Introduction to RNNs

Arun Mallya

Best viewed with [Computer Modern fonts] installed

Outline

- Why Recurrent Neural Networks (RNNs)?
- The Vanilla RNN unit
- The RNN forward pass
- Backpropagation refresher
- The RNN backward pass
- Issues with the Vanilla RNN
- The Long Short-Term Memory (LSTM) unit
- The LSTM Forward & Backward pass
- LSTM variants and tips
  - Peephole LSTM
  - GRU

Motivation

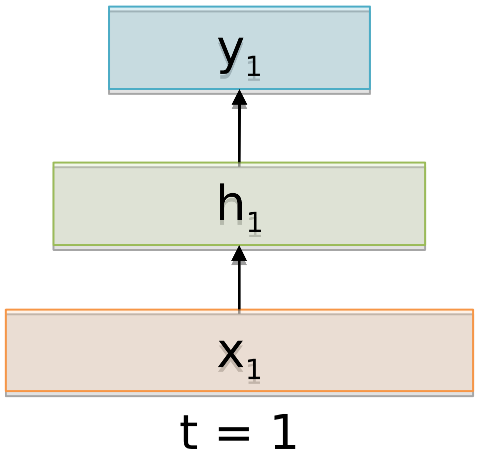
- Not all problems can be converted into one with fixed-length inputs and outputs
- Problems such as Speech Recognition or Time-series Prediction require a system to store and use context information
  - Simple case: Output YES if the number of 1s is even, else NO
    1000010101 – YES, 100011 – NO, …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

Recurrent Neural Networks (RNNs)

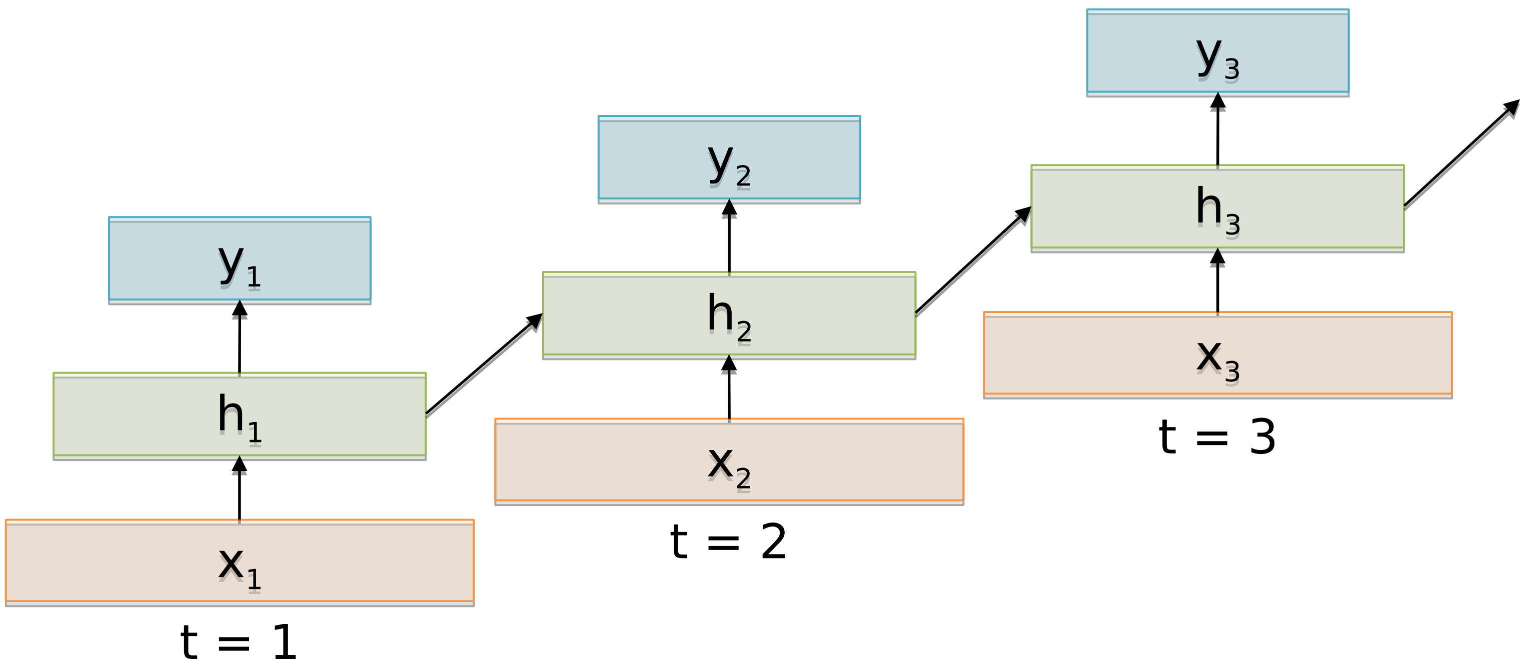
- Recurrent Neural Networks take the previous output or hidden states as inputs.
  The composite input at time t has some historical information about the happenings at time $T < t$

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

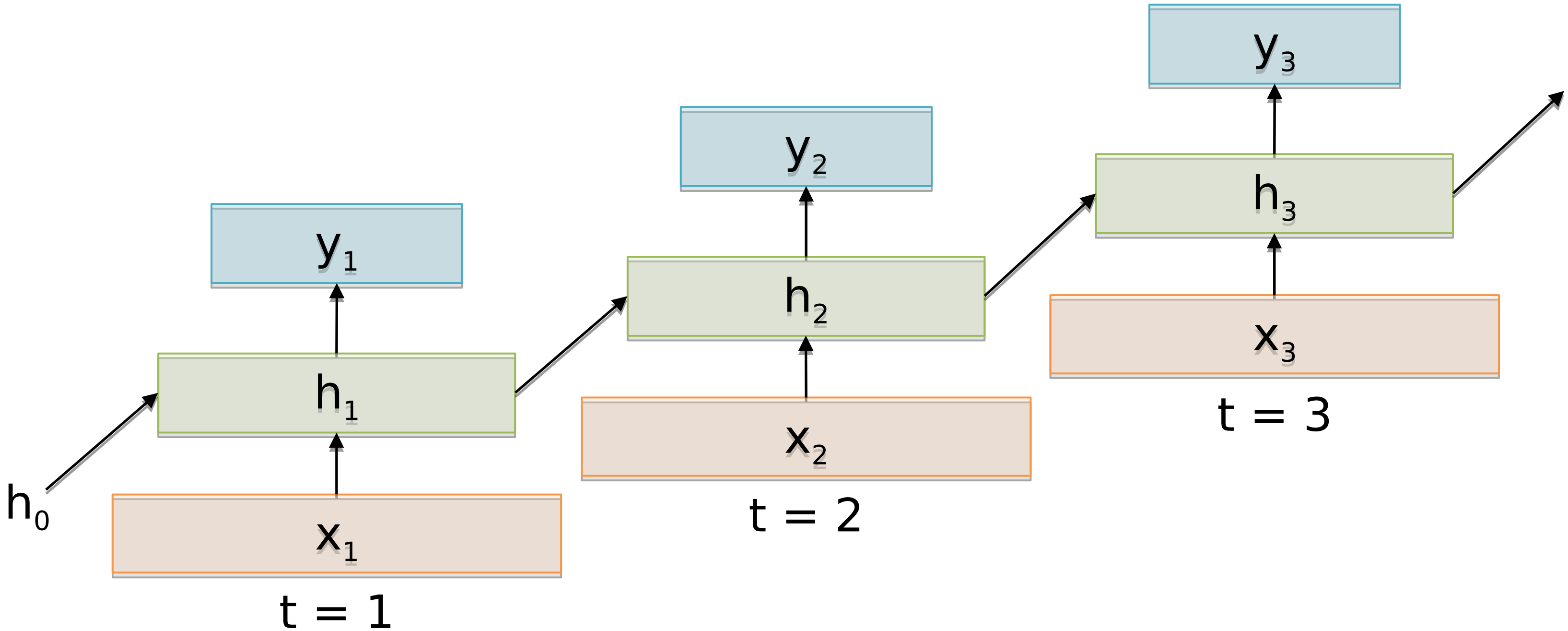
Sample Feed-forward Network



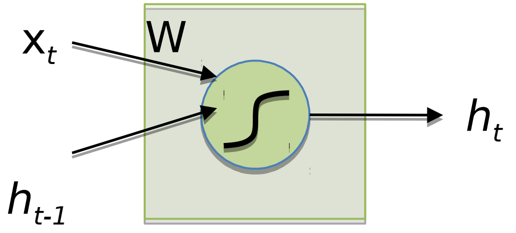
Sample RNN



Sample RNN

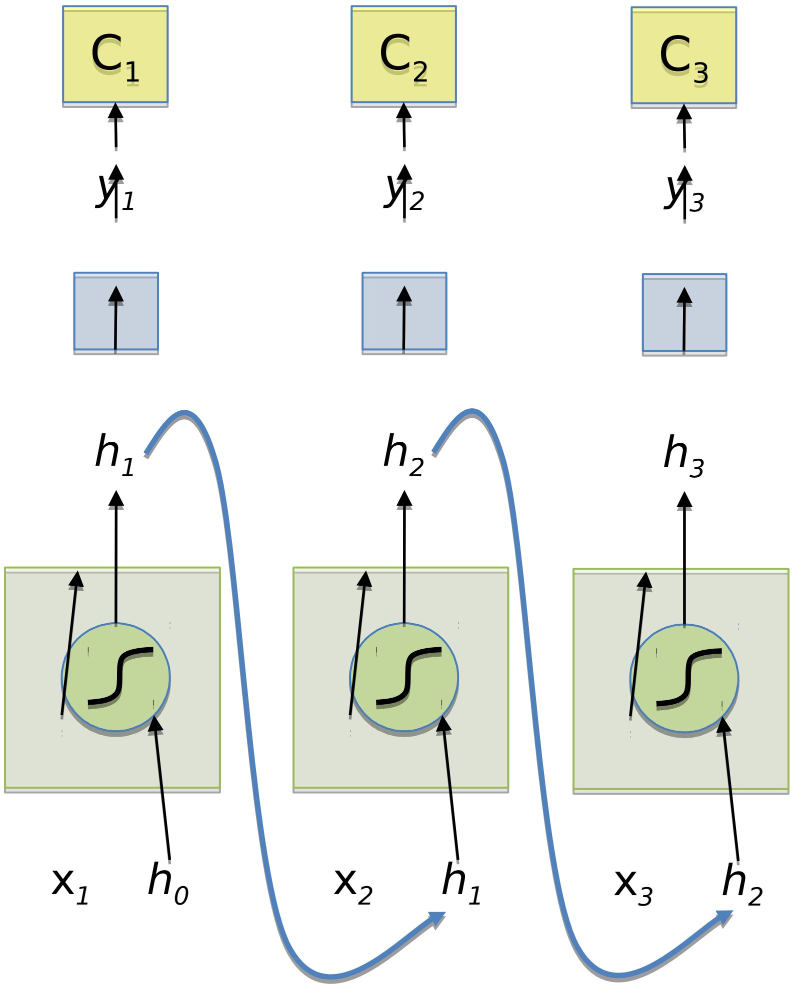


The Vanilla RNN Cell



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

The Vanilla RNN Forward

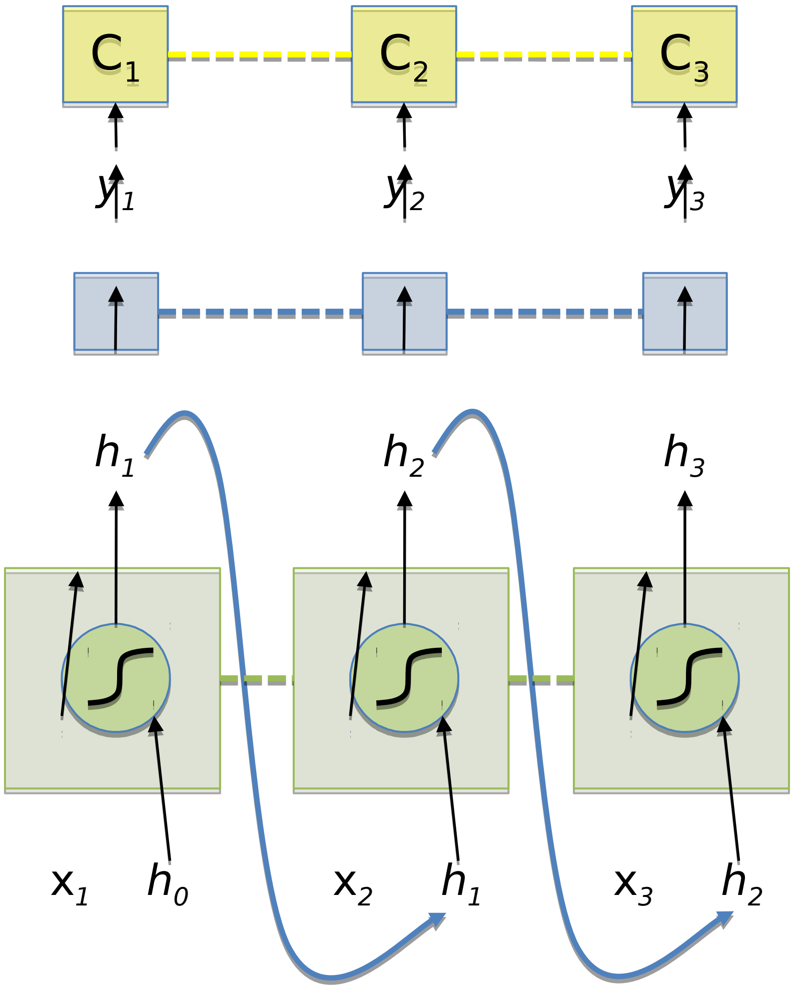


$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

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

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

The Vanilla RNN Forward



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

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

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

----- indicates shared weights

Recurrent Neural Networks (RNNs)

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

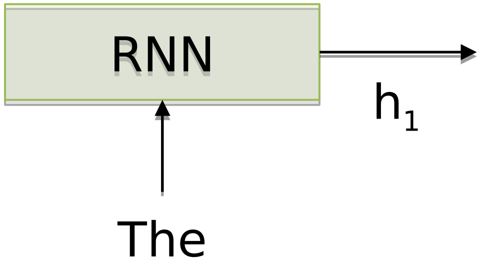
Sentiment Classification

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

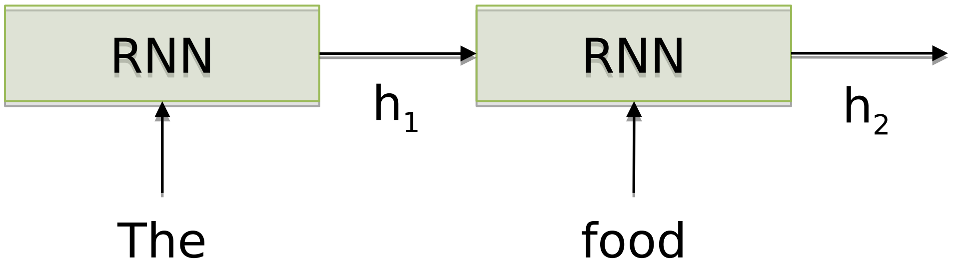
- Inputs: Multiple words, one or more sentences
- Outputs: Positive / Negative classification

- "The food was really good"
-

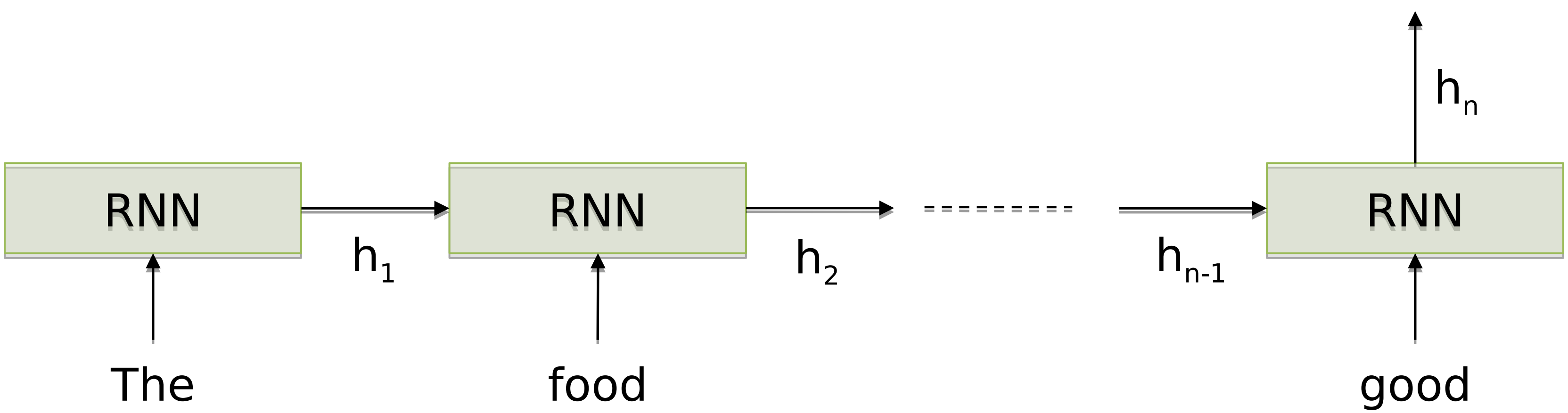
Sentiment Classification



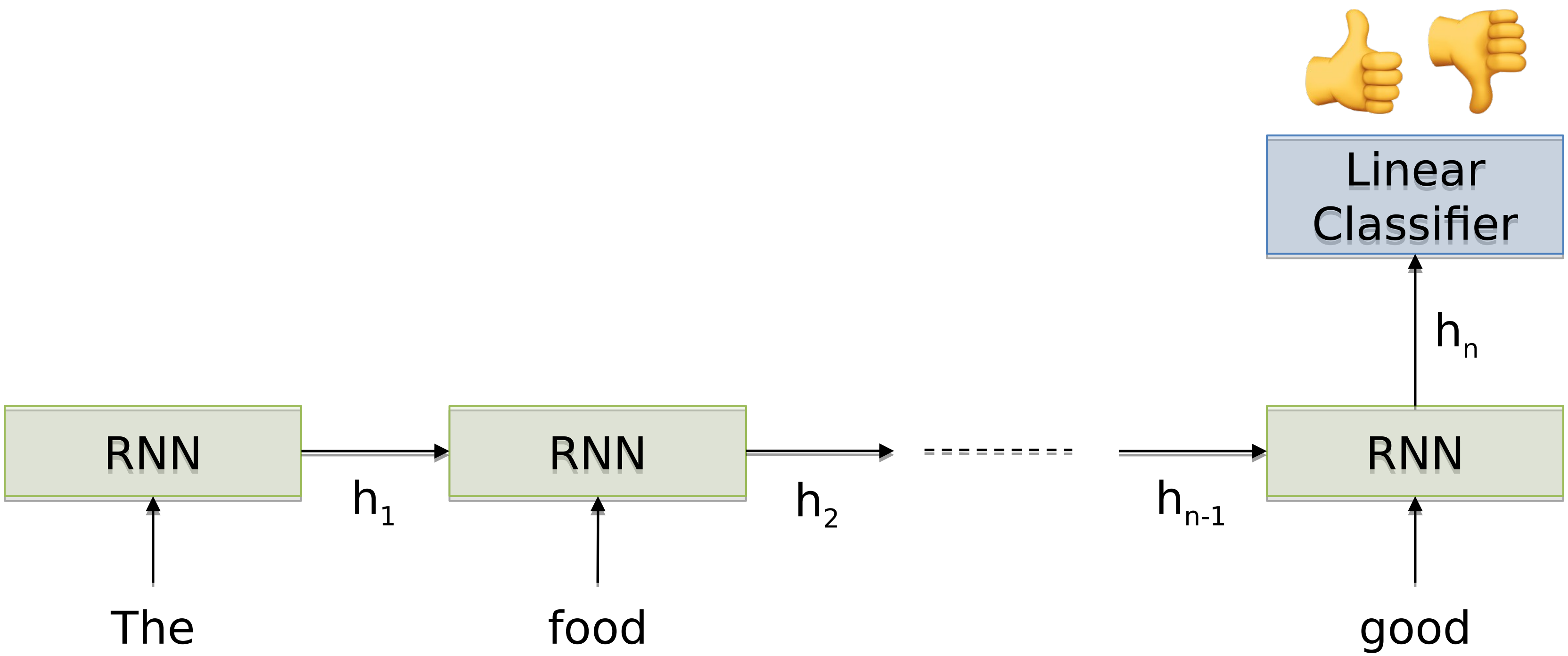
Sentiment Classification



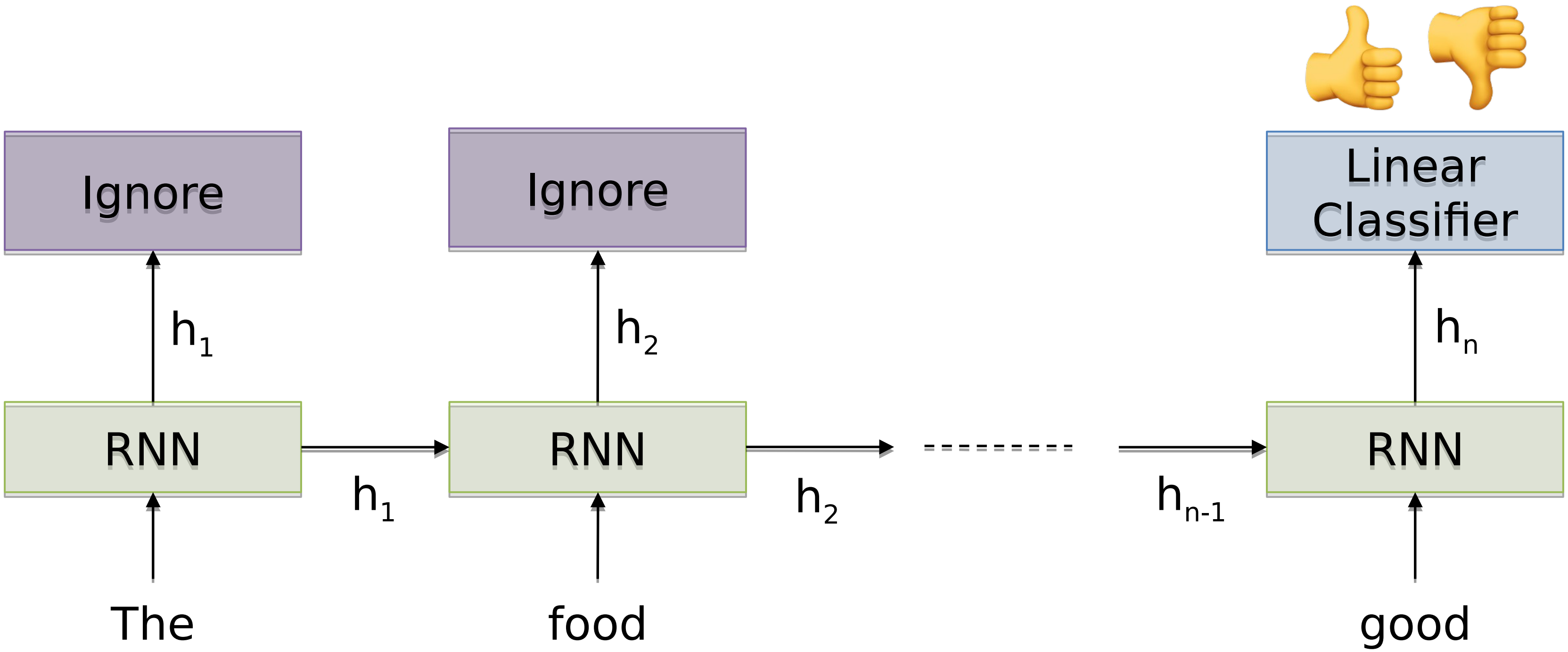
Sentiment Classification



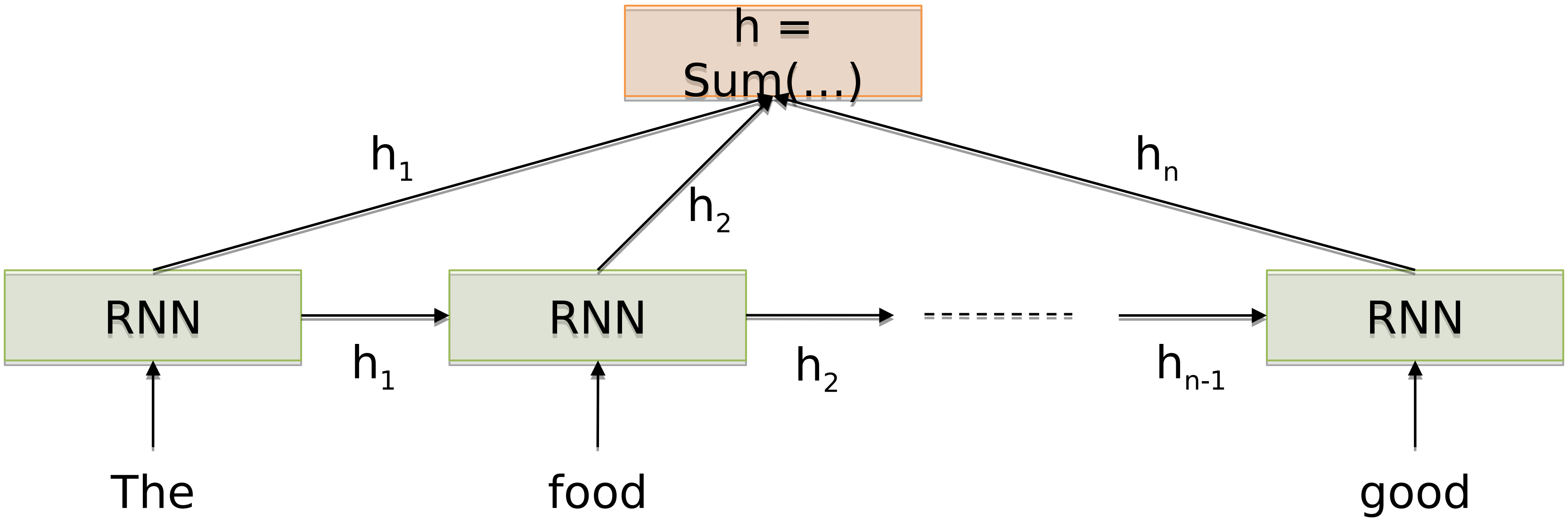
Sentiment Classification



Sentiment Classification

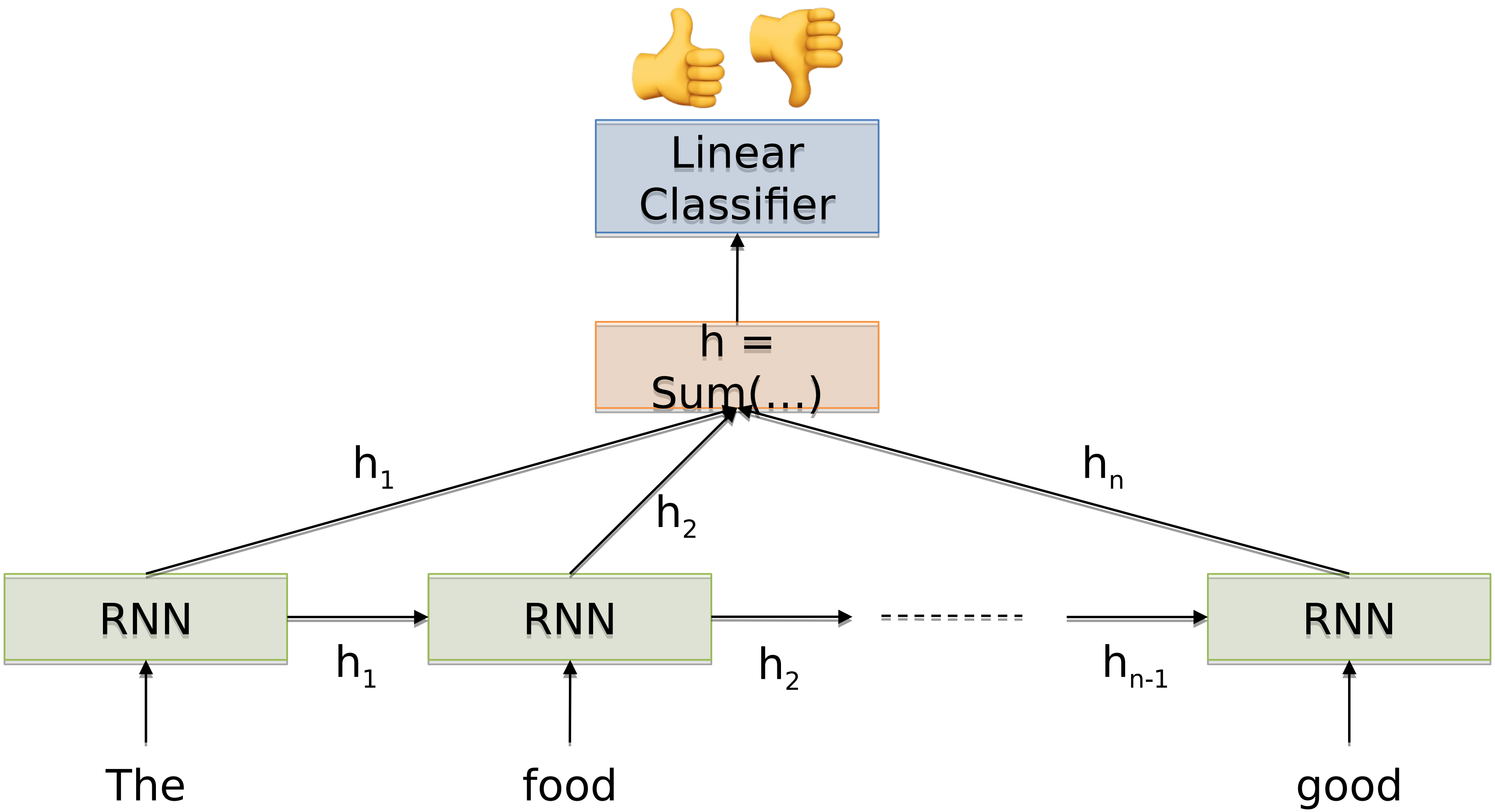


Sentiment Classification



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

Sentiment Classification

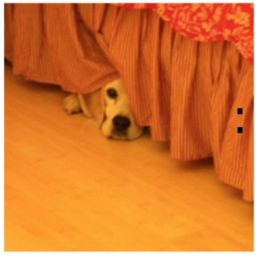


http://deeplearning.net/tutorial/lstm.html

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

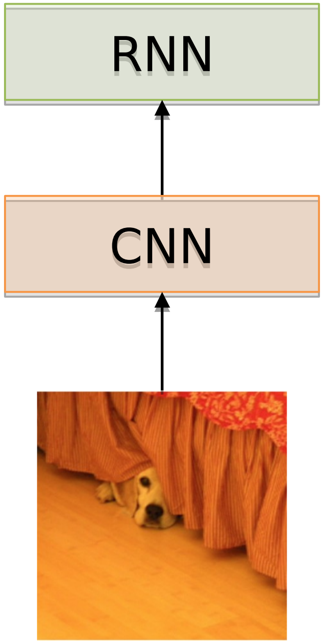


Image Captioning

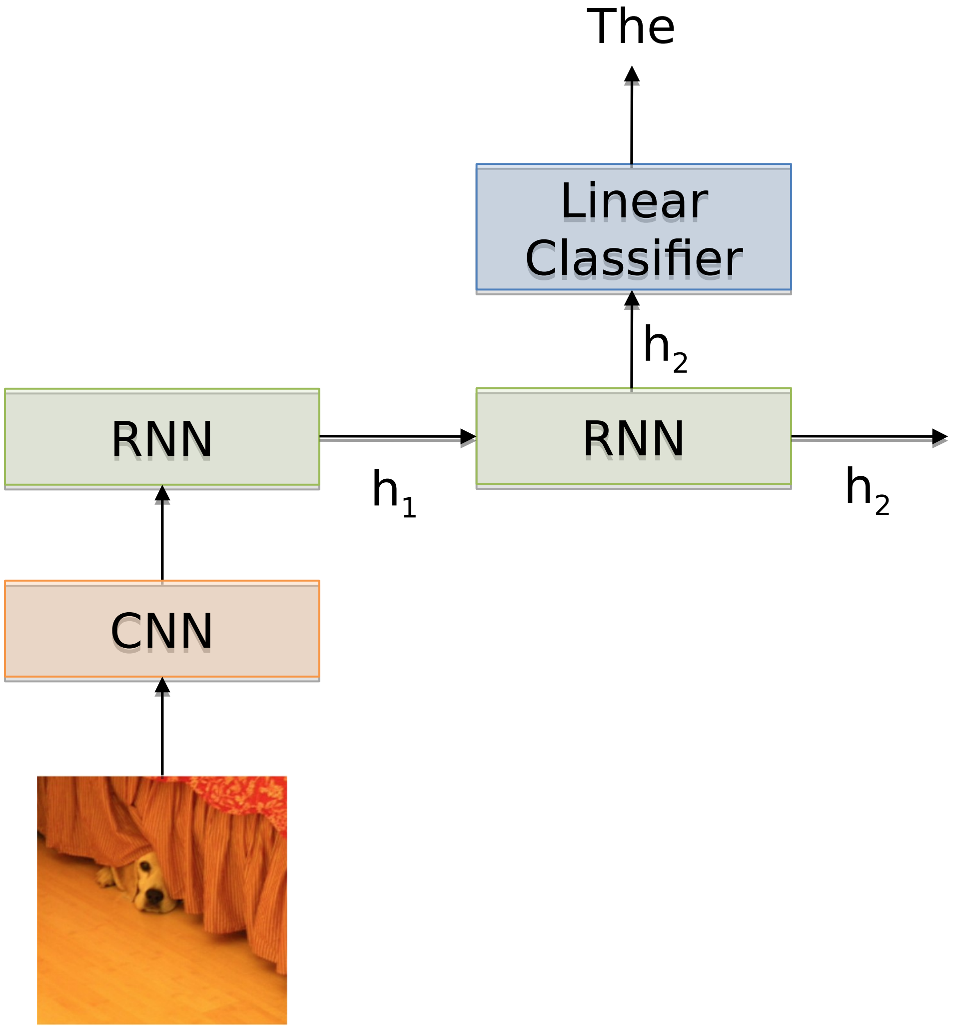
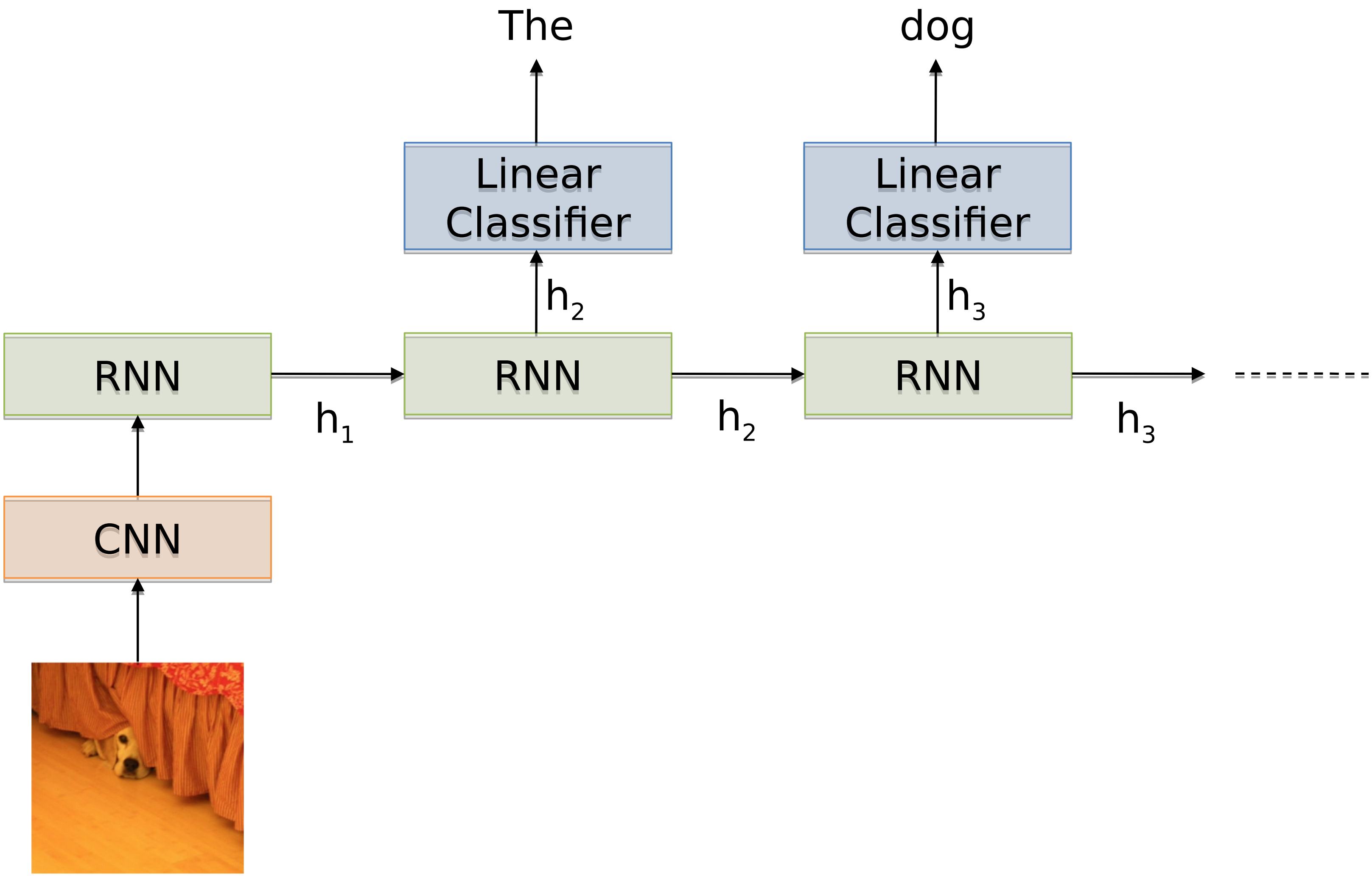


Image Captioning



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.

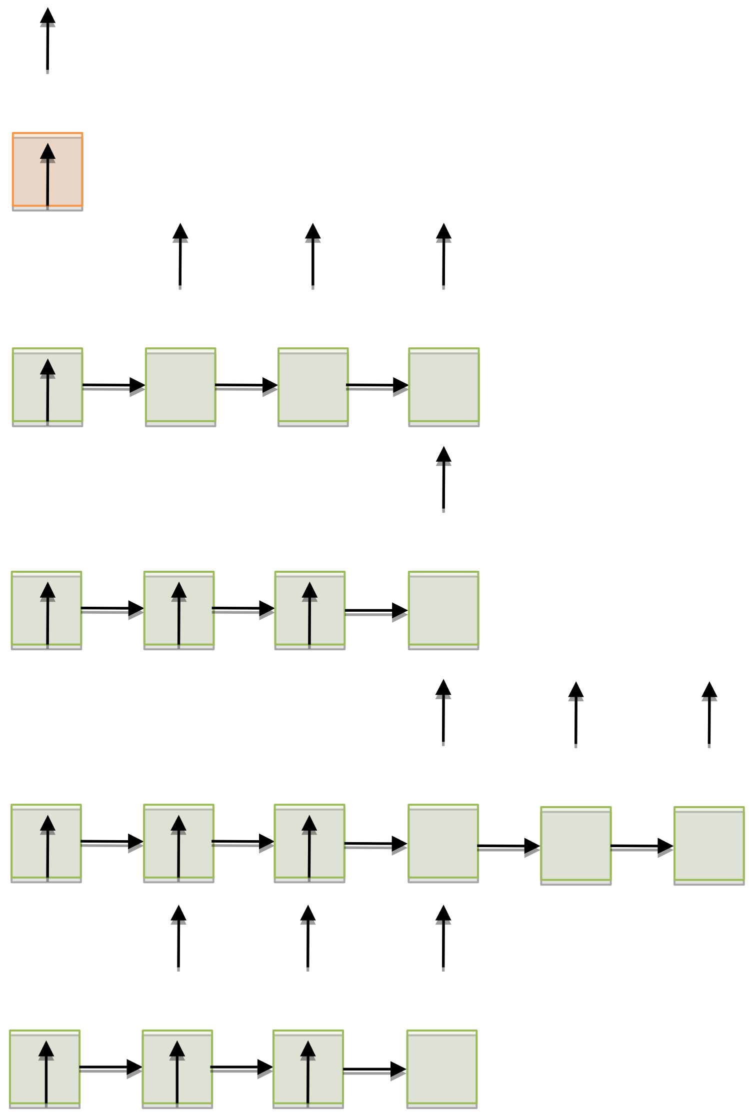
RNN Outputs: Language Modeling

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

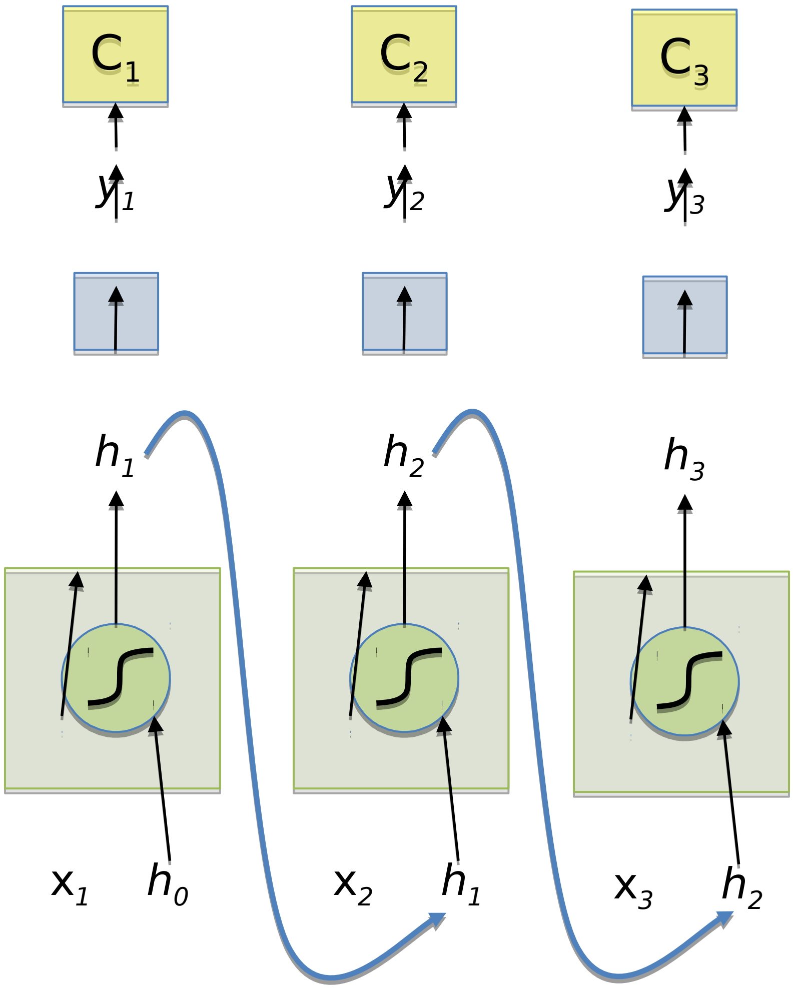
Image Captioning

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



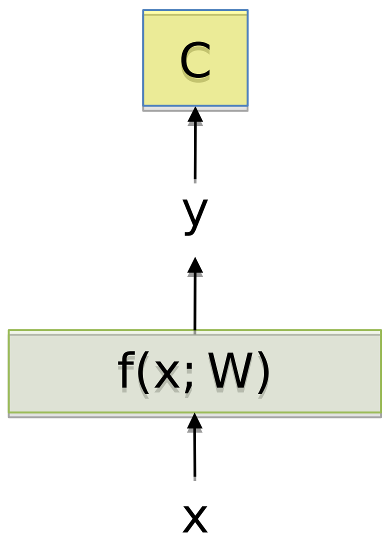
$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

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

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

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

BackPropagation Refresher



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

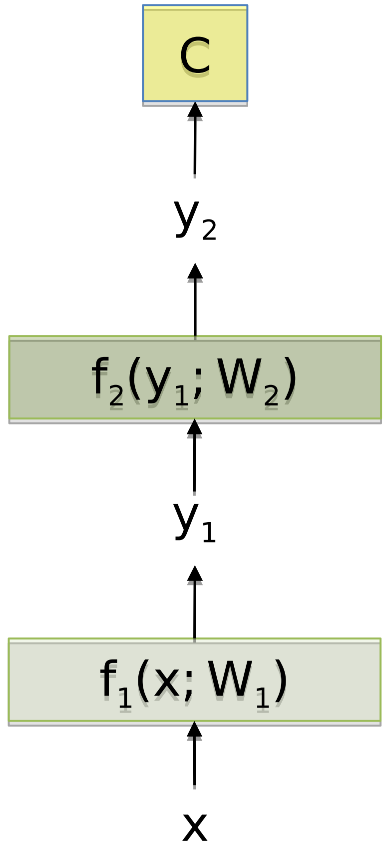
$$C = \text{Loss}(y, y_{GT})$$

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



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

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

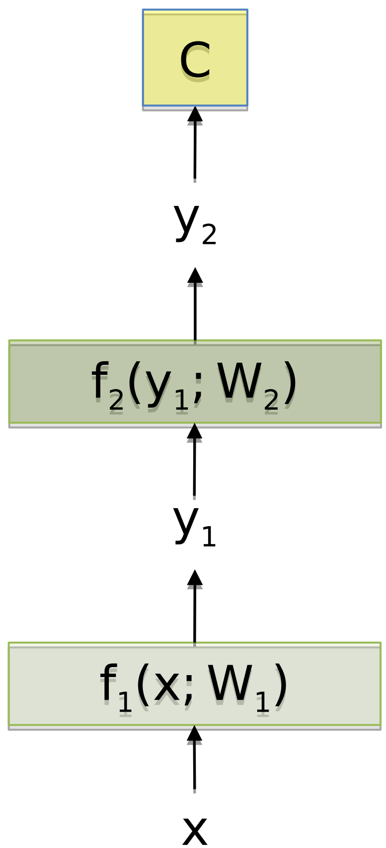
$$C = \mathrm{Loss}(y_2, y_{GT})$$

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



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

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

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

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

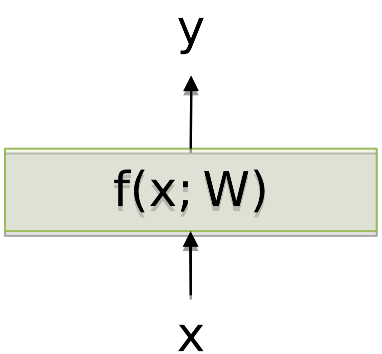
$$\frac{\partial C}{\partial W_2} = \left(\frac{\partial C}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial W_2}\right)$$

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

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

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:

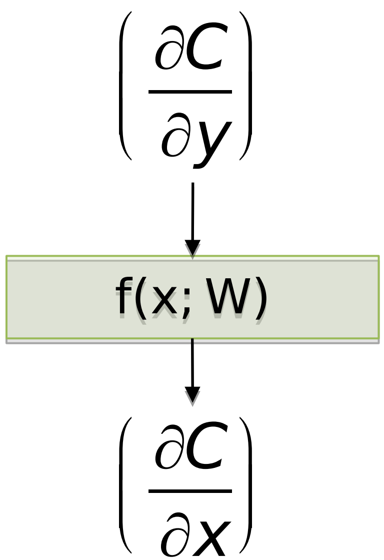
$\left(\frac{\partial y}{\partial W}\right)$ - How does output change due to params

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

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

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:

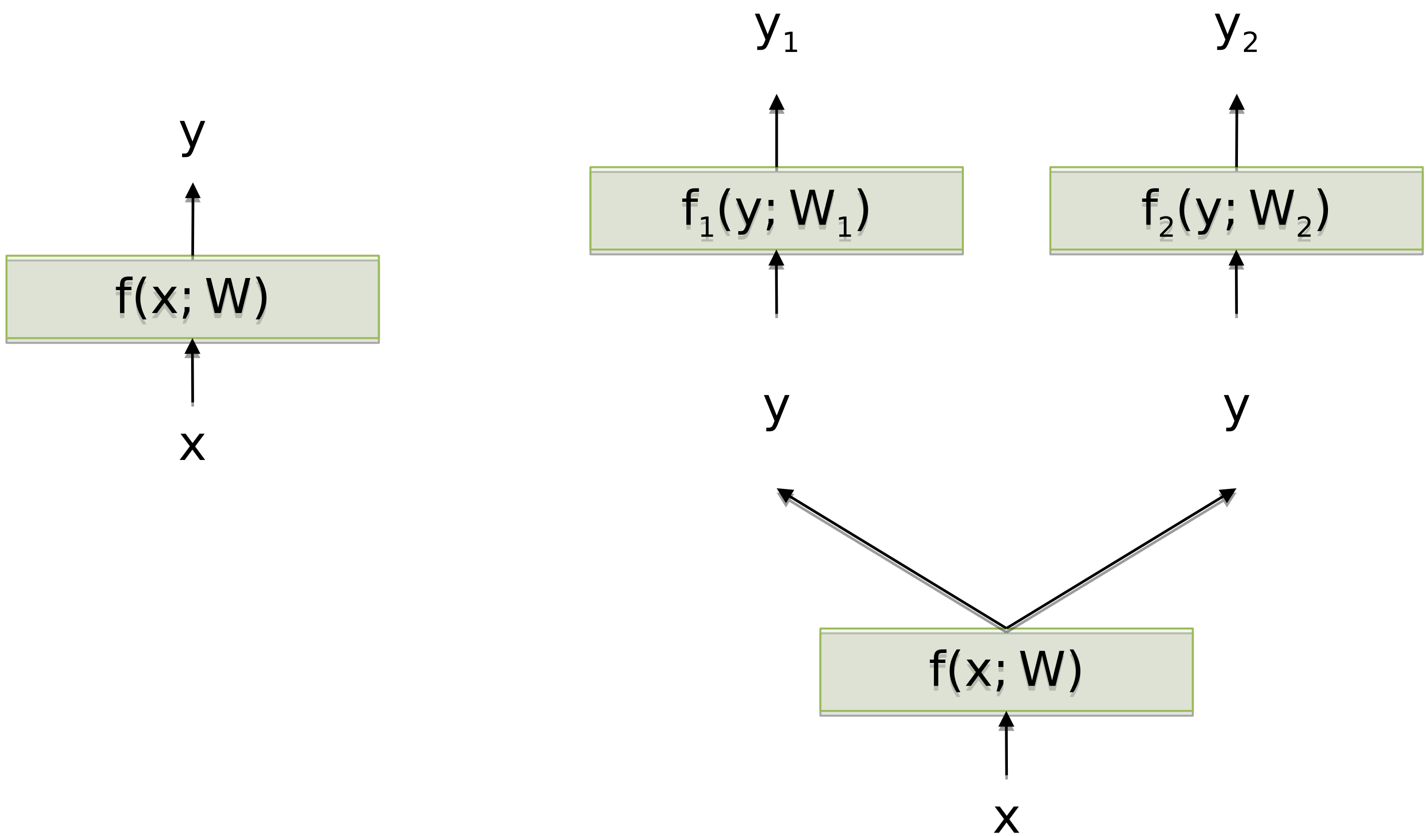
$\left(\frac{\partial y}{\partial W}\right)$ - How does output change due to params

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

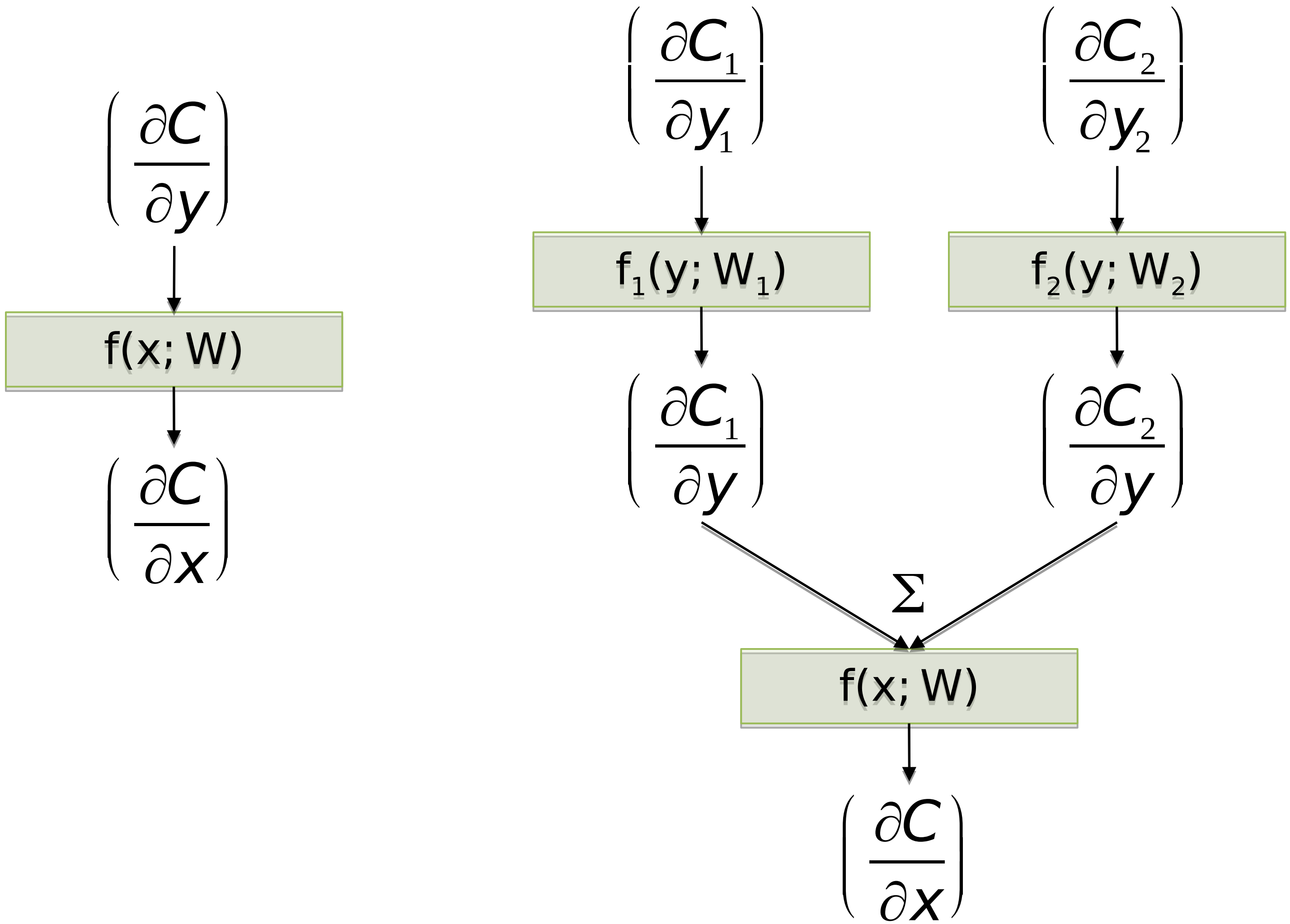
$$\left(\frac{\partial C}{\partial W}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial W}\right) \qquad \left(\frac{\partial C}{\partial x}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial x}\right)$$

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

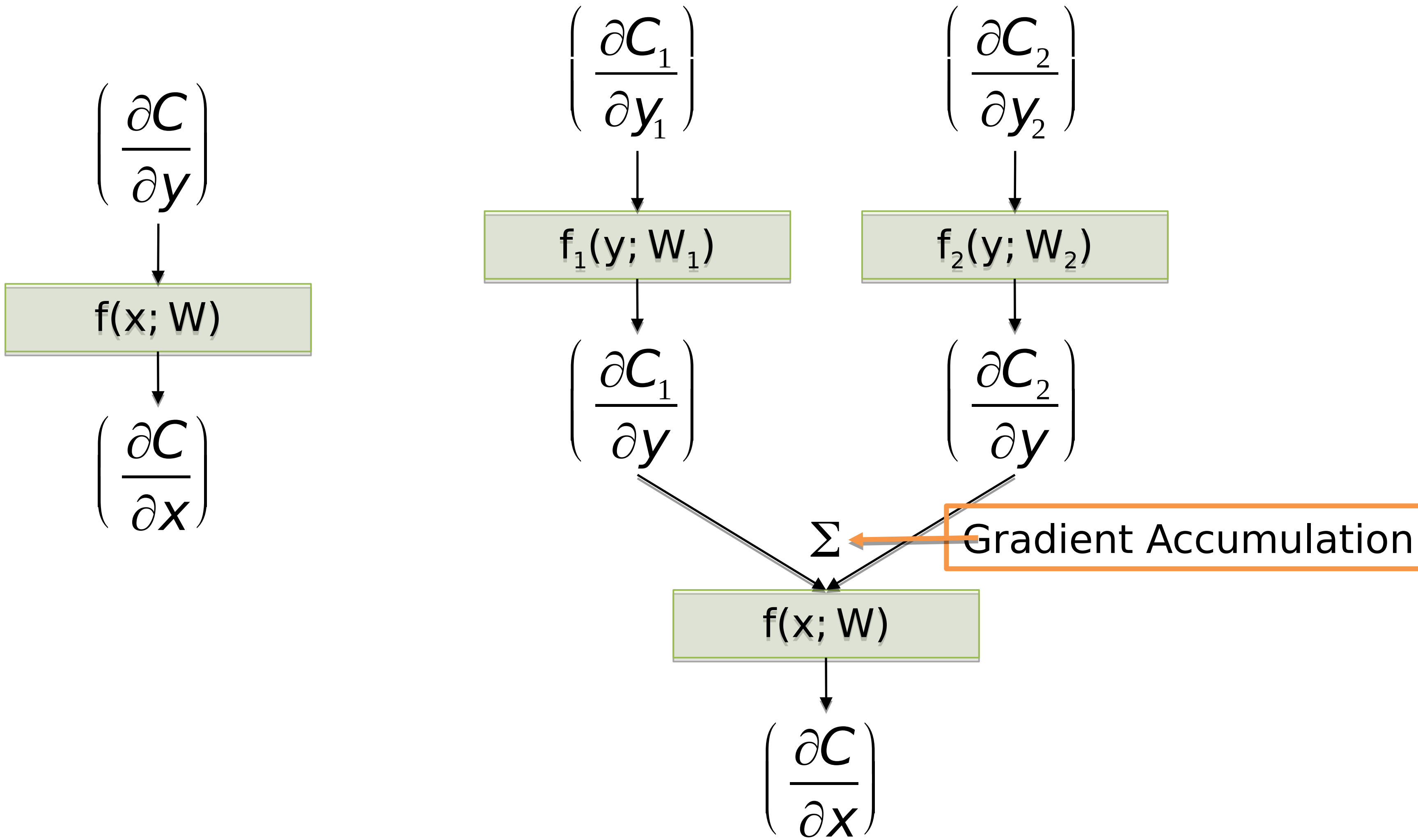
Extension to Computational Graphs



Extension to Computational Graphs



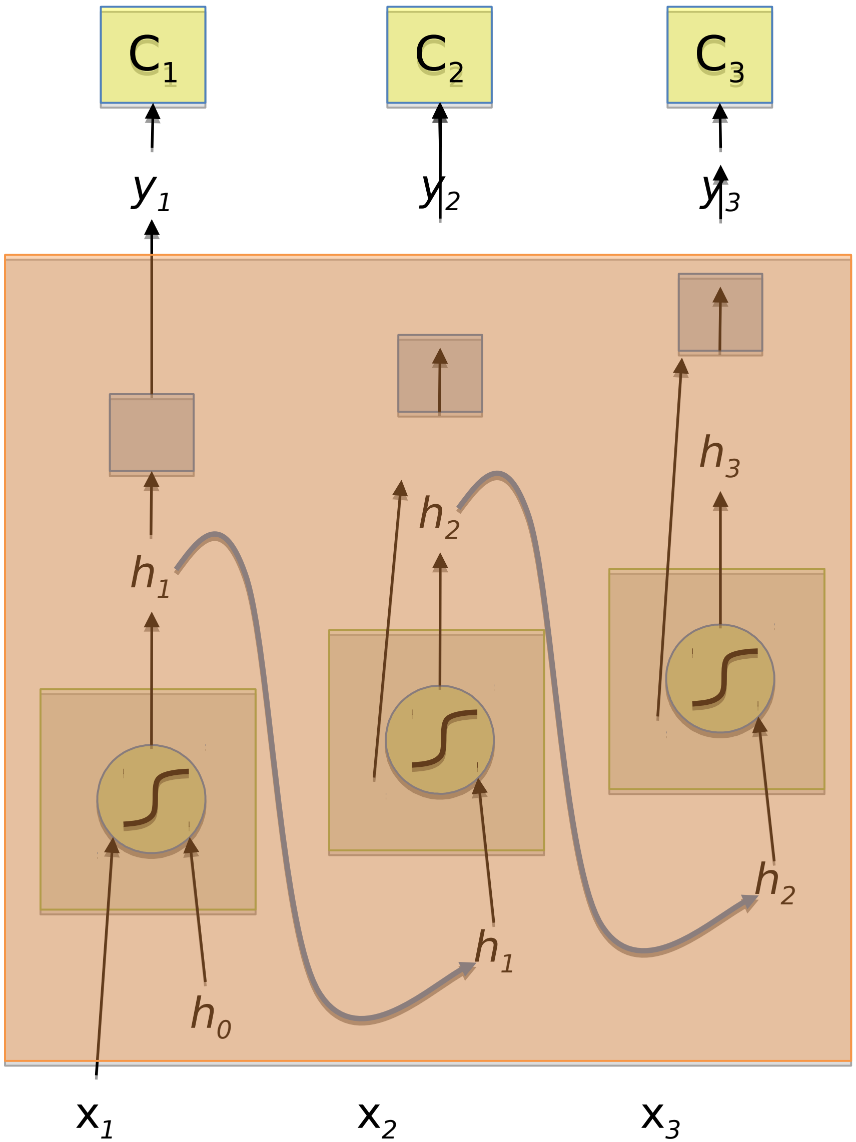
Extension to Computational Graphs



BackPropagation Through Time (BPTT)

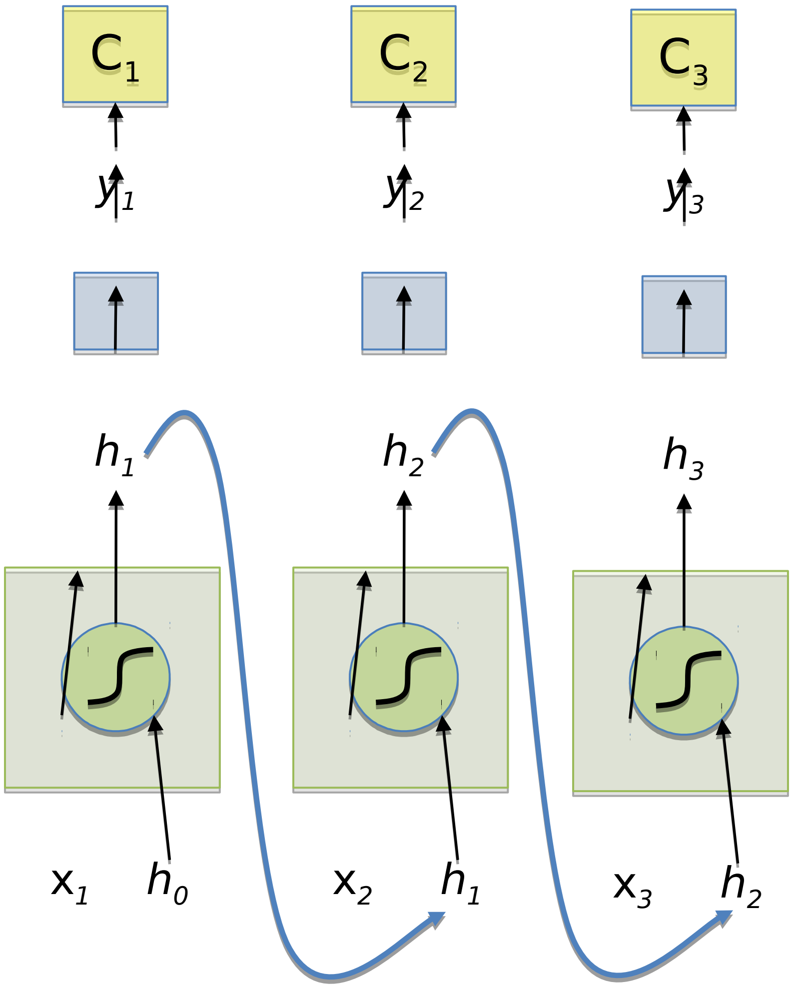
- One of the methods used to train RNNs
- The unfolded network (used during forward pass) is treated as one big feed-forward network
- This unfolded network accepts the whole time series as input
-

The Unfolded Vanilla RNN

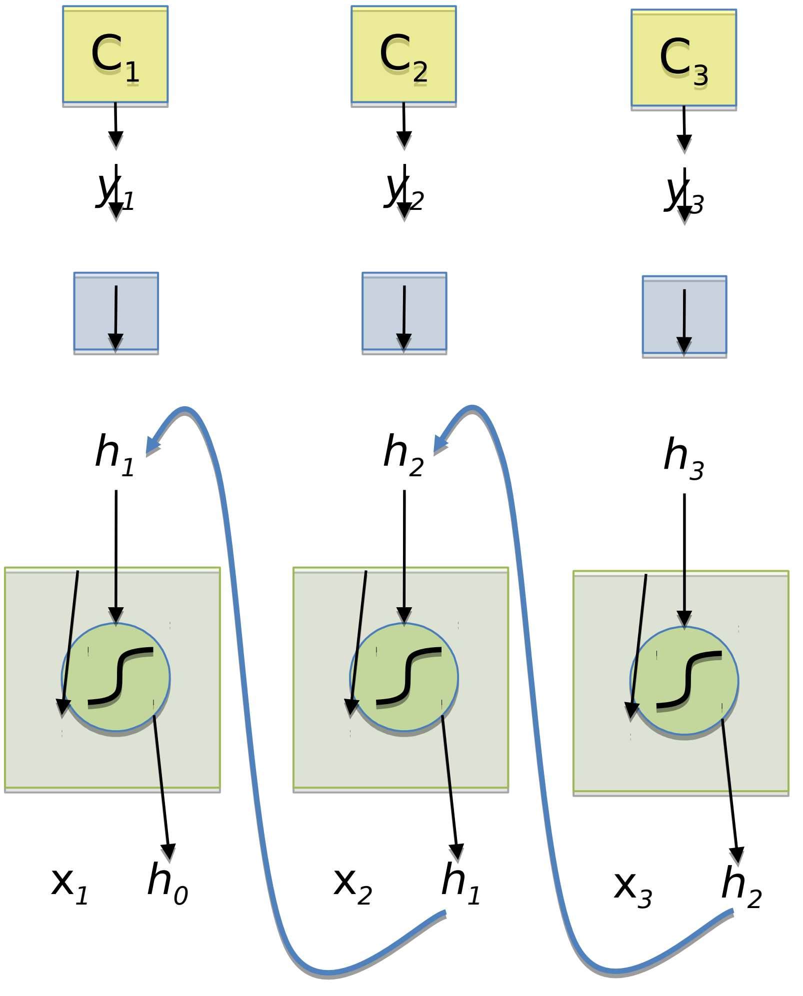


- Treat the unfolded network as one big feed-forward network!
- This big network takes in entire sequence as an input
- Compute gradients through the usual backpropagation
- Update shared weights

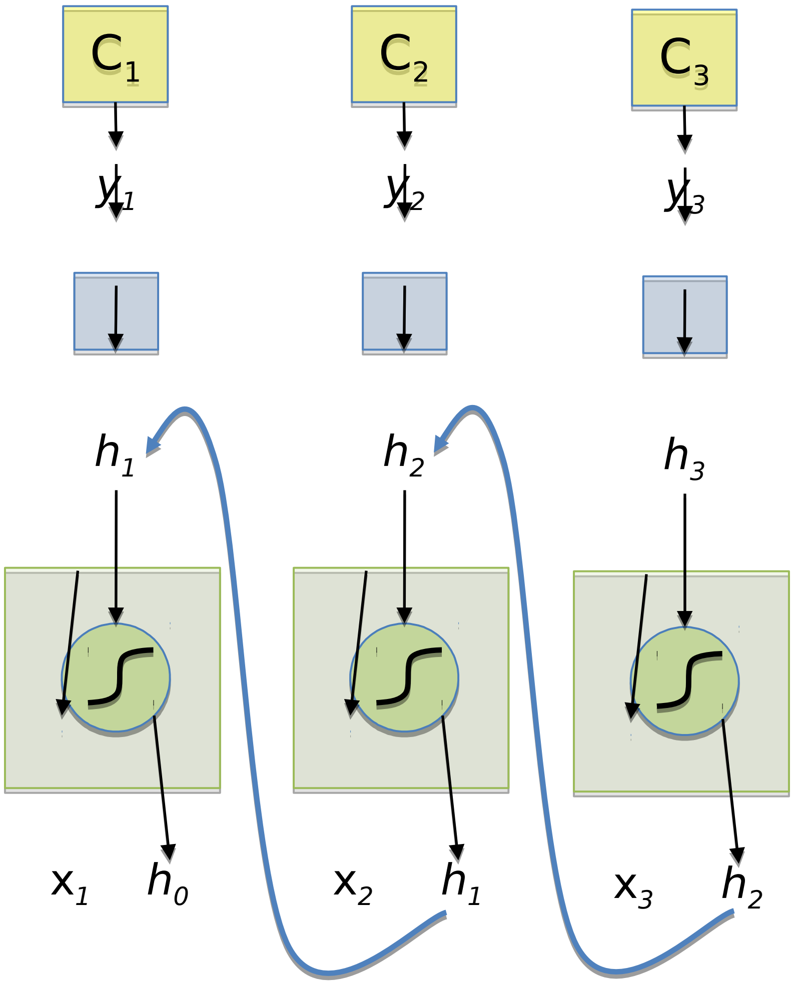
The Unfolded Vanilla RNN Forward



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]
-

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

The Identity Relationship

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

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

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

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

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

- Suppose that instead of a matrix multiplication, we had an identity relationship between the hidden states $h_t = h_{t-1} + F(x_t)$

$$\Rightarrow \left(\frac{\partial h_t}{\partial h_{t-1}}\right) = 1$$

- The gradient does not decay as the error is propagated all the way back aka “Constant Error Flow”

The Identity Relationship

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

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

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

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

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

- Suppose that instead of a matrix multiplication, we had an identity relationship between the hidden states $h_t = h_{t-1} + F(x_t)$

Remember Resnets?

$$\Rightarrow \left(\frac{\partial h_t}{\partial h_{t-1}}\right) = 1$$

- The gradient does not decay as the error is propagated all the way back aka "Constant Error Flow"

Disclaimer

- The explanations in the previous few slides are handwavy

- For rigorous proofs and derivations, please refer to
  On the difficulty of training recurrent neural networks, Pascanu *et al*., 2013
  Long Short-Term Memory, Hochreiter *et al*., 1997
  And other sources

Long Short-Term Memory (LSTM)[1]

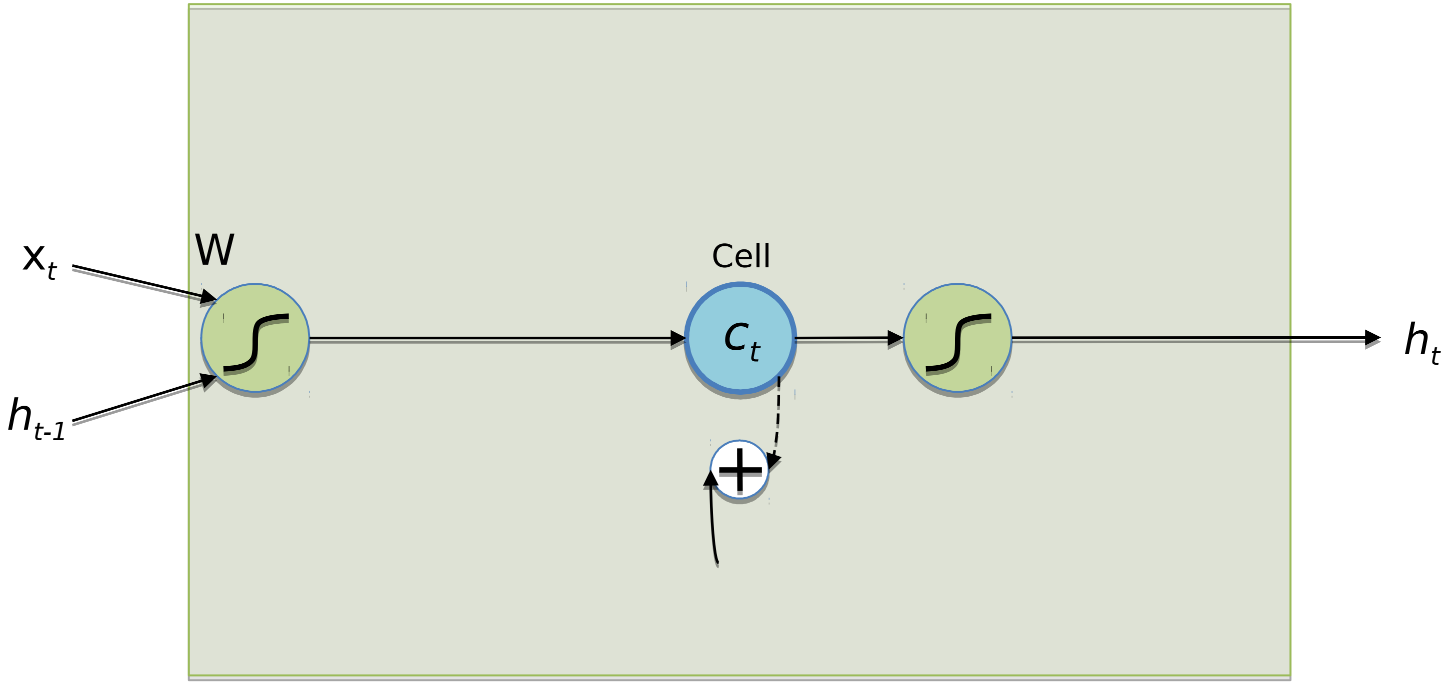
- The LSTM uses this idea of “Constant Error Flow” for RNNs to create a “Constant Error Carousel” (CEC) which ensures that gradients don’t decay

- 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

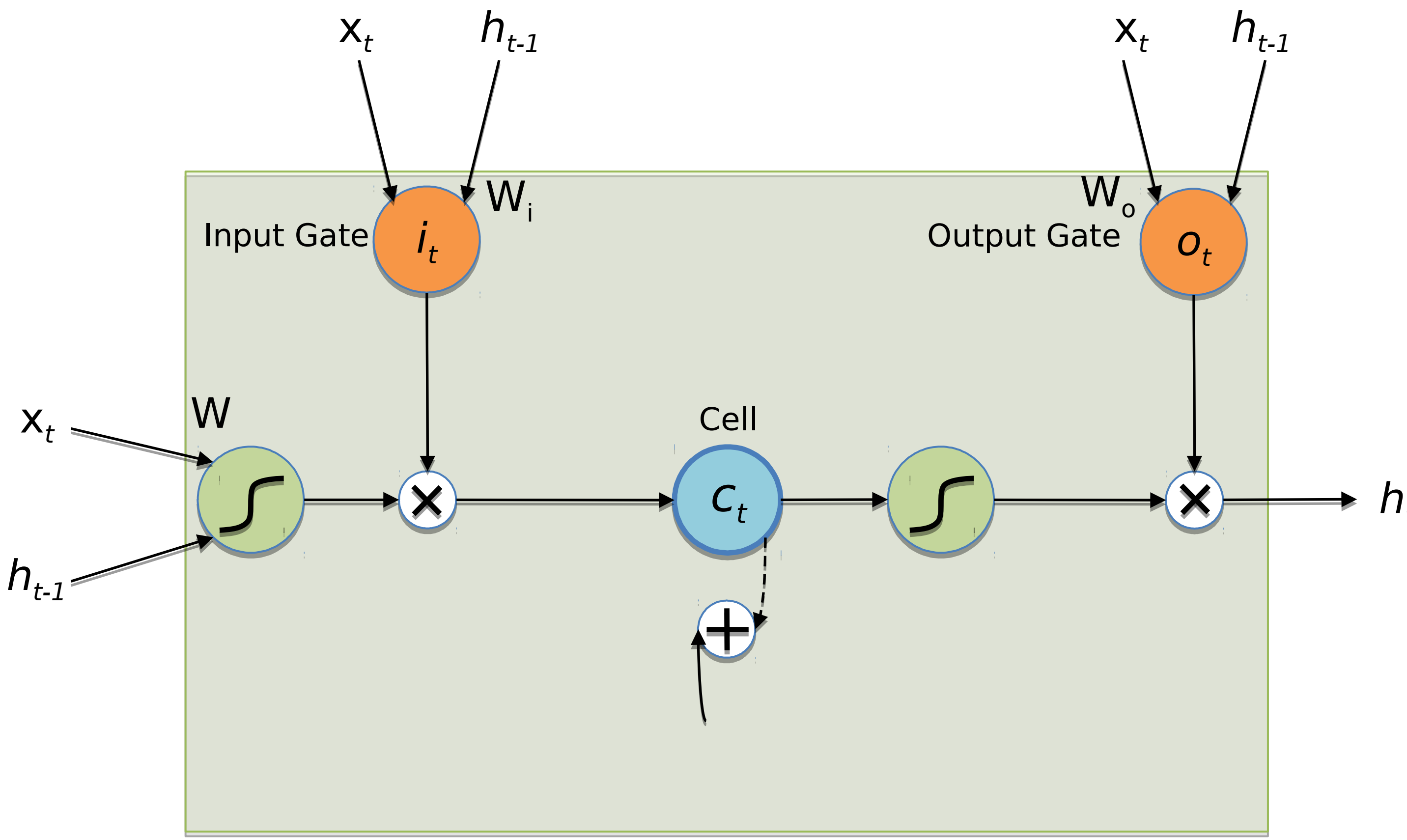
The LSTM Idea



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

* Dashed line indicates time-lag

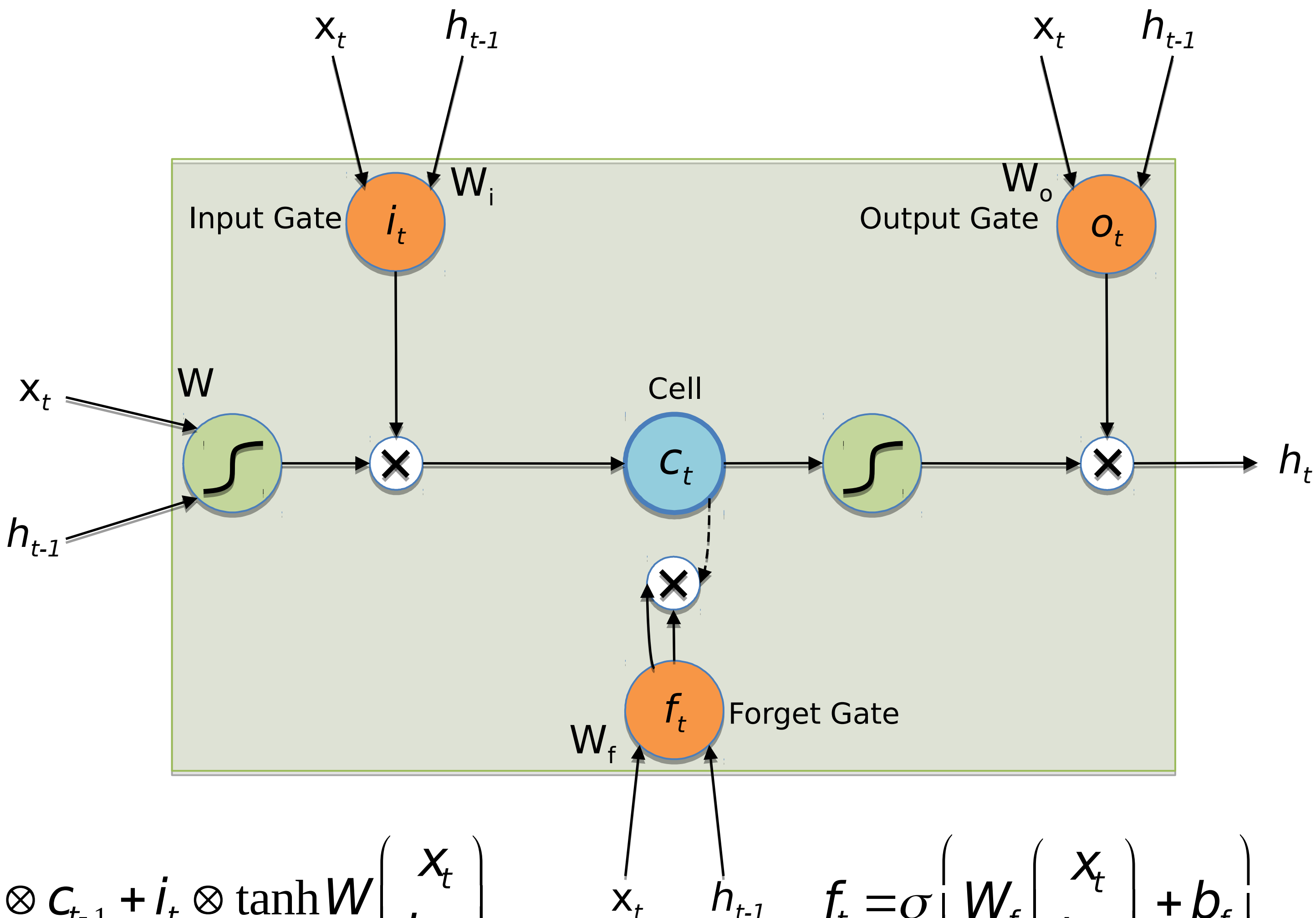
The Original LSTM Cell



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

Similarly for o_t

The Popular LSTM Cell



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

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

LSTM – Forward/Backward

Go To: Illustrated LSTM Forward and Backward Pass

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 through the CEC
- Exploding gradients are handled by gradient clipping

- More complex architectures are listed in the course materials for you to read, understand, and present

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