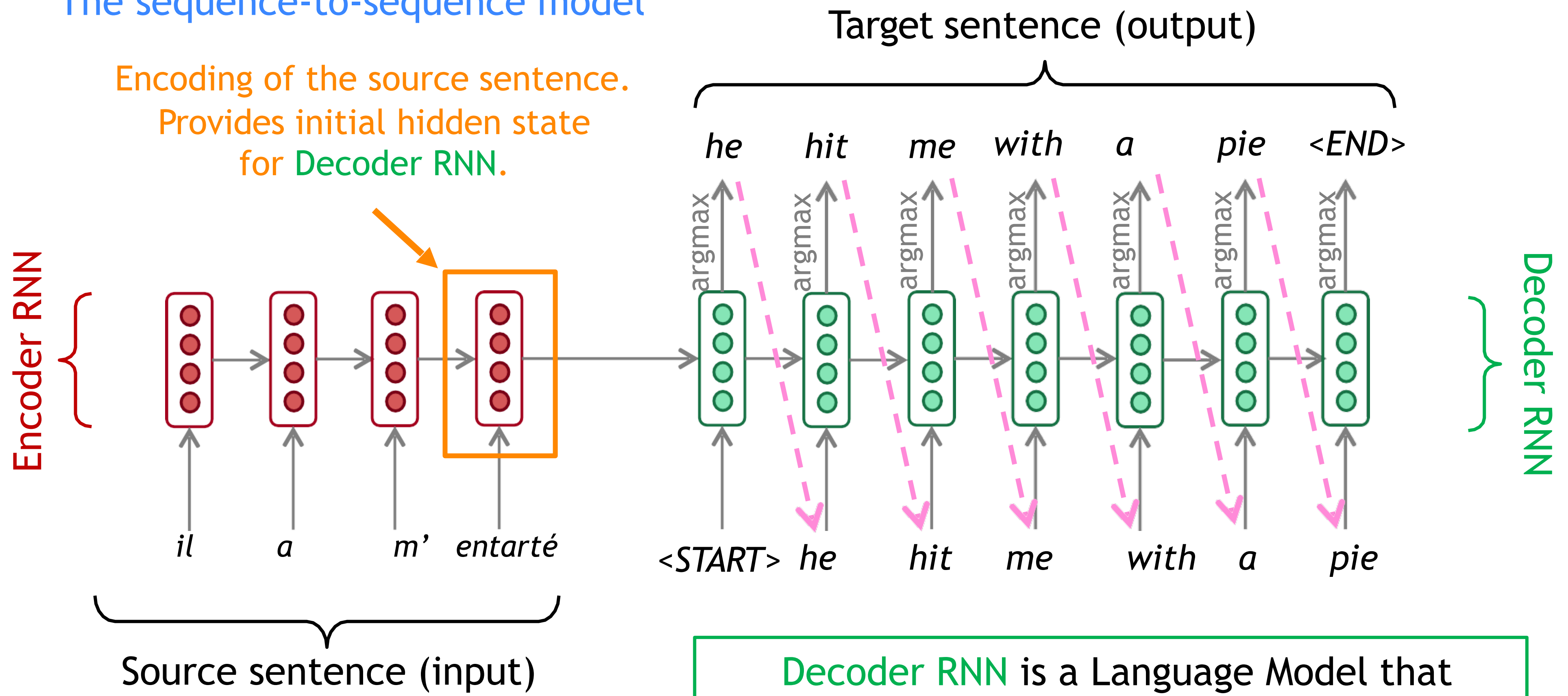
Attention and Transformers in NLP

Nishtha Madaan
Research Scientist
IBM Research AI

Neural Machine Translation (NMT)

The sequence-to-sequence model

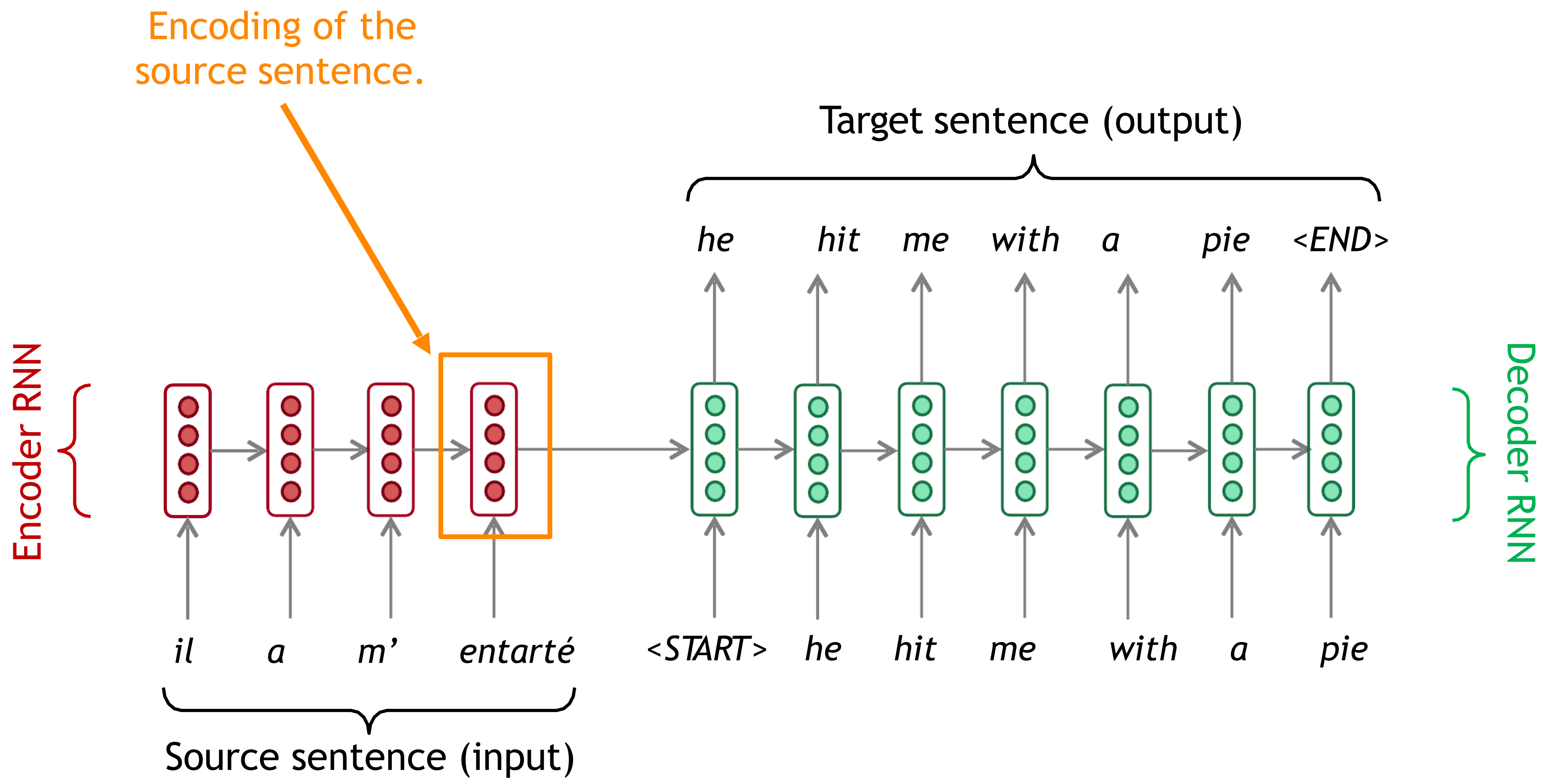


Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

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

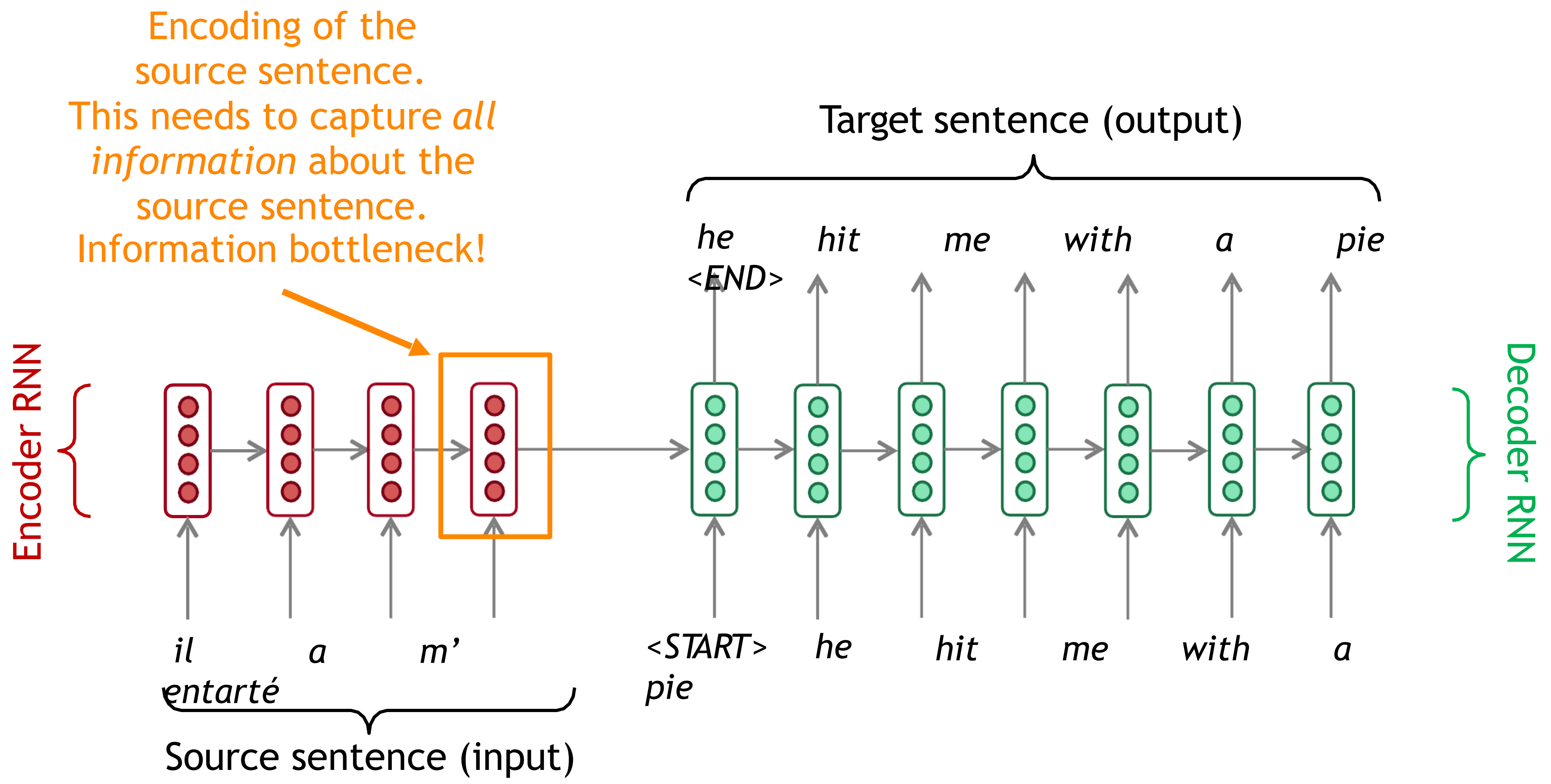
Encoder RNN produces an encoding of the source sentence.

Sequence-to-sequence: the bottleneck problem


Encoding of the source sentence.
Target sentence (output)
he hit me with a pie <END>
Encoder RNN
Decoder RNN
il a m' entarté
<START> he hit me with a pie
Source sentence (input)

Problems with this architecture?

Sequence-to-sequence: the bottleneck problem


Encoding of the source sentence.
This needs to capture *all information* about the source sentence.
Information bottleneck!
Target sentence (output)
he hit me with a pie <END>
Encoder RNN
Decoder RNN
il a m' entarté
<START> he hit me with a pie
Source sentence (input)

Reference: CS224 Abigail See, Matthew Lamm

Attention

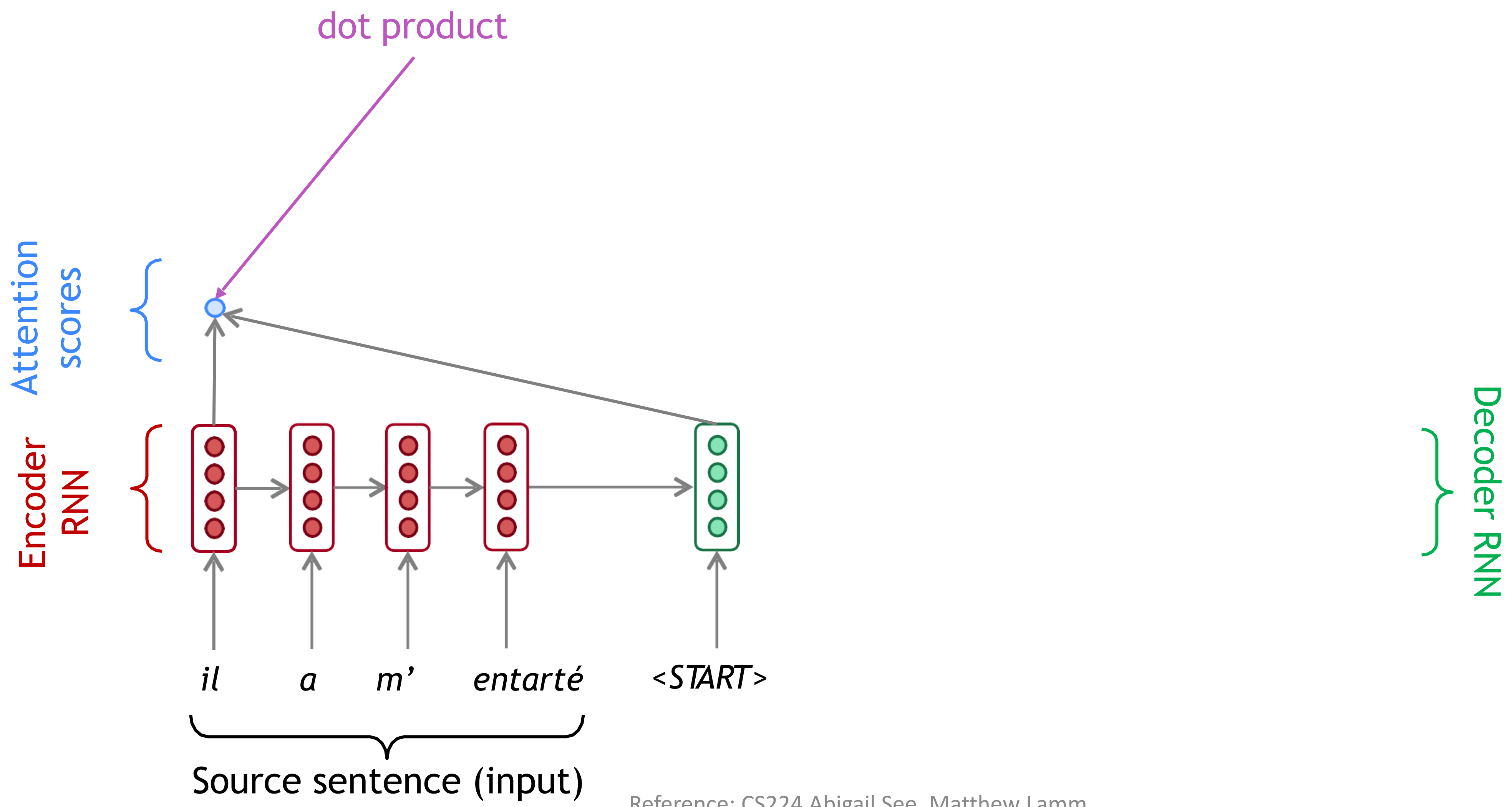
- **Attention** provides a solution to the bottleneck problem.

- 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

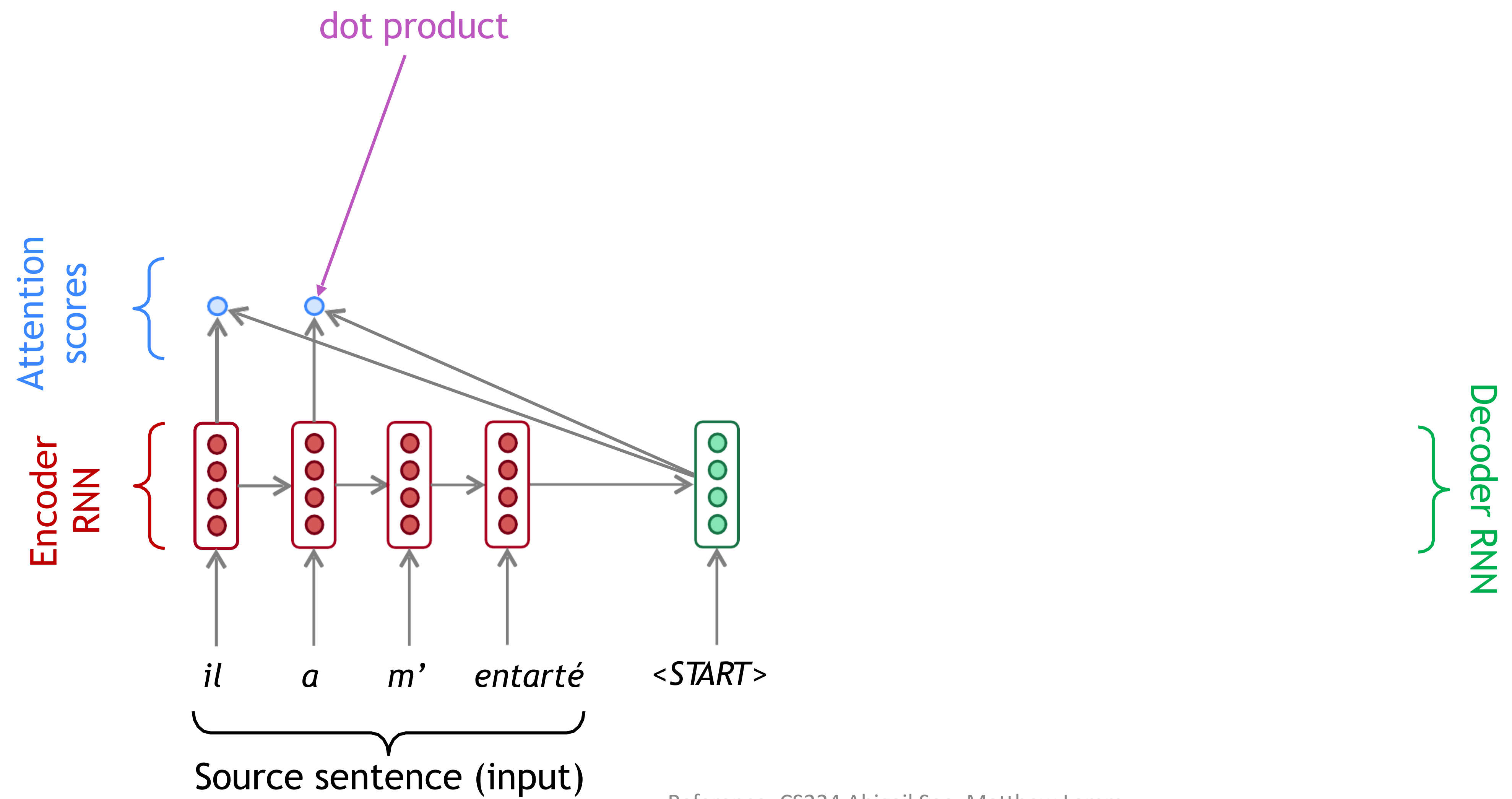
Reference: CS224 Abigail See, Matthew Lamm

Sequence-to-sequence with attention

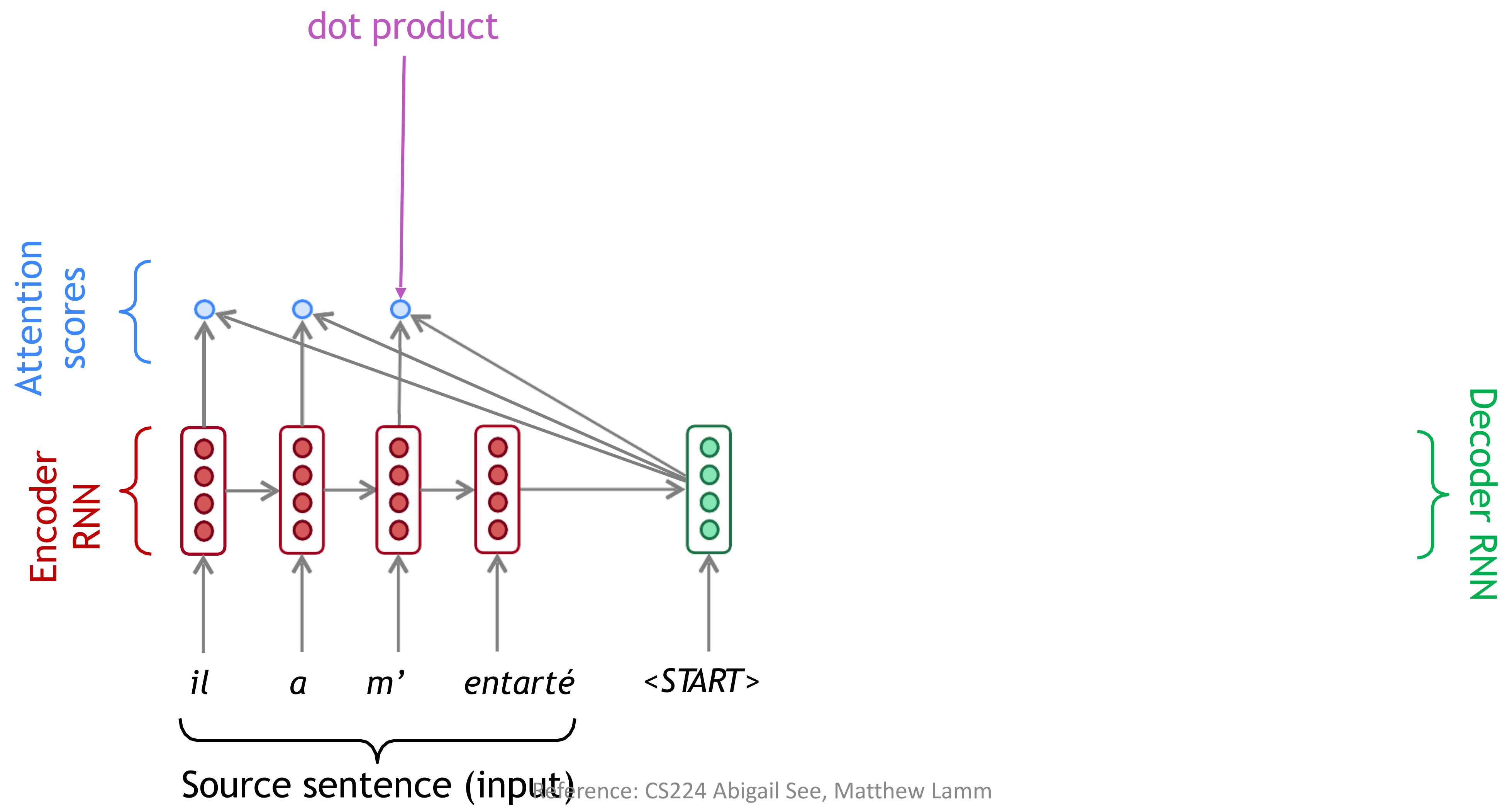


Reference: CS224 Abigail See, Matthew Lamm

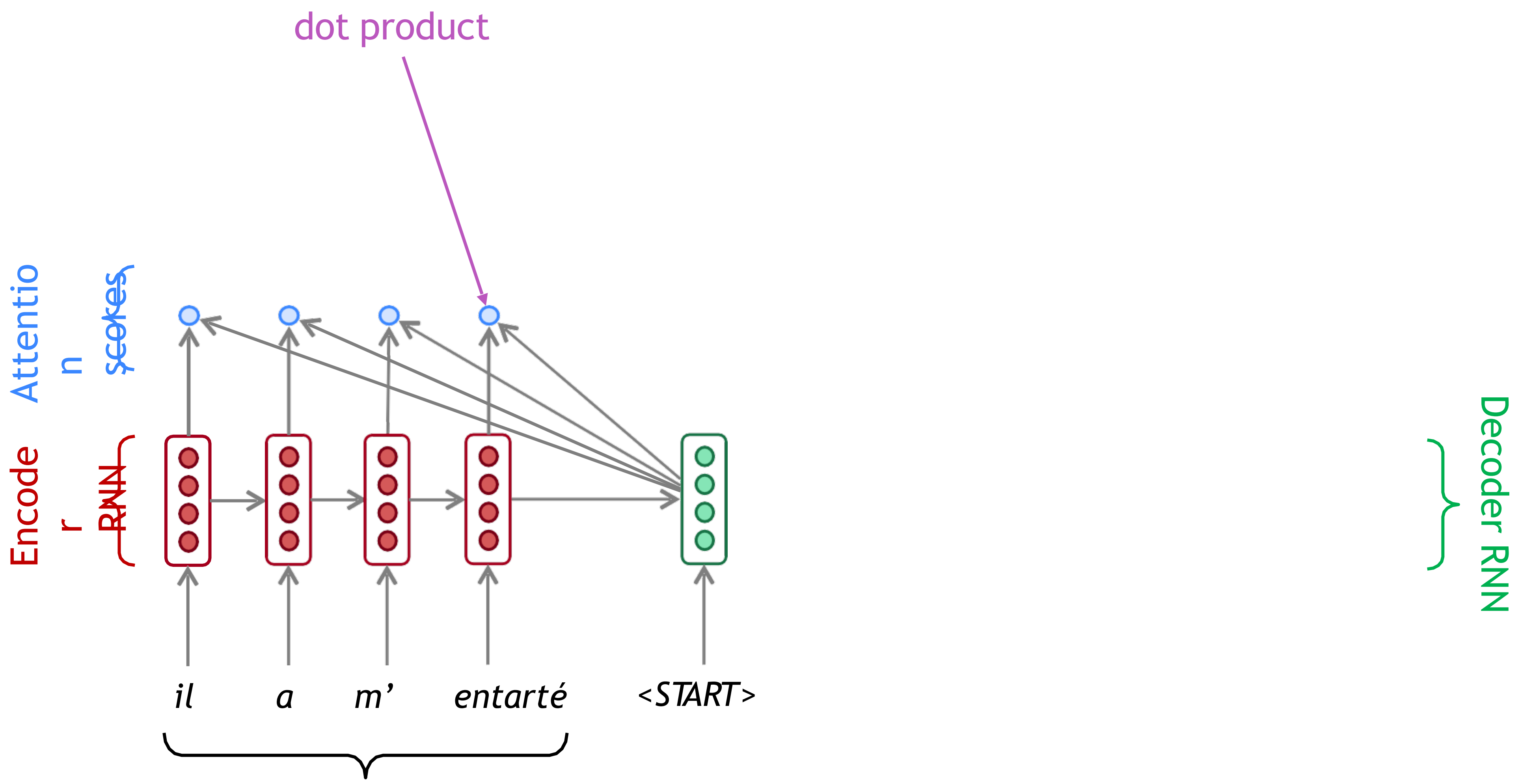
Sequence-to-sequence with attention



Sequence-to-sequence with attention

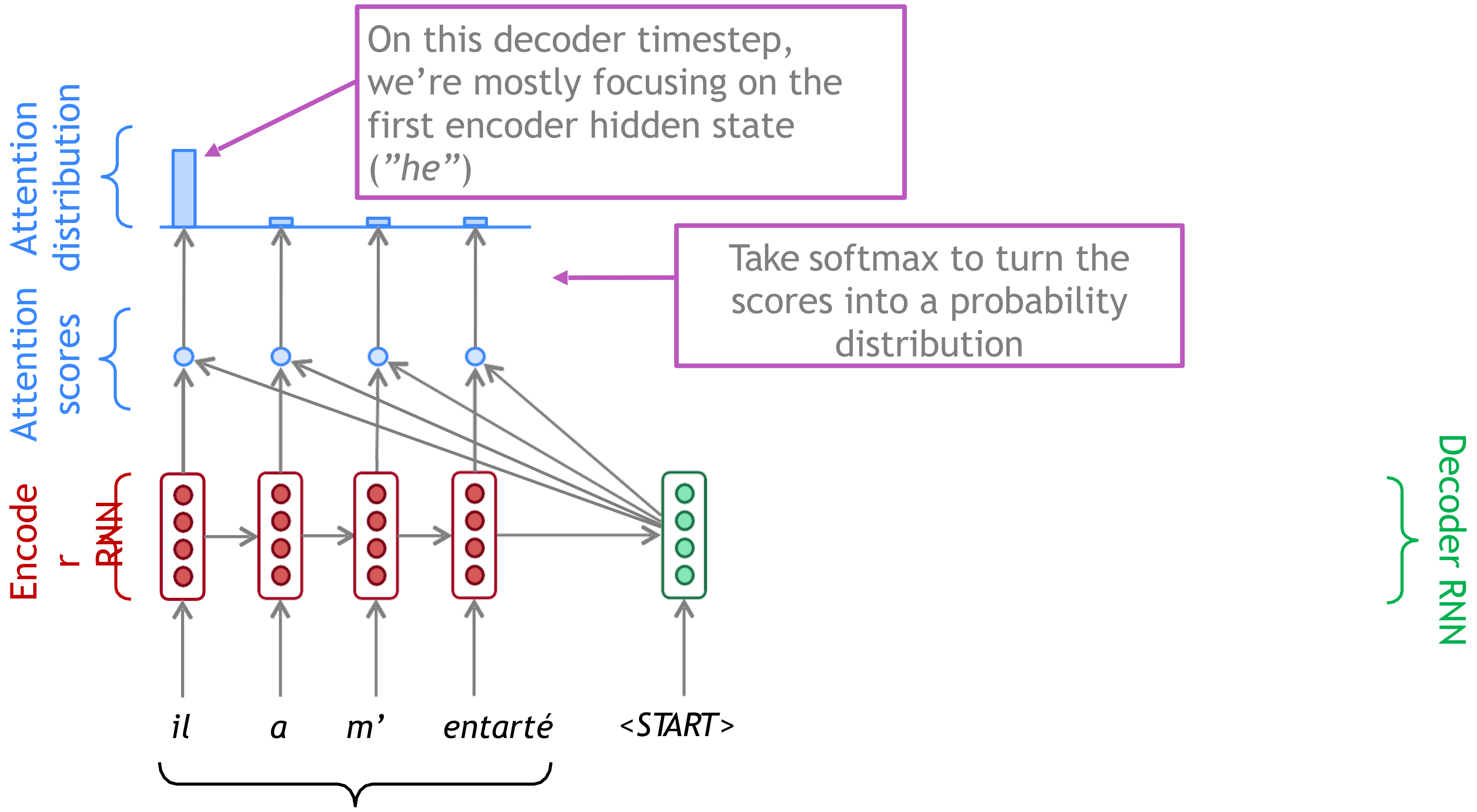


Sequence-to-sequence with attention



Reference: CS224 Abigail See, Matthew Lamm

Sequence-to-sequence with attention

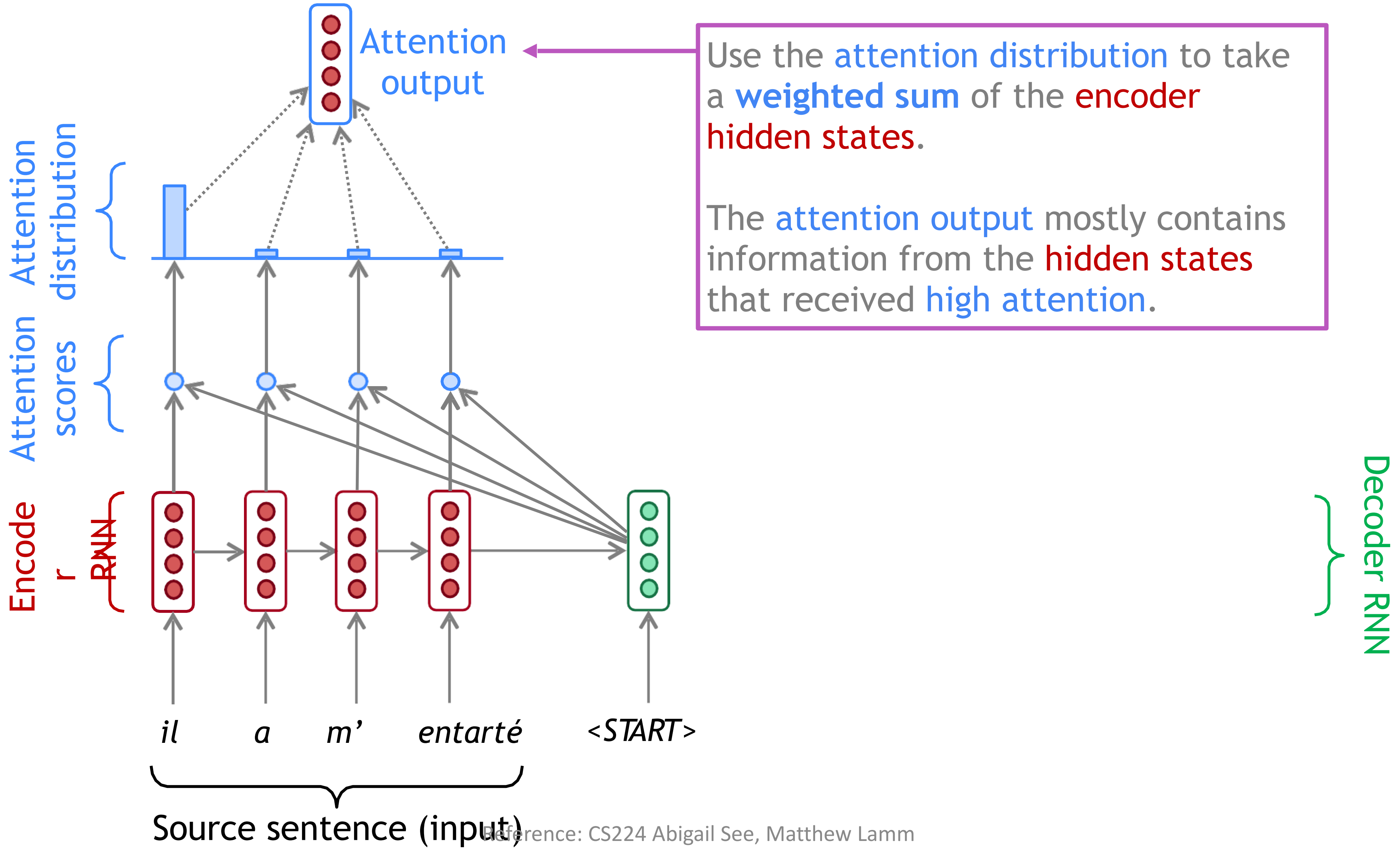


Attention distribution
Attention scores
Encoder RNN
Decoder RNN
On this decoder timestep, we're mostly focusing on the first encoder hidden state (*"he"*)
Take softmax to turn the scores into a probability distribution
il
a
m'
entarté
<START>

Source sentence (input)

Reference: CS224 Abigail See, Matthew Lamm

Sequence-to-sequence with attention

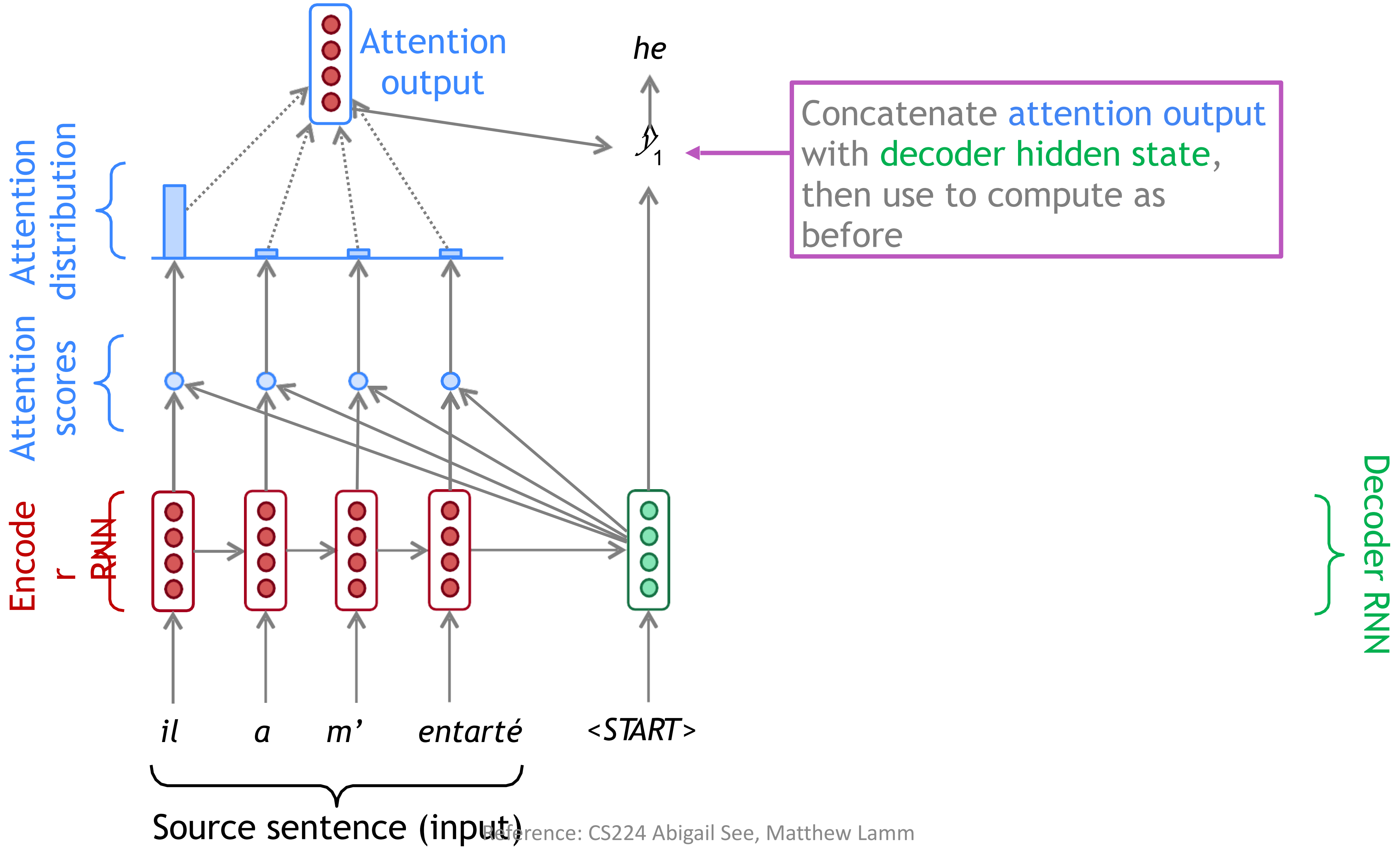


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

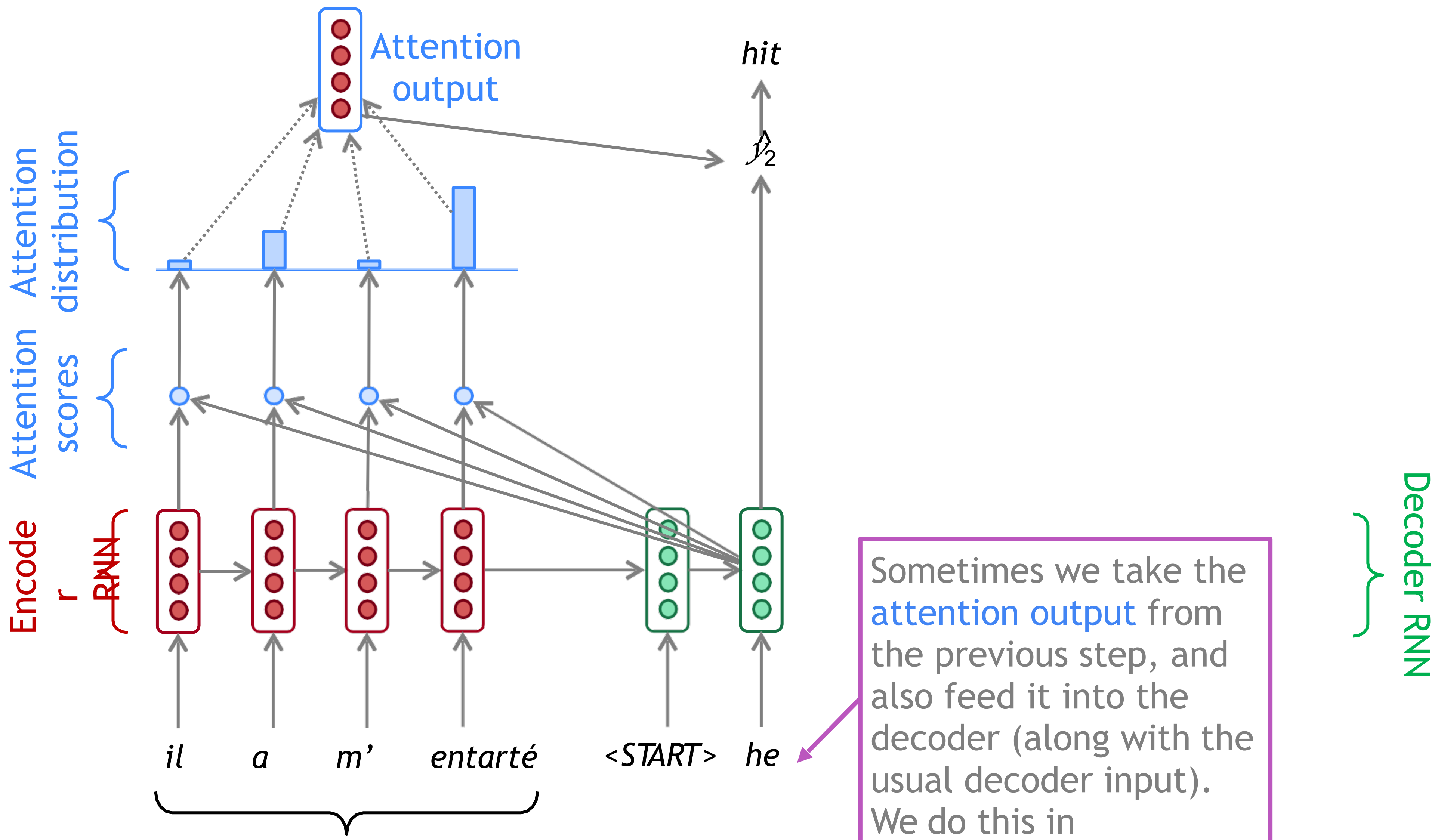
Reference: CS224 Abigail See, Matthew Lamm

Sequence-to-sequence with attention



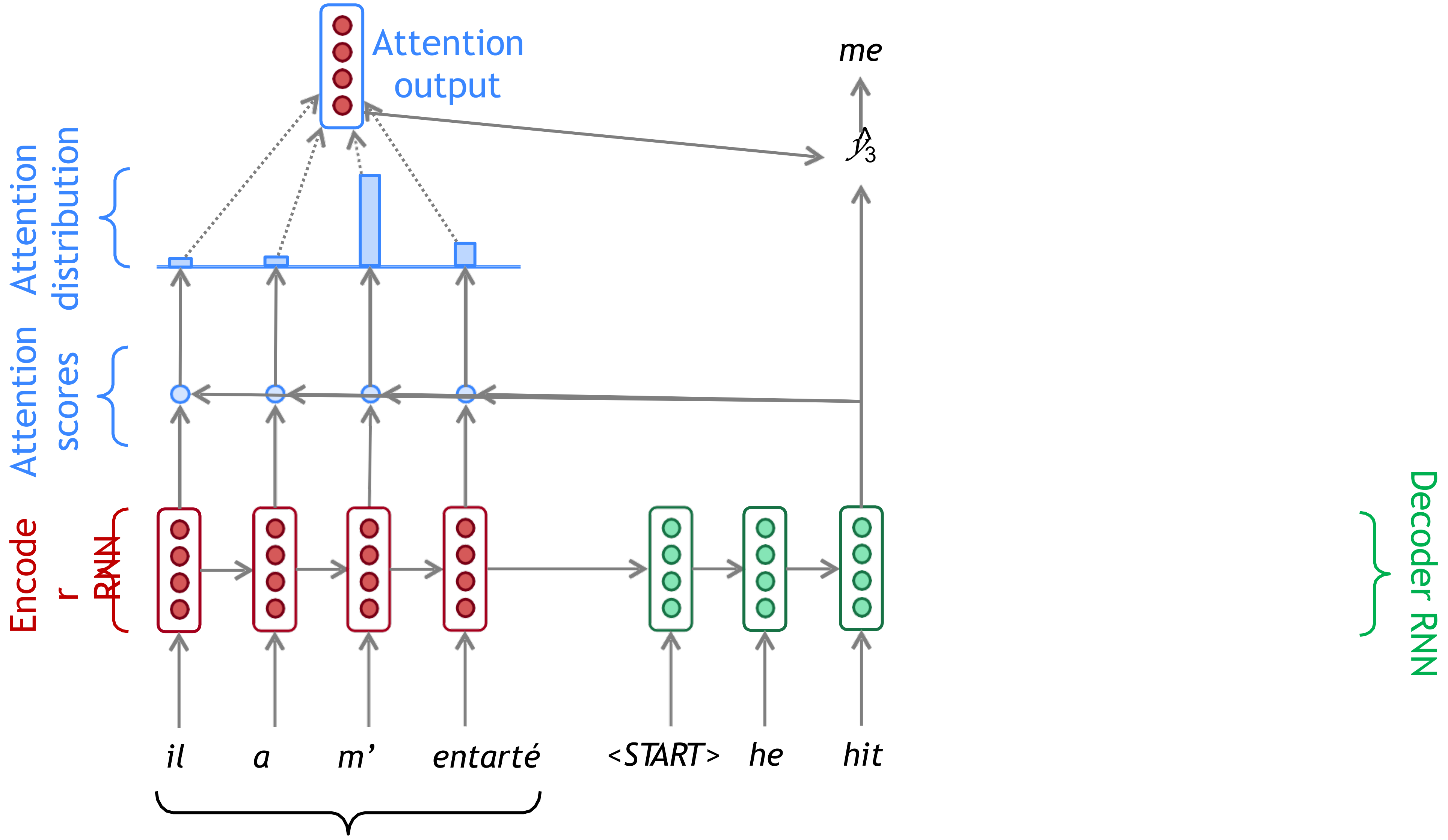
Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

Sequence-to-sequence with attention



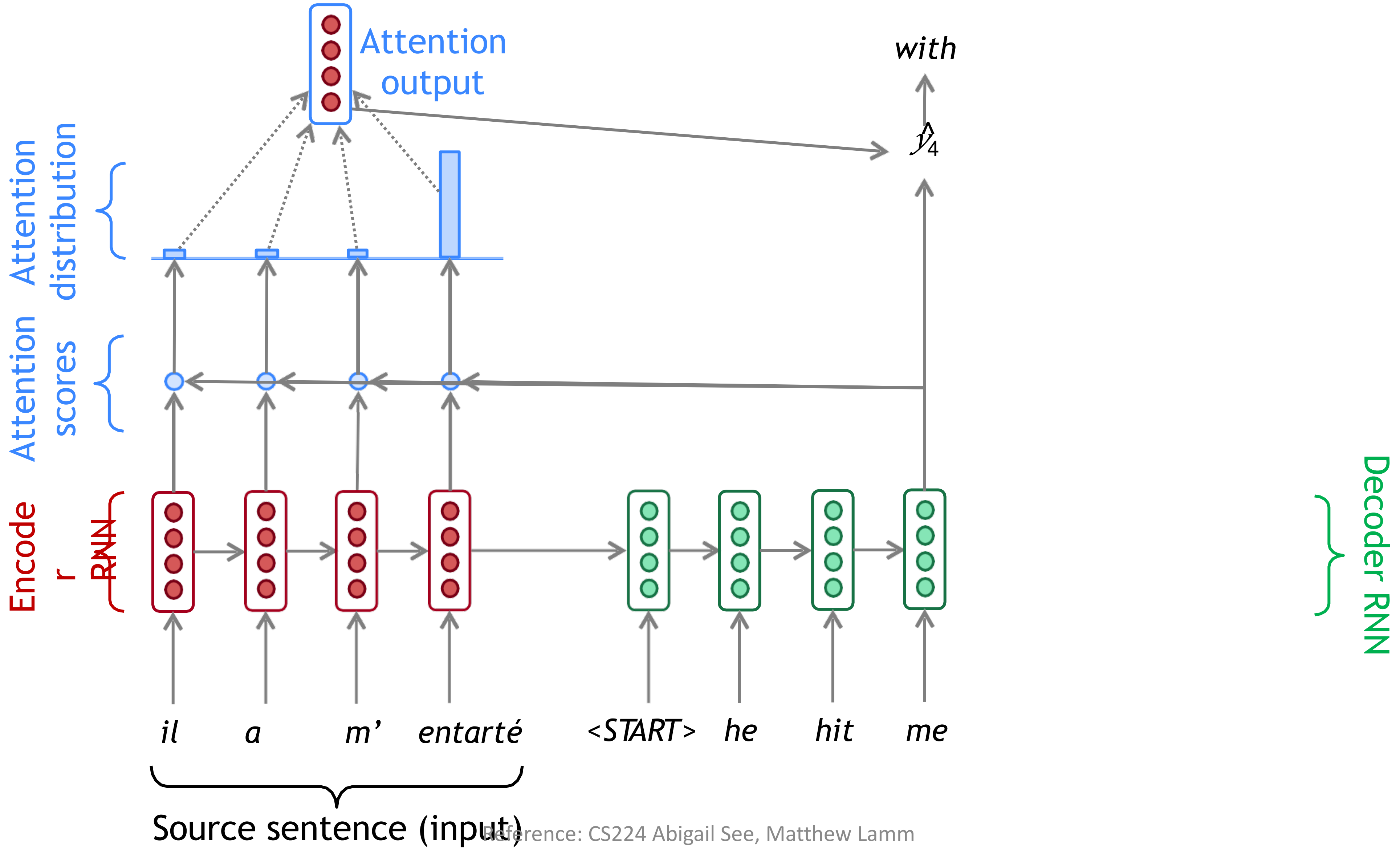
Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input). We do this in Assignment 4.

Sequence-to-sequence with attention



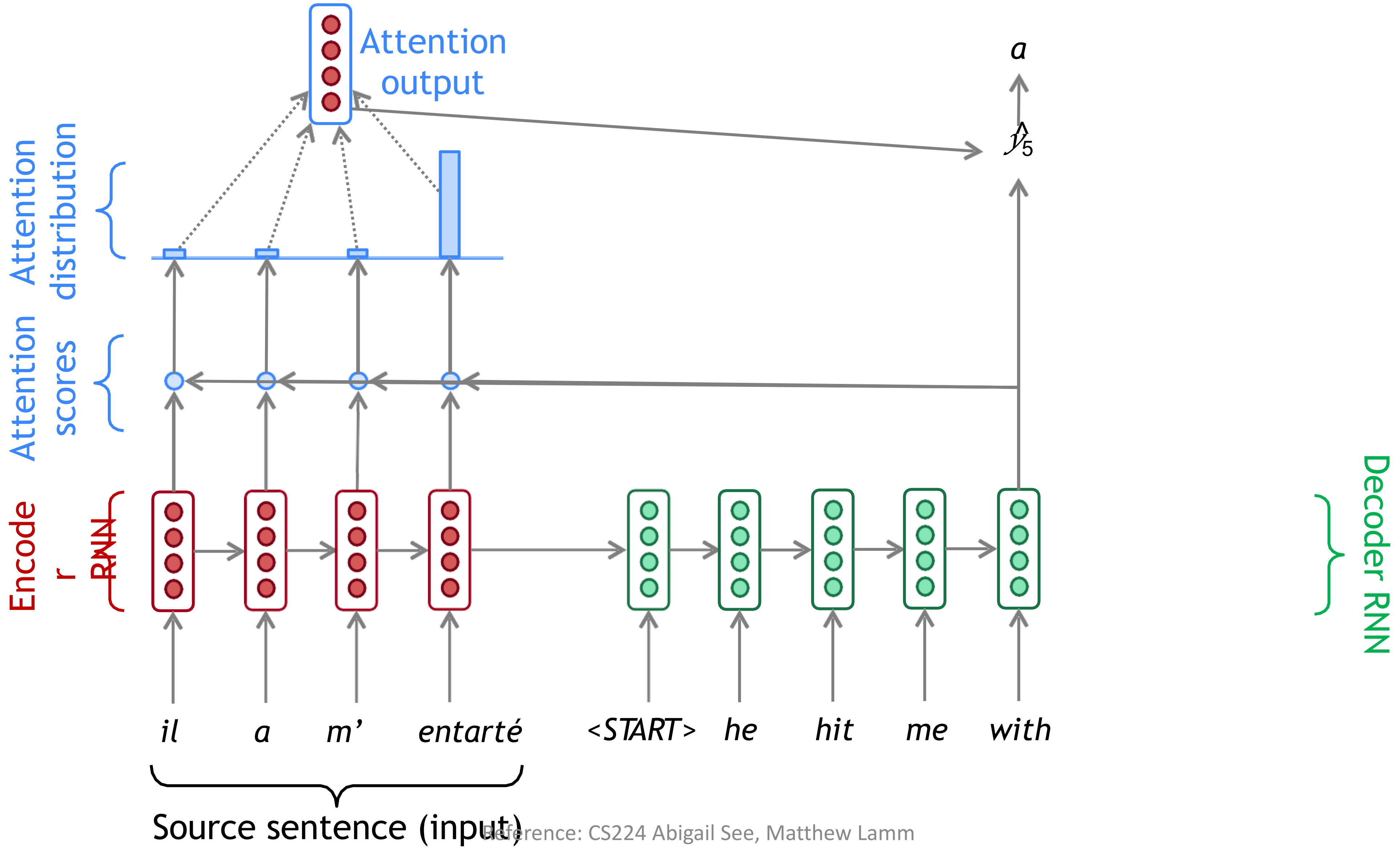
Reference: CS224 Abigail See, Matthew Lamm

Sequence-to-sequence with attention



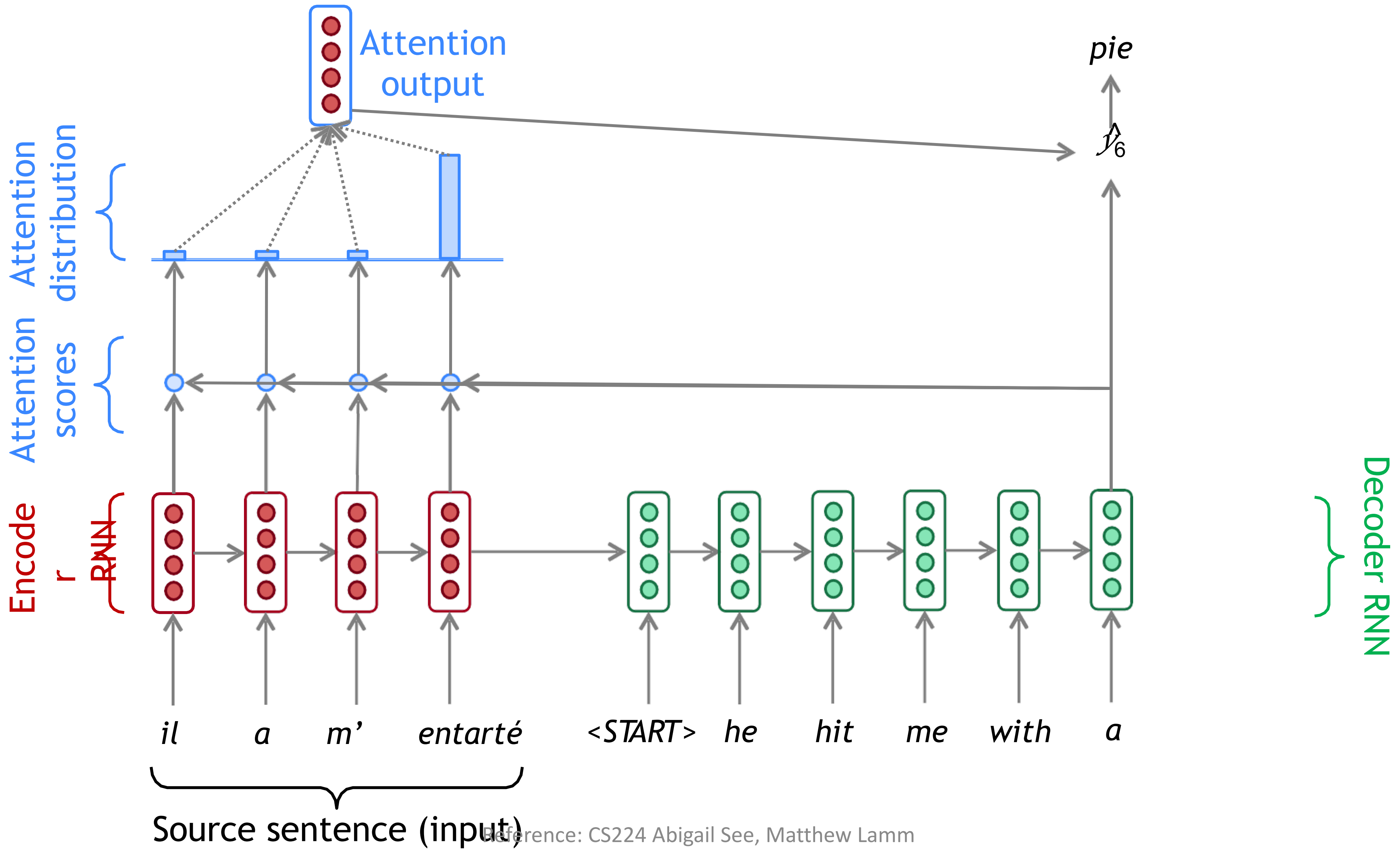
Reference: CS224 Abigail See, Matthew Lamm

Sequence-to-sequence with attention



Reference: CS224 Abigail See, Matthew Lamm

Sequence-to-sequence with attention



Reference: CS224 Abigail See, Matthew Lamm

Attention: in equations

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

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

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

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

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

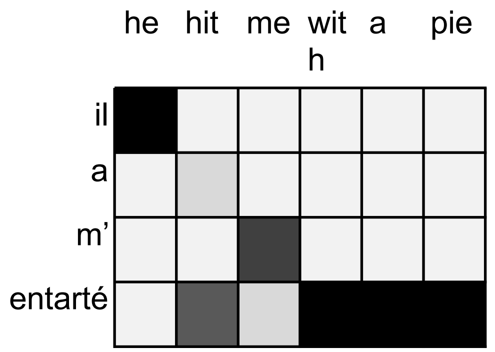
$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

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

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

Attention is great

- Attention significantly improves NMT performance
  - It’s very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself



Attention is a ***general*** Deep Learning technique

- We'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).

Reference: CS224 Abigail See, Matthew Lamm

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).

There are ***several*** attention variants

- We have some *values* $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a $\boldsymbol{s} \in \mathbb{R}^{d_2}$

- Attention always involves:
  1. Computing the *attention scores* $\boldsymbol{e} \in \mathbb{R}^N$ ← There are multiple ways to do this
  2. Taking softmax to get *attention distribution* α:

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

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

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

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

Attention variants

$\boldsymbol{e} \in \mathbb{R}^N$ $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$

$\boldsymbol{s} \in \mathbb{R}^{d_2}$

- There are several ways you can comp and :

$$\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$$

$d_1 = d_2$

- Basic dot-product attention:
  - Note: this assumes
  - This is the version $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$

$\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix

$$\boldsymbol{e}_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s}) \in \mathbb{R}$$

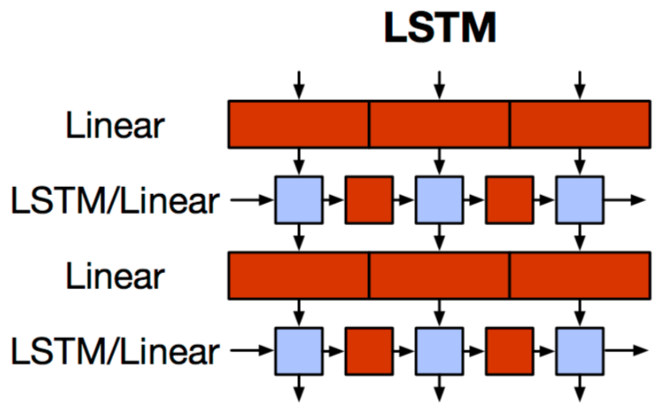
$\boldsymbol{W}_1 \in \mathbb{R}^{d_3 \times d_1}, \boldsymbol{W}_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices $\boldsymbol{v} \in \mathbb{R}^{d_3}$

- d_3 (the attention dimensionality) is a hyperparameter

More information:
"Deep Learning for NLP Best Practices", Ruder, 2017. http://ruder.io/deep-learning-nlp-best-practices/index.html#attention
Reference: CS224 Abigail See, Matthew Lamm
"Massive Exploration of Neural Machine Translation Architectures", Britz et al, 2017, https://arxiv.org/pdf/1703.03906.pdf

The Motivation for Transformers

- We want **parallelization** but RNNs are inherently sequential



- Despite LSTMs, RNNs generally need attention mechanism to deal with long range dependencies – **path length** between states grows with distance otherwise
- But if **attention** gives us access to any state… maybe we can just use attention and don't need the RNN? 🤔

- And then NLP can have deep models … and solve our vision envy

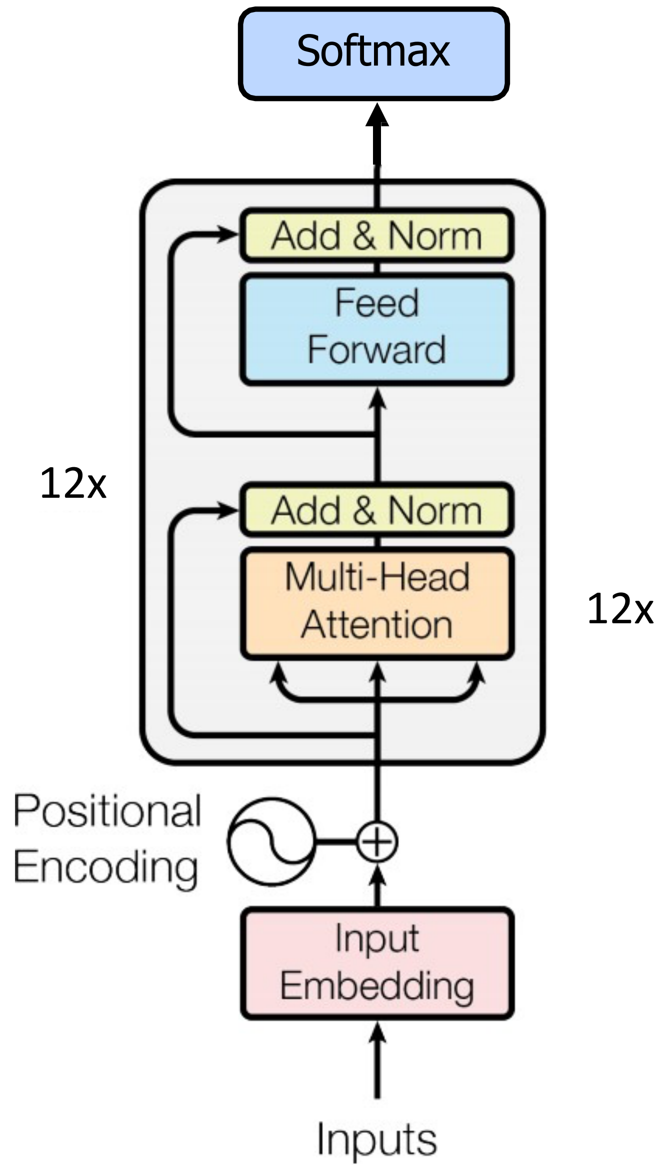
Reference: CS224 Abigail See, Matthew Lamm

Transformer (Vaswani et al. 2017) “Attention is all you need”

https://arxiv.org/pdf/1706.03762.pdf

- **Non-recurrent** sequence (or sequence-to-sequence) model
- A **deep** model with a sequence of **attention**-based transformer blocks
- Depth allows a certain amount of lateral information transfer in understanding sentences, in slightly unclear ways
- Final cost/error function is standard cross-entropy error on top of a softmax classifier

Initially built for NMT



Transformer block

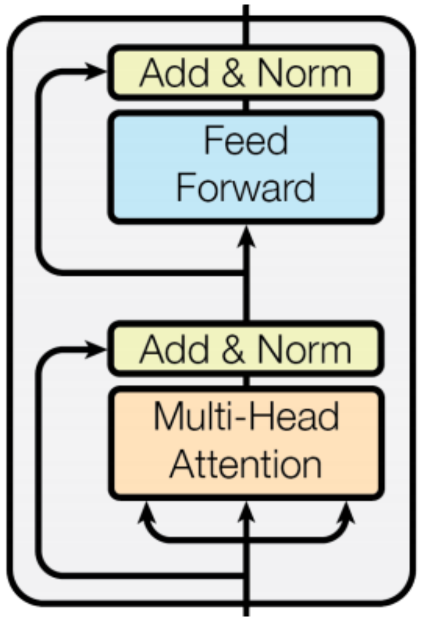
Each block has two "sublayers"

1. Multihead attention
2. 2-layer feed-forward NNet (with ReLU)

Each of these two steps also has:

Residual (short-circuit) connection

LayerNorm (scale to mean 0, var 1; Ba et al. 2016)



Multi-head (self) attention

With simple self-attention: Only one way for a word to interact with others

Solution: Multi-head attention

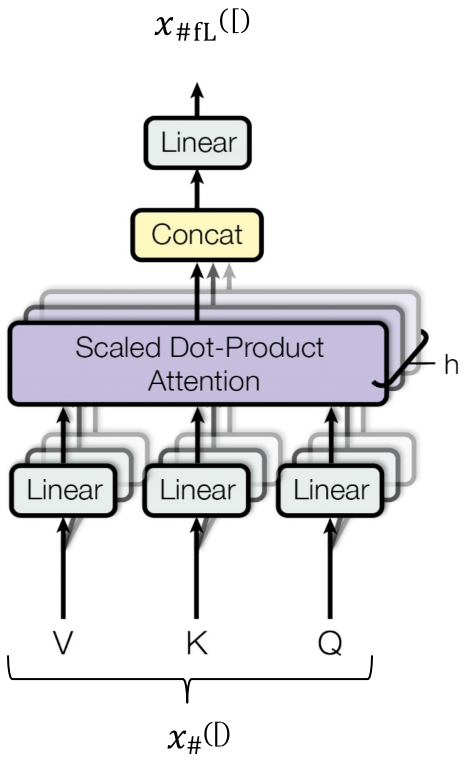
Map input into $h = 12$ many lower dimensional spaces via W_{V}matrices

Then apply attention, then concatenate outputs and pipe through linear layer

$$\text{Multihead}\left(x_{\#}^{([)}\right) = \text{Concat}(head_{'})W^{`}$$

$$head_{'} = \text{Attention}(x_{\#}^{([)}W^{'\mathrm{b}},\ x_{\#}^{([)}W^{'\mathrm{c}},\ x_{\#}^{([)}W^{'\mathrm{d}})$$

So attention is like bilinear: $x_{\#}^{([)}(W^{'\mathrm{b}}(W^{'\mathrm{c}})^{?})x_{\#}^{(\mathrm{e})}$



BERT: Devlin, Chang, Lee, Toutanova (2018)

BERT (Bidirectional Encoder Representations from Transformers):

Pre-training of Deep Bidirectional Transformers for Language Understanding, which is then fine-tuned for a particular task

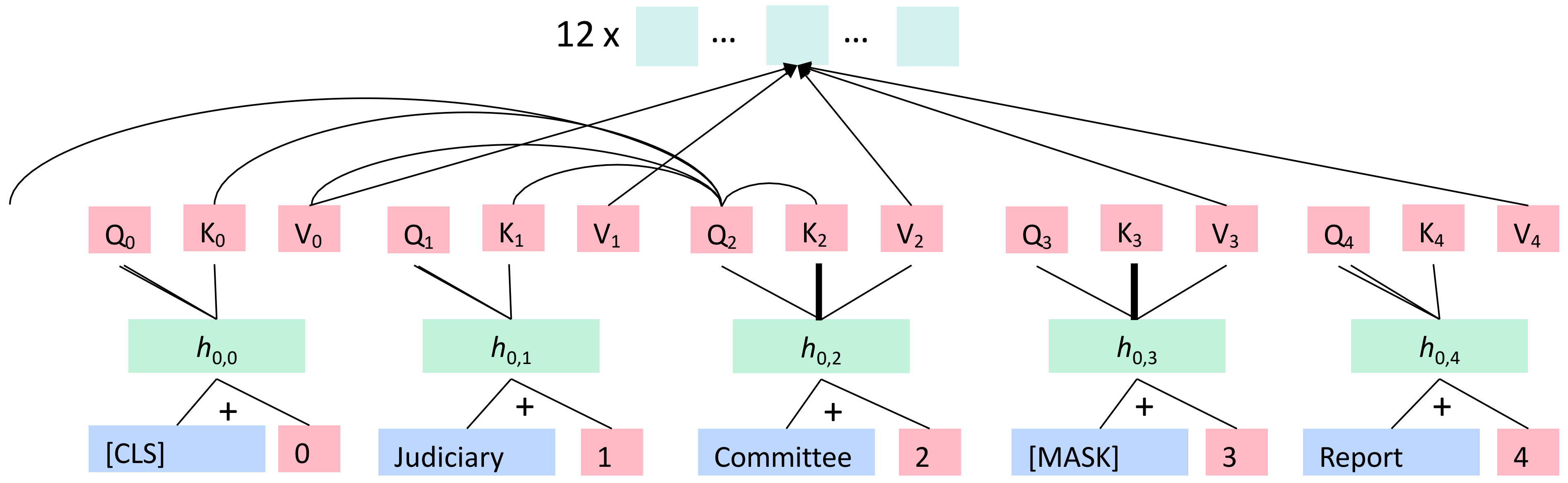
Pre-training uses a cloze task formulation where 15% of words are masked out and predicted:

$$\begin{tikzcd}[column sep=small] \text{the man went to the} & \text{[MASK]} \arrow[u] & \text{to buy a} & \text{[MASK]} \arrow[u] & \text{of milk} \end{tikzcd}$$

store ↑ gallon ↑

the man went to the [MASK] to buy a [MASK] of milk

Transformer (Vaswani et al. 2017)
BERT (Devlin et al. 2018)



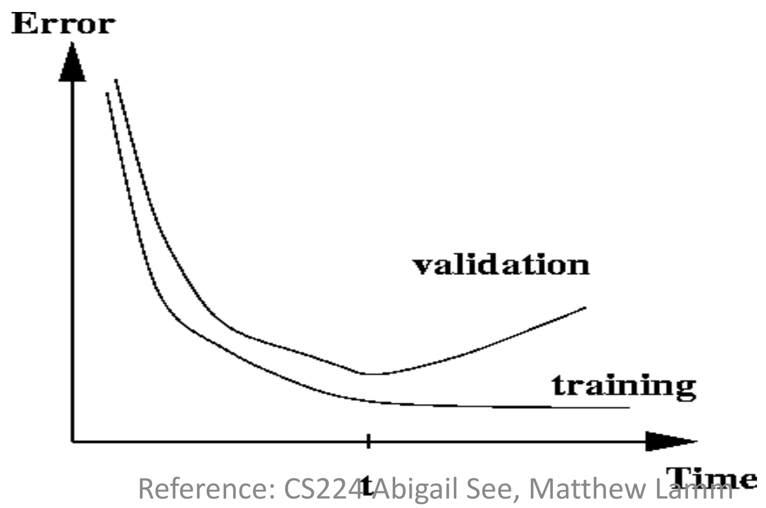
Reference: CS224 Abigail See, Matthew Lamm

Pots of data

- Many publicly available datasets are released with a **train/dev/test** structure. **We're all on the honor system to do test-set runs only when development is complete.**
- Splits like this presuppose a fairly large dataset.
- If there is no dev set or you want a separate tune set, then you create one by splitting the training data, though you have to weigh its size/usefulness against the reduction in train-set size.
- Having a fixed test set ensures that all systems are assessed against the same gold data. This is generally good, but it is problematic where the test set turns out to have unusual properties that distort progress on the task.

Training models and pots of data

- When training, models **overfit** to what you are training on
  - The model correctly describes what happened to occur in particular data you trained on, but the patterns are not general enough patterns to be likely to apply to new data
- The way to monitor and avoid problematic overfitting is using **independent** validation and test sets …



Reference: CS224 Abigail See, Matthew Lamm

Training models and pots of data

- You build (estimate/train) a model on a **training set**.
- Often, you then set further hyperparameters on another, independent set of data, the **tuning set**
  - The tuning set is the training set for the hyperparameters!
- You measure progress as you go on a **dev set** (development test set or validation set)
  - If you do that a lot you overfit to the dev set so it can be good to have a second dev set, the **dev2** set
- **Only at the end**, you evaluate and present final numbers on a **test set**
  - Use the final test set **extremely** few times … ideally only once

Training models and pots of data

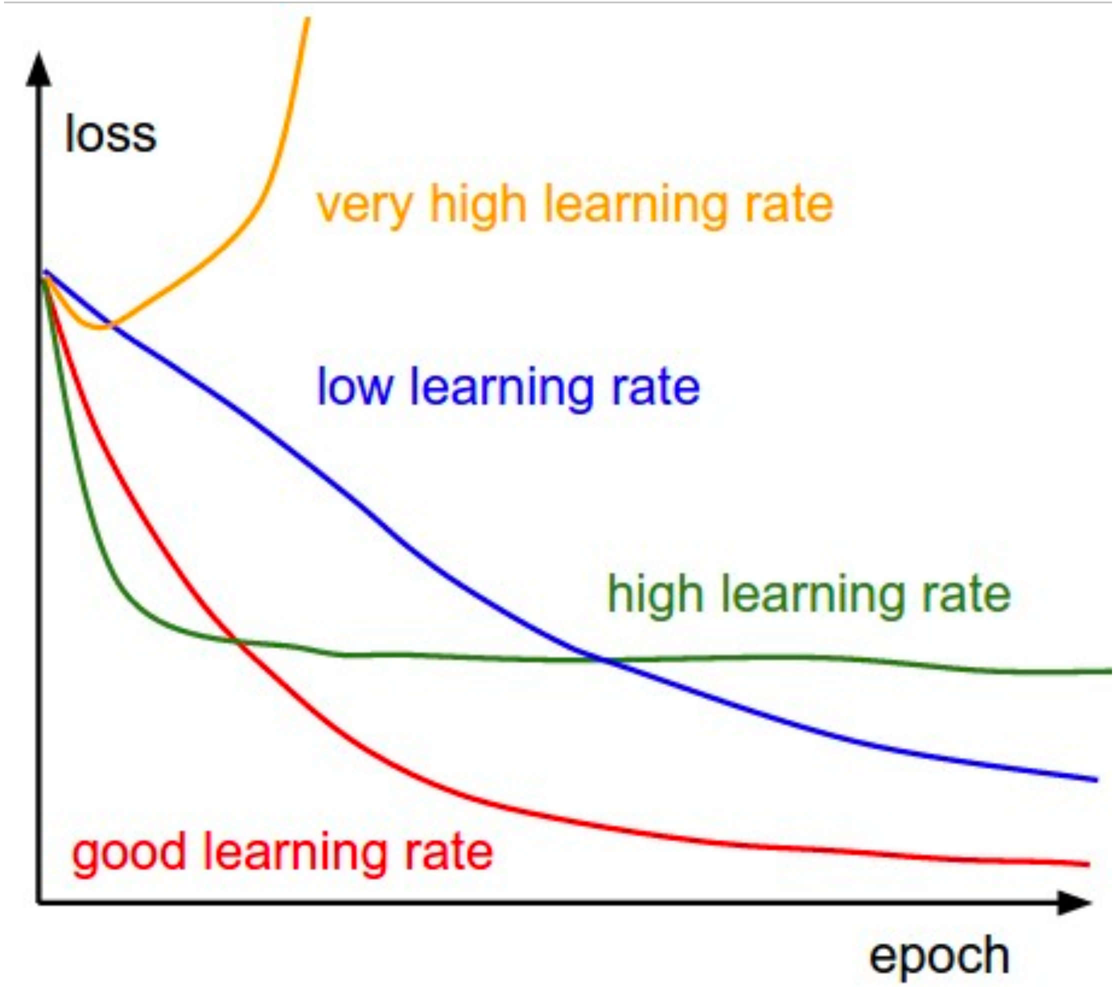
- The **train**, **tune**, **dev**, and **test** sets need to be completely distinct
- It is invalid to test on material you have trained on
  - You will get a falsely good performance. We usually overfit on train
- You need an independent tuning set
  - The hyperparameters won't be set right if tune is same as train
- If you keep running on the same evaluation set, you begin to overfit to that evaluation set
  - Effectively you are "training" on the evaluation set … you are learning things that do and don't work on that particular eval set and using the info
- To get a valid measure of system performance you need another untrained on, **independent** test set … hence dev2 and final test

Getting your neural network to train

- Start with a positive attitude!
    - **Neural networks want to learn!**
        - If the network isn’t learning, you’re doing something to prevent it from learning successfully
- Realize the grim reality:
    - **There are lots of things that can cause neural nets to not learn at all or to not learn very well**
        - Finding and fixing them (“debugging and tuning”) can often take more time than implementing your model
- It’s hard to work out what these things are
    - But experience, experimental care, and rules of thumb help!

Models are sensitive to learning rates

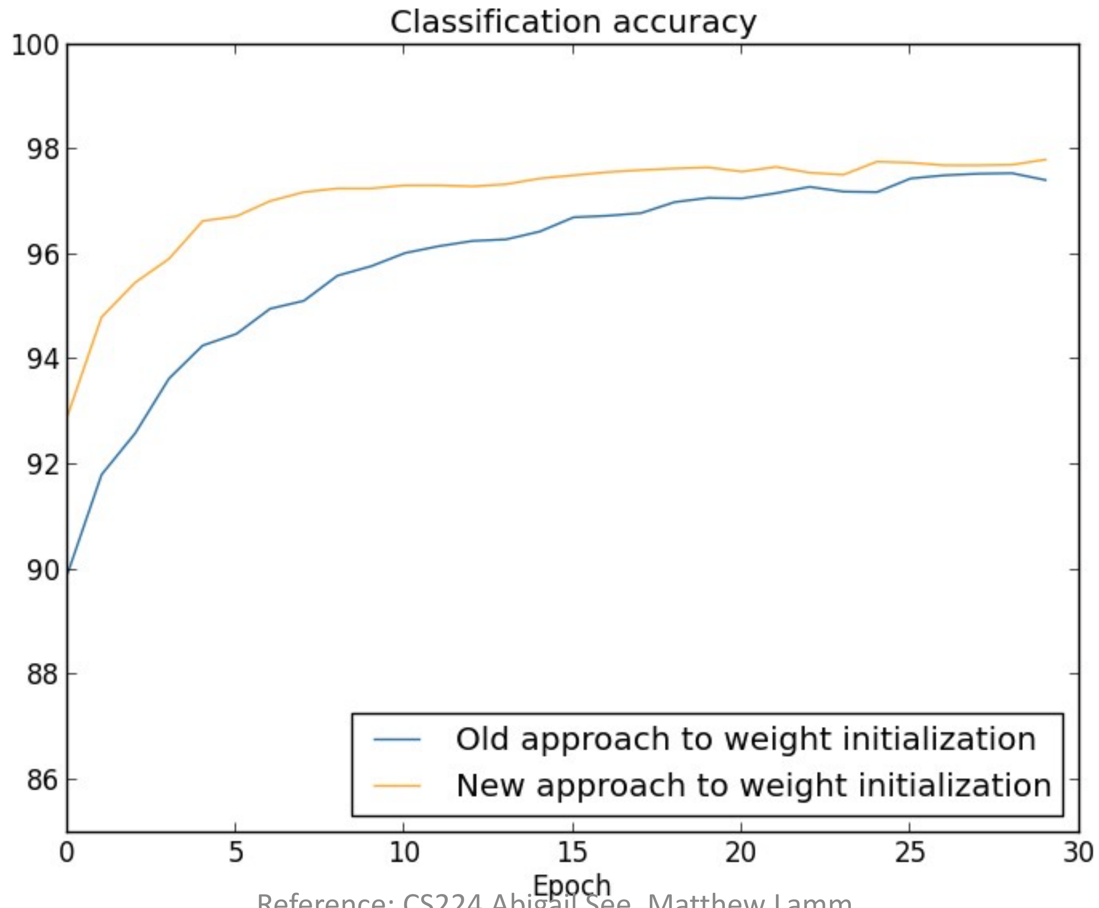
- From Andrej Karpathy, CS231n course notes



Reference: CS224 Abigail See, Matthew Lamm

Models are sensitive to initialization

- From Michael Nielsen
  http://neuralnetworksanddeeplearning.com/chap3.html



Training a gated RNN

1. Use an LSTM or GRU: *it makes your life so much simpler!*
2. Initialize recurrent matrices to be orthogonal
3. Initialize other matrices with a sensible (**small!**) scale
4. Initialize forget gate bias to 1: *default to remembering*
5. Use adaptive learning rate algorithms: *Adam, AdaDelta, …*
6. Clip the norm of the gradient: *1–5 seems to be a reasonable threshold when used together with Adam or AdaDelta.*
7. Either only dropout vertically or look into using Bayesian Dropout (Gal and Gahramani – not natively in PyTorch)
8. *Be patient! Optimization takes time*

[Saxe et al., ICLR2014;
Ba, Kingma, ICLR2015;
Zeiler, arXiv2012;
Pascanu et al., ICML2013]