Deep Learning with Neural Networks

Sequential Learning with Recurrent Neural Networks

Pablo Martínez Olmos, pamartin@ing.uc3m.es

A motivating example: predict the next word

A motivating example: predict the next word

"This morning I took my cat for a walk."

given these words

predict the next word

A motivating example: predict the next word

Idea #1: use a fixed window

"This morning I took my cat for a walk."

given these two words

predict the next word

One-hot feature encoding: tells us what each word is

[1 0 0 0 0 0 1 0 0 0]

for

a

prediction

A motivating example: predict the next word

Idea #1: use a fixed window

"This morning I took my cat for a walk."

given these two words — predict the next word

One-hot feature encoding: tells us what each word is

[1 0 0 0 0 0 1 0 0 0]

for — a

prediction

Problem #1: can't model long-term dependencies

"**France** is where I grew up, but I now live in Boston. I speak fluent ____."

A motivating example: predict the next word

"This morning I took my cat for a walk."

given these words

predict the next word

A motivating example: predict the next word

Idea #2: use entire sequence as set of counts

"This morning I took my cat for a"

"bag of words"

[0 1 0 0 1 0 0 ... 0 0 1 1 0 0 0 1]

prediction

A motivating example: predict the next word

Idea #2: use entire sequence as set of counts

"This morning I took my cat for a"

↓

"bag of words"

Problem #2: counts don't preserve order

The food was good, not bad at all.

vs.

The food was bad, not good at all.

A motivating example: predict the next word

"This morning I took my cat for a walk."

given these words

predict the next word

A motivating example: predict the next word

Idea #3: use a really big fixed window

"This morning I took my cat for a walk."

given these words

predict the next word

[1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 ...]

morning I took this cat

prediction

A motivating example: predict the next word

Idea #3: use a really big fixed window

"This morning I took my cat for a walk."

given these words

predict the next word

[1 0 0 0 0 0 0

morning

Problem #3: no parameter sharing

[1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 ...]

this morning took the cat

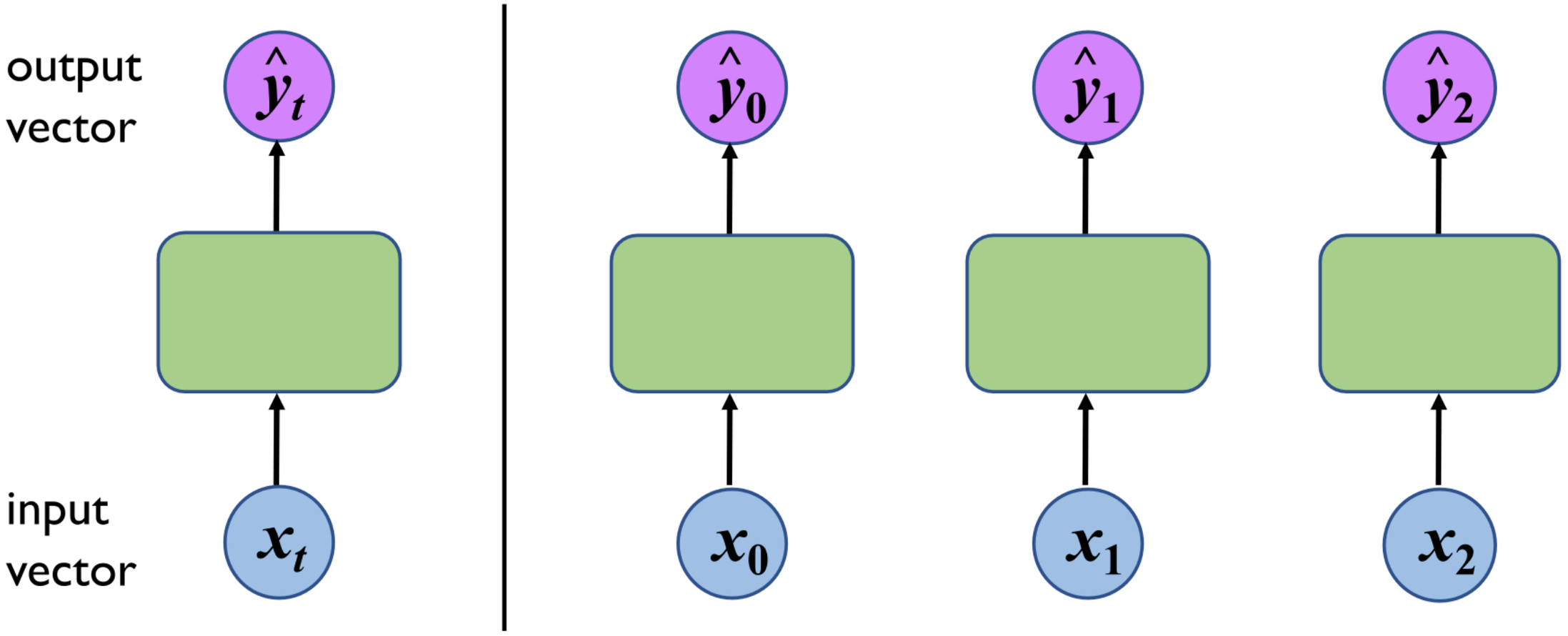
Each of these inputs has a **separate parameter:**

[0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 ...]

this morning

Things we learn about the sequence **won't transfer** if they appear **elsewhere** in the sequence.

Handling individual time steps

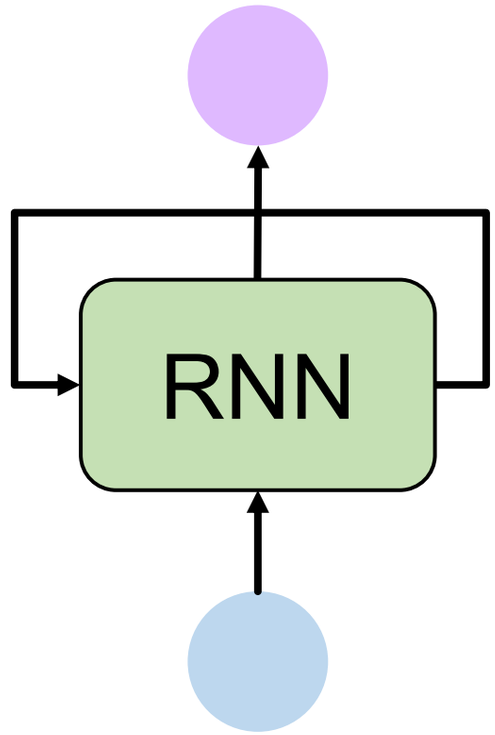


$$\hat{y}_t = f(x_t)$$

Sequence modeling: design criteria

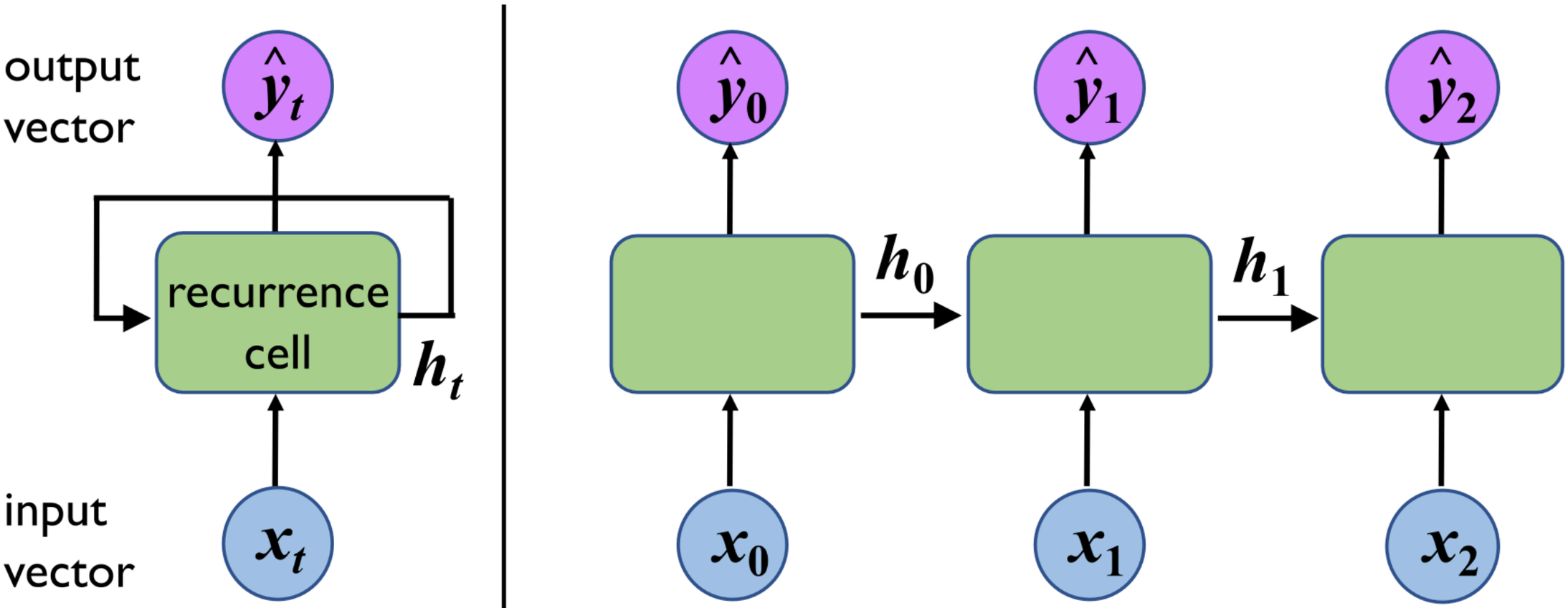
To modeling sequences, we need to:

1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence



Today: **Recurrent Neural Networks (RNNs)** as an approach to sequence modeling problems

Neurons with recurrence



$$\hat{y}_t = f(x_t, h_{t-1})$$

output input past memory

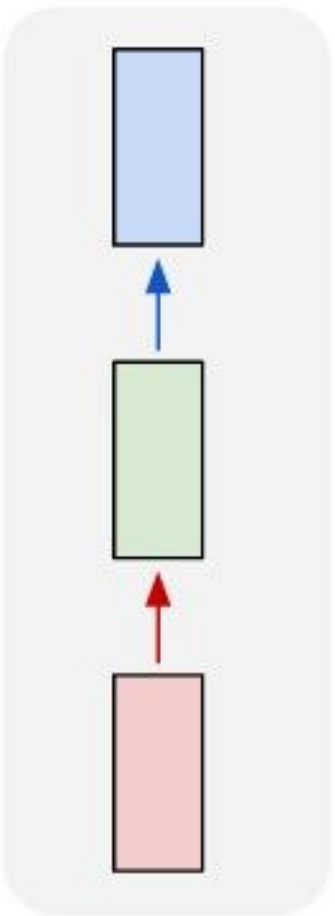
Recurrent Neural Networks

uc3m

Recurrent Neural Networks come in many flavours

Recurrent Neural Networks come in many flavours

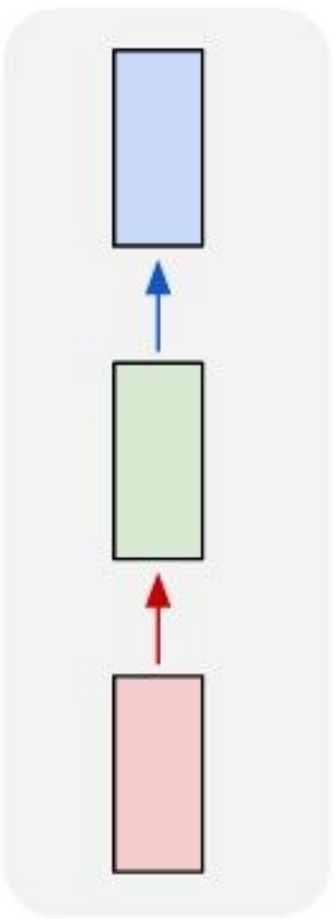
one to one



"Vanilla" NN

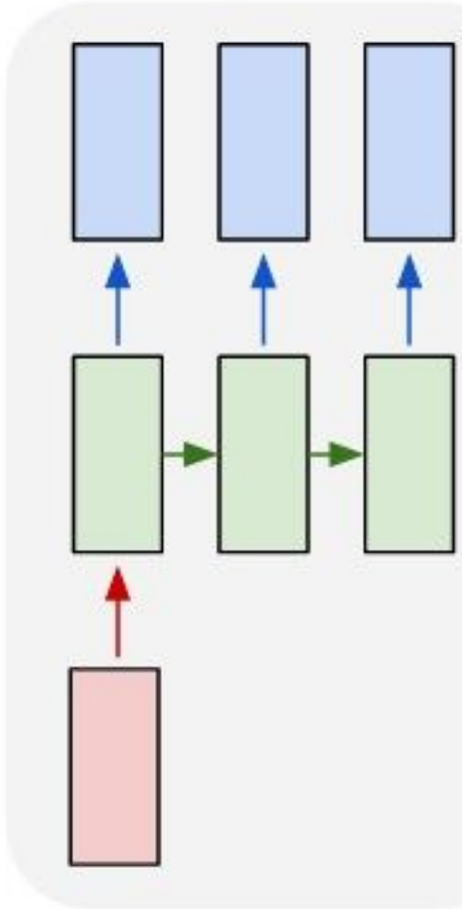
Recurrent Neural Networks come in many flavours

one to one



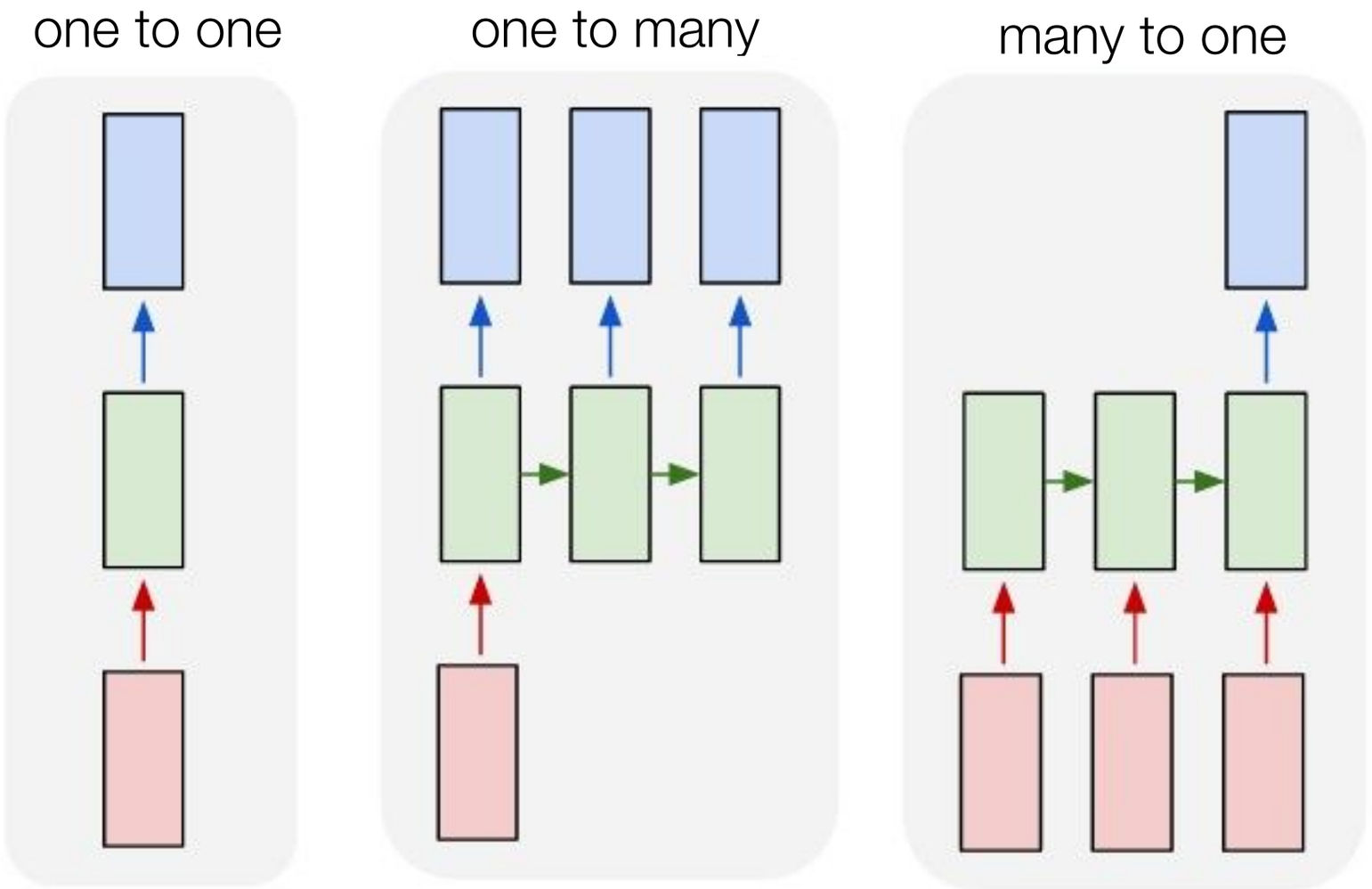
“Vanilla” NN

one to many



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

Recurrent Neural Networks come in many flavours



“Vanilla” NN

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

e.g. **Sentiment Classification**
sequence of words -> sentiment

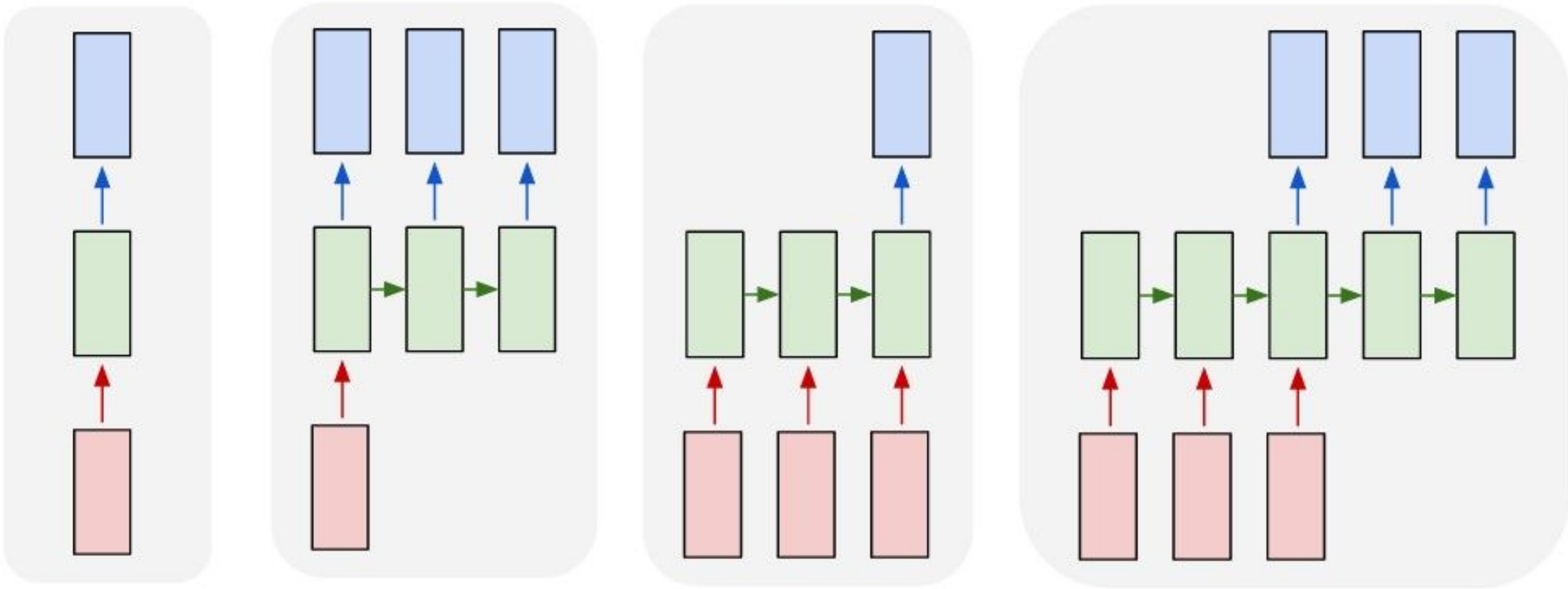
Recurrent Neural Networks come in many flavours

one to one

one to many

many to one

many to many



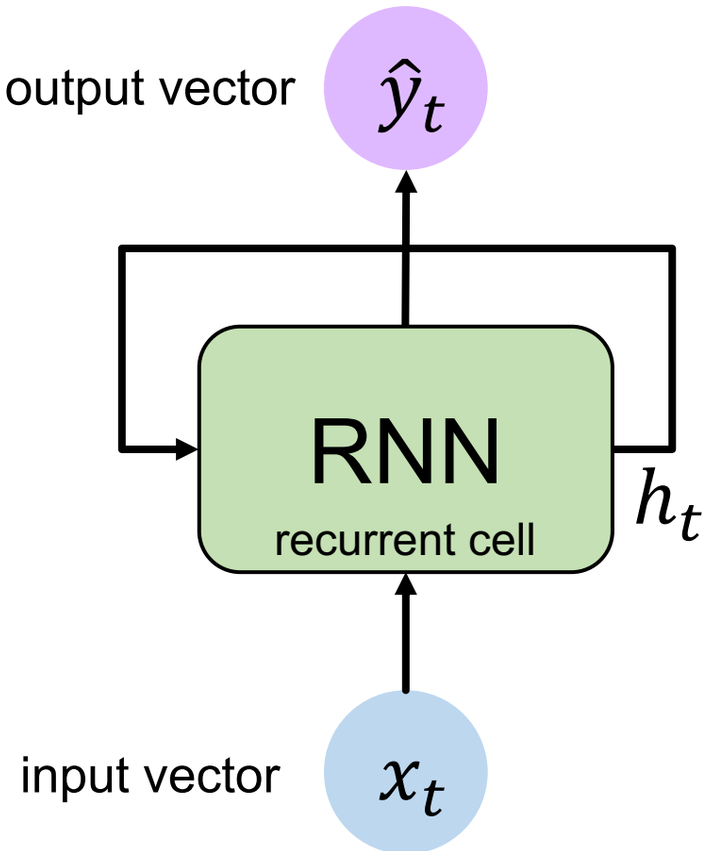
"Vanilla" NN

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

e.g. **Sentiment Classification**
sequence of words -> sentiment

e.g. **Machine Translation**
seq. of words -> seq. of words

The standard RNN

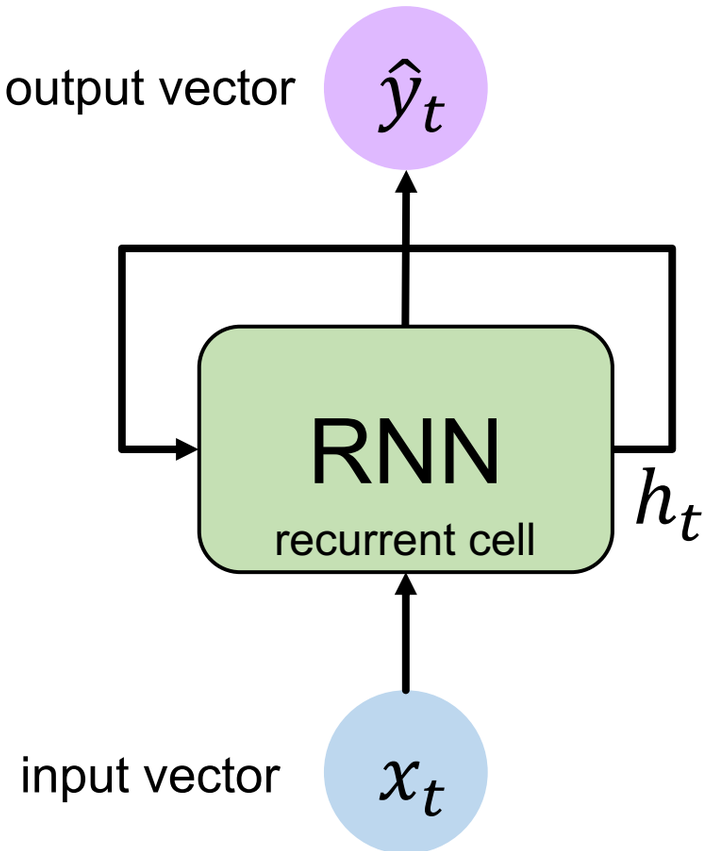


Apply a **recurrence relation** at every time step to process a sequence:

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

The same function and set of parameters are used **at every time step**

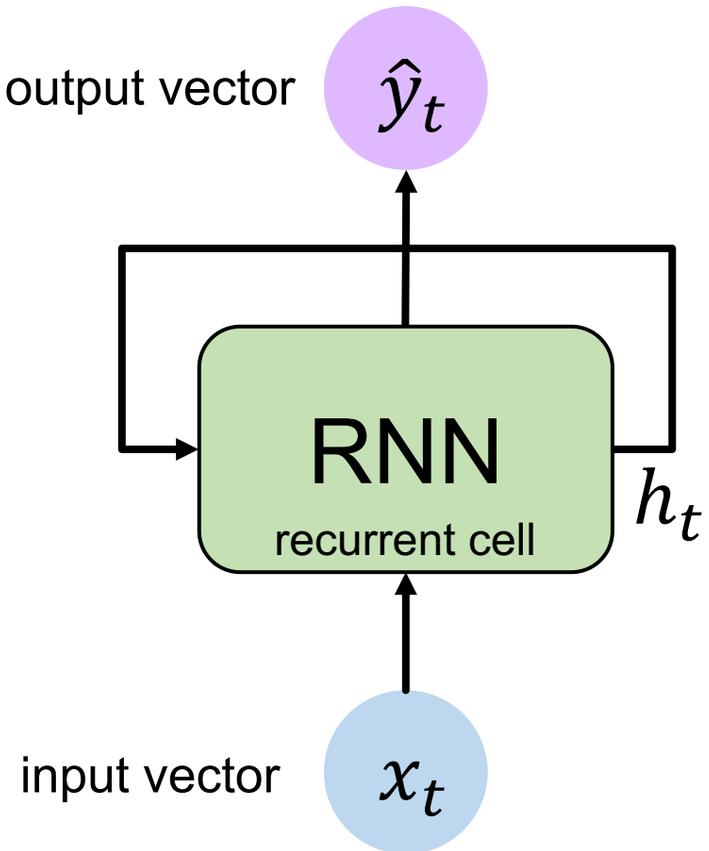
The standard RNN



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

$$\hat{y}_t = \mathbf{W}_{hy} h_t$$

The standard RNN

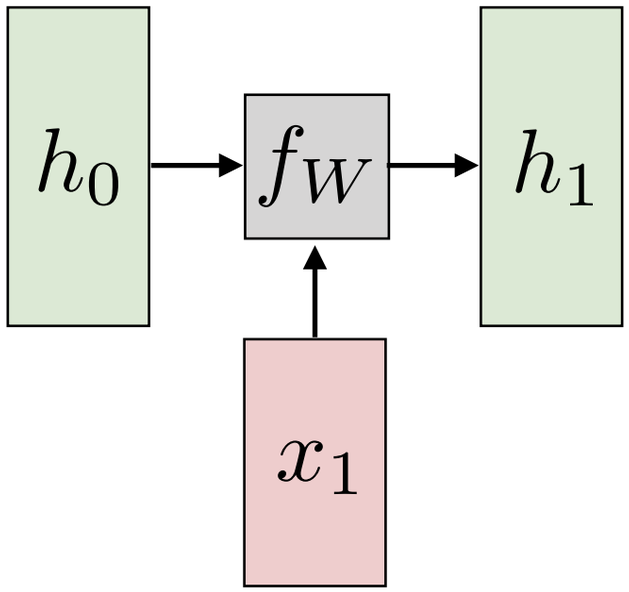


$$h_t = \tanh\left(\boldsymbol{W_{hh}} h_{t-1} + \boldsymbol{W_{xh}} x_t\right)$$

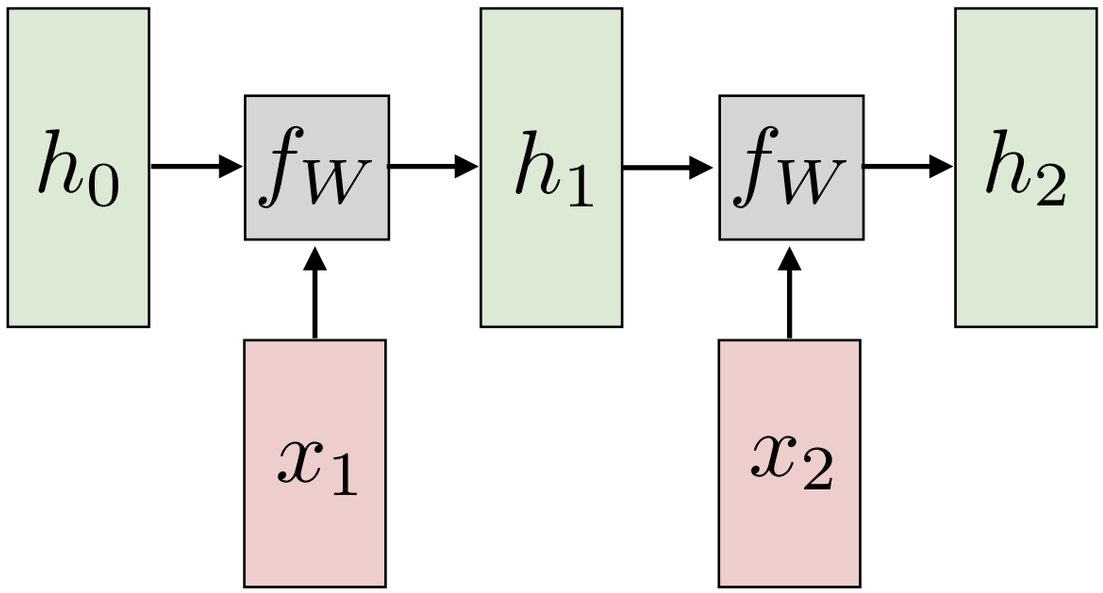
$$\hat{y}_t = \mathbf{W}_{hy} h_t$$

Apply a feed-forward dense layer to get final prediction

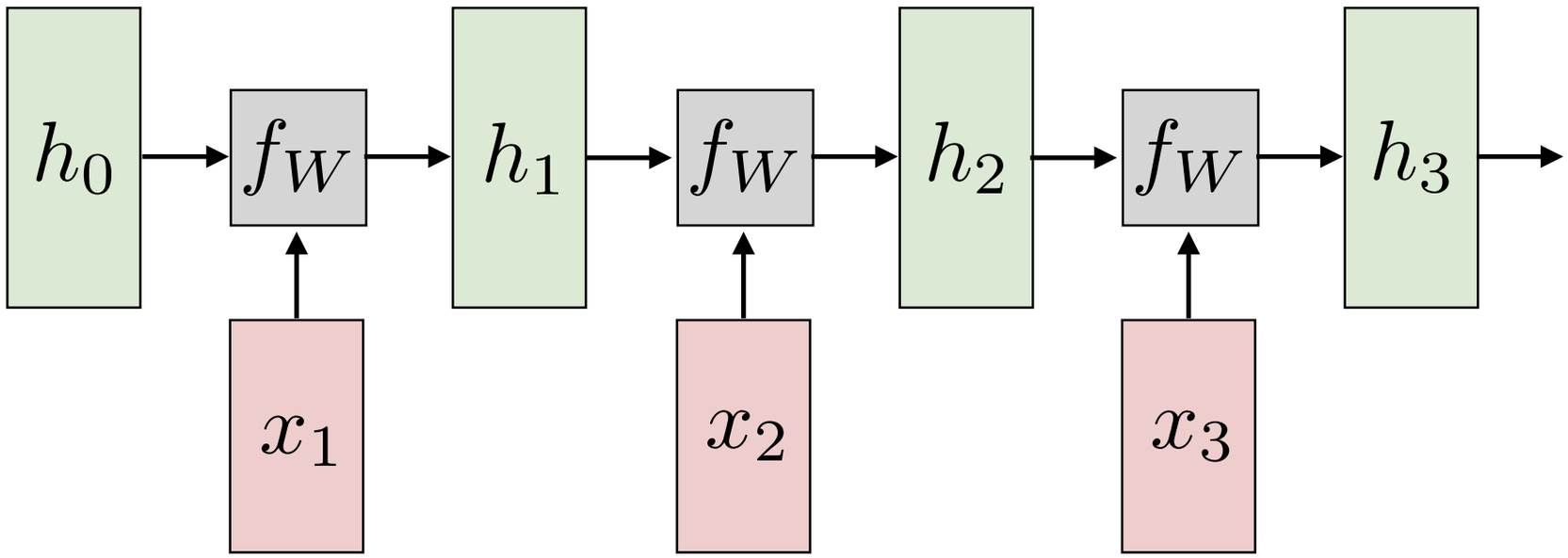
RNN computational graph



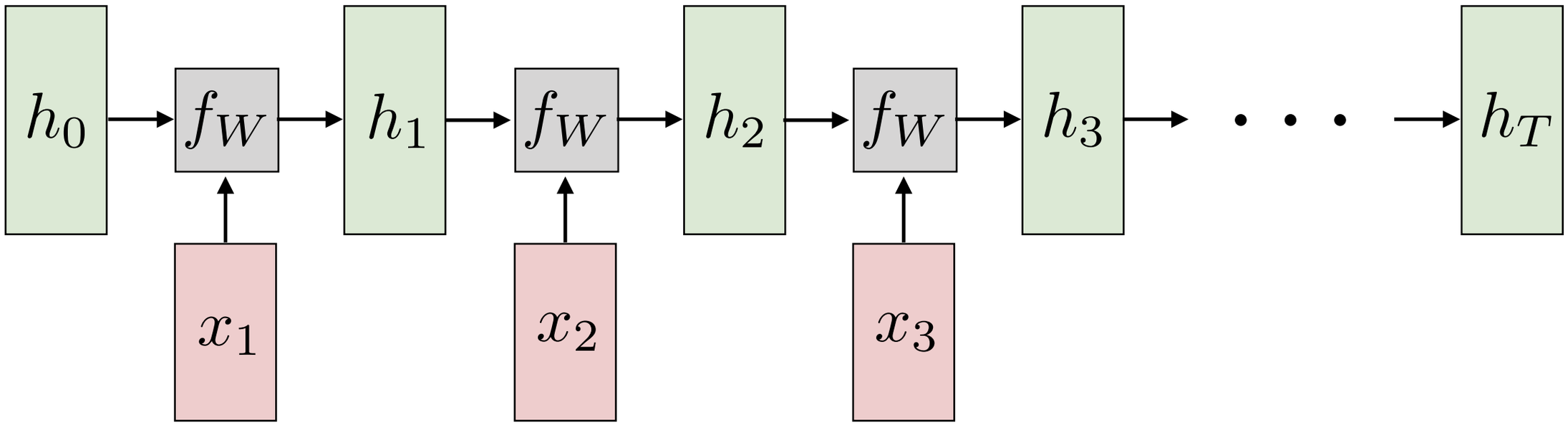
RNN computational graph



RNN computational graph

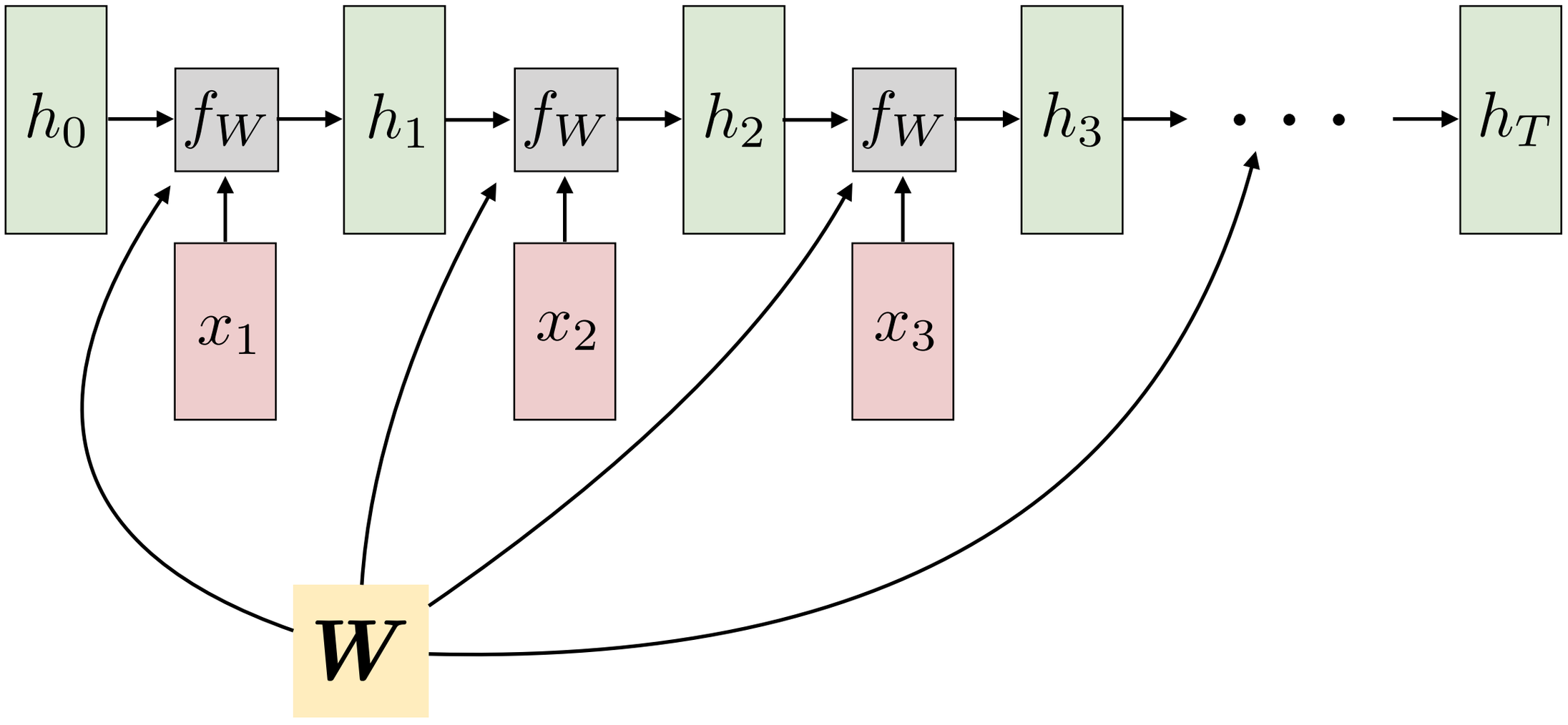


RNN computational graph



RNN computational graph

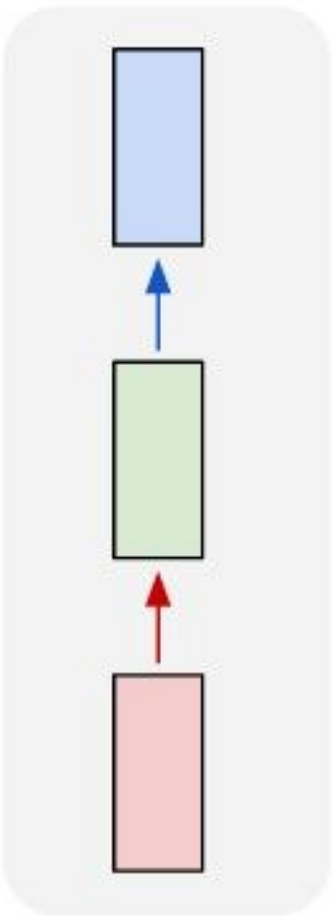
Re-use the same weight matrix at every time-step



Recurrent Neural Networks come in many flavours

Recurrent Neural Networks come in many flavours

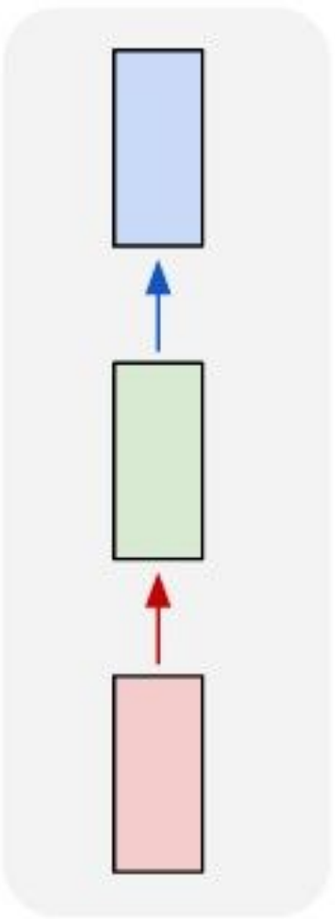
one to one



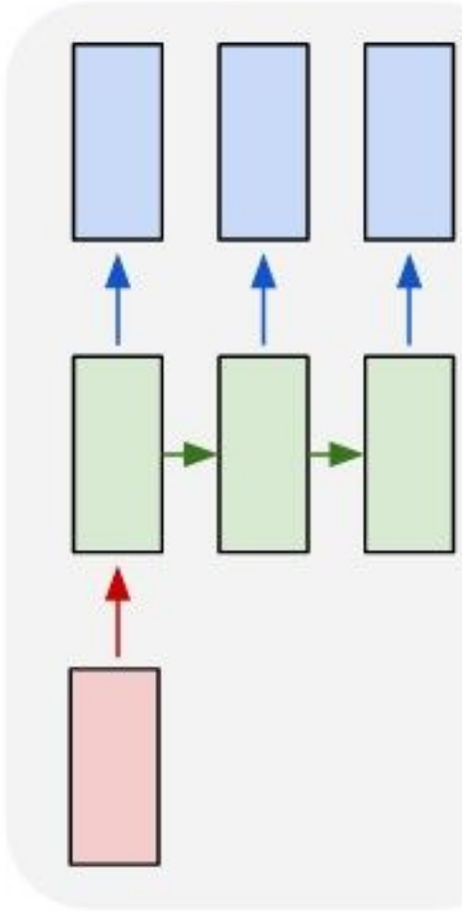
"Vanilla" NN

Recurrent Neural Networks come in many flavours

one to one



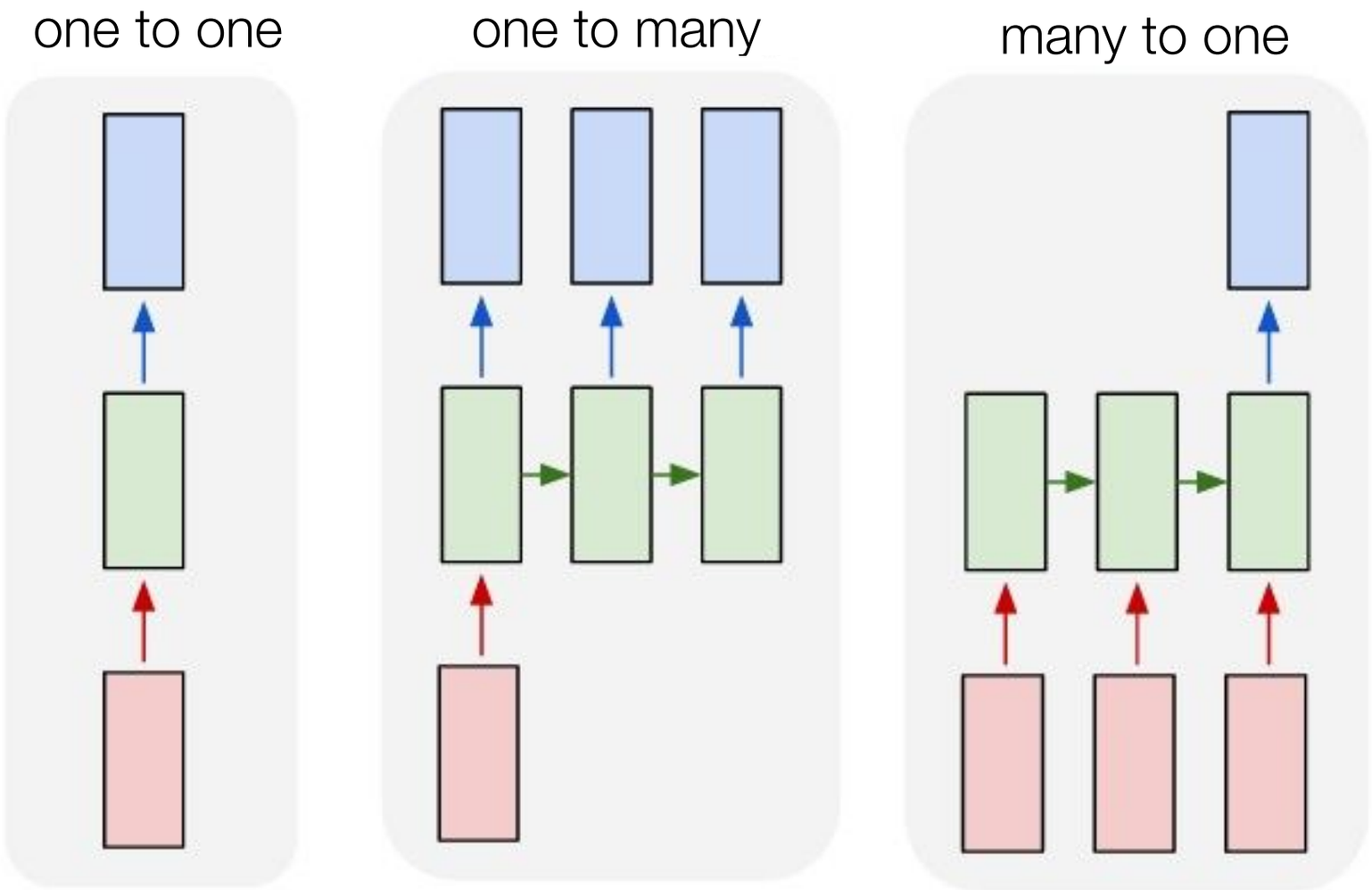
one to many



“Vanilla” NN

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

Recurrent Neural Networks come in many flavours



"Vanilla" NN

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

e.g. **Sentiment Classification**
sequence of words -> sentiment

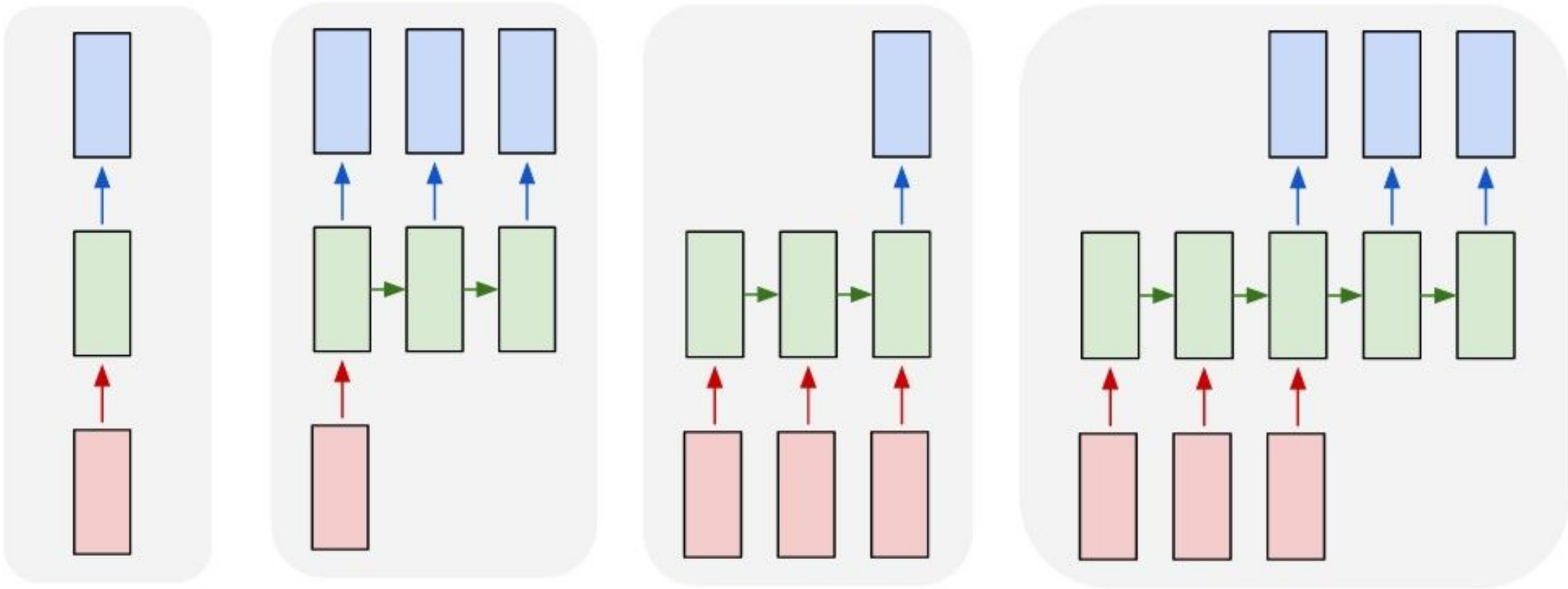
Recurrent Neural Networks come in many flavours

one to one

one to many

many to one

many to many



"Vanilla" NN

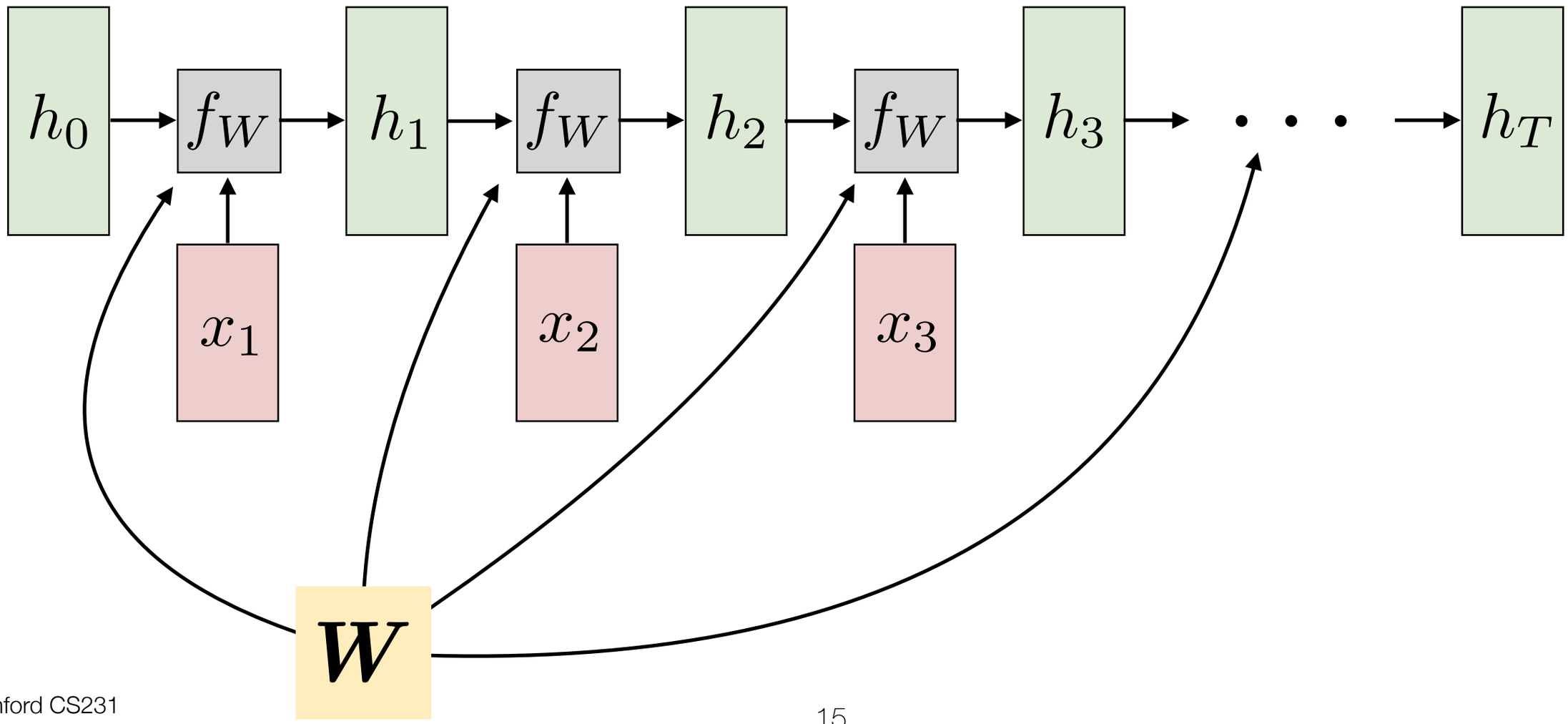
e.g. **Image Captioning**
image -> sequence of words

e.g. **Sentiment Classification**
sequence of words -> sentiment

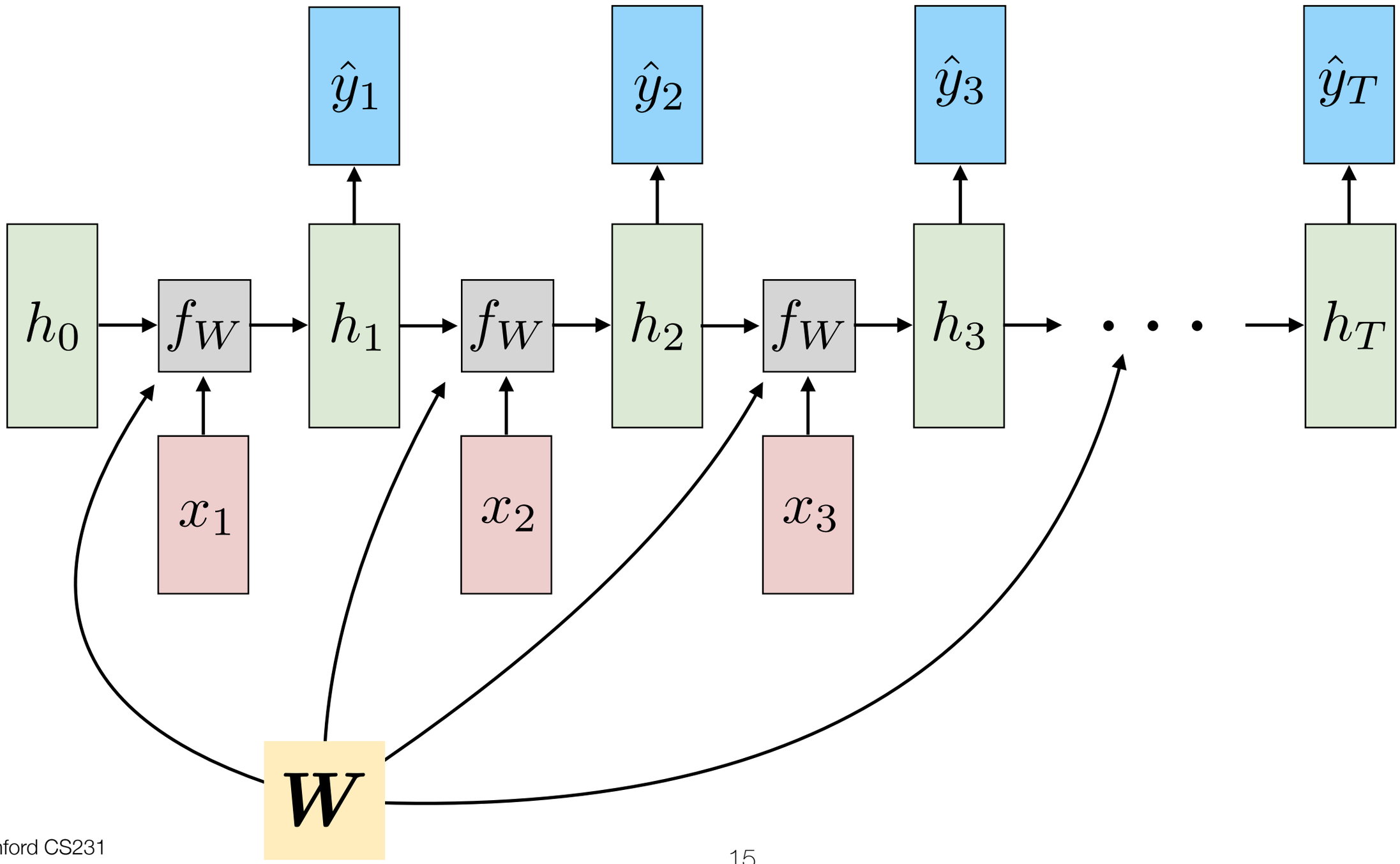
e.g. **Machine Translation**
seq. of words -> seq. of words

RNN computational graph (many to many)

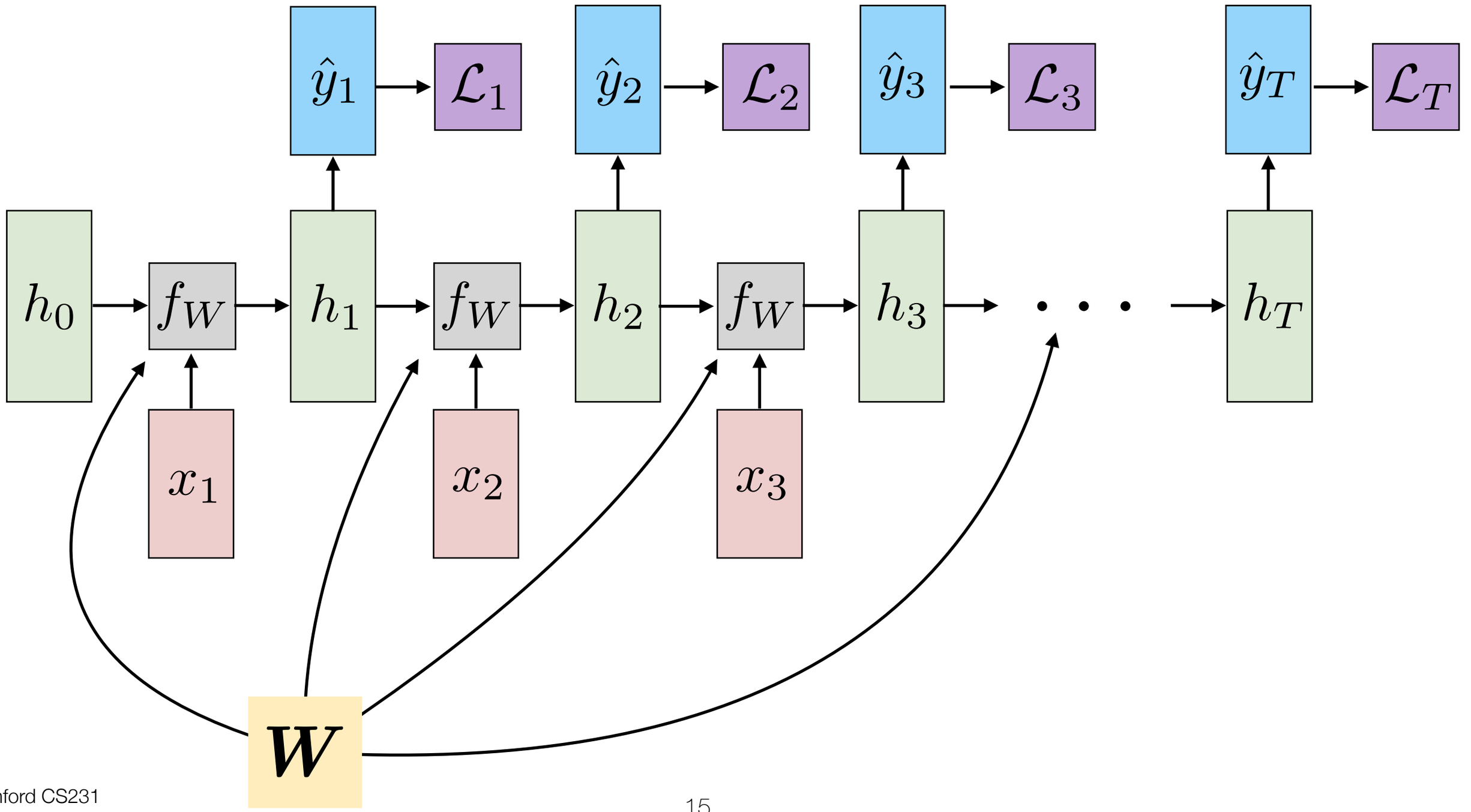
RNN computational graph (many to many)



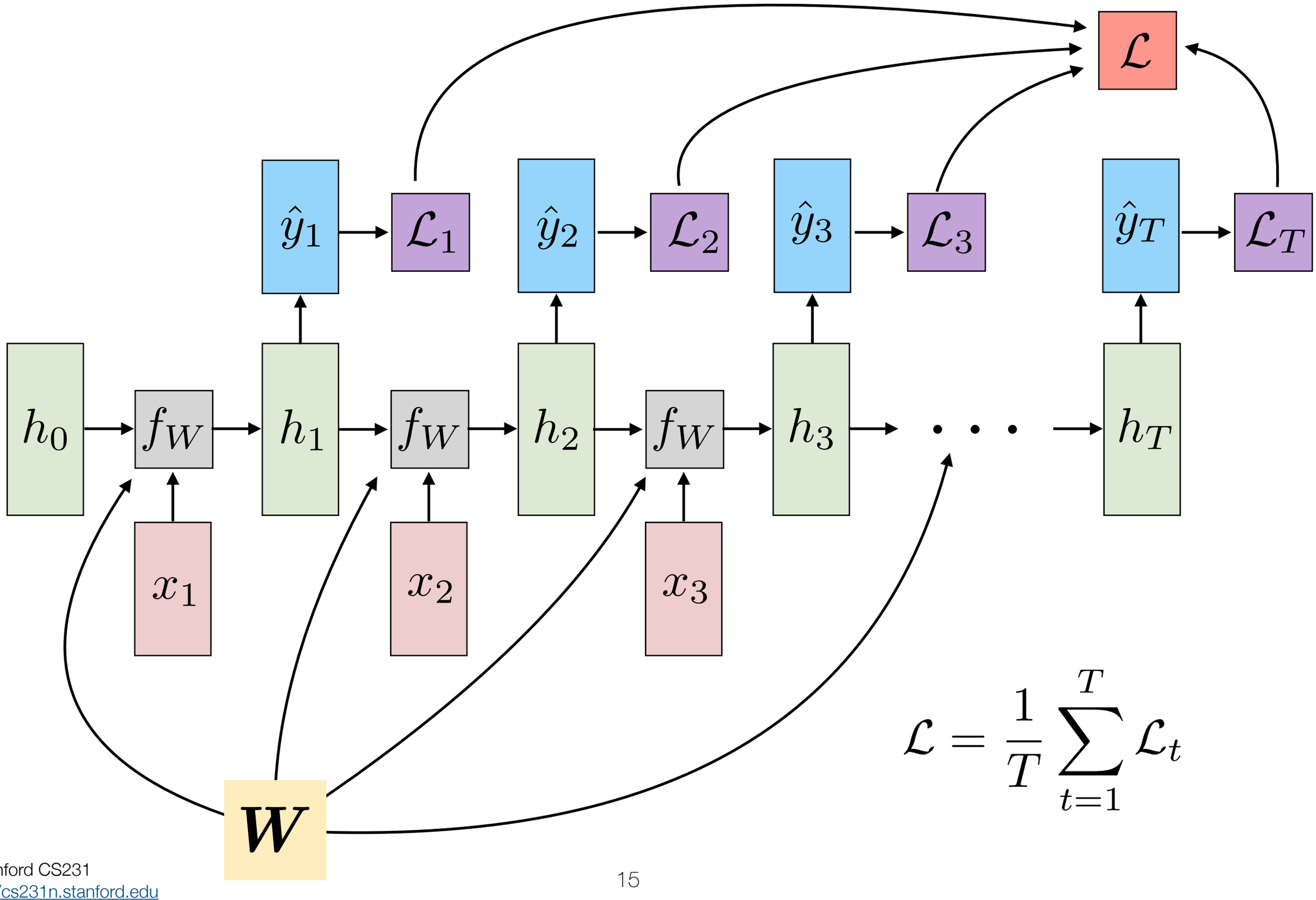
RNN computational graph (many to many)



RNN computational graph (many to many)



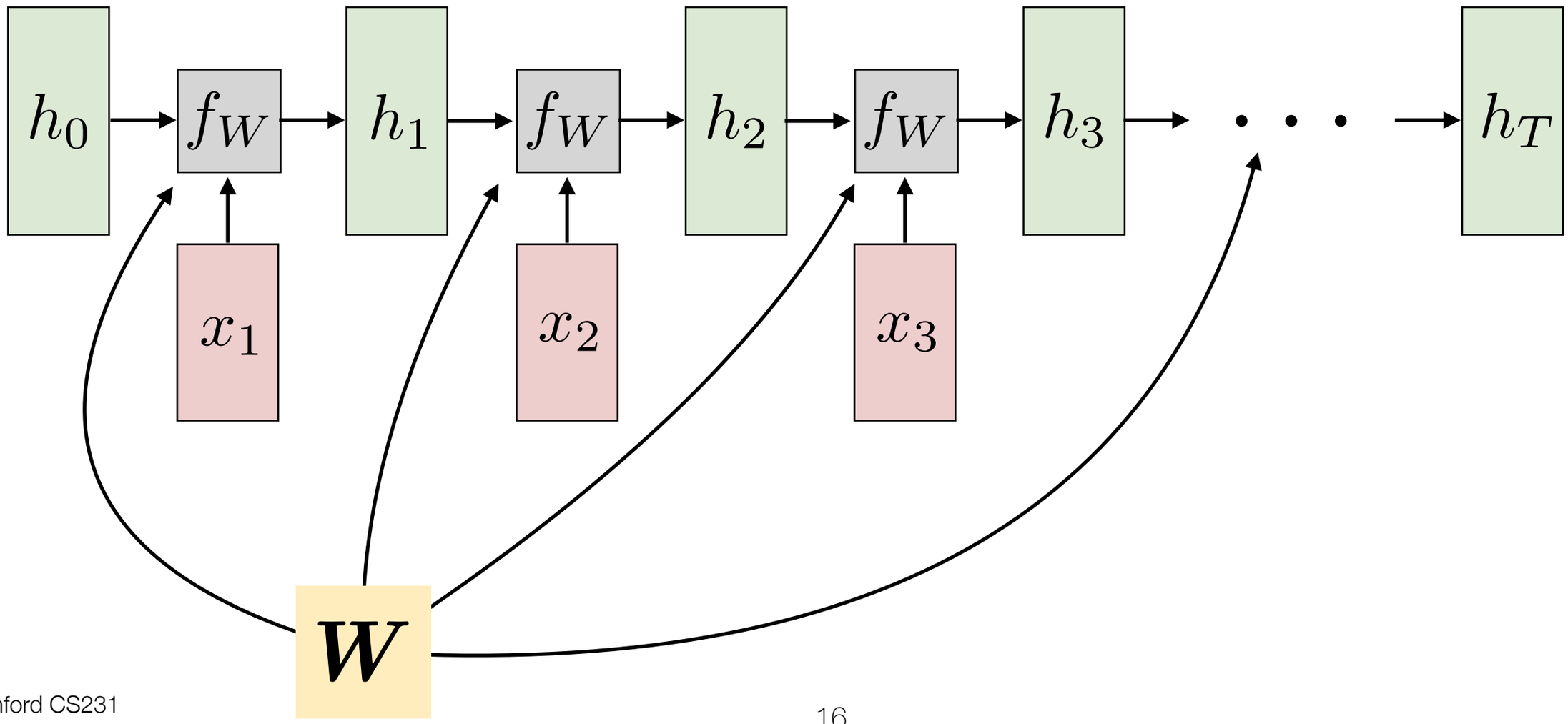
RNN computational graph (many to many)



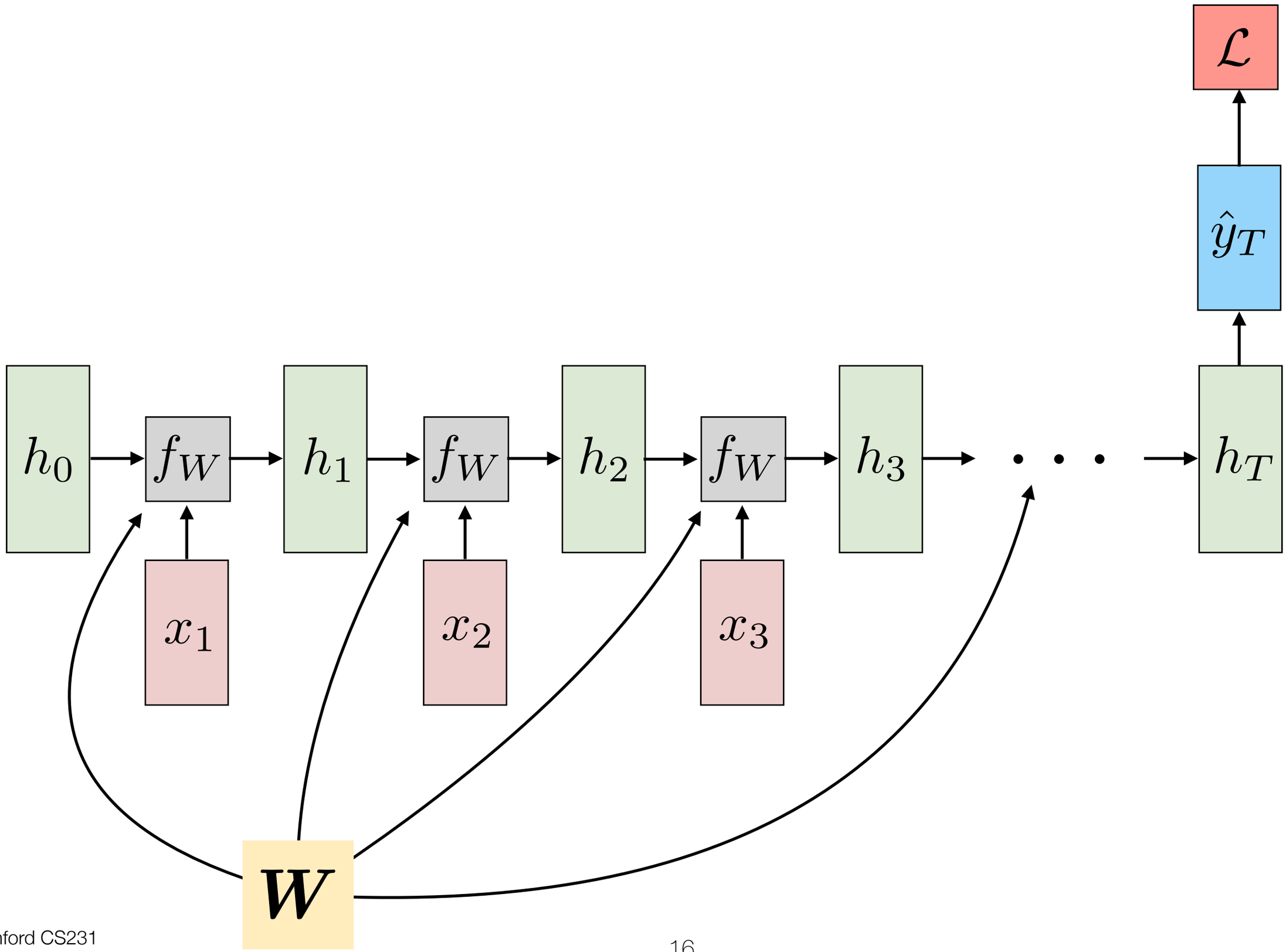
$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T}\mathcal{L}_t$$

RNN computational graph (many to one)

RNN computational graph (many to one)

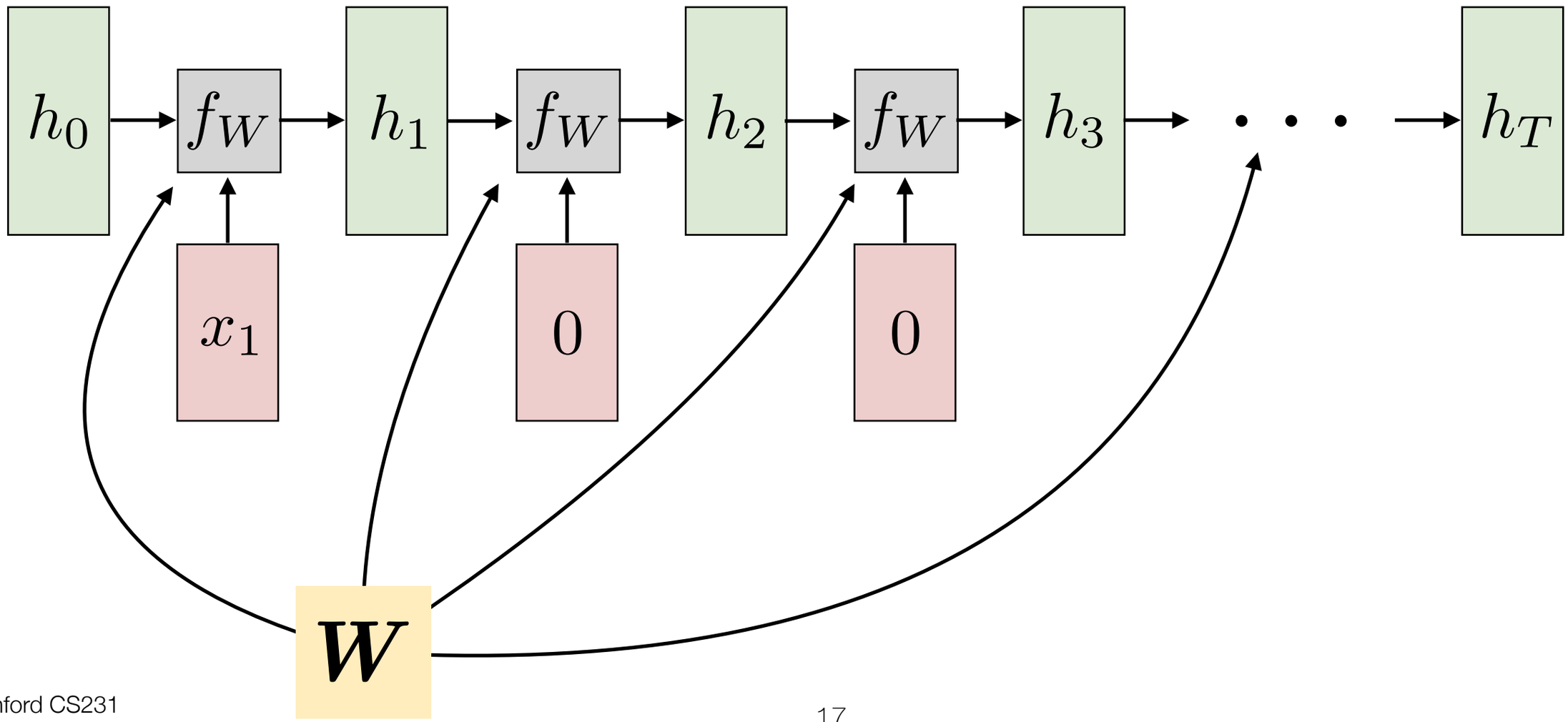


RNN computational graph (many to one)

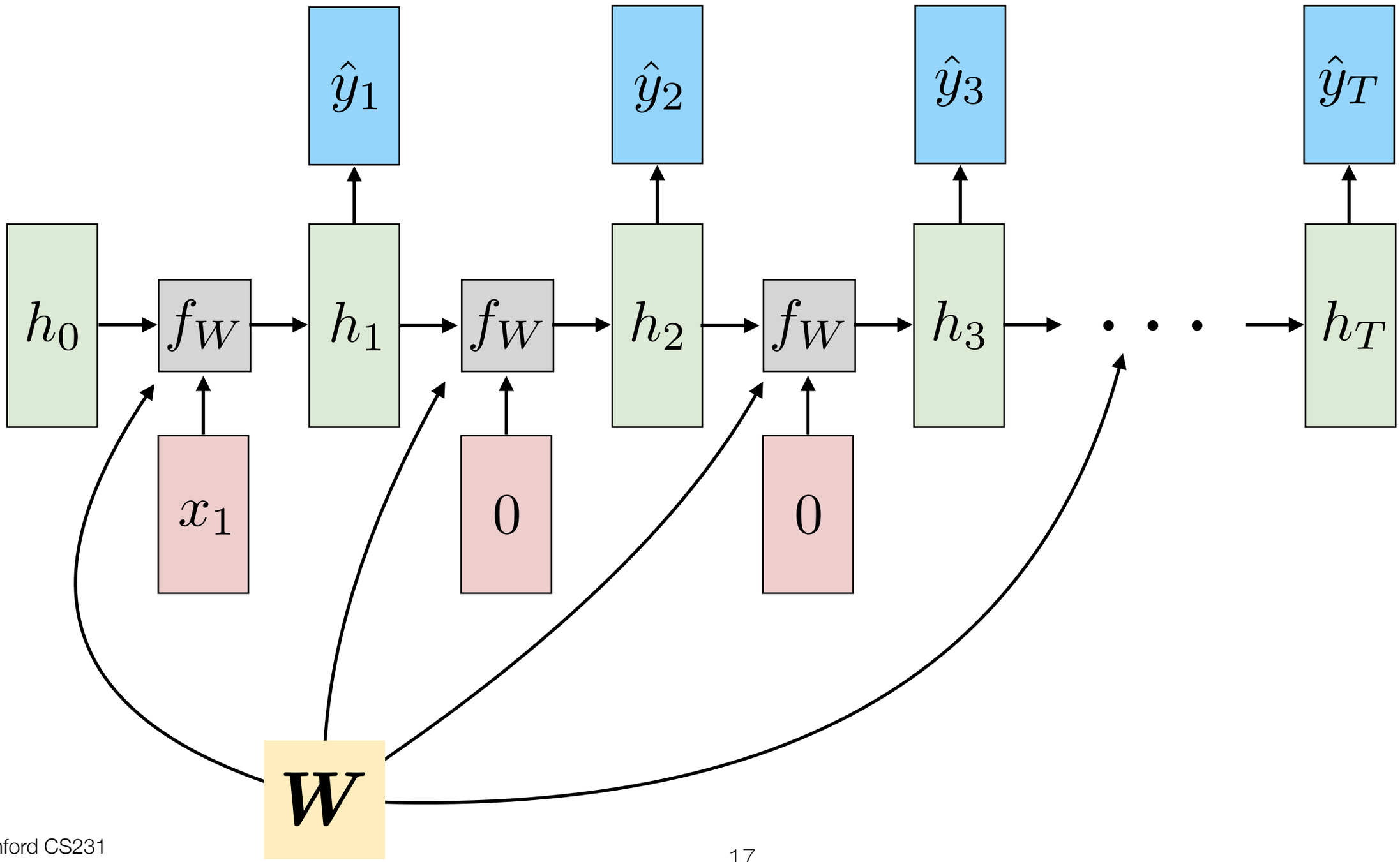


RNN computational graph (one to many)

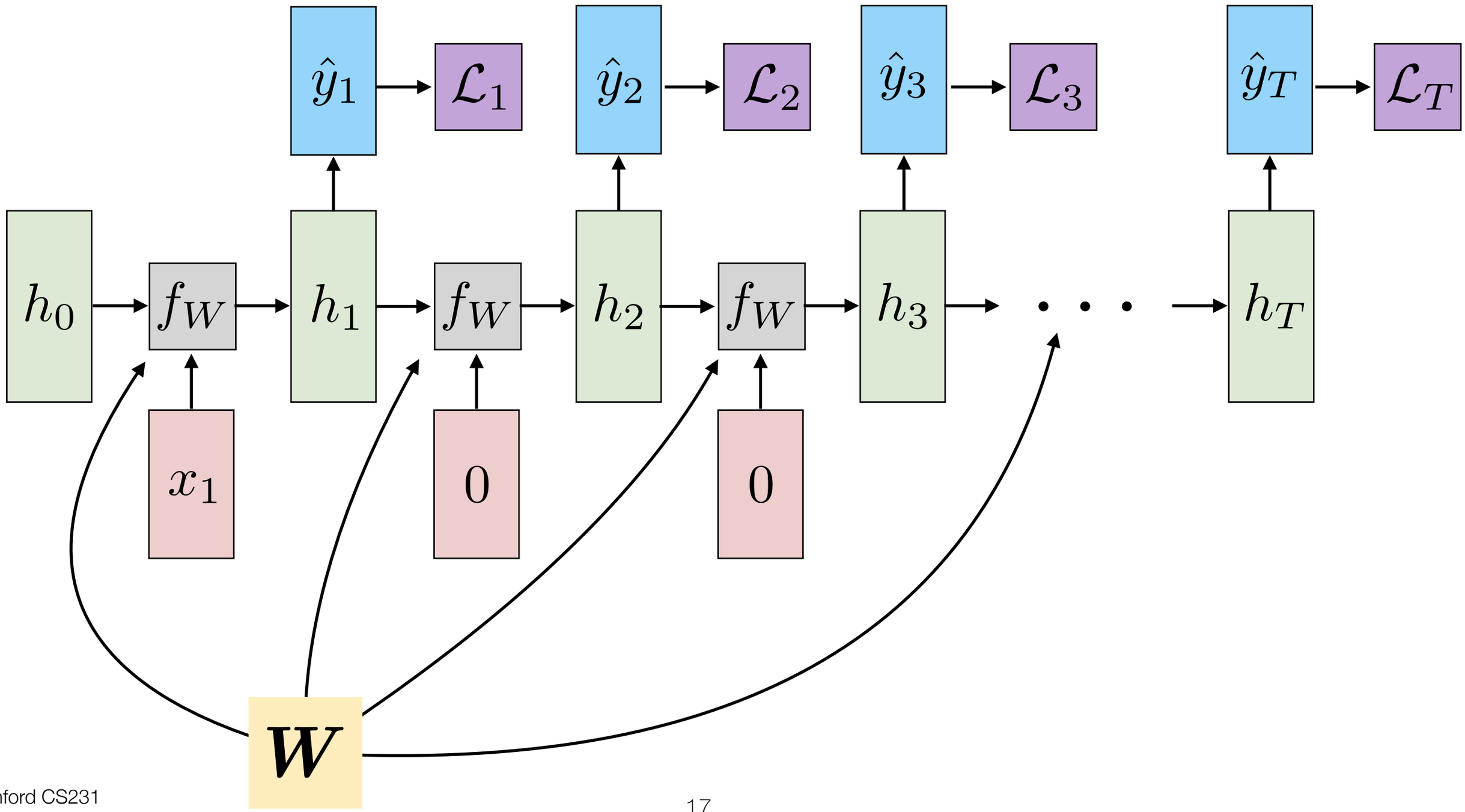
RNN computational graph (one to many)



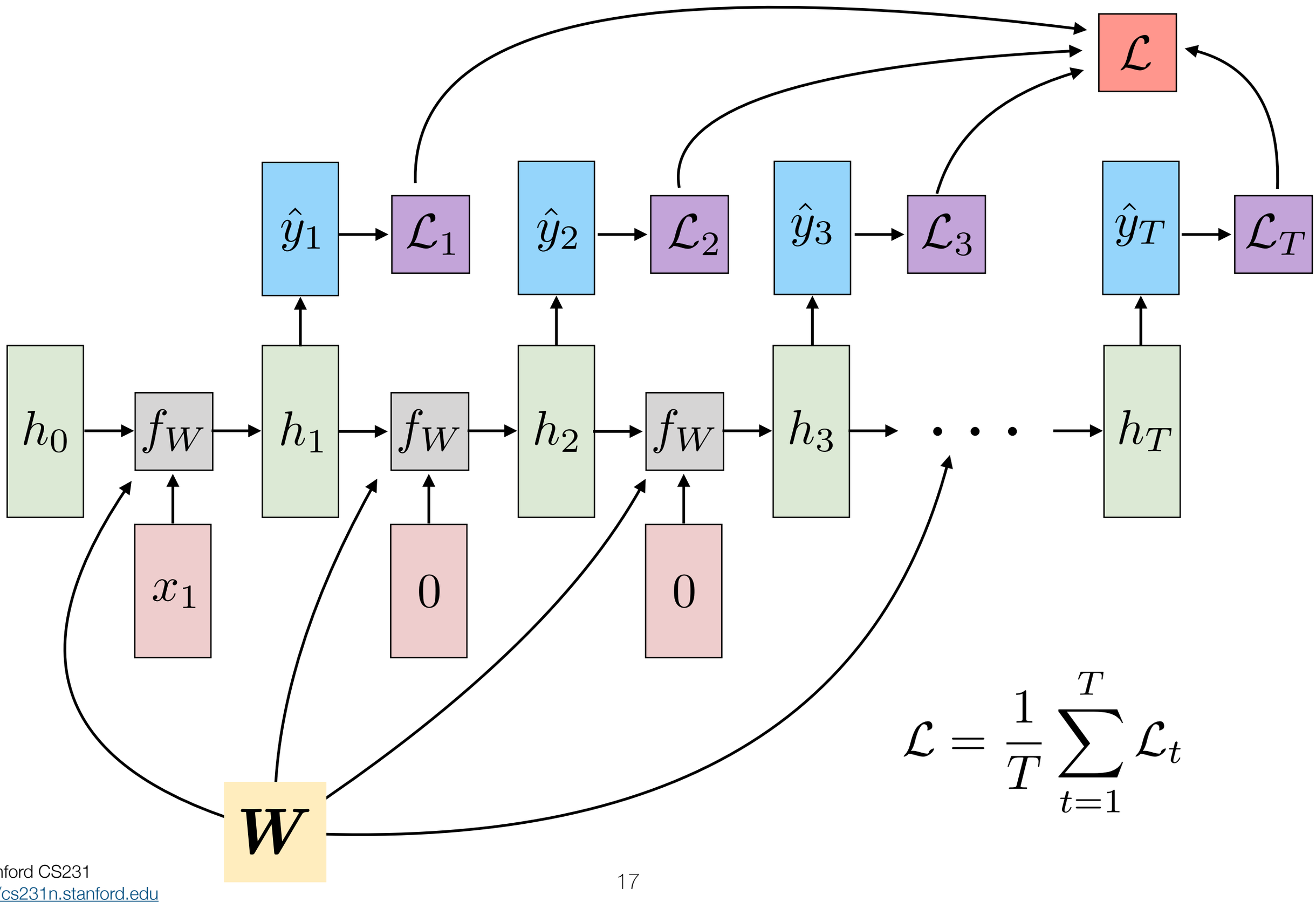
RNN computational graph (one to many)



RNN computational graph (one to many)

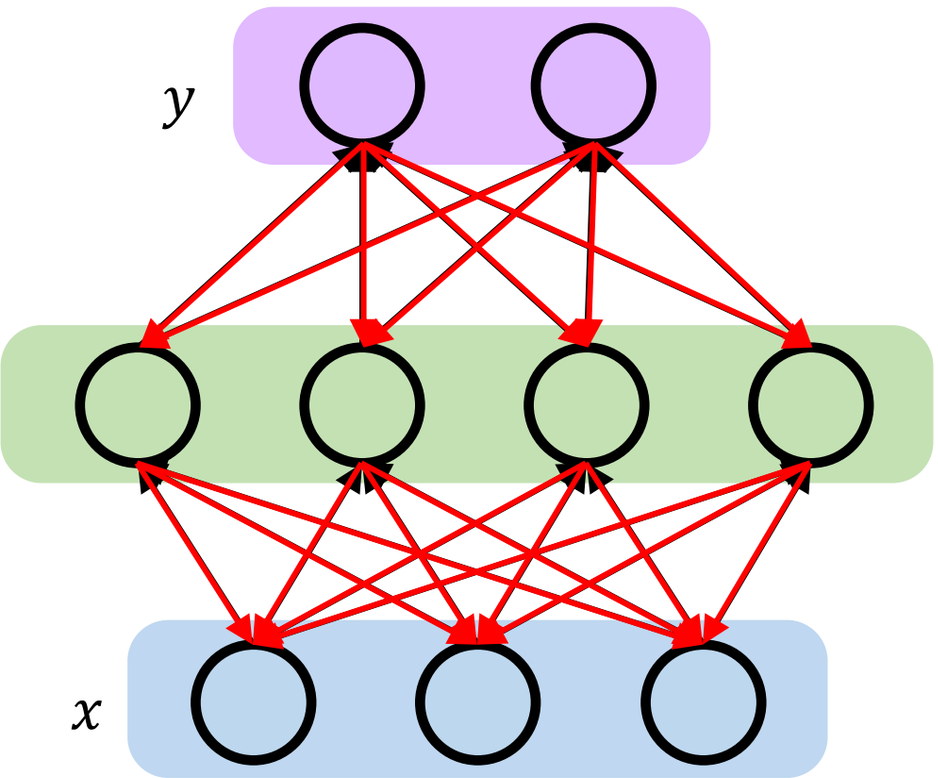


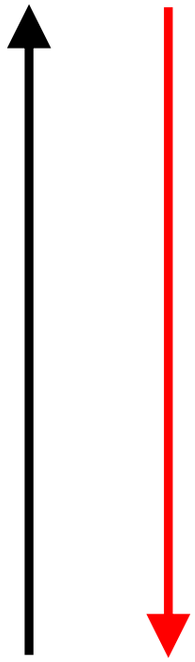
RNN computational graph (one to many)



Backpropagation through time

Recall: backpropagation in dense NNs



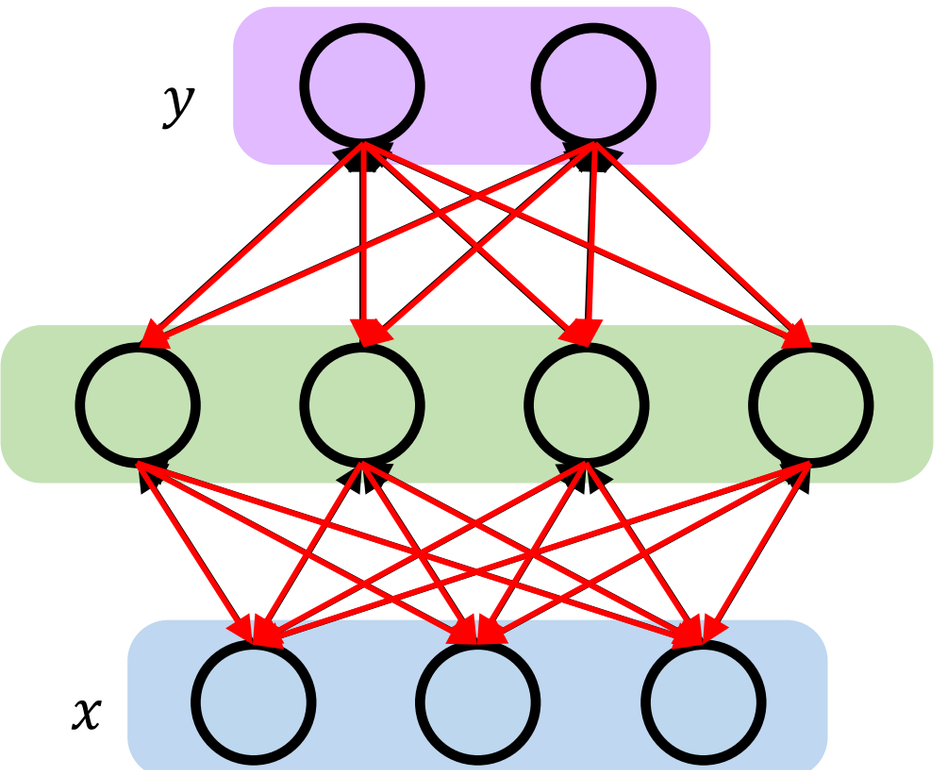


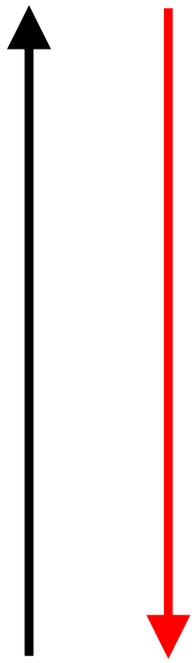
Backpropagation algorithm:

1. Take the derivative (gradient) of the loss with respect to each parameter

2. Shift parameters in order to minimize loss

Recall: backpropagation in dense NNs





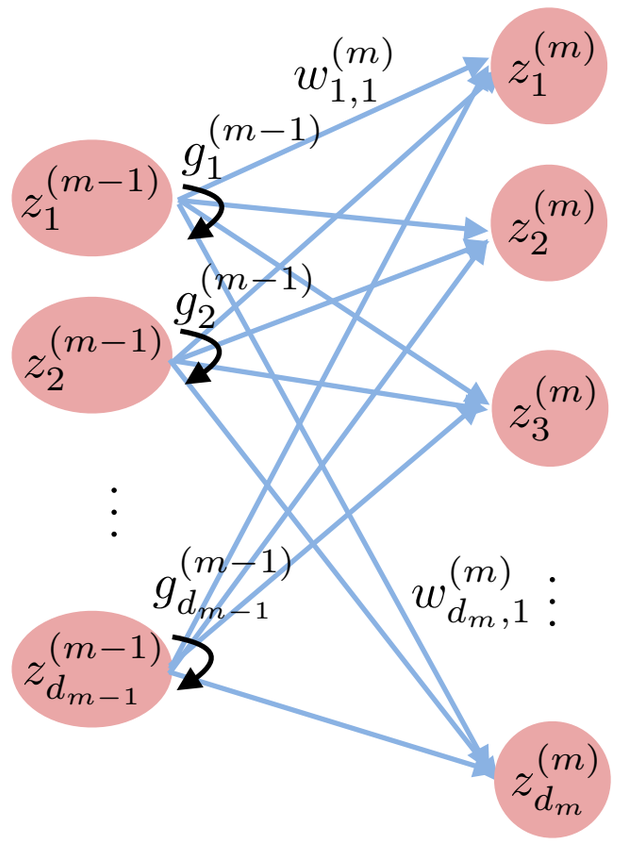
Backpropagation algorithm:

1. Take the derivative (gradient) of the loss with respect to each parameter

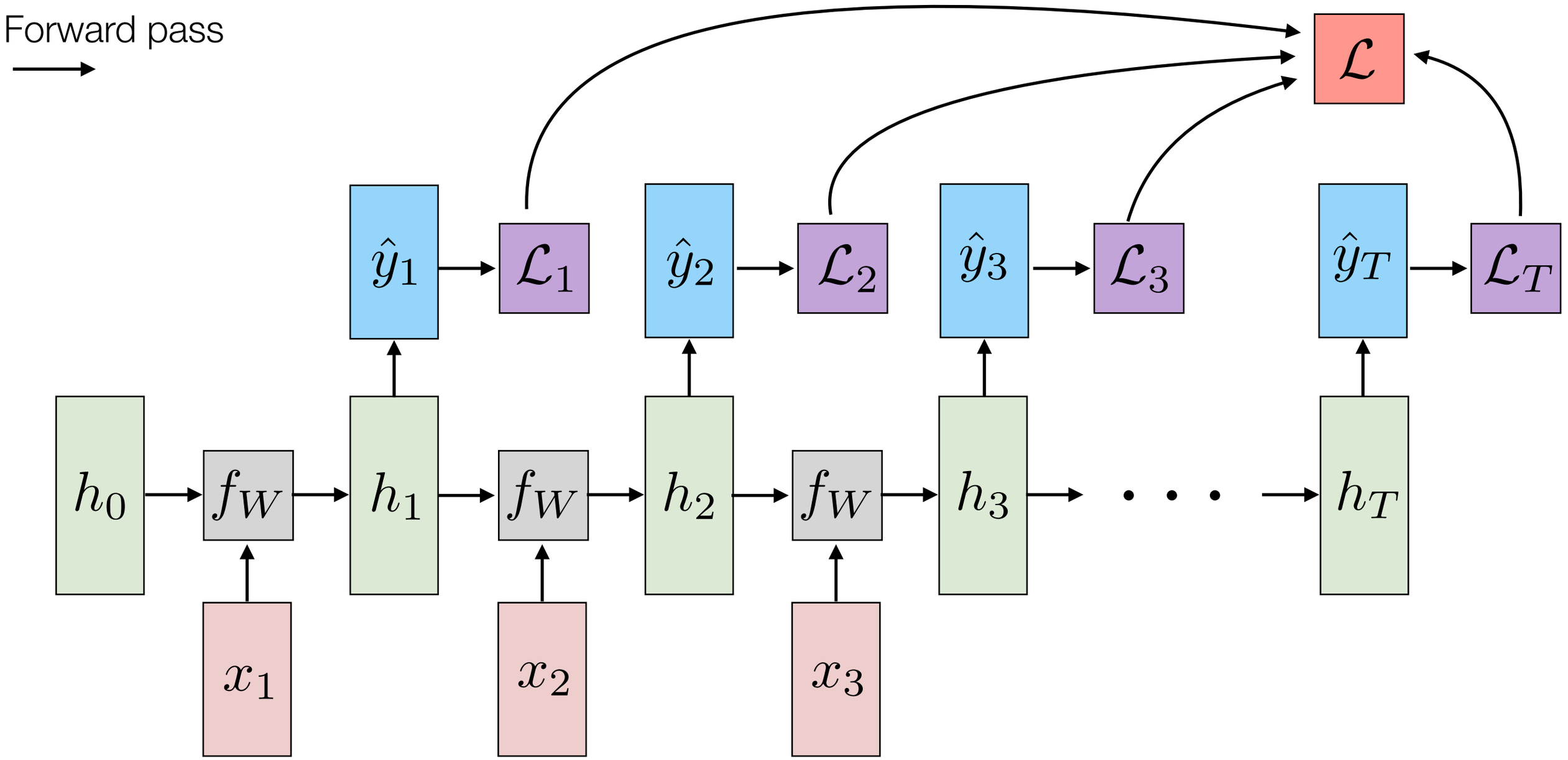
2. Shift parameters in order to minimize loss

$$J(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}\left(\hat{\mathbf{y}}(\mathbf{x}_n), \mathbf{y}_n\right) = \sum_{n=1}^{N}\mathcal{L}_n$$

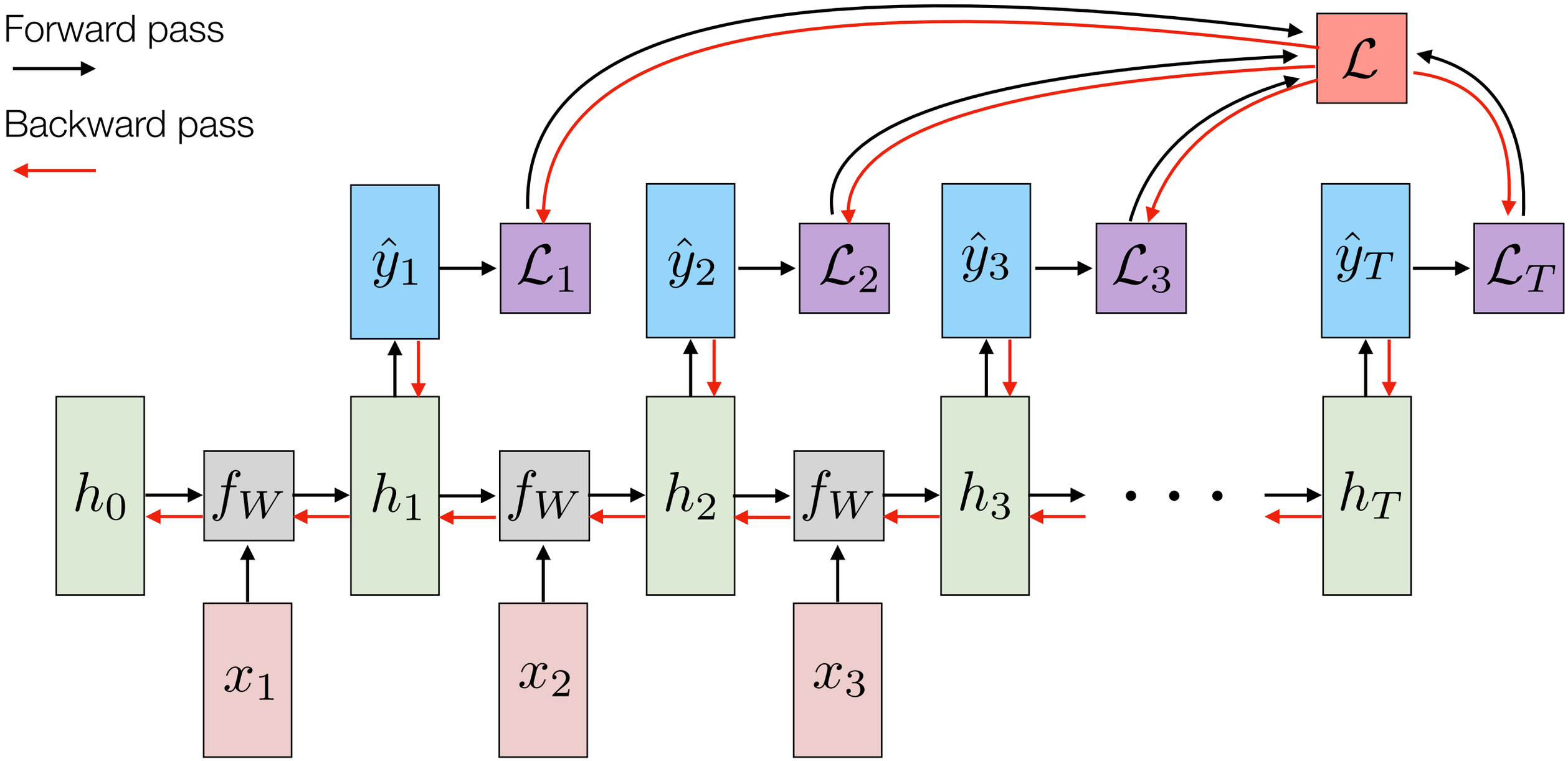
$$\frac{\partial \mathcal{L}_n}{\partial w_{ji}^{(m)}} = \delta_j^{(m)} g_i^{(m-1)} \qquad \delta_j^{(m-1)} = g'(z_j^{(m-1)}) \sum_{k=1}^{d_m} w_{k,j}\delta_k^{(m)}$$



Backpropagation through time



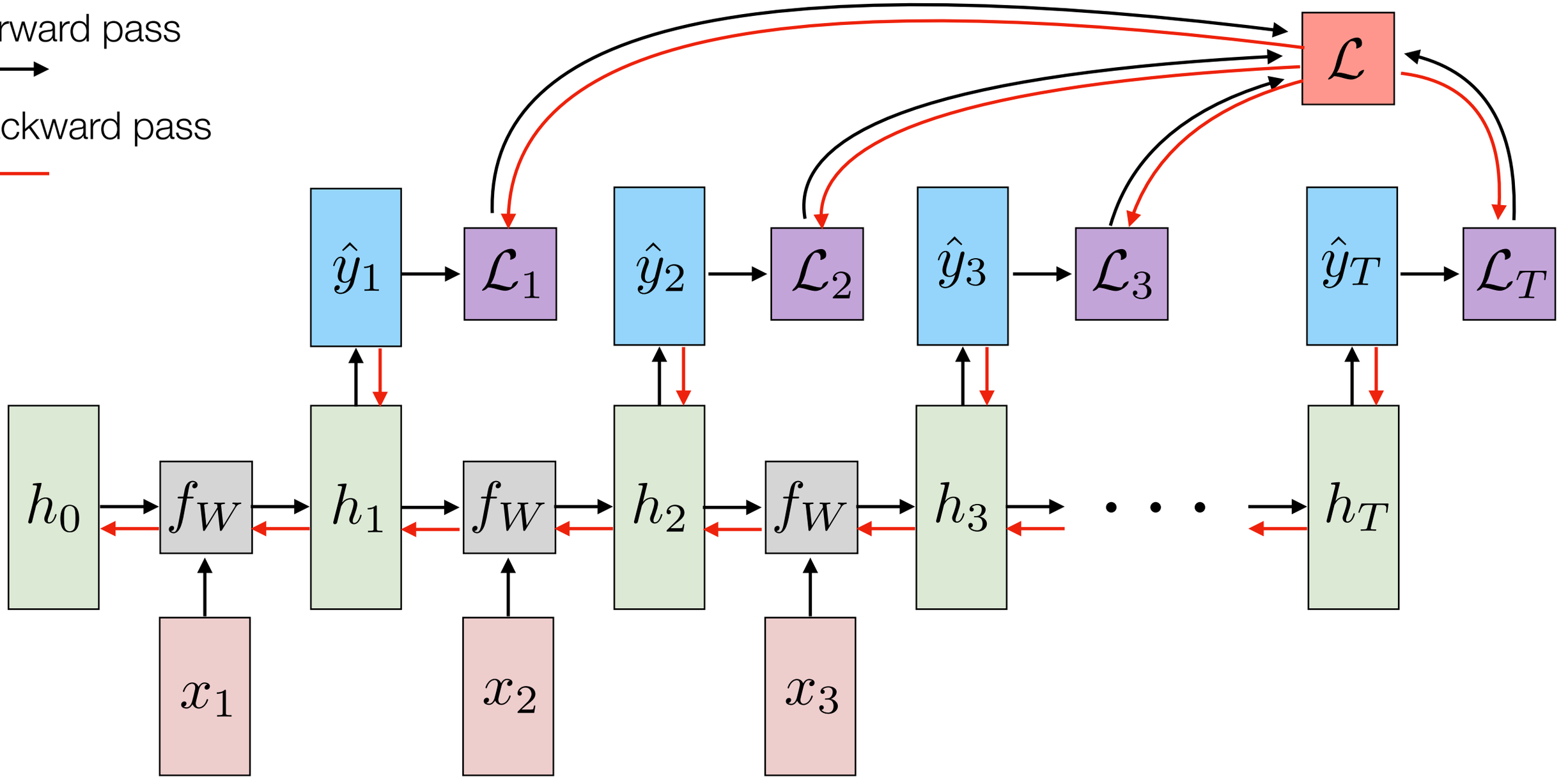
Backpropagation through time



Backpropagation through time

Forward pass

Backward pass



Backward pass -> Chain Rule -> Number of factors of the order of chain length!

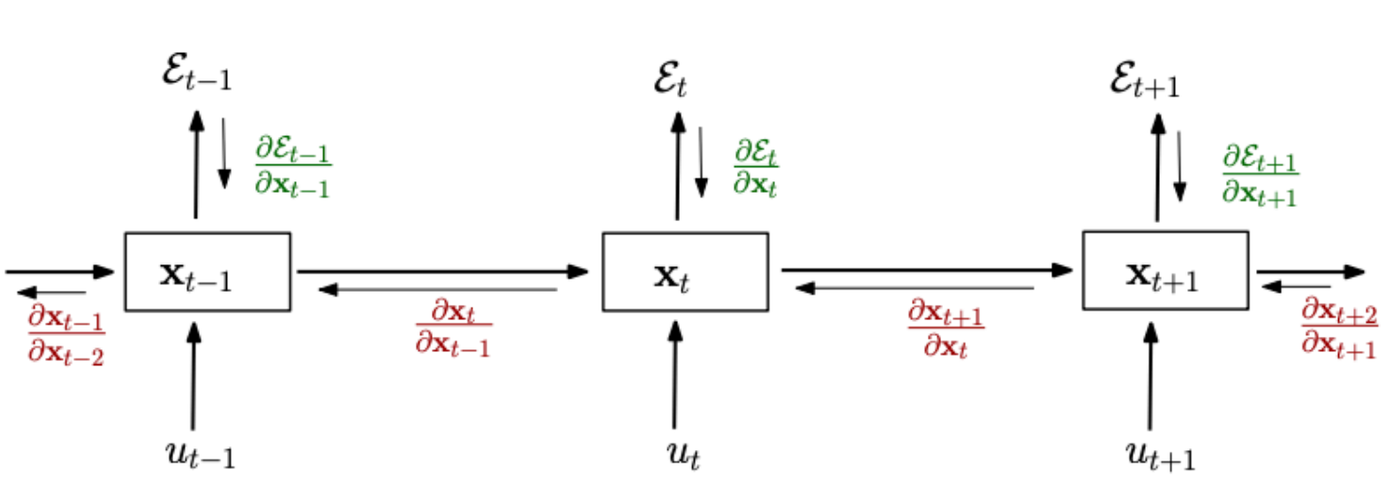
Backpropagation through time

On the difficulty of training Recurrent Neural Networks

Razvan Pascanu PASCANUR@IRO.UMONTREAL.CA
Universite de Montreal

Tomas Mikolov T.MIKOLOV@GMAIL.COM
Brno University

Yoshua Bengio YOSHUA.BENGIO@UMONTREAL.CA
Universite de Montreal



$$\mathbf{x}_t = F(\mathbf{x}_{t-1}, \mathbf{u}_t, \theta) \tag{1}$$

$$\mathbf{x}_t = \mathbf{W}_{rec}\sigma(\mathbf{x}_{t-1}) + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b} \tag{2}$$

$$\mathcal{E}_t = \mathcal{L}(\mathbf{x}_t).$$

$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial \mathcal{E}_t}{\partial \theta} \tag{3}$$

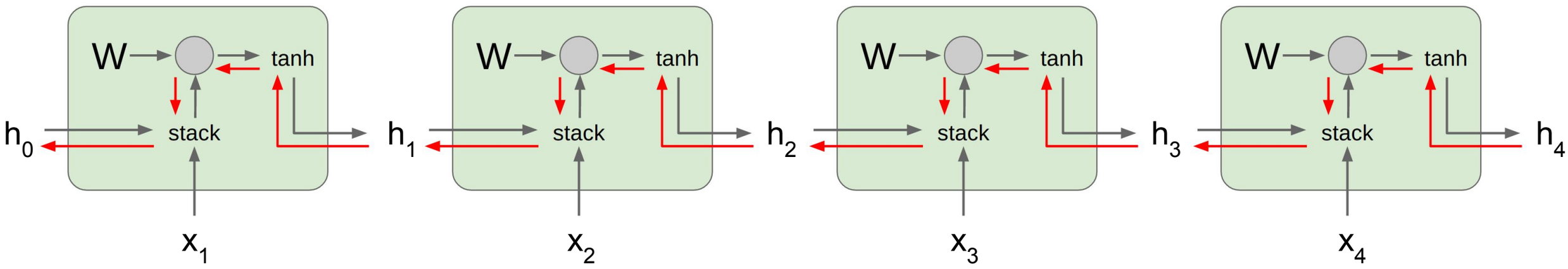
$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \leq k \leq t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right) \tag{4}$$

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \geq i > k} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1})) \tag{5}$$

Standard RNN gradient flow: exploiting gradients

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

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

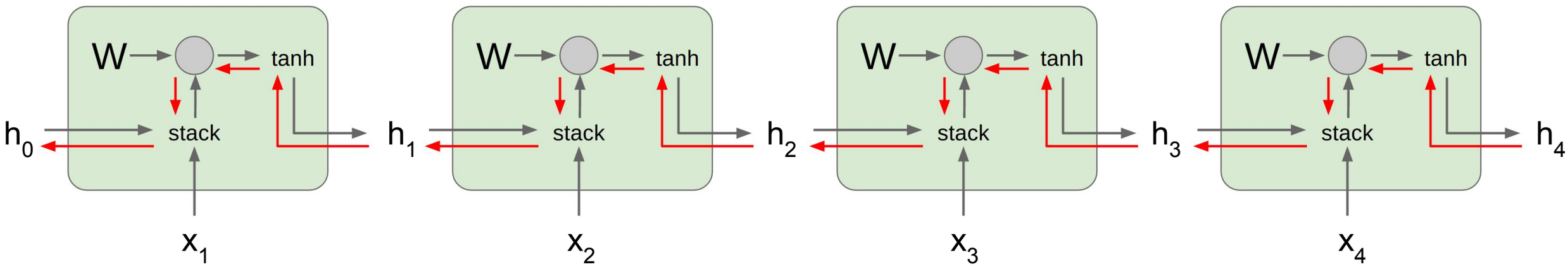


Computing the gradient wrt h_0 involves many factors of $\boldsymbol{W}$ (and repeated f'_W !)

Standard RNN gradient flow: exploiting gradients

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

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



Computing the gradient wrt h_0 involves many factors of $\boldsymbol{W}$ (and repeated f'_W !)

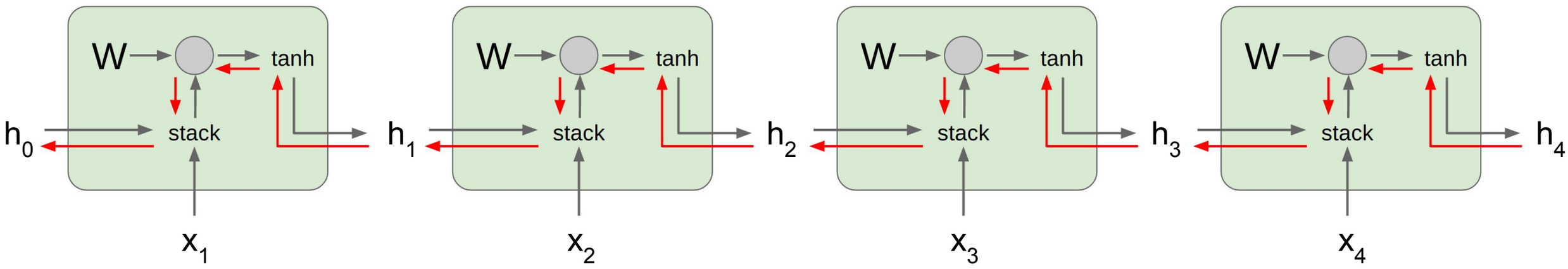
Many values > 1: exploding gradients

Gradient clipping to scale big gradients

Standard RNN gradient flow: exploiting gradients

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

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



Computing the gradient wrt h_0 involves many factors of $\boldsymbol{W}$ (and repeated f'_W !)

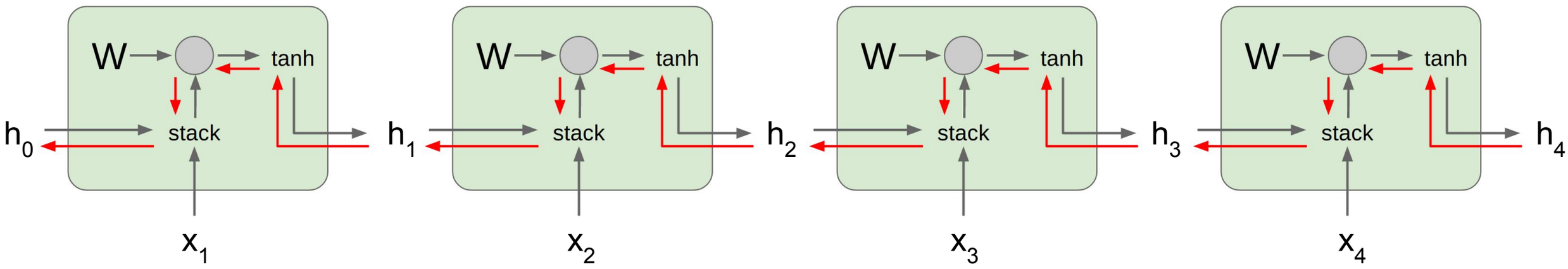
Largest singular value > 1: exploding gradients

Gradient clipping to scale big gradients

Standard RNN gradient flow: vanishing gradients

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

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

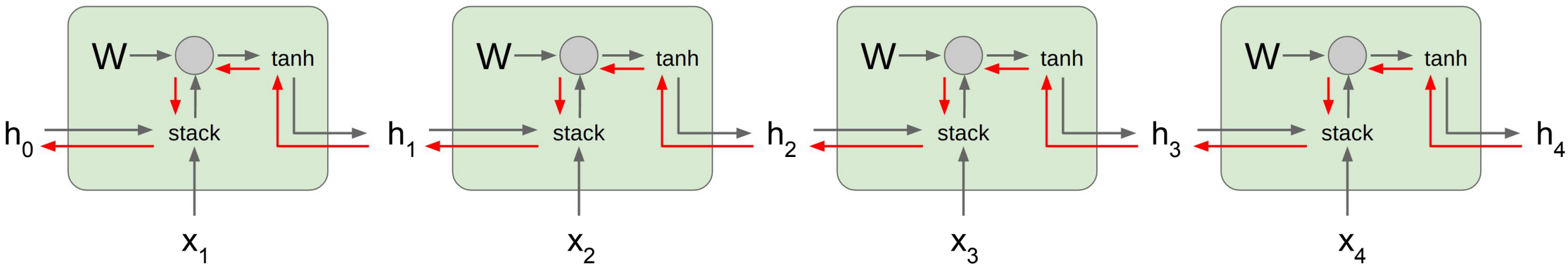


Computing the gradient wrt h_0 involves many factors of $\boldsymbol{W}$ (and repeated f'_W !)

Standard RNN gradient flow: vanishing gradients

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

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



Computing the gradient wrt h_0 involves many factors of $\boldsymbol{W}$ (and repeated f'_W !)

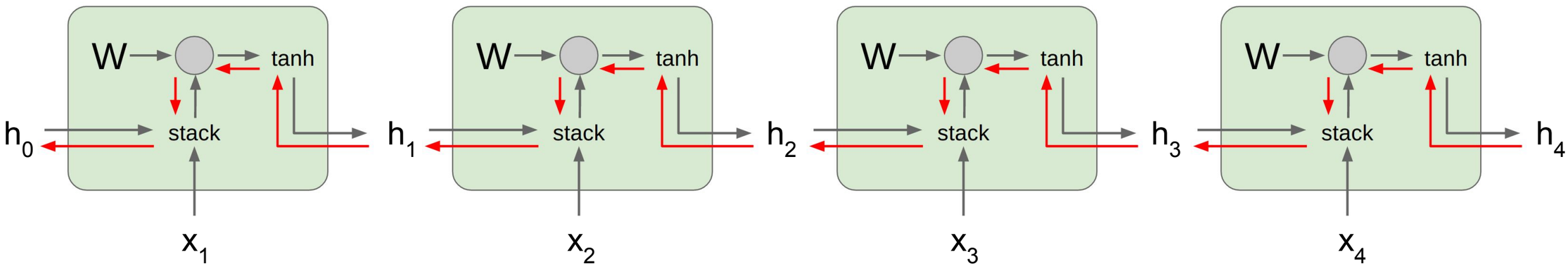
Many values < 1: vanishing gradients

Activation Function, Weight Initializaiton, Network architecture, …

Standard RNN gradient flow: vanishing gradients

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

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



Computing the gradient wrt h_0 involves many factors of $\boldsymbol{W}$ (and repeated f'_W !)

Larger singular value < 1: vanishing gradients

Activation Function, Weight Initializaiton, Network architecture, …

The problem of long-term dependencies

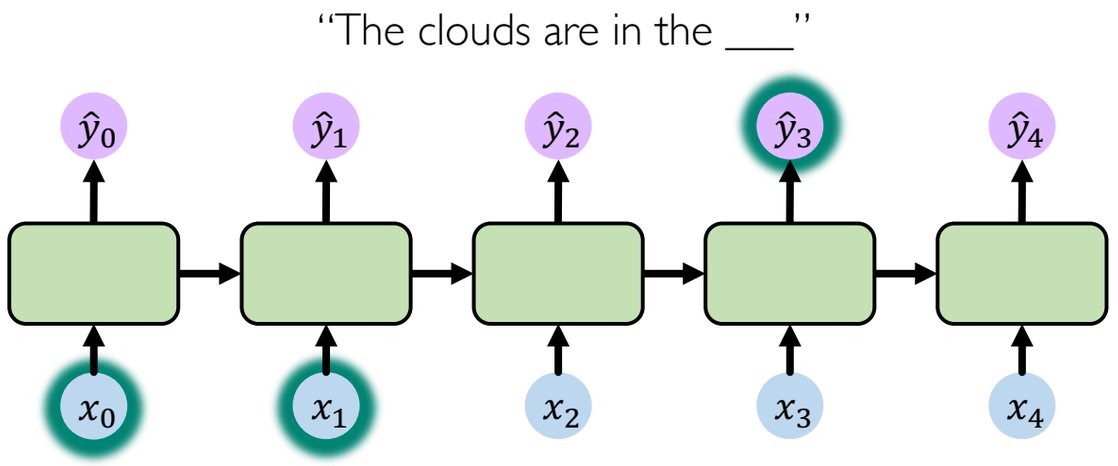
Multiply many **small numbers** together

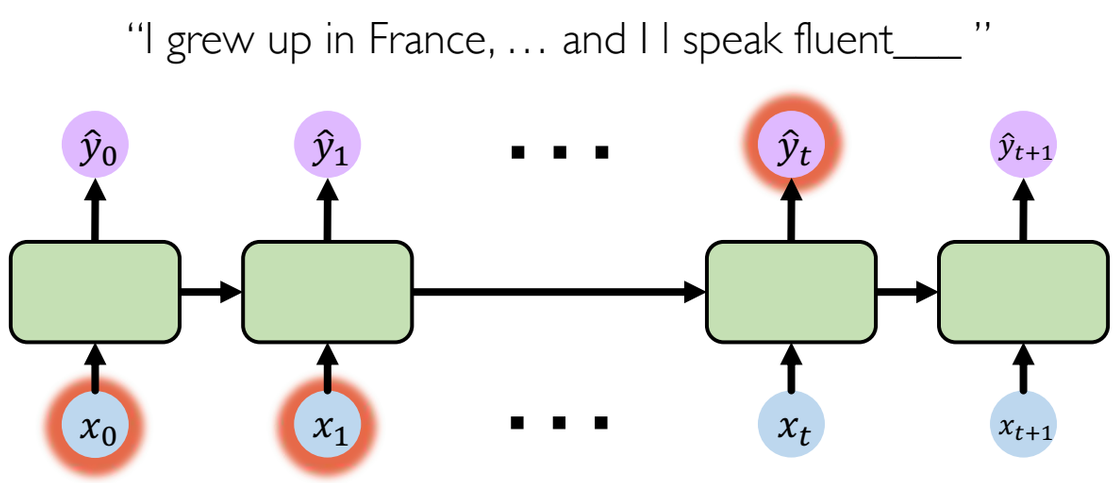
↓

Errors due to further back time steps have smaller and smaller gradients

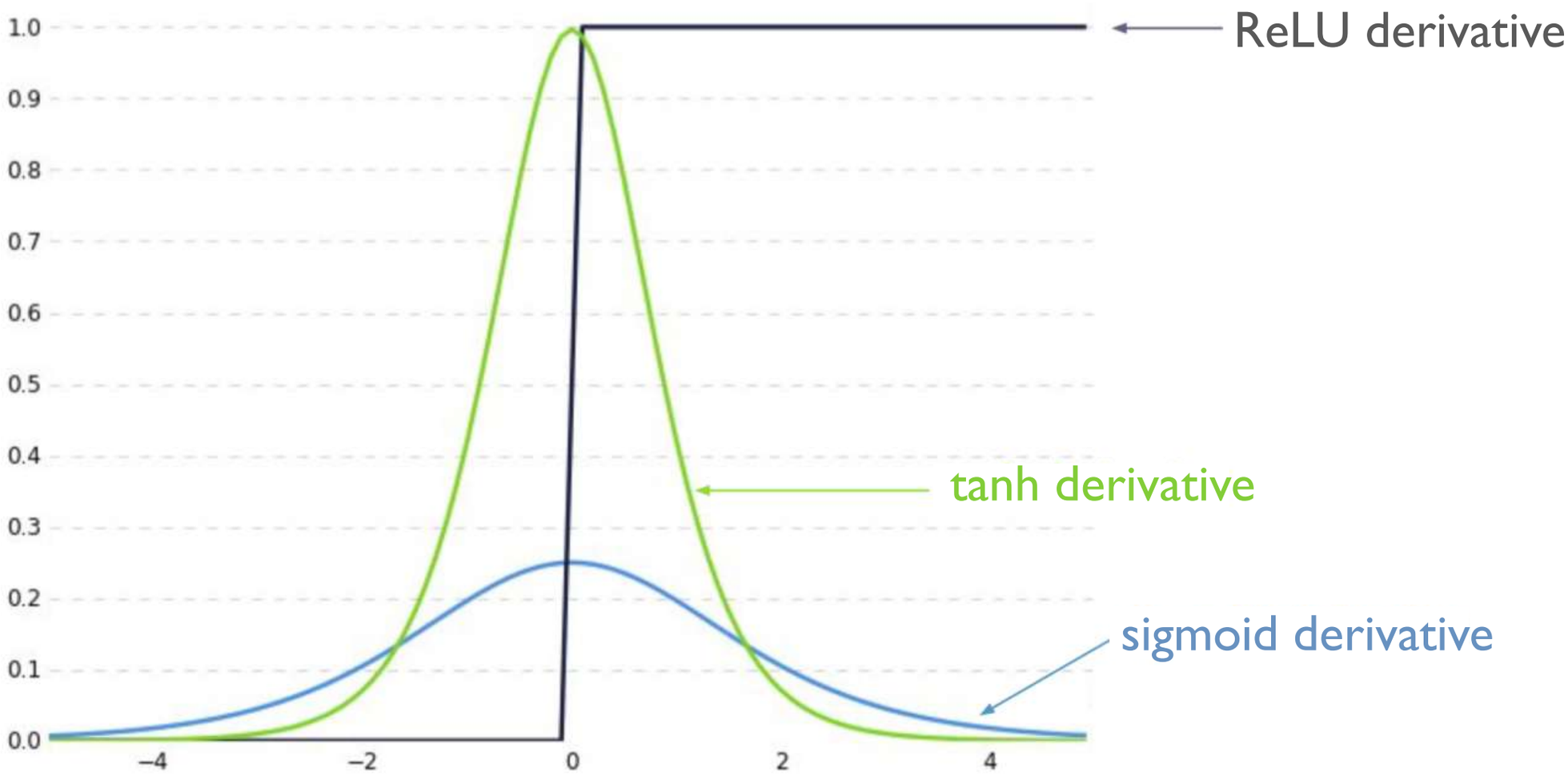
↓

Bias parameters to capture **short-term dependencies**





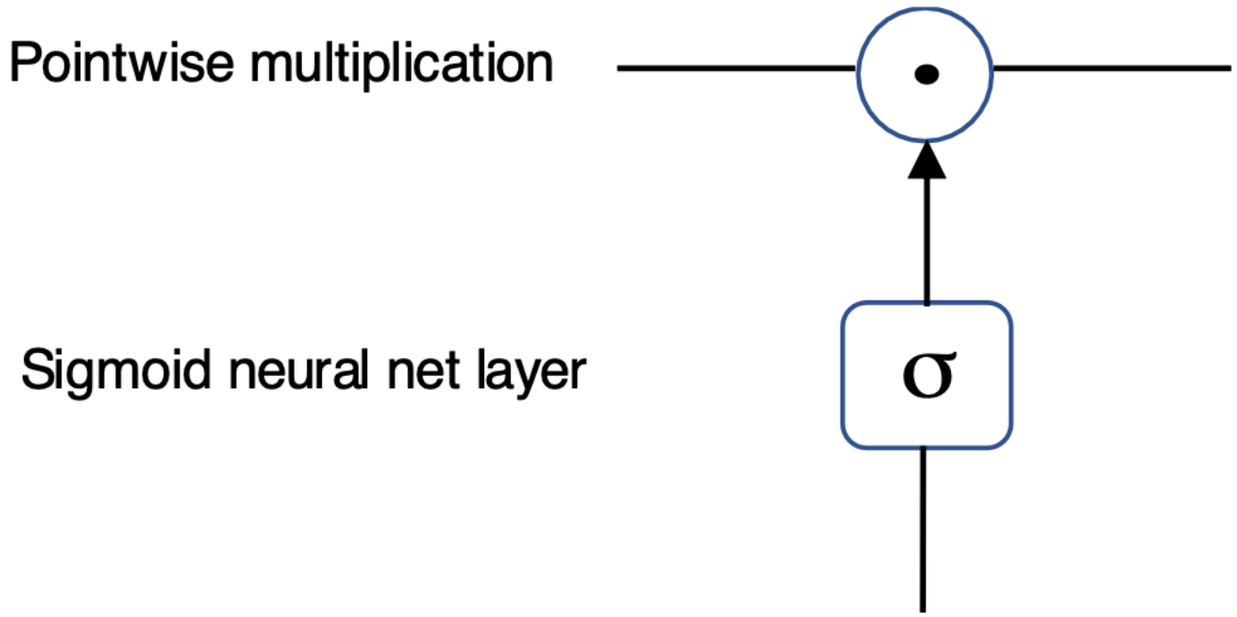
Trick #1: activation functions



Using ReLU prevents f'_W from shrinking the gradients when $x > 0$

Trick #2: gated cells

Idea: use a more **complex recurrent unit with gates** to control what information is passed through



gated cell

LSTM, GRU, etc.

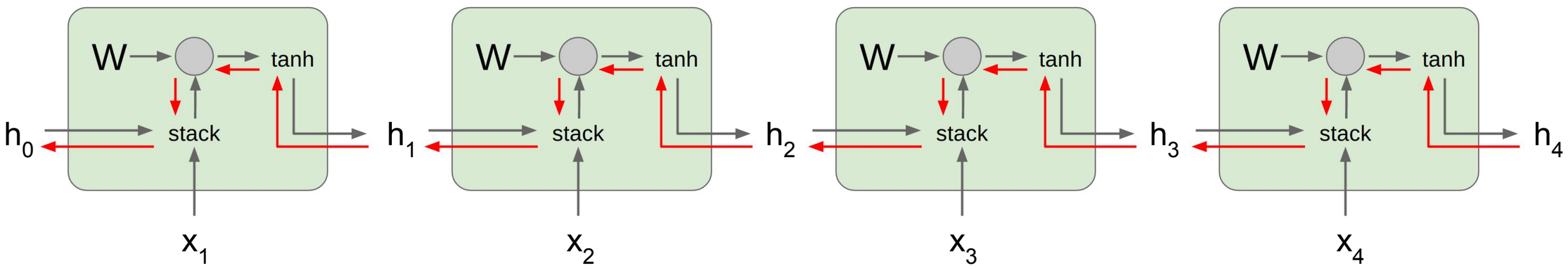
Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps

Long Short Term Memory (LSTMs)

Hochreiter and Schmidhuber, Neural Computation 1997

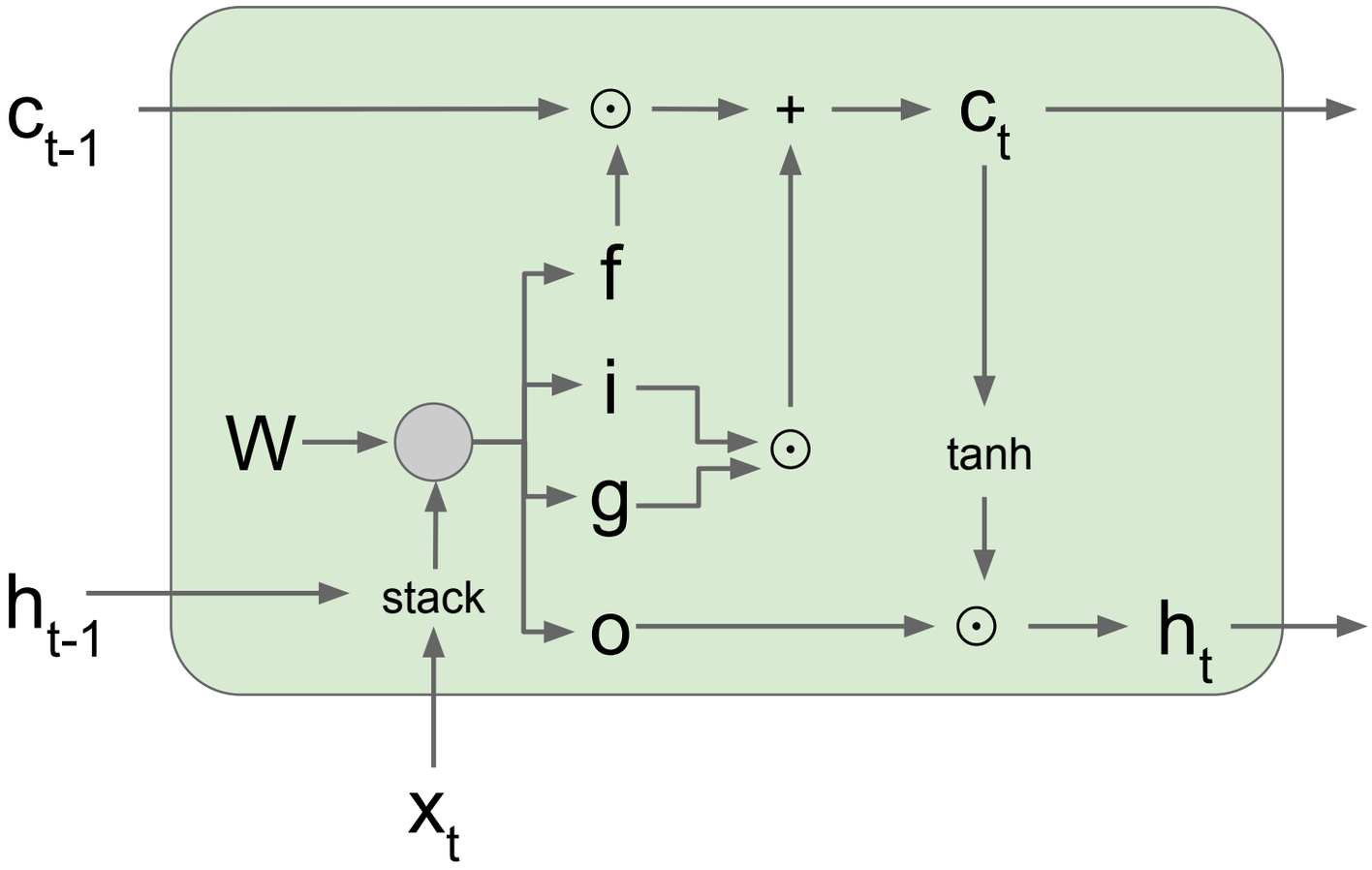
Basic RNN limitations

Only one operation mode: each new input modifies the state



The network doesn't have mechanisms to remember past events or ignore current observations.

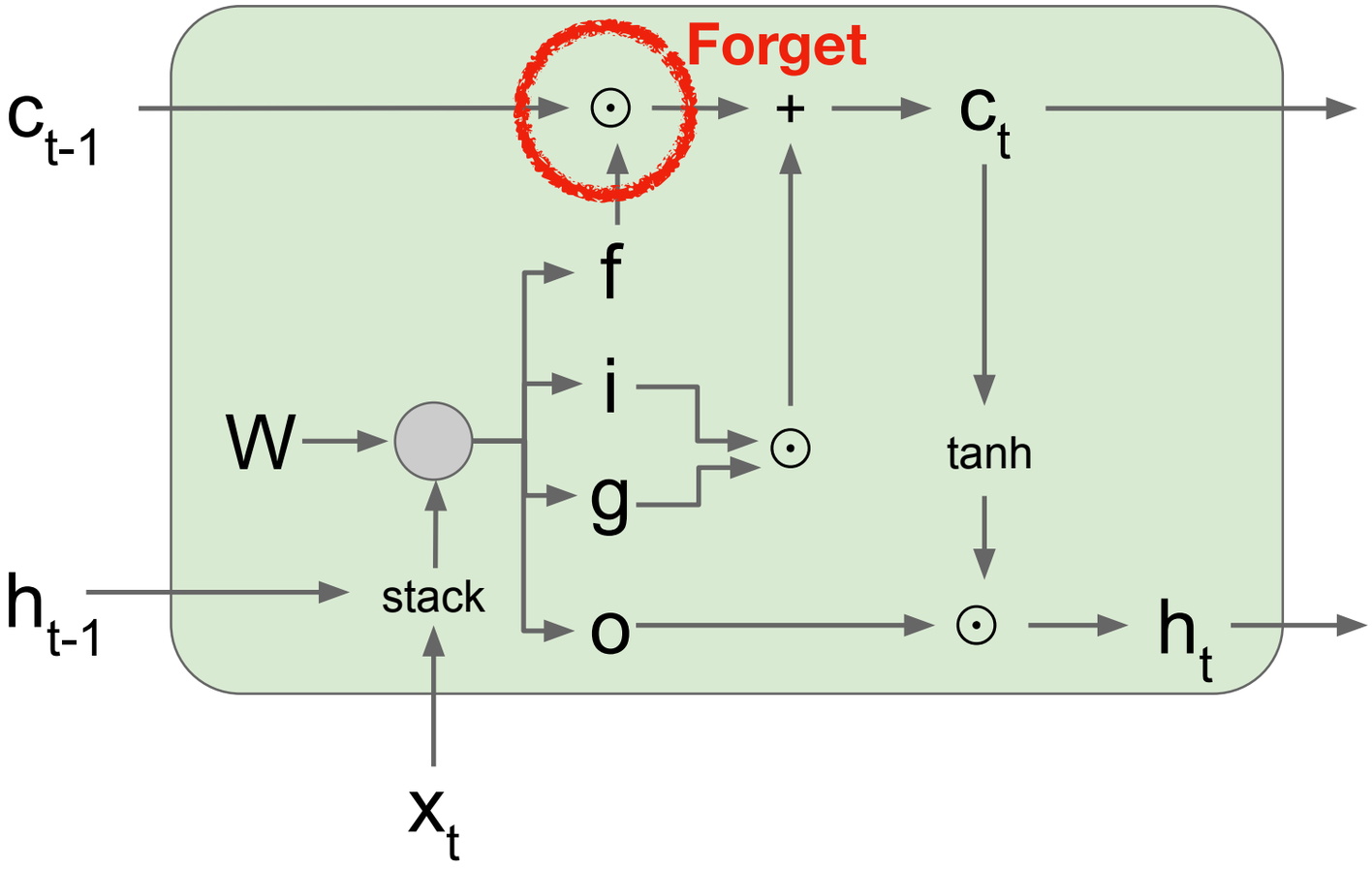
LSTMs: gating mechanism



$$f = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f)$$
$$i = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$o = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o)$$
$$g = \tanh(W_{cx}x_t + W_{ch}h_{t-1} + b_c)$$
$$c_t = f \circ c_{t-1} + i \circ g$$
$$h_t = o_t \circ \tanh(c_t)$$

Gates optionally let information through, via a **sigmoid neural** net layer and **pointwise multiplication**

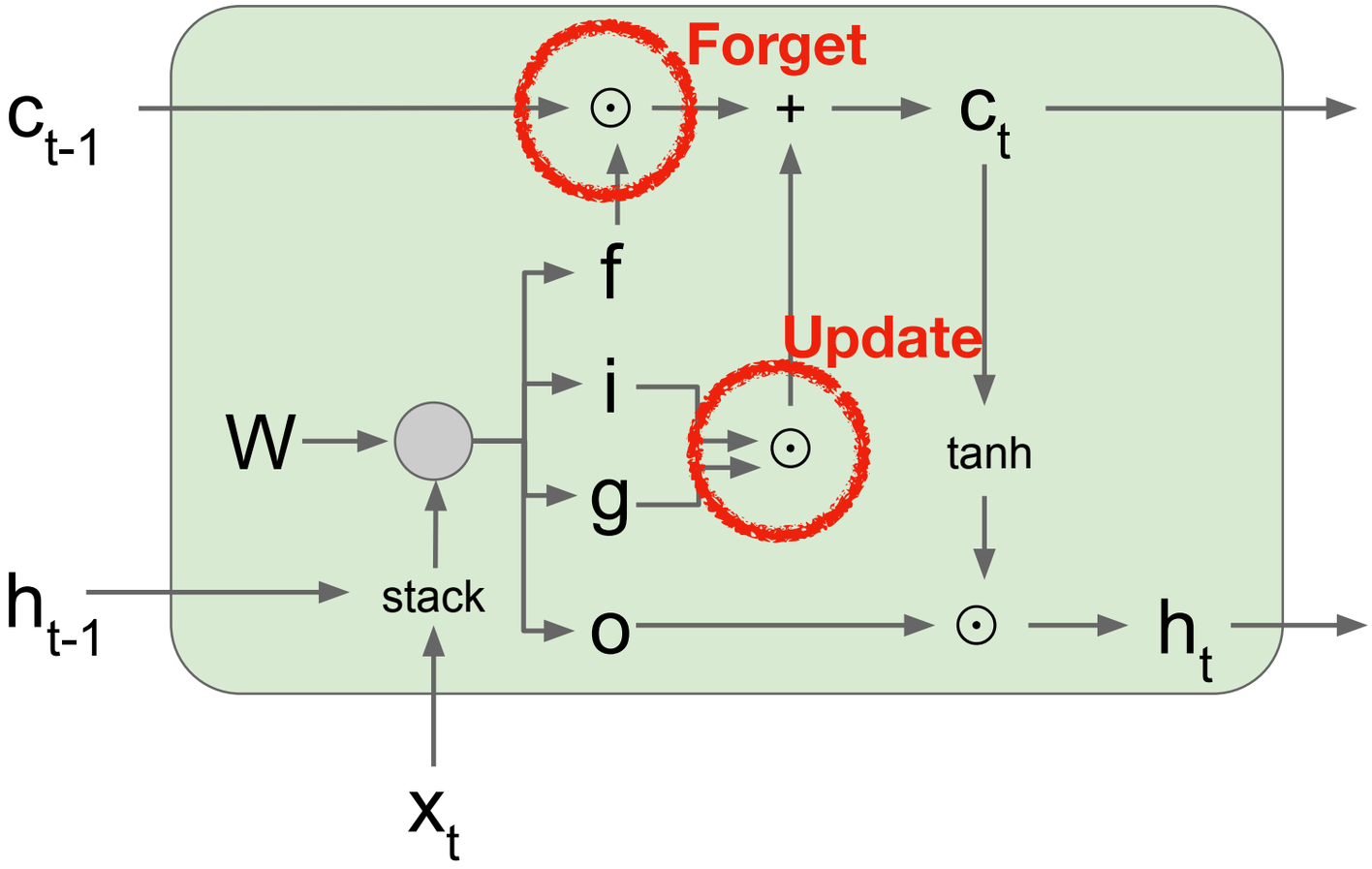
LSTMs: gating mechanism



$$f = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f)$$
$$i = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$o = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o)$$
$$g = \tanh(W_{cx}x_t + W_{ch}h_{t-1} + b_c)$$
$$c_t = f \circ c_{t-1} + i \circ g$$
$$h_t = o_t \circ \tanh(c_t)$$

Gates optionally let information through, via a **sigmoid neural** net layer and **pointwise multiplication**

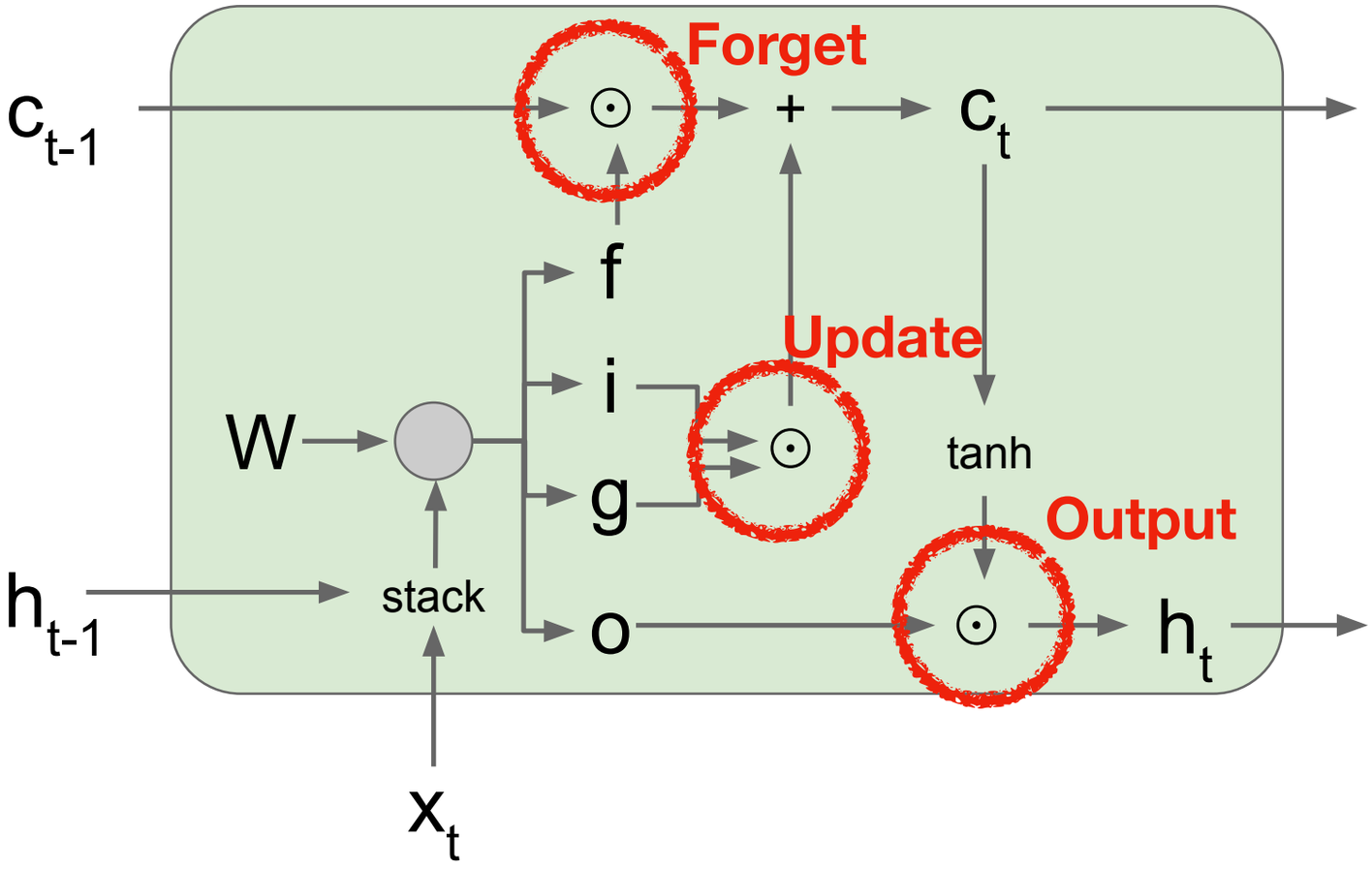
LSTMs: gating mechanism



$$
\begin{aligned}
f &= \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \\
i &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
o &= \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \\
g &= \tanh(W_{cx}x_t + W_{ch}h_{t-1} + b_c) \\
c_t &= f \circ c_{t-1} + i \circ g \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

Gates optionally let information through, via a **sigmoid neural** net layer and **pointwise multiplication**

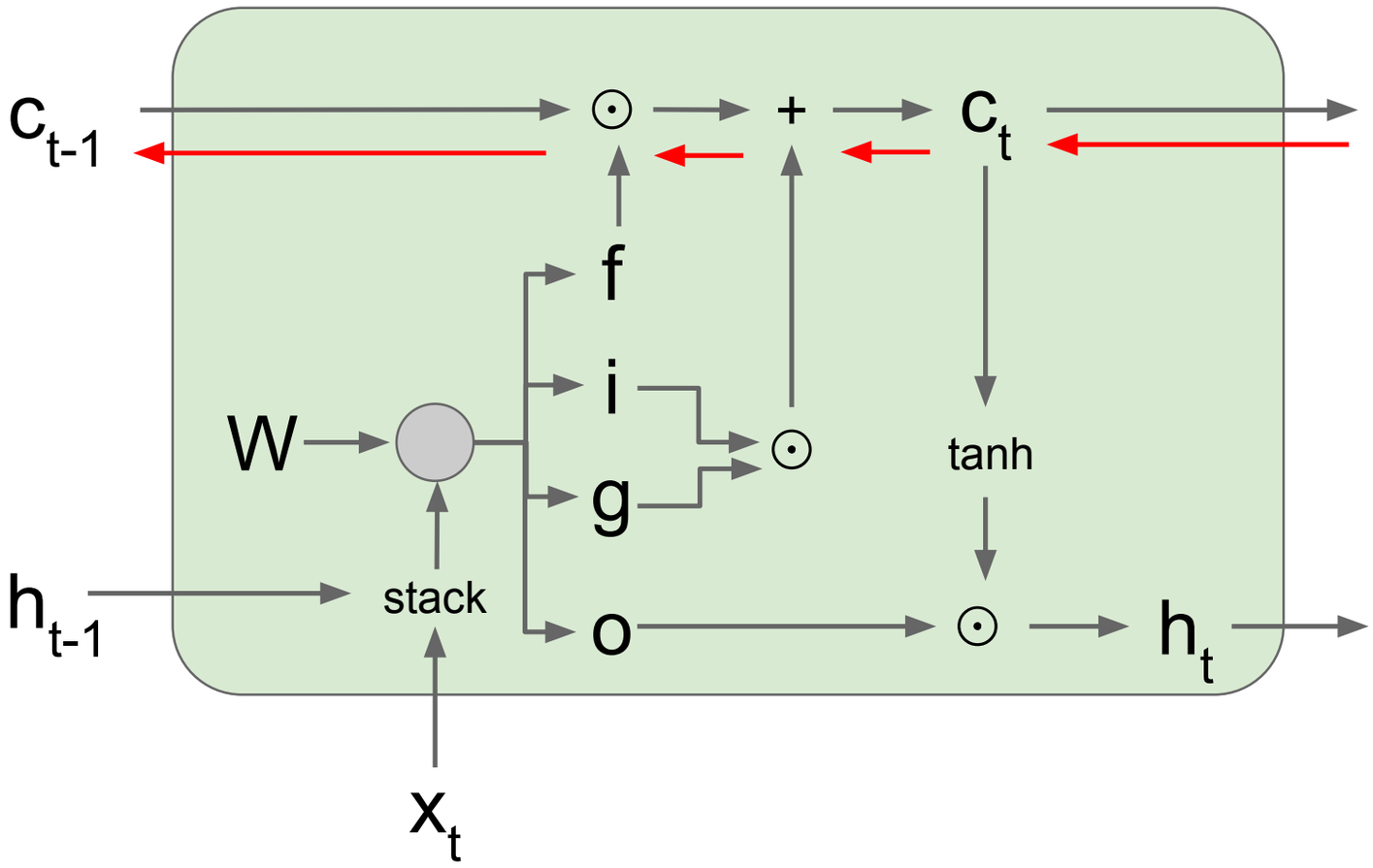
LSTMs: gating mechanism



$$
\begin{aligned}
f &= \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \\
i &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
o &= \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \\
g &= \tanh(W_{cx}x_t + W_{ch}h_{t-1} + b_c) \\
c_t &= f \circ c_{t-1} + i \circ g \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

Gates optionally let information through, via a **sigmoid neural** net layer and **pointwise multiplication**

LSTMs gradient flow



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

$$f = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f)$$
$$i = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$o = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o)$$
$$g = \tanh(W_{cx}x_t + W_{ch}h_{t-1} + b_c)$$
$$c_t = f \circ c_{t-1} + i \circ g$$
$$h_t = o_t \circ \tanh(c_t)$$

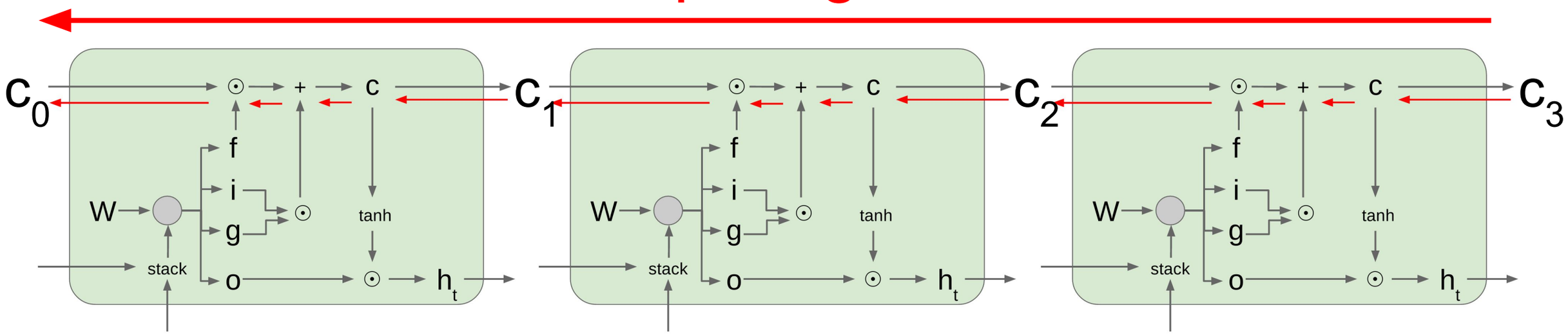
Check this link: How LSTM networks solve the problem of vanishing gradients

LSTMs gradient flow

Check this link: How LSTM networks solve the problem of vanishing gradients

LSTMs gradient flow

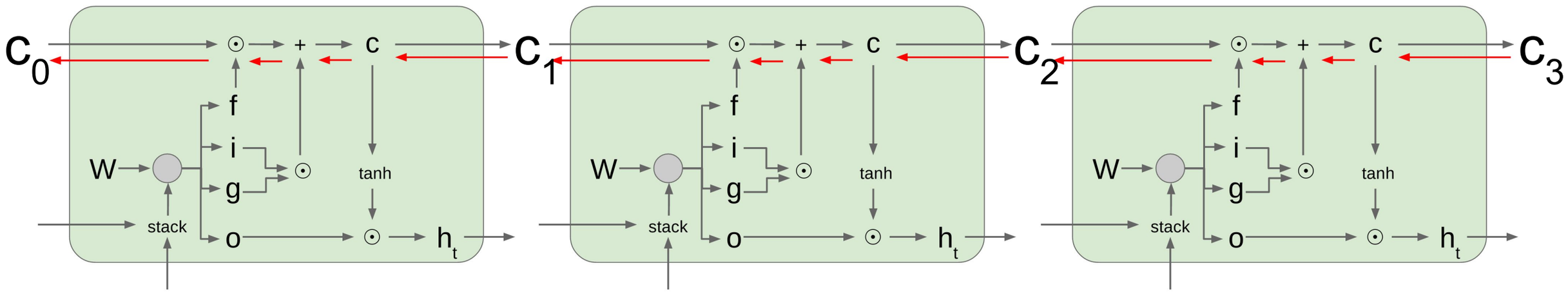
Uninterrupted gradient flow!



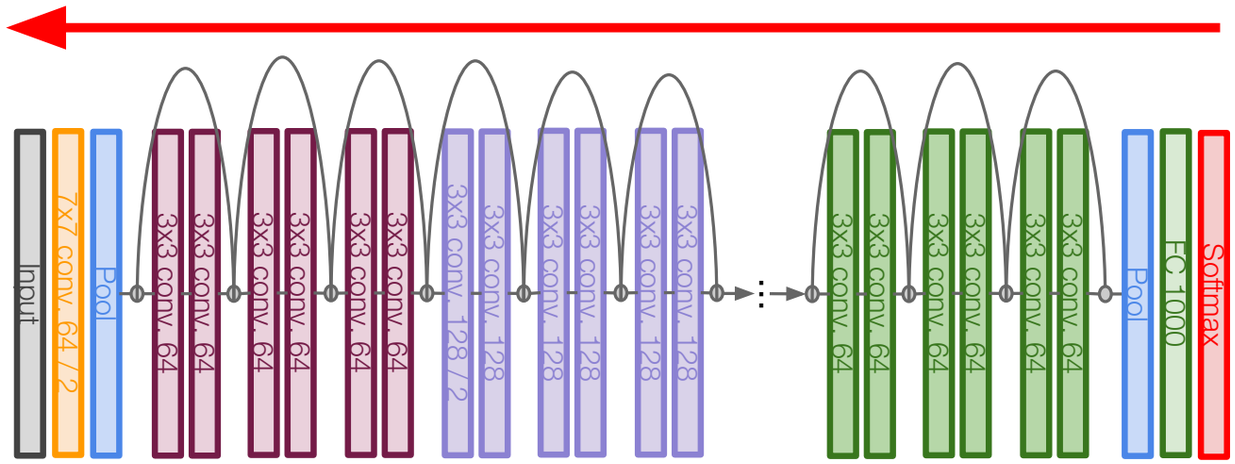
Check this link: How LSTM networks solve the problem of vanishing gradients

LSTMs gradient flow

Uninterrupted gradient flow!



Similar to ResNet!



Check this link: How LSTM networks solve the problem of vanishing gradients

LSTMs key concepts

1. Maintain a **separate cell state** from what is outputted

2. Use **gates** to control the flow of information

    - Forget gate gets rid of irrelevant information

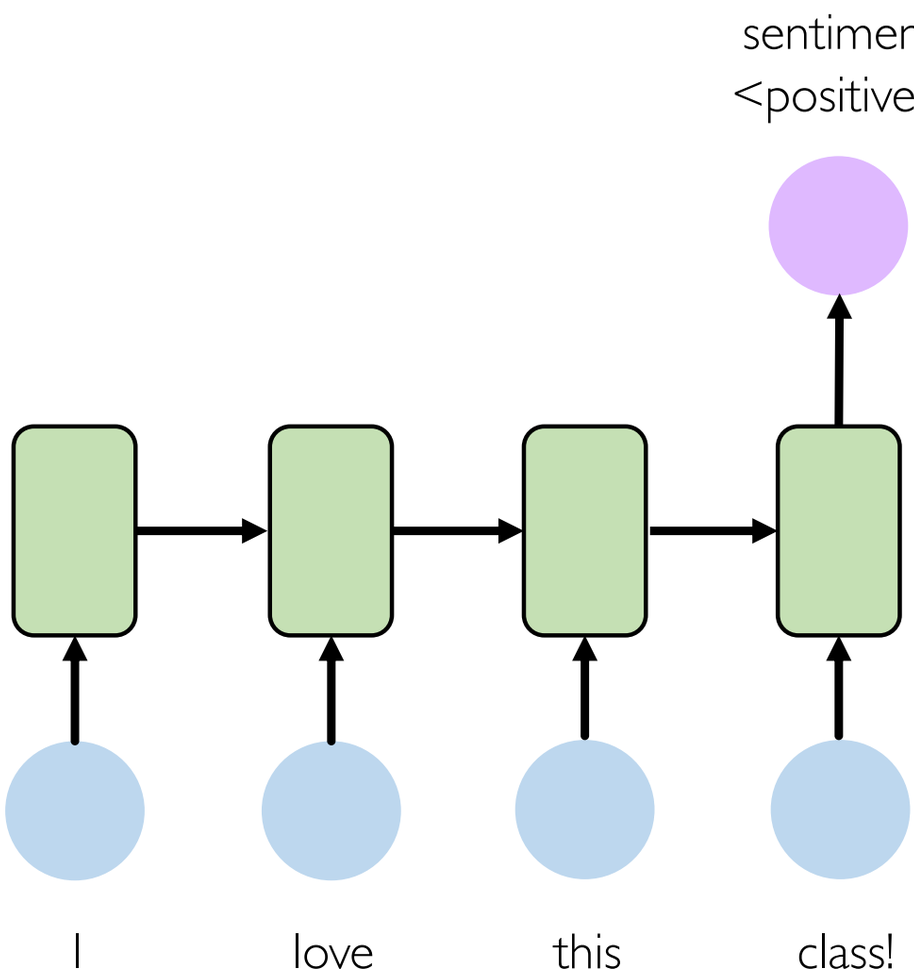
    - Selectively update cell state

    - Output gate returns a filtered version of the cell state

3. Backpropagation from c_t to c_{t-1} doesn't require matrix multiplication

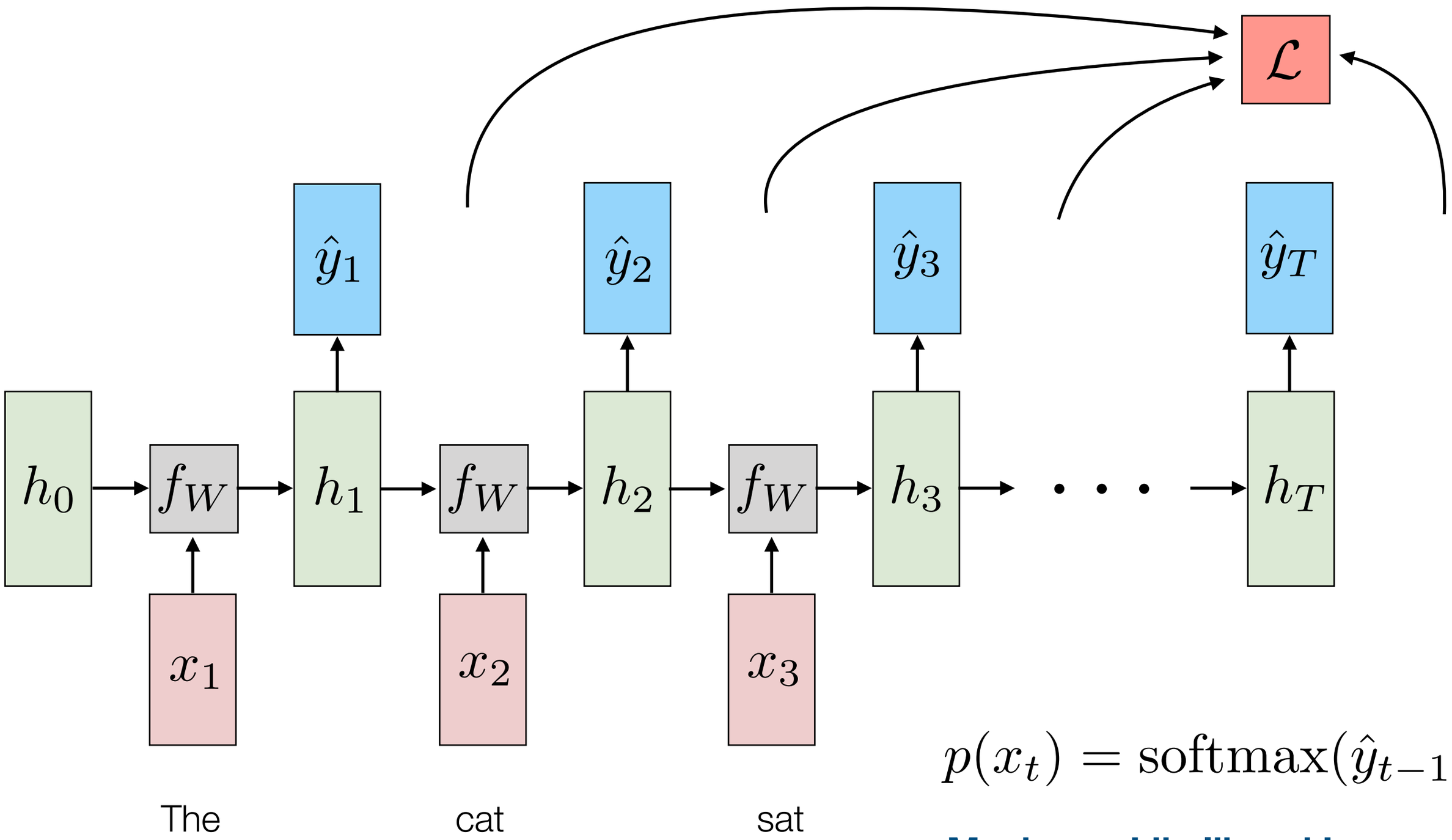
RNN Applications

Sentiment Classification



Tweet sentiment classification

Recurrent Neural Language Model (Training)

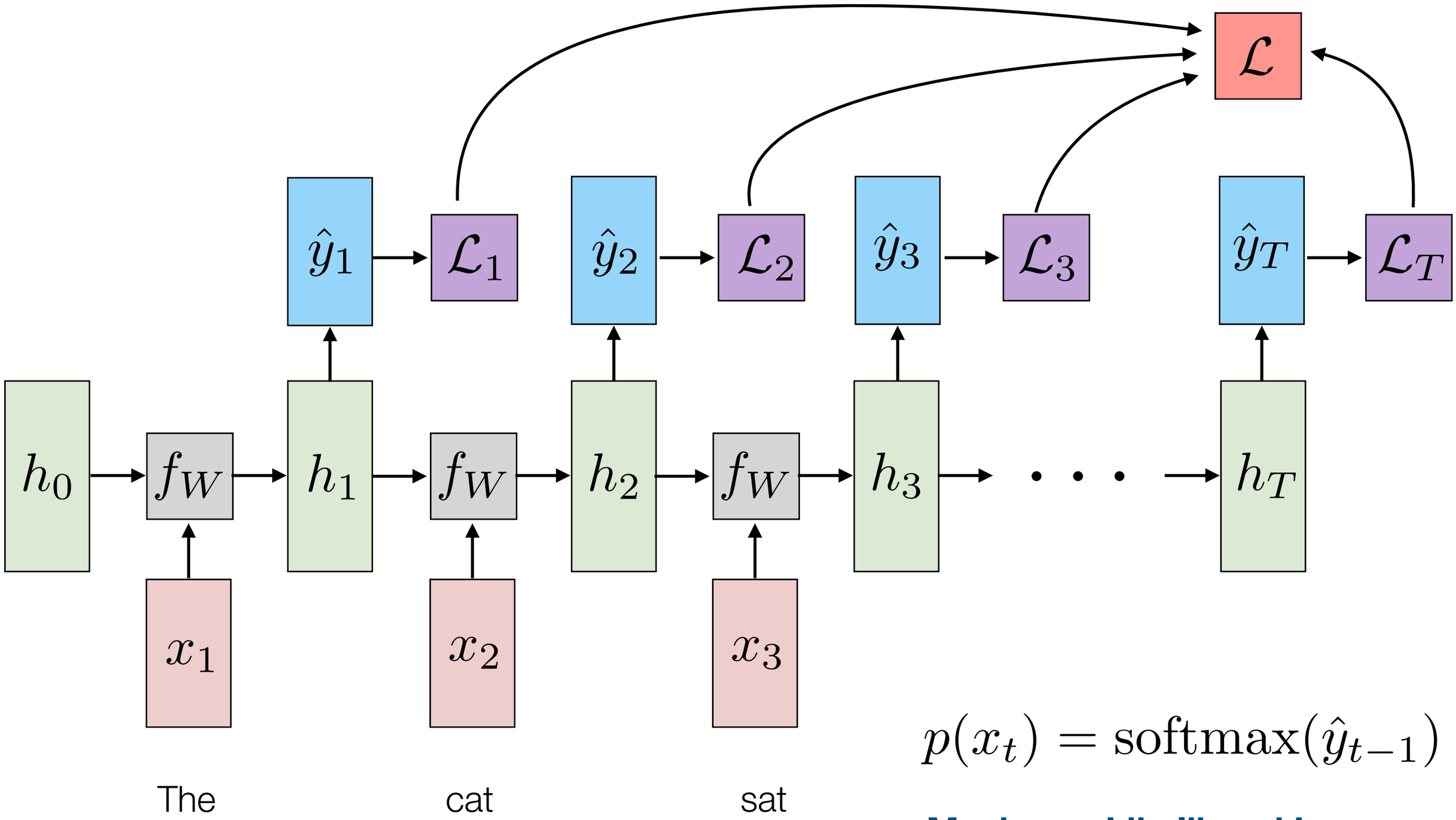


$$p(x_t) = \text{softmax}(\hat{y}_{t-1})$$

Maximum Likelihood Loss

Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Recurrent Neural Language Model (Training)



$$p(x_t) = \text{softmax}(\hat{y}_{t-1})$$

Maximum Likelihood Loss

Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

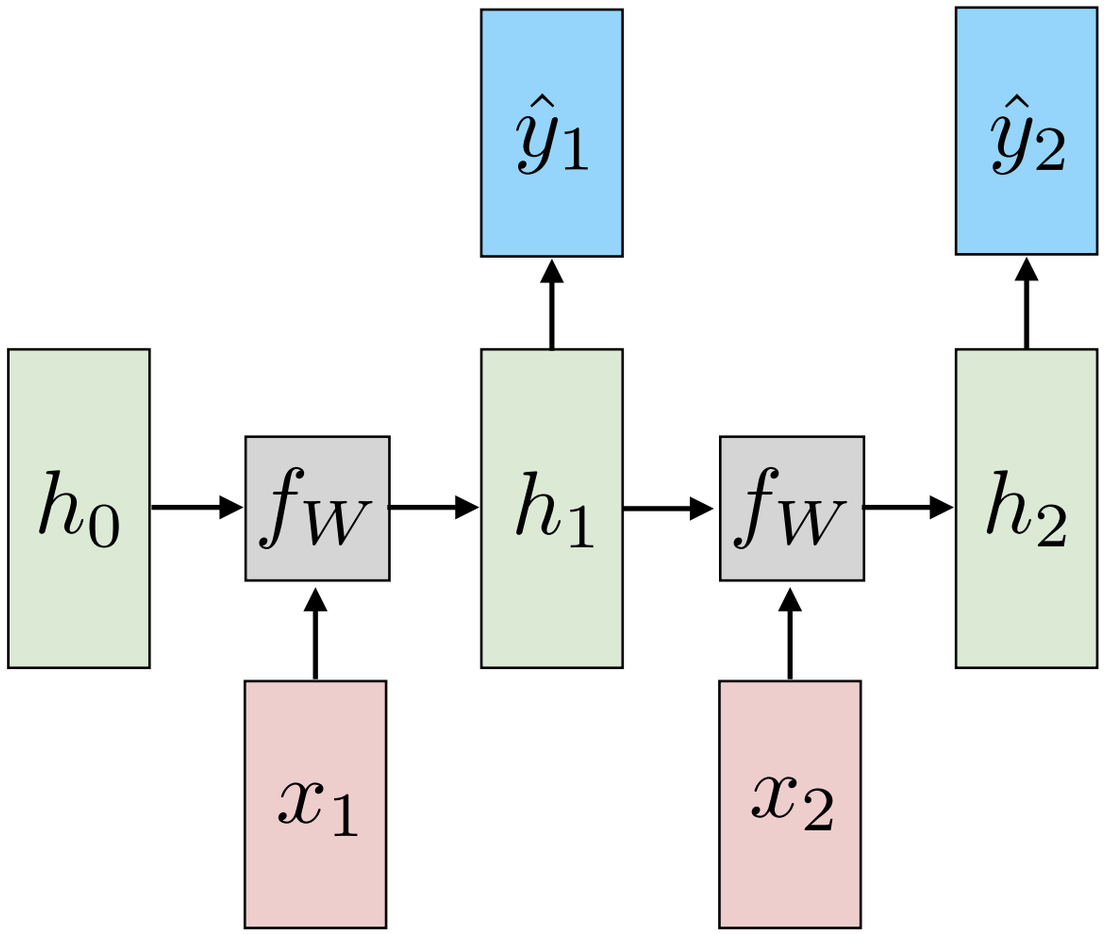
Recurrent Neural Language Model (Test)

Sequentially Sample from Output Distribution

Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Recurrent Neural Language Model (Test)

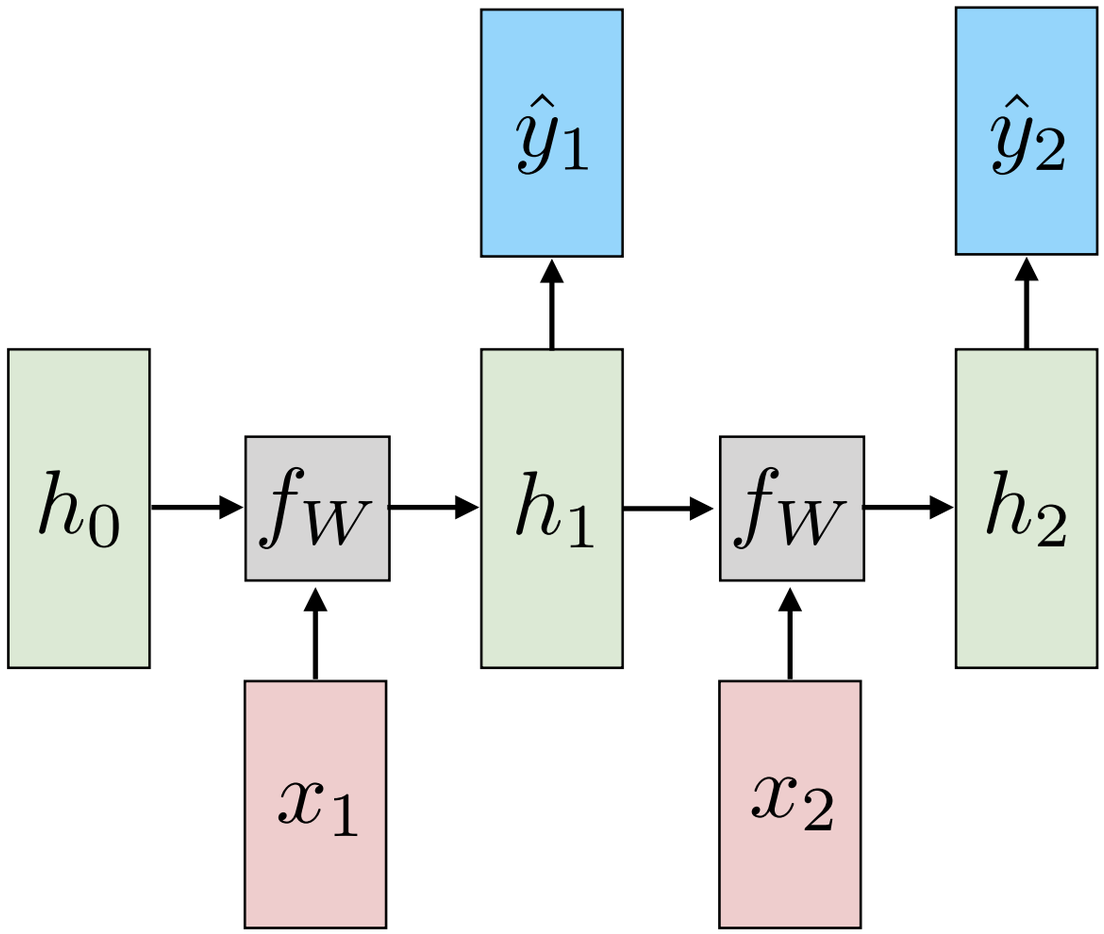
Sequentially Sample from Output Distribution



Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Recurrent Neural Language Model (Test)

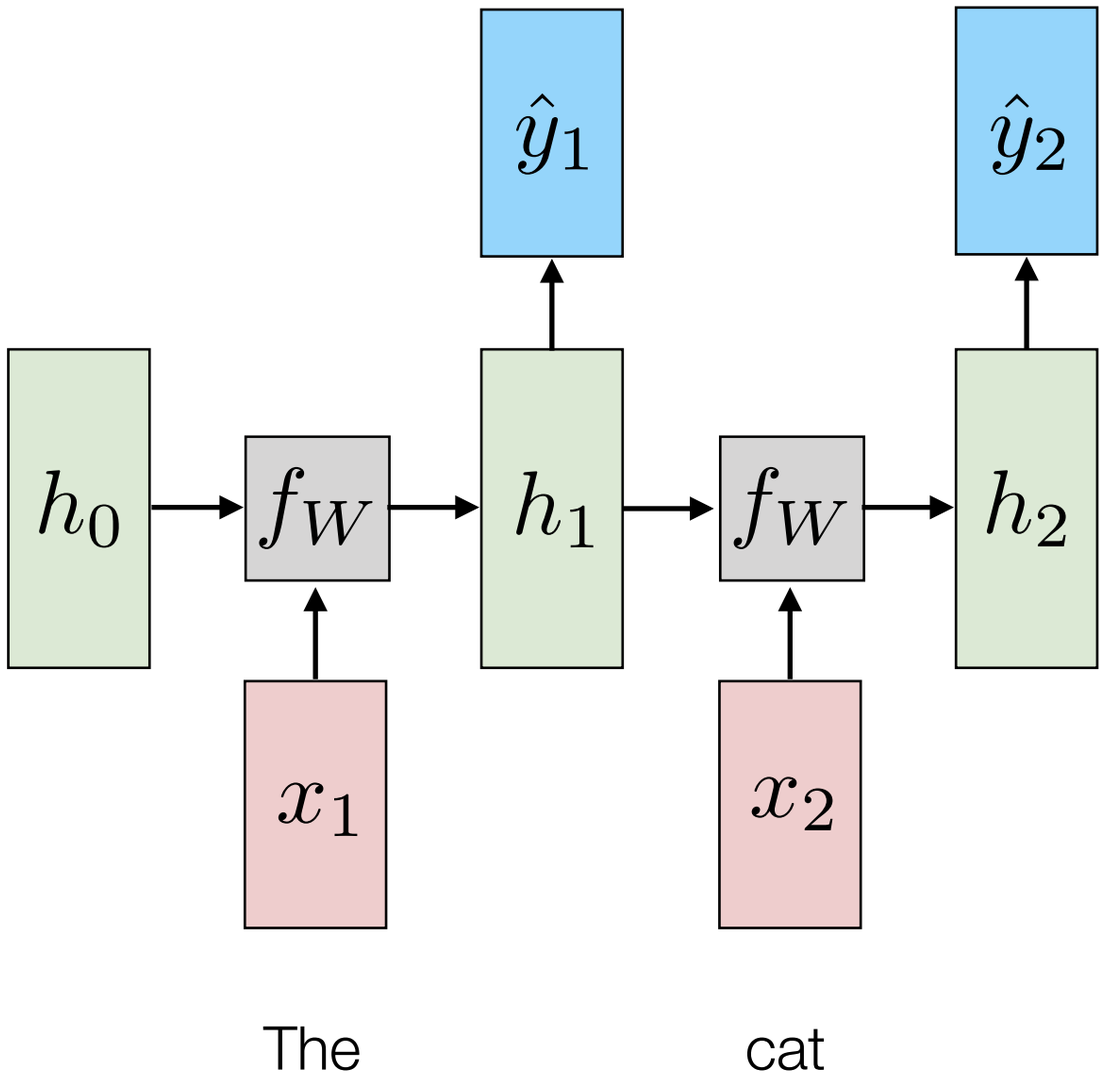
Sequentially Sample from Output Distribution



Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Recurrent Neural Language Model (Test)

Sequentially Sample from Output Distribution

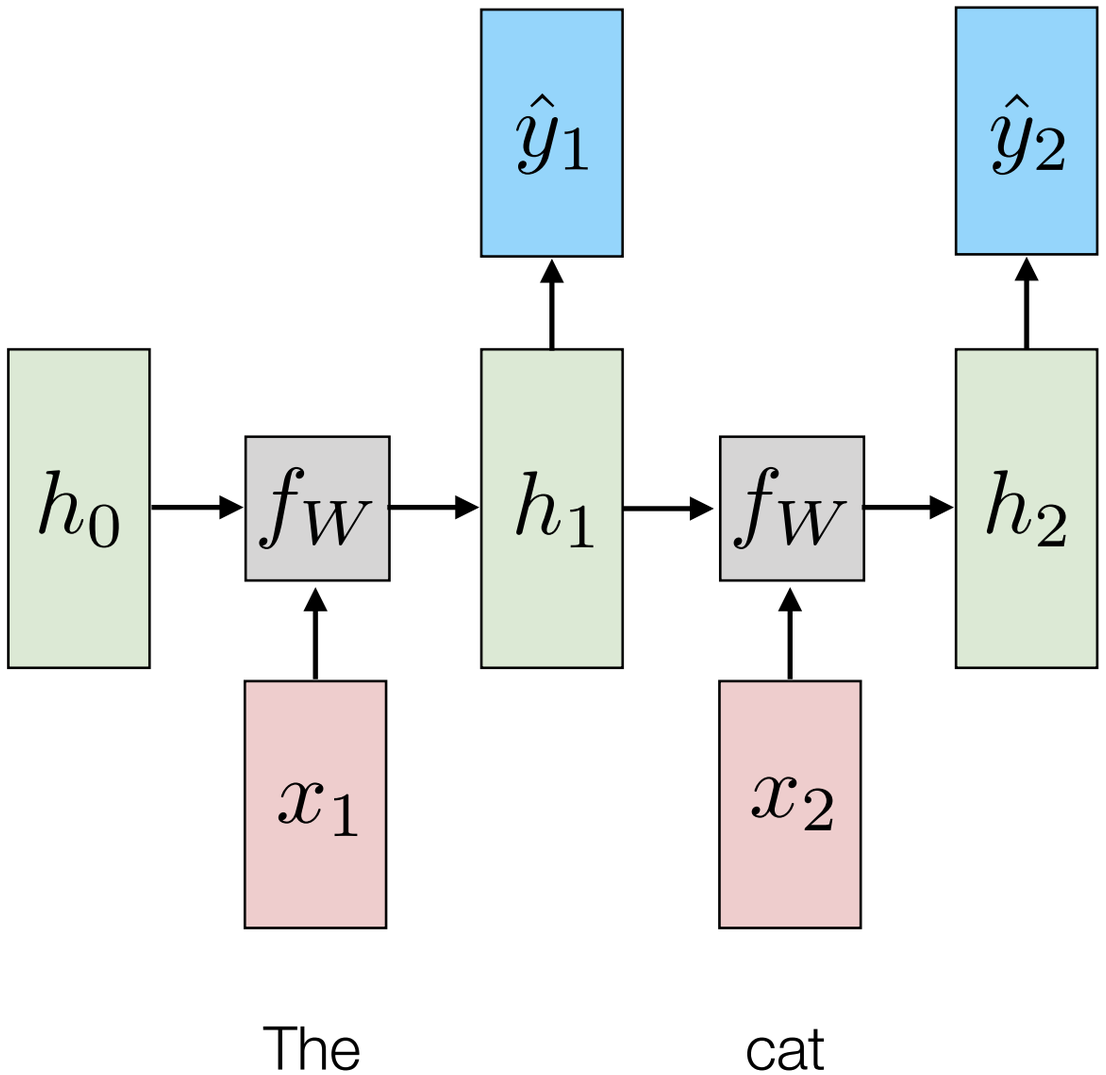


Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Recurrent Neural Language Model (Test)

Sequentially Sample from Output Distribution

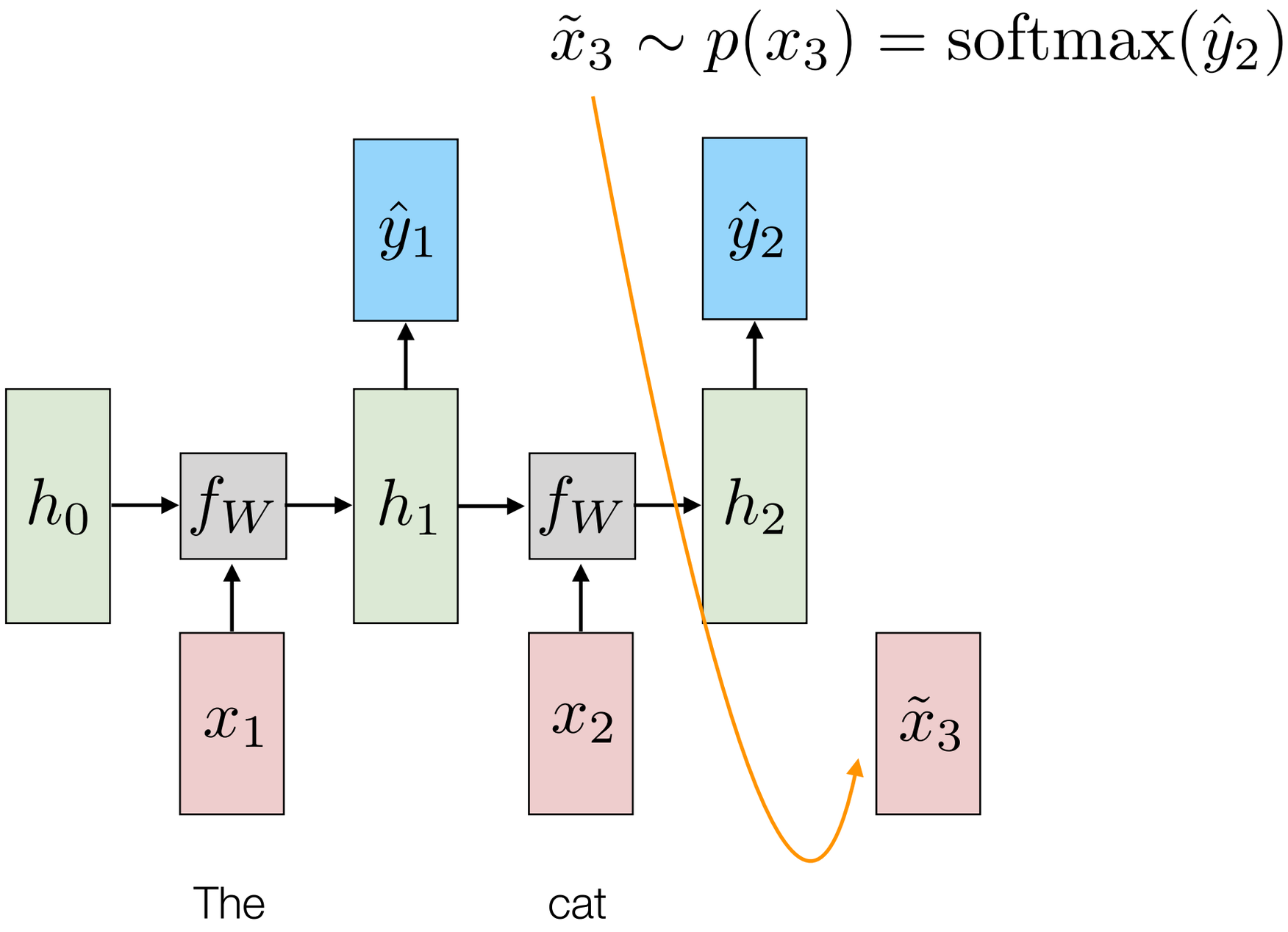
$$\tilde{x}_3 \sim p(x_3) = \text{softmax}(\hat{y}_2)$$



Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Recurrent Neural Language Model (Test)

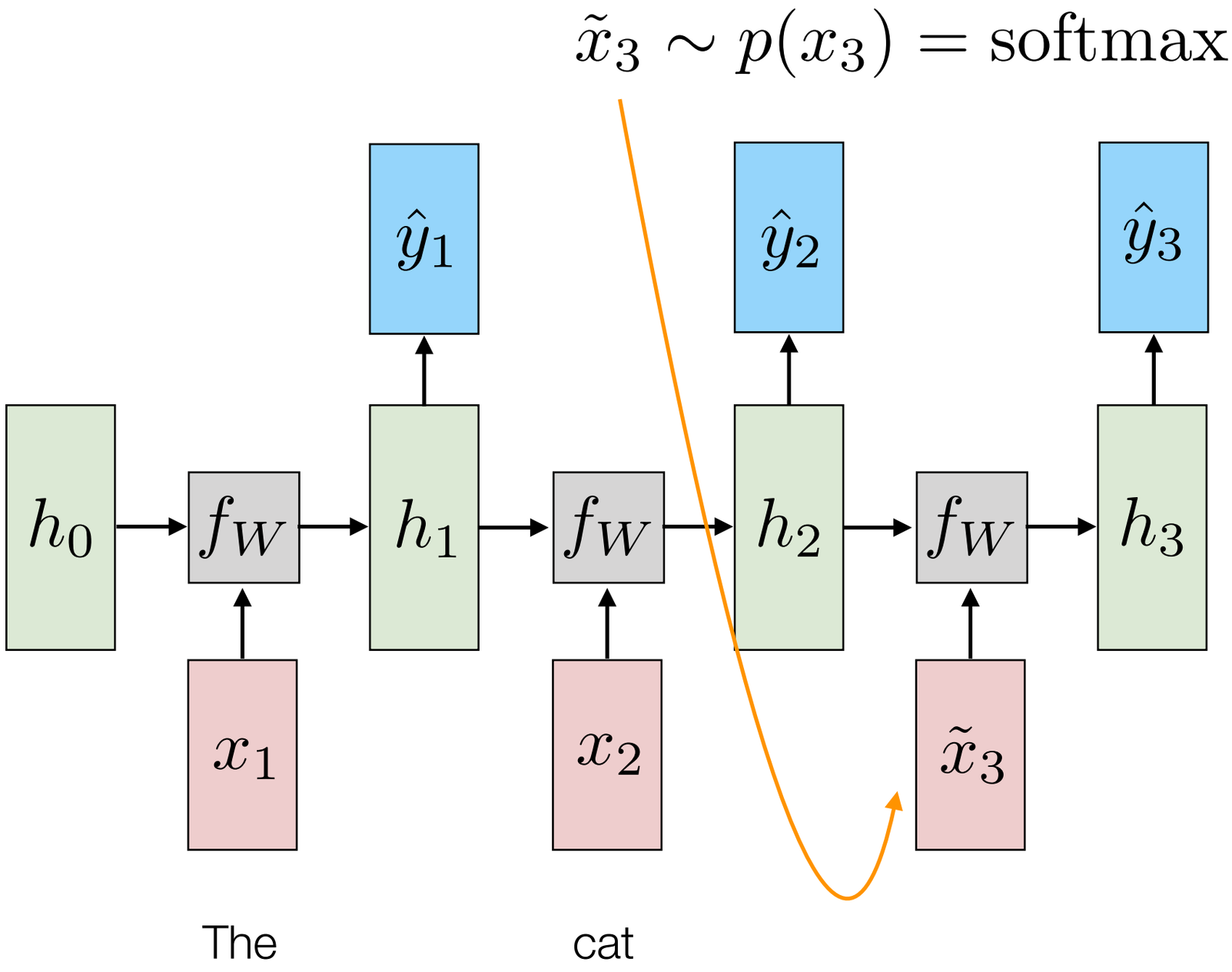
Sequentially Sample from Output Distribution



Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Recurrent Neural Language Model (Test)

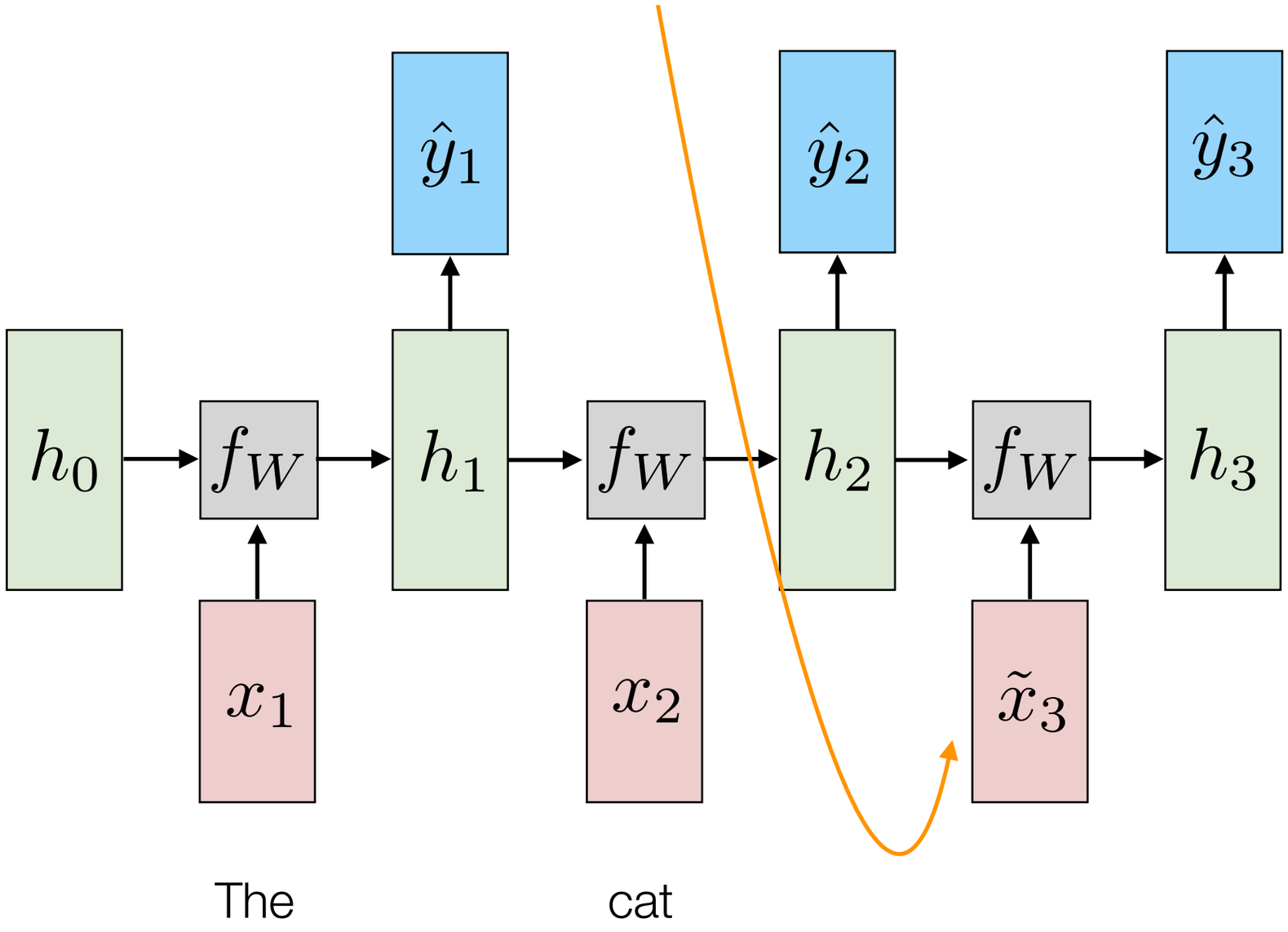
Sequentially Sample from Output Distribution



Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Recurrent Neural Language Model (Test)

Sequentially Sample from Output Distribution

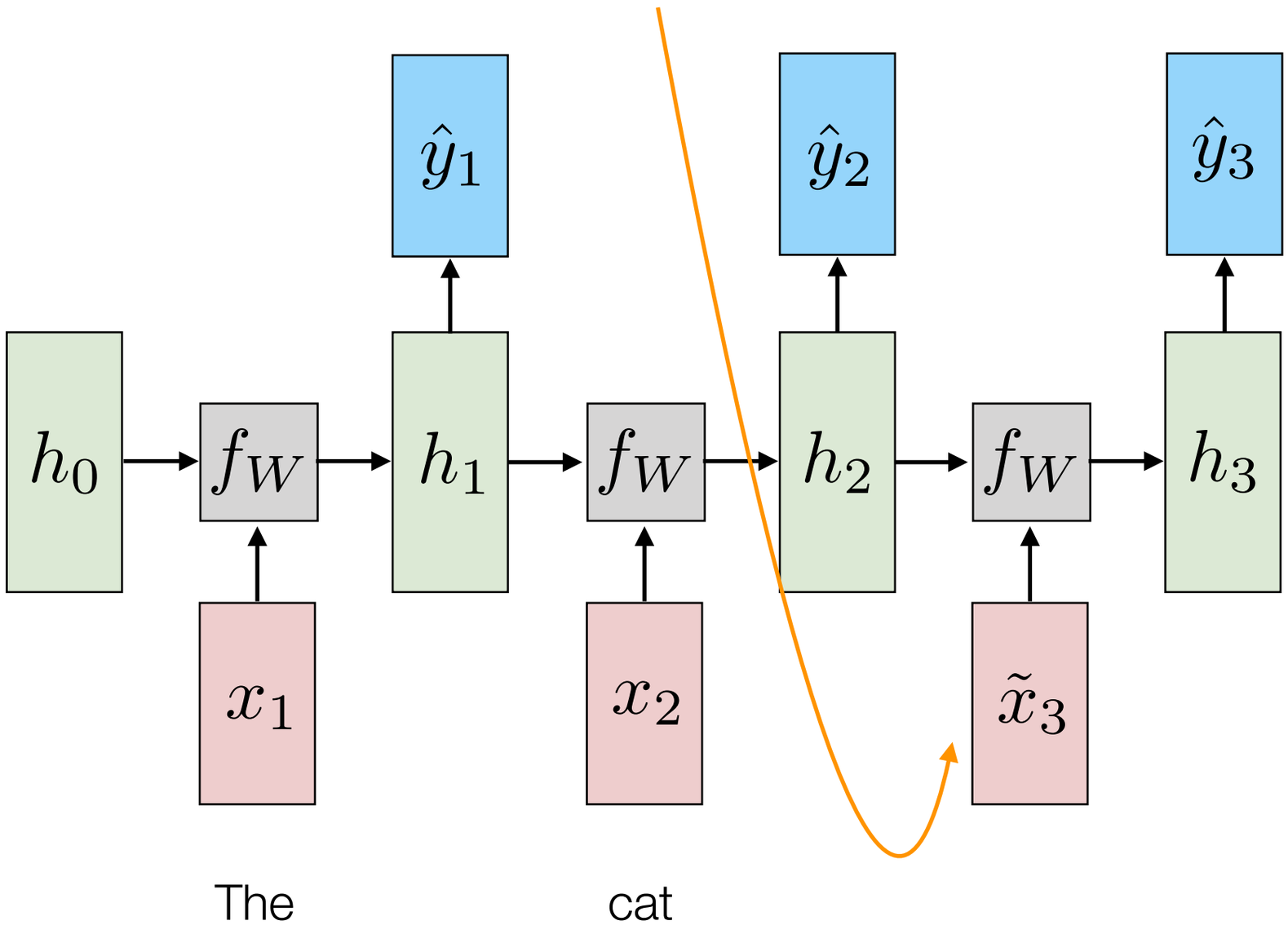


Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Recurrent Neural Language Model (Test)

Sequentially Sample from Output Distribution

$$\tilde{x}_4 \sim p(x_4) = \text{softmax}(\hat{y}_3)$$

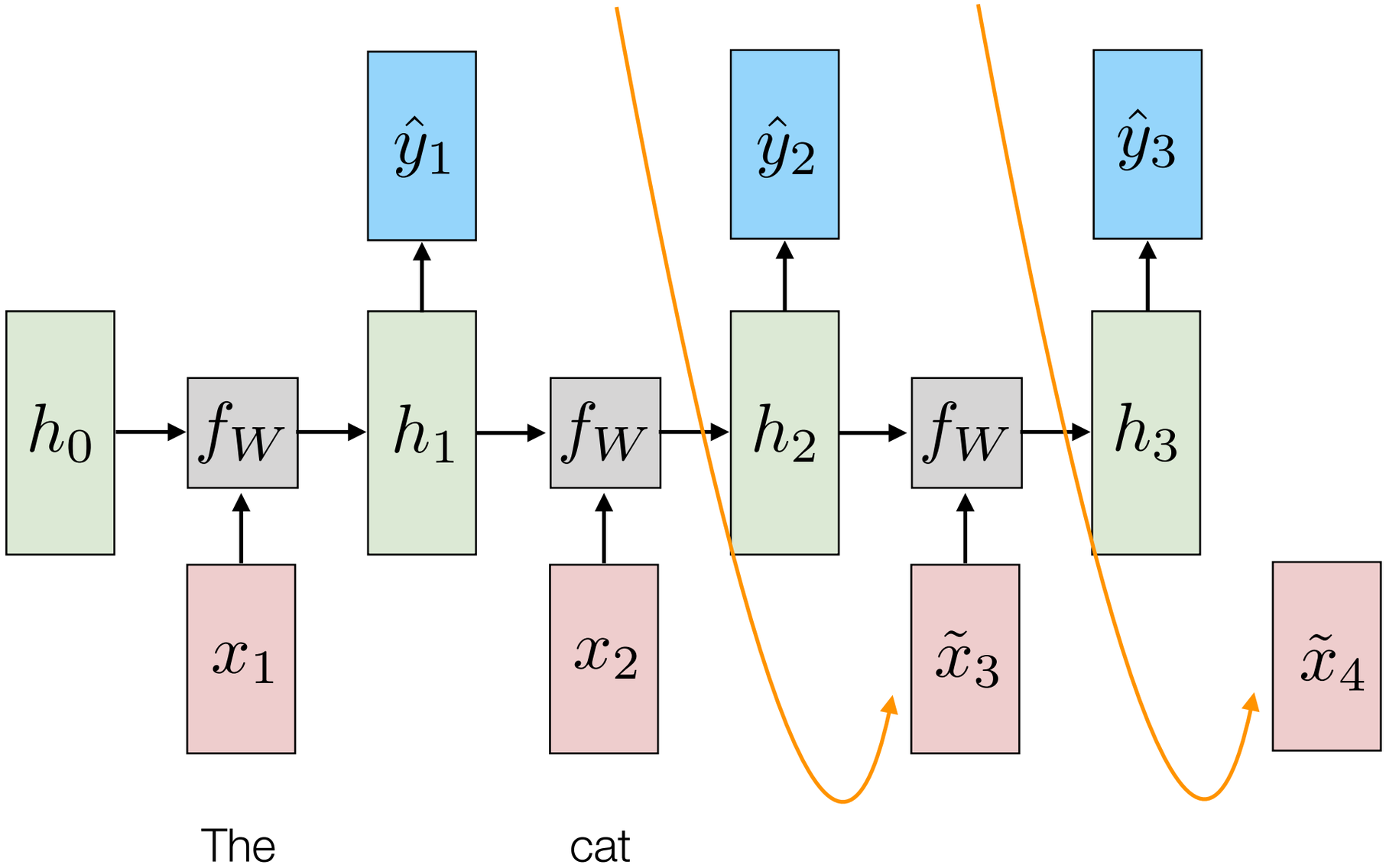


Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Recurrent Neural Language Model (Test)

Sequentially Sample from Output Distribution

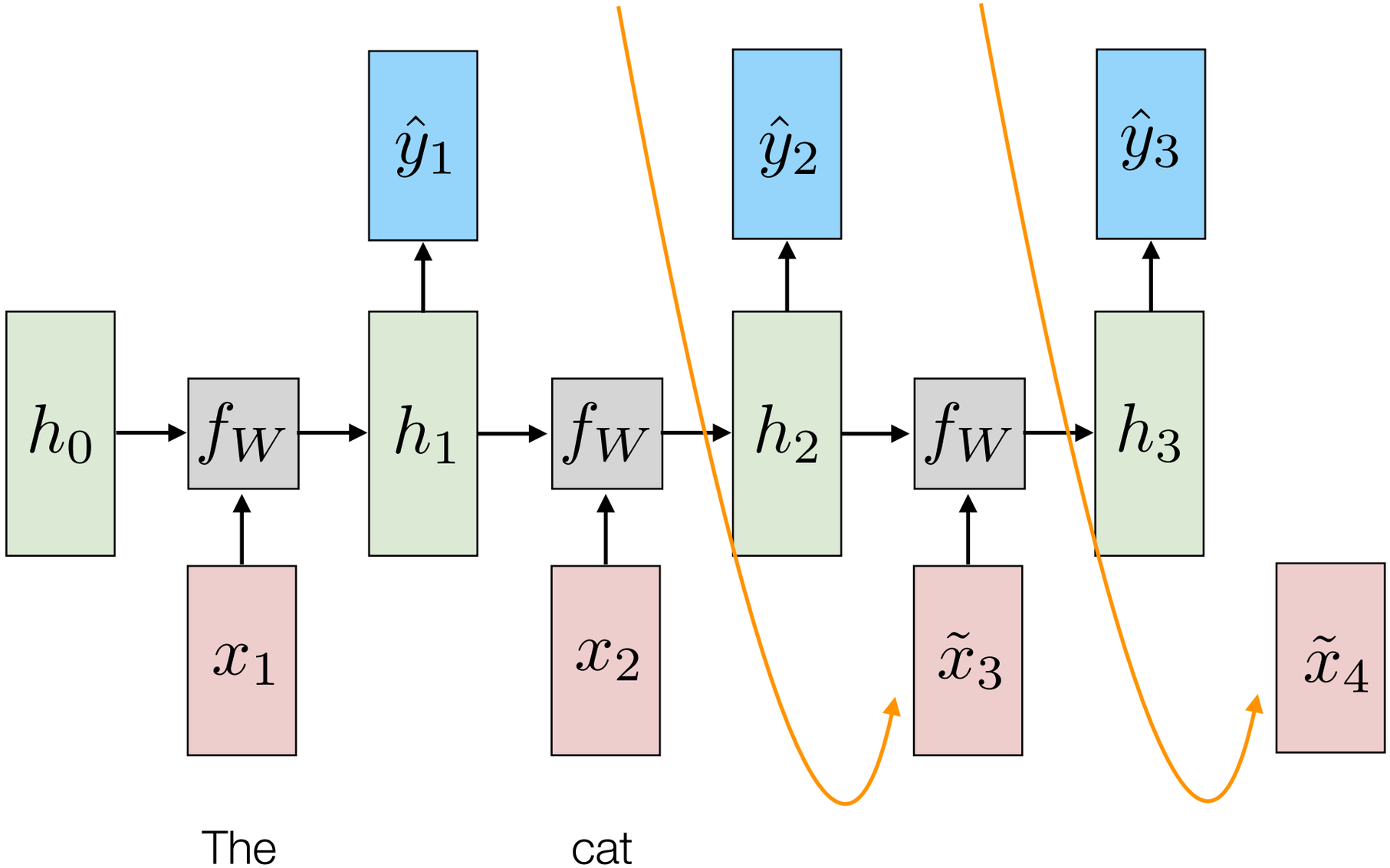
$$\tilde{x}_4 \sim p(x_4) = \text{softmax}(\hat{y}_3)$$



Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Recurrent Neural Language Model (Test)

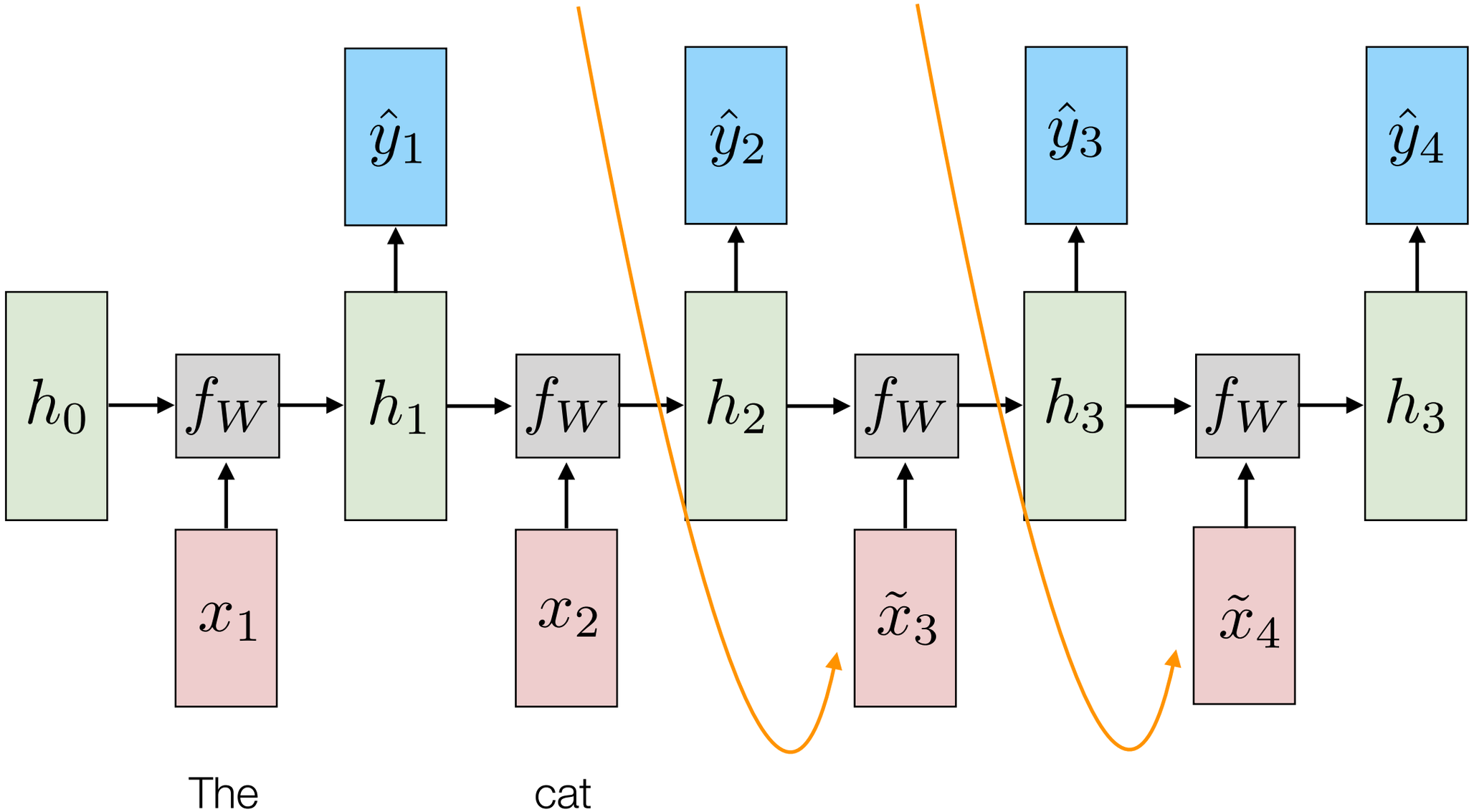
Sequentially Sample from Output Distribution



Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Recurrent Neural Language Model (Test)

Sequentially Sample from Output Distribution

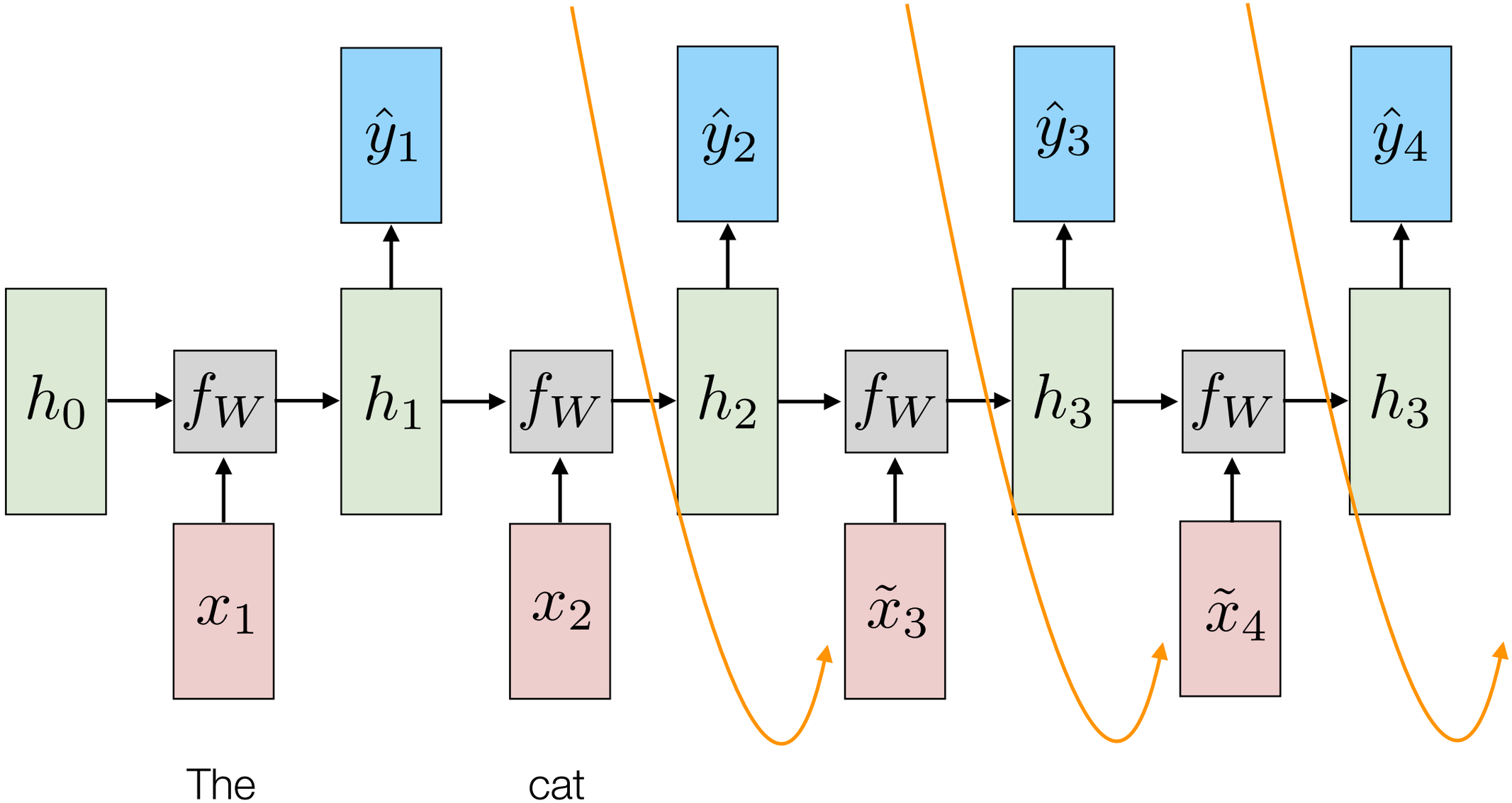


Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Recurrent Neural Language Model (Test)

Sequentially Sample from Output Distribution

$$\tilde{x}_5 \sim p(x_5) = \text{softmax}(\hat{y}_4)$$

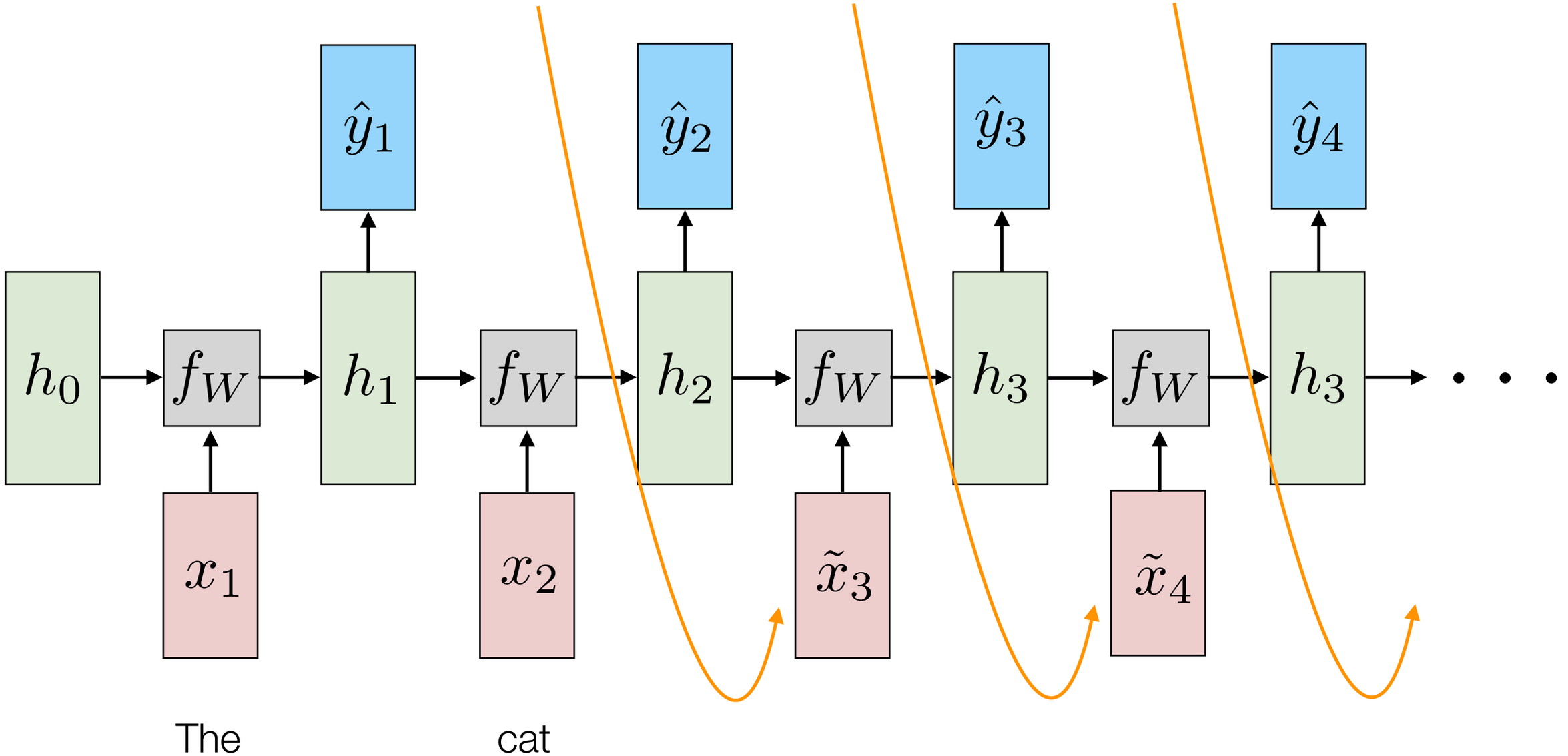


Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Recurrent Neural Language Model (Test)

Sequentially Sample from Output Distribution

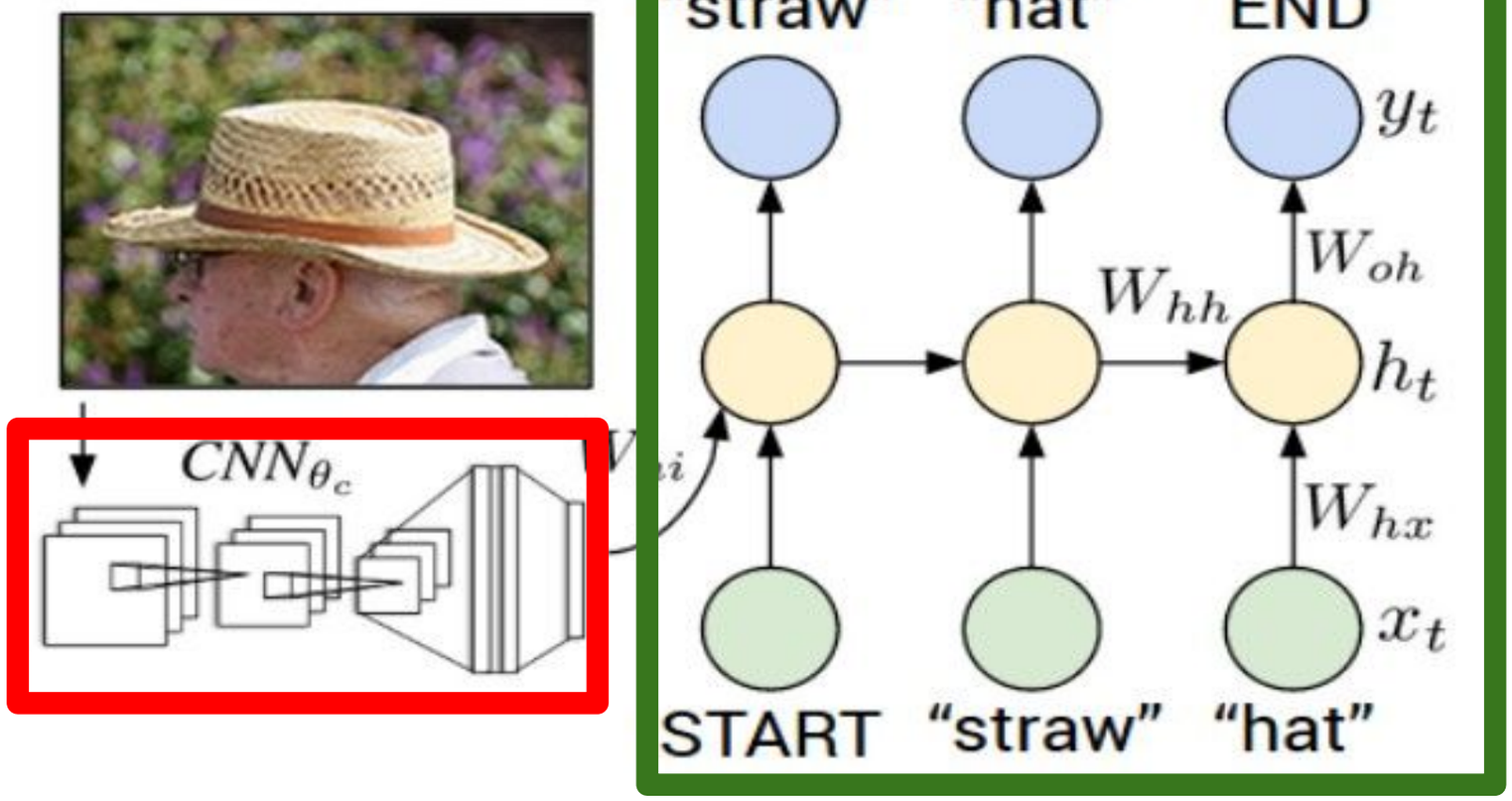
$$\tilde{x}_5 \sim p(x_5) = \text{softmax}(\hat{y}_4)$$



Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. InProceedings of INTERSPEECH

Image Captioning

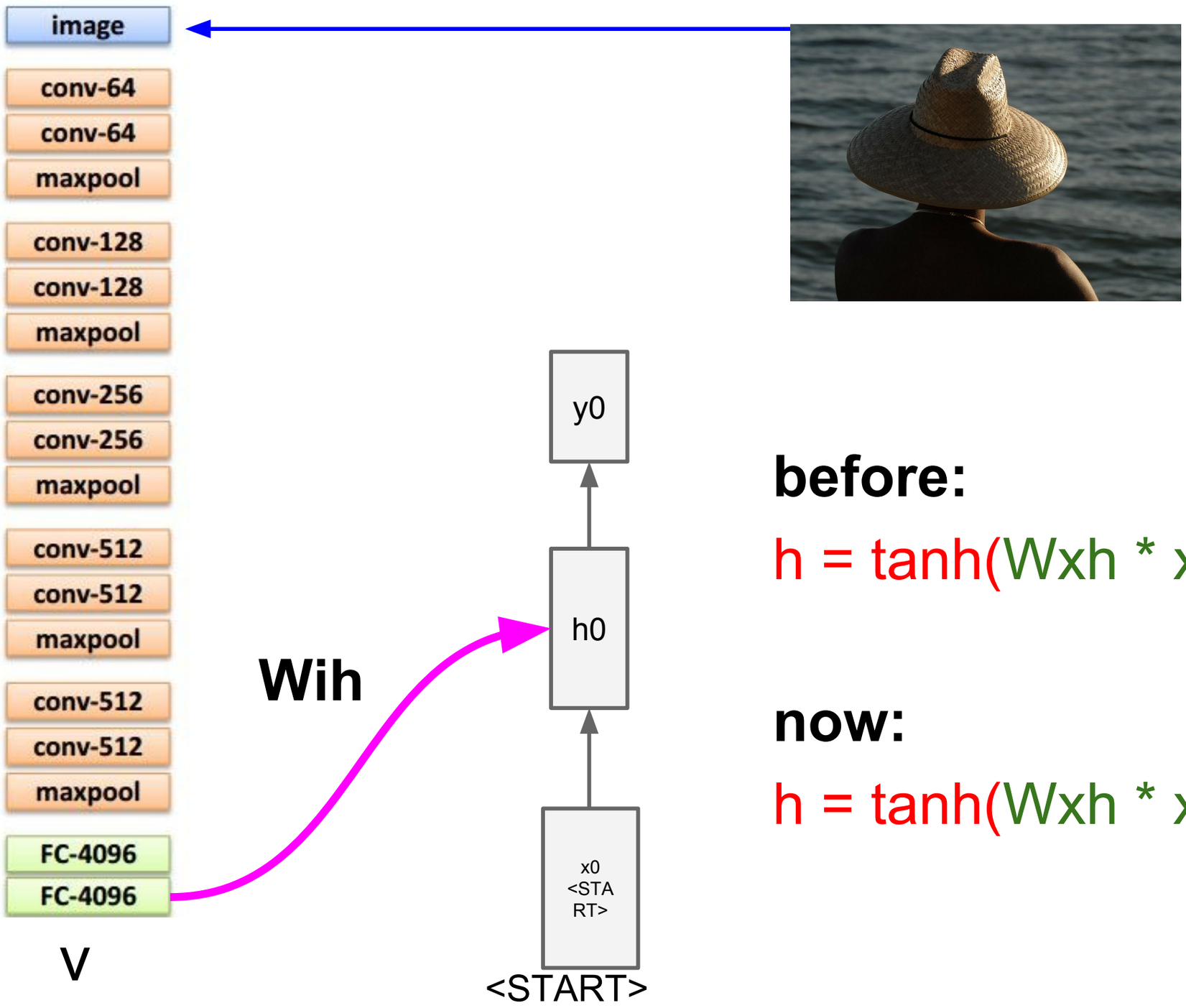
Recurrent Neural Network



Convolutional Neural Network

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
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

Image Captioning



before:

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

now:

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

Image Captioning

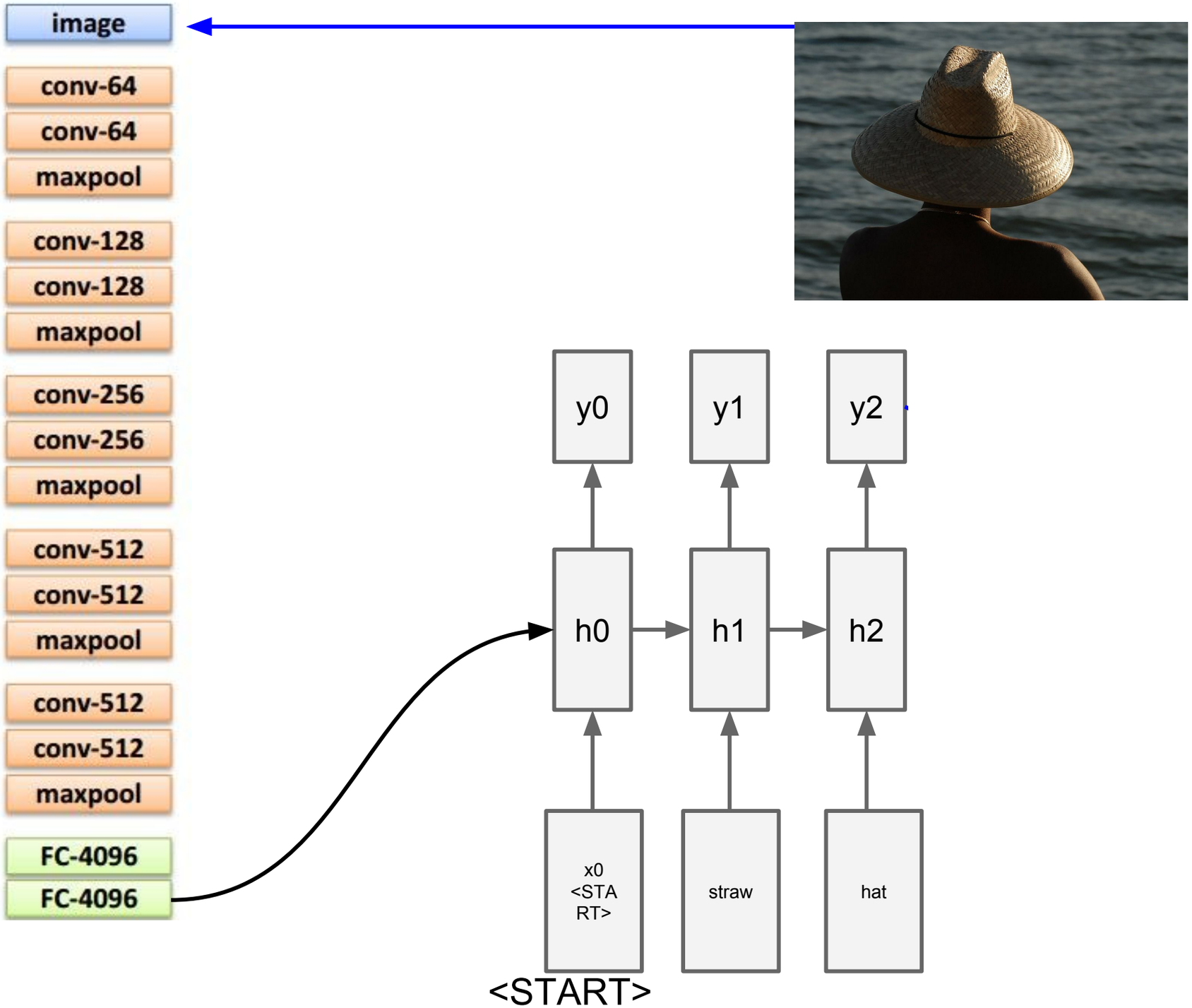


Image Captioning: Example Results

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