Recurrent Neural Network (RNN)

Outline

- Why Recurrent Neural Networks (RNNs)?
- The Vanilla RNN unit
- The RNN forward pass
- RNN Back propagation
- Issues with the Vanilla RNN

Motivation

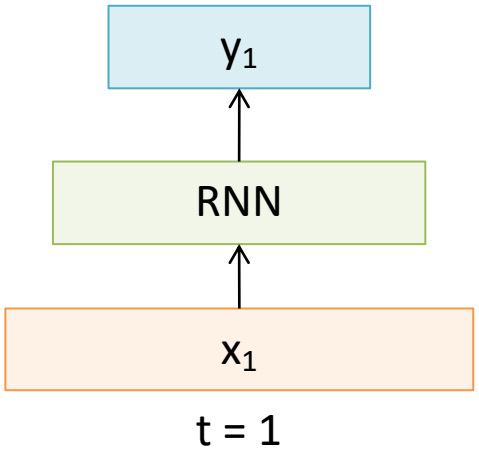
- Not all problems can be converted into one with fixed-length inputs and outputs
- Problems such as Speech Recognition or Time-series Prediction require a system to store and use context information
  - Simple case: Output YES if the number of 1s is even, else NO
    1000010101 – YES, 100011 – NO, …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

Recurrent Neural Networks (RNNs)

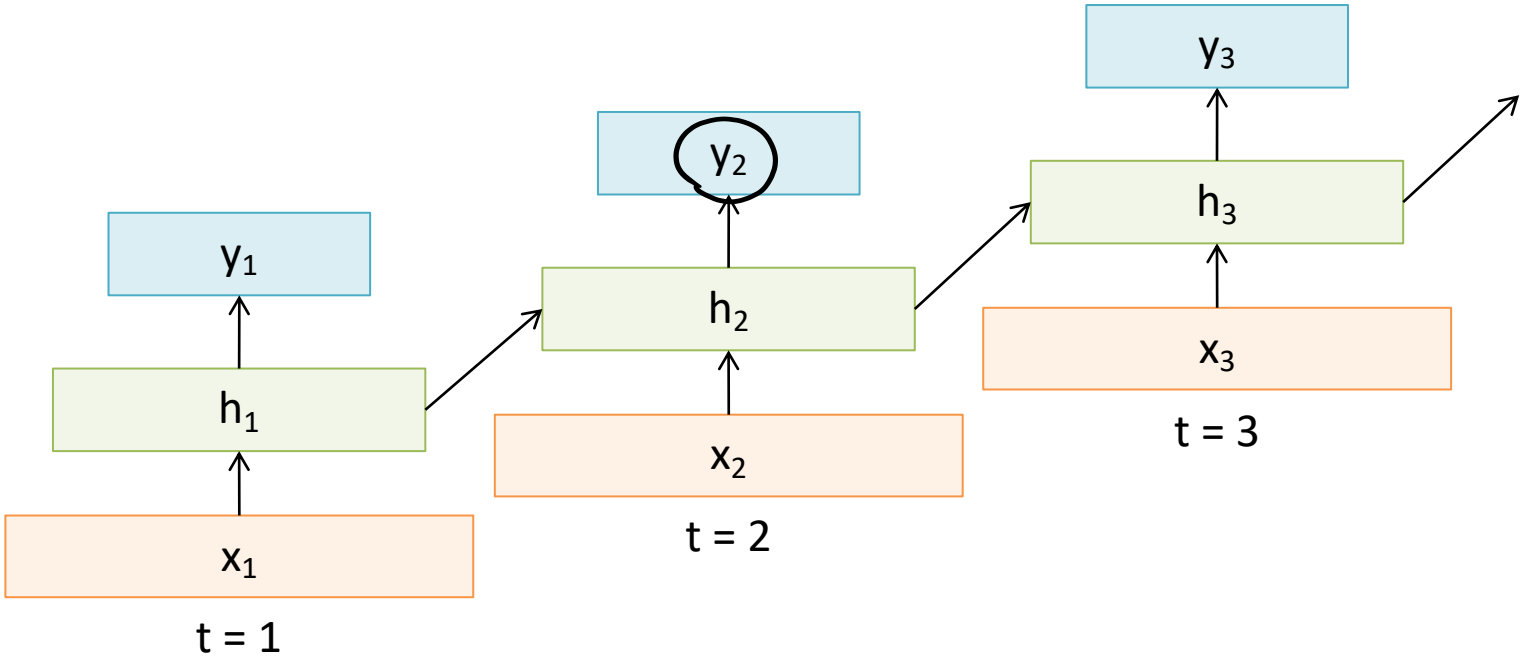
- Recurrent Neural Networks take the previous output or hidden states as inputs.
  The composite input at time t has some historical information about the happenings at time T < t

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

Sample Feed-forward Network

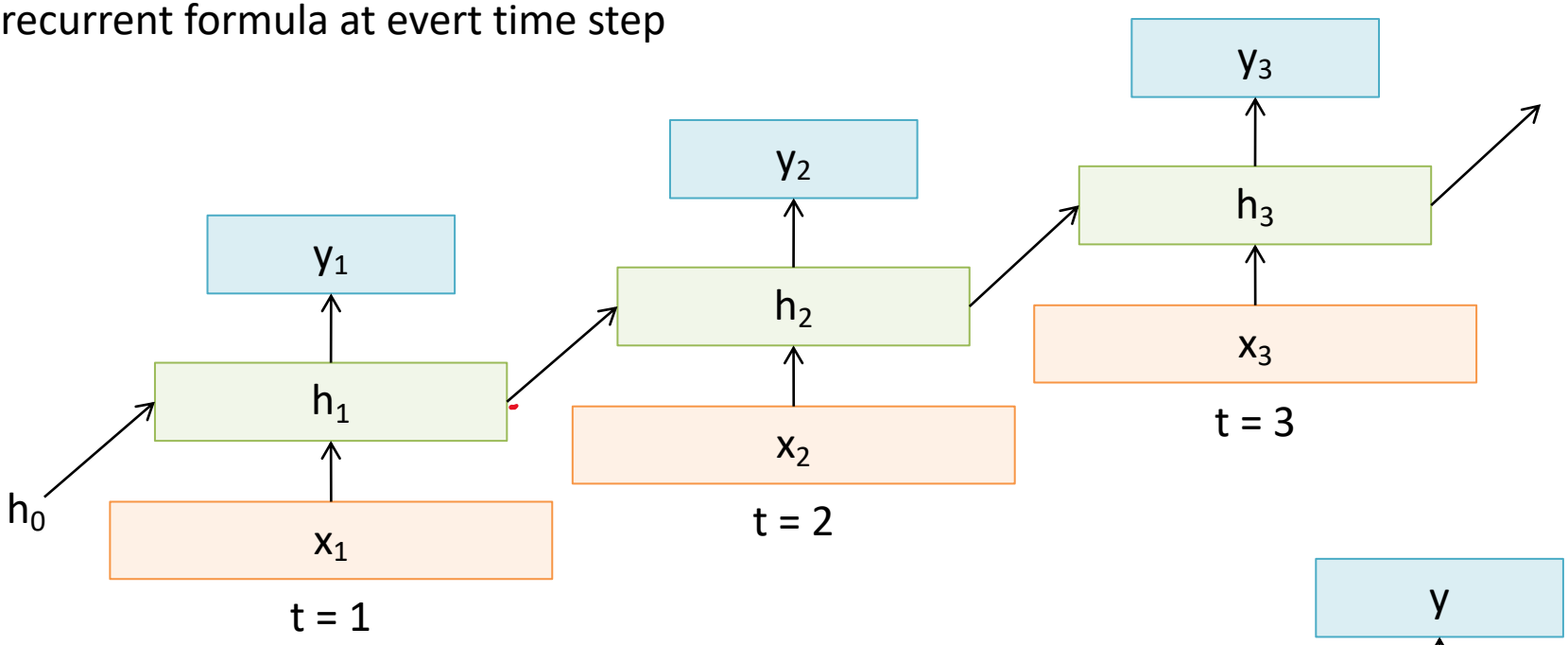


Sample RNN



Recurrent Neural Network

We can process a sequence of vectors x by applying a recurrent formula at evert time step

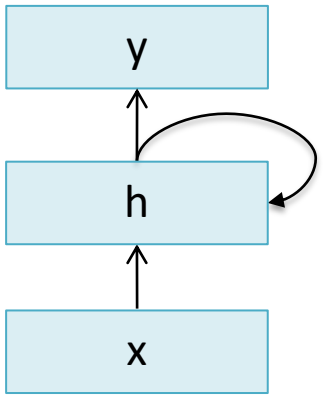


$$h_t = f_w(h_{t-1}, x_t)$$

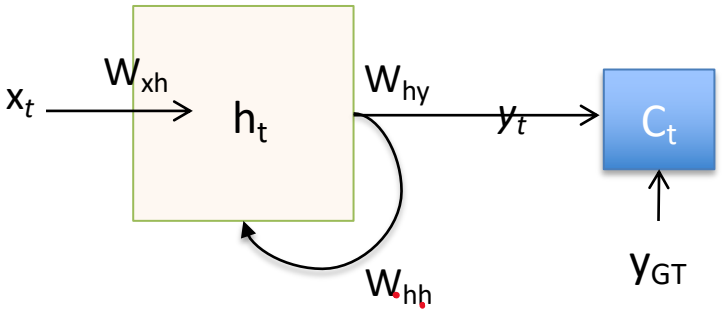
New state

old state

Input



Vanilla Recurrent Neural Network



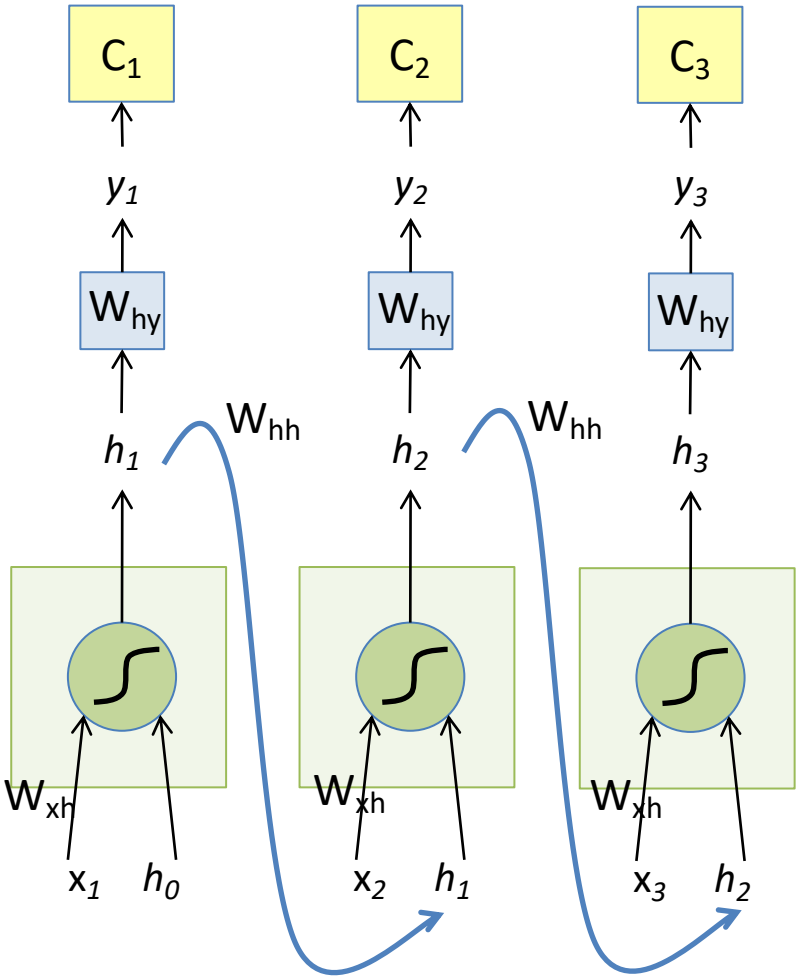
$$h_t = f_w(h_{t-1}, x_t)$$

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

$$y_t = W_{hy}h_t$$

$$C_t = \text{Loss}(y_t, y_{Gt})$$

The Vanilla RNN Forward



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

$$y_t = W_{hy}h_t$$

$$C_t = \text{Loss}\,(y_t, y_{Gt})$$

SGD Update

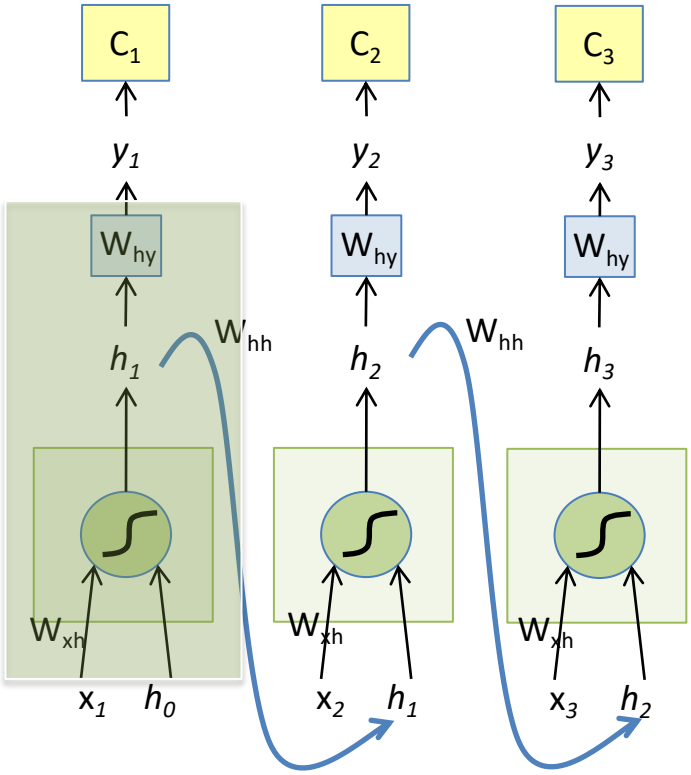
$$W \leftarrow W - \eta \frac{\partial C}{\partial W}$$

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

shared weights across time: W_{hy}, W_{hh}, W_{xh}

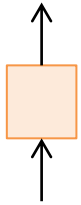
Recurrent Neural Networks (RNNs)

- Note that the weights (W_{hy}, W_{hh}, W_{xh}) are shared over time
- Essentially, copies of the RNN cell are made over time (unrolling/unfolding), with different inputs at different time steps



Input – Output Scenarios

Single - Single



Feed-forward Network

Single - Multiple

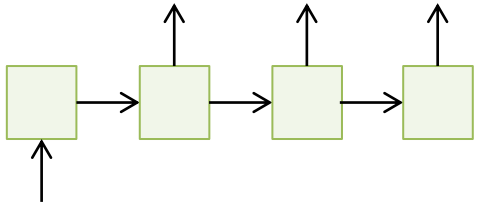
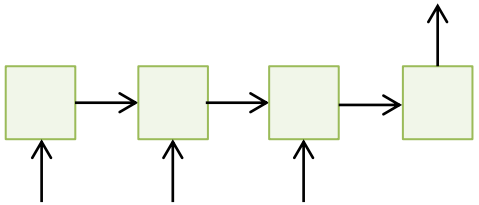


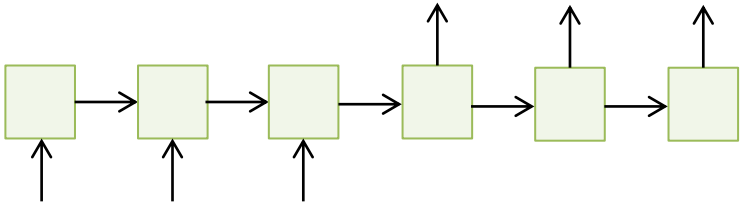
Image Captioning

Multiple - Single

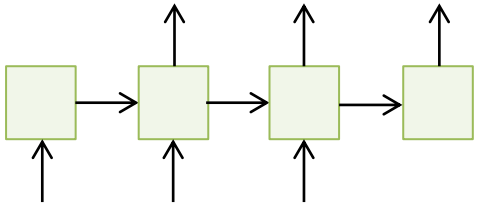


Sentiment Classification

Multiple - Multiple



Translation



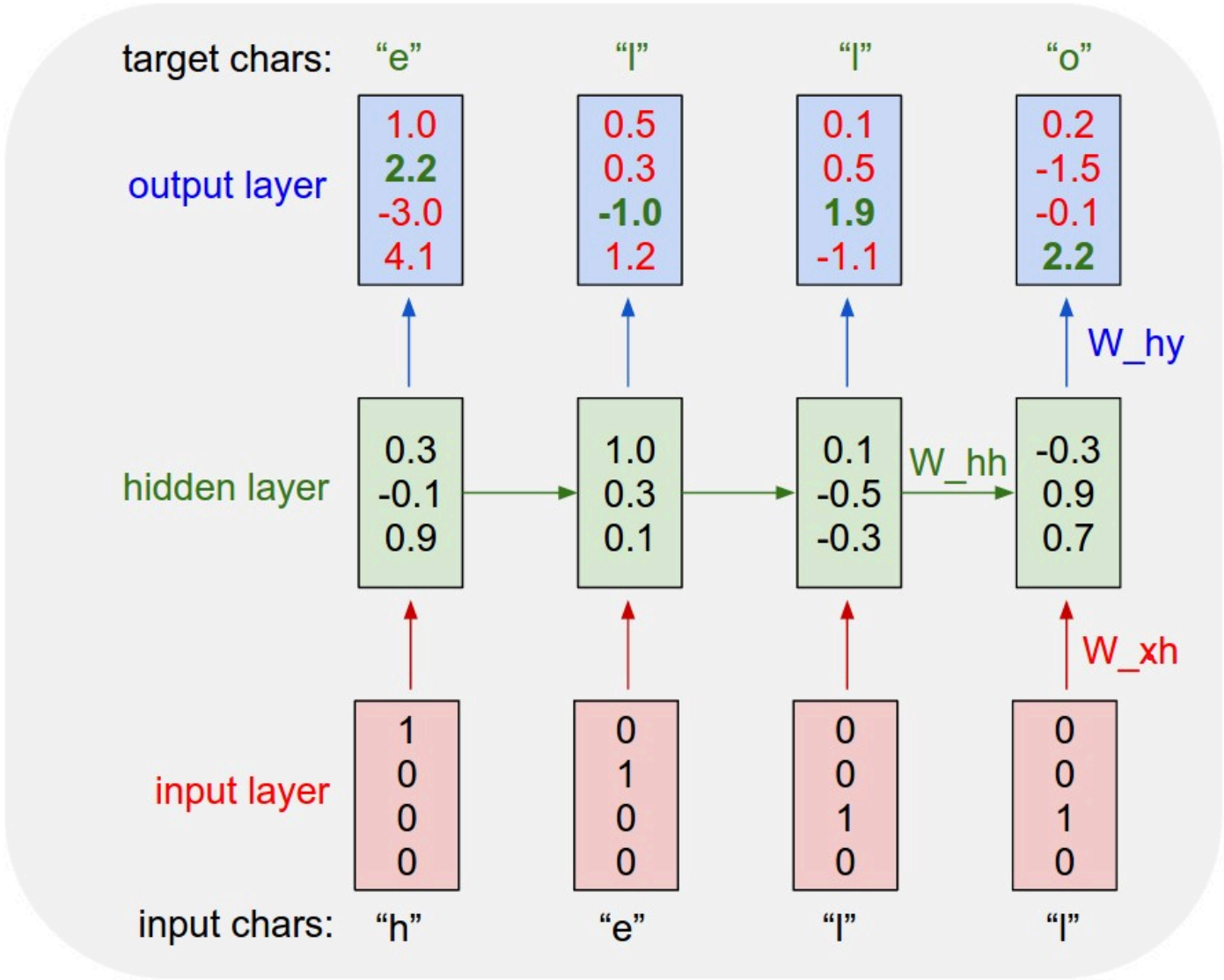
Video Captioning

Sample Sentiment Classification

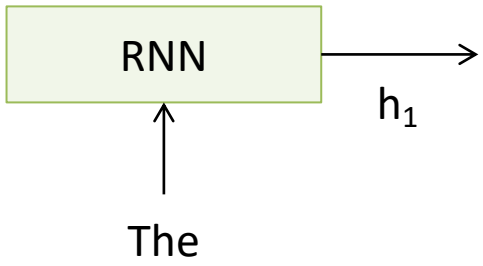
- Classify a restaurant review from Yelp! OR movie review from IMDB as positive or negative

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

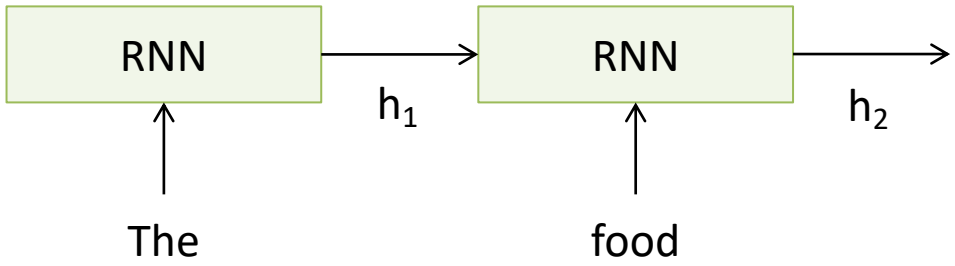
- “The food was really good”
- “The chicken crossed the road because it was uncooked”


target chars:
"e"
"l"
"l"
"o"
output layer
1.0
2.2
-3.0
4.1
0.5
0.3
-1.0
1.2
0.1
0.5
1.9
-1.1
0.2
-1.5
-0.1
2.2
W_hy
hidden layer
0.3
-0.1
0.9
1.0
0.3
0.1
0.1
-0.5
-0.3
W_hh
-0.3
0.9
0.7
W_xh
input layer
1
0
0
0
0
1
0
0
0
0
1
0
0
0
1
0
input chars:
"h"
"e"
"l"
"l"

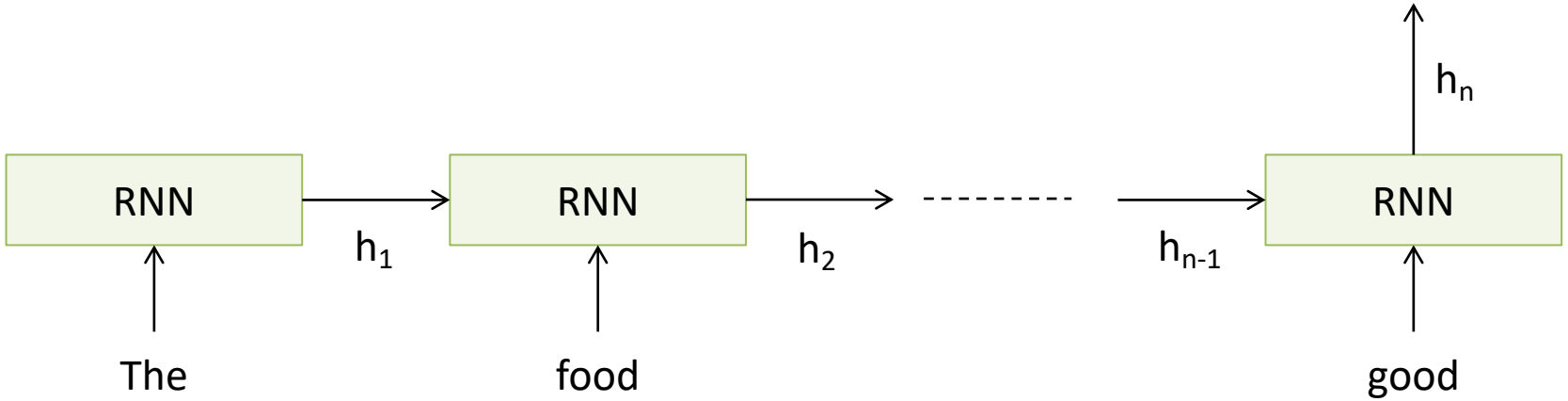
Sentiment Classification



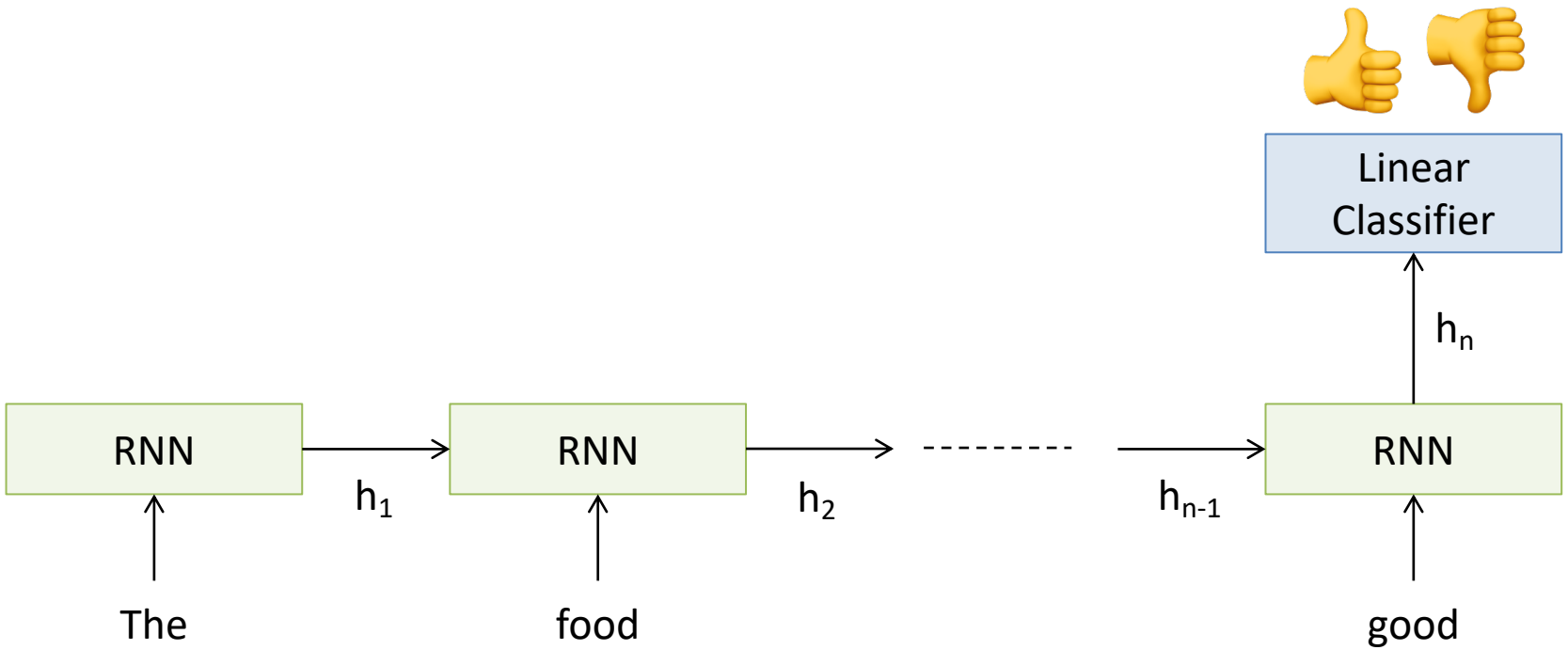
Sentiment Classification



Sentiment Classification



Sentiment Classification



Sentiment Classification

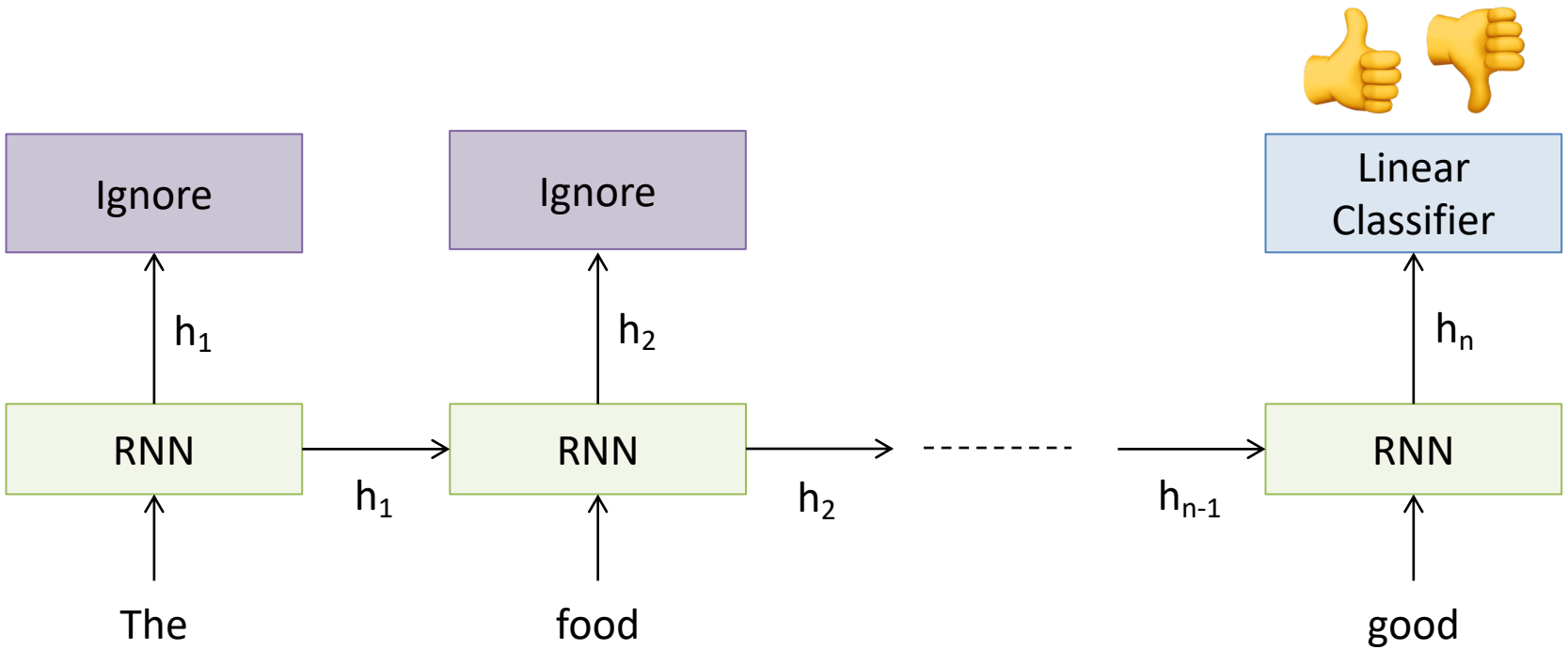
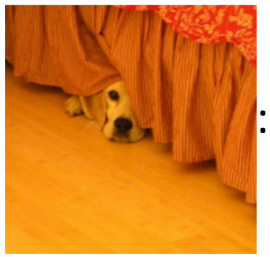


Image Captioning

- 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

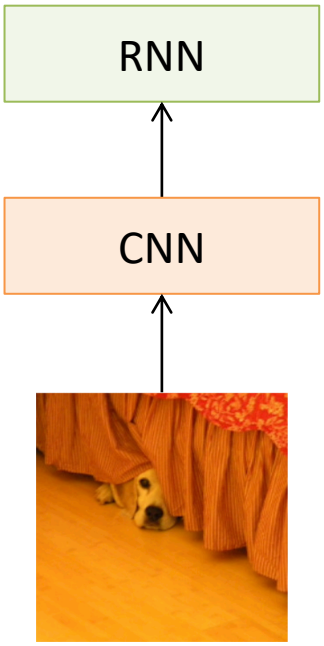


Image Captioning

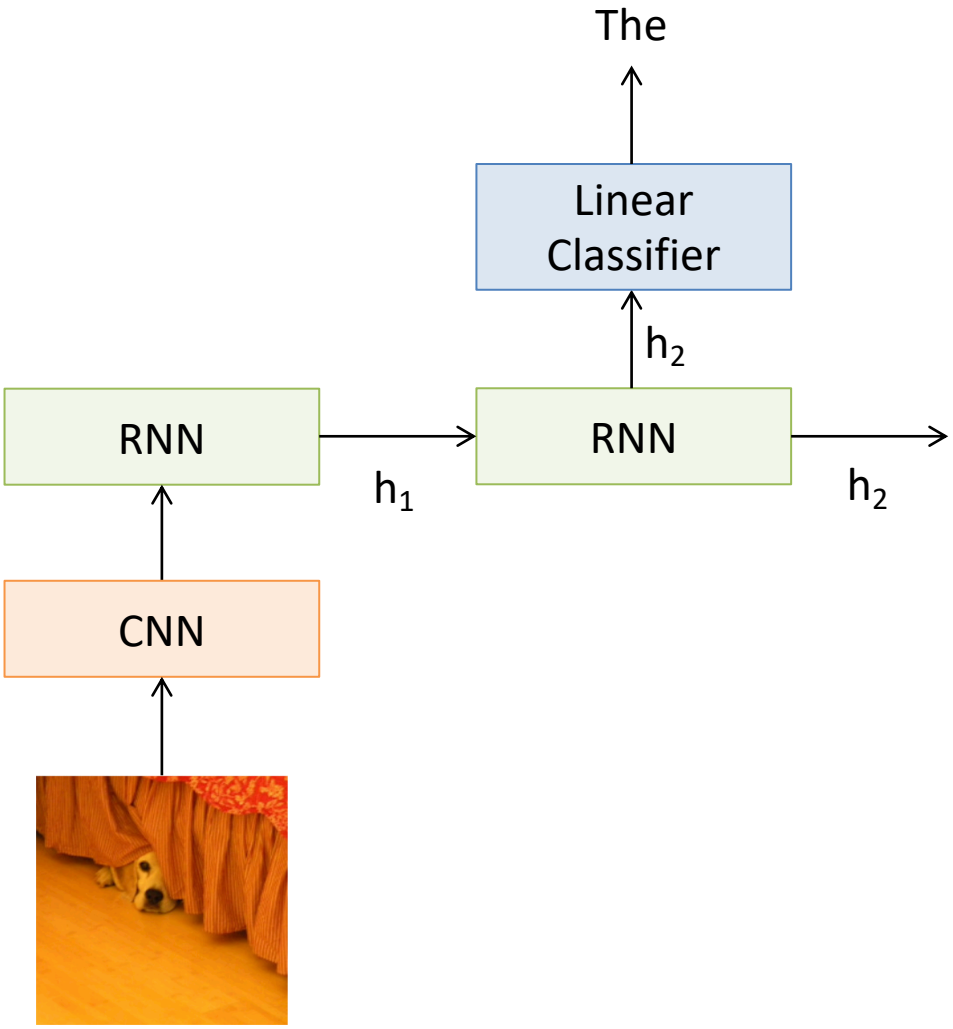
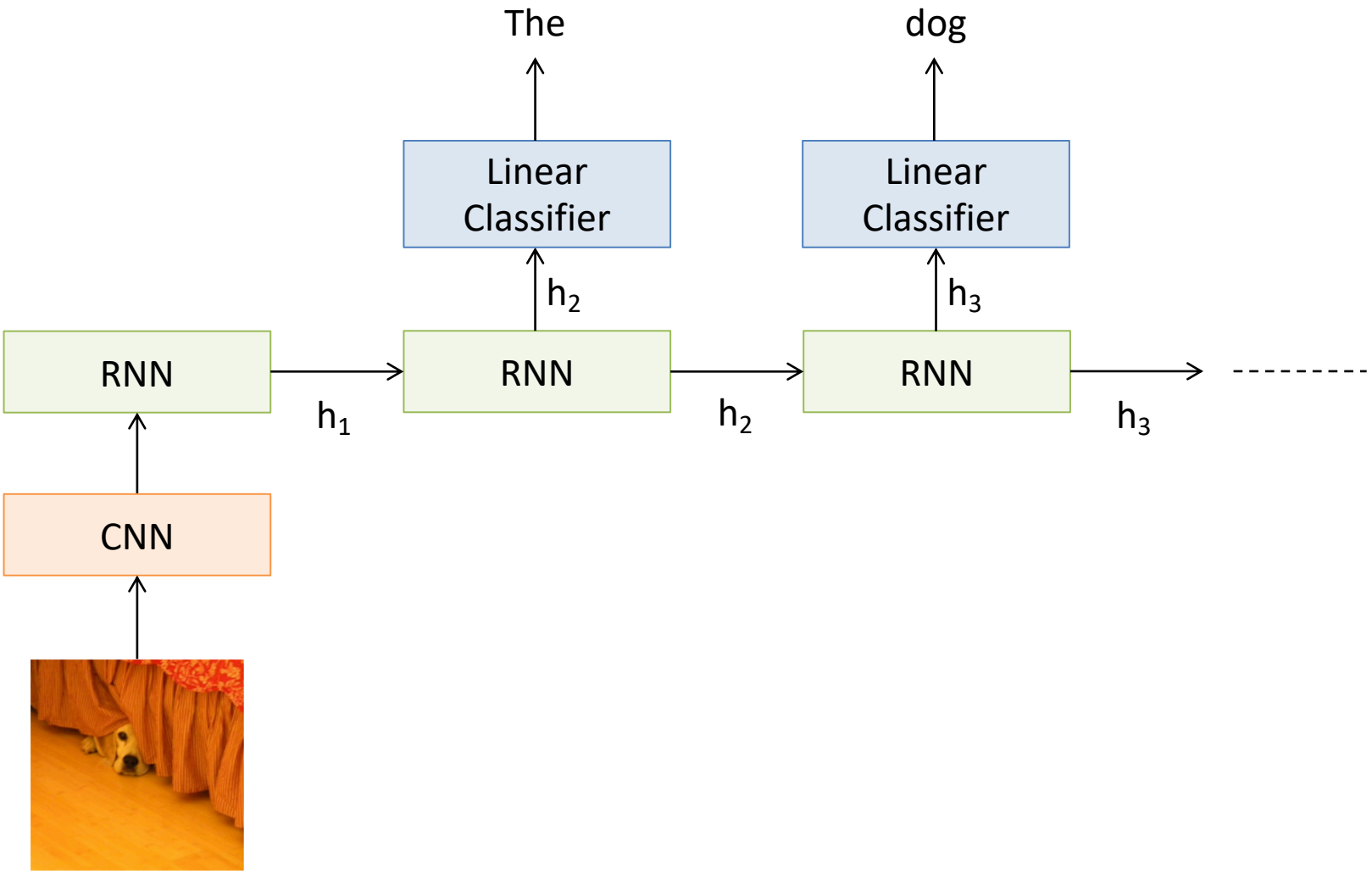


Image Captioning



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.

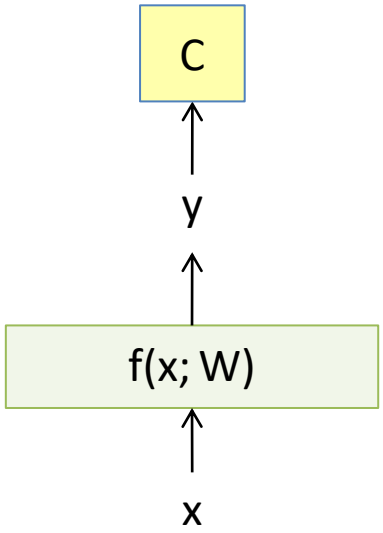
A group of young people playing a game of frisbee.

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]

Back Propagation Refresher



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

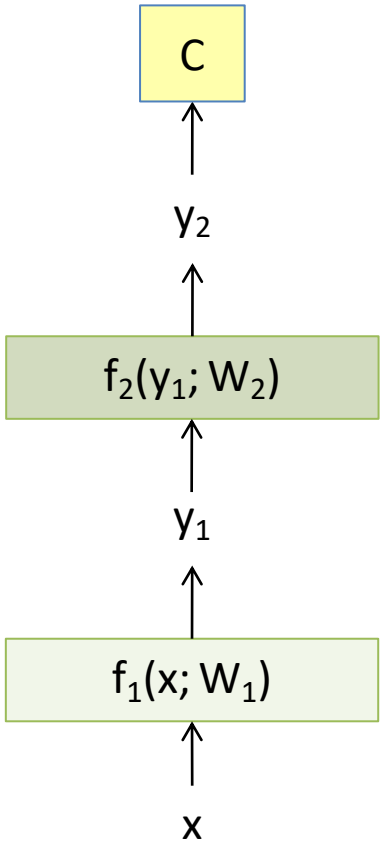
$$C = \text{Loss}(y, y_{GT})$$

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



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

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

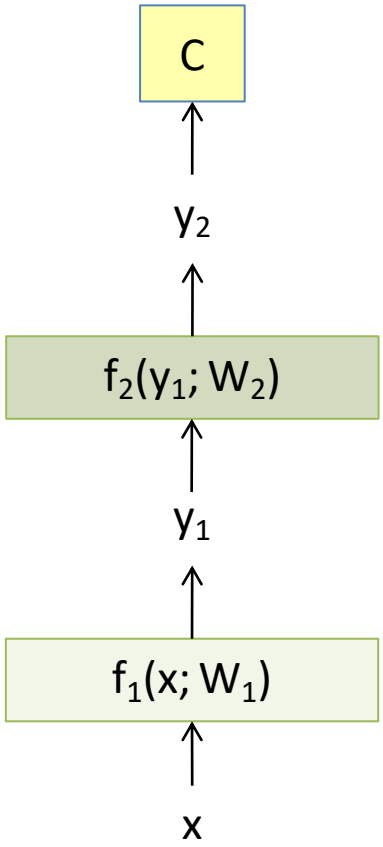
$$C = \text{Loss}(y_2, y_{GT})$$

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



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

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

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

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

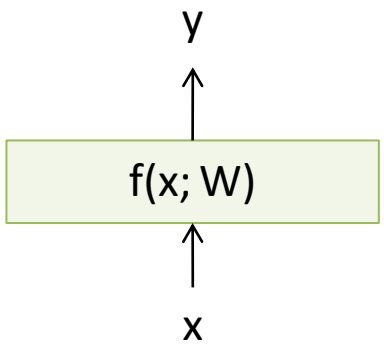
$$\frac{\partial C}{\partial W_2} = \left(\frac{\partial C}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial W_2}\right)$$

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

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

Application of the Chain Rule

Chain Rule for Gradient Computation



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

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

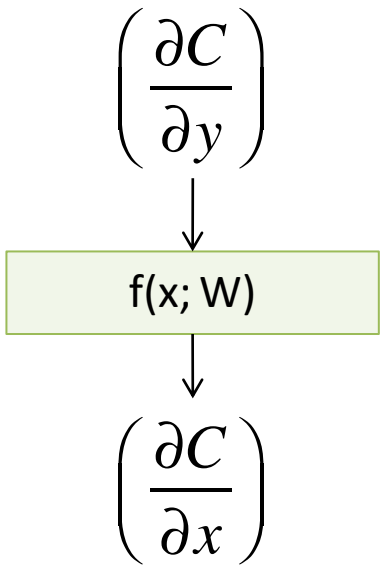
Intrinsic to the layer are:

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

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

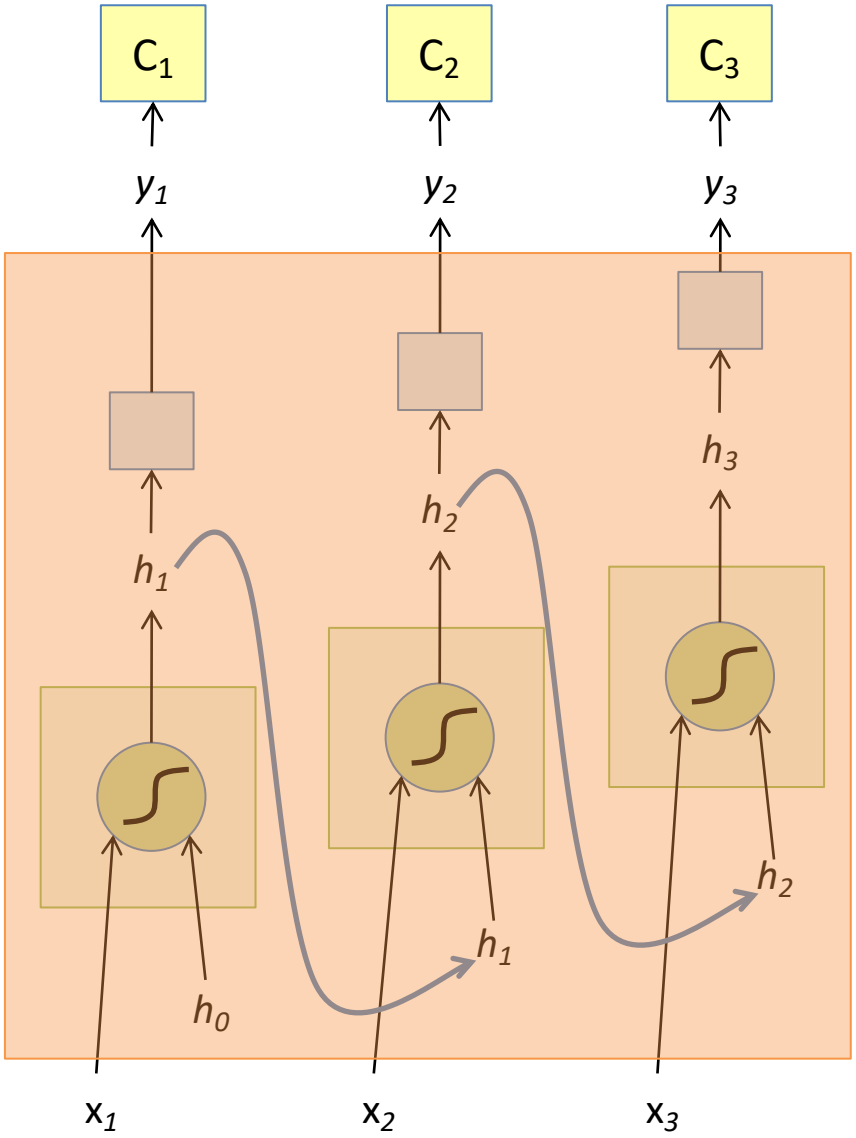
Equations for common layers: http://arunmallya.github.io/writeups/nn/backprop.html

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

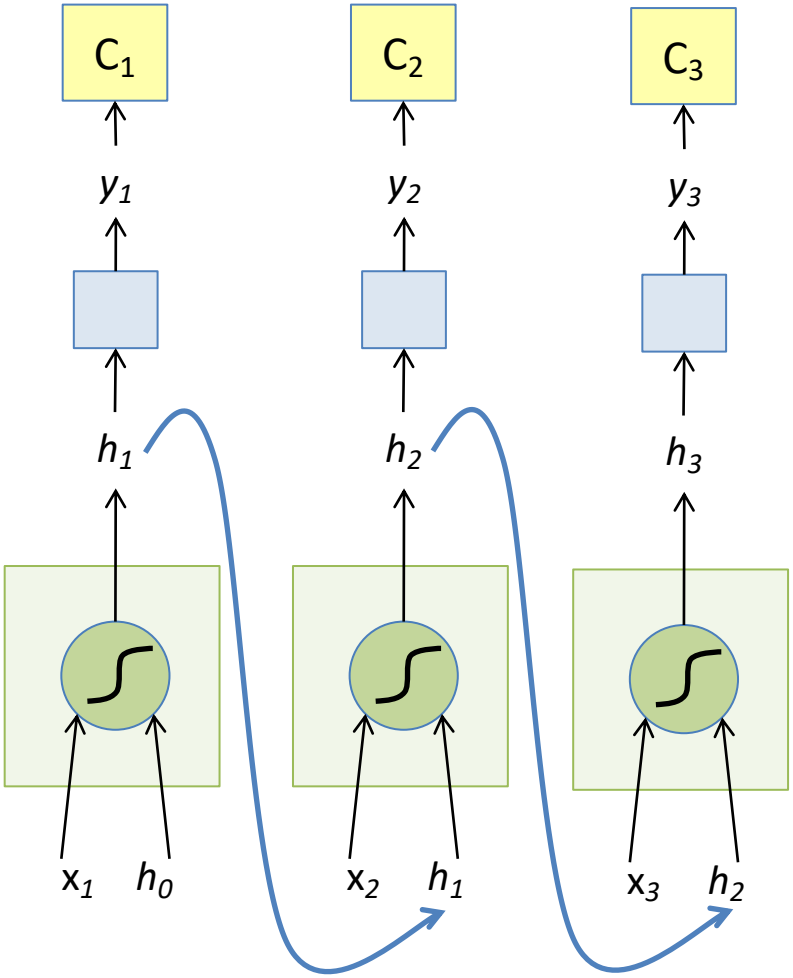
- The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and then applied to the RNN weights

The Unfolded Vanilla RNN

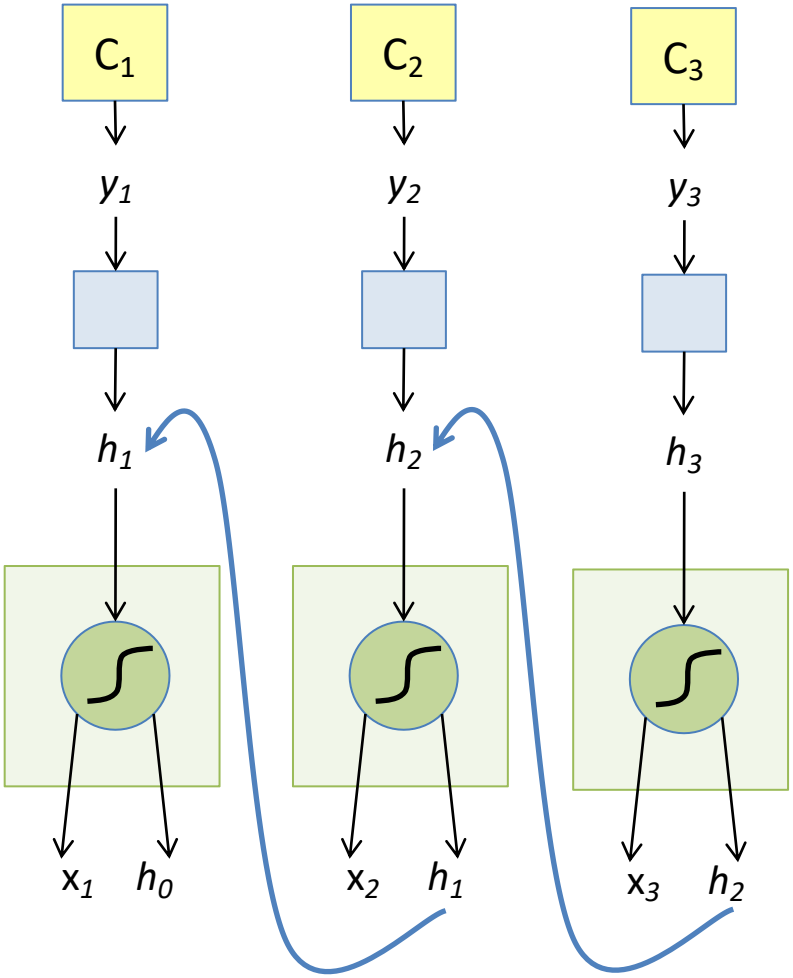


- Treat the unfolded network as one big feed-forward network!
- This big network takes in entire sequence as an input
- Compute gradients through the usual backpropagation
- Update shared weights

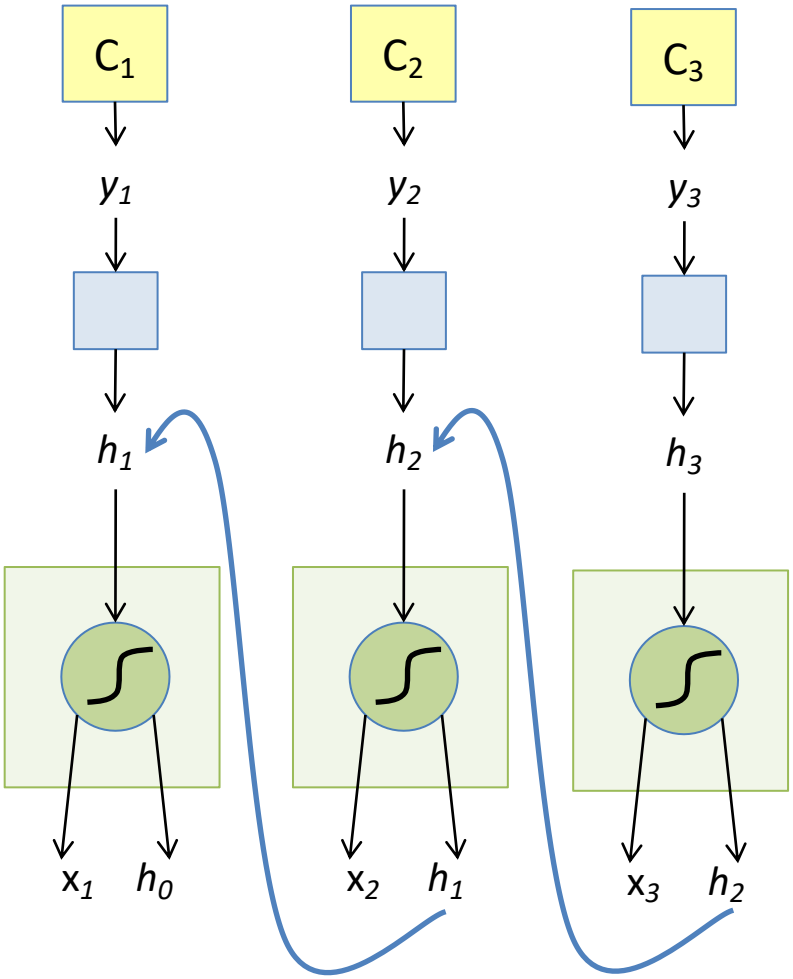
The Unfolded Vanilla RNN Forward



The Unfolded Vanilla RNN Backward



The Vanilla RNN Backward



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

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

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

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

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