

CS6375: Machine Learning

Gautam Kunapuli

Recurrent Neural Networks

Slides by Arun Mallya, Fei-Fei Li, Justin Johnson, Serena Yeung



THE UNIVERSITY OF TEXAS AT DALLAS

Erik Jonsson School of Engineering and Computer Science

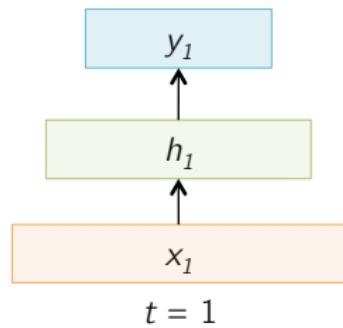
Motivation

- 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

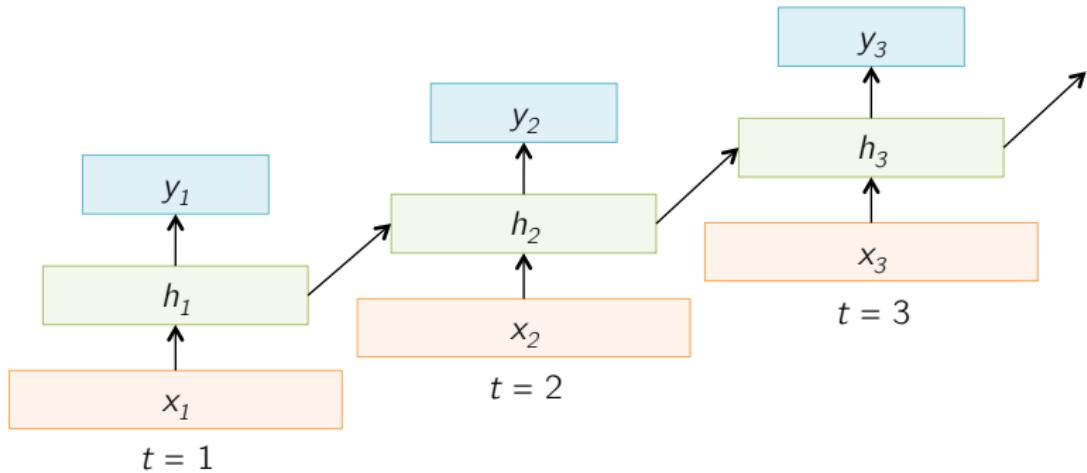
Recurrent Neural Networks (RNNs)

- 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



Input – Output Scenarios

Single - Single



Feed-forward Network

Single - Multiple

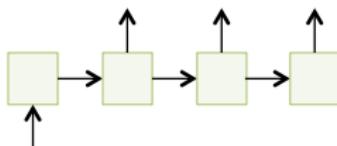
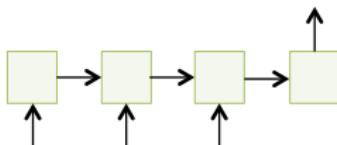


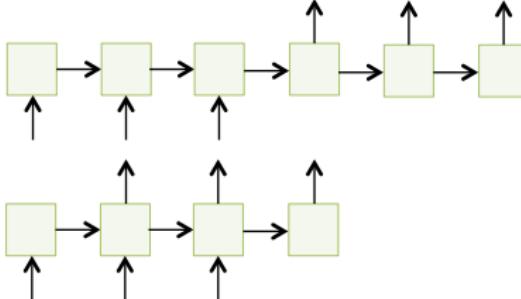
Image Captioning

Multiple - Single



Sentiment Classification

Multiple - Multiple

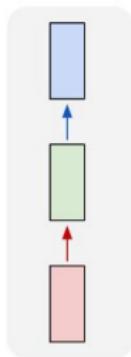


Translation

Image Captioning

“Vanilla” Neural Network

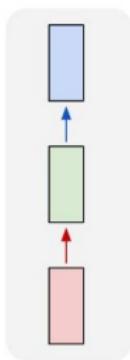
one to one



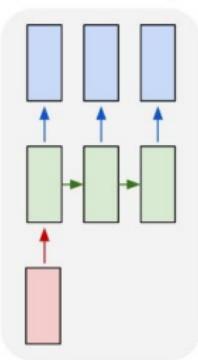
← **Vanilla Neural Networks**

Recurrent Neural Networks: Process Sequences

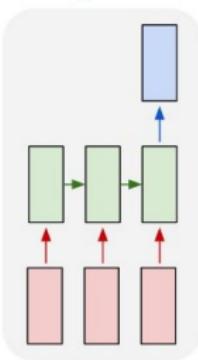
one to one



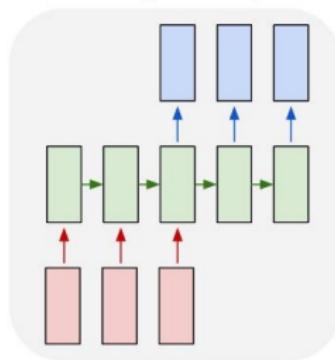
one to many



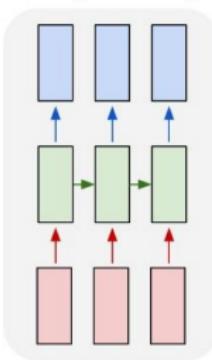
many to one



many to many



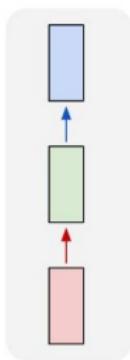
many to many



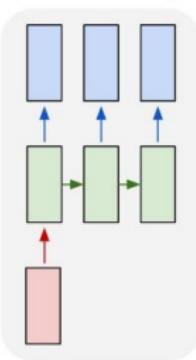
→ e.g. **Image Captioning**
image → sequence of words

Recurrent Neural Networks: Process Sequences

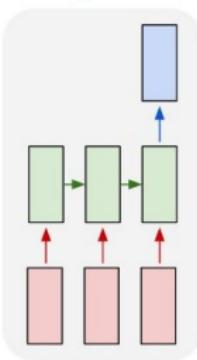
one to one



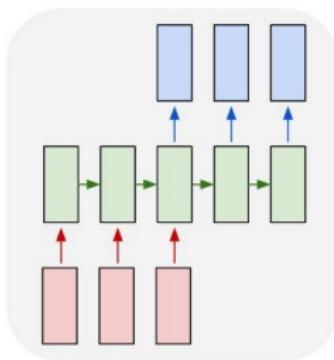
one to many



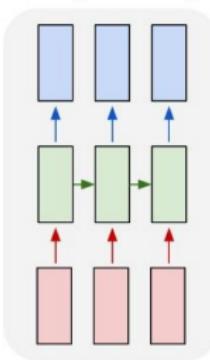
many to one



many to many



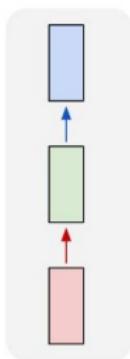
many to many



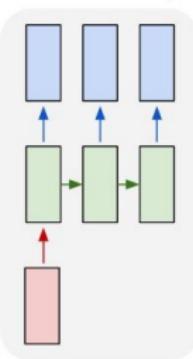
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Neural Networks: Process Sequences

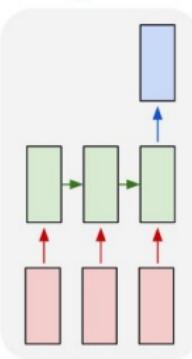
one to one



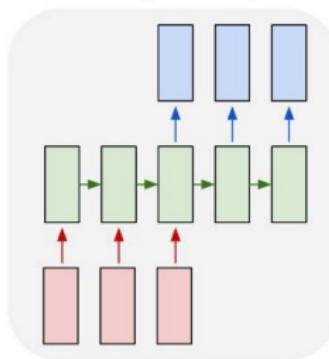
one to many



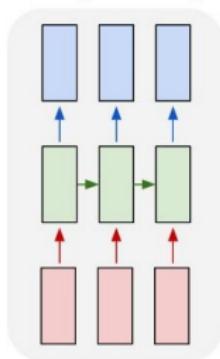
many to one



many to many



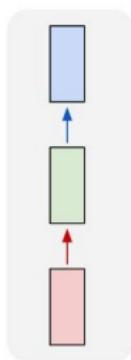
many to many



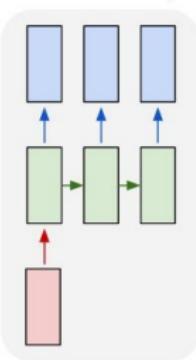
↑
e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Neural Networks: Process Sequences

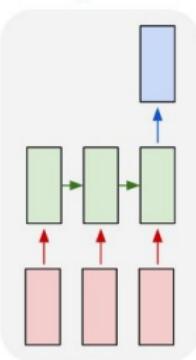
one to one



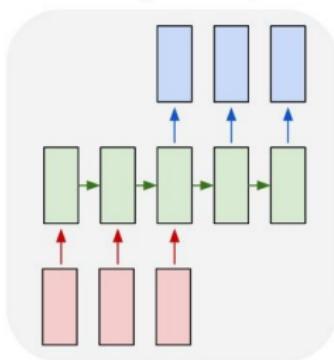
one to many



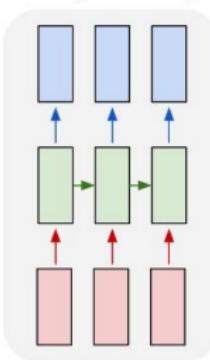
many to one



many to many

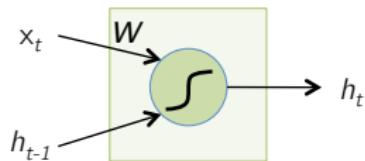


many to many



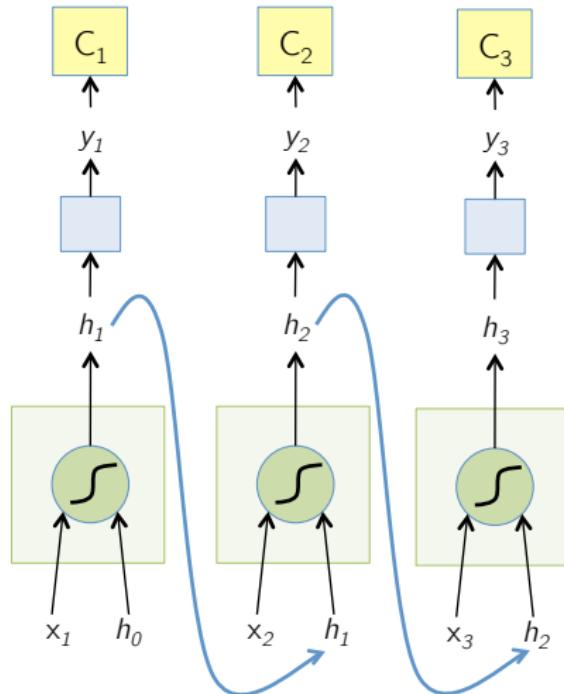
e.g. **Video classification on frame level**

The Vanilla RNN Cell



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

The Vanilla RNN Forward



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

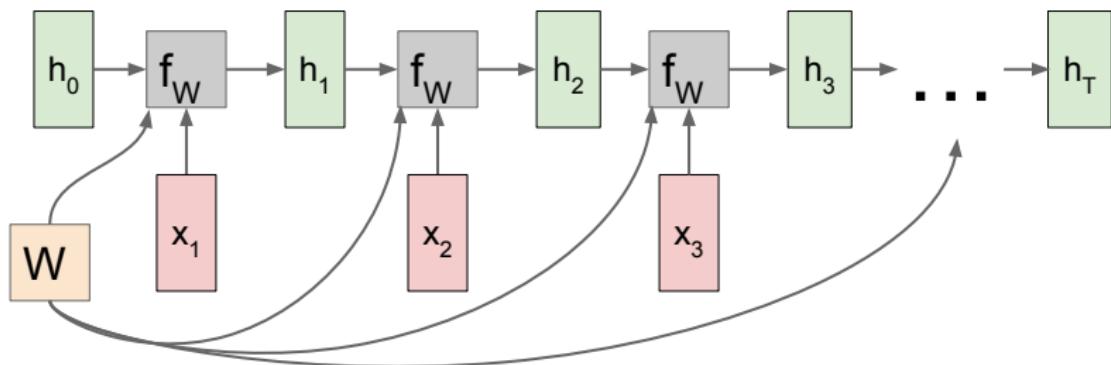
$$y_t = F(h_t)$$

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

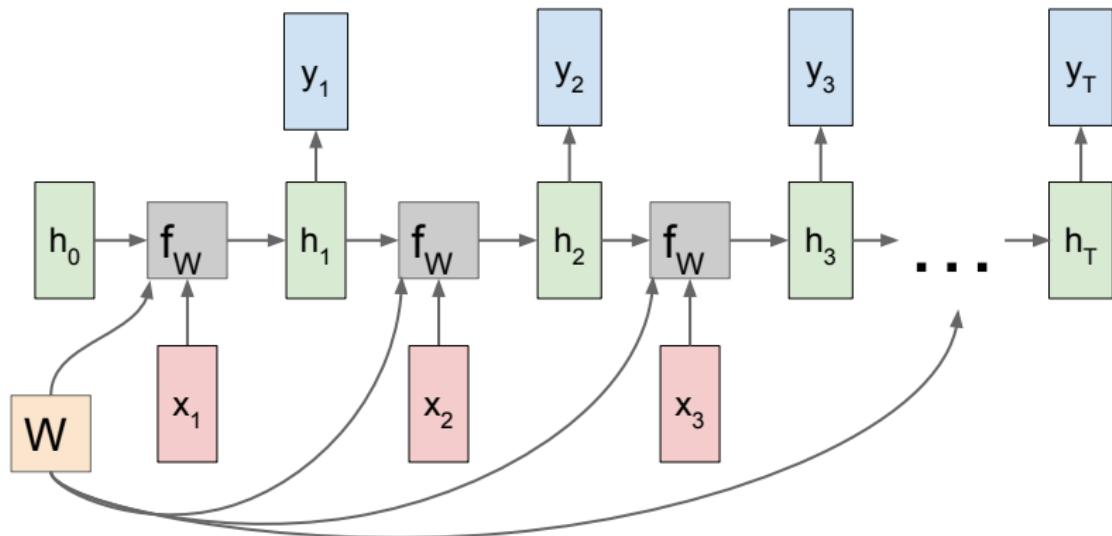
“Unfold” network through time by making copies at each time-step

RNN: Computational Graph

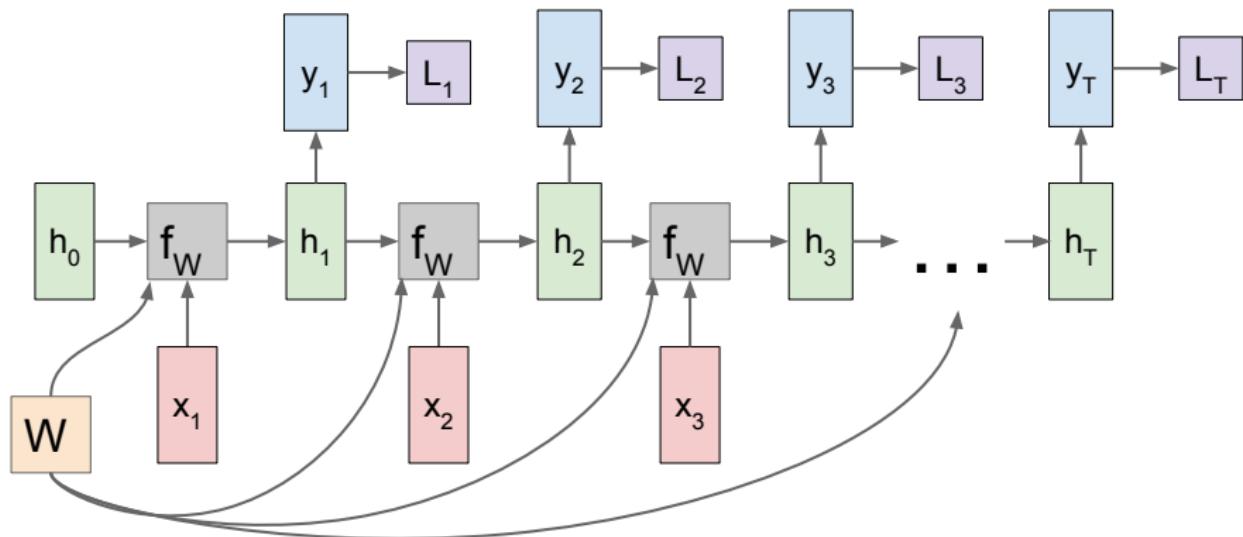
Re-use the same weight matrix at every time-step



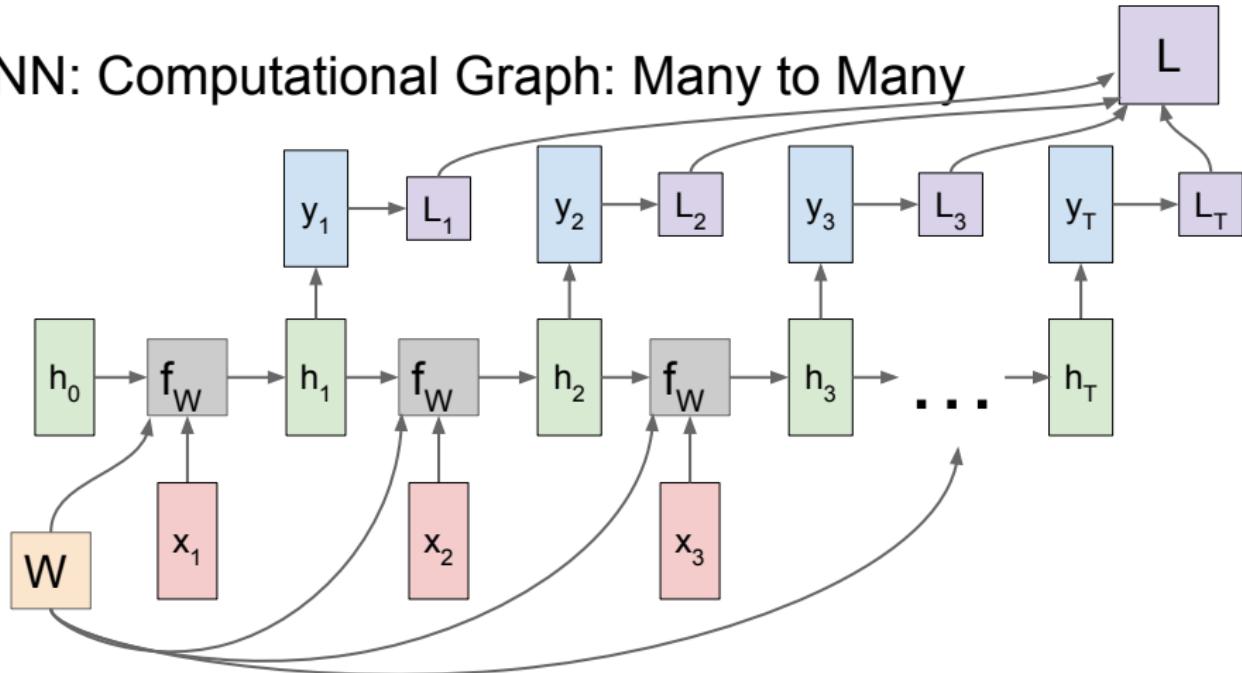
RNN: Computational Graph: Many to Many



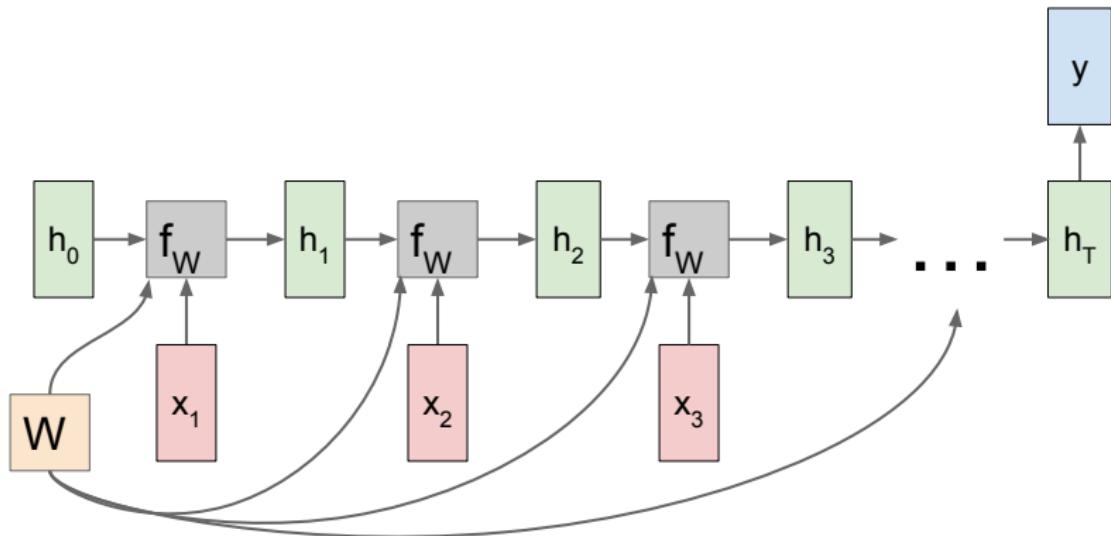
RNN: Computational Graph: Many to Many



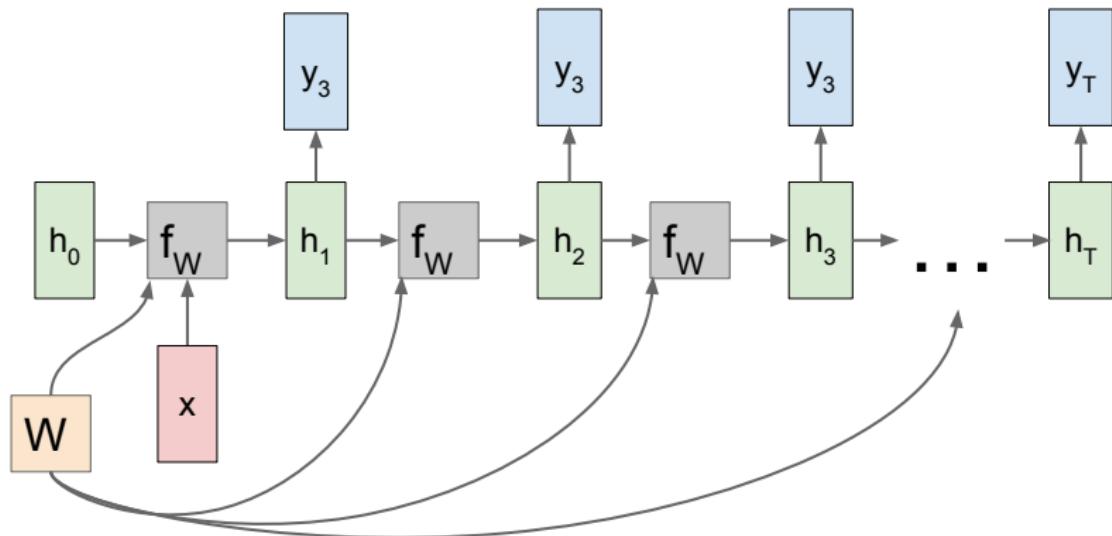
RNN: Computational Graph: Many to Many



RNN: Computational Graph: Many to One

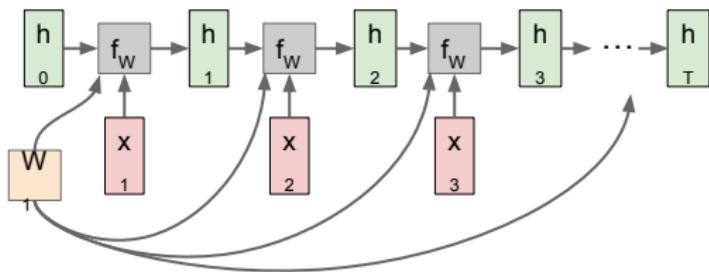


RNN: Computational Graph: One to Many



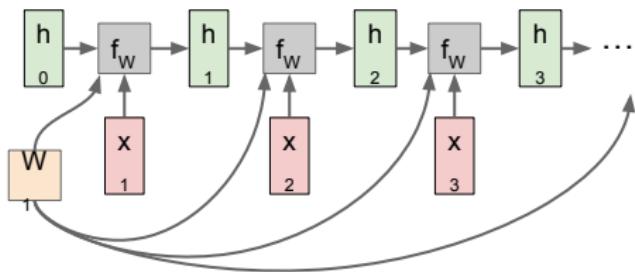
Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

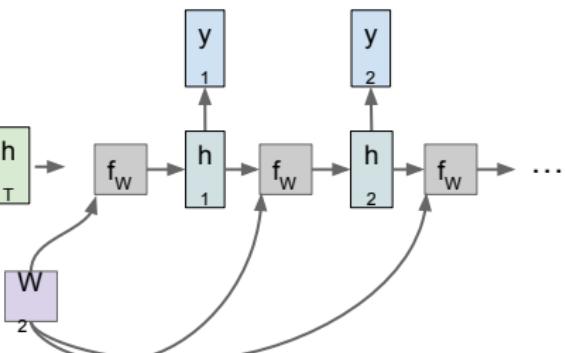


Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector



One to many: Produce output sequence from single input vector

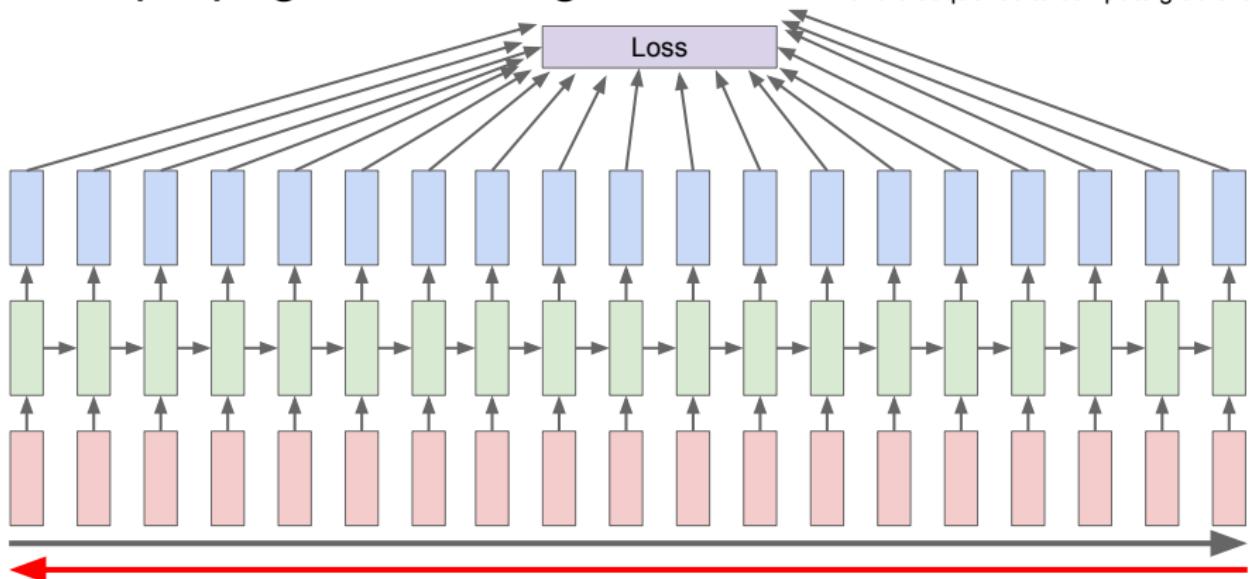


BackPropagation Through Time (BPTT)

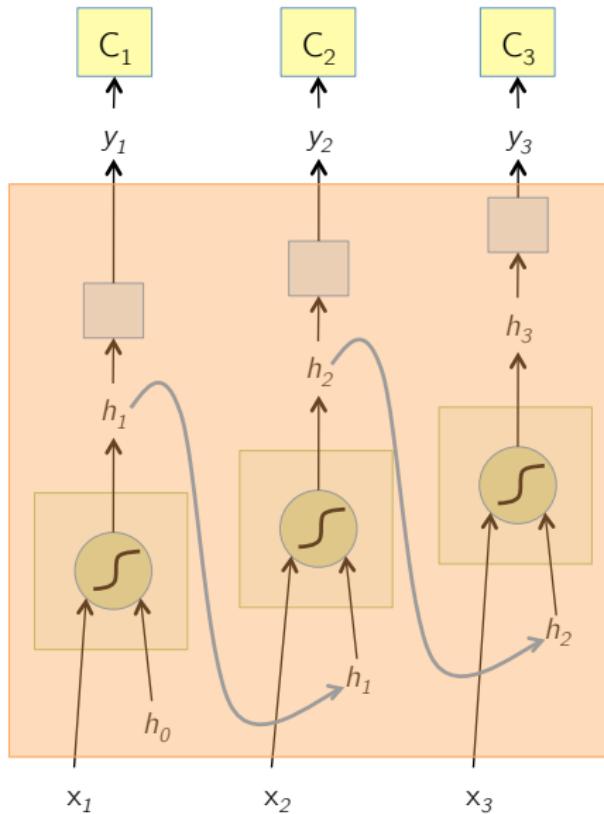
- One of the methods used to train RNNs
- The unfolded network (used during forward pass) is treated as one big feed-forward network
- This unfolded network accepts the whole time series as input
- The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and then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

Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

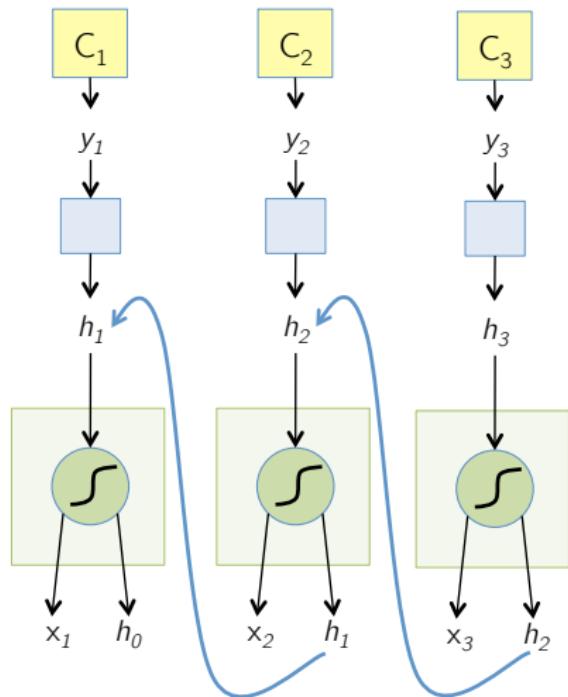


The Unfolded Vanilla RNN



- 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 Vanilla RNN Backward



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

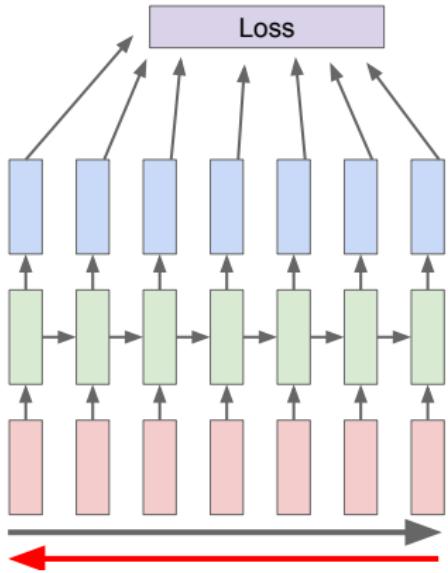
$$y_t = F(h_t)$$

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

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

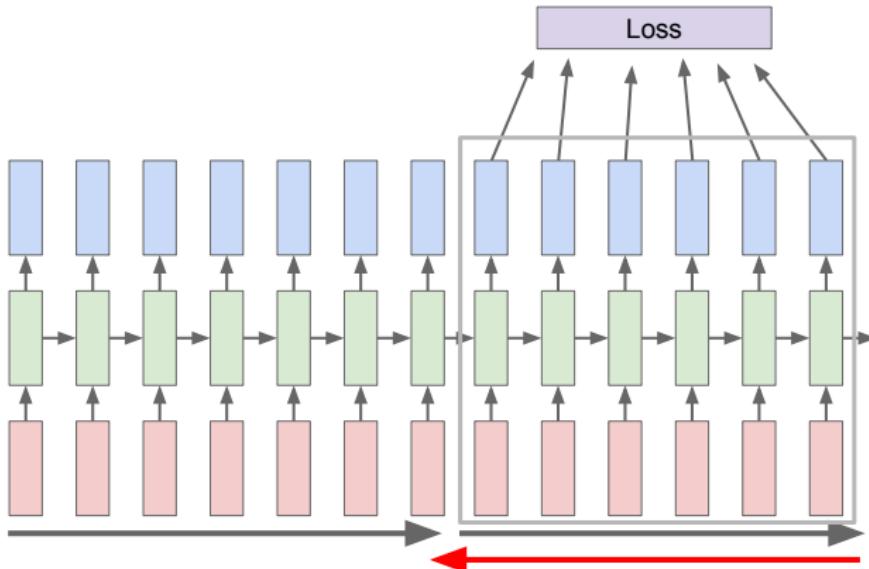
$$= \left(\frac{\partial C_t}{\partial y_t} \right) \left(\frac{\partial y_t}{\partial h_t} \right) \left(\frac{\partial h_t}{\partial h_{t-1}} \right) \dots \left(\frac{\partial h_2}{\partial h_1} \right)$$

Truncated Backpropagation through time



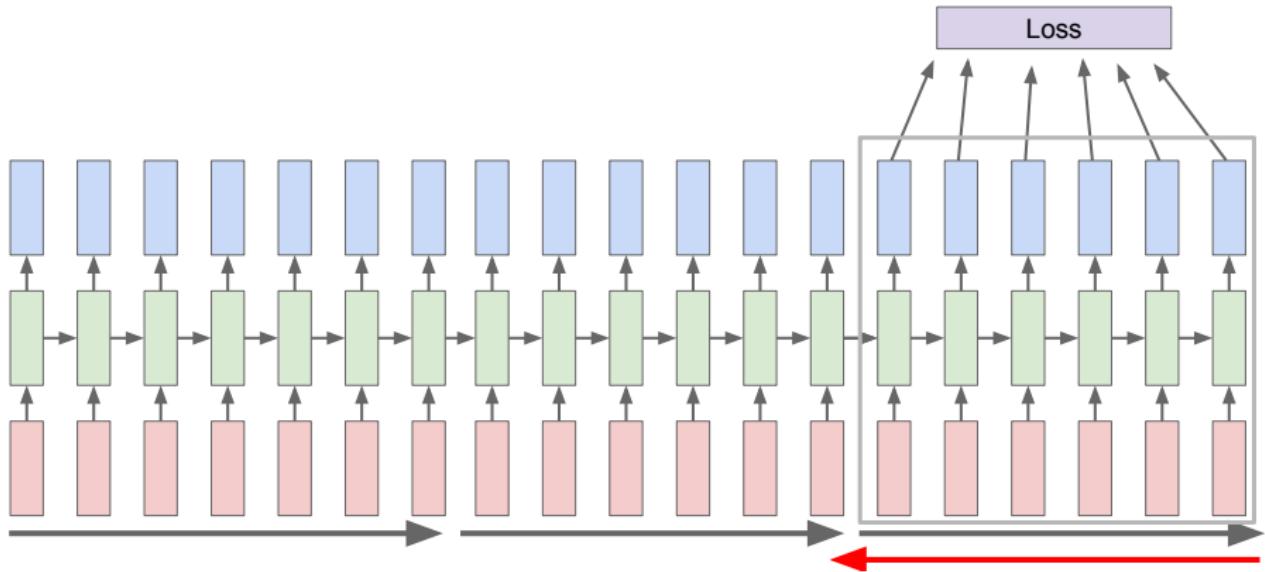
Run forward and backward
through chunks of the
sequence instead of whole
sequence

Truncated Backpropagation through time



Carry hidden states forward in time forever,
but only backpropagate
for some smaller number of steps

Truncated Backpropagation through time



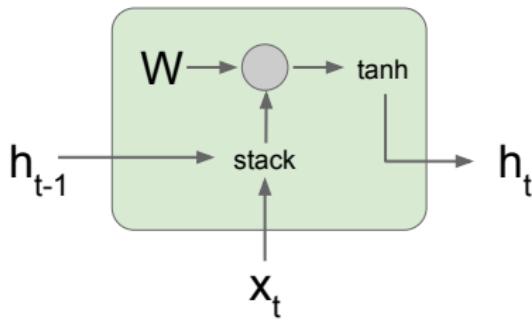
Issues with the Vanilla RNNs

- 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](#)

Vanilla RNN Gradient Flow

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

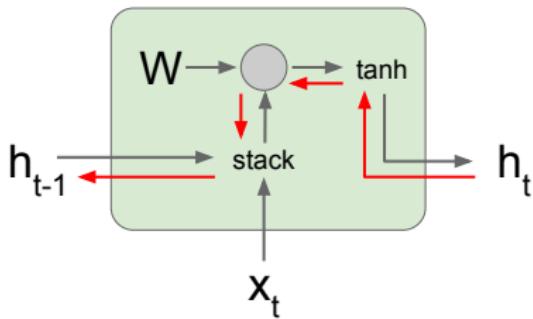


$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh \left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\ &= \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \end{aligned}$$

Vanilla RNN Gradient Flow

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

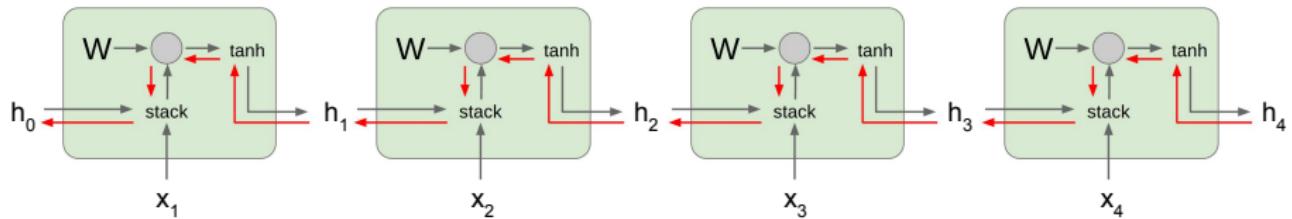
Backpropagation from h_t
to h_{t-1} multiplies by W
(actually W_{hh}^T)



$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh \left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\ &= \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \end{aligned}$$

Vanilla RNN Gradient Flow

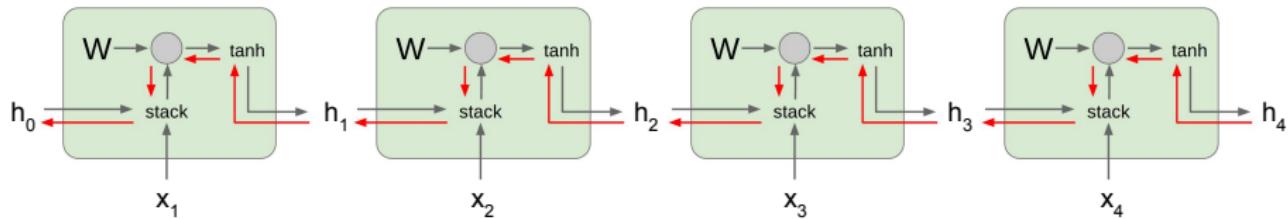
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W
(and repeated tanh)

Vanilla RNN Gradient Flow

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



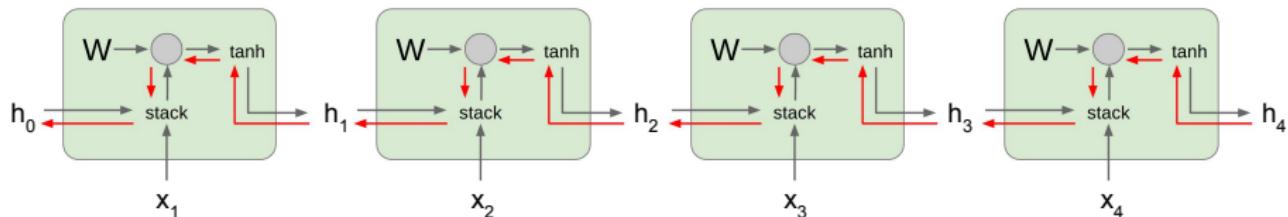
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1 :
Exploding gradients

Largest singular value < 1 :
Vanishing gradients

Vanilla RNN Gradient Flow

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 gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1 :
Exploding gradients

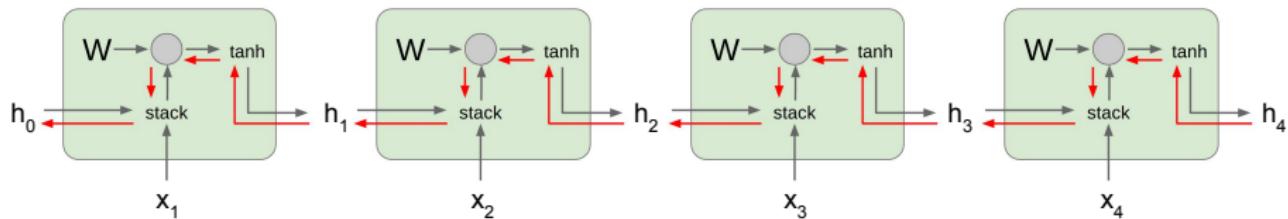
Largest singular value < 1 :
Vanishing gradients

Gradient clipping: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)
```

Vanilla RNN Gradient Flow

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 gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1 :
Exploding gradients

Largest singular value < 1 :
Vanishing gradients

→ Change RNN architecture

The Identity Relationship

- Recall $\frac{\partial C_t}{\partial h_1} = \left(\frac{\partial C_t}{\partial y_t} \right) \left(\frac{\partial y_t}{\partial h_1} \right)$ $h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$
 $= \left(\frac{\partial C_t}{\partial y_t} \right) \left(\frac{\partial y_t}{\partial h_t} \right) \left(\frac{\partial h_t}{\partial h_{t-1}} \right) \dots \left(\frac{\partial h_2}{\partial h_1} \right)$ $y_t = F(h_t)$
 $C_t = \text{Loss}(y_t, \text{GT}_t)$
- 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”

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](#)

Long Short Term Memory (LSTM)

Vanilla RNN

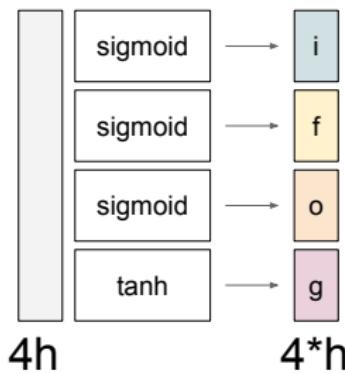
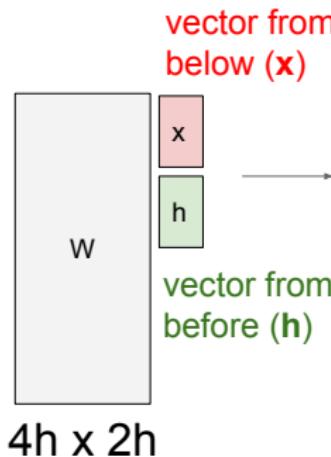
$$h_t = \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



- f: Forget gate, Whether to erase cell
- i: Input gate, whether to write to cell
- g: Gate gate (?), How much to write to cell
- o: Output gate, How much to reveal cell

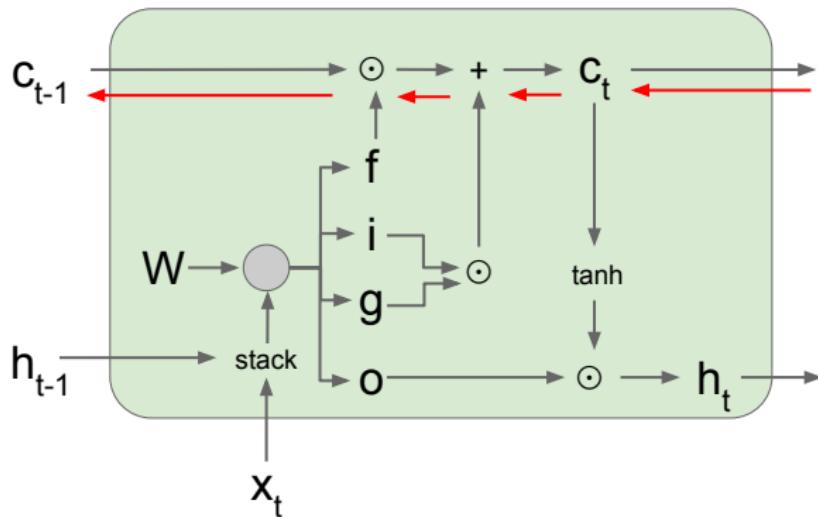
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



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

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

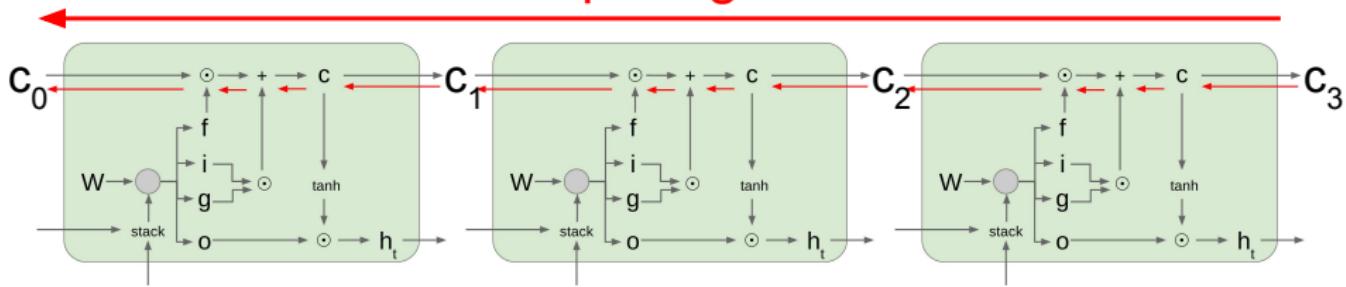
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!



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 Merriënboer, 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