# CSCE 5218 & 4930
# Deep Learning

## Recurrent Neural Networks

Slides adapted from Adriana Kovashka

# Plan for this lecture

- Recurrent neural networks
  - Basics
  - Training (backprop through time, vanishing gradient)
  - Recurrent networks with gates (GRU, LSTM)
- Applications in NLP and vision
  - Neural machine translation
  - Image/video captioning

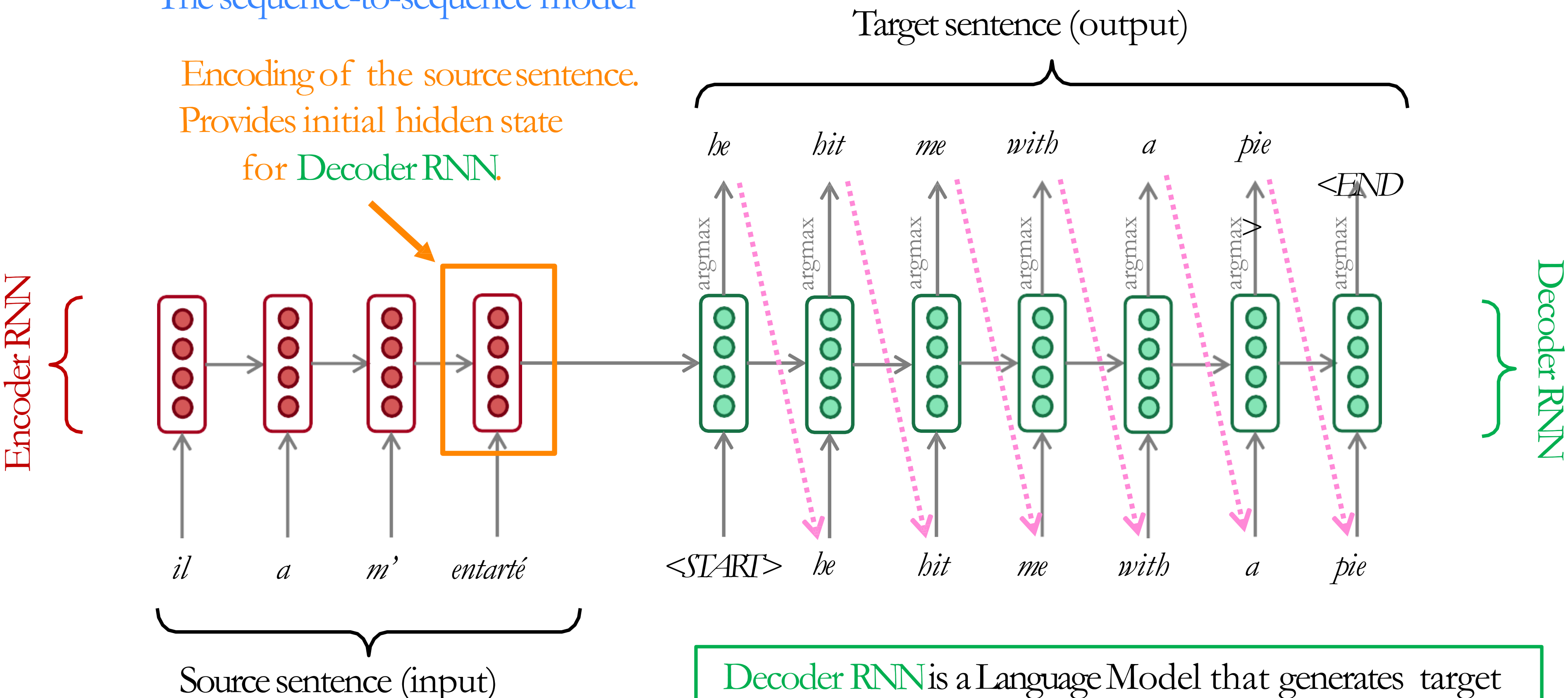
# Applications

# Neural Machine Translation

- Neural Machine Translation (NMT) is a way to do Machine Translation with a *single neural network*
- The neural network architecture is called sequence-to-sequence (aka seq2seq) and it involves *two* RNNs.

# Neural Machine Translation (NMT)

## The sequence-to-sequence model




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

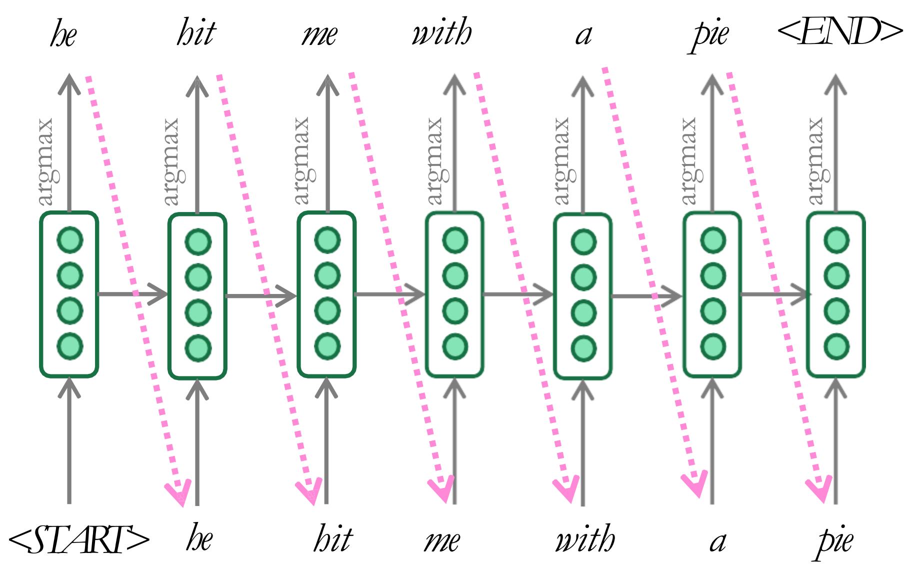
Encoder RNN produces an encoding of the source sentence.

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

Abigail See

# Greedy decoding

- We saw how to generate (or "decode") the target sentence by taking argmax on each step of the decoder




- This is greedy decoding (take most probable word on each step)
- **Problems with this method?**

# Problems with greedy decoding

- Greedy decoding has no way to undo decisions!
  - Input: *il a m'entarté* *(he hit me with a pie)*
  - → *he* ______
  - → *he hit* ______
  - → *he hit a* ______ (whoops! no going back now…)

- How to fix this?

# Exhaustive search decoding

- Ideally we want to find a (length $T$) translation $y$ that maximizes

$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots,P(y_T|y_1,\ldots,y_{T-1},x)$$
$$= \prod_{t=1}^{T} P(y_t|y_1,\ldots,y_{t-1},x)$$

- We could try computing all possible sequences $y$
  - This means that on each step $t$ of the decoder, we're tracking $V^T$ possible partial translations, where $V$ is vocabulary size
  - This $O(V^T)$ complexity is far too expensive!

# Beam search decoding

- Core idea: On each step of decoder, keep track of the *k* most probable partial translations (hypotheses)
  - *k* is the beam size (in practice around 5 to 10)

- A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top *k* on each step

- Beam search is not guaranteed to find optimal solution
- But much more efficient than exhaustive search!

Abigail See

# Beam search decoding: example

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

<START>

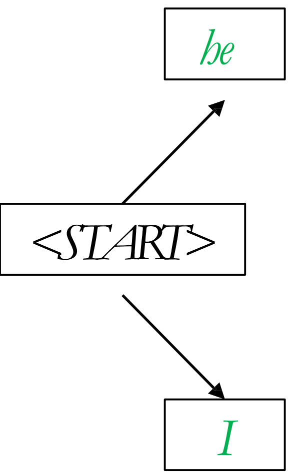
Calculate prob dist of next word

Abigail See

# Beam search decoding: example

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

$-0.7 = \log P_{LM}(\textit{he} \,|\, <\textit{START}>)$



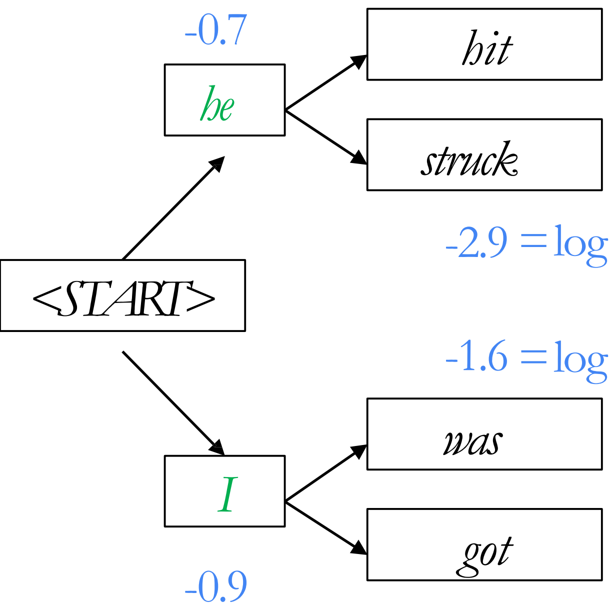

$-0.9 = \log P_{LM}(\textit{I} \,|\, <\textit{START}>)$

Take top *k* words
and compute scores

# Beam search decoding: example

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

-1.7 = log $P_{LM}$(*hit*| <*START*> *he*) + -0.7




-2.9 = log $P_{LM}$(*struck*| <*START*> *he*) + -0.7

-1.6 = log $P_{LM}$(*was*| <*START*> *I*) + -0.9

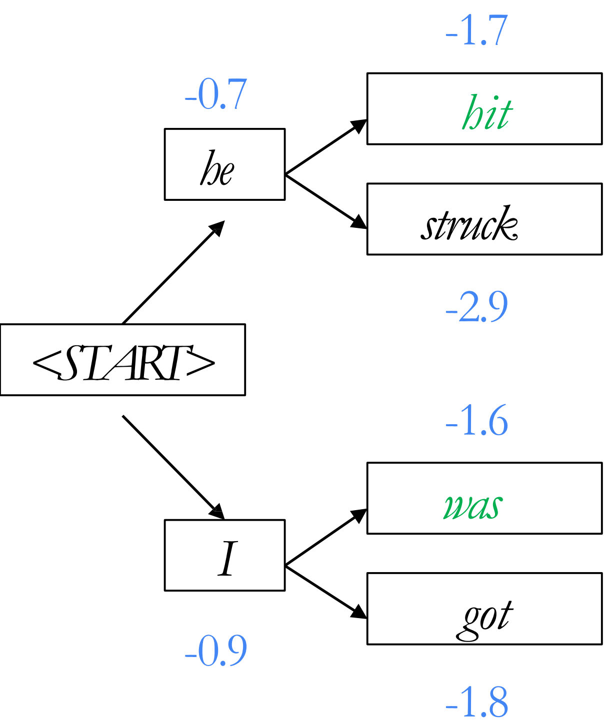
-1.8 = log $P_{LM}$(*got*| <*START*> *I*) + -0.9

For each of the *k* hypotheses, find top *k* next words and calculate scores

Abigail See

# Beam search decoding: example

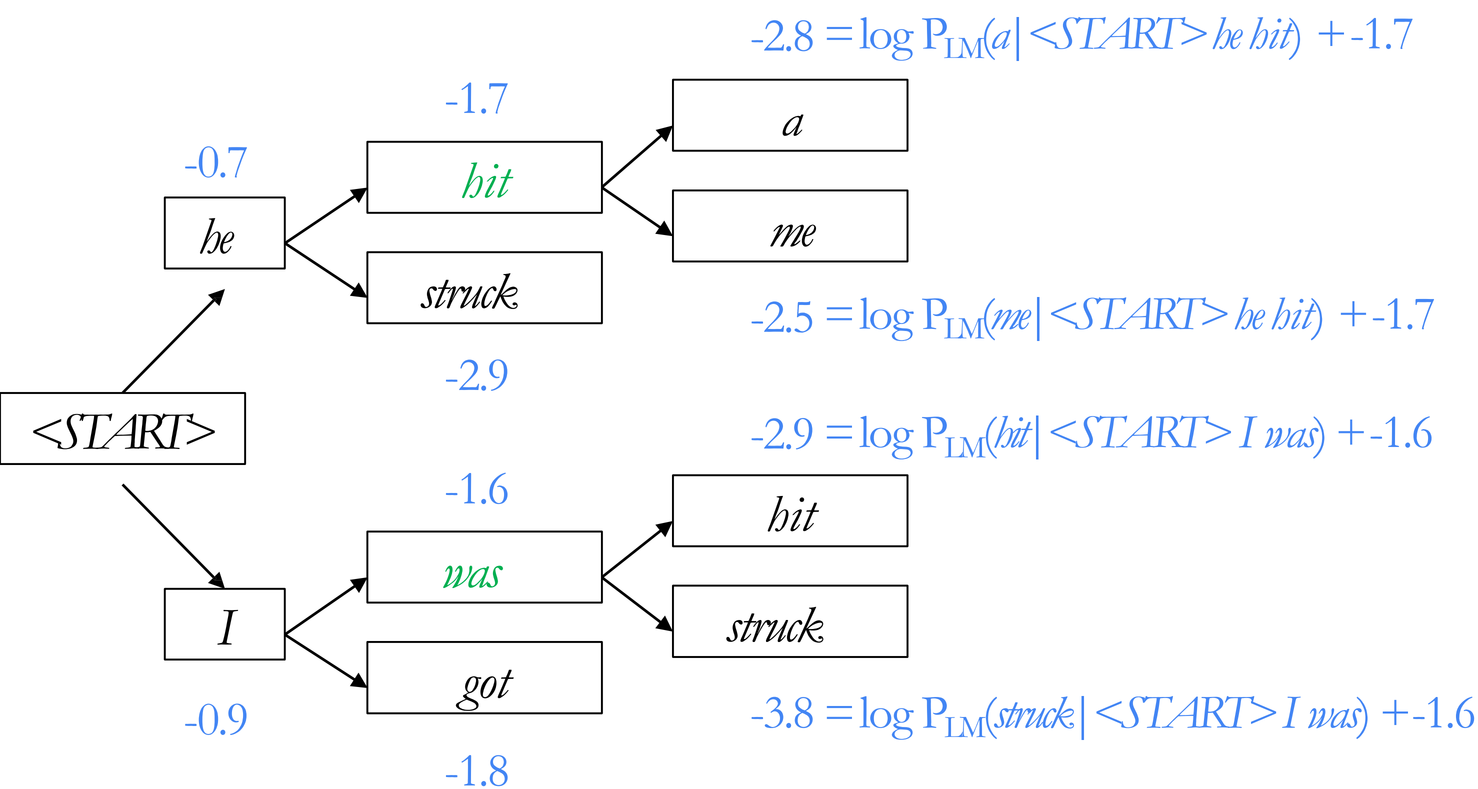
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




Of these $k^2$ hypotheses,
just keep $k$ with highest scores

# Beam search decoding: example

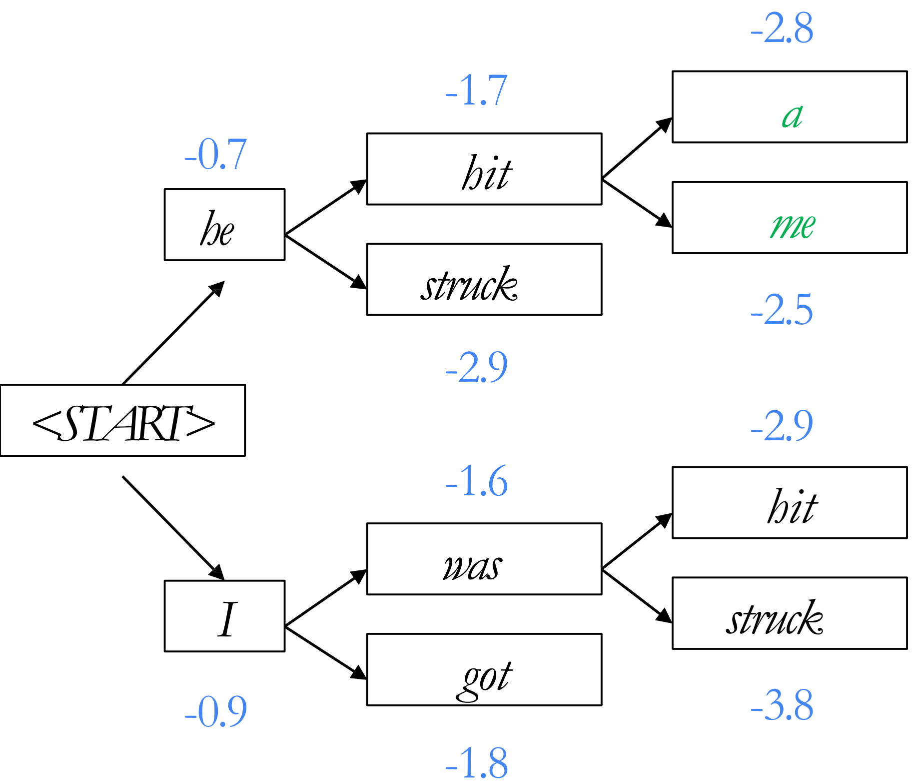
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




For each of the *k* hypotheses, find top *k* next words and calculate scores

# Beam search decoding: example

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

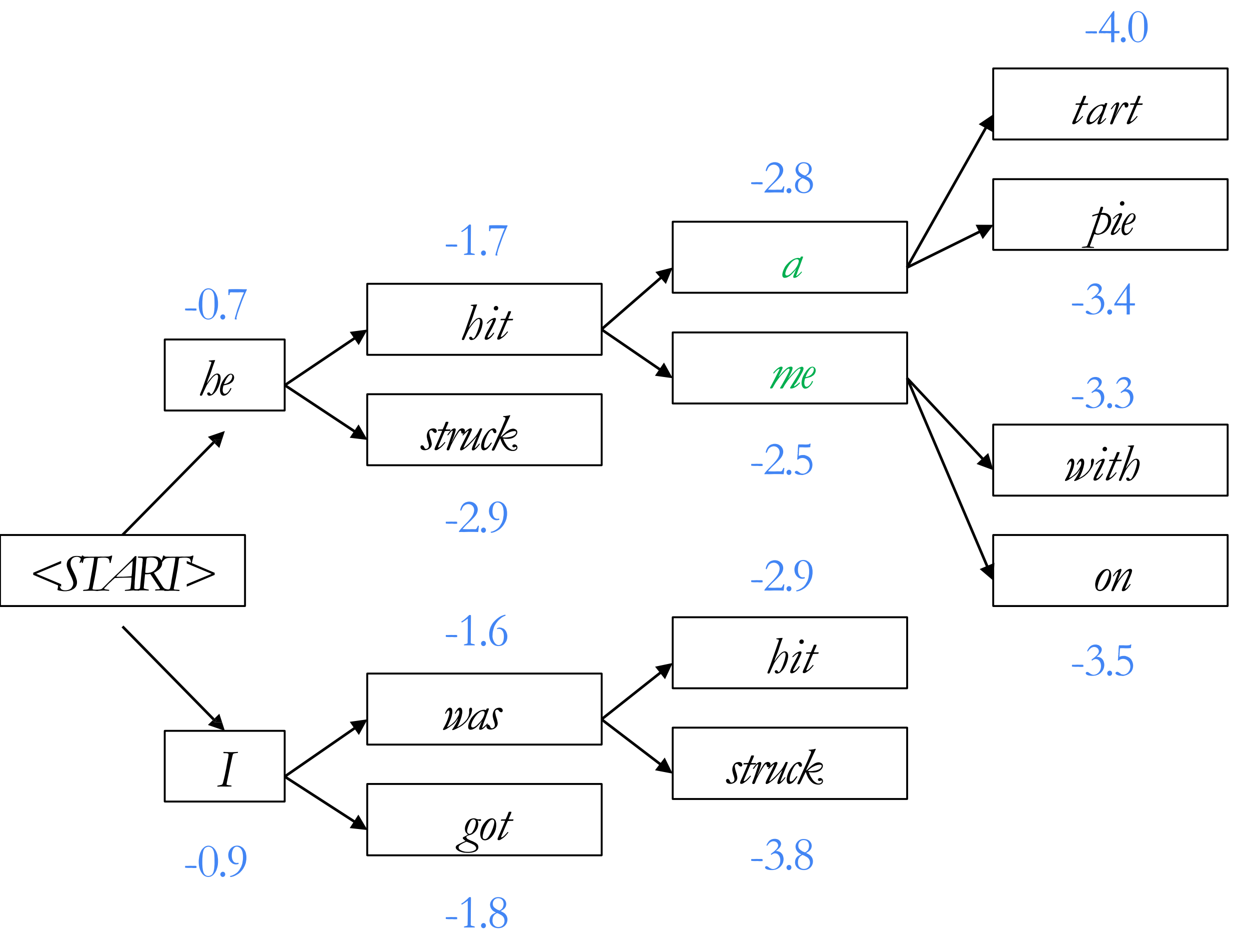



Of these $k^2$ hypotheses,
just keep $k$ with highest scores

Abigail See

# Beam search decoding: example

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

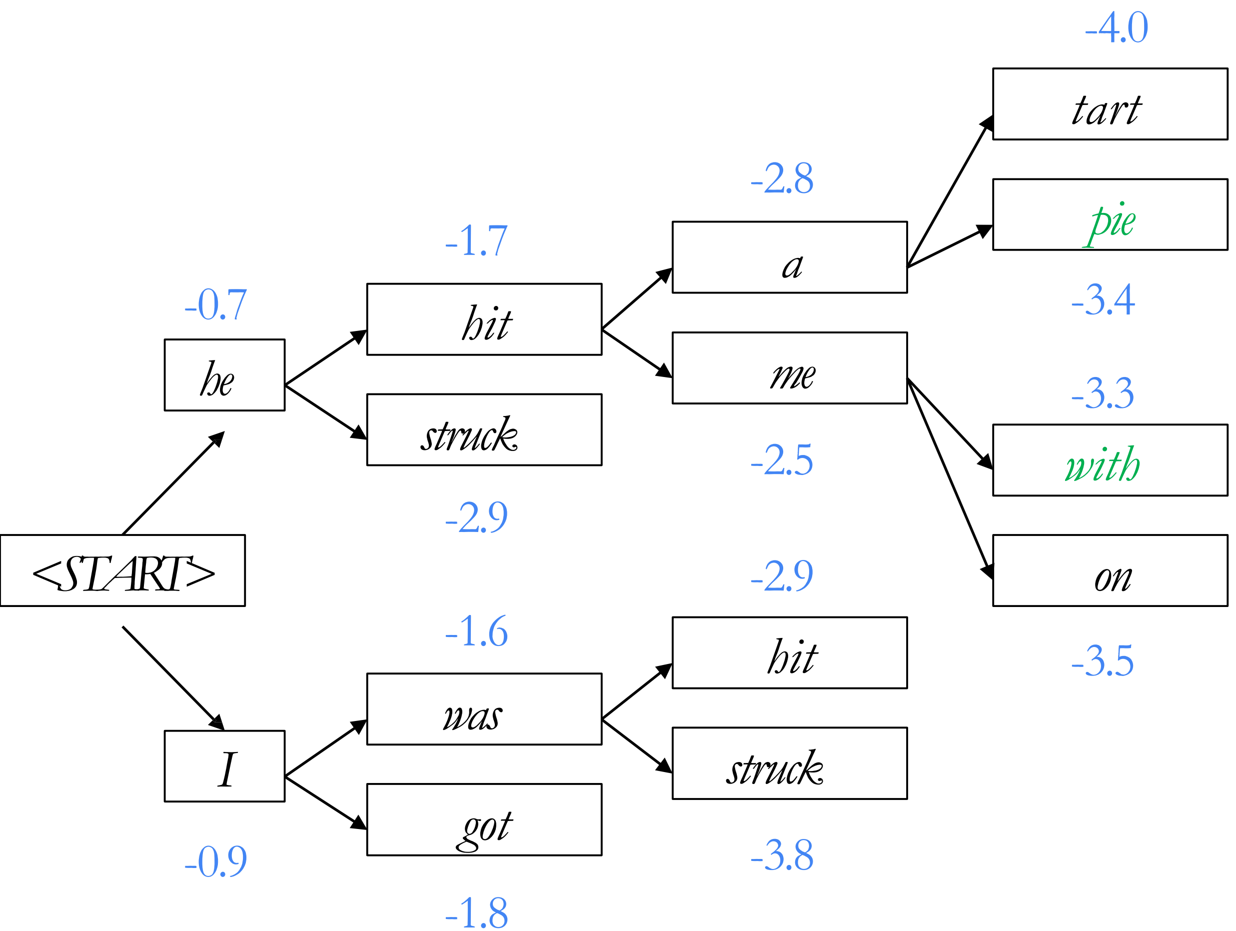



For each of the $k$ hypotheses, find top $k$ next words and calculate scores

Abigail See

# Beam search decoding: example

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

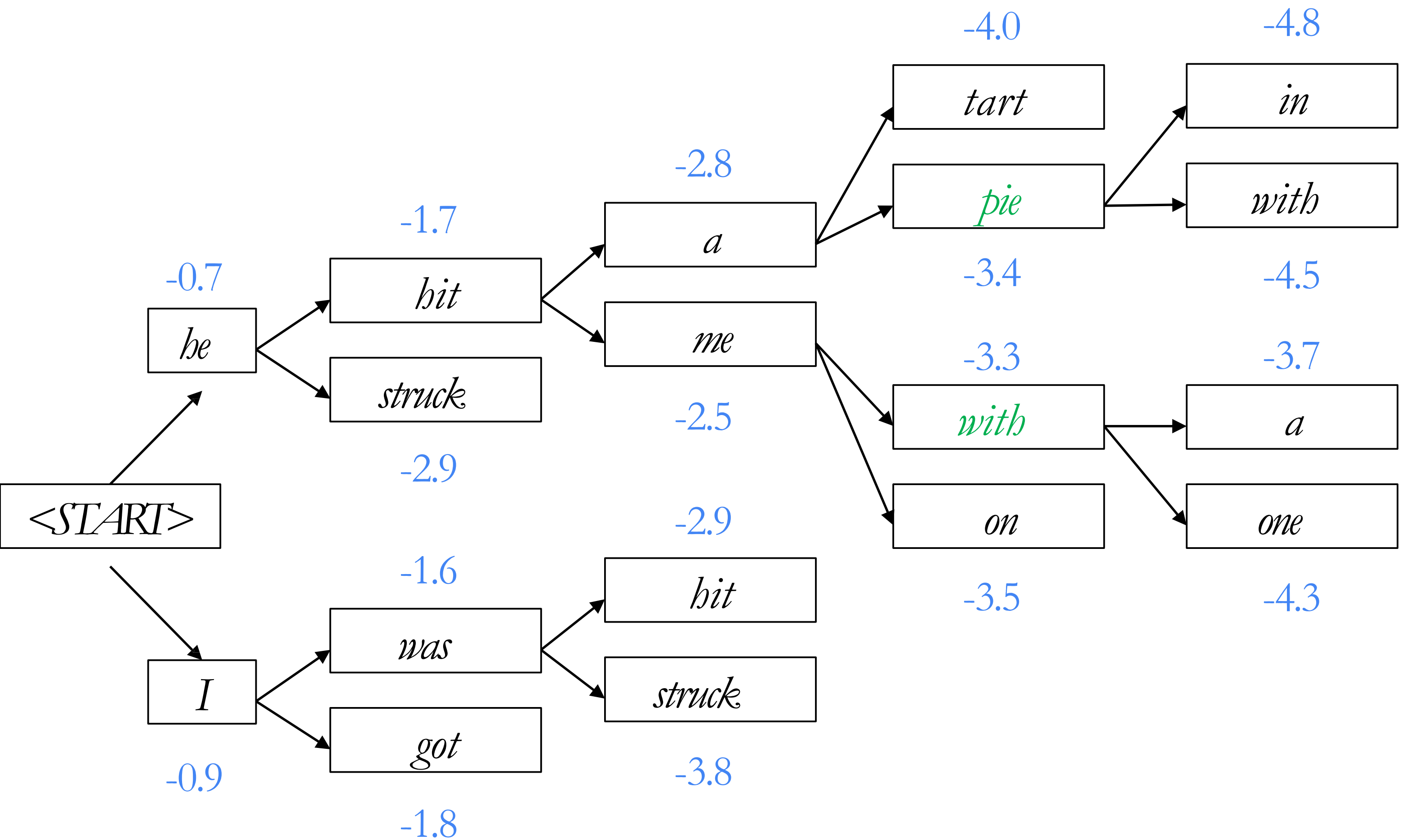



Of these $k^2$ hypotheses, just keep $k$ with highest scores

Abigail See

# Beam search decoding: example

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

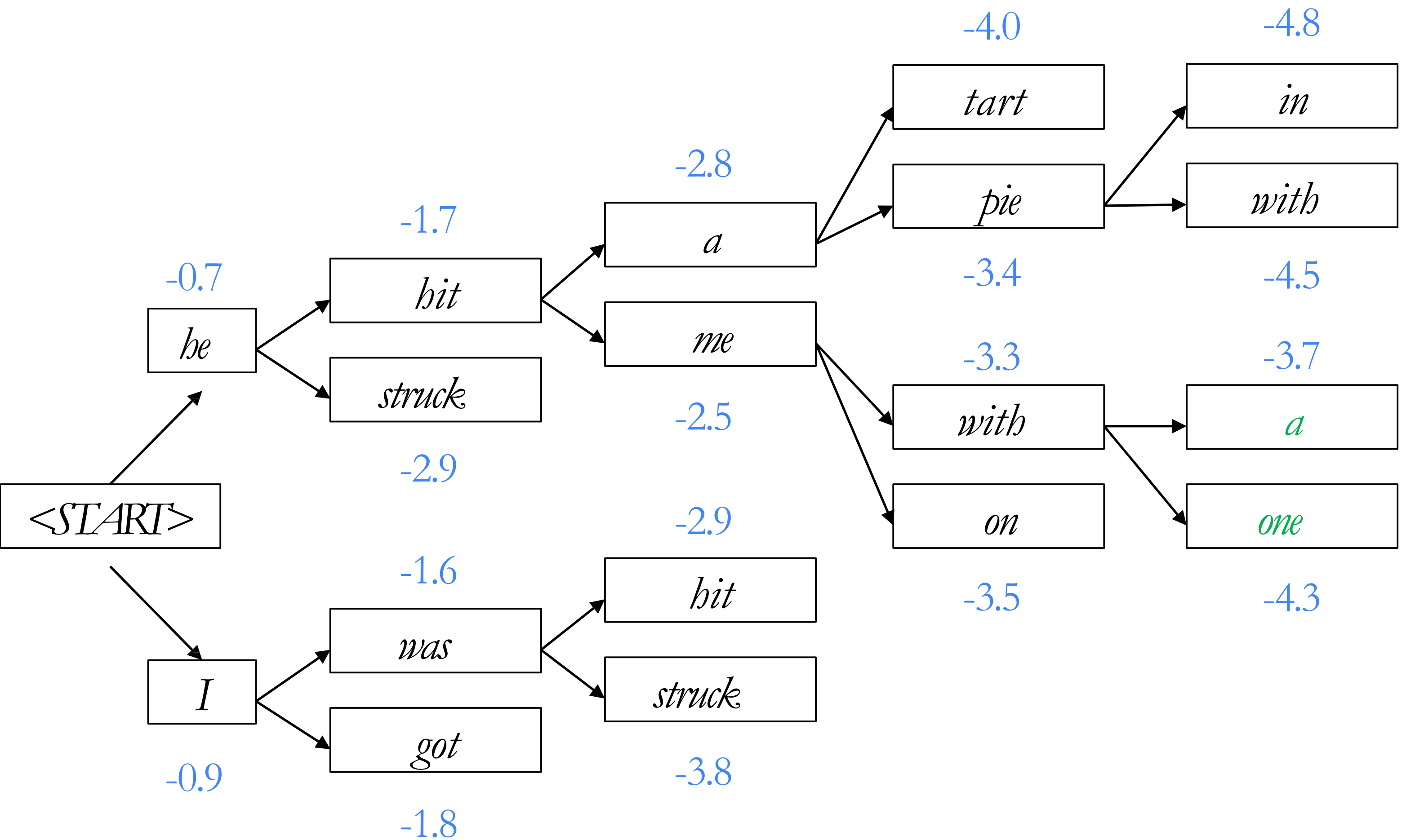



For each of the *k* hypotheses, find top *k* next words and calculate scores

Abigail See

# Beam search decoding: example

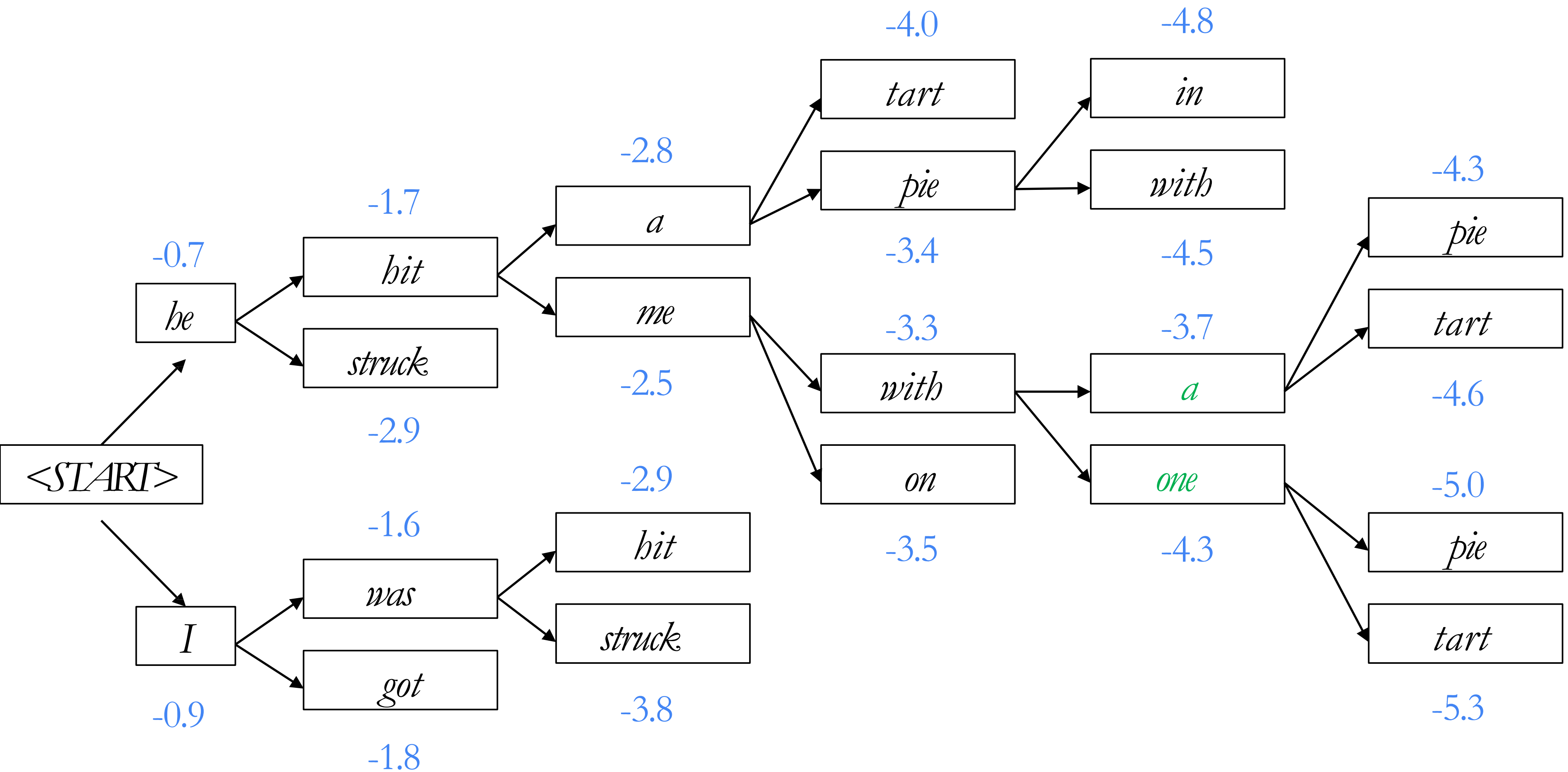
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




Of these $k^2$ hypotheses,
just keep $k$ with highest scores

# Beam search decoding: example

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

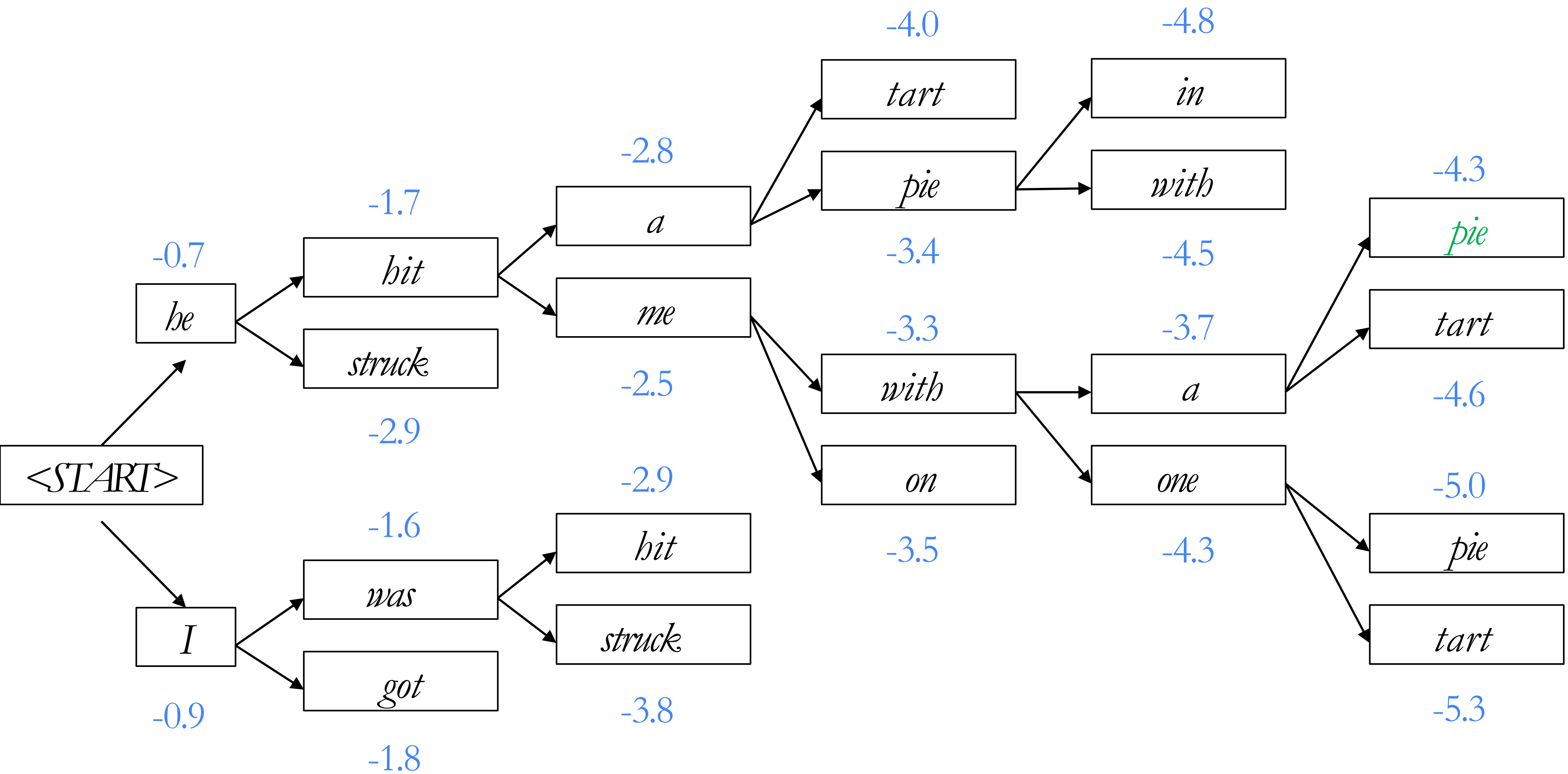



For each of the *k* hypotheses, find top *k* next words and calculate scores

Abigail See

# Beam search decoding: example

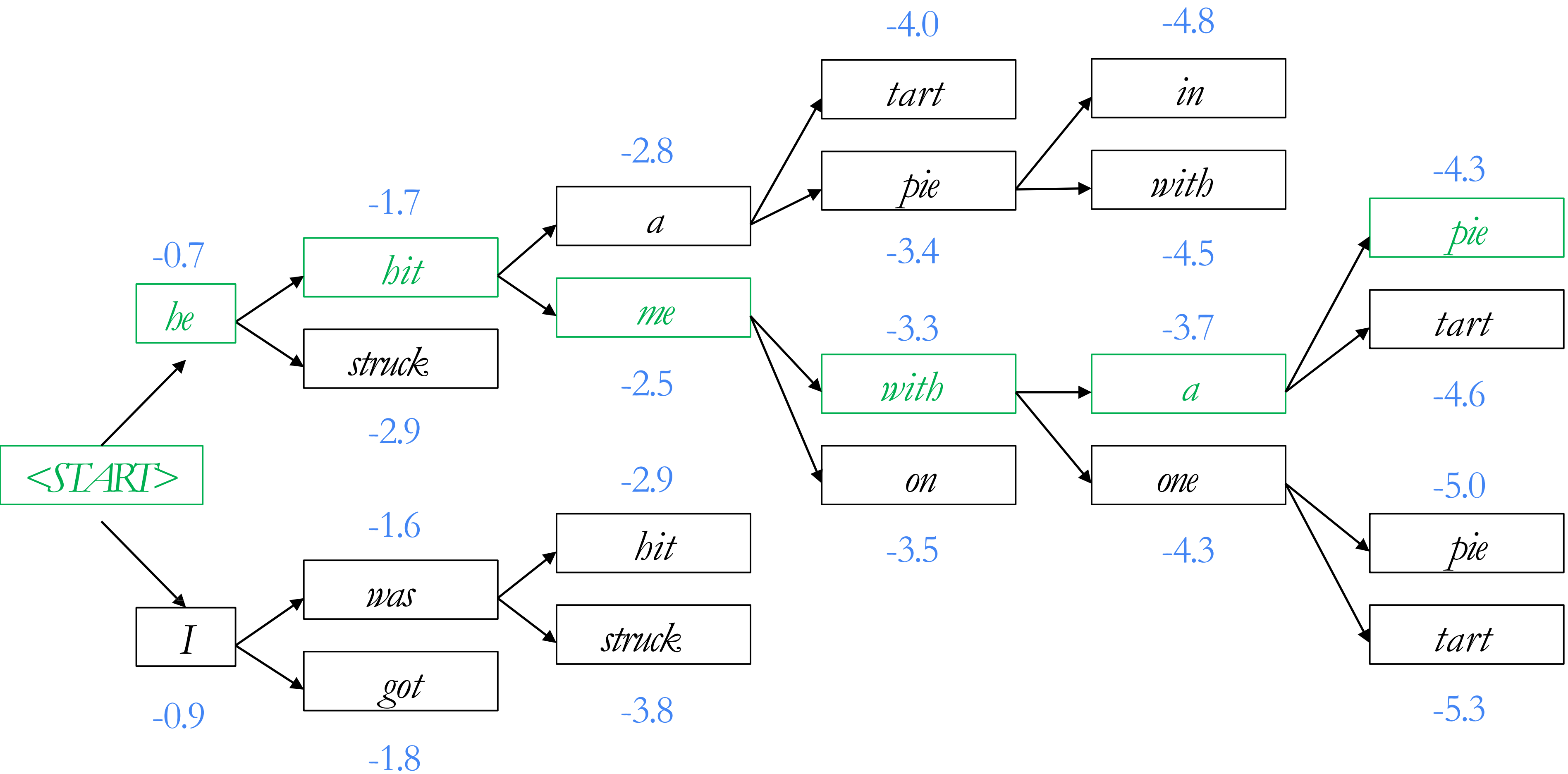
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




This is the top-scoring hypothesis!

# Beam search decoding: example

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




Backtrack to obtain the full hypothesis

# How do we evaluate Machine Translation?

**BLEU** (**B**ilingual **E**valuation **U**nderstudy)

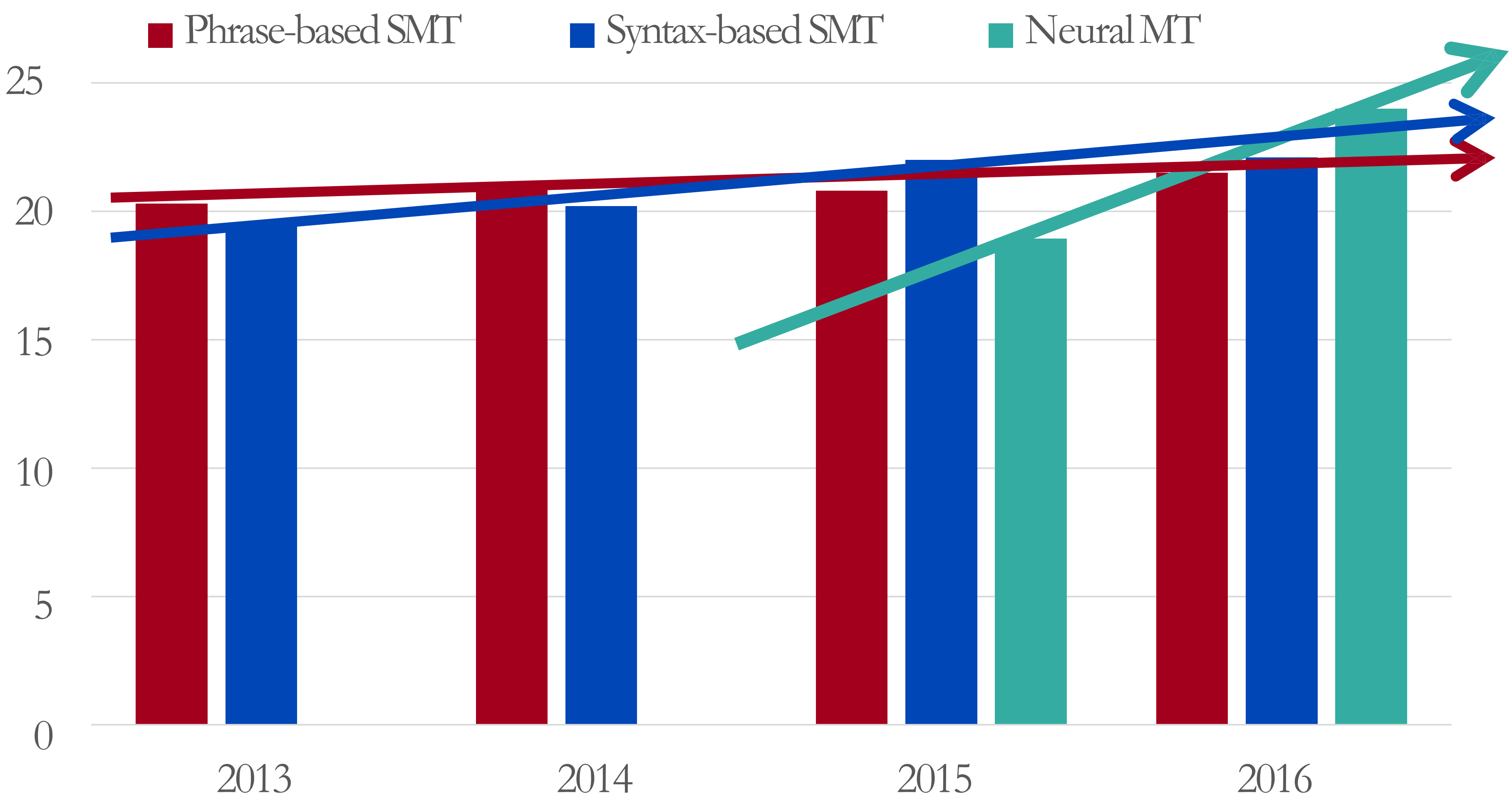
- BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:
  - *n*-gram precision (usually for 1, 2, 3 and 4-grams)
  - Plus a penalty for too-short system translations

- BLEU is useful but imperfect
  - There are many valid ways to translate a sentence
  - So 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.

# MT progress overtime

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




**Source:** http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf

# NMT: the biggest success story of NLP Deep Learning

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

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

# So is Machine Translation solved?

- **Nope!**
- Many difficulties remain:
  - Out-of-vocabulary words
  - Domain mismatch between train and test data
  - Maintaining context over longer text

**Further reading:** *"Has AI surpassed humans at translation? Not even close!"*
https://www.skynettoday.com/editorials/state_of_nmt

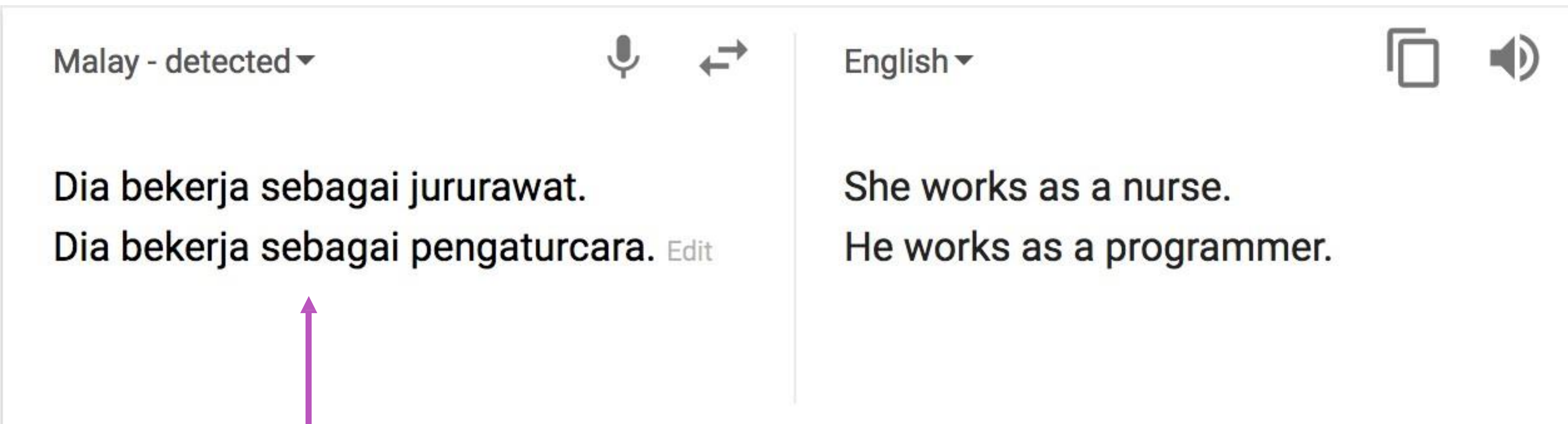
# So is Machine Translation solved?

- **Nope!**
- Using common sense is still hard


?

# So is Machine Translation solved?

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




Didn't specify gender

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

# NMT research continues

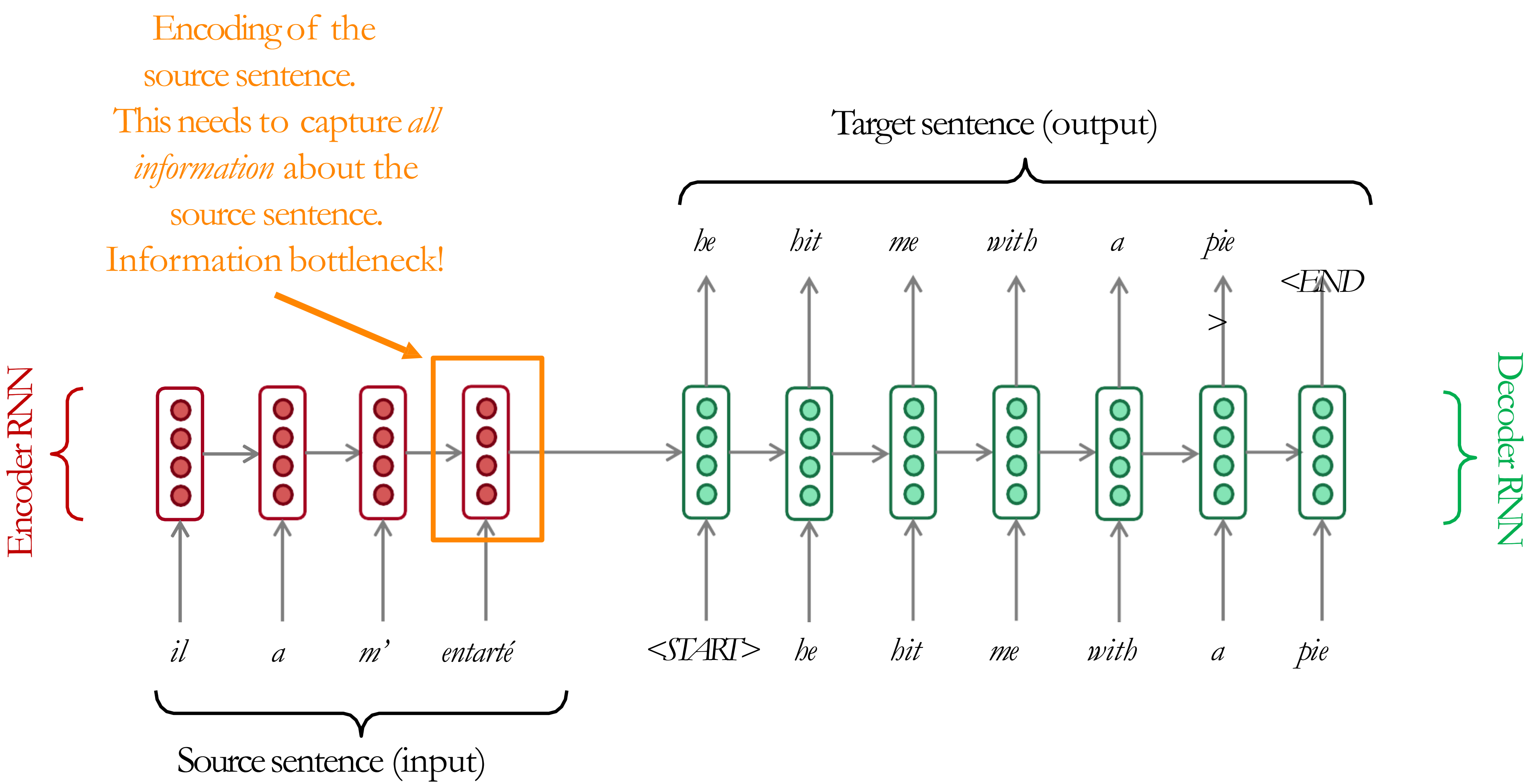
NMT is the **flagship task** for NLP Deep Learning

- NMT research has pioneered many of the recent innovations of NLP Deep Learning

- In **2019**: NMT research continues to thrive
  - Researchers have found *many, many* improvements to the "vanilla" seq2seq NMT system
  - But one improvement is so integral that it is the new vanilla…

**ATTENTION**

# Sequence-to-sequence: the bottleneck problem



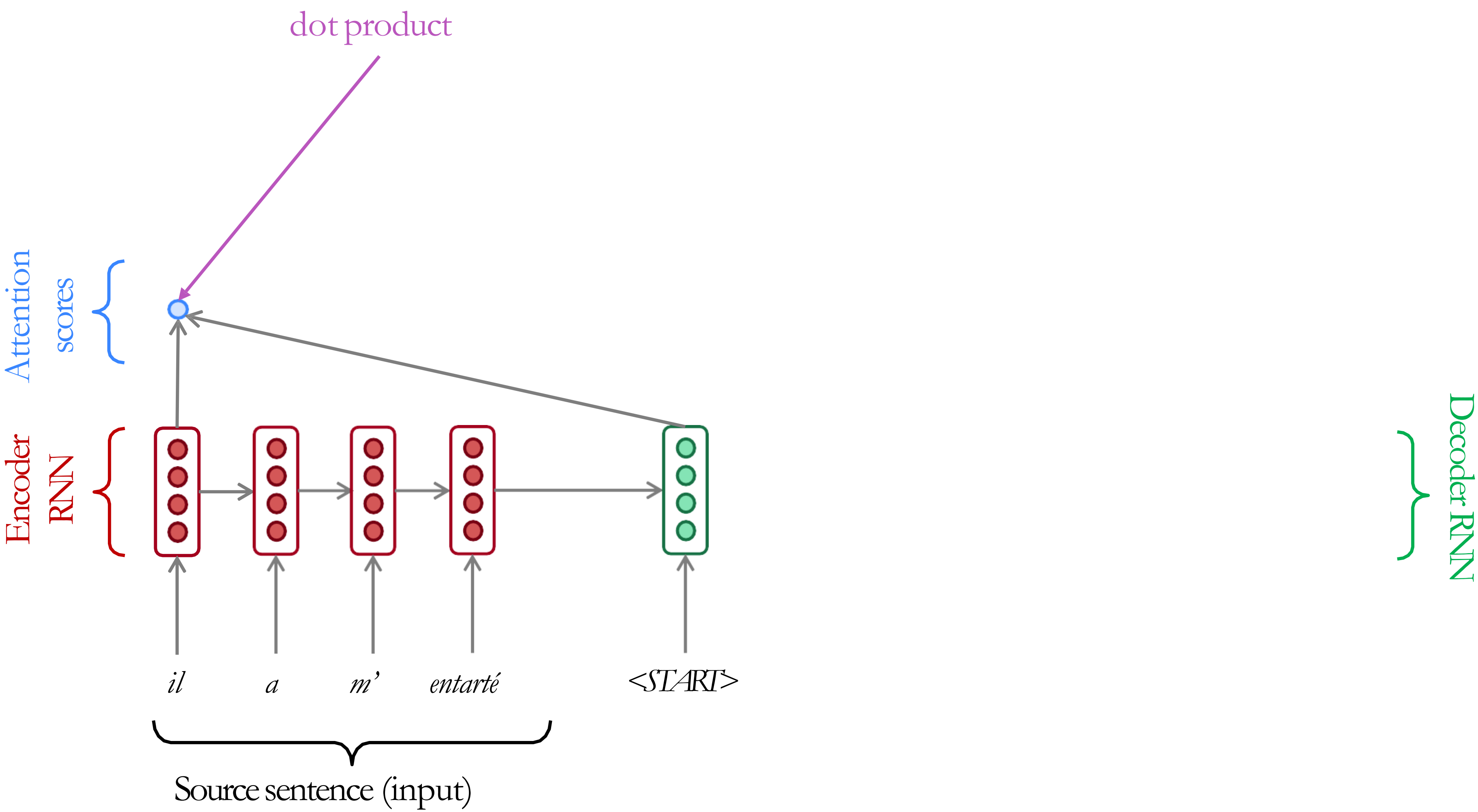
# Attention

- **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

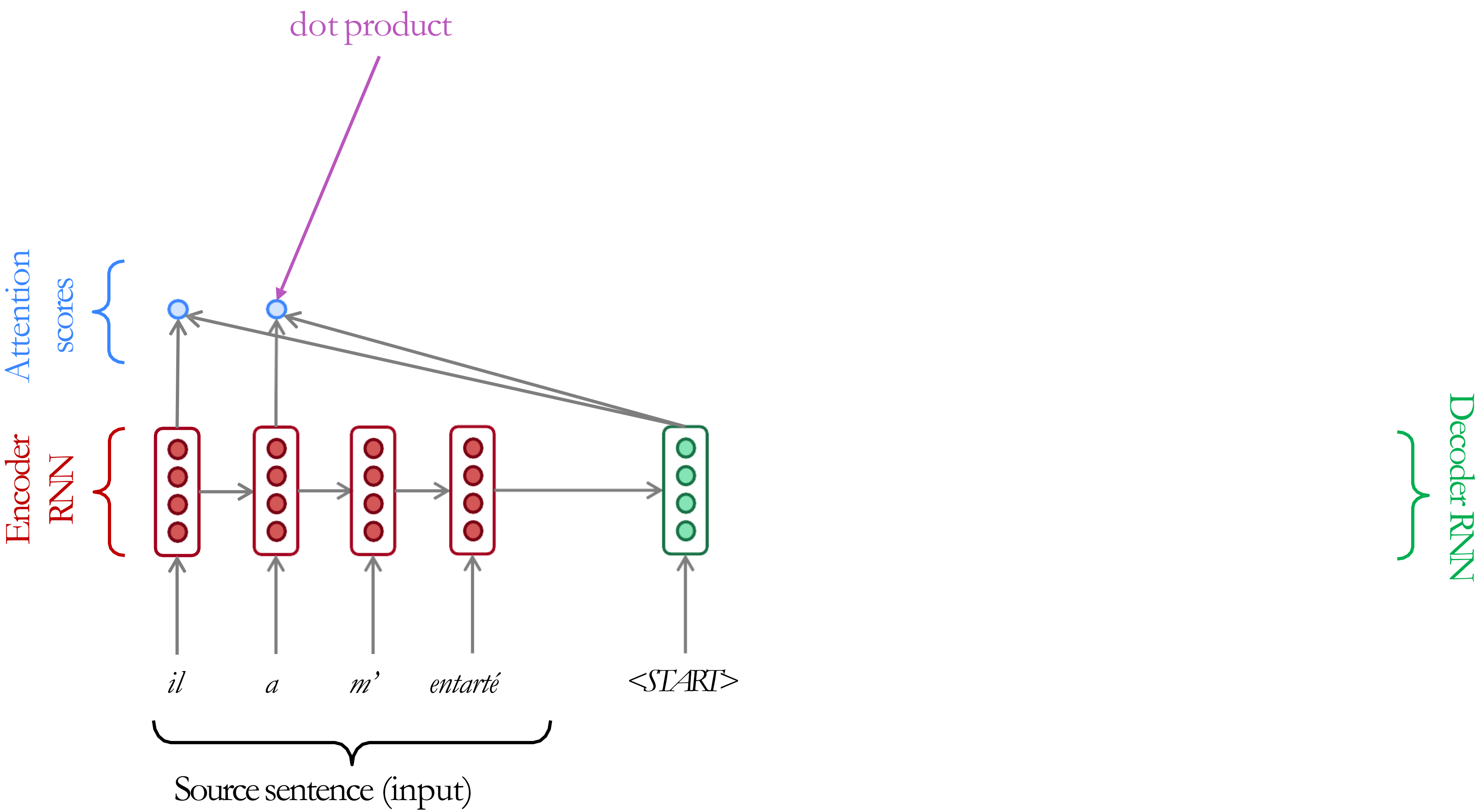
- First we will show via diagram (no equations), then we will show with equations

# Sequence-to-sequence with attention



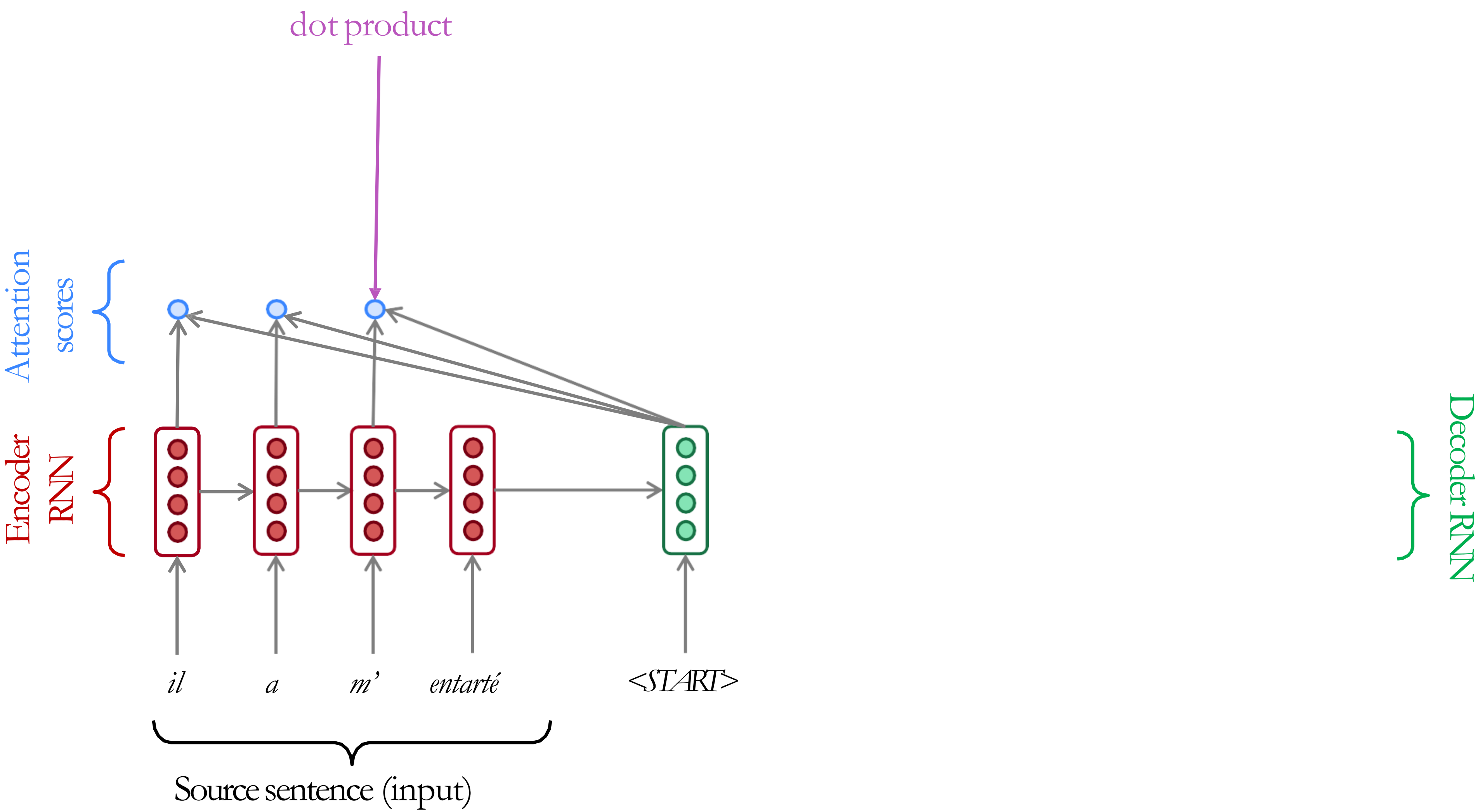

Abigail See

# Sequence-to-sequence with attention

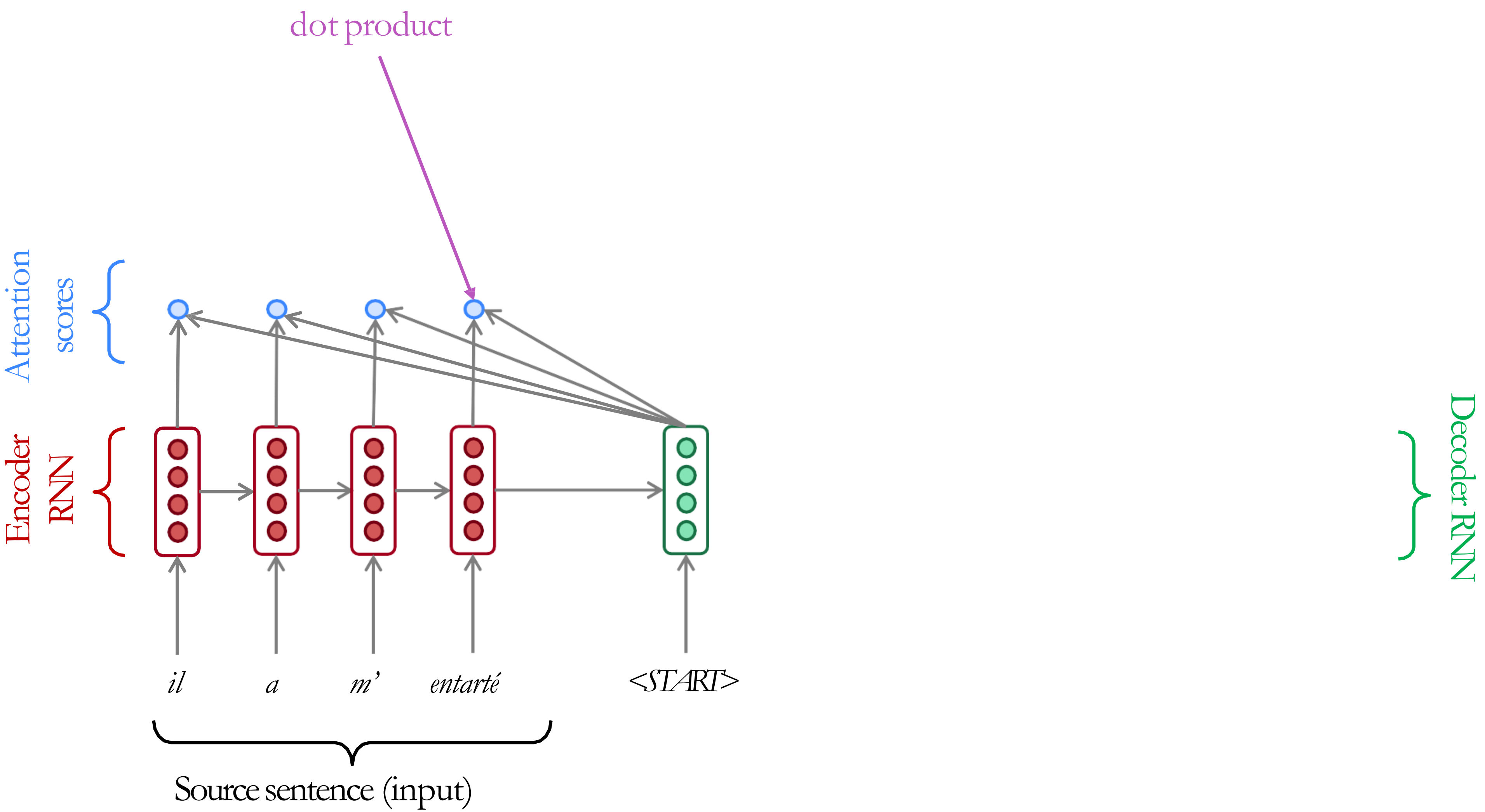



Abigail See

# Sequence-to-sequence with attention

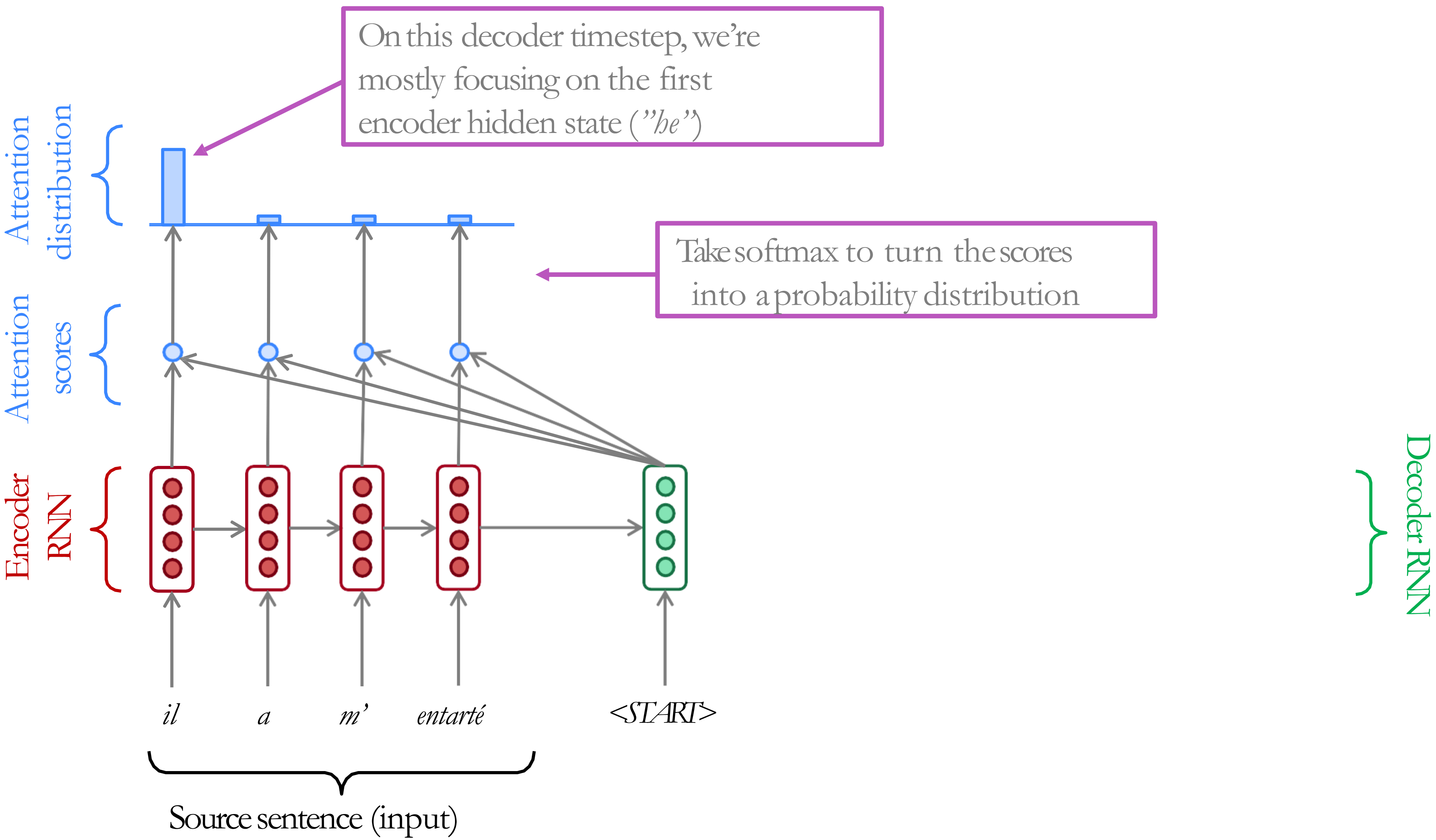


# Sequence-to-sequence with attention



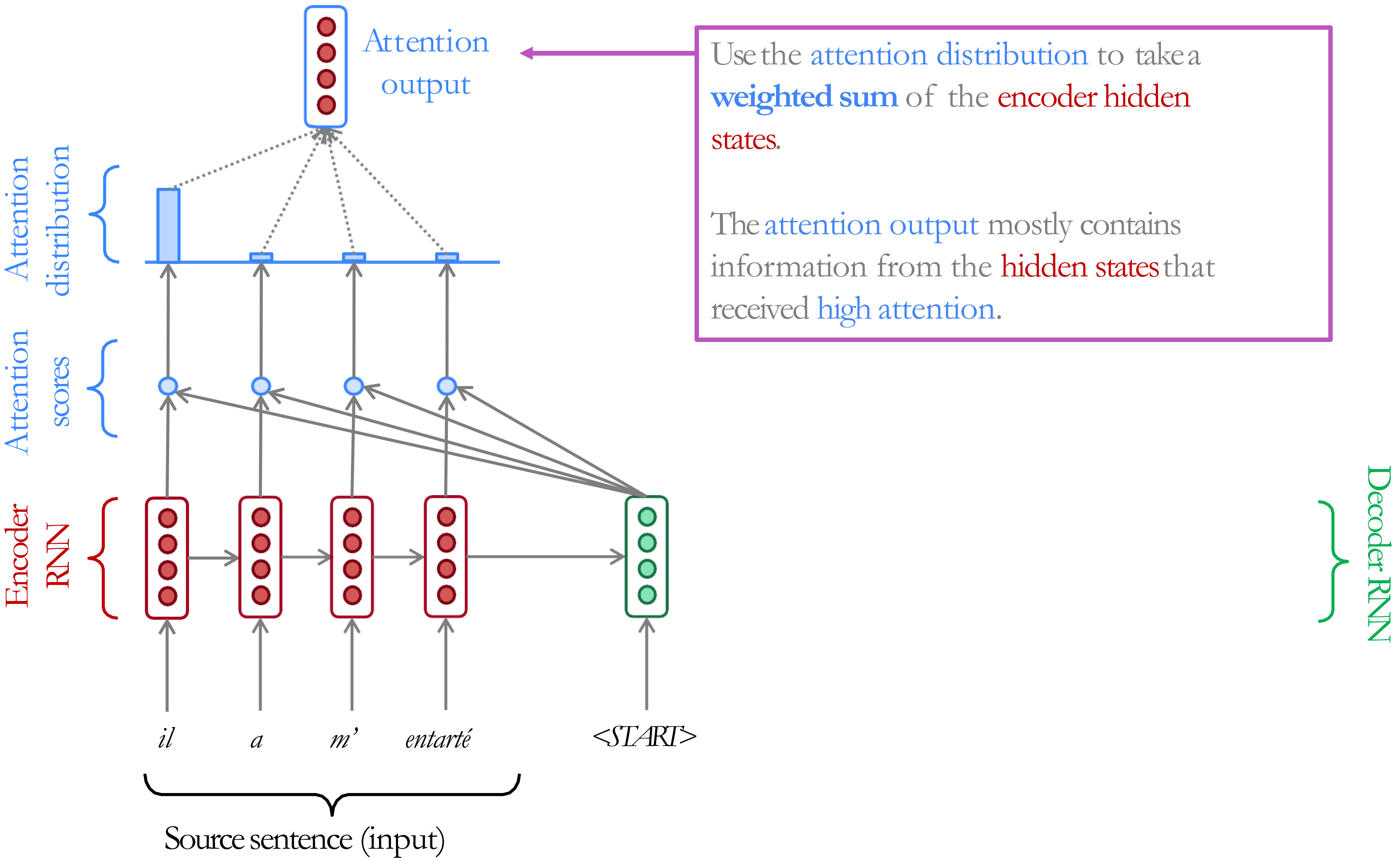

Abigail See

# Sequence-to-sequence with attention




Abigail See

# Sequence-to-sequence with attention

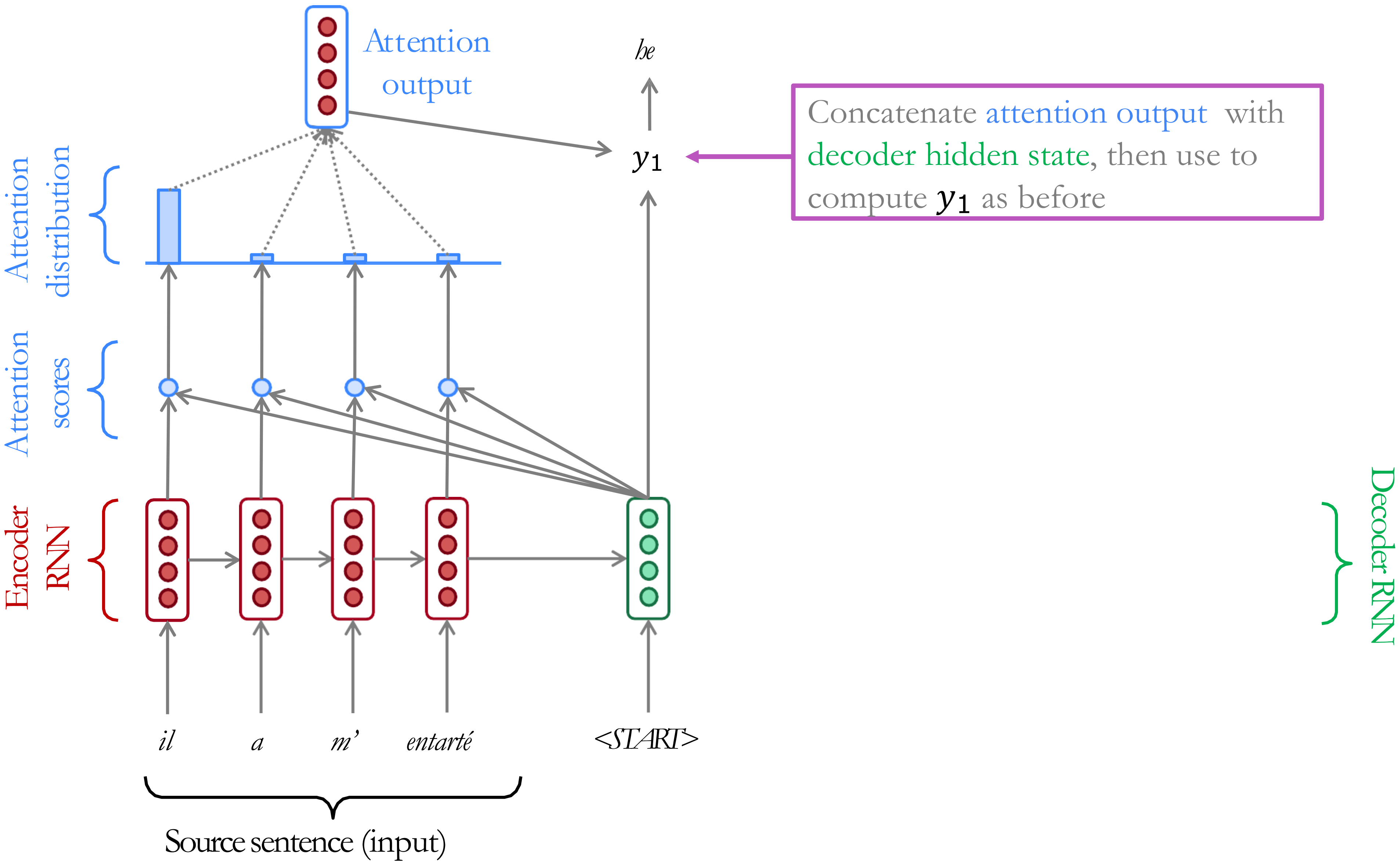



Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

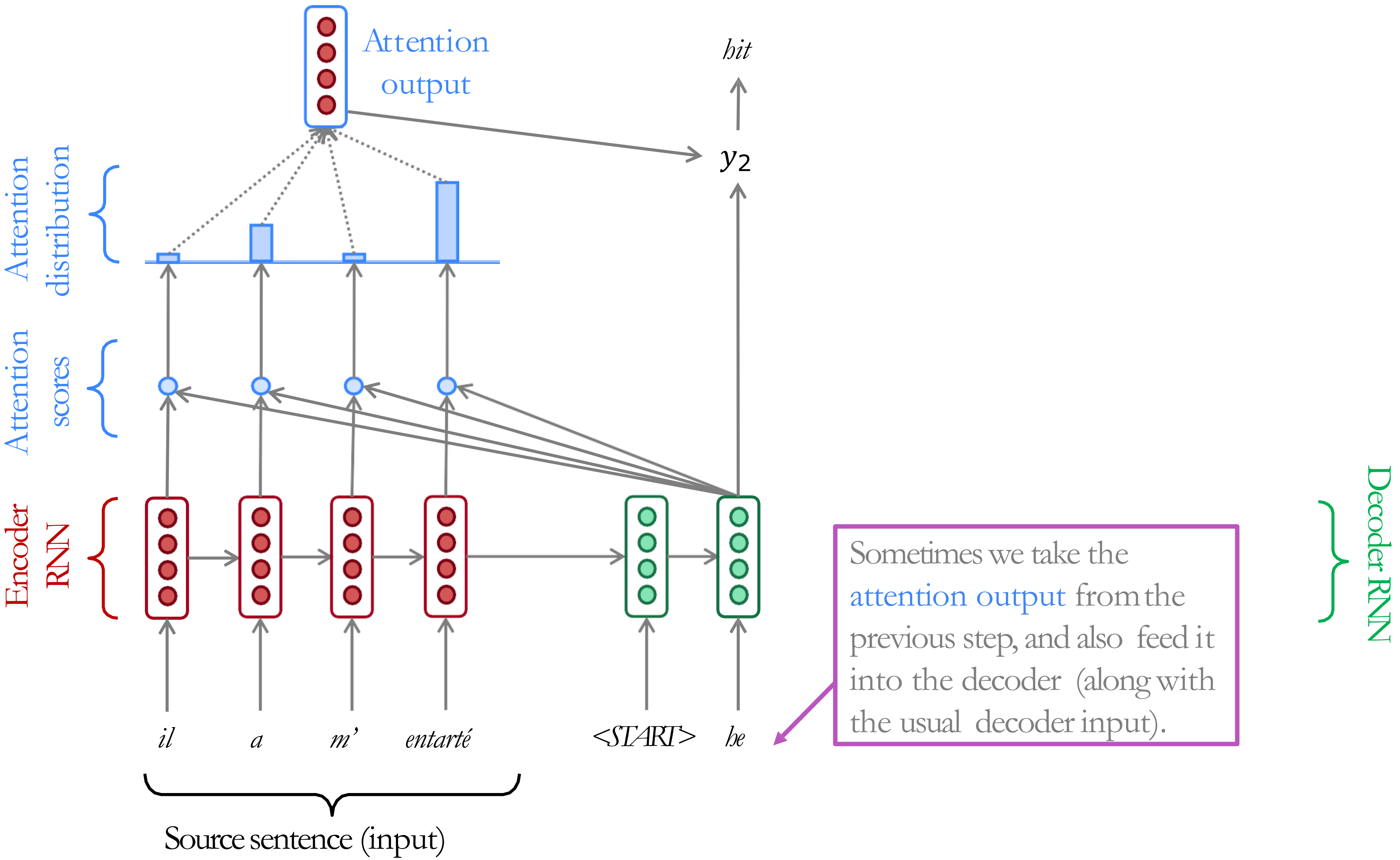
Abigail See

# Sequence-to-sequence with attention




Abigail See

# 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).

Abigail See

# 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 $\alpha^t$ for this step (this is a probability distribution and sums to 1)

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

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

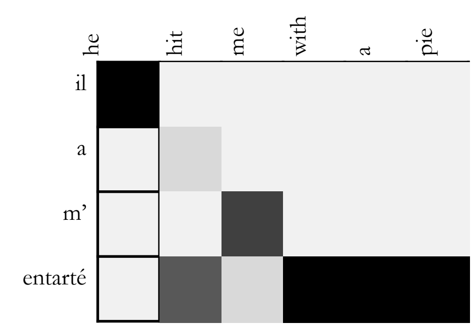
$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

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



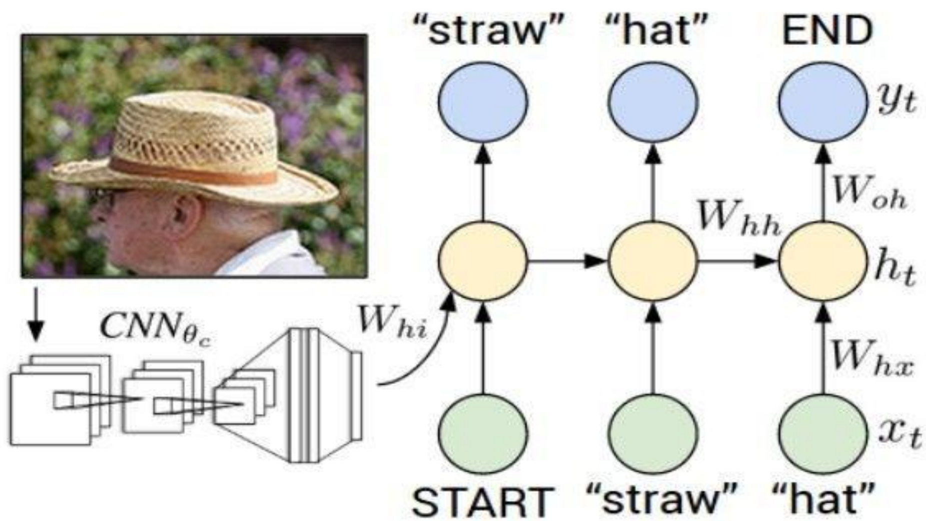
# Attention is a *general* Deep Learning technique

- 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 seq2seq + attention model, each decoder hidden state (query) *attends to* all encoder hidden states (values).

# Image Captioning




CVPR 2015:

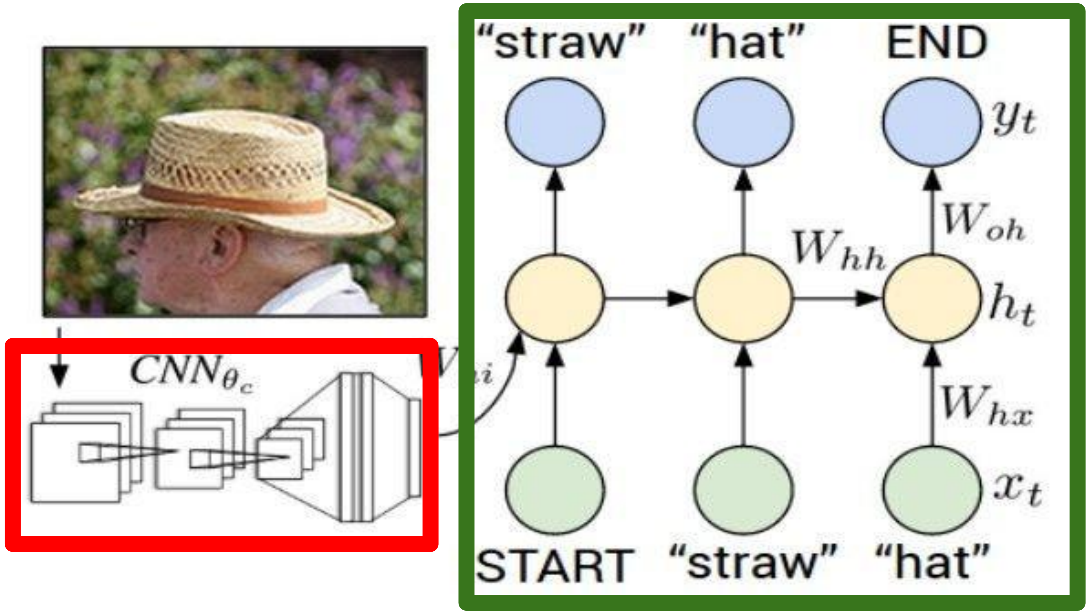
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei Show and Tell: A Neural Image Caption Generator, Vinyals et al.

Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al. Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Adapted from Andrej Karpathy

# Image Captioning

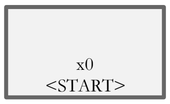
Recurrent Neural Network




Convolutional Neural Network

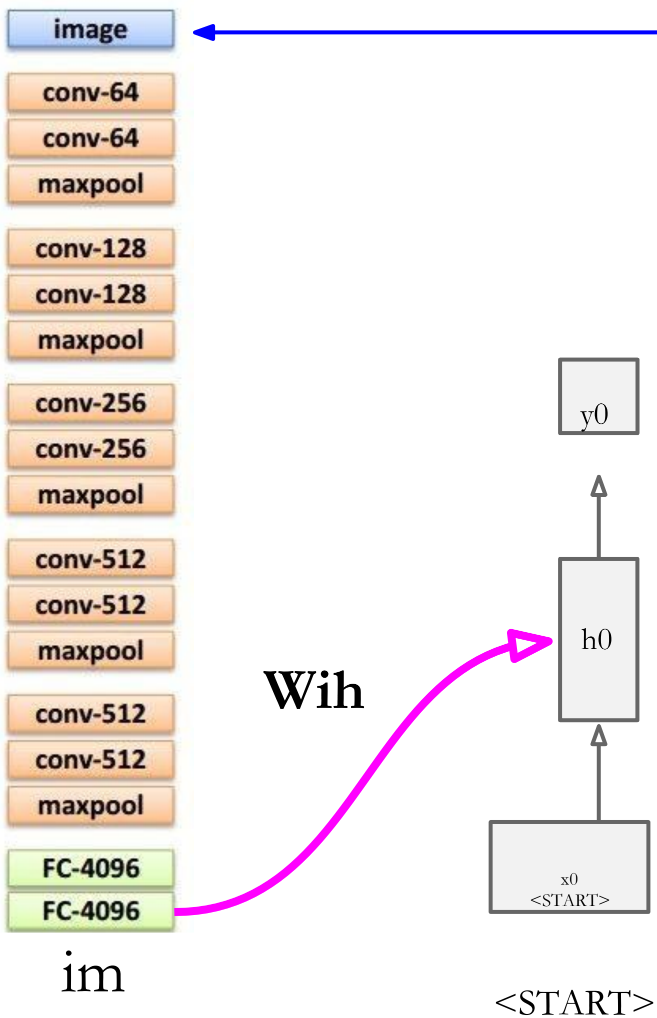
# Image Captioning


test image




<START>

# Image Captioning




test image

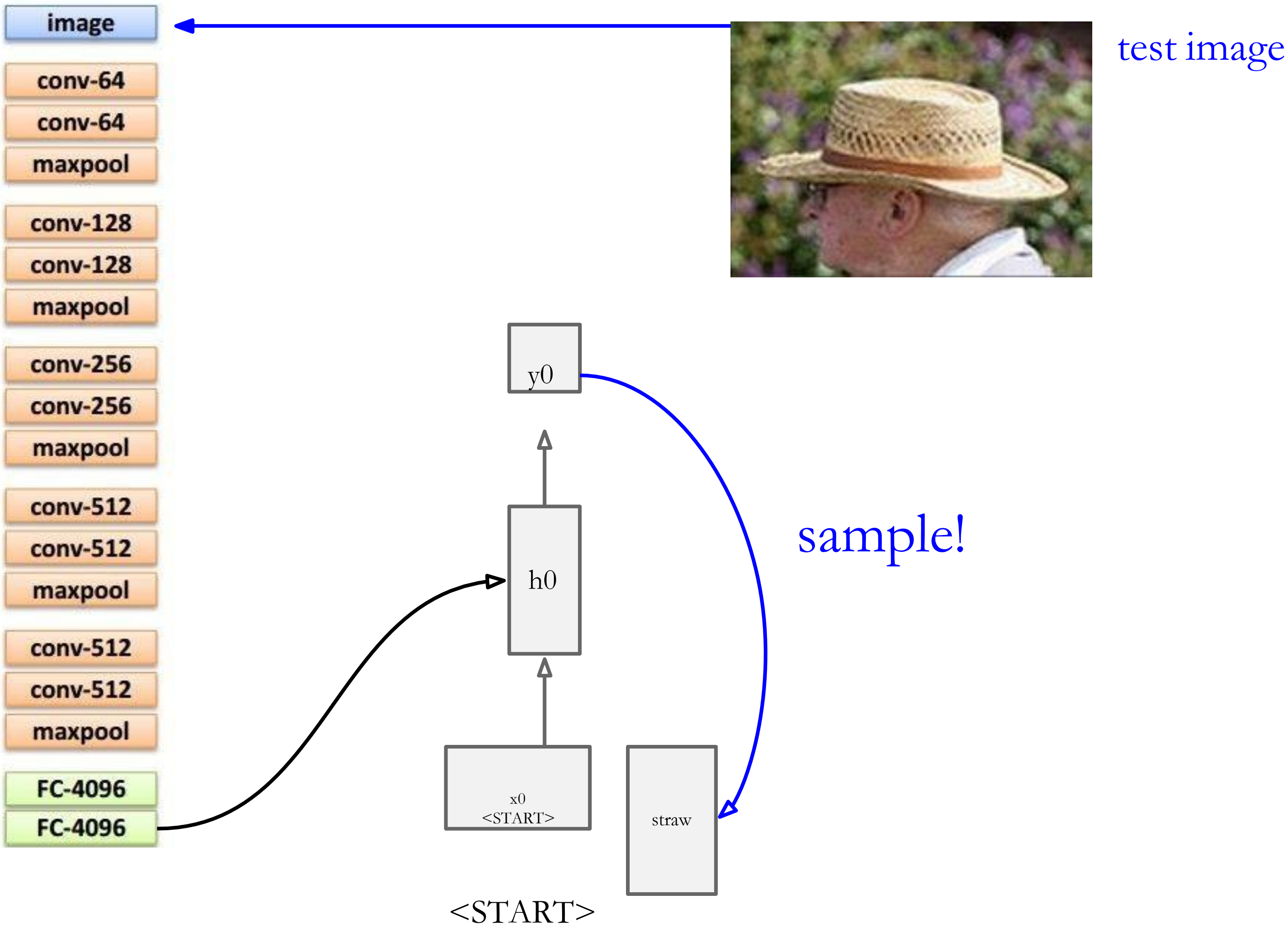
**before:**

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

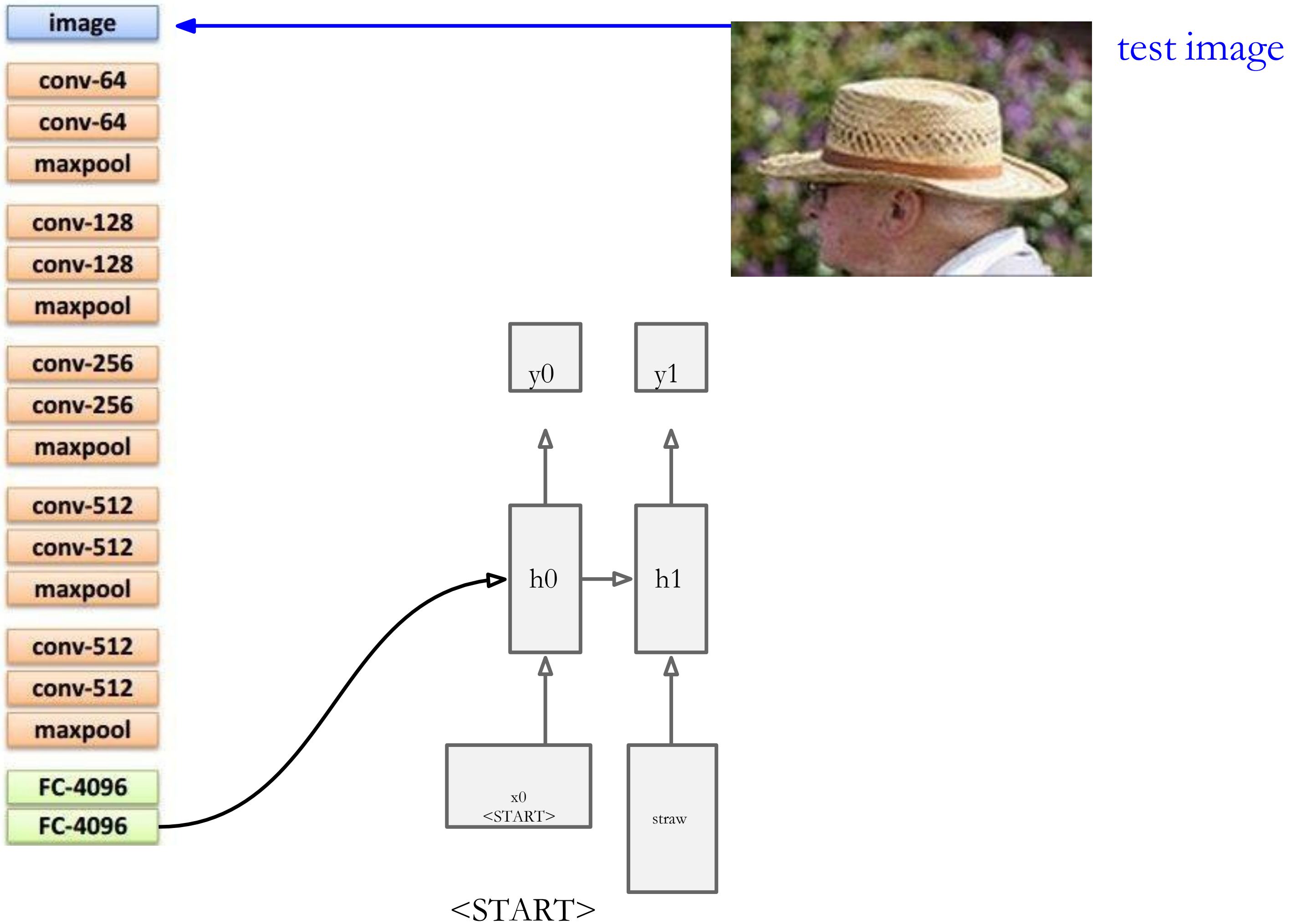
**now:**

$$h = \tanh(W_{xh} * x + W_{hh} * h + \mathbf{W_{ih} * im})$$

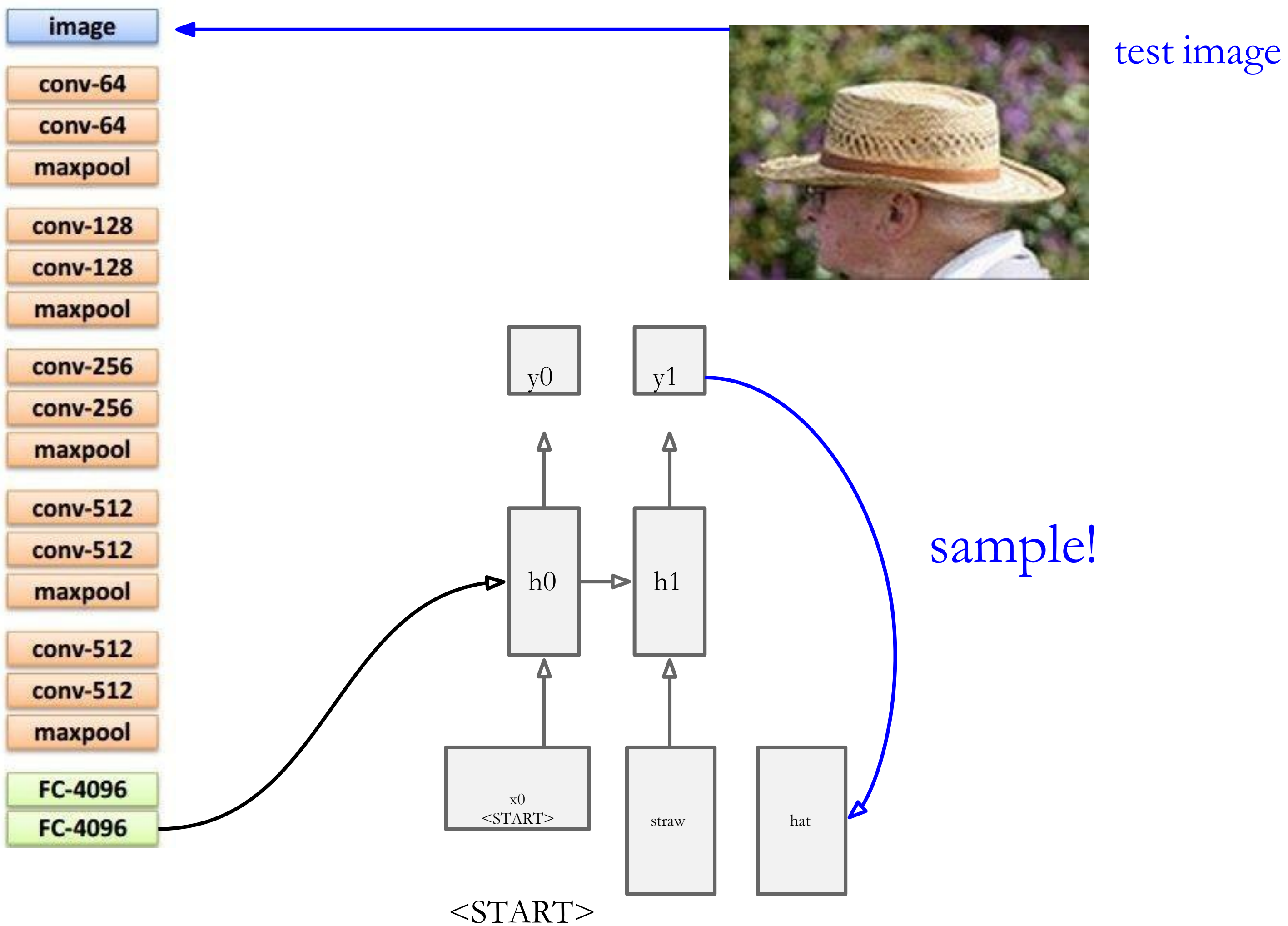
# Image Captioning



# Image Captioning

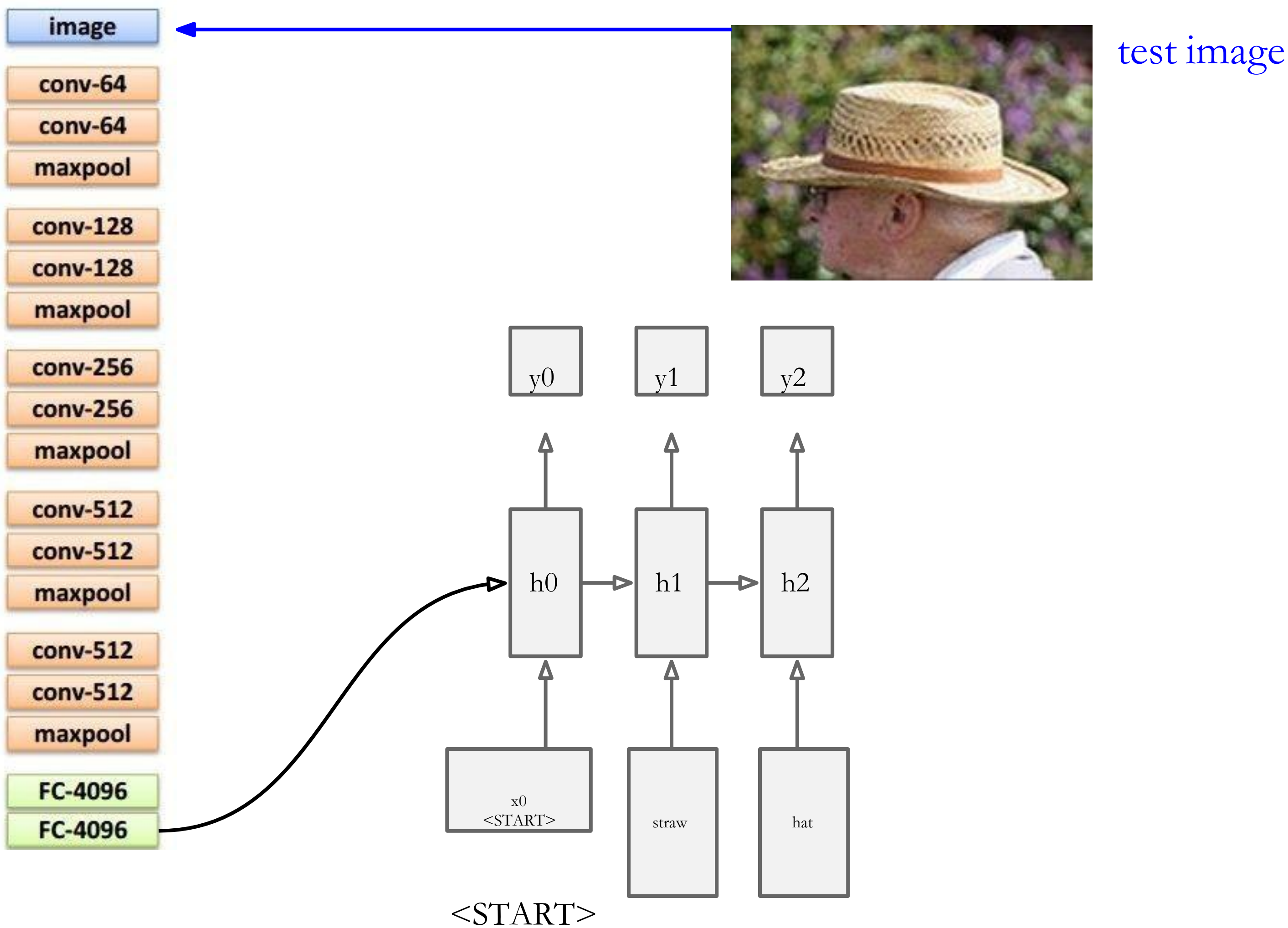


# Image Captioning

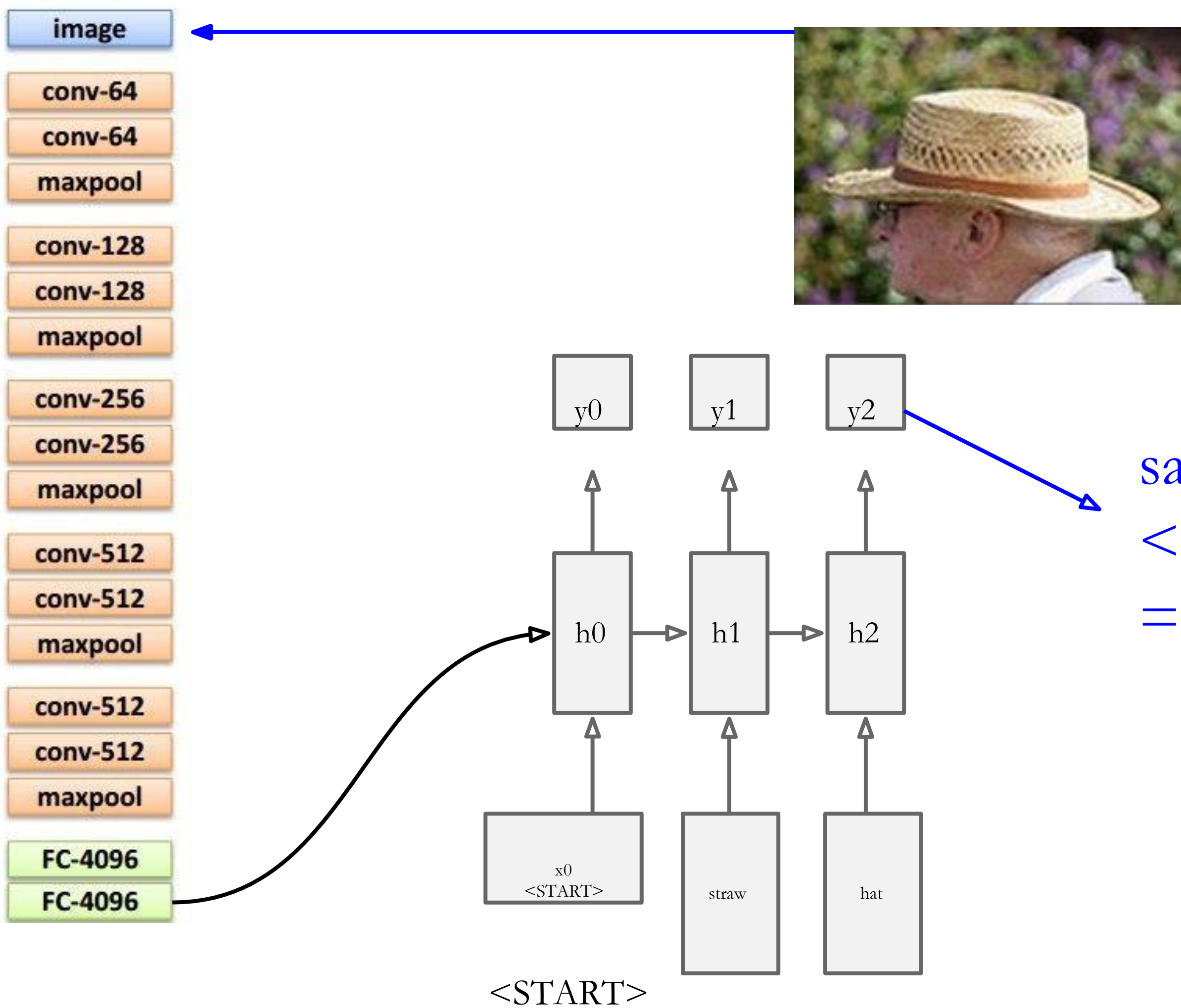



Andrej Karpathy

# Image Captioning



# Image Captioning




test image

Caption generated:
"straw hat"

sample
<END> token
=> finish.

Adapted from Andrej Karpathy

# Image Captioning

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"a young boy is holding a baseball bat."

"a cat is sitting on a couch with a remote control."


"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."

Andrej Karpathy