Introduction to Natural Language Processing

Goals, Main Problems

AUA, DS330 Deep Learning 2021, Hovhannes Tamoyan

Sequence to Sequence Modeling

Sequence-to-Sequence Models

Example usages of Seq2Seq architecture:

- Machine Translation (sentence in A lang -> sentence in B lang)
- Text Summarization (long text -> short text)
- Dialogue (previous utterances -> next utterance)
- Code Generation (natural language -> Python code)

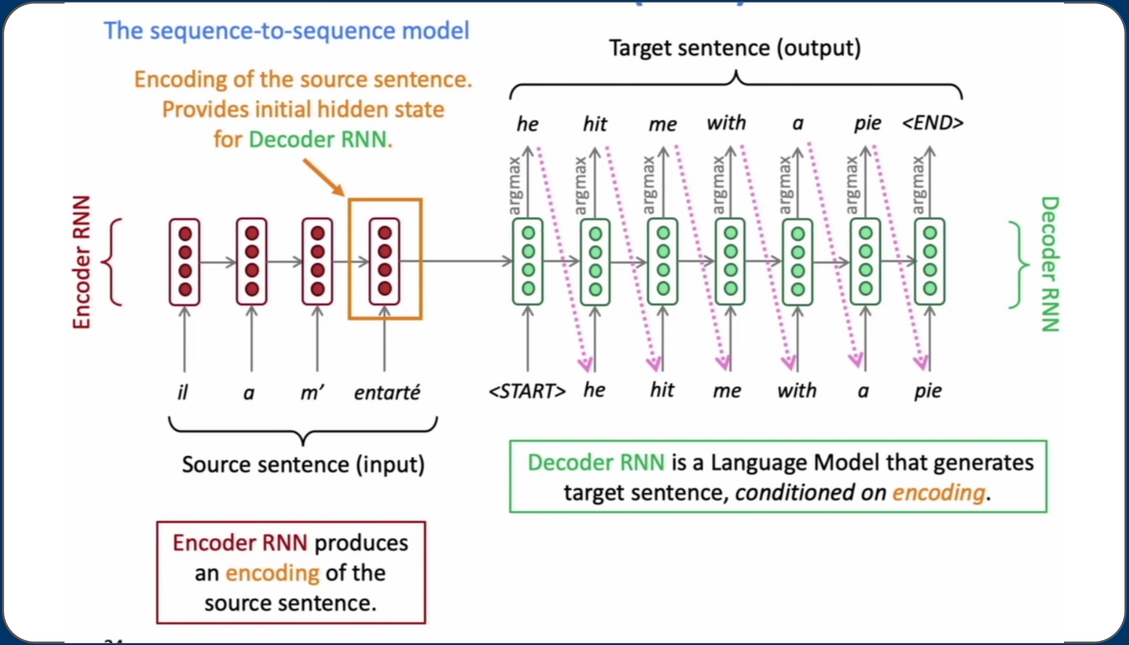
Seq2Seq architectures involve two RNNs. One is for encoding a sequence and the other one for decoding.

We will consider the Neural Machine Translation problem.

It is a way to do Machine Translation with a single NN.

NMT

The Decoder is a conditional LM, which takes the encoded output from the Encoder and generates a new sequence.


The sequence-to-sequence model
Target sentence (output)
Encoding of the source sentence. Provides initial hidden state for Decoder RNN.
he hit me with a pie <END>
argmax
Encoder RNN
Decoder RNN
il a m' entarté
<START> he hit me with a pie
Source sentence (input)
Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.
Encoder RNN produces an encoding of the source sentence.

NMT - task formulation

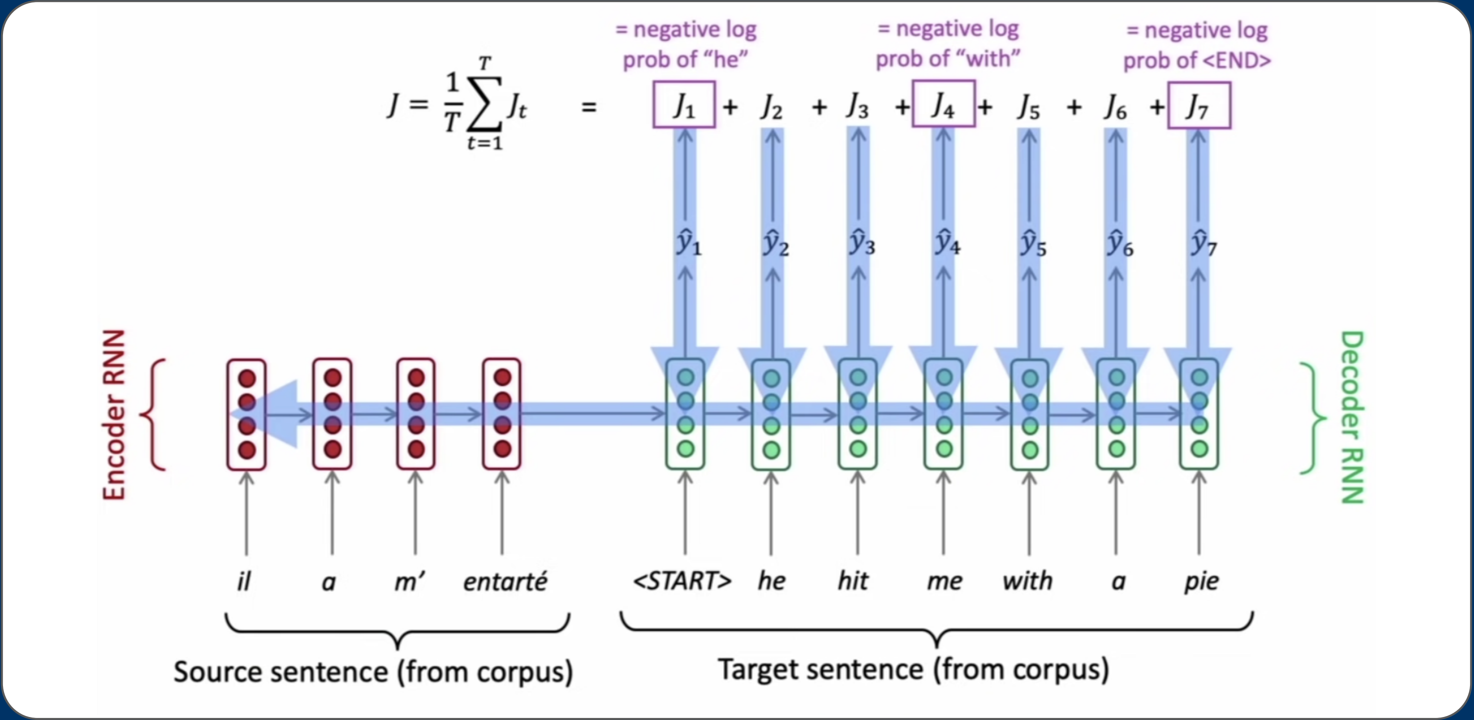
Probability of next target word, given target words so far and source sentence x.

$$P(y|x) = P(y_1|x)P(y_2|y_1, x)P(y_3|y_1, y_2, x)\dots P(y_T|y_1, y_2 \dots y_{T-1}, x)$$

So how to train this system?

Get a Big parallel corpus and do the following:

NMT - Training Architecture


= negative log prob of "he"
= negative log prob of "with"
= negative log prob of <END>
$J = \frac{1}{T}\sum_{t=1}^{T} J_t = \boxed{J_1} + J_2 + J_3 + \boxed{J_4} + J_5 + J_6 + \boxed{J_7}$
$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$ $\hat{y}_5$ $\hat{y}_6$ $\hat{y}_7$
Encoder RNN
Decoder RNN
il a m' entarté
<START> he hit me with a pie
Source sentence (from corpus)
Target sentence (from corpus)

Decoding Algorithms

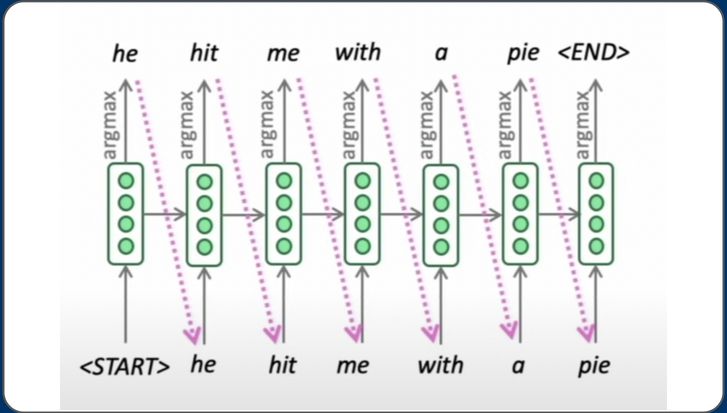
How to generate/choose the most plausible output sentence?

Decoding strategies (most common):

- Greedy Decoding
- Beam Search

Greedy Decoding

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



This is **greedy decoding** - take most probable word on each step

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 too expensive.

Beam Search Decoding

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:

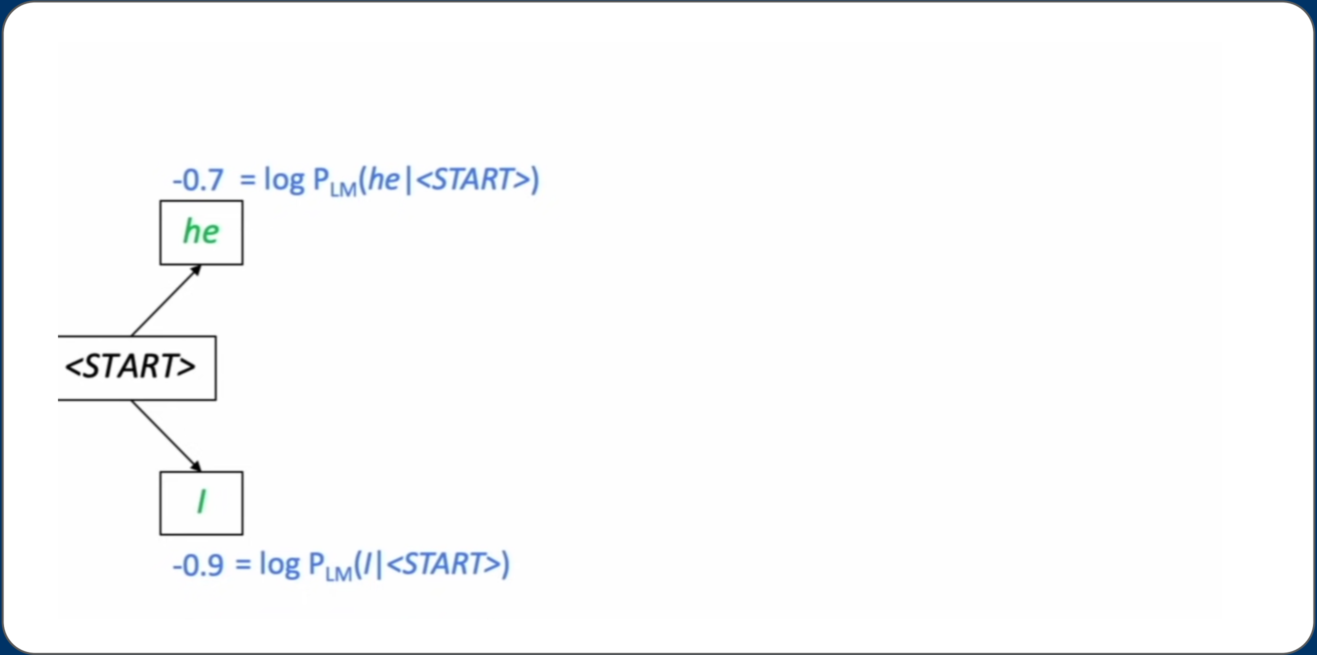
$$score(y_1,\ldots,y_t) = logP_{LM}(y_1,\ldots,y_t|x) = \sum_{i=1}^{t} logP_{LM}(y_i|y_1,\ldots,y_{i-1},x)$$

The scores are all negative, and higher score is better

We search for high-scoring hypotheses, tracking top k on each step.

Beam Search Decoding: Example

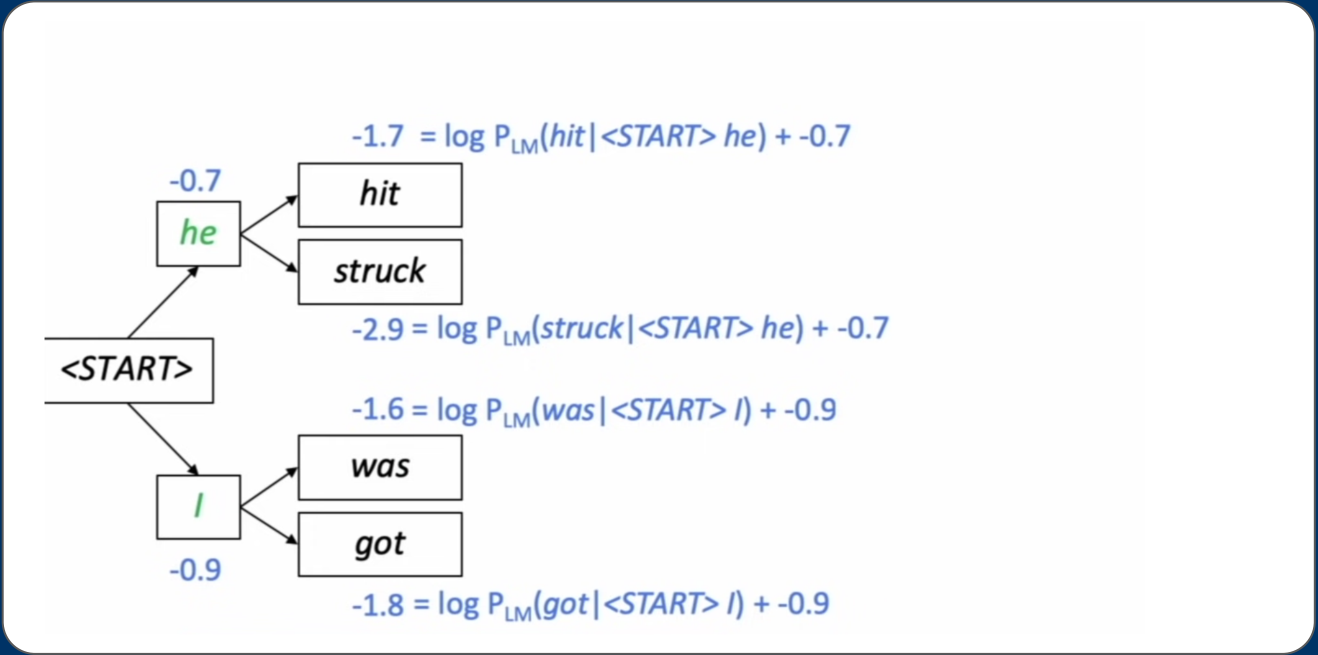
Beam size: k=2



-0.7 = log P_{LM}(*he*|<START>)
he
<START>
I
-0.9 = log P_{LM}(*I*|<START>)

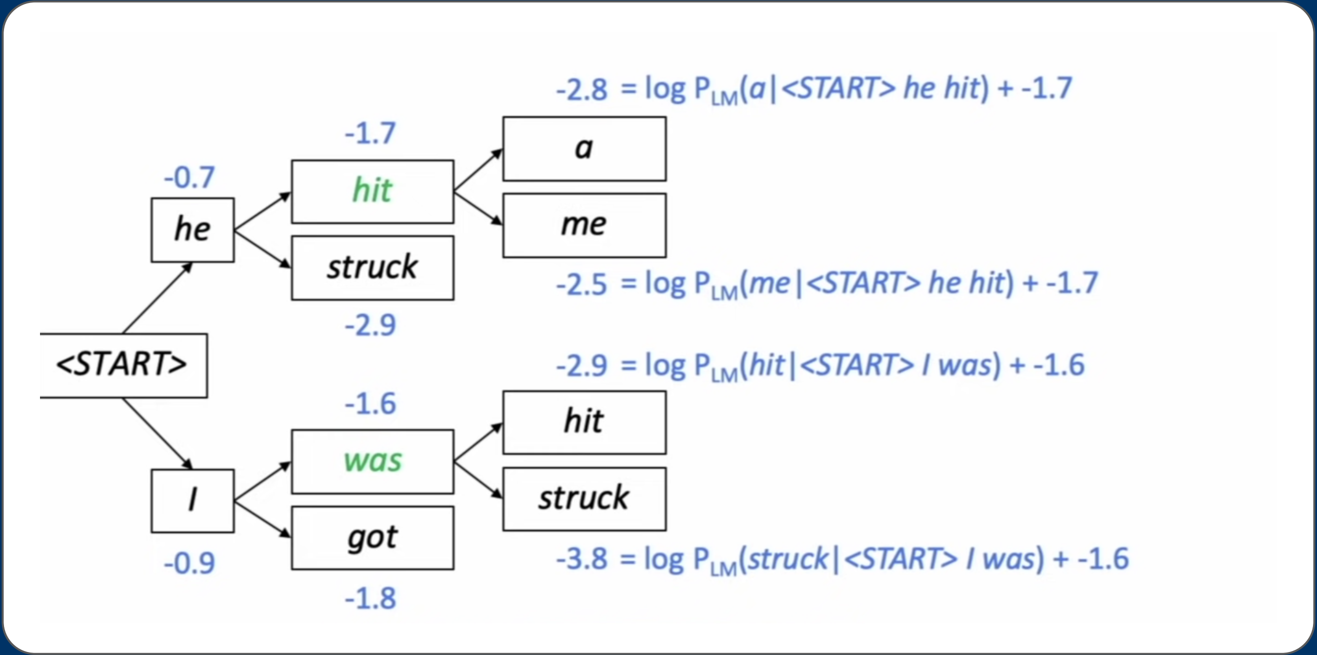
Beam Search Decoding: Example

Beam size: k=2


-1.7 = log P_LM(hit|<START> he) + -0.7
-0.7
hit
he
struck
-2.9 = log P_LM(struck|<START> he) + -0.7
<START>
-1.6 = log P_LM(was|<START> I) + -0.9
was
I
got
-0.9
-1.8 = log P_LM(got|<START> I) + -0.9

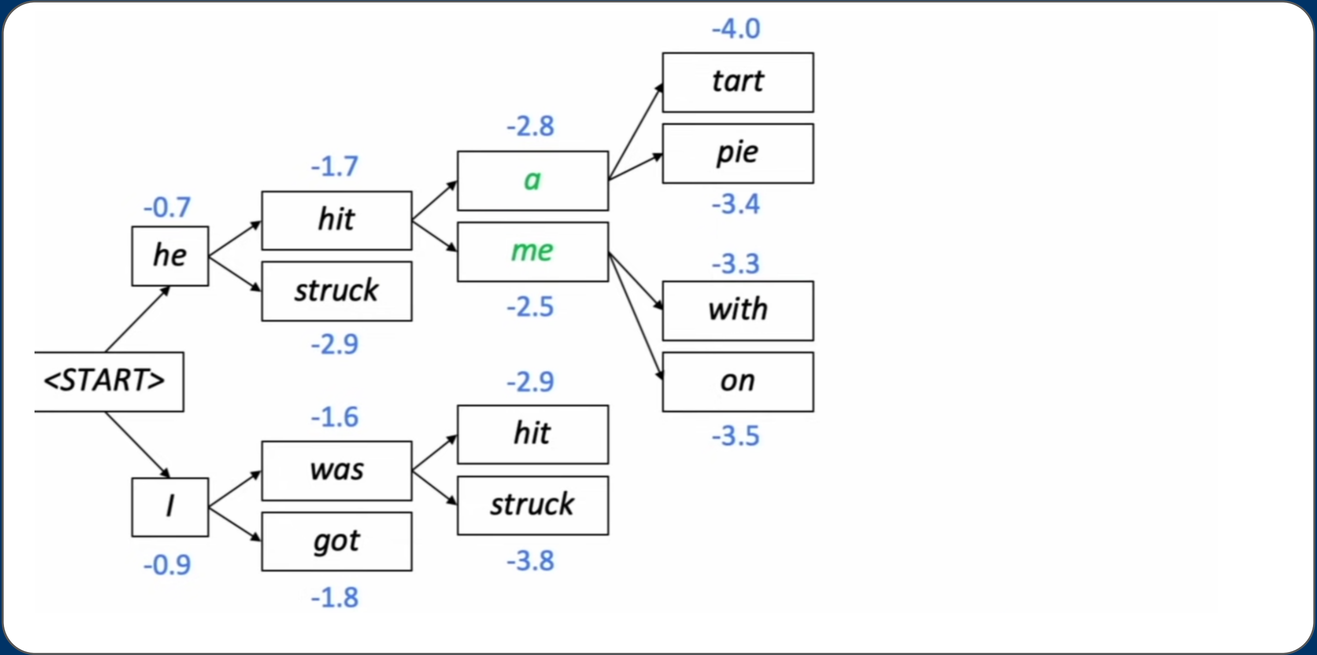
Beam Search Decoding: Example

Beam size: k=2


-2.8 = log P_LM(a|<START> he hit) + -1.7
-1.7
-0.7
a
hit
he
me
struck
-2.5 = log P_LM(me|<START> he hit) + -1.7
-2.9
<START>
-2.9 = log P_LM(hit|<START> I was) + -1.6
-1.6
hit
was
I
struck
got
-0.9
-3.8 = log P_LM(struck|<START> I was) + -1.6
-1.8

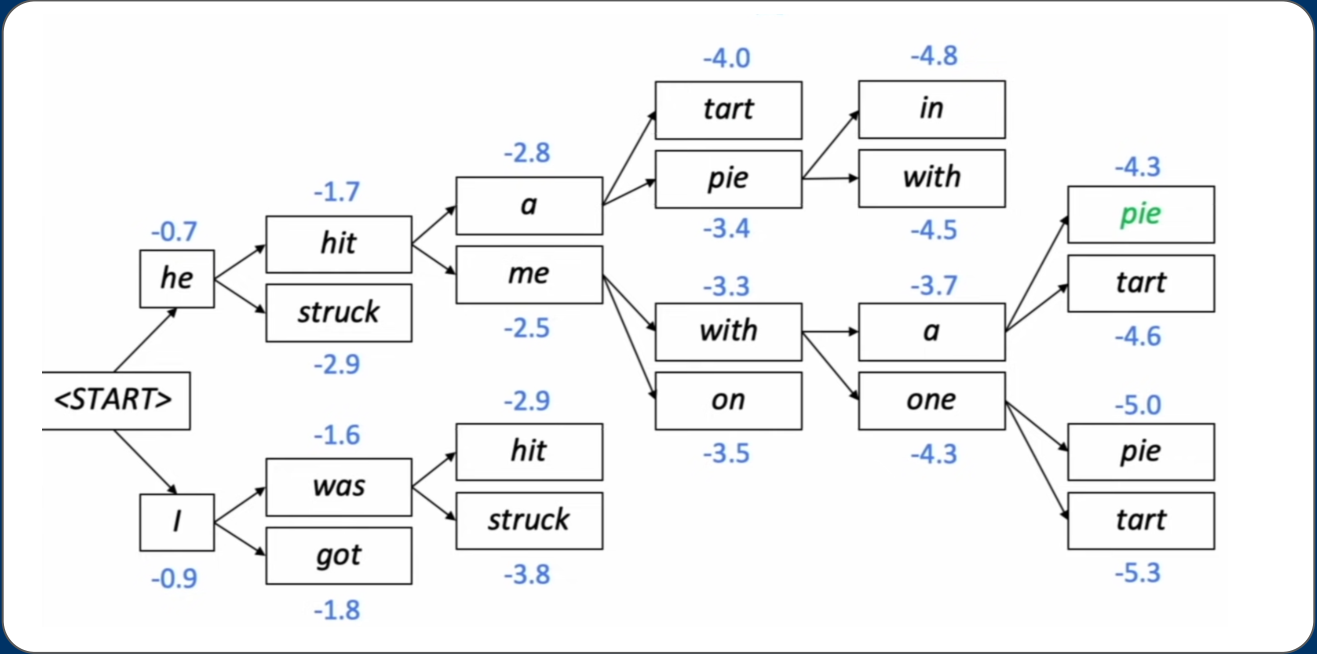
Beam Search Decoding: Example

Beam size: k=2



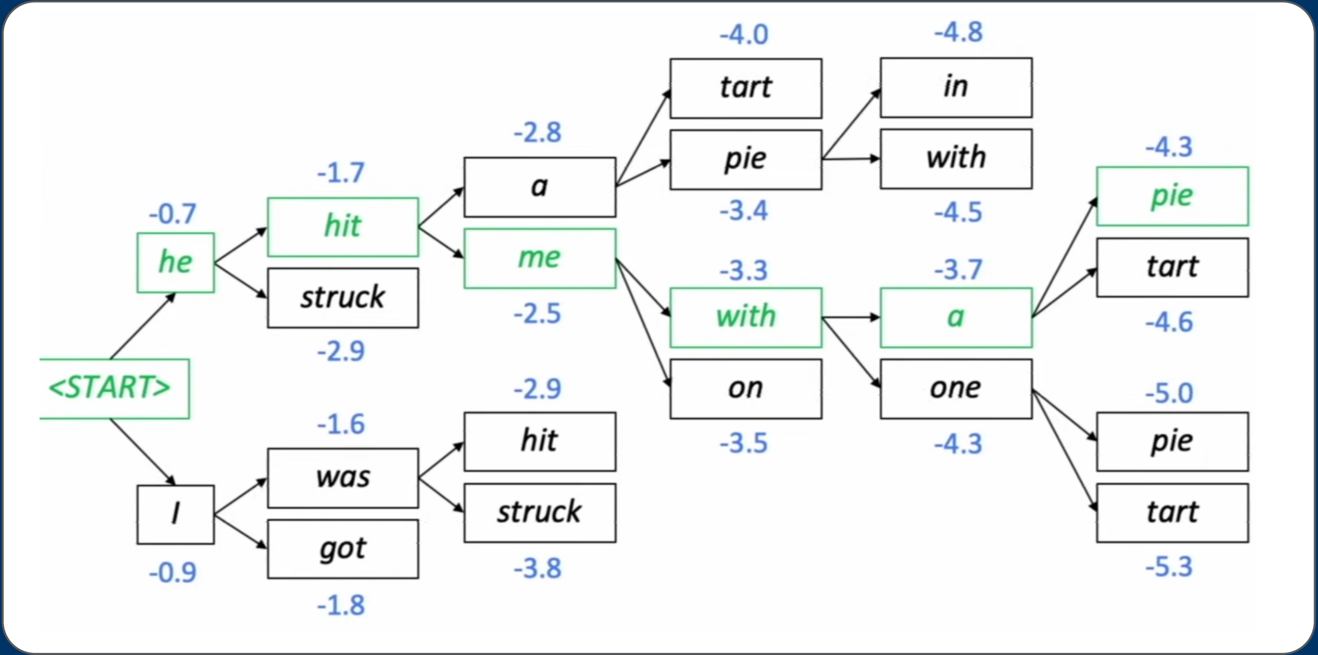
Beam Search Decoding: Example

Beam size: k=2



Beam Search Decoding: Example

Beam size: k=2



Beam Search: Stopping Criterion

In **greedy decoding**, usually we decode until the model produces a <END> token.

In beam **search decoding**, different hypotheses may produce <END> token 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 reached timestep T (where T is some predefined cutoff), or
- We have at least n completed hypothesis (where n i predefined cutoff)

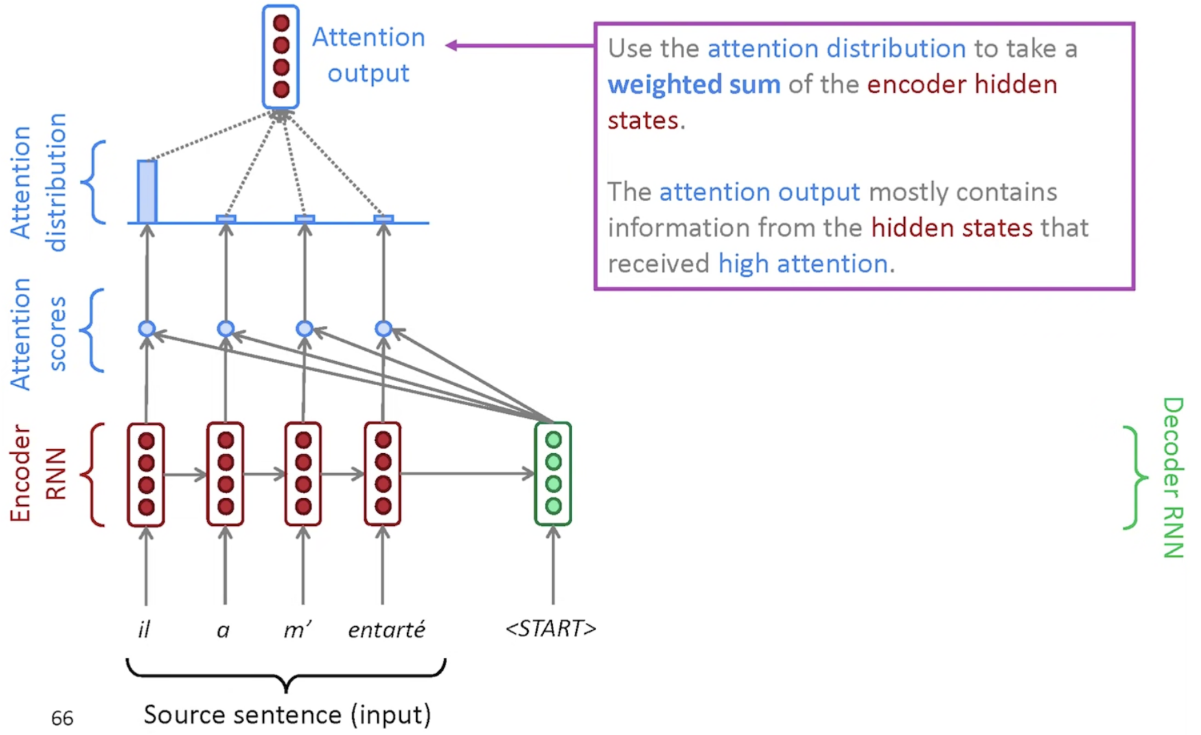
Attention

The NMT can be potentially improved with the mechanism of attention.

Attention provides a solution to the bottleneck problem, which is we are pushing all the information in the encoder state to a single vector.

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

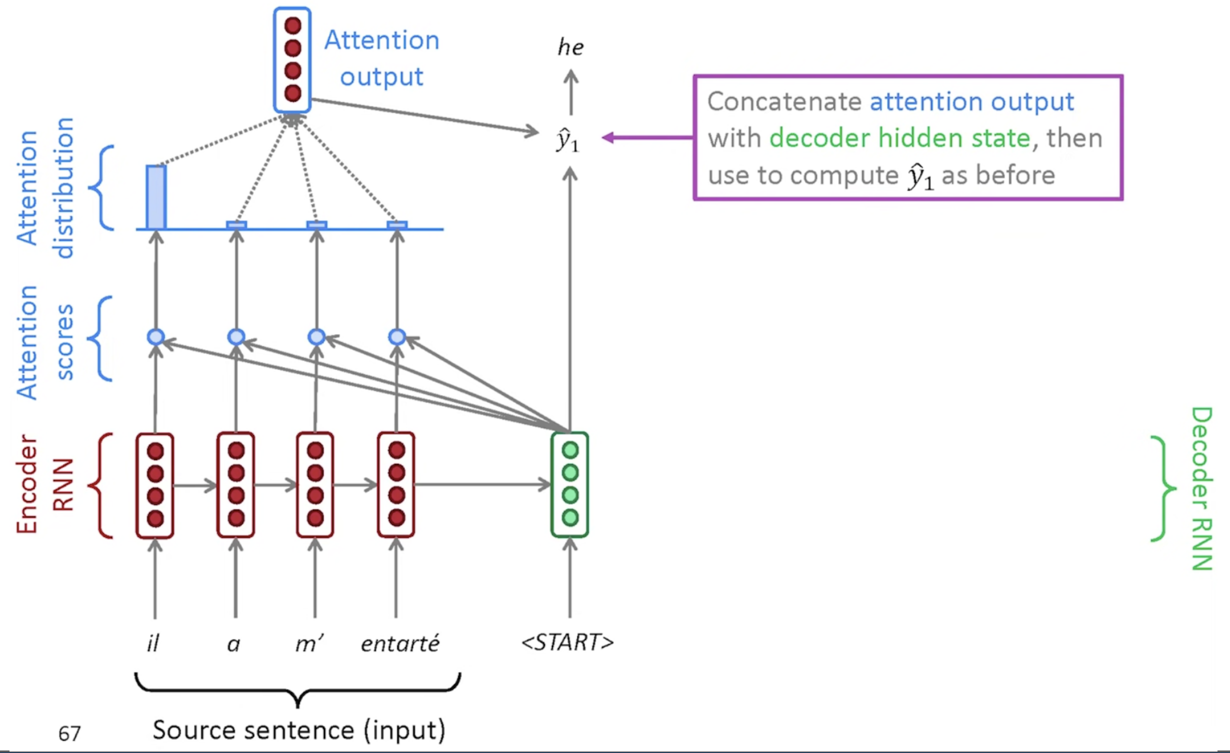
Attention: Example



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

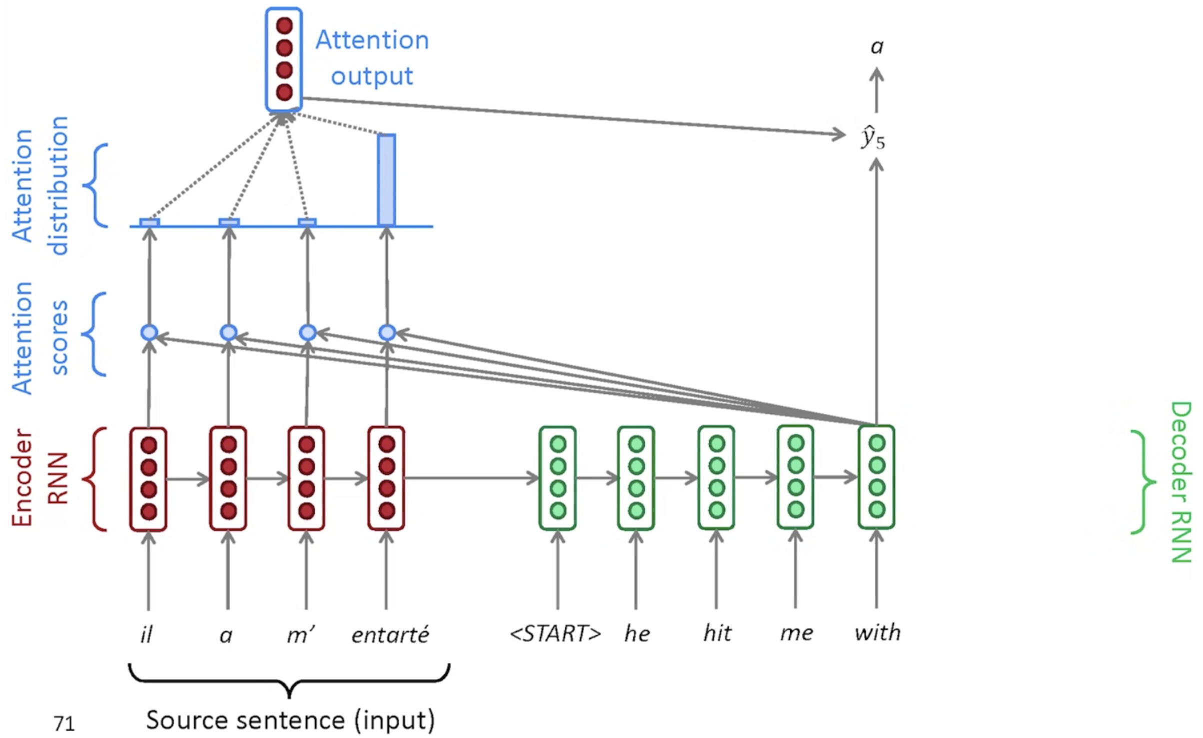
The attention output mostly contains information from the hidden states that received high attention.

Attention: Example



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

Attention: Example



Attention

- 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

Thank You