Deep Learning for Natural Language Processing

Cornelia Caragea

Computer Science
University of Illinois at Chicago

Credits for slides: Manning, Socher, See

Sequence-to-Sequence Models and Attention Mechanism

Today

Lecture Structure:

- Machine Translation - an NLP task
- Sequence-to-Sequence - a neural architecture
- Attention - a neural technique

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

Motivation for Machine Translation

- Machine Translation research has began in the early 1950s.
  - Russian $\rightarrow$ English (motivated by the Cold War)
  - Systems were mostly rule-based, using a bilingual dictionary to map Russian words to their English counterparts

- Statistical Machine Translation: 1990s-2010s
  - Core idea: Learn a probabilistic model from data that maximizes the probability of y (the English sentence) given x (the French sentence), $\text{argmax}_y P(y|x)$.

Using Bayes Rule, we have:

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

Translation Model

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

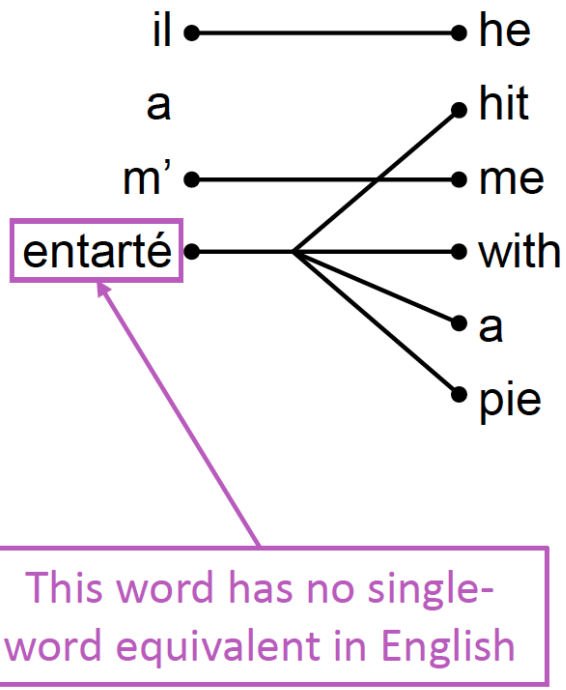
Language Model

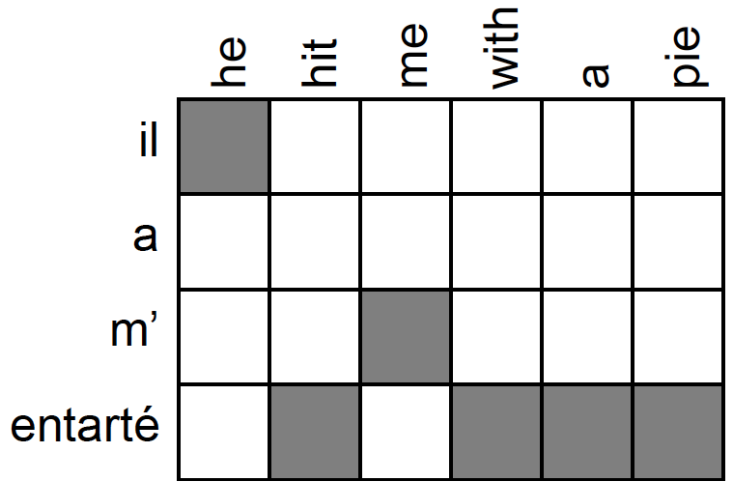
Models how to write good English (*fluency*). Learnt from monolingual data.

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).
- What is actually learned is: $P(x, a|y)$ where a is the alignment, i.e., word-level correspondence between x and y.
    - Alignments can be very complex, e.g., one-to-many, many-to-one and many-to-many.

Example of alignment





Statistical Machine Translation

- SMT was a huge research field.
- The best systems were extremely complex.
  - Lots of feature engineering
  - Require compiling and maintaining extra resources
    - E.g., tables of equivalent phrases
  - Lots of human effort to maintain
    - Repeated effort for each language pair!
- 2014 and on: Neural Machine Translation

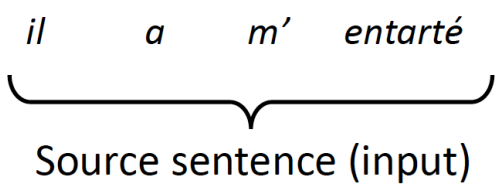
What is Neural Machine Translation?

- 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 (or seq2seq) and it involves two RNNs.

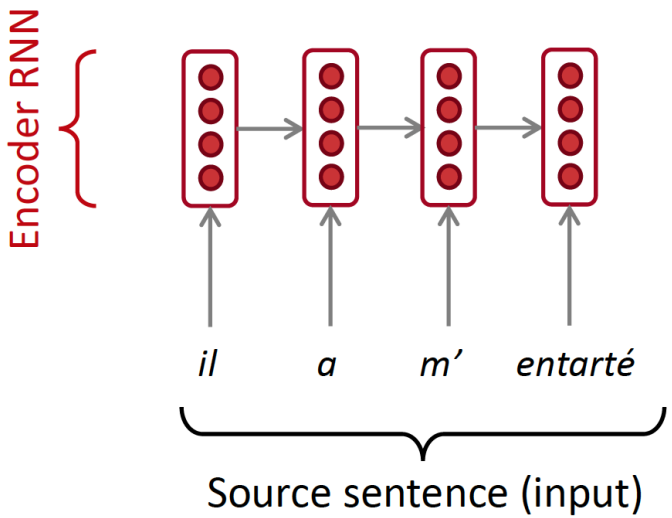
Neural Machine Translation (NMT)

The sequence-to-sequence model



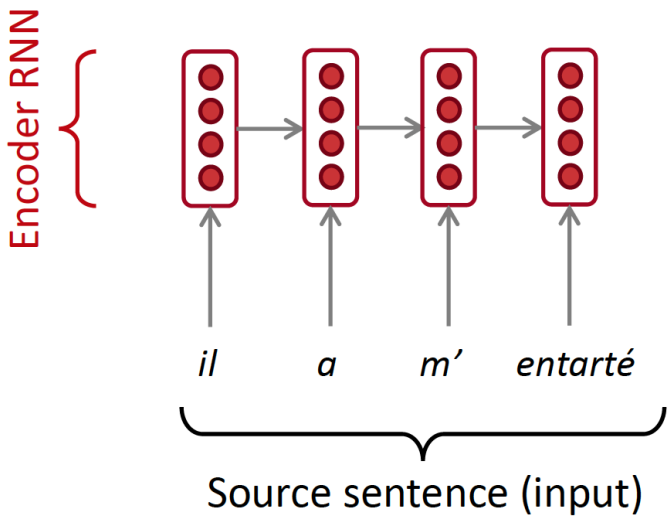
Neural Machine Translation (NMT)

The sequence-to-sequence model



Neural Machine Translation (NMT)

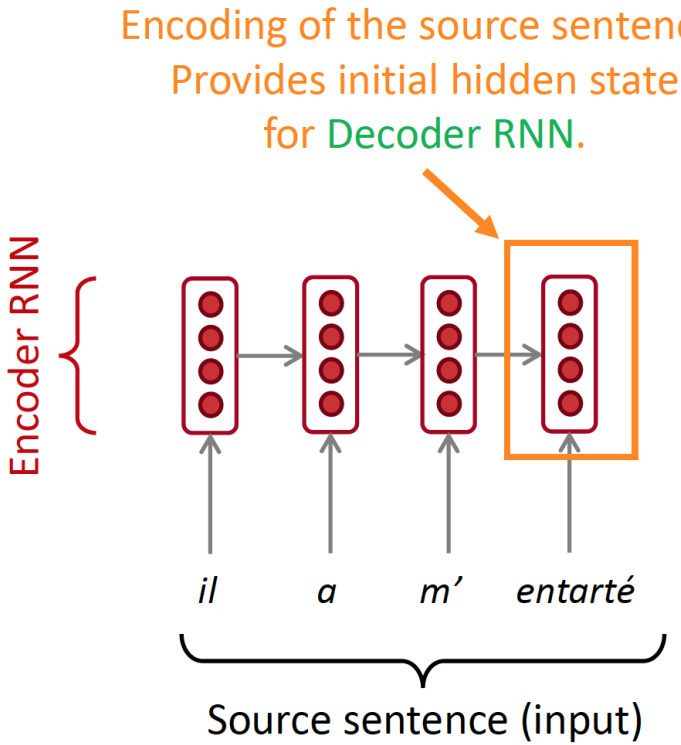
The sequence-to-sequence model



Encoder RNN produces an encoding of the source sentence.

Neural Machine Translation (NMT)

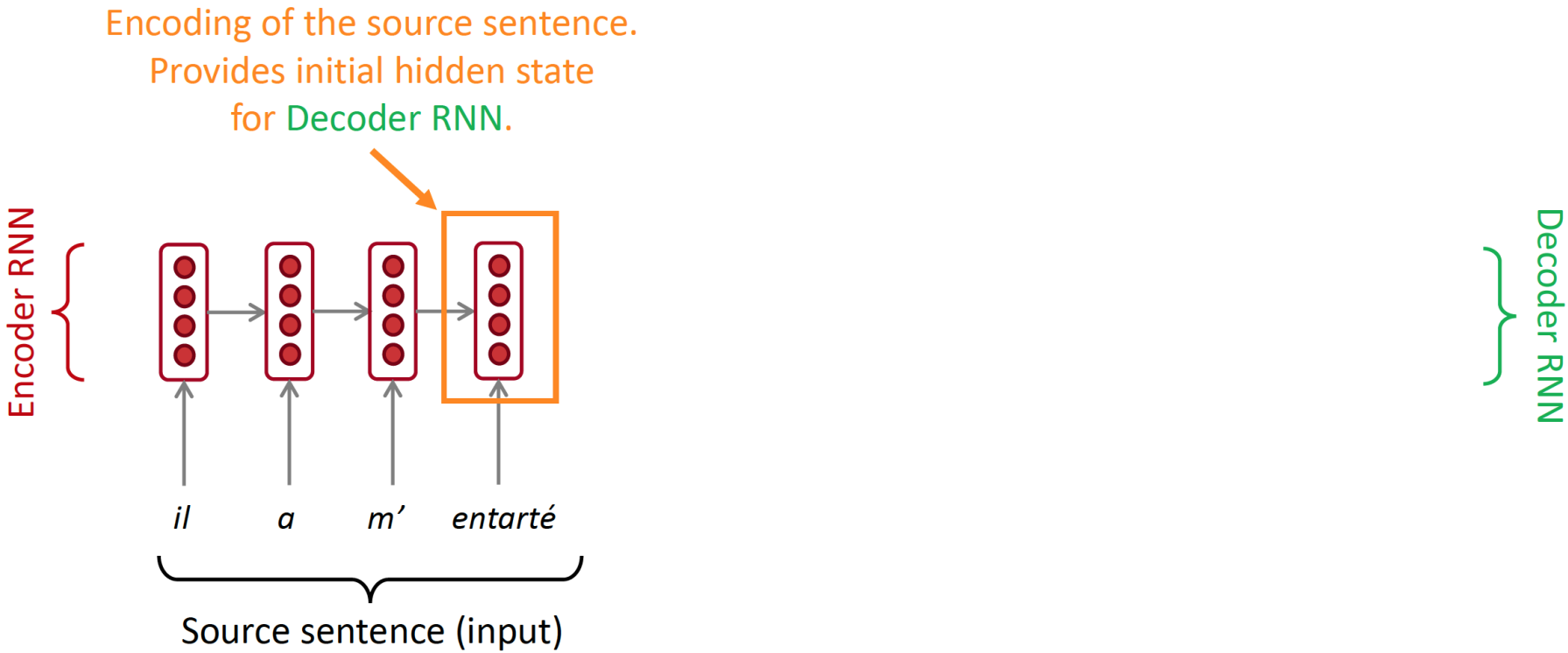
The sequence-to-sequence model



Encoder RNN produces an encoding of the source sentence.

Neural Machine Translation (NMT)

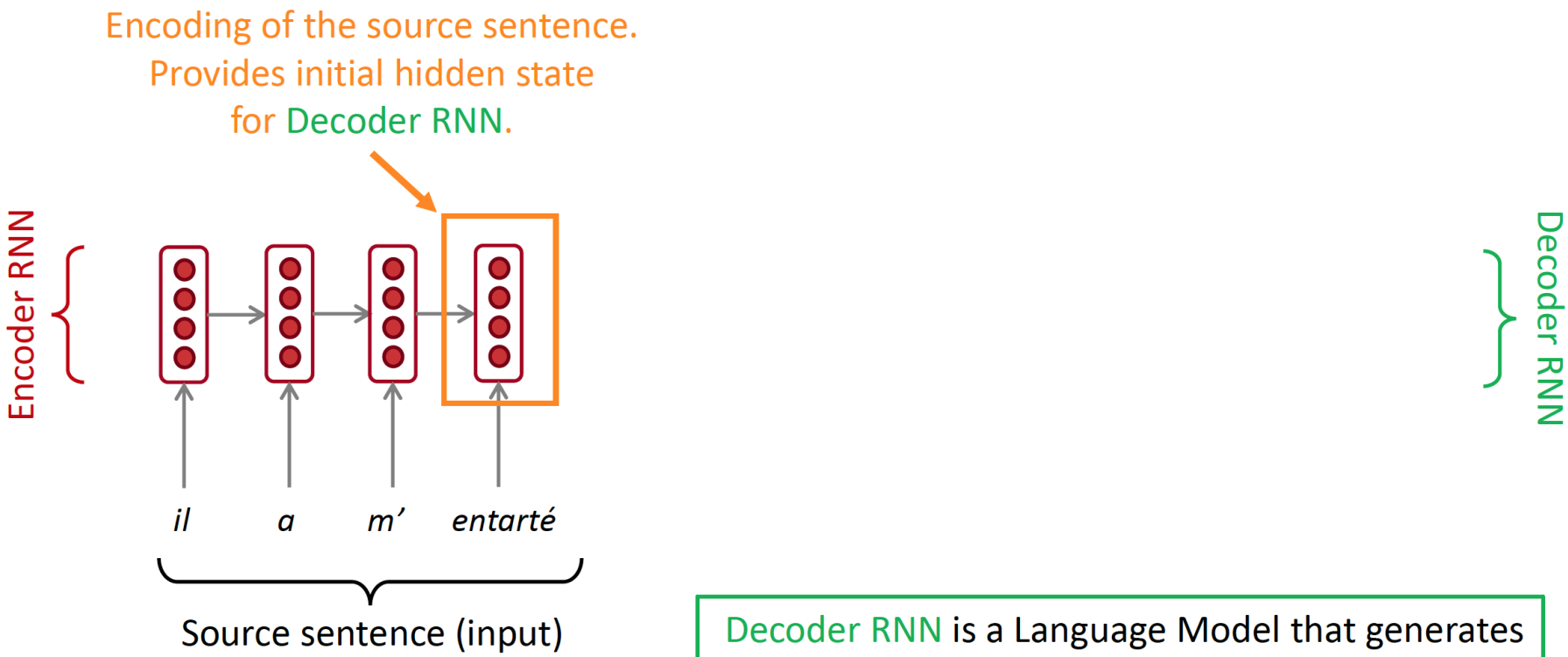
The sequence-to-sequence model



Encoder RNN produces an encoding of the source sentence.

Neural Machine Translation (NMT)

The sequence-to-sequence model

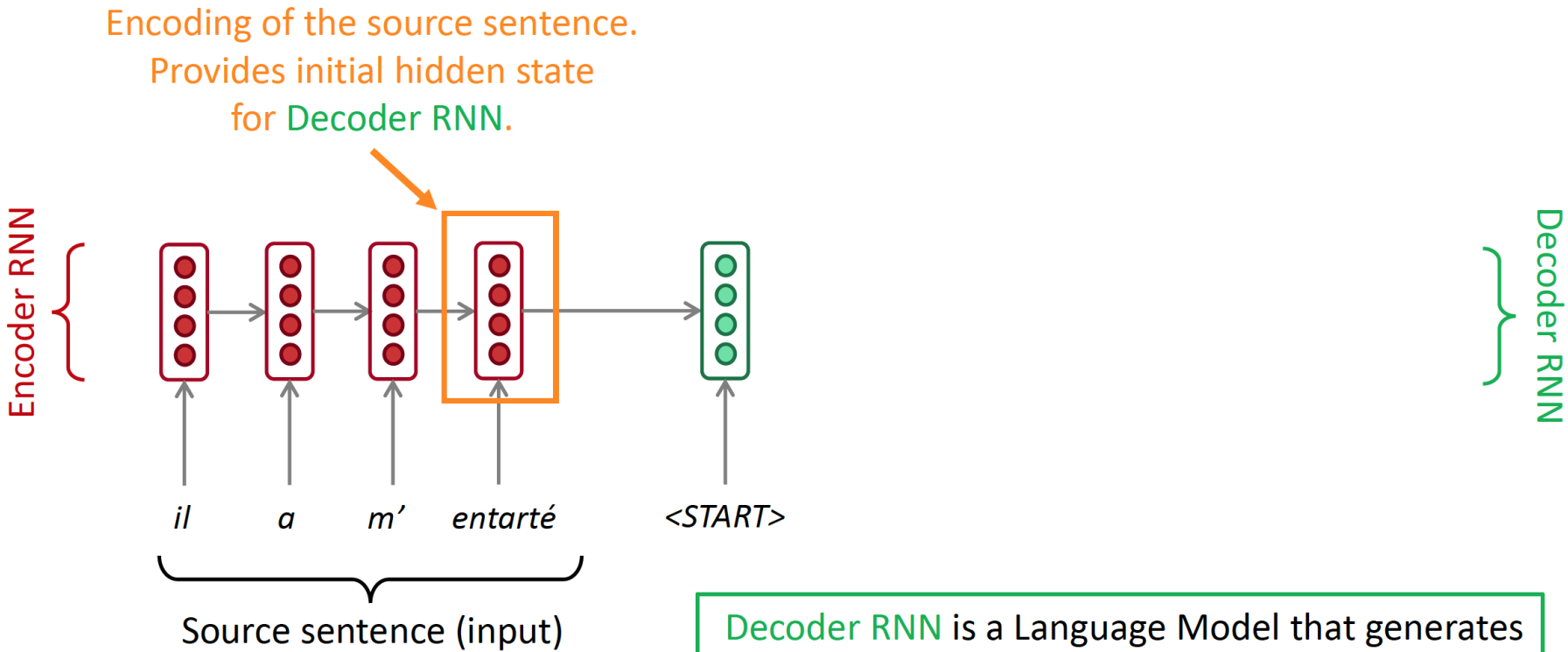


Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Neural Machine Translation (NMT)

The sequence-to-sequence model

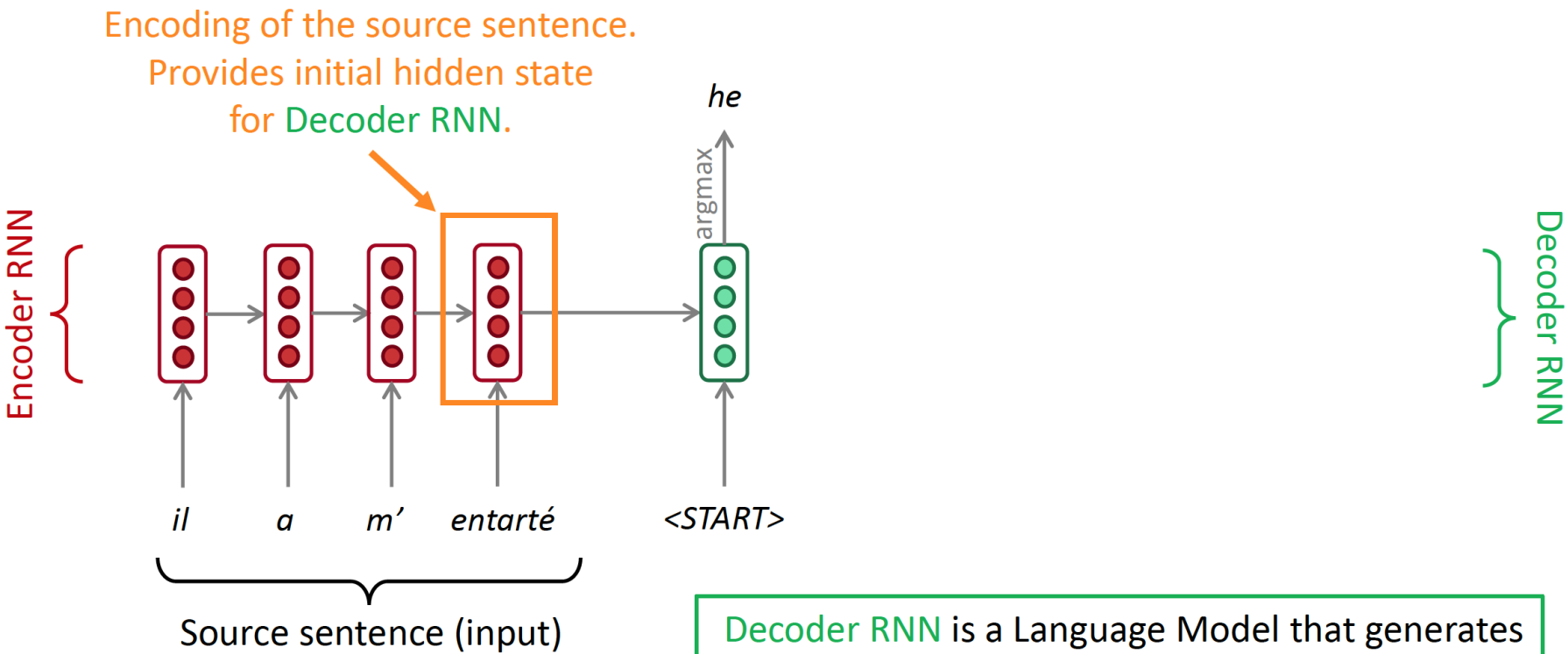


Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Neural Machine Translation (NMT)

The sequence-to-sequence model

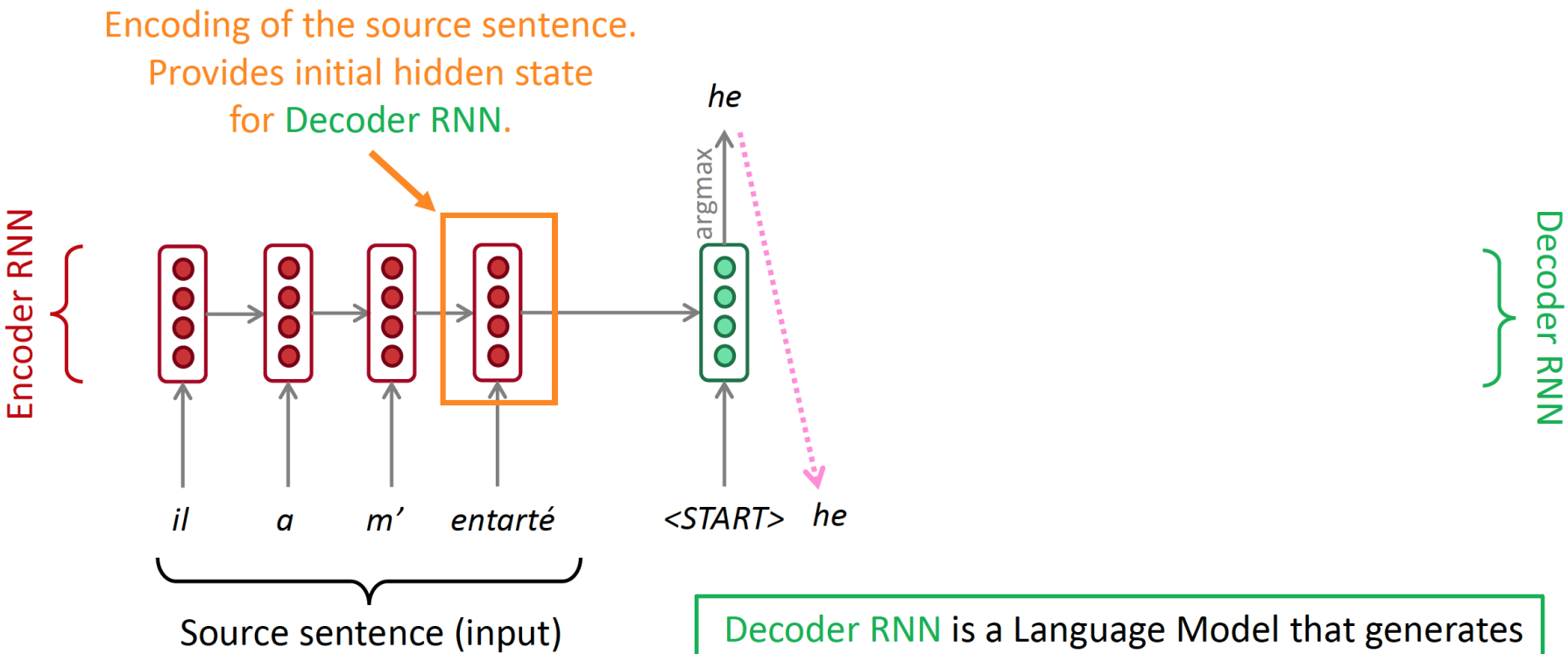


Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Encoder RNN produces an encoding of the source sentence.

Neural Machine Translation (NMT)

The sequence-to-sequence model

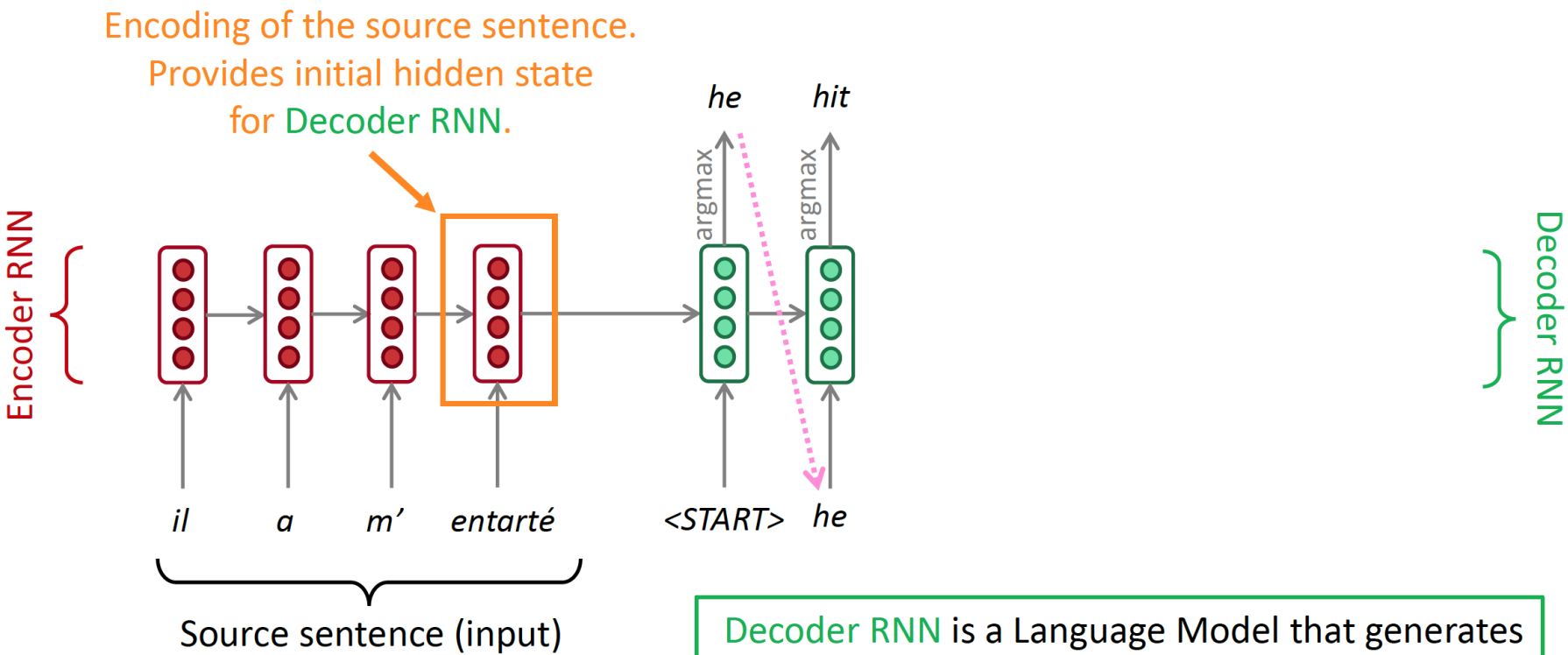


Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Encoder RNN produces an encoding of the source sentence.

Neural Machine Translation (NMT)

The sequence-to-sequence model

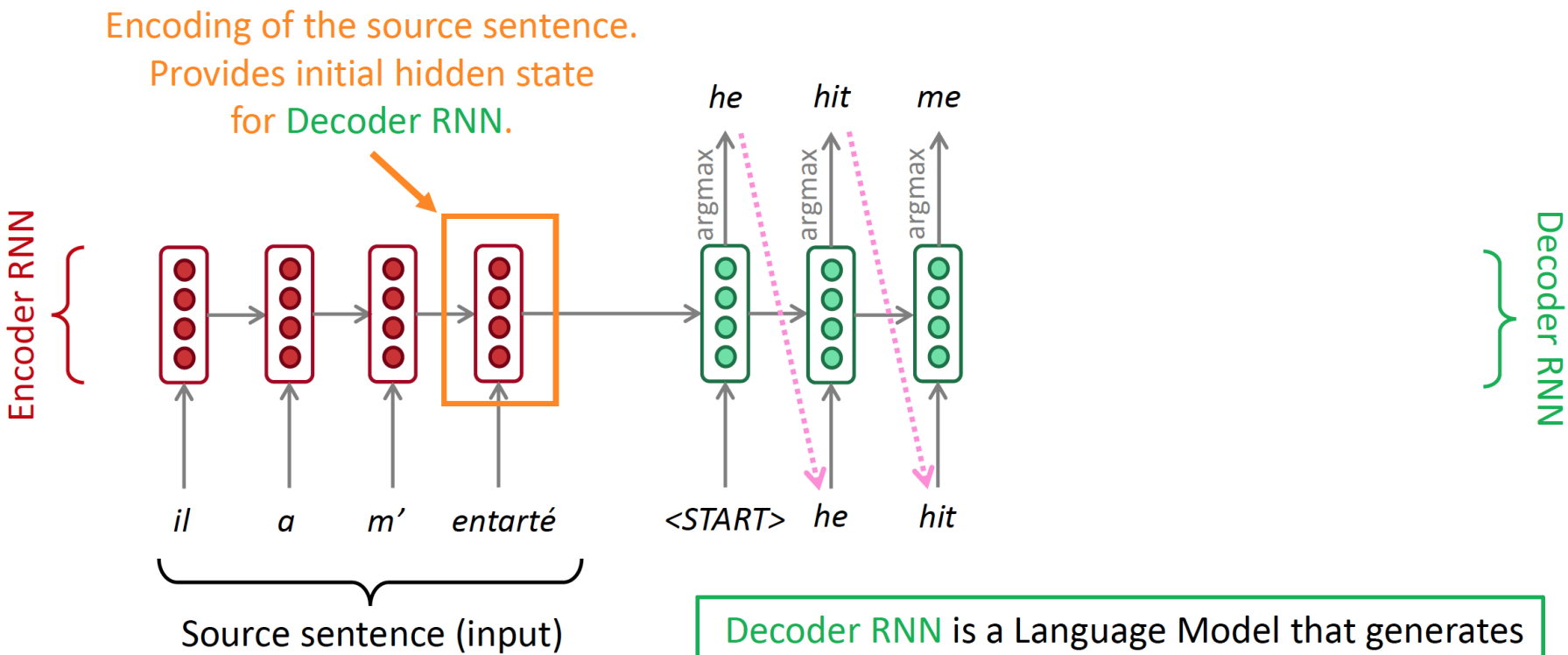


Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Neural Machine Translation (NMT)

The sequence-to-sequence model

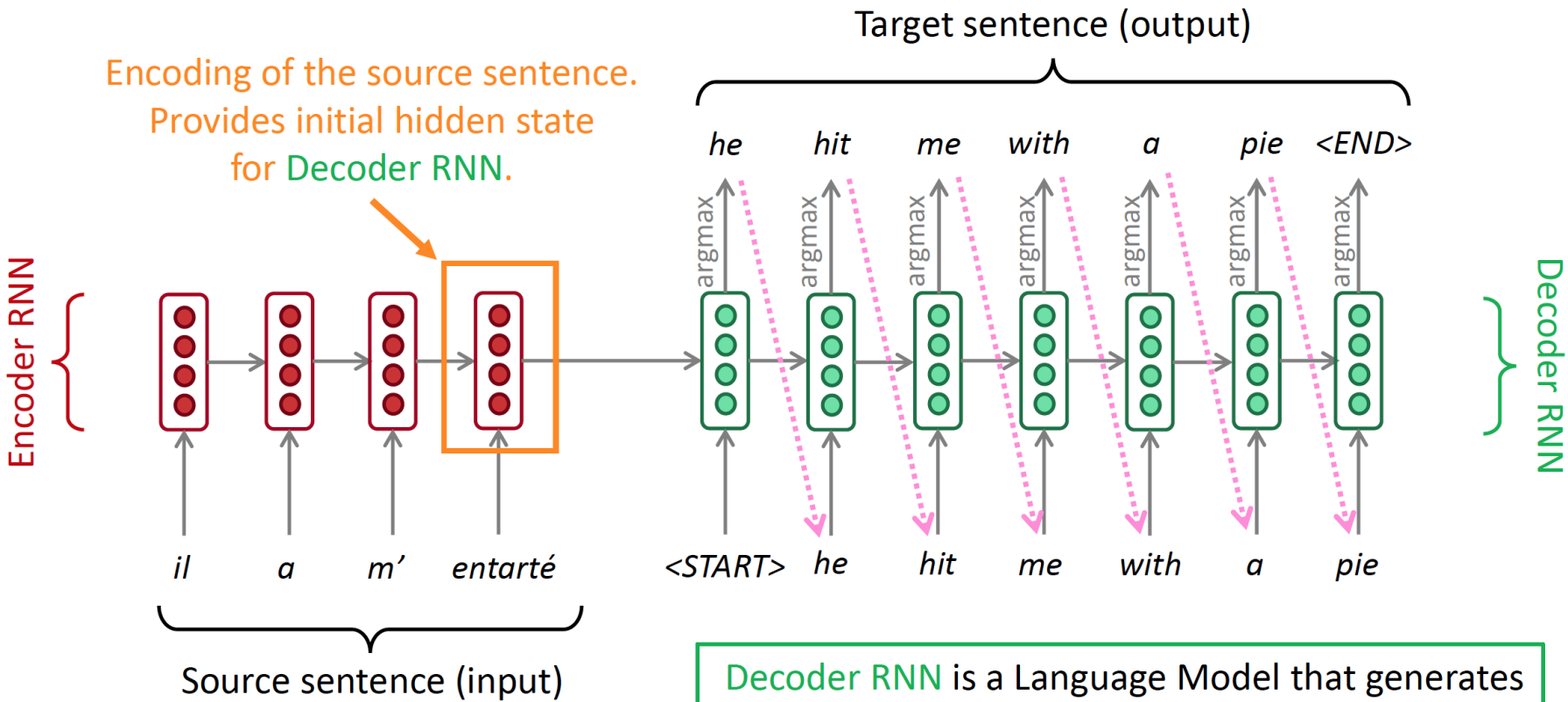


Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Neural Machine Translation (NMT)

The sequence-to-sequence model

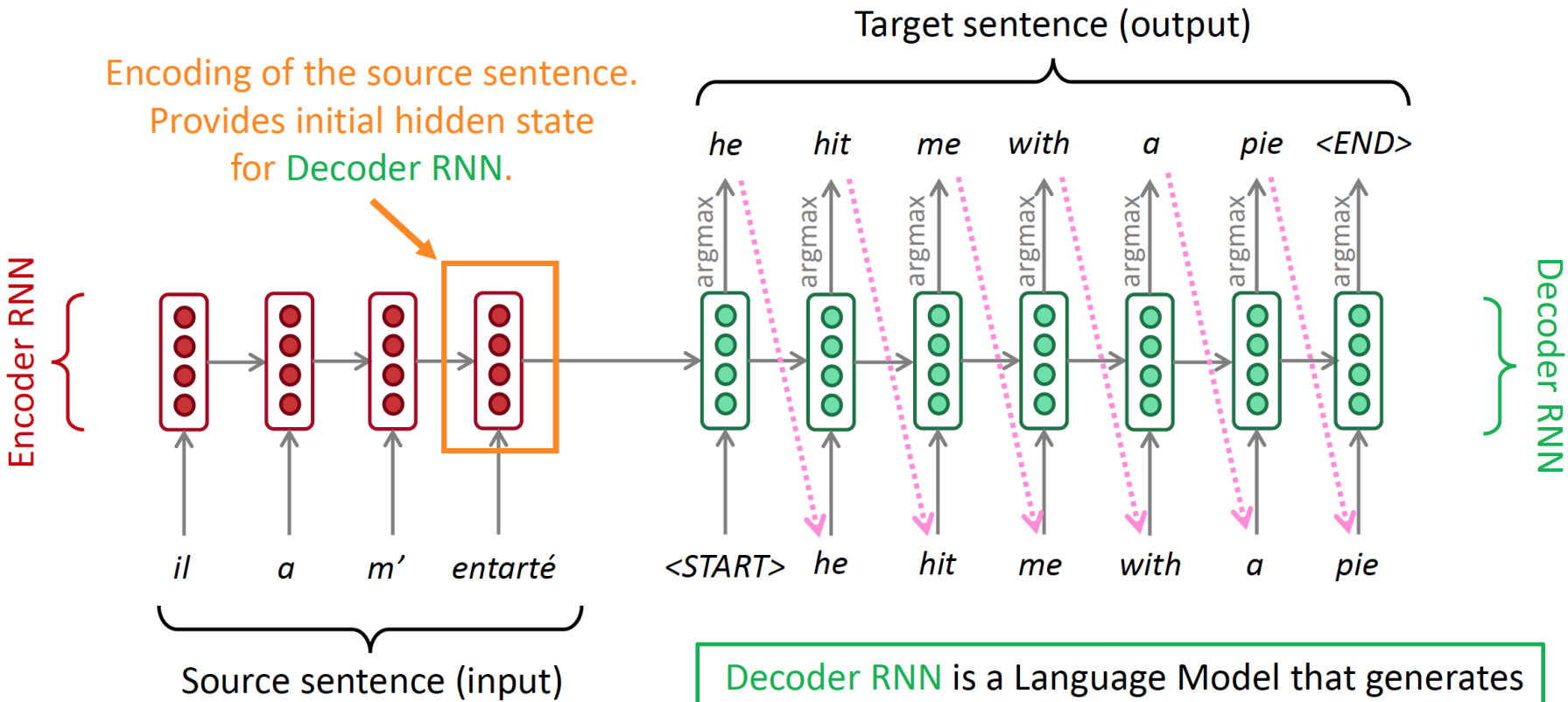


Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Encoder RNN produces an encoding of the source sentence.

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

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)
  - Simplification (input text → simplified text)
  - Dialogue (previous utterances → next utterance)
  - Keyphrase extraction/generation (input text → keyphrases 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.

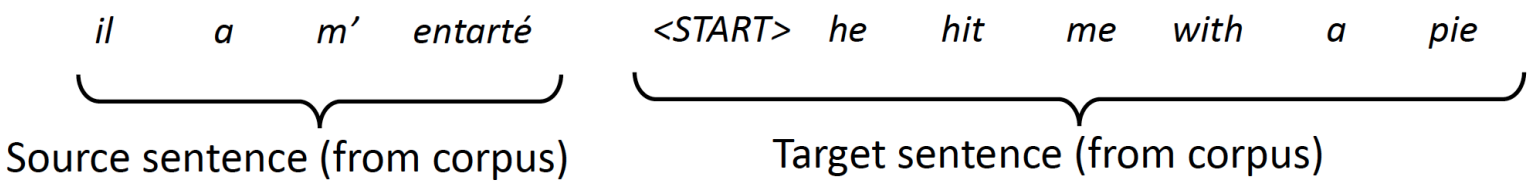
- NMT directly calculates $P(y|x)$:

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

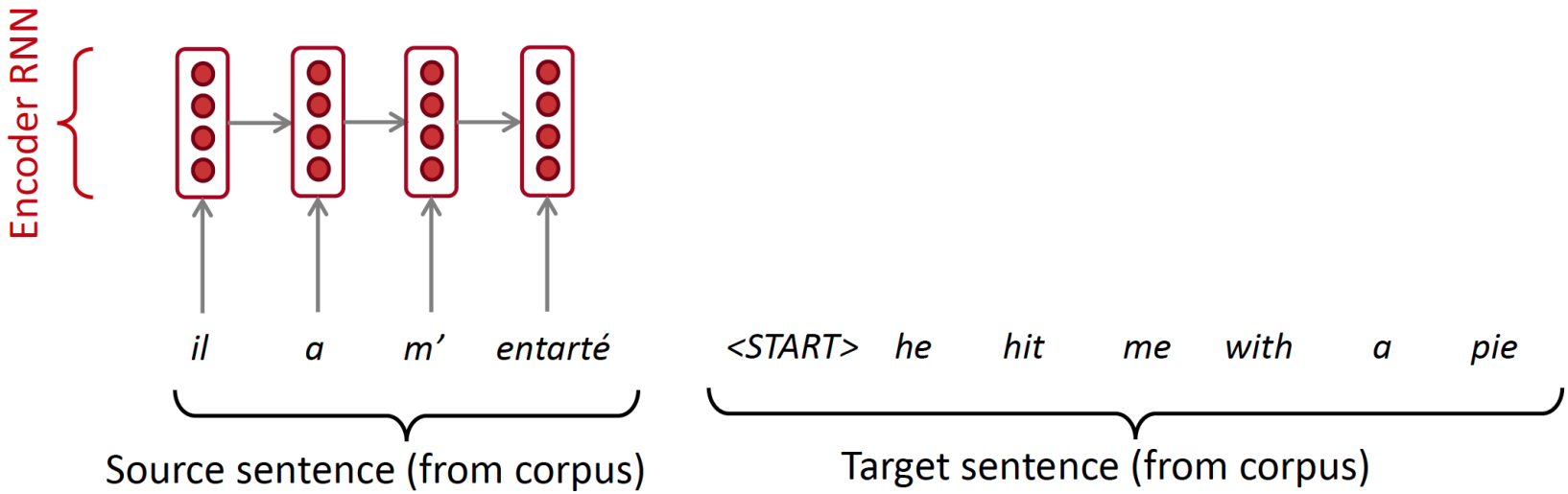
Probability of next target word, given target words so far and source sentence x

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

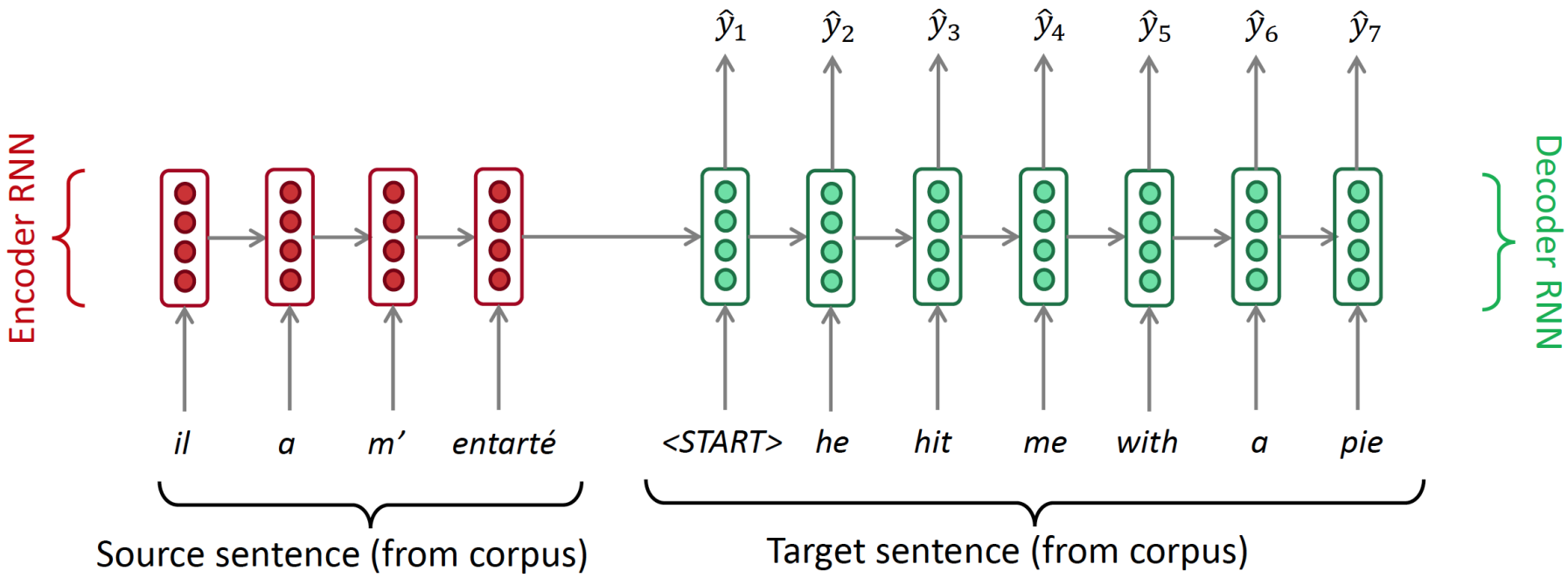
Training a Neural Machine Translation system



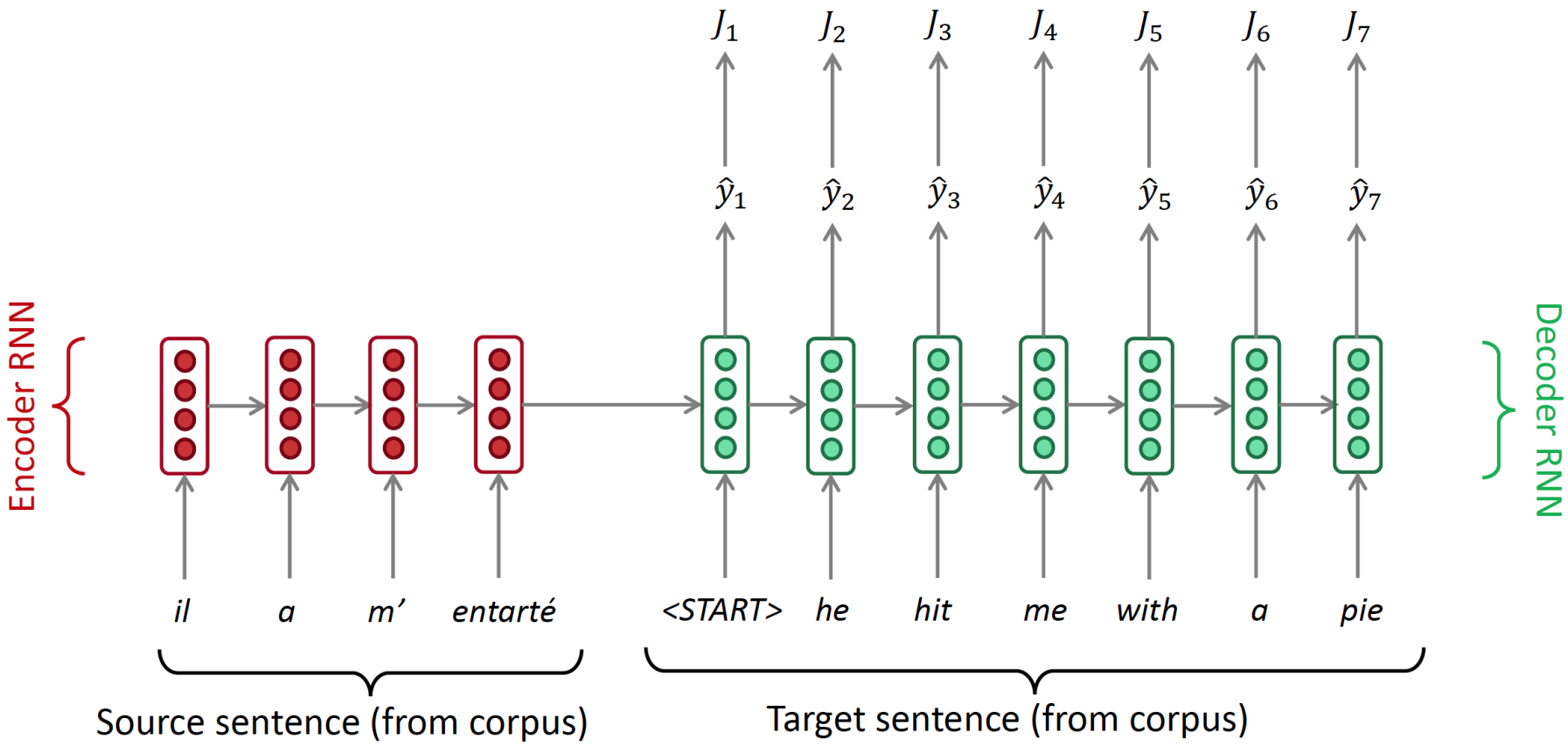
Training a Neural Machine Translation system



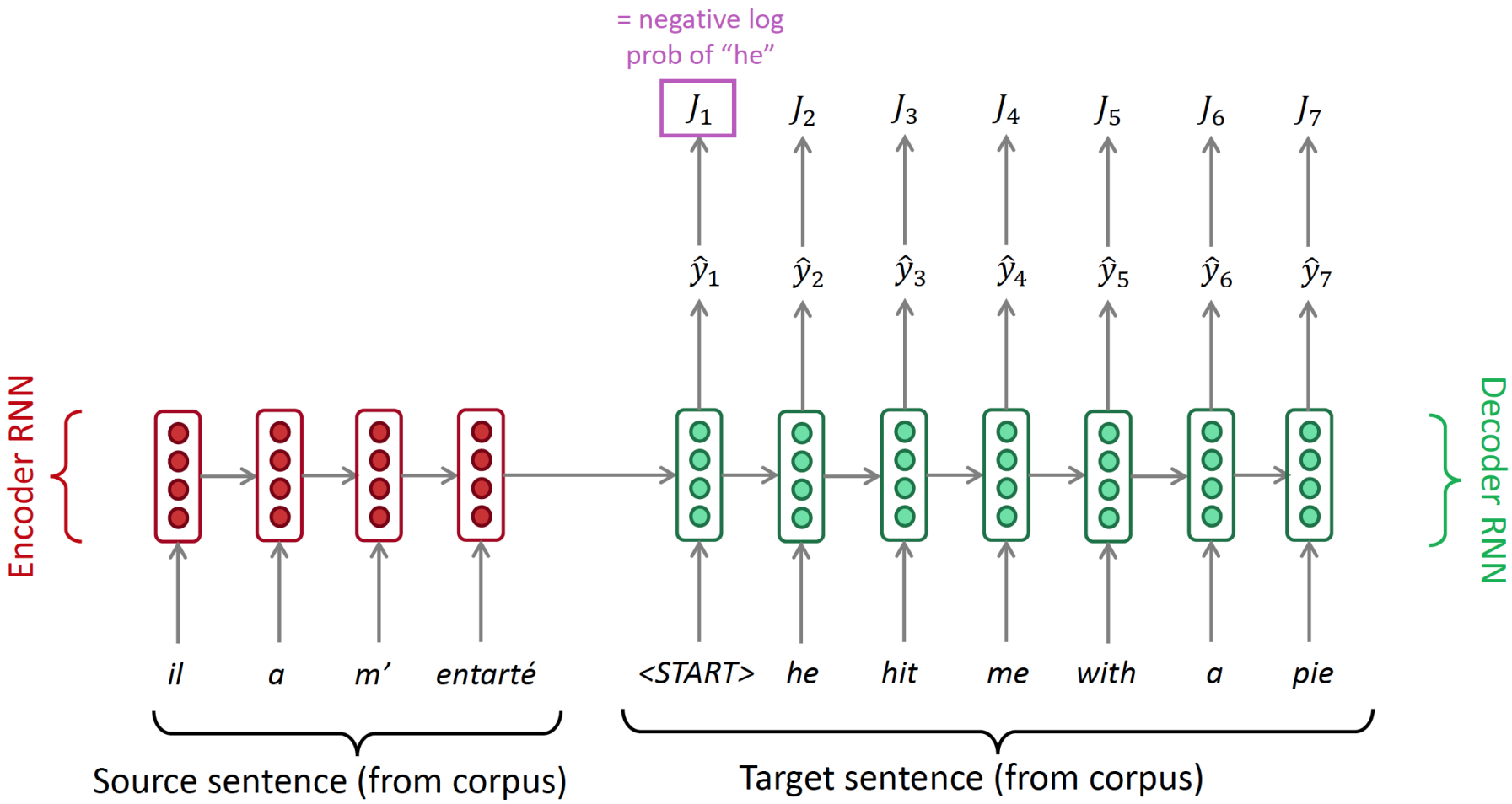
Training a Neural Machine Translation system



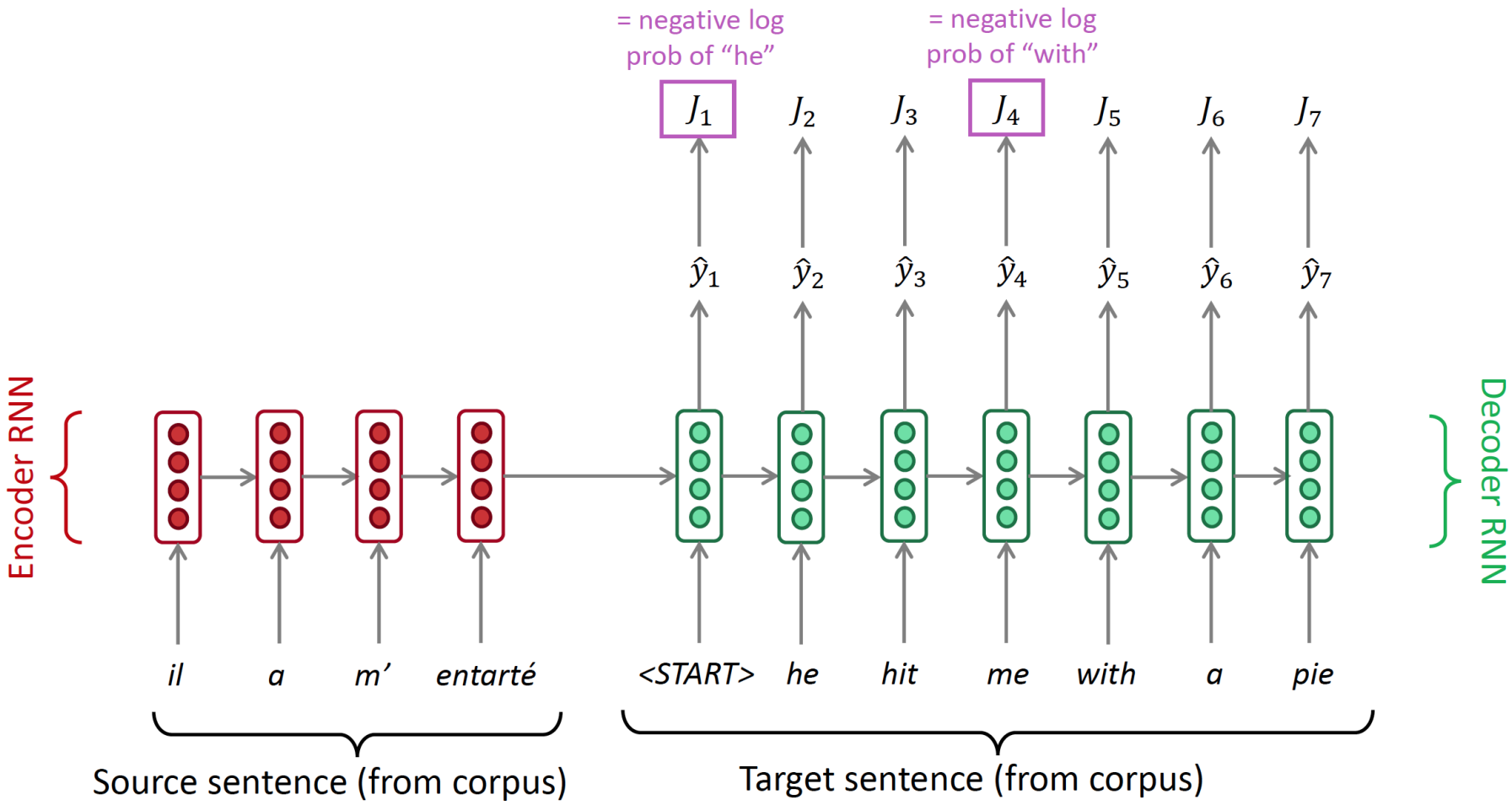
Training a Neural Machine Translation system



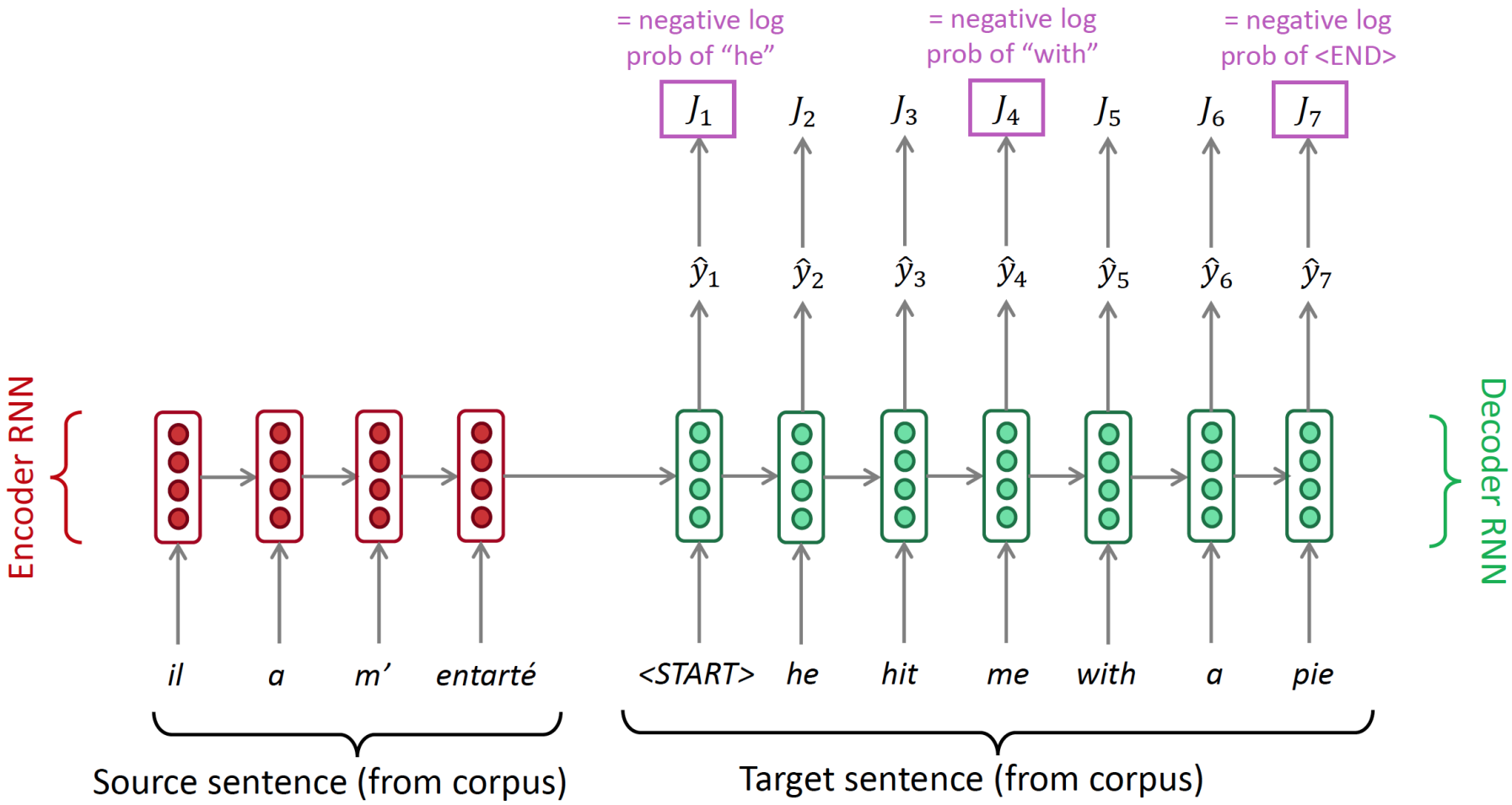
Training a Neural Machine Translation system



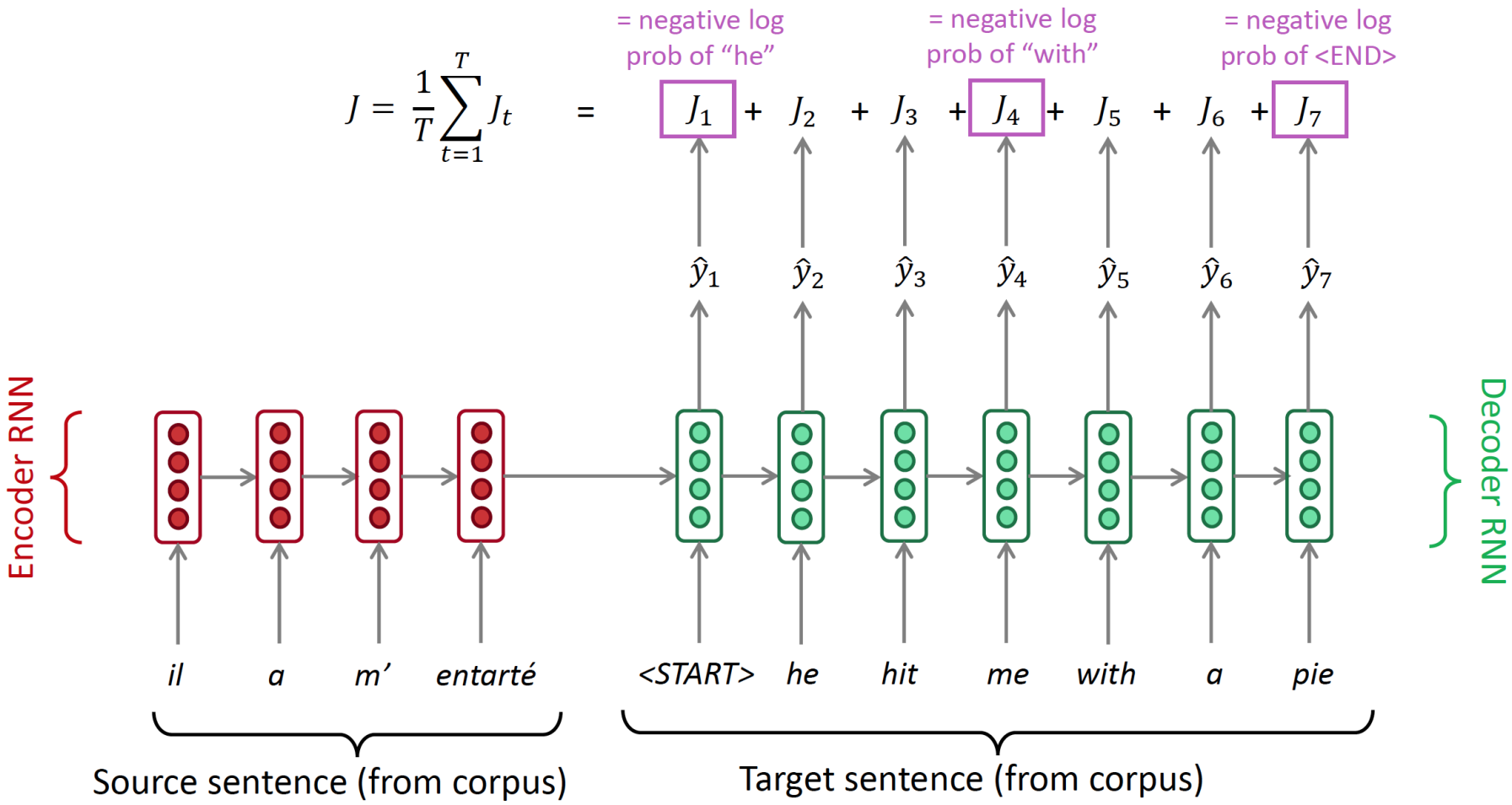
Training a Neural Machine Translation system



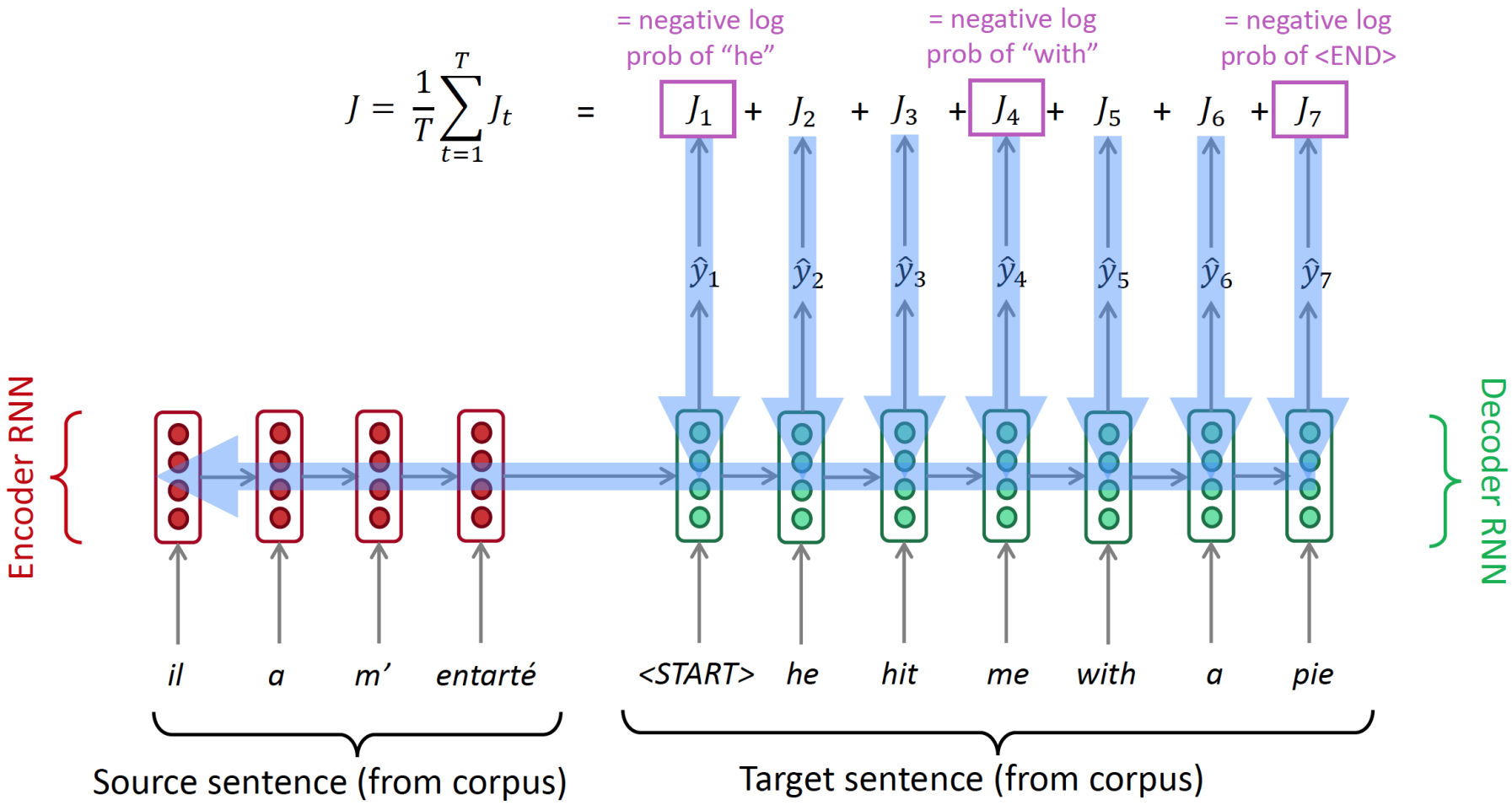
Training a Neural Machine Translation system



Training a Neural Machine Translation system



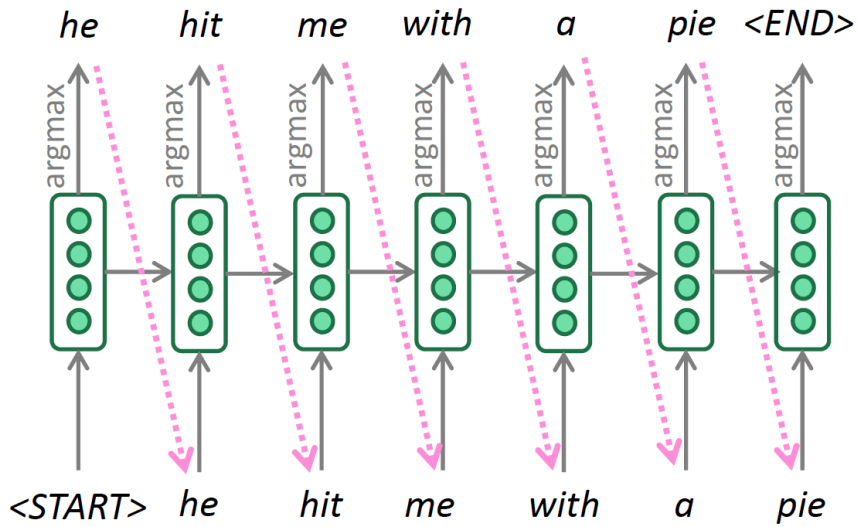
Training a Neural Machine Translation system



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

Greedy decoding

- "Decoding" 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 cannot undo decisions!
  - Input: il m'a entarté (he hit me with a pie)
  - $\rightarrow he\_\_\_$
  - $\rightarrow he\ hit\_\_\_$
  - $\rightarrow he\ hit\ a\_\_\_$ (no going back!)
- 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 are 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 the 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 translation $y_1, \cdots, y_t$ 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 it is much more efficient than exhaustive search!

Beam search decoding: example

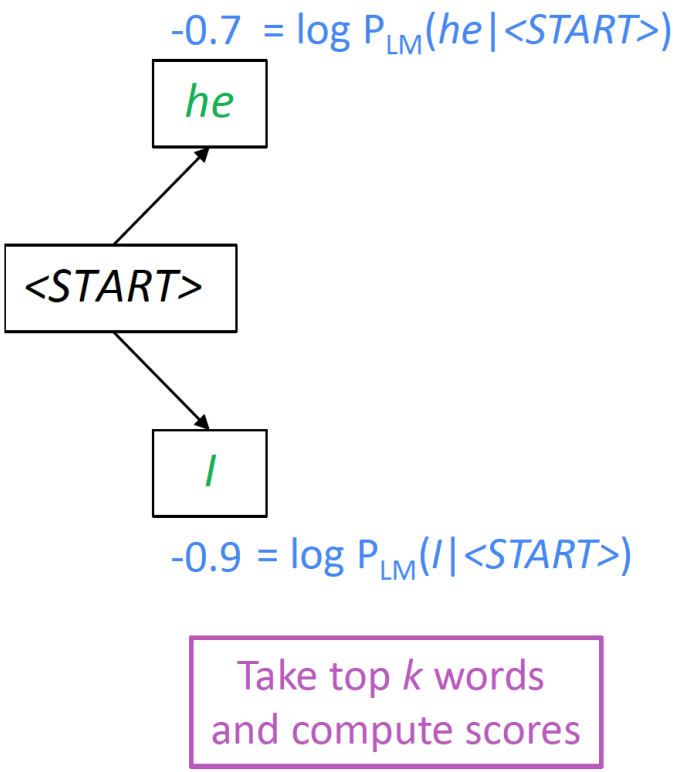
Beam size = k = 2. $score(y_1, \cdots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, \cdots, y_{i-1}, x)$

<START>

Calculate prob dist of next word

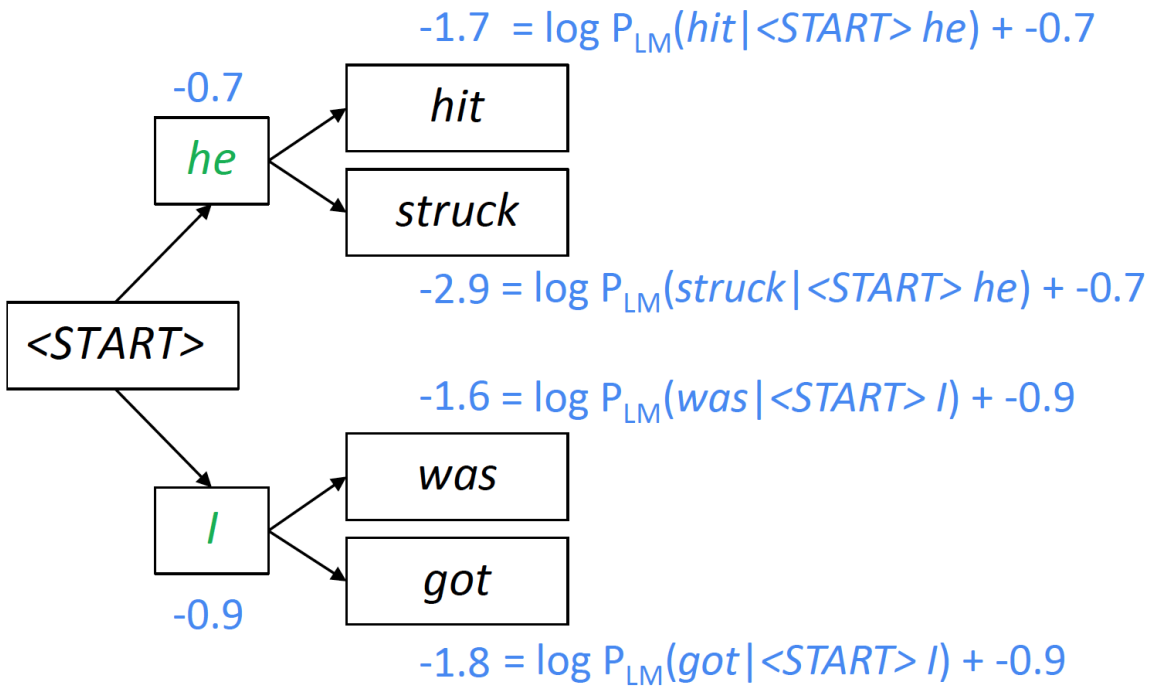
Beam search decoding: example

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



Beam search decoding: example

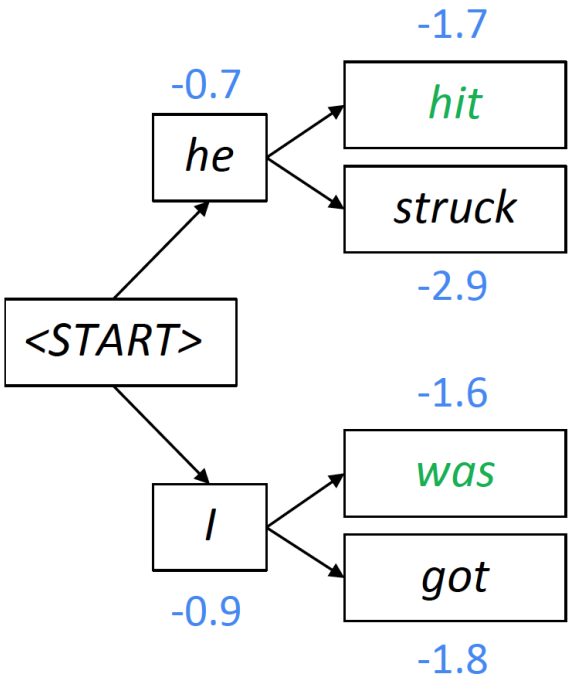
Beam size = k = 2. $score(y_1, \cdots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \cdots, y_{i-1}, x)$



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

Beam search decoding: example

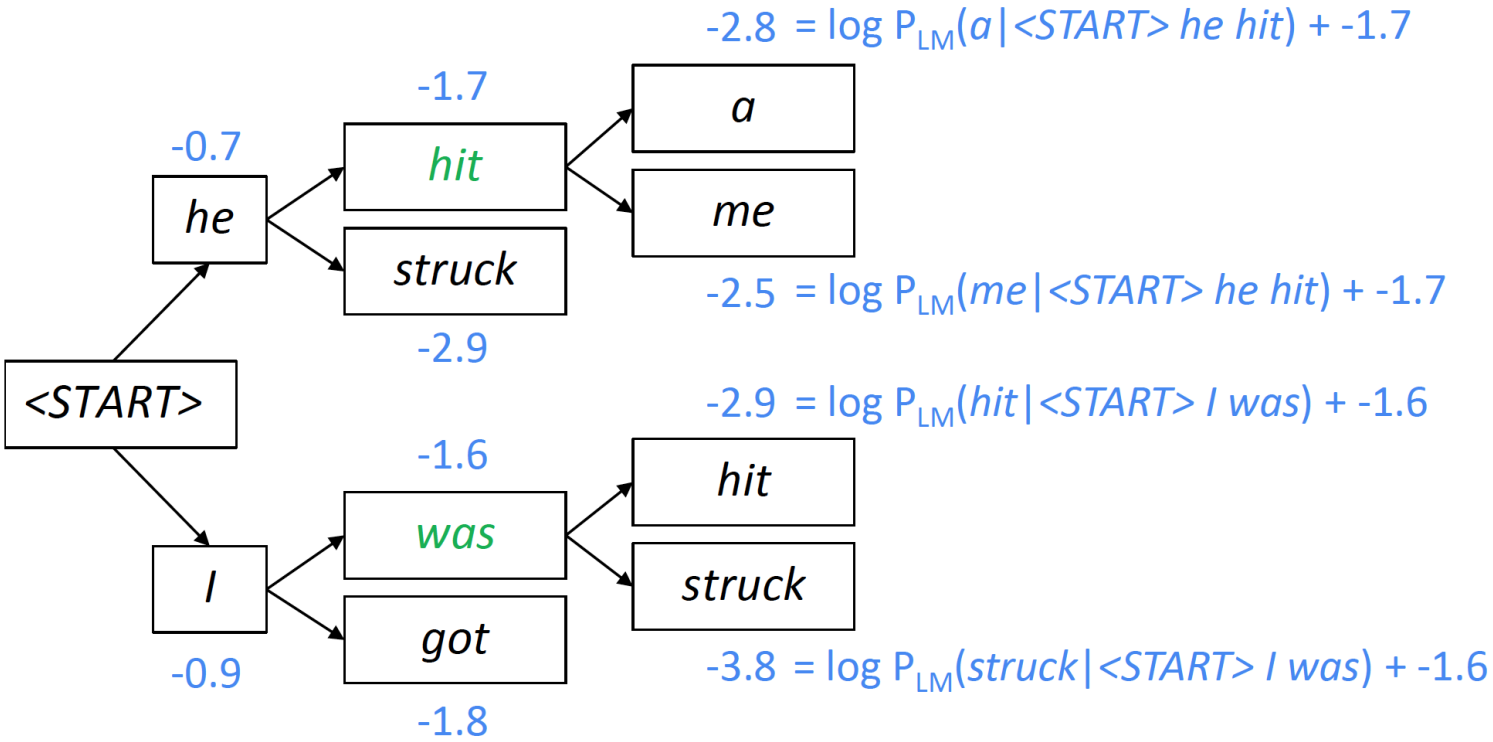
Beam size = k = 2. $score(y_1, \cdots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \cdots, y_{i-1}, x)$



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

Beam search decoding: example

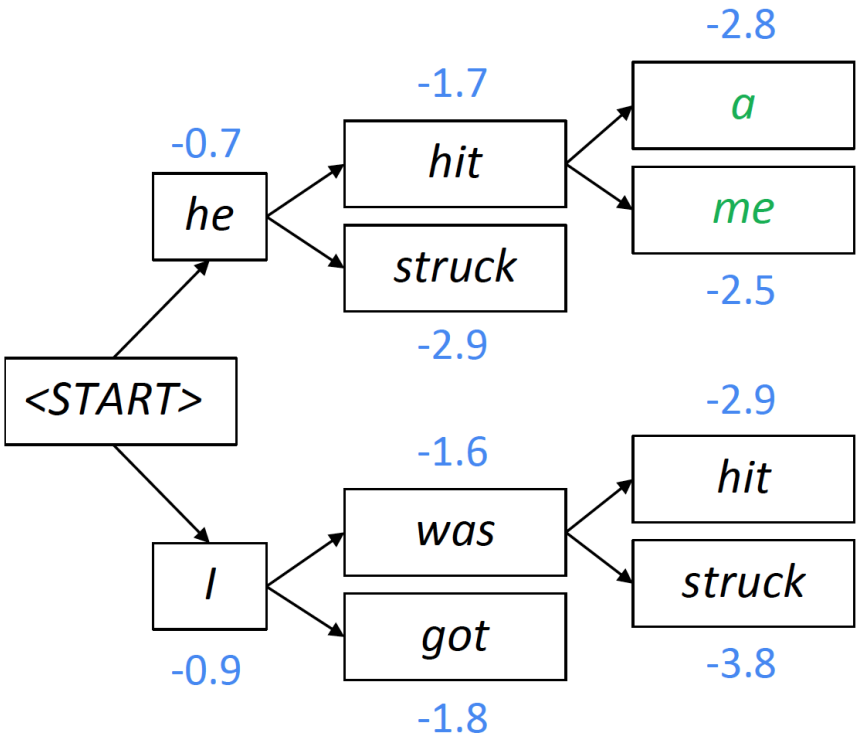
Beam size = k = 2. $score(y_1, \cdots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \cdots, y_{i-1}, x)$



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

Beam search decoding: example

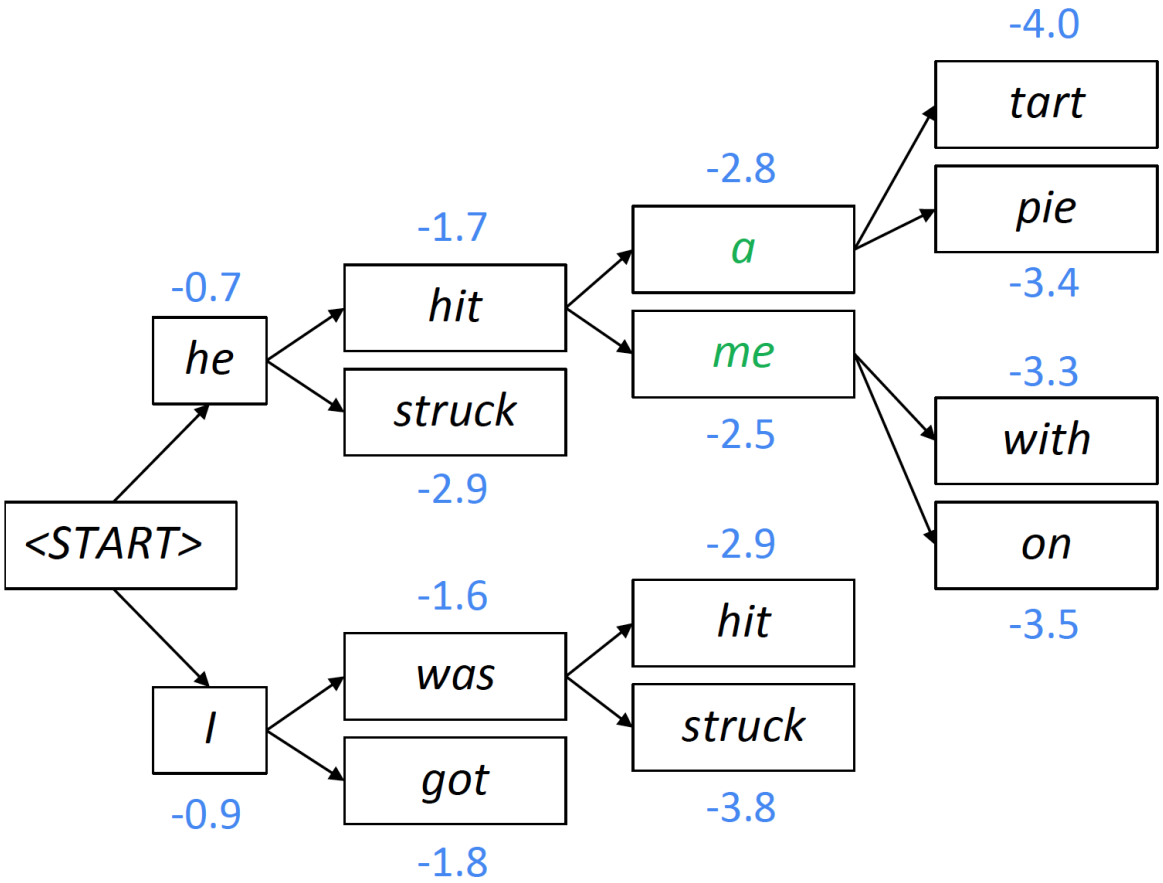
Beam size = k = 2. $score(y_1, \cdots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \cdots, y_{i-1}, x)$



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

Beam search decoding: example

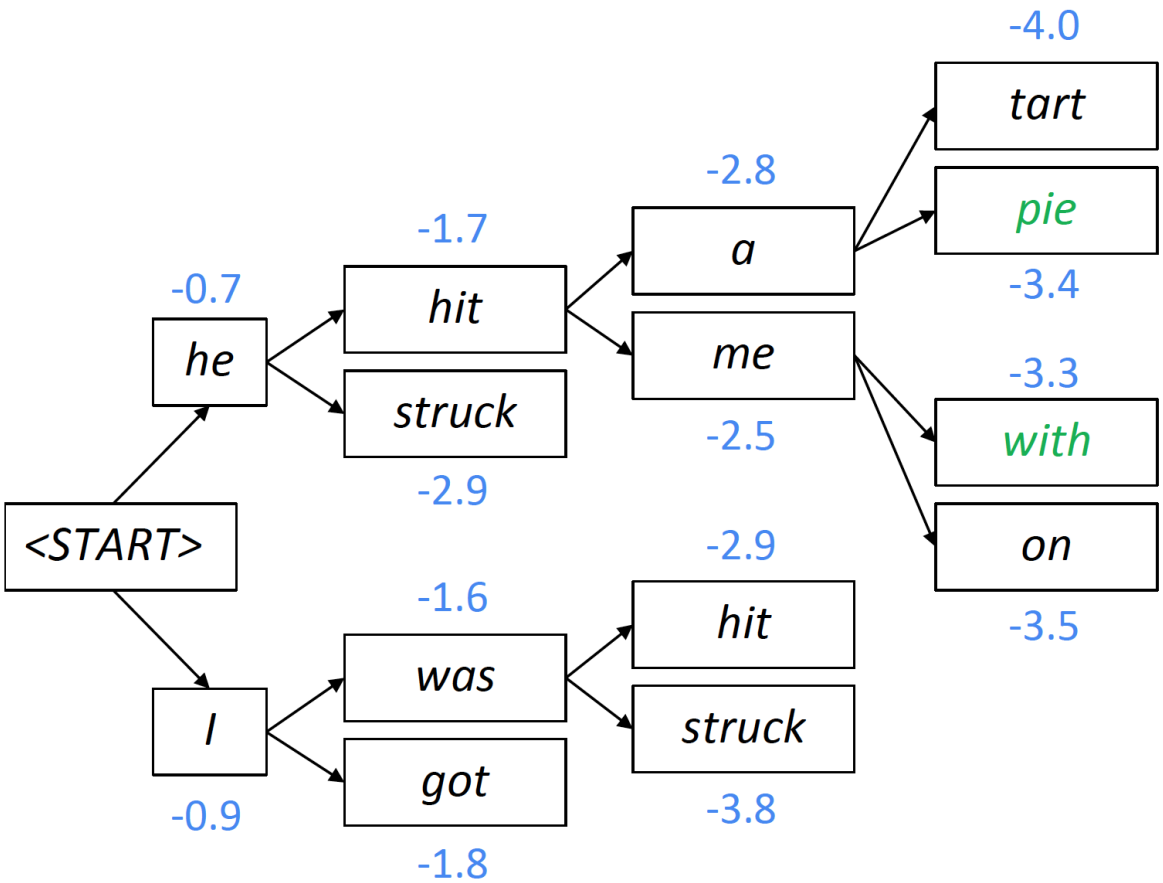
Beam size = k = 2. $score(y_1, \cdots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \cdots, y_{i-1}, x)$



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

Beam search decoding: example

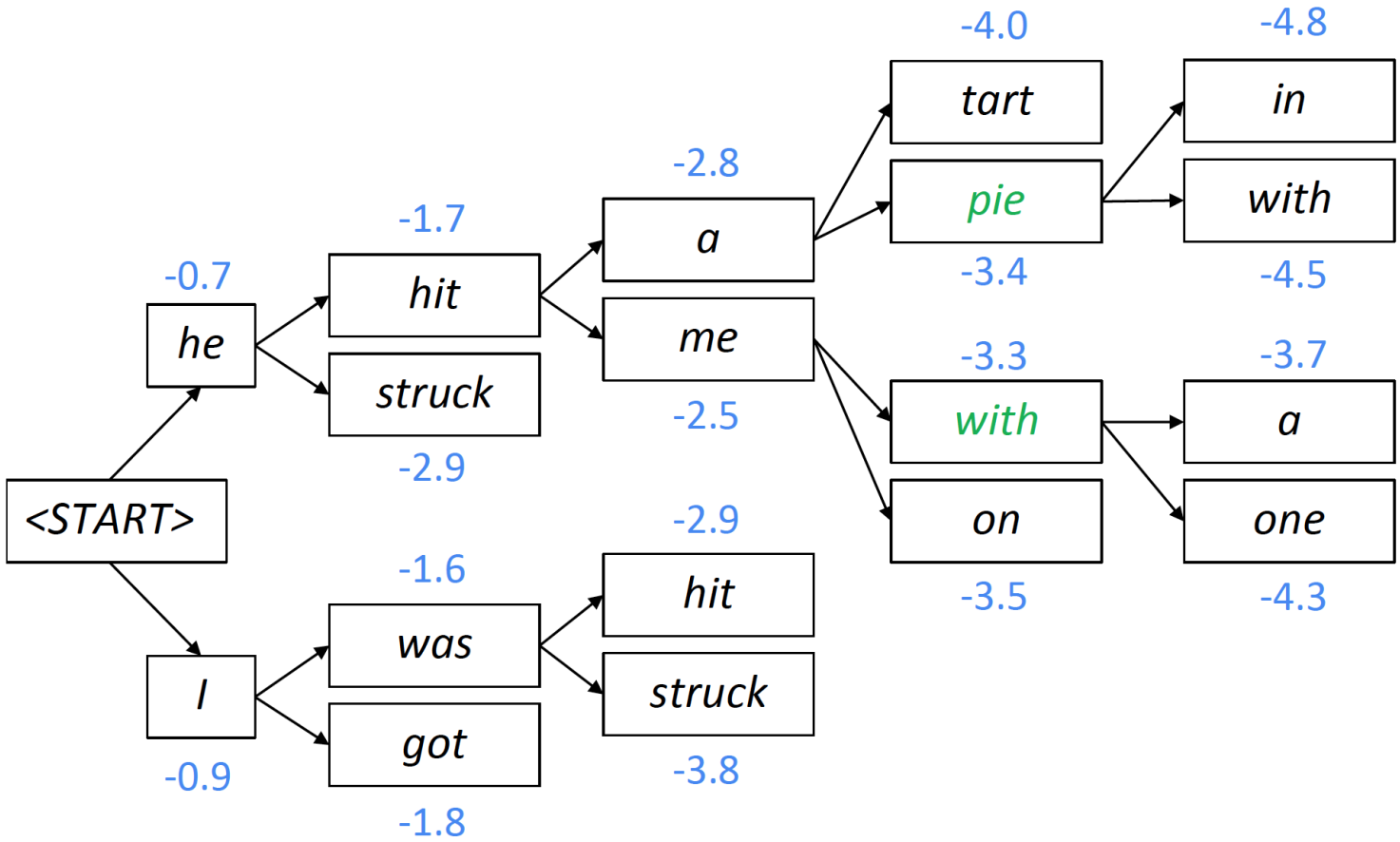
Beam size = k = 2. $score(y_1, \cdots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, \cdots, y_{i-1}, x)$



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

Beam search decoding: example

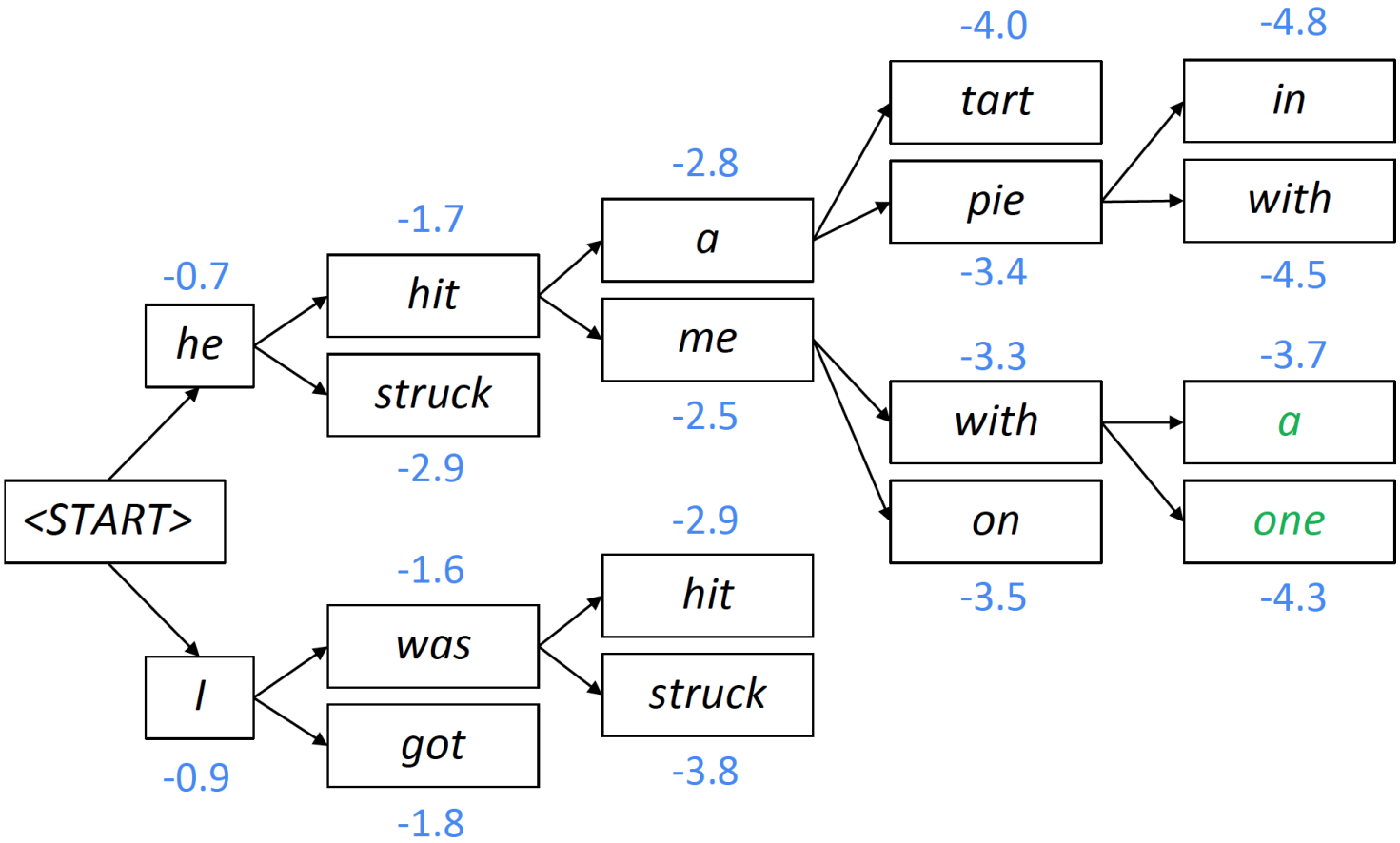
Beam size = k = 2. $score(y_1, \cdots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \cdots, y_{i-1}, x)$



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

Beam search decoding: example

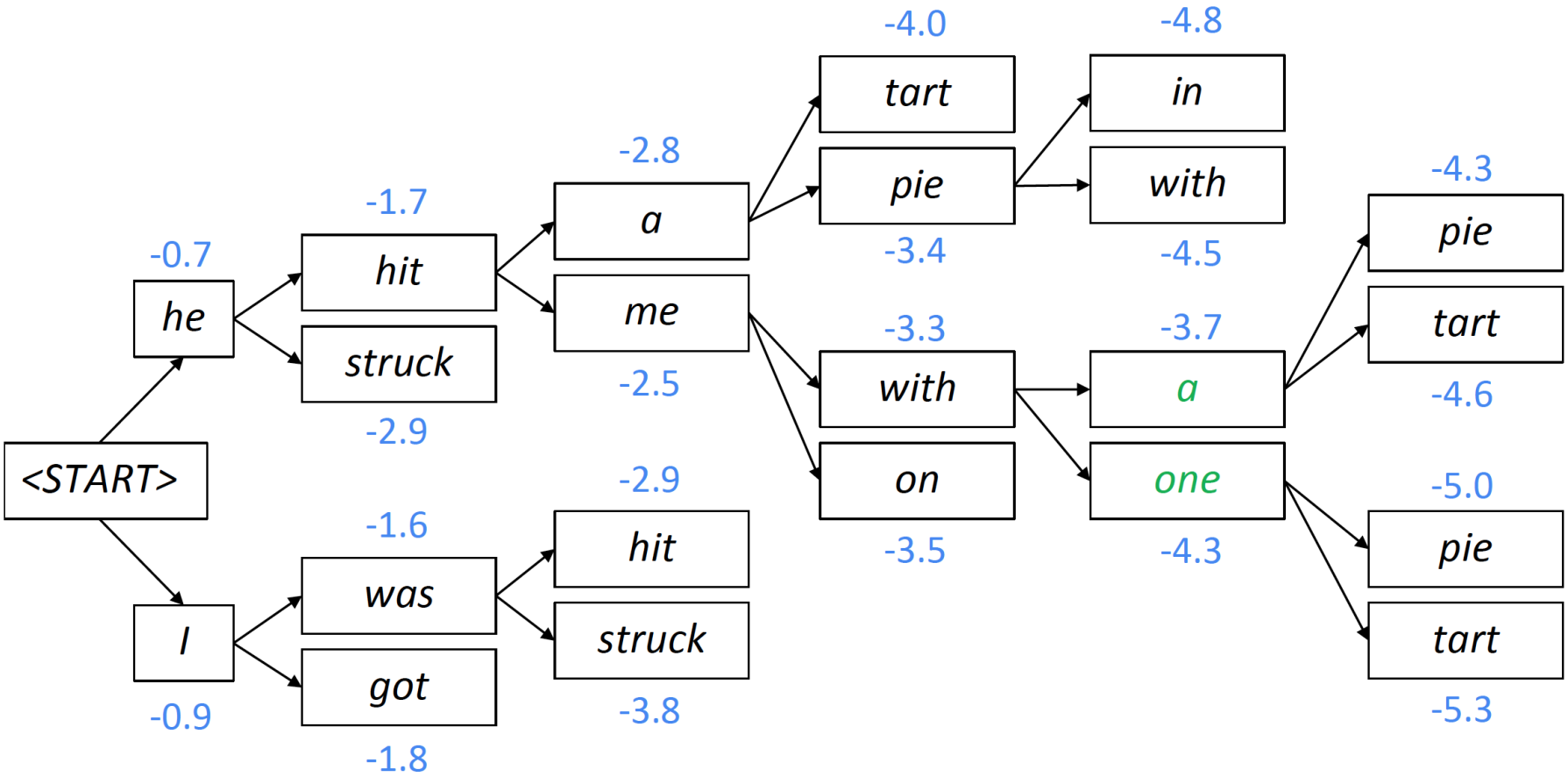
Beam size = k = 2. $score(y_1, \cdots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, \cdots, y_{i-1}, x)$



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

Beam search decoding: example

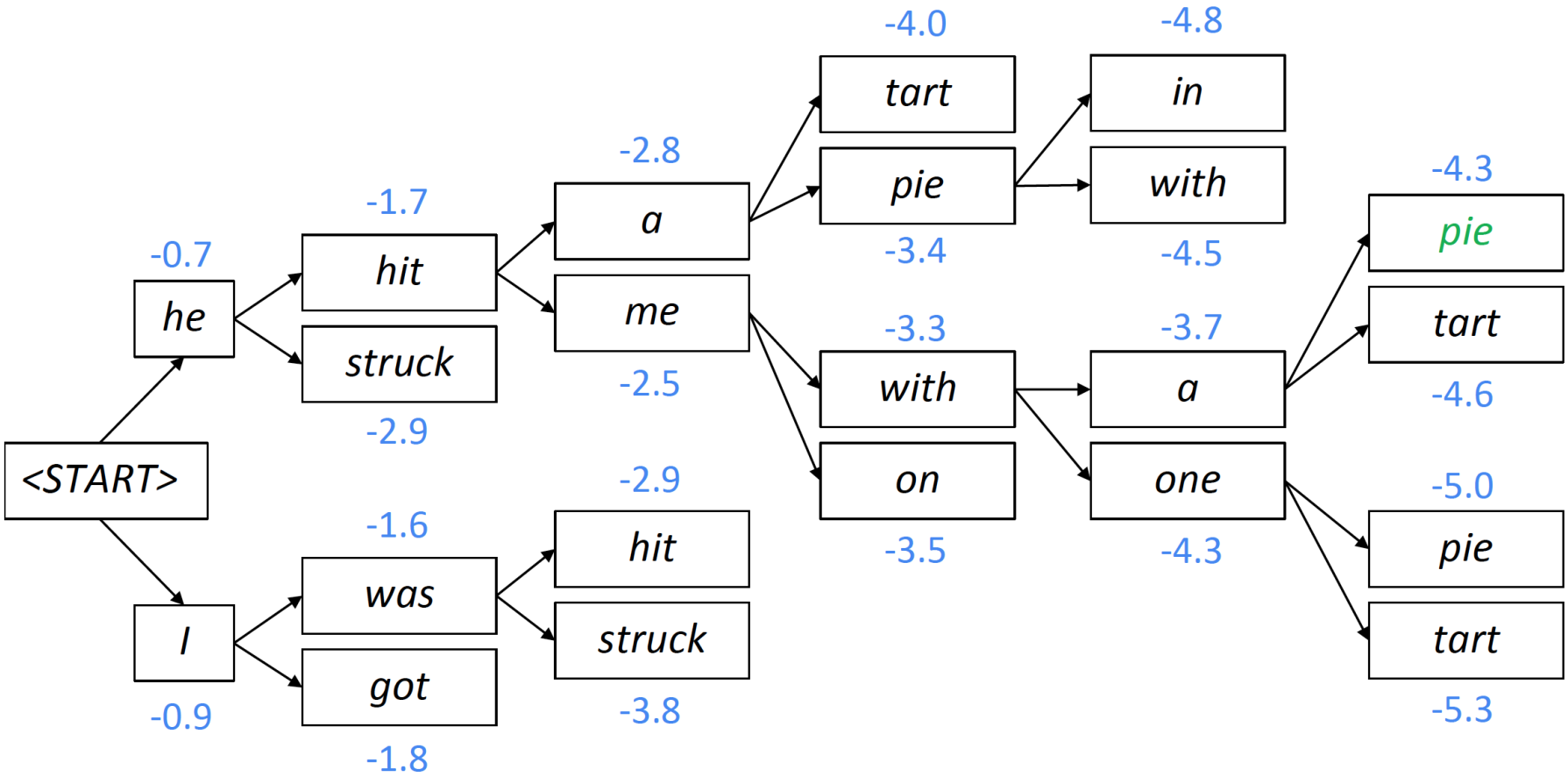
Beam size = k = 2. $score(y_1, \cdots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \cdots, y_{i-1}, x)$



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

Beam search decoding: example

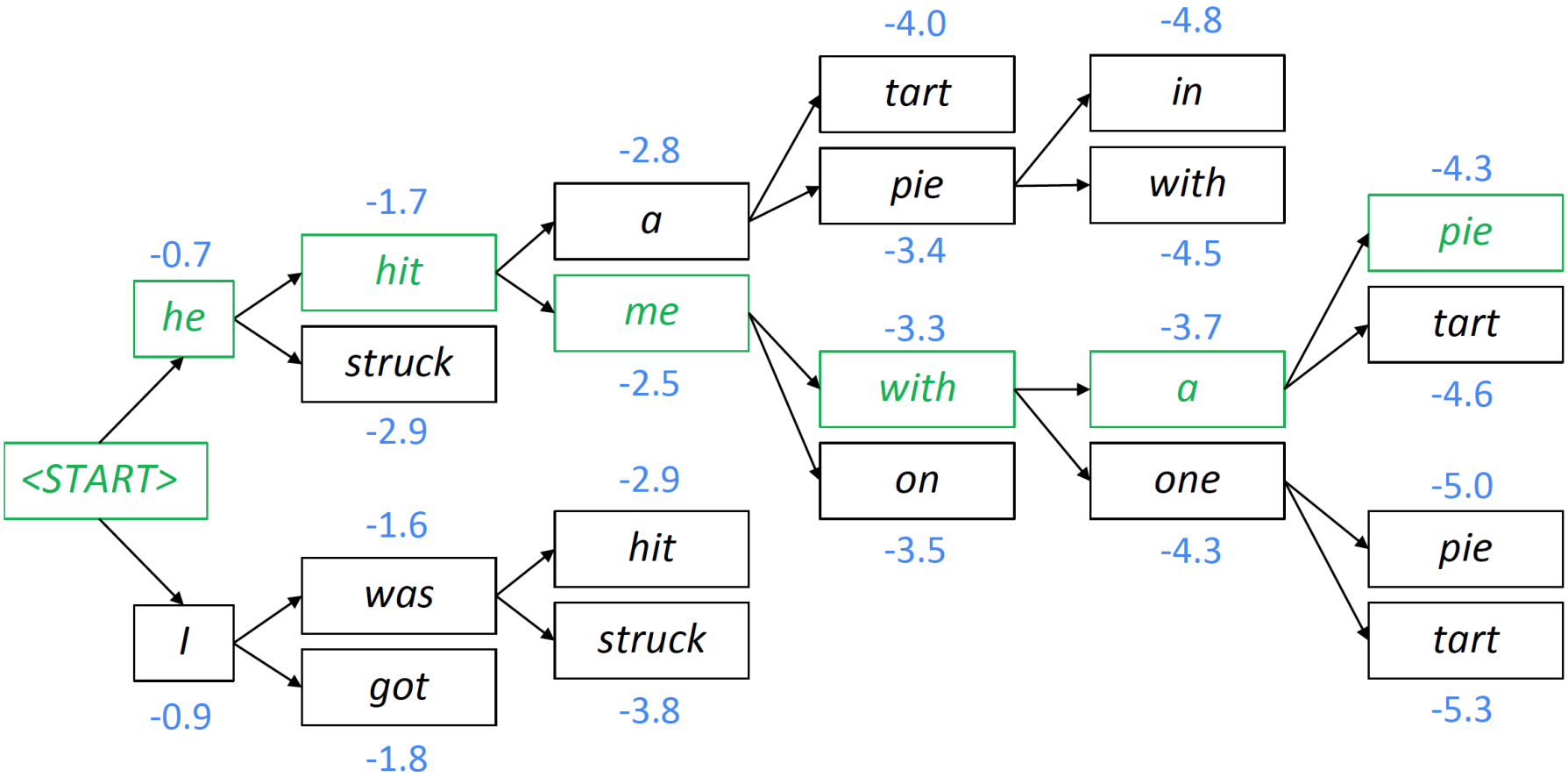
Beam size = k = 2. $score(y_1, \cdots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \cdots, y_{i-1}, x)$



This is the top-scoring hypothesis!

Beam search decoding: example

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



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

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

- Problem with this: longer hypotheses have lower scores
- Fix: Normalize by length.

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

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)

- 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

NMT: the biggest success story of NLP Deep Learning

- 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

Is Machine Translation solved?

- No!
- Many difficulties remain:
  - Out-of-vocabulary words
  - Domain mismatch between train and test data
  - Maintaining context over longer text
  - Low-resource language pairs
  - NMT picks up biases in training data

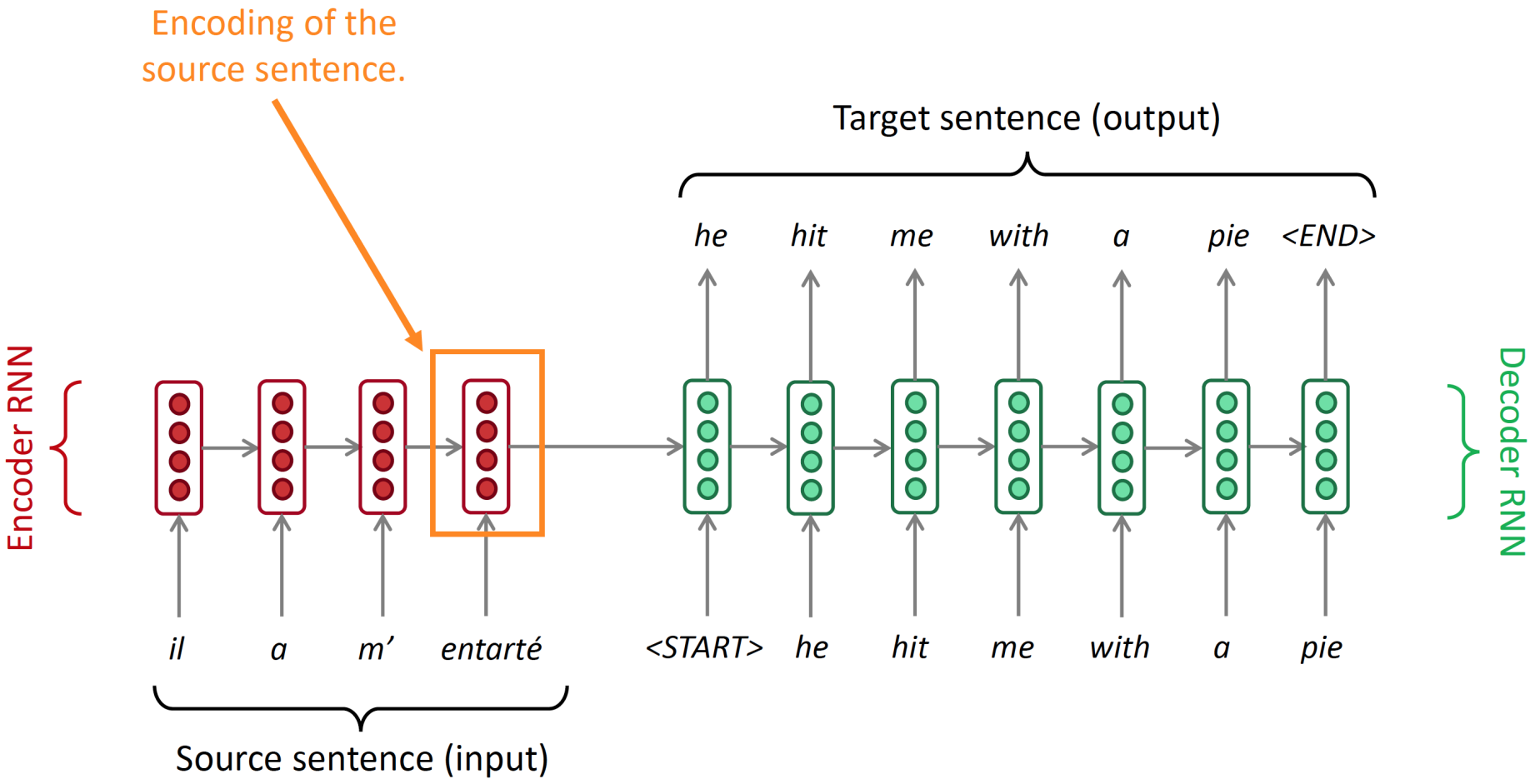
NMT research continues

NMT is the flagship task for NLP Deep Learning

- NMT research has pioneered many of the recent innovations of NLP Deep Learning
- In 2020: NMT research continues to thrive
  - Researchers have found many, many improvements to the "vanilla" seq2seq NMT system
  - But one improvement is so integral that it is the new vanilla... **ATTENTION**.

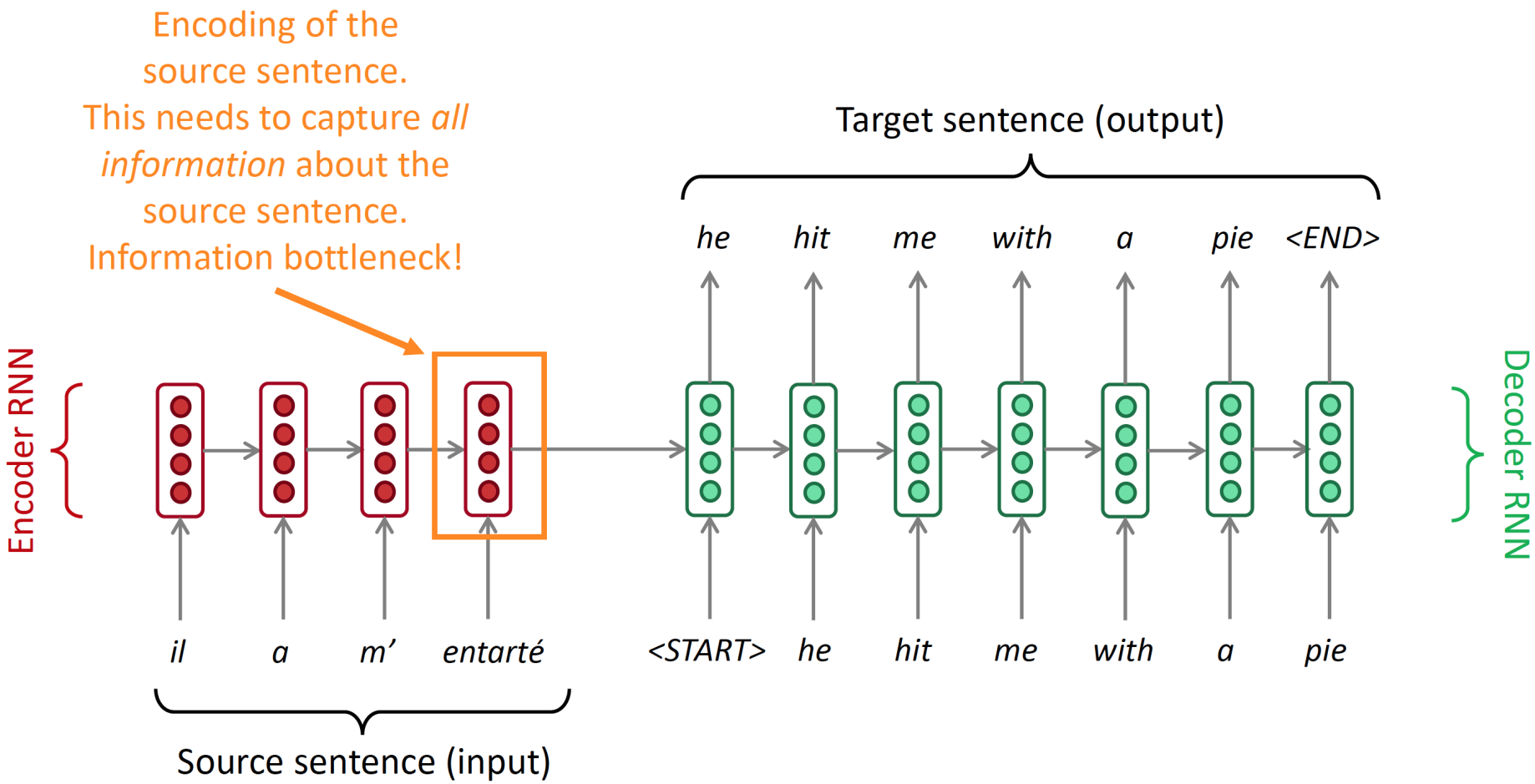
Attention

Sequence-to-sequence: the bottleneck problem



Problems with this architecture?

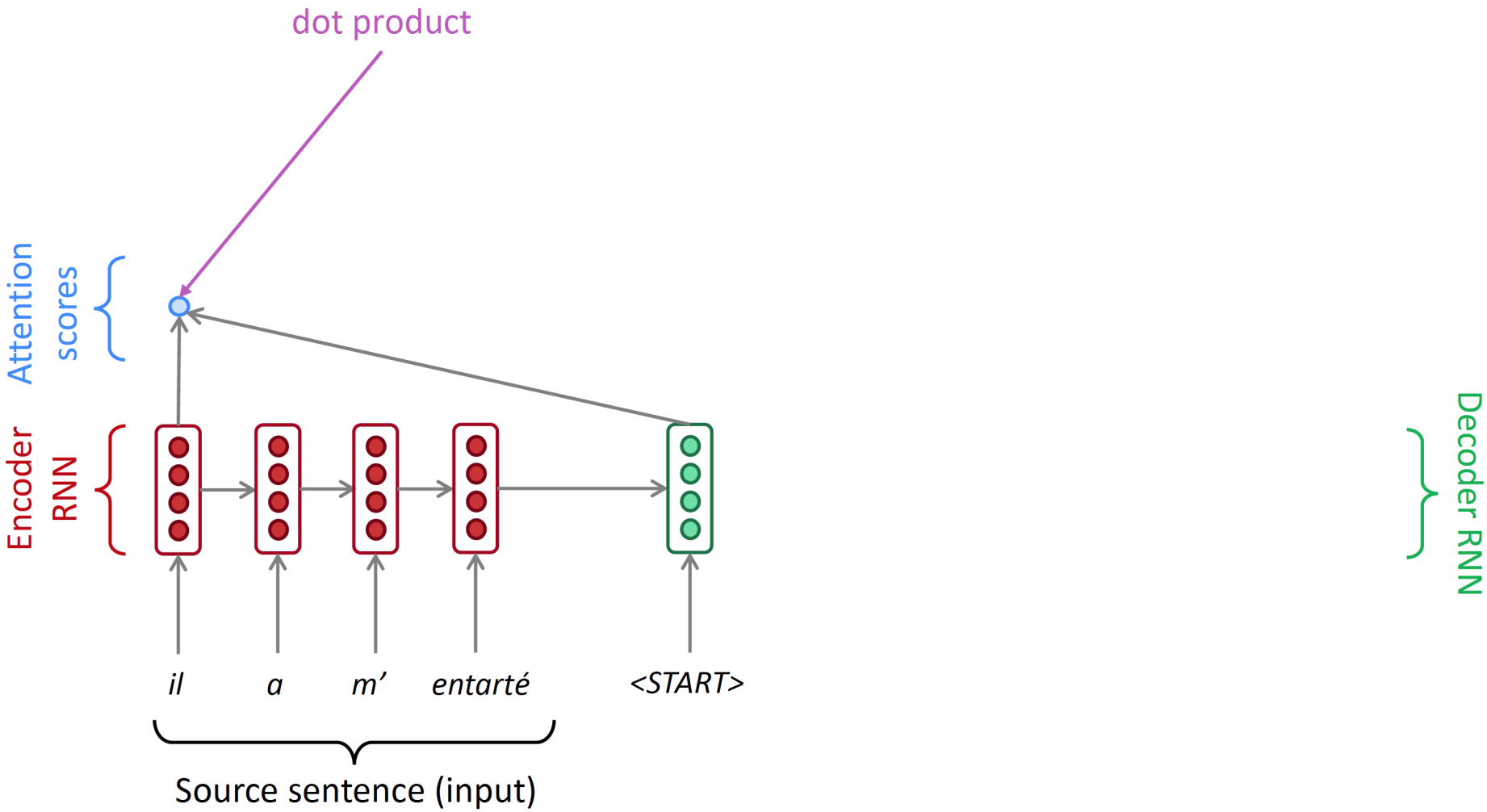
Sequence-to-sequence: the bottleneck problem



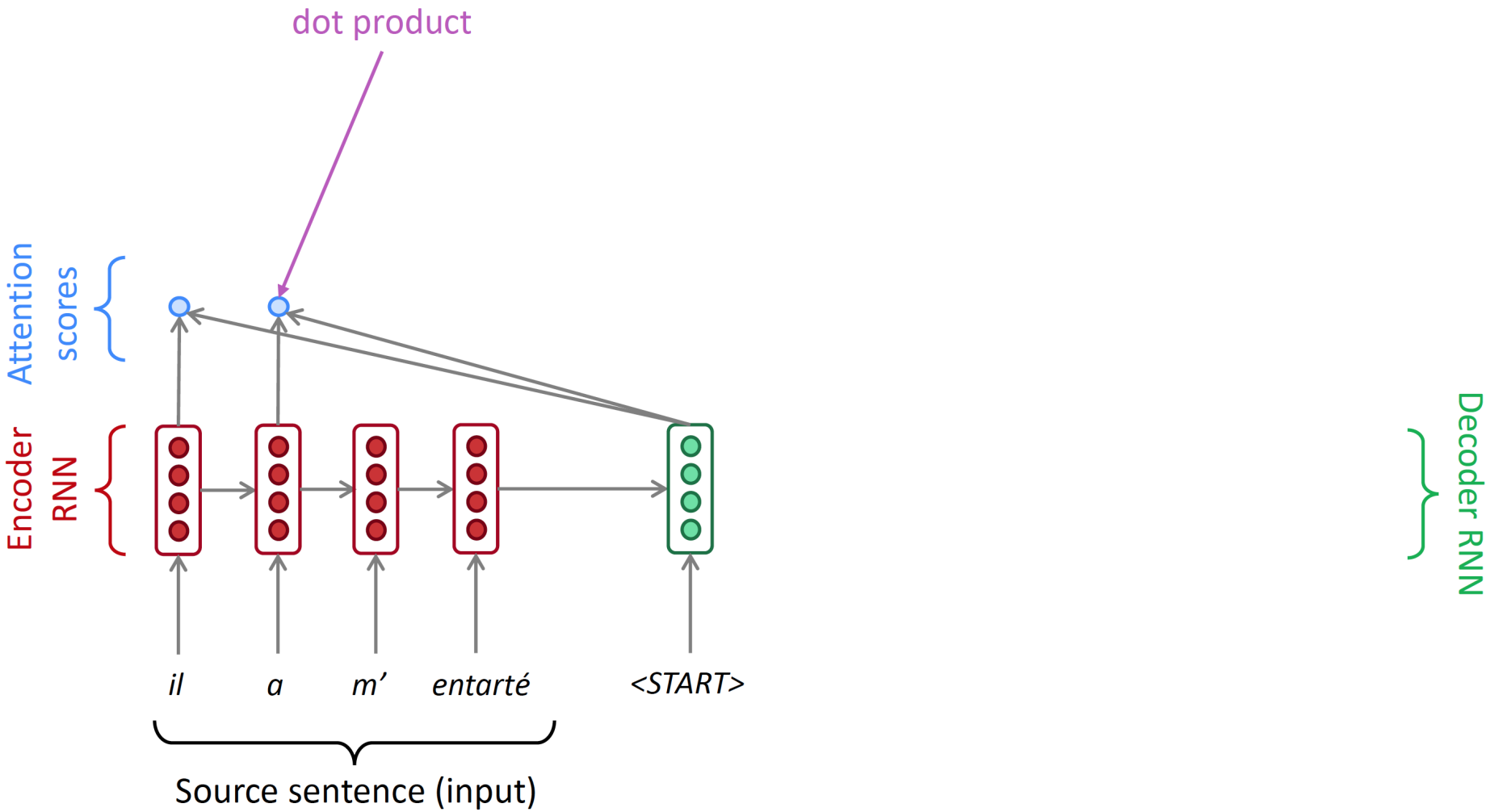
Attention

- **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.

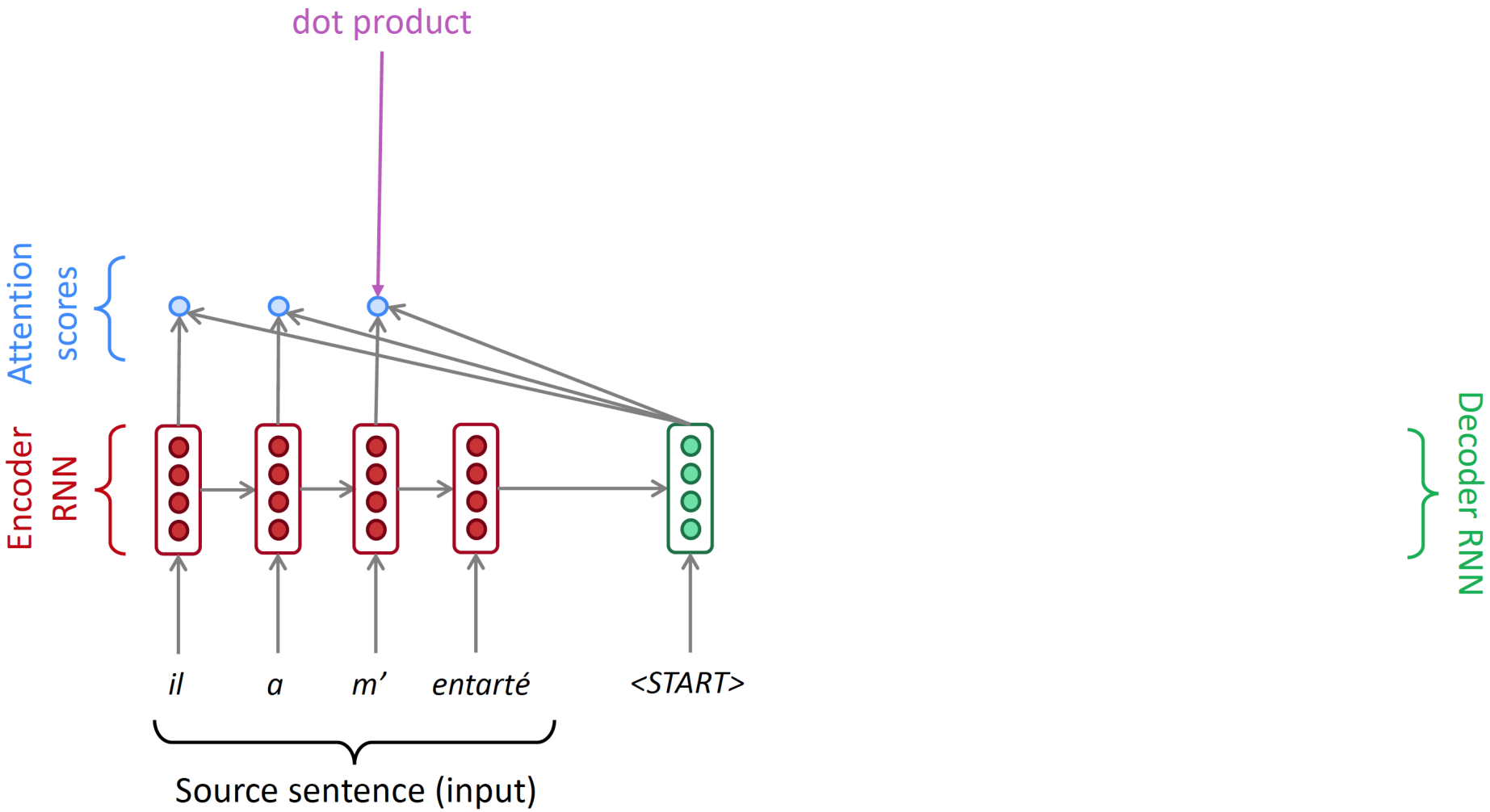
Sequence-to-sequence with attention



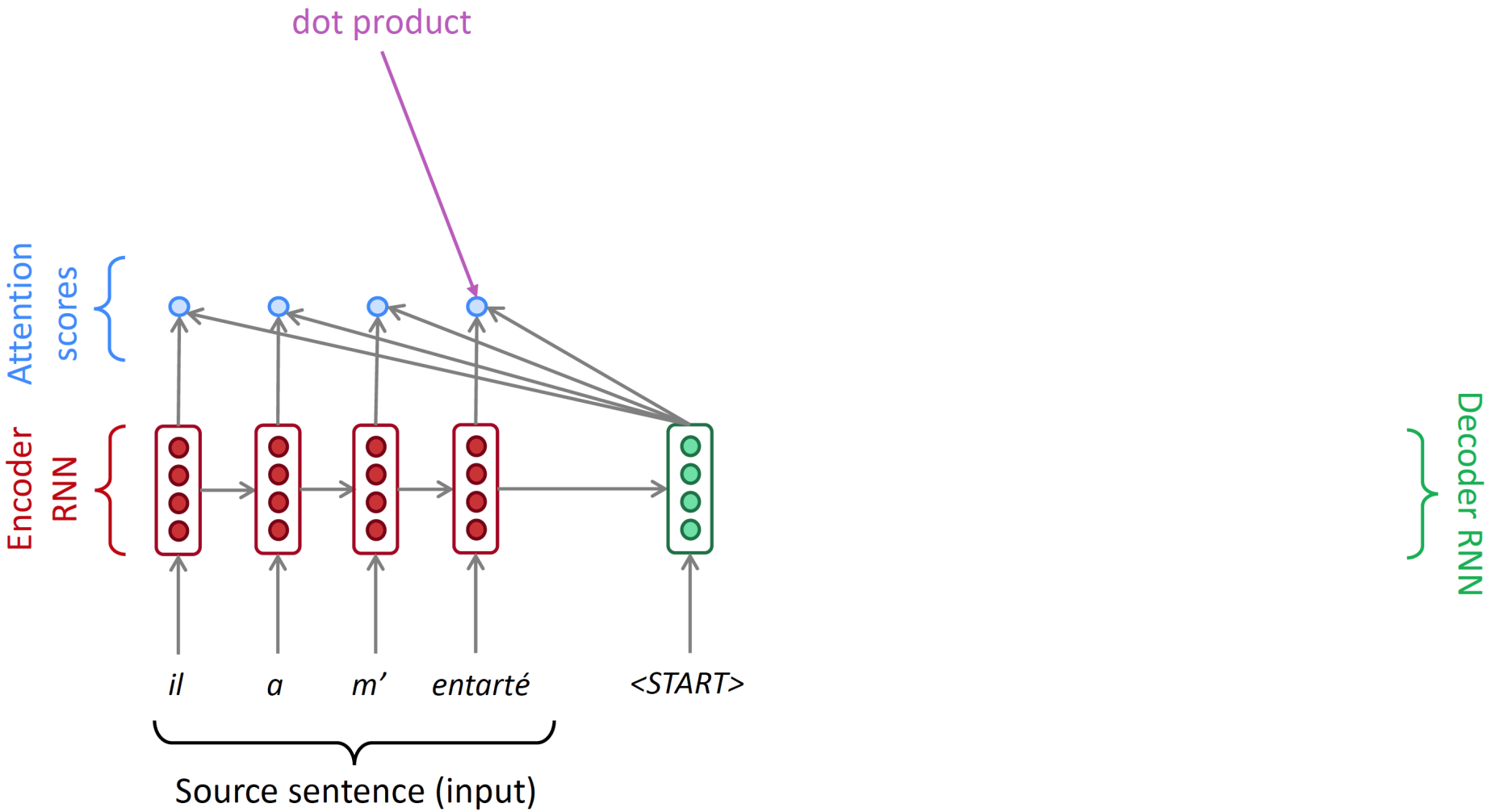
Sequence-to-sequence with attention



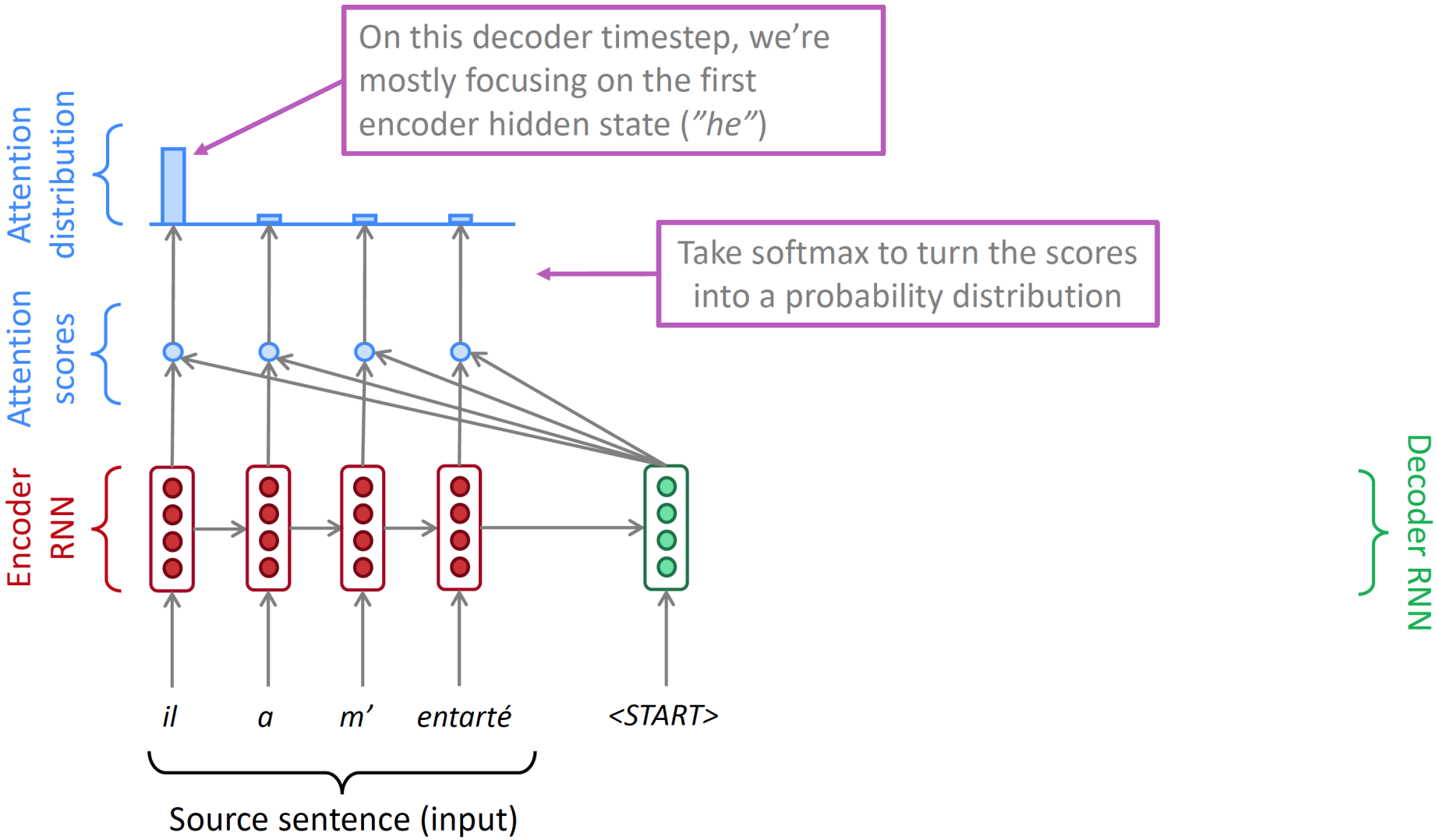
Sequence-to-sequence with attention



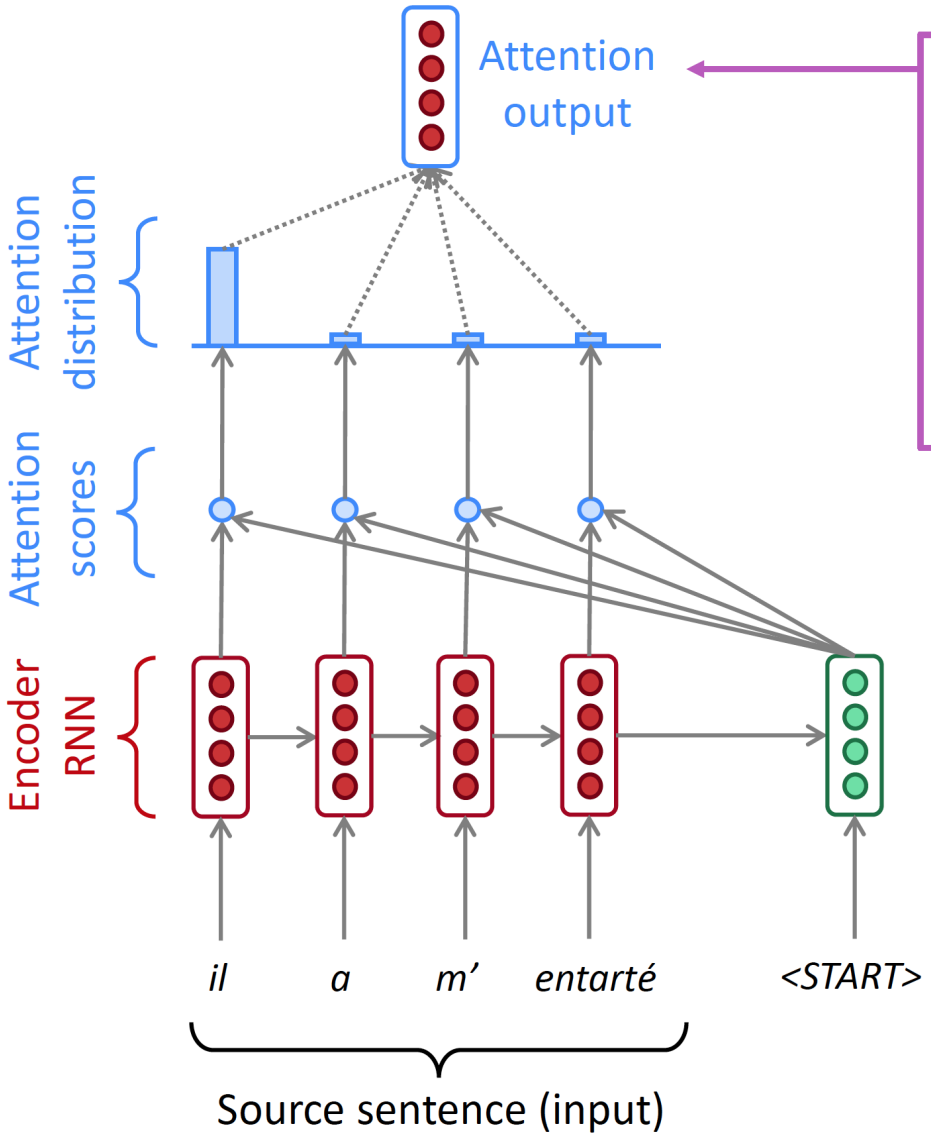
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention

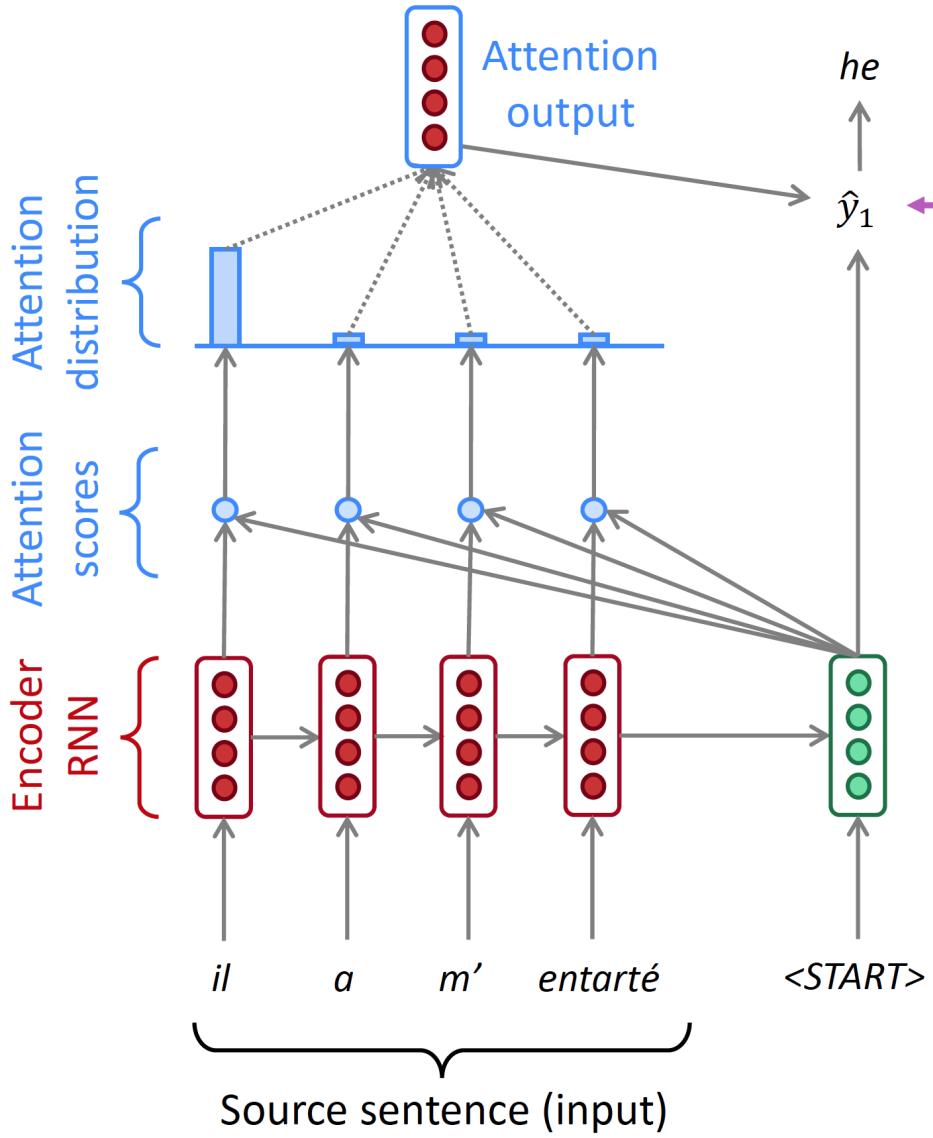


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

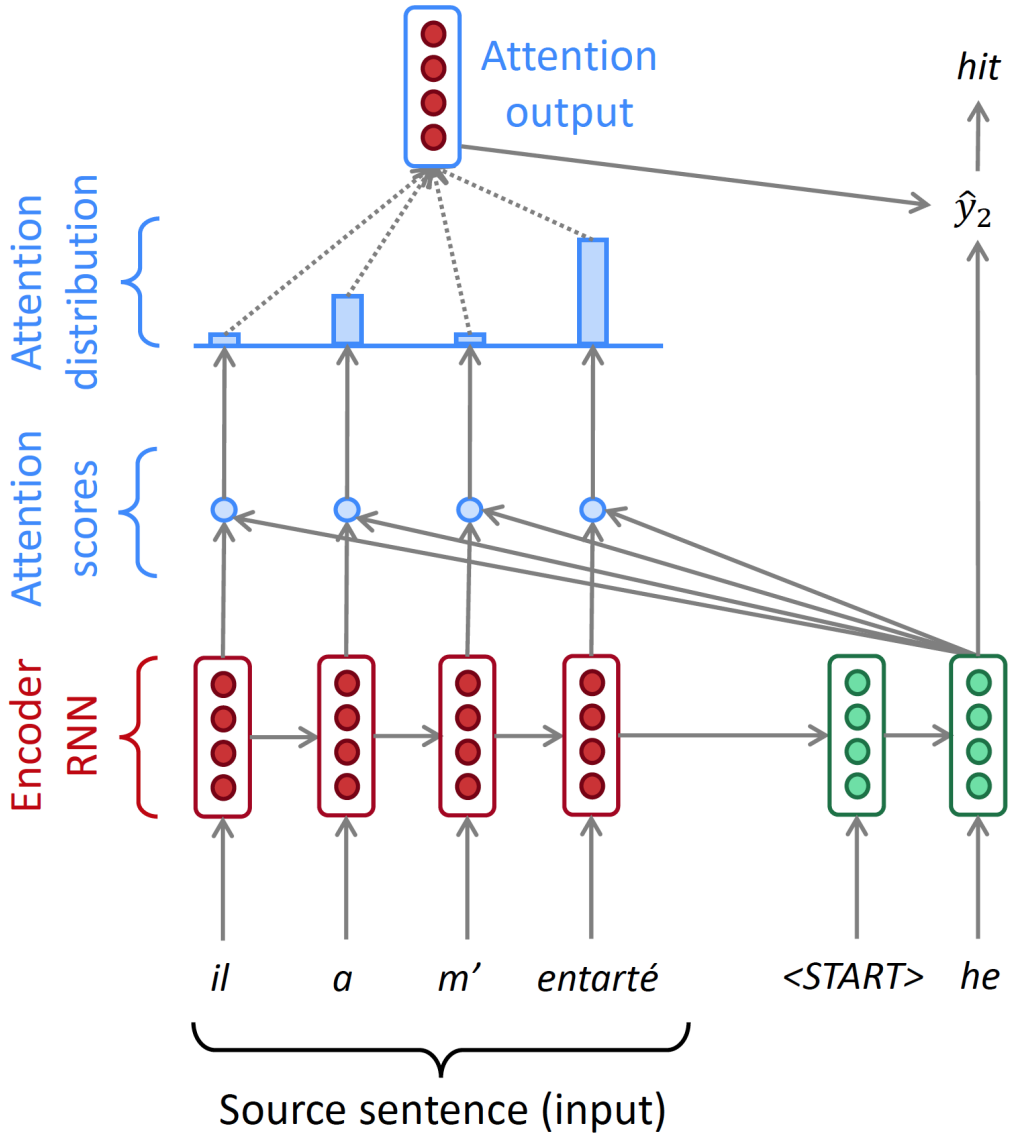
The attention output mostly contains information from the hidden states that received high attention.

Decoder RNN

Sequence-to-sequence with attention



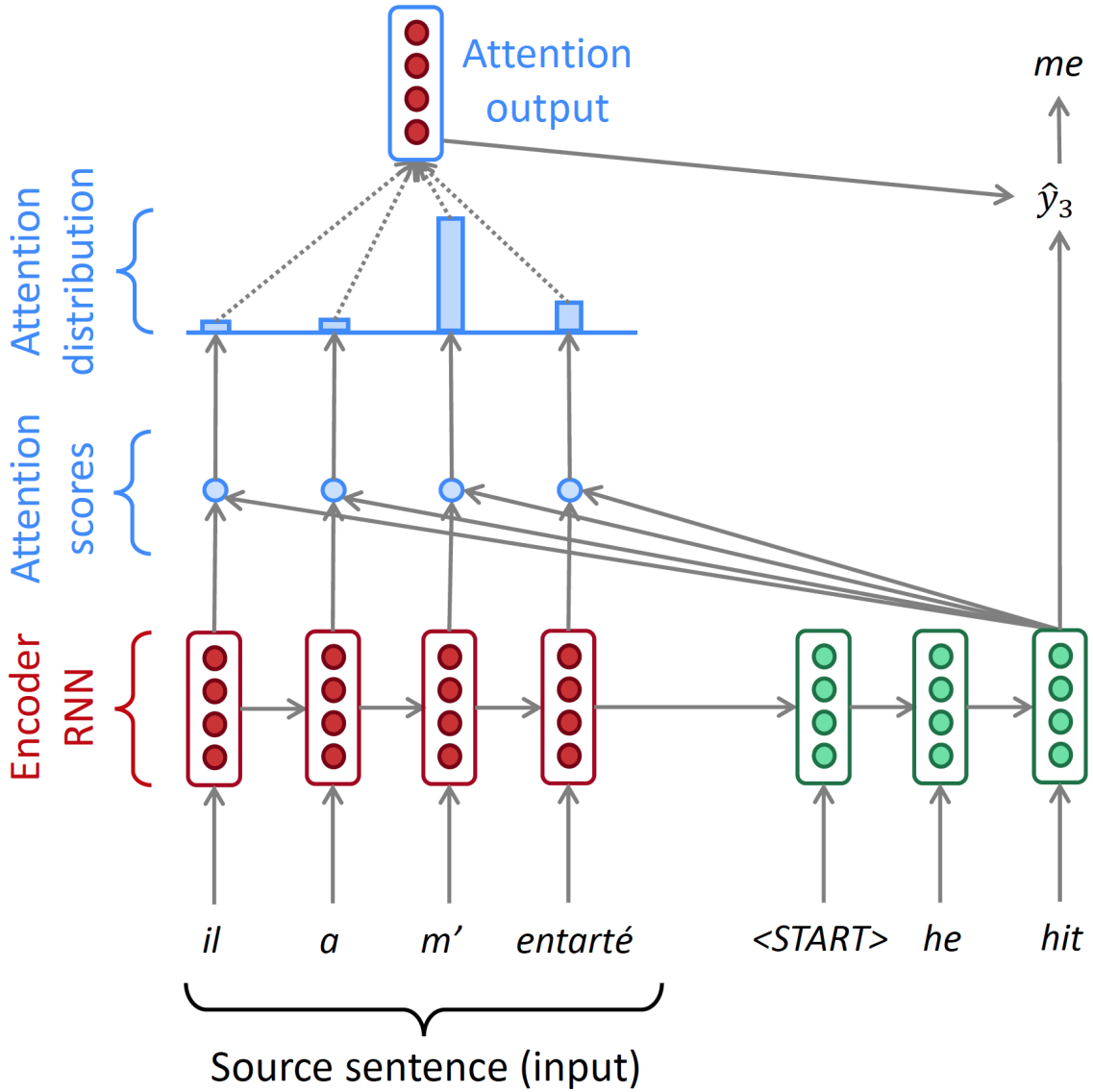
Sequence-to-sequence with attention



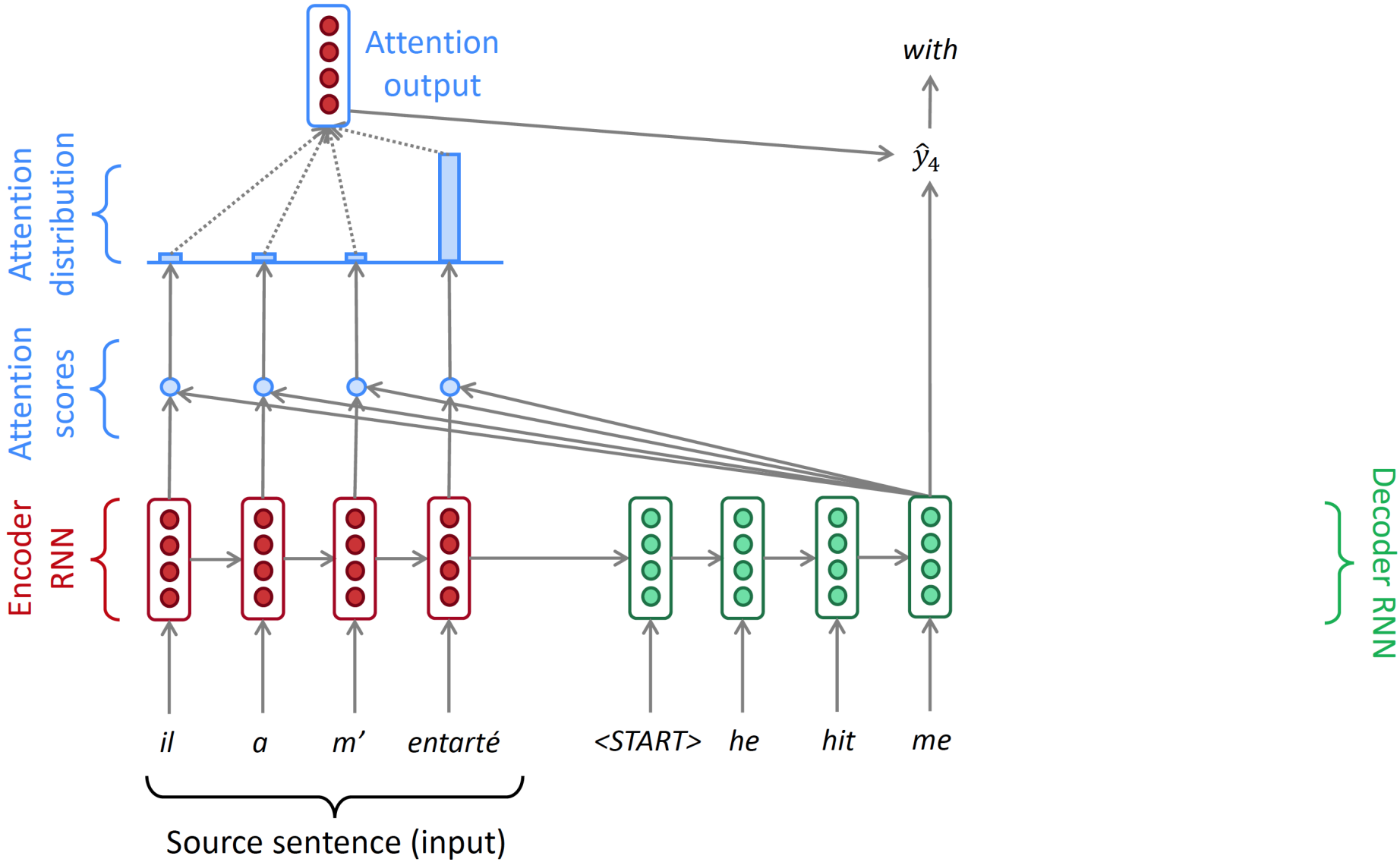
Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input).

Decoder RNN

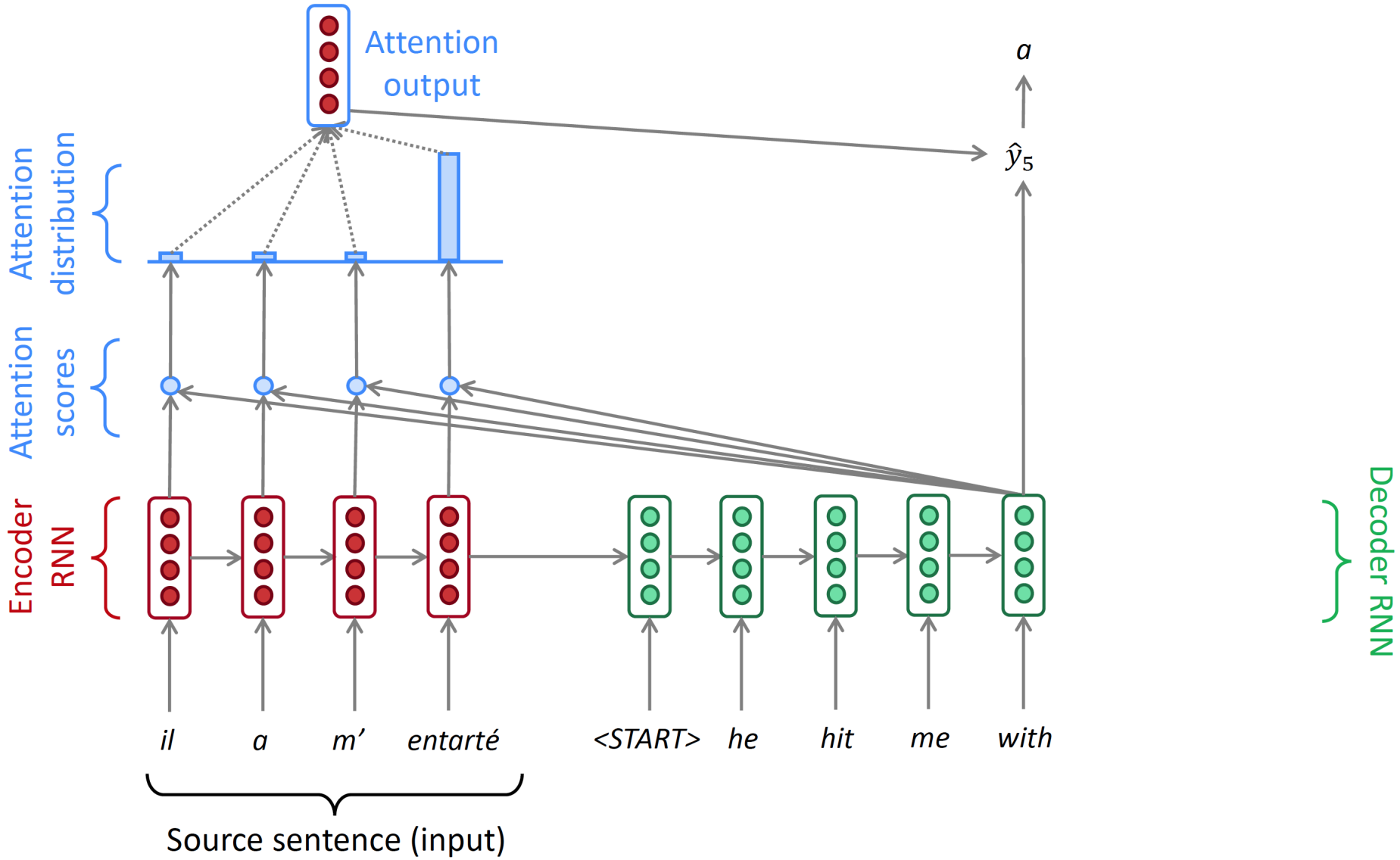
Sequence-to-sequence with attention



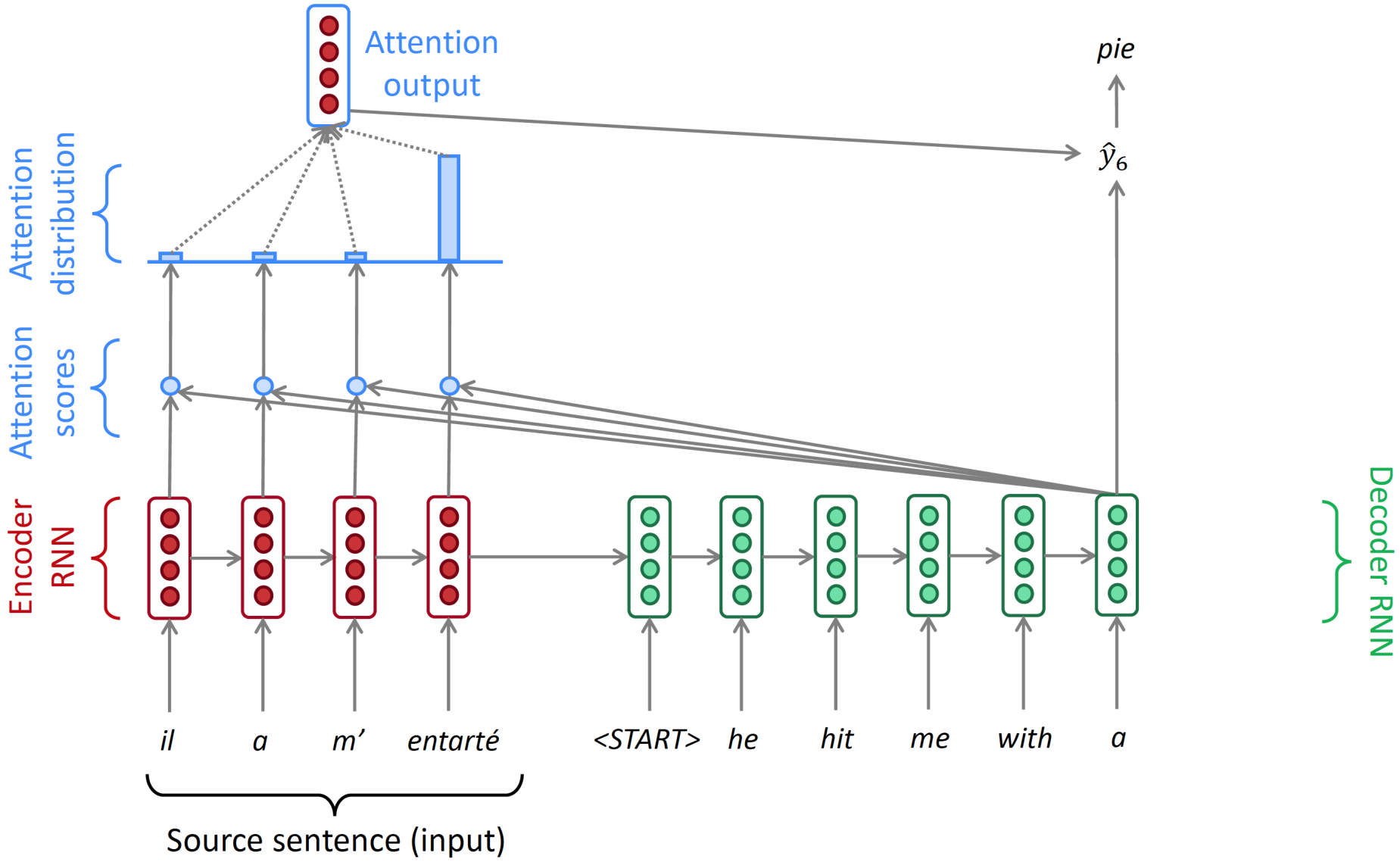
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention



Attention: in equations

- We have encoder hidden states $\boldsymbol{h}_1, ..., \boldsymbol{h}_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores $\boldsymbol{e}^t$ for this step:

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

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

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

- 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 $\boldsymbol{a}_t$ with the decoder hidden state $\boldsymbol{s}_t$ 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 is 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 great because we never explicitly trained an alignment system
  - The network just learned alignment by itself

Attention is a general Deep Learning technique

- We have seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- However: We 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 (encoder hidden states) $\boldsymbol{h}_1, ..., \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a query (decoder hidden state) $\boldsymbol{s} \in \mathbb{R}^{d_2}$
- Attention always involves:
  1. Computing the attention scores $\boldsymbol{e} \in \mathbb{R}^N$
  2. Taking softmax to get attention distribution α:

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

  3. 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 $\boldsymbol{a}$ (sometimes called the context vector).

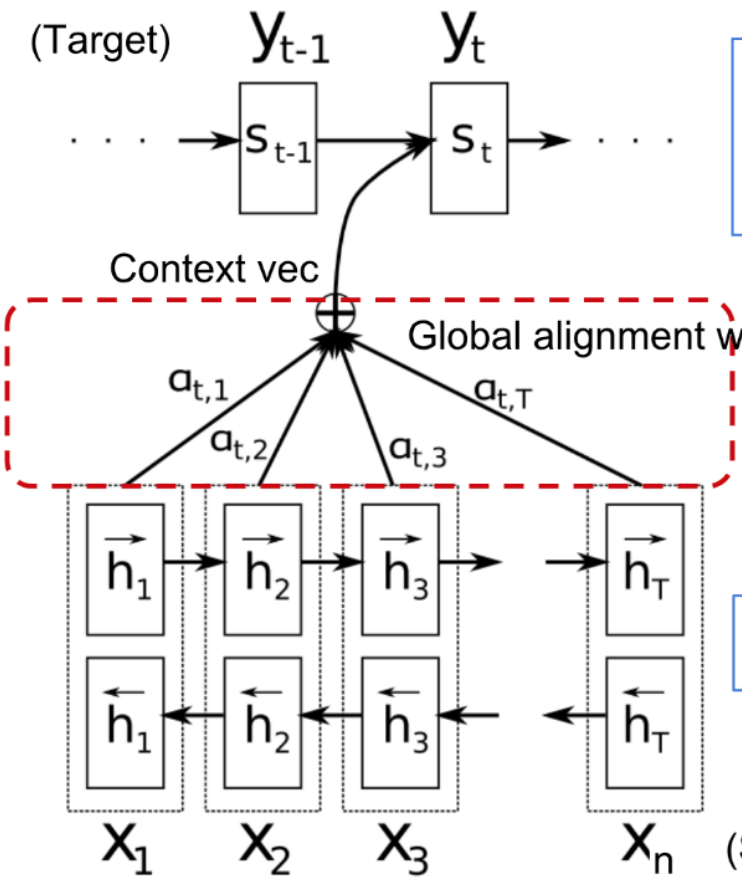
Attention variants

There are several ways we can compute $\boldsymbol{e} \in \mathbb{R}^N$ from $\boldsymbol{h}_1, ..., \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and $\boldsymbol{s} \in \mathbb{R}^{d_2}$:

- Basic dot-product attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$
  [Luong, Pham, Manning, EMNLP 2015]
  - 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, Manning, EMNLP 2015]
  - Where $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix
- 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, Bengio, ICLR 2015]
  - 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

Note that $\boldsymbol{e}_{it} = a(s_t, h_i)$ is an alignment model which scores how well the inputs around position i and the output at position t match.

Additive attention [Bahdanau et al., ICLR 2015]



Decoder: RNN with input from previous state + dynamic context vector.

Attention layer: parameterized by a simple feed-forward network

Additive Attention

Encoder: bidirectional RNN

Additive attention [Bahdanau et al., ICLR 2015]

- We have:
  - $\mathbf{x} = [x_1, x_2, \cdots, x_n]$ (source sequence)
  - $\mathbf{y} = [y_1, y_2, \cdots, y_m]$ (target sequence)
- The encoder is a bidirectional RNN
  - $\boldsymbol{h}_i = [\overrightarrow{\boldsymbol{h}_i}; \overleftarrow{\boldsymbol{h}_i}], i = 1, \cdots, n.$

$$\mathbf{c}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i \quad ; \text{Context vector for output } y_t$$

$$\alpha_{t,i} = \text{align}(y_t, x_i) \quad ; \text{How well two words } y_t \text{ and } x_i \text{ are aligned.}$$

$$= \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))} \quad ; \text{Softmax of some predefined alignment score..}$$

The alignment model assigns a score $\alpha_{t,i}$ to the pair of input at position i and output at position t, (y_t, x_i), based on how well they match. The set of $\{\alpha_{t,i}\}$ are weights defining how much of each source hidden state should be considered for each output.

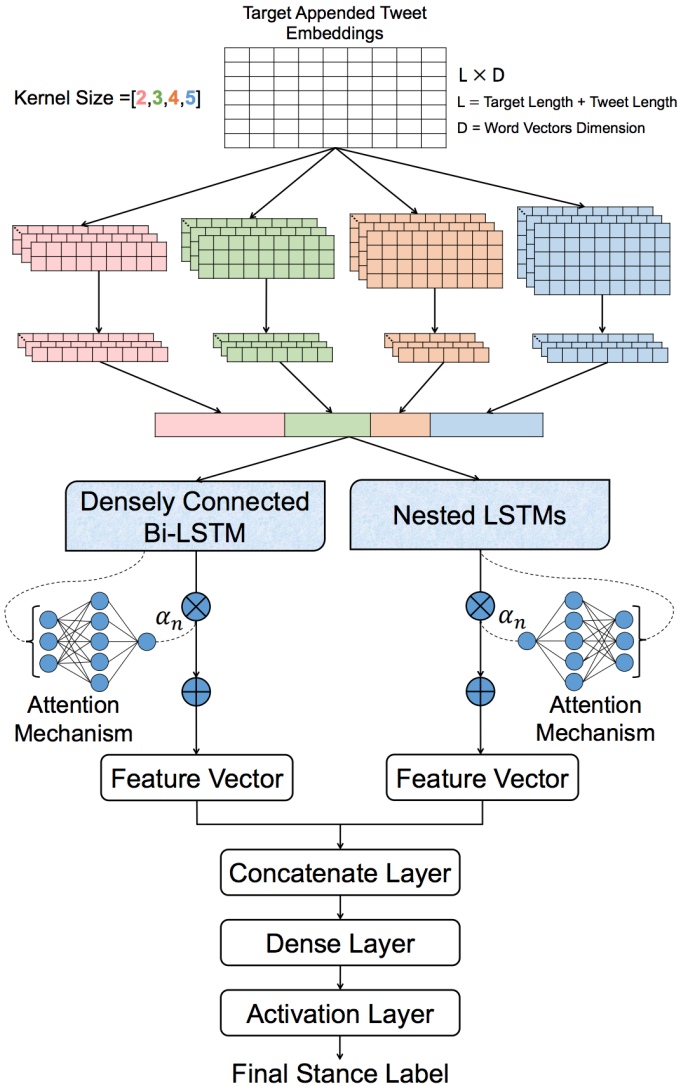
Additive attention [Bahdanau et al., ICLR 2015]

In Bahdanau's paper, the alignment score α is parametrized by a feed-forward network with a single hidden layer and this network is jointly trained with other parts of the model. The score function is therefore in the following form, given that tanh is used as the non-linear activation function:

$$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$$

where both $\mathbf{v}_a$ and $\mathbf{W}_a$ are weight matrices to be learned in the alignment model.

An attention-based model for stance detection [Siddiqua et al., NAACL 2019]



https://www.aclweb.org/anthology/N19-1185.pdf