CSCI 631
Foundations of Computer Vision

Ifeoma Nwogu
ion@cs.rit.edu

Lecture -Recurrent Neural Networks (RNN)

Schedule

- Last class
  - CNN
- Today
  - Recurrent neural networks (RNN)

Slides curtsey of Svetlana Lazebnik and Arun Mallya

Motivation for RNN

- Not all problems can be converted into one with fixed-length inputs and outputs
- Problems such as Speech Recognition or Time-series Prediction require a system to store and use context information
  - Simple case: Output YES if the number of 1s is even, else NO
    1000010101 – YES, 100011 – NO, …
- Hard/Impossible to choose a fixed context window
  - There can always be a new sample longer than anything seen

The parity problem

- Data transmission in communication systems
- A source sends a bitstream to a receiver
- How can we detect if one bit is reversed?
- Add a parity bit to the end of each stream
- Agree that streams have even number of ones
- If #1s in stream is odd, add a parity bit 1 to make total even
- If #1s in stream is even, add a parity bit 0 to leave total even
- When stream is received if total number of bits is odd, there is an error

More on RNN motivation

Fill in the blanks:

- I just took a _____
- I'm tired; I just took a _____
- Finals were today; I'm tired; I just took a _____

Motivation continued

- RNNs take in a large corpus of text or other sequences and analyzes the sequences for context
  - Context info gets better as sequences get larger.

- RNNs analyze sequences to make **predictions** about them and **infer context** from them.

Why not CNN?

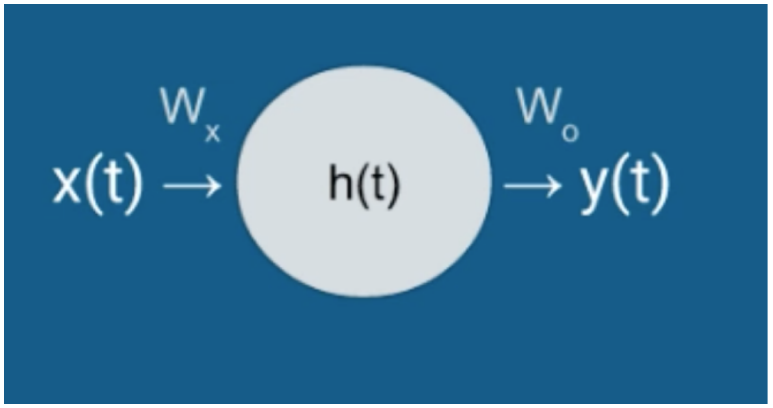
- RNNs have memory - they are suited to problems when we incorporate previous predictions to make new prediction.
  - E.g. with sequential signals such as audio, video, text.

Recurrent Neural Networks (RNNs)

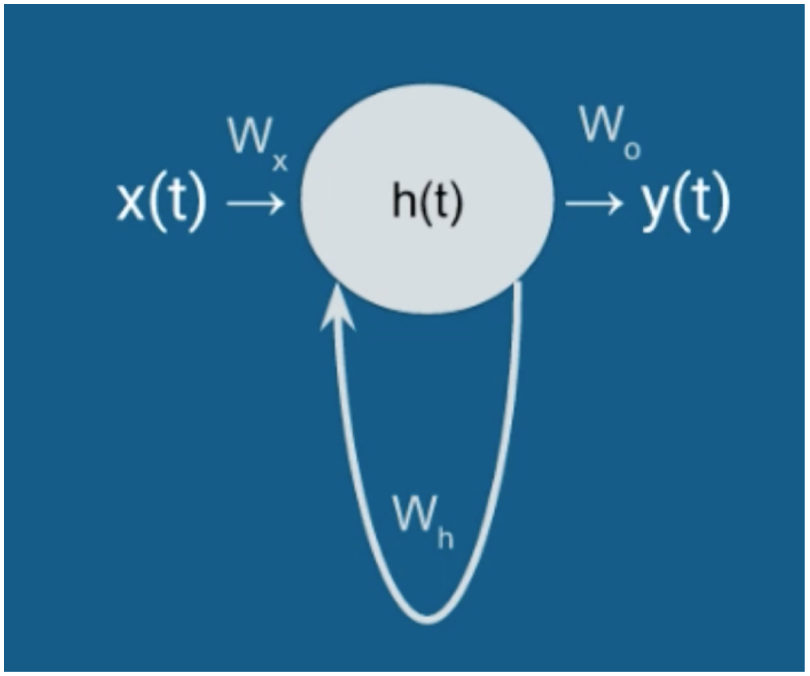
- Recurrent Neural Networks take the previous output or hidden states as inputs.
  The composite input at time t has some historical information about the happenings at time T < t

- RNNs are useful as their intermediate values (state) can store information about past inputs for a time that is not fixed a priori

Simple recurrent unit



Basic feed forward network



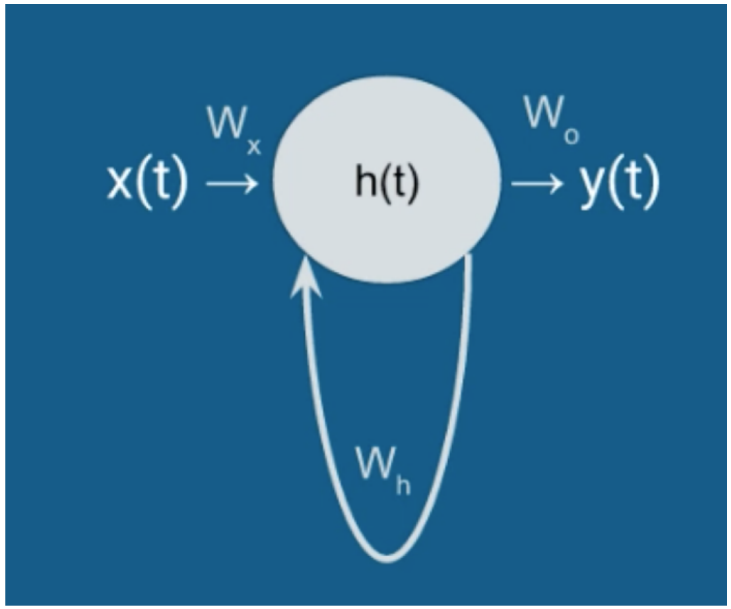
Recurrent network

Simple recurrent unit

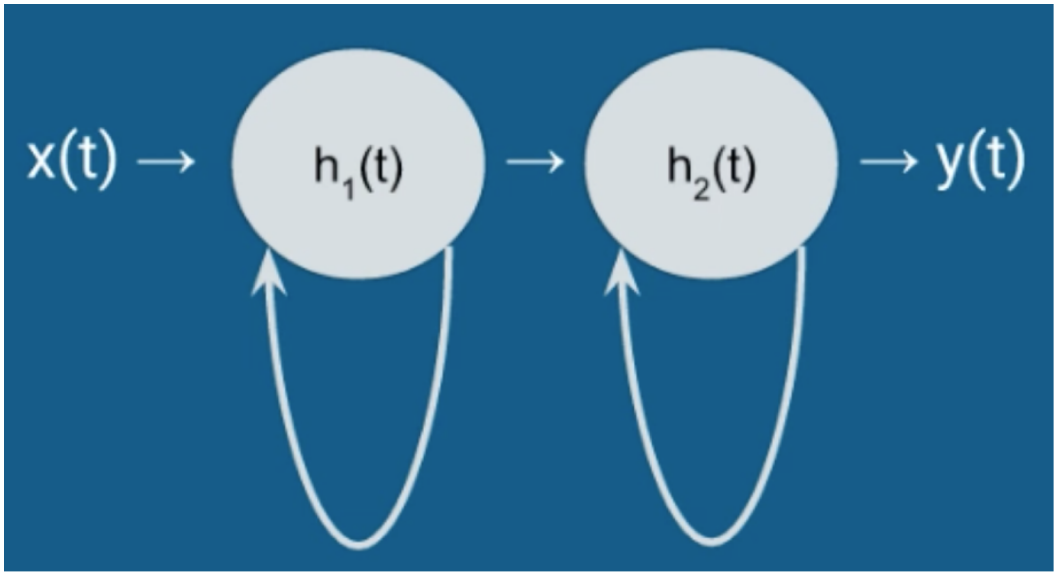
- What is the size of W_h?
- Like previous networks, across layers, everything connects to everything
  - One node in h(t) connects back to all nodes (from previous time step)
  - If h(t) has M nodes, then W_h is an M-by-M matrix

Simple recurrent unit

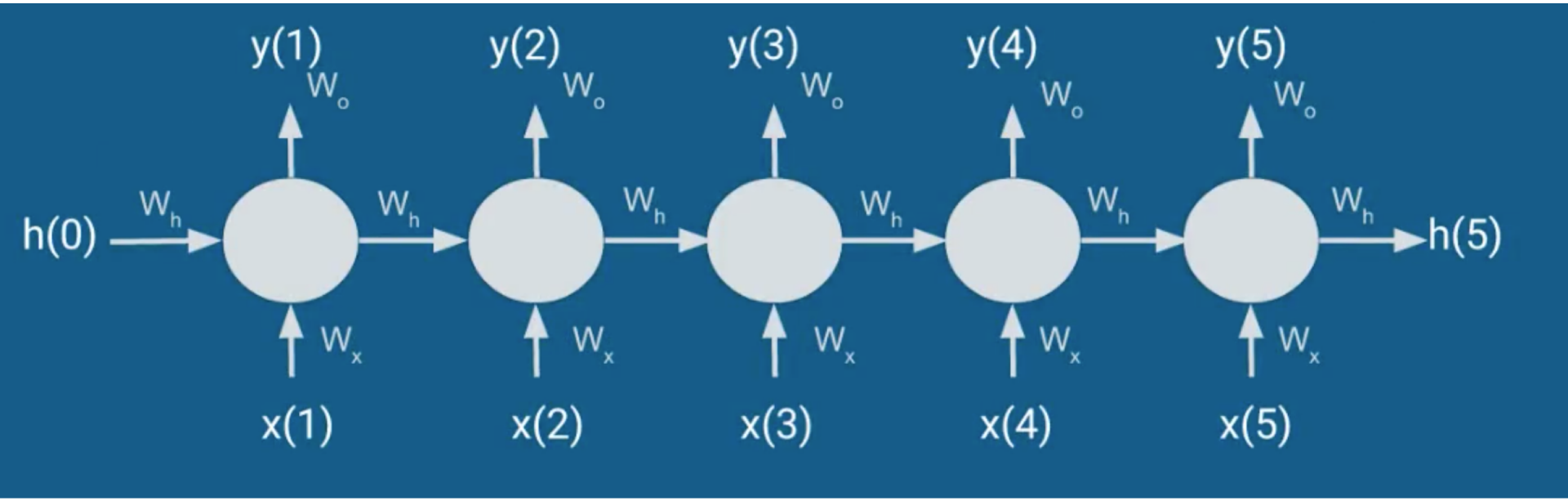
- What is the size of W_h?
- $h(t) = f(W_h^T h(t-1) + W_x^T x(t) + b_h)$
- $y(t) = \text{softmax}(W_o^T h(t) + b_o)$
- f = tanh, sigmoid or ReLU



RNN with multiple hidden layers



The unfolded RNN

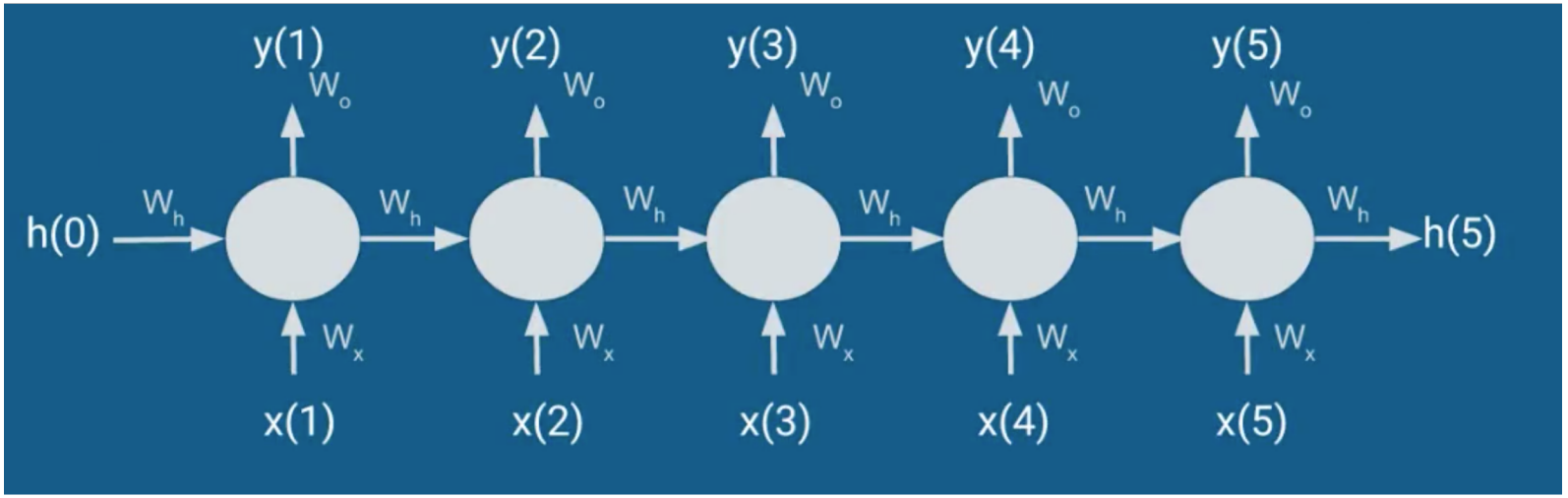


RNN uses

1. Predict an entire sequence (class labels)
   - Male versus female gait
2. Predict a class label for every time step
   - Continuous emotion monitoring (election debates)
3. Predict the next value (no labels)
   - Image/caption generations from video sequences

Backpropagating an RNN

Backpropagation is just a fancy name for gradient descent!



$$y(t) = softmax(W_o{}^T h(t))$$

$$y(t) = softmax(W_o{}^T f(W_h{}^T h(t-1) + W_x{}^T x(t)))$$

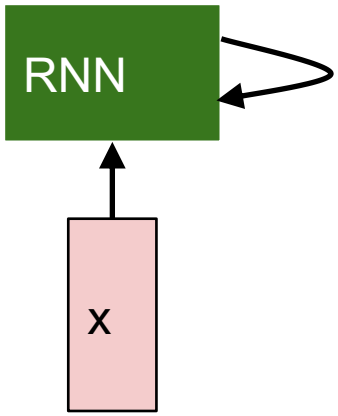
$$y(t) = softmax(W_o{}^T f(W_h{}^T f(W_h{}^T h(t-2) + W_x{}^T x(t-1)) + W_x{}^T x(t)))$$

$$y(t) = softmax(W_o{}^T f(W_h{}^T f(W_h{}^T f(W_h{}^T h(t-3) + W_x{}^T x(t-2)) + W_x{}^T x(t-1)) + W_x{}^T x(t)))$$

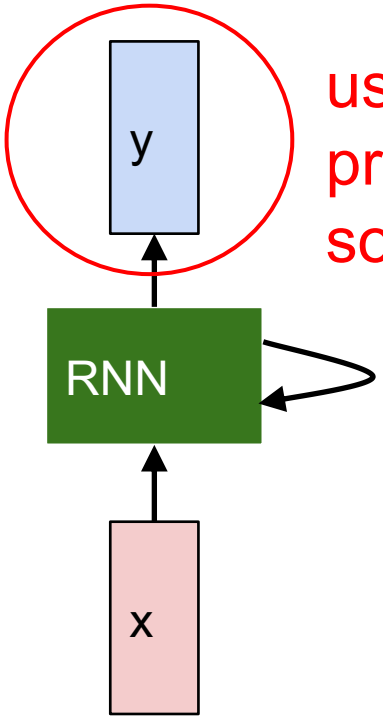
2 Key Ideas

- Parameter Sharing
  - in computation graphs = adding gradients
- "Unrolling"
  - in computation graphs with parameter sharing
- Parameter sharing + Unrolling
  - Allows modeling arbitrary sequence lengths!
  - Keeps numbers of parameters in check

Recurrent Neural Network



Recurrent Neural Network



usually want to predict a vector at some time steps

Recurrent Neural Network

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

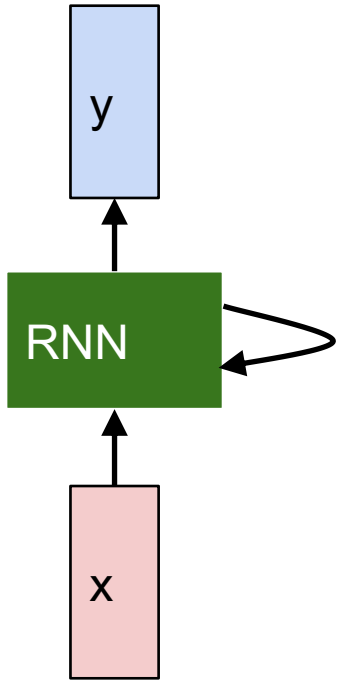
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

input vector at some time step

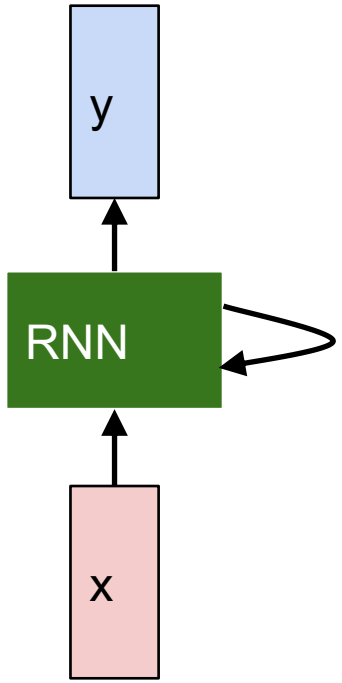
some function with parameters W



Recurrent Neural Network

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

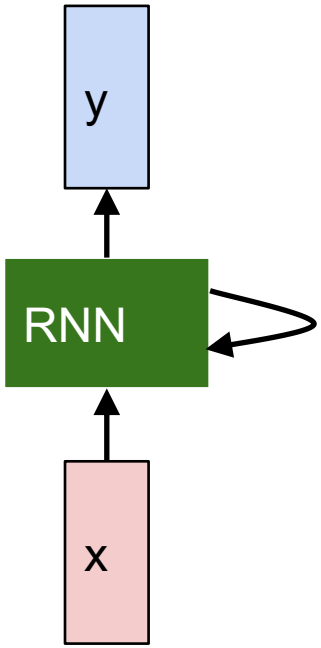
$$h_t = f_W(h_{t-1}, x_t)$$



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

(Vanilla) Recurrent Neural Network

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



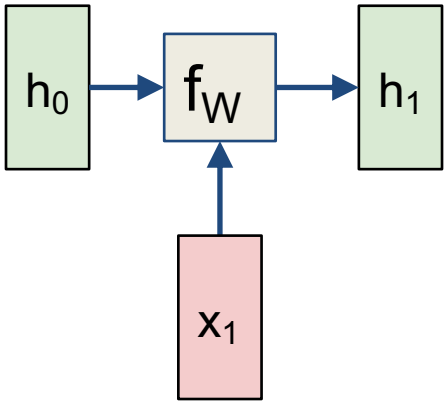
$$y_t = W_{hy} h_t + b_y$$

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

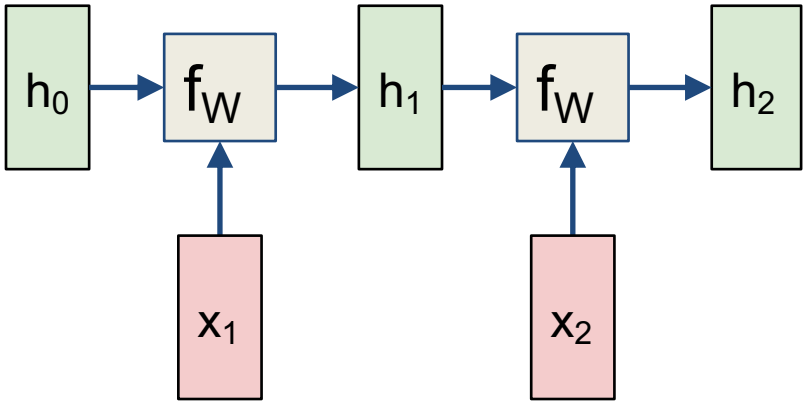
$$\downarrow$$

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

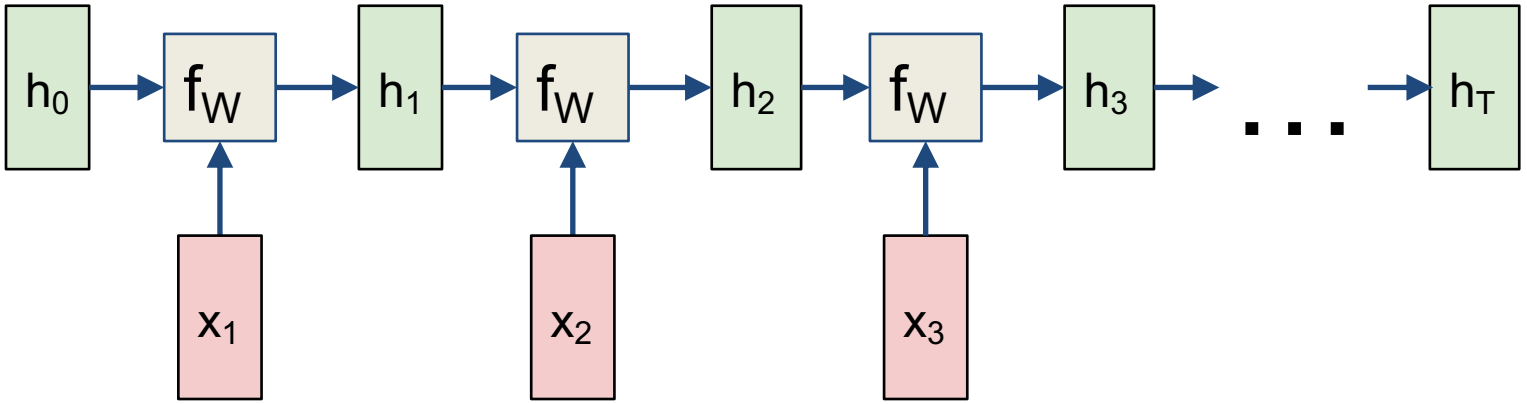
RNN: Computational Graph



RNN: Computational Graph



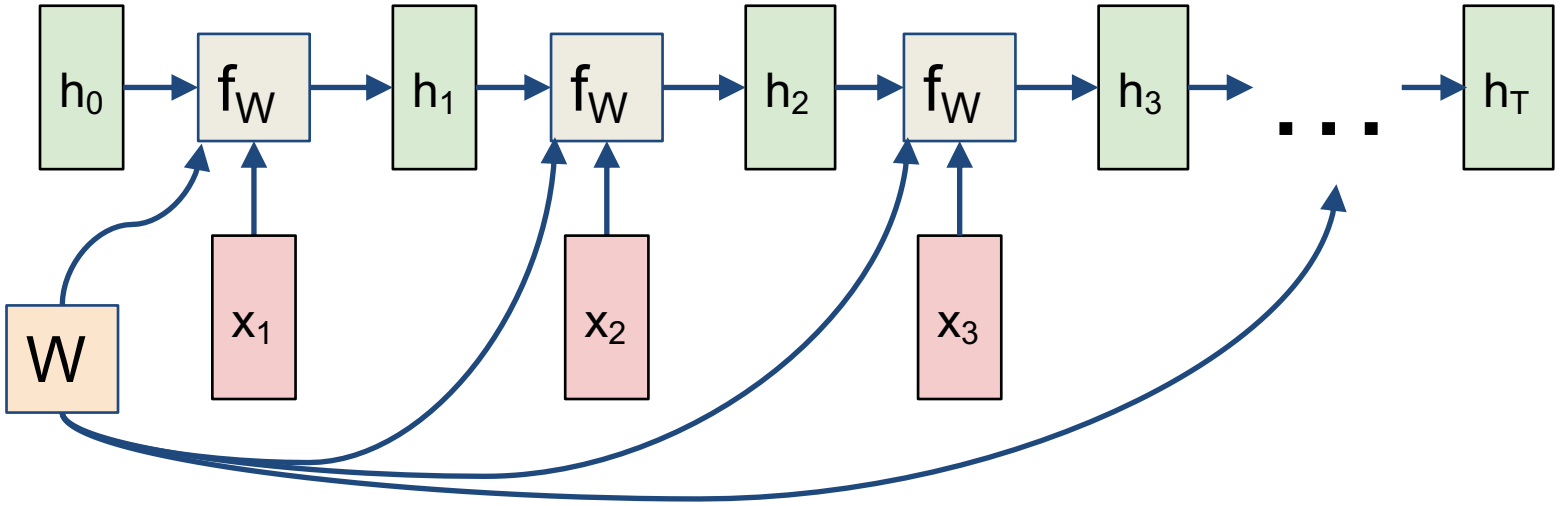
RNN: Computational Graph



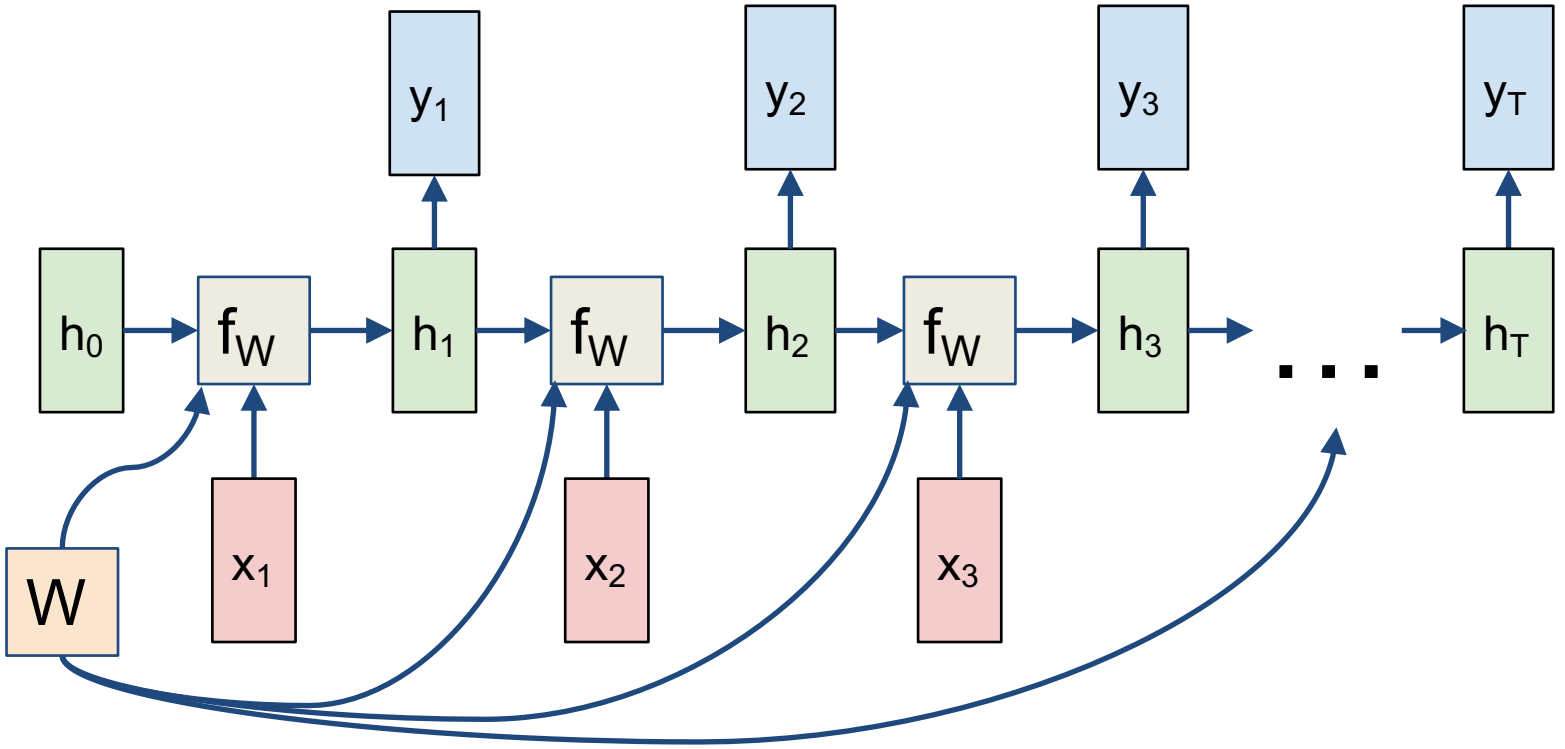
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

RNN: Computational Graph

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

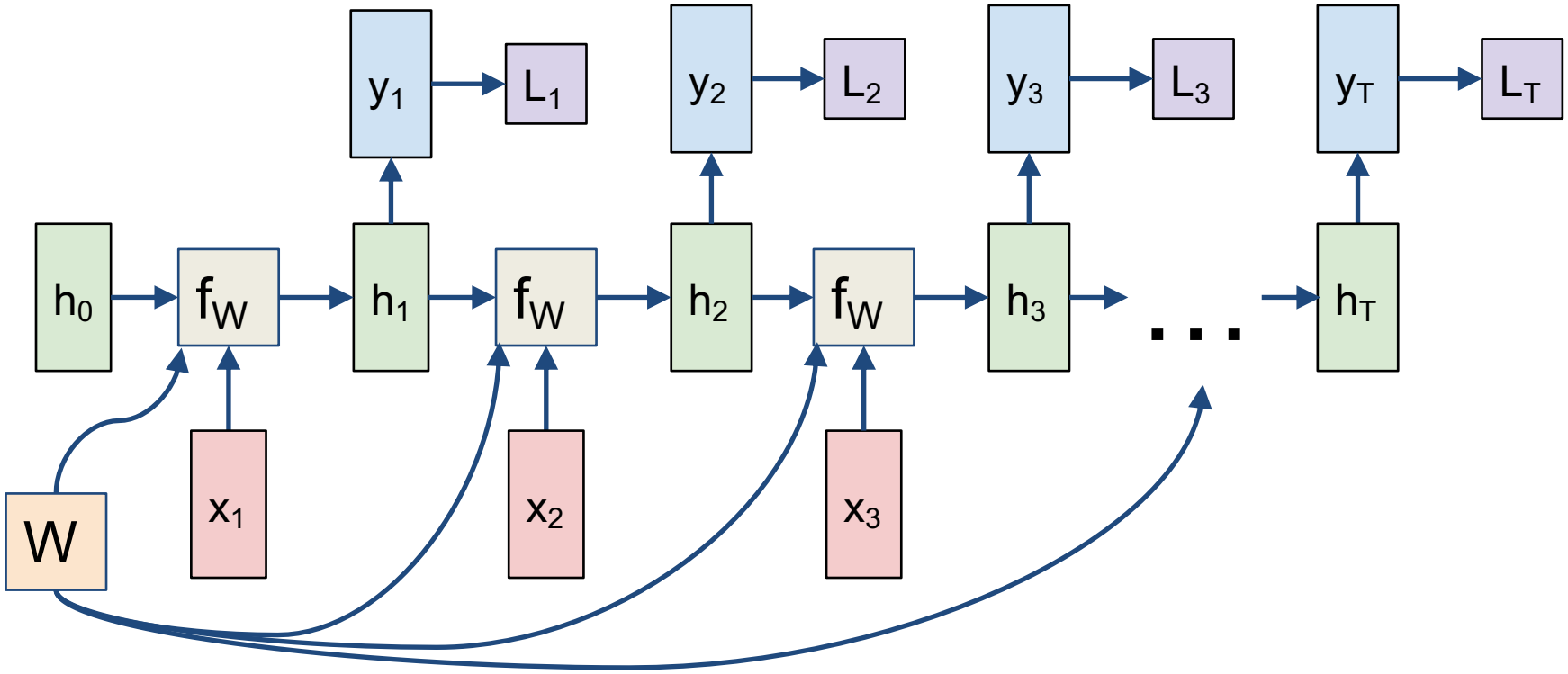


RNN: Computational Graph: Many to Many

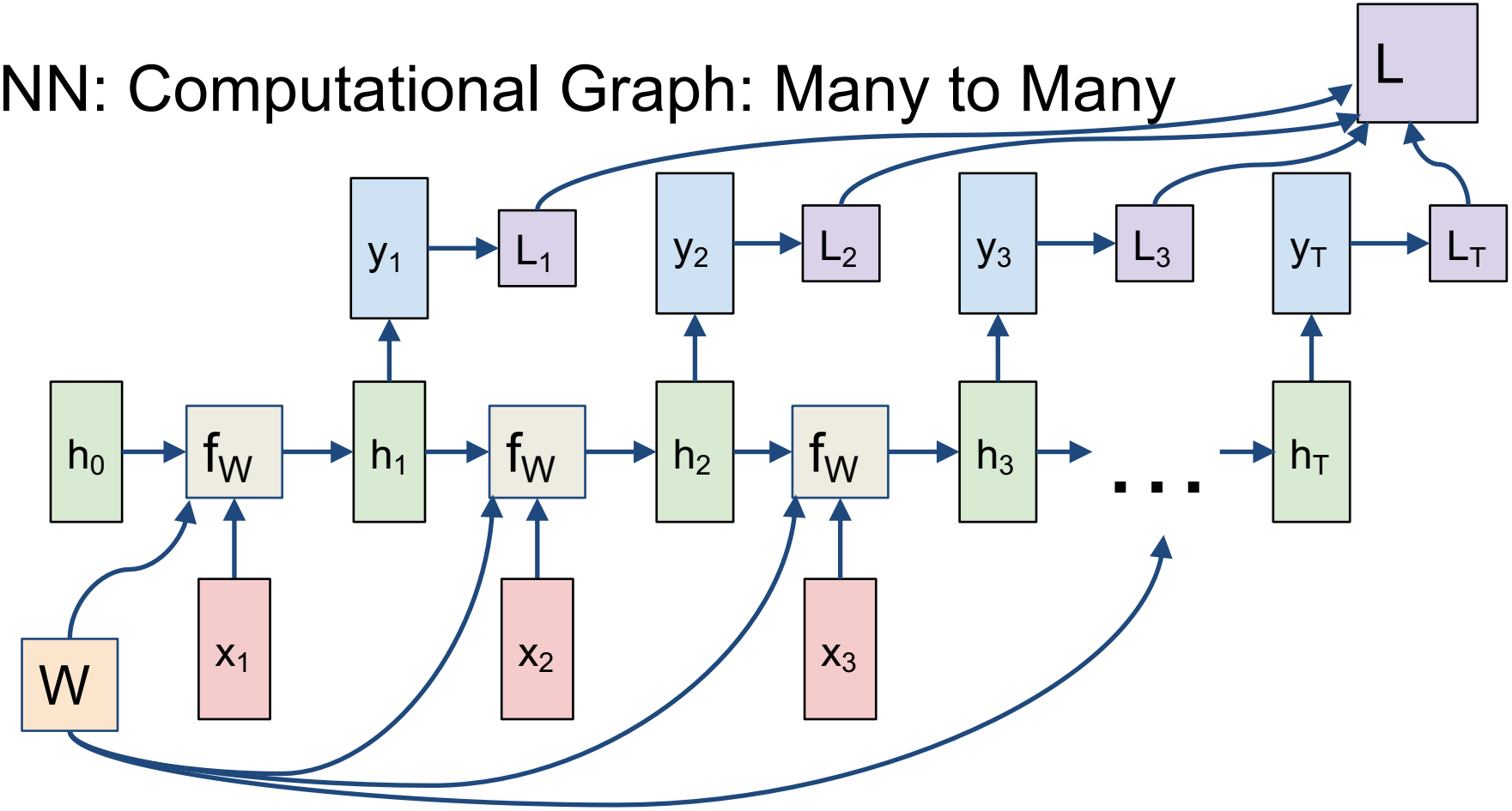


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

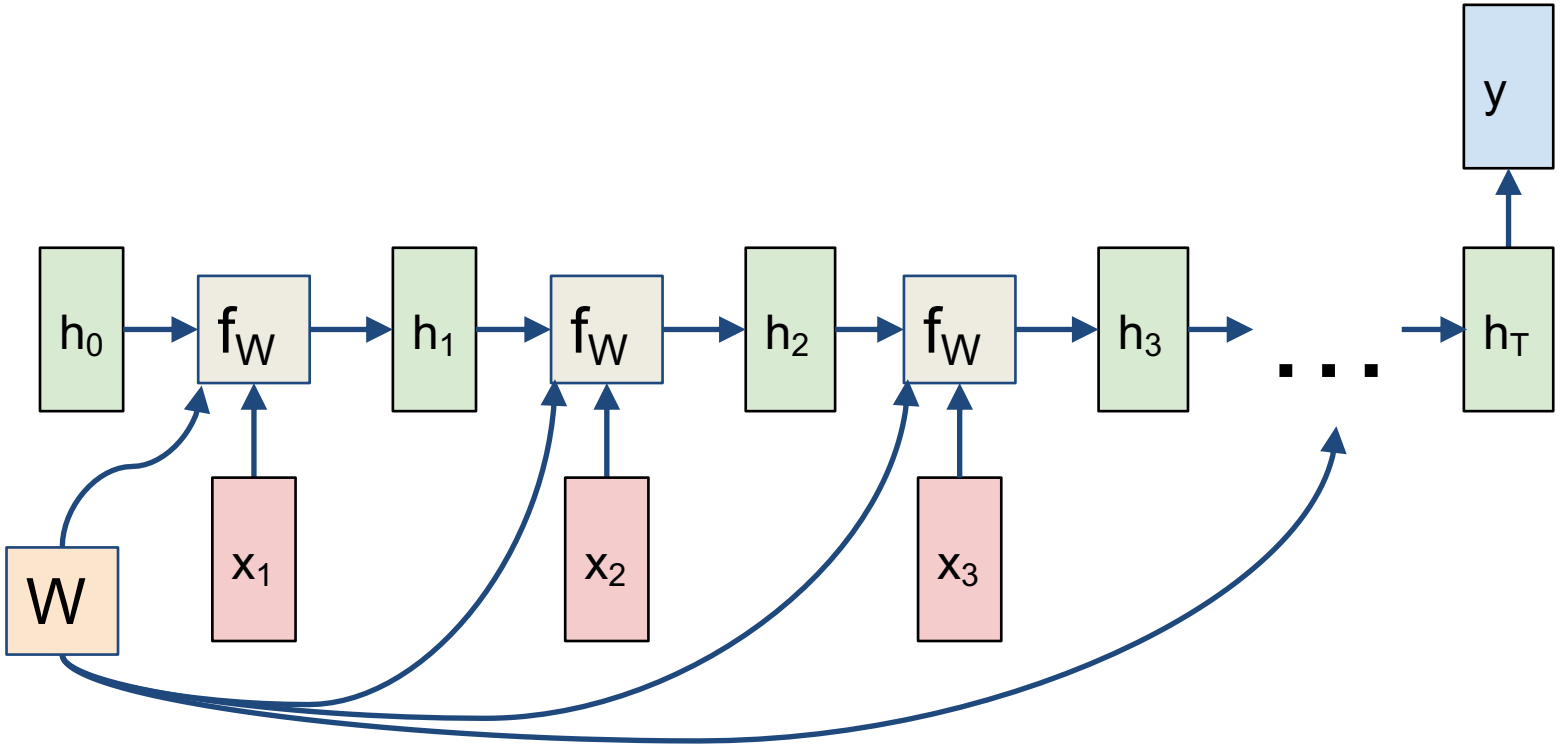
RNN: Computational Graph: Many to Many



RNN: Computational Graph: Many to Many

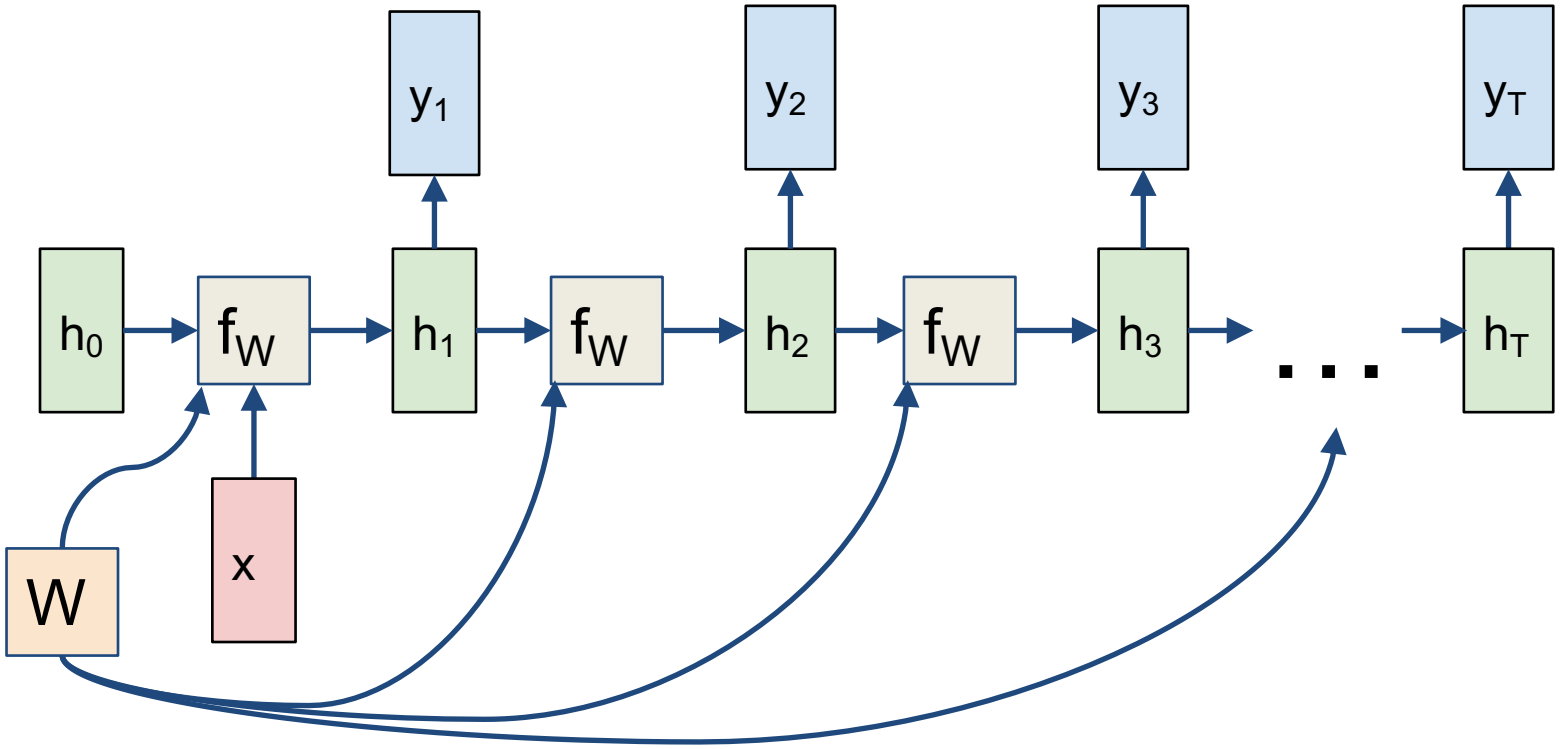


RNN: Computational Graph: Many to One



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

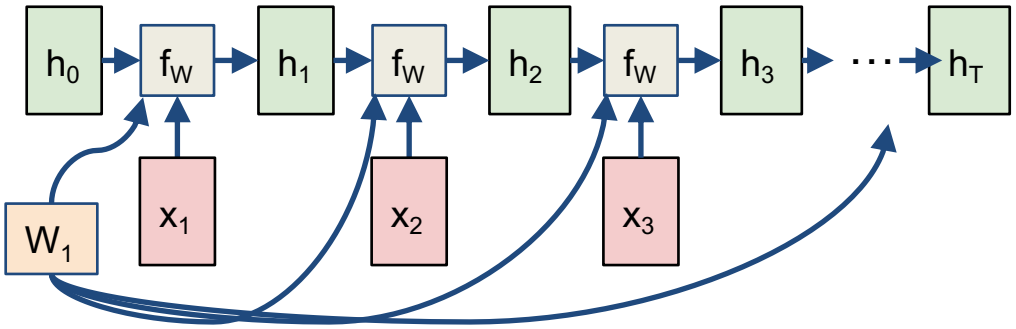
RNN: Computational Graph: One to Many



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Sequence to Sequence: Many-to-one + one-to-many

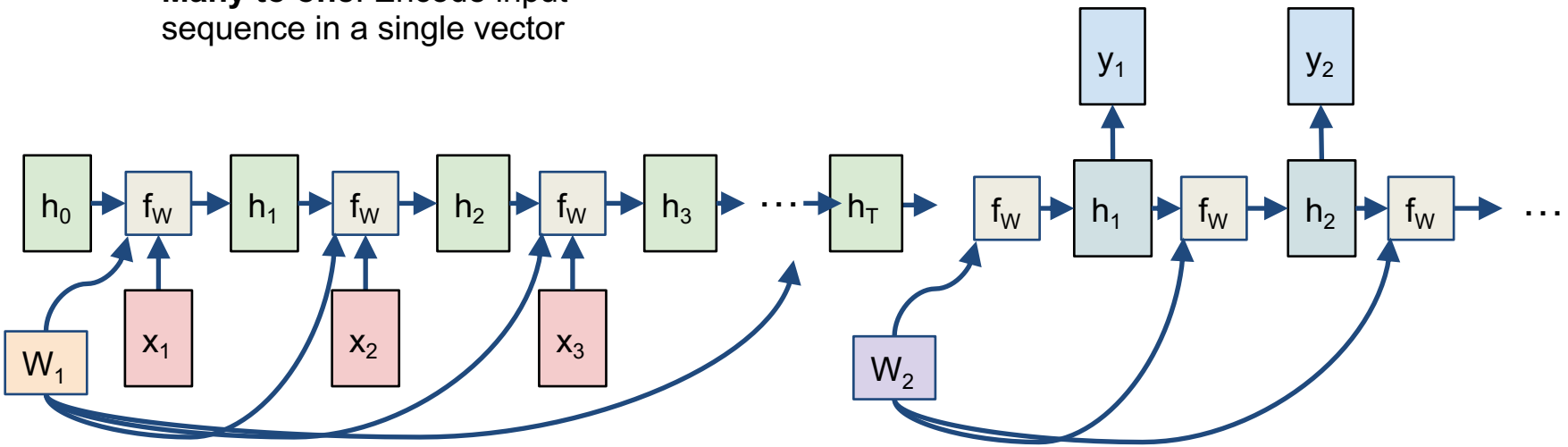
Many to one: Encode input sequence in a single vector



Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

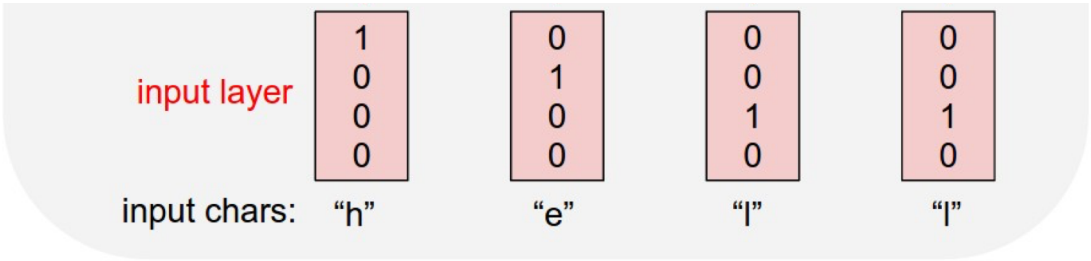
One to many: Produce output sequence from single input vector



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

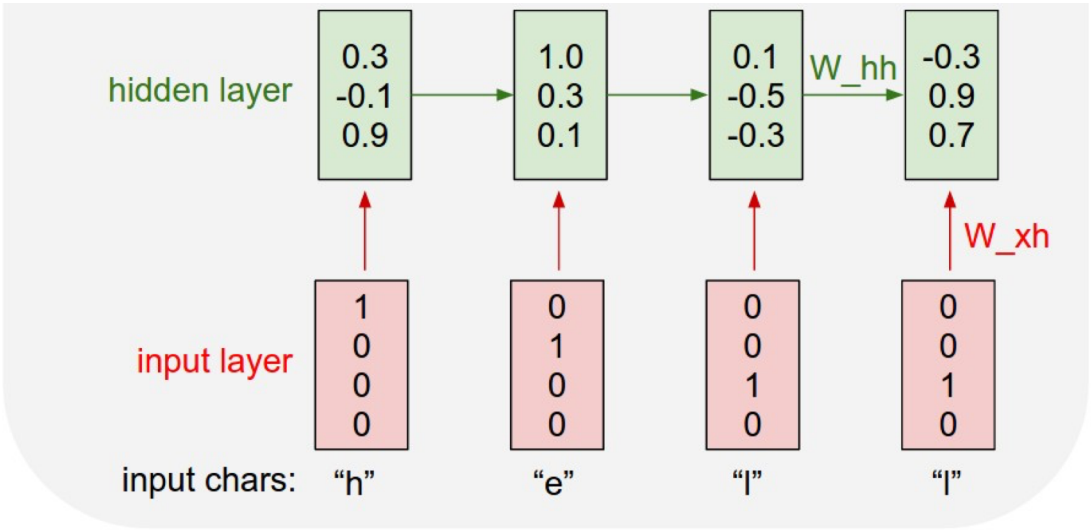


Example: Character-level Language Model

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

Vocabulary:
[h,e,l,o]

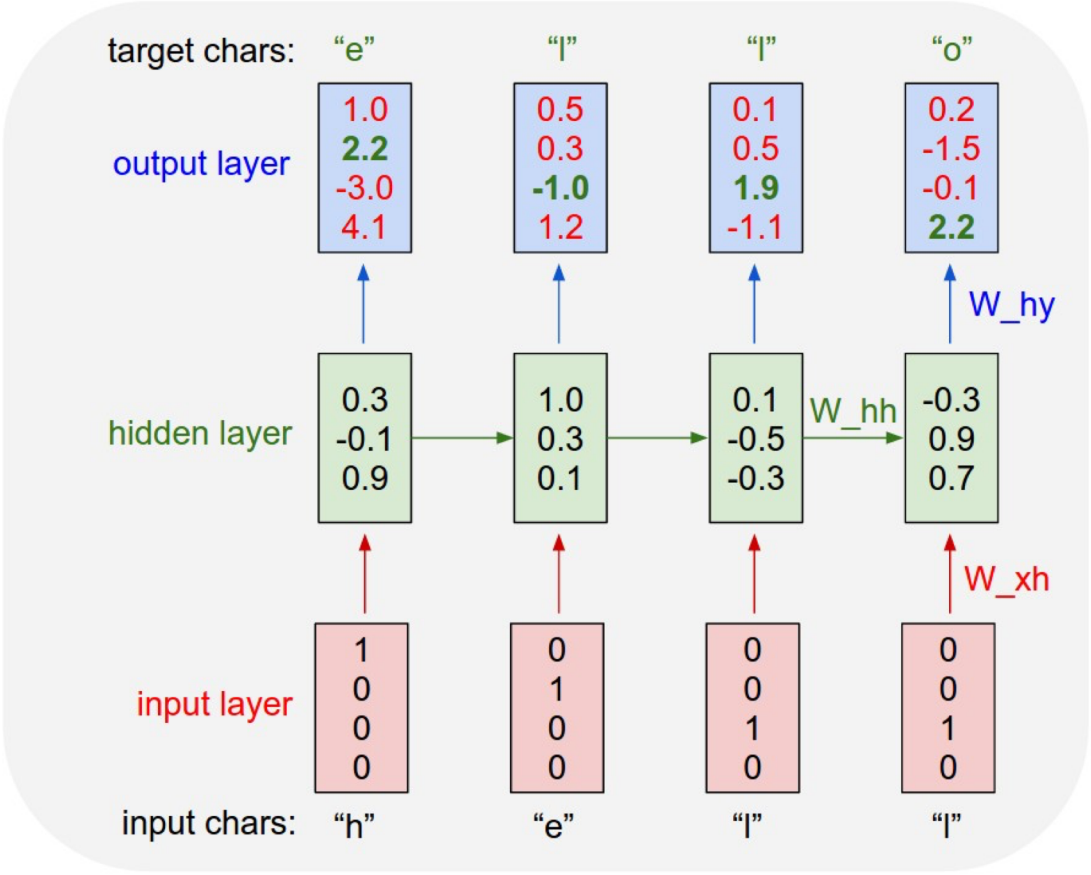
Example training sequence:
“hello”



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

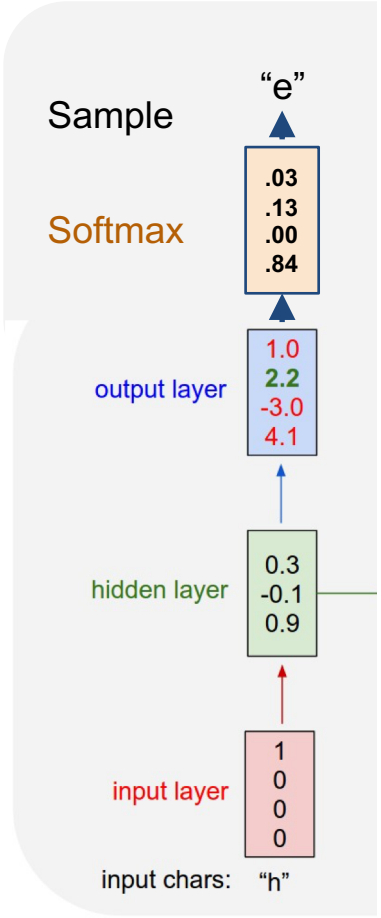
Example training sequence:
“hello”



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

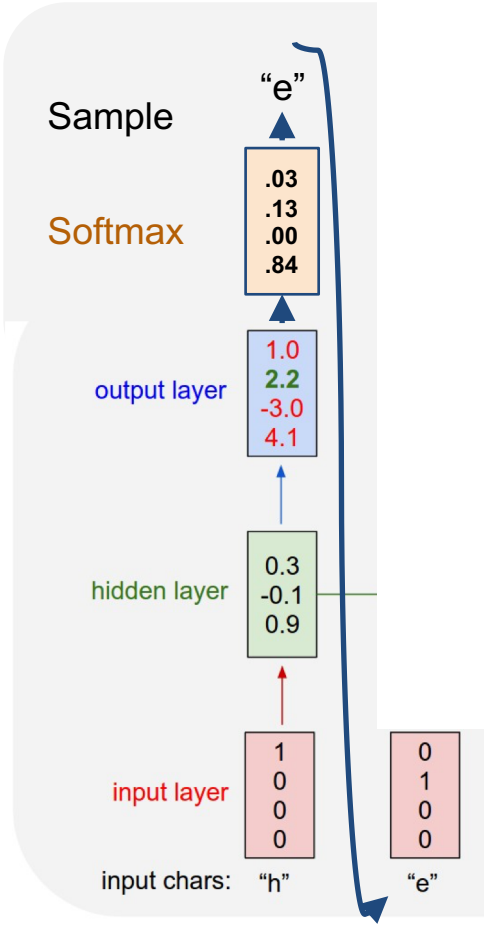
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

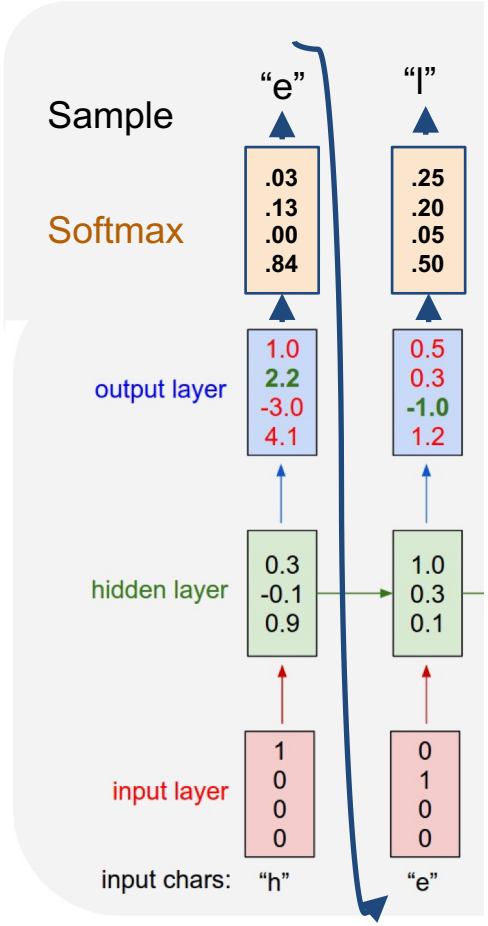
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

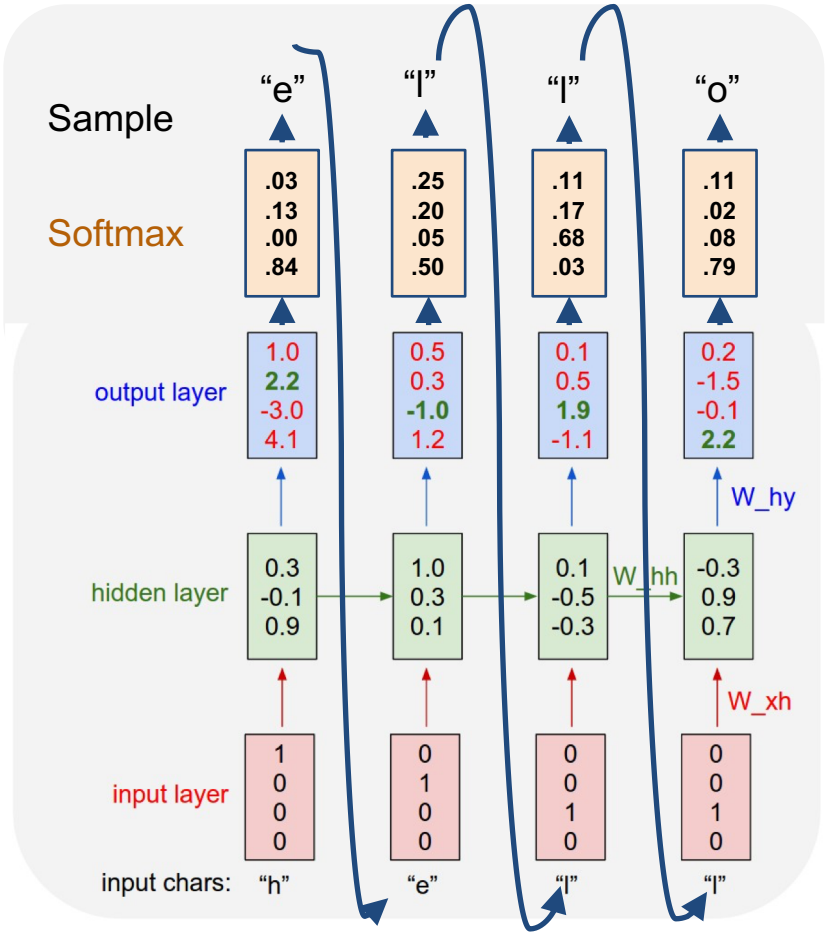
At test-time sample characters one at a time, feed back to model



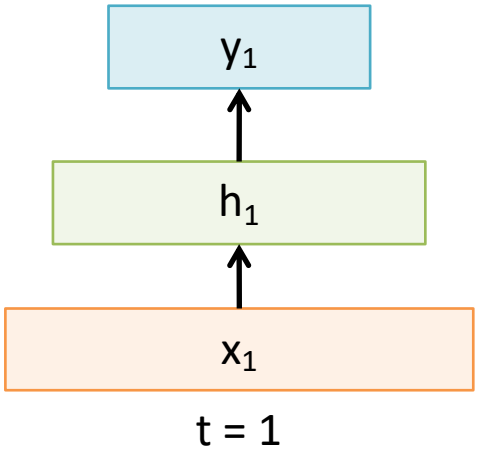
Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

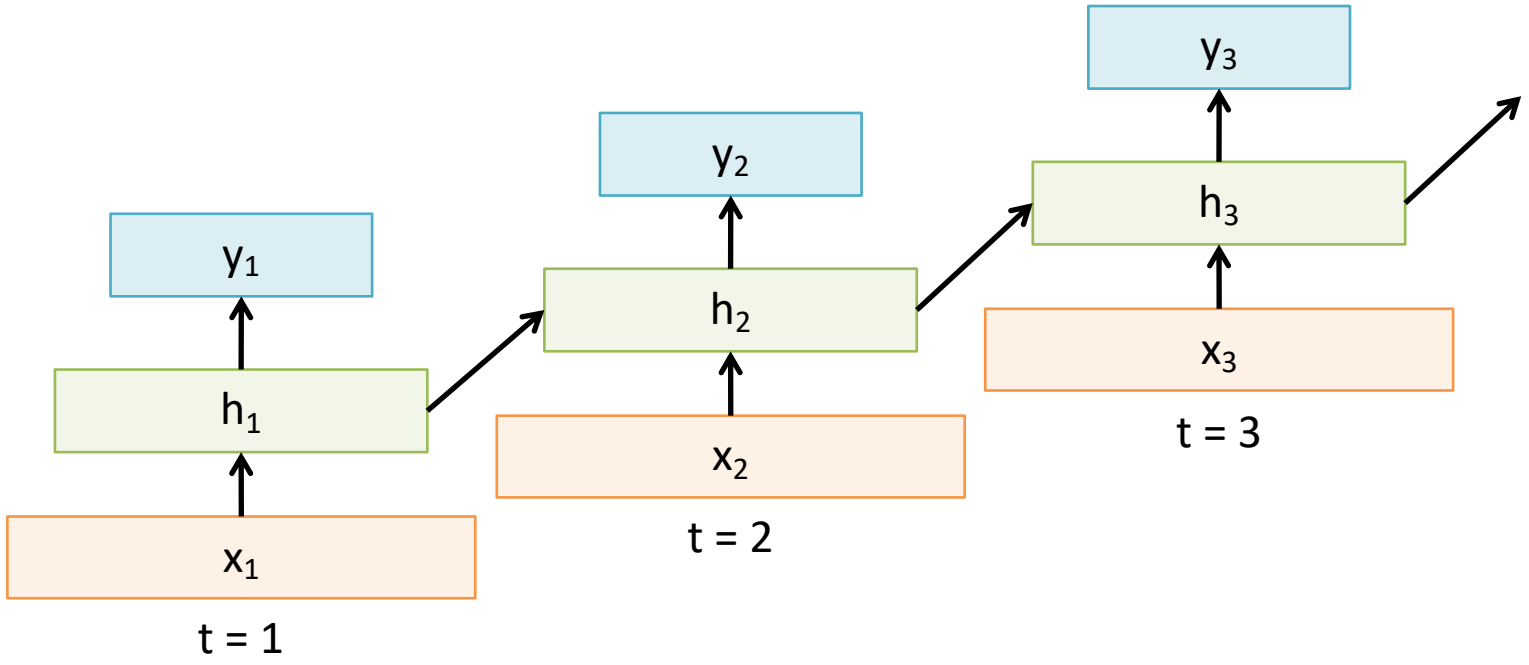
At test-time sample characters one at a time, feed back to model



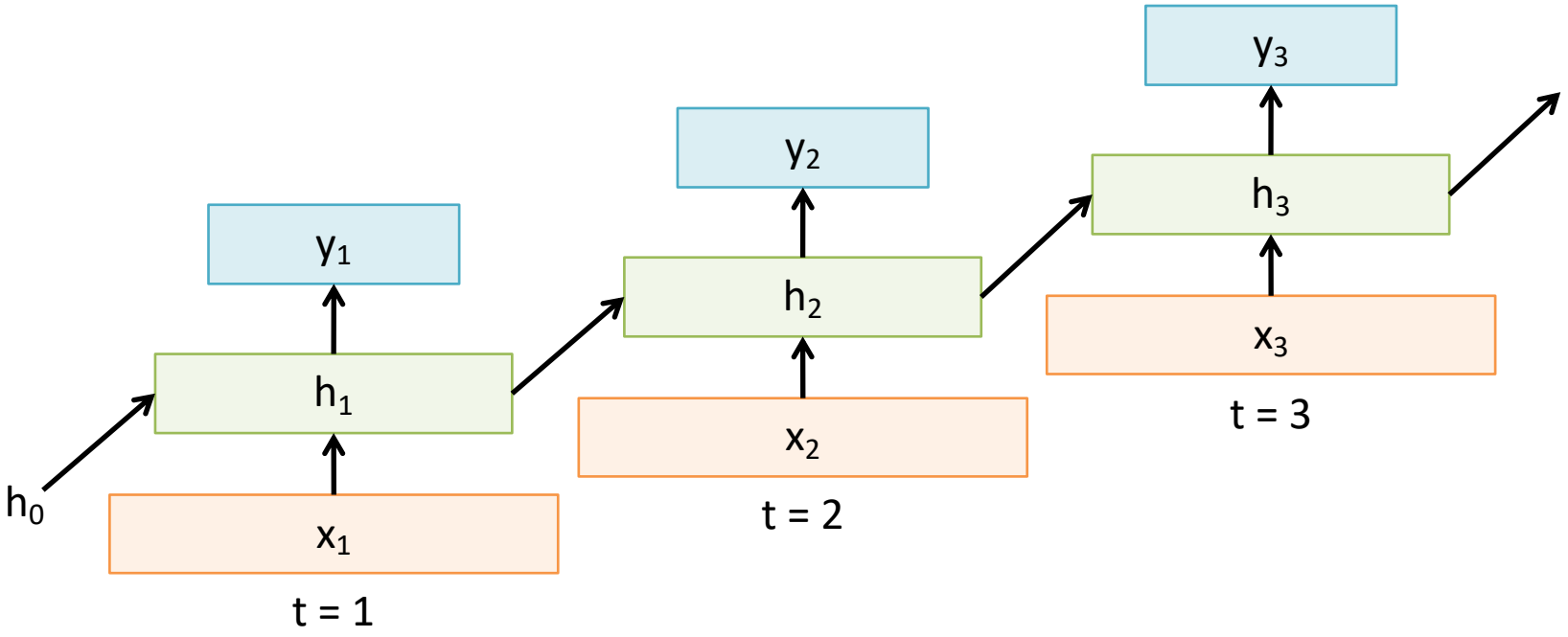
Basic Feed-forward Network



Basic RNN



Basic RNN



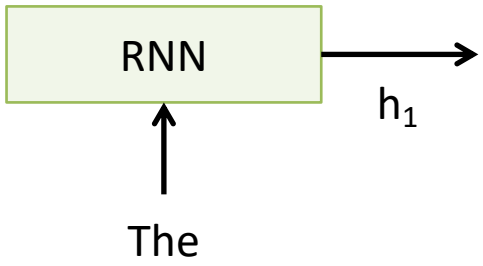
Sentiment Classification

- Classify a
  restaurant review from Yelp! OR
  movie review from IMDB OR
  …
  as positive or negative

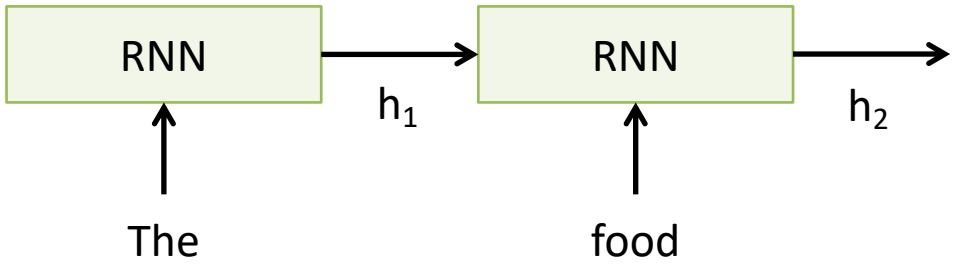
- Inputs: Multiple words, one or more sentences
- Outputs: Positive / Negative classification

- “The food was really good”
- “The chicken crossed the road because it was uncooked”

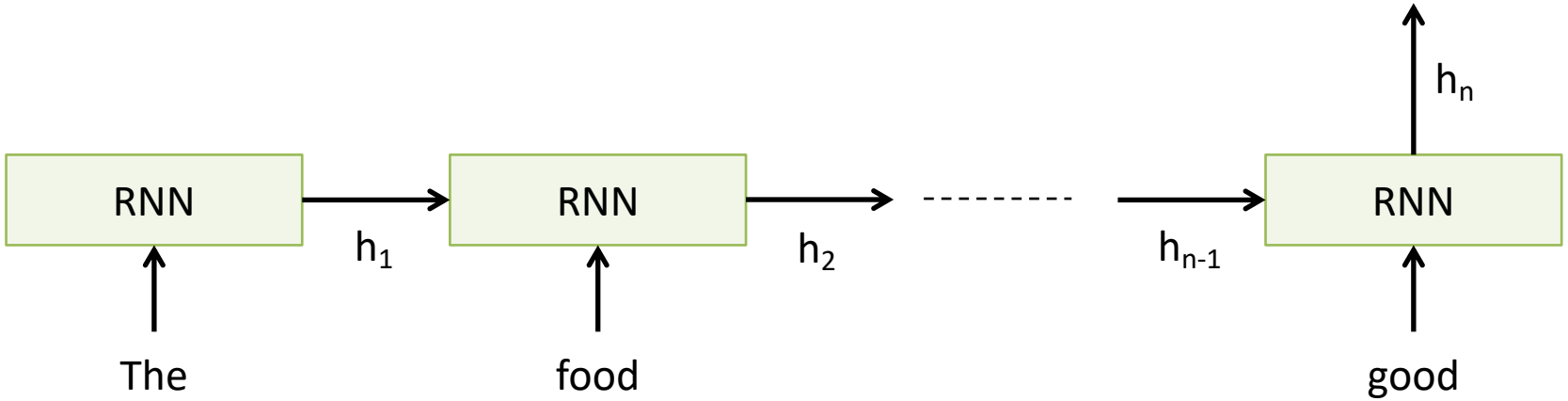
Sentiment Classification



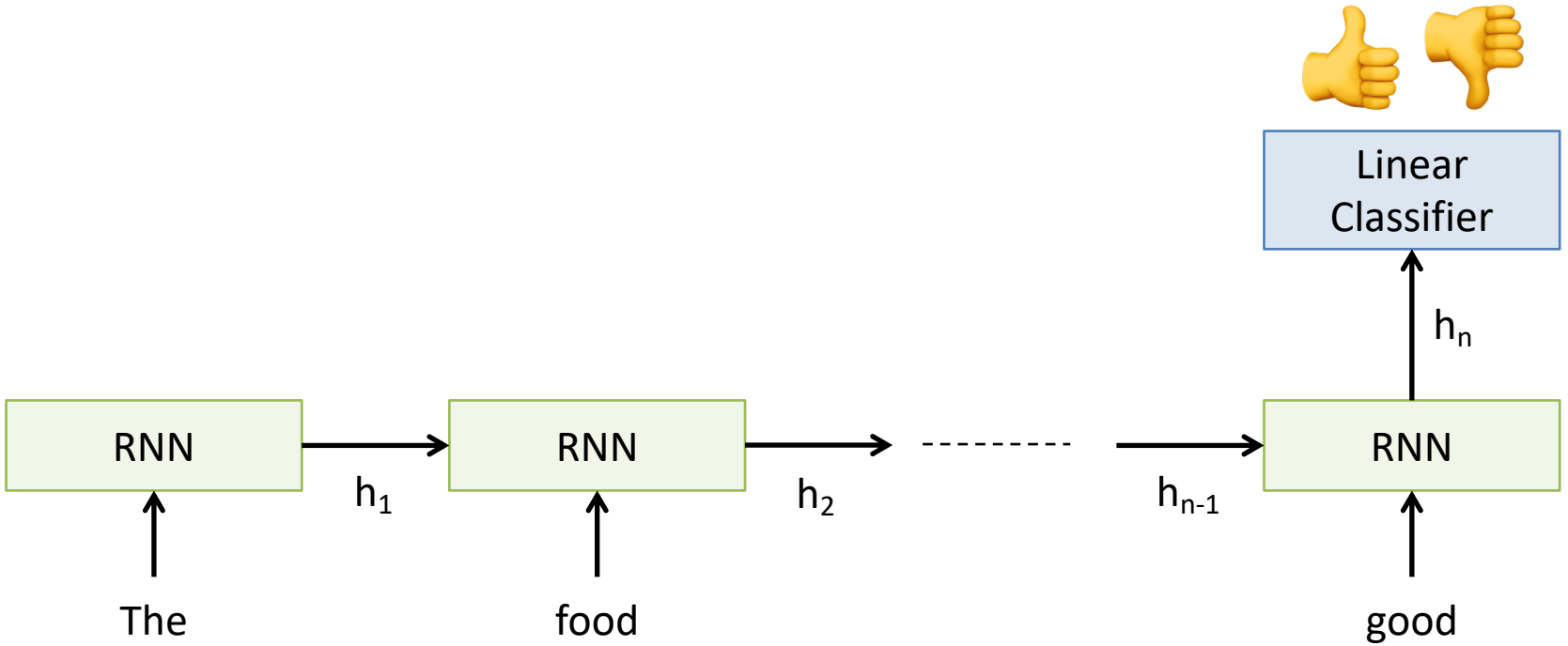
Sentiment Classification



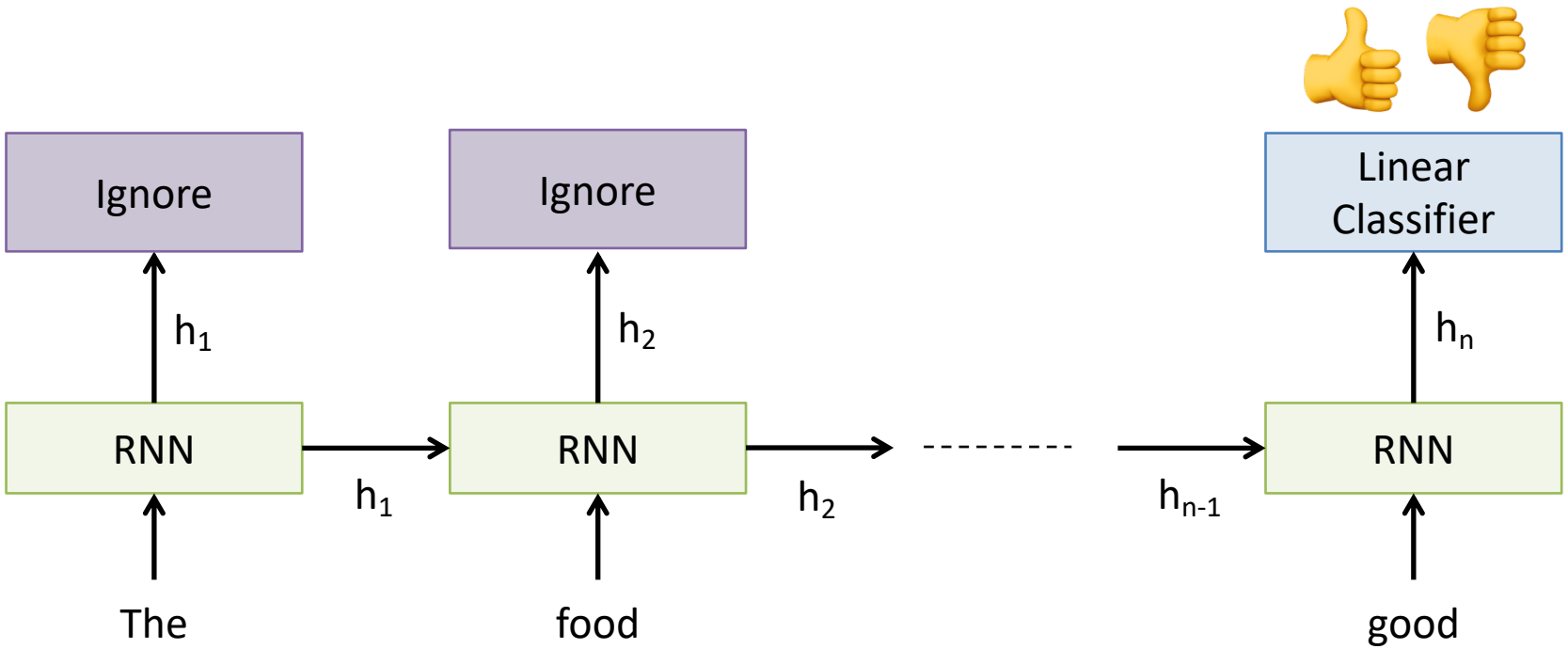
Sentiment Classification



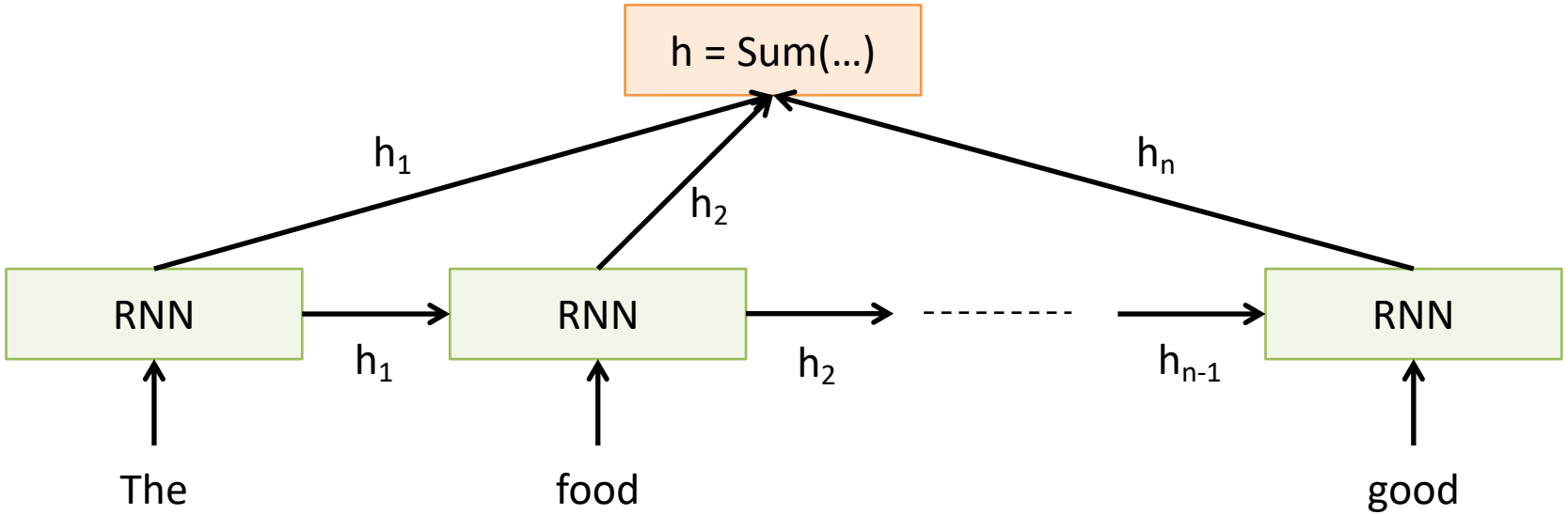
Sentiment Classification



Sentiment Classification

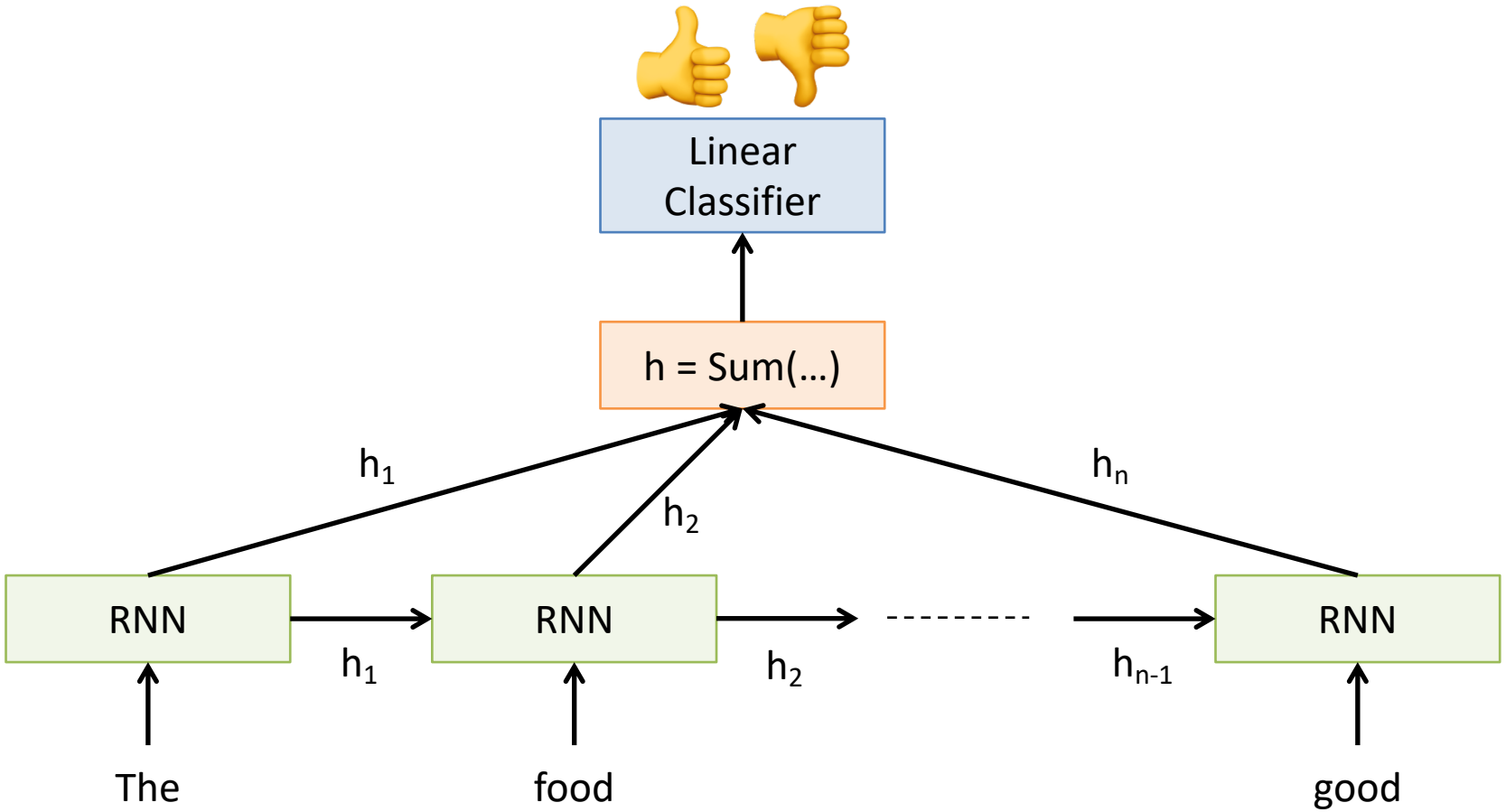


Sentiment Classification



http://deeplearning.net/tutorial/lstm.html

Sentiment Classification

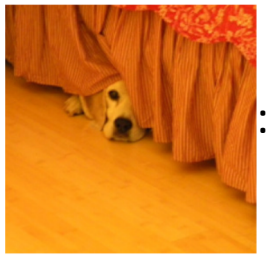


http://deeplearning.net/tutorial/lstm.html

Image Captioning

- Given an image, produce a sentence describing its contents

- Inputs: Image feature (from a CNN)
- Outputs: Multiple words (let's consider one sentence)



: The dog is hiding

Image Captioning

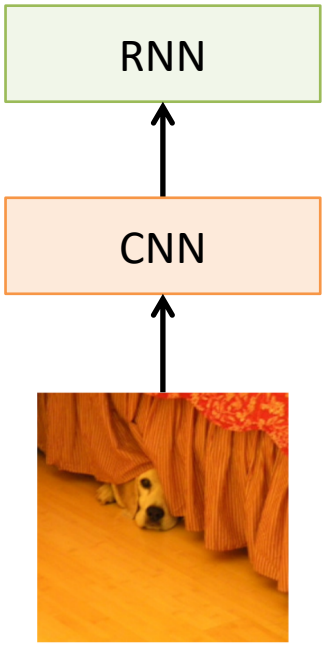


Image Captioning

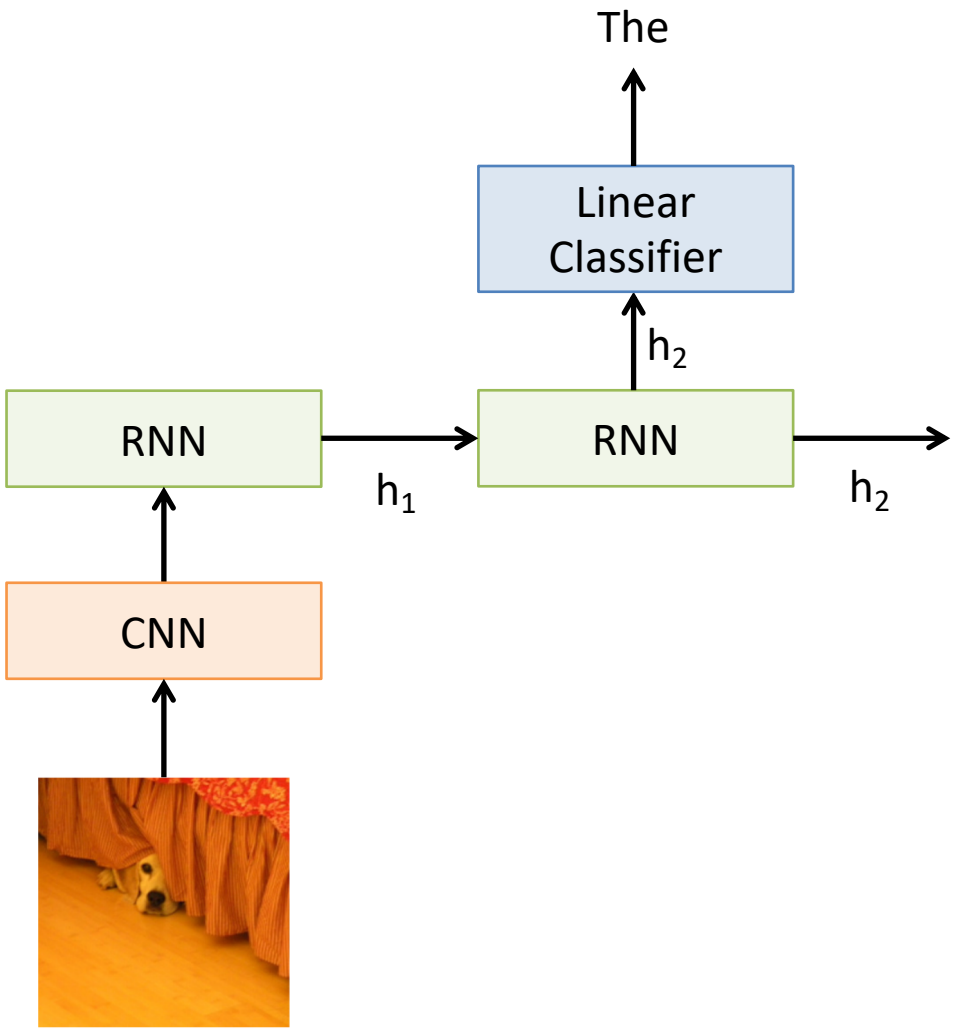
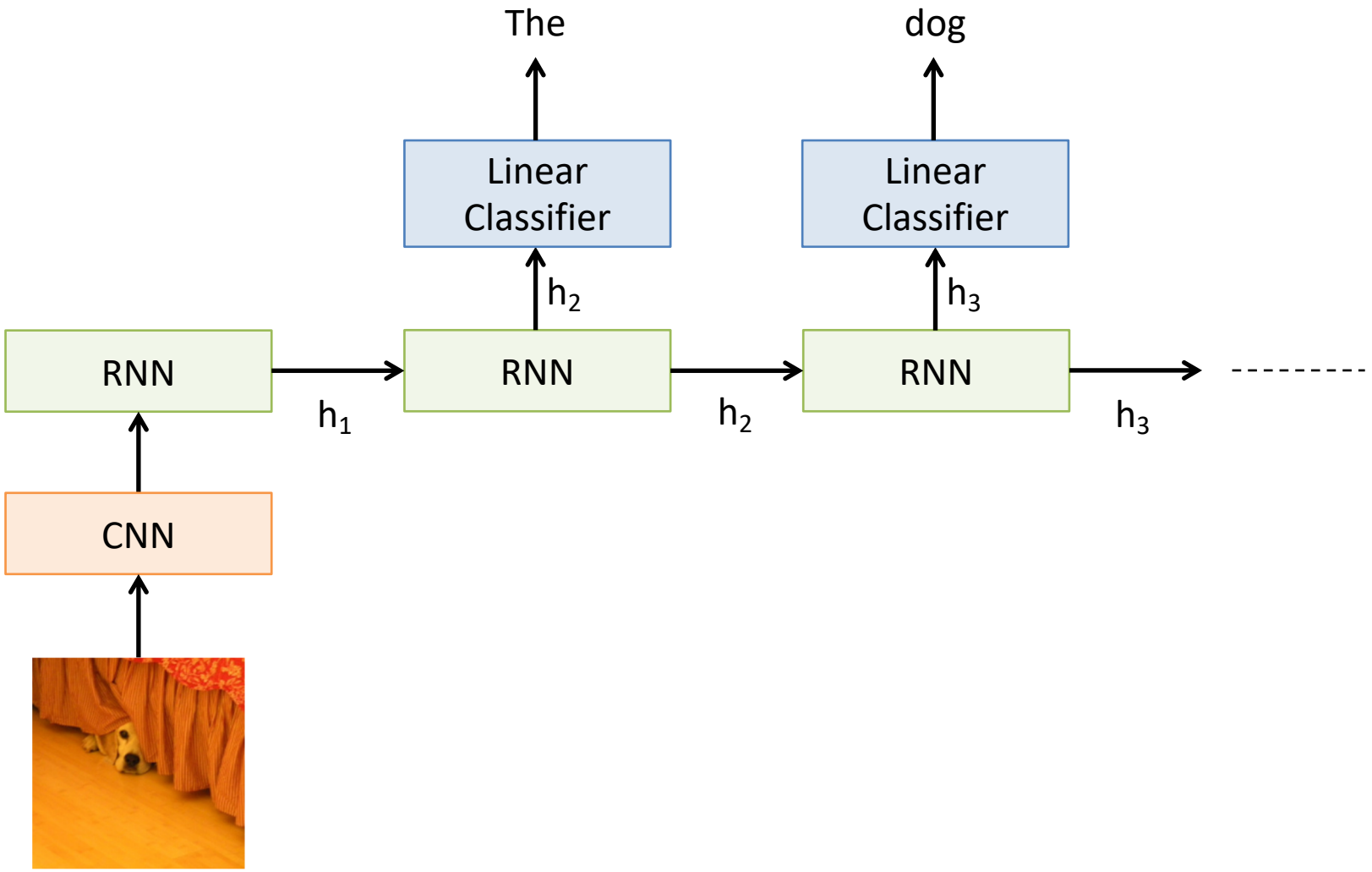


Image Captioning



RNN Outputs: Image Captions

A person riding a motorcycle on a dirt road.

Two dogs play in the grass.

A herd of elephants walking across a dry grass field.

A group of young people playing a game of frisbee.

Two hockey players are fighting over the puck.

A close up of a cat laying on a couch.

[Show and Tell: A Neural Image Caption Generator, CVPR 15]

RNN Outputs: Language Modeling

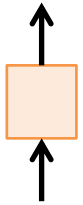
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the
courtesy of your law,
Your sight and several breath, will
wear the gods
With his heads, and my hands are
wonder'd at the deeds,
So drop upon your lordship's head,
and your opinion
Shall be against your honour.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Input – Output Scenarios

Single - Single



Feed-forward Network

Single - Multiple

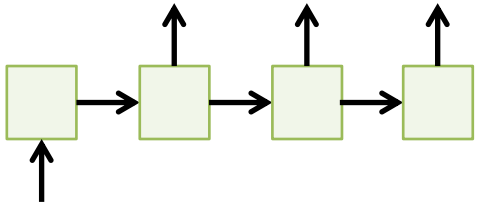
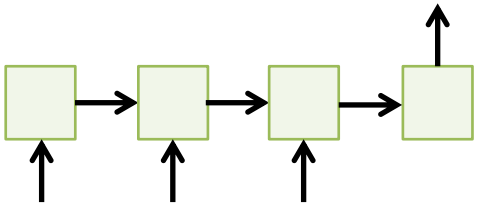


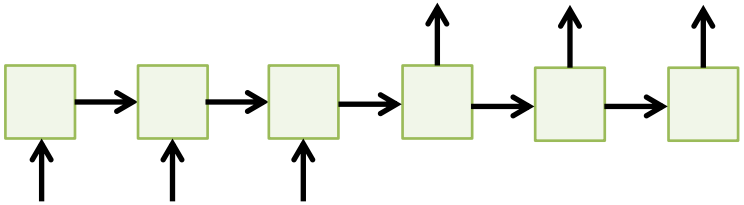
Image Captioning

Multiple - Single

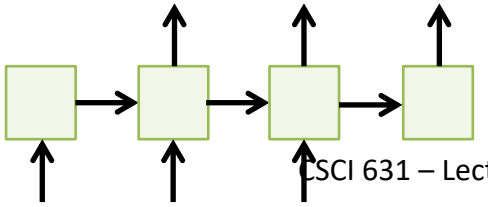


Sentiment Classification

Multiple - Multiple



Translation

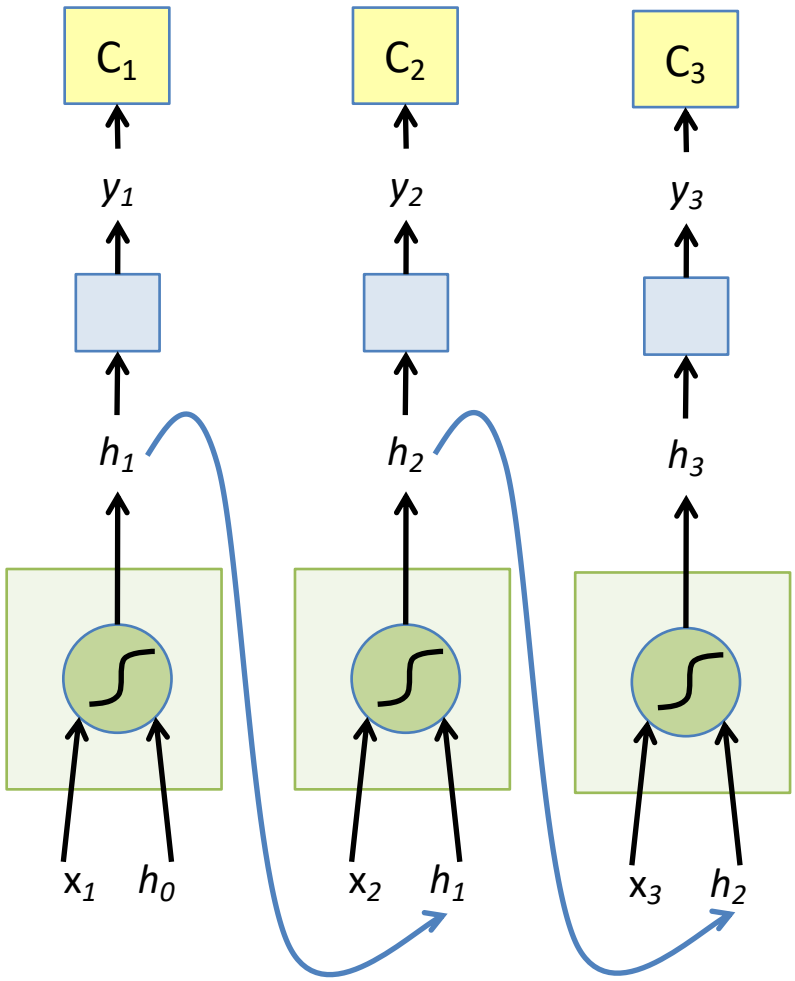


Image/Video Captioning

Input – Output Scenarios

- Note: We might deliberately choose to frame our problem as a particular input-output scenario for ease of training or better performance.
  - For example, at each time step, provide previous word as input for image captioning (Single-Multiple to Multiple-Multiple)

The Vanilla RNN Forward



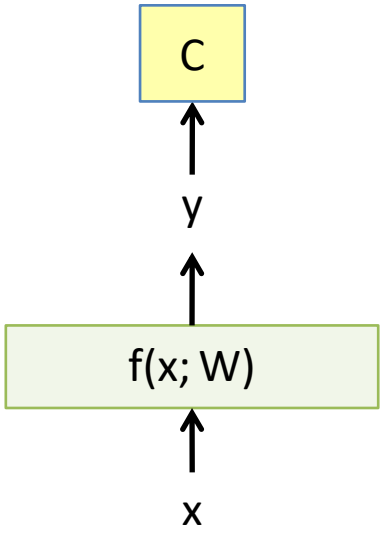
$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

"Unfold" network through time by making copies at each time-step

BackPropagation Refresher



$$y = f(x; W)$$

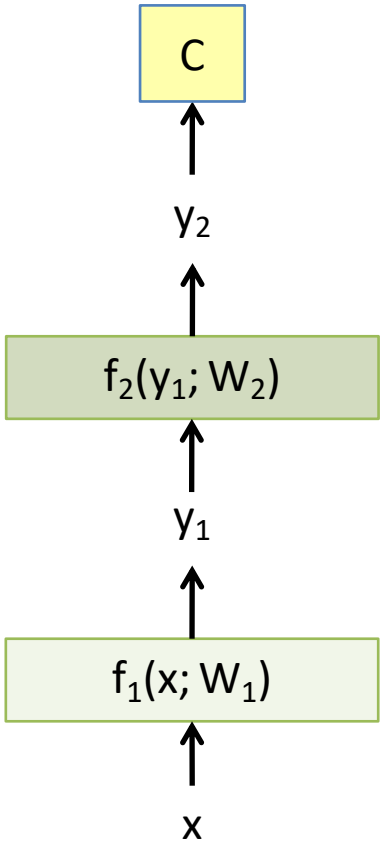
$$C = \text{Loss}(y, y_{GT})$$

SGD Update

$$W \leftarrow W - \eta \frac{\partial C}{\partial W}$$

$$\frac{\partial C}{\partial W} = \left(\frac{\partial C}{\partial y} \right) \left(\frac{\partial y}{\partial W} \right)$$

Multiple Layers



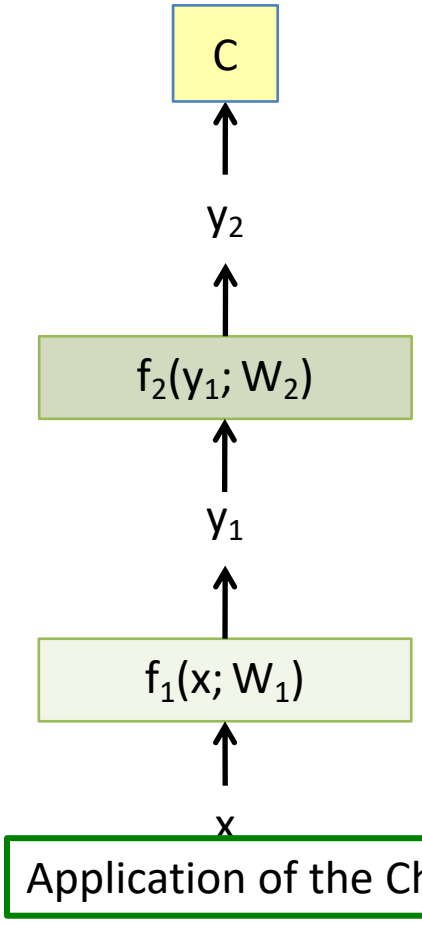
$$y_1 = f_1(x; W_1)$$
$$y_2 = f_2(y_1; W_2)$$
$$C = \text{Loss}(y_2, y_{GT})$$

SGD Update

$$W_2 \leftarrow W_2 - \eta \frac{\partial C}{\partial W_2}$$

$$W_1 \leftarrow W_1 - \eta \frac{\partial C}{\partial W_1}$$

Chain Rule for Gradient Computation



$$y_1 = f_1(x; W_1)$$

$$y_2 = f_2(y_1; W_2)$$

$$C = \text{Loss}(y_2, y_{GT})$$

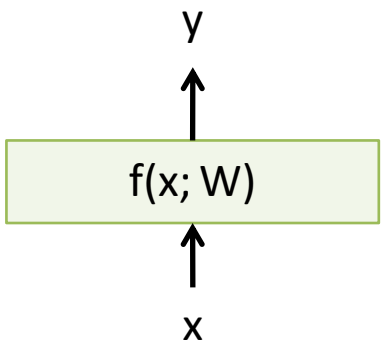
$$\text{Find } \frac{\partial C}{\partial W_1}, \frac{\partial C}{\partial W_2}$$

$$\frac{\partial C}{\partial W_2} = \left(\frac{\partial C}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial W_2}\right)$$

$$\frac{\partial C}{\partial W_1} = \left(\frac{\partial C}{\partial y_1}\right)\left(\frac{\partial y_1}{\partial W_1}\right)$$

$$= \left(\frac{\partial C}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial y_1}\right)\left(\frac{\partial y_1}{\partial W_1}\right)$$

Chain Rule for Gradient Computation



Given: $\left(\frac{\partial C}{\partial y}\right)$

We are interested in computing: $\left(\frac{\partial C}{\partial W}\right), \left(\frac{\partial C}{\partial x}\right)$

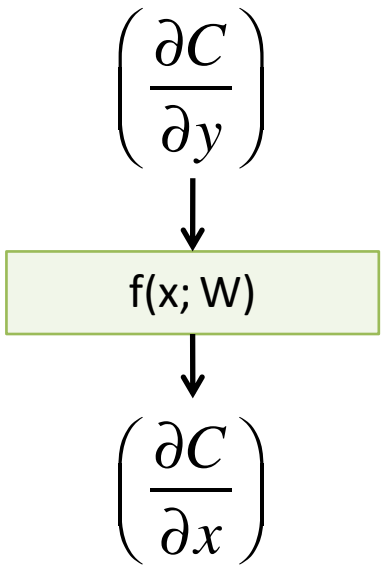
Intrinsic to the layer are:

$\left(\frac{\partial y}{\partial W}\right)$ – How does output change due to params

$\left(\frac{\partial y}{\partial x}\right)$ – How does output change due to inputs

$$\left(\frac{\partial C}{\partial W}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial W}\right) \quad \left(\frac{\partial C}{\partial x}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial x}\right)$$

Chain Rule for Gradient Computation



Given: $\left(\frac{\partial C}{\partial y}\right)$

We are interested in computing: $\left(\frac{\partial C}{\partial W}\right), \left(\frac{\partial C}{\partial x}\right)$

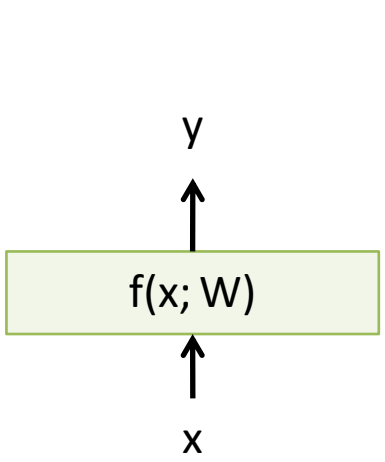
Intrinsic to the layer are:

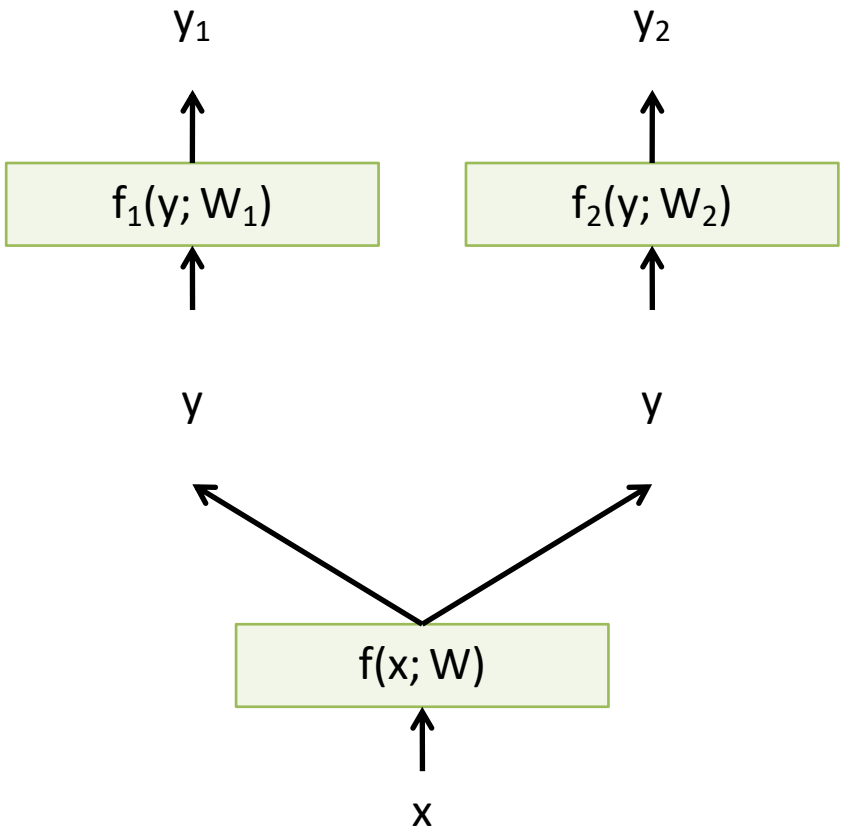
$\left(\frac{\partial y}{\partial W}\right)$ – How does output change due to params

$\left(\frac{\partial y}{\partial x}\right)$ – How does output change due to inputs

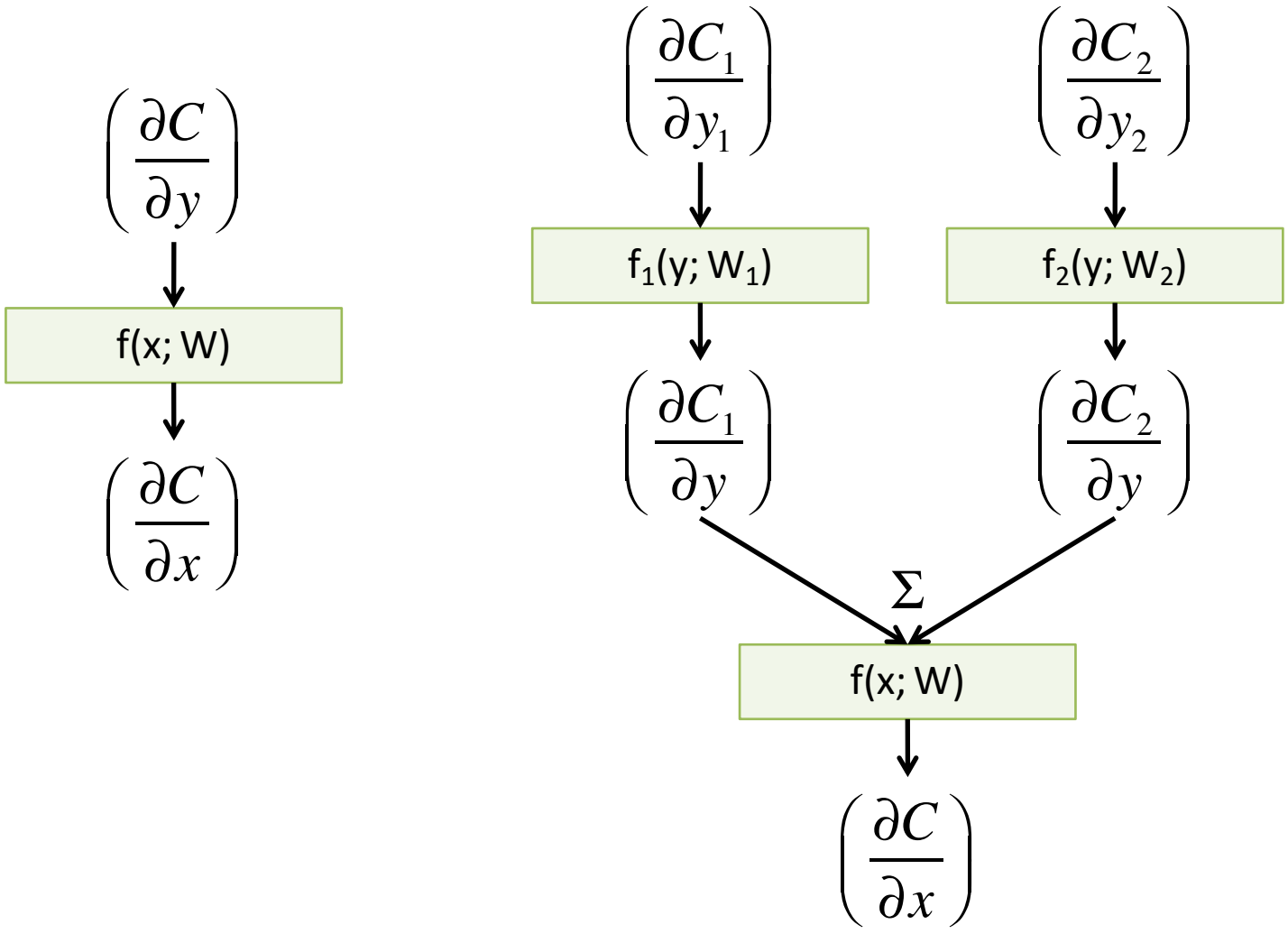
$$\left(\frac{\partial C}{\partial W}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial W}\right) \quad \left(\frac{\partial C}{\partial x}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial x}\right)$$

Extension to Computational Graphs

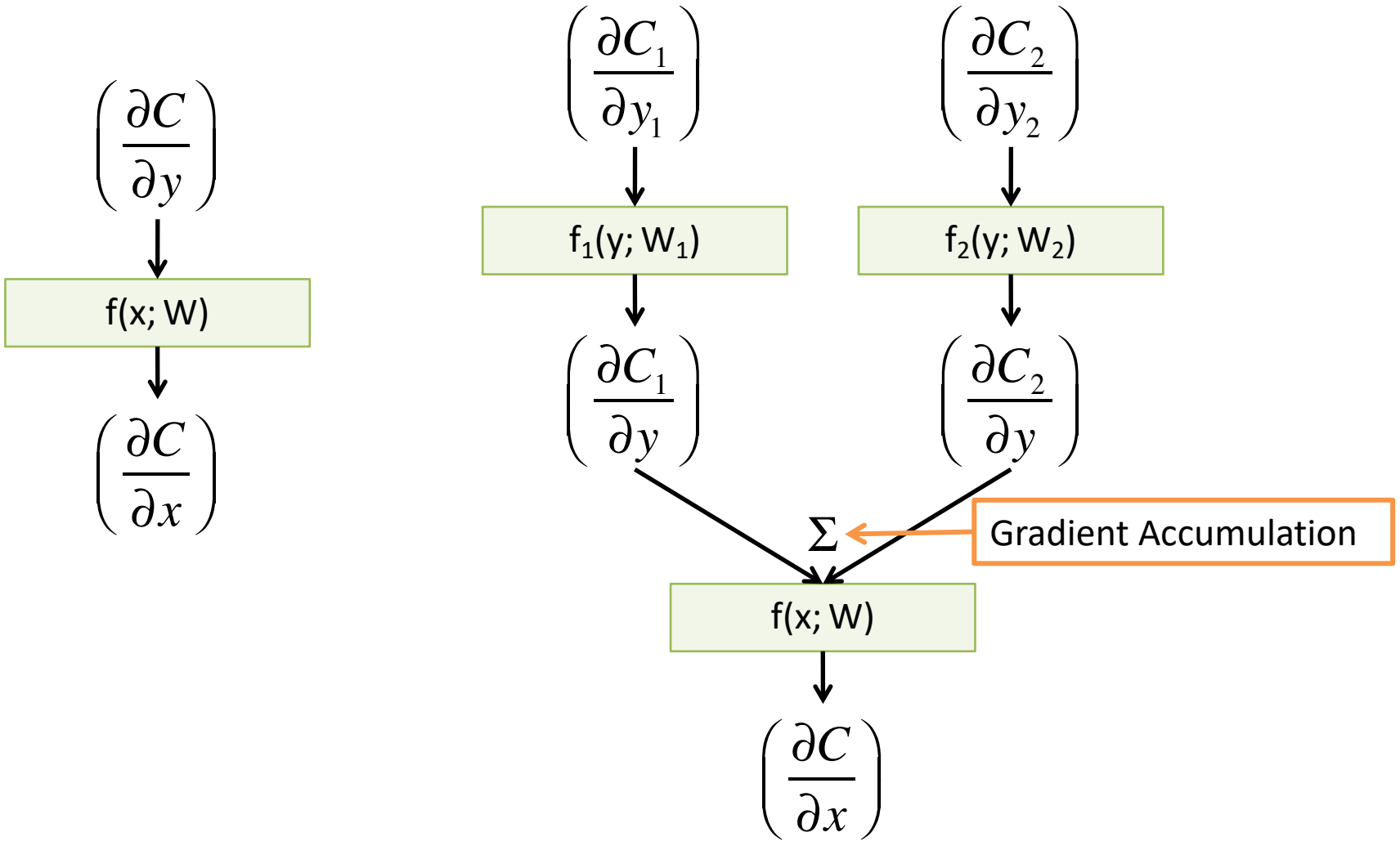




Extension to Computational Graphs



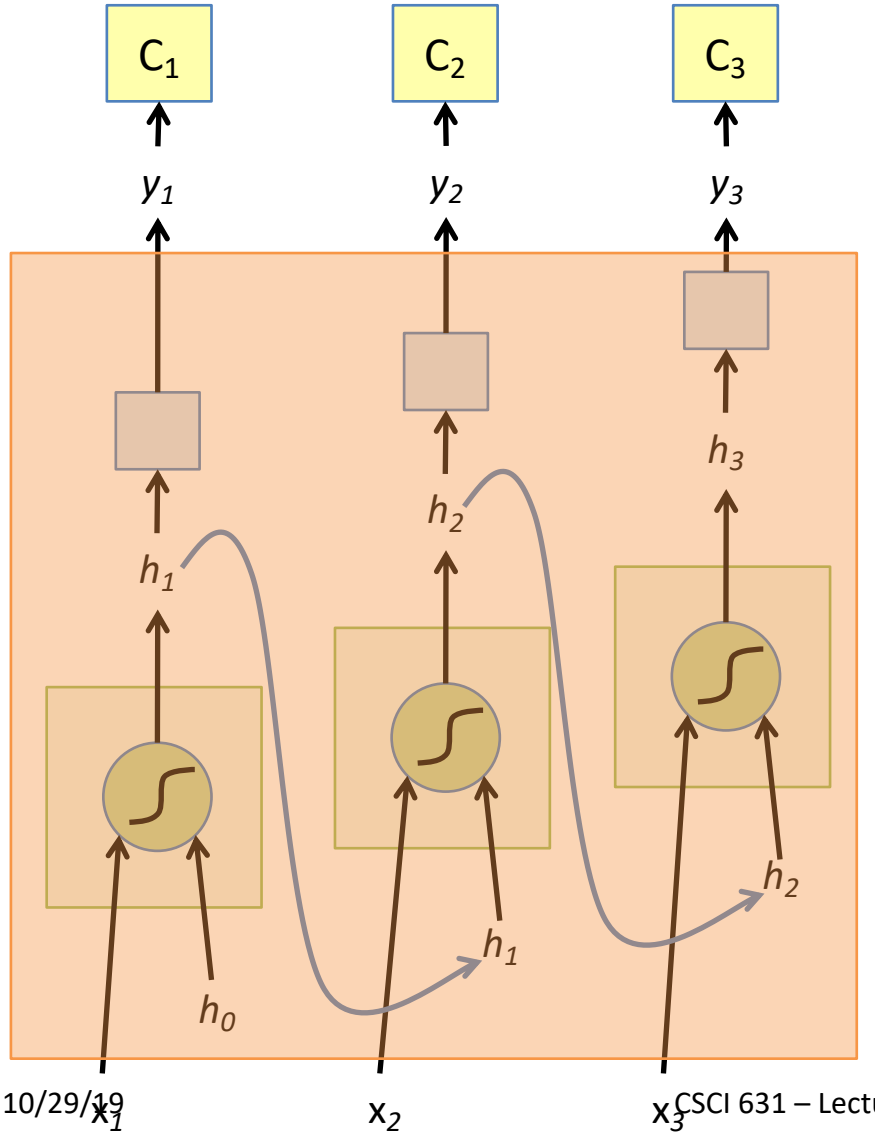
Extension to Computational Graphs



BackPropagation Through Time (BPTT)

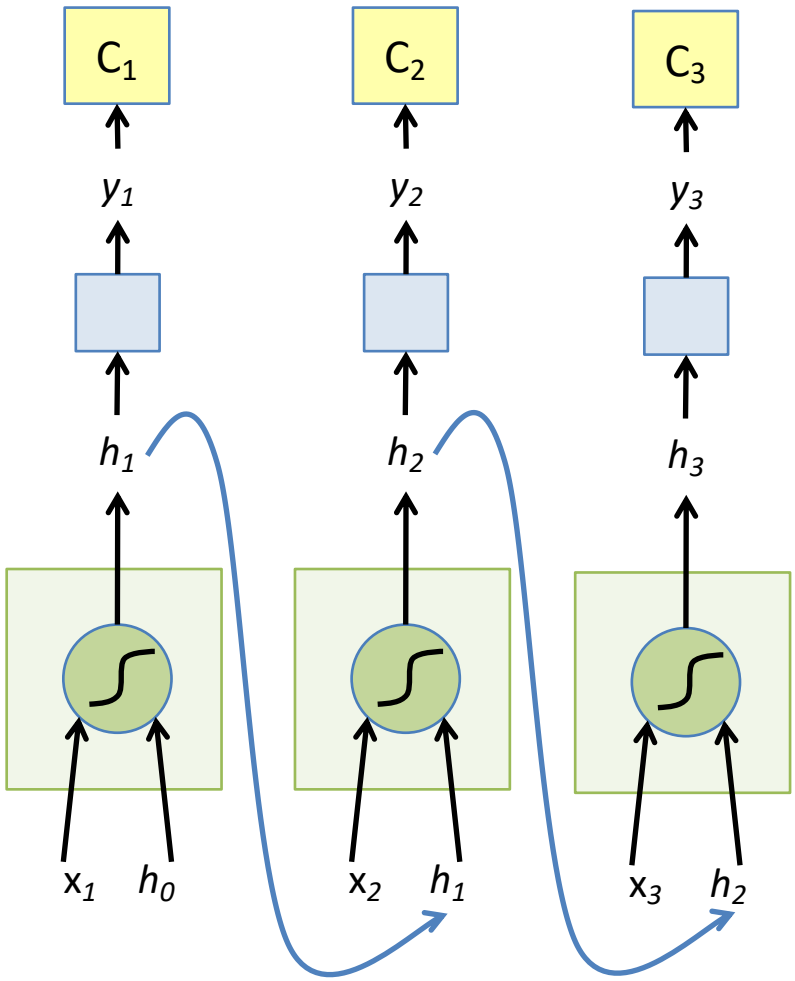
- One of the methods used to train RNNs
  - The unfolded network (used during forward pass) is treated as one big feed-forward network
  - This unfolded network accepts the whole time series as input
  - The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and then applied to the RNN weights

The Unfolded Vanilla RNN

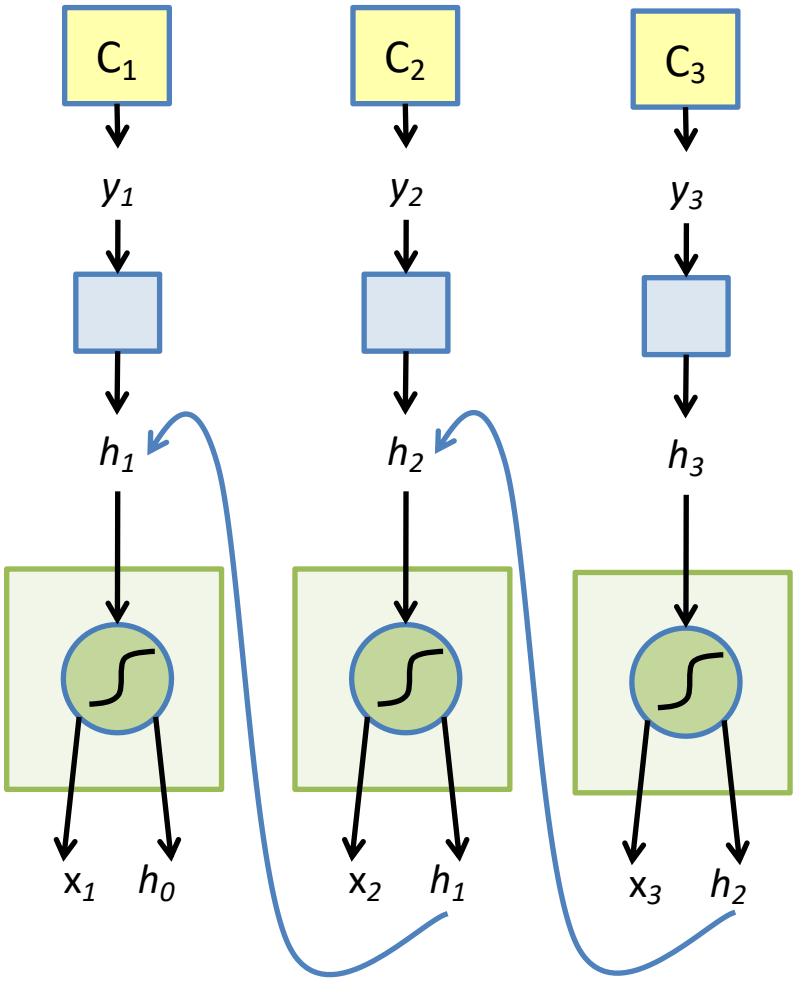


- Treat the unfolded network as one big feed-forward network!
- This big network takes in entire sequence as an input
- Compute gradients through the usual backpropagation
- Update shared weights

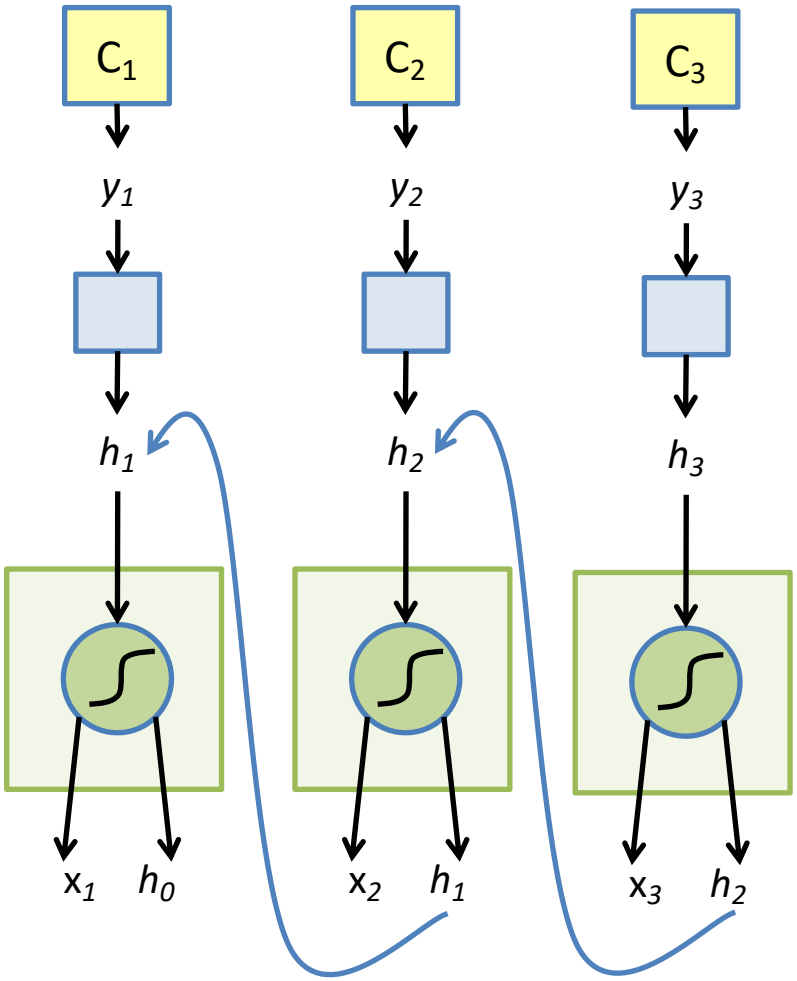
The Unfolded Vanilla RNN Forward



The Unfolded Vanilla RNN Backward



The Vanilla RNN Backward



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

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

$$\frac{\partial C_t}{\partial h_1} = \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_1}\right)$$

$$= \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_t}\right)\left(\frac{\partial h_t}{\partial h_{t-1}}\right)\cdots\left(\frac{\partial h_2}{\partial h_1}\right)$$

Issues with the Vanilla RNNs

- In the same way a product of k real numbers can shrink to zero or explode to infinity, so can a product of matrices

- It is sufficient for $\lambda_1 < 1/\gamma$, where λ_1 is the largest singular value of W, for the vanishing gradients problem to occur and it is necessary for exploding gradients that $\lambda_1 > 1/\gamma$, where $\gamma = 1$ for the tanh non-linearity and $\gamma = 1/4$ for the sigmoid non-linearity [1]

- Exploding gradients are often controlled with gradient element-wise or norm clipping

[1] On the difficulty of training recurrent neural networks, Pascanu *et al.*, 2013

Long Short-Term Memory (LSTM)[1]

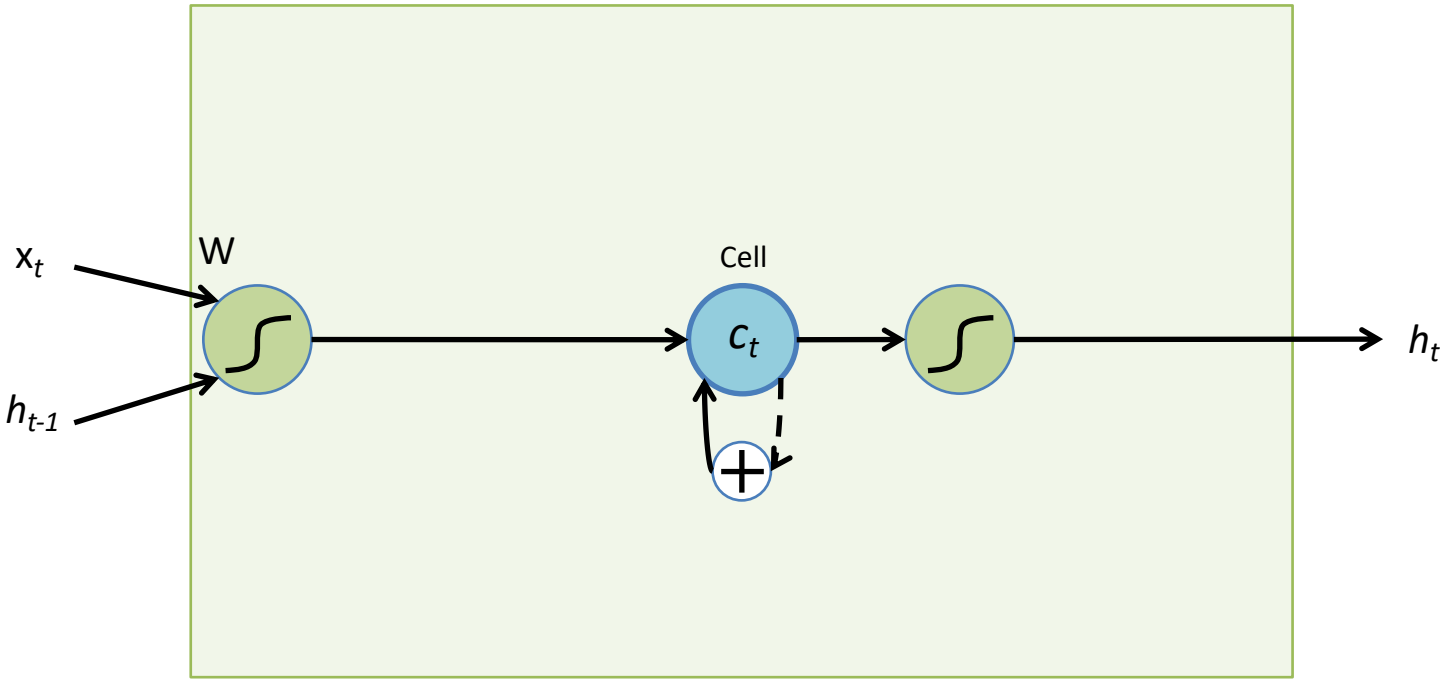
- The LSTM uses this idea of "Constant Error Flow" for RNNs to create a "Constant Error Carousel" (CEC) which ensures that gradients don't decay

- The key component is a memory cell that acts like an accumulator (contains the identity relationship) over time

- Instead of computing new state as a matrix product with the old state, it rather computes the difference between them. Expressivity is the same, but gradients are better behaved

[1] Long Short-Term Memory, Hochreiter *et al.*, 1997

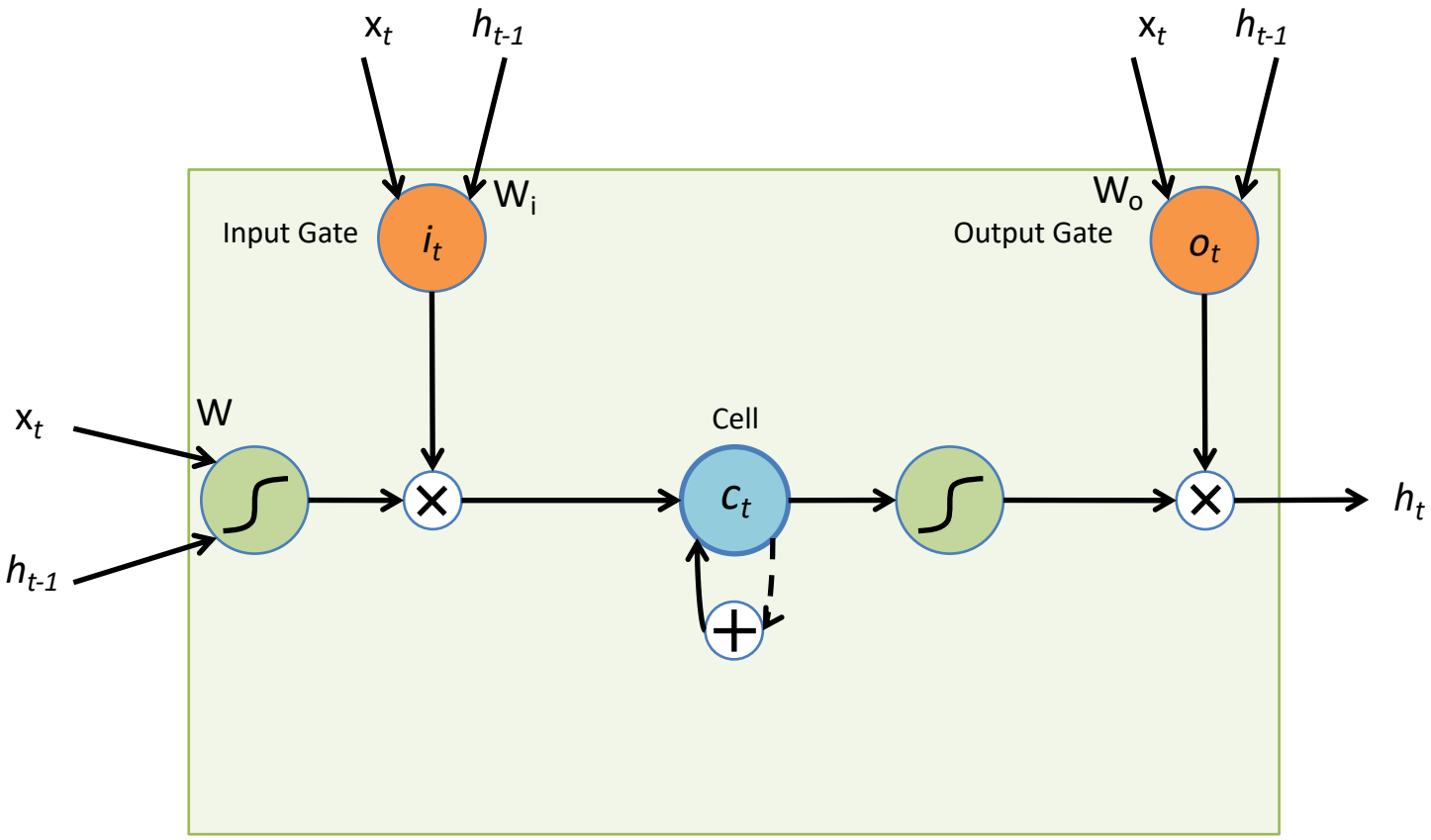
The LSTM Idea



* Dashed line indicates time-lag

$$c_t = c_{t-1} + \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} \qquad h_t = \tanh c_t$$

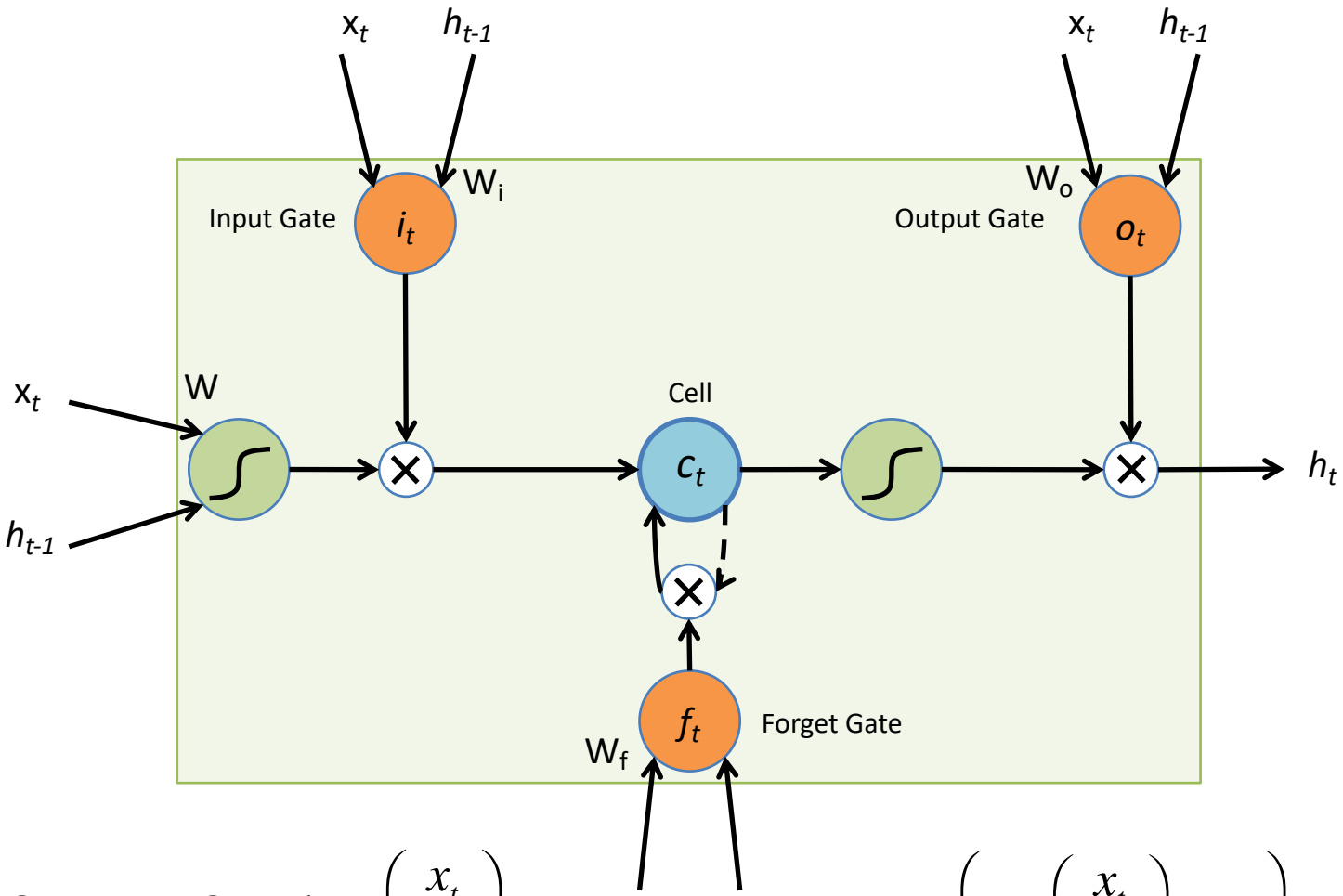
The Original LSTM Cell



$$c_t = c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} \quad h_t = o_t \otimes \tanh c_t \quad i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i \right)$$

Similarly for o_t

The Popular LSTM Cell



$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

Summary

- RNNs allow for processing of variable length inputs and outputs by maintaining state information across time steps
- Various Input-Output scenarios are possible (Single/Multiple)

- Vanilla RNNs are improved upon by LSTMs which address the vanishing gradient problem. Exploding gradients are handled by gradient clipping

- More complex architectures exist, but there are the building blocks to understanding them well.

Other Useful Resources / References

- http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf
- http://www.cs.toronto.edu/~rgrosse/csc321/lec10.pdf

- R. Pascanu, T. Mikolov, and Y. Bengio, On the difficulty of training recurrent neural networks, ICML 2013
- S. Hochreiter, and J. Schmidhuber, Long short-term memory, Neural computation, 1997 9(8), pp.1735-1780
- F.A. Gers, and J. Schmidhuber, Recurrent nets that time and count, IJCNN 2000
- K. Greff , R.K. Srivastava, J. Koutník, B.R. Steunebrink, and J. Schmidhuber, LSTM: A search space odyssey, IEEE transactions on neural networks and learning systems, 2016
- K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, ACL 2014
- R. Jozefowicz, W. Zaremba, and I. Sutskever, An empirical exploration of recurrent network architectures, JMLR 2015

Questions