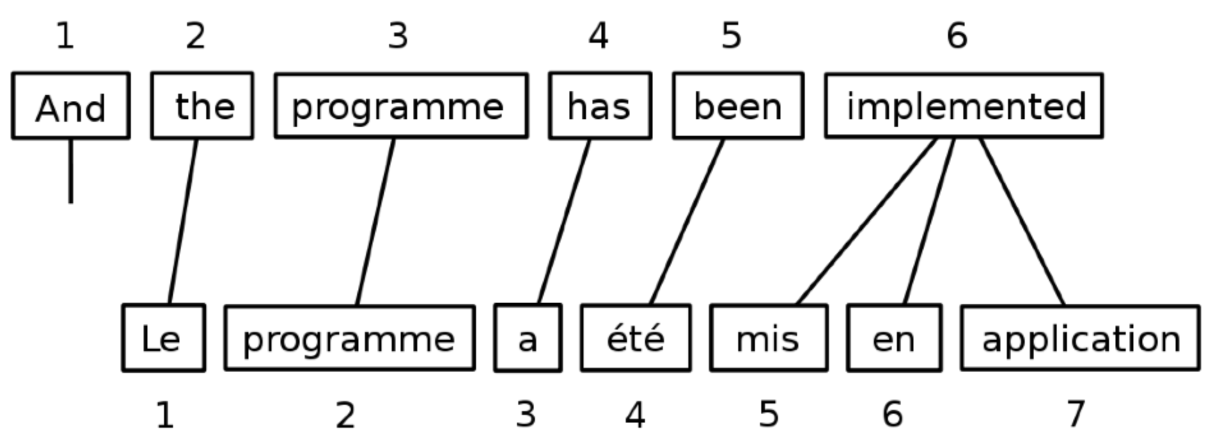
CSEP 517
Natural Language Processing

Neural Machine Translation

Luke Zettlemoyer

(Slides adapted from Karthik Narasimhan, Greg Durrett, Chris Manning, Dan Jurafsky)

Last time



- Statistical MT
  - Word-based
  - Phrase-based
  - Syntactic

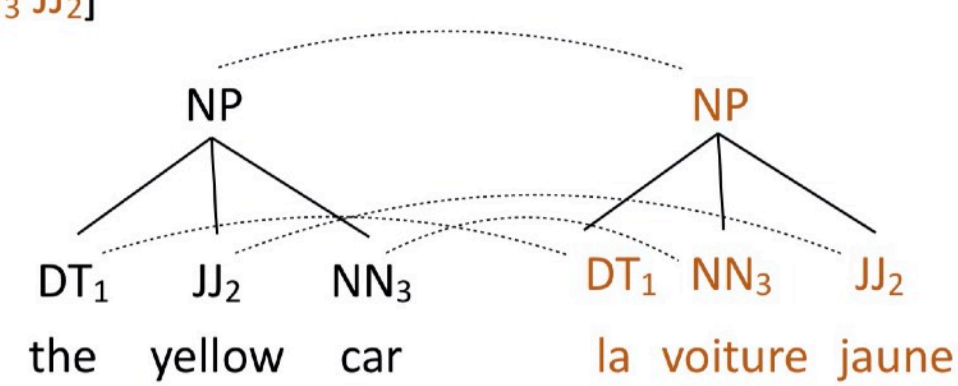
$NP \rightarrow [DT_1\ JJ_2\ NN_3;\ DT_1\ NN_3\ JJ_2]$

$DT \rightarrow$ [the, la]

$DT \rightarrow$ [the, le]

$NN \rightarrow$ [car, voiture]

$JJ \rightarrow$ [yellow, jaune]



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**

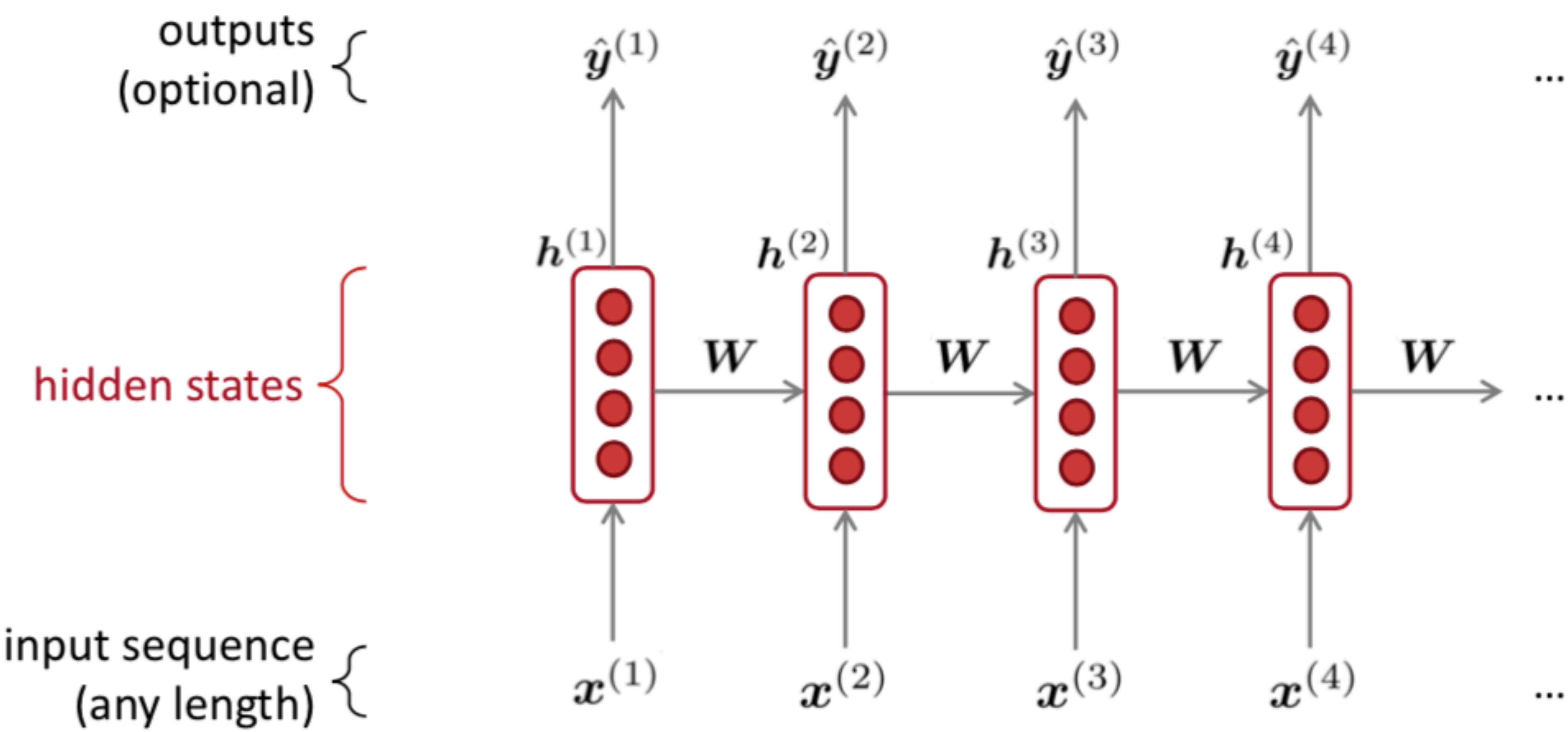
- **2014**: First seq2seq paper published
- **2016**: Google Translate switches from SMT to NMT
- This is amazing!
  - **SMT** systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by a handful of engineers in a few months

Neural Machine Translation

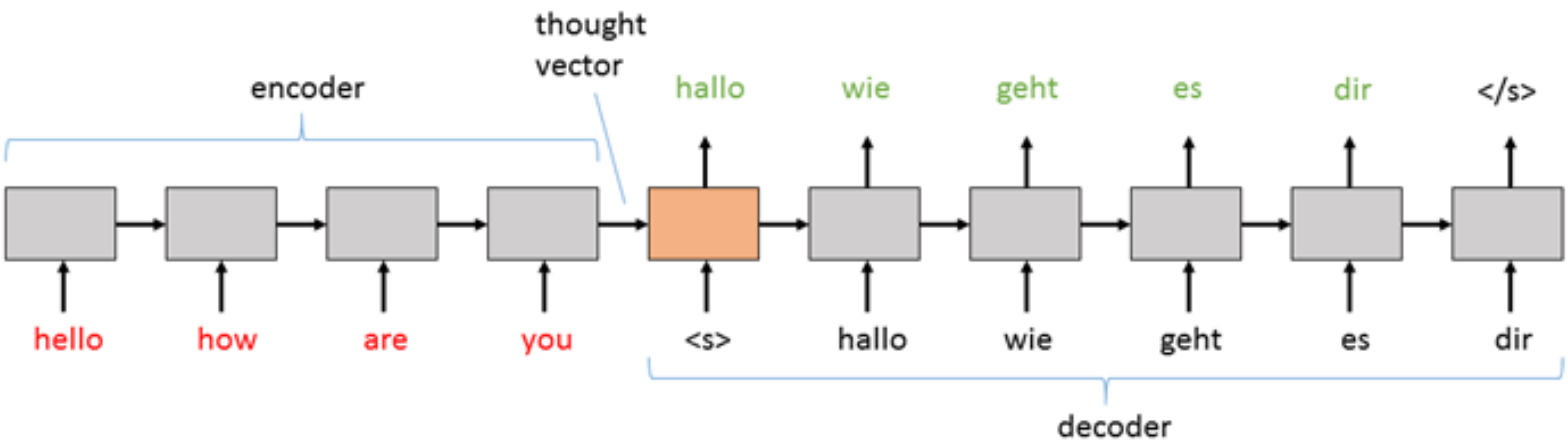
- A **single neural network** is used to translate from source to target
- Architecture: Encoder-Decoder
  - Two main components:
    - Encoder: Convert source sentence (input) into a vector/matrix
    - Decoder: Convert encoding into a sentence in target language (output)

Recall: RNNs

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^d$$



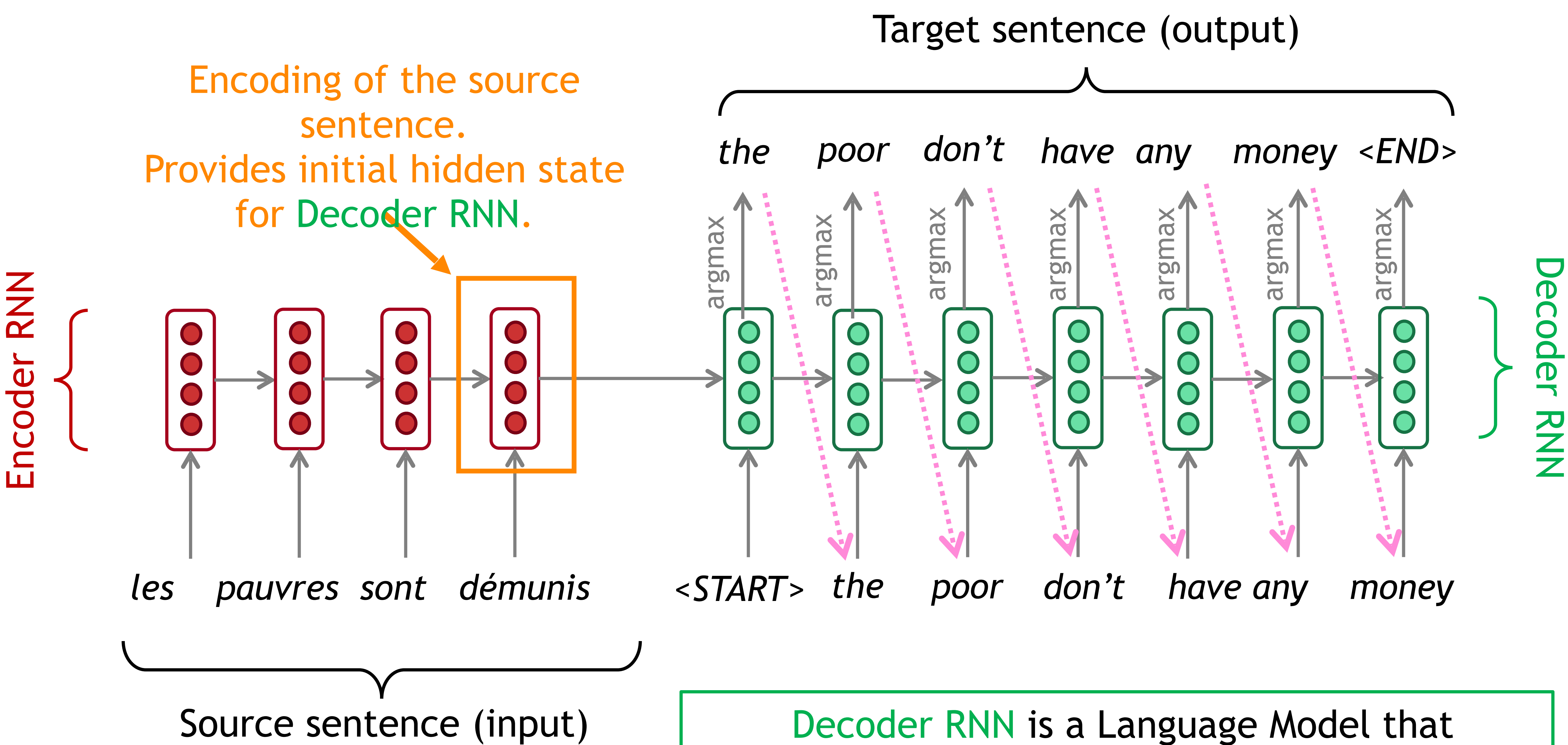
Sequence to Sequence learning (Seq2seq)



- Encode entire input sequence into a single vector **(using an RNN)**
- Decode one word at a time **(again, using an RNN!)**
- Beam search for better inference
- Learning is not trivial! (vanishing/exploding gradients)

(Sutskever et al., 2014)

Neural Machine Translation (NMT)



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

Seq2seq training

- Similar to training a language model!
- Minimize cross-entropy loss:

$$\sum_{t=1}^{T} - \log P(y_t \mid y_1, \ldots, y_{t-1}, x_1, \ldots, x_n)$$

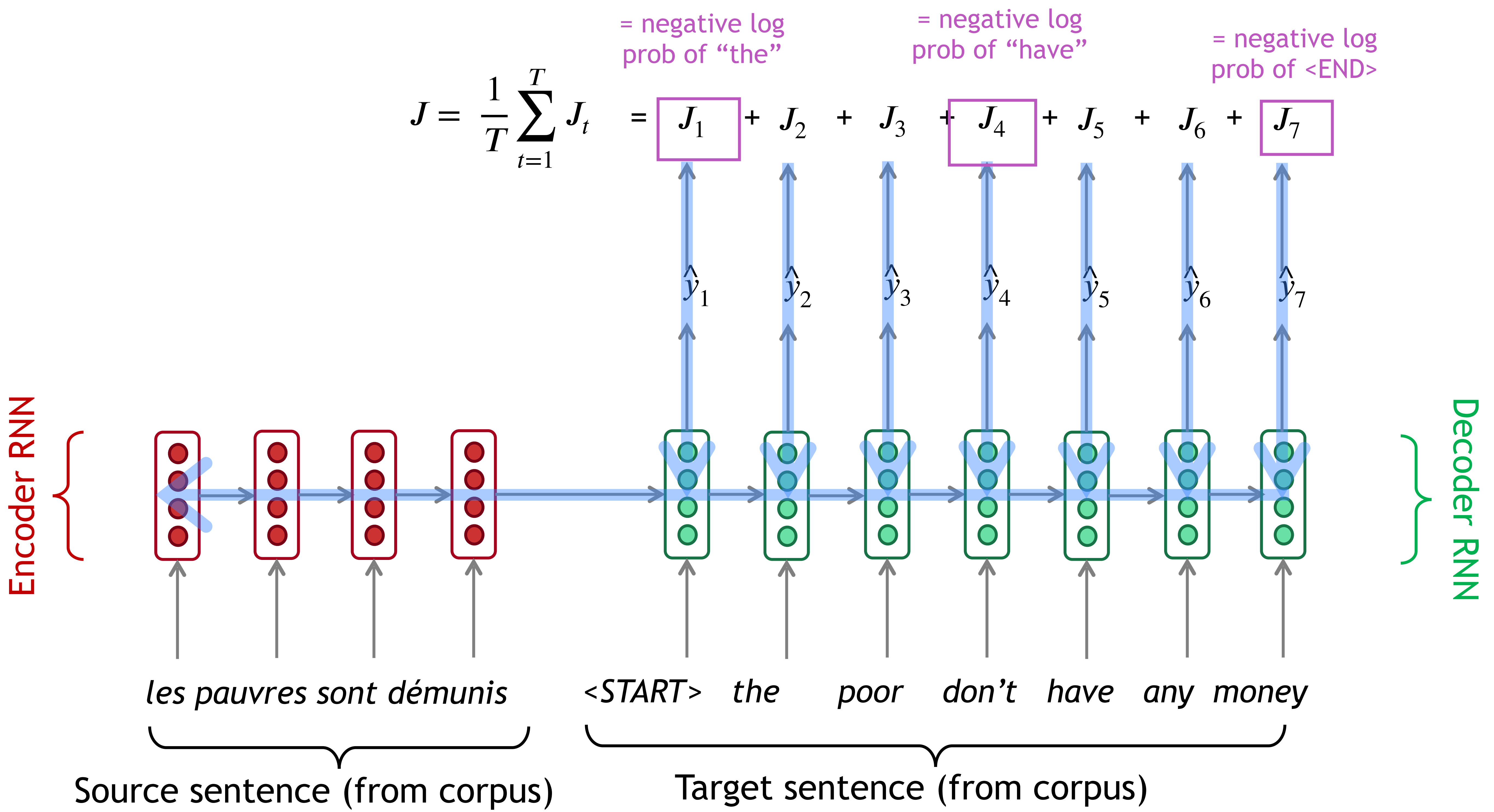
- Back-propagate gradients through *both decoder and encoder*
- Need a really big corpus

36M sentence pairs

Russian: Машинный перевод - это круто!

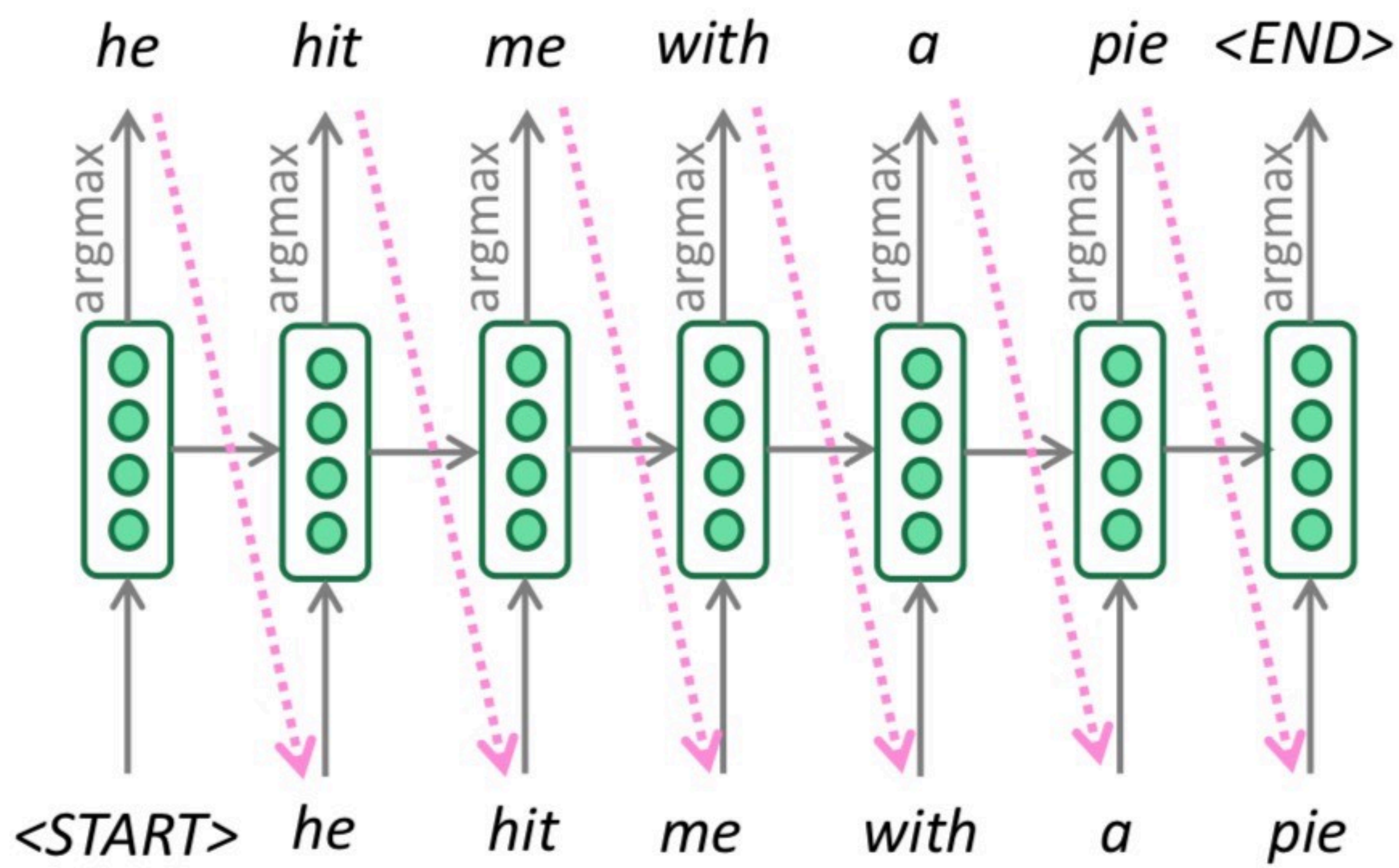
English: Machine translation is cool!

Training a Neural Machine Translation system



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

Greedy decoding



- Compute argmax at every step of decoder to generate word
- What's wrong?

Exhaustive search?

- Find $\arg\max_{y_1,\ldots,y_T} P(y_1, \ldots, y_T | x_1, \ldots, x_n)$

- Requires computing all possible sequences
  - $O(V^T)$ complexity!
  - Too expensive

A middle ground: Beam search

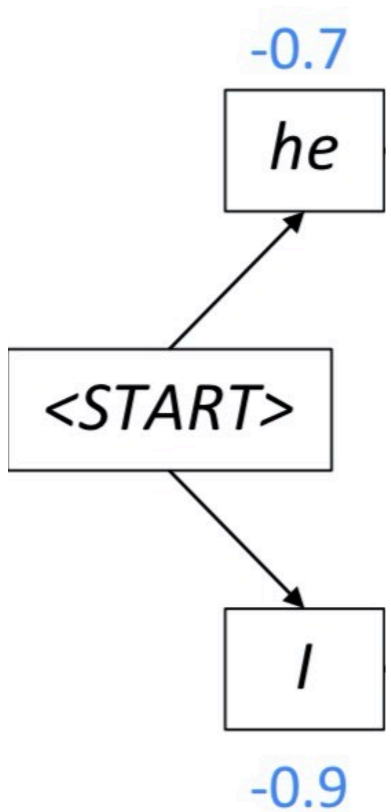
- **Key idea:** At every step, keep track of the k most probable partial translations (hypotheses)
- Score of each hypothesis = log probability

$$\sum_{t=1}^{j} \log P(y_t \,|\, y_1, \ldots, y_{t-1}, x_1, \ldots, x_n)$$

- Not guaranteed to be optimal
- More efficient than exhaustive search

Beam decoding

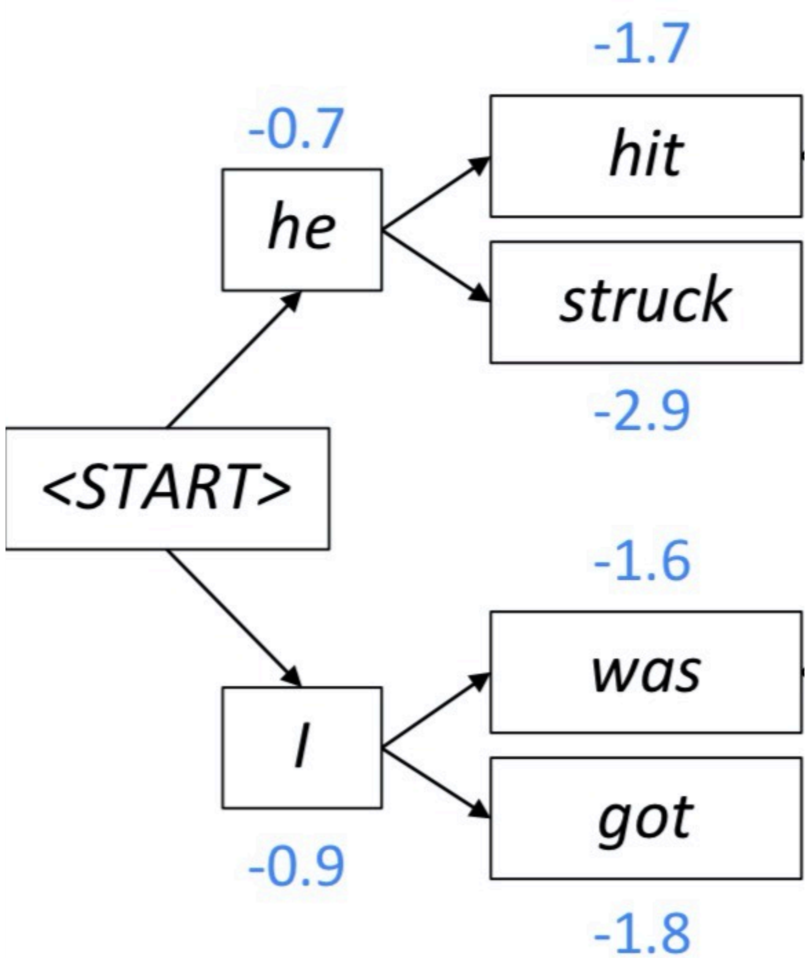
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



(slide credit: Abigail See)

Beam decoding

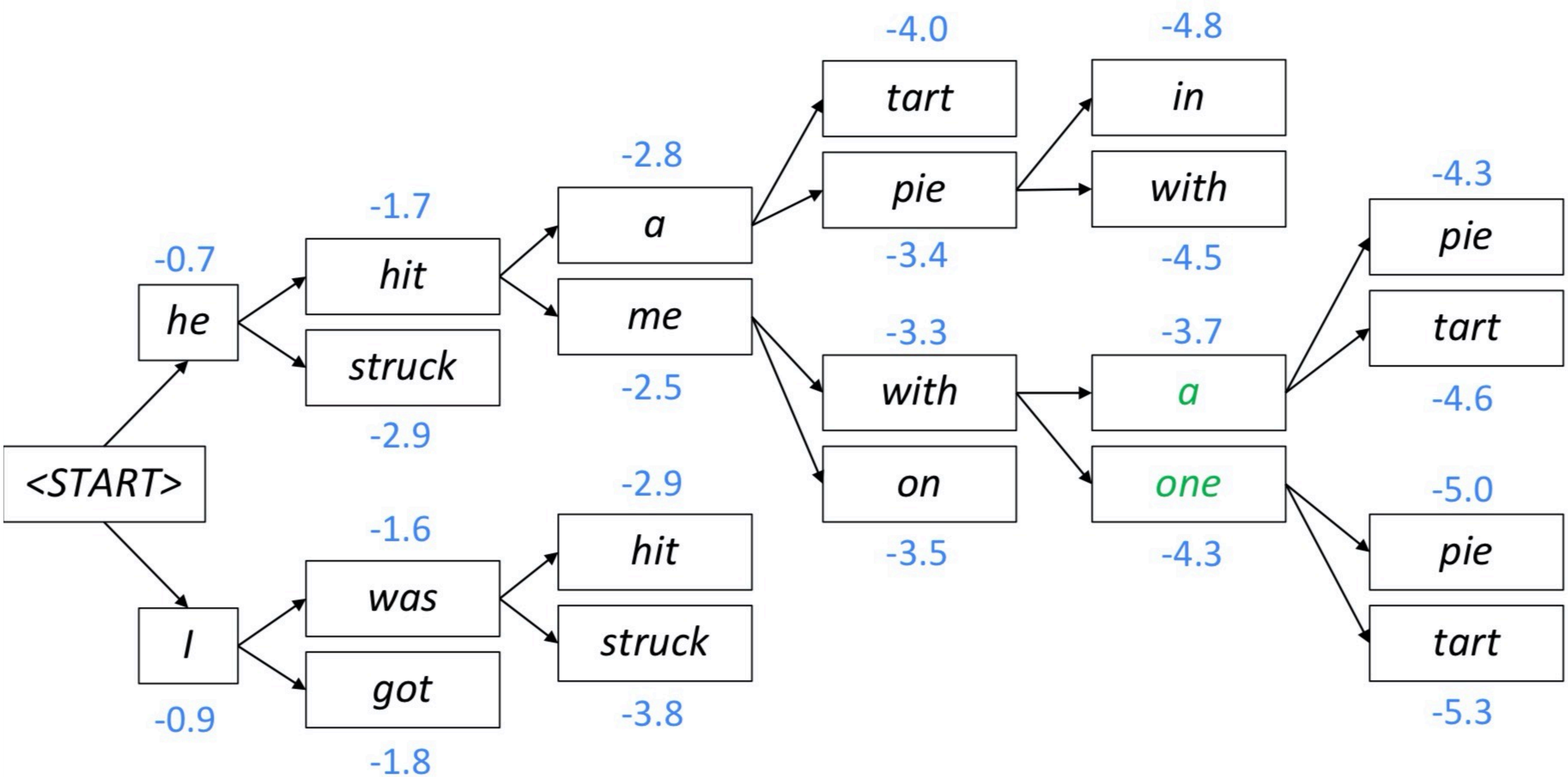
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



(slide credit: Abigail See)

Beam decoding

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



(slide credit: Abigail See)

Beam decoding

- Different hypotheses may produce $\langle e \rangle$ (end) token at different time steps
  - When a hypothesis produces $\langle e \rangle$, stop expanding it and place it aside
- Continue beam search until:
  - All k hypotheses produce $\langle e \rangle$ OR
  - Hit max decoding limit T
- Select top hypotheses using the *normalized* likelihood score

$$\frac{1}{T}\sum_{t=1}^{T}\log P(y_t \mid y_1, \ldots, y_{t-1}, x_1, \ldots, x_n)$$

  - Otherwise shorter hypotheses have higher scores

NMT vs SMT

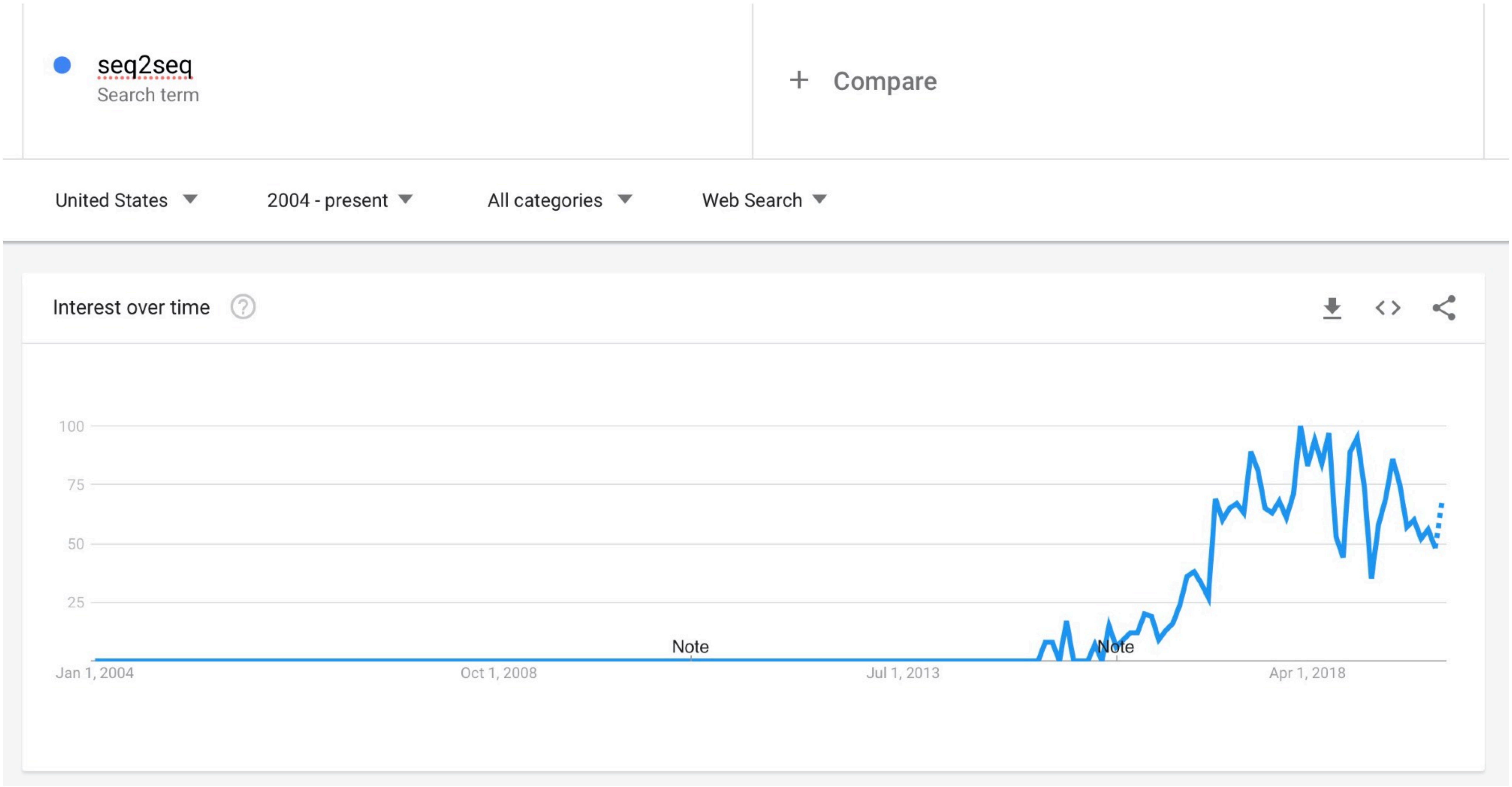
Pros

- Better performance
  - Fluency
  - Longer context
- Single NN optimized end-to-end
- Less engineering
- Works out of the box for many language pairs

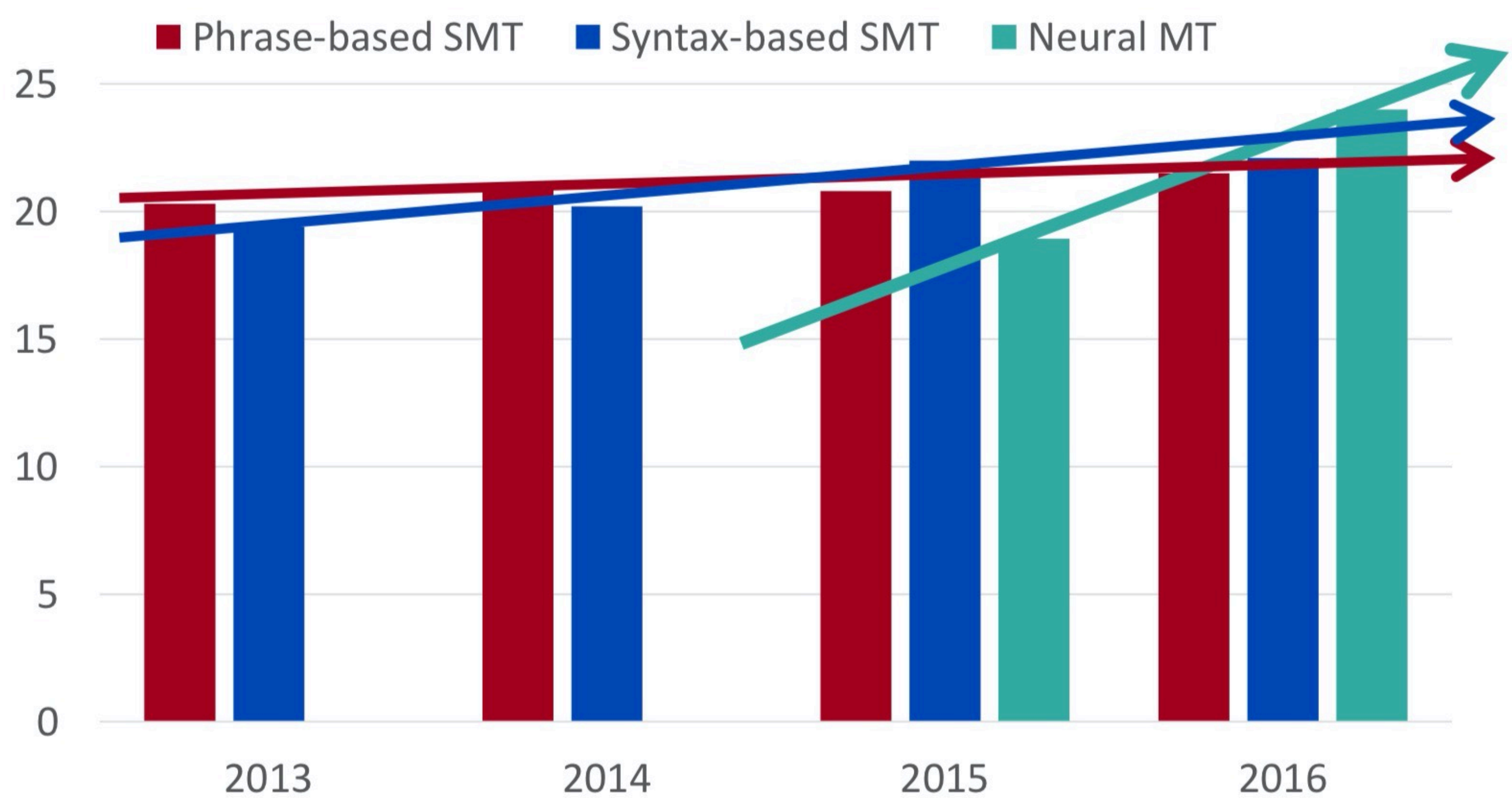
Cons

- Requires more data and compute
- Less interpretable
  - Hard to debug
- Uncontrollable
  - Heavily dependent on data - could lead to unwanted biases
- More parameters

How seq2seq changed the MT landscape



MT Progress

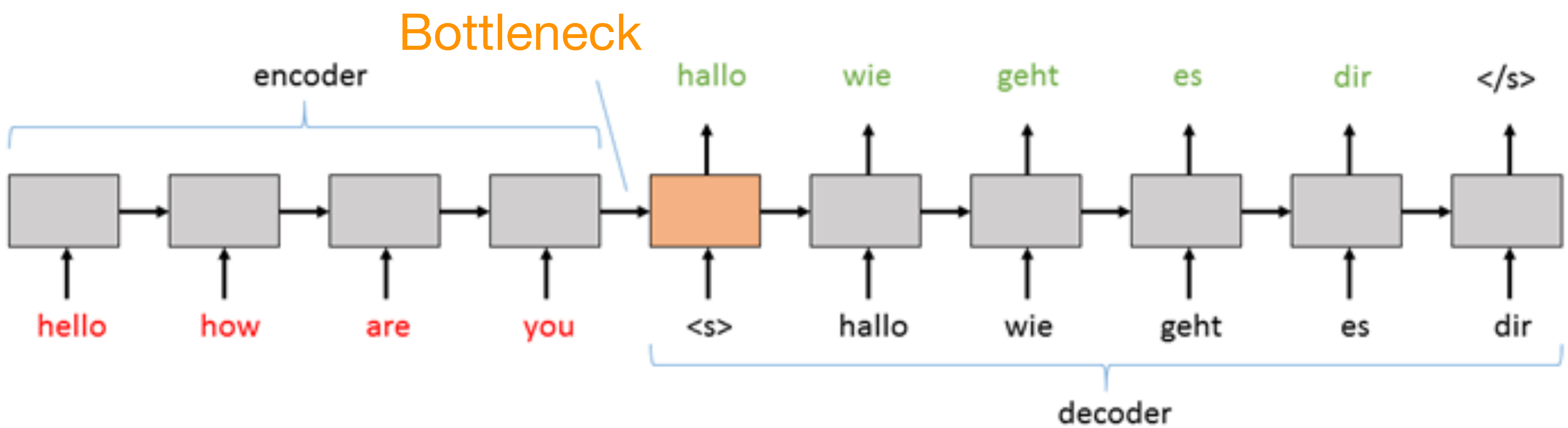


(source: Rico Sennrich)

Versatile seq2seq

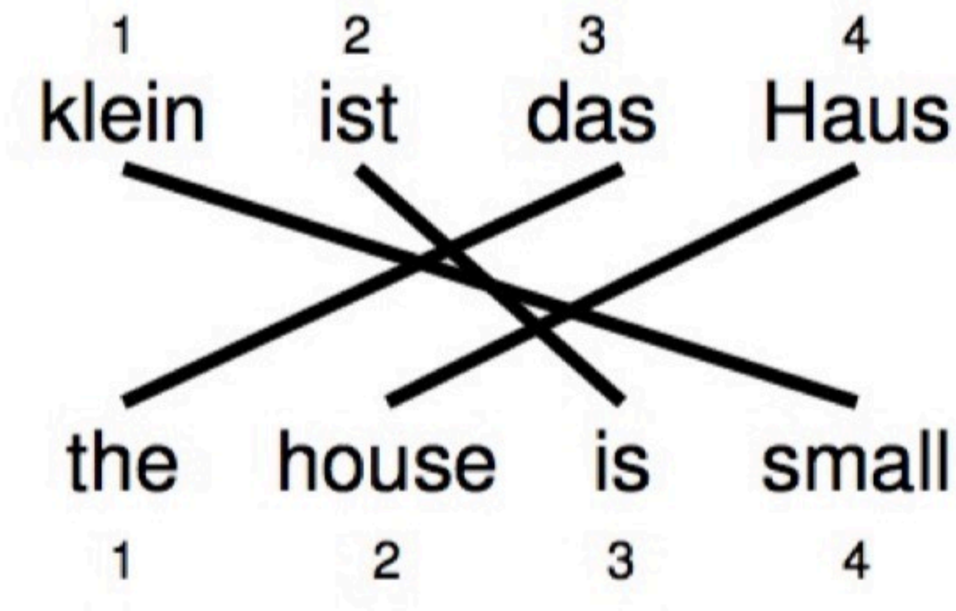
- Seq2seq finds applications in many other tasks!
- Any task where inputs and outputs are sequences of words/characters
  - Summarization (input text → summary)
  - Dialogue (previous utterance → reply)
  - Parsing (sentence → parse tree in sequence form)
  - Question answering (context+question → answer)

Issues with vanilla seq2seq

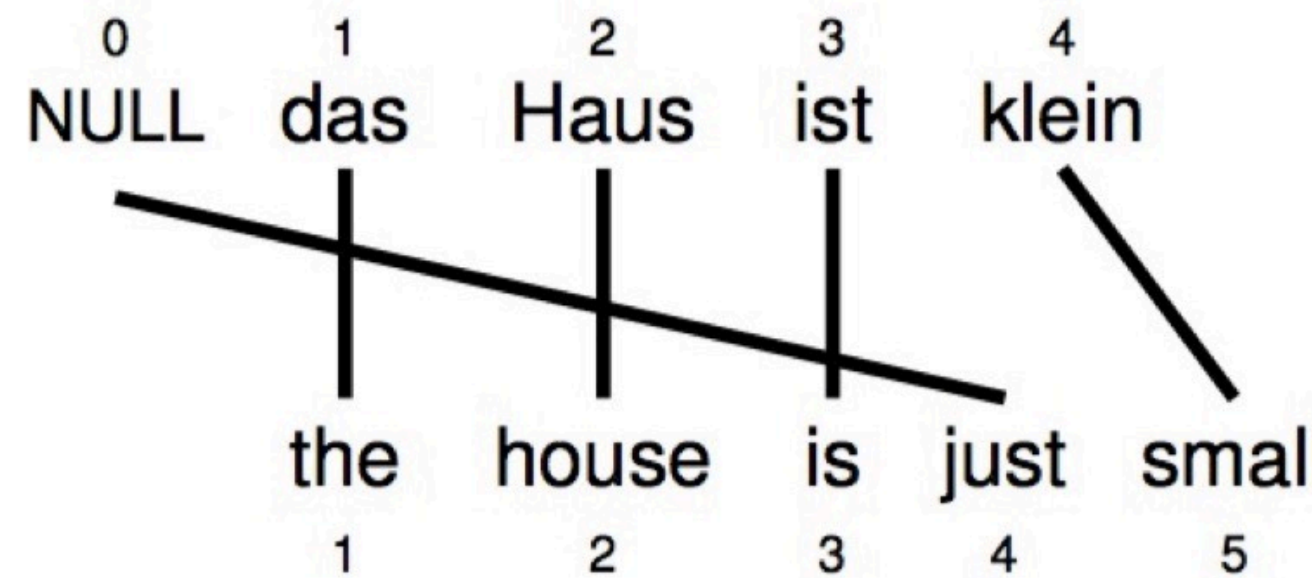


- A single encoding vector, h^{enc}, needs to capture all the information about source sentence
- Longer sequences can lead to vanishing gradients
- Overfitting

Remember alignments?



$$\mathbf{a} = (3, 4, 2, 1)^\top$$

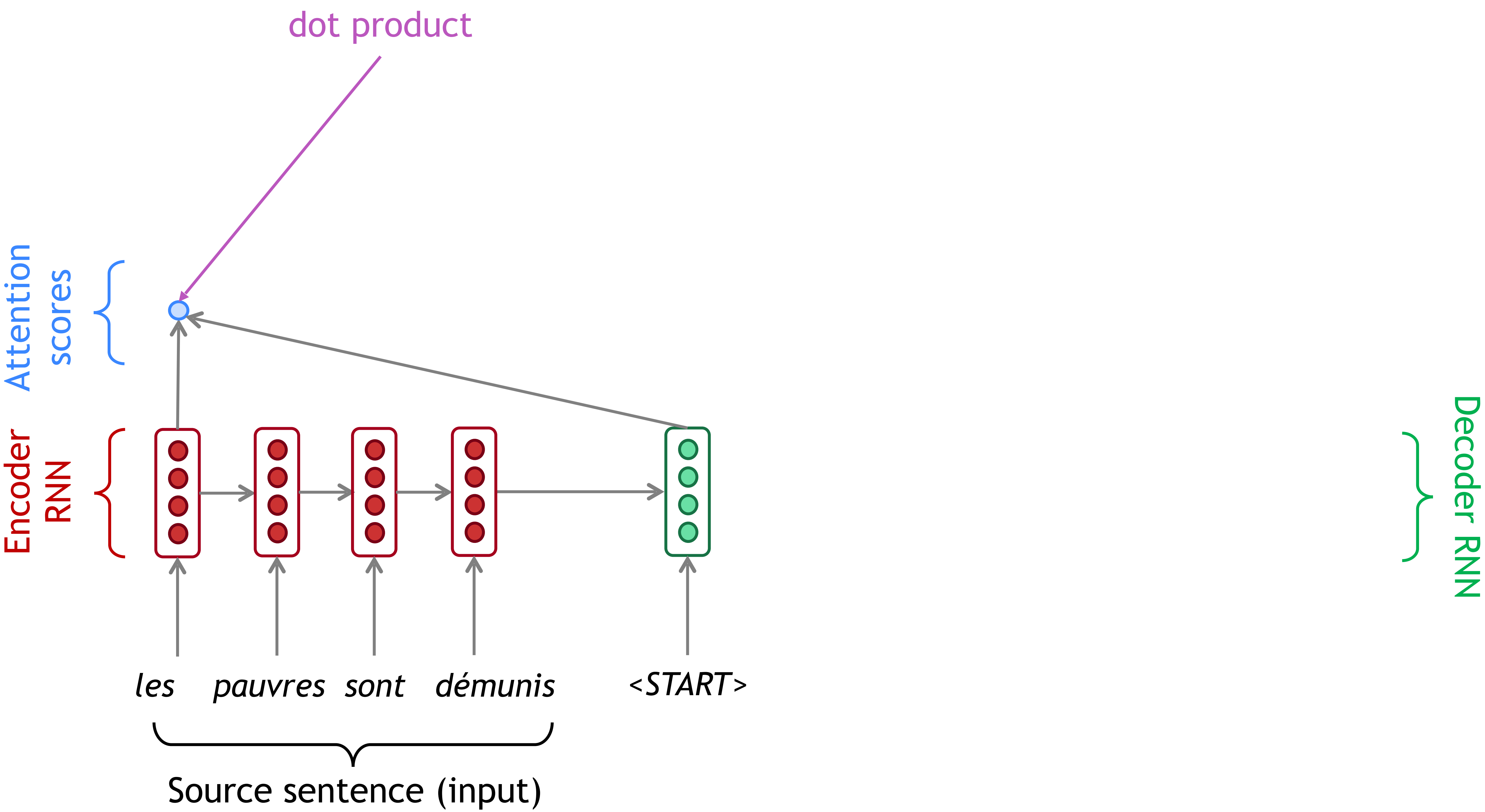


$$\mathbf{a} = (1, 2, 3, 0, 4)^\top$$

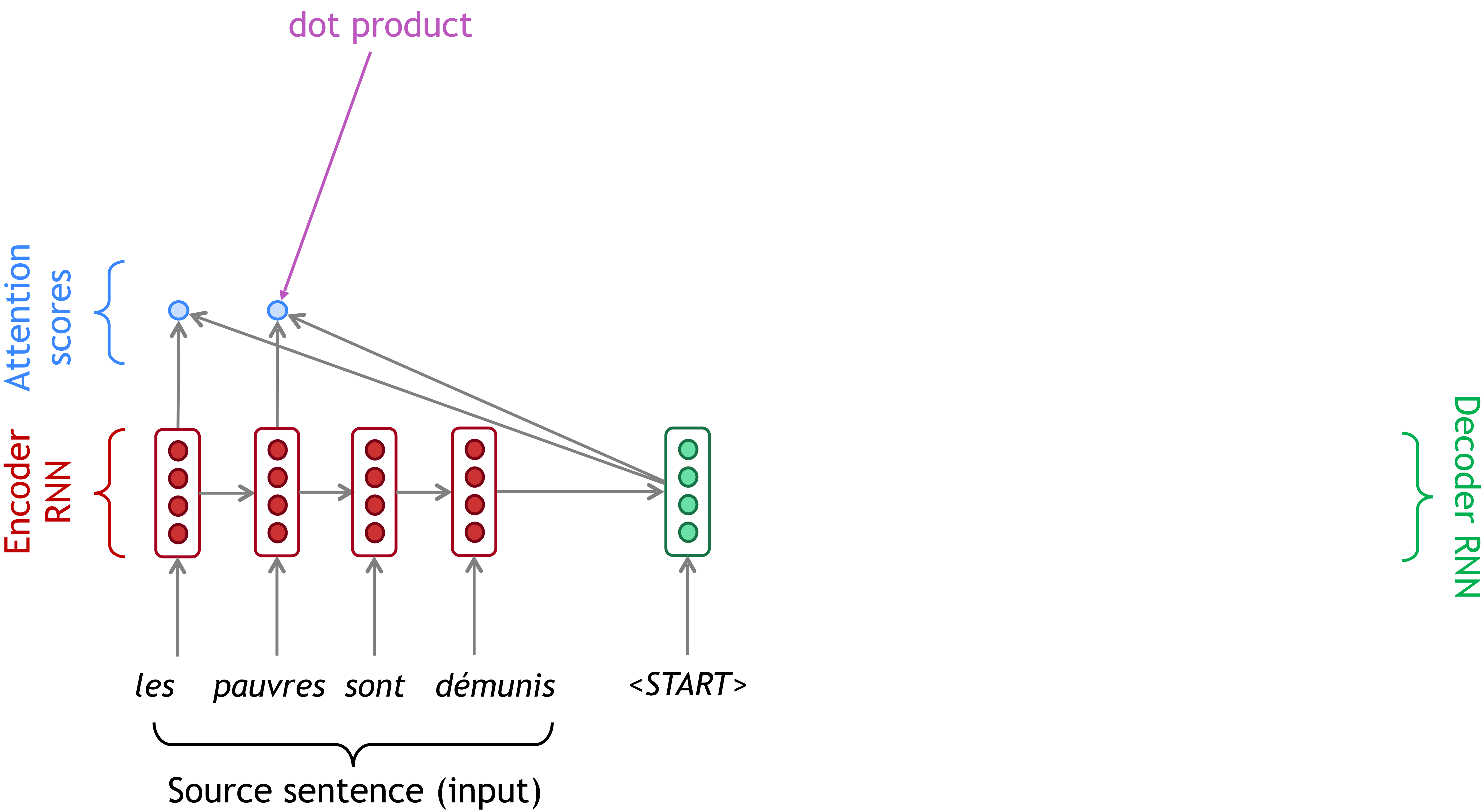
Attention

- The neural MT equivalent of alignment models
- Key idea: At each time step during decoding, **focus on a particular part** of source sentence
  - This depends on the decoder's current hidden state (i.e. notion of what you are trying to decode)
  - Usually implemented as a probability distribution over the hidden states of the encoder (h_i^{enc})

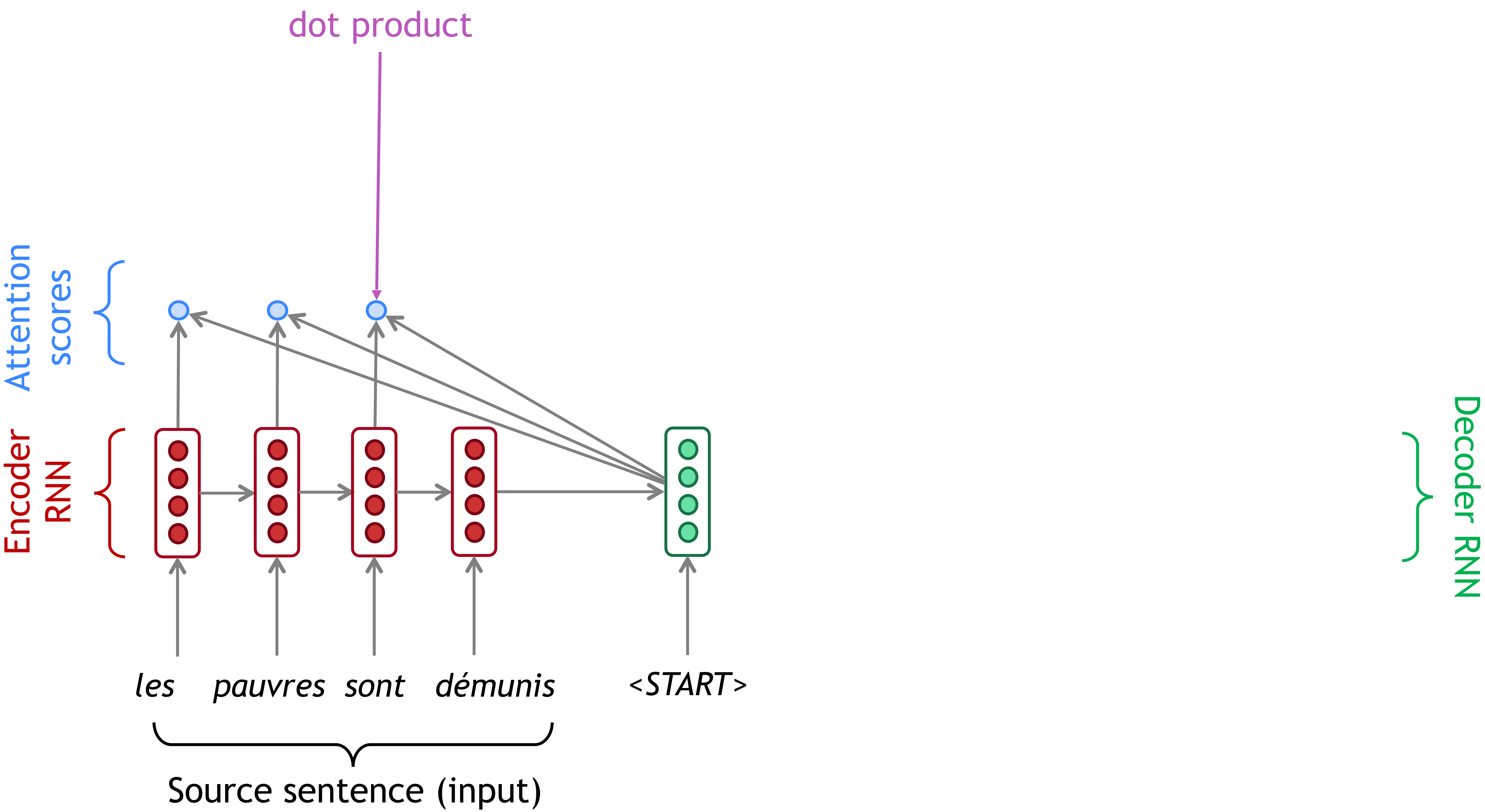
Sequence-to-sequence with attention



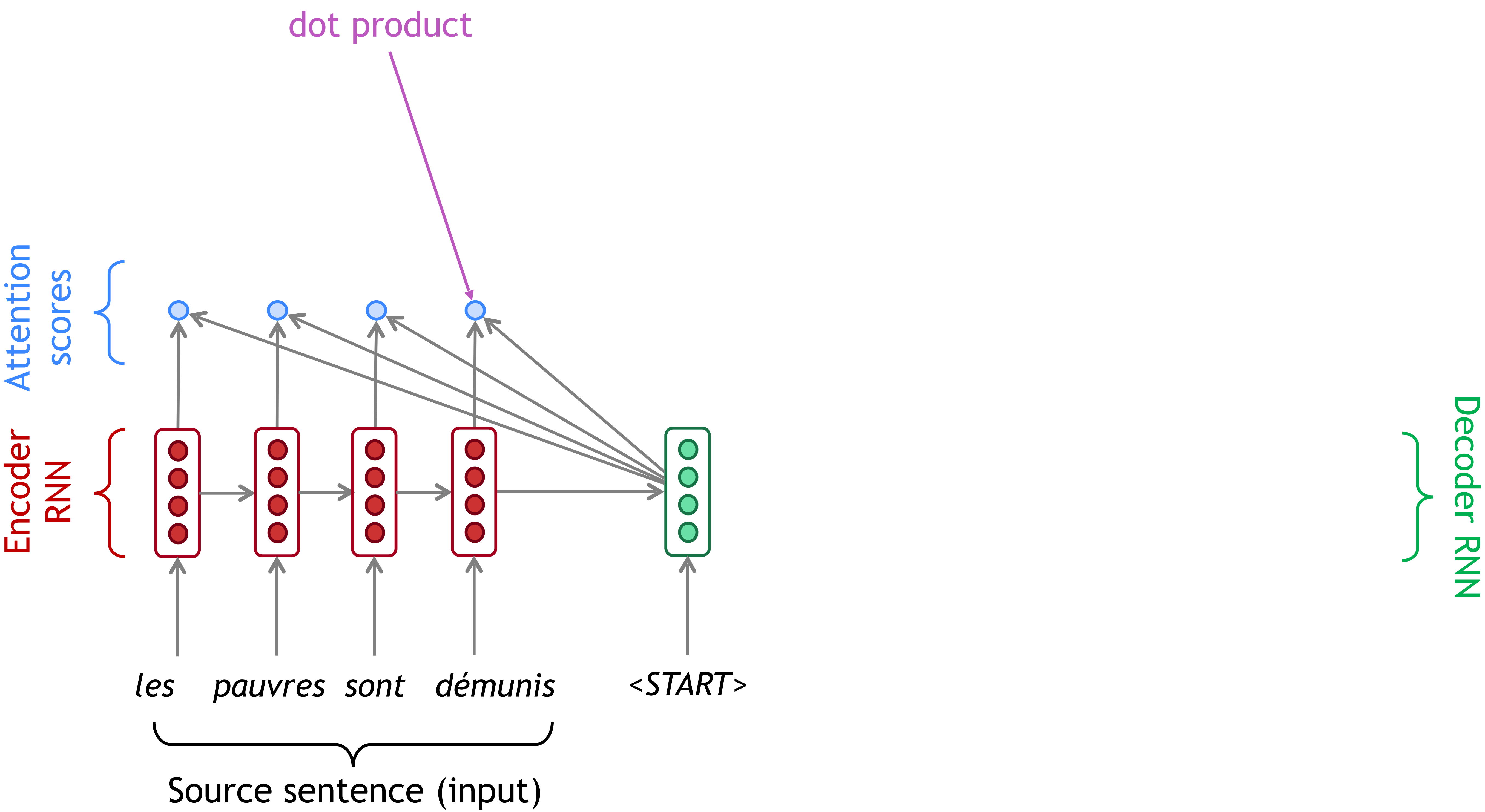
Sequence-to-sequence with attention



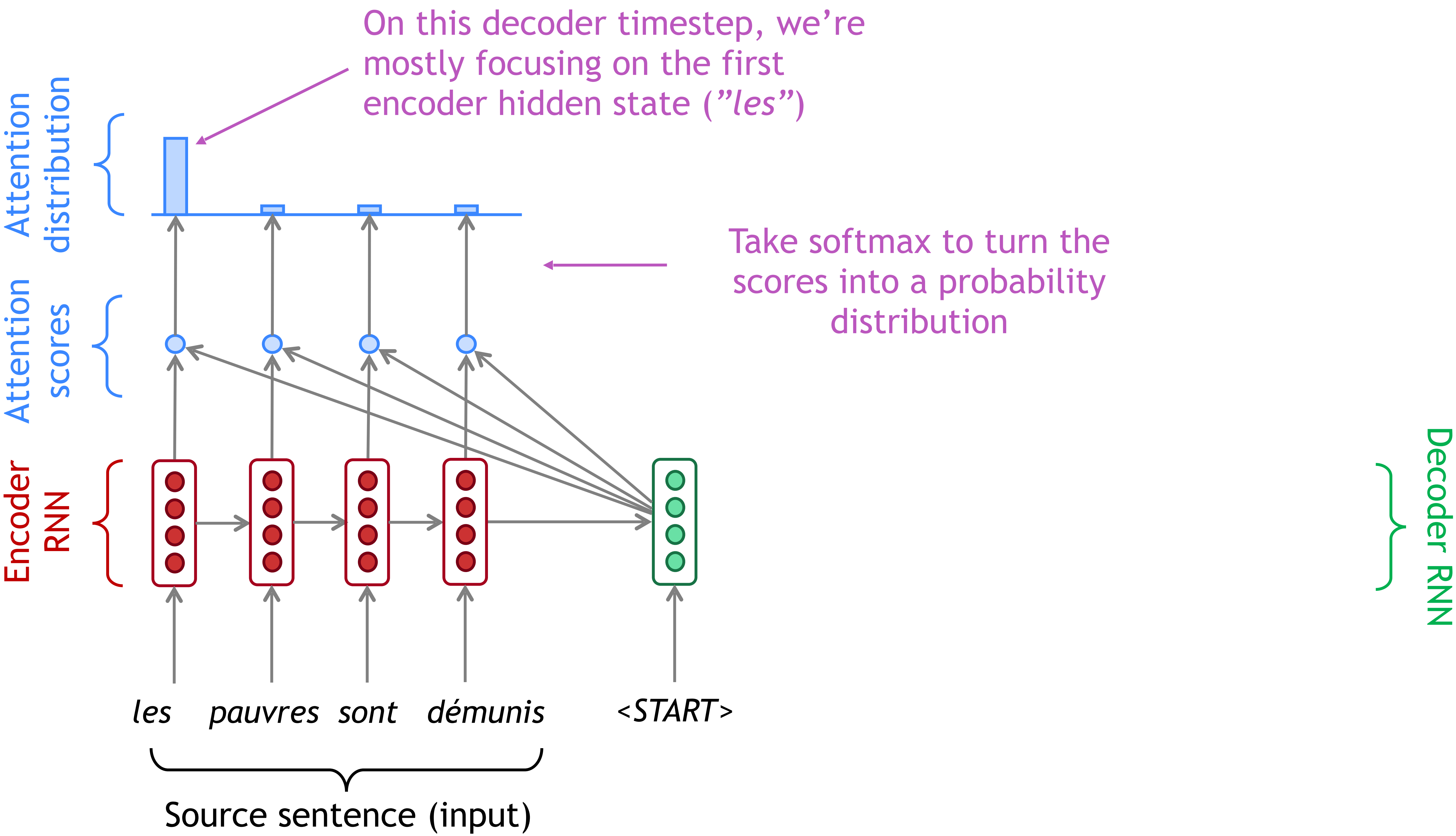
Sequence-to-sequence with attention



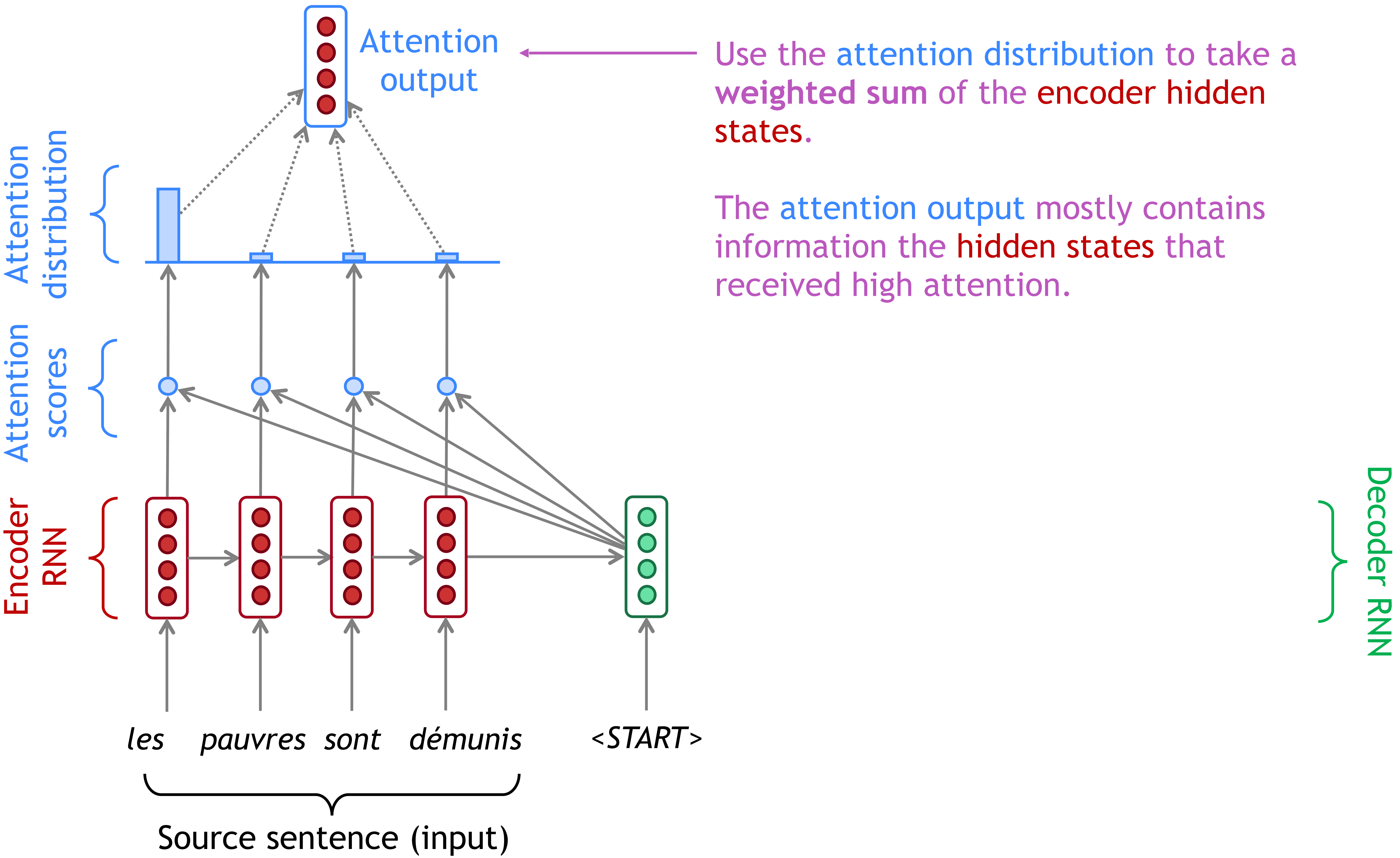
Sequence-to-sequence with attention



Sequence-to-sequence with attention



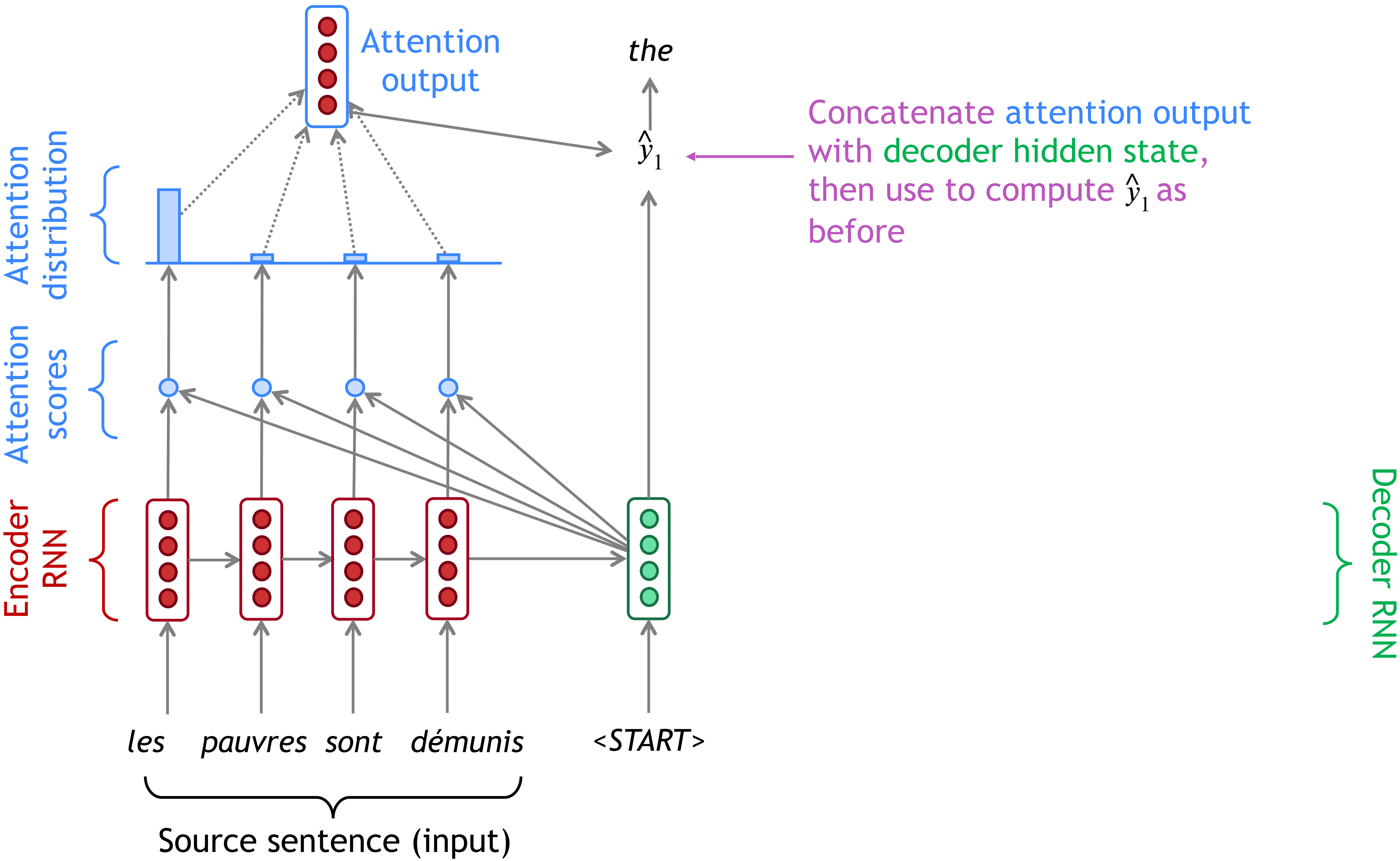
Sequence-to-sequence with attention



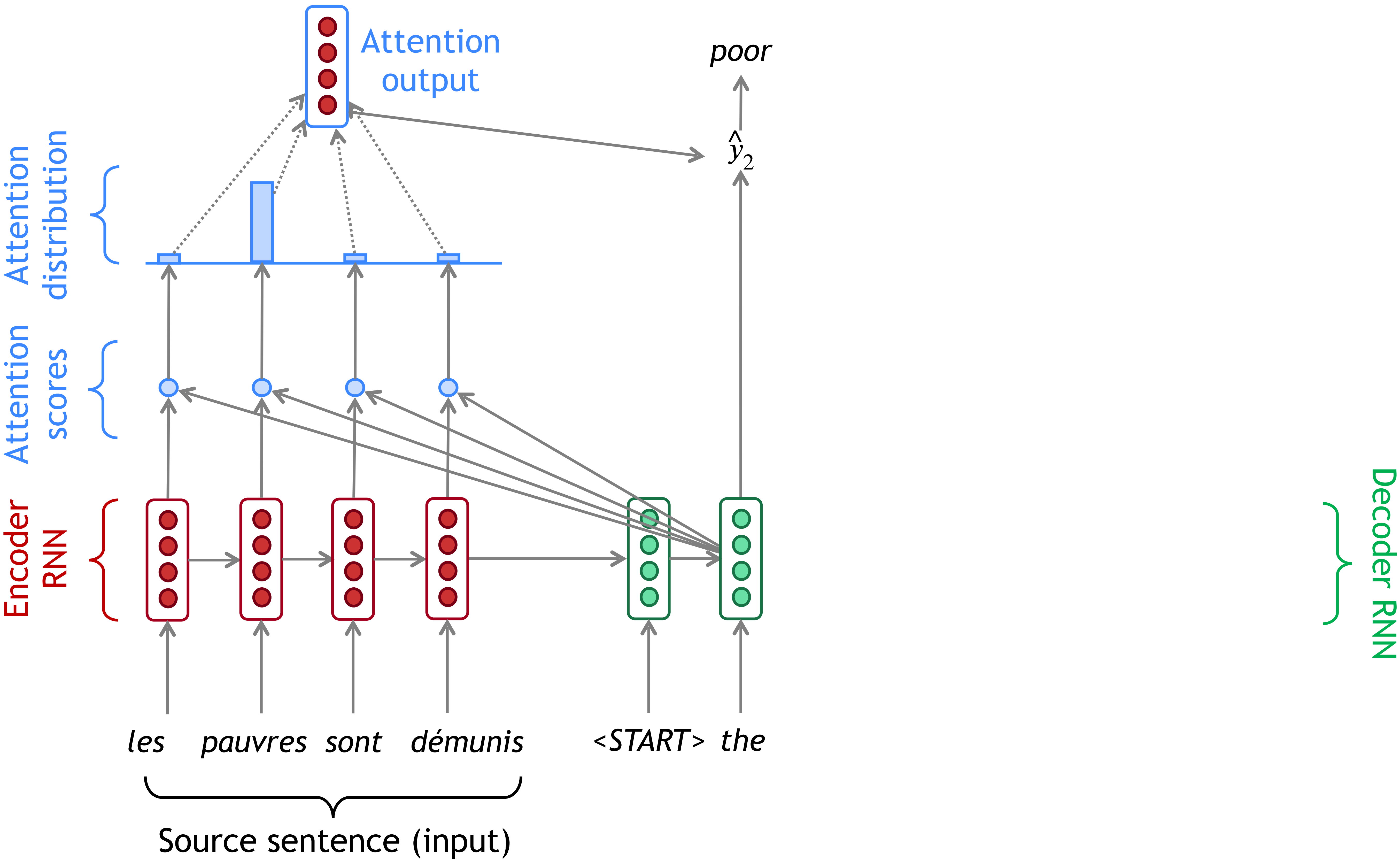
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

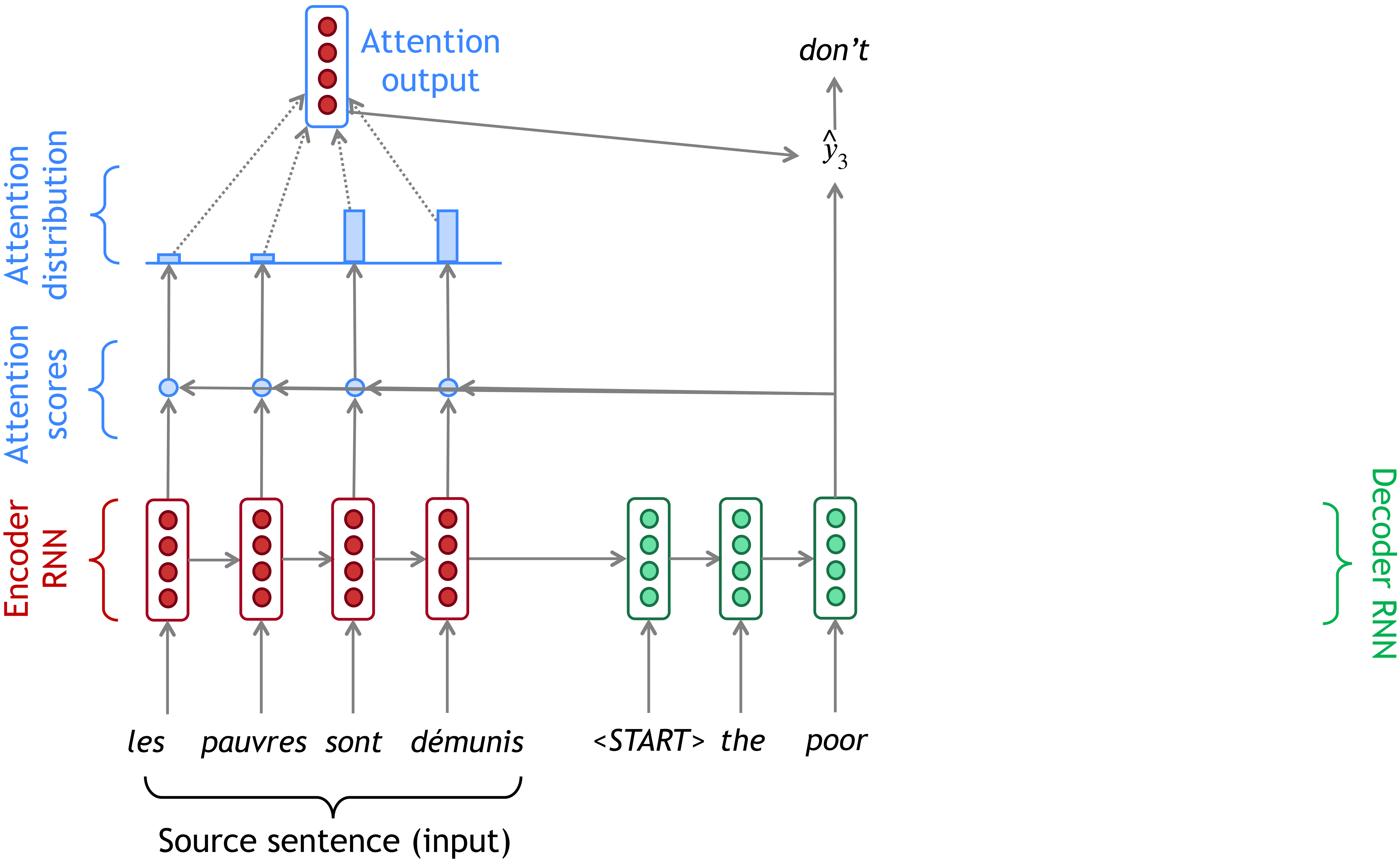
Sequence-to-sequence with attention



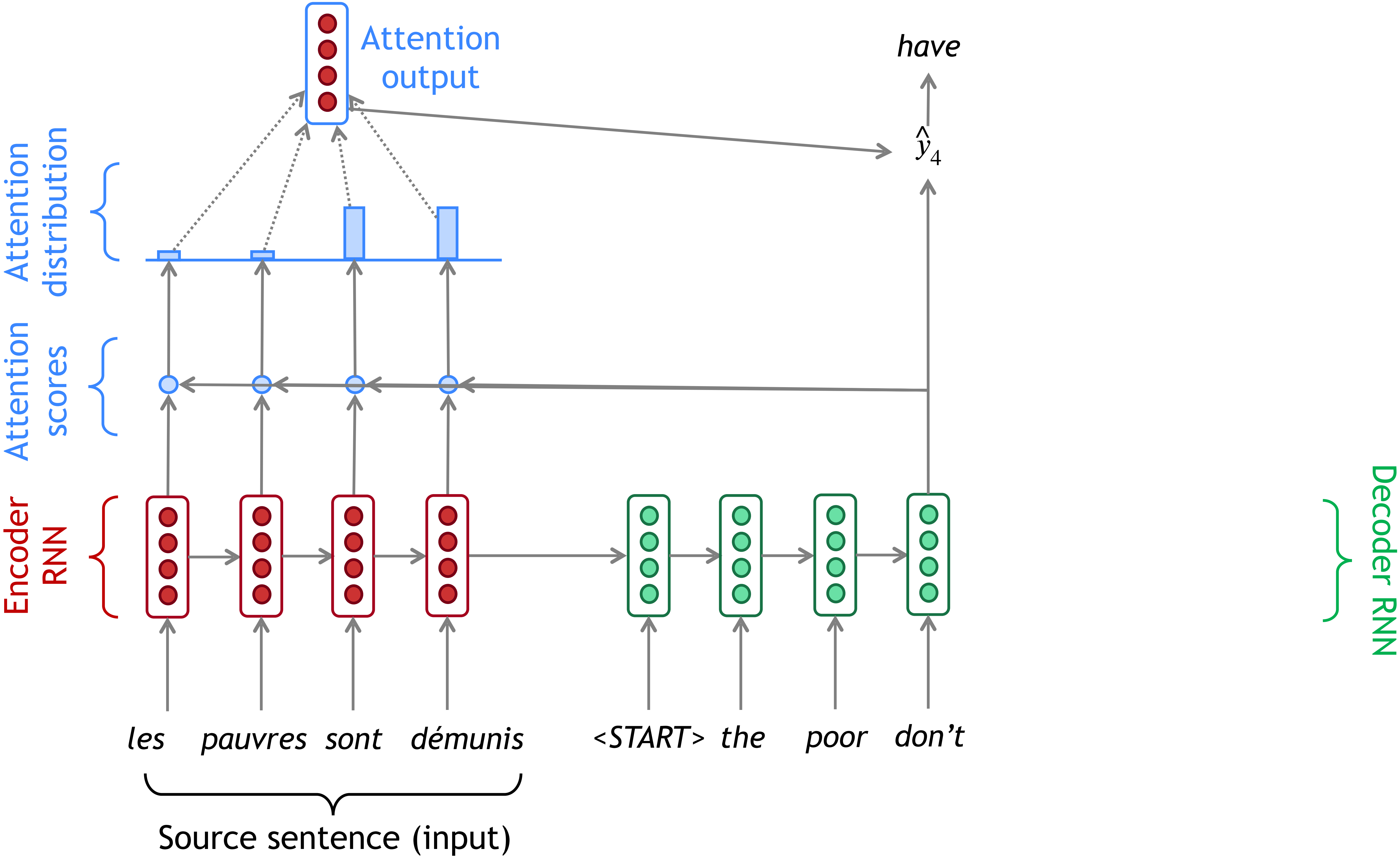
Sequence-to-sequence with attention



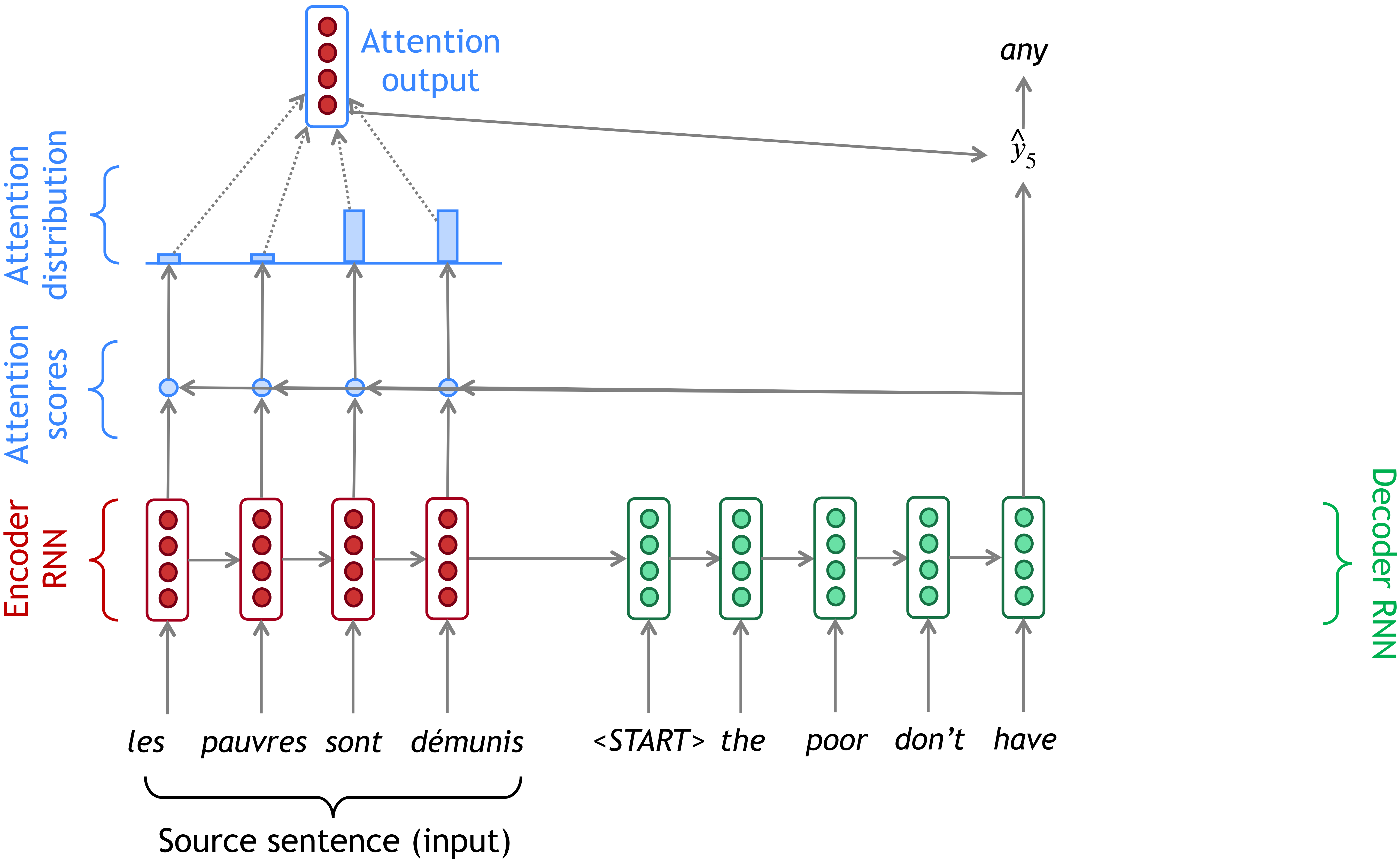
Sequence-to-sequence with attention



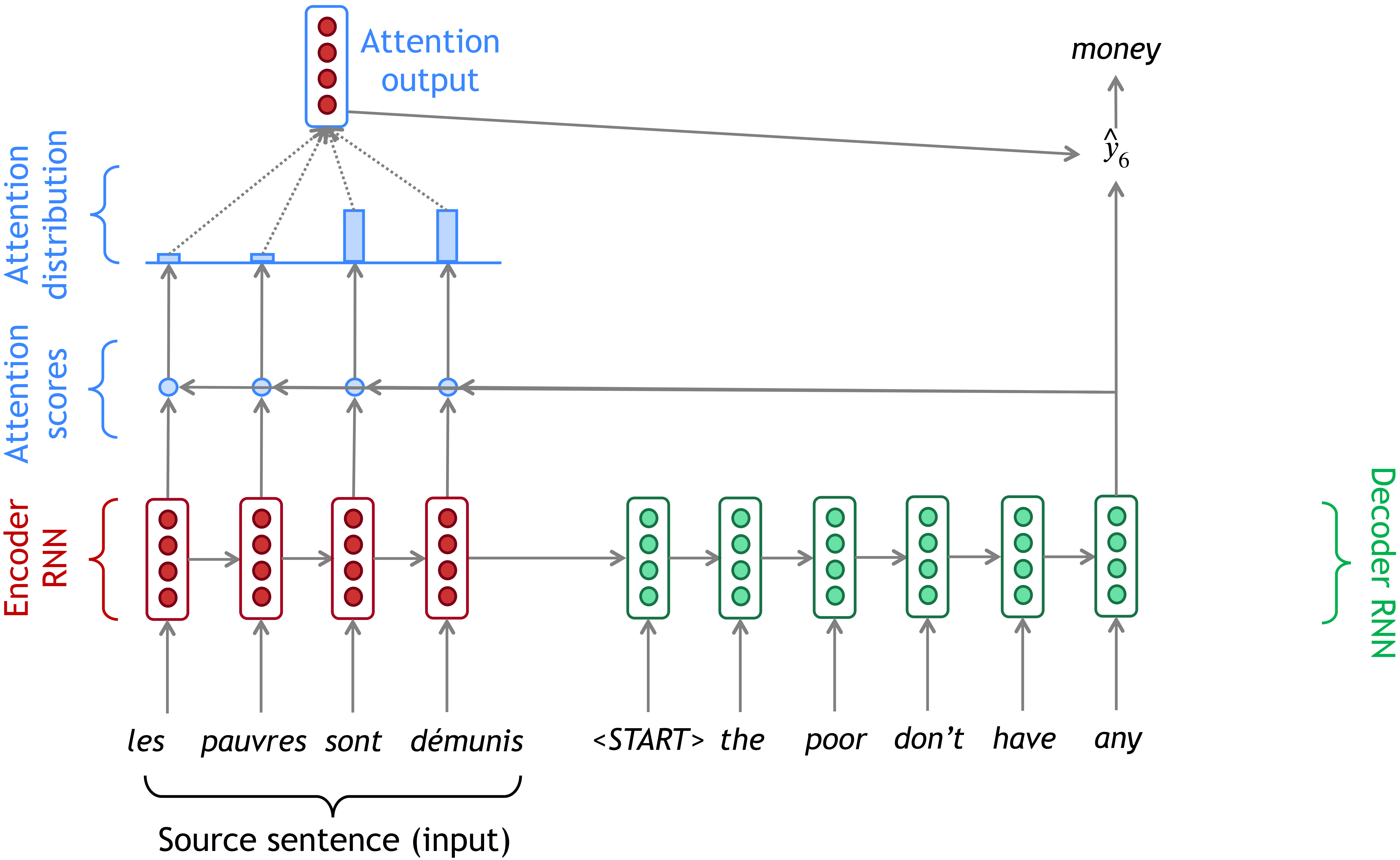
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention



Computing attention

- Encoder hidden states: $h_1^{enc}, \ldots, h_n^{enc}$
- Decoder hidden state at time t: h_t^{dec}
- First, get attention scores for this time step (we will see what g is soon!):

$$e^t = [g(h_1^{enc}, h_t^{dec}), \ldots, g(h_n^{enc}, h_t^{dec})]$$

- Obtain the attention distribution using softmax:

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

- Compute weighted sum of encoder hidden states:

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

- Finally, concatenate with decoder state and pass on to output layer: $[a_t; h_t^{dec}] \in \mathbb{R}^{2h}$

Types of attention

- Assume encoder hidden states $h_1, h_2, \ldots, h_n$ and decoder hidden state z

1. **Dot-product attention** (assumes equal dimensions for a and b:

$$e_i = g(h_i, z) = z^T h_i \in \mathbb{R}$$

2. **Multiplicative attention:**

$g(h_i, z) = z^T W h_i \in \mathbb{R}$, where W is a weight matrix

3. **Additive attention:**

$$g(h_i, z) = v^T \tanh\left(W_1 h_i + W_2 z\right) \in \mathbb{R}$$

where W_1, W_2 are weight matrices and v is a weight vector

Rare Words and Monolingual Text

Handling Rare Words

- Words are a difficult unit to work with, e.g. vocabularies get very large, how to handle OOV?
- Character-level models are possible, but expensive
- Compromise solution: use thousands of “word pieces” (which may be full words but may also be parts of words)

Input: _the **_eco tax** _port i co _in _Po nt - de - Bu is …

Output: _le _port ique **_éco taxe** _de _Pont - de - Bui s

- Can do transliteration, model sub-word regularities, etc.

Sennrich et al. (2016)

Byte Pair Encoding (BPE)

- Start with every individual byte (basically character) as its own symbol

```
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
```

- Count bigram character cooccurrences
- Merge the most frequent pair of adjacent characters

- Use large corpus of text for counting
- 8k merges => vocabulary of around 8000 word pieces. Includes many whole words
- Most SOTA NMT systems use this on both source + target

Sennrich et al. (2016)

Backtranslation

- Classical MT methods used a bilingual corpus of sentences B = (S, T) and a large monolingual corpus T' to train a language model. Can neural MT do the same?

- Approach 1: force the system to generate T' as targets from null inputs

s_1, t_1
s_2, t_2
...
[null], t'_1
[null], t'_2
...

- Approach 2: generate synthetic sources with a T->S machine translation system (backtranslation)

s_1, t_1
s_2, t_2
...
$MT(t'_1), t'_1$
$MT(t'_2), t'_2$
...

Sennrich et al. (2015)

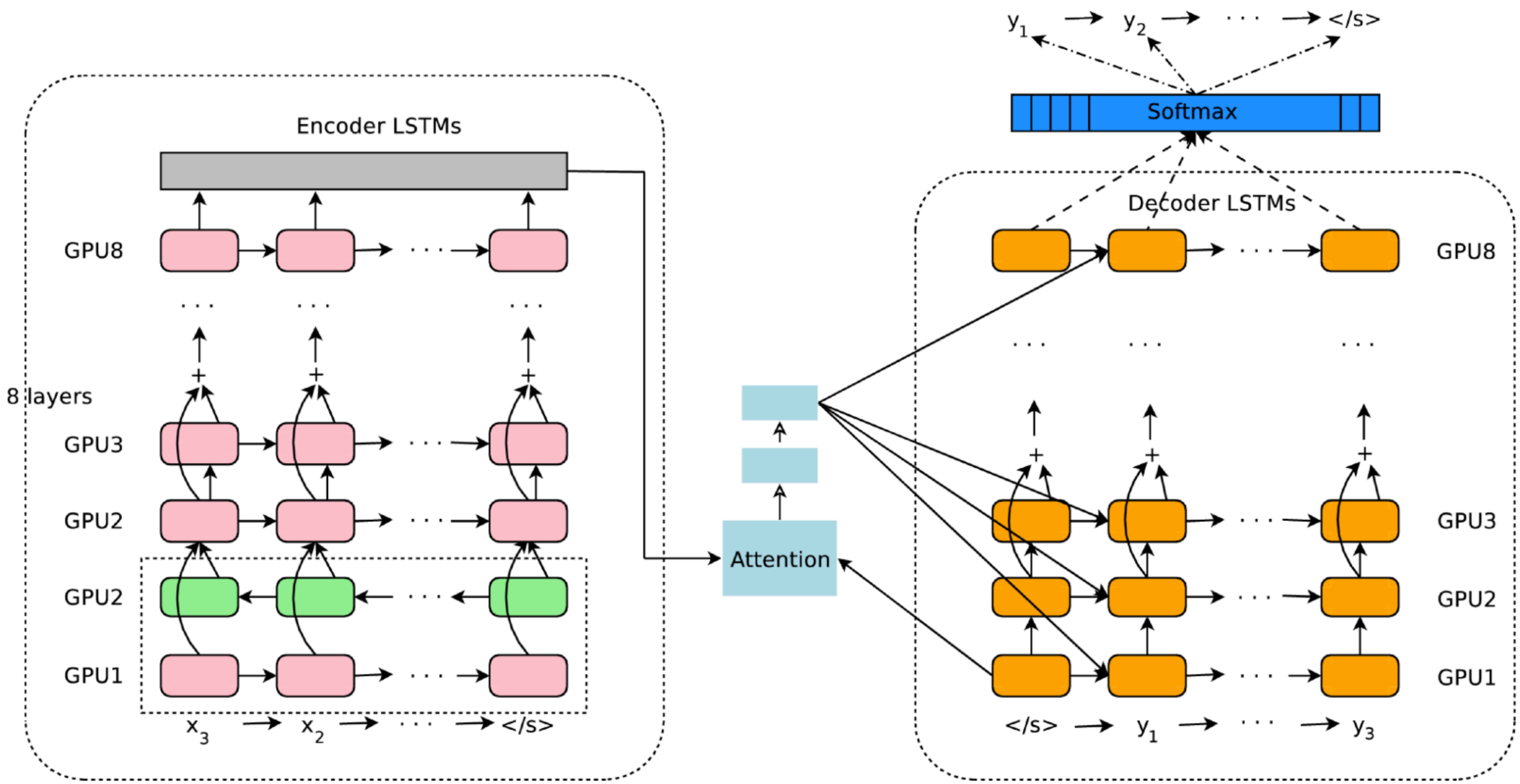
Backtranslation

name	training		BLEU			
	data	instances	tst2011	tst2012	tst2013	tst2014
baseline (Gülçehre et al., 2015)			18.4	18.8	19.9	18.7
deep fusion (Gülçehre et al., 2015)			20.2	20.2	21.3	**20.6**
baseline	parallel	7.2m	18.6	18.2	18.4	18.3
$\text{parallel}_{\text{synth}}$	$\text{parallel/parallel}_{\text{synth}}$	6m/6m	19.9	20.4	20.1	20.0
$\text{Gigaword}_{\text{mono}}$	$\text{parallel/Gigaword}_{\text{mono}}$	7.6m/7.6m	18.8	19.6	19.4	18.2
$\text{Gigaword}_{\text{synth}}$	$\text{parallel/Gigaword}_{\text{synth}}$	8.4m/8.4m	**21.2**	**21.1**	**21.8**	20.4

- Gigaword: large monolingual English corpus
- $\text{parallel}_{\text{synth}}$: backtranslate training data; makes additional noisy source sentences which could be useful

Sennrich et al. (2015)

Google's NMT System



- 8-layer LSTM encoder-decoder with attention, word piece vocabulary of 8k-32k

Wu et al. (2016)

RESEARCH › PUBLICATIONS ›

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

Table 10: Mean of side-by-side scores on production data

	PBMT	GNMT	Human	Relative Improvement
English → Spanish	4.885	5.428	5.504	87%
English → French	4.932	5.295	5.496	64%
English → Chinese	4.035	4.594	4.987	58%
Spanish → English	4.872	5.187	5.372	63%
French → English	5.046	5.343	5.404	83%
Chinese → English	3.694	4.263	4.636	60%

(Wu et al., 2016)

Google's NMT System

Source	She was spotted three days later by a dog walker trapped in the quarry	
PBMT	Elle a été repéré trois jours plus tard par un promeneur de chien piégé dans la carrière	6.0
GNMT	Elle a été repérée trois jours plus tard par un traîneau à chiens piégé dans la carrière.	2.0
Human	Elle a été repérée trois jours plus tard par une personne qui promenait son chien coincée dans la carrière	5.0

Gender is correct in GNMT but not in PBMT

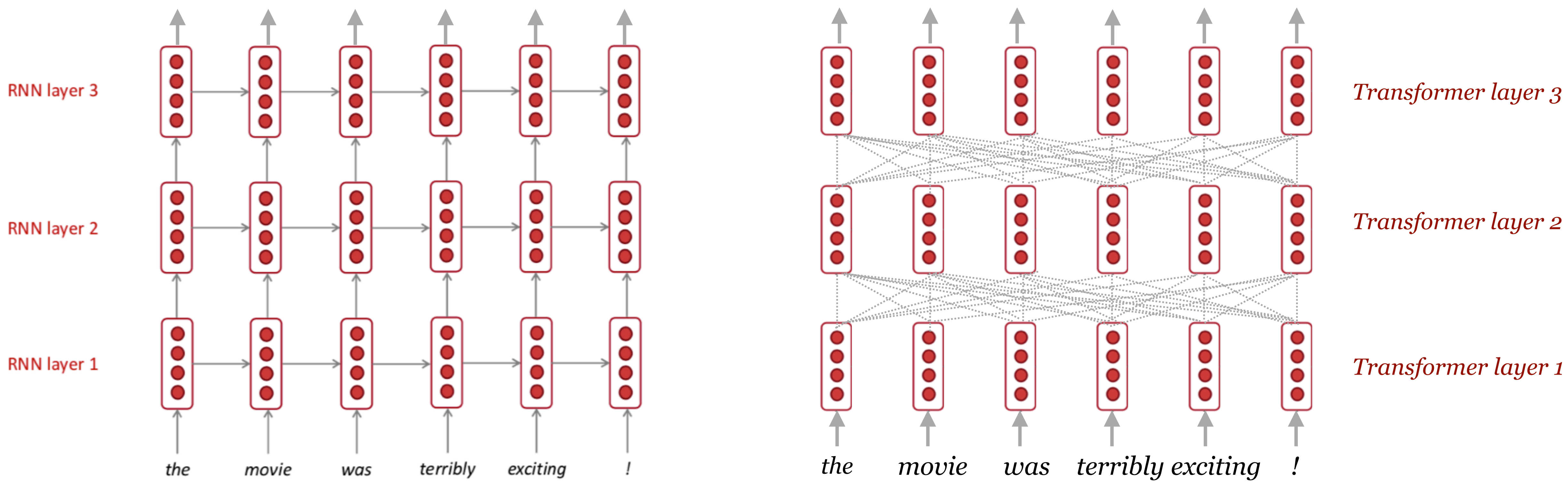
"sled"

"walker"

Wu et al. (2016)

Transformers for MT

RNNs vs Transformers

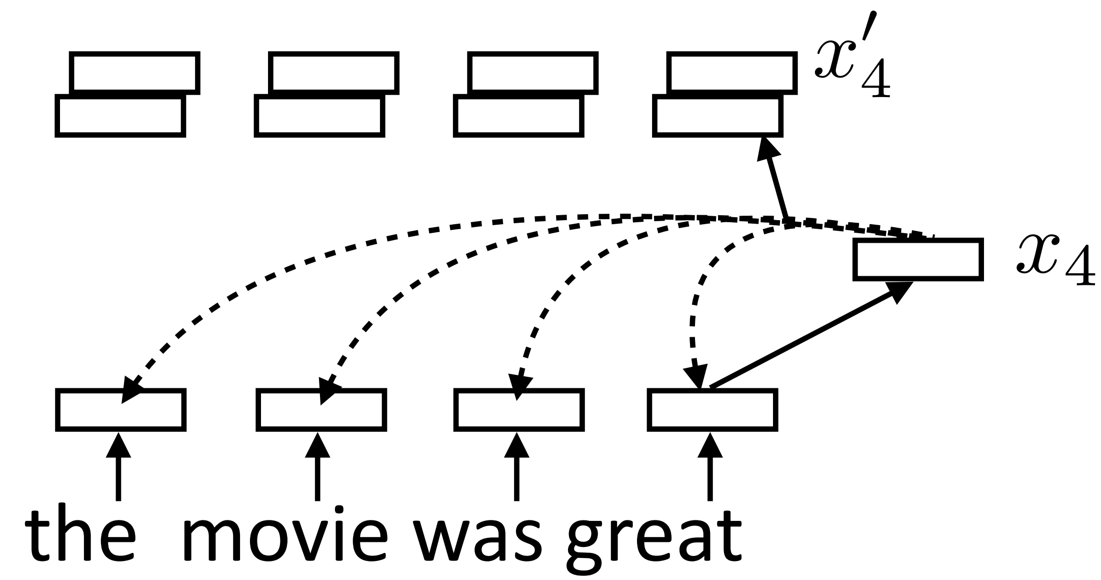


New Twist: Self-Attention

- Each word computes attention over every other word

$$\alpha_{i,j} = \mathrm{softmax}(x_i^\top x_j)$$ scalar

$$x'_i = \sum_{j=1}^{n} \alpha_{i,j} x_j$$ vector = sum of scalar * vector



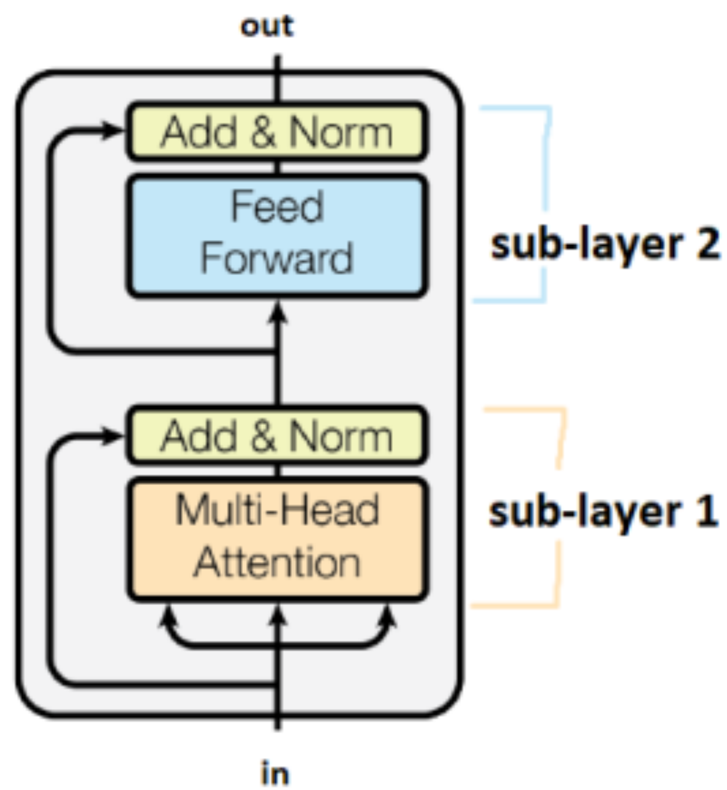
- Multiple "heads": Use parameters W_k and V_k to get different attention values + transform vectors

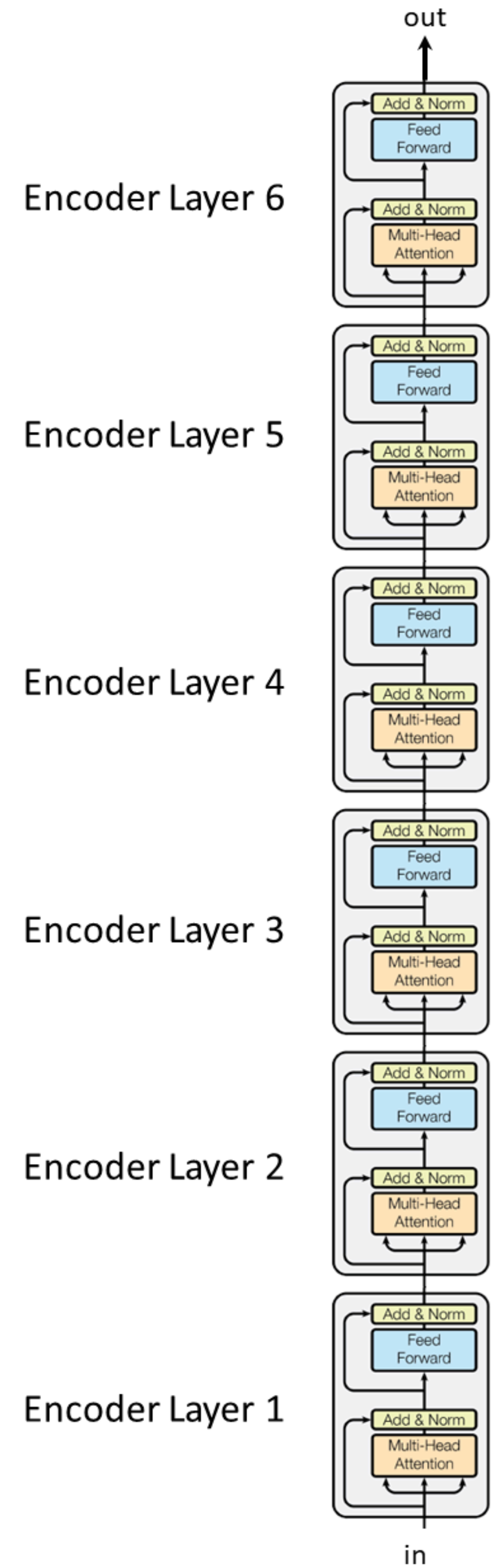
$$\alpha_{k,i,j} = \mathrm{softmax}(x_i^\top W_k x_j) \qquad x'_{k,i} = \sum_{j=1}^{n} \alpha_{k,i,j} V_k x_j$$

Vaswani et al. (2017)

Transformers

- Transformers





Attention is **all you need**

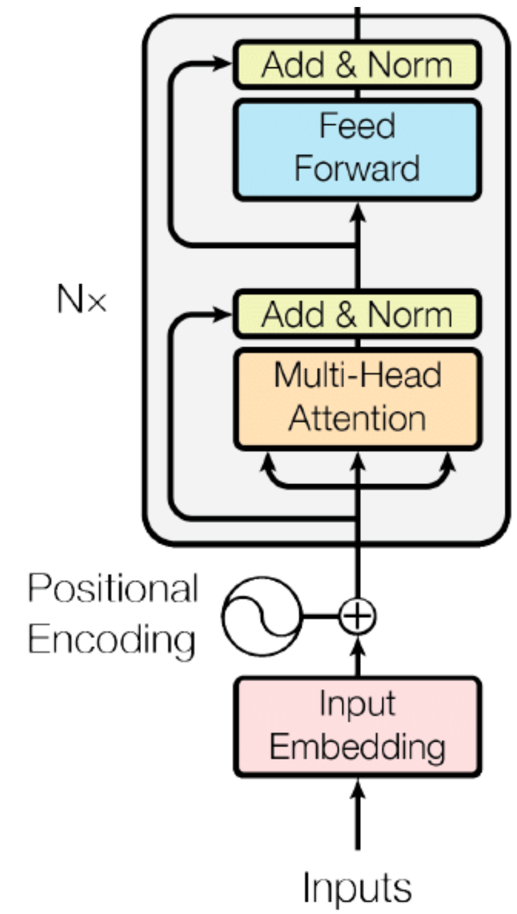
A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017 - papers.nips.cc

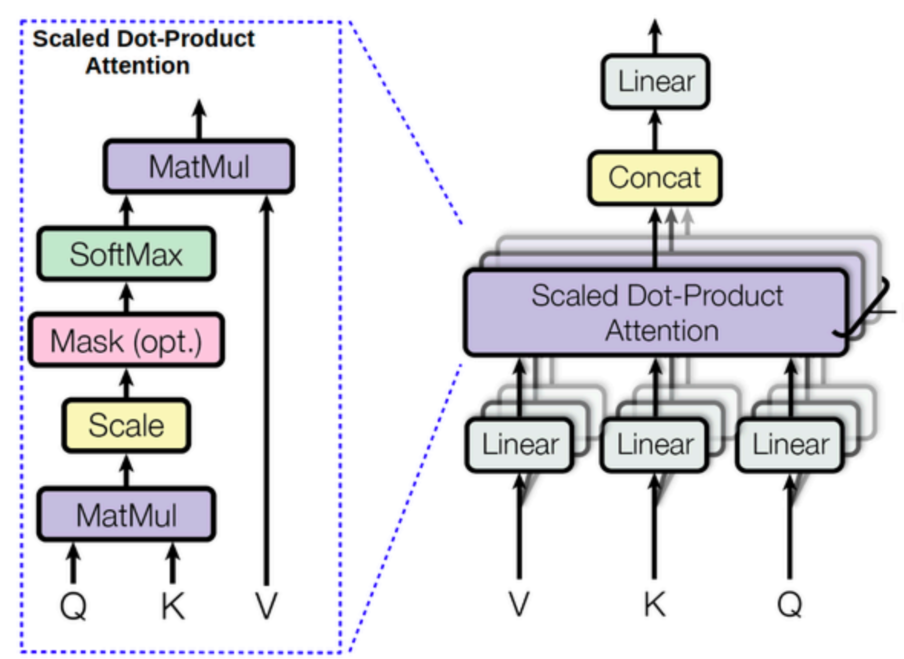
The dominant sequence transduction models are based on complex recurrent orconvolutional neural networks in an encoder and decoder configuration. The best performing such models also connect the encoder and decoder through an attentionm …

☆ 99 Cited by 6254 Related articles All 20 versions »

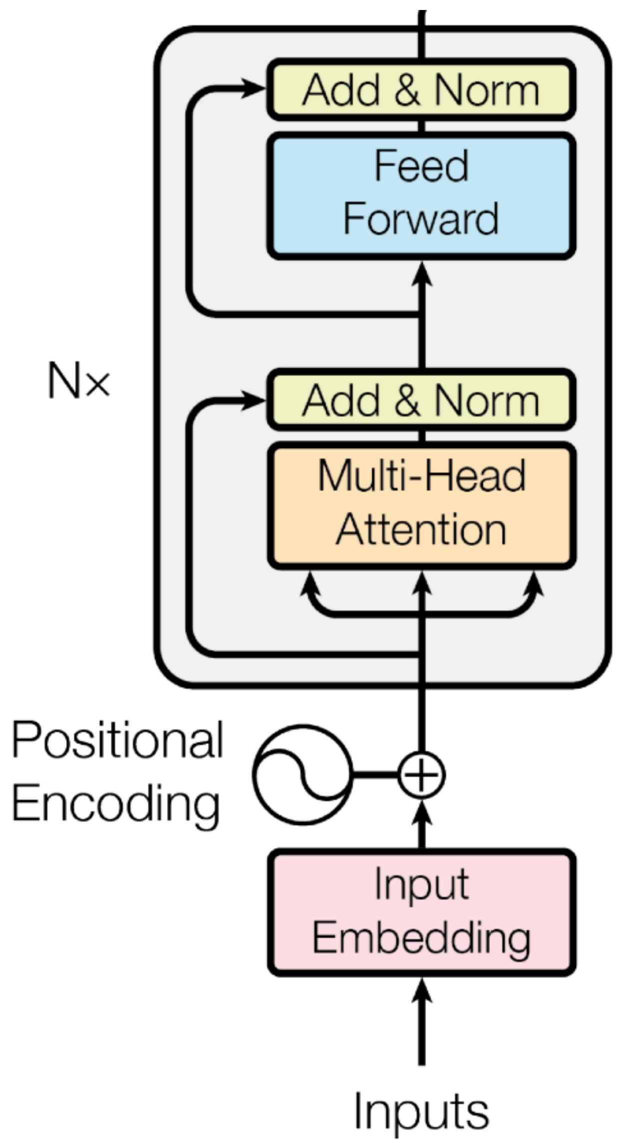
Transformers

- NIPS'17: Attention is All You Need
- Key idea: Multi-head self-attention
- No recurrence structure any more so it trains much faster
- Originally proposed for NMT (encoder-decoder framework)
- Used as the base model for lots of follow up work





Transformers



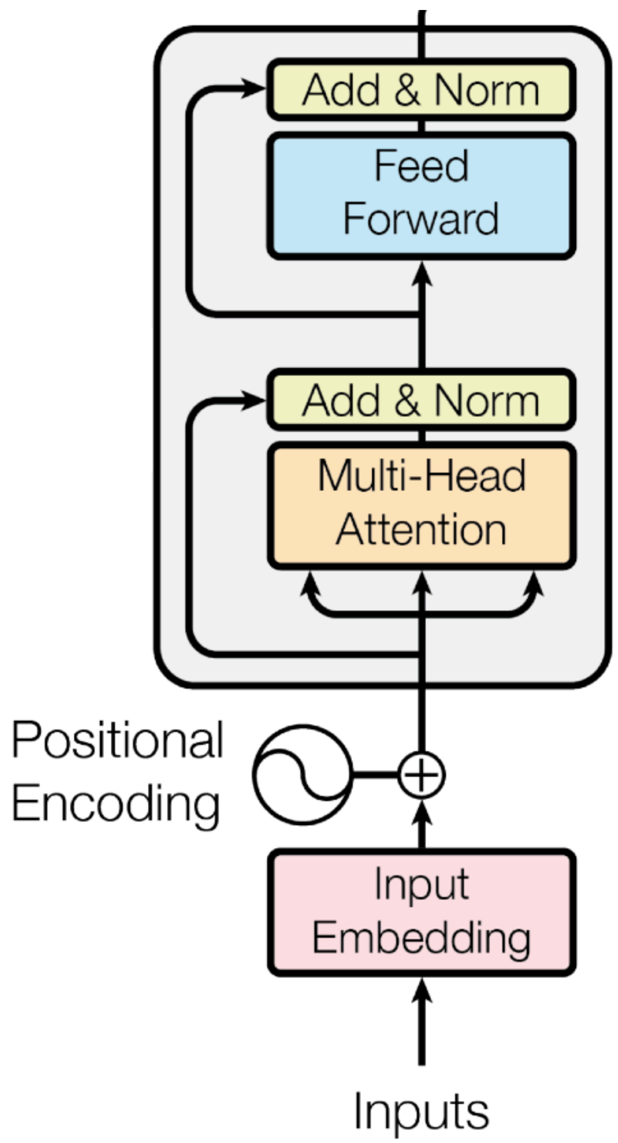
- Each Transformer block has two sub-layers
  - Multi-head attention
  - 2-layer feedforward NN (with ReLU)
- Each sublayer has a residual connection and a layer normalization

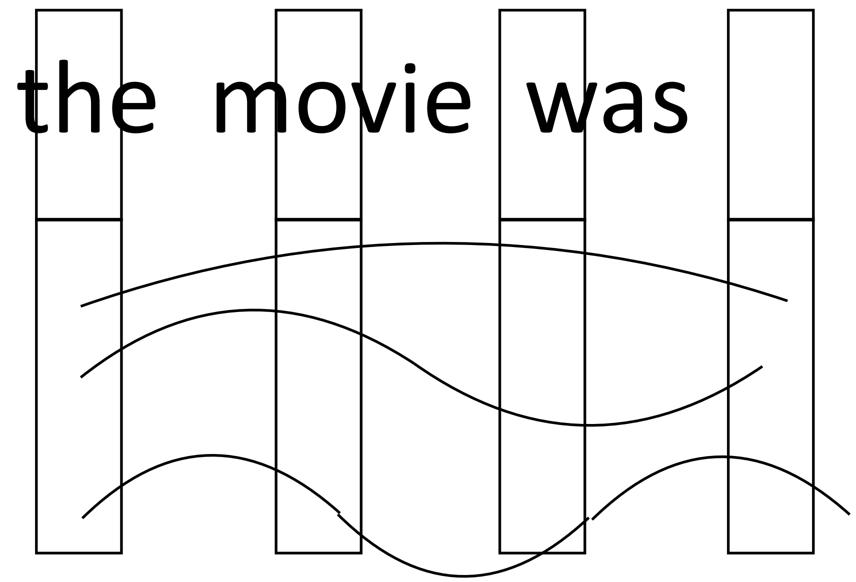
$$\mathrm{LayerNorm}(x + \mathrm{SubLayer}(x))$$

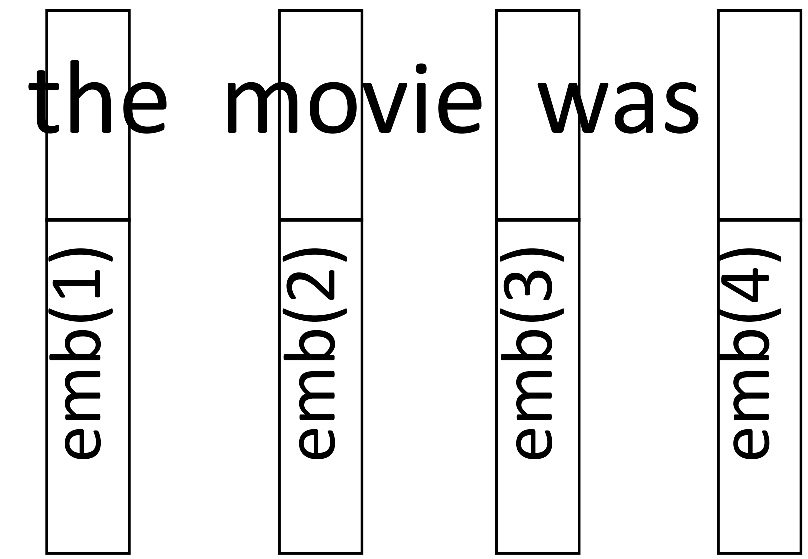
- Input layer has a positional encoding

(Ba et al, 2016): Layer Normalization

Transformers and Word Order



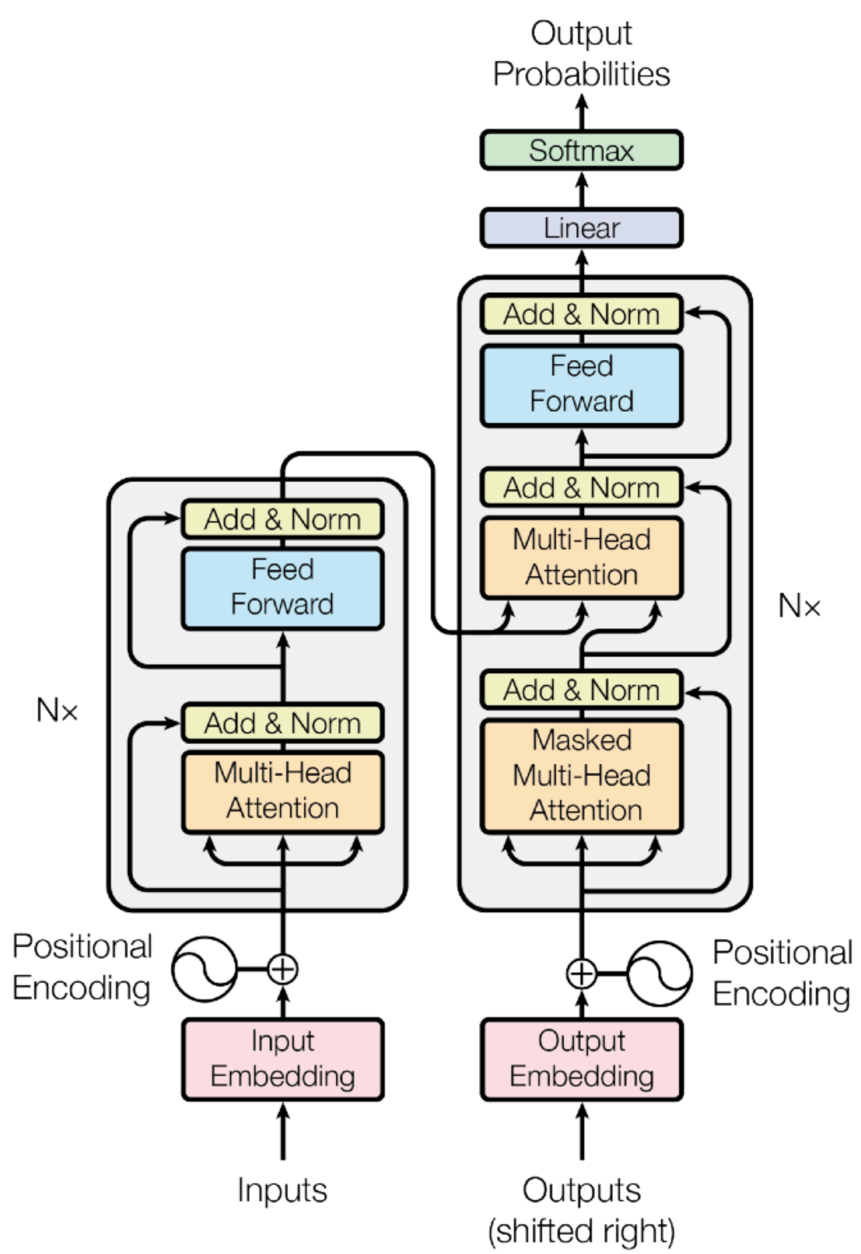




- Augment word embedding with position embeddings, each dim is a sine/cosine wave of a different frequency. Closer points = higher dot products
- Works essentially as well as just encoding position as a one-hot vector

Vaswani et al. (2017)

Transformers for MT



- Encoder and decoder are both transformers
- Decoder consumes the previous generated token (and attends to input), but has *no recurrent state*

Vaswani et al. (2017)

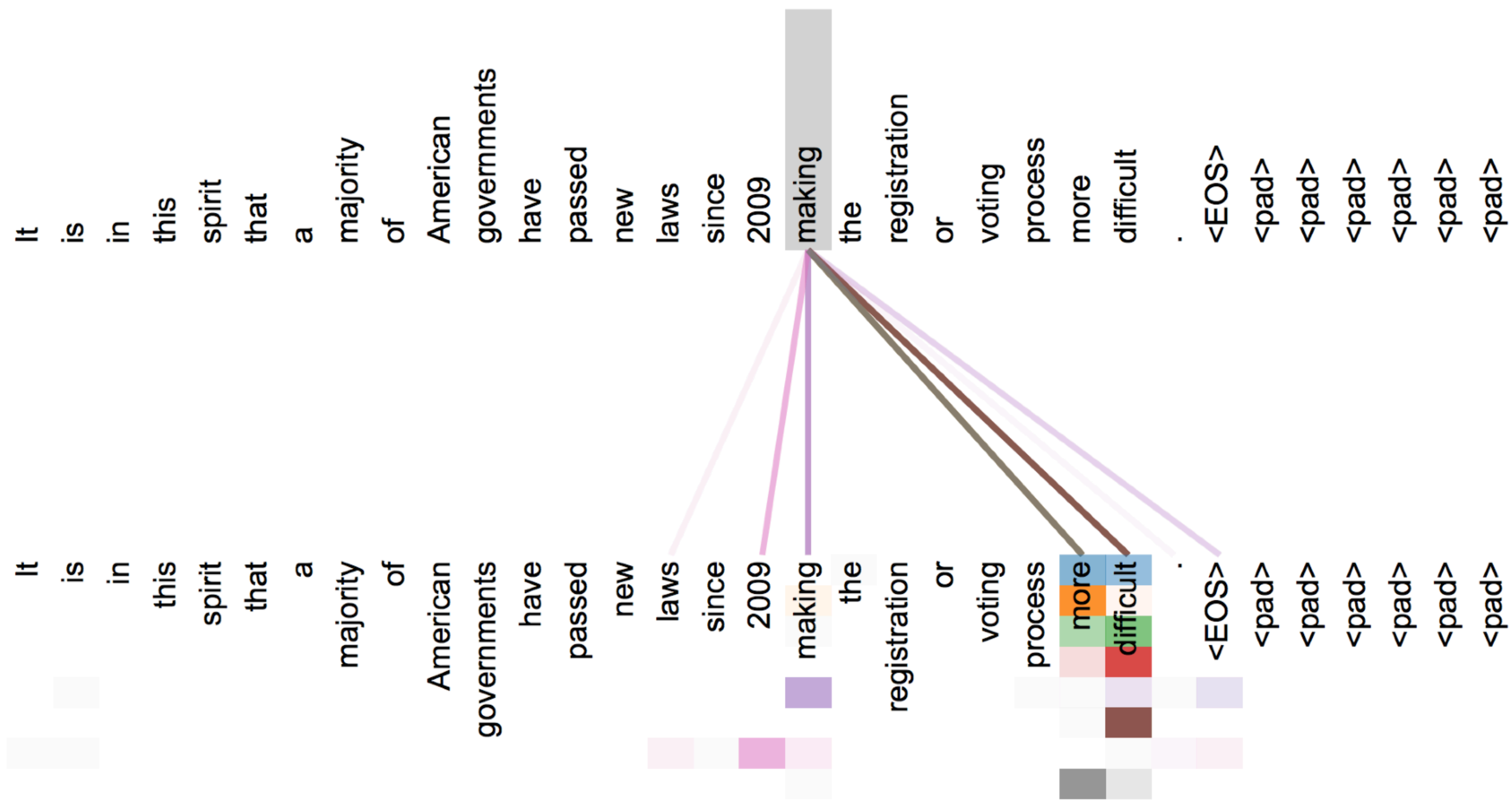
Transformers

- Big = 6 layers, 1000 dim for each token, 16 heads, base = 6 layers + other params halved

Model	BLEU	
	EN-DE	EN-FR
ByteNet [18]	23.75	
Deep-Att + PosUnk [39]		39.2
GNMT + RL [38]	24.6	39.92
ConvS2S [9]	25.16	40.46
MoE [32]	26.03	40.56
Deep-Att + PosUnk Ensemble [39]		40.4
GNMT + RL Ensemble [38]	26.30	41.16
ConvS2S Ensemble [9]	26.36	**41.29**
Transformer (base model)	27.3	38.1
Transformer (big)	**28.4**	**41.8**

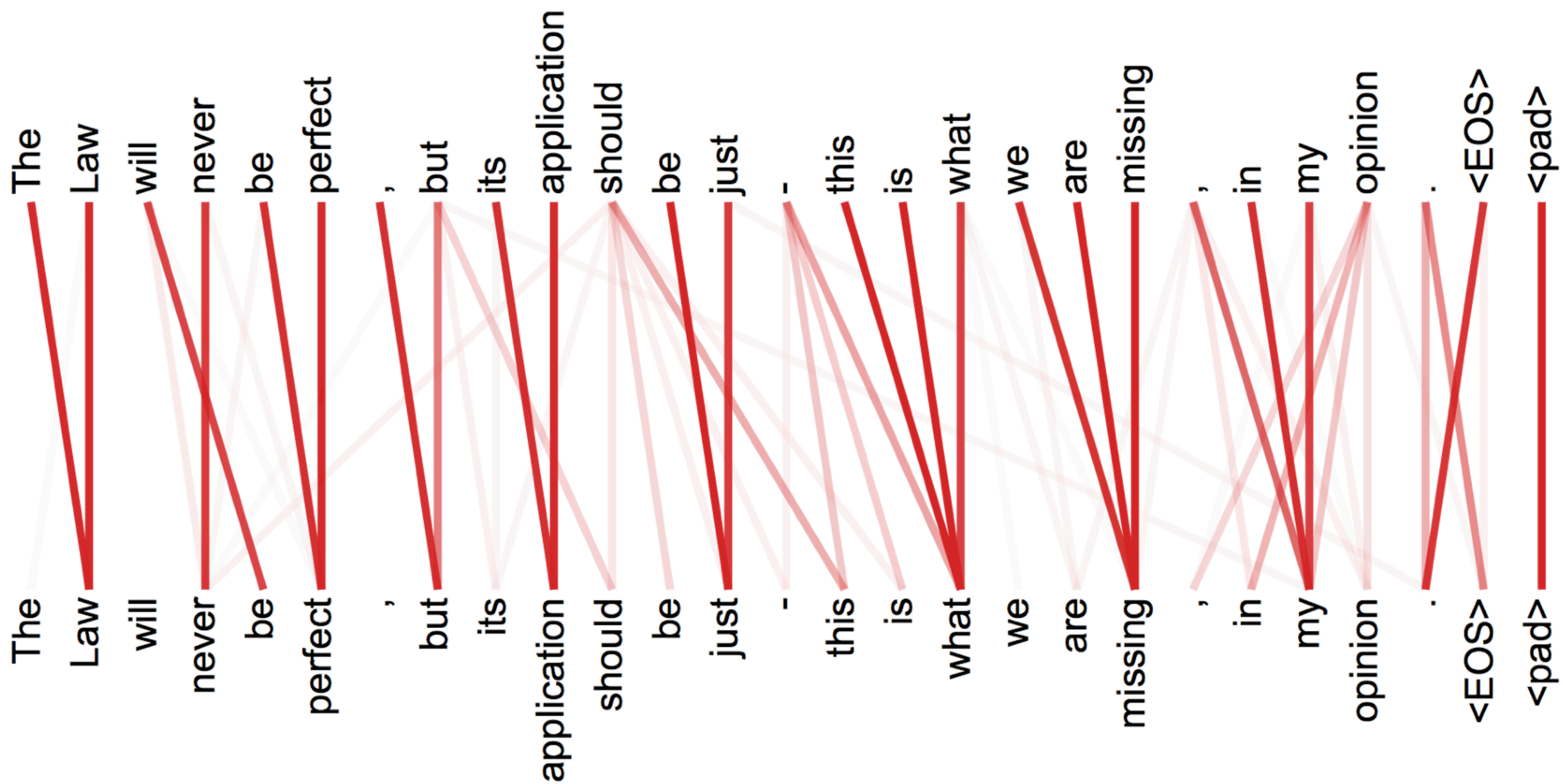
Vaswani et al. (2017)

Visualization



Vaswani et al. (2017)

Visualization



Vaswani et al. (2017)

Useful Resources

nn.Transformer:

```
>>> transformer_model = nn.Transformer(nhead=16, num_encoder_layers=12)
>>> src = torch.rand((10, 32, 512))
>>> tgt = torch.rand((20, 32, 512))
>>> out = transformer_model(src, tgt)
```

nn.TransformerEncoder:

```
>>> encoder_layer = nn.TransformerEncoderLayer(d_model=512, nhead=8)
>>> transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers=6)
>>> src = torch.rand(10, 32, 512)
>>> out = transformer_encoder(src)
```

The Annotated Transformer:

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

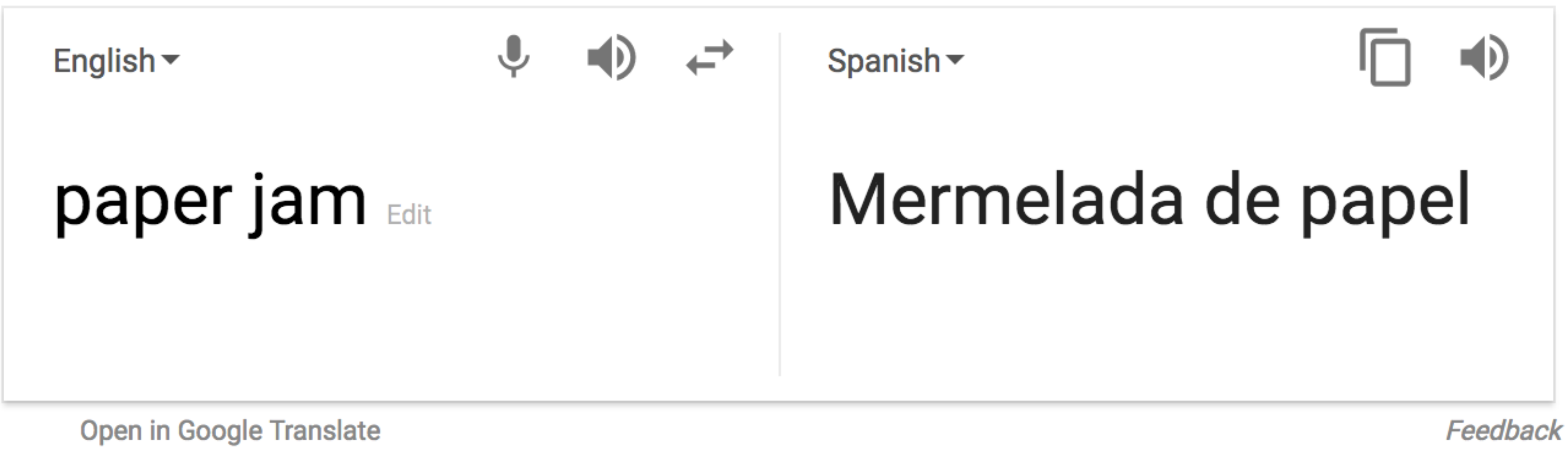
A Jupyter notebook which explains how Transformer works line by line in PyTorch!

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

So is Machine Translation solved?

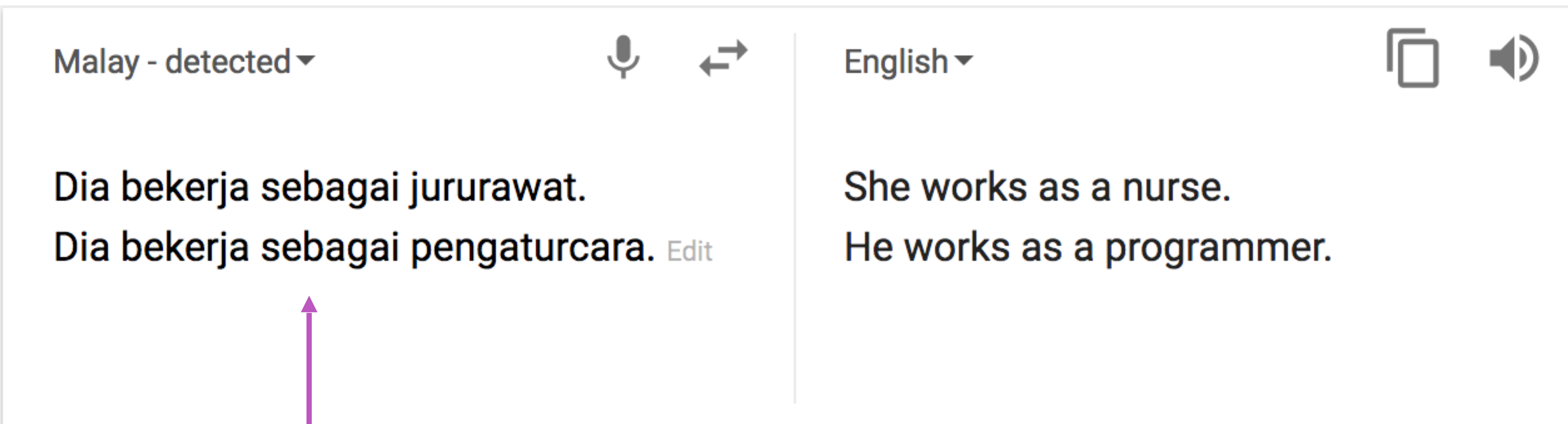
- Nope!
- Using common sense is still hard



?

So is Machine Translation solved?

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

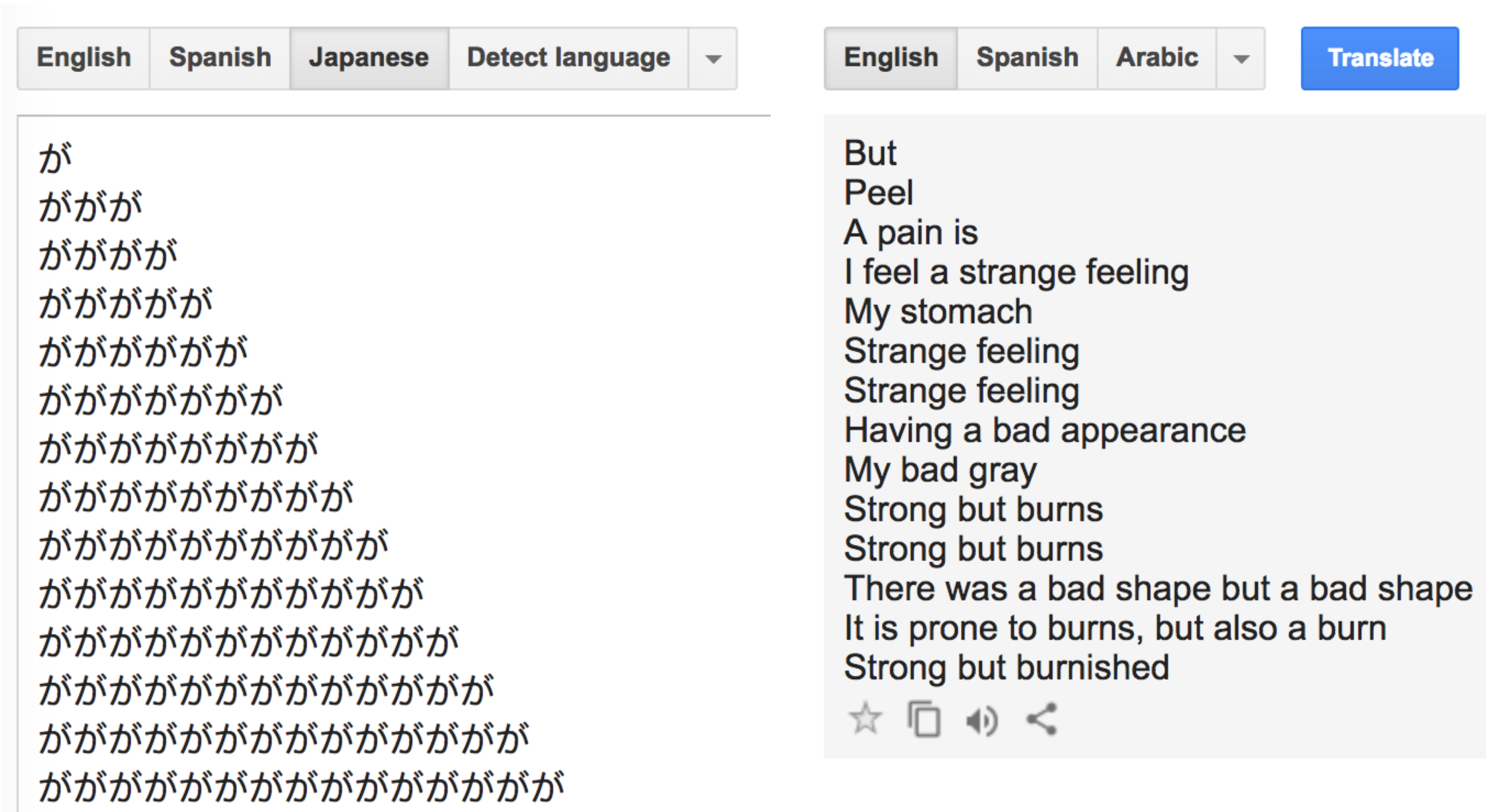


Didn’t specify gender

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

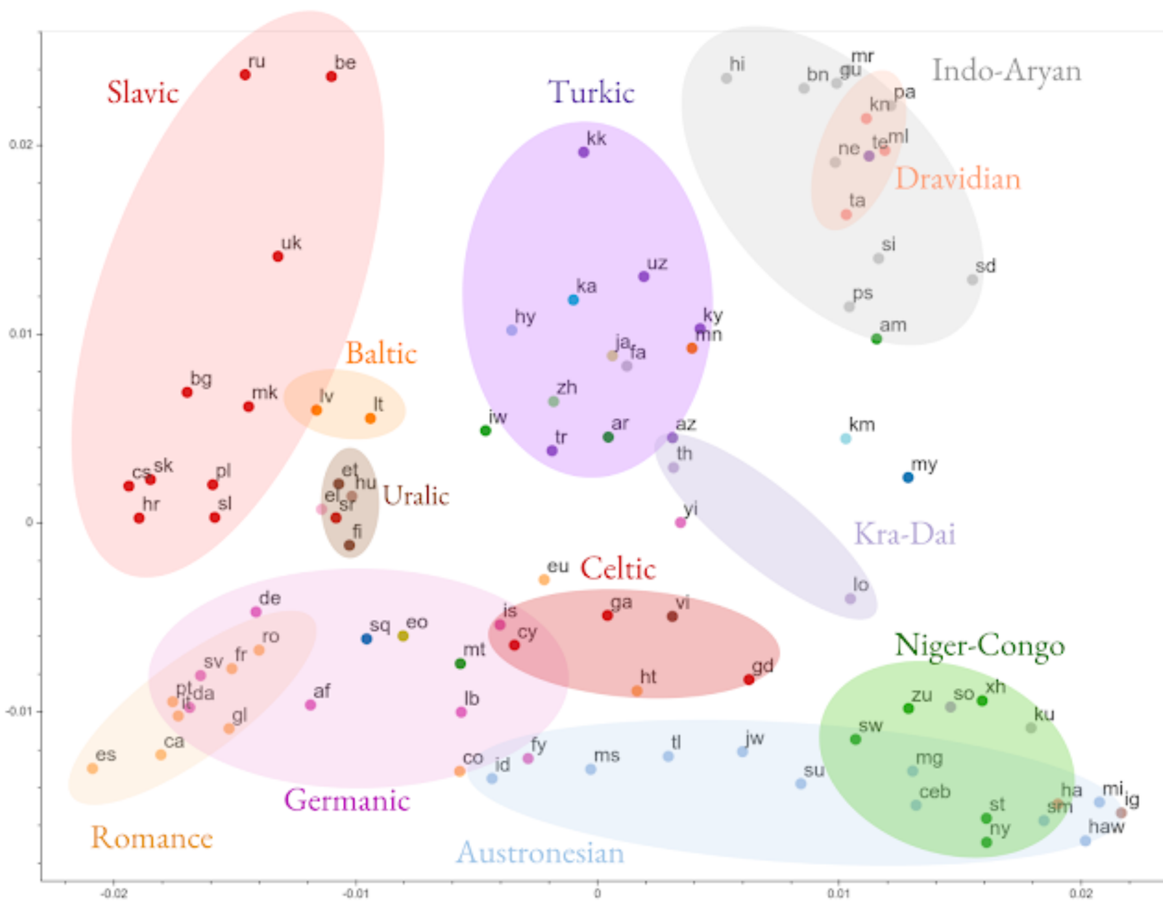
So is Machine Translation solved?

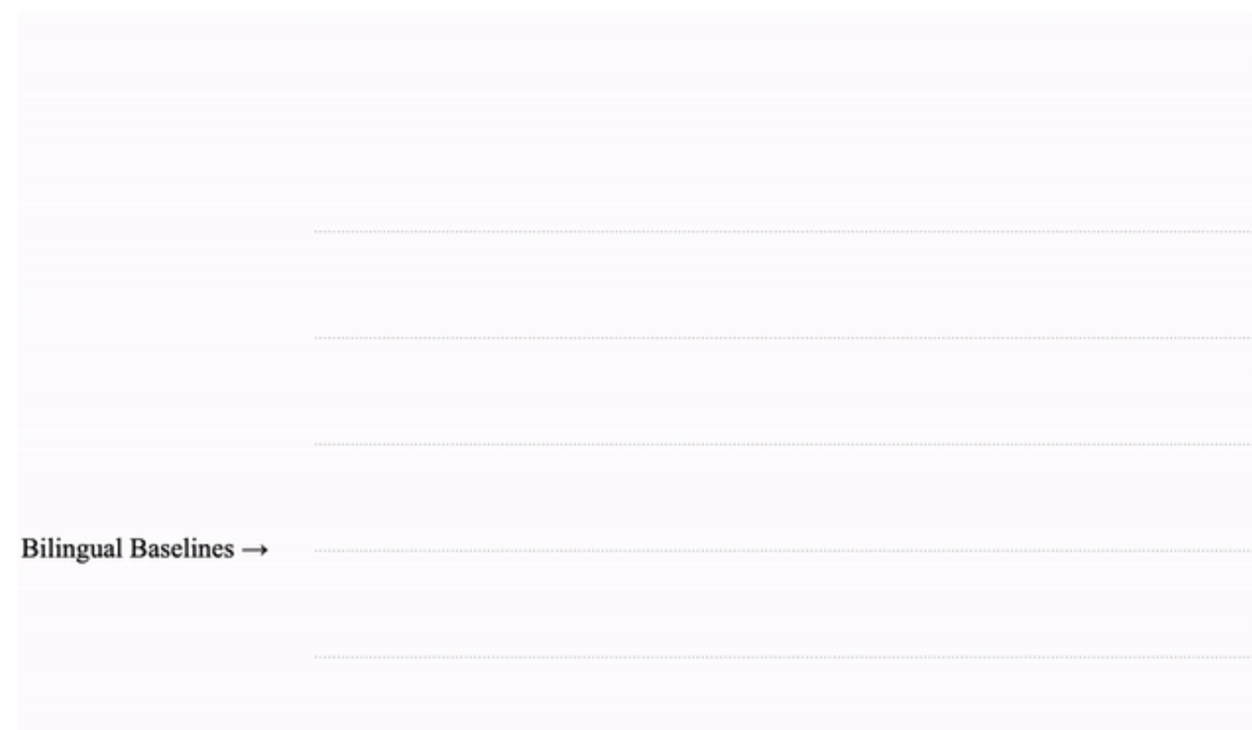
- Nope!
- Uninterpretable systems do strange things



Source: http://languagelog.ldc.upenn.edu/nll/?p=35120#more-35120

Massively multilingual MT





- Train a *single* neural network on 103 languages paired with English (remember Interlingua?)
- Massive improvements on low-resource languages

(Arivazhagan et al., 2019)