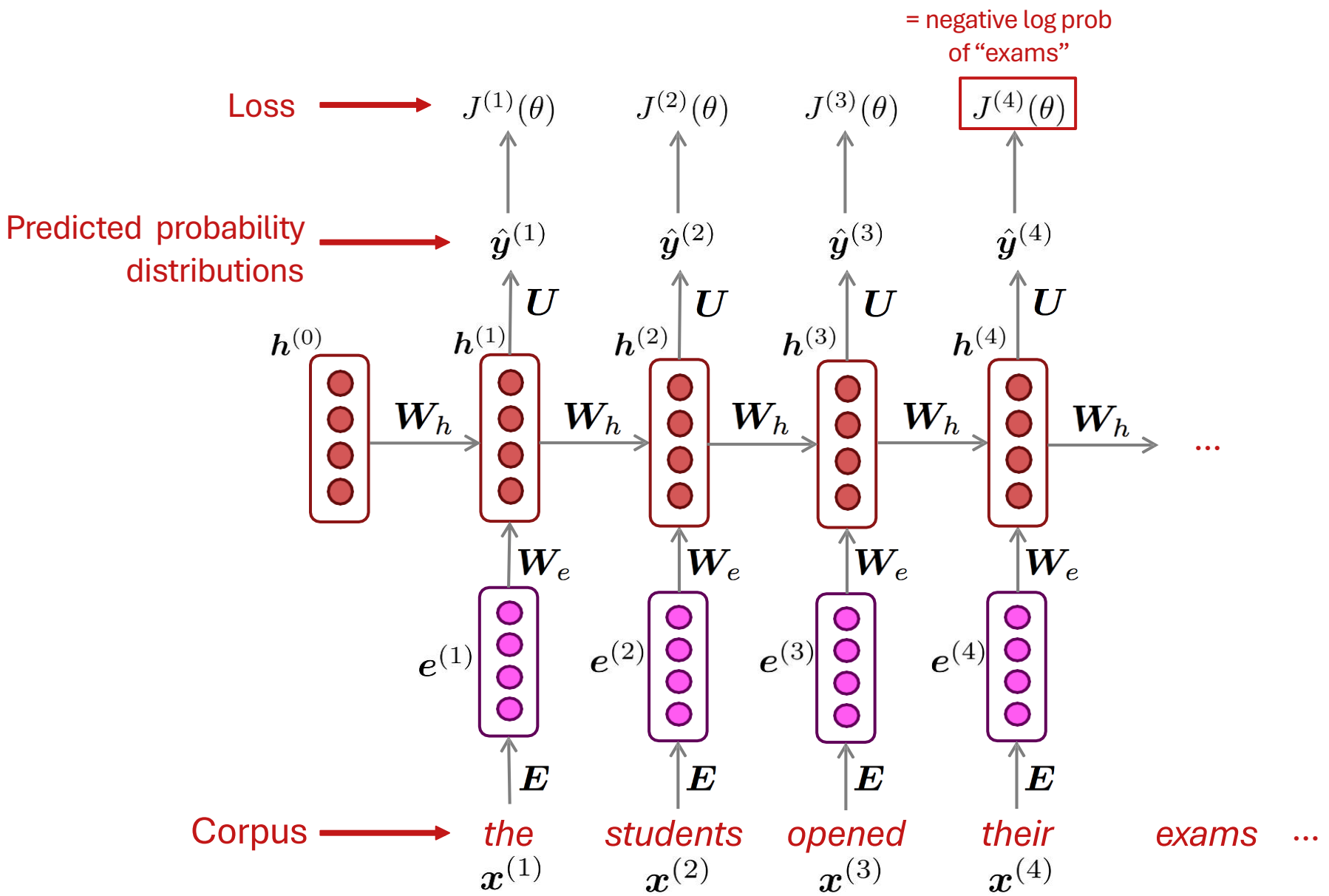


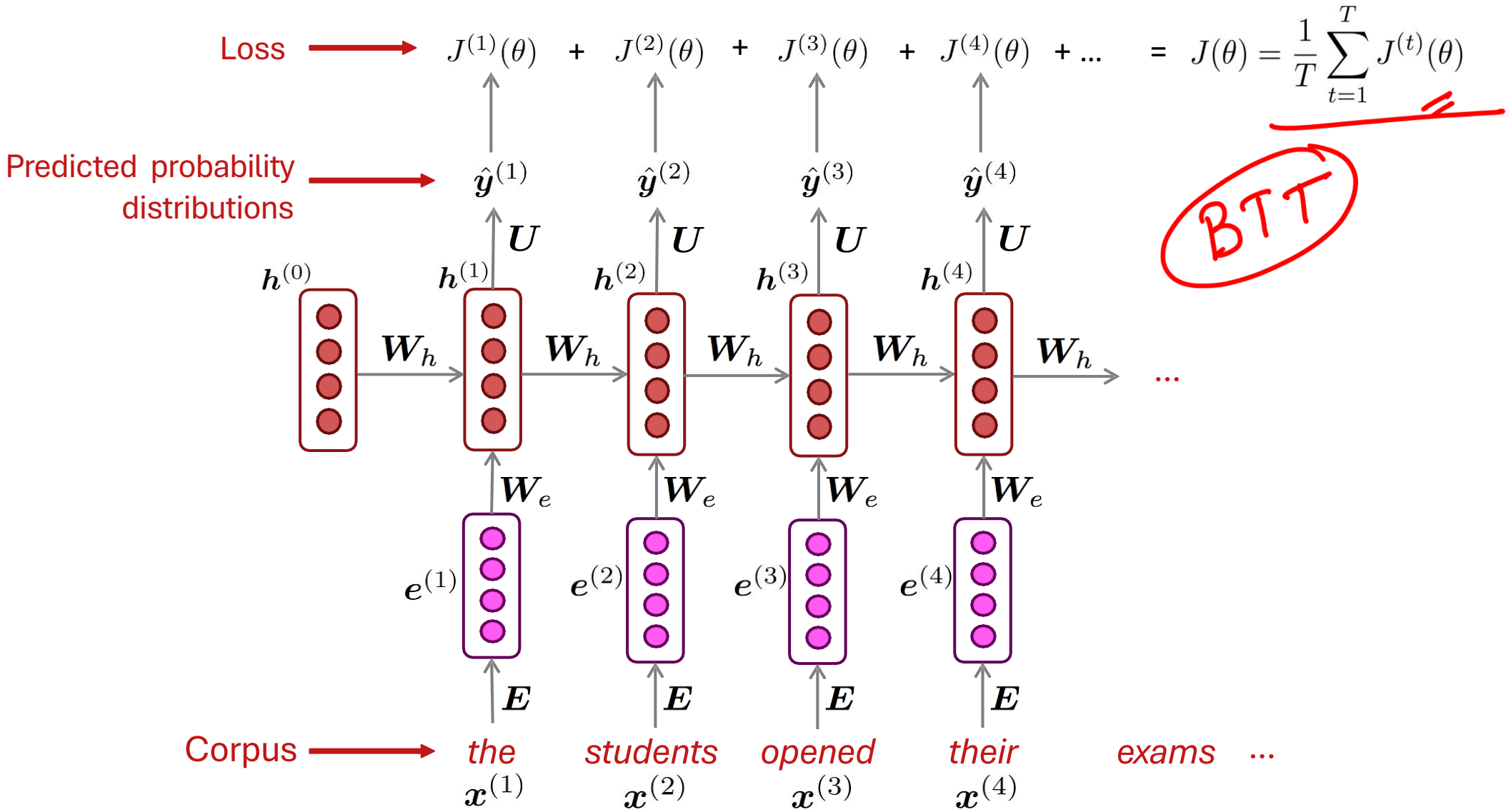
Training an RNN Language Model


= negative log prob of "students"
CE
Loss
$J^{(1)}(\theta)$
$J^{(2)}(\theta)$
$J^{(3)}(\theta)$
$J^{(4)}(\theta)$
Predicted probability distributions
$\hat{y}^{(1)}$
$\hat{y}^{(2)}$
$\hat{y}^{(3)}$
$\hat{y}^{(4)}$
U
$h^{(0)}$
$h^{(1)}$
$h^{(2)}$
$h^{(3)}$
$h^{(4)}$
W_h
...
W_e
$e^{(1)}$
$e^{(2)}$
$e^{(3)}$
$e^{(4)}$
E
Corpus
the
students
opened
their
exams
...
$x^{(1)}$
$x^{(2)}$
$x^{(3)}$
$x^{(4)}$


= negative log prob of "opened"
Loss
Predicted probability distributions
Corpus
$J^{(1)}(\theta)$
$J^{(2)}(\theta)$
$J^{(3)}(\theta)$
$J^{(4)}(\theta)$
$\hat{y}^{(1)}$
$\hat{y}^{(2)}$
$\hat{y}^{(3)}$
$\hat{y}^{(4)}$
$h^{(0)}$
$h^{(1)}$
$h^{(2)}$
$h^{(3)}$
$h^{(4)}$
U
W_h
W_e
E
$e^{(1)}$
$e^{(2)}$
$e^{(3)}$
$e^{(4)}$
the
$x^{(1)}$
students
$x^{(2)}$
opened
$x^{(3)}$
their
$x^{(4)}$
exams
...

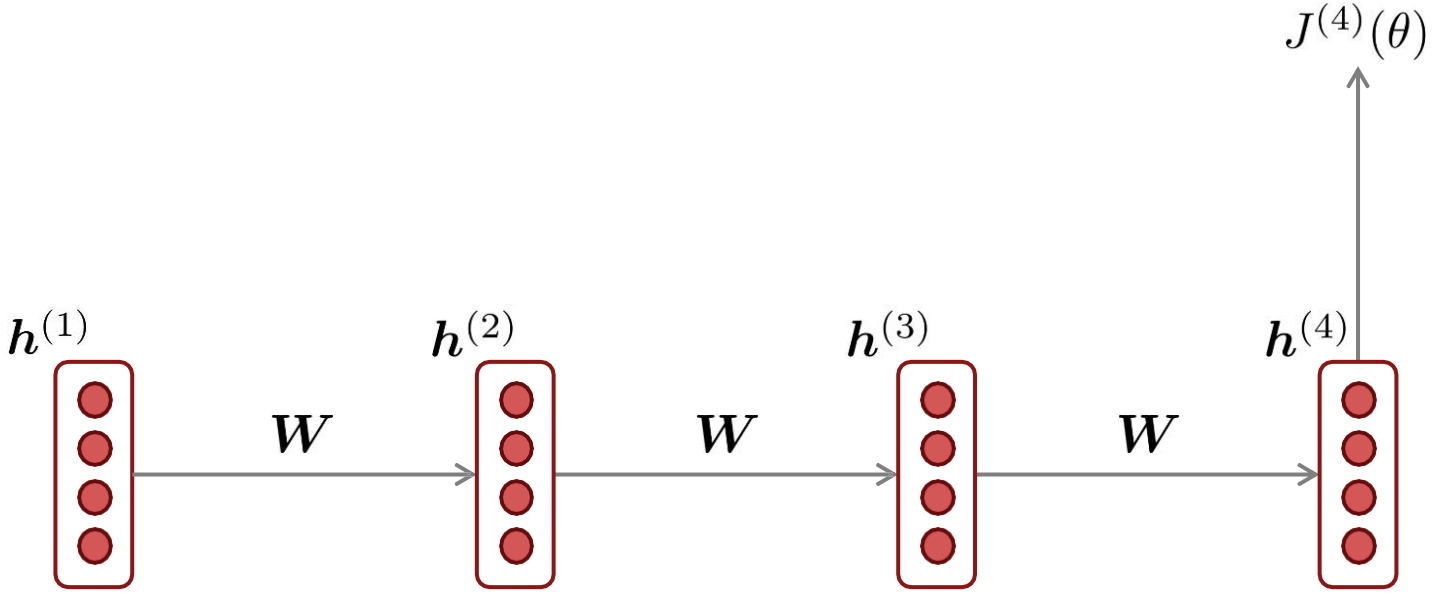

= negative log prob of "their"
Loss
Predicted probability distributions
Corpus
$J^{(1)}(\theta)$
$J^{(2)}(\theta)$
$J^{(3)}(\theta)$
$J^{(4)}(\theta)$
$\hat{y}^{(1)}$
$\hat{y}^{(2)}$
$\hat{y}^{(3)}$
$\hat{y}^{(4)}$
$h^{(0)}$
$h^{(1)}$
$h^{(2)}$
$h^{(3)}$
$h^{(4)}$
U
W_h
W_e
E
$e^{(1)}$
$e^{(2)}$
$e^{(3)}$
$e^{(4)}$
the
$x^{(1)}$
students
$x^{(2)}$
opened
$x^{(3)}$
their
$x^{(4)}$
exams
...


= negative log prob of "exams"
Loss
$J^{(1)}(\theta)$
$J^{(2)}(\theta)$
$J^{(3)}(\theta)$
$J^{(4)}(\theta)$
Predicted probability distributions
$\hat{\boldsymbol{y}}^{(1)}$
$\hat{\boldsymbol{y}}^{(2)}$
$\hat{\boldsymbol{y}}^{(3)}$
$\hat{\boldsymbol{y}}^{(4)}$
$\boldsymbol{U}$
$\boldsymbol{h}^{(0)}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(3)}$
$\boldsymbol{h}^{(4)}$
$\boldsymbol{W}_h$
...
$\boldsymbol{W}_e$
$\boldsymbol{e}^{(1)}$
$\boldsymbol{e}^{(2)}$
$\boldsymbol{e}^{(3)}$
$\boldsymbol{e}^{(4)}$
$\boldsymbol{E}$
Corpus
the
$\boldsymbol{x}^{(1)}$
students
$\boldsymbol{x}^{(2)}$
opened
$\boldsymbol{x}^{(3)}$
their
$\boldsymbol{x}^{(4)}$
exams
...

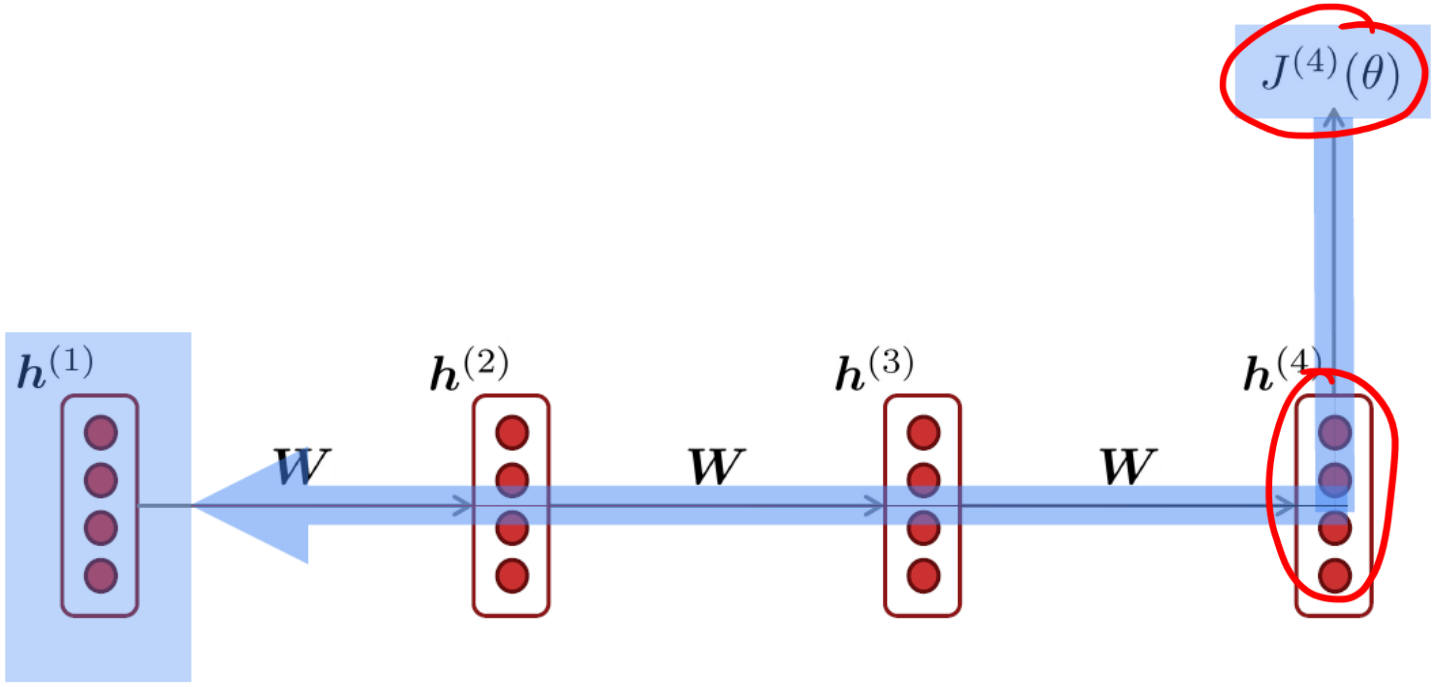

Loss
$J^{(1)}(\theta)$ + $J^{(2)}(\theta)$ + $J^{(3)}(\theta)$ + $J^{(4)}(\theta)$ + ... = $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$
Predicted probability distributions
$\hat{y}^{(1)}$ $\hat{y}^{(2)}$ $\hat{y}^{(3)}$ $\hat{y}^{(4)}$
U
$h^{(0)}$ $h^{(1)}$ $h^{(2)}$ $h^{(3)}$ $h^{(4)}$
W_h
...
W_e
$e^{(1)}$ $e^{(2)}$ $e^{(3)}$ $e^{(4)}$
E
Corpus
the $x^{(1)}$
students $x^{(2)}$
opened $x^{(3)}$
their $x^{(4)}$
exams ...
BTT

Problems with RNNs

Vanishing and Exploding Gradients

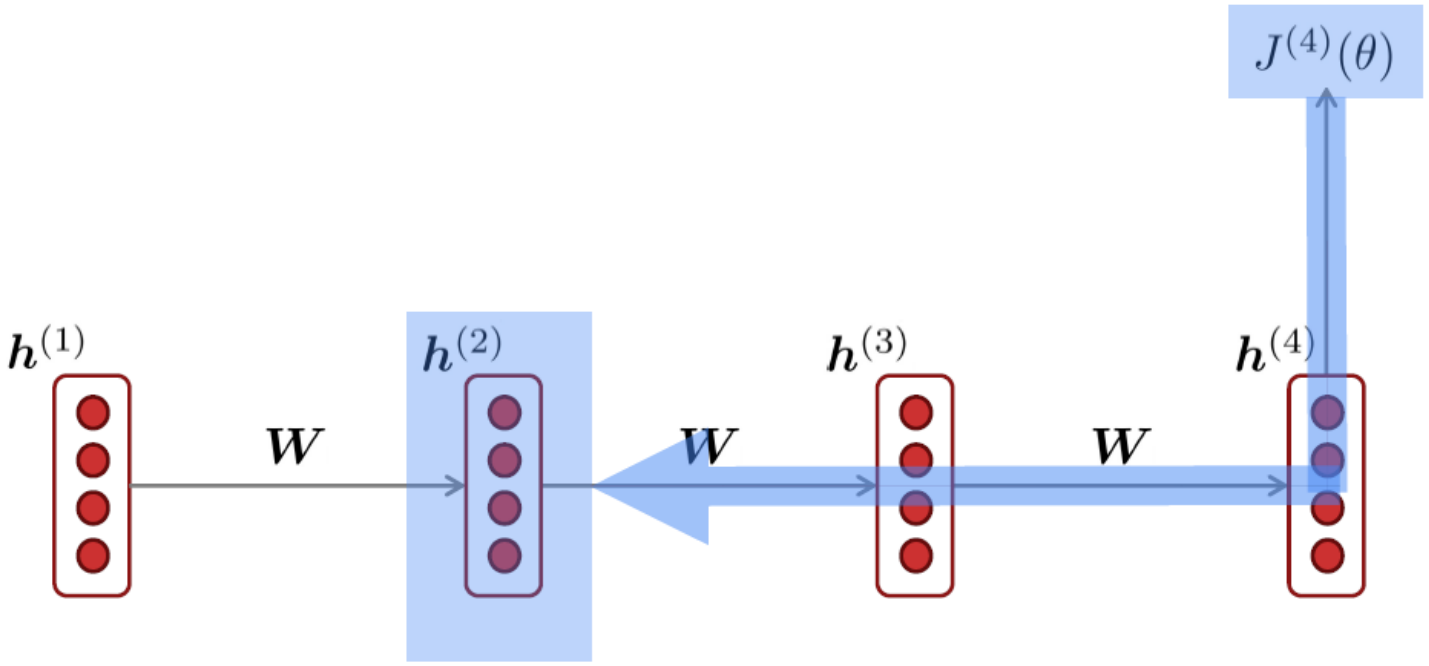


Vanishing Gradient Intuition



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \ ?$$

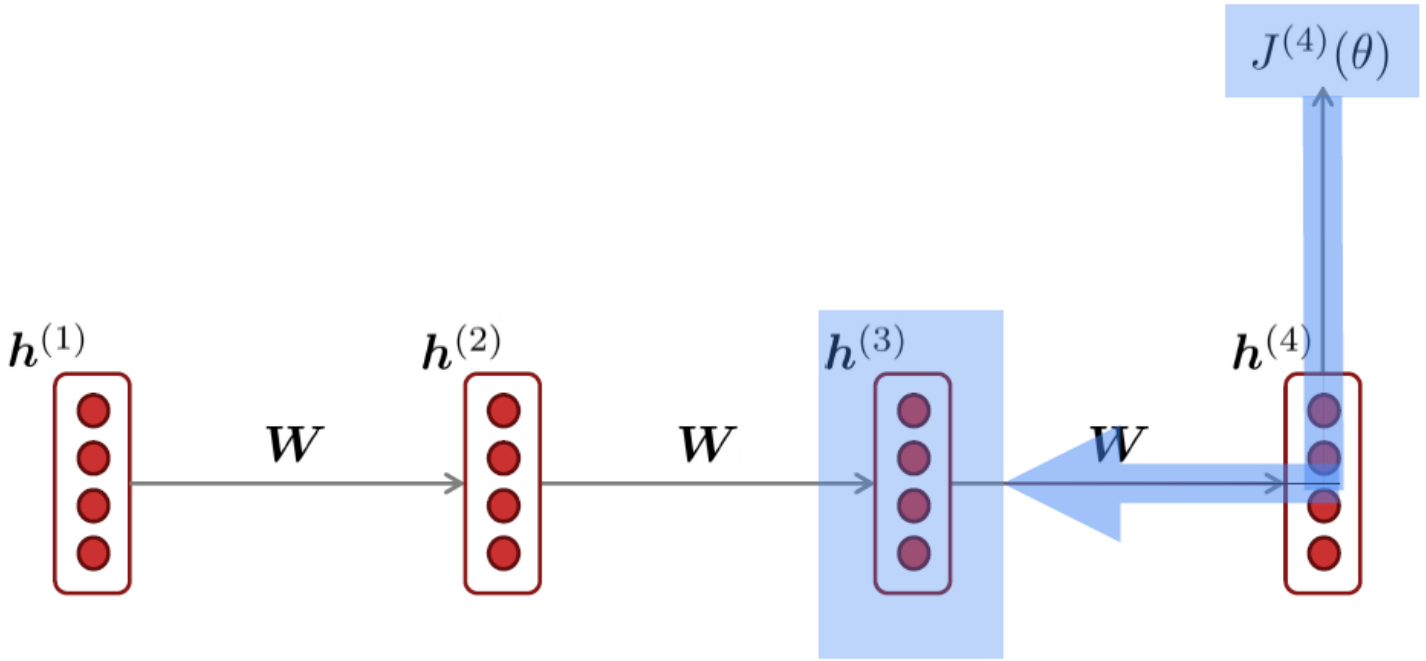
Vanishing Gradient Intuition



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

Chain Rule!

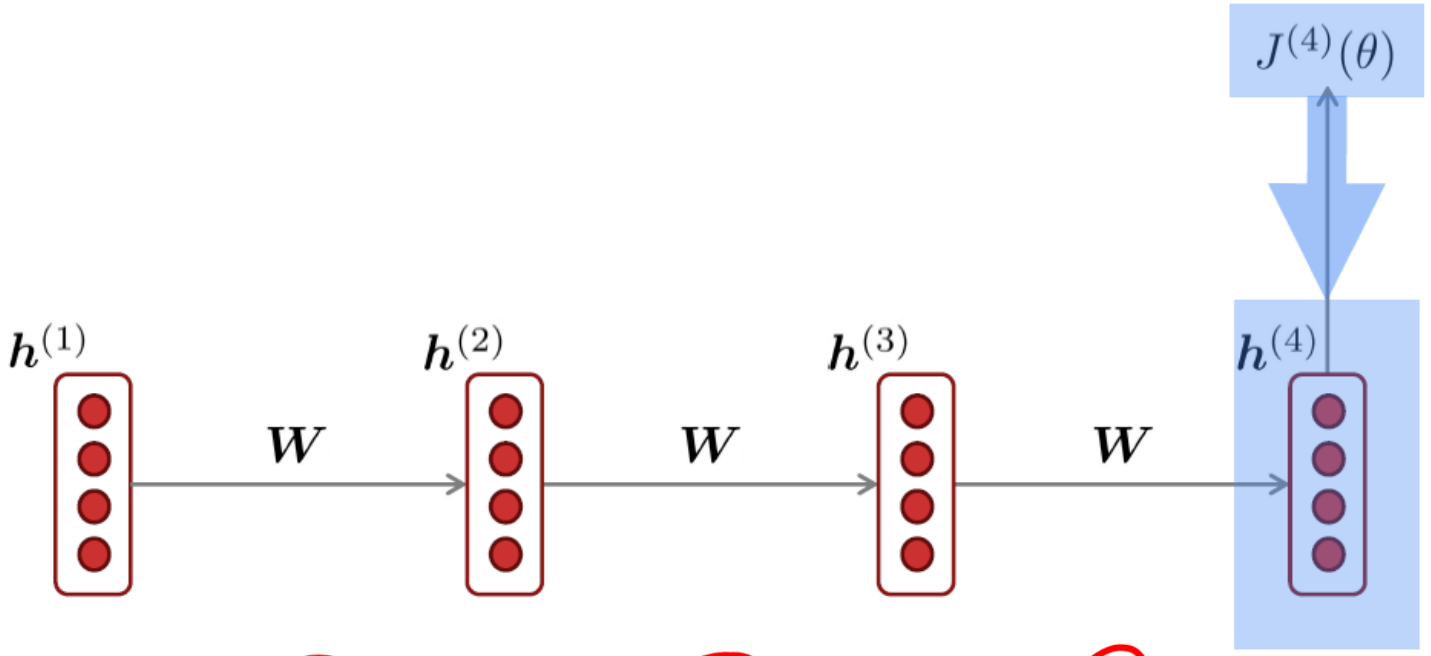
Vanishing Gradient Intuition



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

Chain Rule!

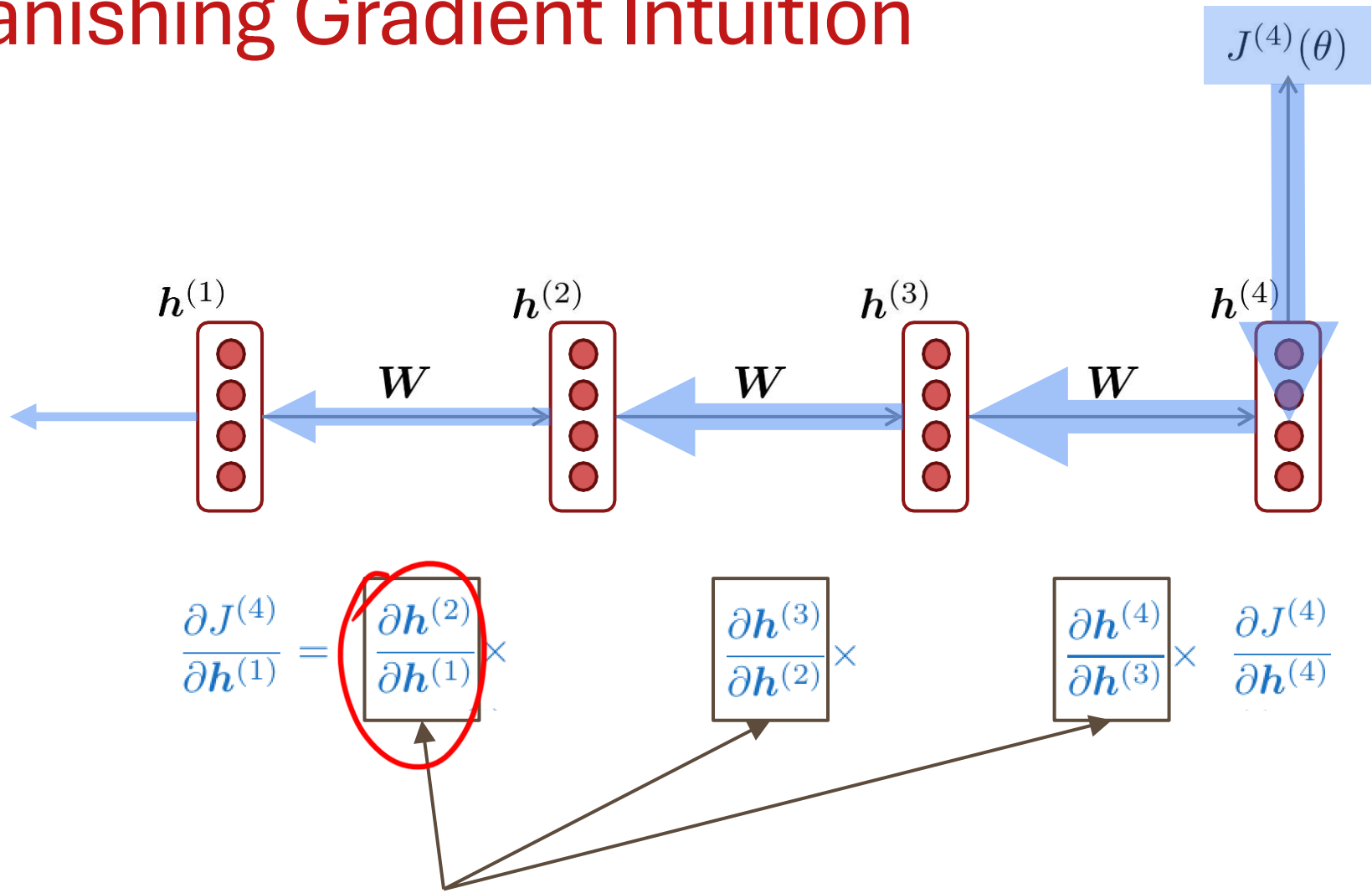
Vanishing Gradient Intuition



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

Chain Rule!

Vanishing Gradient Intuition



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

Effect of Vanishing Gradient on RNN-LM

- **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 7th 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

LSTMs & GRUs

LSTM

We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep *t*:

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

$$f^{(t)} = \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right)$$
$$i^{(t)} = \sigma\left(W_i h^{(t-1)} + U_i x^{(t)} + b_i\right)$$
$$o^{(t)} = \sigma\left(W_o h^{(t-1)} + U_o x^{(t)} + b_o\right)$$

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

$$\tilde{c}^{(t)} = \tanh\left(W_c h^{(t-1)} + U_c x^{(t)} + b_c\right)$$
$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \tilde{c}^{(t)}$$
$$h^{(t)} = o^{(t)} \odot \tanh c^{(t)}$$

Gates are applied using element-wise (or Hadamard) product: $\odot$

All these are vectors of same length *n*

Gated Recurrent Units (GRUs)

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep t, we have input $x^{(t)}$ and hidden state $h^{(t)}$ (no cell state).

Update gate: controls what parts of hidden state are updated vs preserved

Reset gate: controls what parts of previous hidden state are used to compute new content

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

How does this solve vanishing gradient?
Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{W}_u \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u \boldsymbol{x}^{(t)} + \boldsymbol{b}_u\right)$$

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

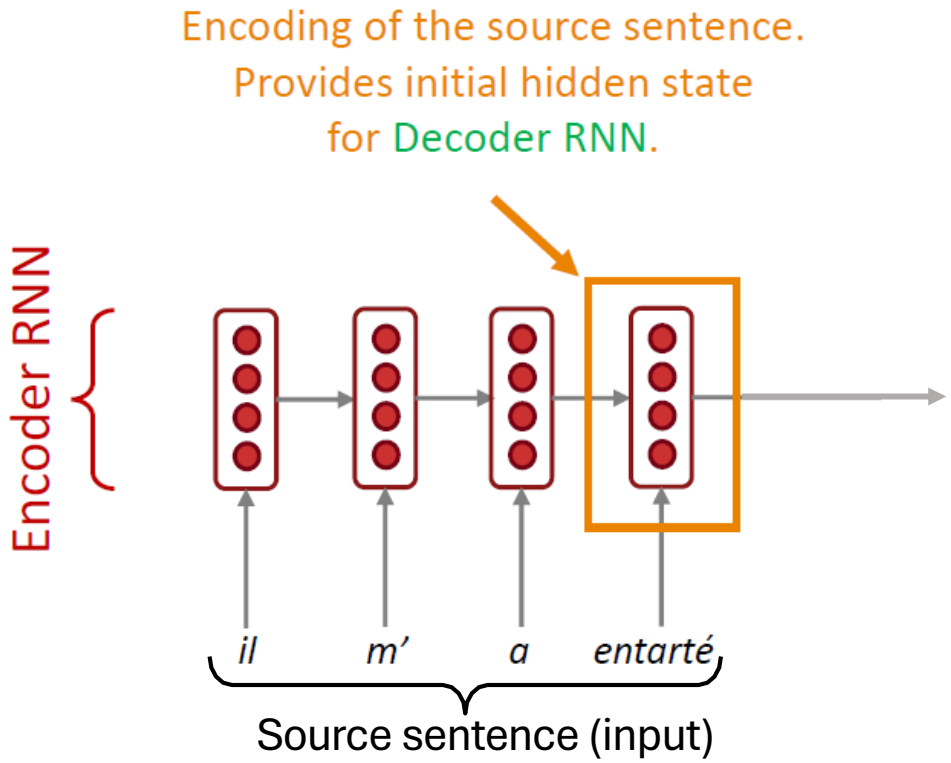
$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

"Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", Cho et al. 2014, https://arxiv.org/pdf/1406.1078v3.pdf

Sequence-to-Sequence Modeling

Neural Machine Translation (NMT)

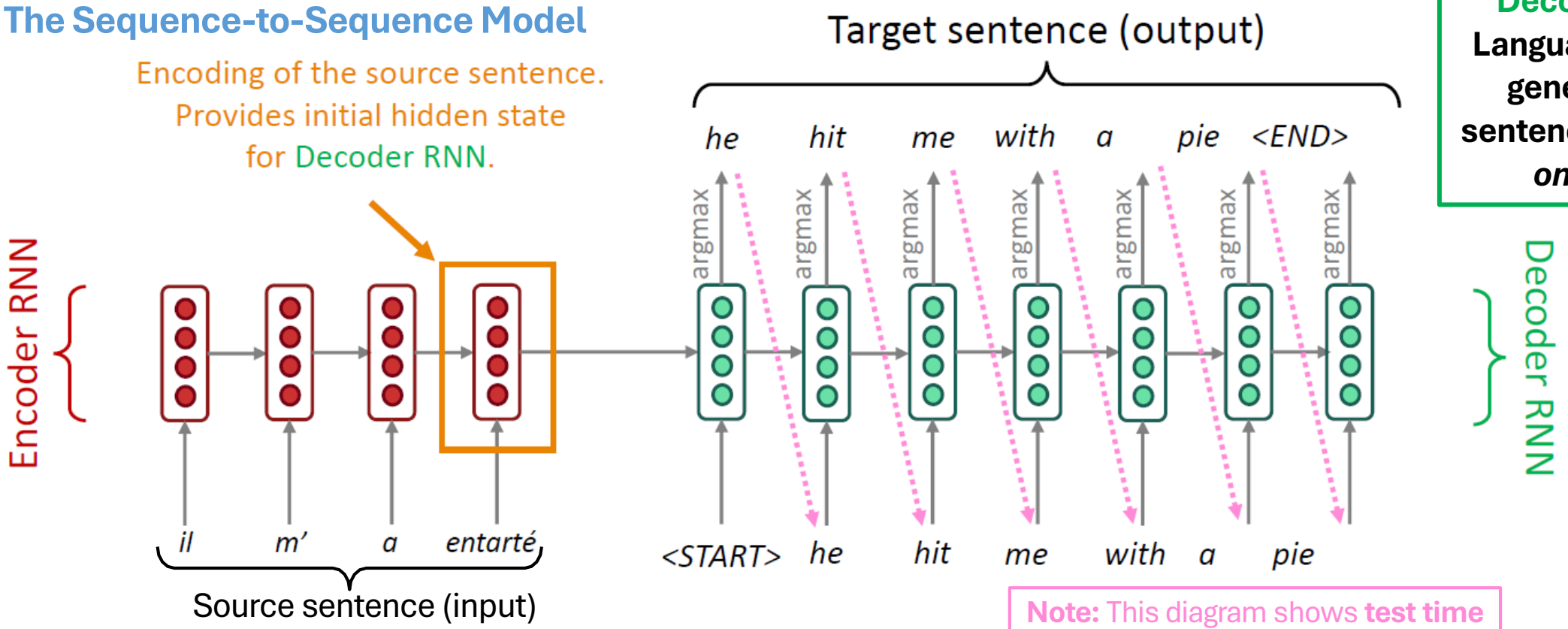
The Sequence-to-Sequence Model



Encoder RNN produces an encoding of the source sentence.

Neural Machine Translation (NMT)

The Sequence-to-Sequence Model



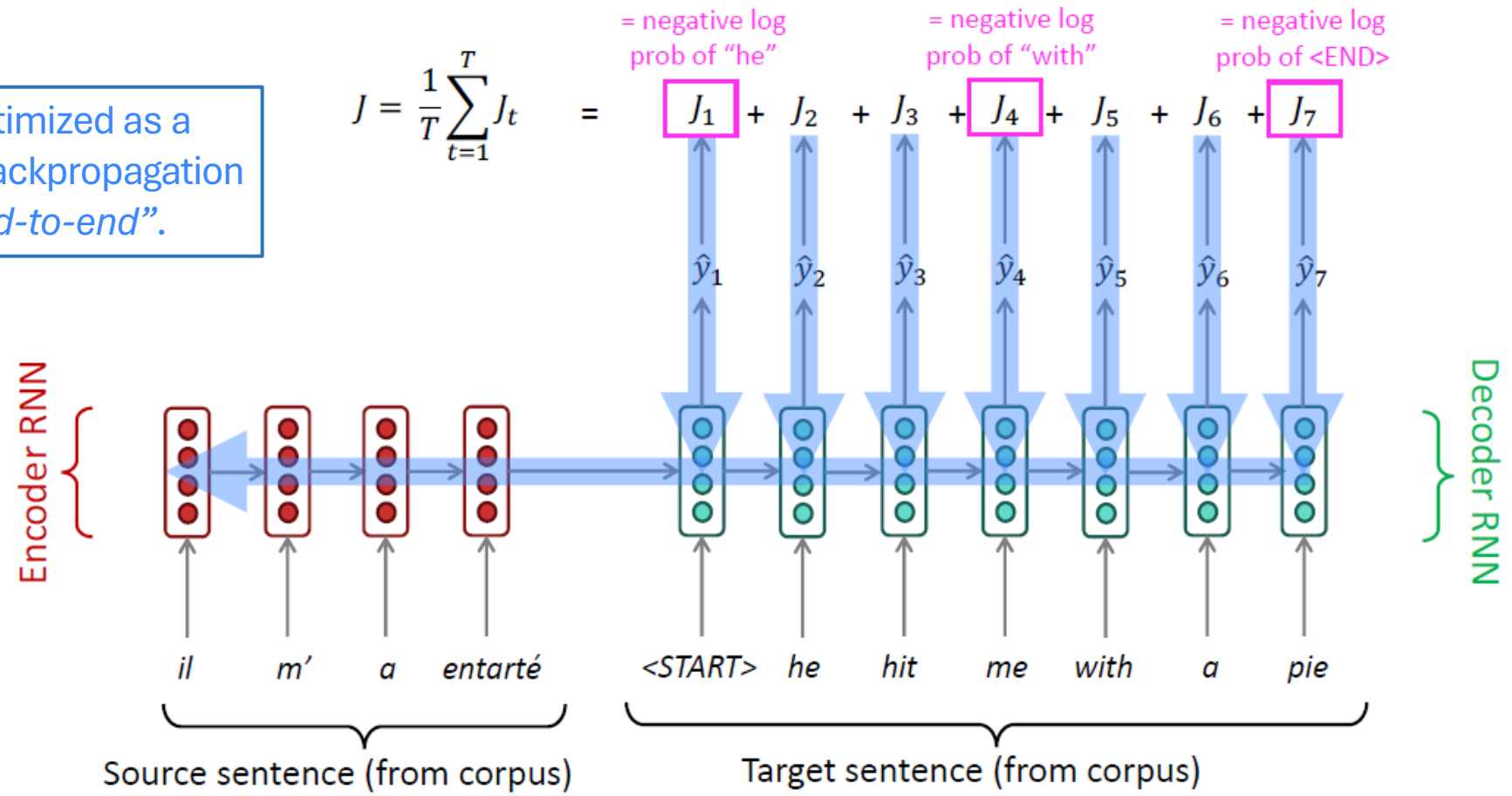
Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Encoder RNN produces an encoding of the source sentence.

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

Training an NMT System

Seq2seq is optimized as a **single system.** Backpropagation operates "*end-to-end*".

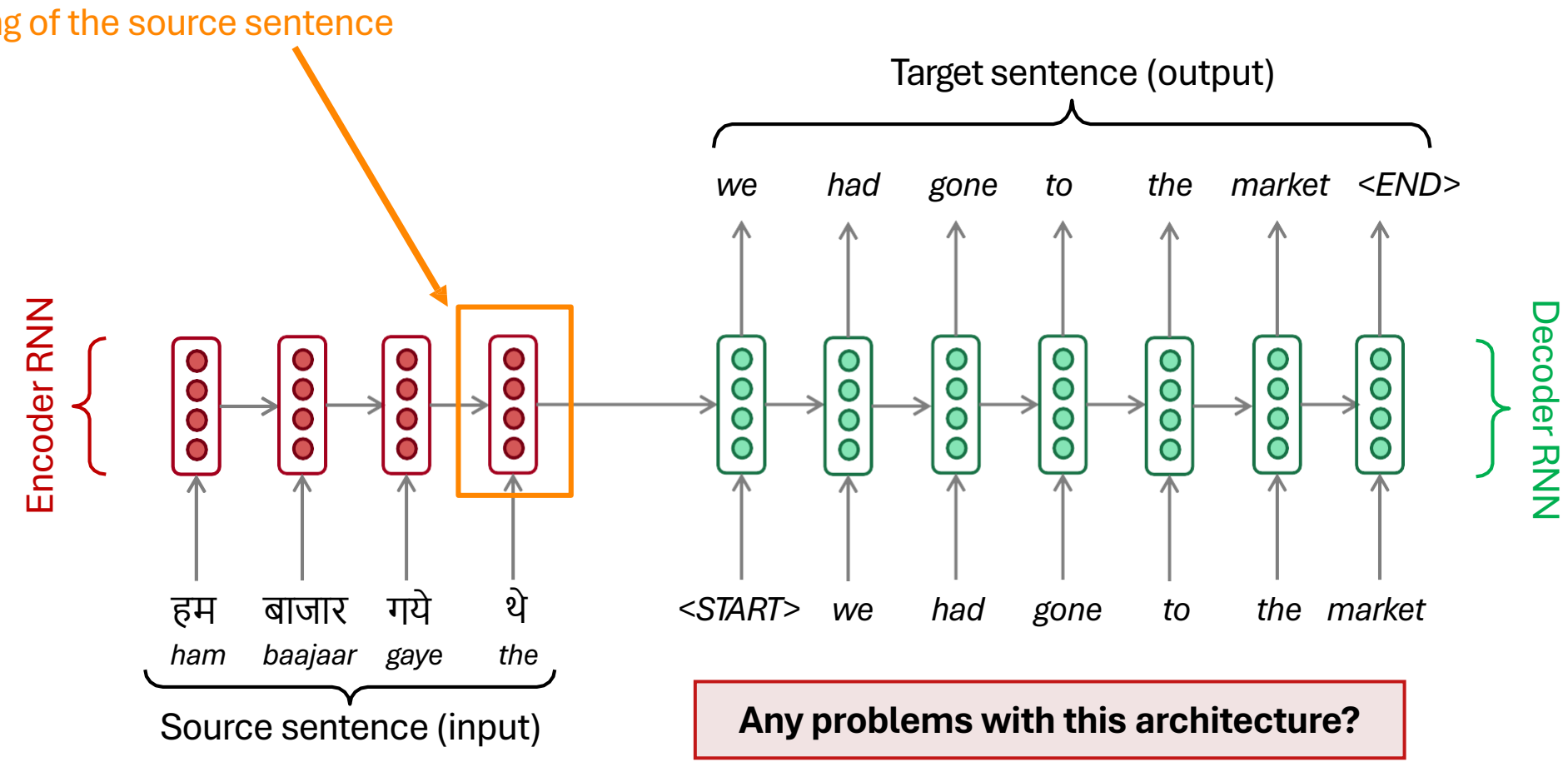


Issues With RNN

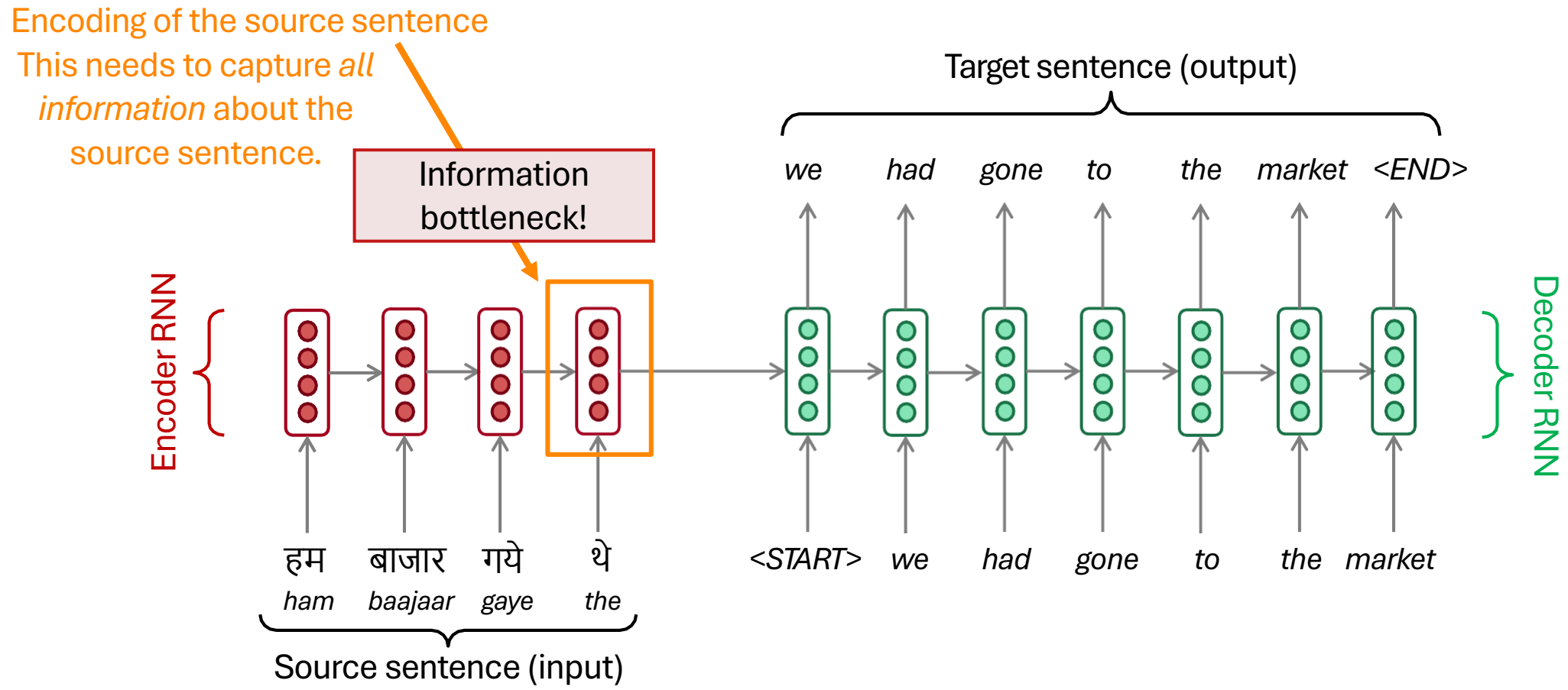
- Linear interaction distance
- Bottleneck problem
- Lack of parallelizability

ATTENTION

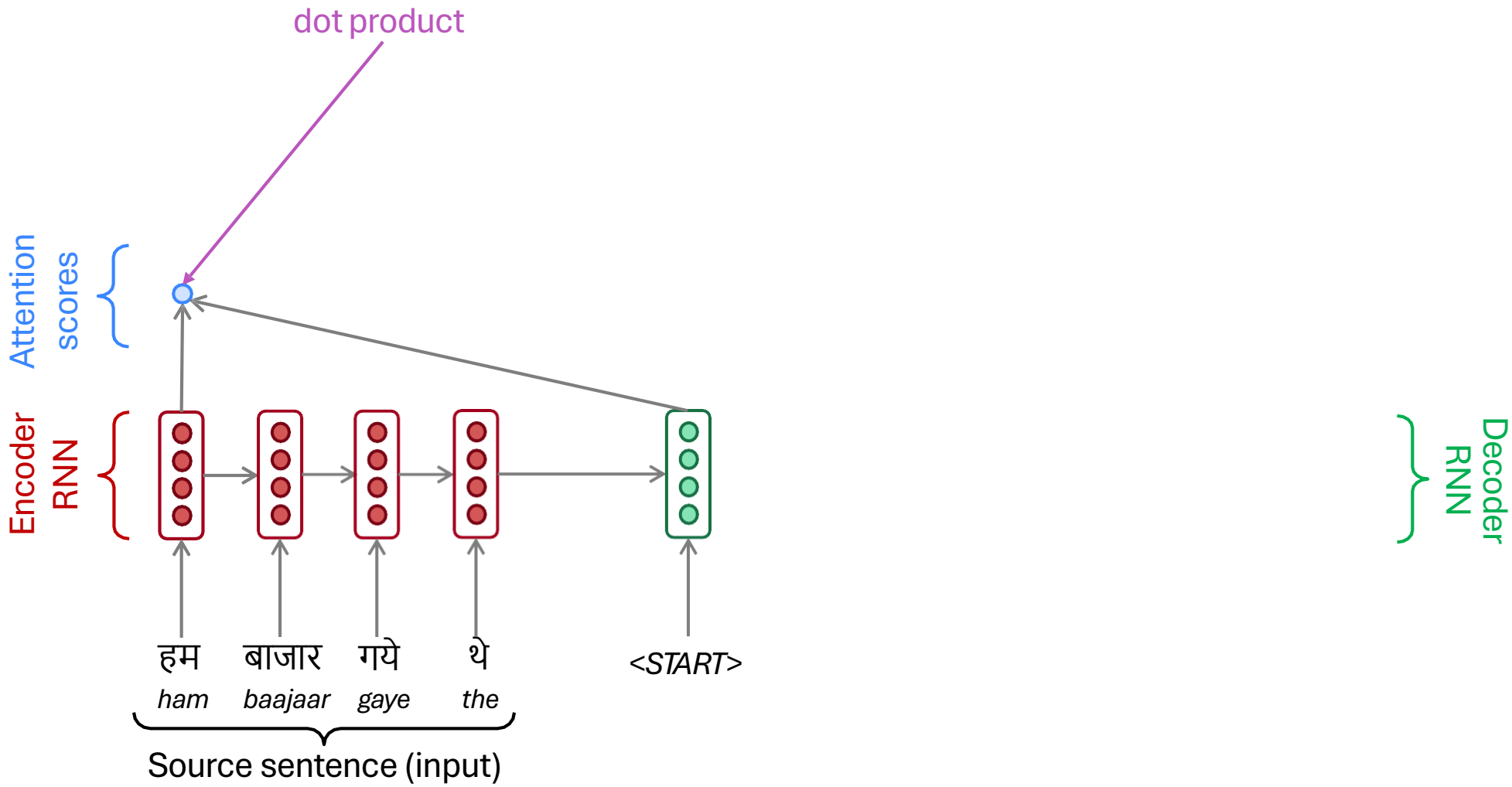
Sequence-to-Sequence: The Bottleneck Problem



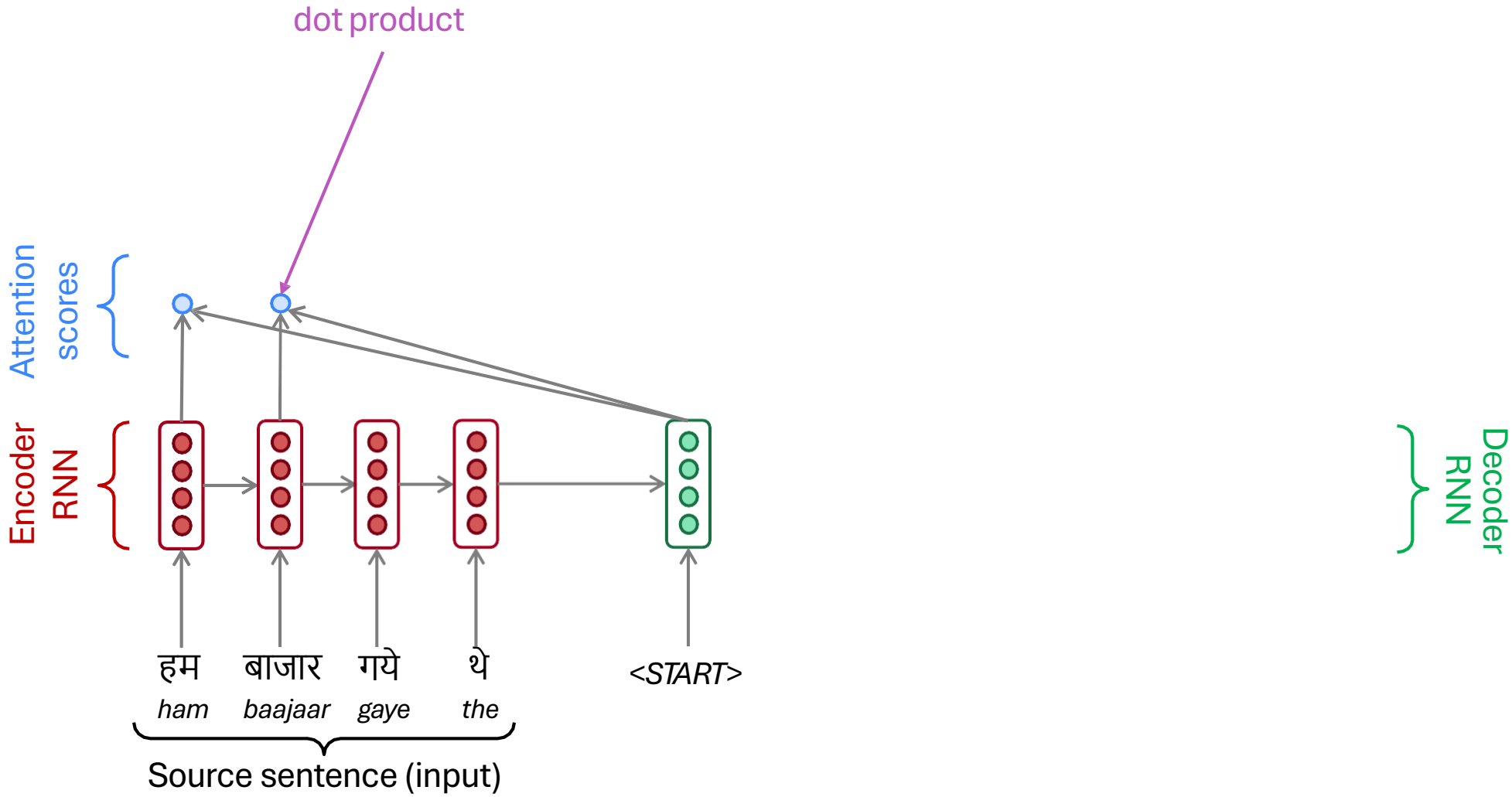
Sequence-to-Sequence: The Bottleneck Problem



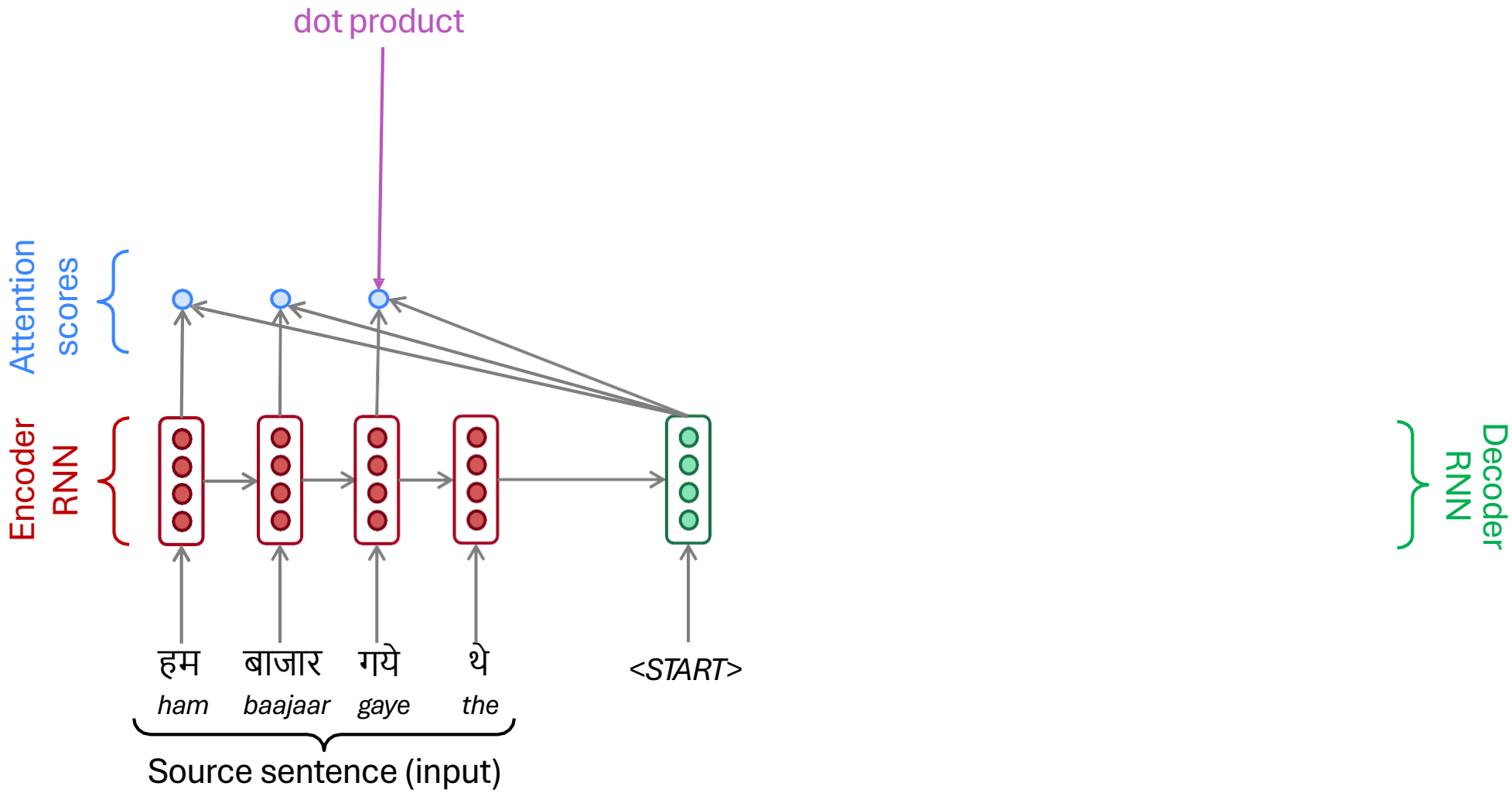
Sequence-to-Sequence With Attention



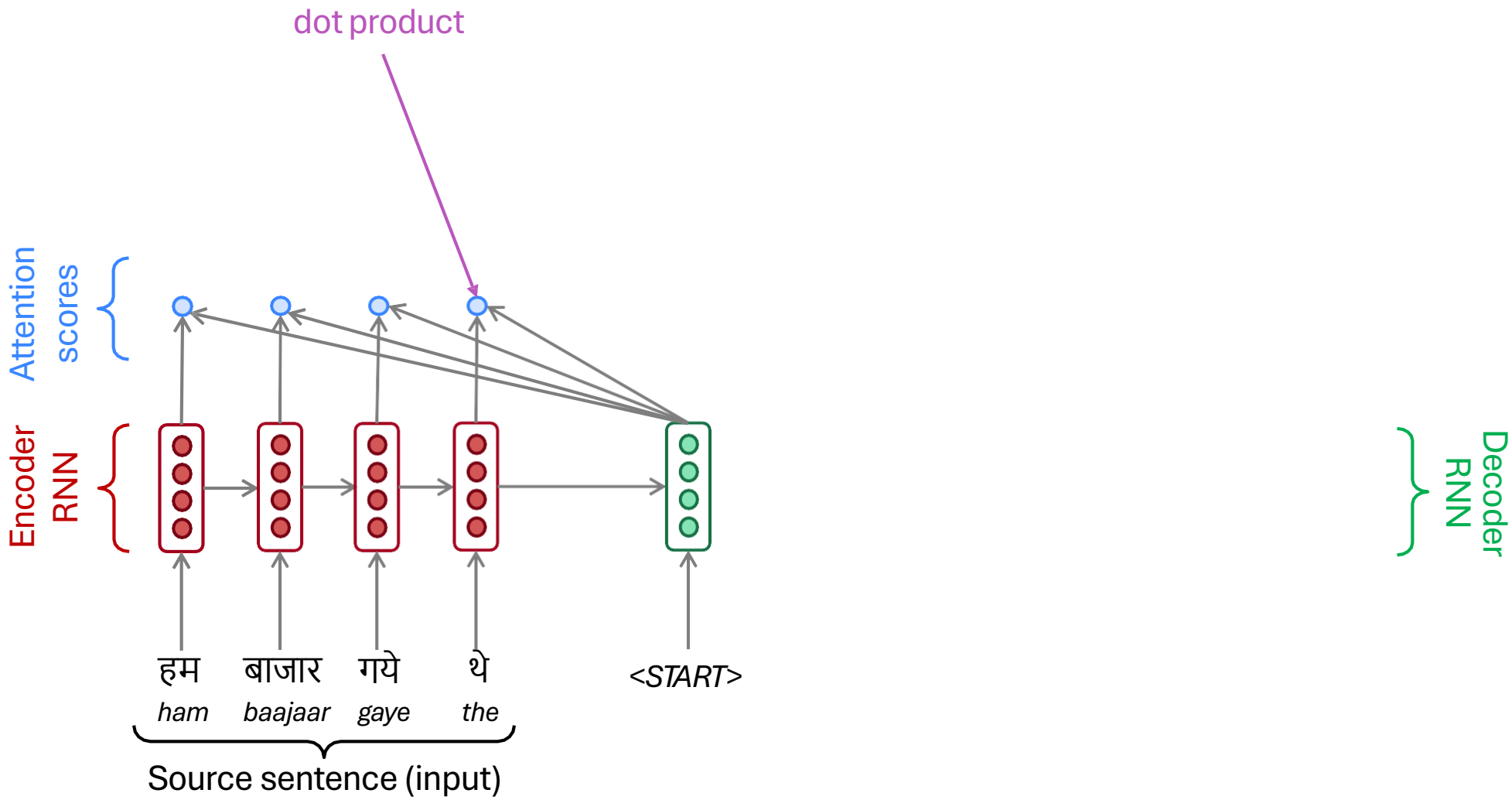
Sequence-to-Sequence With Attention



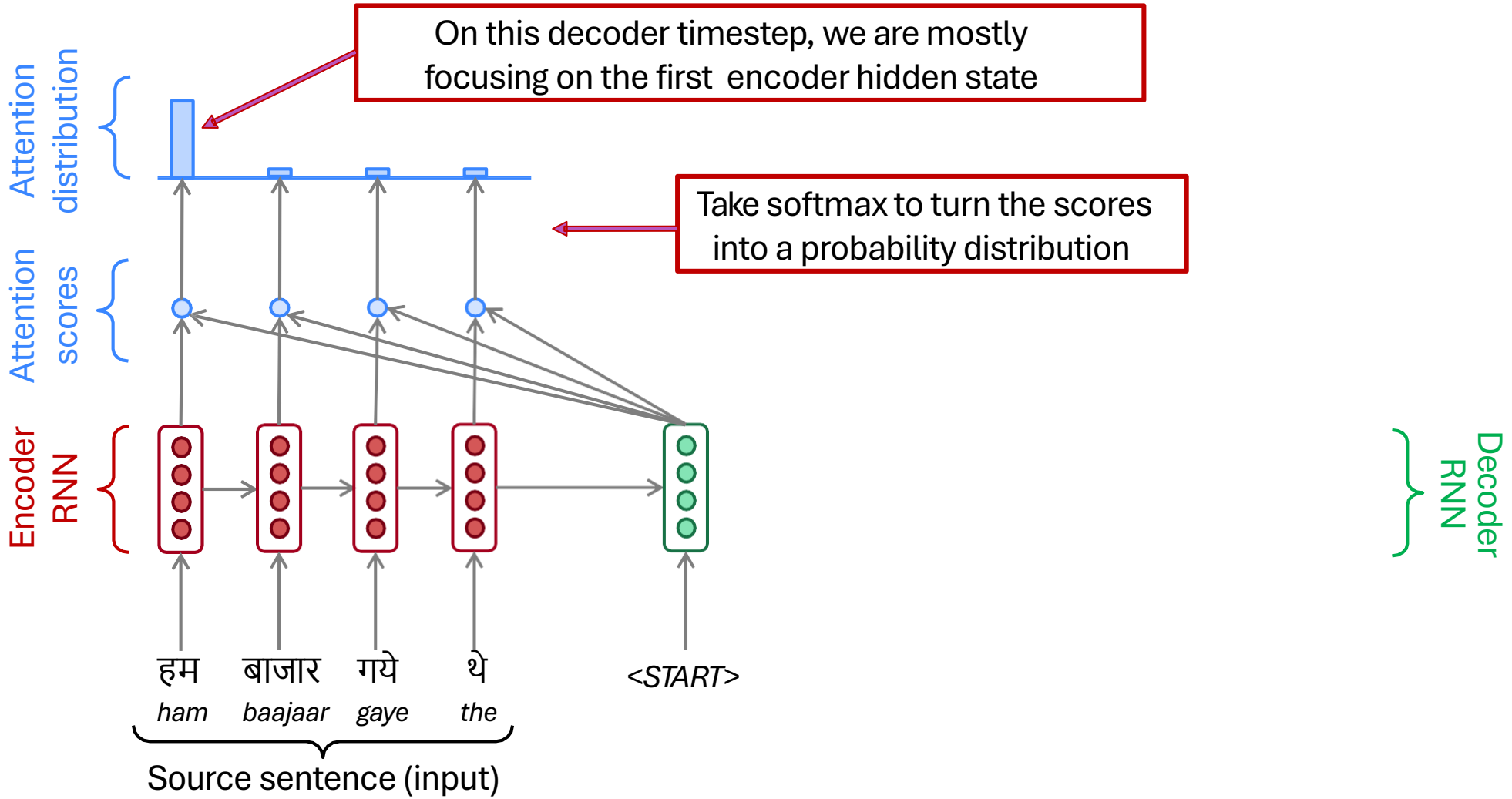
Sequence-to-Sequence With Attention



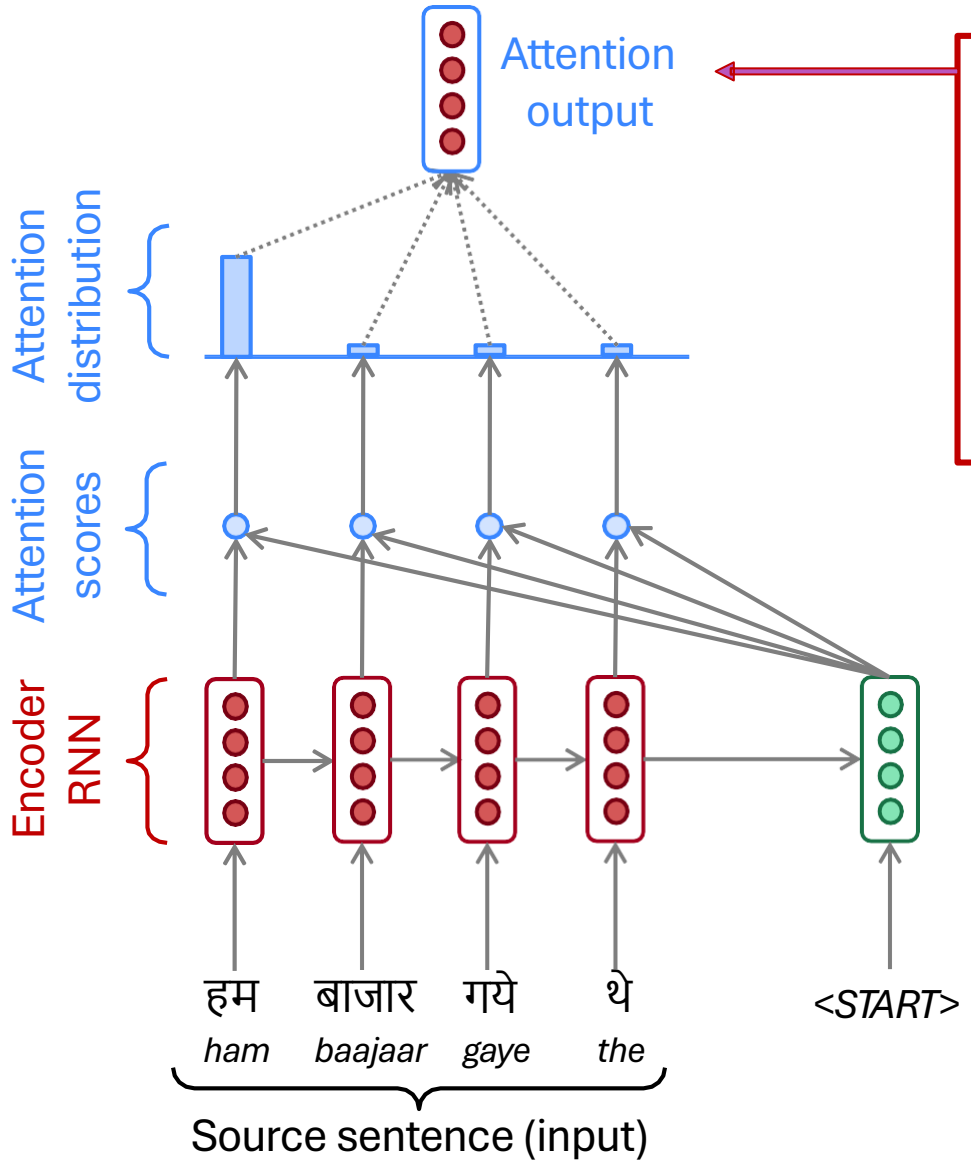
Sequence-to-Sequence With Attention



Sequence-to-Sequence With Attention



Sequence-to-Sequence With Attention

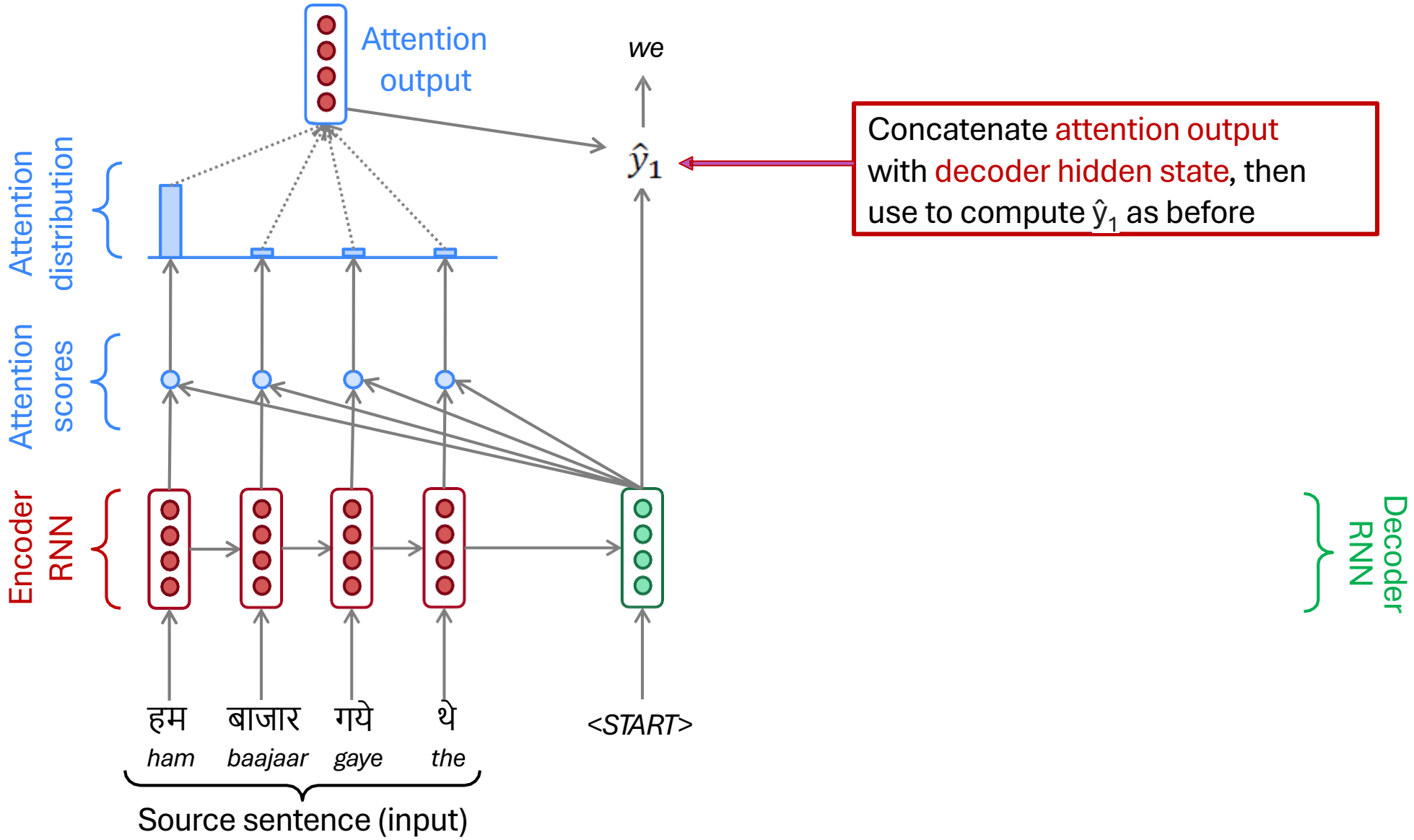


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

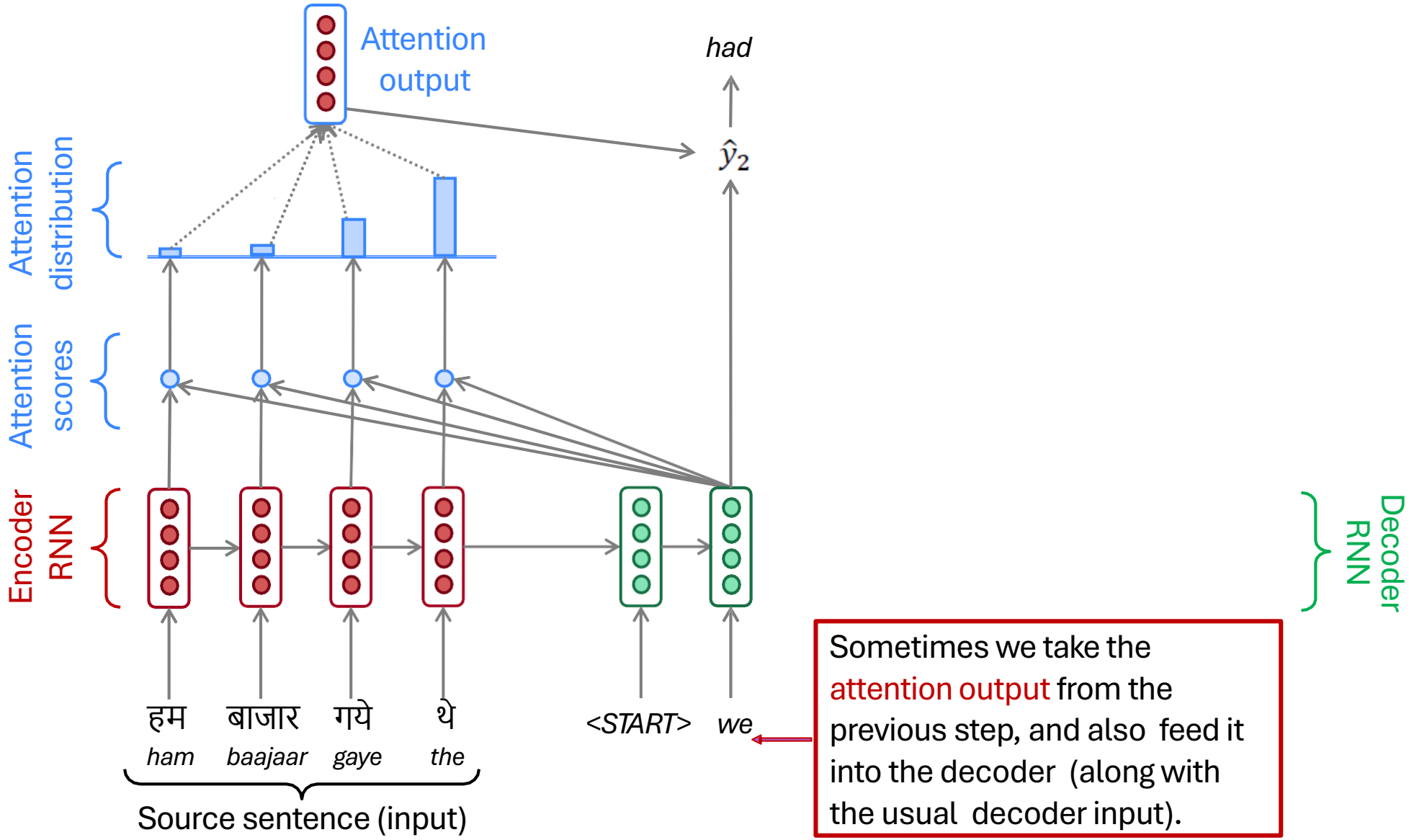
The attention output mostly contains information from the hidden states that received high attention.

Decoder RNN

Sequence-to-Sequence With Attention

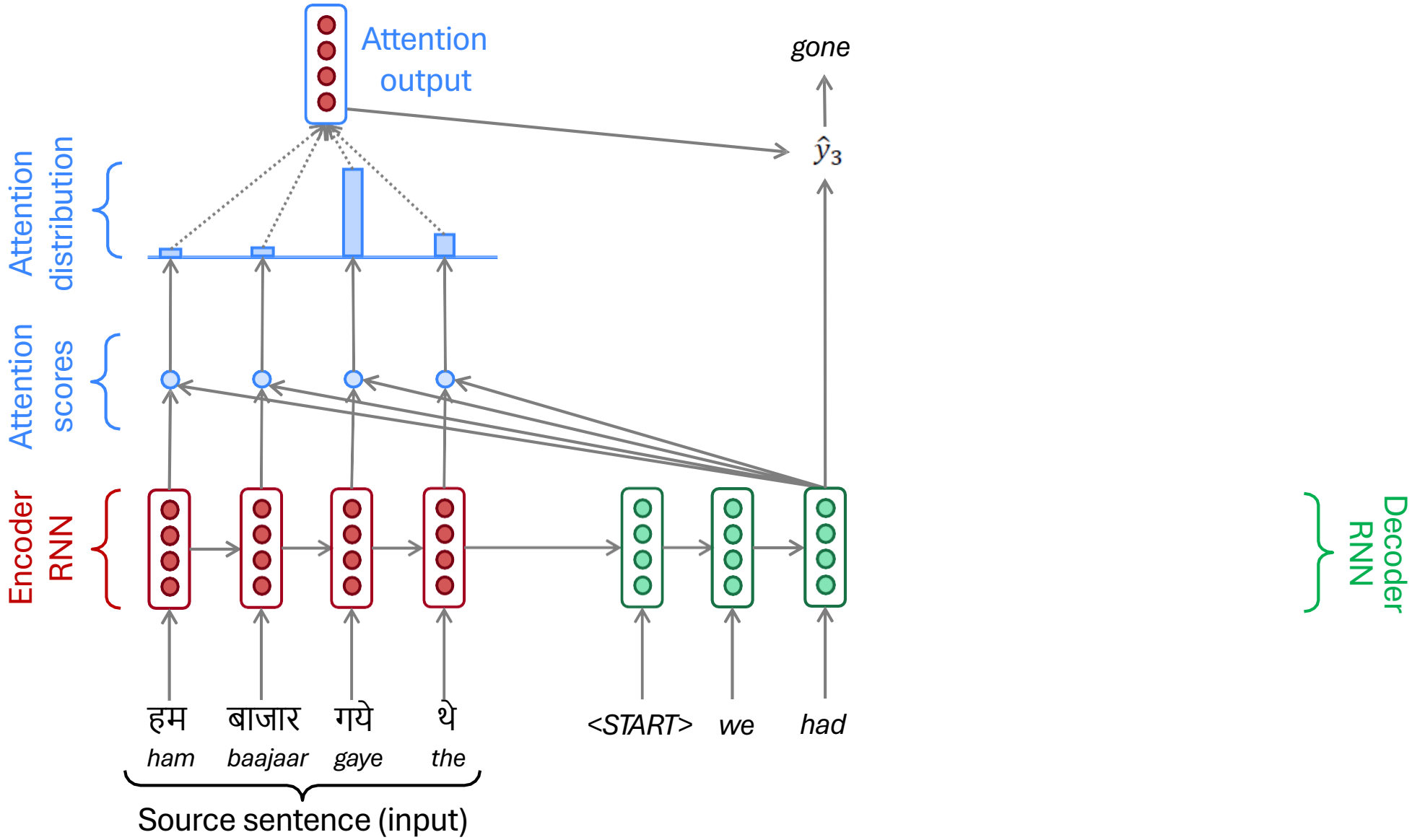


Sequence-to-Sequence With Attention

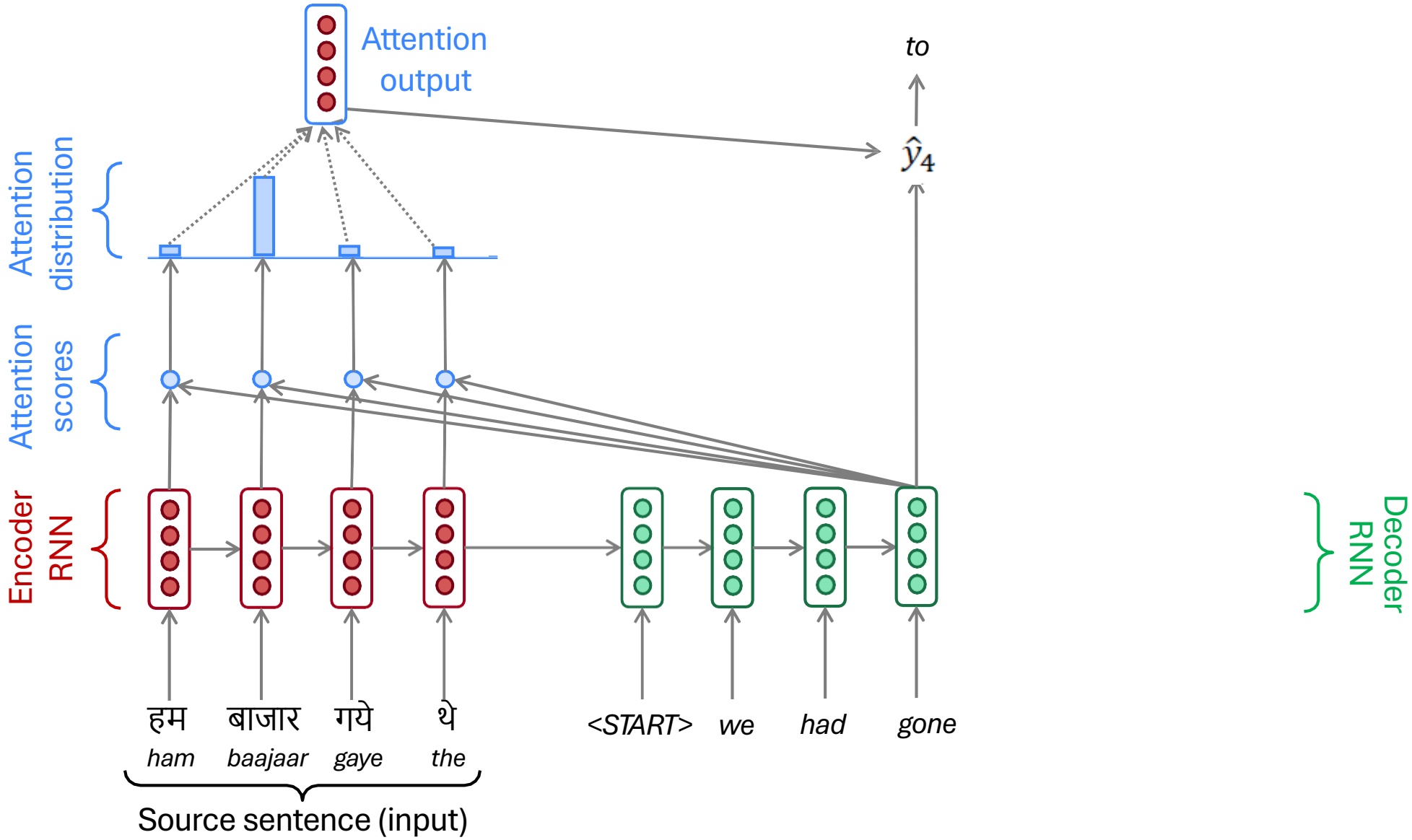


Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input).

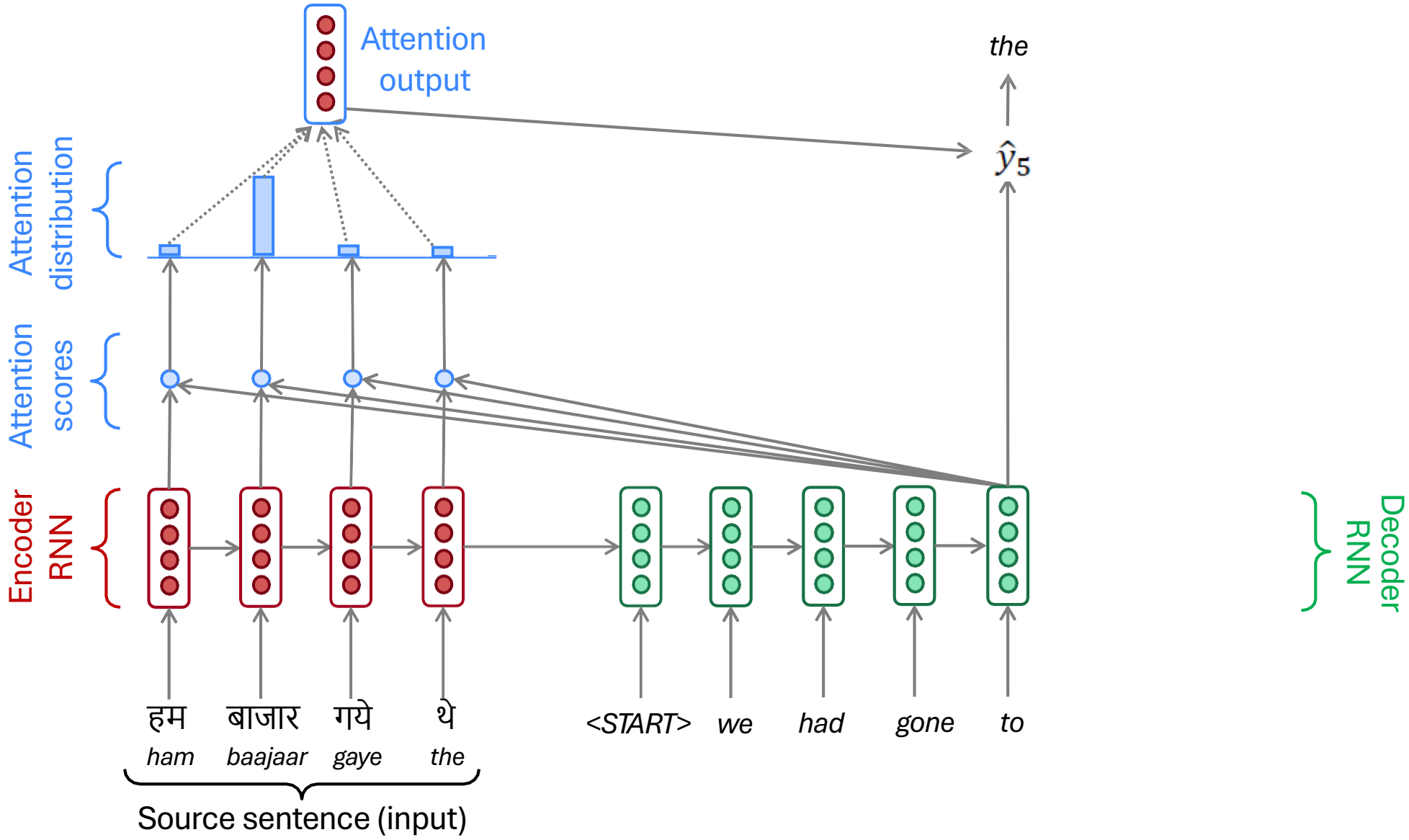
Sequence-to-Sequence With Attention



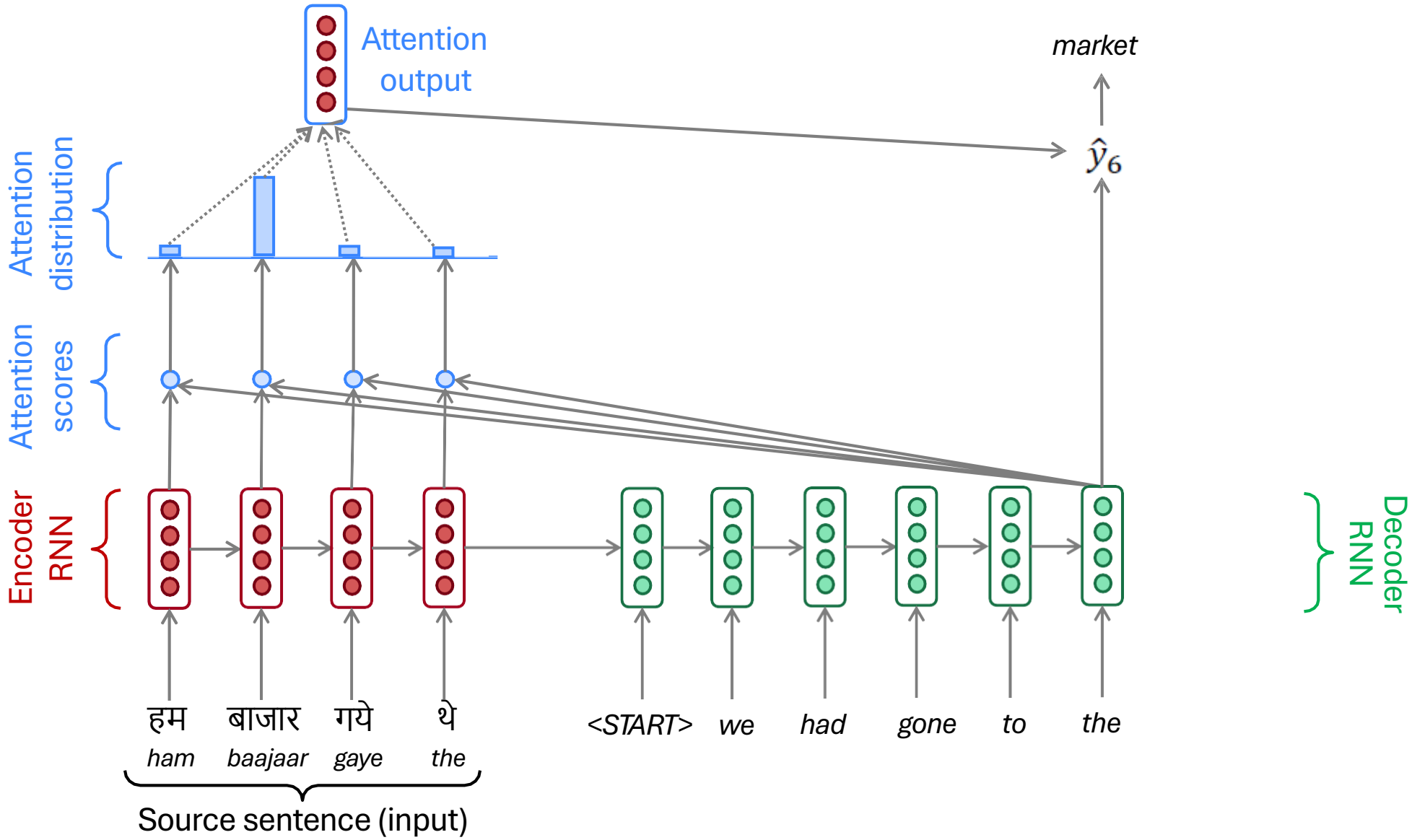
Sequence-to-Sequence With Attention



Sequence-to-Sequence With Attention

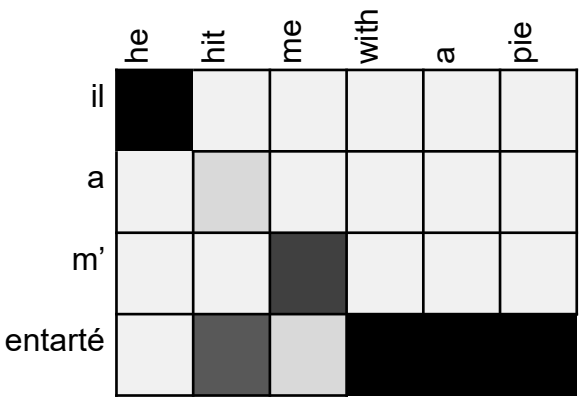


Sequence-to-Sequence With Attention



Attention is Great

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself



Attention is a *General* Deep Learning Technique

- We've seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- However: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)
- More general definition of attention:
  - Given a set of vector *values*, and a vector *query*, attention is a technique to compute a weighted sum of the values, dependent on the query.
- We sometimes say that the query *attends to* the values.
- For example, in the seq2seq + attention model, each decoder hidden state (query) *attends to* all the encoder hidden states (values).
- **Intuition**:
  - The weighted sum is a *selective summary* of the information contained in the values, where the query determines which values to focus on.
  - Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).

Variants of Attention

- Original formulation: $a(\mathbf{q}, \mathbf{k}) = w_2^T \tanh(W_1[\mathbf{q}; \mathbf{k}])$
- Bilinear product: $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T W \mathbf{k}$ Luong et al., 2015
- Dot product: $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T \mathbf{k}$ Luong et al., 2015
- Scaled dot product: $a(\mathbf{q}, \mathbf{k}) = \frac{\mathbf{q}^T \mathbf{k}}{\sqrt{|\mathbf{k}|}}$ Vaswani et al., 2017

More information:
"Deep Learning for NLP Best Practices", Ruder, 2017. http://ruder.io/deep-learning-nlp-best-practices/index.html#attention
"Massive Exploration of Neural Machine Translation Architectures", Britz et al, 2017, https://arxiv.org/pdf/1703.03906.pdf

Self-Attention