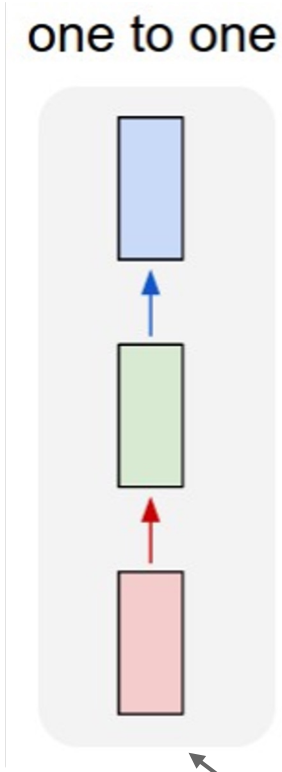
Sequence Modeling (RNNs)

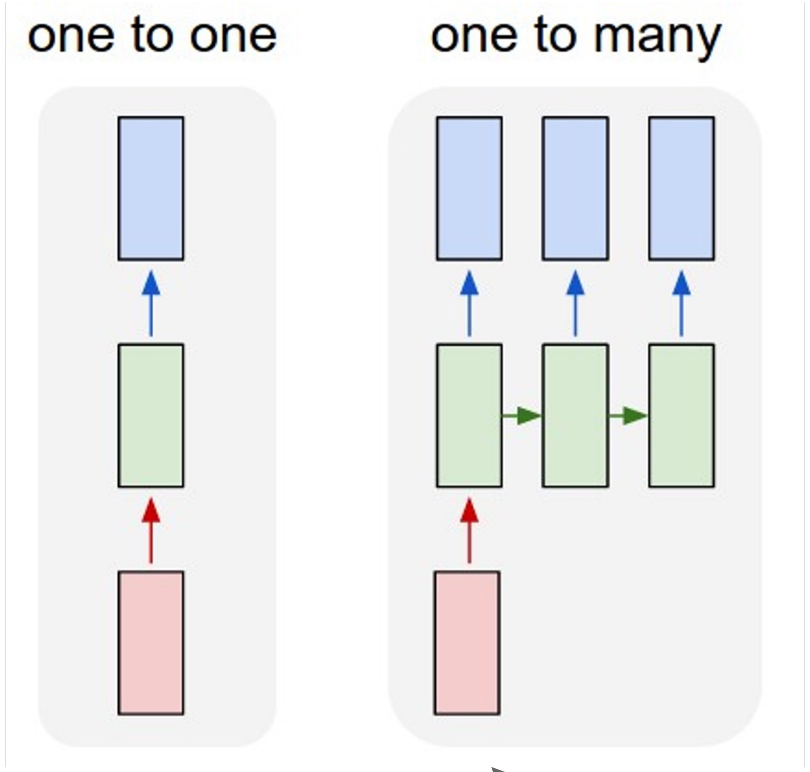
Slides from Justin Johnson

So far: “Feedforward” Neural Networks



e.g. **Image classification**
Image -> Label

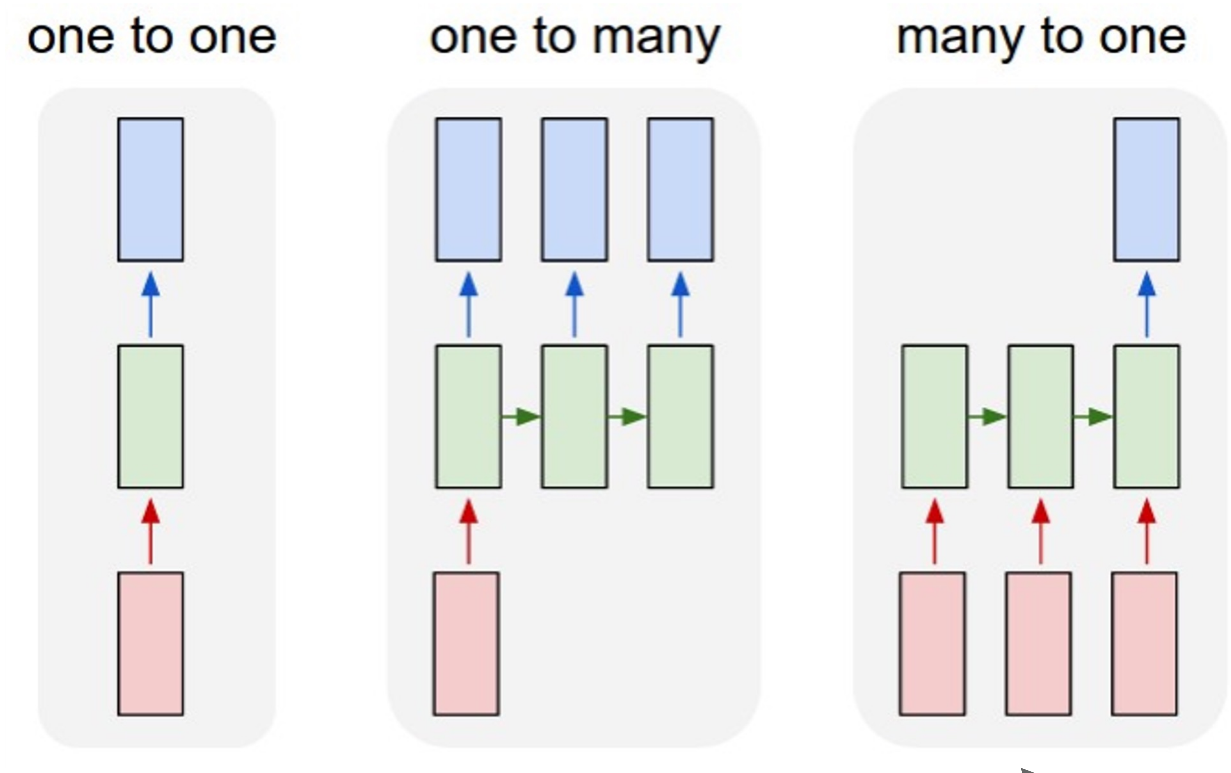
Recurrent Neural Networks: Process Sequences



e.g. **Image Captioning**:
Image -> sequence of words

Slide from Justin Johnson

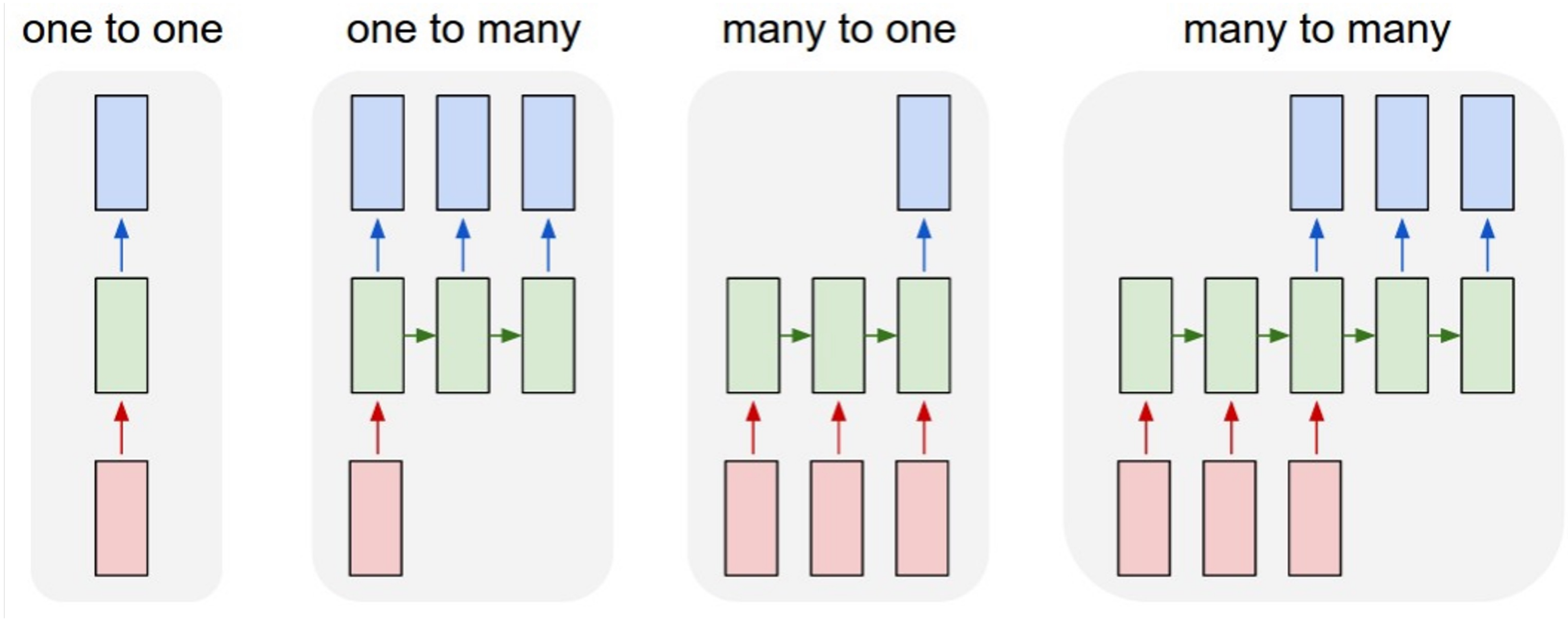
Recurrent Neural Networks: Process Sequences



e.g. **Video classification:**
Sequence of images -> label

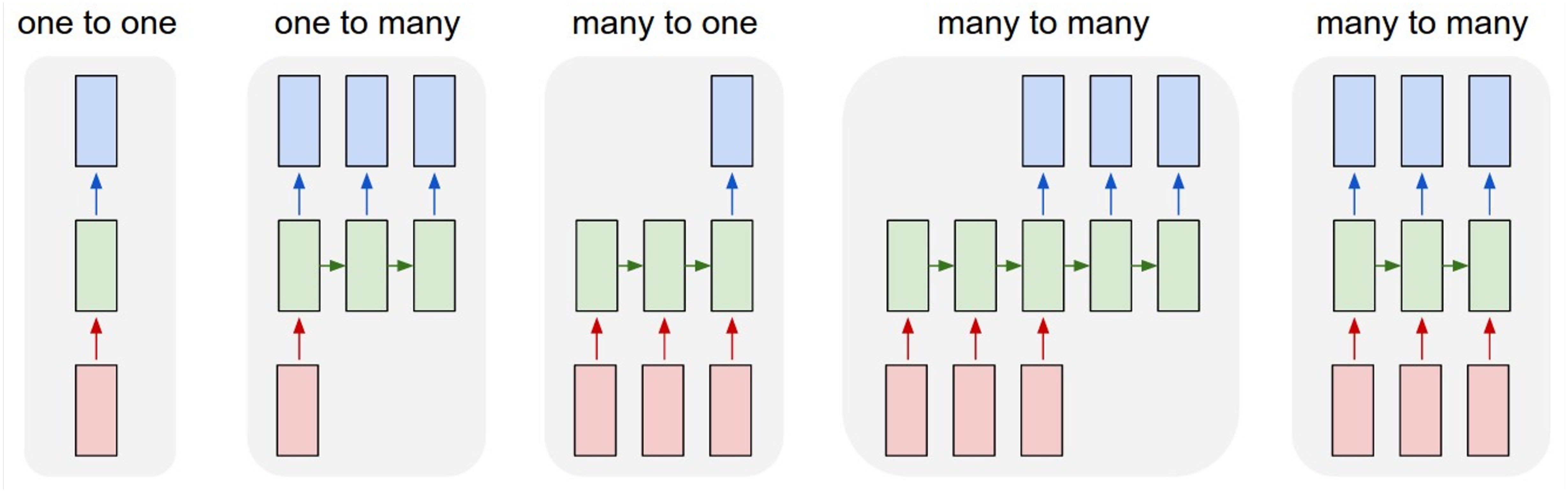
Slide from Justin Johnson

Recurrent Neural Networks: Process Sequences



e.g. **Machine Translation:**
Sequence of words -> Sequence of words

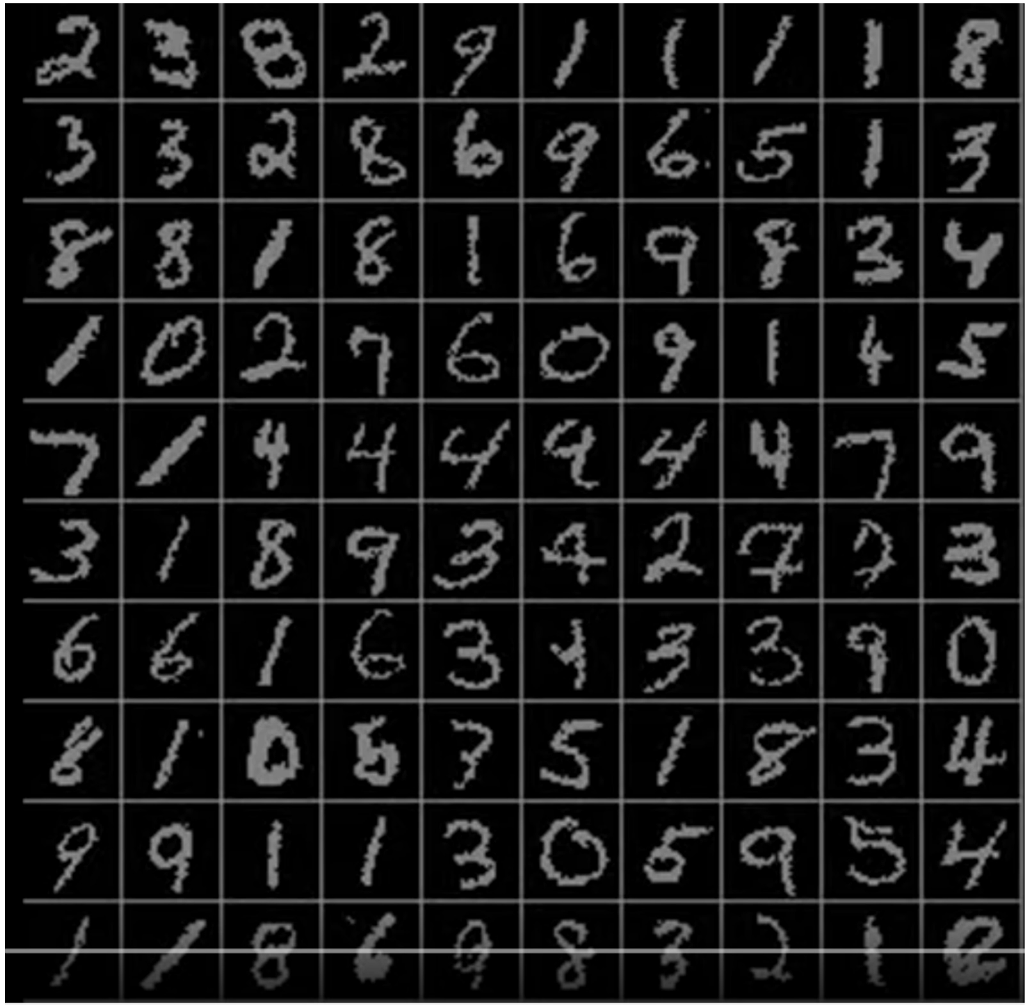
Recurrent Neural Networks: Process Sequences



e.g. **Per-frame video classification:**
Sequence of images -> Sequence of labels

Sequential Processing of Non-Sequential Data

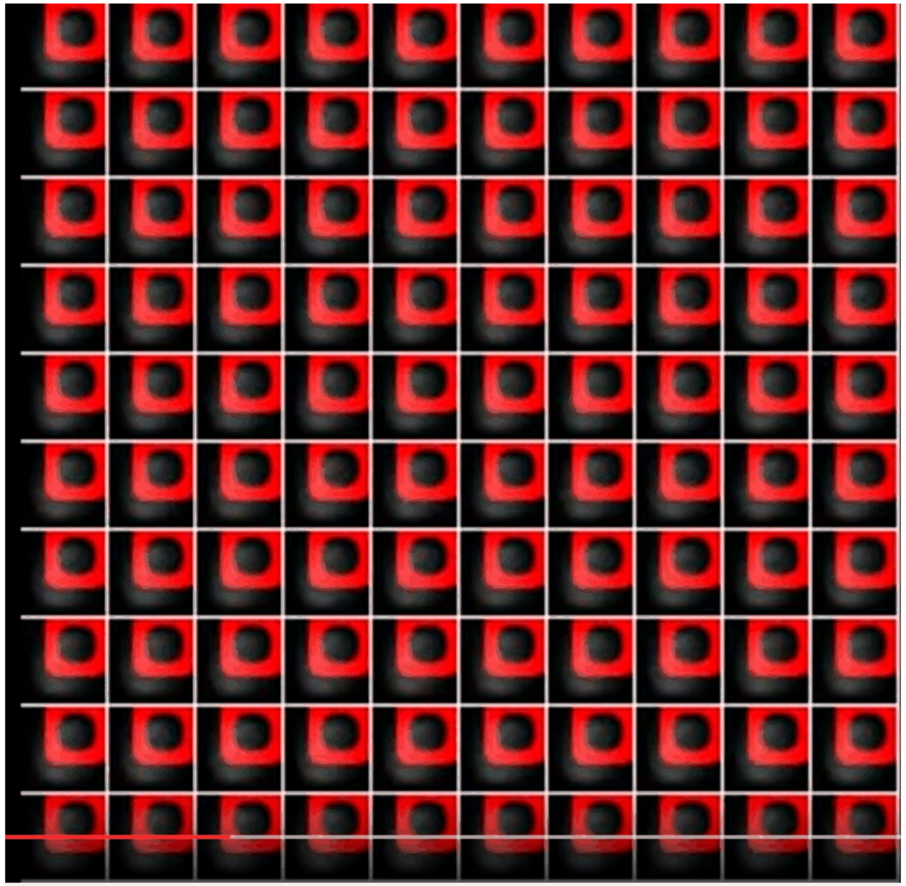
Classify images by taking a series of “glimpses”

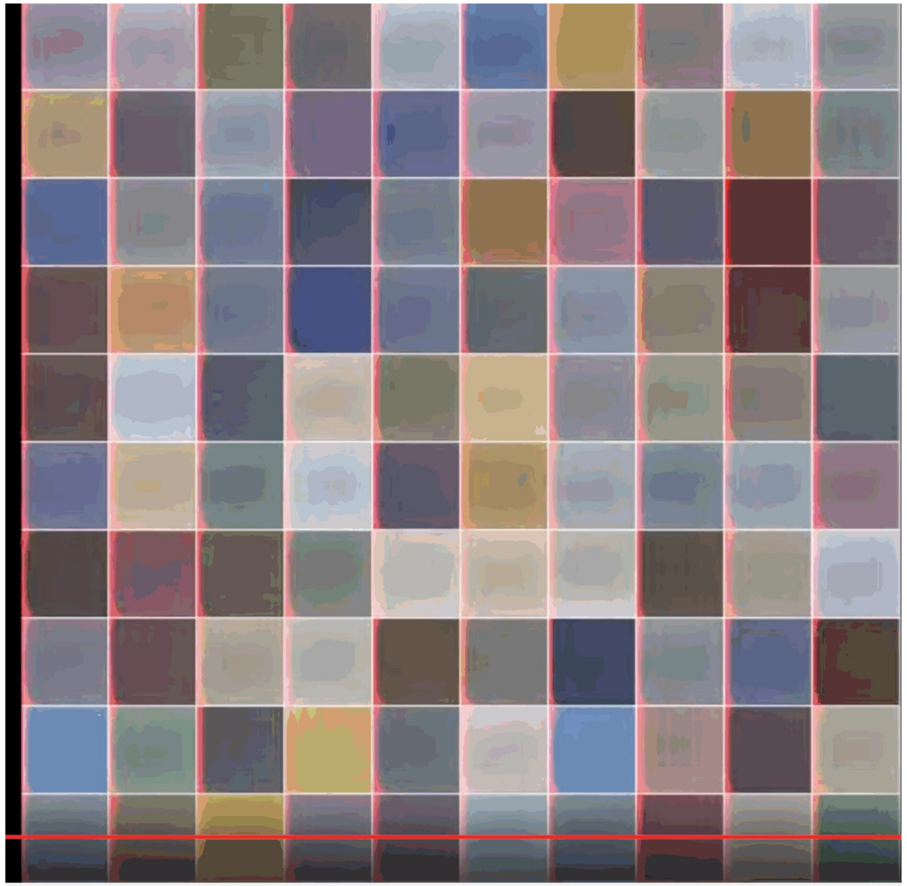


Ba, Mnih, and Kavukcuoglu, “Multiple Object Recognition with Visual Attention”, ICLR 2015.
Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015

Sequential Processing of Non-Sequential Data

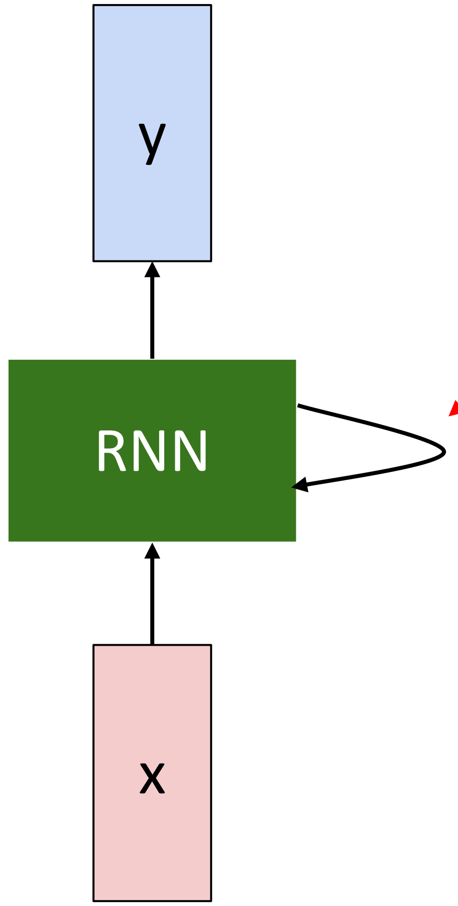
Generate images one piece at a time!





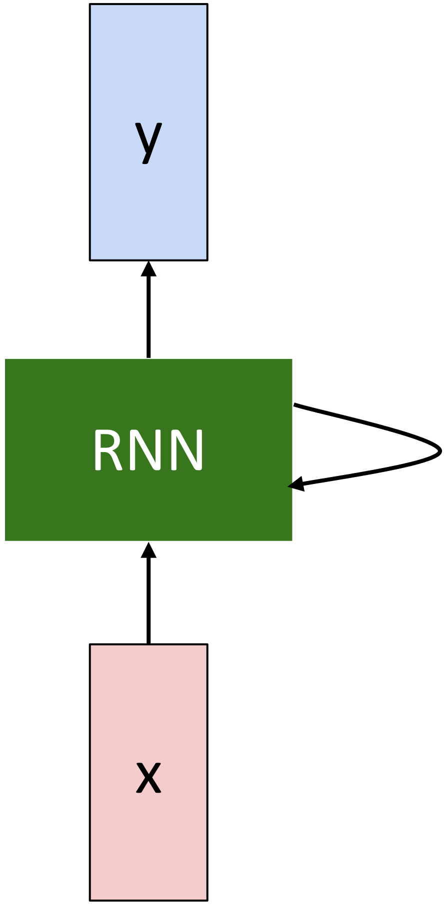
Gregor et al, "DRAW: A Recurrent Neural Network For Image Generation", ICML 2015

Recurrent Neural Networks



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

Recurrent Neural Networks



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

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

new state

old state

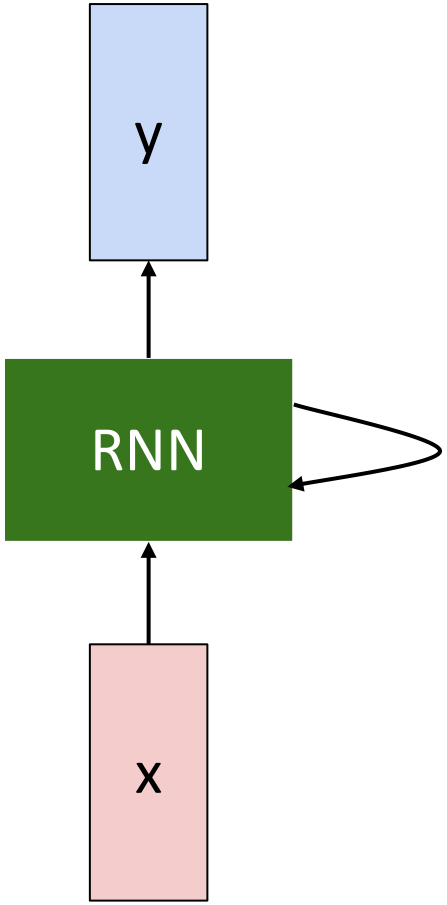
input vector at some time step

some function with parameters W

Slide from Justin Johnson

Recurrent Neural Networks

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



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

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

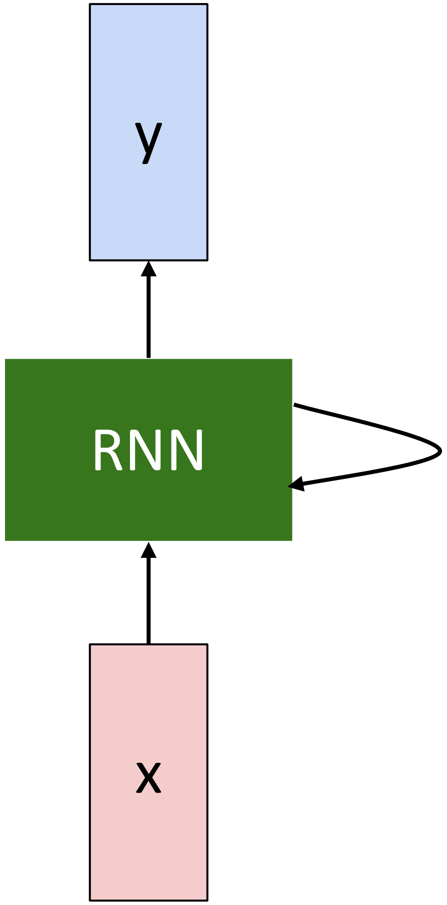
new state

old state

input vector at some time step

some function with parameters W

(Vanilla) Recurrent Neural Networks



The state consists of a single *"hidden"* vector **h**:

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

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

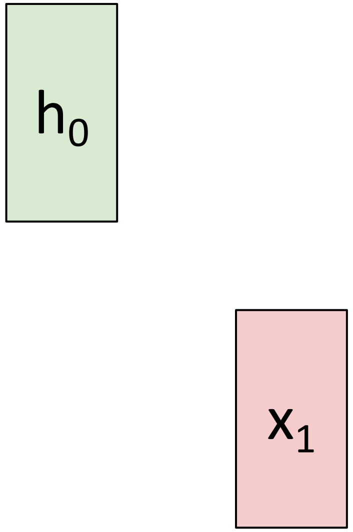
$$y_t = W_{hy}h_t + b_y$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

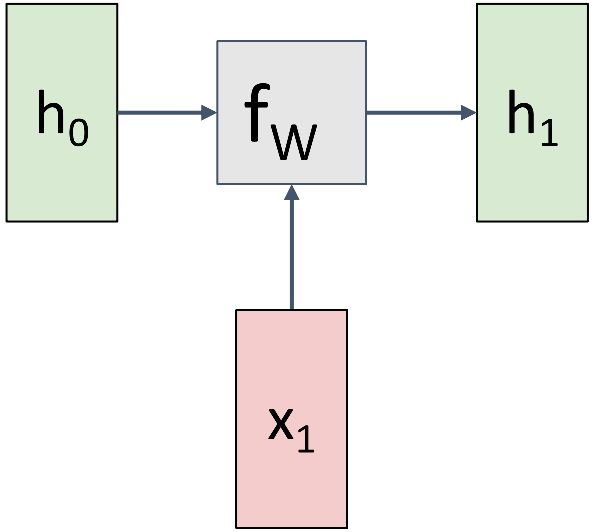
Slide from Justin Johnson

RNN Computational Graph

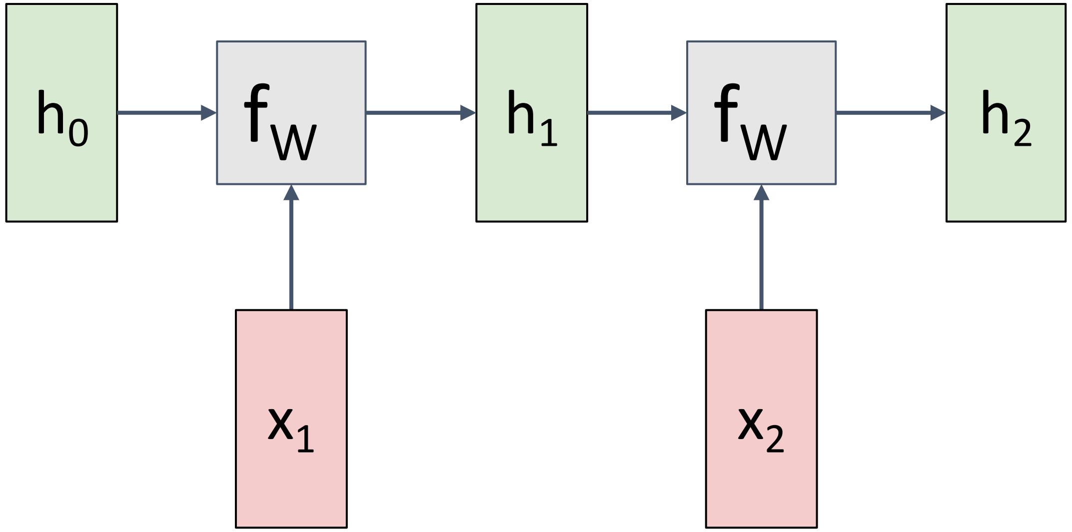
Initial hidden state
Either set to all 0,
Or learn it



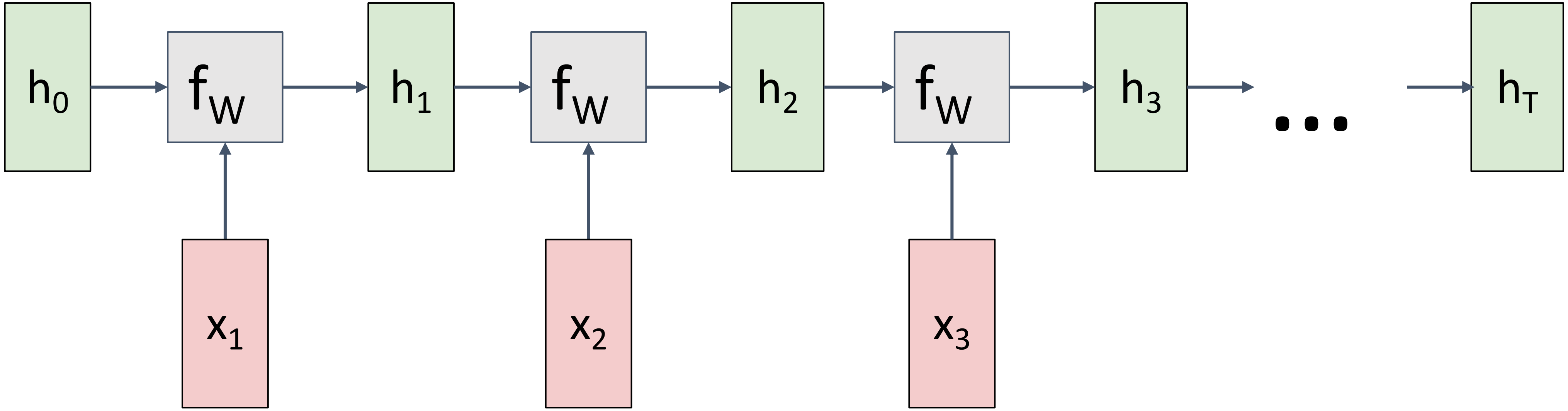
RNN Computational Graph



RNN Computational Graph

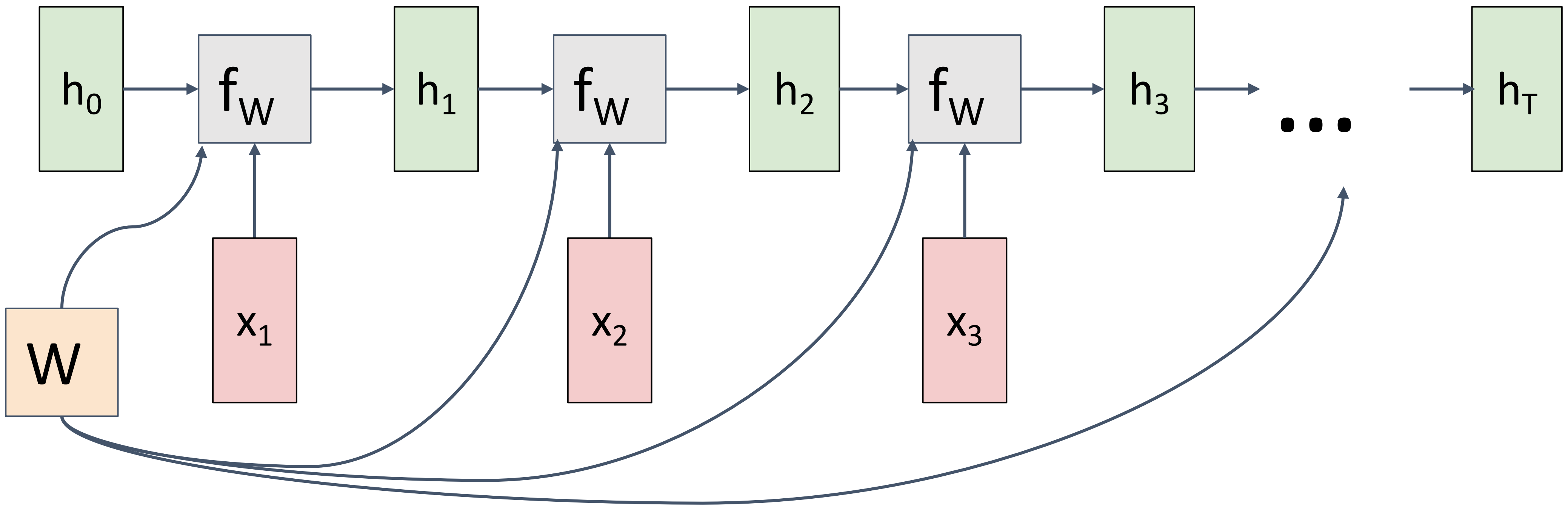


RNN Computational Graph



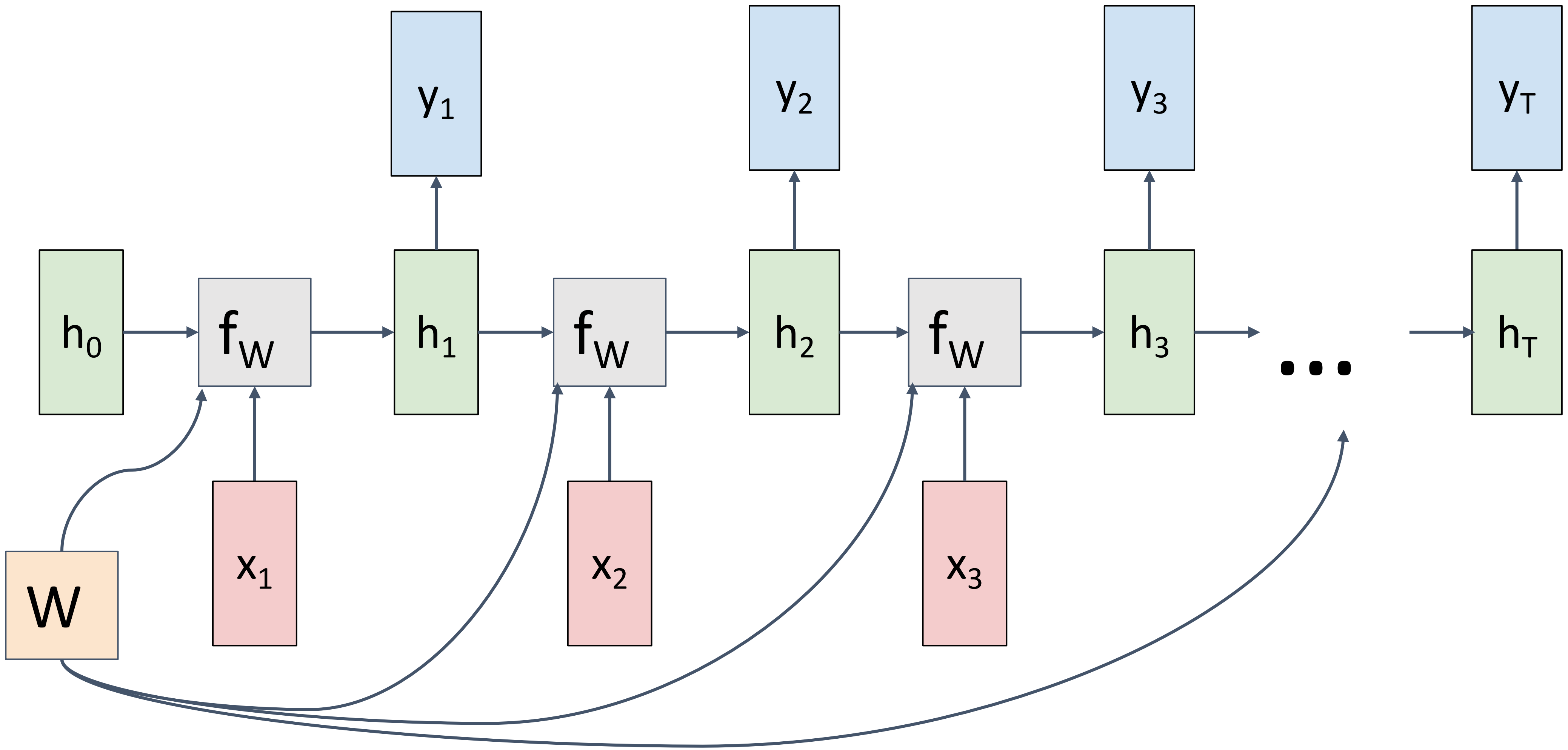
RNN Computational Graph

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



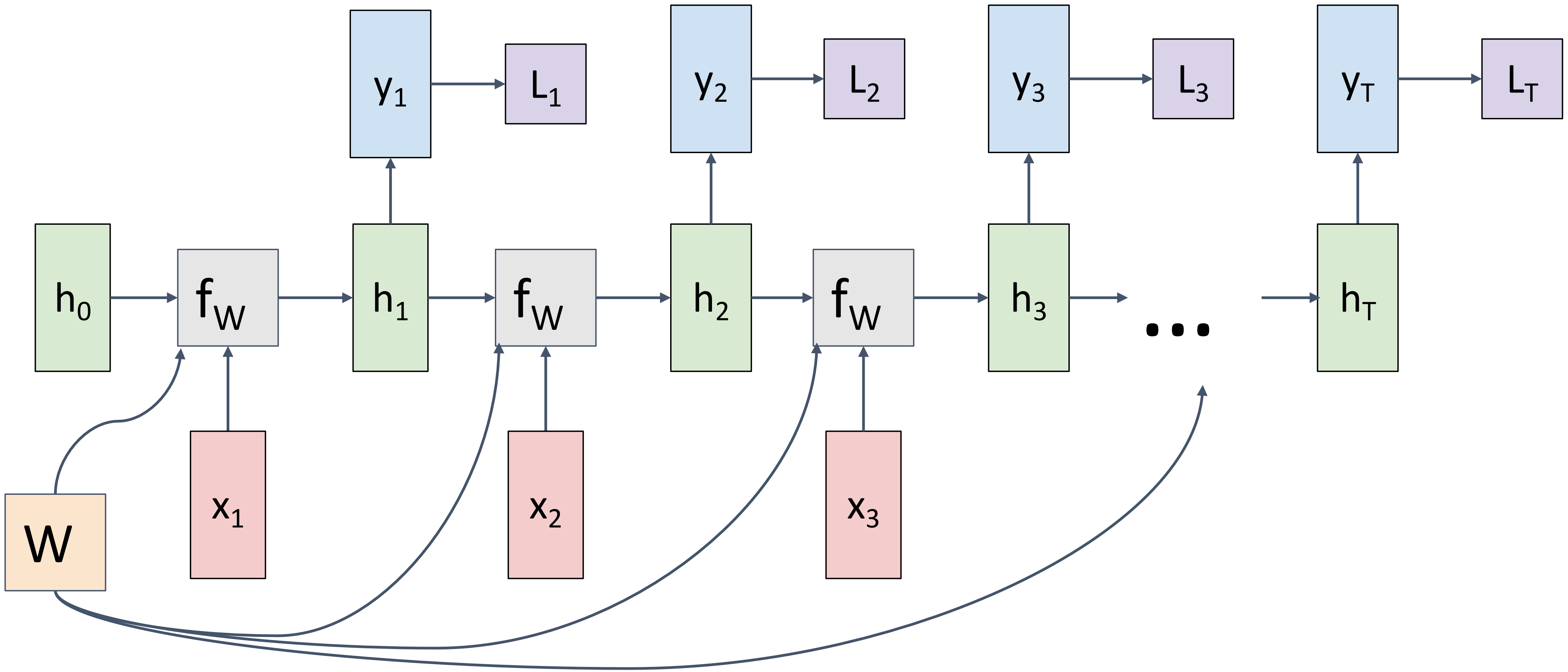
Slide from Justin Johnson

RNN Computational Graph (Many to Many)



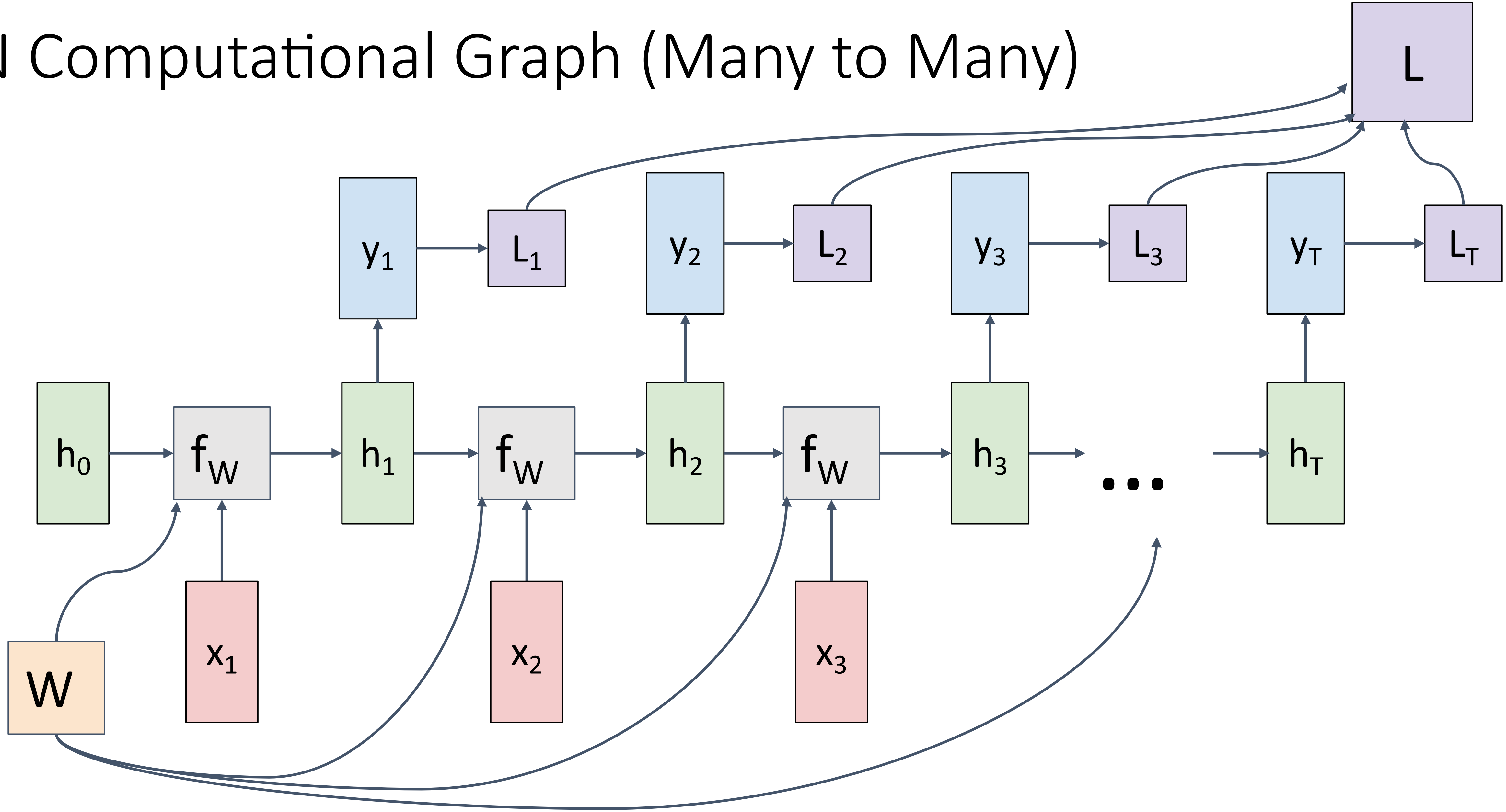
Slide from Justin Johnson

RNN Computational Graph (Many to Many)



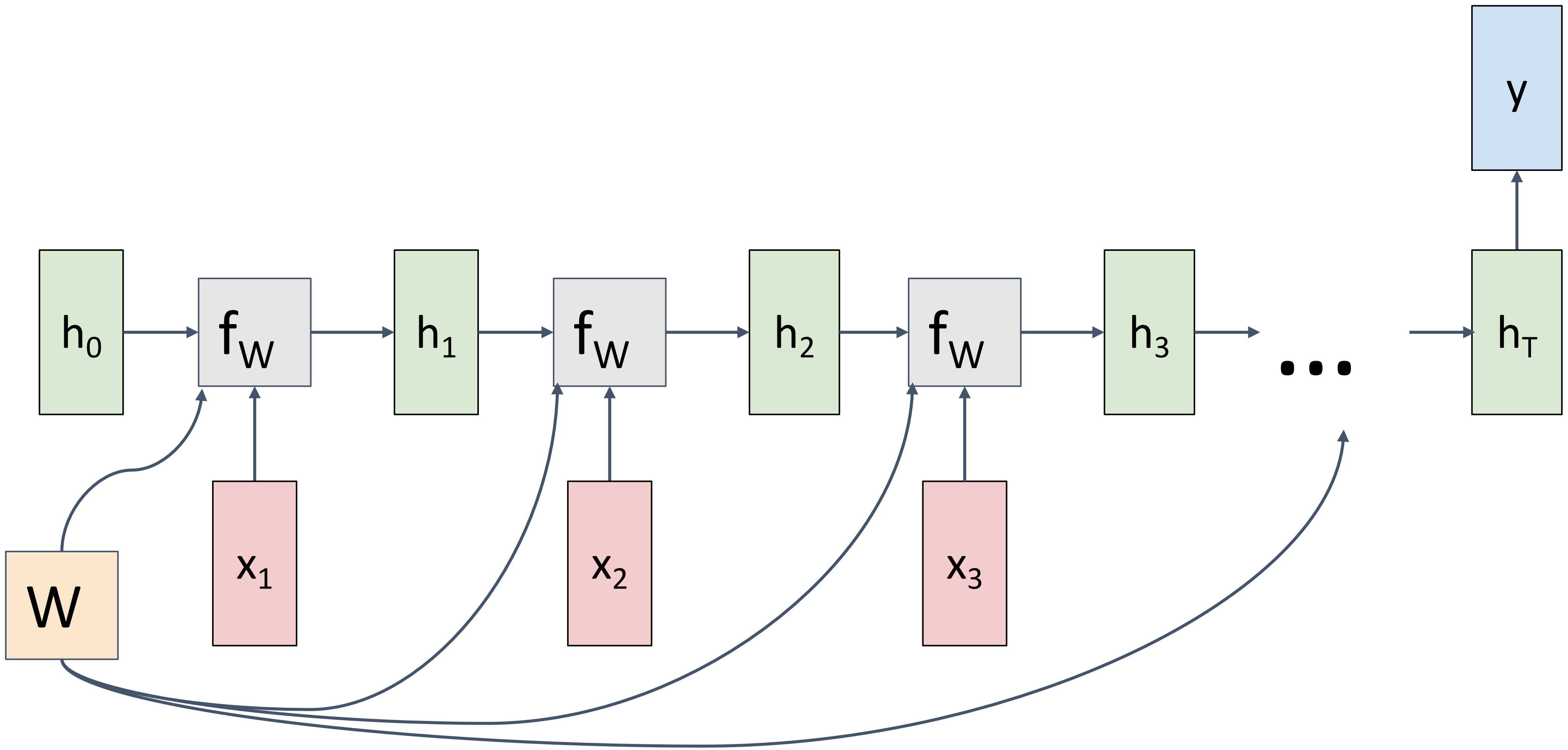
Slide from Justin Johnson

RNN Computational Graph (Many to Many)



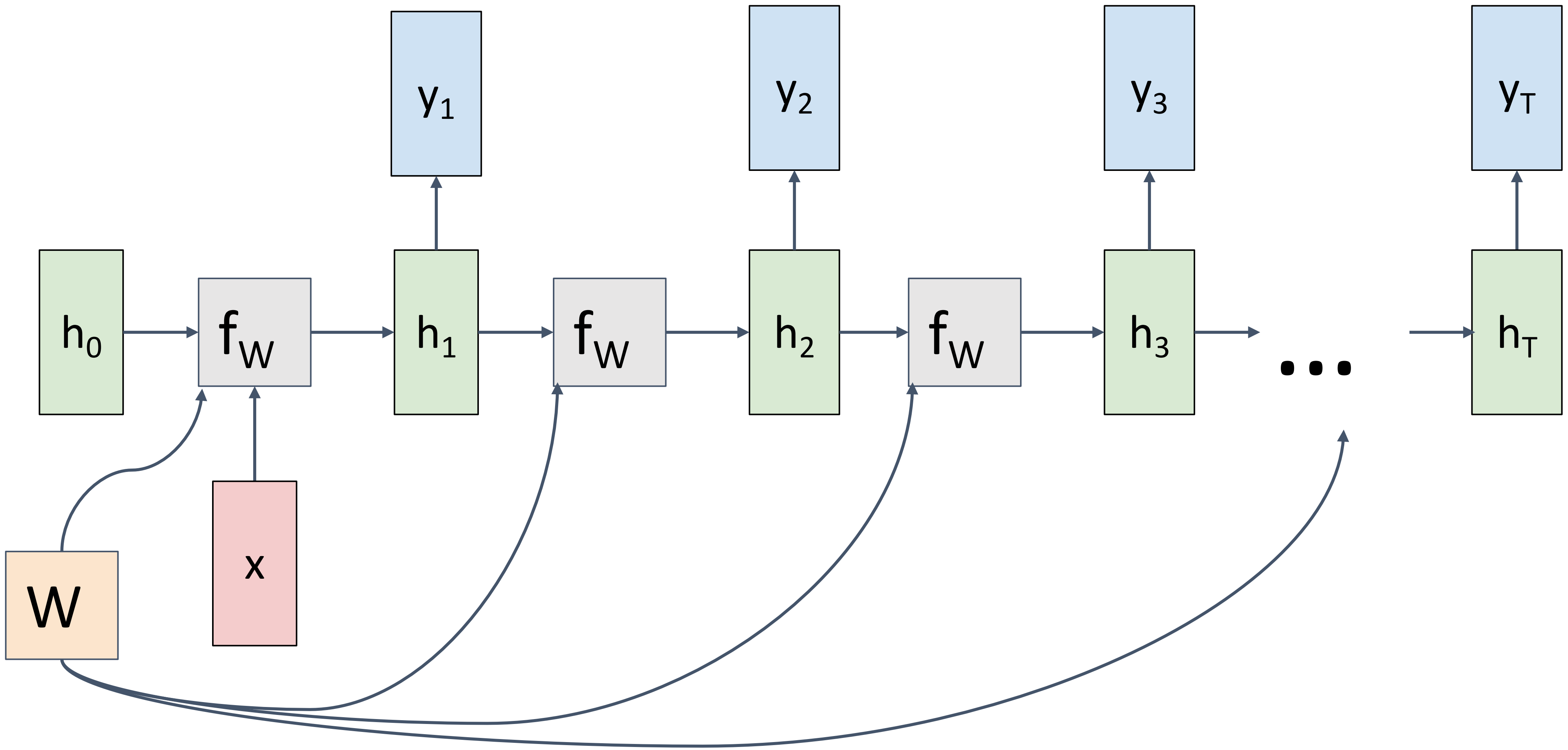
Slide from Justin Johnson

RNN Computational Graph (Many to One)



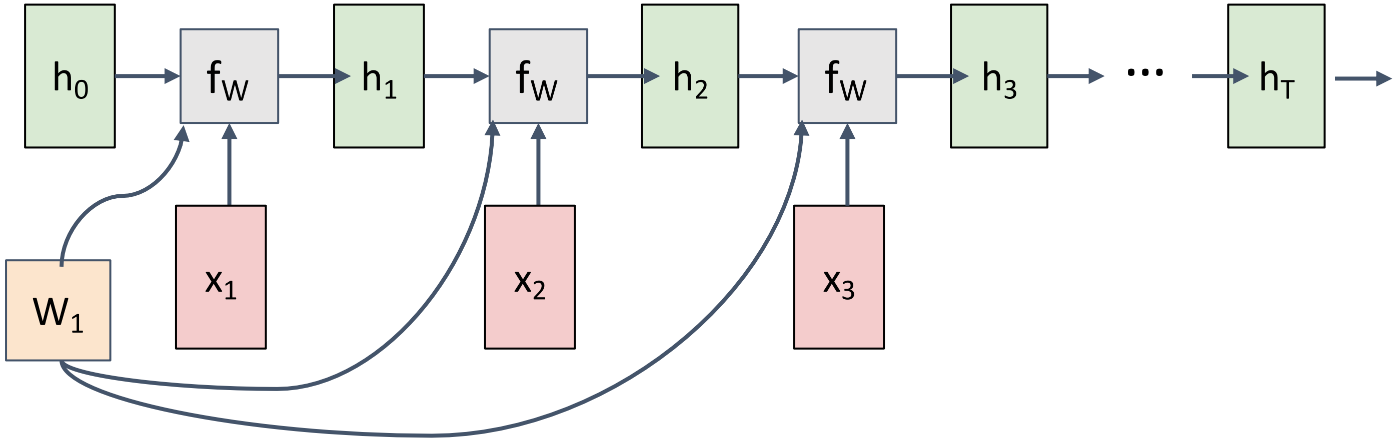
Slide from Justin Johnson

RNN Computational Graph (One to Many)



Sequence to Sequence (seq2seq)
(Many to one) + (One to many)

Many to one: Encode input sequence in a single vector

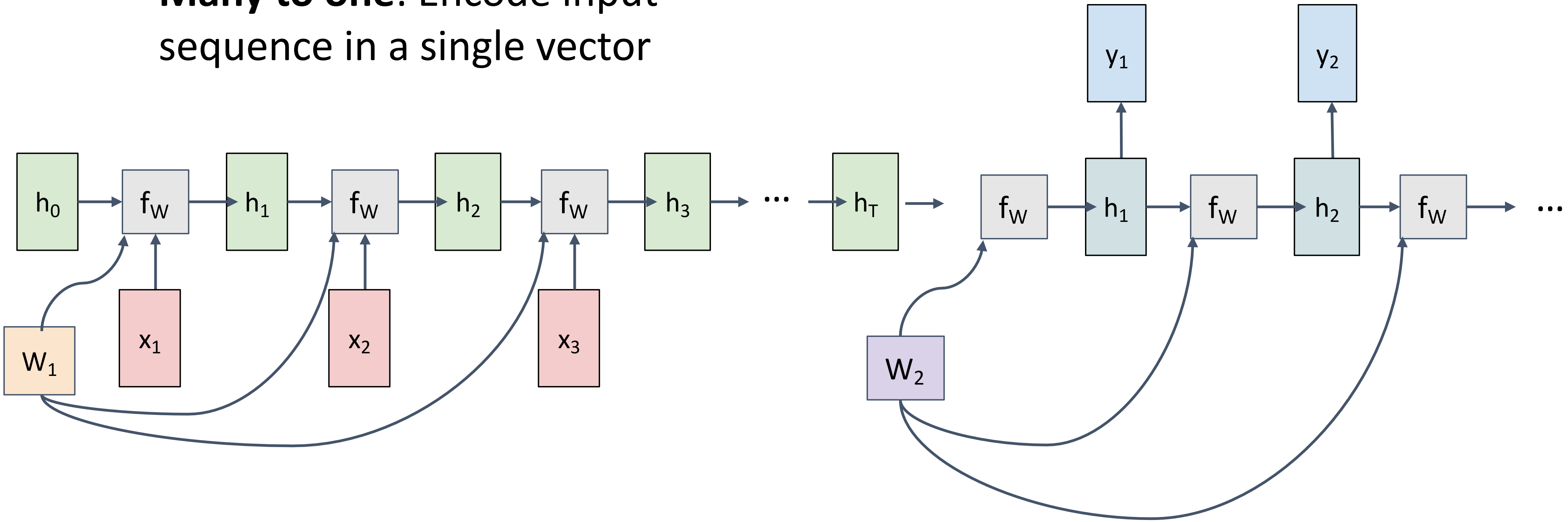


Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

Sequence to Sequence (seq2seq)
(Many to one) + (One to many)

One to many: Produce output sequence from single input vector

Many to one: Encode input sequence in a single vector



Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

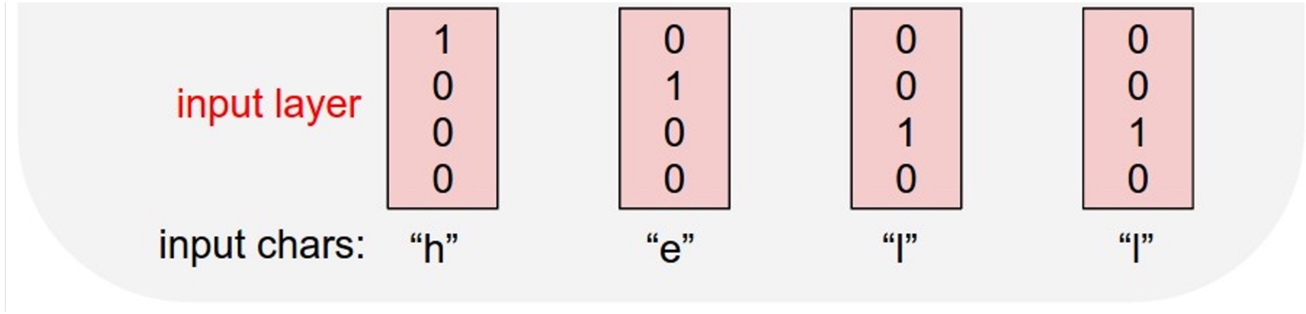
Slide from Justin Johnson

Example: Language Modeling

Given characters 1, 2, …, t-1, model predicts character t

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



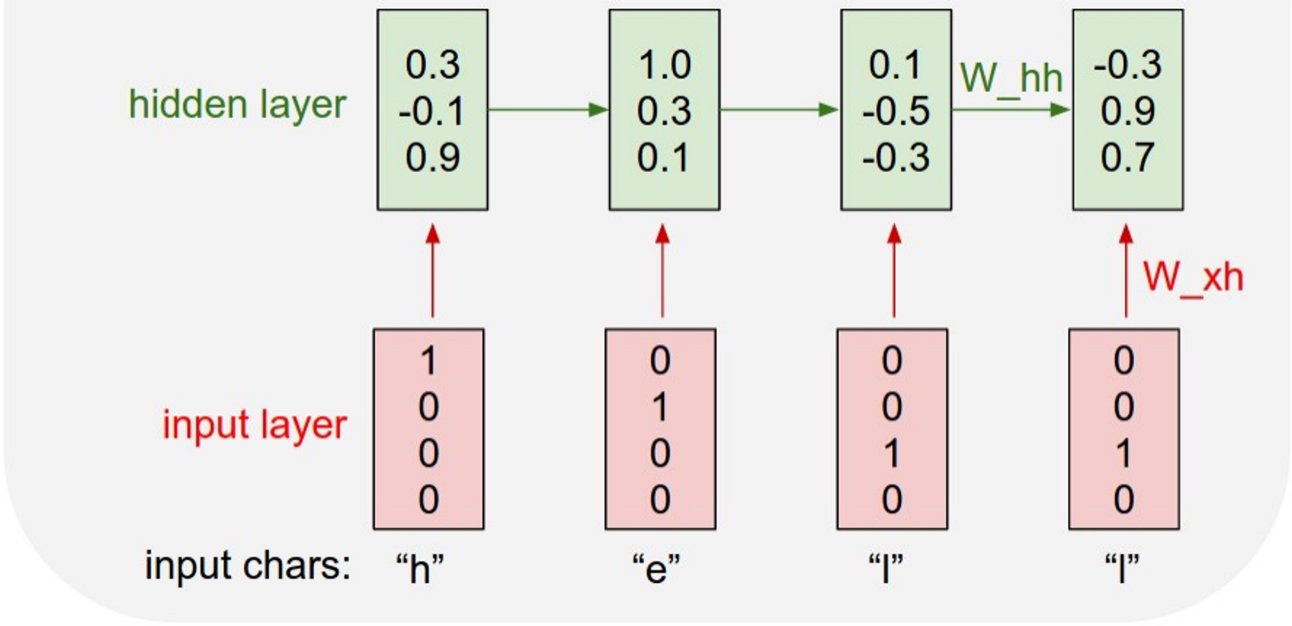
Example: Language Modeling

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



Slide from Justin Johnson

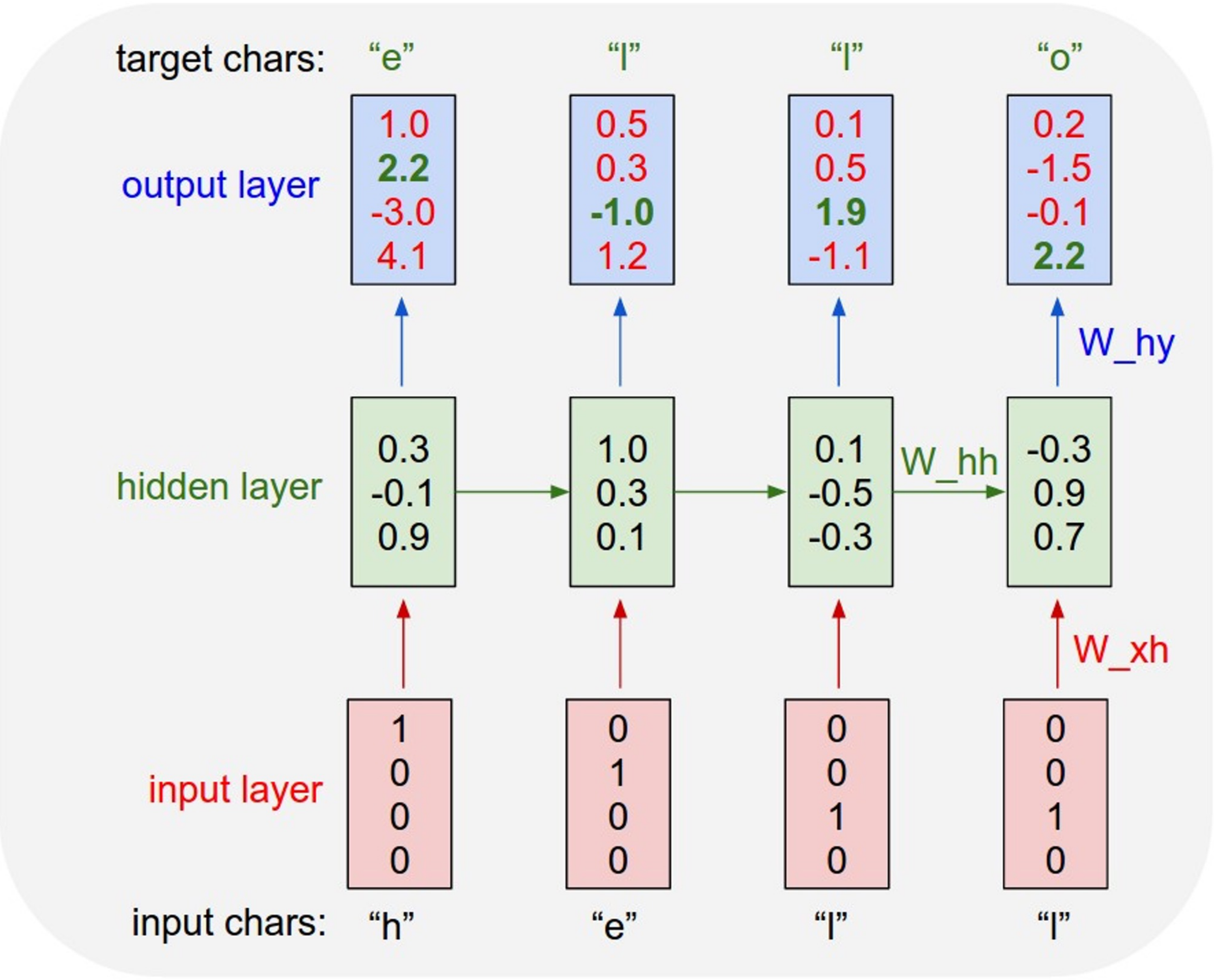
Example: Language Modeling

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



Slide from Justin Johnson

Example: Language Modeling

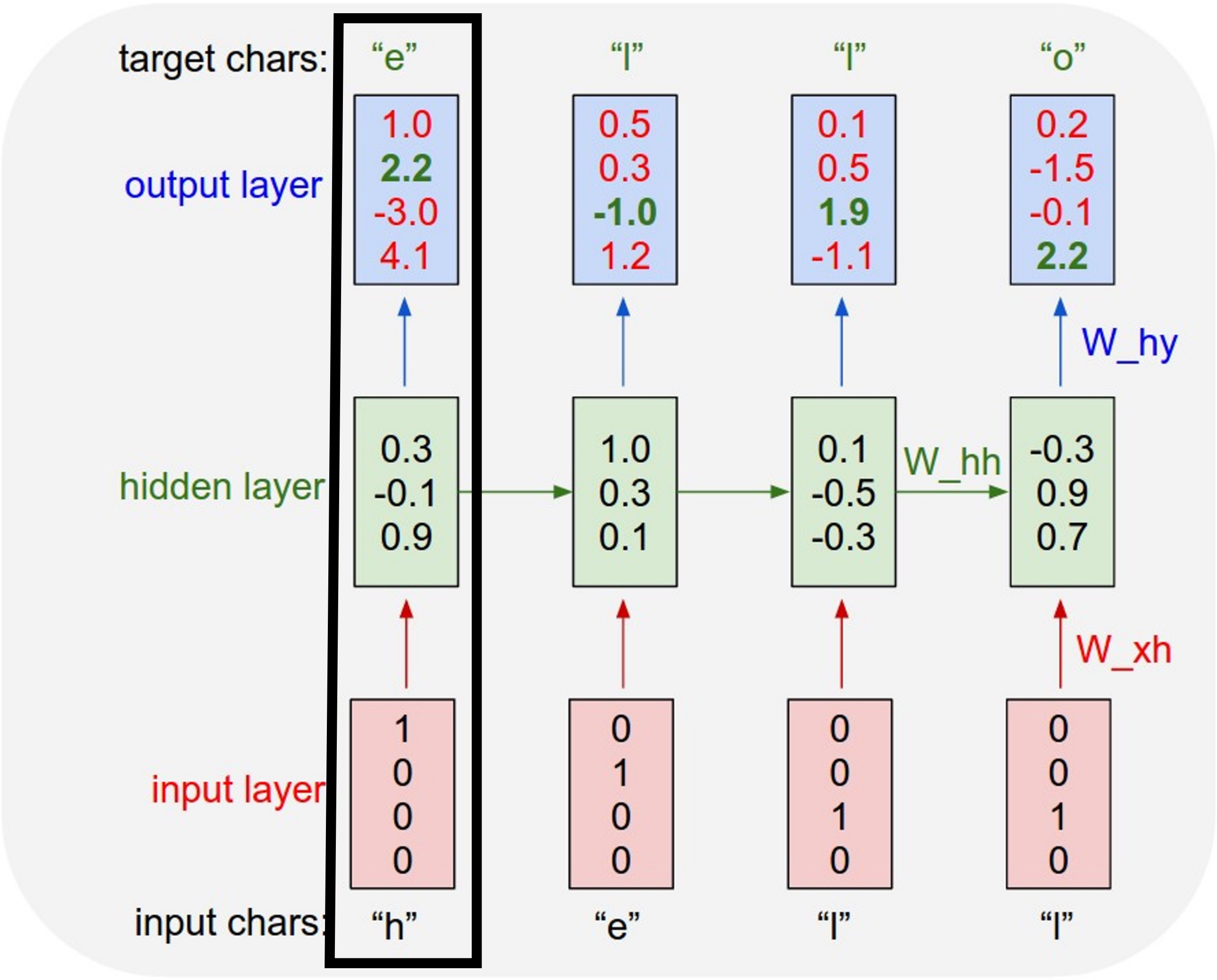
Given "h", predict "e"

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: "hello"

Vocabulary: [h, e, l, o]



Example: Language Modeling

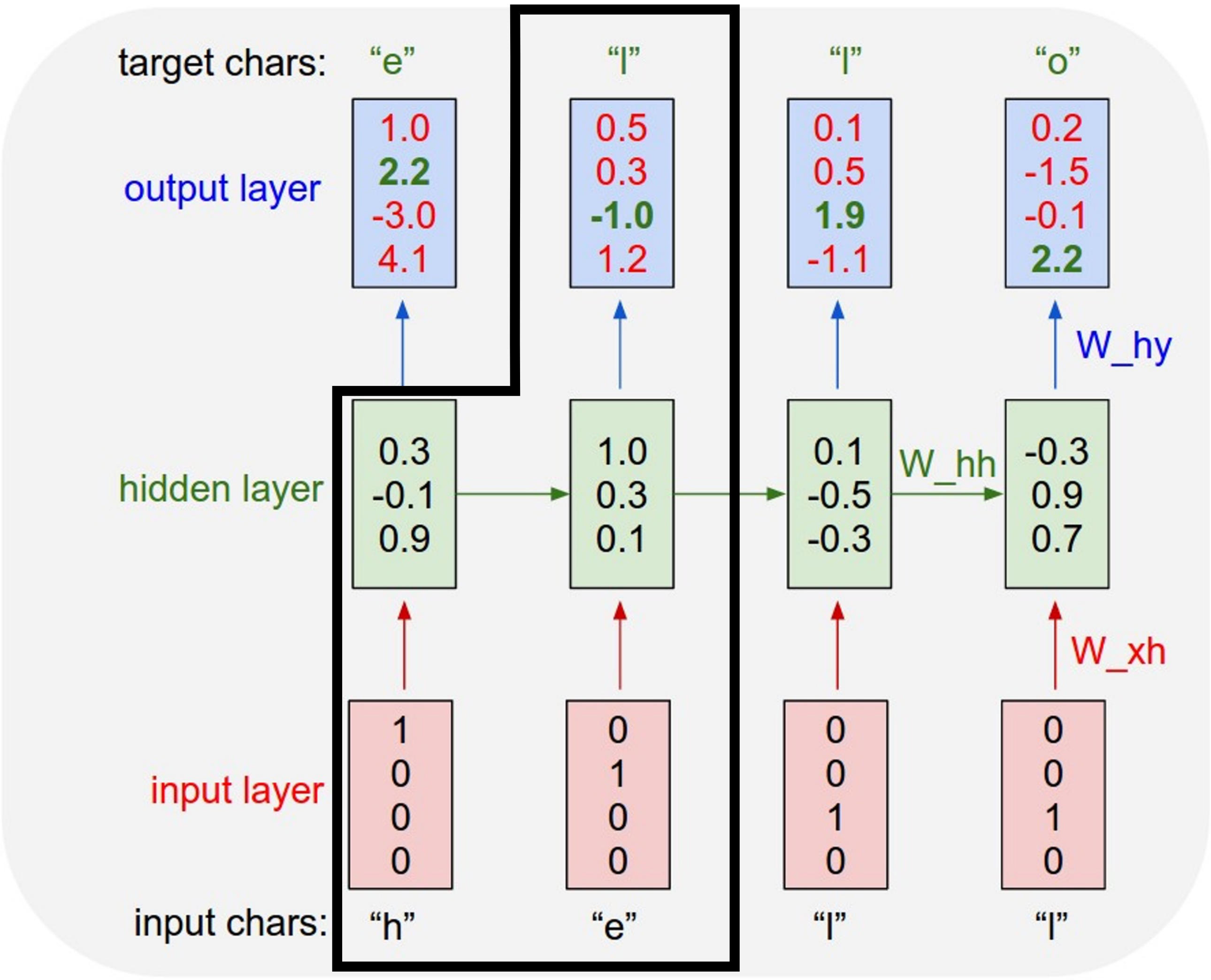
Given "he", predict "l"

Given characters 1, 2, ..., t-1, model predicts character t

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

Training sequence: "hello"

Vocabulary: [h, e, l, o]



Slide from Justin Johnson

Example: Language Modeling

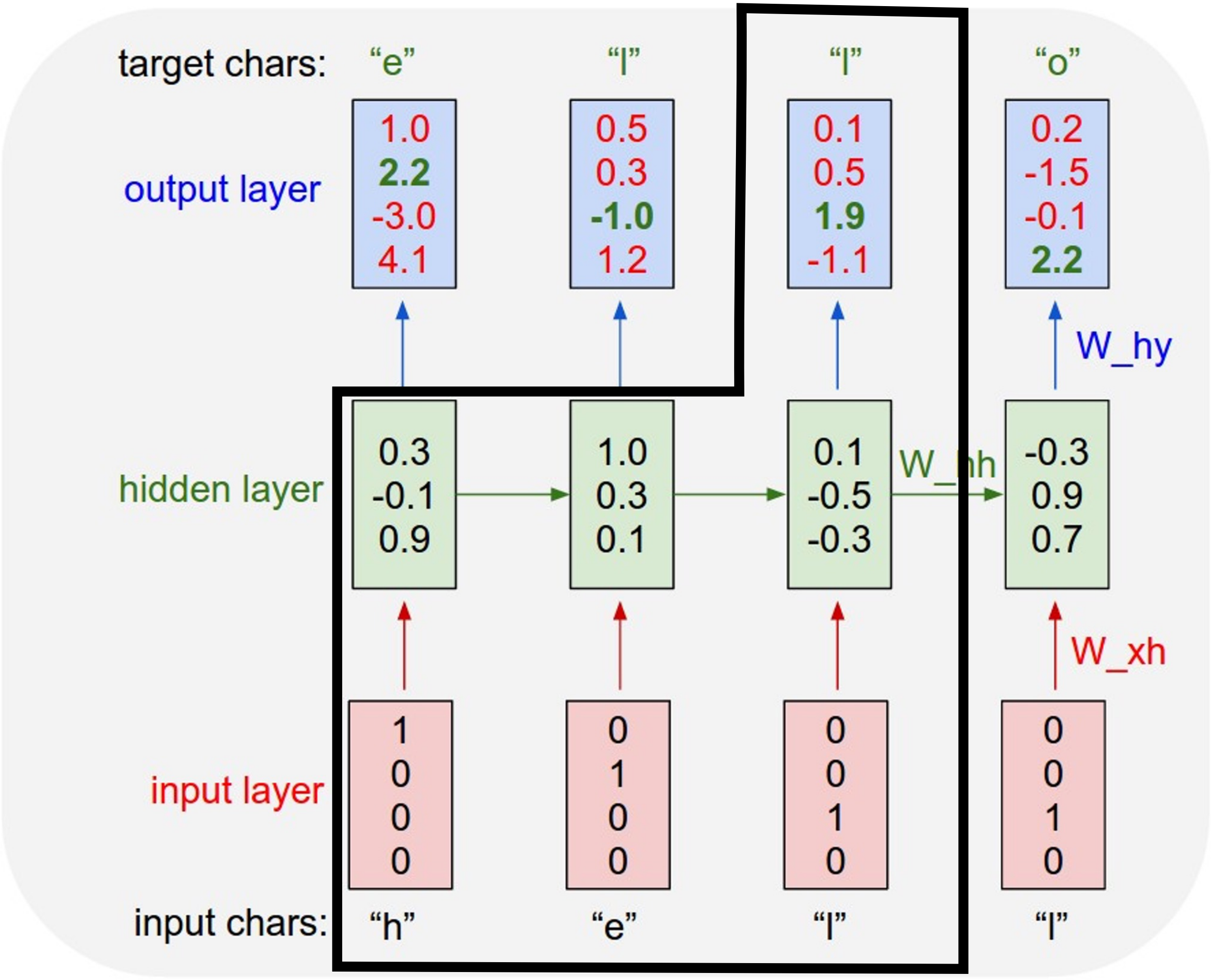
Given "hel", predict "l"

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: "hello"

Vocabulary: [h, e, l, o]



Slide from Justin Johnson

Example: Language Modeling

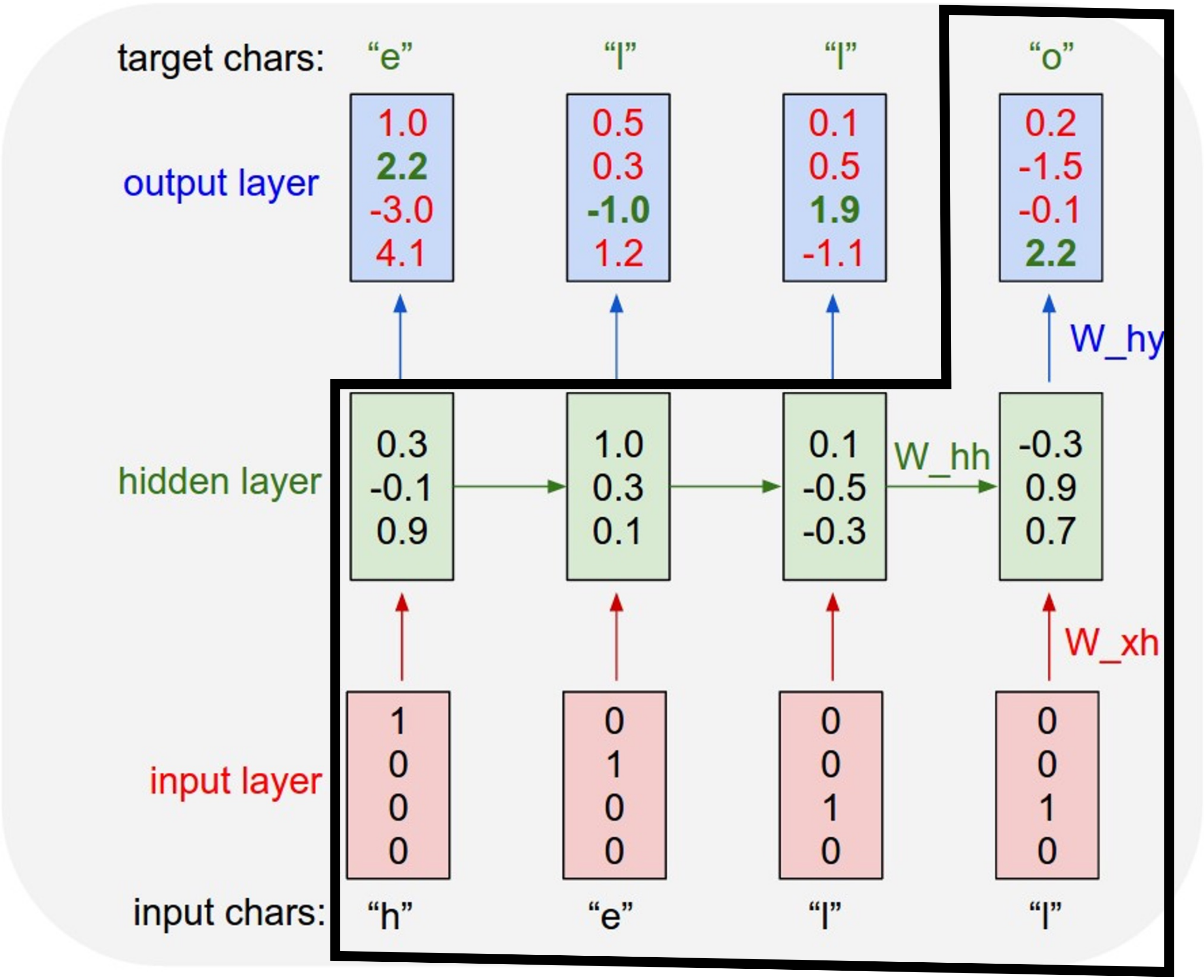
Given "hell", predict "o"

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: "hello"

Vocabulary: [h, e, l, o]

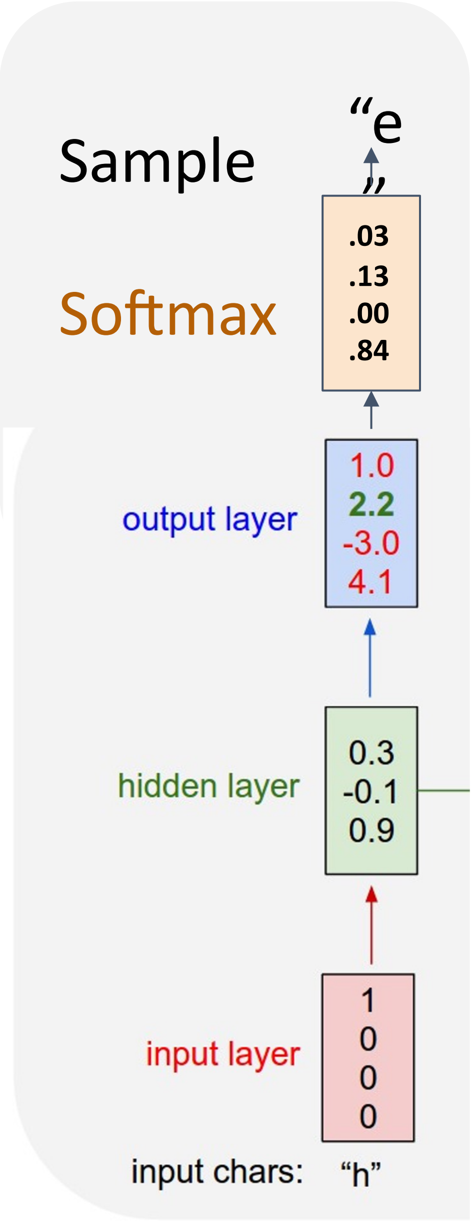


Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

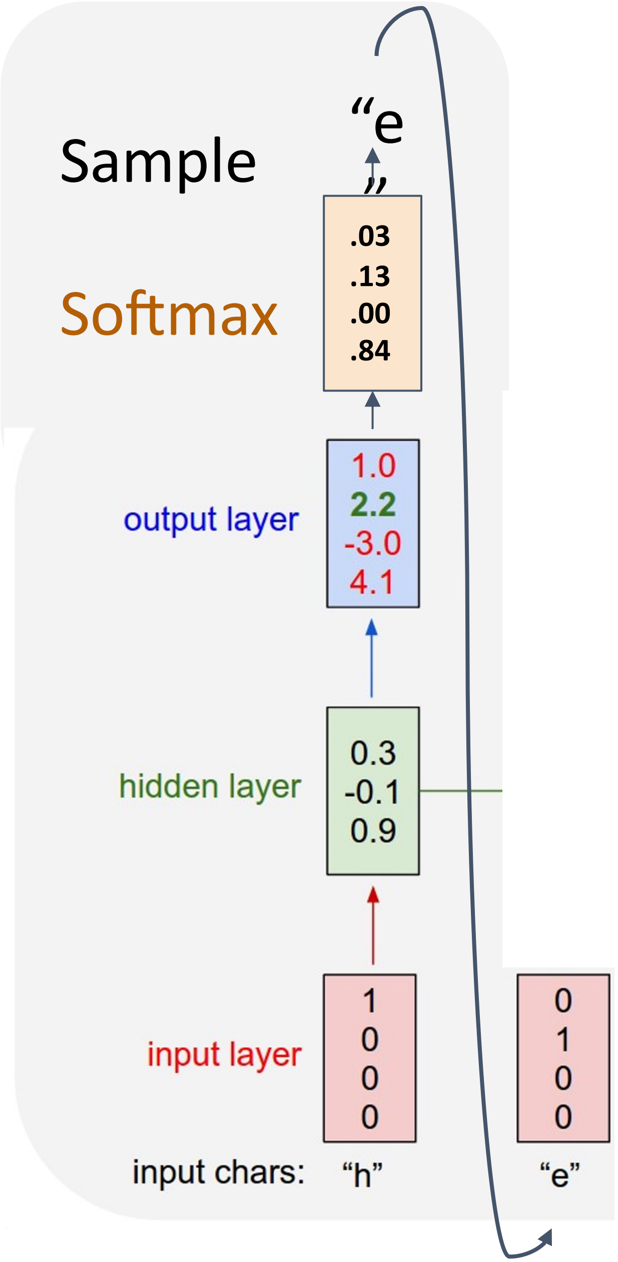


Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



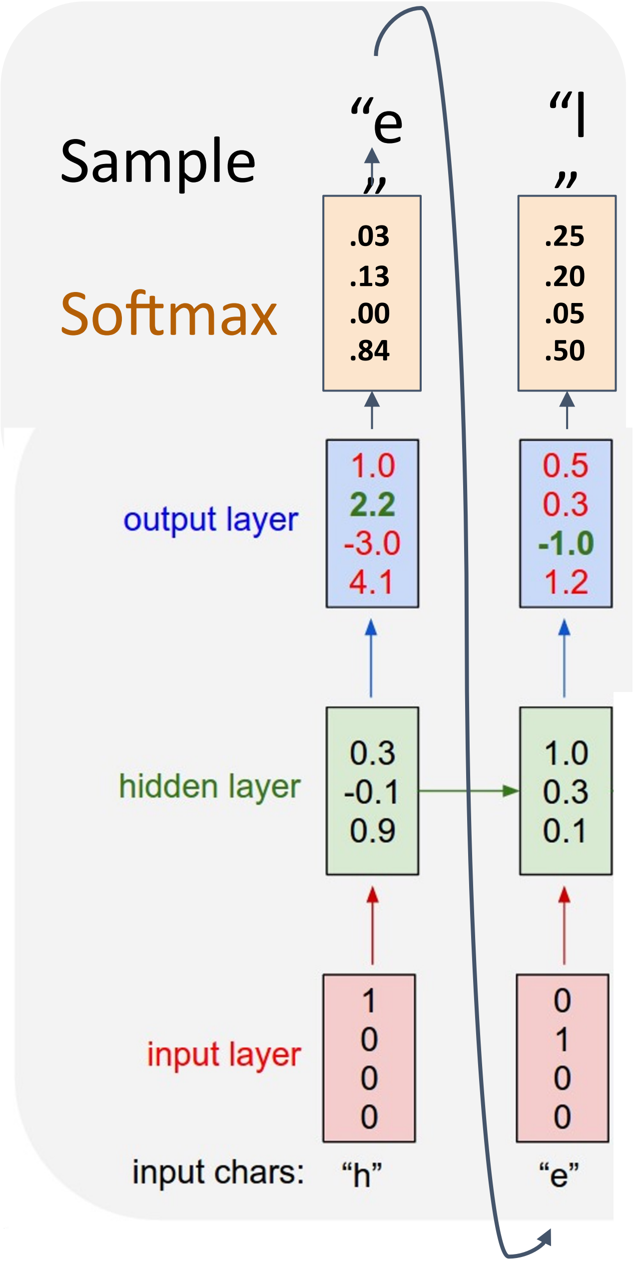
Slide from Justin Johnson

Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

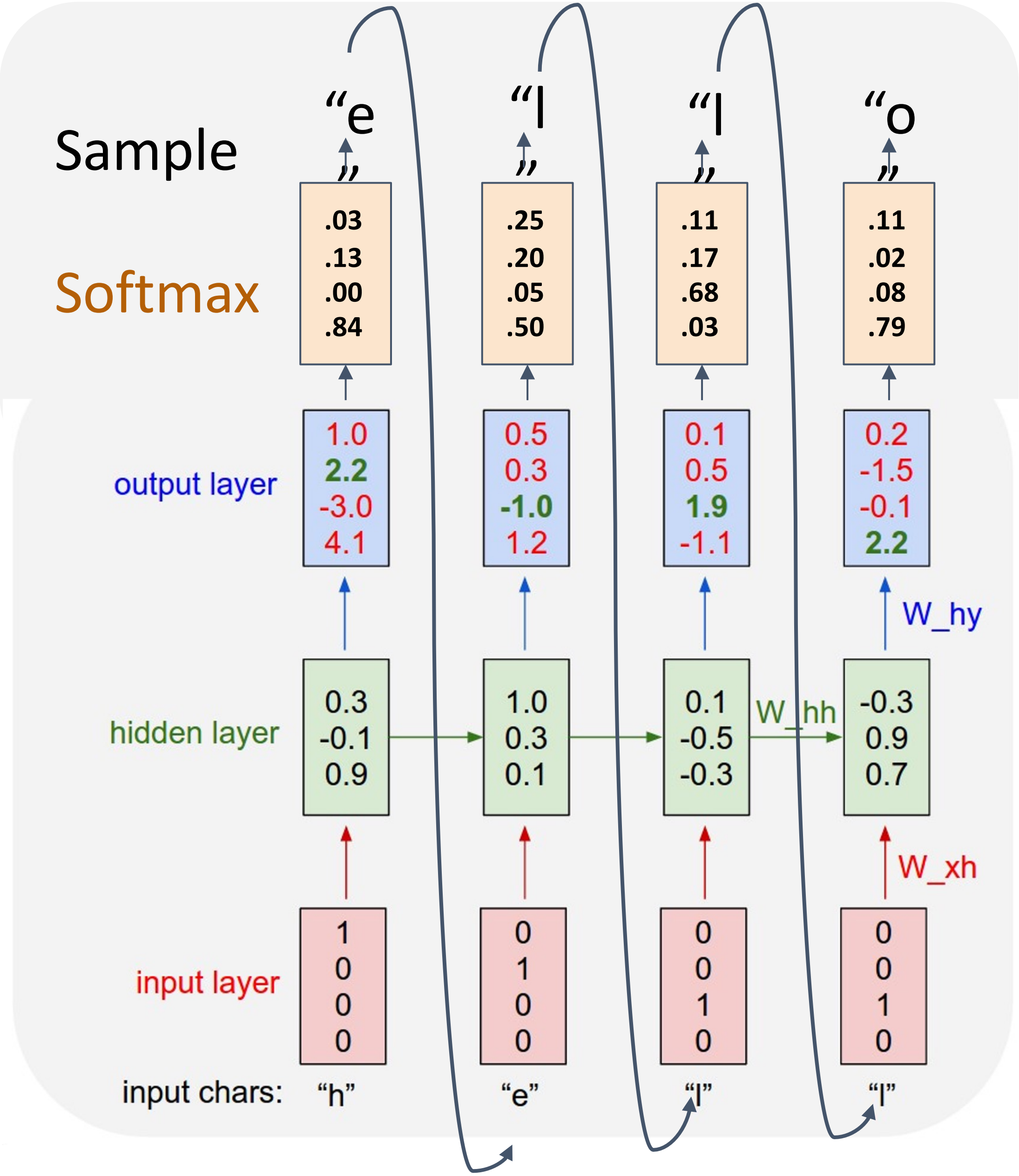


Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

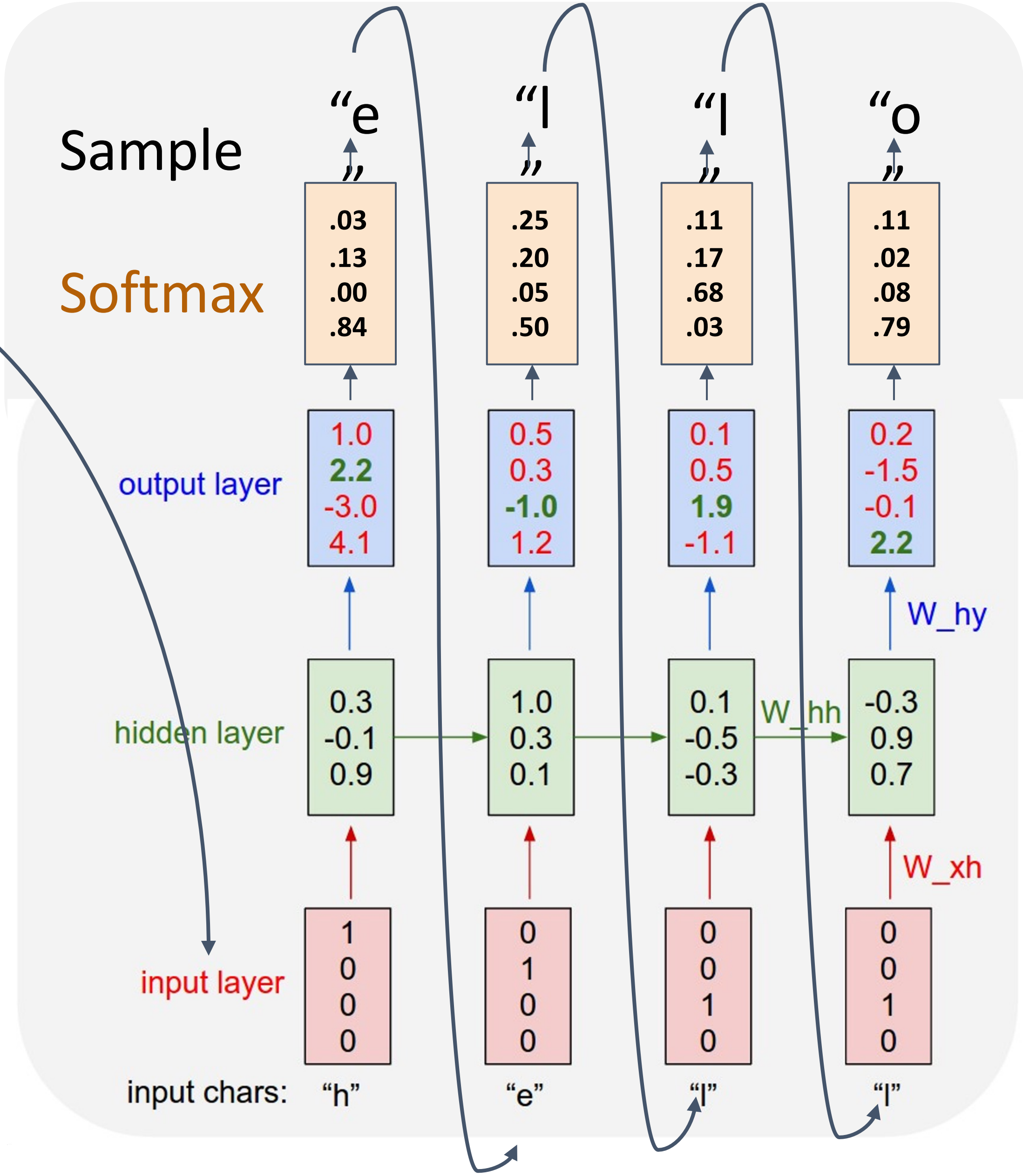


Example: Language Modeling

So far: encode inputs as **one-hot-vector**

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. Often extract this into a separate **embedding** layer



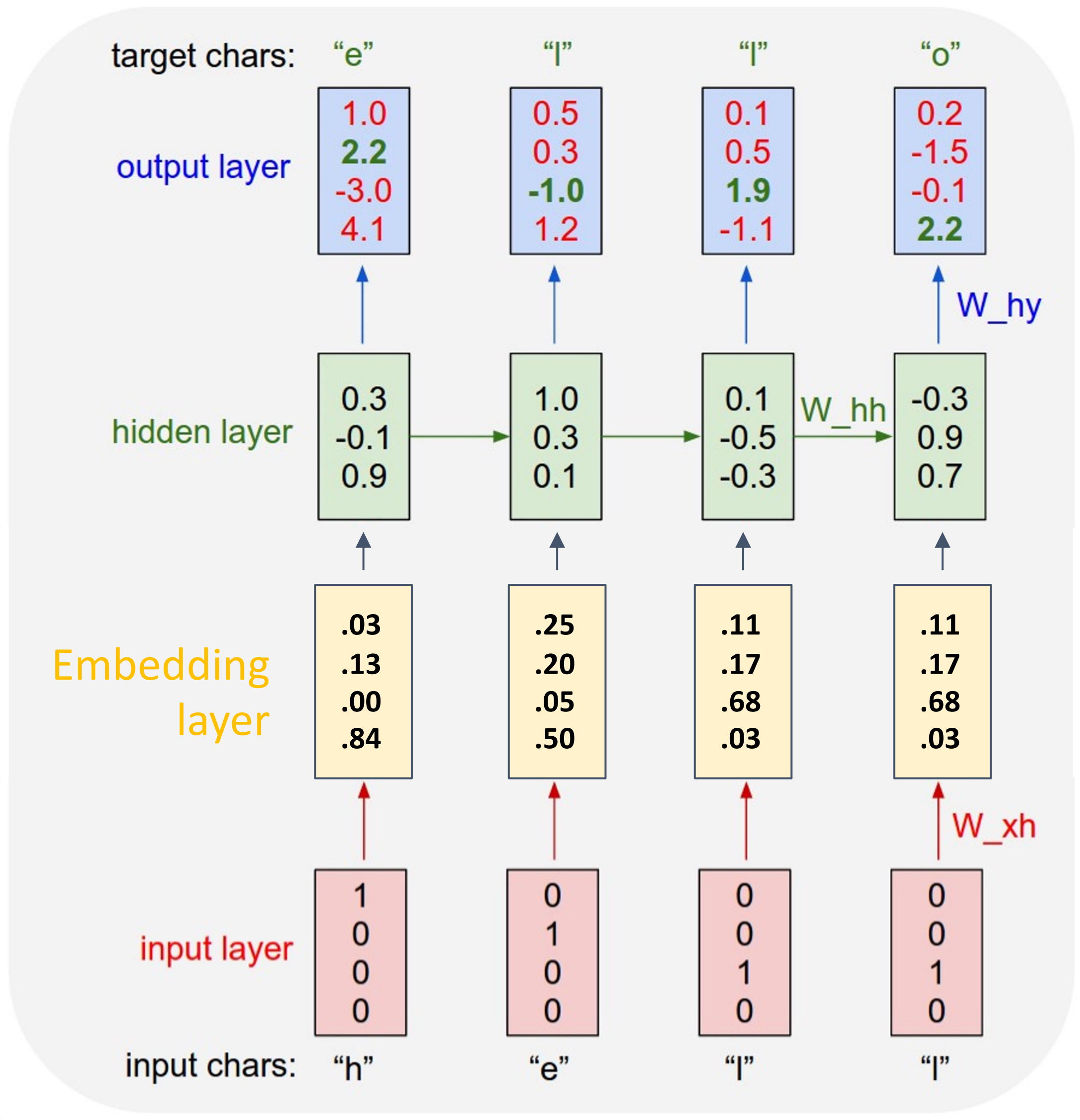
Slide from Justin Johnson

Example: Language Modeling

So far: encode inputs as **one-hot-vector**

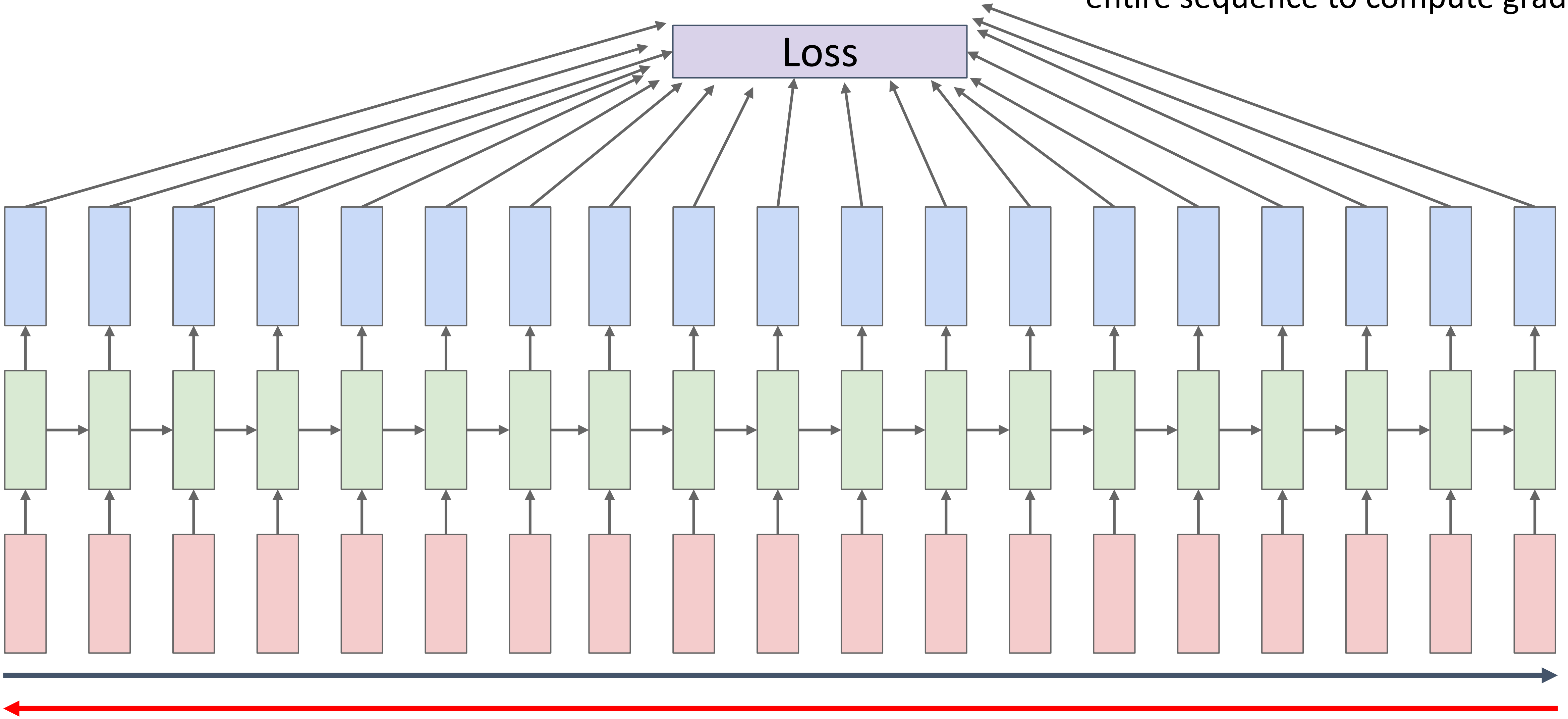
$$\begin{matrix}[w_{11}\ w_{12}\ w_{13}\ w_{14}] \\ [w_{21}\ w_{22}\ w_{23}\ w_{14}] \\ [w_{31}\ w_{32}\ w_{33}\ w_{14}] \\ \end{matrix}\ \begin{matrix}[1] \\ [0] \\ [0] \\ [0]\end{matrix} \ = \ \begin{matrix}[w_{11}] \\ [w_{21}] \\ [w_{31}] \\ \end{matrix}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. Often extract this into a separate **embedding** layer



Backpropagation Through Time

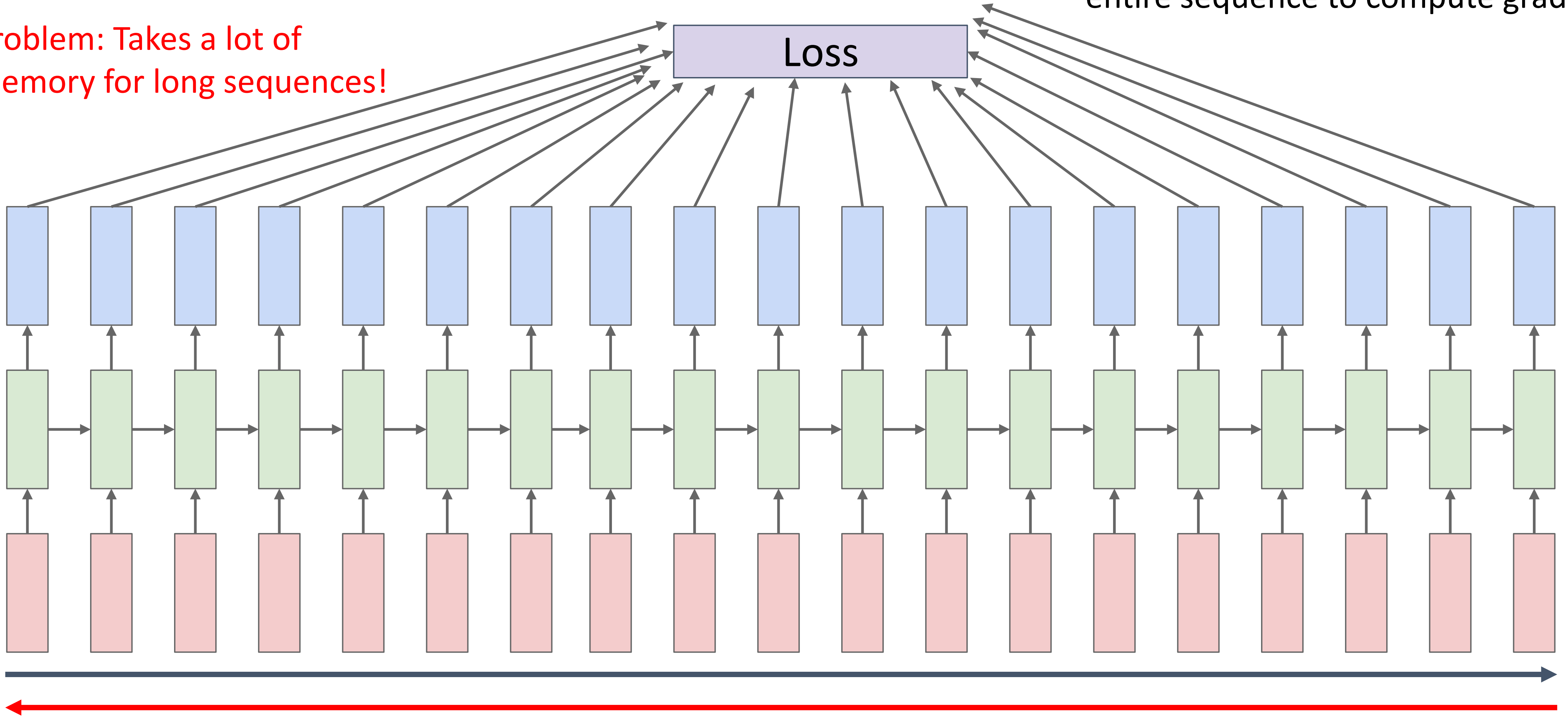
Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient



Backpropagation Through Time

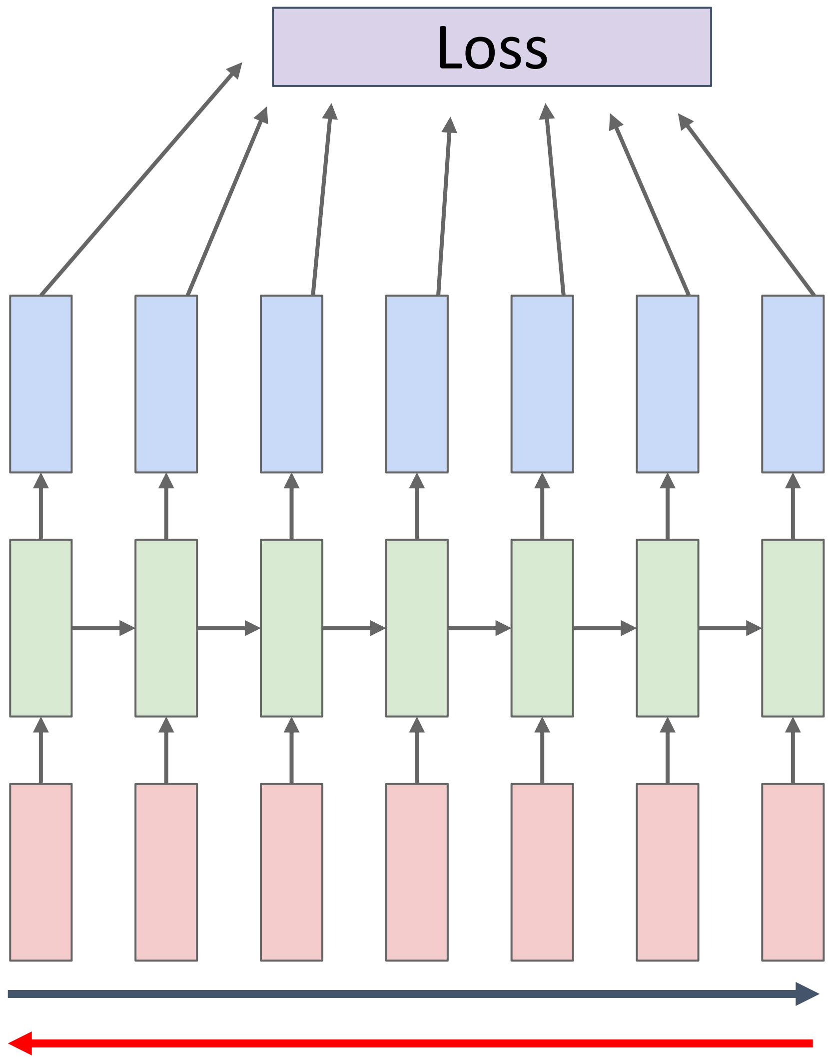
Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

Problem: Takes a lot of memory for long sequences!



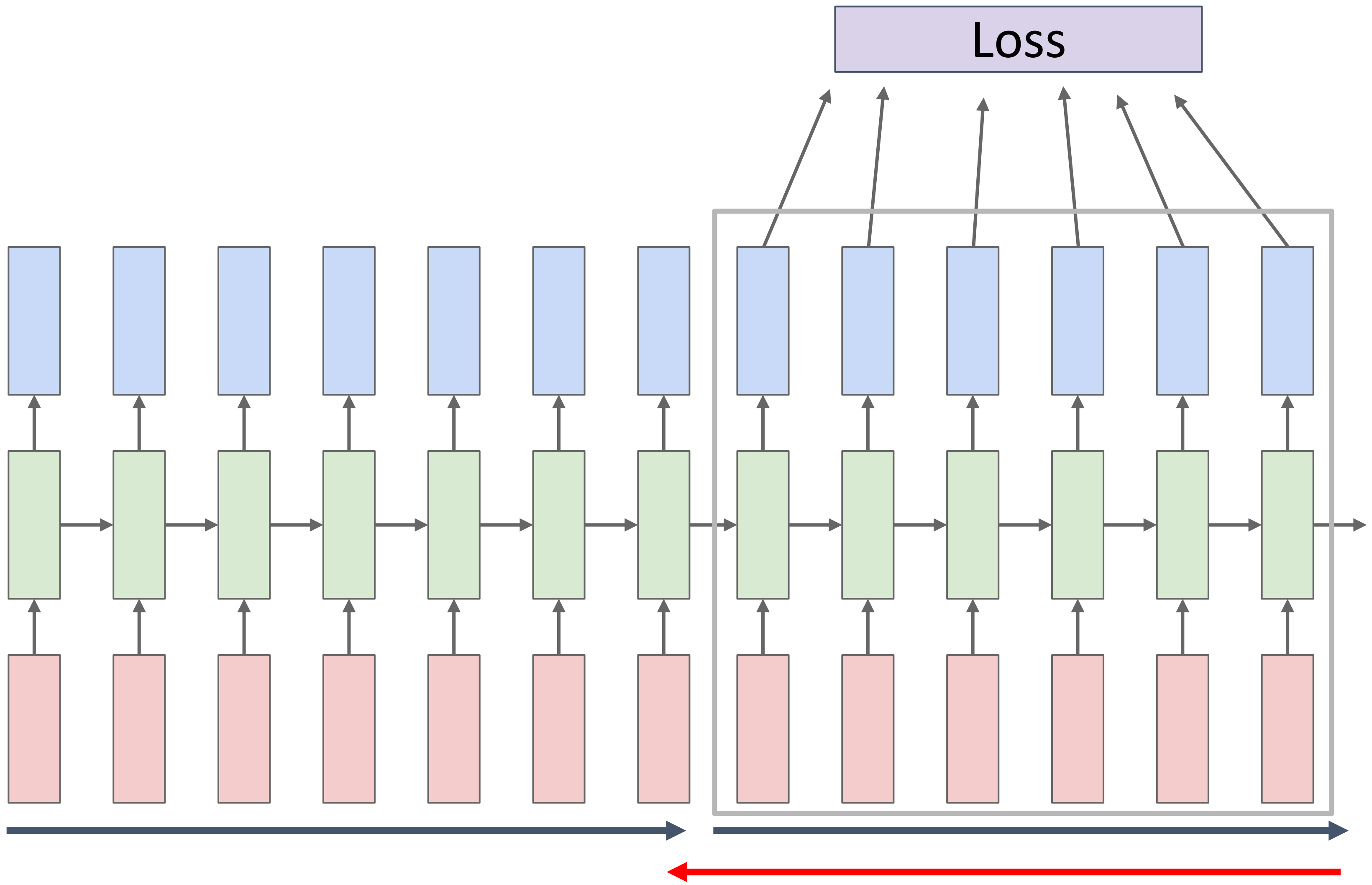
Slide from Justin Johnson

Truncated Backpropagation Through Time



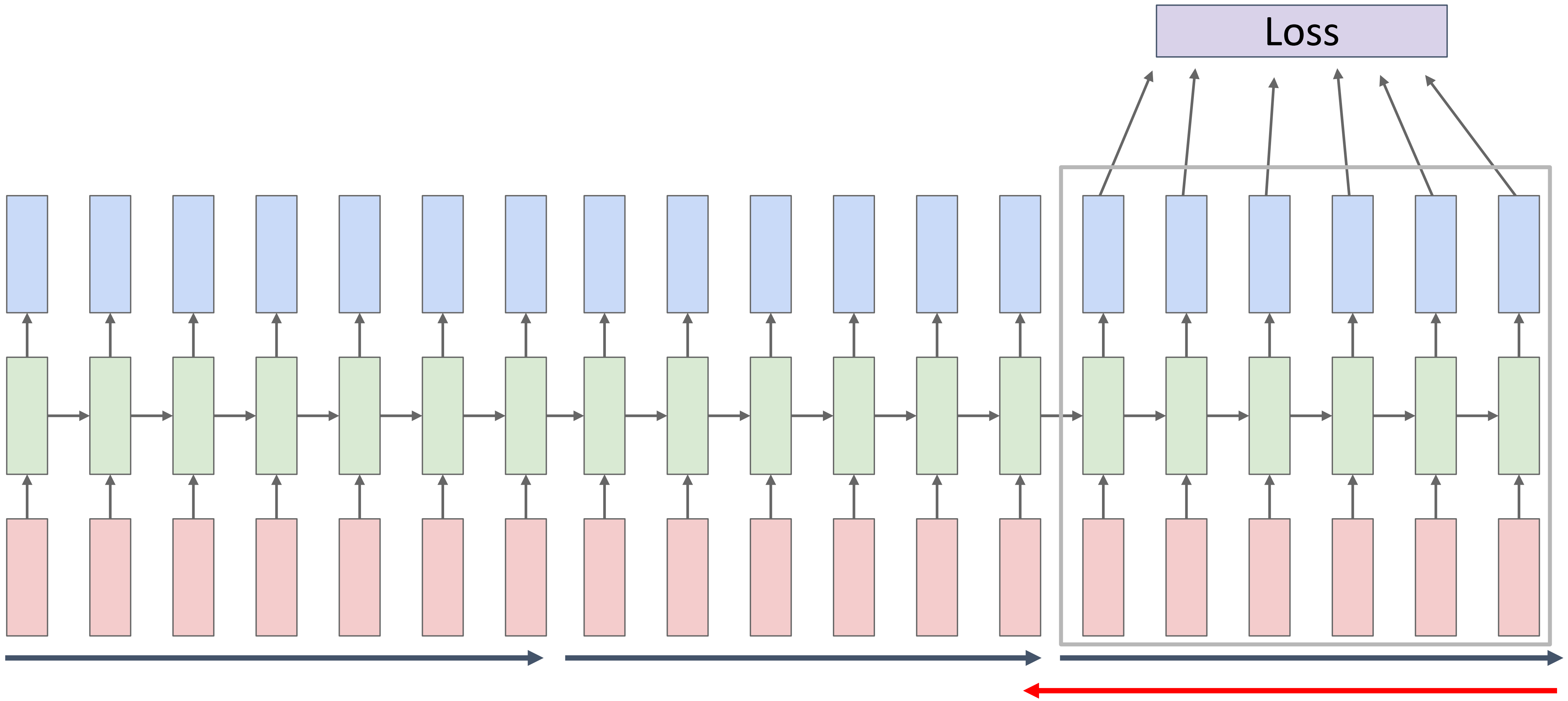
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation Through Time



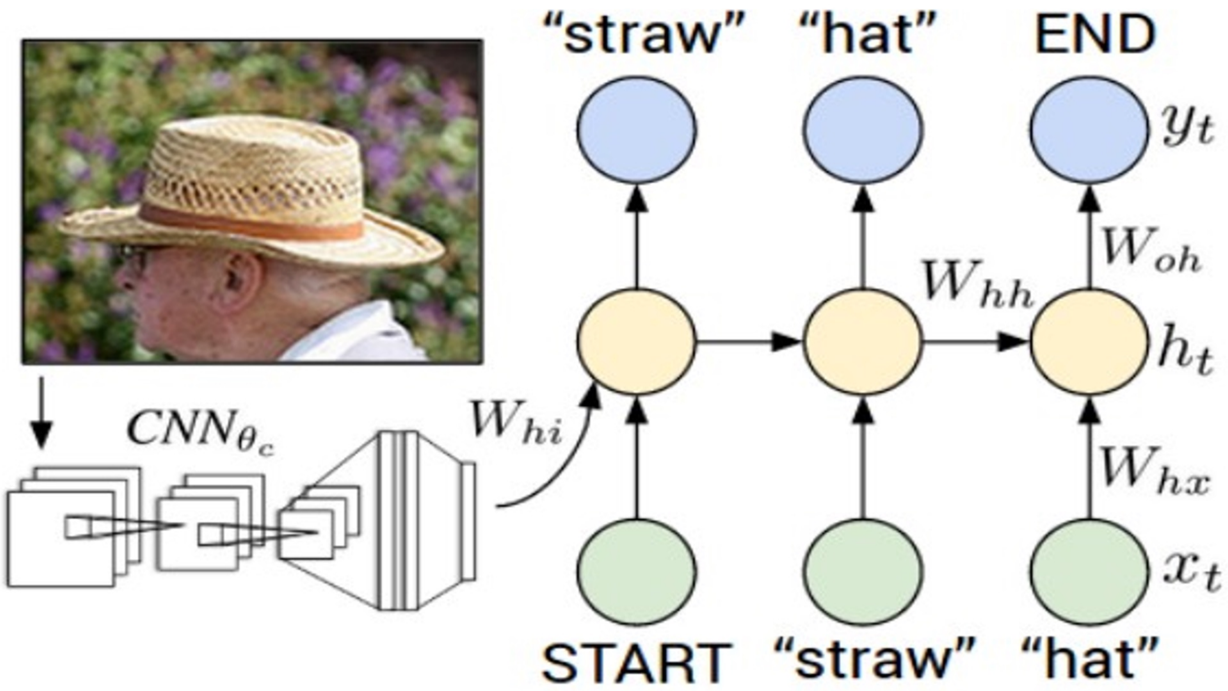
Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Truncated Backpropagation Through Time



Slide from Justin Johnson

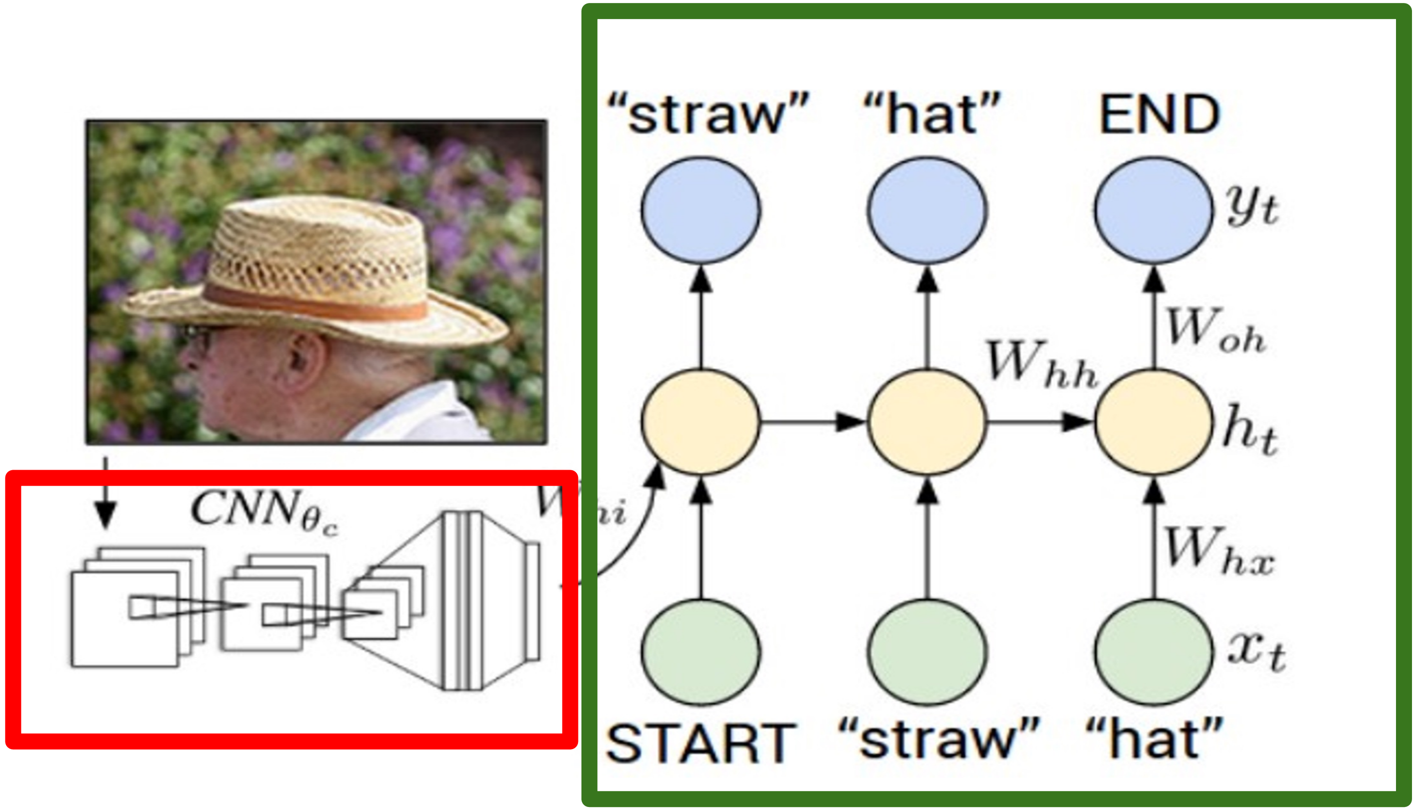
Example: Image Captioning



Mao et al, "Explain Images with Multimodal Recurrent Neural Networks", NeurIPS 2014 Deep Learning and Representation Workshop
Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Vinyals et al, "Show and Tell: A Neural Image Caption Generator", CVPR 2015
Donahue et al, "Long-term Recurrent Convolutional Networks for Visual Recognition and Description", CVPR 2015
Chen and Zitnick, "Learning a Recurrent Visual Representation for Image Caption Generation", CVPR 2015

Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

Example: Image Captioning



Recurrent Neural Network

Convolutional Neural Network

Figure from Karpathy et a, “Deep Visual-Semantic Alignments for Generating Image Descriptions”, CVPR 2015

Transfer learning: Take CNN trained on ImageNet, chop off last layer

Slide from Justin Johnson

This image is CC0 public domain

image

conv-64
conv-64
maxpool

conv-128
conv-128
maxpool

conv-256
conv-256
maxpool

conv-512
conv-512
maxpool

conv-512
conv-512
maxpool

FC-4096
FC-4096

x0

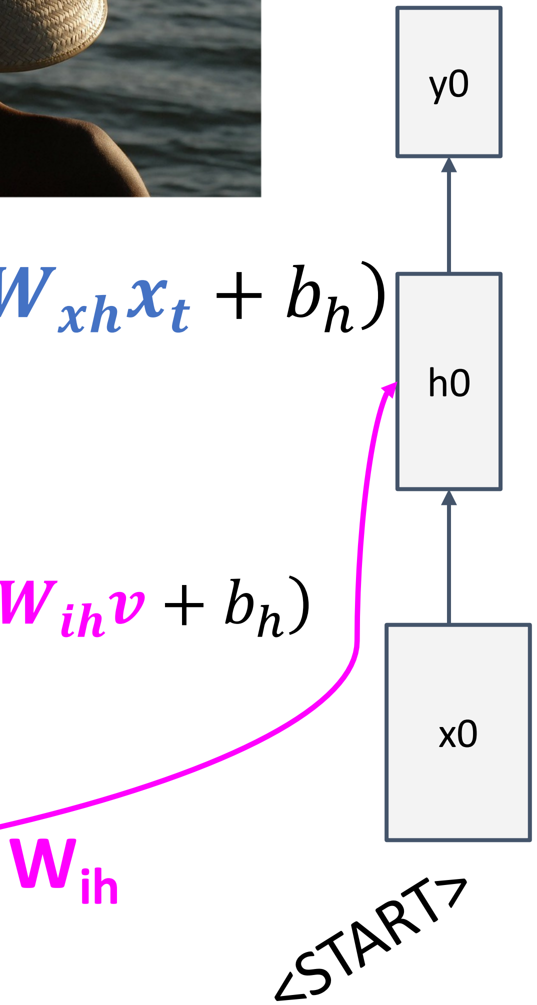
<START>

Before:

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

Now:

$$\tanh(\boldsymbol{W_{hh}h_{t-1}} + \boldsymbol{W_{xh}x_t} + \boldsymbol{W_{ih}v} + b_h)$$



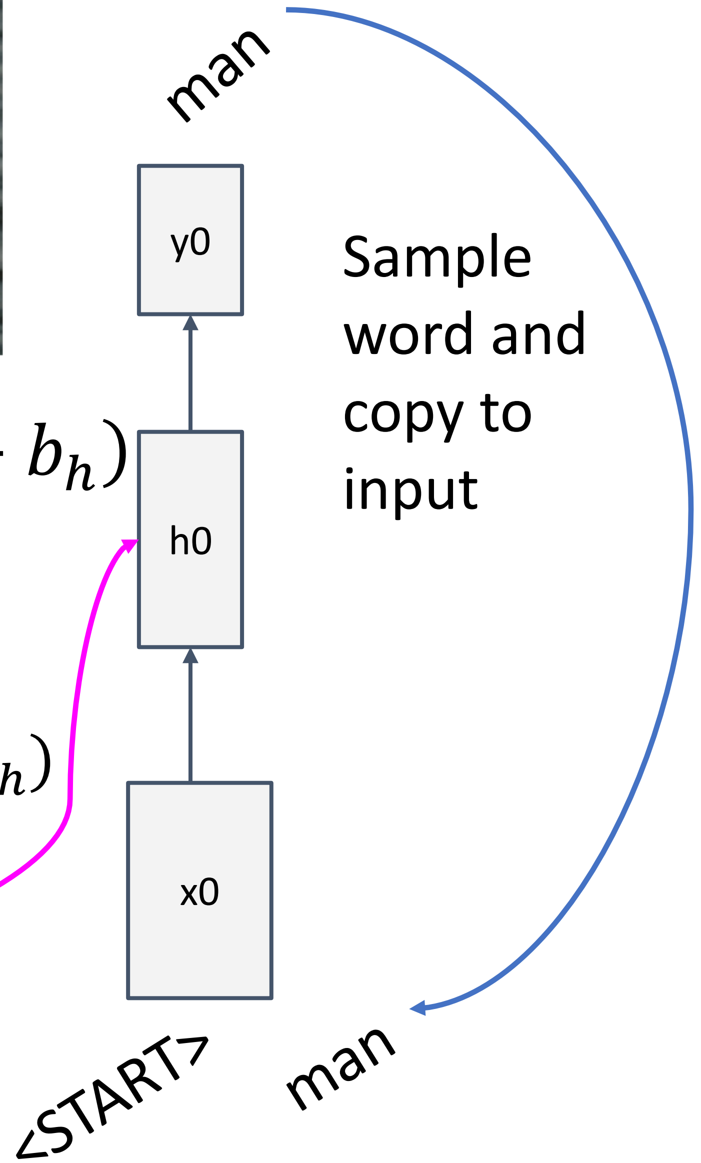
Before:

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

Now:

$$\tanh(\boldsymbol{W_{hh}h_{t-1}} + \boldsymbol{W_{xh}x_t} + \boldsymbol{W_{ih}v} + b_h)$$

W_{ih}



Sample word and copy to input

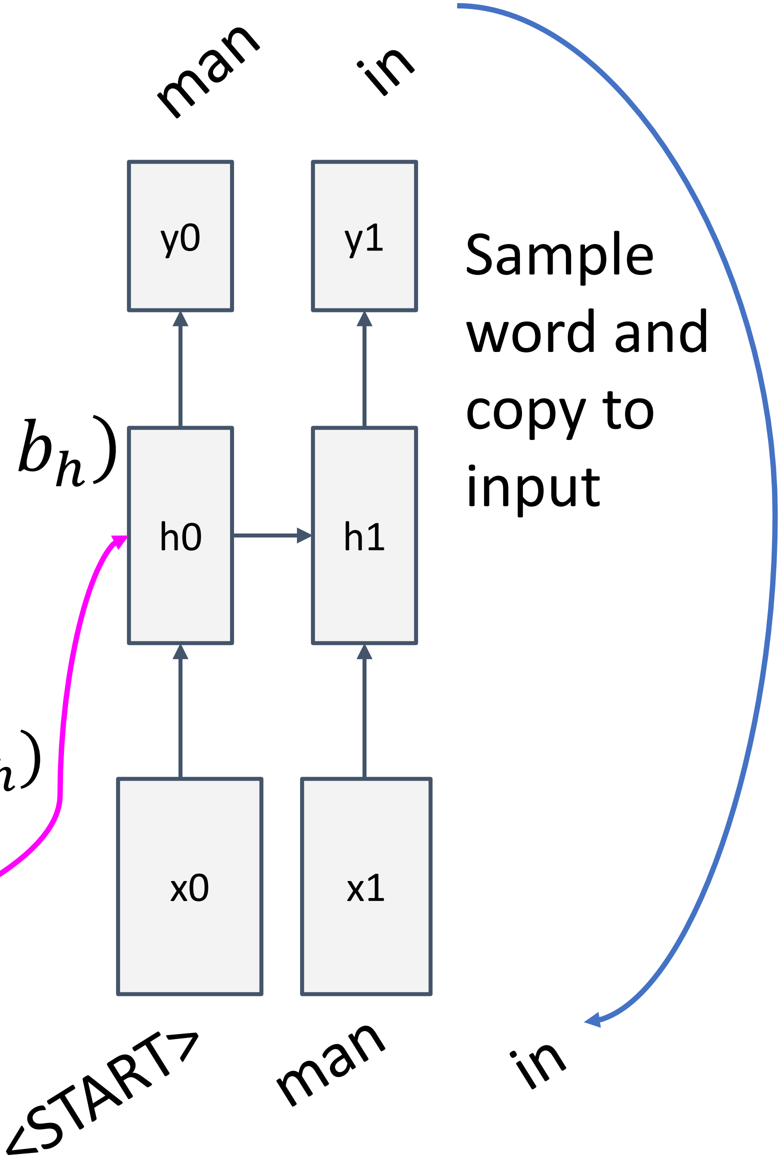
Slide from Justin Johnson

Before:

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

Now:

$$\tanh(\boldsymbol{W_{hh}h_{t-1}} + \boldsymbol{W_{xh}x_t} + \boldsymbol{W_{ih}v} + b_h)$$



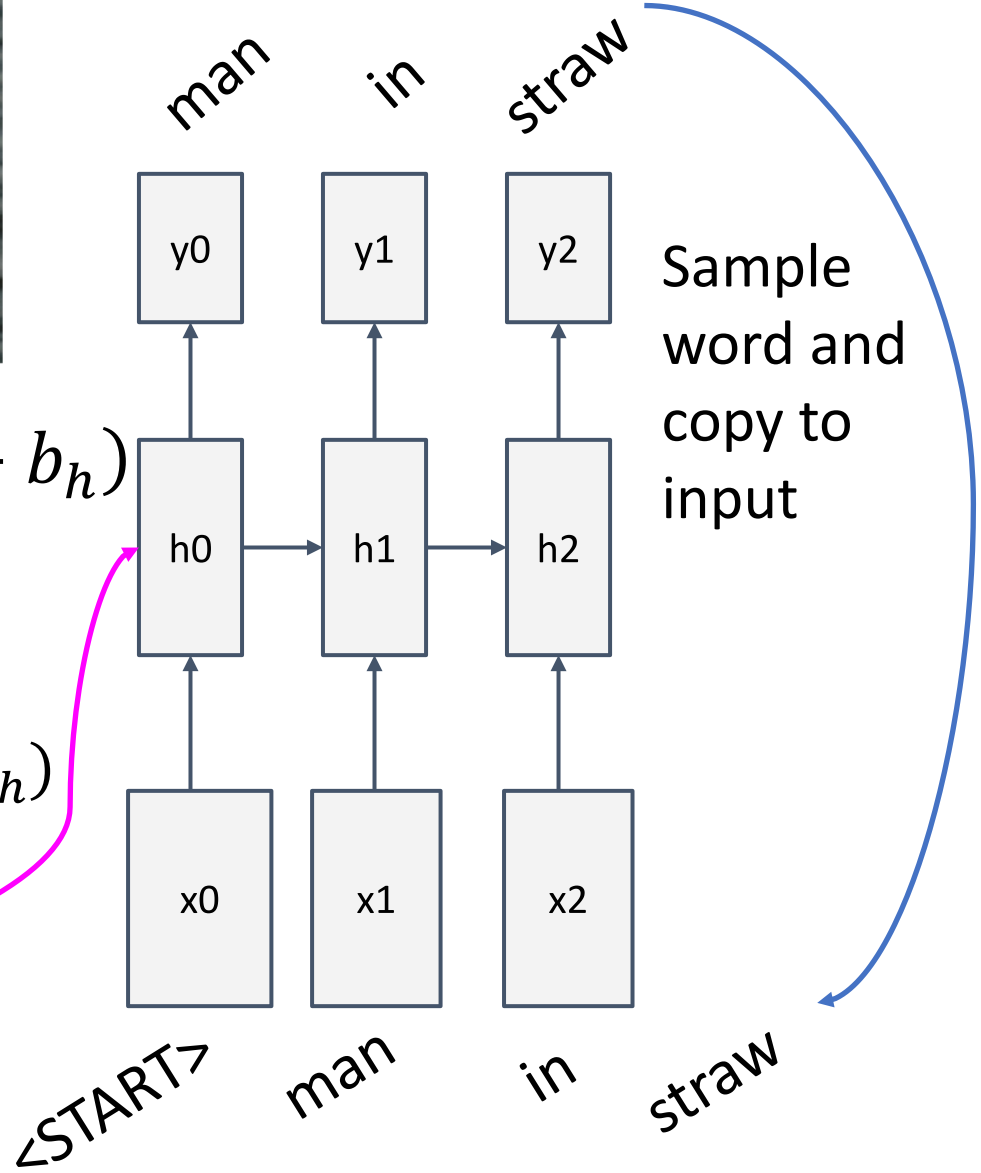
Before:

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

Now:

$$\tanh(\boldsymbol{W_{hh}} \boldsymbol{h_{t-1}} + \boldsymbol{W_{xh}} \boldsymbol{x_t} + \boldsymbol{W_{ih}} \boldsymbol{v} + b_h)$$

W_{ih}



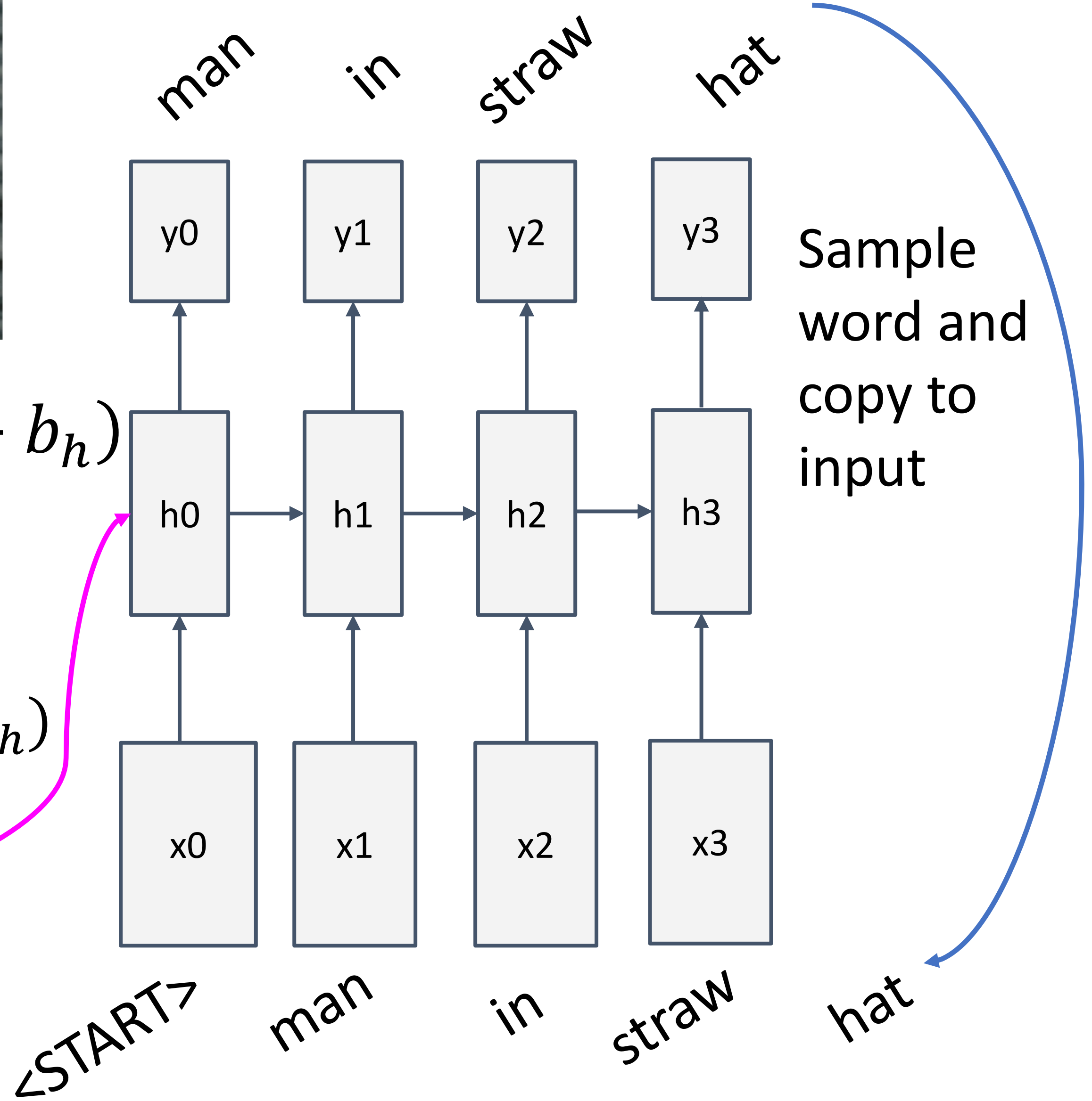
Before:

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

Now:

$$\tanh(\boldsymbol{W_{hh}h_{t-1}} + \boldsymbol{W_{xh}x_t} + \boldsymbol{W_{ih}v} + b_h)$$

W_{ih}



This image is CC0 public domain

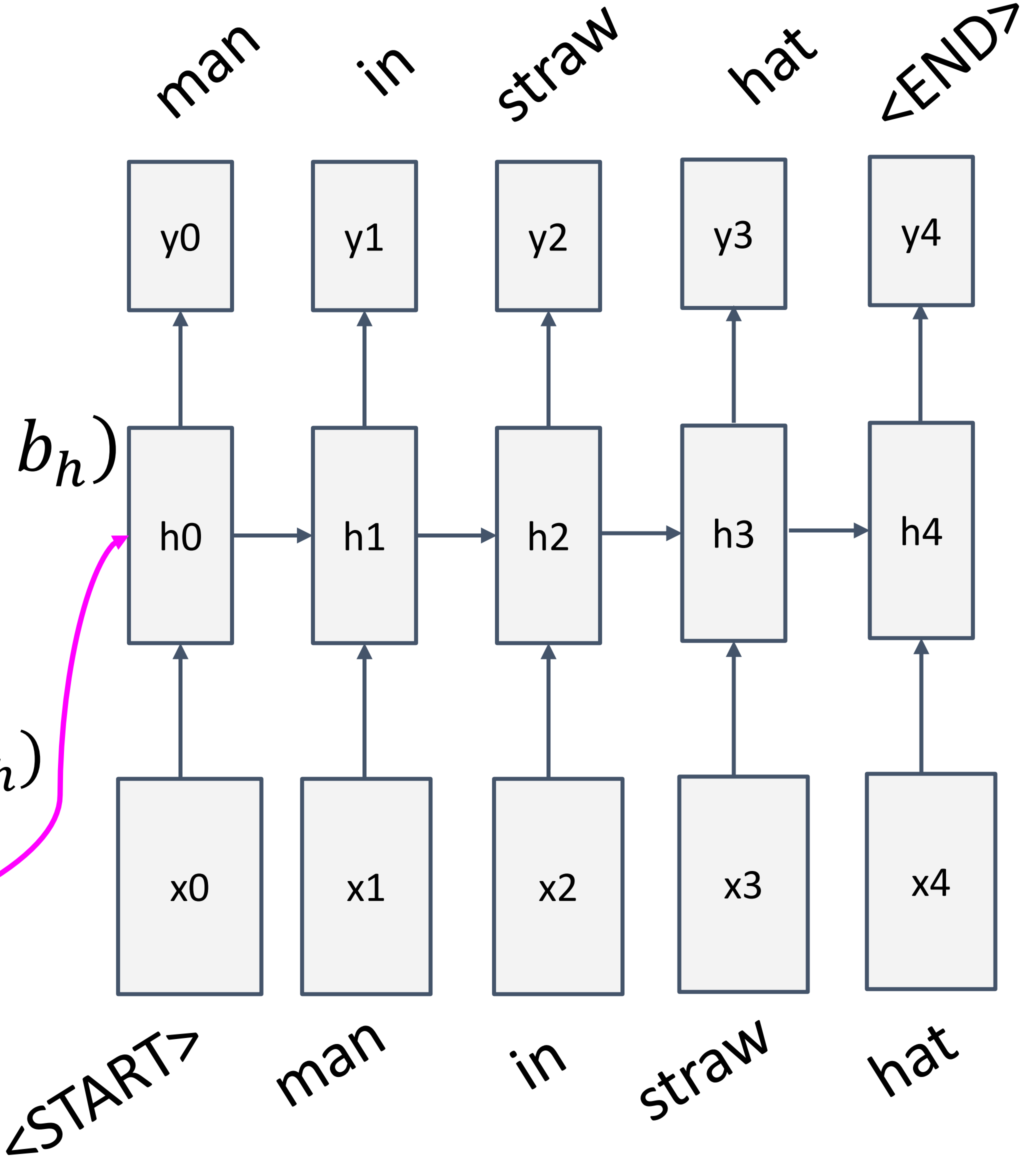
Stop after sampling <END> token

Before:

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

Now:

$$\tanh(\boldsymbol{W_{hh}h_{t-1}} + \boldsymbol{W_{xh}x_t} + \boldsymbol{W_{ih}v} + b_h)$$



W$_{ih}$

Image Captioning: Example Results

Captions generated using neuraltalk2
All images are CC0 Public domain: cat suitcase, cat tree, dog, bear, surfers, tennis, giraffe, motorcycle

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

Slide from Justin Johnson

Captions generated using neuraltalk2
All images are CC0 Public domain: fur coat, handstand, spider web, baseball

Image Captioning: Failure Cases

A woman is holding a cat in her hand

A person holding a computer mouse on a desk

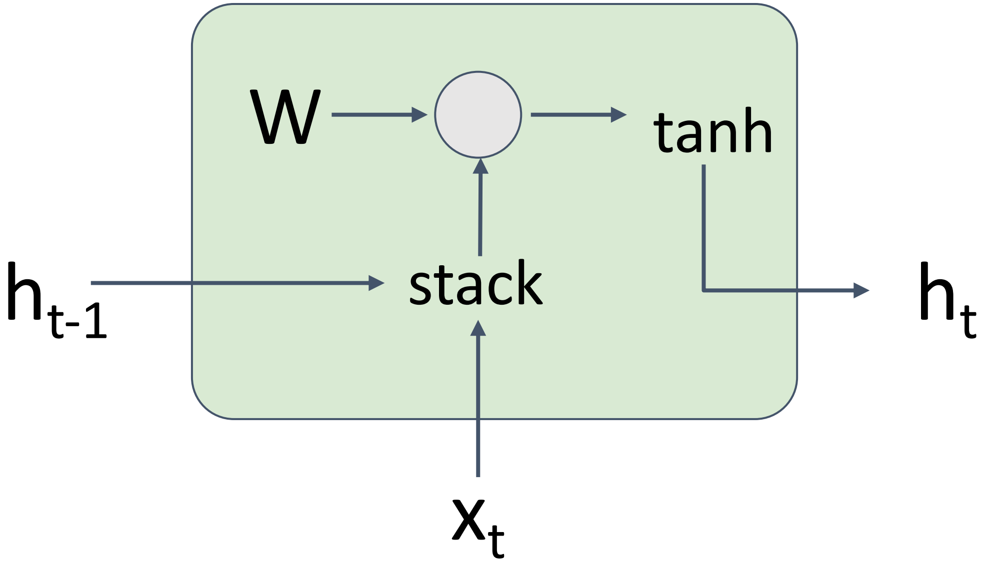
A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

Slide from Justin Johnson

Vanilla RNN Gradient Flow



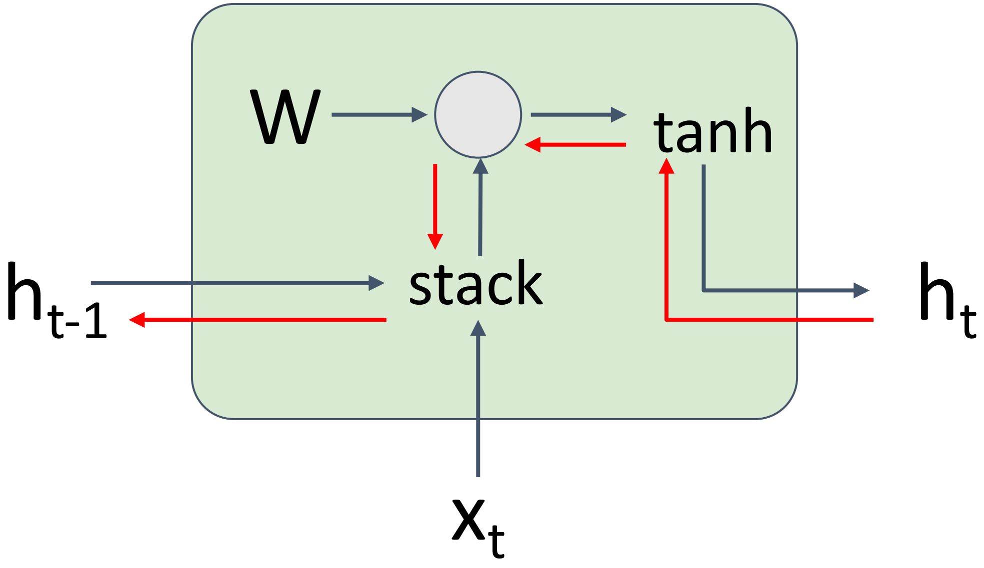
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h\right)$$
$$= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h\right)$$

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Slide from Justin Johnson

Vanilla RNN Gradient Flow

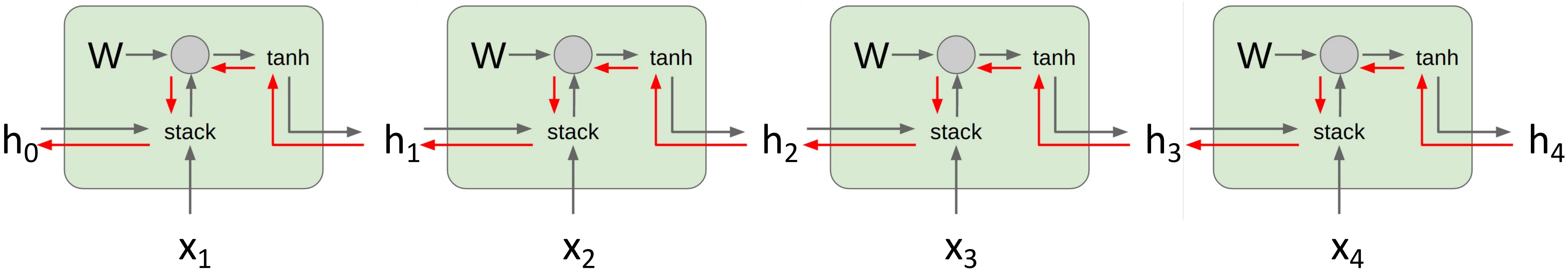
Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



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

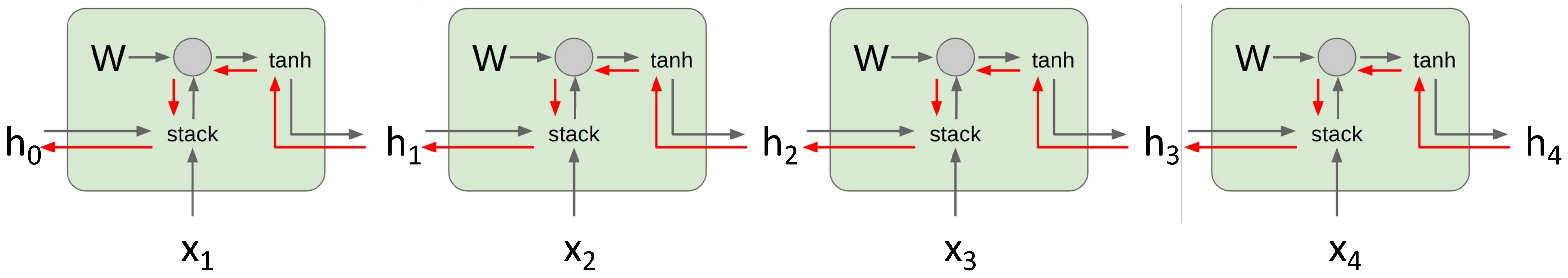
Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

Vanilla RNN Gradient Flow



Computing gradient of
h_0 involves many
factors of W
(and repeated tanh)

Vanilla RNN Gradient Flow

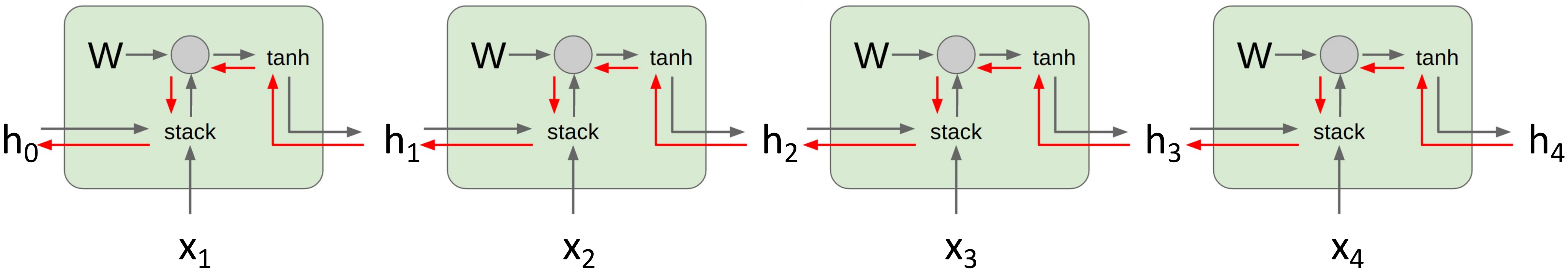


Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

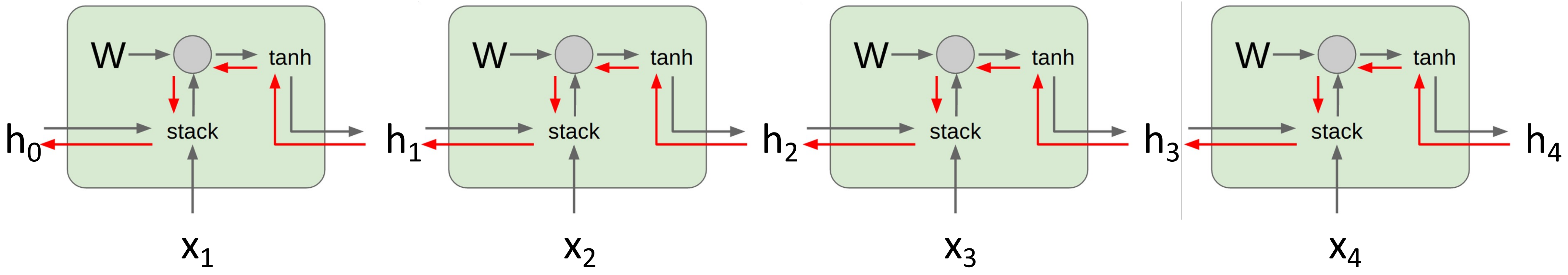
Largest singular value > 1:
Exploding gradients

→ **Gradient clipping**: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

→ **Change RNN architecture!**

Slide from Justin Johnson

Long Short Term Memory (LSTM)

Vanilla RNN

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

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

Long Short Term Memory (LSTM)

Vanilla RNN

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

LSTM

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h\right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

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

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

Slide from Justin Johnson

Long Short Term Memory (LSTM)

Vanilla RNN

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

LSTM

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h\right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

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

Two vectors at each timestep:

Cell state: $c_t \in \mathbb{R}^H$

Hidden state: $h_t \in \mathbb{R}^H$

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

Slide from Justin Johnson

Long Short Term Memory (LSTM)

Vanilla RNN

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

LSTM

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h\right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

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

Compute four "gates" per timestep:

Input gate: $i_t \in \mathbb{R}^H$

Forget gate: $f_t \in \mathbb{R}^H$

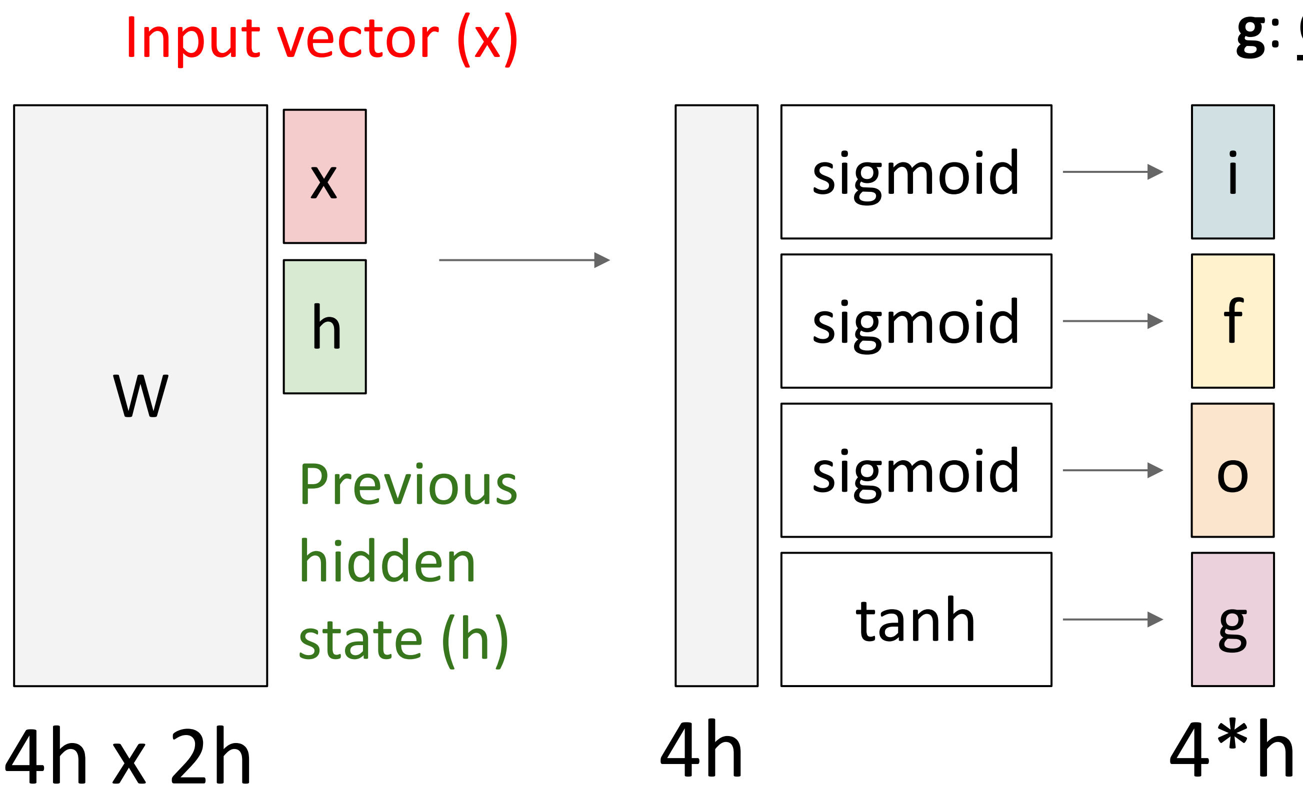
Output gate: $o_t \in \mathbb{R}^H$

"Gate?" gate: $g_t \in \mathbb{R}^H$

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997

Long Short Term Memory (LSTM)

i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Gate gate (?), How much to write to cell

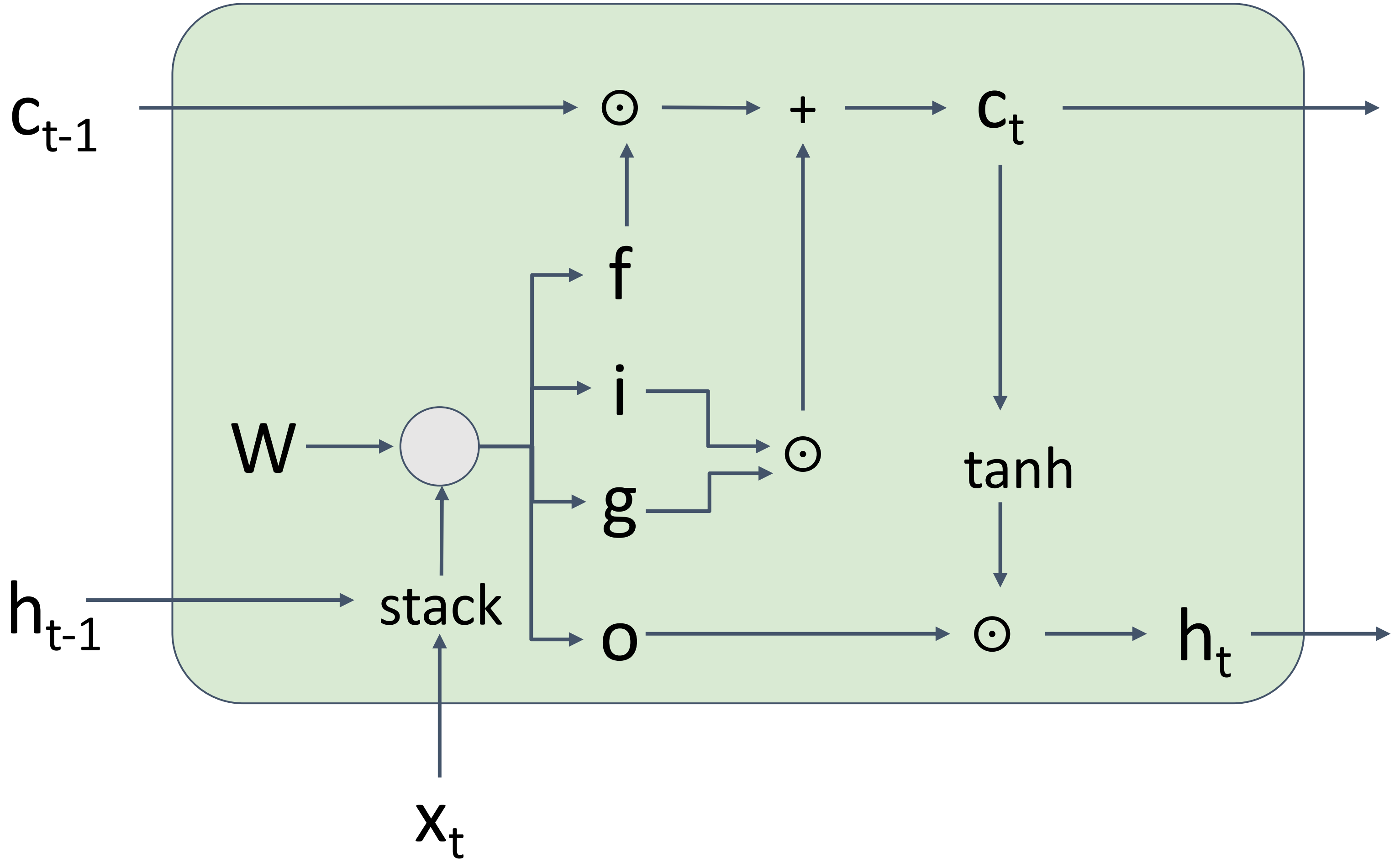


$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

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

Long Short Term Memory (LSTM)

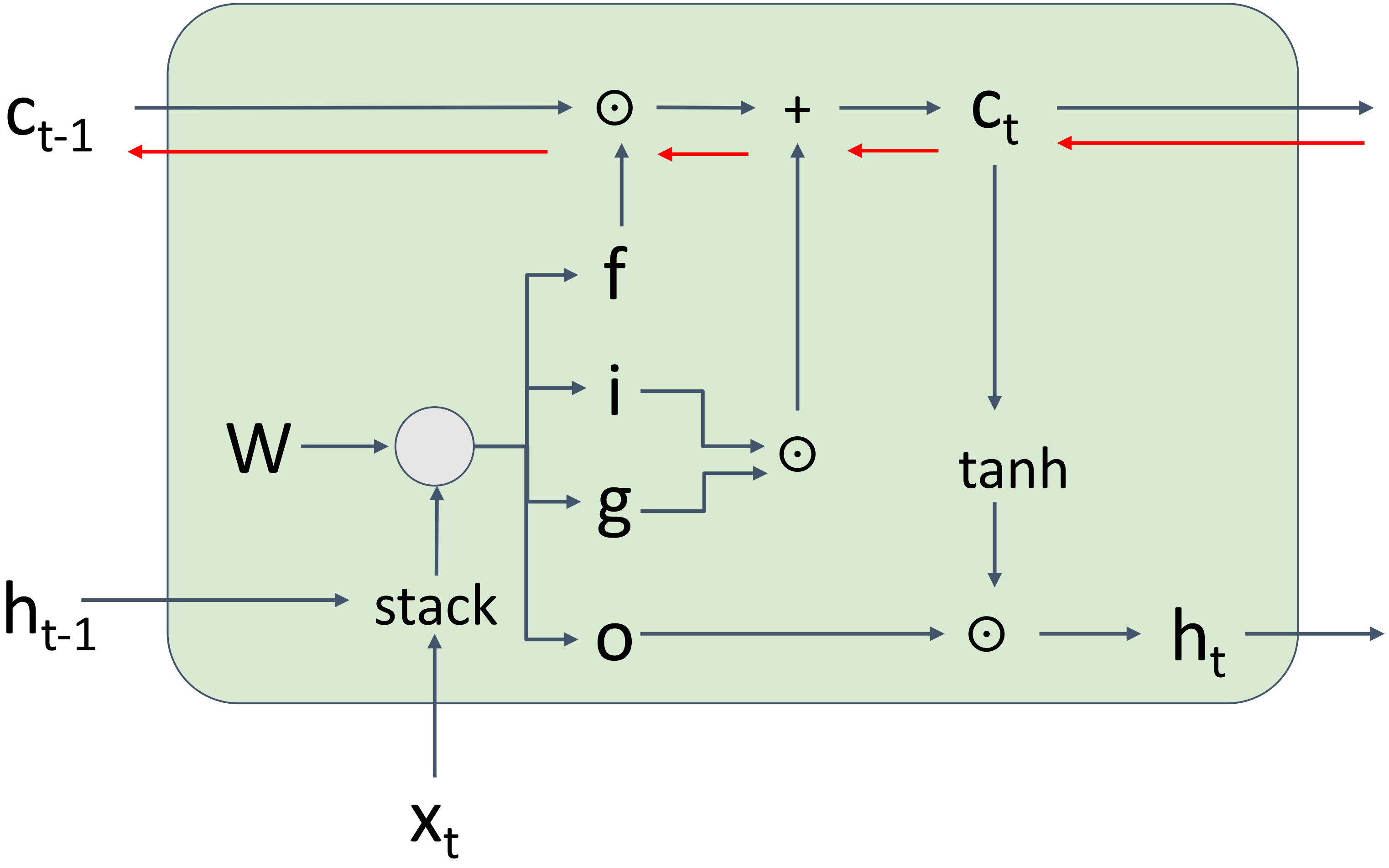


$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

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

Long Short Term Memory (LSTM): Gradient Flow



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

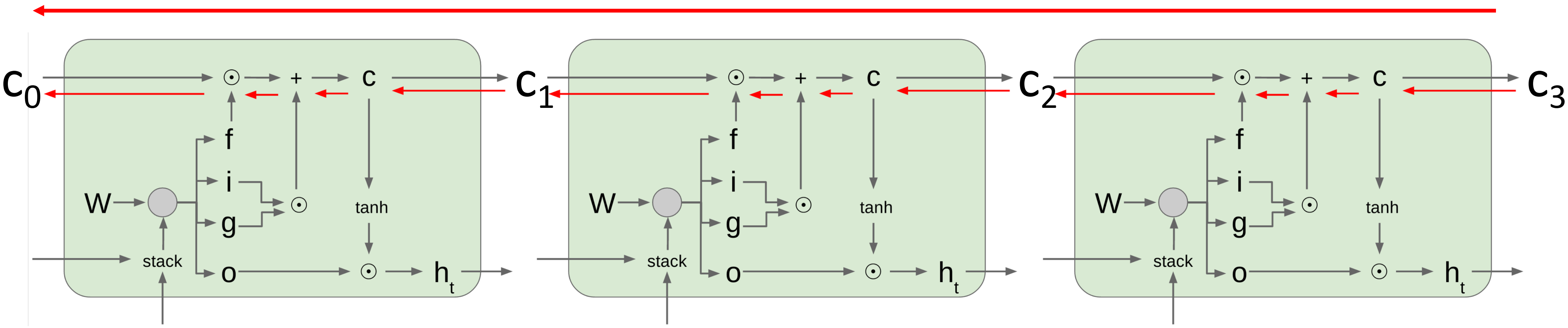
$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

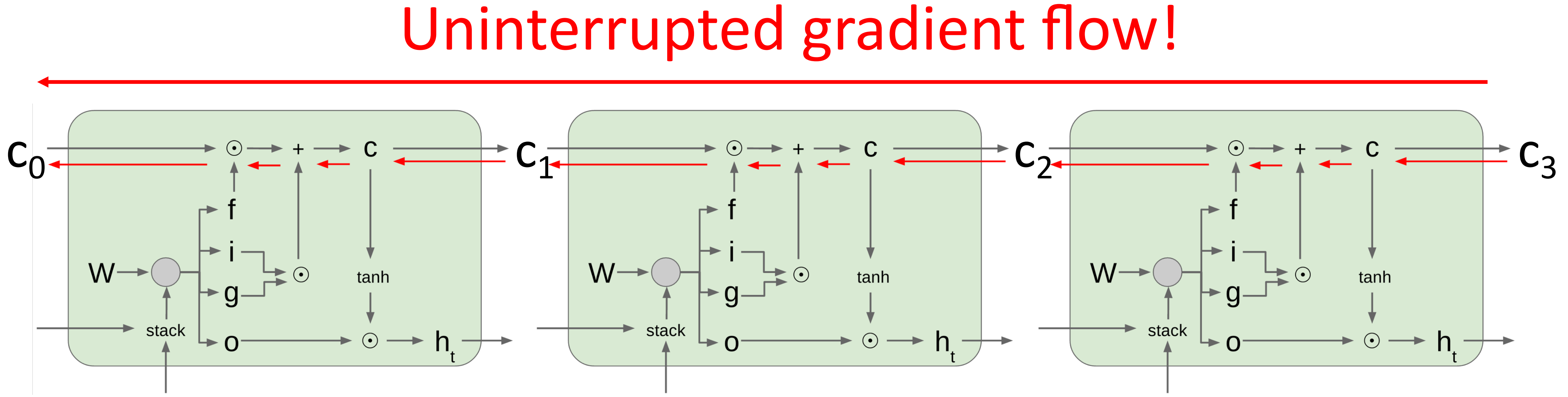
$$h_t = o_t \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow

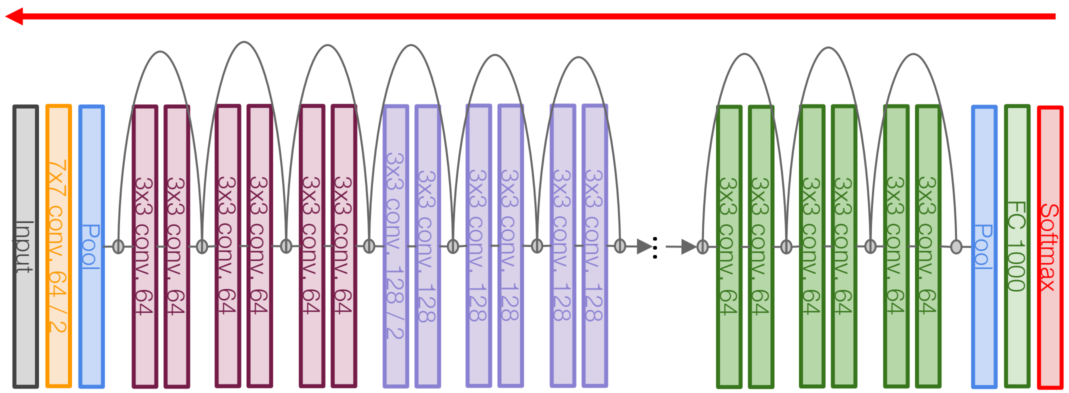
Uninterrupted gradient flow!



Long Short Term Memory (LSTM): Gradient Flow

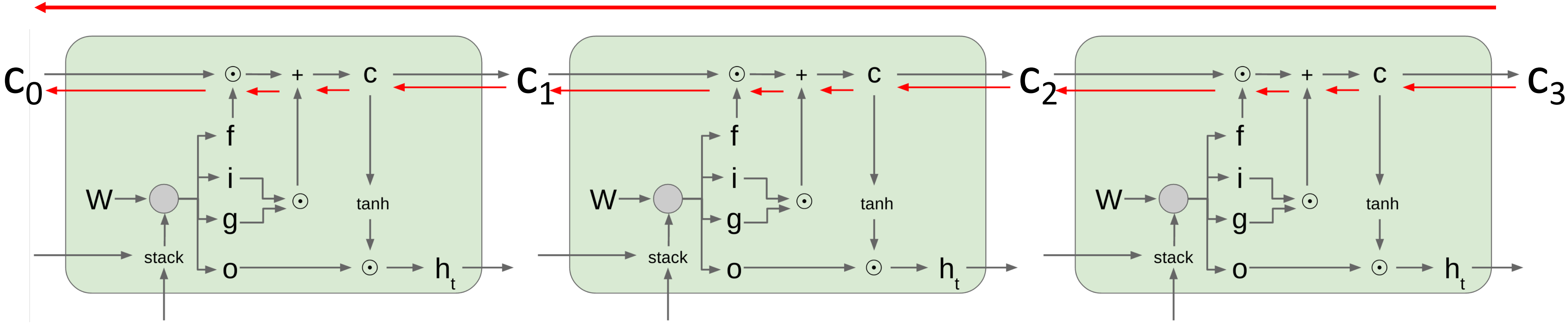


Similar to ResNet!

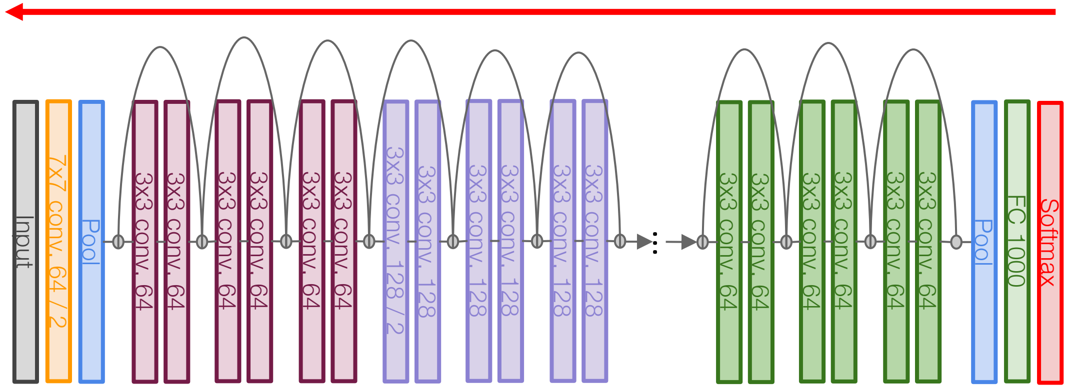


Long Short Term Memory (LSTM): Gradient Flow

Uninterrupted gradient flow!



Similar to ResNet!



In between: **Highway Networks**

$$g_t = F(x, W_t)$$
$$y_t = g_t \odot H(x, W_h) + (1 - g_t) \odot x_t$$

Srivastava et al, "Highway Networks", ICML DL Workshop 2015

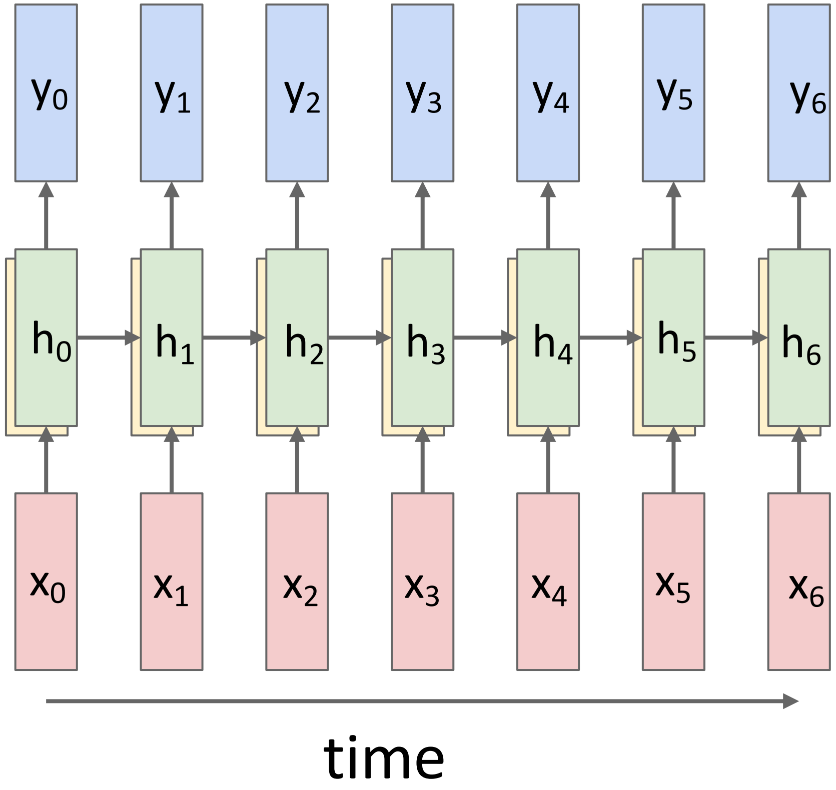
Slide from Justin Johnson

Single-Layer RNNs

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

LSTM:

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h\right)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$



Mutilayer RNNs

$$h_t^{\ell} = \tanh\left(W \begin{pmatrix} h_{t-1}^{\ell} \\ h_t^{\ell-1} \end{pmatrix} + b_h^{\ell}\right)$$

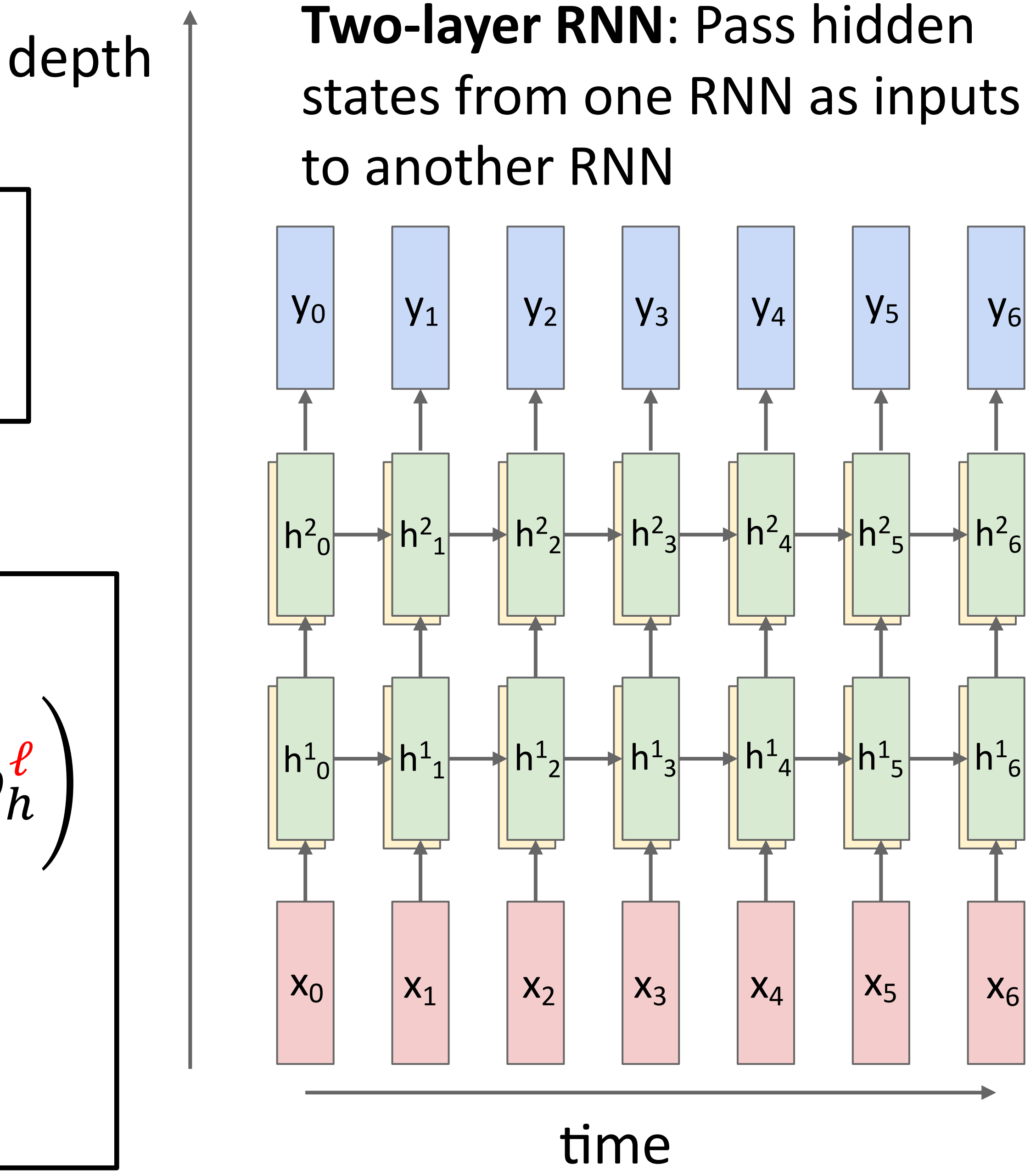
LSTM:

$$\begin{pmatrix} i_t^{\ell} \\ f_t^{\ell} \\ o_t^{\ell} \\ g_t^{\ell} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix}\left(W \begin{pmatrix} h_{t-1}^{\ell} \\ h_t^{\ell-1} \end{pmatrix} + b_h^{\ell}\right)$$

$$c_t^{\ell} = f_t^{\ell} \odot c_{t-1}^{\ell} + i_t^{\ell} \odot g_t^{\ell}$$

$$h_t^{\ell} = o_t^{\ell} \odot \tanh\left(c_t^{\ell}\right)$$

Two-layer RNN: Pass hidden states from one RNN as inputs to another RNN



Slide from Justin Johnson

Mutilayer RNNs

$$h_t^\ell = \tanh\left(W\begin{pmatrix} h_{t-1}^\ell \\ h_t^{\ell-1}\end{pmatrix} + b_h^\ell\right)$$

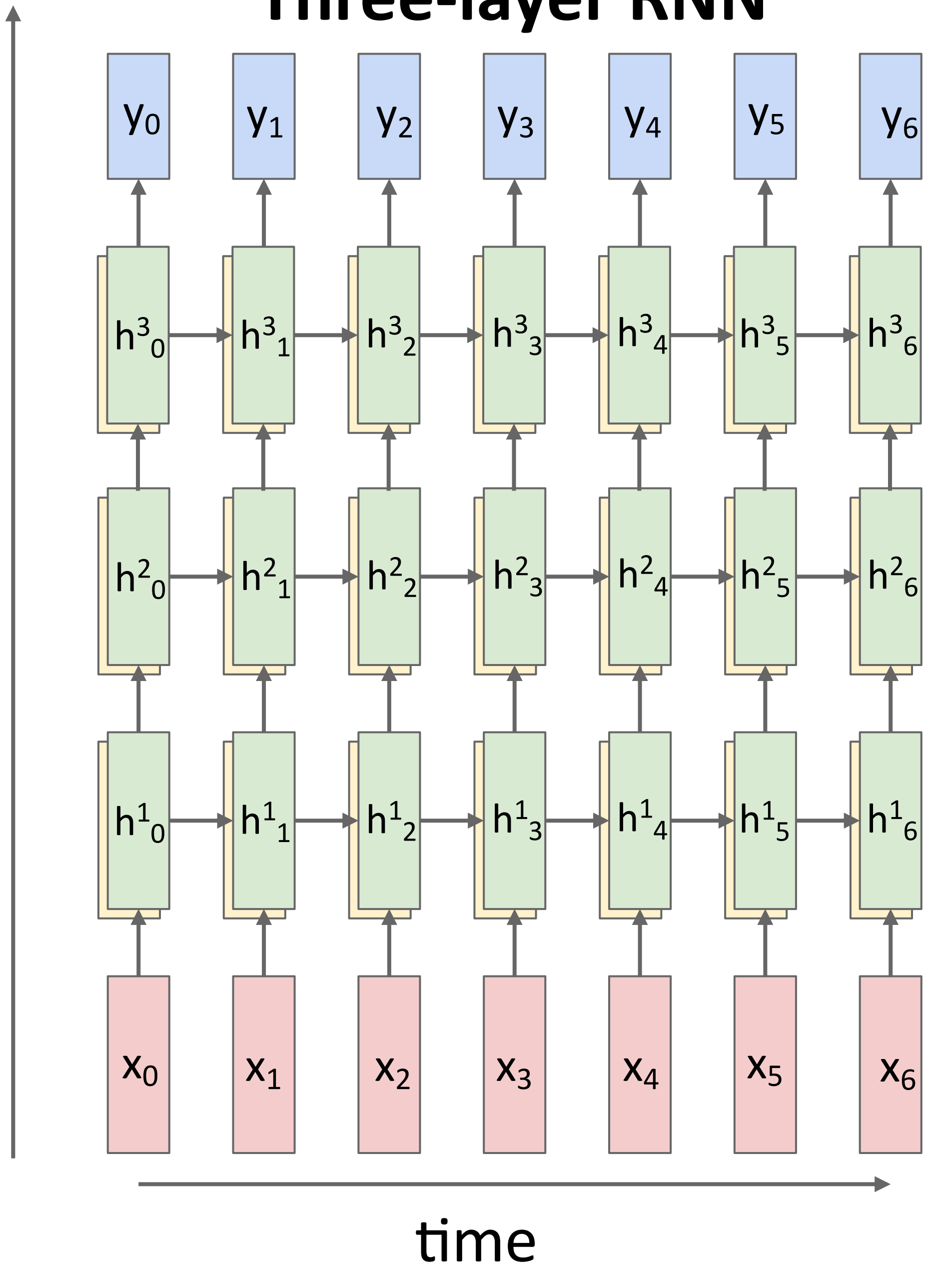
LSTM:

$$\begin{pmatrix} i_t^\ell \\ f_t^\ell \\ o_t^\ell \\ g_t^\ell \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix}\left(W\begin{pmatrix} h_{t-1}^\ell \\ h_t^{\ell-1}\end{pmatrix} + b_h^\ell\right)$$

$$c_t^\ell = f_t^\ell \odot c_{t-1}^\ell + i_t^\ell \odot g_t^\ell$$

$$h_t^\ell = o_t^\ell \odot \tanh(c_t^\ell)$$

Three-layer RNN



Slide from Justin Johnson

Other RNN Variants

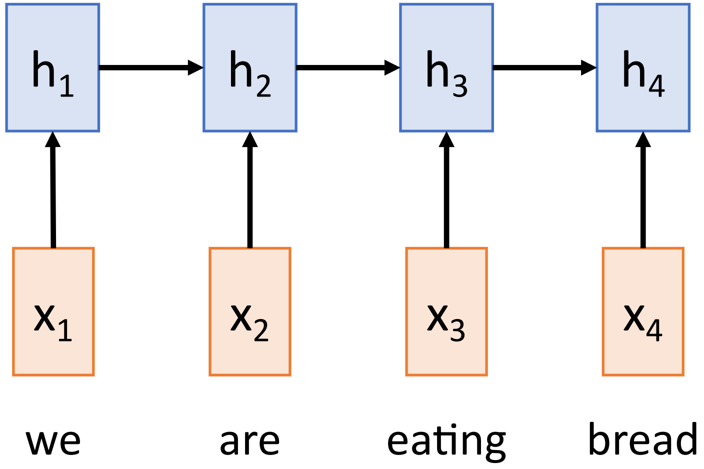
Gated Recurrent Unit (GRU)
Cho et al "Learning phrase representations using RNN encoder-decoder for statistical machine translation", 2014

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_T \odot h_{t-1}) + b_h)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

Sequence-to-Sequence with RNNs

Input: Sequence x_1, … x_T
Output: Sequence y_1, …, $y_{T'}$

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



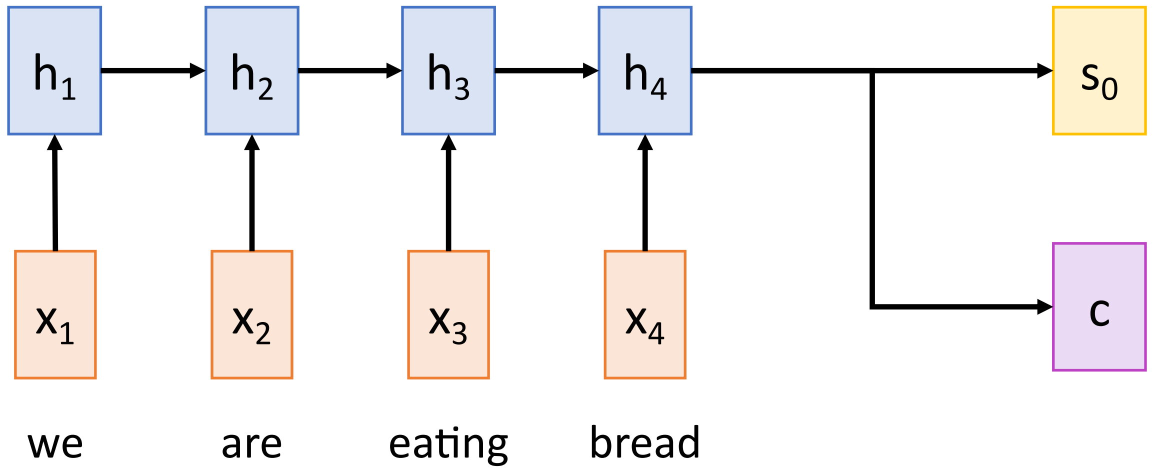
Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

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

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

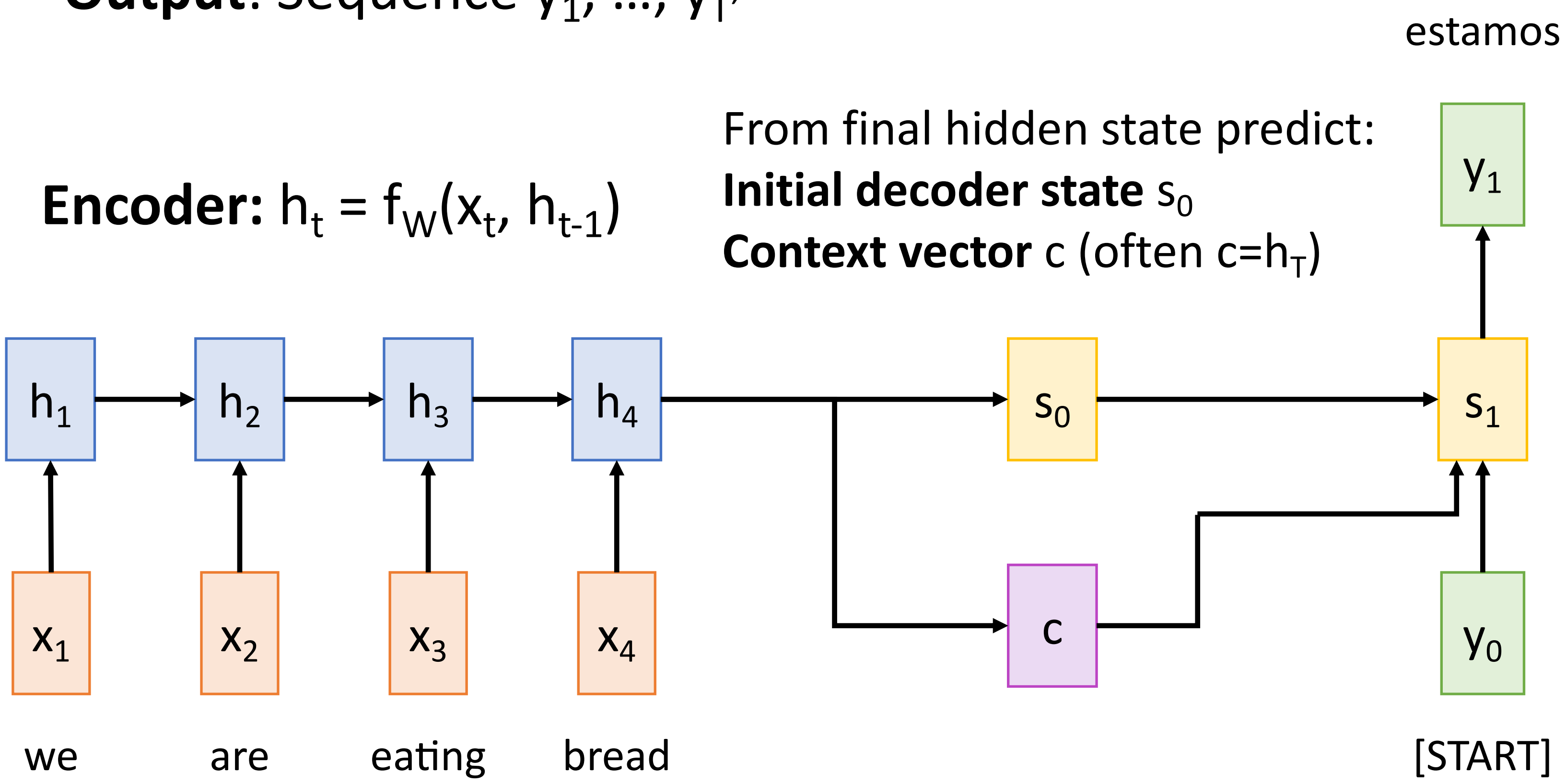
Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$

Output: Sequence $y_1, \ldots, y_{T'}$

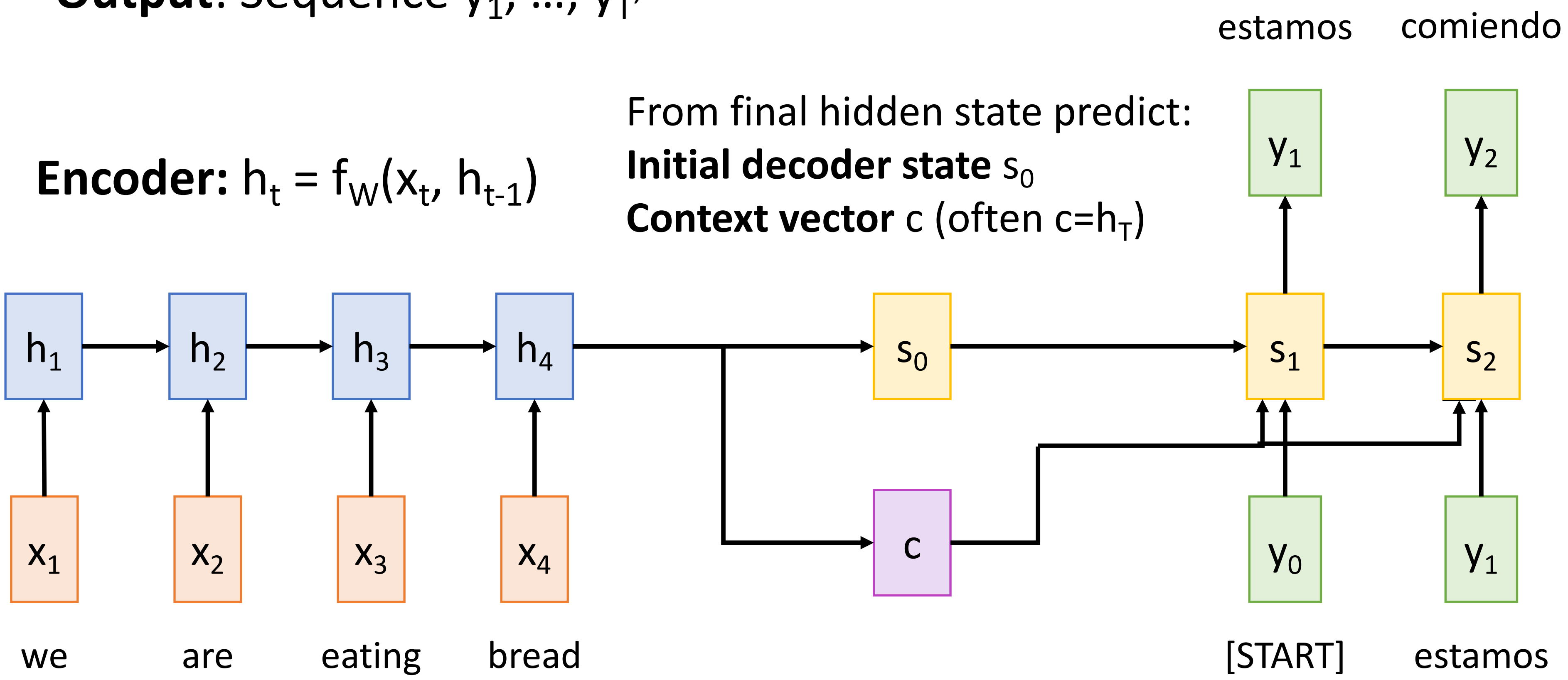
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

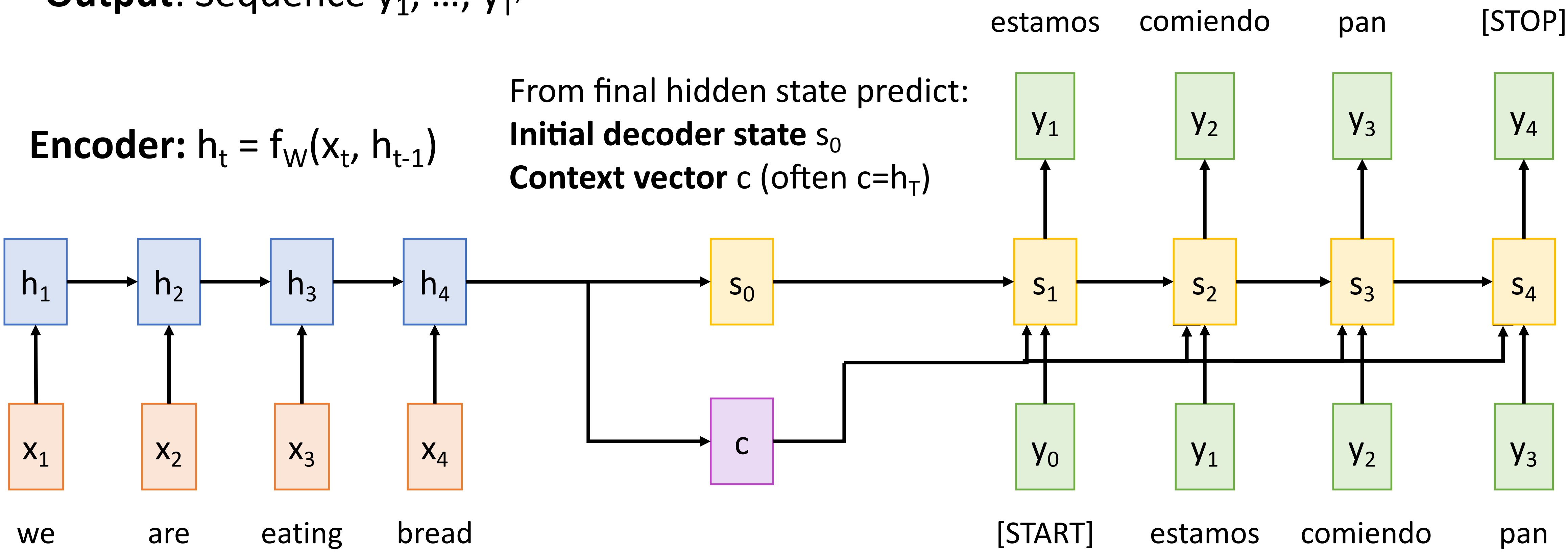
Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Slide from Justin Johnson

Sequence-to-Sequence with RNNs

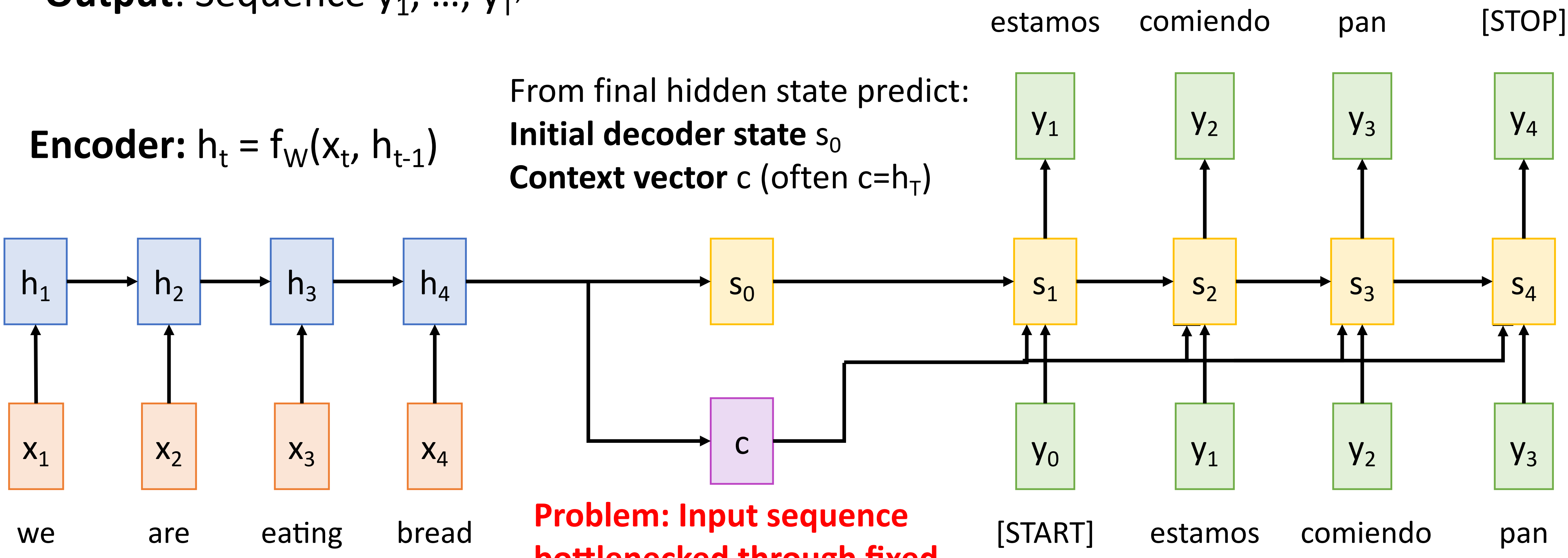
Input: Sequence $x_1, \ldots x_T$

Output: Sequence $y_1, \ldots, y_{T'}$

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



Problem: Input sequence bottlenecked through fixed-sized vector. What if T=1000?

Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence-to-Sequence with RNNs

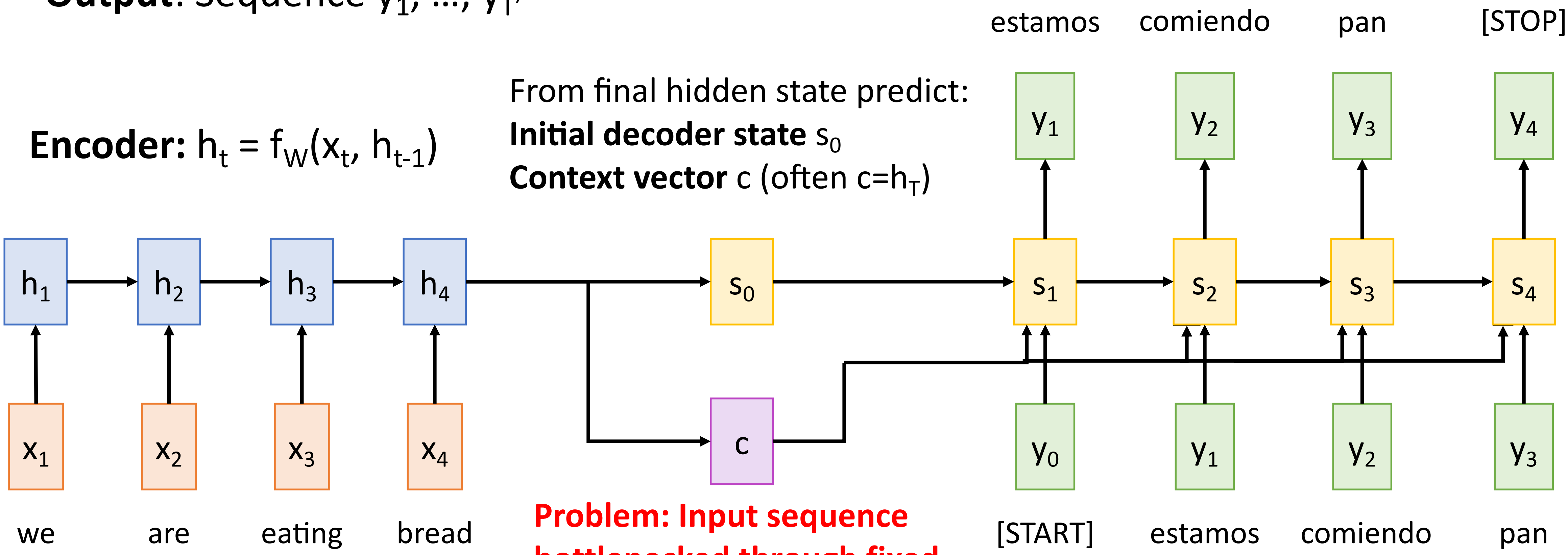
Input: Sequence $x_1, \ldots x_T$

Output: Sequence $y_1, \ldots, y_{T'}$

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



Problem: Input sequence bottlenecked through fixed-sized vector. What if T=1000?

Idea: use new context vector at each step of decoder!

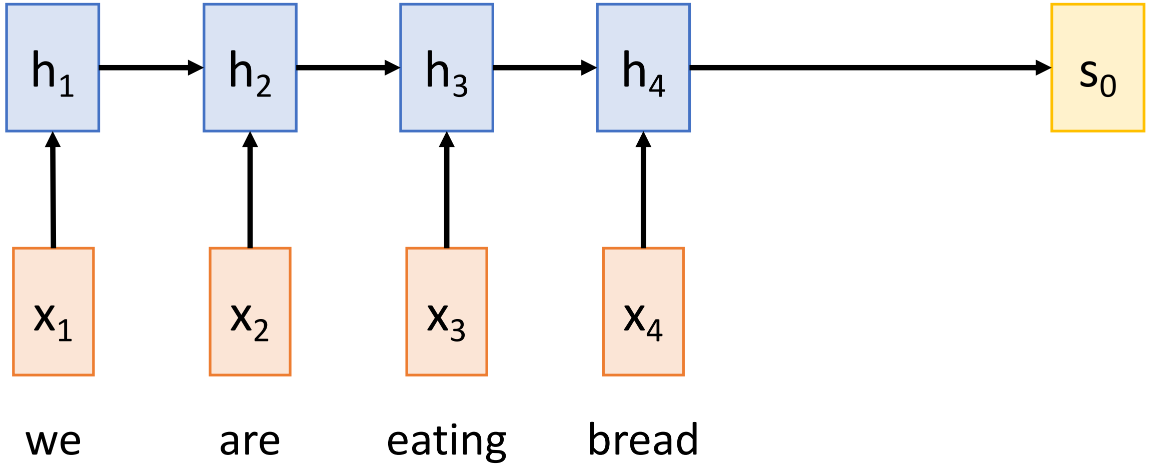
Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence-to-Sequence with RNNs **and Attention**

Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

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

From final hidden state:
Initial decoder state s_0

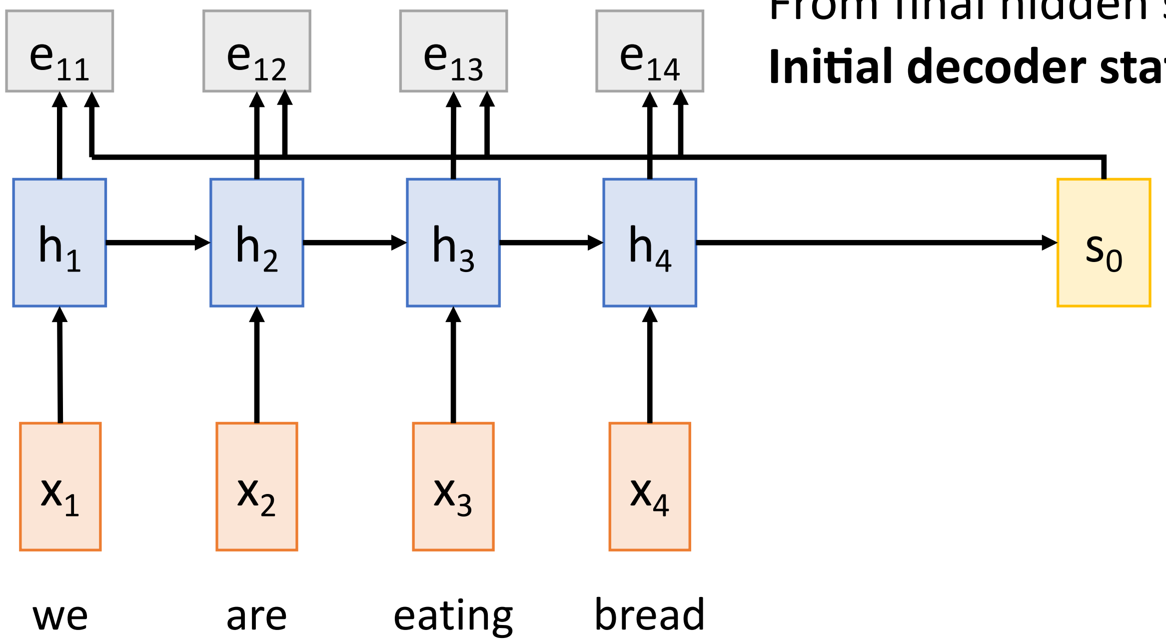


Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**

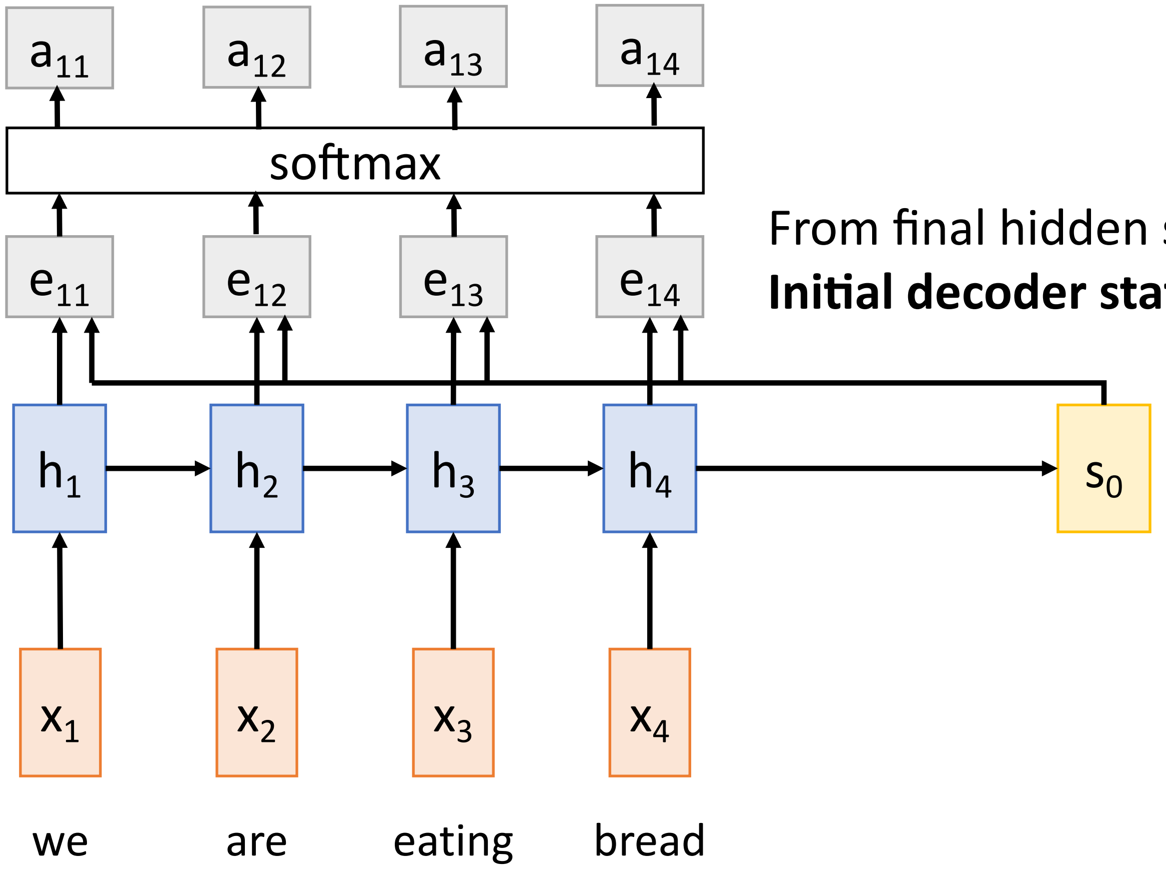
Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

From final hidden state:
Initial decoder state s_0



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**



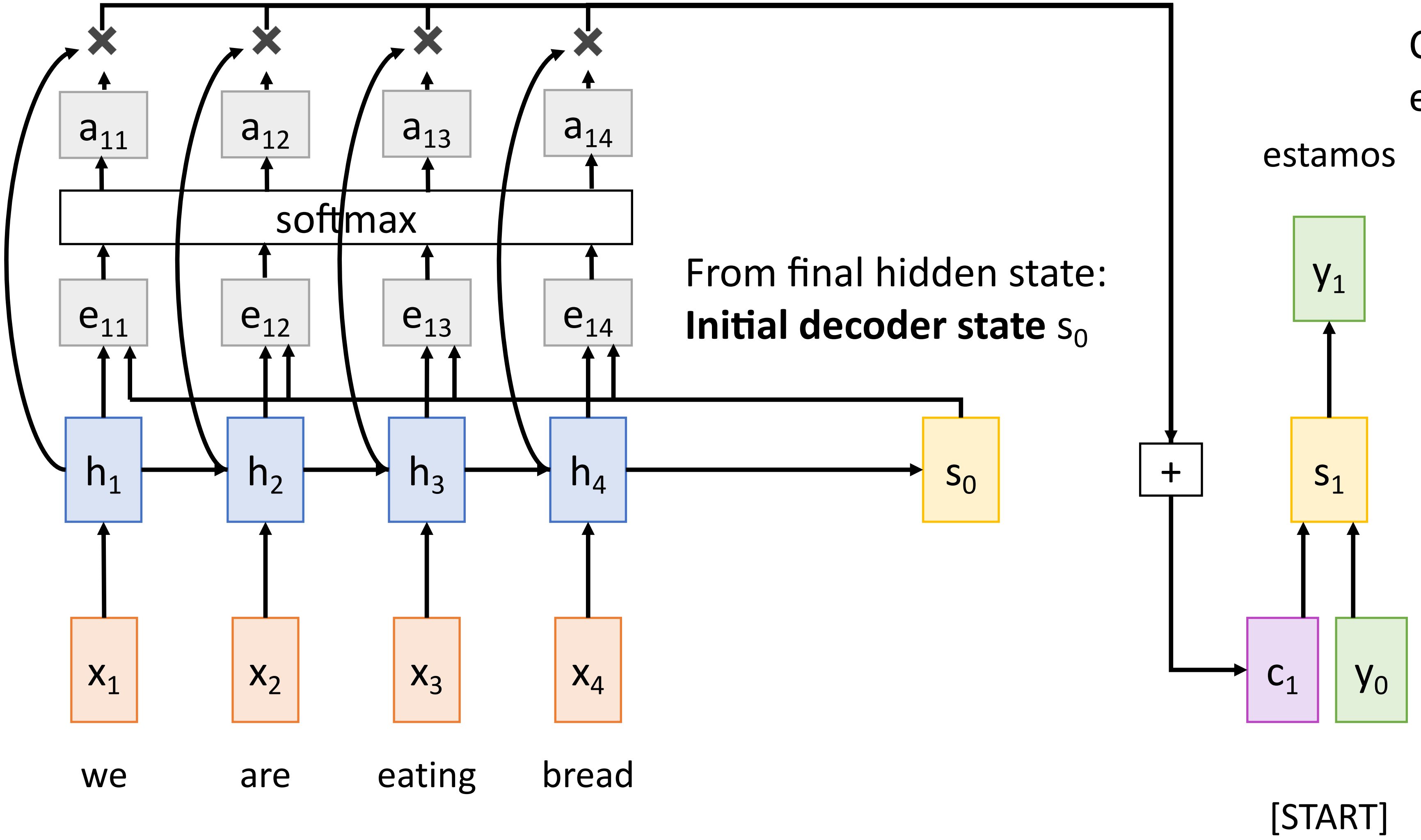
From final hidden state:
Initial decoder state s_0

Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**
$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**



Compute (scalar) **alignment scores**

$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**

$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states

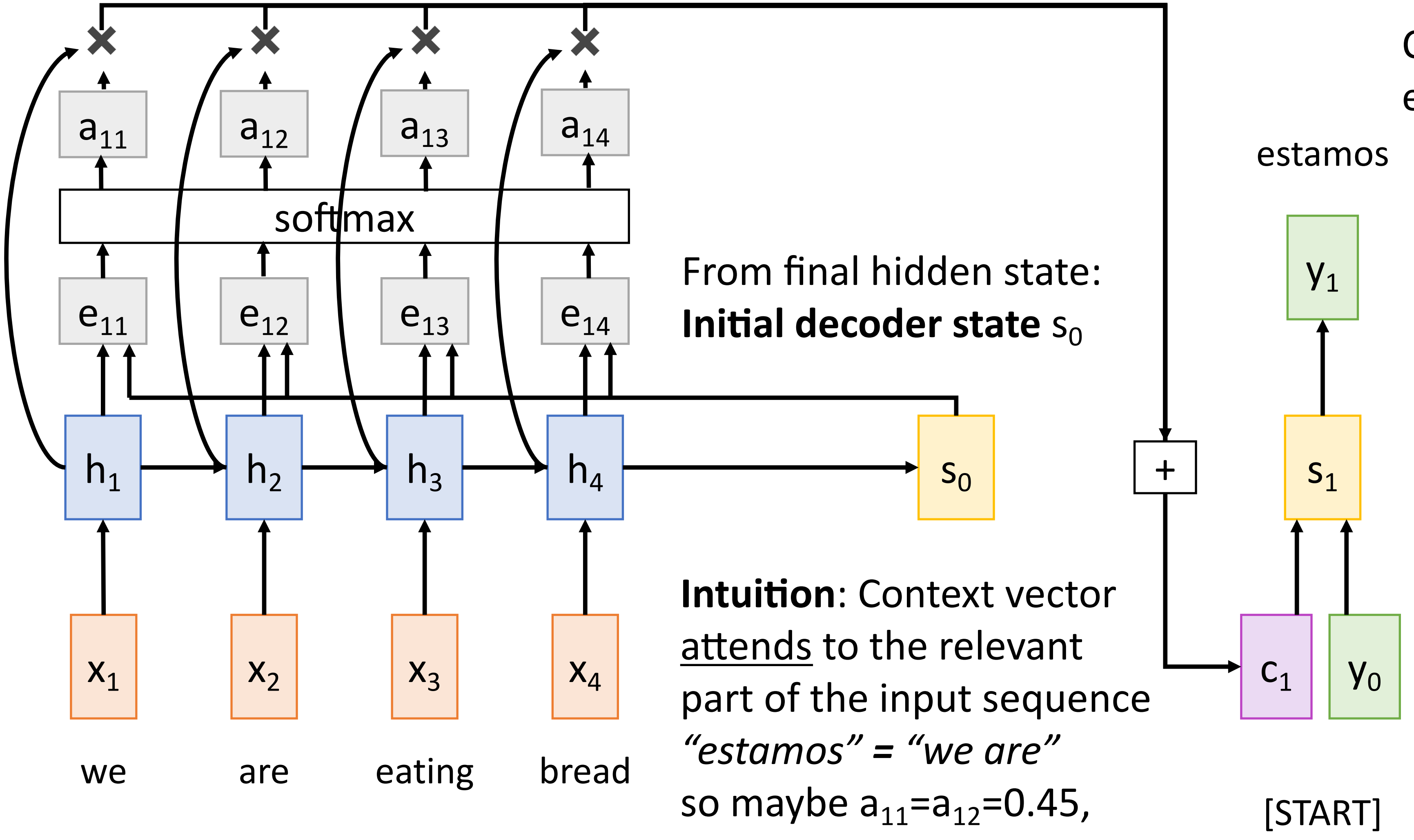
$c_t = \sum_i a_{t,i} h_i$

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

This is all differentiable! Do not supervise attention weights – backprop through everything

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**



Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**
$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states
$c_t = \sum_i a_{t,i} h_i$

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

From final hidden state: **Initial decoder state** s_0

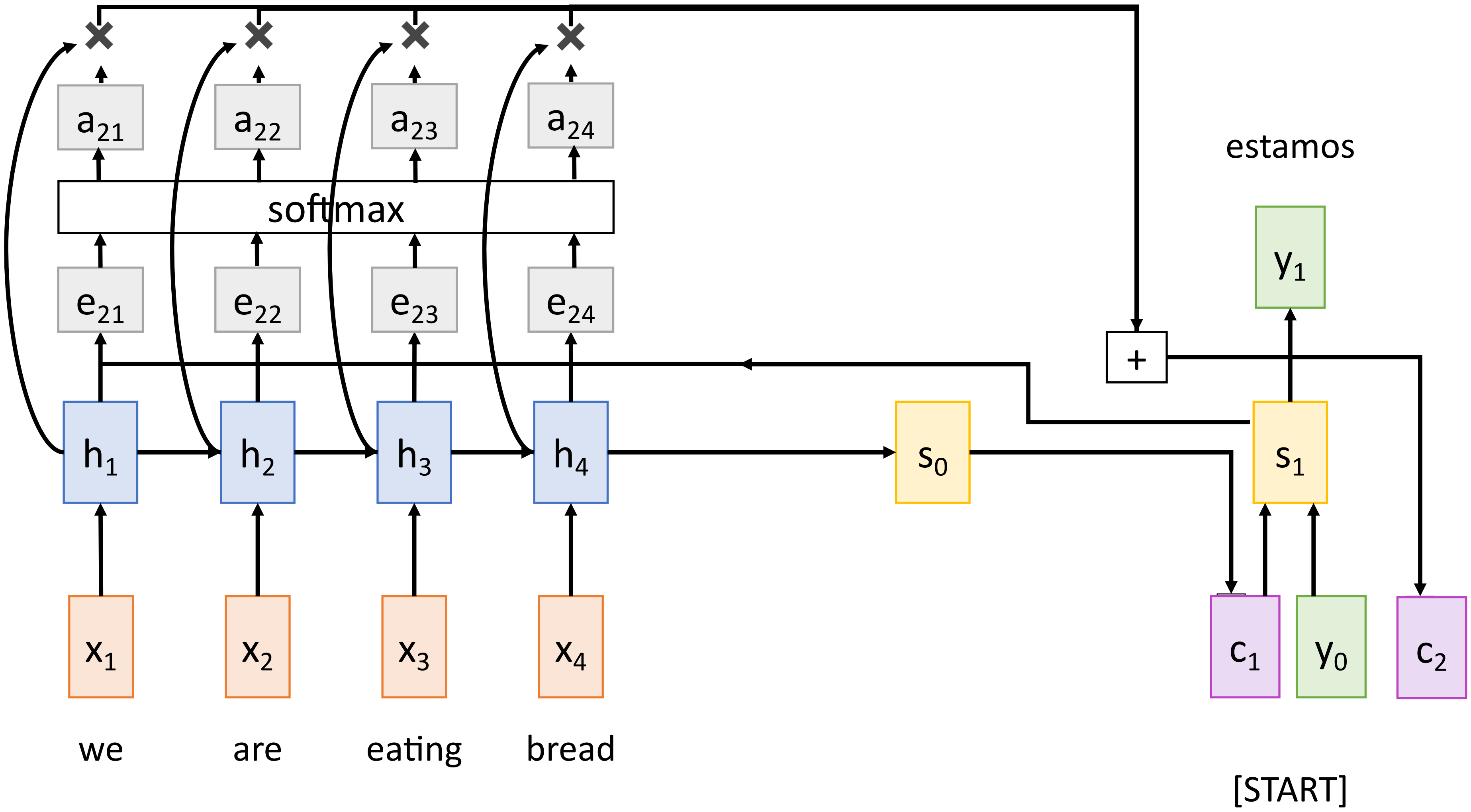
Intuition: Context vector <u>attends</u> to the relevant part of the input sequence
"estamos" = "we are"
so maybe $a_{11}=a_{12}=0.45$, $a_{13}=a_{14}=0.05$

This is all differentiable! Do not supervise attention weights – backprop through everything

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs

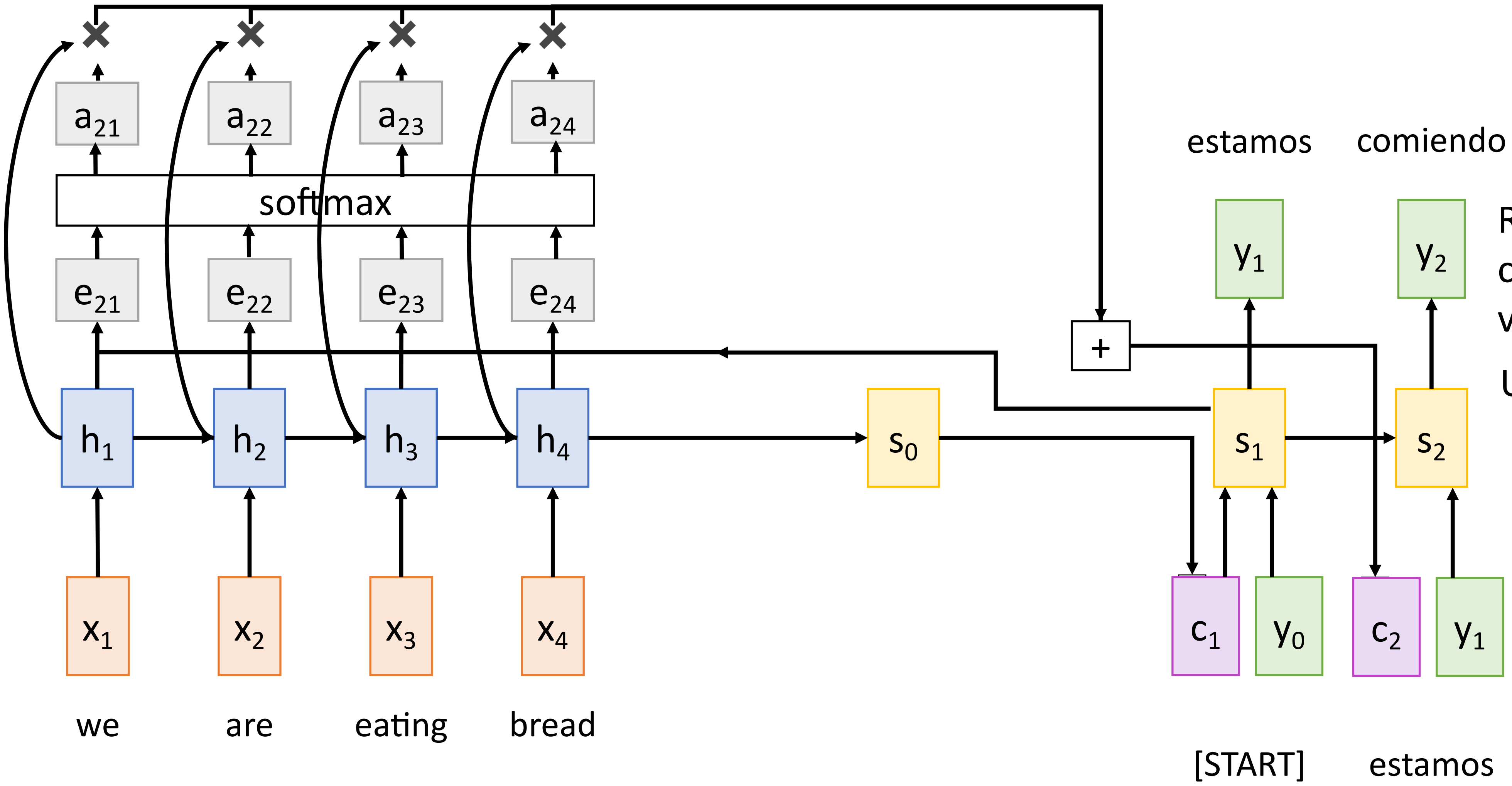
Repeat: Use s_1 to compute new context vector c_2



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Slide from Justin Johnson

Sequence-to-Sequence with RNNs **and Attention**

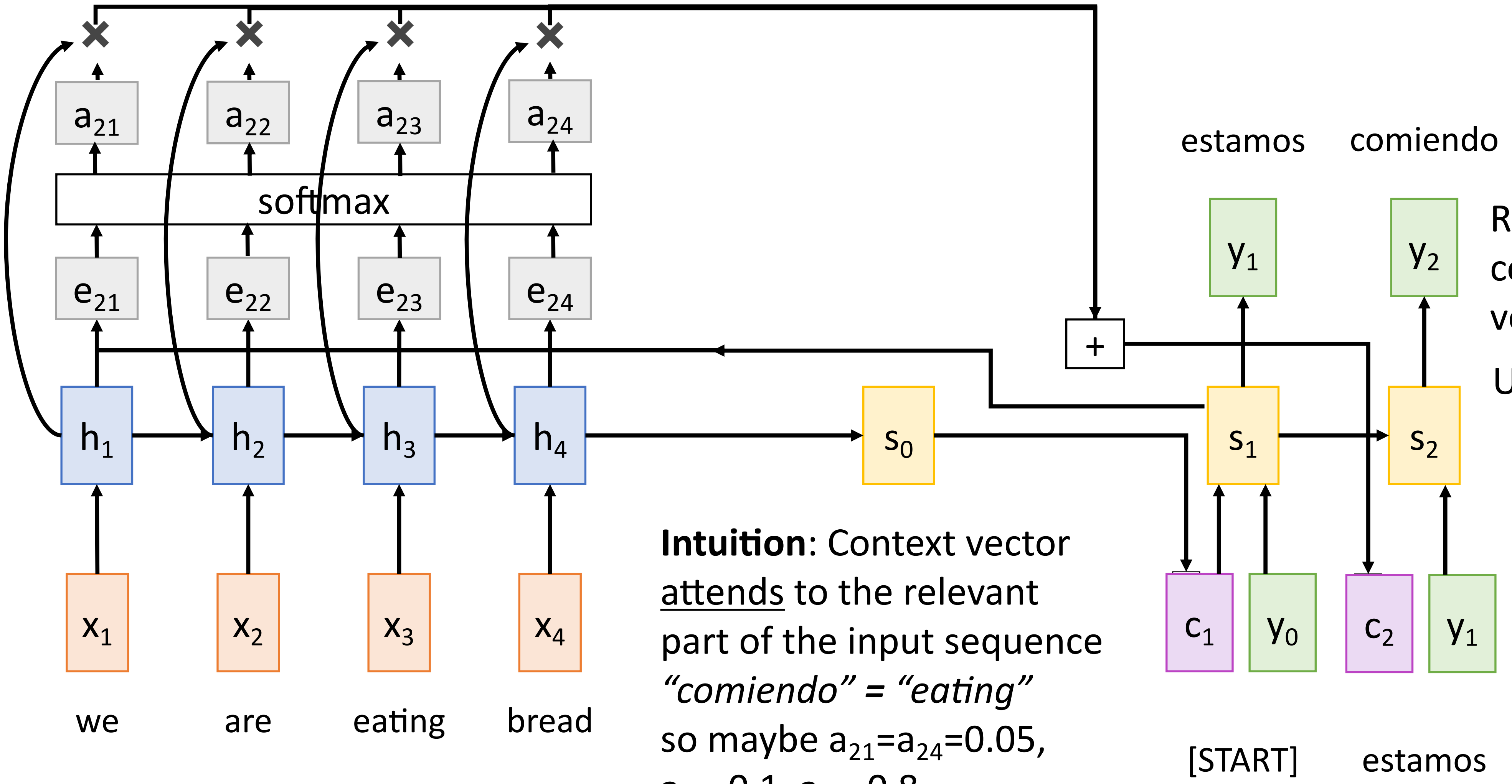


Repeat: Use s_1 to compute new context vector c_2

Use c_2 to compute s_2, y_2

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**



Repeat: Use s_1 to compute new context vector c_2

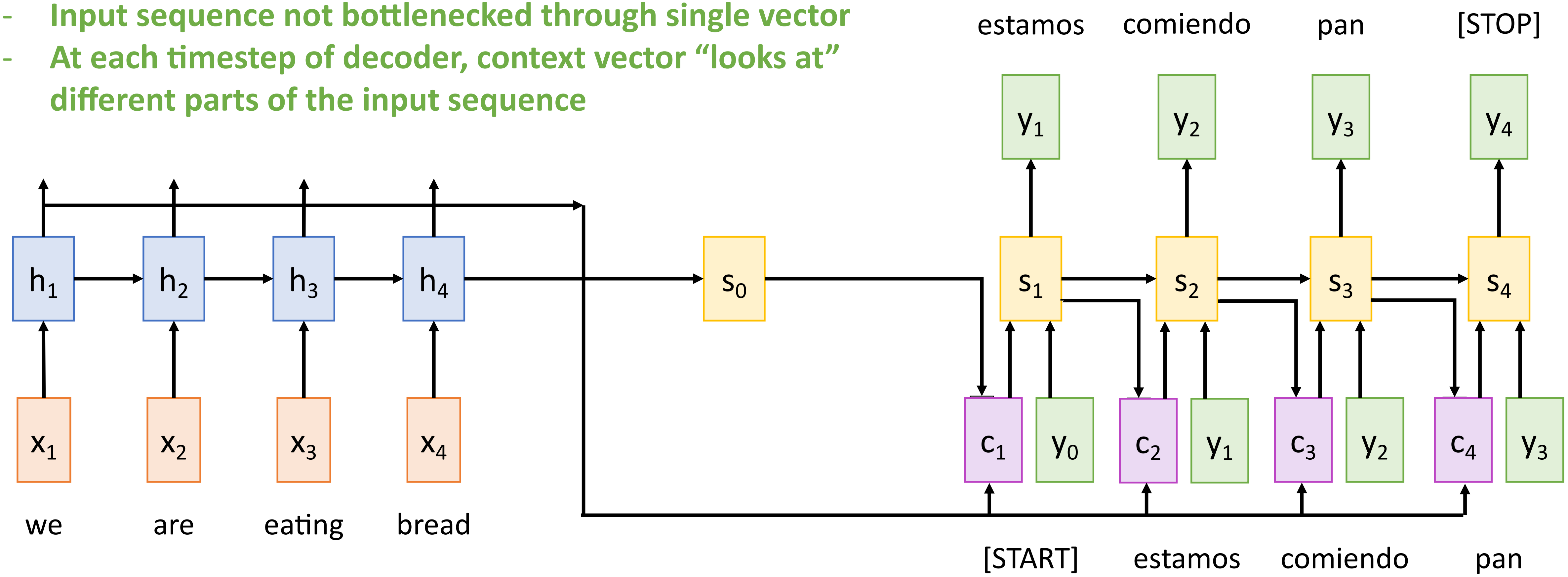
Use c_2 to compute s_2, y_2

Intuition: Context vector attends to the relevant part of the input sequence
"comiendo" = "eating"
so maybe a_{21}=a_{24}=0.05, a_{22}=0.1, a_{23}=0.8

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**

Use a different context vector in each timestep of decoder
- **Input sequence not bottlenecked through single vector**
- **At each timestep of decoder, context vector "looks at" different parts of the input sequence**



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

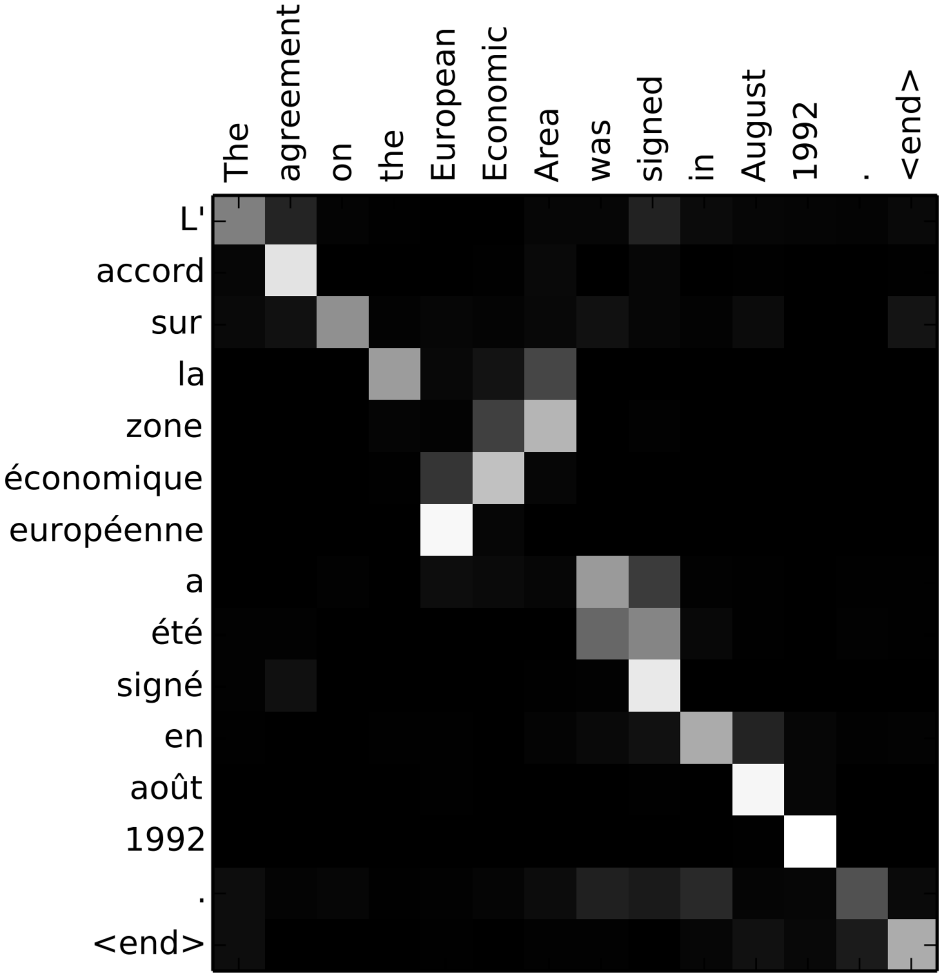
Sequence-to-Sequence with RNNs **and Attention**

Example: English to French translation

Input: “The agreement on the European Economic Area was signed in August 1992.”

Output: “L’accord sur la zone économique européenne a été signé en août 1992.”

Visualize attention weights $a_{t,i}$



Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

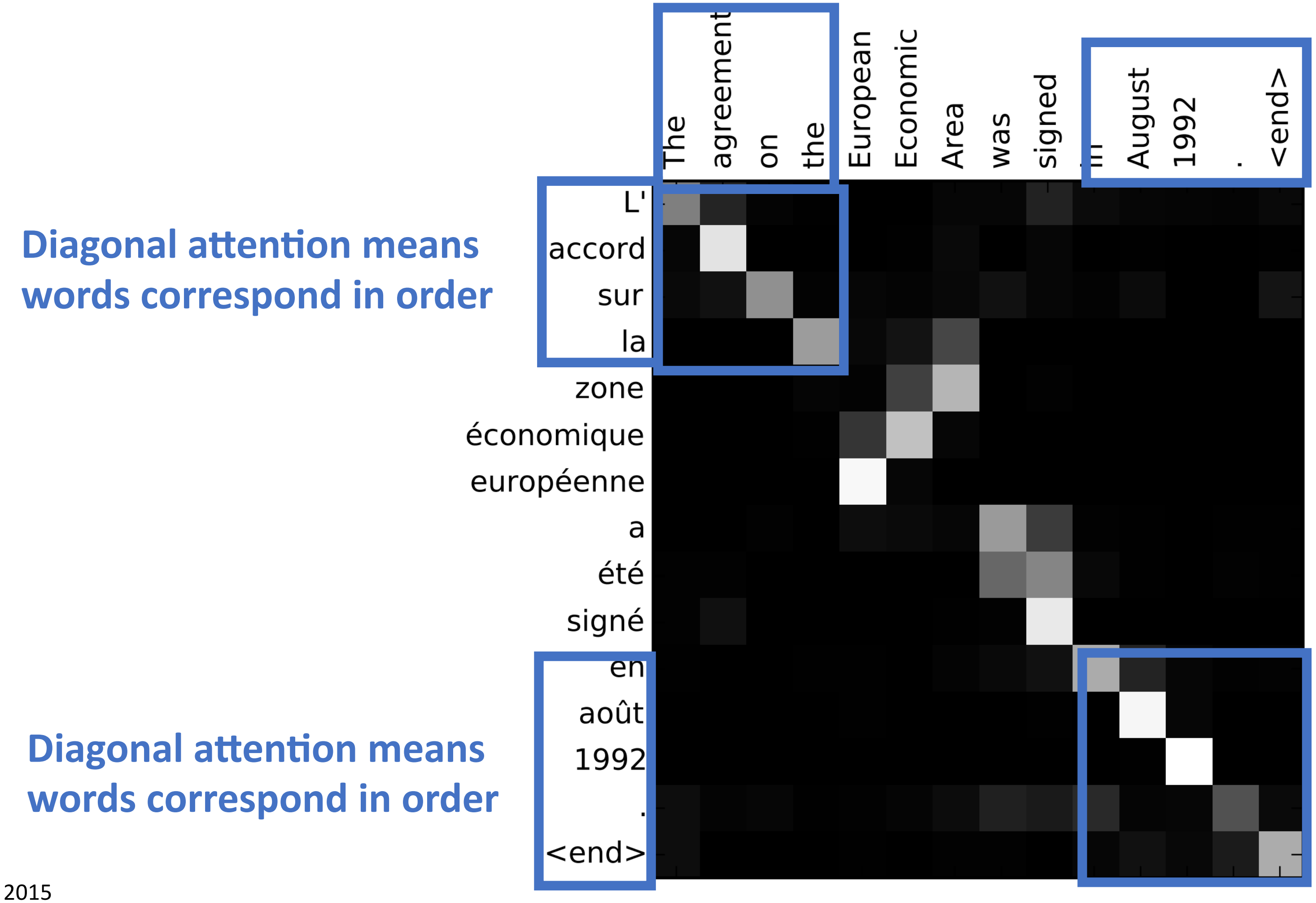
Sequence-to-Sequence with RNNs **and Attention**

Example: English to French translation

Input: "**The agreement on the** European Economic Area was signed **in August 1992**."

Output: "**L'accord sur la** zone économique européenne a été signé **en août 1992**."

Visualize attention weights $a_{t,i}$



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

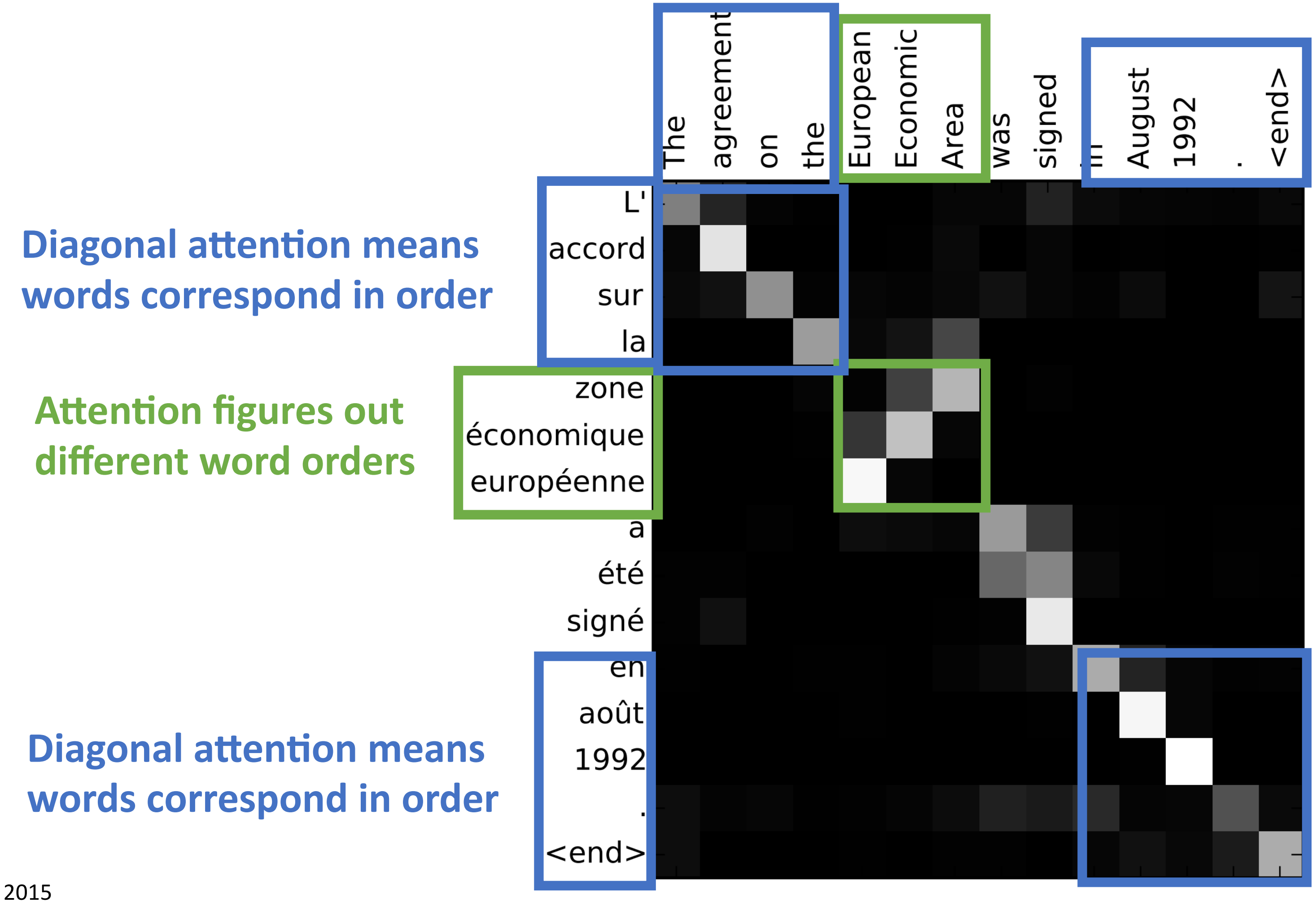
Sequence-to-Sequence with RNNs and Attention

Example: English to French translation

Input: "The agreement on the European Economic Area was signed in August 1992."

Output: "L'accord sur la zone économique européenne a été signé en août 1992."

Visualize attention weights $a_{t,i}$



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**

Example: English to French translation

Input: "The agreement on the European Economic Area was signed in August 1992."

Output: "L'accord sur la zone économique européenne a été signé en août 1992."

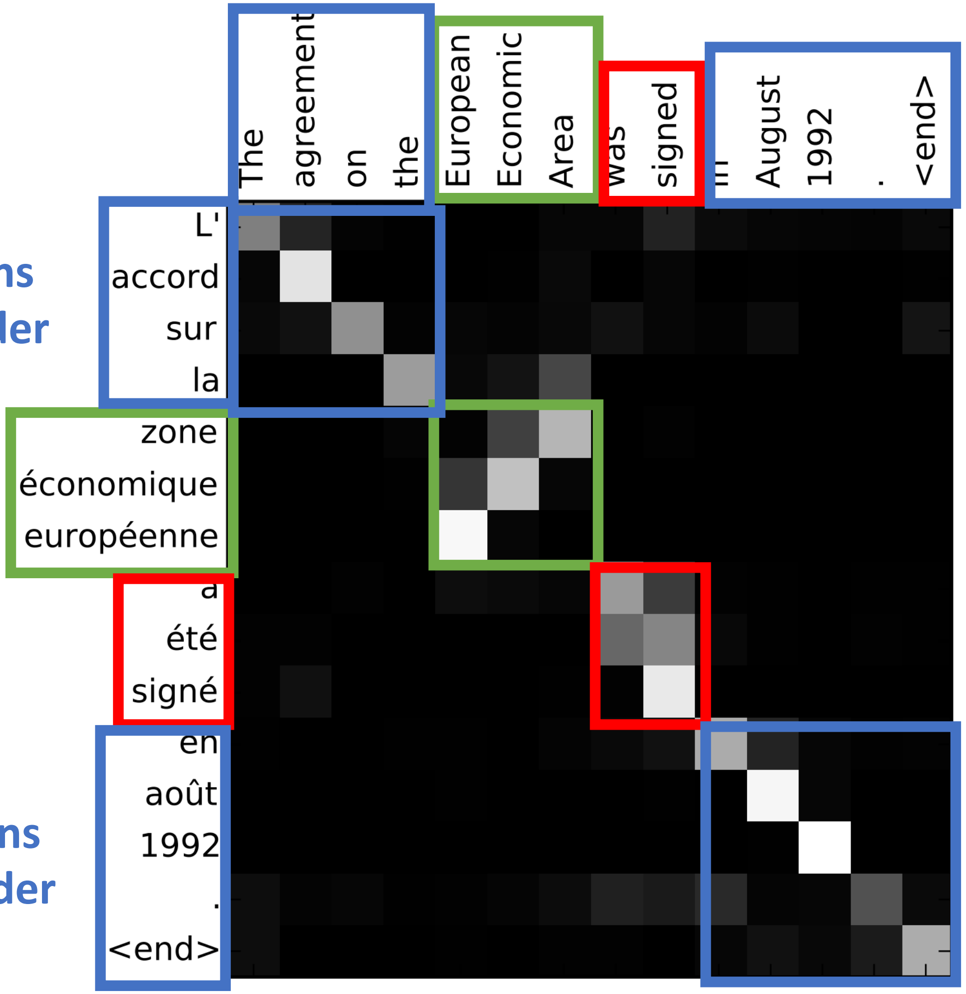
Visualize attention weights $a_{t,i}$

Diagonal attention means words correspond in order

Attention figures out different word orders

Verb conjugation

Diagonal attention means words correspond in order

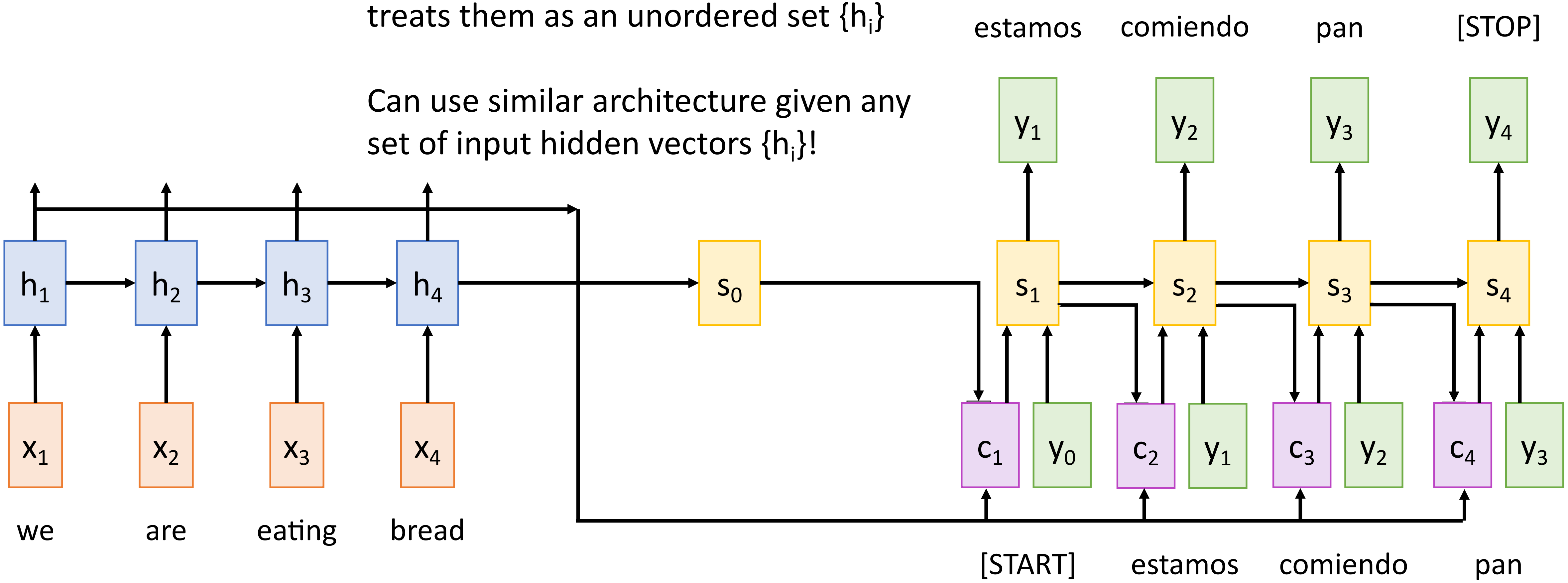


Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**

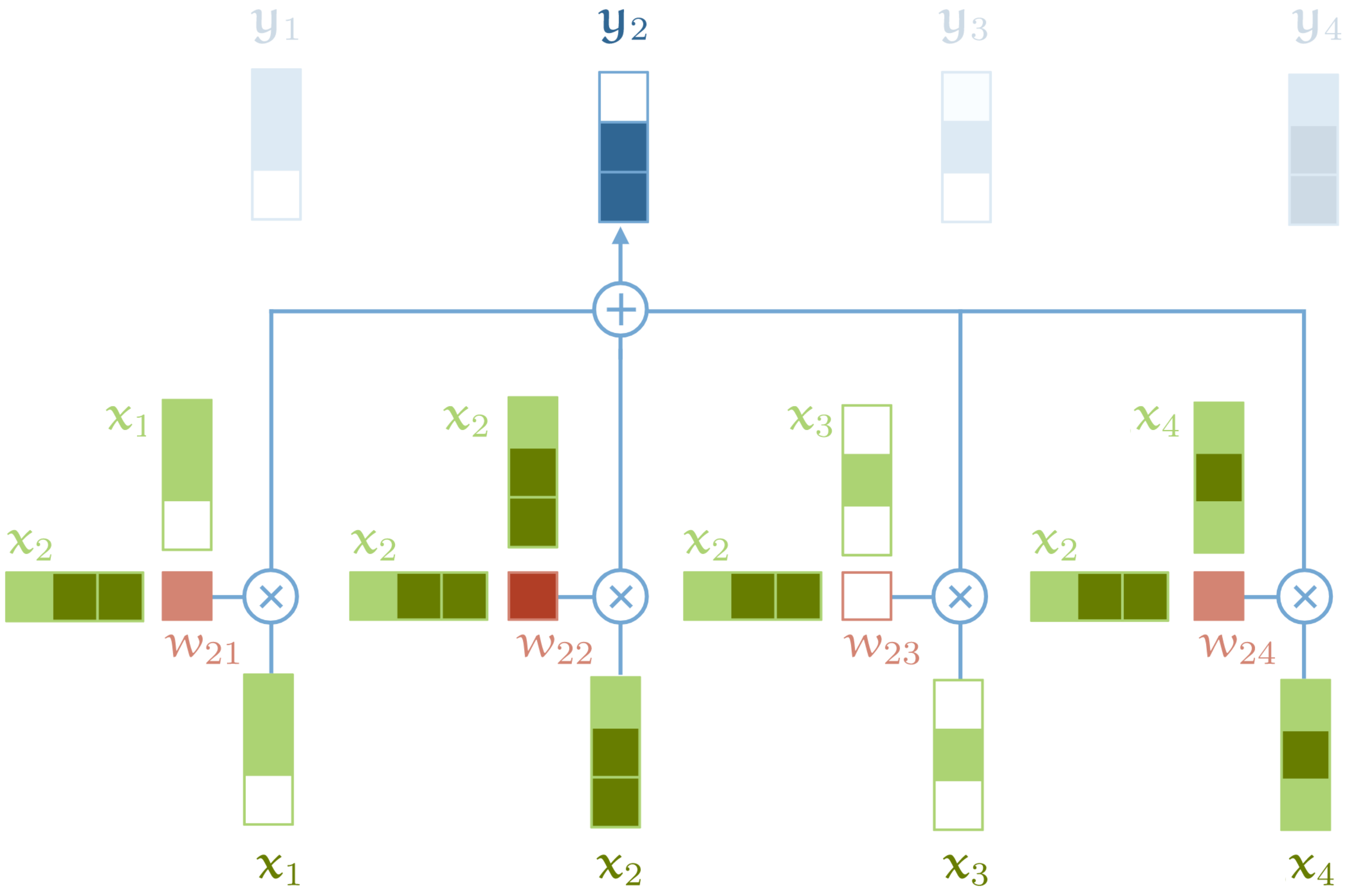
The decoder doesn't use the fact that h_i form an ordered sequence – it just treats them as an unordered set $\{h_i\}$

Can use similar architecture given any set of input hidden vectors $\{h_i\}$!



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Attention



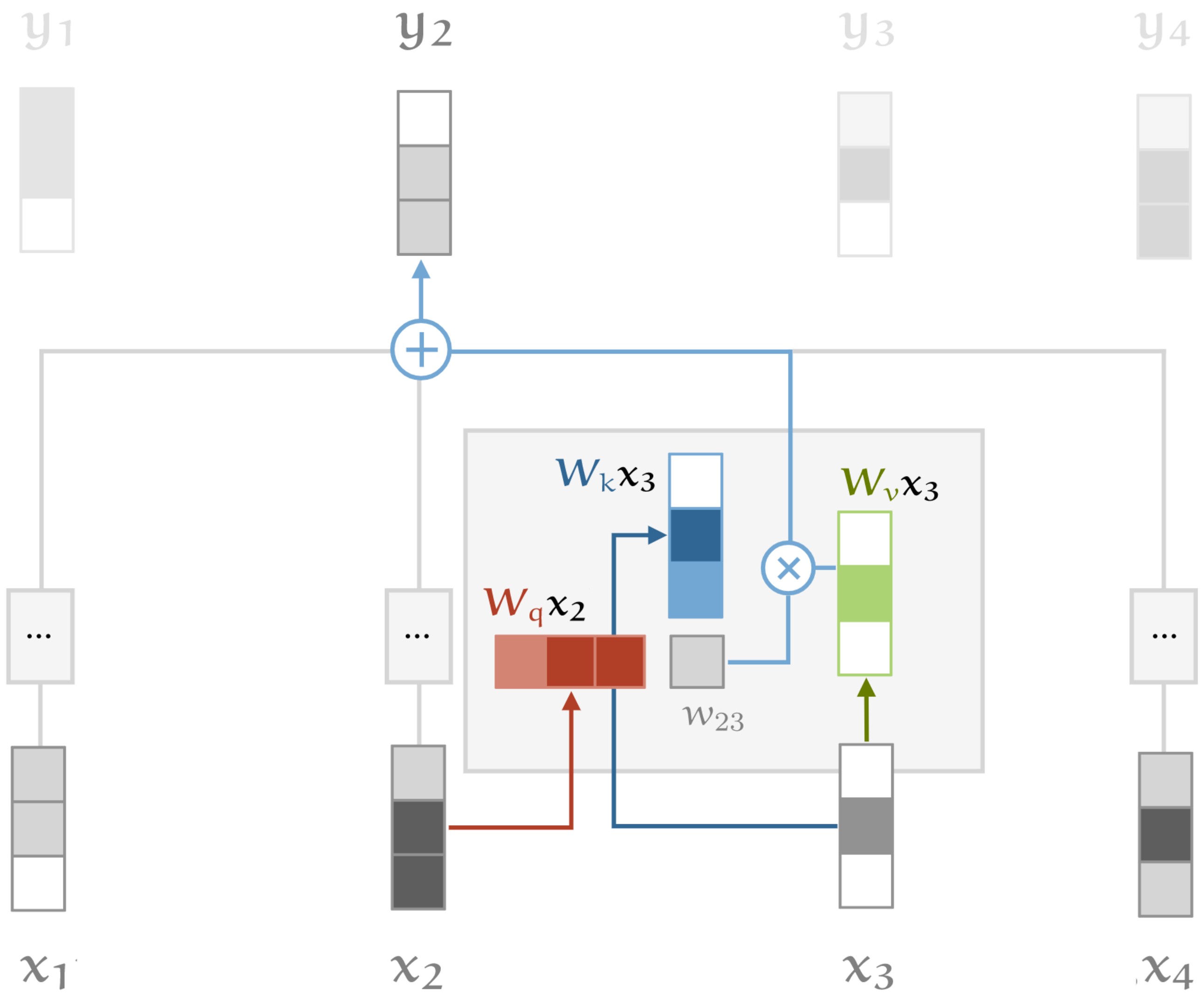
$$y_i = \sum_j w_{ij} x_{ij}$$

$$w_{ij} = softmax_j(x_i^T x_j / \sqrt{d_k})$$

$$w_{ij} = \frac{e^{x_i^T x_j}}{\sum_j e^{x_i^T x_j}}$$

Source: http://peterbloem.nl/blog/transformers See also: [Attention is all you need]()

Attention (with key, query and value)

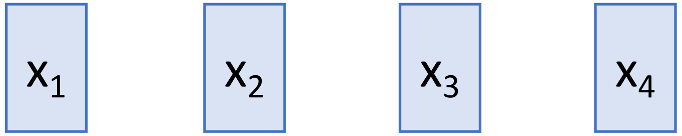


$$y_i = \sum_j w_{ij} \boldsymbol{W}_v \boldsymbol{x}_{ij}$$

$$w_{ij} = softmax_j((\boldsymbol{W}_q \boldsymbol{x}_i)^T \boldsymbol{W}_k \boldsymbol{x}_j / \sqrt{d_k})$$

Source: http://peterbloem.nl/blog/transformers See also: [Attention is all you need]()

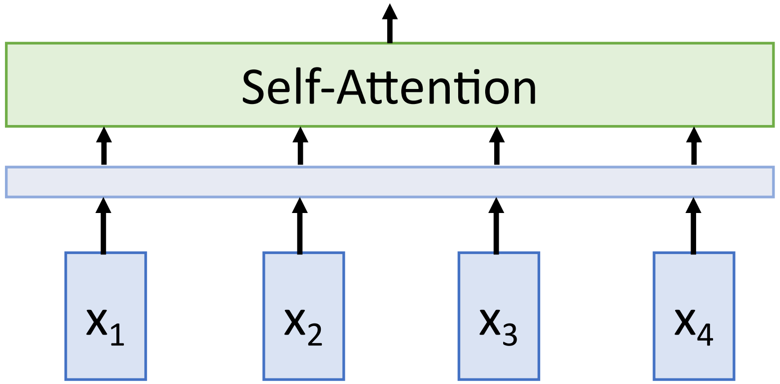
The Transformer



Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer

All vectors interact with each other

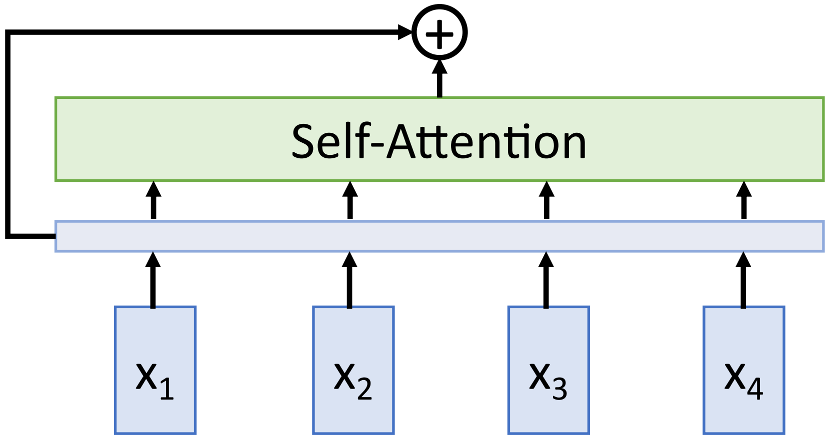


Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer

Residual connection

All vectors interact with each other



Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

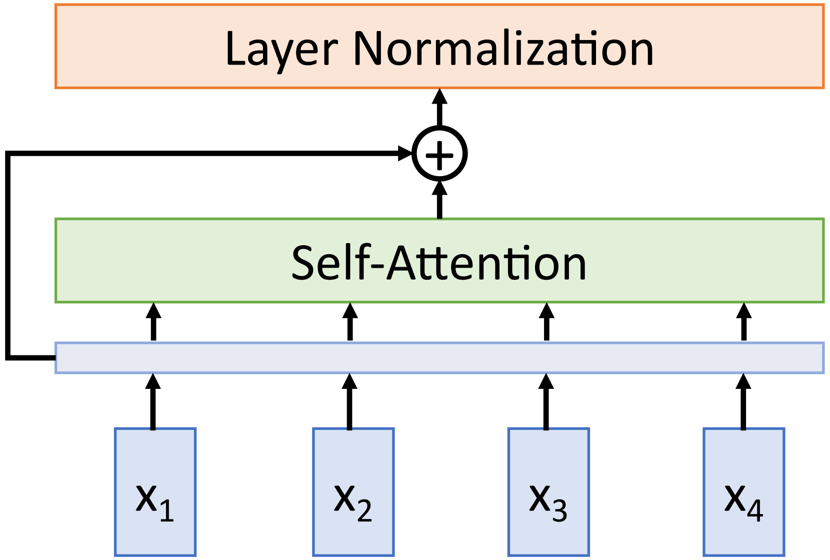
$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

Ba et al, 2016

Residual connection

All vectors interact with each other



Vaswani et al, "Attention is all you need", NeurIPS 2017

Slide from Justin Johnson

The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

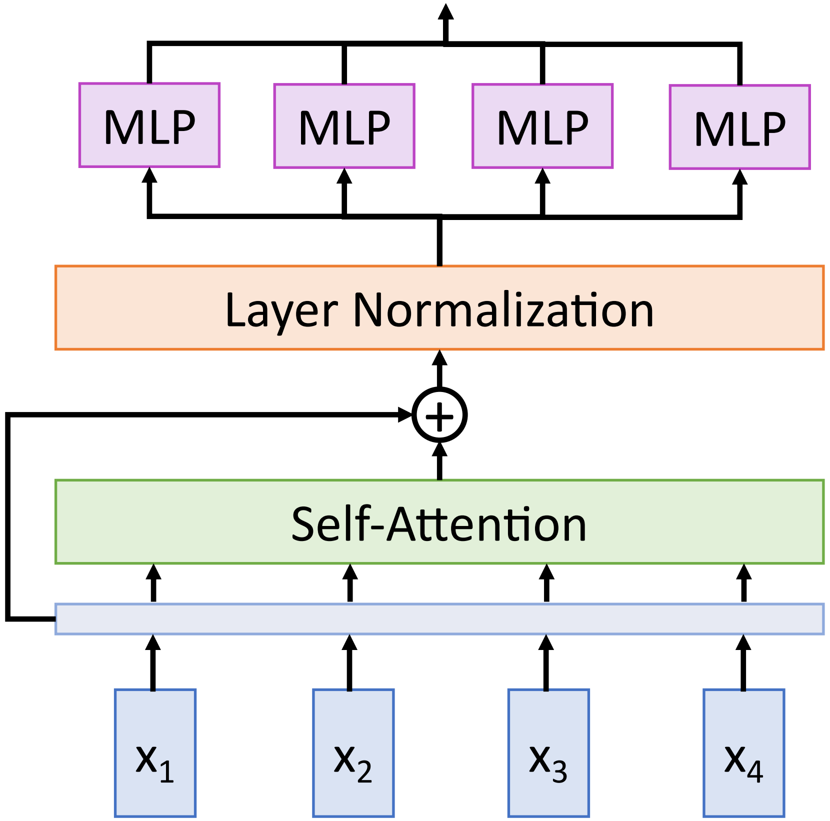
$y_i = \gamma * z_i + \beta$

Ba et al, 2016

MLP independently on each vector

Residual connection

All vectors interact with each other



Vaswani et al, "Attention is all you need", NeurIPS 2017

Slide from Justin Johnson

The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

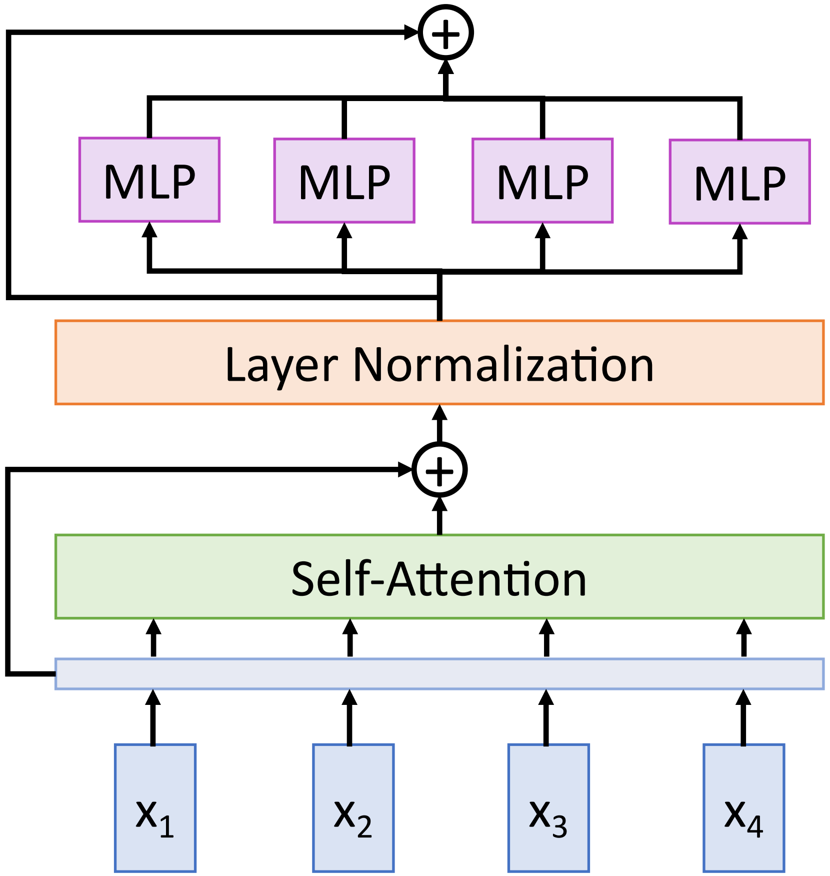
Ba et al, 2016

Residual connection

MLP independently on each vector

Residual connection

All vectors interact with each other



Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

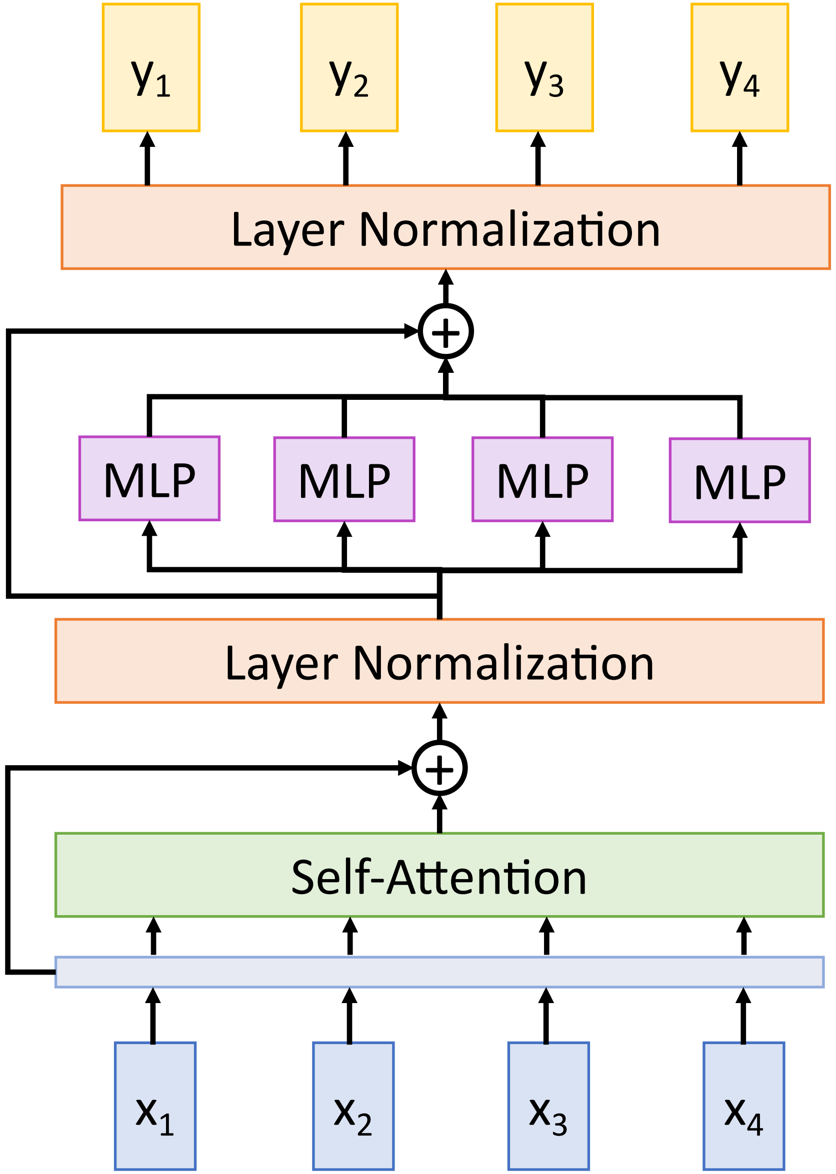
Ba et al, 2016

Residual connection

MLP independently on each vector

Residual connection

All vectors interact with each other



Vaswani et al, "Attention is all you need", NeurIPS 2017

Slide from Justin Johnson

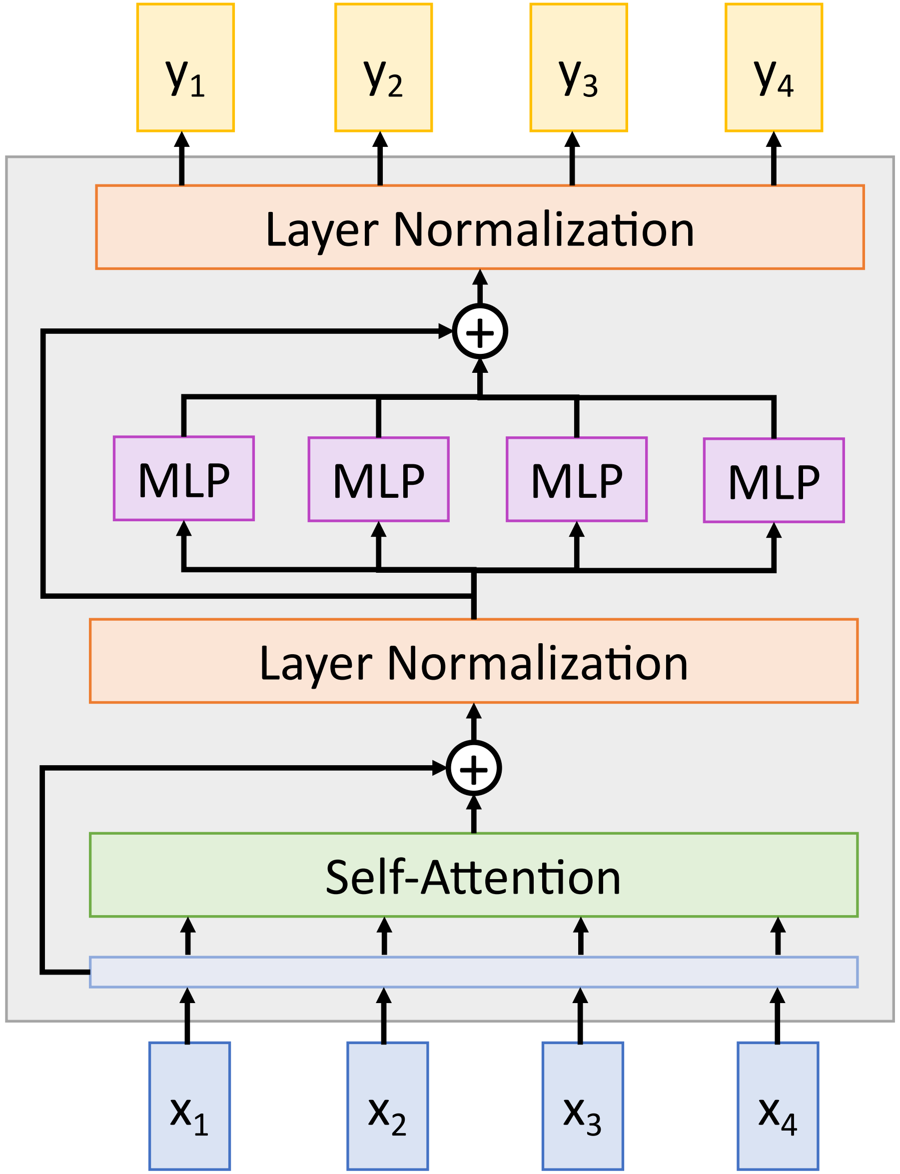
The Transformer

Transformer Block:
Input: Set of vectors x
Output: Set of vectors y

Self-attention is the only interaction between vectors!

Layer norm and MLP work independently per vector

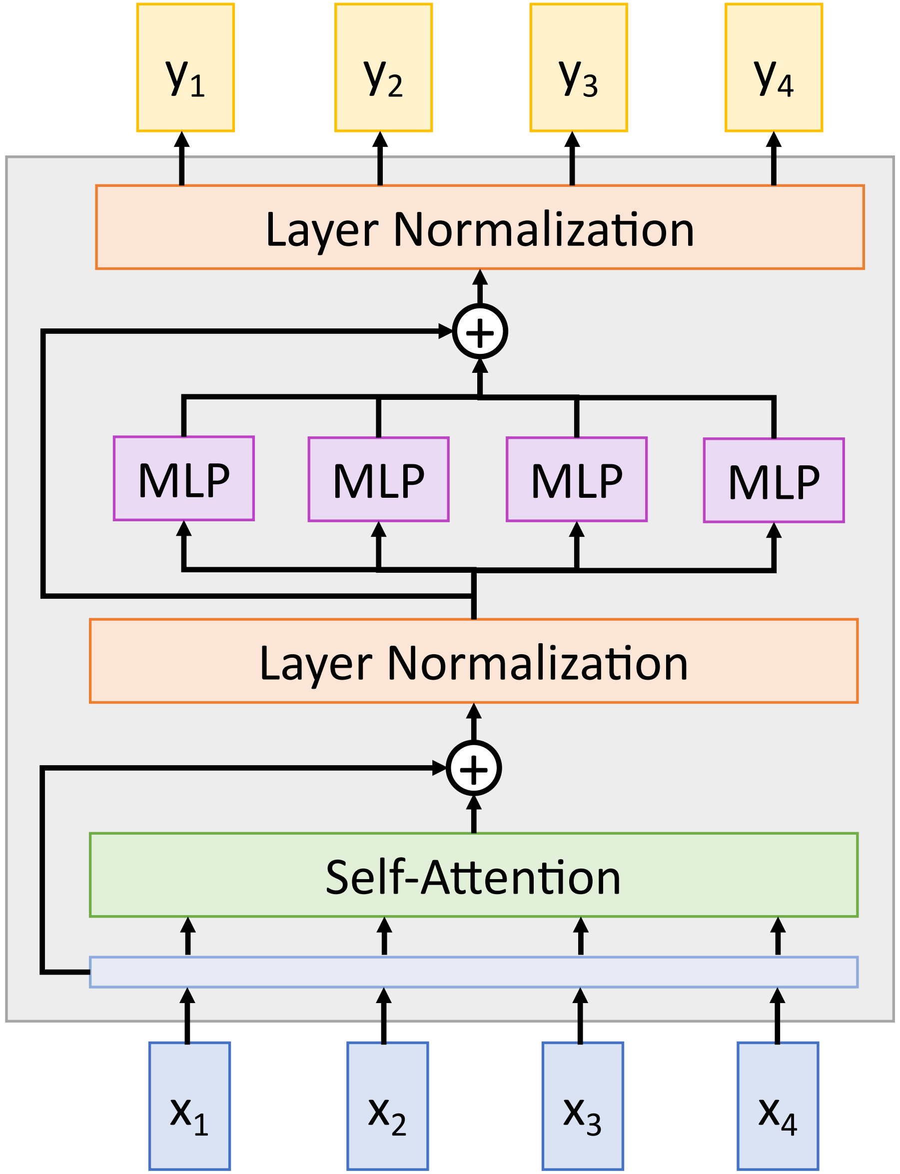
Highly scalable, highly parallelizable



Vaswani et al, "Attention is all you need", NeurIPS 2017

Slide from Justin Johnson

Post-Norm Transformer

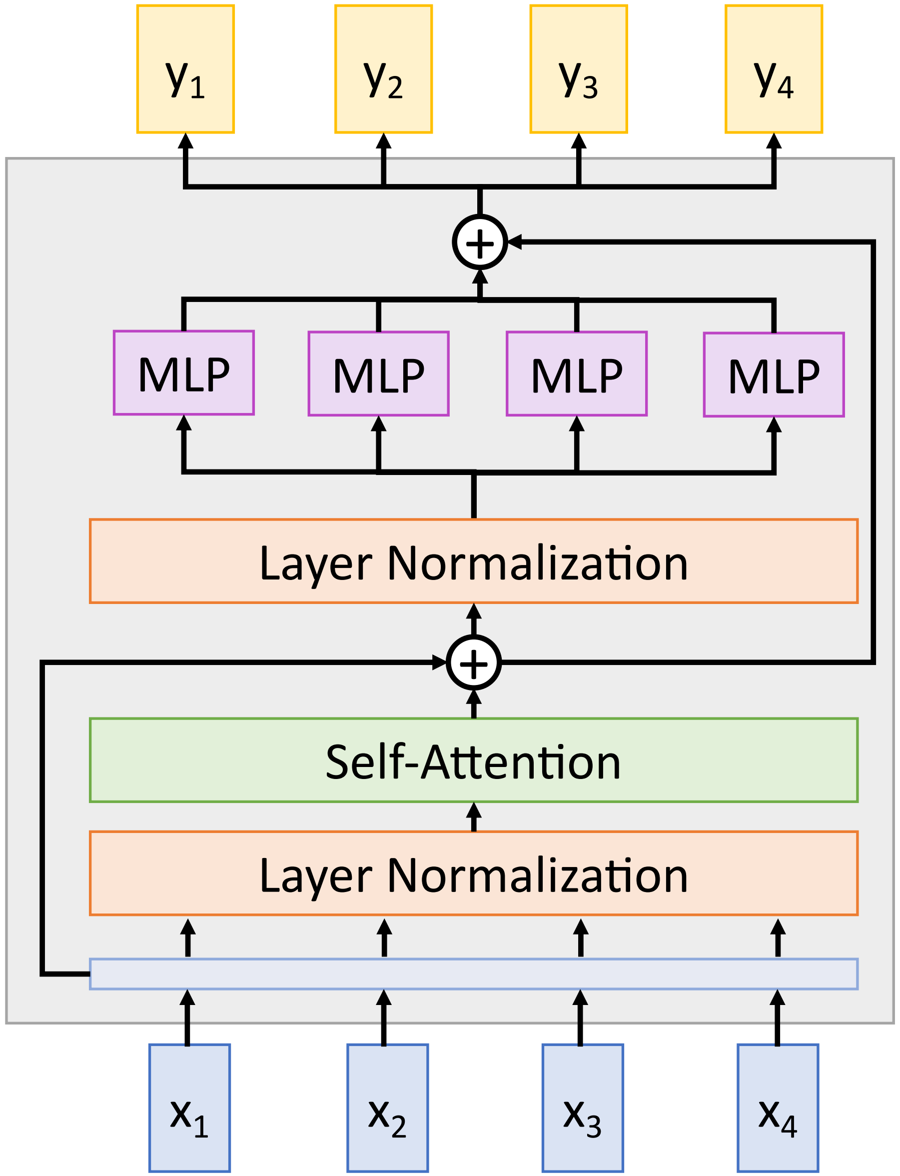


Layer normalization is **after** residual connections

Vaswani et al, "Attention is all you need", NeurIPS 2017

Slide from Justin Johnson

Pre-Norm Transformer



Layer normalization is **inside** residual connections

Gives more stable training, commonly used in practice

Baevski & Auli, "Adaptive Input Representations for Neural Language Modeling", arXiv 2018

Slide from Justin Johnson

The Transformer

<u>Transformer Block</u>:
Input: Set of vectors x
Output: Set of vectors y

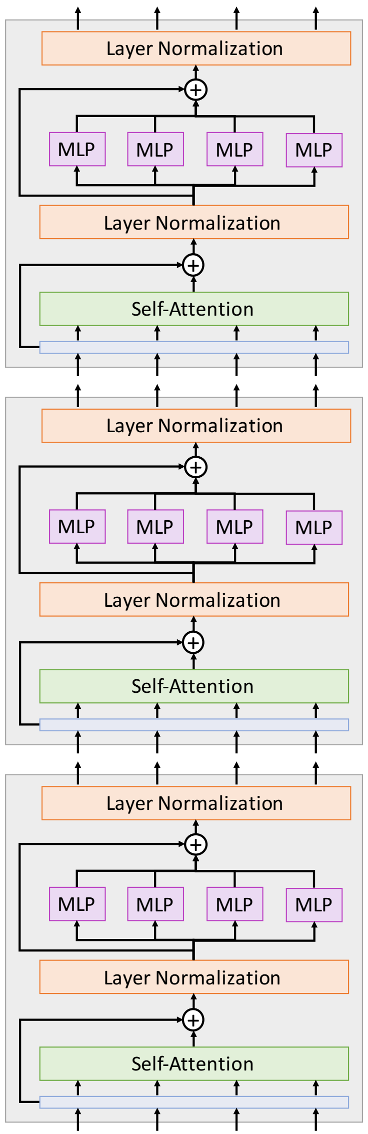
Self-attention is the only interaction between vectors!

Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable

A **Transformer** is a sequence of transformer blocks

Vaswani et al:
12 blocks, D_Q=512, 6 heads



Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Vaswani et al, "Attention is all you need", NeurIPS 2017

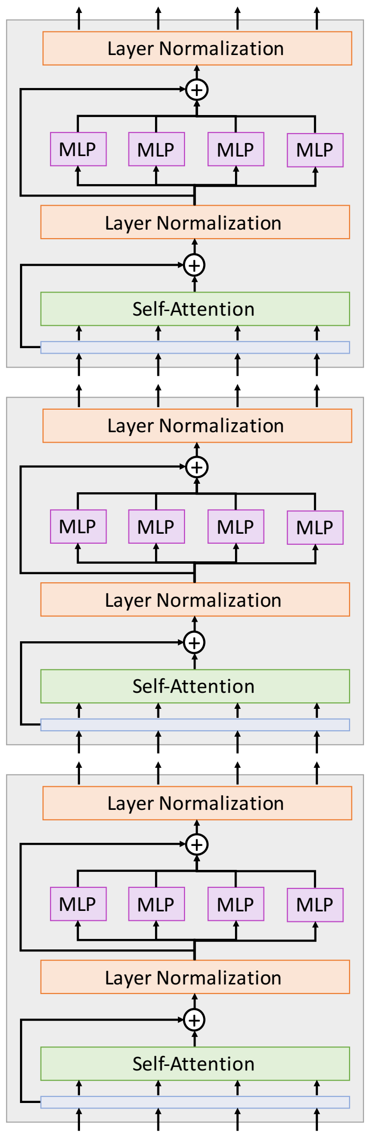
The Transformer: Transfer Learning

"ImageNet Moment for Natural Language Processing"

Pretraining:
Download a lot of text from the internet

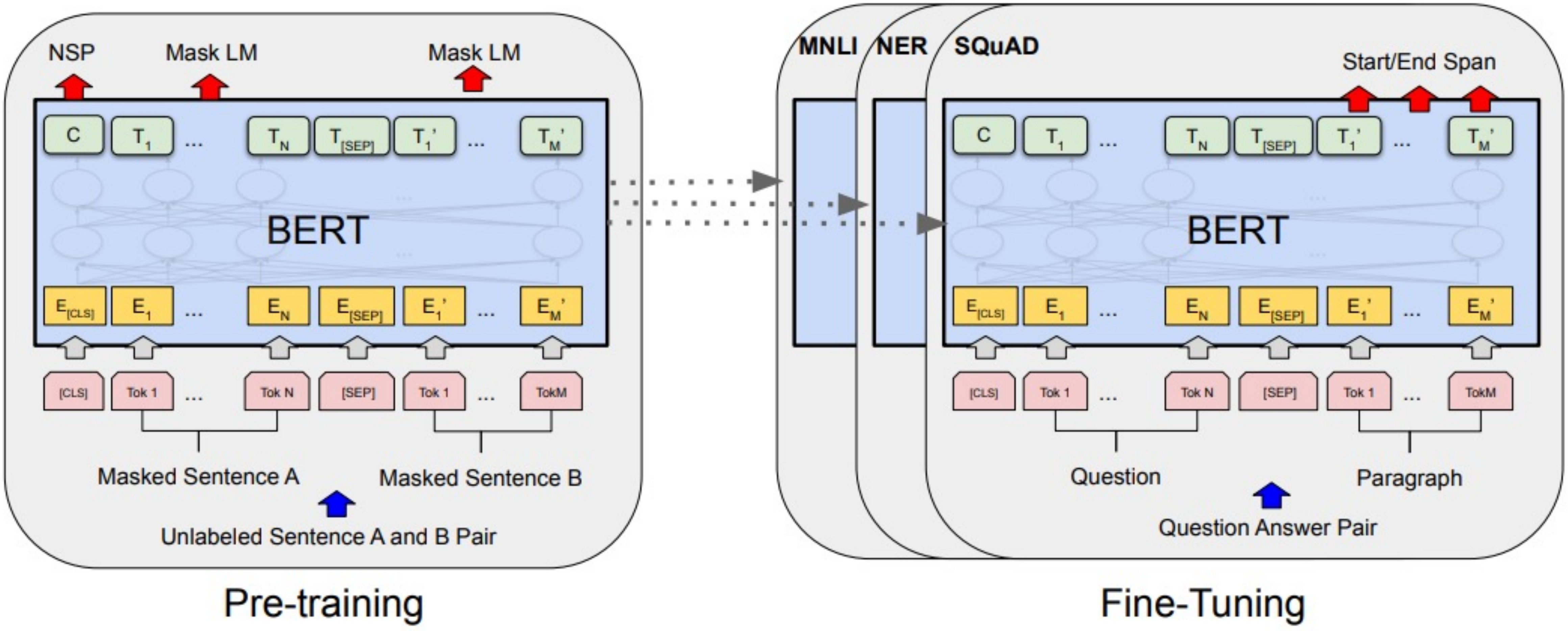
Train a giant Transformer model for language modeling

Finetuning:
Fine-tune the Transformer on your own NLP task



Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", EMNLP 2018

The Transformer: Transfer Learning



Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", EMNLP 2018

The Transformer: Transfer Learning

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", EMNLP 2018