Sequence-to-Sequence Modeling & Attention

Large Language Models: Introduction and Recent Advances

ELL881 · AIL821

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

Slides are adopted from the Stanford course 'NLP with DL' by C. Manning and UMass course 'Advanced NLP' by M Iyyer

Released on July 31, 2024

Google Developers Blog

Gemma 2 2B released!

Google Deepmind releases this 2B model of Gemma 2 family, prioritizing safety and accessibility.

Along with the Gemma 2 2B model, they have also released **ShieldGemma** , a suite of safety content classifier models to **filter the input and outputs of AI models** and keep the user safe, and **Gemma Scope**, a new **model interpretability tool** that offers unparalleled insight into our models' inner workings.

This 2B model is also trained using **distillation from larger models**.

Gemma 2 2B **surpasses larger models** like **GPT-3.5 Turbo, Mixtral, Llama 2 70b** on the **LMSYS Chatbot Arena leaderboard**, demonstrating its exceptional conversational AI abilities.

Sequence-to-Sequence Modeling

Seq2Seq

Neural Machine Translation?

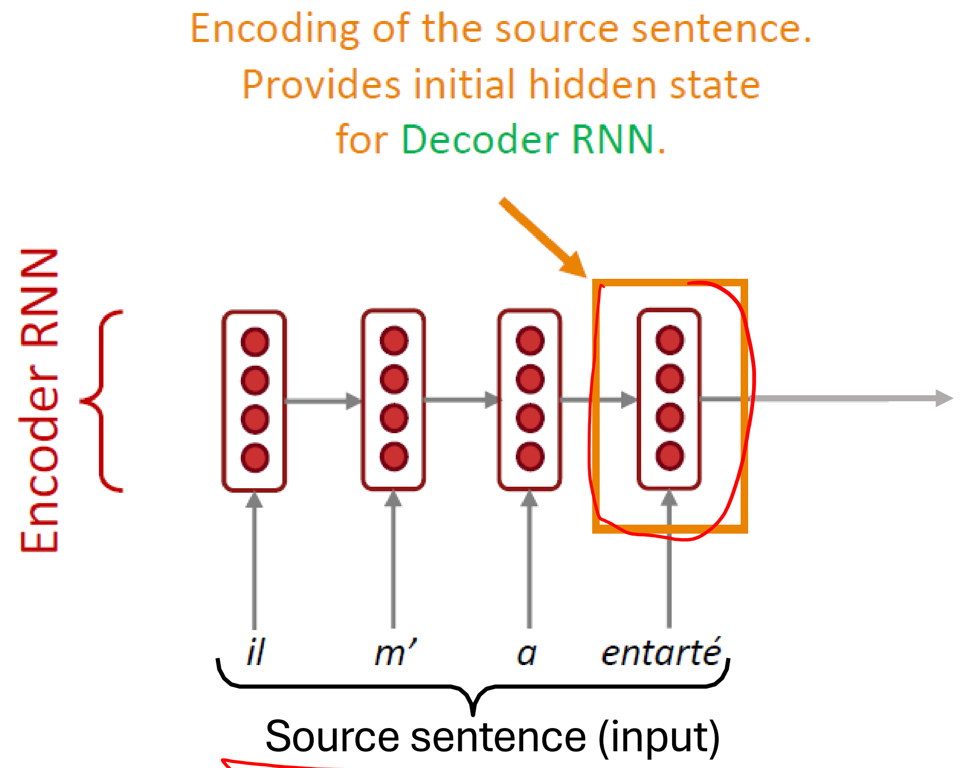
many to many
many to one
one to many

- Neural Machine Translation (NMT) is a way to do Machine Translation with a *single neural network*.

- The neural network architecture is called sequence-to-sequence (aka seq2seq) and it involves *two* RNNs.

Neural Machine Translation (NMT)

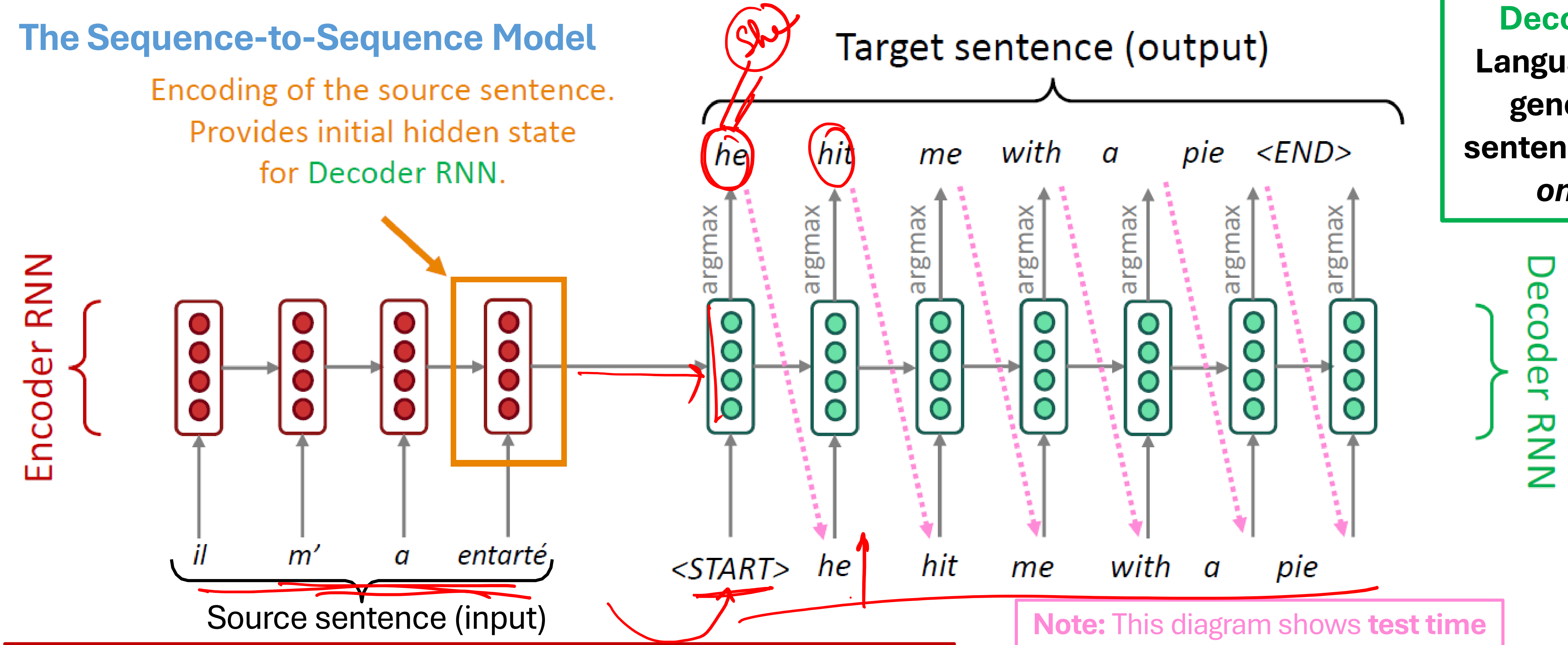
The Sequence-to-Sequence Model



Encoder RNN produces an encoding of the source sentence.

Neural Machine Translation (NMT)

The Sequence-to-Sequence Model



Encoder RNN produces an encoding of the source sentence.

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

Sequence-to-Sequence is Versatile!

- 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
  - If the input and output are sequences, we call it a seq2seq model
- Sequence-to-sequence is useful for *more than just MT*
- Many NLP tasks can be phrased as sequence-to-sequence:
  - **Summarization** (long text → short text)
  - **Dialogue** (previous utterances → next utterance)
  - **Parsing** (input text → output parse as sequence)
  - **Code generation** (natural language → Python code)

Neural Machine Translation (NMT)

- The sequence-to-sequence model is an example of a **Conditional Language Model**
  - **Language Model** because the decoder is predicting the next word of the target sentence *y*
  - **Conditional** because its predictions are also conditioned on the source sentence *x*
- NMT directly calculates $P(y|x)$

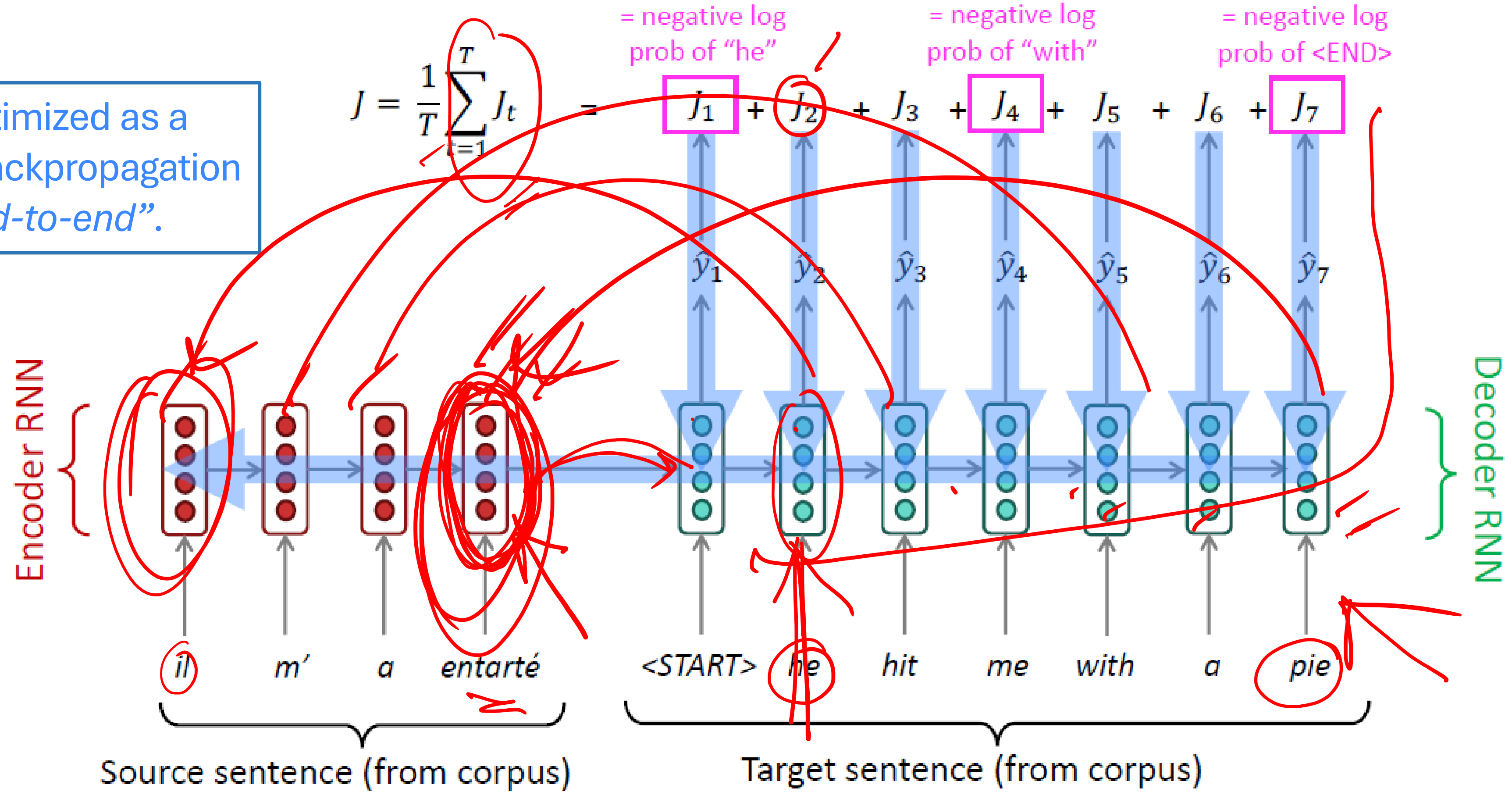
$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots \underbrace{P(y_T|y_1, \ldots, y_{T-1}, x)}_{\text{Probability of next target word, given target words so far and source sentence } x}$$

- How to train an NMT system?

end-to-end training

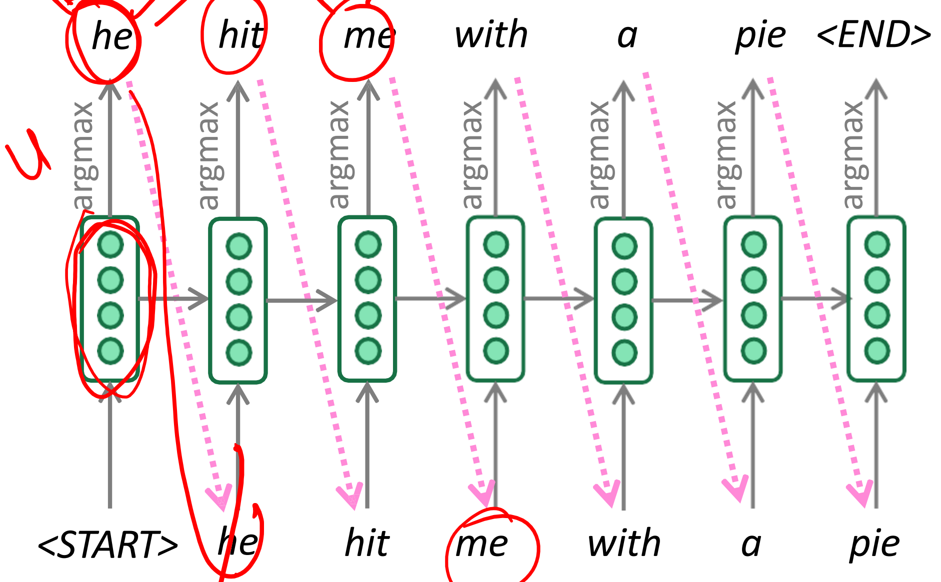
Training an NMT System

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



Greedy decoding

- We saw how to generate (or “decode”) the target sentence by taking argmax on each step of the decoder.



- This is greedy decoding (take most probable word on each step)
- **Problems with this method?**

Problems With Greedy Decoding

- Greedy decoding has no way to undo decisions!
- Input: *il a m'entarté* *(he hit me with a pie)*
- → *he* _____
- → *he hit* _____
- → *he hit a* _____ (whoops! no going back now…)

How to fix this?

Exhaustive Search Decoding

- Ideally we want to find a (length *T*) translation *y* that maximizes

$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots, P(y_T|y_1, \ldots, y_{T-1}, x)$$
$$= \prod_{t=1}^{T} P(y_t|y_1, \ldots, y_{t-1}, x)$$

- We could try computing all possible sequences *y*
- This means that on each step *t* of the decoder, we're tracking V^t possible partial translations, where *V* is vocab size
- This $O(V^T)$ complexity is **far too expensive**!

Beam Search Decoding

- **Core idea:** On each step of decoder, keep track of the *k most probable* partial translations (which we call *hypotheses*)
  - *k* is the beam size (in practice around 5 to 10)
- A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top *k* on each step
- Beam search is not guaranteed to find optimal solution
  - But much more efficient than exhaustive search!

<START

<EOS

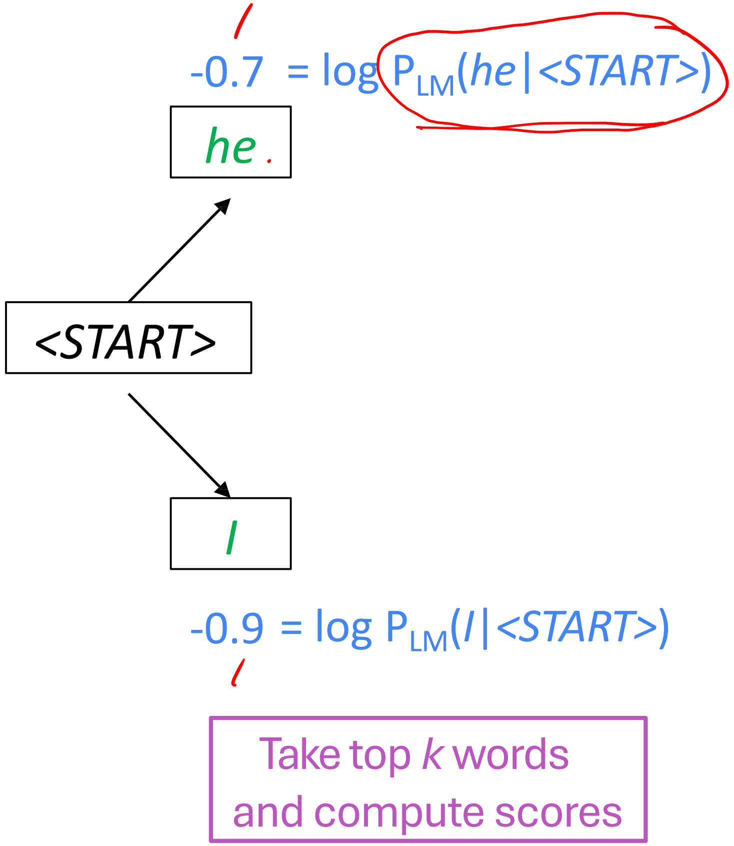
Beam Search Decoding: Example

Beam size = k = 2.

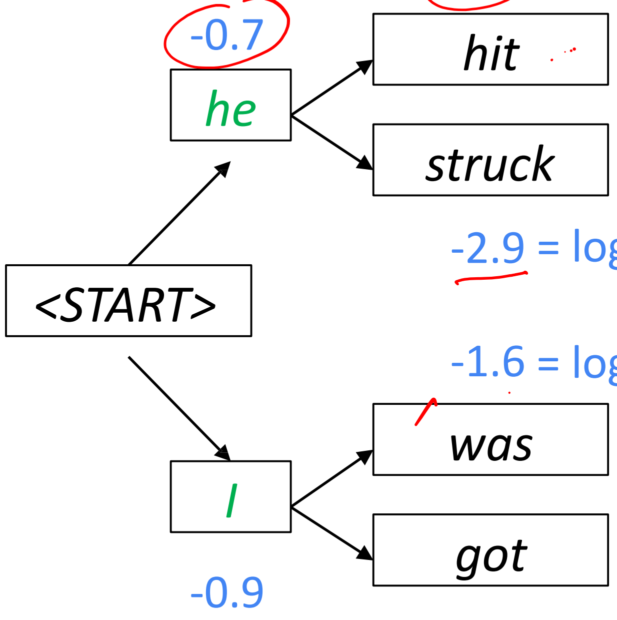
<START>

Calculate prob distribution of next word

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



-1.7 = log P_{LM}(*hit*|*<START> he*) + -0.7

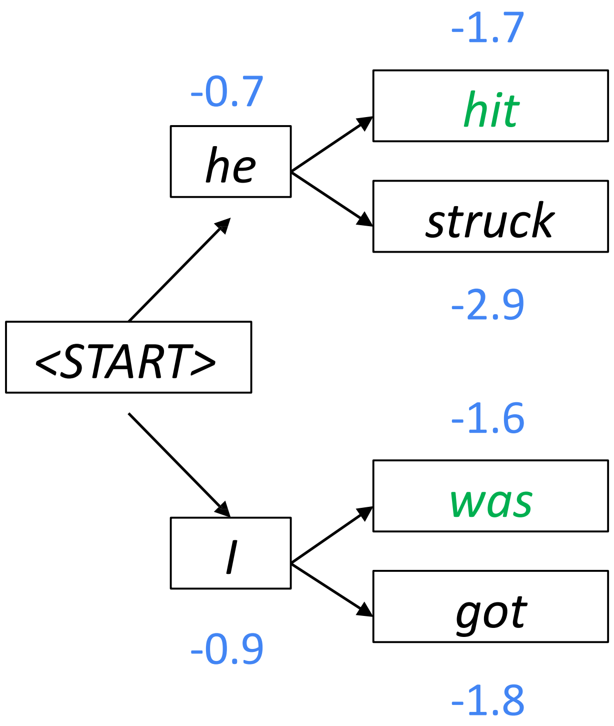
-2.9 = log P_{LM}(*struck*|*<START> he*) + -0.7

-1.6 = log P_{LM}(*was*|*<START> I*) + -0.9

-1.8 = log P_{LM}(*got*|*<START> I*) + -0.9

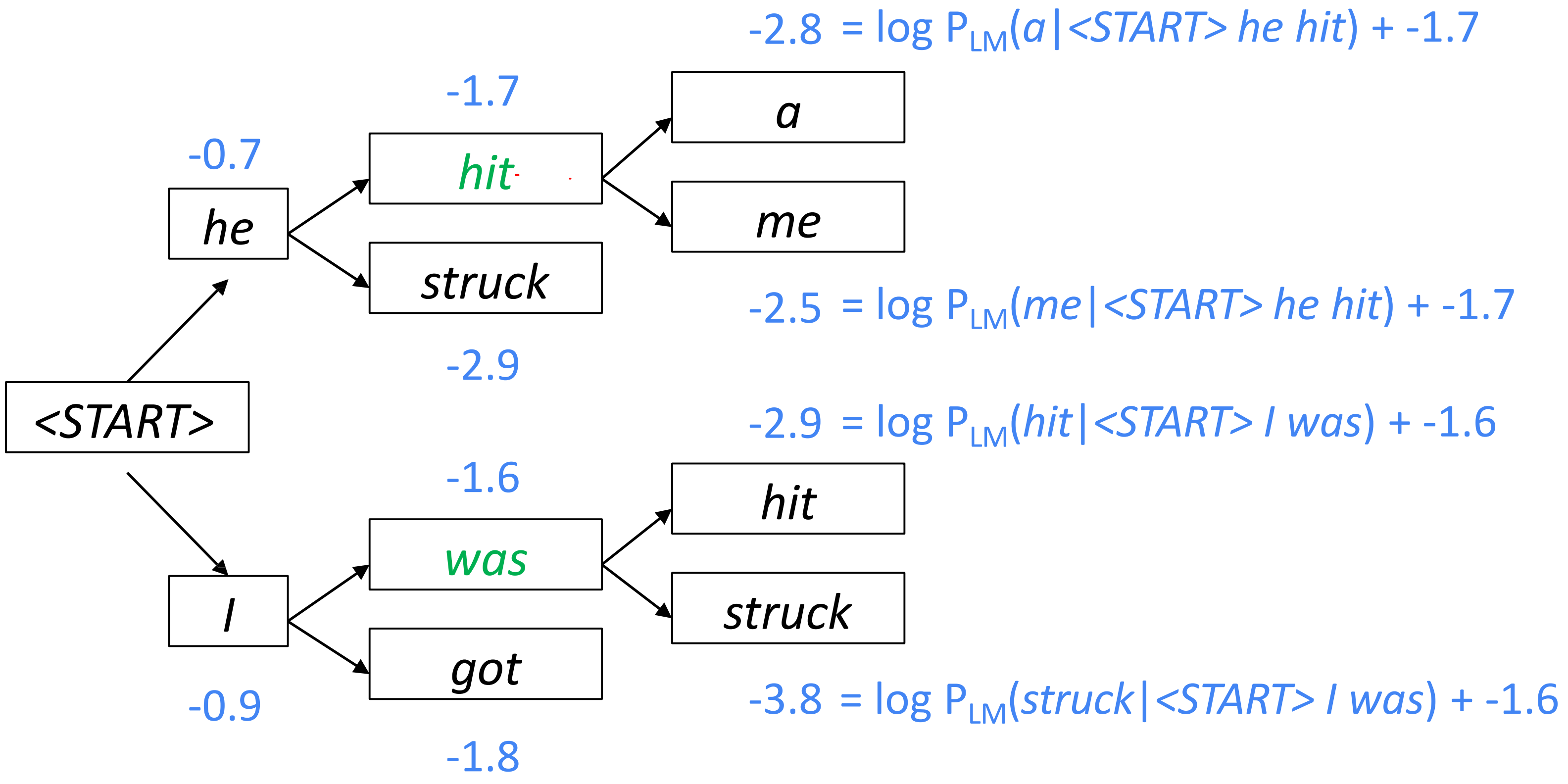
For each of the *k* hypotheses, find
top *k* next words and calculate scores

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



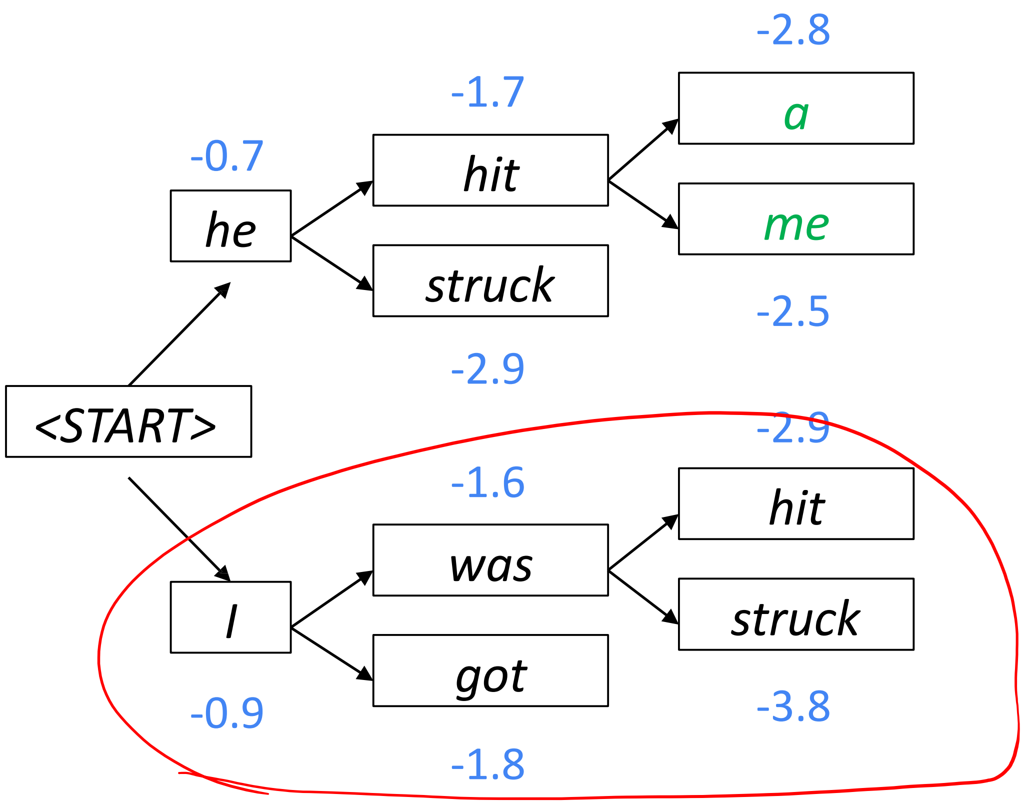
Of these k^2 hypotheses,
just keep *k* with highest scores

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



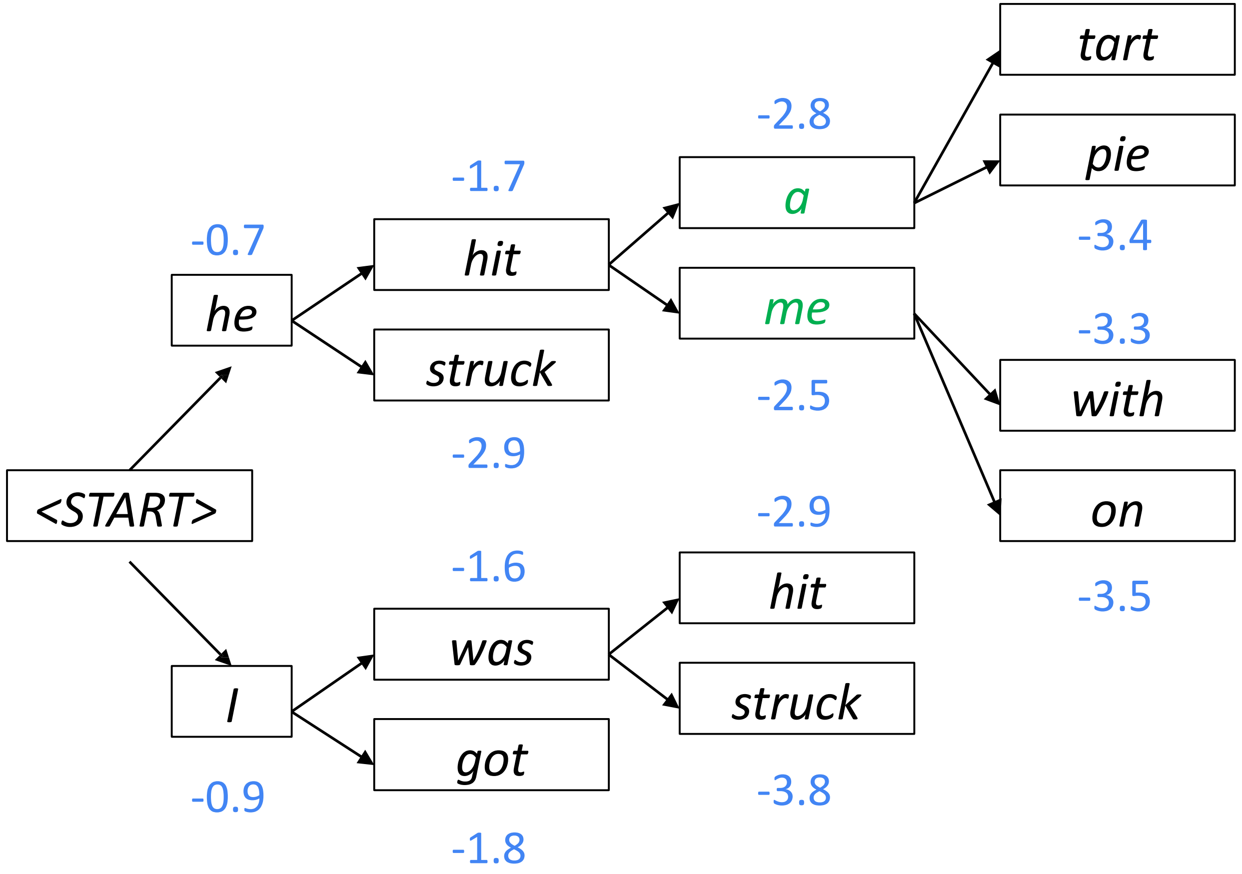
For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



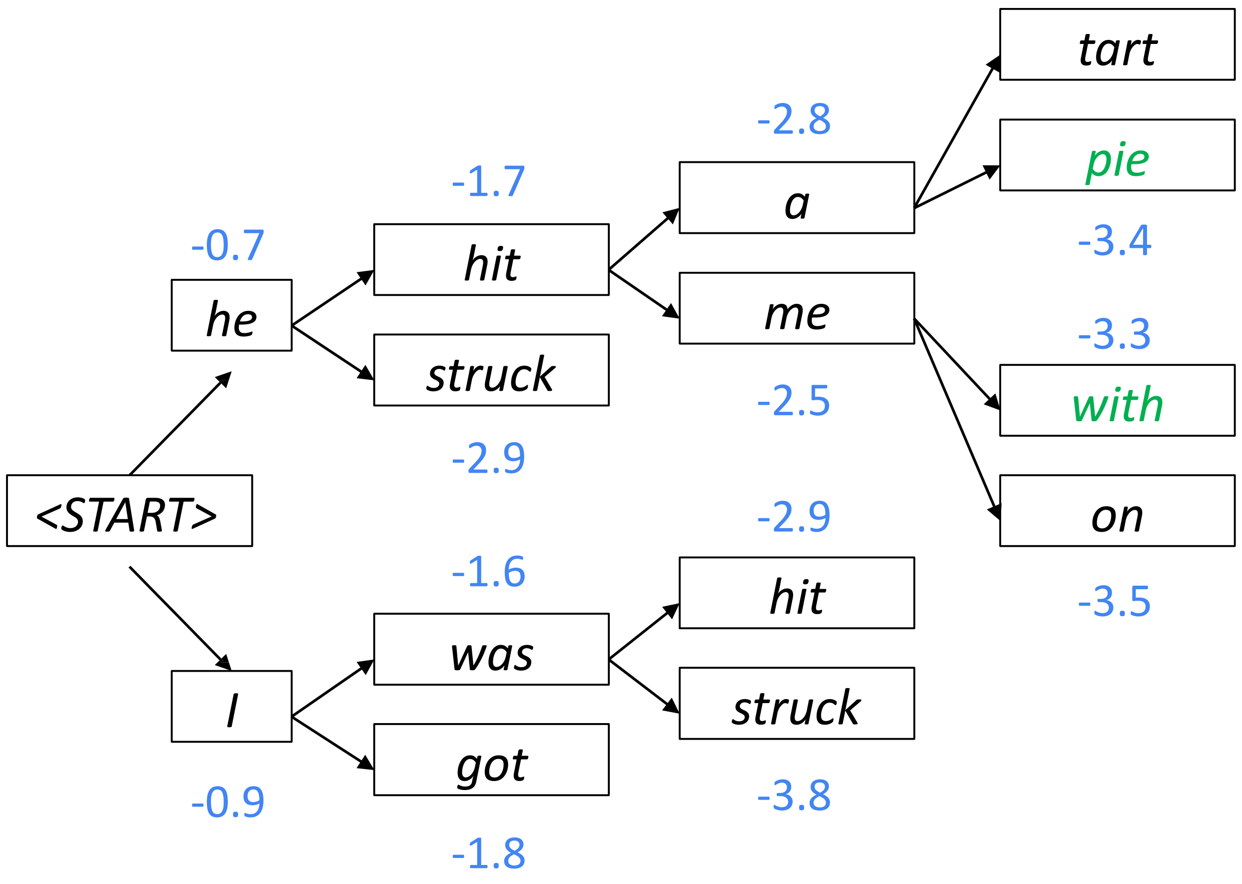
Of these k^2 hypotheses,
just keep *k* with highest scores

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



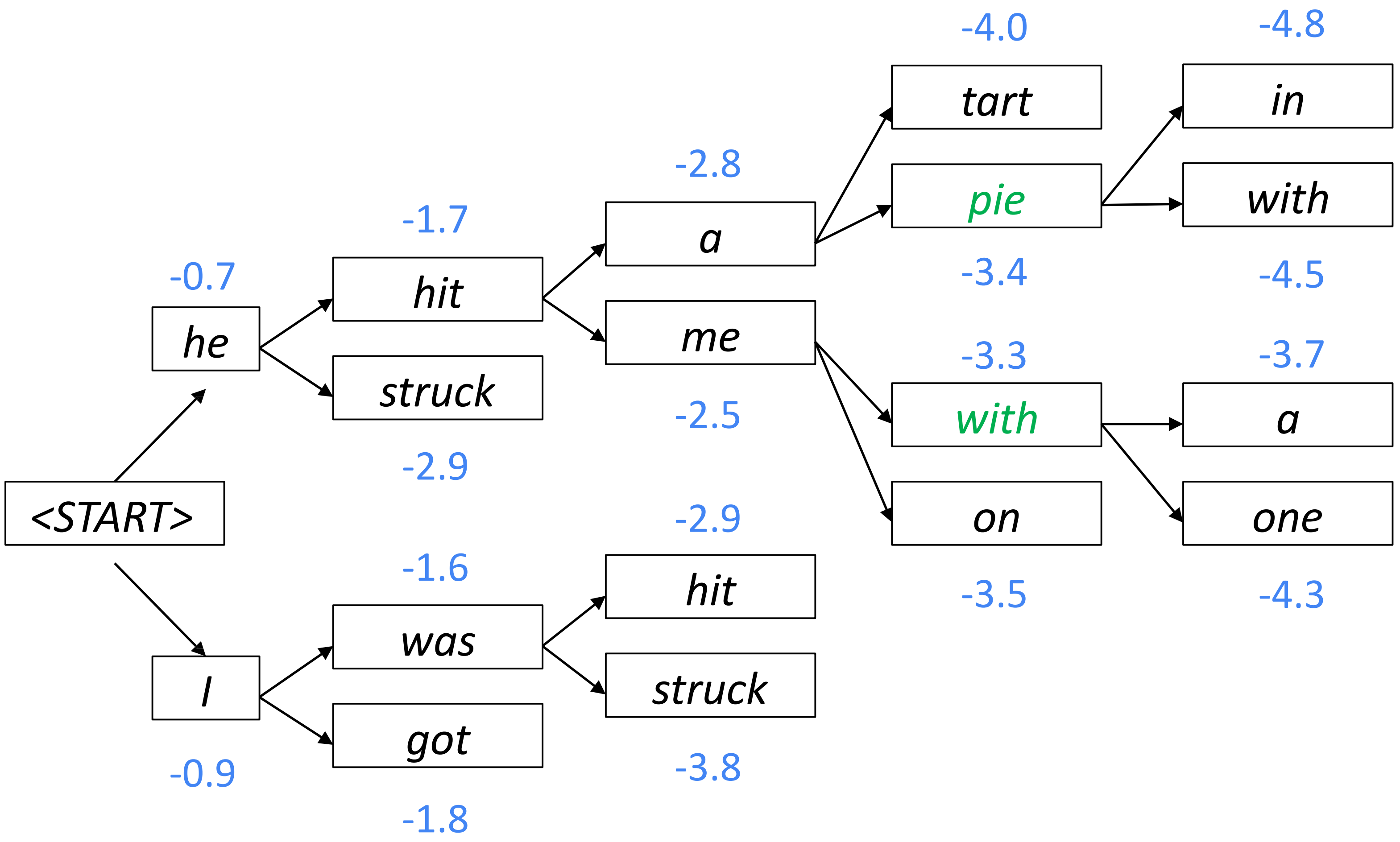
For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



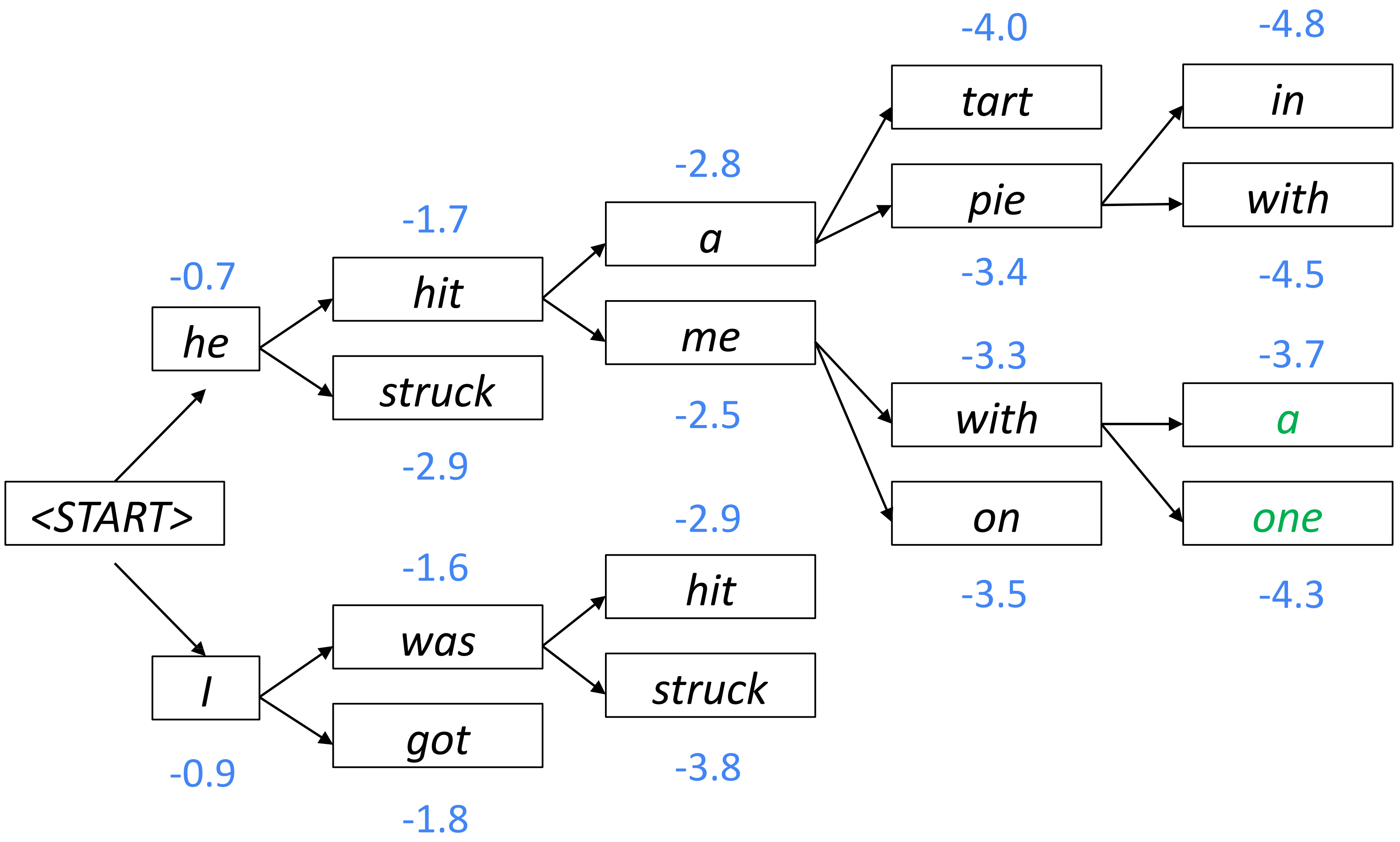
Of these k^2 hypotheses,
just keep k with highest scores

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



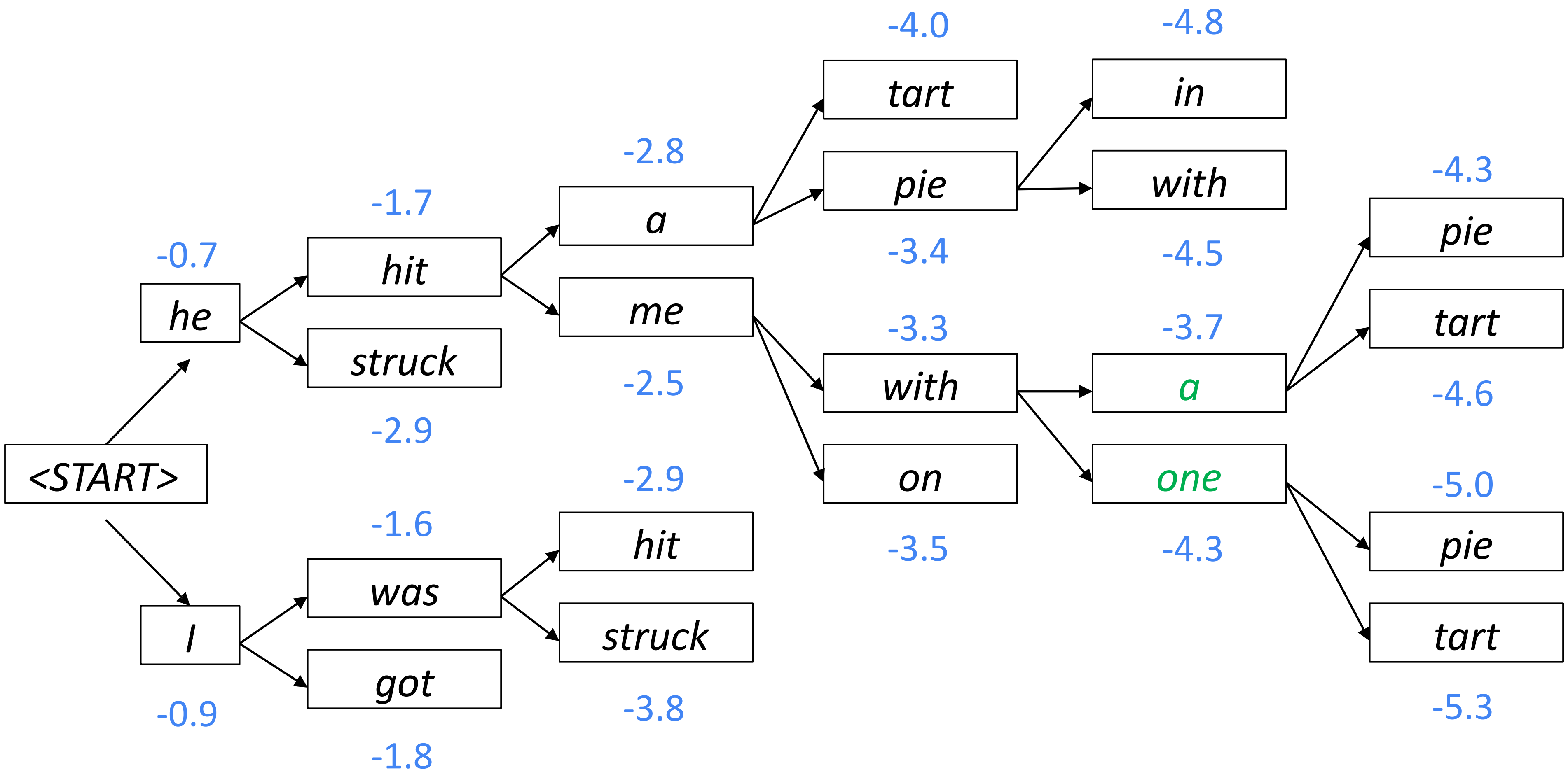
For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



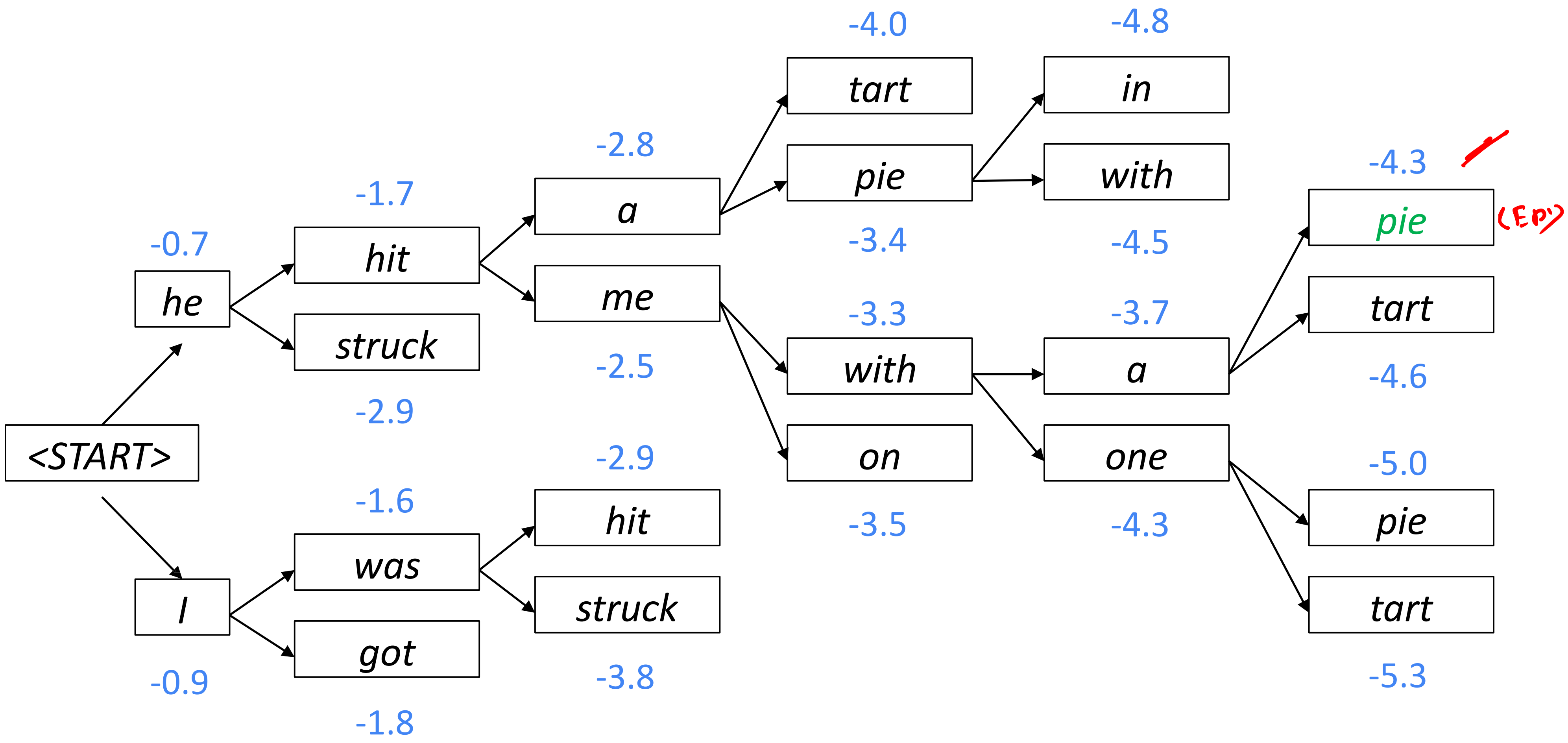
Of these k^2 hypotheses,
just keep *k* with highest scores

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



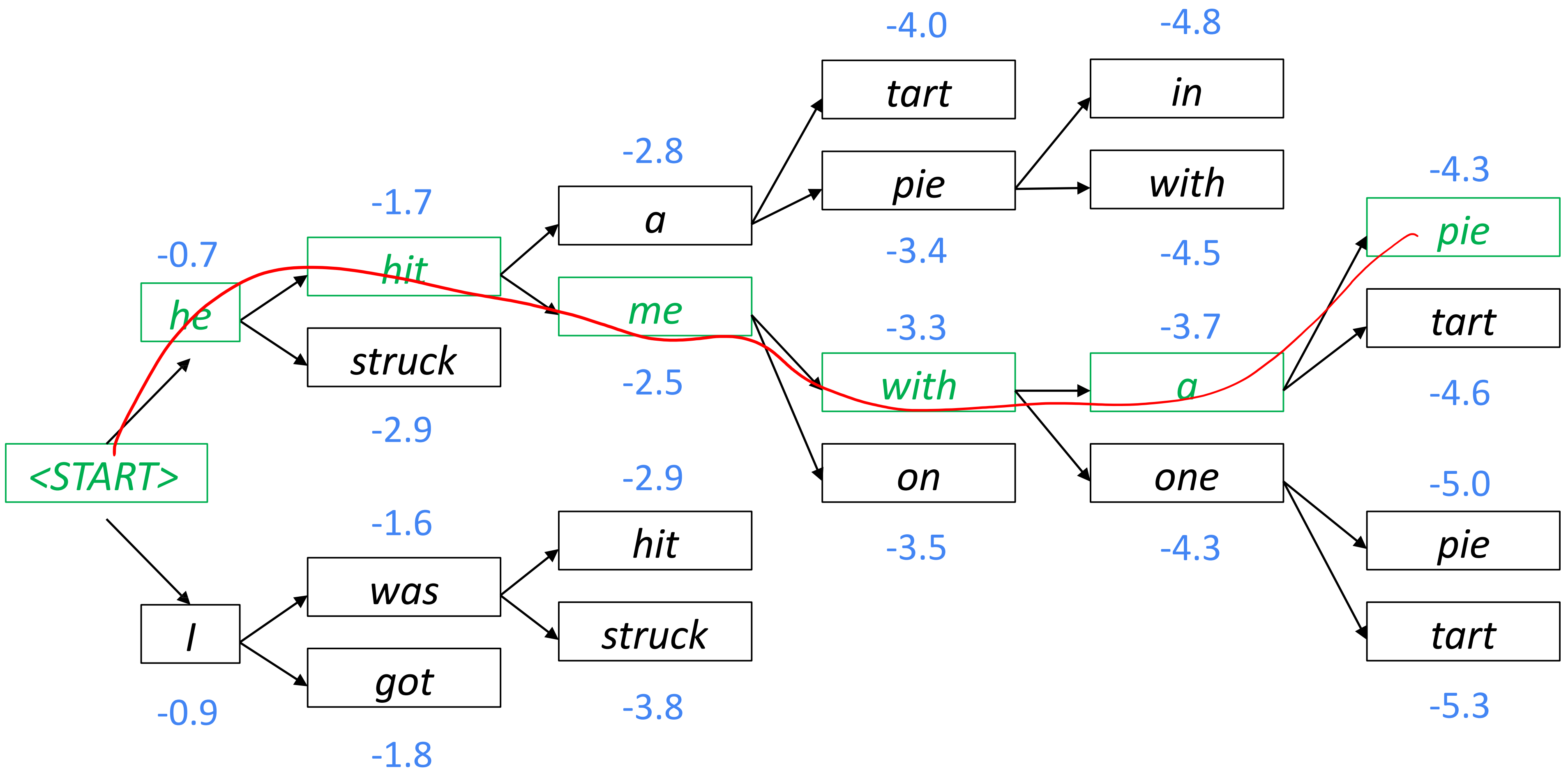
For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



This is the top-scoring hypothesis!

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Backtrack to obtain the full hypothesis

Beam Search Decoding: Stopping Criterion

- In greedy decoding, usually we decode until the model produces a <END> token
  - **For example:** <START> he hit me with a pie <END>
- In beam search decoding, different hypotheses may produce <END> tokens on different timesteps
  - When a hypothesis produces <END>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.
- Usually we continue beam search until:
  - We reach timestep T (where T is some pre-defined cutoff), or
  - We have at least n completed hypotheses (where n is pre-defined cutoff)

Beam Search Decoding: Finishing Up

- We have our list of completed hypotheses.
- How to select top one with highest score?
- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

$$\text{score}(y_1, \ldots, y_t) = \underline{\log P_{\text{LM}}(y_1, \ldots, y_t|x)} = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1, \ldots, y_{i-1}, x)$$

- **Problem:** longer hypotheses have lower scores
- **Fix:** Normalize by length. Use this to select the top one instead:

$$\frac{1}{t}\sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1, \ldots, y_{i-1}, x)$$

NMT: The First Big Success Story of NLP Deep Learning

Neural Machine Translation went from a fringe research attempt in 2014 to the leading standard method in 2016

- 2014: First seq2seq paper published [Sutskever et al. 2014]
- 2016: Google Translate switches from SMT to NMT – and by 2018 everyone had
  - https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html

- This was amazing!
  - SMT systems, built by hundreds of engineers over many years, were outperformed by NMT systems trained by small groups of engineers in a few months

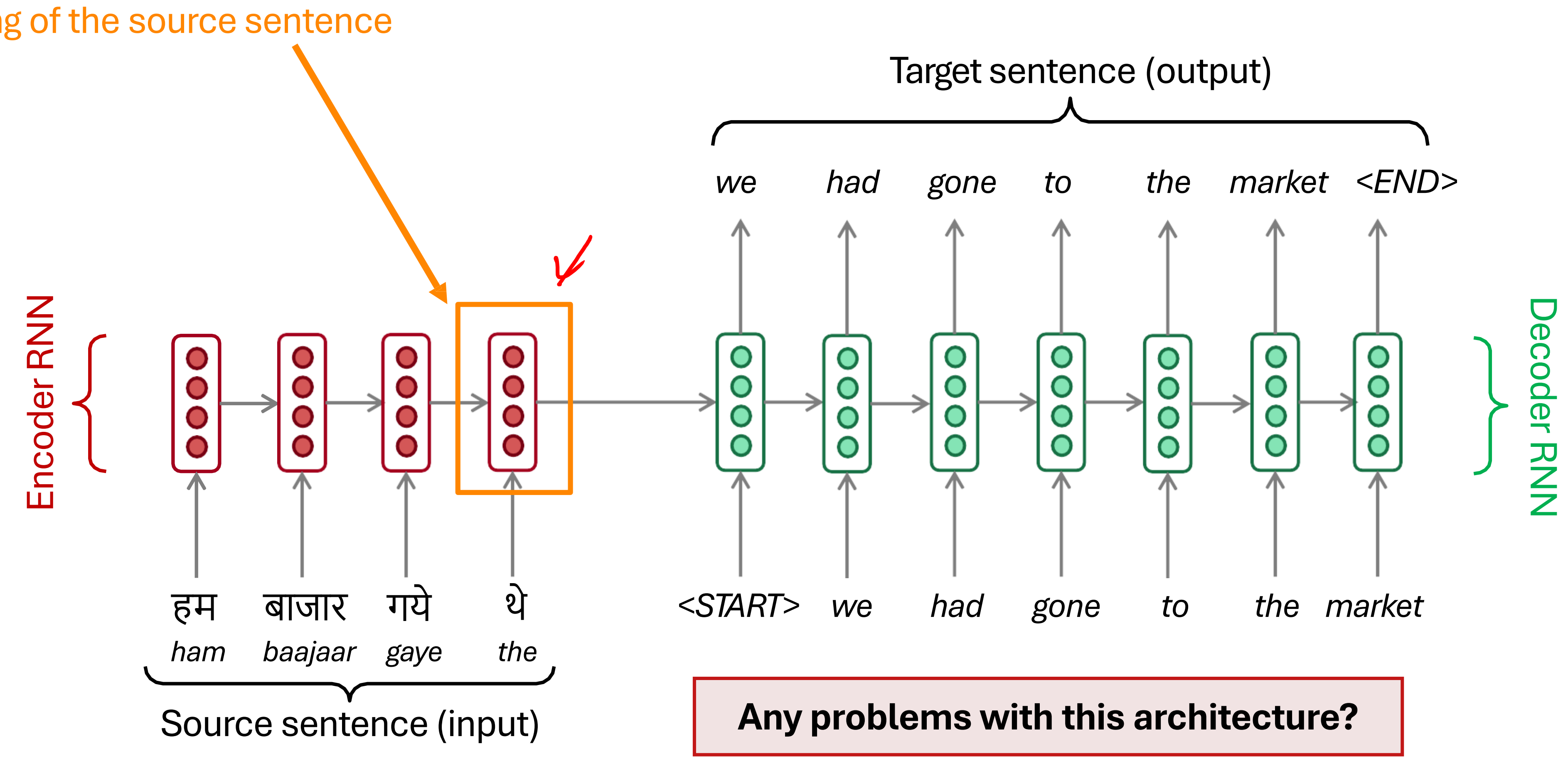
Issues With RNN

- Linear interaction distance
- Bottleneck problem
- Lack of parallelizability

ATTENTION

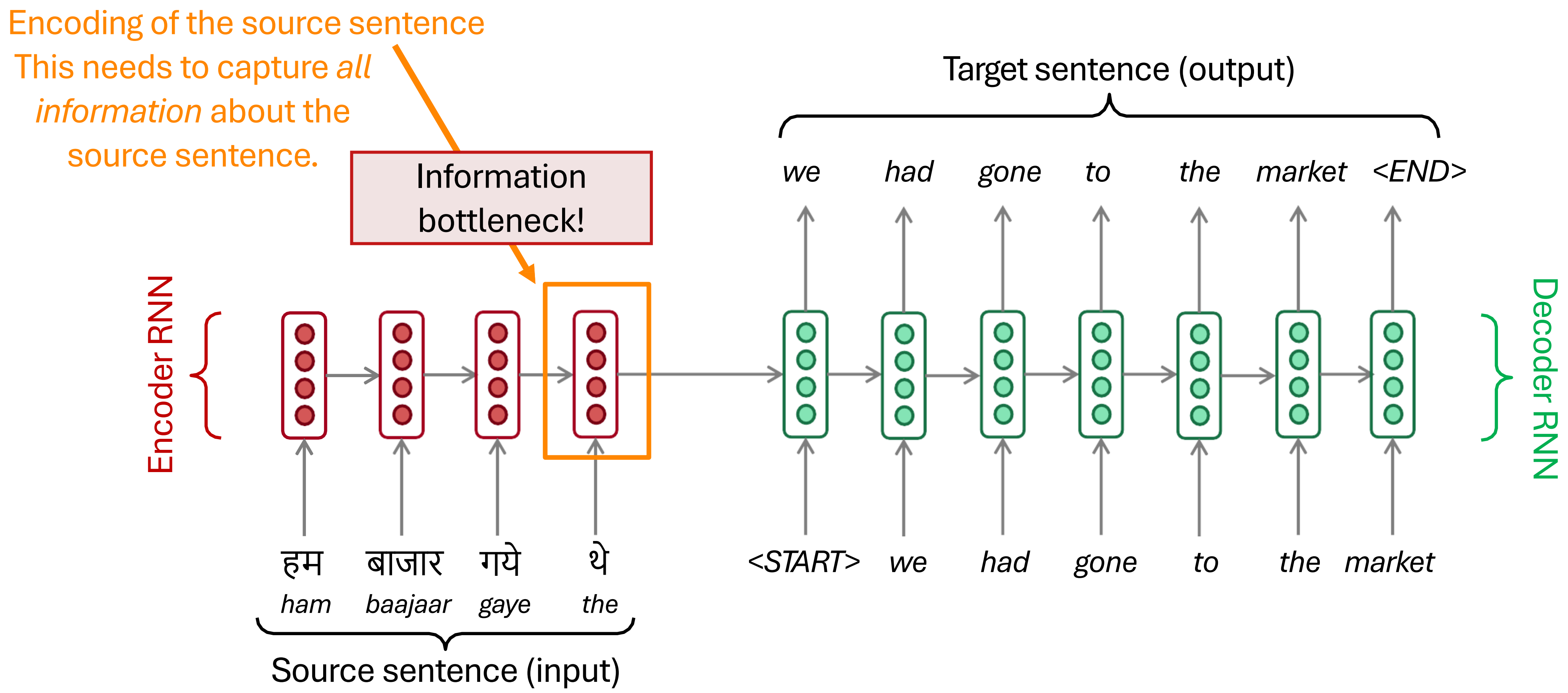
Attention

Sequence-to-Sequence: The Bottleneck Problem



Any problems with this architecture?

Sequence-to-Sequence: The Bottleneck Problem



Attention

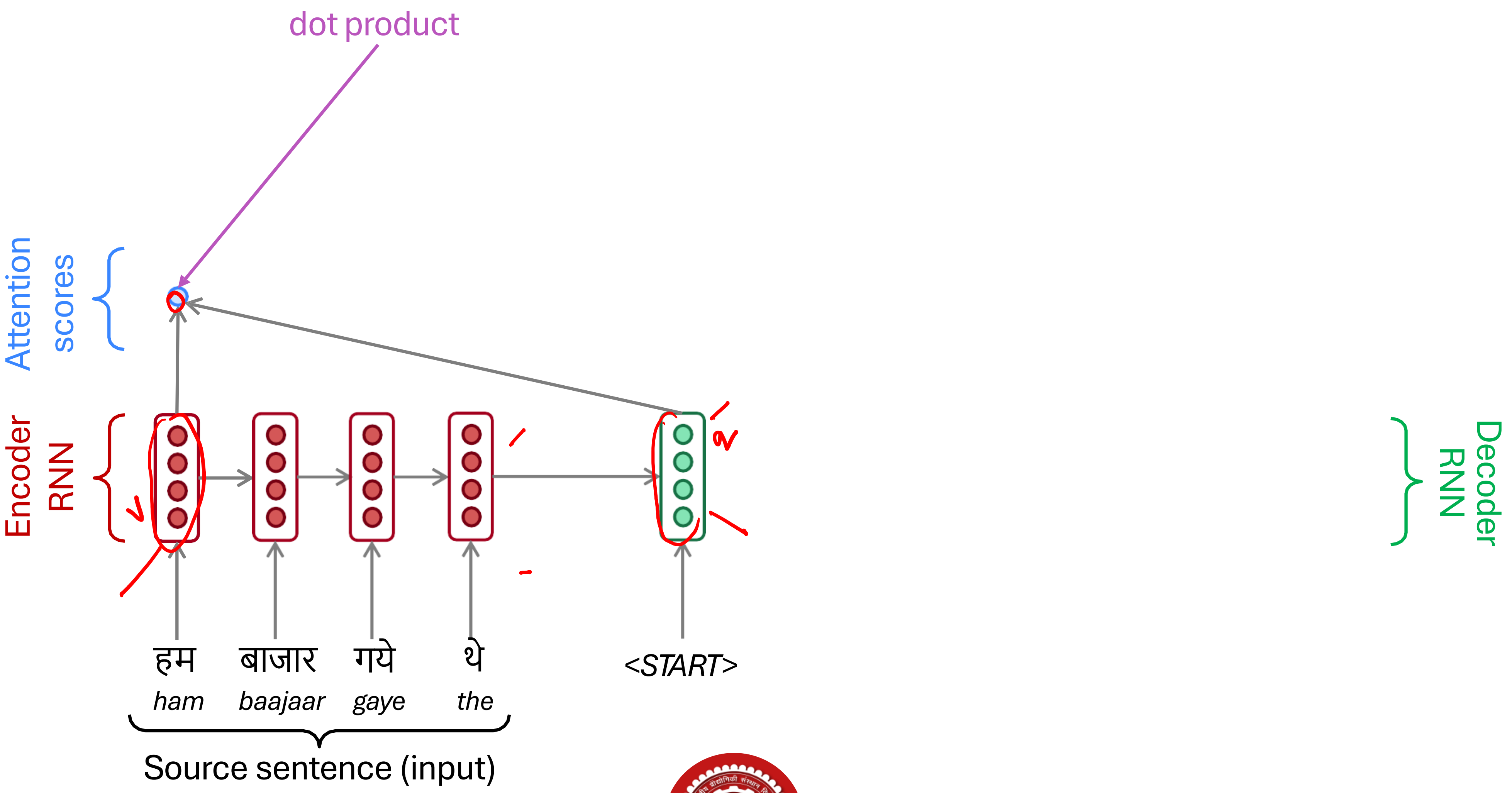
- **Attention** provides a solution to the bottleneck problem.

- **Core idea**: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence

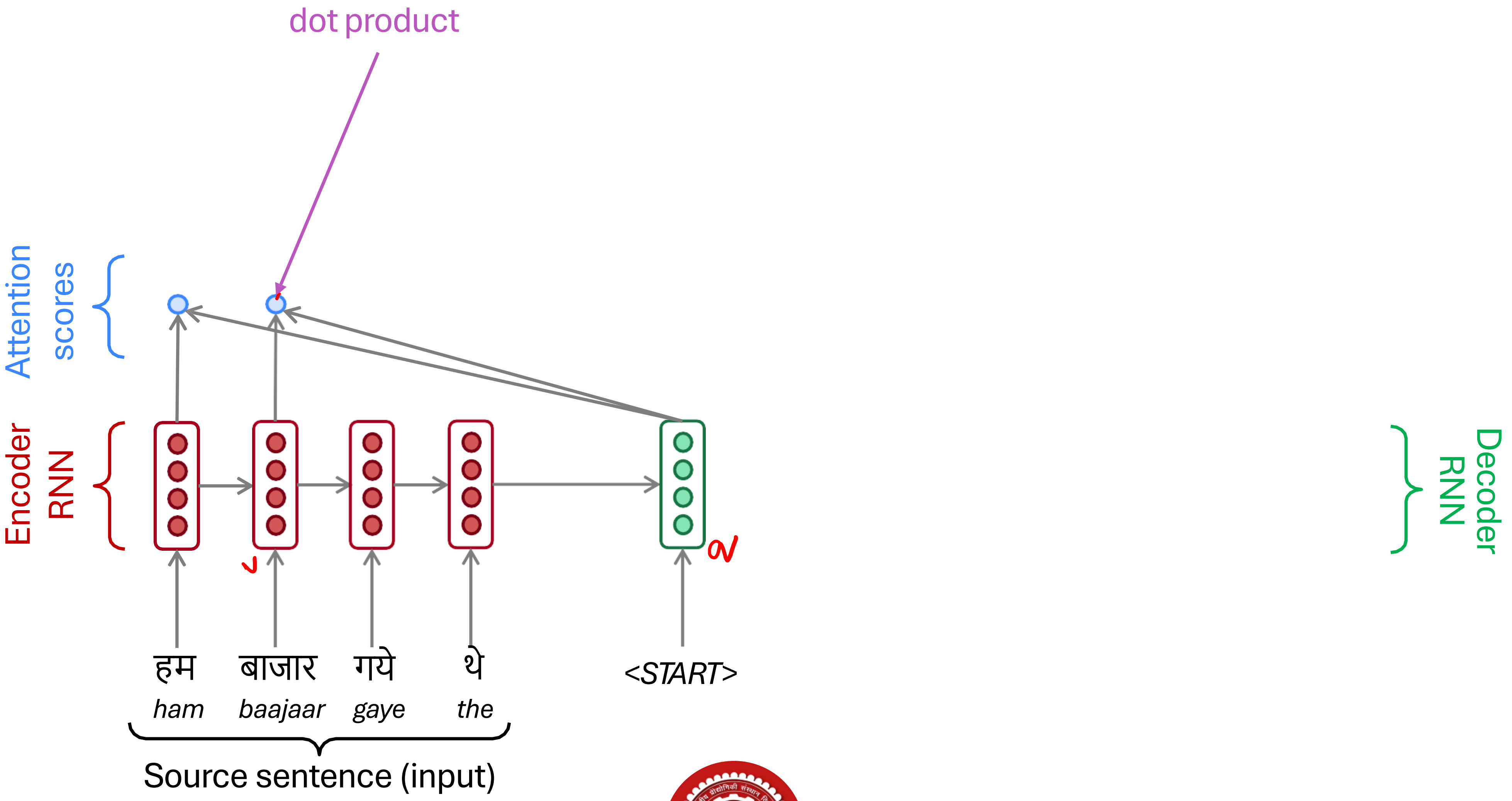
- Let's start with the visualization of the attention mechanism.

(query value)

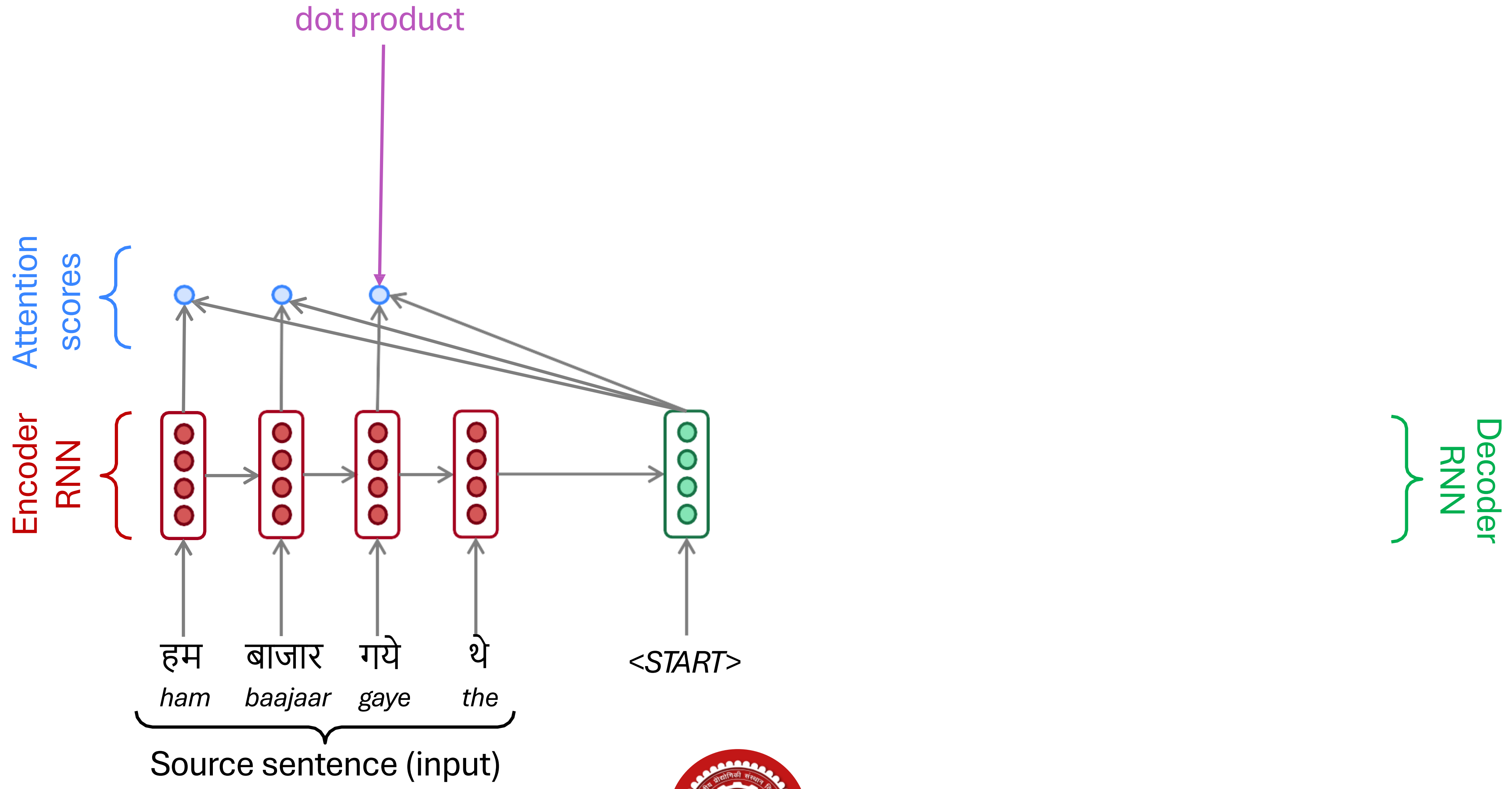
Sequence-to-Sequence With Attention



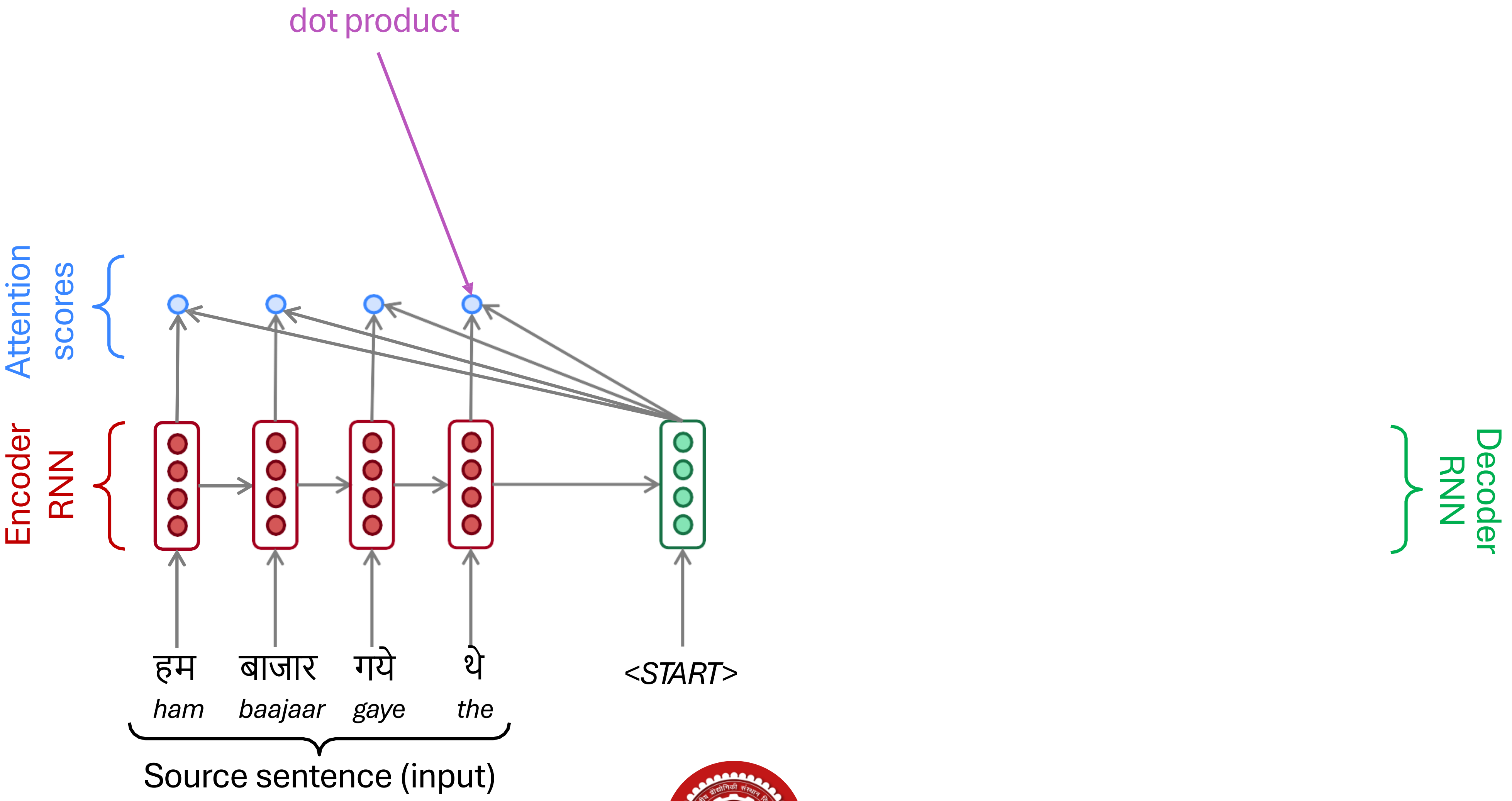
Sequence-to-Sequence With Attention



Sequence-to-Sequence With Attention



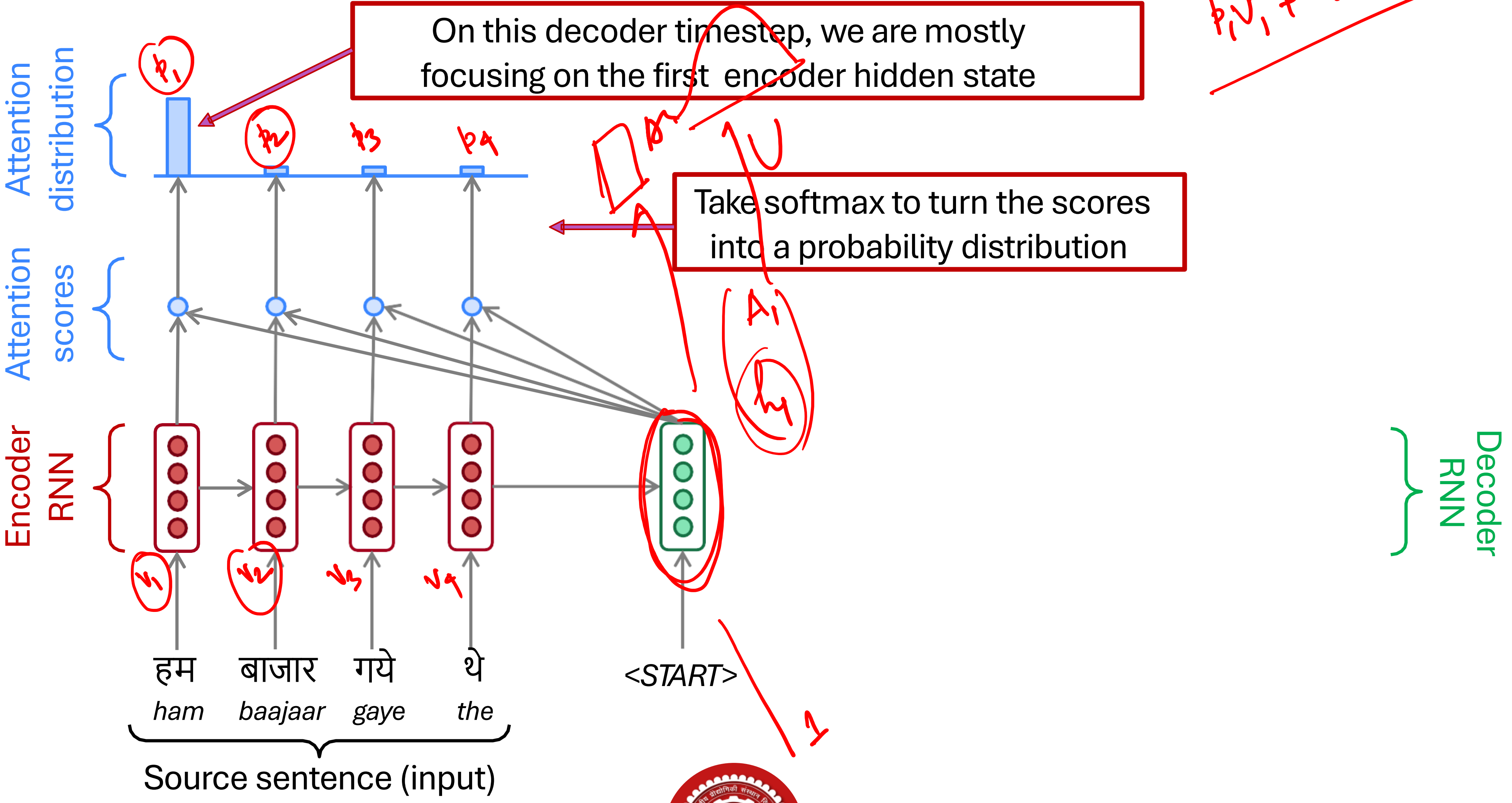
Sequence-to-Sequence With Attention



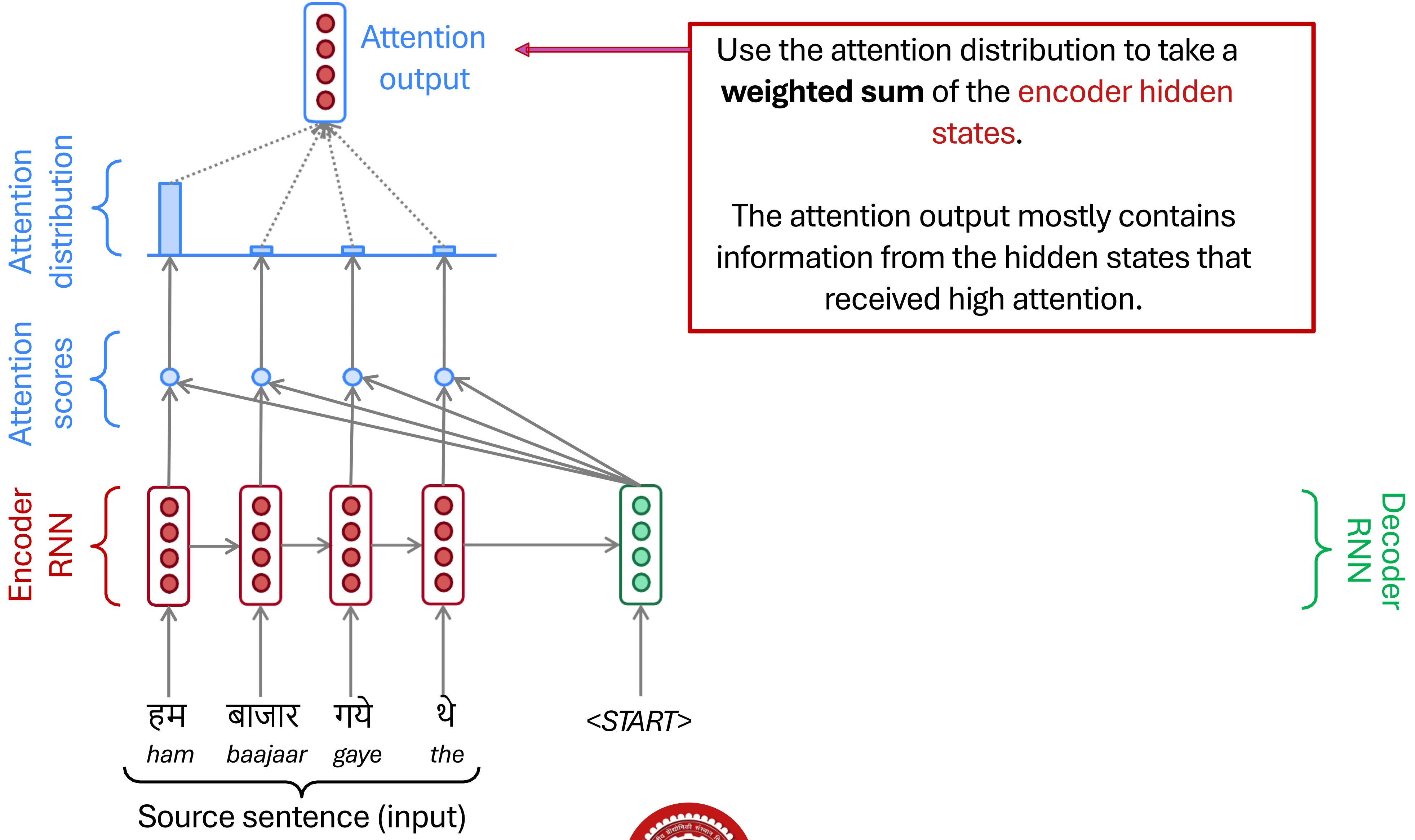
Sequence-to-Sequence With Attention

Attention vector

$p_1V_1 + p_2V_2 + p_3V_3 + p_4V_4$



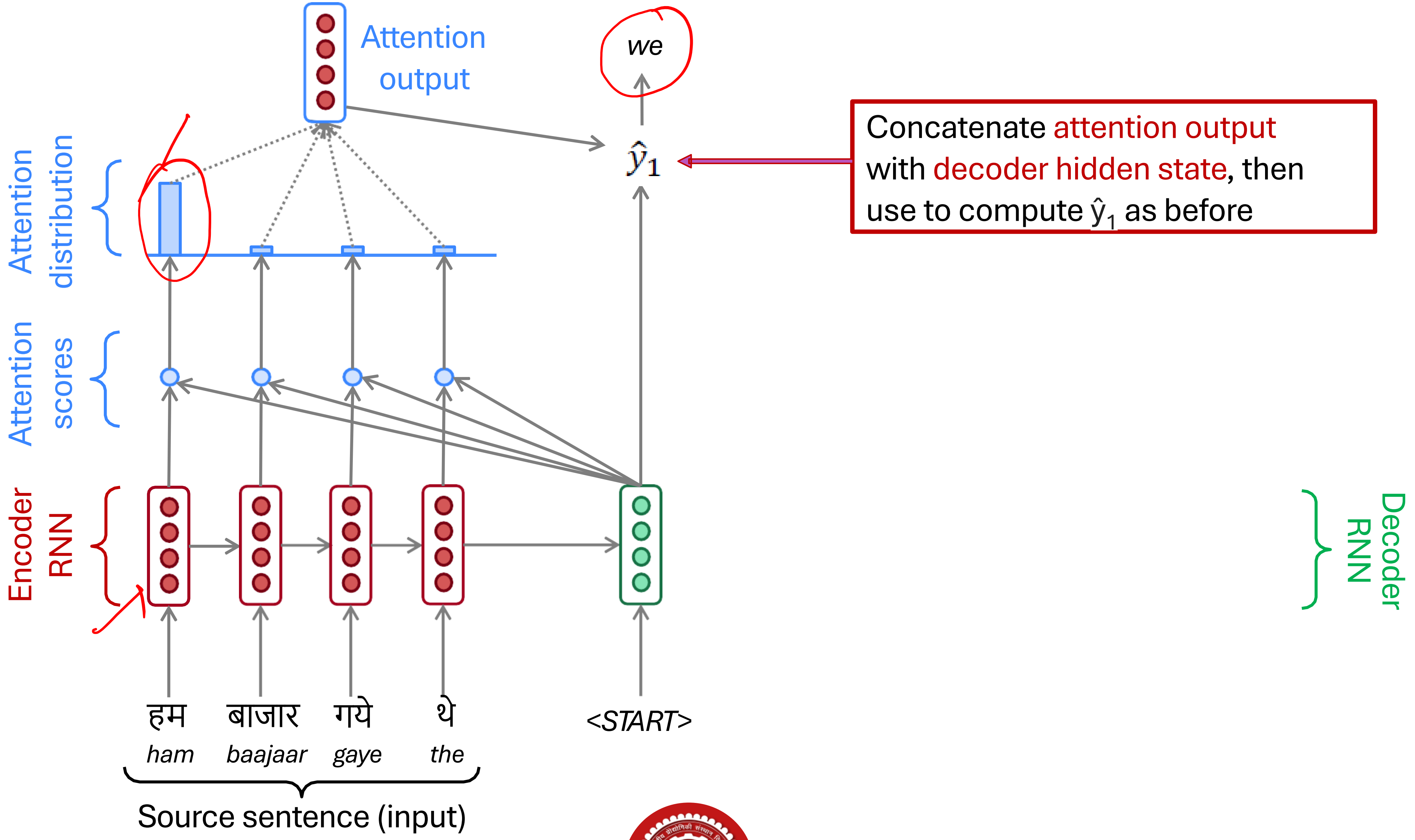
Sequence-to-Sequence With Attention



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

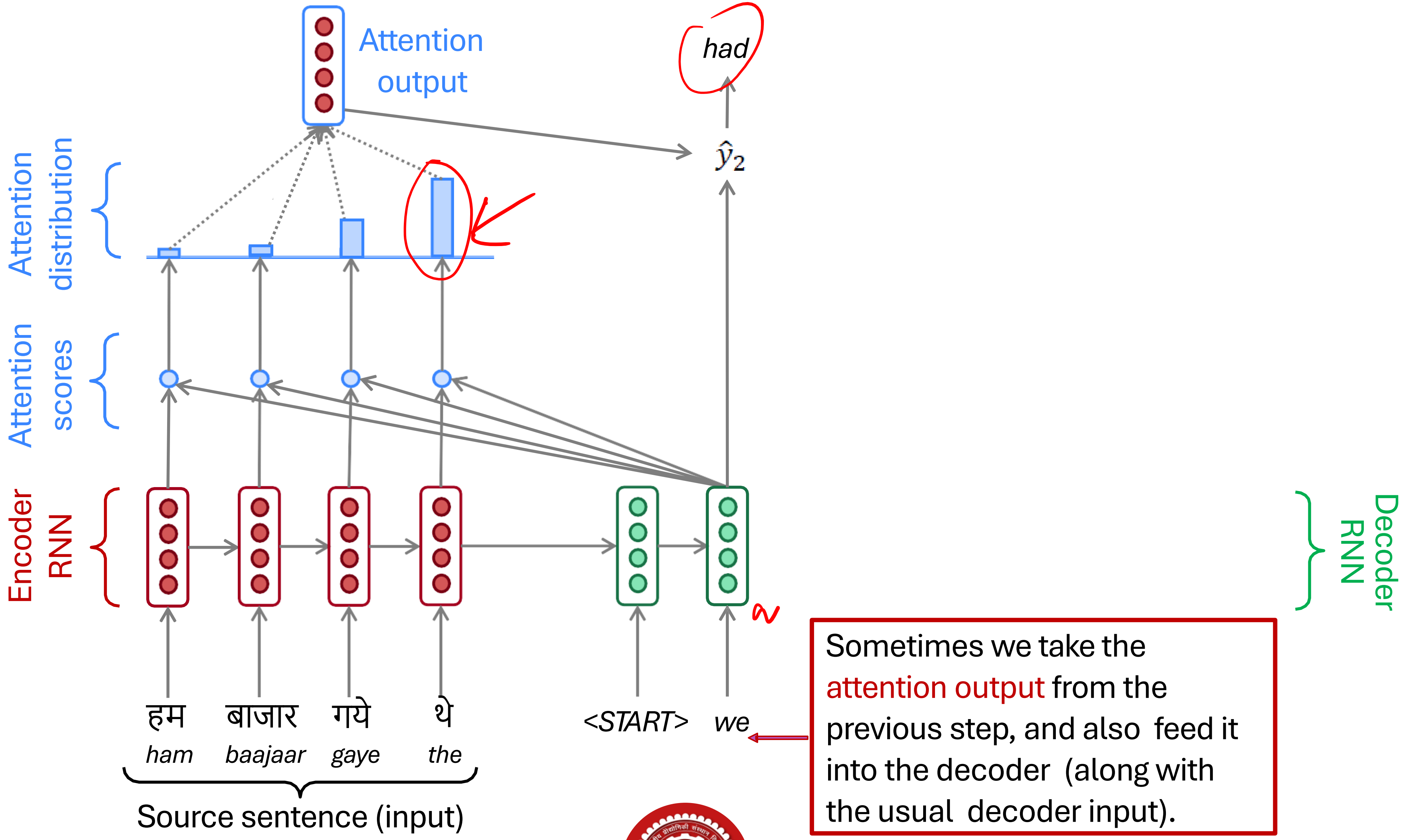
The attention output mostly contains information from the hidden states that received high attention.

Sequence-to-Sequence With Attention



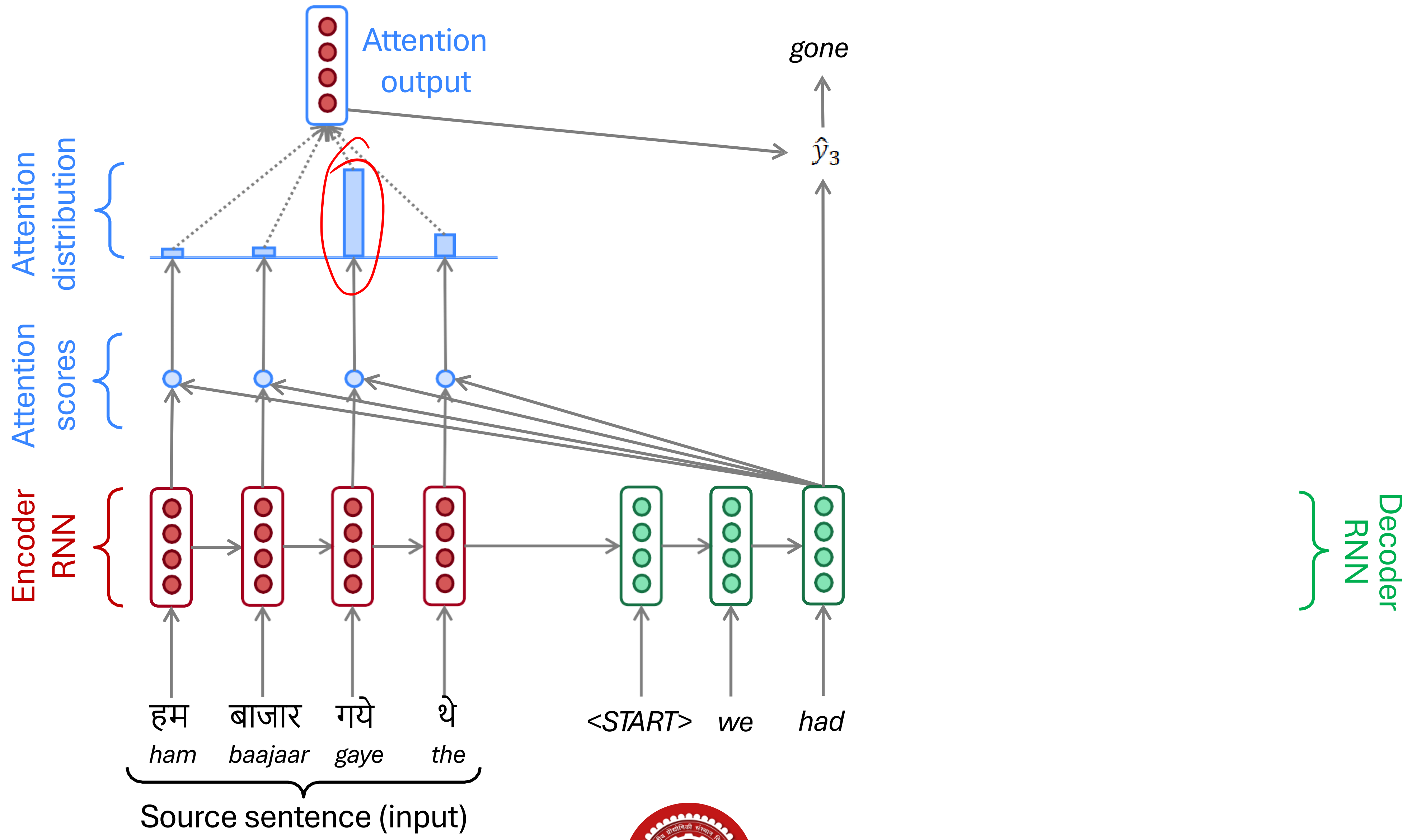
Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

Sequence-to-Sequence With Attention

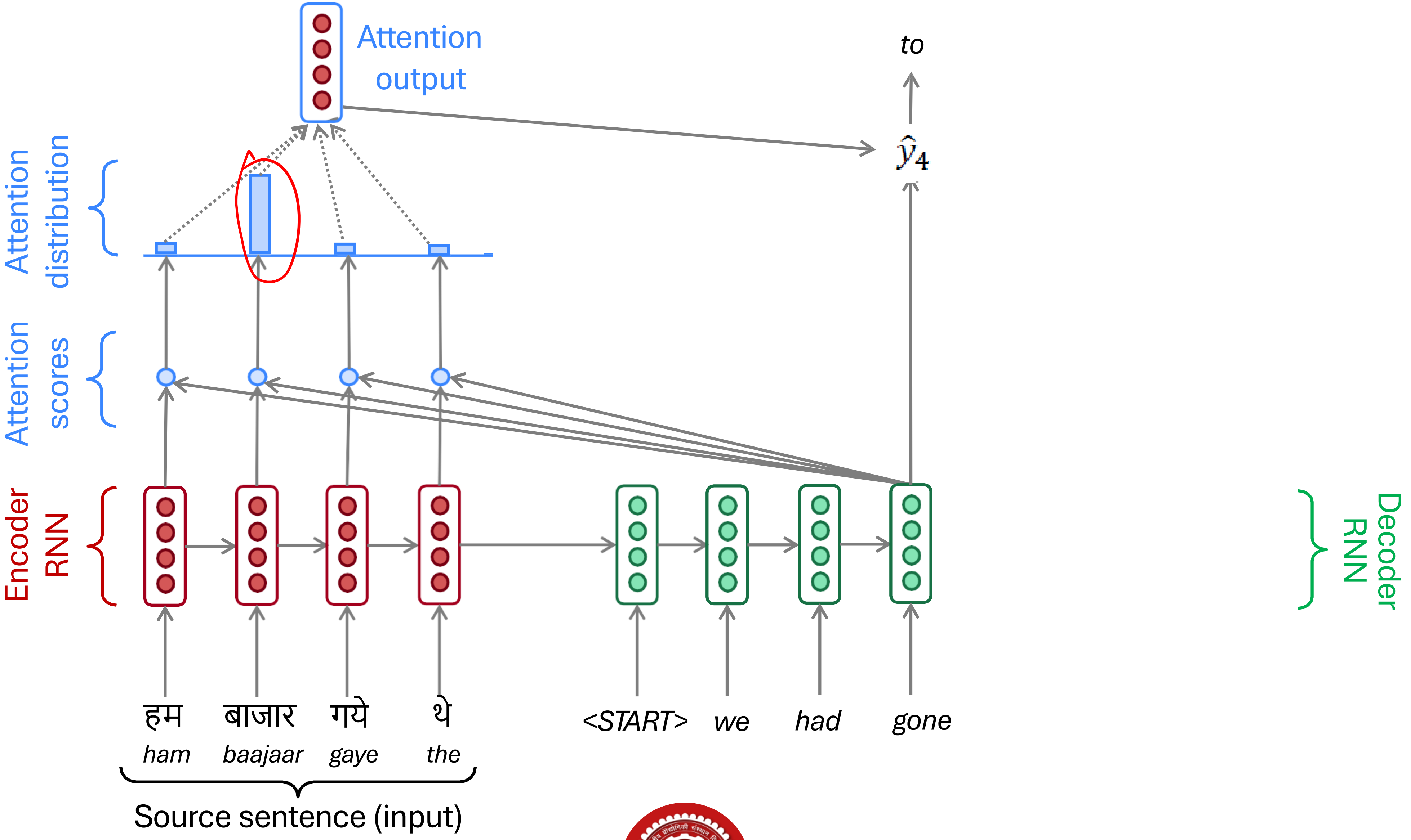


Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input).

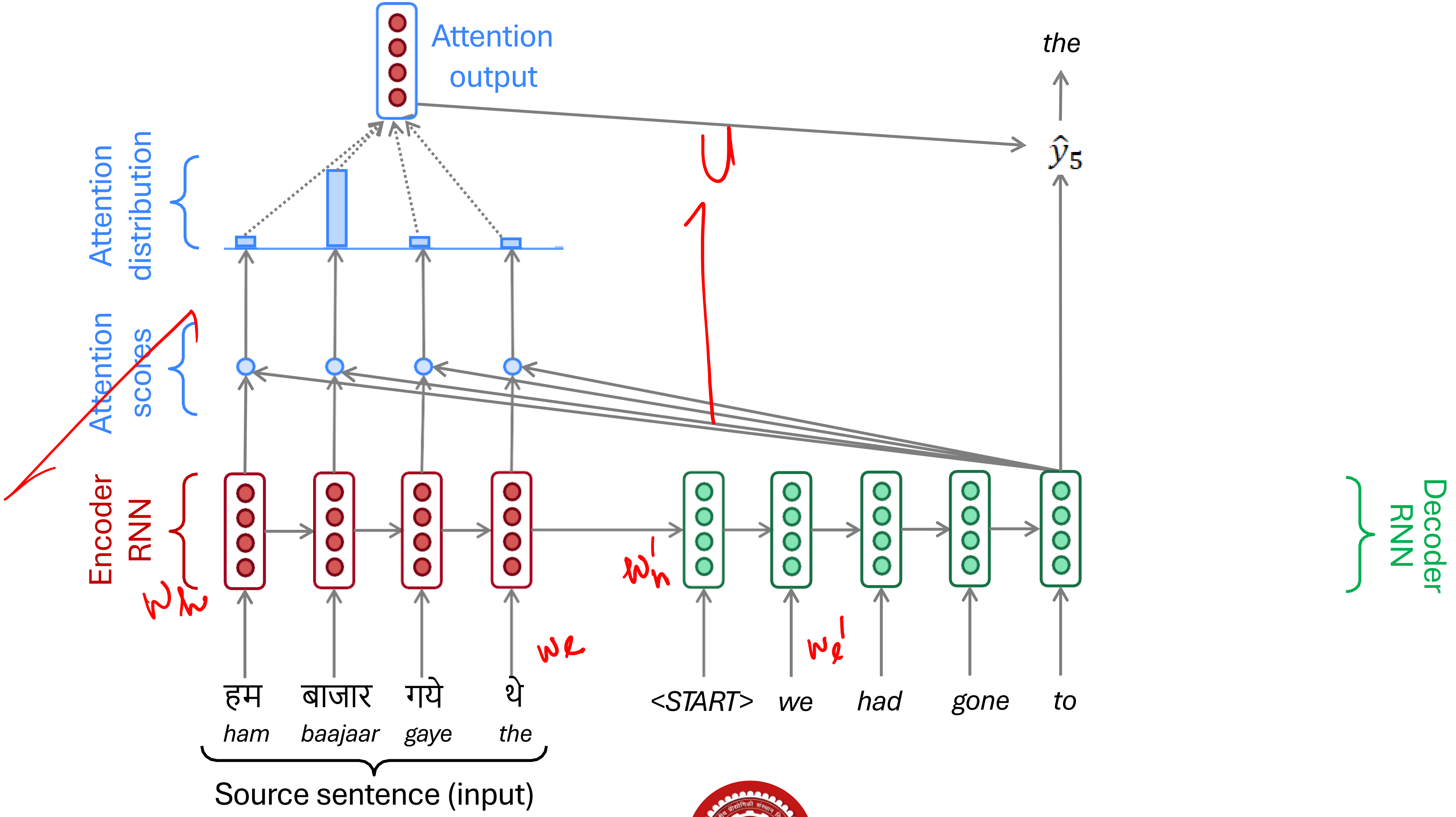
Sequence-to-Sequence With Attention



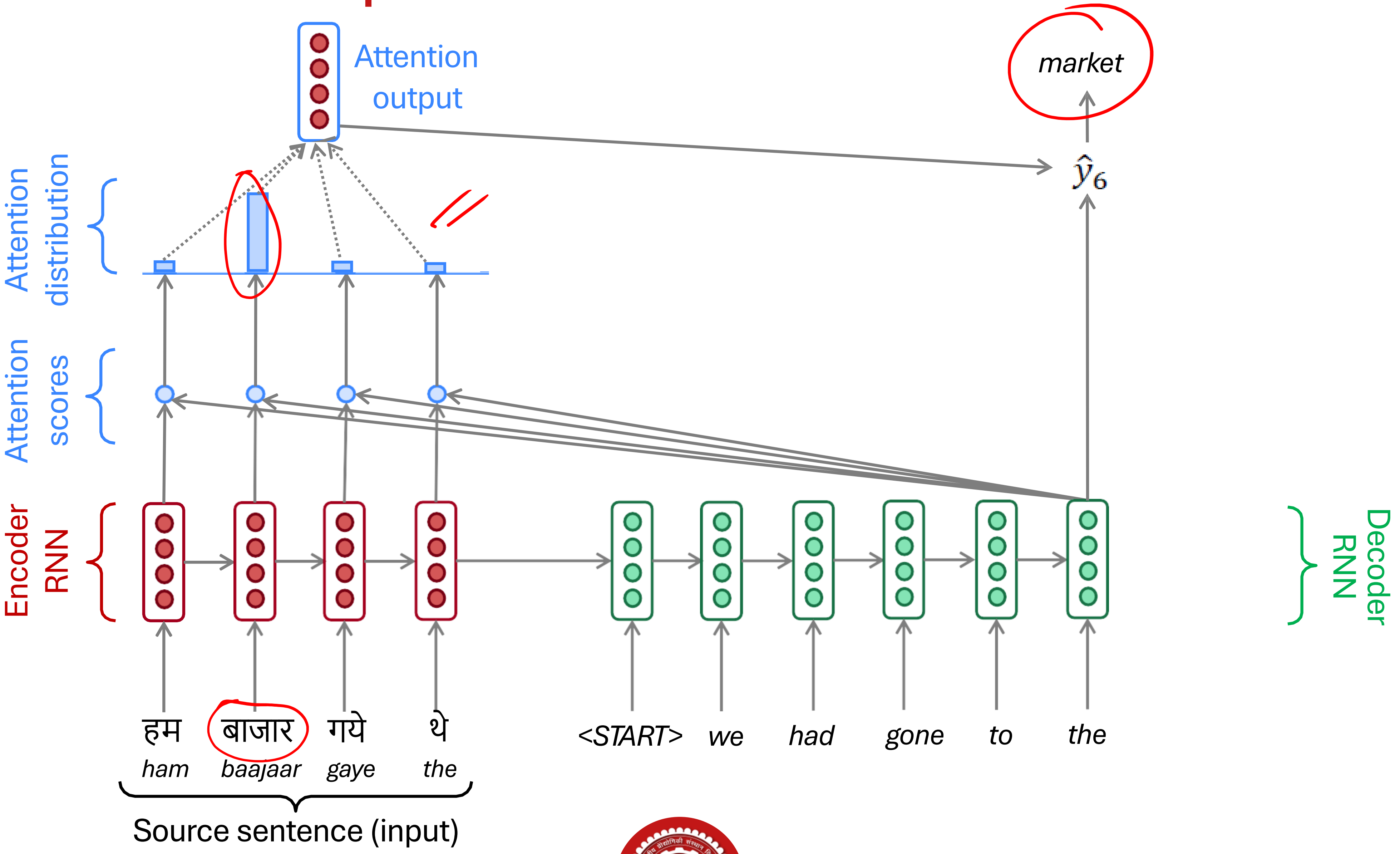
Sequence-to-Sequence With Attention



Sequence-to-Sequence With Attention



Sequence-to-Sequence With Attention



Attention: In Equations

$S_t h_1$, $S_t h_2$,

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution, sums to 1)

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output a_t

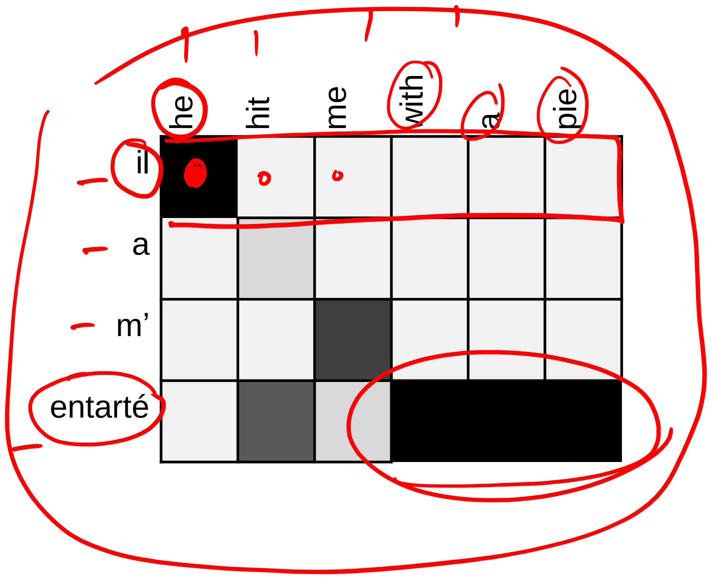
$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output a_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

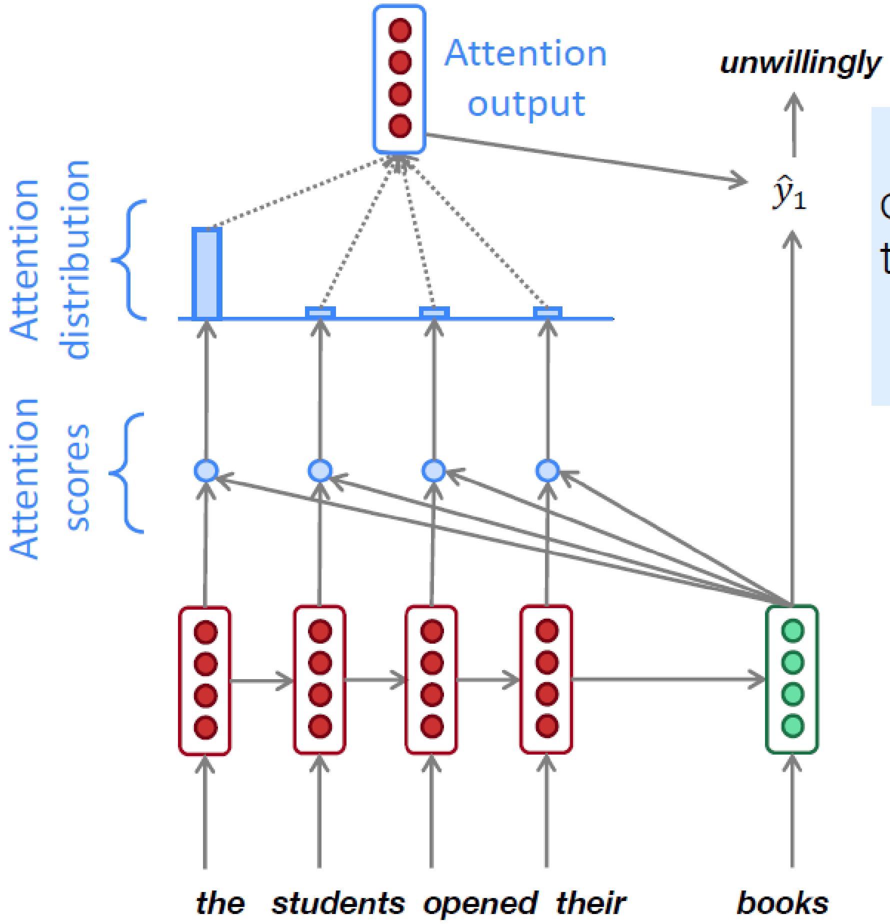
$$[a_t; s_t] \in \mathbb{R}^{2h}$$

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



Seq2Seq+Attention for LM

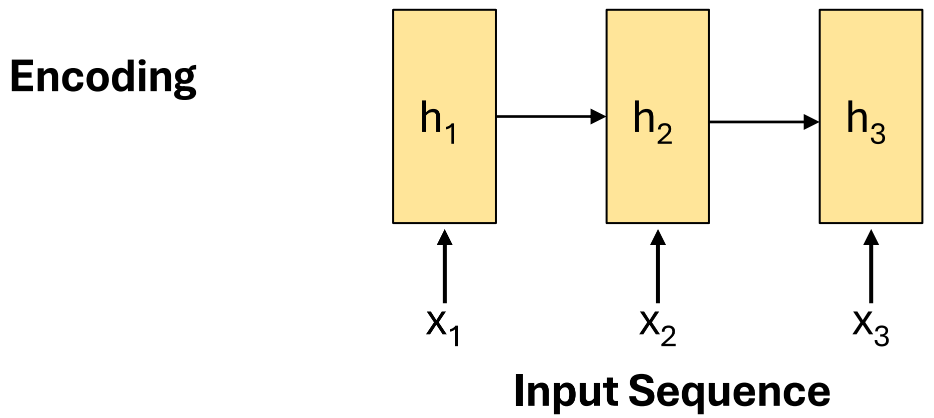


Concatenate (or otherwise compose) the attention output with the current hidden state, then pass through a softmax layer to predict the next word

Attention is a *General* Deep Learning Technique

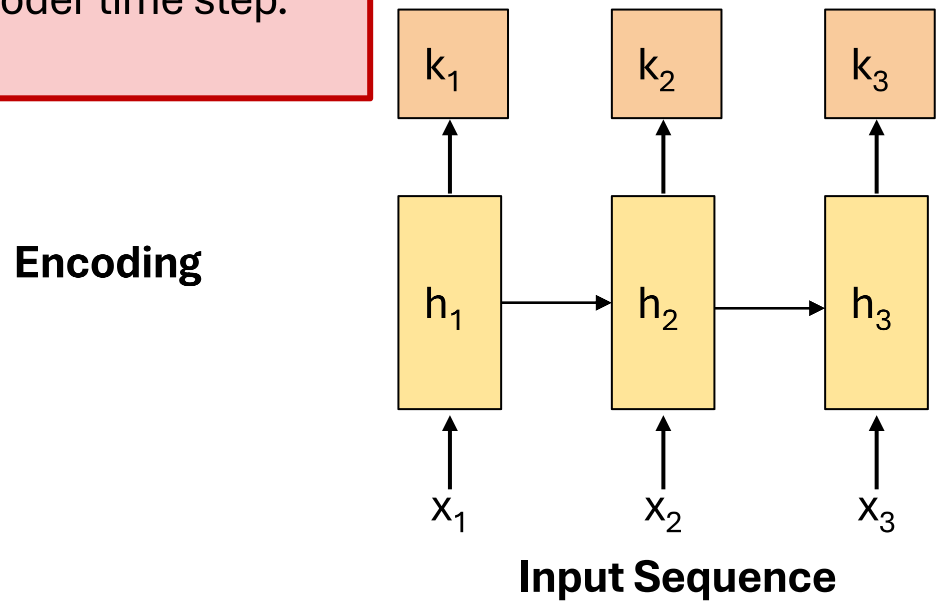
- We've seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- <u>However</u>: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)
- More general definition of attention:
  - Given a set of vector *values*, and a 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).

Attention

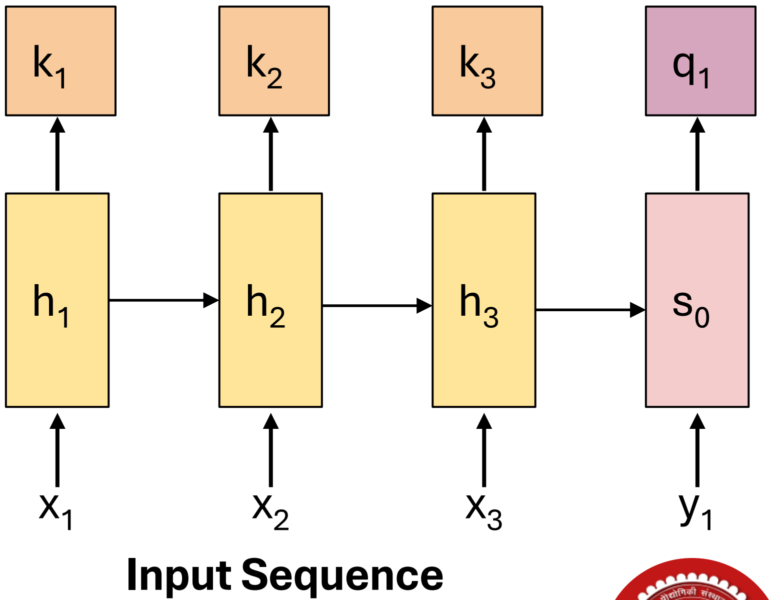


Attention

Key vectors represent what **information** is **encoded** at each encoder time step.



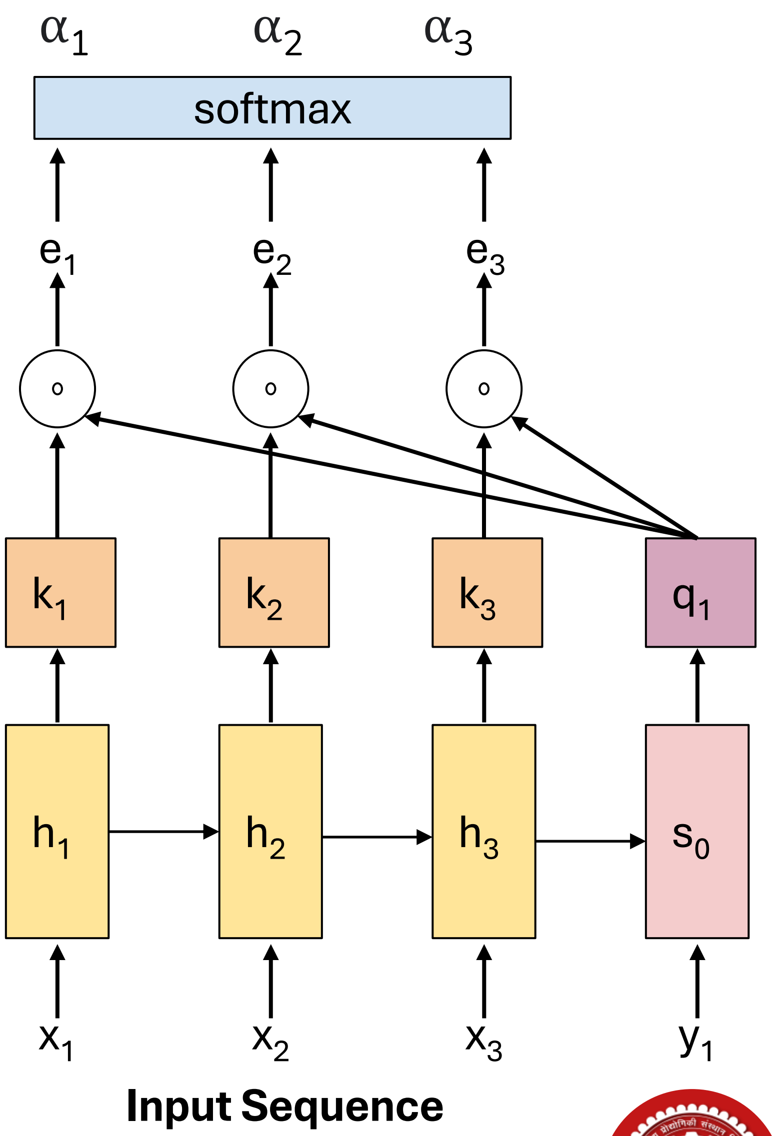
Attention



Query vectors represent what information we are **looking for** at each decoder time step.

Decoding

Attention

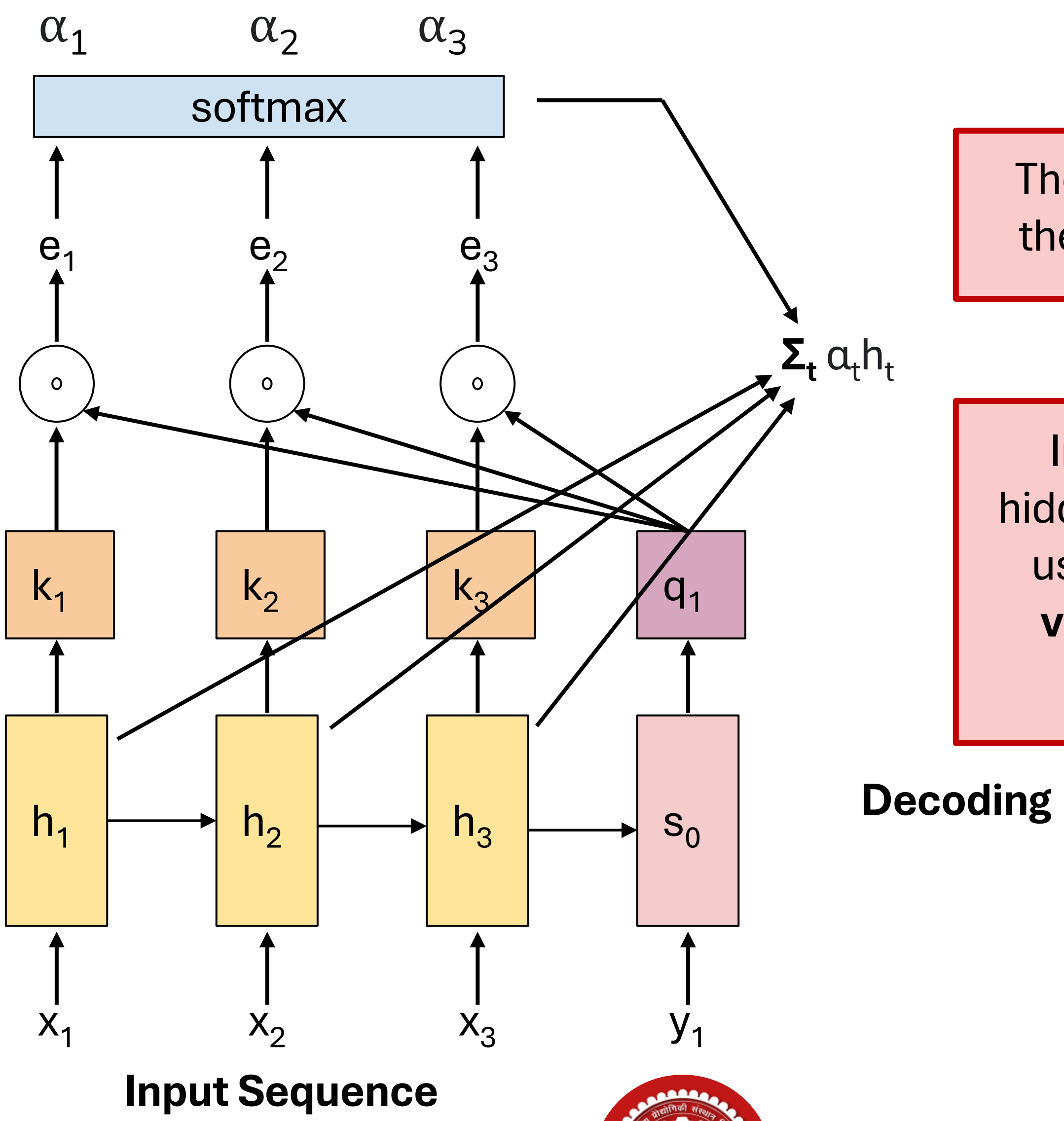


Softmax converts the similarity scores into a **probability distribution.**

Dot product between query vector and every key vector gives **similarity score.**

Decoding

Attention



The output of attention mechanism is the **weighted sum** of hidden vectors.

Instead of simply summing up the hidden vectors, we can transform them using a learned function to generate **value vectors** and then compute a weighted sum.

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: $a(\mathbf{q}, \mathbf{k}) = \dfrac{\mathbf{q}^T \mathbf{k}}{\sqrt{|\mathbf{k}|}}$ 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