Sequence-to-Sequence Modelling

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

Introduction to Large Language Models

Sequence-to-Sequence Modeling

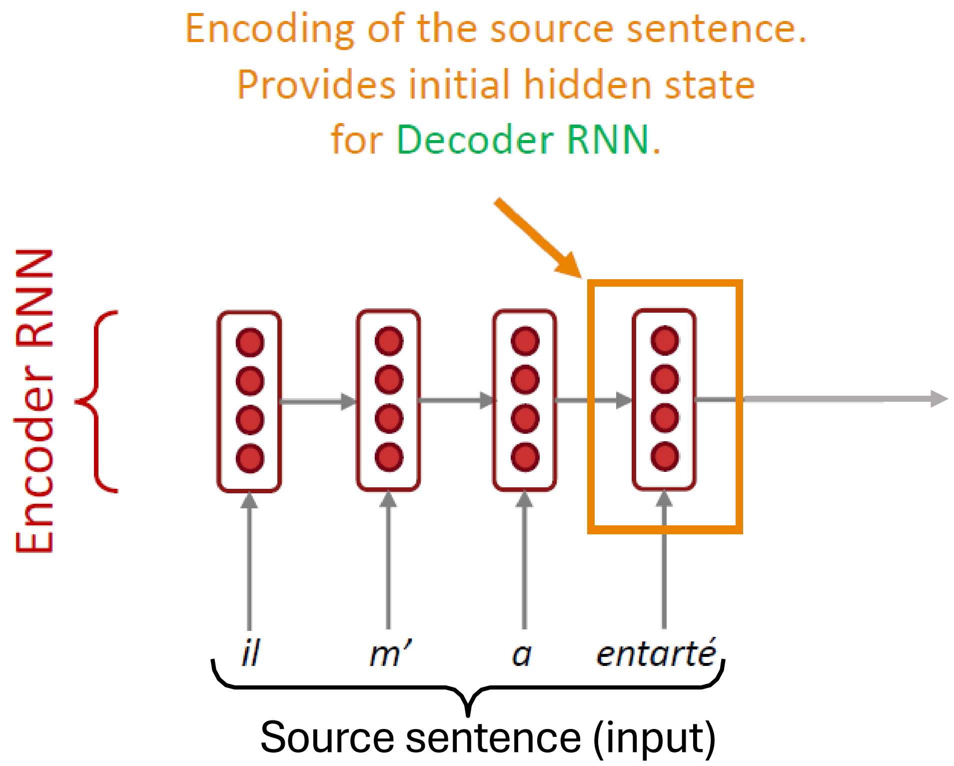
Neural Machine Translation?

- 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)

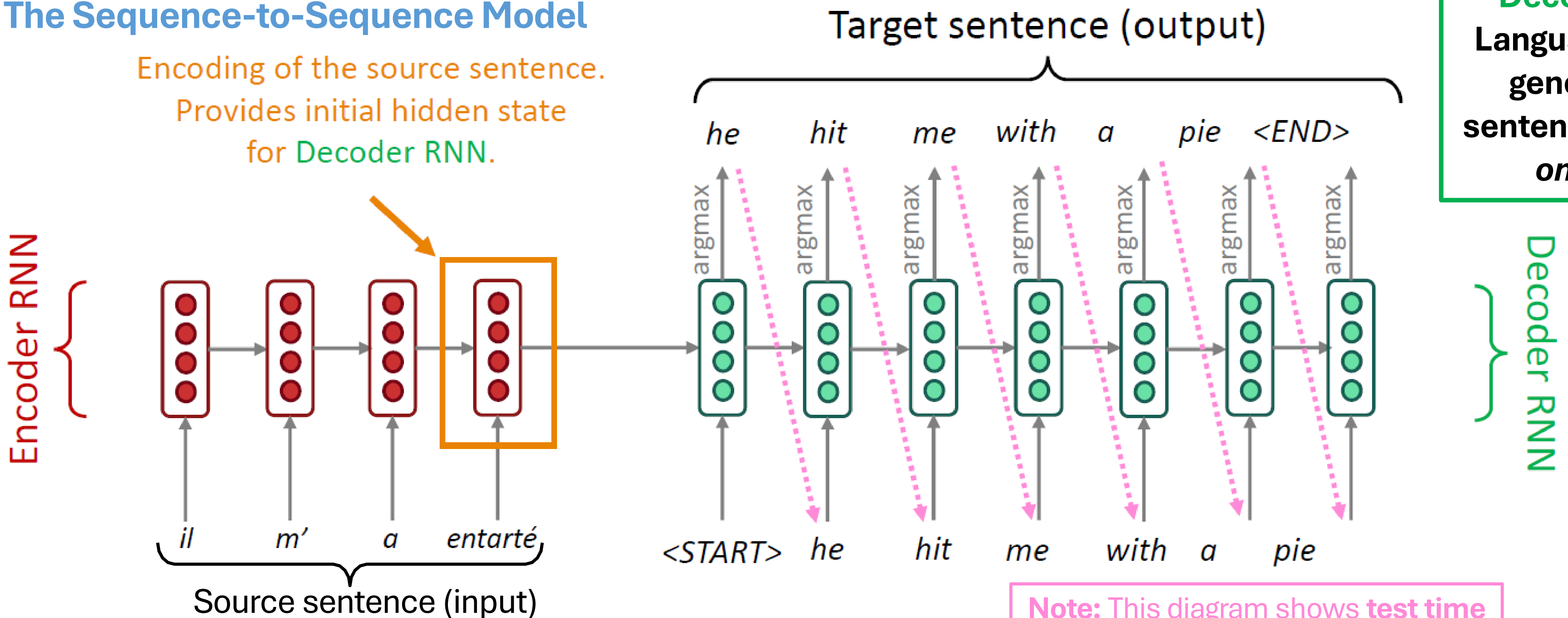
The Sequence-to-Sequence Model



Encoder RNN produces an encoding of the source sentence.

Neural Machine Translation (NMT)

The Sequence-to-Sequence Model



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

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)

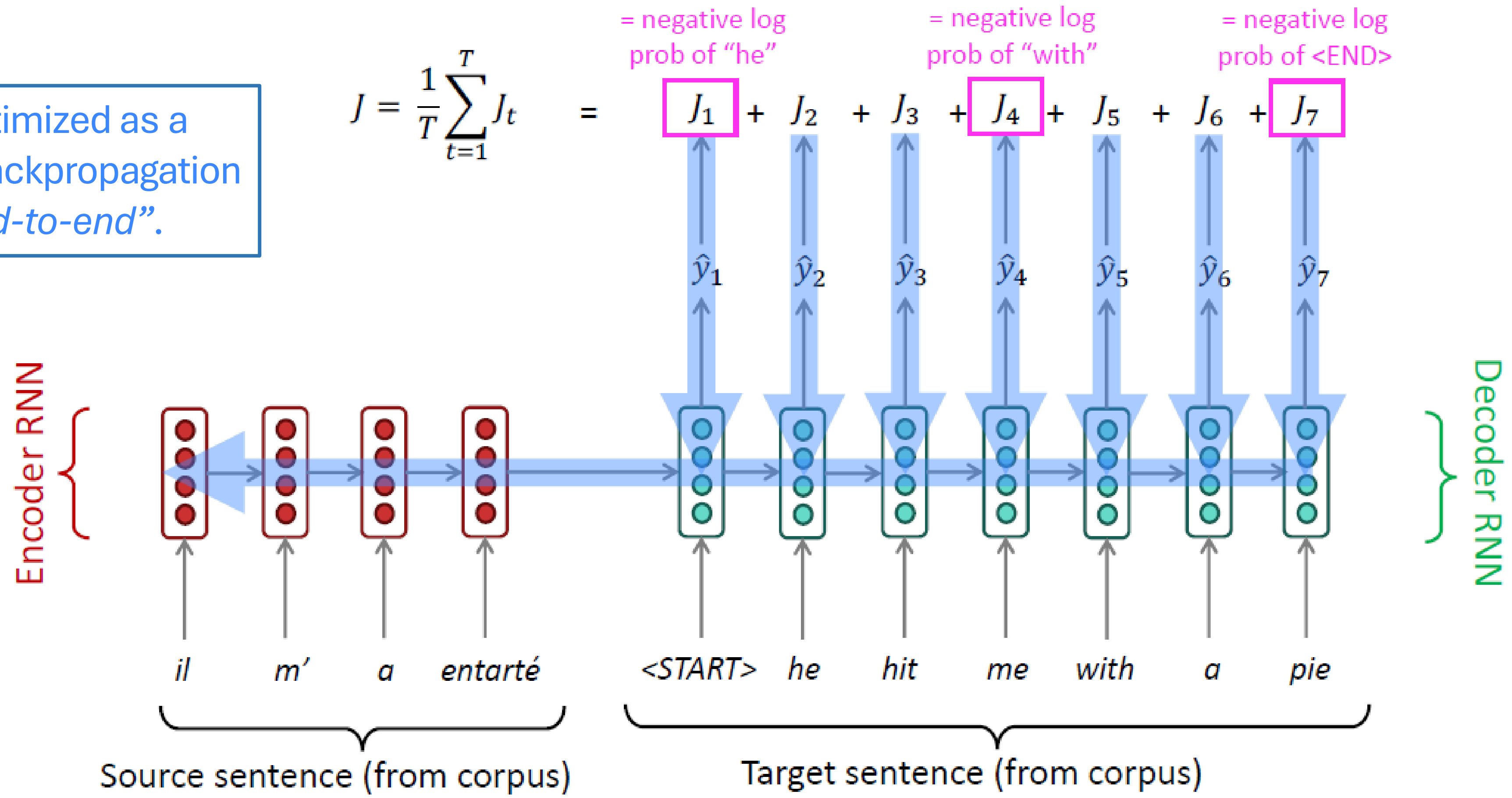
- The sequence-to-sequence model is an example of a **Conditional Language Model**
  - **Language Model** because the decoder is predicting the next word of the target sentence *y*
  - **Conditional** because its predictions are also conditioned on the source sentence *x*
- NMT directly calculates $P(y|x)$

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

- How to train an NMT system?

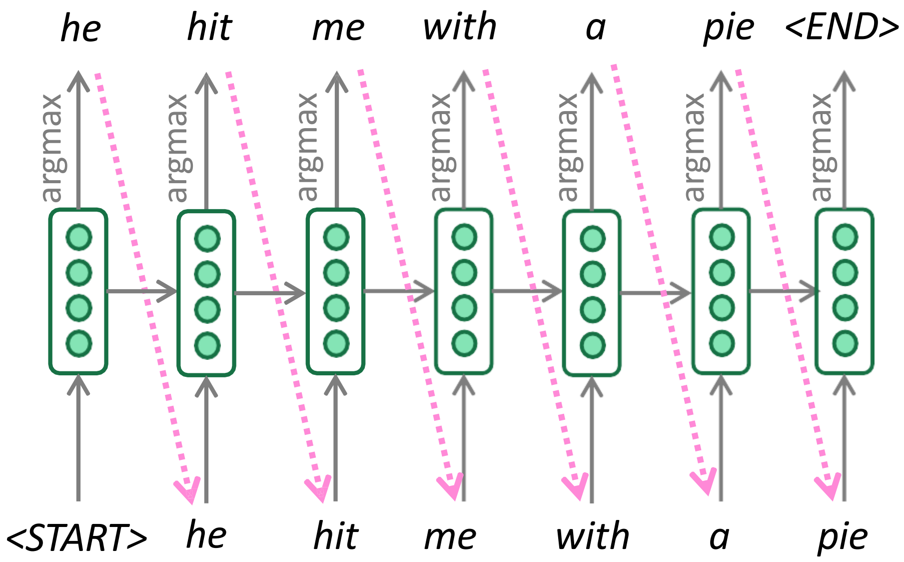
Training an NMT System

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

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

- 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!

Beam Search Decoding: Example

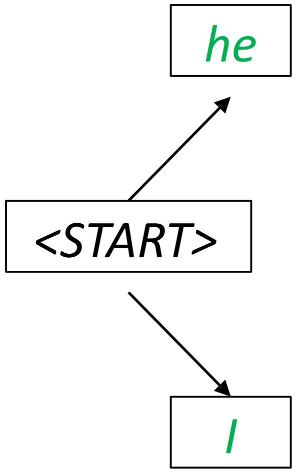
Beam size = k = 2.

<START>

Calculate prob distribution of next word

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

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

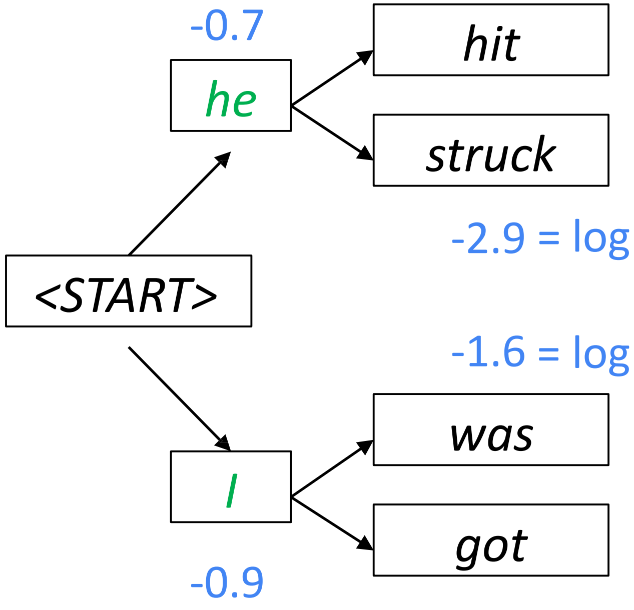


-0.9 = log P_{LM}(*I*|*<START>*)

Take top *k* words
and compute scores

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

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



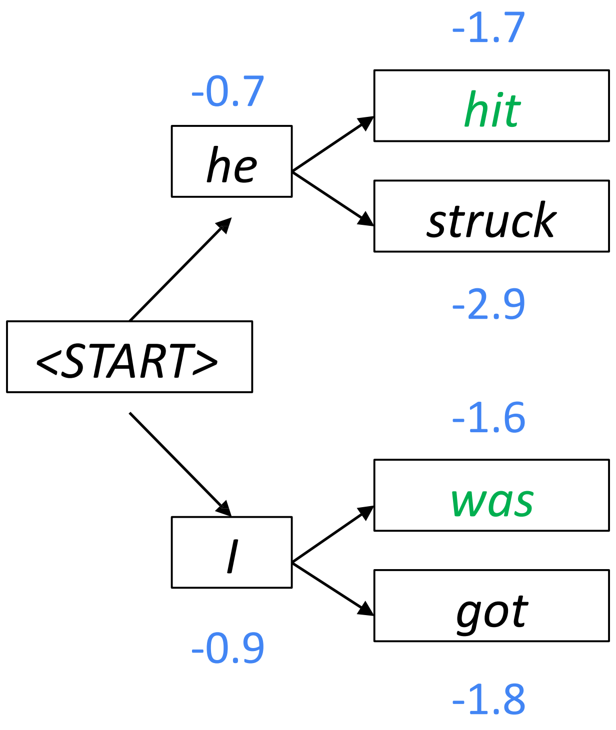
-2.9 = log P_{LM}(*struck*|*<START> he*) + -0.7

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

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

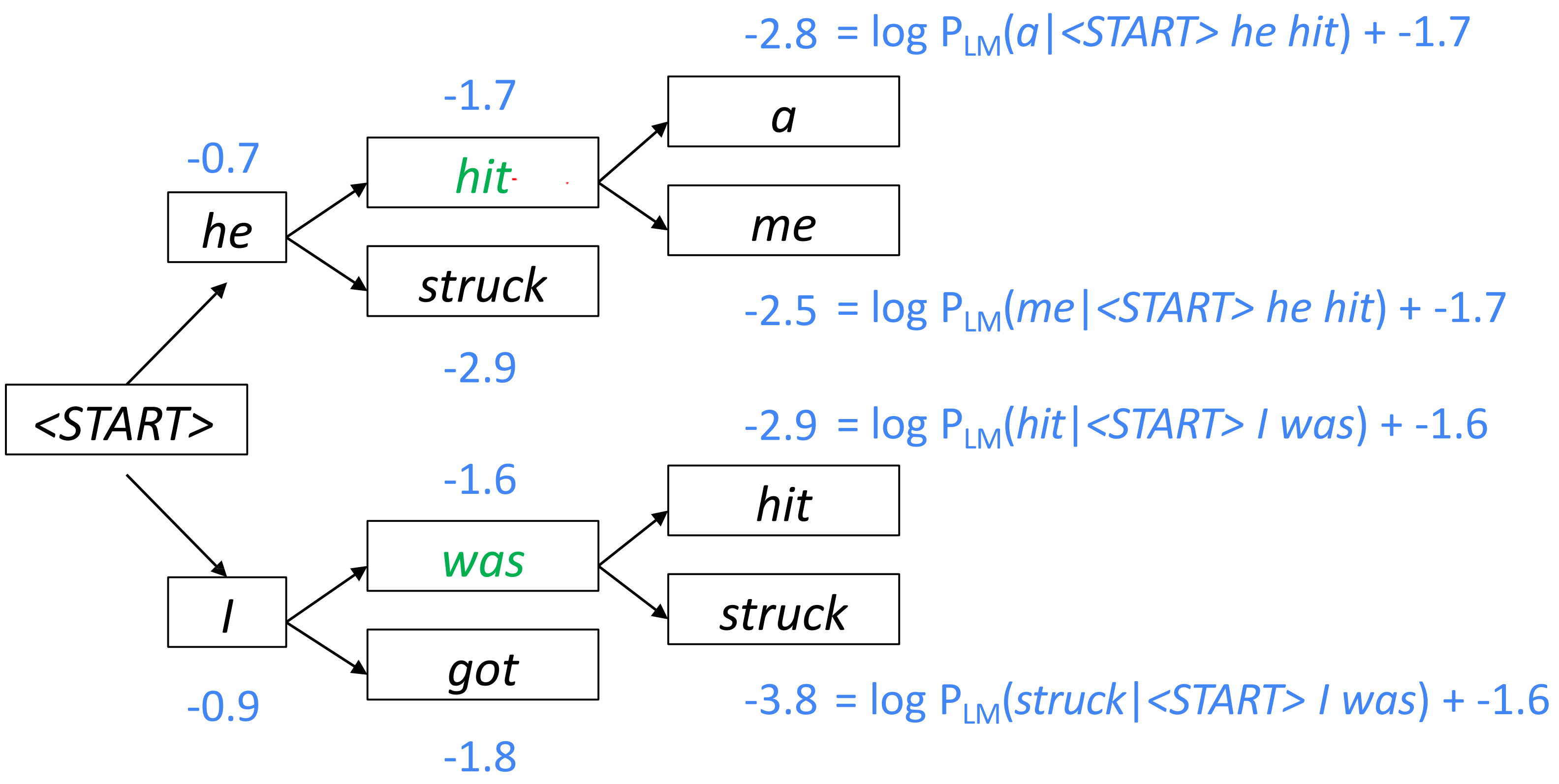
For each of the *k* hypotheses, find top *k* next words and calculate scores

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



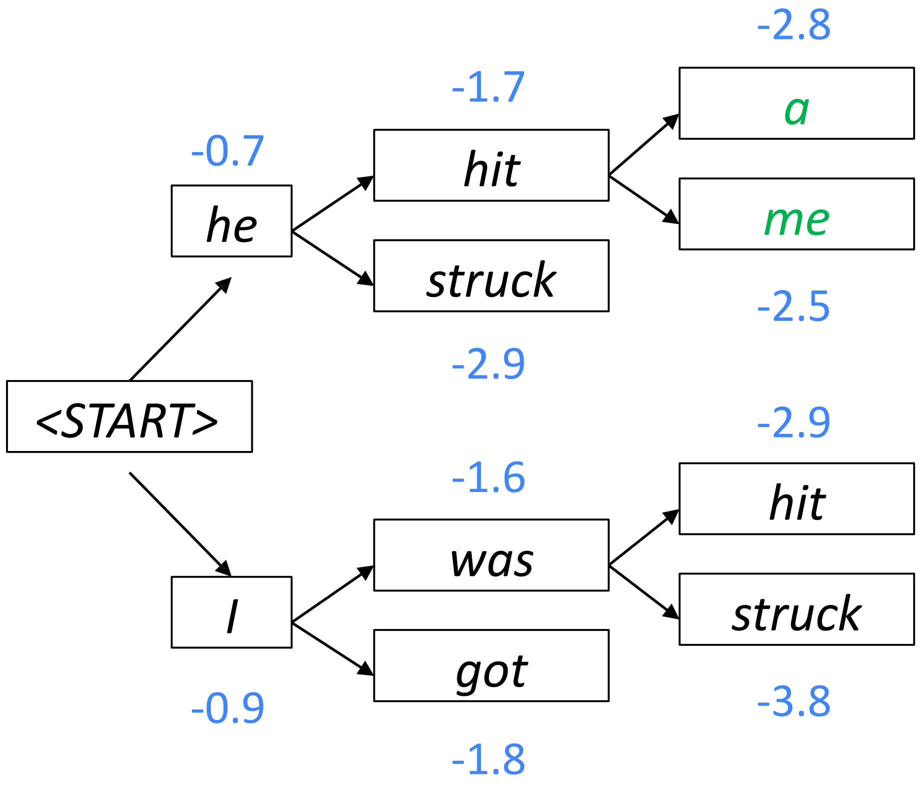
Of these k^2 hypotheses,
just keep k with highest scores

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



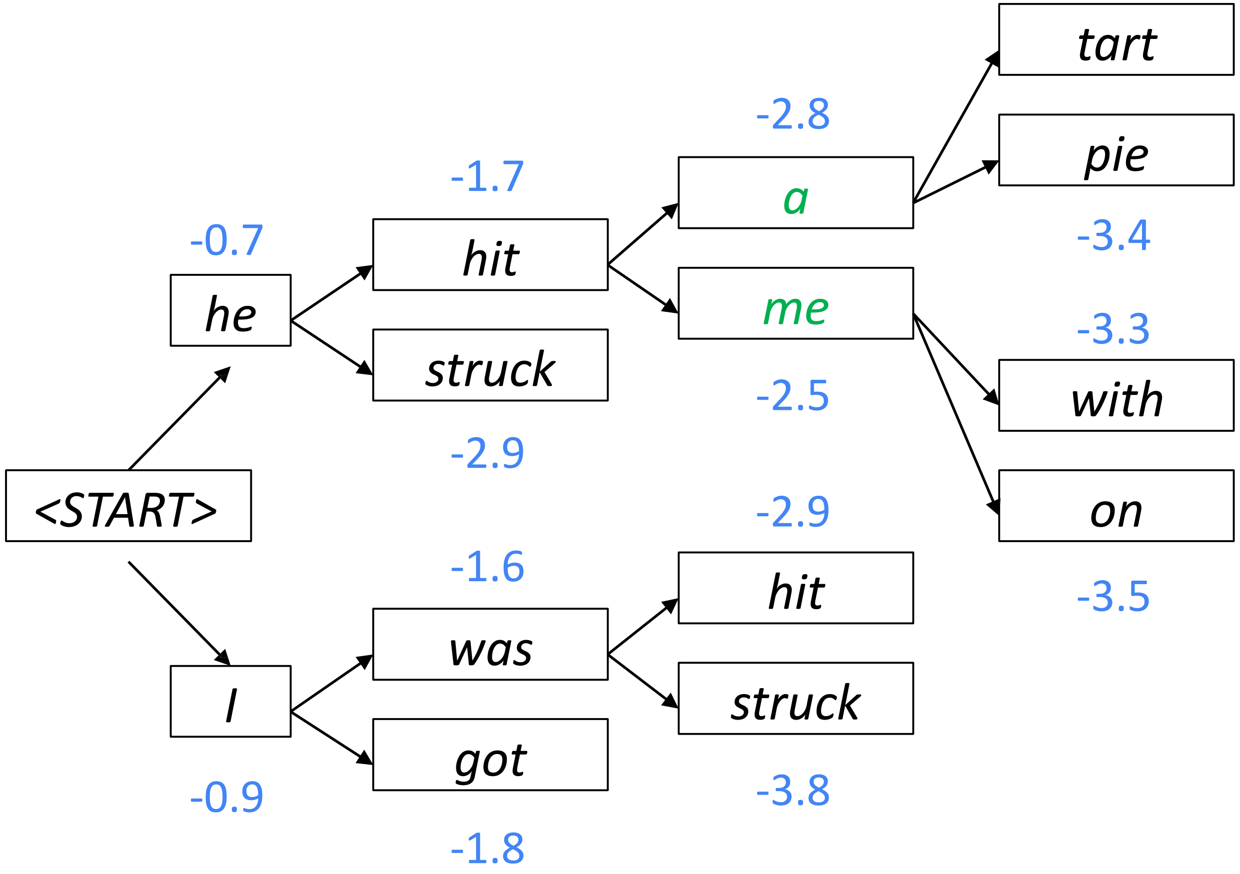
For each of the *k* hypotheses, find top *k* next words and calculate scores

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



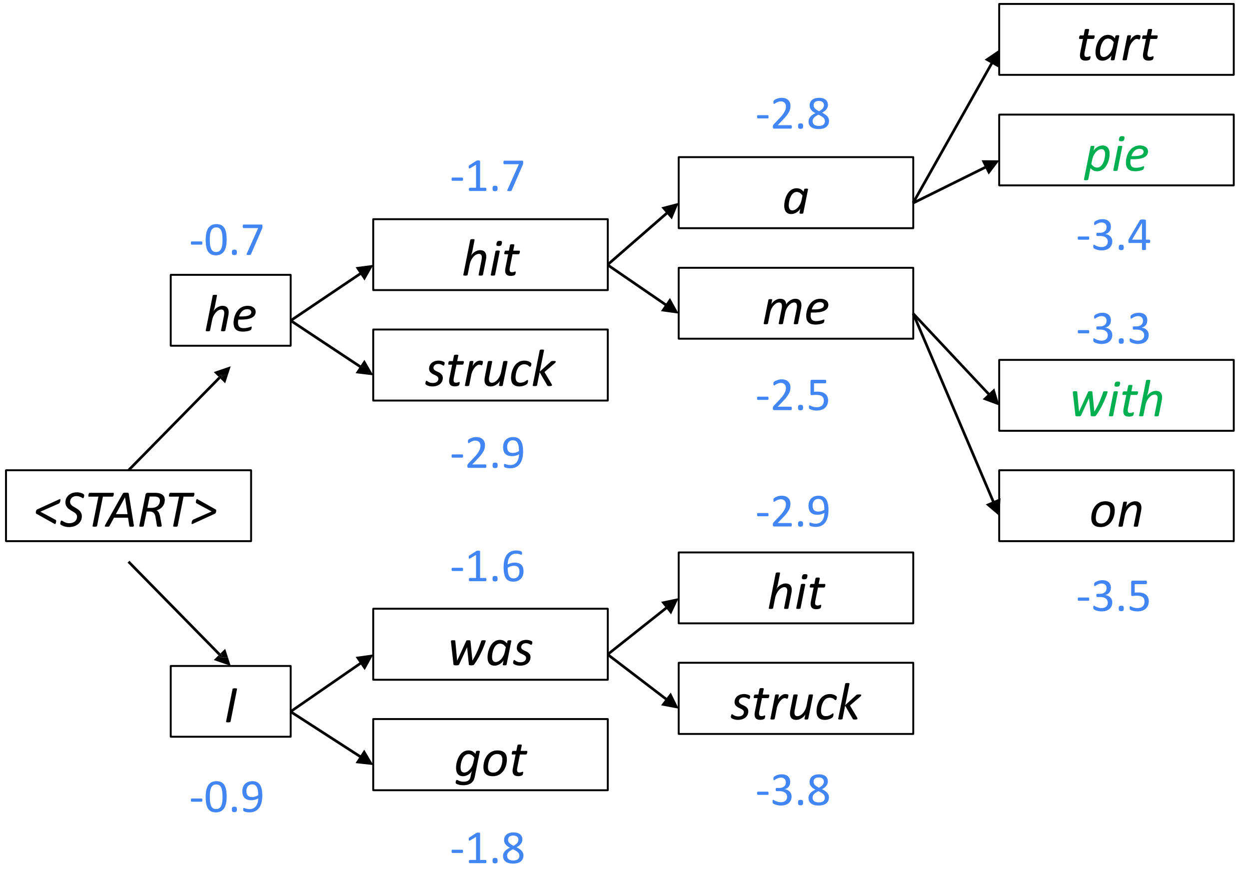
Of these k^2 hypotheses,
just keep k with highest scores

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



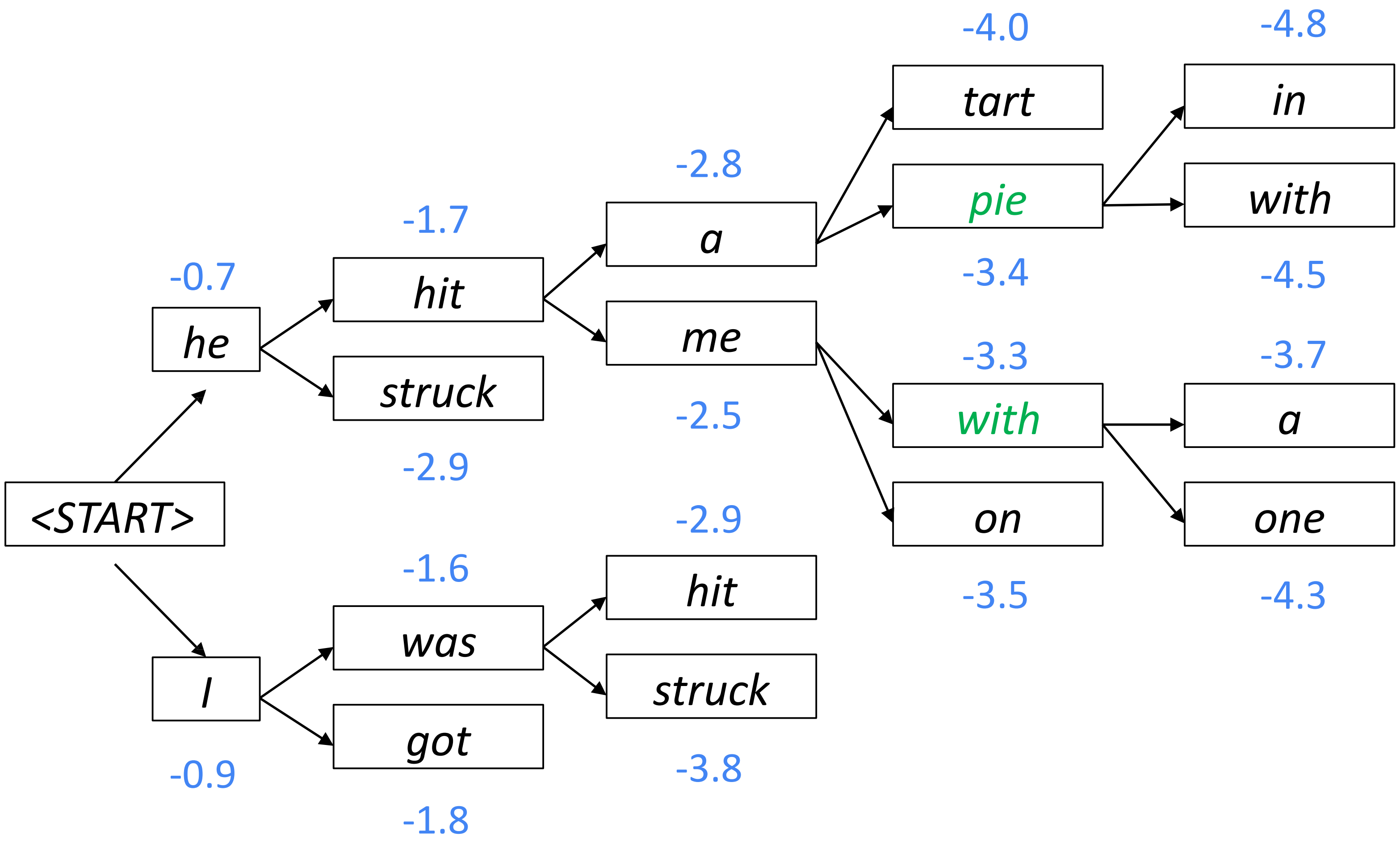
For each of the *k* hypotheses, find top *k* next words and calculate scores

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



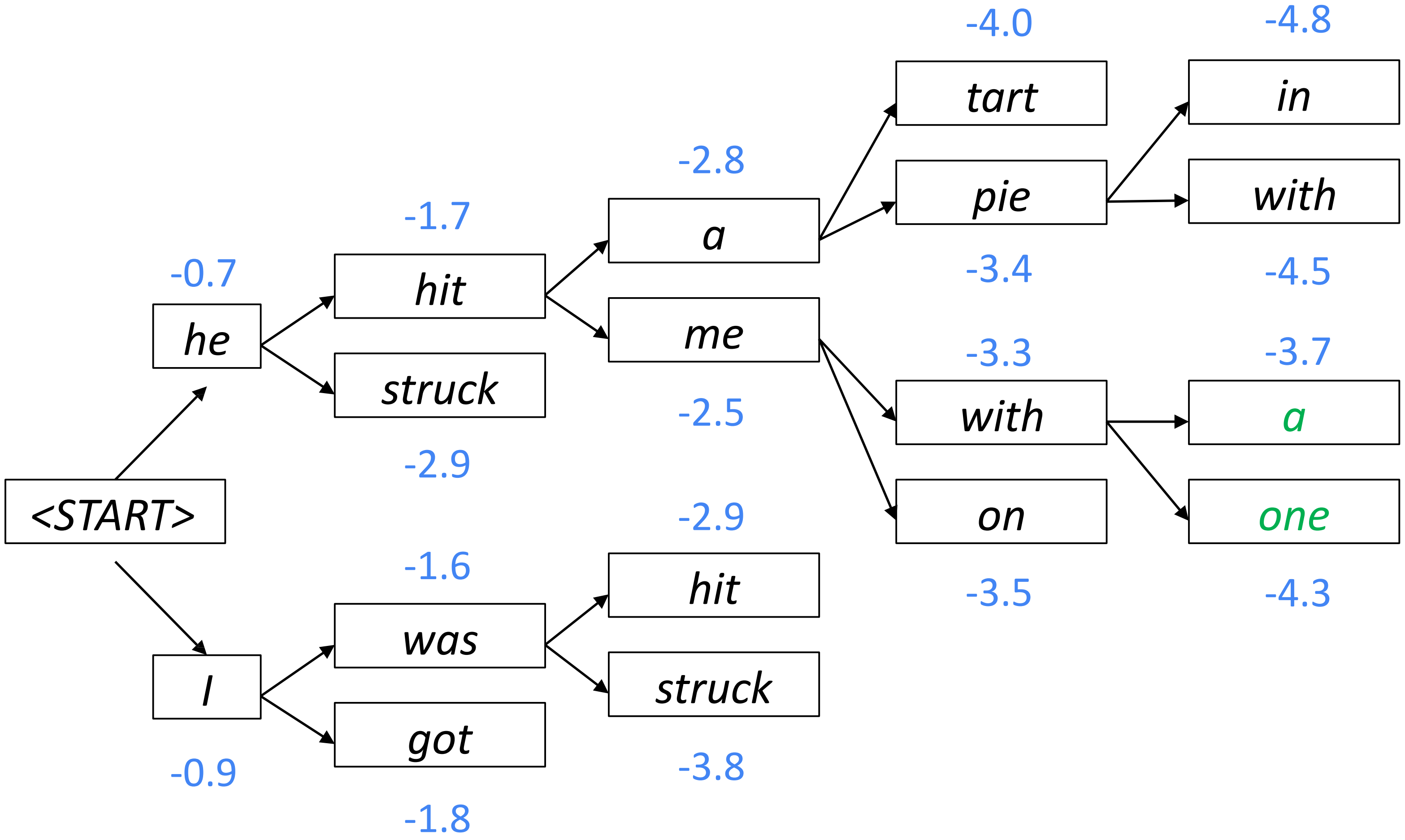
Of these k^2 hypotheses,
just keep k with highest scores

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



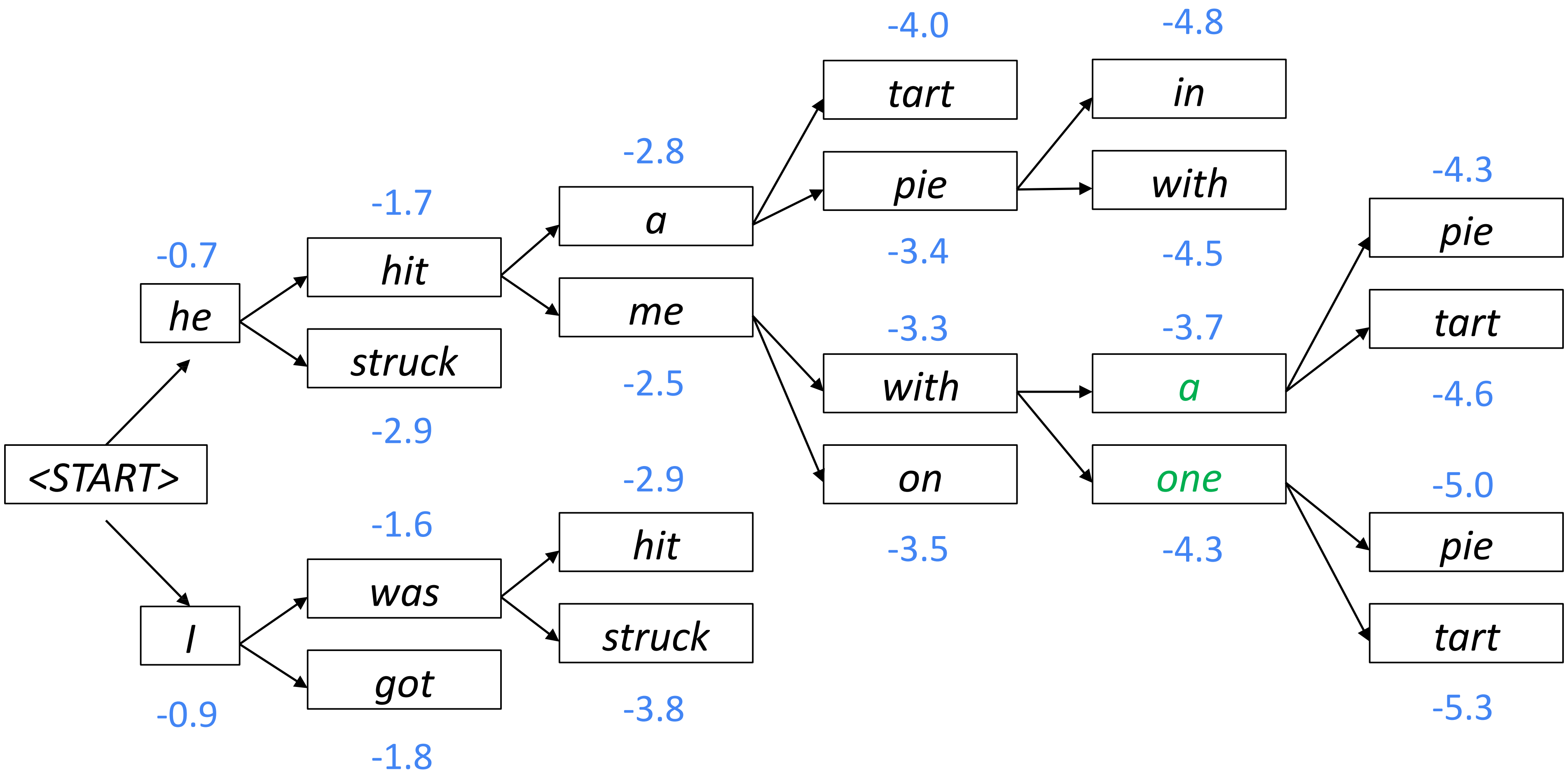
For each of the *k* hypotheses, find top *k* next words and calculate scores

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



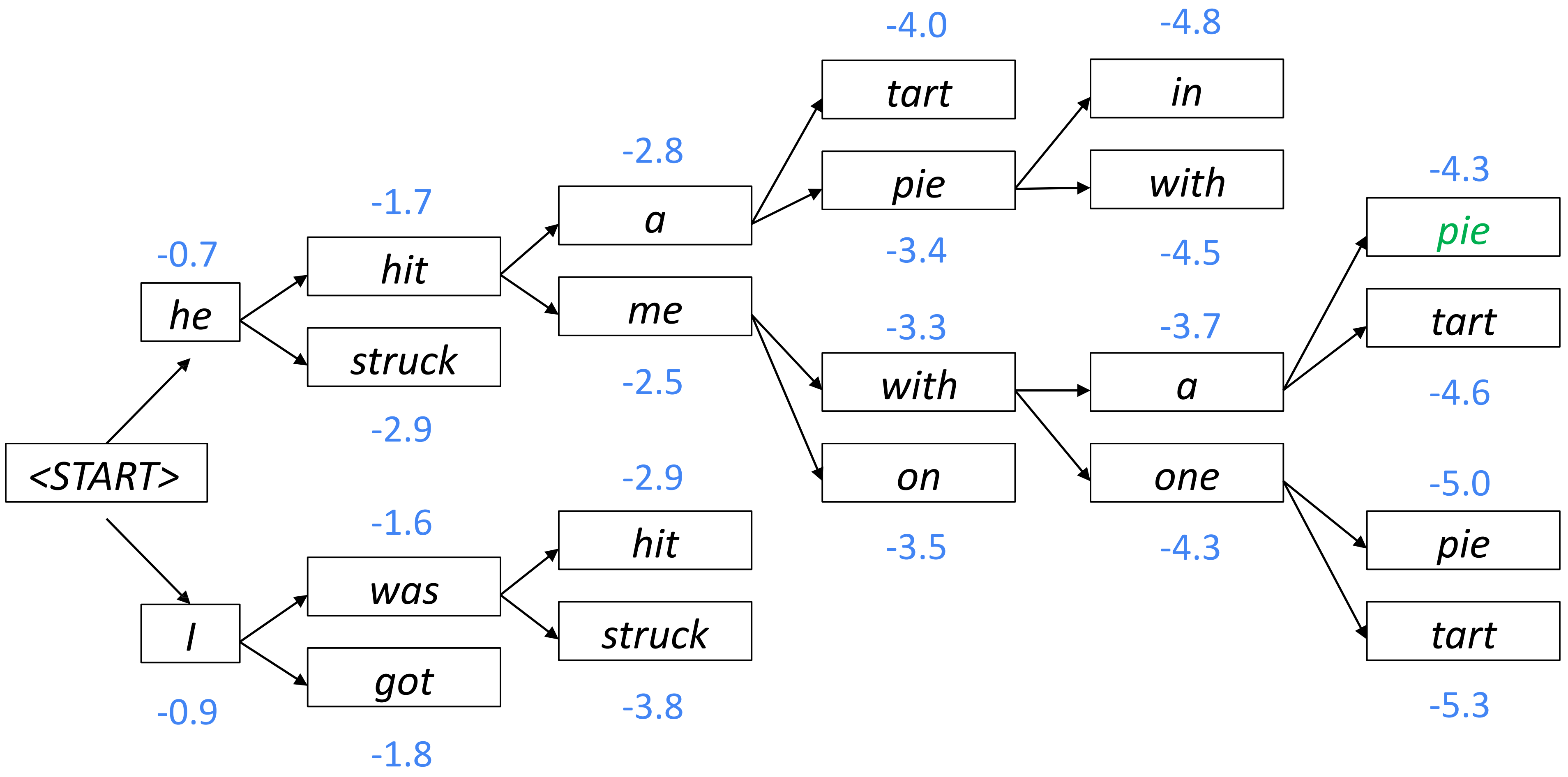
Of these k^2 hypotheses, just keep k with highest scores

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



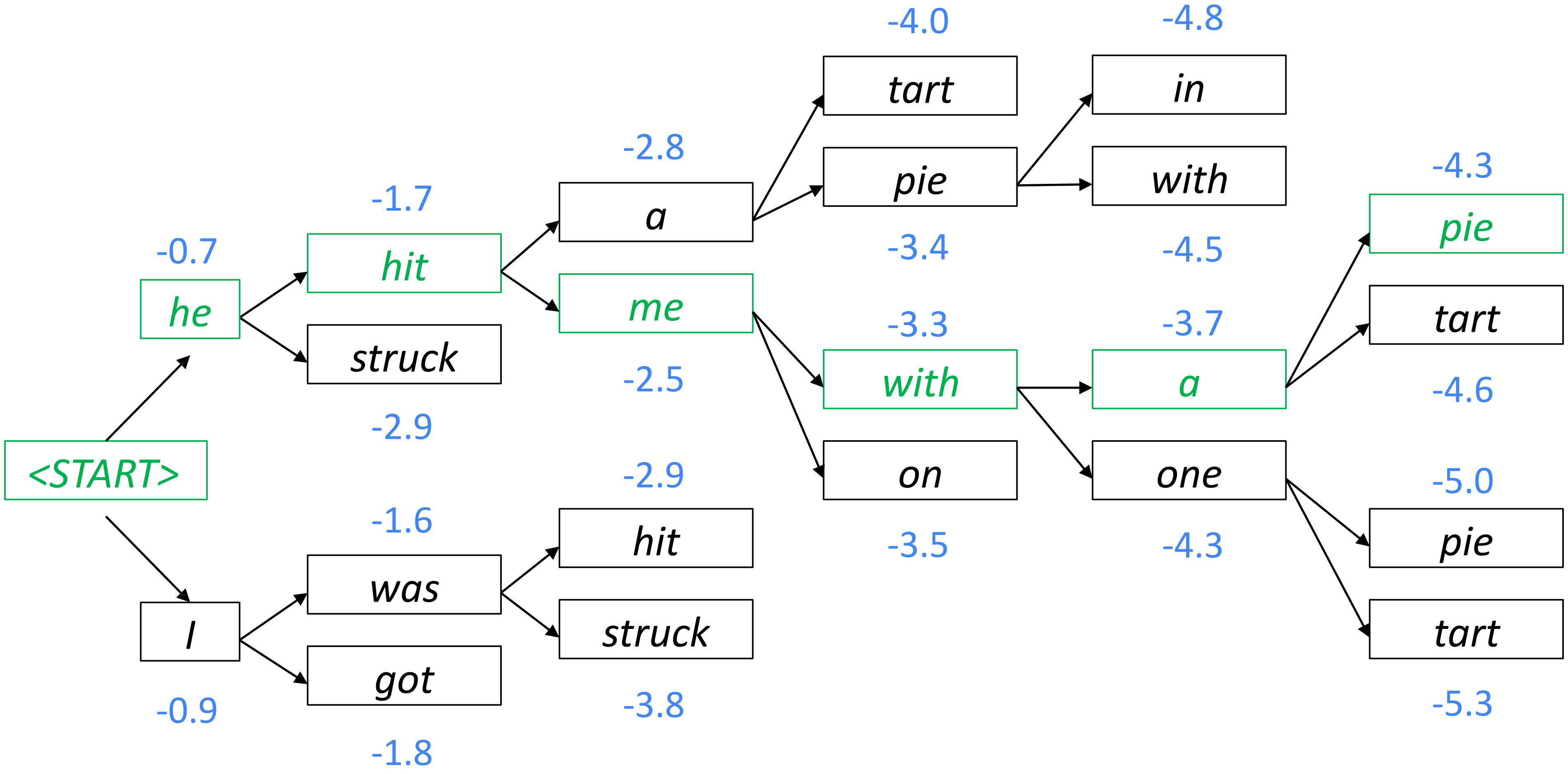
For each of the *k* hypotheses, find top *k* next words and calculate scores

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

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

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

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