

Sequence Modeling

Recurrent Neural Network

EE6132: Deep learning for Image Processing


Sequence-to-sequence learning

- Map a sequence of vectors in one domain to another sequence of vectors in some other domain
 - Input sequence $\{x_1, x_2, \dots, x_t, \dots, x_{n-1}, x_n\}$
 - Output sequence $\{y_1, y_2, \dots, y_t, \dots, y_{m-1}, y_m\}$

Can you name some sequence-to-sequence learning problems?

Sequence-to-sequence problems

- **Language translation**

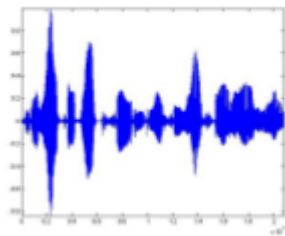
Hello, how are you?  नमस्ते आप कैसे हैं?

Sequence-to-sequence problems

- **Language translation**

Hello, how are you?  नमस्ते आप कैसे हैं?

- **Speech signal to text**



Hello, how are you?

Sequence-to-sequence problems

- Image to text (OCR)



"Text RECOGNITION"

Sequence-to-sequence problems

- **Dialogue / Question Answering**
 - **Input:** Are you free tomorrow?
 - **Output:** Yes, what's up?

Sequence-to-sequence problems

- **Dialogue / Question Answering**

- **Input:** Are you free tomorrow?
- **Output:** Yes, what's up?

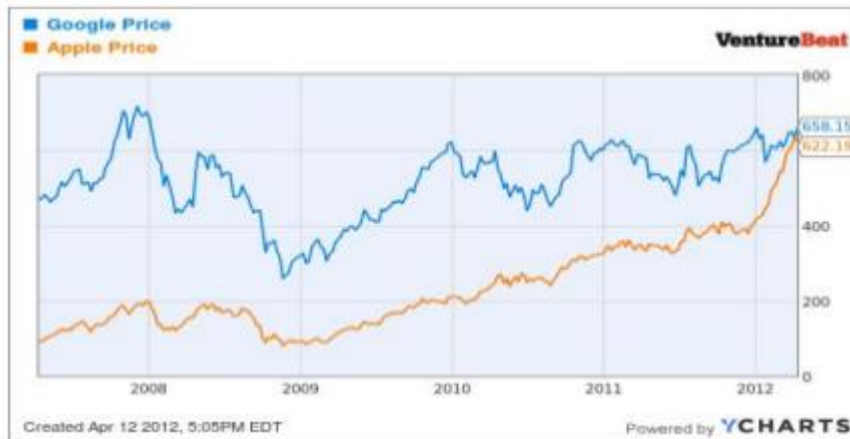
- **Image captioning**



A bird flying over a body of water

Sequence modeling problem

- **Time series data: Given a sequence, predict the next element**
 - Stock prices
 - Weather prediction

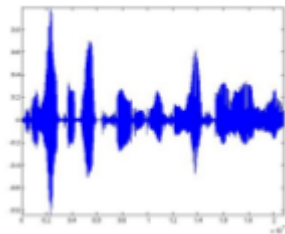


Sequence learning

- **Language translation**

Hello, how are you?  नमस्ते आप कैसे हैं?

- **Speech signal to text**



Hello, how are you?

Can you design an architecture for these using MLPs or CNNs?

Some points to note

- How to represent text in a Neural Network?

Some points to note

- How to represent text in a Neural Network?
- Represent each word as a one-hot vector with size same as its vocabulary size

Some points to note

- How to represent text in a Neural Network?
- Represent each word as a one-hot vector with size same as its vocabulary size
 - Issues with that method?


Some points to note

- How to represent text in a Neural Network?
- Represent each word as a one-hot vector with size same as its vocabulary size
 - Issues with that method?
 - Memory required will increase as order of $\Theta(n^2)$ with the vocabulary size
 - This is not so good given that most of the elements are 0's

Some points to note

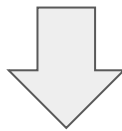
- How to represent text in a Neural Network?
- Represent each word as a one-hot vector with size same as its vocabulary size
 - Issues with that method?
 - Memory required will increase as order of $\Theta(n^2)$ with the vocabulary size
 - This is not so good given that most of the elements are 0's
- Word embeddings - represent every word as a vector of real numbers (fixed size)
 - Word2vec
 - GloVe

Sequence learning

- **Language translation** Hello, how are you?  नमस्ते आप कैसे हैं?

Can you design an architecture for this using MLPs or CNNs?

$$\{x_1, x_2, \dots, x_t, \dots, x_{n-1}, x_n\}$$



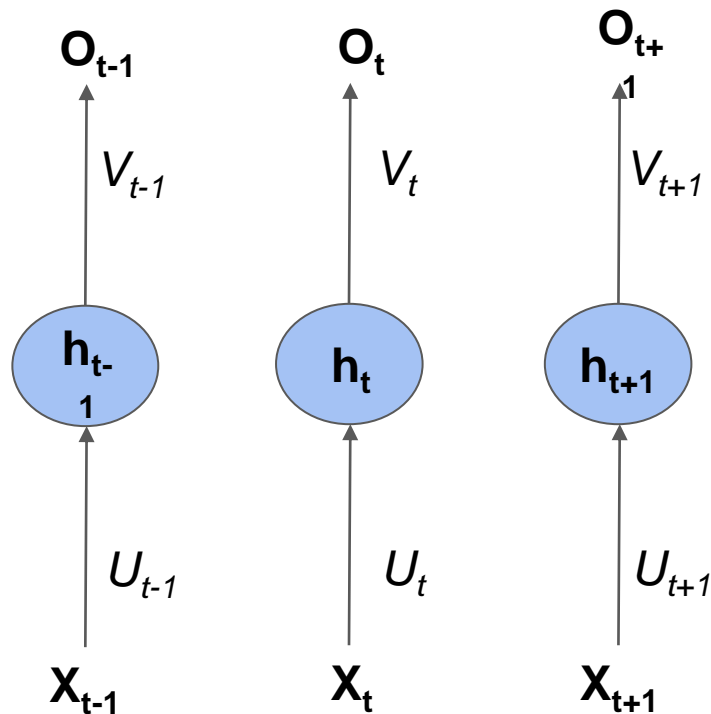
$$\{y_1, y_2, \dots, y_t, \dots, y_{m-1}, y_m\}$$

A naive approach

- Use multiple MLPs, one for each time-step

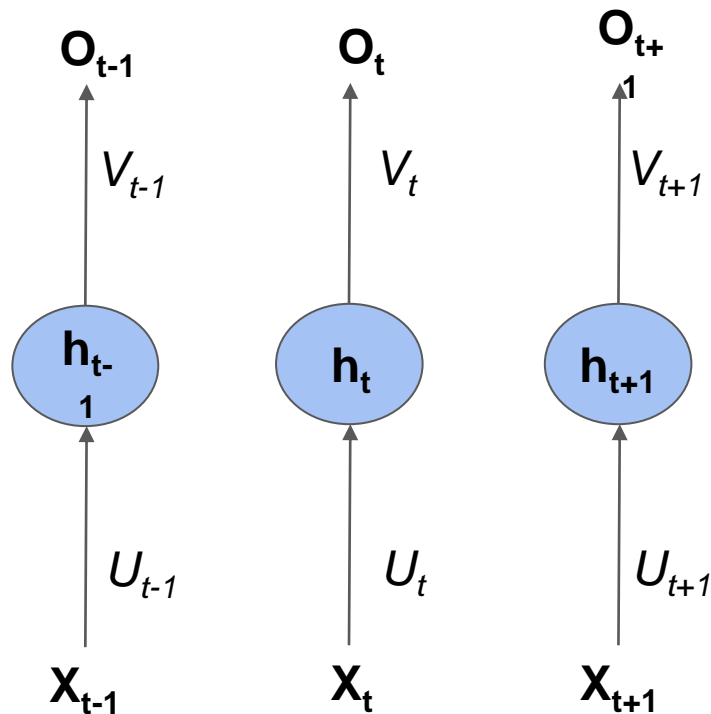
A naive approach

- Use multiple MLPs, one for each time-step



A naive approach

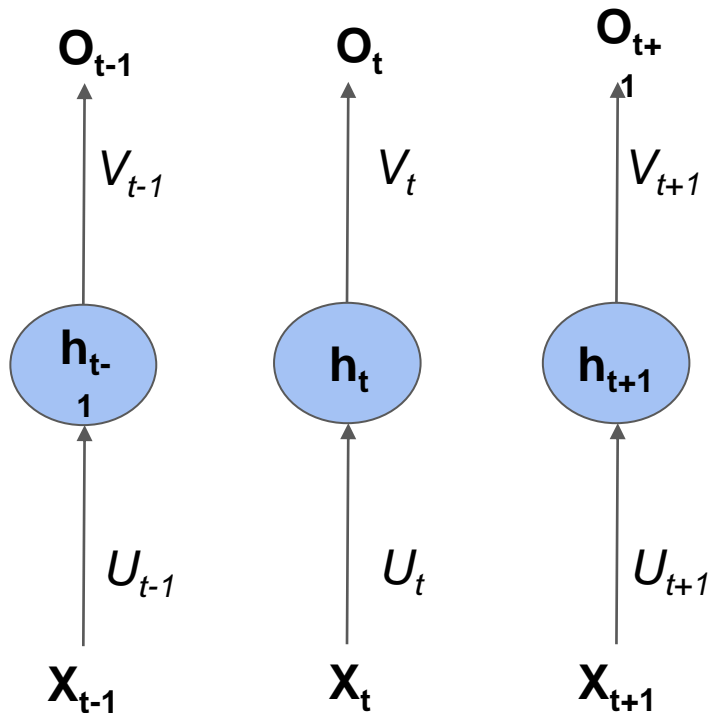
- Use multiple MLPs, one for each time-step



- Handles sequence?

A naive approach

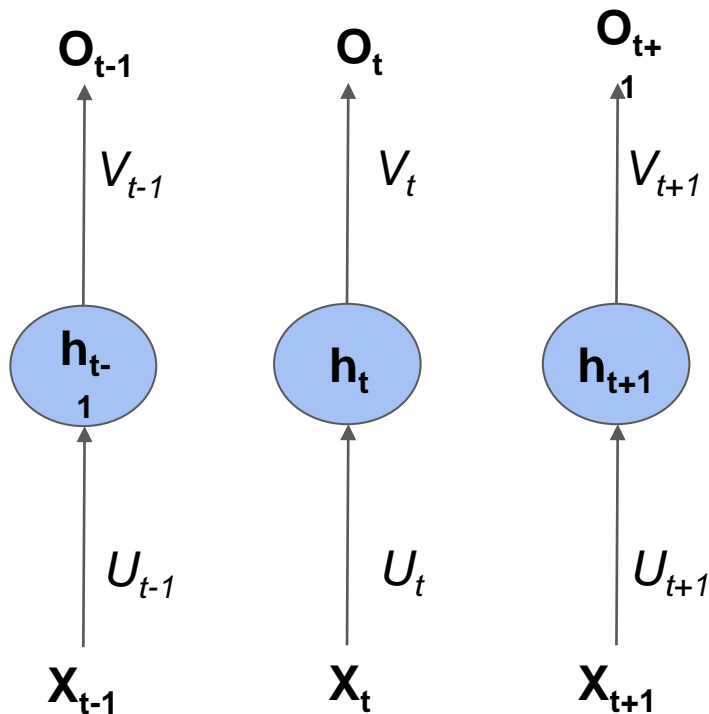
- Use multiple MLPs, one for each time-step



- Handles sequence? Yes ✓

A naive approach

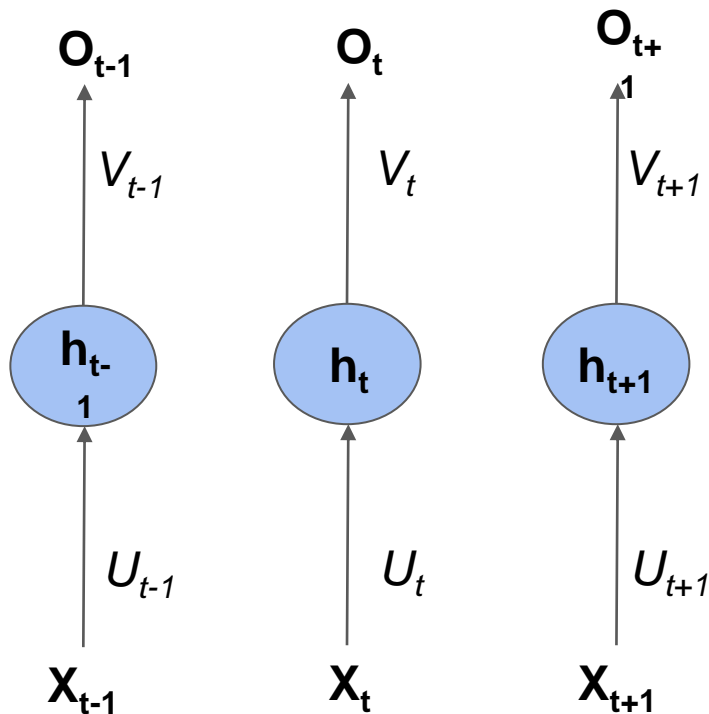
- Use multiple MLPs, one for each time-step



- Handles sequence? Yes ✓
- Models dependency?

A naive approach

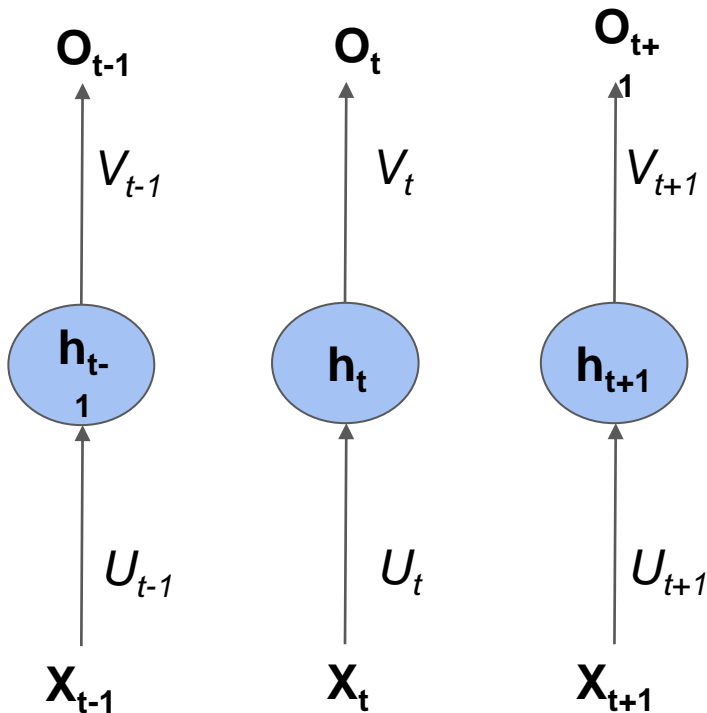
- Use multiple MLPs, one for each time-step



- Handles sequence? Yes ✓
- Models dependency? No ✗

A naive approach

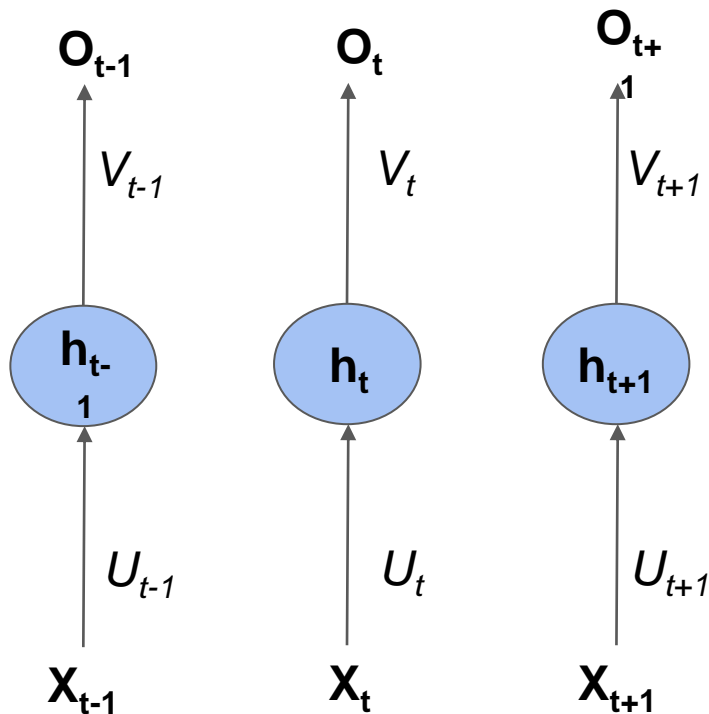
- Use multiple MLPs, one for each time-step



- Handles sequence? Yes ✓
- Models dependency? No ✗
- Variable length sequences?

A naive approach

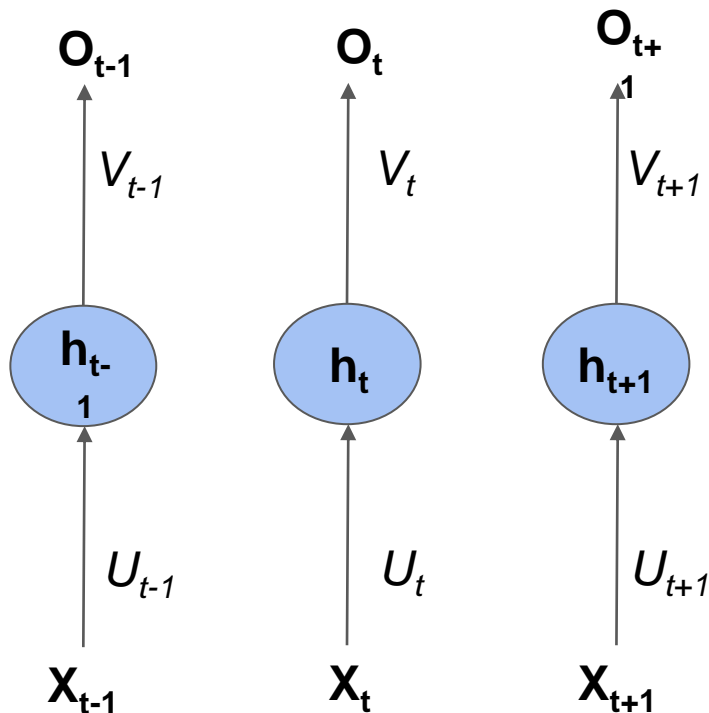
- Use multiple MLPs, one for each time-step



- Handles sequence? Yes ✓
- Models dependency? No ✗
- Variable length sequences? No ✗

A naive approach

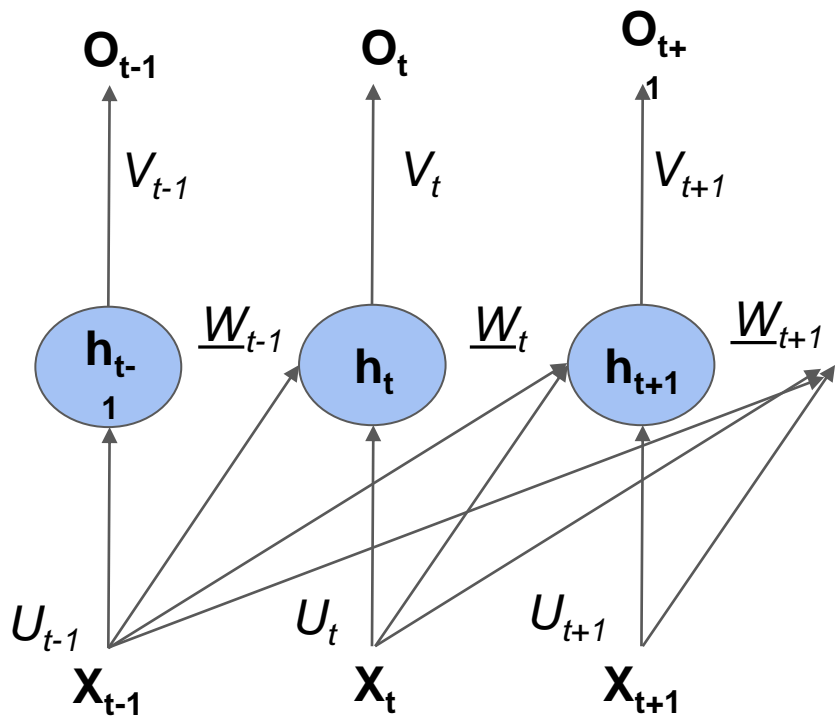
- Use multiple MLPs, one for each time-step



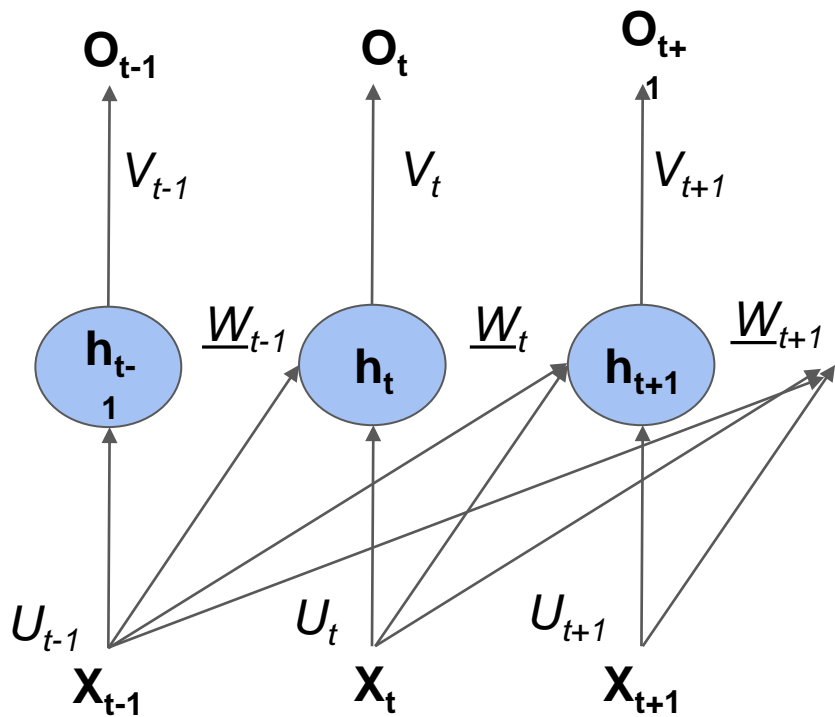
- Handles sequence? Yes ✓
- Models dependency? No ✗
- Variable length sequences? No ✗
- Many parameters to train :(✗

A revised architecture

$$h_t = f(\underline{U}_t x_t + \underline{W}_{t-1} x_{t-1} + \dots)$$
$$o_t = g(\underline{V}_t h_t)$$

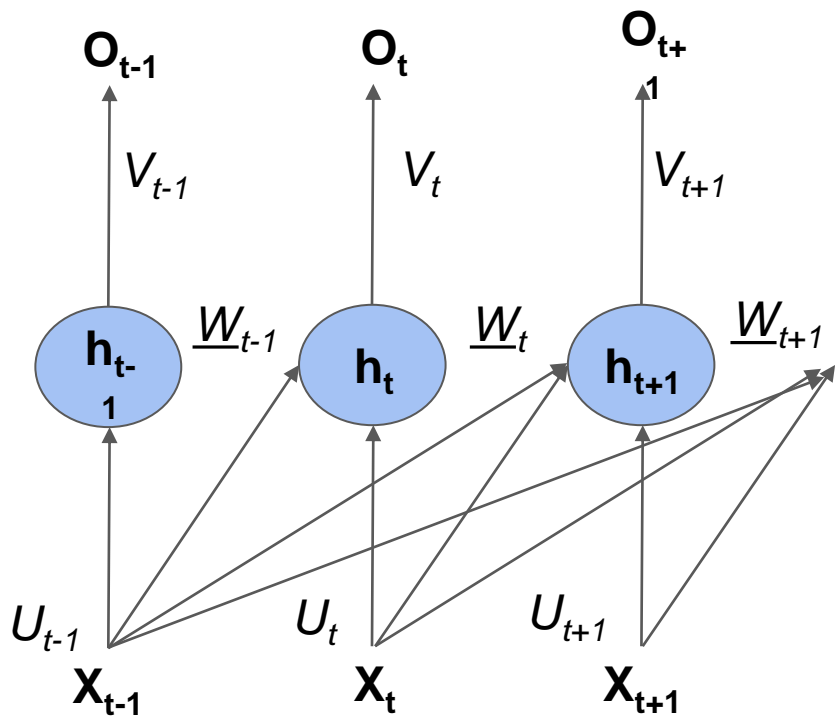


A revised architecture



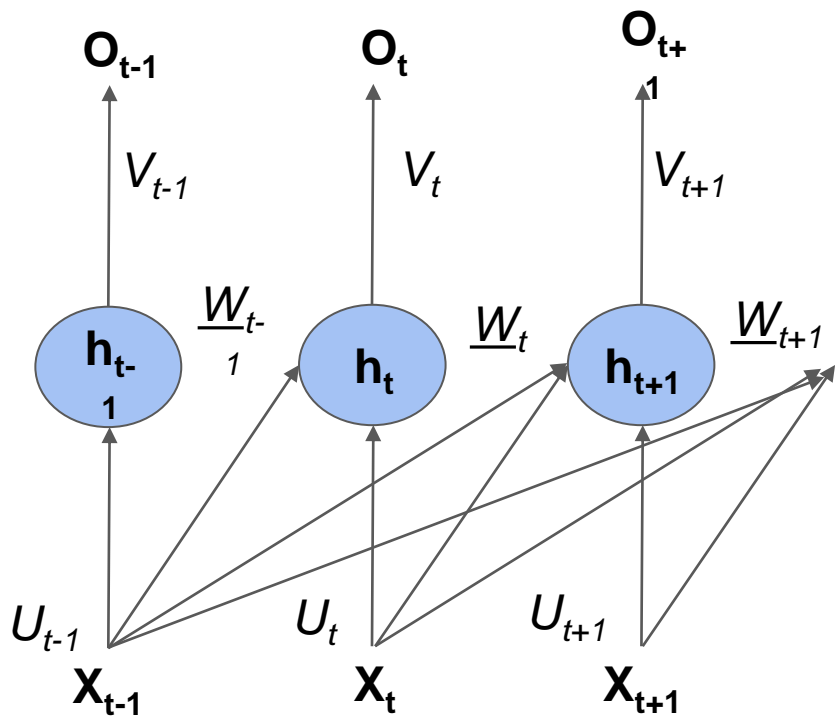
- Handles sequence?

A revised architecture



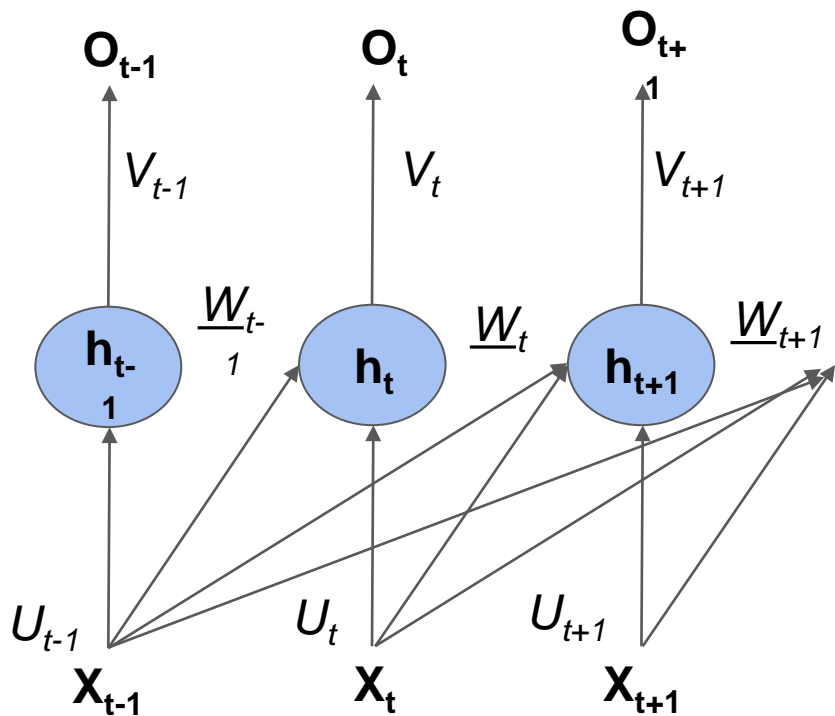
- Handles sequence? Yes ✓

A revised architecture



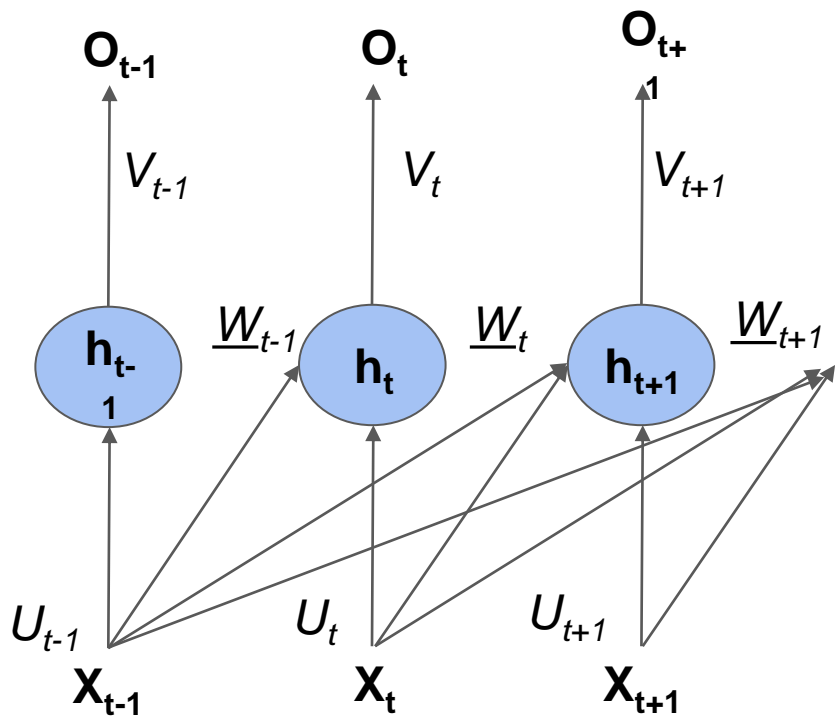
- Handles sequence? Yes ✓
- Models dependency?

A revised architecture



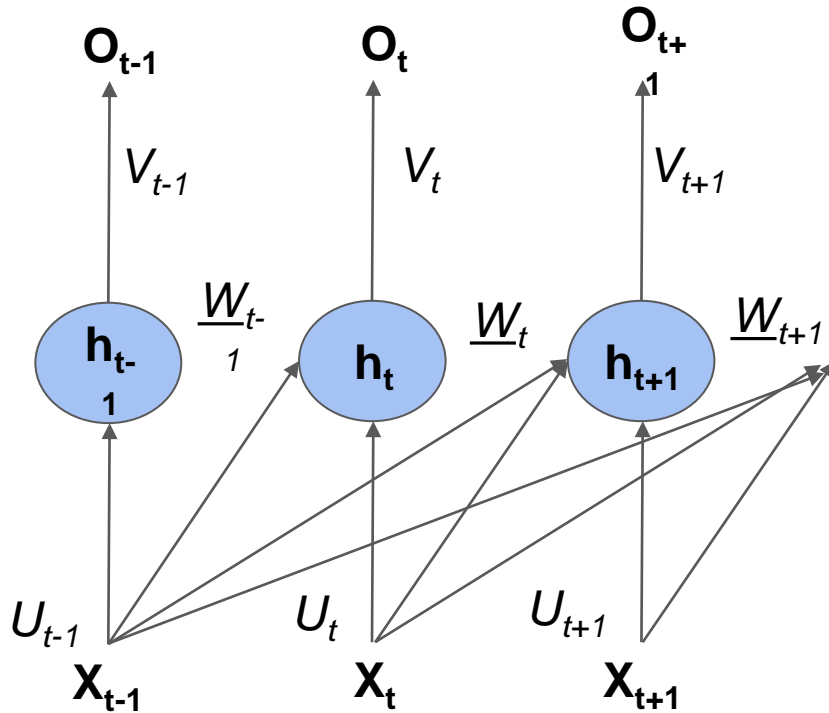
- Handles sequence? Yes ✓
- Models dependency? Yes ✓

A revised architecture



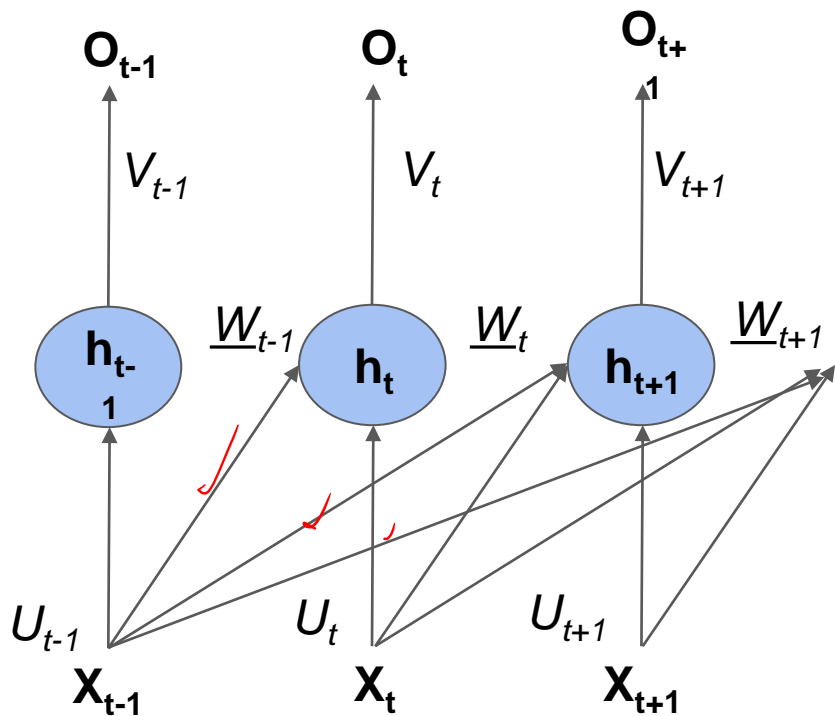
- Handles sequence? Yes ✓
- Models dependency? Yes ✓
- Variable length sequences?

A revised architecture



- Handles sequence? Yes ✓
- Models dependency? Yes ✓
- Variable length sequences? No ✗

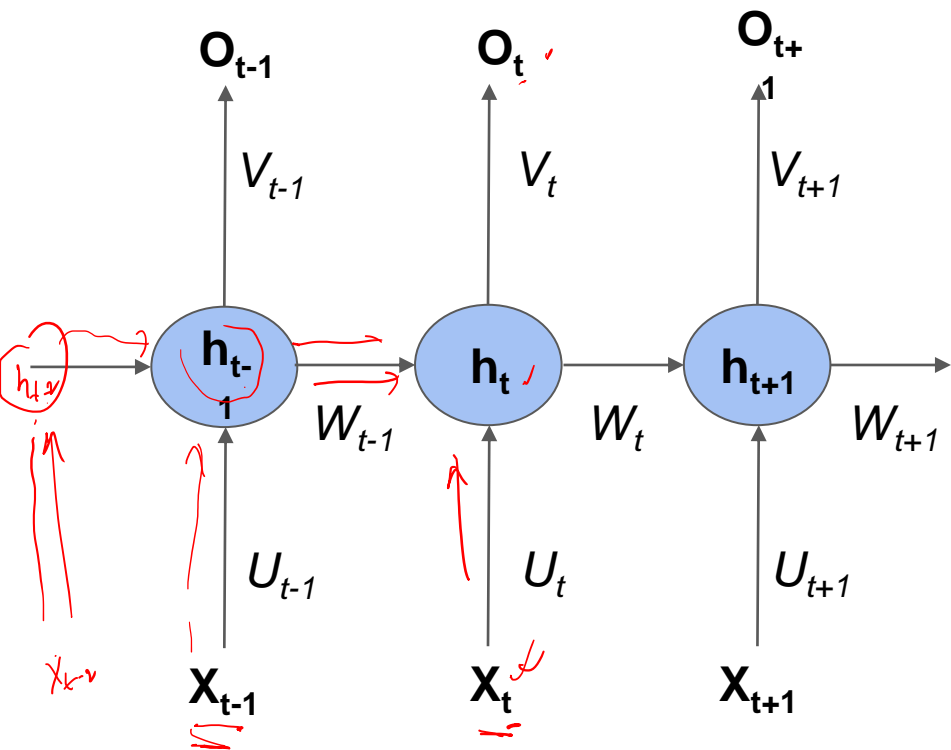
A revised architecture



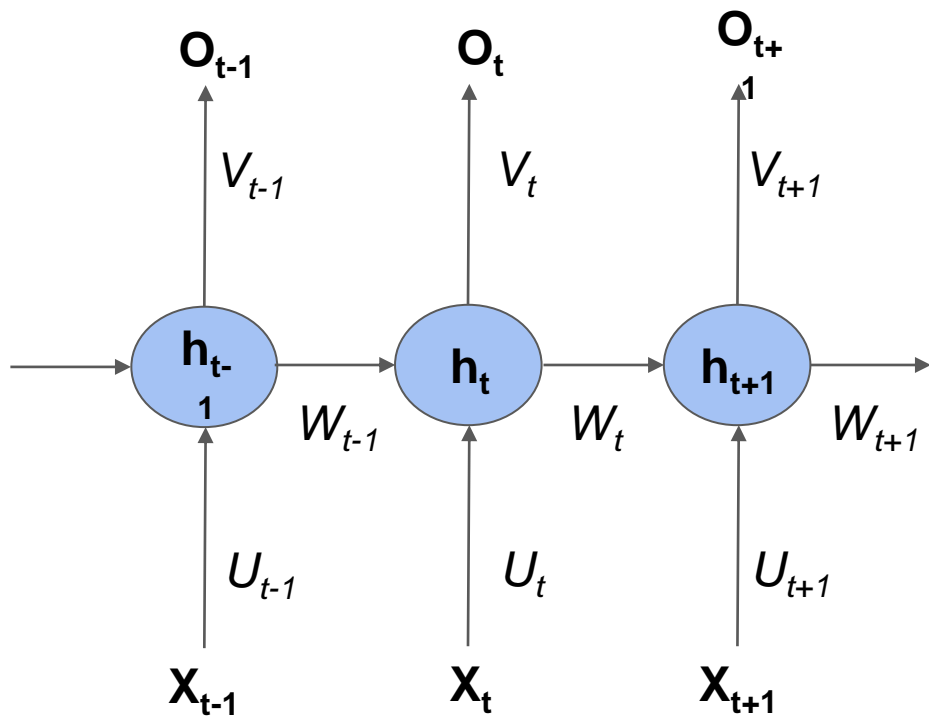
- Handles sequence? Yes ✓
- Models dependency? Yes ✓
- Variable length sequences? No ✗
- Too many parameters to train :(✗

A revised architecture

$$h_t = f(U_t x_t + W_{t-1} h_{t-1})$$
$$o_t = g(V_t h_t)$$

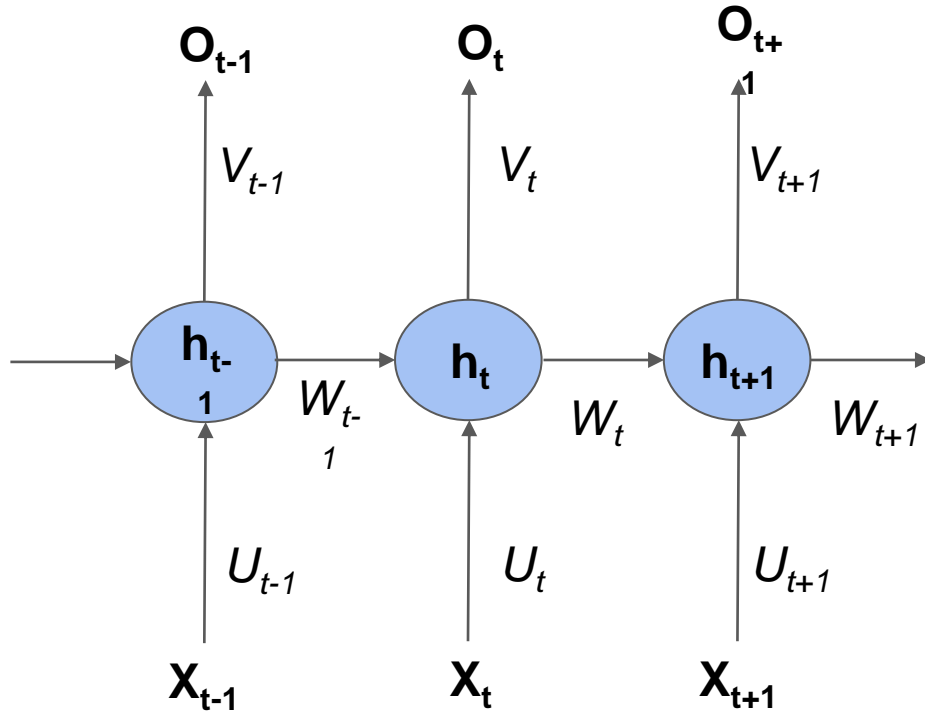


A revised architecture



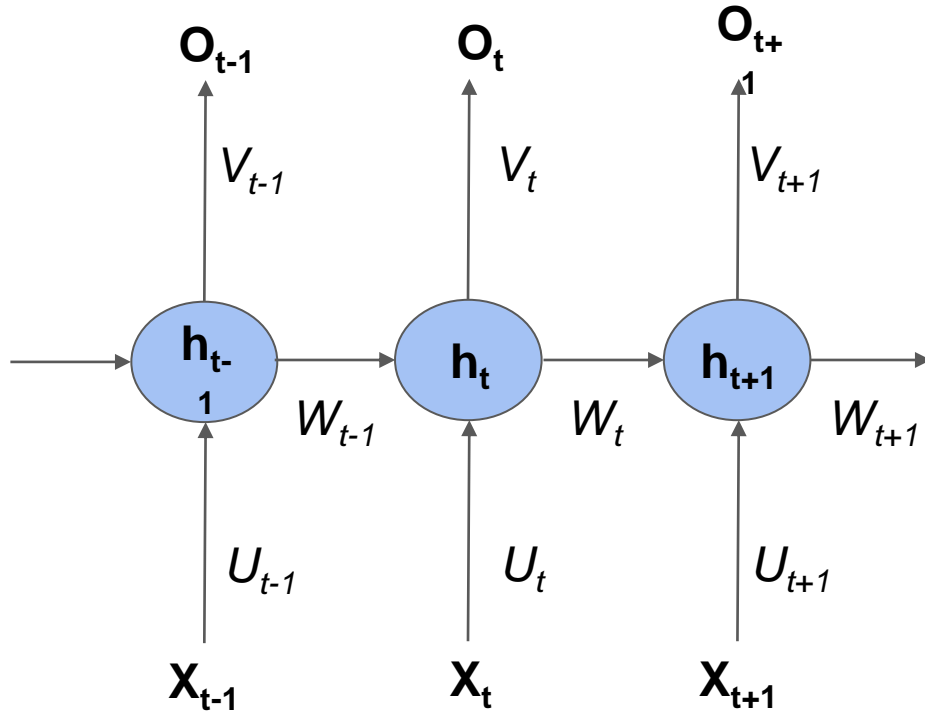
- Handles sequence?

A revised architecture



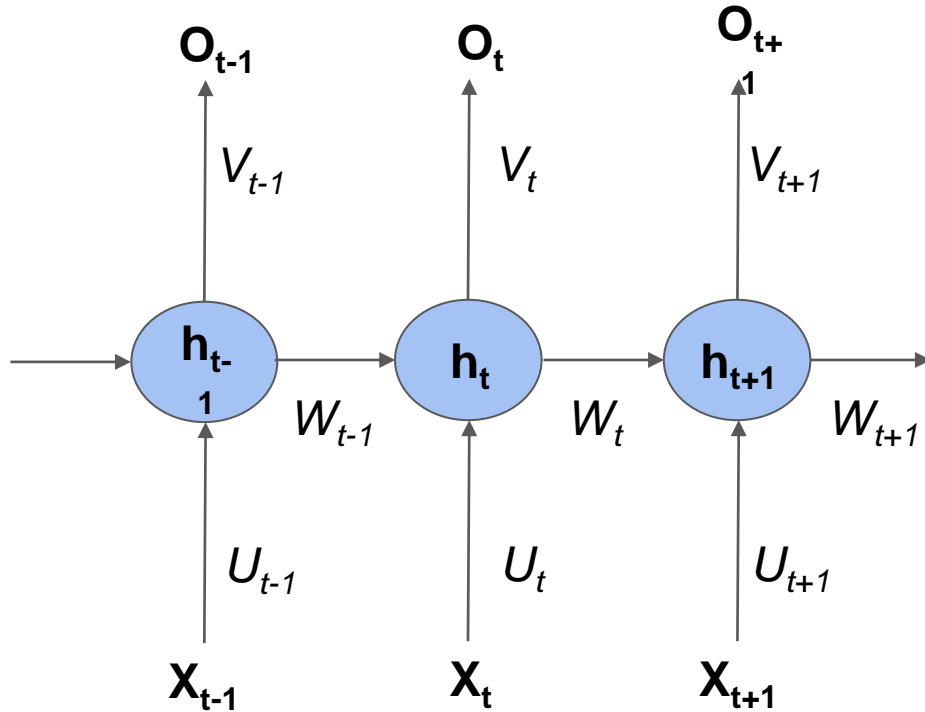
- Handles sequence? Yes ✓

A revised architecture



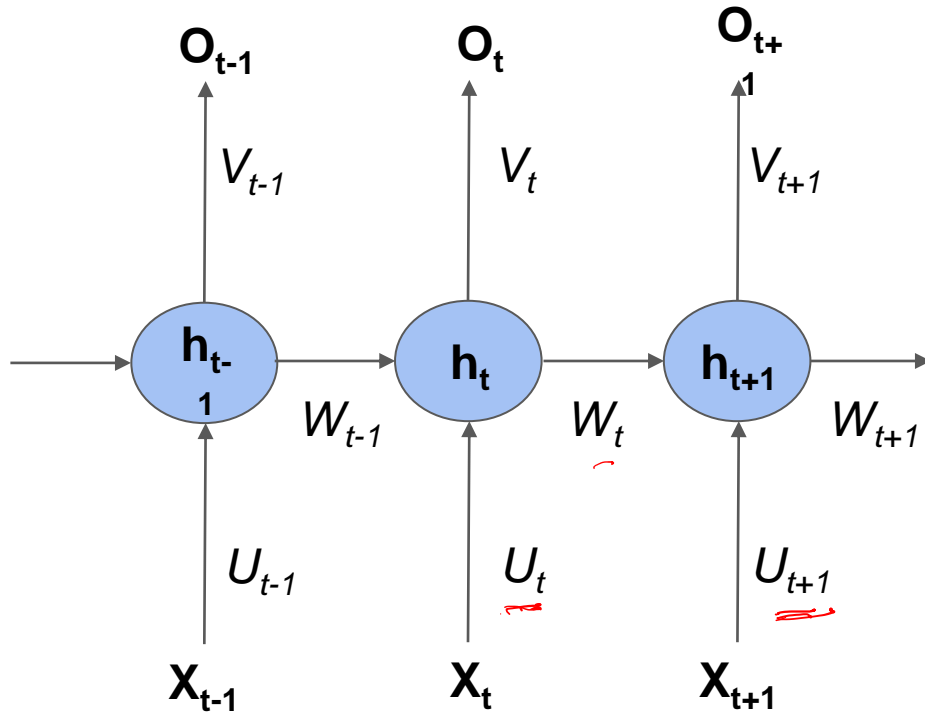
- Handles sequence? Yes ✓
- Models dependency?

A revised architecture



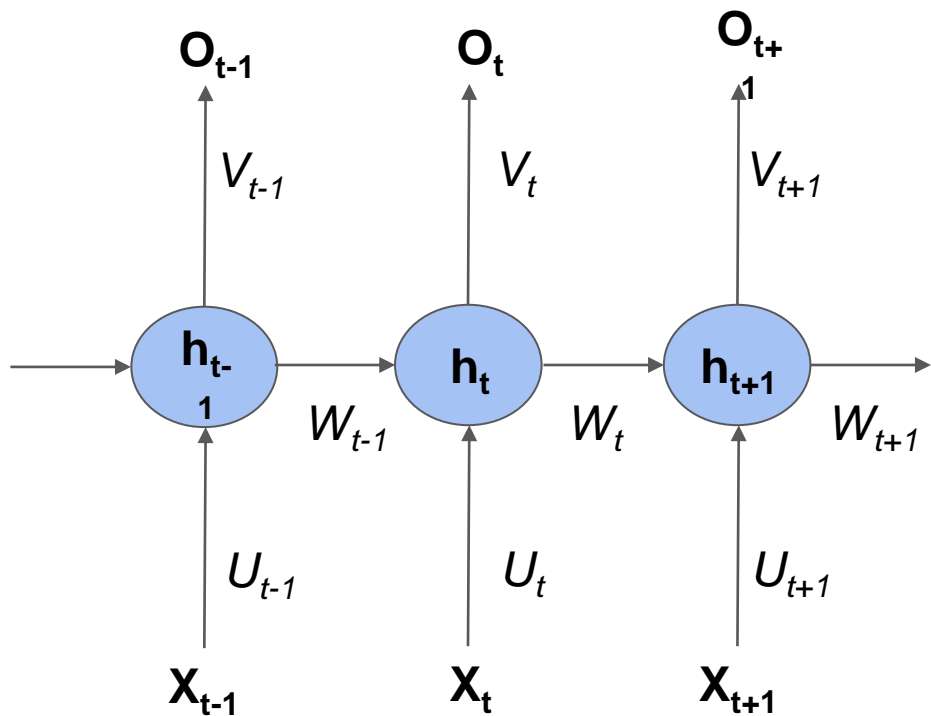
- Handles sequence? Yes ✓
- Models dependency? Yes ✓

A revised architecture



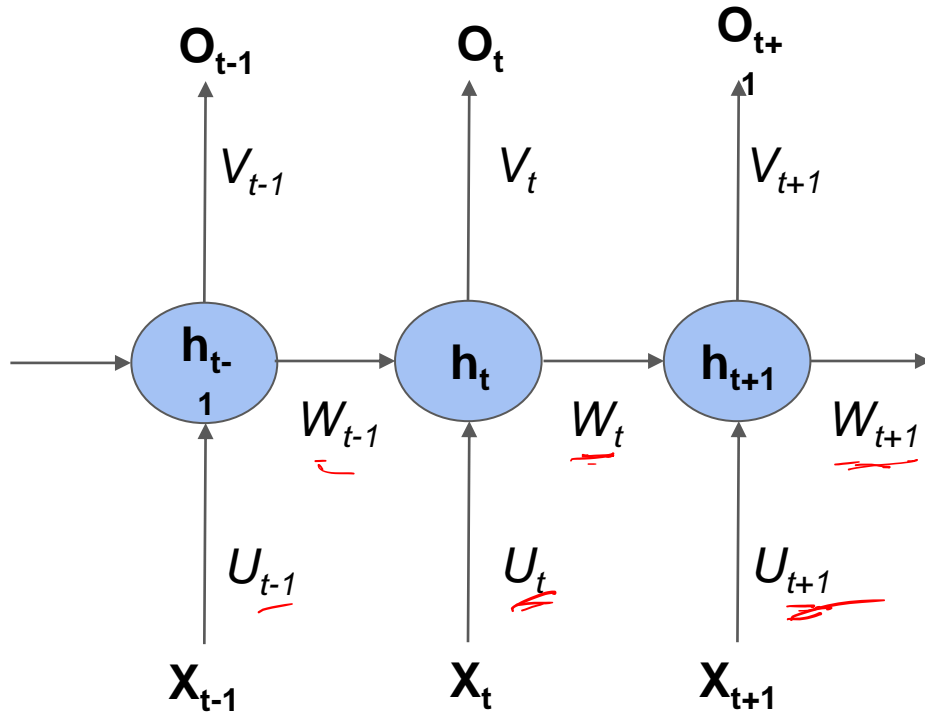
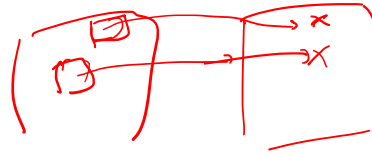
- Handles sequence? Yes ✓
- Models dependency? Yes ✓
- Variable length sequences?

A revised architecture



- Handles sequence? Yes ✓
- Models dependency? Yes ✓
- Variable length sequences? No ✗

A revised architecture



- Handles sequence? Yes ✓
- Models dependency? Yes ✓
- Variable length sequences? No ✗
- Still many parameters to train :(✗

Lessons from CNNs and vanilla networks

- Fully connected layers are the main cause for huge number of parameters in any neural network
- Increases the memory required and time taken to train the network
- Issues with convergence to minima
- High chance to overfit the data

Lessons from CNNs and vanilla networks

- Fully connected layers are the main cause for huge number of parameters in any neural network
 - Increases the memory required and time taken to train the network
 - Issues with convergence to minima
 - High chance to overfit the data
-
- Use CNNs to solve these problems
 - Main idea of a CNN is **parameter sharing**

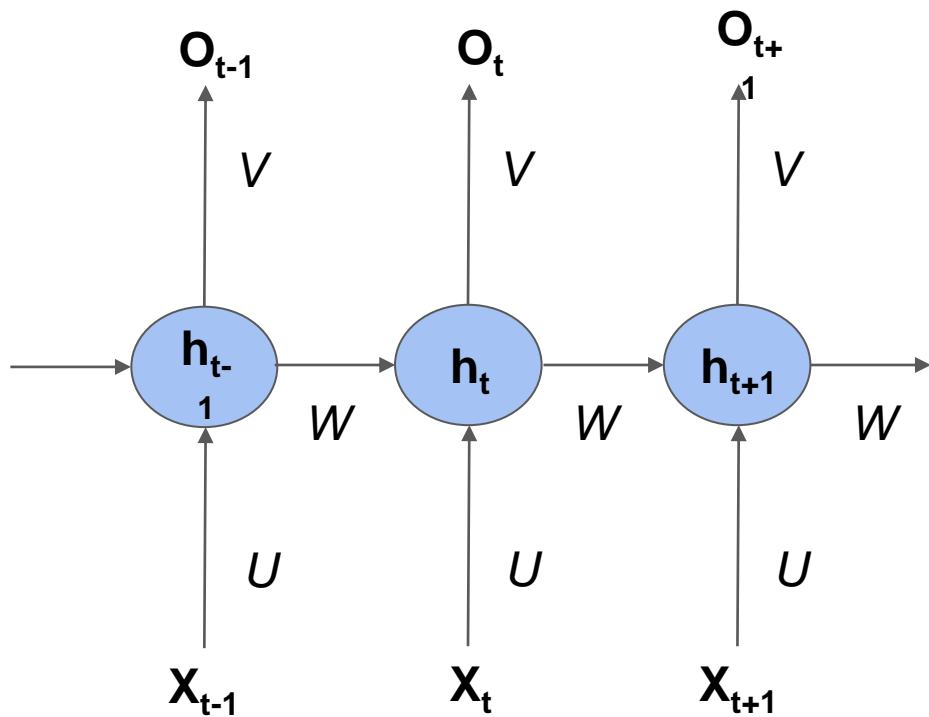
Lessons from CNNs and vanilla networks

- Fully connected layers are the main cause for huge number of parameters in any neural network
- Increases the memory required and time taken to train the network
- Issues with convergence to minima
- High chance to overfit the data

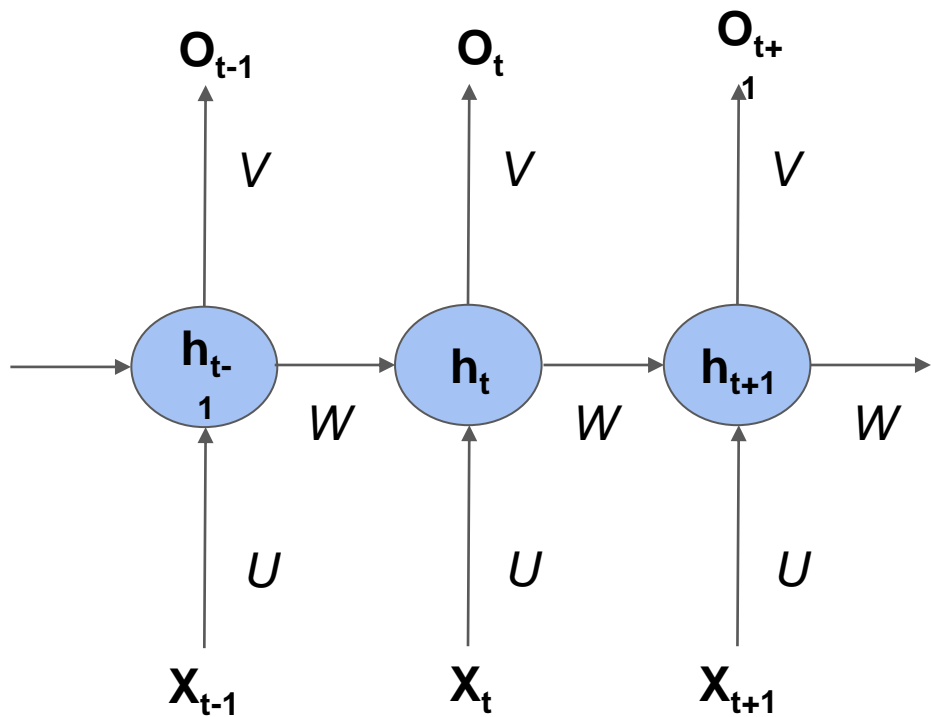
- Use CNNs to solve these problems
- Main idea of a CNN is **parameter sharing**

Can we now modify our network based on these?

A better architecture

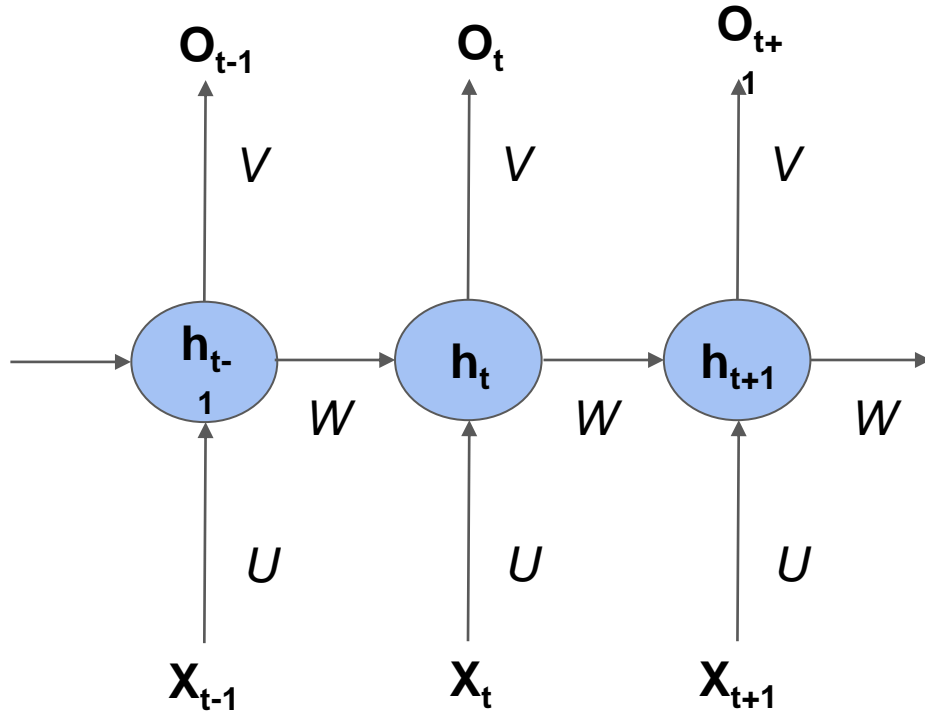


A better architecture



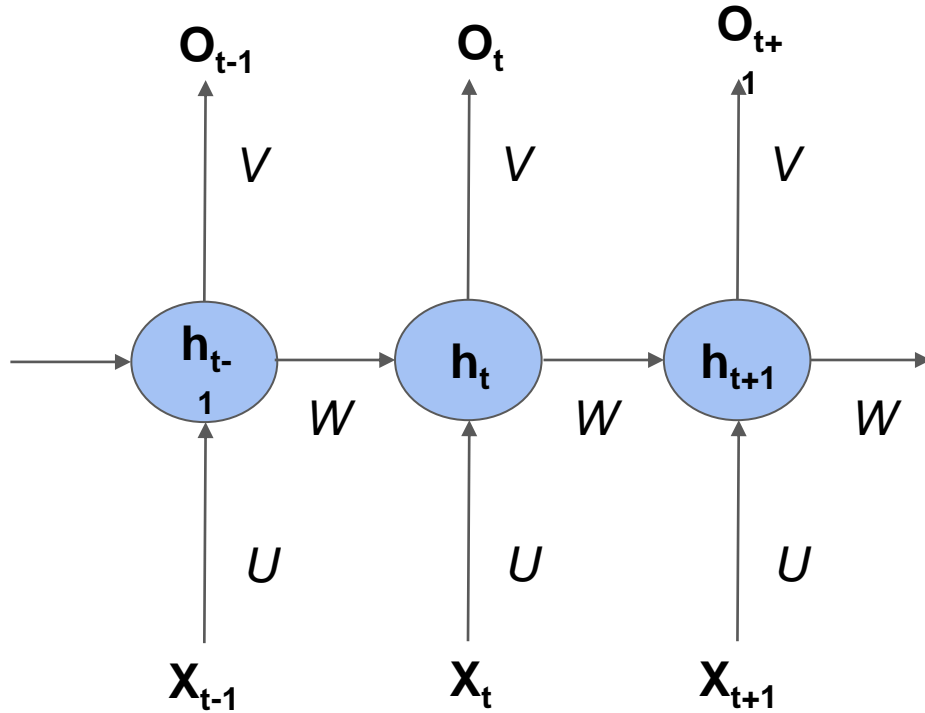
- Handles sequence?

A better architecture



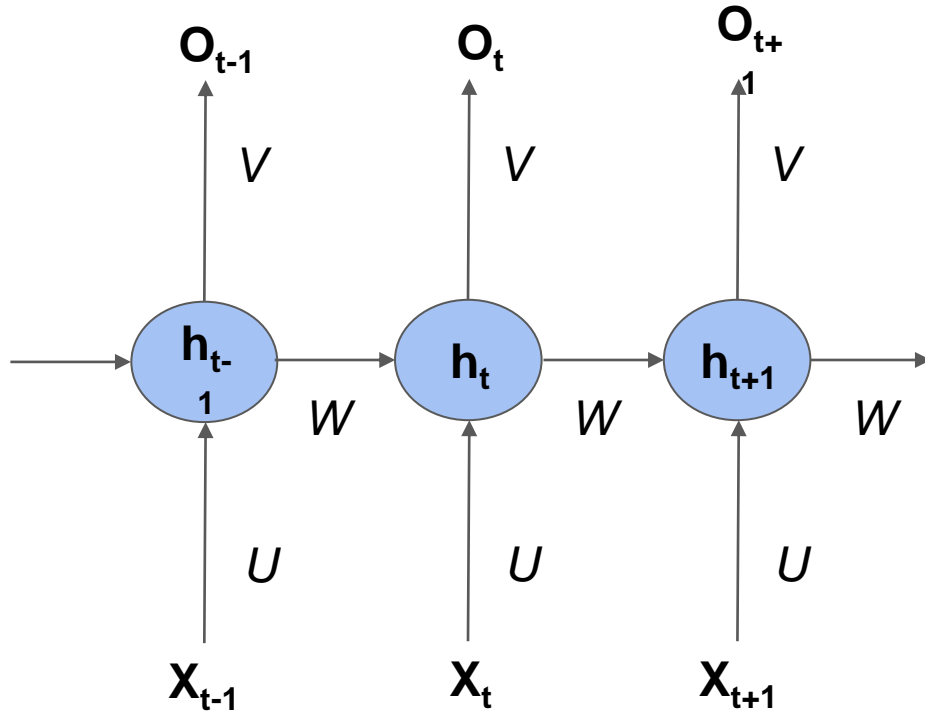
- Handles sequence? Yes ✓

A better architecture



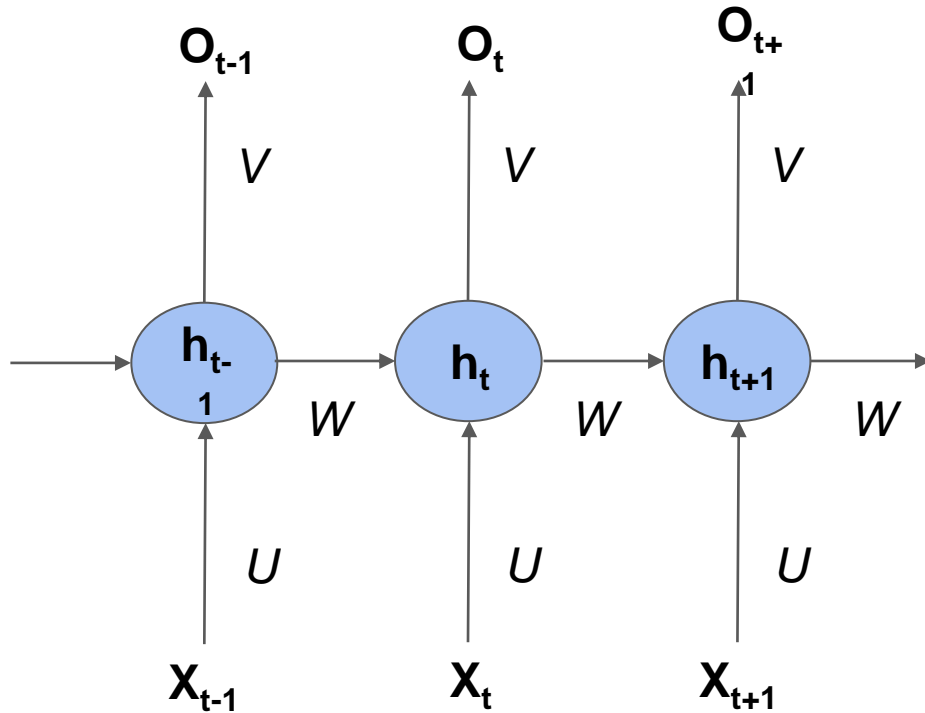
- Handles sequence? Yes ✓
- Models dependency?

A better architecture



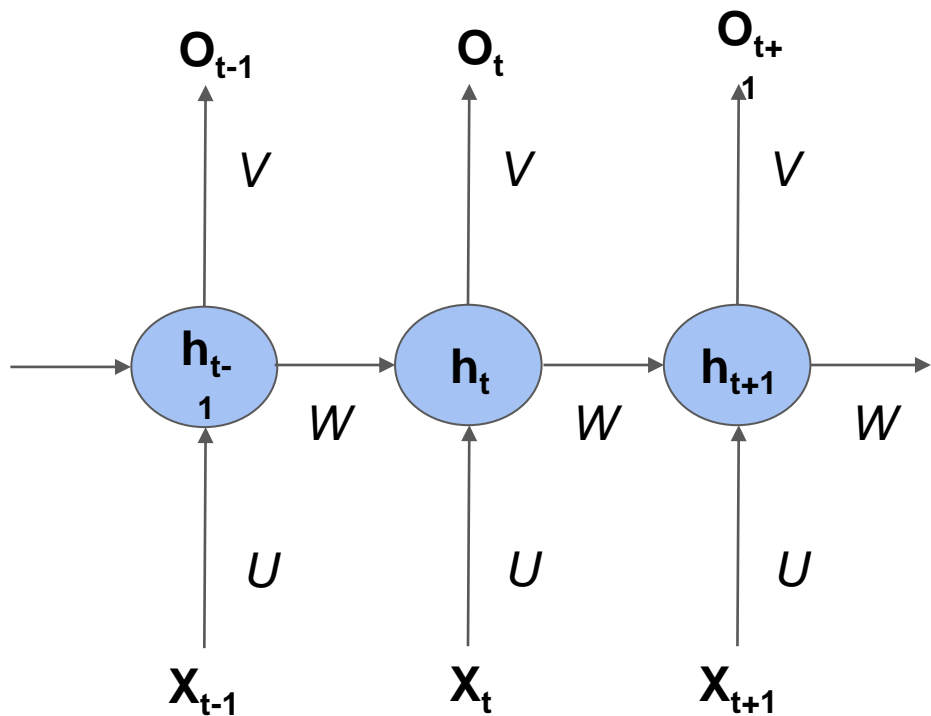
- Handles sequence? Yes ✓
- Models dependency? Yes ✓

A better architecture



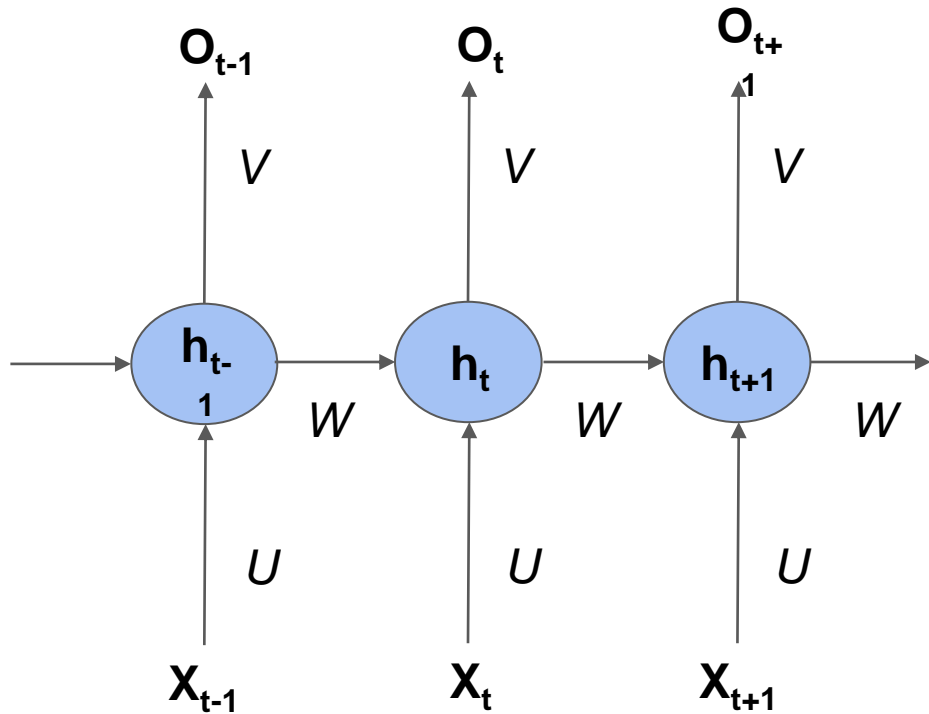
- Handles sequence? Yes ✓
- Models dependency? Yes ✓
- Variable length sequences?

A better architecture



- Handles sequence? Yes ✓
- Models dependency? Yes ✓
- Variable length sequences? Yes ✓

A better architecture



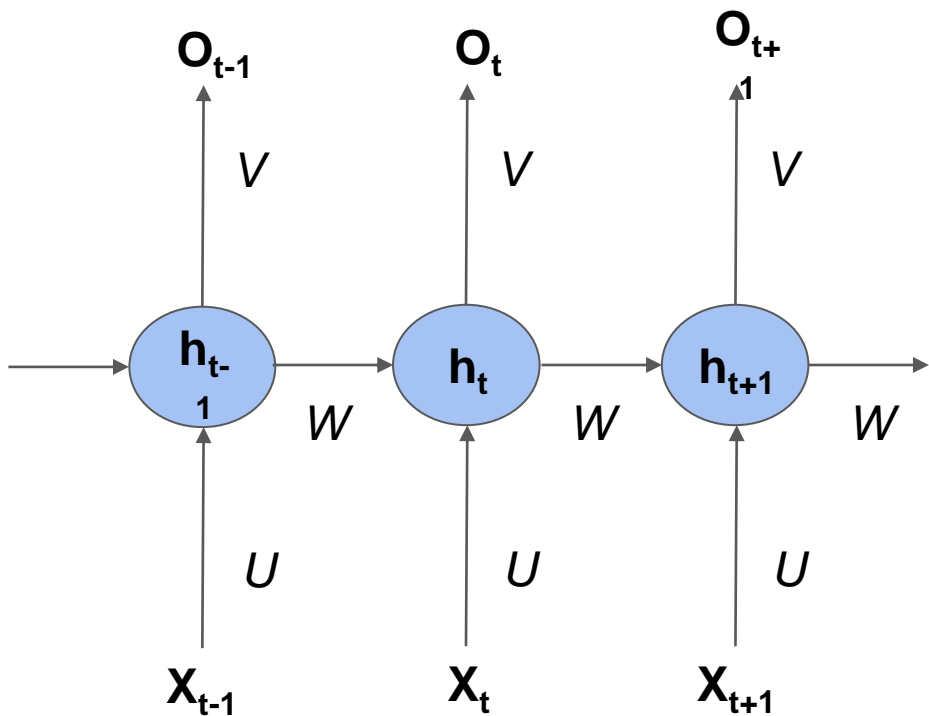
$$h_t = f(Ux_t + Wh_{t-1} + \dots)$$
$$o_t = g(Vh_t)$$

- Handles sequence? Yes ✓
- Models dependency? Yes ✓
- Variable length sequences? Yes ✓
- Parameter sharing :) ✓

To summarize...

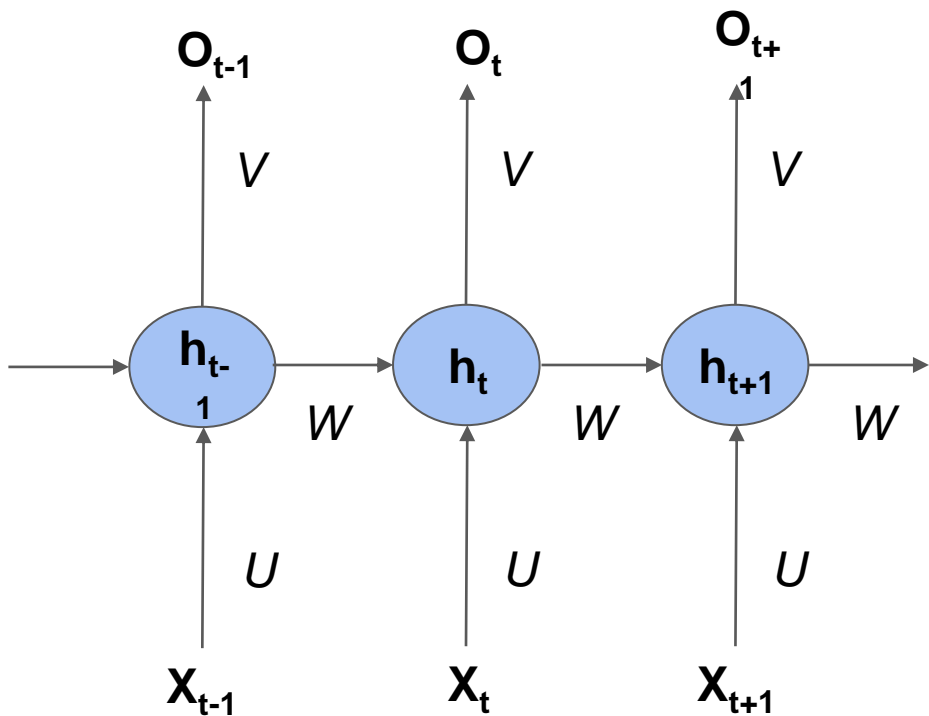
- Recurrent networks model sequences of data
$$\{x_1, x_2, \dots, x_t, \dots, x_{n-1}, x_n\} \longrightarrow \{y_1, y_2, \dots, y_t, \dots, y_{m-1}, y_m\}$$
- In transition from multi-layer networks to recurrent networks, **parameter sharing** plays an important role
- Instead of using different weights at each time-step, we use same weights at all time-steps
- Parameter sharing makes it possible to generalize and apply the model to cases where inputs/outputs are of different lengths, not seen during training

Recurrent networks



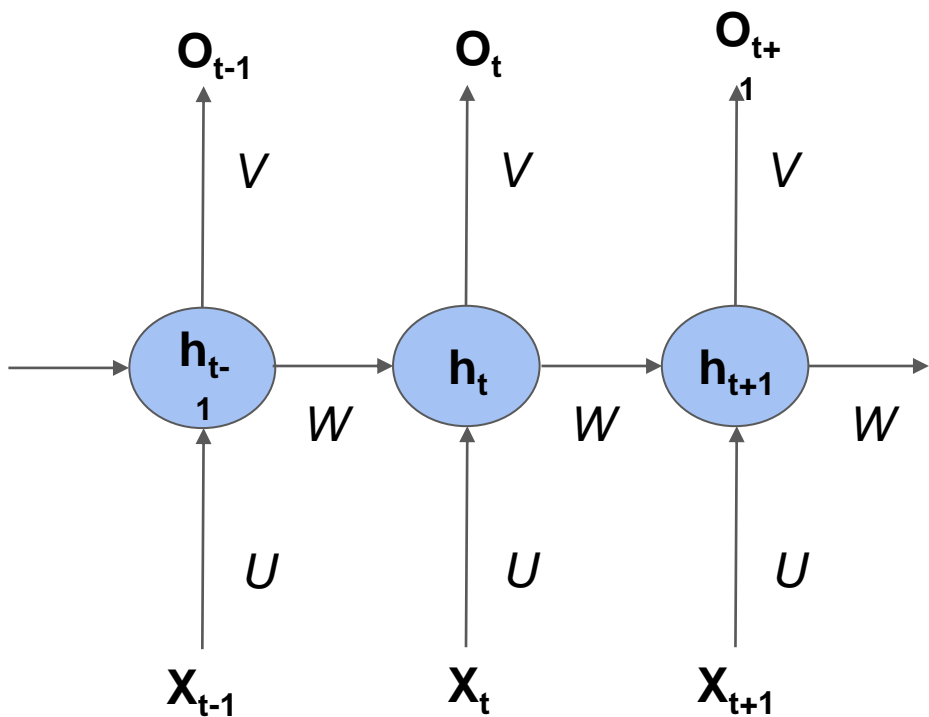
- At each time-step t ,
$$h_t = f(W h_{t-1} + U x_t)$$
$$o_t = g(V h_t)$$

Recurrent networks



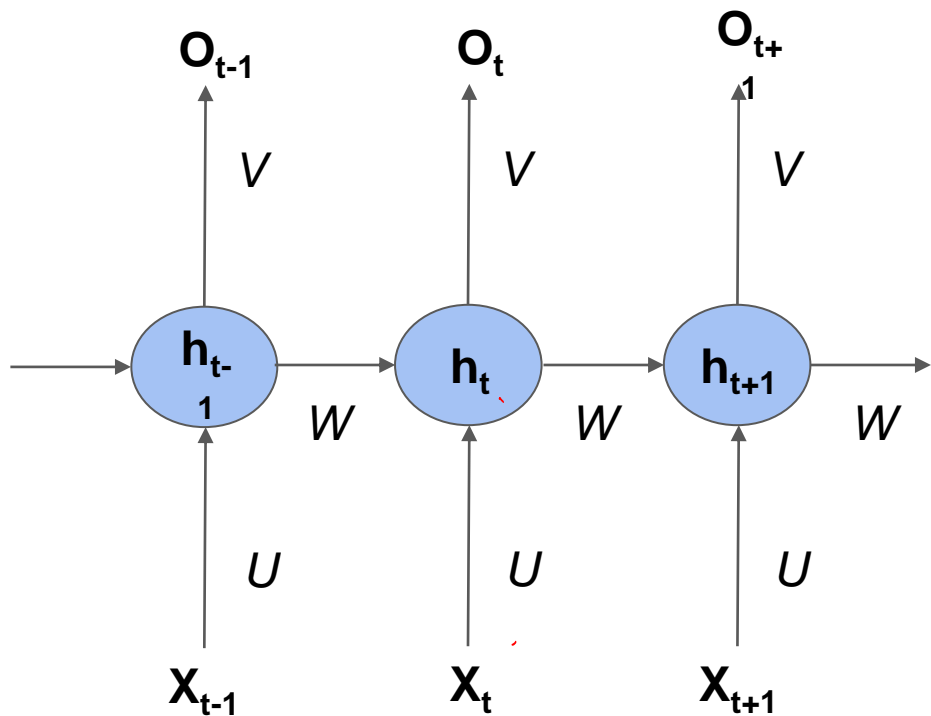
- At each time-step t ,
$$h_t = f(W h_{t-1} + U x_t)$$
$$o_t = g(V h_t)$$
- Here, f and g are activation functions, tanh, sigmoid, ReLU etc., and h_t is called the **hidden state** at time-step t and W, U, V are shared weights

Recurrent networks



- At each time-step t ,
$$h_t = f(W h_{t-1} + U x_t)$$
$$o_t = g(V h_t)$$
- Here, f and g are activation functions, tanh, sigmoid, ReLU etc., and h_t is called the **hidden state** at time-step t and W, U, V are shared weights

Recurrent networks



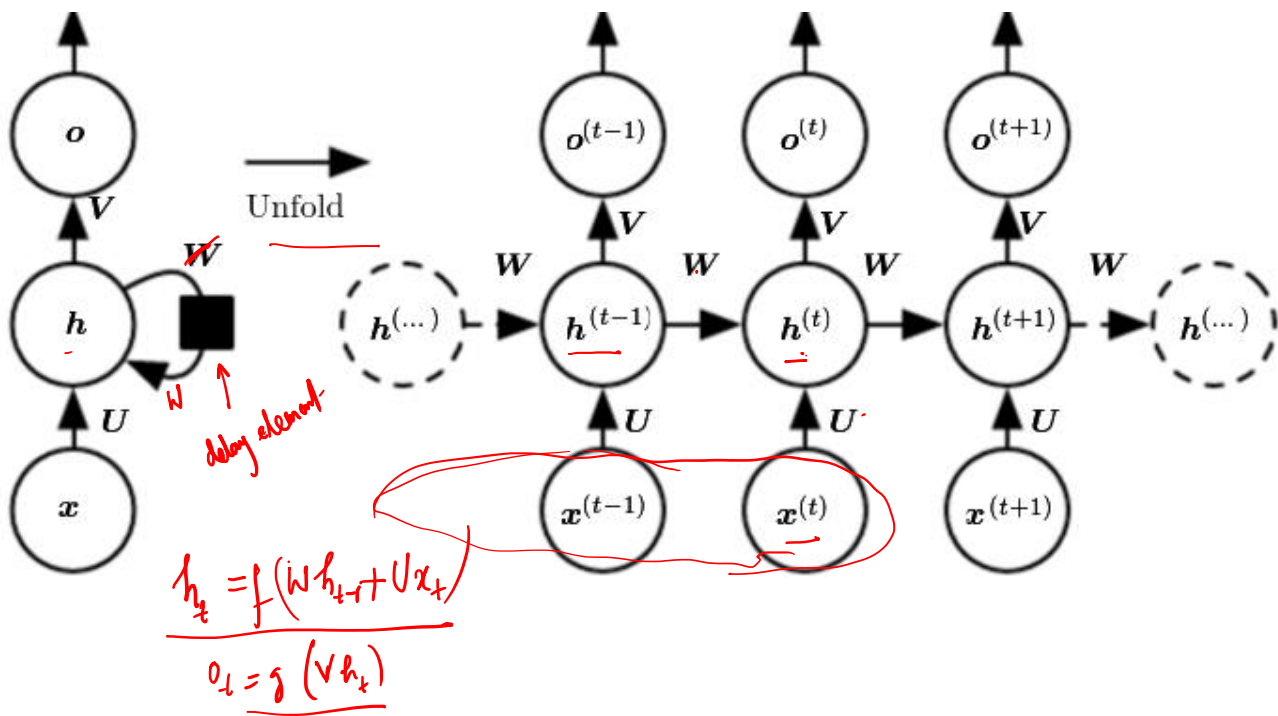
$$h_t = f \left(\underset{\substack{d \times n \\ \text{matrix}}}{U} \underset{\substack{n \times 1 \\ \text{vector}}}{x_t} + \underset{\substack{d \times d \\ \text{matrix}}}{W} \underset{\substack{d \times 1 \\ \text{vector}}}{h_{t-1}} \right)$$

$$o_t = g \left(\underset{\substack{k \times d \\ \text{matrix}}}{V} \underset{\substack{d \times 1 \\ \text{vector}}}{h_t} \right)$$

- Dimensions of the parameters

- $x_i \in \mathbb{R}^n$ (n -dimensional)
- $h_i \in \mathbb{R}^d$ (d -dimensional)
- $o_i \in \mathbb{R}^k$ (k -classes)
- $U \in \mathbb{R}^{d \times n}$
- $V \in \mathbb{R}^{k \times d}$
- $W \in \mathbb{R}^{d \times d}$

Unfolding Recurrent Networks



RNNs can be modelled as a computational graph with repetitive structure, corresponding to a chain of events

Recurrent Neural Network

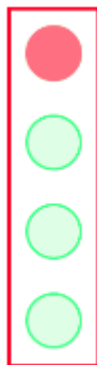
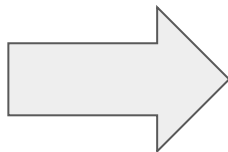
- RNNs are called recurrent because they perform the same task for every element in the sequence
- The only thing that differs is the input at each time-step
- Output at each time-step is dependent on previous computations (states)
- RNNs can be seen as a NN having memory about what has been calculated so far

Key challenges in sequence learning

- Different input and output sequence lengths
 - **Eg:** Machine translation, Speech-to-text
- Long range dependencies (context i.e, output depends on previous inputs)
 - **Eg:** Sentiment analysis, Readability analysis, Language modelling
- Dependence of output on the previous outputs so far
 - **Eg:** Word completion, sentence completion/predictions
- Unknown input-output alignment
 - **Eg:** Text image \rightarrow text string, or speech \rightarrow text

Different flavors of RNNs

Different flavors of RNNs



dog

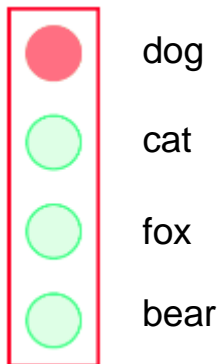
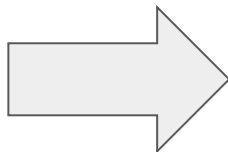
cat

fox

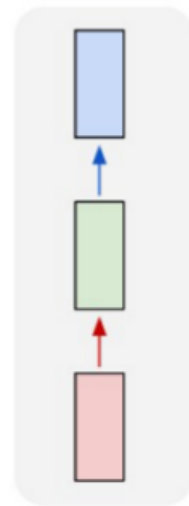
bear

Architecture?
RNN? CNN?

Different flavors of RNNs



one to one



Different flavors of RNNs



"Text RECOGNITION"

Architecture?



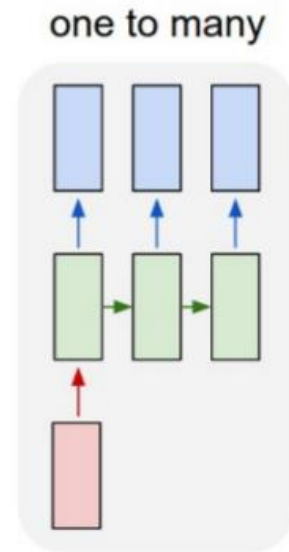
A bird flying over a body of water

Different flavors of RNNs

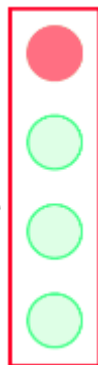
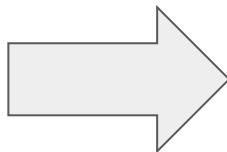
  "Text RECOGNITION"



 A bird flying over a body of water



Different flavors of RNNs



running

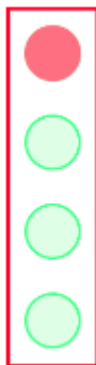
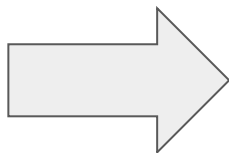
walking

clapping

boxing

Architecture?

Different flavors of RNNs



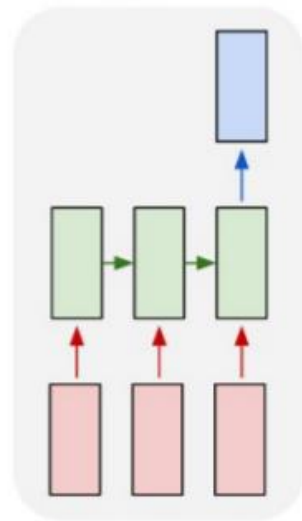
running

walking

clapping

boxing

many to one



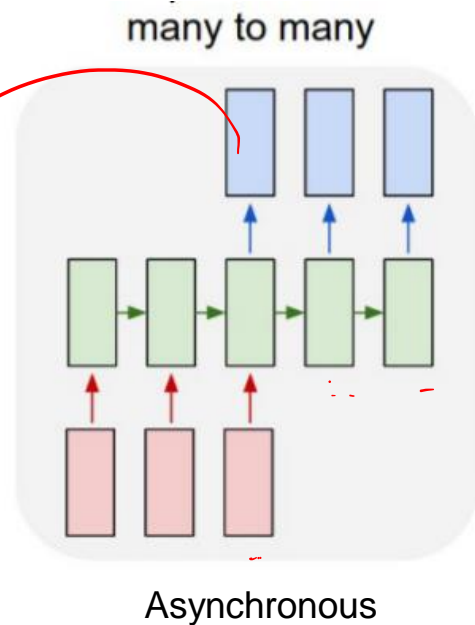
Different flavors of RNNs

Hello, how are you?  नमस्ते आप कैसे हैं?

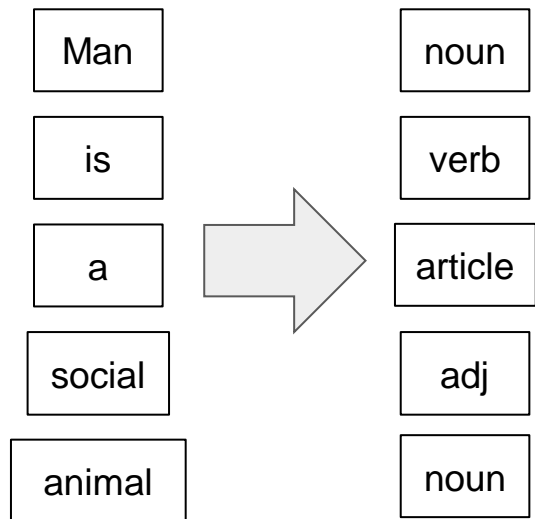
Architecture?

Different flavors of RNNs

Hello, how are you? ↔ नमस्ते आप कैसे हैं?



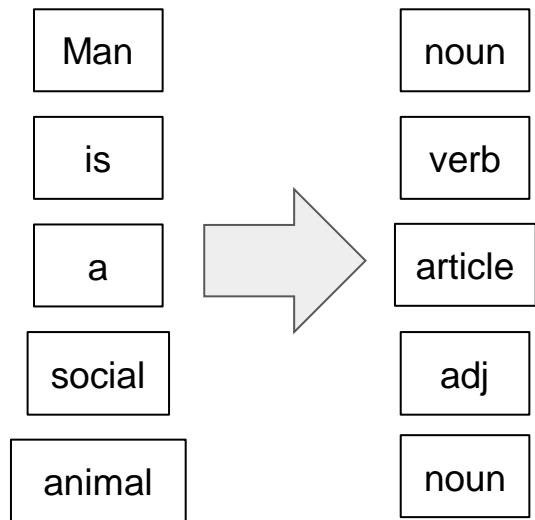
Different flavors of RNNs



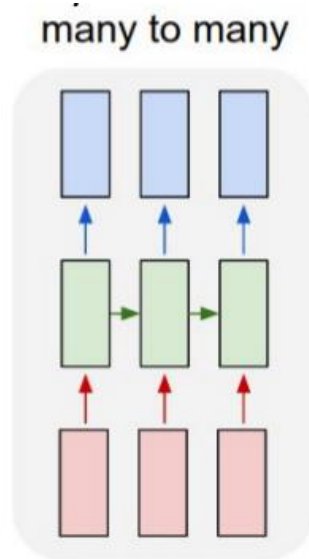
Part-of-speech
tagging

Architecture?

Different flavors of RNNs



Part-of-speech
tagging

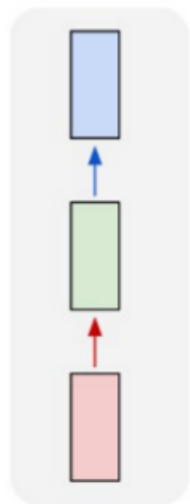


Synchronous

Different flavors of RNNs

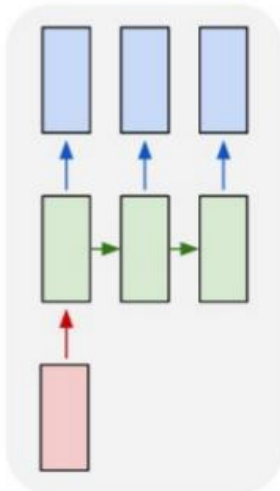
Image
classification

one to one



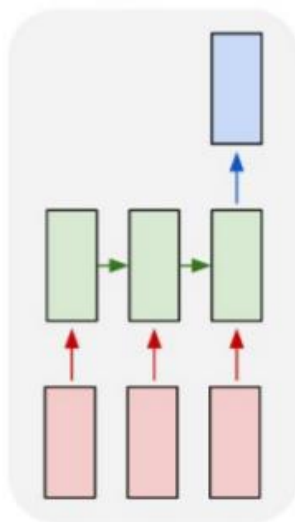
Scene text,
Image
Captioning

one to many



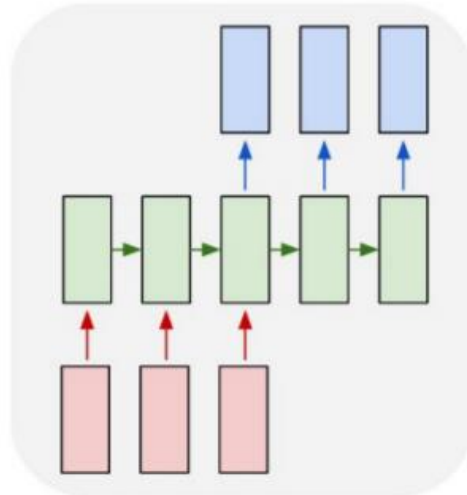
Sentiment analysis,
Activity recognition
from video

many to one



Machine translation

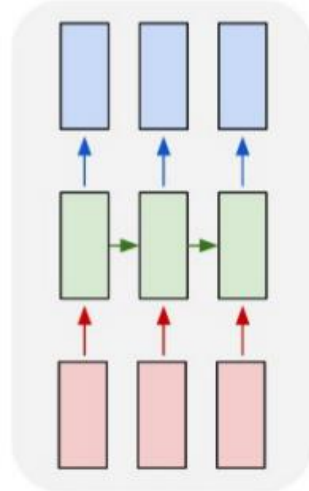
many to many



Asynchronous

Part-of-speech
tagging

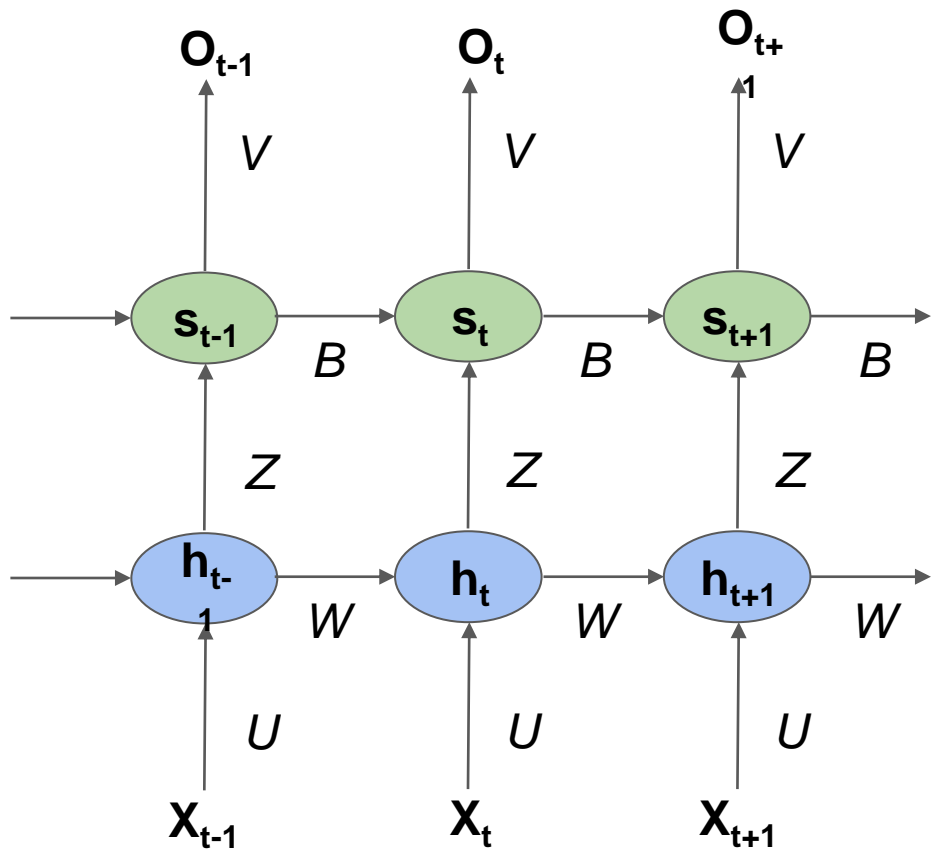
many to many



Synchronous

Different flavors of RNNs

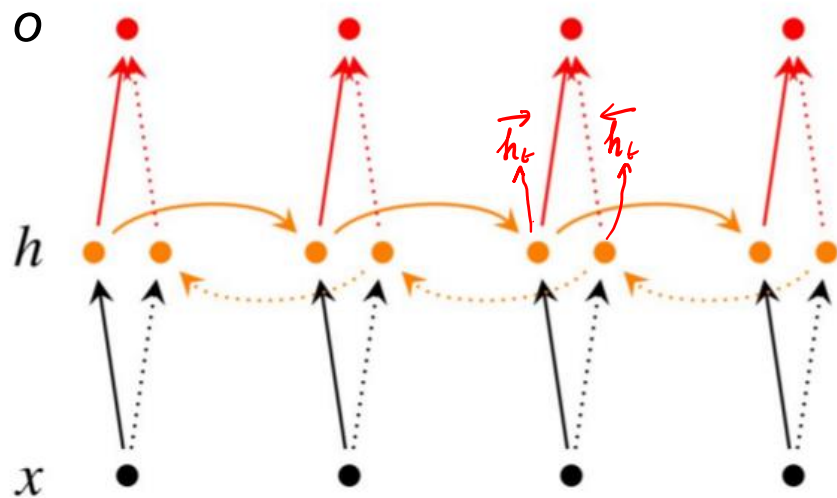
$$s_t = f(z h_t + B s_{t-1}) \quad o_t = g(V s_t)$$



Stacked RNNs

- Stacked RNN with two hidden layers
- Hidden state of the first RNN is passed as the input to the second RNN
- The RNN state sizes can be changed
- How many layers to stack?
 - “Deeper is better”, but usually two layers should suffice

Bidirectional RNN



$h_t = [\overleftarrow{h}_t; \vec{h}_t]$ summarizes the past and the future in a single vector

$$\vec{h}_t = f(\vec{W} \vec{h}_{t-1} + \vec{U} x_t)$$

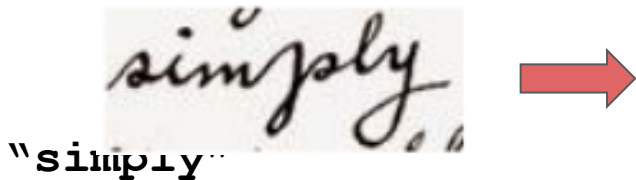
$$\overleftarrow{h}_t = f(\overleftarrow{W} \overleftarrow{h}_{t+1} + \overleftarrow{U} x_t)$$

$$o_t = g\left(\underbrace{V}_{k=1} \left[\overleftarrow{h}_t; \vec{h}_t \right] \right)$$

$2d \times 1$

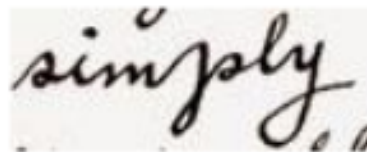
Case study: Scene text recognition

- **Problem:** Extract the text from an image



Case study: Scene text recognition

- **Problem:** Extract the text from an image



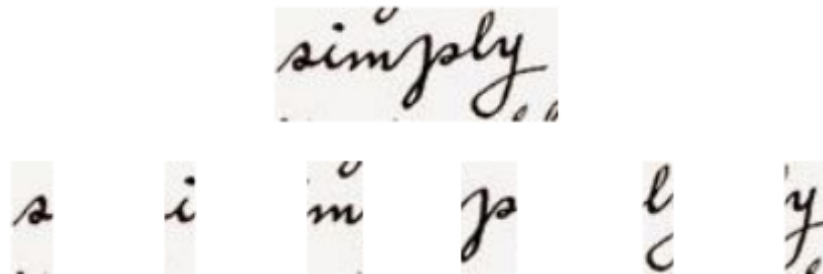
"simply"

Can you design an architecture for this?

A simple architecture with character segmentation

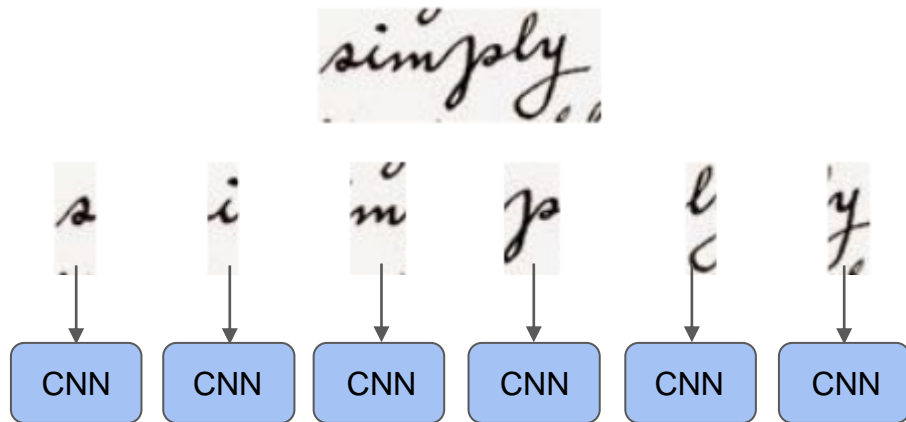
A simple architecture with character segmentation

- Segment the image into characters



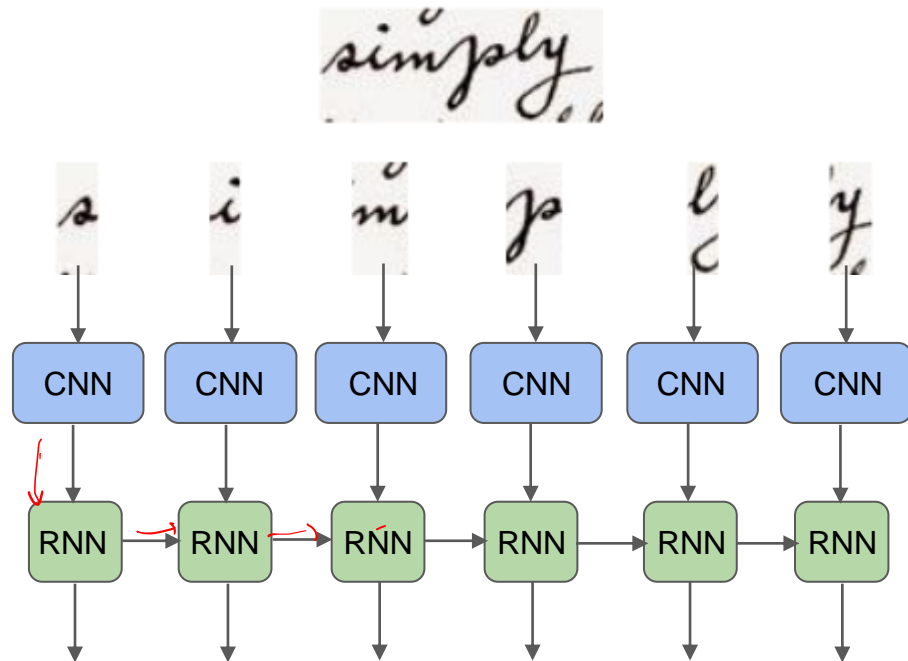
A simple architecture with character segmentation

- Segment the image into characters
- Extract the features using CNN for each segment



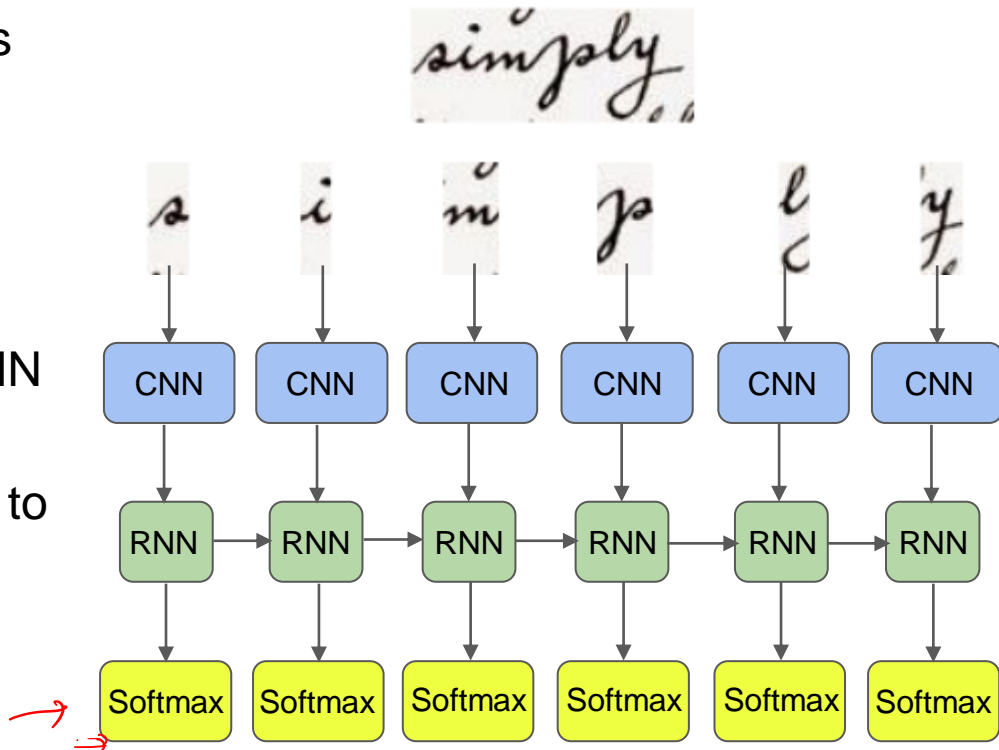
A simple architecture with character segmentation

- Segment the image into characters
- Extract the features using CNN for each segment
- Pass those features as input to RNN



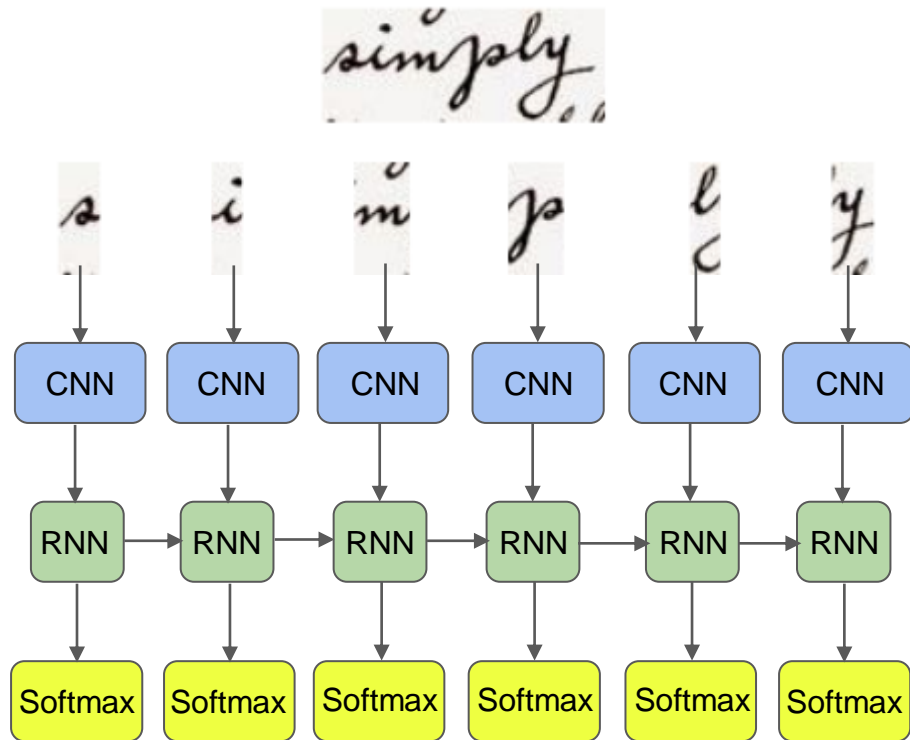
A simple architecture with character segmentation

- Segment the image into characters
- Extract the features using CNN for each segment
- Pass those features as input to RNN
- Pass RNN output through softmax to get alphabet probabilities



A simple architecture with character segmentation

- Segment the image into characters
- Extract the features using CNN for each segment
- Pass those features as input to RNN
- Pass RNN output through softmax to get alphabet probabilities
- What loss function should we use?

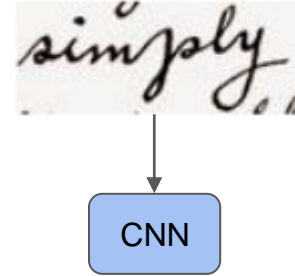


Can we improve this architecture?

Architecture without character segmentation

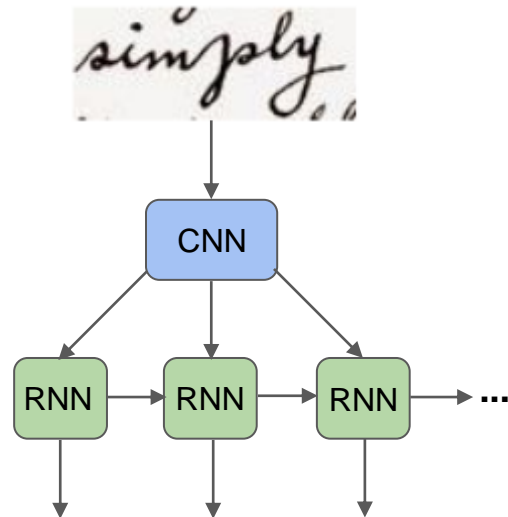
Architecture without character segmentation

- Extract the features of the image using CNN



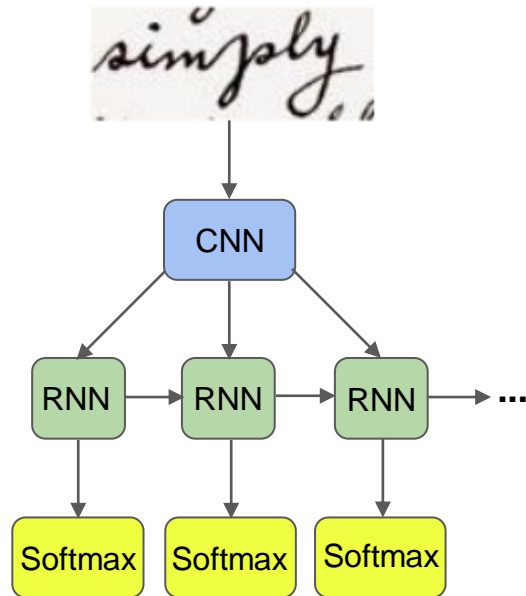
Architecture without character segmentation

- Extract the features of the image using CNN
- Pass those features as input to RNN at each time-step



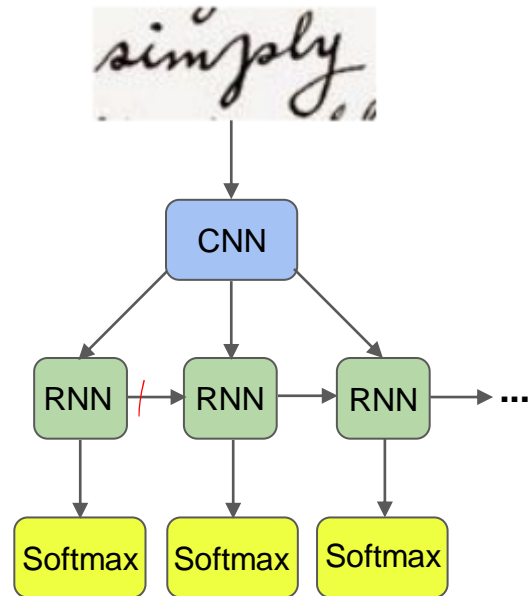
Architecture without character segmentation

- Extract the features of the image using CNN
- Pass those features as input to RNN at each time-step
- Pass RNN output through softmax to get alphabet probabilities



Architecture without character segmentation

- Extract the features of the image using CNN
- Pass those features as input to RNN at each time-step
- Pass RNN output through softmax to get alphabet probabilities
- What loss function should we use?



How to train RNNs??

But before that

How to calculate the Loss??

Calculating the loss

- The total loss is nothing but the **sum of the loss over all time-steps**
- Let's say the RNNs outputs a probability distribution over all classes at each time-step \mathbf{O}_t
- Let the ground truth be represented as one-hot vector \mathbf{Y}_t
- Then the loss at that time-step is simply the **cross-entropy** loss between the vectors \mathbf{O}_t and \mathbf{Y}_t
- We can replace that cross-entropy loss function with any other loss depending on the application and the output of the RNN

Calculating the loss

- The total loss is,

$$\underline{\mathbb{L}}(\underline{\theta}) = \sum_{t=1}^T \mathbb{L}_t(\theta)$$

$$\frac{1}{N} \sum_{i=1}^N \sum_{t=1}^{T_i} L_{t,i}(\theta)$$

$\mathbb{L}_t(\theta)$ = loss at time-step t

T = Number of time-steps

$$\theta = (U, V, W)$$

How to train the network?

How to train the network?

Back propagation

Training RNNs

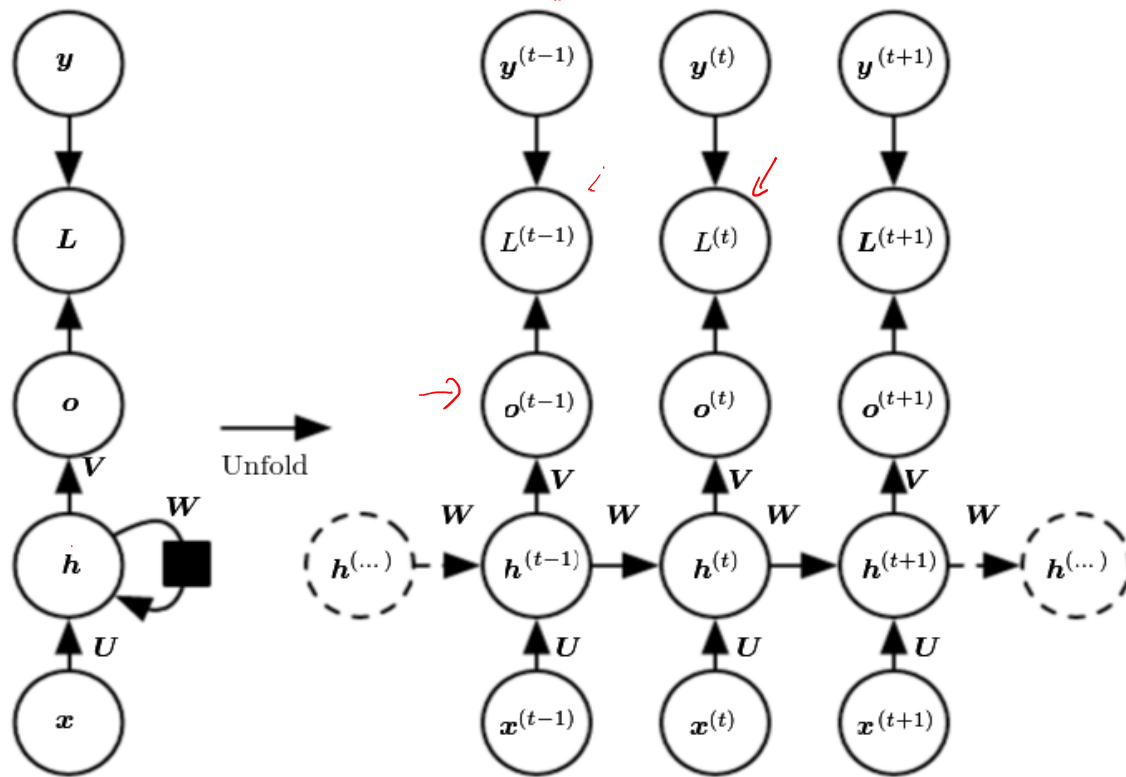
- Let's say the weights are \mathbf{U} , \mathbf{V} , \mathbf{W} as denoted previously
- Using backpropagation, we need to compute the gradients of the loss with respect to \mathbf{U} , \mathbf{V} , \mathbf{W}

- Let's see how to do that

$$\begin{aligned} \mathbf{V} &\leftarrow \mathbf{V} - \alpha \left[\frac{\partial L}{\partial \mathbf{V}} \right] \\ \mathbf{V} & \\ \mathbf{W} & \end{aligned} \quad \rightarrow \quad \frac{\partial L}{\partial \mathbf{V}} = \frac{1}{N} \sum \left(\frac{\partial L_i}{\partial \mathbf{V}} \right)$$

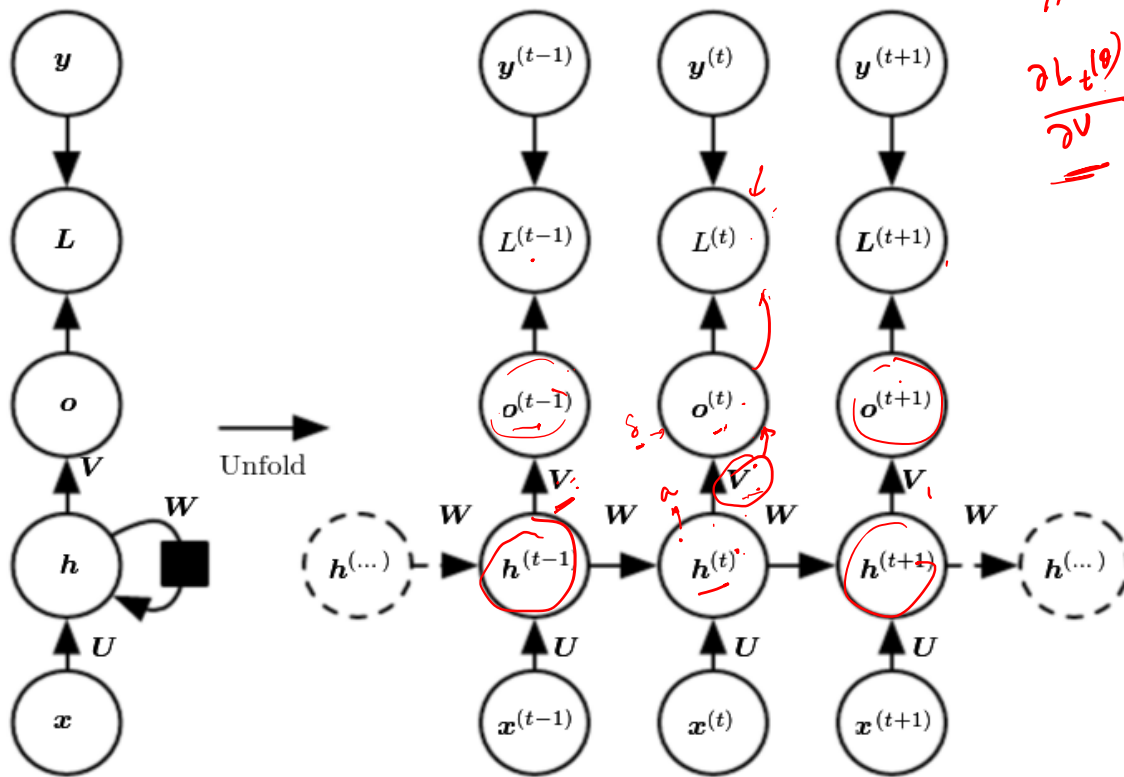
$L = \frac{1}{N} \sum_{i=1}^N L_i$

Training RNNs



- Let us consider $\nabla_V \mathbb{L}(\theta)$

Training RNNs



$$o^{(t)} = f(z^{(t)})$$

$$\delta = -\frac{\partial L}{\partial z^{(t)}}$$

$$a\delta = \frac{\partial L}{\partial z^{(t)}} h^{(t)}$$

$$o^{(t)} = f(Vh^{(t)})$$

$$z^{(t)} = Vh^{(t)}$$

$$\frac{\partial z^{(t)}}{\partial V} = h^{(t)}$$

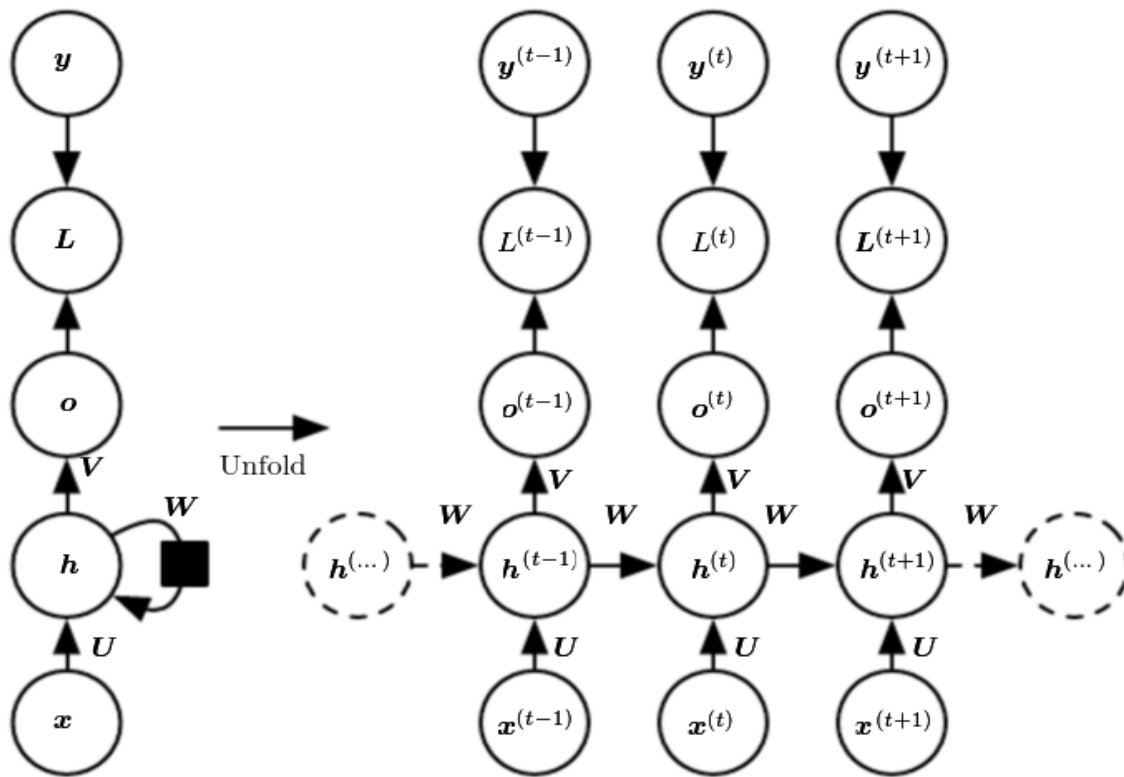
- Let us consider $\nabla_V \mathbb{L}(\theta)$

$$\frac{\partial L_t(\theta)}{\partial V} = \frac{\partial L_t(\theta)}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial V} = \delta \frac{\partial L_t(\theta)}{\partial z^{(t)}} \frac{\partial z^{(t)}}{\partial V} = \delta h^{(t)}$$

- But for simplicity, denote it as

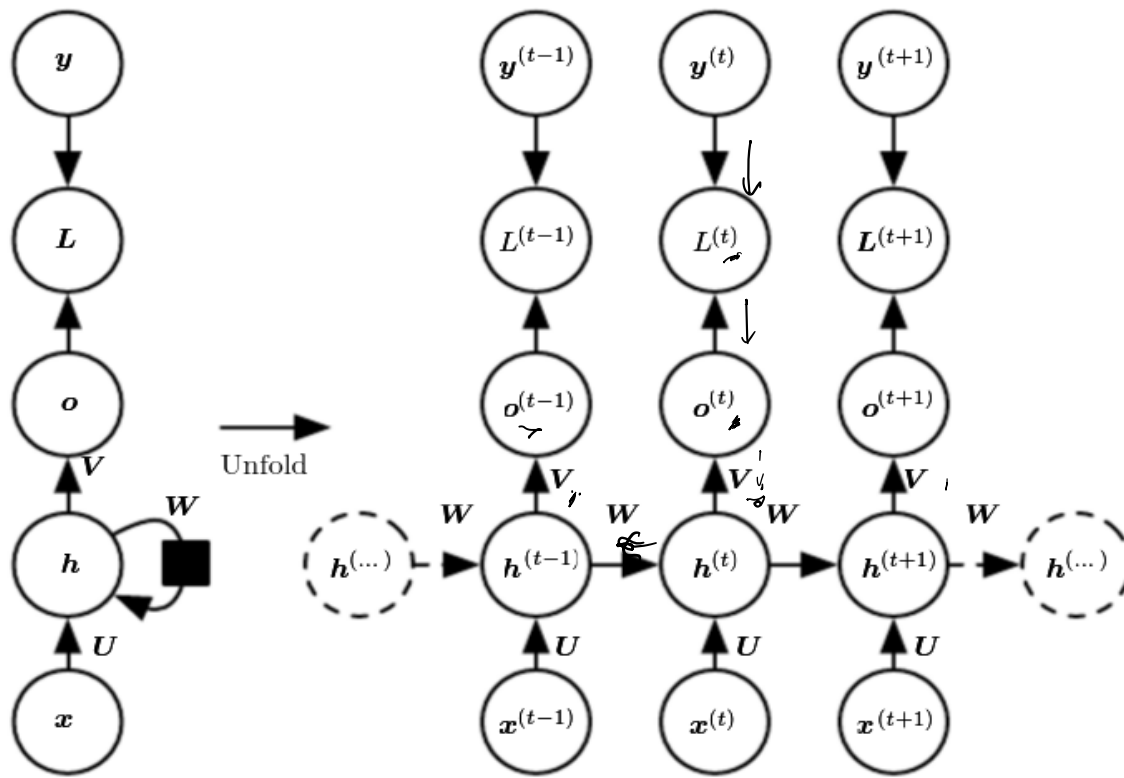
$$\frac{\partial \mathbb{L}(\theta)}{\partial V} = \sum_{t=0}^T \frac{\partial \mathbb{L}_t(\theta)}{\partial V}$$

Training RNNs



- Let us consider $\nabla_V \mathbb{L}(\theta)$
- But for simplicity, denote it as
$$\frac{\partial \mathbb{L}(\theta)}{\partial V} = \sum_{t=0}^T \frac{\partial \mathbb{L}_t(\theta)}{\partial V}$$
- Each term in the summation is the derivative of the loss w.r.t the weights in V

Training RNNs



$$\sum_{t=1}^T \frac{\partial \mathcal{L}}{\partial V} = \sum_{t=1}^T \frac{\partial \mathcal{L}^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial V}$$

- Let us consider $\nabla_V \mathbb{L}(\theta)$

- But for simplicity, denote it as

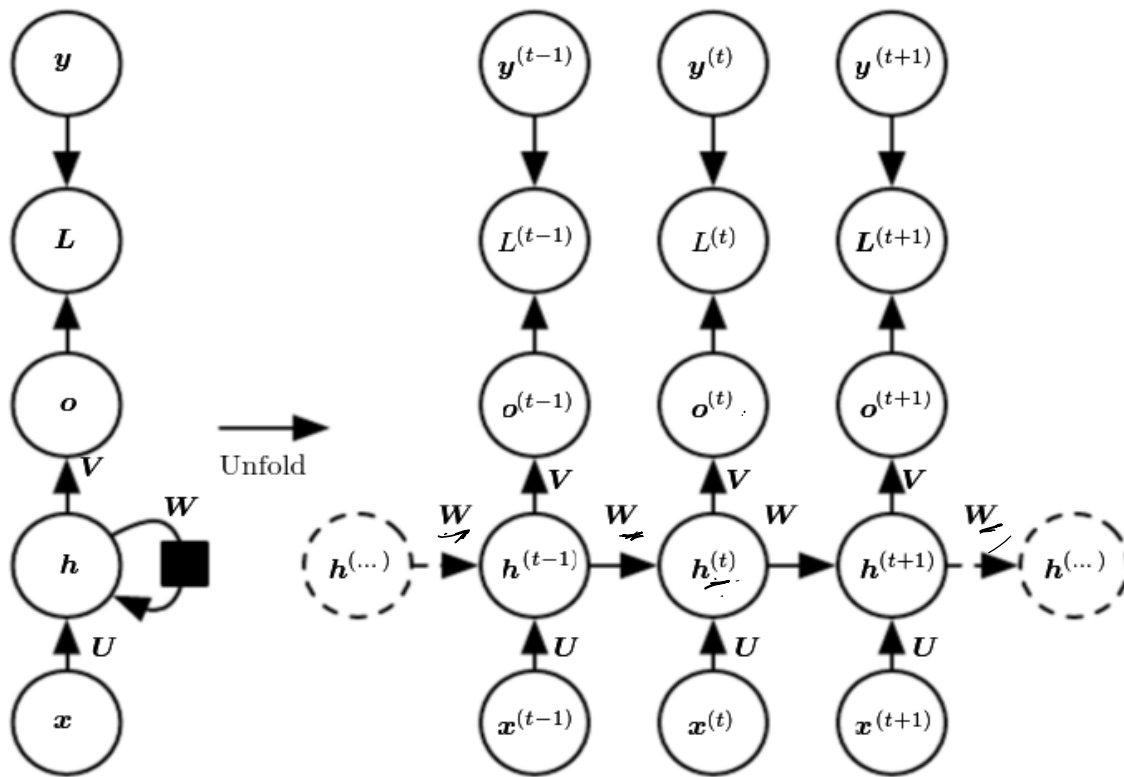
$$\frac{\partial \mathbb{L}(\theta)}{\partial V} = \sum_{t=0}^T \frac{\partial \mathbb{L}_t(\theta)}{\partial V}$$

- Each term in the summation is the derivative of the loss w.r.t the weights in V
- We've seen how to compute this when we studied backpropagation

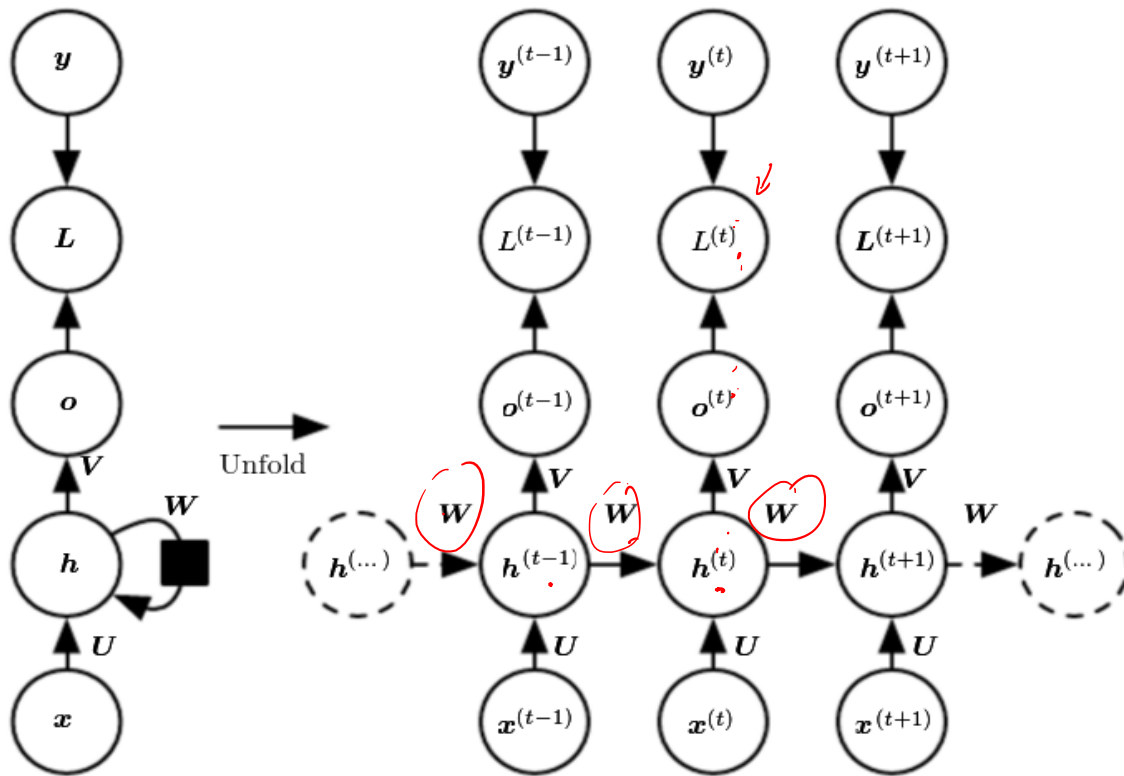
Training RNNs

$$\frac{\partial \mathcal{L}^{(t)}}{\partial W} = \frac{\partial \mathcal{L}^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \left[\frac{\partial h^{(t)}}{\partial W} \right]$$

- Let us consider $\nabla_{\underline{W}} \mathbb{L}(\theta)$



Training RNNs

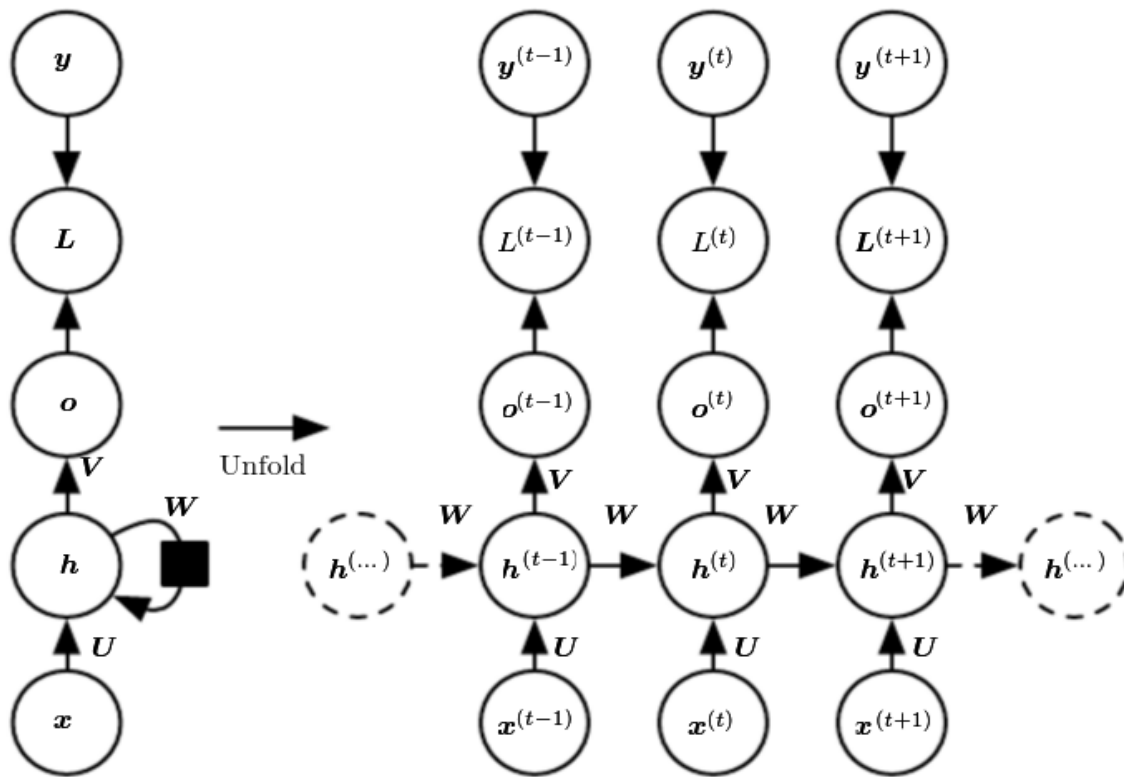


- Let us consider $\nabla_W \mathbb{L}(\theta)$

- But for simplicity, denote it as,

$$\frac{\partial \mathbb{L}(\theta)}{\partial W} = \sum_{t=0}^T \frac{\partial \mathbb{L}_t(\theta)}{\partial W}$$

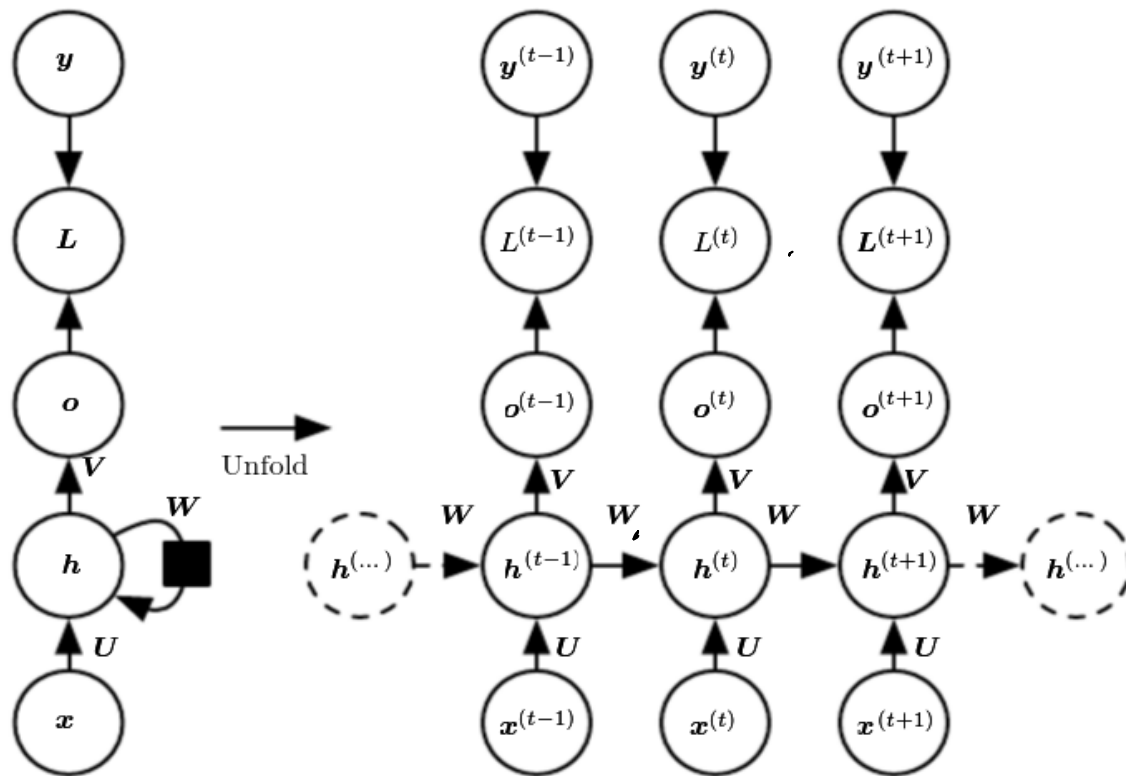
Training RNNs



- Let us consider $\nabla_W \mathbb{L}(\theta)$
- But for simplicity, denote it as,

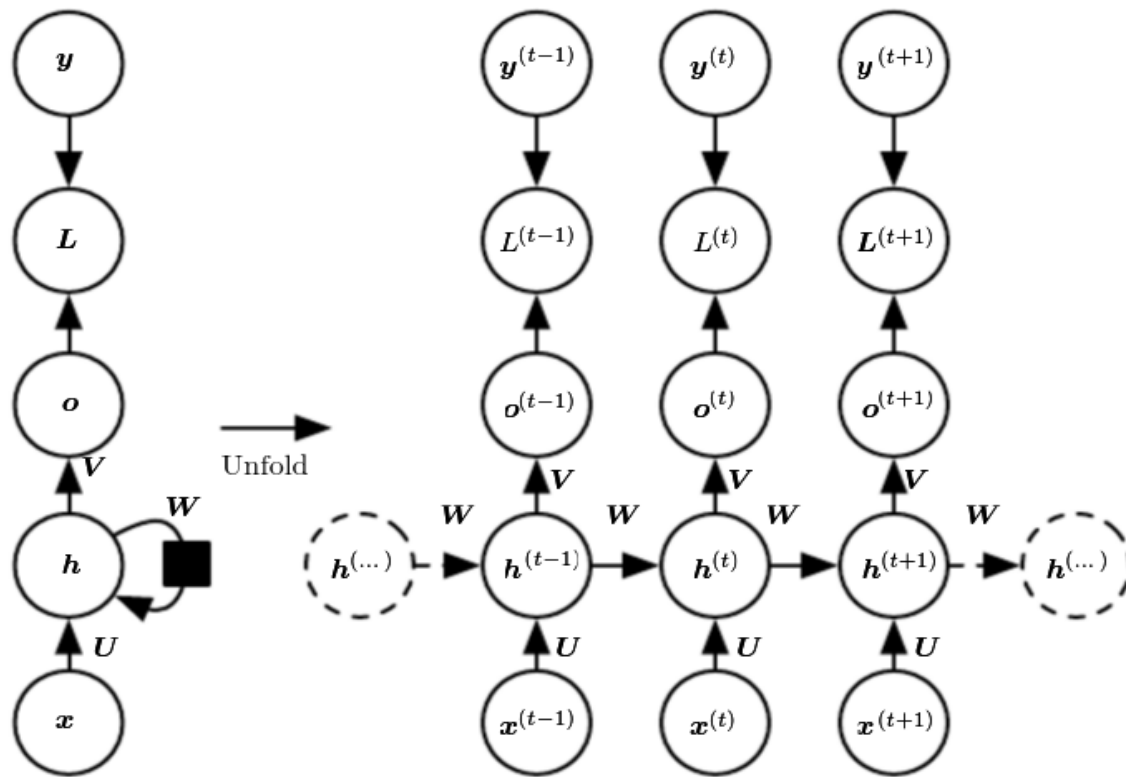
$$\frac{\partial \mathbb{L}(\theta)}{\partial W} = \sum_{t=0}^T \frac{\partial \mathbb{L}_t(\theta)}{\partial W}$$
- By chain rule of derivatives, we know that each term in RHS is obtained by summing gradients along all the paths from $\mathbb{L}_t(\theta)$ to \underline{W}

Training RNNs



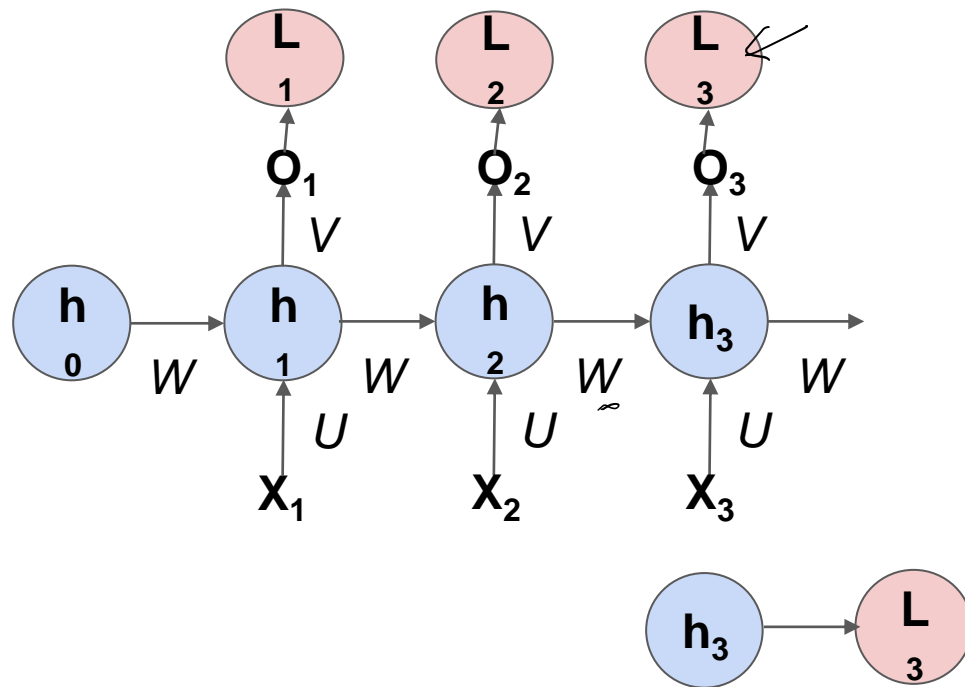
- What are the paths connecting $\mathbb{L}_t(\theta)$ to \mathbf{W} ?

Training RNNs



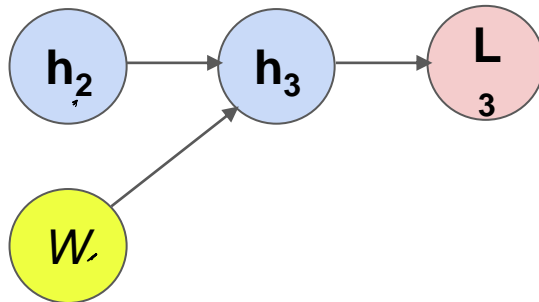
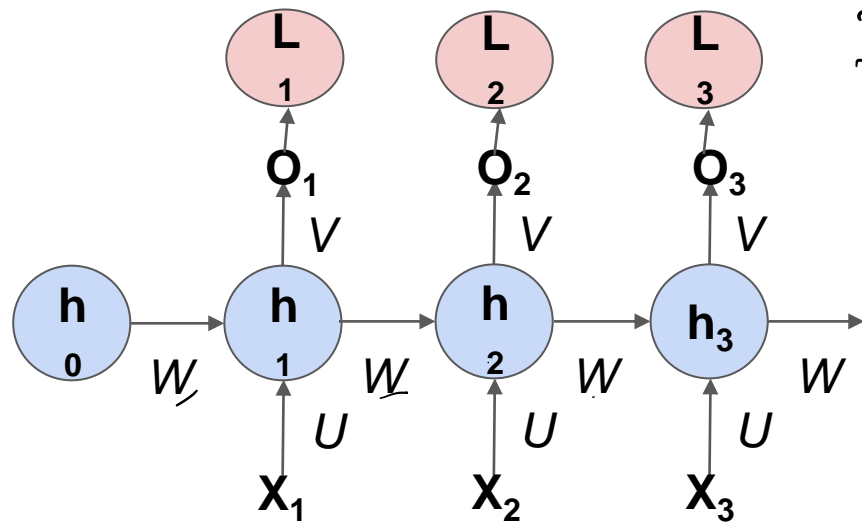
- What are the paths connecting $\mathbb{L}_t(\theta)$ to \mathbf{W} ?
- Let us analyze this by calculating $\mathbb{L}_3(\theta)$

Training RNNs



- $\mathbb{L}_3(\theta)$ depends on h_3

Training RNNs



$$\frac{\partial \mathcal{L}_3}{\partial o_3} \frac{\partial o_3}{\partial h_3}$$

↓

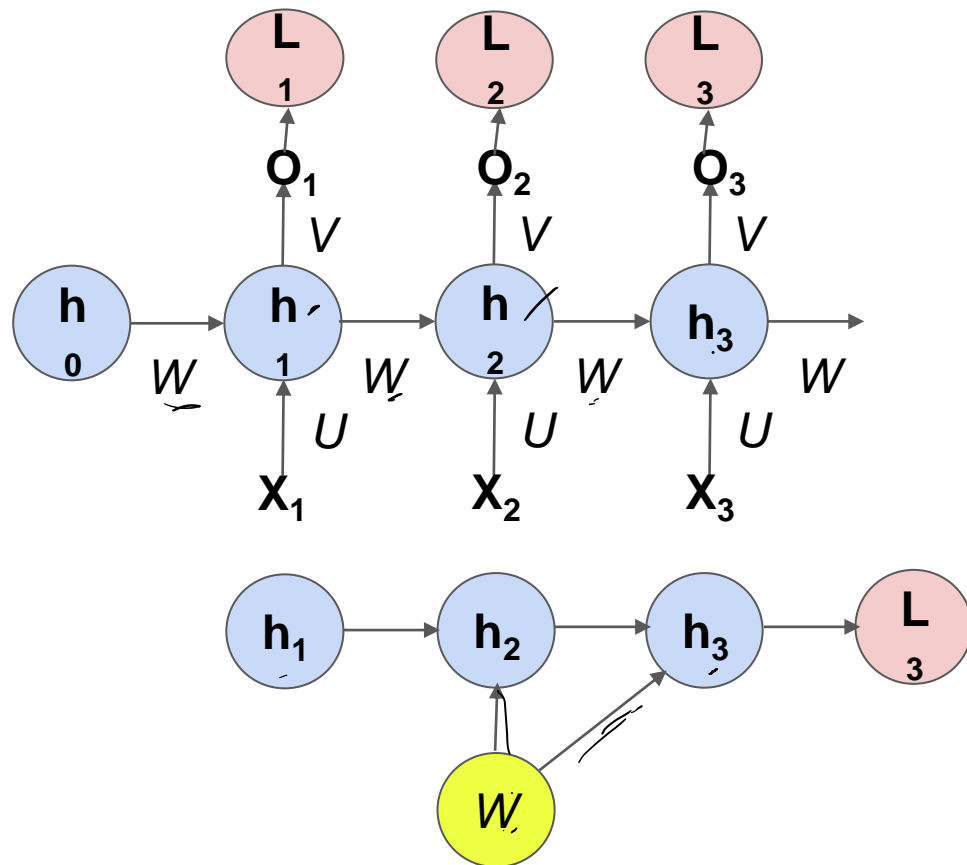
$$\frac{\partial \mathcal{L}_3}{\partial h_3}$$

• $\mathcal{L}_3(\theta)$ depends on h_3

• h_3 in-turn depends on h_2 and W

$$h_3 = f(h_2^{(w)} + Ux_3)$$

Training RNNs

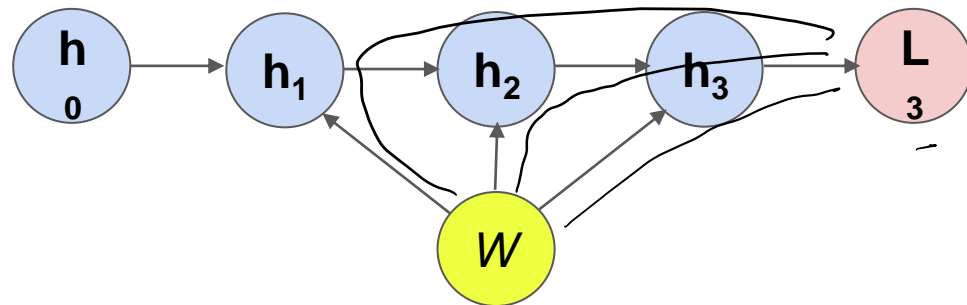
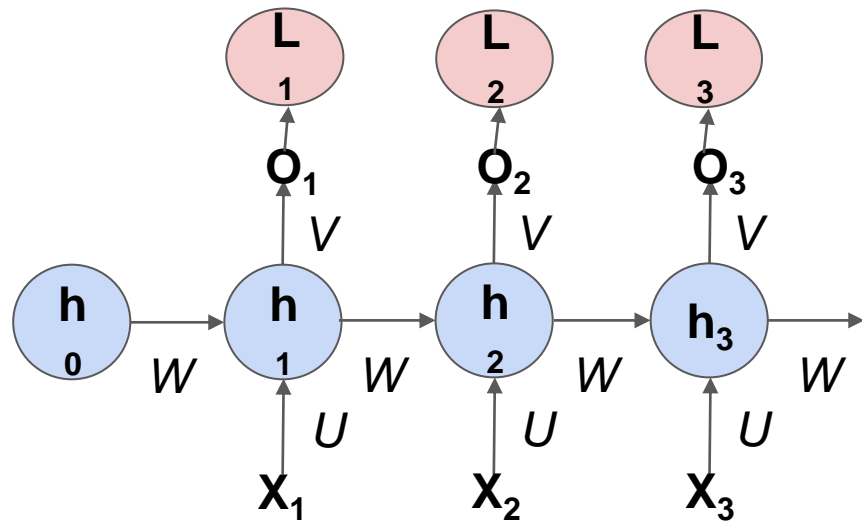


$$h_3 = f(W h_2 + U x_3)$$

- $\mathbb{L}_3(\theta)$ depends on h_3
- h_3 in-turn depends on h_2 and W
- h_2 in-turn depends on h_1 and W

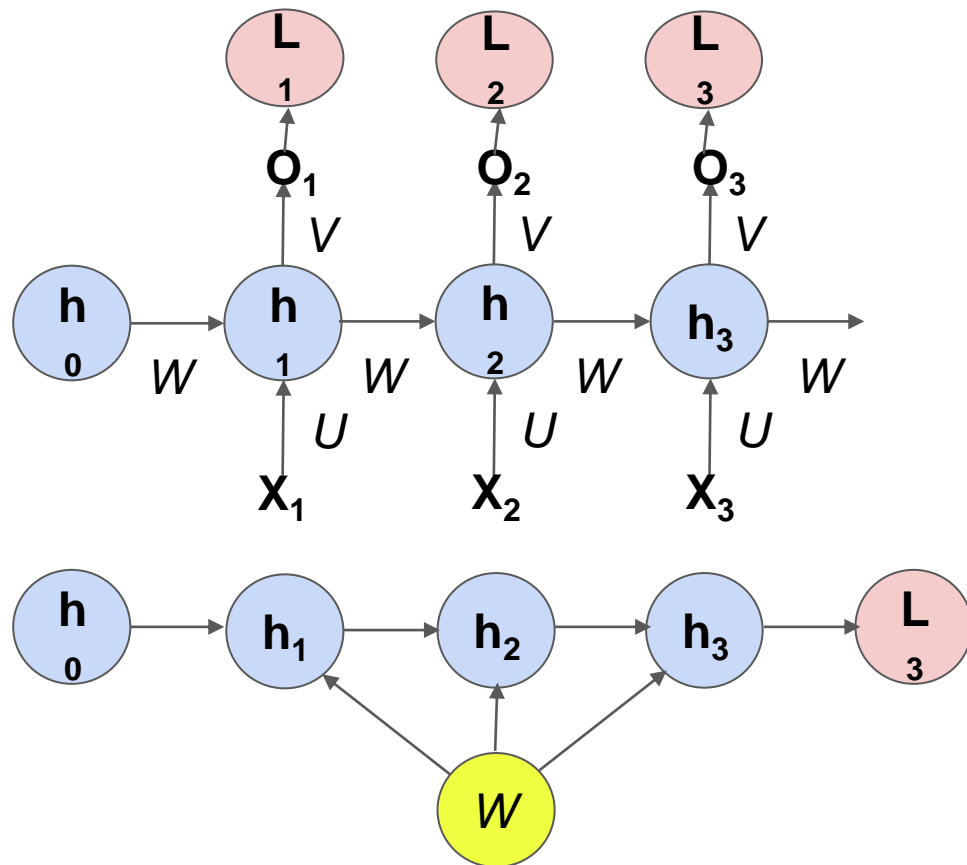
Training RNNs

$$\frac{\partial \mathbb{L}_3}{\partial W}$$



- $\mathbb{L}_3(\theta)$ depends on h_3
- h_3 in-turn depends on h_2 and W
- h_2 in-turn depends on h_1 and W
- h_1 depends on h_0 and W and h_0 is a constant vector (start token)

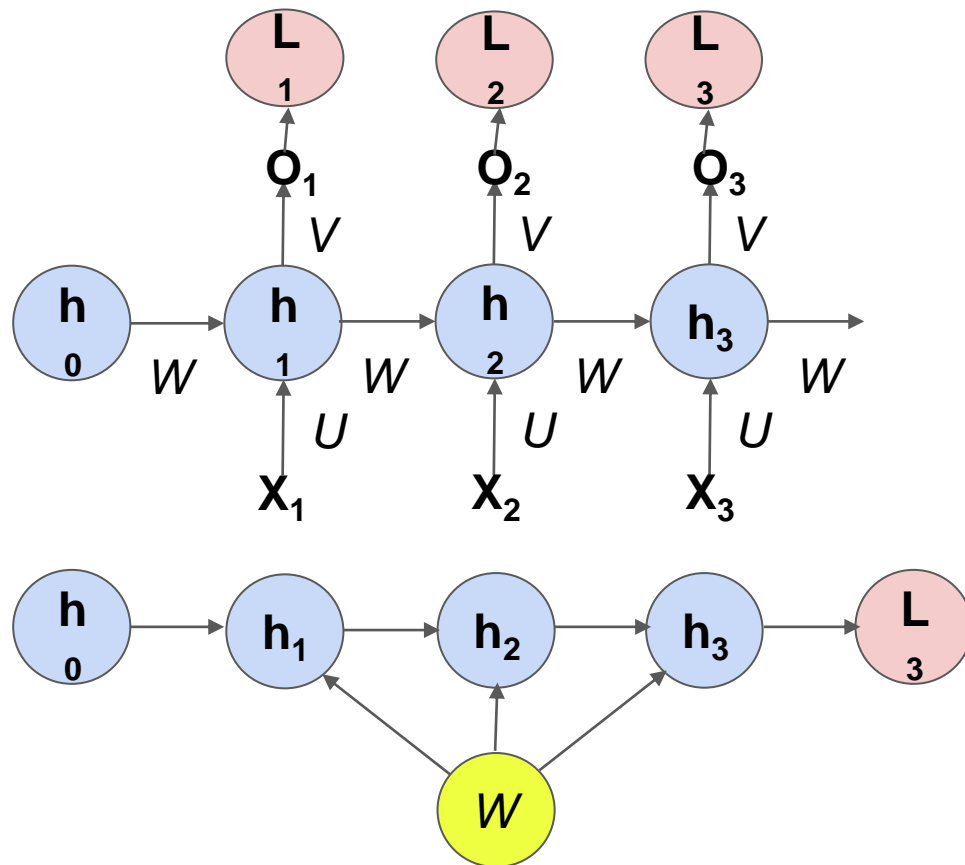
Training RNNs



- By back propagation we get,

$$\frac{\partial \mathbb{L}_3(\theta)}{\partial W} = \frac{\partial \mathbb{L}_3(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$

Training RNNs

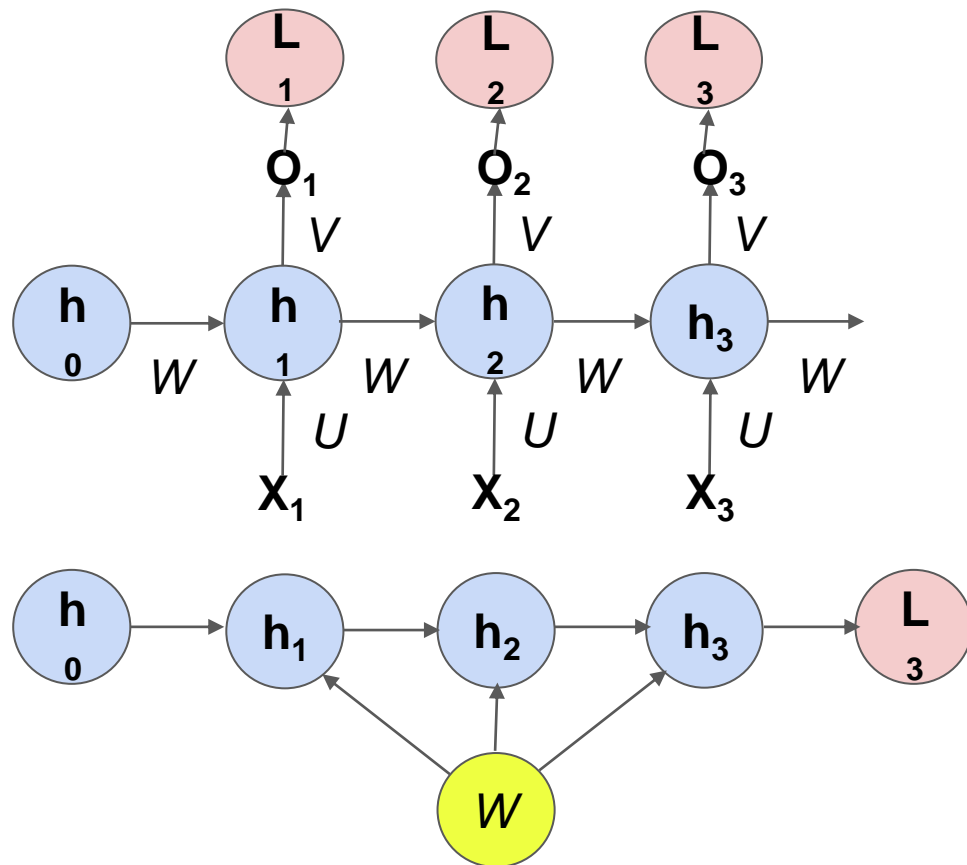


- By back propagation we get,

$$\frac{\partial \mathbb{L}_3(\theta)}{\partial W} = \frac{\partial \mathbb{L}_3(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$

- We already know how to compute, $\frac{\partial \mathbb{L}_3(\theta)}{\partial h_3}$

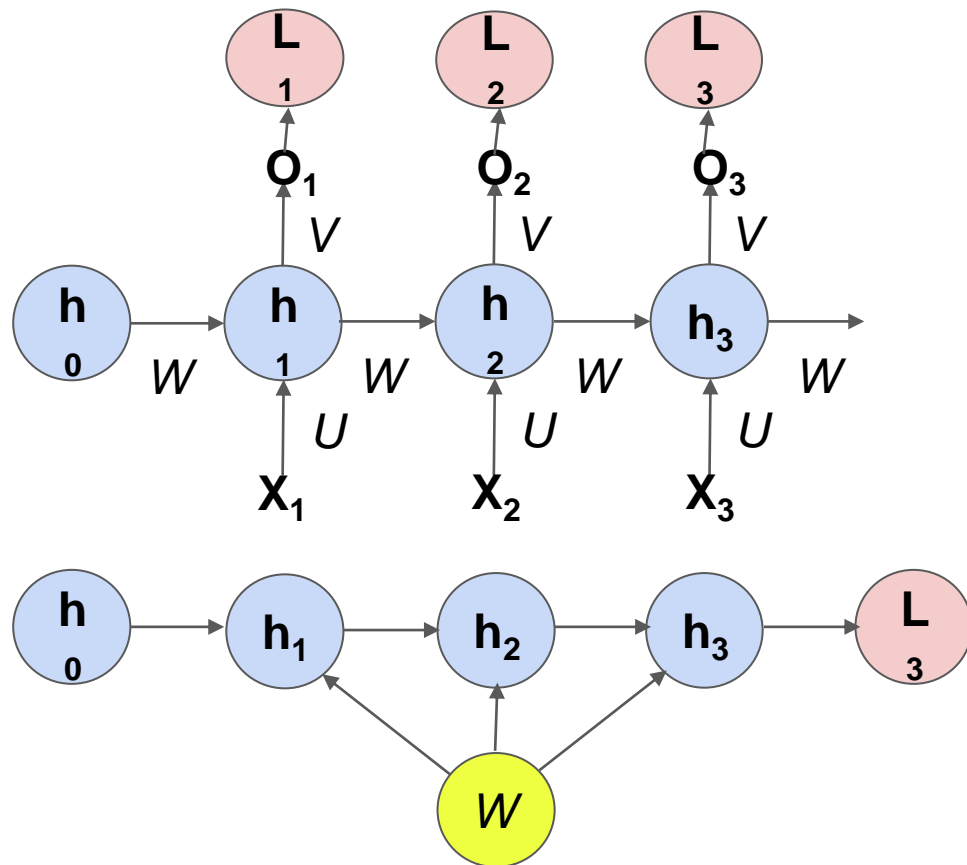
Training RNNs



- By back propagation we get,

$$\frac{\partial \mathbb{L}_3(\theta)}{\partial W} = \frac{\partial \mathbb{L}_3(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$
- We already know how to compute, $\frac{\partial \mathbb{L}_3(\theta)}{\partial h_3}$
- But how do we compute, $\frac{\partial h_3}{\partial W}$

Training RNNs

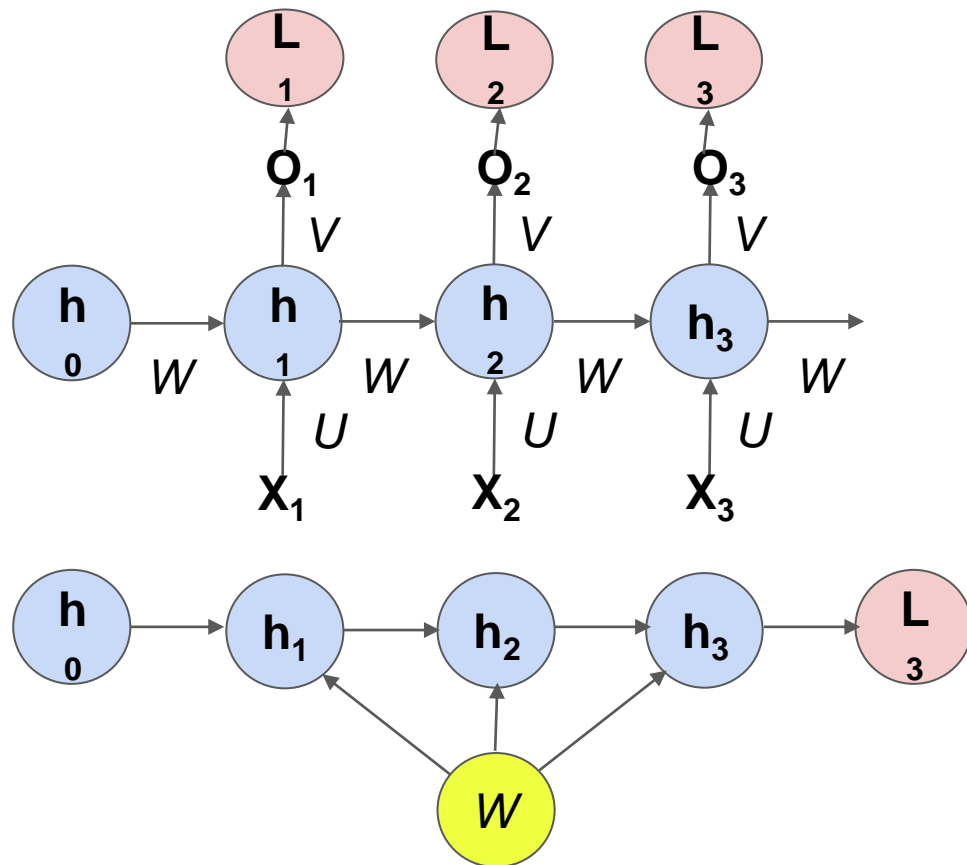


- Recall that,

$$h_t = f(W h_{t-1} + U x_t)$$

$$h_3 = f(W h_2 + U x_3)$$

Training RNNs

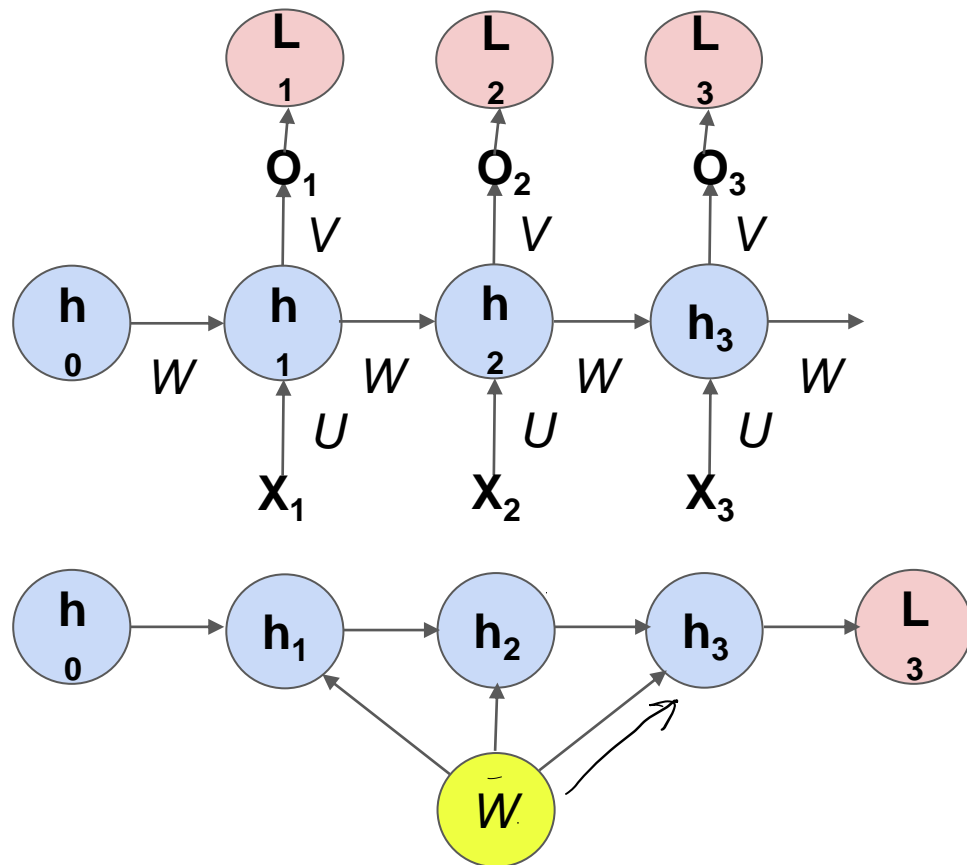


- Recall that,

$$h_t = f(W h_{t-1} + U x_t)$$

$$h_3 = f(\underline{W} \underline{h_2} + U x_3)$$
- Expanding $\frac{\partial h_3}{\partial \underline{W}}$, we get two parts because $\underline{h_2}$ also depends on \underline{W}

Training RNNs



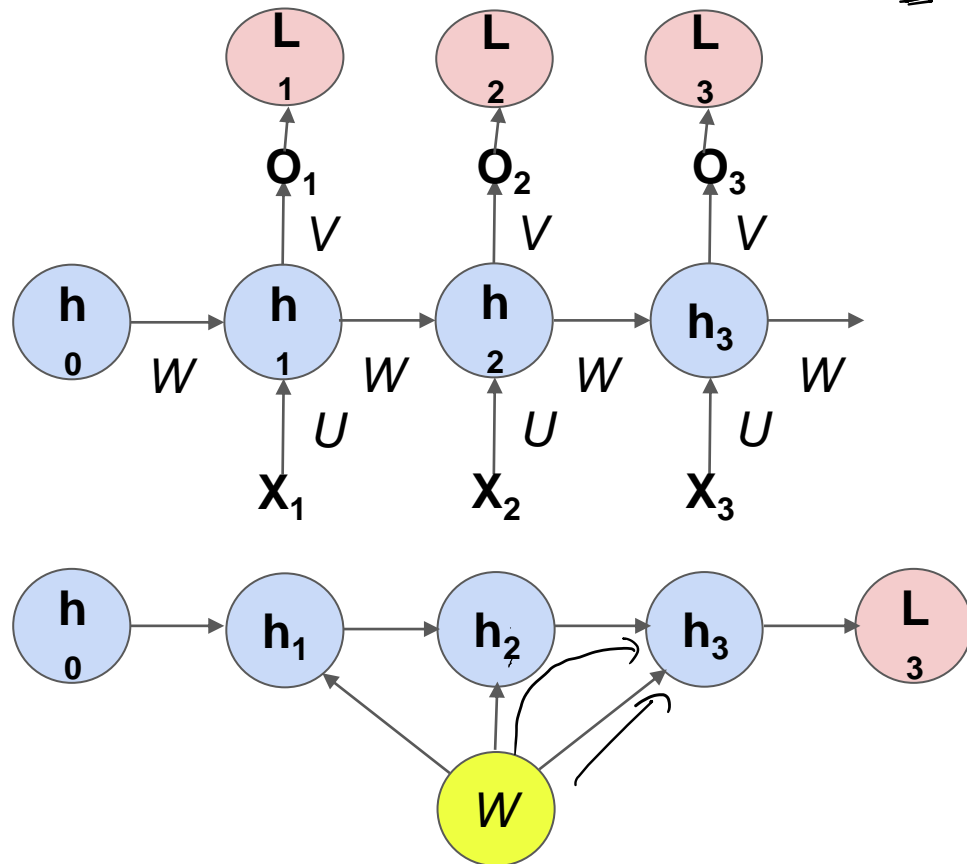
- Recall that,

$$h_t = f(W h_{t-1} + U x_t)$$

$$h_3 = f(W \underline{h_2} + U \underline{x_3})$$
- Expanding $\frac{\partial h_3}{\partial W}$, we get two parts because $\underline{h_2}$ also depends on \underline{W}
- Explicit:** $\frac{\partial^+ h_3}{\partial W}$, treating all other inputs as $\overline{\text{constant}}$

Training RNNs

$$\frac{\partial h_3}{\partial W} = \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial W}$$



- Recall that,

$$h_t = f(W h_{t-1} + U x_t)$$

$$h_3 = f(W h_2 + U x_3)$$
- Expanding $\frac{\partial h_3}{\partial W}$, we get two parts because h_2 also depends on W
- Explicit:** $\frac{\partial^+ h_3}{\partial W}$, treating all other inputs as constant
- Implicit:** Summing over all the indirect paths from h_3 to W

Training RNNs

- Therefore,

$$\frac{\partial h_3}{\partial \underline{W}} = \frac{\partial^+ h_3}{\partial \underline{W}} + \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial \underline{W}}$$

$$h_2 = f\left(\underline{W} h_1 + \cancel{U x_2}\right)$$
$$\frac{\partial h_2}{\partial \underline{W}} = \frac{\partial^+ h_2}{\partial \underline{W}} + \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial \underline{W}}$$

Training RNNs

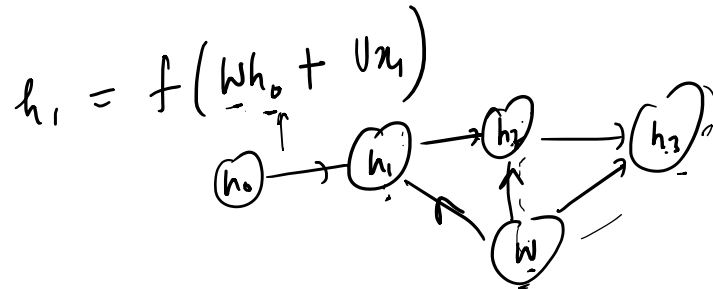
- Therefore,

$$\begin{aligned}\frac{\partial h_3}{\partial W} &= \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial W} \\ &= \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2} \left[\frac{\partial^+ h_2}{\partial W} + \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W} \right]\end{aligned}$$

Training RNNs

- Therefore,

$$\begin{aligned}
 \frac{\partial h_3}{\partial W} &= \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial W} \\
 &\sim \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2} \left[\frac{\partial^+ h_2}{\partial W} + \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W} \right] \\
 &= \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial^+ h_2}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial^+ h_1}{\partial W} \quad (\text{How?})
 \end{aligned}$$



Training RNNs

- Therefore,

$$\begin{aligned}\frac{\partial h_3}{\partial W} &= \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial W} \\ &= \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2} \left[\frac{\partial^+ h_2}{\partial W} + \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W} \right] \\ &= \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial^+ h_2}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial^+ h_1}{\partial W} \quad \left(\text{Since } \frac{\partial h_1}{\partial W} = \frac{\partial^+ h_1}{\partial W} \right)\end{aligned}$$

Training RNNs

- Therefore,

$$\frac{\partial h_3}{\partial W} = \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial W}$$

$$= \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2} \left[\frac{\partial^+ h_2}{\partial W} + \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W} \right]$$

$$= \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial^+ h_2}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial^+ h_1}{\partial W} \quad \left(\text{Since } \frac{\partial h_1}{\partial W} = \frac{\partial^+ h_1}{\partial W} \right)$$

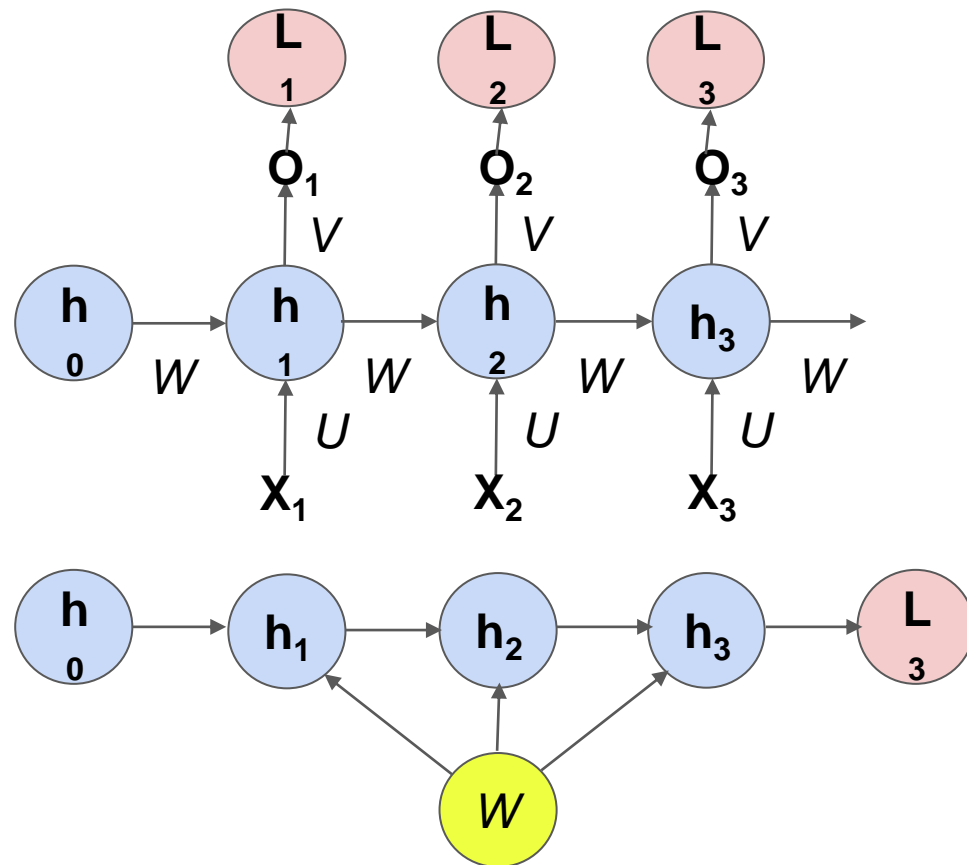
- Simplifying,

$$\frac{\partial h_3}{\partial W} = \frac{\partial h_3}{\partial h_3} \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial^+ h_2}{\partial W} + \frac{\partial h_3}{\partial h_1} \frac{\partial^+ h_1}{\partial W} = \left(\sum_{k=1}^3 \frac{\partial h_3}{\partial h_k} \frac{\partial^+ h_k}{\partial W} \right)$$

$$h_t \rightarrow h_{t-1} \rightarrow h_{t-2} \dots h_1$$

$$\frac{\partial h_t}{\partial W} = \sum_{k=1}^t \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial W}$$

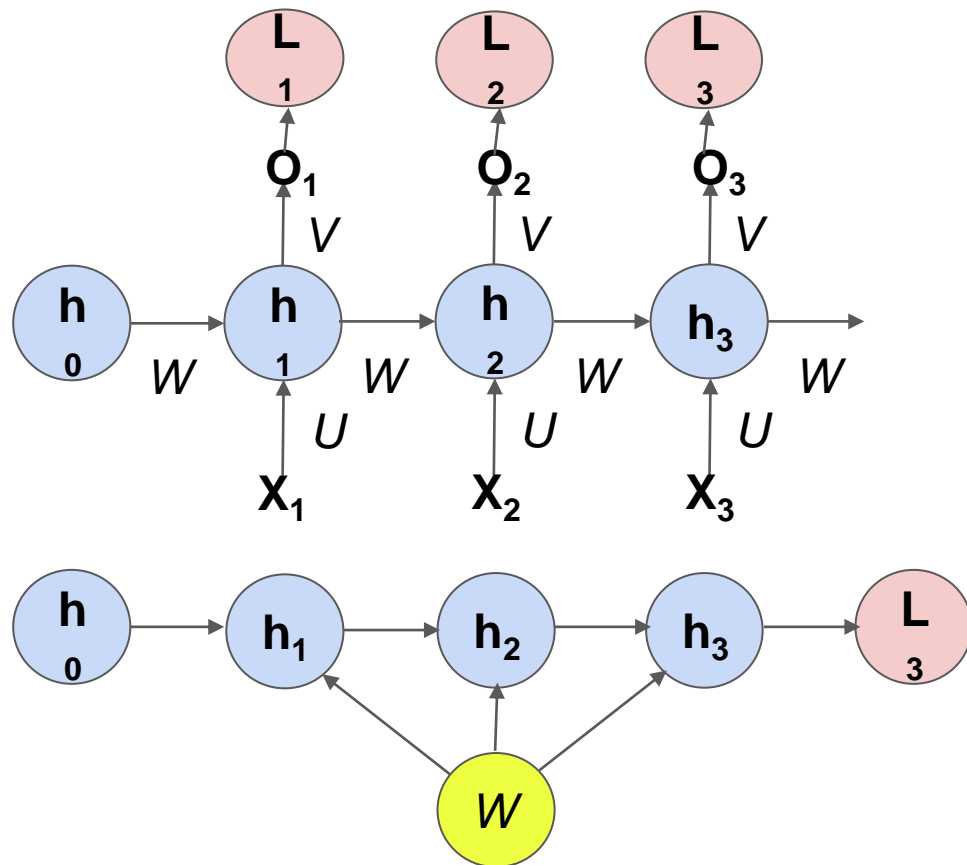
Training RNNs



- Finally,

$$\frac{\partial \mathbb{L}_3(\theta)}{\partial W} = \frac{\partial \mathbb{L}_3(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$

Training RNNs

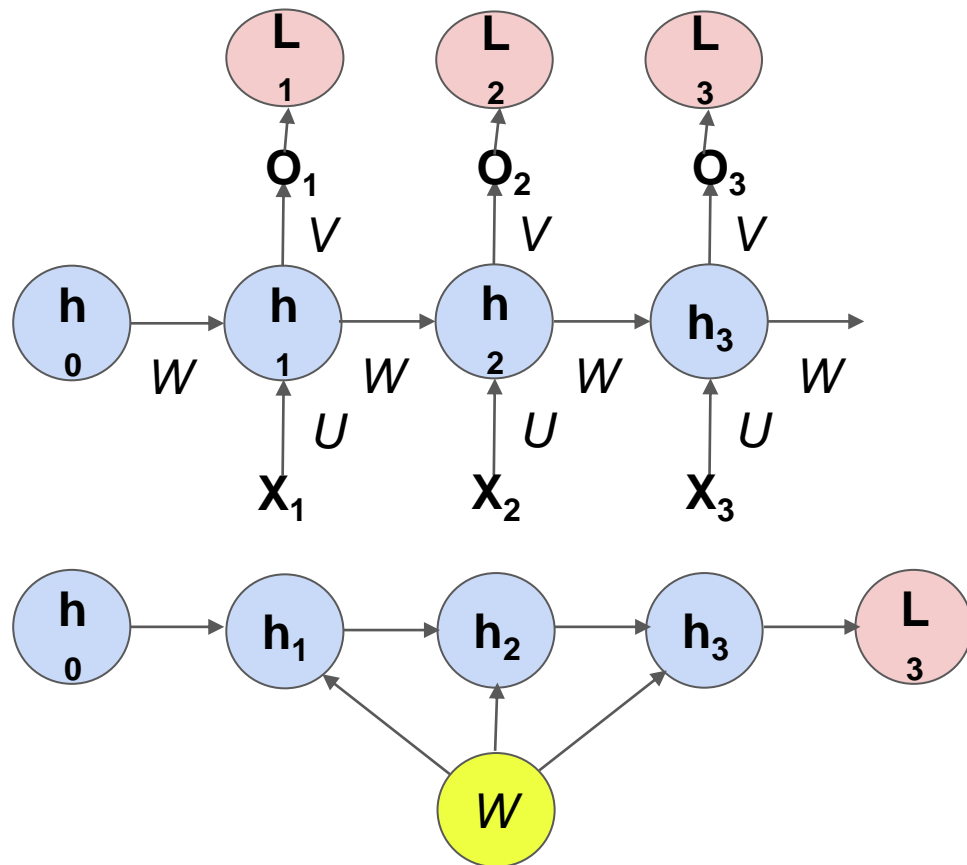


- Finally,

$$\frac{\partial \mathcal{L}_3(\theta)}{\partial W} = \frac{\partial \mathcal{L}_3(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$

$$\frac{\partial h_3}{\partial W} = \sum_{k=1}^3 \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W}$$

Training RNNs



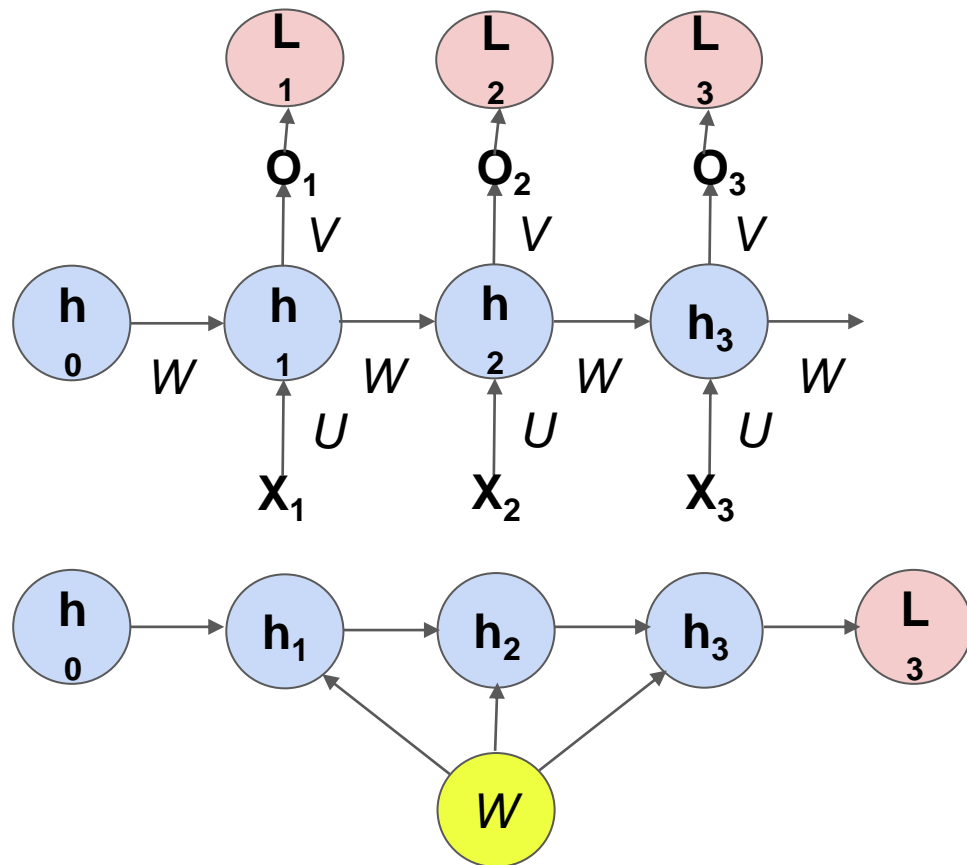
- Finally,

$$\frac{\partial \mathbb{L}_3(\theta)}{\partial W} = \frac{\partial \mathbb{L}_3(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$

$$\frac{\partial h_3}{\partial W} = \sum_{k=1}^3 \frac{\partial h_3}{\partial h_k} \frac{\partial^+ h_k}{\partial W}$$

$$\frac{\partial \mathbb{L}_3(\theta)}{\partial W} = \frac{\partial \mathbb{L}_3(\theta)}{\partial h_3} \sum_{k=1}^3 \frac{\partial h_3}{\partial h_k} \frac{\partial^+ h_k}{\partial W}$$

Training RNNs



- Finally,

$$\frac{\partial \mathbb{L}_3(\theta)}{\partial W} = \frac{\partial \mathbb{L}_3(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$

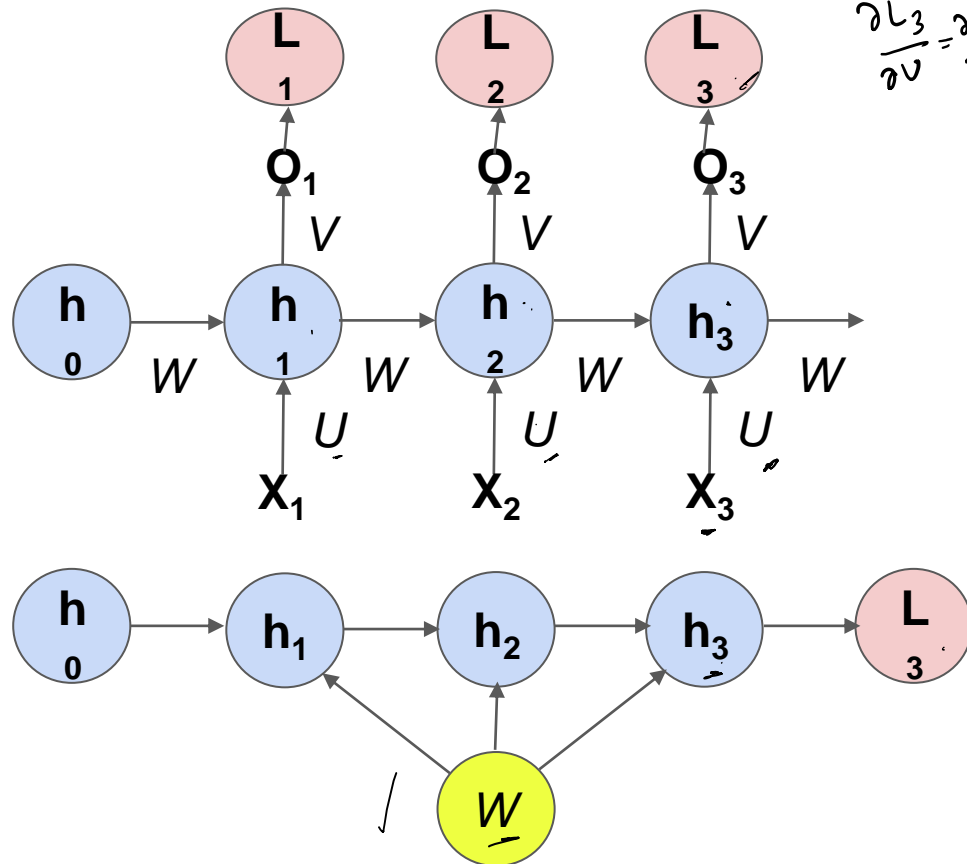
$$\frac{\partial h_3}{\partial W} = \sum_{k=1}^3 \frac{\partial h_3}{\partial h_k} \frac{\partial^+ h_k}{\partial W}$$

$$\frac{\partial \mathbb{L}_3(\theta)}{\partial W} = \frac{\partial \mathbb{L}_3(\theta)}{\partial h_3} \sum_{k=1}^3 \frac{\partial h_3}{\partial h_k} \frac{\partial^+ h_k}{\partial W}$$

$$\therefore \frac{\partial \mathbb{L}_t(\theta)}{\partial W} = \frac{\partial \mathbb{L}_t(\theta)}{\partial h_t} \underbrace{\sum_{k=1}^t \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial W}}_{\frac{\partial h_t}{\partial W}}$$

Training RNNs

$$\frac{\partial \mathcal{L}}{\partial u}, \frac{\partial \mathcal{L}}{\partial W}, \frac{\partial \mathcal{L}}{\partial v}$$



• Finally,

$$\frac{\partial \mathcal{L}_3}{\partial u} = \frac{\partial \mathcal{L}_3}{\partial o_3} \frac{\partial o_3}{\partial v} \frac{\partial v}{\partial u}$$

$$\frac{\partial \mathcal{L}_3(\theta)}{\partial W} = \frac{\partial \mathcal{L}_3(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$

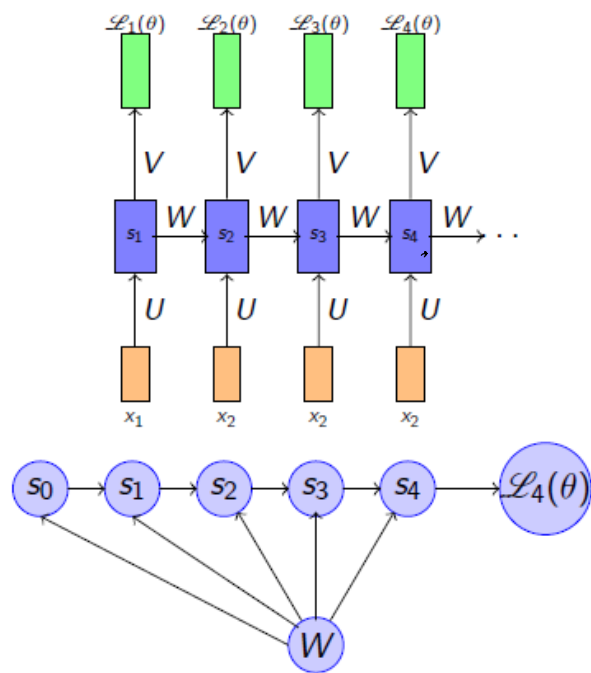
$$\frac{\partial h_3}{\partial W} = \sum_{k=1}^3 \frac{\partial h_3}{\partial h_k} \frac{\partial^+ h_k}{\partial W}$$

$$\frac{\partial \mathcal{L}_3(\theta)}{\partial W} = \frac{\partial \mathcal{L}_3(\theta)}{\partial h_3} \sum_{k=1}^3 \frac{\partial h_3}{\partial h_k} \frac{\partial^+ h_k}{\partial W}$$

$$\therefore \frac{\partial \mathcal{L}_t(\theta)}{\partial W} = \frac{\partial \mathcal{L}_t(\theta)}{\partial h_t} \sum_{k=1}^t \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial W}$$

- This algorithm is called **backpropagation through time (BPTT)** as we backpropagate over all previous time steps

Vanishing Gradients



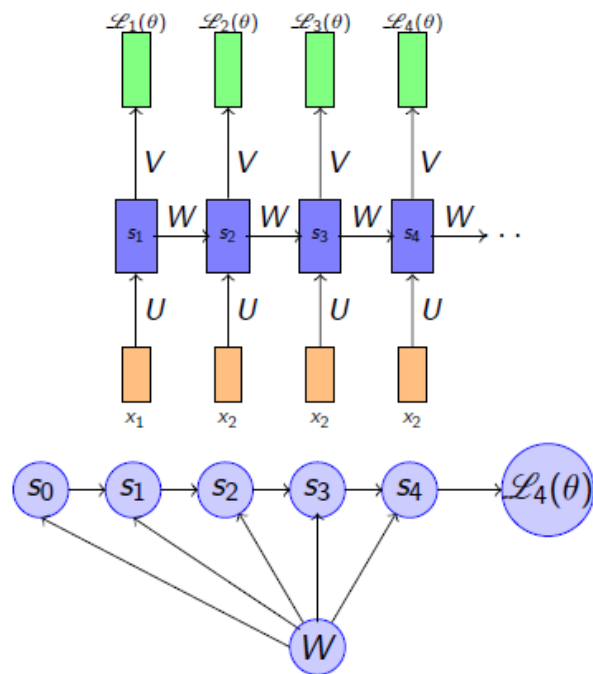
- Finally we have

$$\begin{aligned} \frac{\partial \mathcal{L}_4(\theta)}{\partial W} &= \frac{\partial \mathcal{L}_4(\theta)}{\partial s_4} \frac{\partial s_4}{\partial W} \\ \frac{\partial s_4}{\partial W} &= \sum_{k=1}^4 \frac{\partial s_4}{\partial s_k} \frac{\partial s_k}{\partial W} \\ \therefore \frac{\partial \mathcal{L}_t(\theta)}{\partial W} &= \frac{\partial \mathcal{L}_t(\theta)}{\partial s_t} \sum_{k=1}^t \frac{\partial s_t}{\partial s_k} \frac{\partial s_k}{\partial W} \end{aligned}$$

$\frac{\partial s_t}{\partial W}$

- This algorithm is called backpropagation through time (BPTT) as we back-propagate over all previous time steps

Vanishing Gradients



- We will now focus on $\frac{\partial s_t}{\partial s_k}$ and highlight an important problem in training RNN's using BPTT

$$\frac{\partial s_t}{\partial s_k} = \frac{\partial s_t}{\partial s_{t-1}} \frac{\partial s_{t-1}}{\partial s_{t-2}} \cdots \frac{\partial s_{k+1}}{\partial s_k} \quad (t-k) \text{ terms}$$

$$= \prod_{j=k}^{t-1} \frac{\partial s_{j+1}}{\partial s_j}$$

- Let us look at one such term in the product (i.e. $\frac{\partial s_{j+1}}{\partial s_j}$)

Vanishing Gradients

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \dots, a_{jd}]_d$$

$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \dots, \sigma(a_{jd})]_d$$

$$\frac{\partial s_j}{\partial a_j} = \begin{bmatrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{bmatrix}_{d \times 1}$$

$$s_j = \sigma(Ws_{j-1} + Ua_j)$$

$$\rightarrow a_j = Ws_{j-1} + Ua_j$$

$$\rightarrow s_j = \sigma(a_j)$$

- We are interested in $\frac{\partial s_j}{\partial s_{j-1}}$

$$a_j = Ws_{j-1} + b$$

$$s_j = \sigma(a_j)$$

$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}}$$

Vanishing Gradients

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \dots, a_{jd}]$$

$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \dots, \sigma(a_{jd})]$$

$$\frac{\partial s_j}{\partial a_j} = \begin{bmatrix} \frac{\partial s_{j1}}{\partial a_{j1}} & \frac{\partial s_{j2}}{\partial a_{j1}} & \frac{\partial s_{j3}}{\partial a_{j1}} & \dots \\ \frac{\partial s_{j1}}{\partial a_{j2}} & \frac{\partial s_{j2}}{\partial a_{j2}} & \dots & \\ \vdots & \vdots & \vdots & \frac{\partial s_{jd}}{\partial a_{jd}} \end{bmatrix}^T$$

$d \times d$

- We are interested in $\frac{\partial s_j}{\partial s_{j-1}}$

$$a_j = W s_{j-1} + b$$

$$s_j = \sigma(a_j)$$

$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}}$$

Vanishing Gradients

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \dots, a_{jd},]$$

$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \dots, \sigma(a_{jd})]$$

$$\begin{aligned} \frac{\partial s_j}{\partial a_j} &= \begin{bmatrix} \frac{\partial s_{j1}}{\partial a_{j1}} & \frac{\partial s_{j2}}{\partial a_{j1}} & \frac{\partial s_{j3}}{\partial a_{j1}} & \dots \\ \frac{\partial s_{j1}}{\partial a_{j2}} & \frac{\partial s_{j2}}{\partial a_{j2}} & \ddots & \\ \vdots & \vdots & \vdots & \frac{\partial s_{jd}}{\partial a_{jd}} \end{bmatrix}^T \\ &= \begin{bmatrix} \sigma'(a_{j1}) & 0 & 0 & 0 \end{bmatrix} \end{aligned}$$

- We are interested in $\frac{\partial s_j}{\partial s_{j-1}}$

$$a_j = W s_{j-1} + b$$

$$s_j = \sigma(a_j)$$

$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}}$$

Vanishing Gradients

$$\begin{pmatrix} 1 \\ a_j \end{pmatrix} \approx \begin{pmatrix} \text{---} \\ W \end{pmatrix} \begin{pmatrix} s_{j-1} \end{pmatrix}$$

We are interested in $\frac{\partial s_j}{\partial s_{j-1}}$

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \dots, a_{jd}]$$

$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \dots, \sigma(a_{jd})]$$

$$a_{j1} = W_{11}s_{j-1} + W_{12}$$

$$a_j = Ws_{j-1} + b$$

$$s_j = \sigma(a_j)$$

$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}}$$

$$\frac{\partial a_j}{\partial s_{j-1}} = \begin{pmatrix} \frac{\partial a_{j1}}{\partial s_{j-1}} & \frac{\partial a_{j2}}{\partial s_{j-1}} & \dots \end{pmatrix} = W$$

$$\begin{aligned} \frac{\partial s_j}{\partial a_j} &= \begin{bmatrix} \frac{\partial s_{j1}}{\partial a_{j1}} & \frac{\partial s_{j2}}{\partial a_{j1}} & \frac{\partial s_{j3}}{\partial a_{j1}} & \dots \\ \frac{\partial s_{j1}}{\partial a_{j2}} & \frac{\partial s_{j2}}{\partial a_{j2}} & \dots & \\ \vdots & \vdots & \vdots & \frac{\partial s_{jd}}{\partial a_{jd}} \end{bmatrix}^T \\ &= \begin{bmatrix} \sigma'(a_{j1}) & 0 & 0 & 0 \\ 0 & \sigma'(a_{j2}) & 0 & 0 \\ 0 & 0 & \ddots & \\ 0 & 0 & \dots & \sigma'(a_{jd}) \end{bmatrix} \end{aligned}$$

Vanishing Gradients

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \dots, a_{jd},]$$

$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \dots, \sigma(a_{jd})]$$

$$\begin{aligned}\frac{\partial s_j}{\partial a_j} &= \begin{bmatrix} \frac{\partial s_{j1}}{\partial a_{j1}} & \frac{\partial s_{j2}}{\partial a_{j1}} & \frac{\partial s_{j3}}{\partial a_{j1}} & \dots \\ \frac{\partial s_{j1}}{\partial a_{j2}} & \frac{\partial s_{j2}}{\partial a_{j2}} & \ddots & \\ \vdots & \vdots & \vdots & \frac{\partial s_{jd}}{\partial a_{jd}} \end{bmatrix}^T \\ &= \begin{bmatrix} \sigma'(a_{j1}) & 0 & 0 & 0 \\ 0 & \sigma'(a_{j2}) & 0 & 0 \\ 0 & 0 & \ddots & \\ 0 & 0 & \dots & \sigma'(a_{jd}) \end{bmatrix} \\ &= \text{diag}(\sigma'(a_j))\end{aligned}$$

- We are interested in $\frac{\partial s_j}{\partial s_{j-1}}$

$$a_j = Ws_{j-1} + b$$

$$s_j = \sigma(a_j)$$

$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}}$$

Vanishing Gradients

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \dots, a_{jd},]$$

$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \dots, \sigma(a_{jd})]$$

$$\begin{aligned} \frac{\partial s_j}{\partial a_j} &= \begin{bmatrix} \frac{\partial s_{j1}}{\partial a_{j1}} & \frac{\partial s_{j2}}{\partial a_{j1}} & \frac{\partial s_{j3}}{\partial a_{j1}} & \dots \\ \frac{\partial s_{j1}}{\partial a_{j2}} & \frac{\partial s_{j2}}{\partial a_{j2}} & \ddots & \\ \vdots & \vdots & \vdots & \frac{\partial s_{jd}}{\partial a_{jd}} \end{bmatrix}^T \\ &= \begin{bmatrix} \sigma'(a_{j1}) & 0 & 0 & 0 \\ 0 & \sigma'(a_{j2}) & 0 & 0 \\ 0 & 0 & \ddots & \\ 0 & 0 & \dots & \sigma'(a_{jd}) \end{bmatrix} \\ &= \text{diag}(\sigma'(a_j)) \end{aligned}$$

- We are interested in $\frac{\partial s_j}{\partial s_{j-1}}$

$$a_j = Ws_{j-1} + b$$

$$s_j = \sigma(a_j)$$

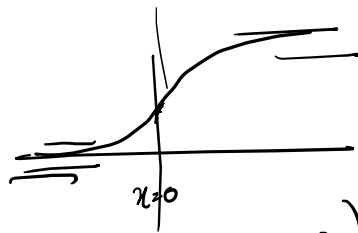
$$\begin{aligned} \frac{\partial s_j}{\partial s_{j-1}} &= \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}} \\ &= \underbrace{\text{diag}(\sigma'(a_j))}_{} W \end{aligned}$$

Vanishing Gradients

$$\frac{\partial s_t}{\partial s_k} = \frac{\partial s_t}{\partial s_{k+1}} \cdot \frac{\partial s_{k+1}}{\partial s_k} \rightarrow \text{diag}(\sigma'(a_{t-1})) \dots W^{(t-k)}$$

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \dots, a_{jd}]$$

$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \dots, \sigma(a_{jd})]$$



$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) = \frac{1}{2} \left(\frac{1}{2} \right) = \frac{1}{4}$$

$$\left(\frac{1}{4} \right)^0 \rightarrow 0$$

- We are interested in $\frac{\partial s_j}{\partial s_{j-1}}$

$$a_j = W s_{j-1} + b$$

$$s_j = \sigma(a_j)$$

$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}} = \text{diag}(\sigma'(a_j)) W$$

- We are interested in the magnitude of $\frac{\partial s_j}{\partial s_{j-1}} \leftarrow$ if it is small (large) $\frac{\partial s_t}{\partial s_k}$ and hence $\frac{\partial L_t}{\partial W}$ will vanish (explode)

$$\begin{aligned} \frac{\partial s_j}{\partial a_j} &= \begin{bmatrix} \frac{\partial s_{j1}}{\partial a_{j1}} & \frac{\partial s_{j2}}{\partial a_{j1}} & \frac{\partial s_{j3}}{\partial a_{j1}} & \dots \\ \frac{\partial s_{j1}}{\partial a_{j2}} & \frac{\partial s_{j2}}{\partial a_{j2}} & \dots & \\ \vdots & \vdots & \vdots & \frac{\partial s_{jd}}{\partial a_{jd}} \end{bmatrix}^T \\ &= \begin{bmatrix} \sigma'(a_{j1}) & 0 & 0 & 0 \\ 0 & \sigma'(a_{j2}) & 0 & 0 \\ 0 & 0 & \ddots & \\ 0 & 0 & \dots & \sigma'(a_{jd}) \end{bmatrix} \\ &= \text{diag}(\sigma'(a_j)) \end{aligned}$$

Vanishing Gradients

Induced norm $\|A\|_p = \max_v \frac{\|Av\|_p}{\|v\|_p}$

$$\|A\|_2 = \max_v \frac{\|Av\|_2}{\|v\|_2} = \max_i \lambda_i(A)$$

$$\left\| \frac{\partial s_j}{\partial s_{j-1}} \right\| = \left\| \text{diag}(\sigma'(a_j)) W \right\|$$

$$\|A\| = \sum_{ij} |a_{ij}|$$

$$\|A\| = \sqrt{\sum_{ij} a_{ij}^2} \rightarrow \text{Frobenius norm}$$

1.) $\|A\| \geq 0$

2.) $\|A+B\| \leq \|A\| + \|B\|$

3.) $\|sA\| = |s| \|A\|$

4.) $\|A\| = 0 \Leftrightarrow A = 0$

5.) $\|AB\| \leq \|A\| \|B\|$

$$\left\| \frac{\partial s_t}{\partial s_k} \right\| = \left\| \prod_{j=k+1}^t \frac{\partial s_j}{\partial s_{j-1}} \right\|$$

Vanishing Gradients

$$\begin{aligned}\left\|\frac{\partial s_j}{\partial s_{j-1}}\right\| &= \|\text{diag}(\sigma'(a_j))W\| \\ &\leq \|\text{diag}(\sigma'(a_j))\| \|W\|\end{aligned}$$

$$\left\|\frac{\partial s_t}{\partial s_k}\right\| = \left\|\prod_{j=k+1}^t \frac{\partial s_j}{\partial s_{j-1}}\right\|$$

Vanishing Gradients

$$\begin{aligned}\left\|\frac{\partial s_j}{\partial s_{j-1}}\right\| &= \|\text{diag}(\sigma'(a_j))W\| \\ &\leq \|\text{diag}(\sigma'(a_j))\| \|W\|\end{aligned}$$

\therefore For the given functions (sigmoid, tanh) $\sigma'(a_j)$ is bounded

$$\left\|\frac{\partial s_t}{\partial s_k}\right\| = \left\|\prod_{j=k+1}^t \frac{\partial s_j}{\partial s_{j-1}}\right\|$$

Vanishing Gradients

$$\begin{aligned}\left\| \frac{\partial s_j}{\partial s_{j-1}} \right\| &= \| \text{diag}(\sigma'(a_j)) W \| \\ &\leq \| \text{diag}(\sigma'(a_j)) \| \| W \| \end{aligned}$$

\therefore For the given functions (sigmoid, tanh) $\sigma'(a_j)$ is bounded

$$\sigma'(a_j) = \frac{1}{4} = \gamma \text{ [if } \sigma \text{ is logistic]}$$

$$\left\| \frac{\partial s_t}{\partial s_k} \right\| = \left\| \prod_{j=k+1}^t \frac{\partial s_j}{\partial s_{j-1}} \right\|$$

Vanishing Gradients

$$\begin{aligned}\left\| \frac{\partial s_j}{\partial s_{j-1}} \right\| &= \| \text{diag}(\sigma'(a_j)) W \| \\ &\leq \| \text{diag}(\sigma'(a_j)) \| \| W \| \end{aligned}$$

\therefore For the given functions (sigmoid, tanh) $\sigma'(a_j)$ is bounded

$$\begin{aligned}\sigma'(a_j) &= \frac{1}{4} = \gamma \text{ [if } \sigma \text{ is logistic]} \\ &= 1 = \gamma \text{ [if } \sigma \text{ is tanh]} \end{aligned}$$

$$\left\| \frac{\partial s_t}{\partial s_k} \right\| = \left\| \prod_{j=k+1}^t \frac{\partial s_j}{\partial s_{j-1}} \right\|$$

Vanishing Gradients

$$\begin{aligned}\left\| \frac{\partial s_j}{\partial s_{j-1}} \right\| &= \| \text{diag}(\sigma'(a_j)) W \| \\ &\leq \| \text{diag}(\sigma'(a_j)) \| \| W \| \end{aligned}$$

\therefore For the given functions (sigmoid, tanh) $\sigma'(a_j)$ is bounded

$$\begin{aligned}\sigma'(a_j) &= \frac{1}{4} = \gamma \text{ [if } \sigma \text{ is logistic]} \\ &= 1 = \gamma \text{ [if } \sigma \text{ is tanh]} \end{aligned}$$

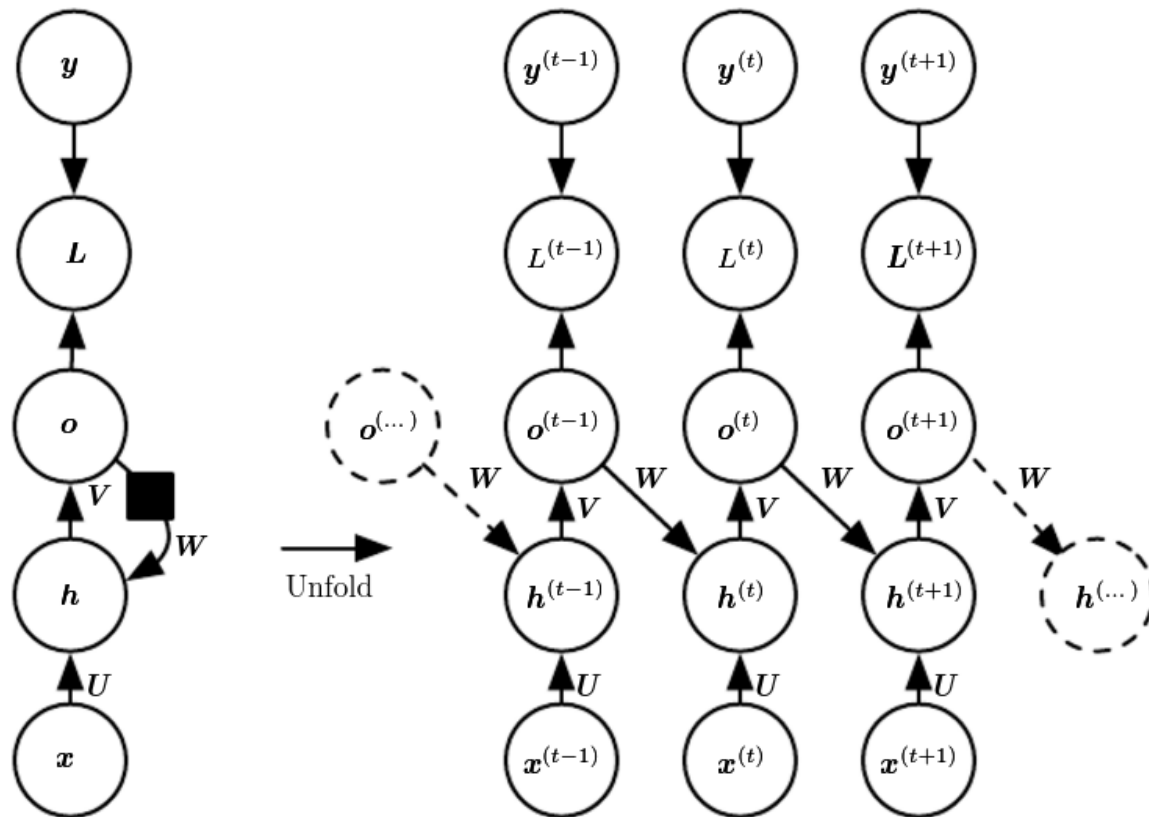
$$\begin{aligned}\left\| \frac{\partial s_j}{\partial s_{j-1}} \right\| &\leq \gamma \| W \| \\ &\leq \gamma \lambda \end{aligned}$$

$$\begin{aligned}\left\| \frac{\partial s_t}{\partial s_k} \right\| &= \left\| \prod_{j=k+1}^t \frac{\partial s_j}{\partial s_{j-1}} \right\| \\ &\leq \prod_{j=k+1}^t \gamma \lambda \\ &\leq (\gamma \lambda)^{t-k} \end{aligned}$$

- If $\gamma \lambda < 1$ the gradient will vanish
- If $\gamma \lambda > 1$ the gradient could explode
- This is known as the problem of vanishing/ exploding gradients

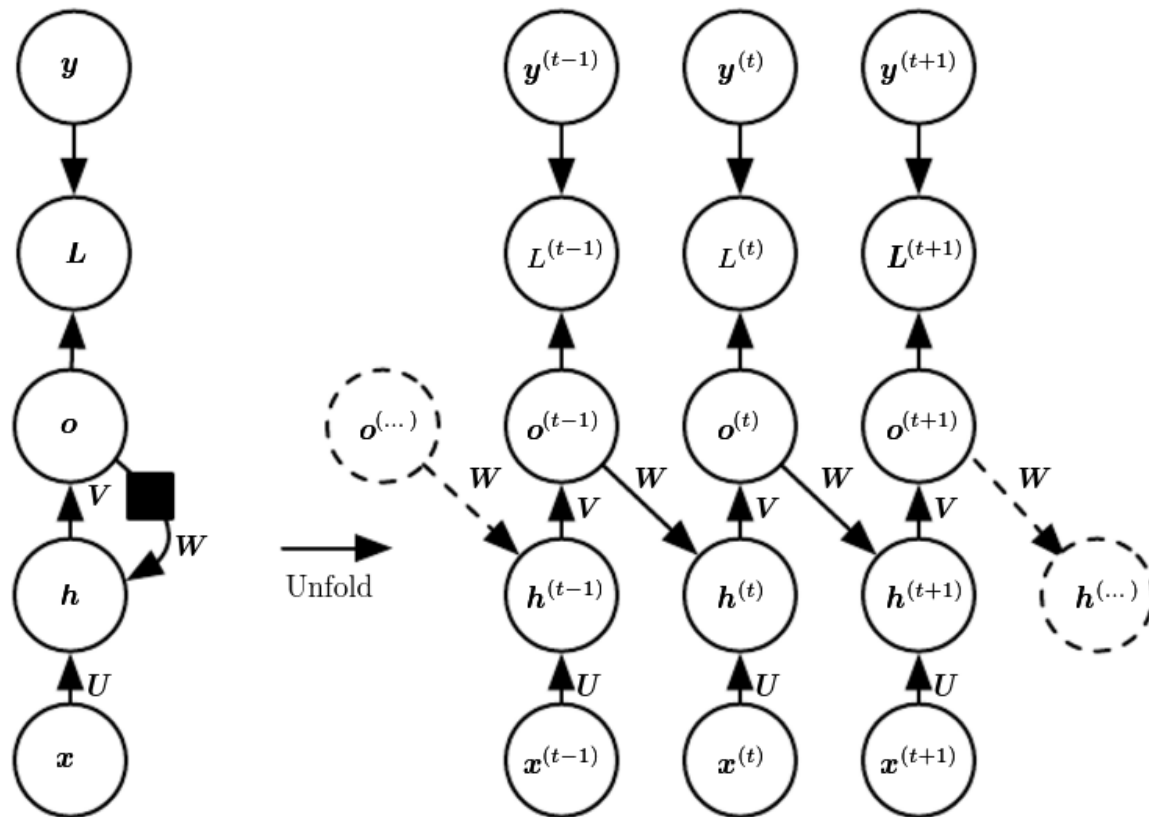
Variations of RNNs

Variations of RNNs (Output recurrence)



Network with recurrent connections only from the output at one time-step to the hidden units at the next time-step

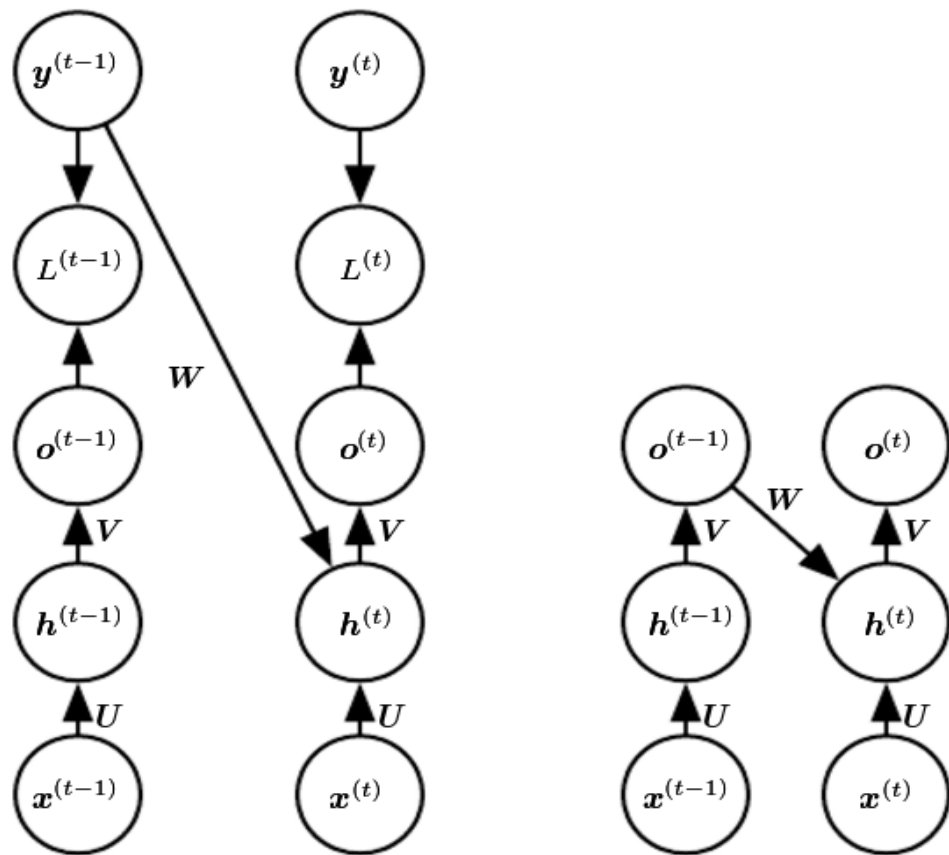
Variations of RNNs (Output recurrence)



Network with recurrent connections only from the output at one time-step to the hidden units at the next time-step

How to train these kind of networks? And what to do during testing?

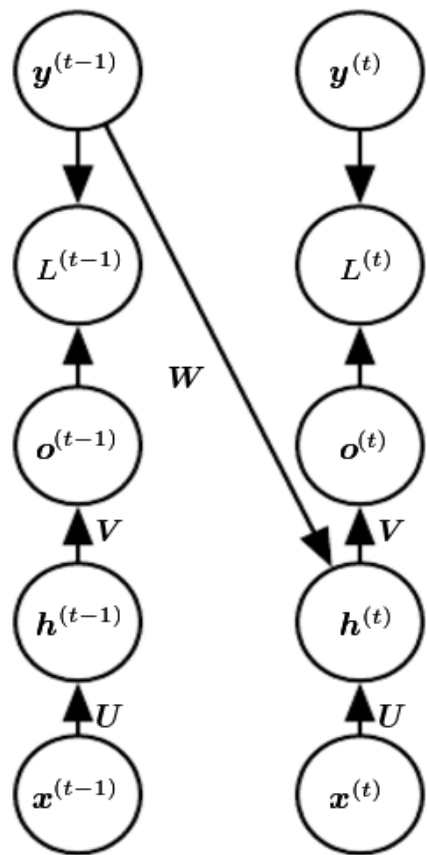
Output recurrence (or Teacher forcing)



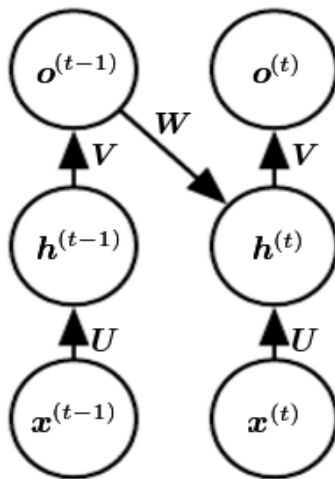
Train time

Test time

Output recurrence (or Teacher forcing)



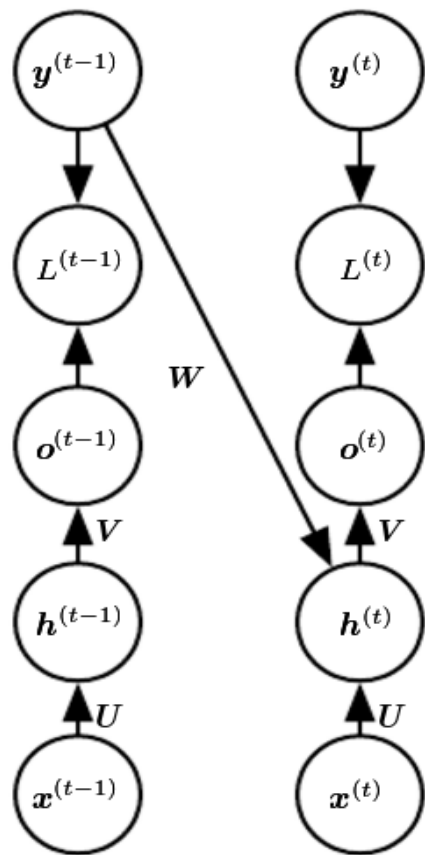
Train time



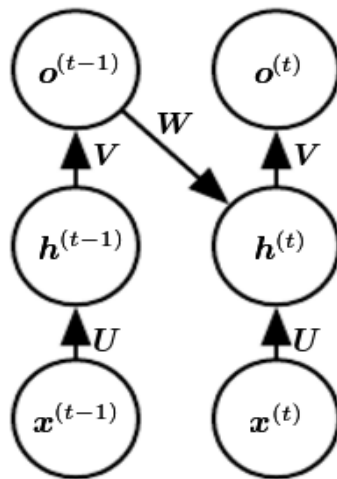
Test time

Can you write back propagation, and state equations for this network?

Output recurrence (or Teacher forcing)



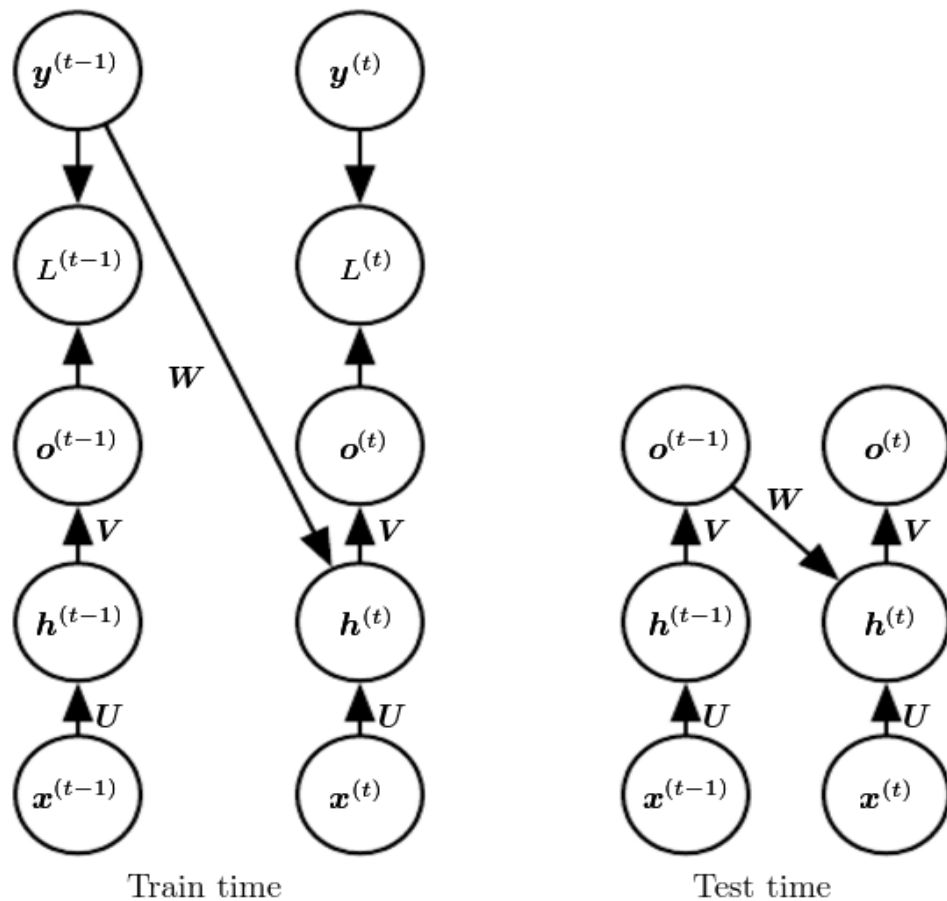
Train time



Test time

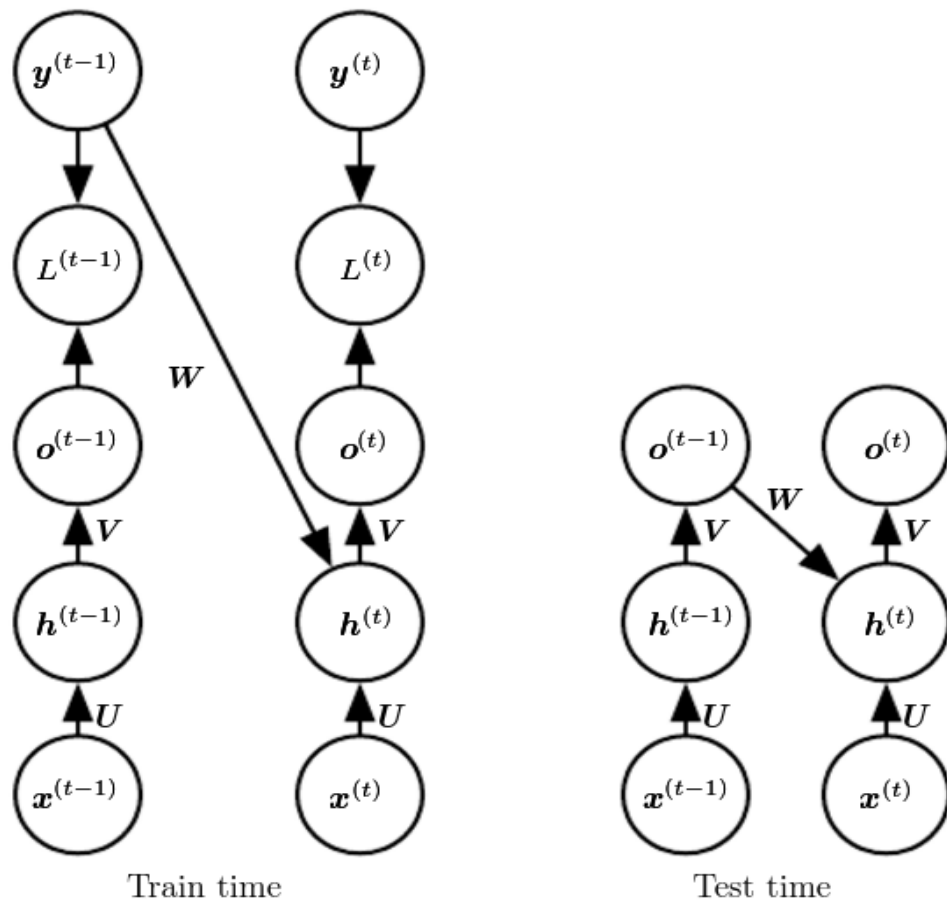
- These networks are strictly less powerful than the previous ones (Why?)

Output recurrence (or Teacher forcing)



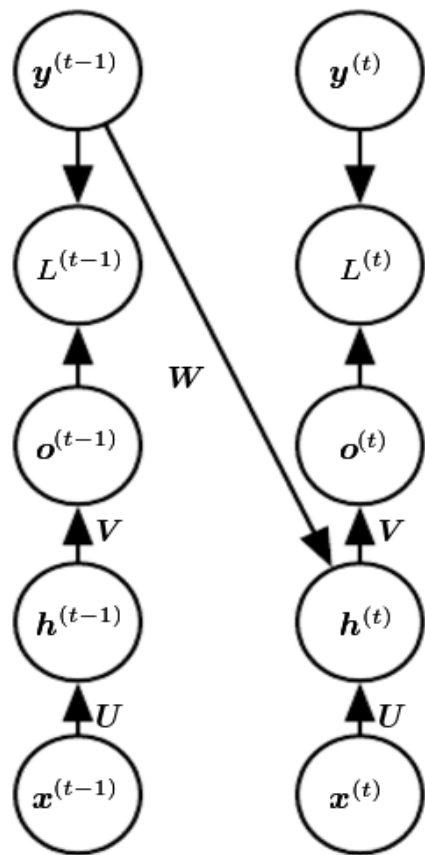
- These networks are strictly less powerful than the previous ones (Why?)
- This is since the network lacks hidden-to-hidden recurrence and it requires that the output units capture all the information about the past

Output recurrence (or Teacher forcing)

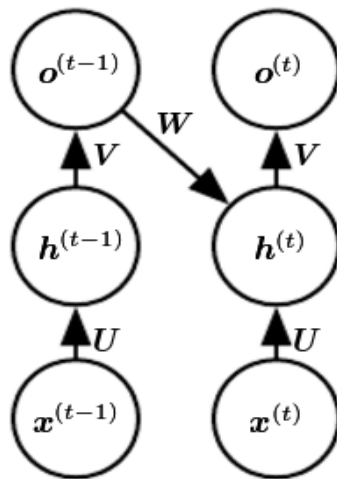


- These networks are strictly less powerful than the previous ones (Why?)
- This is since the network lacks hidden-to-hidden recurrence and it requires that the output units capture all the information about the past
- Since the outputs are explicitly trained to match the ground truth, this is very unlikely to happen

Output recurrence (or Teacher forcing)



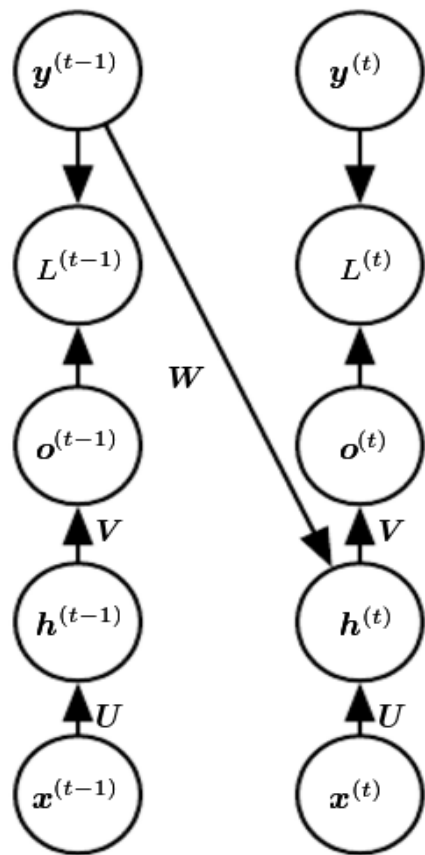
Train time



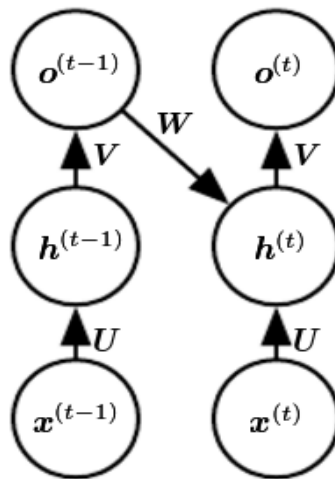
Test time

- So why do we study it?

Output recurrence (or Teacher forcing)



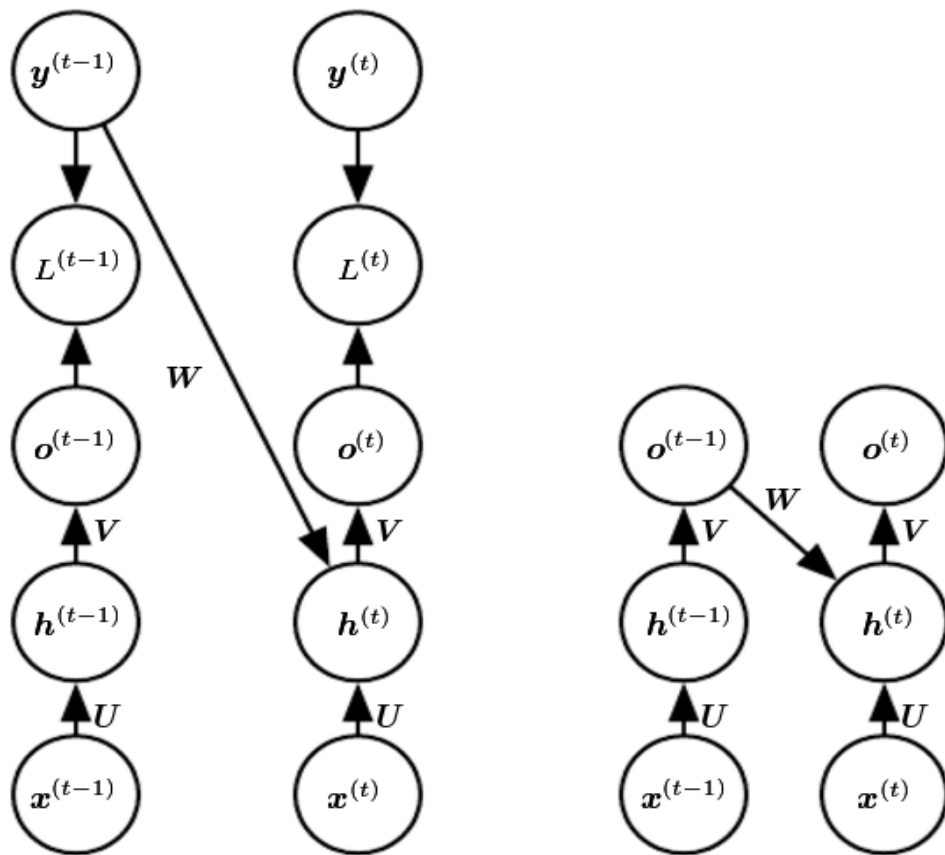
Train time



Test time

- So why do we study it?
- Because training is very simple (Why?)

Output recurrence (or Teacher forcing)



Train time

Test time

- So why do we study it?
- Because training is very simple (Why?)
- Training can be speeded up by completely parallelizing it since we do not need to wait for previous value (How?)

Next coming up...

- (i) Problems with RNNs
- (ii) Different types of RNNs
- (iii) Approximations in BPTT