DSA4213 Natural Language Processing for Data Science

Doudou Zhou (ddzhou@nus.edu.sg)

Lecture 3 LSTM, Machine Translation, and Attention

Lecture Plan

1. LSTM RNNs
2. Machine Translation
3. Attention

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*
- Became really well-known after Hinton brought it to Google in 2013
  - *Following Graves having been a postdoc with Hinton*

References:

- Hochreiter and Schmidhuber, 1997. Long short-term memory. Link
- Gers, Schmidhuber, and Cummins, 2000. Learning to Forget: Continual Prediction with LSTM. Link
- Graves, Fernandez, Gomez, and Schmidhuber, 2006. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural nets. Link

Long Short-Term Memory (LSTM)

- On step t, there is a **hidden state** $\mathbf{h}^{(t)}$ and a **cell state** $\mathbf{c}^{(t)}$
  - Both are vectors of 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*

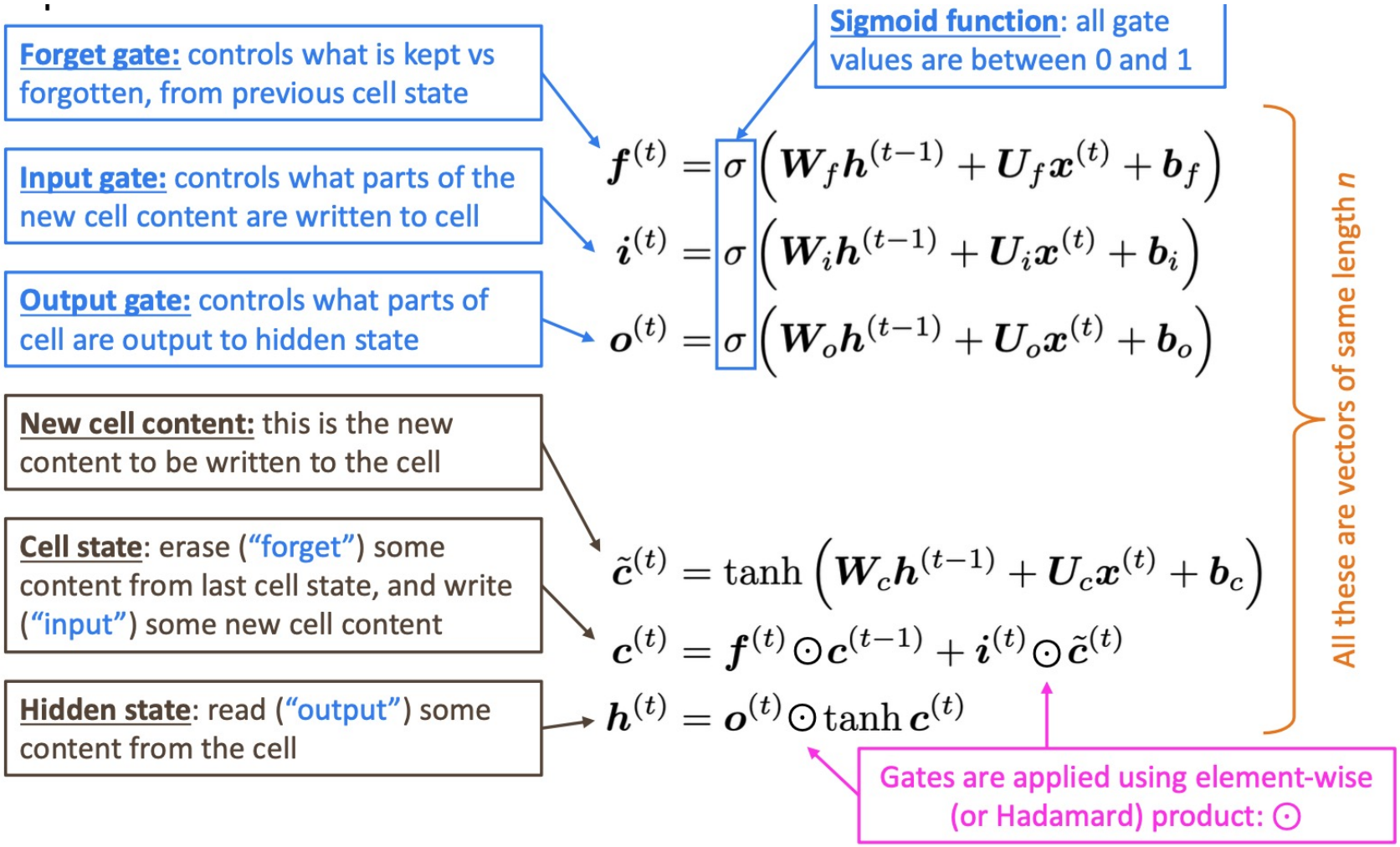
Long Short-Term Memory (LSTM)

- On step t, there is a **hidden state** $\mathbf{h}^{(t)}$ and a **cell state** $\mathbf{c}^{(t)}$
  - Both are vectors of 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 (gates are calculated things whose values are probabilities)**
  - 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 $\mathbf{x}^{(t)}$, and we will compute a sequence of hidden states $\mathbf{h}^{(t)}$ and cell states $\mathbf{c}^{(t)}$. On timestep t:



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** neural networks), 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)

Is vanishing/exploding gradient just an RNN problem?

- **No!** It can be a problem for all neural architectures (including **feed-forward** and **convolutional** neural networks), 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**

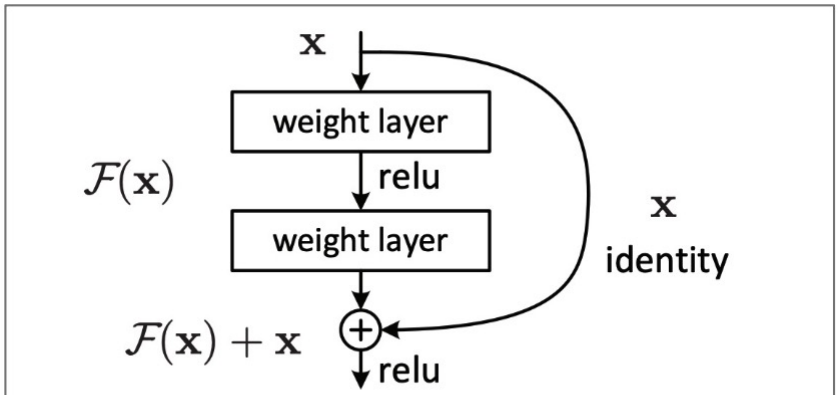


Figure 2. Residual learning: a building block.

"Deep Residual Learning for Image Recognition," He et al., 2015. Link

Is vanishing/exploding gradient just a RNN problem?

Other methods:

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

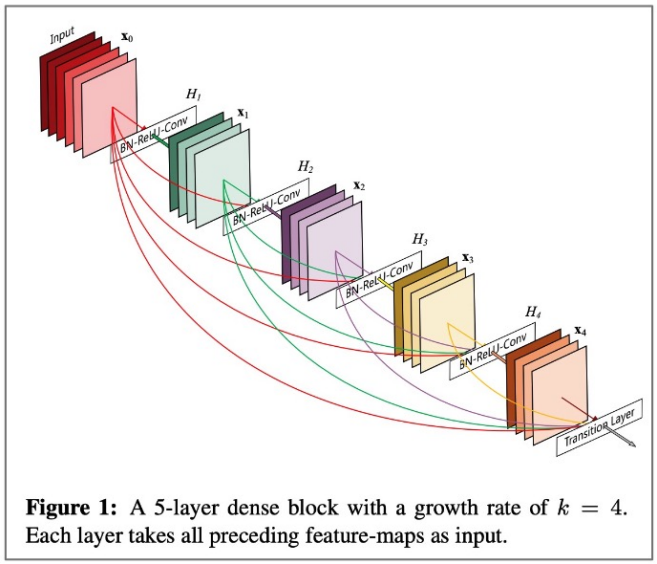
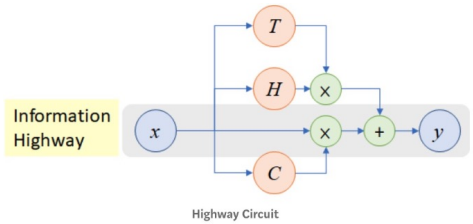


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

- **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

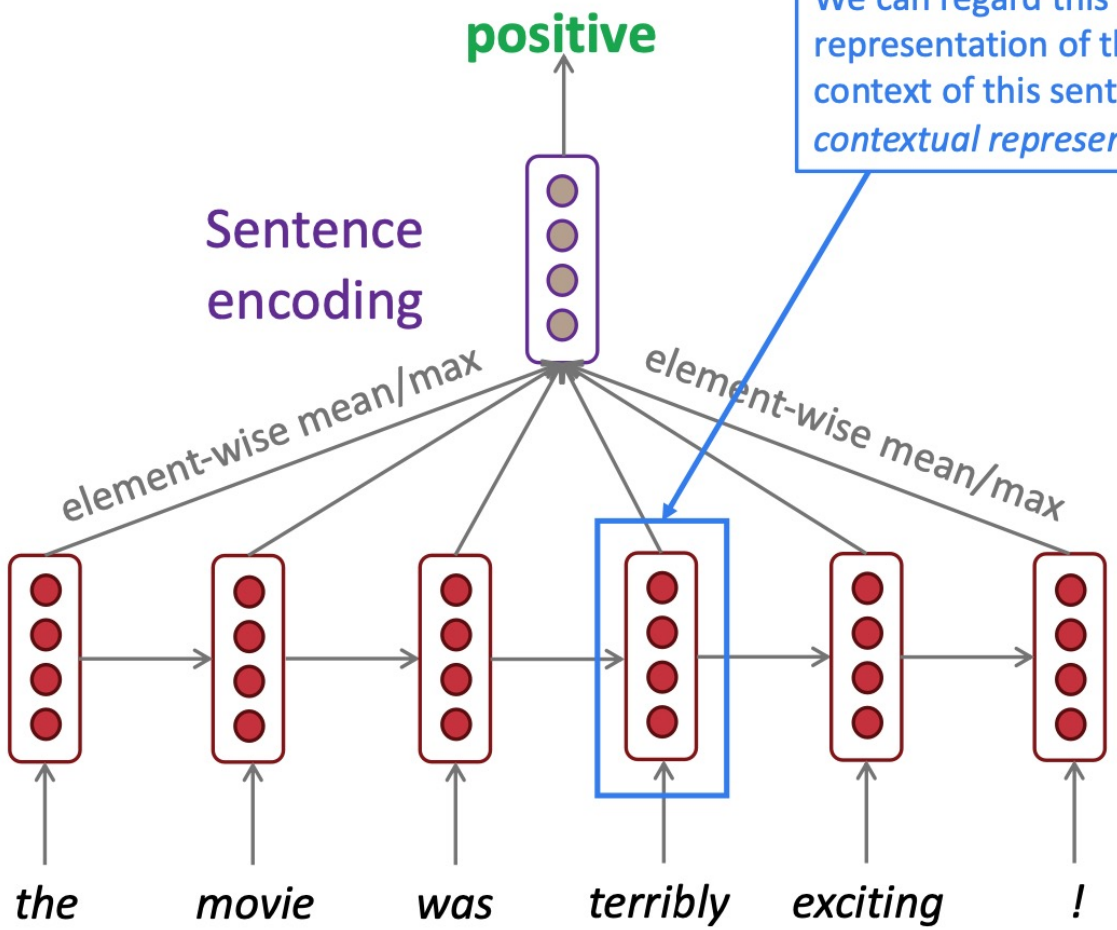


Conclusion: Though vanishing/exploding gradients are a general problem, **RNNs are particularly unstable** due to the repeated multiplication by the **same weight matrix** [Bengio et al., 1994]. *References:*

- *"Densely Connected Convolutional Networks," Huang et al., 2017.* Link
- *"Highway Networks," Srivastava et al., 2015.* Link

Bidirectional and Multi-layer RNNs: motivation

Task: Sentiment Classification



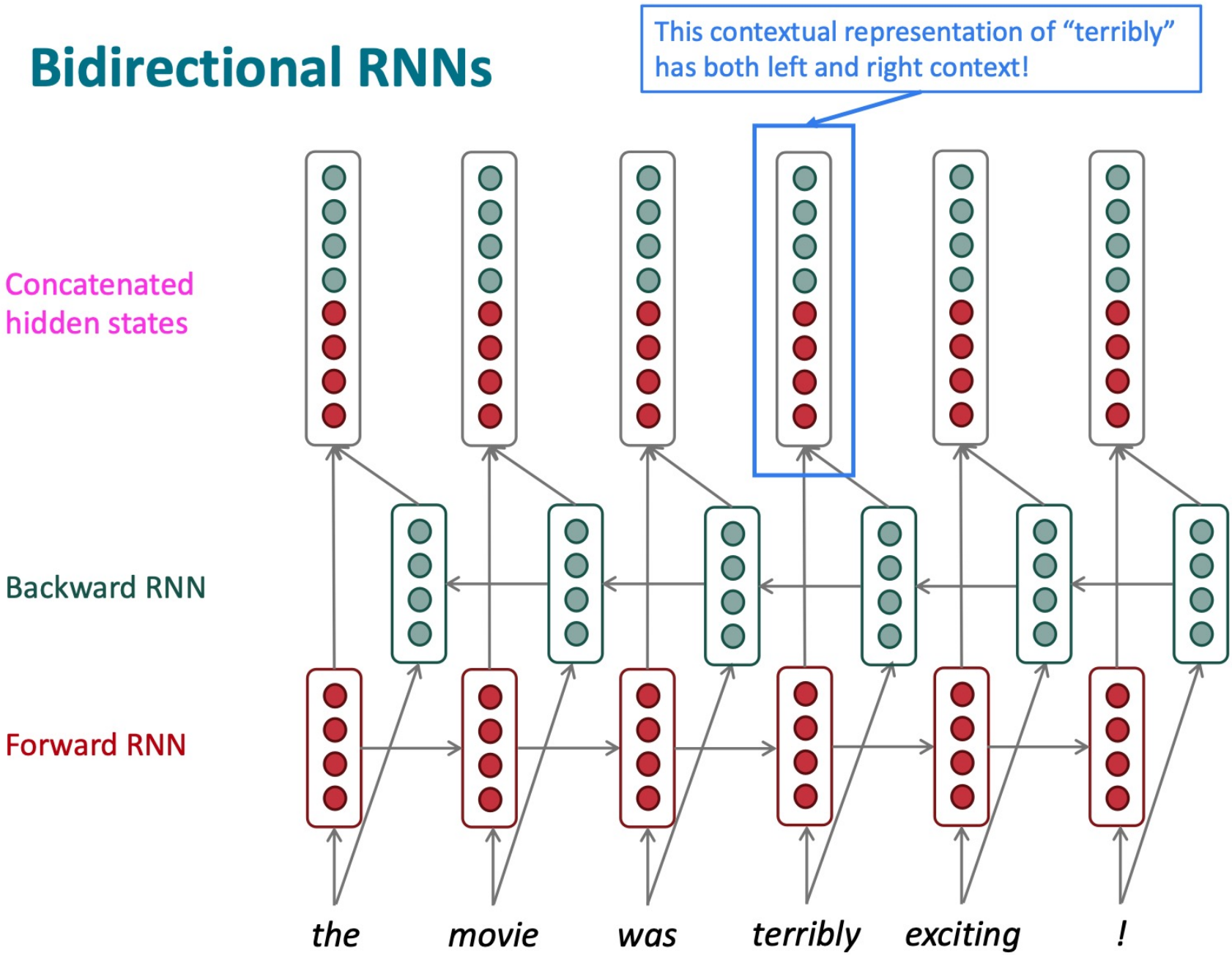
We can regard this hidden state as a representation of the word *"terribly"* in the context of this sentence. We call this a *contextual representation.*

These contextual representations only contain information about the *left* context (e.g. *"the movie was"*).

What about *right* context?

In this example, *"exciting"* is in the right context and this modifies the meaning of *"terribly"* (from negative to positive)

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 $\overrightarrow{\boldsymbol{h}}^{(t)} = \boxed{\mathrm{RNN}_{\mathrm{FW}}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

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

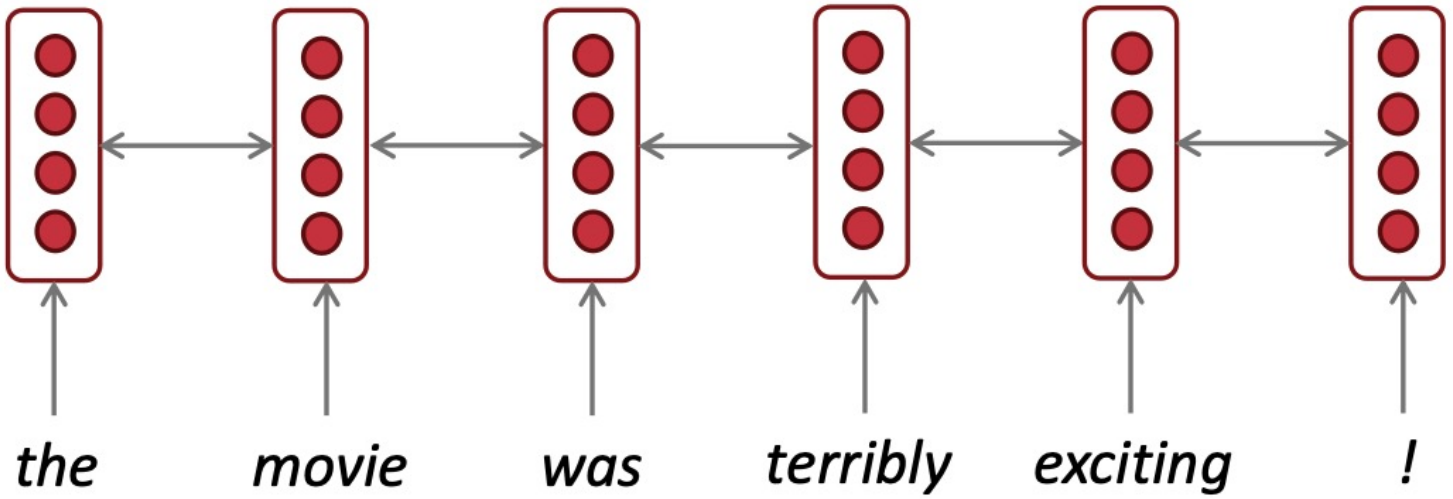
Generally, these two RNNs have separate weights

Concatenated hidden states $\boxed{\boldsymbol{h}^{(t)}} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

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

35

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.

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).

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).

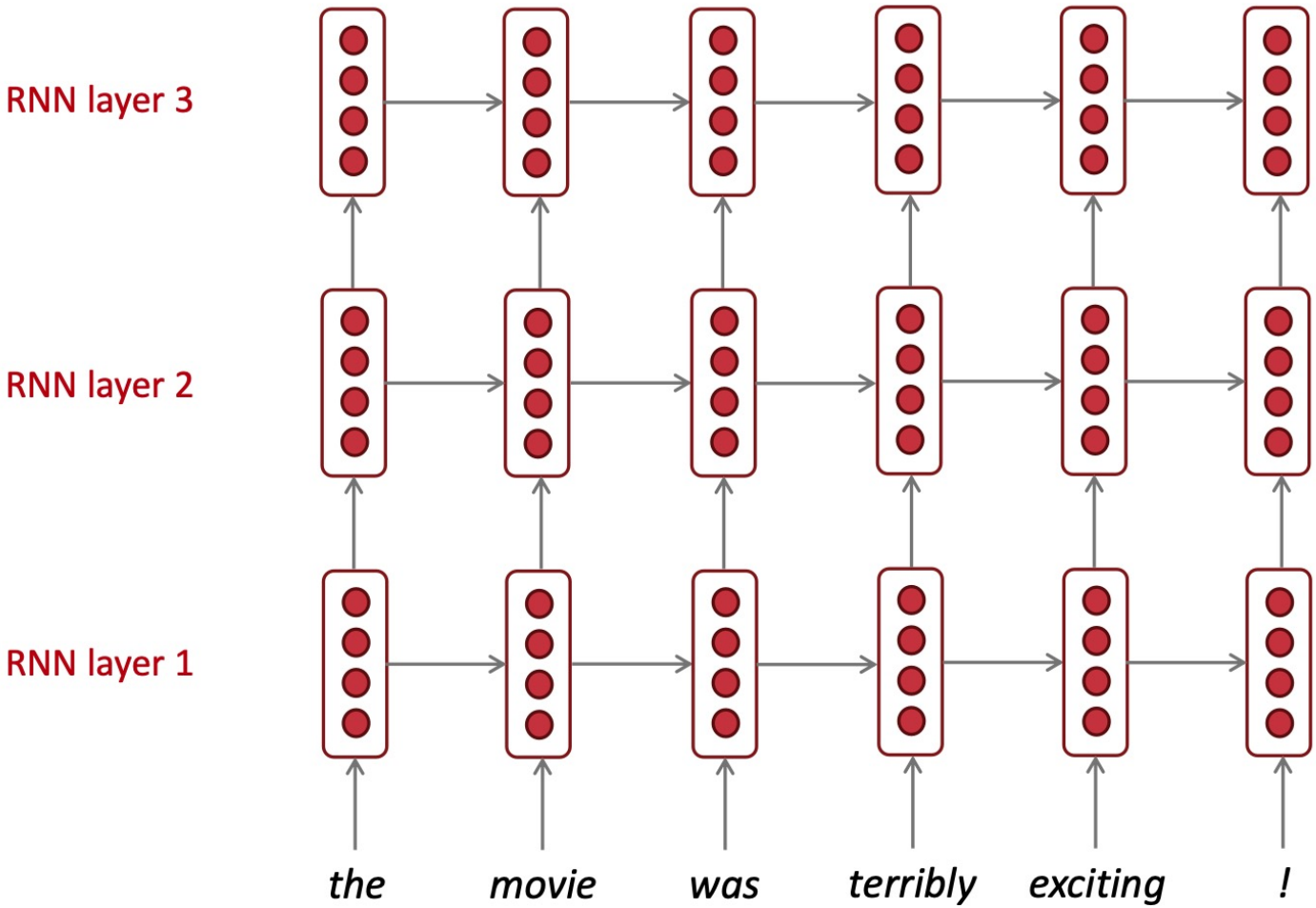
- For example, **BERT** (**Bidirectional** Encoder Representations from Transformers) is a powerful pretrained contextual representation system **built on bidirectionality**.
  - You will learn more about **transformers**, including BERT, in a couple of weeks!

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 having one layer of high-dimensional encodings!
  - The **lower RNNs** should compute lower-level features and the **higher RNNs** should compute higher-level features.

Multi-layer RNNs in practice

- Multi-layer or stacked RNNs allow a network to compute **more complex representations** – they work better than just having 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).

Multi-layer RNNs in practice

- Multi-layer or stacked RNNs allow a network to compute **more complex representations** – they work better than just having 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**).

Multi-layer RNNs in practice

- Multi-layer or stacked RNNs allow a network to compute **more complex representations** – they work better than just having 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.

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.

- Now (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.

Source:

- *"Findings of the 2016 Conference on Machine Translation (WMT16)," Bojar et al., 2016.* Link
- *"Findings of the 2018 Conference on Machine Translation (WMT18)," Bojar et al., 2018.* Link
- *"Findings of the 2019 Conference on Machine Translation (WMT19)," Barrault et al., 2019.* Link

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.

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:

$$\arg\max_{y} P(y \mid x)$$

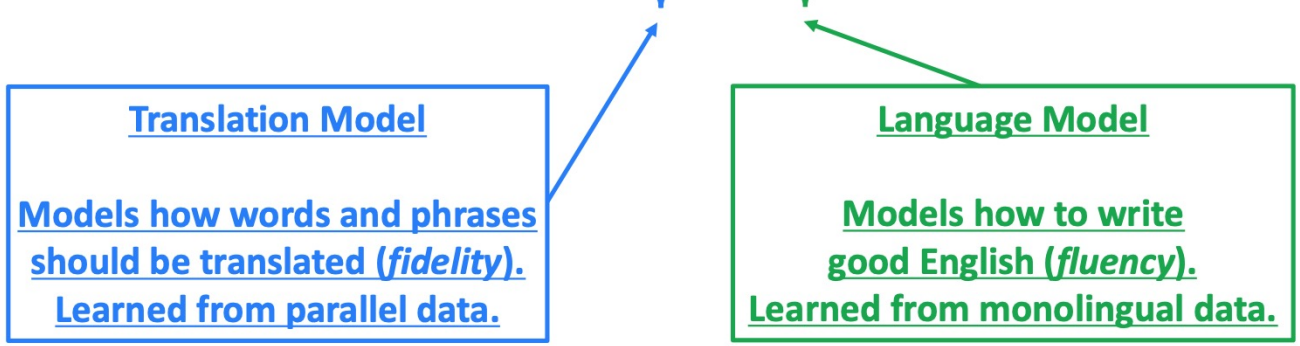
1990s-2010s: Statistical Machine Translation

- **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:

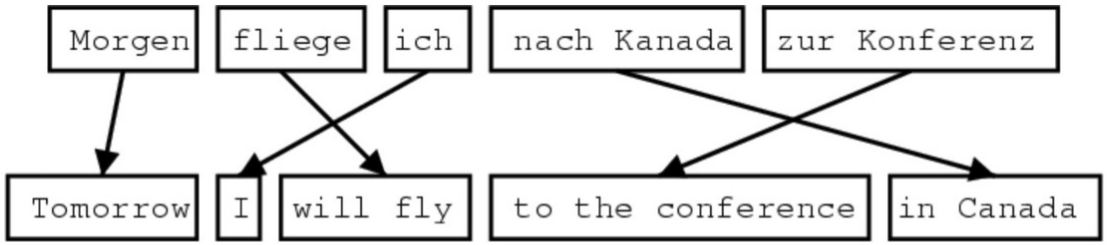
$$\arg\max_y P(y \mid x)$$

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

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



What happens in translation isn't trivial to model!



1519年600名西班牙人在墨西哥登陆，去征服几百万人口的阿兹特克帝国，初次交锋他们损兵三分之二。

In 1519, six hundred Spaniards landed in Mexico to conquer the Aztec Empire with a population of a few million. They lost two thirds of their soldiers in the first clash.

translate.google.com (2009): 1519 600 Spaniards landed in Mexico, millions of people to conquer the Aztec empire, the first two-thirds of soldiers against their loss.

translate.google.com (2013): 1519 600 Spaniards landed in Mexico to conquer the Aztec empire, hundreds of millions of people, the initial confrontation loss of soldiers two-thirds.

translate.google.com (2015): 1519 600 Spaniards landed in Mexico, millions of people to conquer the Aztec empire, the first two-thirds of the loss of soldiers they clash.

1990s–2010s: Statistical Machine Translation

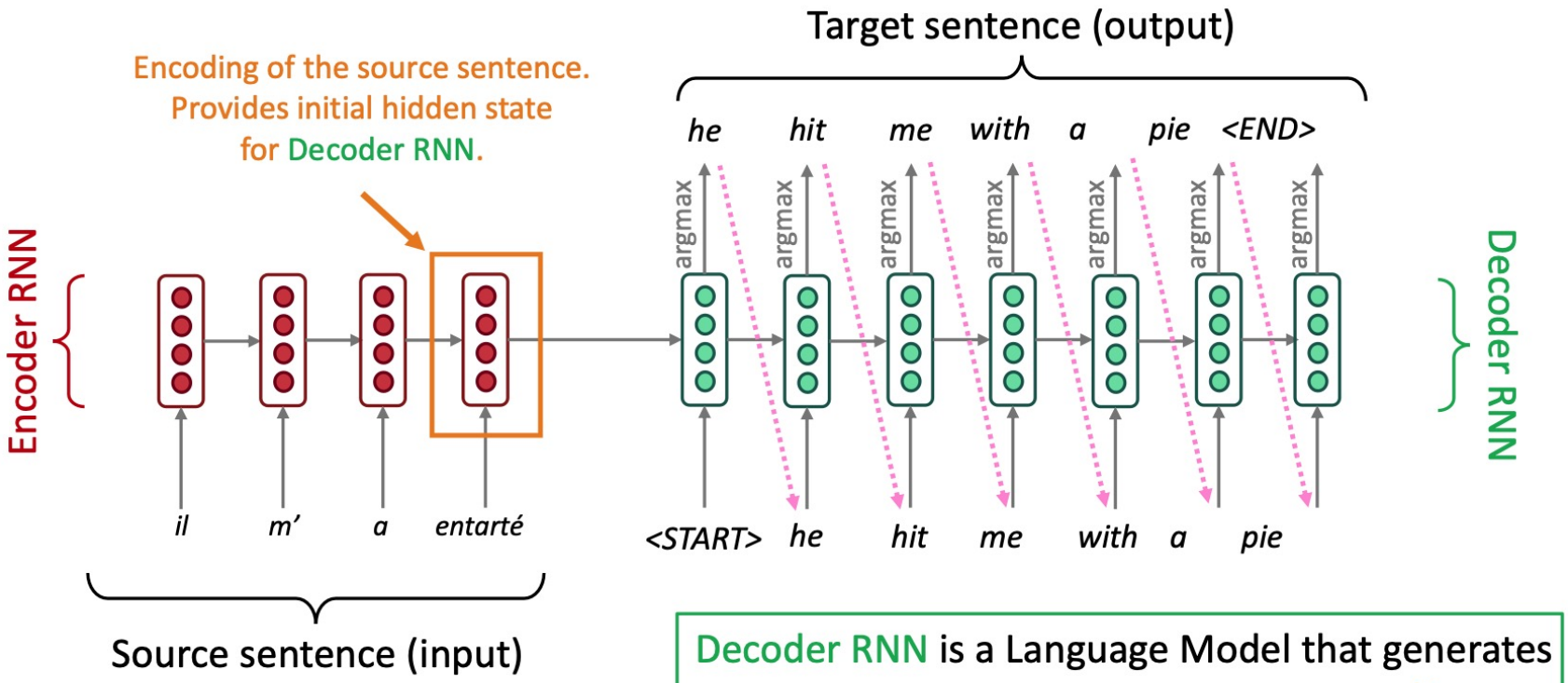
- SMT was a huge research field.
- The best systems were extremely complex.
  - Hundreds of important details.
- Systems had many separately-designed subcomponents.
  - Lots of **feature engineering**:
    - Need to design features to capture particular language phenomena.
  - Required compiling and 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)

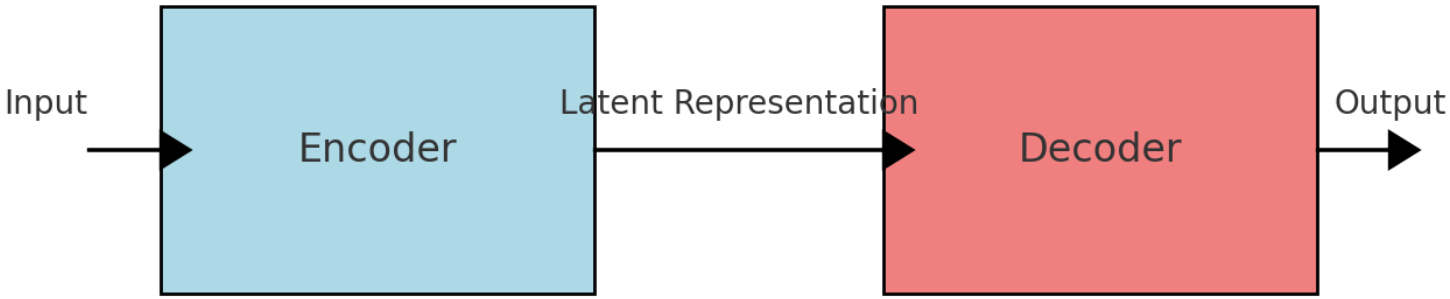
What is Neural Machine Translation?

- **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**.

The sequence-to-sequence model



Encoder and Decoder



- The Encoder (left) compresses the input into a latent representation.
- The Decoder (right) reconstructs or generates an output from this latent representation.
- The latent representation (middle) acts as an intermediate summary of the input, enabling transformations such as translation, summarization, or data compression.

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 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* $\rightarrow$ *short text*)
  - **Dialogue** (*previous utterances* $\rightarrow$ *next utterance*)
  - **Parsing** (*input text* $\rightarrow$ *output parse as sequence*)
  - **Code generation** (*natural language* $\rightarrow$ *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.

Neural Machine Translation (NMT)

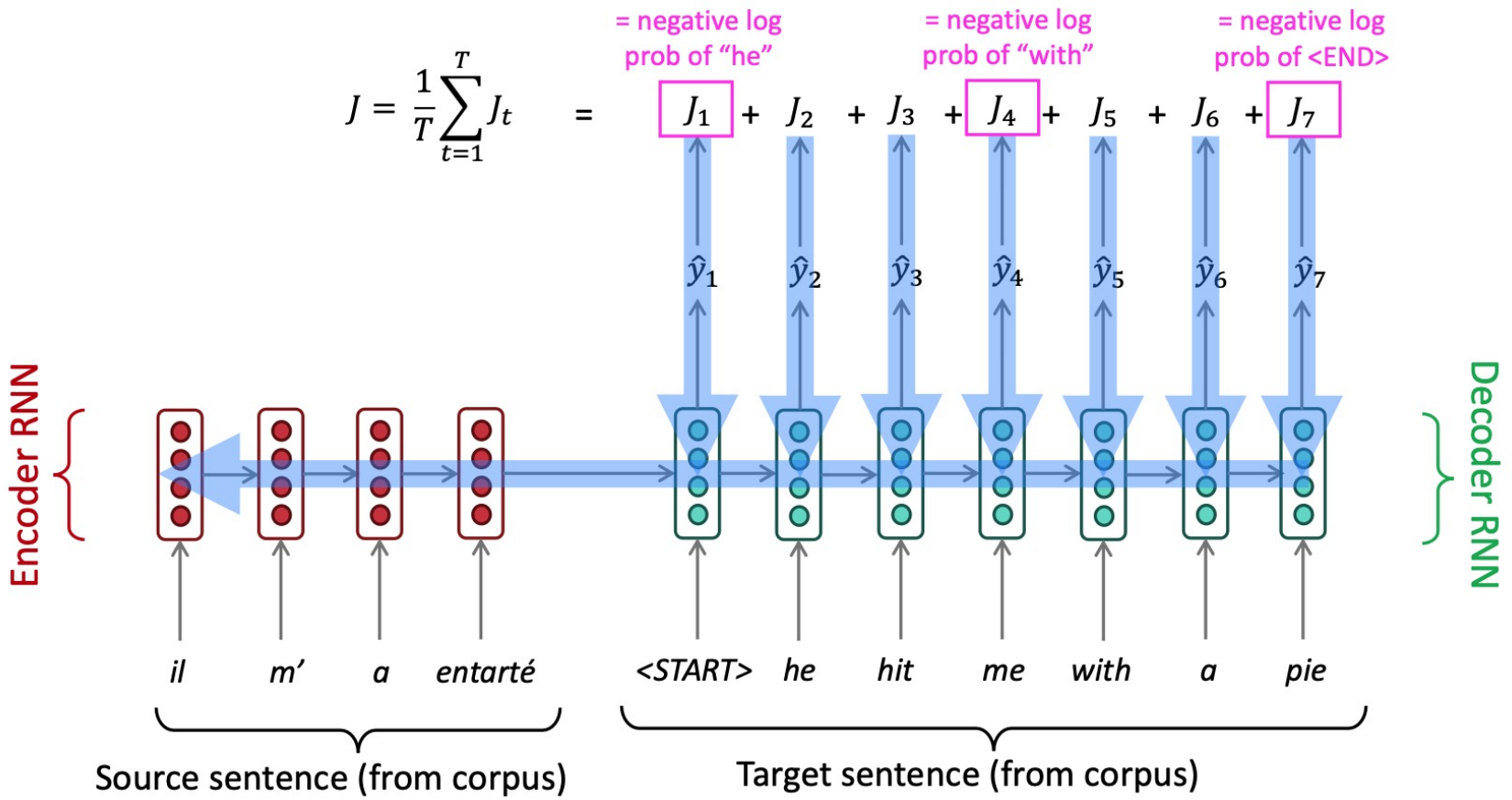
- The **sequence-to-sequence model** is an example of a **Conditional Language Model**:
  - **Language Model** because the decoder is predicting the next word of the target sentence y.
  - **Conditional** because its predictions are *also* conditioned on the source sentence x.
- **NMT directly calculates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

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

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

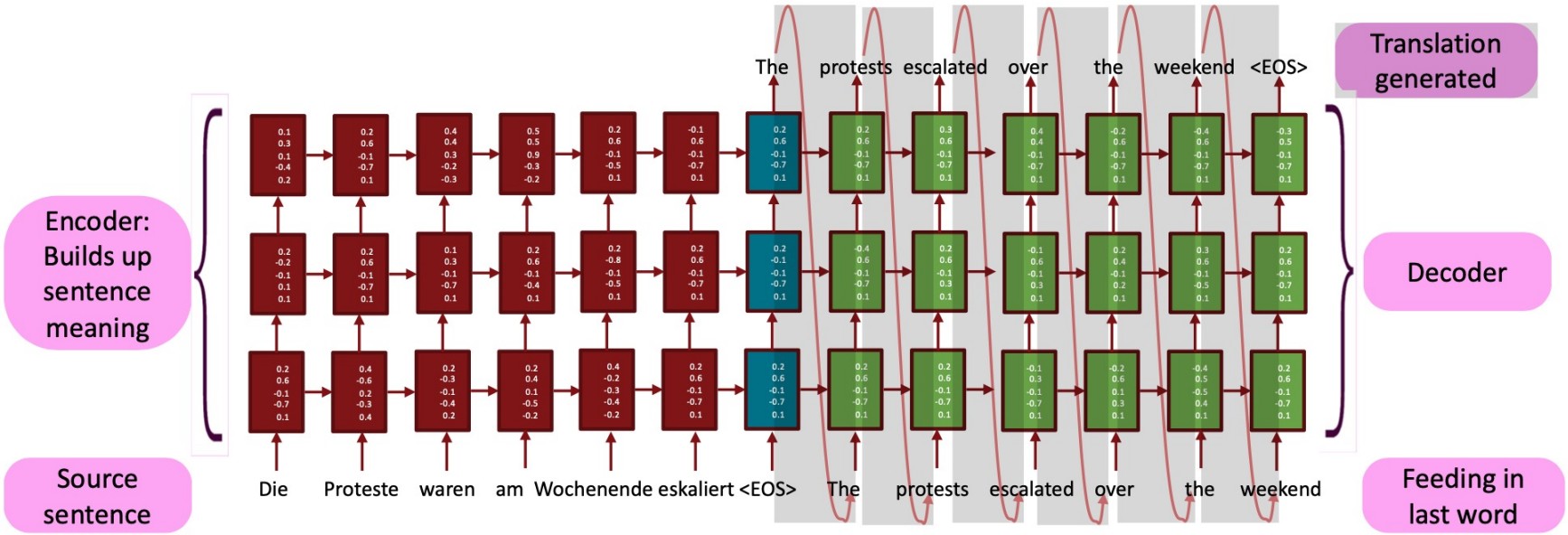


Seq2seq is optimized as a **single system.** Backpropagation operates *"end-to-end"*.

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



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 had.
  - NYTimes article: The Great AI Awakening

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

How do we evaluate Machine Translation?

Commonest way: BLEU (Bilingual Evaluation Understudy)

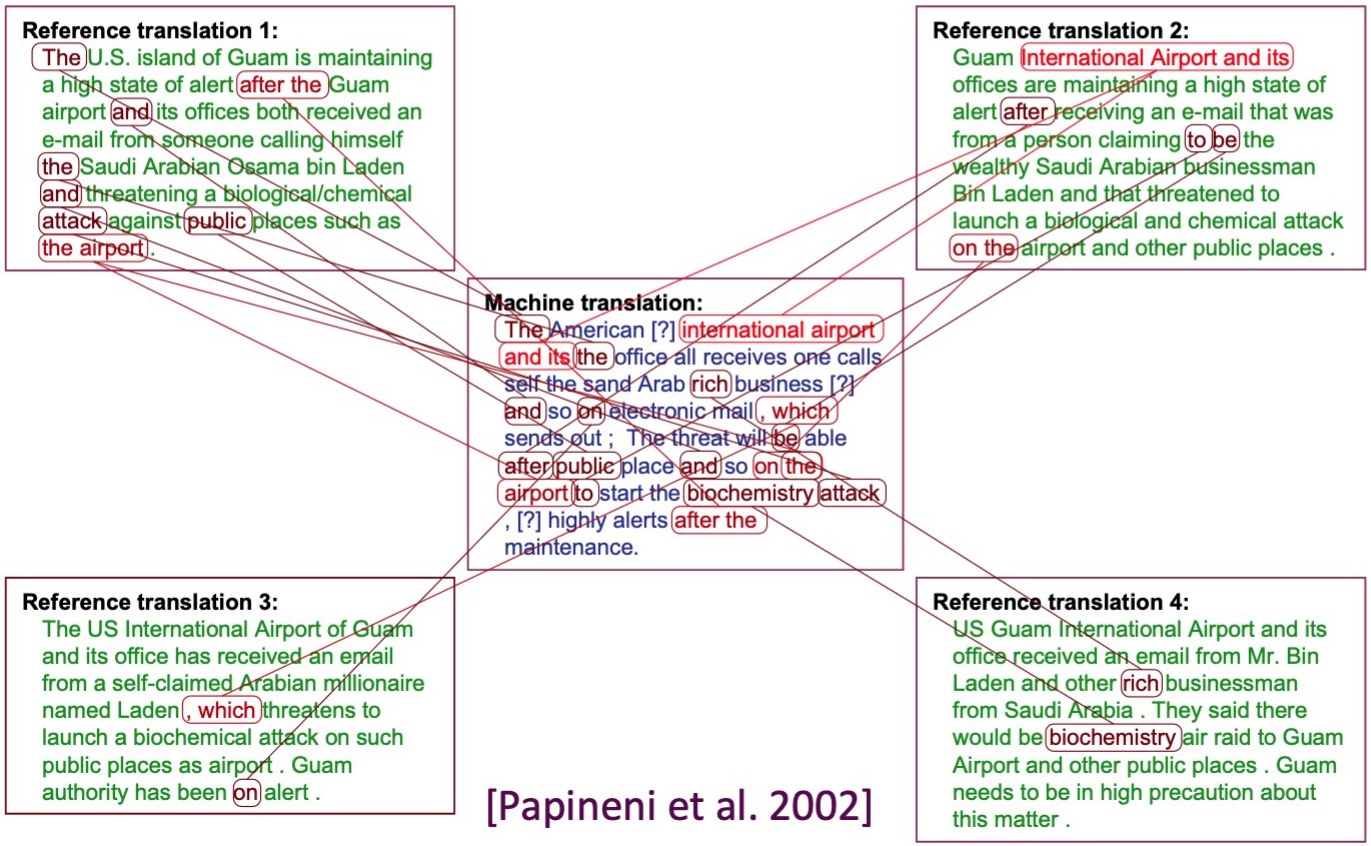
- BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:
  - Geometric mean of n-gram precision (usually for 1, 2, 3 and 4-grams)
  - Plus a penalty for too-short system translations
- BLEU is useful but imperfect
  - There are many valid ways to translate a sentence
  - Therefore, 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.* Link

BLEU score against 4 reference translations

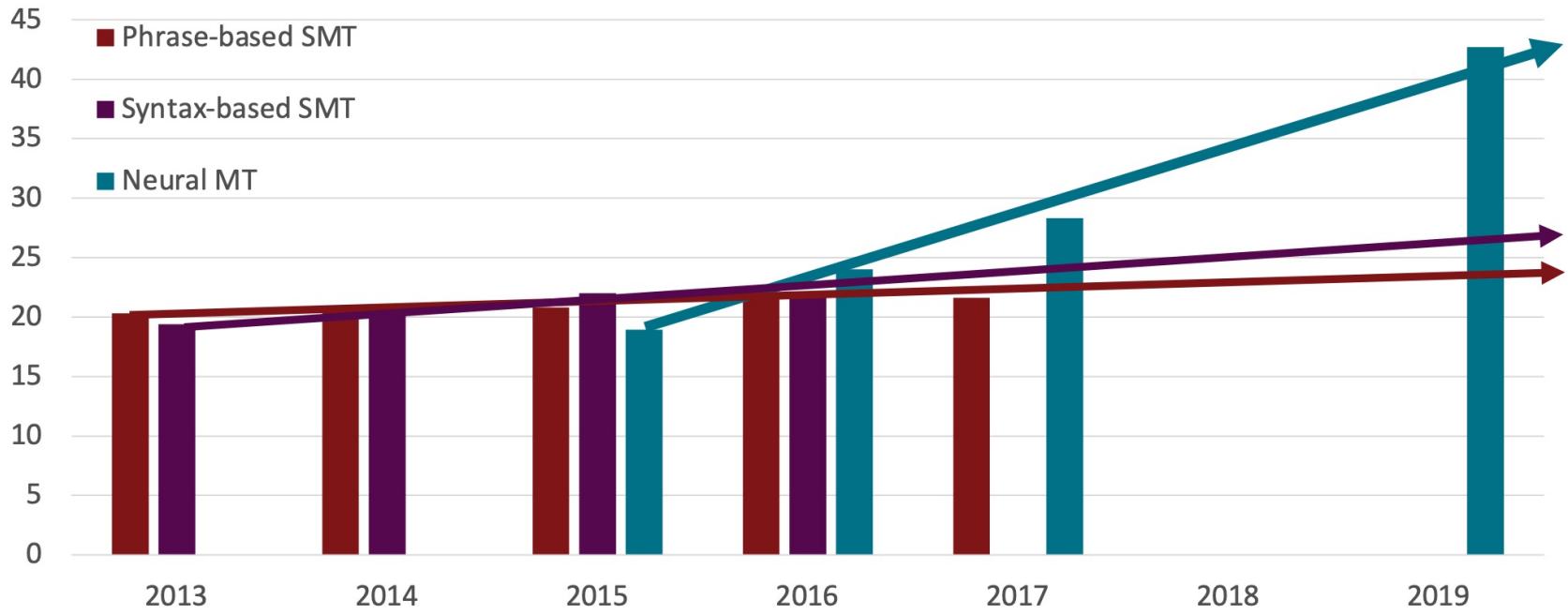
But commonly now there is only one references and so the results are more "in expectation"



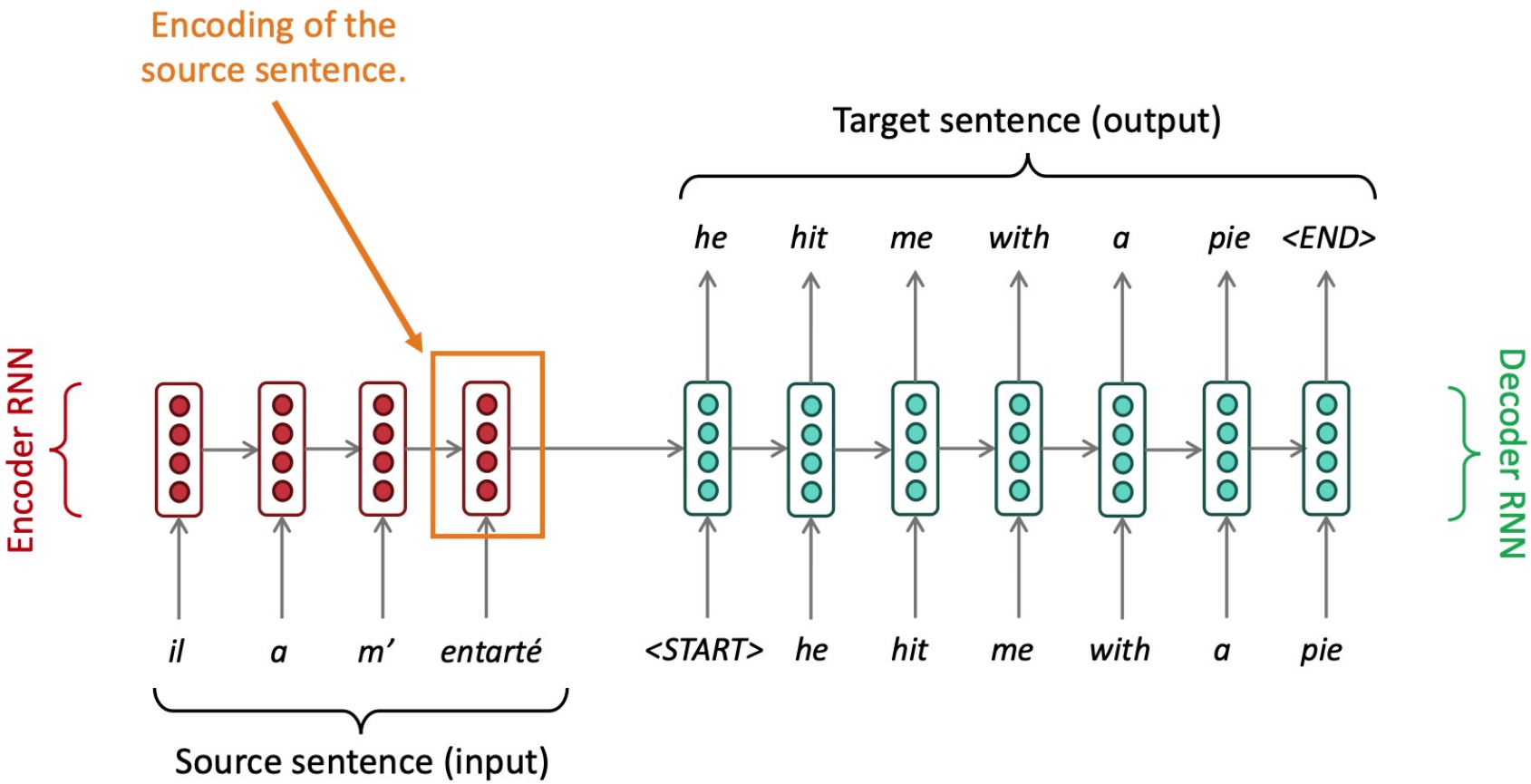
[Papineni et al. 2002]

MT progress over time

[Edinburgh En-De WMT newstest2013 Cased BLEU; NMT 2015 from U. Montréal; NMT 2019 FAIR on newstest2019]

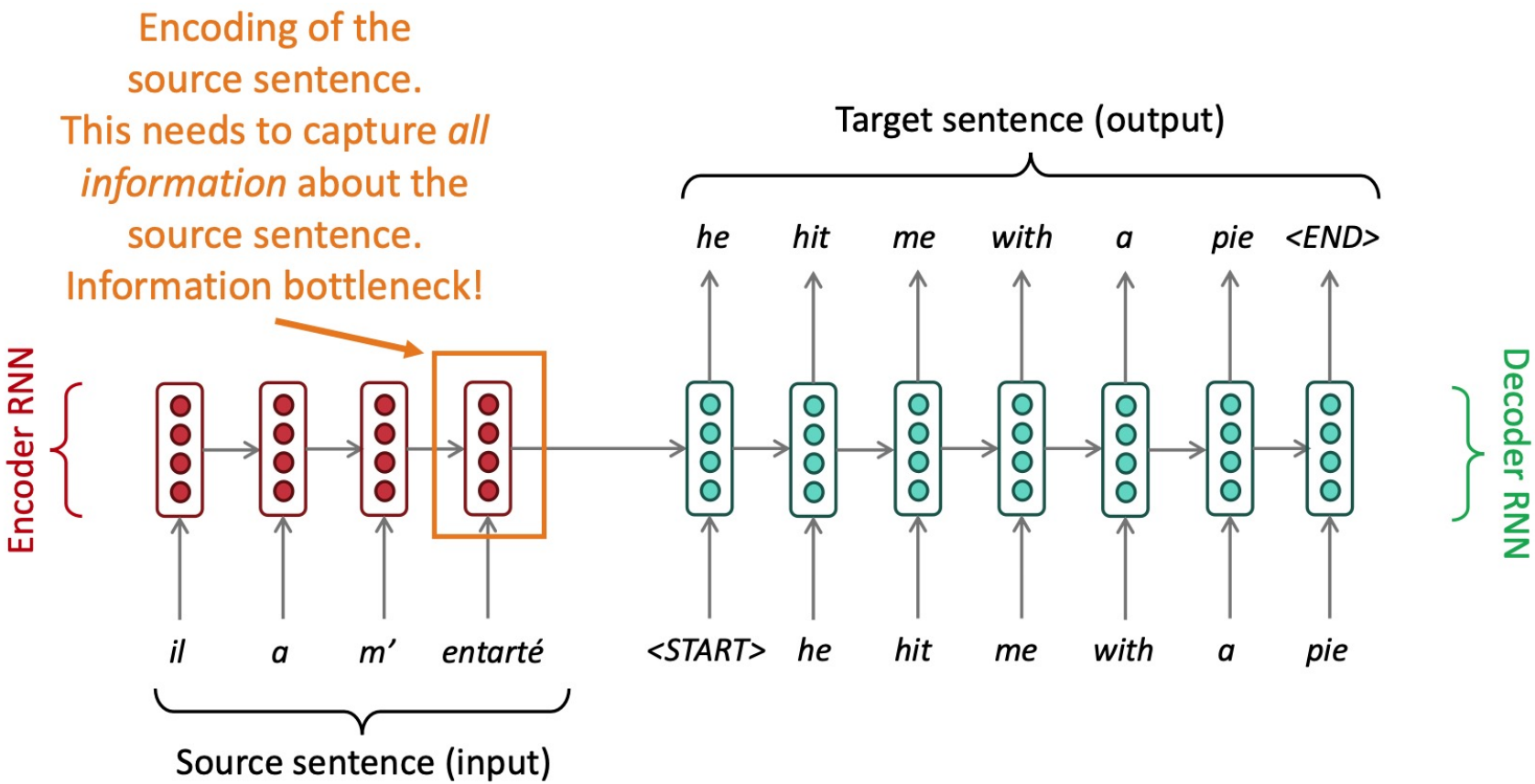


Why attention? Sequence-to-sequence: the bottleneck problem



Problems with this architecture?

Why attention? Sequence-to-sequence: the bottleneck problem

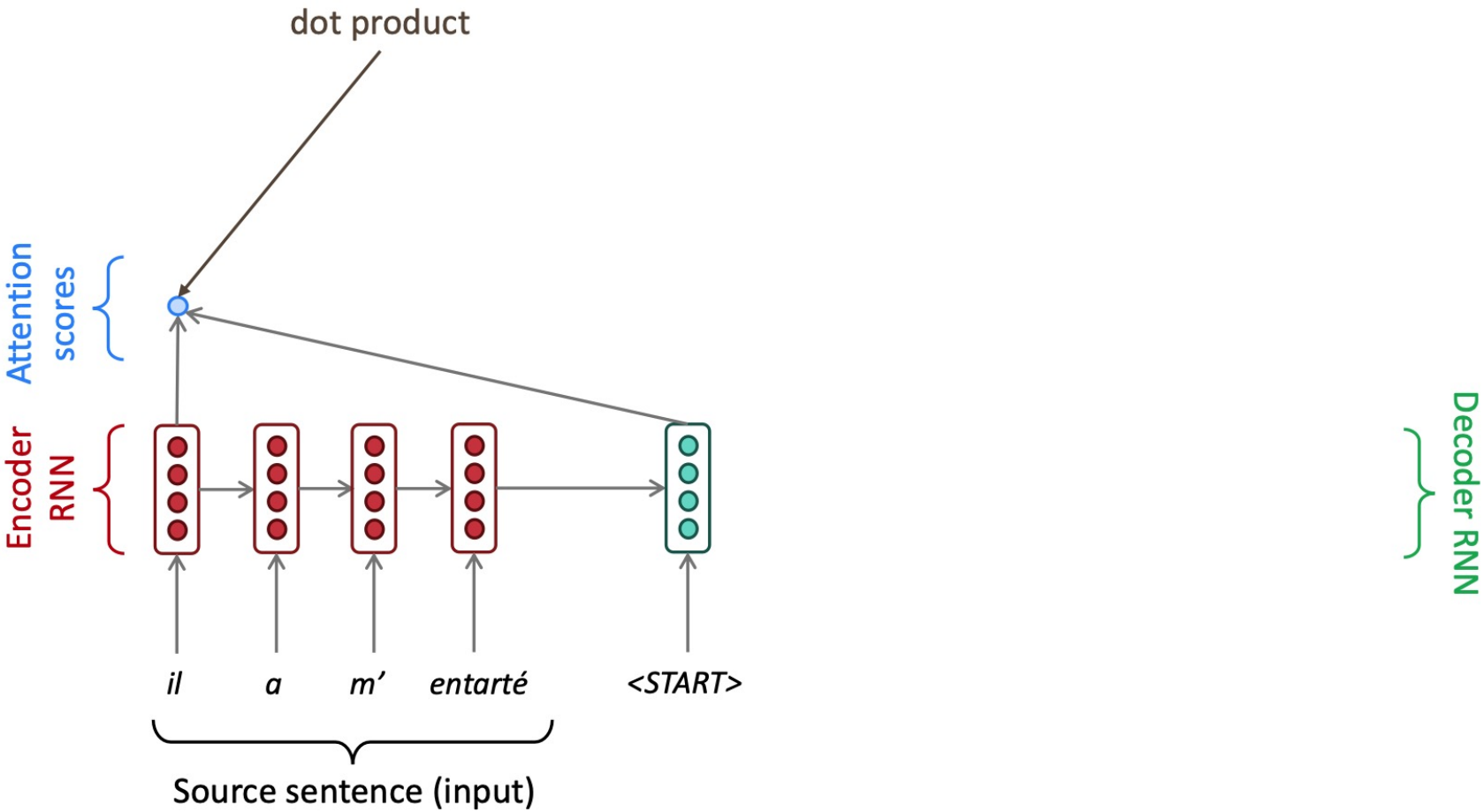


Attention

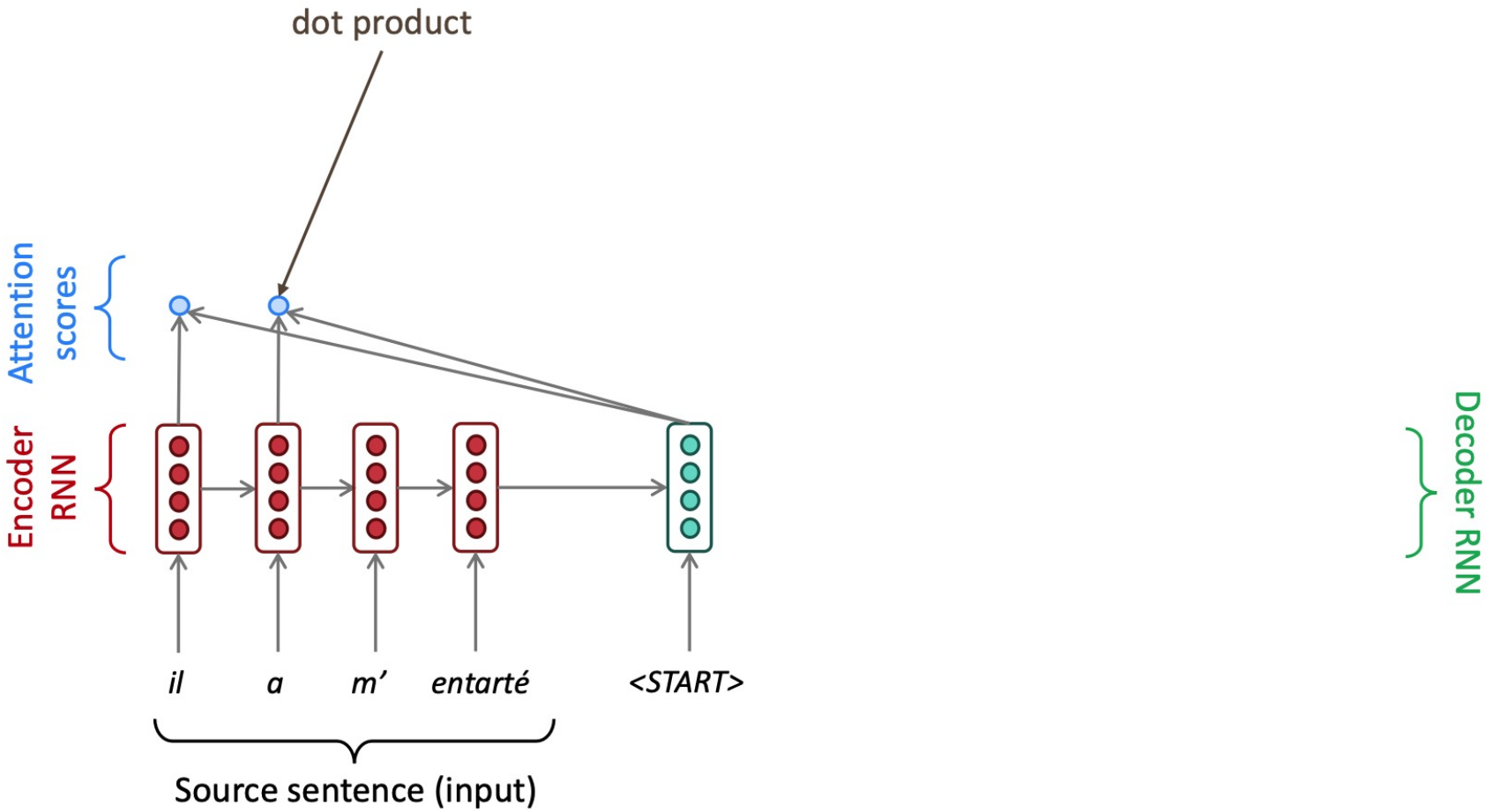
- **Attention** provides a solution to the bottleneck problem.
- **Core idea:** On each step of the decoder, **use direct connection to the encoder to focus on a particular part of the source sequence**.
- First, we will show via diagram (no equations), then we will show with equations.

Sequence-to-sequence with attention

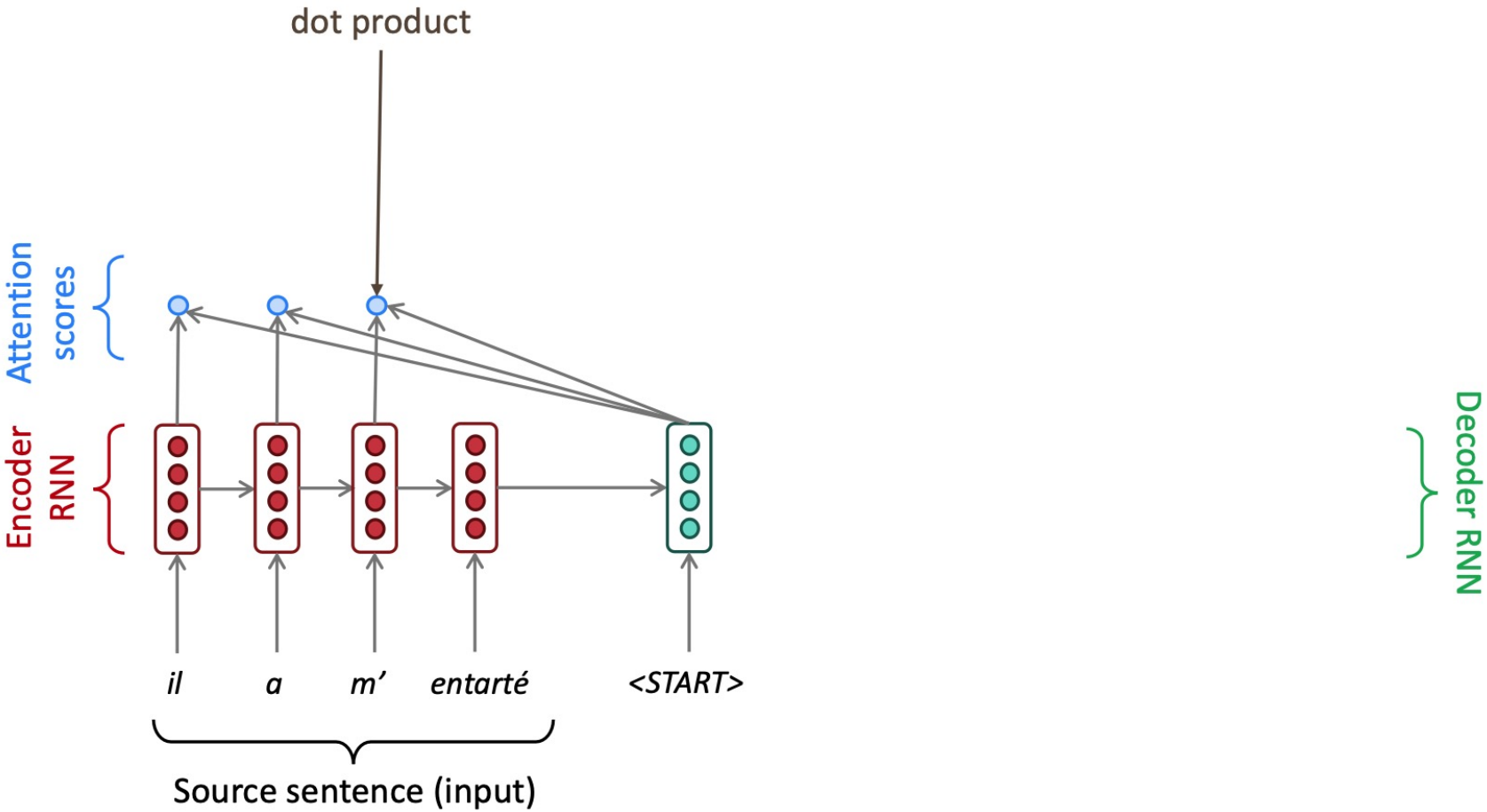
Core idea: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence



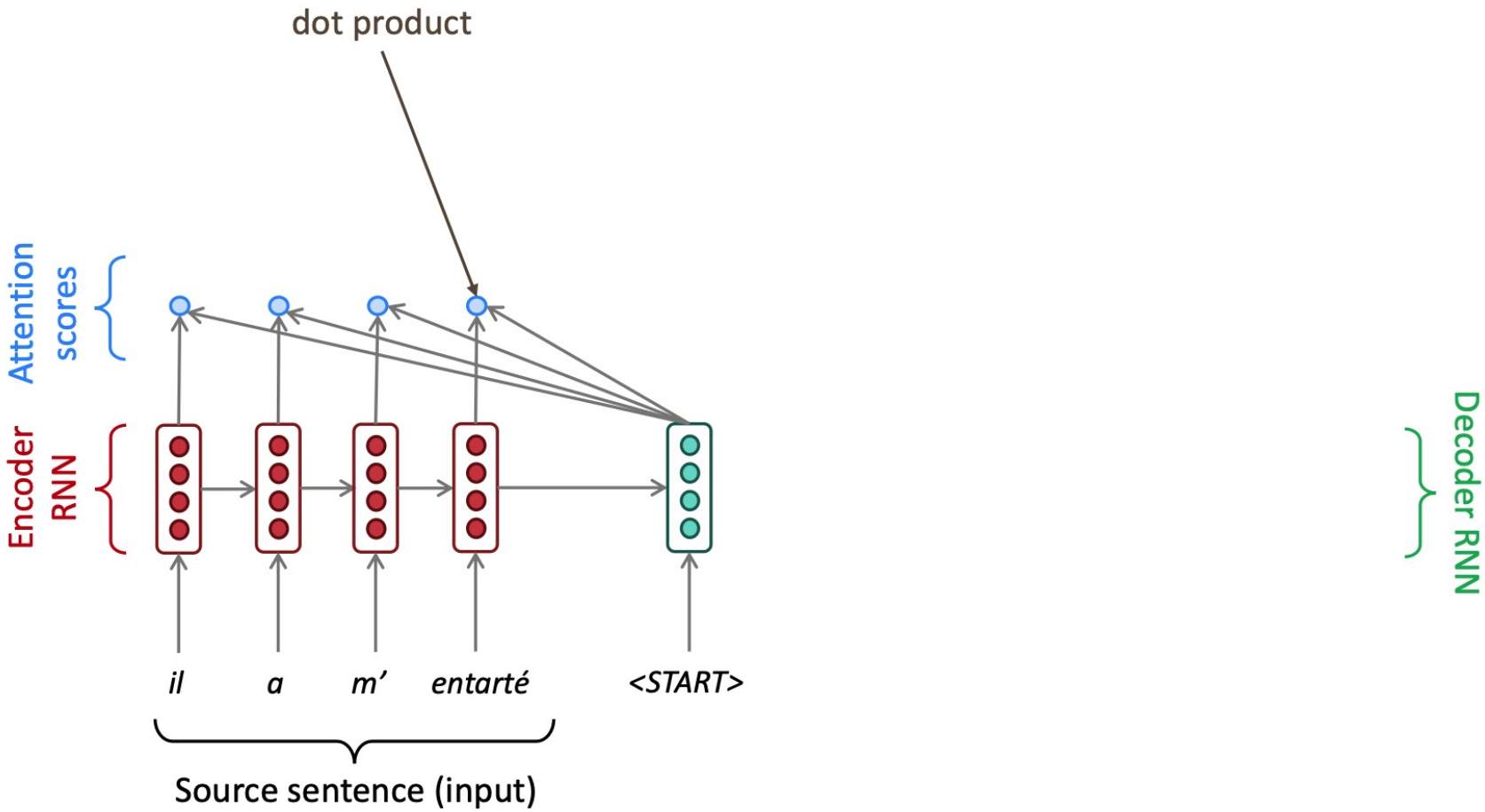
Sequence-to-sequence with attention



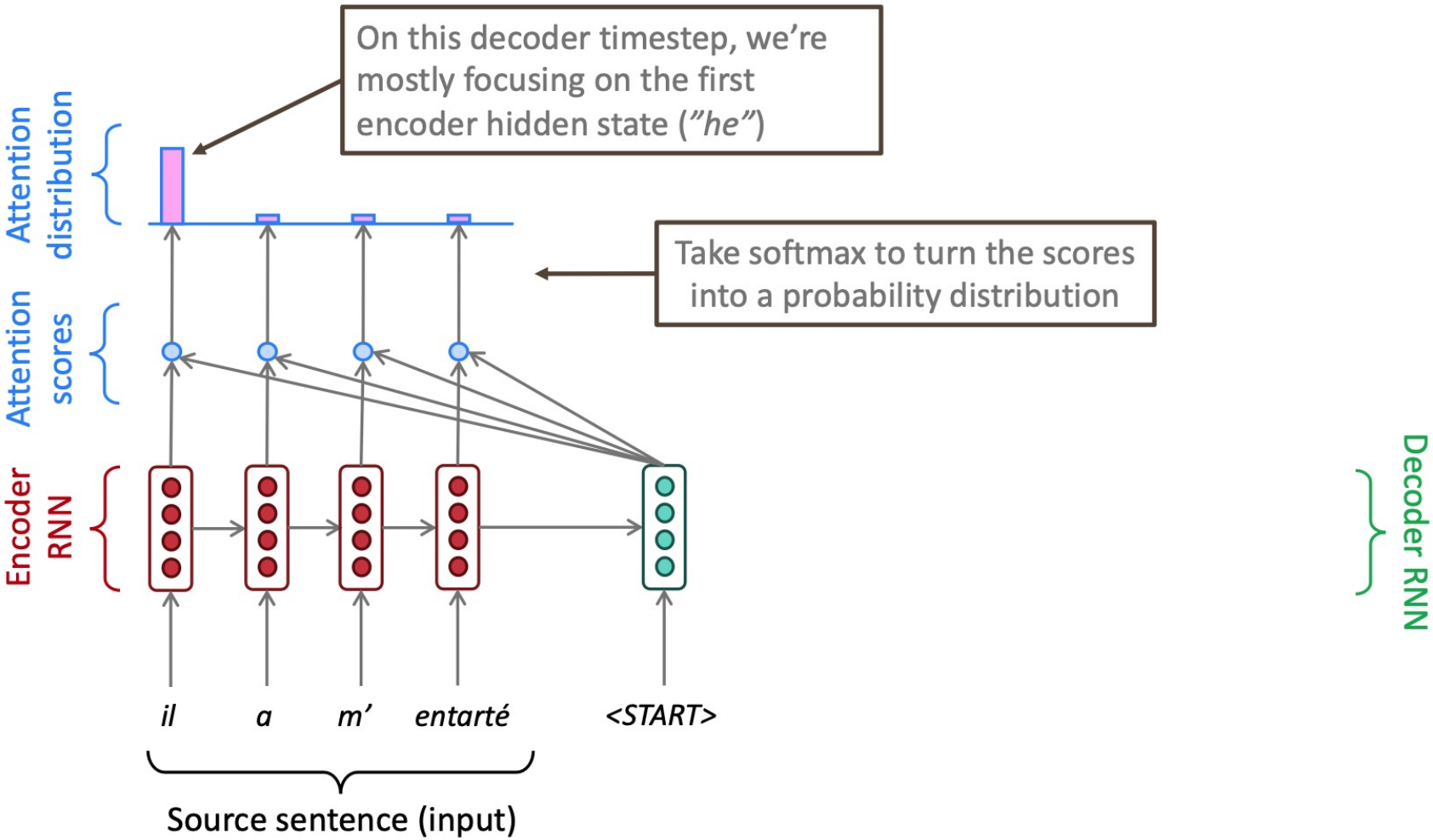
Sequence-to-sequence with attention



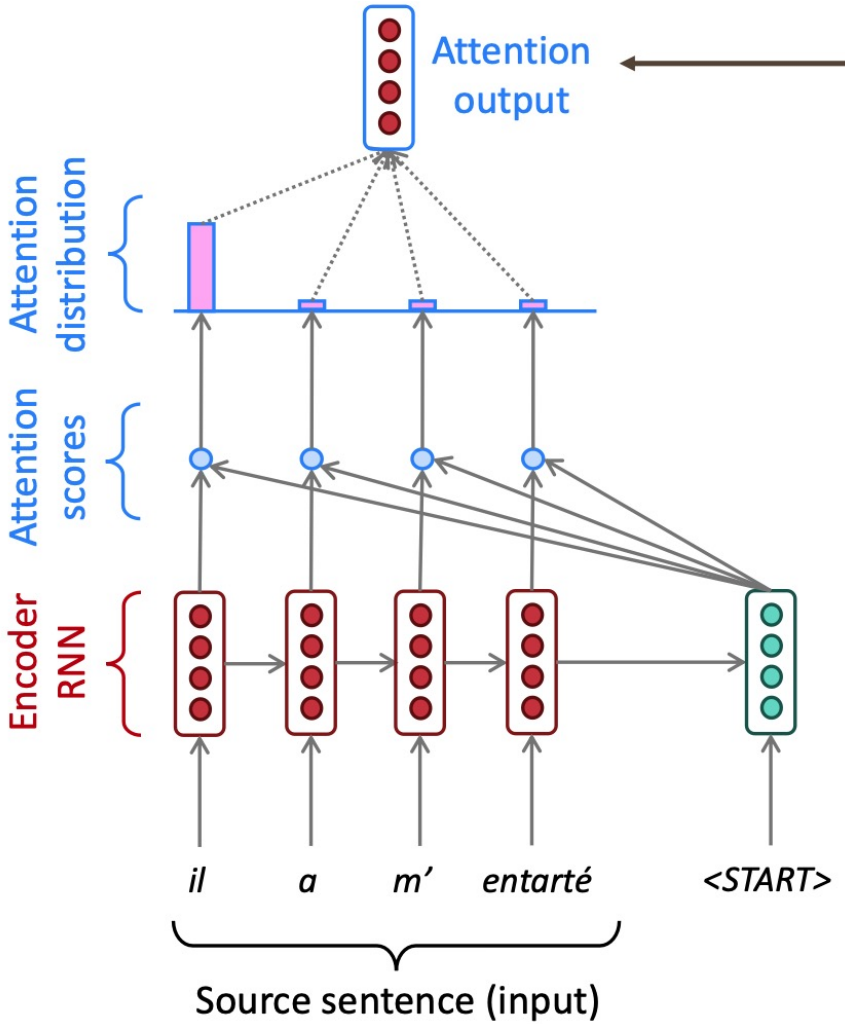
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention

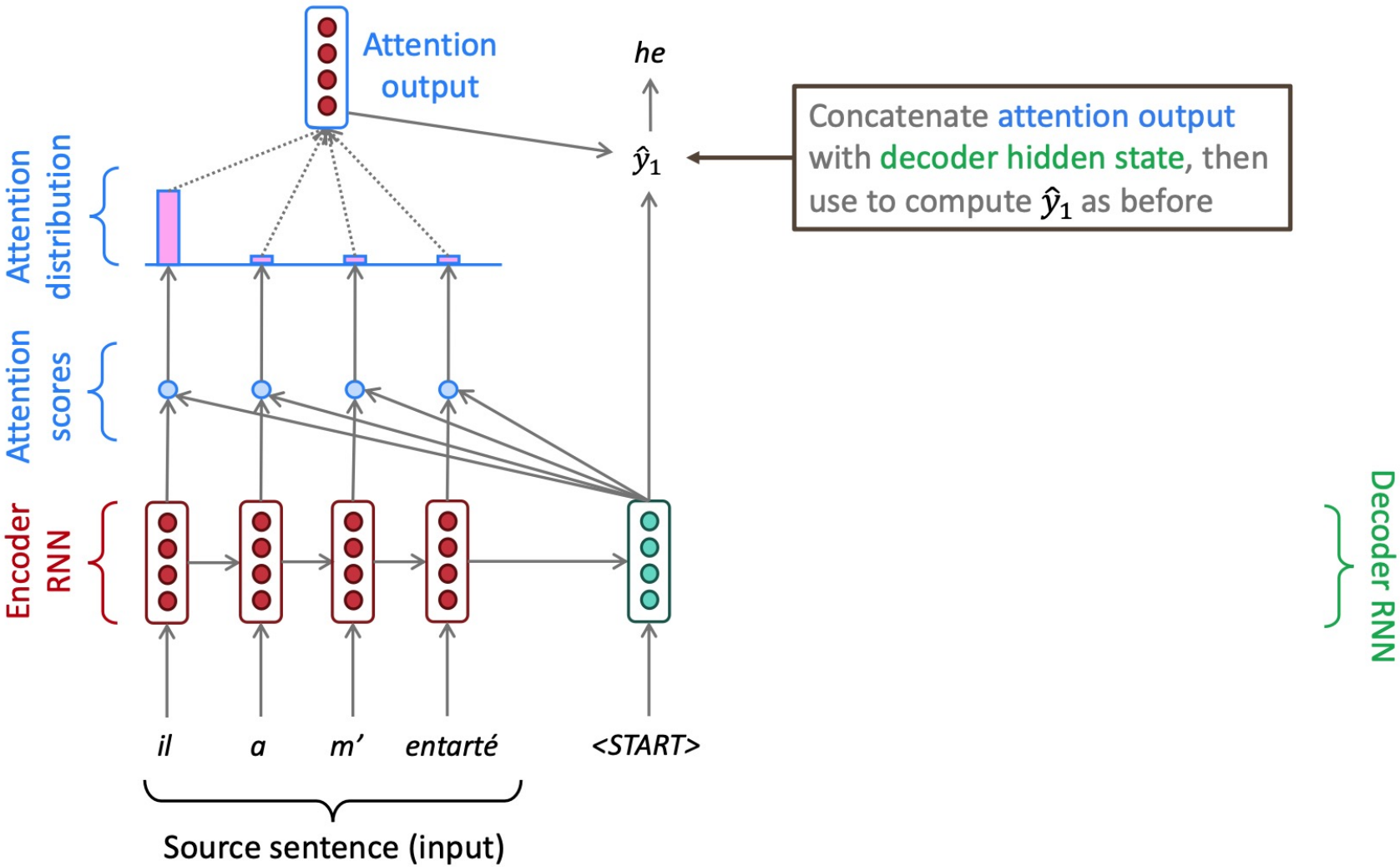


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

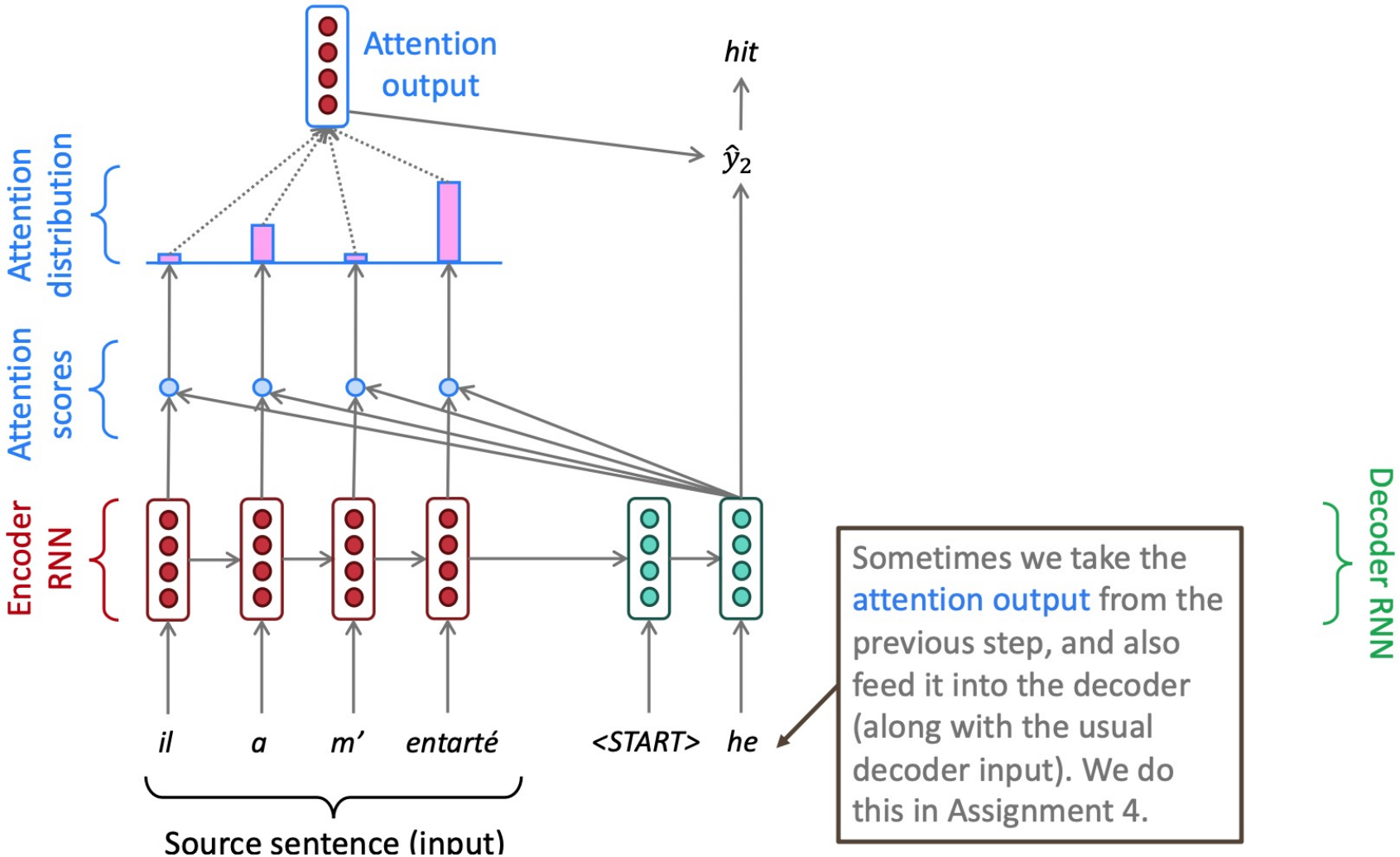
The attention output mostly contains information from the hidden states that received high attention.

Decoder RNN

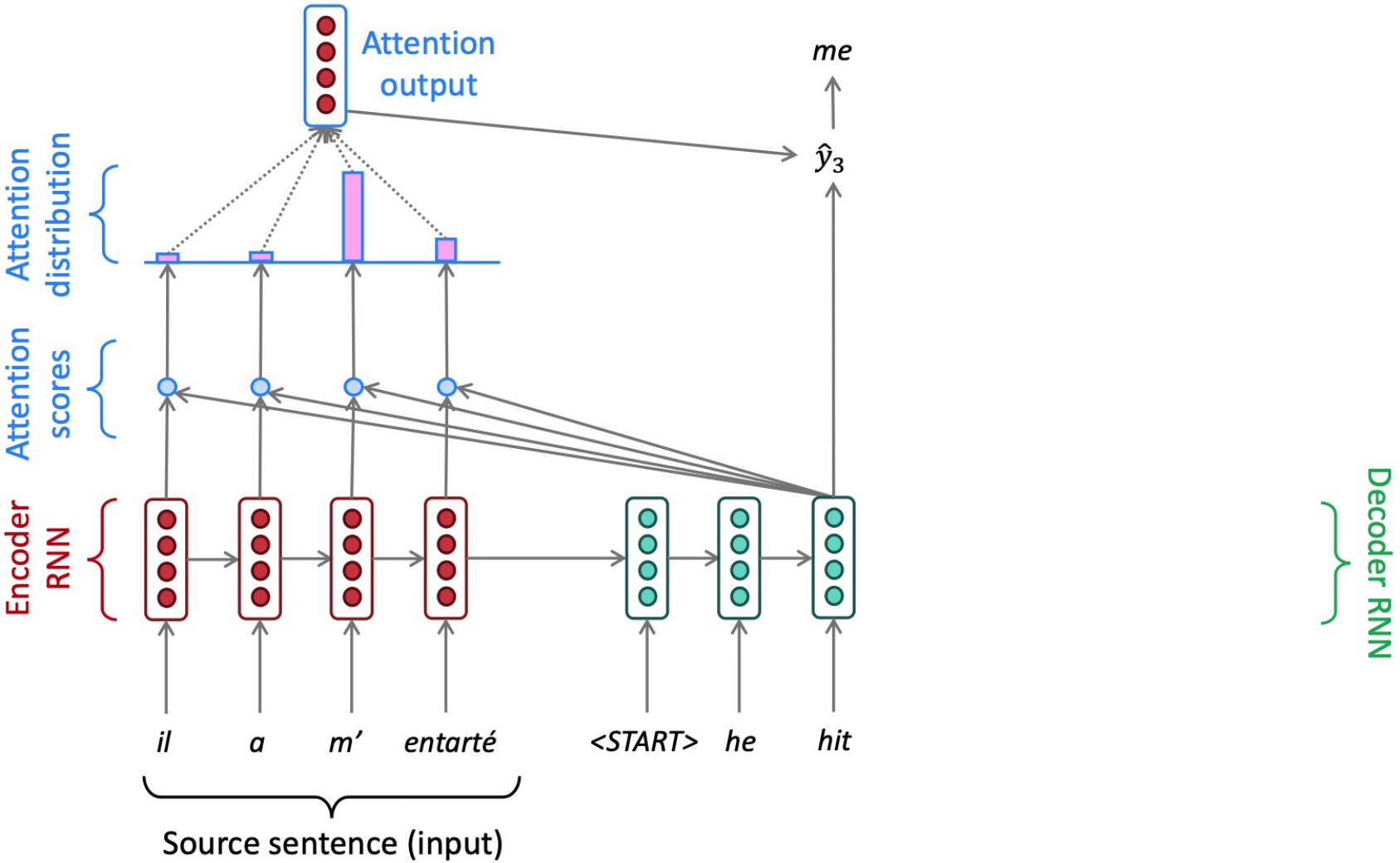
Sequence-to-sequence with attention



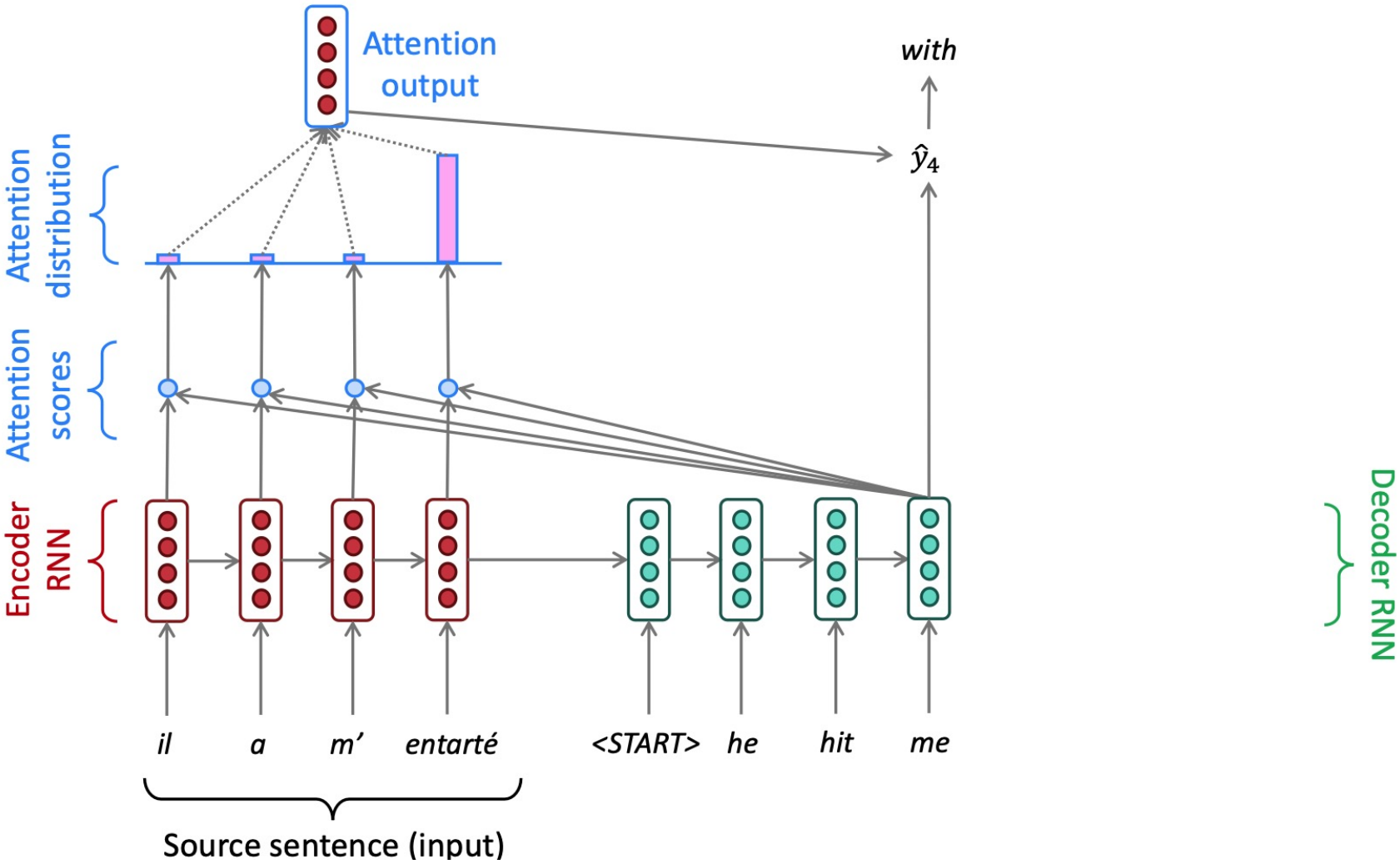
Sequence-to-sequence with attention



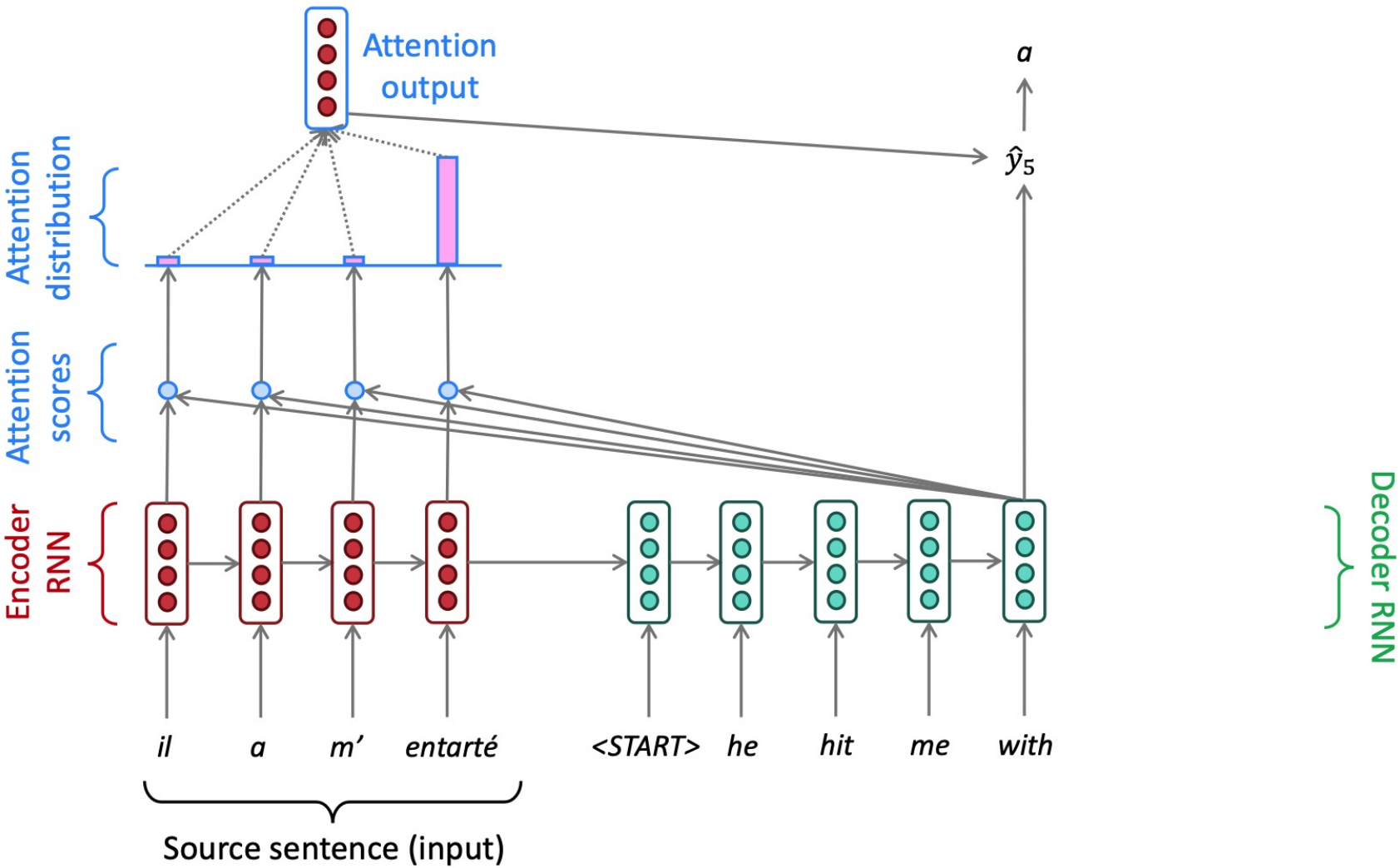
Sequence-to-sequence with attention



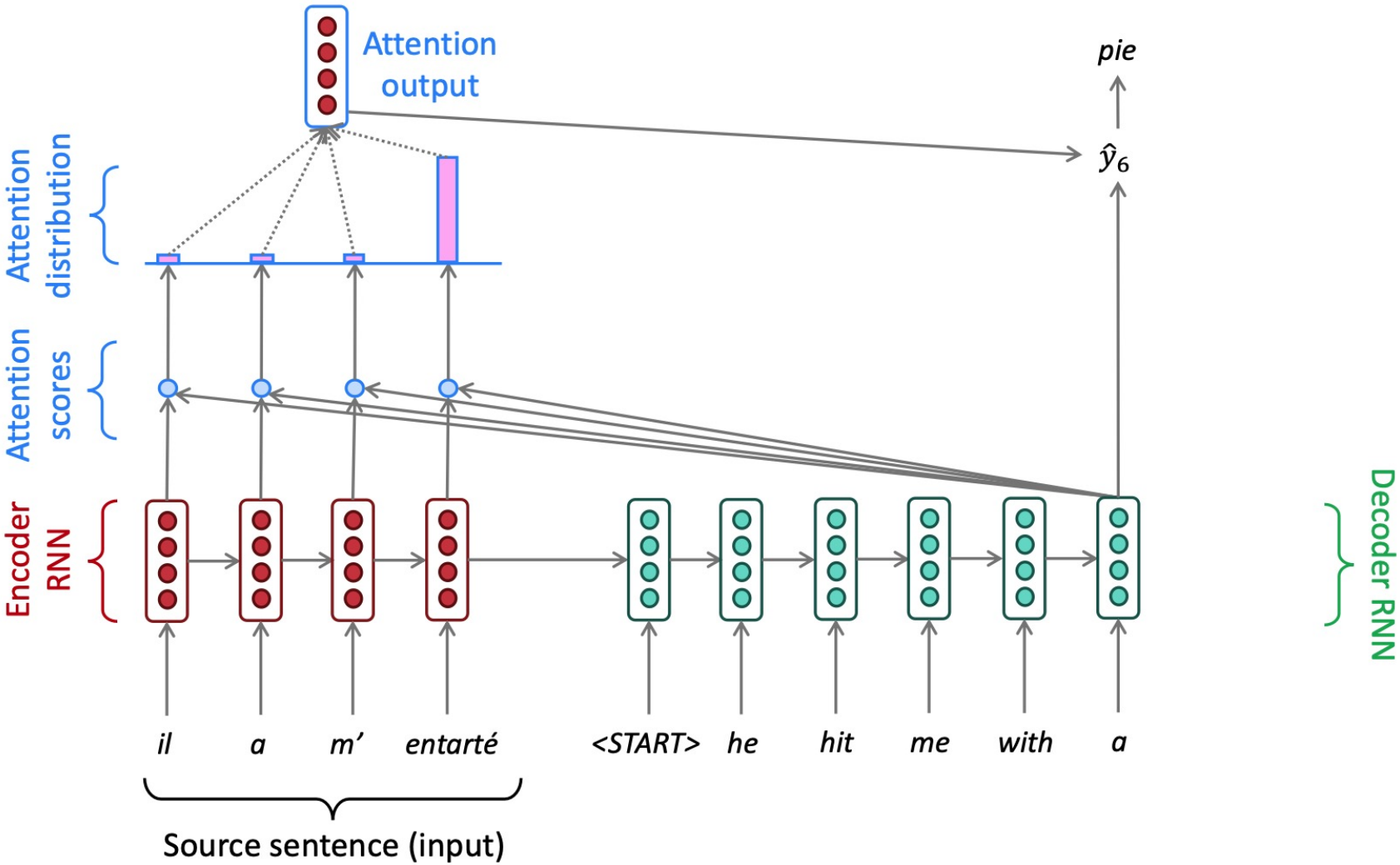
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention



Attention: In Equations

- Encoder hidden states: $h_1, h_2, \ldots, h_N \in \mathbb{R}^h$

Attention: In Equations

- Encoder hidden states: $h_1, h_2, \ldots, h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden $s_t \in \mathbb{R}^h$

Attention: In Equations

- Encoder hidden states: $h_1, h_2, \ldots, h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = \left[s_t^\top h_1,\ s_t^\top h_2,\ \ldots,\ s_t^\top h_N\right] \in \mathbb{R}^N.$$

Attention: In Equations

- Encoder hidden states: $h_1, h_2, \ldots, h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = \left[s_t^\top h_1,\ s_t^\top h_2,\ \ldots,\ s_t^\top h_N\right] \in \mathbb{R}^N.$$

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

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

Attention: In Equations

- Encoder hidden states: $h_1, h_2, \ldots, h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = \left[s_t^\top h_1,\ s_t^\top h_2,\ \ldots,\ s_t^\top h_N\right] \in \mathbb{R}^N.$$

- 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

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

Attention: In Equations

- Encoder hidden states: $h_1, h_2, \ldots, h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = \left[s_t^\top h_1,\ s_t^\top h_2,\ \ldots,\ s_t^\top h_N\right] \in \mathbb{R}^N.$$

- 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 great!

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

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 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 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 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!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself

There are several attention variants

- We have some *values* $h_1, \ldots, h_N \in \mathbb{R}^{d_1}$ and a *query* $s \in \mathbb{R}^{d_2}$
- Attention always involves:
  1. Computing the *attention scores* $e \in \mathbb{R}^N$
  2. Taking softmax to get *attention distribution* α:

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

  3. Using attention distribution to take weighted sum of values:

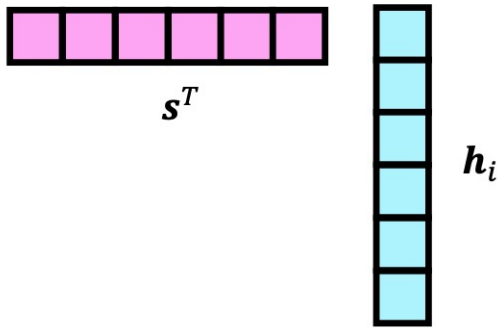
$$a = \sum_{i=1}^{N} \alpha_i h_i \in \mathbb{R}^{d_1}$$

thus obtaining the *attention output* a (sometimes called the *context vector*)

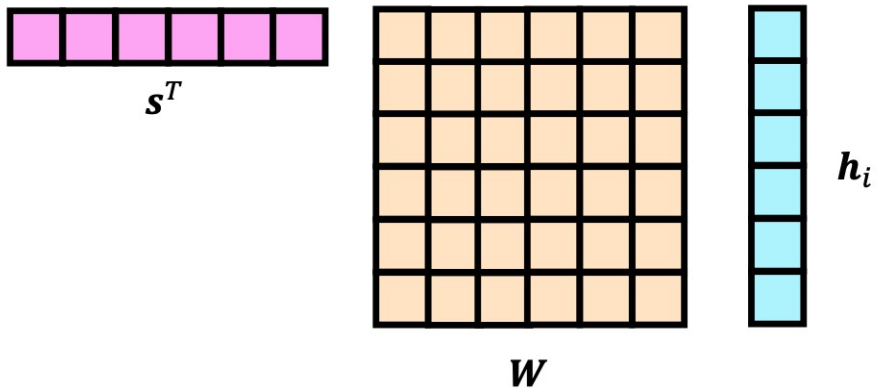
Attention variants

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: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$
  - This assumes $d_1 = d_2$. This is the version we saw earlier.

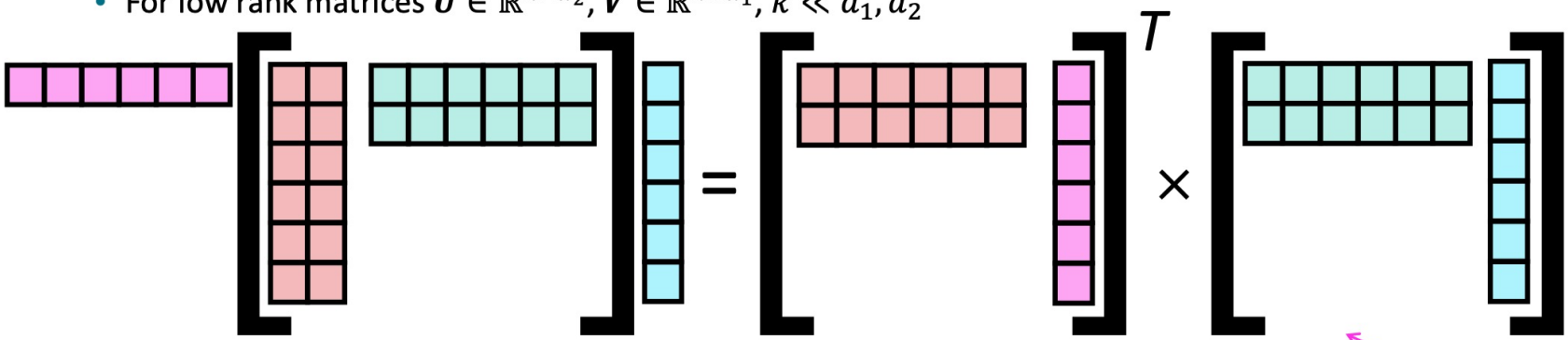


- Multiplicative attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$ [Luong, Pham, and Manning 2015]
  - Where $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix. Perhaps better called "bilinear attention"



Attention variants

- Reduced-rank multiplicative attention: $e_i = s^T(\boldsymbol{U}^T\boldsymbol{V})h_i = (\boldsymbol{U}s)^T(\boldsymbol{V}h_i)$
  - For low rank matrices $\boldsymbol{U} \in \mathbb{R}^{k \times d_2}$, $\boldsymbol{V} \in \mathbb{R}^{k \times d_1}$, $k \ll d_1, d_2$



- Additive attention: $e_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s}) \in \mathbb{R}$ [Bahdanau, Cho, and Bengio 2014]
  - 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
  - "Additive" is a weird/bad name. It's really using a feed-forward neural net layer.

Remember this when we look at Transformers next week!

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)

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!