Deep Neural Networks for Natural Language Processing (AI6127)

JUNG-JAE KIM

SEQUENCE-TO-SEQUENCE WITH ATTENTION (FEAT. MACHINE TRANSLATION)

Lecture Plan

- NLP application: Machine Translation
  - Pre-neural Machine Translation
- Neural Machine Translation (NMT) using sequence-to-sequence (seq2seq)
- Neural Machine Translation using sequence-to-sequence with attention

Machine Translation

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

- Rousseau

Source: CS224n

Pre-Neural Machine Translation

1950s: Early Machine Translation

- Machine Translation research began in the early 1950s
  - on machines less powerful than high school calculators
- Russian → English (motivated by the Cold War!)
- Systems were mostly rule-based, using a bilingual dictionary to map Russian words to their English counterparts
  - Human language is more complicated than that, and varies more across languages!
  - Problem soon appeared intractable

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

Source: CS224n

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

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

- Use Bayes Rule to break this down into two components to be learnt separately: $= \mathrm{argmax}_y \underbrace{P(x|y)}\underbrace{P(y)}$

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

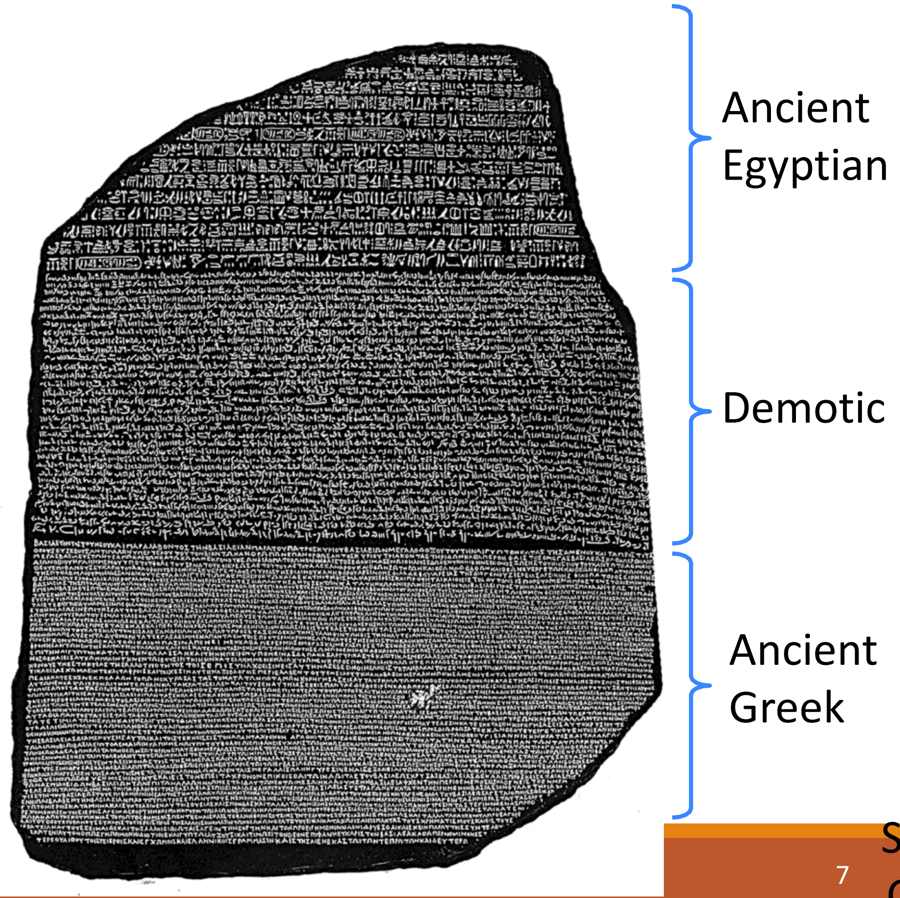
Language Model
Models how to write good English (*fluency*).
Learnt from monolingual data.

Source: CS224n

1990s-2010s: Statistical Machine Translation

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

The Rosetta Stone



Source: CS224n

Learning alignment for SMT

- Question: How to learn translation model $P(x|y)$ from parallel corpus?
- Second, break it down further

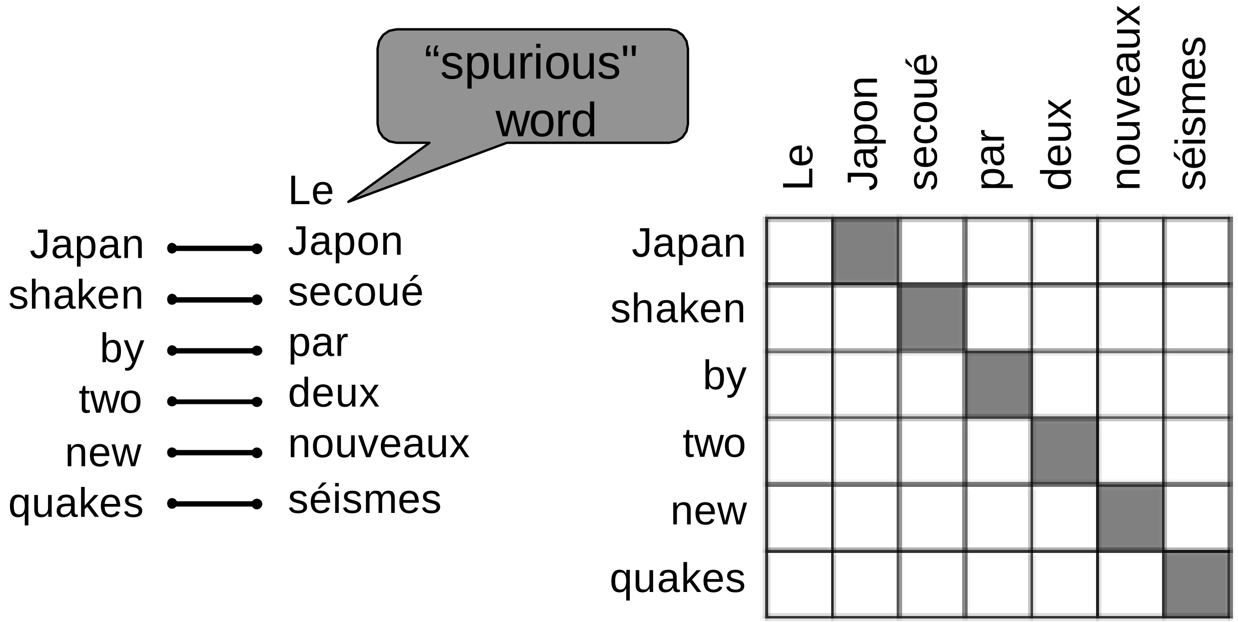
$$P(x, a|y)$$

  - where *a* is the alignment (or word alignment), i.e. word-level correspondence between French sentence *x* and English sentence *y*
    - Cf. sentence-aligned parallel corpus

Source: CS224n

What is alignment?

- Alignment is the correspondence between particular words in the translated sentence pair.
  - Note: Some words have no counterpart

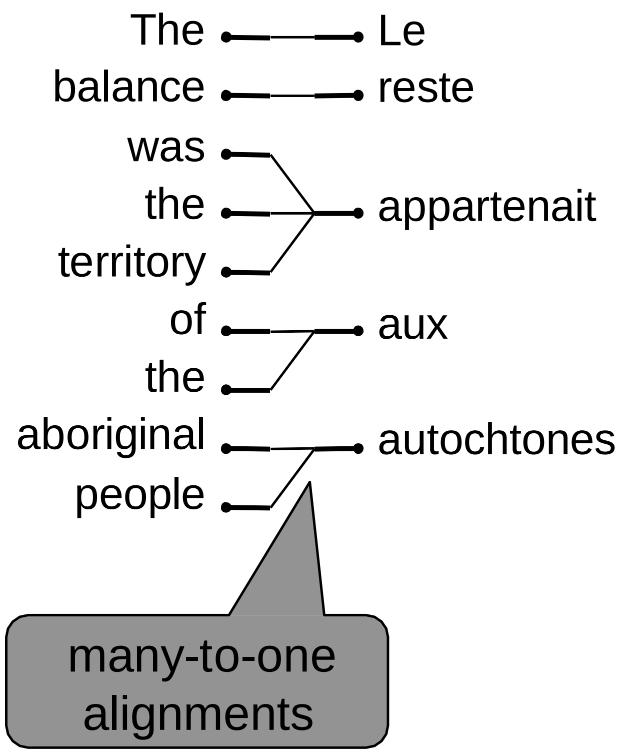


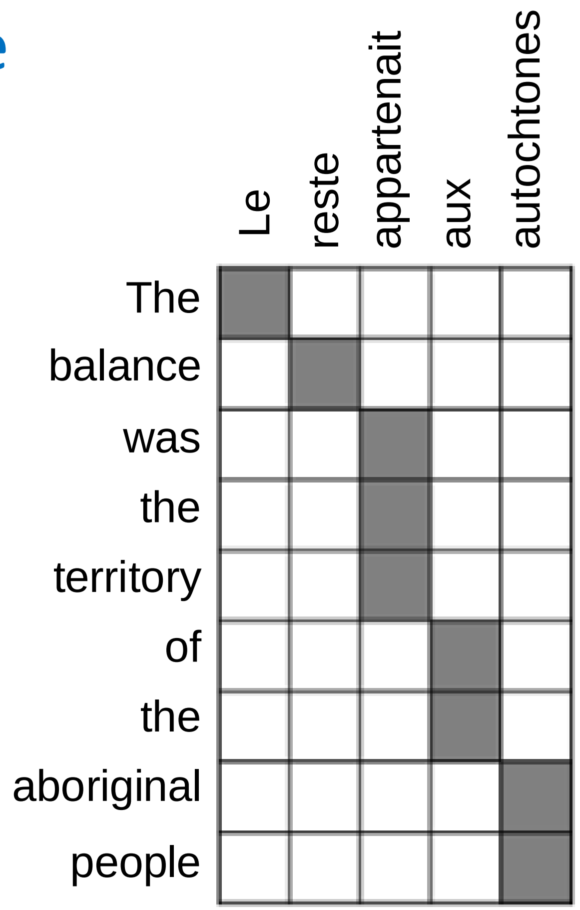
Examples from: "The Mathematics of Statistical Machine Translation: Parameter Estimation", Brown et al, 1993. http://www.aclweb.org/anthology/J93-2003

Source: CS224n

Alignment is complex

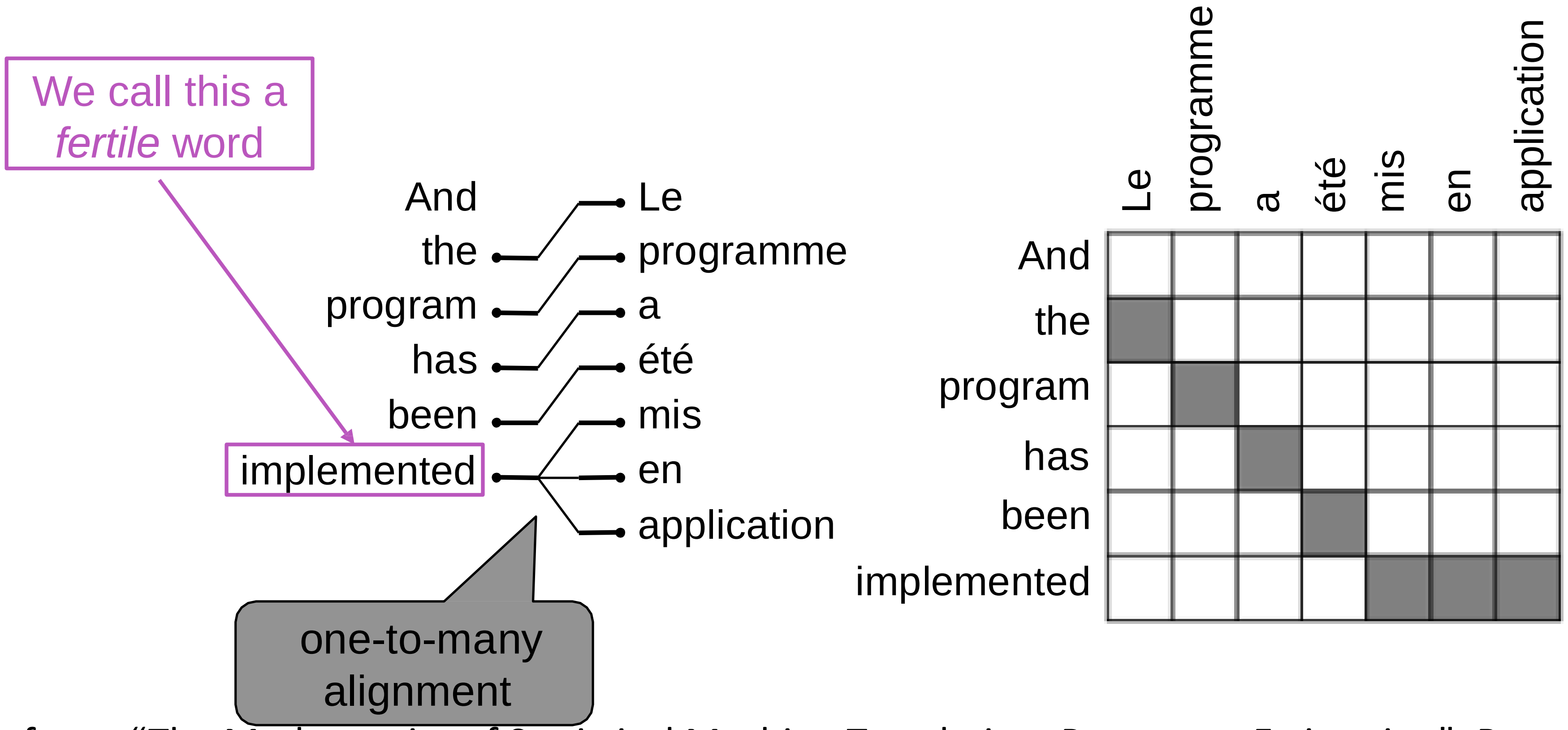
- Alignment can be many-to-one





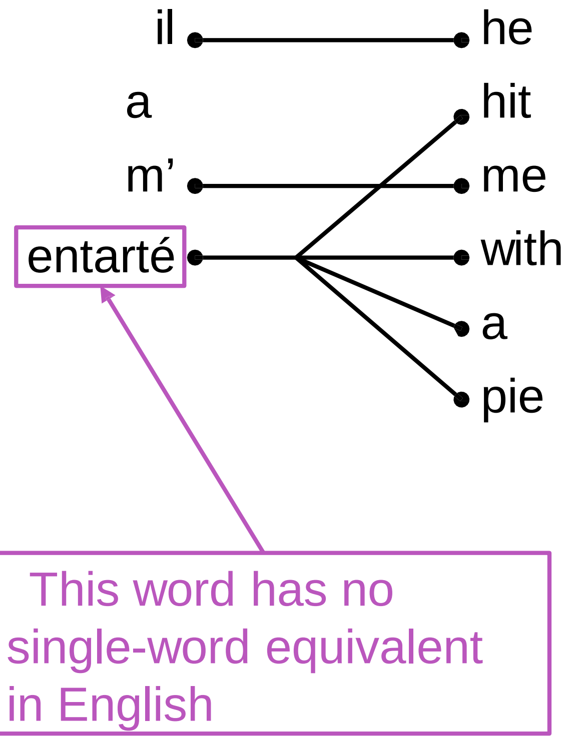
Alignment is complex

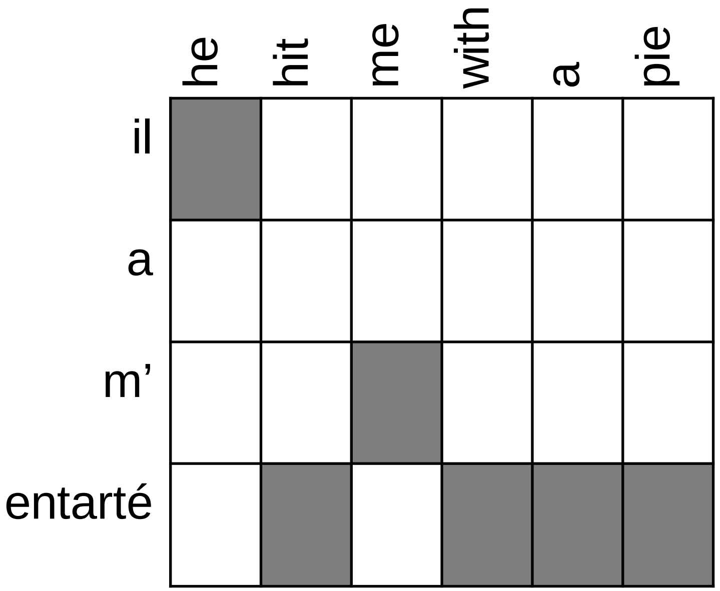
- Alignment can be one-to-many



Alignment is complex

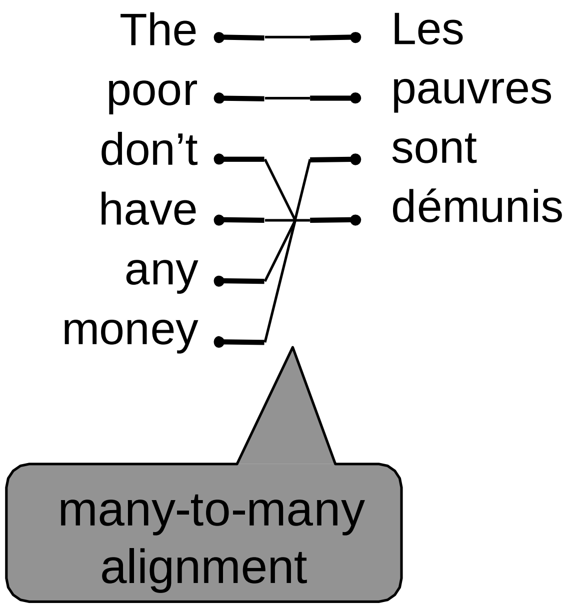
- Some words are very fertile!

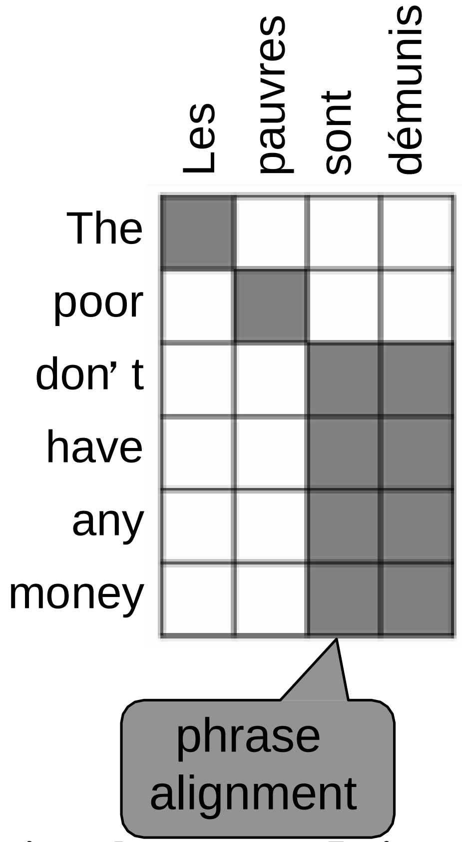




Alignment is complex

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





Examples from: “The Mathematics of Statistical Machine Translation: Parameter Estimation", Brown et al, 1993. http://www.aclweb.org/anthology/J93-2003

Source: CS224n

Learning alignment for SMT

- We learn as a combination of many factors, including:
  - Probability of particular words aligning (also depends on position in sentence)
  - 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
    - See for more details https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1162/handouts/Collins_annotated.pdf

Source: CS224n

Decoding for SMT

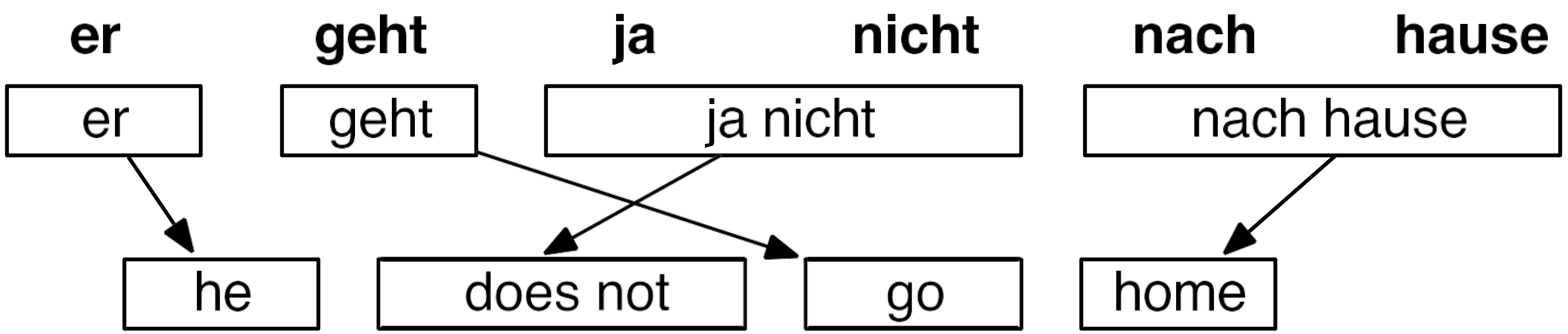
$$\operatorname{argmax}_y \underbrace{P(x|y)}_{\text{Translation Model}} \underbrace{P(y)}_{\text{Language Model}}$$

Question:
How to compute this argmax?

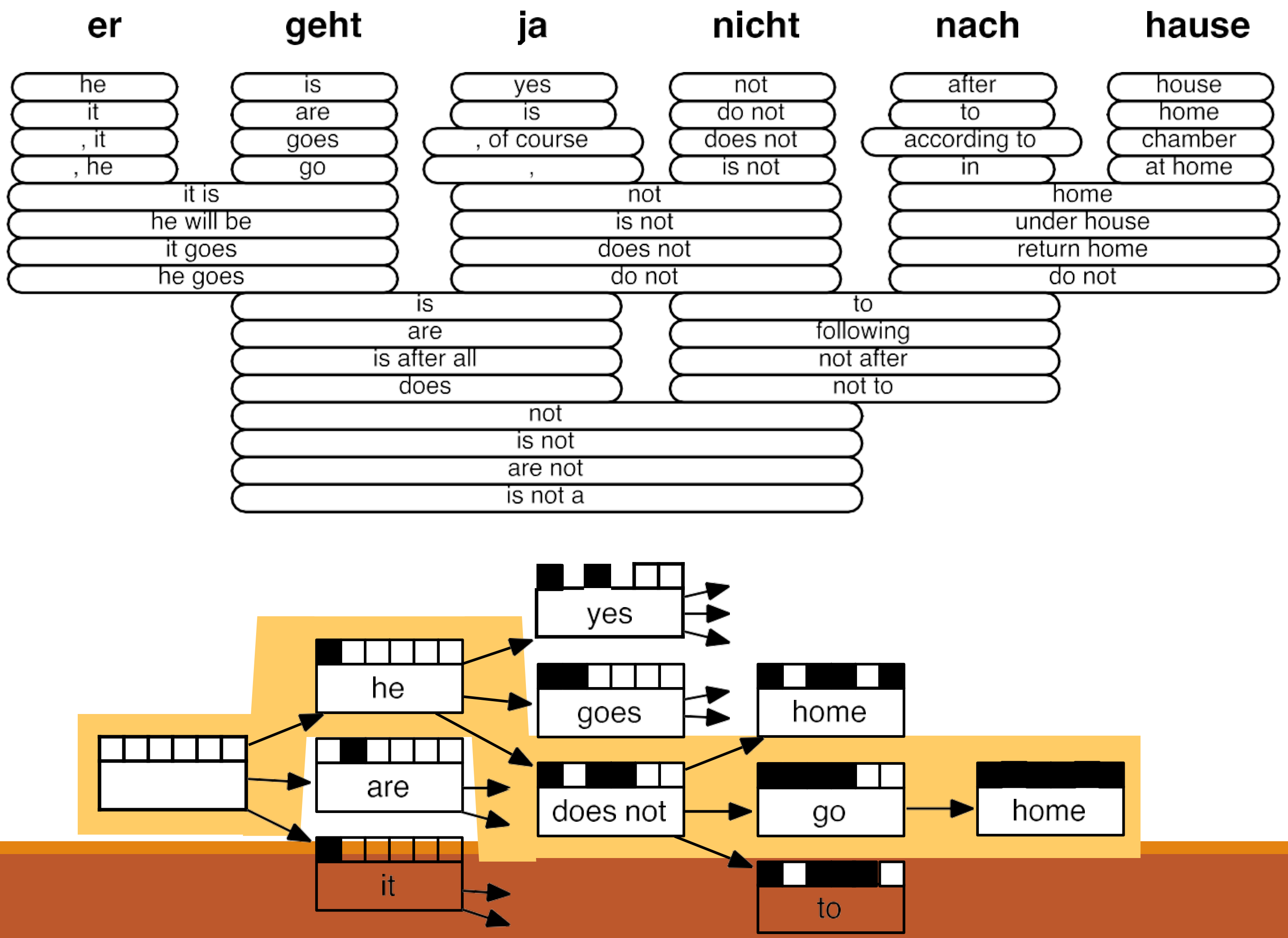
- We could enumerate every possible *y* and calculate the probability?
  → Too expensive!
- **Answer:** Use a heuristic search algorithm to search for the best translation, discarding hypotheses that are too low-probability
- This process is called decoding

Source: CS224n

Decoding for SMT: Example



Decoding for SMT: Example



- Translation model
  - Phrase translation probability
  - Reordering costs
  - …
- Language model
  - p("he does not") = p('he'|START) * p('does'|'he',START) * p('not'|'does','he',START)

Source: "Statistical Machine Translation", Chapter 6, Koehn, 2009. https://www.cambridge.org/core/books/statistical-machine-translation/94EADF9F680558E13BE759997553CDE5

Source: CS224n

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!

Source: CS224n

Neural Machine Translation using Seq2Seq

What is Neural Machine Translation?

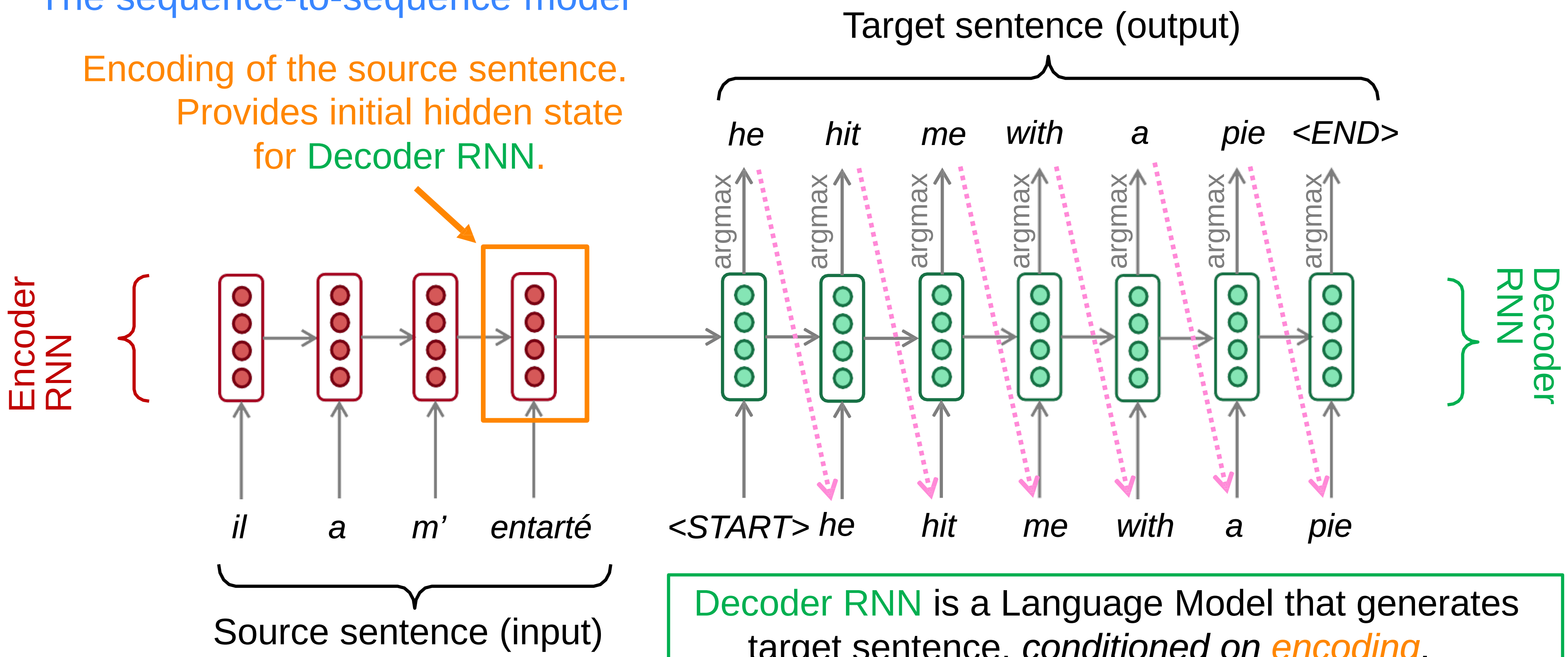
- Neural Machine Translation (NMT) is a way to do Machine Translation with a *single neural network*

- The neural network architecture is called sequence-to-sequence (aka seq2seq) and it involves *two* RNNs.

Source: CS224n

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.

Note: This diagram shows **test time** behavior: decoder output is fed in ······> as next step's input

Source: CS224n

Sequence-to-sequence is versatile!

- Sequence-to-sequence is useful for *more than just MT*

- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

Source: CS224n

Neural Machine Translation (NMT)

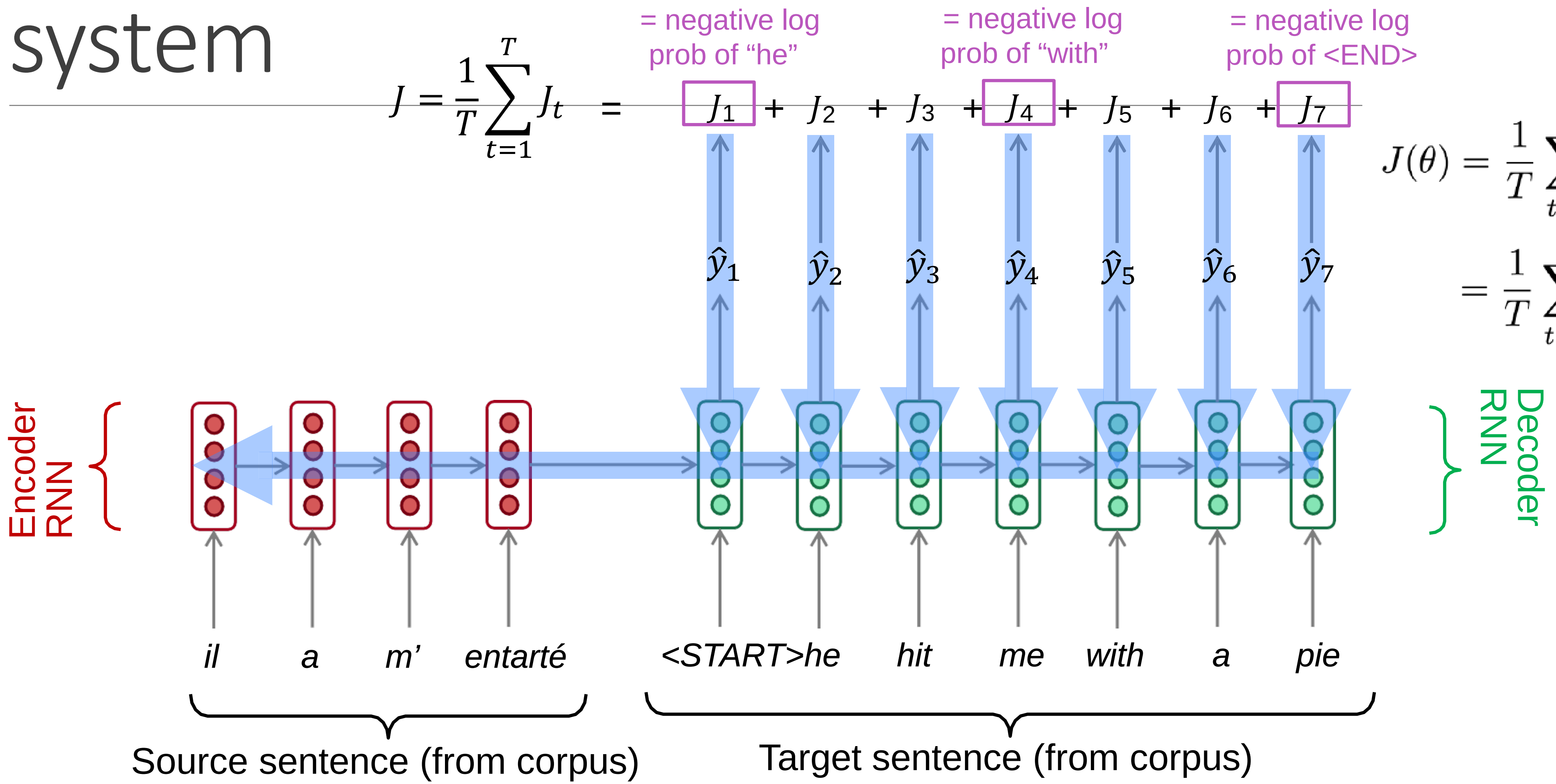
- The sequence-to-sequence model is an example of a Conditional Language Model.
  - **Language Model** because the decoder is predicting the next word of the target sentence y
  - **Conditional** because its predictions are also conditioned on the source sentence x
- NMT directly calculates : $P(y|x)$

$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots \underbrace{P(y_T|y_1, \ldots, y_{T-1}, x)}_{\text{Probability of next target word, given target words so far and source sentence x}}$$

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

Source: CS224n

Training a Neural Machine Translation system



$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$$

$$= \frac{1}{T}\sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}$$

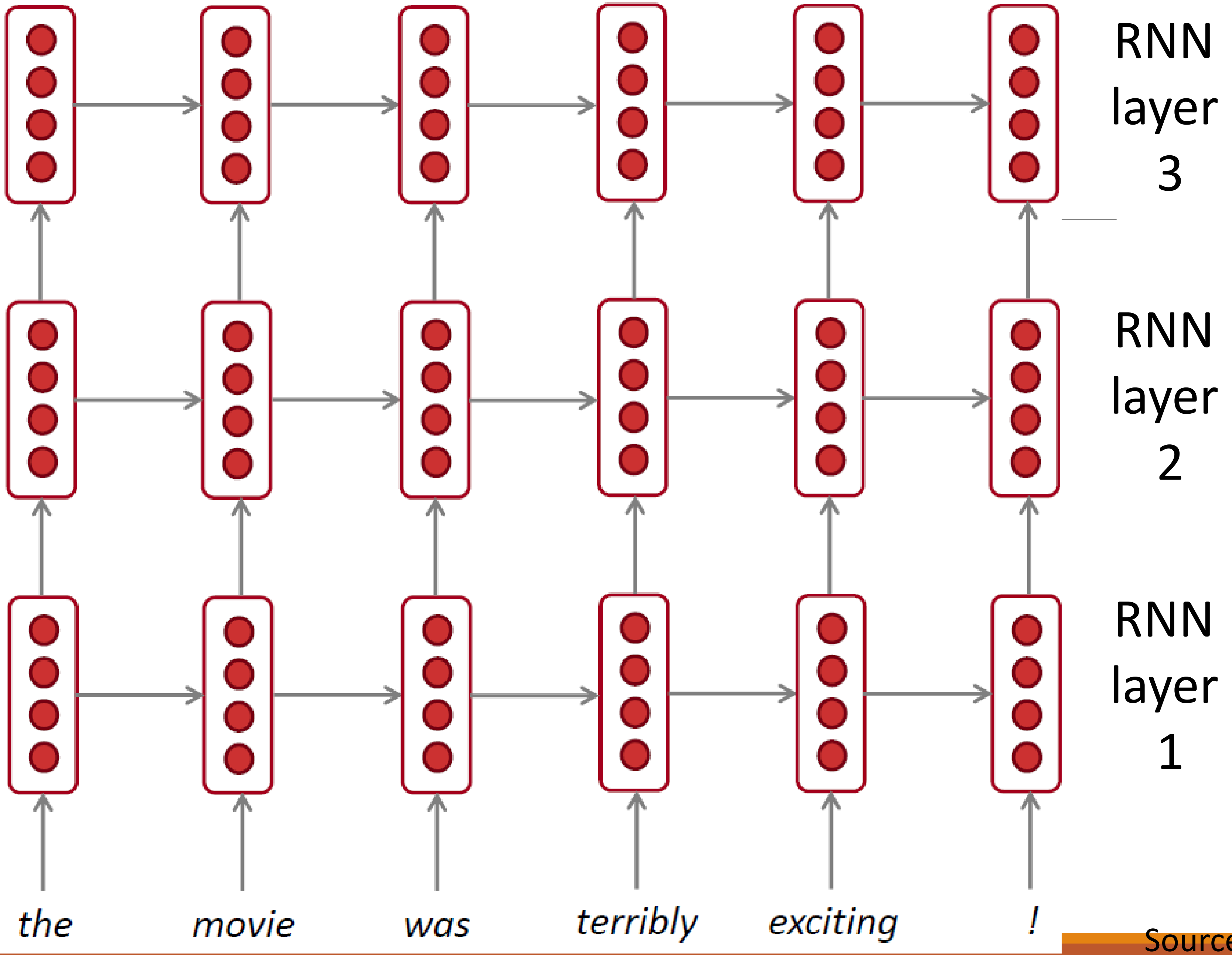
Seq2seq is optimized as a **single system.**
Backpropagation operates “*end-to-end*”.

Source: CS224n

Recall

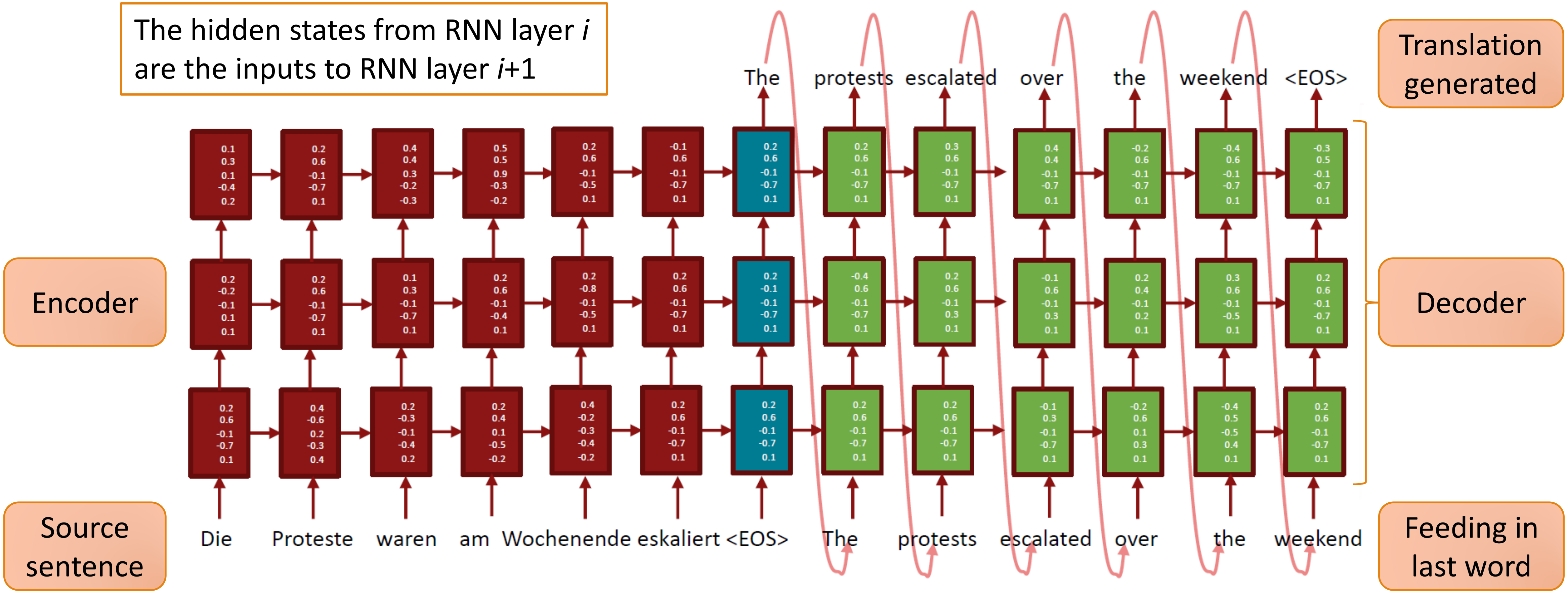
Multi-layer RNNs

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



Source: CS224n

Multi-layer deep encoder-decoder machine translation net



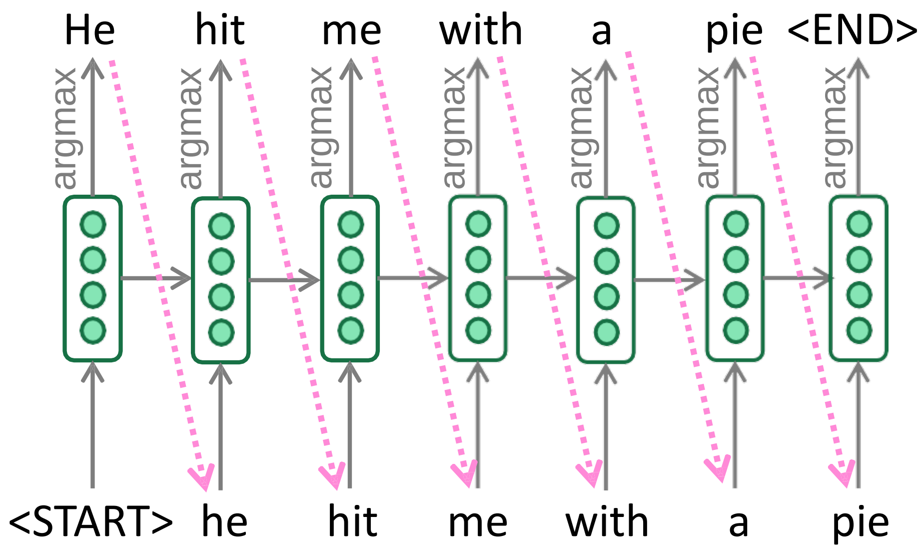
Hands-on: MT with seq2seq

- Load data files
- Seq2Seq model
- Training
- Evaluation

https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

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?

Source: CS224n

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?

Source: CS224n

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!

Source: CS224n

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 $y_1, \ldots, y_t$ 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!

Source: CS224n

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>

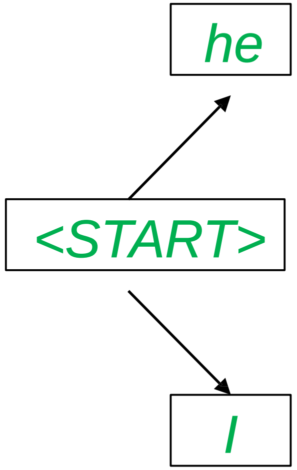
Calculate prob dist of next word

Source: CS224n

Beam search decoding: example

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

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



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

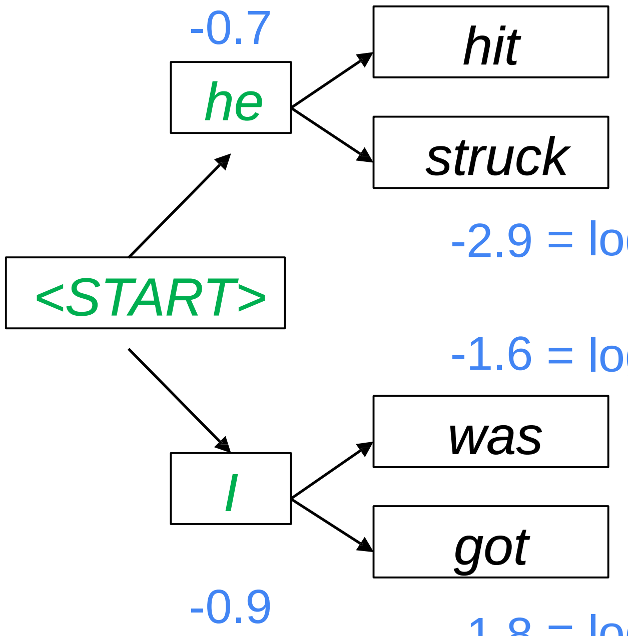
Take top *k* words
and compute scores

Source: CS224n

Beam search decoding: example

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

-1.7 = log P_{LM}(*hit*|<*START*> *he*) + -0.7



-2.9 = log P_{LM}(*struck*|<*START*> *he*) + -0.7

-1.6 = log P_{LM}(*was*|<*START*> *I*) + -0.9

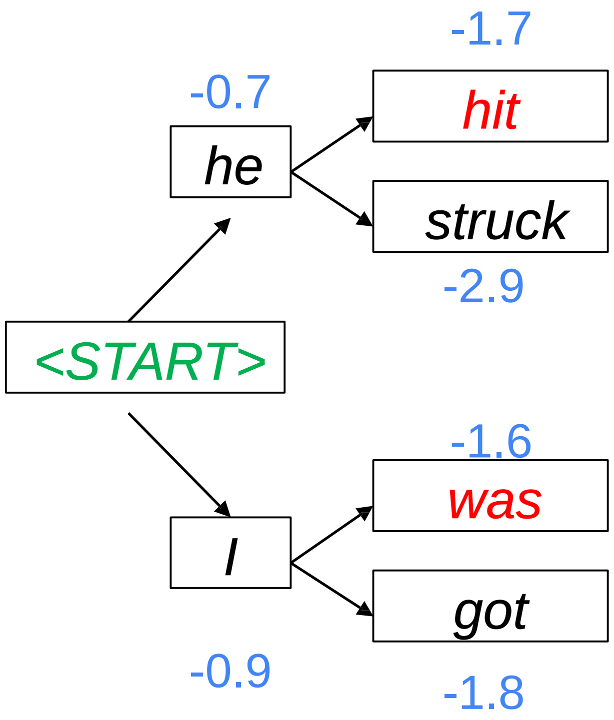
-1.8 = log P_{LM}(*got*|<*START*> *I*) + -0.9

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

Source: CS224n

Beam search decoding: example

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

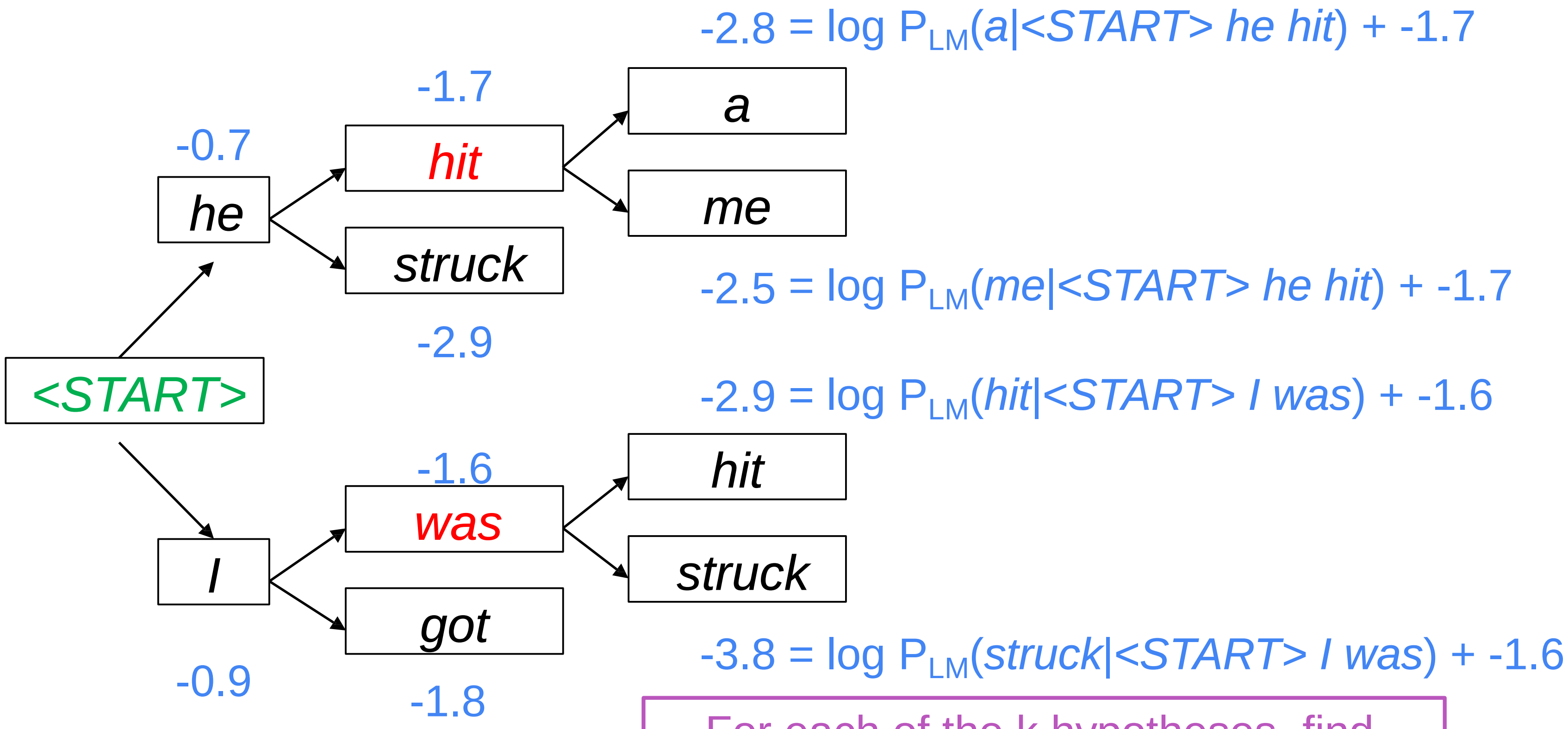


Of these k^2 hypotheses,
just keep *k* with highest scores

Source: CS224n

Beam search decoding: example

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

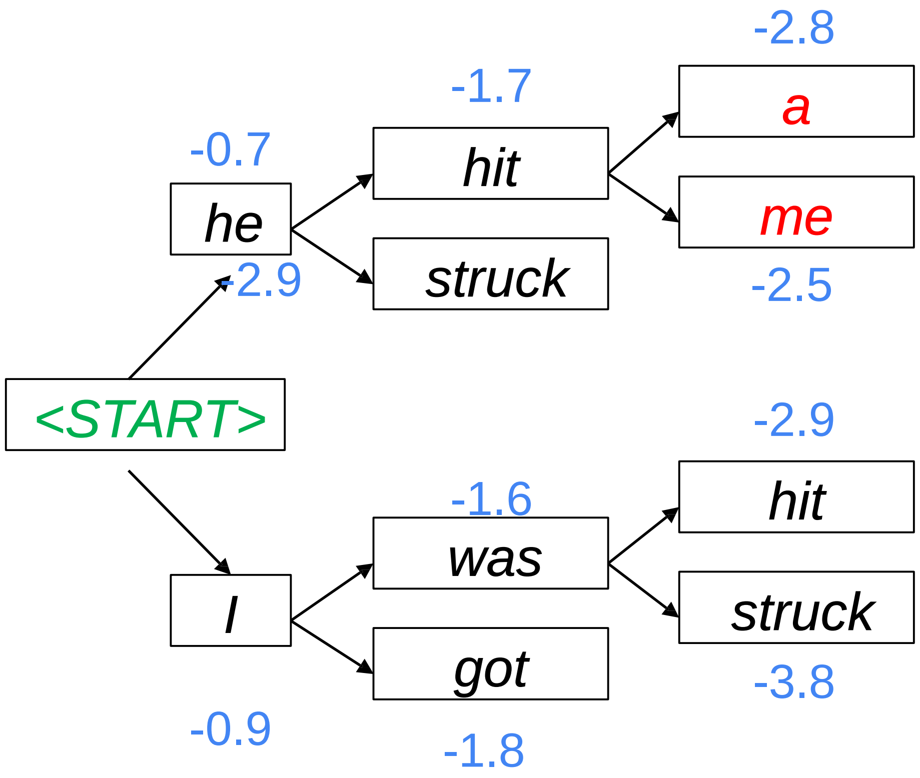


For each of the k hypotheses, find top k next words and calculate scores

Source: CS224n

Beam search decoding: example

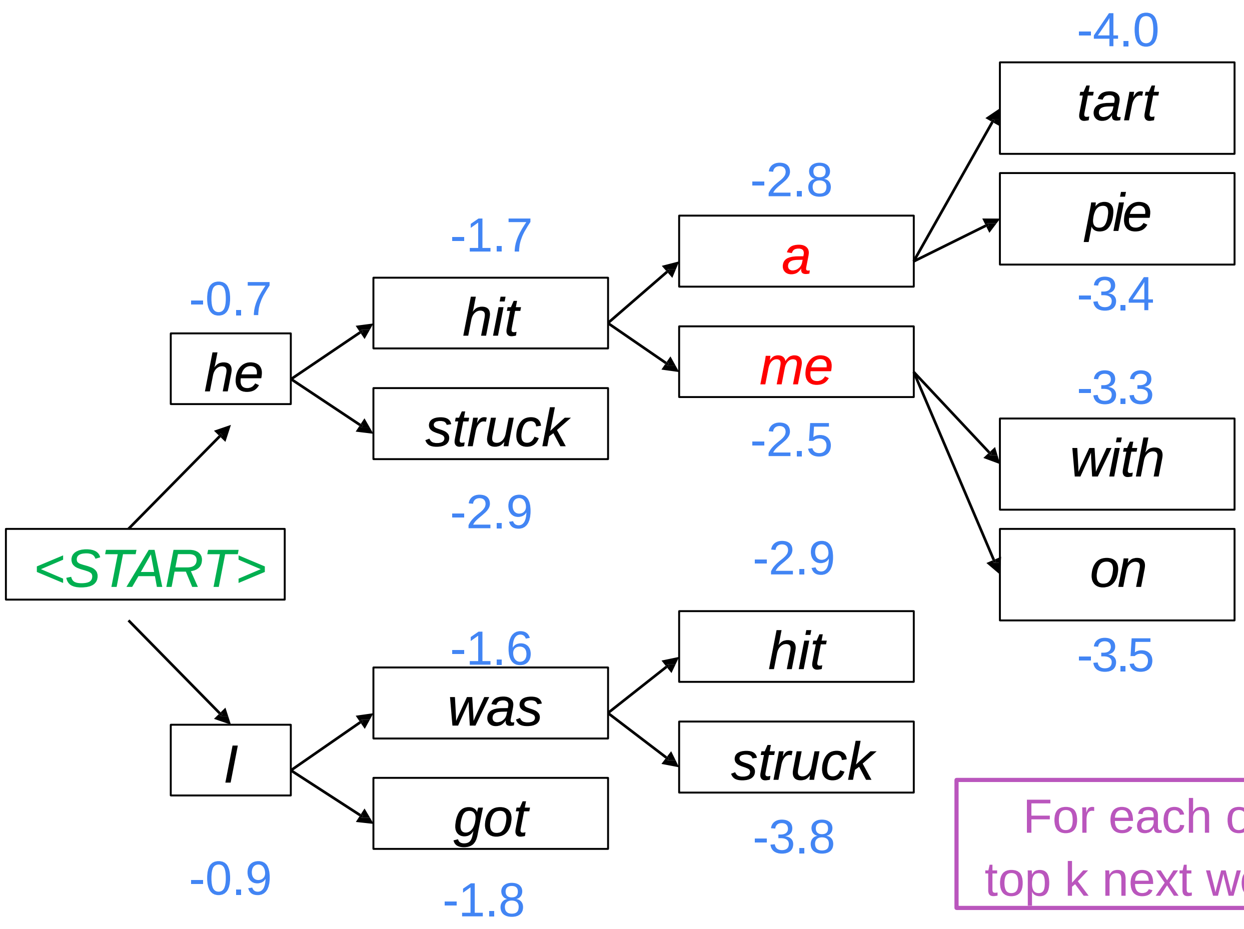
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Source: CS224n

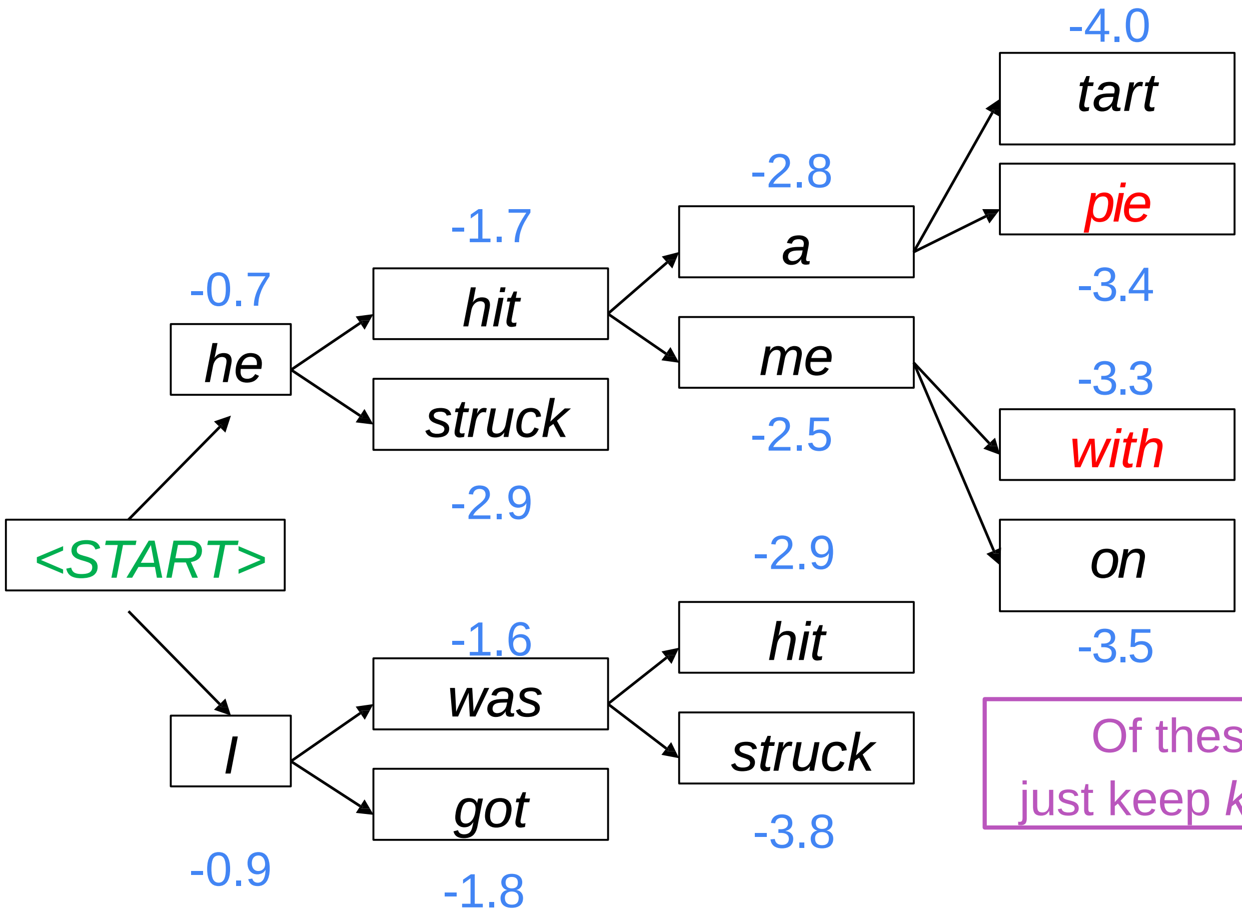
Beam search decoding: example



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

Source: CS224n

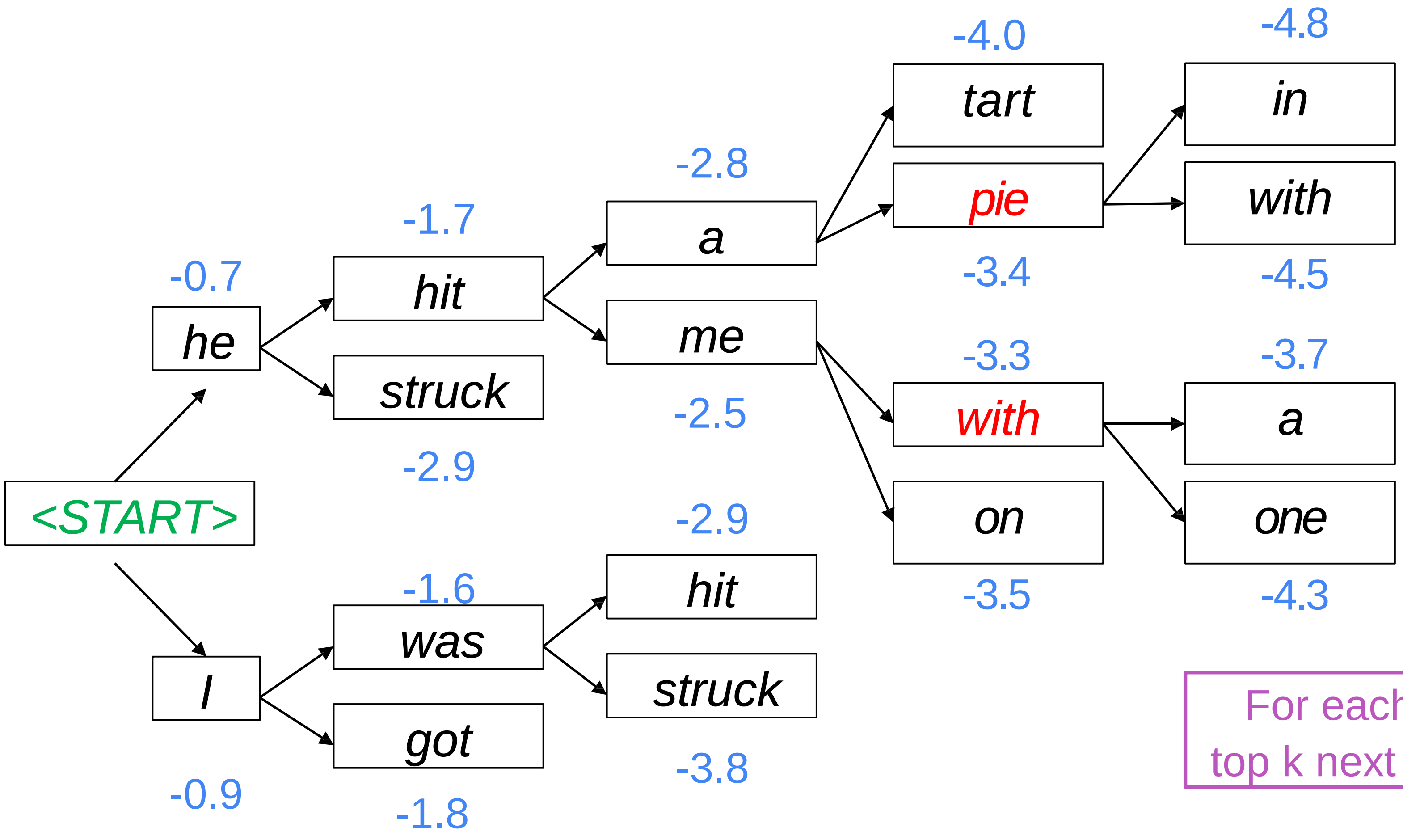
Beam search decoding: example



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

Source: CS224n

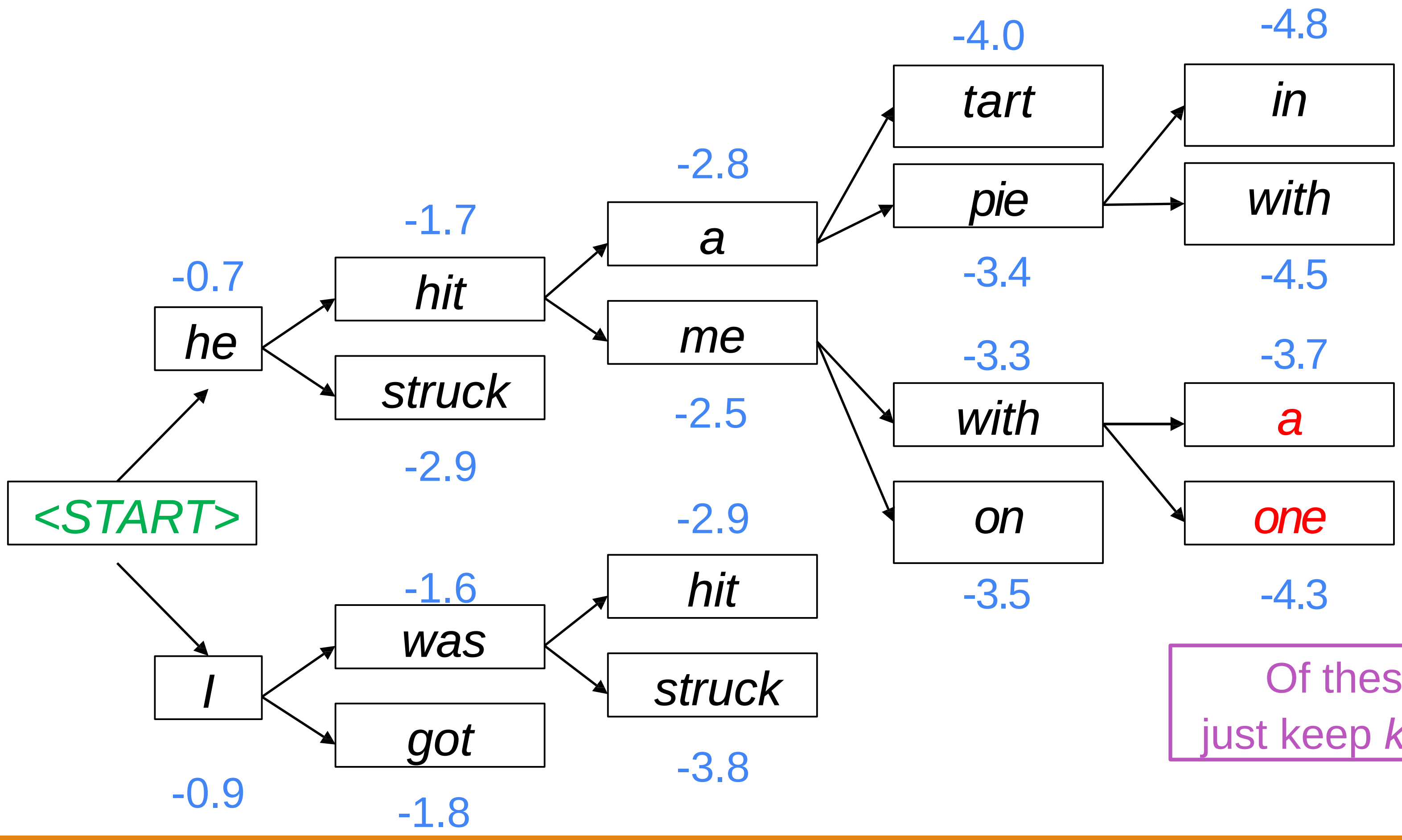
Beam search decoding: example



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

Source: CS224n

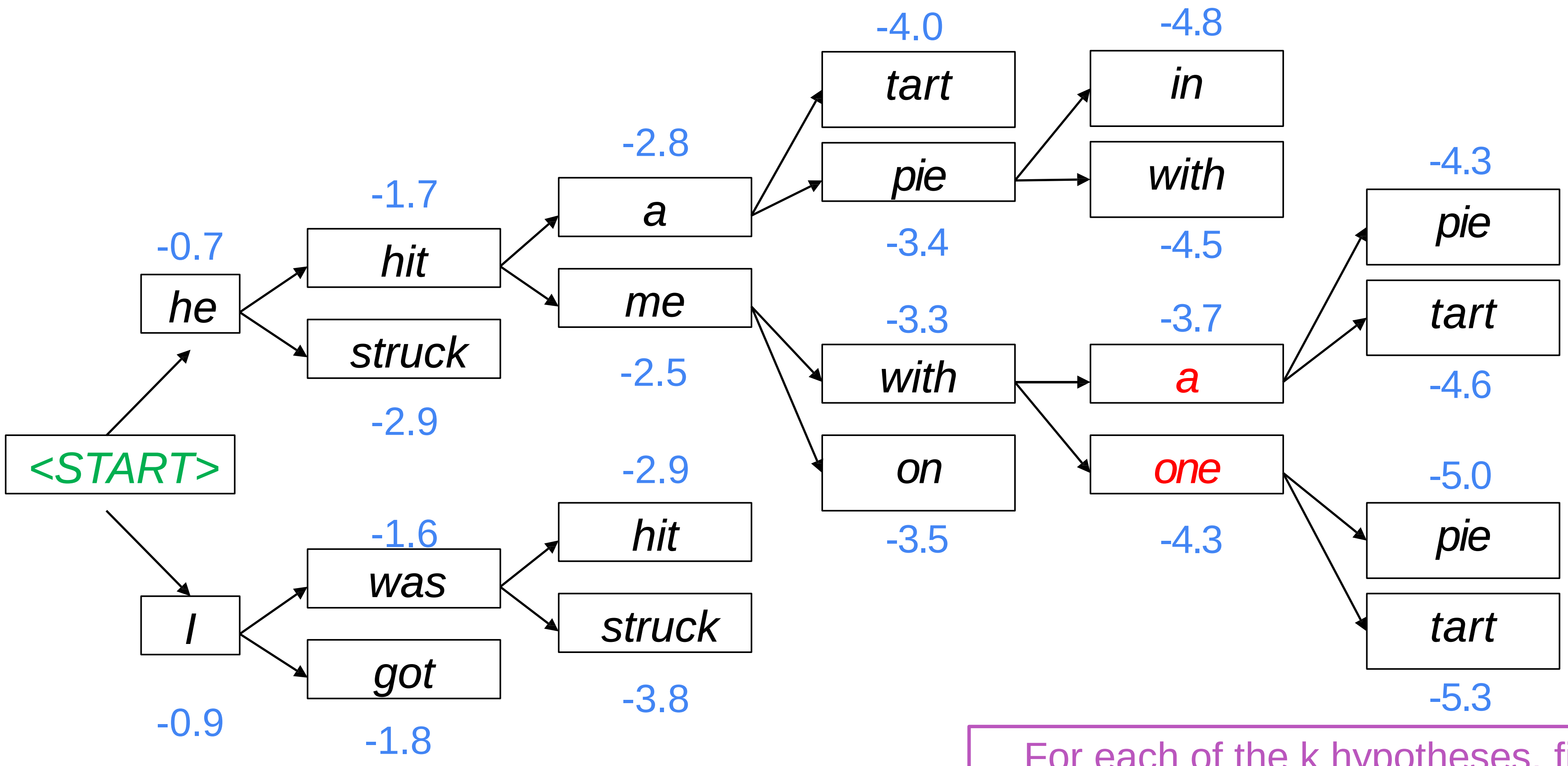
Beam search decoding: example



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

Source: CS224n

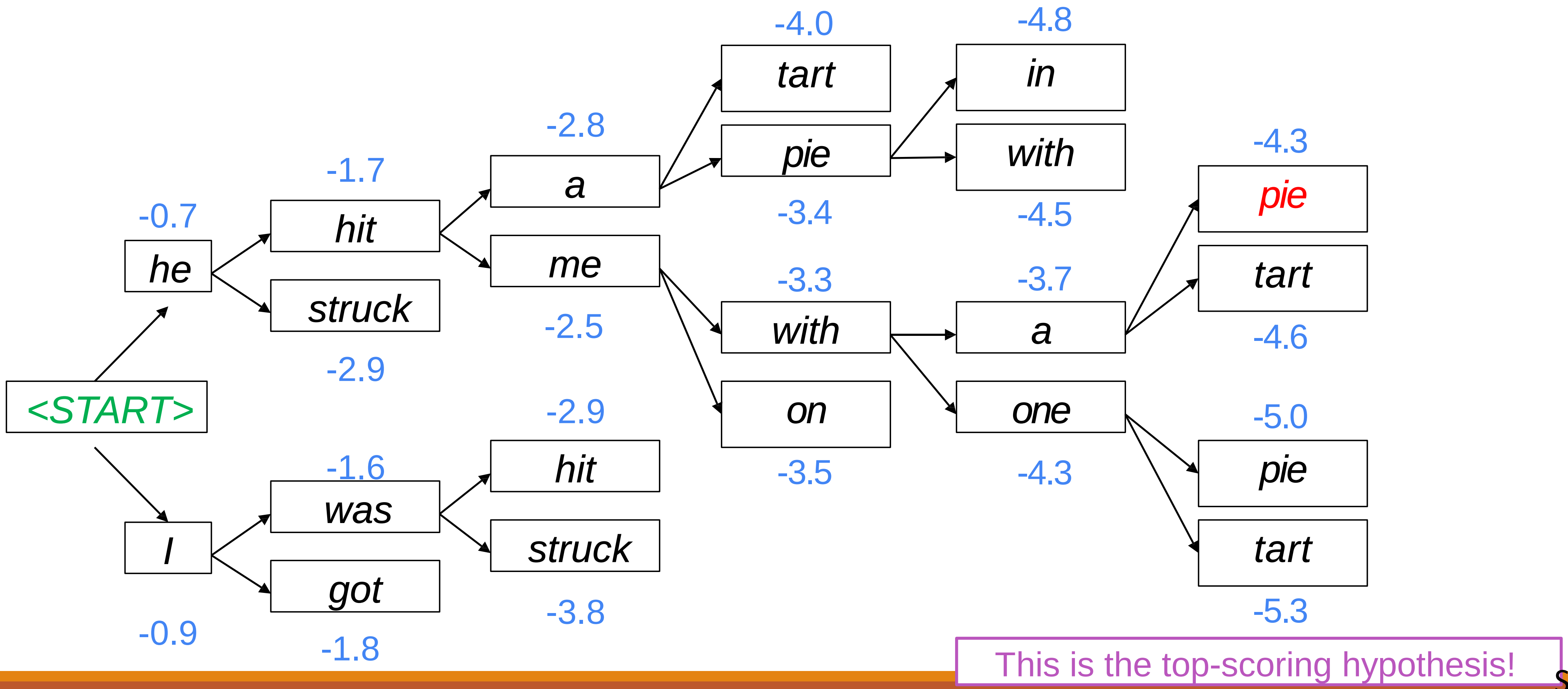
Beam search decoding: example



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

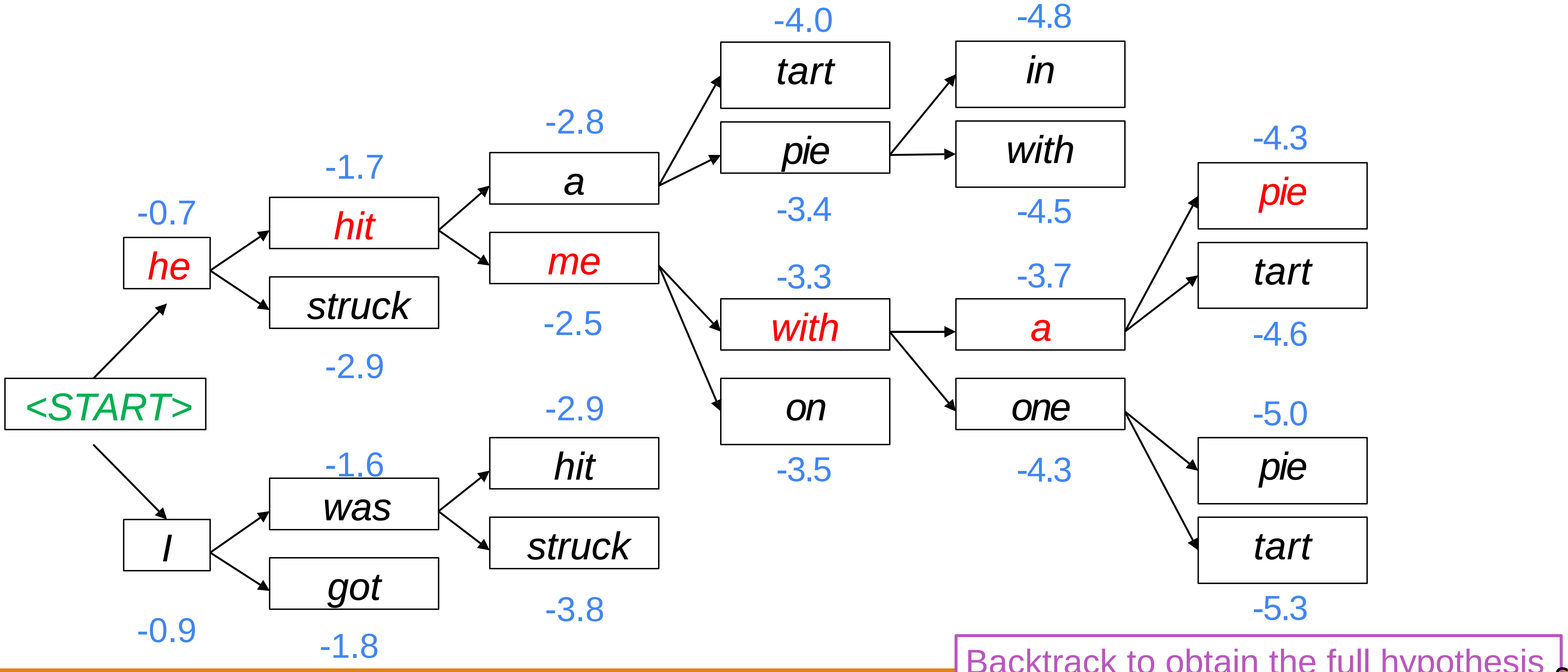
Source: CS224n

Beam search decoding: example



Source: CS224n

Beam search decoding: example



Beam search decoding: stopping criterion

- In greedy decoding, usually we decode until the model produces a <END> token
  - For example: *<START> he hit me with a pie <END>*
- In beam search decoding, different hypotheses may produce <END> tokens on different timesteps
  - When a hypothesis produces <END>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.
- Usually we continue beam search until:
  - We reach timestep T (where T is some pre-defined cutoff), or
  - We have at least n completed hypotheses (where n is pre-defined cutoff)

Source: CS224n

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:

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

Source: CS224n

Hands-on: Beam search

- class BeamSearchNode
- def beam_decode

https://github.com/budzianowski/PyTorch-Beam-Search-Decoding

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

Source: CS224n

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!

Source: CS224n

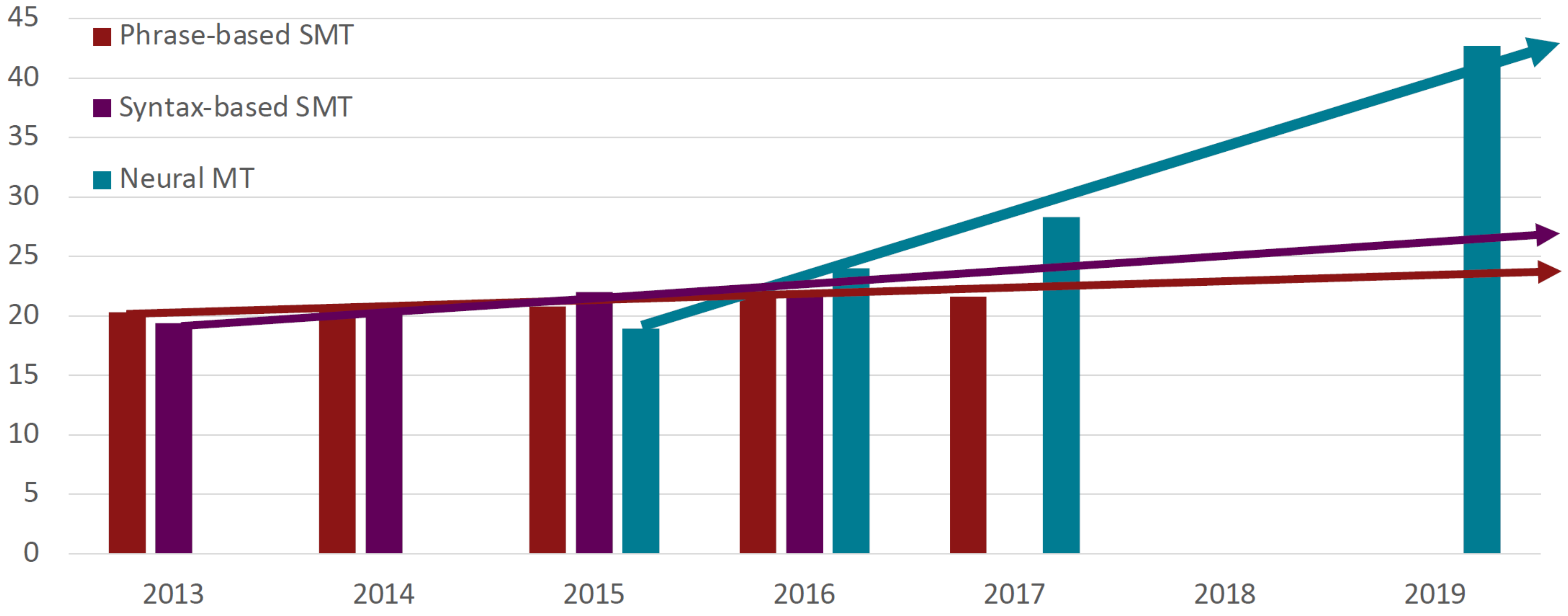
How do we evaluate Machine Translation?

- 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

Source: ” BLEU: a Method for Automatic Evaluation of Machine Translation", Papineni et al, 2002.

Source: CS224n

MT progress over time



NMT: the biggest success story of NLP Deep Learning

- Neural Machine Translation went from a fringe research activity in 2014 to the leading standard method in 2016
  - 2014: First seq2seq paper published
  - 2016: Google Translate switches from SMT to NMT
- This is amazing!
  - **SMT** systems, built by hundreds of engineers over many years, outperformed by **NMT** systems trained by a handful of engineers in a few months

Source: CS224n

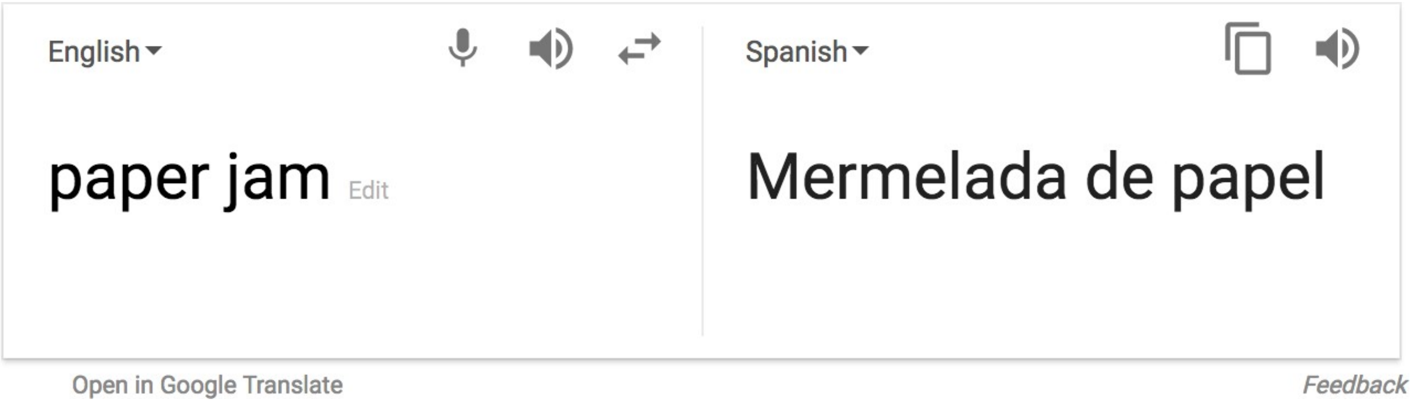
So is Machine Translation solved?

- 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
- Idioms are difficult to translate

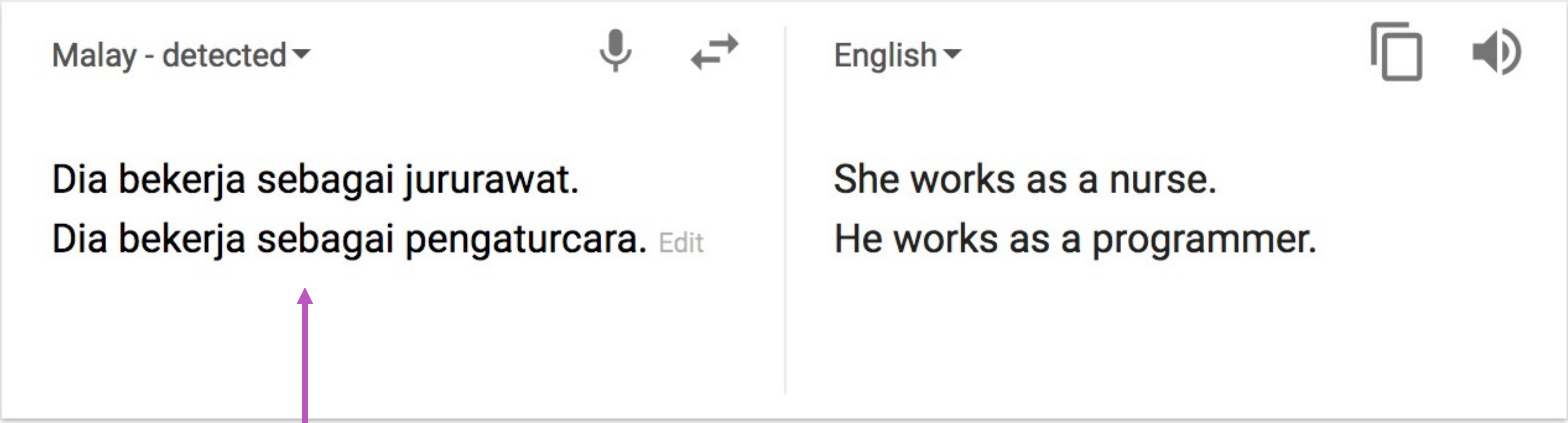


?

Source: CS224n

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

Source: CS224n

NMT research continues

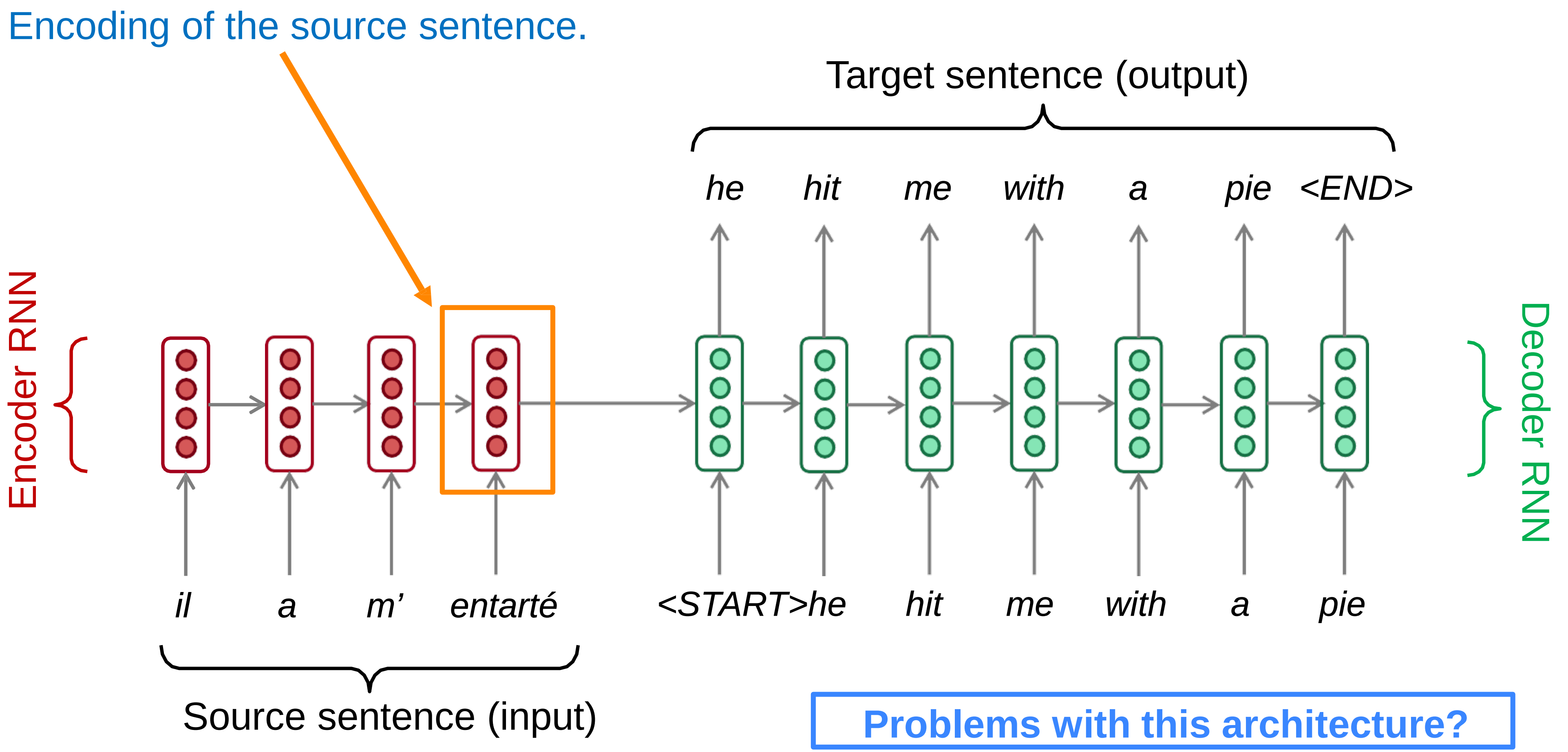
- NMT research has pioneered many of the recent innovations of NLP Deep Learning
- Researchers have found many improvements to the “vanilla” seq2seq NMT system we’ve presented today
- But one improvement is so integral that it is the new vanilla…

ATTENTION

Source: CS224n

Attention

Sequence-to-sequence: the bottleneck problem



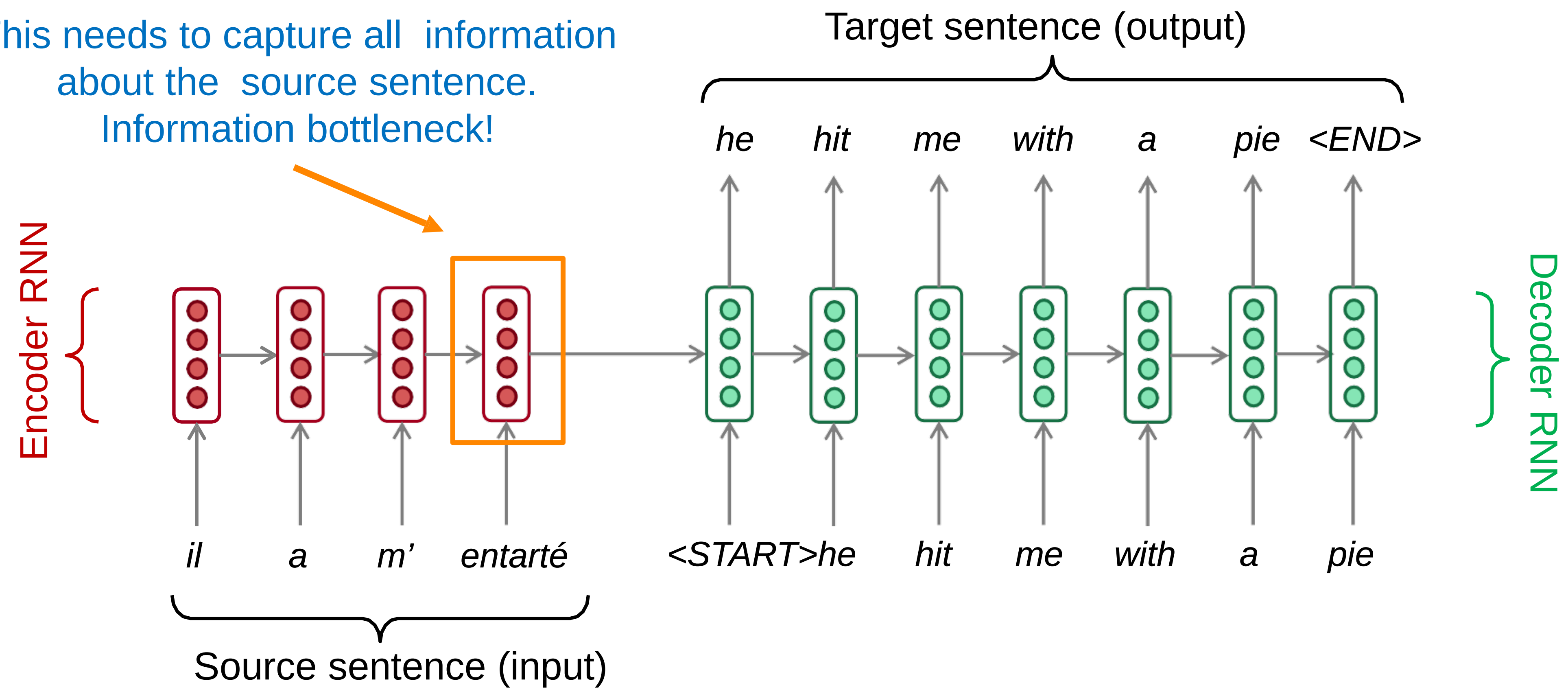
Problems with this architecture?

Source: CS224n

Sequence-to-sequence: the bottleneck problem

Encoding of the source sentence.

This needs to capture all information about the source sentence. Information bottleneck!



Source: CS224n

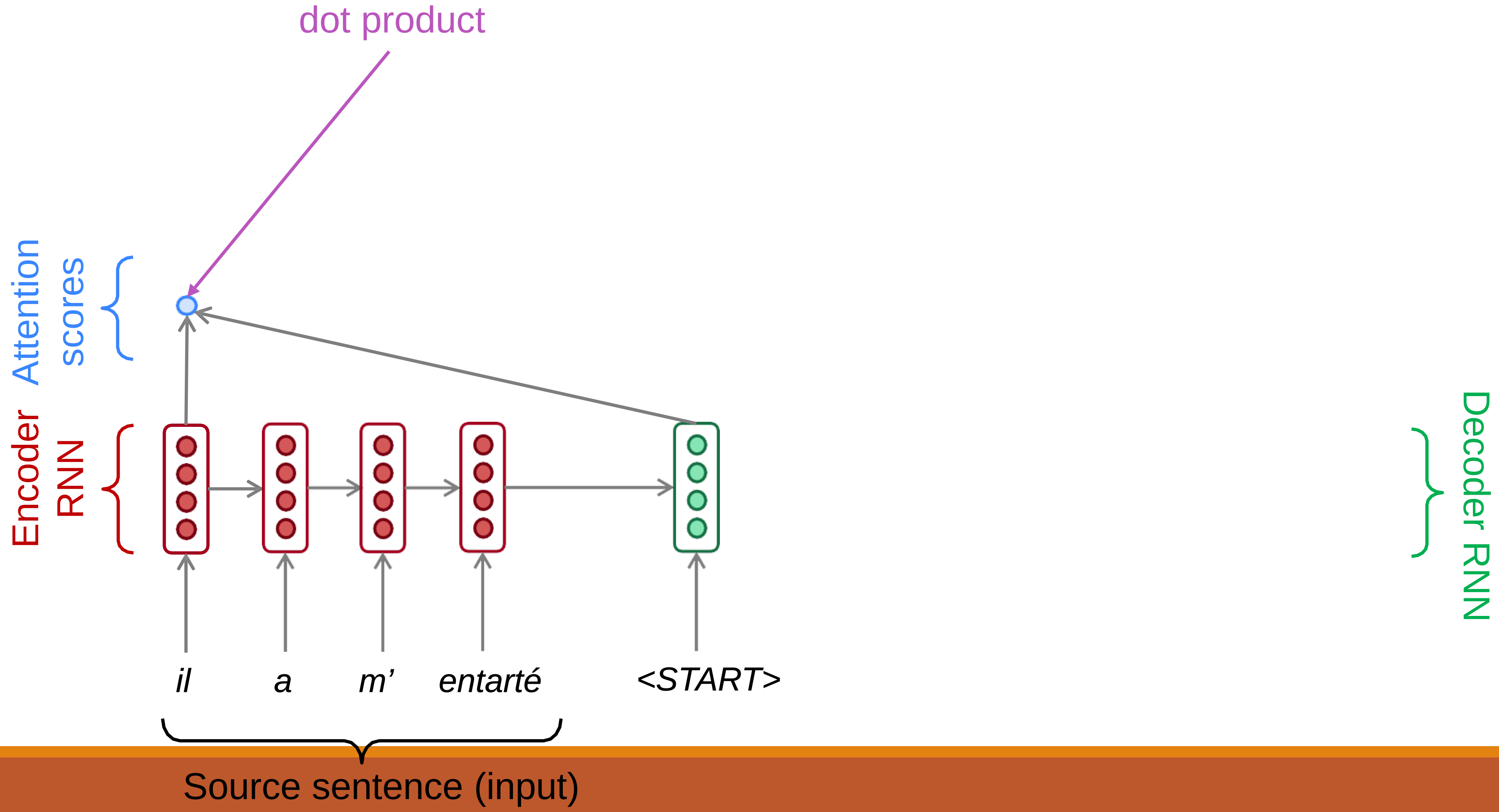
Attention

- 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

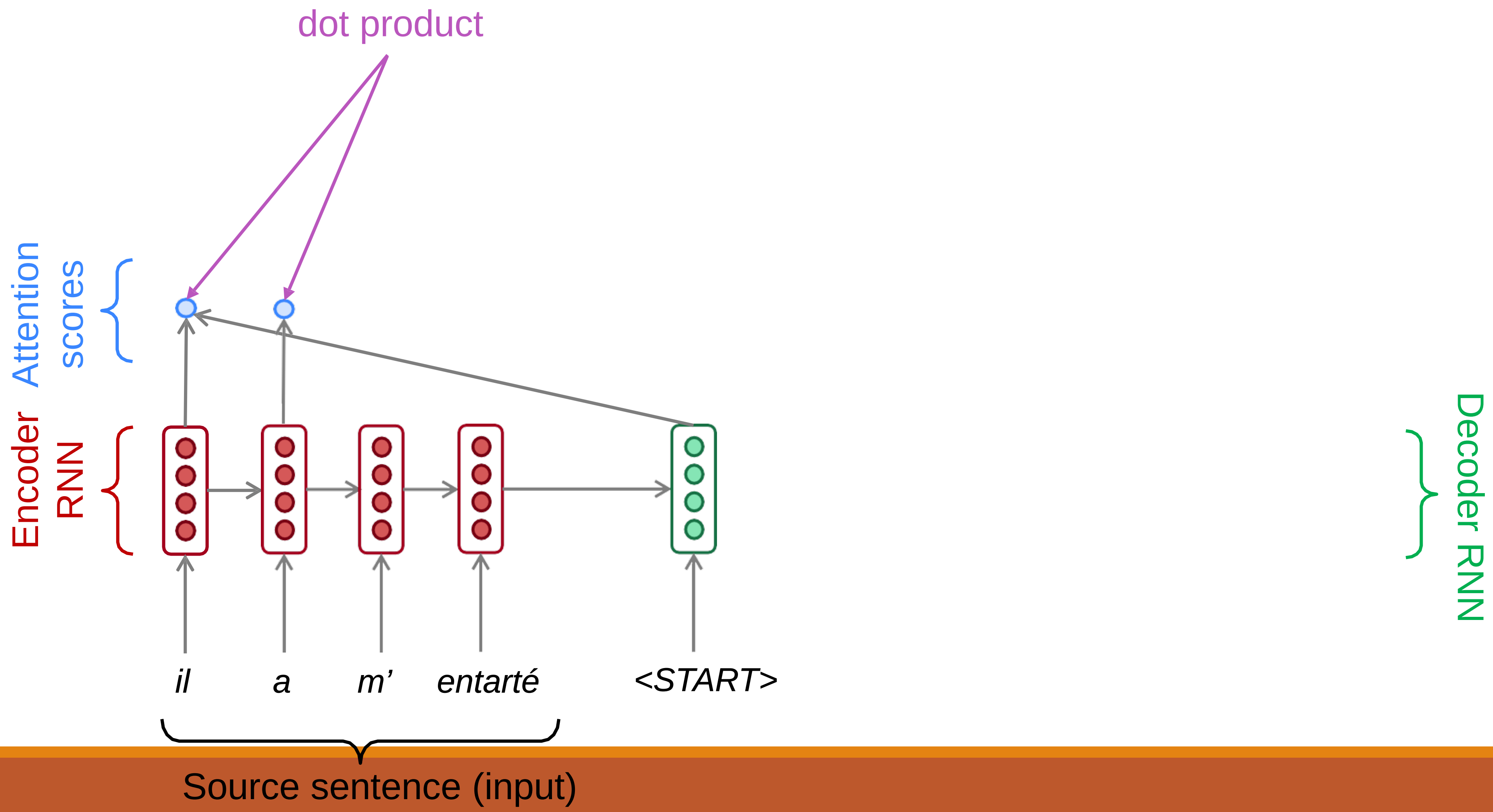
Source: CS224n

Sequence-to-sequence with attention



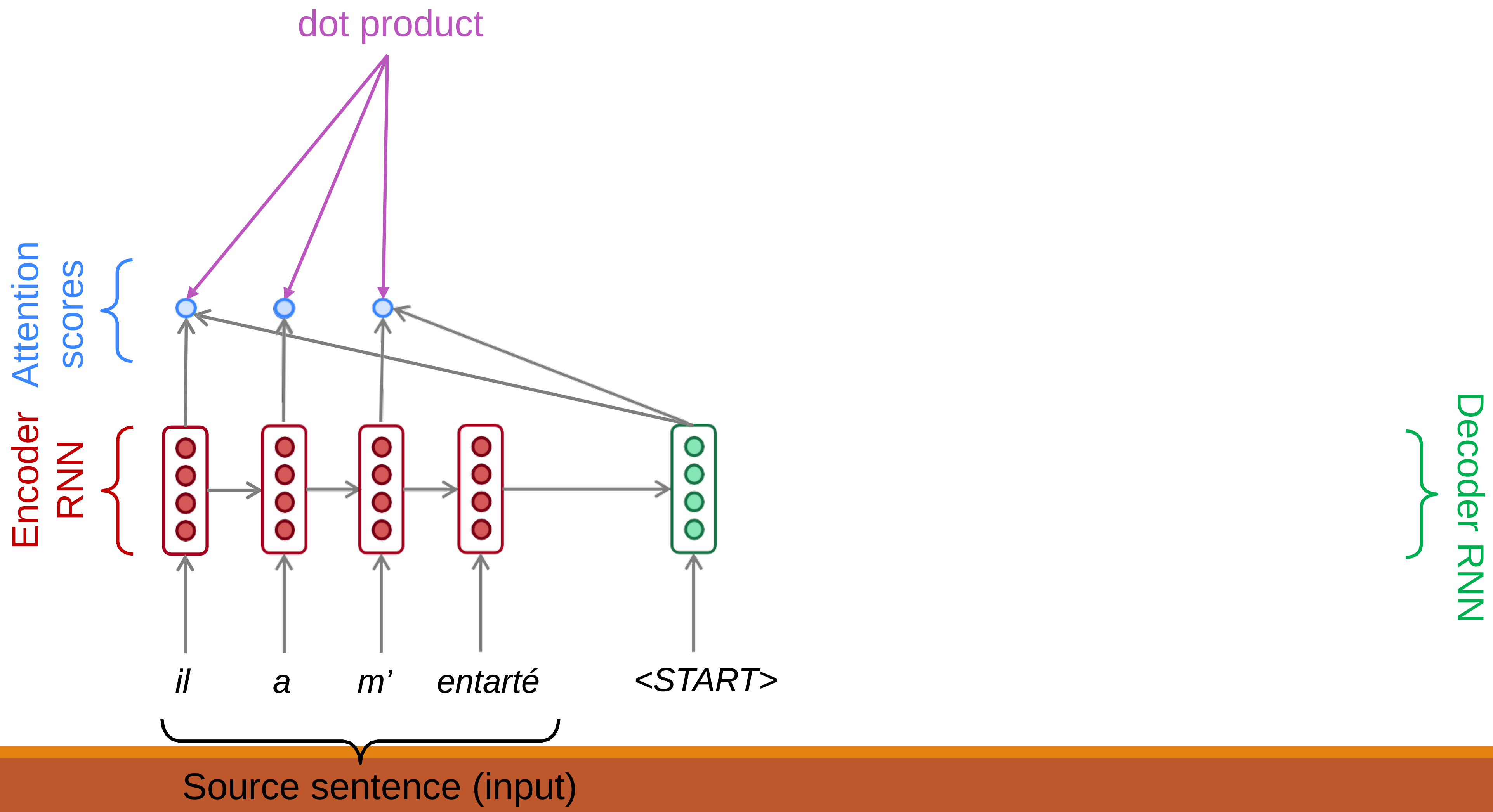
Source: CS224n

Sequence-to-sequence with attention



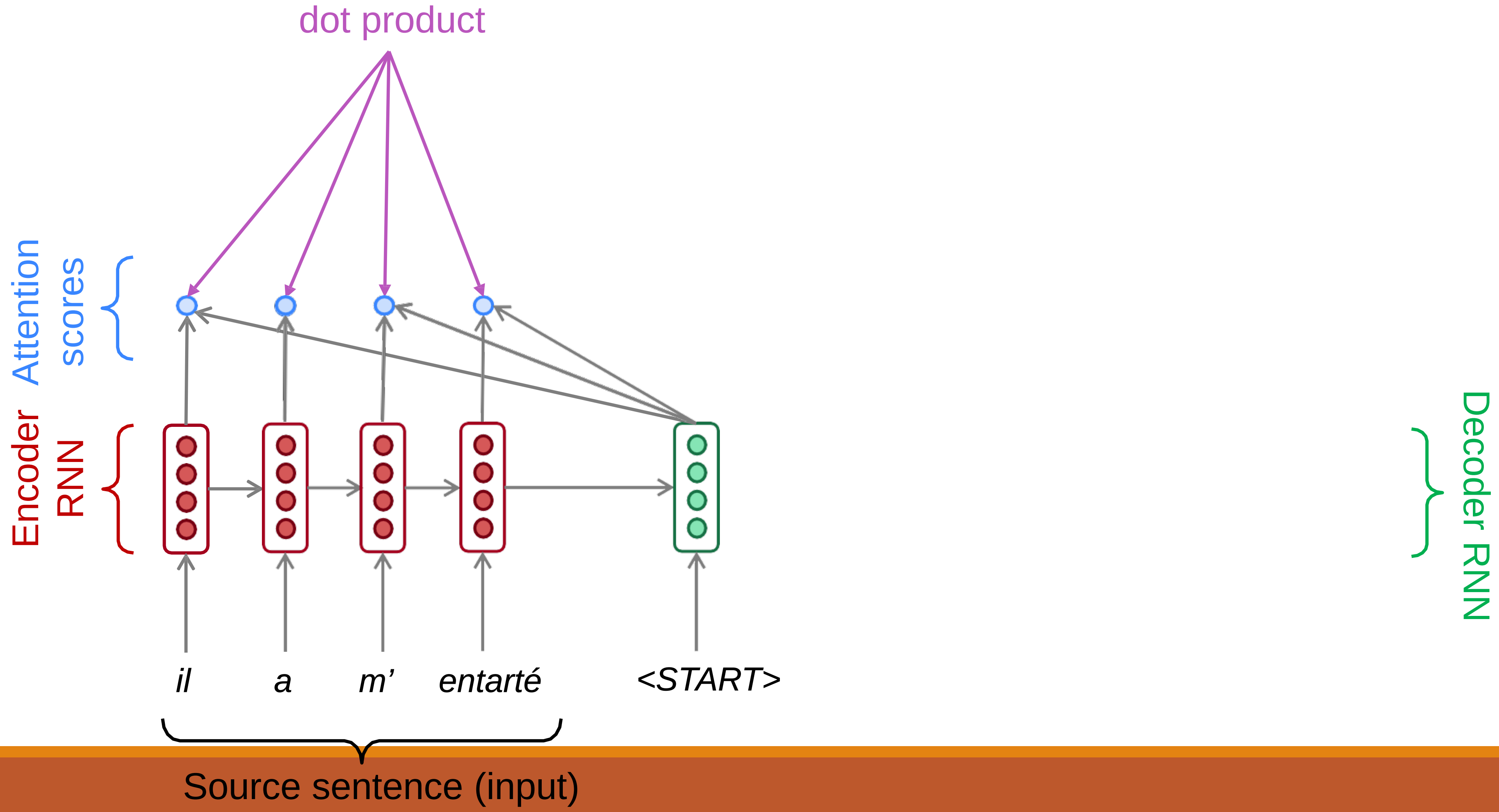
Source: CS224n

Sequence-to-sequence with attention



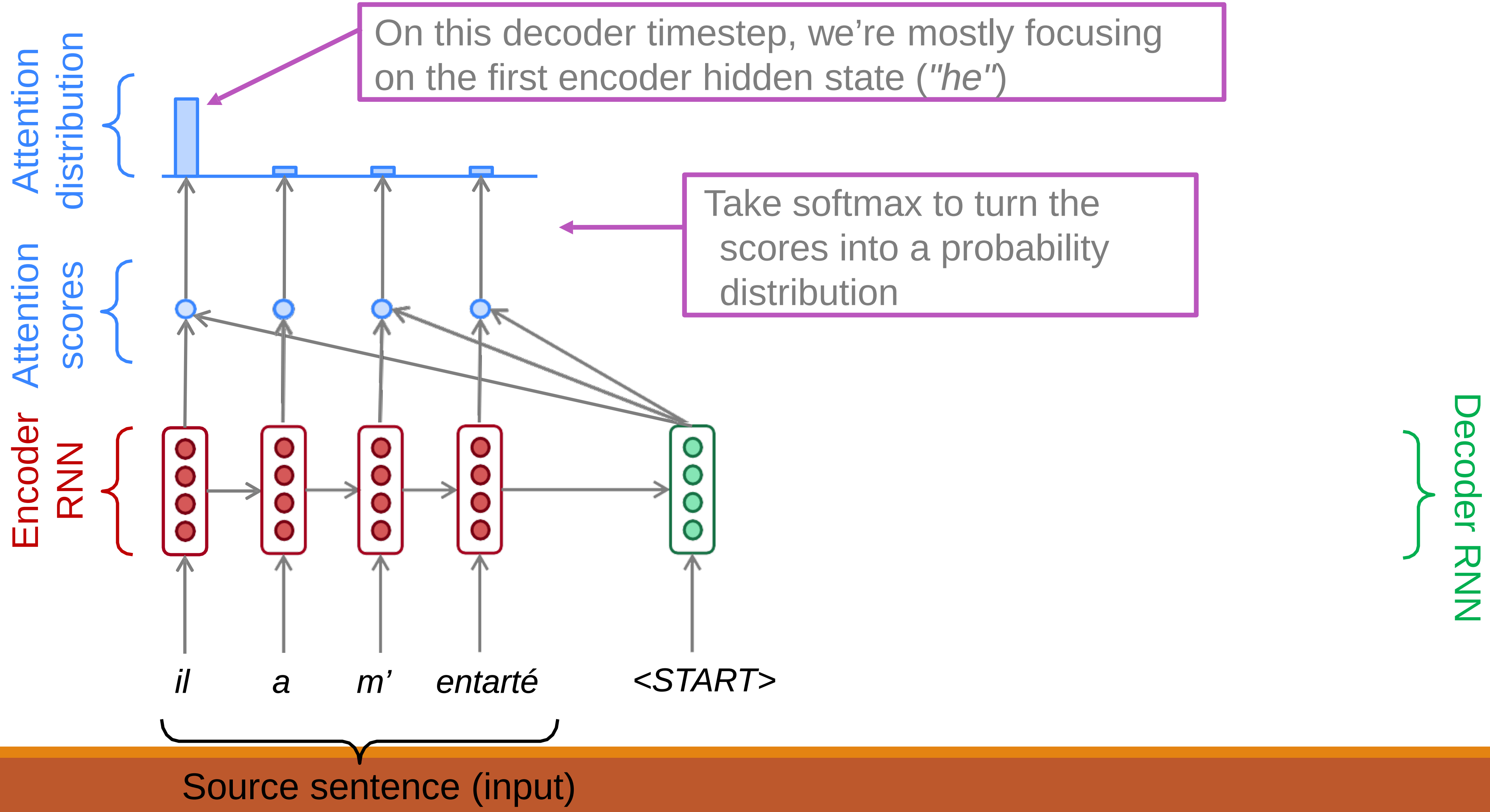
Source: CS224n

Sequence-to-sequence with attention



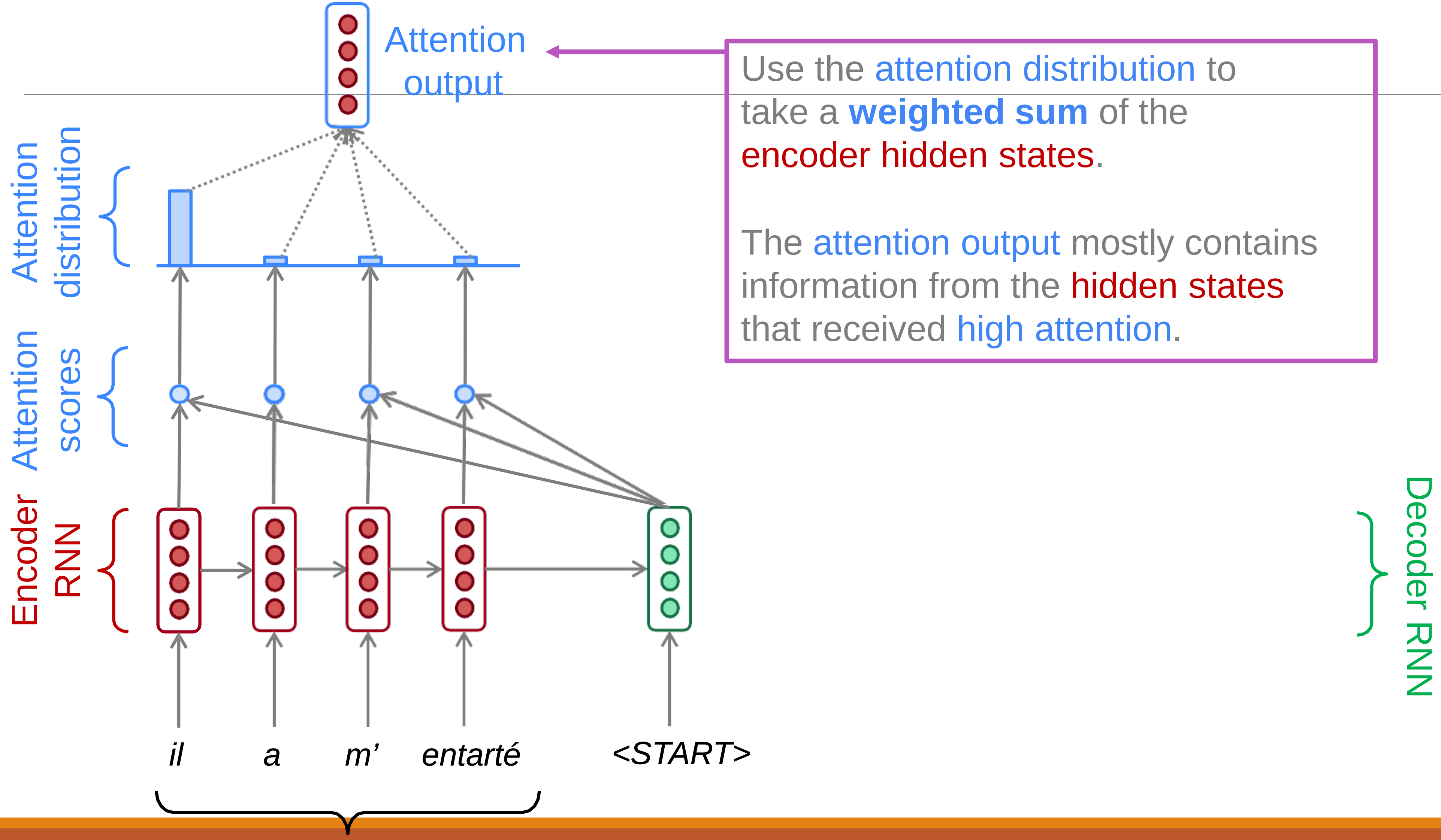
Source: CS224n

Sequence-to-sequence with attention



Source: CS224n

Sequence-to-sequence with attention

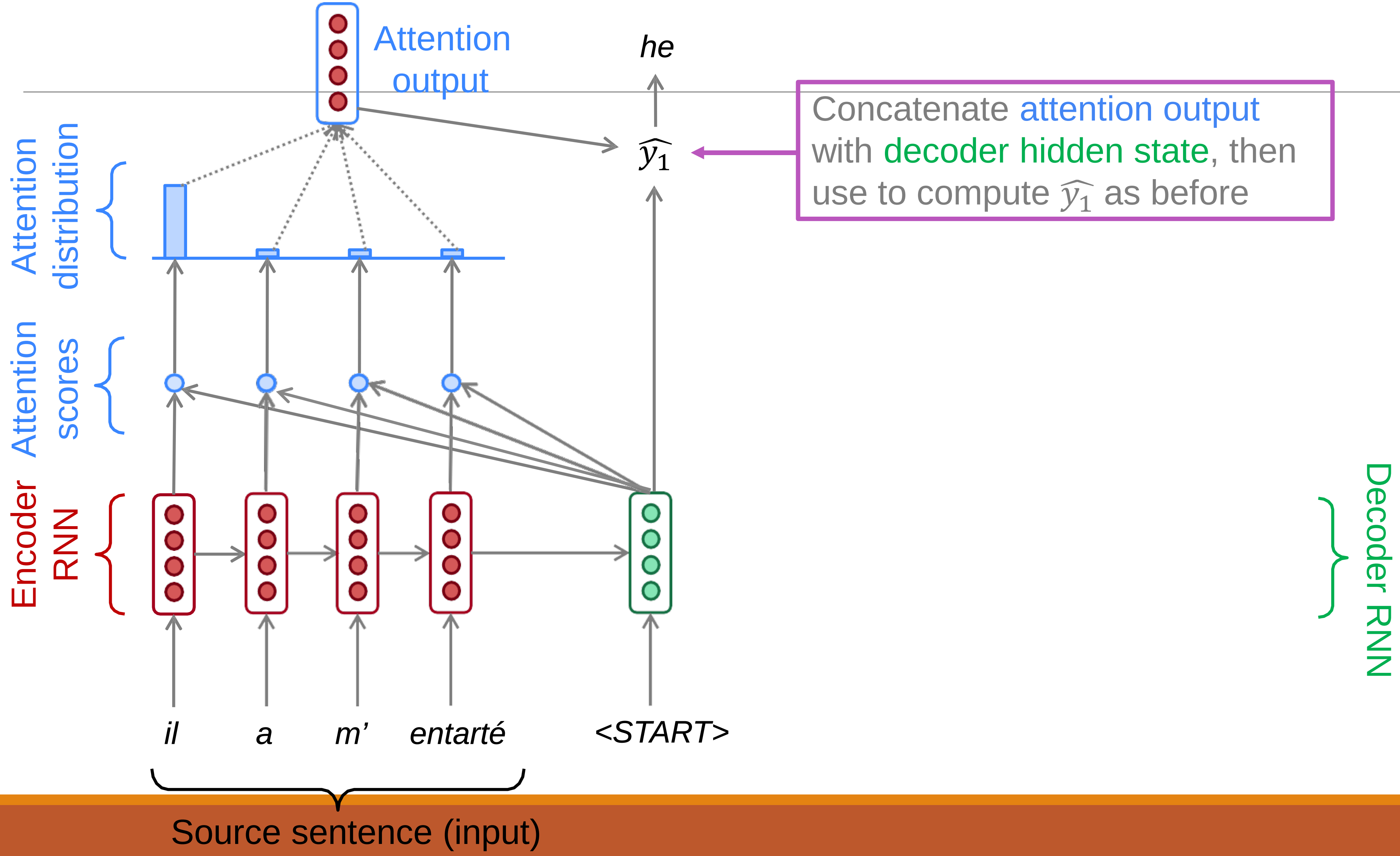


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

Source: CS224n

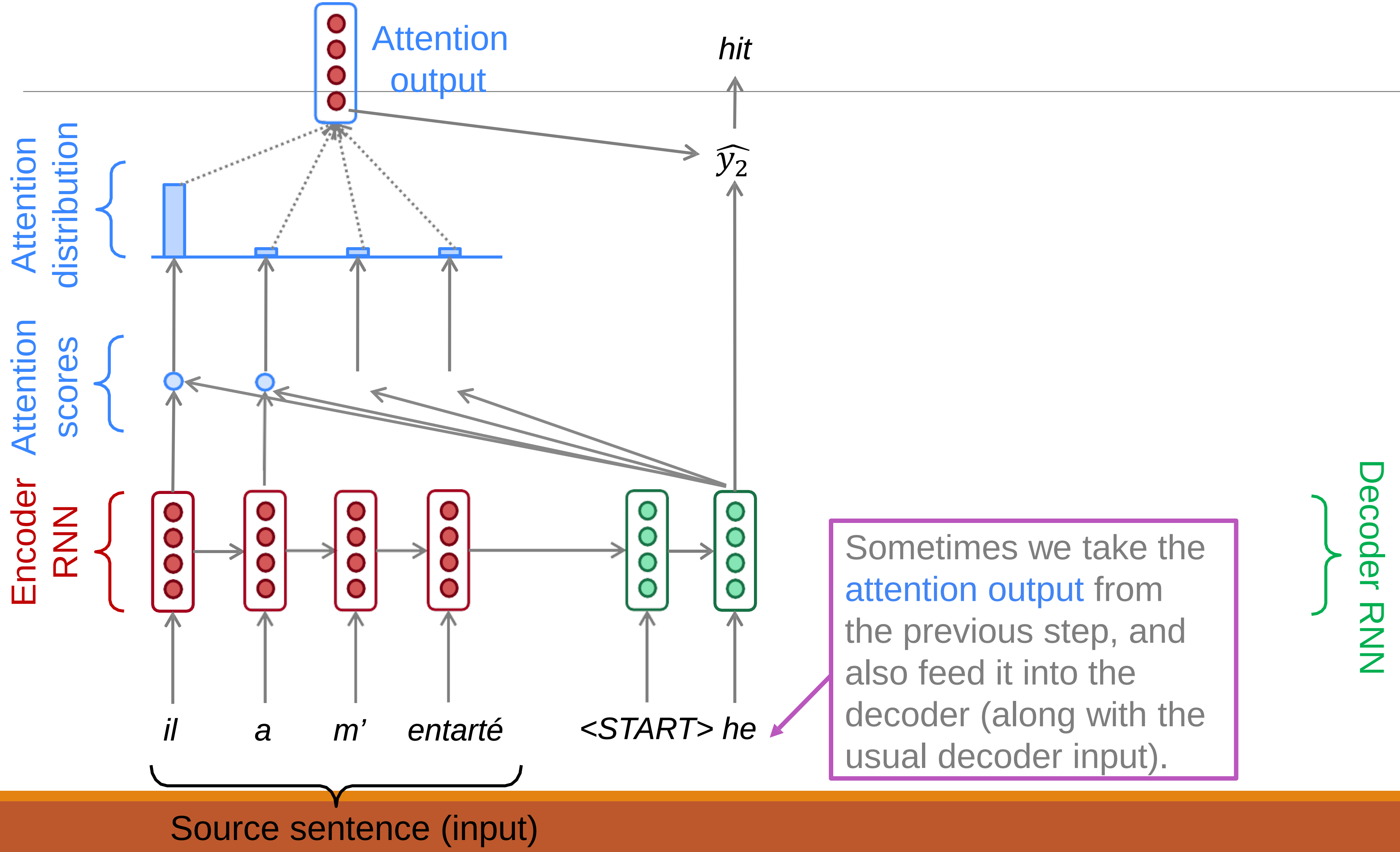
Sequence-to-sequence with attention



Concatenate attention output with decoder hidden state, then use to compute $\widehat{y_1}$ as before

Source: CS224n

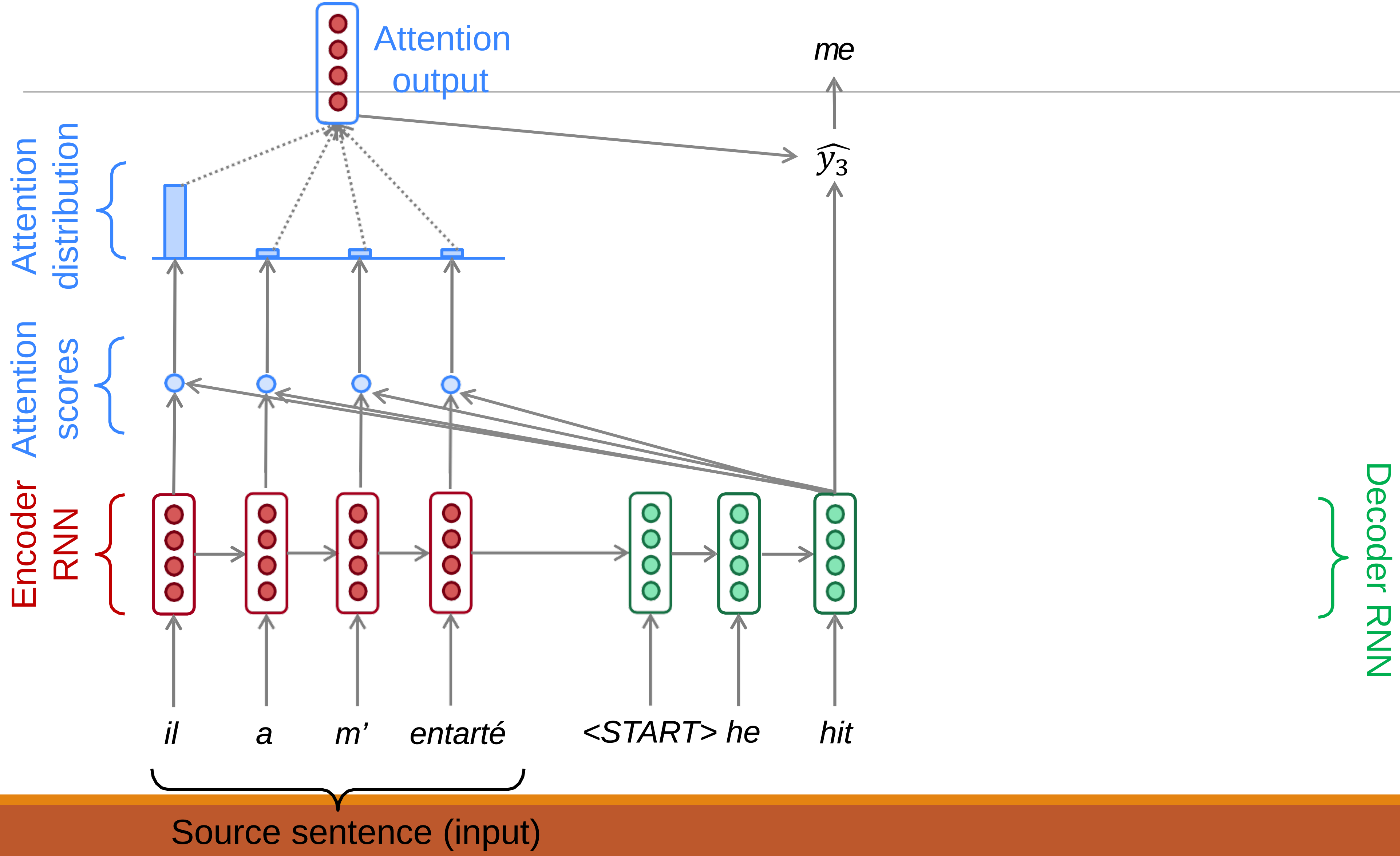
Sequence-to-sequence with attention



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

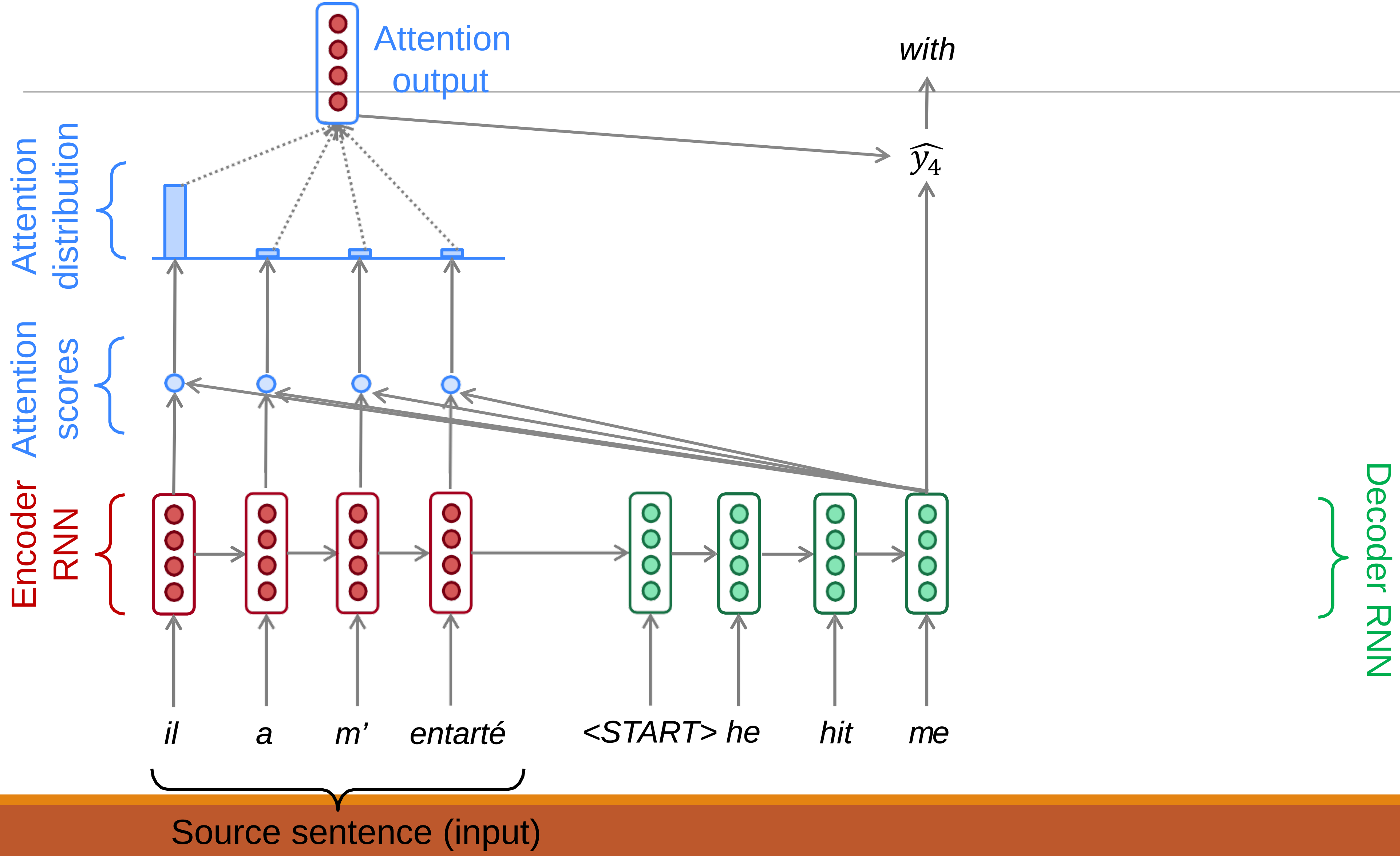
Source: CS224n

Sequence-to-sequence with attention



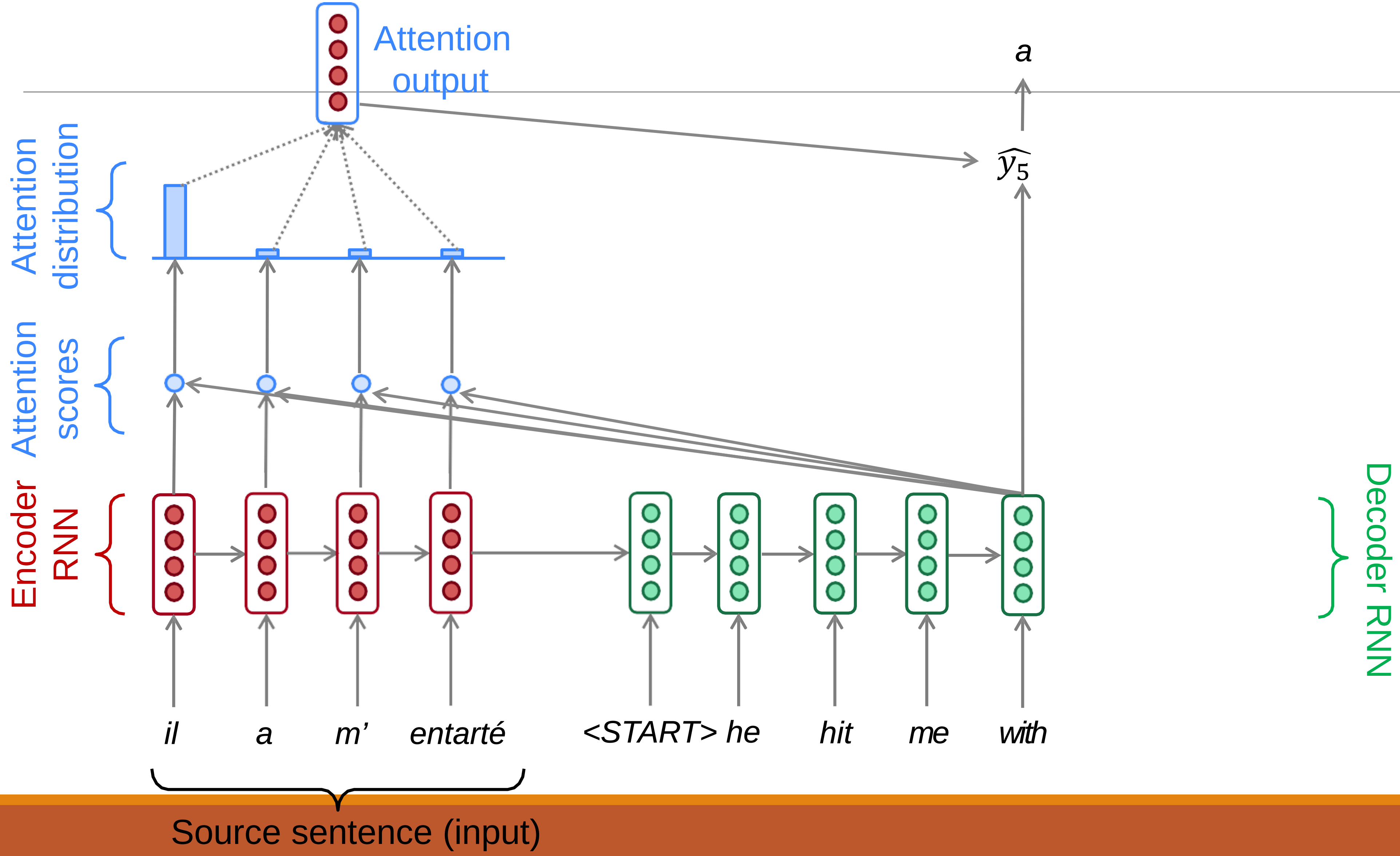
Source: CS224n

Sequence-to-sequence with attention



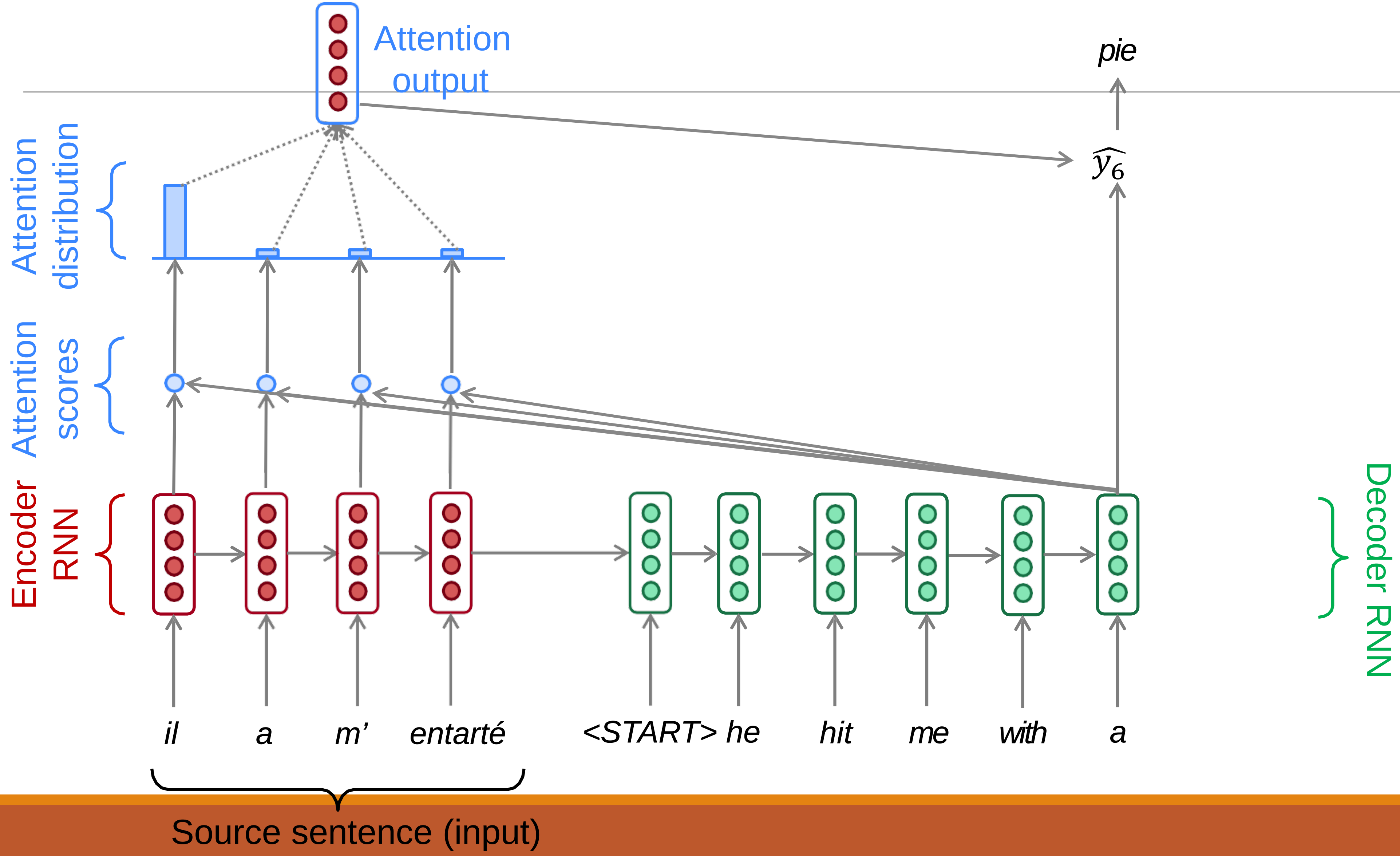
Source: CS224n

Sequence-to-sequence with attention



Source: CS224n

Sequence-to-sequence with attention



Source: CS224n

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 $\boldsymbol{e}^t$ for this step: $\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \dots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$
- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1) $\alpha^t = \mathrm{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$
- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

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

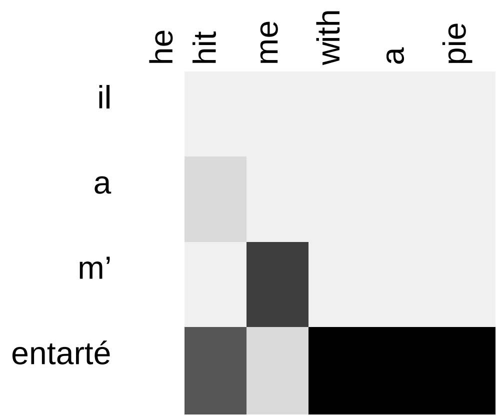
- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

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

Source: CS224n

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



Source: CS224n

Hands-on: MT using seq2seq with attention

- Load data files
- Seq2Seq model
  - Attention decoder
- Training
- Evaluation
  - Visualizing attention

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).

Source: CS224n

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).

Source: CS224n

Attention variants

- There are several ways you can compute $\boldsymbol{e} \in \mathbb{R}^N$ from $\boldsymbol{h}_1, \cdots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ 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: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$
  - where $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix
- Additive attention: $\boldsymbol{e}_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s}) \in \mathbb{R}$
  - 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

"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

Source: CS224n