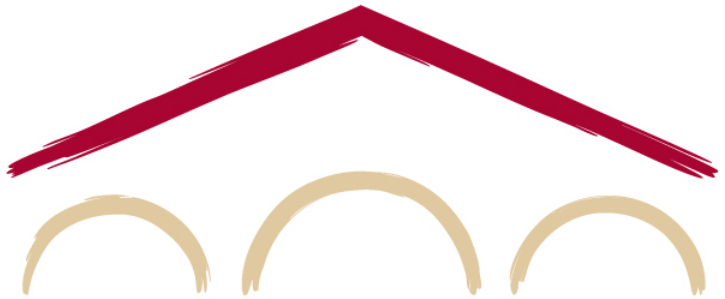
Natural Language Processing with Deep Learning CS224N/Ling284



Christopher Manning

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

Section 1: Pre-Neural Machine Translation

Machine Translation

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

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

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

– Rousseau

The early history of MT: 1950s

- Machine translation re
  powerful than high sc
- Foundational work on
  information theory
- MT heavily funded by
  systems doing word s
- Human language is m
  languages!
- Little understanding c
- Problem soon appeare

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

1990s-2010s: Statistical Machine Translation

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

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

- 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.

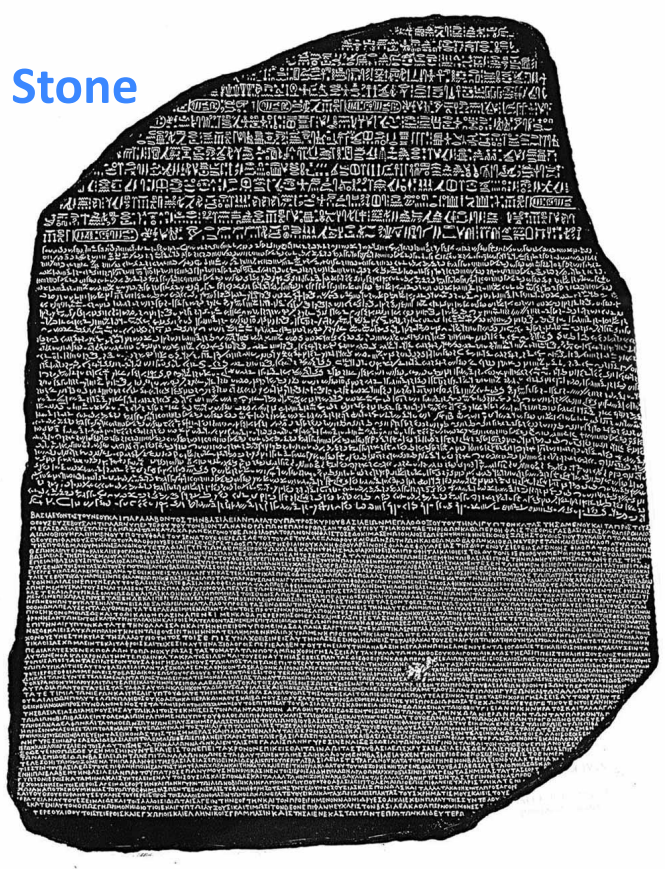
Language Model

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

1990s-2010s: Statistical Machine Translation

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

The Rosetta Stone



Ancient Egyptian

Demotic

Ancient Greek

The Rosetta Stone

Three different translations of the same text:

- Hieroglyphic Egyptian (used by priests)
- Demotic Egyptian (used for daily purposes)
- Classical Greek (used by the administration)

Instrumental in our understanding of ancient Egyptian

This is an instance of parallel text:

The Greek inscription allowed scholars to decipher the hieroglyphs

Machine Translation History

WW II: Code-breaking efforts at Bletchley Park, England (Alan Turing)

1948: Shannon/Weaver: Information theory

1949: Weaver's memorandum defines the machine translation task

1954: IBM/Georgetown demo: 60 sentences Russian-English

1960: Bar-Hillel: MT to difficult

1966: ALPAC report: human translation is far cheaper and better:
kills MT for a long time

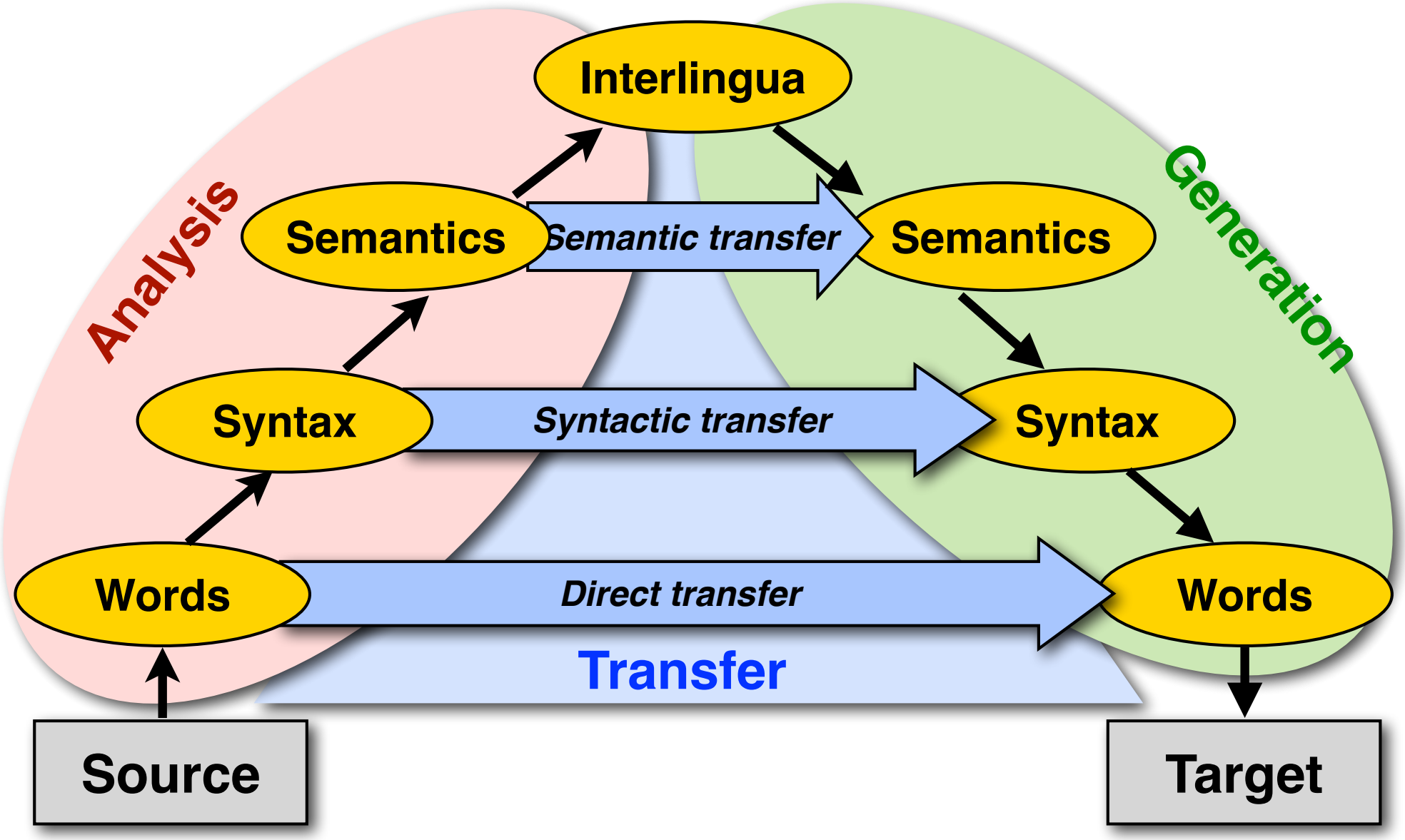
1980s/90s: Transfer and interlingua-based approaches

1990: IBM's CANDIDE system (first modern statistical MT system)

2000s: Huge interest and progress in wide-coverage statistical MT:
phrase-based MT, syntax-based MT, open-source tools

since mid/late 2010's: Neural machine translation
(seq2seq models with attention)

The Vauquois triangle

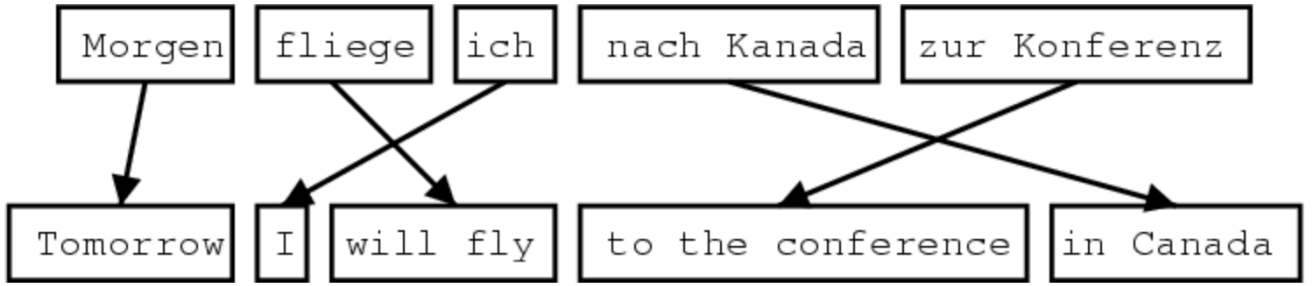


Learning alignment for SMT

- Question: How to learn translation model $P(x|y)$ from the parallel corpus?

- Break it down further: Introduce latent *a* variable into the model: $P(x, a|y)$

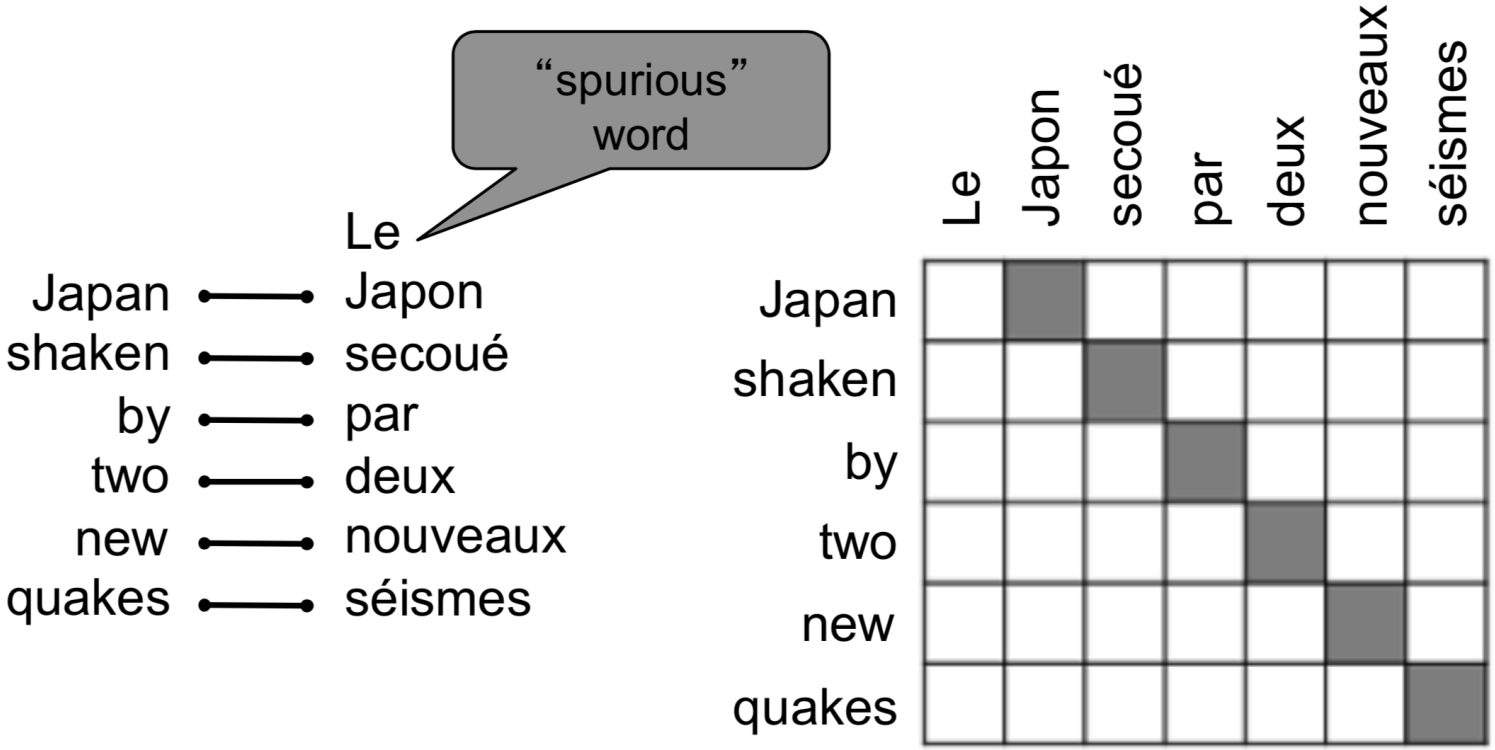
  where *a* is the alignment, i.e. word-level correspondence between source sentence *x* and target sentence *y*



What is alignment?

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

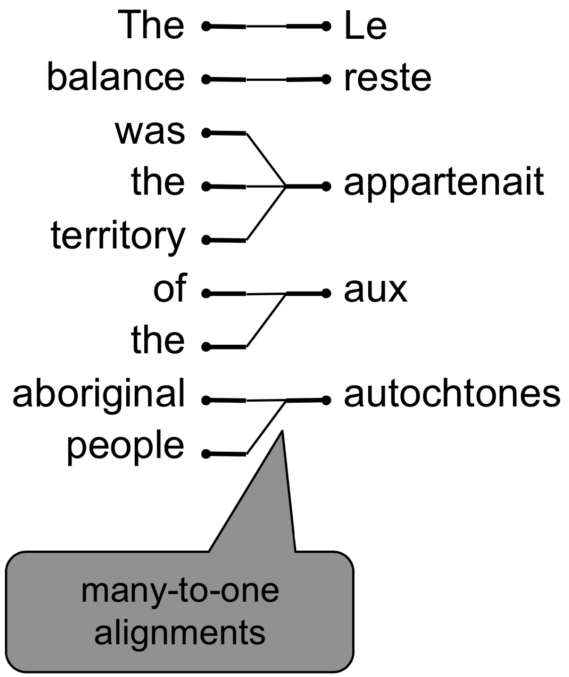
- Typological differences between languages lead to complicated alignments!
- Note: Some words have no counterpart

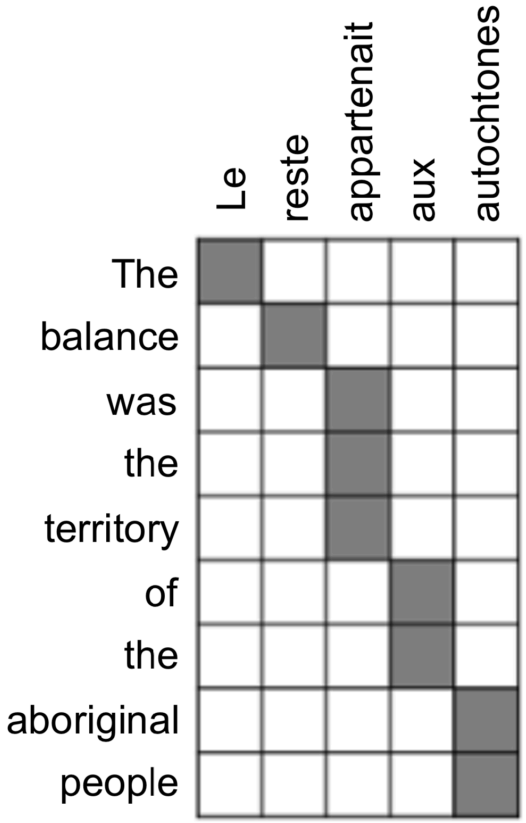


Examples from: "The Mathematics of Statistical Machine Translation: Parameter Estimation", Brown et al, 1993. http://www.aclweb.org/anthology/J93-2003

Alignment is complex

Alignment can be many-to-one

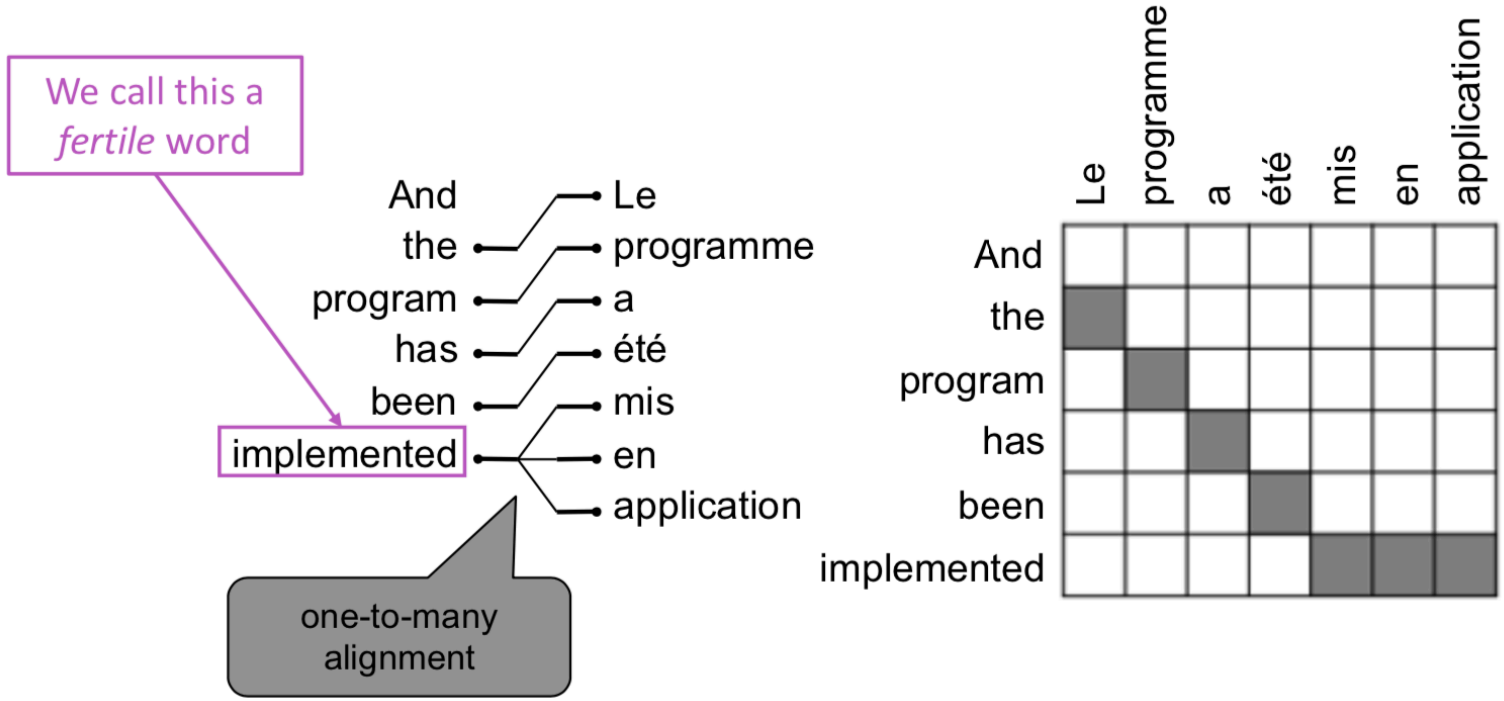




Examples from: "The Mathematics of Statistical Machine Translation: Parameter Estimation", Brown et al, 1993. http://www.aclweb.org/anthology/J93-2003

Alignment is complex

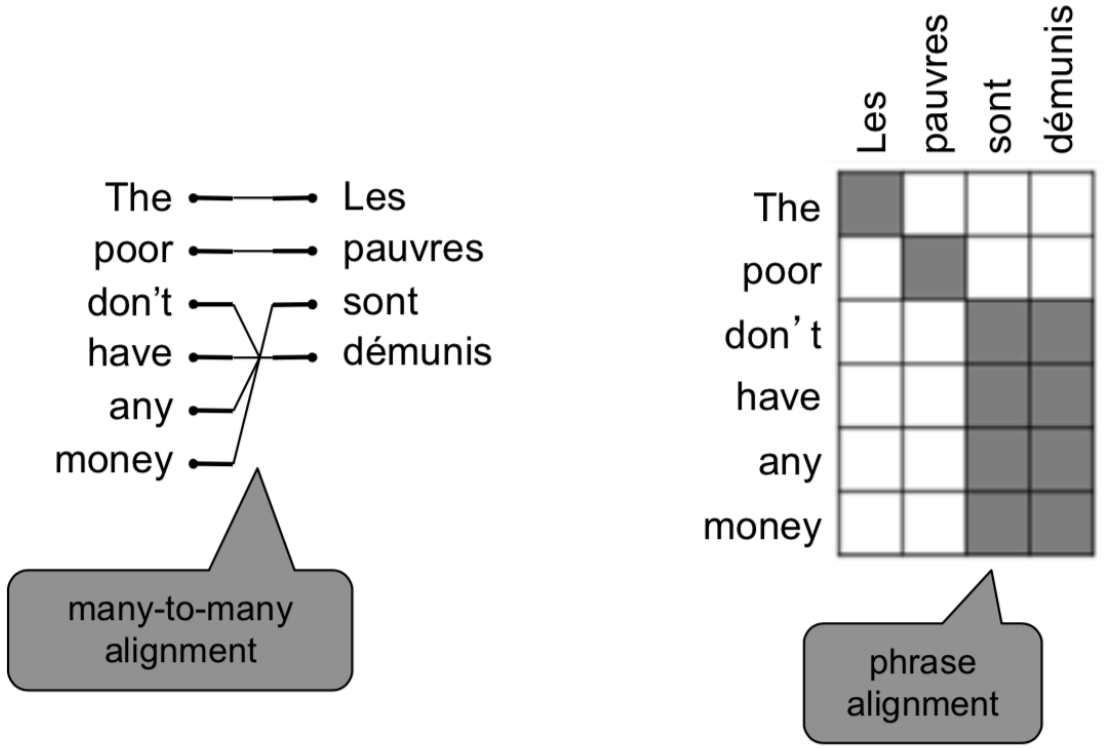
Alignment can be one-to-many



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

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

Lexical divergences

The different senses of **homonymous words** generally have different translations:

English-German: (river) bank - Ufer
(financial) bank - Bank

The different senses of **polysemous words** may also have different translations:

I **know that** he bought the book: Je **sais qu**'il a acheté le livre.

I **know** Peter: Je **connais** Peter.

I **know** math: Je **m'y connais en** maths**.**

Lexical divergences

Lexical specificity

German ***Kürbis*** = English ***pumpkin*** or ***(winter) squash***

English ***brother*** = Chinese ***gege*** (older) or ***didi*** (younger)

Morphological divergences

English: ***new*** *book(s),* ***new*** *story/stories*

French: un **nouveau** *livre (sg.m), une* **nouvelle** *histoire (sg.f),*
des **nouveaux** *livres (pl.m), des* **nouvelles** *histoires (pl.f)*

- How much **inflection** does a language have?
  (cf. Chinese vs.Finnish)
- How many **morphemes** does each word have?
- How easily can the morphemes be **separated**?

Syntactic divergences

Word order: fixed or free?

If fixed, which one? [SVO (Sbj-Verb-Obj), SOV, VSO,…]

Head-marking vs. dependent-marking

Dependent-marking (English) *the man's house*

Head-marking (Hungarian) *the man house-**his***

Pro-drop languages can omit pronouns:

Italian (with inflection): *I eat = mangi**o**; he eats = mangi**a***

Chinese (without inflection): *I/he eat: chīfàn*

Syntactic divergences: negation

	Normal	**Negated**	
English	I drank coffee.	*I **didn't drink (any)** coffee.*	*do*-support, *any*
French	J'ai bu du café	*Je **n'ai pas** bu **de** café.*	ne..pas *du* → *de*
German	Ich habe Kaffee getrunken	*Ich habe **keinen** Kaffee getrunken*	*keinen Kaffee* = 'no coffee'

Semantic differences

Aspect:

- English has a **progressive aspect**:
  'Peter swims' vs. 'Peter is swimming'
- German can only express this with **an adverb**:
  'Peter schwimmt' vs. 'Peter schwimmt gerade' ('swims currently')

Motion events have two properties:

- **manner** of motion (*swimming*)
- **direction** of motion (*across* the lake)

Languages express either the manner with a verb
and the direction with a 'satellite' or vice versa (L. Talmy):

English (satellite-framed): *He* [*swam*]$_{\text{MANNER}}$ **[*across*]$_{\text{DIR}}$** *the lake*

French (verb-framed): *Il a* **[*traversé*]$_{\text{DIR}}$** *le lac* **[*à la nage*]**$_{\text{MANNER}}$

Learning alignment for SMT

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

Statistical Machine Translation

Given a Chinese input sentence (**source**)…

主席：各位議員，早晨。

…find the best English translation (**target**)

President: Good morning, Honourable Members.

We can formalize this as $\mathrm{T}^* = \mathrm{argmax}_{\mathrm{T}} P(\mathrm{T} \mid \mathrm{S})$

Using **Bayes Rule** simplifies the modeling task, so this was the first approach for statistical MT (the so-called "**noisy-channel model**"):

$$\mathrm{T}^* = \mathrm{argmax}_{\mathrm{T}} P(\mathrm{T} \mid \mathrm{S}) = \mathrm{argmax}_{\mathrm{T}} P(\mathrm{S} \mid \mathrm{T}) P(\mathrm{T})$$

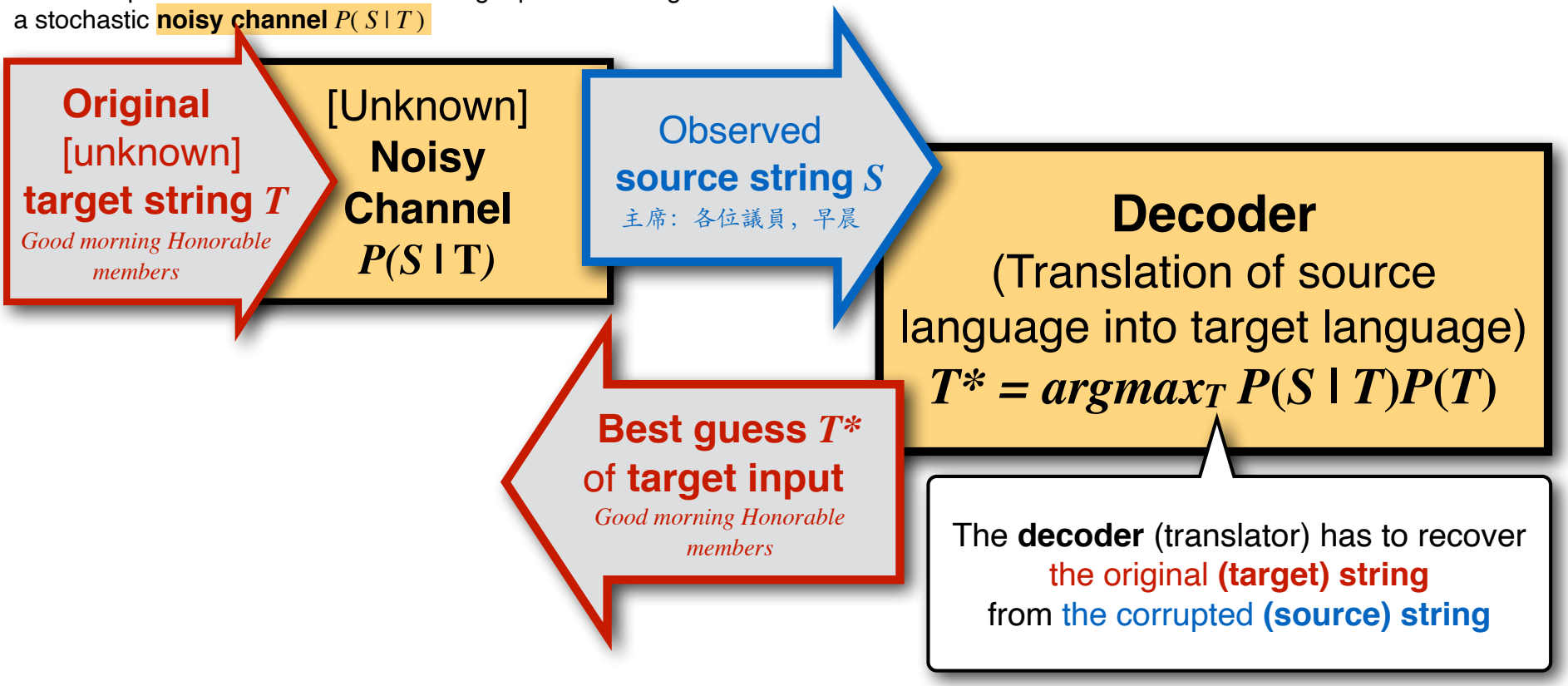
where $P(\mathrm{S} \mid \mathrm{T})$: translation model

$P(\mathrm{T})$: language model

The noisy channel metaphor

$$\text{Target}^* = \text{argmax}_{\text{Target}} \underbrace{P(\text{Source} \mid \text{Target})}_{\text{Translation Model}} \underbrace{P(\text{Target})}_{\text{Language Model}}$$

Noisy Channel Metaphor:
The observed **source string** S that needs to be translated is just a corrupted version of some unknown **original target string** T that translation (decoding) has to recover.
This corruption occurred because the target passed through a stochastic **noisy channel** $P(S \mid T)$



The noisy channel model

This is really just an application of **Bayes' rule**:

$$T^* = \text{argmax}_T \; P(T \mid S)$$

$$= \text{argmax}_T \; \underbrace{P(S \mid T)}_{\text{Translation Model}} \; \underbrace{P(T)}_{\text{Language Model}}$$

The **translation model** $P(S \mid T)$ is intended to capture the **faithfulness** of the translation. [this is the noisy channel]

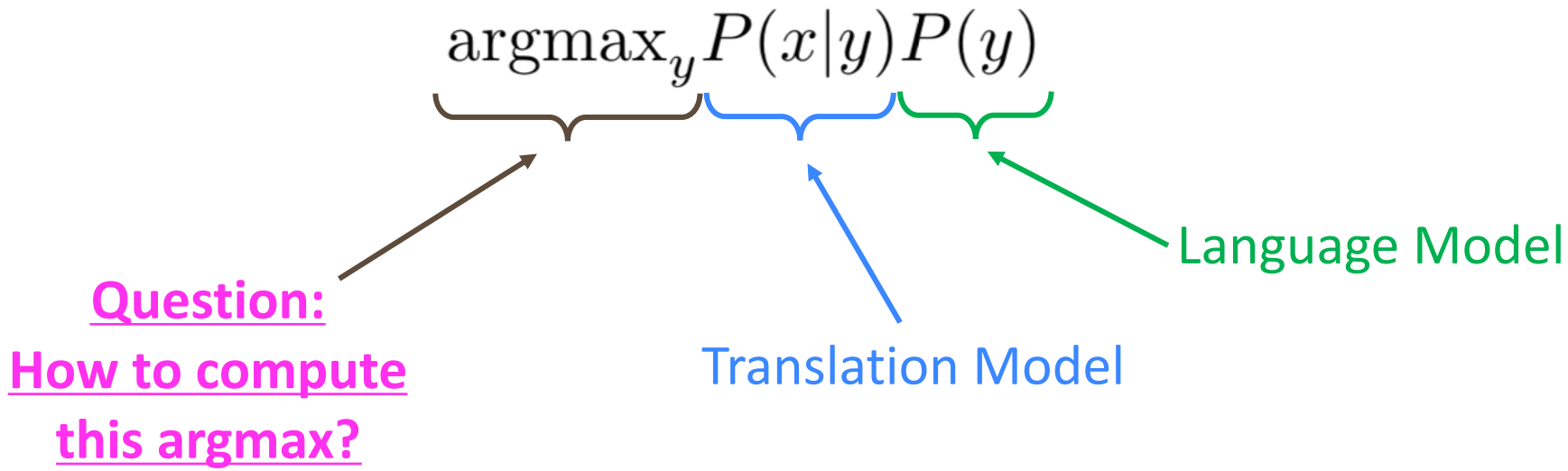
Since we only need $P(S \mid T)$ to score S, and don't need it to generate a grammatical S, it can be a relatively simple model.

$P(S \mid T)$ needs to be trained on a **parallel corpus**

The **language model** $P(T)$ is intended to capture the **fluency** of the translation.

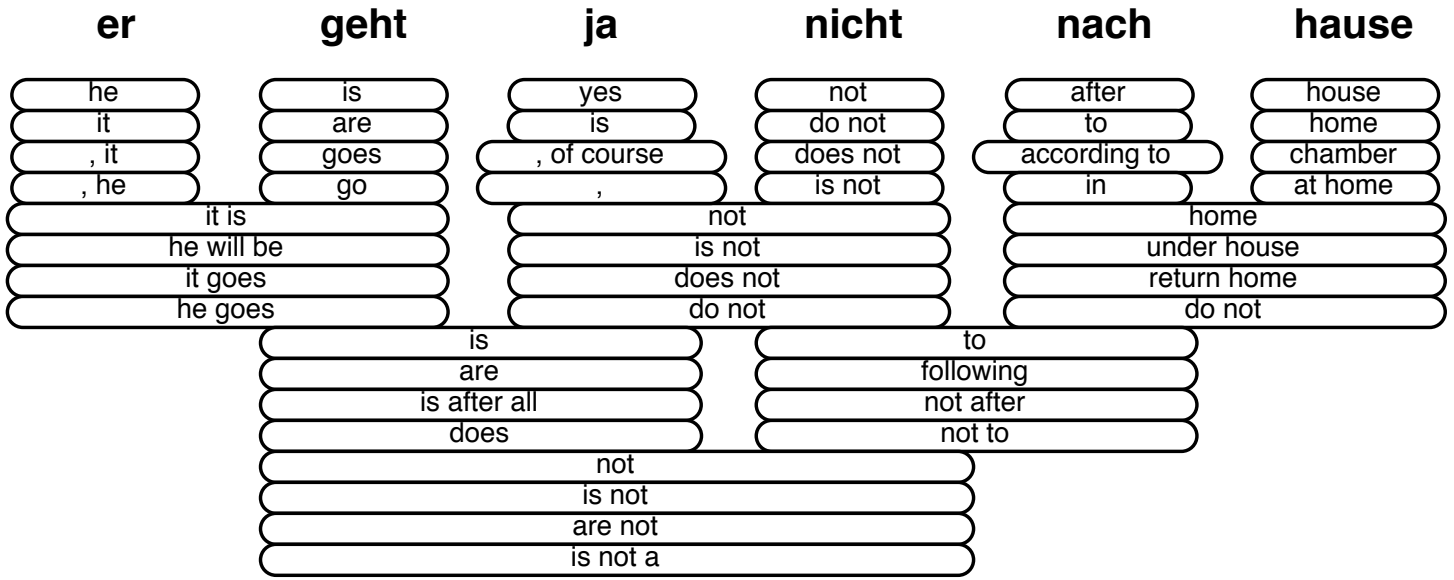
$P(T)$ can be trained on a (very large) **monolingual corpus**

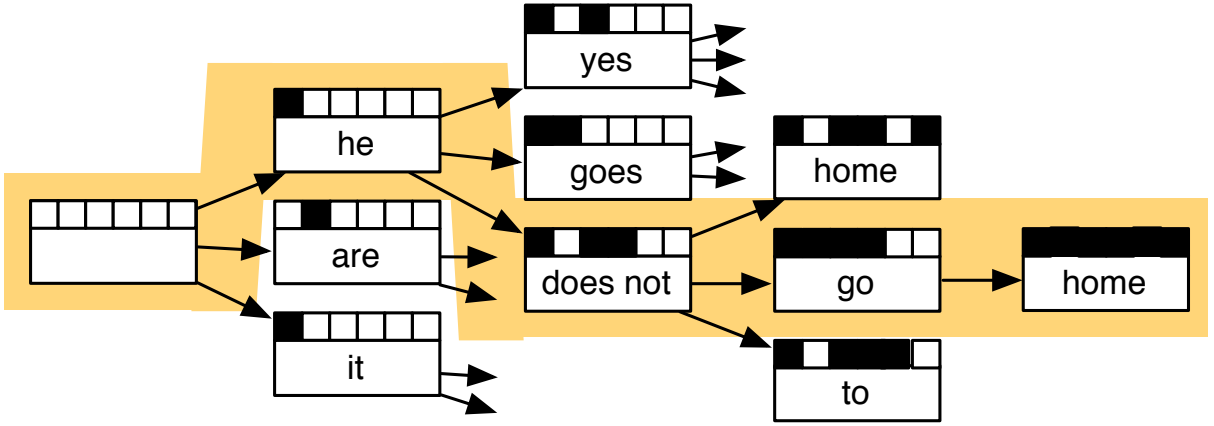
Decoding for SMT



- 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



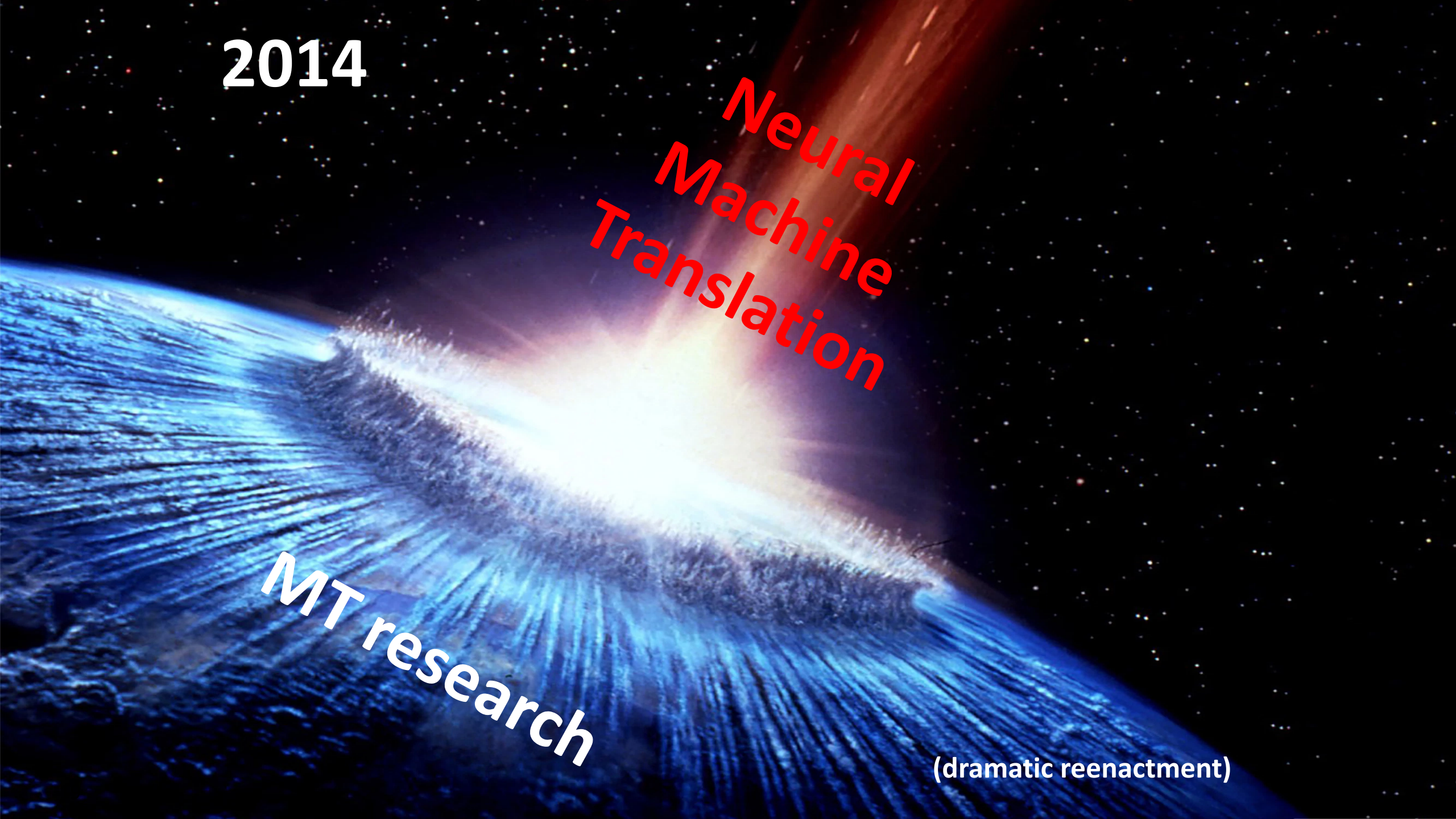


Source: ”Statistical Machine Translation", Chapter 6, Koehn, 2009.
https://www.cambridge.org/core/books/statistical-machine-translation/94EADF9F680558E13BE759997553CDE5

1990s-2010s: Statistical Machine Translation

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

Section 2: Neural Machine Translation


2014
Neural
Machine
Translation
MT research
(dramatic reenactment)

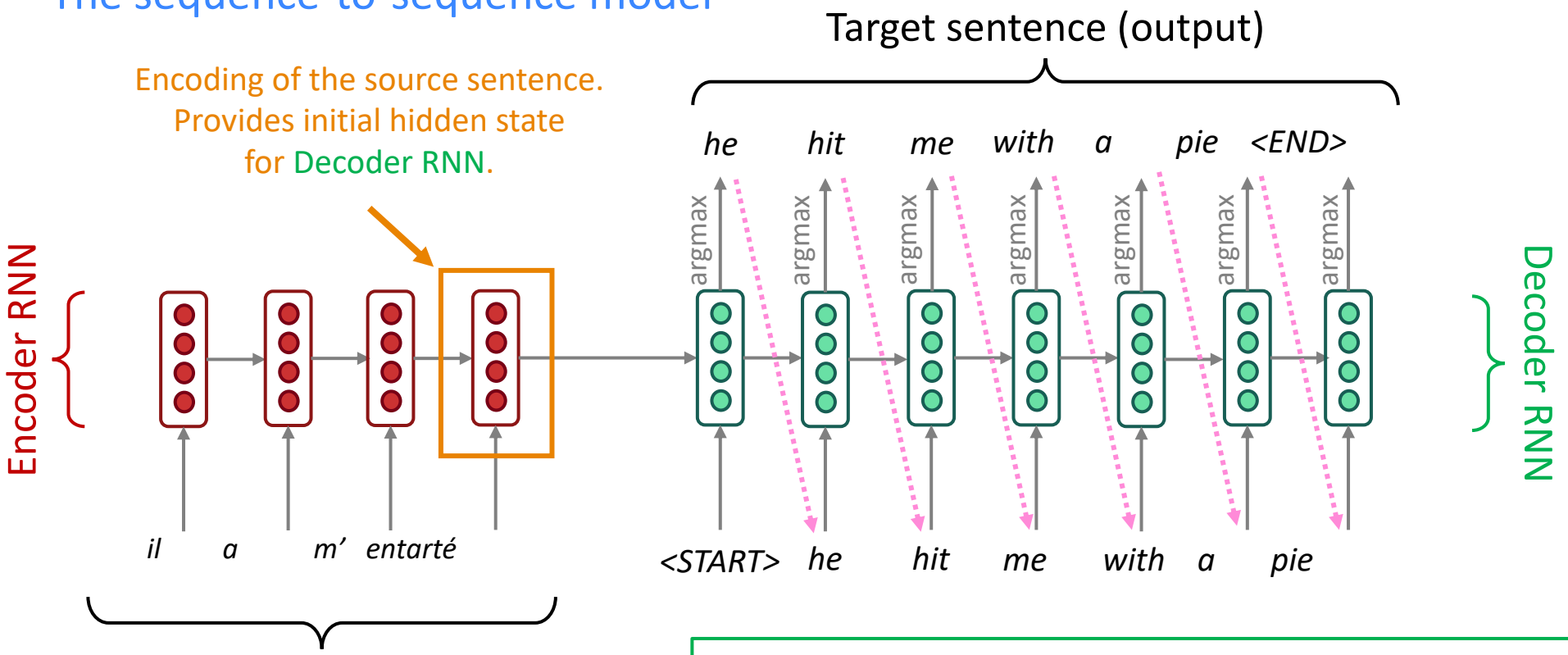
What is Neural Machine Translation?

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

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

Neural Machine Translation (NMT)

The sequence-to-sequence model



Encoder RNN produces an encoding of the source sentence.

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

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

Sequence-to-sequence is versatile!

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

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

Neural Machine Translation (NMT)

- The sequence-to-sequence model is an example of a **Conditional Language Model**
  - **Language Model** because the decoder is predicting the next word of the target sentence *y*
  - **Conditional** because its predictions are *also* conditioned on the source sentence *x*

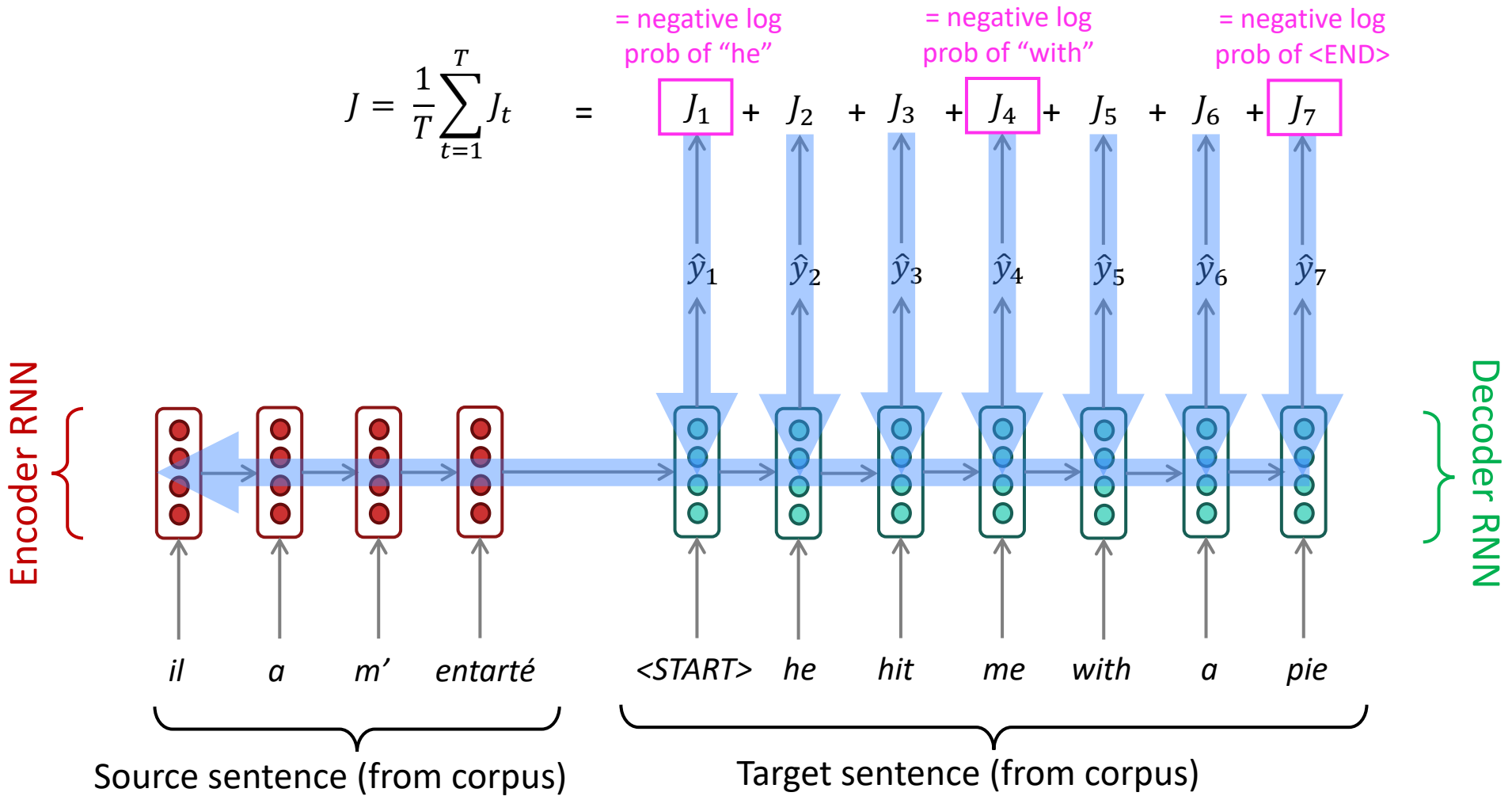
- NMT directly calculates $P(y|x)$:

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

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

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

Training a Neural Machine Translation system



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

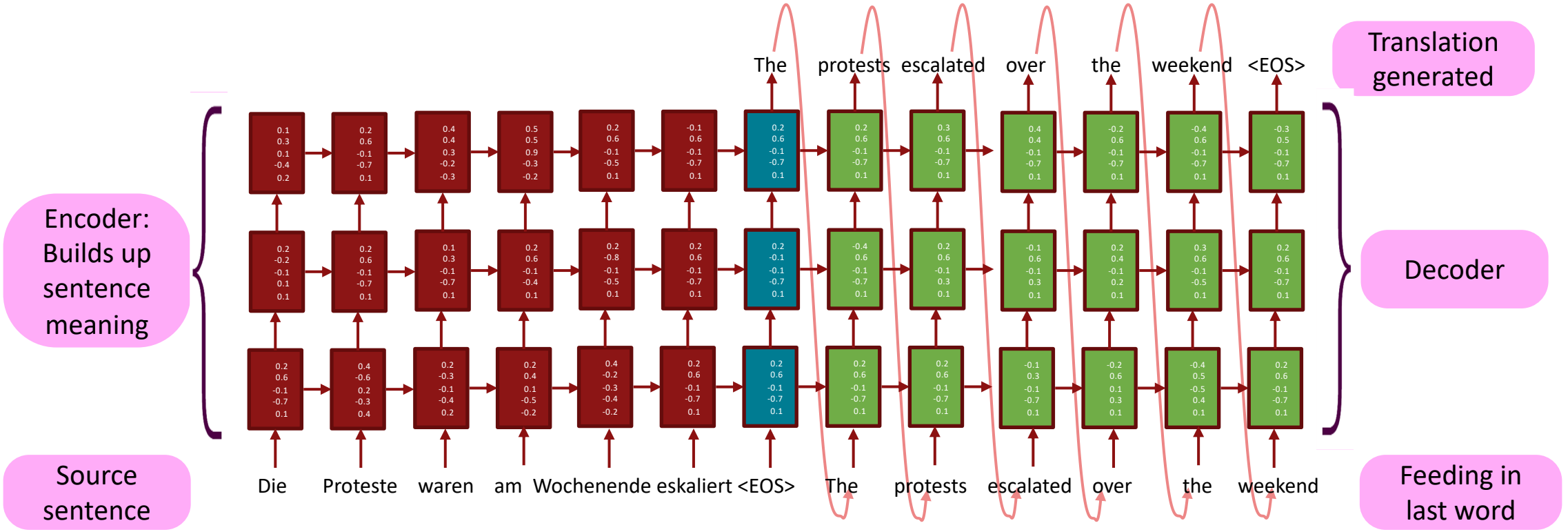
Multi-layer RNNs

- RNNs are already “deep” on one dimension (they unroll over many timesteps)
- We can also make them “deep” in another dimension by applying multiple RNNs – this is a multi-layer RNN.
- This allows the network to compute more complex representations
  - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.
- Multi-layer RNNs are also called *stacked RNNs*.

Multi-layer deep encoder-decoder machine translation net

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

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



Conditioning = Bottleneck

Multi-layer RNNs in practice

- High-performing RNNs are usually multi-layer (but aren't as deep as convolutional or feed-forward networks)

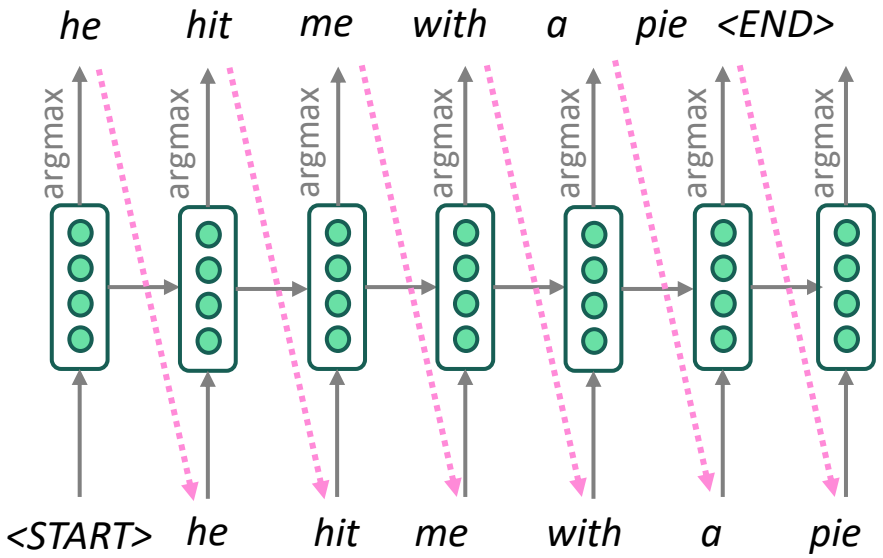
- For example: In a 2017 paper, Britz et al. find that for Neural Machine Translation, 2 to 4 layers is best for the encoder RNN, and 4 layers is best for the decoder RNN
  - Often 2 layers is a lot better than 1, and 3 might be a little better than 2
  - Usually, skip-connections/dense-connections are needed to train deeper RNNs (e.g., 8 layers)

- Transformer-based networks (e.g., BERT) are usually deeper, like 12 or 24 layers.
  - You will learn about Transformers later; they have a lot of skipping-like connections

"Massive Exploration of Neural Machine Translation Architecutres", Britz et al, 2017. https://arxiv.org/pdf/1703.03906.pdf

Greedy decoding

- We saw how to generate (or “decode”) the target sentence by taking argmax on each step of the decoder



- This is greedy decoding (take most probable word on each step)
- **Problems with this method?**

Problems with greedy decoding

- Greedy decoding has no way to undo decisions!
  - Input: *il a m'entarté* *(he hit me with a pie)*
  - → *he* ____
  - → *he hit* ____
  - → *he hit a* ____ *(whoops! no going back now…)*

- How to fix this?

Exhaustive search decoding

- Ideally, we want to find a (length *T*) translation *y* that maximizes

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

- 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)

- 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

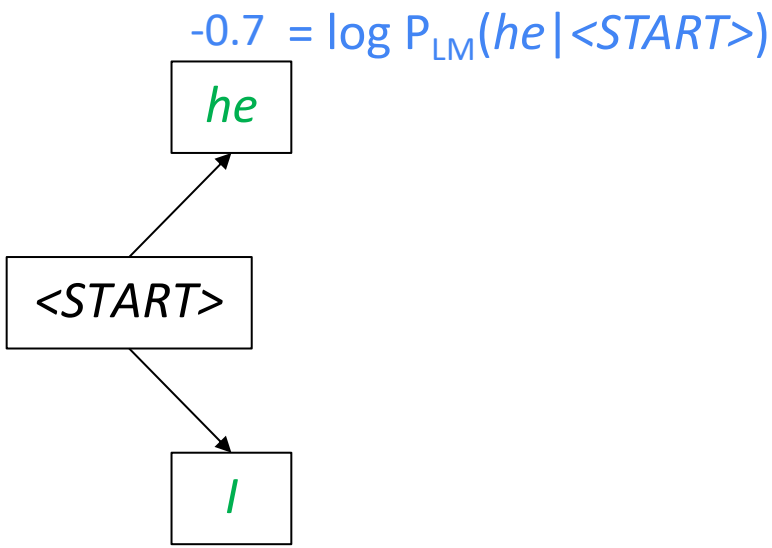
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

<START>

Calculate prob dist of next word

Beam search decoding: example

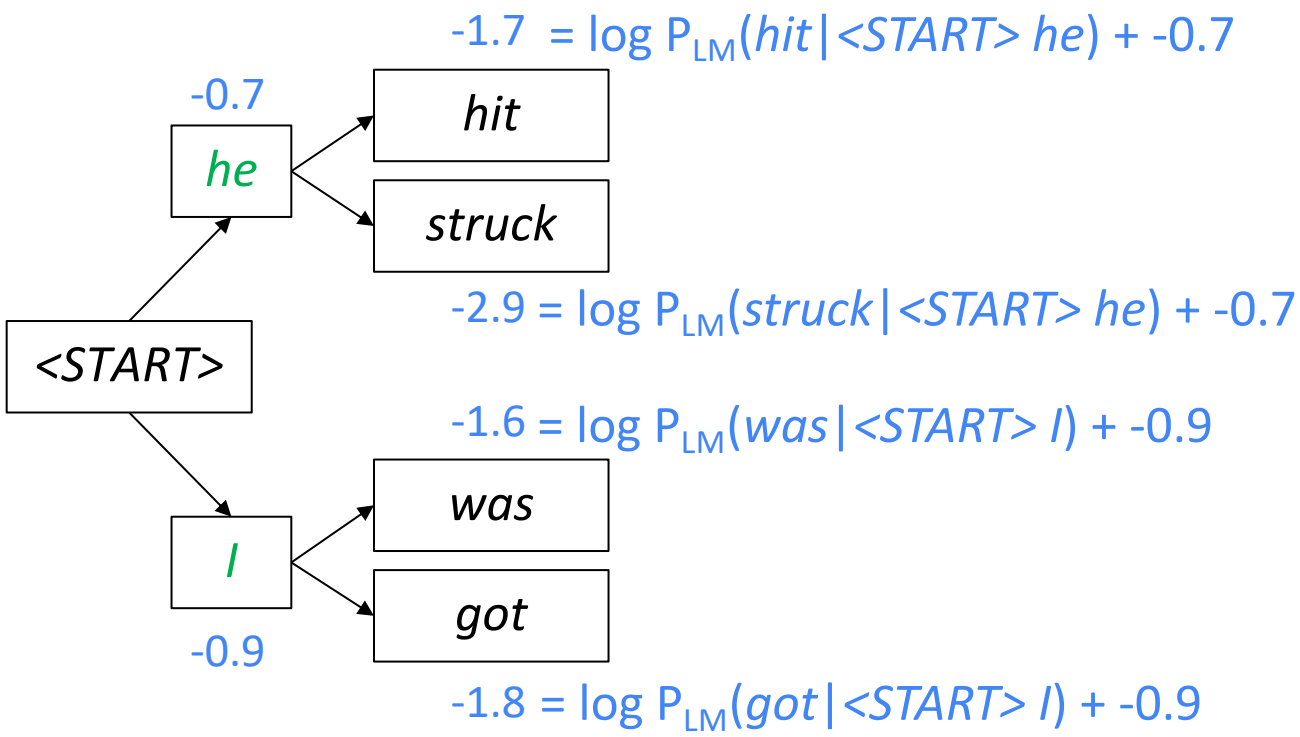
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Take top *k* words and compute scores

Beam search decoding: example

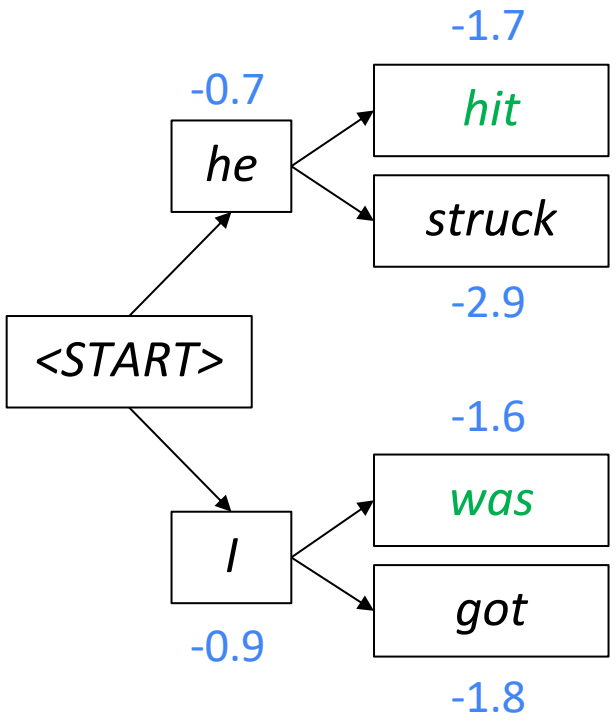
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

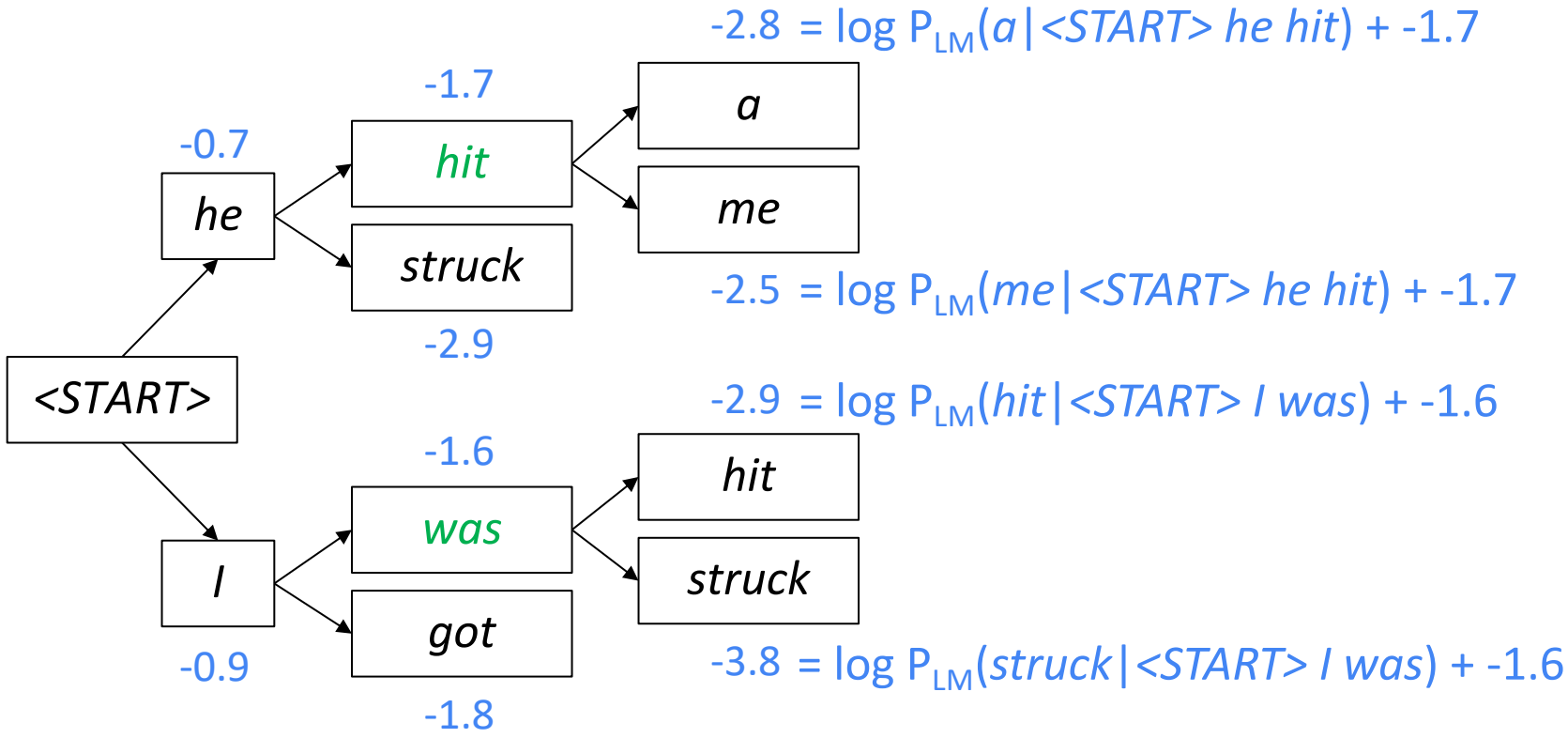
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

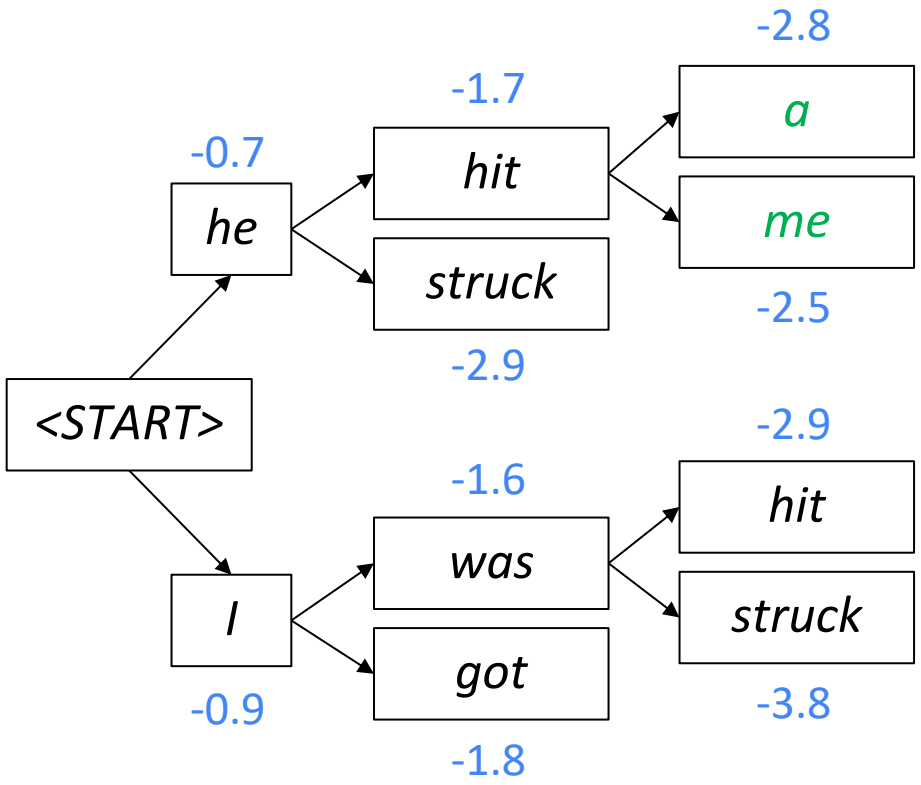
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

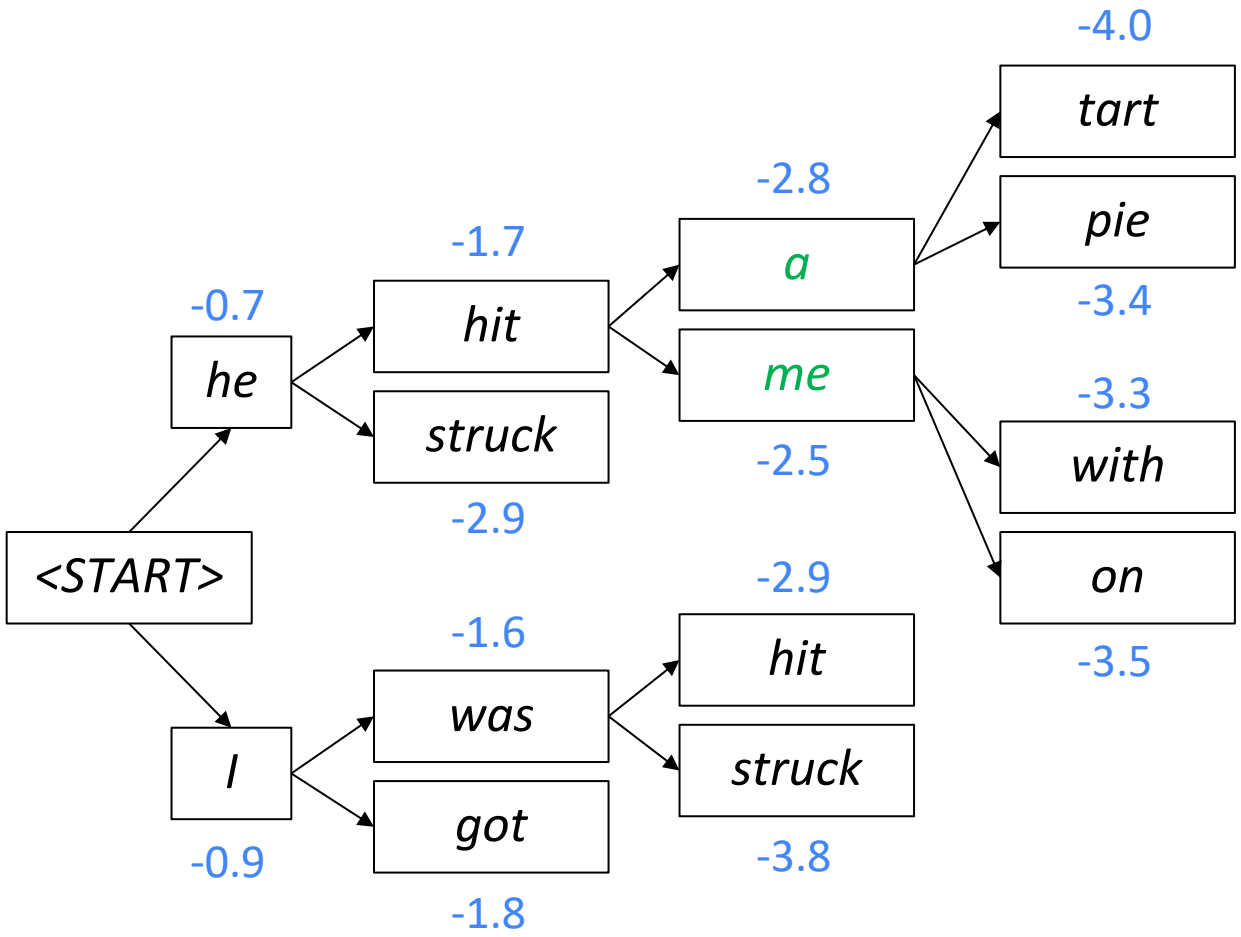
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

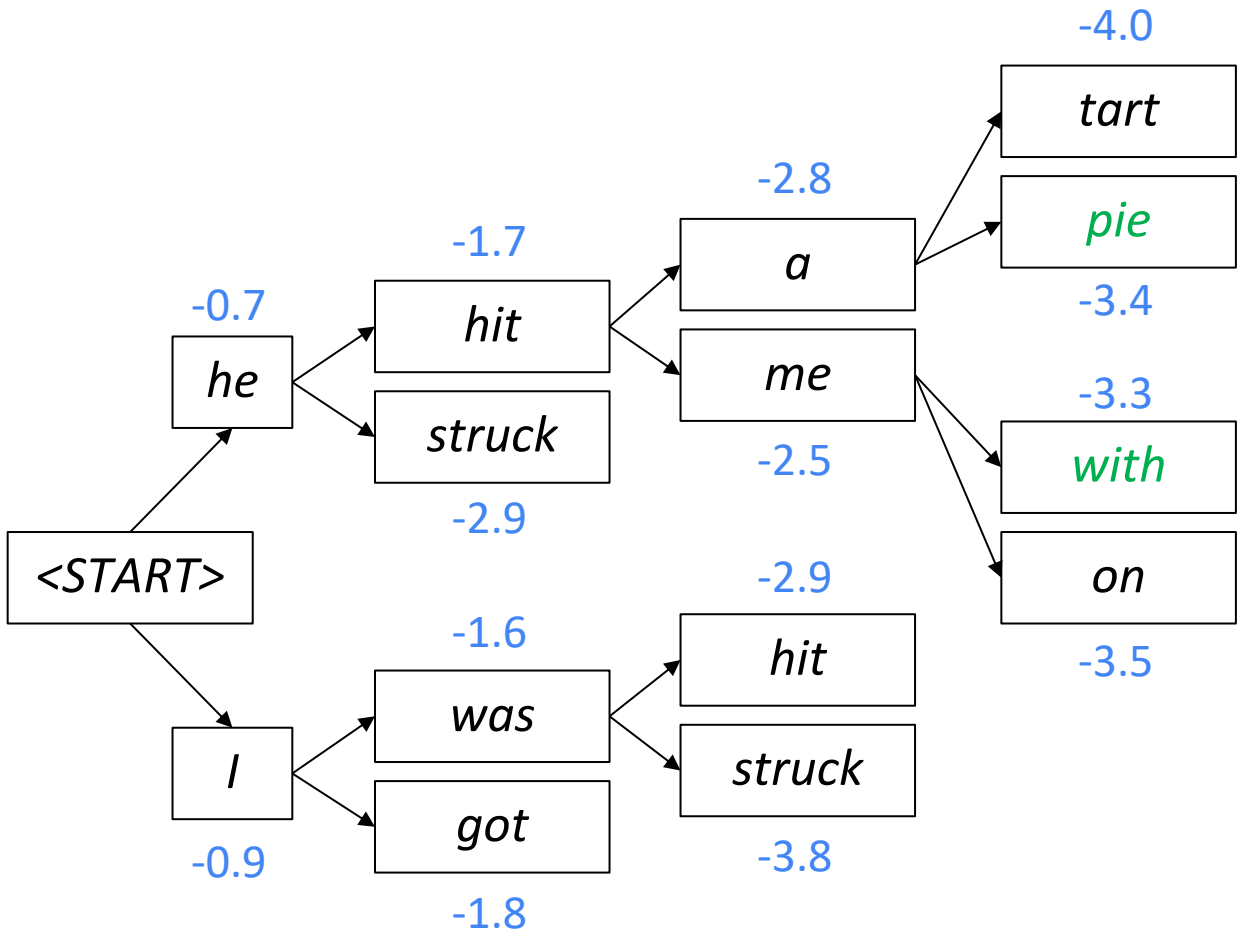
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

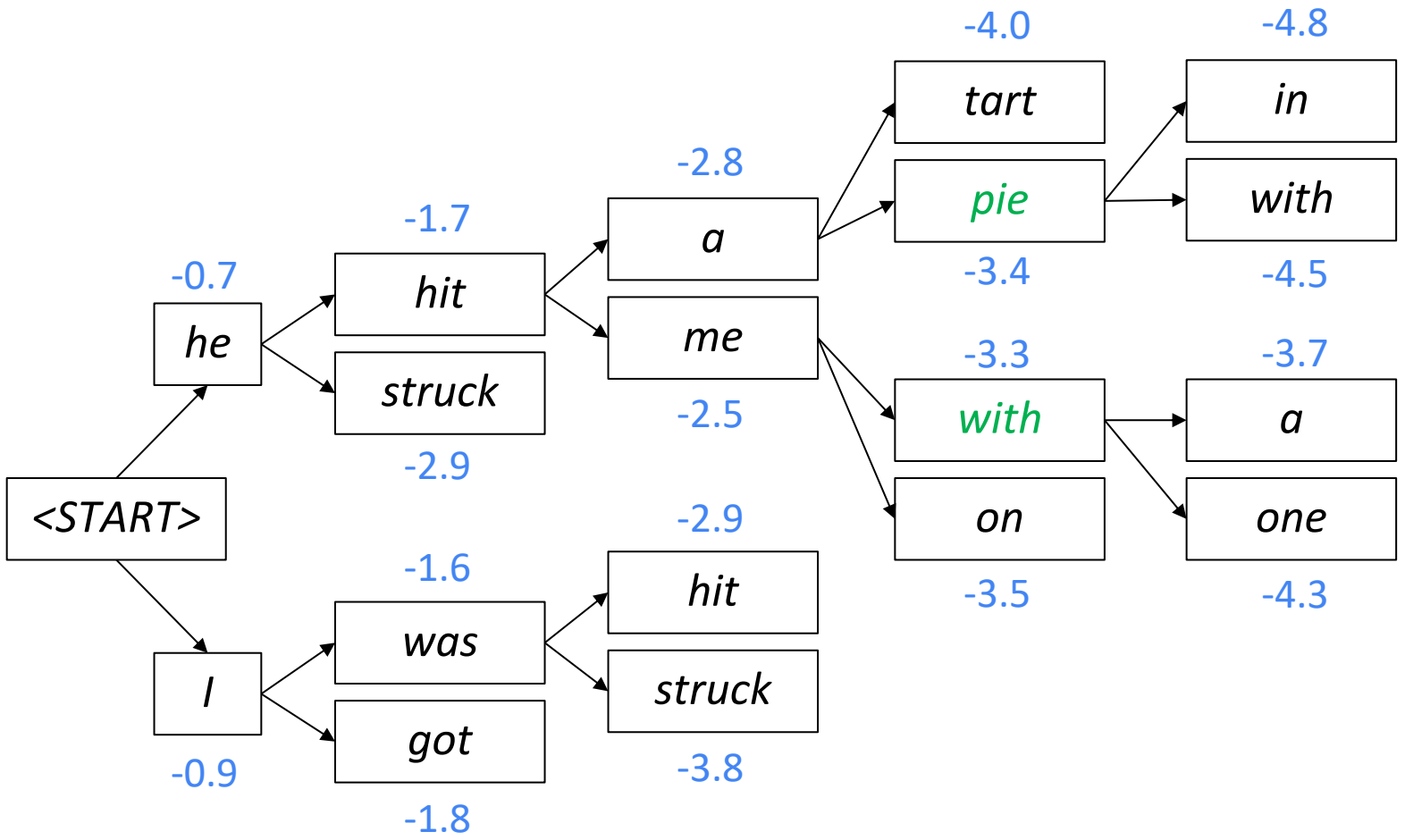
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

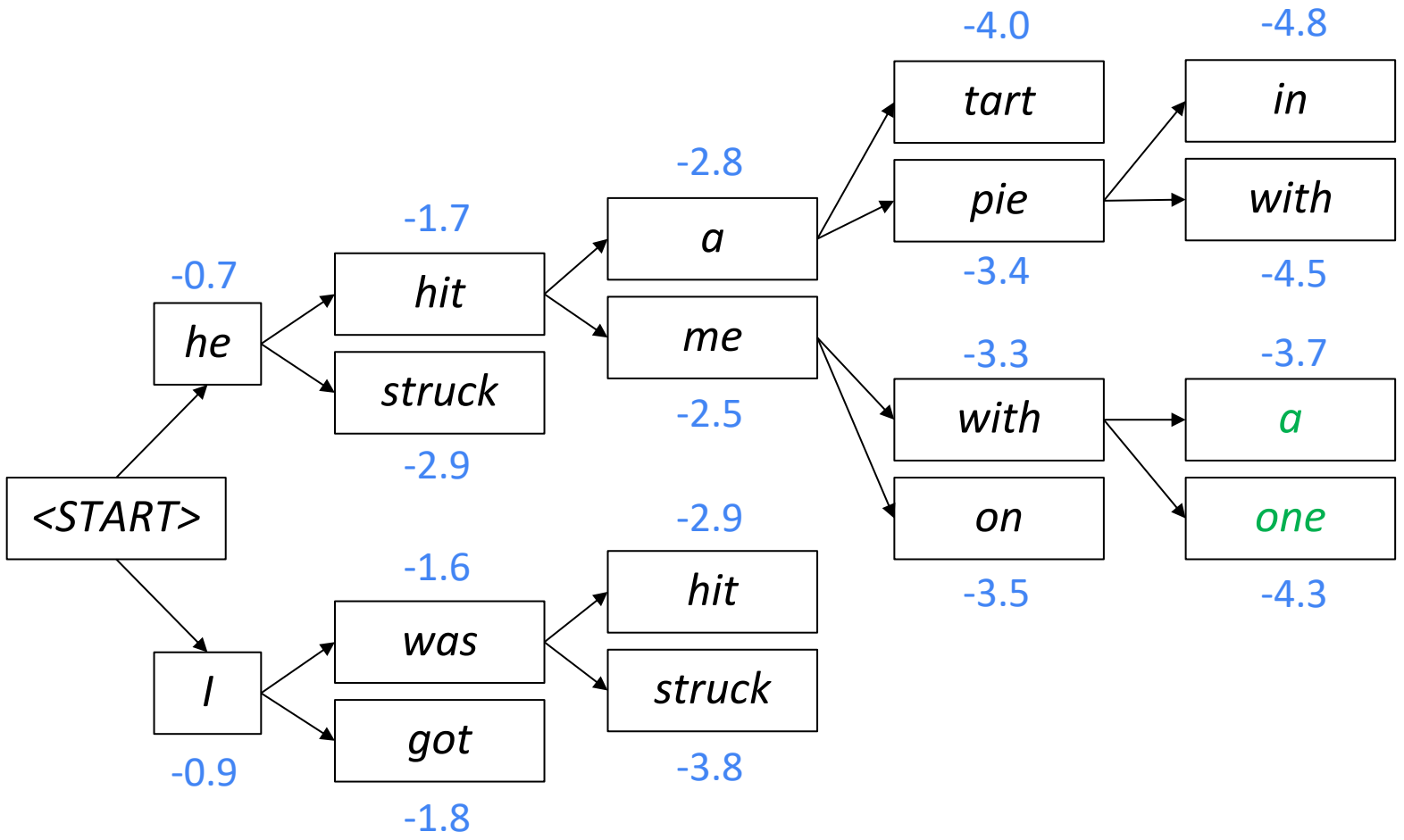
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

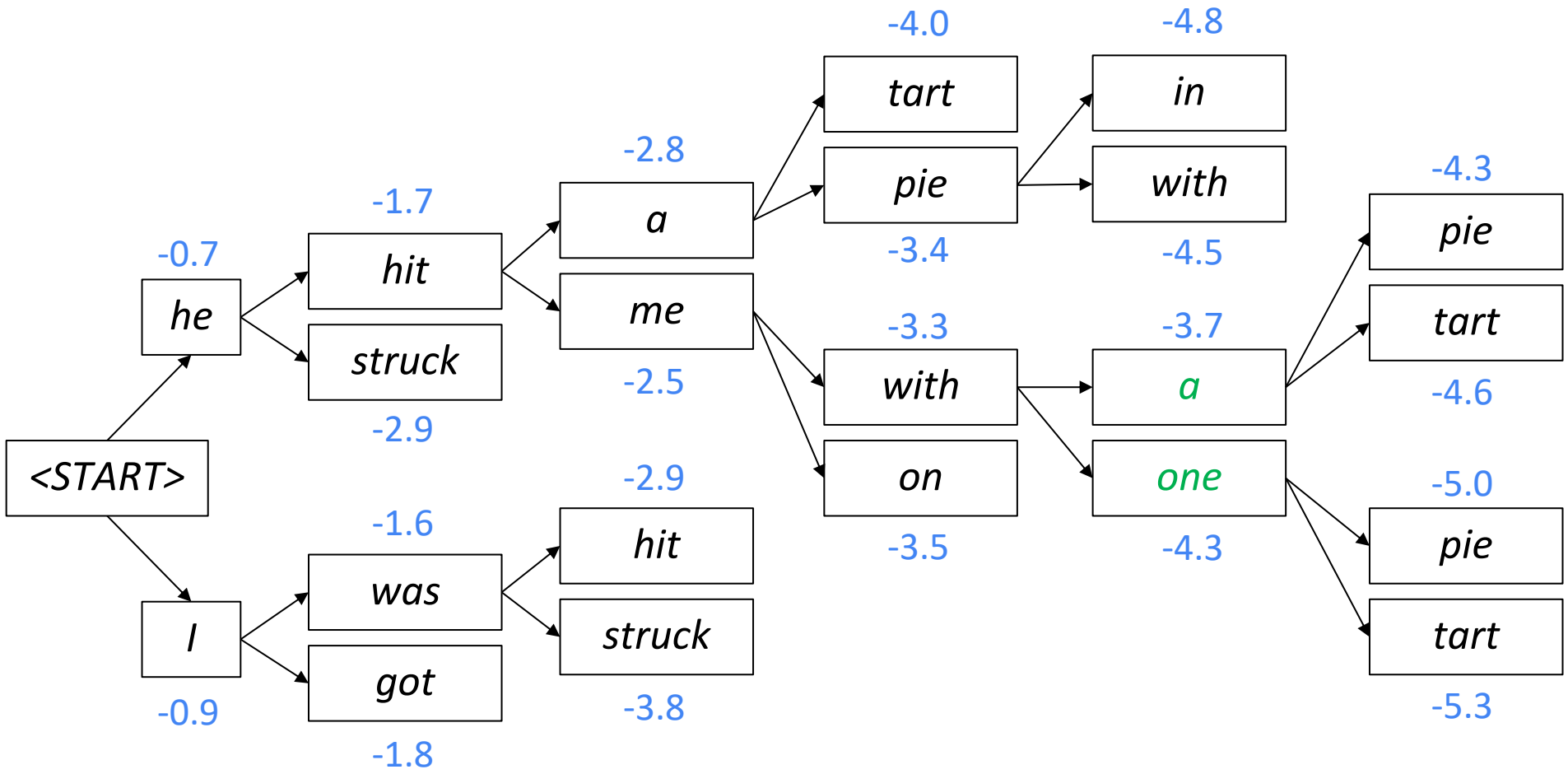
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

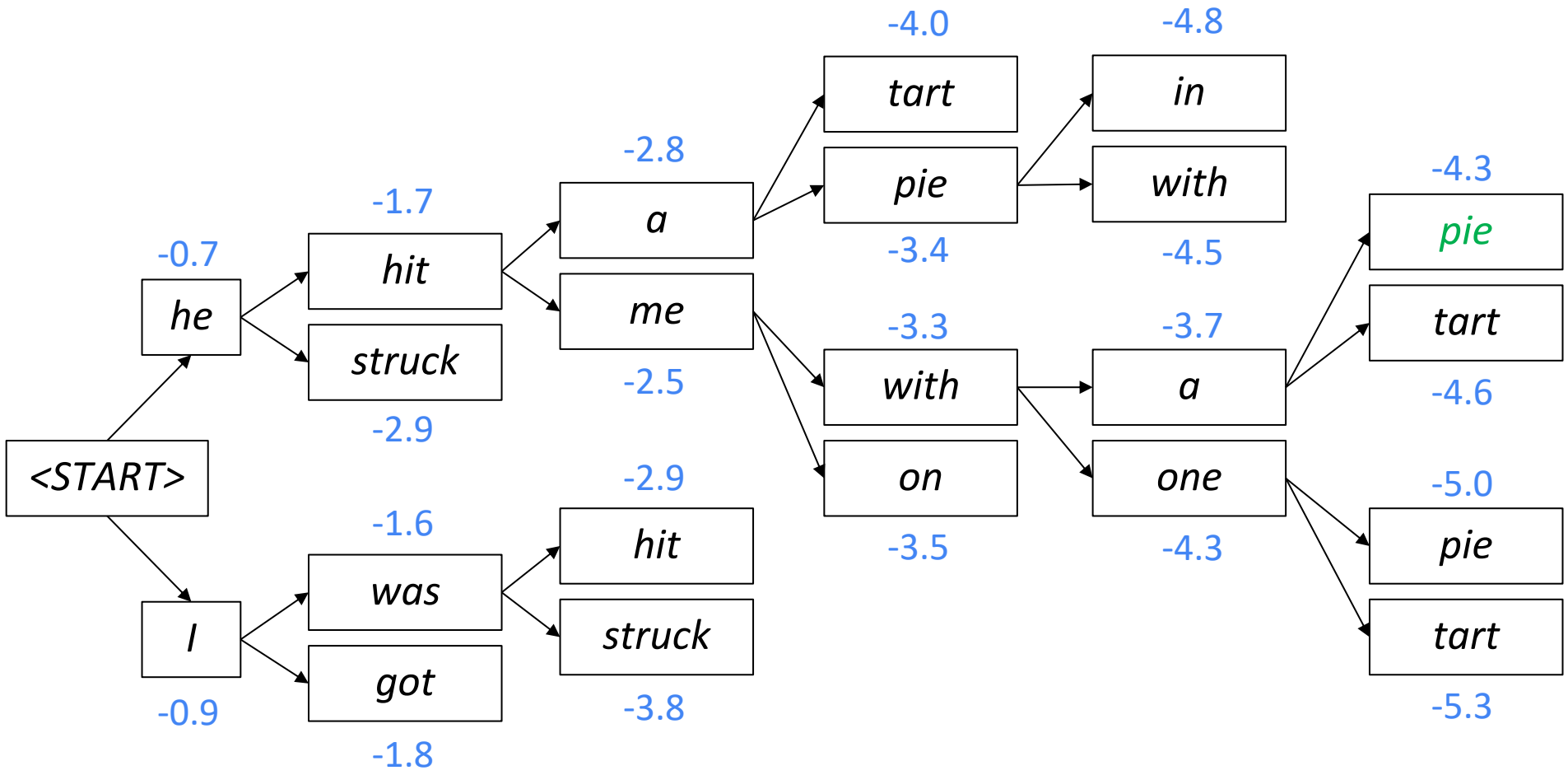
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

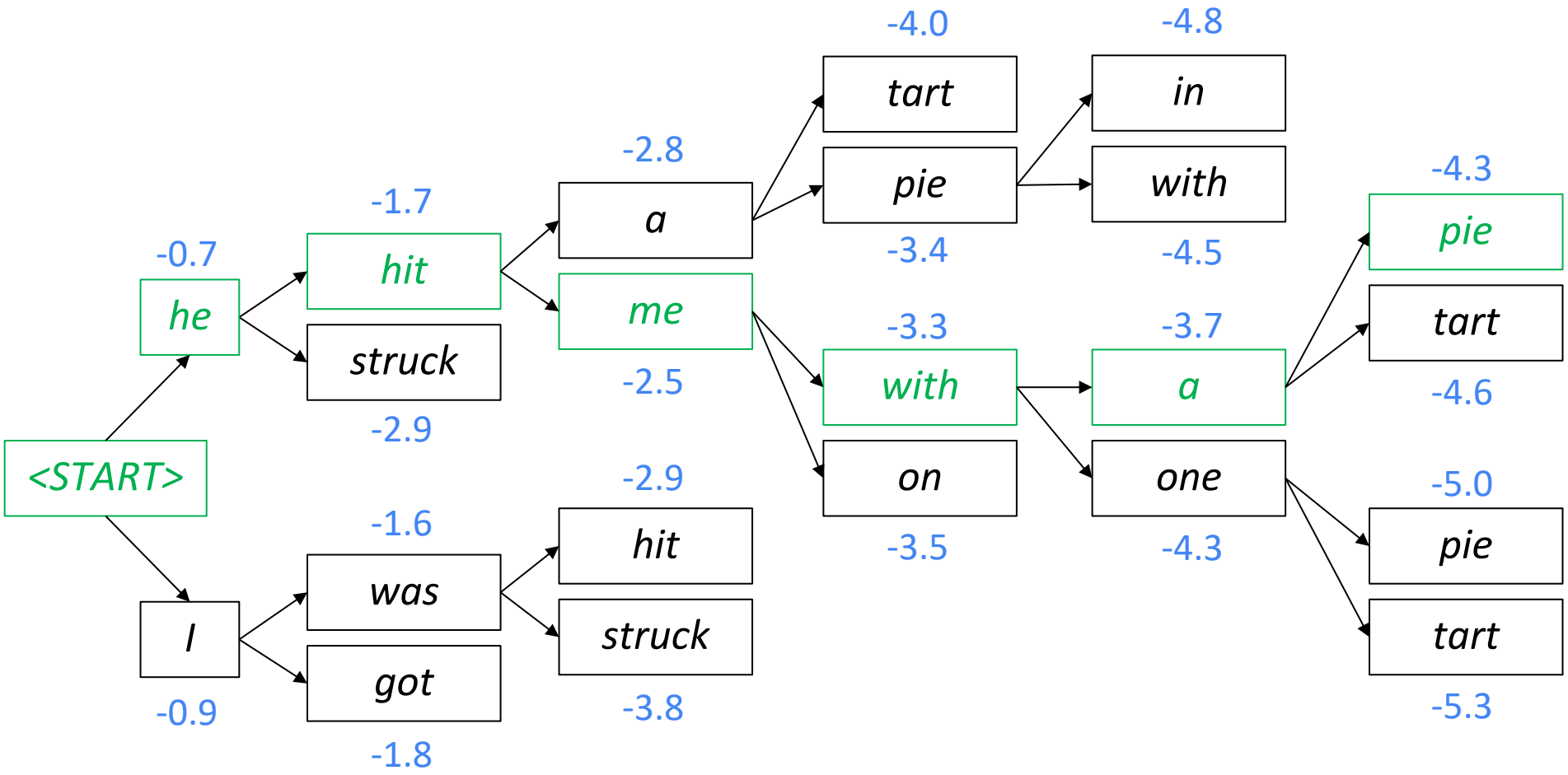
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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 (**Bi**lingual **E**valuation **U**nderstudy)

You'll see BLEU in detail in Assignment 4!

- BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:
  - *n*-gram precision (usually for 1, 2, 3 and 4-grams)
  - Plus a penalty for too-short system translations

- BLEU is useful but imperfect
  - There are many valid ways to translate a sentence
  - So a good translation can get a poor BLEU score because it has low *n*-gram overlap with the human translation ☹

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

Automatic evaluation: BLEU

Evaluate candidate translations against several reference translations.

C1: It is a guide to action which ensures that the military always obeys the commands of the party.
C2: It is to insure the troops forever hearing the activity guidebook that party direct

R1: It is a guide to action that ensures that the military will forever heed Party commands.
R2: It is the guiding principle which guarantees the military forces always being under the command of the Party.
R3: It is the practical guide for the army always to heed the directions of the party.

The **BLEU score** is based on **N-gram precision**:

How many n-grams in the candidate translation occur also in one of the reference translation?

BLEU details

For $n \in \{1,\ldots,4\}$, compute the (modified) **precision of all *n*-grams:**

$$Prec_n = \frac{\sum_{c \in C} \sum_{n\text{-gram} \in c} \text{MaxFreq}_{\text{ref}}(n\text{-gram})}{\sum_{c \in C} \sum_{\text{-gram} \in c} \text{Freq}_{\text{c}}(n\text{-gram})}$$

$\text{MaxFreq}_{\text{ref}}$(*'the party'*) = max. count of *'the party'* in **one** reference translation.

Freq_{c}(*'the party'*) = count of *'the party'* in candidate translation c.

Penalize short candidate translations by a **brevity penalty** BP

c = length (number of words) of the whole candidate translation corpus

r = Pick for each candidate the reference translation that is closest in length; sum up these lengths.

Brevity penalty $\text{BP} = \exp(1-\text{c/r})$ for $\text{c} \leq \text{r}$; $\text{BP} = 1$ for $\text{c} > \text{r}$

(BP ranges from e for c=0 to 1 for c=r)

BLEU score

The BLEU score is the geometric mean of the modified n-gram precision (for n=1..4), weighted by a brevity penalty BP:

$$\mathbf{BLEU} = BP \times \exp\left(\frac{1}{N}\sum_{n=1}^{N} \log Prec_n\right)$$

Geometric mean for $a_1, \ldots, a_N > 0$ = N-th root of $\prod_{n=1}^{N} a_n$

$$\sqrt[N]{\prod_{n=1}^{N} a_n} = \left(\prod_{n=1}^{N} a_n\right)^{\frac{1}{N}} = \exp\left(\frac{1}{N}\sum_{n=1}^{N} \log a_n\right)$$

BLEU details

Compute the (modified) precision of all *n*-grams (for $n = 1 \ldots 4$)

Sum over the translations c of any sentence in the test corpus C…

…sum over all n-grams occurring in c..

… the **maximum frequency** of that n-gram in any ***one*** of c's **reference** translations.

For n = 1..4:

$$Prec_n = \frac{\sum_{c \in C} \sum_{n\text{-gram} \in c} \text{MaxFreq}_{\text{ref}}(n\text{-gram})}{\sum_{c \in C} \sum_{\text{-gram} \in c} \text{Freq}_{\text{c}}(n\text{-gram})}$$

Sum over the translations c of any sentence in the test corpus C…

…sum over all n-grams occurring in c..

… the **frequency** of that n-gram in c.

Penalize short candidate translations by a brevity penalty BP

BP = exp(1–c/r) for $c \leq r$; BP = 1 for $c > r$

(BP ranges from 1 for c=r to e for c=0)

c = Total length (number of words) of the whole candidate translation corpus

r = Total length of all reference translations closest in length to candidates

Human evaluation

We want to know…

whether the translation is **"good" English**, and…

… whether it is an **accurate translation** of the original.

— Ask human raters to judge the **fluency** and the **adequacy** of the translation (e.g. on a scale of 1 to 5)

— Correlated with fluency is accuracy on **cloze task**:

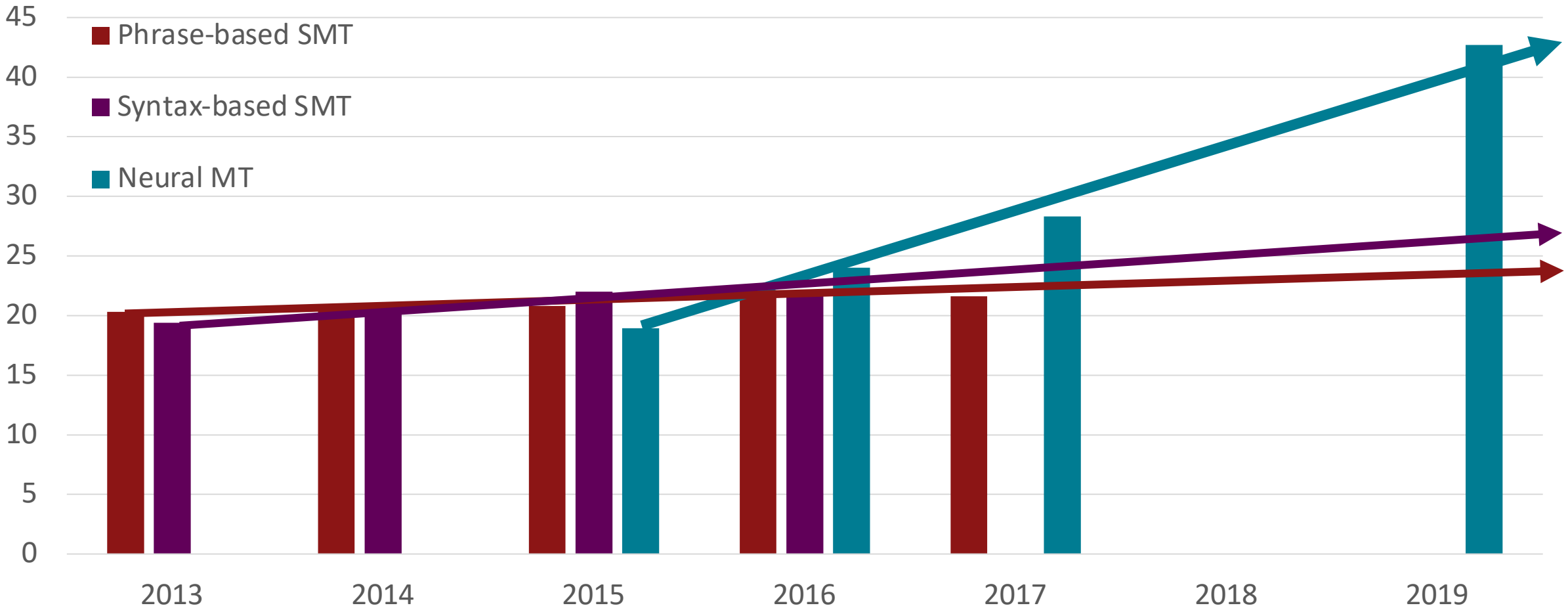
Give rater the sentence with one word replaced by blank.
Ask rater to guess the missing word in the blank.

— Similar to adequacy is **informativeness**

Can you use the translation to perform some task
(e.g. answer multiple-choice questions about the text)

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

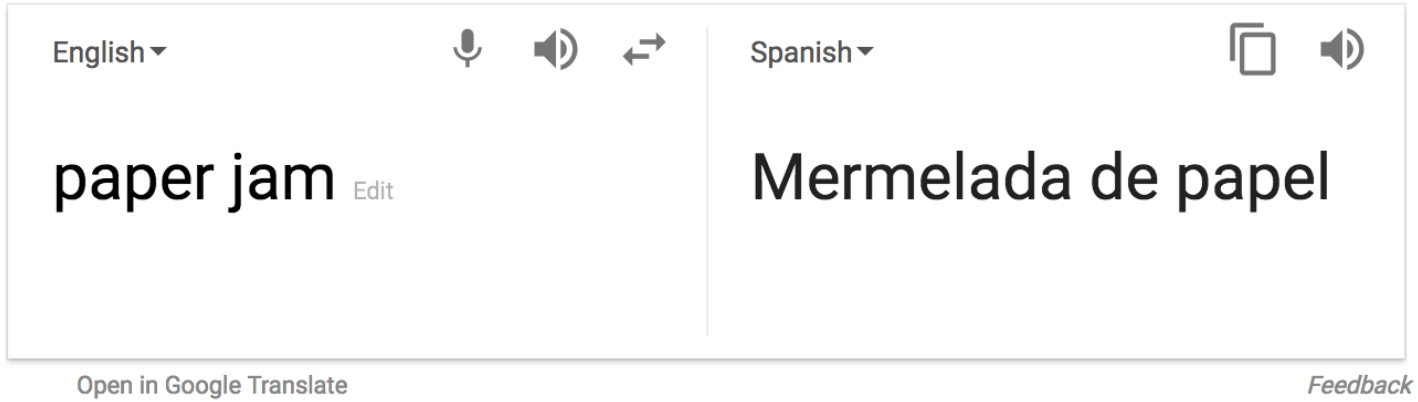
So, is Machine Translation solved?

- **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?

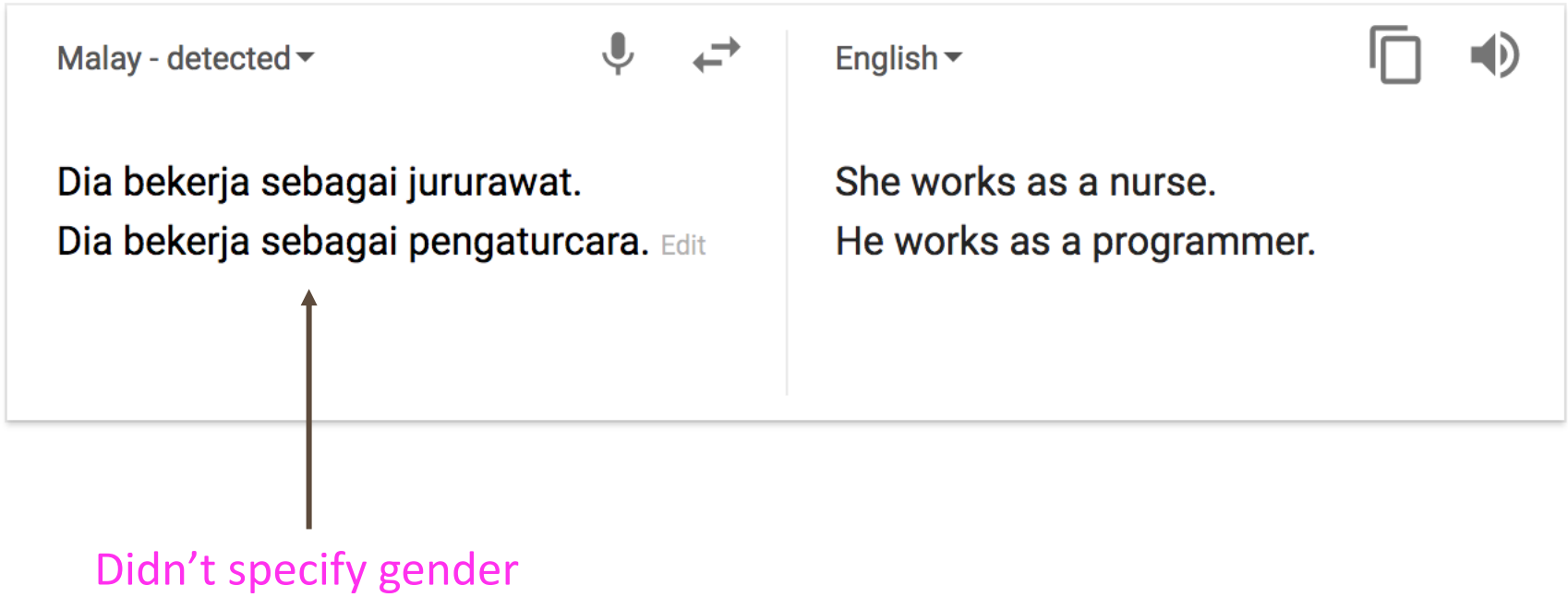
- **Nope!**
- Using common sense is still hard



?

So is Machine Translation solved?

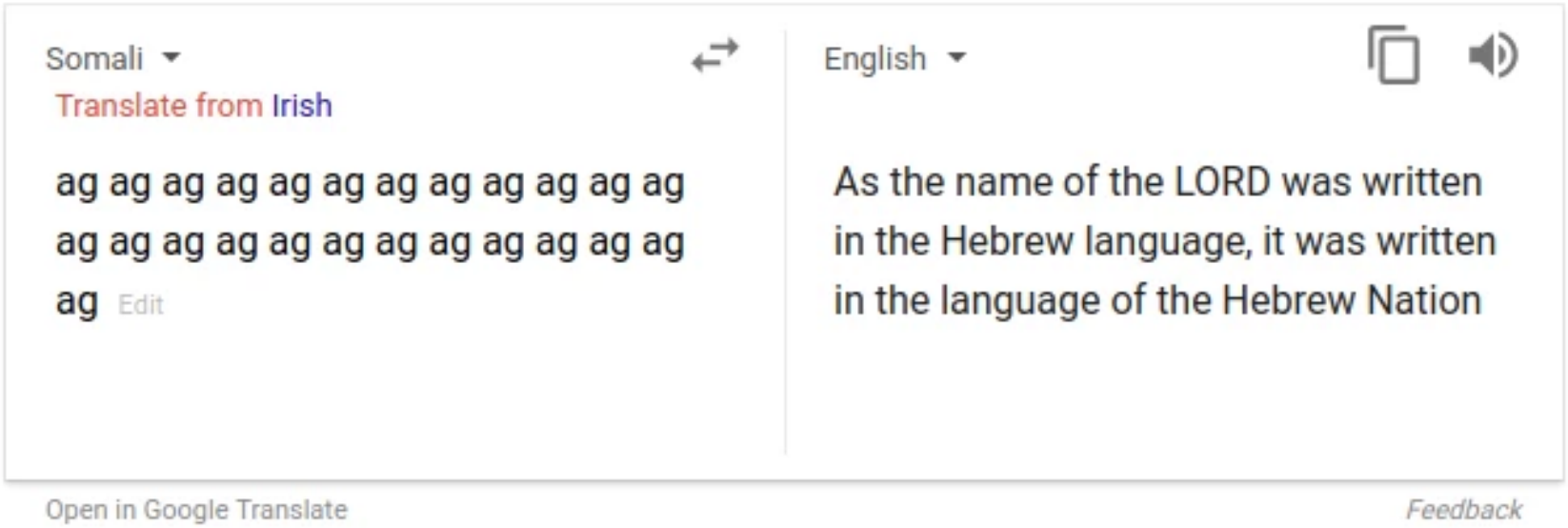
- **Nope!**
- NMT picks up biases in training data



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 I think this problem has been fixed in Google Translate by 2021?)



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

NMT research continues

NMT is a **flagship task** for NLP Deep Learning

- NMT research has pioneered many of the recent innovations of NLP Deep Learning

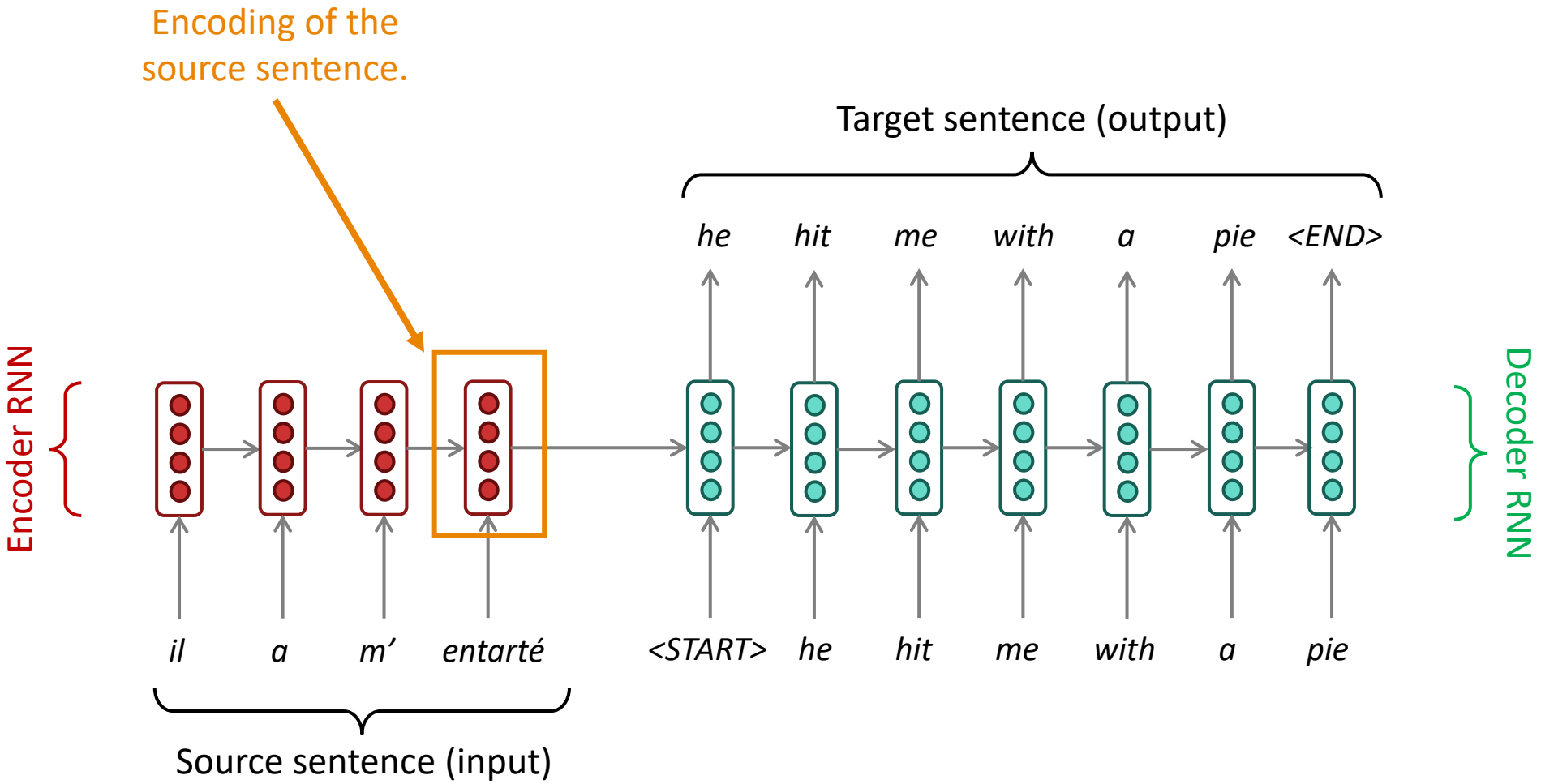
- In **2021**: NMT research continues to thrive
  - Researchers have found *many, many* improvements to the "vanilla" seq2seq NMT system we've just presented

  - But we'll present in a minute one improvement so integral that it is the new vanilla…

ATTENTION

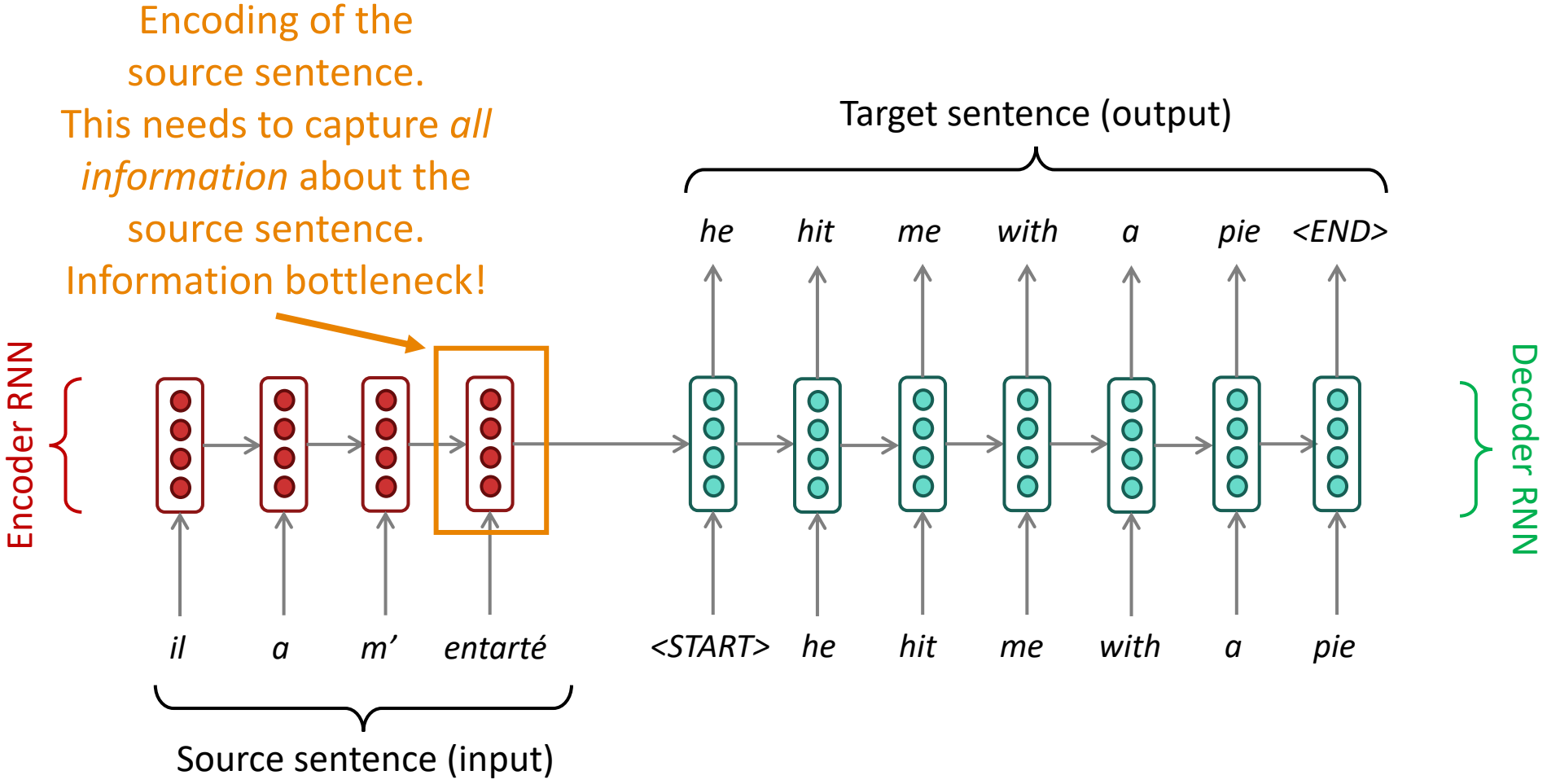
Section 3: Attention

Sequence-to-sequence: the bottleneck problem



Problems with this architecture?

Sequence-to-sequence: the bottleneck problem



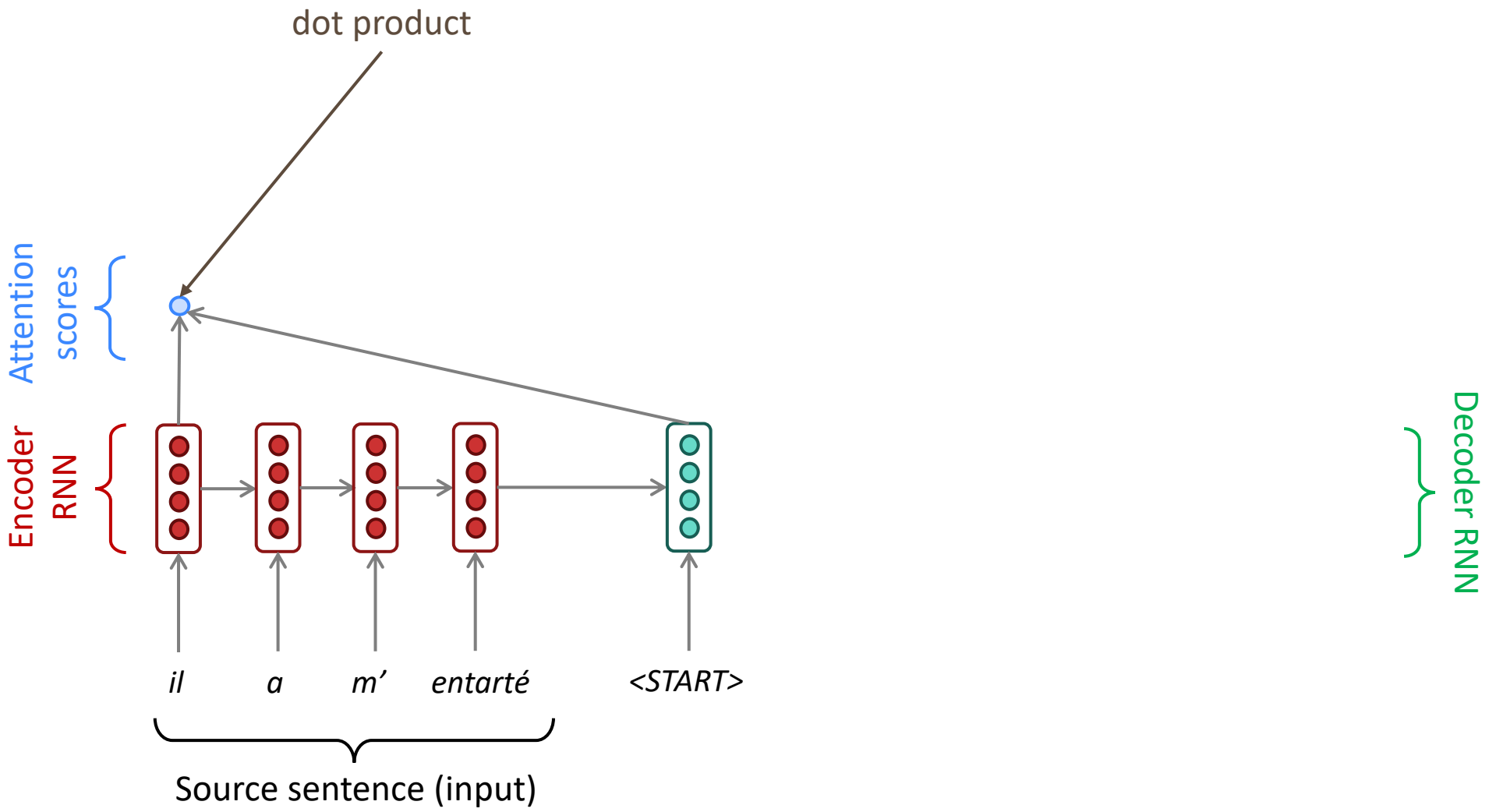
Attention

- **Attention** provides a solution to the bottleneck problem.

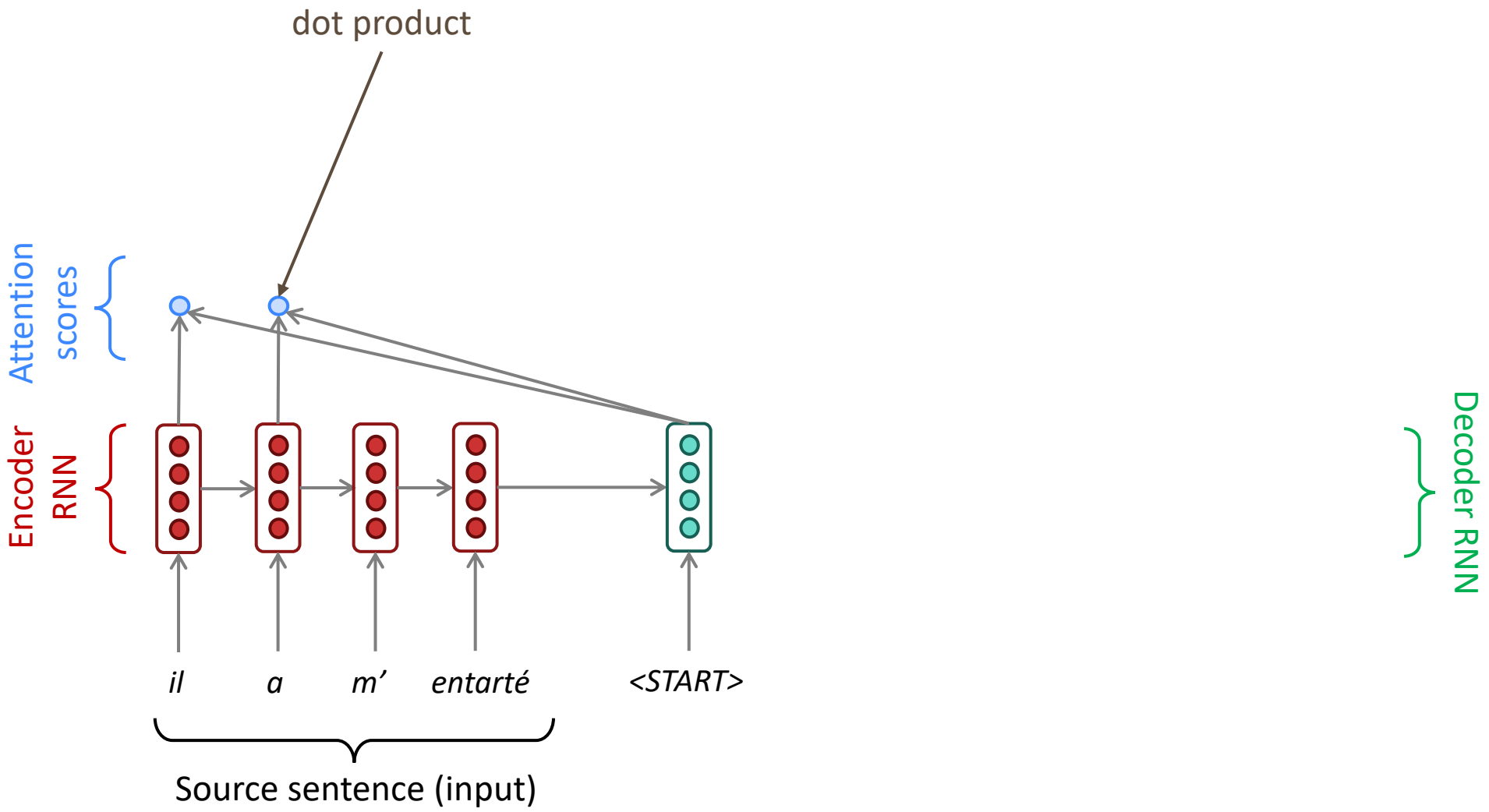
- Core idea: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

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

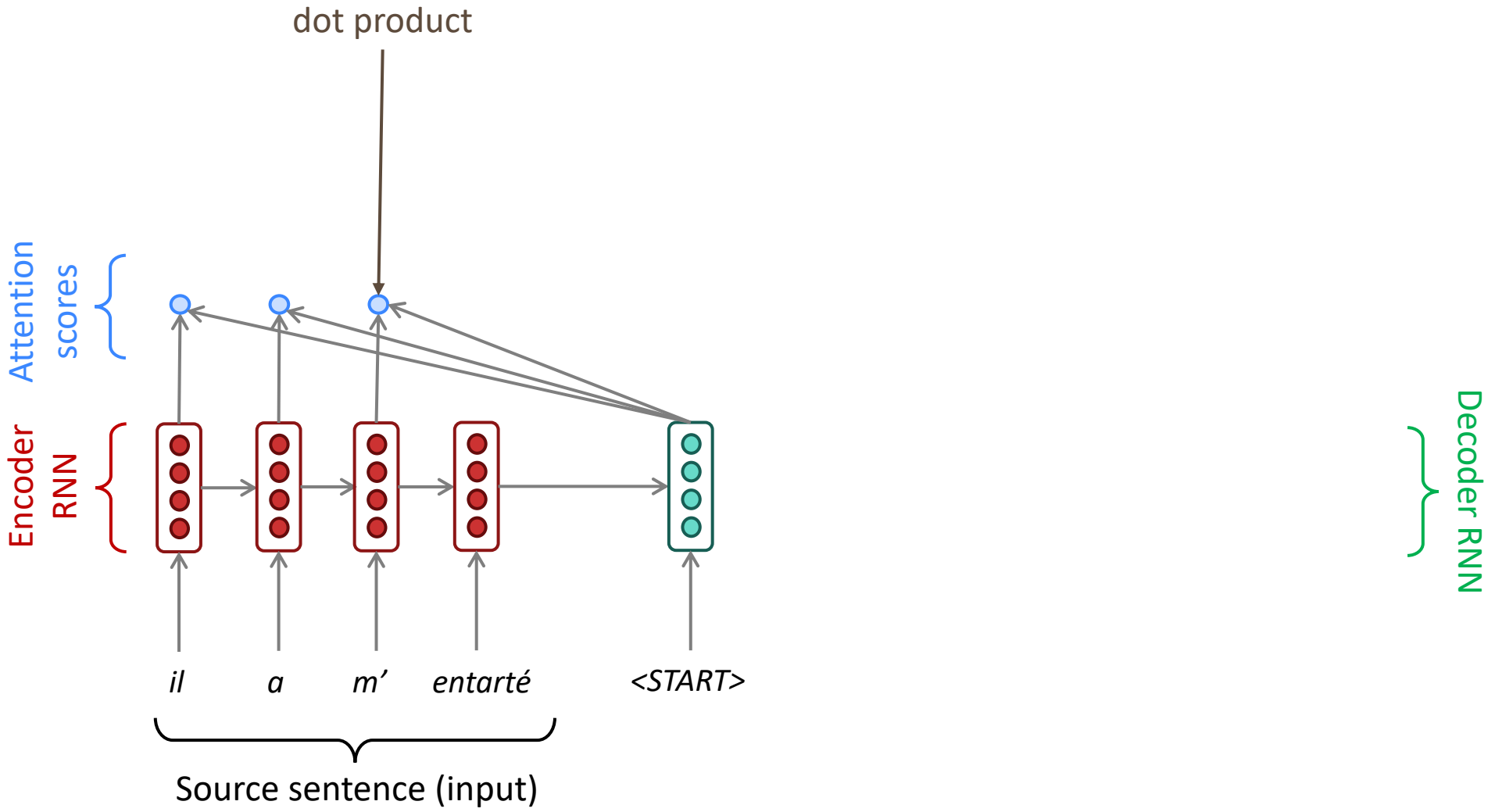
Sequence-to-sequence with attention



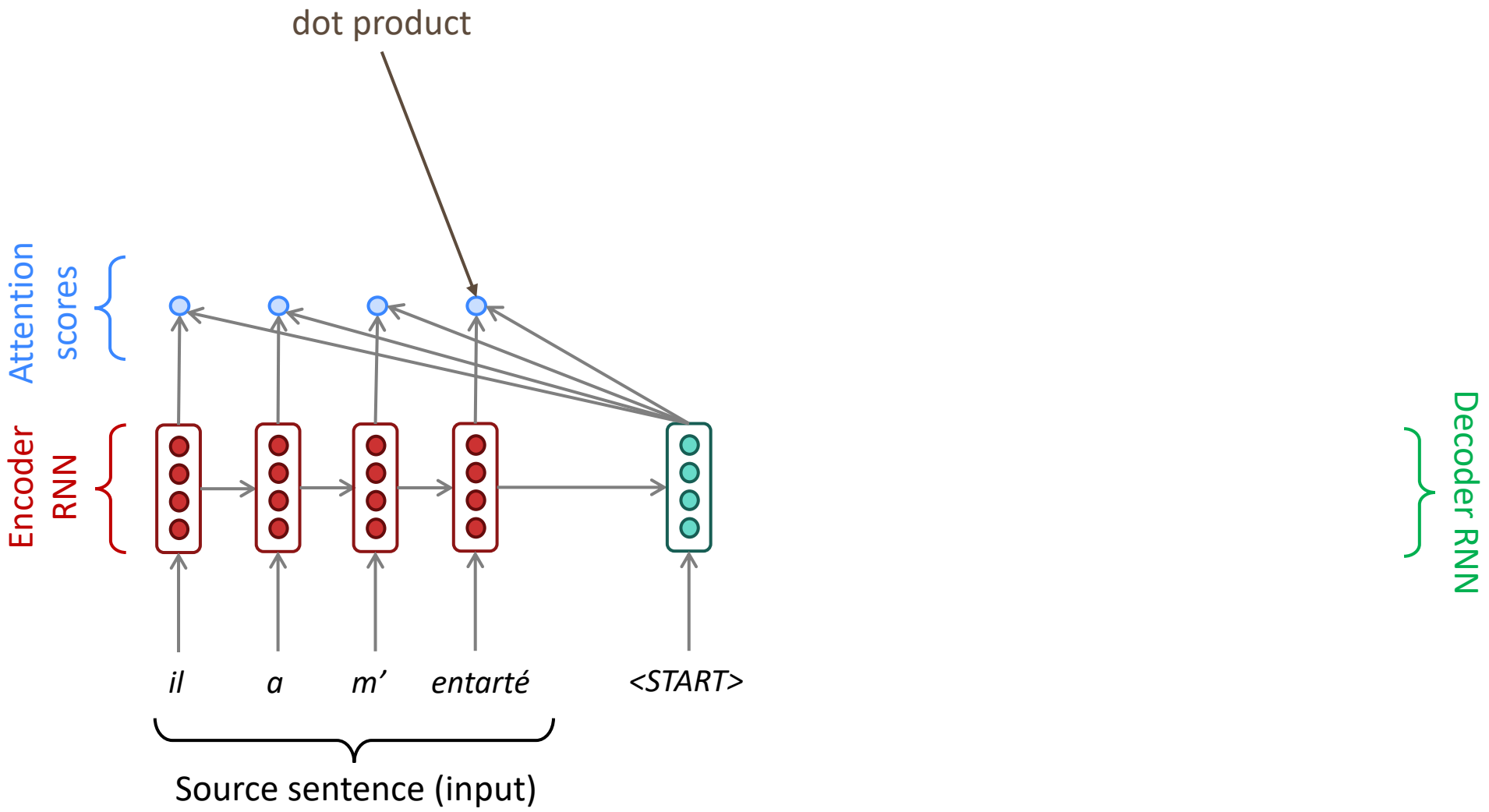
Sequence-to-sequence with attention



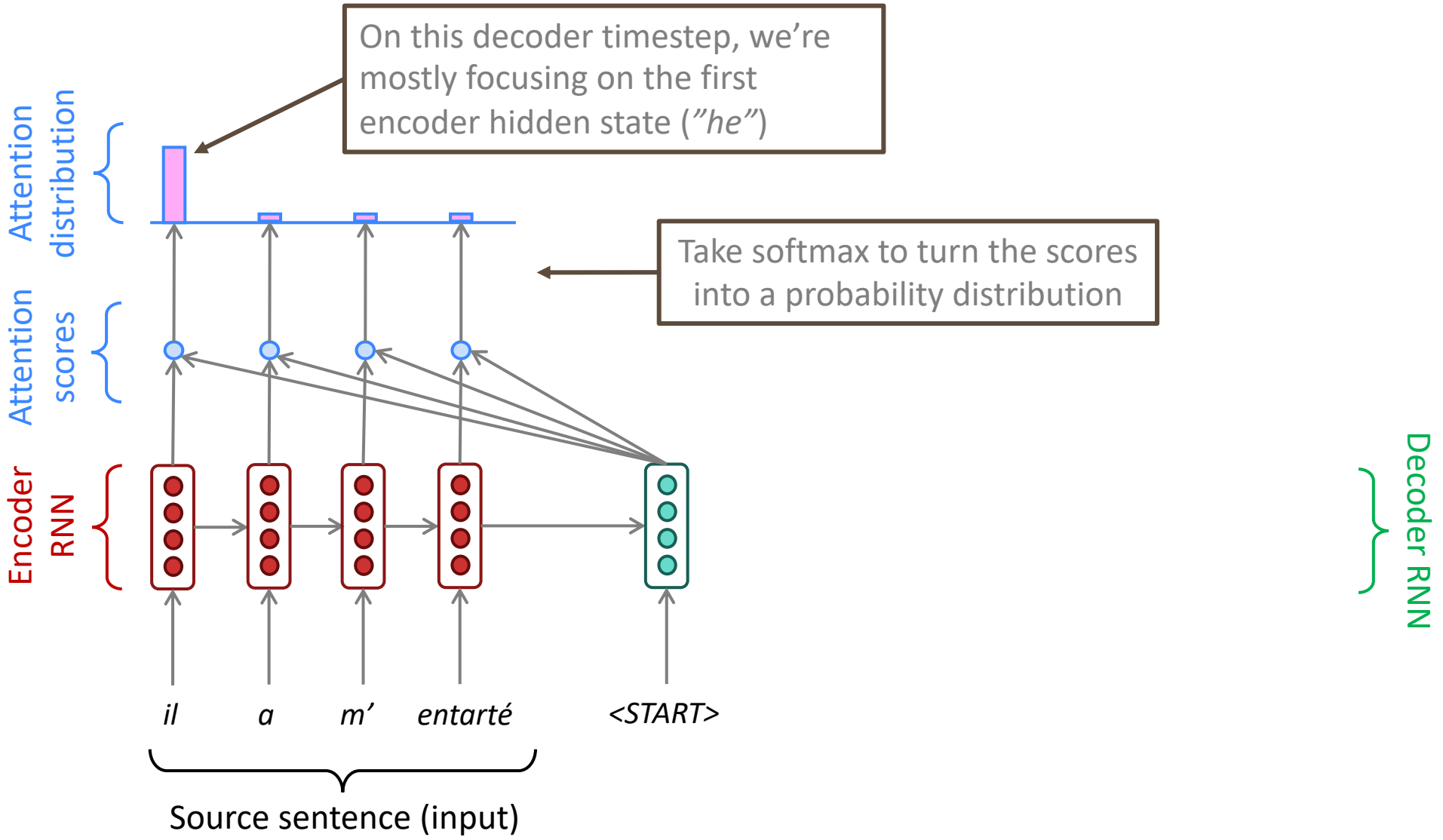
Sequence-to-sequence with attention



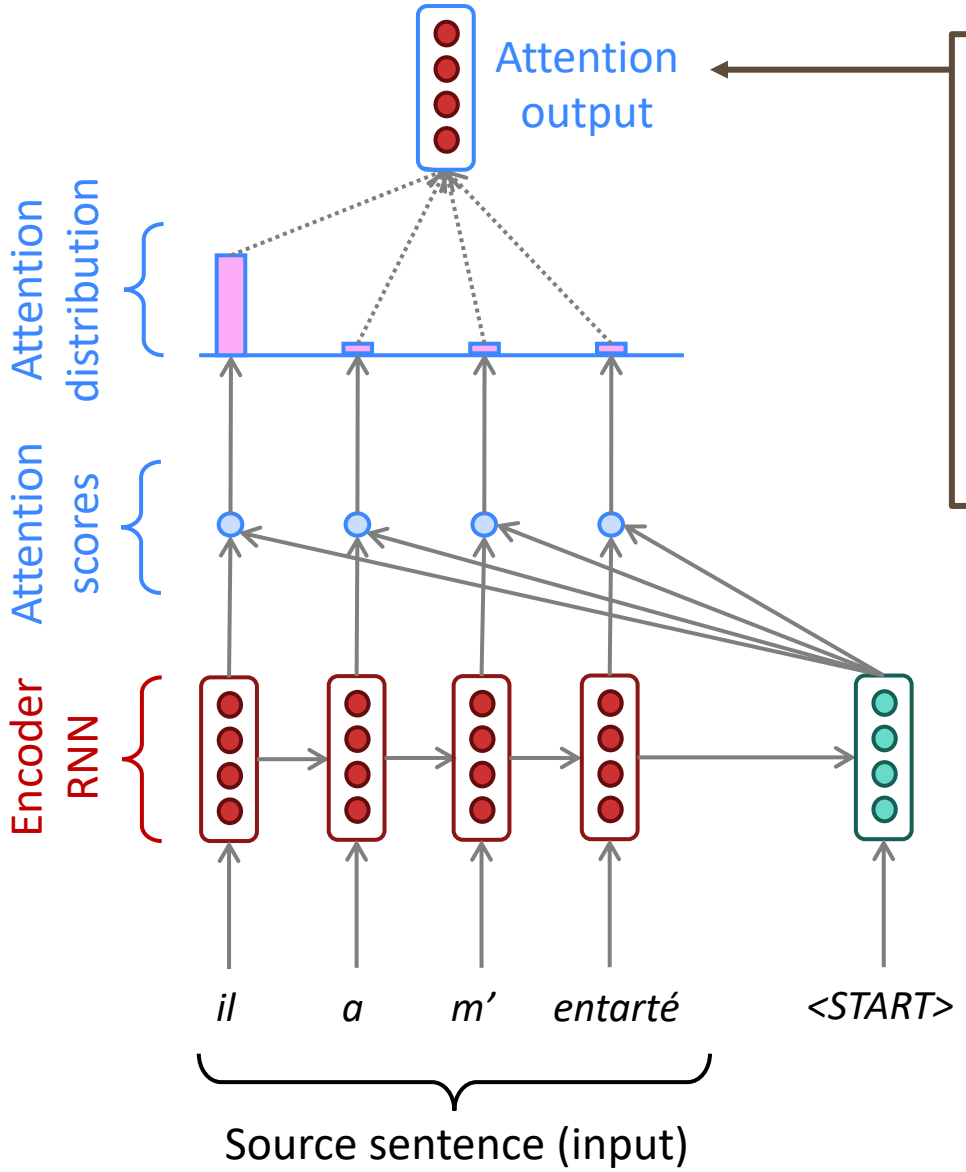
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention

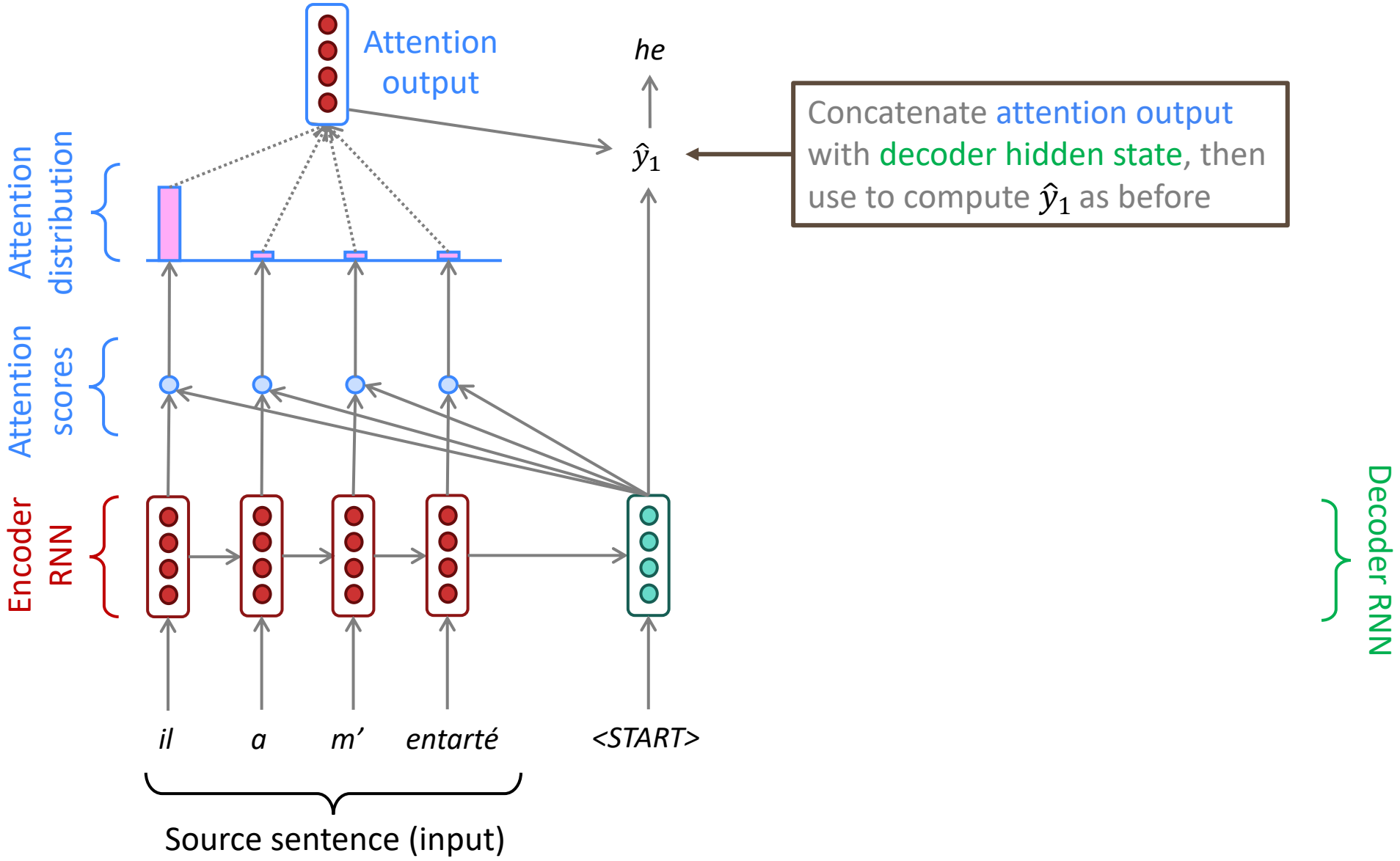


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

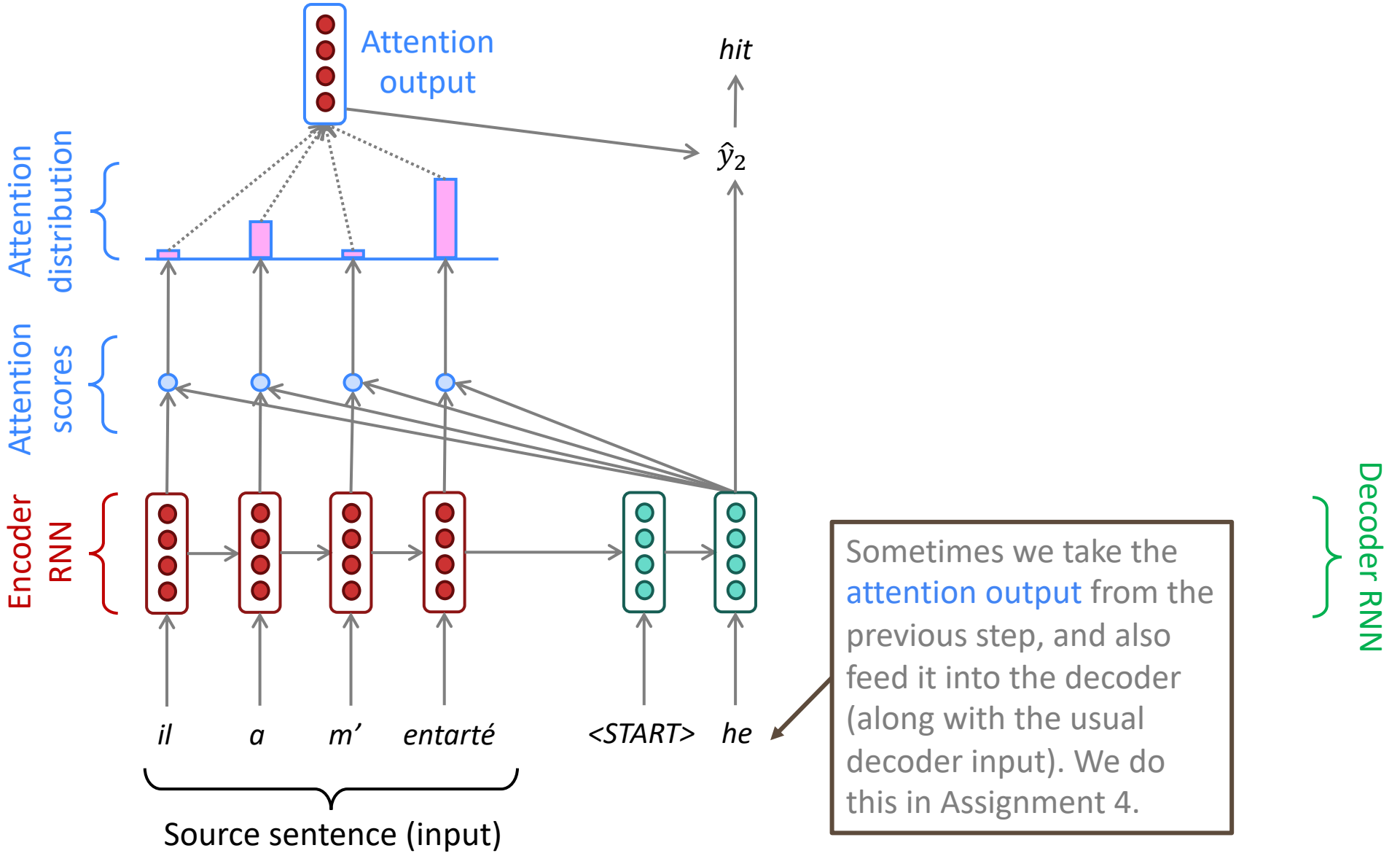
The attention output mostly contains information from the hidden states that received high attention.

Decoder RNN

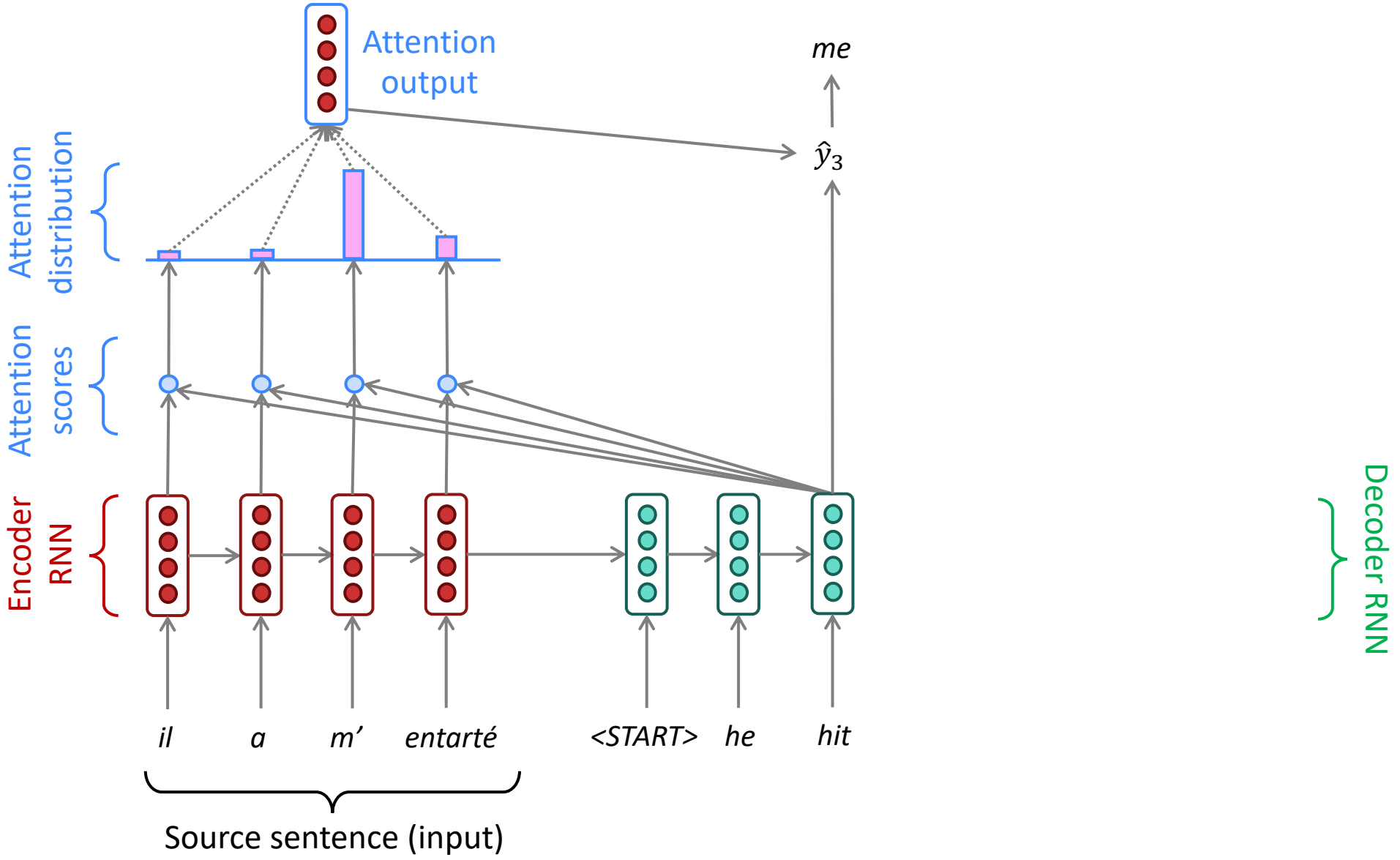
Sequence-to-sequence with attention



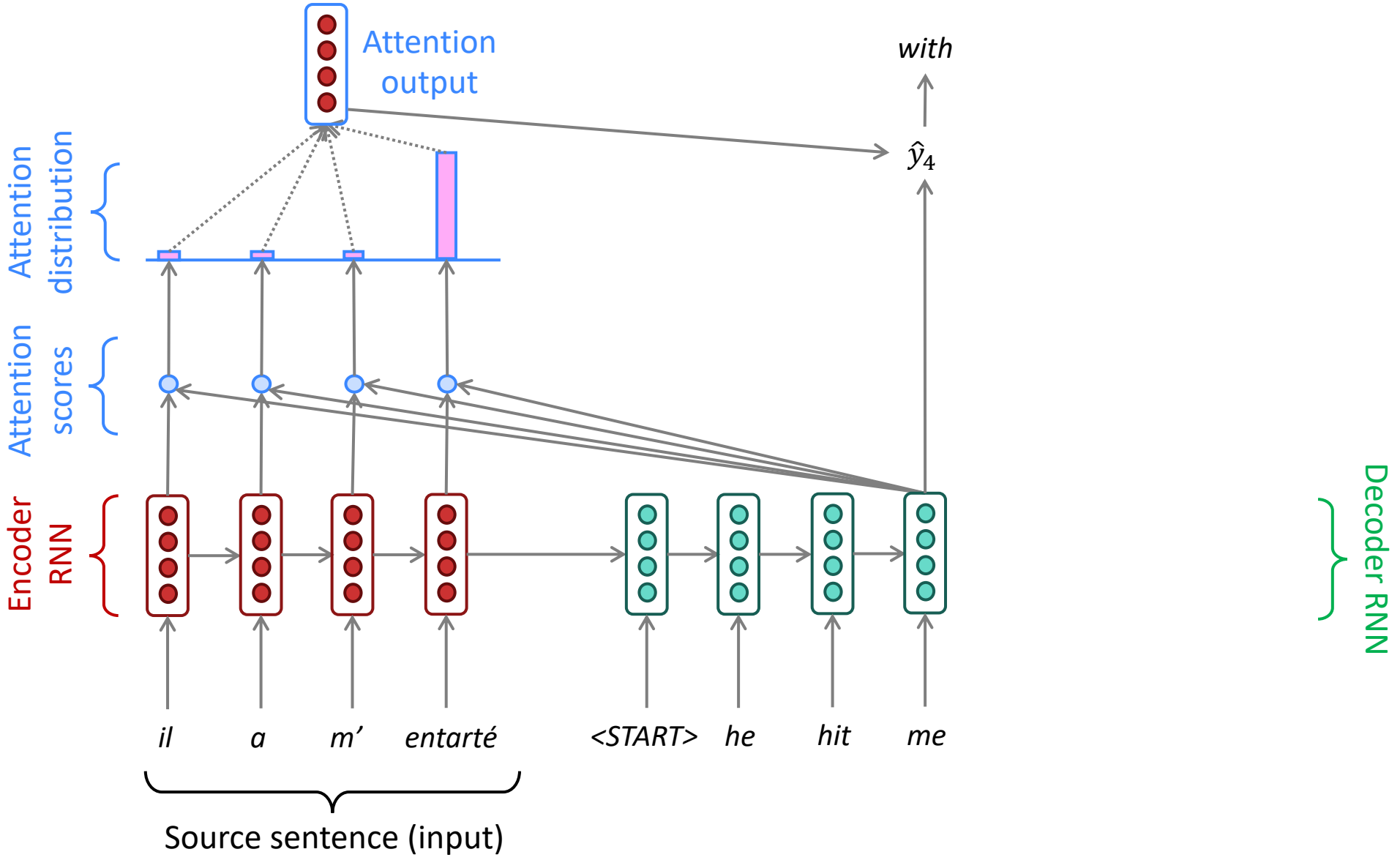
Sequence-to-sequence with attention



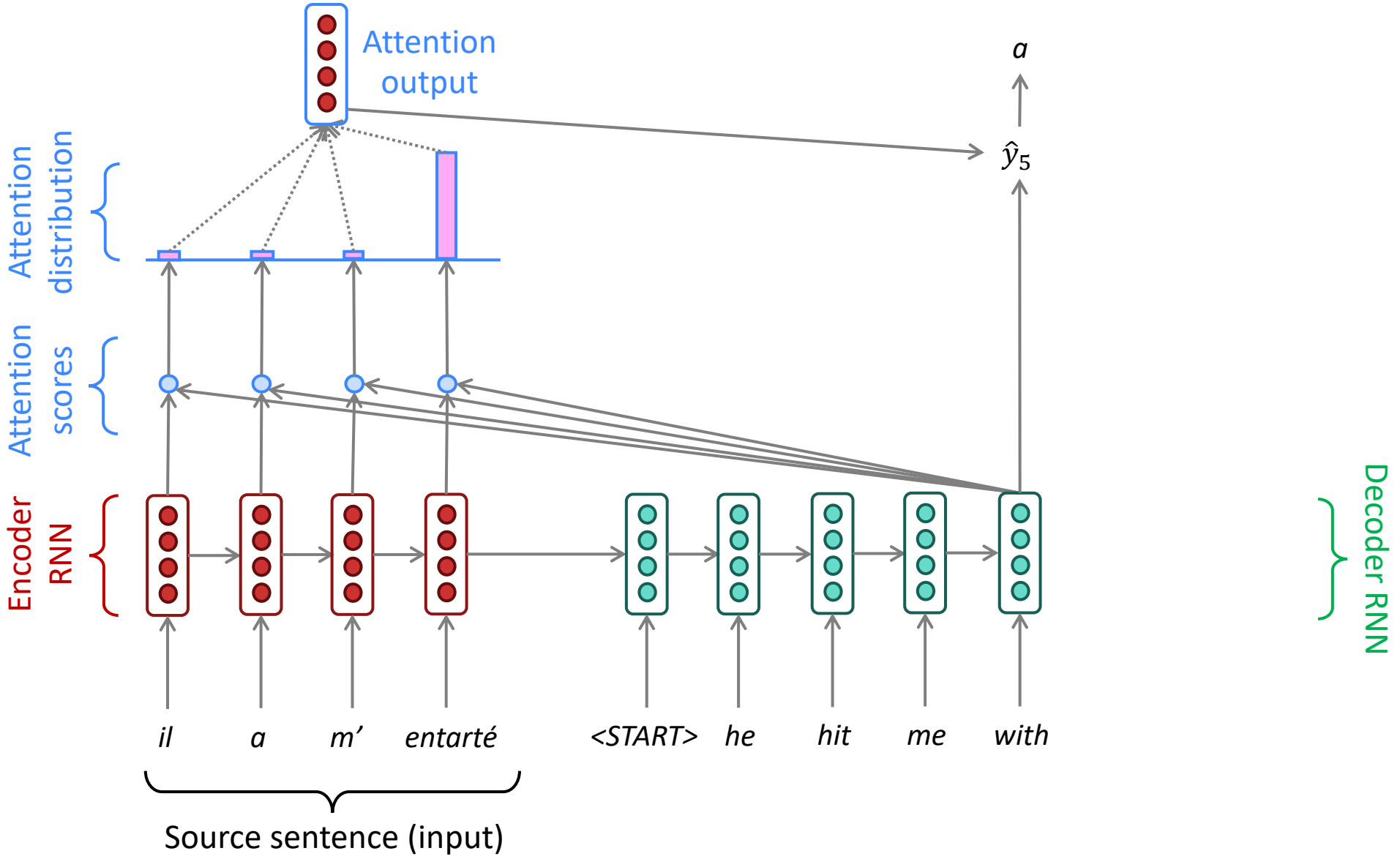
Sequence-to-sequence with attention



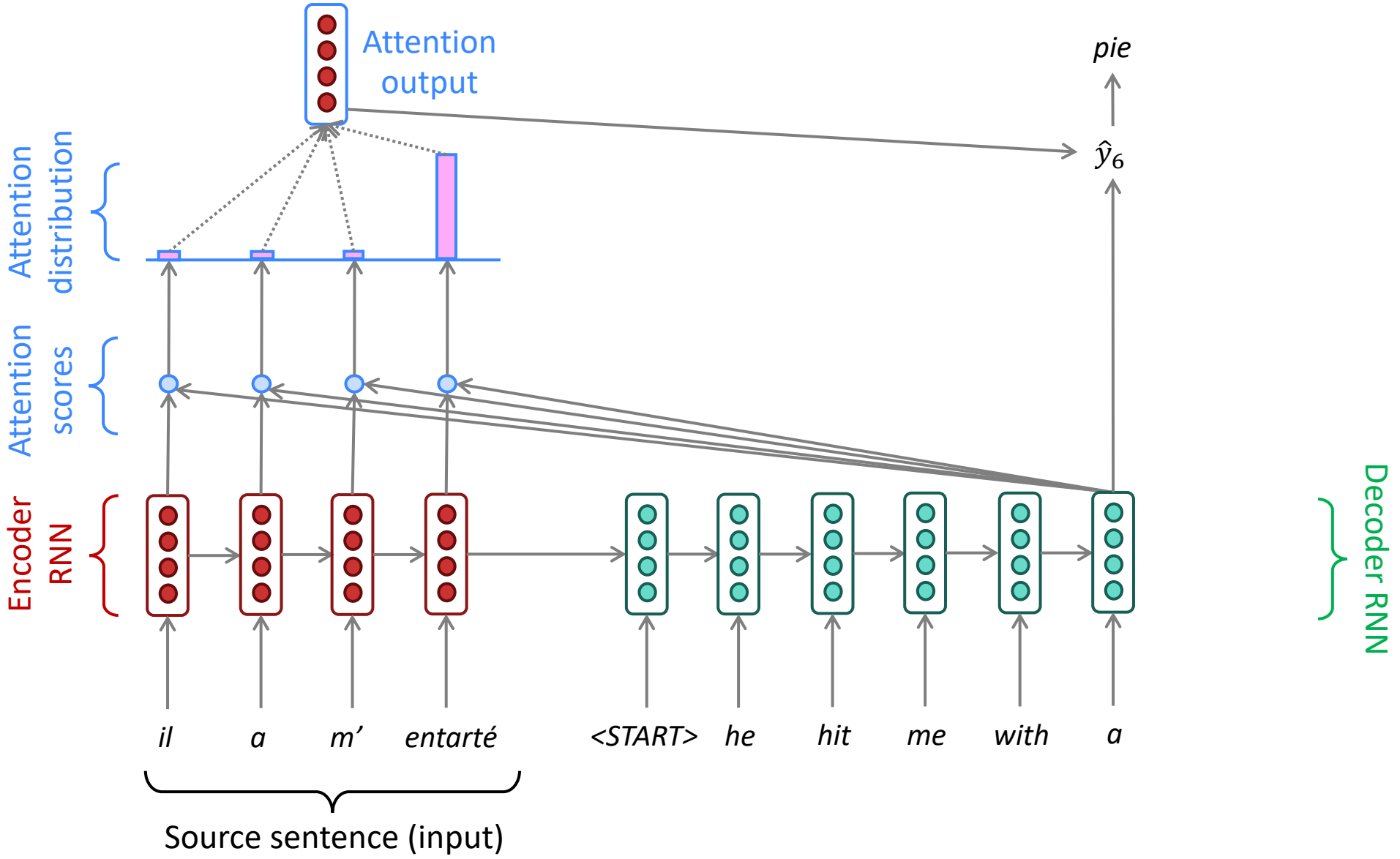
Sequence-to-sequence with attention



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 $\boldsymbol{e}^t$ 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)

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

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

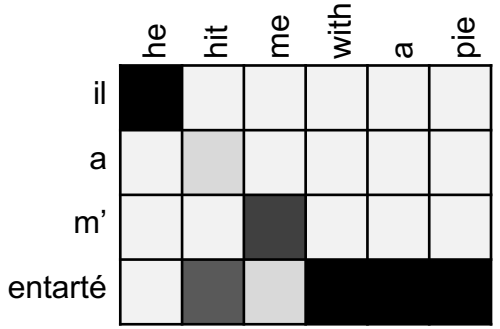
$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- 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

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

Attention is great

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself



Attention is a *general* Deep Learning technique

- We've seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- However: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)

- **More general definition of attention**:
  - Given a set of vector *values*, and a vector *query*, **attention** is a technique to compute a weighted sum of the values, dependent on the query.

- We sometimes say that the query *attends to* the values.
- For example, in the seq2seq + attention model, each decoder hidden state (query) *attends to* all the encoder hidden states (values).

Attention is a *general* Deep Learning technique

More general definition of attention:

Given a set of vector *values*, and a vector *query*, **attention** is a technique to compute a weighted sum of the values, dependent on the query.

Intuition:

- The weighted sum is a *selective summary* of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).

There are *several* attention variants

- We have some *values* $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a *query* $\boldsymbol{s} \in \mathbb{R}^{d_2}$

- Attention always involves:
  1. Computing the *attention scores* $\boldsymbol{e} \in \mathbb{R}^N$ ← There are multiple ways to do this
  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

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

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: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$
  - 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

More information: "Deep Learning for NLP Best Practices", Ruder, 2017. http://ruder.io/deep-learning-nlp-best-practices/index.html#attention
"Massive Exploration of Neural Machine Translation Architectures", Britz et al, 2017, https://arxiv.org/pdf/1703.03906.pdf

Summary of today's lecture

- We learned some history of Machine Translation (MT)
- Since 2014, Neural MT rapidly replaced intricate Statistical MT

- Sequence-to-sequence is the architecture for NMT (uses 2 models: encoder and decoder)
- Attention is a way to *focus on particular parts* of the input
  - Improves sequence-to-sequence a lot!