EECS 545: Machine Learning

Lecture 13. RNNs and LSTMs

Honglak Lee

02/24/2025

UNIVERSITY OF MICHIGAN

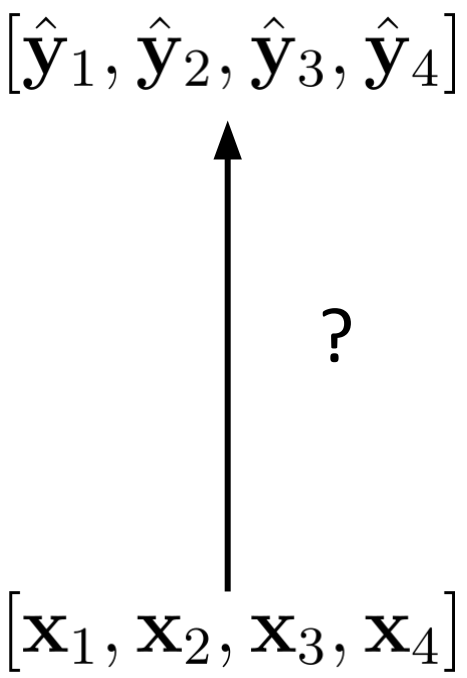
Outline

- **RNN Basics**
- Backpropagation through time (BPTT)
- The vanishing/exploding gradients problem
- Applications

RNN Basics

Sequence modeling

- Can we model sequences with standard NNs?

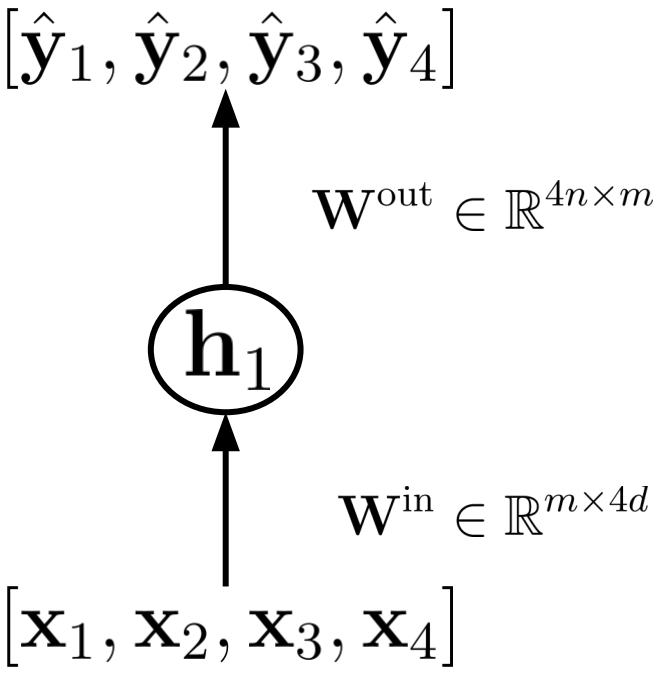


where $\mathbf{x}_t \in \mathbb{R}^d, \mathbf{h}_t \in \mathbb{R}^m, \mathbf{y}_t \in \mathbb{R}^n$

RNN Basics

Sequence modeling

- Can we model sequences with standard NNs?
- Sure, but only if the length is fixed

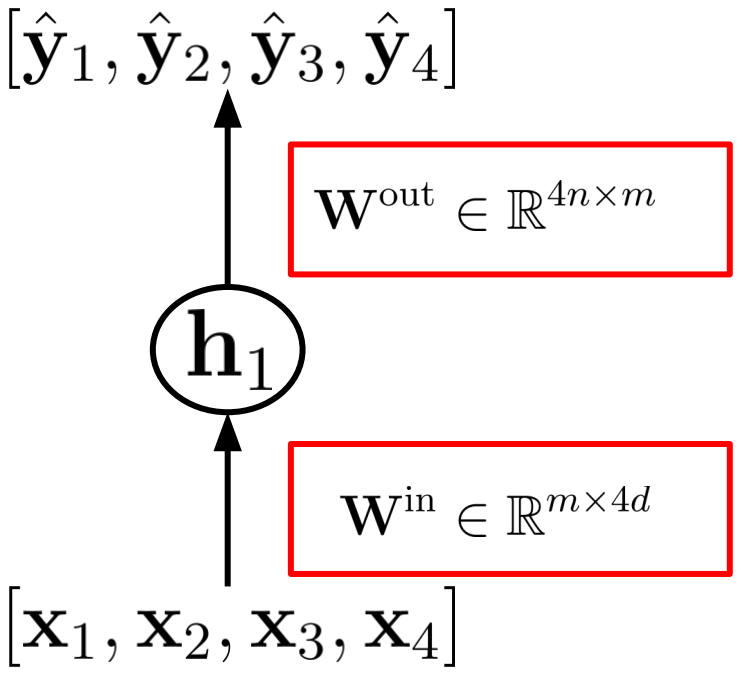


where $\mathbf{x}_t \in \mathbb{R}^d, \mathbf{h}_t \in \mathbb{R}^m, \mathbf{y}_t \in \mathbb{R}^n$

RNN Basics

Sequence modeling

- Can we model sequences with standard NNs?
- Sure, but only if the length is fixed
- **Input/output weight matrices grow with number of inputs/outputs!**



where $\mathbf{x}_t \in \mathbb{R}^d, \mathbf{h}_t \in \mathbb{R}^m, \mathbf{y}_t \in \mathbb{R}^n$

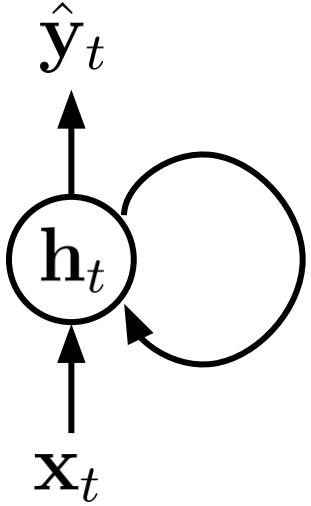
RNN Basics

How can we model sequences in an efficient way?

RNN Basics

How can we model sequences in an efficient way?

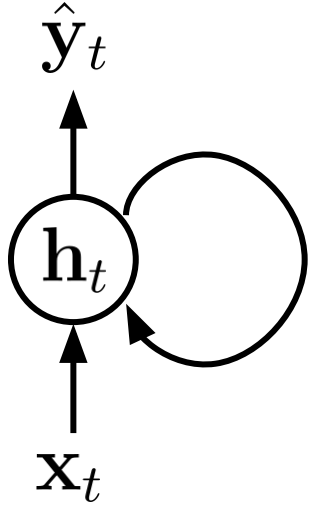
- Use a model that reads each input one at a time



RNN Basics

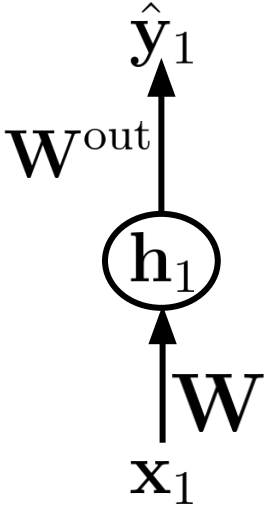
How can we model sequences in an efficient way?

- Use a model that reads each input one at a time
- Parameters can simply be reused (or shared) for each input in a recurrent computation



RNN Basics

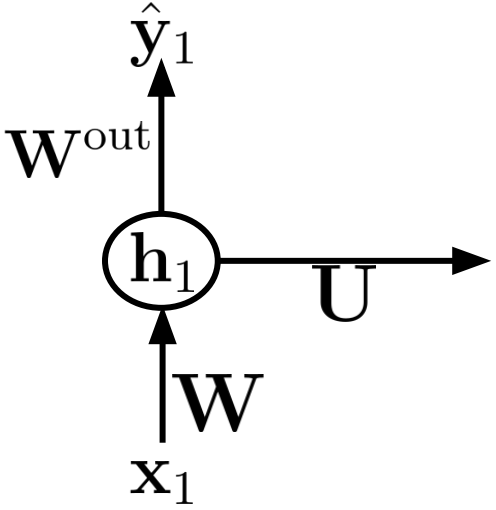
Unrolling RNNs through time



$$\mathbf{h}_1 = \sigma(\mathbf{W}\mathbf{x}_1 + \mathbf{b})$$
$$\hat{\mathbf{y}}_1 = \mathbf{W}^{\text{out}}\mathbf{h}_1$$

RNN Basics

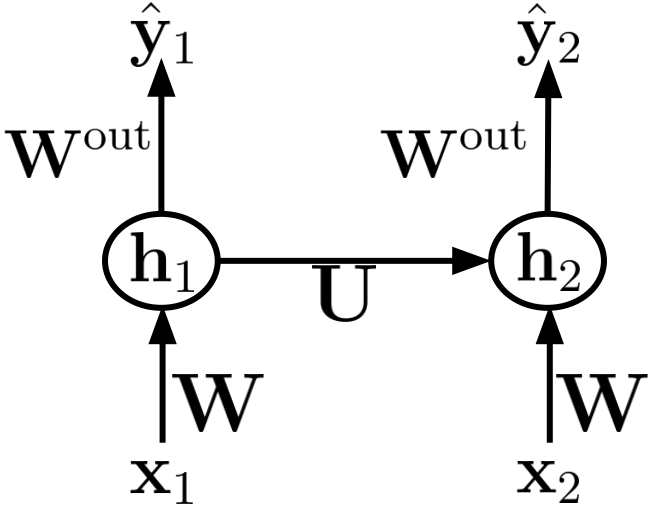
Unrolling RNNs through time



$$\mathbf{h}_1 = \sigma(\mathbf{W}\mathbf{x}_1 + \mathbf{b})$$
$$\hat{\mathbf{y}}_1 = \mathbf{W}^{\text{out}}\mathbf{h}_1$$

RNN Basics

Unrolling RNNs through time

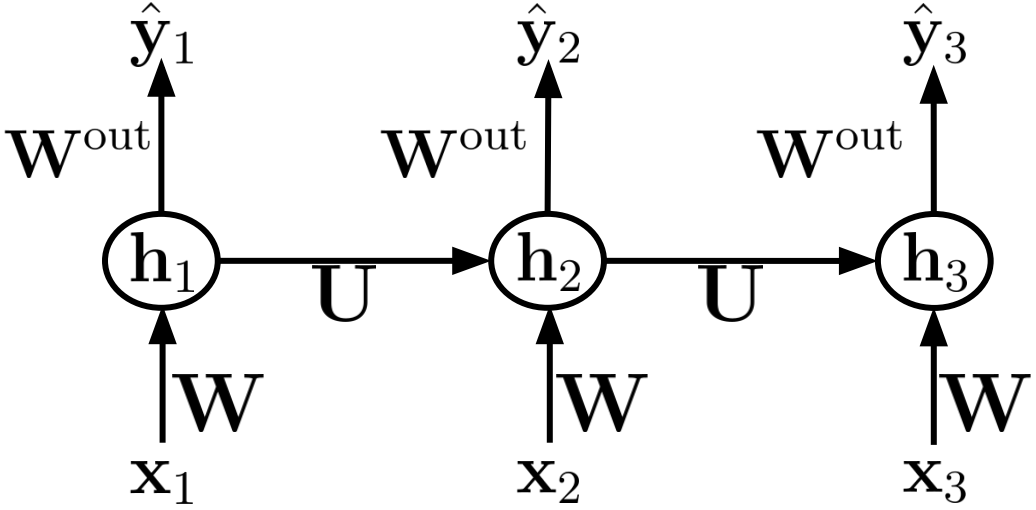


$$\mathbf{h}_1 = \sigma(\mathbf{W}\mathbf{x}_1 + \mathbf{b})$$
$$\hat{\mathbf{y}}_1 = \mathbf{W}^{\text{out}}\mathbf{h}_1$$

$$\mathbf{h}_2 = \sigma(\mathbf{U}\mathbf{h}_1 + \mathbf{W}\mathbf{x}_2 + \mathbf{b})$$
$$\hat{\mathbf{y}}_2 = \mathbf{W}^{\text{out}}\mathbf{h}_2$$

RNN Basics

Unrolling RNNs through time



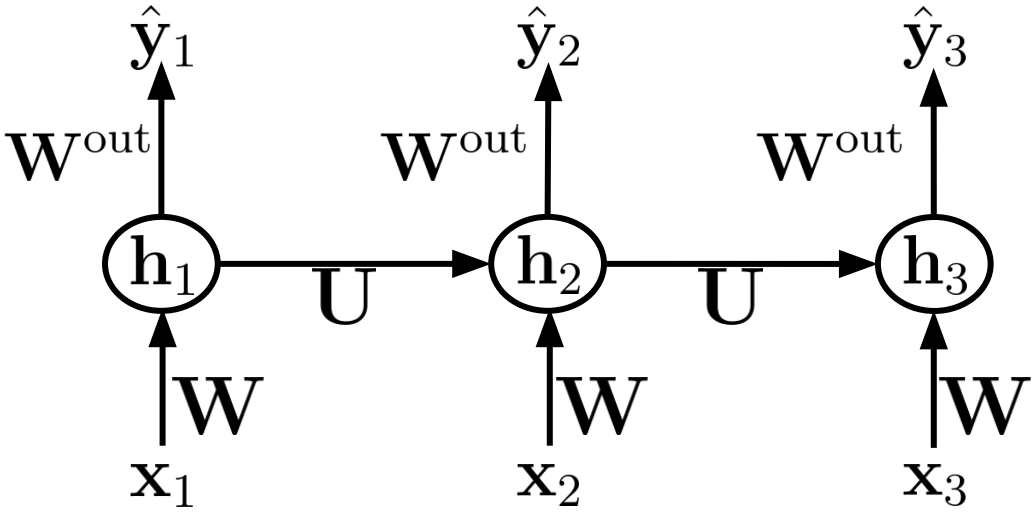
$$\mathbf{h}_1 = \sigma(\mathbf{W}\mathbf{x}_1 + \mathbf{b})$$
$$\hat{\mathbf{y}}_1 = \mathbf{W}^{\text{out}}\mathbf{h}_1$$

$$\mathbf{h}_2 = \sigma(\mathbf{U}\mathbf{h}_1 + \mathbf{W}\mathbf{x}_2 + \mathbf{b})$$
$$\hat{\mathbf{y}}_2 = \mathbf{W}^{\text{out}}\mathbf{h}_2$$

$$\mathbf{h}_3 = \sigma(\mathbf{U}\mathbf{h}_2 + \mathbf{W}\mathbf{x}_3 + \mathbf{b})$$
$$\hat{\mathbf{y}}_3 = \mathbf{W}^{\text{out}}\mathbf{h}_3$$

RNN Basics

Unrolling RNNs through time



In general, for any $t \geq 1$,

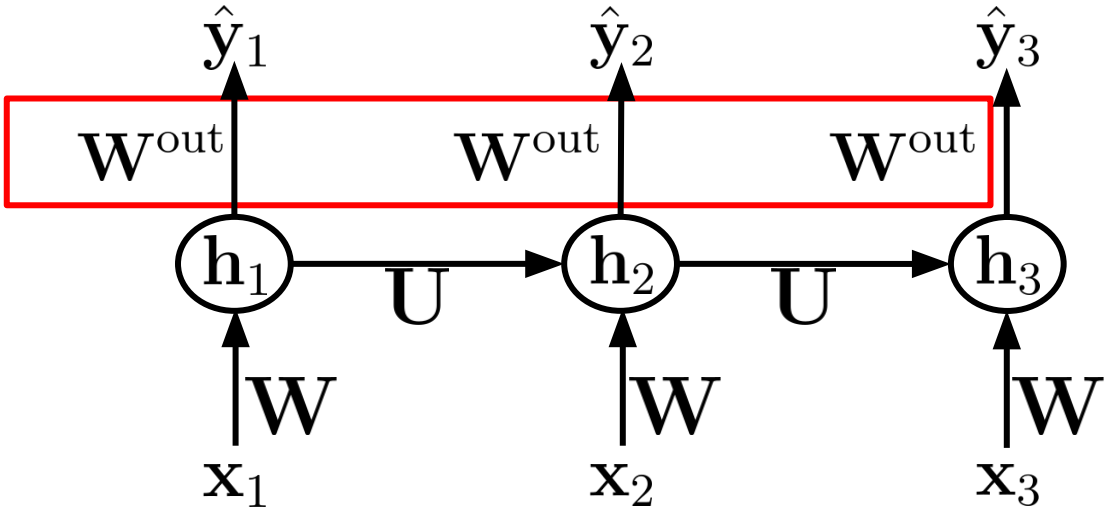
$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}^{\text{out}}\mathbf{h}_t$$

$*$ Note: $h_0 = 0$

RNN Basics

Unrolling RNNs through time

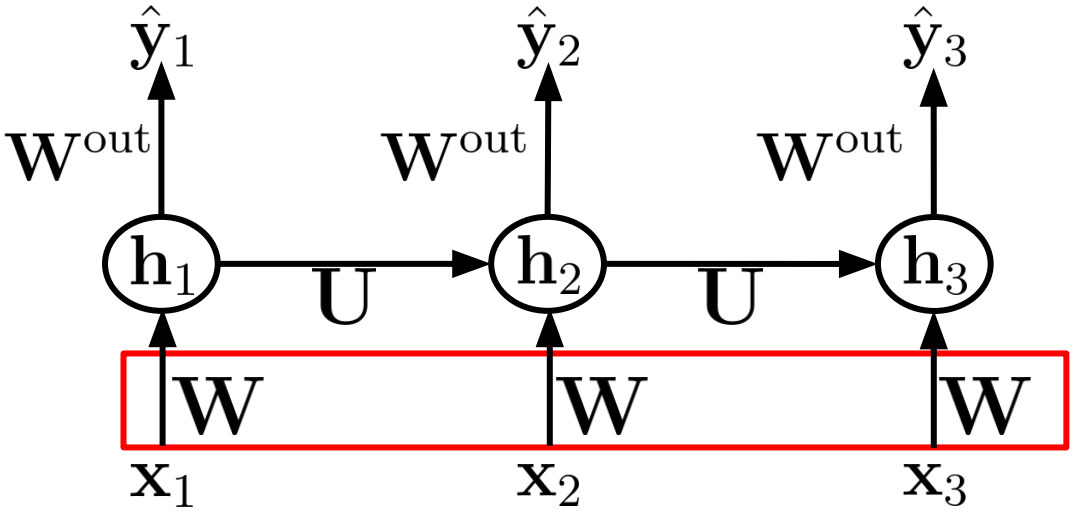


$$\hat{\mathbf{y}}_t = \mathbf{W}^{\text{out}}\mathbf{h}_t$$

Weights are shared!

RNN Basics

Unrolling RNNs through time

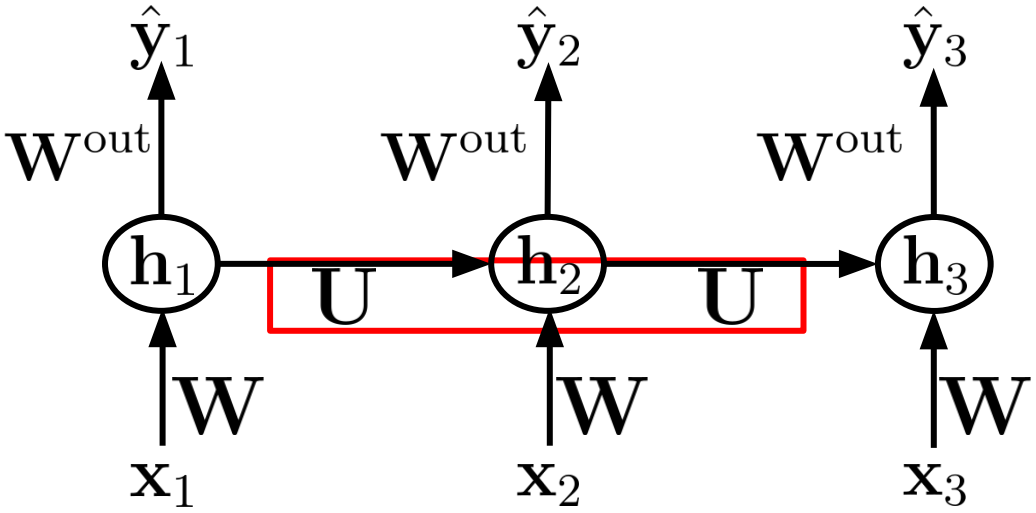


$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$

Weights are shared!

RNN Basics

Unrolling RNNs through time

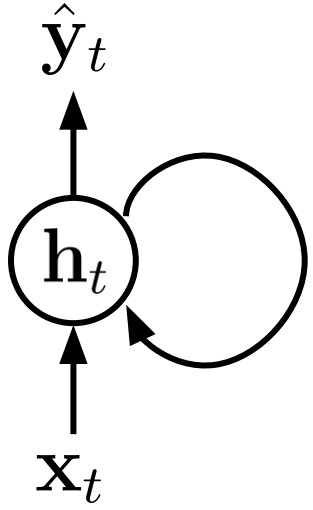


$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$

Weights are shared!

RNN Basics

Single step computation in RNNs



$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$

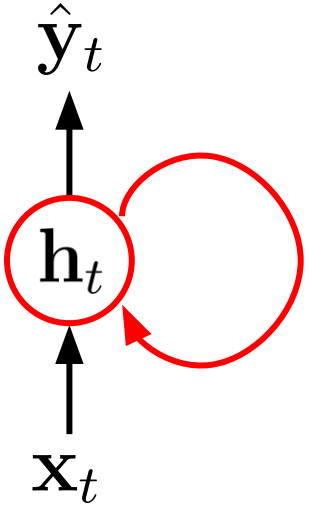
$$\hat{\mathbf{y}}_t = \mathbf{W}^{\text{out}}\mathbf{h}_t$$

where $\mathbf{x}_t \in \mathbb{R}^d, \mathbf{h}_t \in \mathbb{R}^m, \mathbf{y}_t \in \mathbb{R}^n,$

$$\mathbf{U} \in \mathbb{R}^{m \times m}, \mathbf{W} \in \mathbb{R}^{m \times d}, \mathbf{W}^{\text{out}} \in \mathbb{R}^{n \times m}, \mathbf{b} \in \mathbb{R}^m$$

RNN Basics

Single step computation in RNNs



Connections from previous hidden state into the next hidden state

$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$

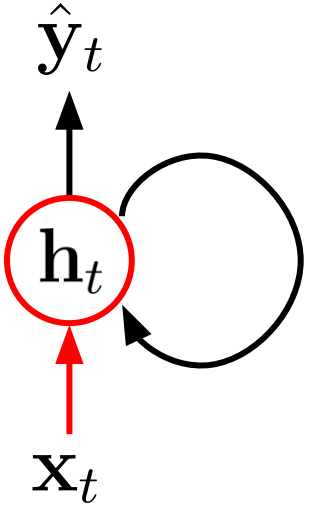
$$\hat{\mathbf{y}}_t = \mathbf{W}^{\text{out}}\mathbf{h}_t$$

where $\mathbf{x}_t \in \mathbb{R}^d, \mathbf{h}_t \in \mathbb{R}^m, \mathbf{y}_t \in \mathbb{R}^n,$

$\mathbf{U} \in \mathbb{R}^{m \times m}, \mathbf{W} \in \mathbb{R}^{m \times d}, \mathbf{W}^{\text{out}} \in \mathbb{R}^{n \times m}, \mathbf{b} \in \mathbb{R}^m$

RNN Basics

Single step computation in RNNs



Connection from input into hidden state

$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$

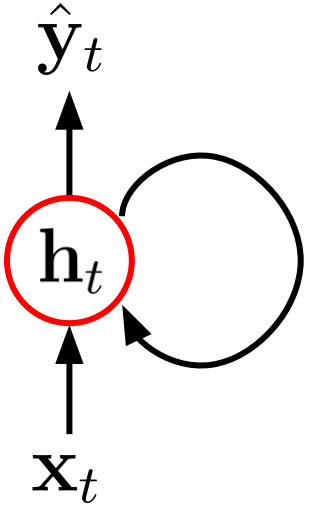
$$\hat{\mathbf{y}}_t = \mathbf{W}^{\text{out}}\mathbf{h}_t$$

where $\mathbf{x}_t \in \mathbb{R}^d, \mathbf{h}_t \in \mathbb{R}^m, \mathbf{y}_t \in \mathbb{R}^n,$

$\mathbf{U} \in \mathbb{R}^{m \times m}, \mathbf{W} \in \mathbb{R}^{m \times d}, \mathbf{W}^{\text{out}} \in \mathbb{R}^{n \times m}, \mathbf{b} \in \mathbb{R}^m$

RNN Basics

Single step computation in RNNs



Bias term

$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$

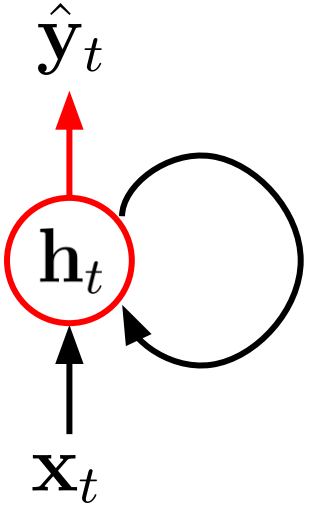
$$\hat{\mathbf{y}}_t = \mathbf{W}^{\text{out}}\mathbf{h}_t$$

where $\mathbf{x}_t \in \mathbb{R}^d, \mathbf{h}_t \in \mathbb{R}^m, \mathbf{y}_t \in \mathbb{R}^n,$

$$\mathbf{U} \in \mathbb{R}^{m \times m}, \mathbf{W} \in \mathbb{R}^{m \times d}, \mathbf{W}^{\text{out}} \in \mathbb{R}^{n \times m}, \mathbf{b} \in \mathbb{R}^m$$

RNN Basics

Single step computation in RNNs



Connection from hidden unit to output

$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}^{\text{out}}\mathbf{h}_t$$

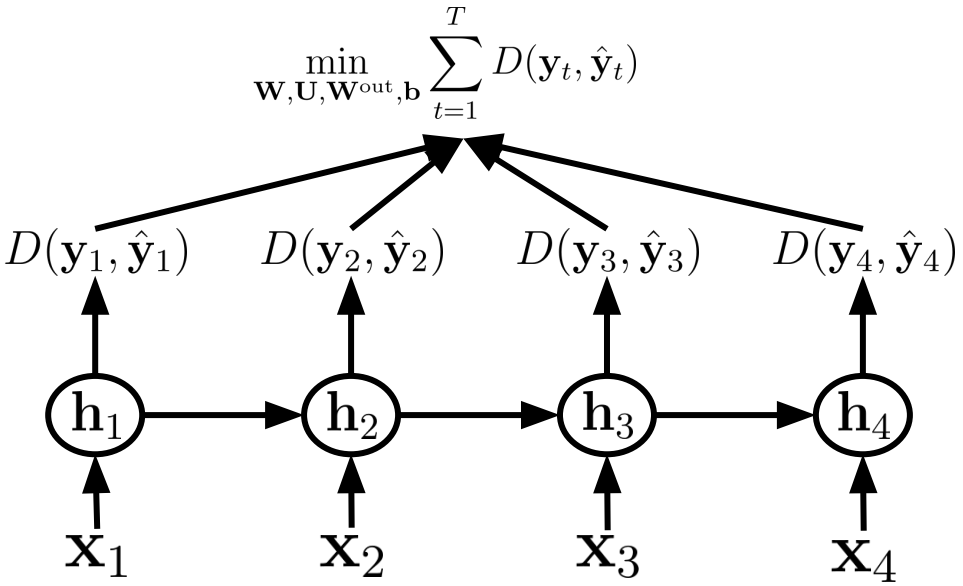
where $\mathbf{x}_t \in \mathbb{R}^d, \mathbf{h}_t \in \mathbb{R}^m, \mathbf{y}_t \in \mathbb{R}^n,$

$\mathbf{U} \in \mathbb{R}^{m \times m}, \mathbf{W} \in \mathbb{R}^{m \times d}, \mathbf{W}^{\text{out}} \in \mathbb{R}^{n \times m}, \mathbf{b} \in \mathbb{R}^m$

RNN Basics: Objective function

RNN general objective function

- Compute loss (if any) for each step in the sequence prediction and aggregate
- Gradient flows through all unrolled steps in the RNN



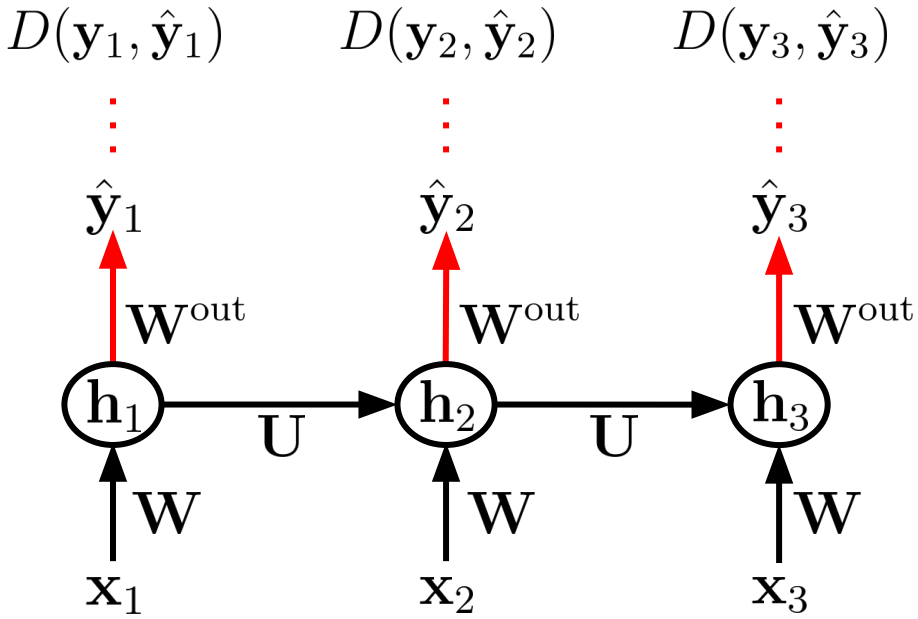
Outline

- RNN Basics
- **Backpropagation through time (BPTT)**
- The vanishing/exploding gradients problem
- Applications

Backpropagation through time (BPTT)

- $\mathcal{L}_t$ aggregates the loss for from time t to T:

$$\mathcal{L}_t = \sum_{\tau=t}^{T} D(\mathbf{y}_\tau, \hat{\mathbf{y}}_\tau)$$



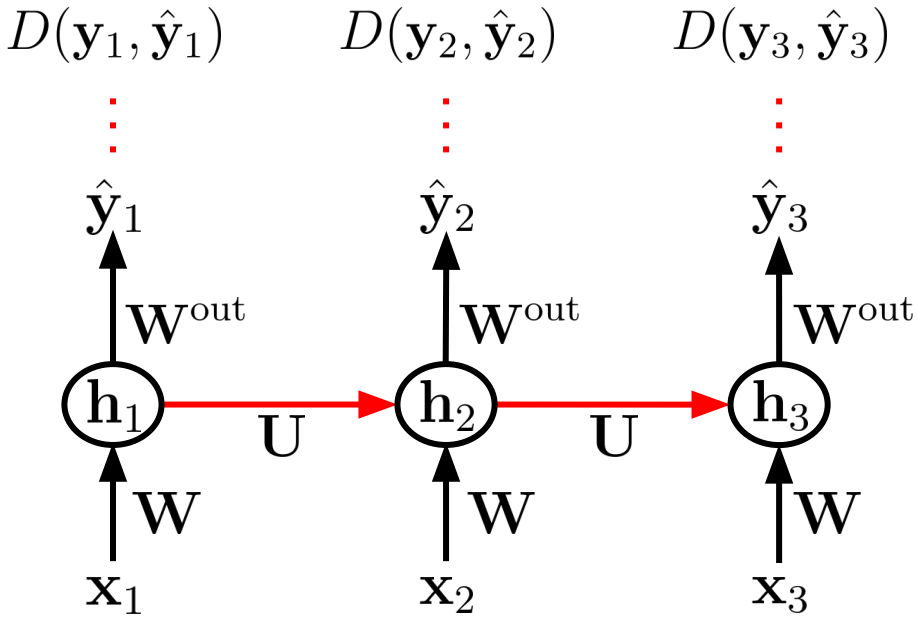
$$\frac{\partial \mathcal{L}_1}{\partial \mathbf{W}^{\text{out}}_{n,m}} = \sum_t \frac{\partial \hat{y}_{t,n}}{\partial \mathbf{W}^{\text{out}}_{n,m}} \frac{\partial D(y_{t,n}, \hat{y}_{t,n})}{\partial \hat{y}_{t,n}}$$

For the output parameters, we only consider the gradient of the current loss and add them up.

Backpropagation through time (BPTT)

- $\mathcal{L}_t$ aggregates the loss for from time t to T:

$$\mathcal{L}_t = \sum_{\tau=t}^{T} D(\mathbf{y}_\tau, \hat{\mathbf{y}}_\tau)$$



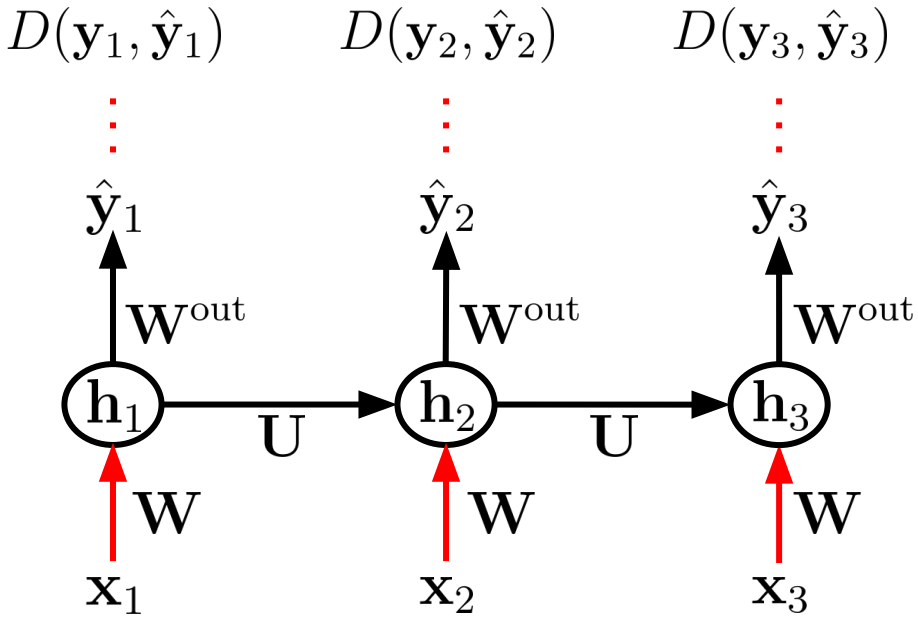
$$\frac{\partial \mathcal{L}_1}{\partial \mathbf{U}_{m,m'}} = \sum_t \frac{\partial h_{t,m}}{\partial \mathbf{U}_{m,m'}} \frac{\partial \mathcal{L}_t}{\partial h_{t,m}}$$

For the other parameters, we use a chain rule involving the hidden activations at each time step and add them up.

Backpropagation through time (BPTT)

- $\mathcal{L}_t$ aggregates the loss for from time t to T:

$$\mathcal{L}_t = \sum_{\tau=t}^{T} D(\mathbf{y}_\tau, \hat{\mathbf{y}}_\tau)$$



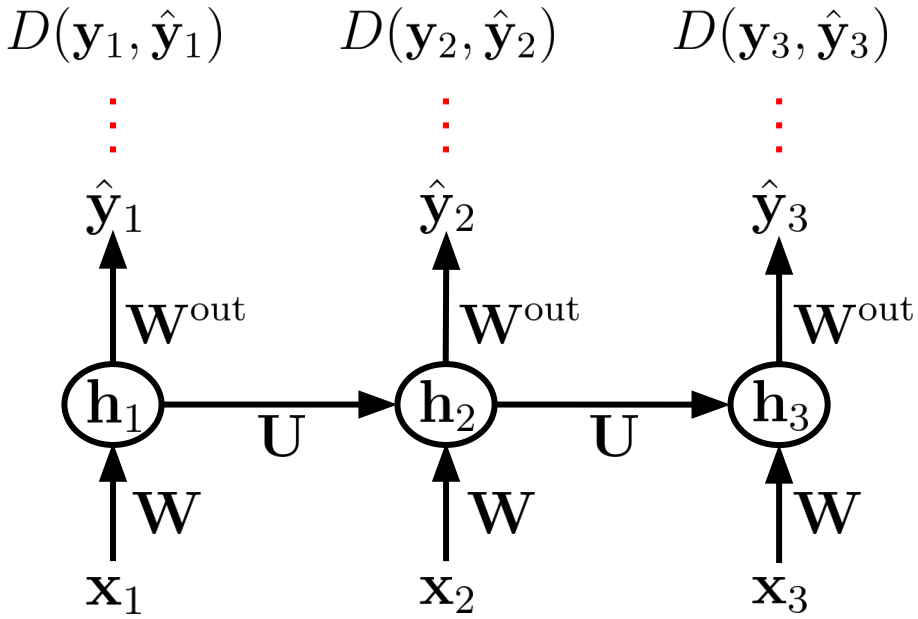
$$\frac{\partial \mathcal{L}_1}{\partial \mathbf{W}_{m,d}} = \sum_t \frac{\partial h_{t,m}}{\partial \mathbf{W}_{m,d}} \frac{\partial \mathcal{L}_t}{\partial h_{t,m}}$$

For the other parameters, we use a chain rule involving the hidden activations at each time step and add them up.

Backpropagation through time (BPTT)

- $\mathcal{L}_t$ aggregates the loss for from time t to T:

$$\mathcal{L}_t = \sum_{\tau=t}^{T} D(\mathbf{y}_\tau, \hat{\mathbf{y}}_\tau)$$



Putting all together:

$$\frac{\partial \mathcal{L}_1}{\partial \mathbf{W}^{\text{out}}_{n,m}} = \sum_t \frac{\partial \hat{y}_{t,n}}{\partial \mathbf{W}^{\text{out}}_{n,m}} \frac{\partial D(y_{t,n}, \hat{y}_{t,n})}{\partial \hat{y}_{t,n}}$$

$$\frac{\partial \mathcal{L}_1}{\partial \mathbf{U}_{m,m'}} = \sum_t \frac{\partial h_{t,m}}{\partial \mathbf{U}_{m,m'}} \frac{\partial \mathcal{L}_t}{\partial h_{t,m}}$$

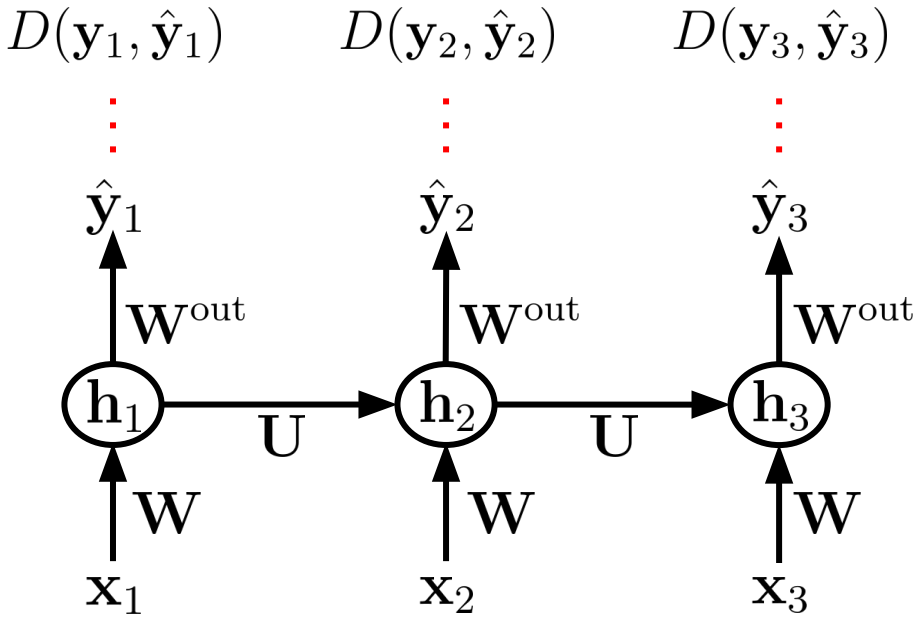
$$\frac{\partial \mathcal{L}_1}{\partial \mathbf{W}_{m,d}} = \sum_t \frac{\partial h_{t,m}}{\partial \mathbf{W}_{m,d}} \frac{\partial \mathcal{L}_t}{\partial h_{t,m}}$$

$$\frac{\partial \mathcal{L}_1}{\partial b_m} = \sum_t \frac{\partial h_{t,m}}{\partial b_m} \frac{\partial \mathcal{L}_t}{\partial h_{t,m}}$$

Backpropagation through time (BPTT)

- $\mathcal{L}_t$ aggregates the loss for from time t to T:

$$\mathcal{L}_t = \sum_{\tau=t}^{T} D(\mathbf{y}_\tau, \hat{\mathbf{y}}_\tau)$$



Summary (simpler notation):

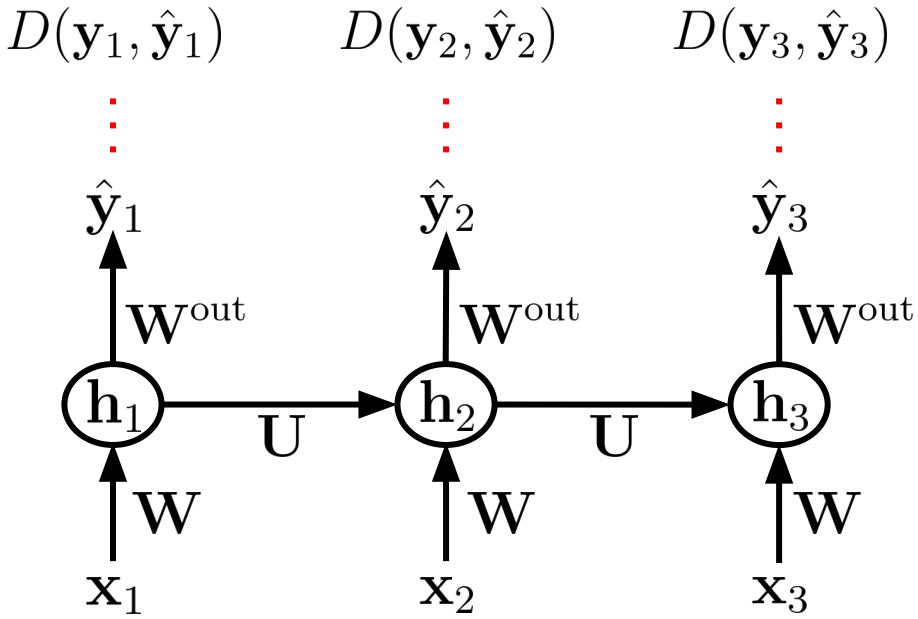
if $\theta \in \{\mathbf{W}^{\text{out}}\}$

$$\frac{\partial \mathcal{L}_1}{\partial \theta} = \sum_t \frac{\partial \hat{y}_{t,n}}{\partial \theta} \frac{\partial D(y_{t,n}, \hat{y}_{t,n})}{\partial \hat{y}_{t,n}}$$

Backpropagation through time (BPTT)

- $\mathcal{L}_t$ aggregates the loss for from time t to T:

$$\mathcal{L}_t = \sum_{\tau=t}^{T} D(\mathbf{y}_\tau, \hat{\mathbf{y}}_\tau)$$



Summary (simpler notation):

if $\theta \in \{\mathbf{W}^{\text{out}}\}$

$$\frac{\partial \mathcal{L}_1}{\partial \theta} = \sum_t \frac{\partial \hat{y}_{t,n}}{\partial \theta} \frac{\partial D(y_{t,n}, \hat{y}_{t,n})}{\partial \hat{y}_{t,n}}$$

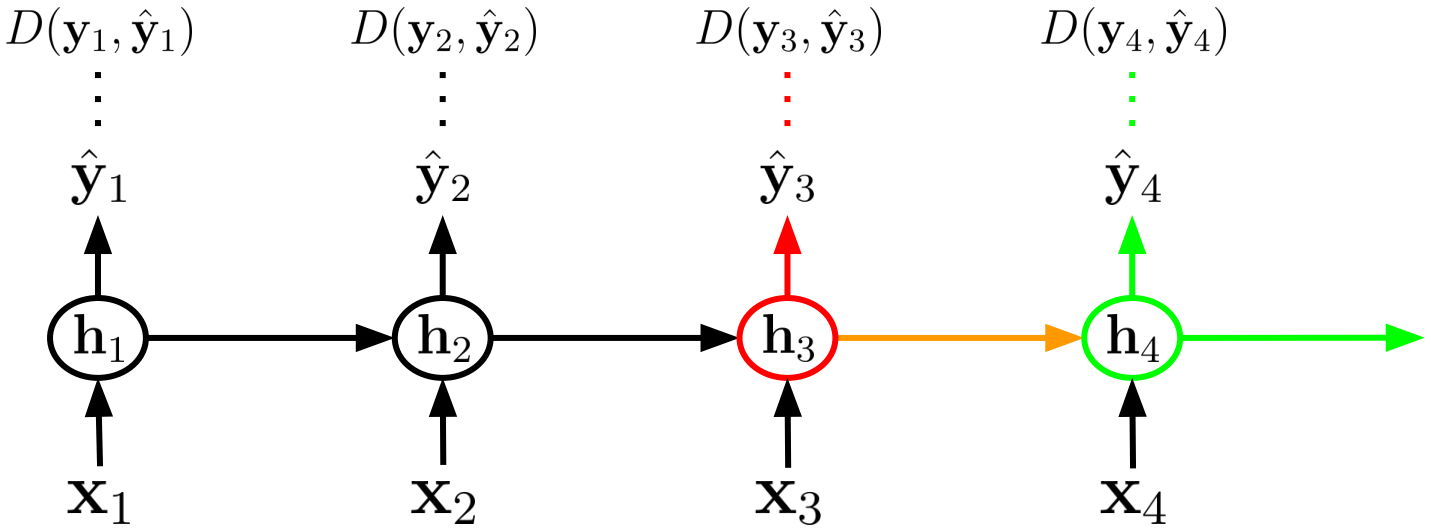
if $\theta \in \{\mathbf{W}, \mathbf{U}, \mathbf{b}\}$

$$\frac{\partial \mathcal{L}_1}{\partial \theta} = \sum_t \sum_m \frac{\partial h_{t,m}}{\partial \theta} \frac{\partial \mathcal{L}_t}{\partial h_{t,m}}$$

Note: we are not done yet! We still need to calculate $\frac{\partial \mathcal{L}_t}{\partial h_{t,m}}$ (see the next slides)

Backprop through time for calculating $\frac{\partial \mathcal{L}_t}{\partial h_{t,m}}$

- Gradient signal comes from the loss at time step 3



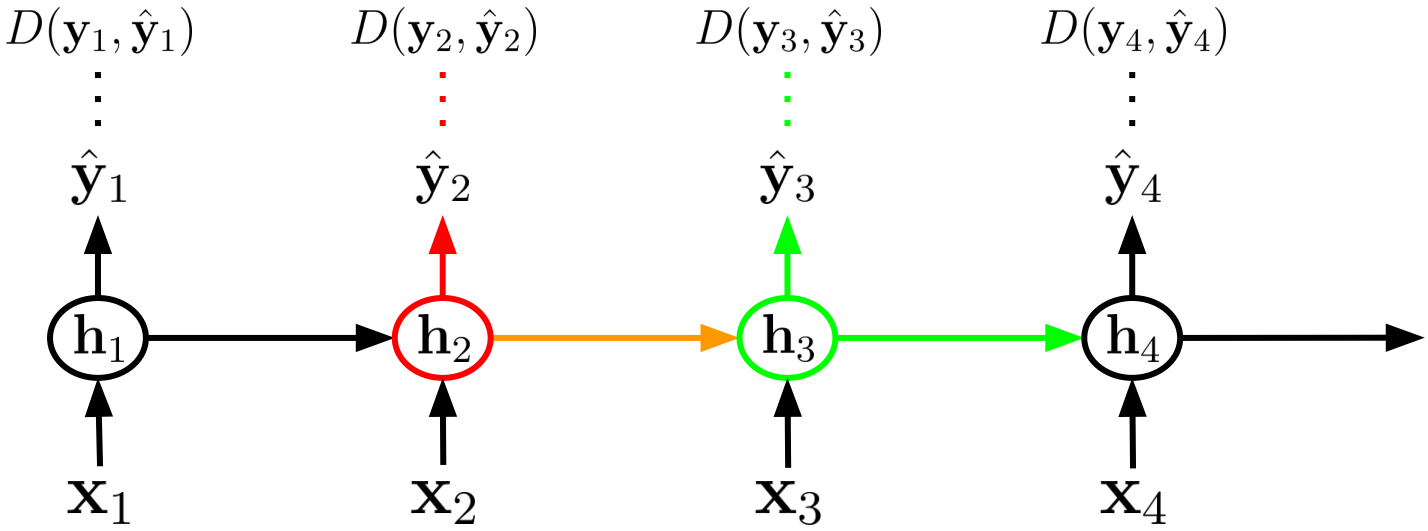
recursion:

$$\frac{\partial \mathcal{L}_3}{\partial h_{3,m}} = \frac{\partial D(\mathbf{y}_3, \hat{\mathbf{y}}_3)}{\partial h_{3,m}} + \sum_{m'} \frac{\partial h_{4,m'}}{\partial h_{3,m}} \frac{\partial \mathcal{L}_4}{\partial h_{4,m'}}$$

where $\mathcal{L}_3 = \sum_{\tau=3}^{T} D(\mathbf{y}_\tau, \hat{\mathbf{y}}_\tau)$

Backprop through time for calculating $\frac{\partial \mathcal{L}_t}{\partial h_{t,m}}$

- A similar process is applied for the hidden state at time step 2



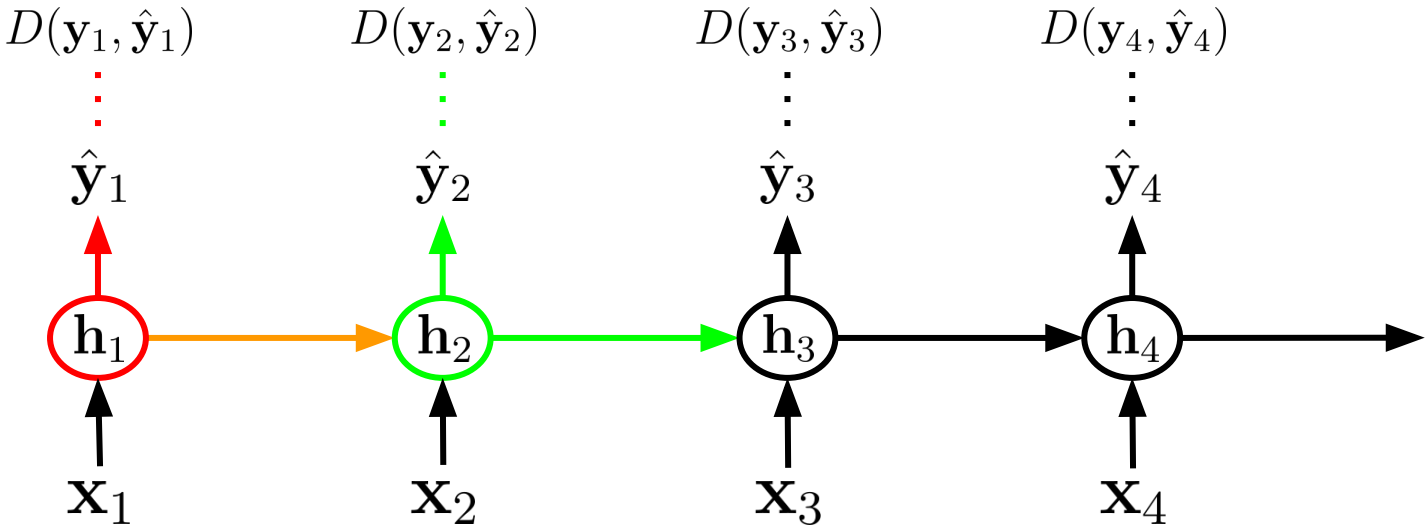
recursion:

$$\frac{\partial \mathcal{L}_2}{\partial h_{2,m}} = \frac{\partial D(\mathbf{y}_2, \hat{\mathbf{y}}_2)}{\partial h_{2,m}} + \sum_{m'} \frac{\partial h_{3,m'}}{\partial h_{2,m}} \frac{\partial \mathcal{L}_3}{\partial h_{3,m'}}$$

where $\mathcal{L}_2 = \sum_{\tau=2}^{T} D(\mathbf{y}_\tau, \hat{\mathbf{y}}_\tau)$

Backprop through time for calculating $\frac{\partial \mathcal{L}_t}{\partial h_{t,m}}$

- And it repeats up to timestep 1



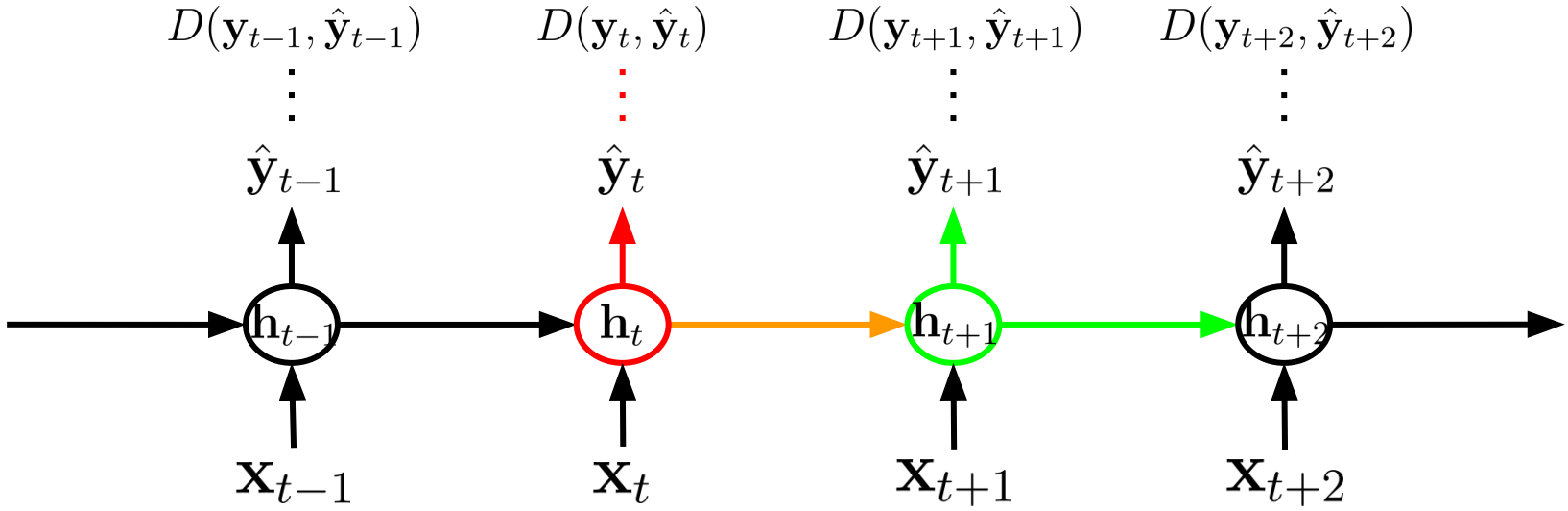
recursion:

$$\frac{\partial \mathcal{L}_1}{\partial h_{1,m}} = \frac{\partial D(\mathbf{y}_1, \hat{\mathbf{y}}_1)}{\partial h_{1,m}} + \sum_{m'} \frac{\partial h_{2,m'}}{\partial h_{1,m}} \frac{\partial \mathcal{L}_2}{\partial h_{2,m'}}$$

where $\mathcal{L}_1 = \sum_{\tau=1}^{T} D(\mathbf{y}_\tau, \hat{\mathbf{y}}_\tau)$

Backprop through time for calculating $\frac{\partial \mathcal{L}_t}{\partial h_{t,m}}$

- The general recursion formula for the gradient w.r.t. $\boldsymbol{h_t}$:



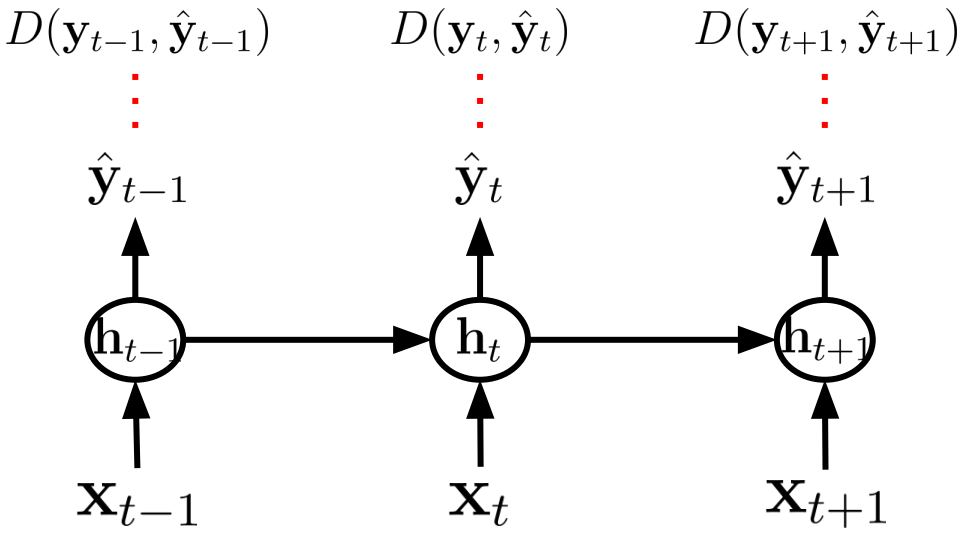
recursion (for general t):

$$\frac{\partial \mathcal{L}_t}{\partial h_{t,m}} = \frac{\partial D(\mathbf{y}_t, \hat{\mathbf{y}}_t)}{\partial h_{t,m}} + \sum_{m'} \frac{\partial h_{t+1,m'}}{\partial h_{t,m}} \frac{\partial \mathcal{L}_{t+1}}{\partial h_{t+1,m'}}$$

$$\text{where } \mathcal{L}_t = \sum_{\tau=t}^{T} D(\mathbf{y}_\tau, \hat{\mathbf{y}}_\tau)$$

Backpropagation through time (BPTT)

- Full summary: $\mathcal{L}_t = \sum_{\tau=t}^{T} D(\mathbf{y}_\tau, \hat{\mathbf{y}}_\tau)$



$$\frac{\partial \mathcal{L}_1}{\partial \mathbf{W}^{\text{out}}_{n,m}} = \sum_t \frac{\partial \hat{y}_{t,n}}{\partial \mathbf{W}^{\text{out}}_{n,m}} \frac{\partial D(y_{t,n}, \hat{y}_{t,n})}{\partial \hat{y}_{t,n}}$$

$$\frac{\partial \mathcal{L}_1}{\partial \mathbf{U}_{m,m'}} = \sum_t \frac{\partial h_{t,m}}{\partial \mathbf{U}_{m,m'}} \frac{\partial \mathcal{L}_t}{\partial h_{t,m}}$$

$$\frac{\partial \mathcal{L}_1}{\partial \mathbf{W}_{m,d}} = \sum_t \frac{\partial h_{t,m}}{\partial \mathbf{W}_{m,d}} \frac{\partial \mathcal{L}_t}{\partial h_{t,m}}$$

$$\frac{\partial \mathcal{L}_1}{\partial b_m} = \sum_t \frac{\partial h_{t,m}}{\partial b_m} \frac{\partial \mathcal{L}_t}{\partial h_{t,m}}$$

Recursion $\rightarrow$

$$\frac{\partial \mathcal{L}_t}{\partial h_{t,m}} = \frac{\partial D(\mathbf{y}_t, \hat{\mathbf{y}}_t)}{\partial h_{t,m}} + \sum_{m'} \frac{\partial h_{t+1,m'}}{\partial h_{t,m}} \frac{\partial \mathcal{L}_{t+1}}{\partial h_{t+1,m'}}$$

Outline

- RNN Basics
- Backpropagation through time (BPTT)
- **The vanishing/exploding gradients problem**
- Applications

The vanishing/exploding gradients problem

- The hidden-to-hidden connections in standard RNNs can cause gradient vanishing during backprop of the loss in the last step.

$$\text{Recall}: \mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$

$$\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_t} = \left[\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t}\right]^\top \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}$$

$$= \left(\prod_{i=t+1}^{T} \left[\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}}\right]^\top\right) \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} = \left(\prod_{i=t+1}^{T} \left[\text{diag}\left(\mathbf{h}_i\left(1-\mathbf{h}_i\right)\right)\mathbf{U}\right]^\top\right) \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}$$

$$= \left(\prod_{i=t+1}^{T} \mathbf{U}^\top \text{diag}\left(\mathbf{h}_i\left(1-\mathbf{h}_i\right)\right)\right) \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}$$

Note: $\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_t}$: mx1 vector (gradient vector) $\left[\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_t}\right]_i = \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_{t,i}}$

$\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t}$: mxm matrix (Jacobian matrix) $\left[\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t}\right]_{i,j} = \frac{\partial \mathbf{h}_{T,i}}{\partial \mathbf{h}_{t,j}}$

The vanishing/exploding gradients problem

- The hidden-to-hidden connections in standard RNNs can cause gradient vanishing during backprop of the loss in the last step.
- This is caused by the gradients with respect to the activation function being multiplied through time!

$$\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_t} = \left[\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t}\right]^\top \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}$$

$$= \left(\prod_{i=t+1}^{T} \mathbf{U}^\top \operatorname{diag}\left(\mathbf{h}_i \left(1 - \mathbf{h}_i\right)\right) \right) \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}$$

gradient of sigmoid causes vanishing!
Numbers < 1 being multiplied together

The vanishing/exploding gradients problem

- The hidden-to-hidden connections in standard RNNs can cause gradient vanishing during backprop of the loss in the last step.
- This is caused by the gradients with respect to the activation function being multiplied through time!
- Gradient can explode if the norm of **U** is large

$$\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_t} = \left[\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t}\right]^\top \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}$$

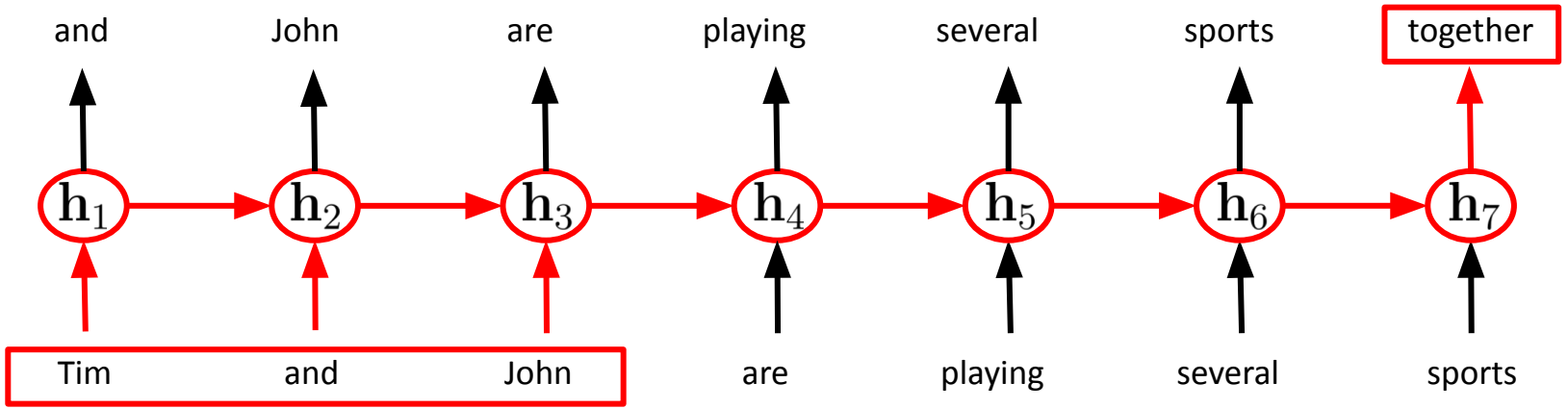
$$= \left(\prod_{i=t+1}^{T} \mathbf{U}^\top \operatorname{diag}\left(\mathbf{h}_i \left(1 - \mathbf{h}_i\right)\right) \right) \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}$$

gradient of sigmoid causes vanishing!
Numbers < 1 being multiplied together

(Gradient can also vanish if the norm of **U** is small.)

The vanishing/exploding gradients problem

- The hidden-to-hidden connections in standard RNNs can cause gradient vanishing during backprop of the loss in the last step.
- This is caused by the gradients with respect to the activation function being multiplied through time!
- Gradient can also explode if the norm of weight (e.g., **U**) is large.
- Long term dependencies in the sequence become affected!



The vanishing/exploding gradients problem

Gradient clipping

- A simple trick to prevent gradient explosion is to limit them to a max value.
- If the gradient is larger than η, then clip the norm to η:

$$\mathbf{g} \leftarrow \eta \frac{\mathbf{g}}{\|\mathbf{g}\|}$$

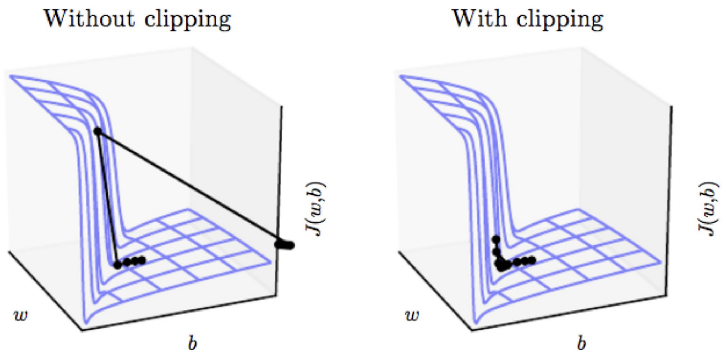


Diagram credit: Goodfellow et al., Deep Learning

The vanishing/exploding gradients problem

Additional tricks

- Identity initialization: Initialize the RNN weights to be the identity function, initialize the bias to zero and use ReLU activation.
  - This helps with stability during training
- Weight Regularization: Regularize the RNN weights to prevent large activations.
  - This prevents exploding gradients.

The vanishing/exploding gradients problem

Initialization tricks: Weight orthogonality*

- Orthogonal matrices have the property of preserving norms:

$$\|\mathbf{U}\mathbf{h}\| = \|\mathbf{h}\|$$

The vanishing/exploding gradients problem

Initialization tricks: Weight orthogonality*

- Orthogonal matrices have the property of preserving norms:

$$\|\mathbf{U}\mathbf{h}\| = \|\mathbf{h}\|$$

- By the definition of operator norms, given matrices **A** and **B**, and vector v, we have:

$$\|\mathbf{A}\mathbf{v}\| \leq \|\mathbf{A}\|\|\mathbf{v}\| \qquad \text{and} \qquad \|\mathbf{A}\mathbf{B}\| \leq \|\mathbf{A}\|\|\mathbf{B}\|$$

* Recall: operator norm for a matrix A

$$\|\mathbf{A}\|_{\text{op}} = \inf\{c \geq 0 : \|\mathbf{A}\mathbf{v}\| \leq c\|\mathbf{v}\| \text{ for all } \mathbf{v} \in R^d\}$$

$$= \sup_{\{\mathbf{v}\in\mathbb{R}^d, \mathbf{v}\neq 0\}} \frac{\|\mathbf{A}\mathbf{v}\|}{\|\mathbf{v}\|}$$

The vanishing/exploding gradients problem

Initialization tricks: Weight orthogonality*

- Now, given the gradient of the loss with respect to h_t:

$$\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} = \left[\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t}\right]^\top \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} = \prod_{k=t}^{T-1} \left[\frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k}\right]^\top \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} = \prod_{k=t}^{T-1} \left[\mathbf{U}^\top \mathbf{D}_{k+1}\right] \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}$$

The vanishing/exploding gradients problem

Initialization tricks: Weight orthogonality*

- Now, given the gradient of the loss with respect to h_t:

$$\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} = \left[\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t}\right]^\top \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} = \prod_{k=t}^{T-1} \left[\frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k}\right]^\top \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} = \prod_{k=t}^{T-1} \left[\mathbf{U}^\top \mathbf{D}_{k+1}\right] \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}$$

- Where $\mathbf{D}_{k+1}$ is a diagonal matrix of activation function gradients. Therefore, given an orthogonal weight matrix **U**, we have:

$$\left\|\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_t}\right\| = \left\|\left(\prod_{k=t}^{T-1} \mathbf{U}^\top \mathbf{D}_{k+1}\right) \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}\right\| \leq \left(\prod_{k=t}^{T-1} \left\|\mathbf{U}^\top \mathbf{D}_{k+1}\right\|\right) \left\|\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}\right\| = \left(\prod_{k=t}^{T-1} \left\|\mathbf{D}_{k+1}\right\|\right) \left\|\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}\right\|$$

$$\|\mathbf{A}\mathbf{v}\| \leq \|\mathbf{A}\|\|\mathbf{v}\|$$

Arjovsky, M., Shah, A., & Bengio, Y. Unitary evolution recurrent neural networks. In ICML 2016.

The vanishing/exploding gradients problem

Initialization tricks: Weight orthogonality*

- Now, given the gradient of the loss with respect to h_t:

$$\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} = \left[\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t}\right]^\top \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} = \prod_{k=t}^{T-1} \left[\frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k}\right]^\top \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} = \prod_{k=t}^{T-1} \left[\mathbf{U}^\top \mathbf{D}_{k+1}\right] \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}$$

- Where $\mathbf{D}_{k+1}$ is a diagonal matrix of activation function gradients. Therefore, given an orthogonal weight matrix **U**, we have:

$$\left\|\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_t}\right\| = \left\|\left(\prod_{k=t}^{T-1} \mathbf{U}^\top \mathbf{D}_{k+1}\right) \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}\right\| \leq \left(\prod_{k=t}^{T-1} \left\|\mathbf{U}^\top \mathbf{D}_{k+1}\right\|\right) \left\|\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}\right\| = \left(\prod_{k=t}^{T-1} \left\|\mathbf{D}_{k+1}\right\|\right) \left\|\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}\right\|$$

$$\|\mathbf{U}\mathbf{h}\| = \|\mathbf{h}\|$$

Arjovsky, M., Shah, A., & Bengio, Y. Unitary evolution recurrent neural networks. In ICML 2016.

The vanishing/exploding gradients problem

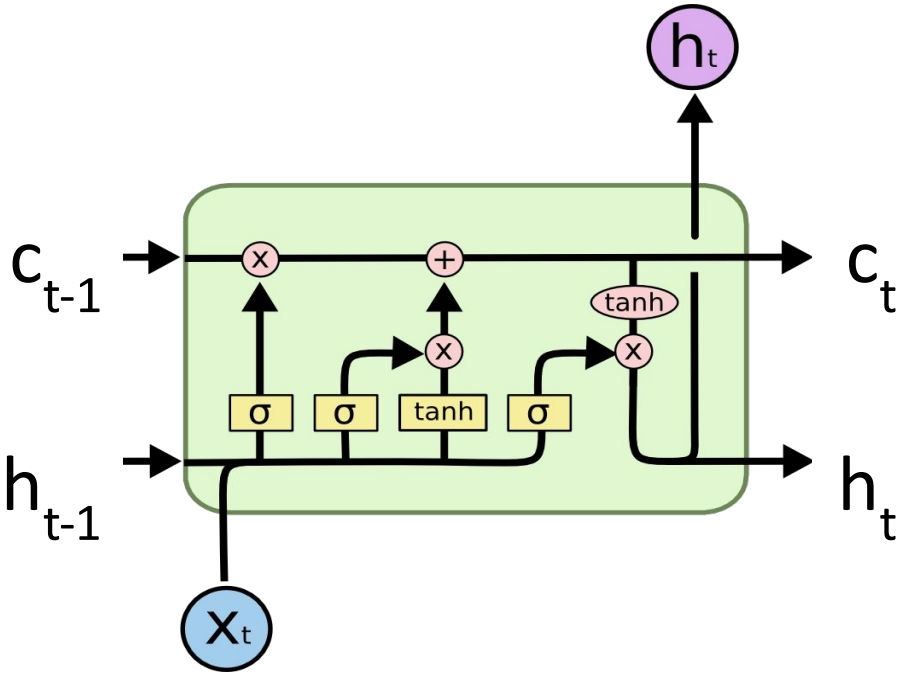
Initialization tricks: Weight orthogonality*

- Summary:
  - Orthogonal matrices have the property of preserving norms
  - Therefore, by initializing model weight matrices to be closer to orthogonal matrices, the norms of the activations are more likely preserved.
  - Thus the gradient is less likely to vanish or explode.

Arjovsky, M., Shah, A., & Bengio, Y. Unitary evolution recurrent neural networks. In ICML 2016.

The vanishing/exploding gradients problem

Long Short-Term Memory (LSTM)

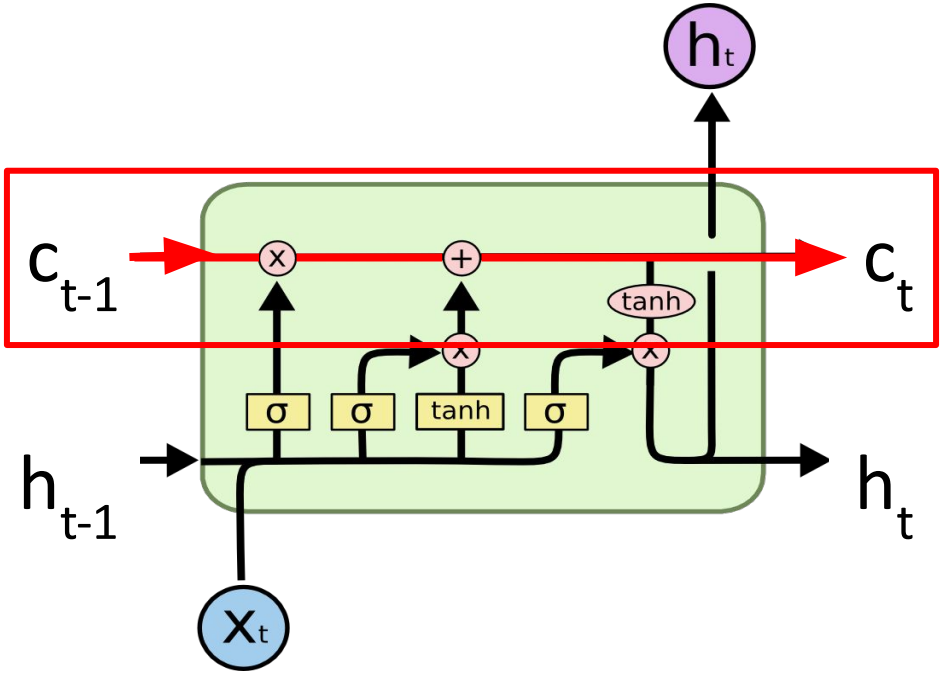


- A recurrent neural network variant designed to retain long-term dependencies.
- Helps dealing with both the vanishing and exploding gradient problem

Diagram credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs

The vanishing/exploding gradients problem

Long Short-Term Memory (LSTM)



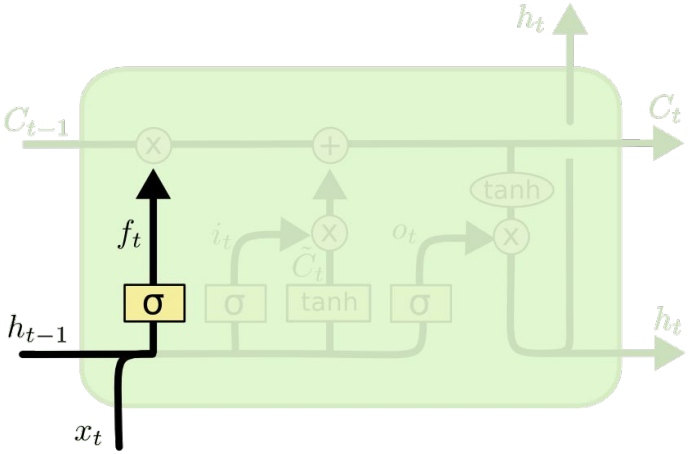
- A recurrent neural network variant designed to retain long-term dependencies.
- Helps dealing with both the vanishing and exploding gradient problem
- The key idea is an additive connection of previous memories passed through time

Diagram credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs

The vanishing/exploding gradients problem

Long Short-Term Memory (LSTM)

- The forget gate allows LSTM to choose to zero out part of previous memories and let others through.



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Notation: $\cdot$ is a matrix/vector product

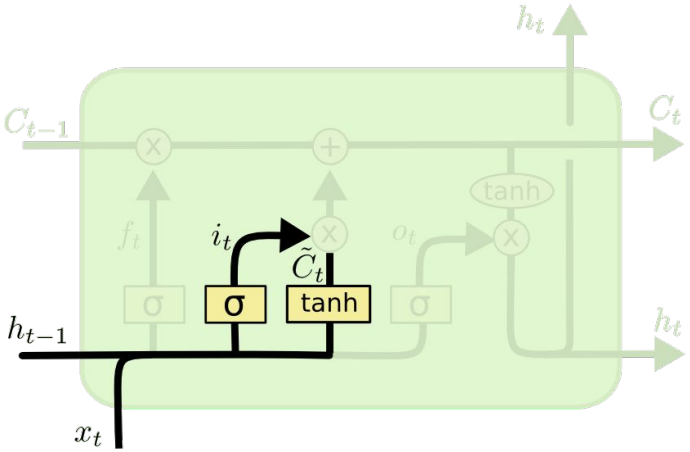
$*$ is an element-wise product

Diagram credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs

The vanishing/exploding gradients problem

Long Short-Term Memory (LSTM)

- The input gate behaves similar to the forget gate with new inputs.



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Notation: $\cdot$ is a matrix/vector product

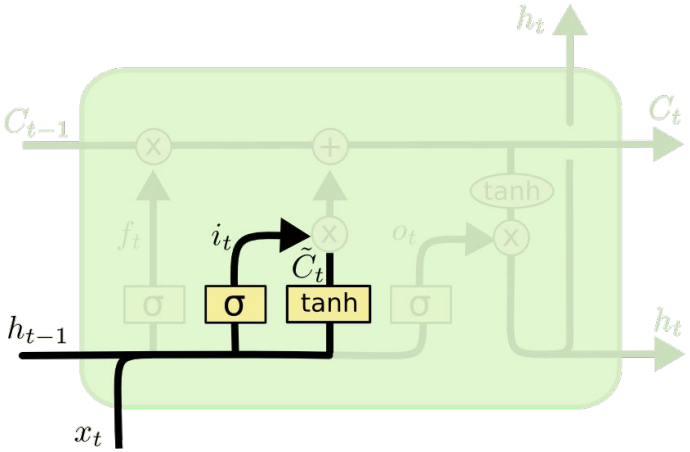
$*$ is an element-wise product

Diagram credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs

The vanishing/exploding gradients problem

Long Short-Term Memory (LSTM)

- The input gate behaves similar to the forget gate with new inputs.
- New information is computed from the current input and previous hidden units.



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Notation: $\cdot$ is a matrix/vector product

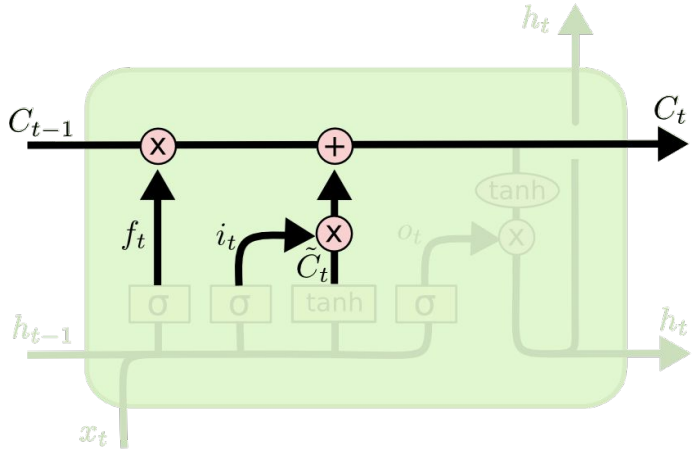
$*$ is an element-wise product

Diagram credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs

The vanishing/exploding gradients problem

Long Short-Term Memory (LSTM)

- Memories to be passed to the next steps are computed using the forget gate on the previous memories and the input gate on the current information found in the sequence



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Notation: $\cdot$ is a matrix/vector product

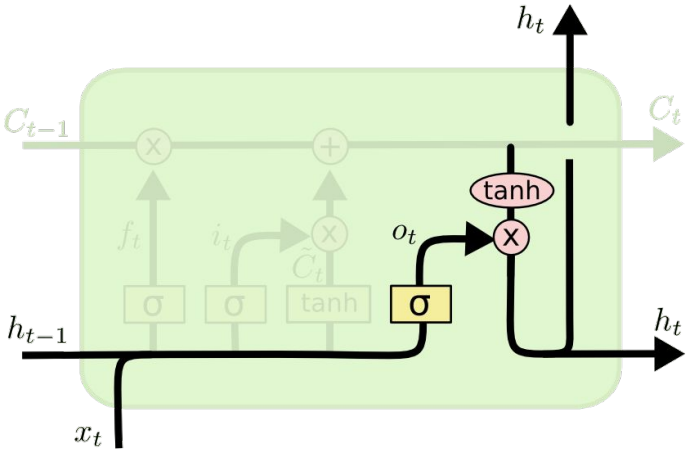
$*$ is an element-wise product

Diagram credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs

The vanishing/exploding gradients problem

Long Short-Term Memory (LSTM)

- An output gate is computed to choose information from the current memories for the next hidden state in the LSTM.



$$o_t = \sigma(W_o[h_{t-1}, x_t \ + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Notation: $\cdot$ is a matrix/vector product

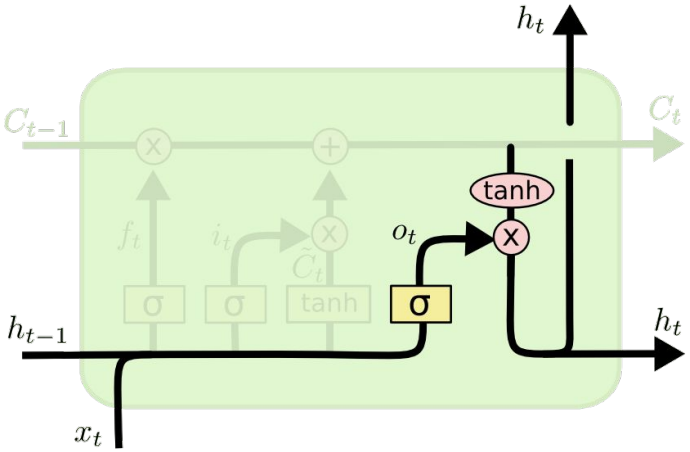
$*$ is an element-wise product

Diagram credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs

The vanishing/exploding gradients problem

Long Short-Term Memory (LSTM)

- An output gate is computed to choose information from the current memories for the next hidden state in the LSTM.
- Next hidden state is computed from the current memories and gate.



$$o_t = \sigma(W_o[h_{t-1}, x_t \ + b_o)$$

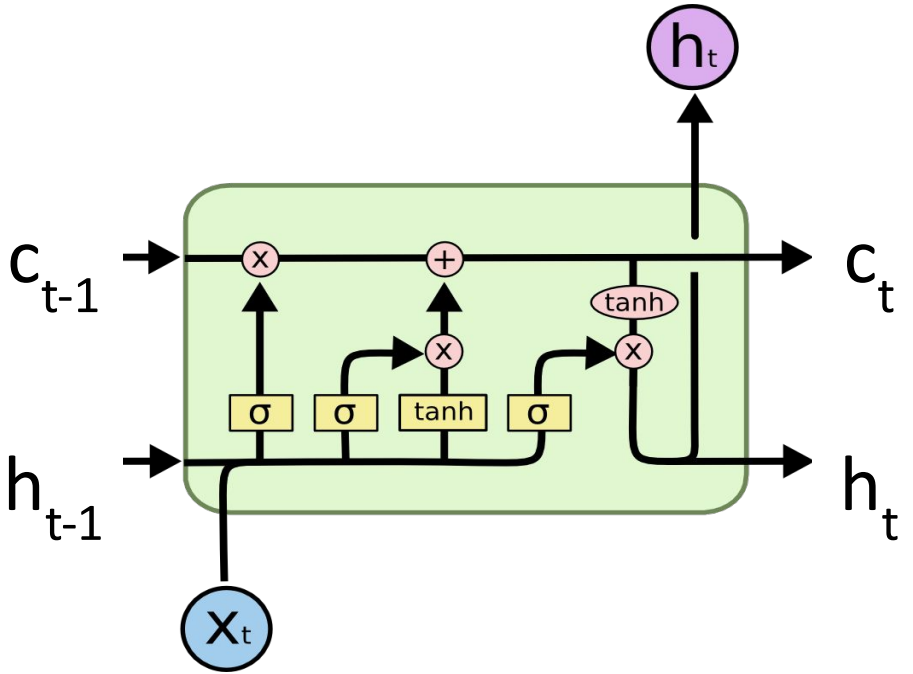
$$h_t = o_t * \tanh(C_t)$$

Notation: $\cdot$ is a matrix/vector product

$*$ is an element-wise product

Diagram credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs

Long Short-Term Memory (LSTM)



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma(W_o[h_{t-1}, x_t \ + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

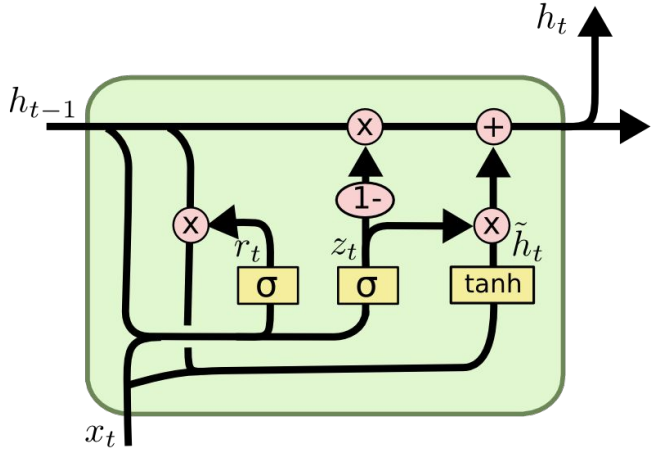
Notation: $\cdot$ is a matrix/vector product

$*$ is an element-wise product

Diagram credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs

Gated Recurrent Unit (GRU)

- A simplified variation of LSTM
  - The input / forget gates are simplified into a single gate z_t



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

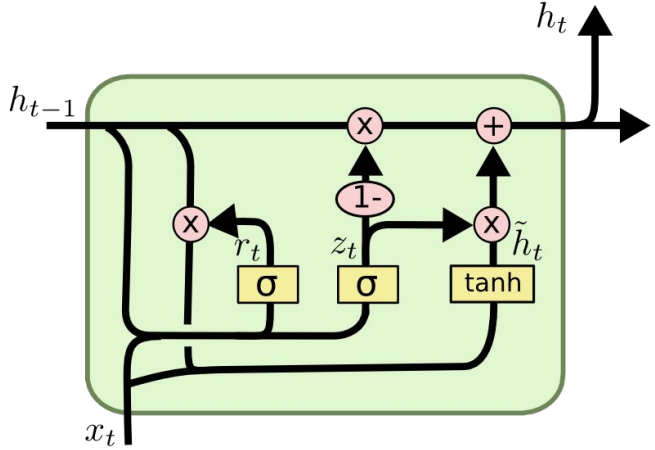
Notation: $\cdot$ is a matrix/vector product

$*$ is an element-wise product

Diagram credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs

Gated Recurrent Unit (GRU)

- A simplified variation of LSTM
  - The input / forget gates are simplified into a single gate z_t
  - C_t removed and consolidated with h_t
  - r_t introduced for gating (similar role as o_t for LSTM)



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$
$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Notation: $\cdot$ is a matrix/vector product
$*$ is an element-wise product

Diagram credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs

Multi-layer RNN

- To make RNNs more powerful, we can stack multiple layers of RNNs on top of each other

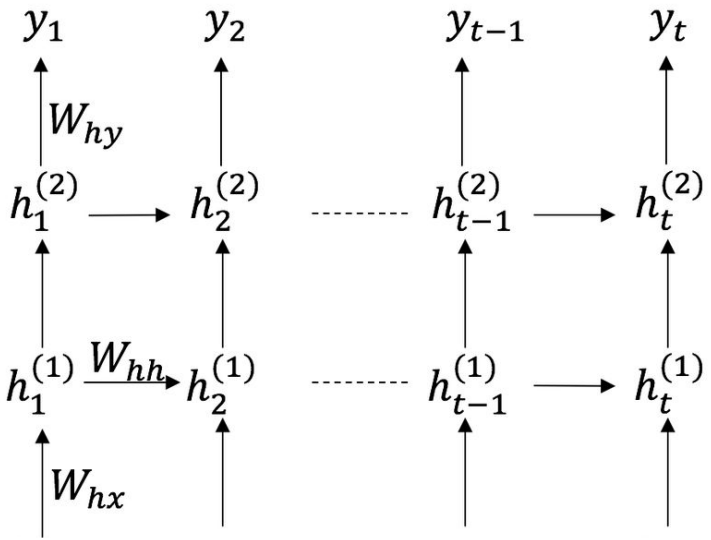


Diagram credit: https://arxiv.org/pdf/1707.08262

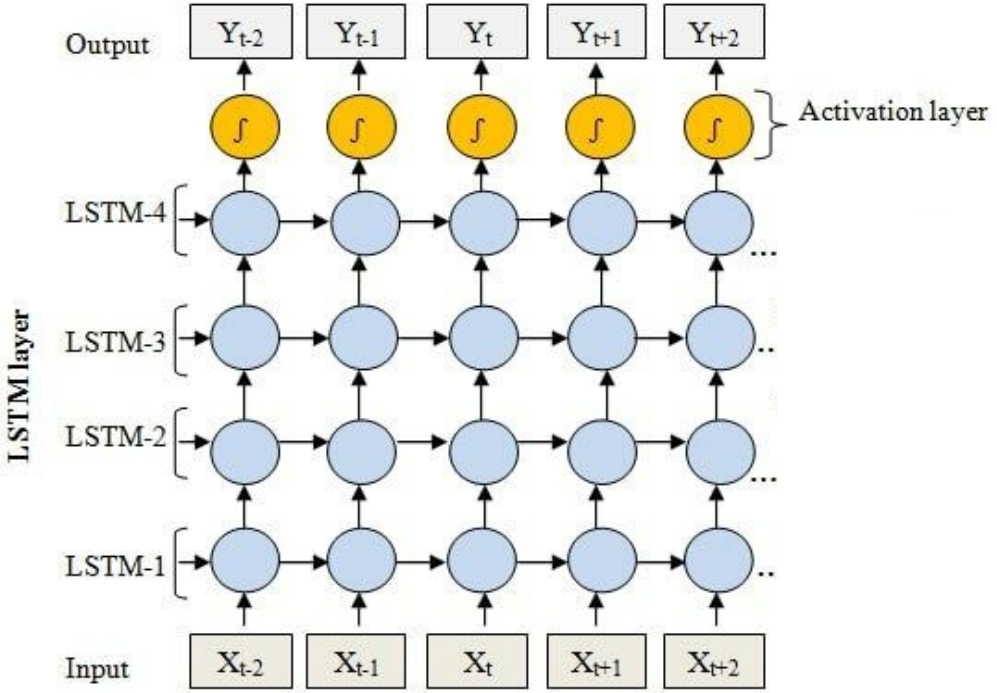


Diagram credit: https://link.springer.com/chapter/10.1007/978-3-030-86970-0_24

Outline

- RNN Basics
- Backpropagation through time (BPTT)
- The vanishing/exploding gradients problem
- **Applications**

Applications

Action Recognition

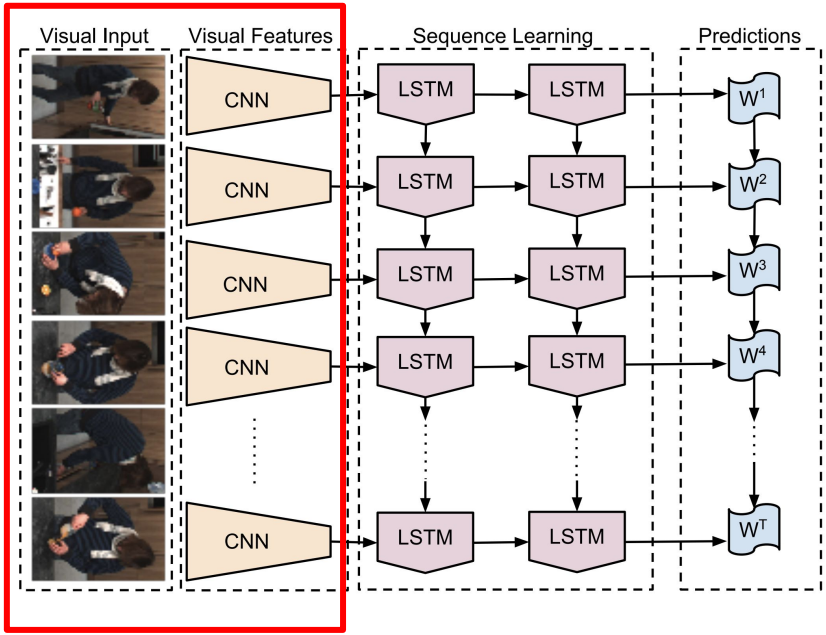
- Determine the action occurring in frame sequences

Figure credit: http://www.thumos.info

Applications

Action Recognition

- Encode images in compact representation



Convolutional Neural Network

Figure credit: Jeff Donahue et al, 2015

Applications

Action Recognition

- Encode images in compact representation
- Feed them to a multi-layer LSTM

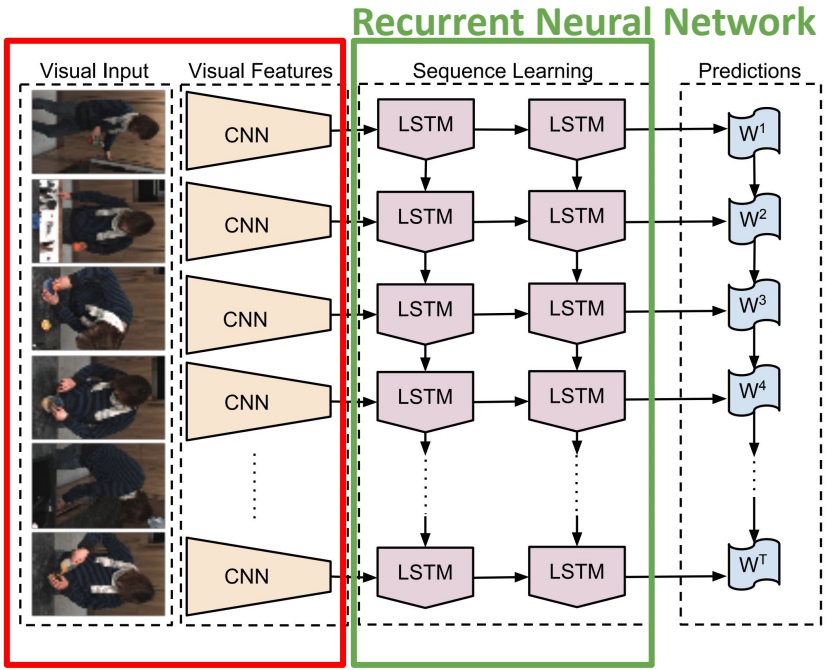


Figure credit: Jeff Donahue et al, 2015

Applications

Action Recognition

- Encode images in compact representation
- Feed them to a multi-layer LSTM
- Average predictions

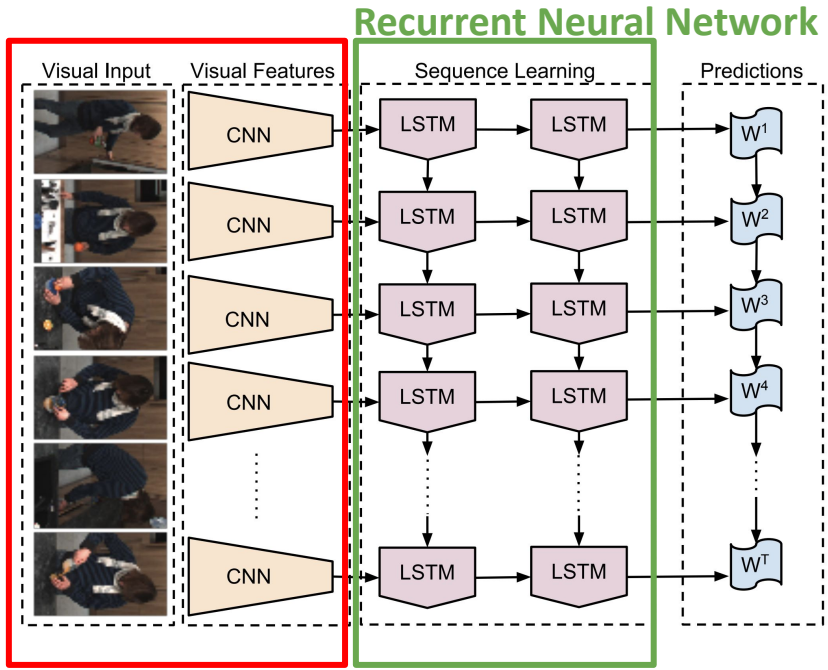
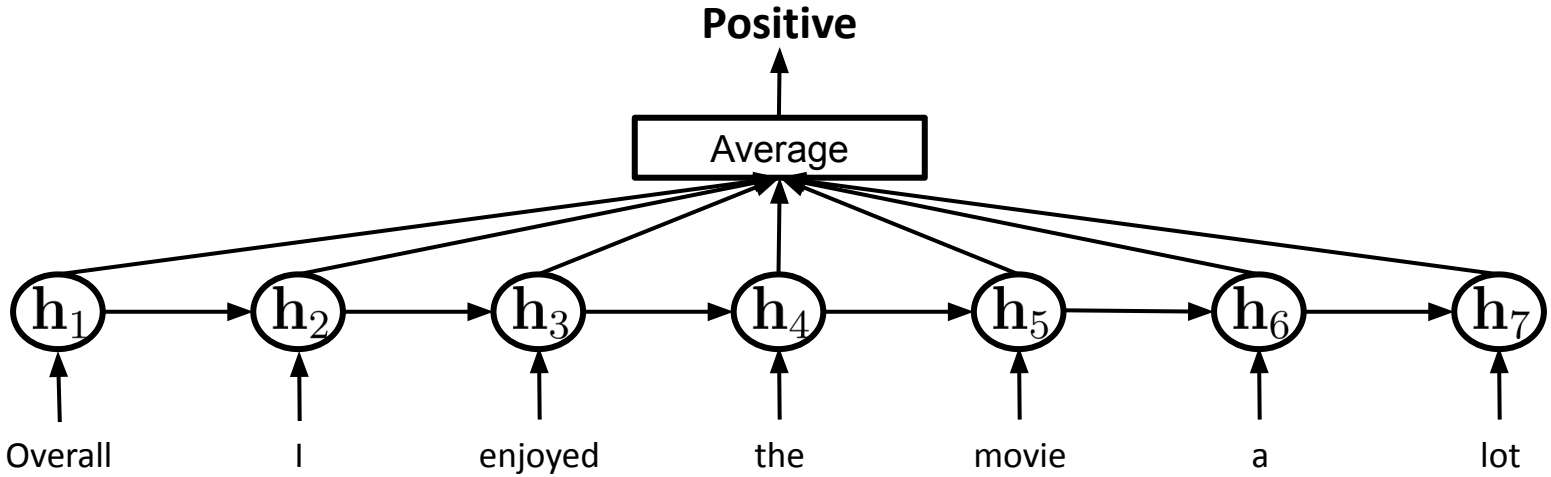


Figure credit: Jeff Donahue et al, 2015

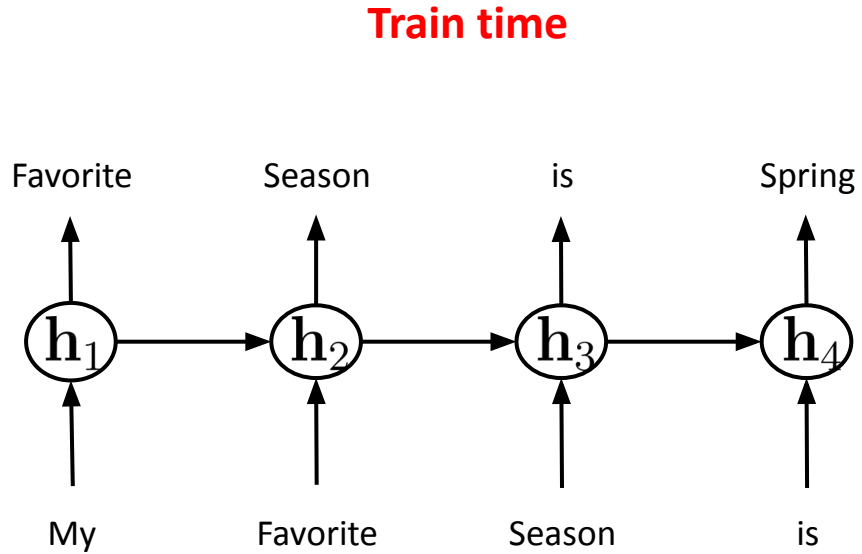
Applications: Sentence Classification

- **You can train RNNs on text!**
- Read a full sentence (or paragraph) and give it a rating
- Example: Movie reviews



Applications: Language modeling

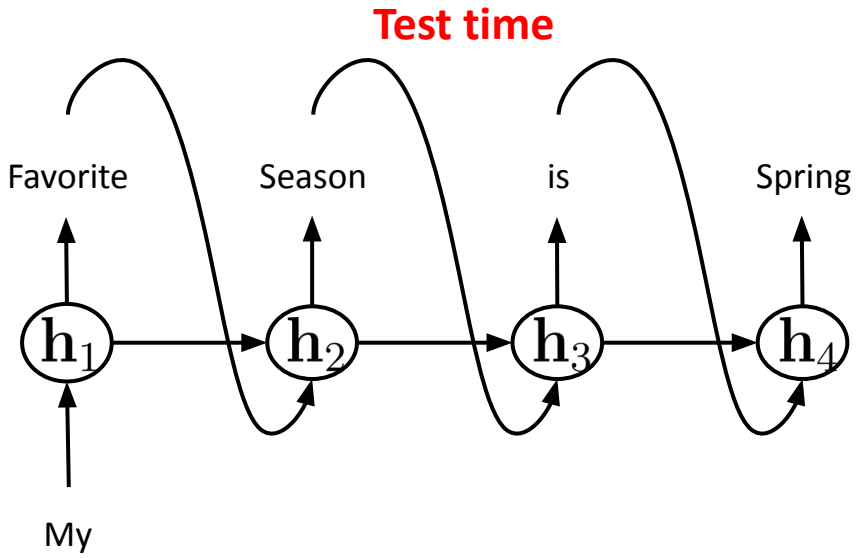
- **You can train RNNs on text!**
- Let the RNN read one word at a time and try to predict the next word
- This task requires long-term dependencies between words
- It can generate as many words as we let it generate



Slide credit: Abigail See

Applications: Language modeling

- **You can train RNNs on text!**
- Let the RNN read one word at a time and try to predict the next word
- This task requires long-term dependencies between words
- It can generate as many words as we let it generate



Slide credit: Abigail See

Applications: Language modeling

- You can train RNNs on any kind of text, and have it generate the same the style of text
- RNN trained on **Obama** speeches

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done.

Slide credit: Abigail See

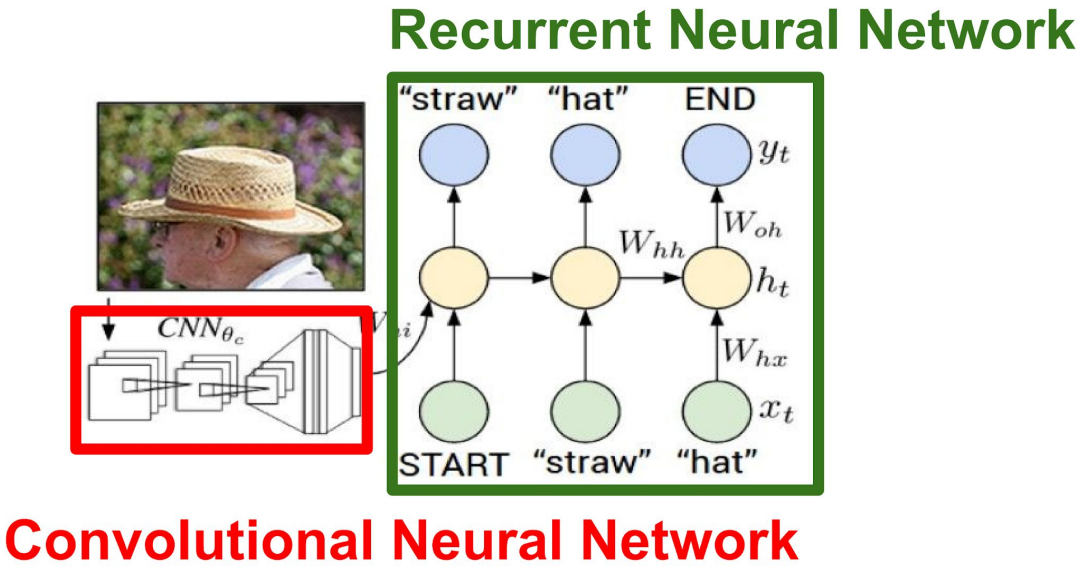
Applications: Language modeling

- You can train RNNs on any kind of text, and have it generate the same the style of text
- RNN trained on **Harry Potter** scripts

"No idea," said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry's shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn't felt it seemed. He reached the teams too.

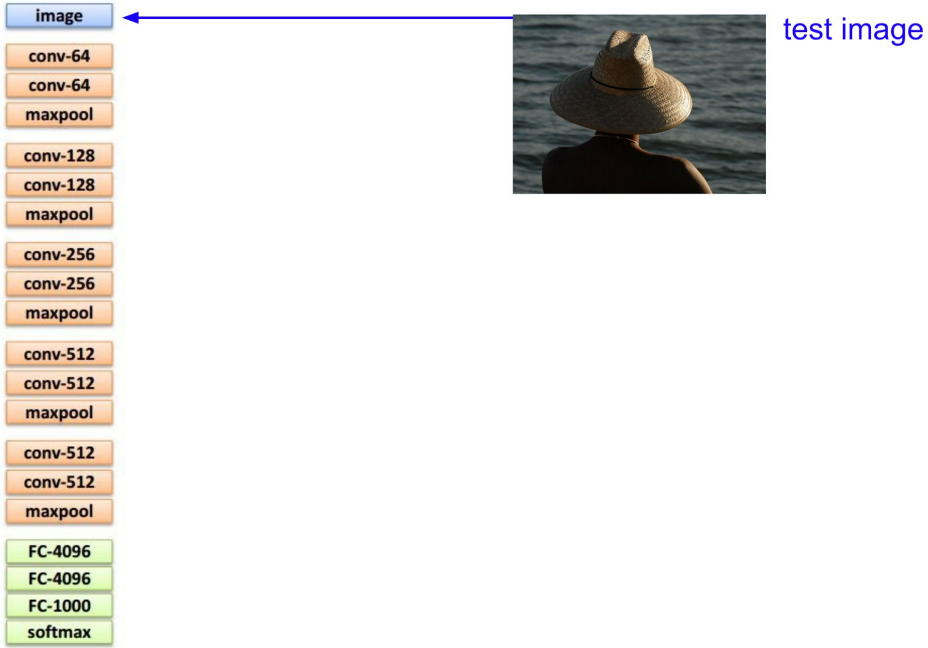
Slide credit: Abigail See

Applications: Image Captioning



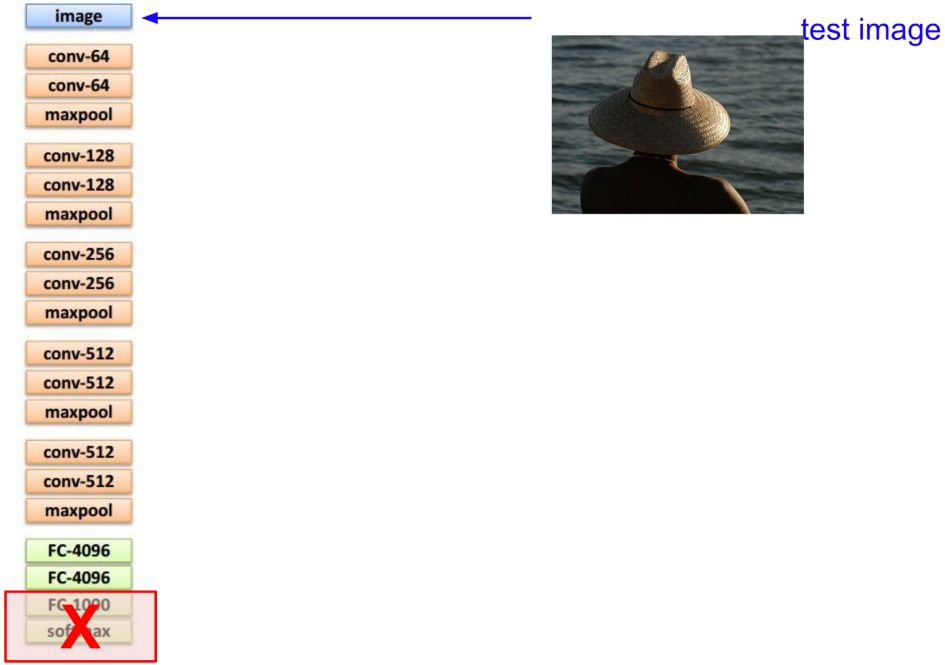
Slide credit: Fei fei Li, Justin Johnson and Serena Yeung

Applications: Image Captioning



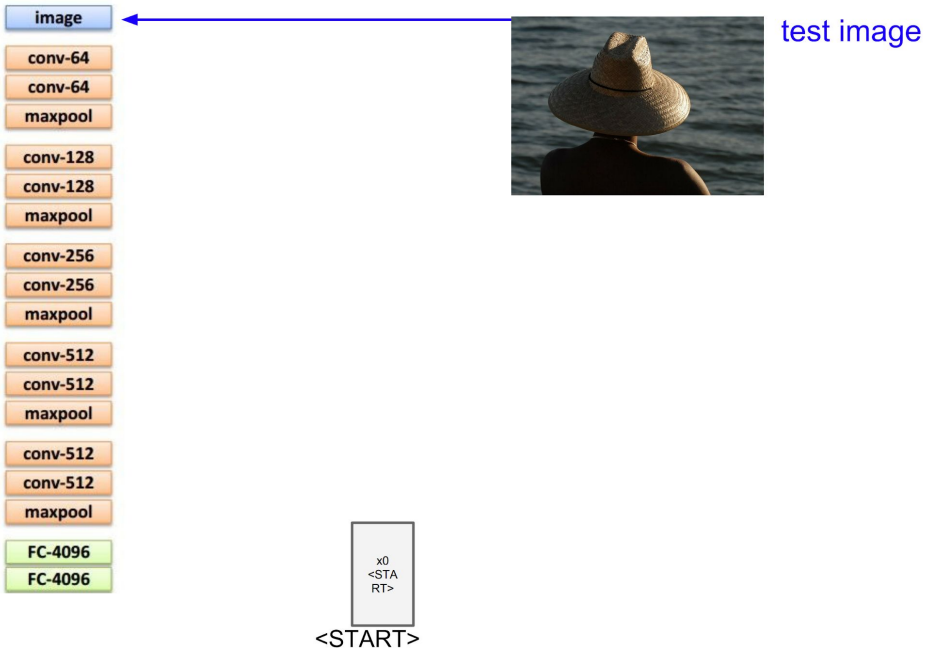
Slide credit: Fei fei Li, Justin Johnson and Serena Yeung

Applications: Image Captioning



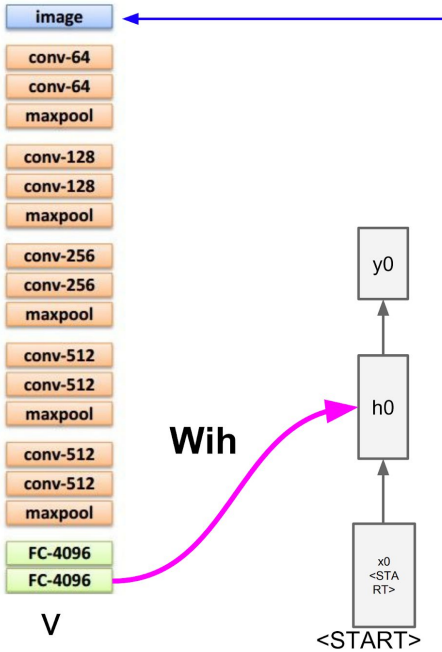
Slide credit: Fei fei Li, Justin Johnson and Serena Yeung

Applications: Image Captioning



Slide credit: Fei fei Li, Justin Johnson and Serena Yeung

Applications: Image Captioning



before:

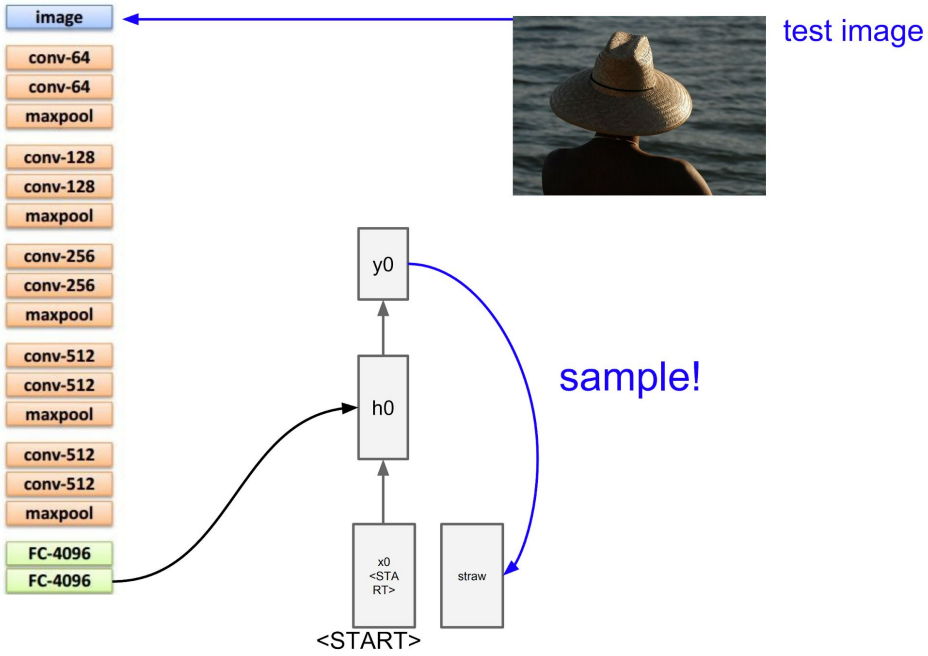
h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h + **Wih** * **v**)

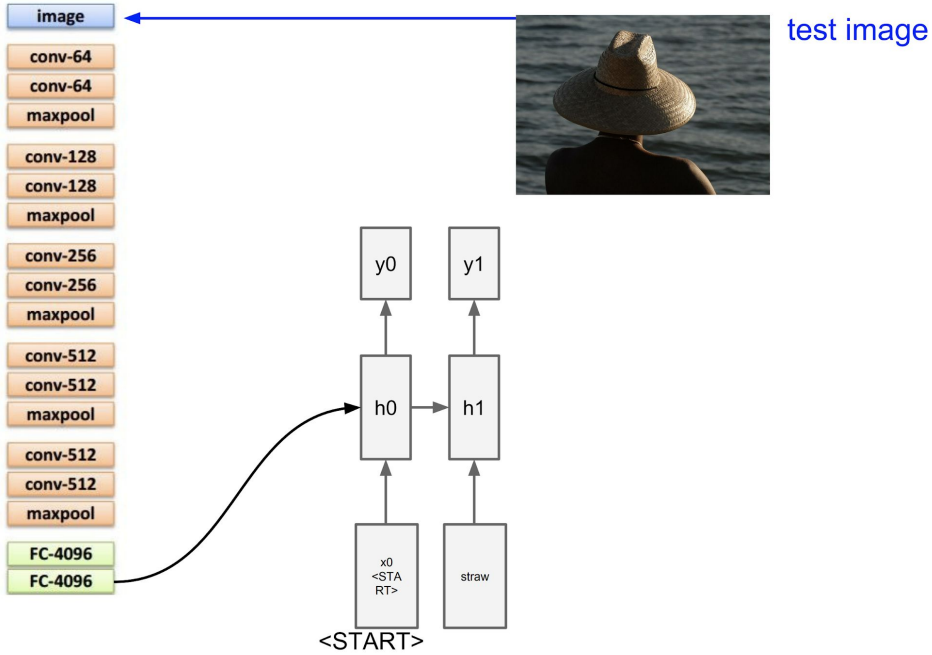
Slide credit: Fei fei Li, Justin Johnson and Serena Yeung

Applications: Image Captioning



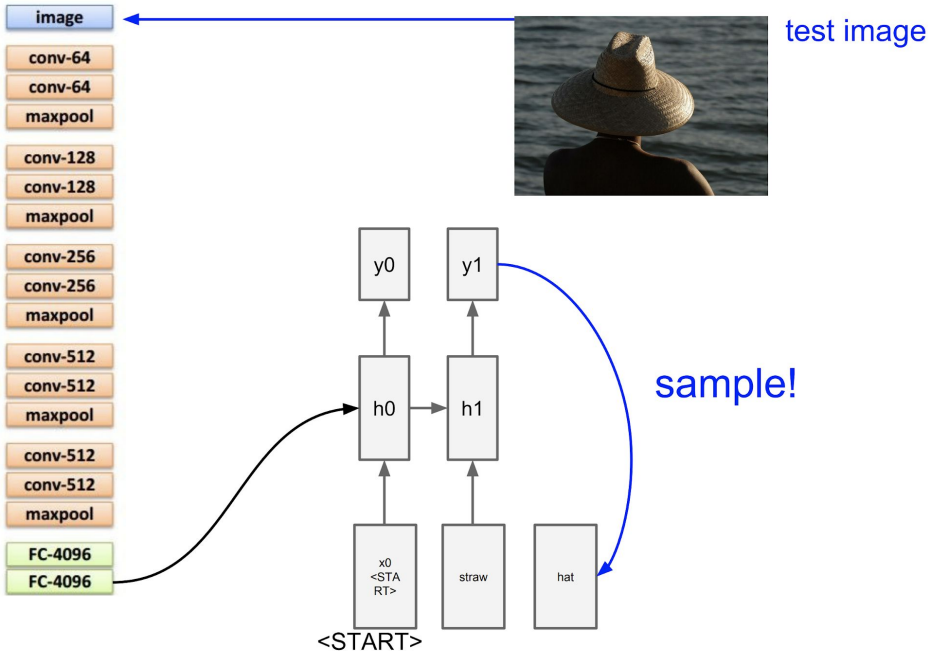
Slide credit: Fei fei Li, Justin Johnson and Serena Yeung

Applications: Image Captioning



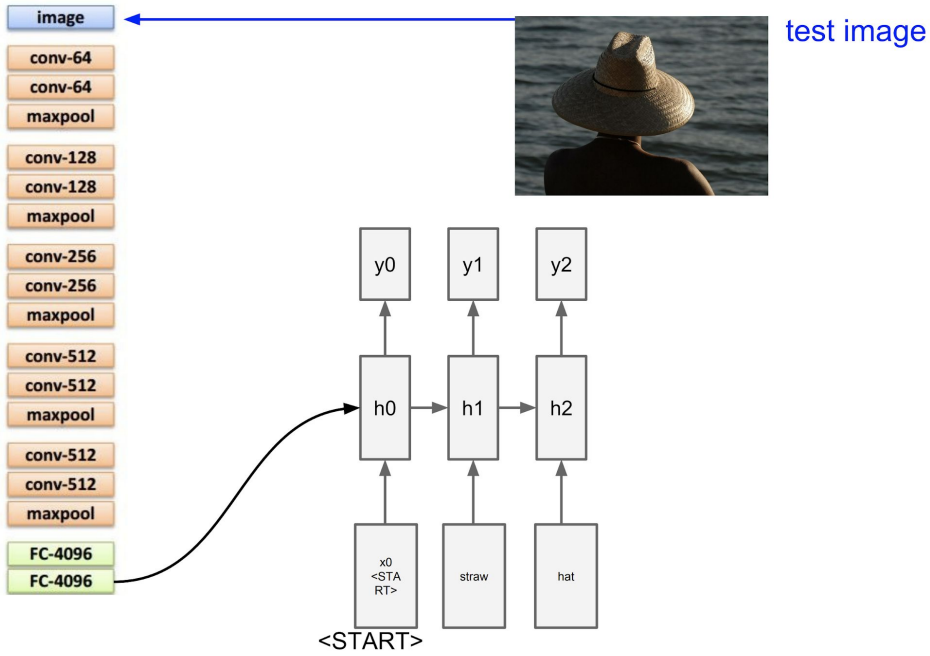
Slide credit: Fei fei Li, Justin Johnson and Serena Yeung

Applications: Image Captioning



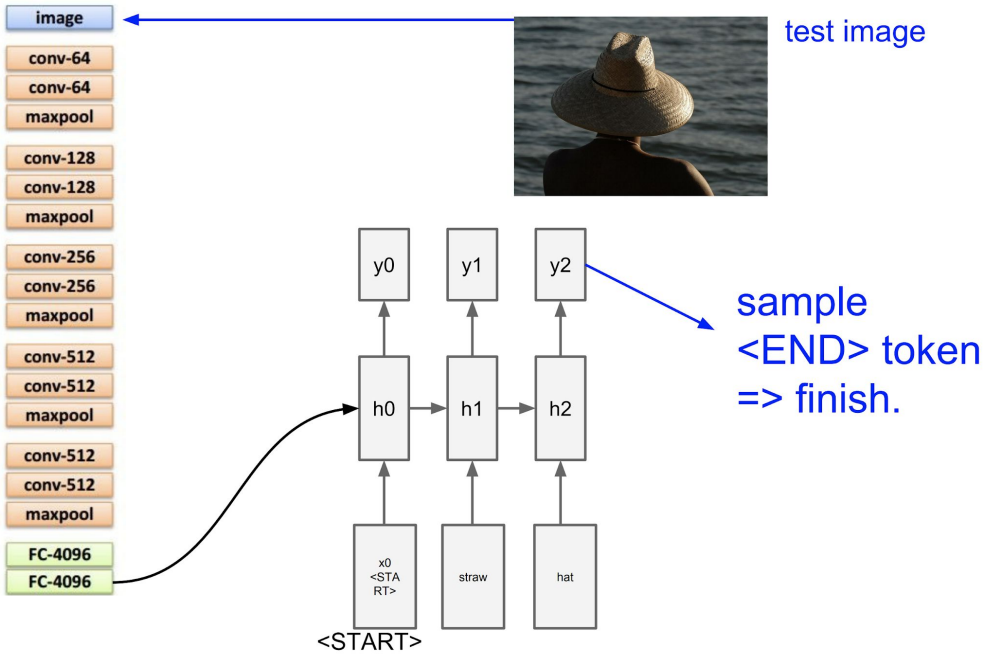
Slide credit: Fei fei Li, Justin Johnson and Serena Yeung

Applications: Image Captioning



Slide credit: Fei fei Li, Justin Johnson and Serena Yeung

Applications: Image Captioning



Slide credit: Fei fei Li, Justin Johnson and Serena Yeung

Applications: Image Captioning

Image Captioning: Example Results

Captions generated using neuraltalk2
All images are CC0 Public domain:
cat suitcase, cat tree, dog, bear,
surfers, tennis, giraffe, motorcycle

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

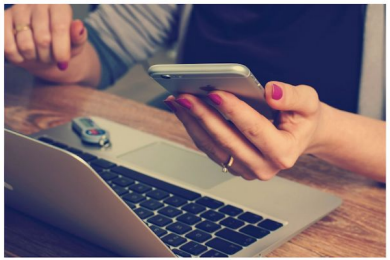
Slide credit: Fei fei Li, Justin Johnson and Serena Yeung

Applications: Image Captioning

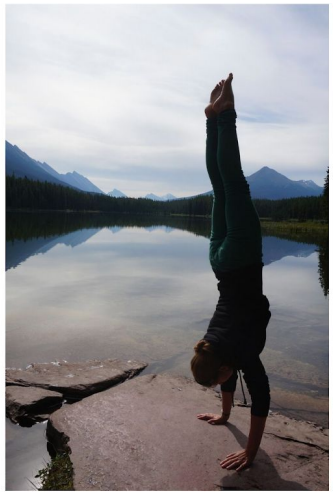
Image Captioning: Failure Cases

Captions generated using neuraltalk2
All images are CC0 Public domain: fur coat, handstand, spider web, baseball

A woman is holding a cat in her hand



A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard

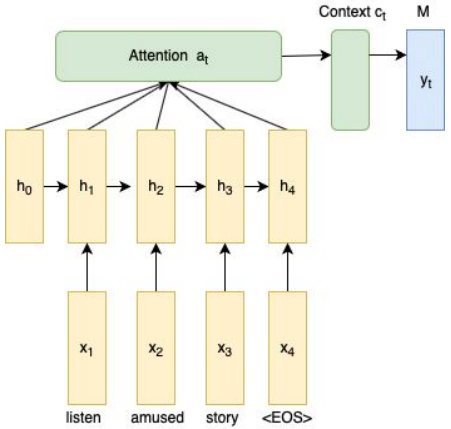
A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

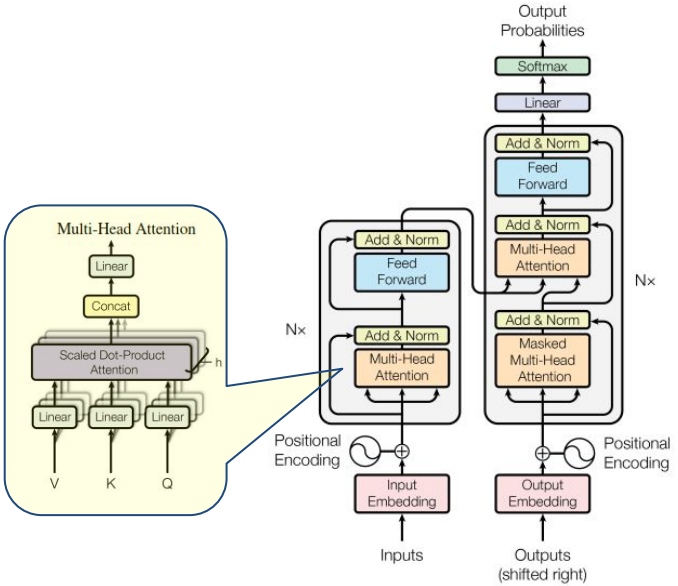
Slide credit: Fei fei Li, Justin Johnson and Serena Yeung

Remarks on SOTA methods

- Recent RNN/LSTM models incorporate attention mechanism, which allows for better modeling of long-term dependencies and more stable gradient flow.



- Many of the current SOTA methods are based on Transformer architectures, which use self-attention mechanism at the core. (Next lecture!)



References:

- Attention in RNNs
- Machine Translation With Sequence To Sequence Models
- Attention Is All You Need, 2017. (The first paper which proposed the transformer architecture.)
- The Illustrated Transformer

Summary

- RNN Basics
- Backpropagation through time (BPTT)
  - Backprop with recursive computation due to hidden-to-hidden connections over time
- The vanishing/exploding gradients problem
  - Tricks: Gradient clipping, Weight Initialization, Orthogonal weight matrices, LSTM/GRU, Transformers, etc.
- Applications
  - Video classification, text classification, image/video captioning, machine translation, etc.