Seq2Seq Model and Attention Mechanism

Sourav Karmakar

souravkarmakar29@gmail.com

Sequence to Sequence Model

A **Sequence-to-Sequence (Seq2Seq) model** in NLP is a type of neural network architecture designed to transform one sequence of tokens (such as words or characters) into another sequence — potentially of different length.

It consists of two main components:

- **Encoder**: Processes the input sequence and compresses its information into a fixed-length context vector.
- **Decoder:** Generates the output sequence step by step, using the context vector (and possibly previous outputs) to predict the next token.

Hence, seq2seq models are often called encoder-decoder architecture.

Seq2seq model is very versatile and is useful for a number of NLP tasks, like:

- **Neural Machine Translation** (translating English to French)
- **Text Summarization** (long text to short text)
- **Paraphrasing** (express the same meaning with different words)
- **Dialogue generation** (previous utterance to next utterance)
- **Code generation** (natural language to computer language)
- **Question answering** (question to answers)

We will focus on Neural Machine Translation (NMT) to understand seq2seq model.

Sequence to Sequence Model

The sequence-to-sequence model is an example of **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.

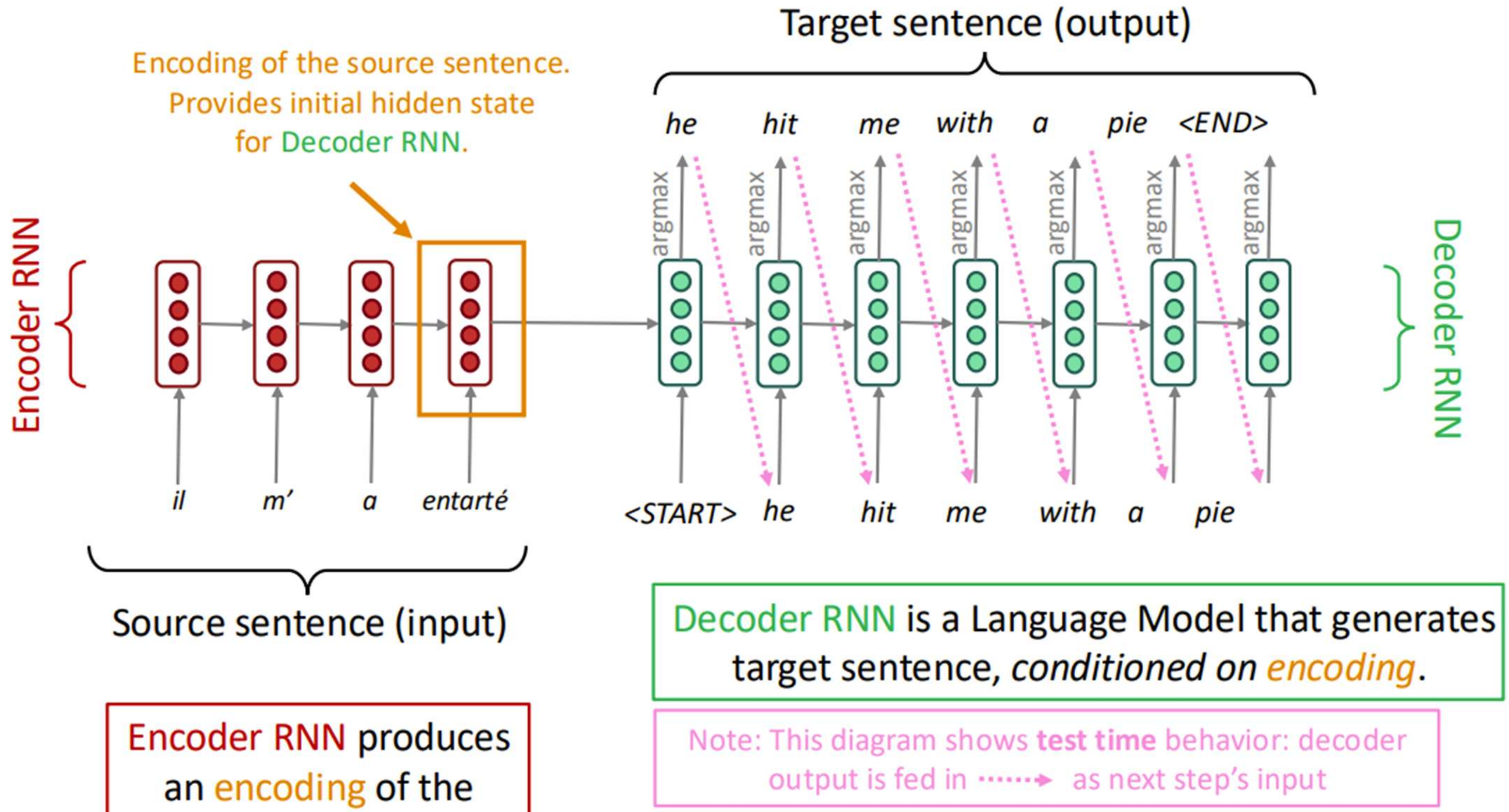
Neural Machine Translation directly calculates $P(y|x)$:

$$P(y|x) = P(y_1|x)\, P(y_2 \mid y_1, x)\, P(y_3 \mid y_1, y_2, x)\, \ldots \underbrace{P(y_T|y_1, y_2, \ldots, y_{T-1}, x)}_{\text{Probability of the next target word, given target words so far and source sentence } x}$$

How to train an NMT system?

We need to get a big parallel corpus with lots of examples of machine translation from one language to another.

Sequence to Sequence Model



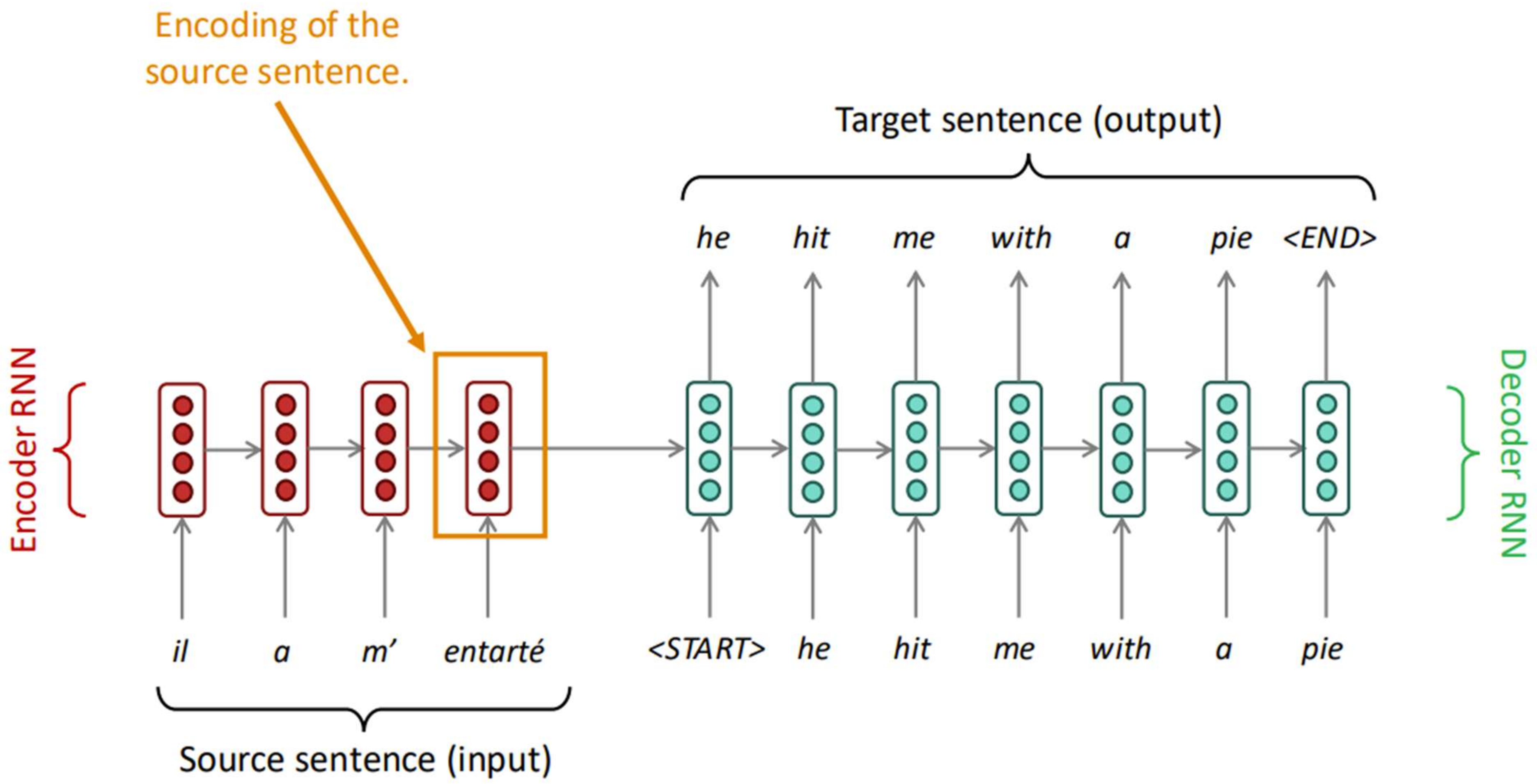
Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

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

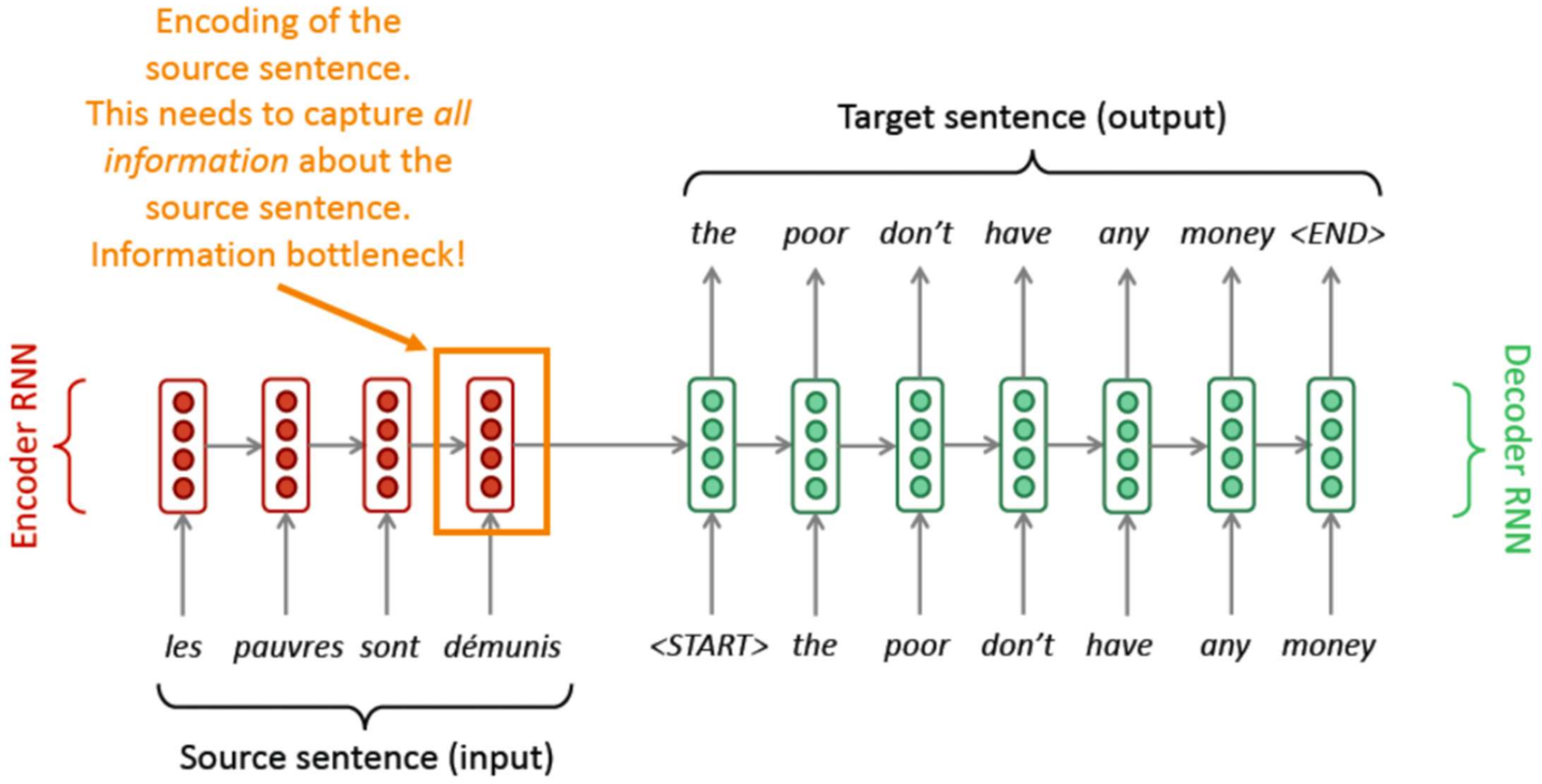
Encoder RNN produces an encoding of the source sentence.

Sequence to Sequence Model

Problem with this architecture?



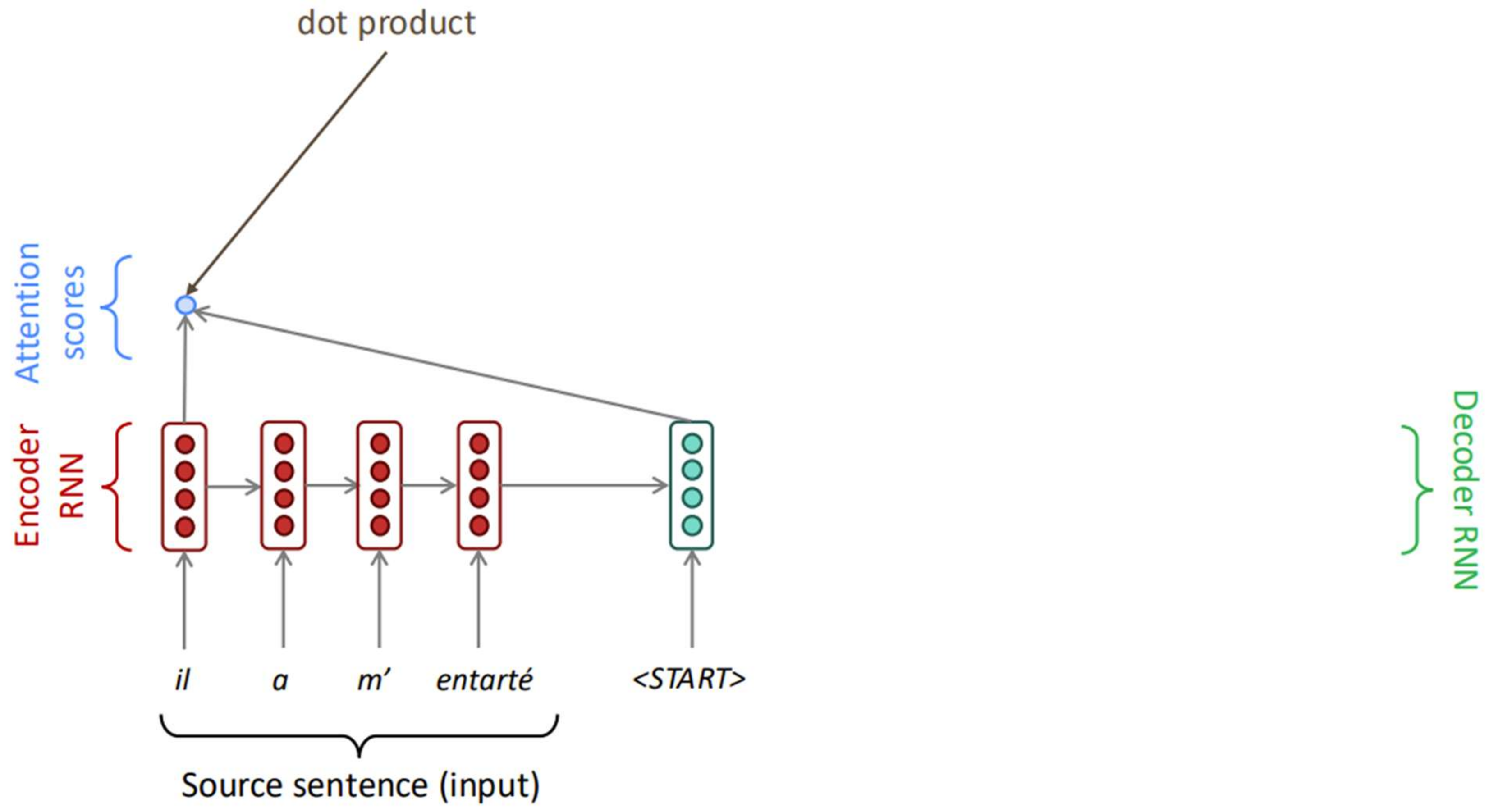
Sequence to Sequence Model: the Bottleneck



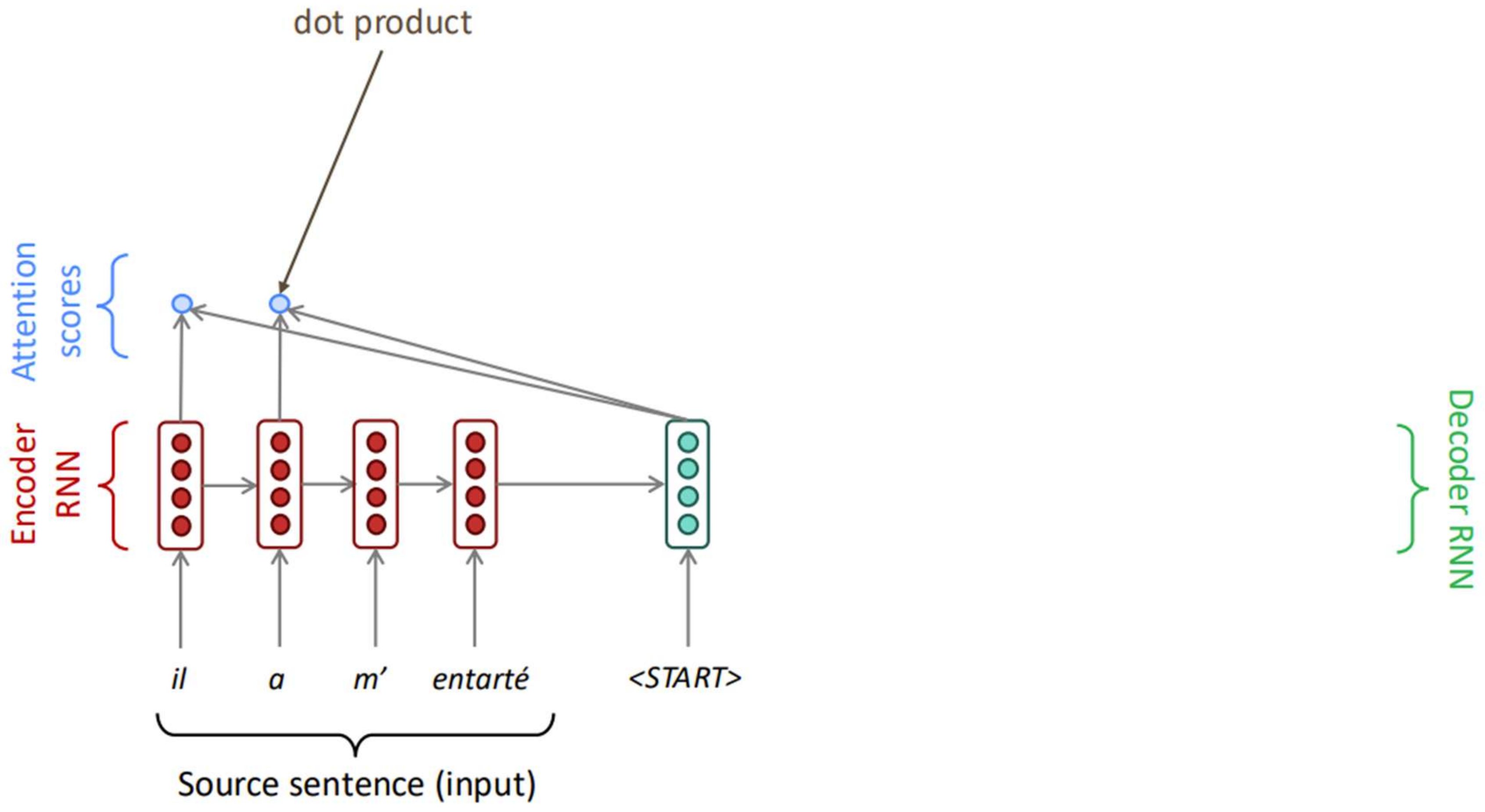
Sequence to Sequence Model with Attention

Attention provides a solution to bottleneck problem.

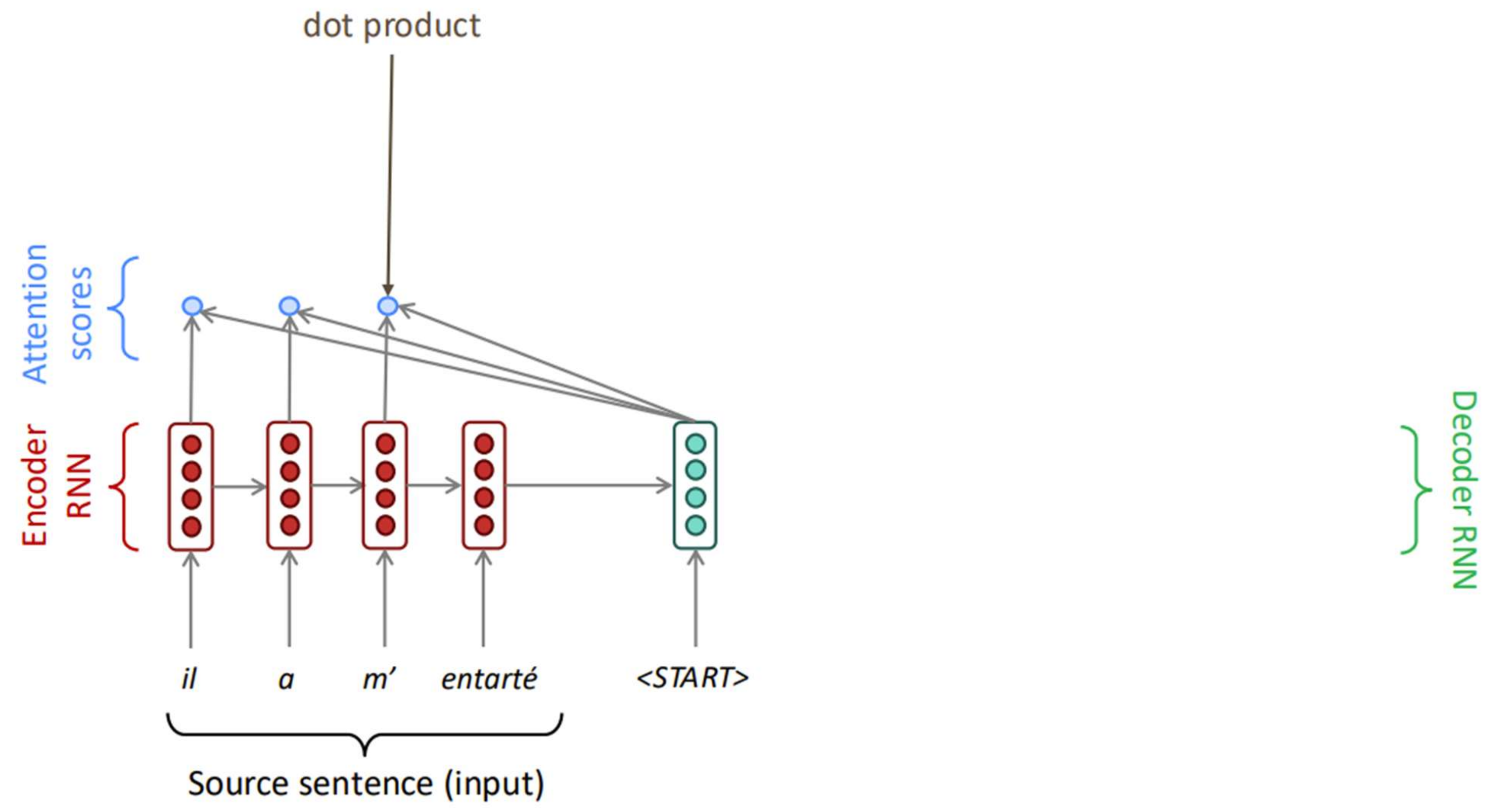
Key Idea: On each step of the decoder, *use direct connection to the encoder* to *focus on a particular part* of the source sequence.



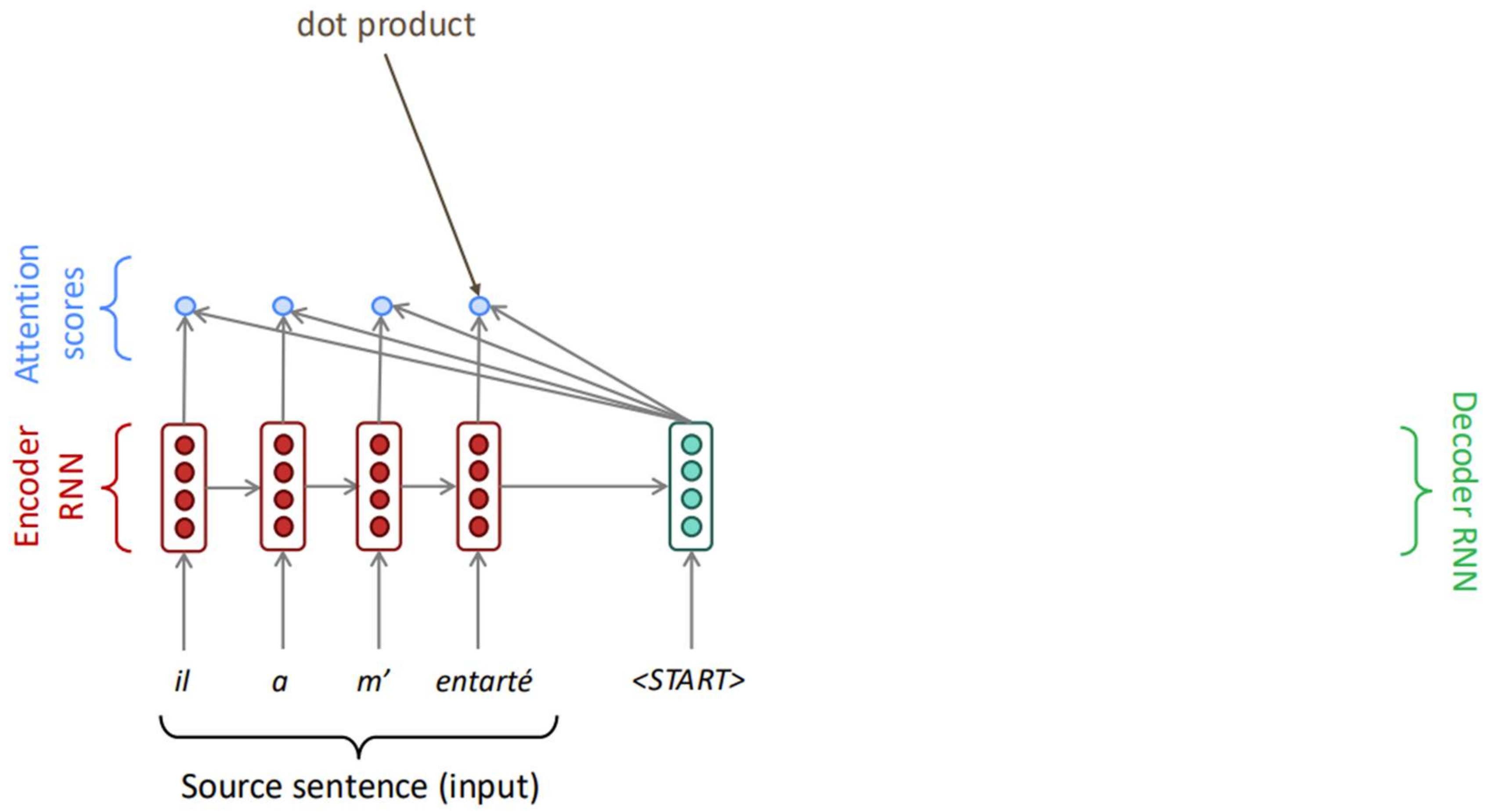
Sequence to Sequence Model with Attention



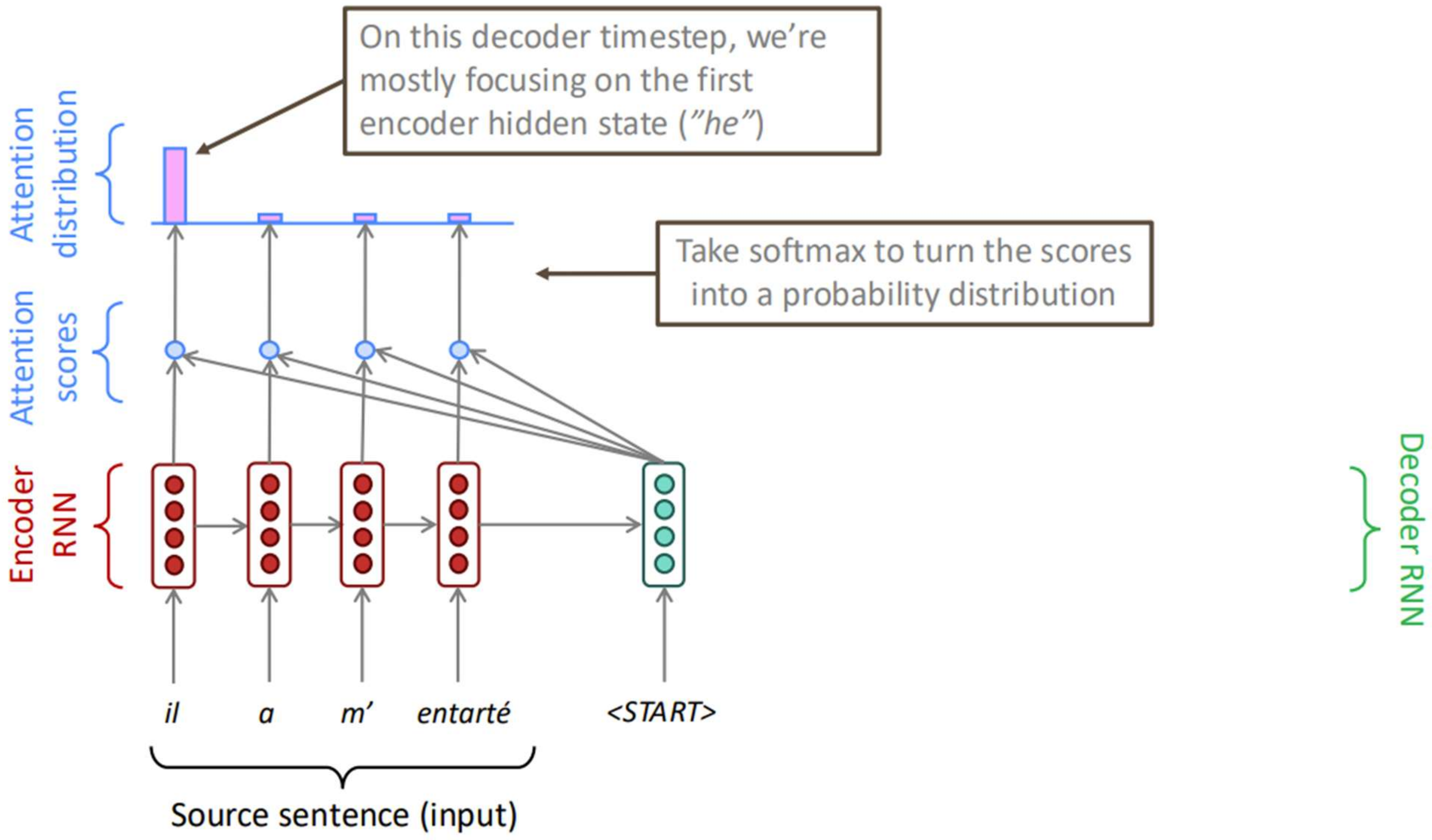
Sequence to Sequence Model with Attention



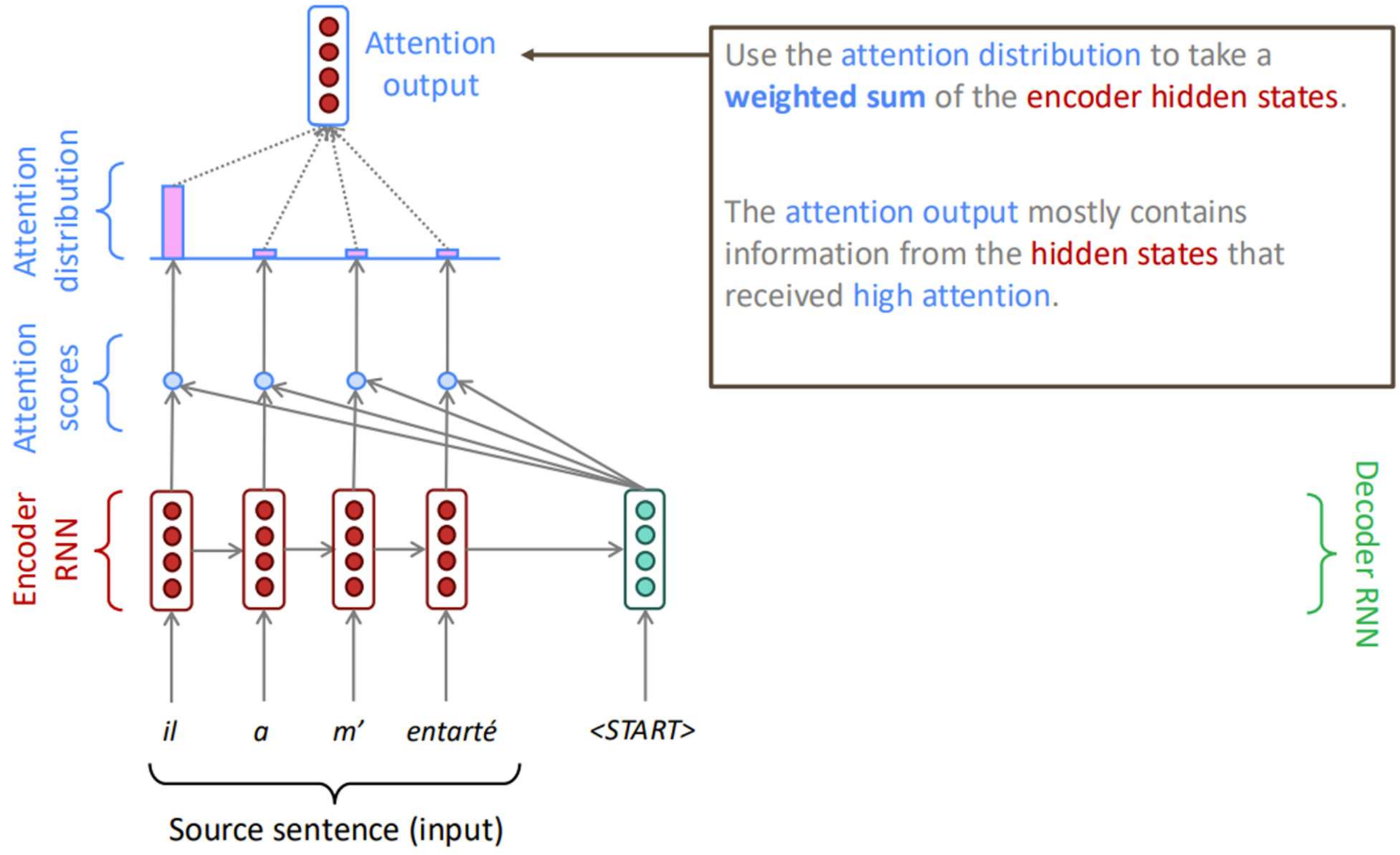
Sequence to Sequence Model with Attention



Sequence to Sequence Model with Attention


On this decoder timestep, we're mostly focusing on the first encoder hidden state ("he")
Take softmax to turn the scores into a probability distribution
Attention distribution
Attention scores
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
Source sentence (input)

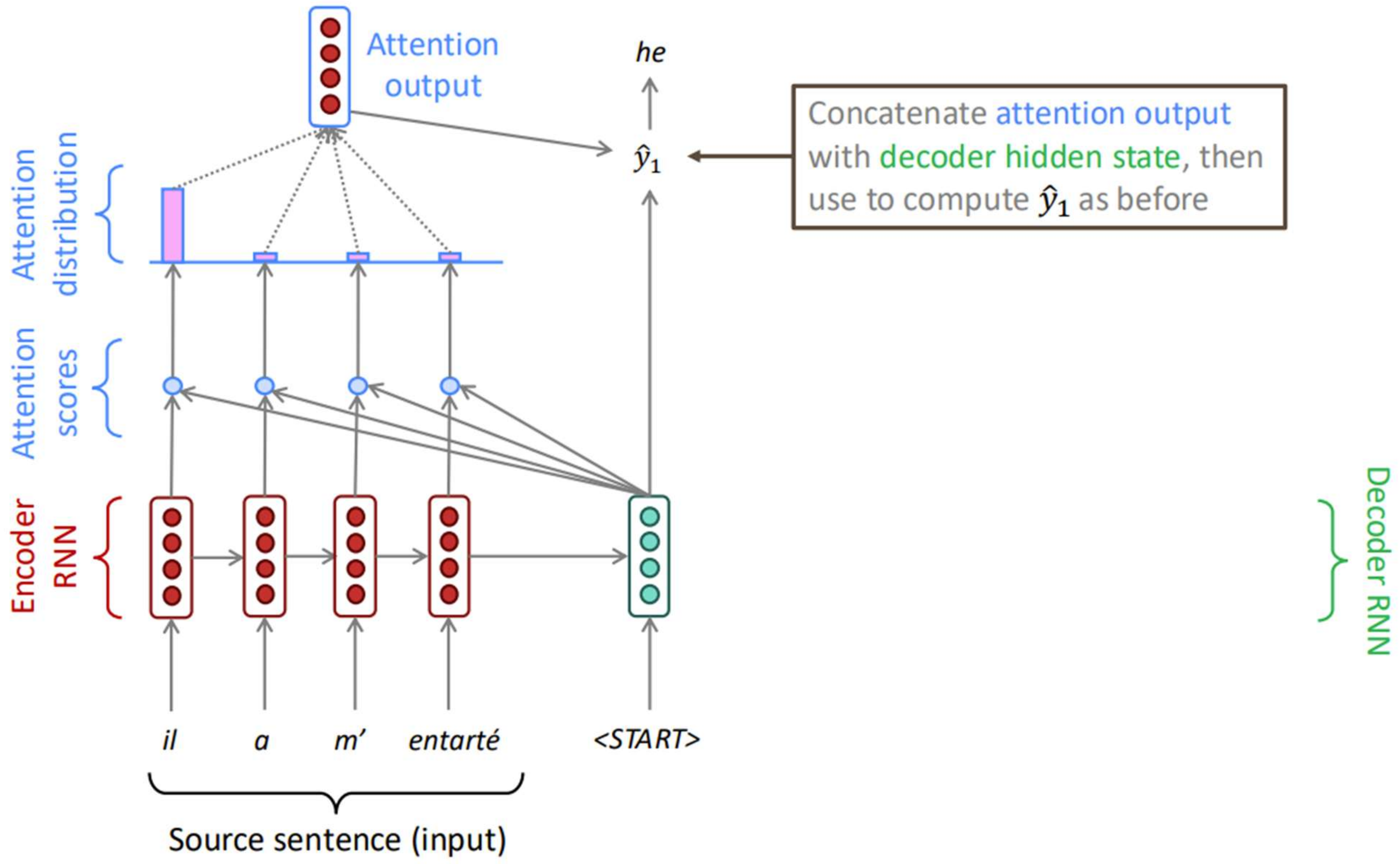
Sequence to Sequence Model with Attention



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

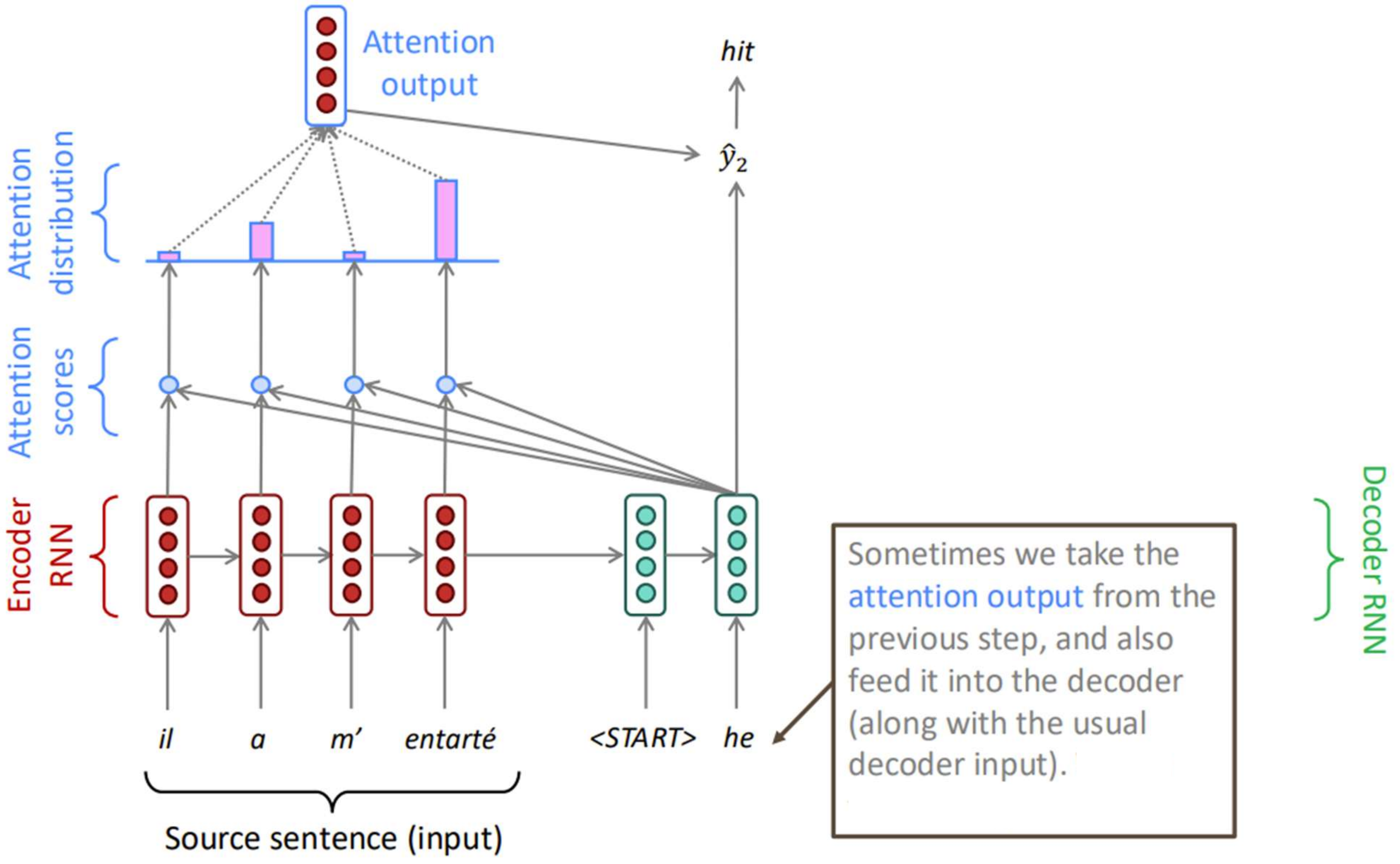
The attention output mostly contains information from the hidden states that received high attention.

Sequence to Sequence Model with Attention



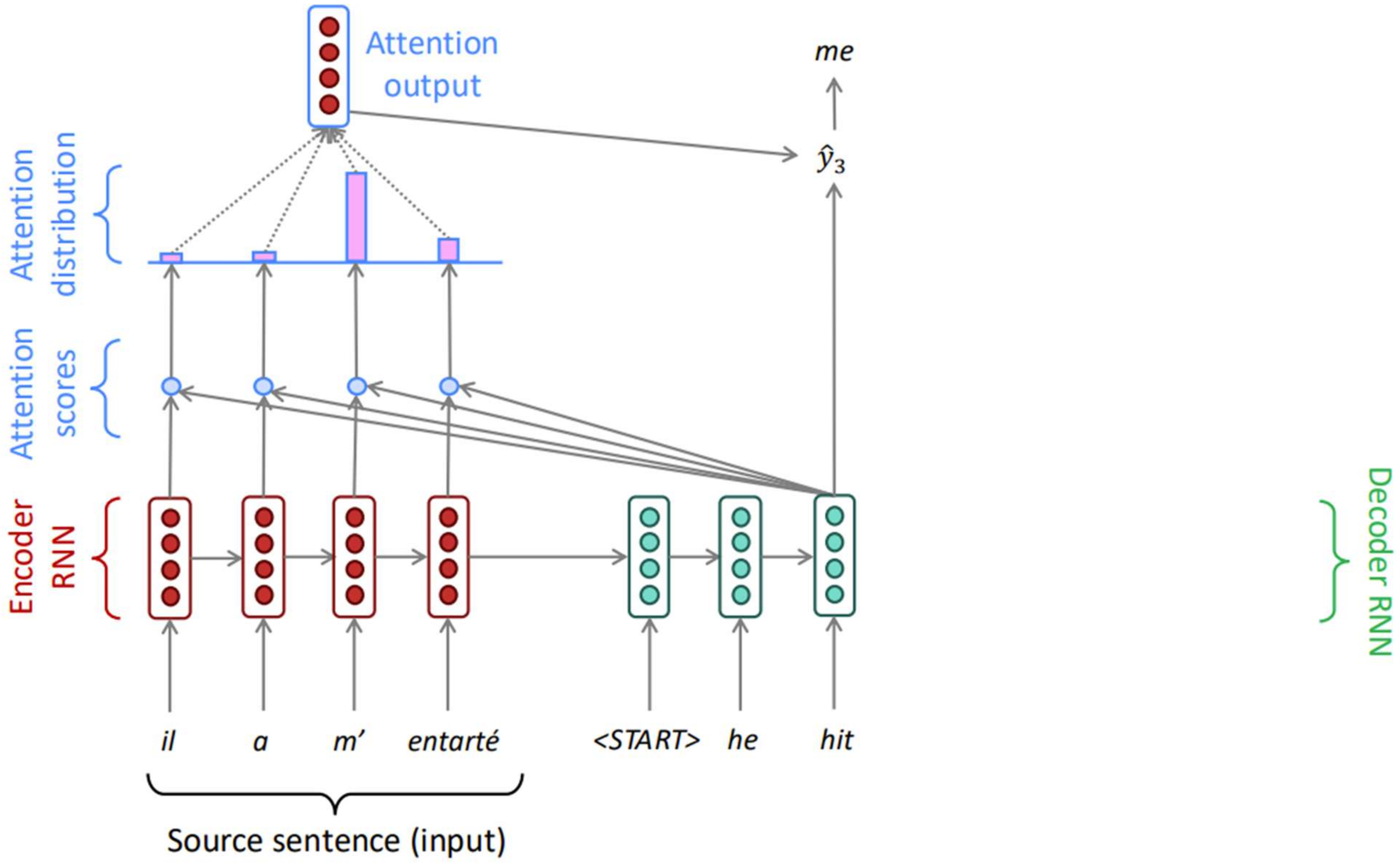
Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

Sequence to Sequence Model with Attention

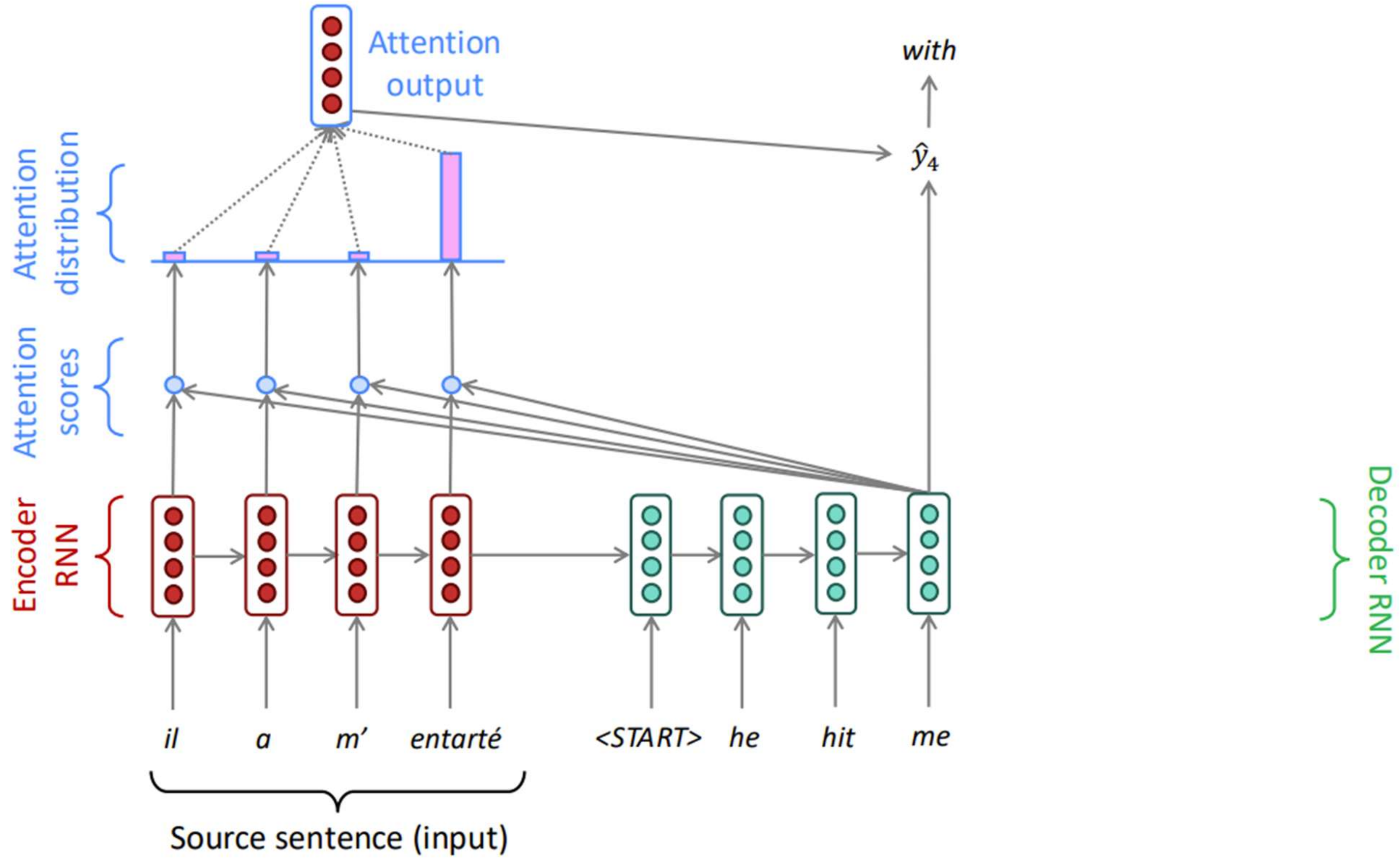


Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input).

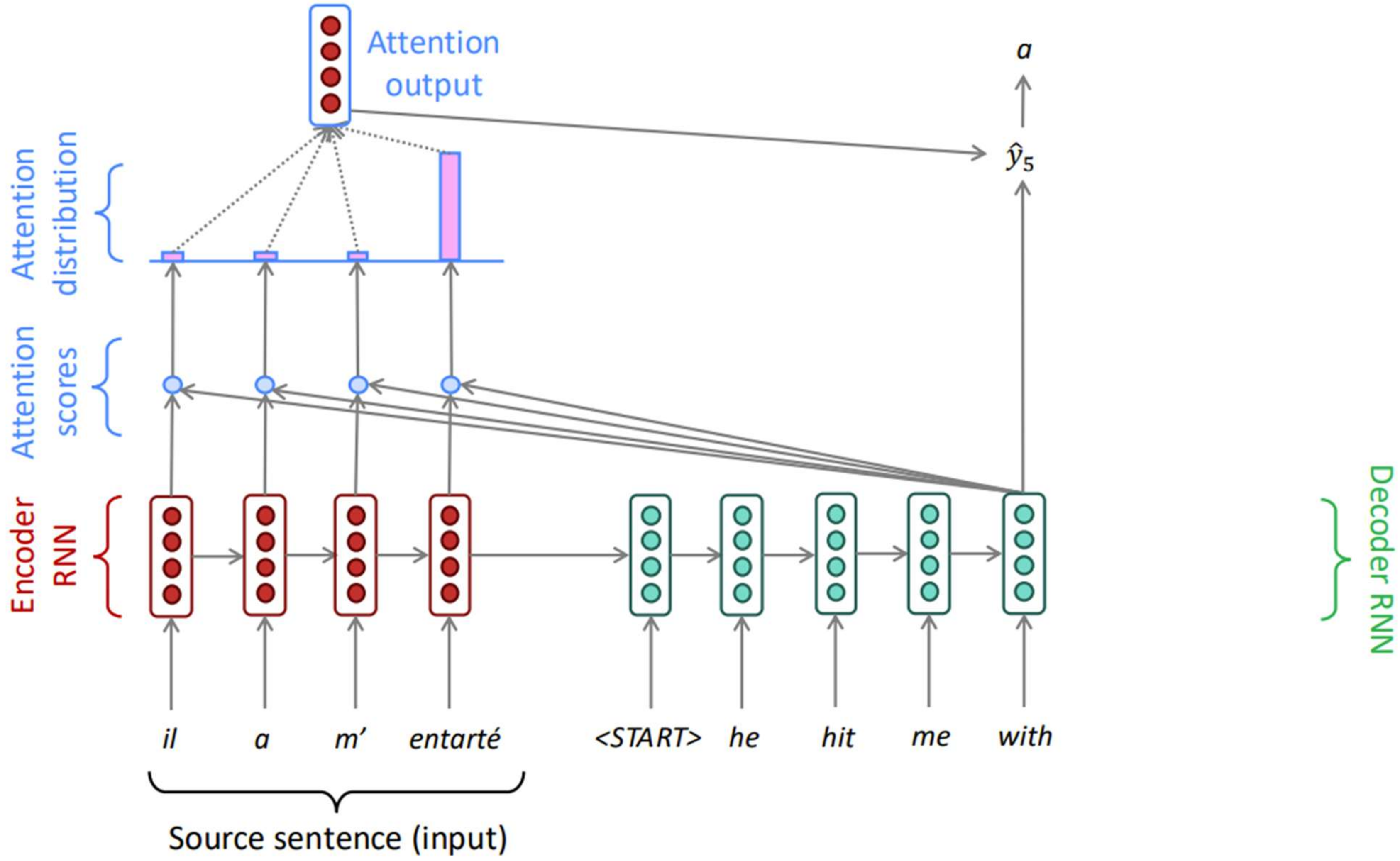
Sequence to Sequence Model with Attention



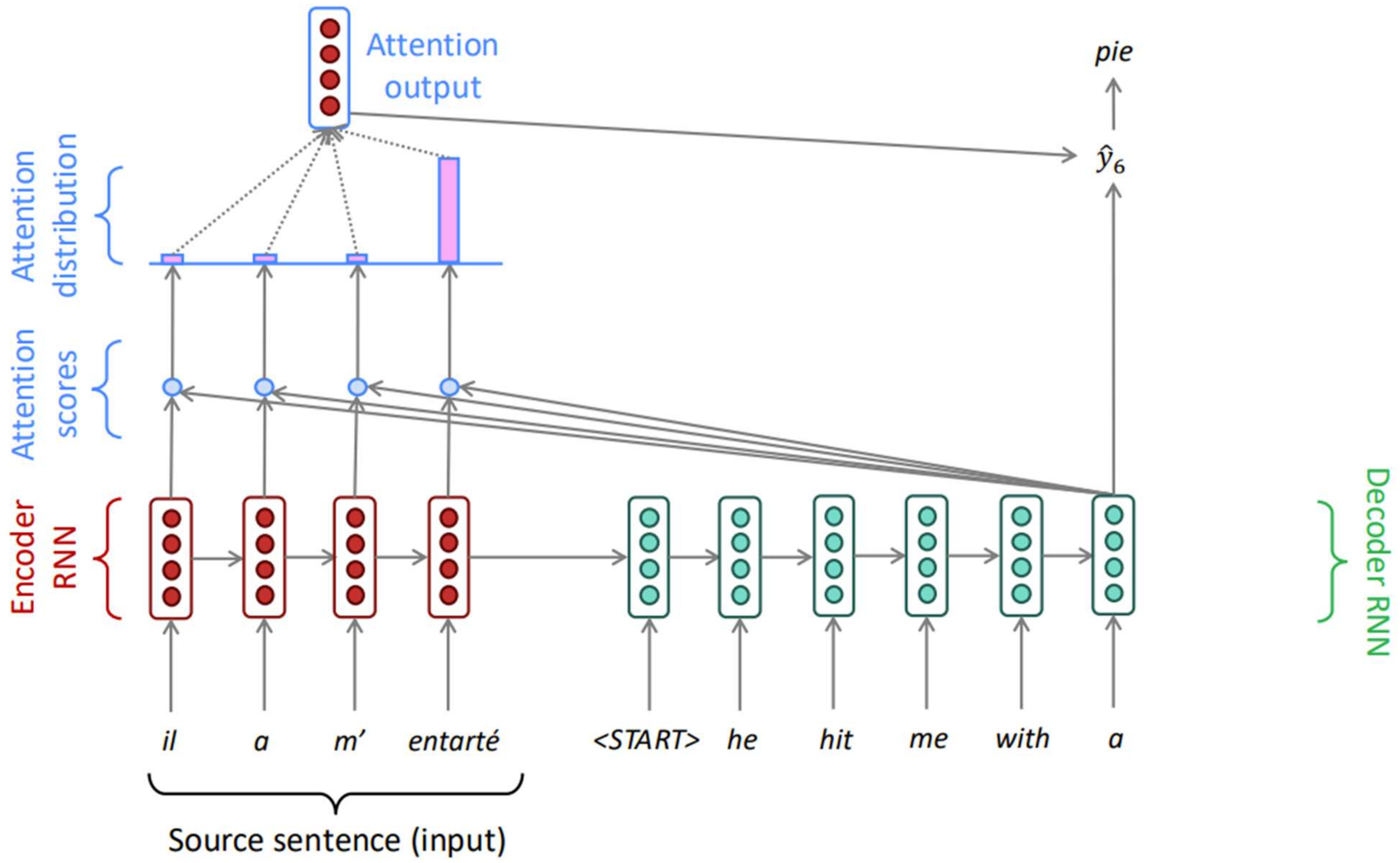
Sequence to Sequence Model with Attention



Sequence to Sequence Model with Attention



Sequence to Sequence Model with Attention



Attention Mechanism

- We have encoder hidden states: $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden state: $\boldsymbol{s}_t \in \mathbb{R}^h$
- We get the attention scores $\boldsymbol{e}^{(t)}$ for this step:

$$\boldsymbol{e}^{(t)} = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \boldsymbol{s}_t^T \boldsymbol{h}_2, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution $\boldsymbol{\alpha}^{(t)}$ for this step (this is a probability distribution and sums to 1)

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

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

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

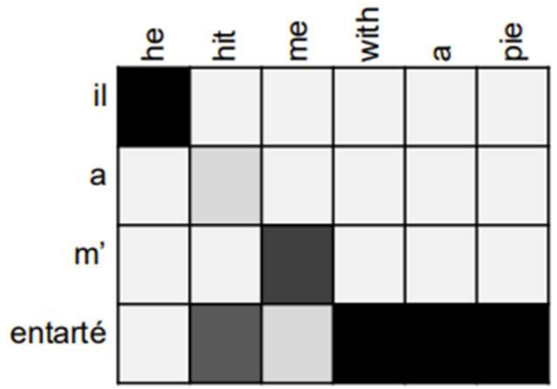
- Finally, we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state $\boldsymbol{s}_t$ and proceed as in the normal seq2seq model

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

Attention Mechanism

Attention is quite helpful

- Attention significantly **improves NMT performances**
  - 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 the vanishing gradient problem**
  - Attention provides a shortcut to faraway steps. Hence, attention-based model can capture long-term dependency.
- Attention **provides some interpretability**
  - By inspecting attention distribution, we can see what the decoder was focusing on.
  - We get alignment for free as the network learns the alignment by itself.
- Attention **is versatile**
  - Attention can be used in many architectures (not just seq2seq).
  - Attention can be used in many tasks (not just machine translation).

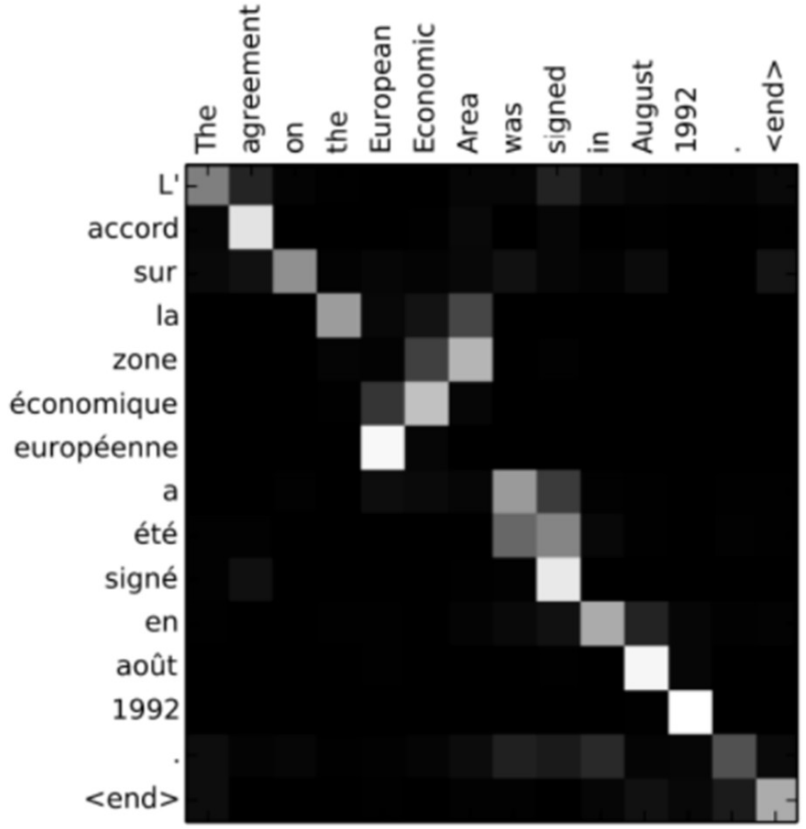


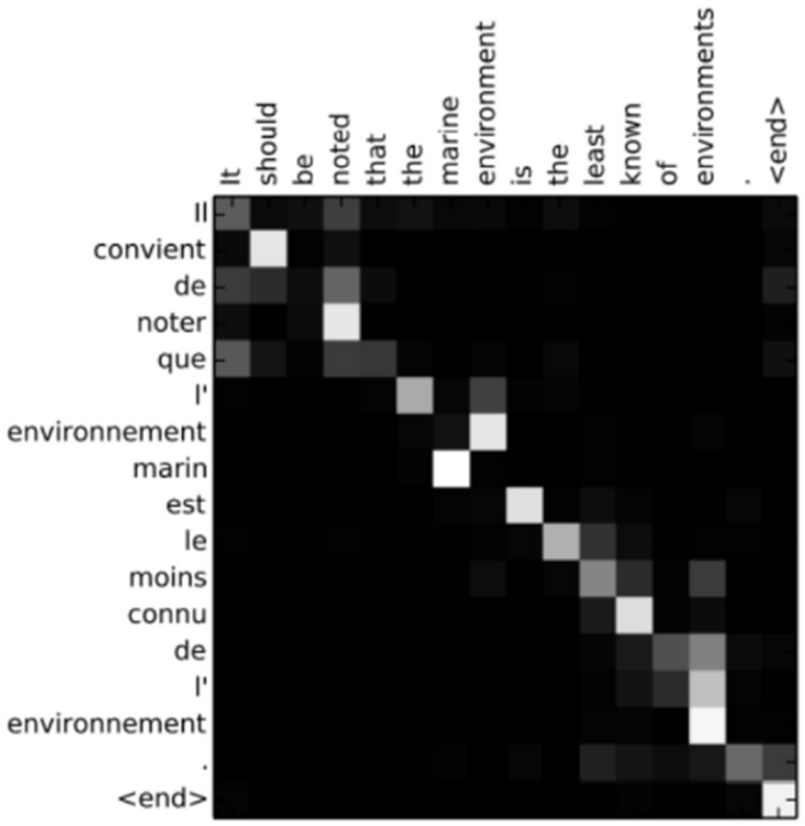
One issue:

Attention has *quadratic cost* with respect to sequence length.
However, attention is parallelizable.

Attention Mechanism

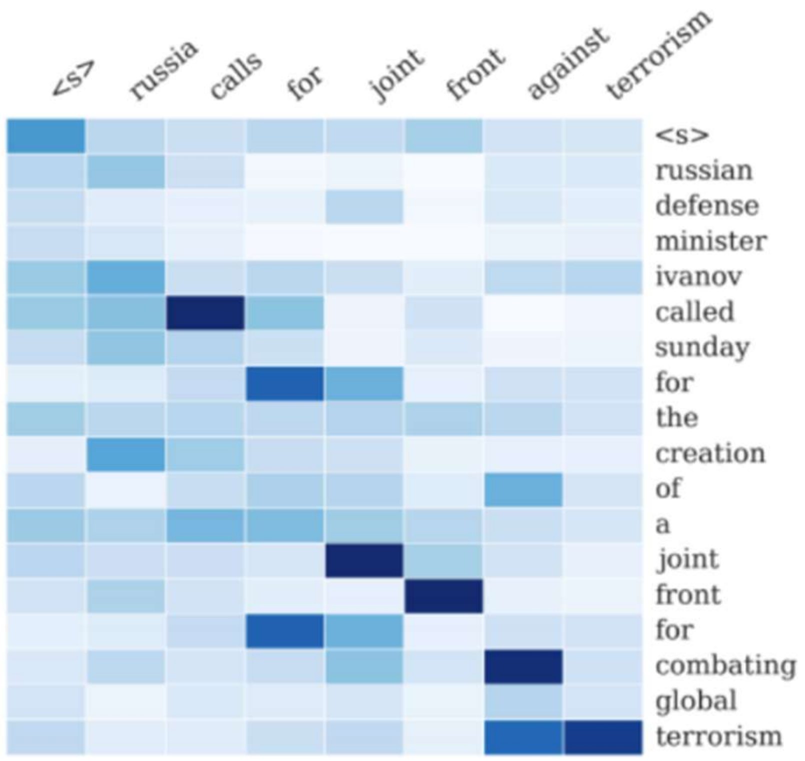
Neural Machine Translation by jointly learning to align and Translate, ICLR 2015



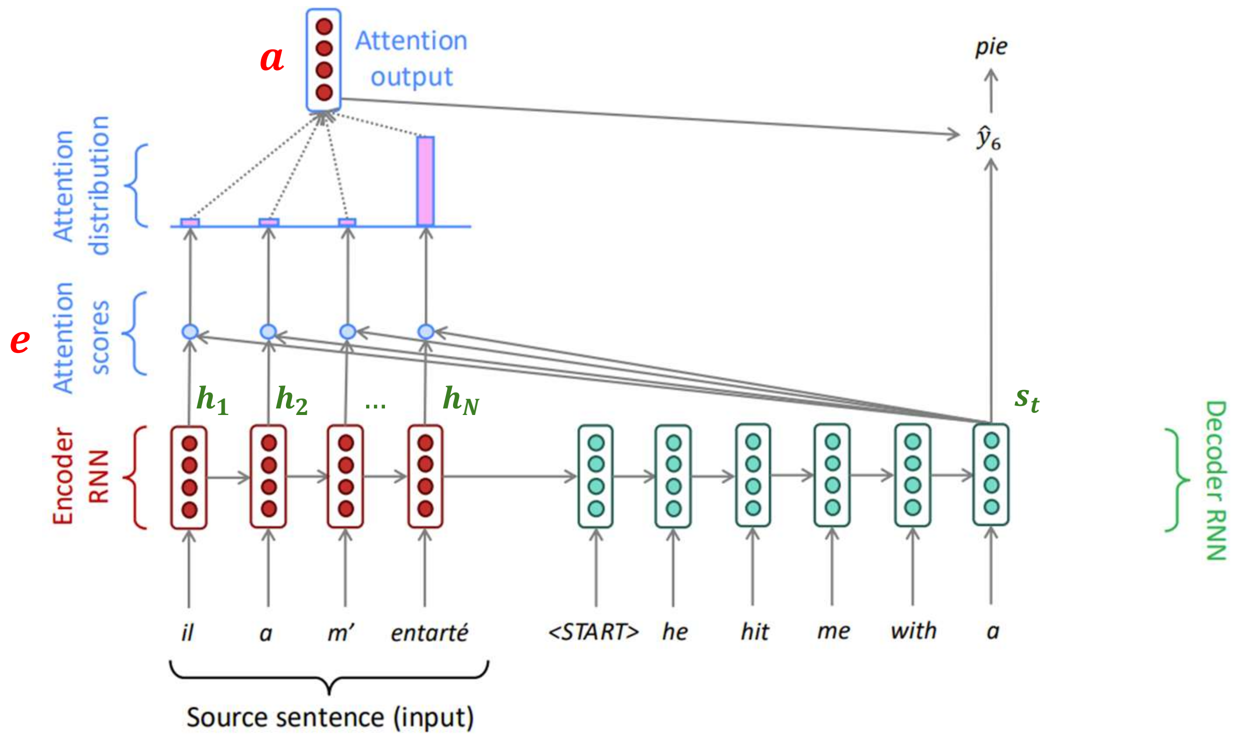


Attention Mechanism

A Neural Attention Model for Sentence Summarization, EMNLP 2015



Different types of Attention

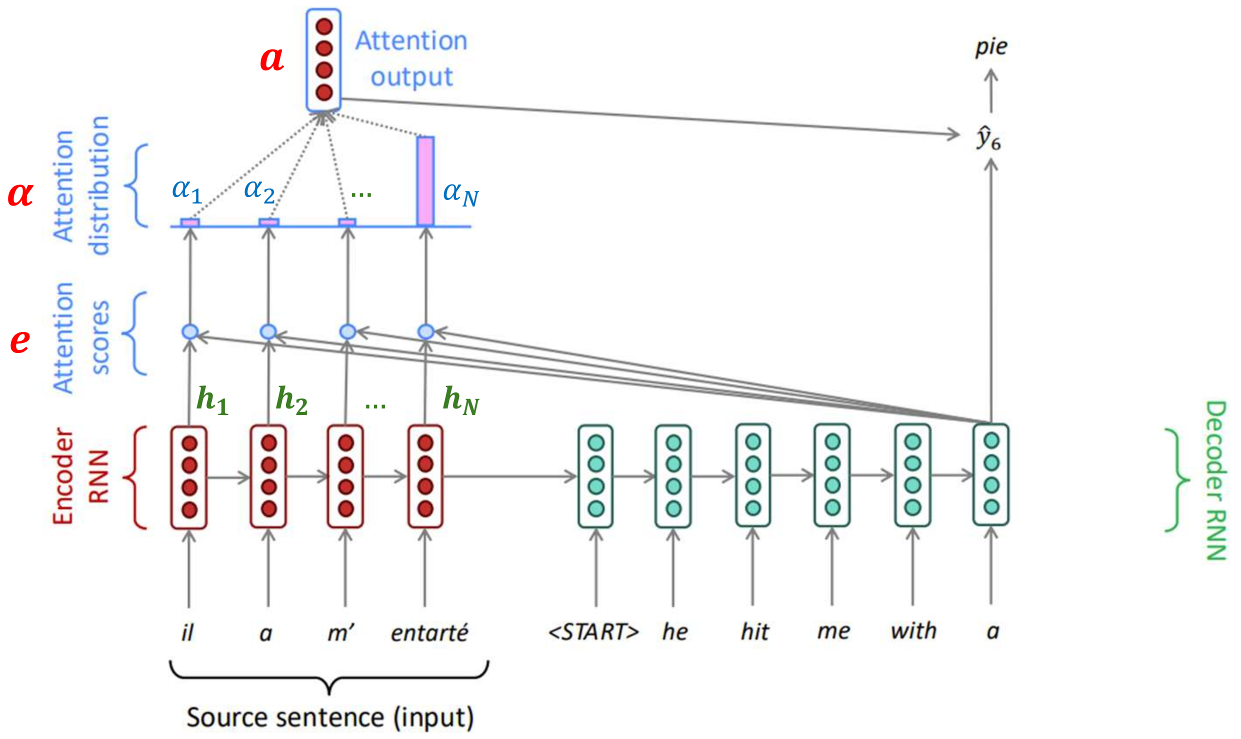


There are different ways of calculating the **attention scores**. Dot product between the encoder hidden states and decoder hidden state is the simplest way of calculating attention scores.

However, if the dimensions of encoder hidden state and decoder hidden state are different, we can't simply calculate dot product (because, dot product can be computed between the vectors of same dimensions)

- Let's assume the dimension of the encoder hidden state is d_1 and dimension of decoder hidden state is d_2 and encoder sequence length is N.
- Attention always involves computing the attention output $\boldsymbol{a} \in \mathbb{R}^{d_1}$ (also called context vector) from the attention scores $\boldsymbol{e} \in \mathbb{R}^{N}$.

Different types of Attention



- We calculate the softmax of attention scores $\boldsymbol{\alpha} = softmax(\boldsymbol{e}) \in \mathbb{R}^N$
- The attention output is calculated as

$$\boldsymbol{a} = \sum_{i=1}^{N} \alpha_i \, \boldsymbol{h}_i \in \mathbb{R}^{d_1}$$

- There are several ways one can compute $\boldsymbol{e} \in \mathbb{R}^{\boldsymbol{N}}$

There are two fundamental approaches for calculating attention weights in seq2seq models. The primary mathematical difference lies in their **alignment scoring function** and the **decoder hidden state** used. One is called the ***Bahdanau*** (additive) attention and another is called ***Luong*** (multiplicative) attention.

Ref:

1. **Neural Machine Translation by Jointly Learning to Align and Translate** by Bahdanau et. al. (2014) [link]
2. **Effective Approaches to Attention-based Neural Machine Translation** by Luong et. al. (2015) [link]

Different types of Attention

Bahdanau (additive) Attention

Key Idea: instead of taking dot product, Bahdanau proposed using a feed-forward neural network to compute alignment scores – hence called additive attention.

$$\text{Attention Score:} \quad e_i^{(t)} = \boldsymbol{v}_a^T \tanh(\boldsymbol{W}_s\, \boldsymbol{s}_{t-1} + \boldsymbol{W}_h\, \boldsymbol{h}_i)$$

Where,

- $\boldsymbol{s}_{t-1} \in \mathbb{R}^{d_2}$: previous decoder hidden state.
- $\boldsymbol{h}_i \in \mathbb{R}^{d_1}$: i^{th} encoder hidden state. (The original Bahdanau paper used a **bi-directional RNN** encoder, where $\boldsymbol{h}_i$ is the concatenation of the forward and backward hidden states).
- $\boldsymbol{W}_s \in \mathbb{R}^{d_a \times d_2}$: Learnable weight matrix that maps decoder hidden state to alignment (attention) dimension. In this case the alignment (attention) dimension is d_a.
- $\boldsymbol{W}_h \in \mathbb{R}^{d_a \times d_1}$: Learnable weight matrix that maps encoder hidden state to alignment (attention) dimension.
- $\boldsymbol{v}_a \in \mathbb{R}^{d_a}$: Learnable vector, also called alignment vector of dimension d_a.

Bahdanau attention generally, uses the **previous decoder hidden state $\boldsymbol{s}_{t-1}$** to attend to the encoder hidden state $\boldsymbol{h}_i$

Different types of Attention

Luong (multiplicative) Attention

Luong attention, often called **Multiplicative Attention**, simplifies the scoring by using direct matrix multiplication, which is often faster in practice. It typically uses the **current decoder hidden state** $\boldsymbol{s}_t$. Luong proposed several scoring functions, all involving some form of multiplication between $\boldsymbol{s}_t$ and $\boldsymbol{h}_i$.

- **The Dot Product:** This is the simplest, but requires $\boldsymbol{s}_t$ and $\boldsymbol{h}_i$ to be in the same dimension.

$$e_i^{(t)} = \boldsymbol{s}_t^T \boldsymbol{h}_i$$

- **General:** It introduces a learnable weight matrix $\boldsymbol{W}_a$.

$$e_i^{(t)} = \boldsymbol{s}_t^T \boldsymbol{W}_a \boldsymbol{h}_i$$

- **Concat:** Similar to Bahdanau's additive approach, but Luong categorizes this as an alternative scoring function, and uses $\boldsymbol{s}_t$ instead of $\boldsymbol{s}_{t-1}$.

$$e_i^{(t)} = \boldsymbol{v}_a^T \tanh(\boldsymbol{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$$

There are other types of attentions as well. Which we will cover later in the course.

Evaluating Seq2Seq Model

Evaluating a Sequence-to-Sequence (Seq2Seq) model, which is used in tasks like machine translation, text summarization, and dialogue generation, primarily involves comparing the machine-generated sequence (candidate) against one or more human-created sequences (references). The most common metrics for this are **BLEU** and **ROUGE.**

BLEU (**B**i**L**ingual **E**valuation **U**nderstudy)

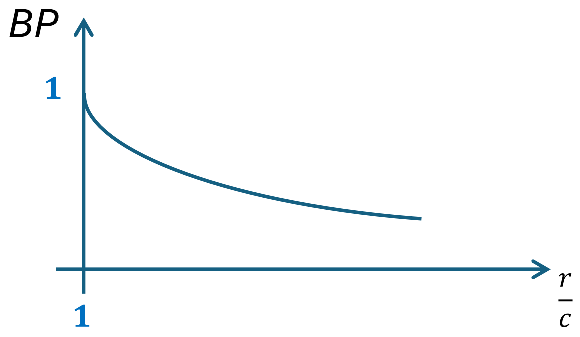
Definition: The BLEU score is a **precision-focused** metric that measures the similarity between a *candidate text* and a *set of reference texts*. It works by counting the number of matching **n-grams** (sequences of n words) between the candidate and the reference, with a penalty for overly short translations (**Brevity Penalty**).
Range: 0 to 1. A score close to 1 is better.

$$BLEU - N = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) = BP \,.\prod_{n=1}^{N} p_n^{w_n}$$

- p_n : precision for n-grams (1-gram, 2-gram, ...), N denotes **the highest n-gram order** used in computing the BLEU score.
- w_n : weight for n-gram (often uniform, e.g. $\frac{1}{N}$)
- BP : brevity penalty (prevents too-short outputs)

$$BP = \begin{cases} 1, & c > r \\ e^{1-\frac{r}{c}}, & c \le r \end{cases}$$

Where, c = candidate length and r = reference length.



Evaluating Seq2Seq Model

How to compute BLEU score?

Suppose, the candidate and the reference sentence looks like following.

Candidate sentence (model output)	**Reference sentence**
"She likes to read books on AI"	"She enjoys reading books about AI"

Step-1: Tokenize

Candidate: [She, likes, to, read, books, on, AI] Reference: [She, enjoys, reading, books, about, AI]

Step-2: Compute n-gram precisions : For BLEU-2 we'll compute 1-gram and 2-gram precisions.

Computing 1-gram precision (p_1)

Candidate 1-grams: She, likes, to, read, books, on, AI

Reference 1-grams: She, enjoys, reading, books, about, AI

Now we need to calculate count of each 1-gram:

$$clipped\ count = \min\left(Count\ in\ canidate\ , Maximum\ count\ in\ reference(s)\right)$$

Evaluating Seq2Seq Model

Candidate sentence (model output)

“She likes to read books on AI”

Reference sentence

“She enjoys reading books about AI”

Computing 1-gram precision (p_1)

Candidate 1-grams	Candidate Count	Reference Count	Clipped Count
She	1	1	1
likes	1	0	0
to	1	0	0
read	1	0	0
books	1	1	1
on	1	0	0
AI	1	1	1

Clipped matches = 3 , total candidate 1-grams = 7

$$p_1 = \frac{3}{7} = 0.429$$

Evaluating Seq2Seq Model

Candidate sentence (model output)

"She likes to read books on AI"

Reference sentence

"She enjoys reading books about AI"

Computing 2-gram precision (p_2)

Candidate 2-grams: She likes, likes to, to read, read books, books on, on AI

Reference 2-grams: She enjoys, enjoys reading, reading books, books about, about AI

Candidate 2-grams	Candidate Count	Reference Count	Clipped Count
She likes	1	0	0
likes to	1	0	0
to read	1	0	0
read books	1	0	0
books on	1	0	0
on AI	1	0	0

Clipped matches = 0 , total candidate 1-grams = 6

$p_2 = \frac{0}{6} = 0 \rightarrow$ this is problematic because we can't take log of zero. Hence, we add a small value ϵ (say 0.01) when the p_n calculates to be zero. With this smoothing, $p_2 = 0.01$

Evaluating Seq2Seq Model

Candidate sentence (model output)	**Reference sentence**
“She likes to read books on AI”	“She enjoys reading books about AI”

Step-3: Compute Brevity Penalty (BP).

$$BP = \begin{cases} 1, & c > r \\ e^{1-\frac{r}{c}}, & c \le r \end{cases}$$

Where, c = candidate length and r = reference length.

Candidate length (c) = 7, reference length (r) = 6 ⇒ $c > r \Rightarrow BP = 1$

Step-4: Calculate the score.

$$BLEU - 2 = BP.\exp\left(\frac{1}{2}\left(\log p_1 + \log p_2\right)\right)$$
$$= 1 \times \exp\left(\frac{1}{2}\left(\log\ 0.429 + \log 0.01\right)\right)$$
$$= \exp\left(\frac{1}{2}\left(-0.847\ - 4.605\right)\right) = \exp(-2.726) \approx 0.065$$

Hence,$BLEU - 2 \approx 6.5\%$

Evaluating Seq2Seq Model

- BLEU-1 → uses only unigram precision, it captures whether the model used the correct words or not.
- BLEU-2 → uses unigram + bigram overlap, it captures whether the adjacent words sound natural.
- BLEU-3 → uses unigram + bigram + trigram overlap, it captures whether the short phrases are correct.
- BLEU-4 → uses overlap up to 4-grams, it captures whether the sentence is grammatically smooth.

Machine translation papers & benchmarks almost always report **BLEU-4** (e.g., "Our model achieves BLEU = 32.5% on WMT14") → In such cases, "BLEU" **implicitly means BLEU-4**.

Drawbacks of BLEU Score

1. **BLEU is lexical, not semantic** : it measures literal overlap, not meaning.
   In the previous example, though the candidate sentence is pretty close to the reference sentence by meaning, it produces a very low BLEU score because of different word forms ('reading' vs 'read') , different word order ('books about AI' vs 'books on AI').
   Another example: "the cat is on the mat" and "the feline sits on the rug" has exactly same meaning but the BLEU score will be very low between them.

2. Hence, BLEU produces low score for sentence level evaluations.

3. Highly precision-focused; may reward short, generic, but accurate phrases that miss most of the target meaning.

Evaluating Seq2Seq Model

ROUGE (**R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation)

Definition: ROUGE is a set of **recall-focused** metrics, primarily used for evaluating **text summarization**. It measures the overlap of n-grams or the *longest common subsequence (LCS)* between the candidate and the reference text.

Key Variants:

- **ROUGE-N:** Measures the overlap of N-gram tokens.
  - **ROUGE-1:** Unigram (single word) overlap.
  - **ROUGE-2:** Bigram (two-word) overlap.
- **ROUGE-L:** Measures the longest common subsequence (LCS) between the candidate and reference. This captures *sequence order* without requiring consecutive matches. This is the most common one.

Range: 0 to 1. A score close to 1 is better.

$$ROUGE - N\ (Recall) = \frac{Count\ of\ matching\ n - grams}{Total\ number\ of\ n - grams\ in\ the\ Reference}$$

$$ROUGE - L\ (F_\beta) = \frac{(1+\beta^2) \times P \times R}{R + \beta^2\ P}$$

Where, $P = \frac{Length\ of\ LCS\ between\ the\ candidate\ and\ reference}{Candidate\ Length}$; $R = \frac{Length\ of\ LCS\ between\ the\ candidate\ and\ reference}{Reference\ Length}$

β is often set to 1 which leads to standard F_1 score.

Evaluating Seq2Seq Model

Longest Common Subsequence (LCS)

LCS is the longest sequence of elements (such as characters or words) that appear in the same relative order in two or more given sequences, but **not necessarily in consecutive positions**.
Example:

Candidate (C): "The quick brown animal jumped over the lazy black dog", Length of candidate (L_c): 10
Reference (R): "The quick brown fox jumps over the lazy dog", Length of reference (L_R): 9

The **Longest Common Subsequence (LCS)** here is: "The quick brown over the lazy dog". Length of LCS: $L_{LCS} = 7$. The words must appear in both sequences in the correct order, but not necessarily adjacent to each other.

Using this example:

1. Precision (P): $\frac{L_{LCS}}{L_c} = \frac{7}{10} = 0.7$
2. Recall (R): $\frac{L_{LCS}}{L_R} = \frac{7}{9} \approx 0.7778$
3. ROUGE-L $(\beta = 1)$: $\frac{2\,P\,R}{P+R} = \frac{2\times0.7\times0.7778}{0.7+0.7778} \approx 0.7368$

ROUGE is also lexical (word-based), not semantic

In Practice
For evaluating Seq2Seq:
- **Translation → BLEU, METEOR, TER**
- **Summarization → ROUGE-L**
- **Dialogue Generation → BLEU, BERTScore**
- **Language Modelling → Perplexity**

Thank You