Neural Machine Translation

Bonan Min

bonanmin@gmail.com

Some slides are based on class materials from Thien Huu Nguyen, Yifan He, Dan Jurasky, Raymond Mooney, Chris Manning and others

Neural Machine Translation (NMT)

A huge breakthrough in NLP appears in 2014.

Neural machine translation is invented that:

- Significantly improves the performance of MT
- Avoids the feature engineering in decades

NMT is the flagship task for NLP Deep Learning

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

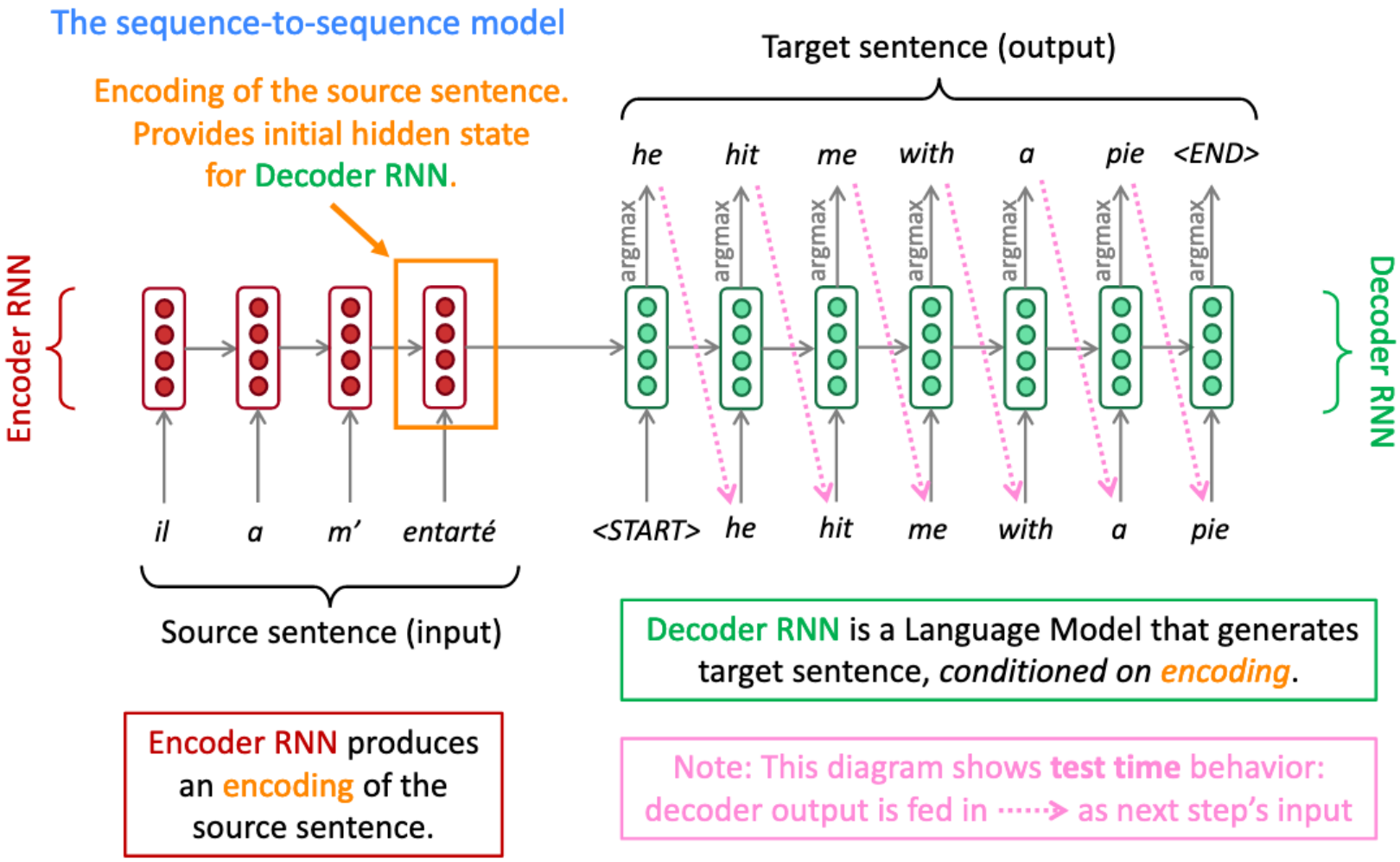
Devlin et al., Fast and Robust Neural Network Joint Models for Statistical Machine Translation. ACL 2014
Cho et al., Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. EMNLP 2014

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.

Neural Machine Translation



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

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 compute $P(y|x)$:

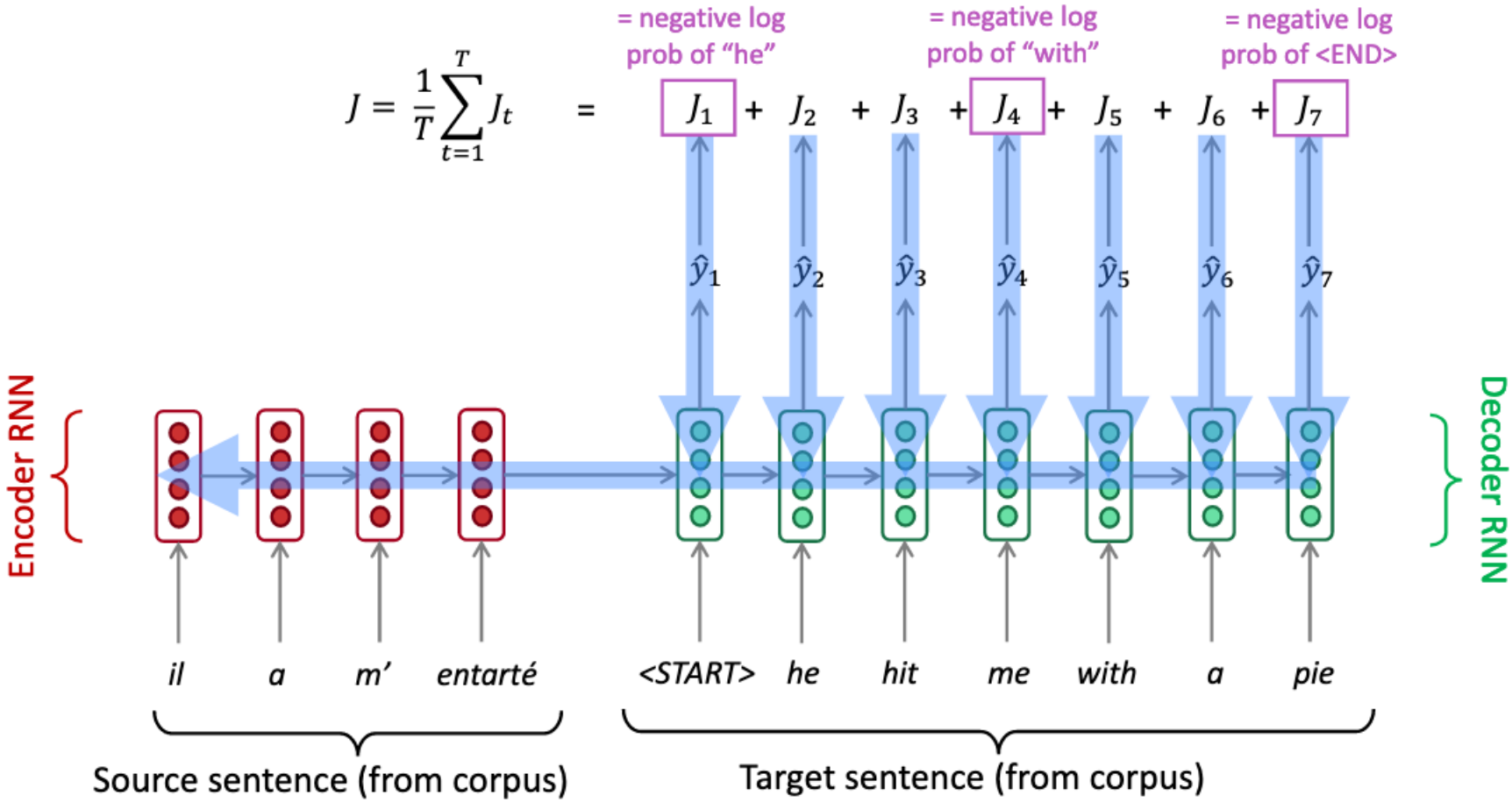
$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots \underbrace{P(y_T|y_1, \ldots, y_{T-1}, x)}$$

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

Question: How do we train a NMT system?

Answer: obtain a large parallel corpus

Training an NMT system

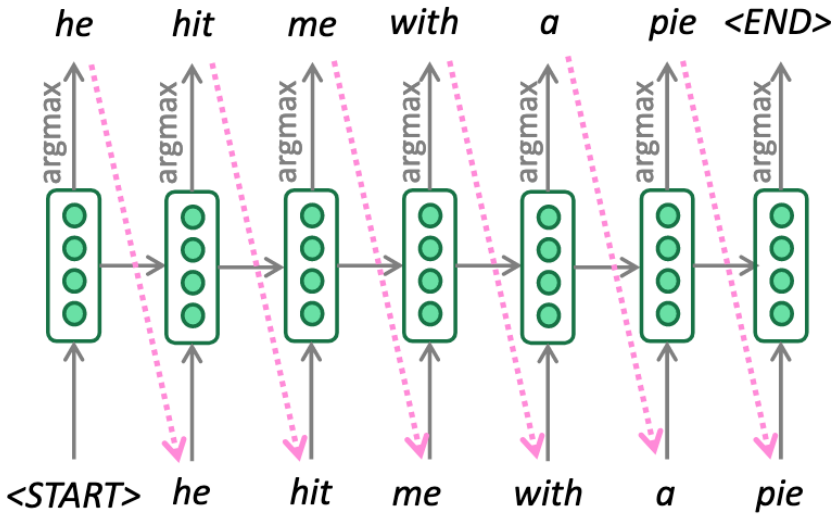


Seq2seq is optimized as **a single system**.
Backpropagation operates "**end-to-end**".

Greedy Decoding

Decoding: finding the best sentence in the target language for a given source sentence based on the current model

Greedy decoding: taking argmax (the most probable word) on each step of the decoder



What are the problems with this approach?

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 $O(VT)$ 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 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. The numbers = $score(y_1, \dots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, \dots, y_{i-1}, x)$

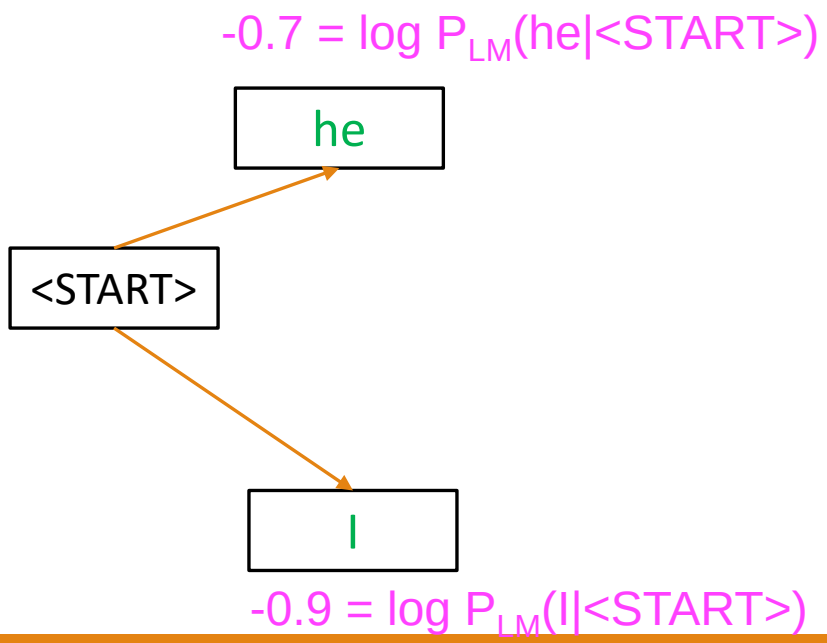
Calculate prob dist of next word

<START>

Beam Search Decoding: Example

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

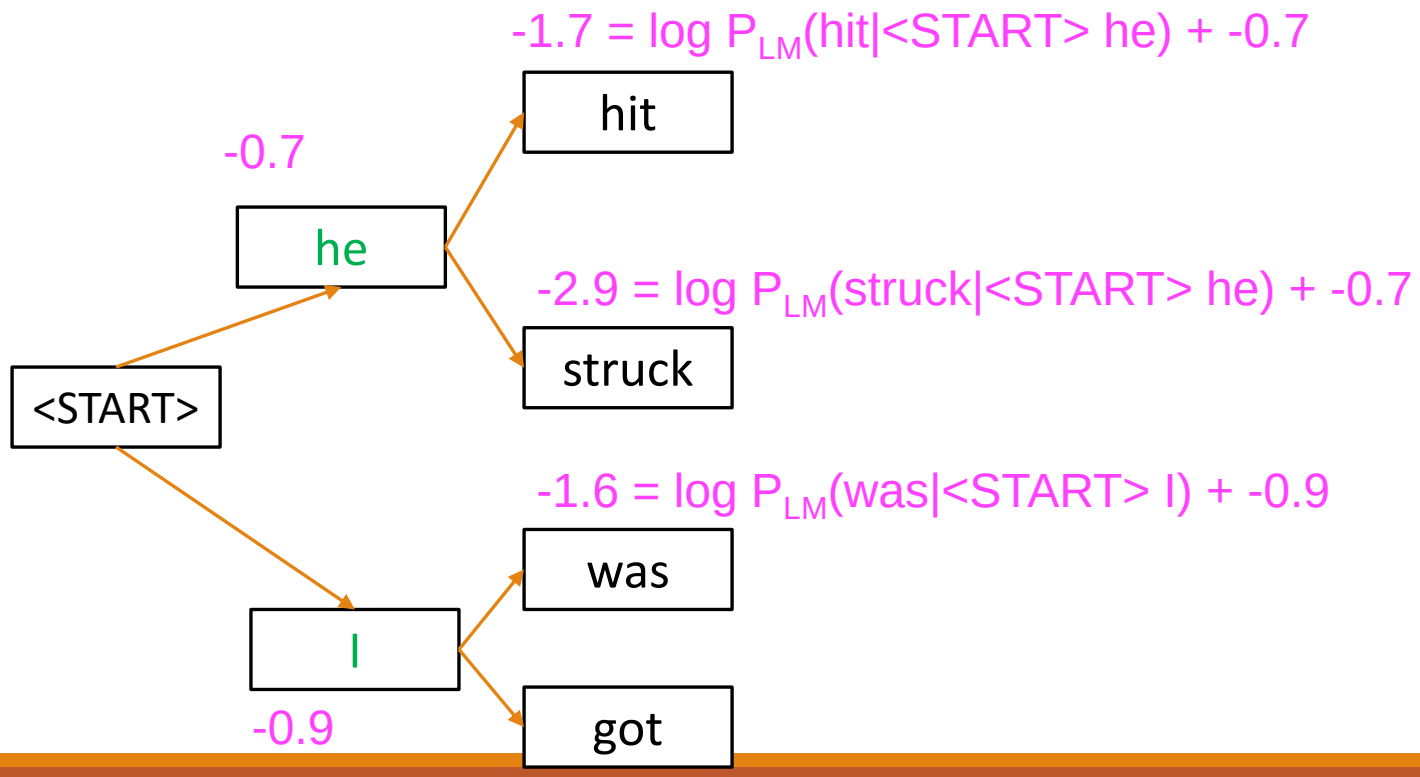
Take top k words and compute scores



Beam Search Decoding: Example

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

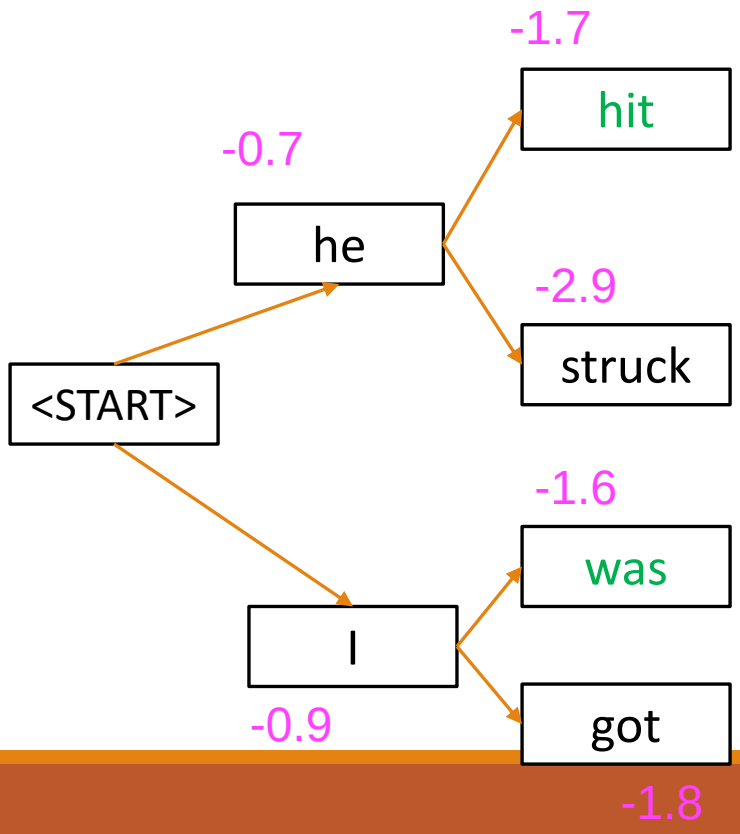
For each of the k hypotheses, find top k next words and calculate scores



Beam Search Decoding: Example

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

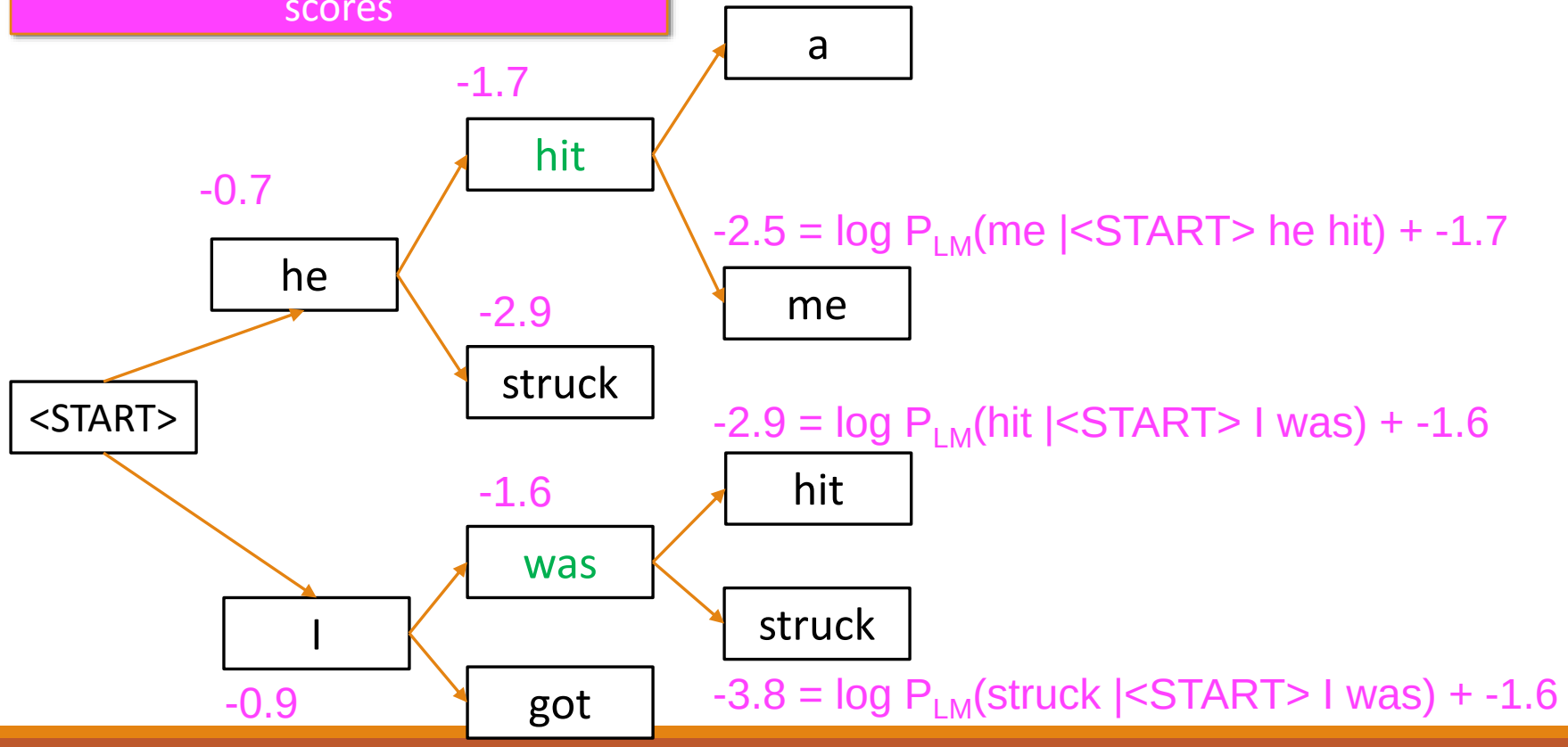
Of these k^2 hypotheses, just keep k with highest scores



Beam Search Decoding: Example

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

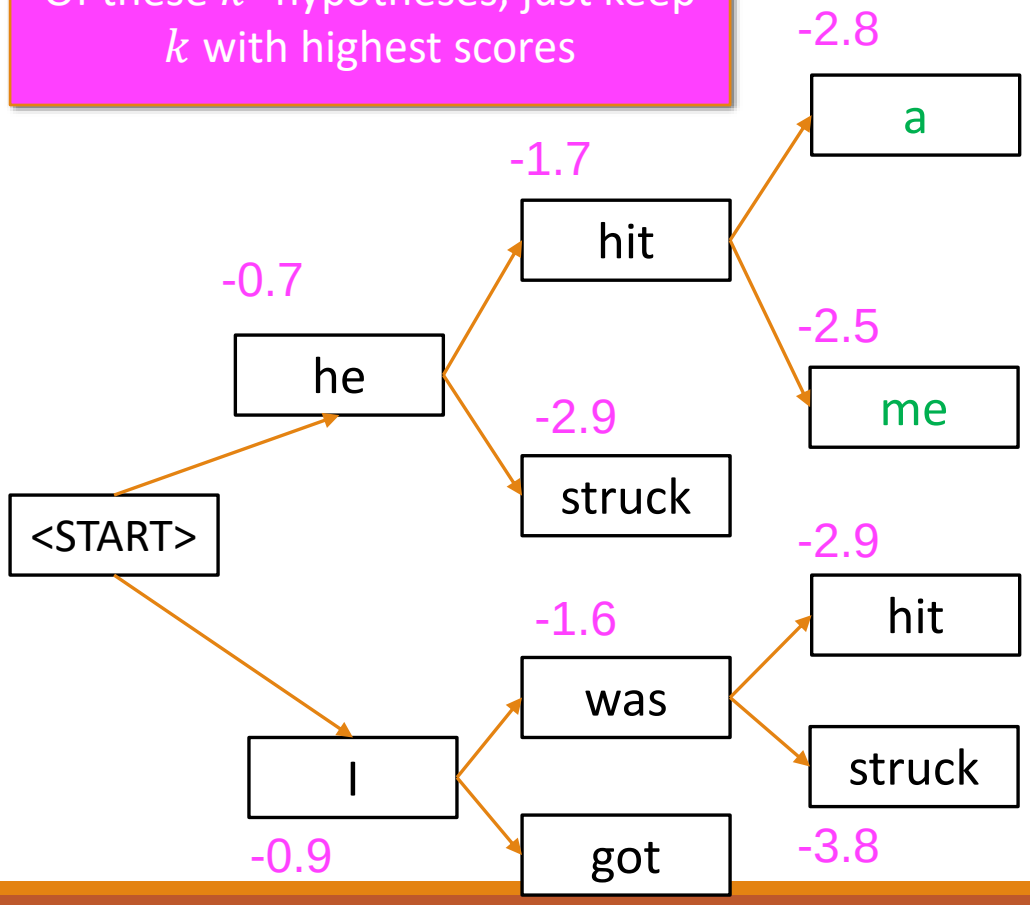
For each of the k hypotheses, find top k next words and calculate scores



Beam Search Decoding: Example

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

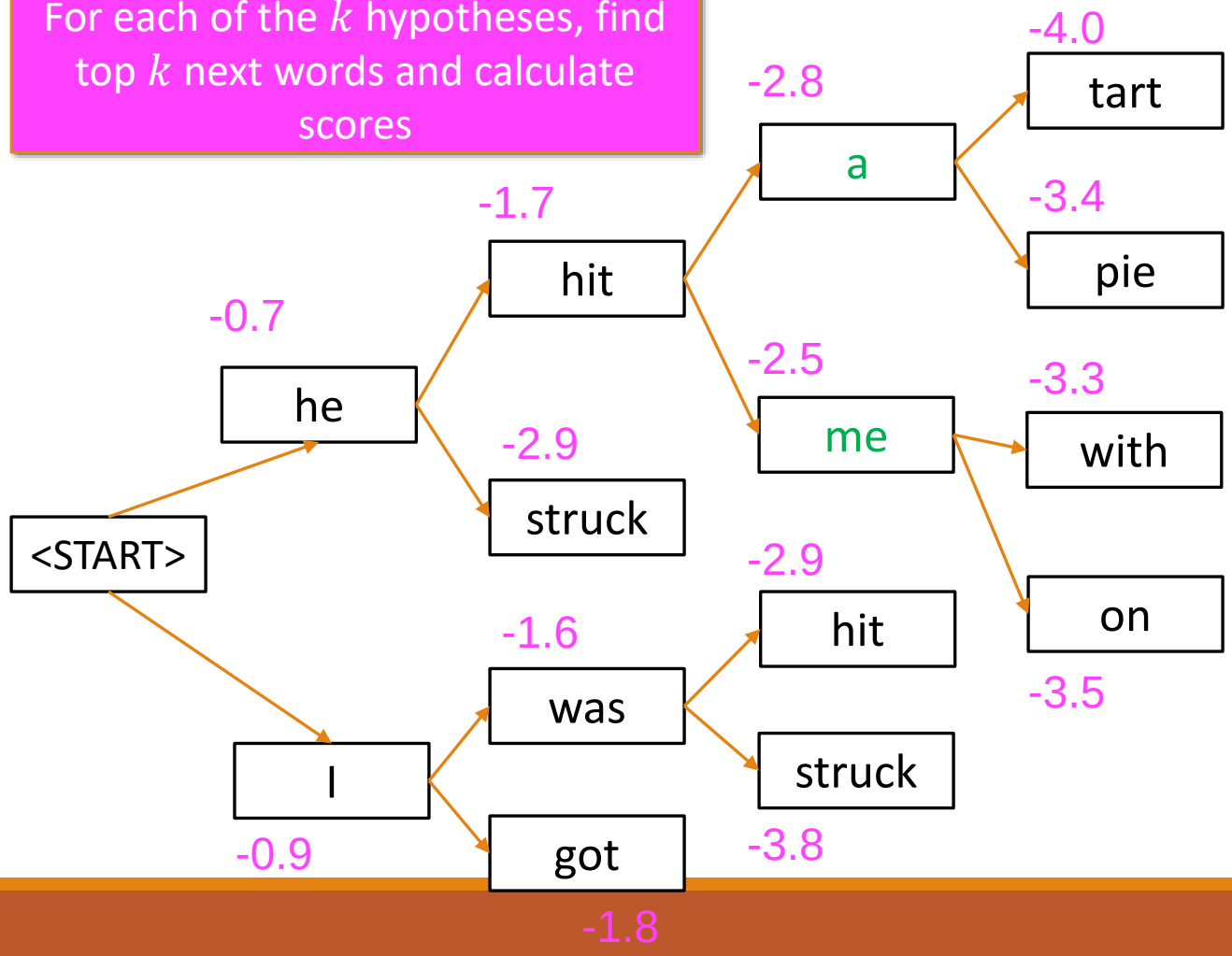
Of these k^2 hypotheses, just keep k with highest scores



Beam Search Decoding: Example

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

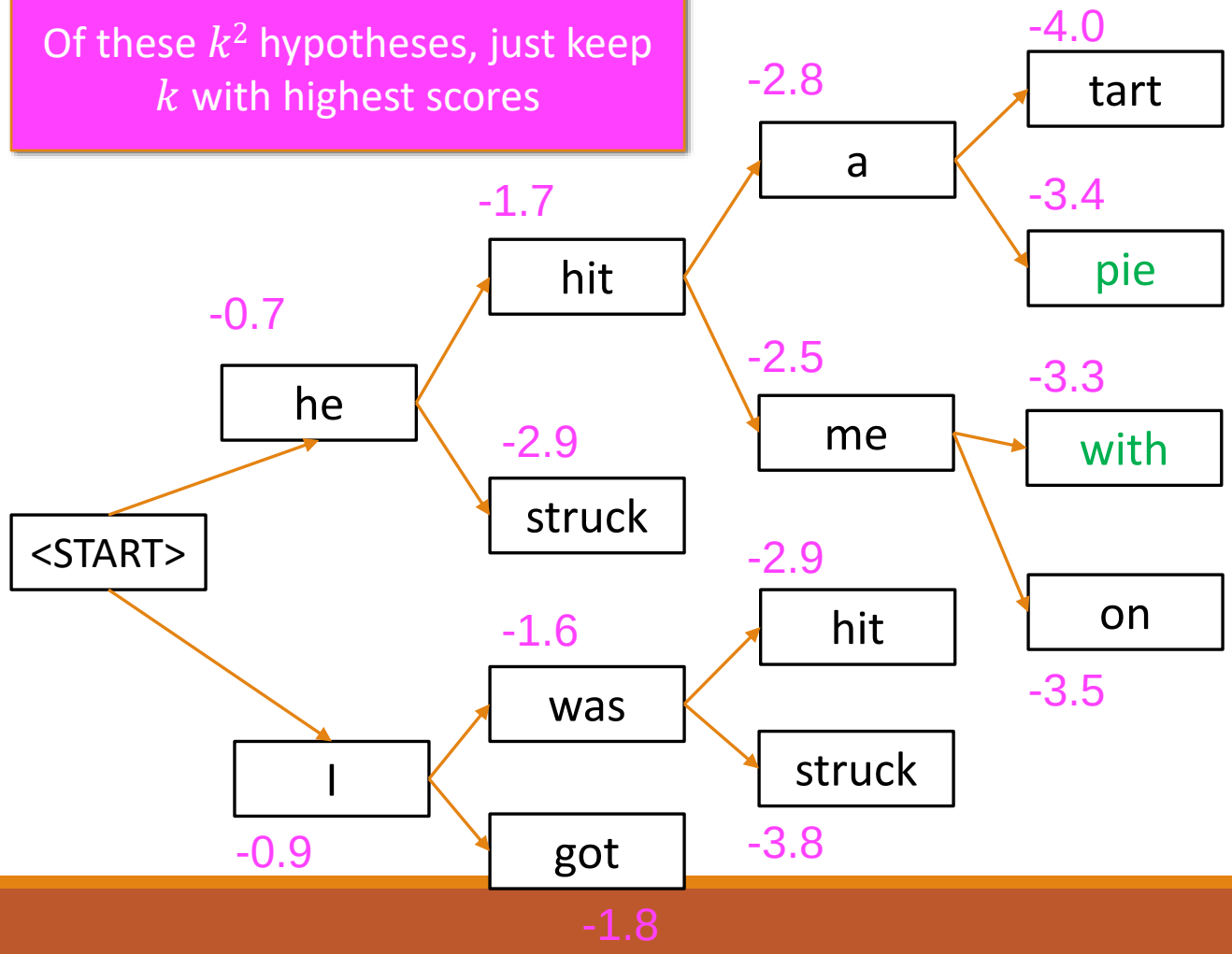
For each of the k hypotheses, find top k next words and calculate scores



Beam Search Decoding: Example

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

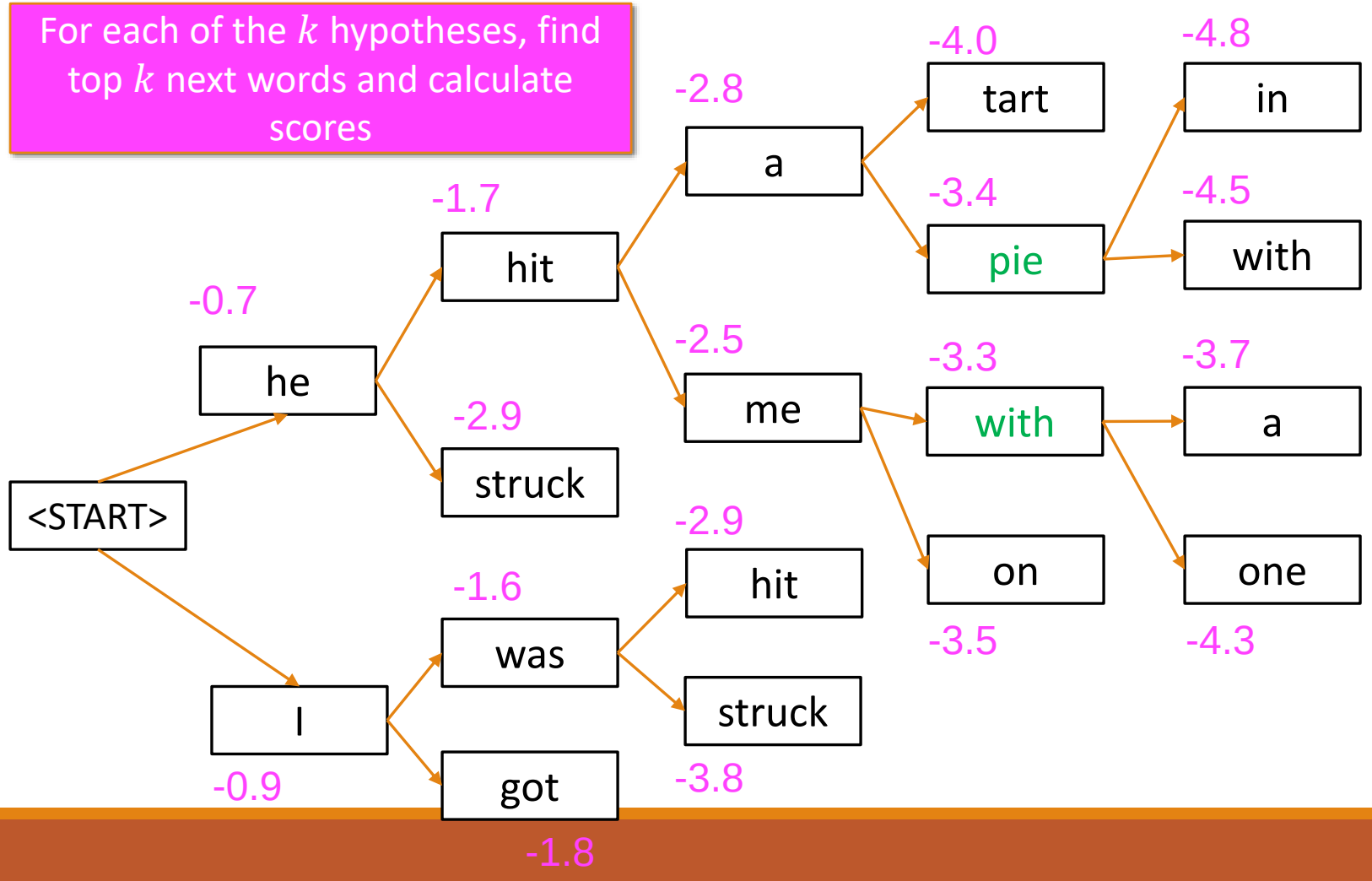
Of these k^2 hypotheses, just keep k with highest scores



Beam Search Decoding: Example

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

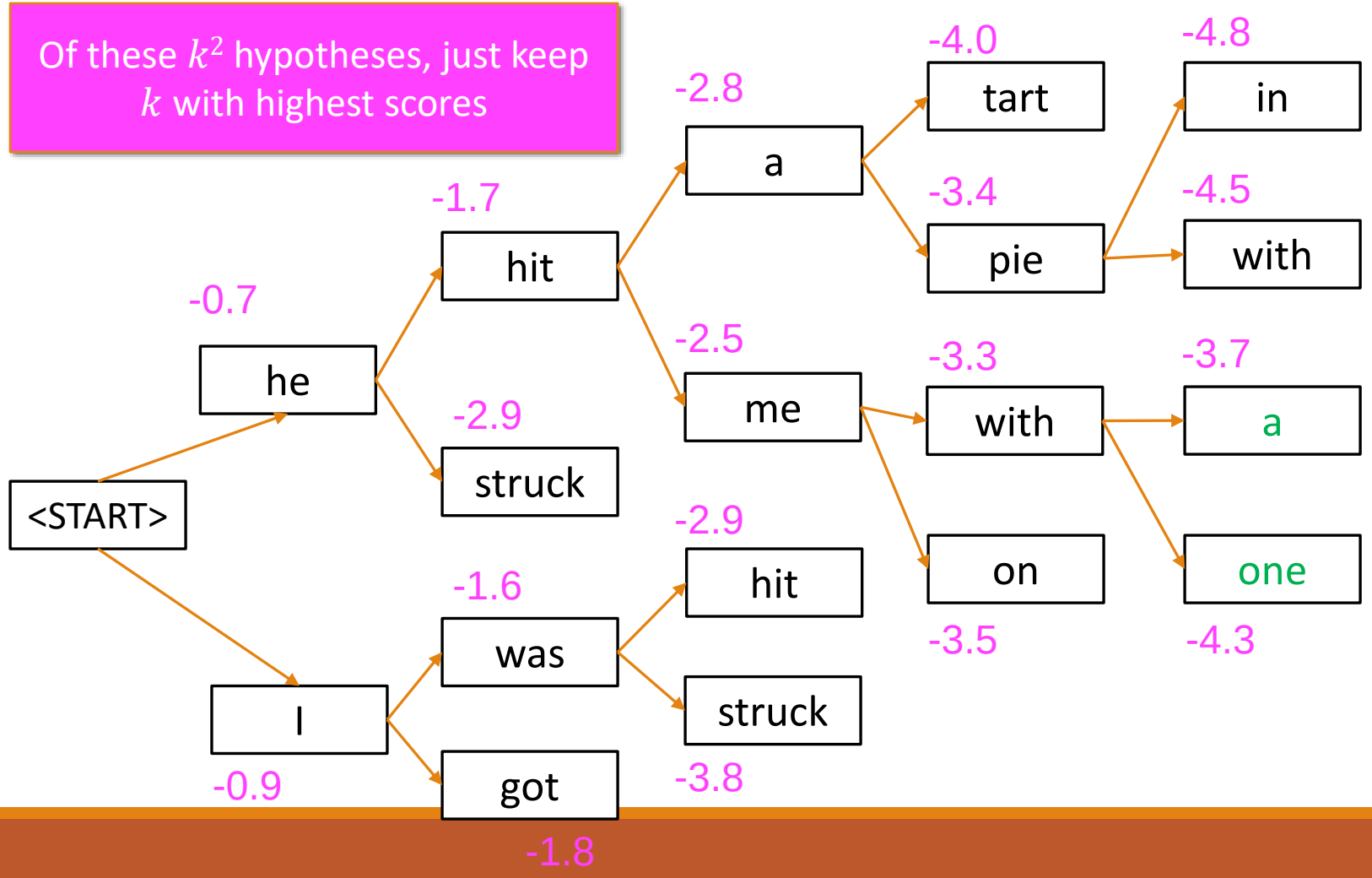
For each of the k hypotheses, find top k next words and calculate scores



Beam Search Decoding: Example

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

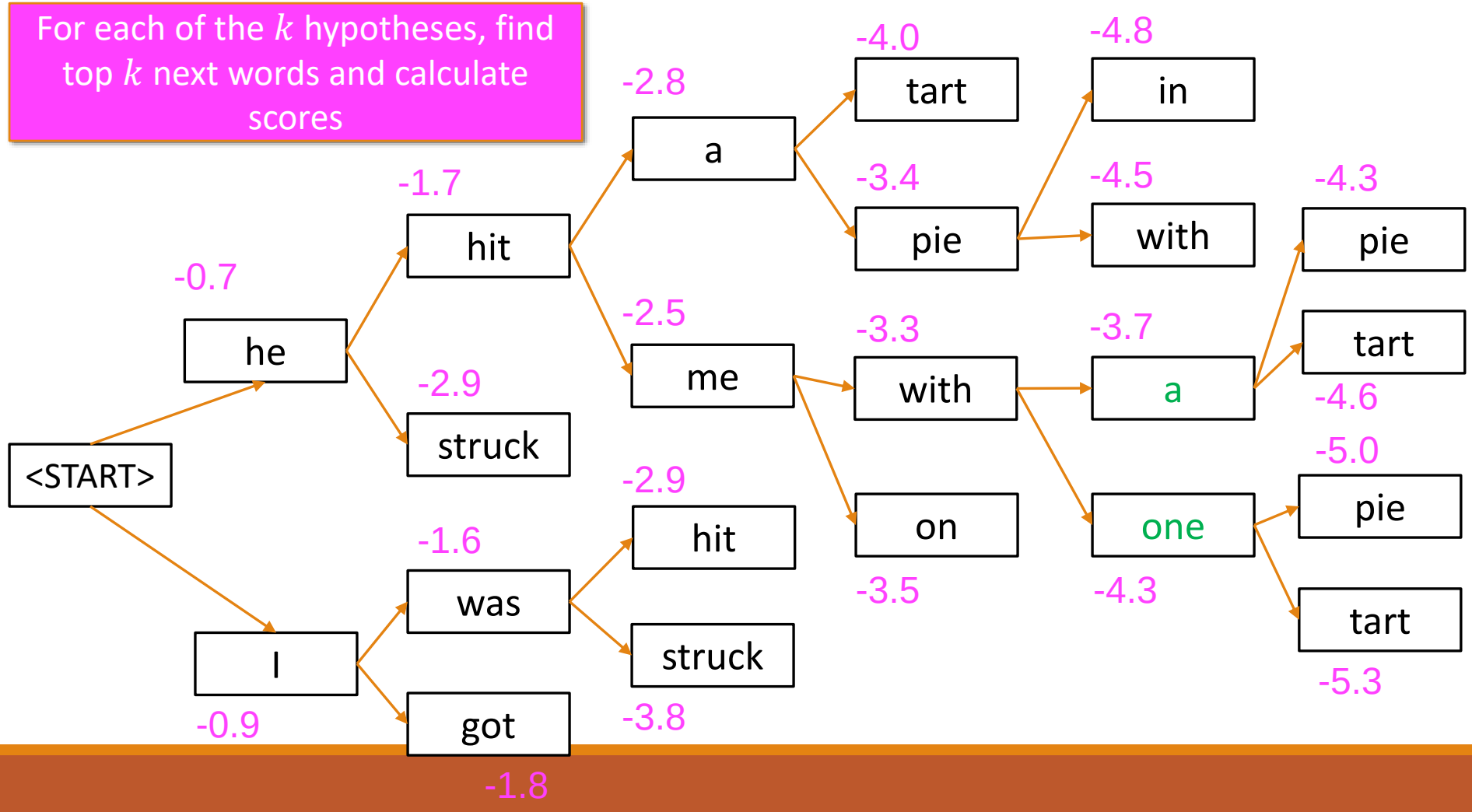
Of these k^2 hypotheses, just keep k with highest scores



Beam Search Decoding: Example

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

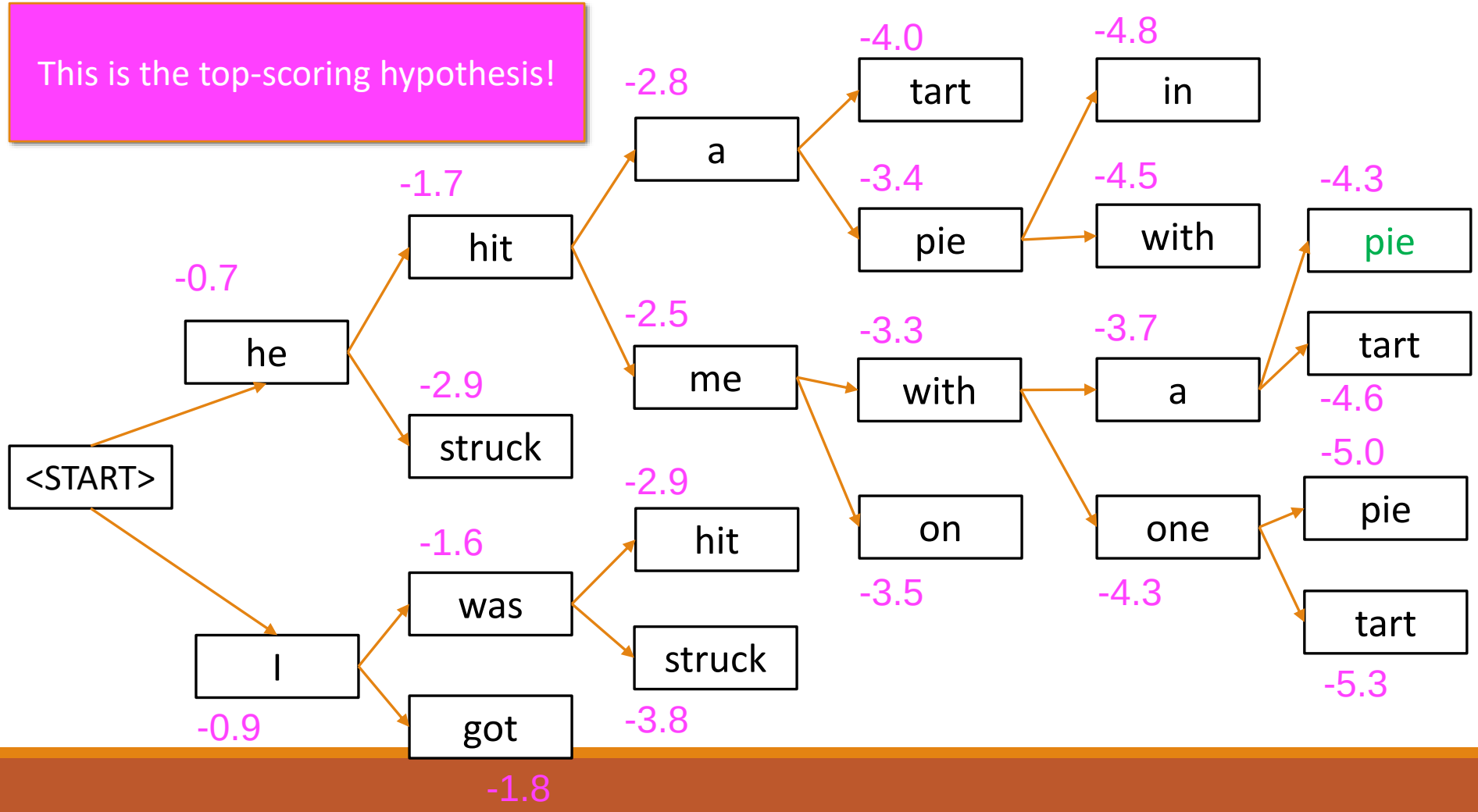
For each of the k hypotheses, find top k next words and calculate scores



Beam Search Decoding: Example

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

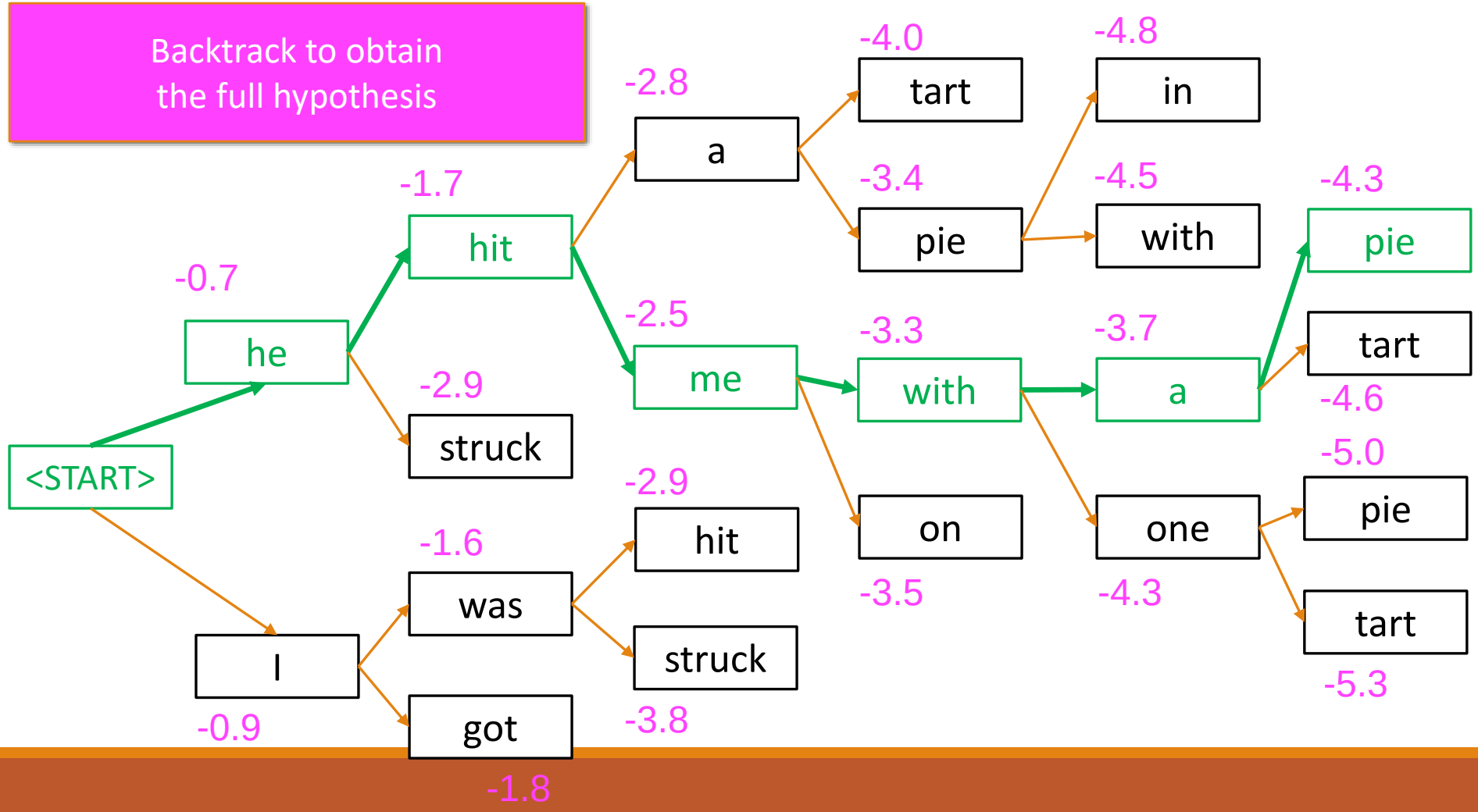
This is the top-scoring hypothesis!



Beam Search Decoding: Example

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

Backtrack to obtain the full hypothesis



Beam Search Decoding: Stopping Criterion

In greedy decoding, usually we decode until the model produces 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)

Beam Search Decoding: Finishing Up

We have our list of completed hypotheses.

How to select top one with highest score?

Each hypothesis 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:

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

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:

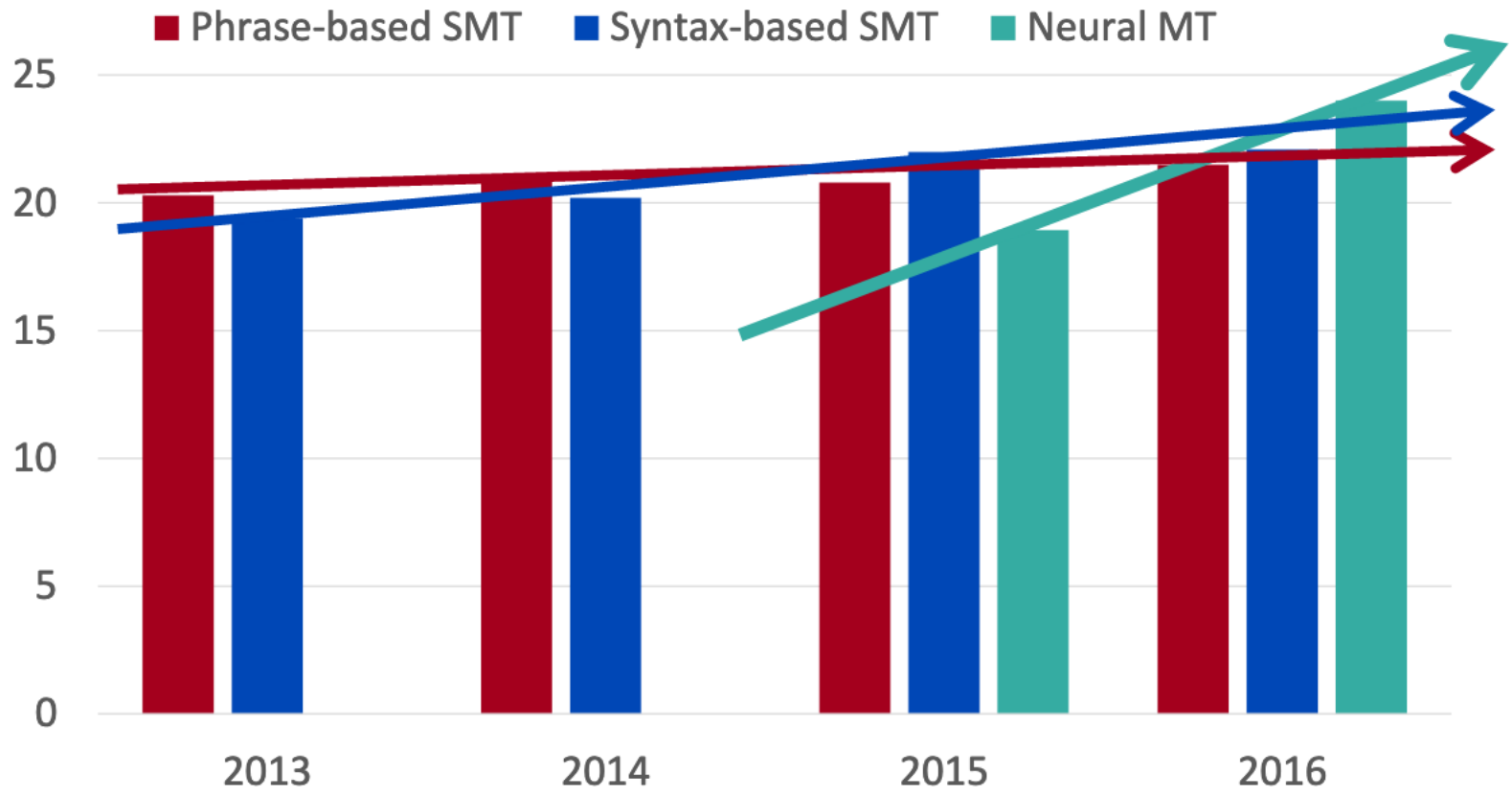
NMT is less interpretable

- Hard to debug

NMT is difficult to control

- For example, can't easily specify rules or guidelines for translation
- Safety concerns!

MT Progress Over Time



NMT: The Biggest Success Story Of NLP Deep Learning

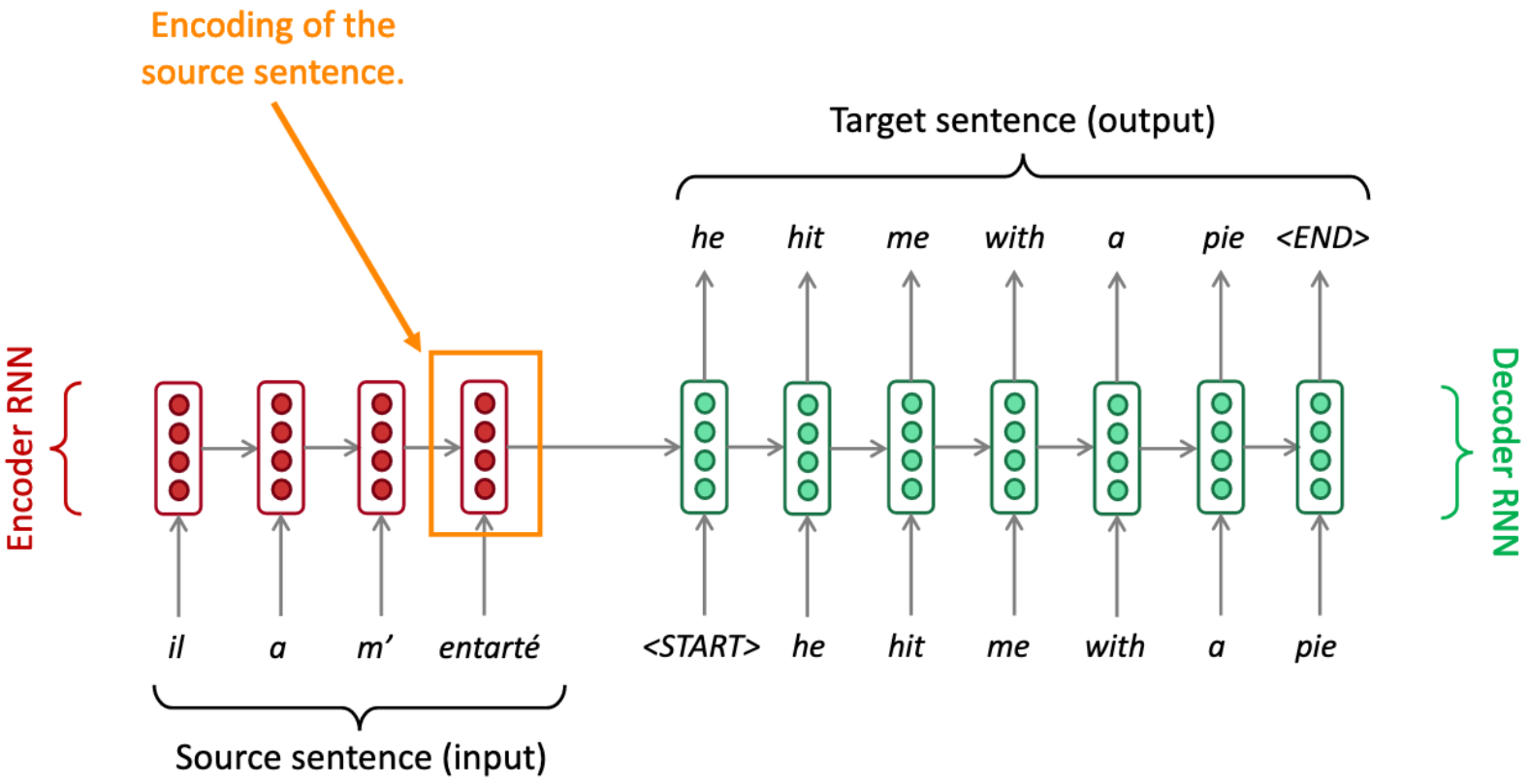
Neural Machine Translation went from a fringe research activity in **2014** to the leading standard method in **2016**

- **2014**: First seq2seq paper published
- **2016**: Google Translate switches from SMT to NMT

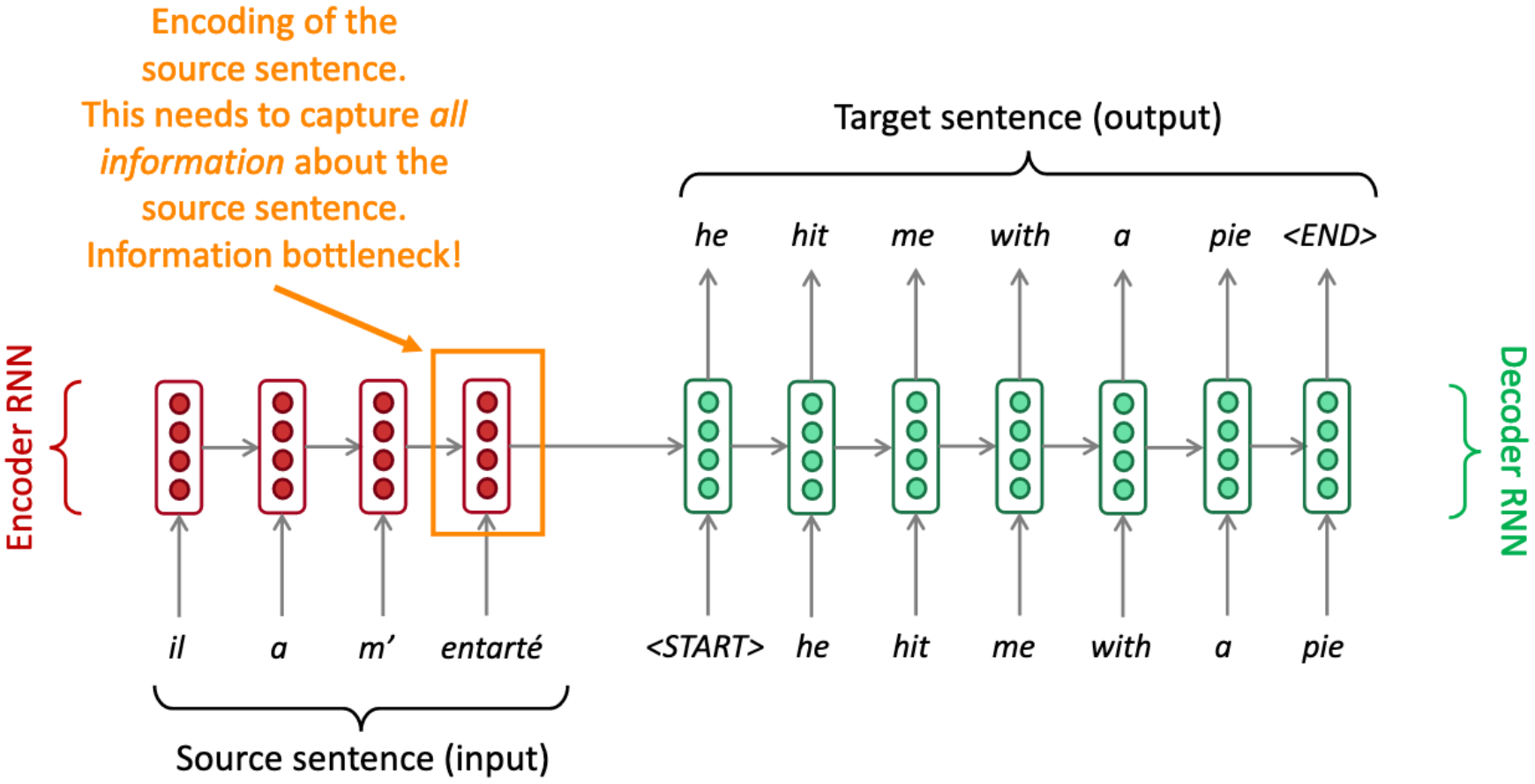
This is amazing!

- **SMT** systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by a handful of engineers in a few months

Problems Of Sequence-to-Sequence



Sequence-to-sequence: The Bottleneck Problem



Attention Mechanism

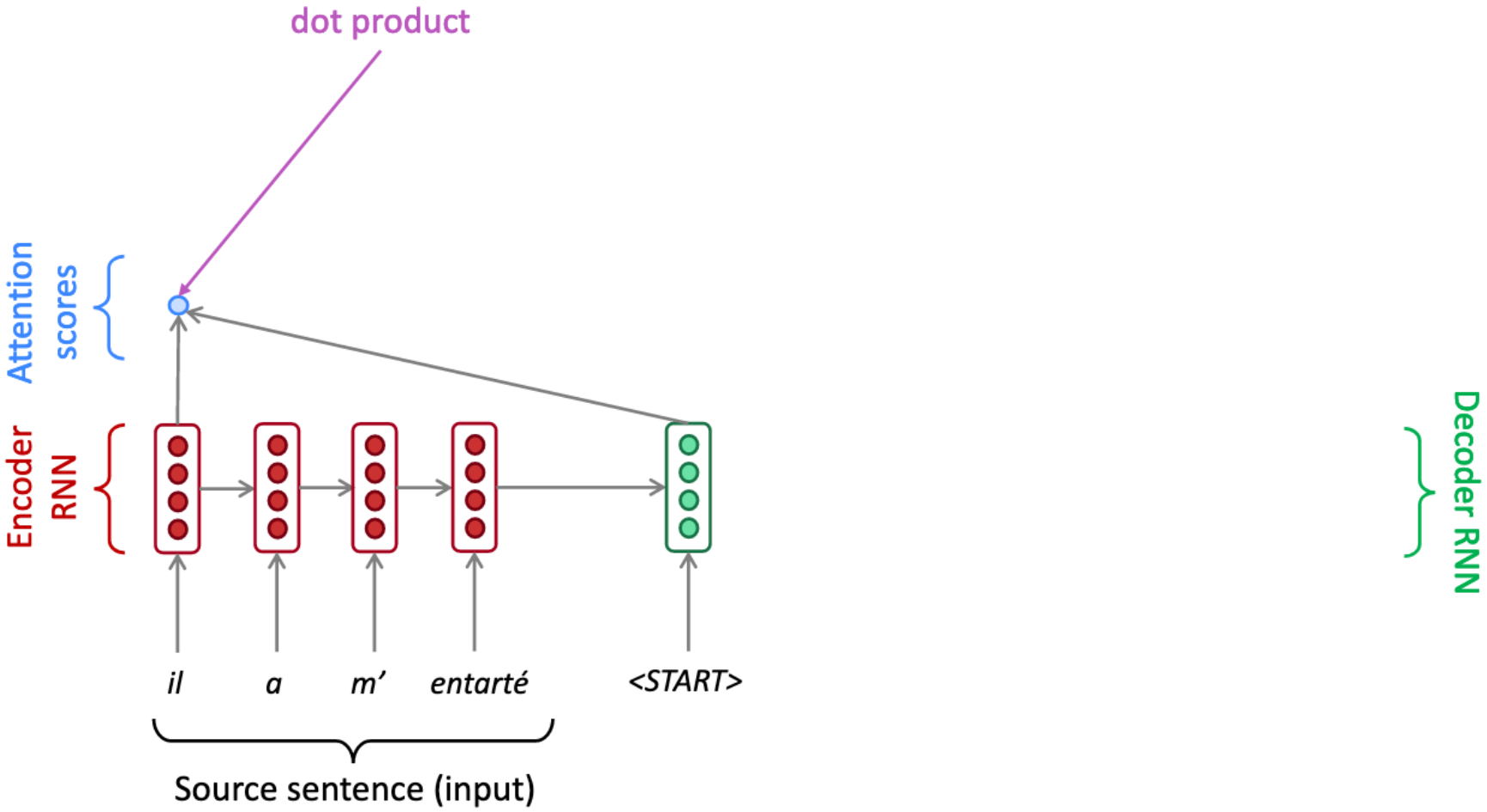
Attention provides a solution to the bottleneck problem.

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

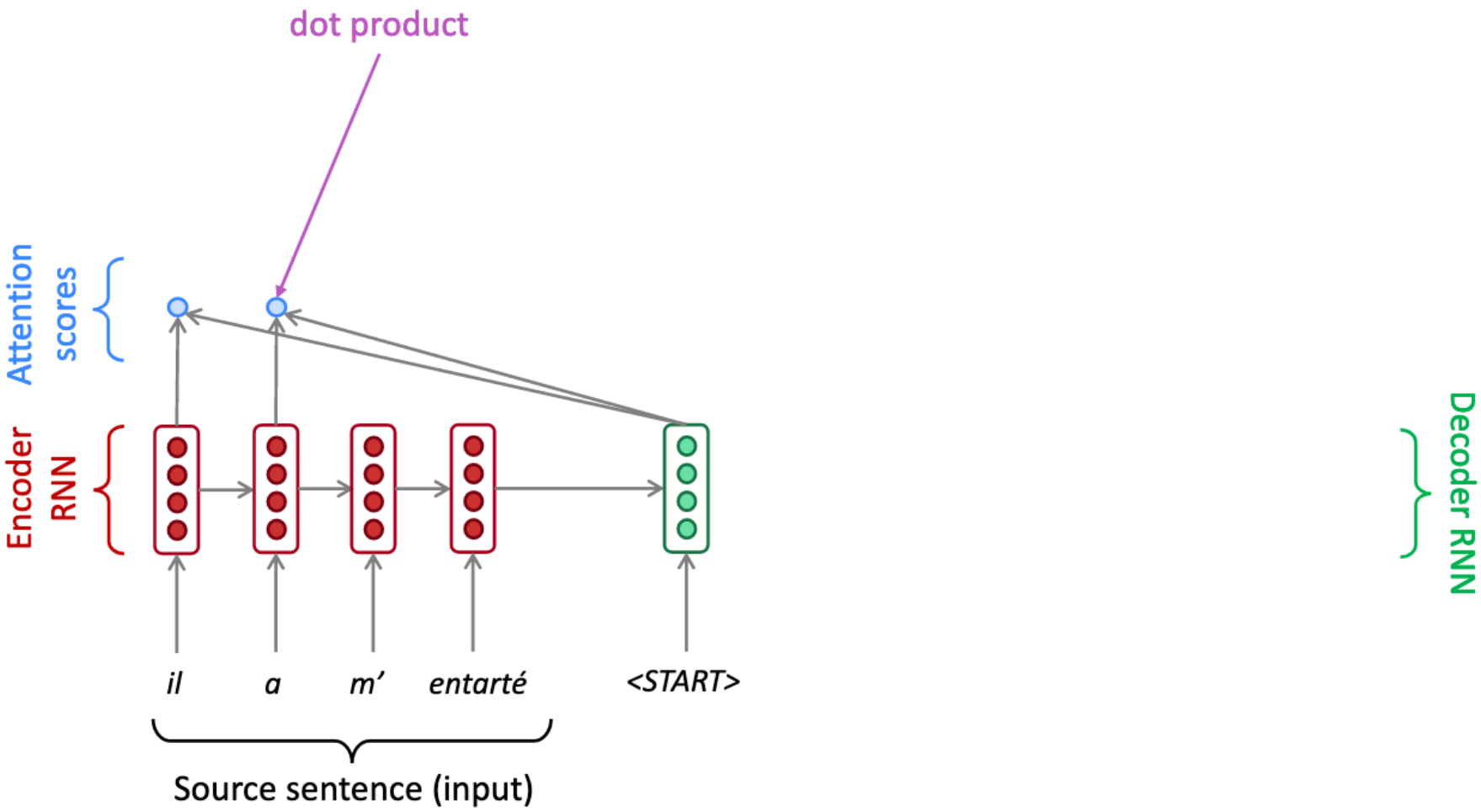
First we will show via diagram (no equations), then we will show with equations

Dzmitry Bahdanau, et al., Neural Machine Translation By Jointly Learning To Align And Translate. ICLR 2015

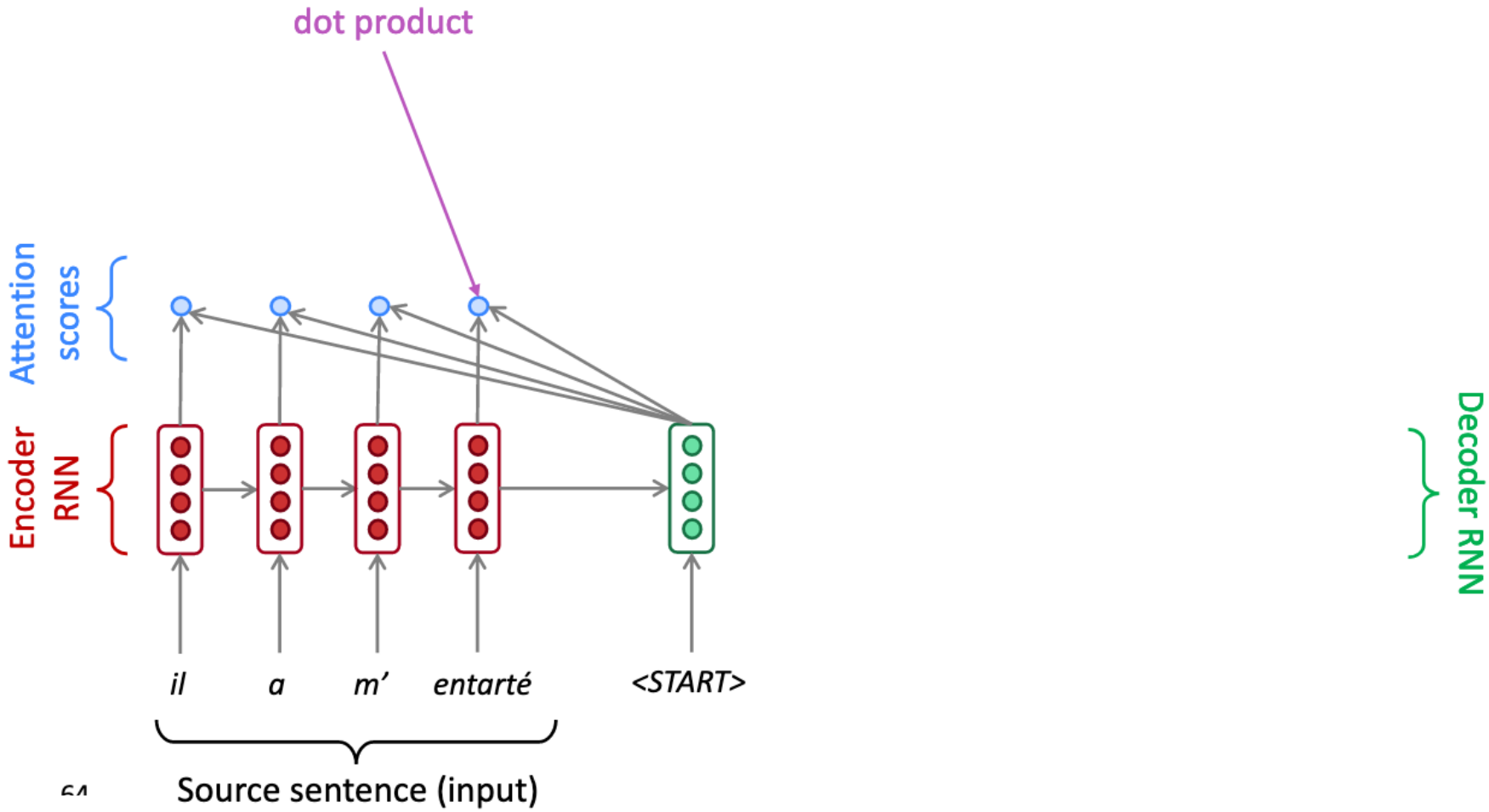
Sequence-to-sequence With Attention



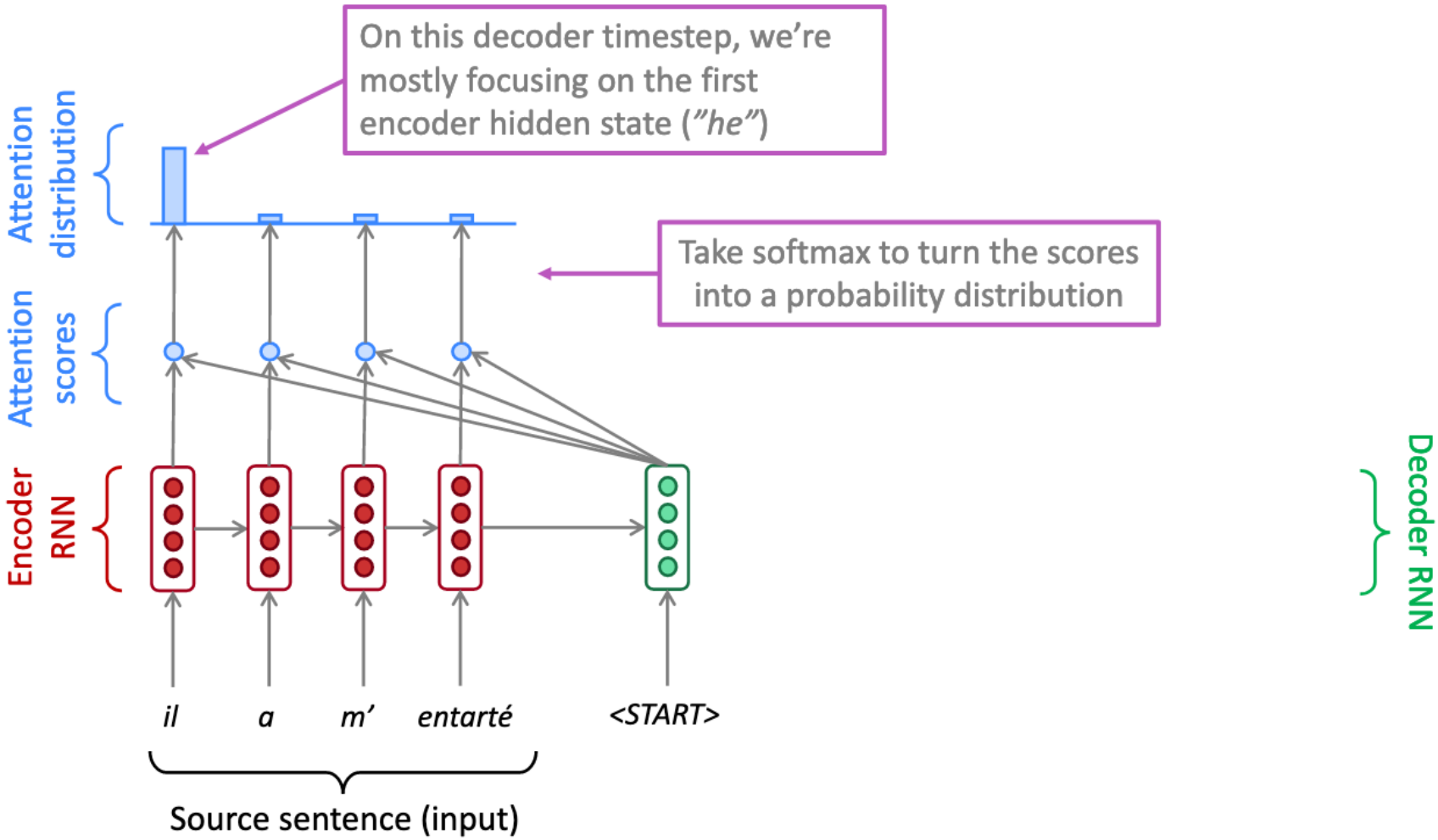
Sequence-to-sequence With Attention



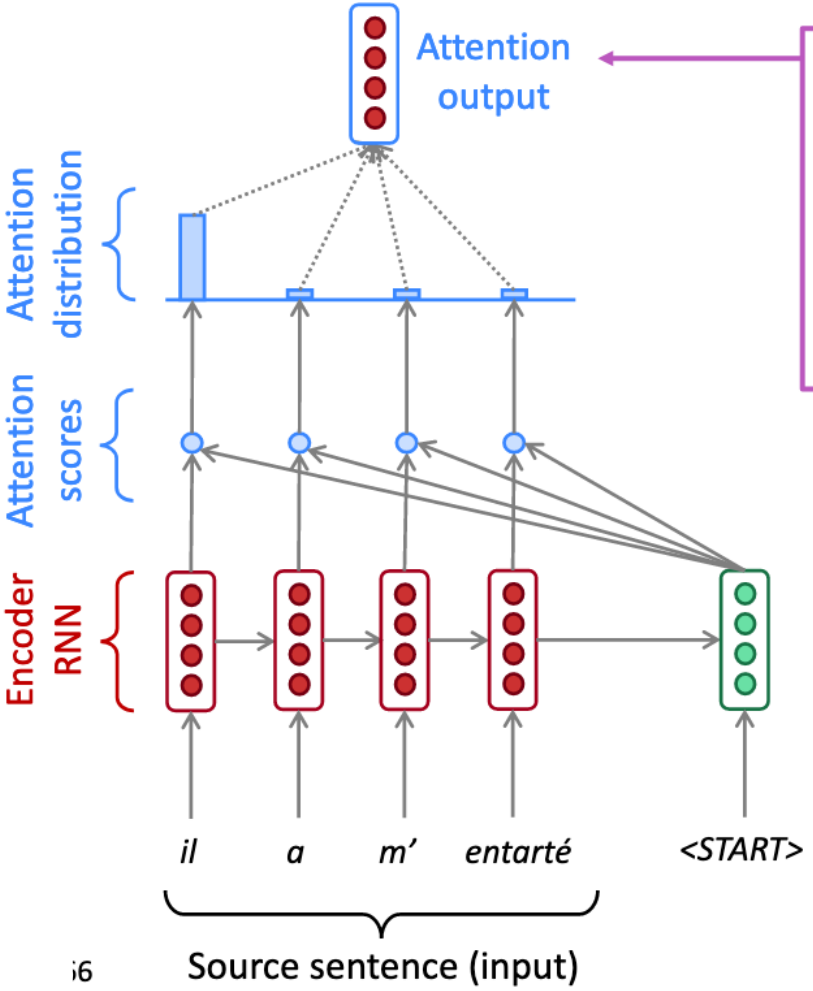
Sequence-to-sequence With Attention



Sequence-to-sequence With Attention



Sequence-to-sequence With Attention

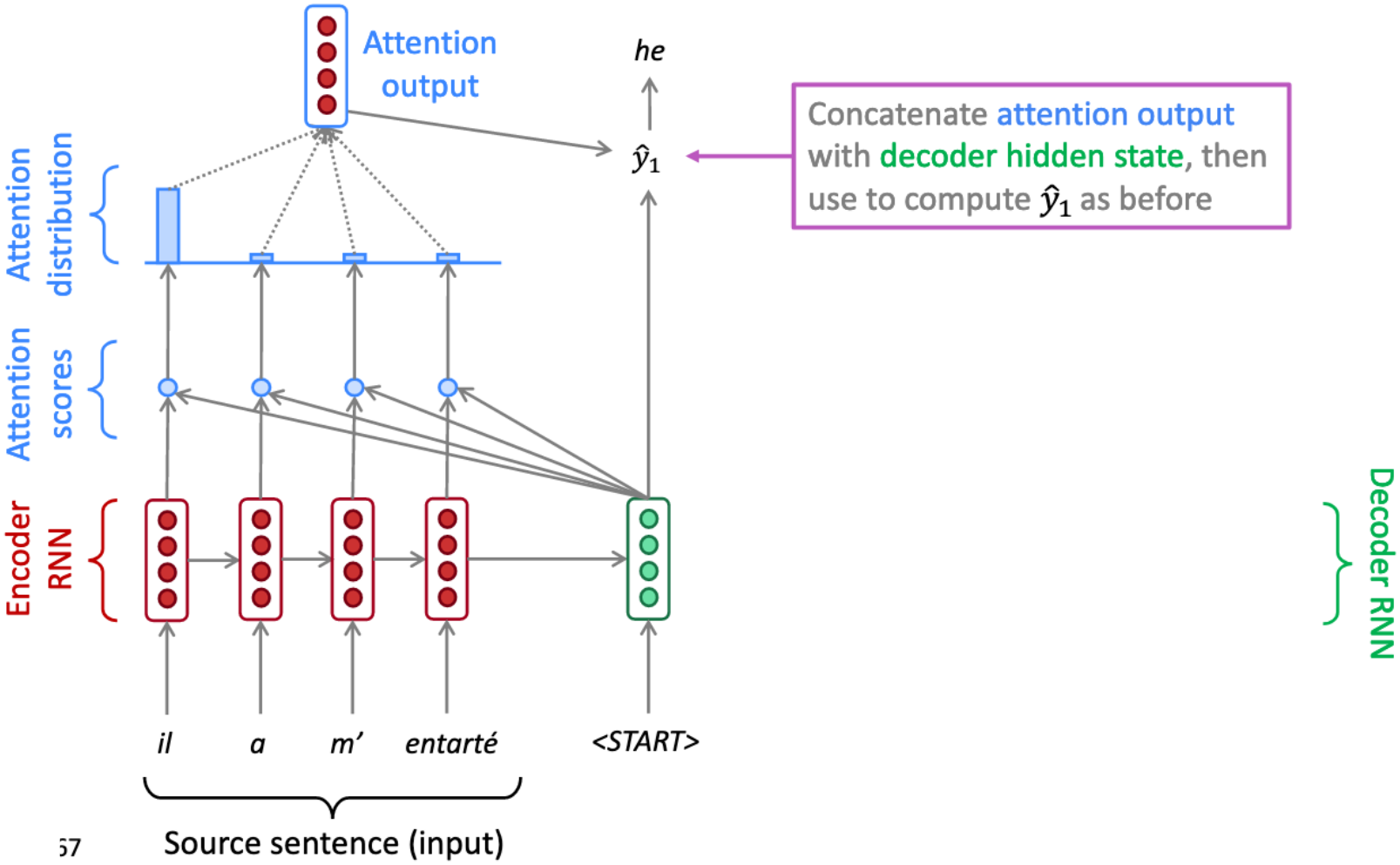


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

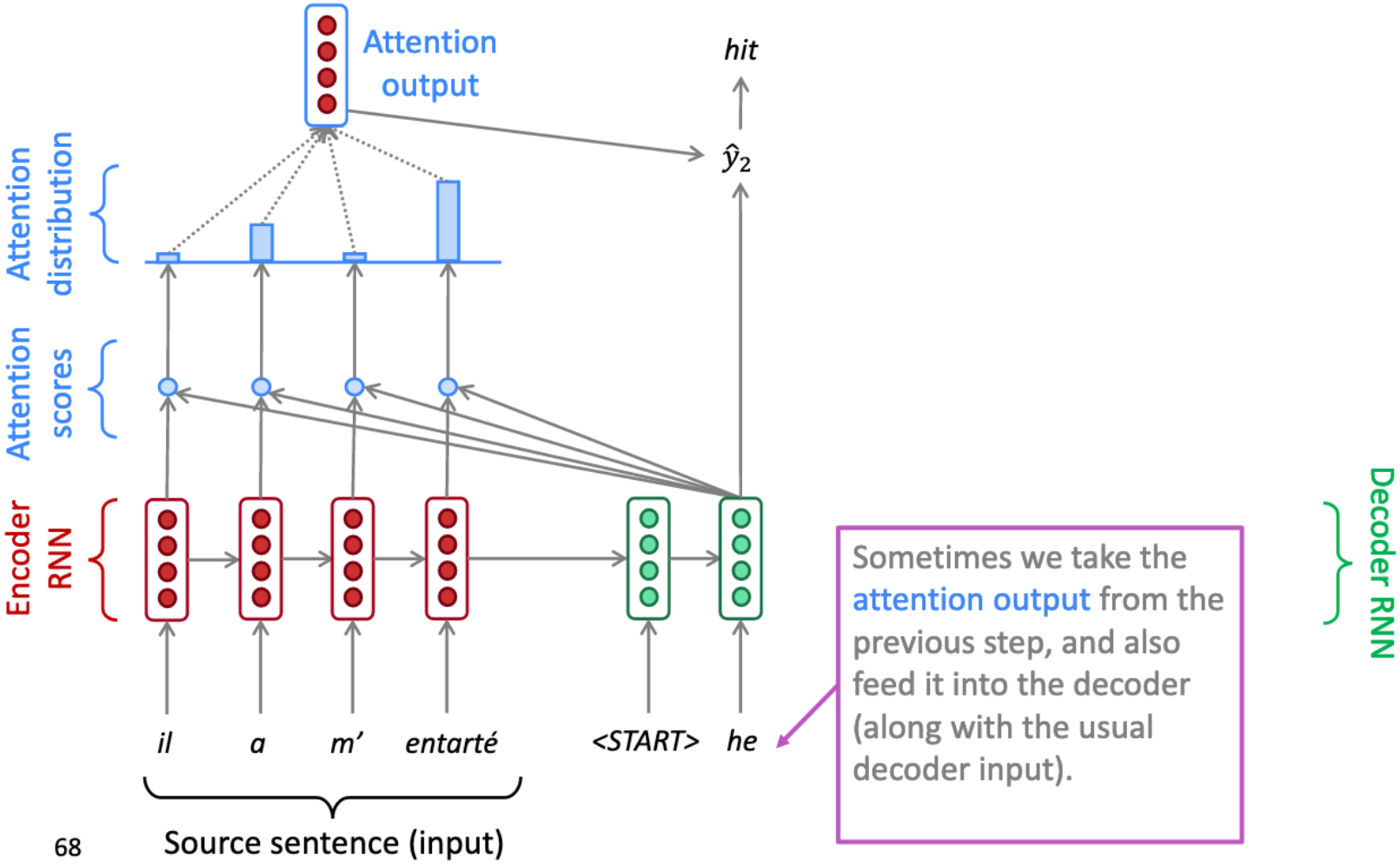
The attention output mostly contains information from the hidden states that received high attention.

Decoder RNN

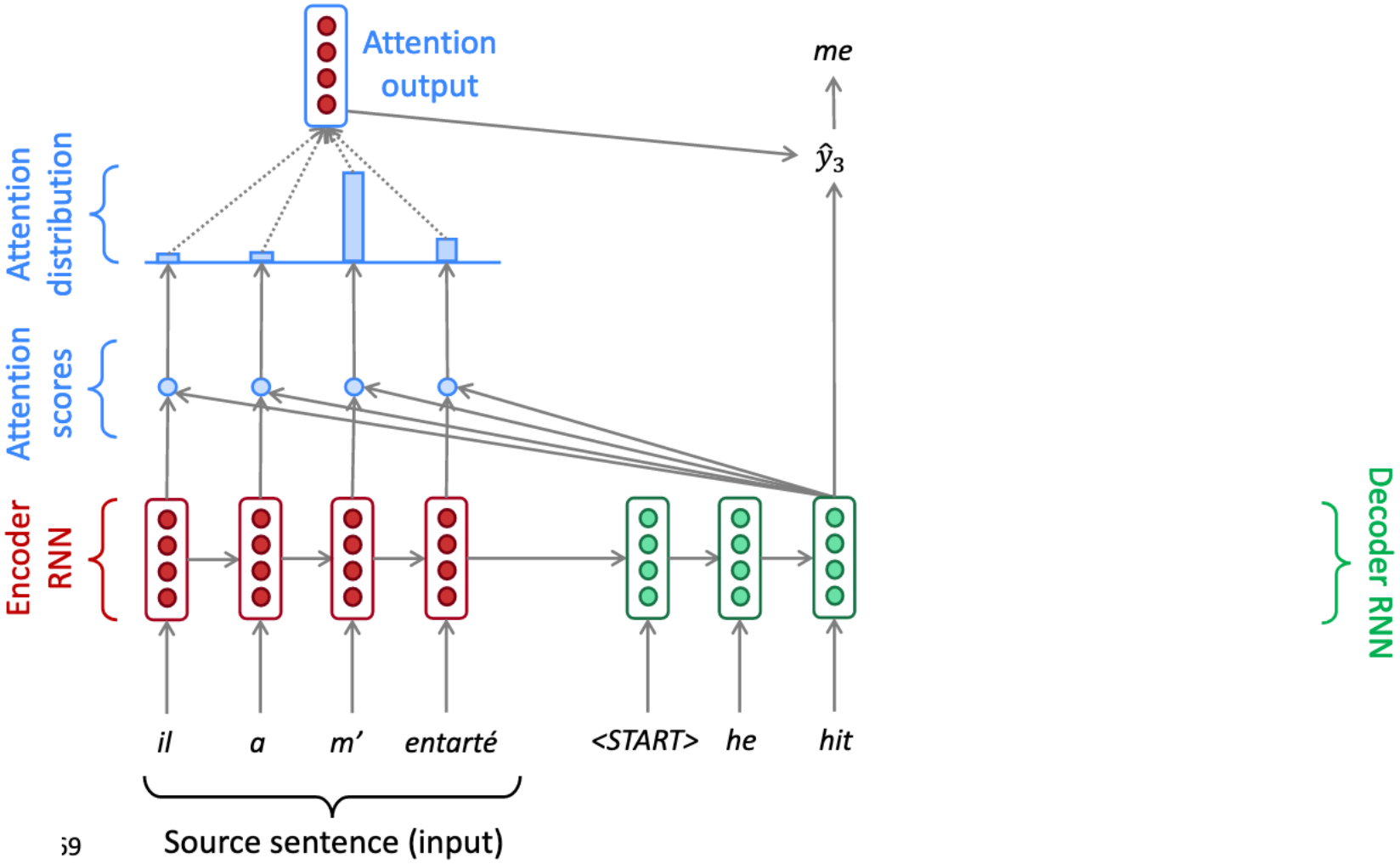
Sequence-to-sequence With Attention



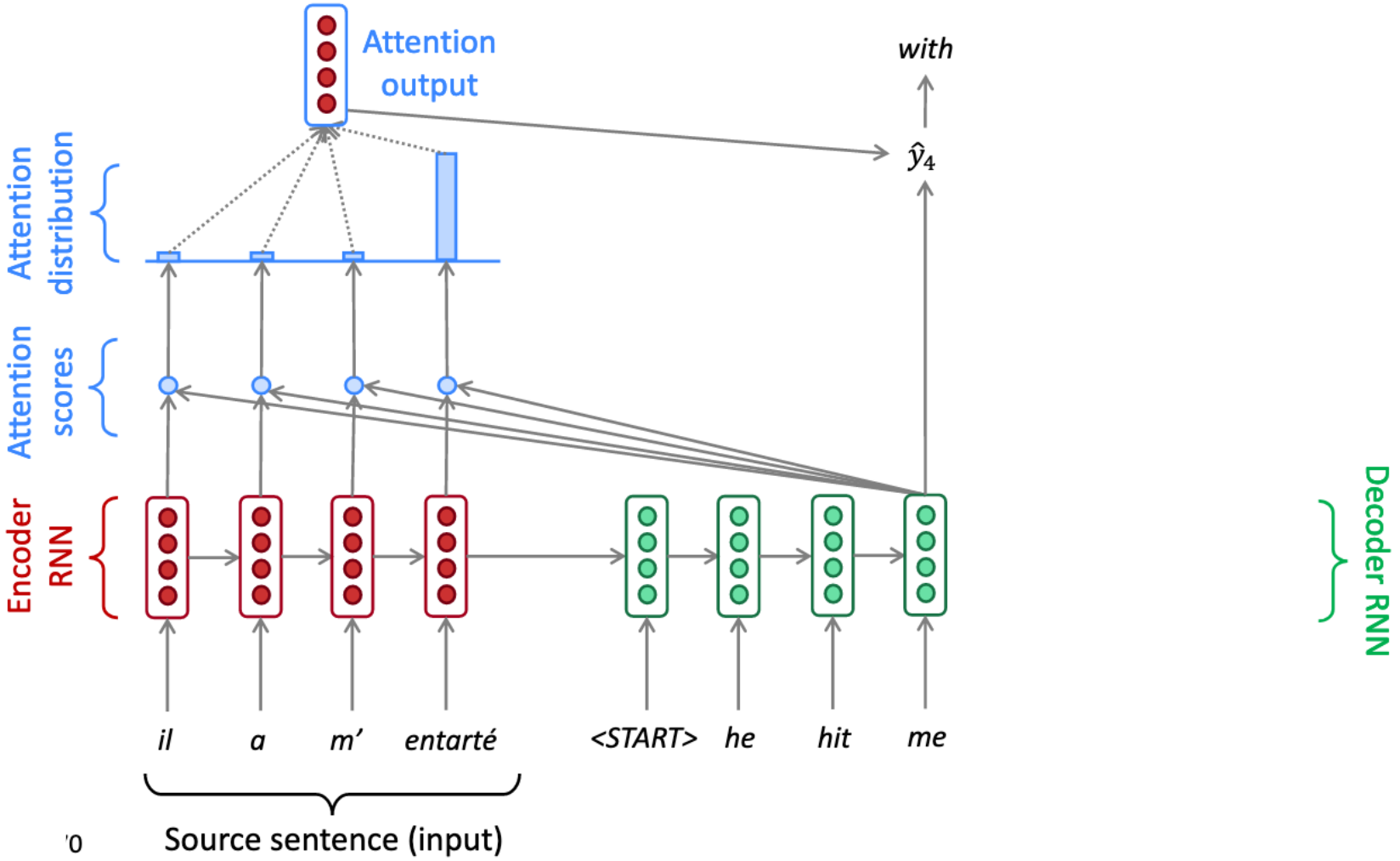
Sequence-to-sequence With Attention



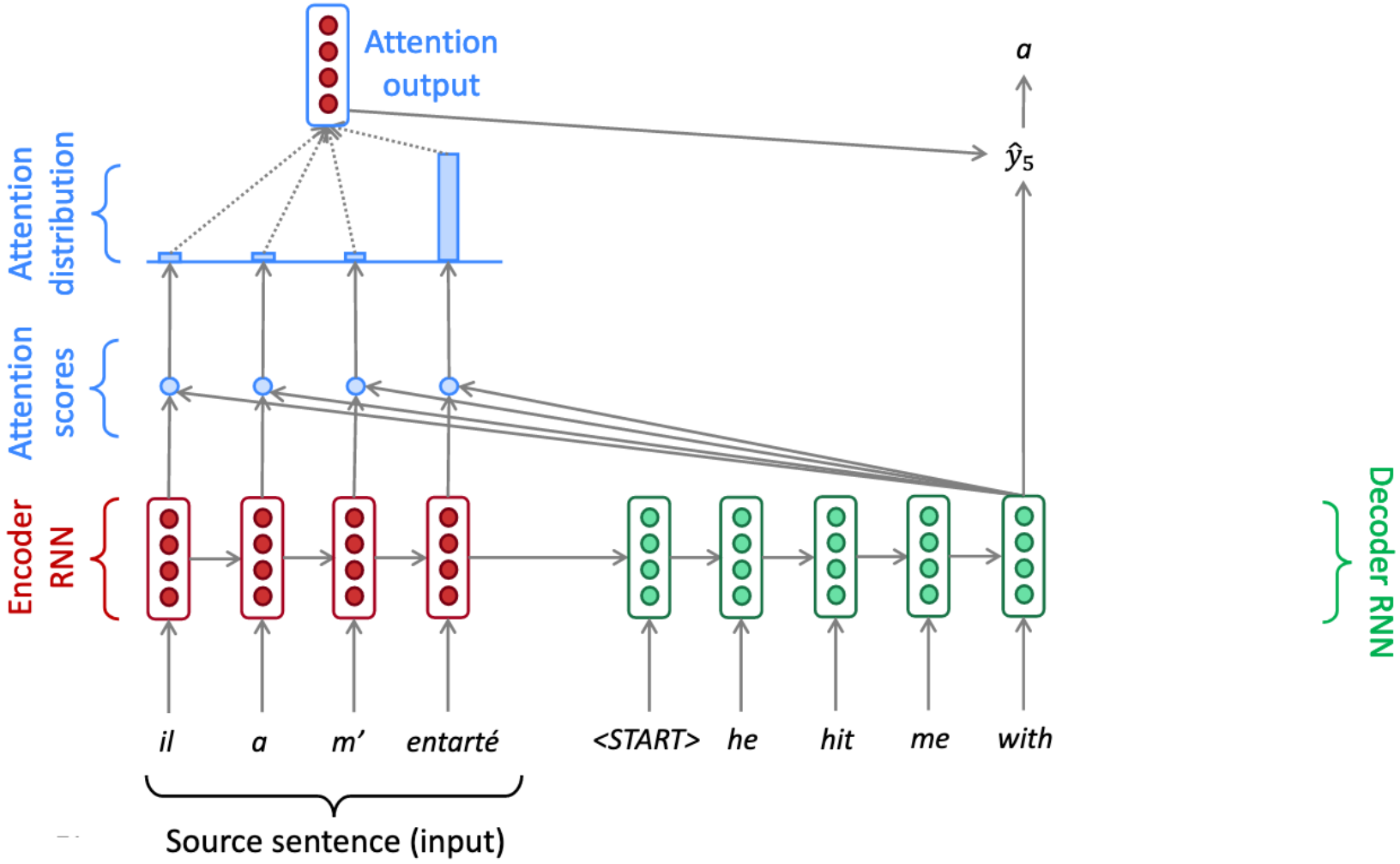
Sequence-to-sequence With Attention



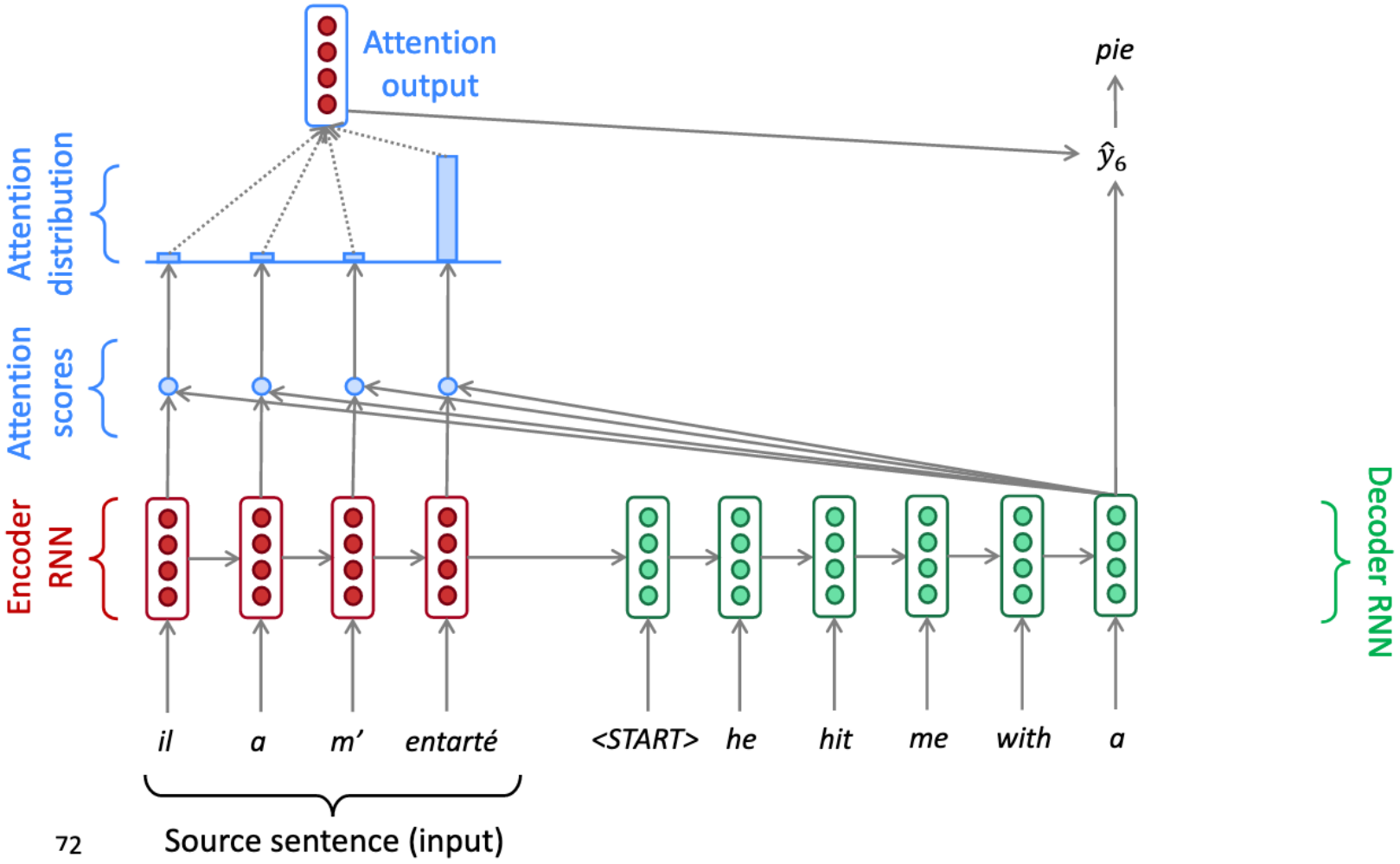
Sequence-to-sequence With Attention



Sequence-to-sequence With Attention



Sequence-to-sequence With Attention



Attention In Equations

We have encoder hidden states $H = h_1, \dots, h_N \in \mathbb{R}^h$

On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$

We get the attention scores e^t for this step:

$$e^t = [s_t h_1, \dots, s_t\, h_N] \in \mathbb{R}^N$$

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

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

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

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

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

$$[a_t; s_t] \in \mathbb{R}^{2h}$$

Benefit Of Attention

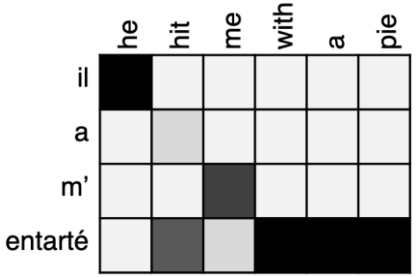
Attention significantly improves NMT performance
- It's very useful to allow decoder to focus on certain parts of the source

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 can use it in many architectures and problems



Some Attention Variants

We need to compute the attention scores:

$$e^t = [s_t h_1, \ldots, s_t\, h_N] \in \mathbb{R}^N$$

Basic dot-product attention: $e_i^t = s_t h_i \in \mathbb{R}$

- Note: this assumes the same dimension for s_t and h_i
- This is the version we mentioned earlier

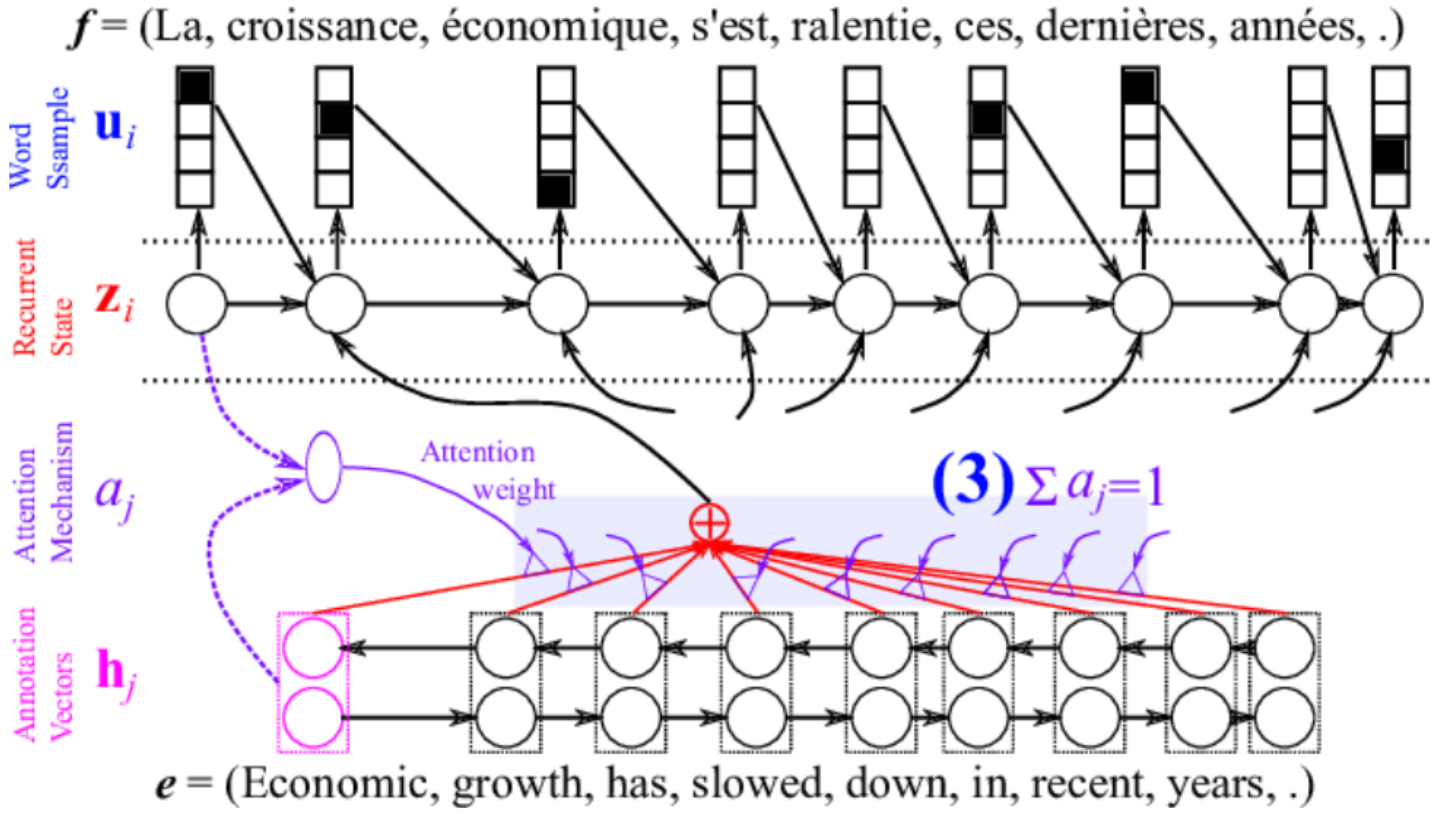
Multiplicative attention: $e_i^t = s_t W h_i \in \mathbb{R}$

- Where $W \in \mathbb{R}^{d_2 \times d_1}$ is a **learnable** weight matrix

Addictive attention: $e_i^t = v \tanh(W_1 s_t + W_2 h_i) \in \mathbb{R}$

- Where $W_1 \in \mathbb{R}^{d_3 \times d_1}$, $W_2 \in \mathbb{R}^{d_3 \times d_2}$, and $v \in \mathbb{R}^{d_3}$ are learnable weight matrices and vectors

Bidirectional Encoding



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

Has AI surpassed humans at translation? Not even close!

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

So Is Machine Translation Solved?

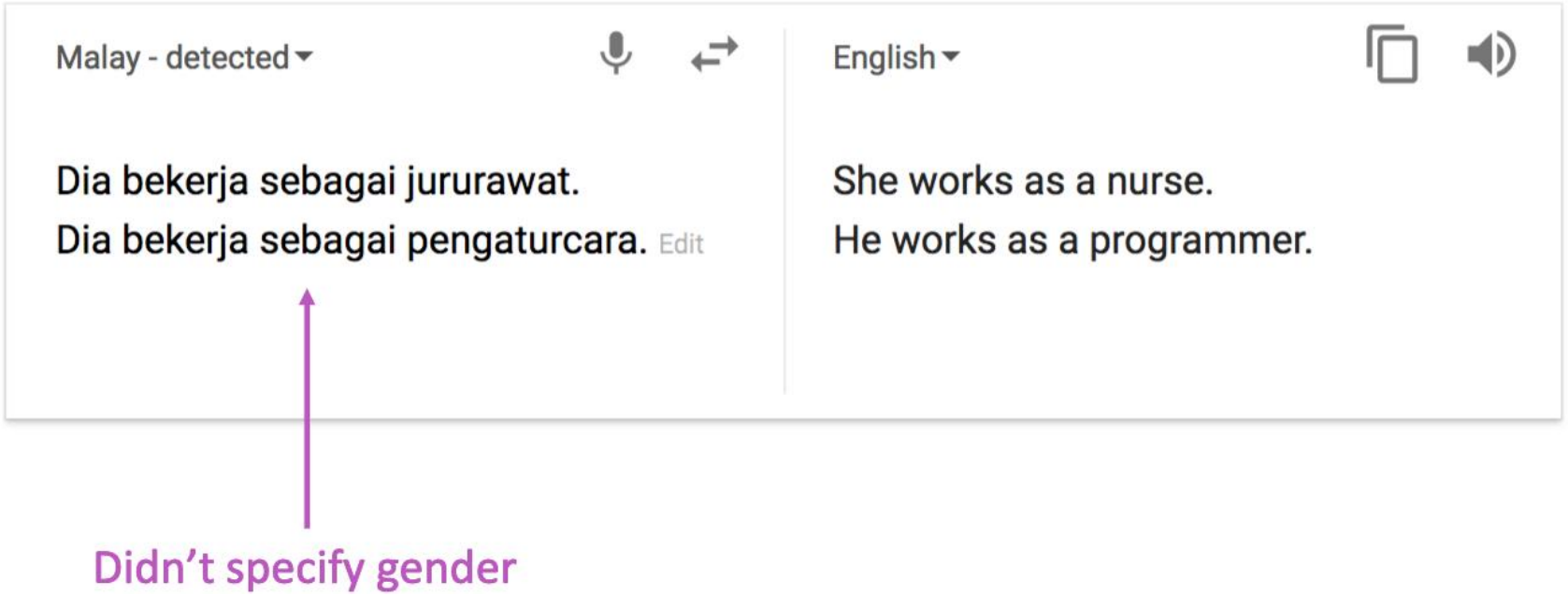
Nope!

Using common sense is still hard

So Is Machine Translation Solved?

Nope!

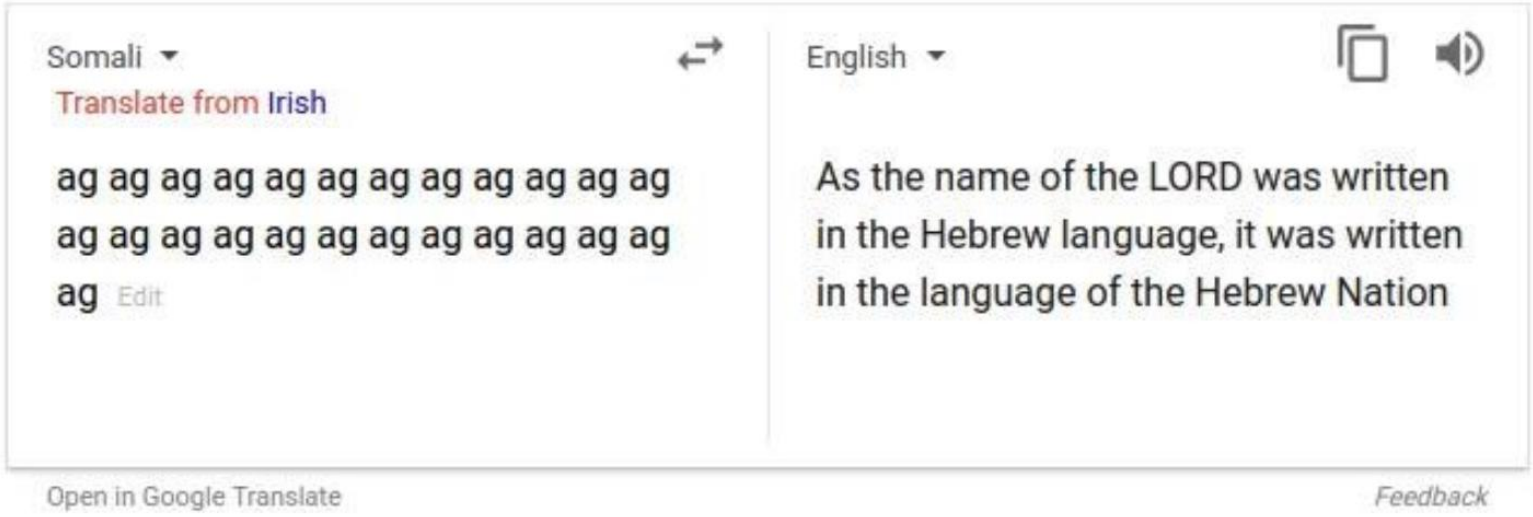
NMT picks up biases in training data



So Is Machine Translation Solved?

Nope!

Uninterpretable systems do strange things



Out-of-vocabulary Words

At translation time, we regularly encounter novel words:

- names: Barack Obama
- morph. complex words: Hand|gepäck|gebühr('carry-on bag fee')
- numbers, URLs etc.

Solutions:

- copy unknown words [Jean et al., 2015, Luong et al., 2015b, Gülçehre et al., 2016] → works for names (if alphabet is shared), and 1-to-1 aligned words
- use subword units (characters or others) for input/output vocabulary

  →model can learn translation of seen words on subword level

  →model can translate unseen words if translation is transparent (following the target language's grammar, syntax and idiom)

Transparent Translations

Some translations are semantically/phonologically transparent

→no memorization needed; can be translated via subword units

Morphologically complex words (e.g. compounds):

- solar system (English)
- Sonnen|system (German)
- Nap|rendszer (Hungarian)

Named entities:

- Barack Obama (English; German)
- バラク・オバマ(ba-ra-ku o-ba-ma) (Japanese)

Cognates and loanwords:

- claustrophobia (English)
- Klaustrophobie (German)

Many rare/unseen words belong to one of these categories

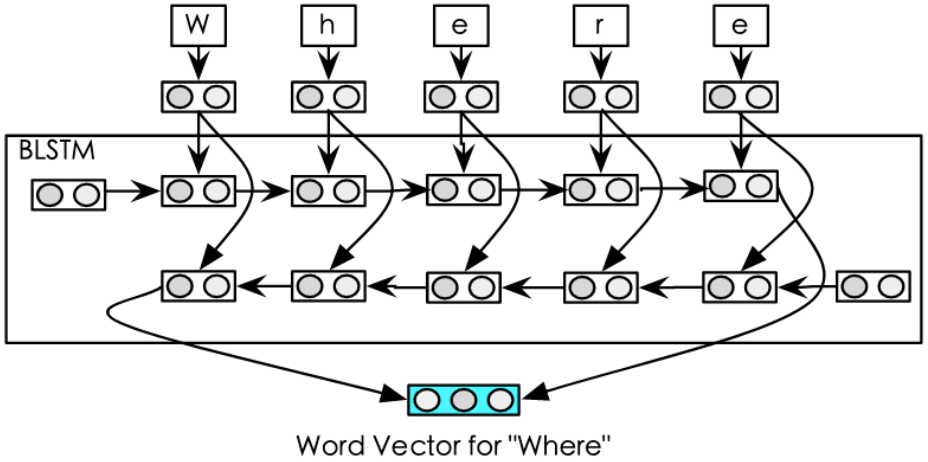
Subword Neural Machine Translation

Flat representation (Sennrich et al., 2015b, Chung et al., 2016)

- sentence is a sequence of subword units

Hierarchical representation (Ling et al., 2015, Luong and Manning, 2016)

- sentence is a sequence of words
- words are a sequence of subword units



open question: should attention be on level of words or subwords?

Subword Neural Machine Translation

Choice of subword unit

- character-level: small vocabulary, long sequences
- morphemes (?): hard to control vocabulary size
- hybrid choice: shortlist of words, subwords for rare words
- variable-length character n-grams: byte-pair encoding (BPE)

Which subword segmentation is best choice in terms of efficiency and effectiveness?

Byte Pair Encoding For Word Segmentation

Word segmentation with byte-pair encoding [Sennrich et al., 2015b]

- actually a merge algorithm, starting from characters
- iteratively replace most frequent pair of symbols (’A’,’B’) with ’AB’
- apply on dictionary, not on full text (for efficiency)
- output vocabulary: original vocabulary + one symbol per merge

Byte Pair Encoding For Word Segmentation

```
'l o w </w>' : 5
'l o w e r </w>' : 2
'n e w e s t </w>' : 6
'w i d e s t </w>' : 3
```

```
                                ('e', 's') : 9
'l o w </w>' : 5
'l o w e r </w>' : 2
'n e w es t </w>' : 6
'w i d es t </w>' : 3
```

```
                                ('e', 's') : 9
                                ('es', 't') : 9
'l o w </w>' : 5
'l o w e r </w>' : 2
'n e w est </w>' : 6
'w i d est </w>' : 3
```

```
                                ('e', 's') : 9
                                ('es', 't') : 9
                                ('est', '</w>') : 9
'l o w </w>' : 5
'l o w e r </w>' : 2
'n e w est</w>' : 6
'w i d est</w>' : 3
```

```
                                ('e', 's') : 9
                                ('es', 't') : 9
                                ('est', '</w>') : 9
'lo w </w>' : 5
'lo w e r </w>' : 2
'n e w est</w>' : 6
'w i d est</w>' : 3
```

```
                                ('e', 's') : 9
                                ('es', 't') : 9
                                ('est', '</w>') : 9
                                ('l', 'o') : 7
'low </w>' : 5
'low e r </w>' : 2
'n e w est</w>' : 6
'w i d est</w>' : 3
```

```
                                ('e', 's') : 9
                                ('es', 't') : 9
                                ('est', '</w>') : 9
                                ('l', 'o') : 7
                                ('lo', 'w') : 7
                                ...
```

Why Byte Pair Encoding (BPE)?

Good trade-off between vocabulary size and text length.

segmentation	# tokens	# types	# UNK
none	100 m	1 750 000	1079
characters	550 m	3000	0
BPE (60k operations)	112 m	63 000	0

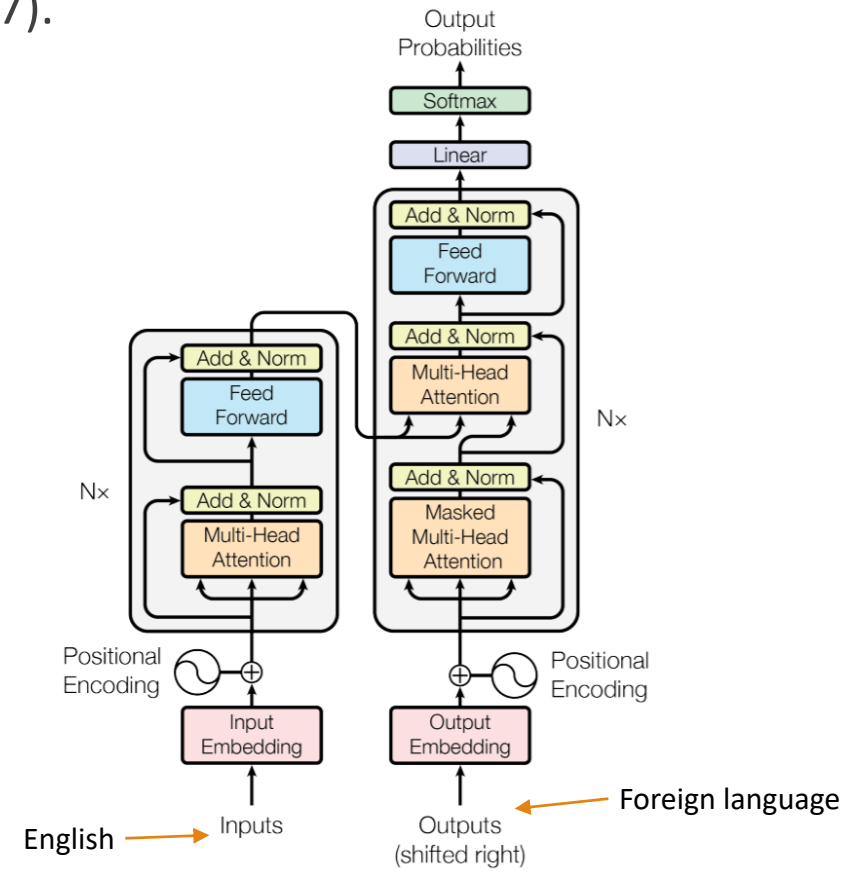
learned operations can be applied to unknown words→ open-vocabulary

Byte Pair Encoding For Word Segmentation

'l o w e s t </w>'	('e', 's') : 9 ('es', 't') : 9 ('est', '</w>') : 9 ('l', 'o') : 7 ('lo', 'w') : 7
'l o w est</w>'	('e', 's') : 9 ('es', 't') : 9 ('est', '</w>') : 9 ('l', 'o') : 7 ('lo', 'w') : 7
'l o w es t </w>'	('e', 's') : 9 ('es', 't') : 9 ('est', '</w>') : 9 ('l', 'o') : 7 ('lo', 'w') : 7
'lo w est</w>'	('e', 's') : 9 ('es', 't') : 9 ('est', '</w>') : 9 ('l', 'o') : 7 ('lo', 'w') : 7
'l o w est </w>'	('e', 's') : 9 ('es', 't') : 9 ('est', '</w>') : 9 ('l', 'o') : 7 ('lo', 'w') : 7
'low est</w>'	('e', 's') : 9 ('es', 't') : 9 ('est', '</w>') : 9 ('l', 'o') : 7 ('lo', 'w') : 7

Transformer Architecture for MT (Recap)

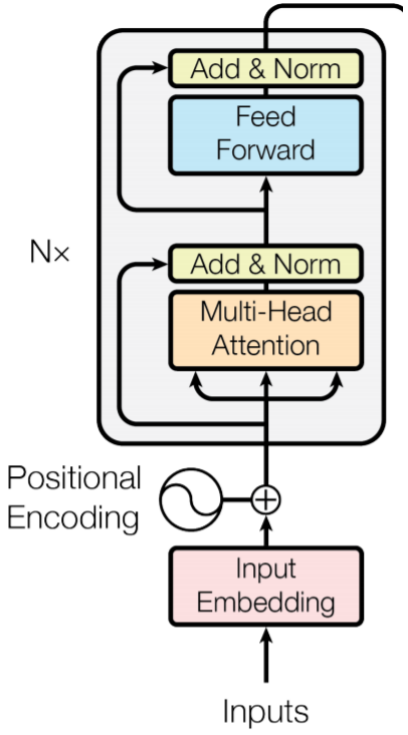
A composition of many multi-head attention operations, originally designed for machine translation with the encoder and decoder networks (Vaswani et al., 2017).



Vaswani et al. Attention Is All You Need. NIPS 2017.

Transformer Architecture for MT (Recap)

Encoder is composed of N layers; each of which has two sublayers (a multi-head attention and feed forward network) with residual connections around them.

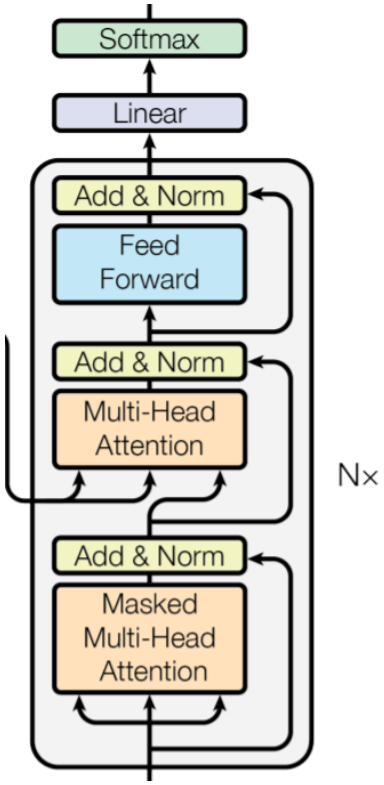



```
x_encoder = word_embeds + pos_embeds
for layer_i in range(N):
    q, k, v = x_encoder, x_encoder, x_encoder
    att = MultiHeadAttention(q, k, v)
    att = LayerNorm(x_encoder + Dropout(att))
    ffw = Feedfw(att)
    x_encoder = LayerNorm(att + dropout(ffw))
```

Vaswani et al. Attention Is All You Need. NIPS 2017.

Transformer Architecture for MT (Recap)

Decoder is designed similarly to Encoder except that it has a new sublayer called Masked Multi-Head Attention to ensure that predictions for position i can only depends on previous positions (for generate a translated sentence in Machine Translation).

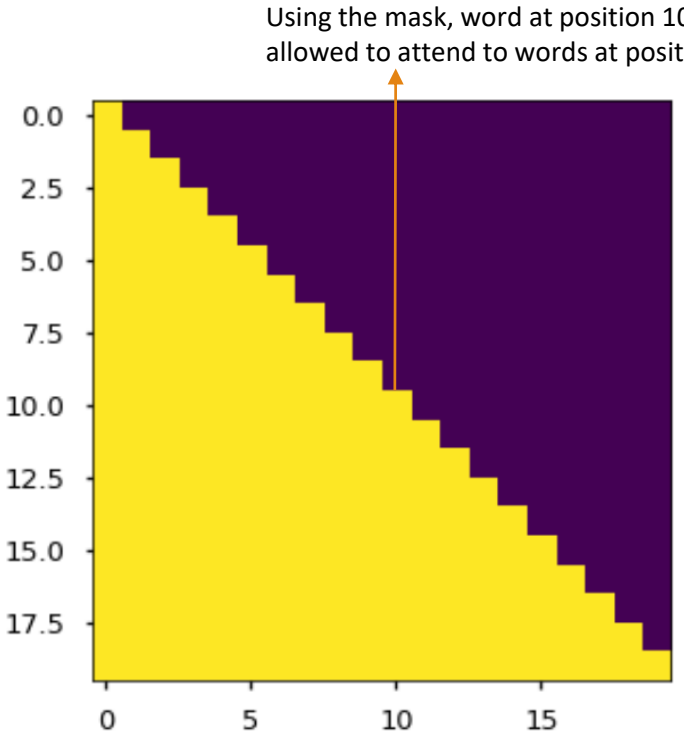


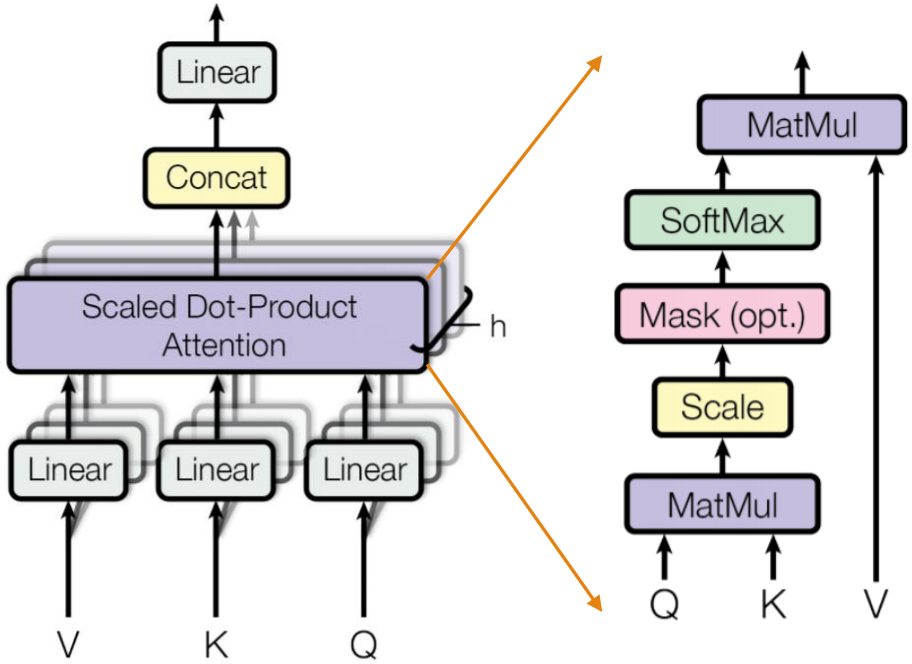

```
x_decoder = word_embeds + pos_embeds

for layer_i in range(N):
  q, k, v = x_decoder, x_decoder, x_decoder
  att = MaskedMultiHeadAttention(q, k, v)
  att = LayerNorm(x_decoder + Dropout(att))

  q = att
  k, v = x_encoder, x_encoder

  att = MultiheadAttention(q, k, v)
  ffw = Feedfw(att)
  x_decoder = LayerNorm(att + dropout(ffw))

logits = Feedfw(x_decoder)
preds = softmax(logits)
```

Vaswani et al. Attention Is All You Need. NIPS 2017.

Transformer Architecture for MT (Recap)

The **masking** schema in Transformer Decoder (multiplied to the attention matrix directly):





Masked Multi-Head Attention

http://nlp.seas.harvard.edu/2018/04/03/attention.html

Transformer Architecture for MT (Recap)

Evaluation

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