Machine Translation

Sequence to Sequence models and Beam Search

cs224n-2020-lecture08-nmt.pdf,

Abigail See, Matthew Lamm

Speech and Language Processing. Daniel Jurafsky & James H. Martin.[Chapter 11]

Andrew Ng

The Encoder-Decoder Model

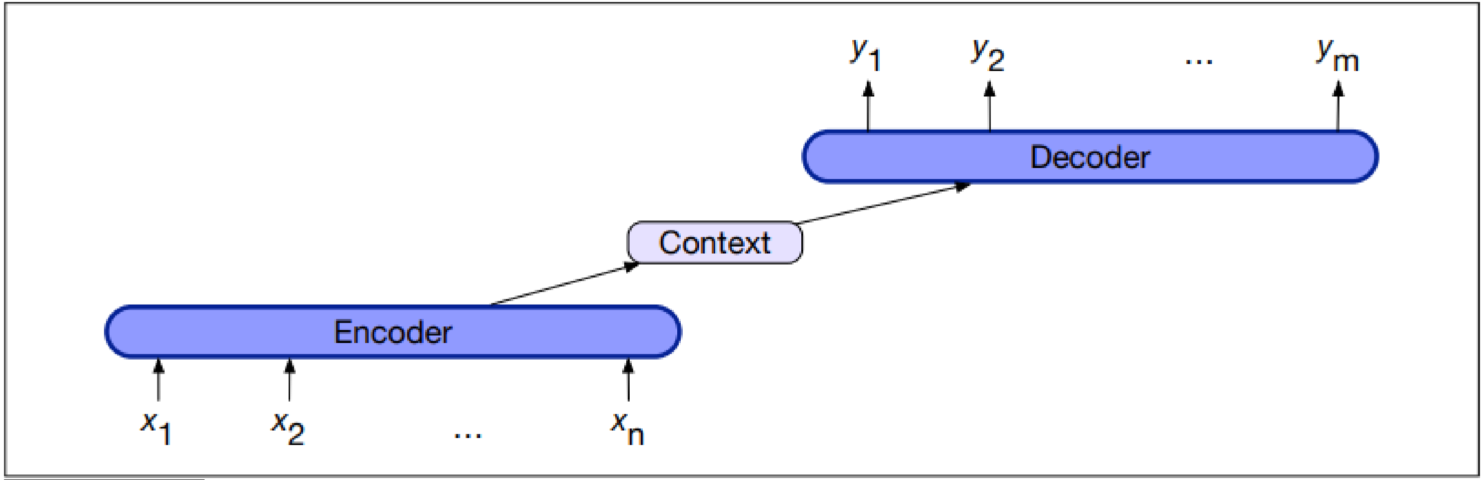


Figure 11.3 The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.

The Encoder-Decoder Model

Encoder-decoder networks consist of three components:

1. An **encoder** that accepts an input sequence, x_1^n, and generates a corresponding sequence of contextualized representations, h_1^n. LSTMs, GRUs, convolutional networks, and Transformers can all be employed as encoders.
2. A **context vector**, c, which is a function of h_1^n, and conveys the essence of the input to the decoder.
3. A **decoder**, which accepts c as input and generates an arbitrary length sequence of hidden states h_1^m, from which a corresponding sequence of output states y_1^m, can be obtained. Just as with encoders, decoders can be realized by any kind of sequence architecture.

Encoder-Decoder with RNNs

$$p(y) = p(y_1)p(y_2|y_1)p(y_3|y_1,y_2)...P(y_m|y_1,...,y_{m-1})$$

$$h_t = g(h_{t-1}, x_t)$$
$$y_t = f(h_t)$$

g is an activation function like tanh or ReLU
f is a softmax over the set of possible vocabulary items

x = source text
y = target text

$$p(y|x) = p(y_1|x)p(y_2|y_1,x)p(y_3|y_1,y_2,x)...P(y_m|y_1,...,y_{m-1},x)$$

Encoder-Decoder with RNNs

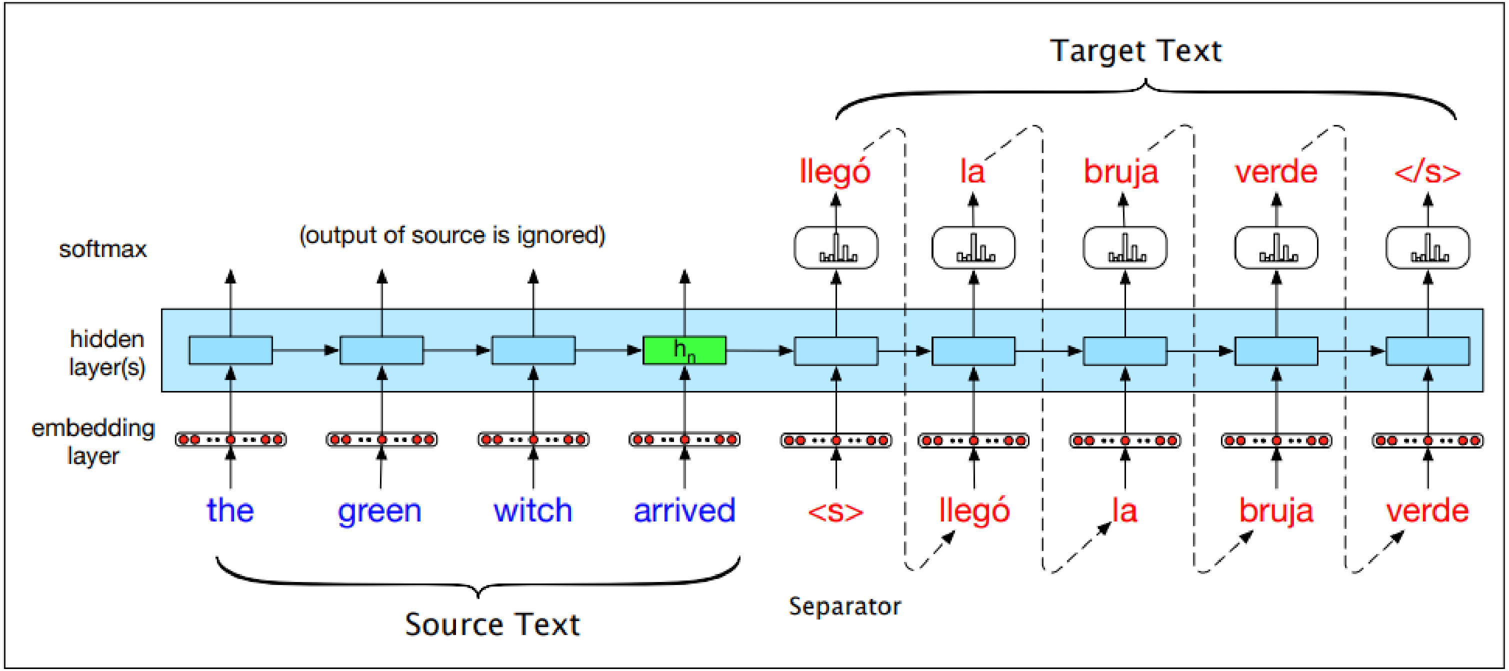


Figure 11.4 Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

Encoder-Decoder with RNNs

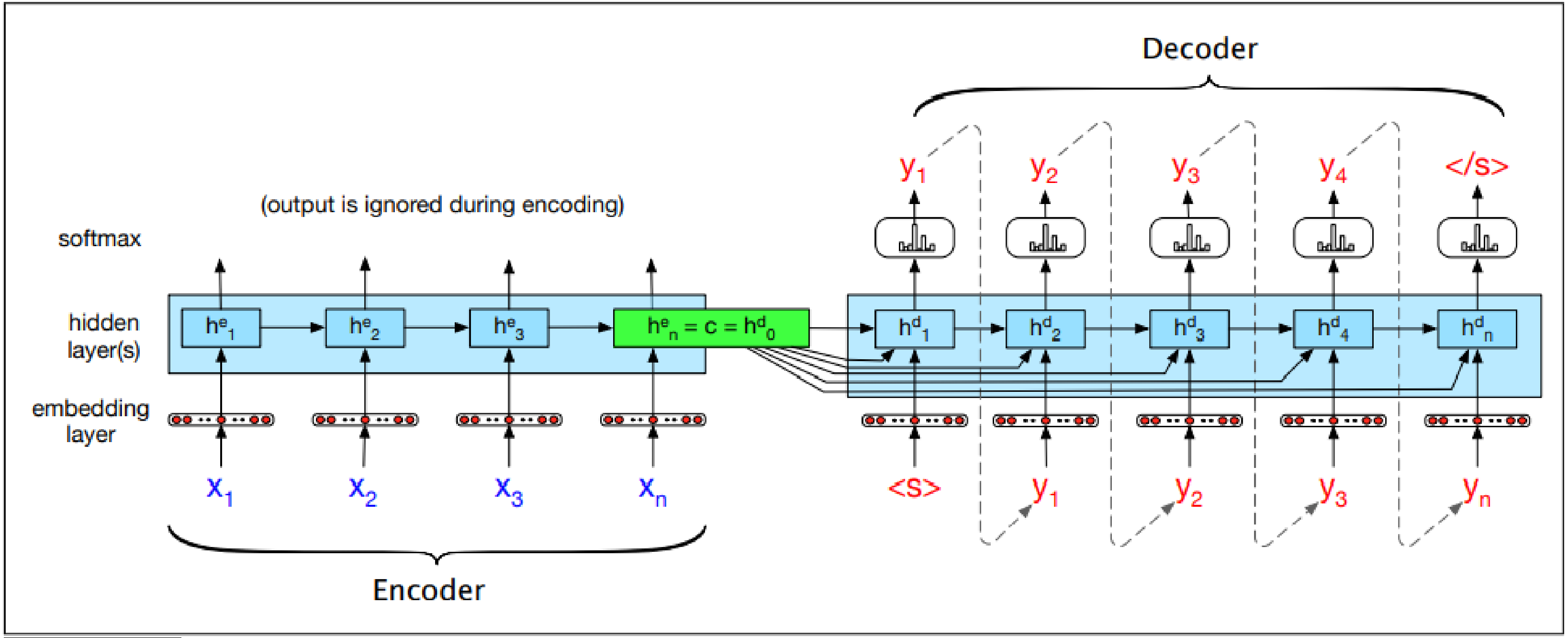


Figure 11.5 A more formal version of translating a sentence at inference time in the basic RNN-based encoder-decoder architecture. The final hidden state of the encoder RNN, h_n^e, serves as the context for the decoder in its role as h_0^d in the decoder RNN.

The entire purpose of the encoder is to generate a contextualized representation of the input.

Encoder-Decoder with RNNs

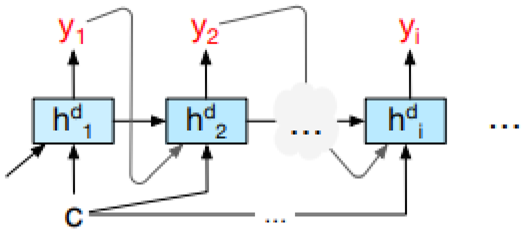


Figure 11.6 Allowing every hidden state of the decoder (not just the first decoder state) to be influenced by the context c produced by the encoder.

$$
\begin{aligned}
c &= h_n^e \\
h_0^d &= c \\
h_t^d &= g(\hat{y}_{t-1}, h_{t-1}^d, c) \\
z_t &= f(h_t^d) \\
y_t &= \text{softmax}(z_t)
\end{aligned}
$$

$$\boxed{\hat{y}_t = \text{argmax}_{w \in V} P(w|x, y_1 \ldots y_{t-1})}$$

$$\boxed{y_t = \text{softmax}(\hat{y}_{t-1}, z_t, c)}$$

Training the Encoder-Decoder Model

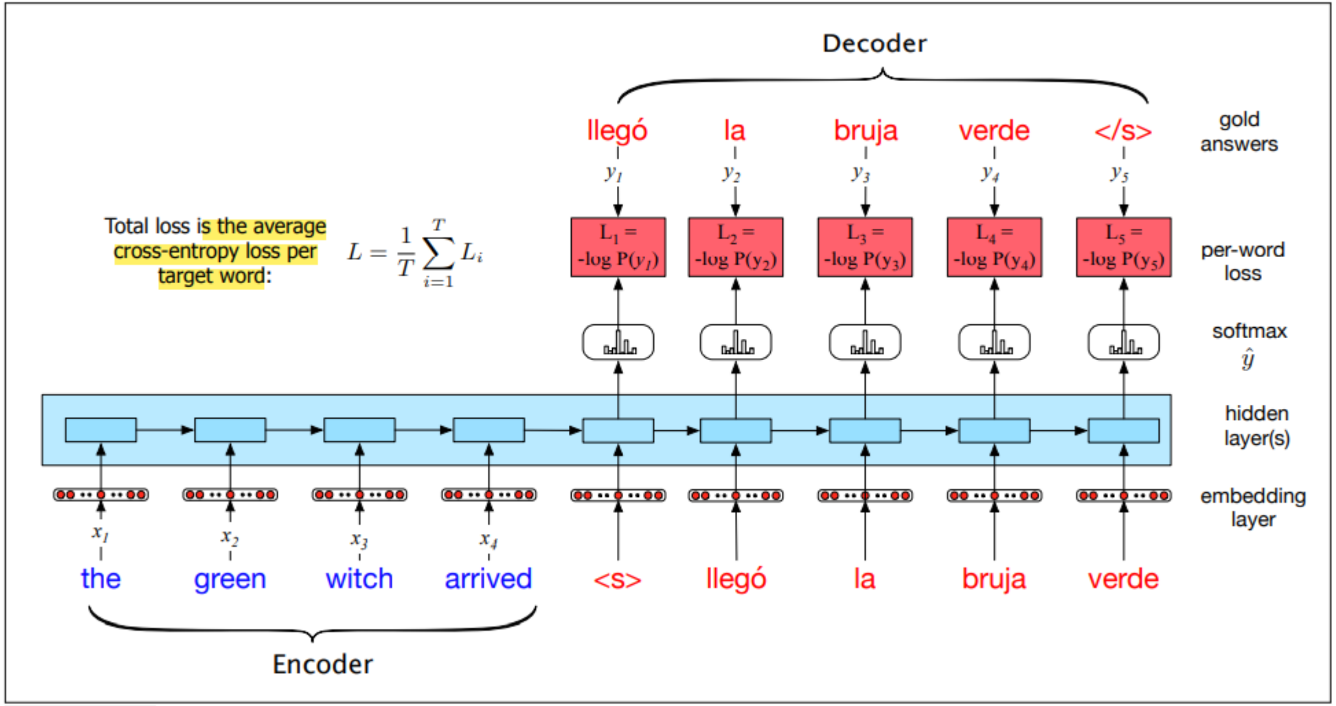


Figure 11.7 Training the basic RNN encoder-decoder approach to machine translation. Note that in the decoder we usually don't propagate the model's softmax outputs $\hat{y}_t$, but use **teacher forcing** to force each input to the correct gold value for training. We compute the softmax output distribution over $\hat{y}$ in the decoder in order to compute the loss at each token, which can then be averaged to compute a loss for the sentence.

Attention

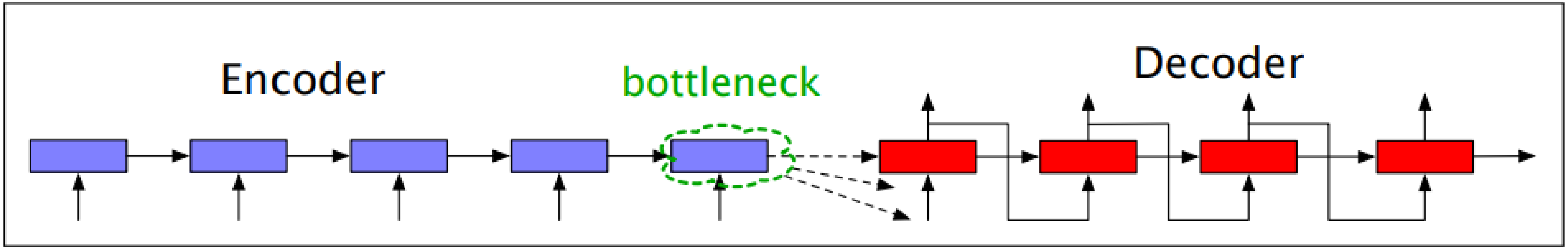


Figure 11.8 Requiring the context c to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck.

Information at the beginning of the sentence, especially for long sentences, may not be equally well represented in the context vector.

Attention

$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c) \rightarrow \boxed{h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)}$$

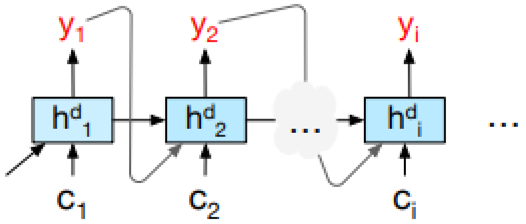


Figure 11.9 The attention mechanism allows each hidden state of the decoder to see a different, dynamic, context, which is a function of all the encoder hidden states.

Dot-product Attention

Measures how similar the decoder hidden state is to an encoder hidden state, by computing the dot product between them

$$score(h^d_{i-1}, h^e_j) = h^d_{i-1} \cdot h^e_j$$

$$\boxed{score(h^d_{i-1}, h^e_j) = h^d_{i-1} W_s h^e_j}$$

To be a more powerful function

$$\alpha_{ij} = \text{softmax}(score(h^d_{i-1}, h^e_j) \; \forall j \in e)$$

$$= \frac{\exp(score(h^d_{i-1}, h^e_j)}{\sum_k \exp(score(h^d_{i-1}, h^e_k))}$$

$$c_i = \sum_j \alpha_{ij} \, h^e_j$$

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

Attention

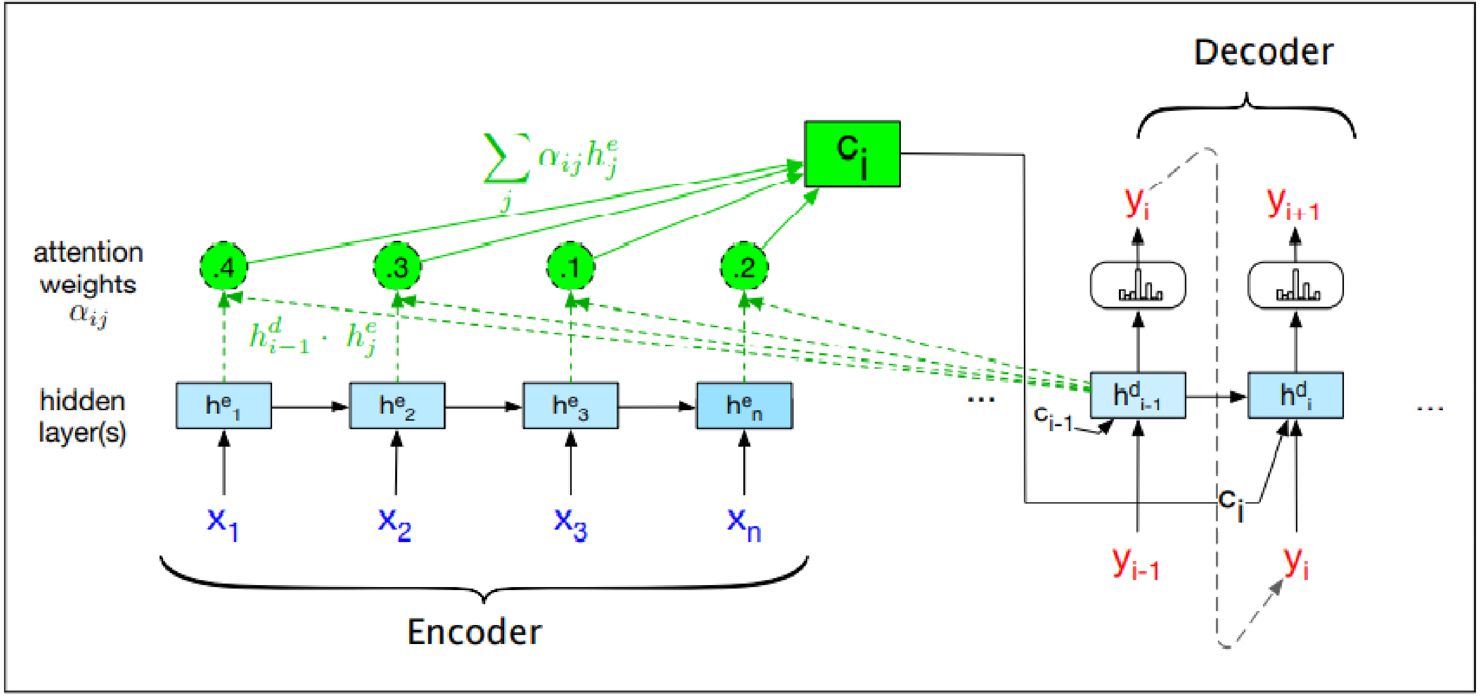


Figure 11.10 A sketch of the encoder-decoder network with attention, focusing on the computation of c_i. The context value c_i is one of the inputs to the computation of h^d_i. It is computed by taking the weighted sum of all the encoder hidden states, each weighted by their dot product with the prior decoder hidden state h^d_{i-1}.

Exhaustive search decoding

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

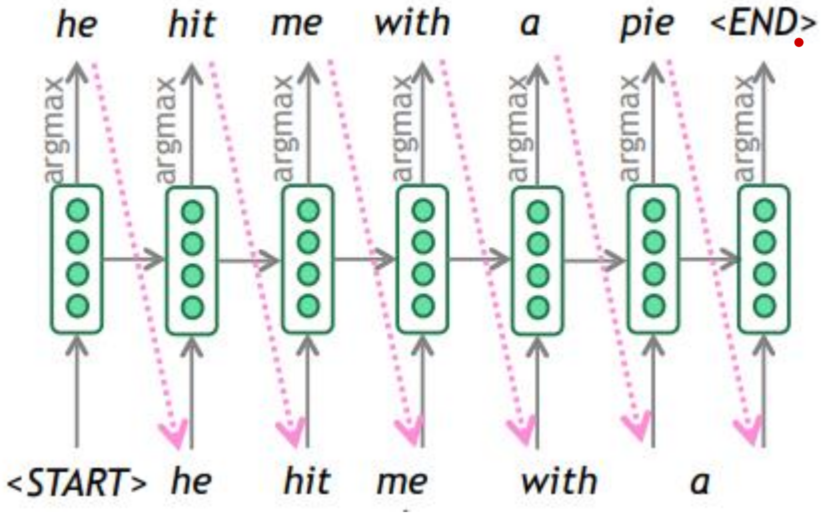
$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots,P(y_T|y_1,\ldots,y_{T-1},x)$$
$$= \prod_{t=1}^{T} P(y_t|y_1,\ldots,y_{t-1},x)$$

- 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

Choosing the single most probable token to generate at each step is called greedy decoding

$$\hat{y}_t = \text{argmax}_{w \in V} P(w|x, y_1 \ldots y_{t-1})$$



- 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…)

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

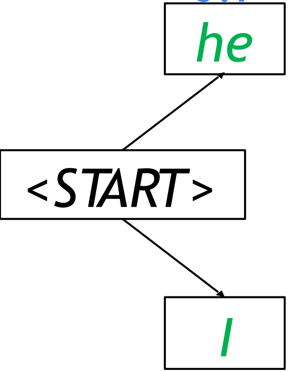
<START>

Calculate prob dist of next word

Beam search decoding: example

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 = $\log P_{LM}(he|\text{<}START\text{>})$



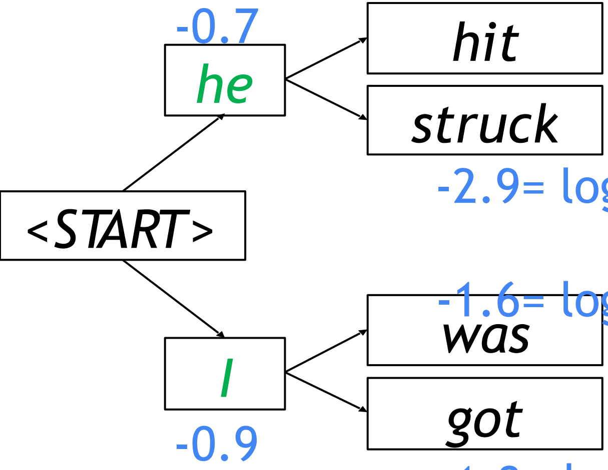
-0.9 = $\log P_{LM}(I|\text{<}START\text{>})$

Take top *k* words
and compute scores

Beam search decoding: example

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

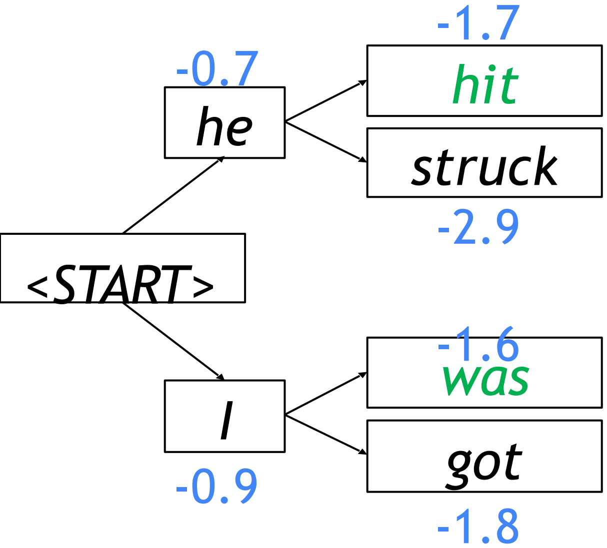
-1.6= log P_{LM}(*was*|<*START*> *I*) + -0.9

-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 search decoding: example

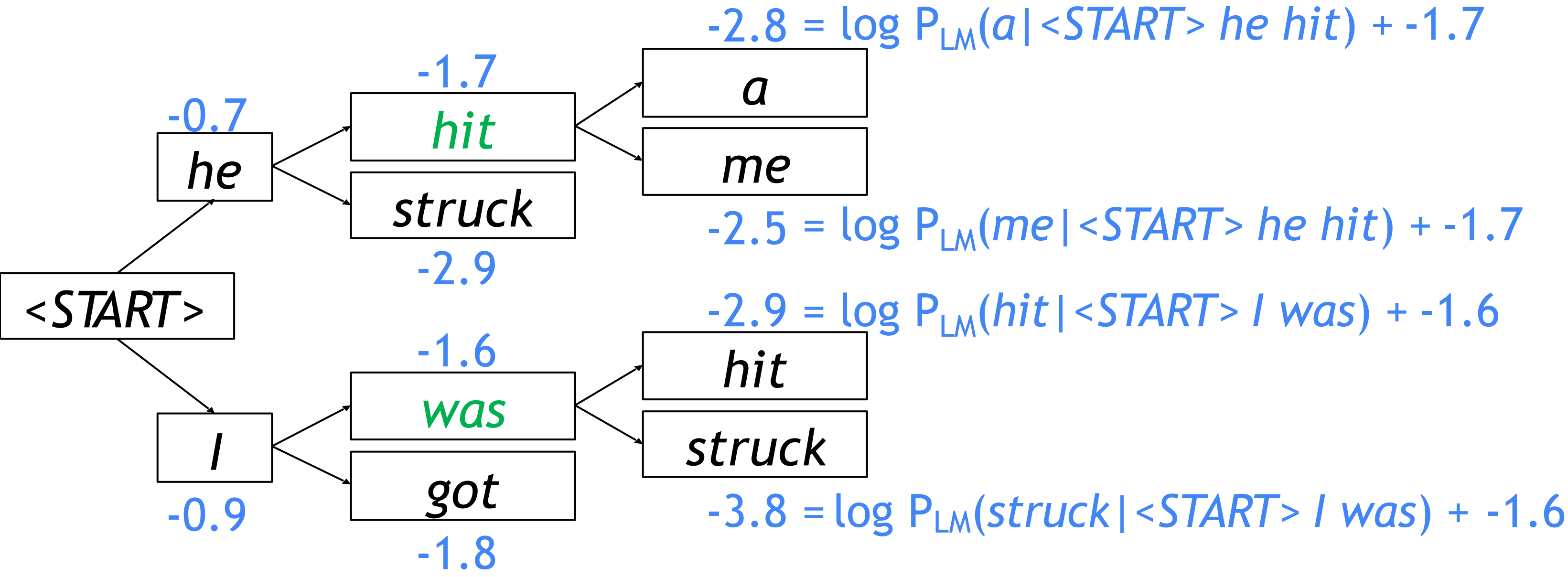
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

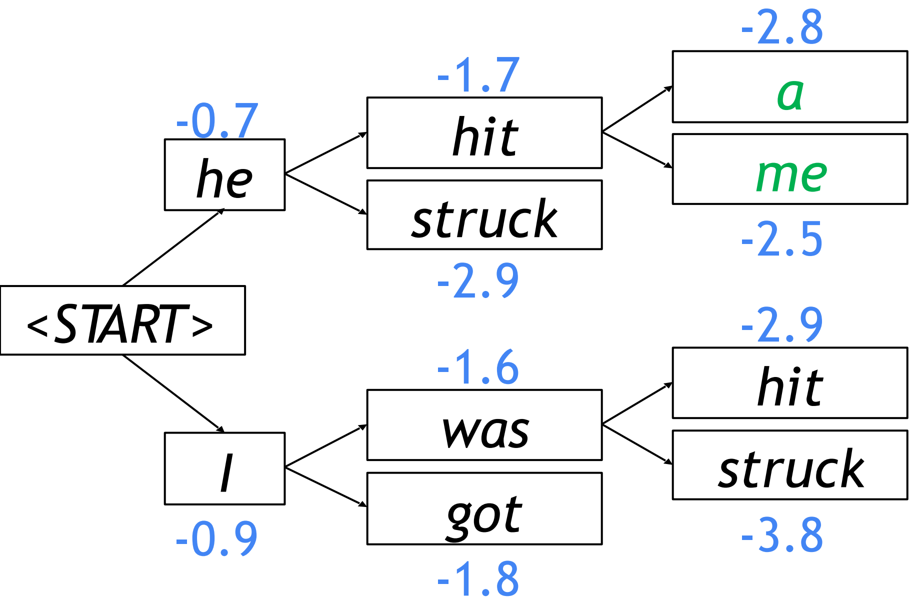
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

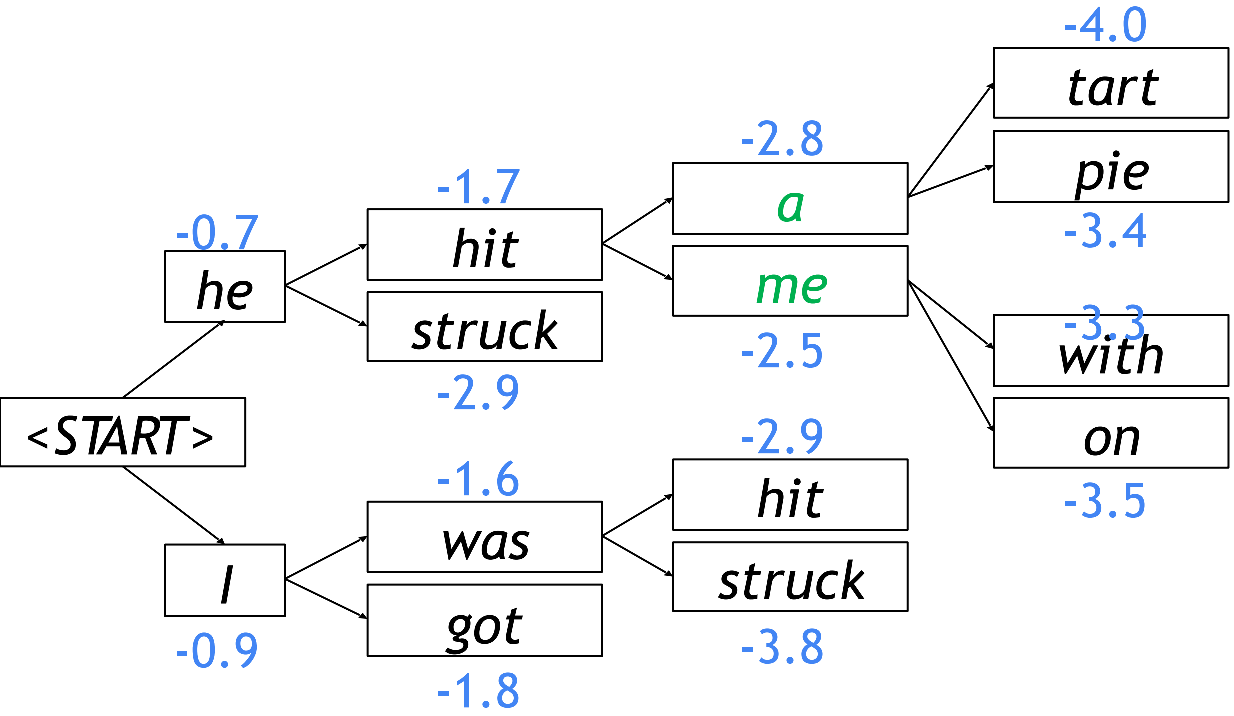
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

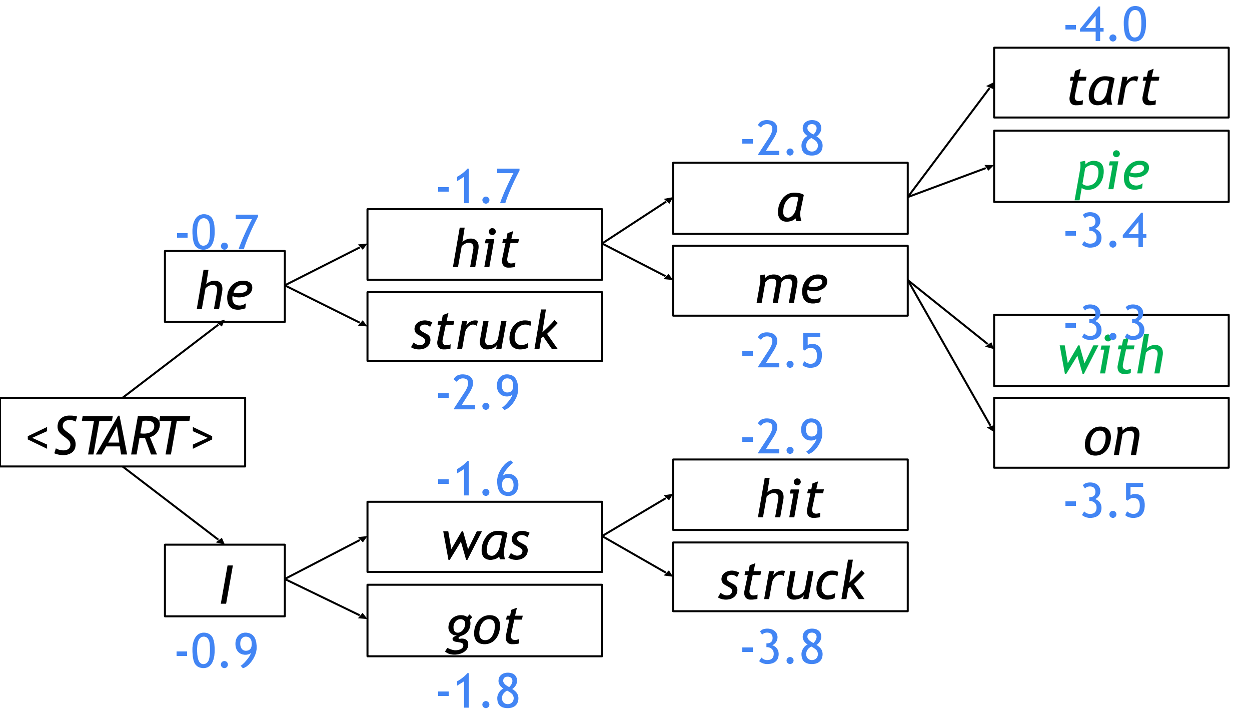
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

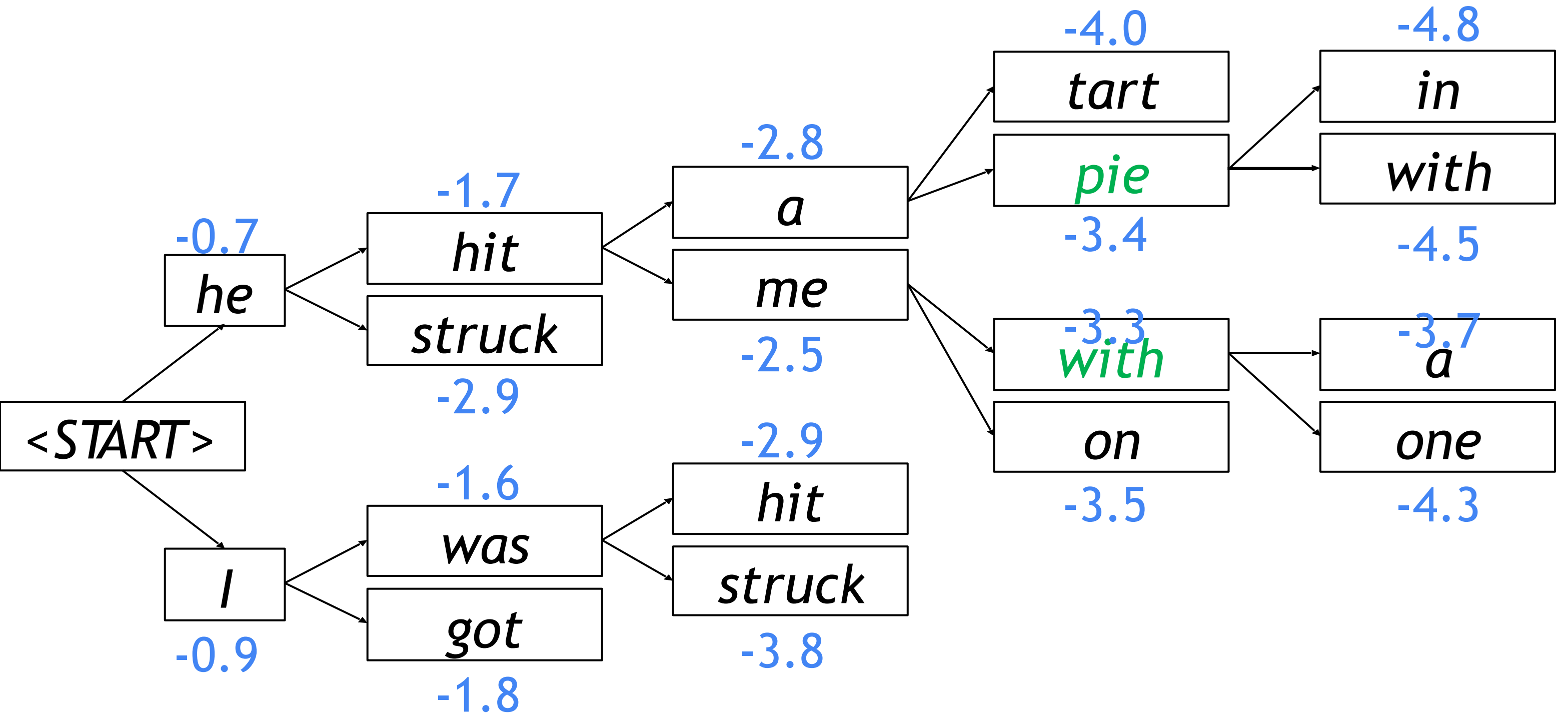
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

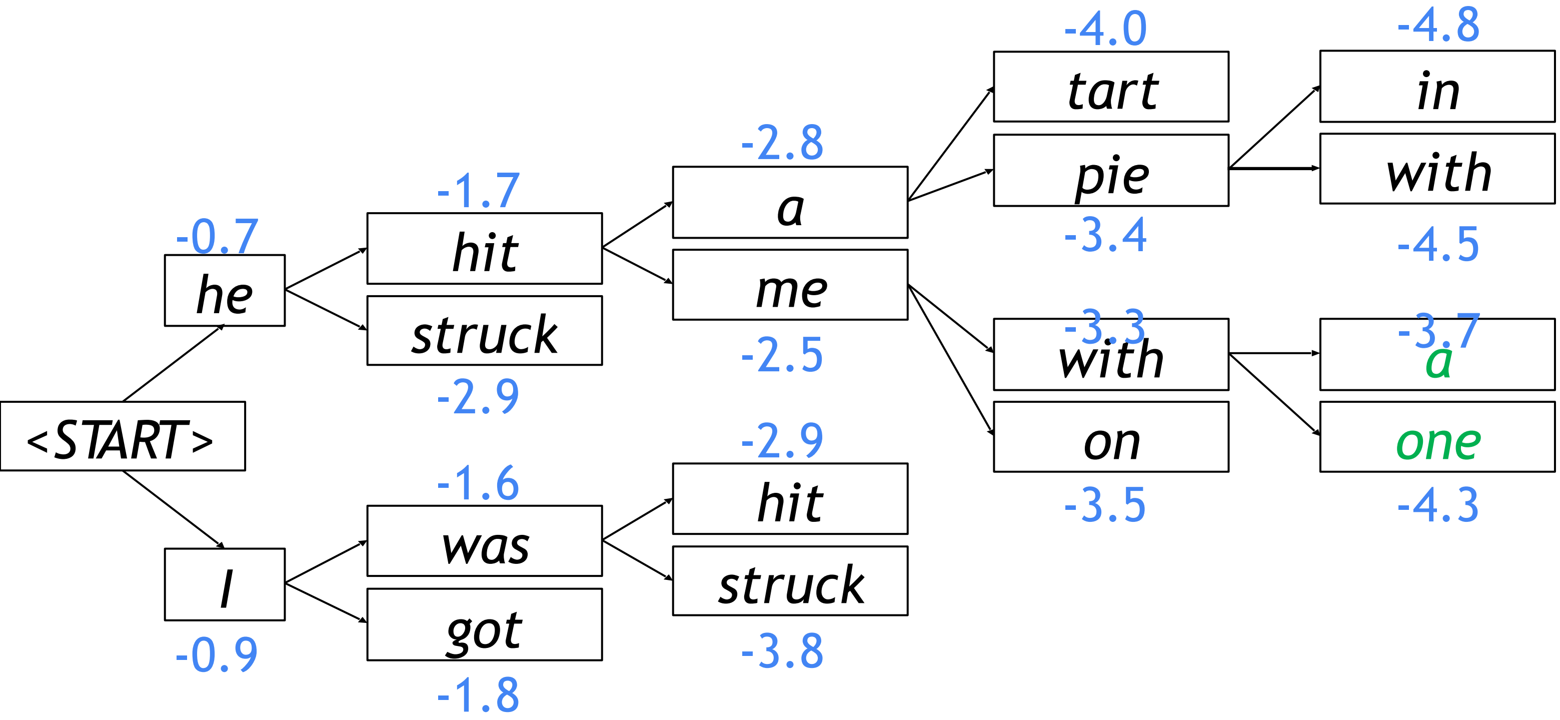
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

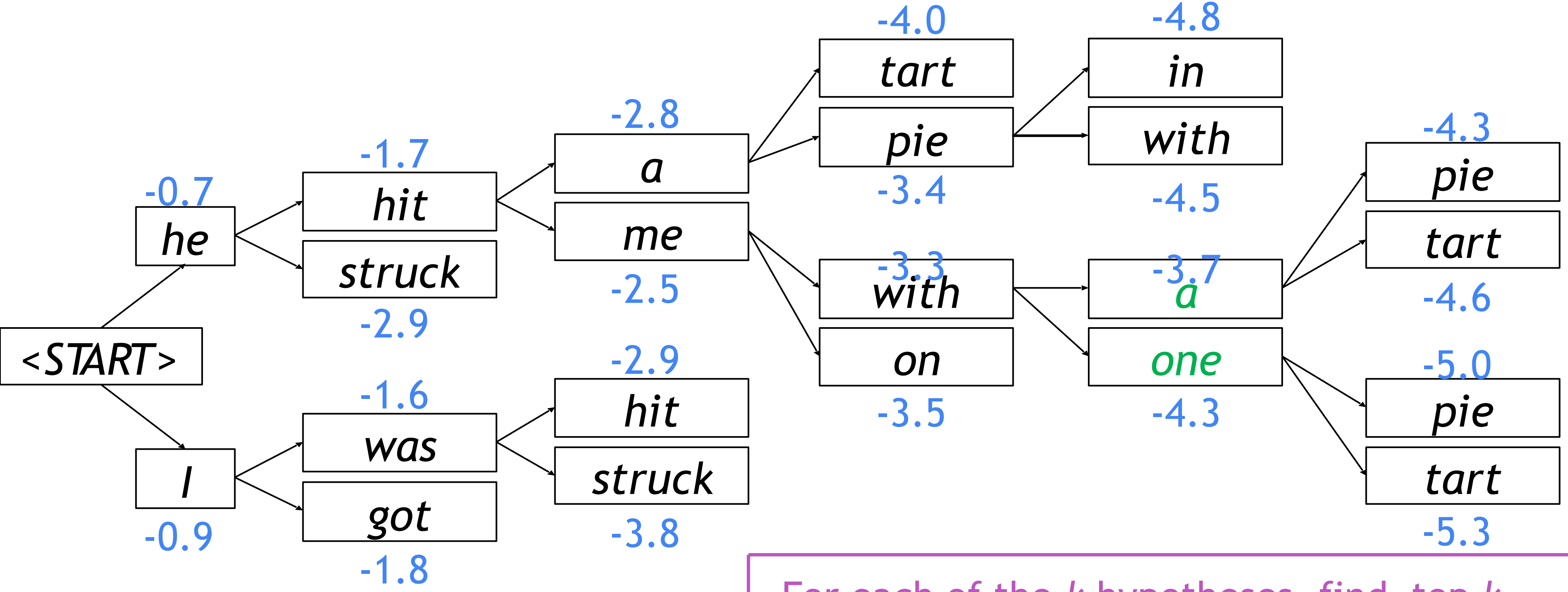
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

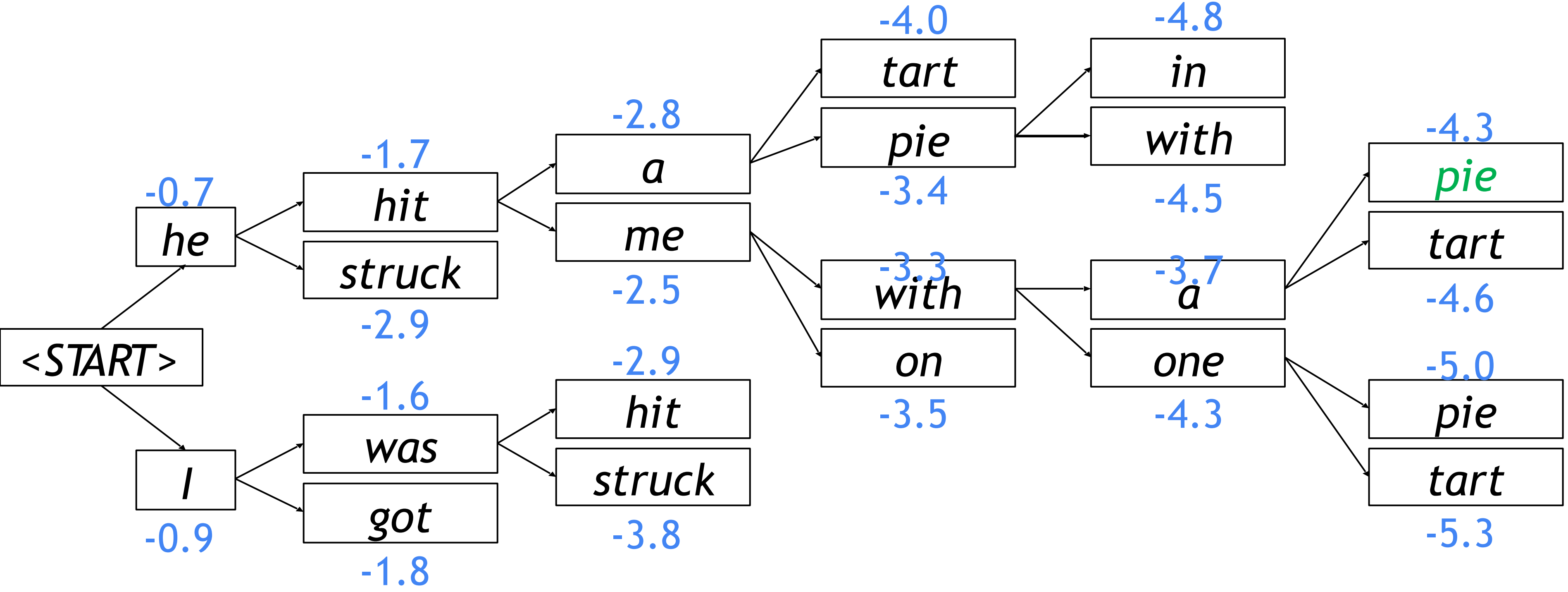
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

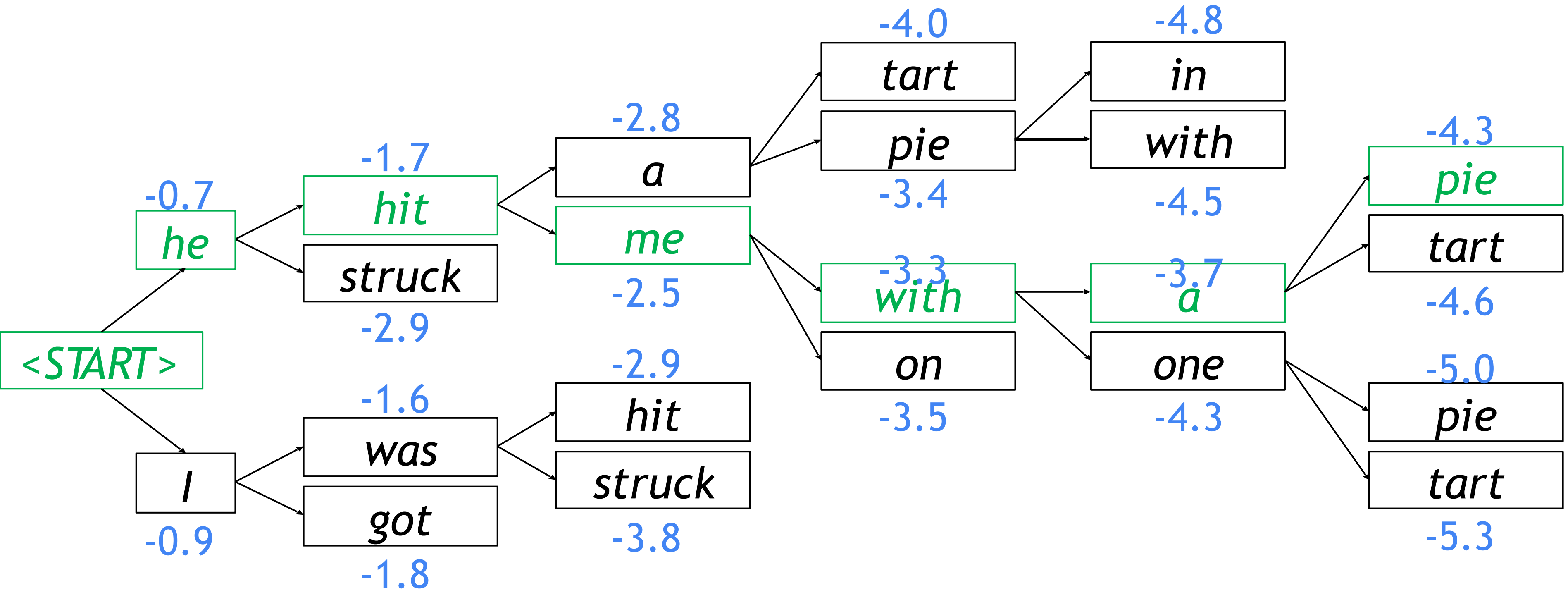
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



This is the top-scoring hypothesis!

Beam search decoding: example

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

- In greedy decoding, usually we decode until the model produces a <END> token
  - <u>For example:</u> *<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

- Problem with this: longer hypotheses have lower scores

- Fix: Normalize by length. Use this to select top one instead:

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

Error analysis on beam search

$P(y^*|x)$

$P(\hat{y}|x)$

Human: Jane visits Africa in September. (y^*)

Algorithm: Jane visited Africa last September. ($\hat{y}$)

Case 1: $P(y^*|x) > P(\hat{y}|x) \leftarrow$

$\arg\max_y P(y|x)$

Beam search chose $\hat{y}$. But y^* attains higher $\boxed{P(y|x)}$.

Conclusion: Beam search is at fault.

Case 2: $P(y^*|x) \leq P(\hat{y}|x) \leftarrow$

y^* is a better translation than $\hat{y}$. But RNN predicted $\underline{P(y^*|x)} < \underline{P(\hat{y}|x)}$.

Conclusion: RNN model is at fault.

Andrew Ng

Beam Search

- if we find that beam search is responsible for a lot of errors, then we increase the beam width
- if we find that the seq2seq model is at fault, then we could do a deeper layer of analysis to try to figure out if we want to add regularization, or get more training data, or try a different network architecture, or something else.

Ref: https://medium.com/@dhartidhami/beam-search-in-seq2seq-model-7606d55b21a5

So is Machine Translation solved?

- **Nope!**
- Many difficulties remain:
  - Out-of-vocabulary words
  - Domain mismatch between train and test data
  - Maintaining context over longer text
  - Low-resource language pairs

Further reading: *"Has AI surpassed humans at translation? Not even close!"*
https://www.skynettoday.com/editorials/state_of_nmt


Thank you