Natural Language Processing

AI51701/CSE71001

Lecture 12

10/26/2023

Instructor: Taehwan Kim

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

People rely on machine translation!

People rely on machine translation!

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*

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

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

$$= \text{argmax}_y \underbrace{P(x|y)}_{\text{Translation Model}} \underbrace{P(y)}_{\text{Language Model}}$$

Translation Model

Models how words and phrases should be translated (*fidelity*). Learnt from parallel data.

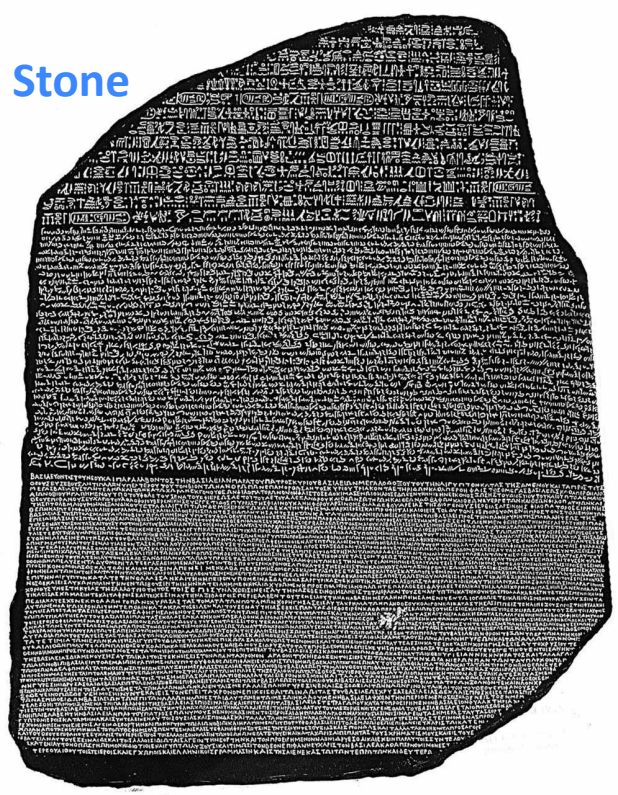
Language Model

Models how to write good English (*fluency*). Learnt from monolingual data.

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

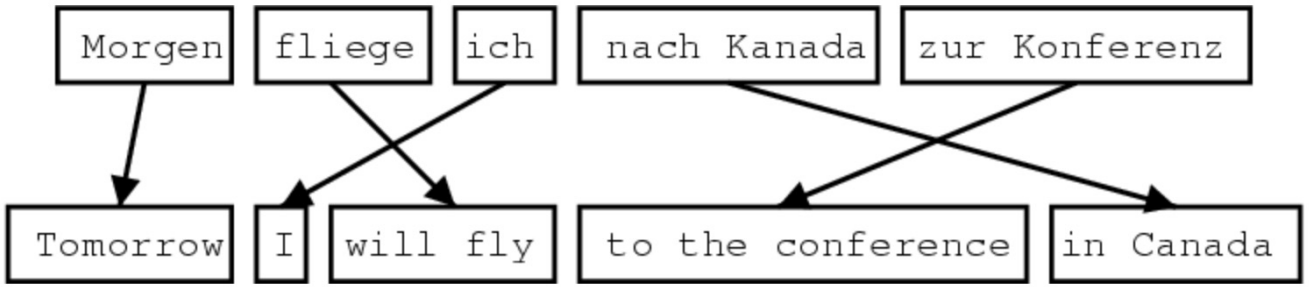
Demotic

Ancient Greek

Learning alignment for SMT

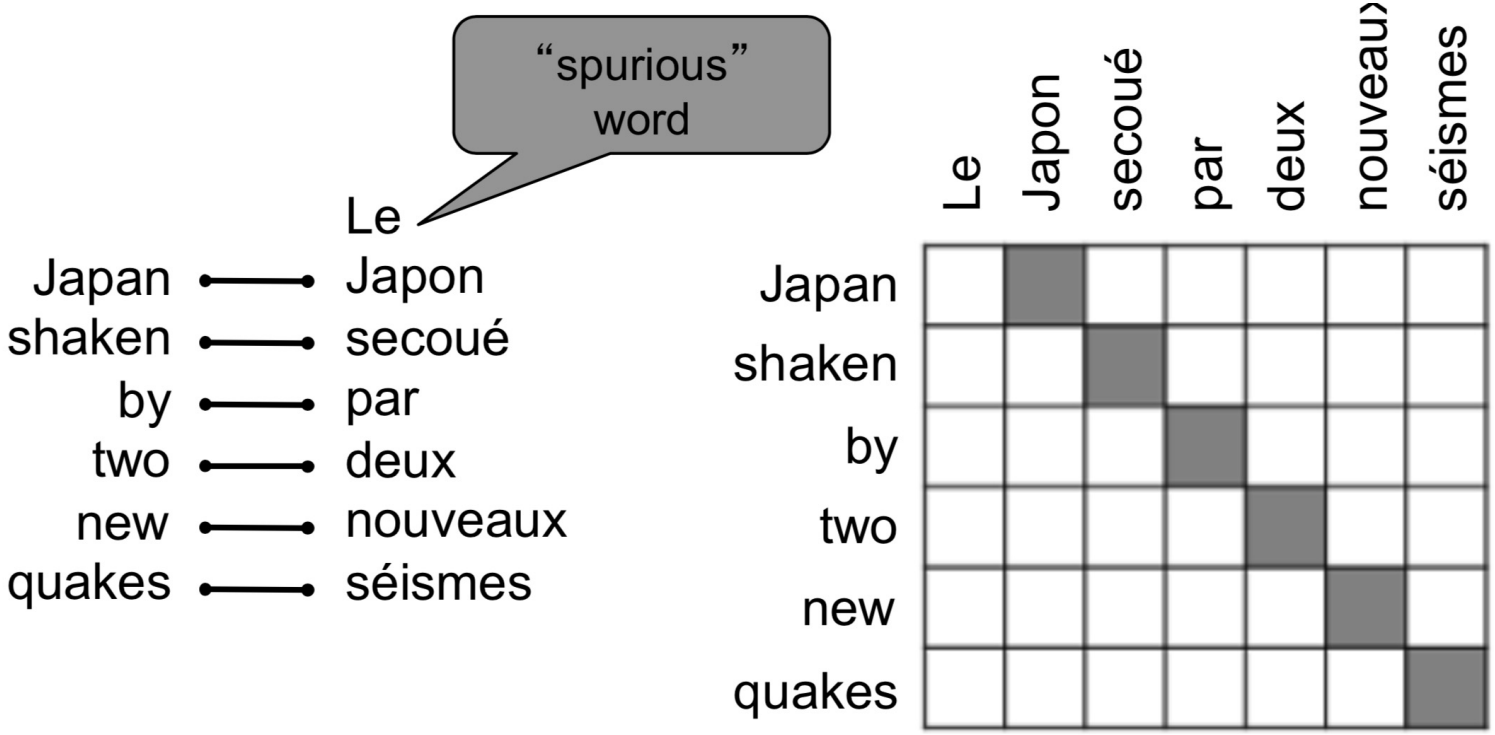
- Question: How to learn translation model $P(x|y)$ from the parallel corpus?
- Break it down further: Introduce latent *a* variable into the model: $P(x, a|y)$

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



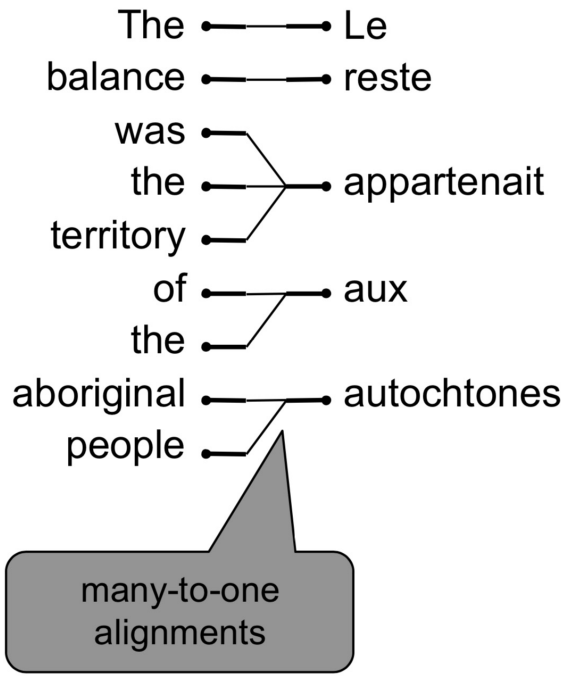
What is alignment?

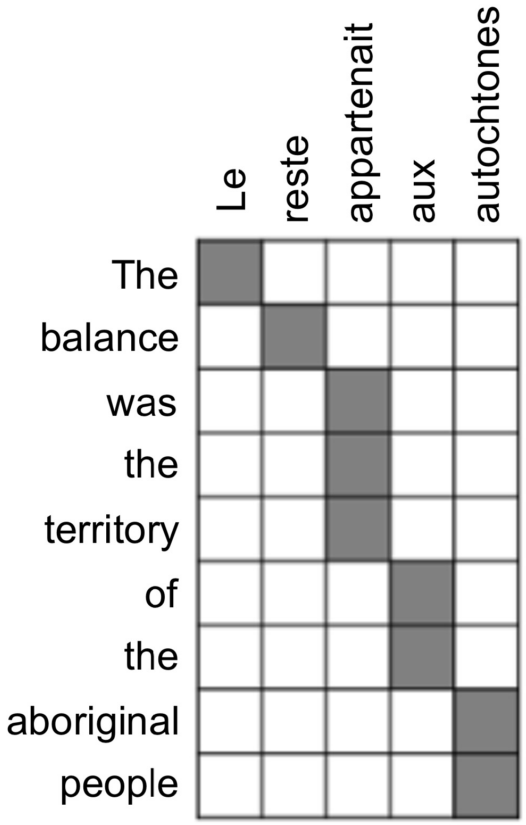
- Alignment is the **correspondence between particular words** in the translated sentence pair.
  - **Typological differences** between languages lead to complicated alignments!
  - Note: Some words have **no counterpart**



Alignment is complex

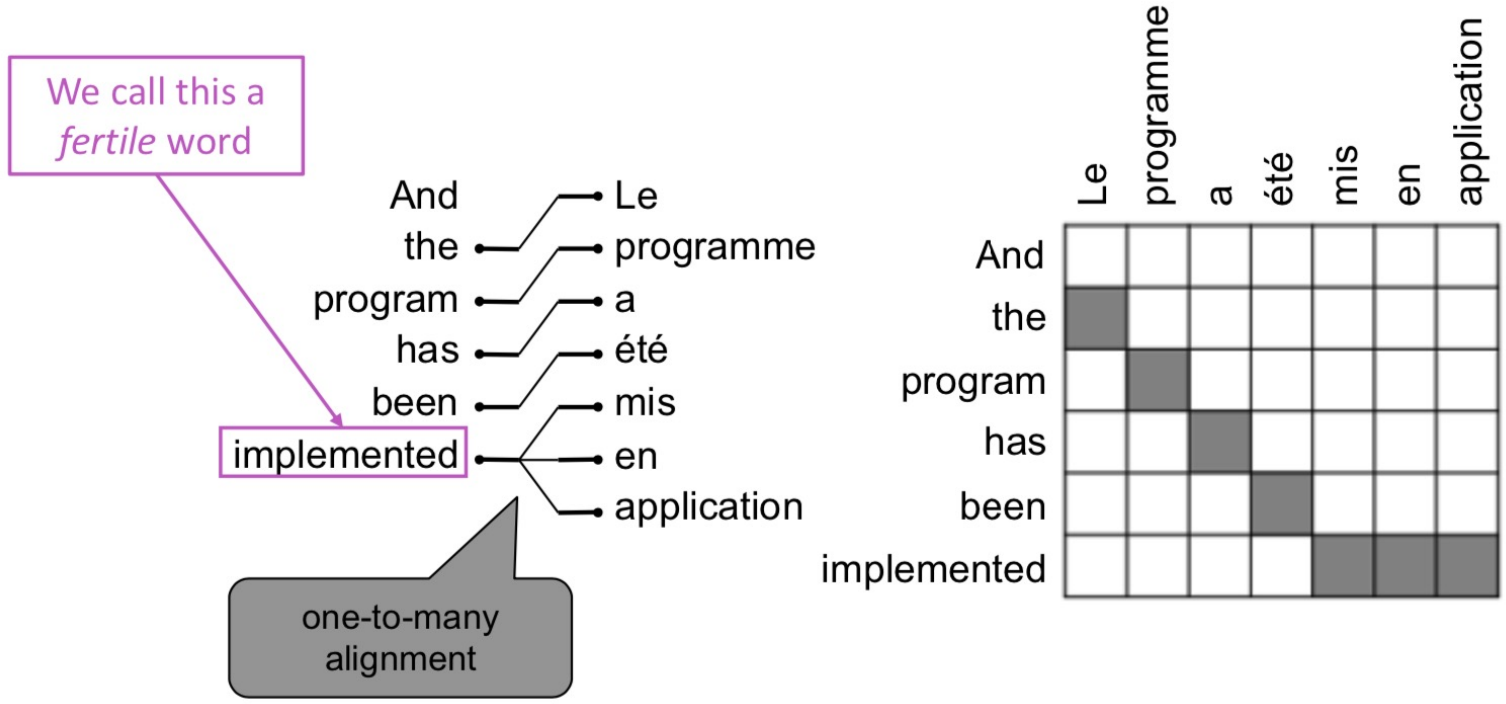
- Alignment can be **many-to-one**





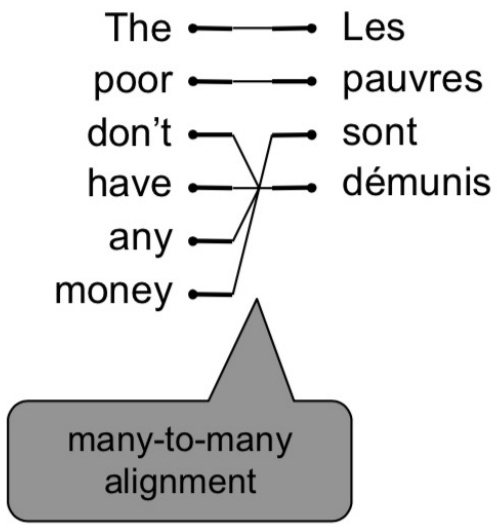
Alignment is complex

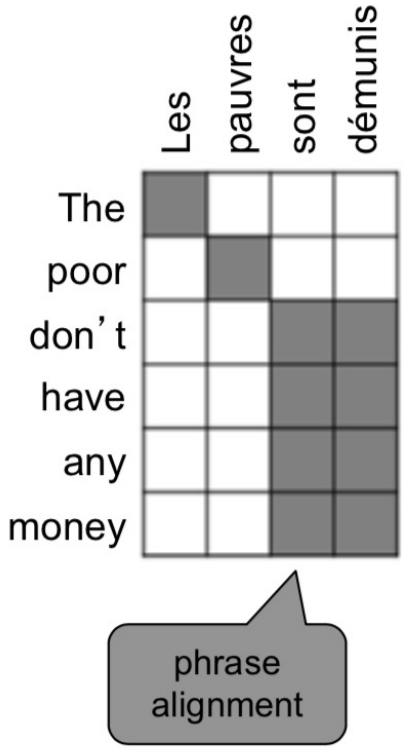
- Alignment can be **one-to-many**



Alignment is complex

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

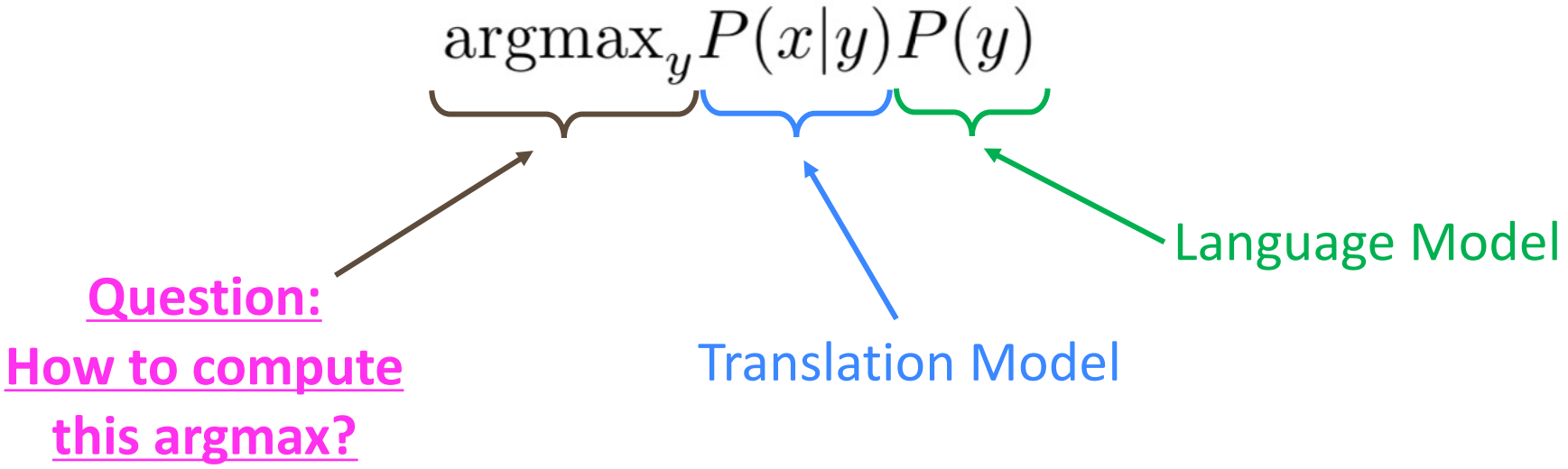




Alignment is complex

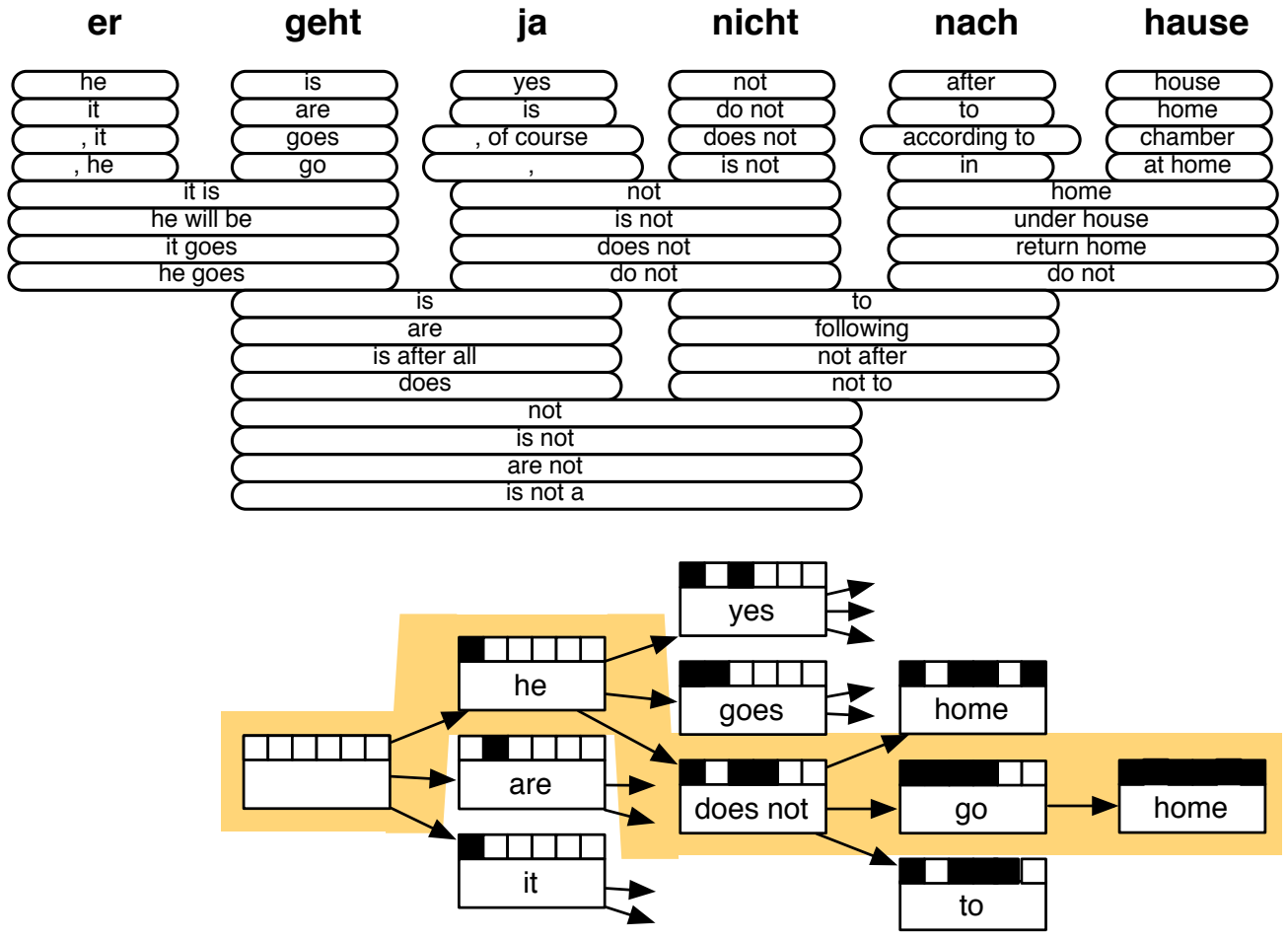
- We learn $P(x, a|y)$ as a combination of several factors, including:
  - Probability of particular words aligning (also depends on position in sent)
  - Probability of particular words having a particular fertility (number of corresponding words)
  - etc.
- Alignments *a* are **latent variables**: 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

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



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

Statistical Machine Translation

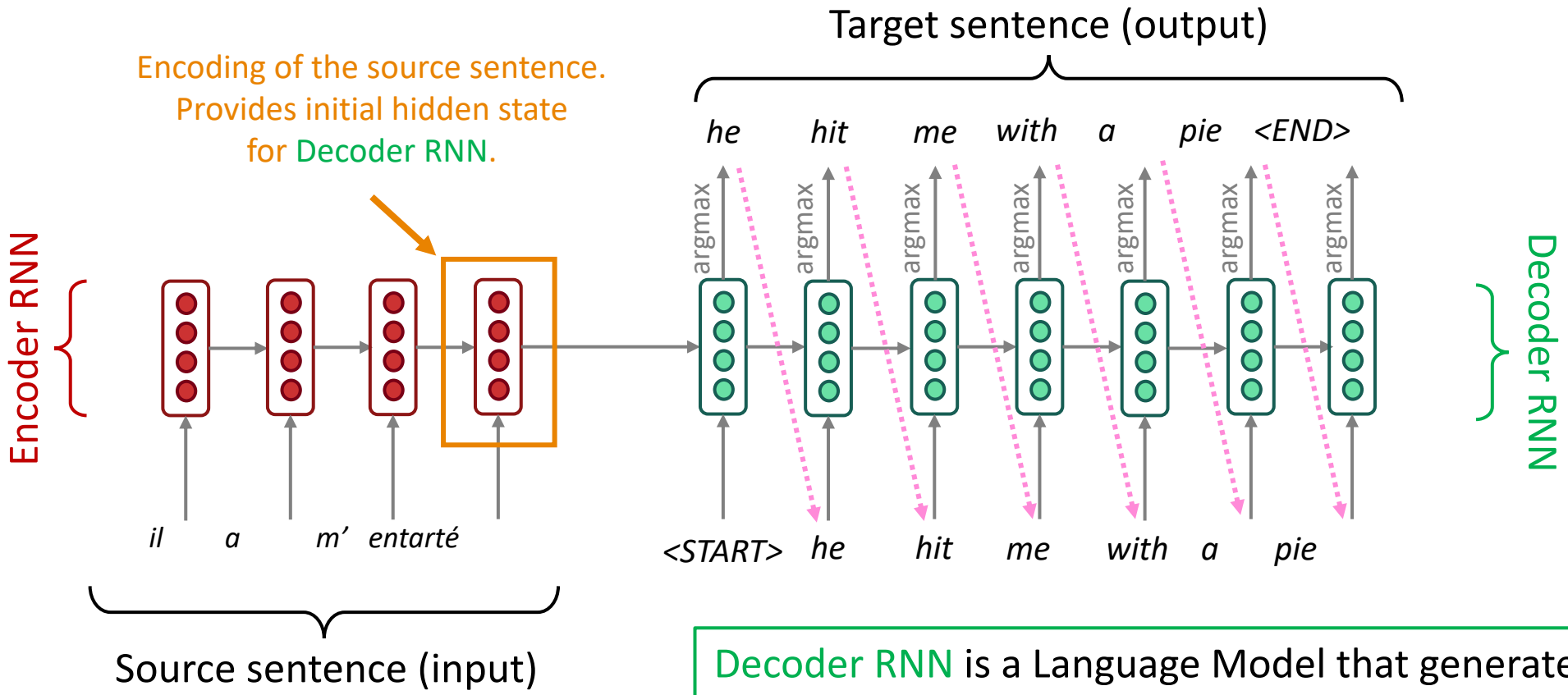
- SMT was a **huge research field**
- The best systems were **extremely complex**
  - Hundreds of important details we haven't mentioned here
- Systems had many **separately-designed subcomponents**
  - Lots of **feature engineering**
    - Need to design features to capture particular language phenomena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

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



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

From Stanford 224n course

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." *Advances in neural information processing systems* 27 (2014).

Sequence-to-sequence is versatile!

- The general notion here is an **encoder-decoder** model
  - One neural network takes input and produces a neural representation
  - Another network produces output based on that neural representation
  - If the input and output are sequences, we call it a seq2seq model
- Sequence-to-sequence is useful for **more than just MT**
- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

Neural Machine Translation (NMT)

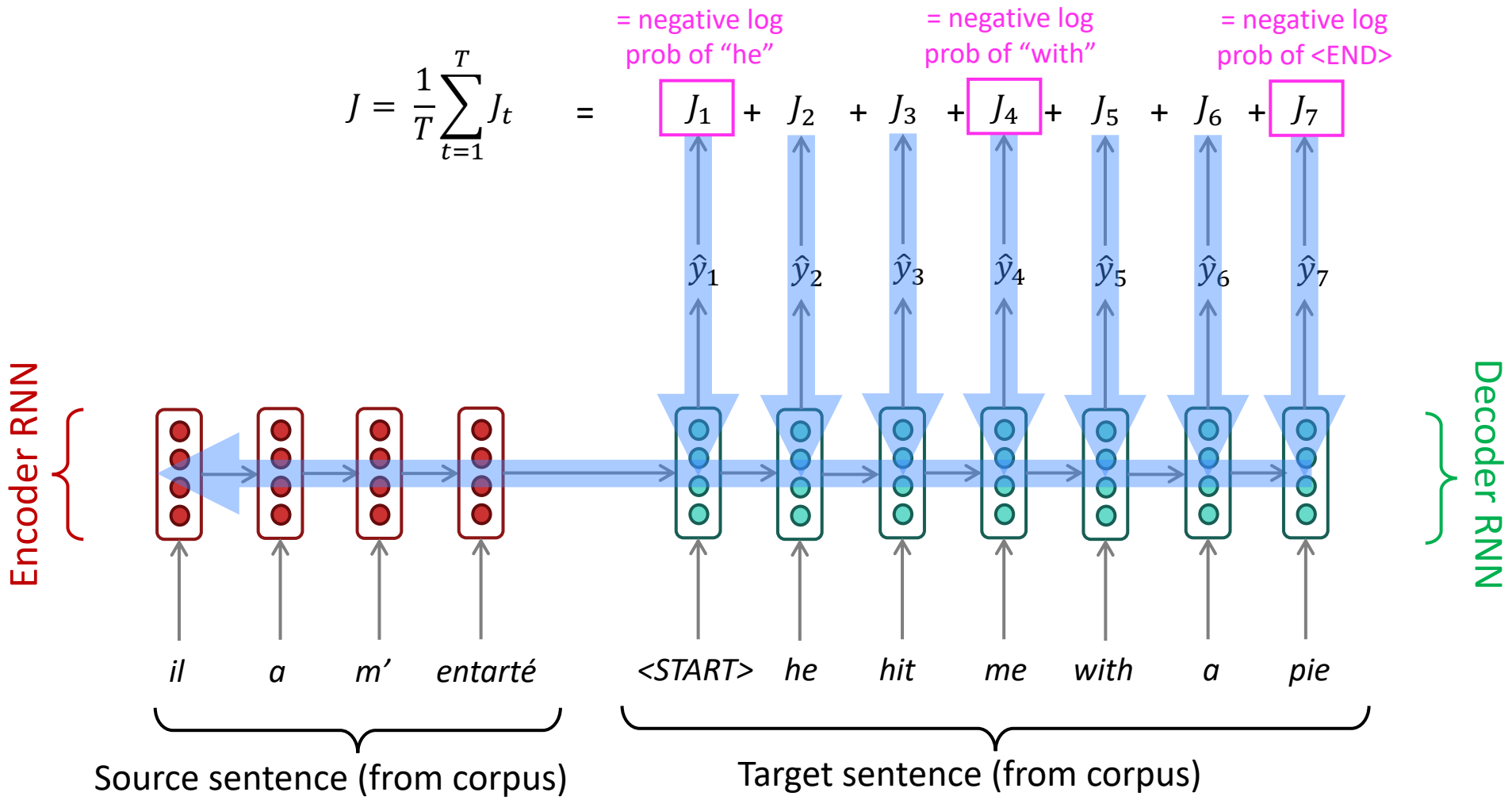
- The **sequence-to-sequence** model is an example of a **Conditional Language Model**
  - Language Model because the decoder is predicting the next word of the target sentence *y*
  - Conditional because its predictions are also conditioned on the source sentence *x*
- NMT directly calculates $P(y|x)$:

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

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

- Question: How to train an NMT system?
- (Easy) Answer: Get a big parallel corpus...
  - But there is now exciting work on "unsupervised NMT", data augmentation, etc.

Training a Neural Machine Translation system

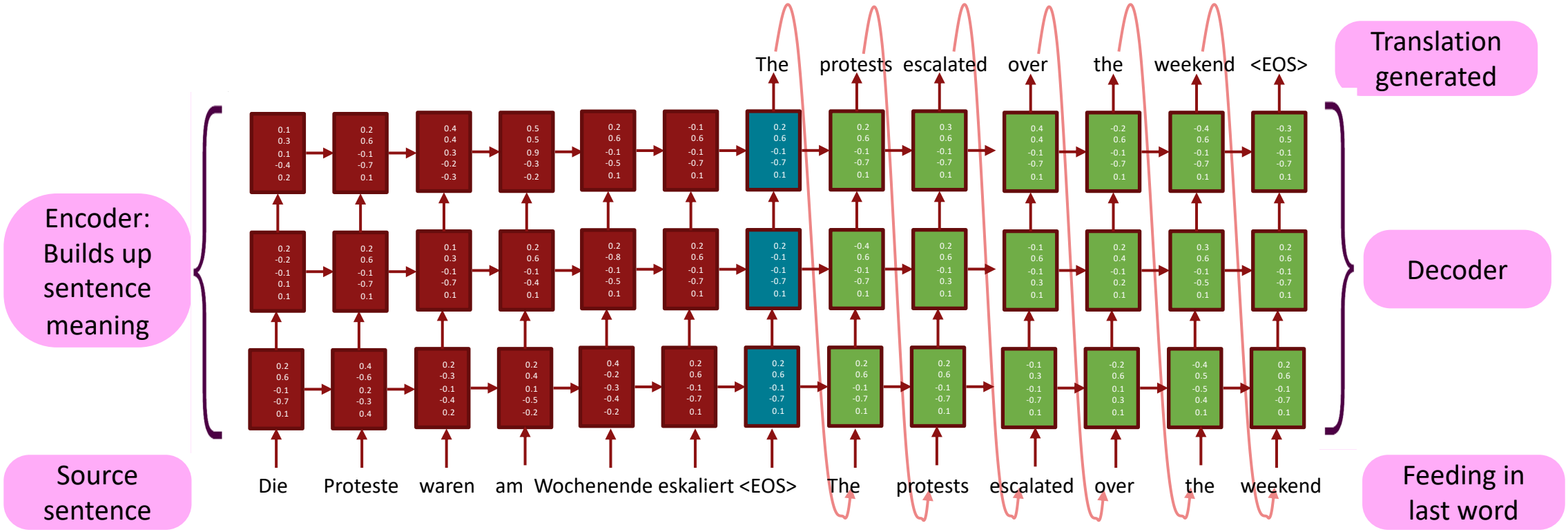


Seq2seq is optimized as a **single system.** Backpropagation operates "*end-to-end*".

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

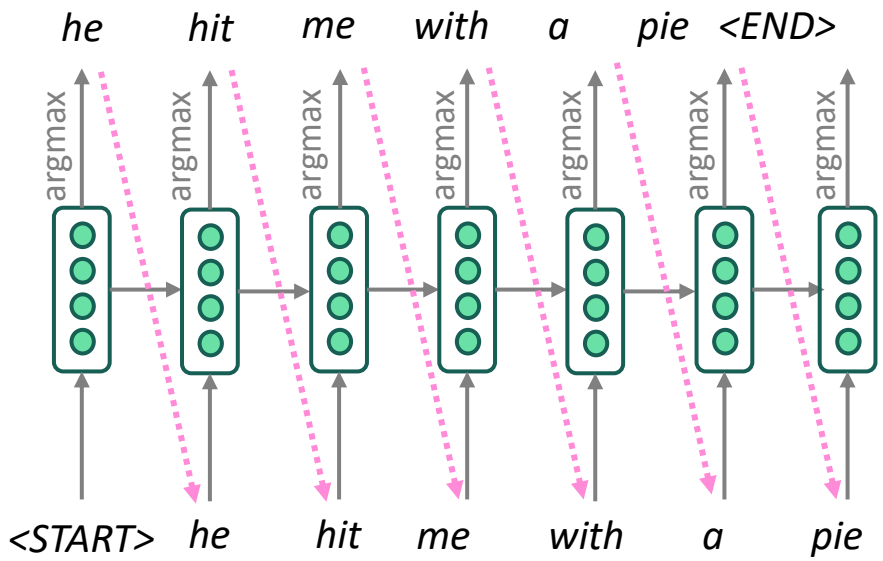
- Multi-layer or stacked RNNs allow the network to compute **more complex representations**
  - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.

- **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?

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

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

- 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 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, in NMT)

- 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

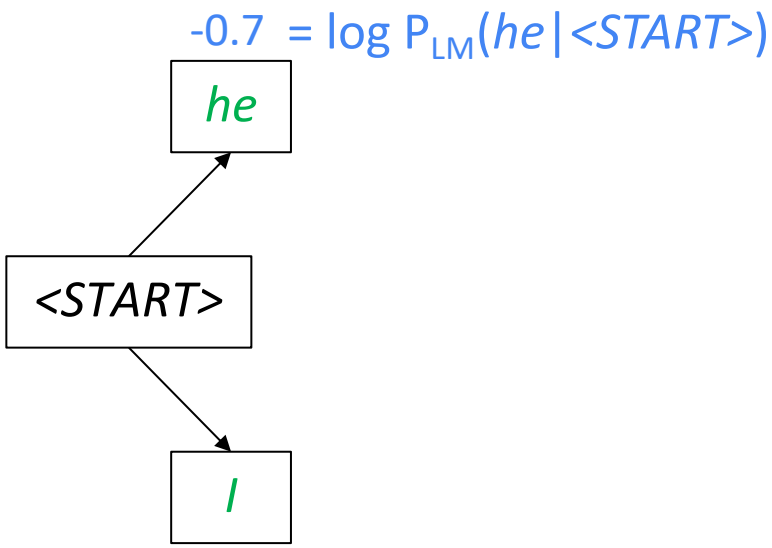
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

<START>

Calculate prob dist of next word

Beam search decoding: example

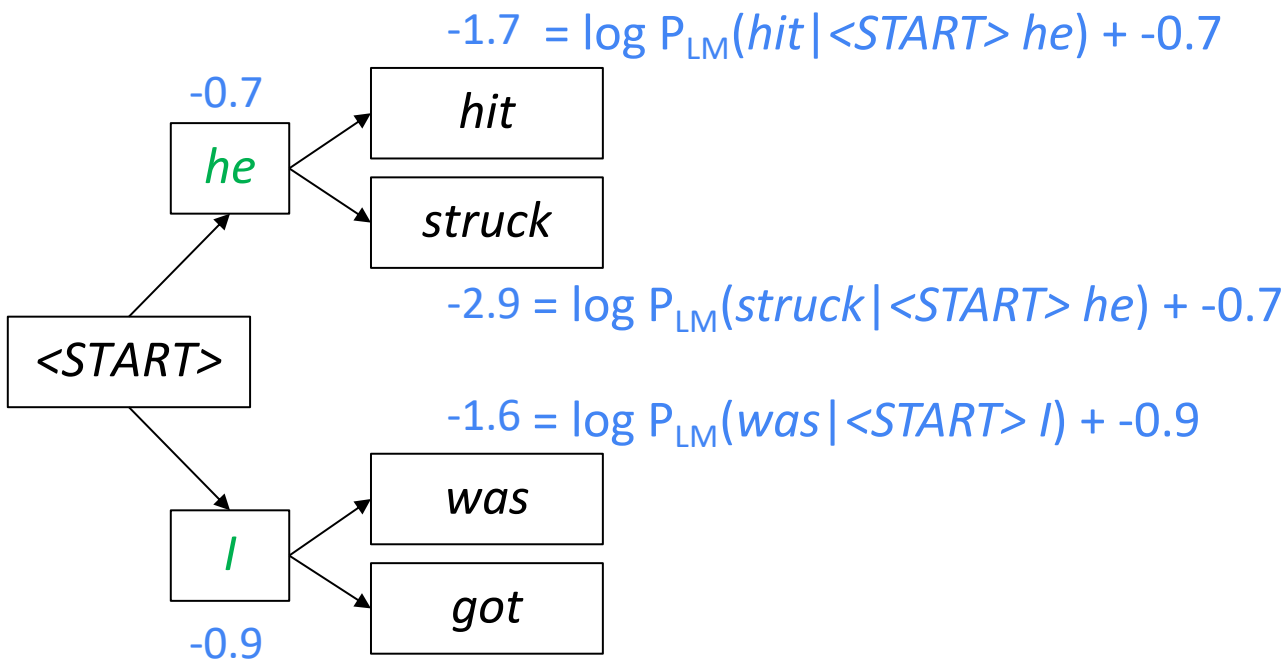
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Take top *k* words and compute scores

Beam search decoding: example

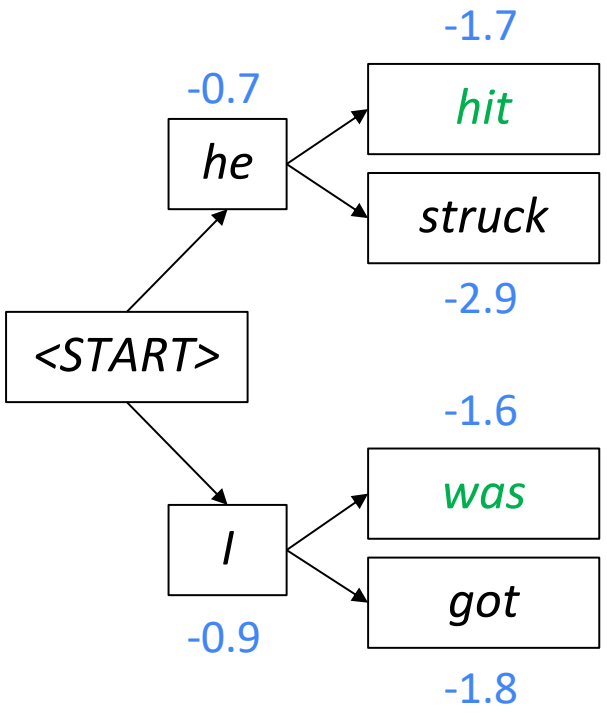
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

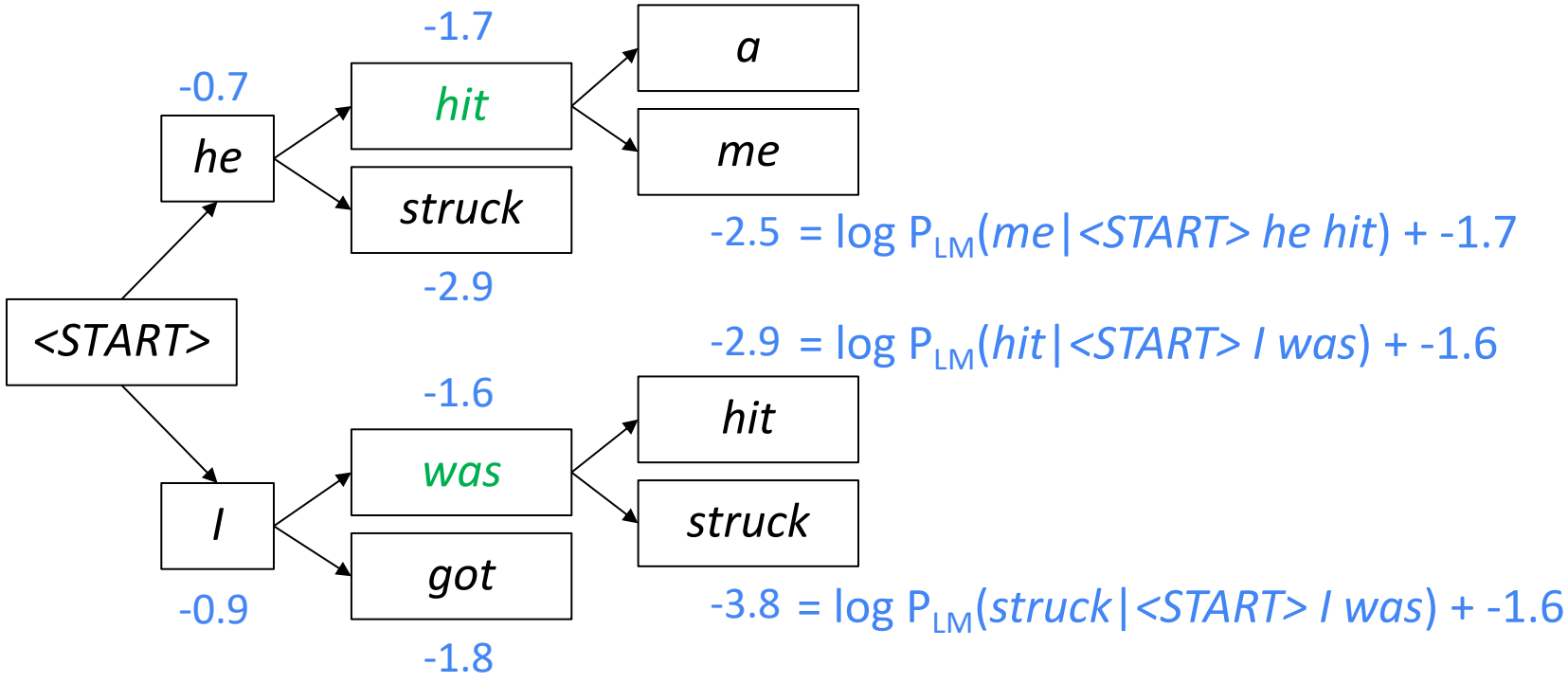
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

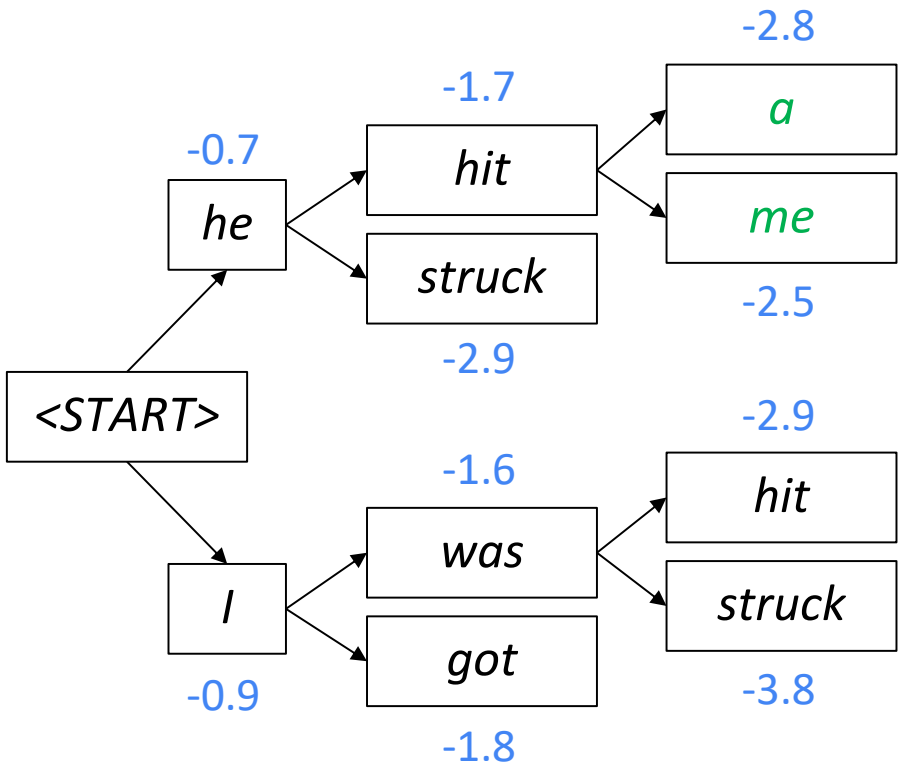
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

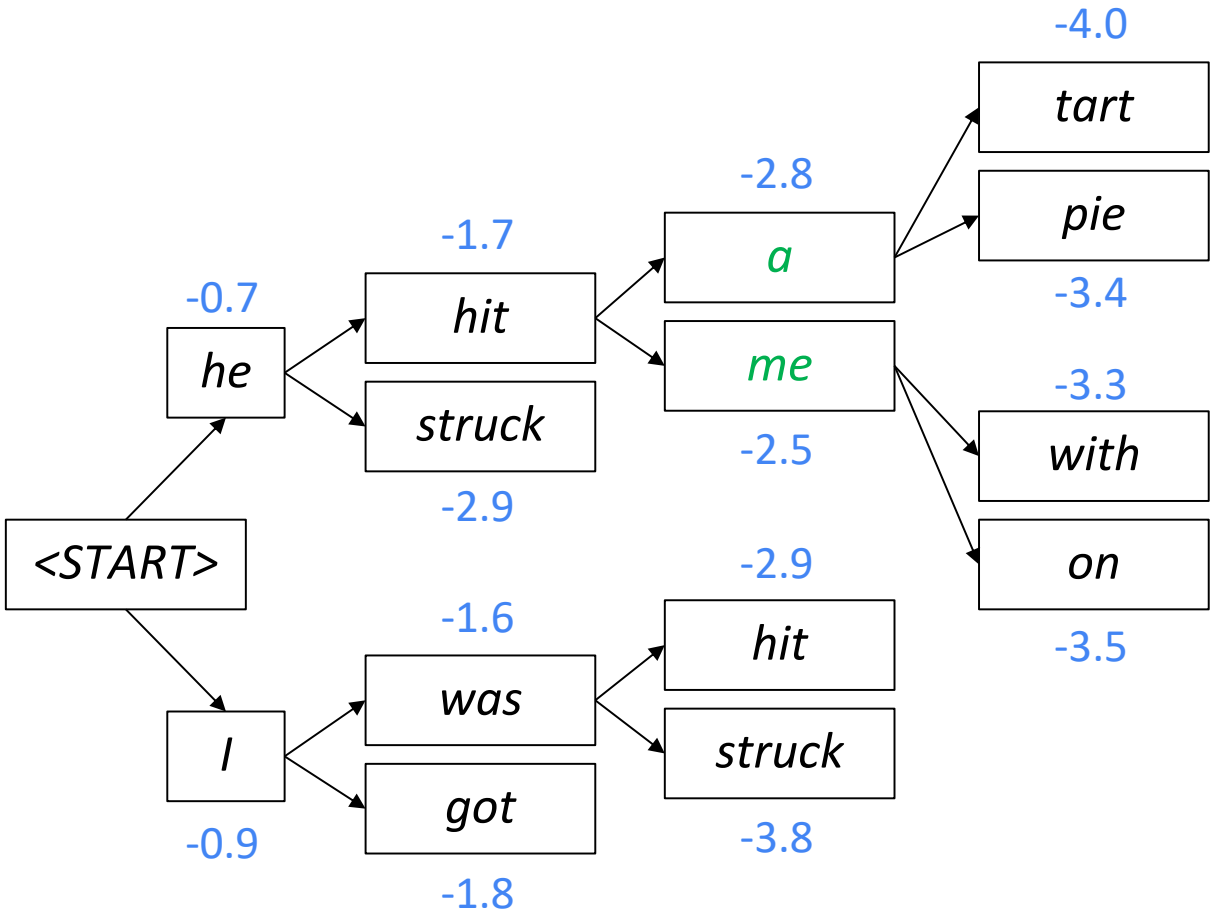
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

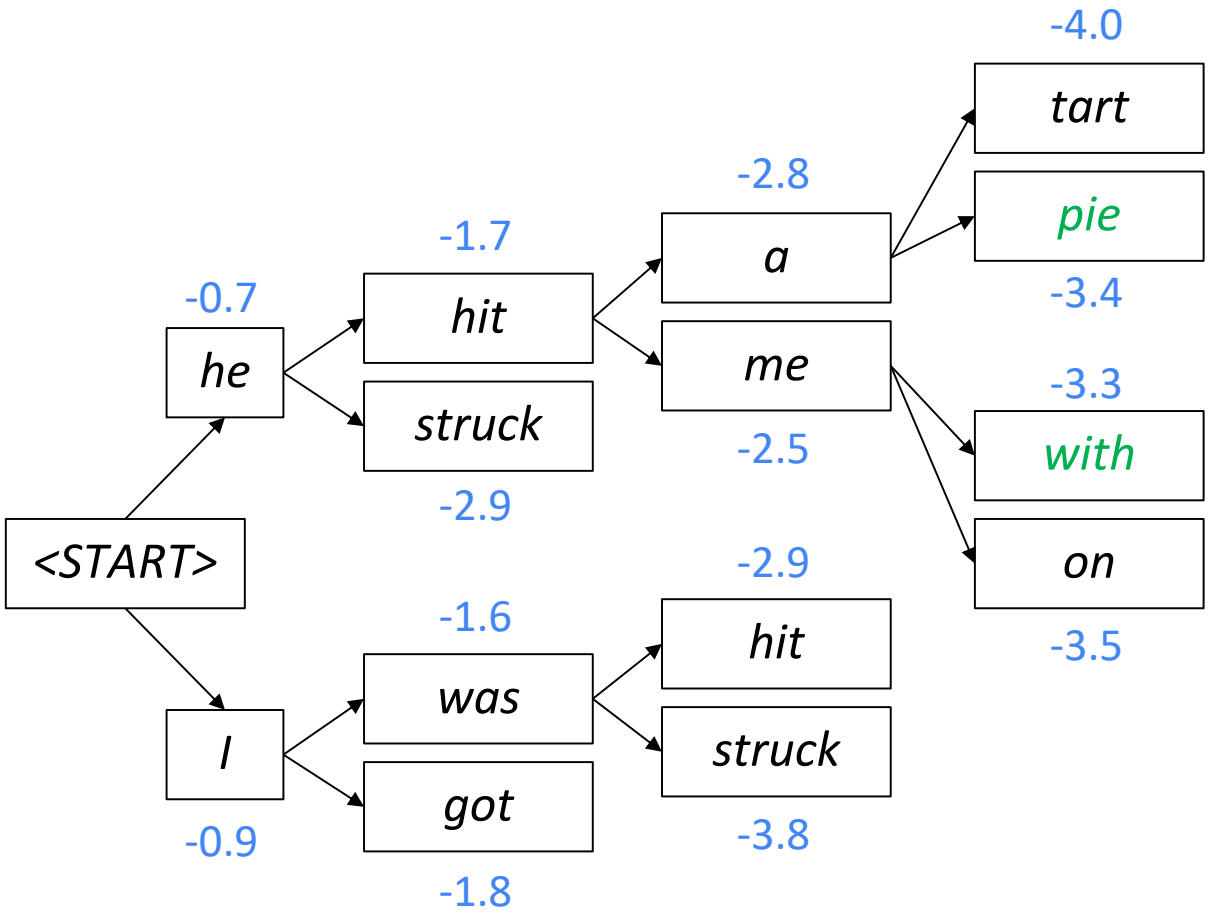
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

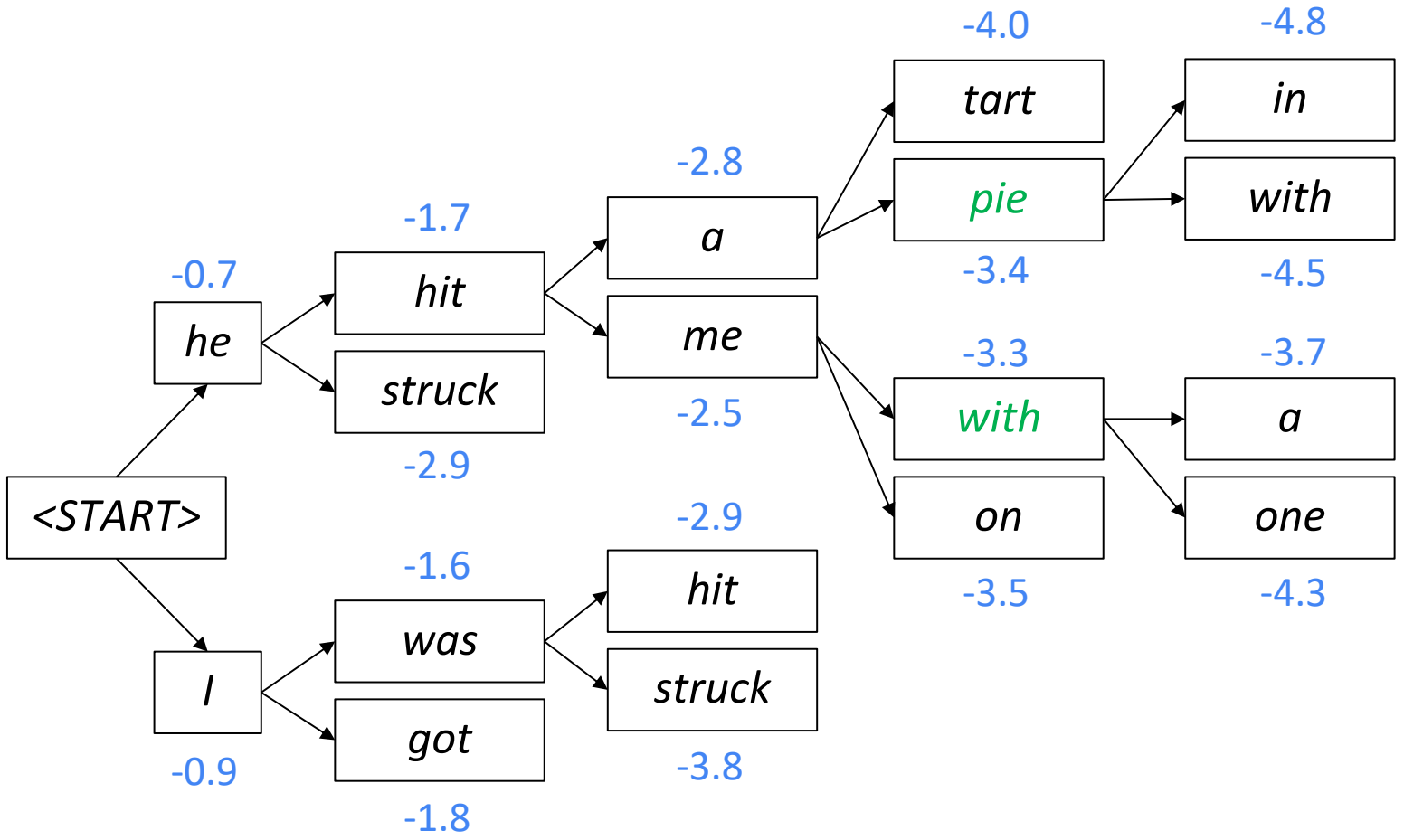
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

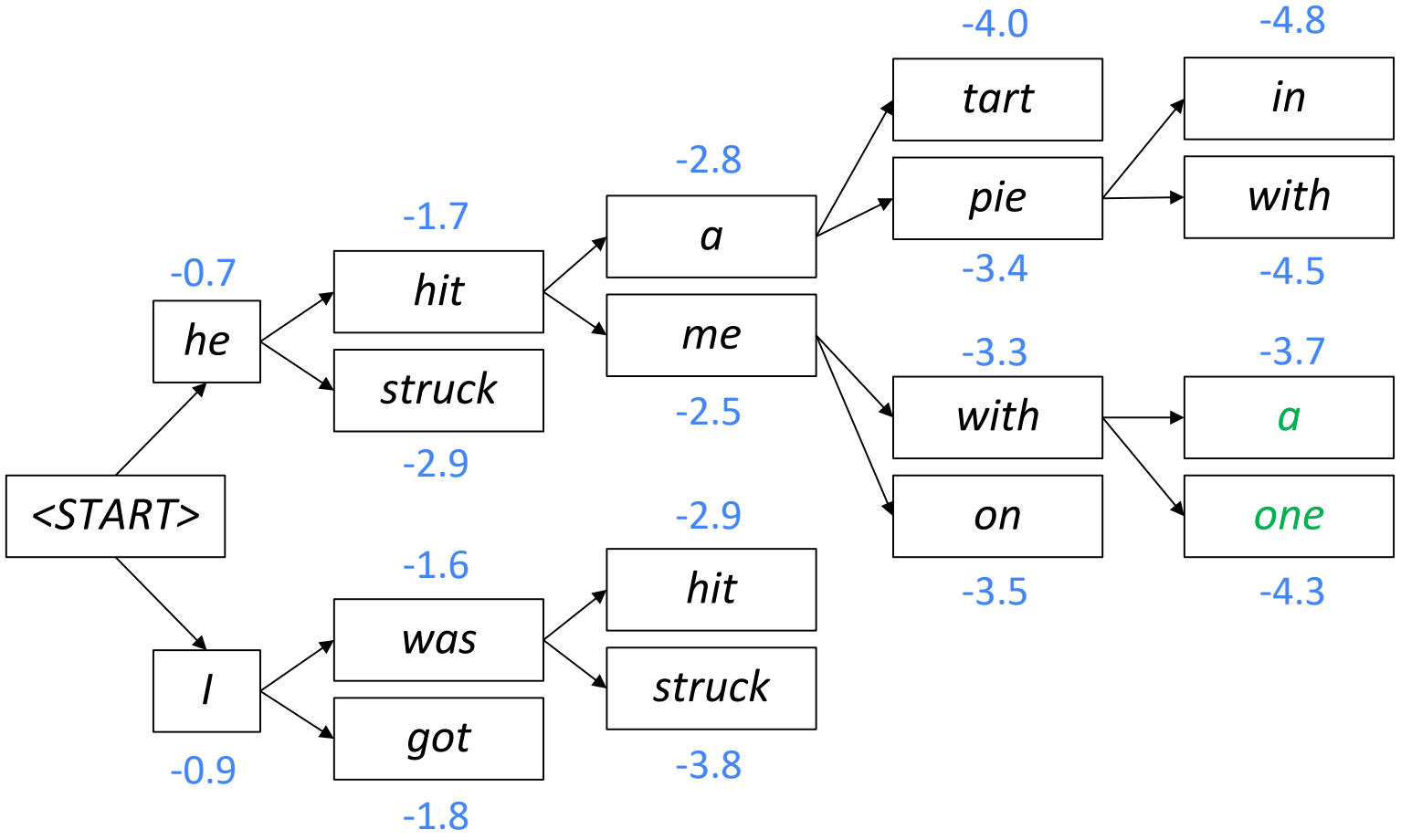
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

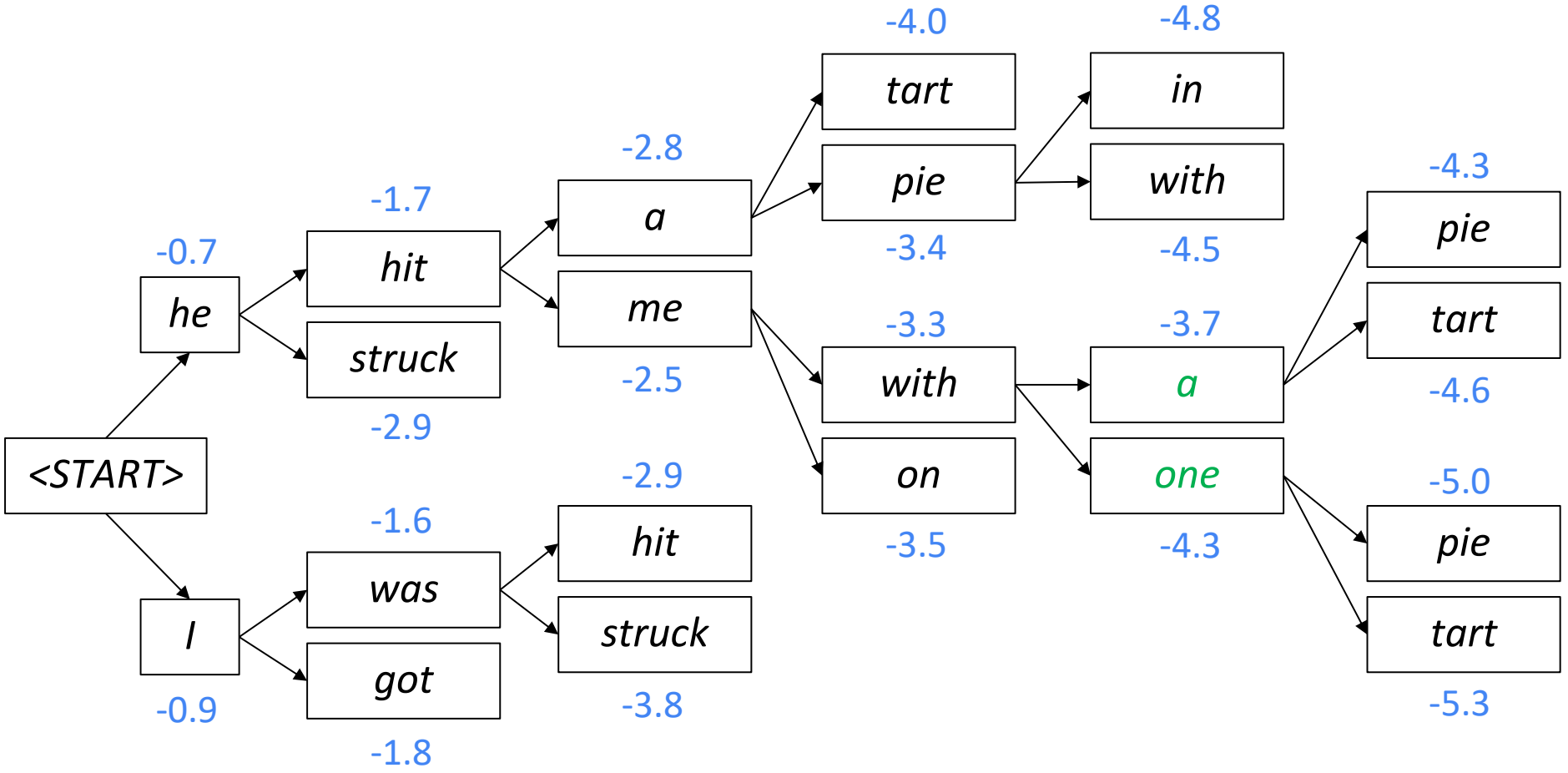
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

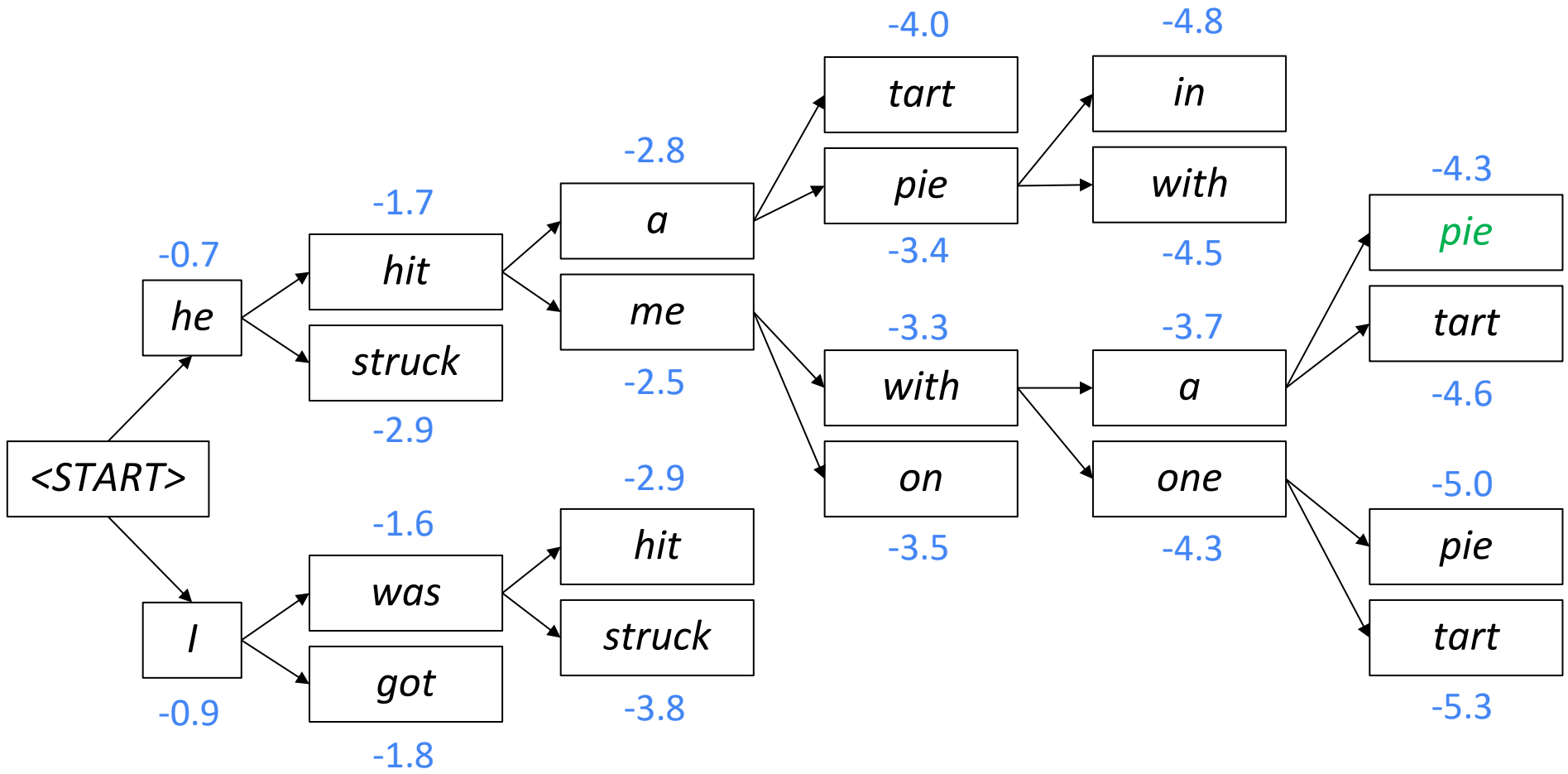
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

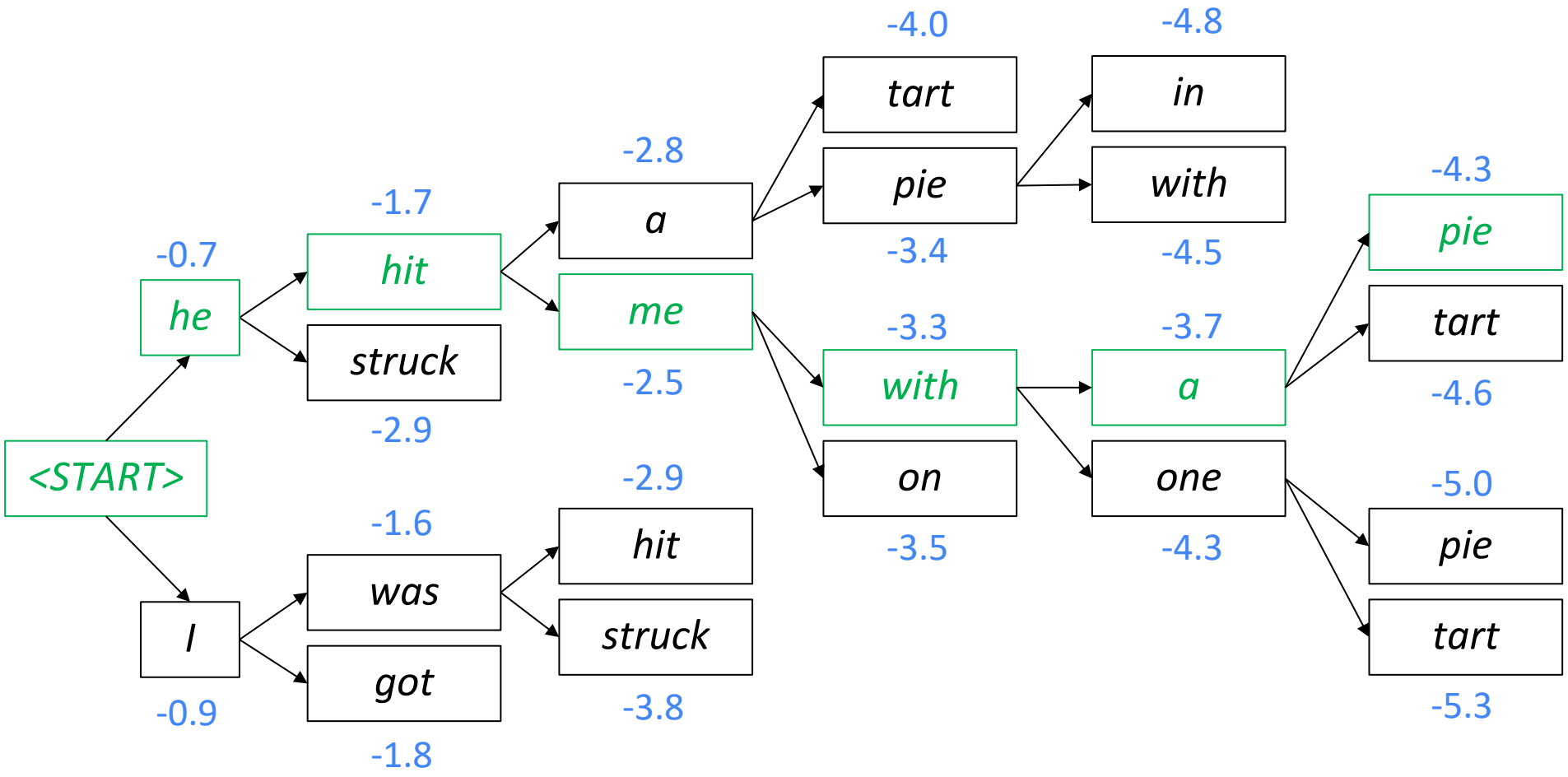
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

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

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

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

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!

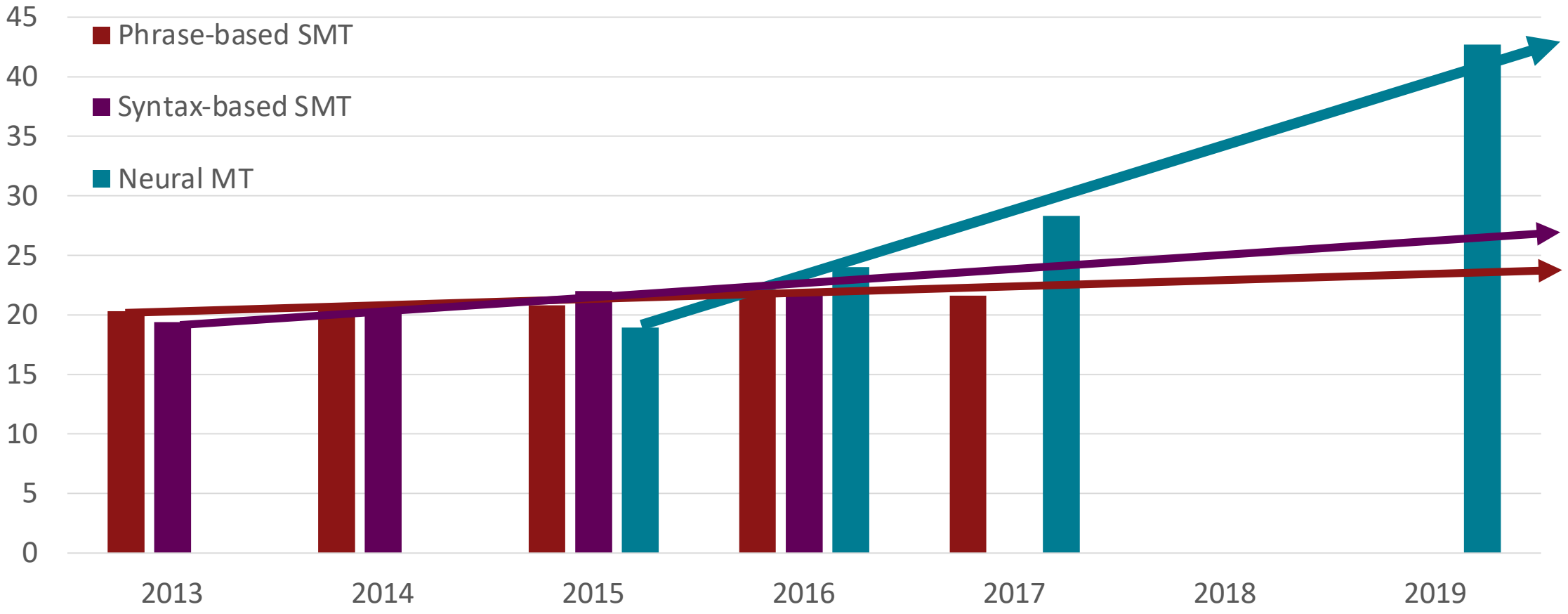
How do we evaluate Machine Translation?

- **BLEU** (**Bil**ingual **E**valuation **U**nderstudy)
  - 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

Source: ”BLEU: a Method for Automatic Evaluation of Machine Translation", Papineni et al, 2002. http://aclweb.org/anthology/P02-1040

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

  - 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

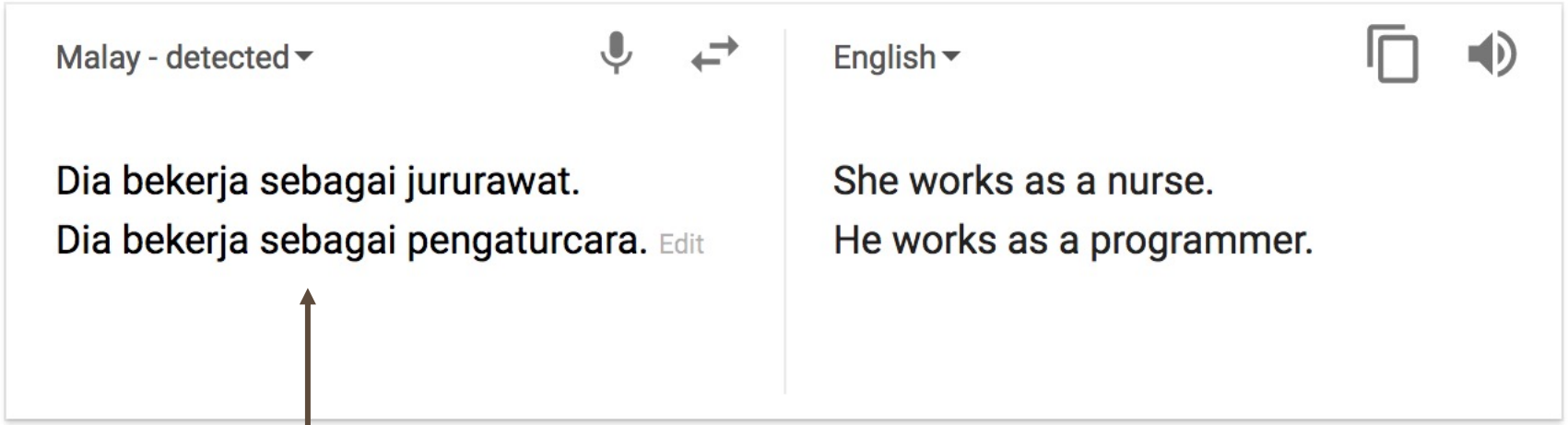
So, is Machine Translation solved?

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

?

So, is Machine Translation solved?

- Nope!
- NMT picks up biases in training data



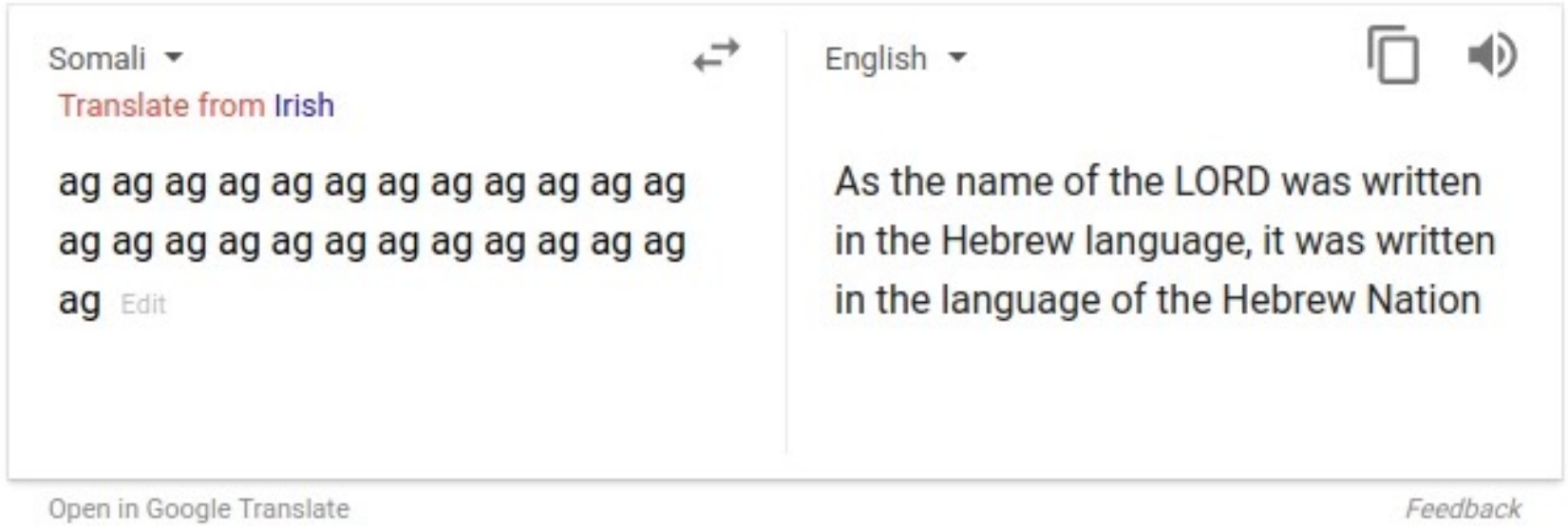
Didn’t specify gender

Source: https://hackernoon.com/bias-sexist-or-this-is-the-way-it-should-be-ce1f7c8c683c

So, is Machine Translation solved?

- Nope!
- **Uninterpretable** systems do **strange things**

(But this problem might have been fixed in Google Translate by 2021.)



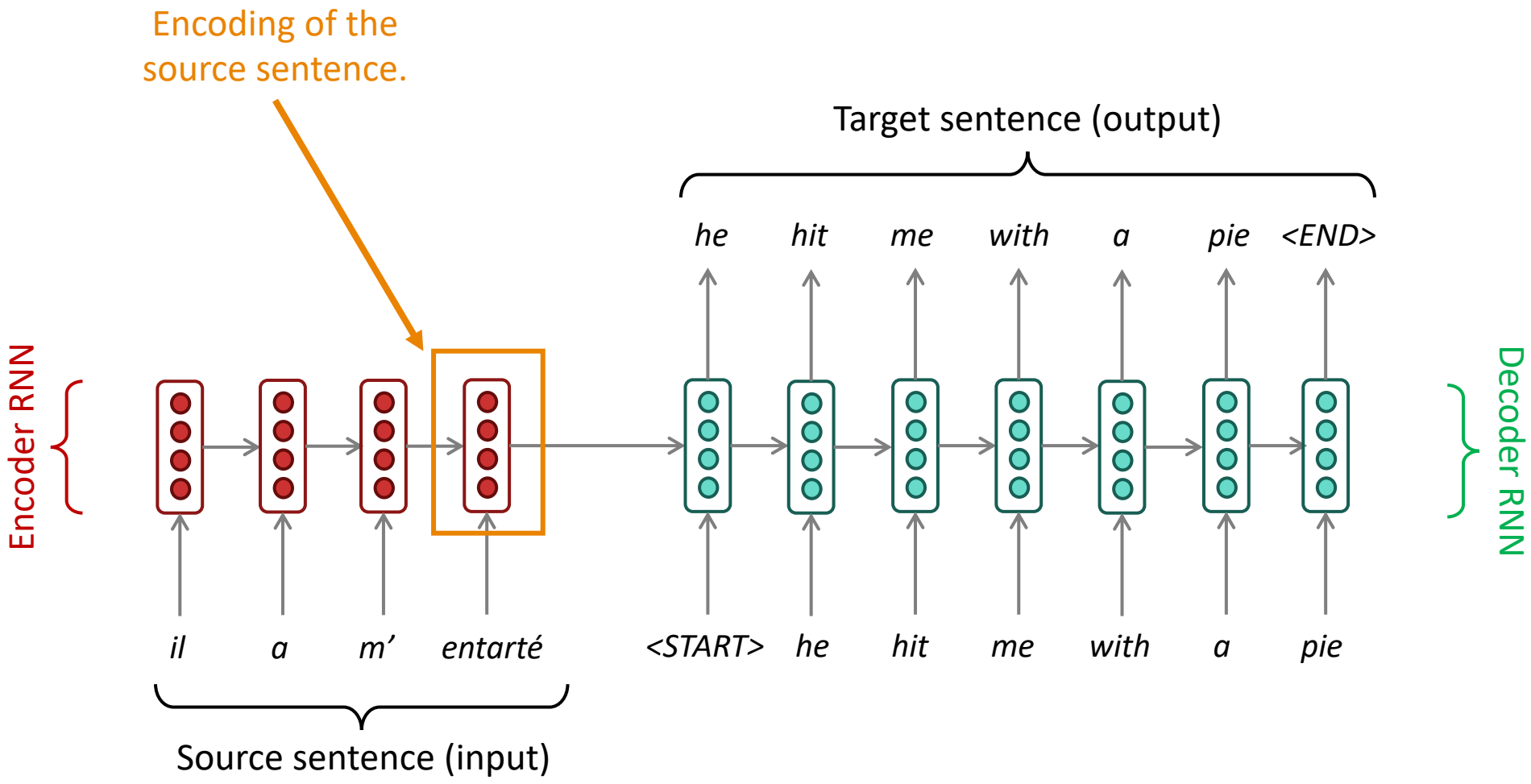
Picture source: https://www.vice.com/en_uk/article/j5npeg/why-is-google-translate-spitting-out-sinister-religious-prophecies
Explanation: https://www.skynettoday.com/briefs/google-nmt-prophecies

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
  - 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 next **one improvement** so integral that it is the new vanilla...

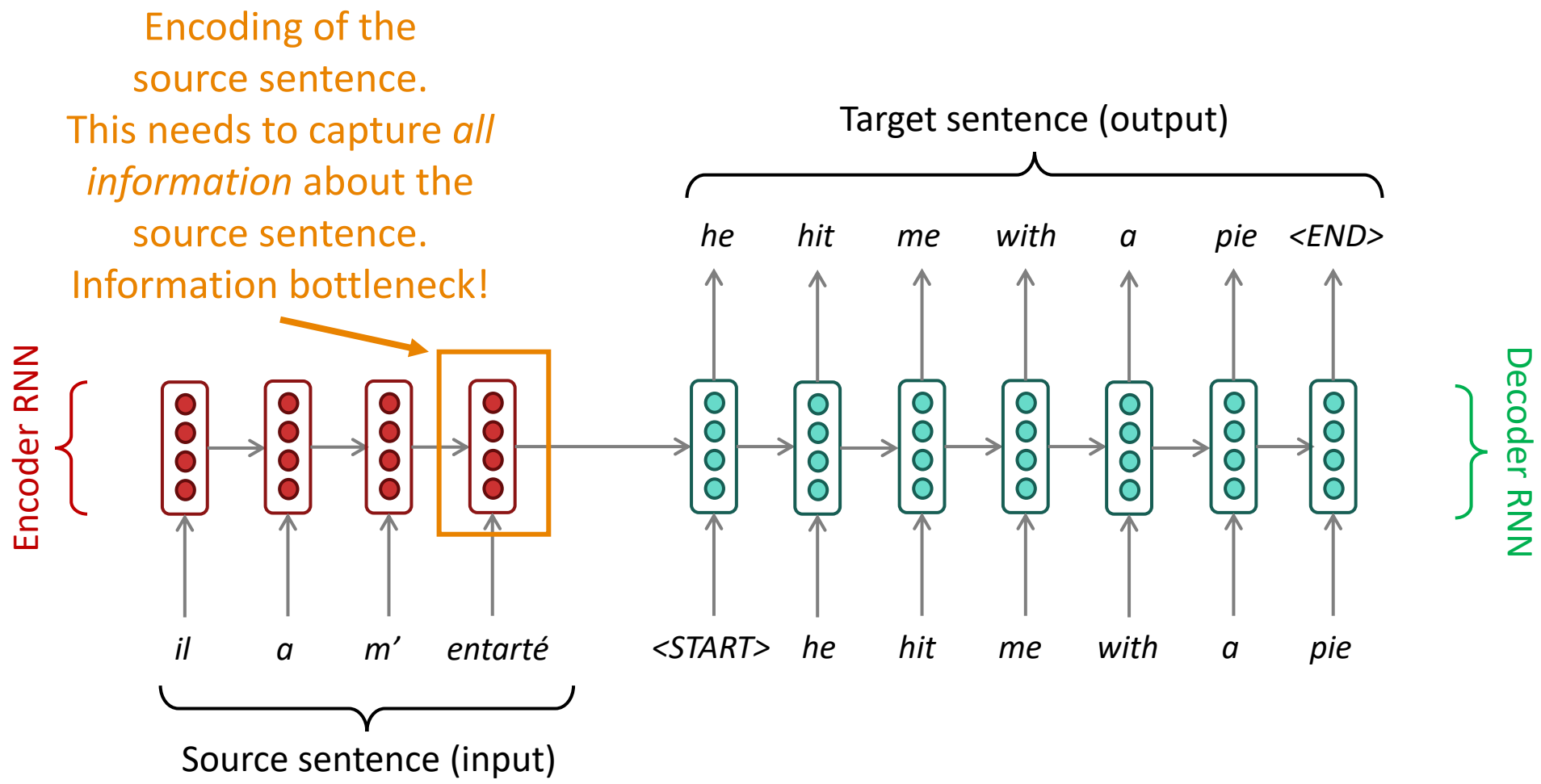
ATTENTION

Sequence-to-sequence: the bottleneck problem



Problems with this architecture?

Sequence-to-sequence: the bottleneck problem

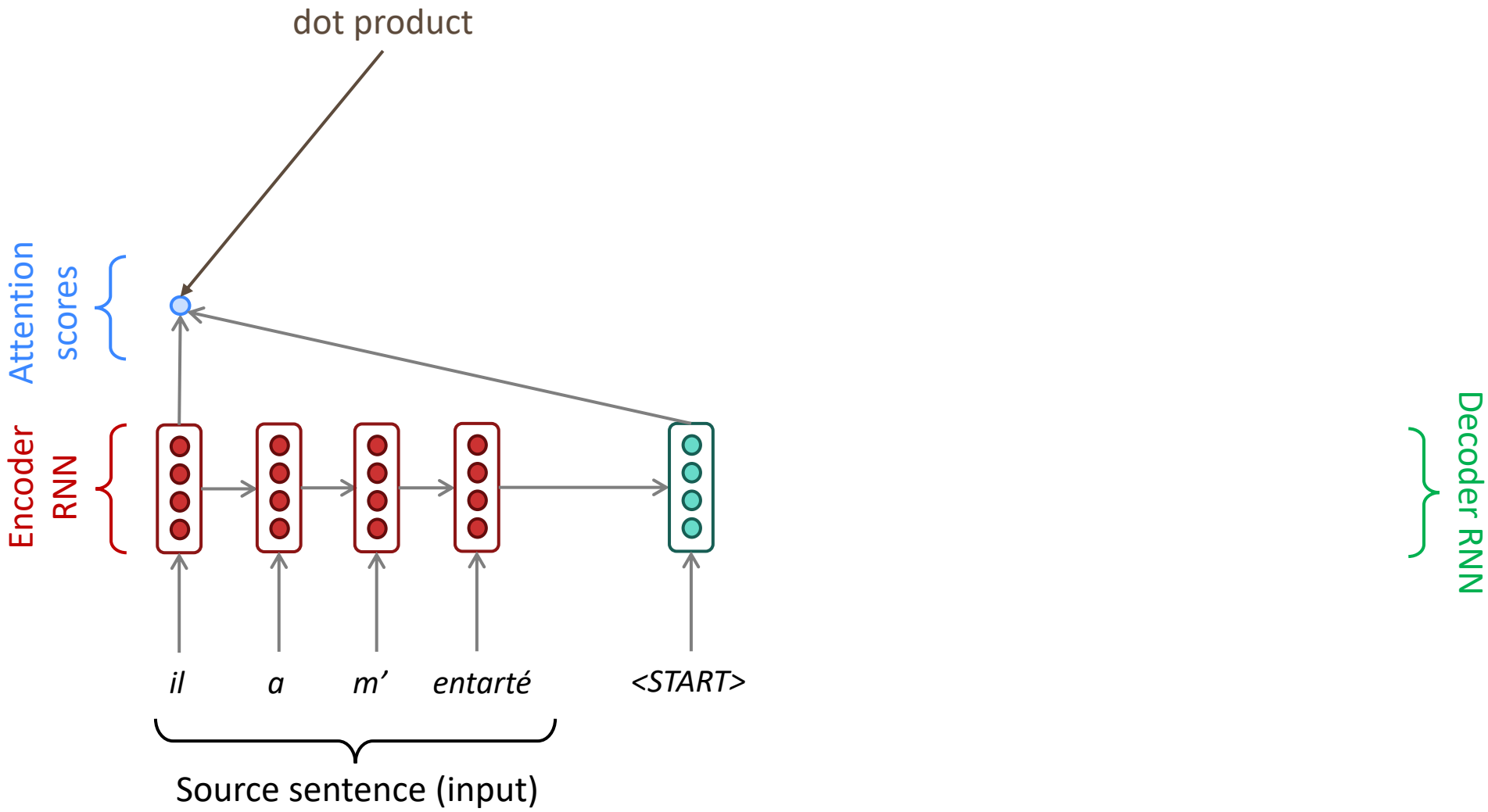


Attention

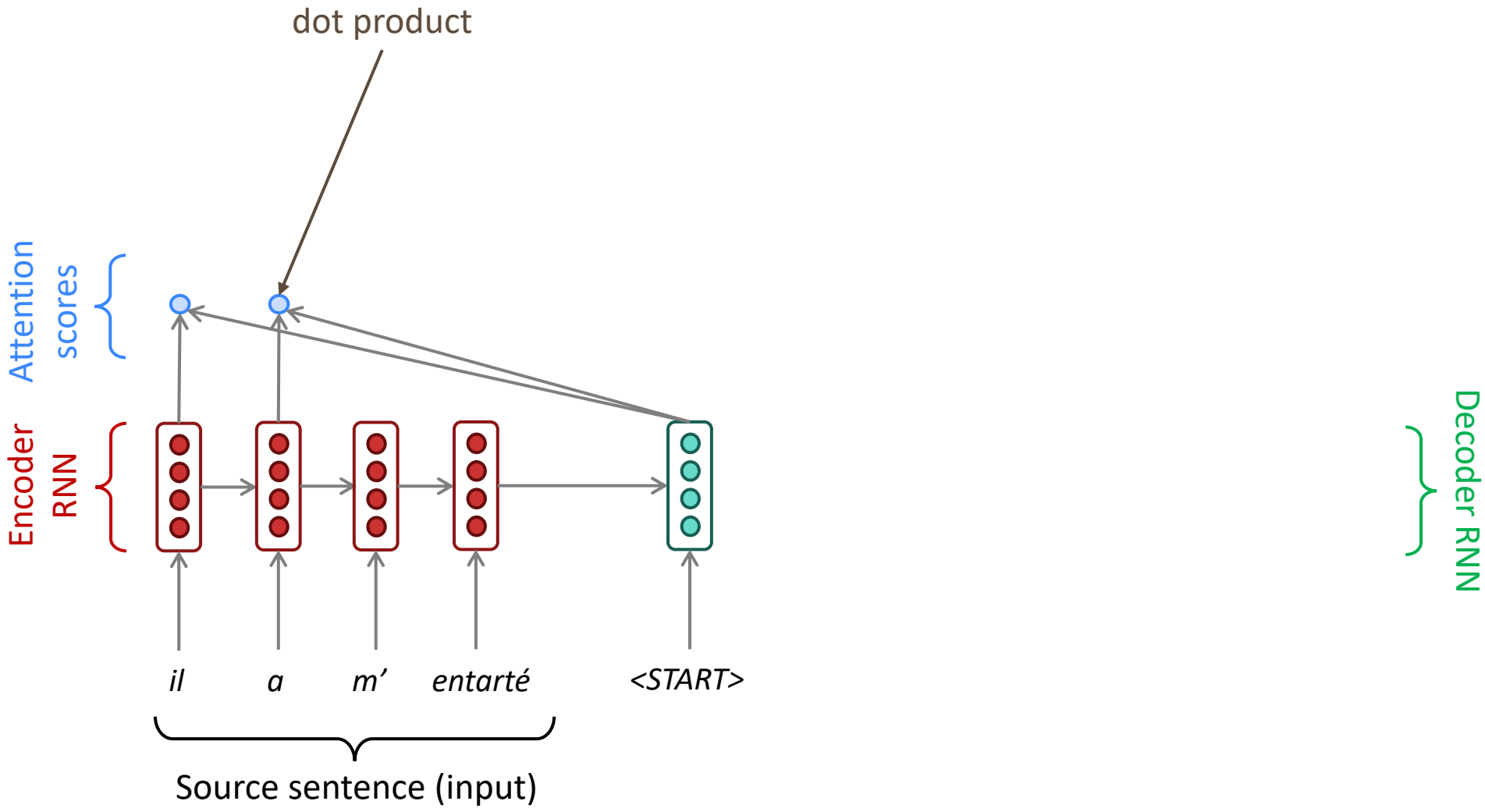
- **Attention** provides a solution to the bottleneck problem.
- **Core idea**: on each step of the decoder, **use direct connection to the encoder to focus on a particular part** of the source sequence

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

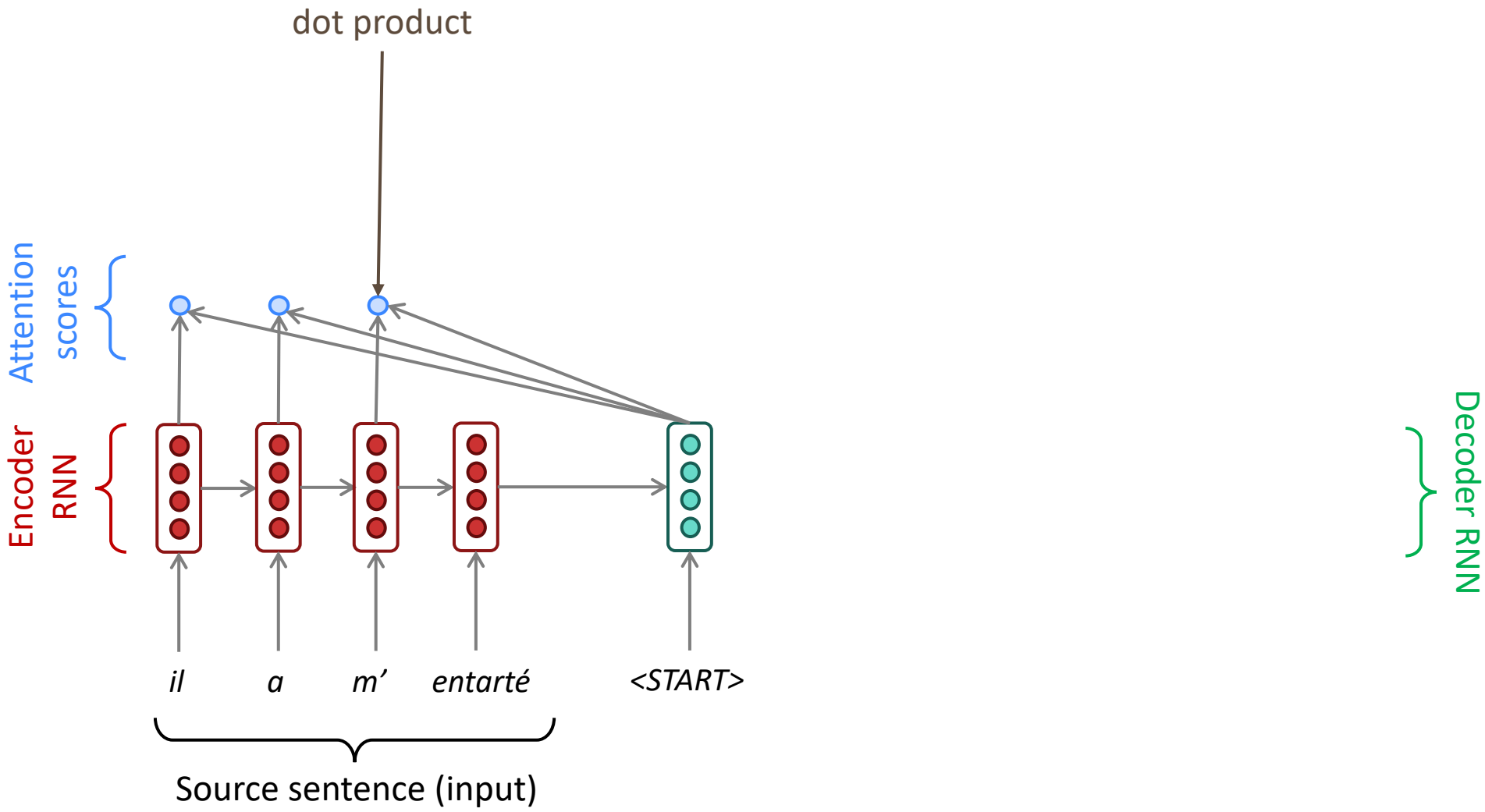
Sequence-to-sequence with attention



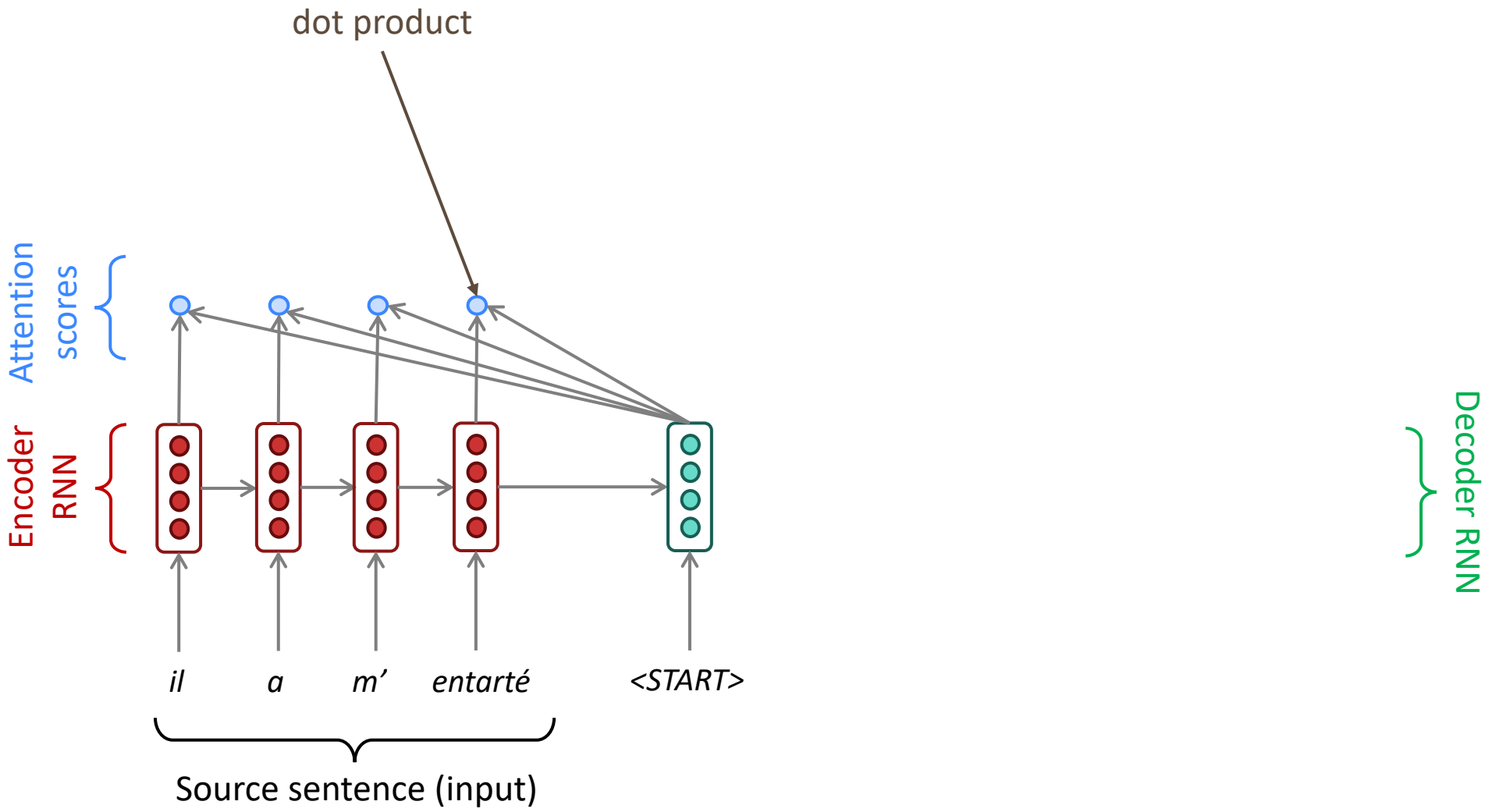
Sequence-to-sequence with attention



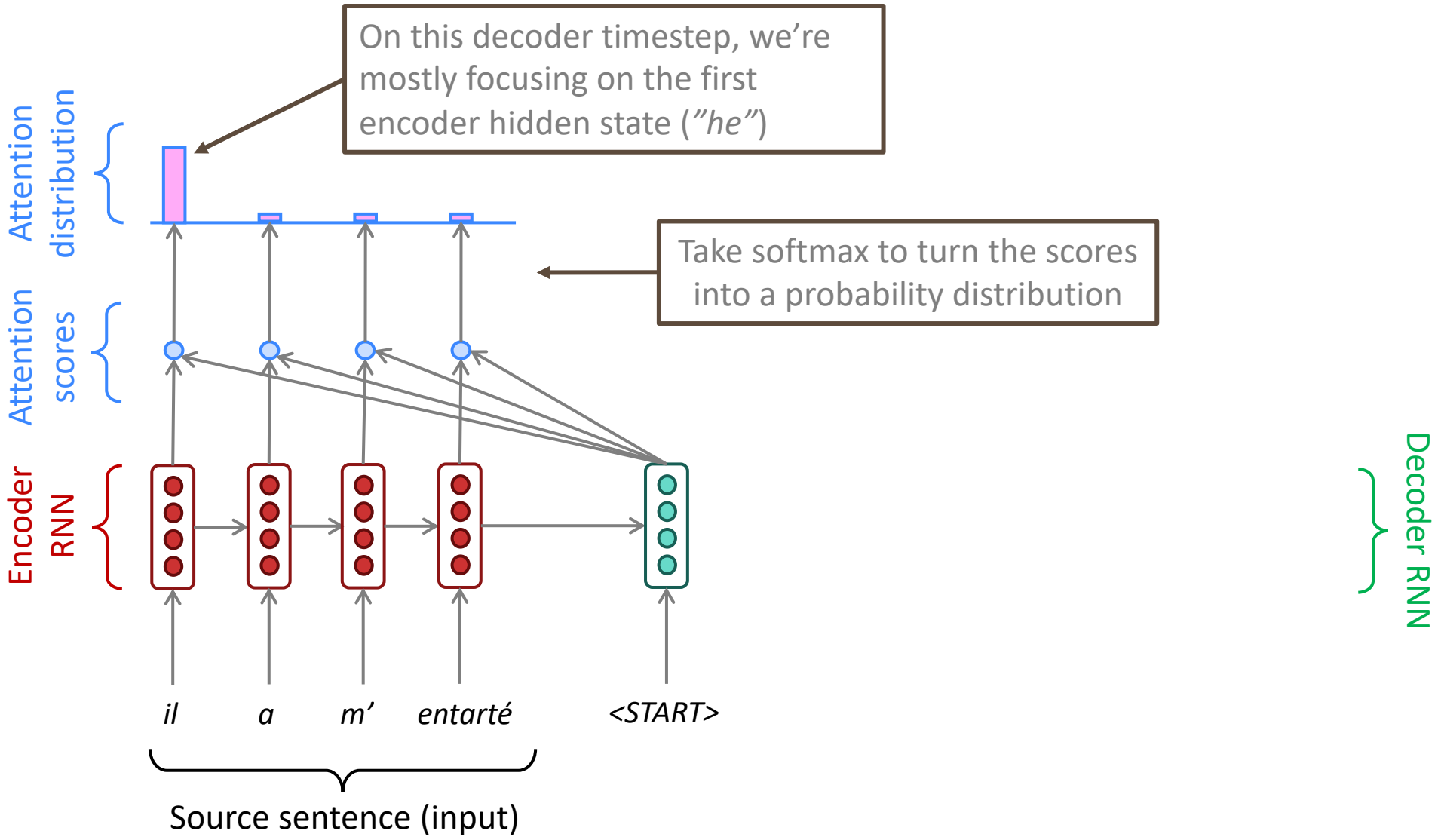
Sequence-to-sequence with attention



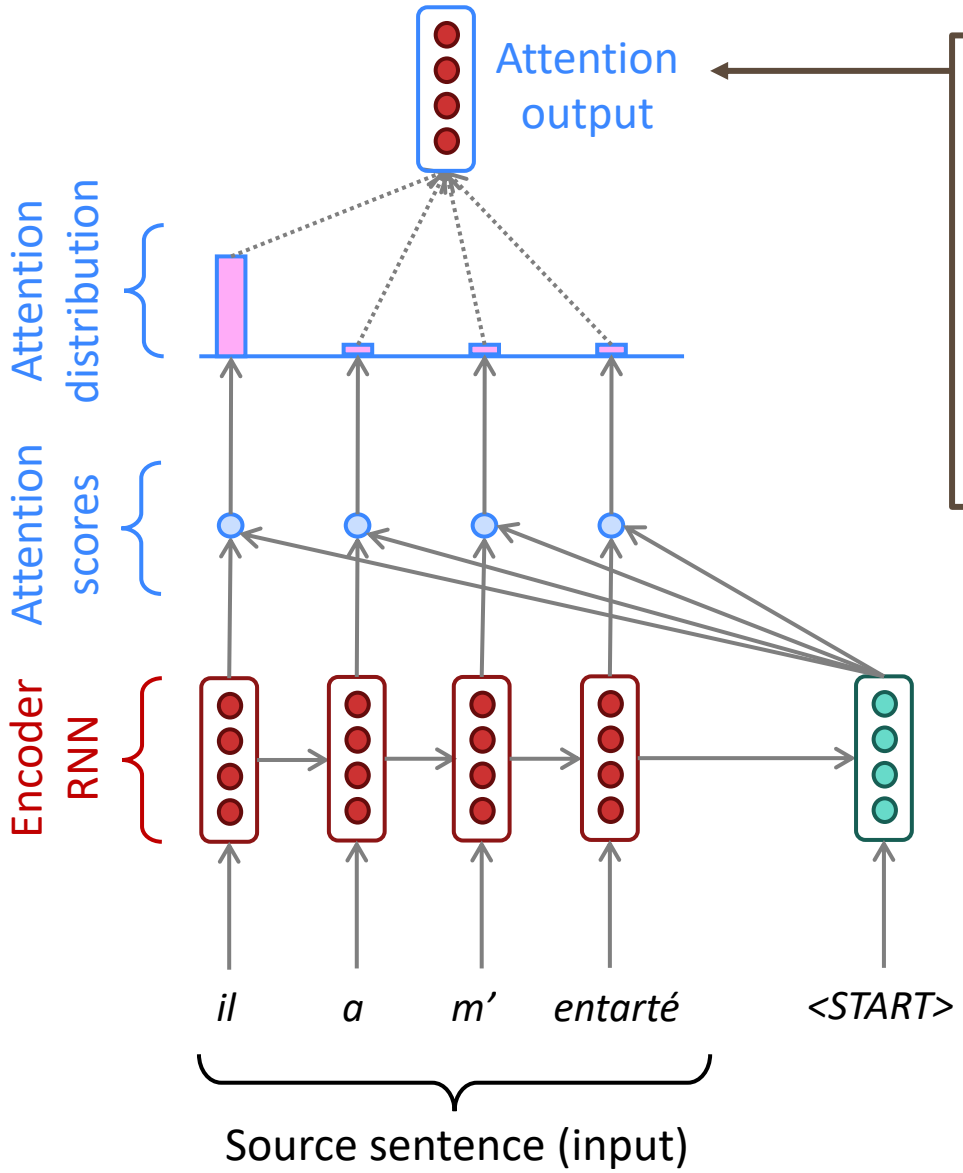
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention

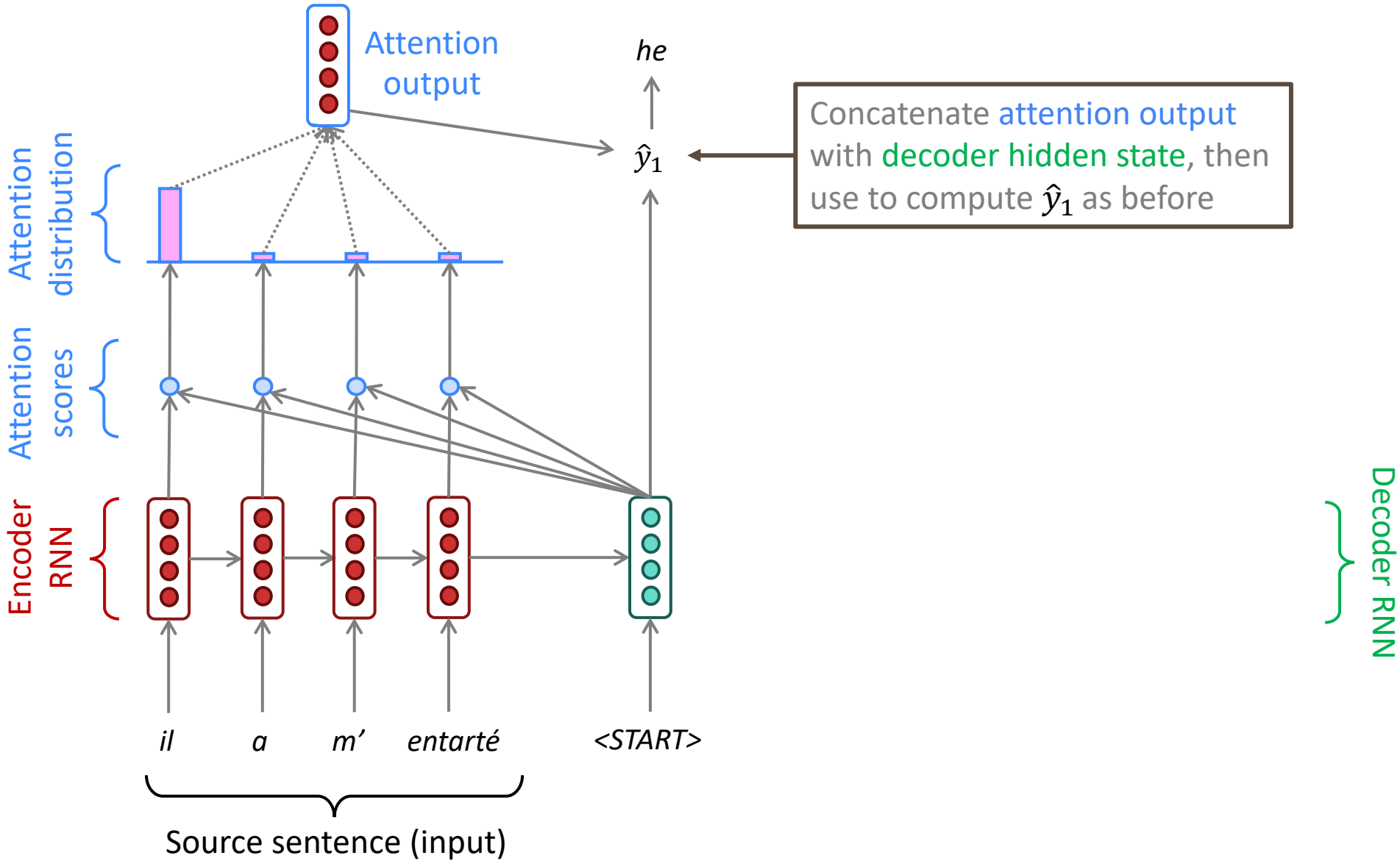


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

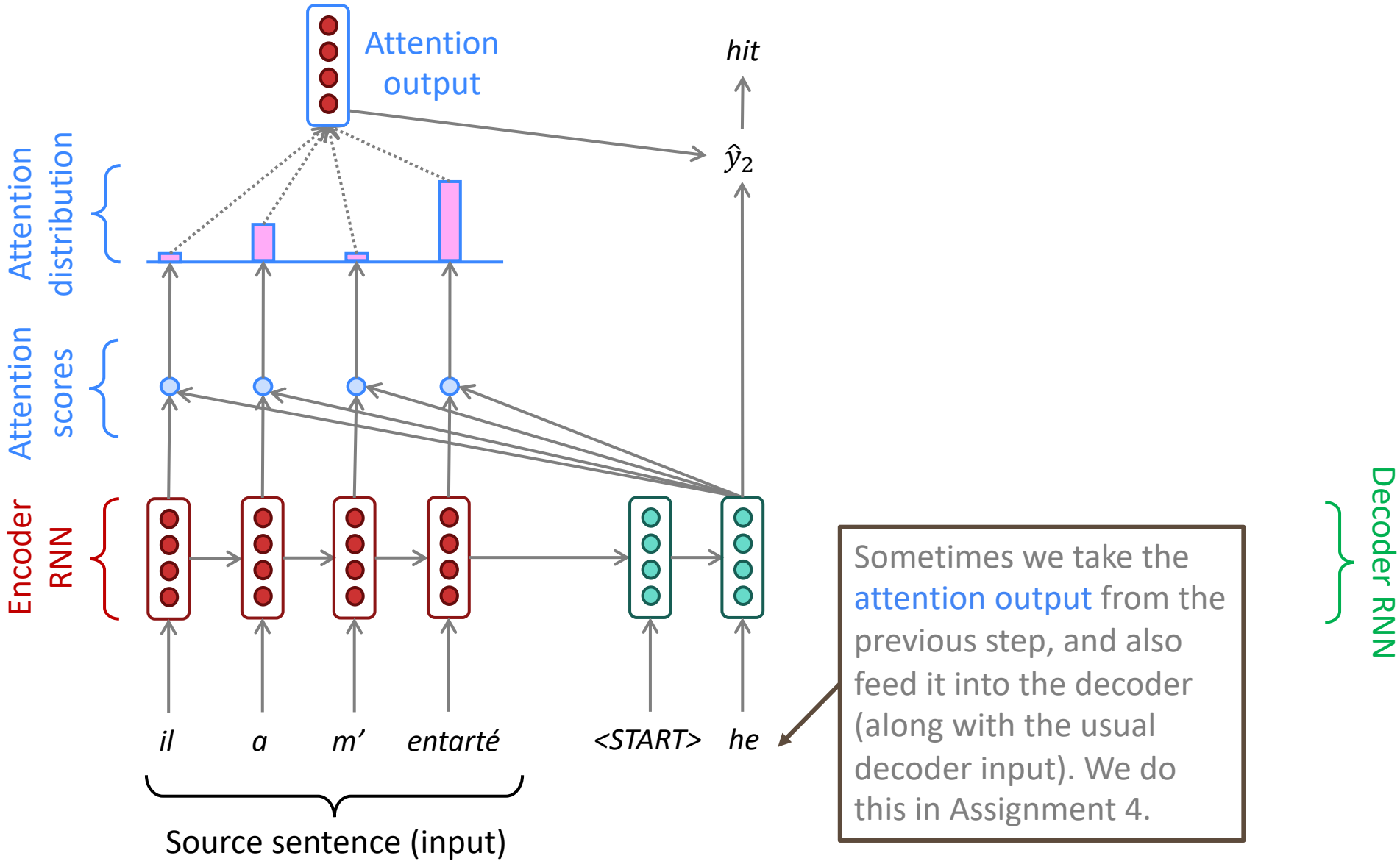
The attention output mostly contains information from the hidden states that received high attention.

Decoder RNN

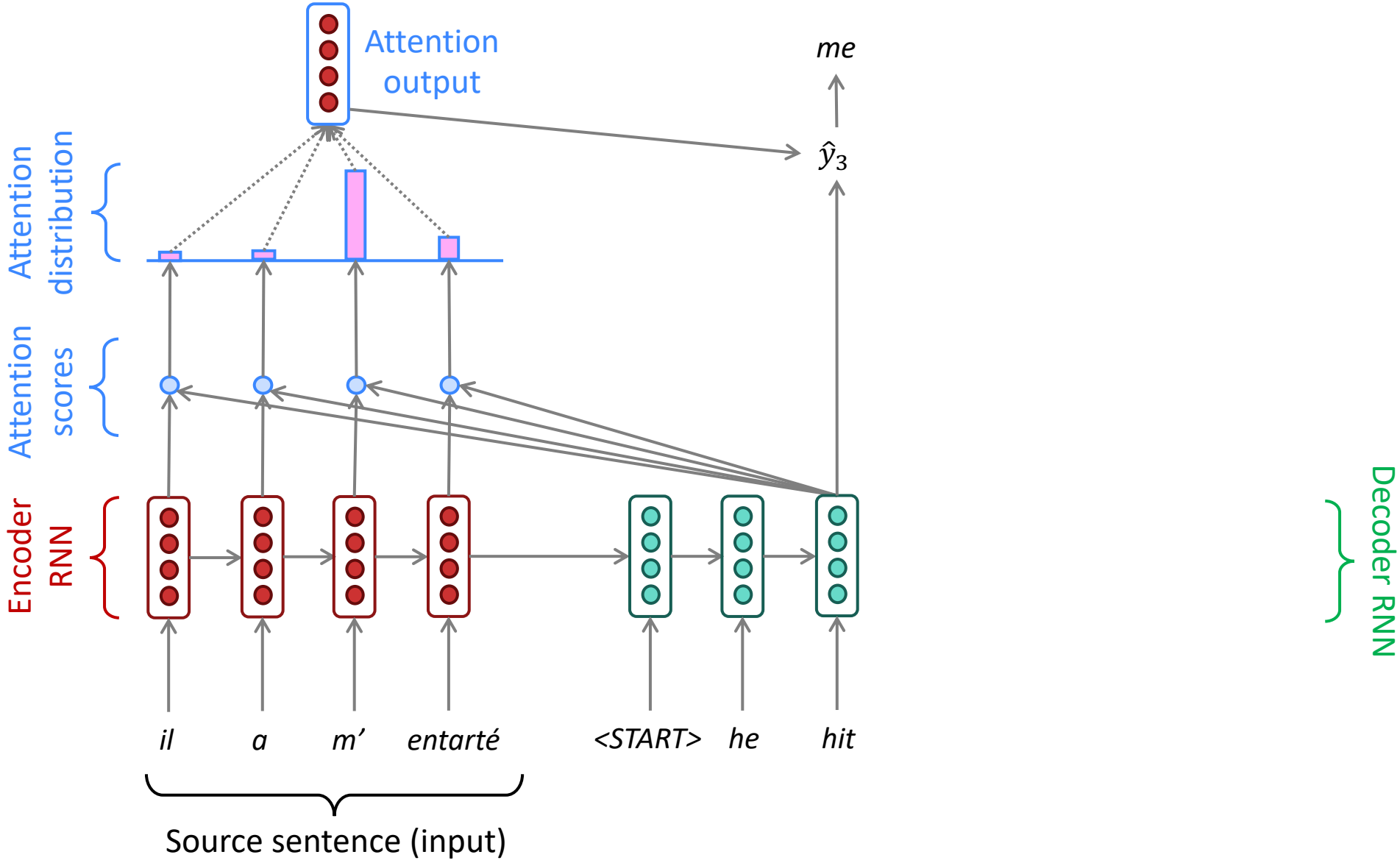
Sequence-to-sequence with attention



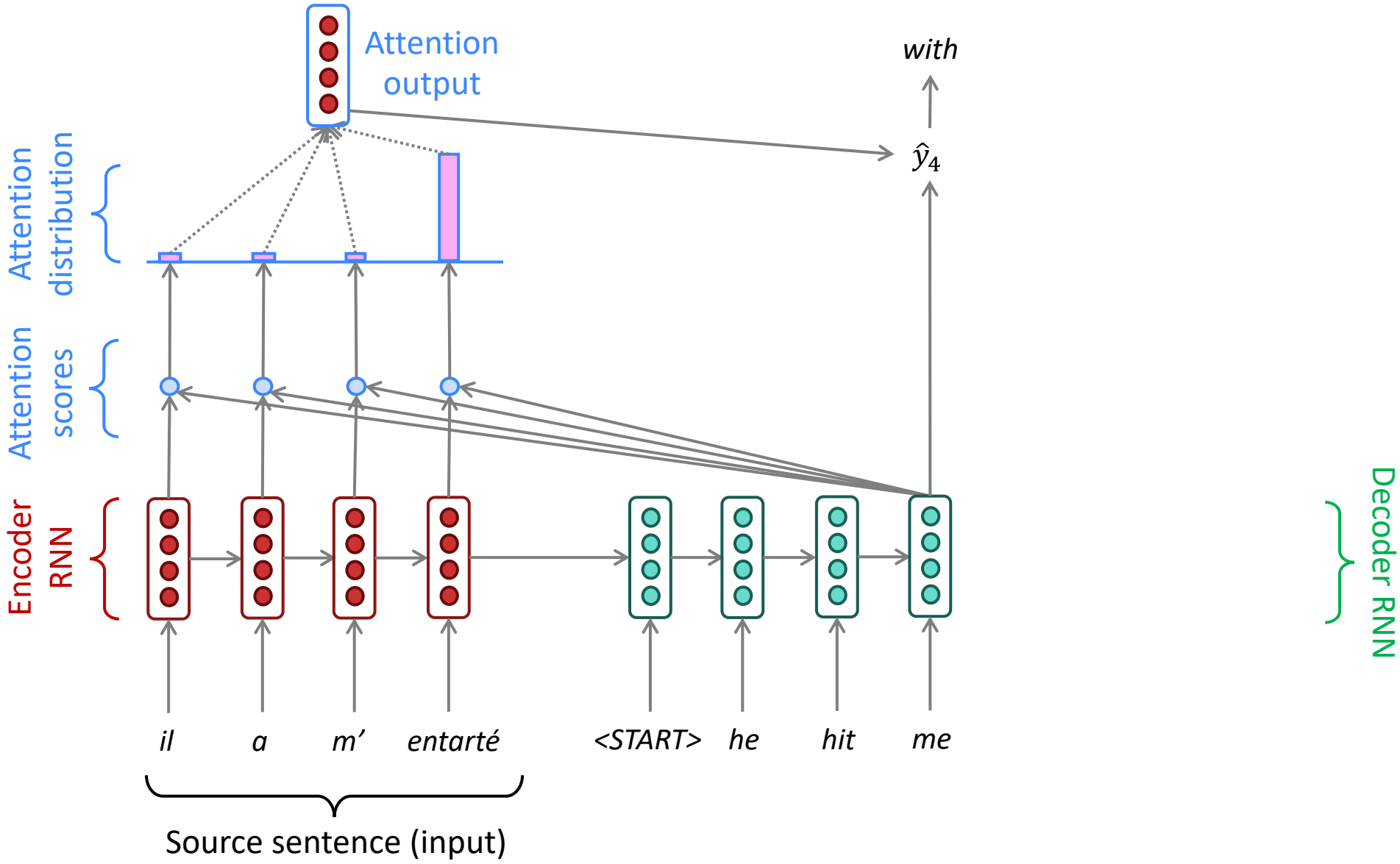
Sequence-to-sequence with attention



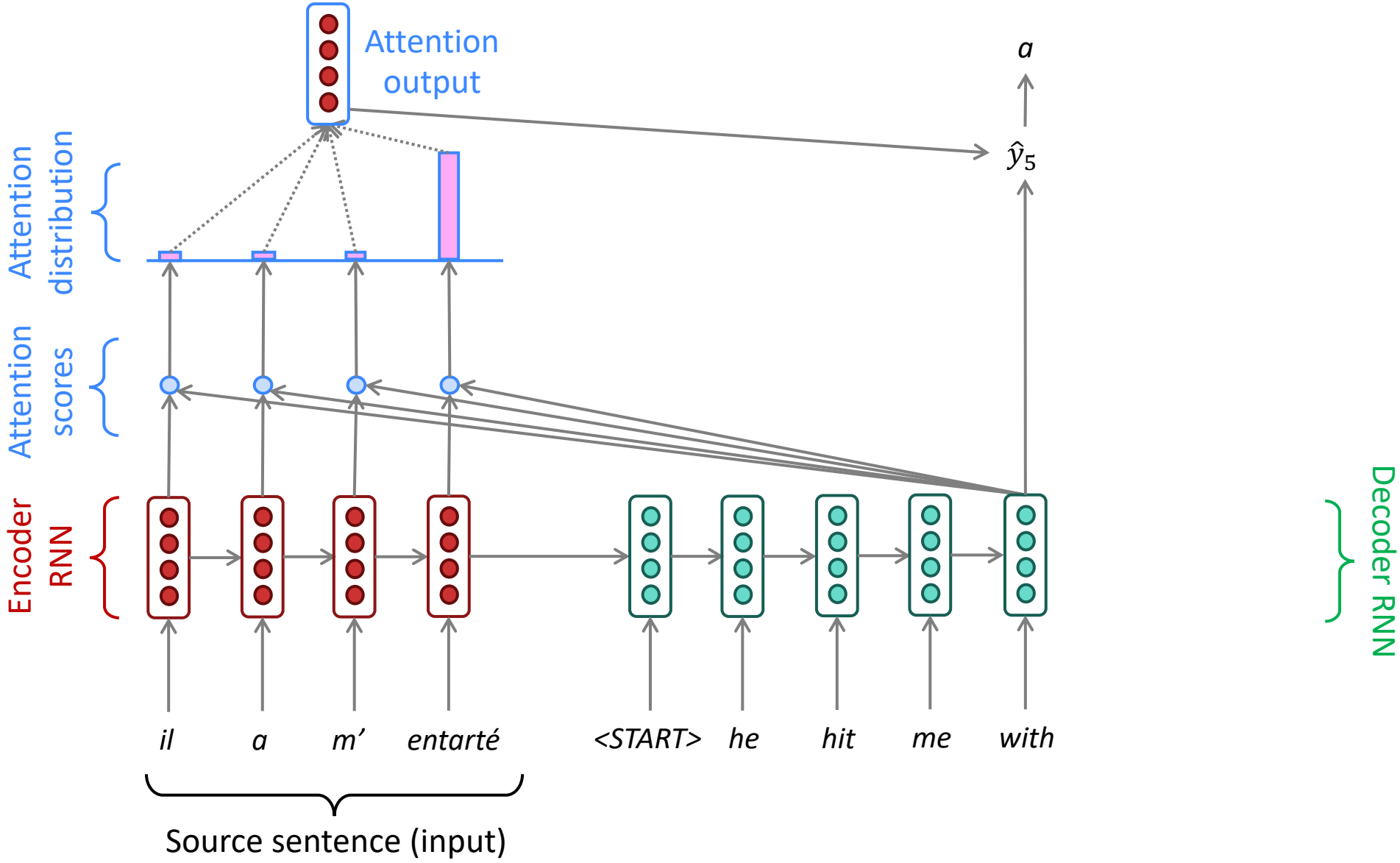
Sequence-to-sequence with attention



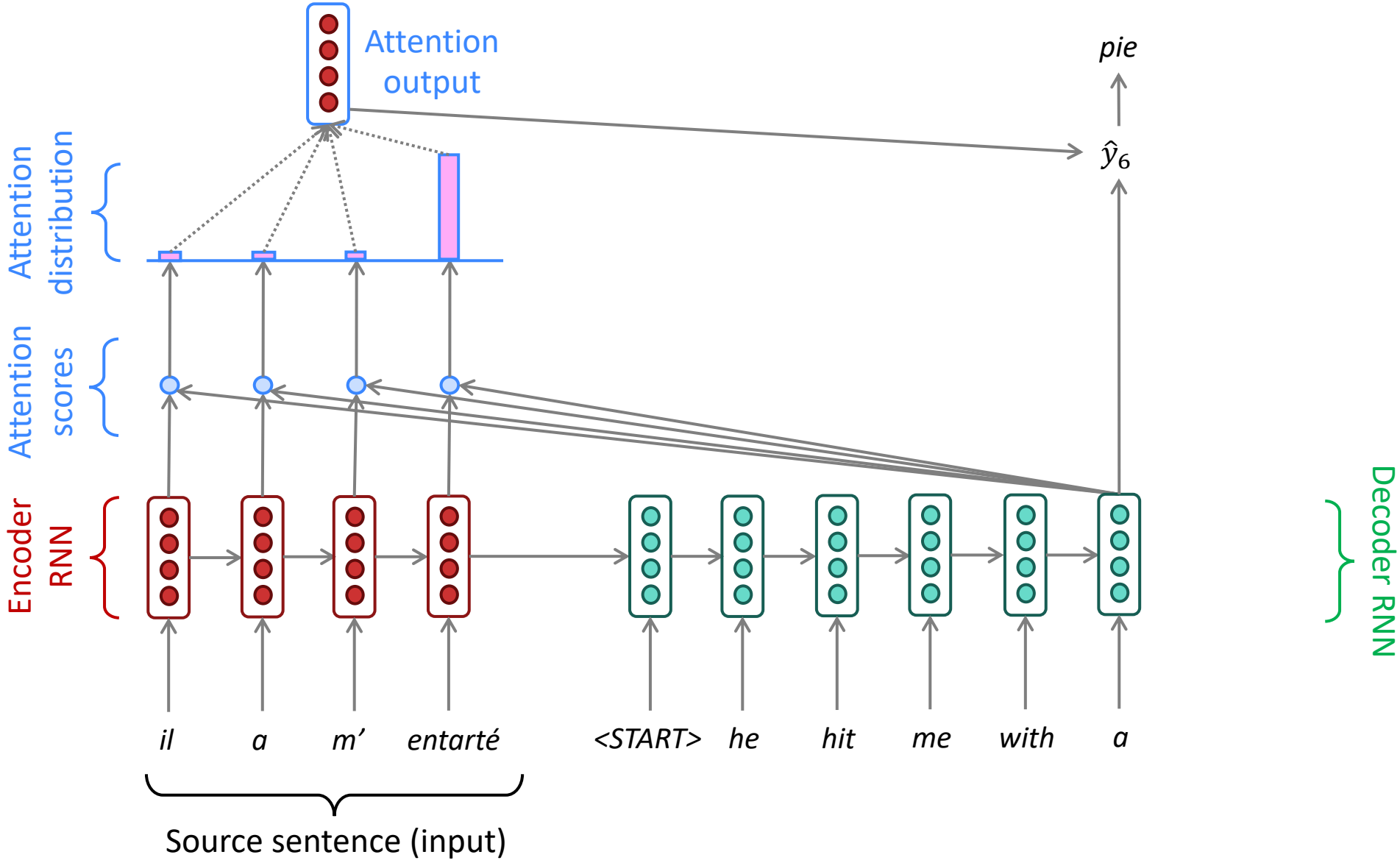
Sequence-to-sequence with attention



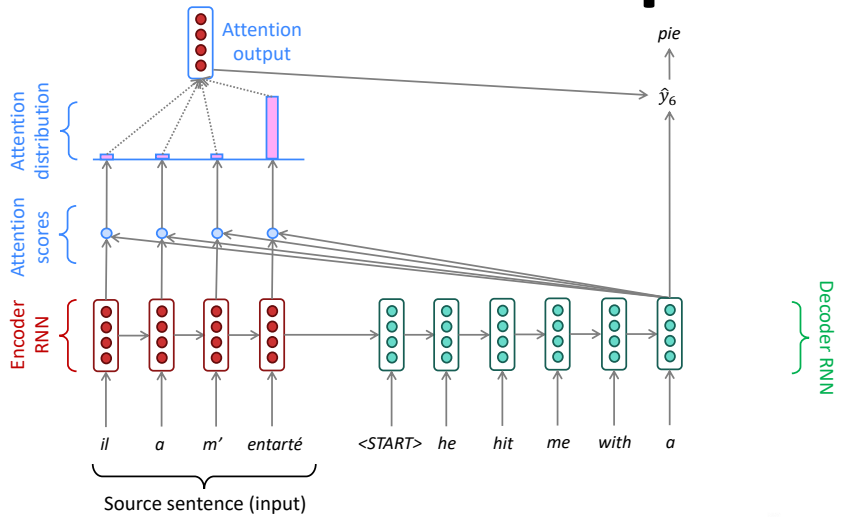
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Attention: in equations



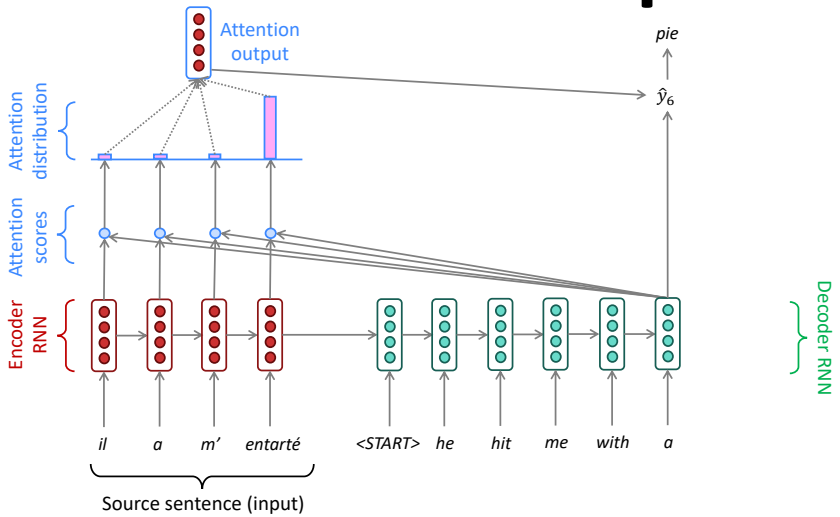
- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

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

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

Attention: in equations



- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

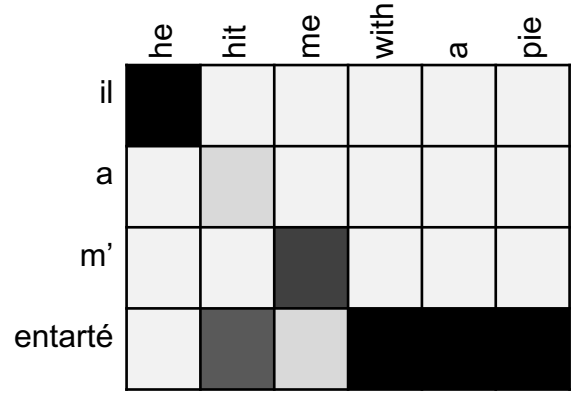
$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

Attention is great!

- Attention significantly **improves NMT performance**
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention provides **more "human-like" model** of the MT process
  - You can look back at the source sentence while translating, rather than needing to remember it all
- Attention **solves the bottleneck problem**
  - Attention allows decoder to look directly at source; bypass bottleneck

Attention is great!

- Attention **helps with the vanishing gradient problem**
  - Provides shortcut to far away states
- Attention provides **some interpretability**
  - By inspecting attention distribution, we 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



There are several attention variants

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

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

  - Using attention distribution to take weighted sum of values:

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

thus obtaining the ***attention output*** *a* (sometimes called the *context vector*)

Attention variants

- There are several ways you can compute $\boldsymbol{e} \in \mathbb{R}^N$ from $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and $\boldsymbol{s} \in \mathbb{R}^{d_2}$
- **Basic dot-product attention**: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$
  - Note: this assumes $d_1 = d_2$. This is the version we saw earlier.
- **Multiplicative attention**: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$ [Luong, Pham, and Manning 2015]
  - Where $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix. Perhaps better called "bilinear attention"

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

Attention variants

- **Reduced-rank multiplicative attention**: $e_i = s^T(\boldsymbol{U}^T\boldsymbol{V})h_i = (\boldsymbol{U}s)^T(\boldsymbol{V}h_i)$
  - For low rank matrices $\boldsymbol{U} \in \mathbb{R}^{k \times d_2}, \boldsymbol{V} \in \mathbb{R}^{k \times d_1}, k \ll d_1, d_2$
- **Additive attention**: $\boldsymbol{e}_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1\boldsymbol{h}_i + \boldsymbol{W}_2\boldsymbol{s}) \in \mathbb{R}$ [Bahdanau, Cho, and Bengio 2014]
  - Where $\boldsymbol{W}_1 \in \mathbb{R}^{d_3 \times d_1}, \boldsymbol{W}_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $\boldsymbol{v} \in \mathbb{R}^{d_3}$ is a weight vector.
  - d_3 (the attention dimensionality) is a hyperparameter
  - "Additive" is a weird/bad name. It's really using a feed-forward neural net layer.

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

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).

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.
- **Upshot:**
  - Attention has become the powerful, flexible, general way pointer and memory manipulation in all deep learning models. A new idea from after 2010! From NMT!

Transformers

Transformers: Is Attention All We Need?

- So far, we learned that attention dramatically improves the performance of recurrent neural networks.
- Today, we will take this one step further and ask **Is Attention All We Need?**

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017)

Transformers: Is Attention All We Need?

- So far, we learned that attention dramatically improves the performance of recurrent neural networks.
- Today, we will take this one step further and ask **Is Attention All We Need?**
- Spoiler: Not Quite!

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

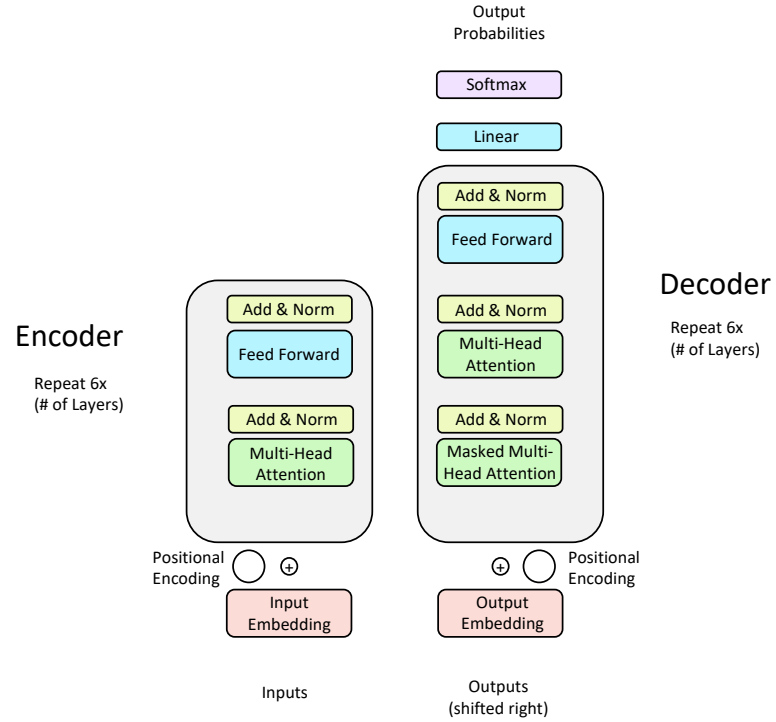
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017)

Transformers Have Revolutionized the Field of NLP

- By the end of this lecture, you will deeply understand the neural architecture that underpins virtually every state-of-the-art NLP model today!

Courtesy of Paramount Pictures



Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017)

Great Results with Transformers: Machine Translation

- First, Machine Translation results from the original Transformers paper!

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

[Test sets: WMT 2014 English-German and English-French]

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017)

Great Results with Transformers: Document Generation

- Next, document generation!
  (For perplexity, lower is better; for ROUGE-L, higher is better.)

Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

The old standard from last week!

Transformers dominating across the board.

[Liu et al., 2018]; WikiSum dataset

Preview: Great Results with (Pre-Trained) Transformers

- Before too long, most Transformers results also incorporate pretraining, a method we'll go over on next lecture.
- Transformers' parallelizability allows for efficient pretraining, and have made them the de-facto standard.

On this popular aggregate benchmark, for example:

GLUE

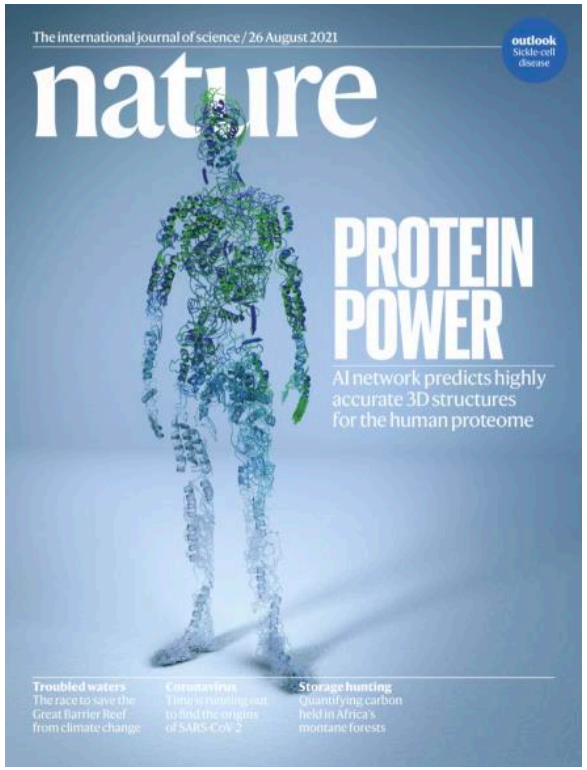
All top models are Transformer (and pretraining)-based.

	Rank	Name	Model	URL	Score
	1	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.8
	2	HFL iFLYTEK	MacALBERT + DKM		90.7
+	3	Alibaba DAMO NLP	StructBERT + TAPT		90.6
+	4	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6
	5	ERNIE Team - Baidu	ERNIE		90.4
	6	T5 Team - Google	T5		90.3

[Liu et al., 2018]

Transformers Even Show Promise Outside of NLP

Protein Folding



[Jumper et al. 2021] aka AlphaFold2!

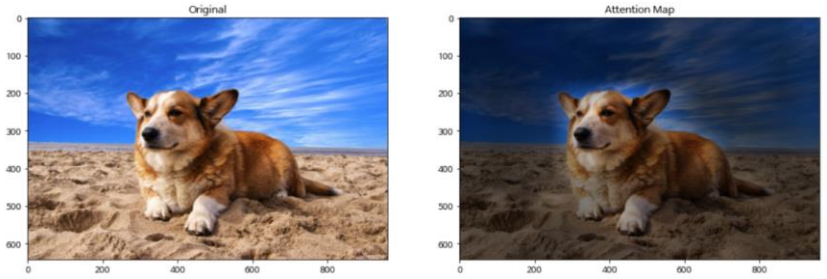
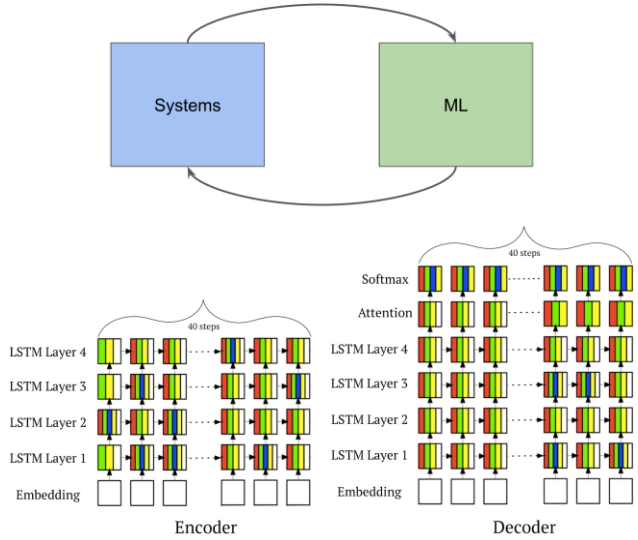


Image Classification

[Dosovitskiy et al. 2020]: Vision Transformer (ViT) outperforms ResNet-based baselines with substantially less compute.

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k



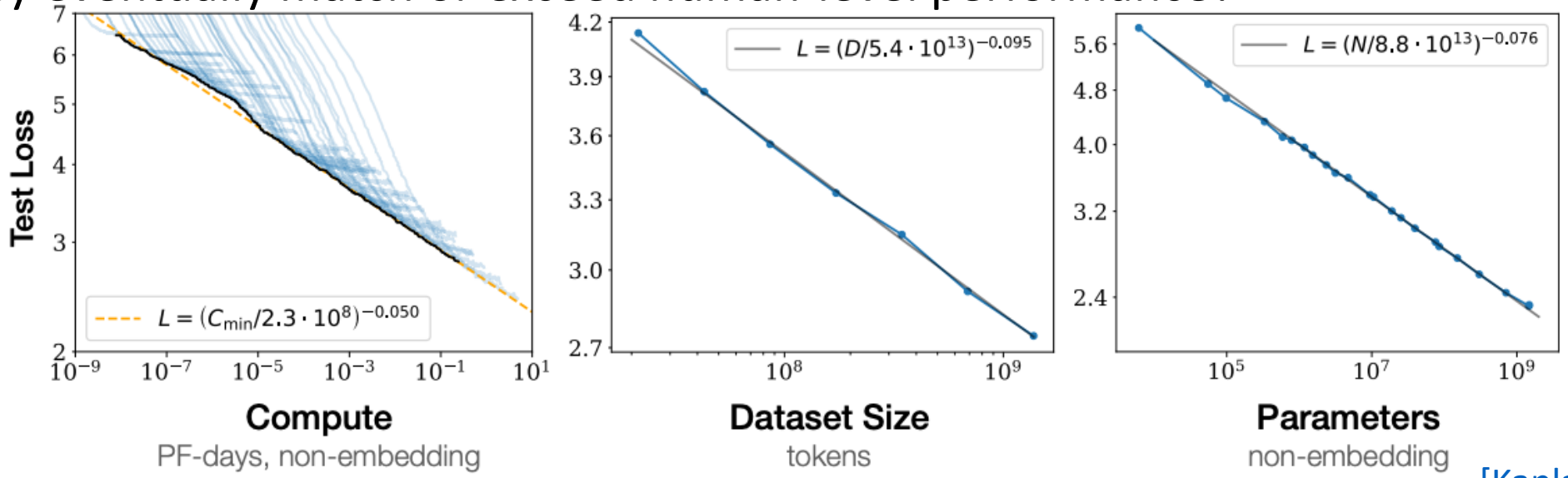
ML for Systems

[Zhou et al. 2020]: A Transformer-based compiler model (GO-one) speeds up a Transformer model!

Model (#devices)	GO-one (s)	HP (s)	METIS (s)	HDP (s)	Run time speed up over HP / HDP	Search speed up over HDP
2-layer RNNLM (2)	0.173	0.192	0.355	0.191	9.9% / 9.4%	2.95x
4-layer RNNLM (4)	0.210	0.239	0.503	0.251	13.8% / 16.3%	1.76x
8-layer RNNLM (8)	0.320	0.332	OOM	0.764	3.8% / 58.1%	27.8x
2-layer GNMT (2)	0.301	0.384	0.344	0.327	27.6% / 14.3%	30x
4-layer GNMT (4)	0.350	0.469	0.466	0.432	34% / 23.4%	58.8x
[illegible]	0.440	0.562	OOM	0.693	21.7% / 36.5%	7.35x
2-layer Transformer-XL (2)	0.223	0.268	0.37	0.262	20.1% / 17.4%	40x
4-layer Transformer-XL (4)	0.230	0.27	OOM	0.259	17.4% / 12.6%	26.7x
8-layer Transformer-XL (8)	0.350	0.46	OOM	0.425	23.9% / 16.7%	16.7x
[illegible]	0.229	0.312	OOM	0.301	26.6% / 23.9%	13.5x
Inception (2) b64	0.423	0.731	OOM	0.498	42.1% / 29.3%	21.0x
AmoebaNet (4)	0.394	0.44	0.426	0.418	26.1% / 6.1%	58.8x
2-stack 18-layer WaveNet (2)	0.317	0.376	OOM	0.354	18.6% / 11.7%	6.67x
4-stack 36-layer WaveNet (4)	0.659	0.988	OOM	0.721	50% / 9.4%	20x
GEOMEAN	-	-	-	-	**20.5% / 18.2%**	**15x**

Scaling Laws: Are Transformers All We Need?

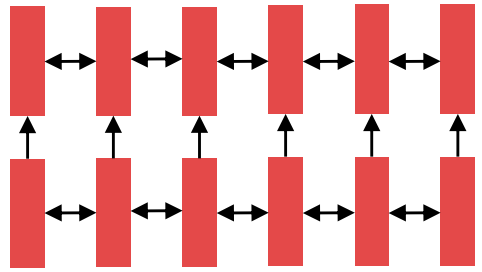
- With Transformers, language modeling performance improves smoothly as we increase model size, training data, and compute resources.
- This power-law relationship has been observed over multiple orders of magnitude with no sign of slowing!
- If we keep scaling up these models (with no change to the architecture), could they eventually match or exceed human-level performance?



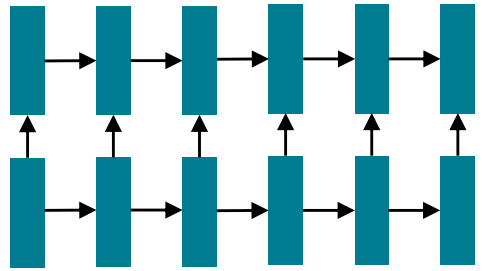
[Kaplan et al., 2020]

As of last week: recurrent models for (most) NLP!

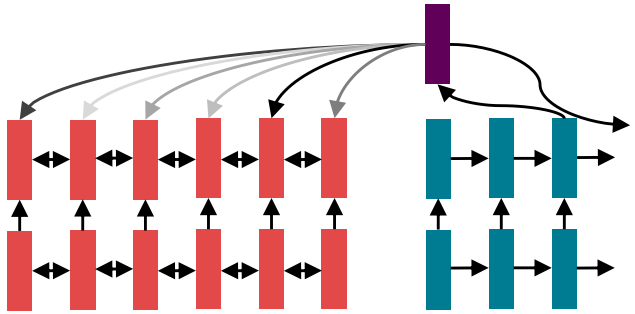
- Circa 2016, the de facto strategy in NLP is to **encode** sentences with a bidirectional LSTM:
  (for example, the source sentence in a translation)



- Define your output (parse, sentence, summary) as a sequence, and use an LSTM to generate it.

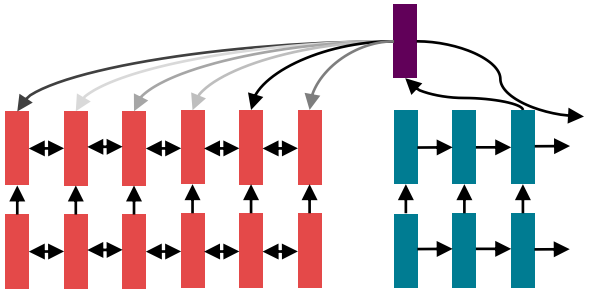


- Use attention to allow flexible access to memory



Today: Same goals, different building blocks

- Last week, we learned about sequence-to-sequence problems and encoder-decoder models.
- Today, we're **not** trying to motivate entirely new ways of looking at problems (like Machine Translation)
- Instead, we're trying to find the best **building blocks** to plug into our models and enable broad progress.

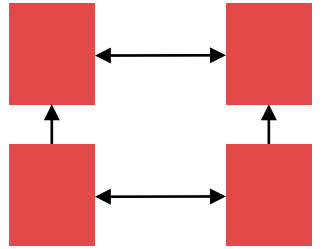


2014-2017ish
Recurrence

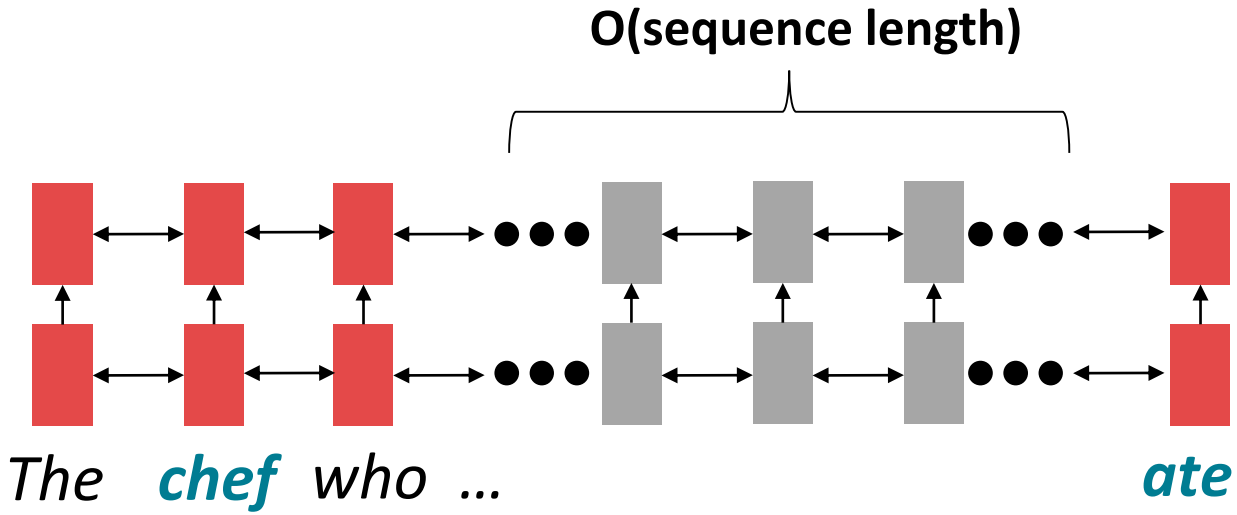
Lots of trial
and error

2021
??????

Issues with recurrent models: **Linear interaction distance**

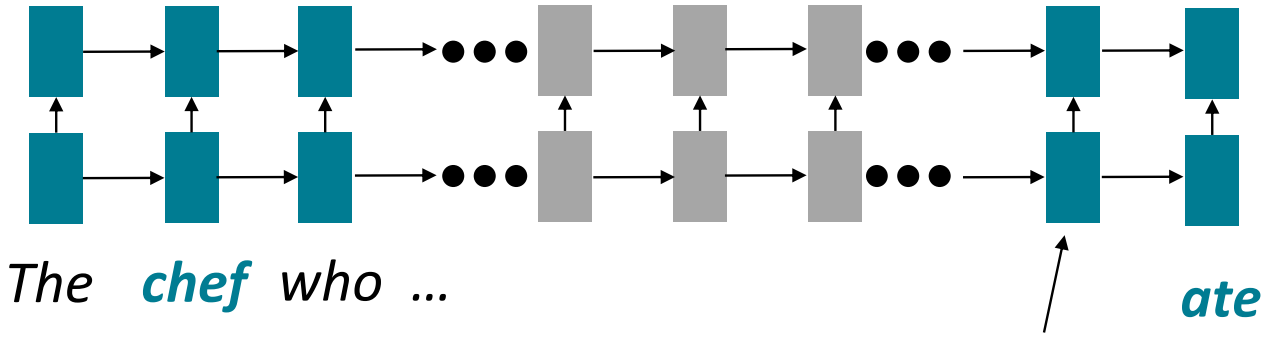


- RNNs are unrolled "left-to-right".
- It encodes linear locality: a useful heuristic!
  - Nearby words often affect each other's meanings
- **Problem:** RNNs take **O(sequence length)** steps for distant word pairs to interact.



Issues with recurrent models: **Linear interaction distance**

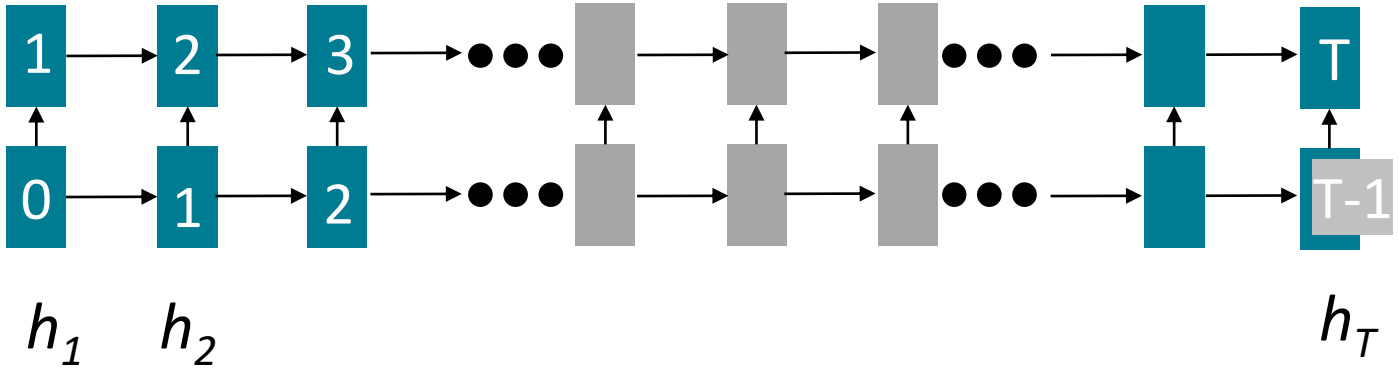
- **O(sequence length)** steps for distant word pairs to interact means:
  - Hard to learn long-distance dependencies (because gradient problems!)
  - Linear order of words is “baked in”; we already know sequential structure doesn't tell the whole story...



Info of ***chef*** has gone through O(sequence length) many layers!

Issues with recurrent models: **Lack of parallelizability**

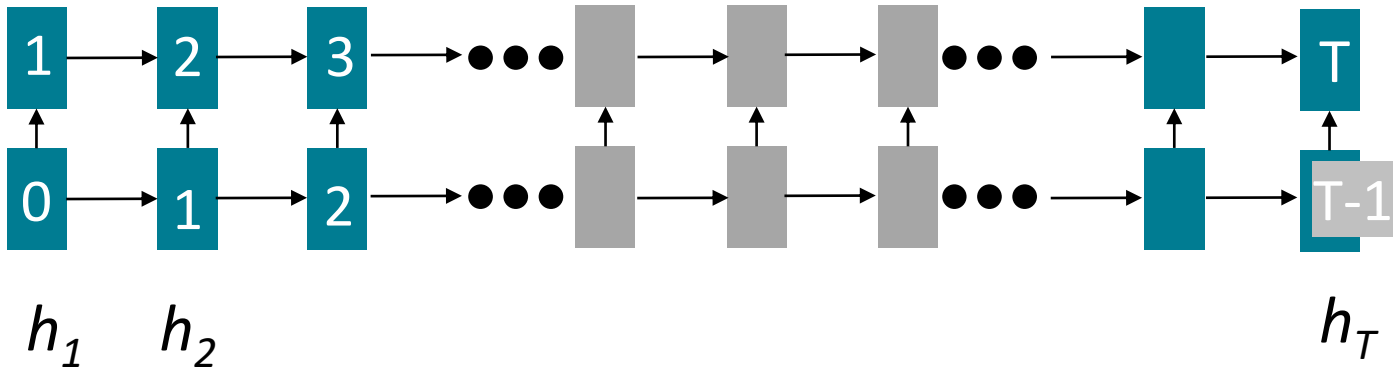
- Forward and backward passes have **O(seq length)** unparallelizable operations
  - GPUs (and TPUs) can perform many independent computations at once!
  - But future RNN hidden states can't be computed in full before past RNN hidden states have been computed



Numbers indicate min # of steps before a state can be computed

Issues with recurrent models: **Lack of parallelizability**

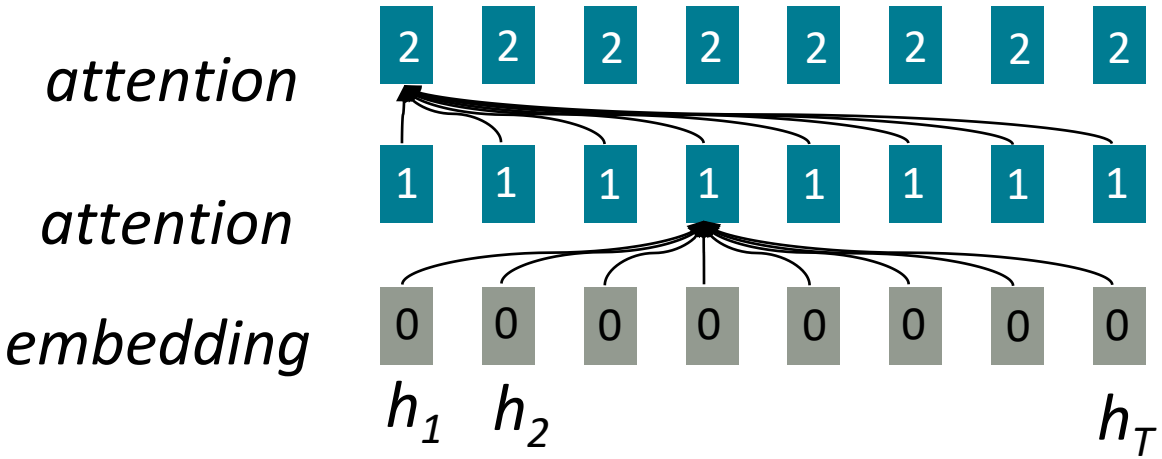
- Forward and backward passes have **O(seq length)** unparallelizable operations
  - Inhibits training on very large datasets!
  - Particularly problematic as sequence length increases, as we can no longer batch many examples together due to memory limitations



Numbers indicate min # of steps before a state can be computed

If not recurrence, then what? **How about (self) attention?**

- To recap, **attention** treats each word's representation as a **query** to access and incorporate information from **a set of values**.
  - Previously, we saw attention from the **decoder** to the **encoder**;
  - **Self-attention** is **encoder-encoder** (or **decoder-decoder**) attention where each word attends to each other word **within the input (or output)**.



All words attend to all words in previous layer; most arrows here are omitted