Recurrent Neural Network

Instructor: Seunghoon Hong

Course overview

- ❏ Image classification
- ❏ Object detection
- ❏ Semantic segmentation
- ❏ Pose estimation
- ❏ Visualization
- ❏ Style transfer
- ❏ Adversarial attacks

- ❏ Text classification
- ❏ Machine translation
- ❏ Image captioning
- ❏ Visual question answering

- ❏ Image generation
- ❏ Text generation
- ❏ Text-to-image synthesis
- ❏ Img-to-img translation
- ❏ Unpaired img-to-img translation
- ❏ Interactive drawing

- ❏ Search algorithms
- ❏ Markov decision processes
- ❏ Reinforcement learning

We are here!

Convolutional Neural Networks (CNN)

Recurrent Neural Networks (RNN)

Deep generative models

Reinforcement Learning (RL)

Recap: Visual recognition with CNN

- Learning to associate input to pre-defined, task-specific labels

Input		**Model** (CNN)		**Output**
x	$\longrightarrow$	$f_\theta(x)$	$\longrightarrow$	y

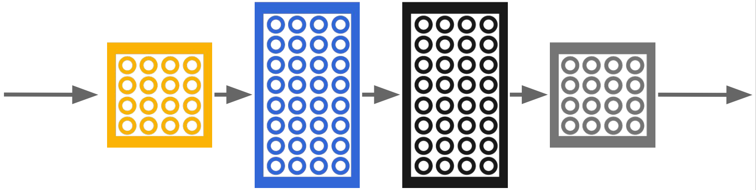
Recap: Visual recognition with CNN

- Learning to associate input to pre-defined, task-specific labels
- Examples: **classification**

input x

model

$$f_\theta(x)$$



output y

"dog"

"person"

"apple"

"elephant"

⋮

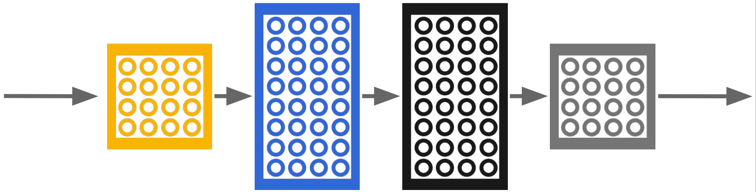
"field"

Recap: Visual recognition with CNN

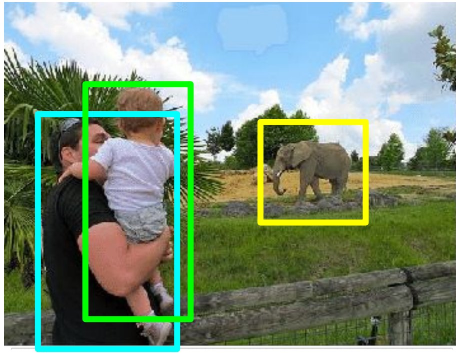
- Learning to associate input to pre-defined, task-specific labels
- Examples: **detection**

input x

$$f_\theta(x)$$



output y

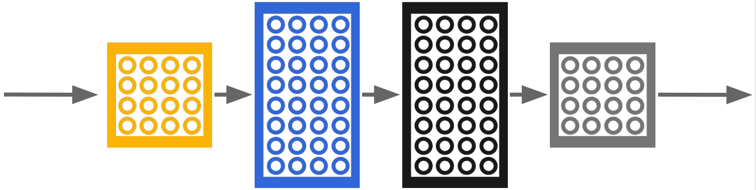


Recap: Visual recognition with CNN

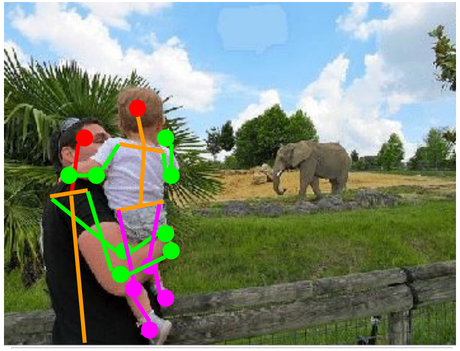
- Learning to associate input to pre-defined, task-specific labels
- Examples: **pose estimation**

input x

$$f_\theta(x)$$



output y

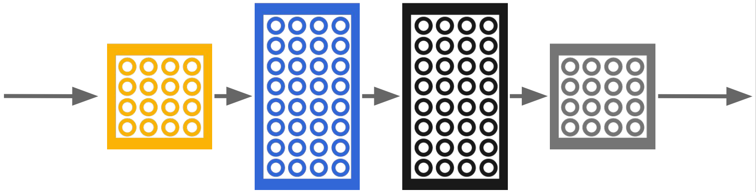


Recap: Visual recognition with CNN

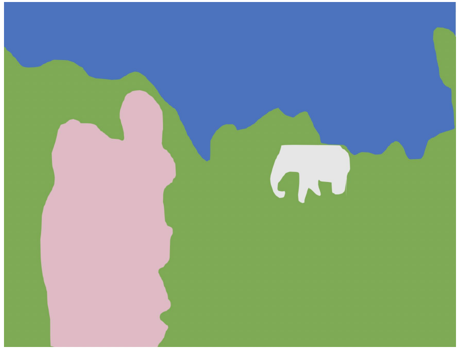
- Learning to associate input to pre-defined, task-specific labels
- Examples: **segmentation**

input x

$$f_\theta(x)$$



output y



Modeling sequences

What if we want to deal with a **sequential** data?

Today’s agenda

- RNN basics
- Backpropagation through time
- The vanishing/exploding gradients problem
- Advanced RNNs

Today’s agenda

- **RNN basics**
- Backpropagation through time
- The vanishing/exploding gradients problem
- Advanced RNNs

Modeling sequences with feedforward network

Simple case: modeling a fixed-size sequence

$$[\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \hat{\mathbf{y}}_3, \boxed{\hat{\mathbf{y}}_4}]$$

$$\mathbf{y}_t \in \mathbb{R}^n$$

How do we design a
neural network in this case?

$$\boxed{?}$$

$$[\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \boxed{\mathbf{x}_4}]$$

$$\mathbf{x}_t \in \mathbb{R}^d$$

Modeling sequences with feedforward network

Simple case: modeling a fixed-size sequence

- Option 1: MLP

$$[\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \hat{\mathbf{y}}_3, \hat{\mathbf{y}}_4]$$

$$\mathbf{W}^{out} \in \mathbb{R}^{4n \times m}$$

$$\mathbf{h}_1 \in \mathbb{R}^m$$

$$\mathbf{W}^{in} \in \mathbb{R}^{m \times 4d}$$

$$[\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4]$$

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

Modeling sequences with feedforward network

Simple case: modeling a fixed-size sequence

- Option 1: MLP
- What if we change the sequence length?
    - We cannot reuse the same network for handling sequences in different length!
    - It is because the network parameter bounds the length of the sequences!

$$[\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \hat{\mathbf{y}}_3, \hat{\mathbf{y}}_4, \hat{\mathbf{y}}_5]$$

$$\mathbf{W}^{out} \in \mathbb{R}^{\boxed{4}n \times m}$$

$$\mathbf{h}_1 \in \mathbb{R}^m$$

$$\mathbf{W}^{in} \in \mathbb{R}^{m \times \boxed{4}d}$$

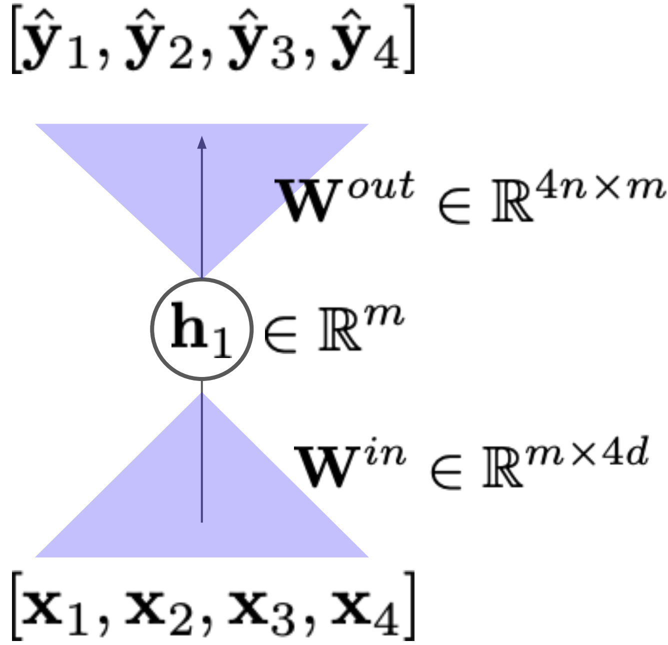
$$[\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5]$$

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

Modeling sequences with feedforward network

Simple case: modeling a fixed-size sequence

- Option 2: CNN

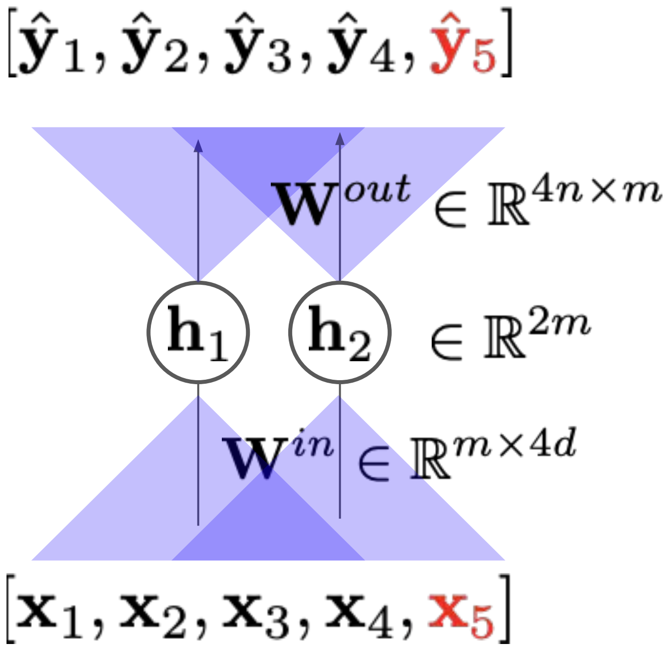


where $\mathbf{x}_t \in \mathbb{R}^d, \mathbf{h}_t \in \mathbb{R}^m, \mathbf{y}_t \in \mathbb{R}^n$

Modeling sequences with feedforward network

Simple case: modeling a fixed-size sequence

- Option 2: CNN
- What if we change the sequence length?
    - We can reuse the same network by sliding convolution filters over the sequence
- Problems?
    - The hidden representation grows with the length of the sequence!
    - The receptive field is fixed!



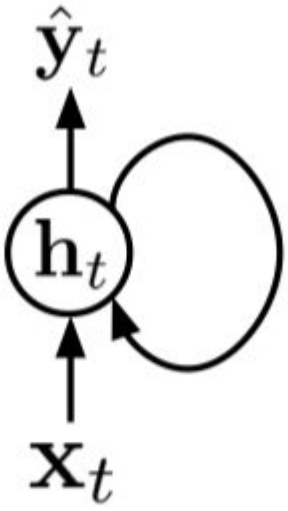
where $\mathbf{x}_t \in \mathbb{R}^d, \mathbf{h}_t \in \mathbb{R}^m, \mathbf{y}_t \in \mathbb{R}^n$

Modeling sequences efficiently

How can we model sequences in an efficient way?

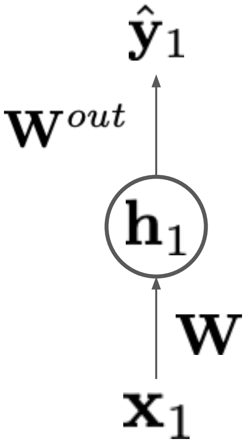
Use recursion!

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



Recurrent Neural Network (RNN)

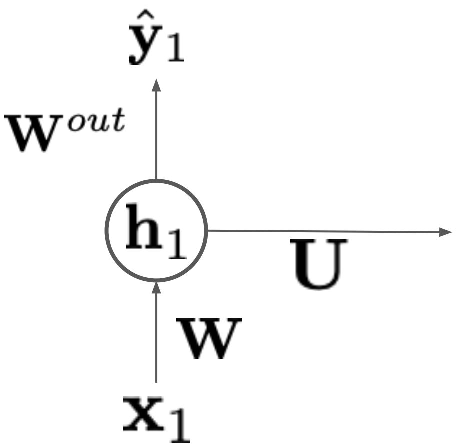
Unrolling RNNs through time



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

Recurrent Neural Network (RNN)

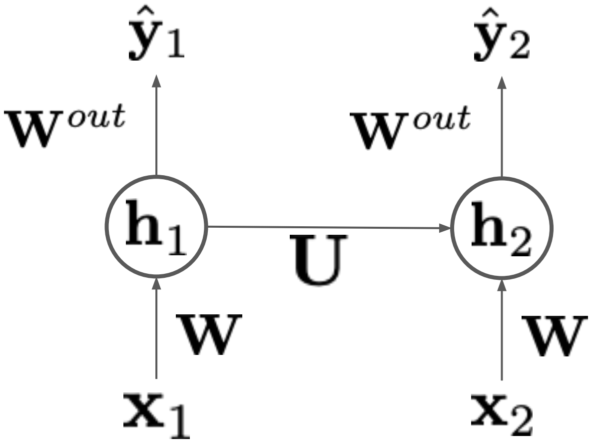
Unrolling RNNs through time



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

Recurrent Neural Network (RNN)

Unrolling RNNs through time



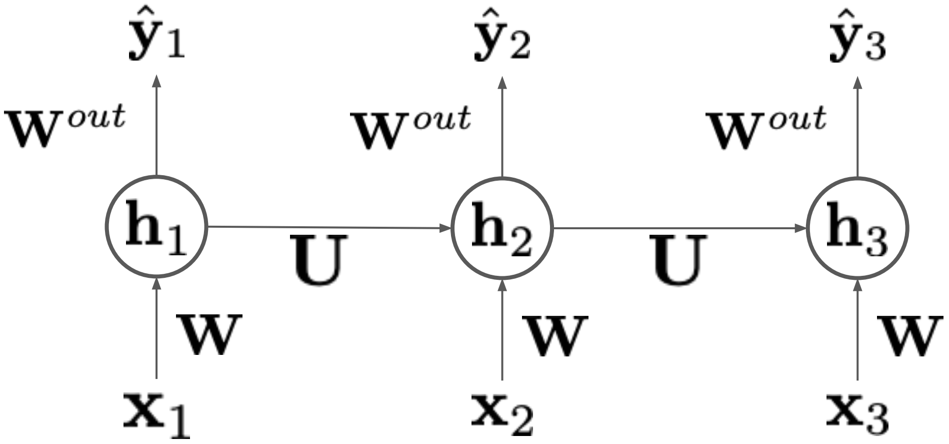
$$\mathbf{h}_1 = \sigma(\mathbf{W}\mathbf{x}_1 + \mathbf{b})$$
$$\hat{\mathbf{y}}_1 = \mathbf{W}^{out}\mathbf{h}_1$$

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

We have a temporal connection that models a temporal dependency!

Recurrent Neural Network (RNN)

Unrolling RNNs through time



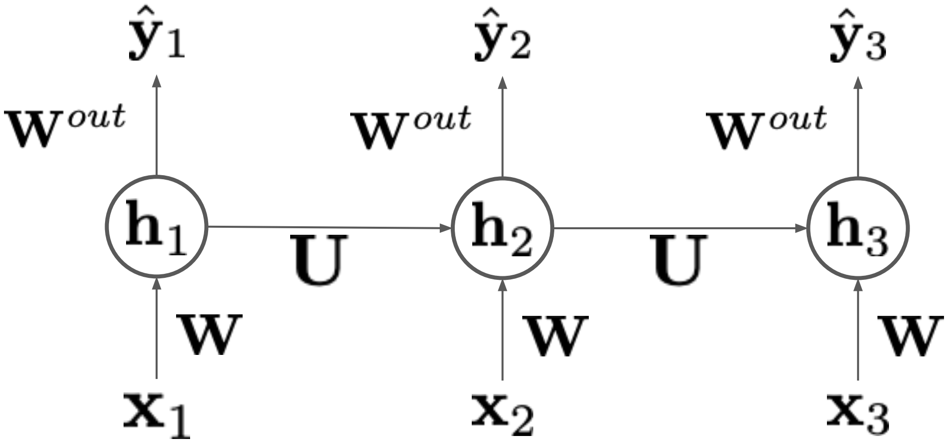
$$\mathbf{h}_1 = \sigma(\mathbf{W}\mathbf{x}_1 + \mathbf{b})$$
$$\hat{\mathbf{y}}_1 = \mathbf{W}^{out}\mathbf{h}_1$$

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

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

Recurrent Neural Network (RNN)

Unrolling RNNs through time



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

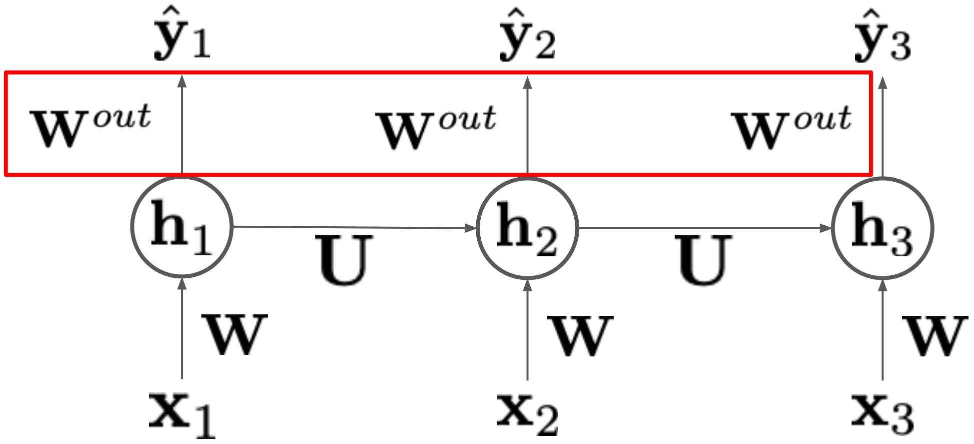
$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$

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

$$\mathbf{h}_0 = \mathbf{0}$$

Recurrent Neural Network (RNN)

Unrolling RNNs through time



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

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

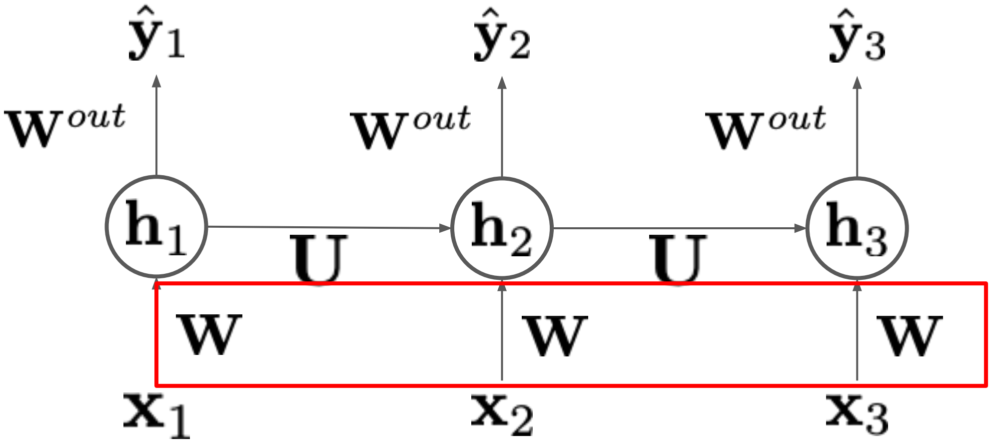
$$\hat{\mathbf{y}}_t = \boxed{\mathbf{W}^{out}}\mathbf{h}_t$$

$$\mathbf{h}_0 = \mathbf{0}$$

Weights are shared over time!

Recurrent Neural Network (RNN)

Unrolling RNNs through time



Weights are shared over time!

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

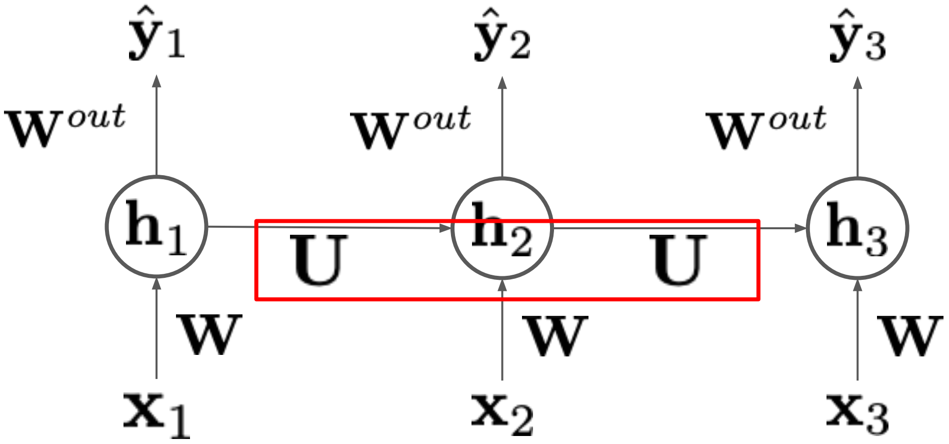
$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \boxed{\mathbf{W}}\mathbf{x}_t + \mathbf{b})$$

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

$$\mathbf{h}_0 = \mathbf{0}$$

Recurrent Neural Network (RNN)

Unrolling RNNs through time



Weights are shared over time!

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

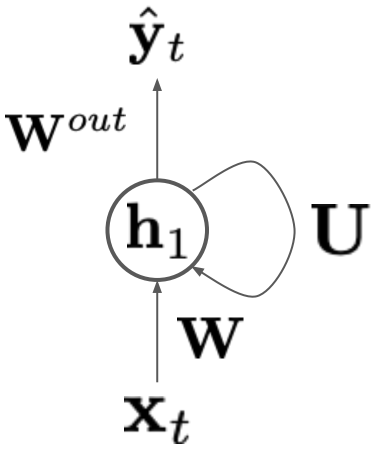
$$\mathbf{h}_t = \sigma(\boxed{\mathbf{U}}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$

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

$$\mathbf{h}_0 = \mathbf{0}$$

Recurrent Neural Network (RNN)

Unrolling RNNs through time



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

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

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

Quiz: size of weights? $(\mathbf{x}_t \in \mathbb{R}^d, \mathbf{h}_t \in \mathbb{R}^m, \mathbf{y}_t \in \mathbb{R}^n)$

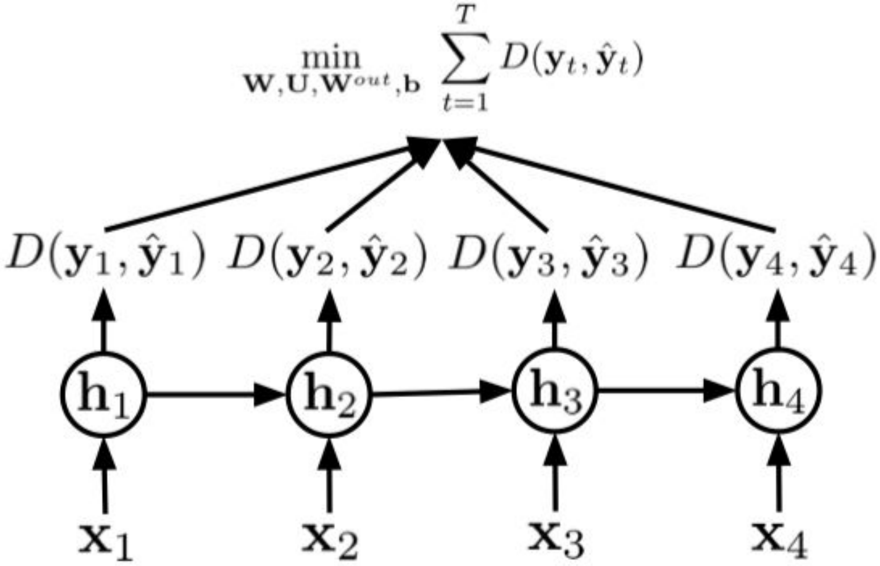
$\mathbf{W} = \mathbb{R}^{m \times d}$ $\quad \mathbf{U} = \mathbb{R}^{m \times m}$ $\quad \mathbf{W}^{out} = \mathbb{R}^{n \times m}$

Recurrent Neural Network (RNN)

Loss computation

- Compute loss (if any) for each step and aggregate

- Gradient flows through all unrolled steps in RNN



Today’s agenda

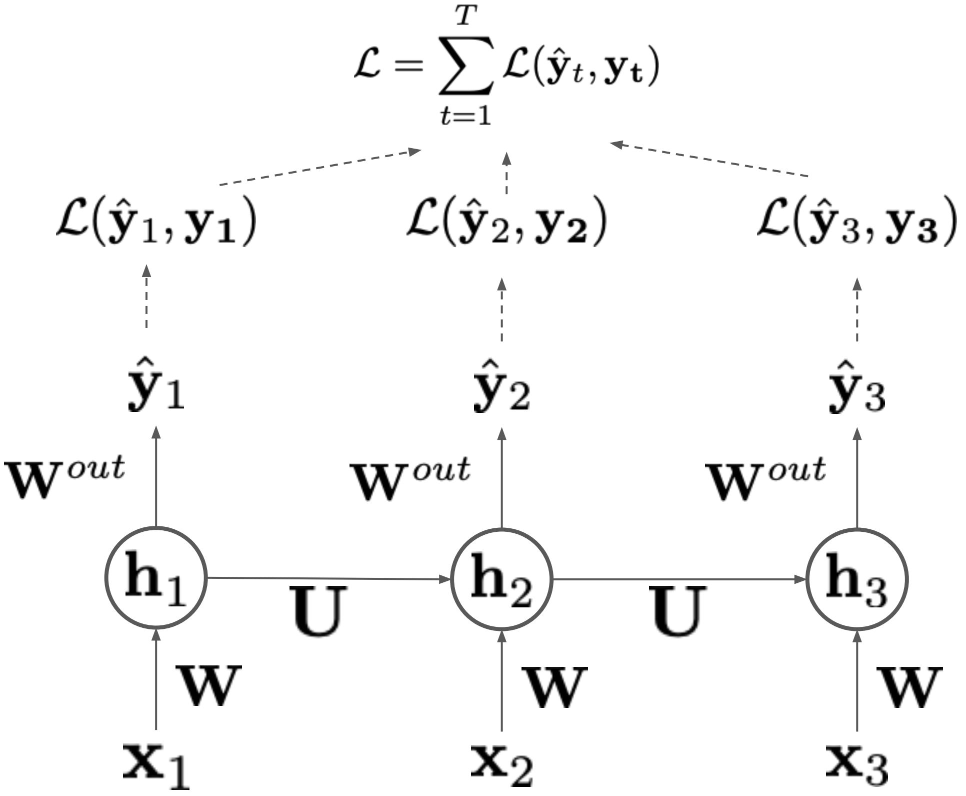
- RNN basics
- **Backpropagation through time**
- The vanishing/exploding gradients problem
- Advanced RNNs

Backpropagation in RNN

Forward propagation

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

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



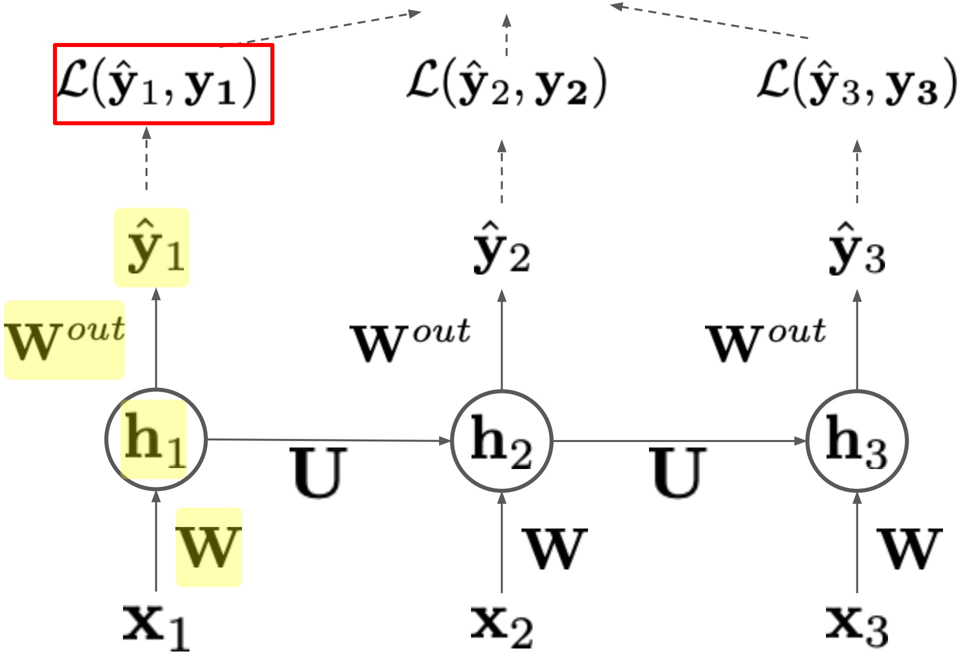
Backpropagation in RNN

Forward propagation

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

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

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}(\hat{\mathbf{y}}_t, \mathbf{y_t})$$



Before we derive gradients, let's think about how the gradient would flow

If we want to reduce $\mathcal{L}(\hat{\mathbf{y}}_1, \mathbf{y_1})$, which variables should we update?

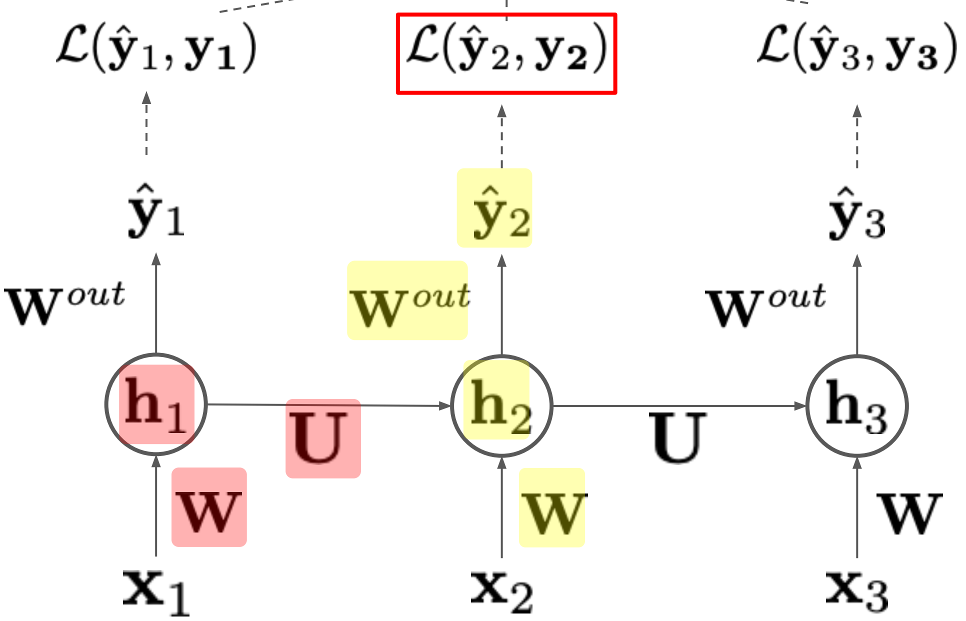
$$\hat{\mathbf{y}}_1, \mathbf{W}^{out}, \mathbf{h}_1, \mathbf{W}$$

Backpropagation in RNN

Forward propagation

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

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}(\hat{\mathbf{y}}_t, \mathbf{y_t})$$



Before we derive gradients, let's think about how the gradient would flow

If we want to reduce $\mathcal{L}(\hat{\mathbf{y}}_1, \mathbf{y_1})$, which variables should we update?

$$\hat{\mathbf{y}}_1, \mathbf{W}^{out}, \mathbf{h}_1, \mathbf{W}$$

If we want to reduce $\mathcal{L}(\hat{\mathbf{y}}_2, \mathbf{y_2})$, which variables should we update?

$$\hat{\mathbf{y}}_2, \mathbf{W}^{out}, \mathbf{h}_2, \mathbf{W}, \underbrace{\mathbf{U}, \mathbf{h}_1, \mathbf{W}}_{\text{These contribute to the value of } h_2}$$

Backpropagation in RNN

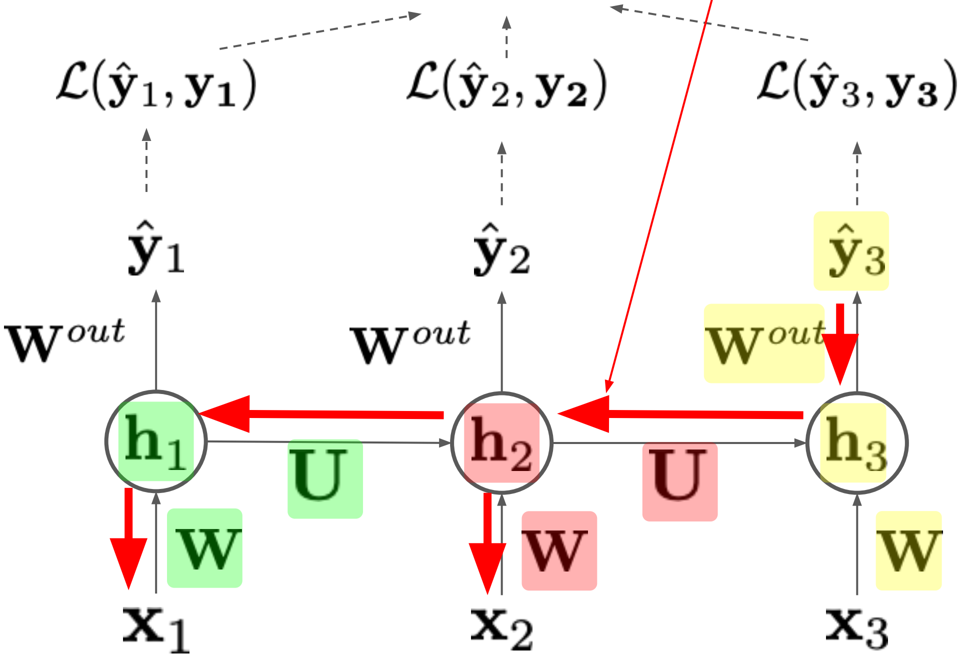
Forward propagation

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

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

Backpropagation through time!

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}(\hat{\mathbf{y}}_t, \mathbf{y_t})$$



Before we derive gradients, let's think about how the gradient would flow

If we want to reduce $\mathcal{L}(\hat{\mathbf{y}}_1, \mathbf{y_1})$, which variables should we update?

$$\hat{\mathbf{y}}_1, \mathbf{W}^{out}, \mathbf{h}_1, \mathbf{W}$$

If we want to reduce $\mathcal{L}(\hat{\mathbf{y}}_2, \mathbf{y_2})$, which variables should we update?

$$\hat{\mathbf{y}}_2, \mathbf{W}^{out}, \mathbf{h}_2, \mathbf{W}, \mathbf{U}, \mathbf{h}_1, \mathbf{W}$$

If we want to reduce $\mathcal{L}(\hat{\mathbf{y}}_3, \mathbf{y_3})$, which variables should we update?

$$\hat{\mathbf{y}}_3, \mathbf{W}^{out}, \mathbf{h}_3, \mathbf{W}, \mathbf{U}, \mathbf{h}_2, \mathbf{W}, \mathbf{U}, \mathbf{h}_1, \mathbf{W}$$

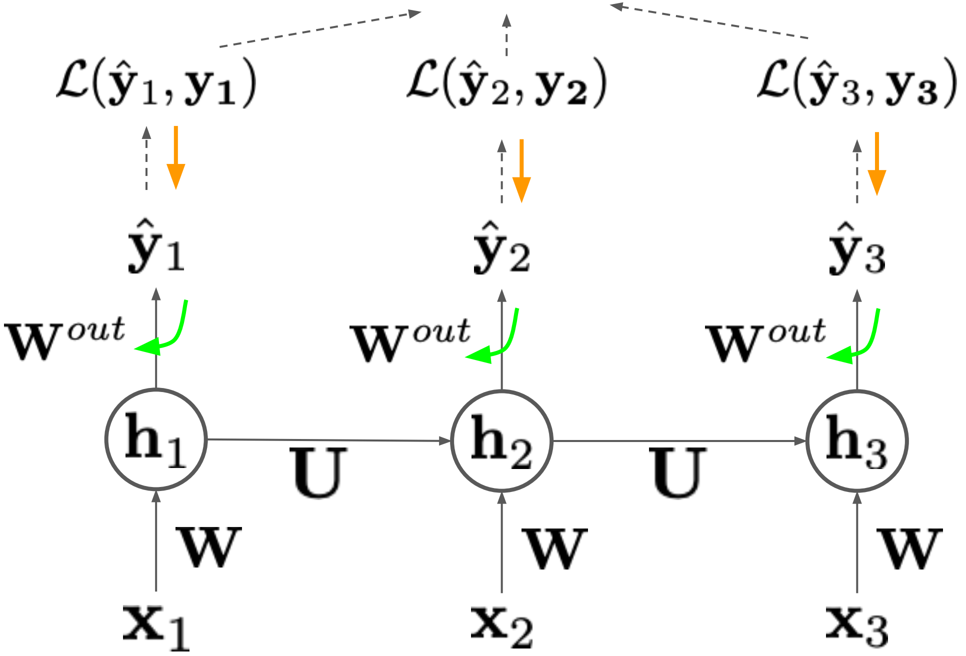
Backpropagation through time

Forward propagation

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

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

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}(\hat{\mathbf{y}}_t, \mathbf{y_t})$$



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

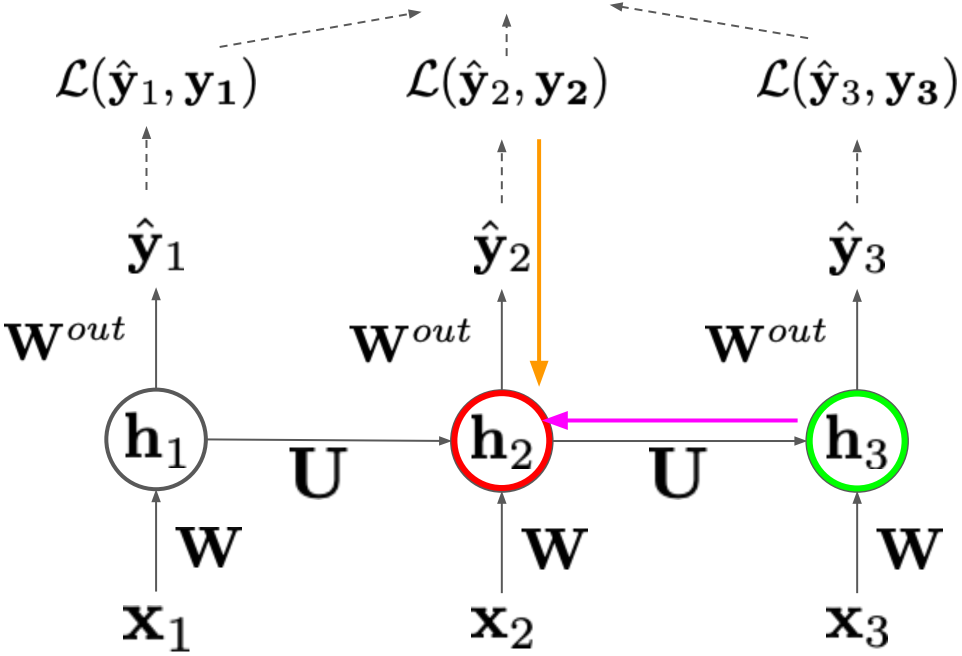
A (n,m)th element of $\mathbf{W}^{out}$

Backpropagation through time

Forward propagation

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

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}(\hat{\mathbf{y}}_t, \mathbf{y_t})$$



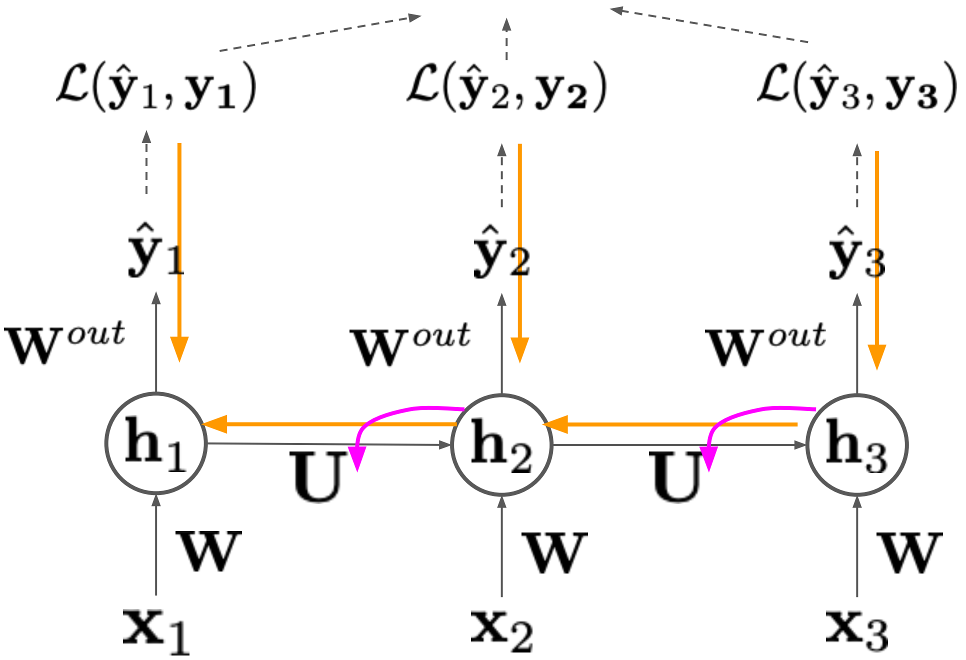
$$\text{Let } \mathcal{L}_t = \sum_{t'=t}^{T} \mathcal{L}(\hat{\mathbf{y}}_{t'}, \mathbf{y}_{t'}) \quad \text{(note: } \mathcal{L}_1 = \mathcal{L}\text{)}$$

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

It's applied recursively over temporal horizon

Backpropagation through time

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}(\hat{\mathbf{y}}_t, \mathbf{y_t})$$



Forward propagation

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

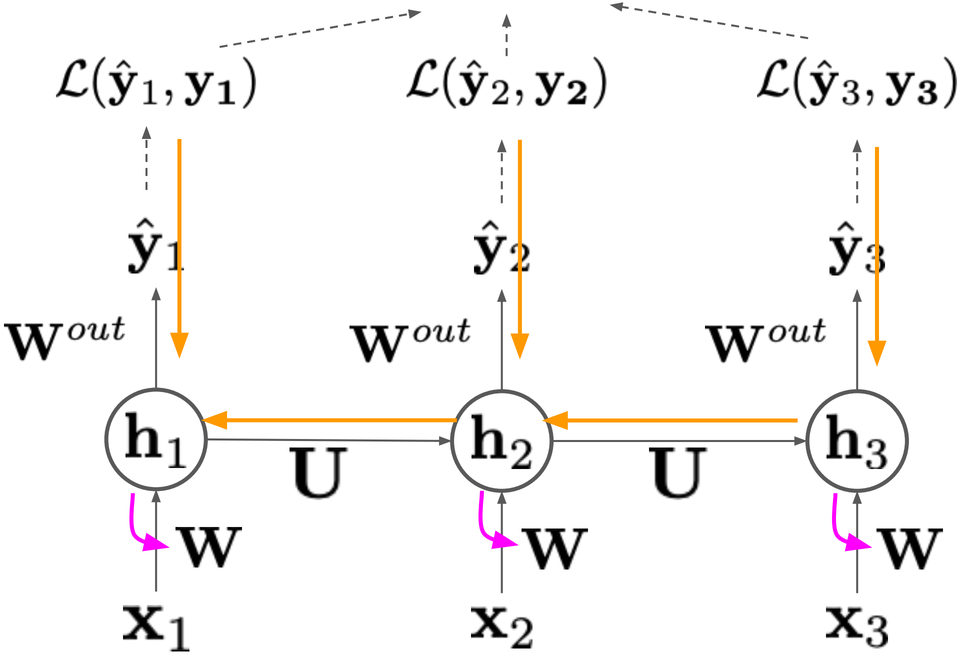
$$\hat{\mathbf{y}}_t = \mathbf{W}^{out}\mathbf{h}_t$$

$$\mathcal{L}_t = \sum_{t'=t}^{T} \mathcal{L}(\hat{\mathbf{y}}_{t'}, \mathbf{y}_{t'})$$

$$\frac{\partial \mathcal{L}}{\partial U_{m,m'}} = \sum_t \frac{\partial \mathcal{L}_t}{\partial h_{t,m}} \frac{\partial h_{t,m}}{\partial U_{m,m'}}$$

Backpropagation through time

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

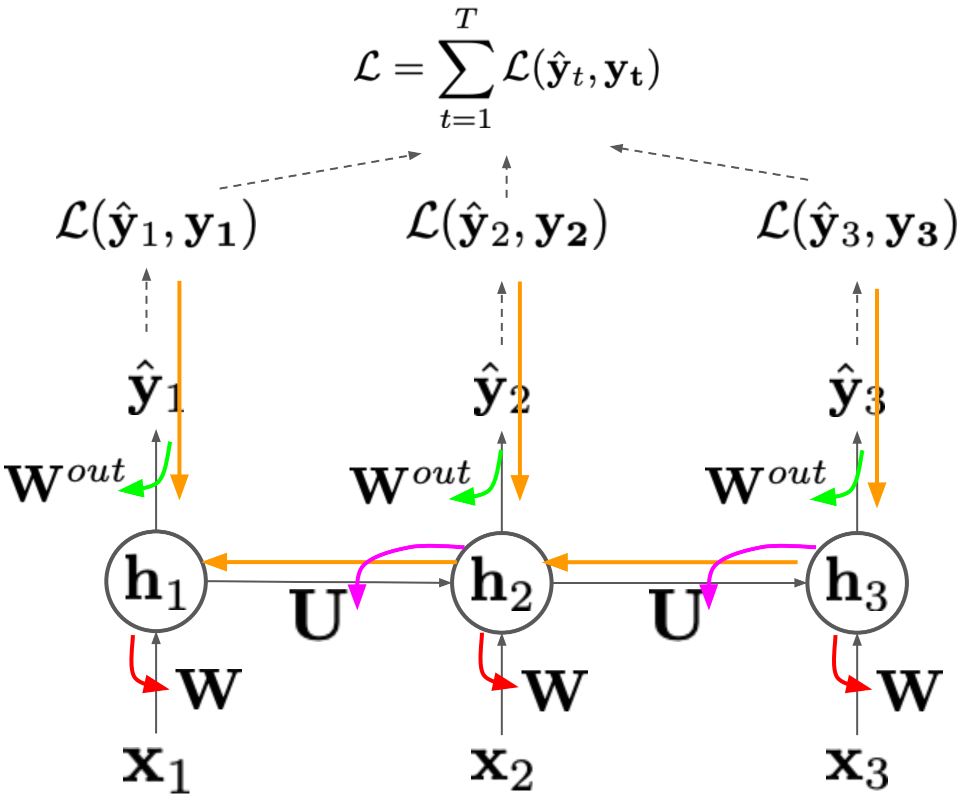


Forward propagation

$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}^{out}\mathbf{h}_t$$
$$\mathcal{L}_t = \sum_{t'=t}^{T} \mathcal{L}(\hat{\mathbf{y}}_{t'}, \mathbf{y}_{t'})$$

$$\frac{\partial \mathcal{L}}{\partial W_{m,d}} = \sum_t \frac{\partial \mathcal{L}_t}{\partial h_{t,m}} \frac{\partial h_{t,m}}{\partial W_{m,d}}$$

Backpropagation through time



Forward propagation

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

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

$$\mathcal{L}_t = \sum_{t'=t}^{T} \mathcal{L}(\hat{\mathbf{y}}_{t'}, \mathbf{y}_{t'})$$

Putting all together:

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

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

$$\frac{\partial \mathcal{L}}{\partial U_{m,m'}} = \sum_t \frac{\partial \mathcal{L}_t}{\partial h_{t,m}} \frac{\partial h_{t,m}}{\partial U_{m,m'}}$$

$$\frac{\partial \mathcal{L}}{\partial W_{m,d}} = \sum_t \frac{\partial \mathcal{L}_t}{\partial h_{t,m}} \frac{\partial h_{t,m}}{\partial W_{m,d}}$$

Today’s agenda

- RNN basics
- Backpropagation through time
- **The vanishing/exploding gradients problem**
- Advanced RNNs

What can go wrong in BPTT?

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

$$\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} = \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t}$$

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

Note:

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

What can go wrong in BPTT?

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

$$\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} = \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t}$$

Note:

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

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

Gradient of sigmoid causes vanishing!
It is always < 1, which can be multiplied over time to very small value!

What can go wrong in BPTT?

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

$$\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} = \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t}$$

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

Techniques to prevent vanishing/exploding gradients

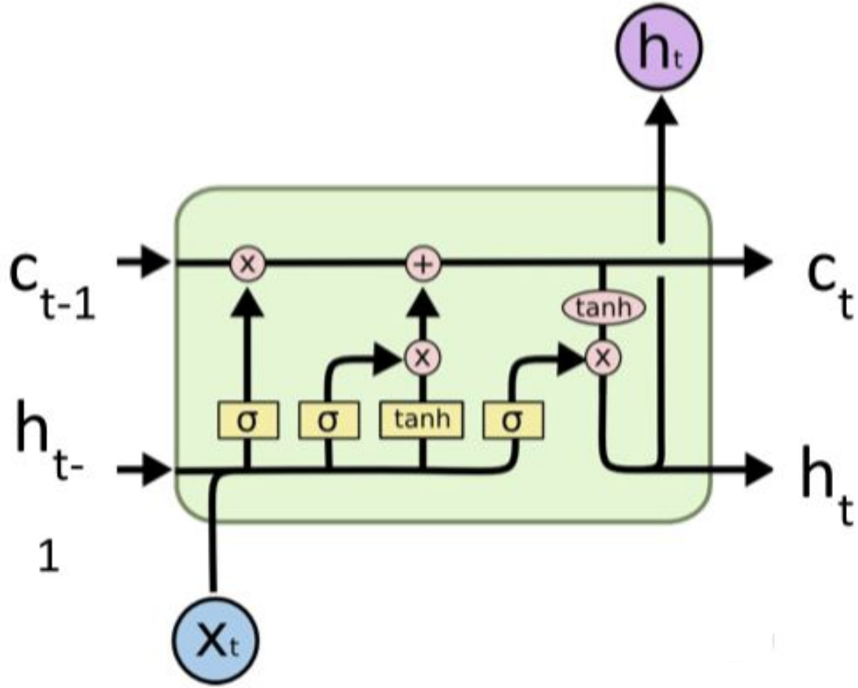
- Initializing weights **U** to be orthogonal
- Clip gradients if the value is too large

Today’s agenda

- RNN basics
- Backpropagation through time
- The vanishing/exploding gradients problem
- **Advanced RNNs**

RNNs addressing vanishing/exploding gradients

Long Short-Term Memory (LSTM)

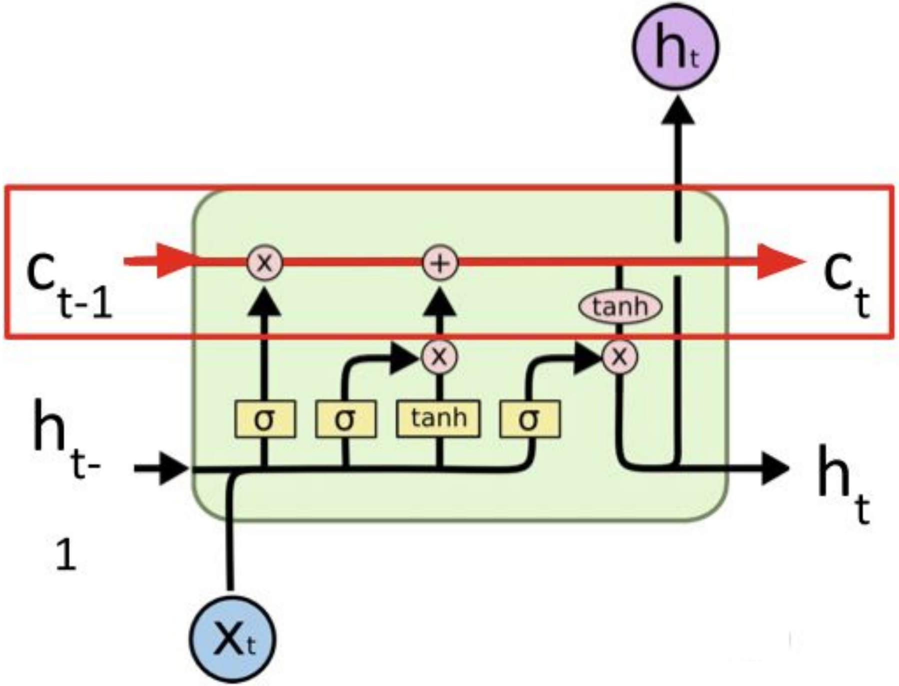


- A recurrent neural network variety designed to retain long-term dependencies.
- Helps dealing with both the vanishing and exploding gradient problem

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

RNNs addressing vanishing/exploding gradients

Long Short-Term Memory (LSTM)

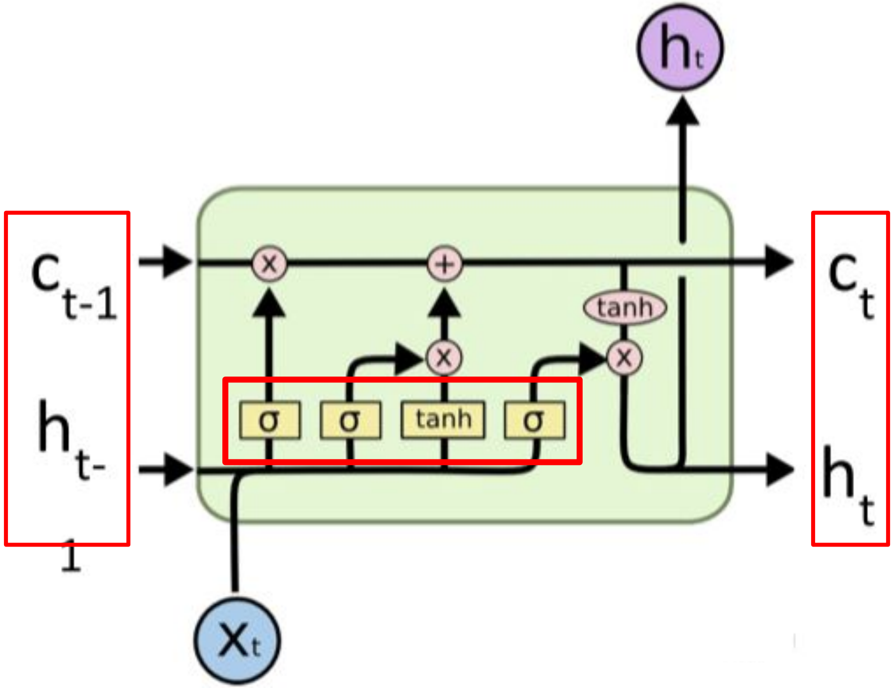


- A recurrent neural network variety designed to retain long-term dependencies.
- Helps dealing with both the vanishing and exploding gradient problem
- The key idea is an additive connection of previous memories passed through time

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

Long Short-Term Memory (LSTM)

- Overview

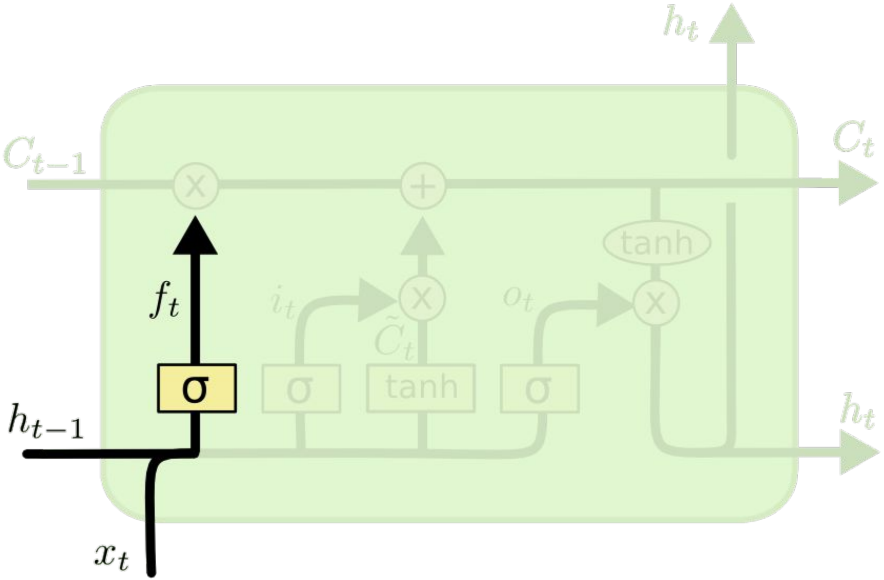


- Information is passed through two variables
- There are **four switch variables** that determines how the information flows through time

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

Long Short-Term Memory (LSTM)

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

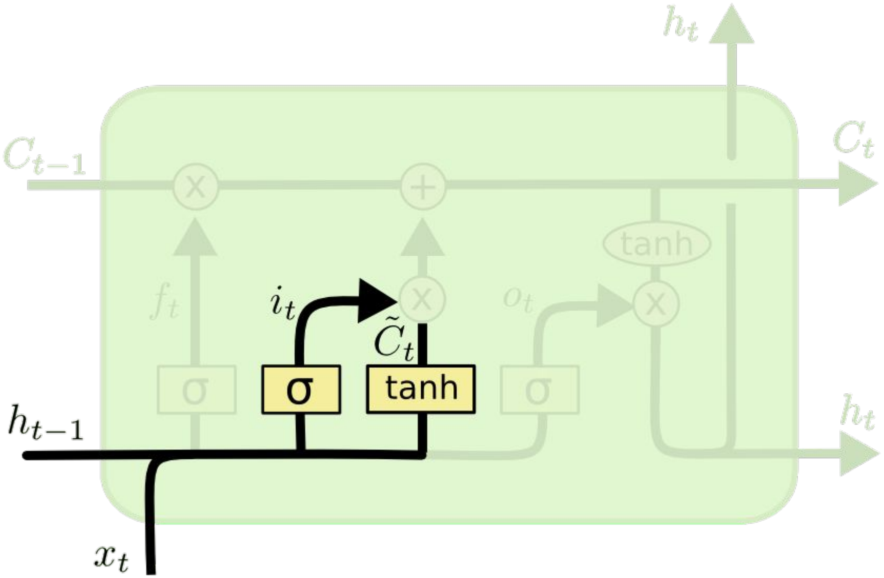


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

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

Long Short-Term Memory (LSTM)

- The input gate behaves similar to forget gates with new inputs



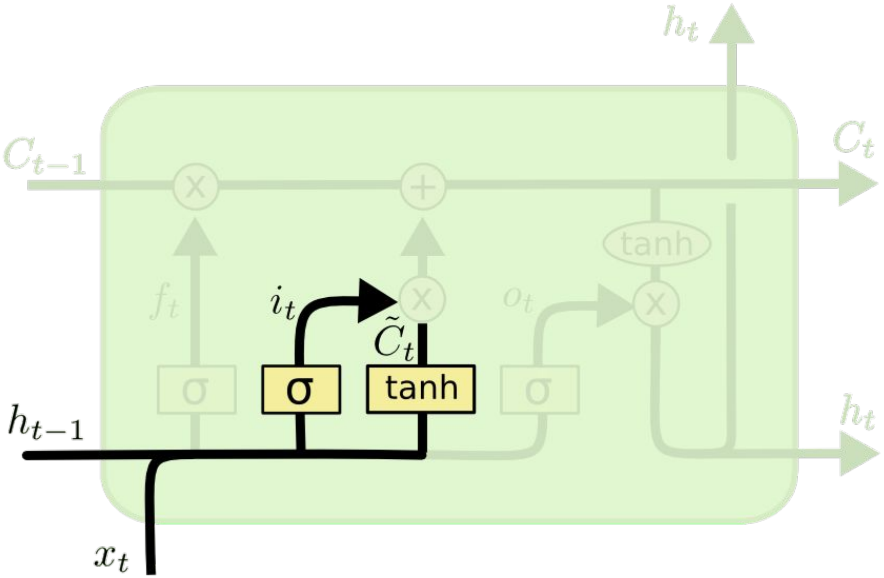
$$\boxed{i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)}$$

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

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

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.



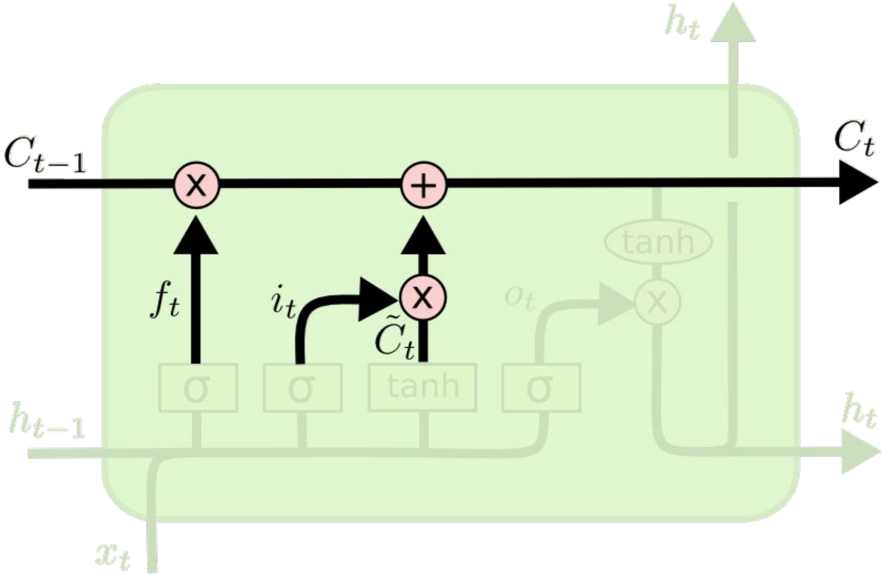
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

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

Long Short-Term Memory (LSTM)

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

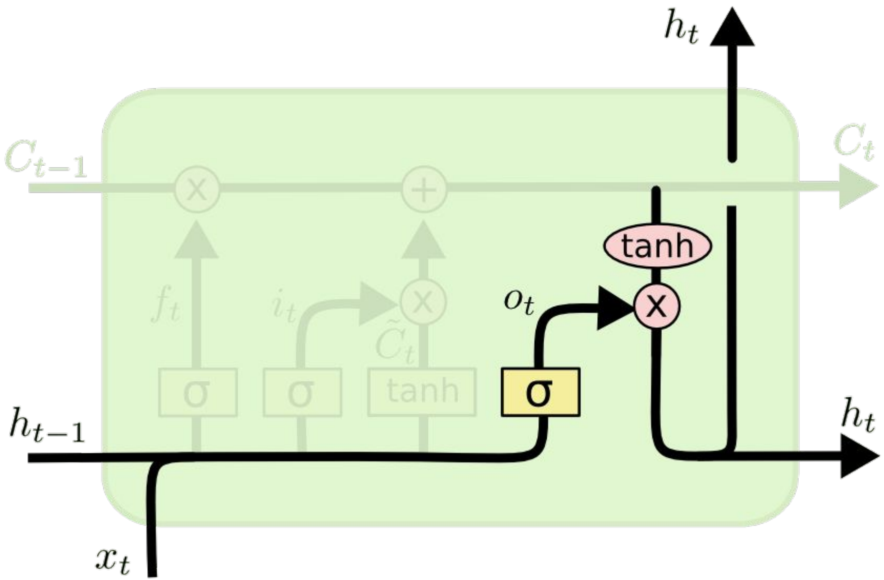


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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

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.



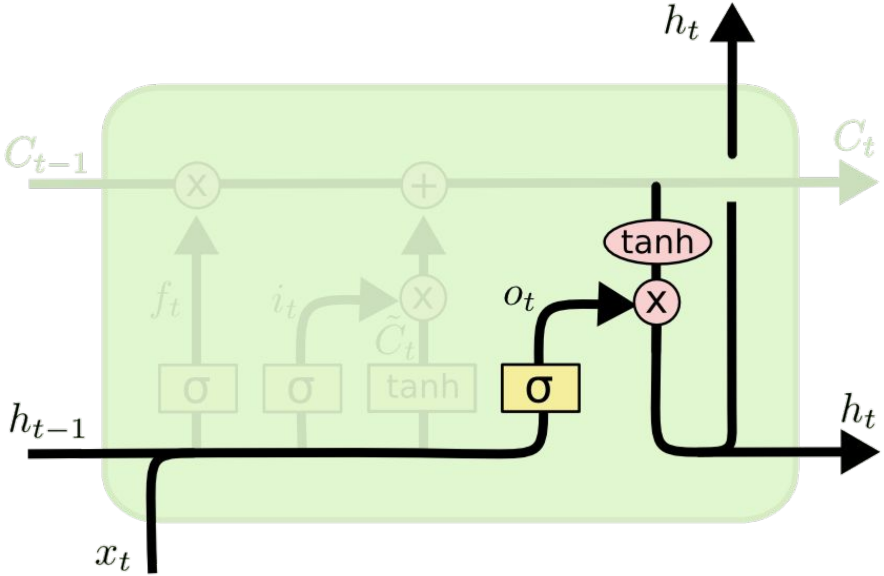
$$\boxed{o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)}$$

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

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

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.

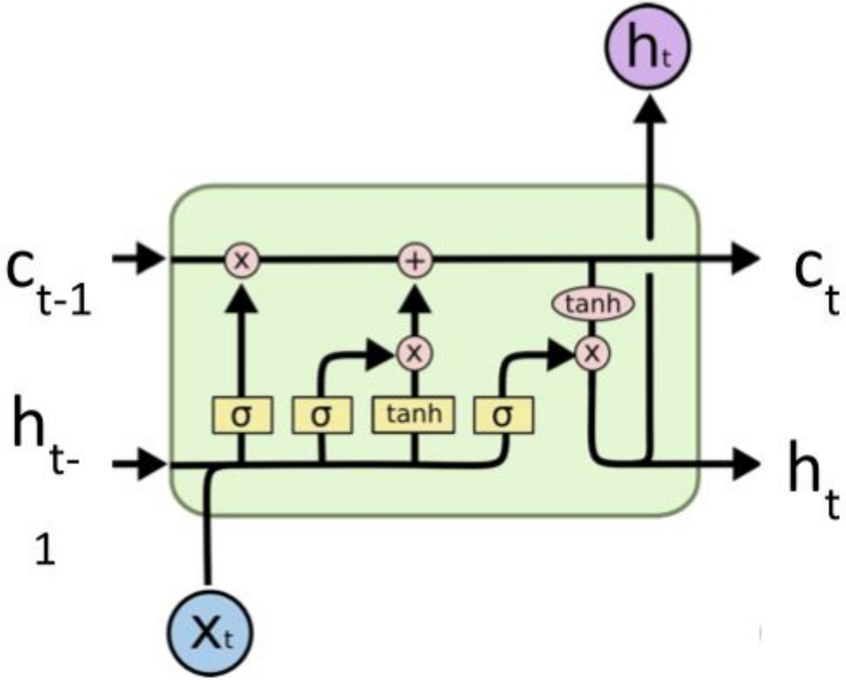


$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

$$\boxed{h_t = o_t * \tanh\left(C_t\right)}$$

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

Long Short-Term Memory (LSTM)



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

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

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

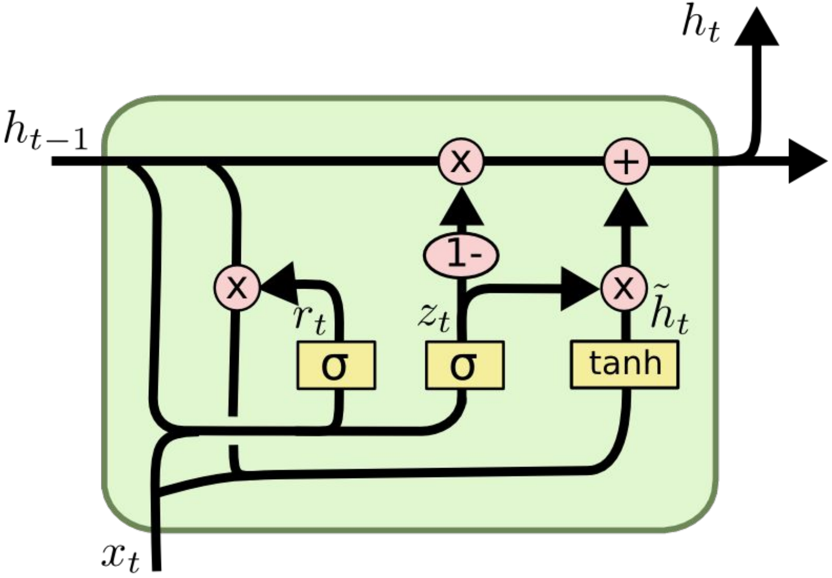
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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

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

Gated Recurrent Unit (GRU)

- A simplified variation of LSTM



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

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

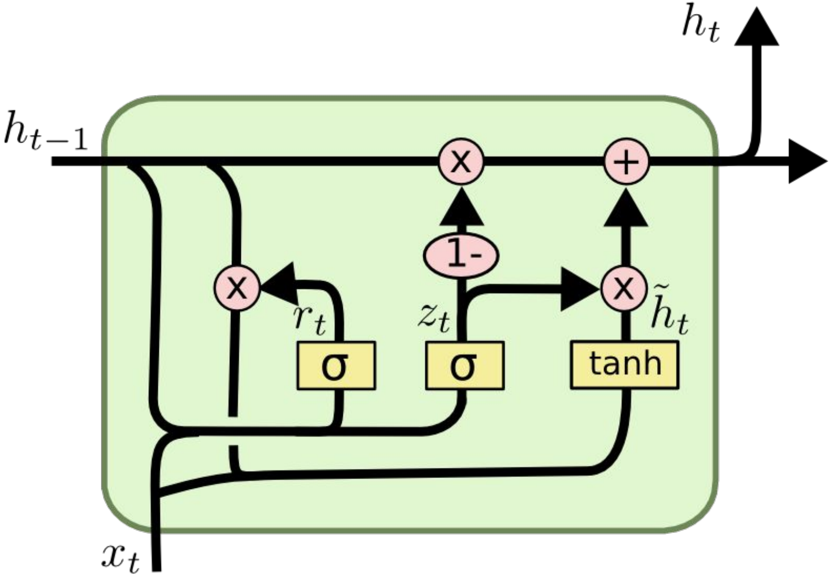
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

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

Gated Recurrent Unit (GRU)

- A simplified variation of LSTM
- The forget, input and output gates are simplified into a single gate



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

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

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

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

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

Next

- RNNs for sequence modeling
    - Language model