RNN & LSTM

These slides are adapted from **"Recurrent Neural Networks"** by **Silvio Savarese**, available at:
https://slazebni.cs.illinois.edu/spring17/lec02_rnn.pdf.

I express my gratitude to the original author for creating and sharing these materials.

Motivation

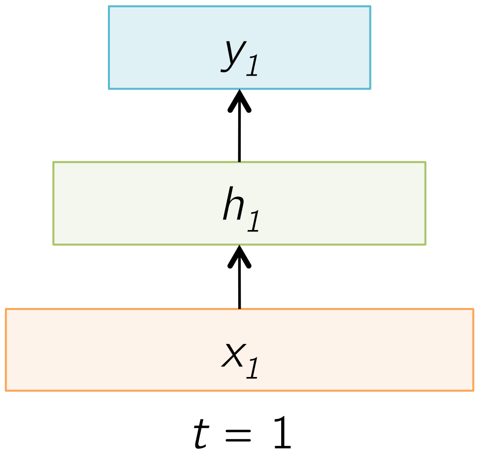
- Not all problems can be converted into one with fixed-length inputs and outputs
- Problems such as Speech Recognition or Time-series Prediction require a system to store and use context information
  - Simple case: Output YES if the number of 1s is even, else NO
    1000010101 – YES, 100011 – NO, …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

Recurrent Neural Networks (RNNs)

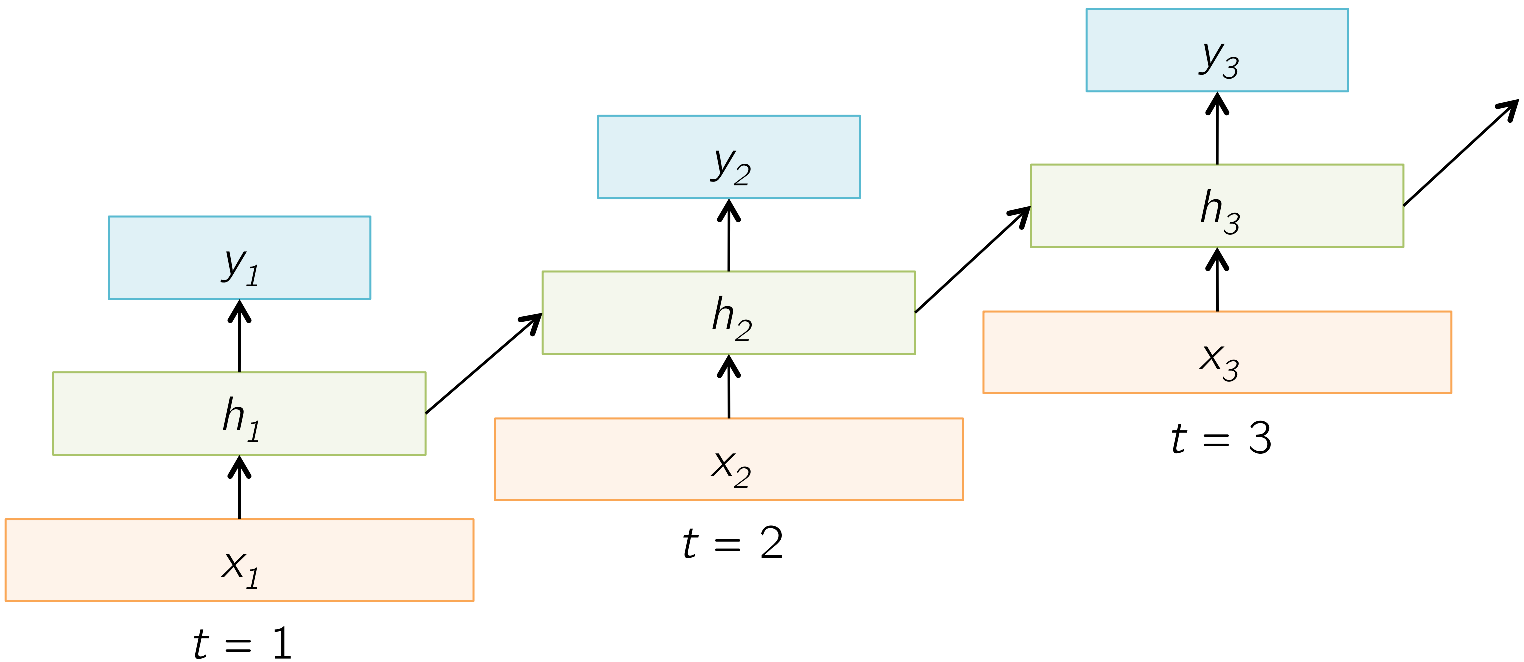
- **R**ecurrent **N**eural **N**etwork**s** take the previous output or hidden states as inputs.
  The composite input at time t has some historical information about the happenings at time $\mathrm{T} < t$

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

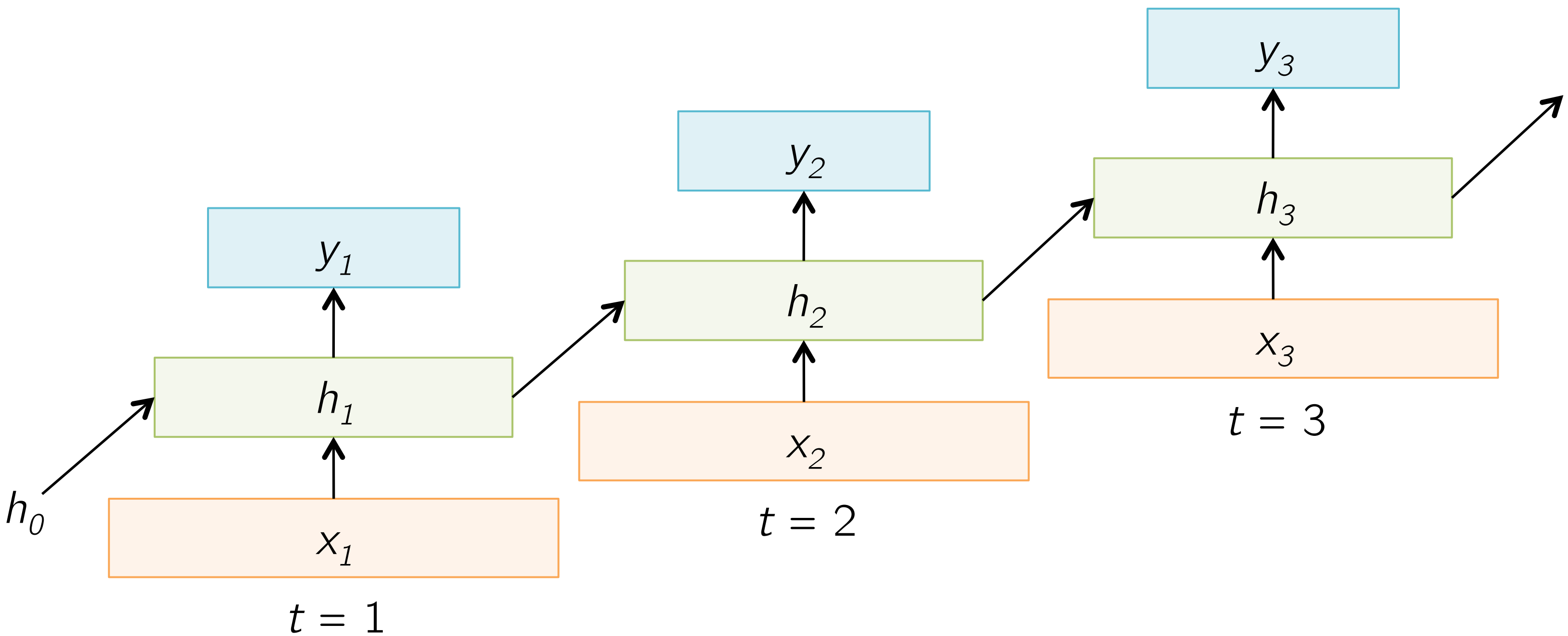
Sample Feed-forward Network



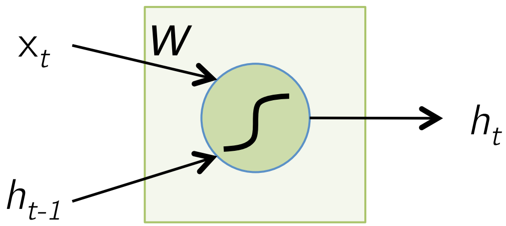
Sample RNN



Sample RNN

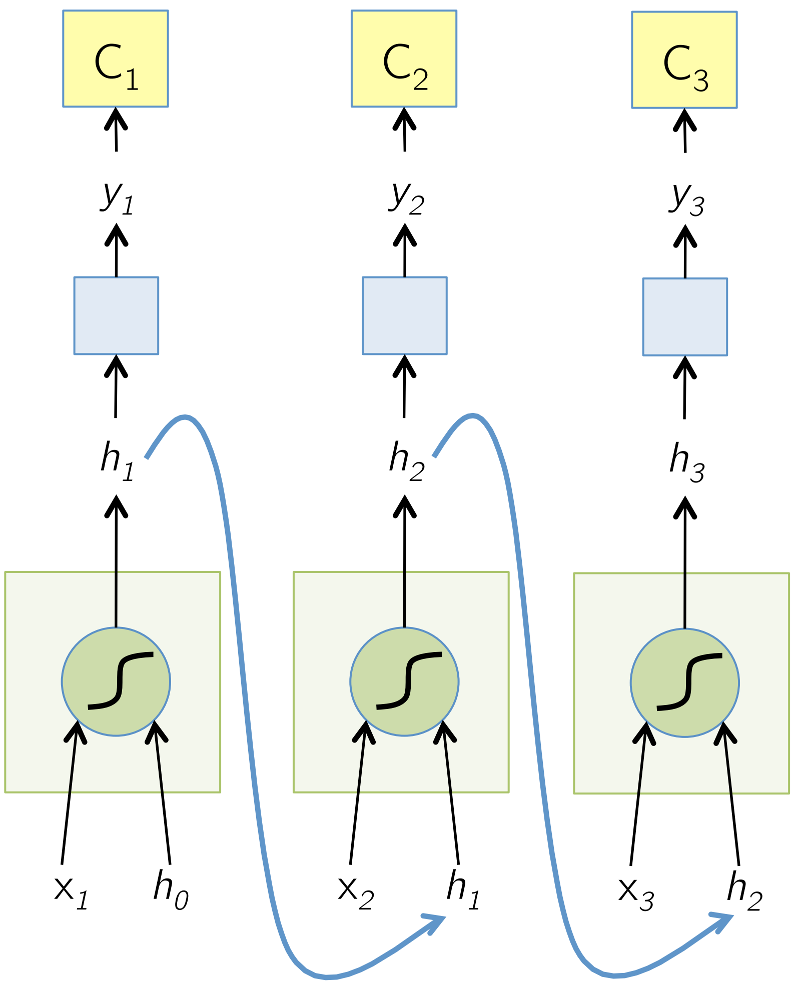


The Vanilla RNN Cell



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

The Vanilla RNN Forward

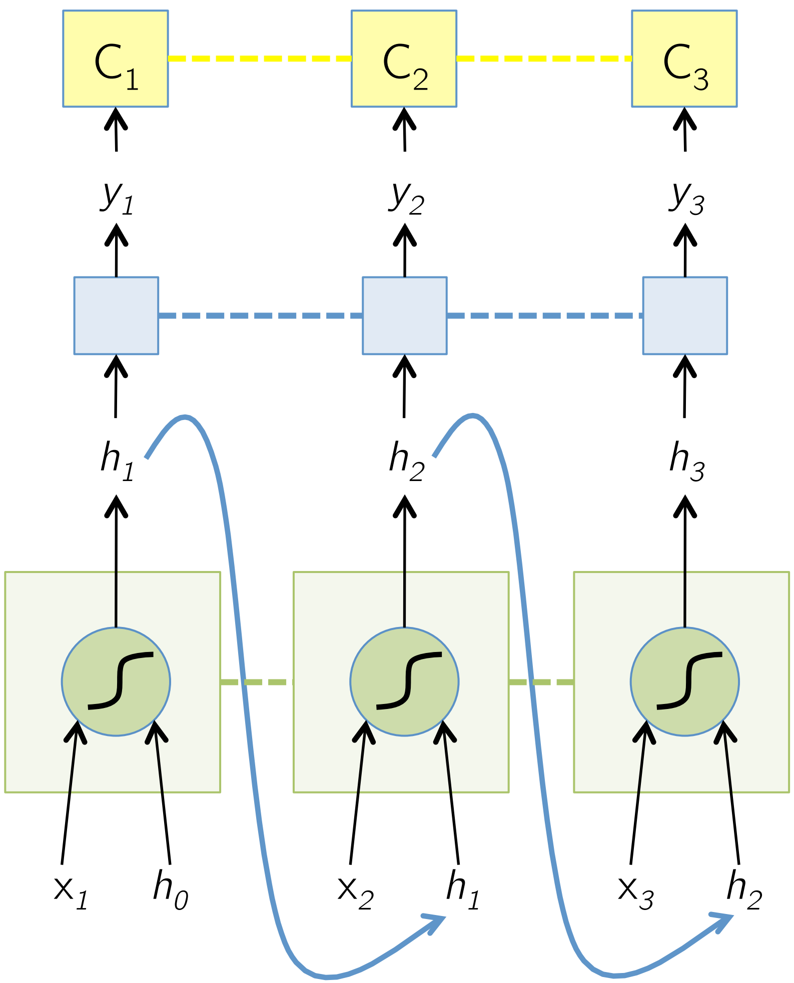


$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

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

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

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

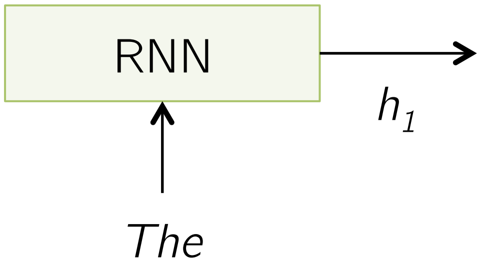
Sentiment Classification

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

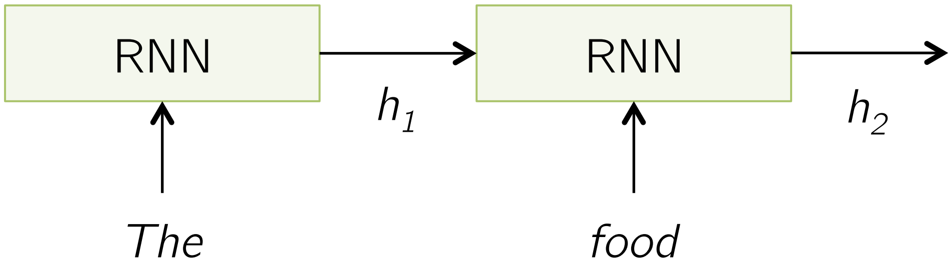
- **Inputs:** Multiple words, one or more sentences
- **Outputs:** Positive / Negative classification

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

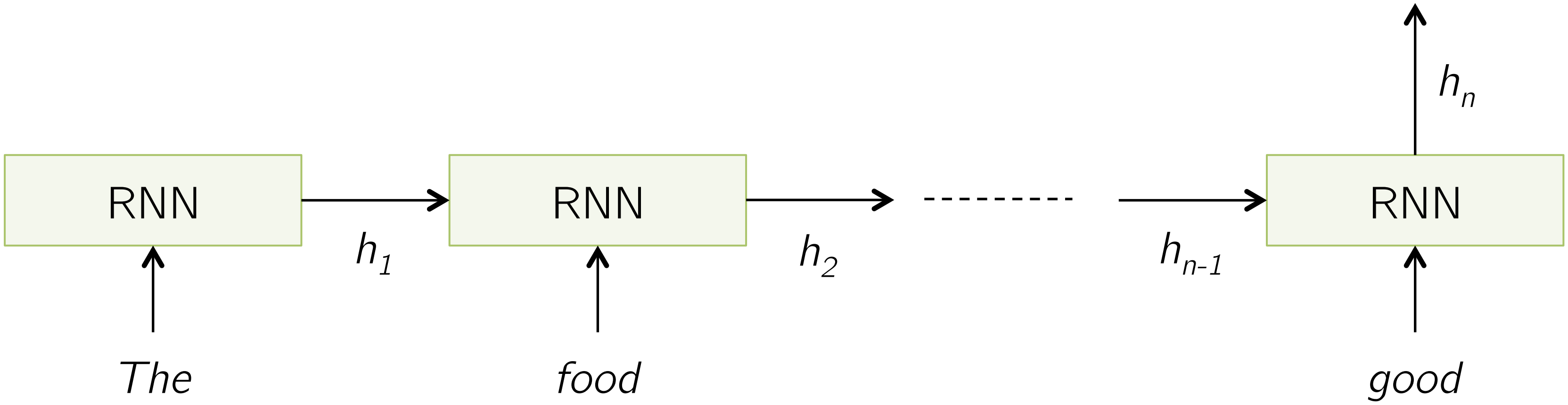
Sentiment Classification



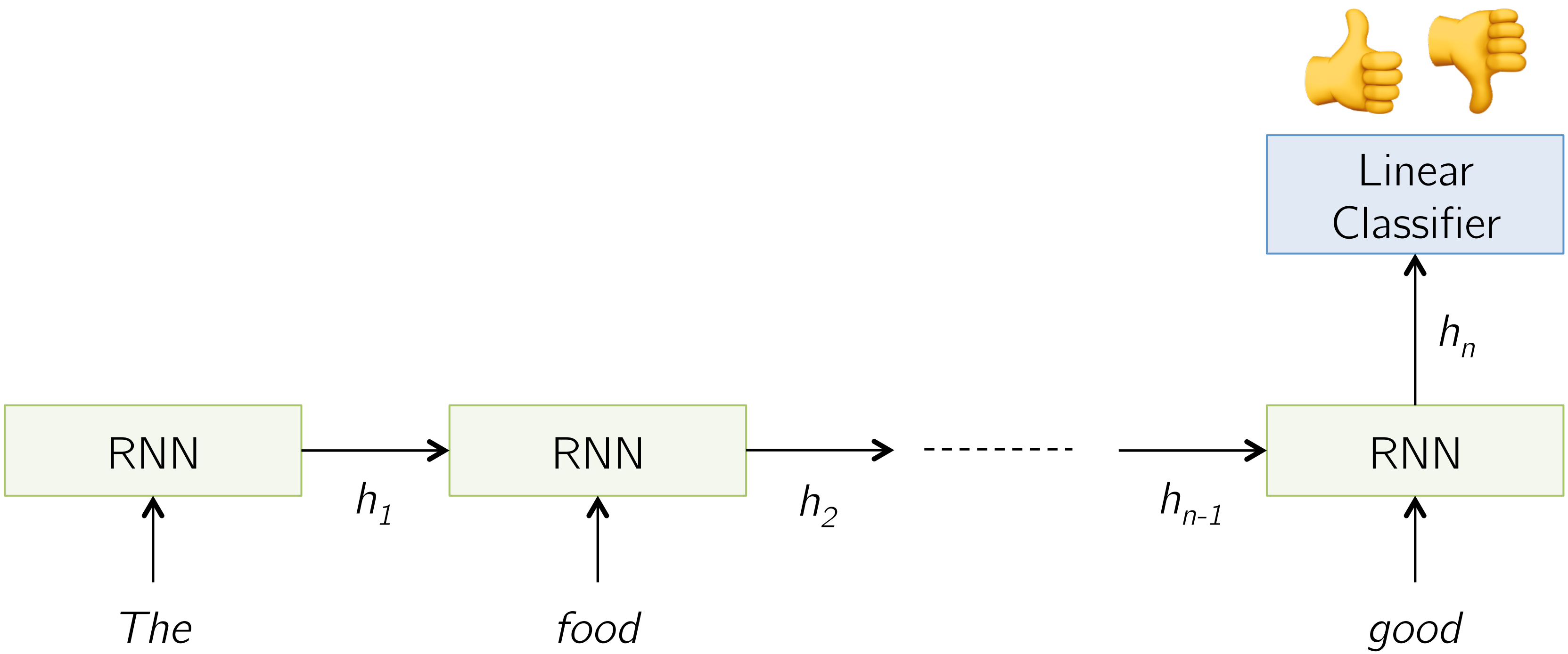
Sentiment Classification



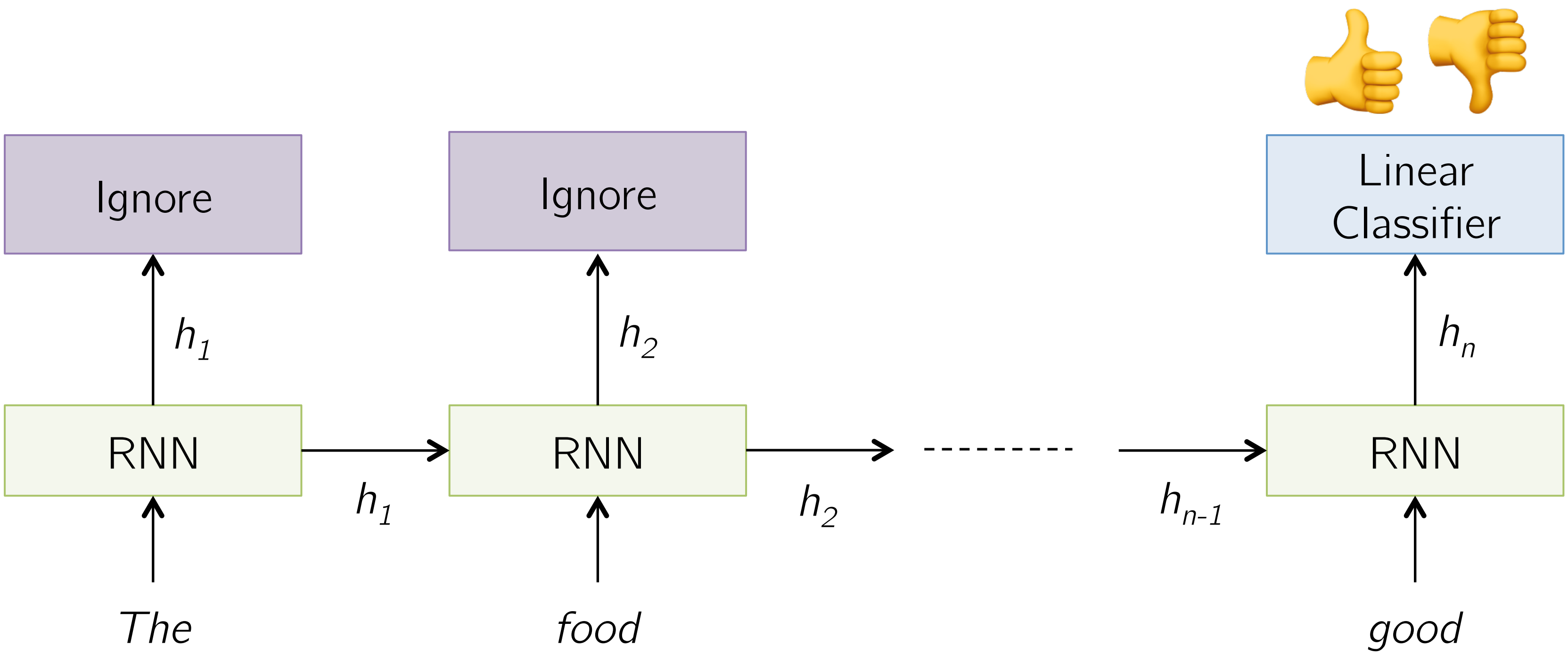
Sentiment Classification



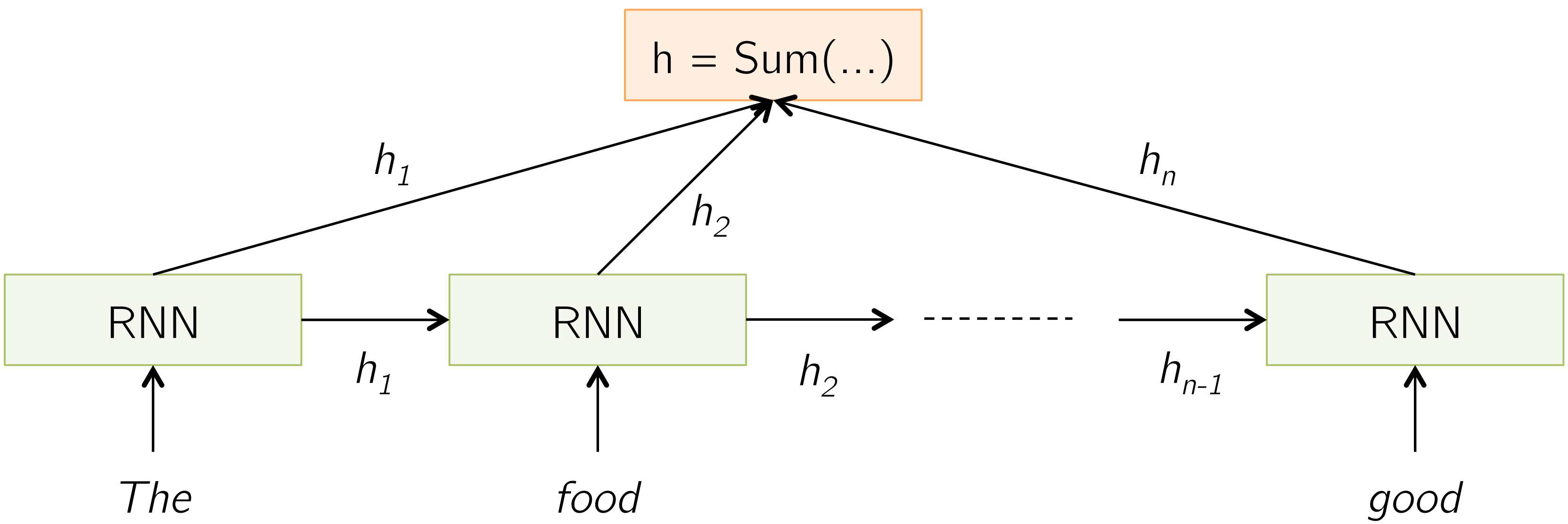
Sentiment Classification



Sentiment Classification

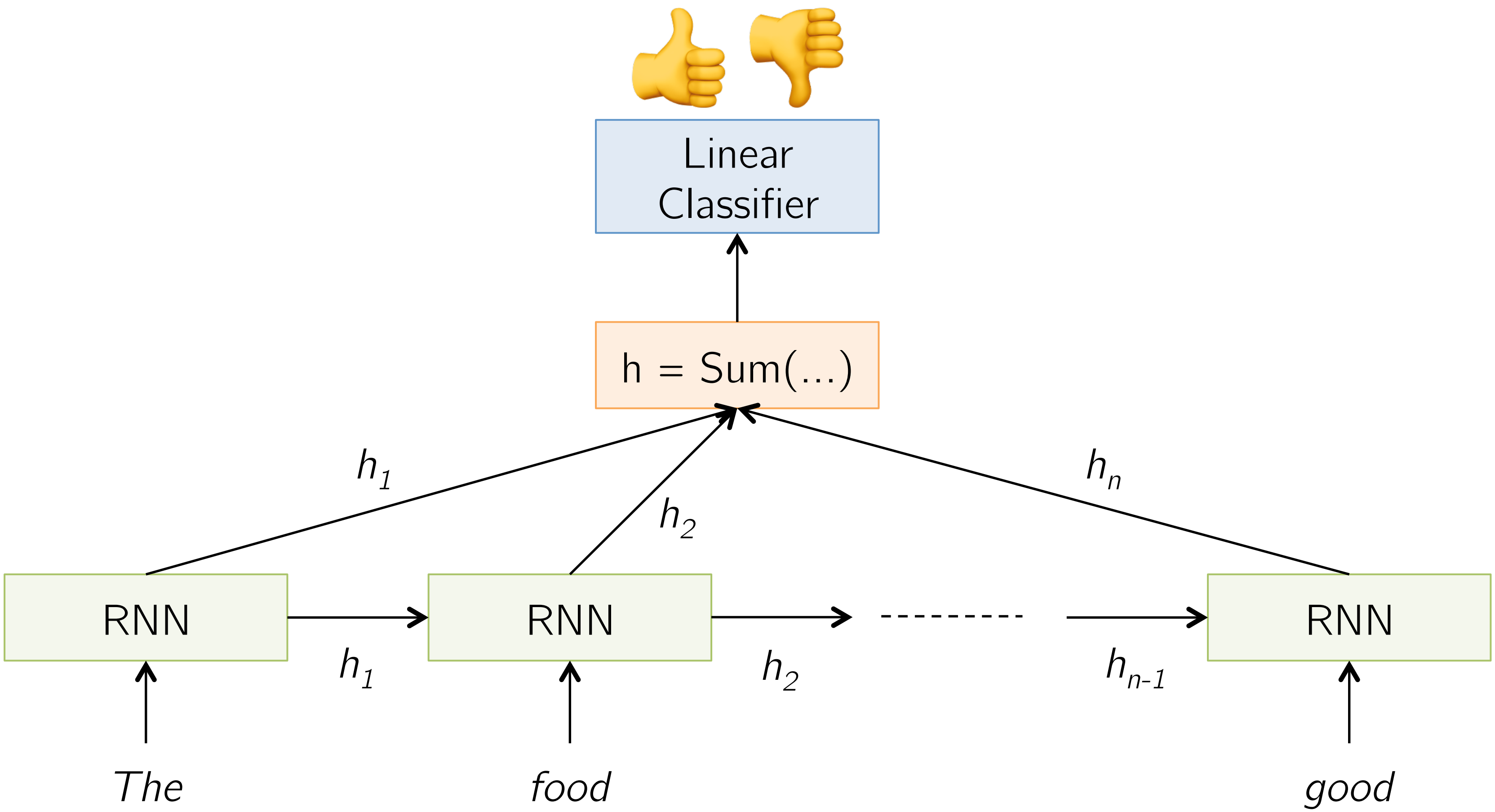


Sentiment Classification



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)

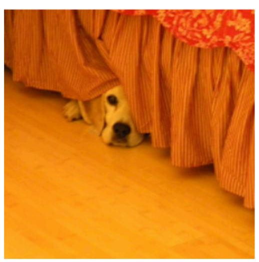

: The dog is hiding

Image Captioning

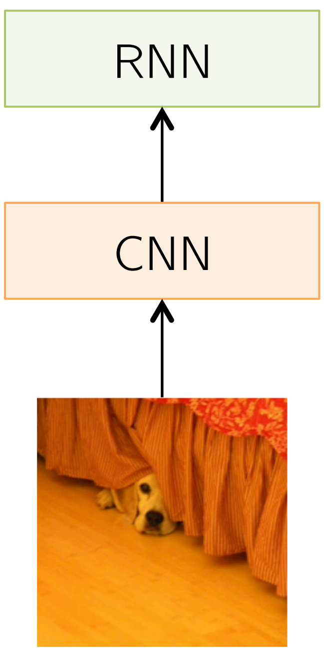


Image Captioning

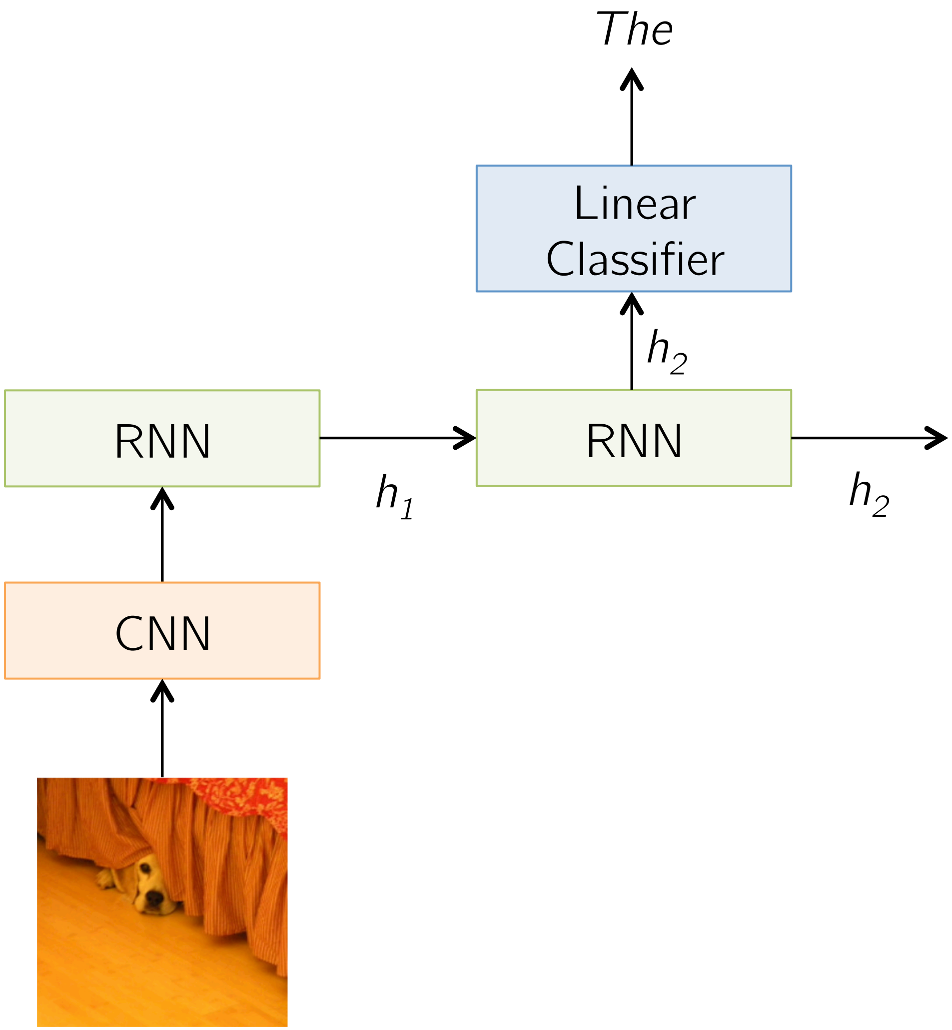
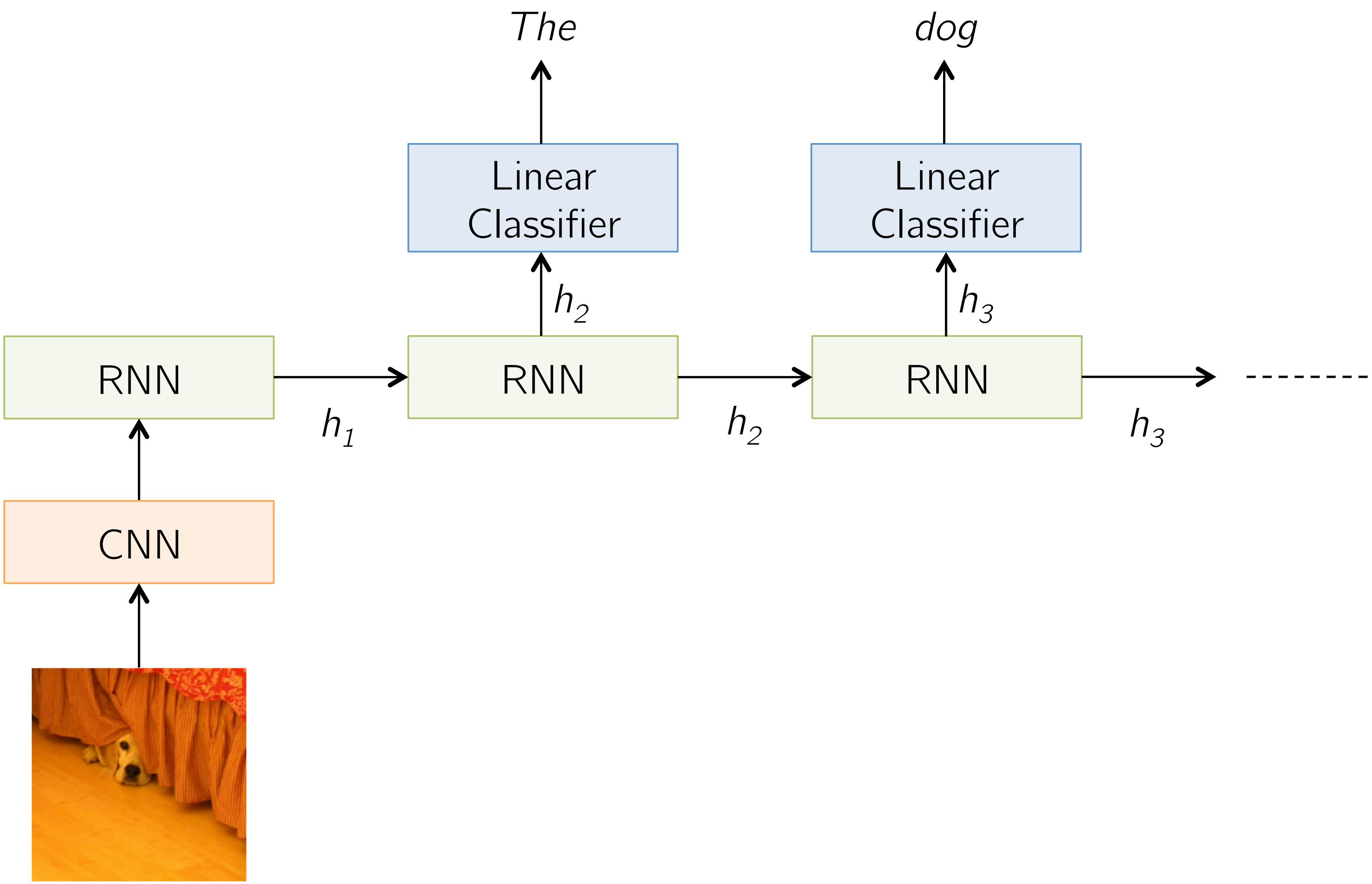
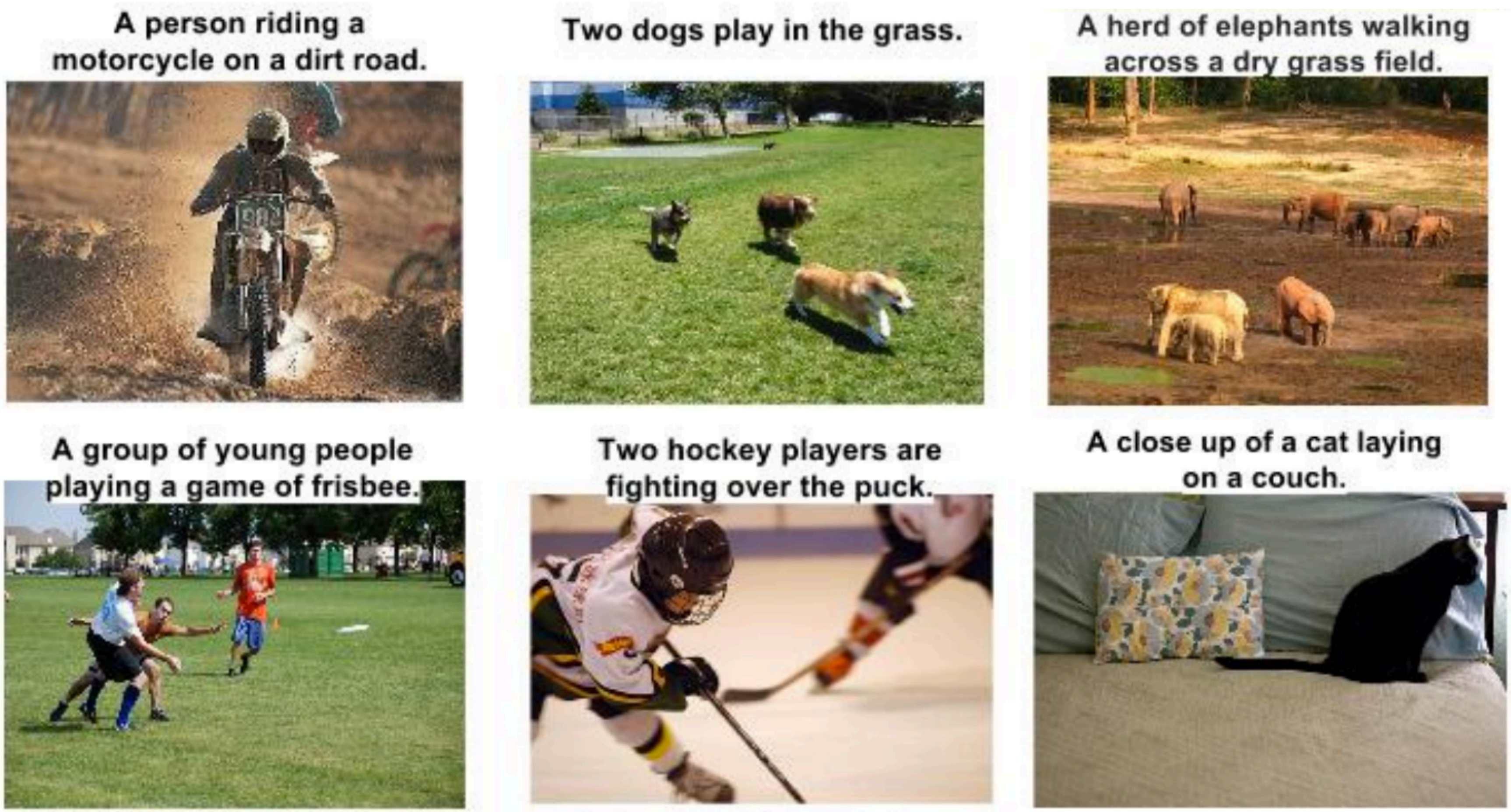


Image Captioning

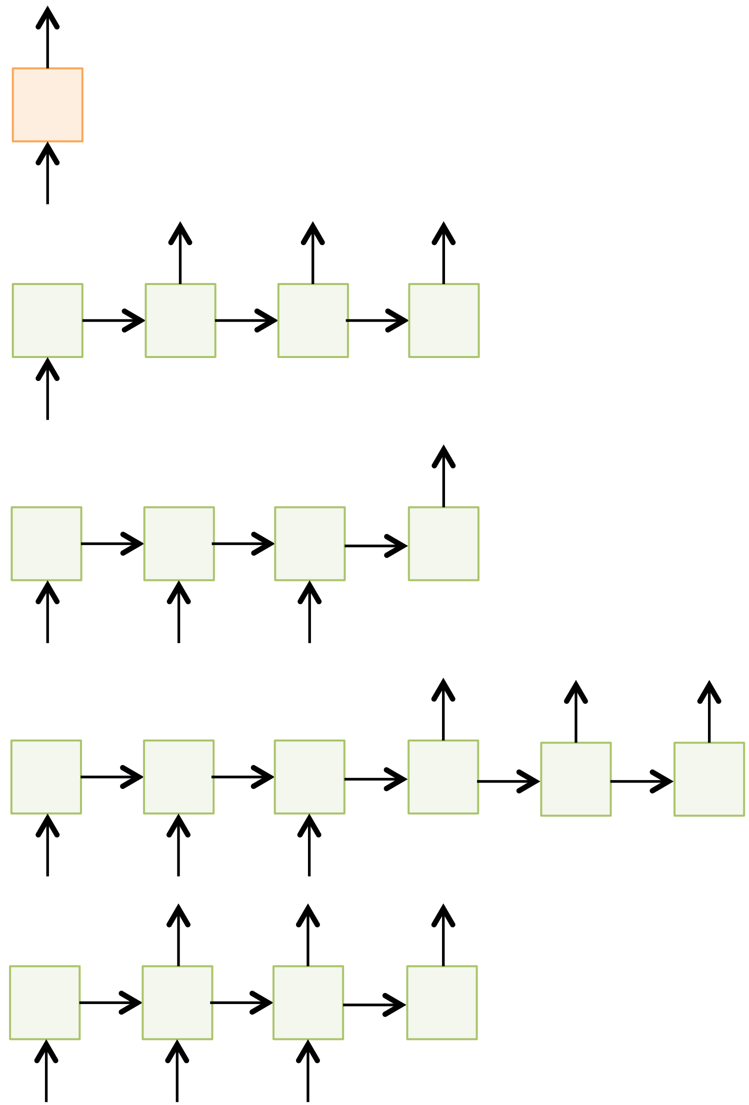


RNN Outputs: Image Captions



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

Input – Output Scenarios



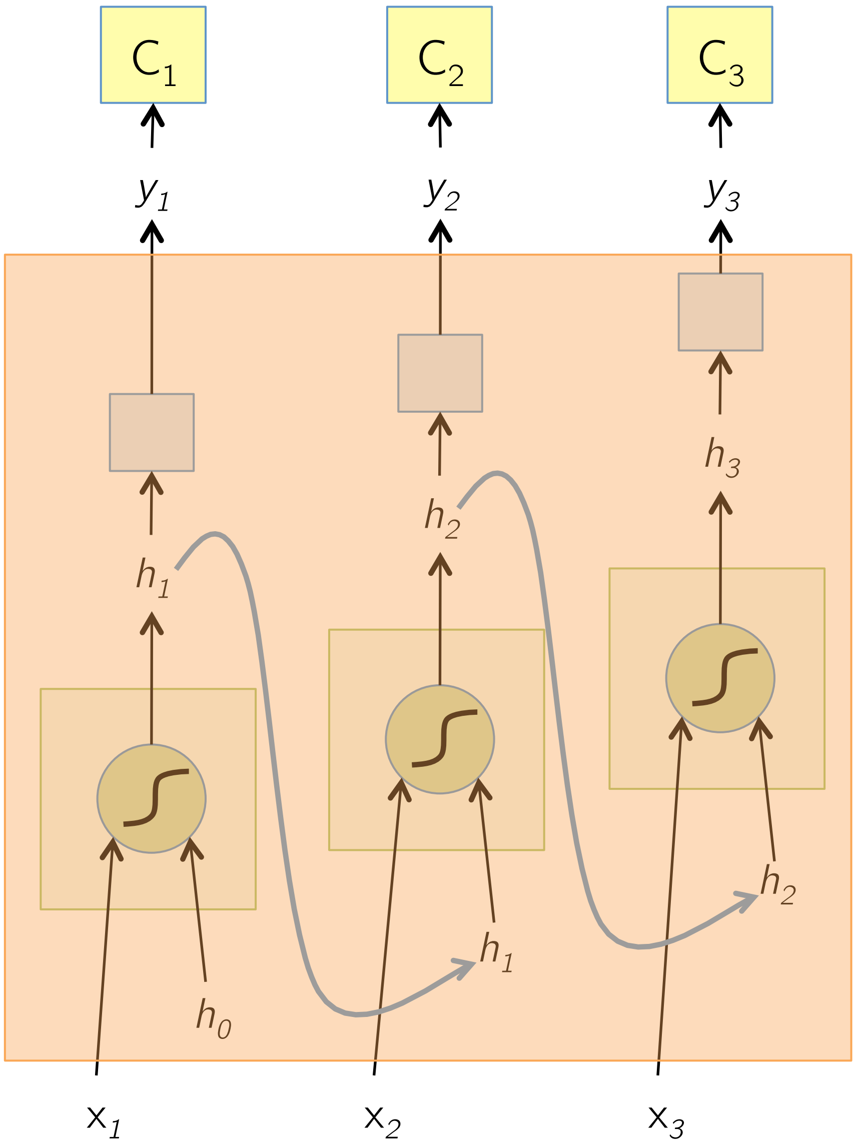
Single - Single	Feed-forward Network
Single - Multiple	Image Captioning
Multiple - Single	Sentiment Classification
Multiple - Multiple	Translation
	Image Captioning

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

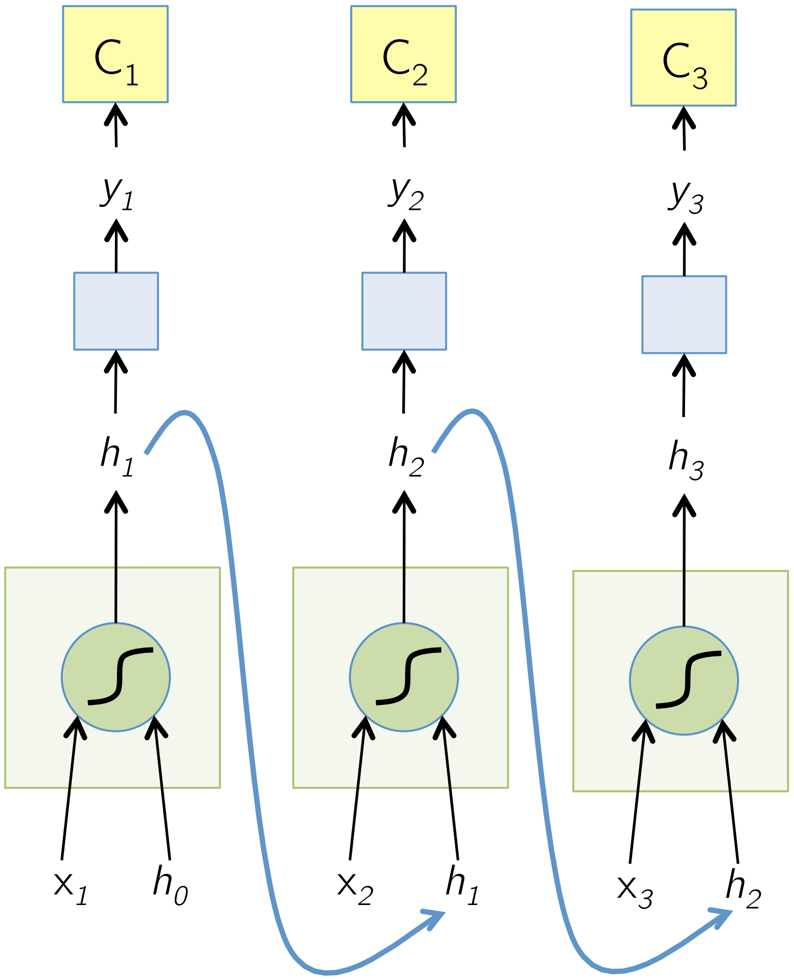
- The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and then applied to the RNN weights

The Unfolded Vanilla RNN

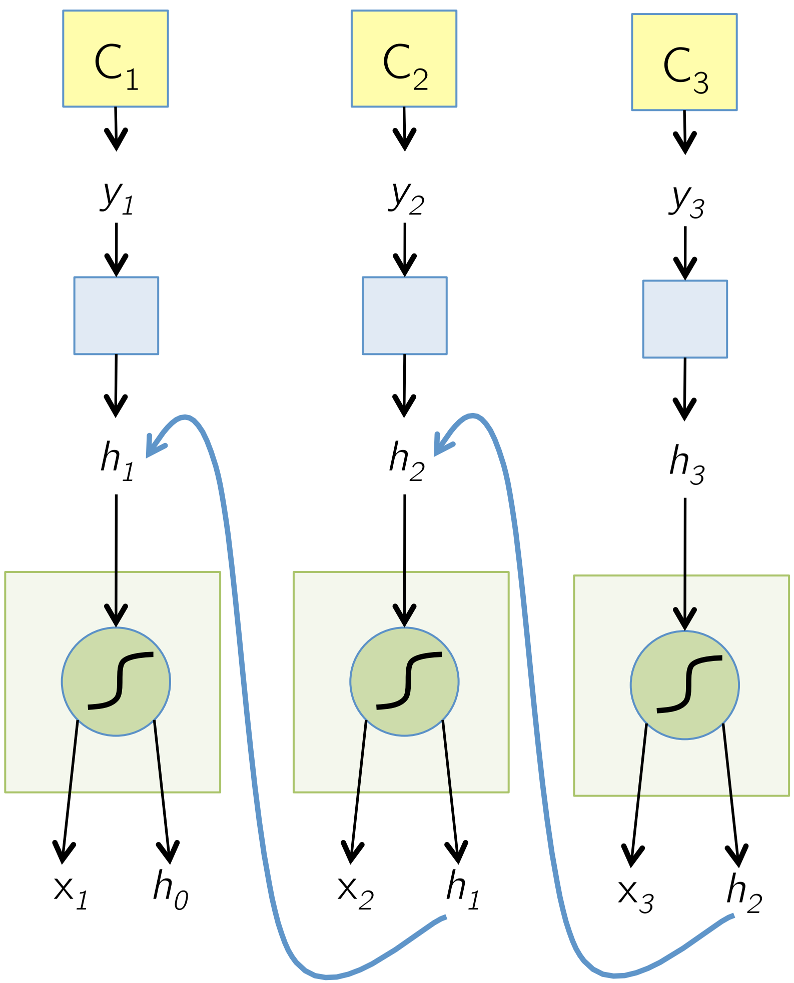


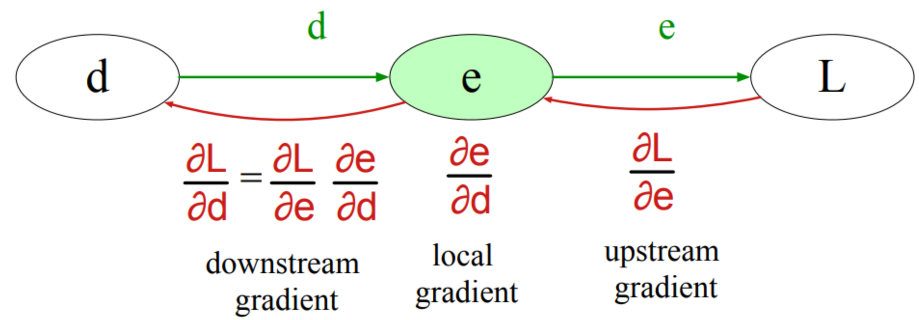
- Treat the unfolded network as one big feed-forward network!
- This big network takes in entire sequence as an input
- Compute gradients through the usual backpropagation
- Update shared weights

The Unfolded Vanilla RNN Forward



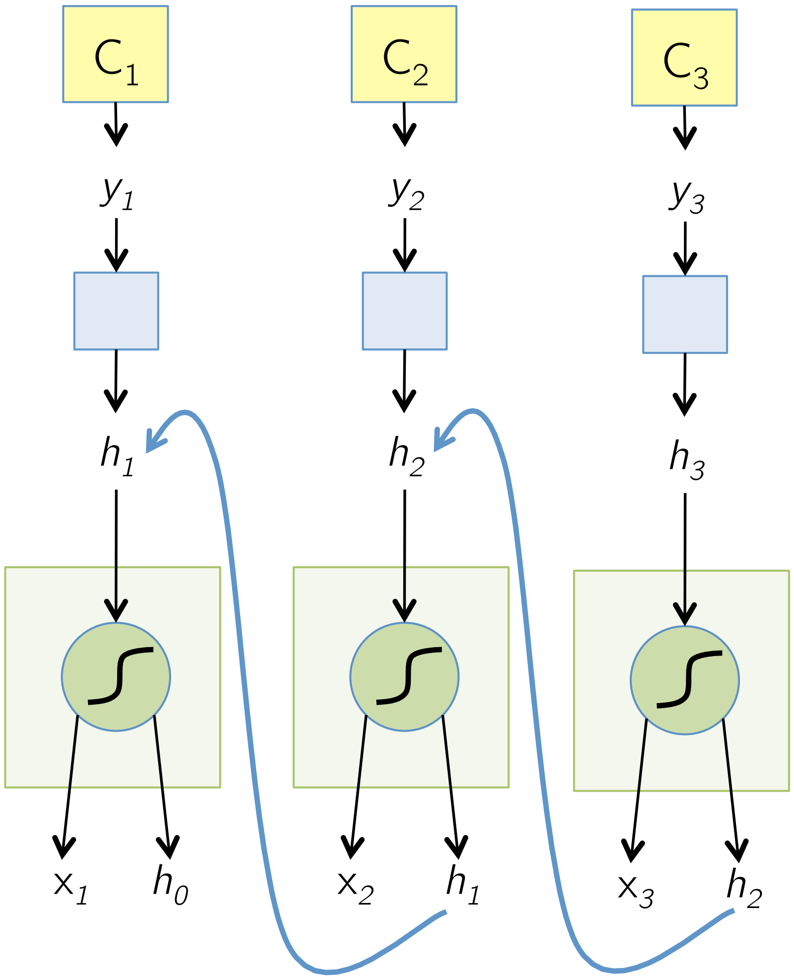
The Unfolded Vanilla RNN Backward





$$\frac{\partial C_3}{\partial h_1} = \frac{\partial C_3}{\partial y_3} \cdot \frac{\partial y_3}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1}.$$

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

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"

Long Short-Term Memory (LSTM)[1]

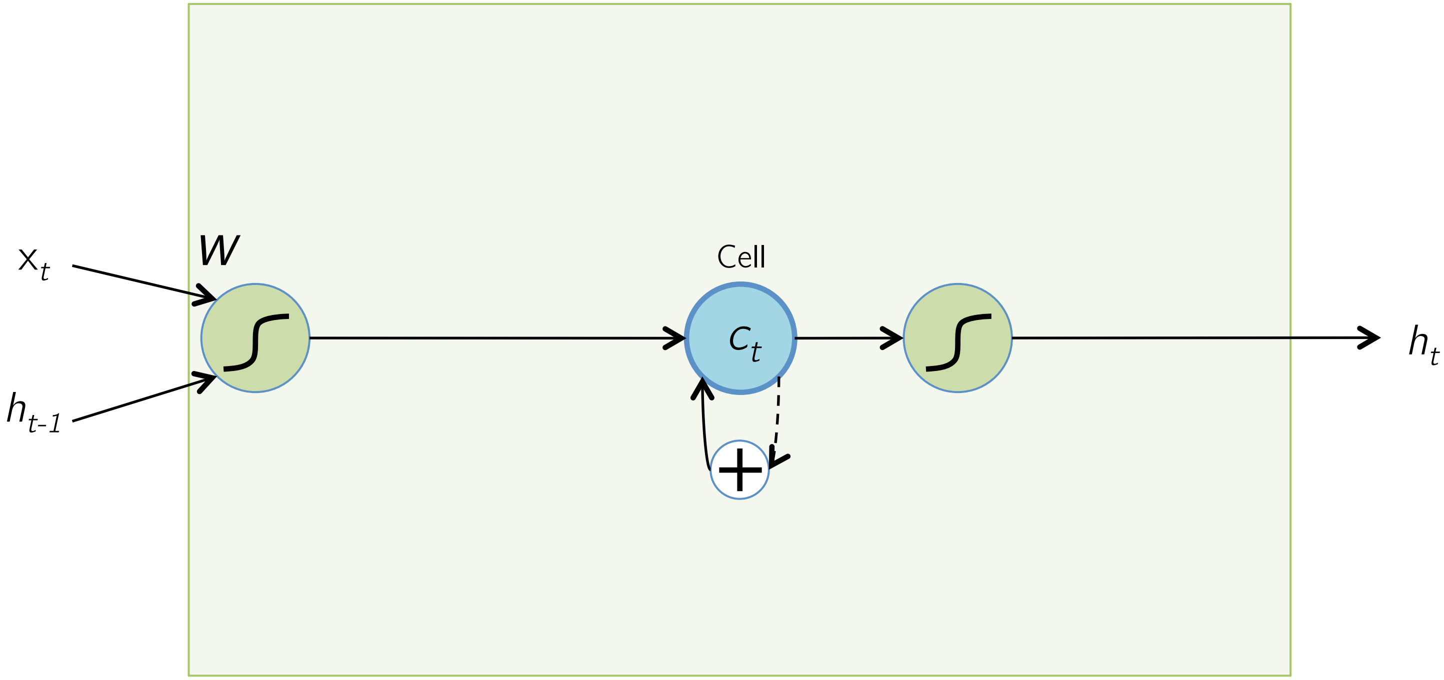
- The LSTM uses this idea of “Constant Error Flow” for RNNs to create a “Constant Error Carousel” (CEC) which ensures that gradients don’t decay

- 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

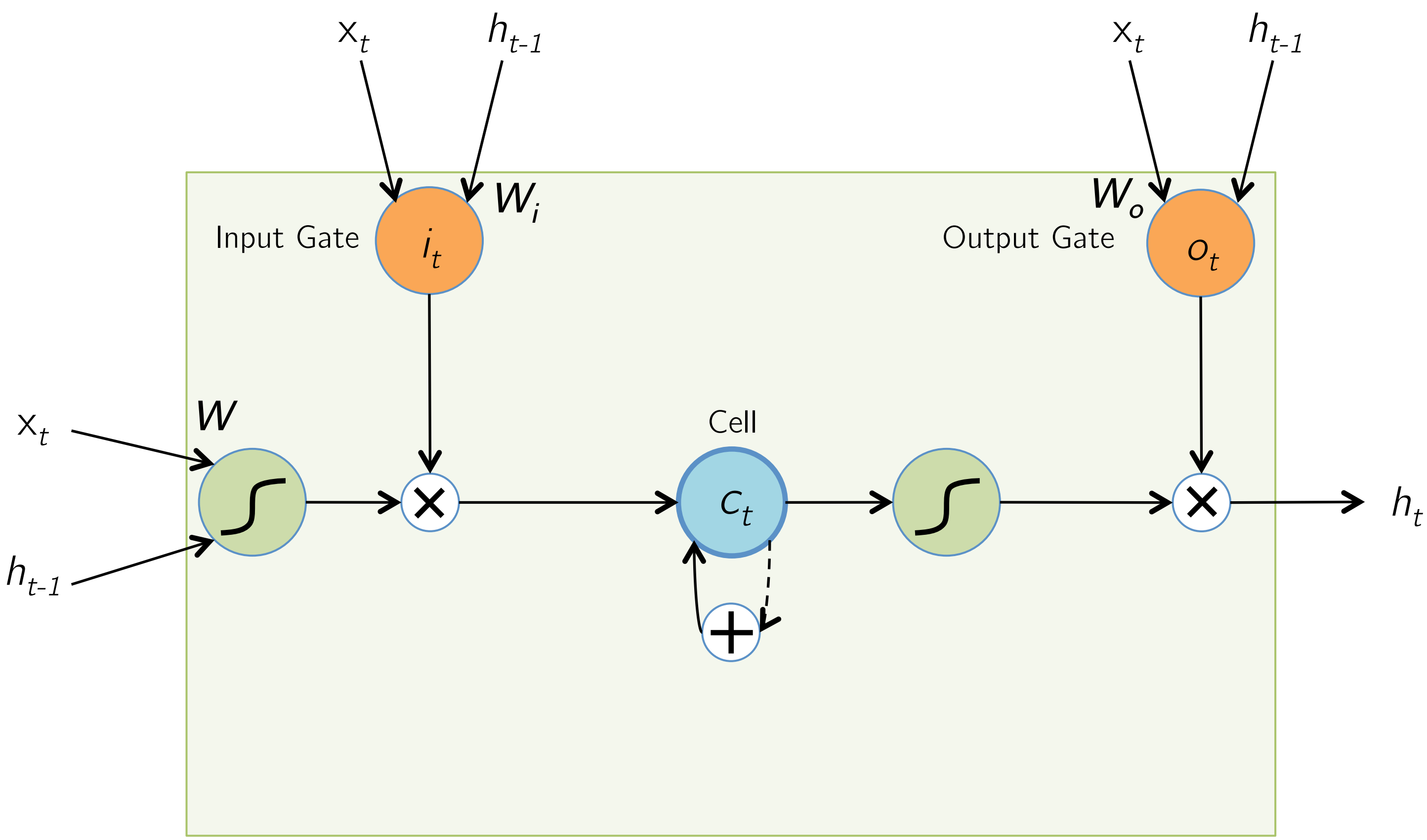
The LSTM Idea



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

* Dashed line indicates time-lag

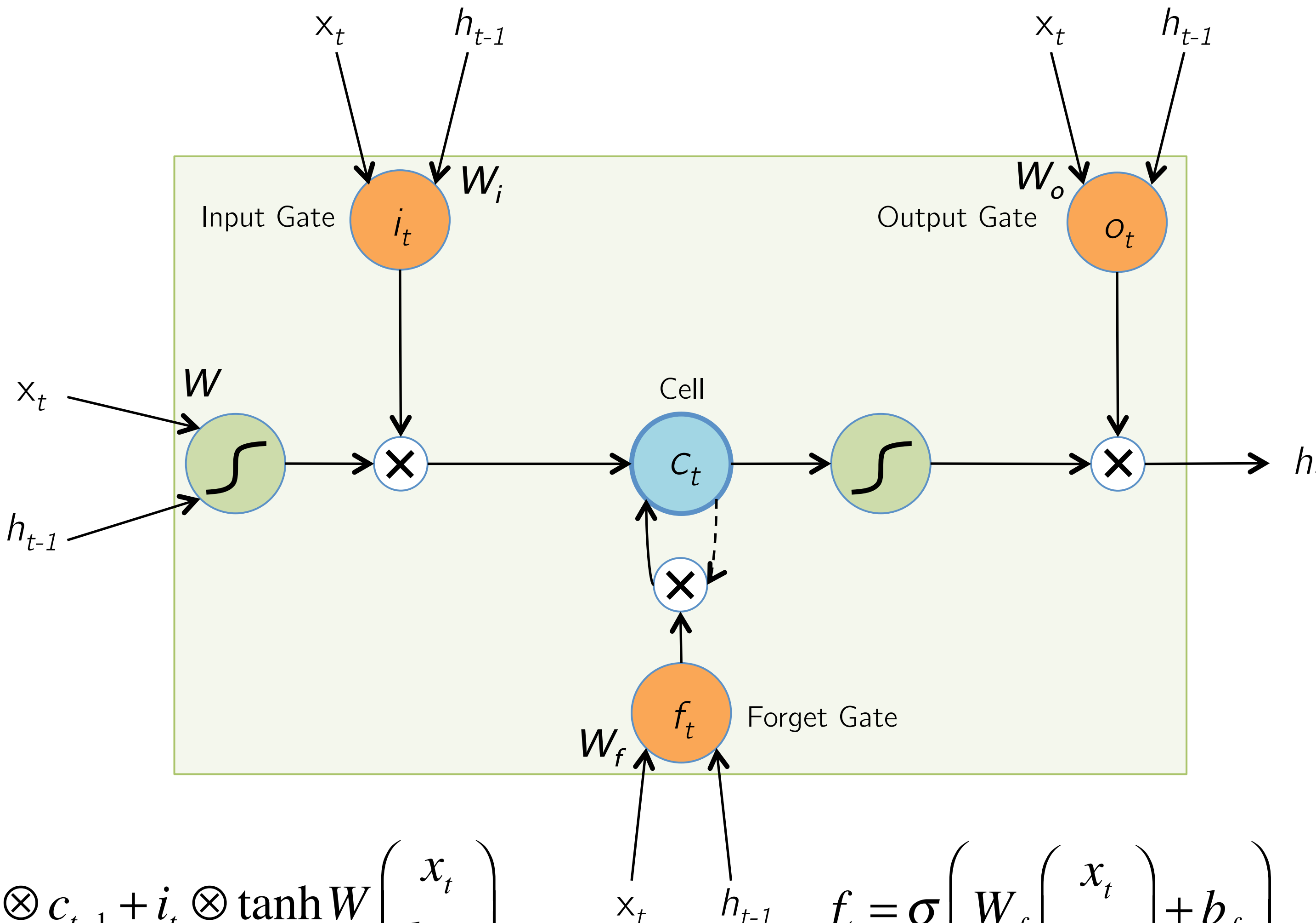
The Original LSTM Cell



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

Similarly for o_t

The Popular LSTM Cell



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

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

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

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