Recurrent Neural Networks

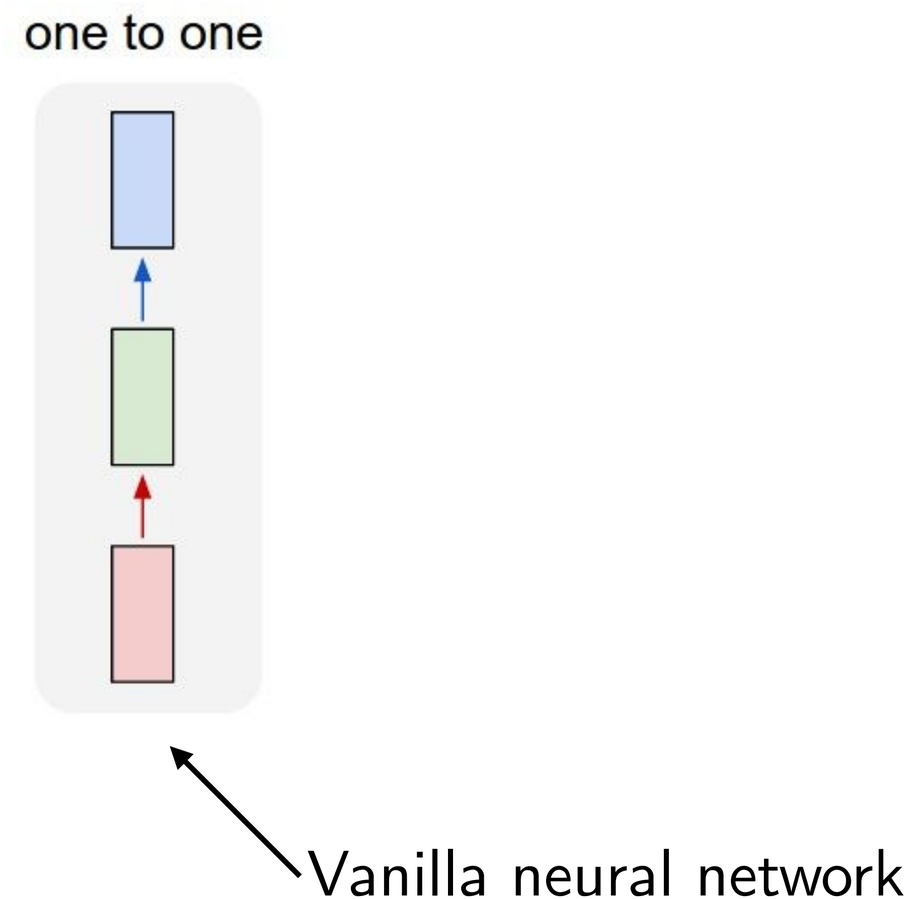
Dongwoo Kim

POSTECH

This slides is available at: https://bit.ly/31gdkWT

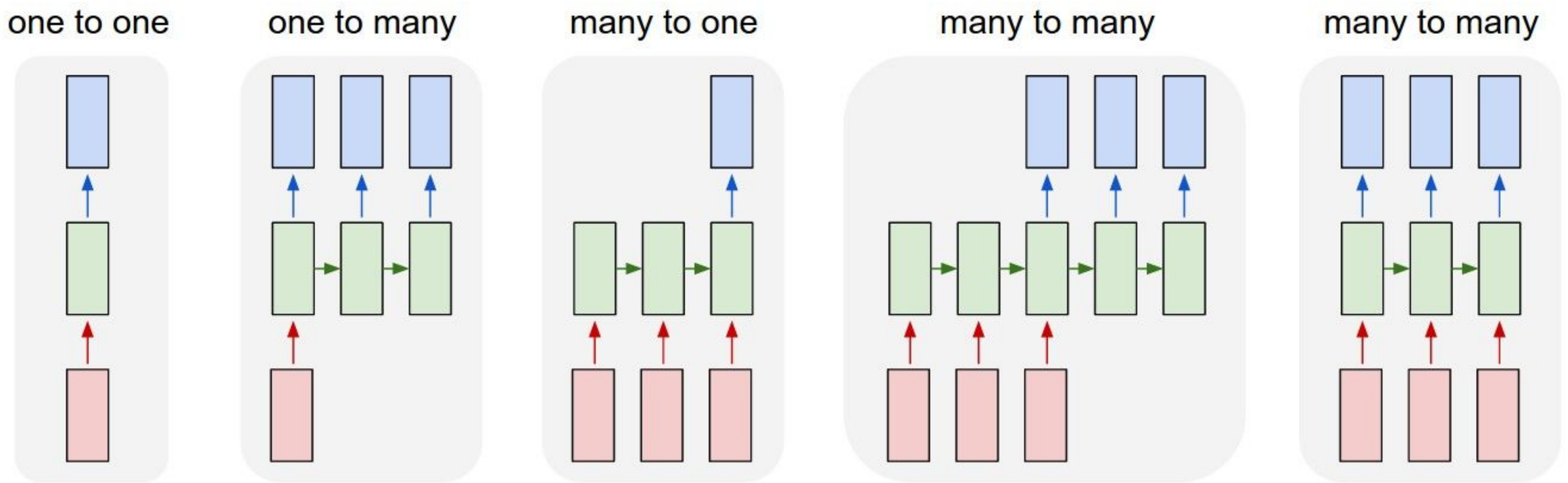
Machine Learning Laboratory
POSTECH

Vanilla Neural Network



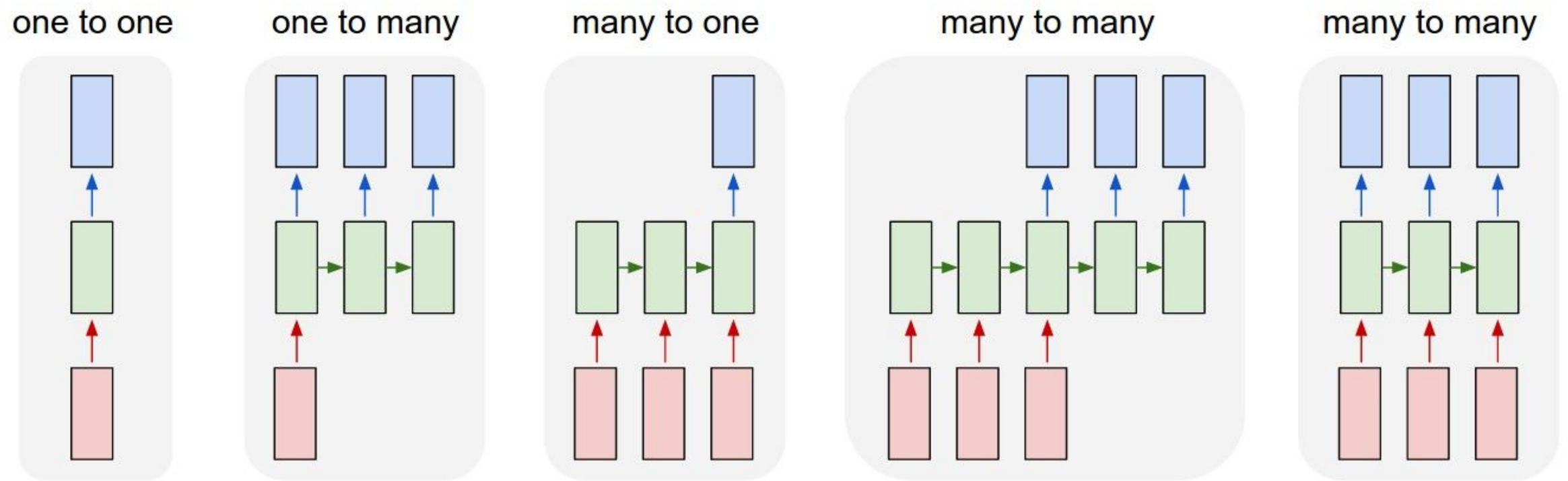
Recurrent Neural Networks: Processing Sequences

Five different types of problems



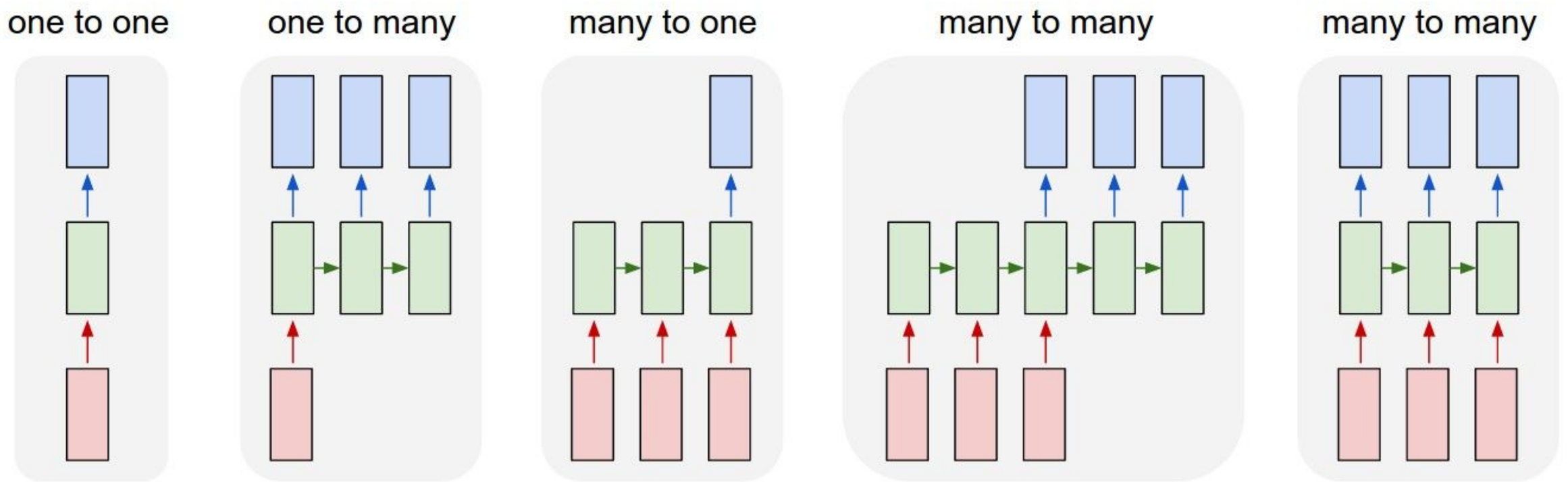
e.g. Image captioning
image → sequence of words

Recurrent Neural Networks: Processing Sequences



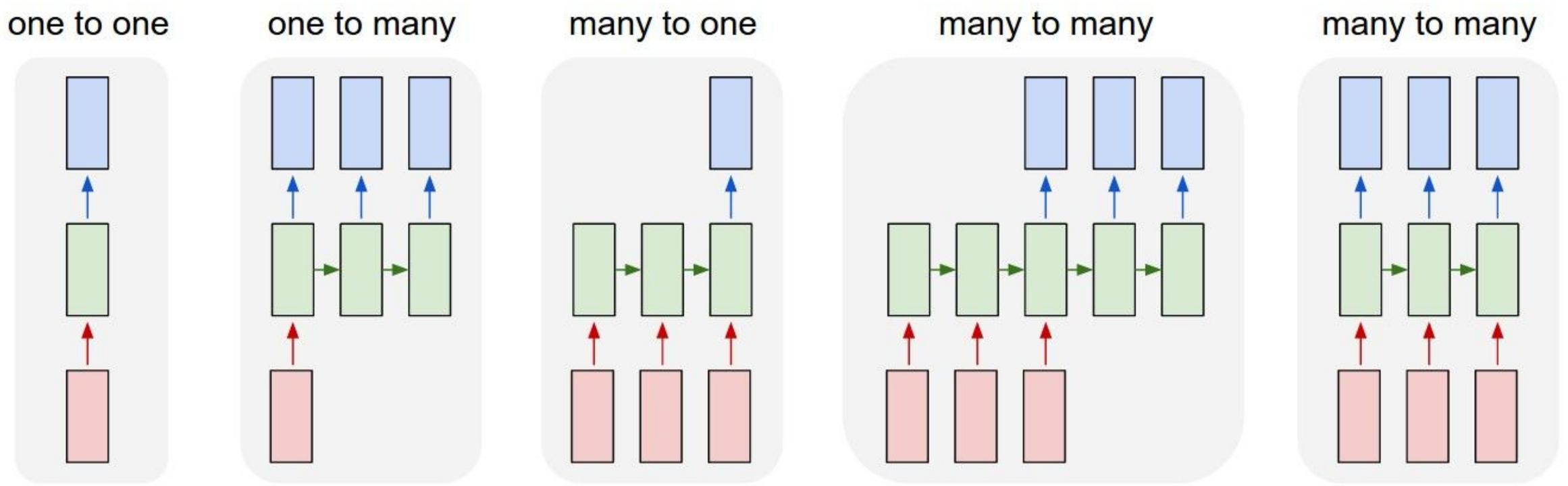
e.g. action prediction
sequence of frames → action

Recurrent Neural Networks: Processing Sequences



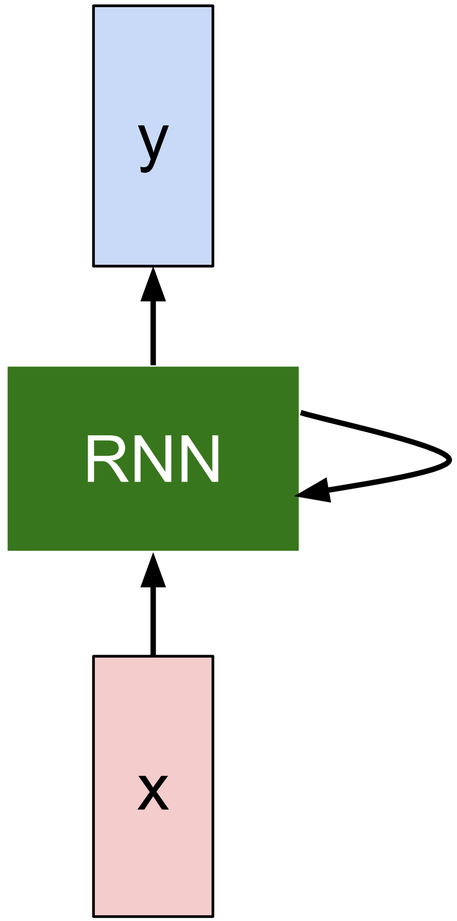
e.g. video captioning
sequence of video frame → caption

Recurrent Neural Networks: Processing Sequences



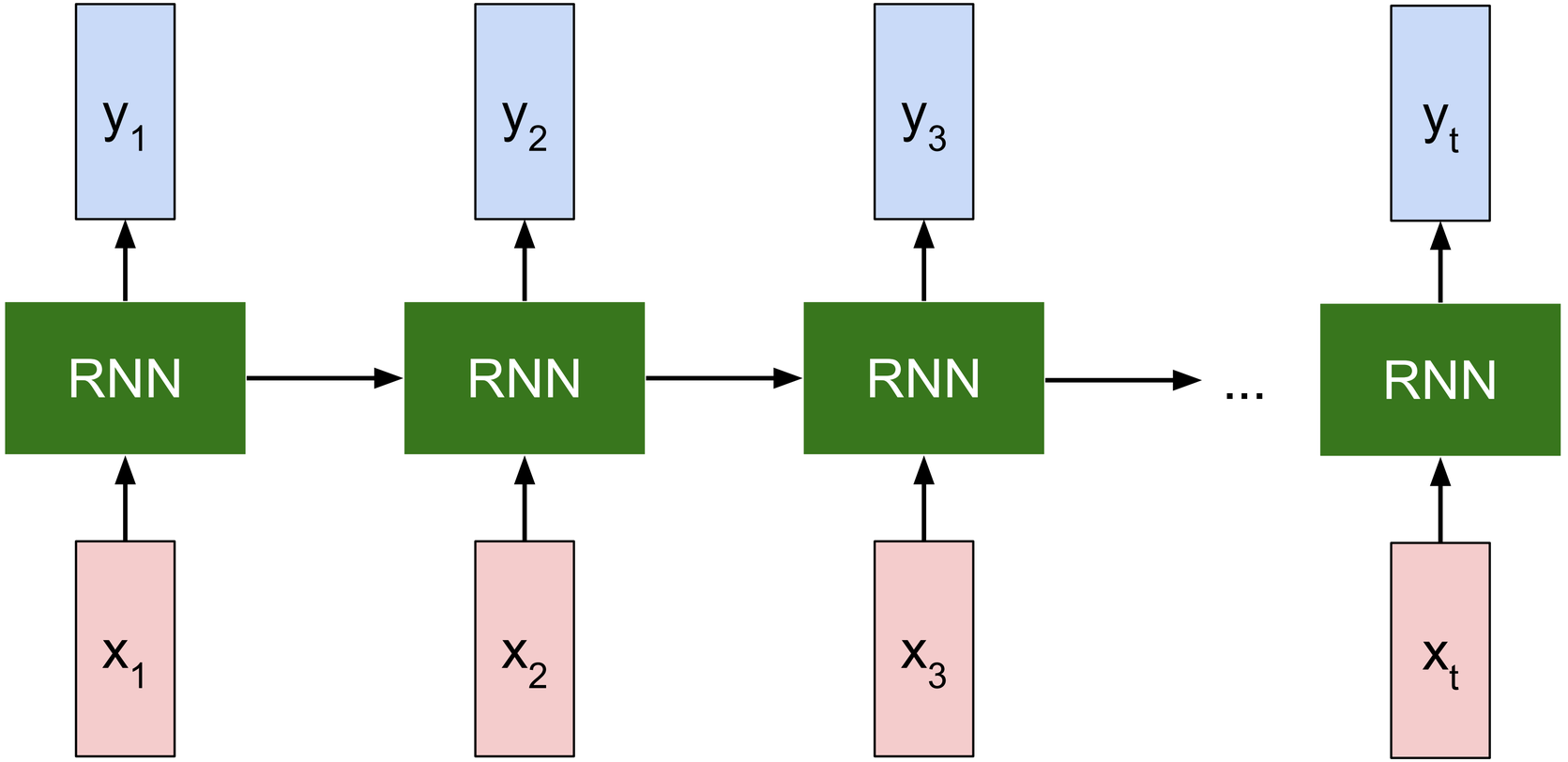
e.g. video classification on frame level

Recurrent Neural Network: The Idea



Key idea: RNNs have an "**internal state**" that is updated as a sequence is processed

Recurrent Neural Network: The Idea



Recurrent Neural Network

- We can process a sequence of vectors x by applying a recurrence formula at every time step.

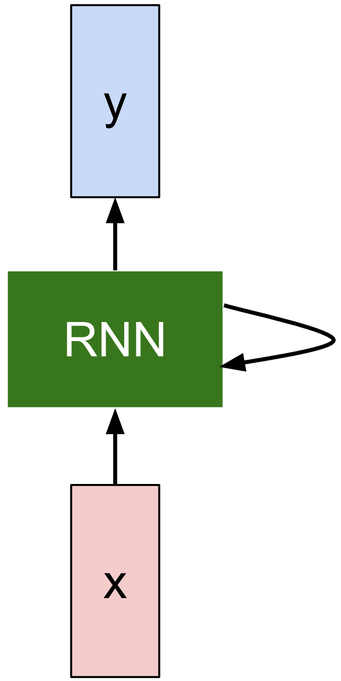
some function with parameter W

input vector at time t

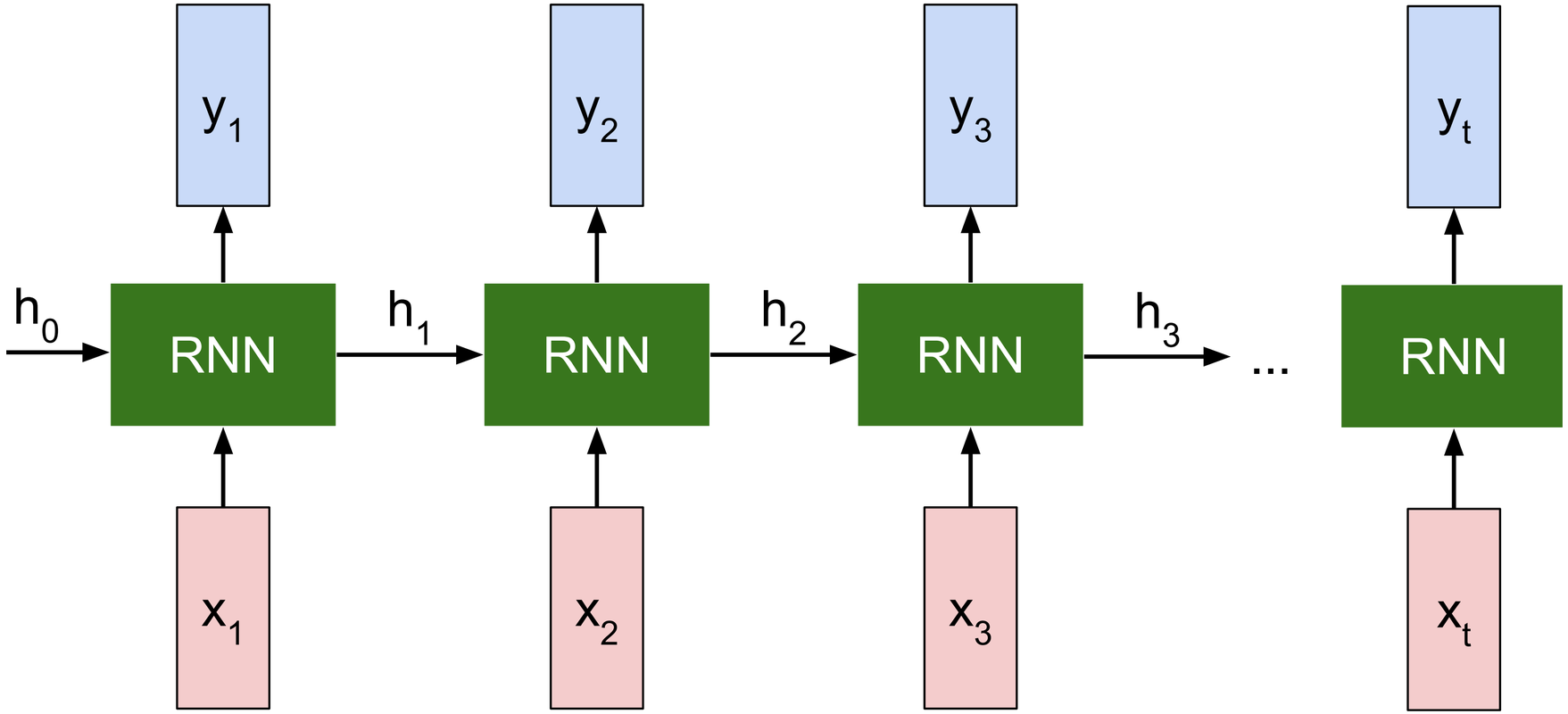
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state



Recurrent Neural Network: The Idea



Recurrent Neural Network

- We can process a sequence of vectors x by applying a recurrence formula at every time step.

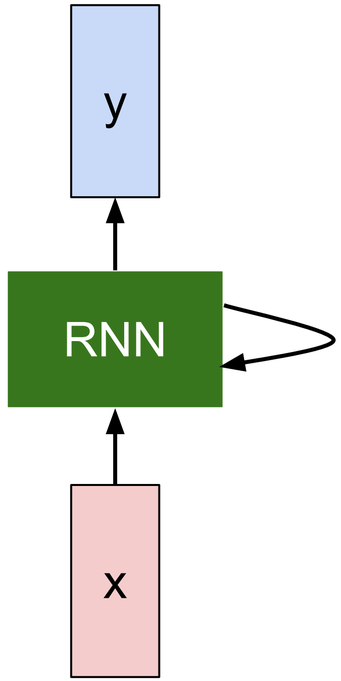
some function with parameter W

input vector at time t

$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state



Notice: the same function and the same set of parameters are used at every time step.

Simple RNN

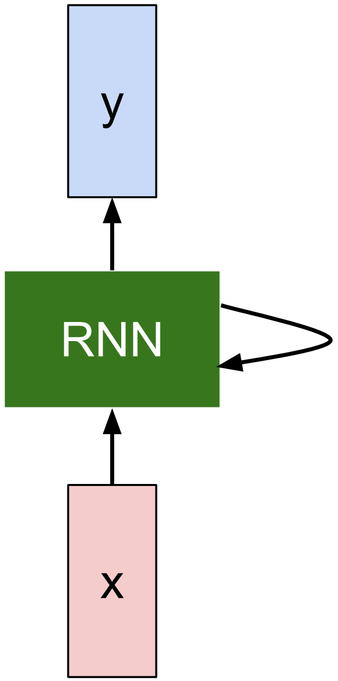
- The state consists of a single "hidden" vector h

$$h_t = f_W(h_{t-1}, x_t)$$

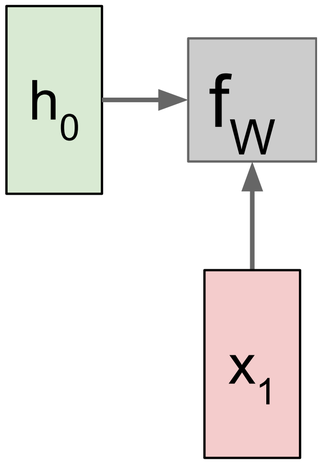
$$\downarrow$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

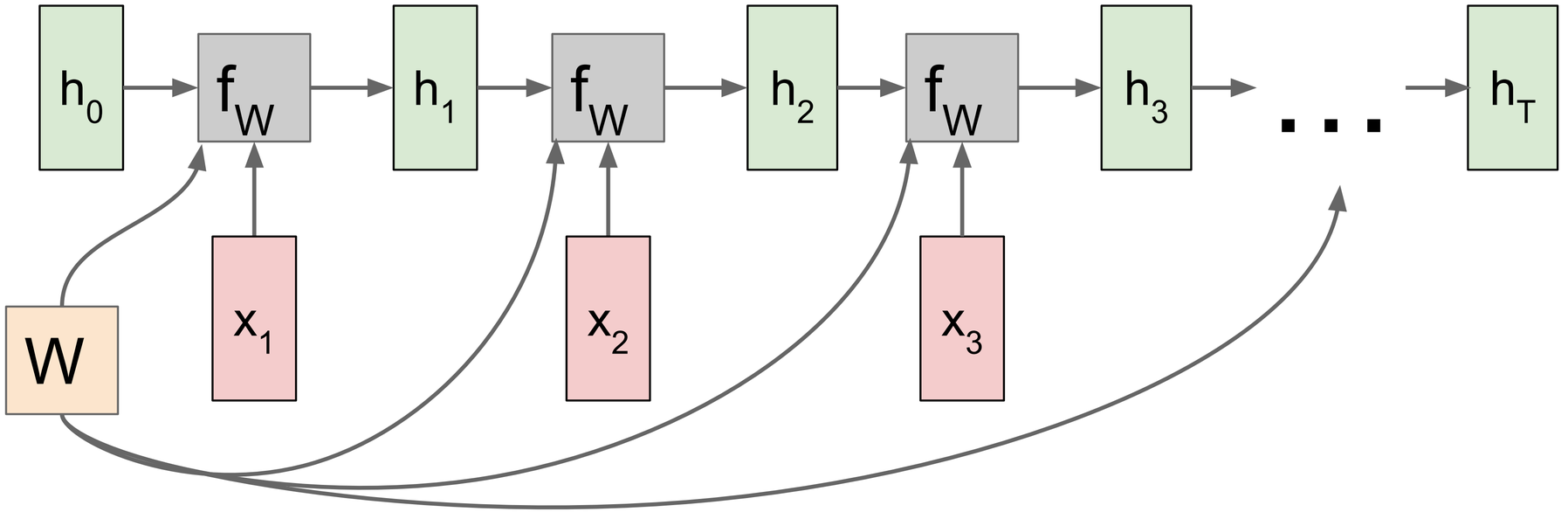


RNN: Computational Graph

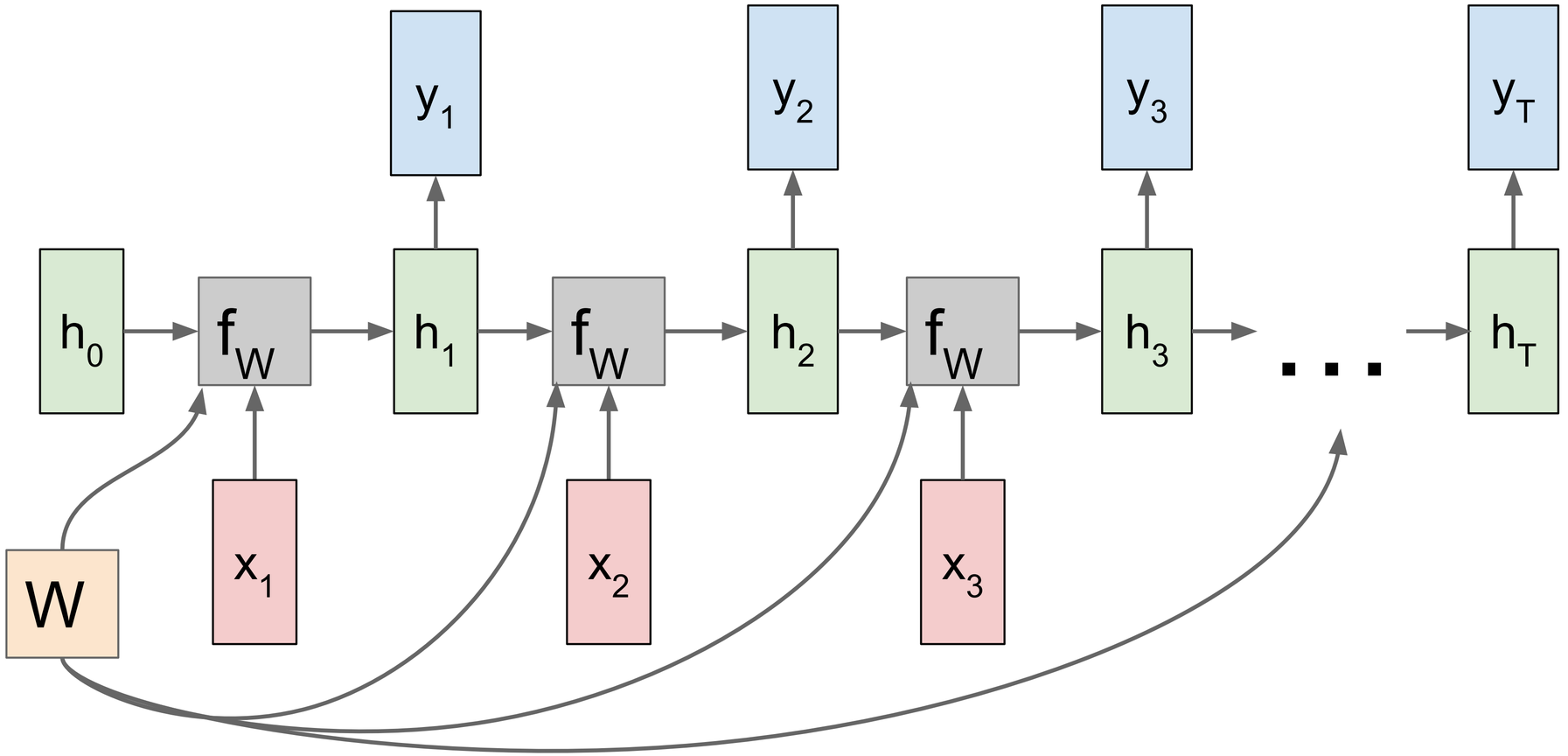


RNN: Shared Parameter

Re-use the same weight matrix at every time-step

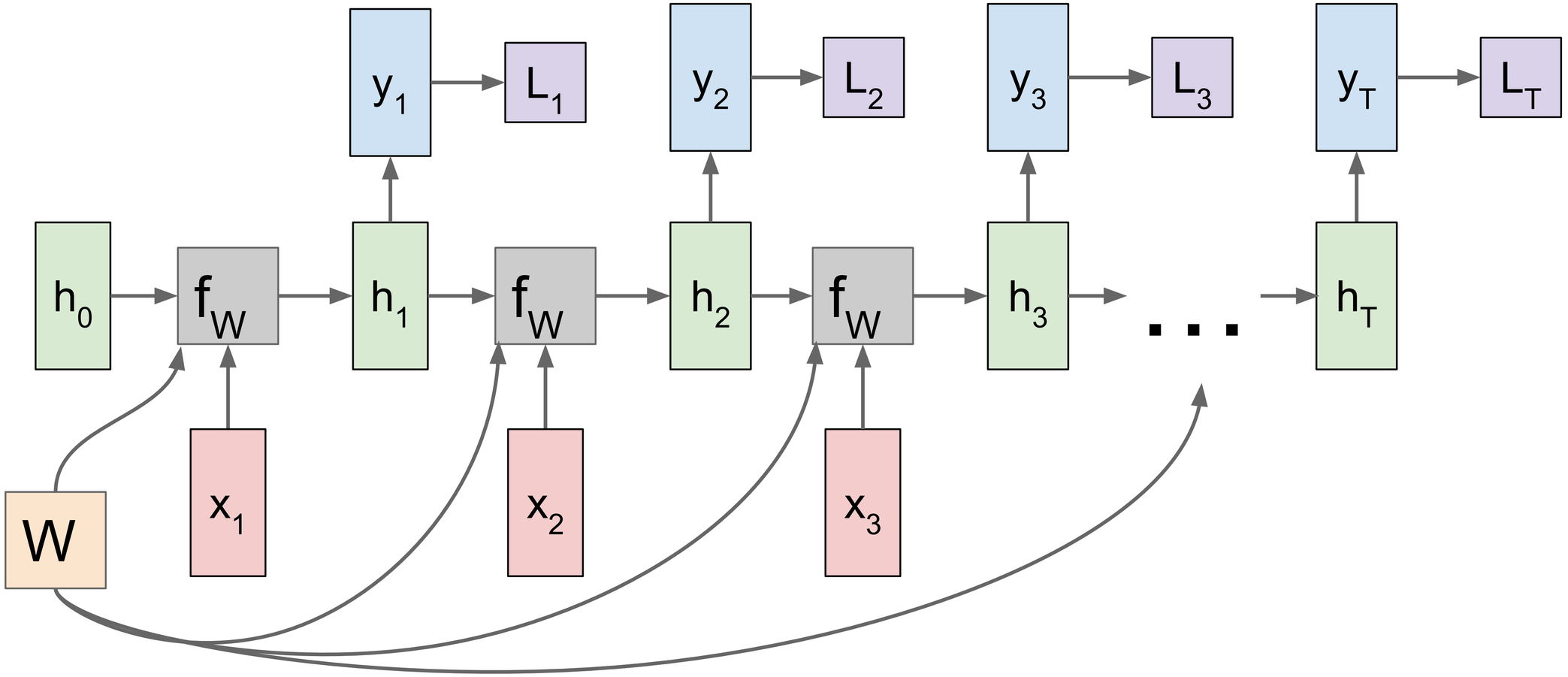


RNN: Many to Many

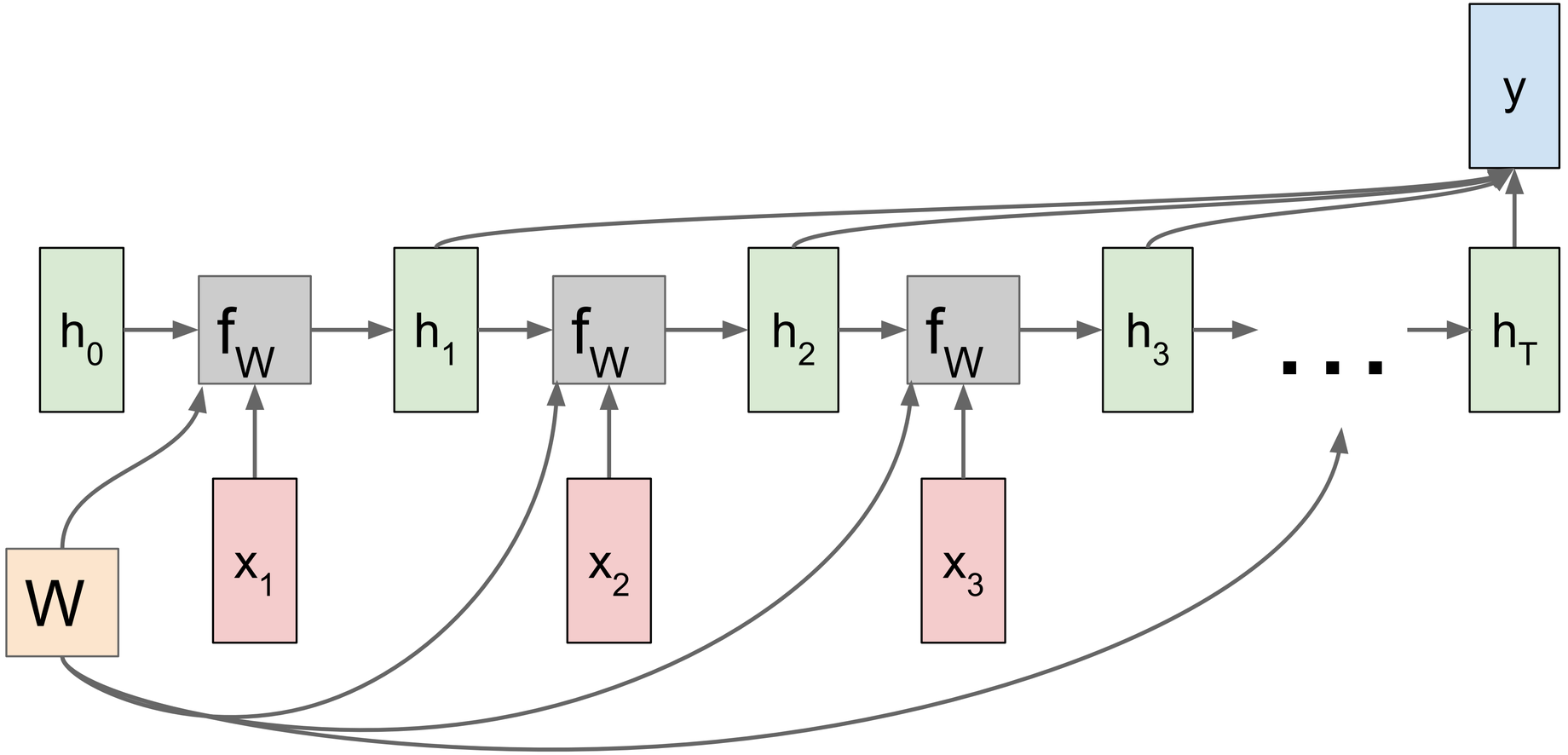


RNN: Many to Many: Sequence of Losses

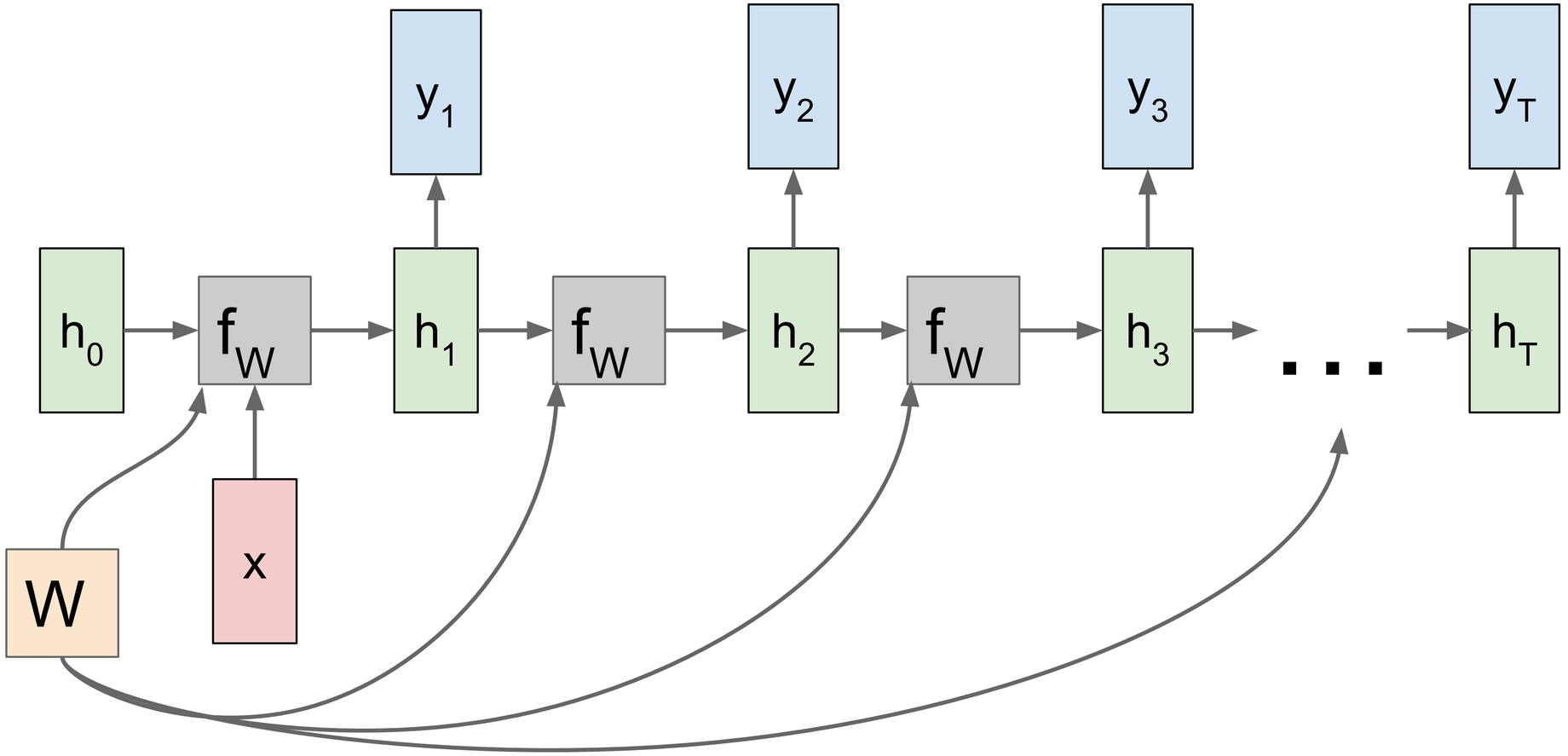
Total loss: sum of individual loss $\sum_{t=1}^{T} L_t$



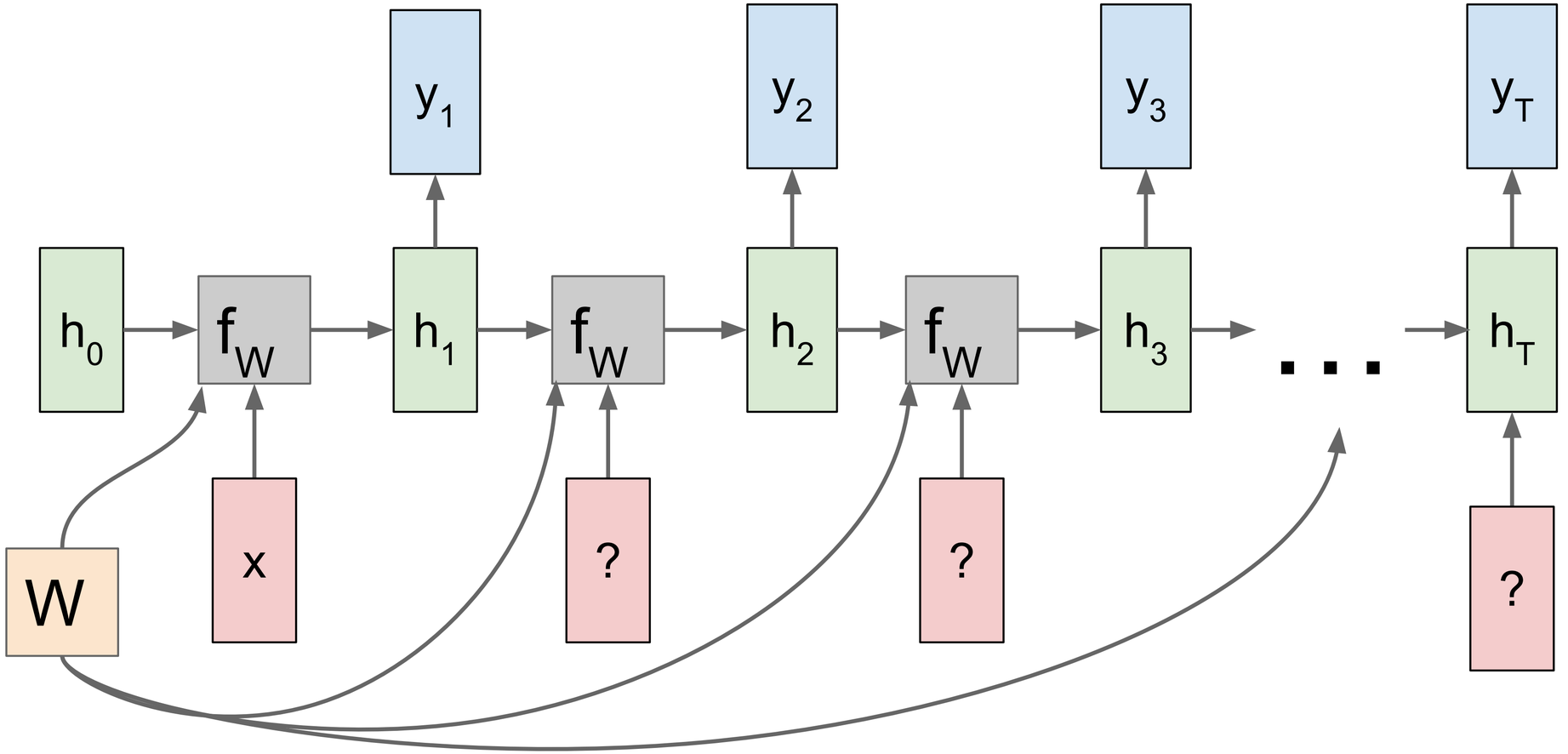
RNN: Many to One



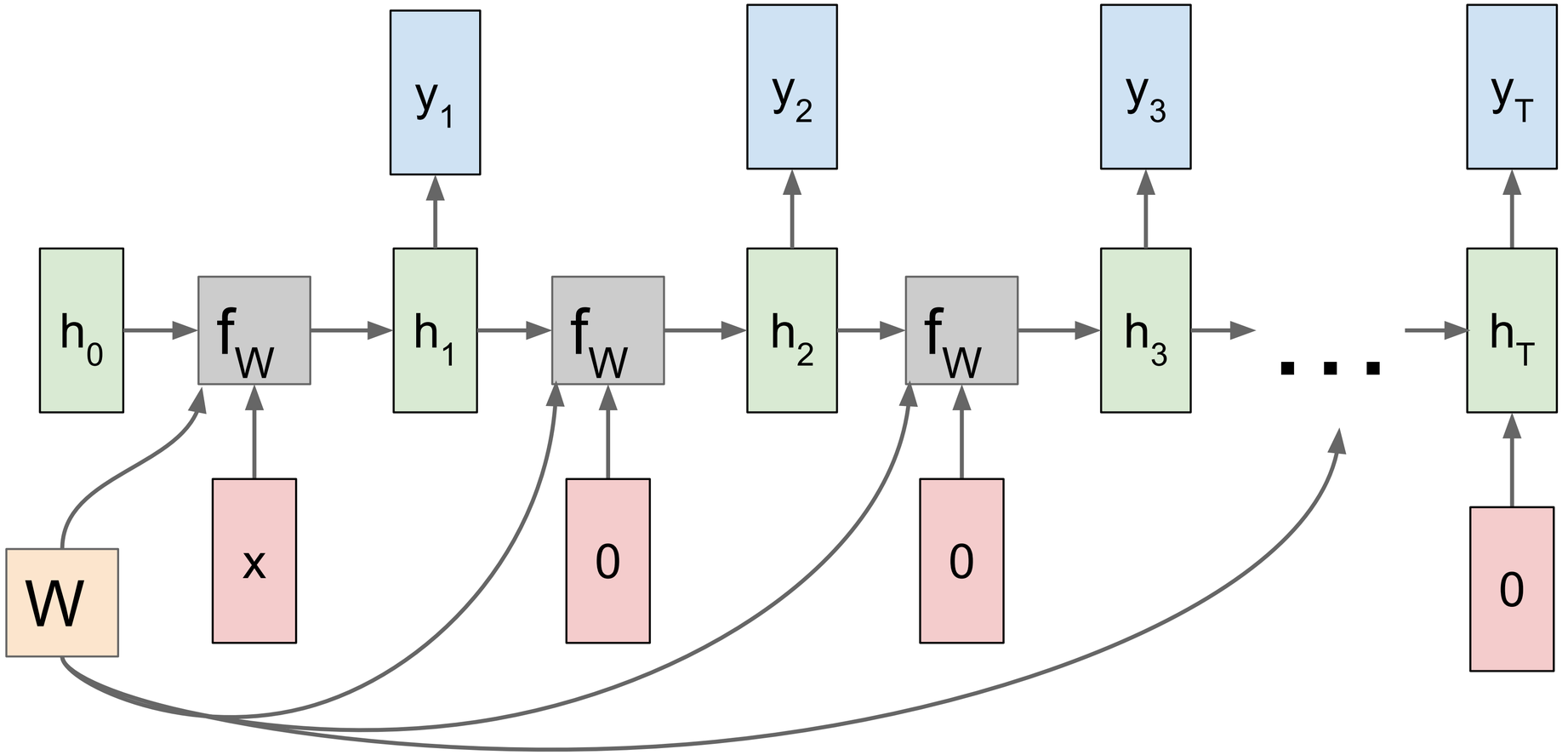
RNN: One to Many



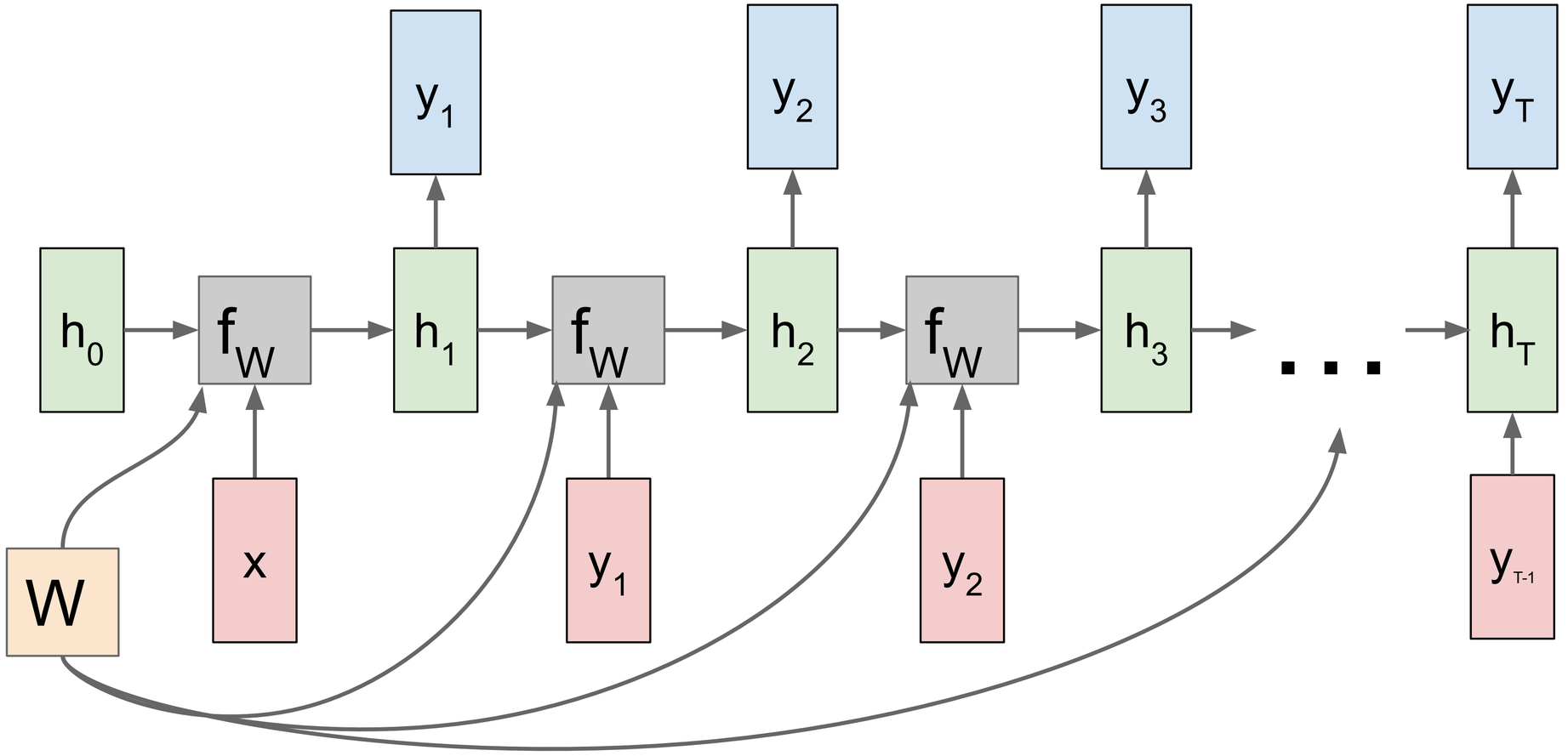
RNN: One to Many



RNN: One to Many



RNN: One to Many

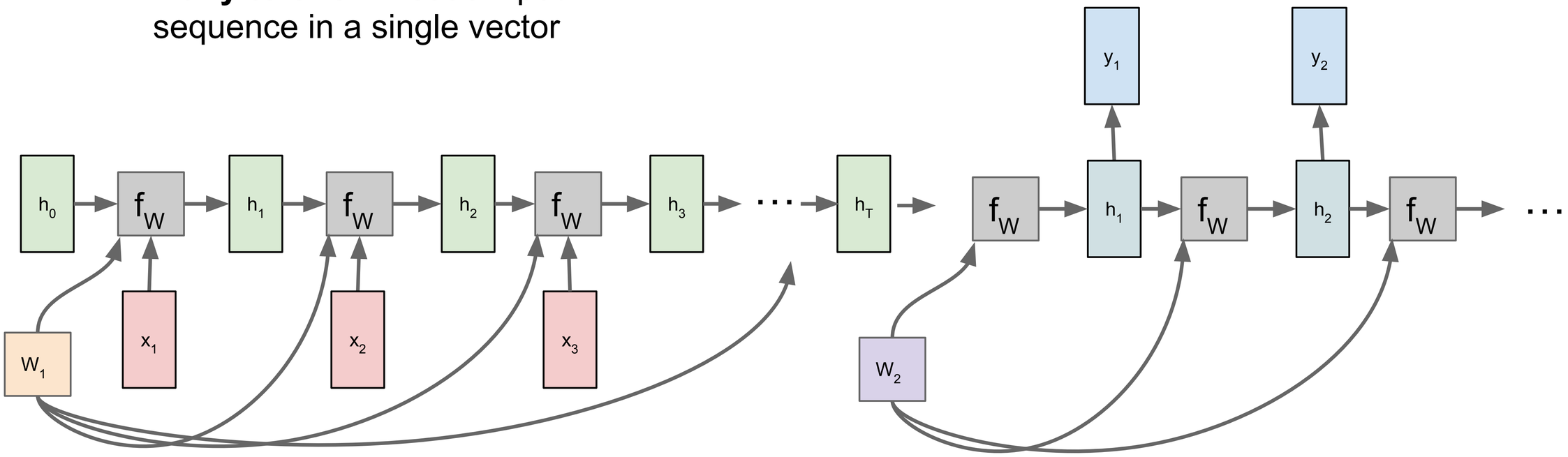


Sequence to Sequence: Many to One + One to Many

e.g. Machine translation

Many to one: Encode input sequence in a single vector

One to many: Produce output sequence from single input vector

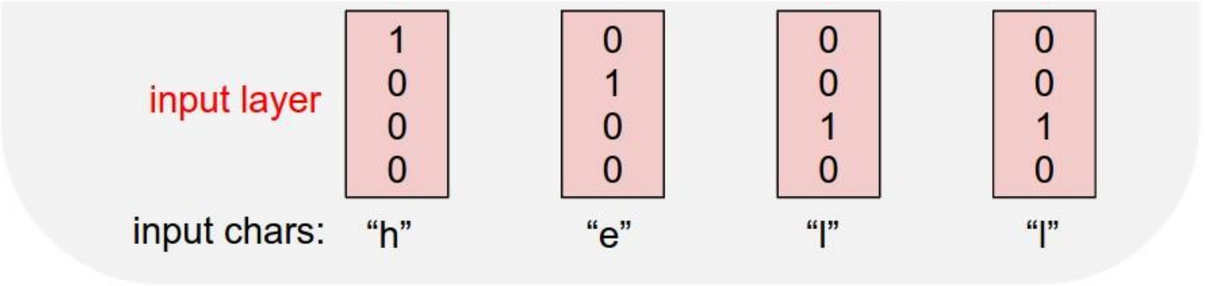


Example: Character-level Language Model

- Vocabulary
  - [h, e, l, o]
- Example training sequence:
  - “hello”

Example: Character-level Language Model

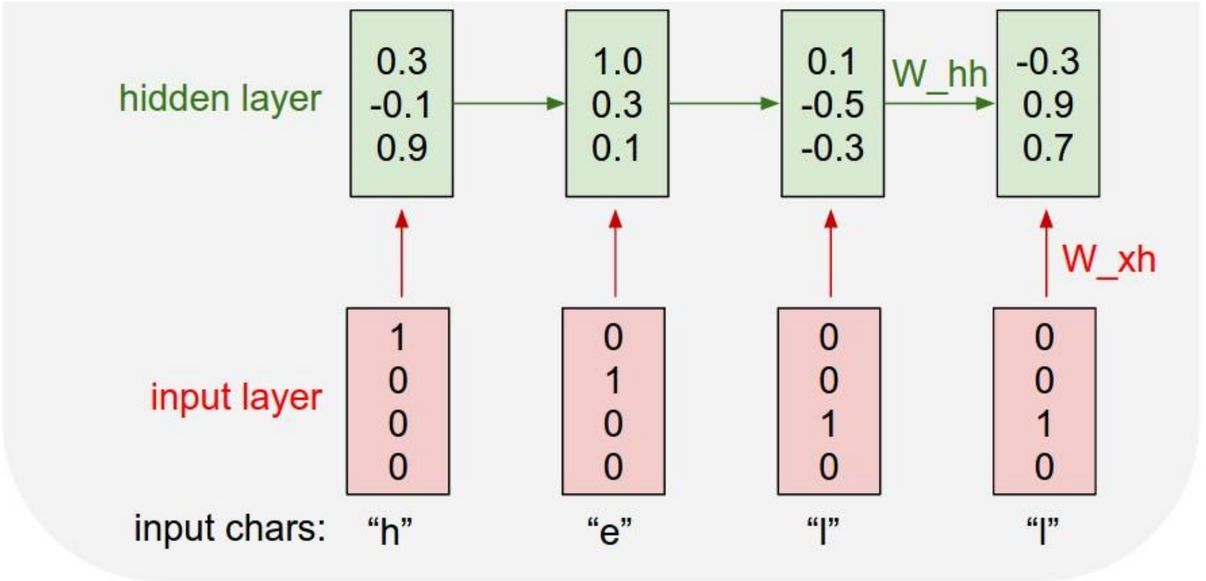
- Vocabulary
  - [h, e, l, o]
- Example training sequence:
  - “hello”
- Many to one



Example: Character-level Language Model

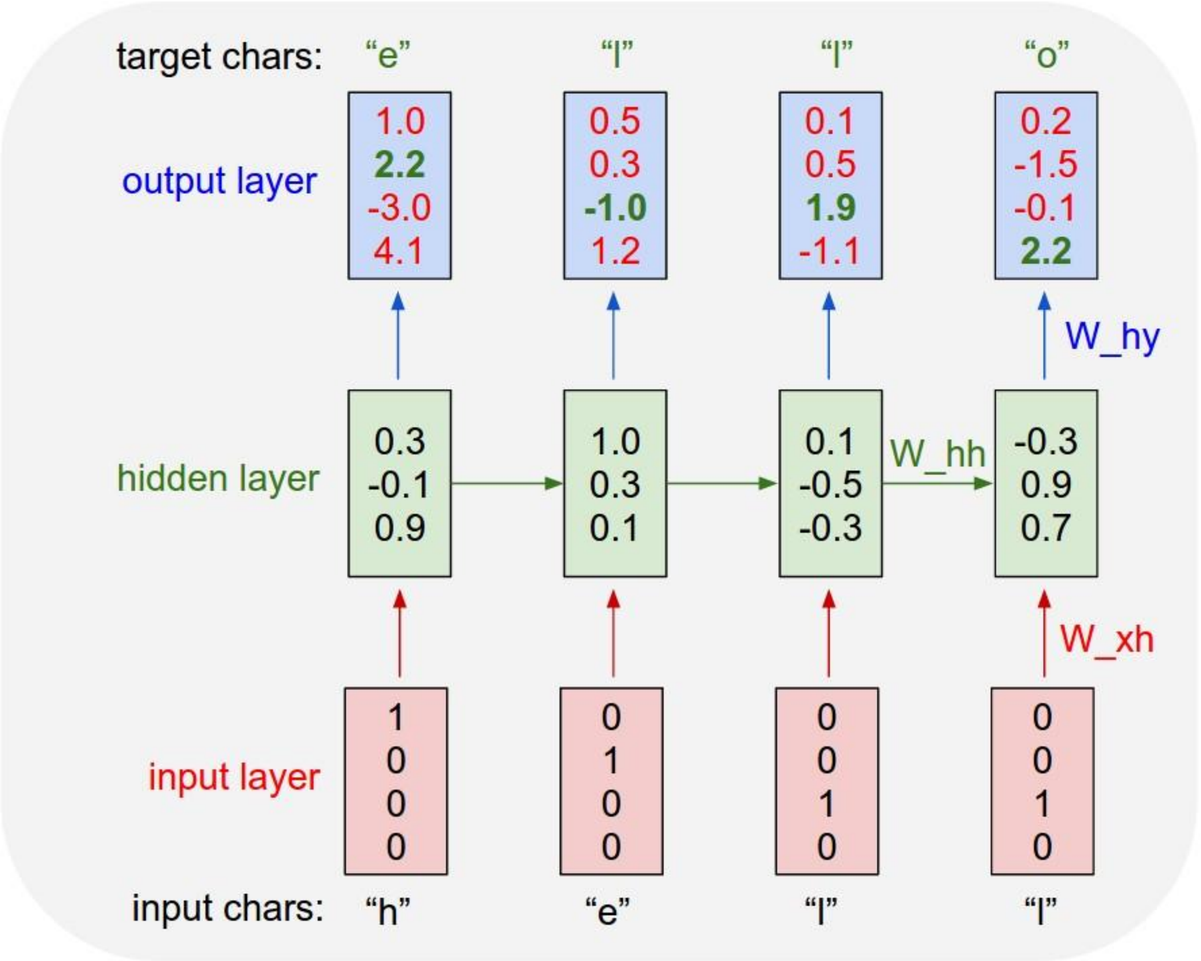
- Vocabulary
  - [h, e, l, o]
- Example training sequence:
  - “hello”
- Many to one

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



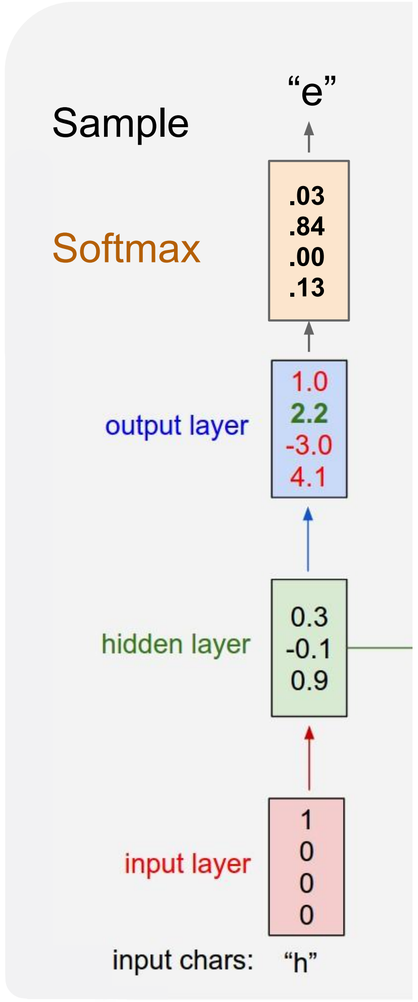
Example: Character-level Language Model

- Vocabulary
  - [h, e, l, o]
- Example training sequence:
  - “hello”
- Many to one



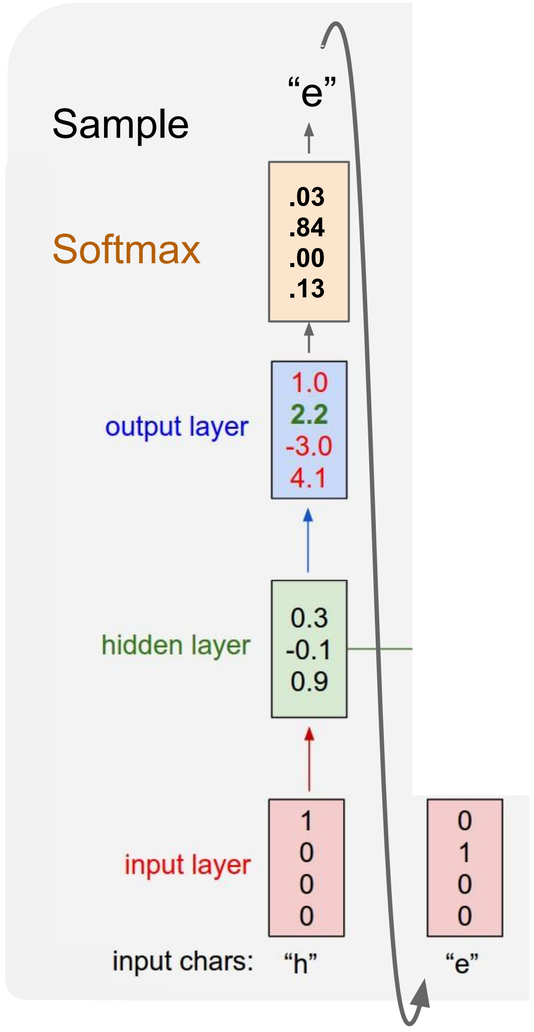
Example: Character-level Language Model

- Vocabulary
  - [h, e, l, o]
- Example training sequence:
  - “hello”
- One to many



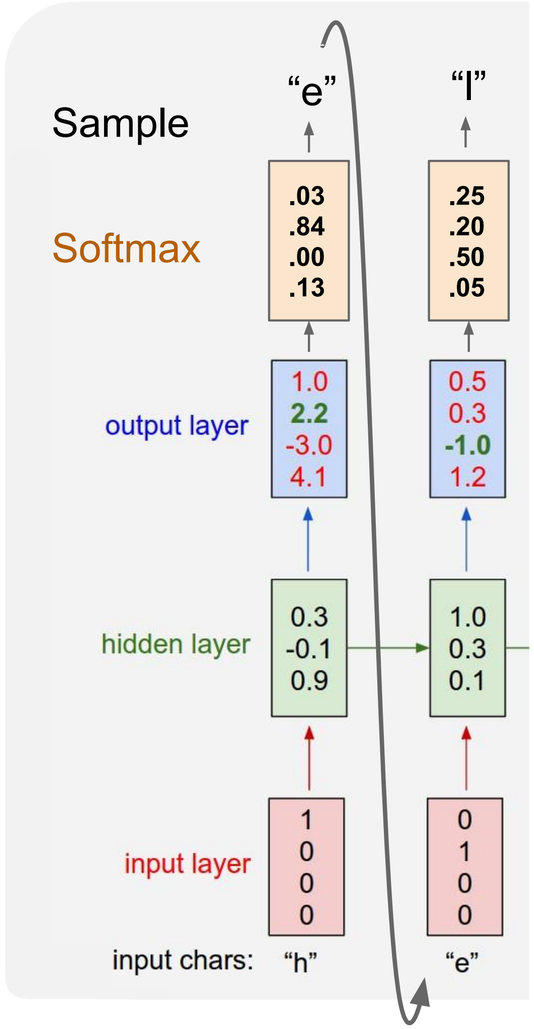
Example: Character-level Language Model

- Vocabulary
  - [h, e, l, o]
- Example training sequence:
  - “hello”
- One to many



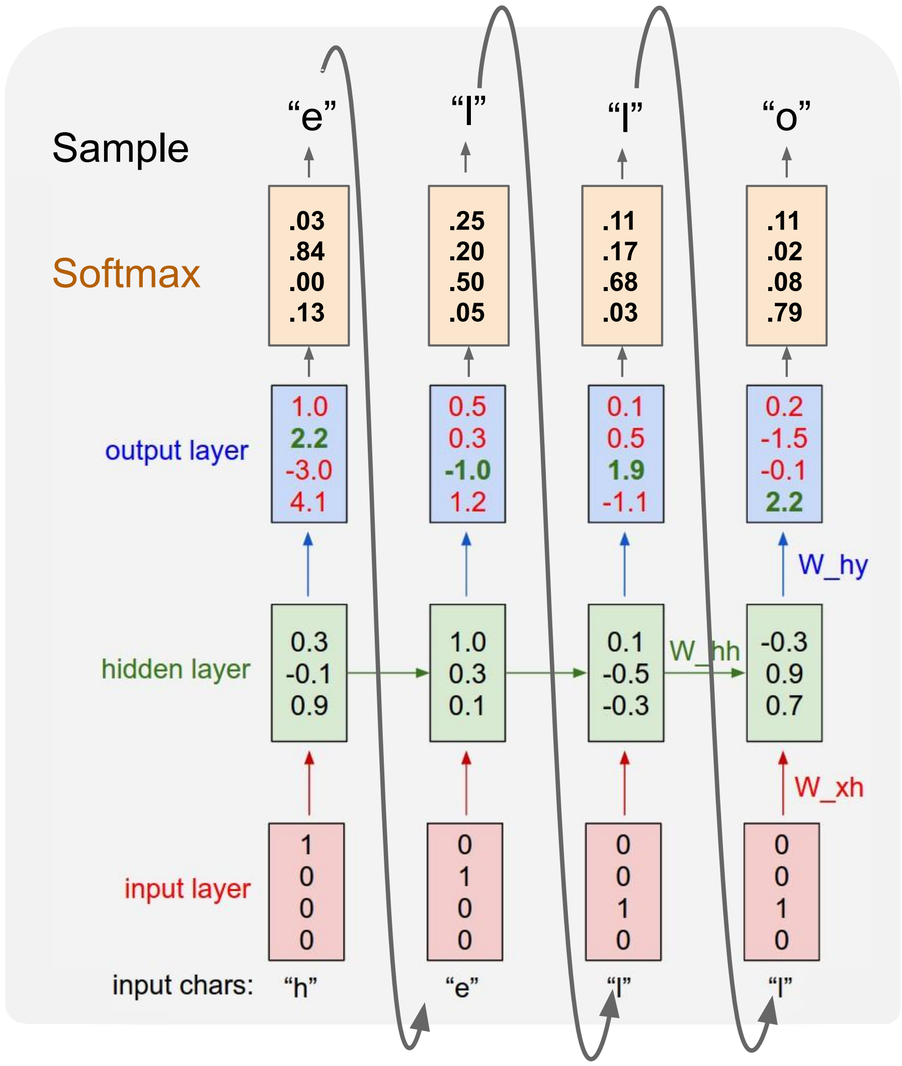
Example: Character-level Language Model

- Vocabulary
  - [h, e, l, o]
- Example training sequence:
  - “hello”
- One to many



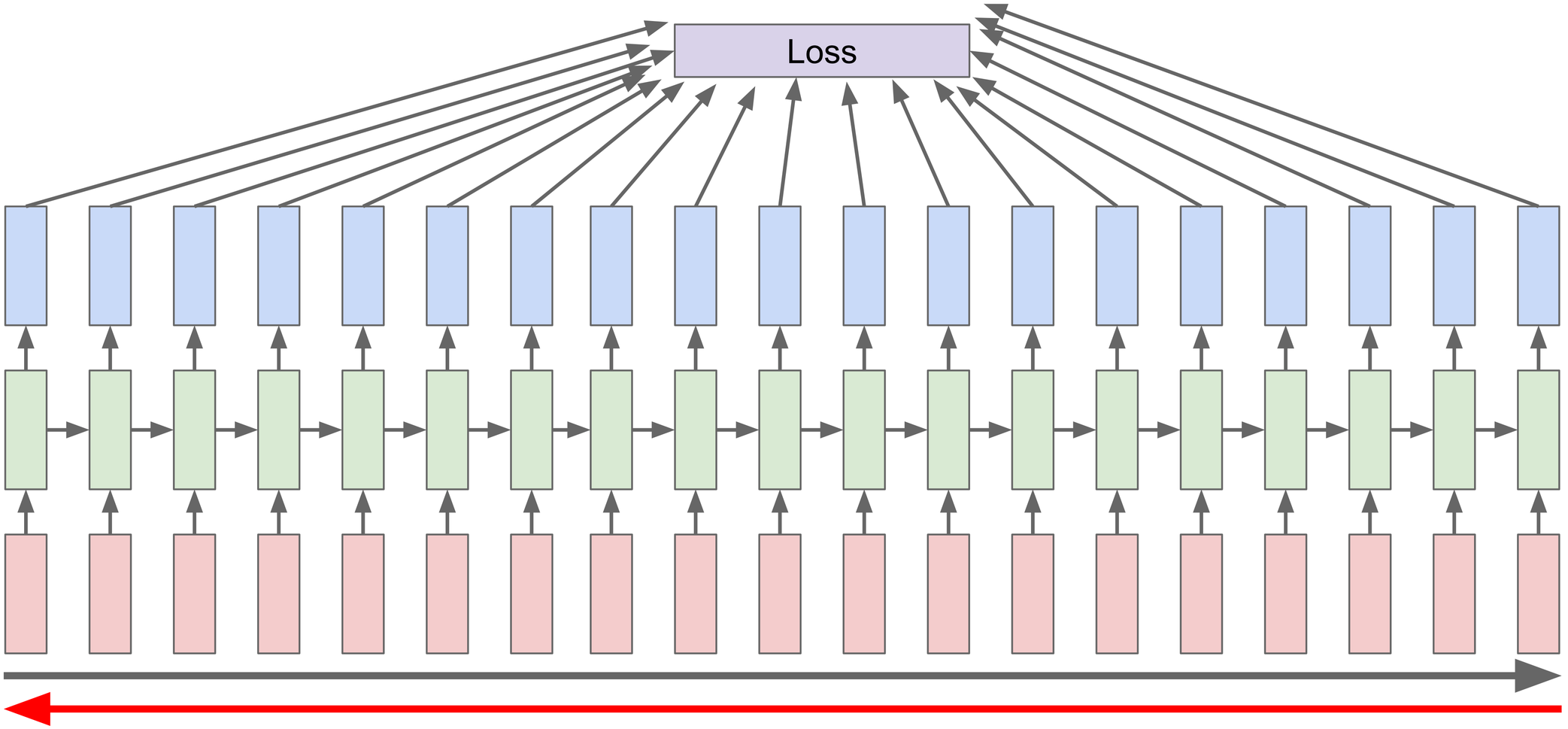
Example: Character-level Language Model

- Vocabulary
  - [h, e, l, o]
- Example training sequence:
  - “hello”
- One to many



Backpropagation Through Time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

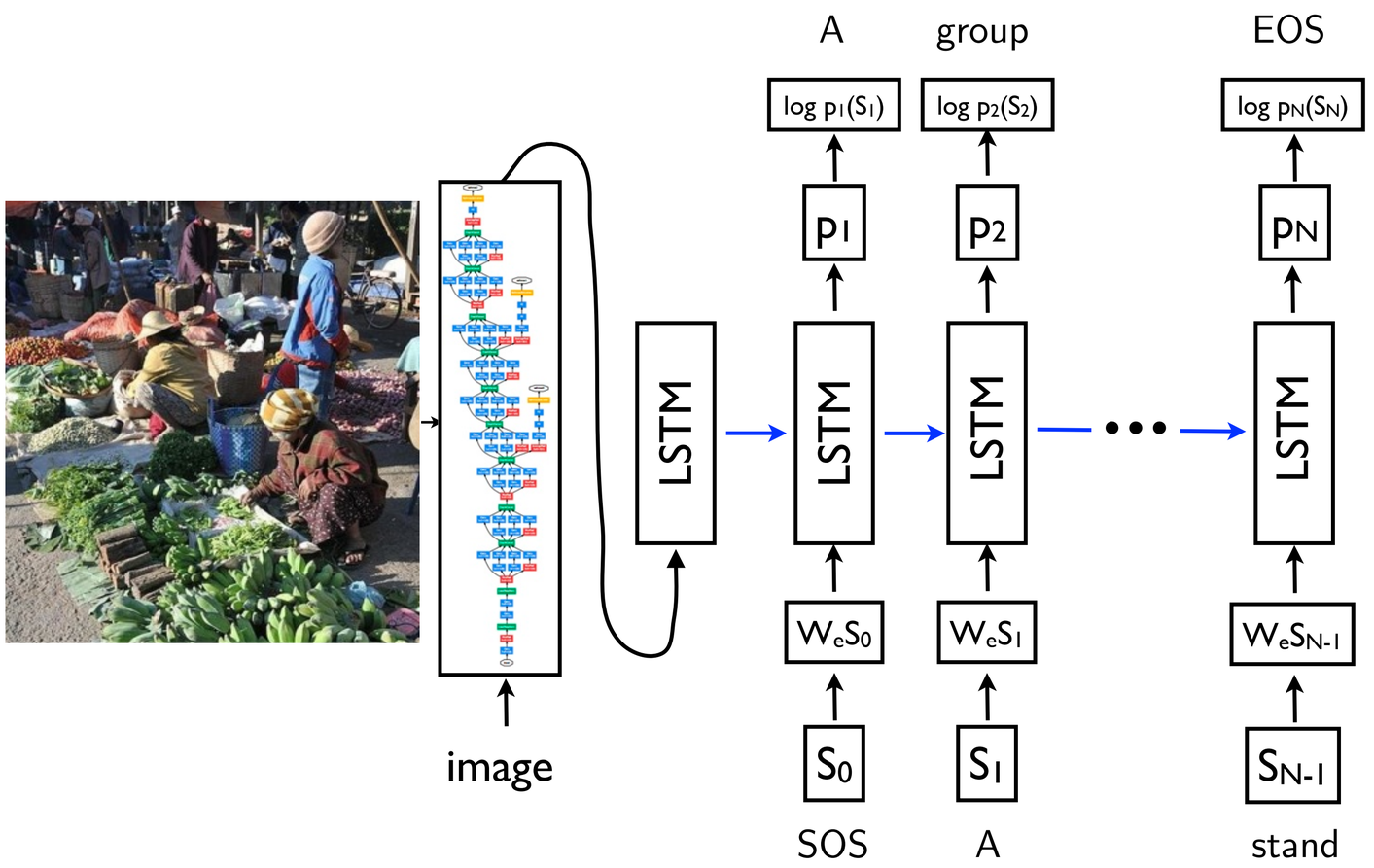


RNN Tradeoffs

- RNN Advantages:
  - Can process *any length* input
  - Computation for step t can (in theory) use information from many steps back
  - Model size doesn't increase for longer input
  - Same weights applied on every timestep, so there is symmetry in how inputs are processed.
- RNN Disadvantages:
  - Recurrent computation is slow
  - In practice, difficult to access information from many steps back

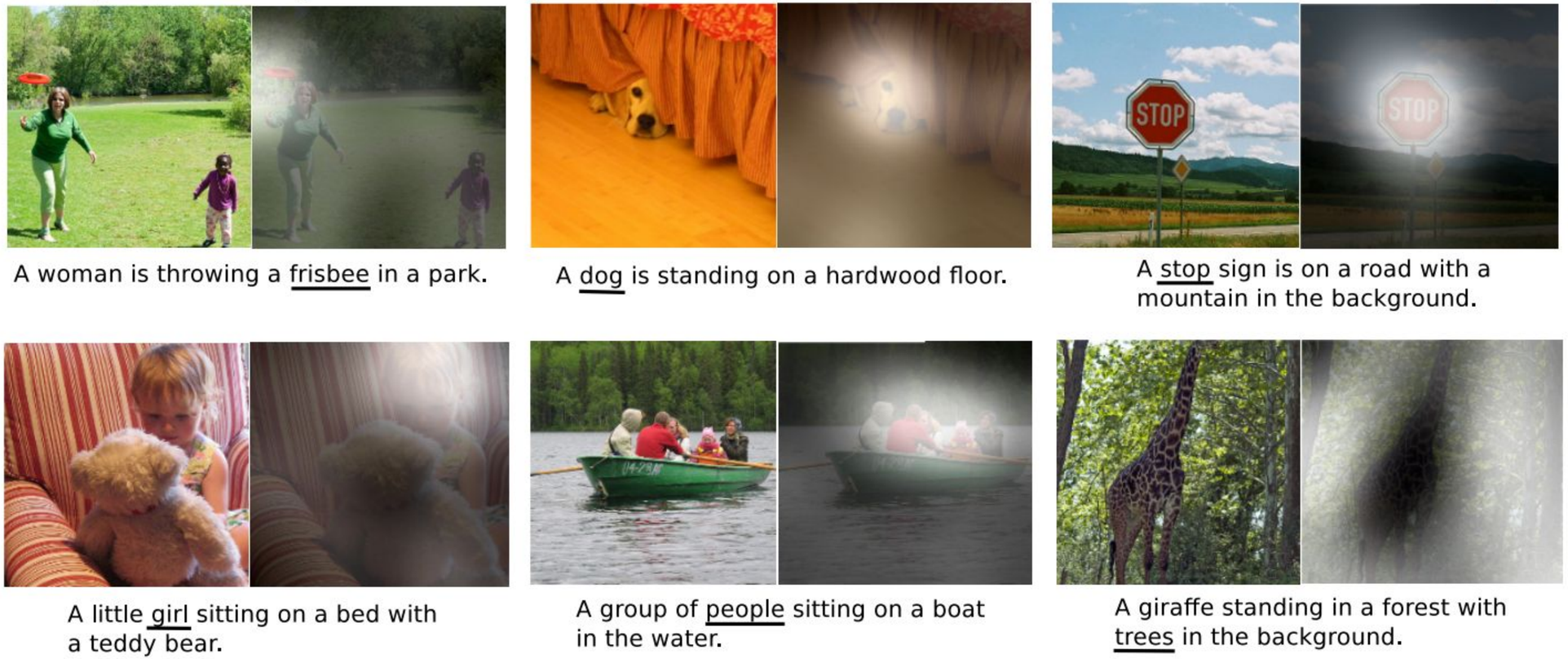
Applications

Image Captioning



A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

Caption Generation: Good Examples



Xu et al, “Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Visual Question and Answering (VQA)

Q: What endangered animal is featured on the truck?

A: **A bald eagle.**
A: A sparrow.
A: A humming bird.
A: A raven.

Q: Where will the driver go if turning right?

A: **Onto 24 ¾ Rd.**
A: Onto 25 ¾ Rd.
A: Onto 23 ¾ Rd.
A: Onto Main Street.

Q: When was the picture taken?

A: **During a wedding.**
A: During a bar mitzvah.
A: During a funeral.
A: During a Sunday church service

Q: Who is under the umbrella?

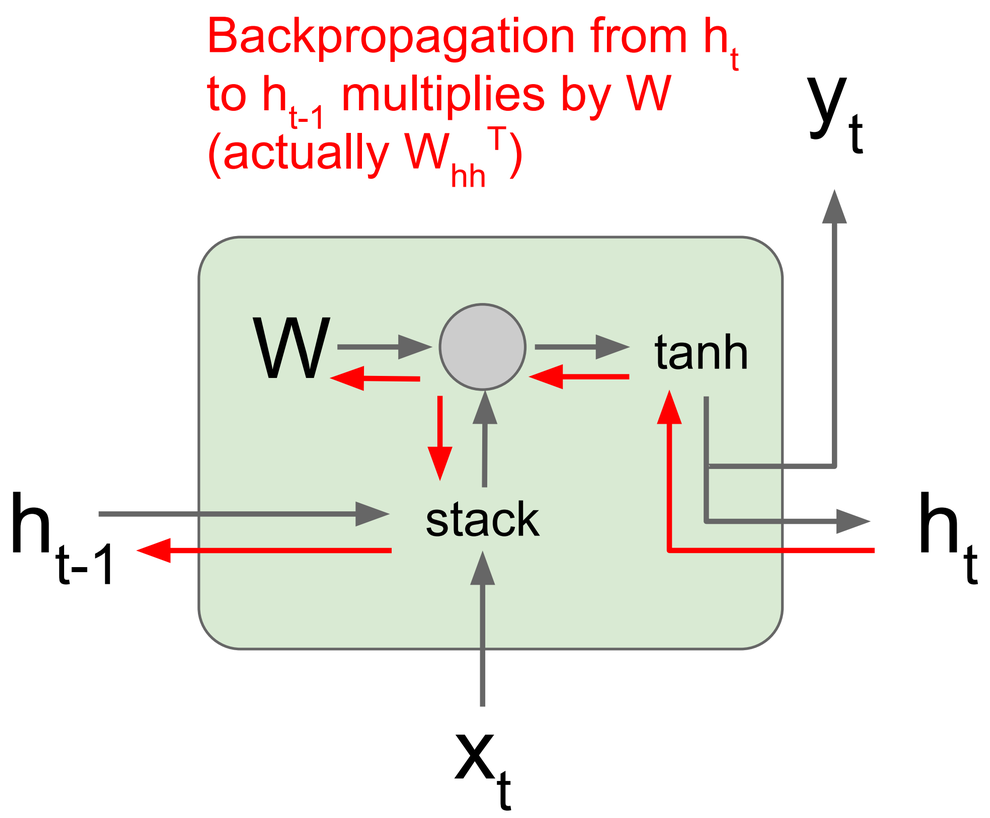
A: **Two women.**
A: A child.
A: An old man.
A: A husband and a wife.

Agrawal et al, “VQA: Visual Question Answering”, ICCV 2015
Zhu et al, “Visual 7W: Grounded Question Answering in Images”, CVPR 2016

Visual Dialog: Conversation about Images

Das et al, “Visual Dialog”, CVPR 2017

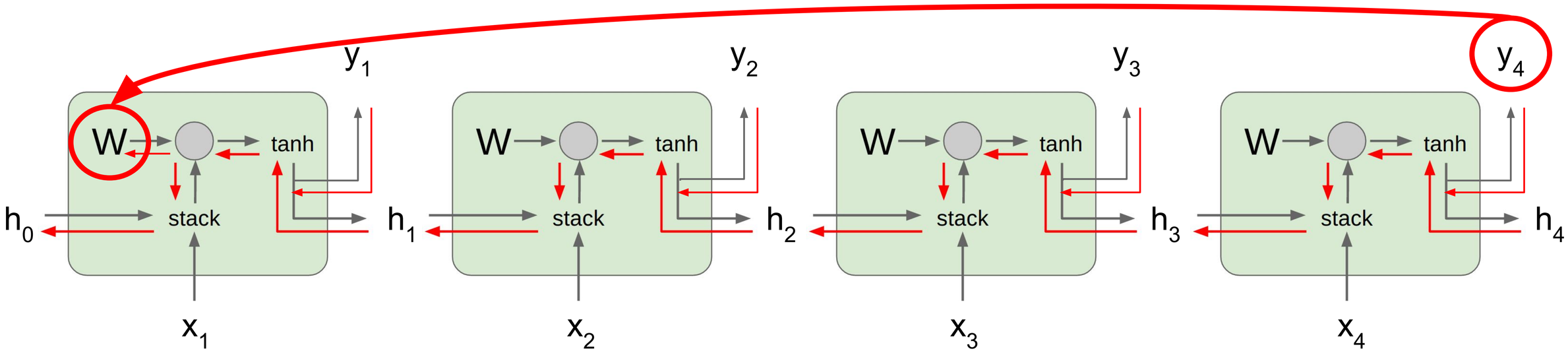
Vanilla RNN Gradient Flow



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

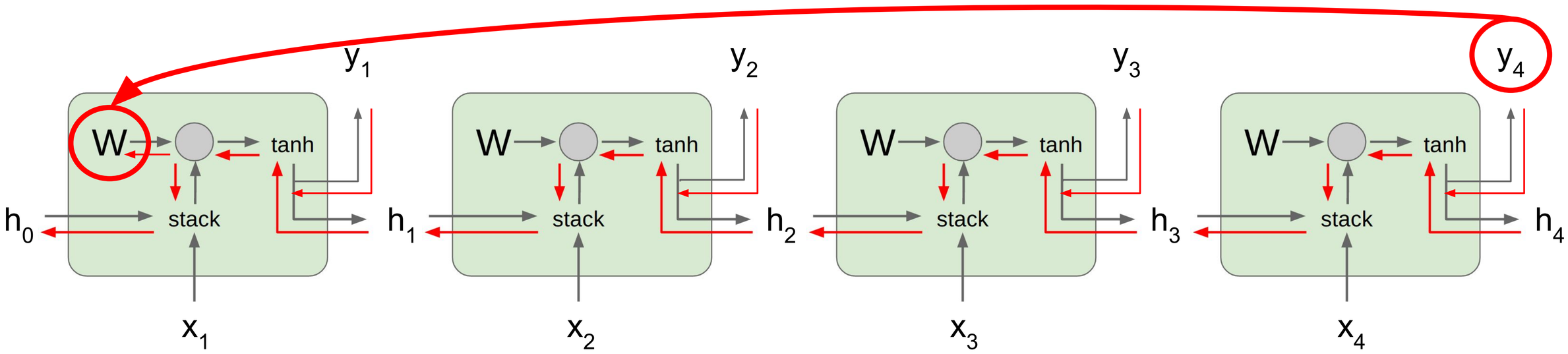
Gradients over Multiple Steps



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}$$

Gradients over Multiple Steps

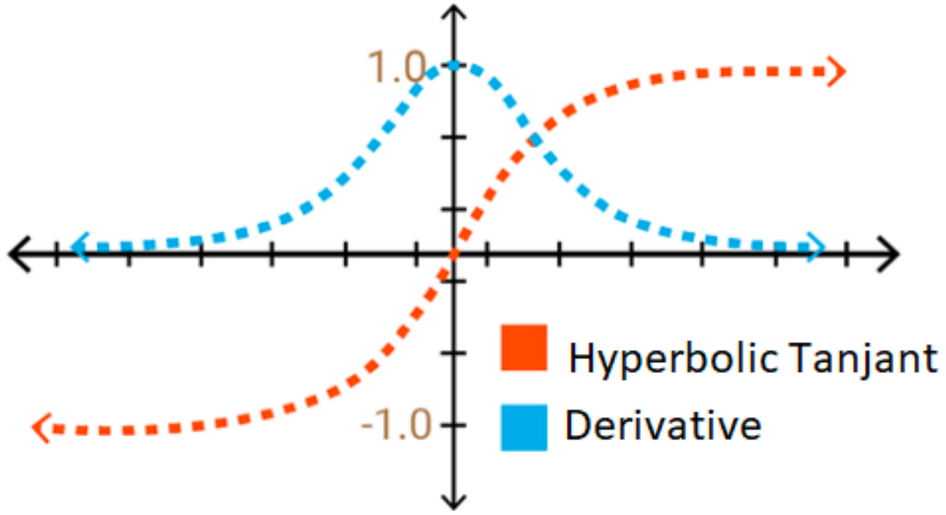


$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh} h_{t-1} + W_{xh} x_t) W_{hh}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}$$

Gradients over Multiple Steps



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Almost always < 1

Vanishing gradients

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \boxed{tanh'\left(W_{hh} h_{t-1} + W_{xh} x_t\right)} \right) W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Long Short Term Memory(LSTM)

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

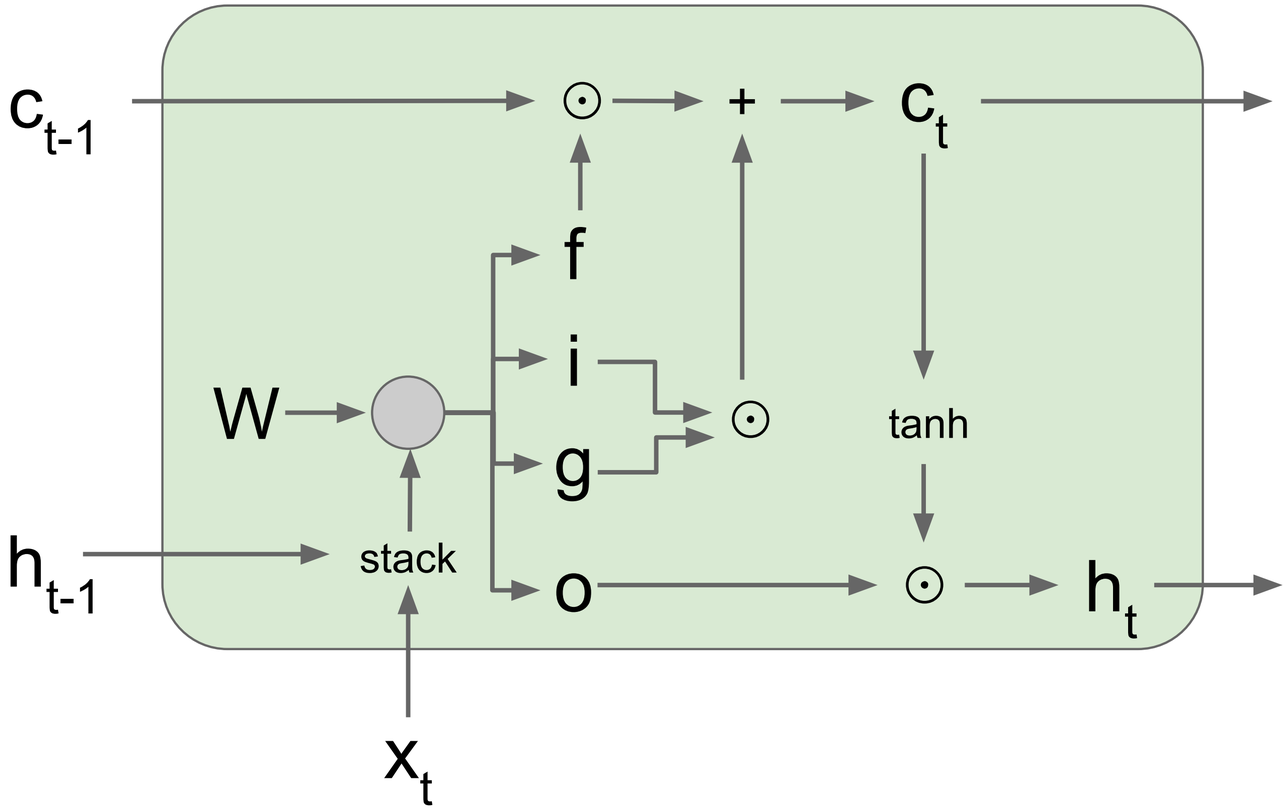
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

LSTM - Interpretation

i: *input gate*, whether to write to cell
f: *forget gate*, whether to erase cell
o: *output gate*, how much to reveal cell
g: *update gate*, how much to write to cell

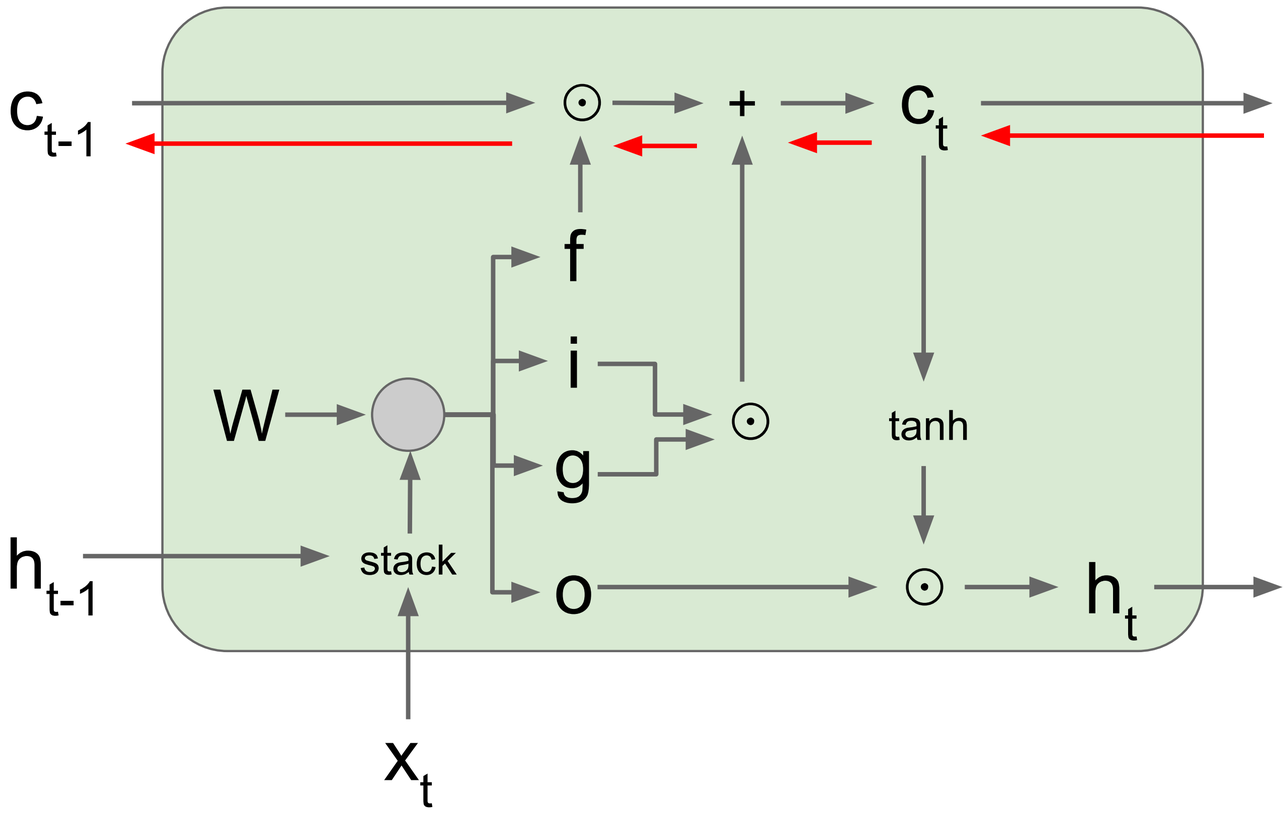


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

LSTM - Gradient Flow



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

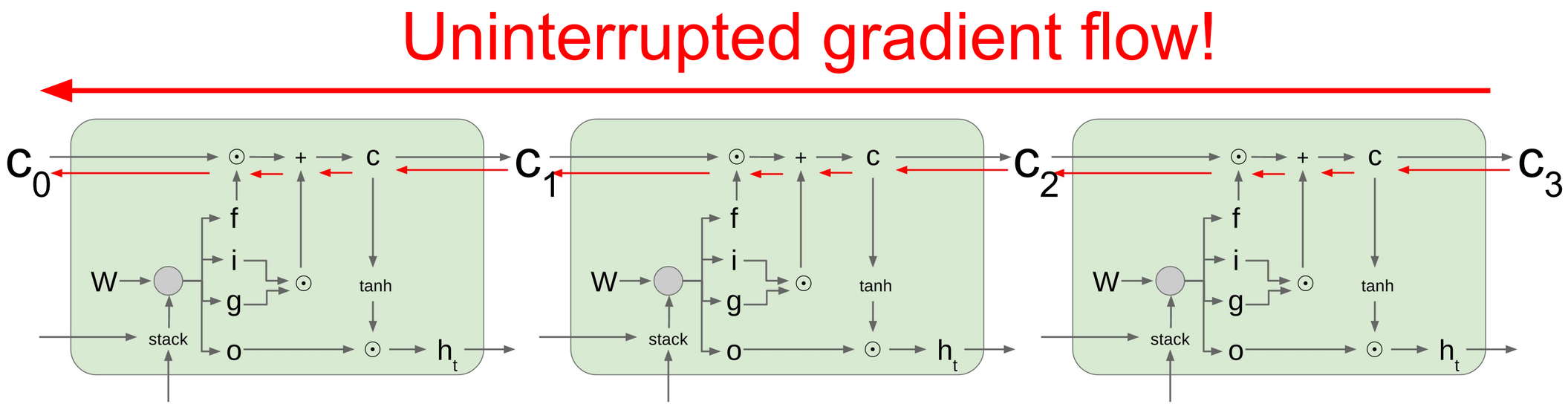
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Does LSTM solve the Vanishing Gradient?

- The LSTM architecture makes it easier for the RNN to preserve information over many timesteps
  - e.g. if the f $= 1$ and the i $= 0$, then the information of that cell is preserved indefinitely.
  - By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix that preserves info in hidden state
- LSTM doesn't guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

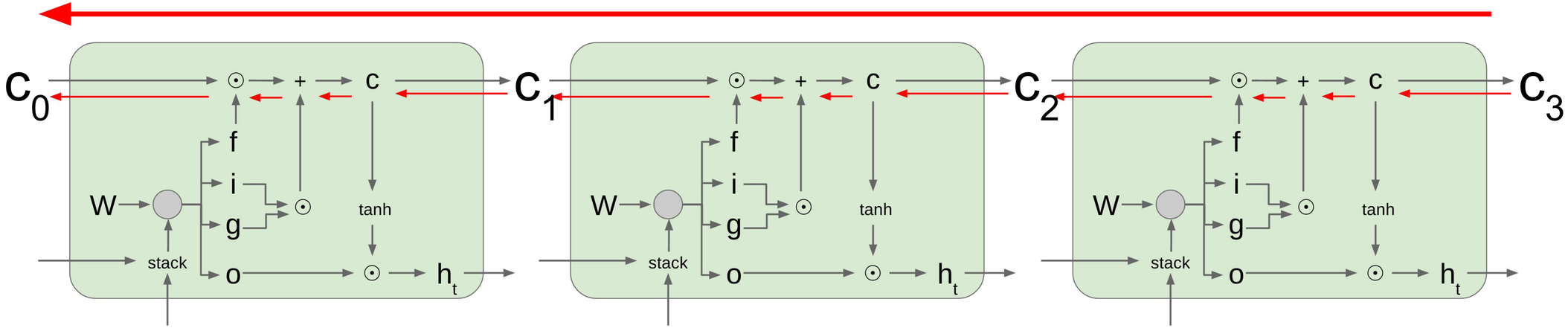
LSTM - Gradient Flow



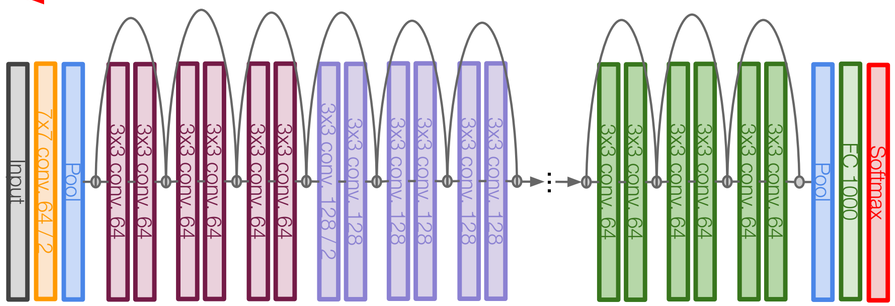
- Notice that the gradient contains the f gate's vector of activations
  - Allows better control of gradients values, using suitable parameter updates of the forget gate.
- Also notice that are added through *f,i,g* and *o* gates
  - Better balancing of gradient values

LSTM - Gradient Flow

Uninterrupted gradient flow!



Similar to ResNet!



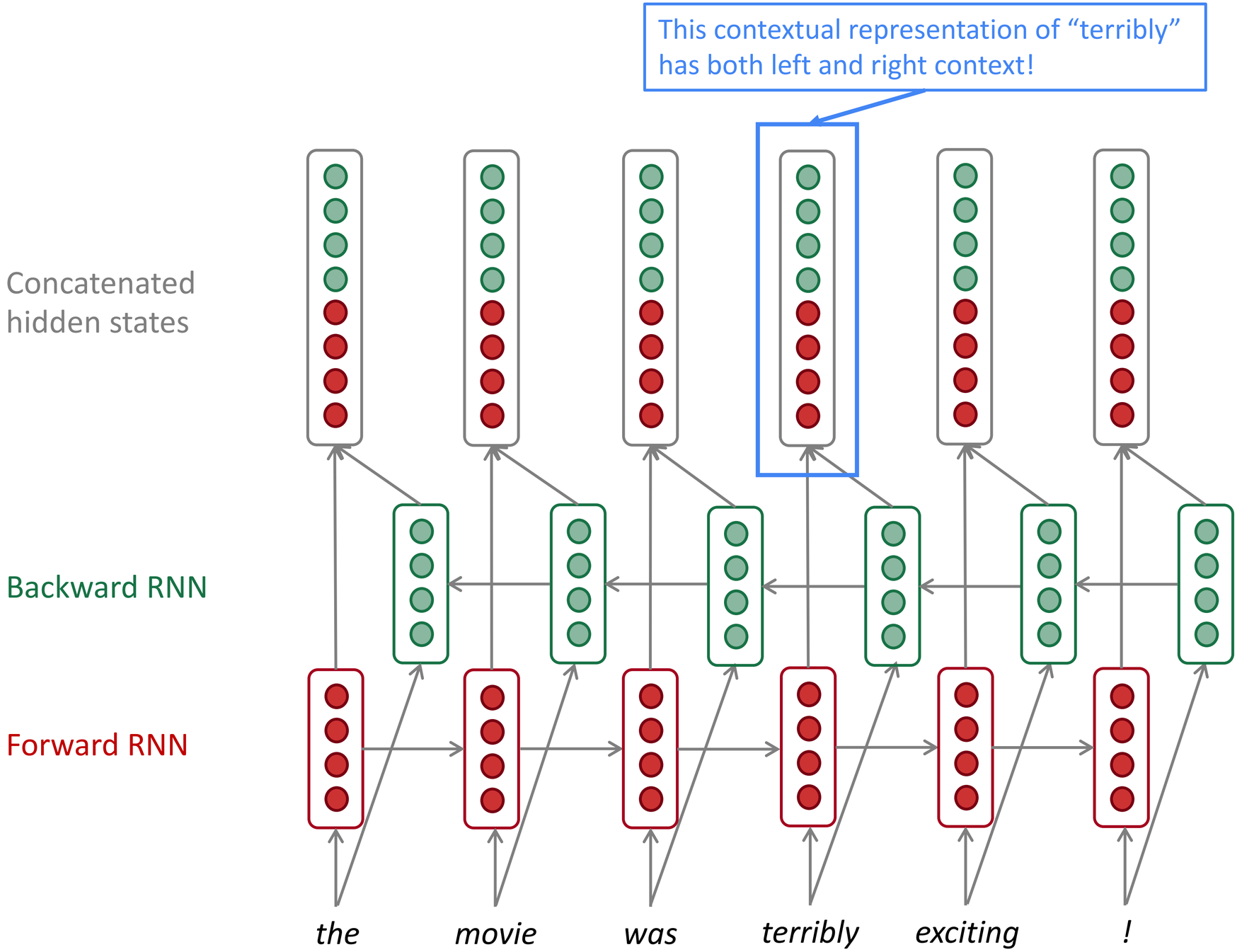
Summary - So far

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research, as well as new paradigms for reasoning over sequences
- Better understanding (both theoretical and empirical) is needed.

Bidirectional RNNs

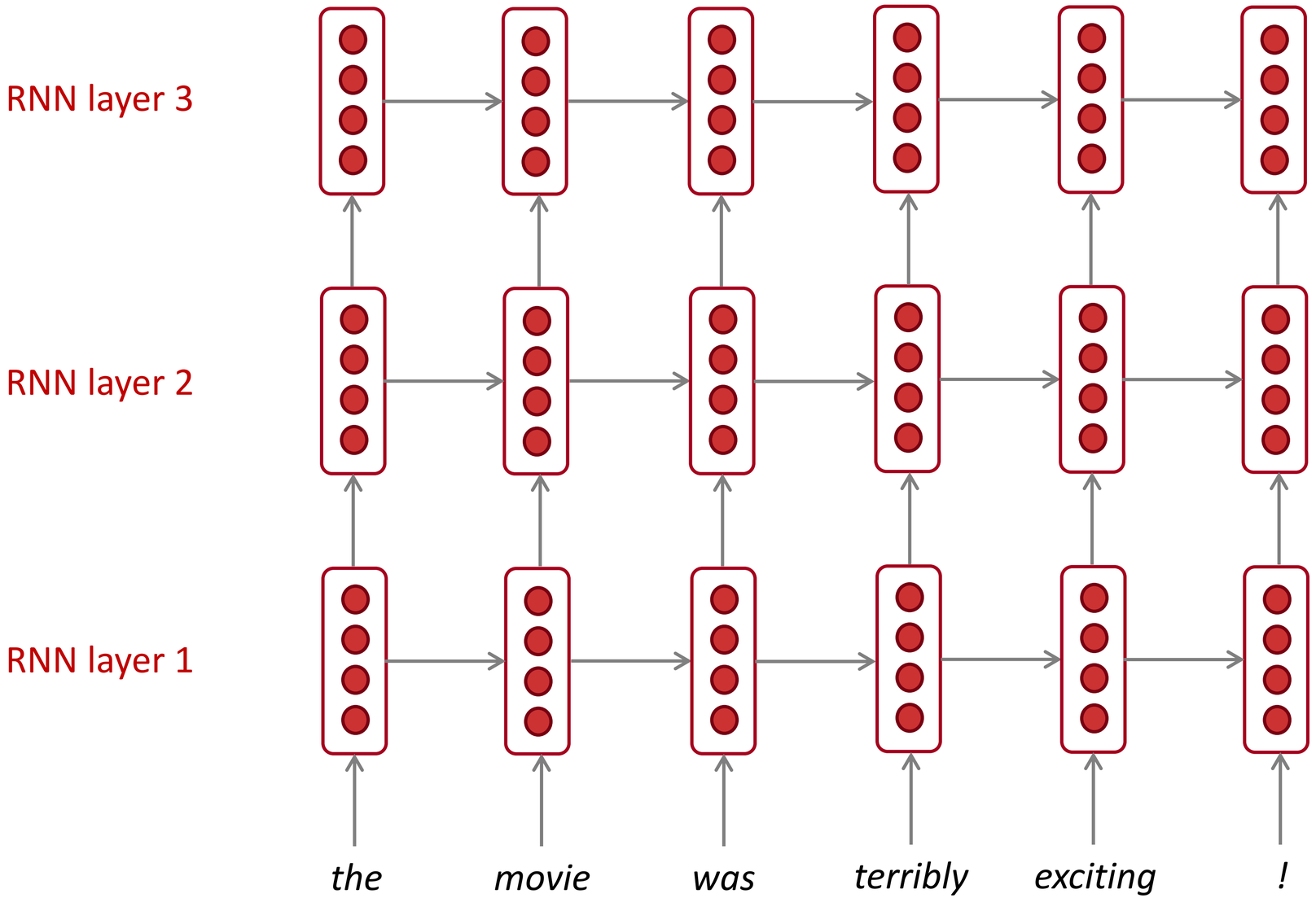
Combination of

- Forward RNN
- Backward RNN



Multi-Layer RNNs

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



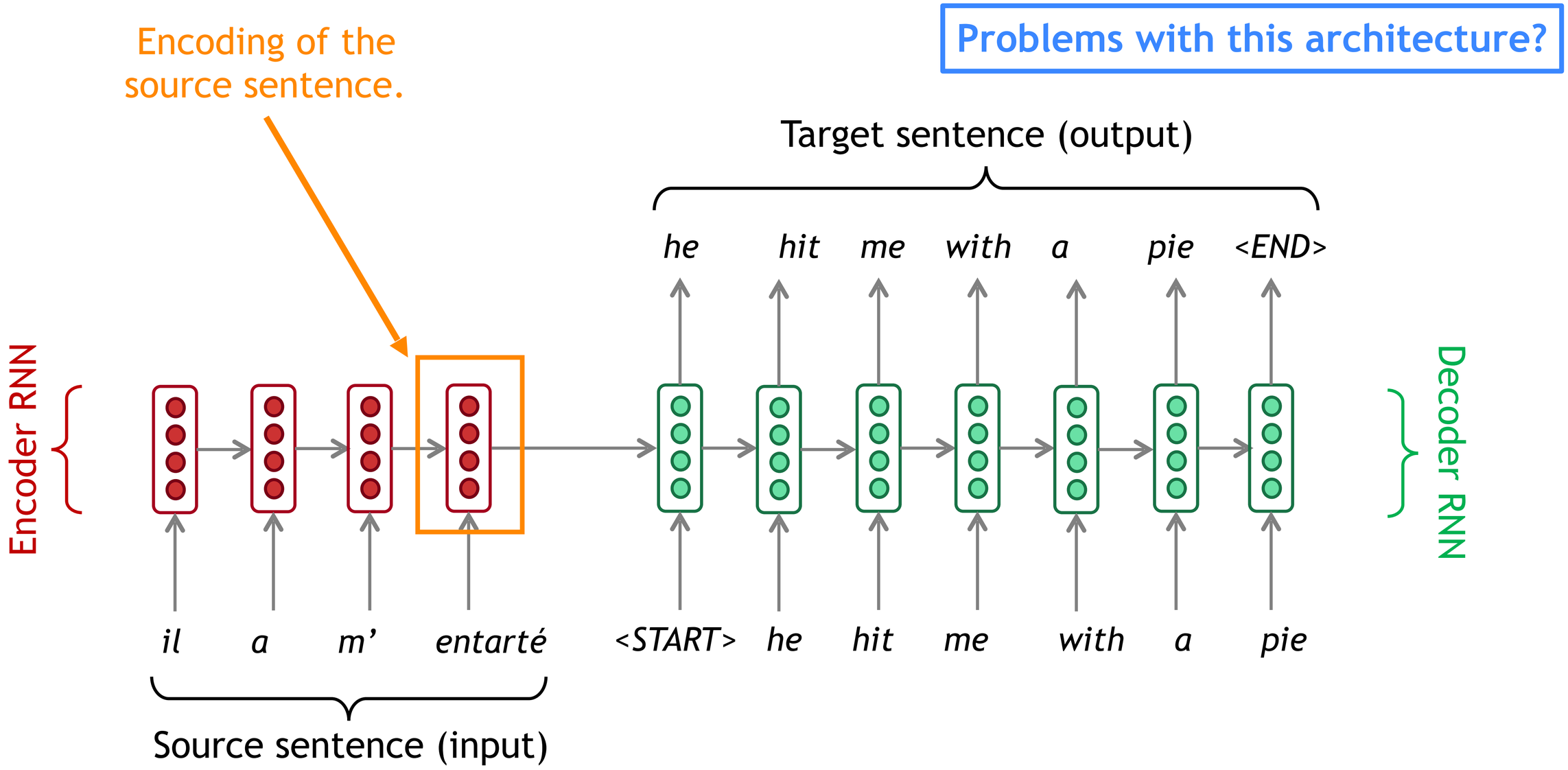
Multi-layer RNNs in practice

- High-performing RNNs are often 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
  - However, skip-connections/dense-connections (ResNet) are needed to train deeper RNNs (e.g. 8 layers)

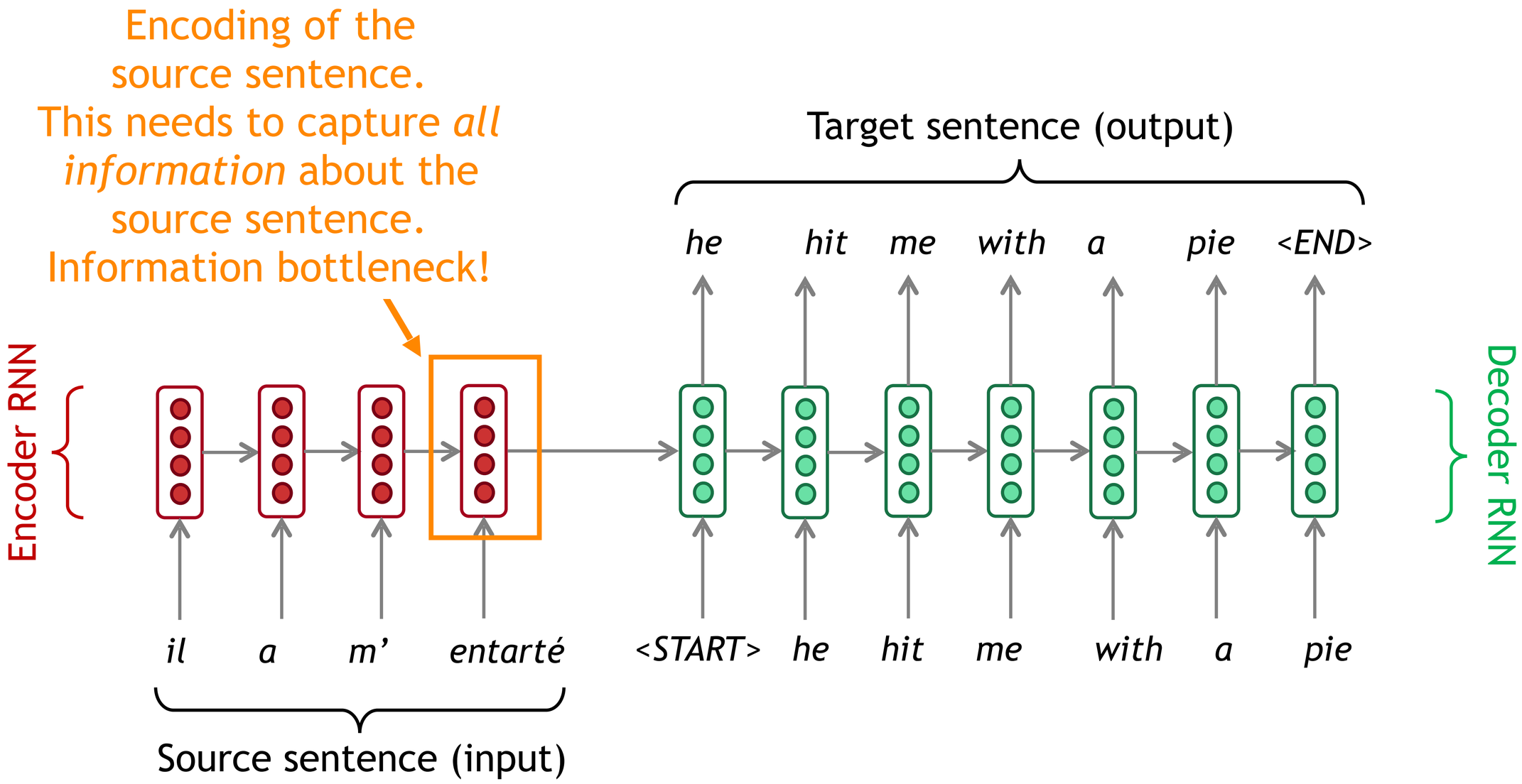
Attention Layer

Slides courtesy of cs224n Stanford

Sequence-to-Sequence: the Bottleneck Problem



Sequence-to-Sequence: the Bottleneck Problem

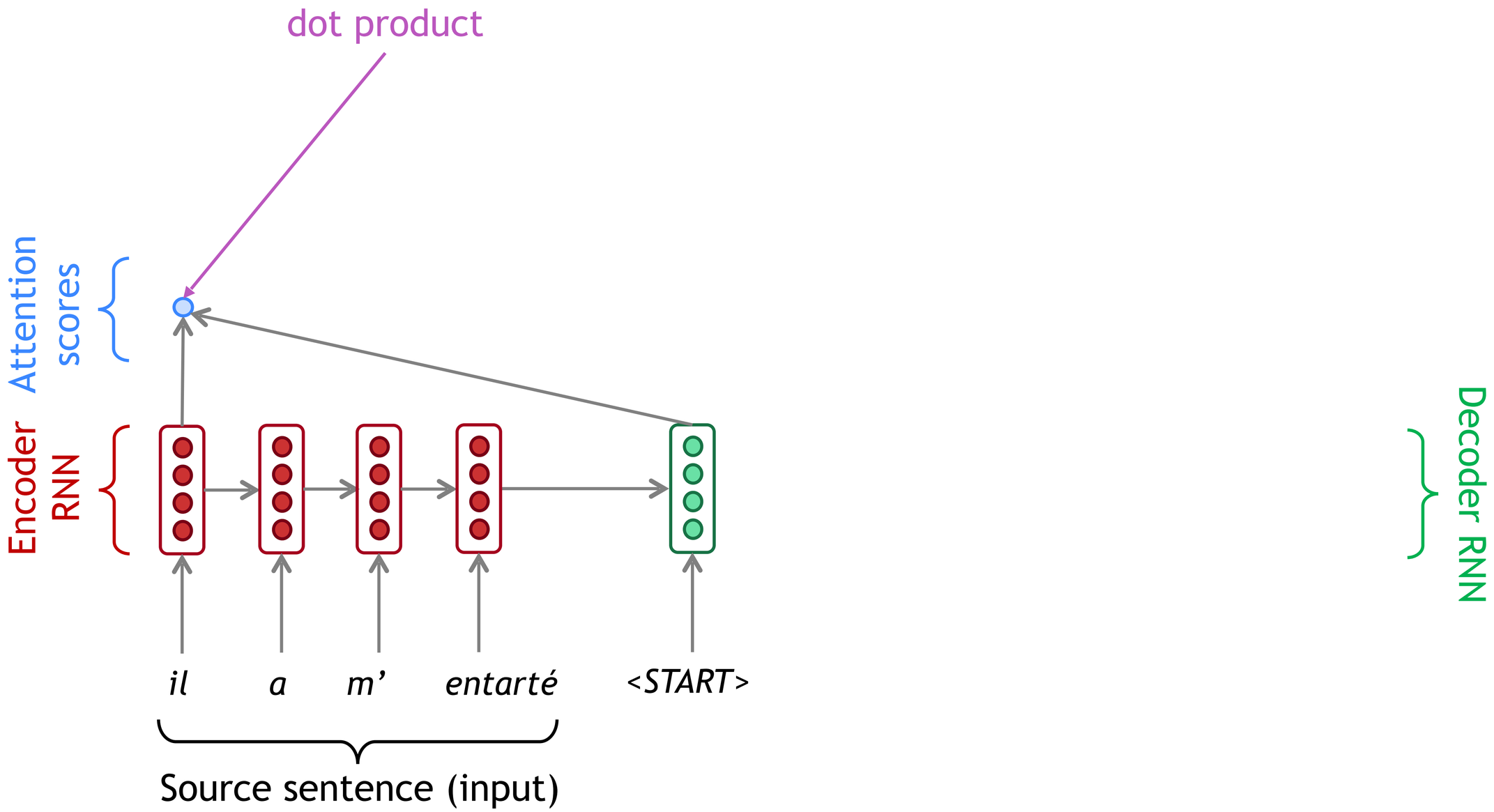


Attention

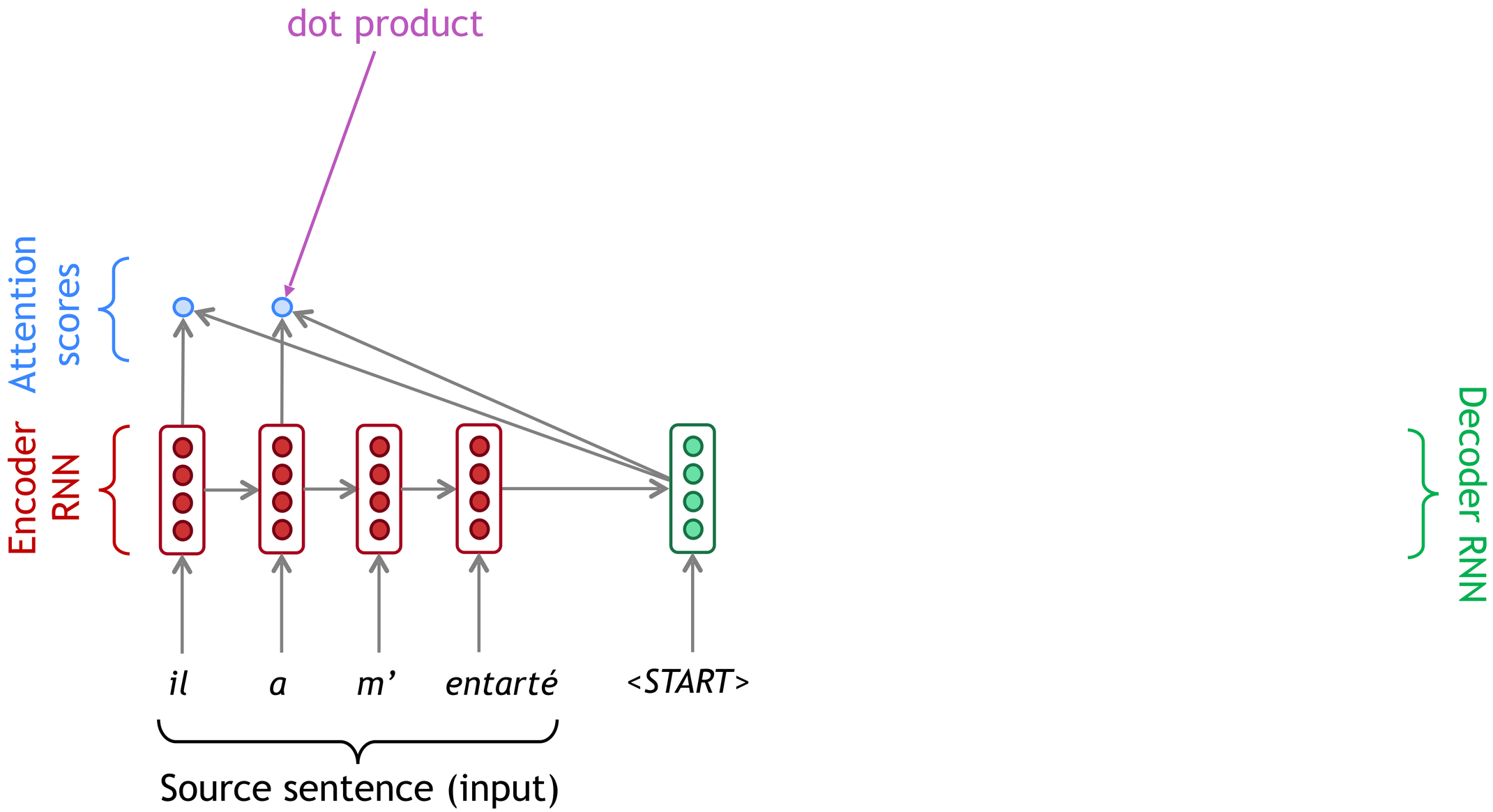
- Attention provides a solution to the bottleneck problem.
- Core idea: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence

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

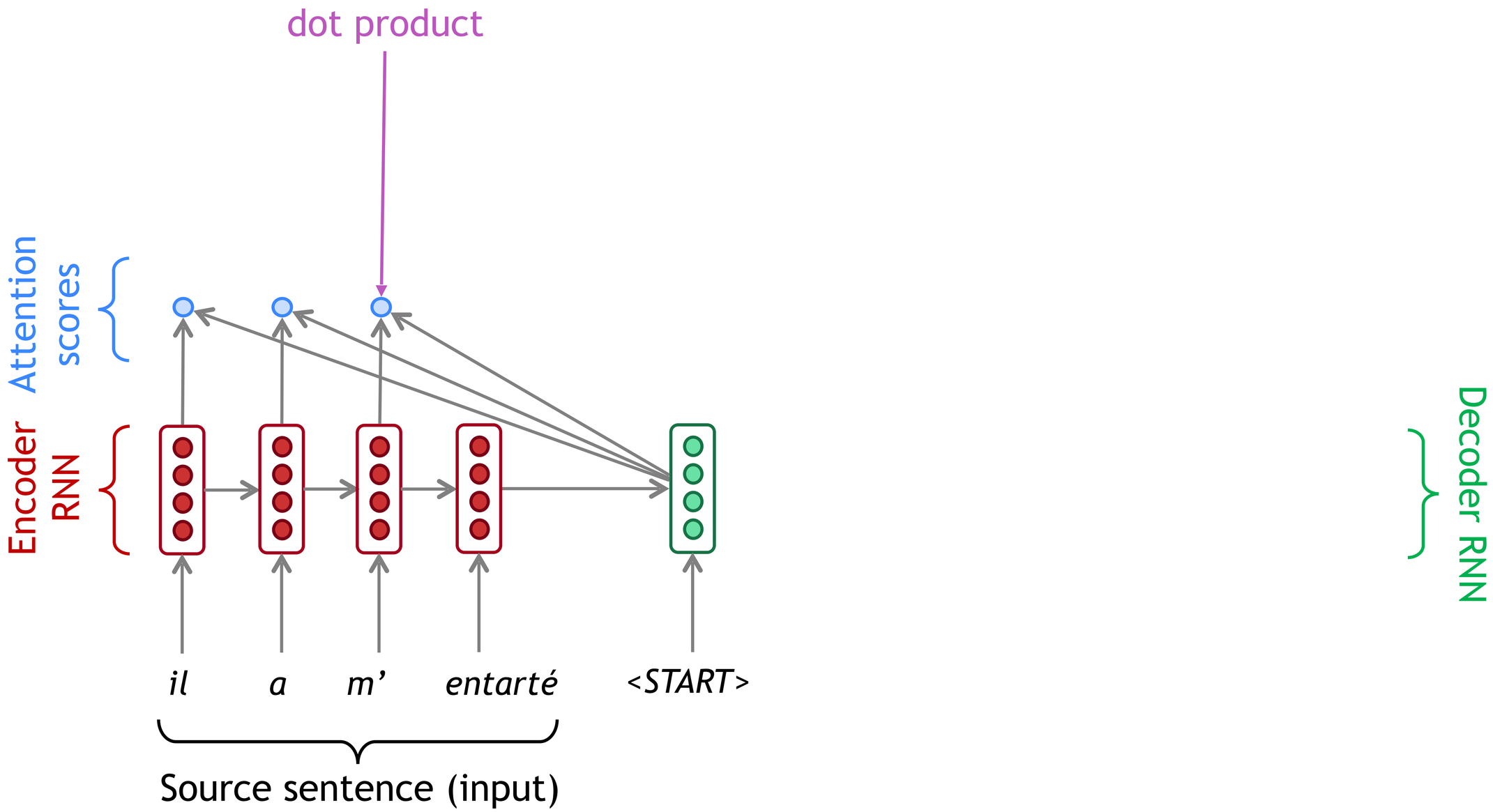
Sequence-to-Sequence with Attention



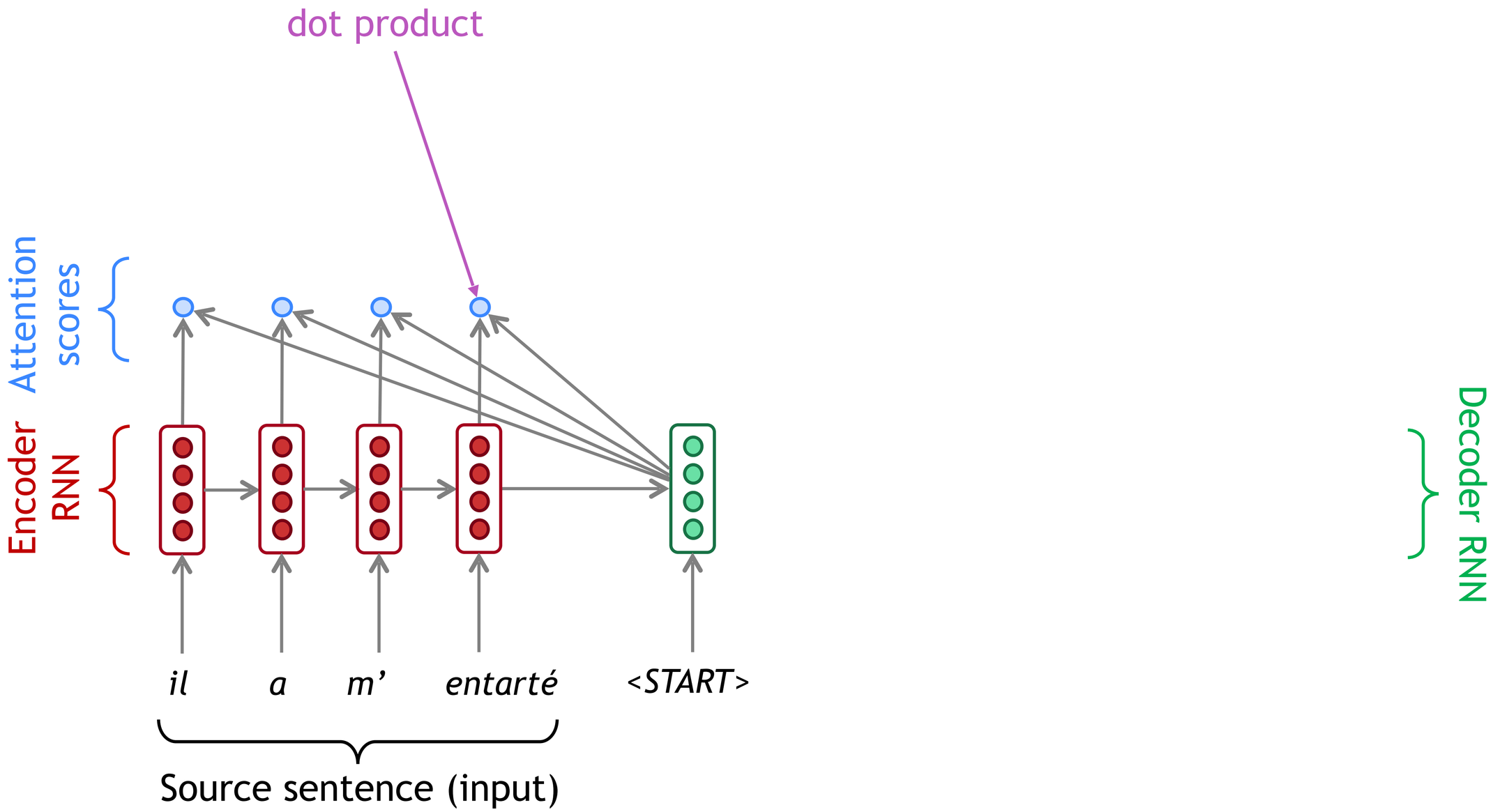
Sequence-to-Sequence with Attention



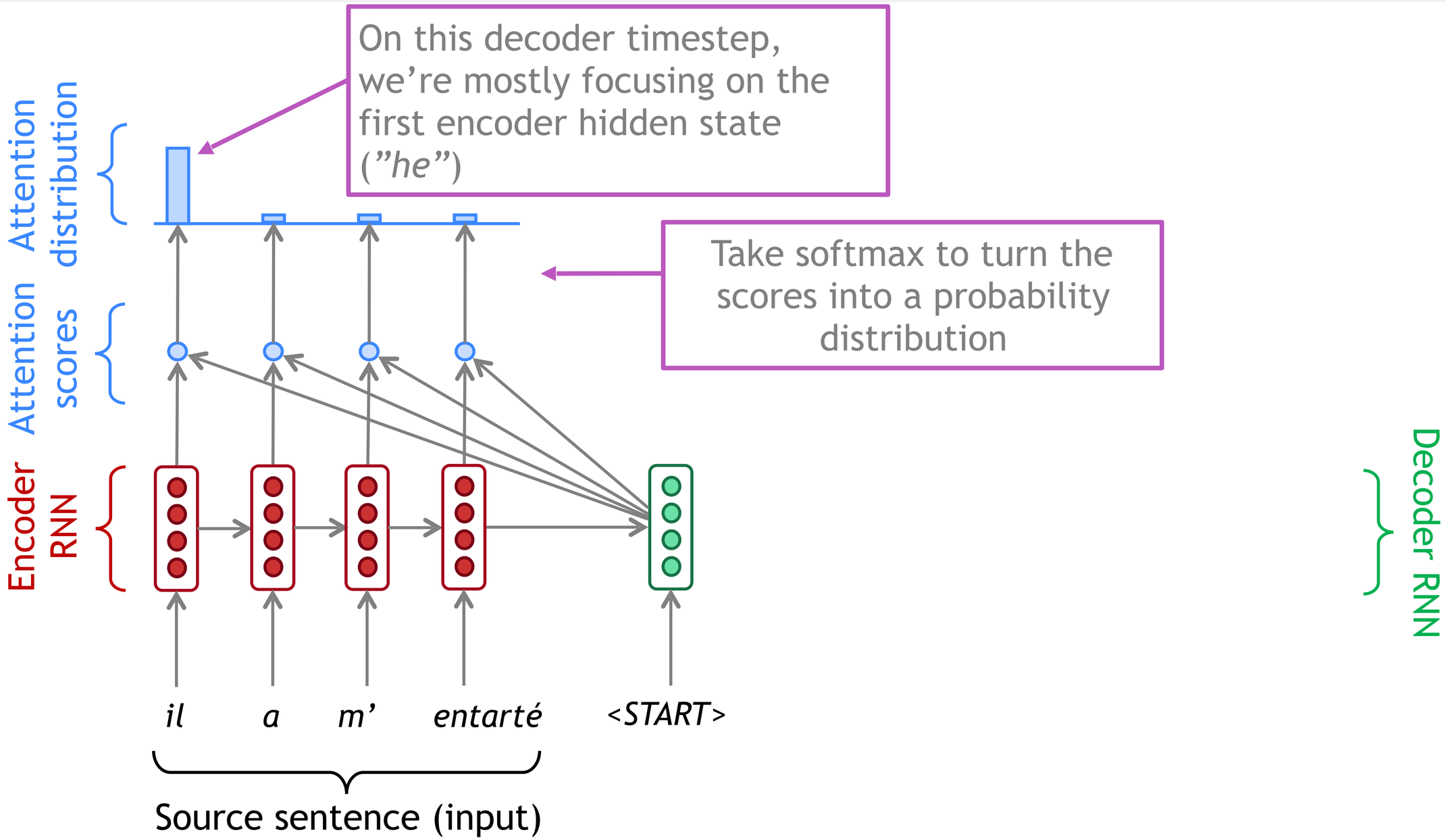
Sequence-to-Sequence with Attention

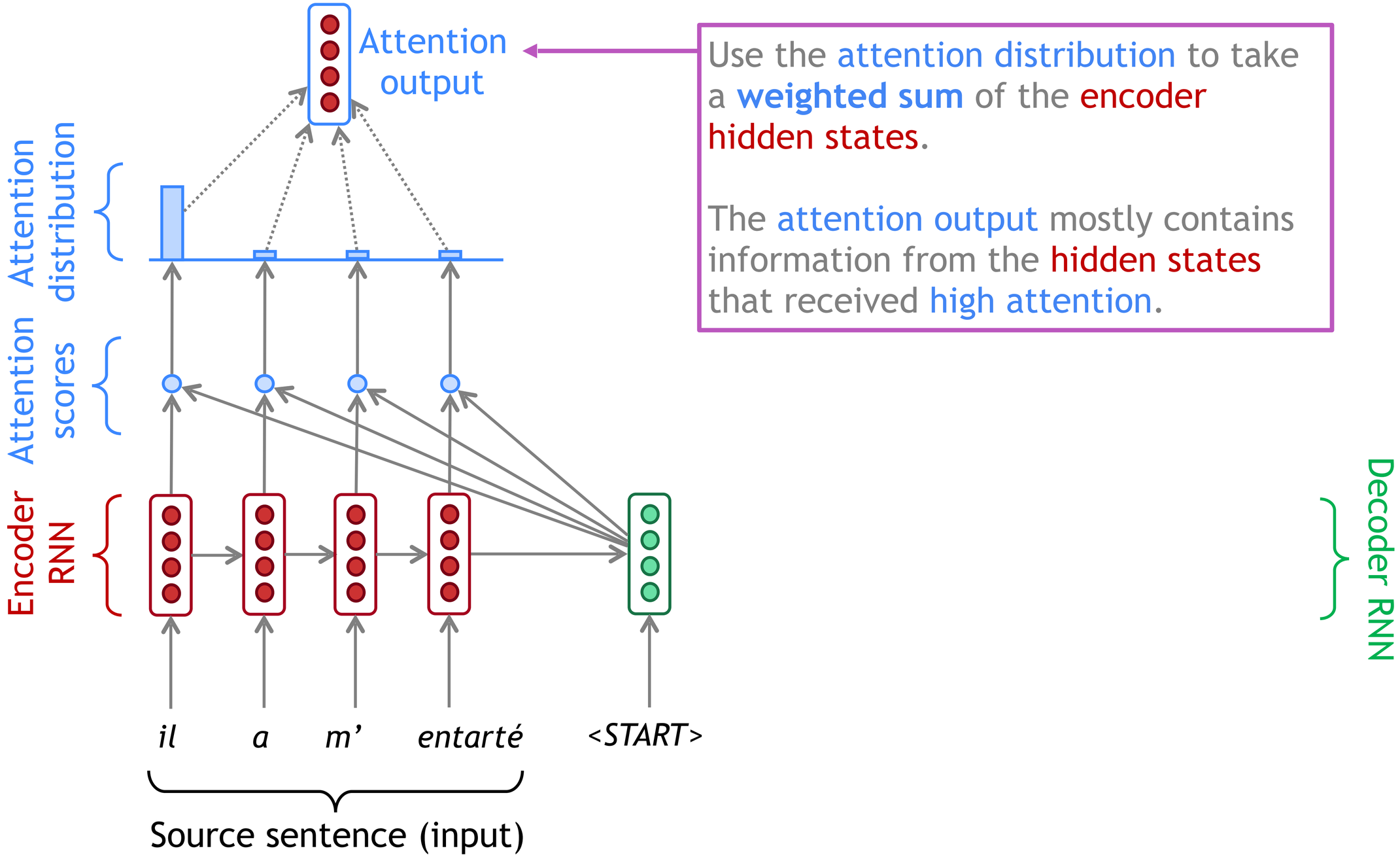


Sequence-to-Sequence with Attention



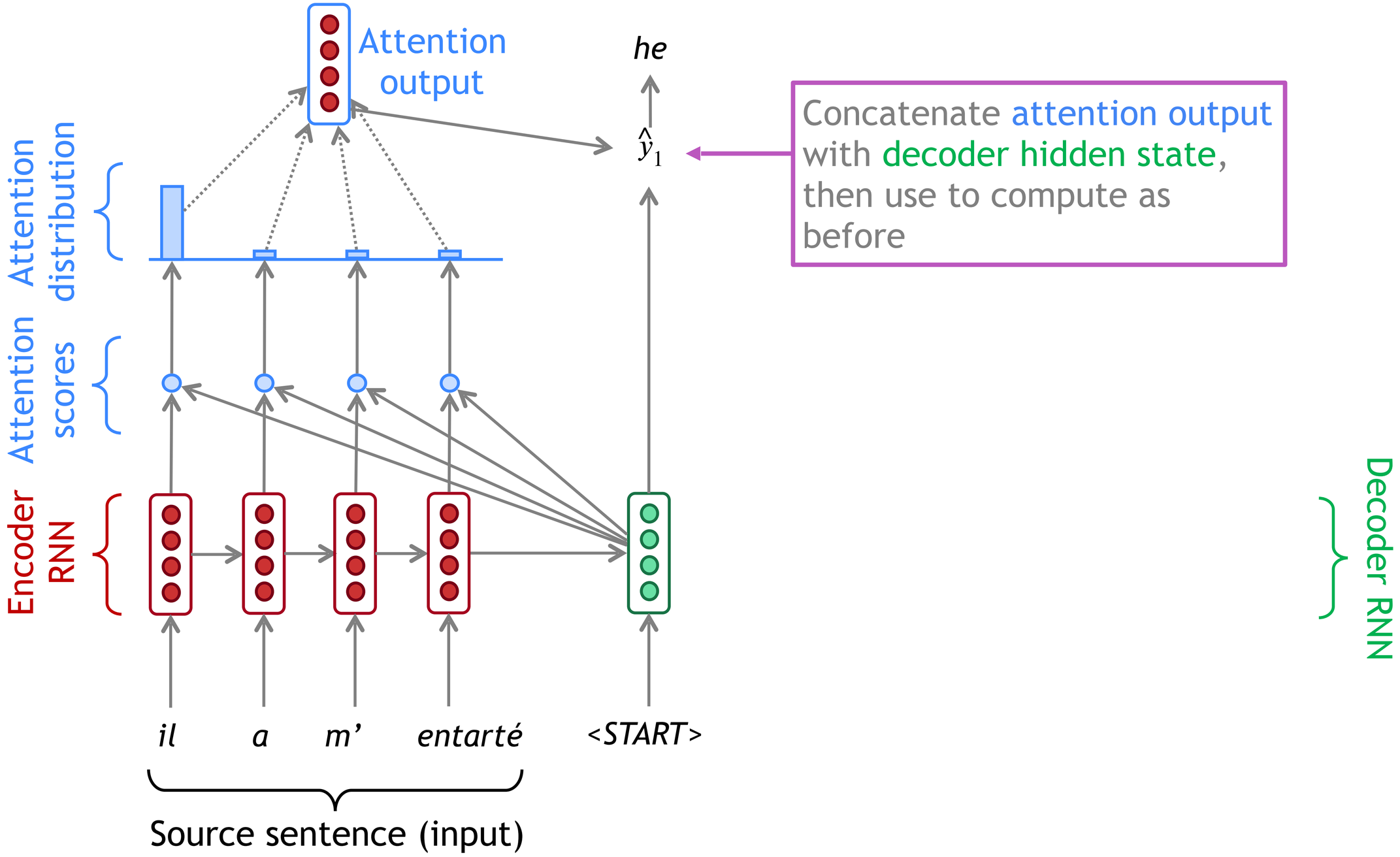
Sequence-to-Sequence with Attention

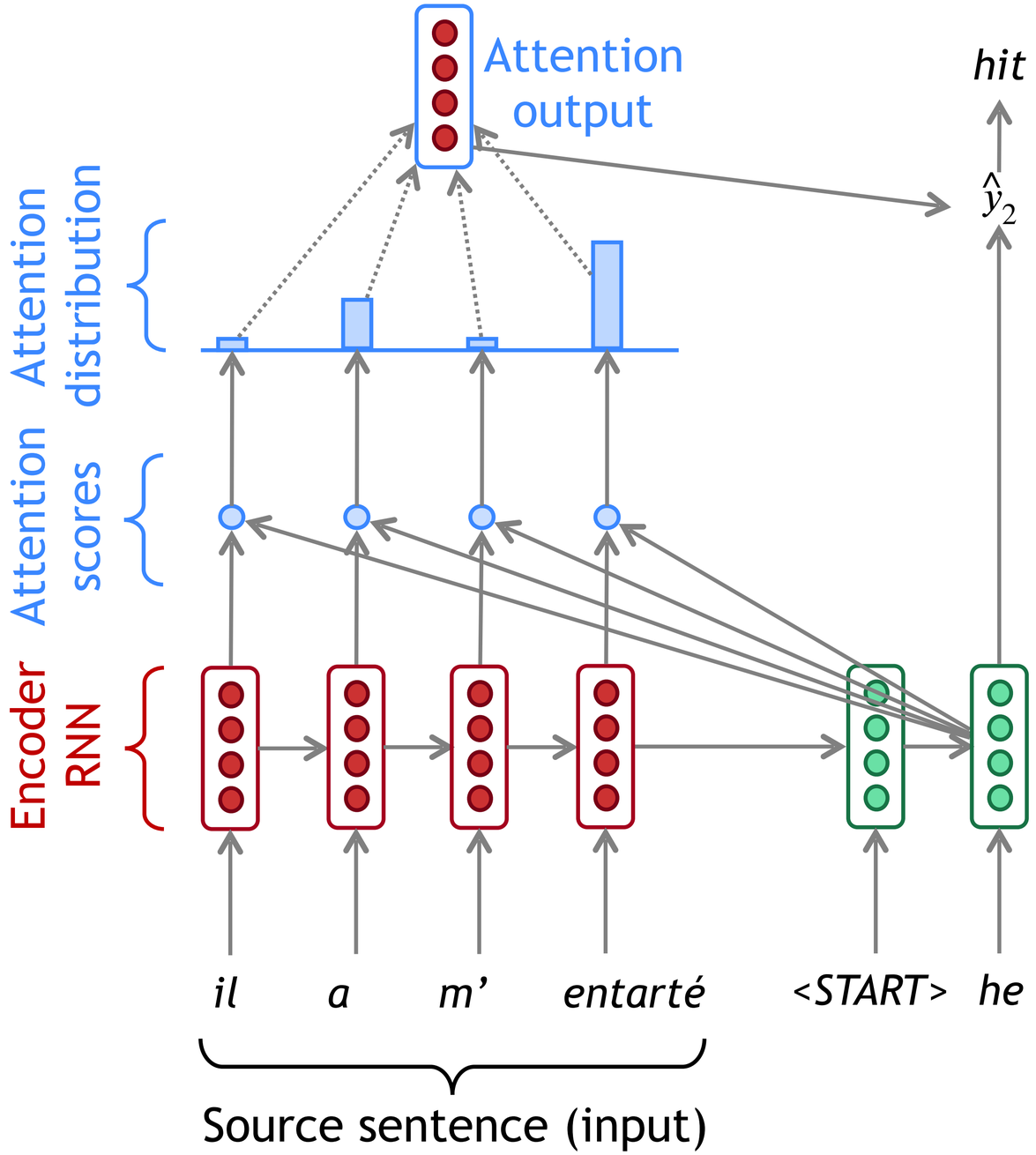


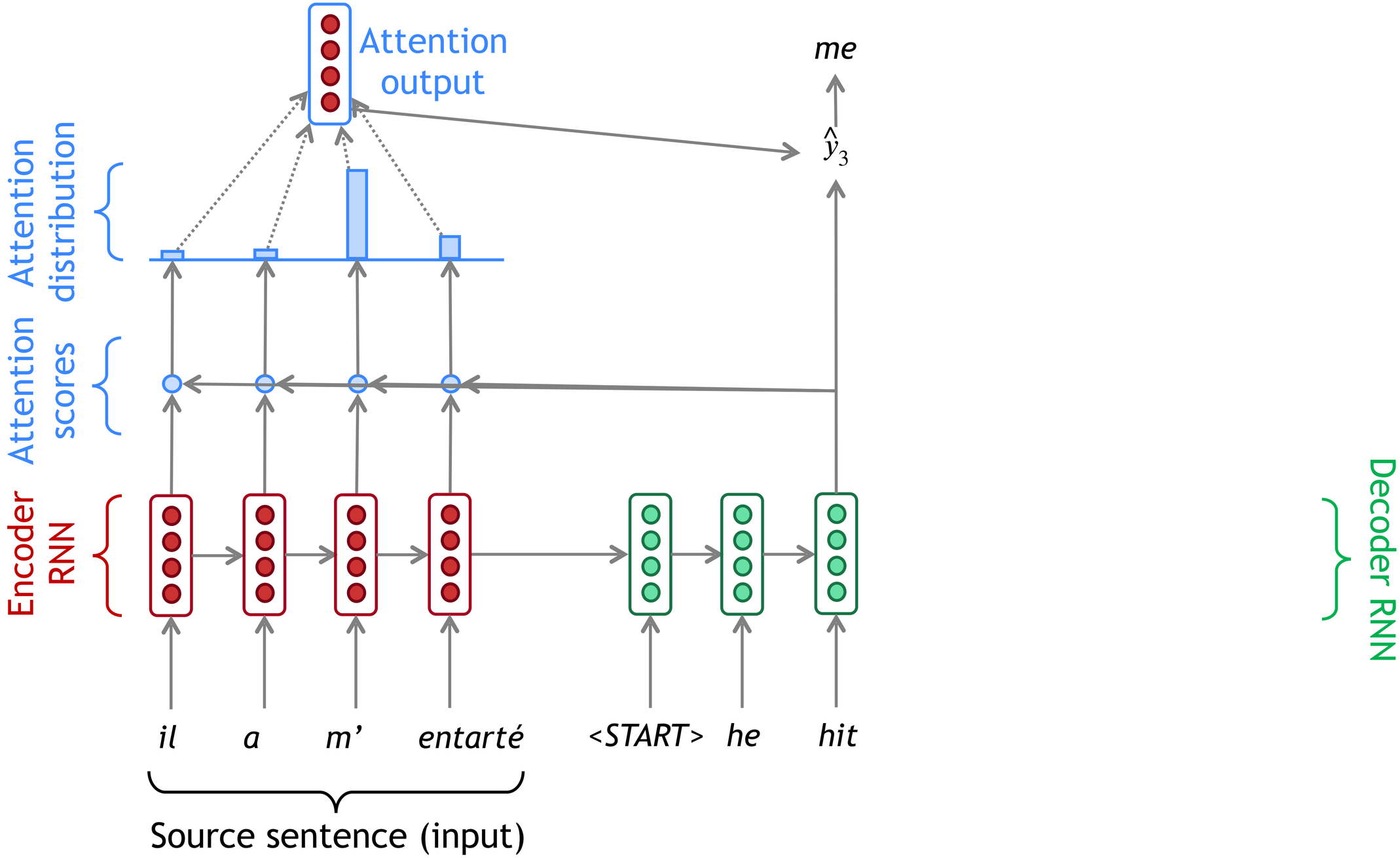


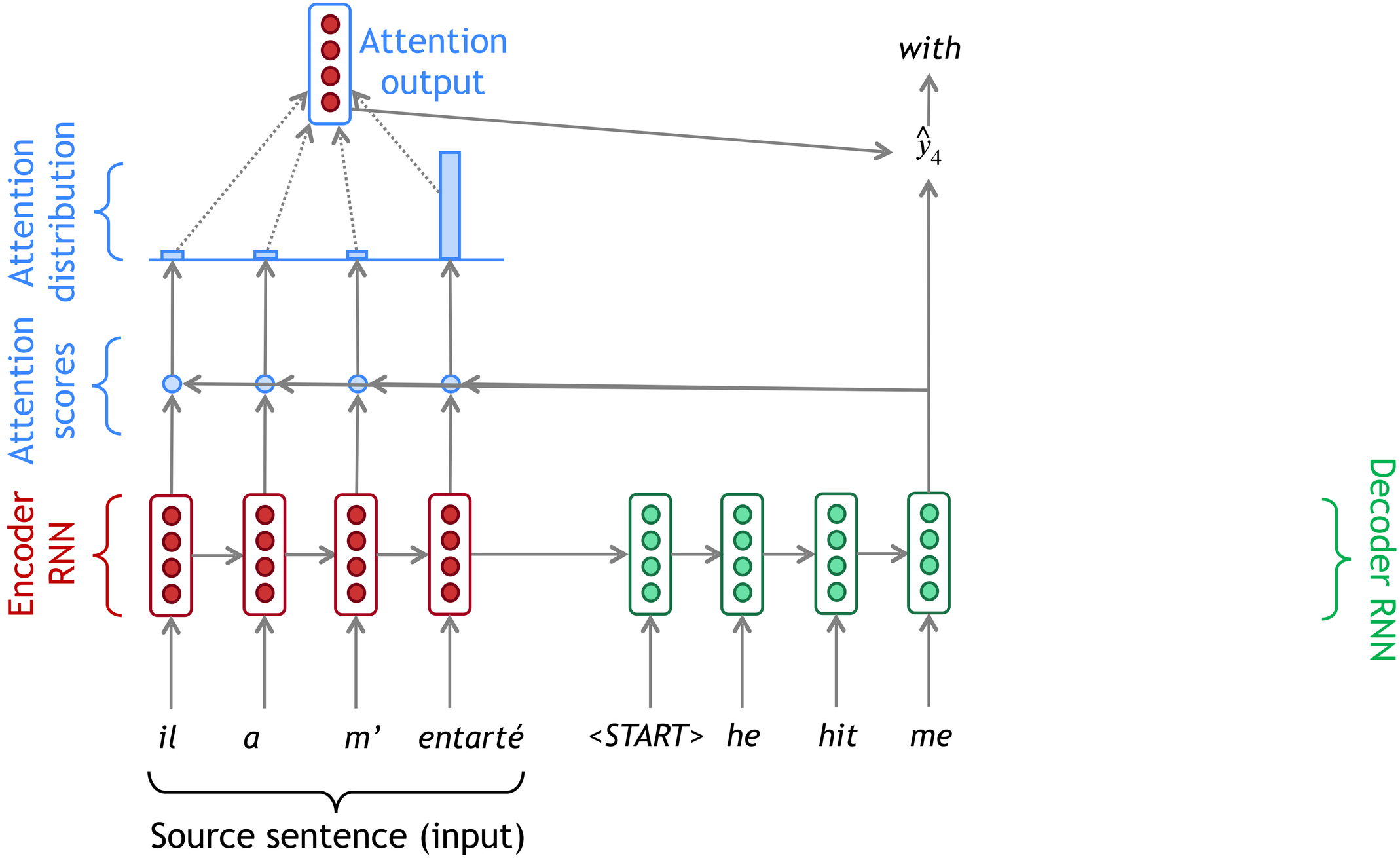
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

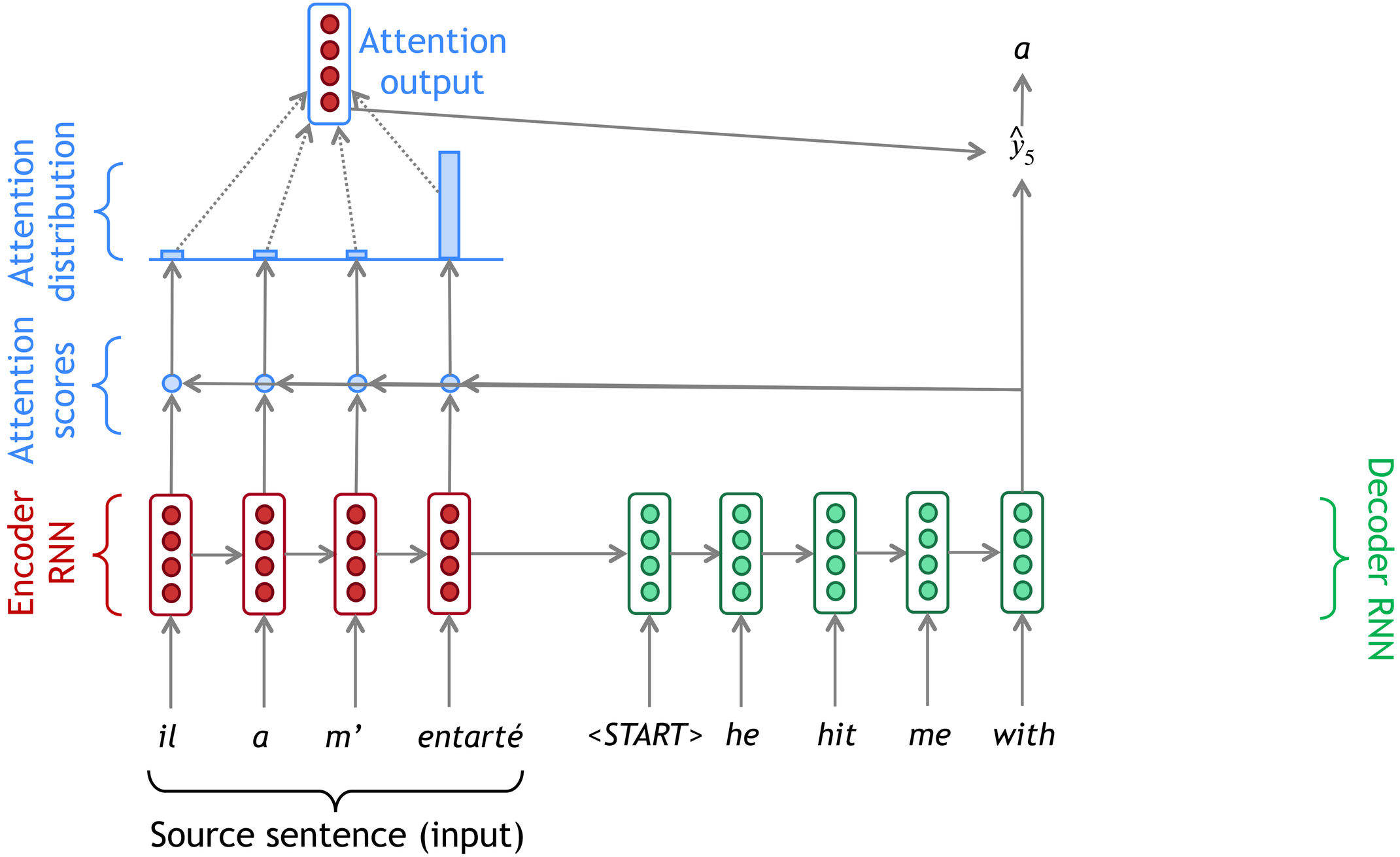
The attention output mostly contains information from the hidden states that received high attention.

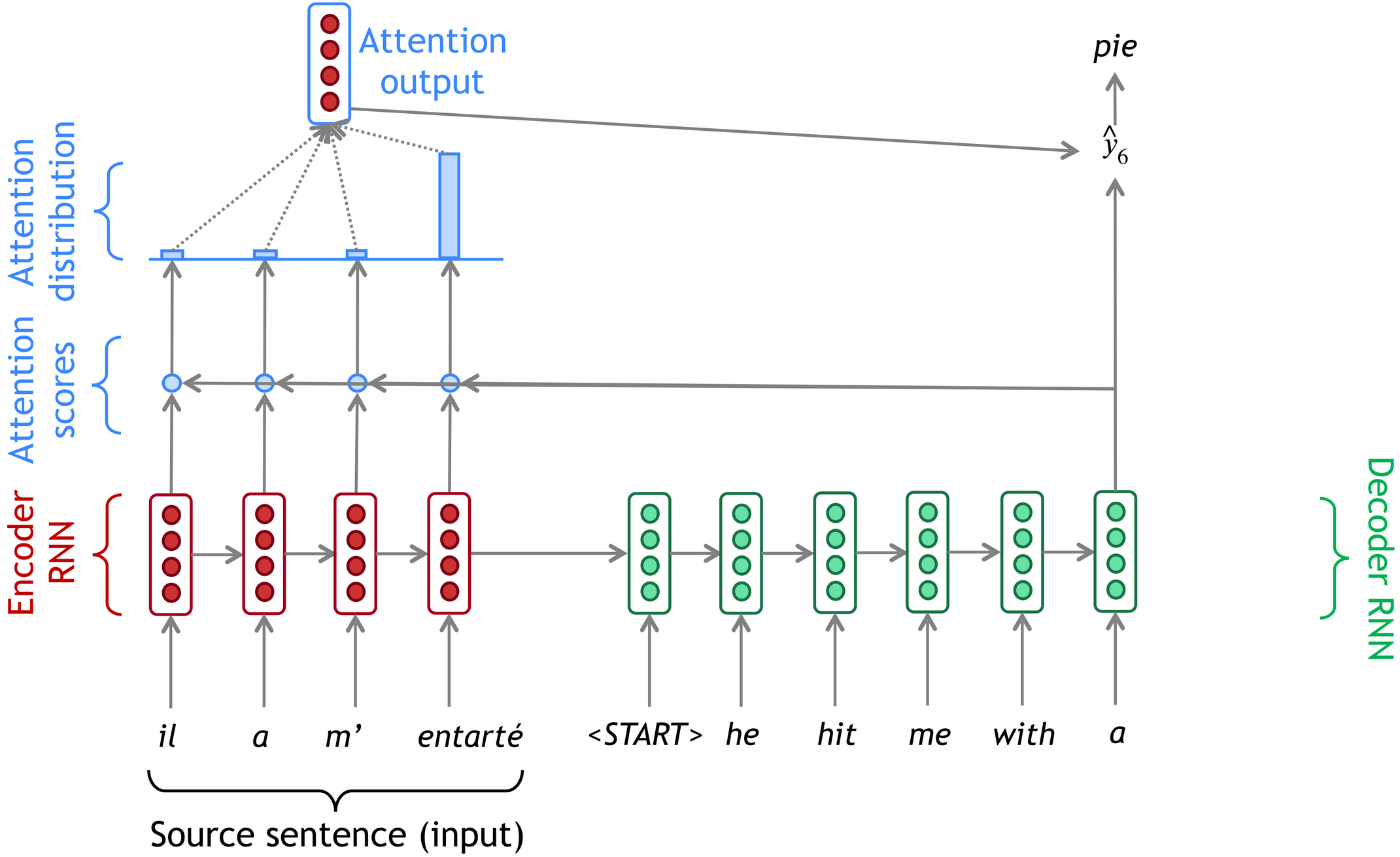

Attention output
he
$\hat{y}_1$
Concatenate attention output with decoder hidden state, then use to compute as before
Attention distribution
Attention scores
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
Source sentence (input)


Attention output
hit
$\hat{y}_2$
Attention distribution
Attention scores
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
he
Source sentence (input)


Attention output
me
$\hat{y}_3$
Attention distribution
Attention scores
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
he
hit
Source sentence (input)


Attention output
with
$\hat{y}_4$
Attention distribution
Attention scores
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
he
hit
me
Source sentence (input)


Attention output
a
$\hat{y}_5$
Attention distribution
Attention scores
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
he
hit
me
with
Source sentence (input)


Attention output
pie
$\hat{y}_6$
Attention distribution
Attention scores
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
he
hit
me
with
a
Source sentence (input)

Attention: In equations

- We have encoder hidden states: $h_1, h_2, \ldots, h_N$
- On timestep t, we have decoder hidden state s_t
- We get the attention scores for this step: $\mathbf{e}^t = [s_t \cdot h_1, \ldots, s_t \cdot h_N]$
- We take softmax to get the attention distribution for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(\mathbf{e}^t)$$

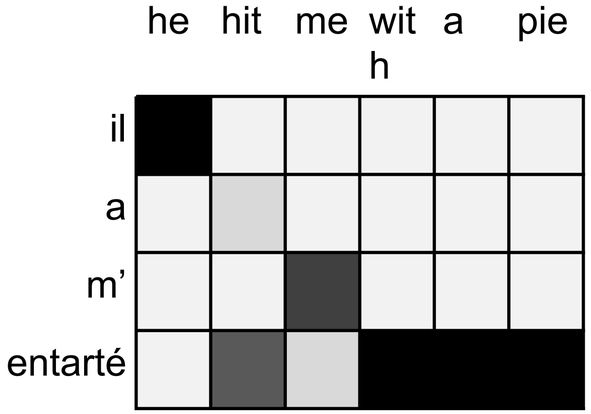
- We use to take a weighted sum of the encoder hidden states to get the attention output

$$\mathbf{a}_t = \sum_{i=1}^{N} \alpha_i^t \mathbf{h}_i$$

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

Attention is Great!

- Attention significantly improves NMT performance
- 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
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself.



Summary

- Attention is a way to focus on particular parts of the input.
- Attention can be used in many applications.
  - Attention on images
- Self-attention is also possible.
  - *Attention is all you need*, Vaswani et al., 2017

A stop sign is on a road with a mountain in the background.

A giraffe standing in a forest with trees in the background.

Attention on Images

Important Ideas!

- Multilayer-Perceptron (Feed-forward nets)
  - Back-propagation / Stochastic gradient descent
  - Bachnorm / Dropout
- CNN
- RNN - LSTM
  - Attention
- GAN