

# Natural Language Processing with Deep Learning

## CS224N/Ling284



Lecture 8:  
Machine Translation,  
Sequence-to-sequence and Attention

**Abigail See, Matthew Lamm**

# Announcements

- We are taking attendance today
  - Sign in with the TAs outside the auditorium
  - No need to get up now - there will be plenty of time to sign in after the lecture ends
  - For attendance policy special cases, see Piazza post for clarification
- Assignment 4 content covered today
  - Get started early! The model takes 4 hours to train!
- Mid-quarter feedback survey:
  - Will be sent out sometime in the next few days (watch Piazza).
  - Complete it for 0.5% credit

# Overview

Today we will:

- Introduce a new task: Machine Translation



is a major use-case of

- Introduce a new neural architecture: sequence-to-sequence



is improved  
by

- Introduce a new neural technique: attention

# Section 1: Pre-Neural Machine Translation

# Machine Translation

**Machine Translation (MT)** is the task of translating a sentence  $x$  from one language (the **source language**) to a sentence  $y$  in another language (the **target language**).

x: *L'homme est né libre, et partout il est dans les fers*



y: *Man is born free, but everywhere he is in chains*

- Rousseau

# 1950s: Early Machine Translation

Machine Translation research began in the early 1950s.

- Russian → English  
(motivated by the Cold War!)



1 minute video showing 1954 MT:  
<https://youtu.be/K-HfpsHPmvw>

- Systems were mostly **rule-based**, using a bilingual dictionary to map Russian words to their English counterparts

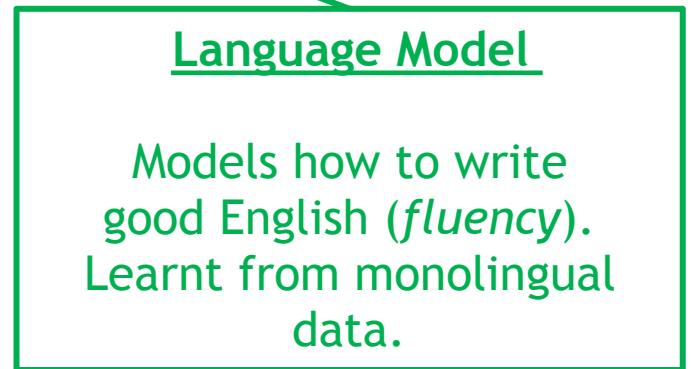
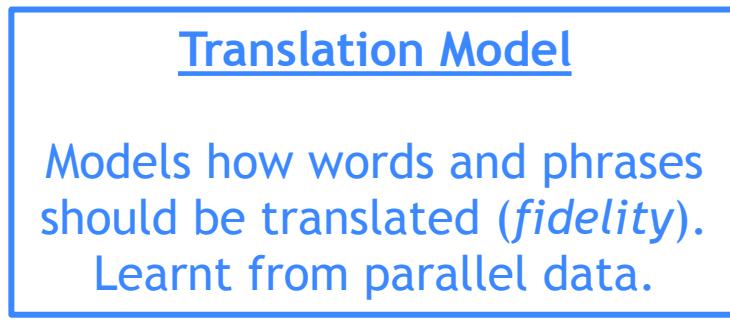
# 1990s-2010s: Statistical Machine Translation

- Core idea: Learn a **probabilistic model** from data
- Suppose we're translating French → English.
- We want to find **best English sentence  $y$ , given French sentence  $x$**

$$\operatorname{argmax}_y P(y|x)$$

- Use Bayes Rule to break this down into **two components** to be learnt separately:

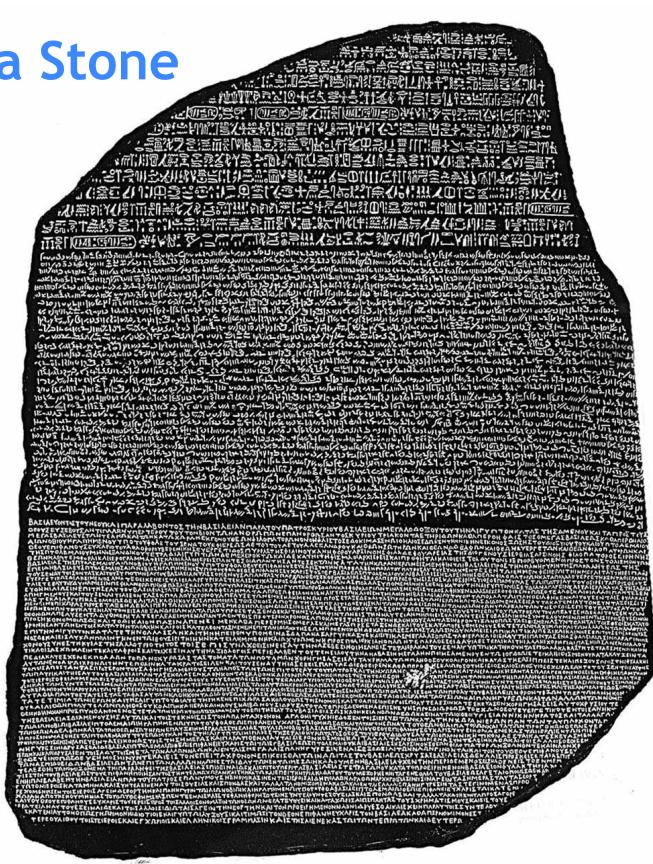
$$= \operatorname{argmax}_y P(x|y)P(y)$$



# 1990s-2010s: Statistical Machine Translation

- Question: How to learn translation model  $P(x|y)$  ?
- First, need large amount of **parallel data**  
(e.g. pairs of human-translated French/English sentences)

The Rosetta Stone



Ancient Egyptian

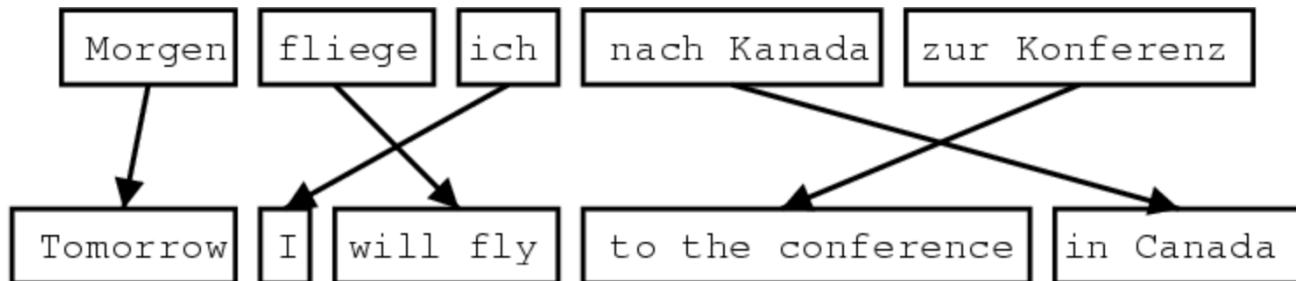
Demotic

Ancient Greek

# Learning alignment for SMT

- Question: How to learn translation model  $P(x|y)$  from the parallel corpus?
- Break it down further: Introduce latent variable into the model:  
$$P(x, a|y)$$

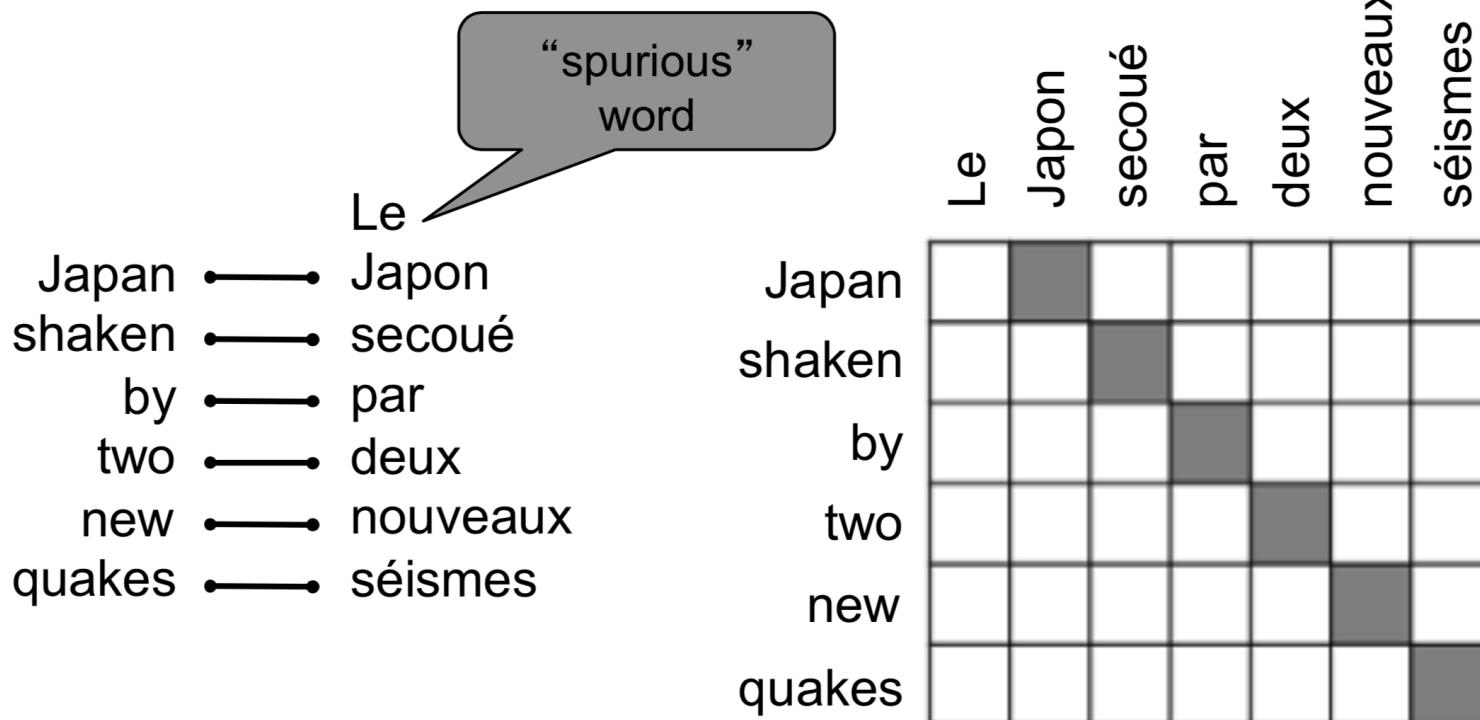
where  $a$  is the **alignment**, i.e. word-level correspondence between source sentence  $x$  and target sentence  $y$



# What is alignment?

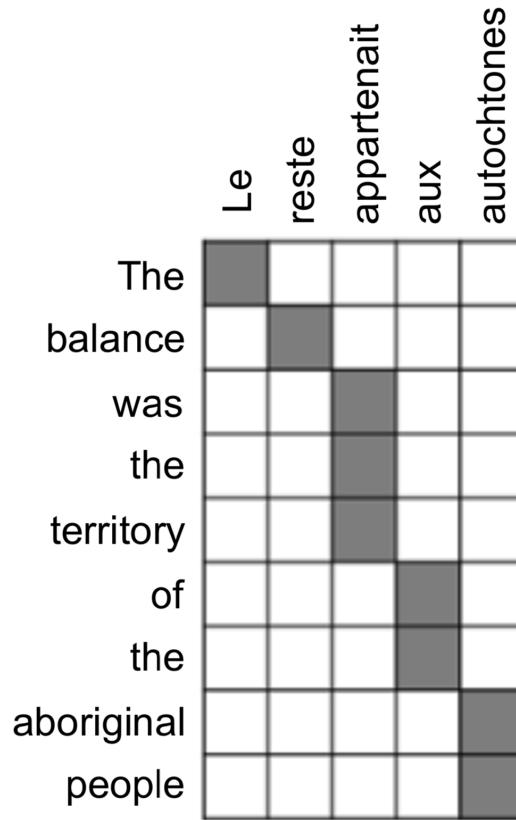
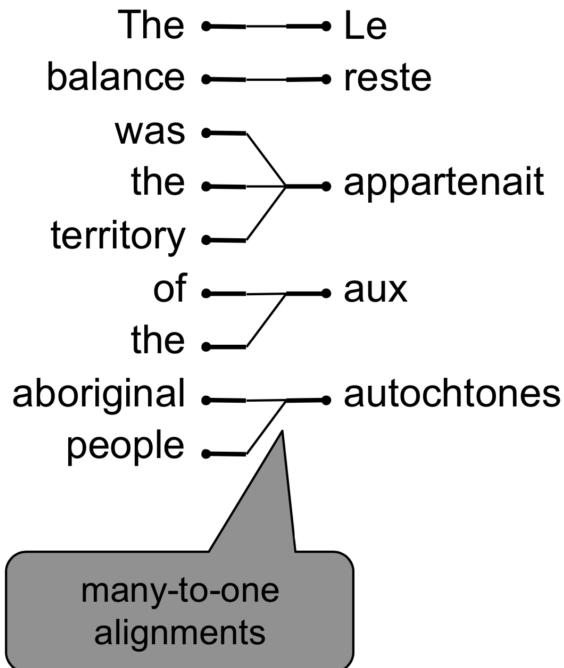
Alignment is the correspondence between particular words in the translated sentence pair.

- Typological differences between languages lead to complicated alignments!
- Note: Some words have no counterpart



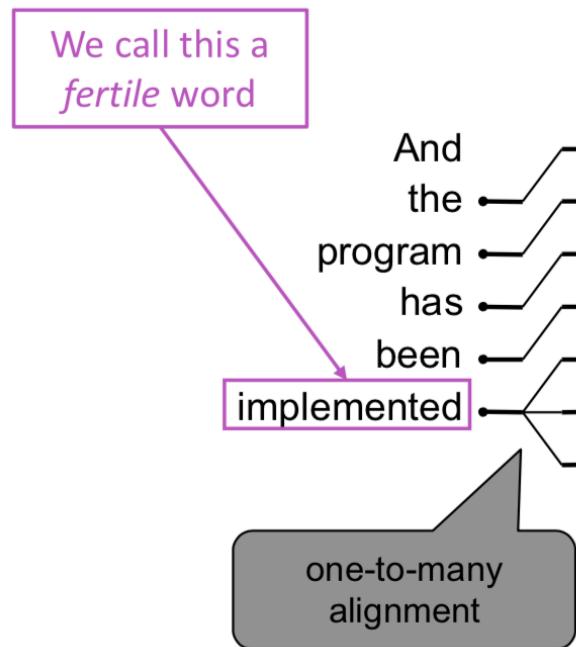
# Alignment is complex

Alignment can be many-to-one



# Alignment is complex

Alignment can be one-to-many

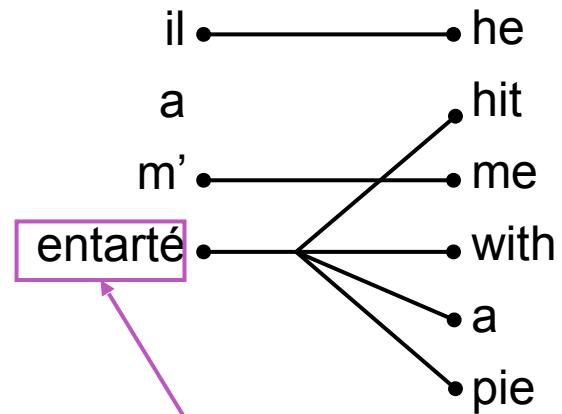


And	Le	programme				
the	a	été				
program	mis					
has	en					
been	application					
implemented						

Le programme a été mis en application

# Alignment is complex

Some words are very fertile!



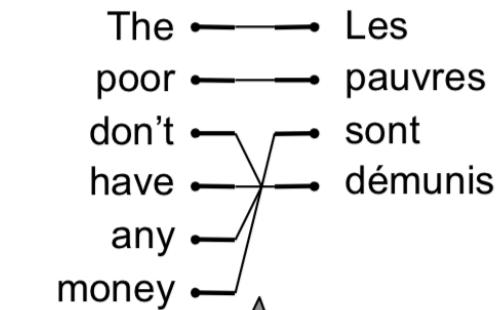
This word has no single-word equivalent in English

	he	hit	me	with	a	pie
il						
a						
m'						
entarté						



# Alignment is complex

Alignment can be **many-to-many** (phrase-level)



many-to-many  
alignment

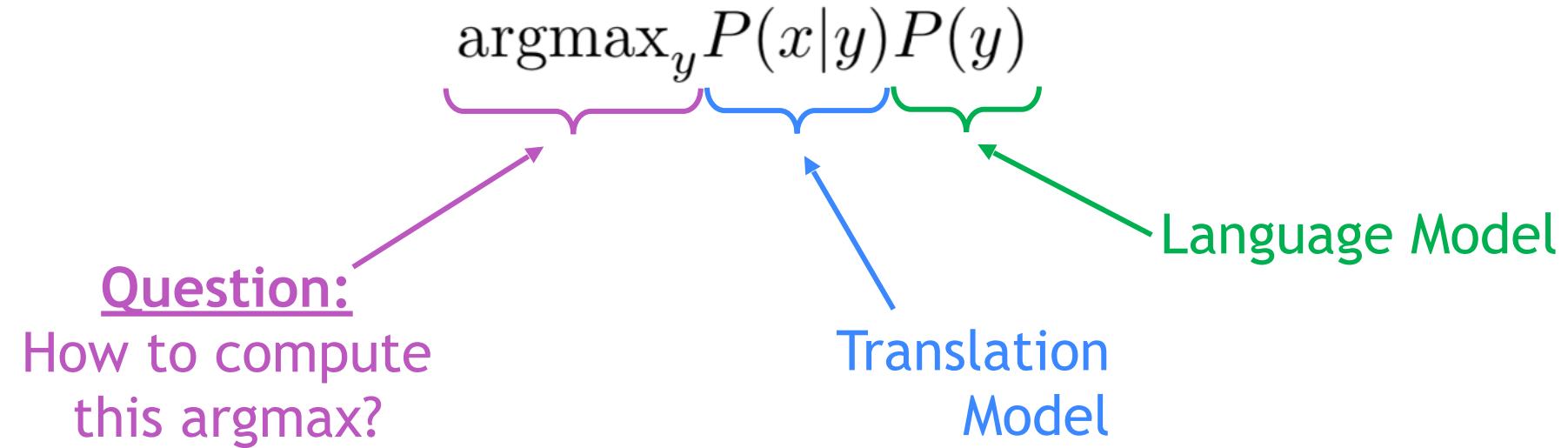
	Les	pauvres	sont	démunis
The				
poor				
don't				
have				
any				
money				

phrase  
alignment

# Learning alignment for SMT

- We learn  $P(x, a|y)$  as a combination of many factors, including:
  - Probability of particular words aligning (also depends on position in sent)
  - Probability of particular words having particular fertility (number of corresponding words)
  - etc.
- Alignments  $a$  are **latent variables**: They aren't explicitly specified in the data!
  - Require the use of special learning algos (like Expectation-Maximization) for learning the parameters of distributions with latent variables (CS 228)

# Decoding for SMT

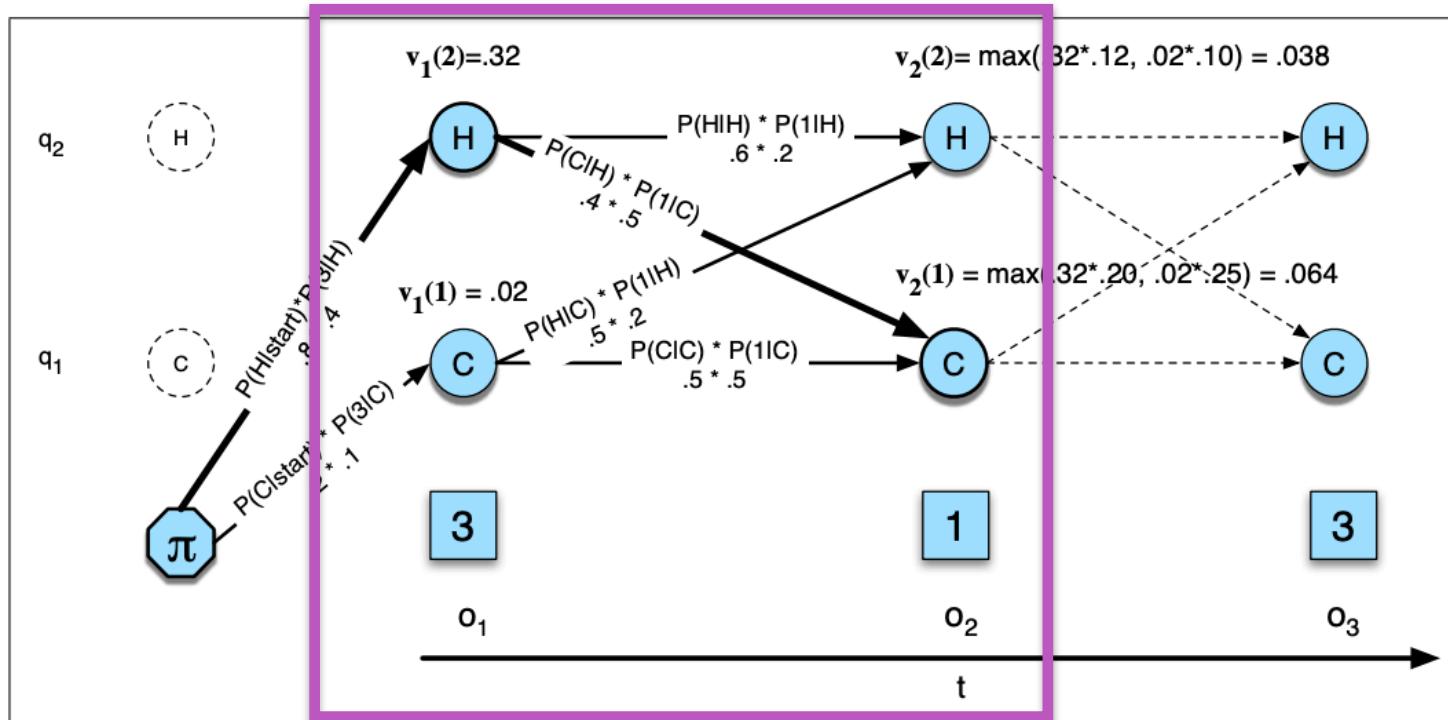


- We could enumerate every possible  $y$  and calculate the probability? → Too expensive!
- Answer: Impose strong **independence assumptions** in model, use dynamic programming for globally optimal solutions (e.g. Viterbi algorithm).
- This process is called *decoding*

# Viterbi: Decoding with Dynamic Programming

- Impose strong independence assumptions in model:

$$p(x, a|y) = \prod_{j=1}^{l_x} p(x_j | f_a(j))$$



Source: "Speech and Language Processing", Chapter A, Jurafsky and Martin, 2019.

# 1990s-2010s: Statistical Machine Translation

- SMT was a huge research field
- The best systems were extremely complex
  - Hundreds of important details we haven't mentioned here
  - Systems had many separately-designed subcomponents
  - Lots of feature engineering
    - Need to design features to capture particular language phenomena
  - Require compiling and maintaining extra resources
    - Like tables of equivalent phrases
  - Lots of human effort to maintain
    - Repeated effort for each language pair!

## Section 2: Neural Machine Translation

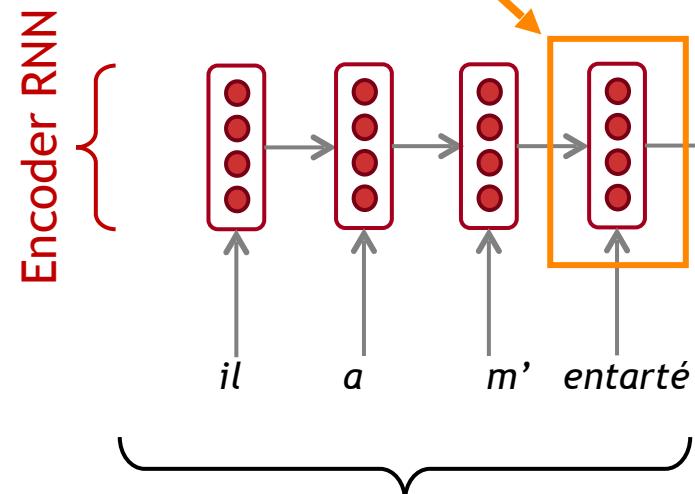
# What is Neural Machine Translation?

- 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*.

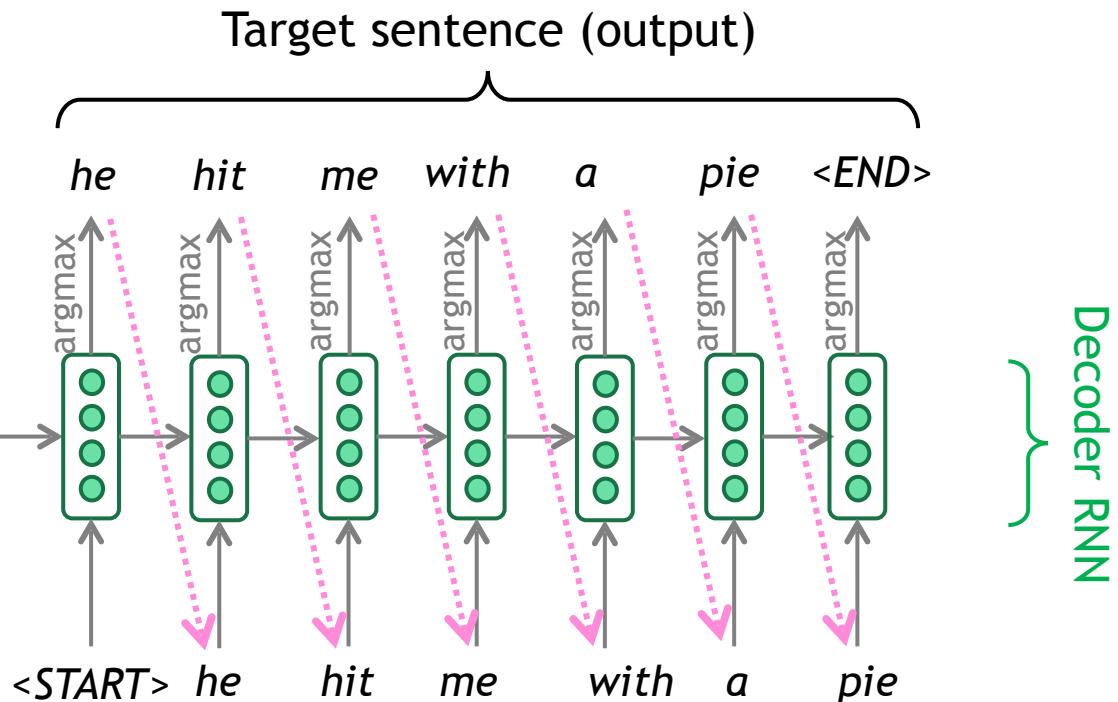
# Neural Machine Translation (NMT)

The sequence-to-sequence model

Encoding of the source sentence.  
Provides initial hidden state  
for Decoder RNN.



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!

- 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)

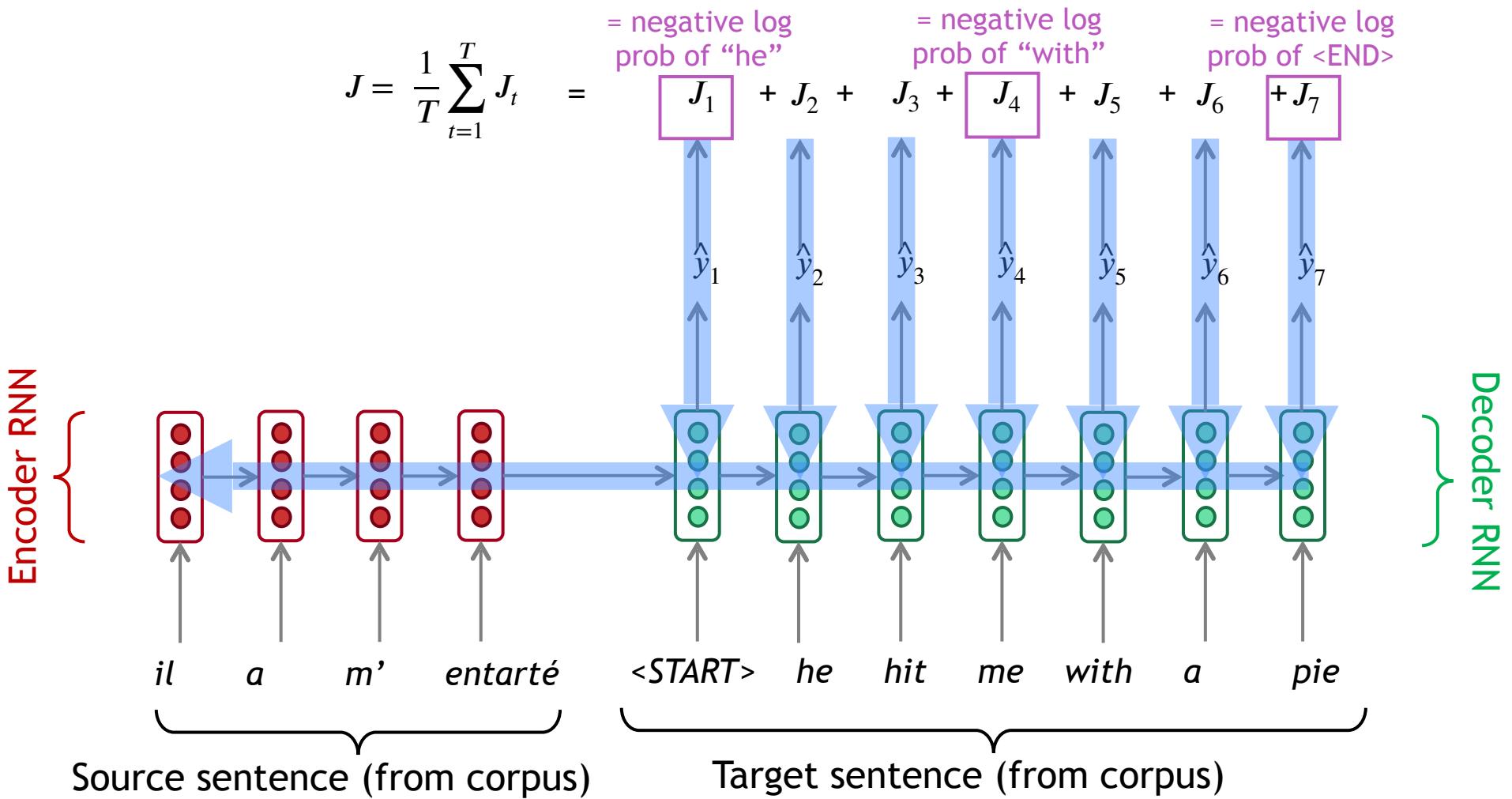
- The sequence-to-sequence model is an example of a **Conditional Language Model**.
  - **Language Model** because the decoder is predicting the next word of the target sentence  $y$
  - **Conditional** because its predictions are *also* conditioned on the source sentence  $x$
- NMT directly calculates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

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

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

- **Question:** How to **train** a NMT system?
- **Answer:** Get a big parallel corpus...

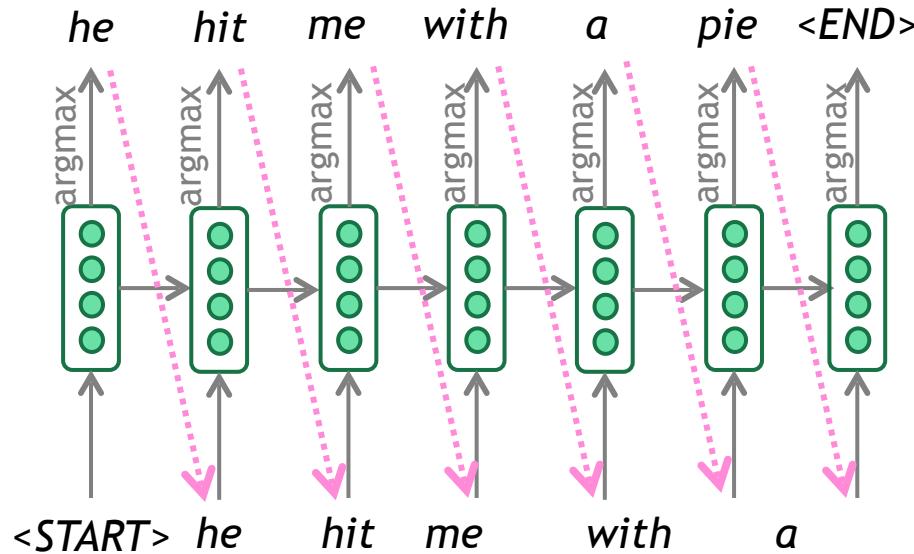
# Training a Neural Machine Translation system



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 <sup>pie</sup> 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

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

- 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  $y_1, \dots, y_t$  has a score which is its log probability:

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

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

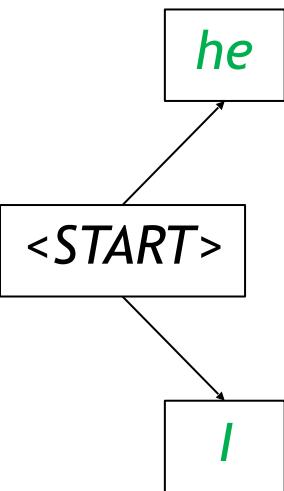
<START>

Calculate prob  
dist of next word

# Beam search decoding: example

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

$$-0.7 = \log P_{\text{LM}}(he | <\text{START}>)$$

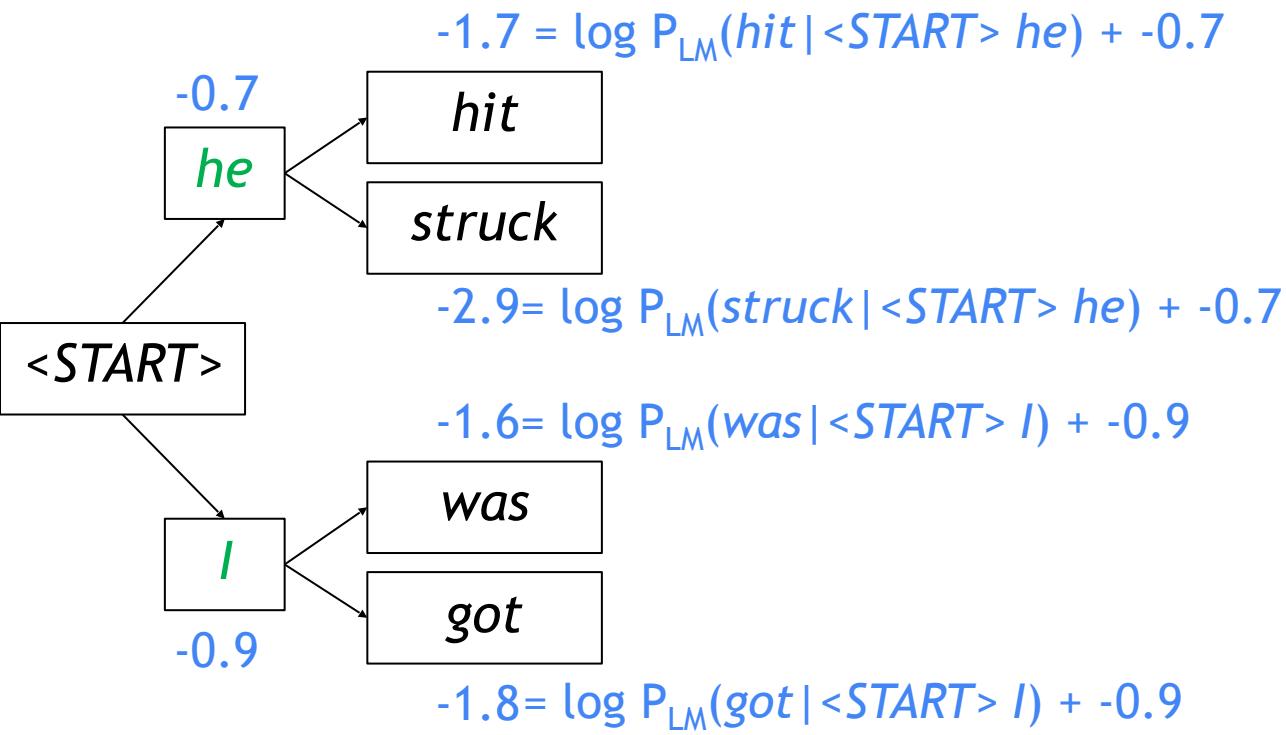


$$-0.9 = \log P_{\text{LM}}(I | <\text{START}>)$$

Take top  $k$  words  
and compute scores

# Beam search decoding: example

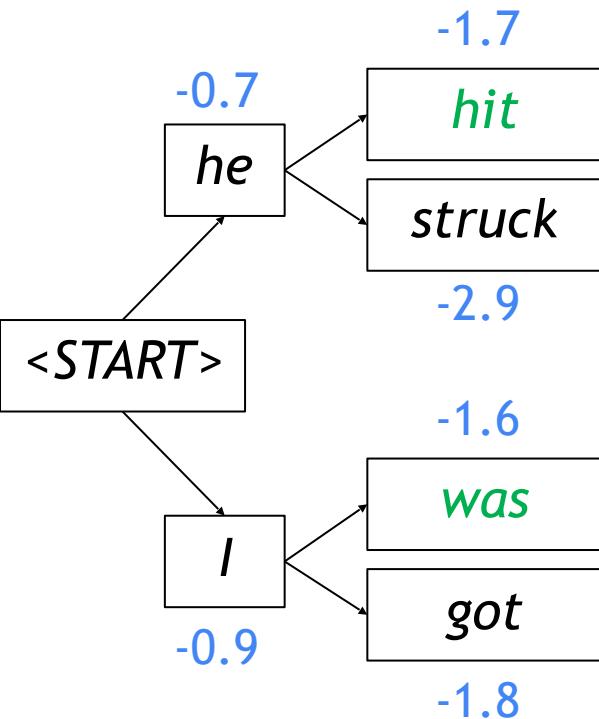
Beam size =  $k = 2$ . Blue numbers =  $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

# Beam search decoding: example

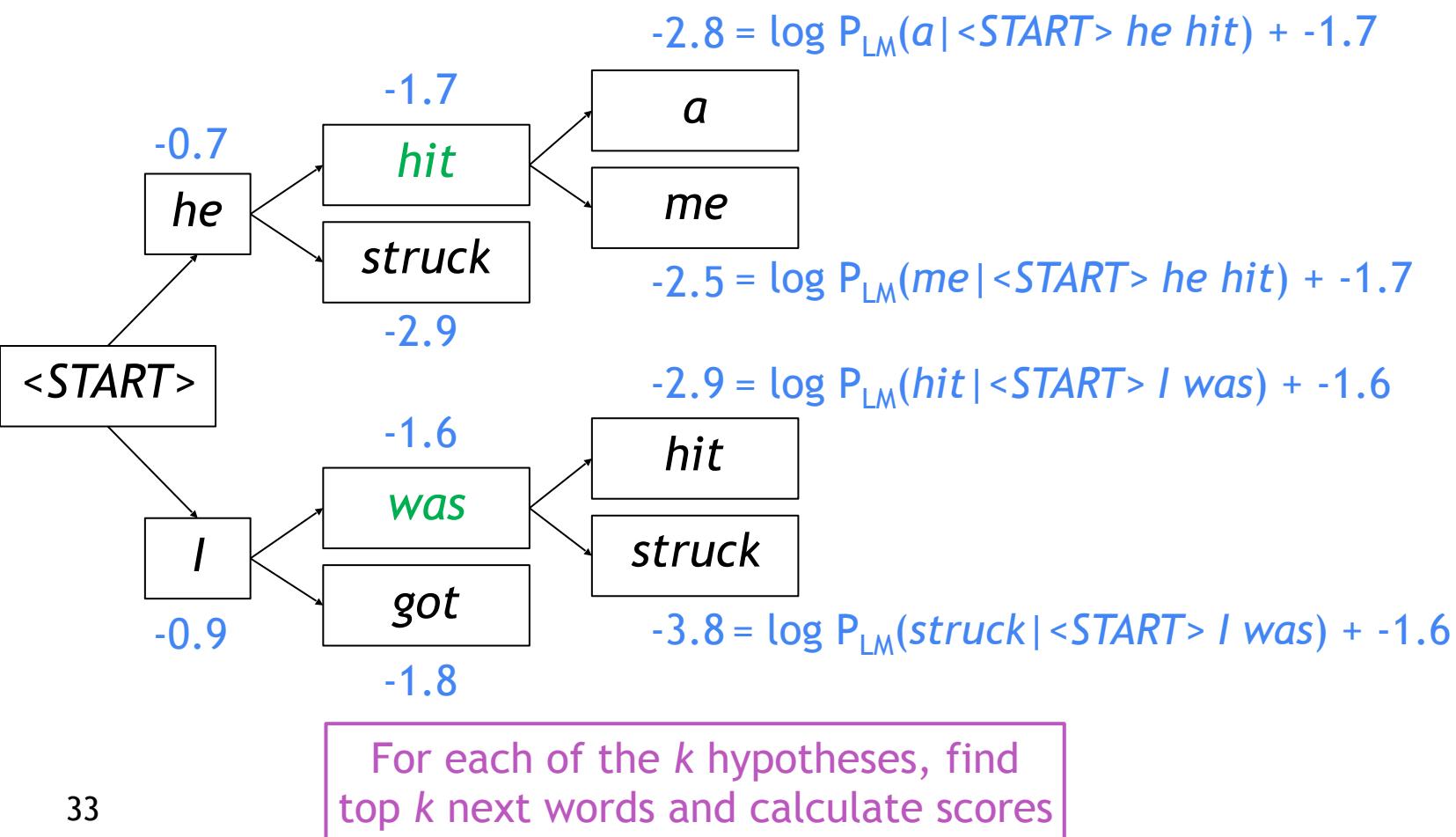
Beam size =  $k = 2$ . Blue numbers =  $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

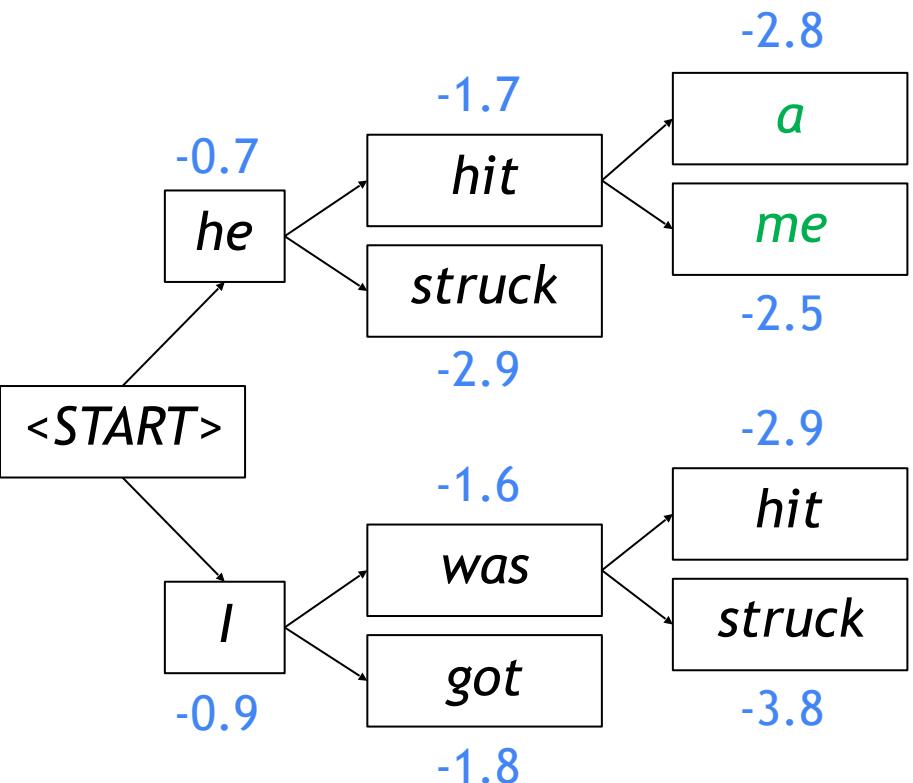
# Beam search decoding: example

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



# Beam search decoding: example

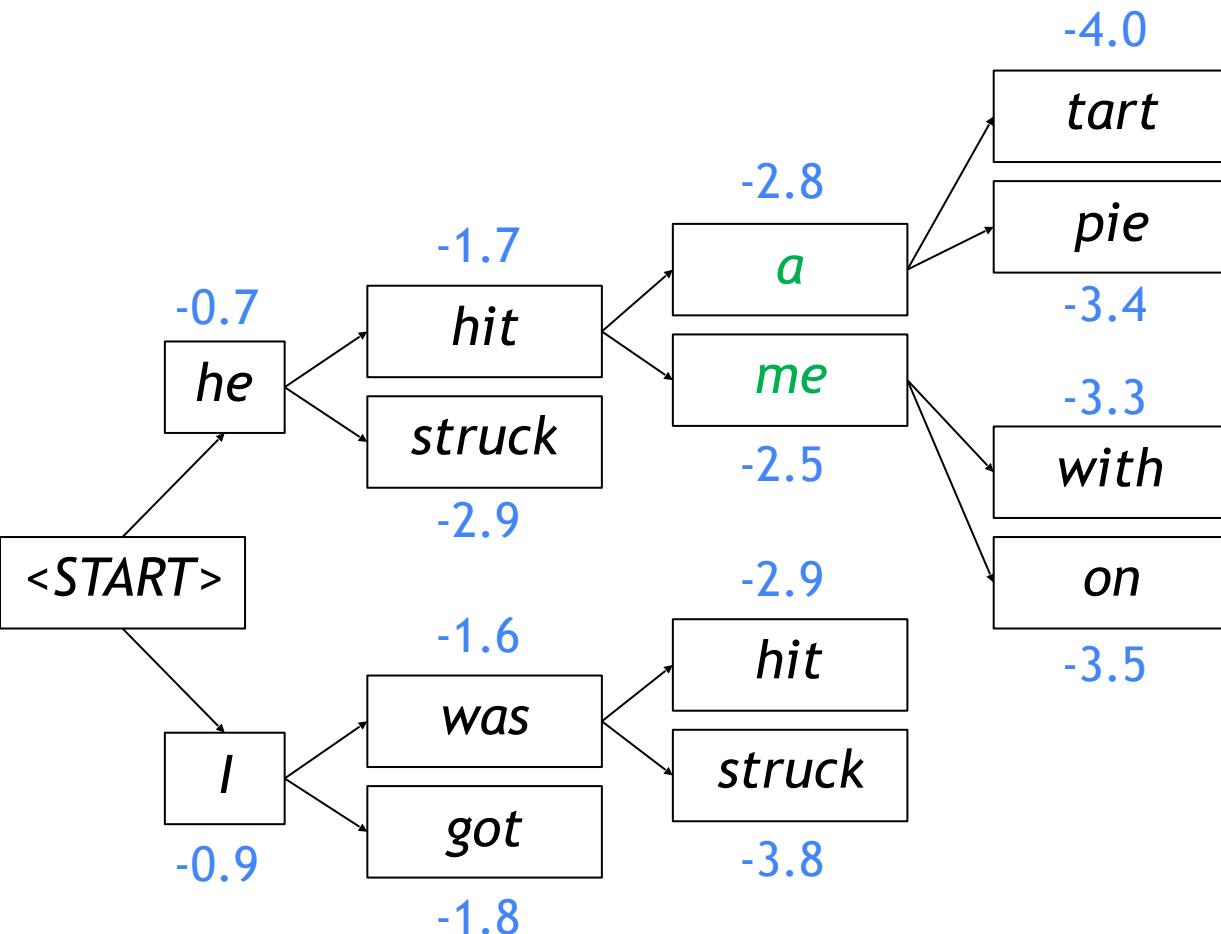
Beam size =  $k = 2$ . Blue numbers =  $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

# Beam search decoding: example

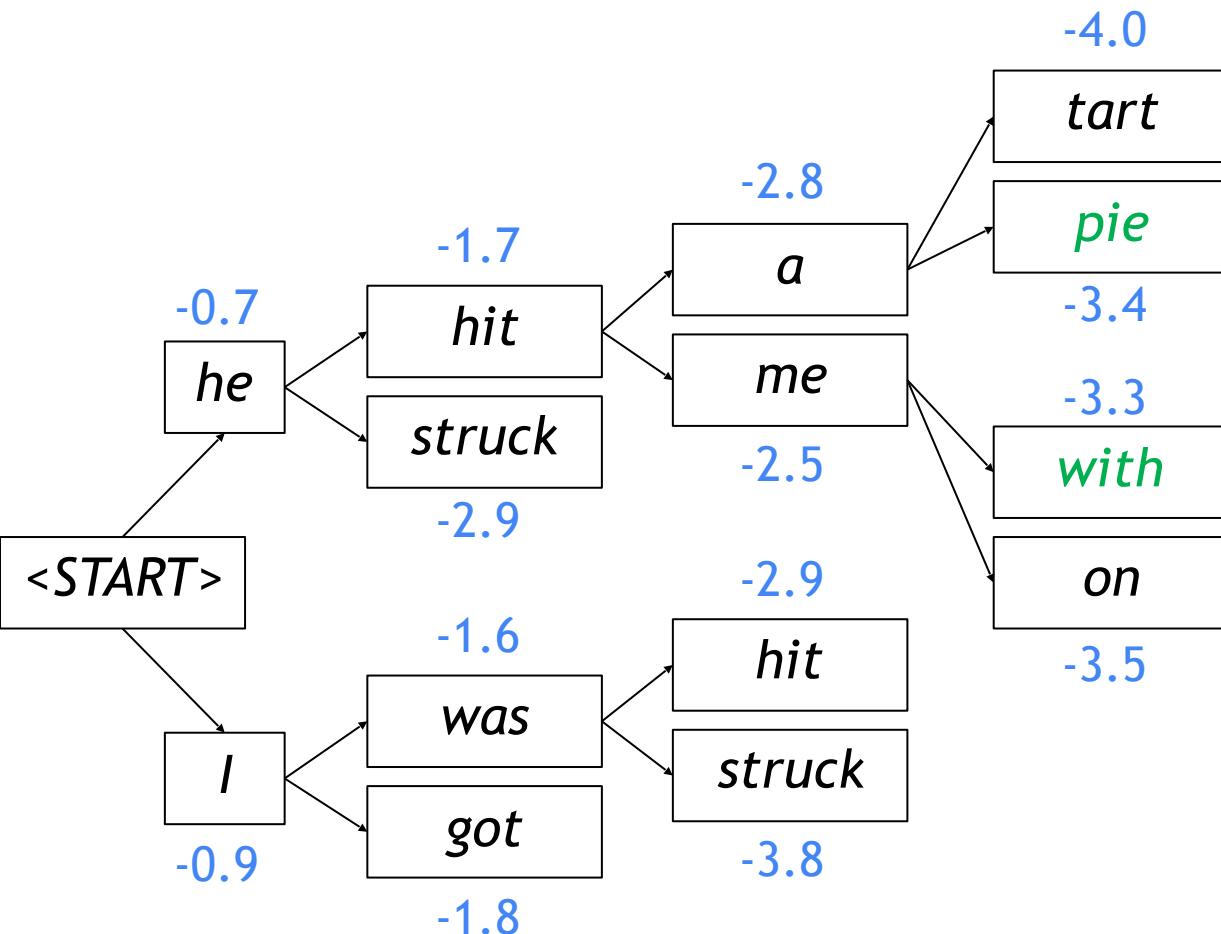
Beam size =  $k = 2$ . Blue numbers =  $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

# Beam search decoding: example

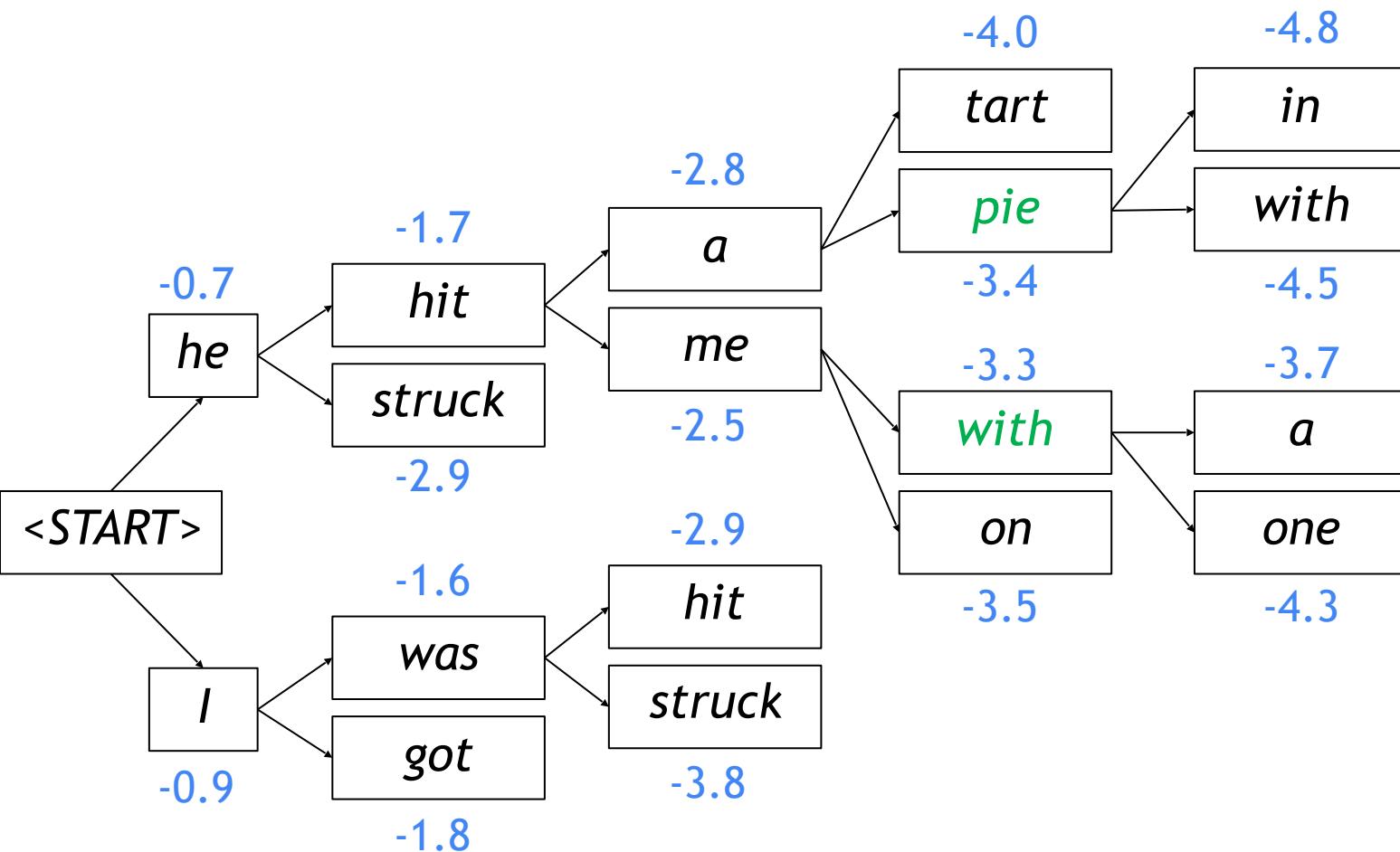
Beam size =  $k = 2$ . Blue numbers =  $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

# Beam search decoding: example

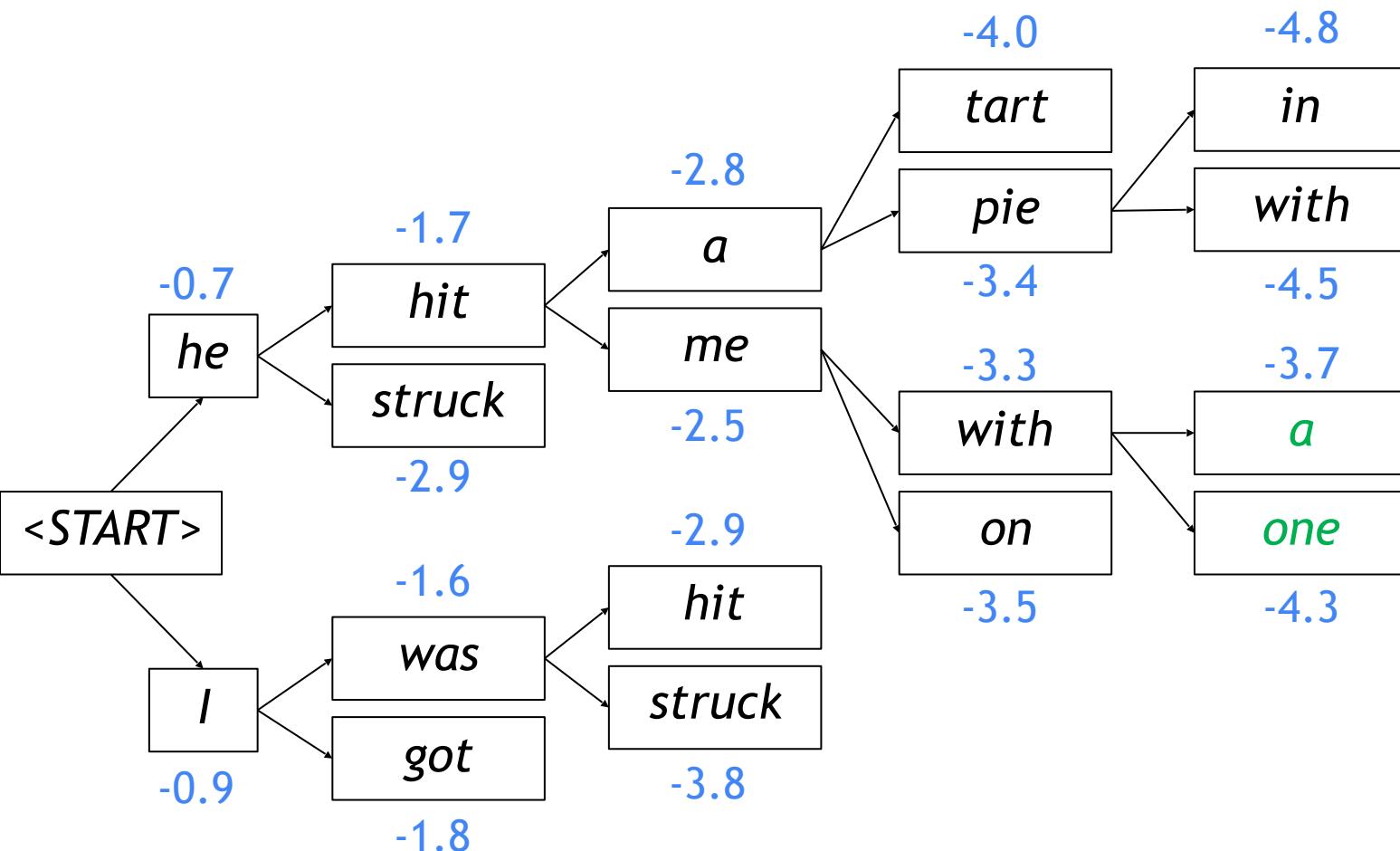
Beam size =  $k = 2$ . Blue numbers =  $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

# Beam search decoding: example

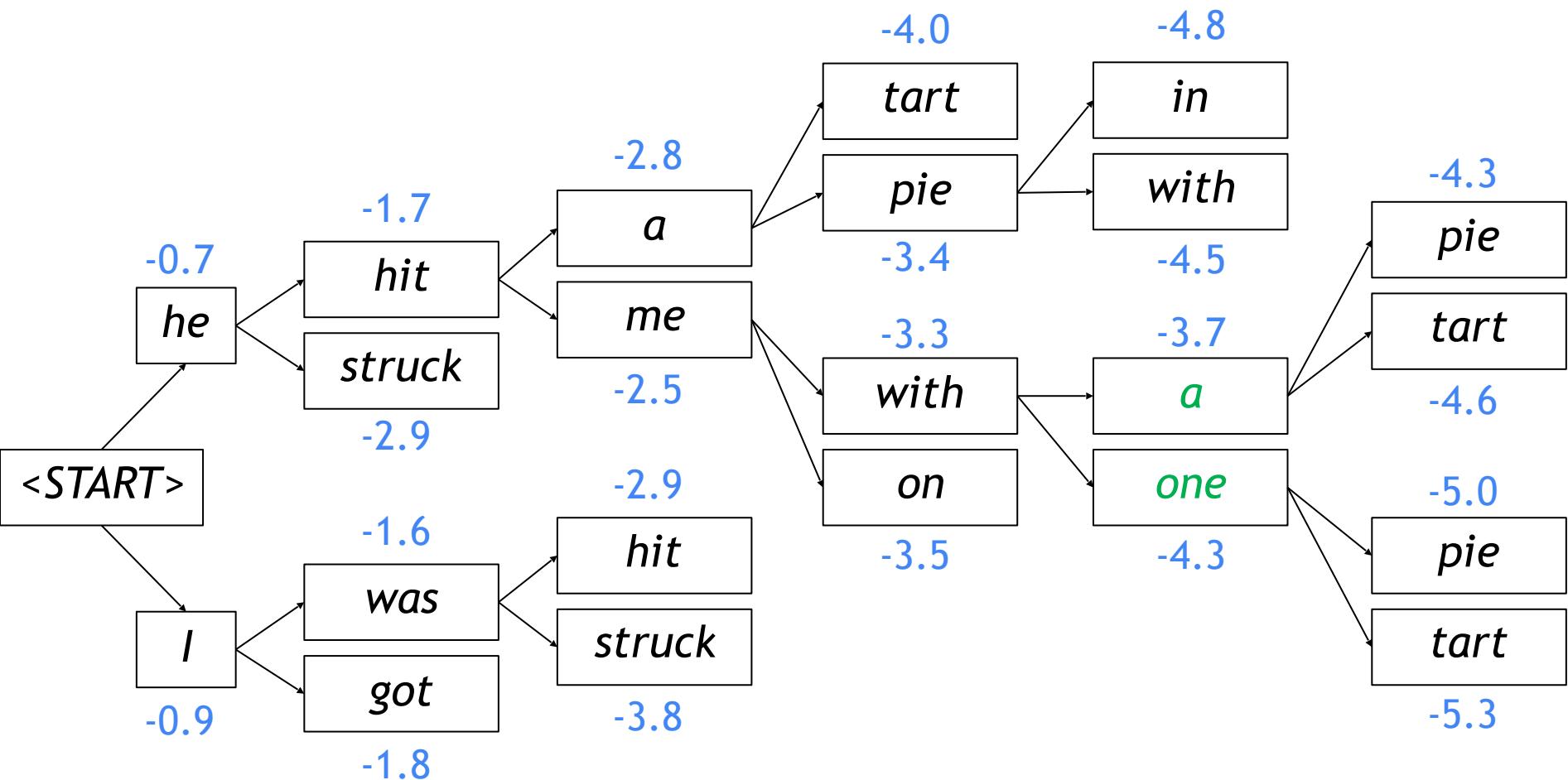
Beam size =  $k = 2$ . Blue numbers =  $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

# Beam search decoding: example

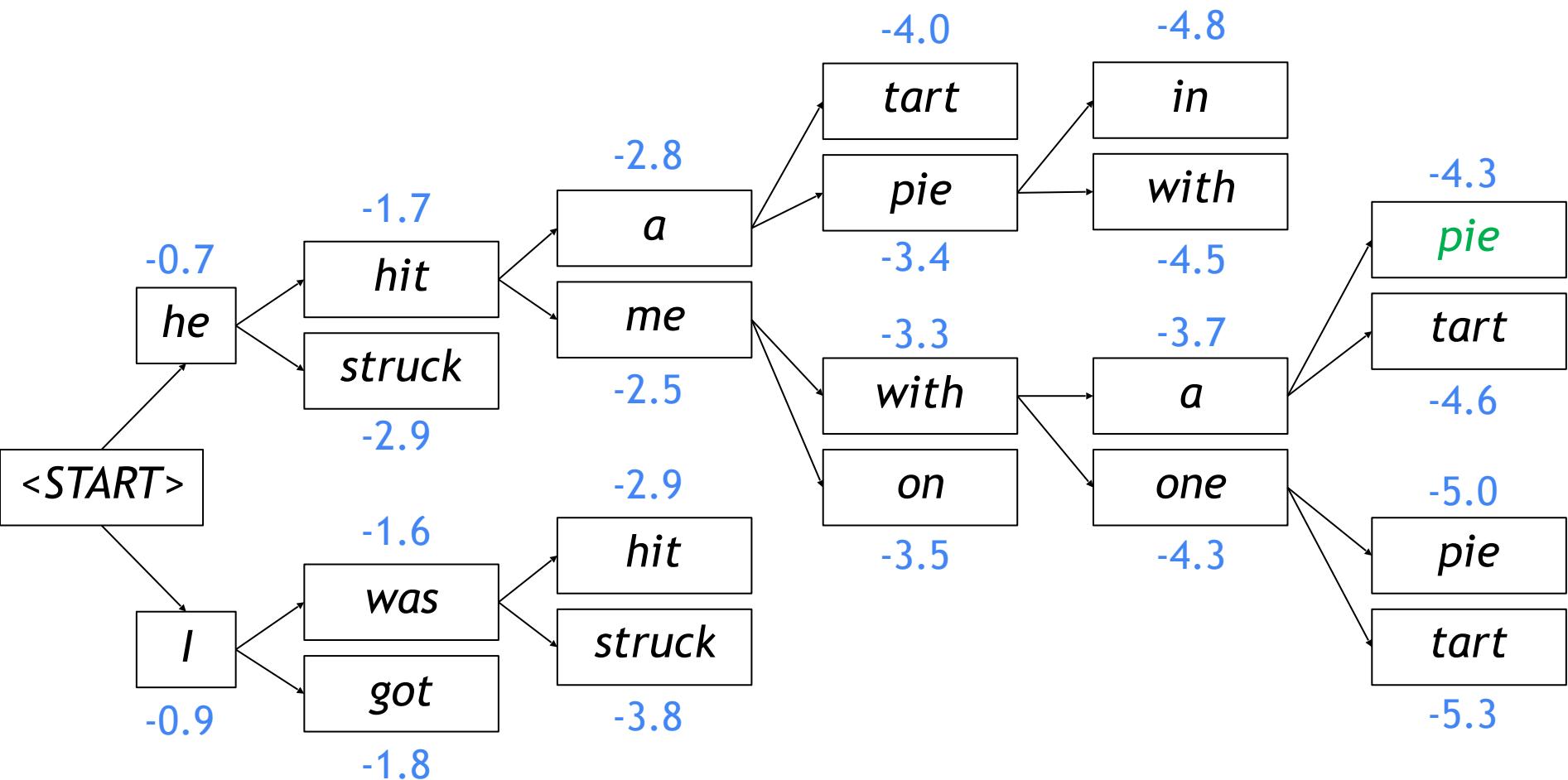
Beam size =  $k = 2$ . Blue numbers =  $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

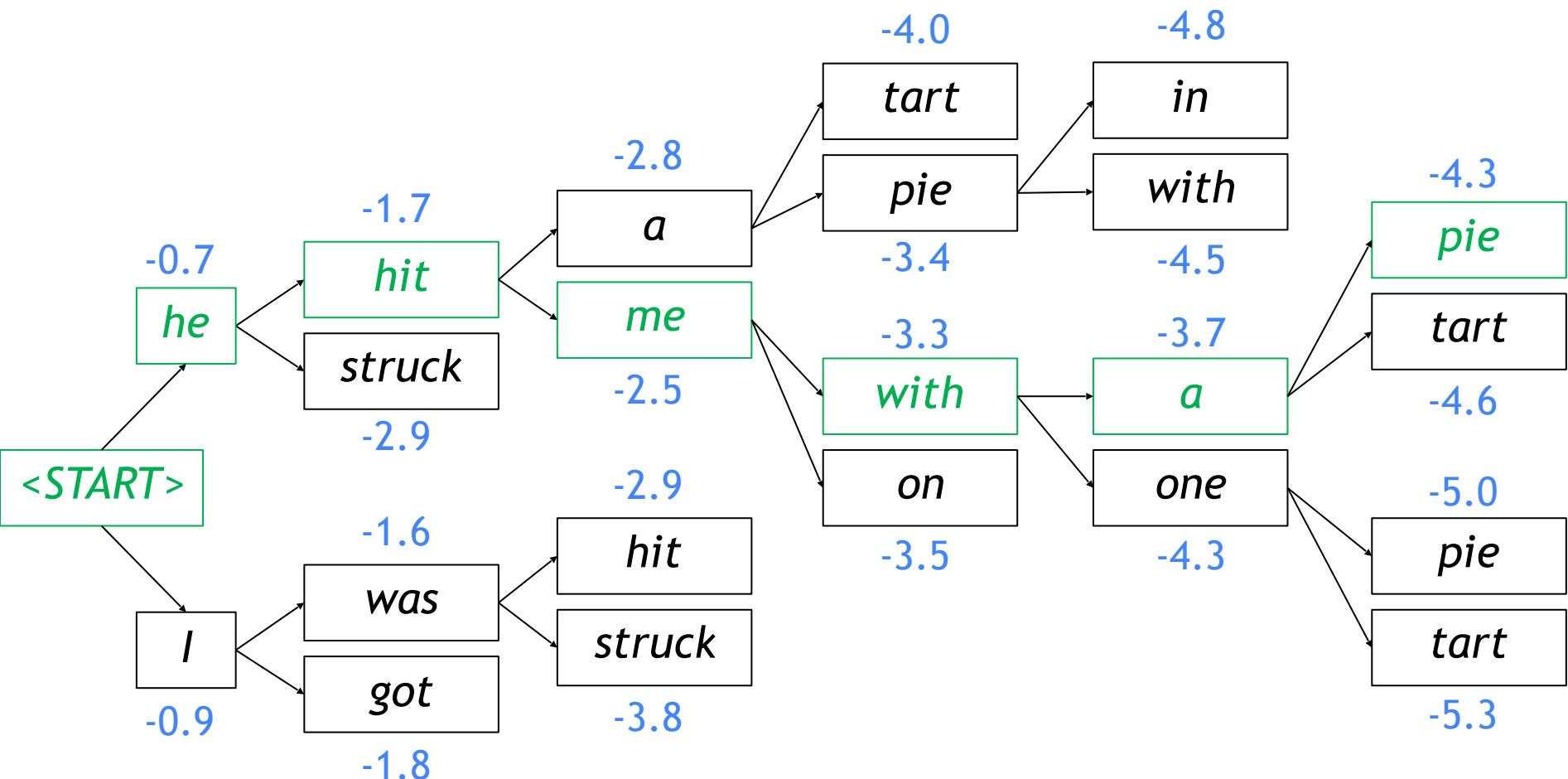
# Beam search decoding: example

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



# Beam search decoding: example

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



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  $y_1, \dots, y_t$  on our list has a score

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

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

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

# Advantages of NMT

Compared to SMT, NMT has many advantages:

- Better performance
  - More fluent
  - Better use of context
  - Better use of phrase similarities
- A single neural network to be optimized end-to-end
  - No subcomponents to be individually optimized
- Requires much less human engineering effort
  - No feature engineering
  - Same method for all language pairs

# Disadvantages of NMT?

Compared to SMT:

- NMT is **less interpretable**
  - Hard to debug
- NMT is **difficult to control**
  - For example, can't easily specify rules or guidelines for translation
  - Safety concerns!

# How do we evaluate Machine Translation?

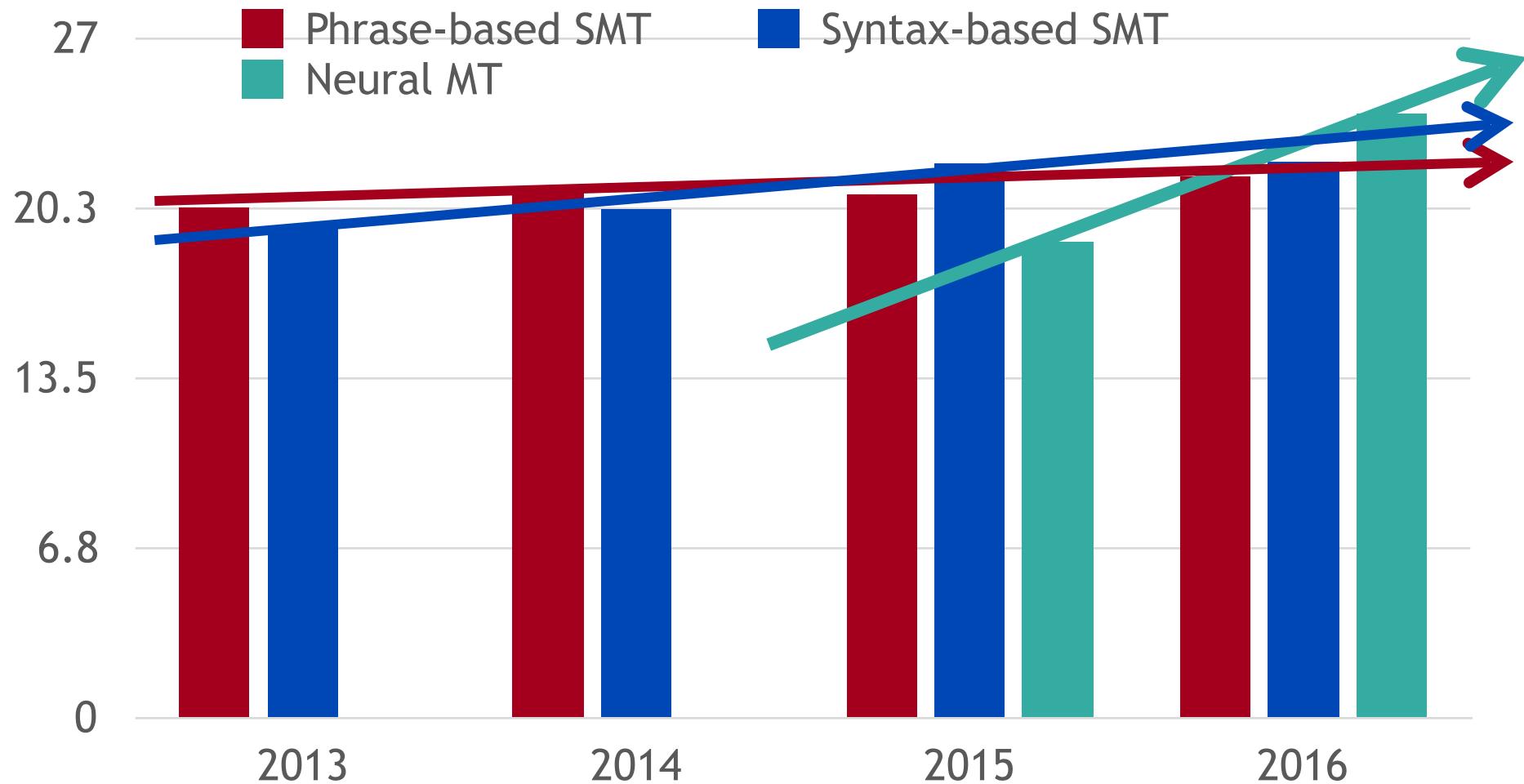
## BLEU (Bilingual Evaluation Understudy)

You'll see BLEU in detail in Assignment 4!

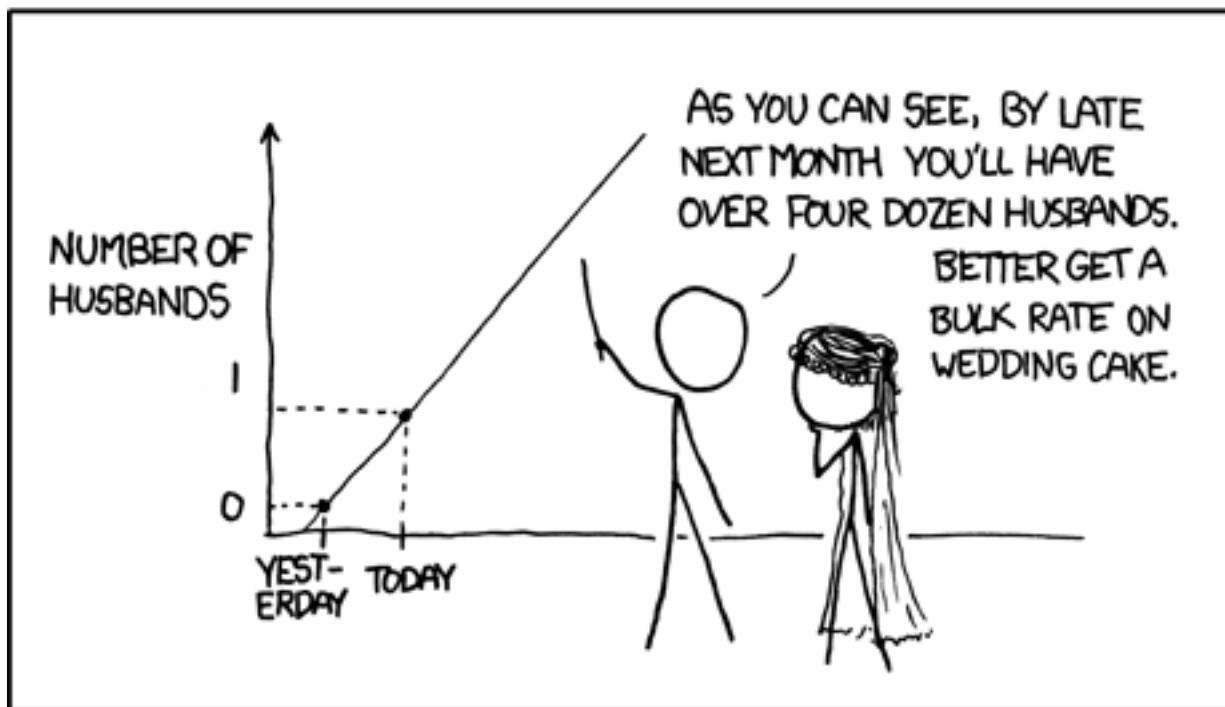
- BLEU compares the machine-written translation to one or several human-written translation(s), and computes a **similarity score** based on:
  - ***n*-gram precision** (usually for 1, 2, 3 and 4-grams)
  - Plus a penalty for too-short system translations
- BLEU is **useful** but **imperfect**
  - There are many valid ways to translate a sentence
  - So a **good** translation can get a **poor** BLEU score because it has low *n*-gram overlap with the human translation 😞

# MT progress over time

[Edinburgh En-De WMT newstest2013 Cased BLEU; NMT 2015 from U. Montréal]



# MT progress over time



# NMT: the biggest success story of NLP Deep Learning

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

- 2014: First seq2seq paper published
- 2016: Google Translate switches from SMT to NMT
- This is amazing!
  - SMT systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by a handful of engineers in a few months

# 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](https://www.skynettoday.com/editorials/state_of_nmt)

# So is Machine Translation solved?

- **Nope!**
- Using common sense is still hard
- Idioms are difficult to translate

This screenshot shows a machine translation interface. The source text is "Mi amigo no tiene pelos en la lengua" (My friend has no hair on the tongue) in Spanish. The target text is "My friend has no hair on the tongue" in English. The interface includes language selection dropdowns (Hindi, Spanish, English, etc.), a character count (36/5000), and various interaction icons.

This screenshot shows another machine translation interface. The source text is "जब श्याम को पता चला कि वो परीक्षा में विफल हो गया, तब उसका चहरा उतर गया." (When Shyam came to know that he failed the exam, his face went down.) in Hindi. The target text is "When Shyam came to know that he failed the exam, his face went down." in English. The interface includes language selection dropdowns (Hindi, English, Spanish, French, etc.), a character count (72/5000), and various interaction icons.

# So is Machine Translation solved?

- **Nope!**
- NMT picks up **biases** in training data

Malay - detected ▾

English ▾

Dia bekerja sebagai jururawat.

Dia bekerja sebagai pengaturcara. Edit

She works as a nurse.

He works as a programmer.

Didn't specify gender

# So is Machine Translation solved?

- Nope!
  - Uninterpretable systems do strange things

Somali	↔	English
Translate from Irish		
ag		As the name of the LORD was written
ag		in the Hebrew language, it was written
ag Edit		in the language of the Hebrew Nation

**Picture source:** [https://www.vice.com/en\\_uk/article/j5npeg/why-is-google-translate-spitting-out-sinister-religious-prophecies](https://www.vice.com/en_uk/article/j5npeg/why-is-google-translate-spitting-out-sinister-religious-prophecies)

**Explanation:** <https://www.skynettoday.com/briefs/google-nmt-prophecies>

# NMT research continues

NMT is the flagship task for NLP Deep Learning

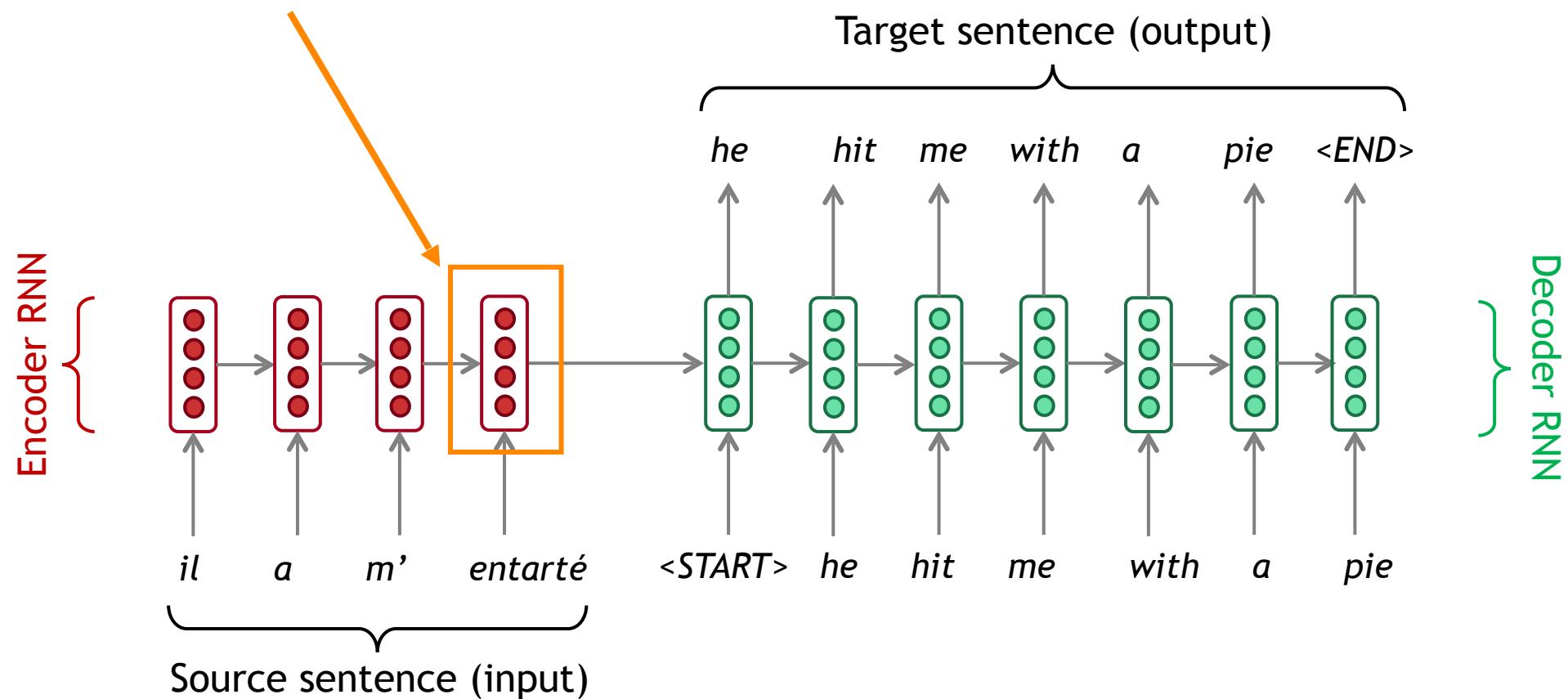
- NMT research has **pioneered** many of the recent **innovations** of NLP Deep Learning
- In **2019**: NMT research continues to **thrive**
  - Researchers have found **many, many improvements** to the “vanilla” seq2seq NMT system we’ve presented today
  - But **one improvement** is so integral that it is the new **vanilla...**

# ATTENTION

## Section 3: Attention

# Sequence-to-sequence: the bottleneck problem

Encoding of the source sentence.



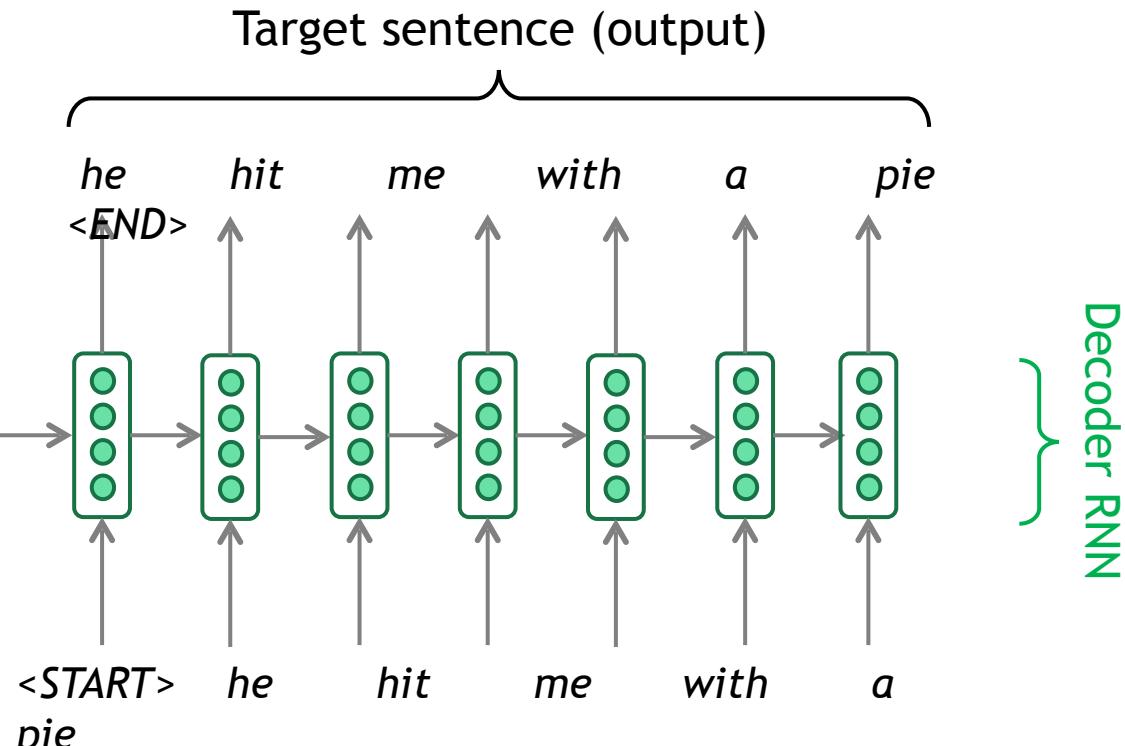
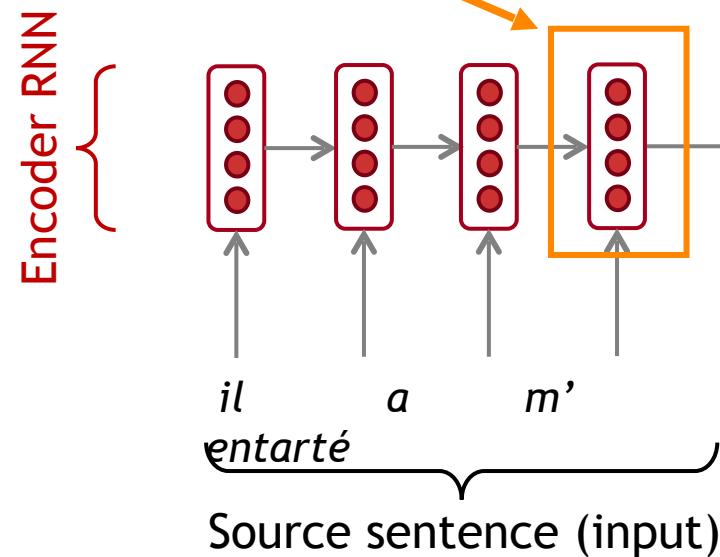
Problems with this architecture?

# Sequence-to-sequence: the bottleneck problem

Encoding of the source sentence.

This needs to capture *all information* about the source sentence.

Information bottleneck!



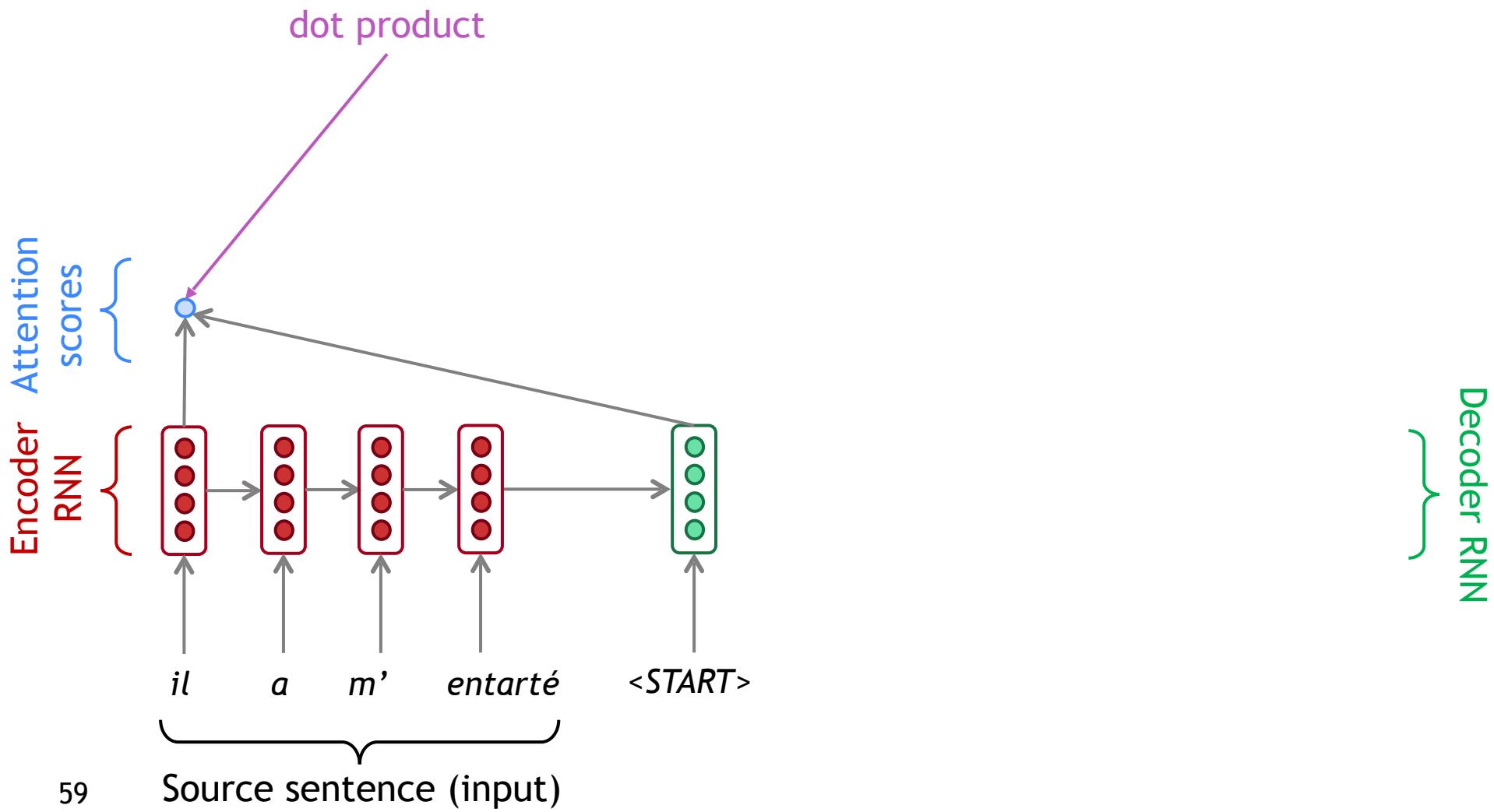
# Attention

- **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

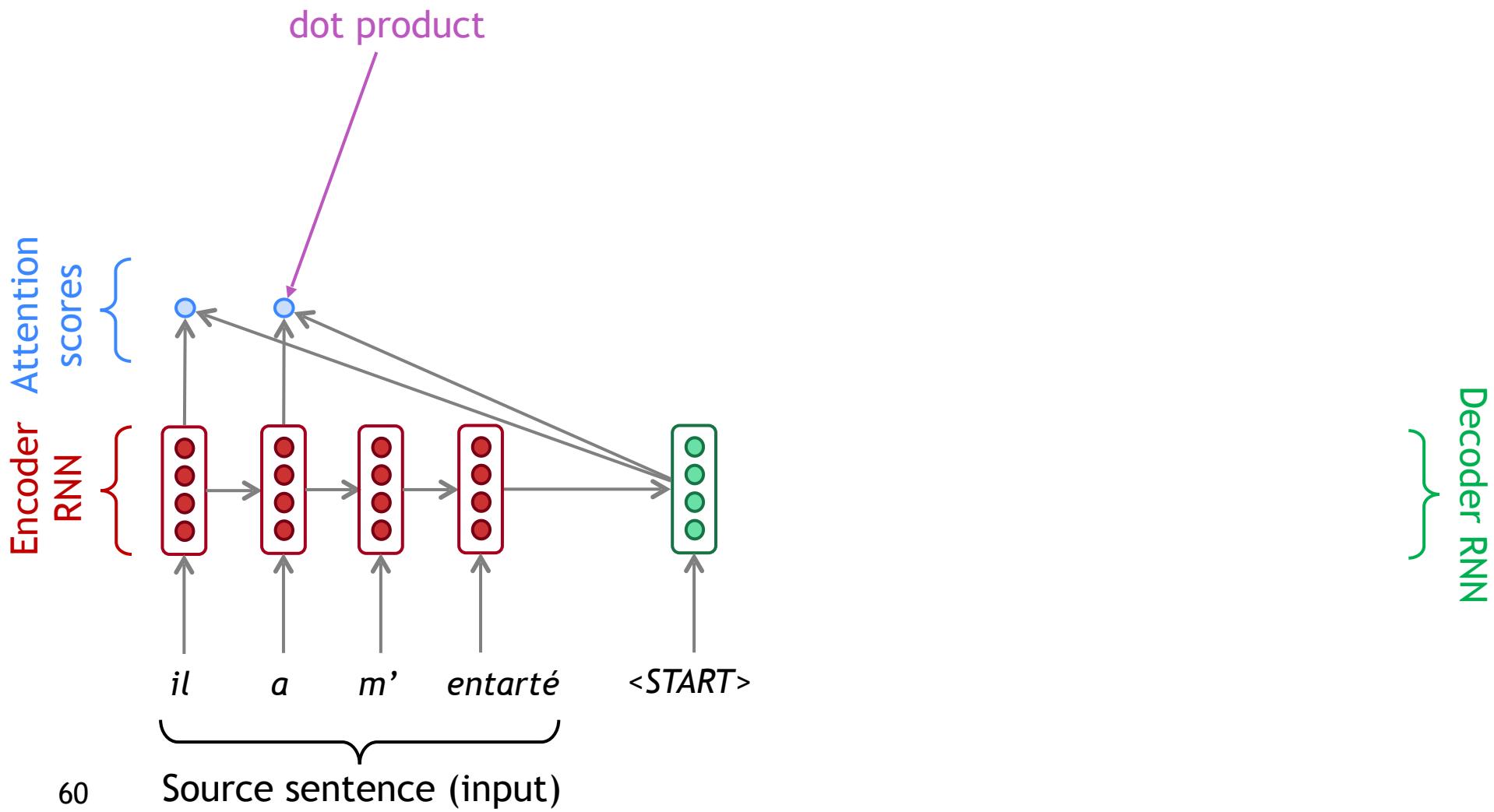


- First we will show via diagram (no equations), then we will show with equations

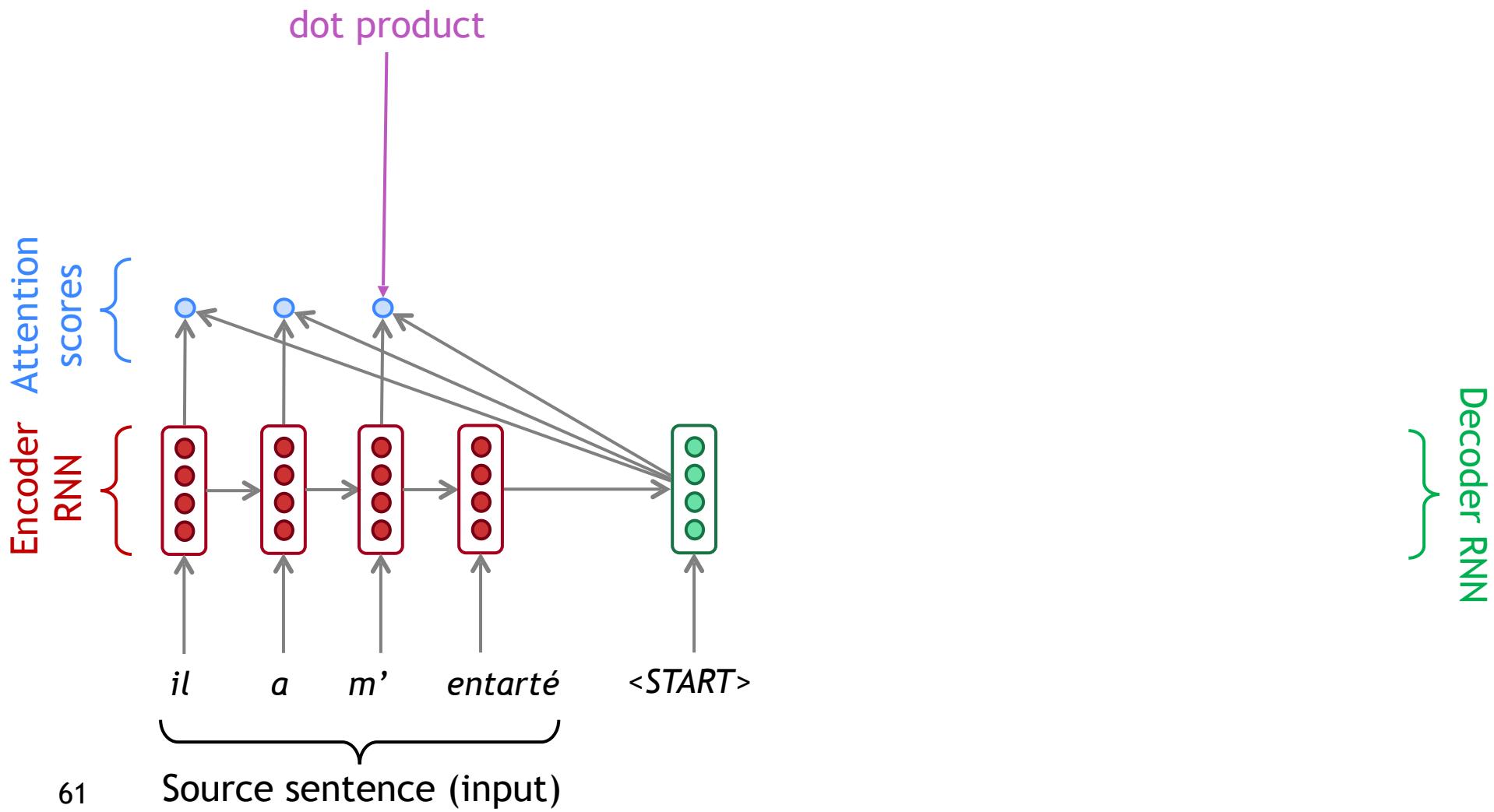
# Sequence-to-sequence with attention



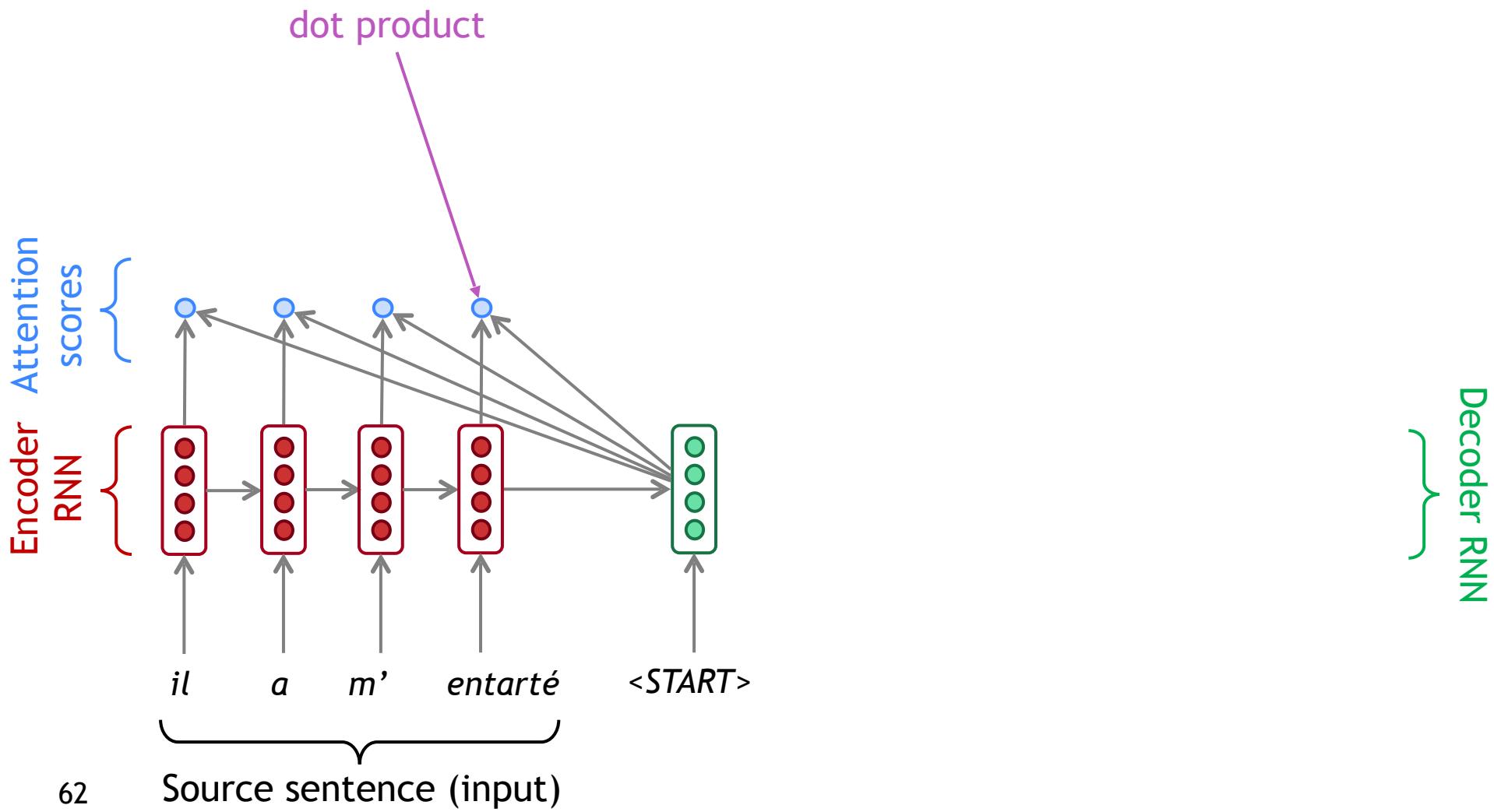
# Sequence-to-sequence with attention



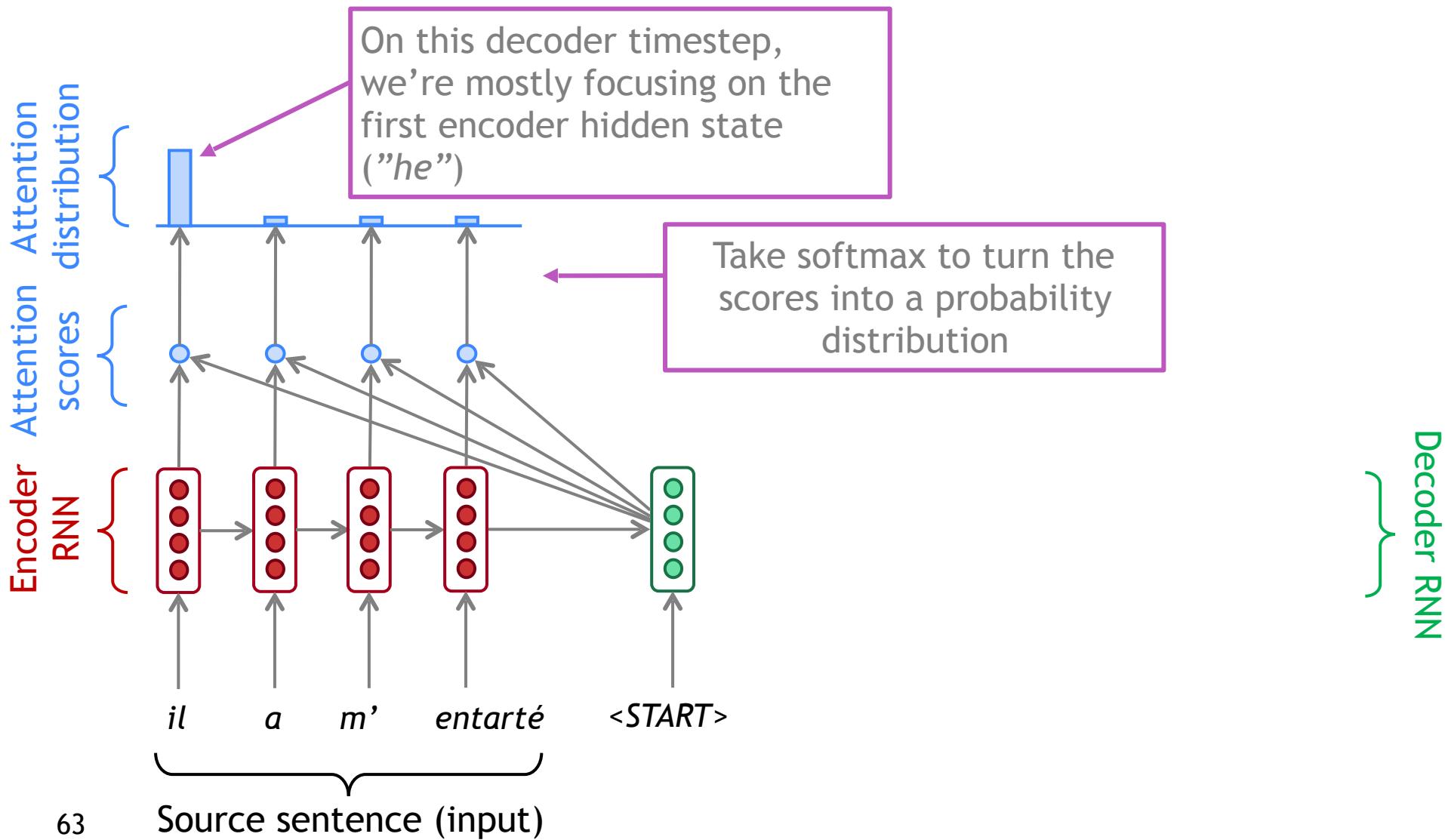
# Sequence-to-sequence with attention



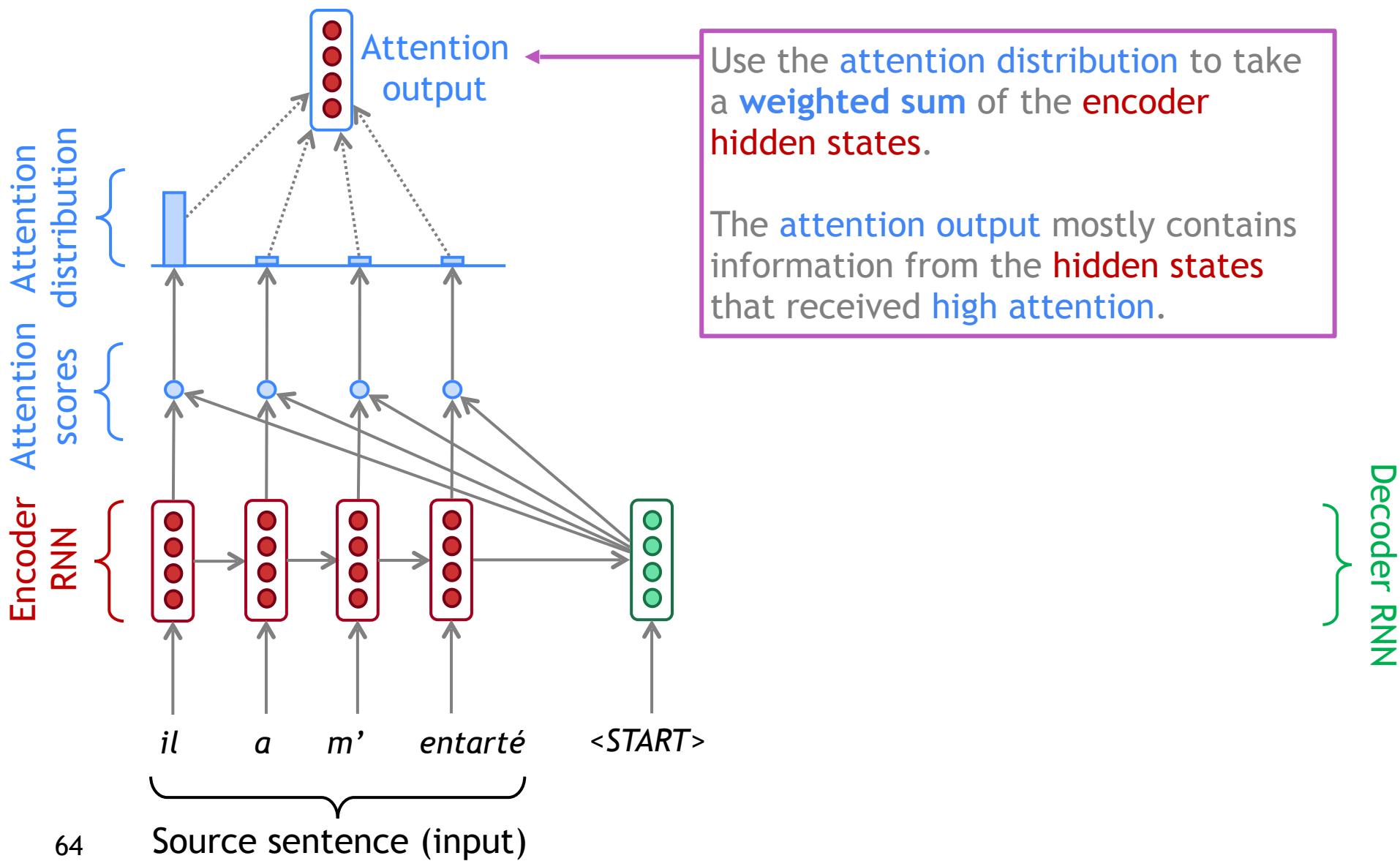
# Sequence-to-sequence with attention



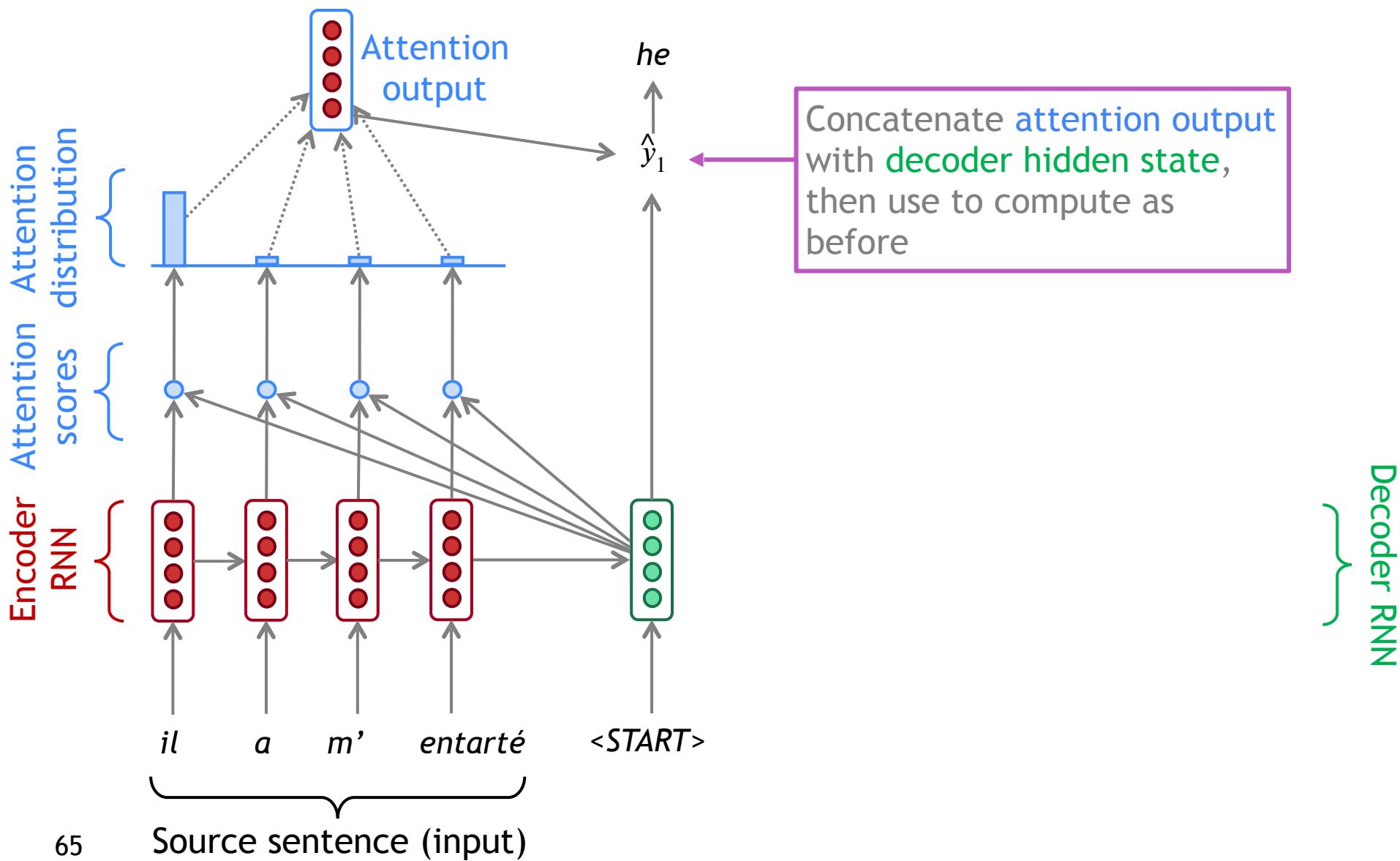
# Sequence-to-sequence with attention



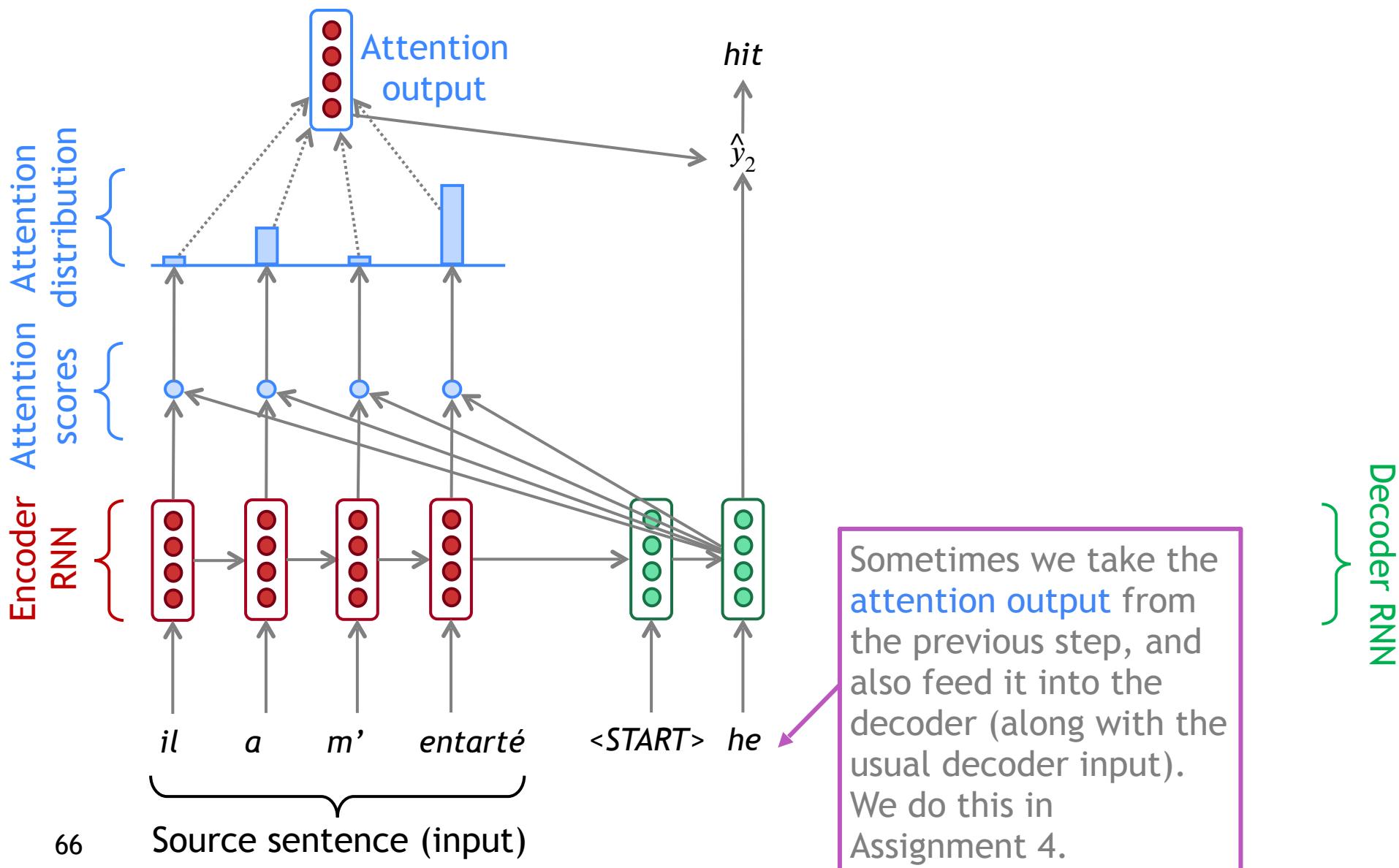
# Sequence-to-sequence with attention



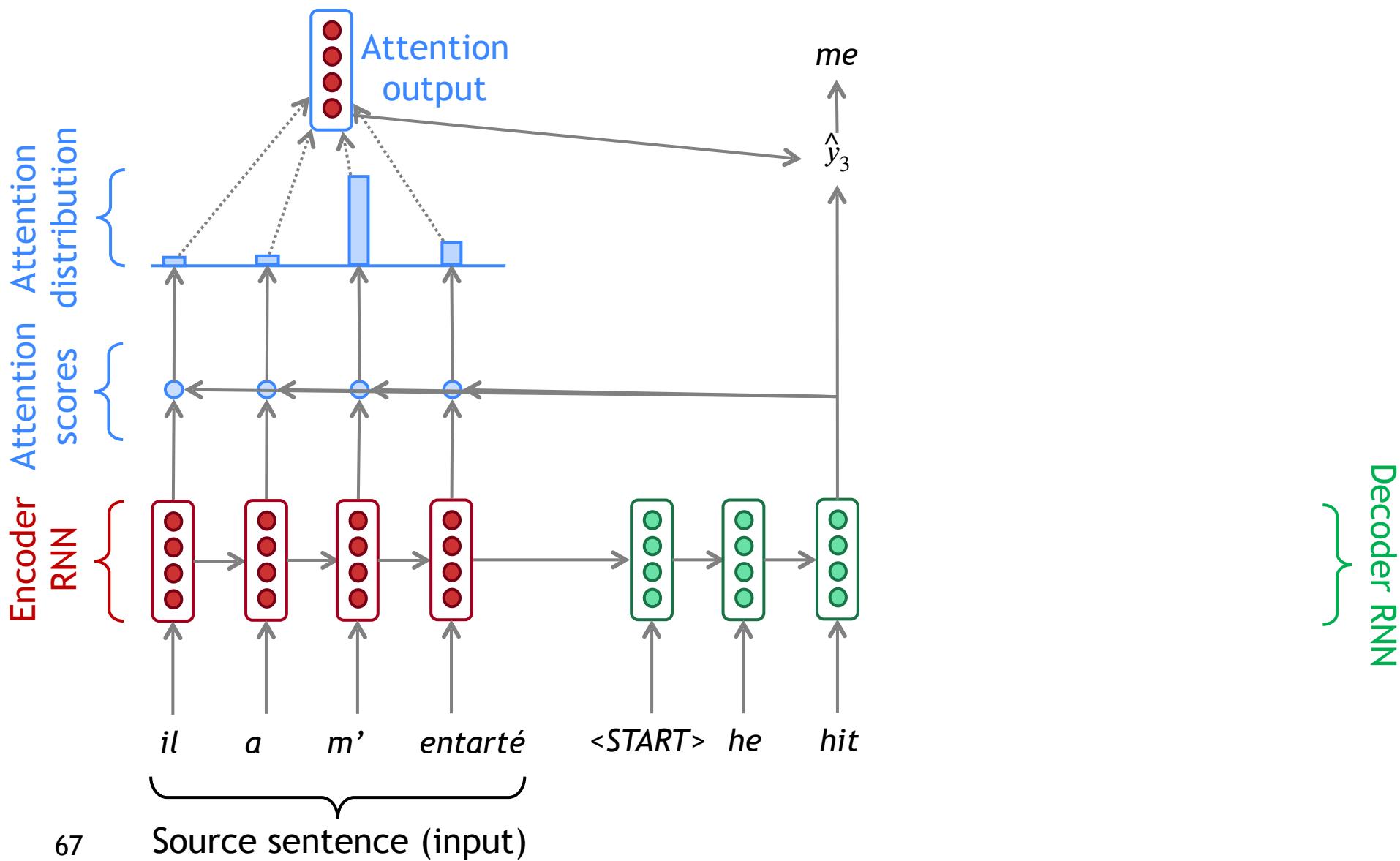
# Sequence-to-sequence with attention



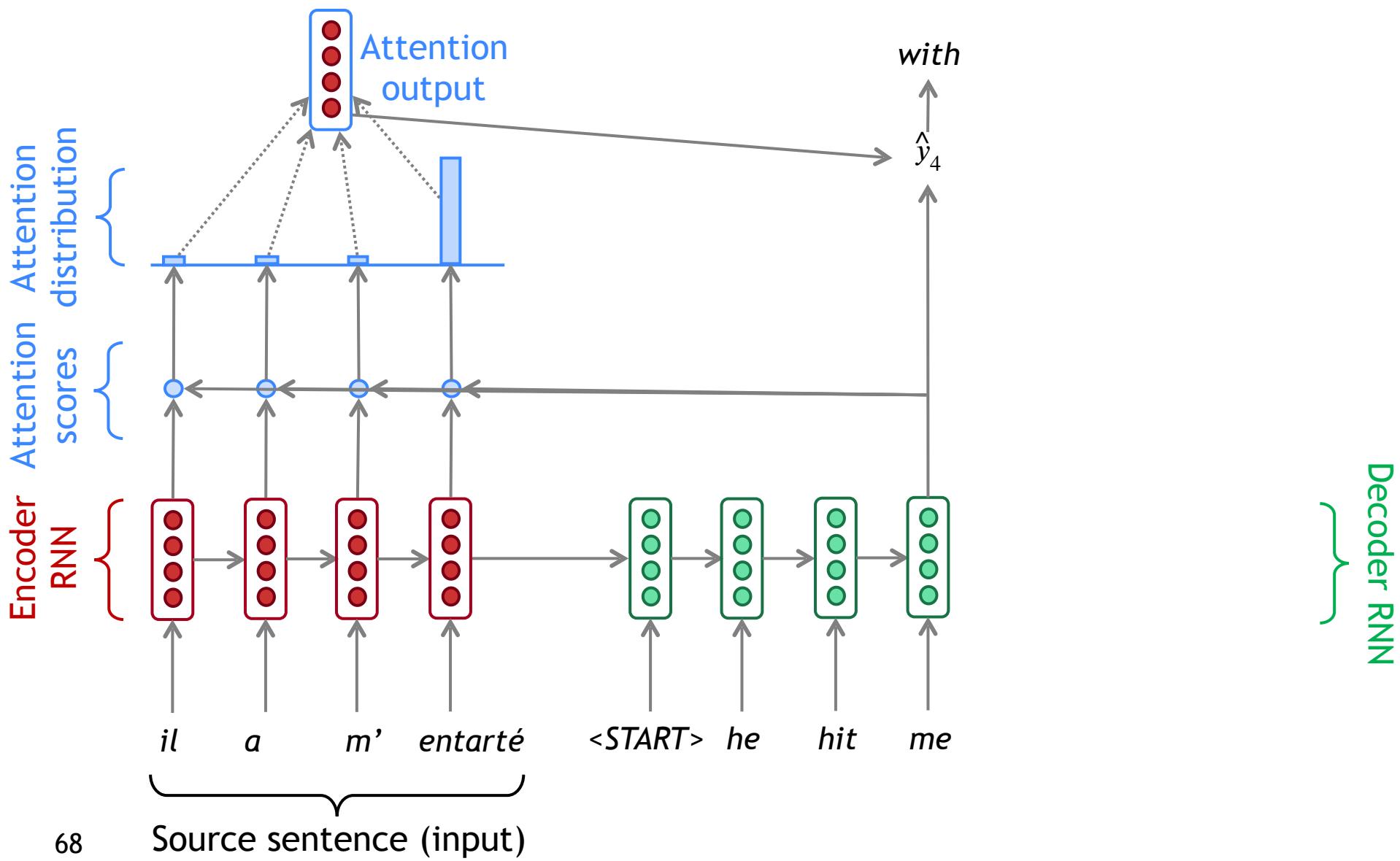
# Sequence-to-sequence with attention



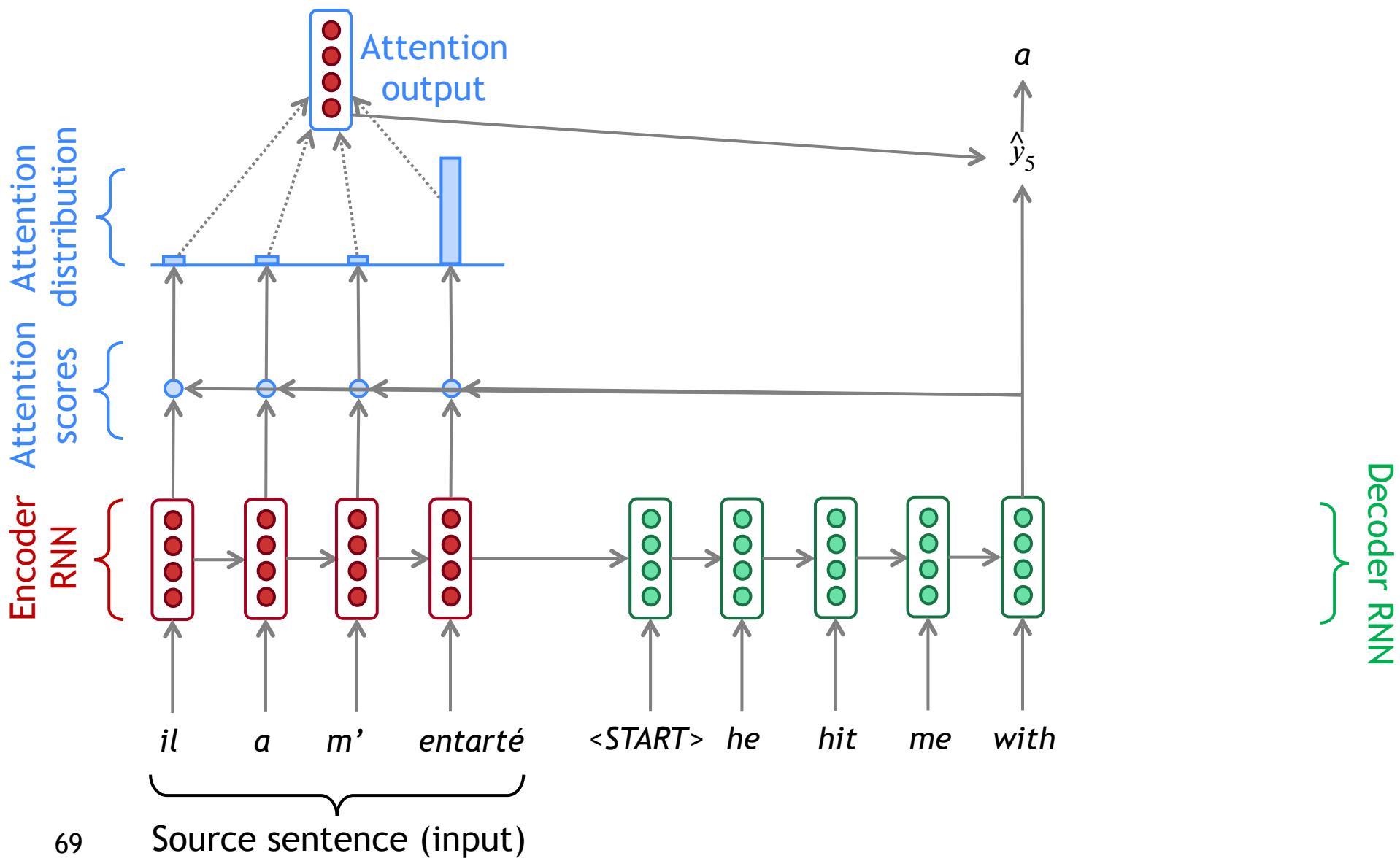
# Sequence-to-sequence with attention



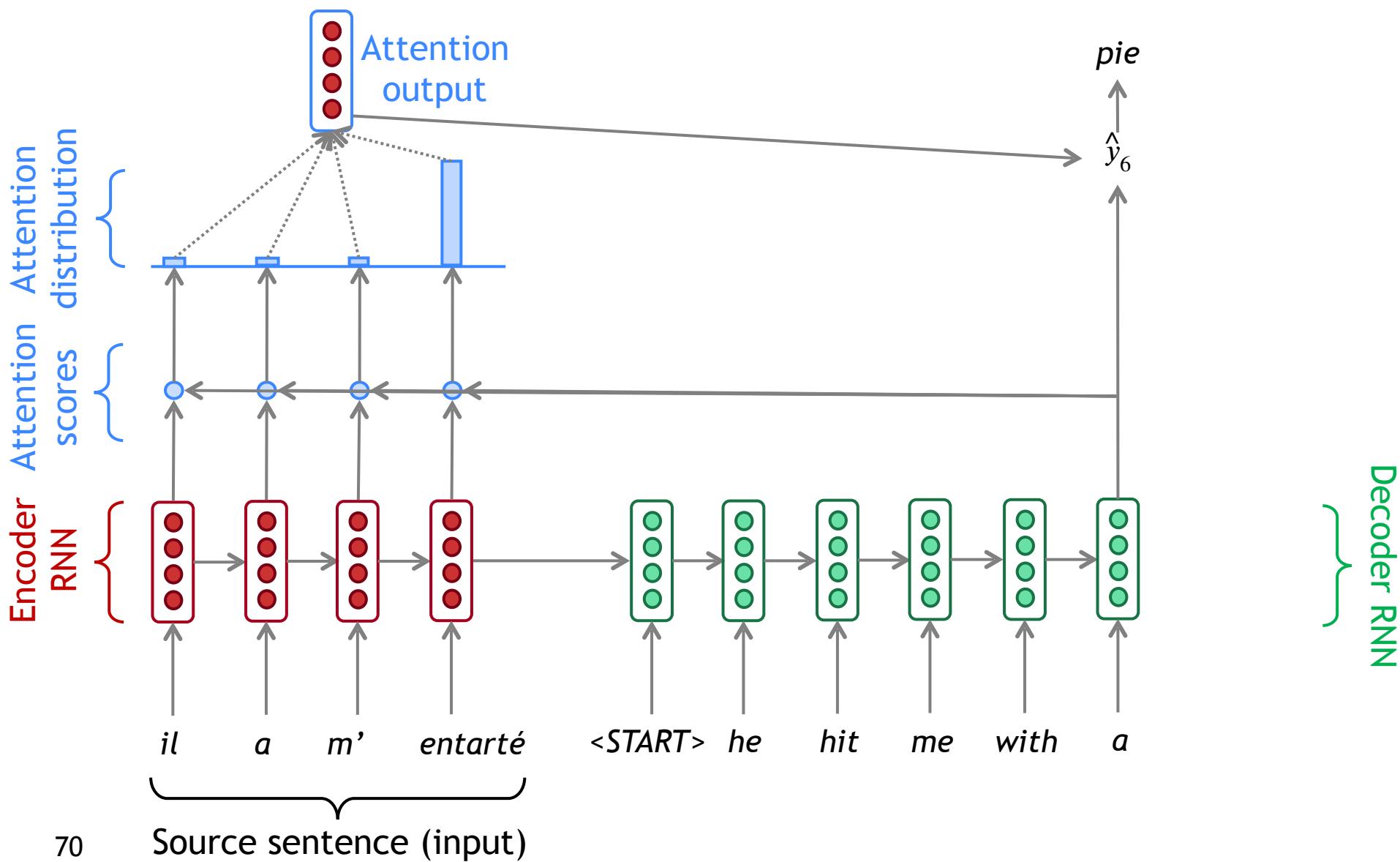
# Sequence-to-sequence with attention



# Sequence-to-sequence with attention



# Sequence-to-sequence with attention



# Attention: in equations

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

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

- We take softmax to get the attention distribution  $\alpha^t$  for this step (this is a probability distribution and sums to 1)

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

- We use  $\alpha^t$  to take a weighted sum of the encoder hidden states to get the attention output

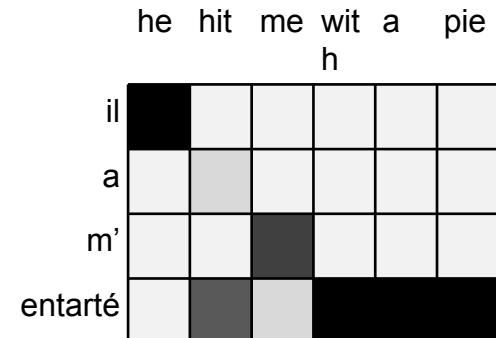
$$a_t = \sum_{i=1}^N \alpha_i^t h_i \in \mathbb{R}^h$$

- 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



# 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).

# Attention is a *general* Deep Learning technique

## 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.

## 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).

# There are *several* attention variants

- We have some *values*  $\mathbf{h}_1, \dots, \mathbf{h}_N \in \mathbb{R}^{d_1}$  and a  $\mathbf{s} \in \mathbb{R}^{d_2}$
- Attention always involves:
  1. Computing the *attention scores*  $\mathbf{e} \in \mathbb{R}^N$
  2. Taking softmax to get *attention distribution*  $\alpha$ :

There are multiple ways to do this

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

3. Using attention distribution to take weighted sum of values:

$$\mathbf{a} = \sum_{i=1}^N \alpha_i \mathbf{h}_i \in \mathbb{R}^{d_1}$$

thus obtaining the *attention output*  $\mathbf{a}$  (sometimes called the *context vector*)

# Attention variants

You'll think about the relative advantages/disadvantages of these in Assignment 4!

There are **several ways** you can compute  $e \in \mathbb{R}^N$        $\mathbf{h}_1, \dots, \mathbf{h}_N \in \mathbb{R}^{d_1}$  and  $s \in \mathbb{R}^{d_2}$  :

- Basic dot-product attention:  $e_i = s^T \mathbf{h}_i \in \mathbb{R}$ 
  - Note: this assumes  $d_1 = d_2$
  - This is the version we saw earlier
- Multiplicative attention:  $e_i = s^T \mathbf{W} \mathbf{h}_i \in \mathbb{R}$ 
  - Where  $\mathbf{W} \in \mathbb{R}^{d_2 \times d_1}$  is a weight matrix
- Additive attention:  $e_i = v^T \tanh(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 s) \in \mathbb{R}$ 
  - Where  $\mathbf{W}_1 \in \mathbb{R}^{d_3 \times d_1}, \mathbf{W}_2 \in \mathbb{R}^{d_3 \times d_2}$  are weight matrices  $v \in \mathbb{R}^{d_3}$  is a weight vector.
  - $d_3$  (the attention dimensionality) is a hyperparameter

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>

# Summary of today's lecture

- We learned some history of Machine Translation (MT)
- Since 2014, Neural MT rapidly replaced intricate Statistical MT
- Sequence-to-sequence is the architecture for NMT (uses 2 RNNs)
- Attention is a way to *focus on particular parts* of the input
  - Improves sequence-to-sequence a lot

