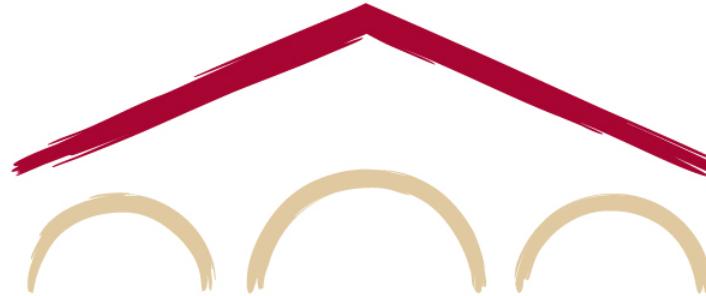


Natural Language Processing with Deep Learning

CS224N/Ling284



Christopher Manning

Lecture 7: Machine Translation, Sequence-to-Sequence and Attention

Lecture Plan

Today we will:

1. Introduce a **new task**: Machine Translation [15 mins], which is a major use-case of
 2. A **new neural architecture**: sequence-to-sequence [45 mins], which is improved by
 3. A **new neural technique**: attention [20 mins]
-
- Announcements
 - Assignment 3 is due today – I hope your dependency parsers are parsing text!
 - Assignment 4 out today – covered in this lecture, you get 9 days for it (!), due Thu
 - Get started early! It's bigger and harder than the previous assignments 😓
 - Thursday's lecture about choosing final projects

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

The early history of MT: 1950s

- Machine translation research more powerful than high school
- Foundational work on information theory
- MT heavily funded by systems doing word substitution
- Human language is much more complex than languages!
- Little understanding of grammar
- Problem soon appeared

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



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

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

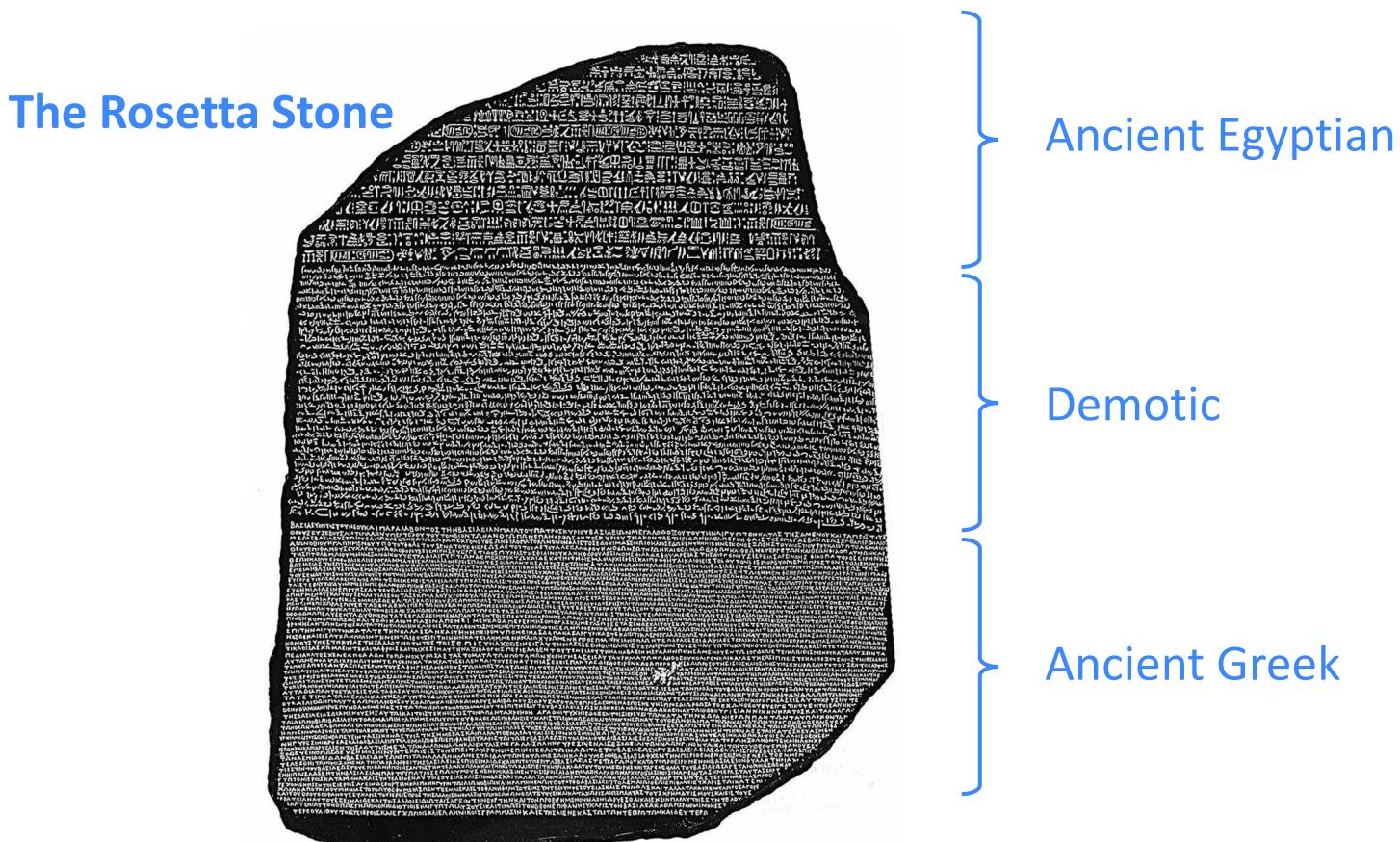
- Use Bayes Rule to break this down into **two components** to be learned separately:

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



1990s-2010s: Statistical Machine Translation

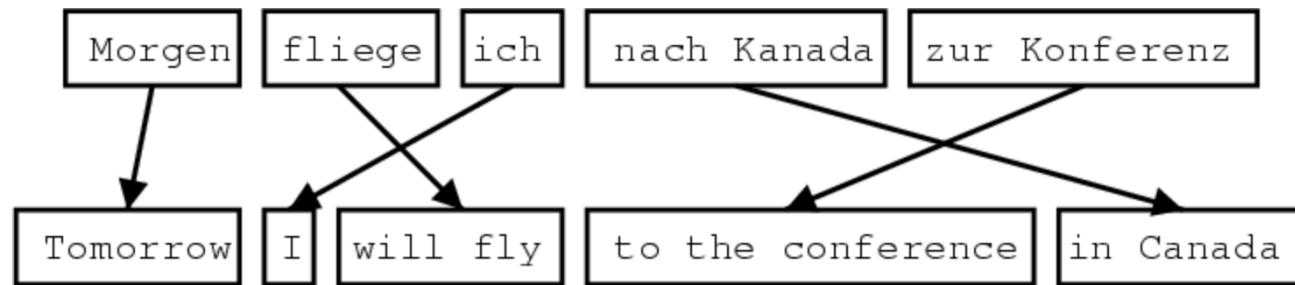
- Question: How to learn translation model $P(x|y)$?
- First, need large amount of **parallel data**
(e.g., pairs of human-translated French/English sentences)



Learning alignment for SMT

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

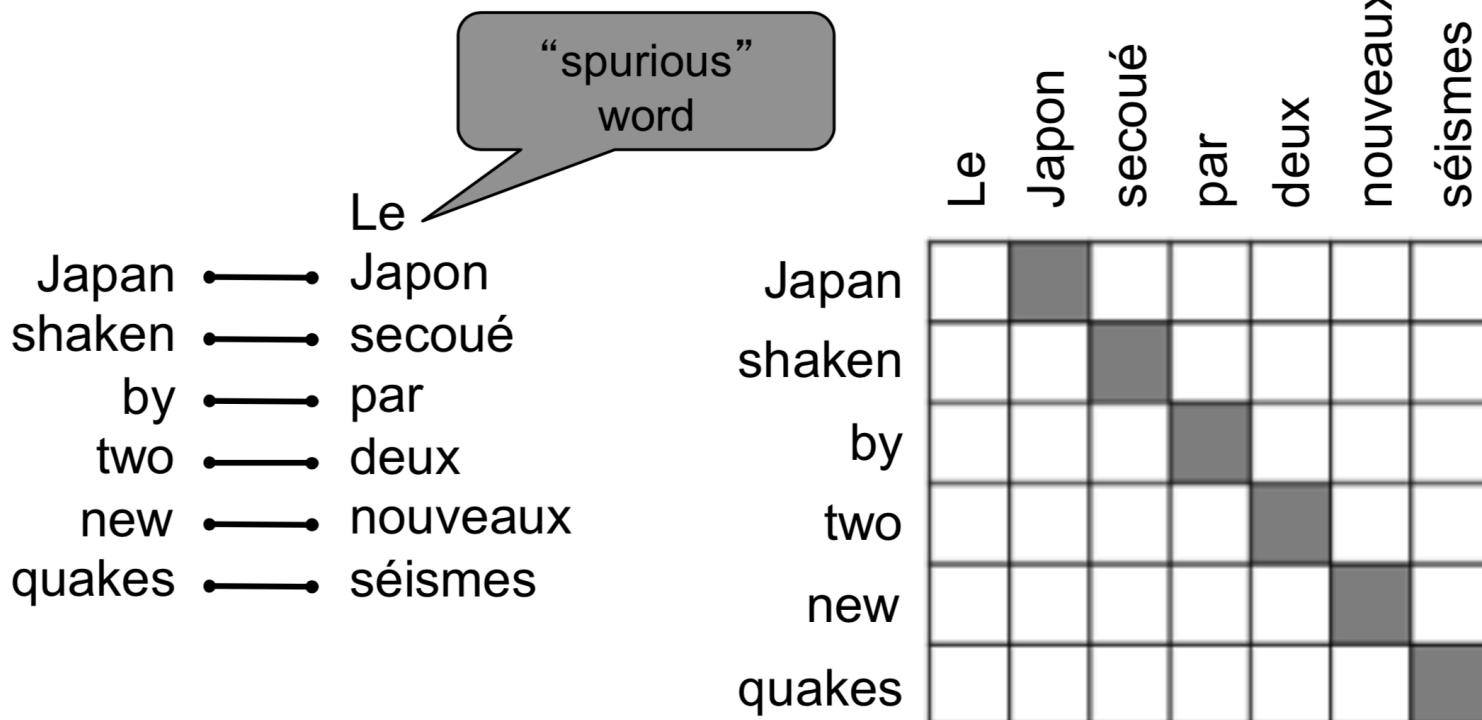
where a is the **alignment**, i.e. word-level correspondence between source sentence x and target sentence y



What is alignment?

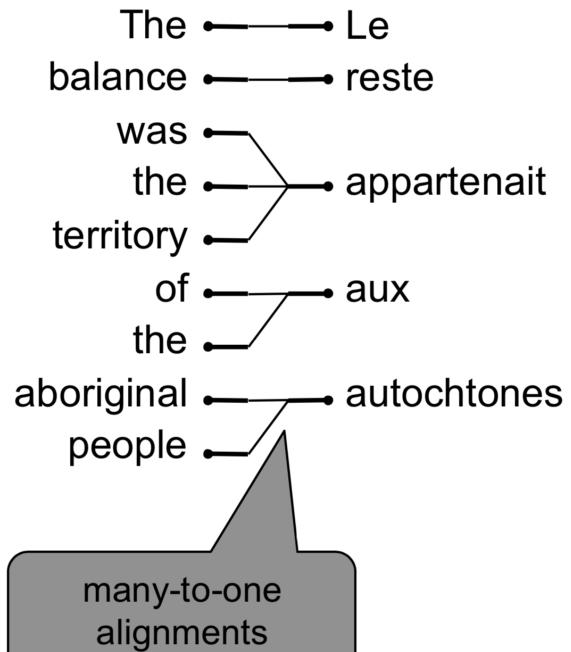
Alignment is the **correspondence between particular words** in the translated sentence pair.

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



Alignment is complex

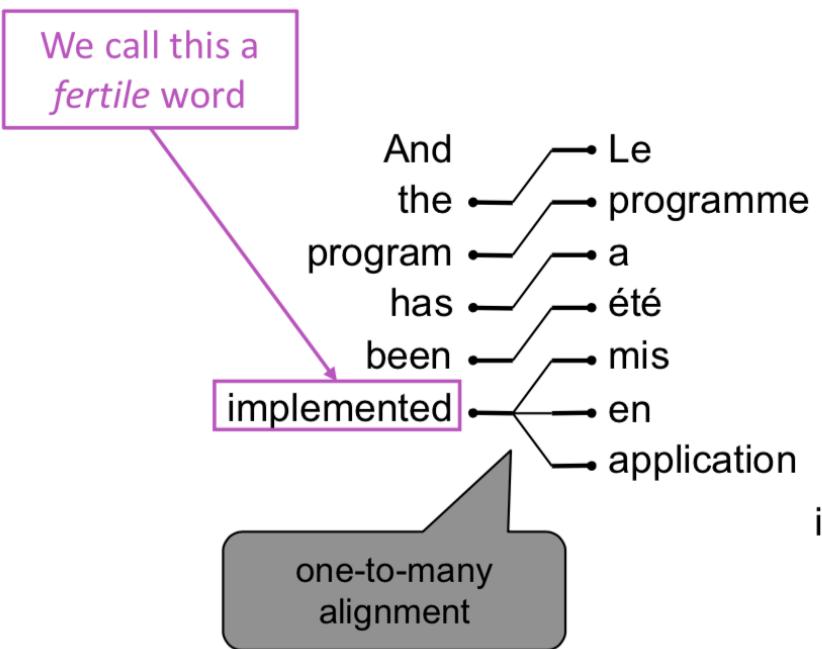
Alignment can be many-to-one



| | Le | reste | appartenait | aux | autochtones |
|------------|----|-------|-------------|-----|-------------|
| The | | | | | |
| balance | | | | | |
| was | | | | | |
| the | | | | | |
| territory | | | | | |
| of | | | | | |
| the | | | | | |
| aboriginal | | | | | |
| people | | | | | |

Alignment is complex

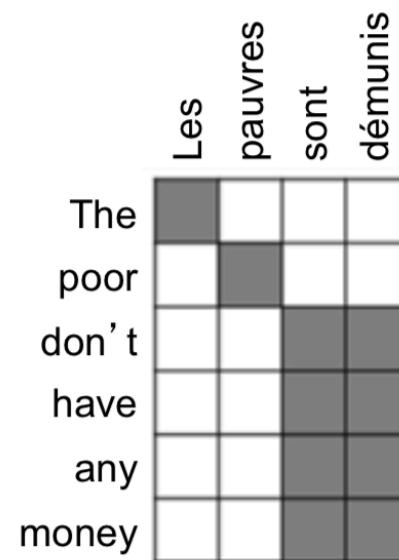
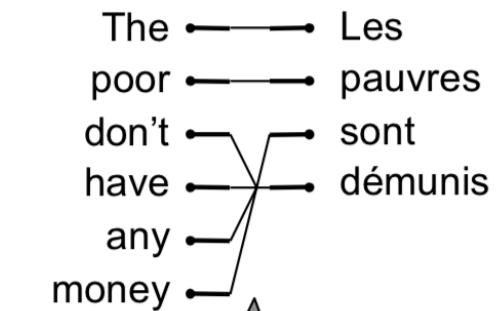
Alignment can be **one-to-many**



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

Alignment is complex

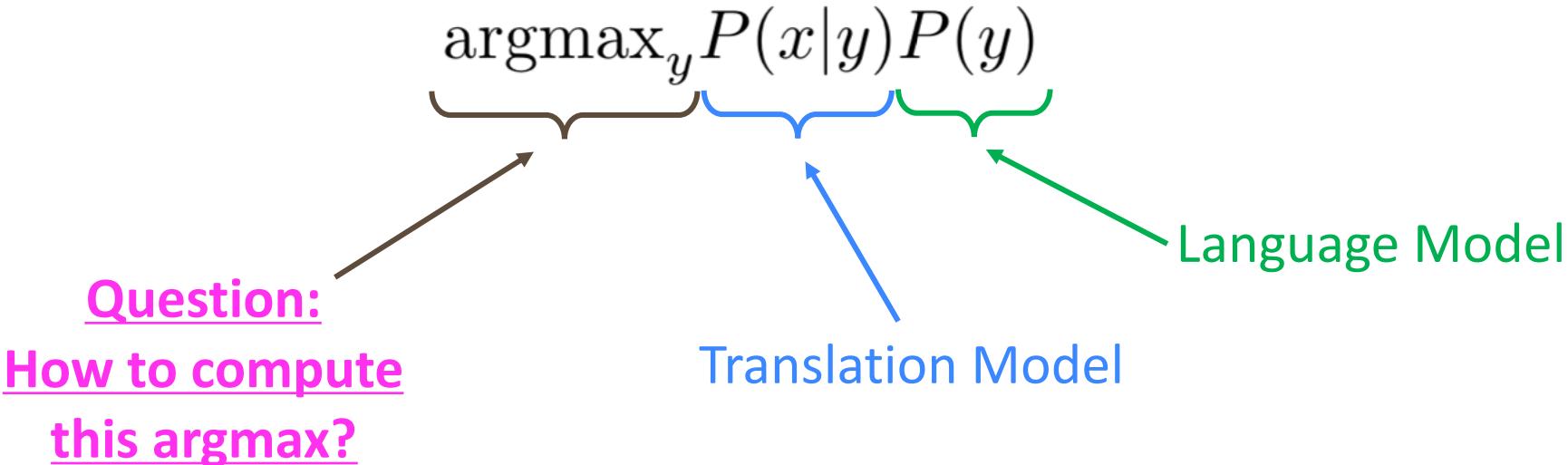
Alignment can be many-to-many (phrase-level)



Learning alignment for SMT

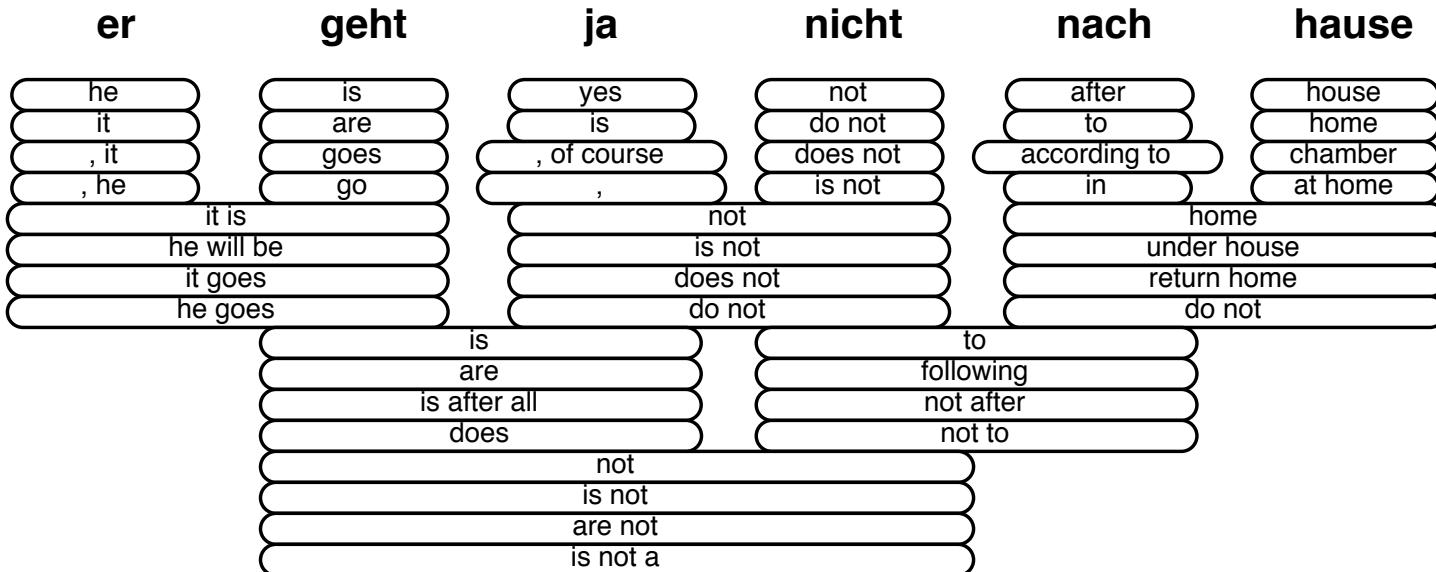
- We learn $P(x, a|y)$ as a combination of many factors, including:
 - Probability of particular words aligning (also depends on position in sent)
 - Probability of particular words having a particular fertility (number of corresponding words)
 - etc.
- Alignments a are **latent variables**: They aren't explicitly specified in the data!
 - Require the use of special learning algorithms (like Expectation-Maximization) for learning the parameters of distributions with latent variables
 - In older days, we used to do a lot of that in CS 224N, but now see CS 228!

Decoding for SMT

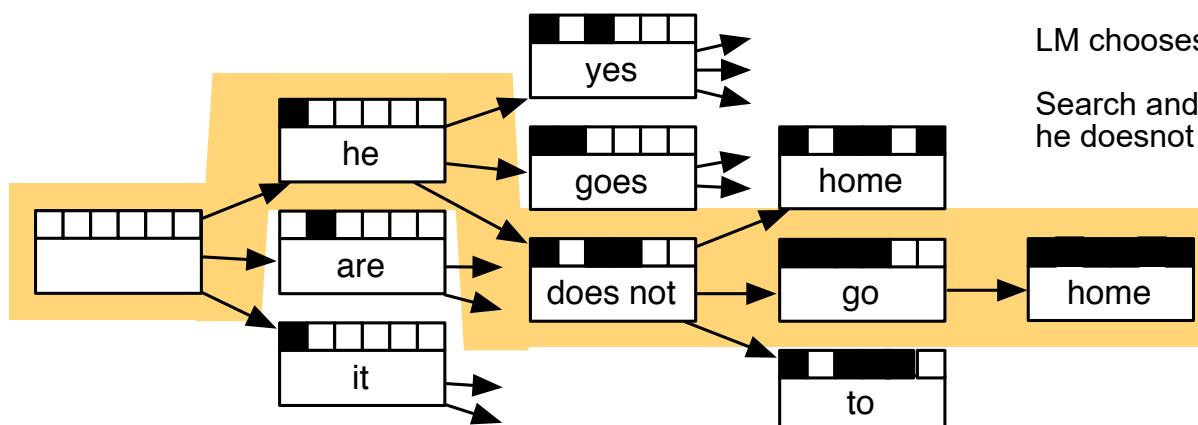


- We could enumerate every possible y and calculate the probability? → Too expensive!
- Answer: Impose strong **independence assumptions** in model, use dynamic programming for globally optimal solutions (e.g. Viterbi algorithm).
- This process is called *decoding*

Decoding for SMT



From translation model we have the words or phrases that are reasonably likely translation of the german words or phrases.



Search process: start with he are or it

LM chooses he

Search and prune for likely translation
he does not go home.

Source: "Statistical Machine Translation", Chapter 6, Koehn, 2009.

<https://www.cambridge.org/core/books/statistical-machine-translation/94EADF9F680558E13BE759997553CDE5>

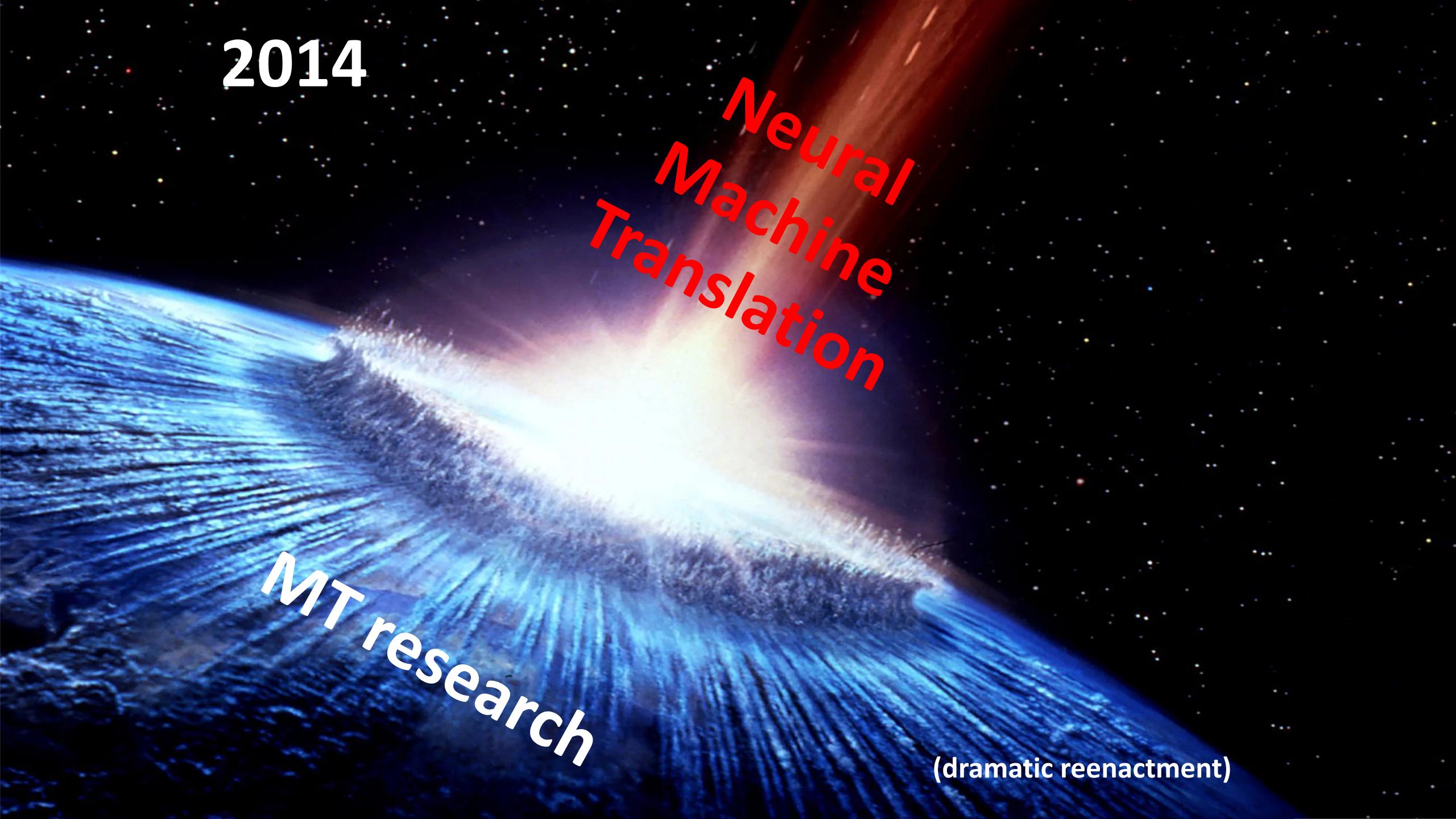
1990s-2010s: Statistical Machine Translation

- 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

2014

(dramatic reenactment)

A dramatic reenactment of the birth of neural machine translation. The scene is set against a dark, star-filled background. A massive, bright, multi-colored wave of light and energy, resembling a supernova or a tidal wave, is shown crashing down from the top left towards the bottom right. The wave's base is a deep blue, transitioning through white, yellow, orange, and red at its peak. In the upper right quadrant, the words "Neural Machine Translation" are written in a large, bold, red serif font, tilted diagonally upwards. In the lower left quadrant, the words "MT research" are written in a large, white, italicized serif font, also tilted diagonally upwards. The overall effect is one of a powerful, transformative event.

2014

Neural
Machine
Translation

MT research

(dramatic reenactment)

What is Neural Machine Translation?

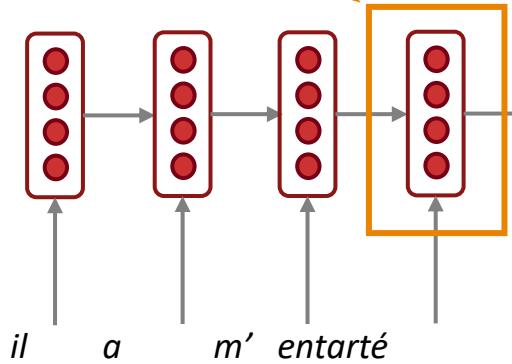
- Neural Machine Translation (NMT) is a way to do Machine Translation with a *single end-to-end neural network*
- The neural network architecture is called a *sequence-to-sequence* model (aka *seq2seq*) and it involves *two RNNs*

Neural Machine Translation (NMT)

The sequence-to-sequence model

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

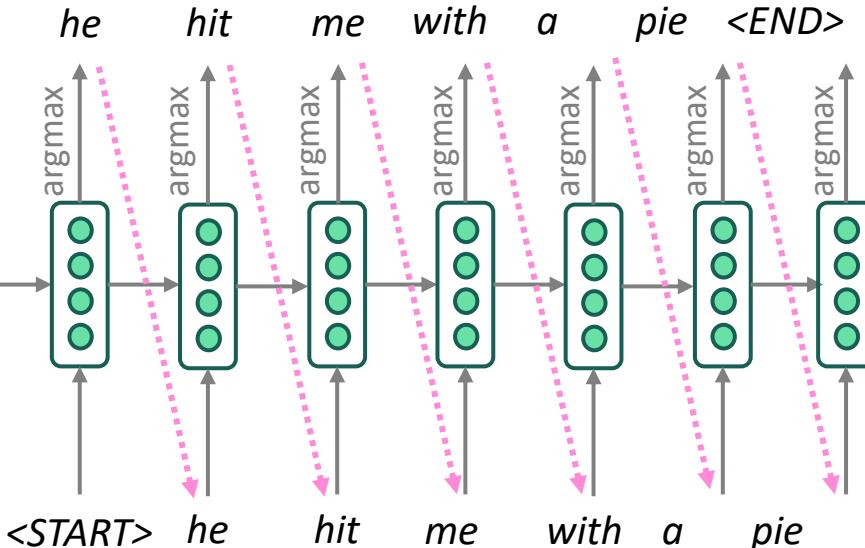
Encoder RNN



Source sentence (input)

Encoder RNN produces
an **encoding** of the
source sentence.

Target sentence (output)



Decoder RNN

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) Dependency Parsing etc.
 - **Code generation** (natural language → Python code)

Neural Machine Translation (NMT)

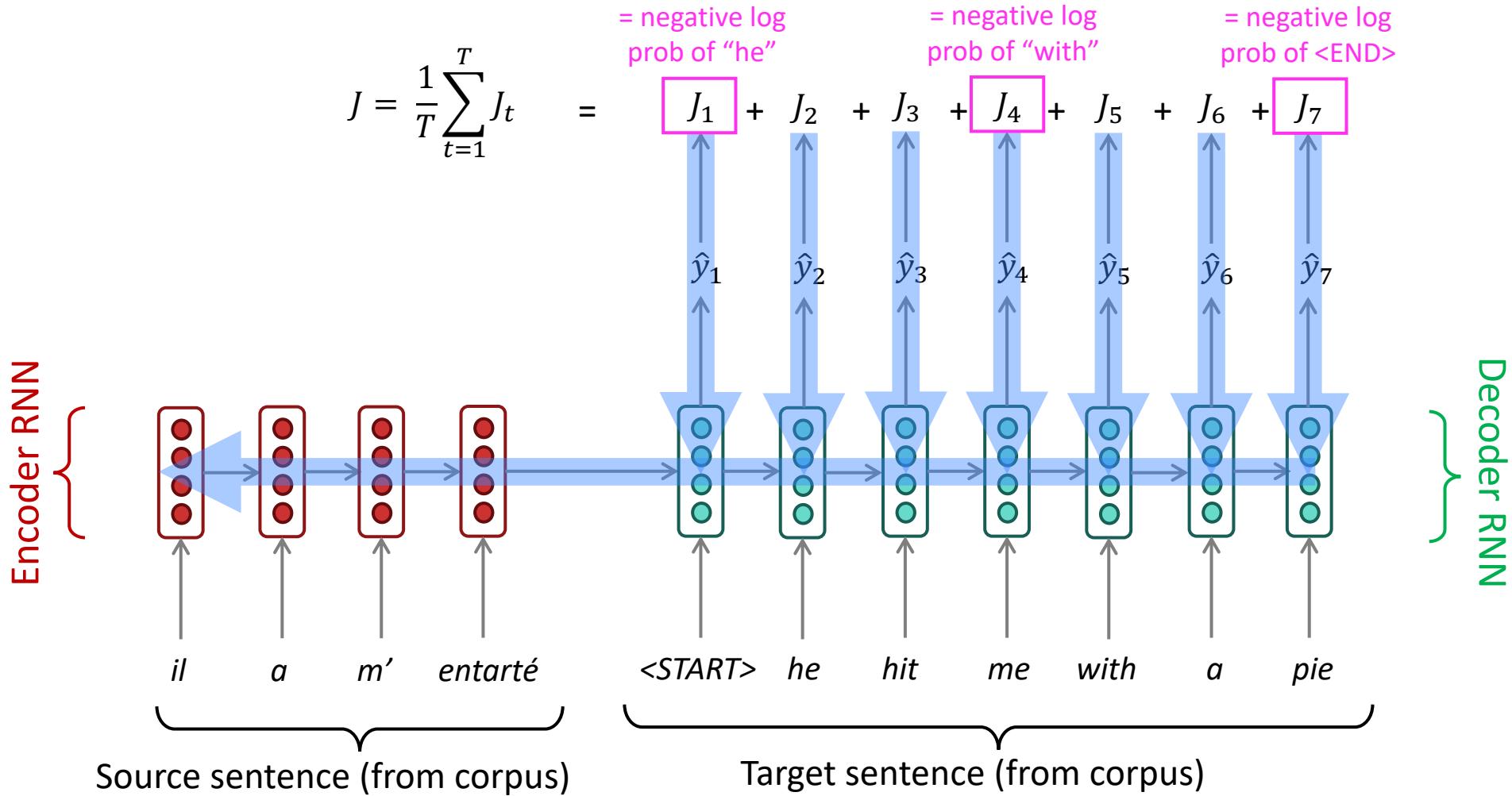
- The **sequence-to-sequence** model is an example of a **Conditional Language Model**
 - **Language Model** because the decoder is predicting the next word of the target sentence y
 - **Conditional** because its predictions are *also* conditioned on the source sentence x
- NMT directly calculates $P(y|x)$:

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

 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”.

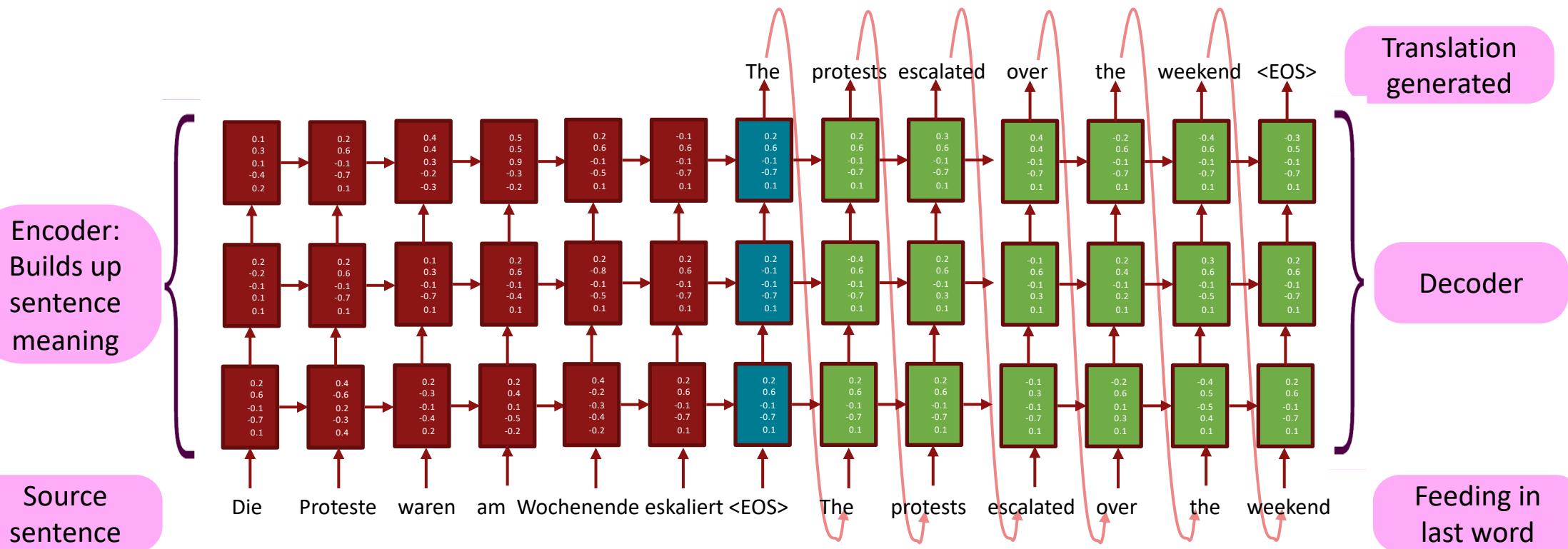
Multi-layer RNNs

- RNNs are already “deep” on one dimension (they unroll over many timesteps)
- We can also make them “deep” in another dimension by applying multiple RNNs
 - this is a multi-layer RNN.
- This allows the network to compute more complex representations
 - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.
 - Lower level Features: More basic things about words and phrases, Parts of Speech, Named entity etc
 - Higher Level Features: Higher semantic level, overall structure of a sentence, positive or negative connotations, what it means, etc
- Multi-layer RNNs are also called *stacked RNNs*.

Multi-layer deep encoder-decoder machine translation net

[Sutskever et al. 2014; Luong et al. 2015]

The hidden states from RNN layer i
are the inputs to RNN layer $i+1$



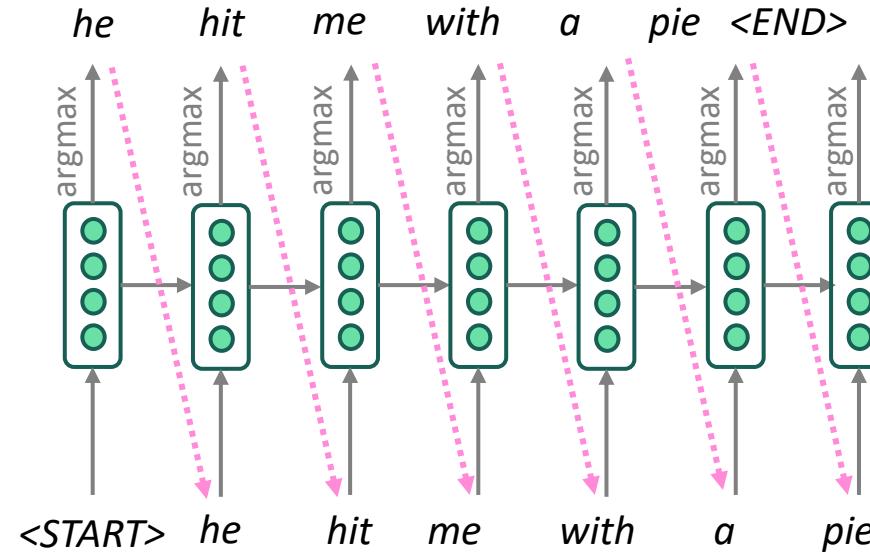
Conditioning =
Bottleneck

Multi-layer RNNs in practice

- High-performing RNNs are usually multi-layer (but aren't as deep as convolutional or feed-forward networks)
- For example: In a 2017 paper, Britz et al. find that for Neural Machine Translation, 2 to 4 layers is best for the encoder RNN, and 4 layers is best for the decoder RNN
 - Often 2 layers is a lot better than 1, and 3 might be a little better than 2
 - Usually, skip-connections/dense-connections are needed to train deeper RNNs (e.g., 8 layers)
- Transformer-based networks (e.g., BERT) are usually deeper, like 12 or 24 layers.
 - You will learn about Transformers later; they have a lot of skipping-like connections

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

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

- 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

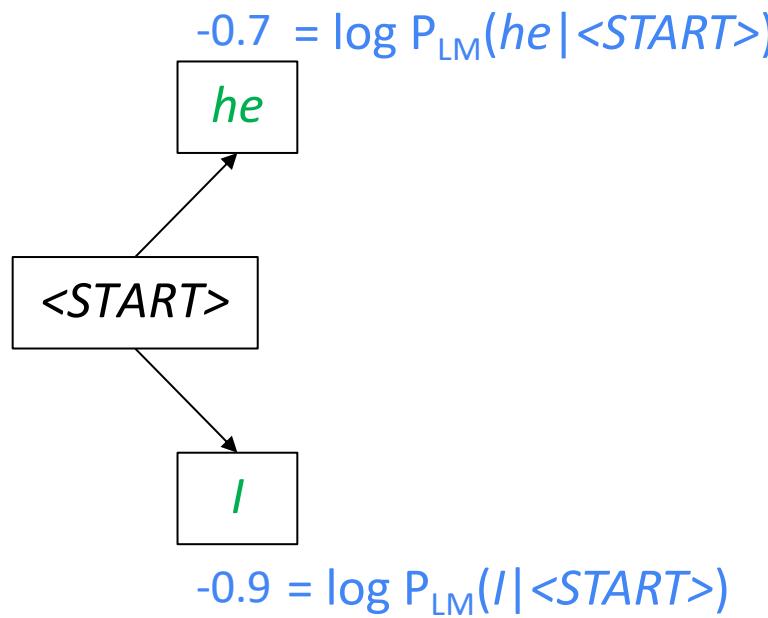
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$

<START>

Calculate prob
dist of next word

Beam search decoding: example

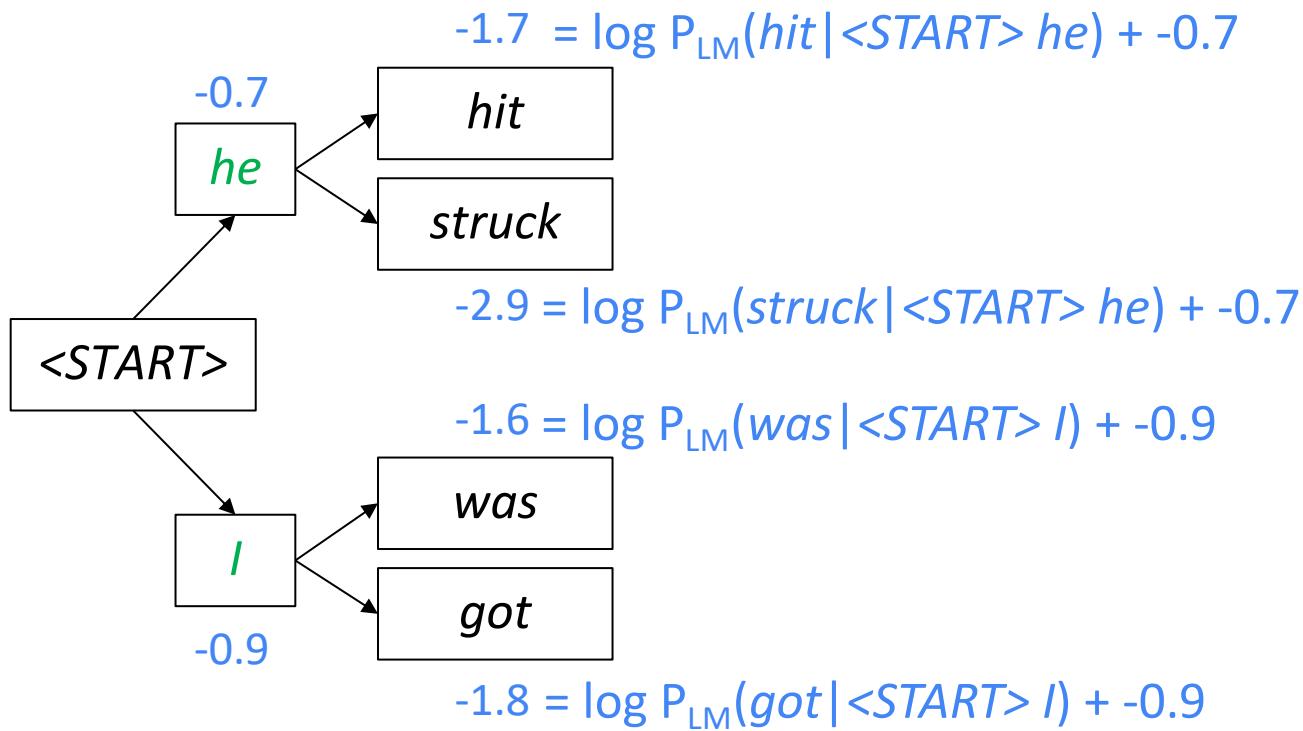
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



Take top k words
and compute scores

Beam search decoding: example

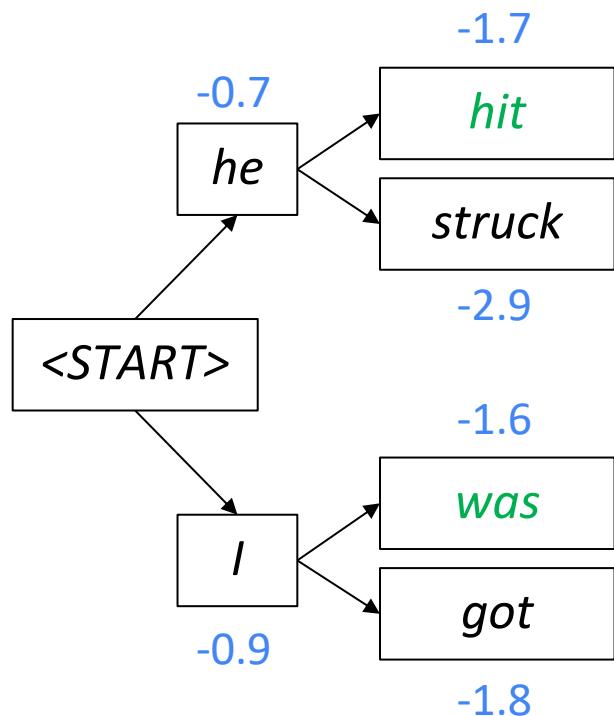
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

Beam search decoding: example

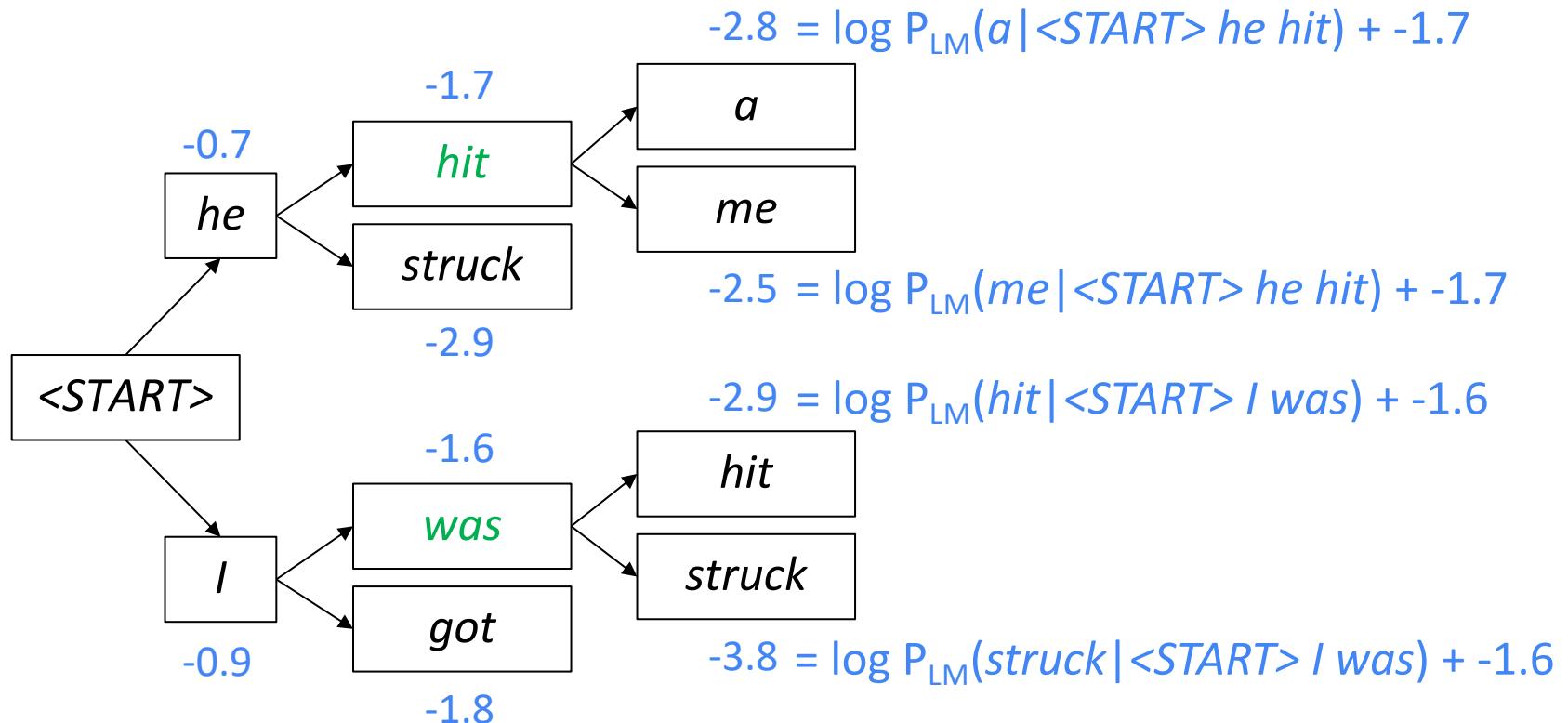
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

Beam search decoding: example

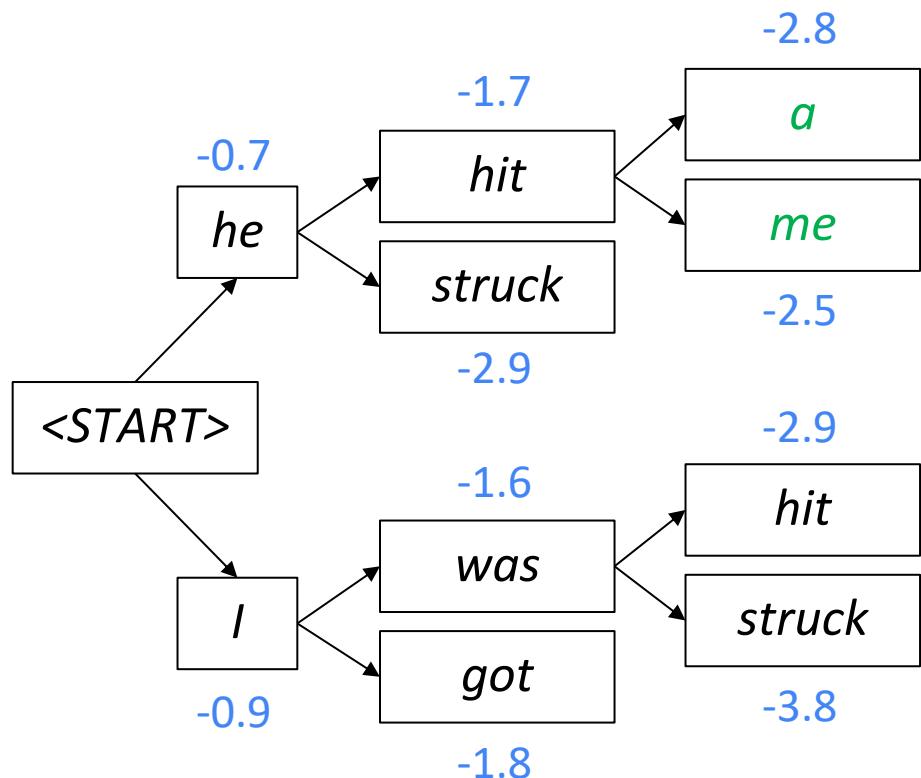
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

Beam search decoding: example

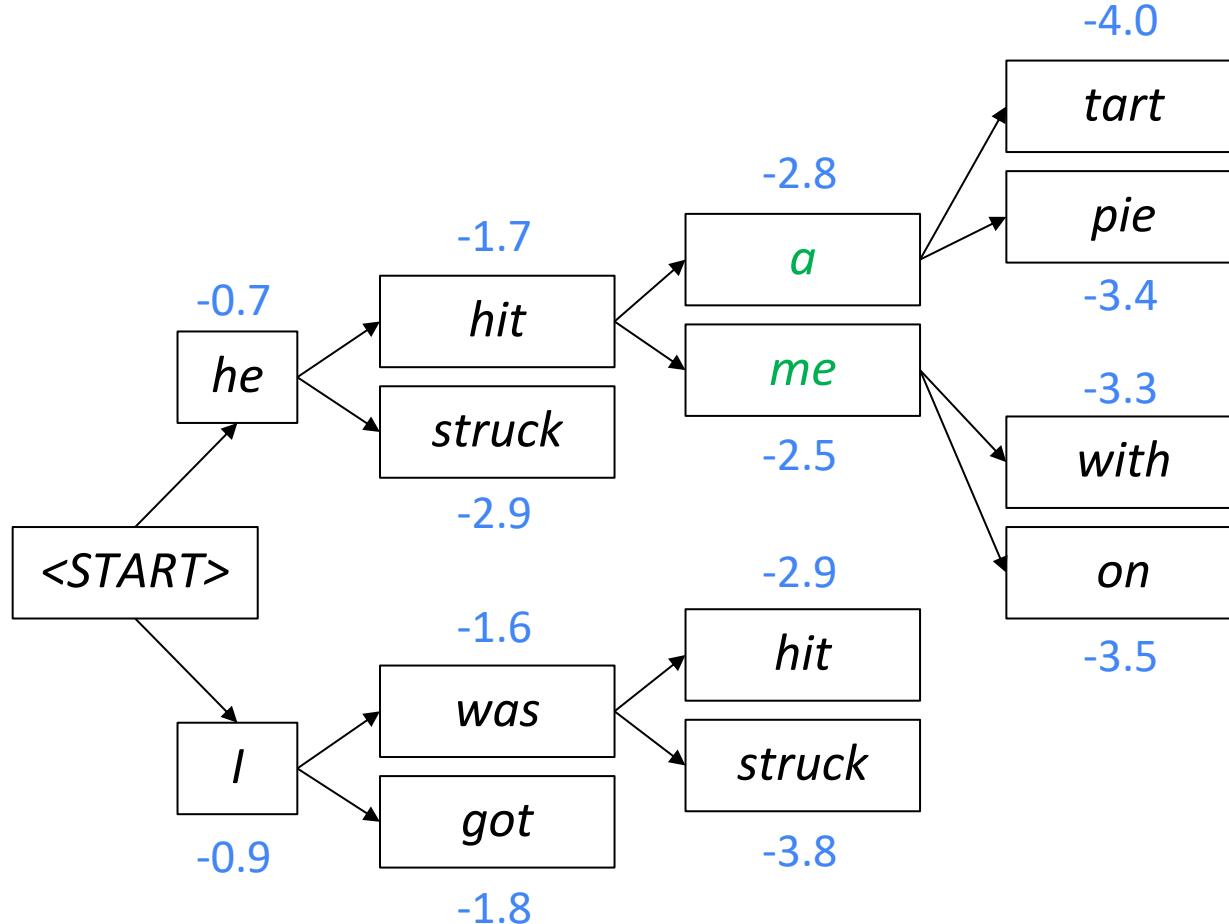
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

Beam search decoding: example

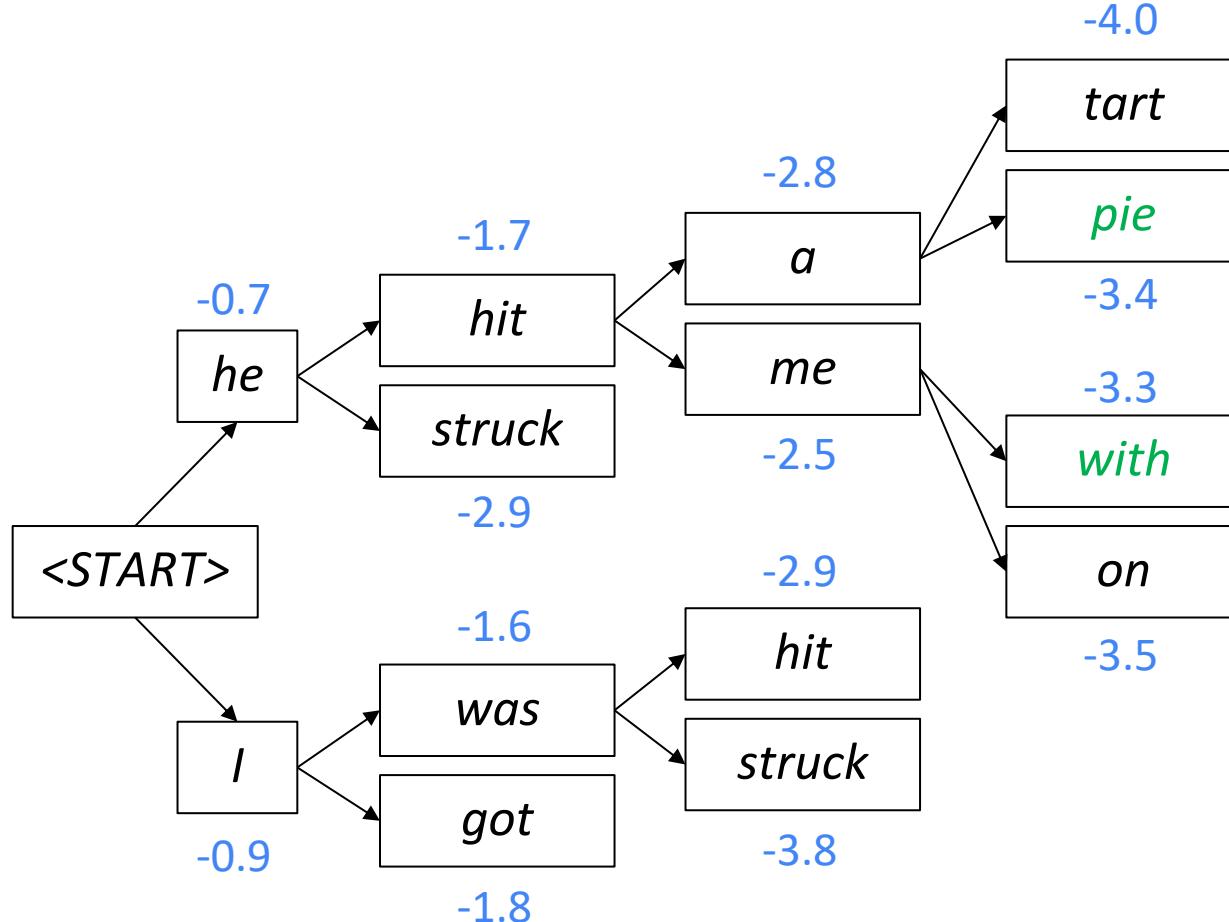
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

Beam search decoding: example

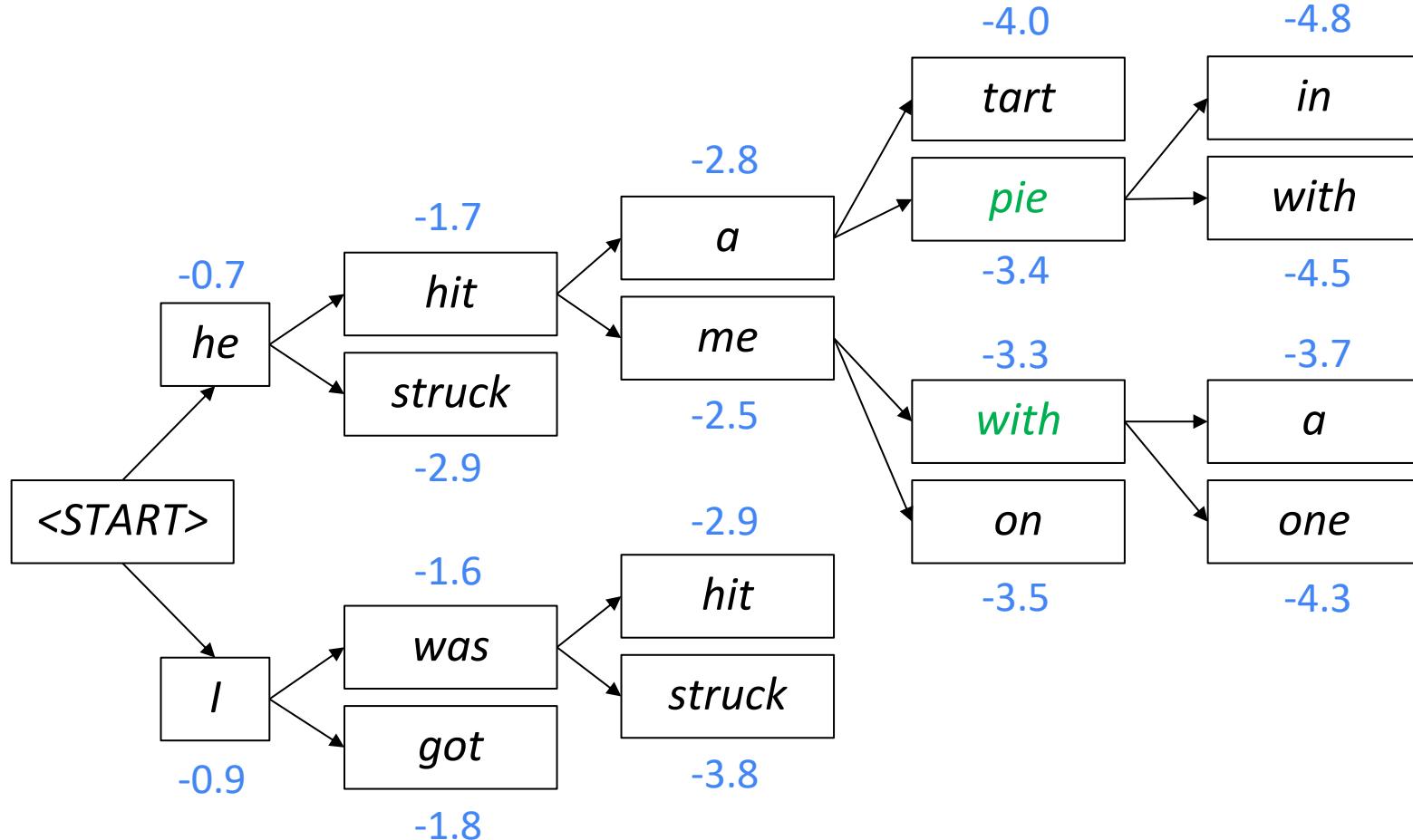
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

Beam search decoding: example

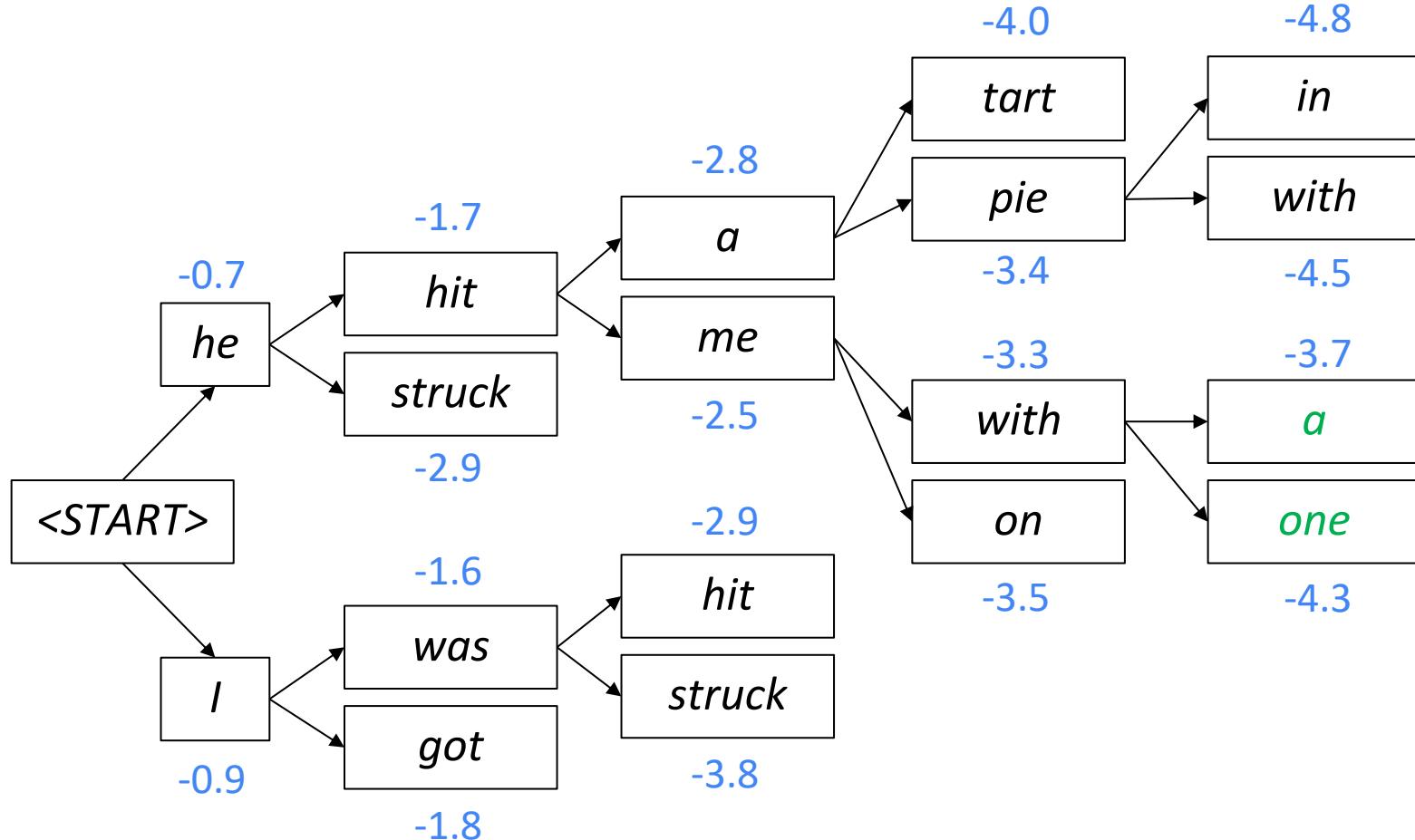
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

Beam search decoding: example

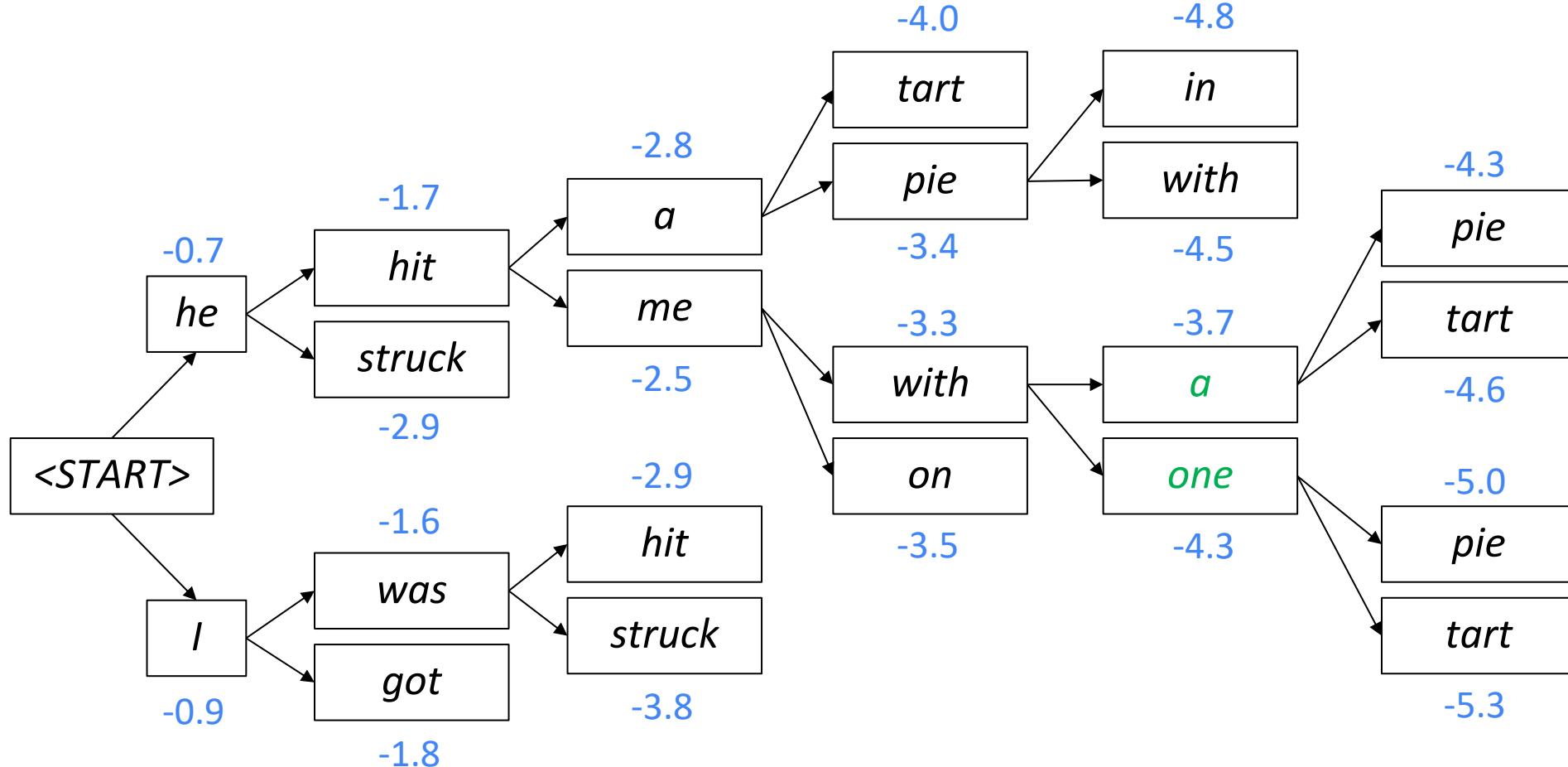
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

Beam search decoding: example

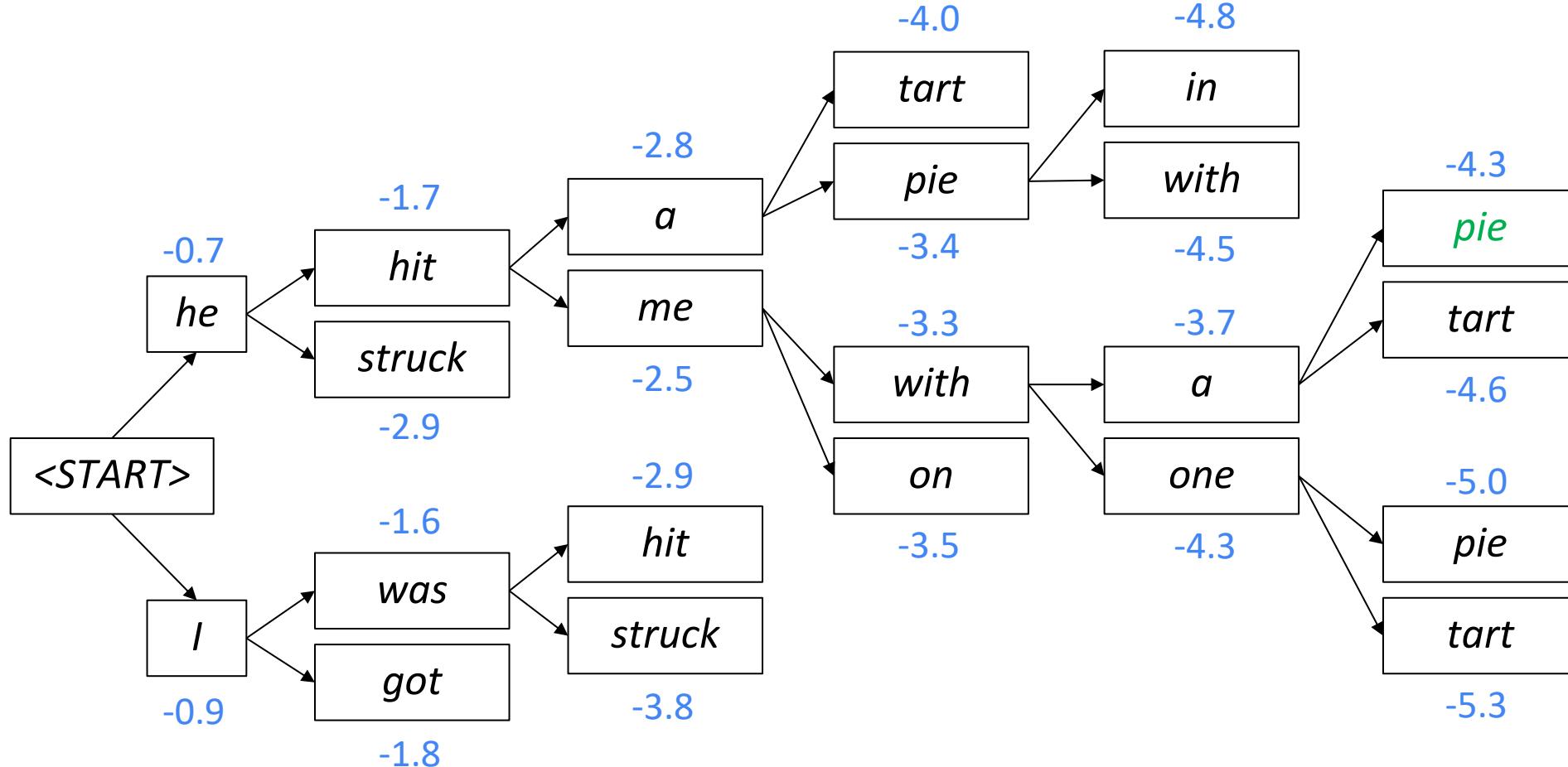
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



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

Beam search decoding: example

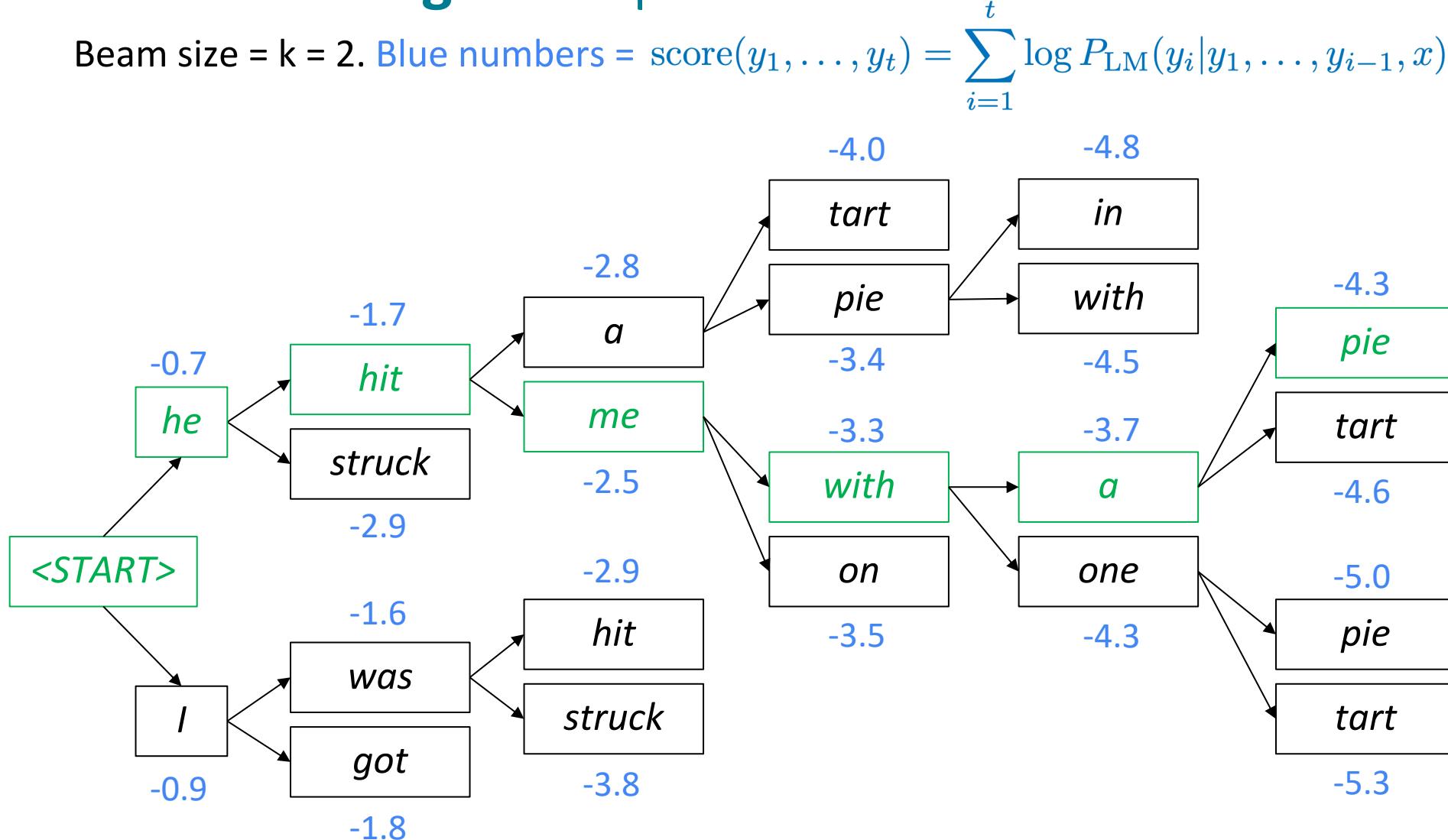
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



This is the top-scoring hypothesis!

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 an <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)

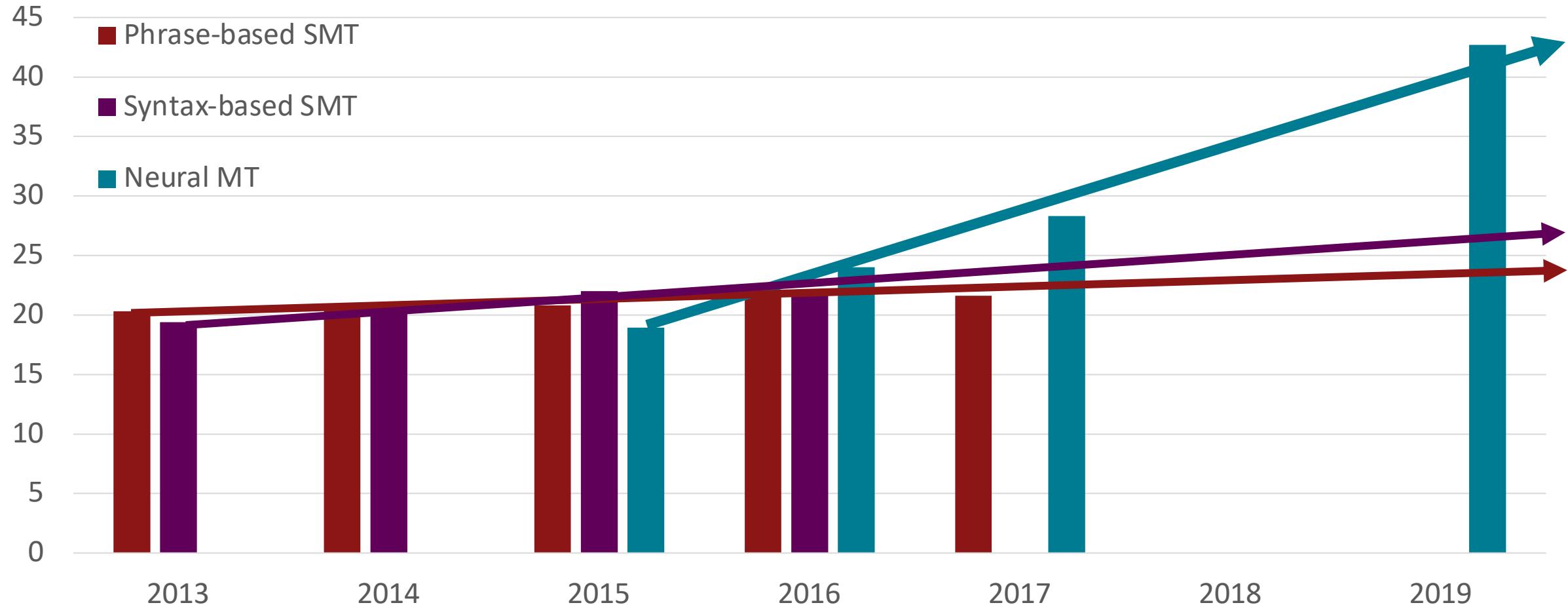
You'll see BLEU in detail
in Assignment 4!

- 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 ☹

BLEU: 0 -100, 100 if exact match to human written translation.
0 if no ngram overlap

MT progress over time

[Edinburgh En-De WMT newstest2013 Cased BLEU; NMT 2015 from U. Montréal; NMT 2019 FAIR on newstest2019]



Sources: http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf & <http://matrix.statmt.org/>

NMT: perhaps the biggest 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
- **2016:** Google Translate switches from SMT to NMT – and by 2018 everyone has



Microsoft



SYSTRAN
beyond language

Google

facebook

Baidu 百度

網易 NETEASE
www.163.com

Tencent 腾讯

S 搜狗搜索

- This is amazing!
 - **SMT** systems, built by **hundreds** of engineers over many **years**, outperformed by NMT systems trained by a **small group** 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
 - Failures to accurately capture sentence meaning
 - Pronoun (or zero pronoun) resolution errors
 - Morphological agreement errors

Further reading: “Has AI surpassed humans at translation? Not even close!”

https://www.skynettoday.com/editorials/state_of_nmt

So is Machine Translation solved?

- **Nope!**
- Using common sense is still hard



The image shows a screenshot of the Google Translate interface. On the left, under "English", the text "paper jam" is displayed with an "Edit" link. On the right, under "Spanish", the text "Mermelada de papel" is displayed. Both sides have microphone and speaker icons above them. Below the English input is a link "Open in Google Translate". Below the Spanish output is a link "Feedback".



So is Machine Translation solved?

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

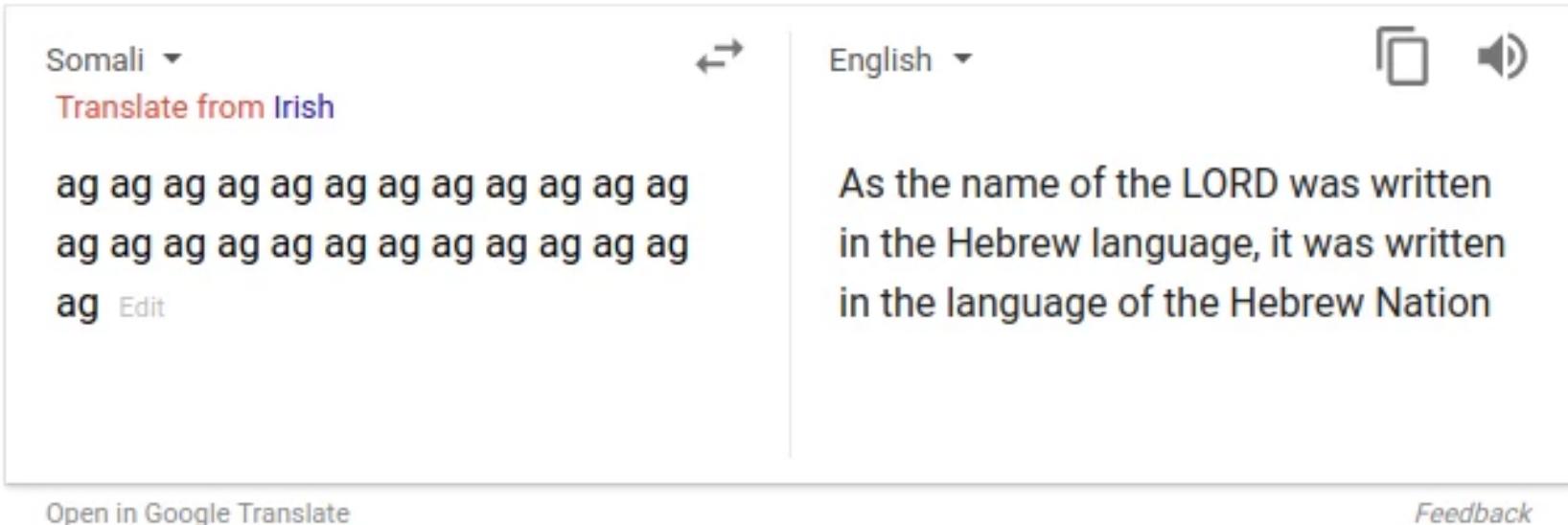
The screenshot shows a machine translation tool with Malay input and English output. The Malay input is "Dia bekerja sebagai jururawat." and "Dia bekerja sebagai pengaturcara." The English output is "She works as a nurse." and "He works as a programmer." An upward arrow points from the Malay input to the English output, highlighting the lack of gender specification in the input.

| Malay - detected | English |
|-----------------------------------|---------------------------|
| Dia bekerja sebagai jururawat. | She works as a nurse. |
| Dia bekerja sebagai pengaturcara. | He works as a programmer. |

Didn't specify gender

So is Machine Translation solved?

- Nope!
- Uninterpretable systems do strange things
- (But I think this problem has been fixed in Google Translate by 2021?)



Picture source: 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 a **flagship task** for NLP Deep Learning

- NMT research has **pioneered** many of the recent **innovations** of NLP Deep Learning
- In **2021**: NMT research continues to **thrive**
 - Researchers have found **many, many improvements** to the “vanilla” seq2seq NMT system we’ve just presented
 - But we’ll present in a minute **one improvement** so integral that it is the new vanilla...

ATTENTION

Assignment 4: Cherokee-English machine translation!

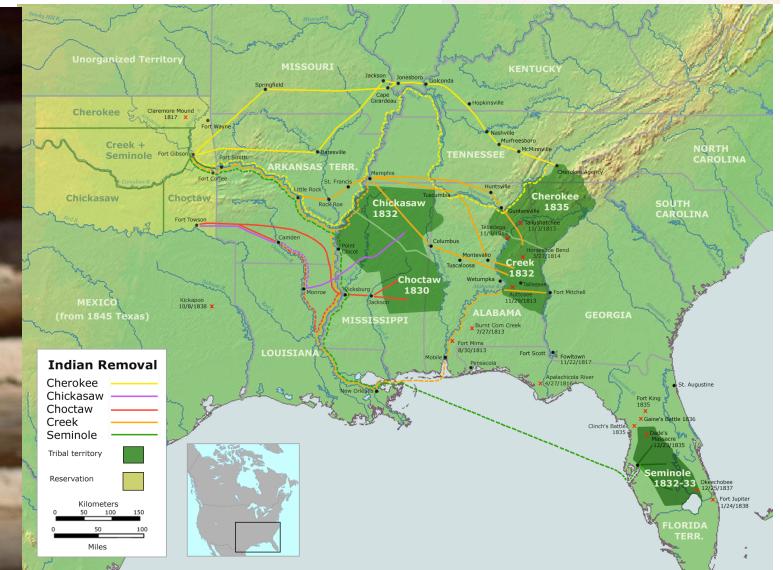
- Cherokee is an endangered Native American language – about 2000 fluent speakers
- Extremely low resource: About 20k parallel sentences available, most from the bible
- A^θYB hFRT SPV^Y T^θhT Dh^θG. hA^θAT J^θW^θT^θhLT Gh^θRT I^θLG^θh^θET O^θ APW^θT S^θI^θ Dh^θITd^θPV^θT D^θ O^θI^θh^θLA S^θo^θh^θLA D^θO^θh^θFT D^θO^θh^θT.
Long ago were seven boys who used to spend all their time down by the townhouse playing games, rolling a stone wheel along the ground, sliding and striking it with a stick
- Writing system is a syllabary of symbols for each CV unit (85 letters)
- Many thanks to Shiyue Zhang, Benjamin Frey, and Mohit Bansal from UNC Chapel Hill for the resources for this assignment!
- Cherokee is not available on Google Translate! 

Cherokee

- Cherokee originally lived in western North Carolina and eastern Tennessee
 - Most speakers now in Oklahoma, following the Trail of Tears; some in NC
 - Writing system Invented by Sequoyah around 1820 – someone who was previously illiterate
 - Very effective: In the following decades Cherokee literacy was higher than for white people in the southeastern United States

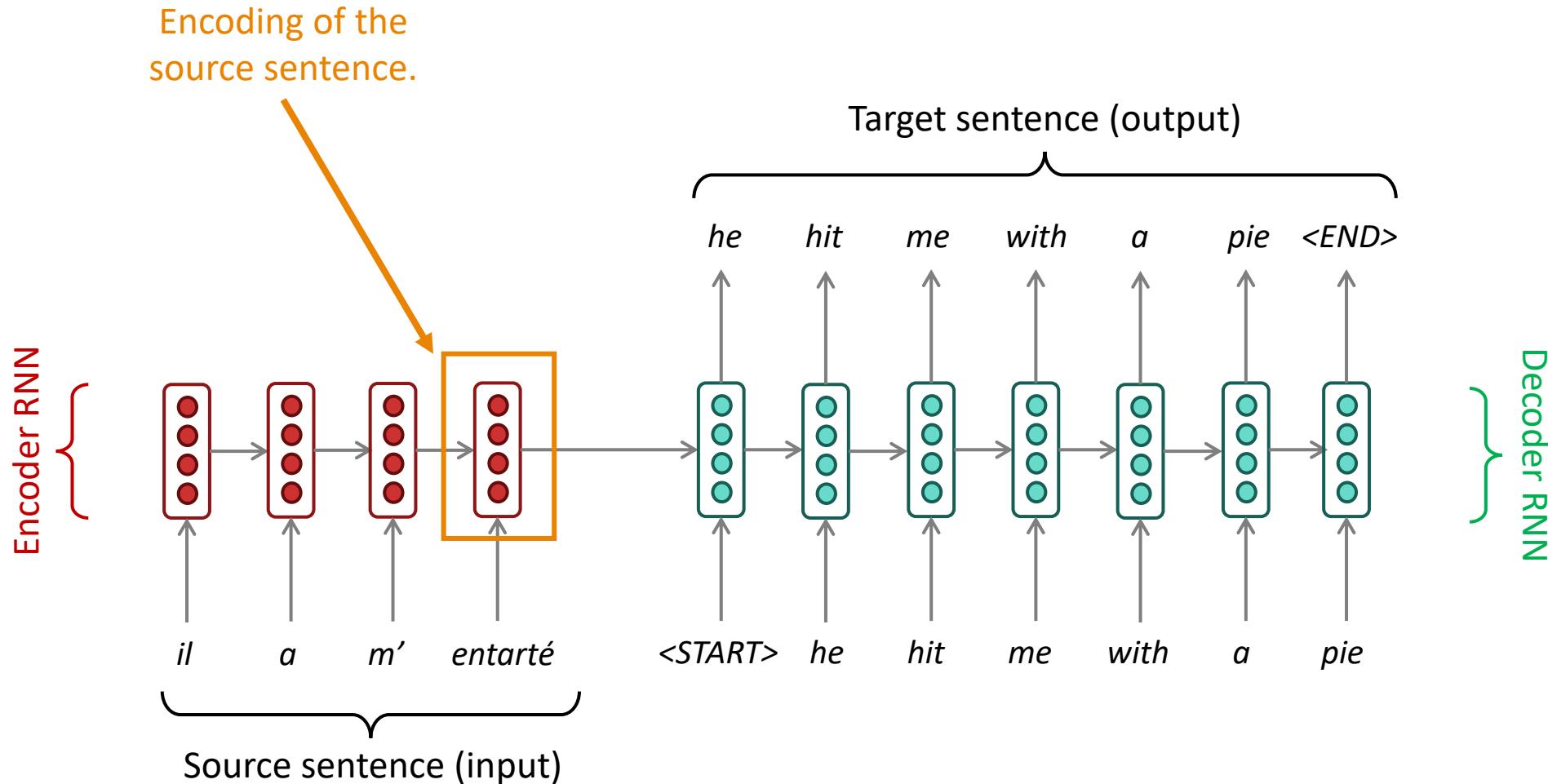


- <https://www.cherokee.org>



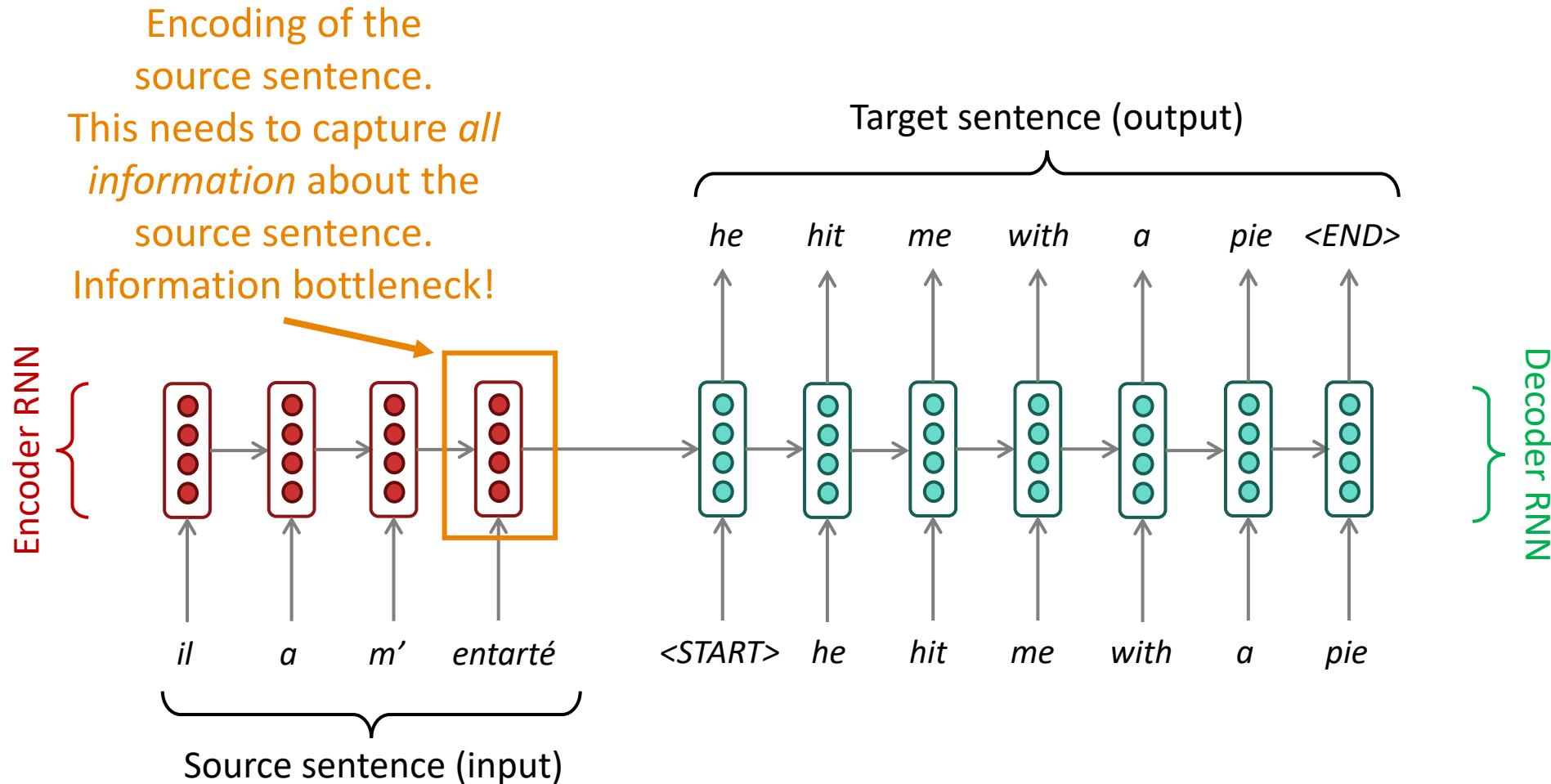
Section 3: Attention

Sequence-to-sequence: the bottleneck problem



Problems with this architecture?

Sequence-to-sequence: the bottleneck problem



Mean or averaging good for sentiment analysis and for other tasks but not for Machine Translation and other tasks that require info on order
Attention solves this issue

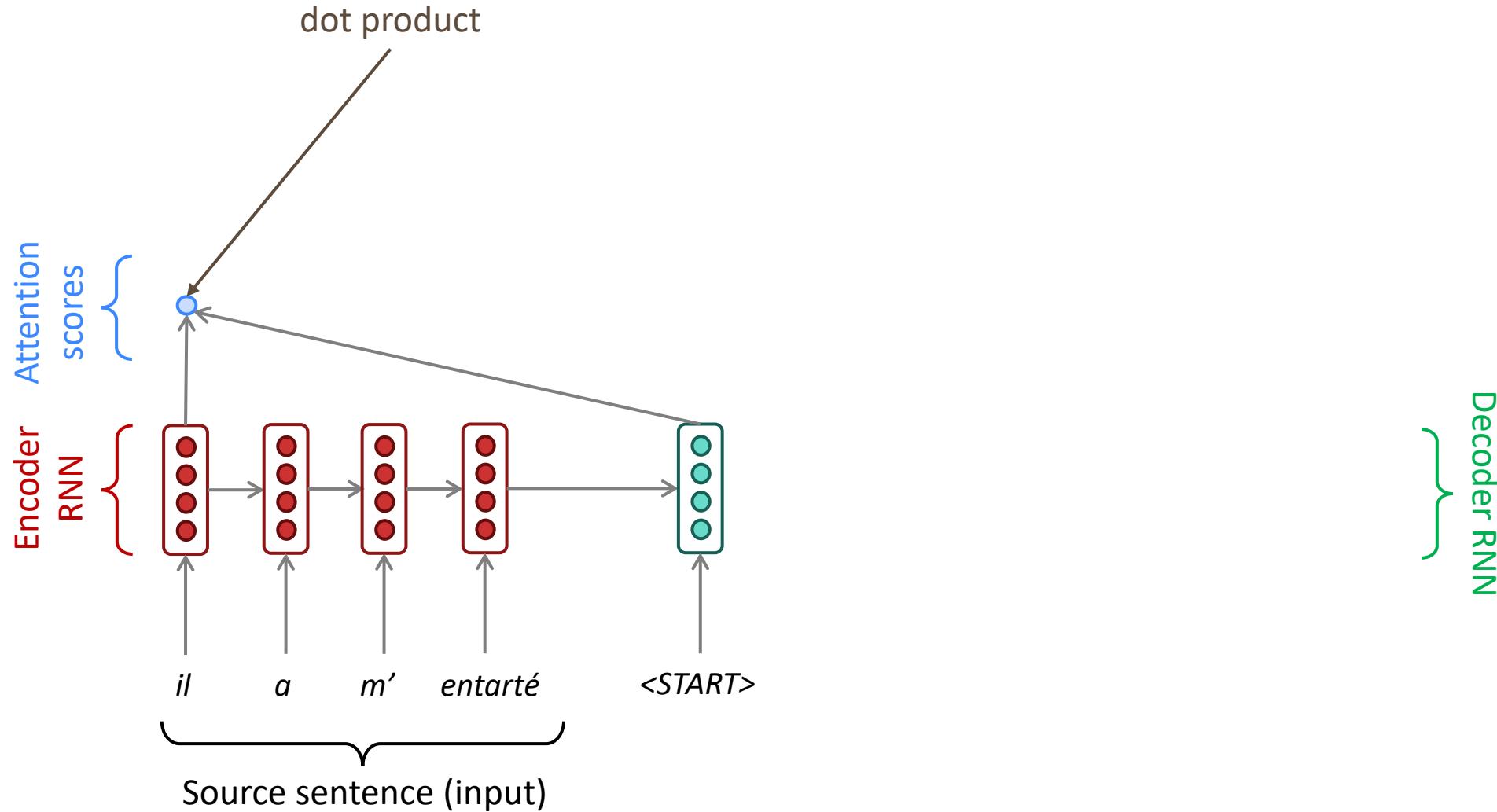
Attention

- 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

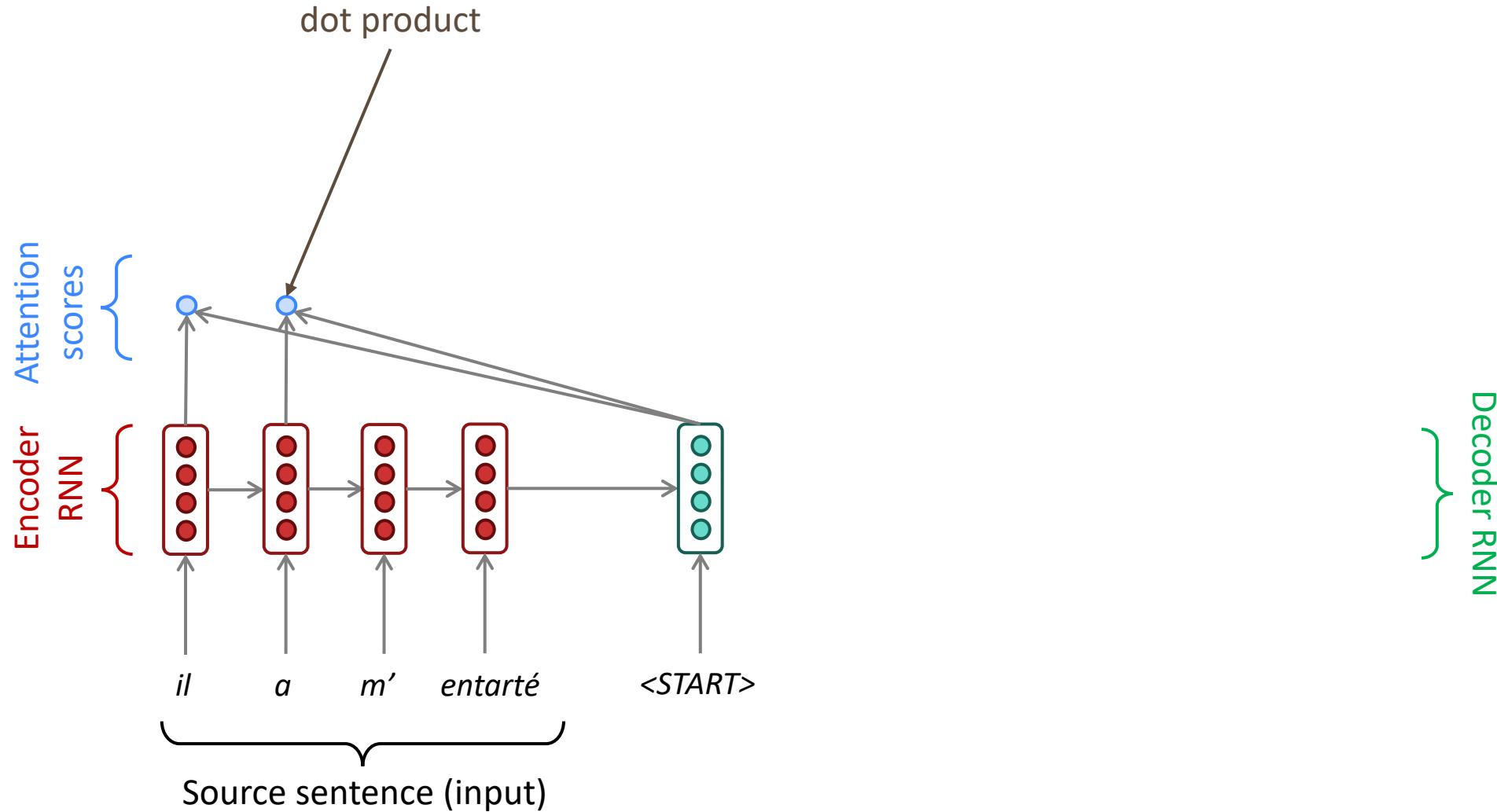


- First, we will show via diagram (no equations), then we will show with equations

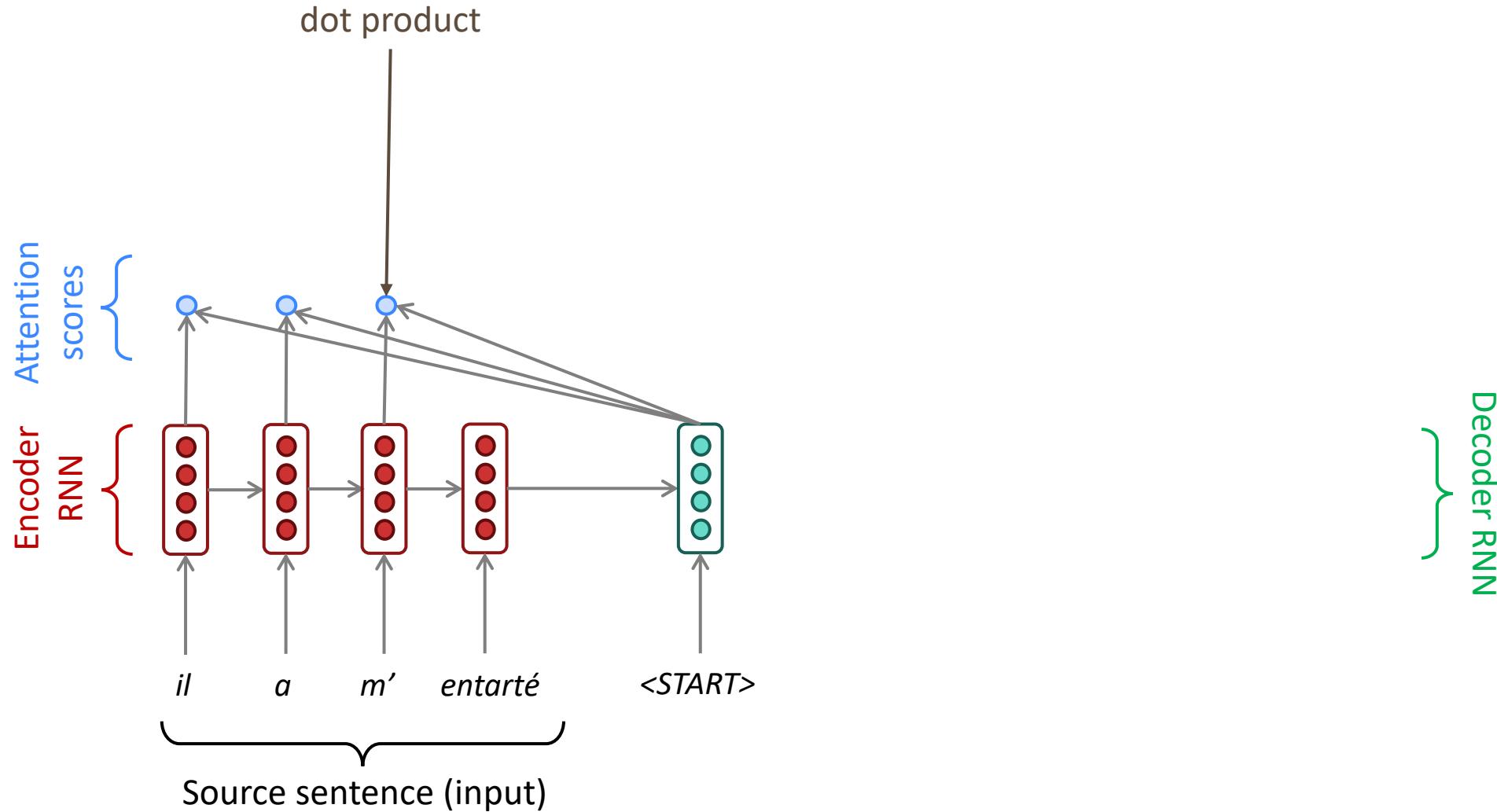
Sequence-to-sequence with attention



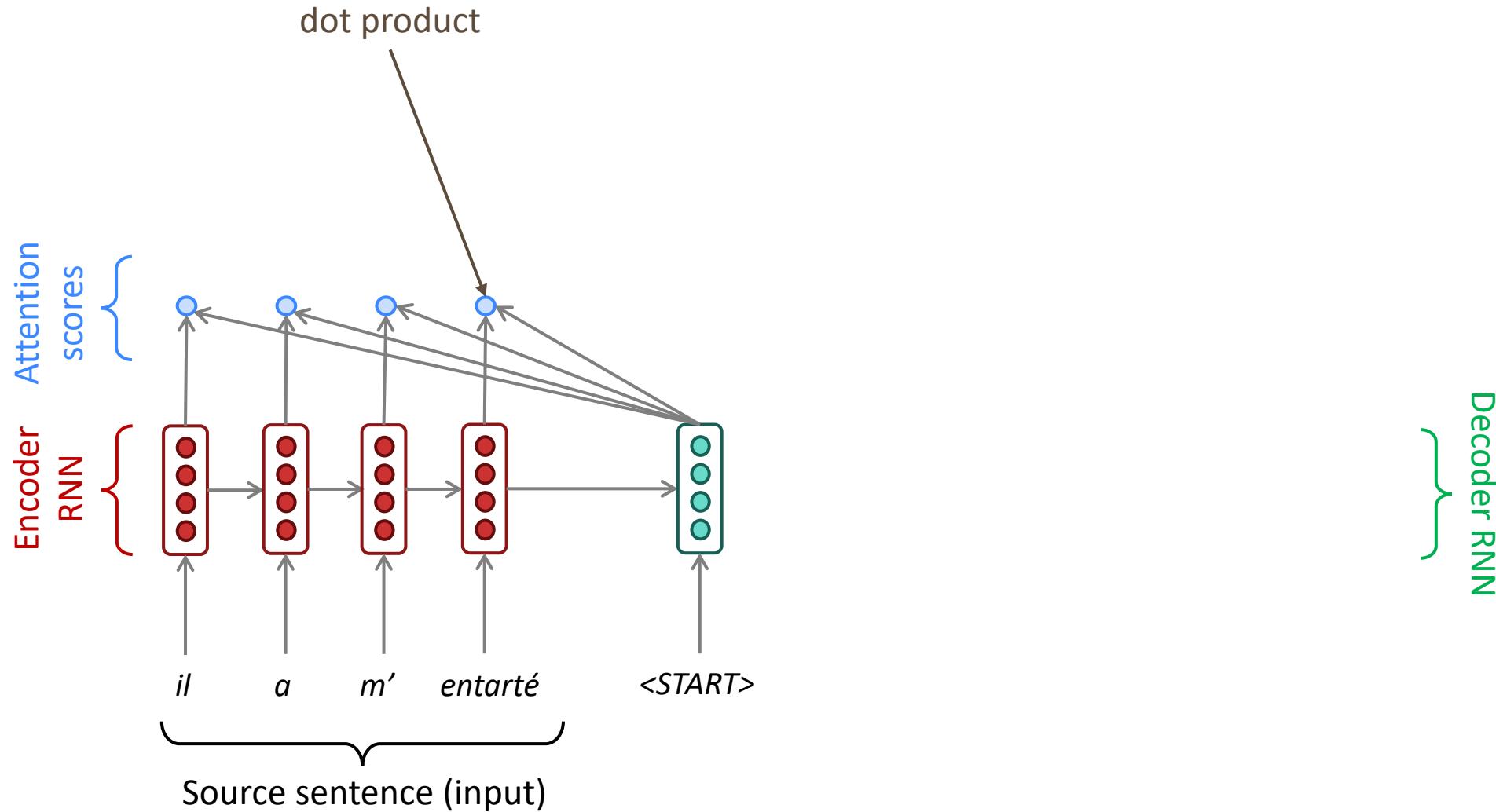
Sequence-to-sequence with attention



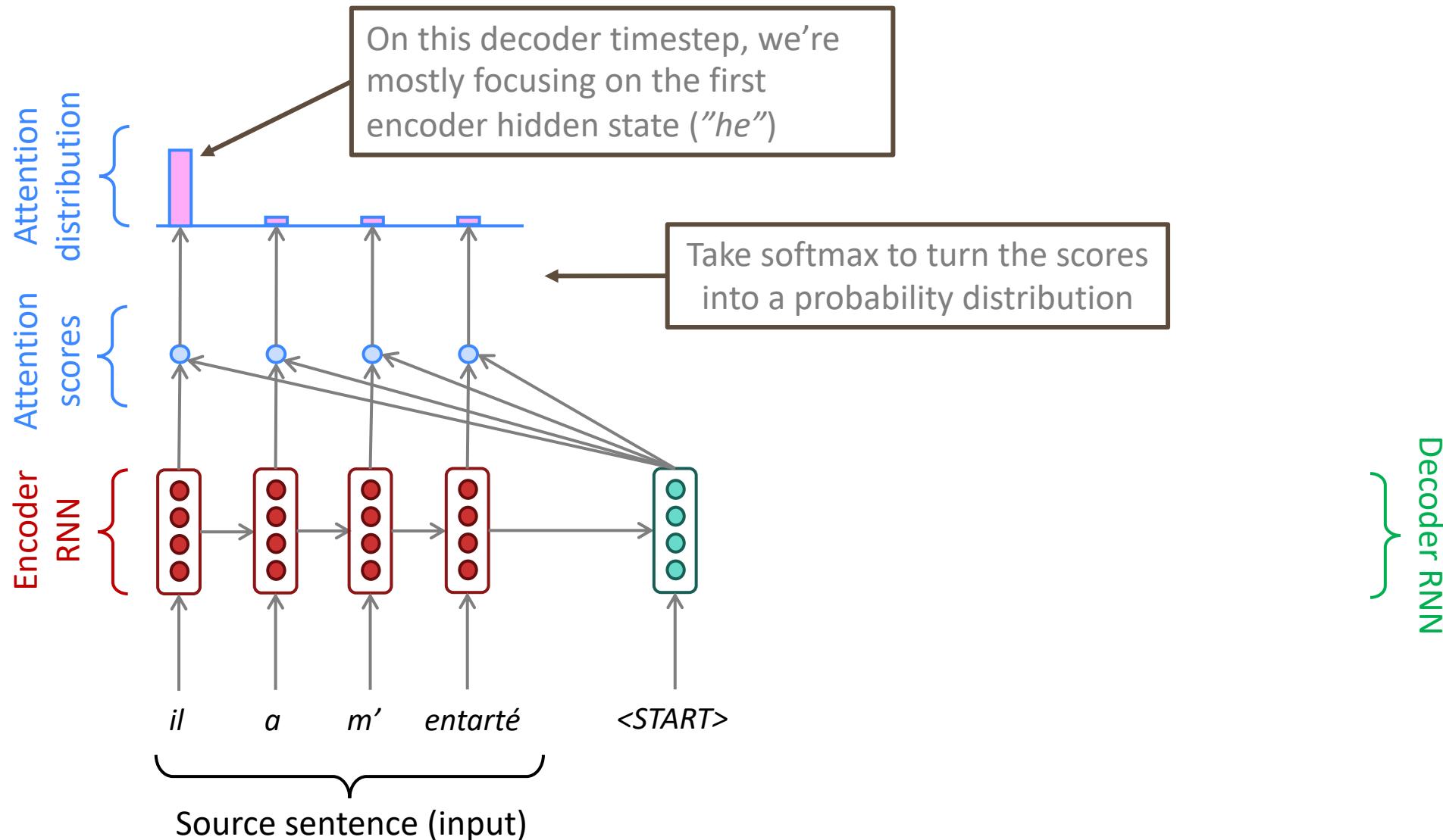
Sequence-to-sequence with attention



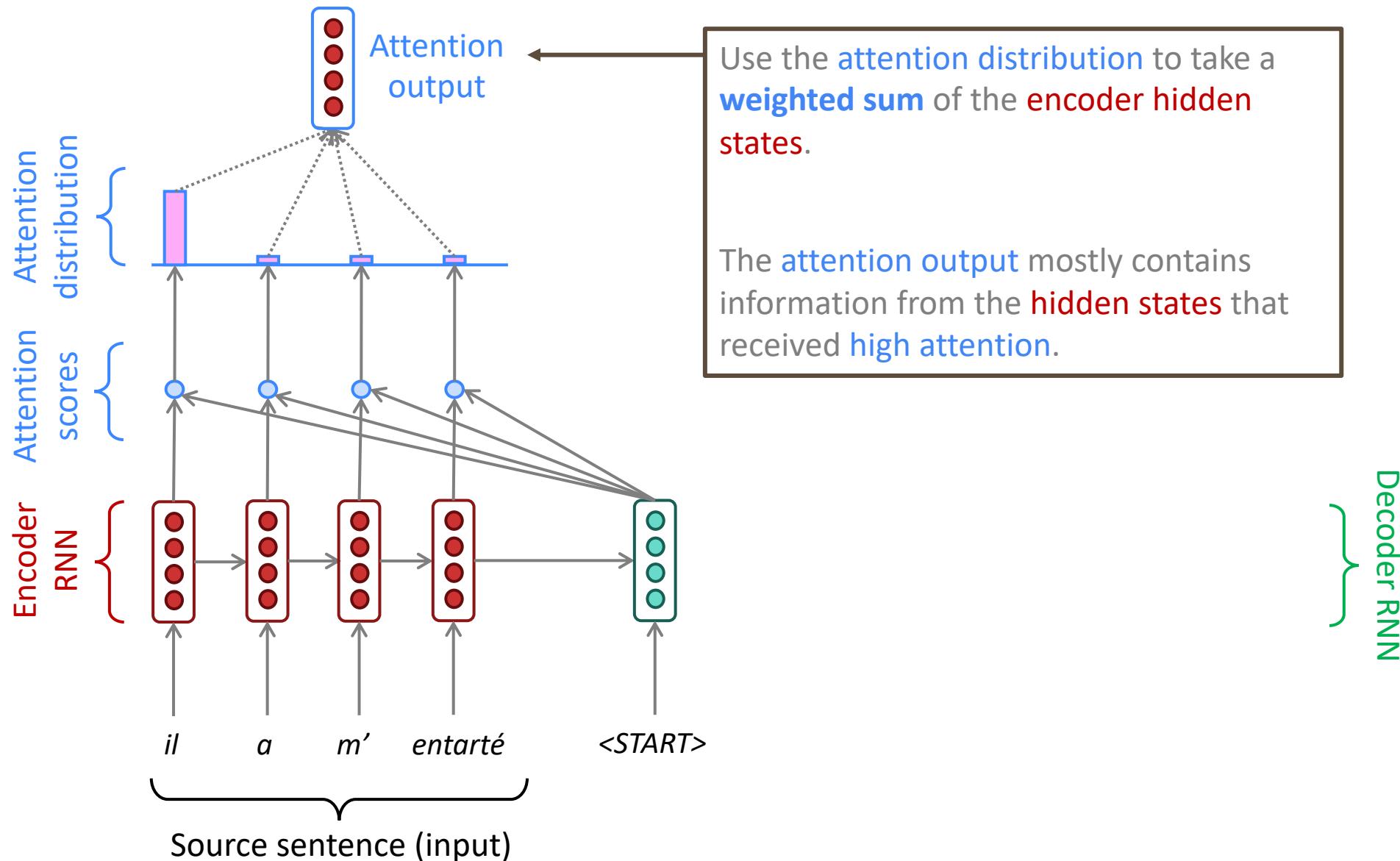
Sequence-to-sequence with attention



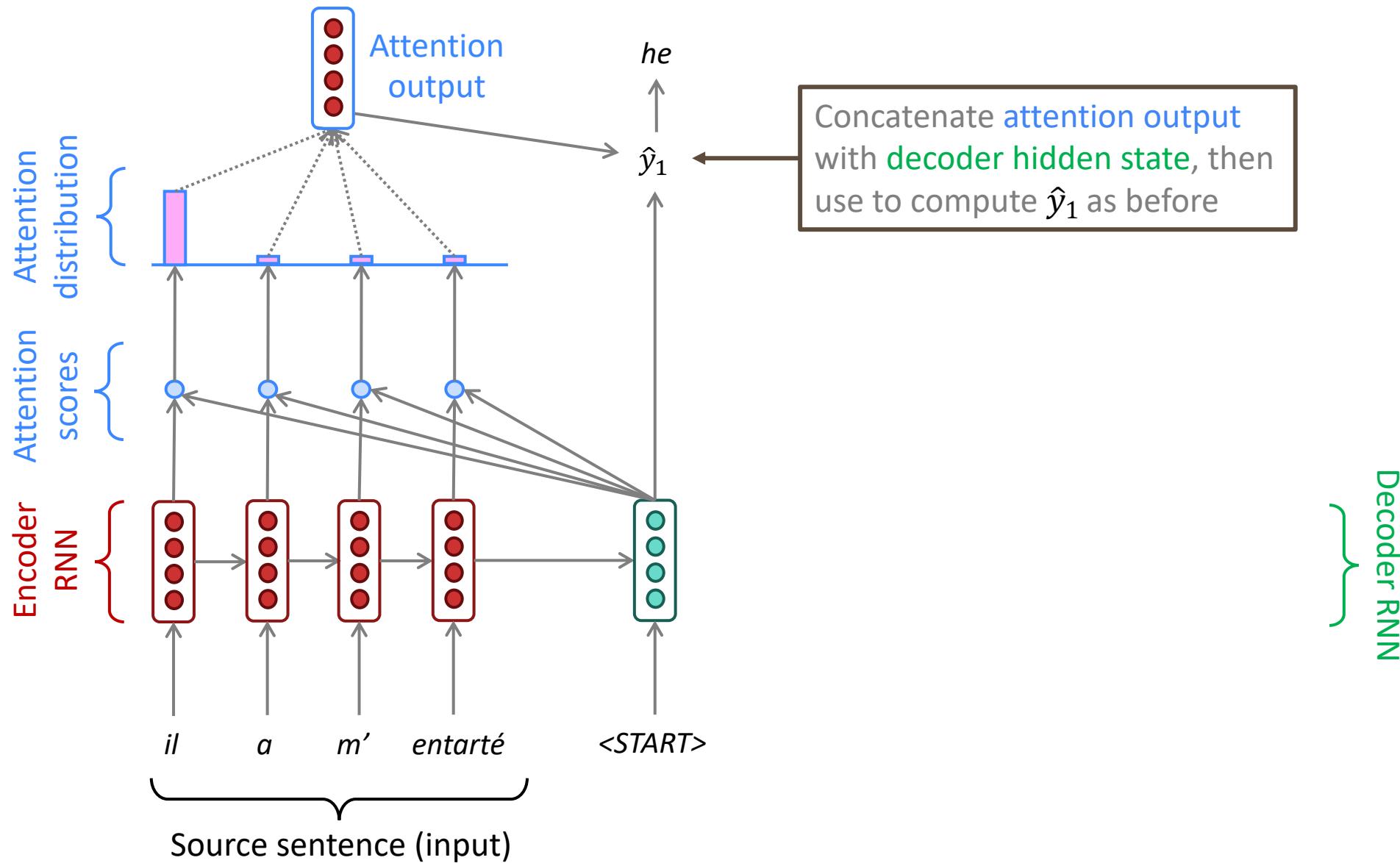
Sequence-to-sequence with attention



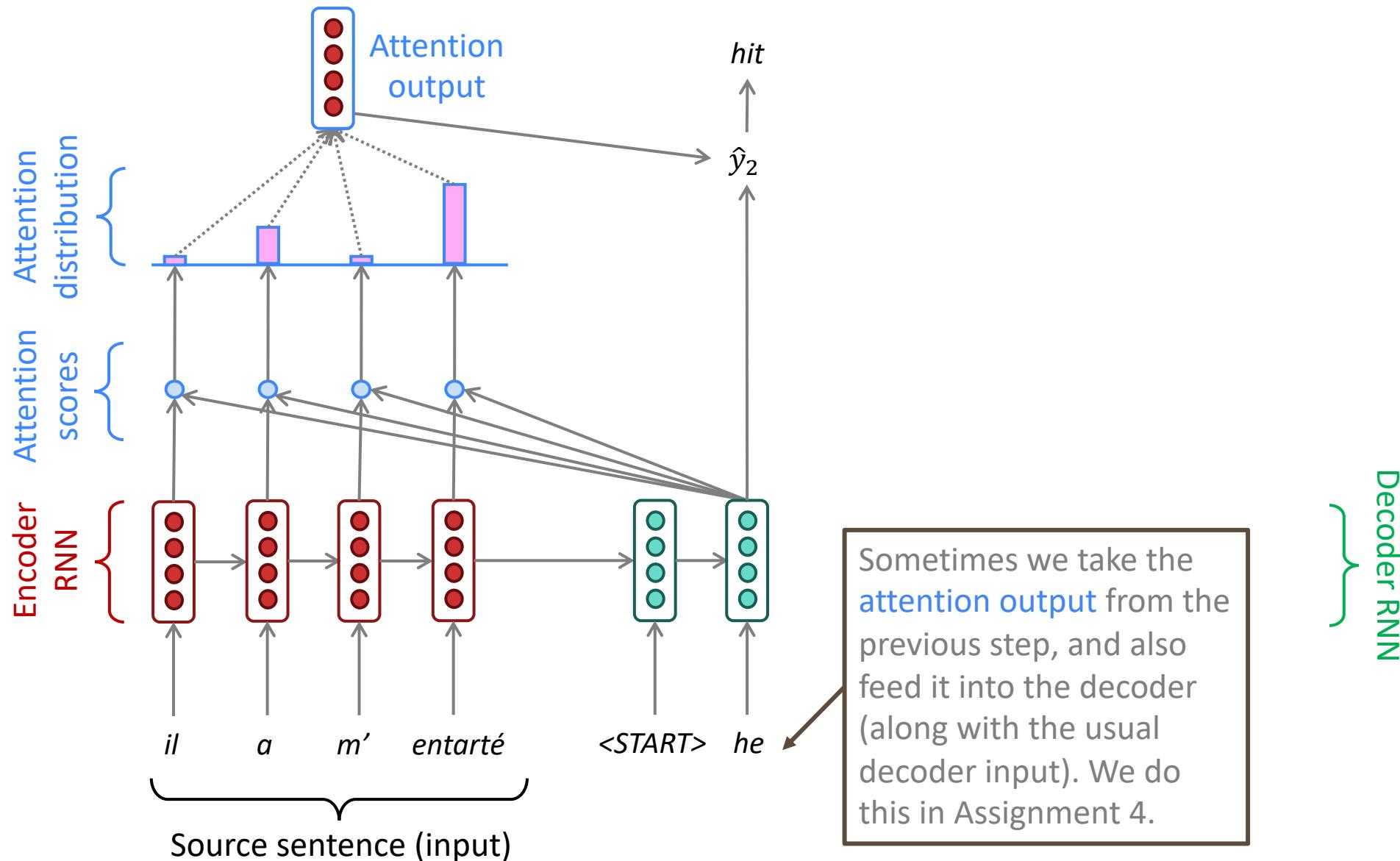
Sequence-to-sequence with attention



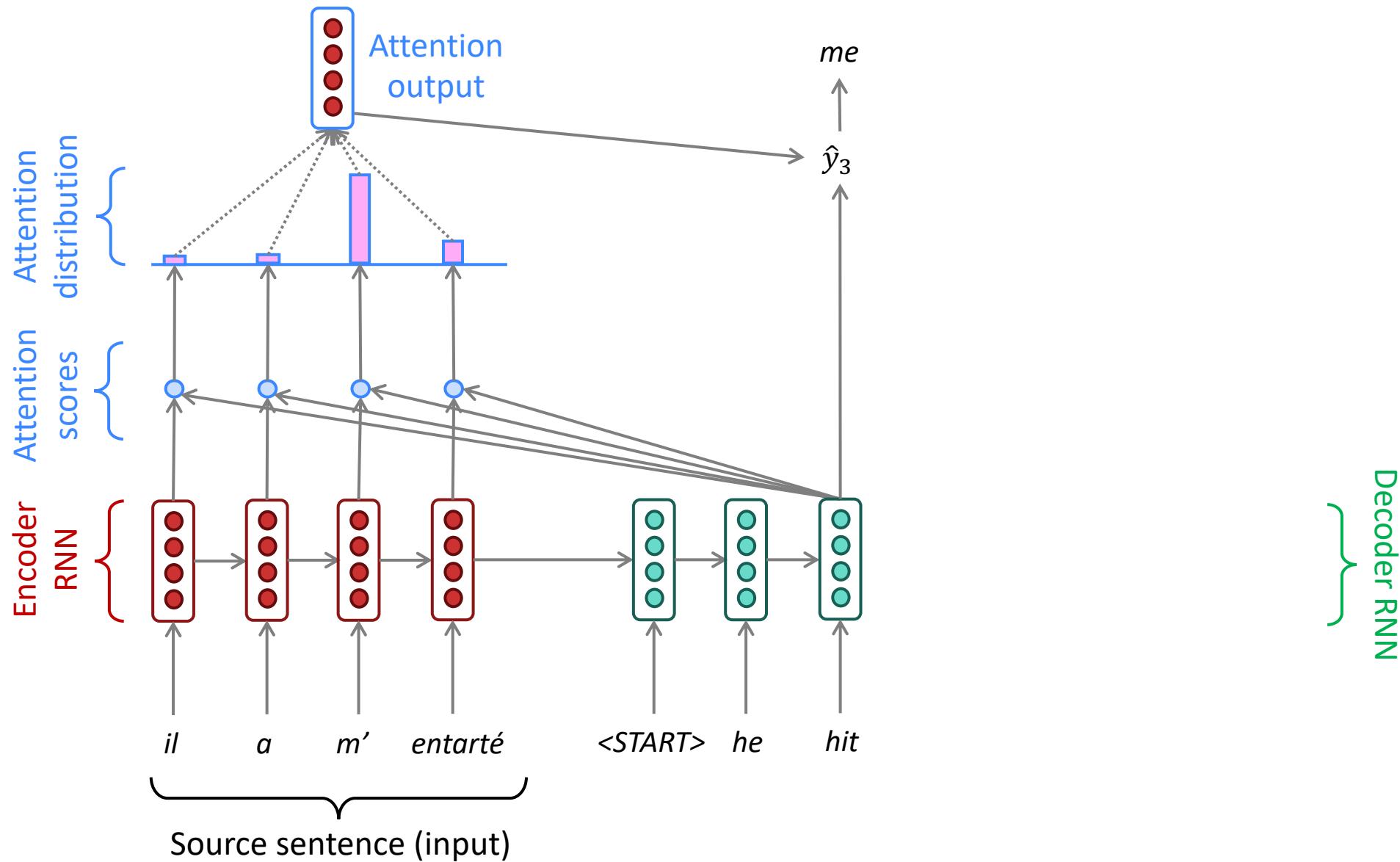
Sequence-to-sequence with attention



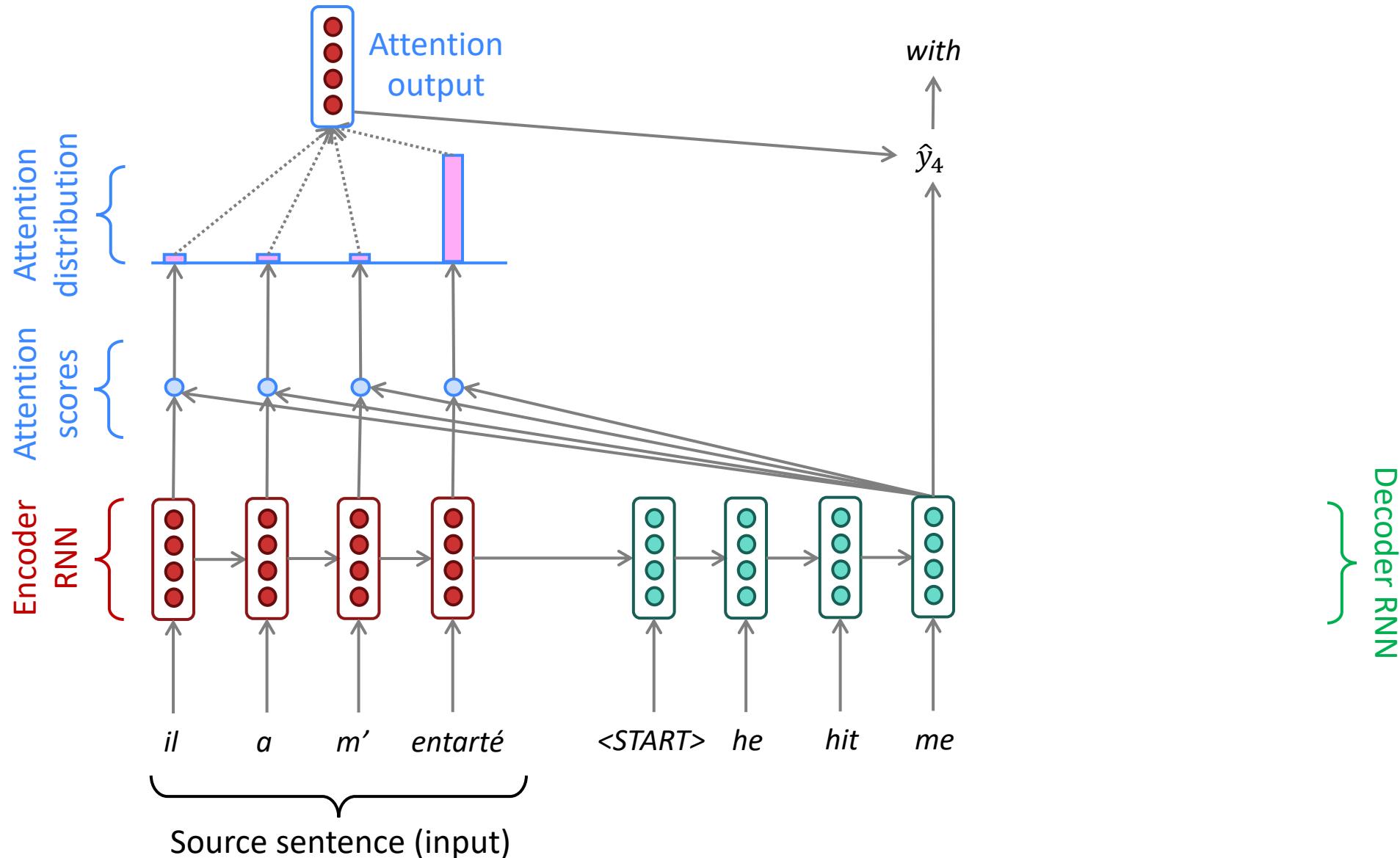
Sequence-to-sequence with attention



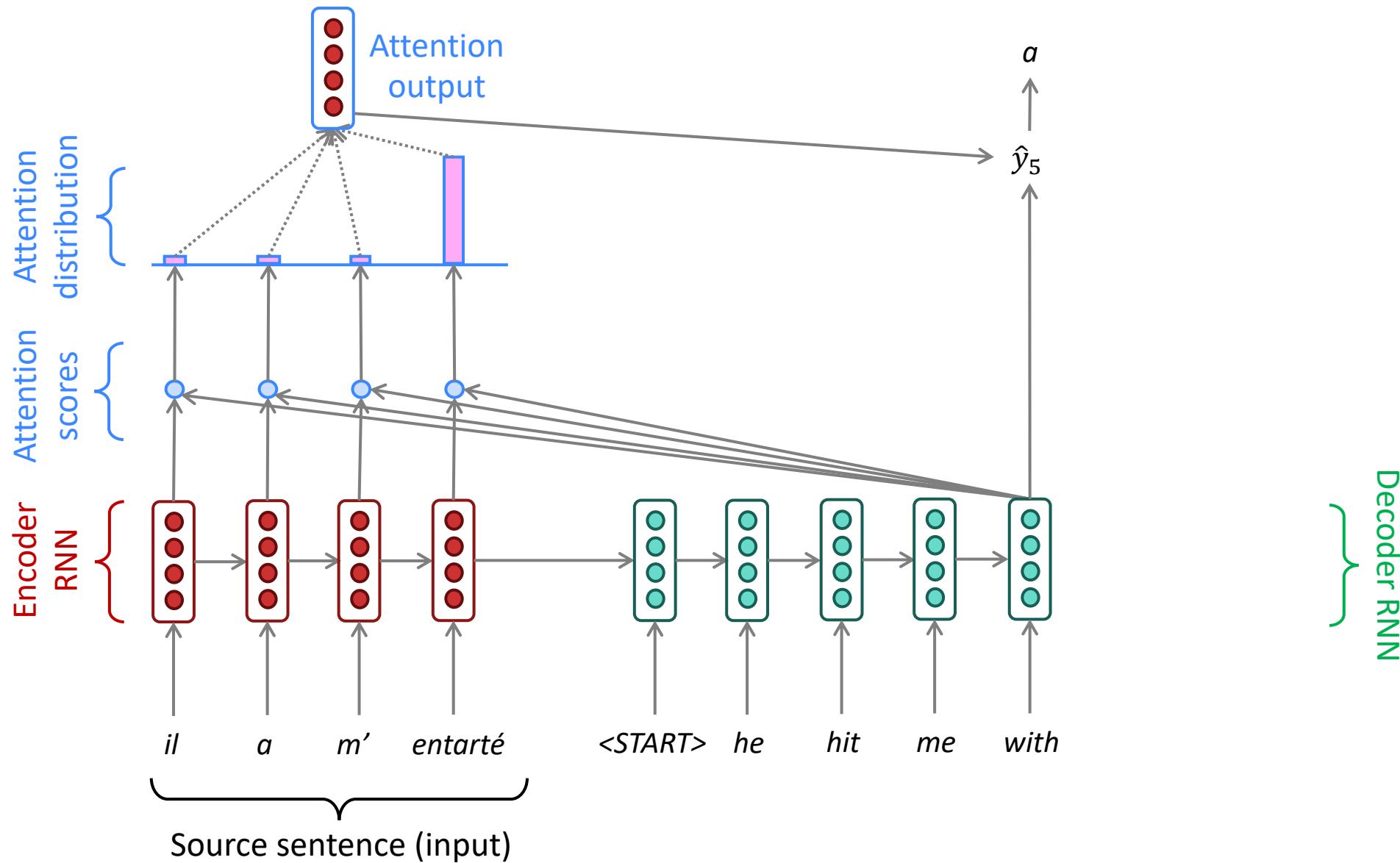
Sequence-to-sequence with attention



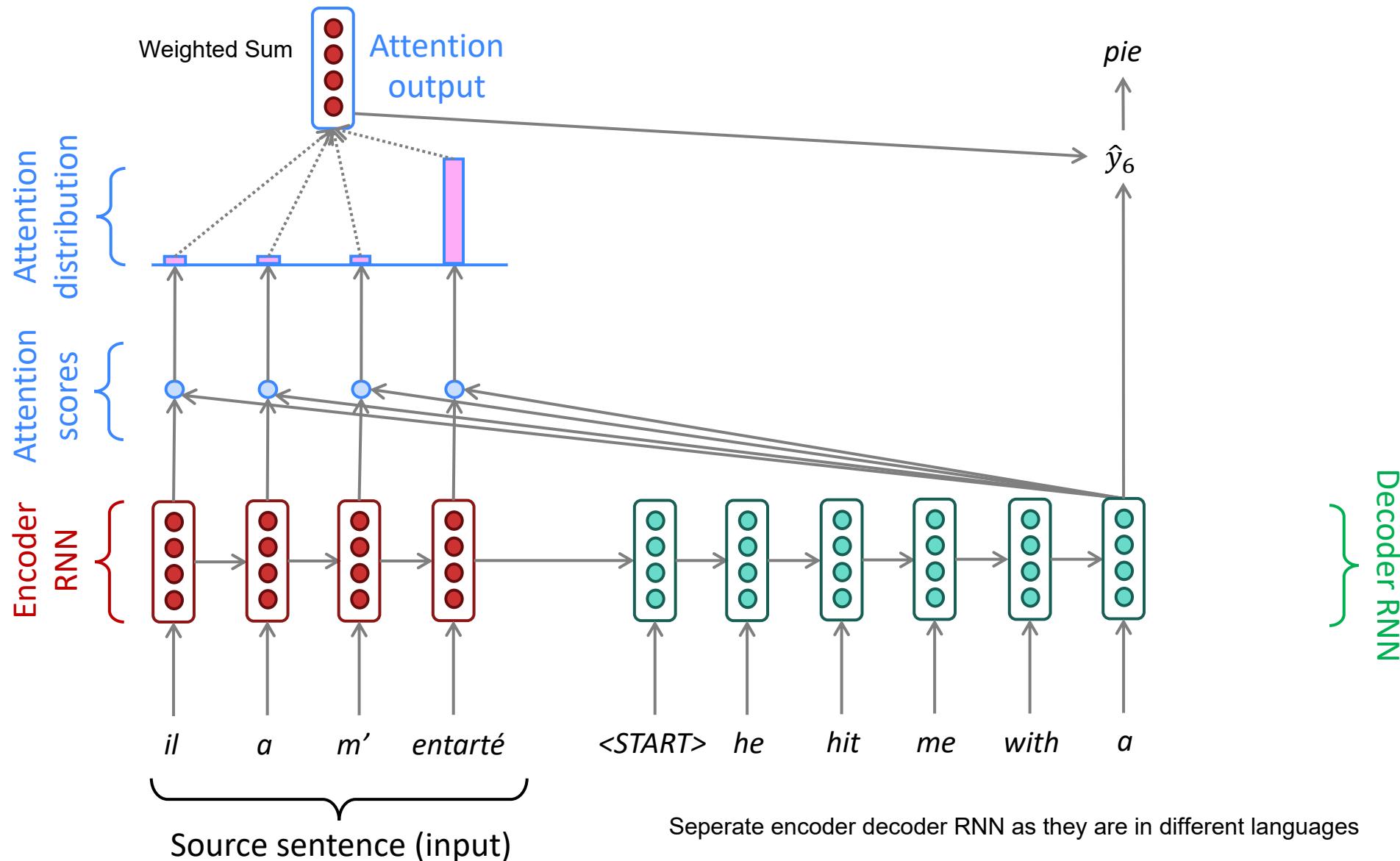
Sequence-to-sequence with attention



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

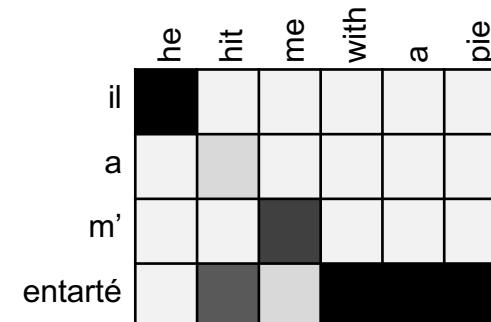
$$a_t = \sum_{i=1}^N \alpha_i^t h_i \in \mathbb{R}^h$$

- 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



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.

Attention is a good idea to get a vector out of a group of vectors instead of just maxpooling or mean

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).

Upshot:

Attention has become the powerful, flexible, general, way pointer and memory manipulation in deep learning models. A new idea from after 2010

There are *several* attention variants

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

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

There are multiple ways to do this

- 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 $e \in \mathbb{R}^N$ from $\mathbf{h}_1, \dots, \mathbf{h}_N \in \mathbb{R}^{d_1}$ and $\mathbf{s} \in \mathbb{R}^{d_2}$:

- Basic dot-product attention: $e_i = \mathbf{s}^T \mathbf{h}_i \in \mathbb{R}$
 - Note: this assumes $d_1 = d_2$
 - This is the version we saw earlier
- Multiplicative attention: $e_i = \mathbf{s}^T \mathbf{W} \mathbf{h}_i \in \mathbb{R}$
 - Where $\mathbf{W} \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix

Matrix of learnable parameters
What parts of s and h to pay attention to.

Too many parameters
so, Reduced rank multiplicative attention: Uses low rank skinny matrices W:U.v
- Additive attention: $e_i = \mathbf{v}^T \tanh(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}) \in \mathbb{R}$
 - 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 and $\mathbf{v} \in \mathbb{R}^{d_3}$ 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 models: encoder and decoder)
- Attention is a way to *focus on particular parts* of the input
 - Improves sequence-to-sequence a lot!

