

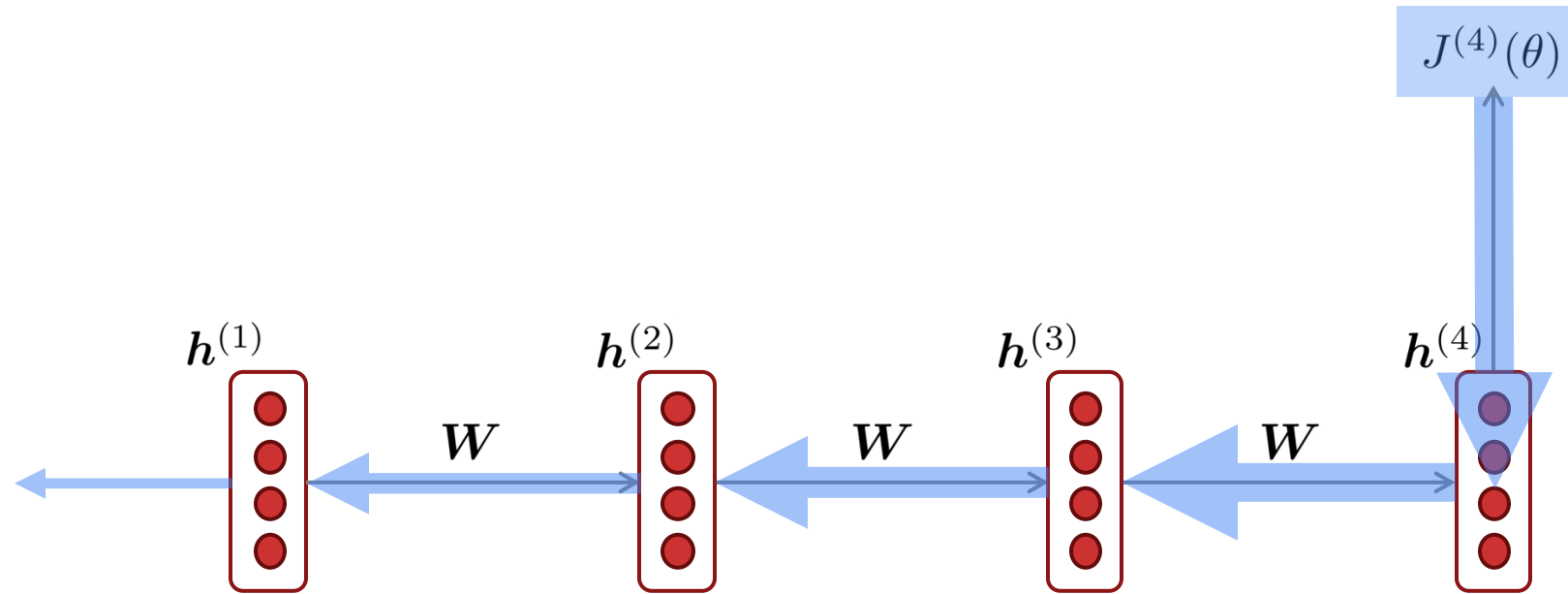
Natural Language Processing with Deep Learning

RNNs, LSTM and Neural Machine Translation

Recap

- **Language Model**: A system that predicts the next word
- **Recurrent Neural Network**: A family of neural networks that:
 - Take sequential input of any length; apply the same weights on each step
 - Can optionally produce output on each step
- **Recurrent Neural Network \neq Language Model**
 - RNNs can be used for many other things (see later)
- **Language Modeling** is a traditional subcomponent of many NLP tasks, all those involving generating text or estimating the probability of text:
 - Now everything in NLP is being rebuilt upon Language Modeling: GPT-3 is an LM!

Our starting point: vanishing gradients




$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

What happens if these are small?

Vanishing gradient problem:
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Long Short-Term Memory RNNs (LSTMs)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the problem of vanishing gradients
 - Everyone cites that paper but really a crucial part of the modern LSTM is from Gers et al. (2000) 
- Only started to be recognized as promising through the work of S's student Alex Graves c. 2006
 - Work in which he also invented CTC (connectionist temporal classification) for speech recognition
- But only really became well-known after Hinton brought it to Google in 2013
 - Following Graves having been a postdoc with Hinton

Hochreiter and Schmidhuber, 1997. Long short-term memory. <https://www.bioinf.jku.at/publications/older/2604.pdf>

Gers, Schmidhuber, and Cummins, 2000. Learning to Forget: Continual Prediction with LSTM. <https://dl.acm.org/doi/10.1162/089976600300015015>

Graves, Fernandez, Gomez, and Schmidhuber, 2006. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural nets. https://www.cs.toronto.edu/~graves/icml_2006.pdf

Long Short-Term Memory RNNs (LSTMs)

- On step t , there is a **hidden state** $\mathbf{h}^{(t)}$ and a **cell state** $\mathbf{c}^{(t)}$
 - Both are vectors length n
 - The cell stores **long-term information**
 - The LSTM can **read**, **erase**, and **write** information from the cell
 - The cell becomes conceptually rather like RAM in a computer
- The selection of which information is erased/written/read is controlled by three corresponding **gates**
 - The gates are also vectors of length n
 - On each timestep, each element of the gates can be **open** (1), **closed** (0), or somewhere in-between
 - The gates are **dynamic**: their value is computed based on the current context

Long Short-Term Memory (LSTM)

We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep t :

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase (“forget”) some content from last cell state, and write (“input”) some new cell content

Hidden state: read (“output”) some content from the cell

Sigmoid function: all gate values are between 0 and 1

$$f^{(t)} = \sigma \left(W_f h^{(t-1)} + U_f x^{(t)} + b_f \right)$$

$$i^{(t)} = \sigma \left(W_i h^{(t-1)} + U_i x^{(t)} + b_i \right)$$

$$o^{(t)} = \sigma \left(W_o h^{(t-1)} + U_o x^{(t)} + b_o \right)$$

$$\tilde{c}^{(t)} = \tanh \left(W_c h^{(t-1)} + U_c x^{(t)} + b_c \right)$$

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

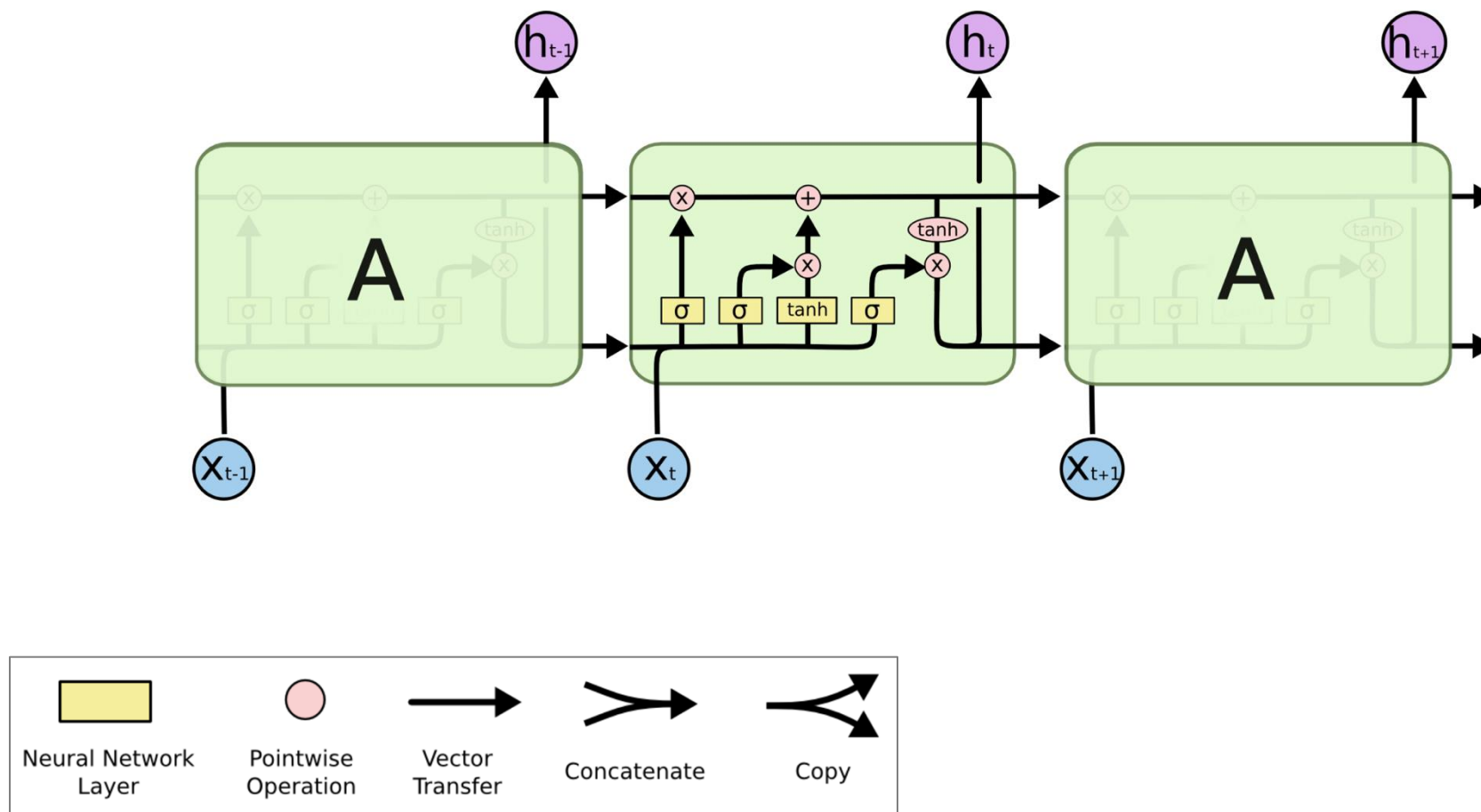
$$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$

All these are vectors of same length n

Gates are applied using element-wise (or Hadamard) product: \odot

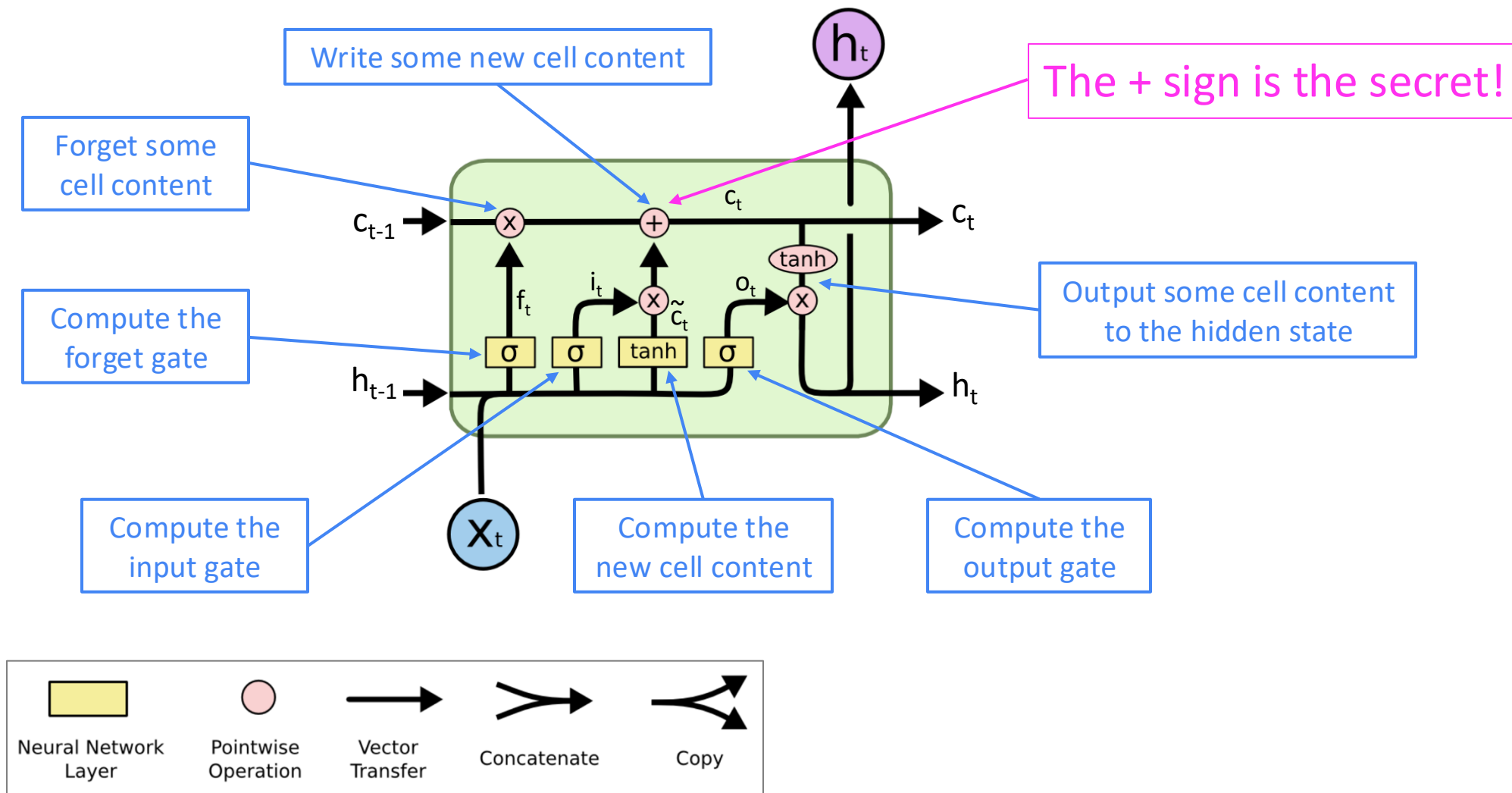
Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



How does LSTM solve vanishing gradients?

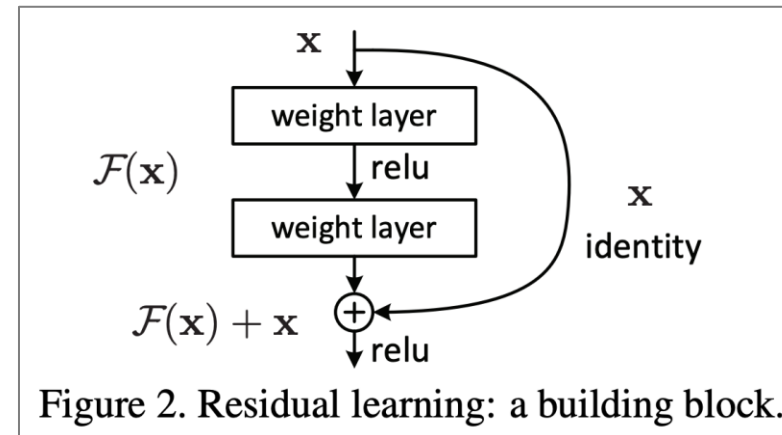
- The LSTM architecture makes it much easier for an RNN to preserve information over many timesteps
 - e.g., if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely.
 - In contrast, it's harder for a vanilla RNN to learn a recurrent weight matrix W_h that preserves info in the hidden state
 - In practice, you get about 100 timesteps rather than about 7
- However, there are alternative ways of creating more direct and linear pass-through connections in models for long distance dependencies

Is vanishing/exploding gradient just an RNN problem?

- No! It can be a problem for all neural architectures (including **feed-forward** and **convolutional**), especially **very deep** ones.
 - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
 - Thus, lower layers are learned very slowly (i.e., are hard to train)
- Another solution: lots of new deep feedforward/convolutional architectures **add more direct connections** (thus allowing the gradient to flow)

For example:

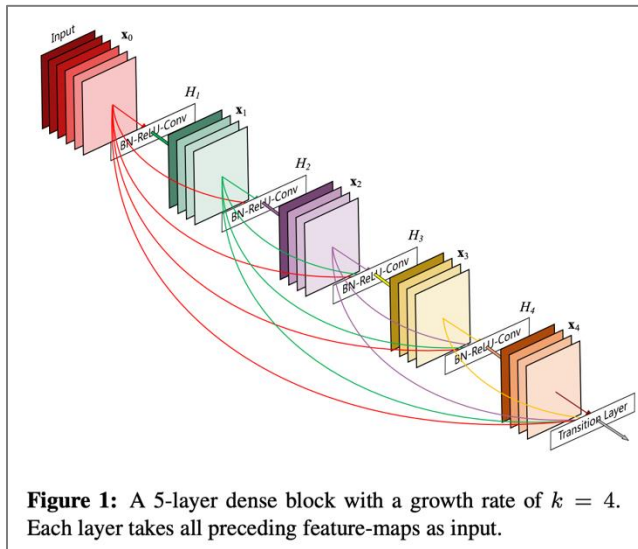
- **Residual connections** aka “ResNet”
- Also known as **skip-connections**
- The **identity connection** **preserves information** by default
- This makes **deep** networks much **easier to train**



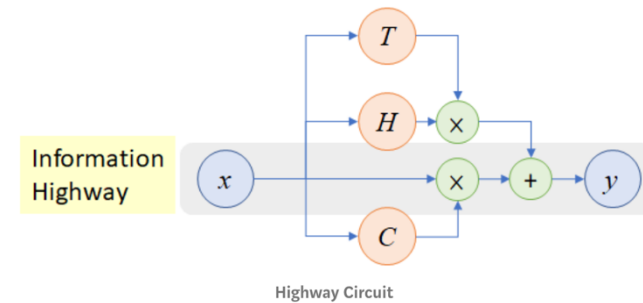
Is vanishing/exploding gradient just a RNN problem?

Other methods:

- **Dense connections** aka “DenseNet”
- Directly connect each layer to all future layers!



- **Highway connections** aka “HighwayNet”
- Similar to residual connections, but the identity connection vs the transformation layer is controlled by a **dynamic gate**
- Inspired by LSTMs, but applied to deep feedforward/convolutional networks



- **Conclusion:** Though vanishing/exploding gradients are a general problem, **RNNs are particularly unstable** due to the repeated multiplication by the **same** weight matrix [Bengio et al, 1994]

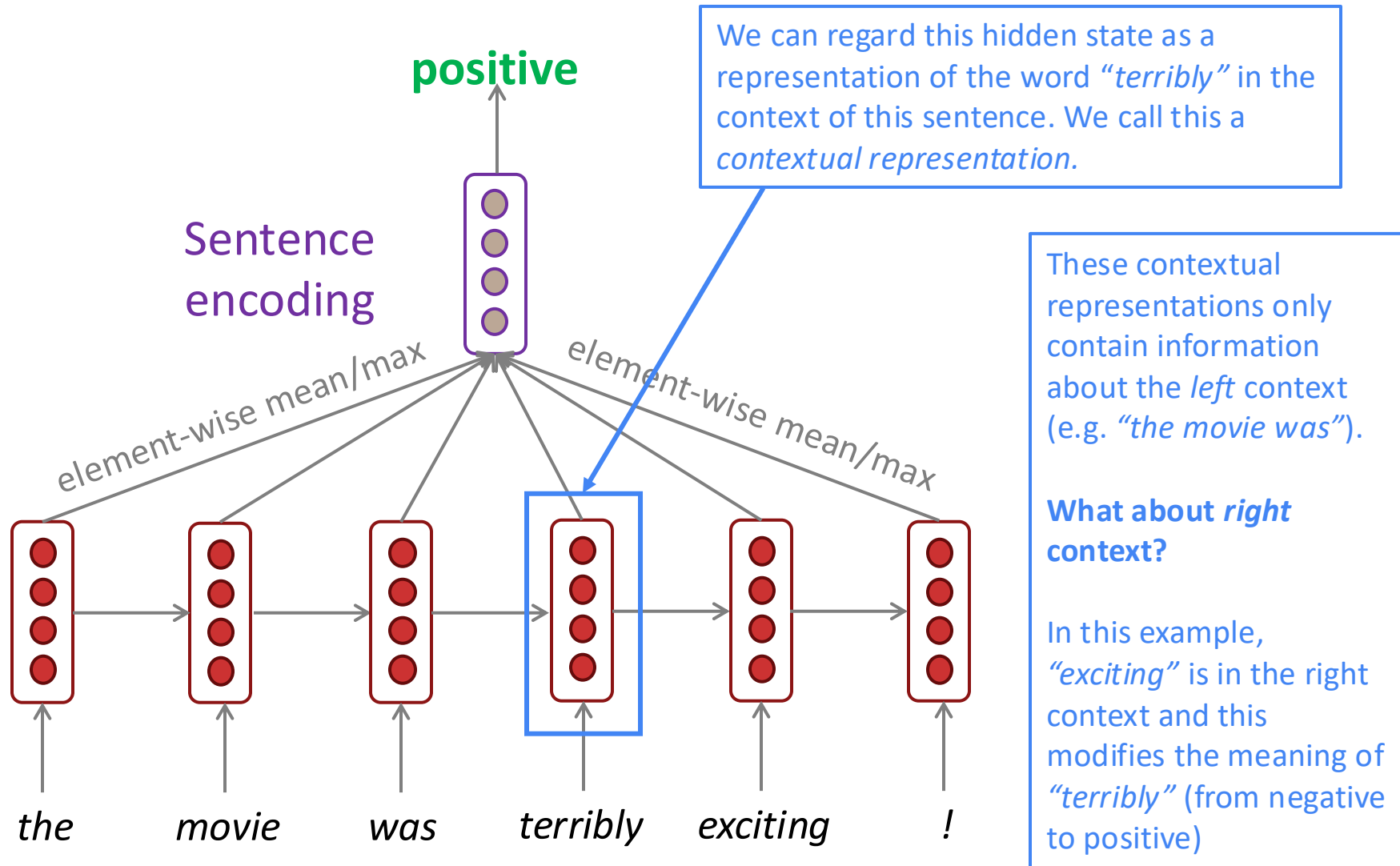
LSTMs: real-world success

- In 2013–2015, LSTMs started achieving state-of-the-art results
 - Successful tasks include handwriting recognition, speech recognition, machine translation, parsing, and image captioning, as well as language models
 - LSTMs became the dominant approach for most NLP tasks
- Recently (2019–2024), Transformers have become dominant for all tasks
 - For example, in WMT (a Machine Translation conference + competition):
 - In WMT 2014, there were 0 neural machine translation systems (!)
 - In WMT 2016, the summary report contains “RNN” 44 times (and these systems won)
 - In WMT 2019: “RNN” 7 times, “Transformer” 105 times
- Now, ‘State space models’ (RNN++) are making a comeback

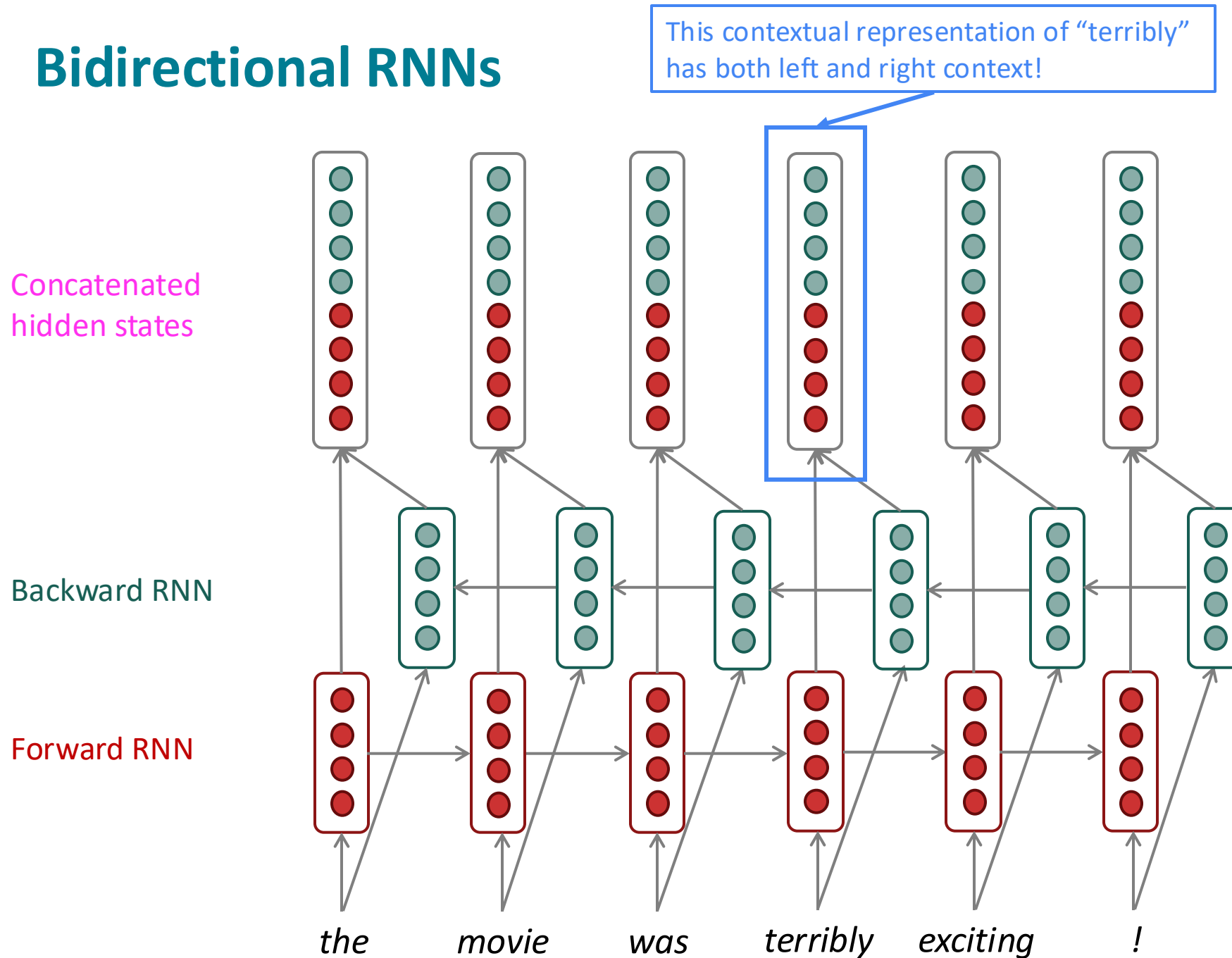
Source: "Findings of the 2016 Conference on Machine Translation (WMT16)", Bojar et al. 2016, <http://www.statmt.org/wmt16/pdf/W16-2301.pdf>
Source: "Findings of the 2018 Conference on Machine Translation (WMT18)", Bojar et al. 2018, <http://www.statmt.org/wmt18/pdf/WMT028.pdf>
Source: "Findings of the 2019 Conference on Machine Translation (WMT19)", Barrault et al. 2019, <http://www.statmt.org/wmt18/pdf/WMT028.pdf>

4. Bidirectional and Multi-layer RNNs: motivation

Task: Sentiment Classification



Bidirectional RNNs



Bidirectional RNNs

On timestep t :

This is a general notation to mean “compute one forward step of the RNN” – it could be a simple RNN or LSTM computation.

Forward RNN $\vec{h}^{(t)} = \text{RNN}_{\text{FW}}(\vec{h}^{(t-1)}, \mathbf{x}^{(t)})$

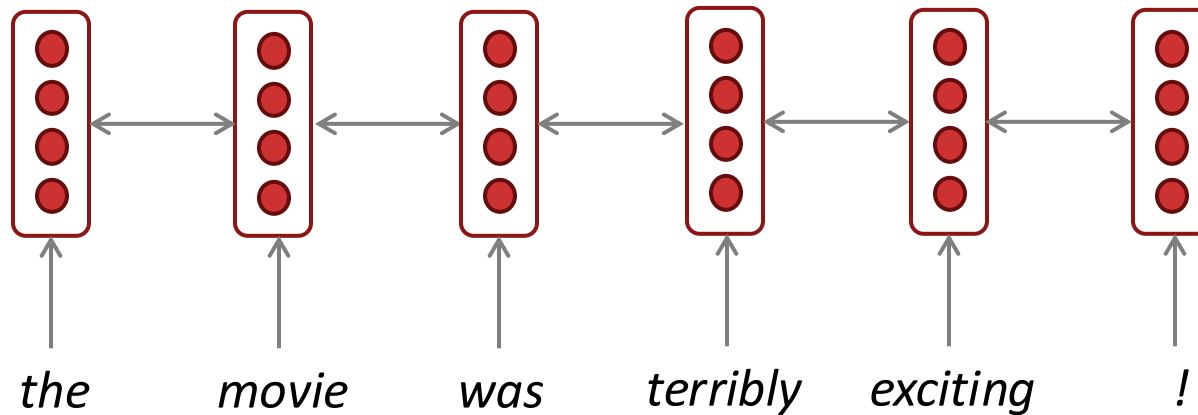
Backward RNN $\overleftarrow{h}^{(t)} = \text{RNN}_{\text{BW}}(\overleftarrow{h}^{(t+1)}, \mathbf{x}^{(t)})$

Generally, these two RNNs have separate weights

Concatenated hidden states $\mathbf{h}^{(t)} = [\vec{h}^{(t)}; \overleftarrow{h}^{(t)}]$

We regard this as “the hidden state” of a bidirectional RNN. This is what we pass on to the next parts of the network.

Bidirectional RNNs: simplified diagram



The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states

Bidirectional RNNs

- Note: bidirectional RNNs are only applicable if you have access to the **entire input sequence**
 - They are **not** applicable to Language Modeling, because in LM you *only* have left context available.
- If you do have entire input sequence (e.g., any kind of encoding), **bidirectionality is powerful** (you should use it by default).
- For example, **BERT** (**Bidirectional** Encoder Representations from Transformers) is a powerful pretrained contextual representation system **built on bidirectionality**.
 - You will learn more about **transformers**, including BERT, in a couple of weeks!

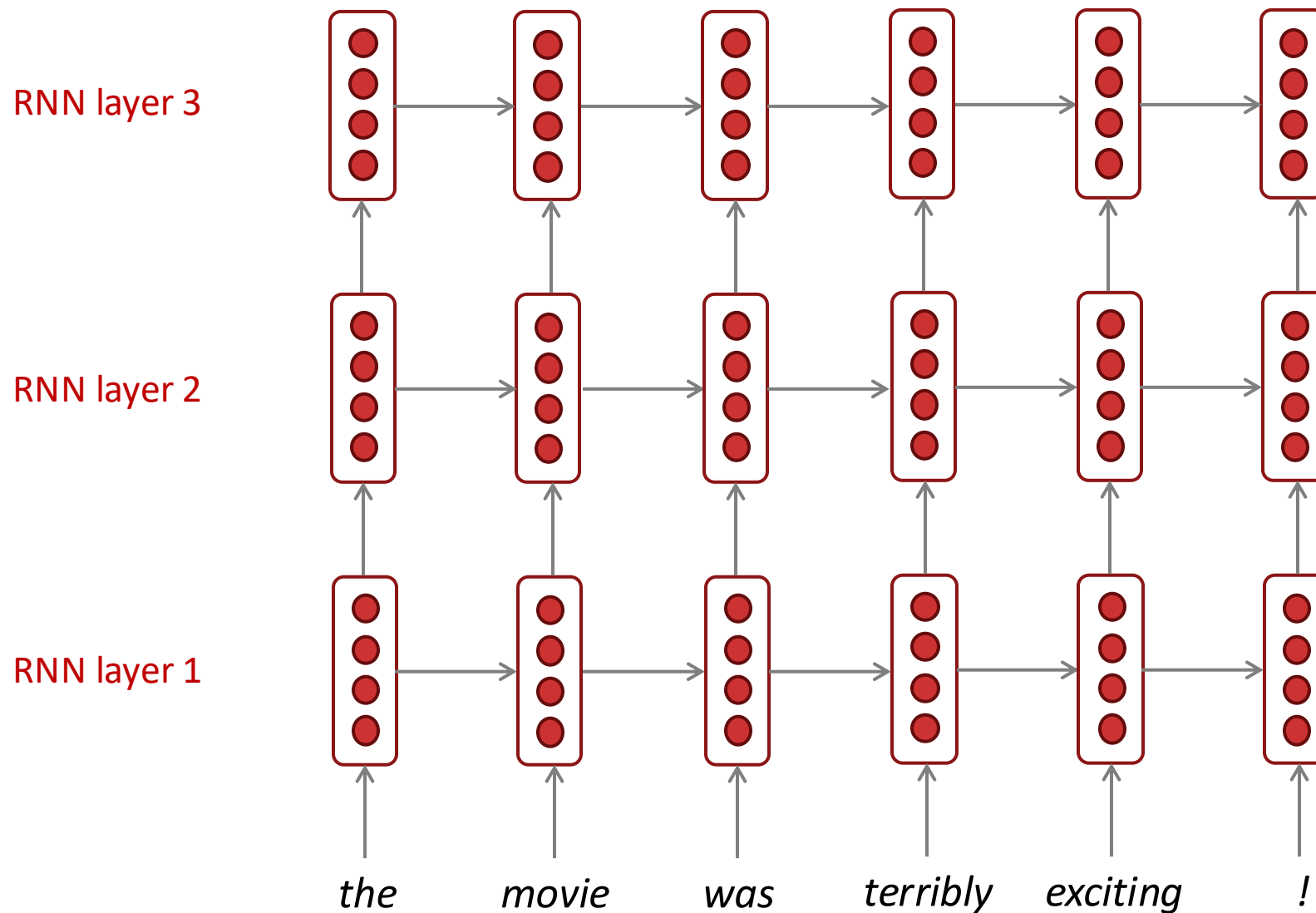
Multi-layer RNNs

- 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 RNNs

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



Multi-layer RNNs in practice

- Multi-layer or stacked RNNs allow a network to compute **more complex representations**
 - they work better than just have one layer of high-dimensional encodings!
 - The **lower RNNs** should **compute lower-level features** and the **higher RNNs** should compute **higher-level features**.
- **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

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 research began in the **early 1950s** on machines less powerful than high school calculators (before term “A.I.” coined!)
- Concurrent with foundational work on automata, formal languages, probabilities, and information theory
- MT heavily funded by military, but basically just simple rule-based systems doing word substitution
- Human language is more complicated than that, and varies more across languages!
- Little understanding of natural language syntax, semantics, pragmatics
- Problem soon appeared intractable

1 minute video showing 1954 MT:

<https://youtu.be/K-HfpsHPmvw>

The early history of MT: 1950s



1990s-2010s: Statistical Machine Translation

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

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

- Use Bayes Rule to break this down into **two components** to be learned separately:

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

Translation Model

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

Language Model

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

1990s–2010s: Statistical Machine Translation

- SMT was a huge research field
- The best systems were extremely complex
 - Hundreds of important details
- Systems had many separately-designed subcomponents
 - Lots of feature engineering
 - Need to design features to capture particular language phenomena
 - Required compiling and maintaining extra resources
 - Like tables of equivalent phrases
 - Lots of human effort to maintain
 - Repeated effort for each language pair!

2014

Neural
Machine
Translation

MT research

(dramatic reenactment)

NMT: the first big success story of NLP Deep Learning

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

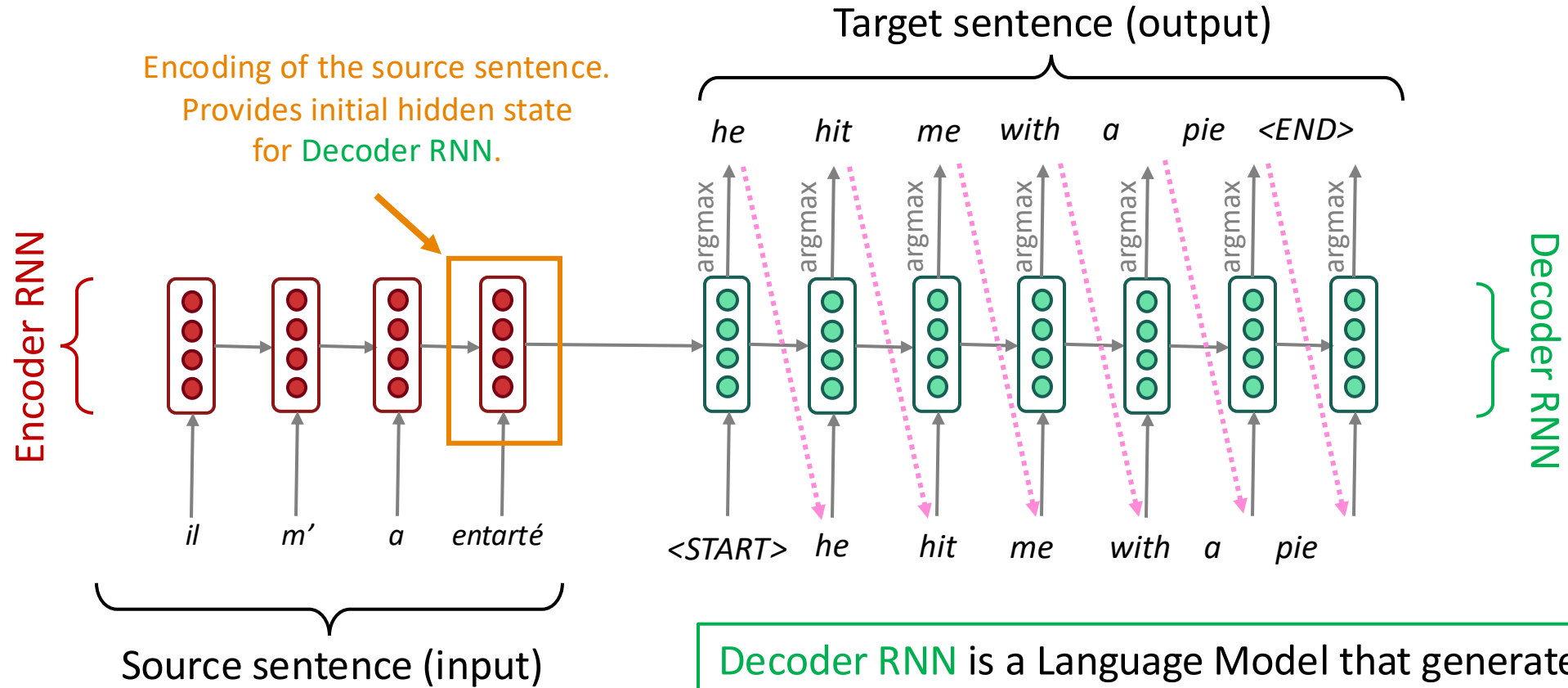
- **2014:** First seq2seq paper published [Sutskever et al. 2014]
- **2016:** Google Translate switches from SMT to NMT – and by 2018 everyone has



- This is amazing!
 - **SMT** systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by small groups of engineers in a few months

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!

- The general notion here is an **encoder-decoder** model
 - One neural network takes input and produces a neural representation
 - Another network produces output based on that neural representation
 - If the input and output are sequences, we call it a seq2seq model
- 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

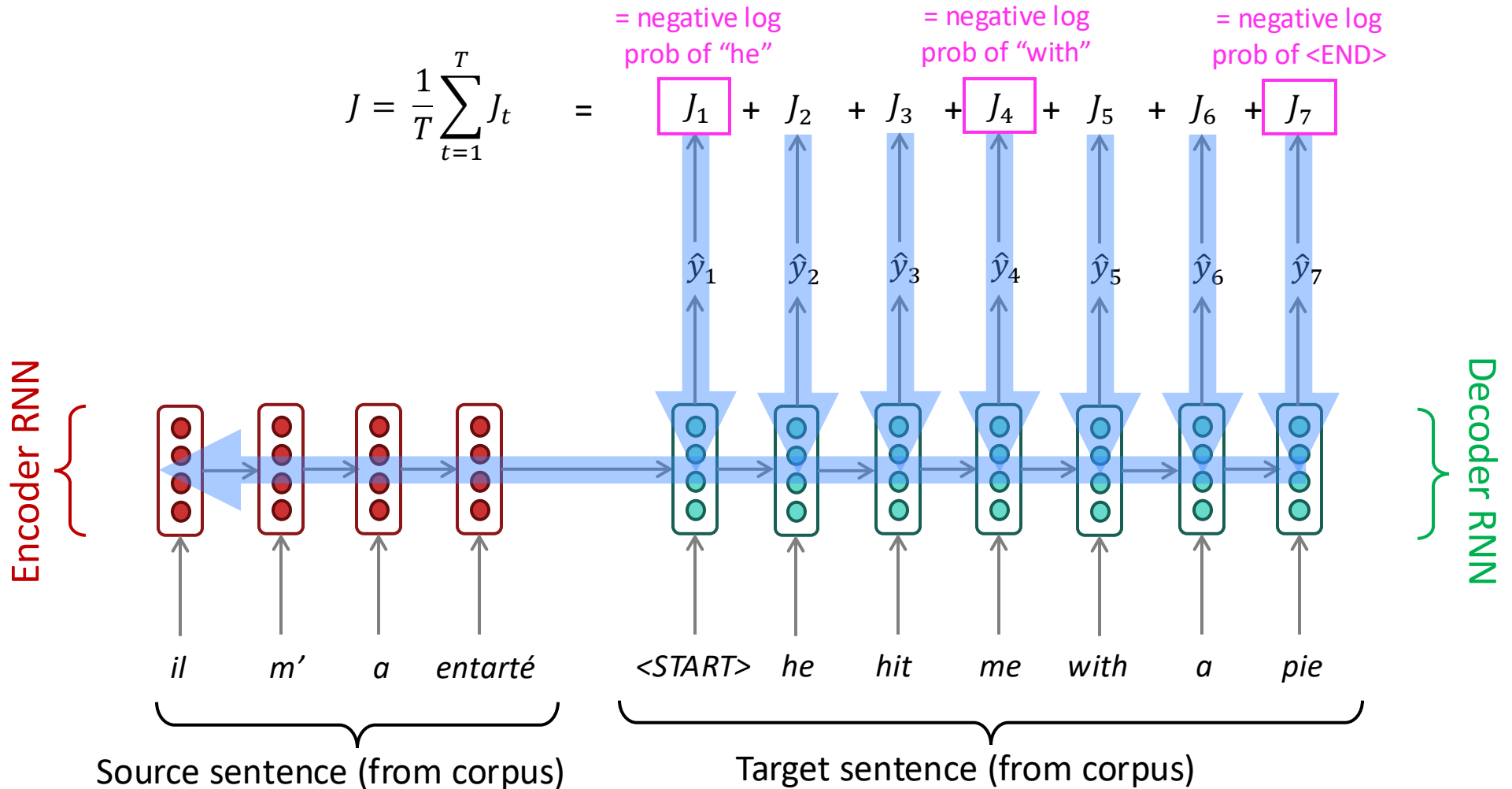
- NMT directly calculates $P(y|x)$:

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

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

- **Question:** How to train an NMT system?
- **(Easy) Answer:** Get a big parallel corpus...
 - But there is now exciting work on “unsupervised NMT”, data augmentation, etc.

Training a Neural Machine Translation system

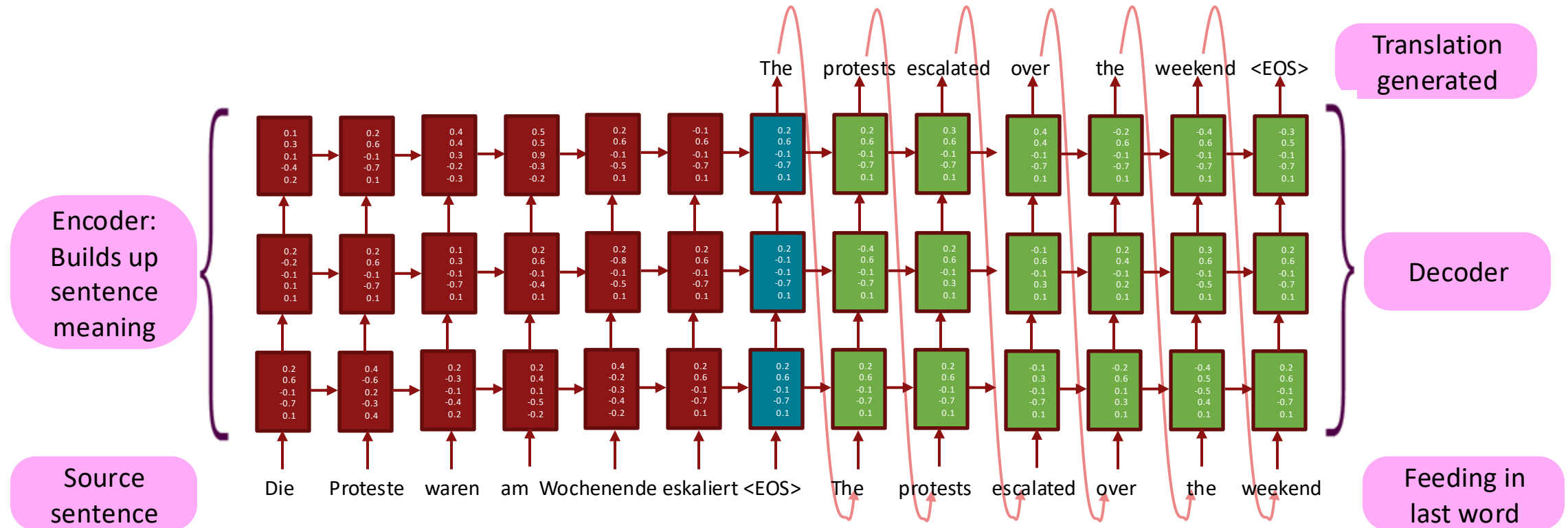


Seq2seq is optimized as a **single system**. Backpropagation operates “*end-to-end*”.

Multi-layer deep encoder-decoder machine translation net

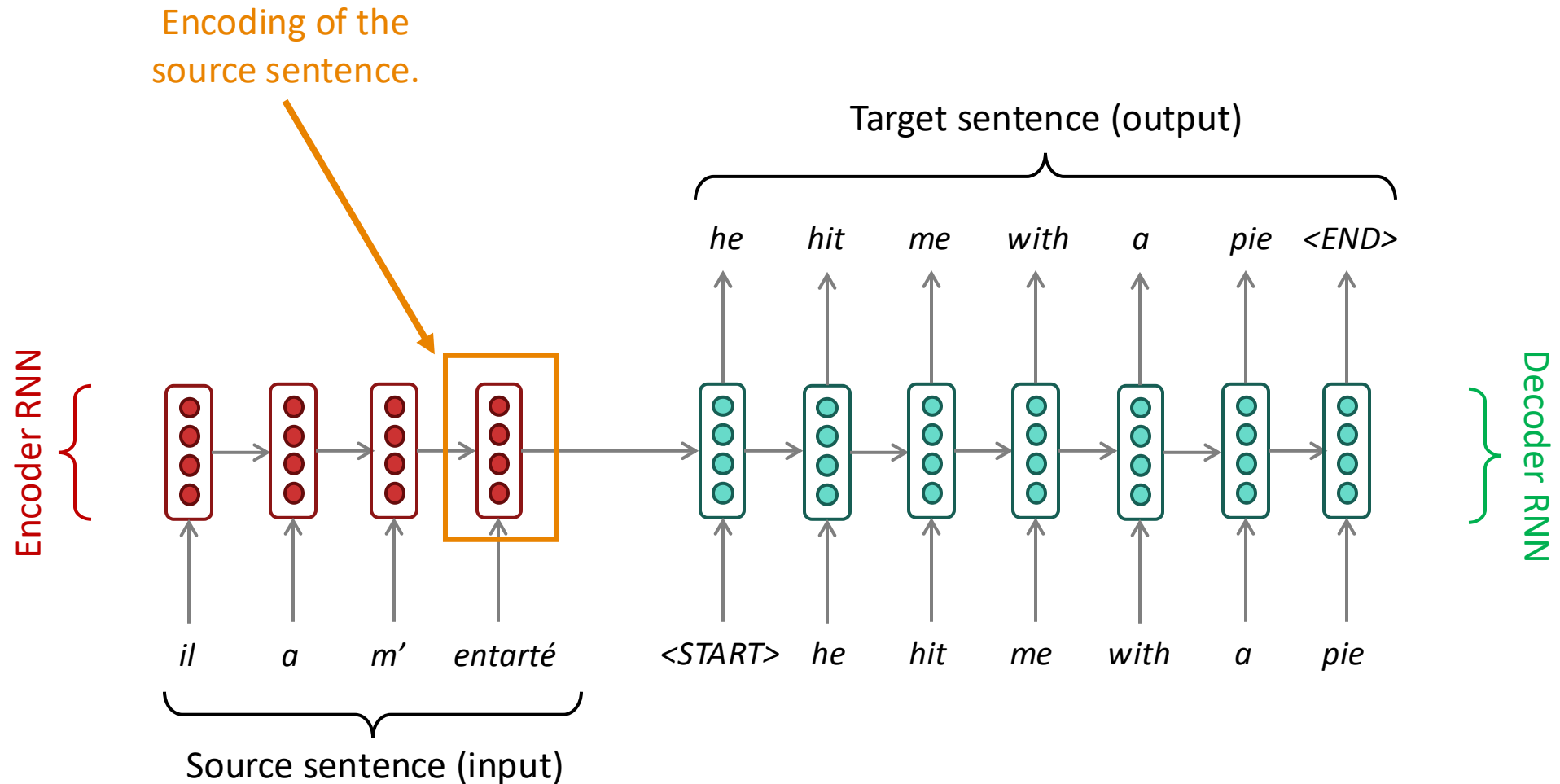
[Sutskever et al. 2014; Luong et al. 2015]

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



Conditioning =
Bottleneck

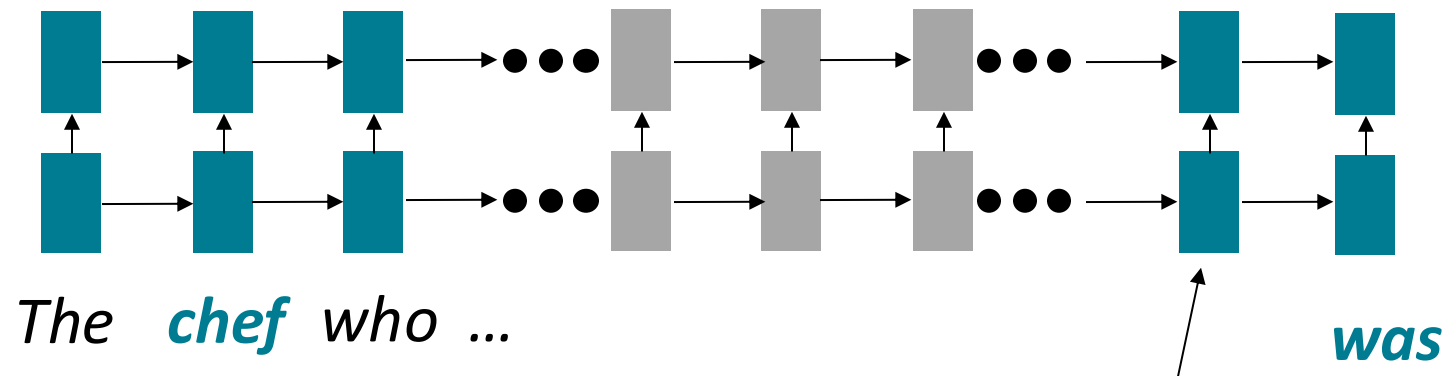
The final piece: the bottleneck problem in RNNs



Problems with this architecture?

Issues with recurrent models: Linear interaction distance

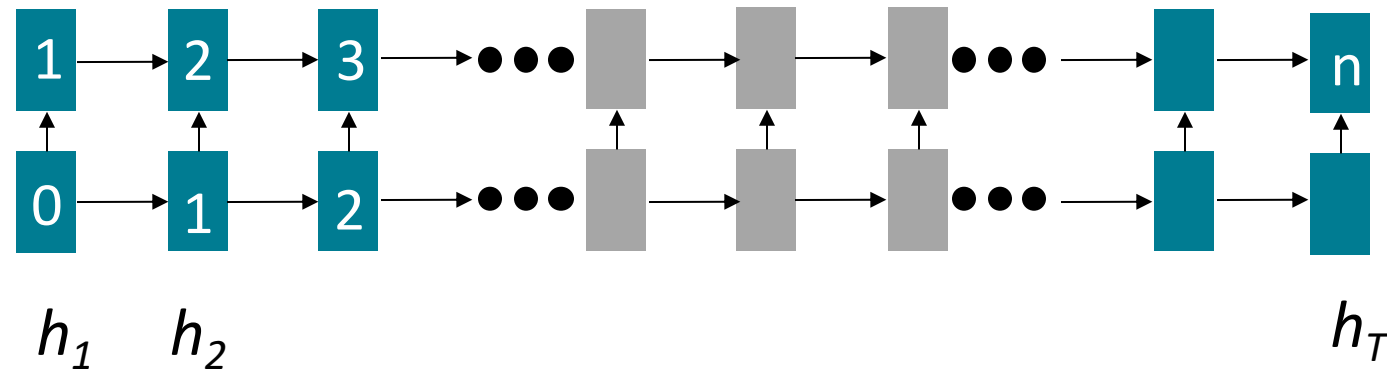
- **$O(\text{sequence length})$** steps for distant word pairs to interact means:
 - Hard to learn long-distance dependencies (because gradient problems!)
 - Linear order of words is “baked in”; we already know linear order isn’t the right way to think about sentences...



Info of *chef* has gone through $O(\text{sequence length})$ many layers!

Issues with recurrent models: Lack of parallelizability

- Forward and backward passes have **$O(\text{sequence length})$** unparallelizable operations
 - GPUs can perform a bunch of independent computations at once!
 - But future RNN hidden states can't be computed in full before past RNN hidden states have been computed
 - Inhibits training on very large datasets!



Numbers indicate min # of steps before a state can be computed

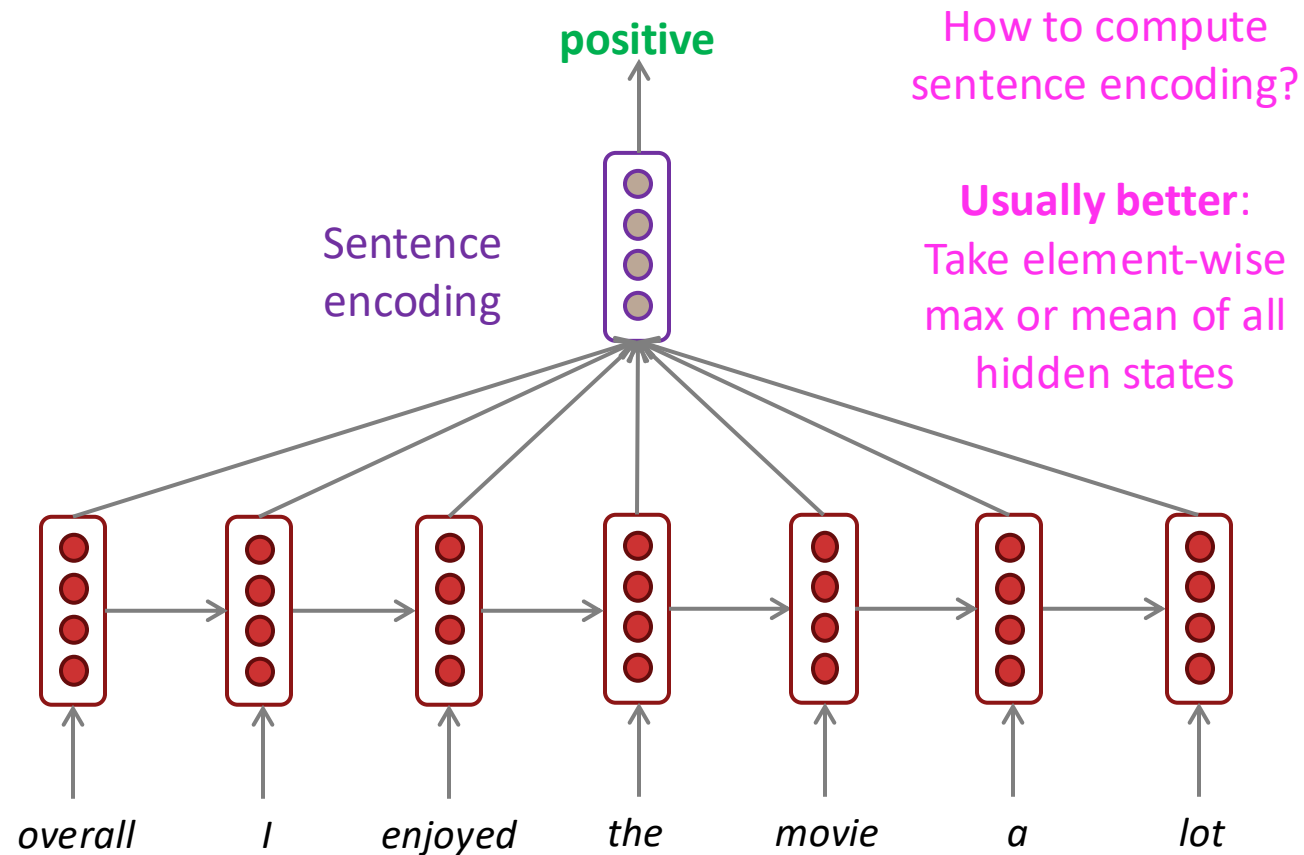
Attention

- **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

The starting point: mean-pooling for RNNs

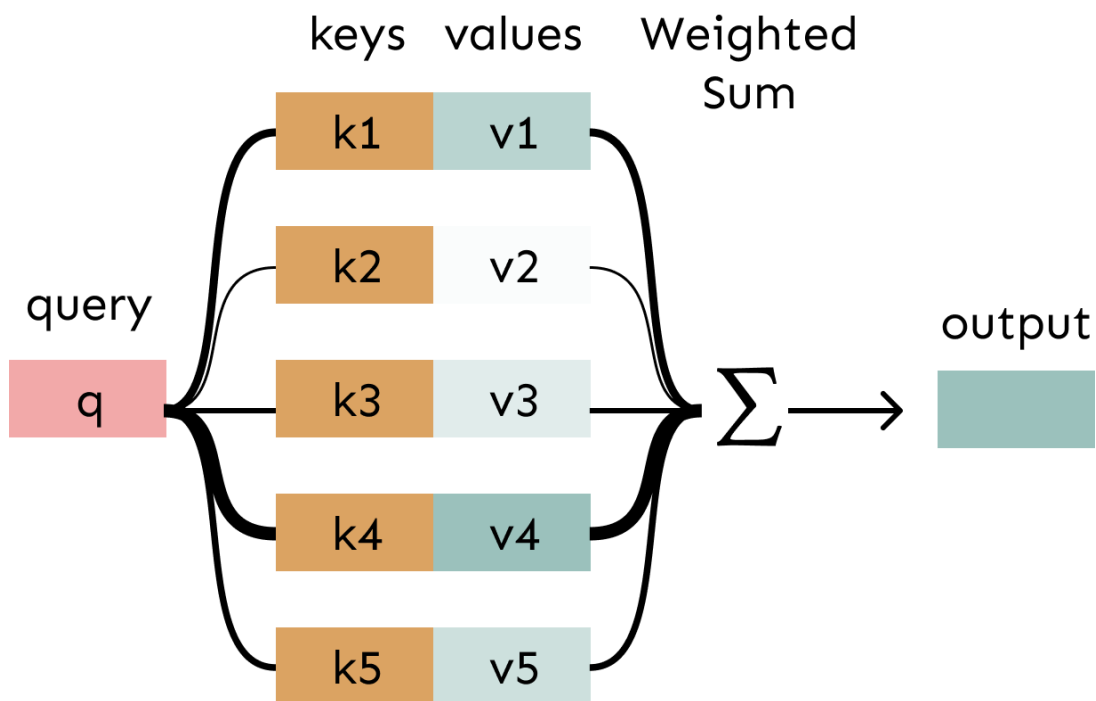


- Starting point: a *very* basic way of 'passing information from the encoder' is to *average*

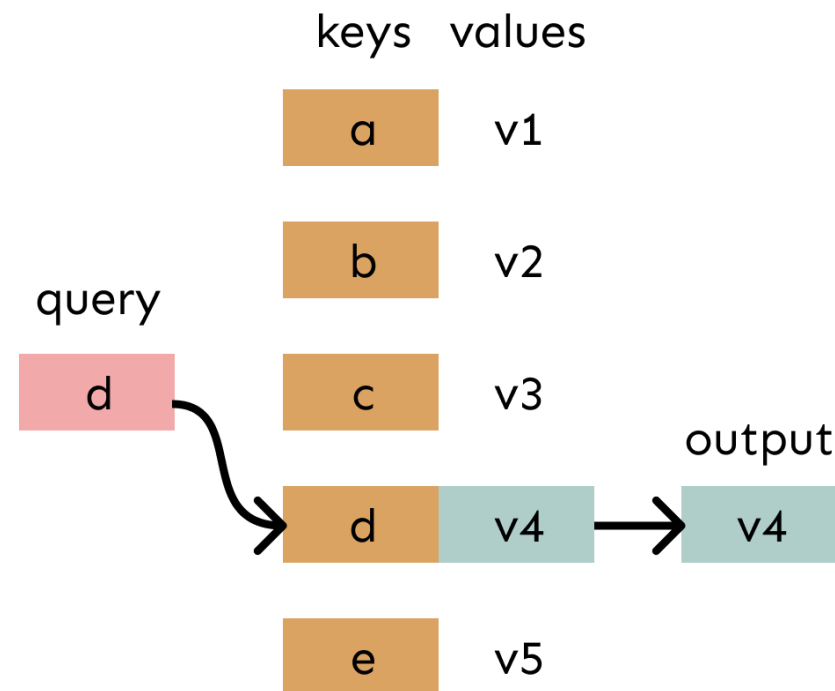
Attention is *weighted* averaging, which lets you do lookups!

Attention is just a **weighted** average – this is very powerful if the weights are learned!

In **attention**, the **query** matches all **keys** *softly*, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.

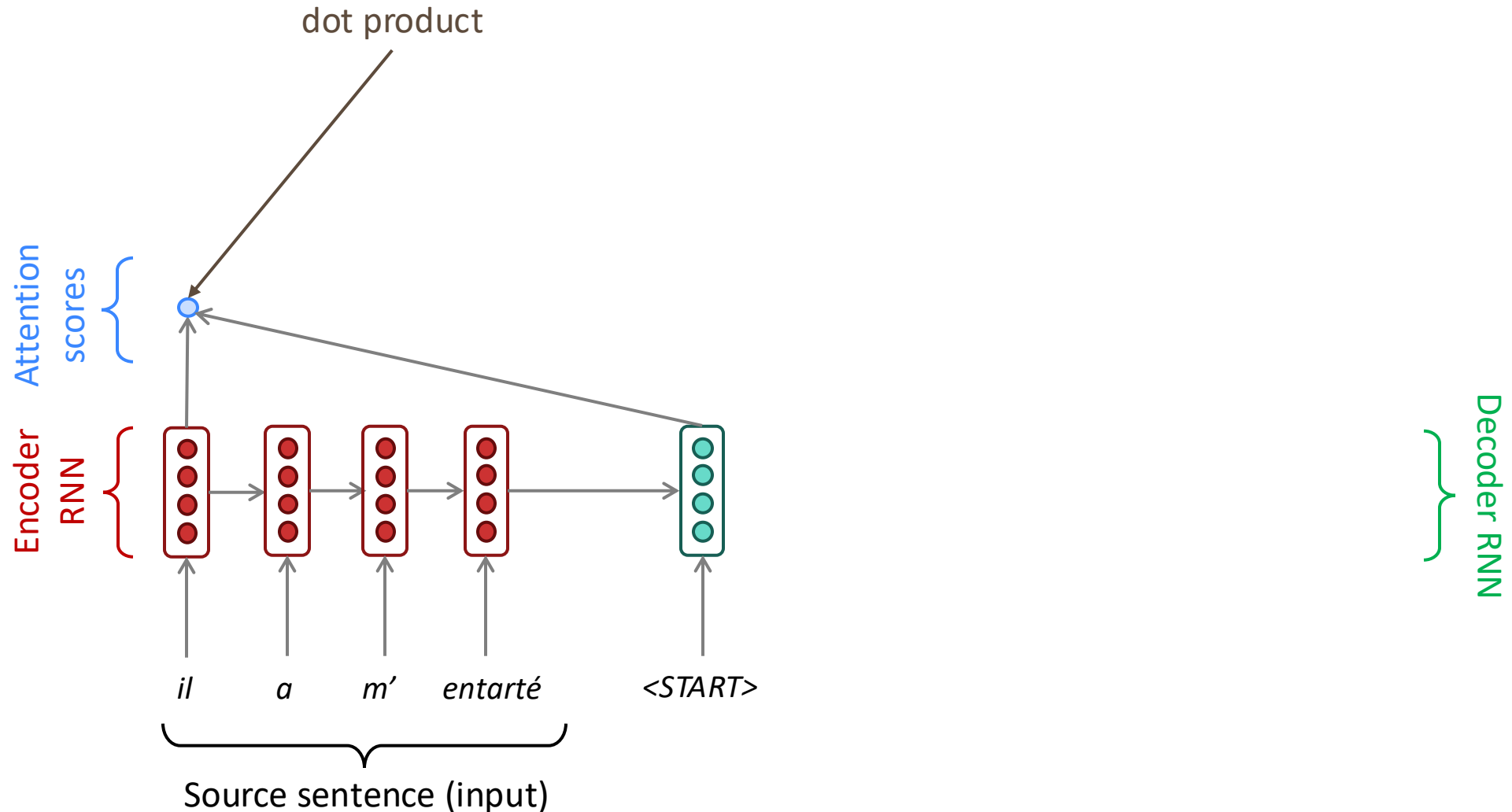


In a **lookup table**, we have a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.

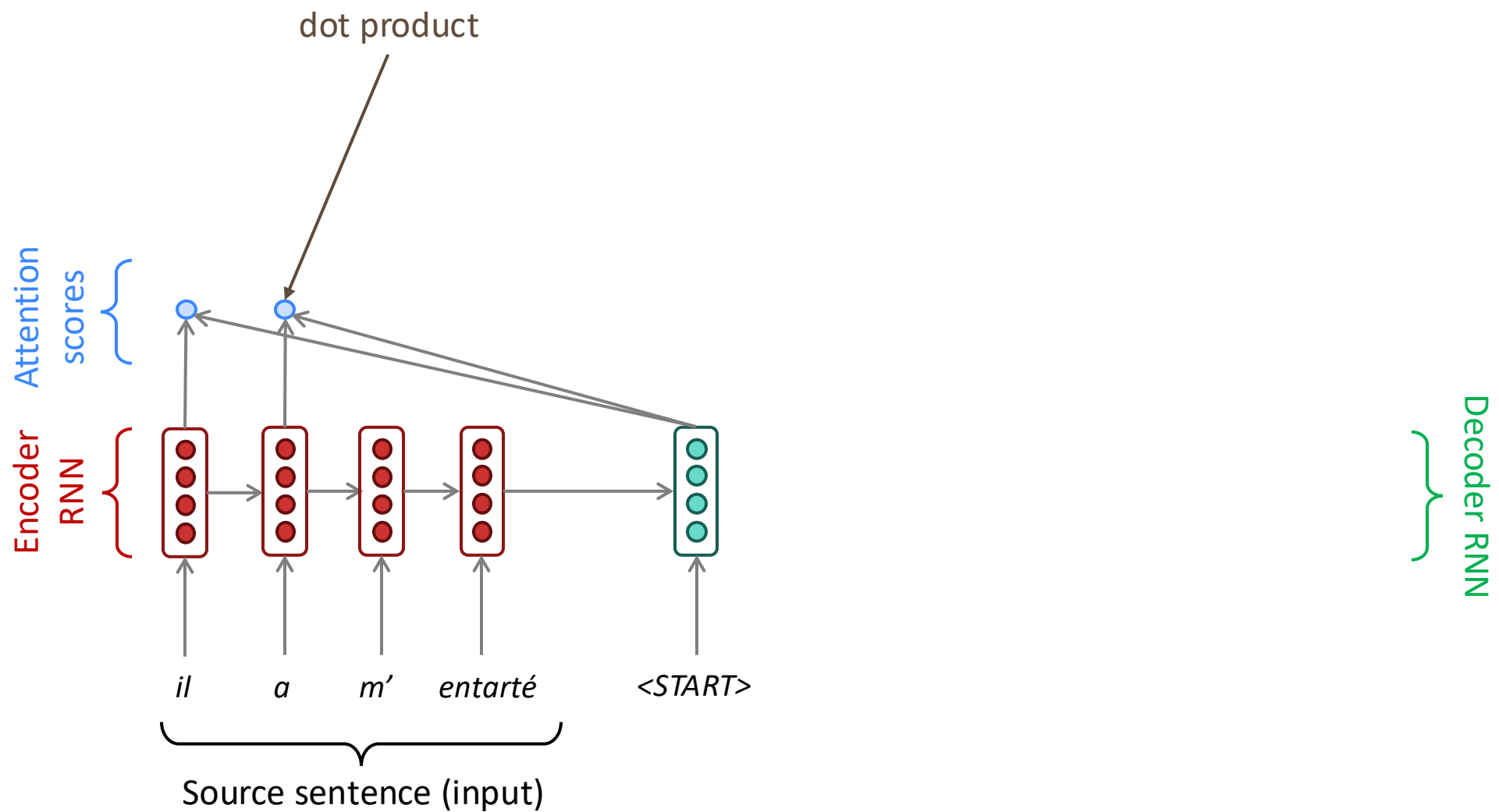


Sequence-to-sequence with attention

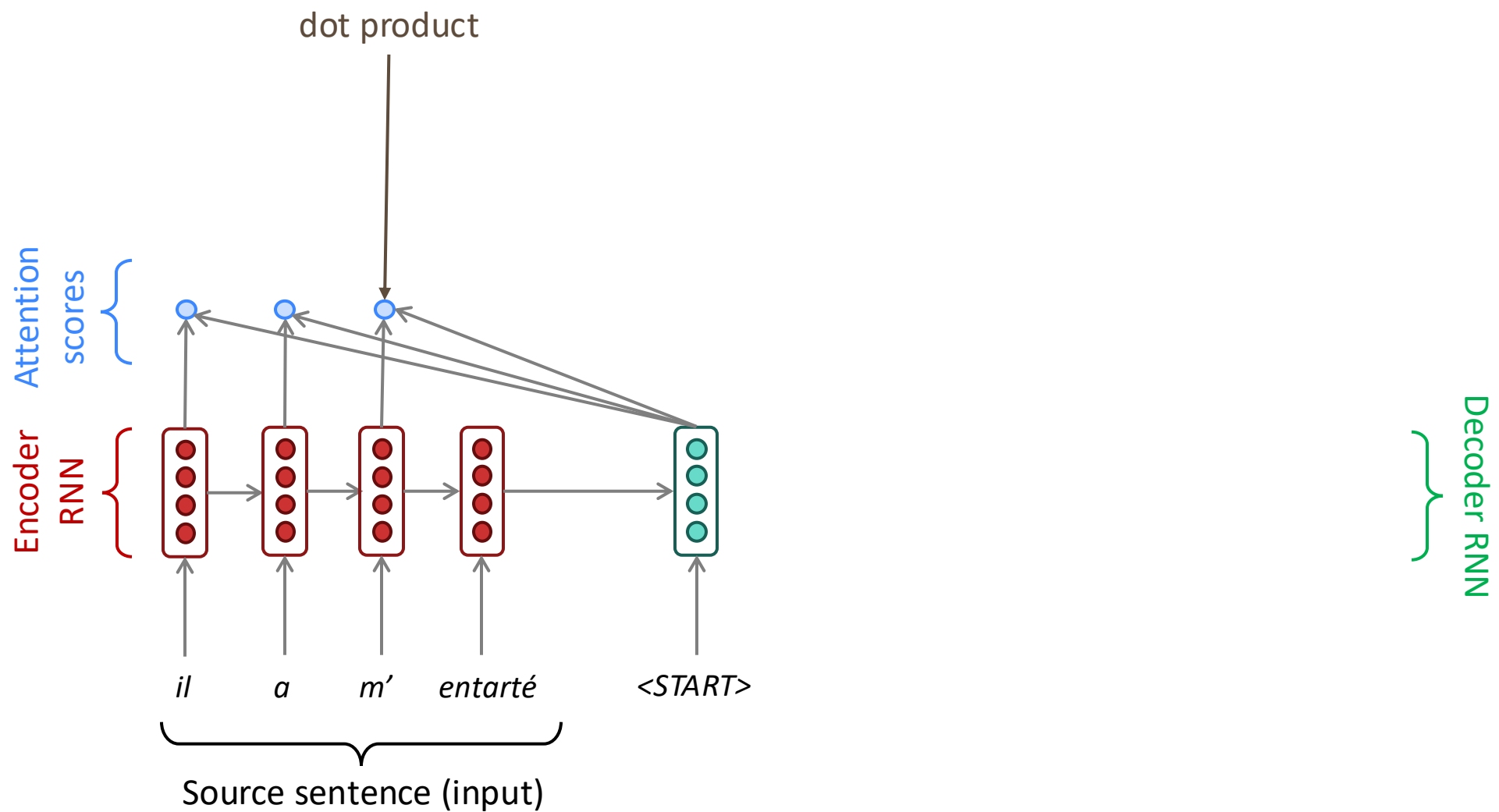
Core idea: on each step of the decoder, *use direct connection to the encoder to focus on a particular part* of the source sequence



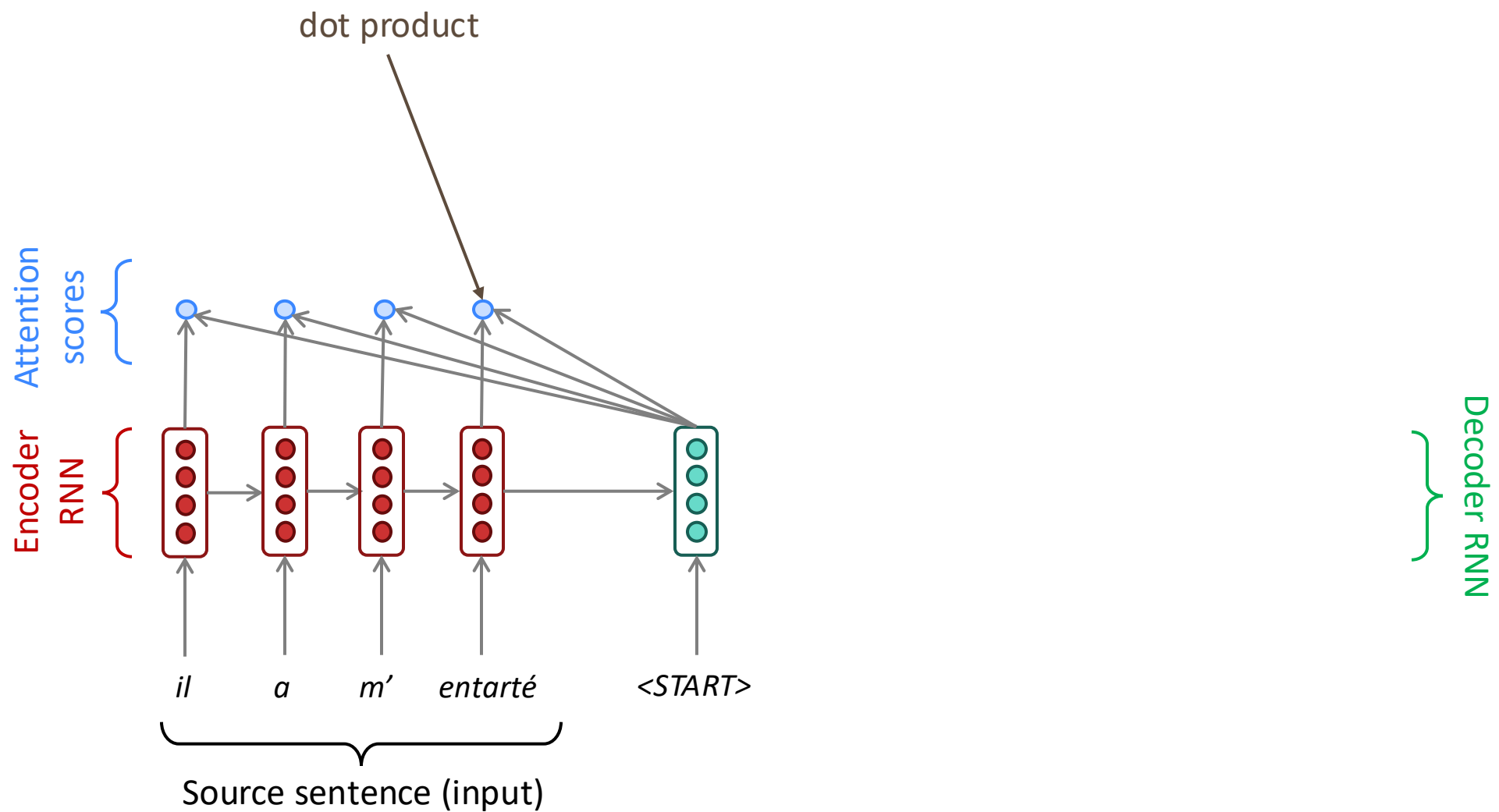
Sequence-to-sequence with attention



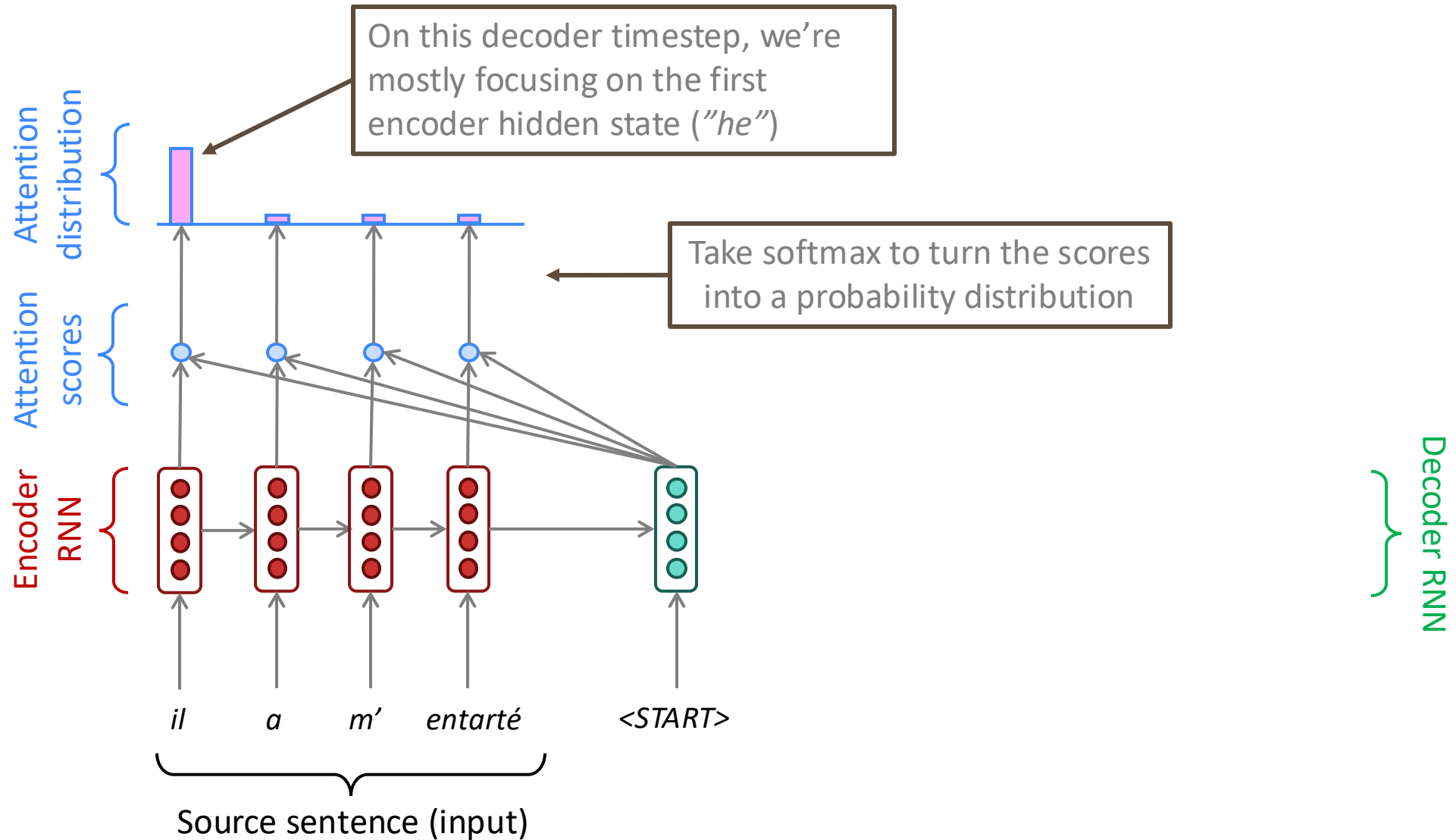
Sequence-to-sequence with attention



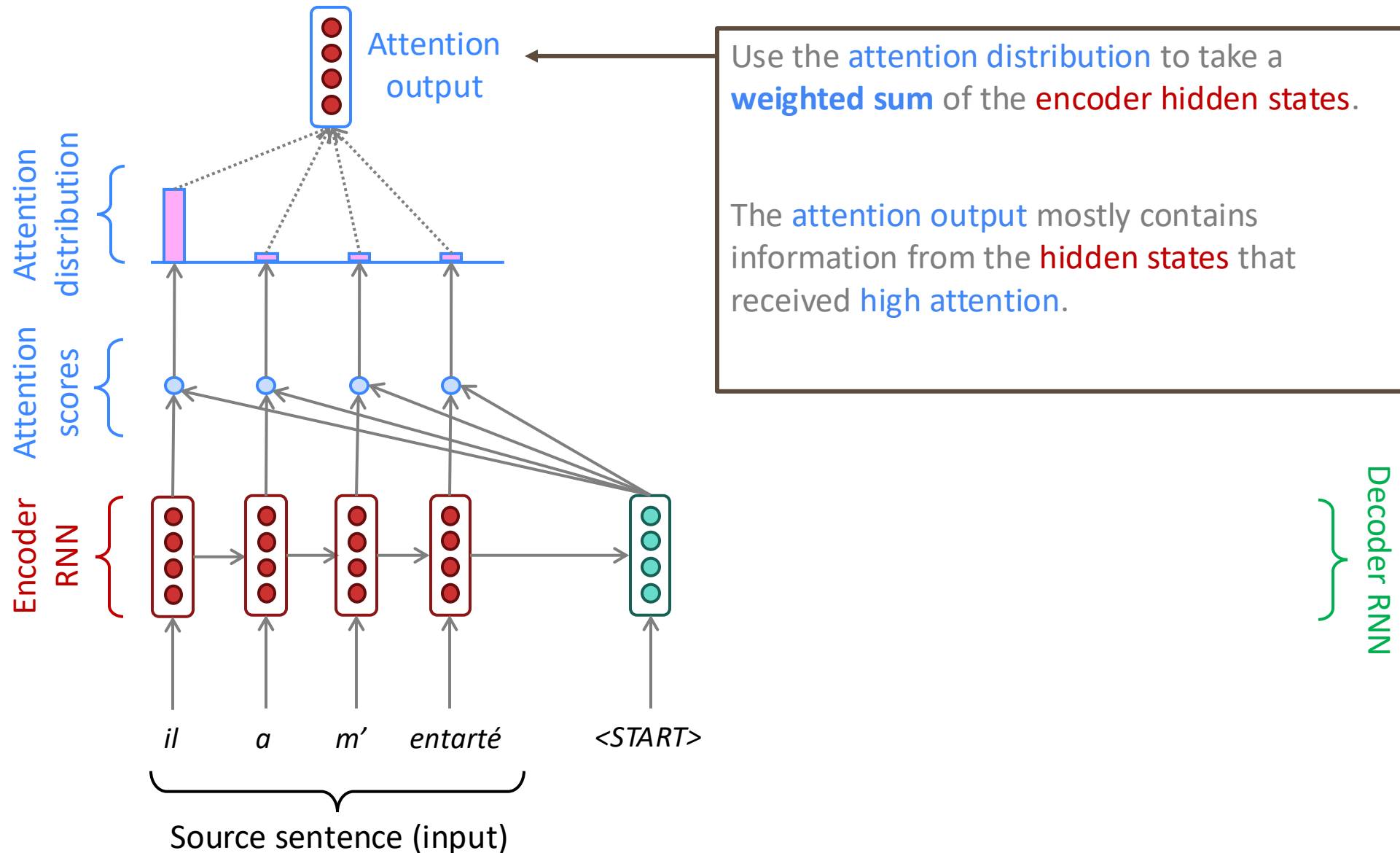
Sequence-to-sequence with attention



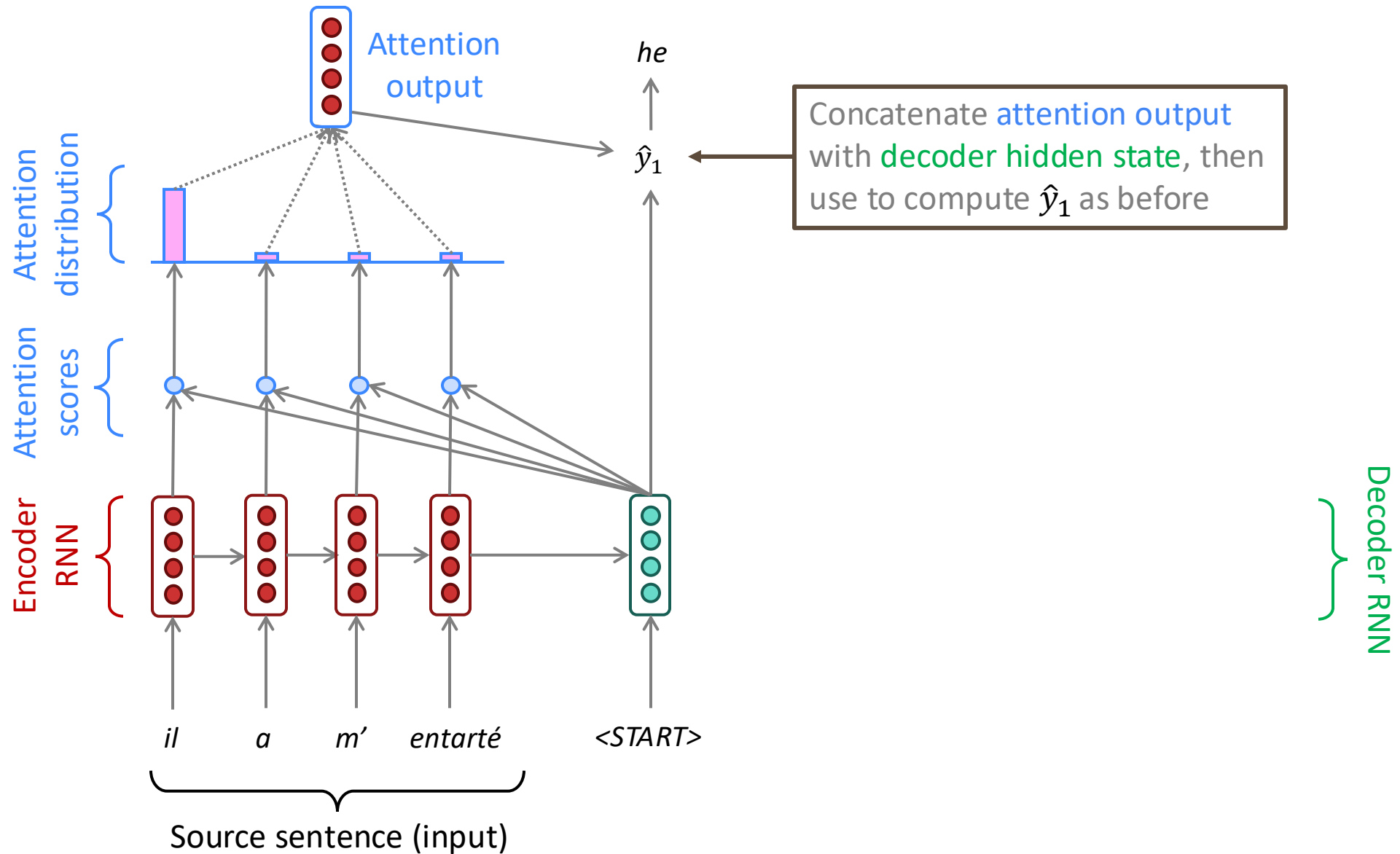
Sequence-to-sequence with attention



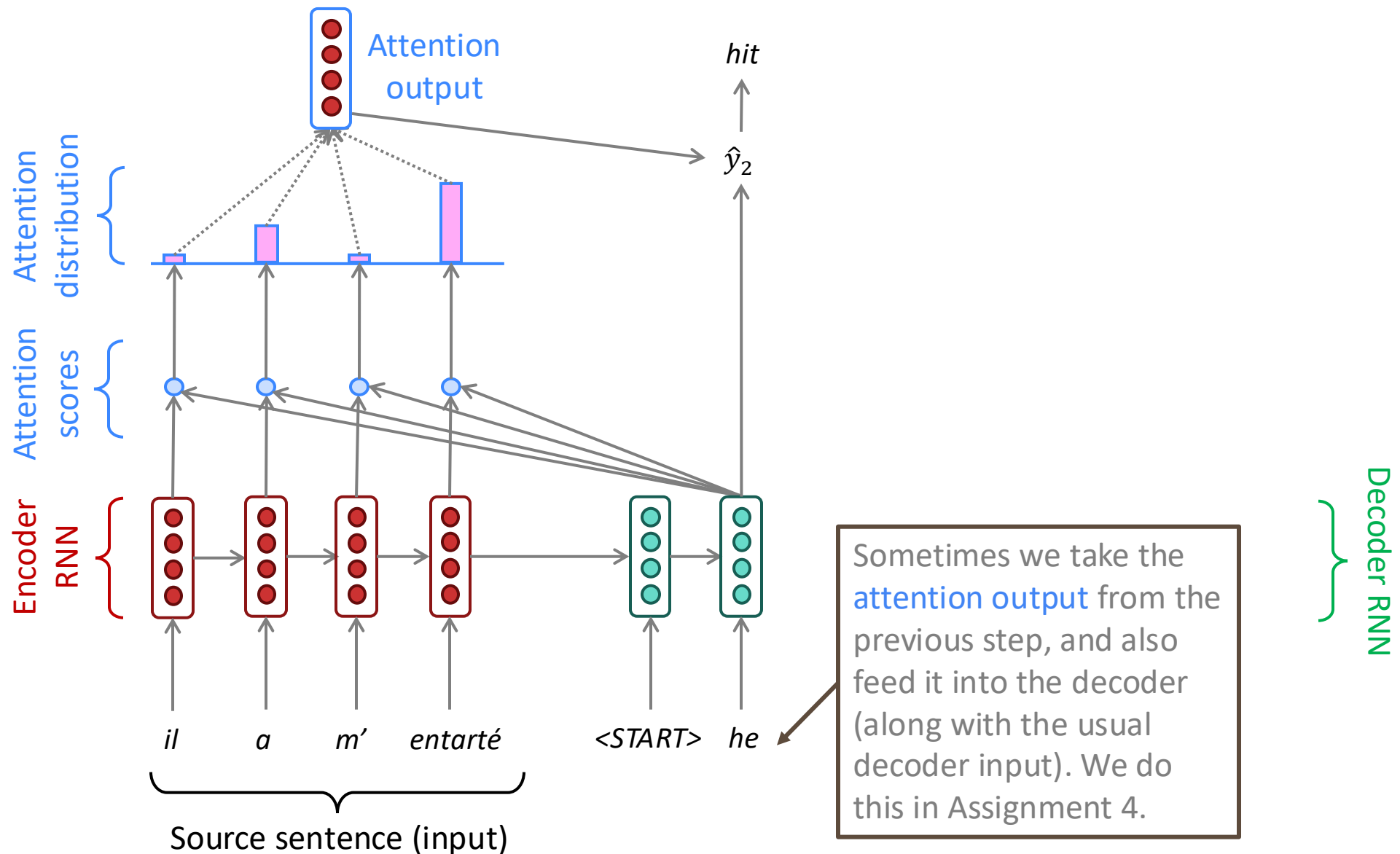
Sequence-to-sequence with attention



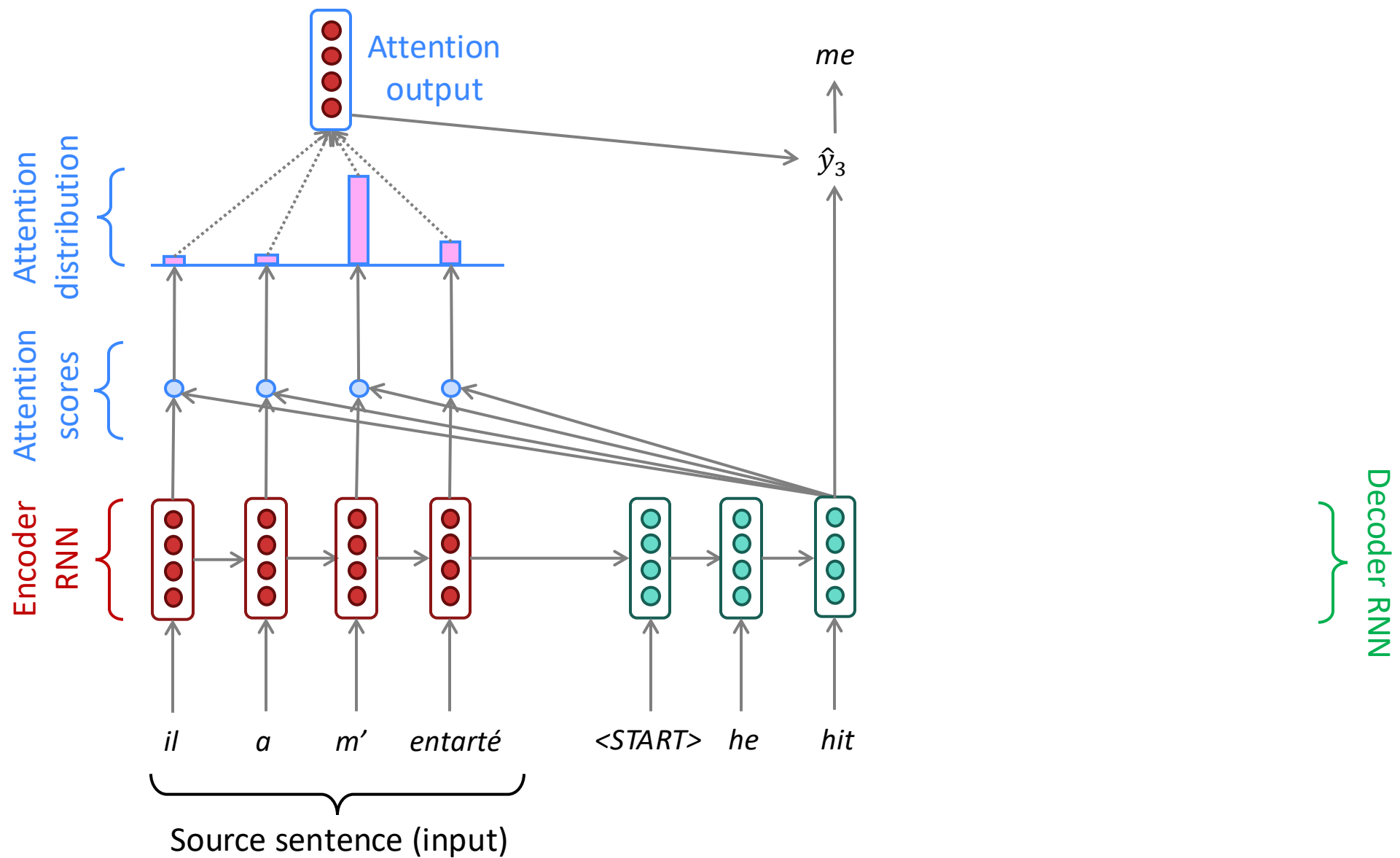
Sequence-to-sequence with attention



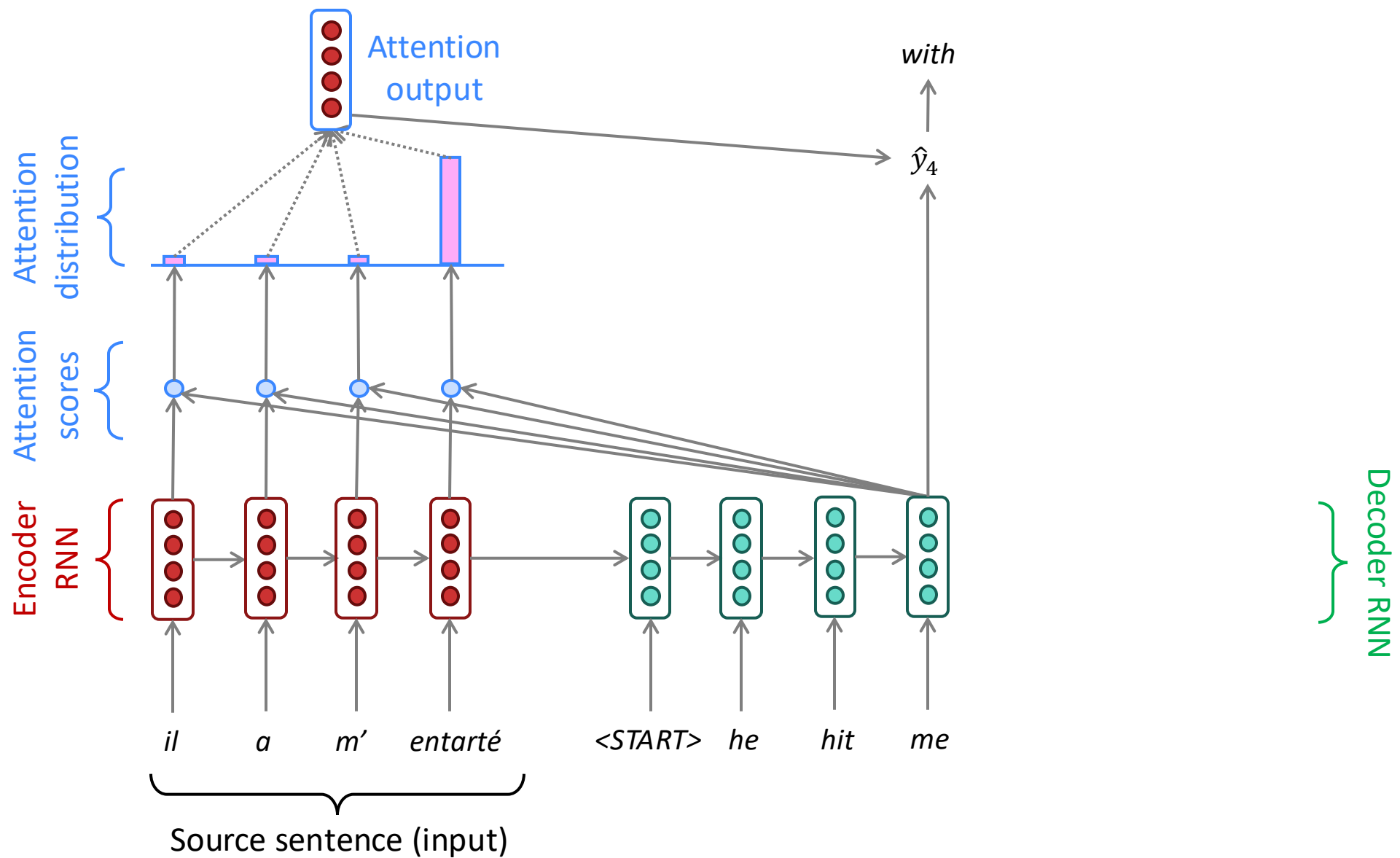
Sequence-to-sequence with attention



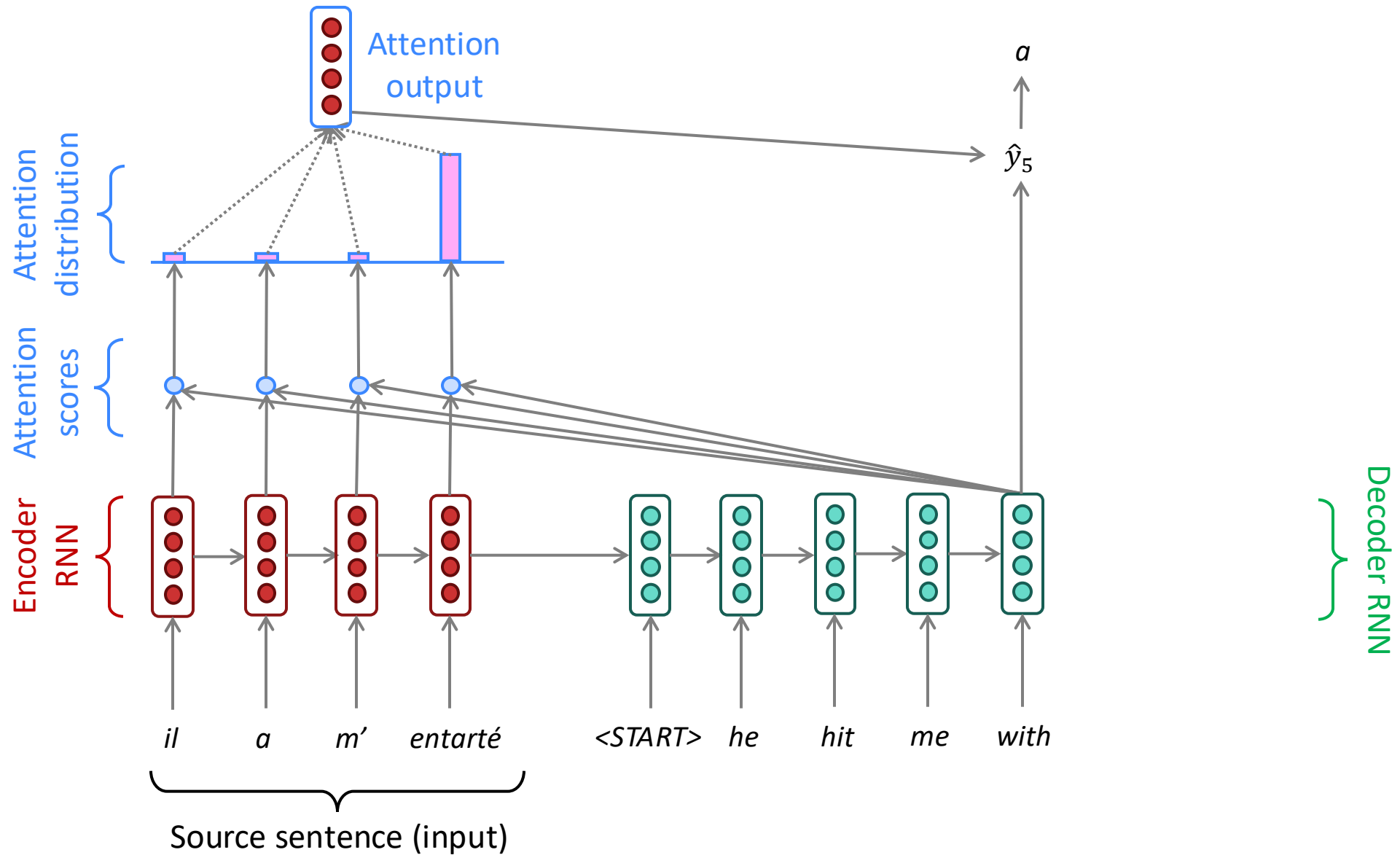
Sequence-to-sequence with attention



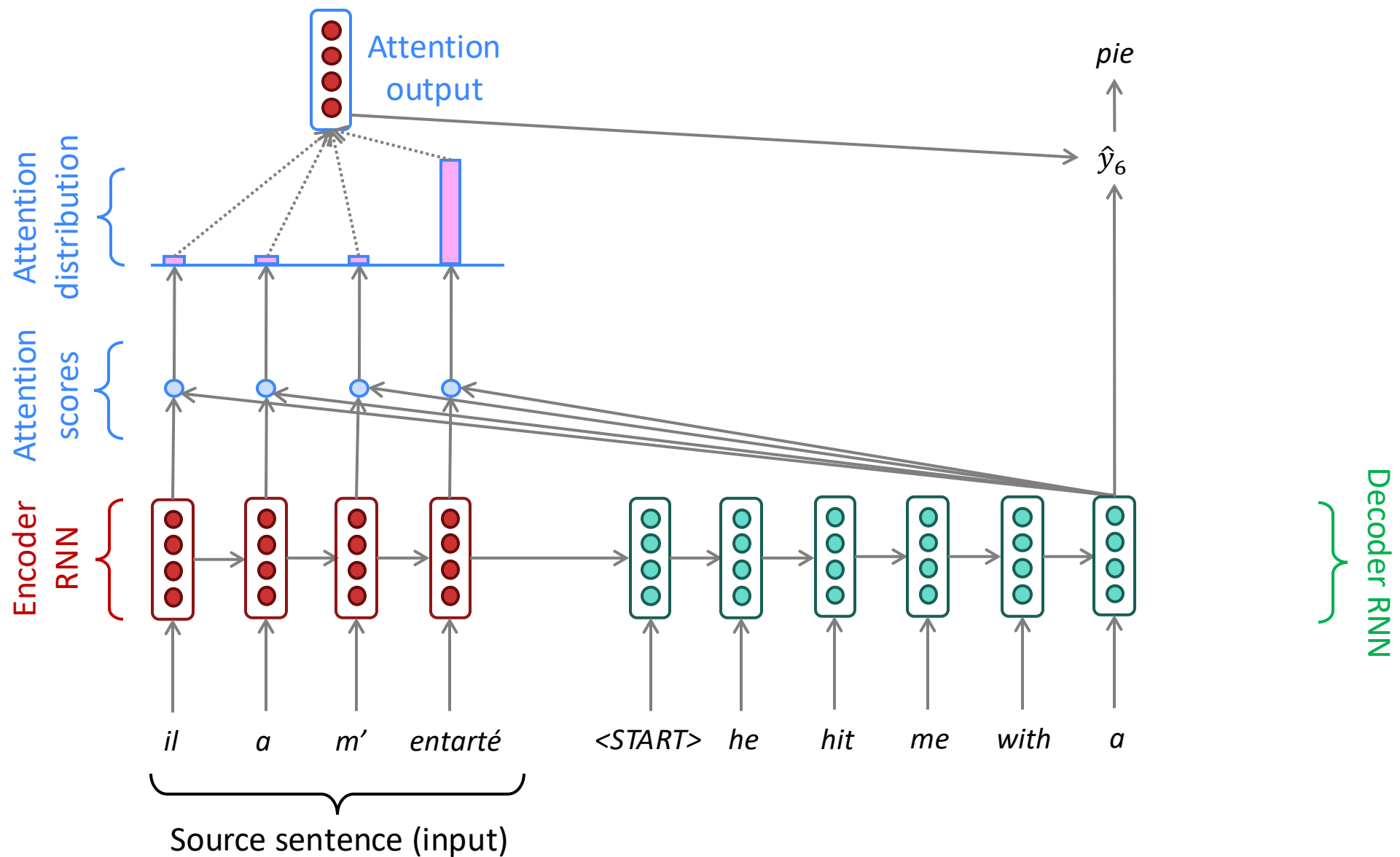
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention



Attention: in equations

- We have encoder hidden states $h_1, \dots, h_N \in \mathbb{R}^h$
- On timestep t , we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

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

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

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

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

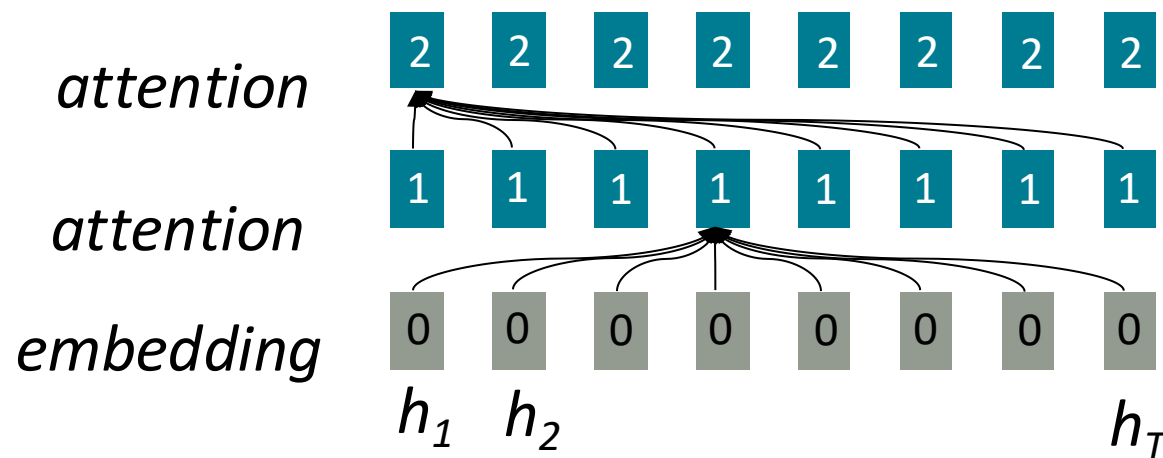
$$a_t = \sum_{i=1}^N \alpha_i^t h_i \in \mathbb{R}^h$$

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

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

Attention is parallelizable, and solves bottleneck issues.

- **Attention** treats each word's representation as a **query** to access and incorporate information from **a set of values**.
 - We saw attention from the **decoder** to the **encoder**; today we'll think about attention **within a single sentence**.
- Number of unparallelizable operations does not increase with sequence length.
- Maximum interaction distance: $O(1)$, since all words interact at every layer!



All words attend to all words in previous layer; most arrows here are omitted

Attention is great!



- Attention significantly **improves NMT performance**
 - It's very useful to allow decoder to focus on certain parts of the source
- Attention provides a **more “human-like” model** of the MT process
 - You can look back at the source sentence while translating, rather than needing to remember it all
- Attention **solves the bottleneck problem**
 - Attention allows decoder to look directly at source; bypass bottleneck
- Attention **helps with the vanishing gradient problem**
 - Provides shortcut to faraway states
- Attention provides **some interpretability**
 - By inspecting attention distribution, we see what the decoder was focusing on
 - We get (soft) **alignment for free!**
 - The network just learned alignment by itself
- (**One issue** – attention has *quadratic* cost with respect to sequence length)

	he	hit	me	with	a	pie
il						
a						
m'						
entarté						

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:

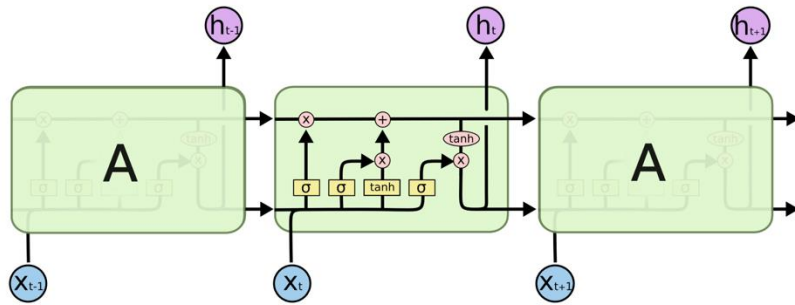
- The weighted sum is a *selective summary* of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).

Upshot:

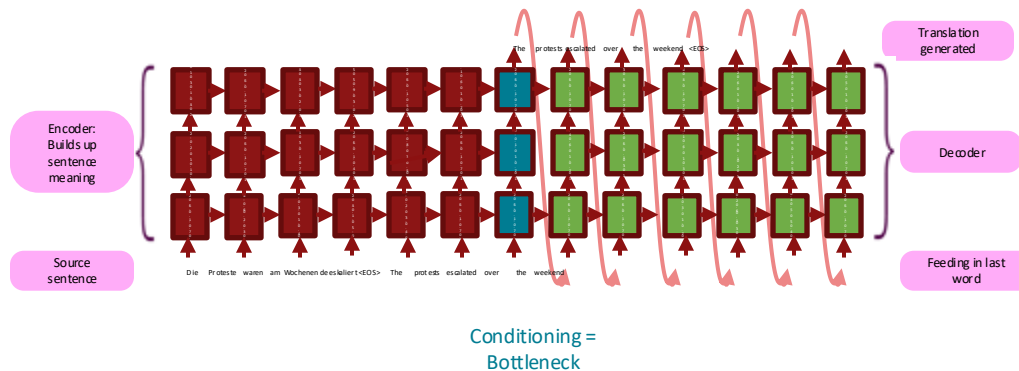
- Attention has become the powerful, flexible, general way pointer and memory manipulation in all deep learning models. A new idea from after 2010! From NMT!

In summary

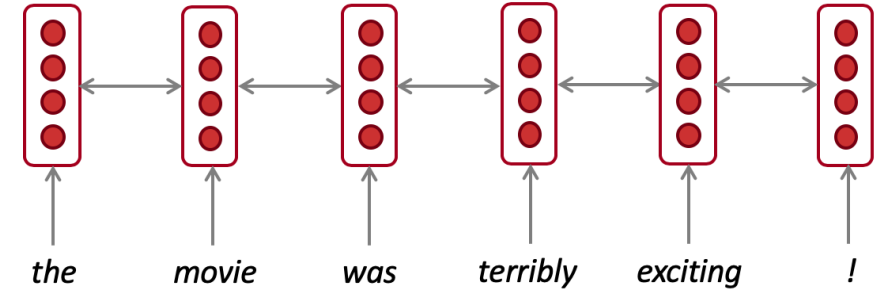
Lots of new information today! What are some of the **practical takeaways?**



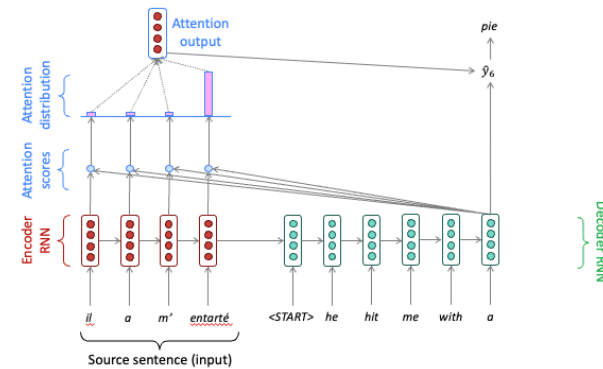
1. LSTMs are powerful



3. Encoder-Decoder Neural Machine Translation Systems work very well



2. Use bidirectionality when possible



4. Attention is a general, useful technique