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 $\{x_1, x_2, \ldots, x_t, \ldots, x_{n-1}, x_n\}$
    - Output sequence $\{y_1, y_2, \ldots, y_t, \ldots, y_{m-1}, y_m\}$

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

Sequence-to-sequence problems

- **Language translation**

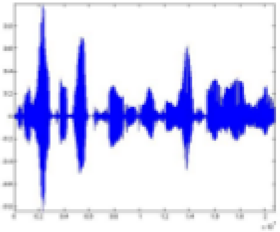
  Hello, how are you? ↔ नमस्ते आप कैसे हैं?

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”

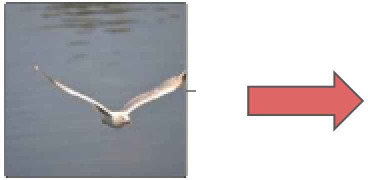
Sequence-to-sequence problems

- **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?

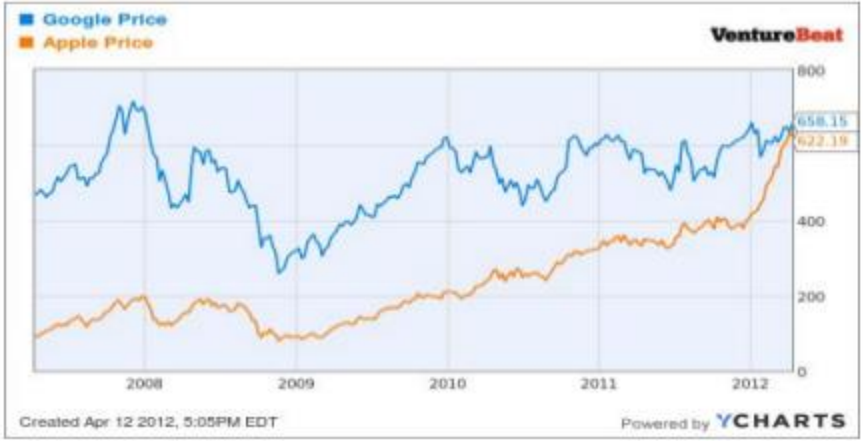
- **Image captioning**



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

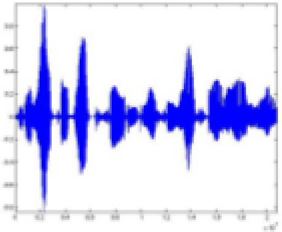


Sequence learning

- **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?

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

$$\Downarrow$$

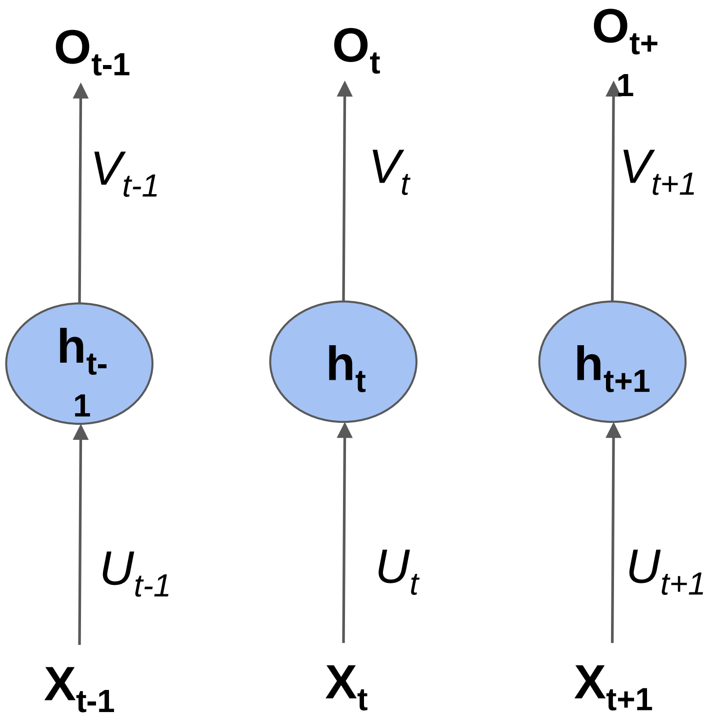
$$\{y_1, y_2, \ldots, y_t, \ldots, y_{m-1}, y_m\}$$

A naive approach

- Use multiple MLPs, one for each time-step

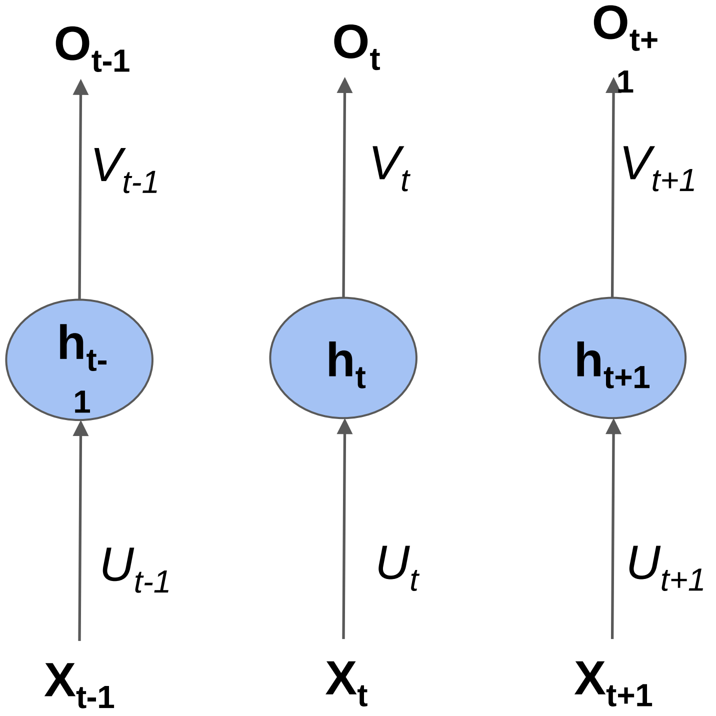
A naive approach

- Use multiple MLPs, one for each time-step



A naive approach

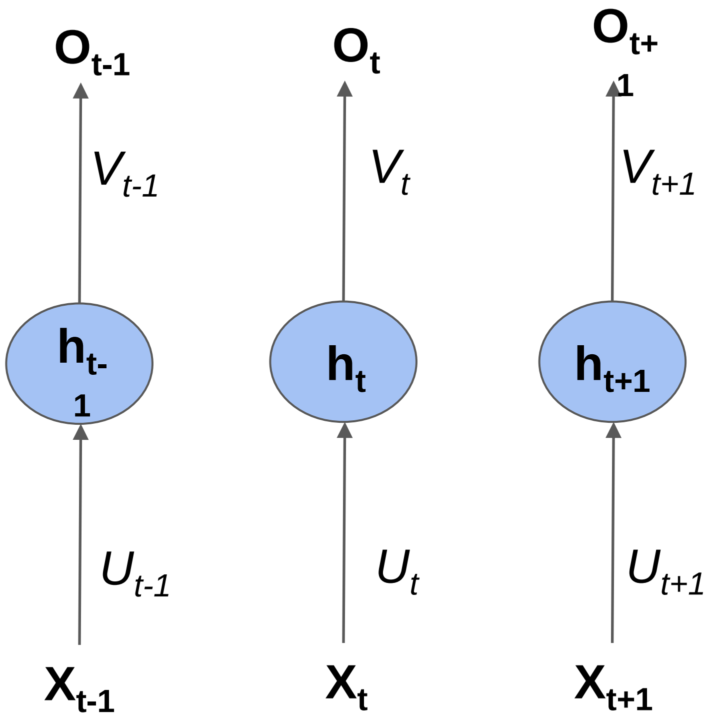
- Use multiple MLPs, one for each time-step



- Handles sequence?

A naive approach

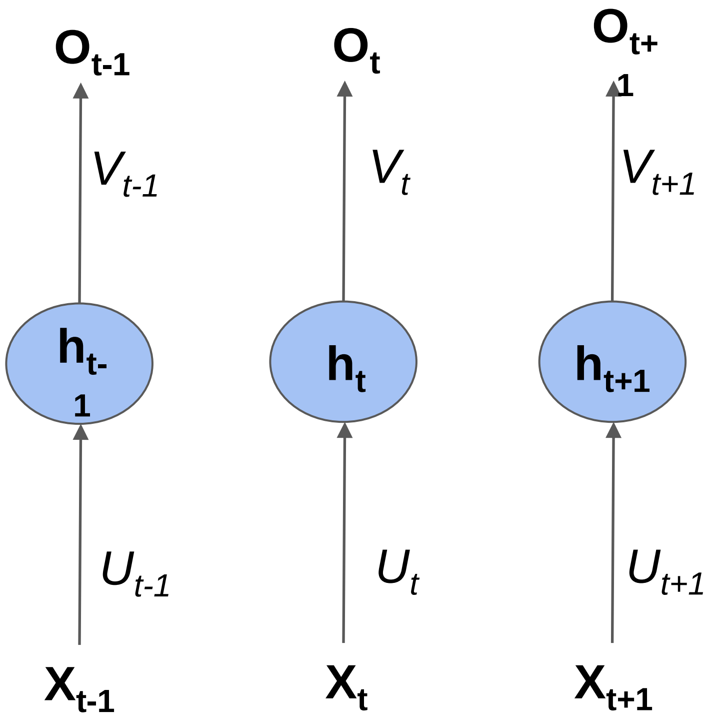
- Use multiple MLPs, one for each time-step



- Handles sequence? Yes ✓

A naive approach

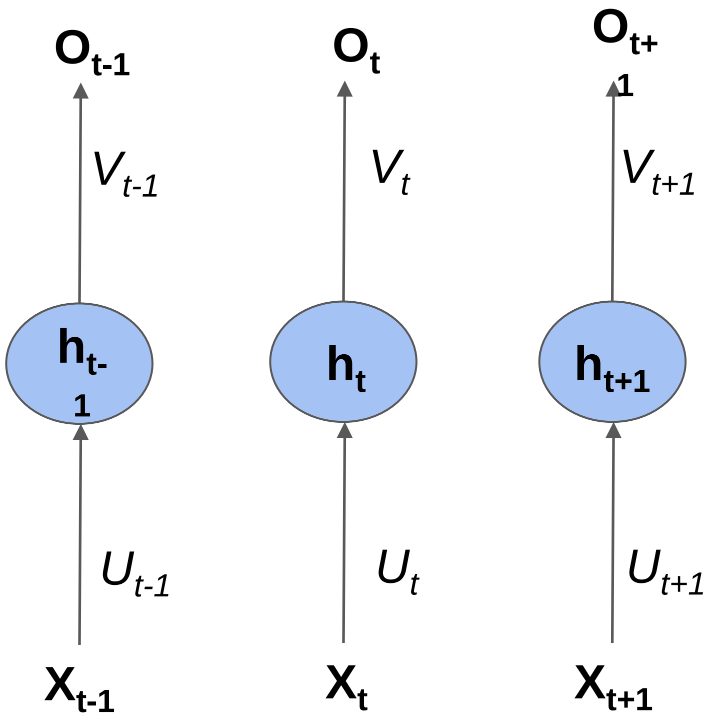
- Use multiple MLPs, one for each time-step



- Handles sequence? Yes ✓
- Models dependency?

A naive approach

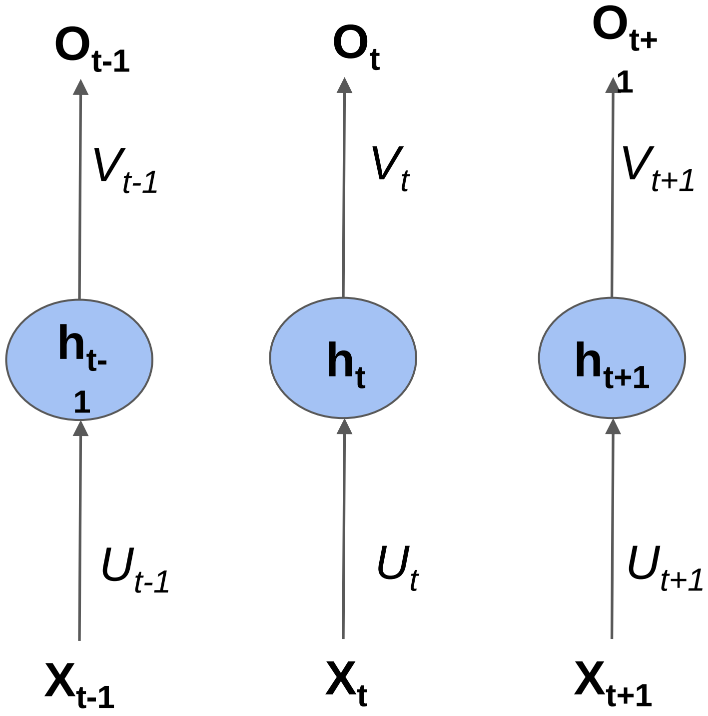
- Use multiple MLPs, one for each time-step



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

A naive approach

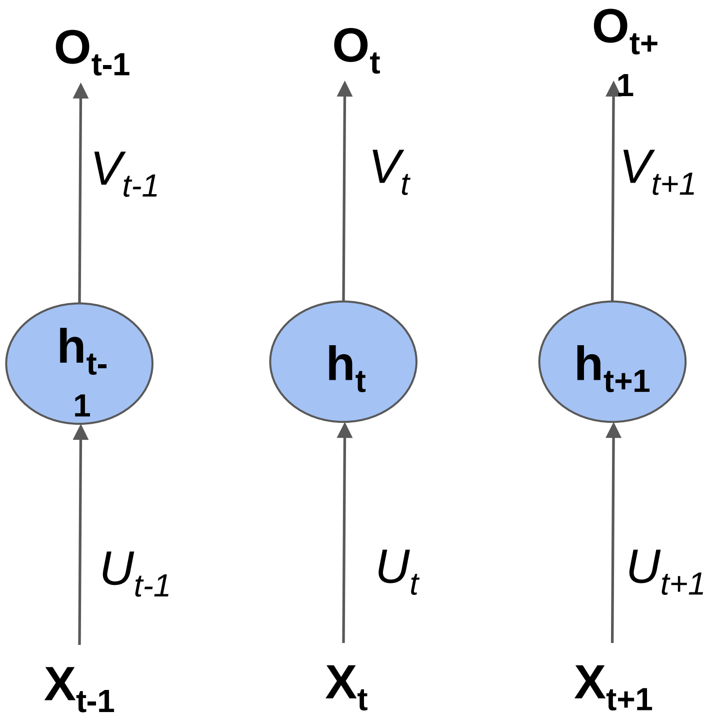
- Use multiple MLPs, one for each time-step



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

A naive approach

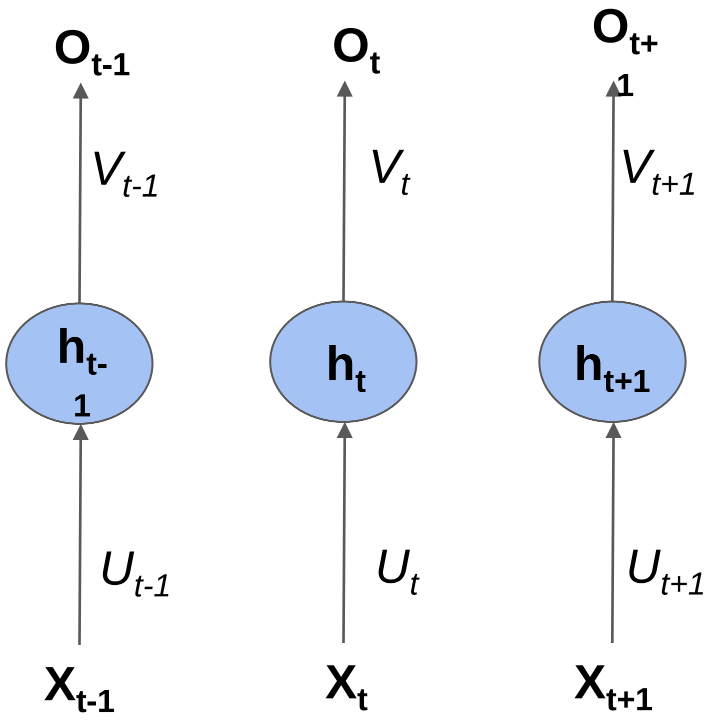
- Use multiple MLPs, one for each time-step



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

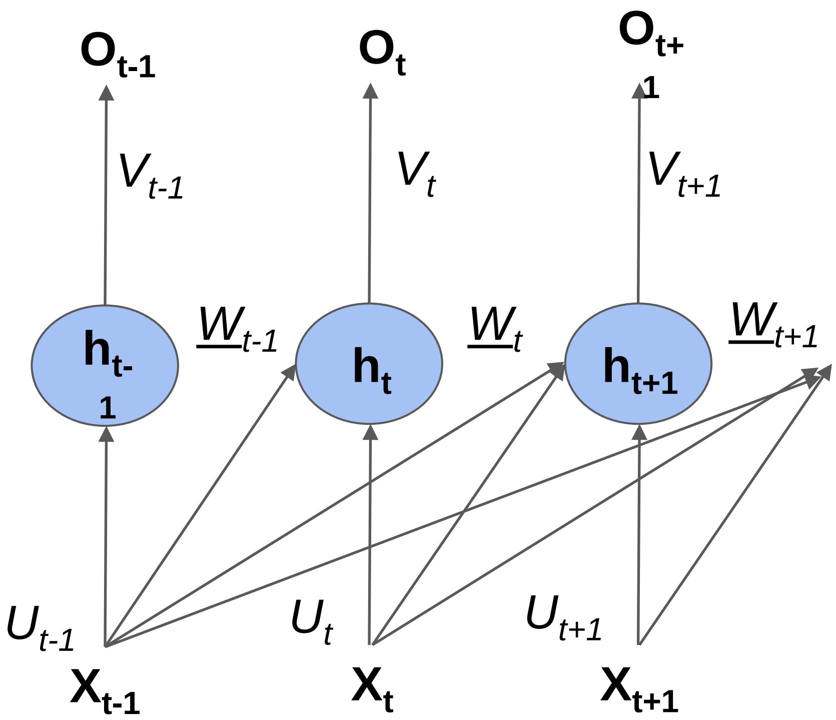
A naive approach

- Use multiple MLPs, one for each time-step

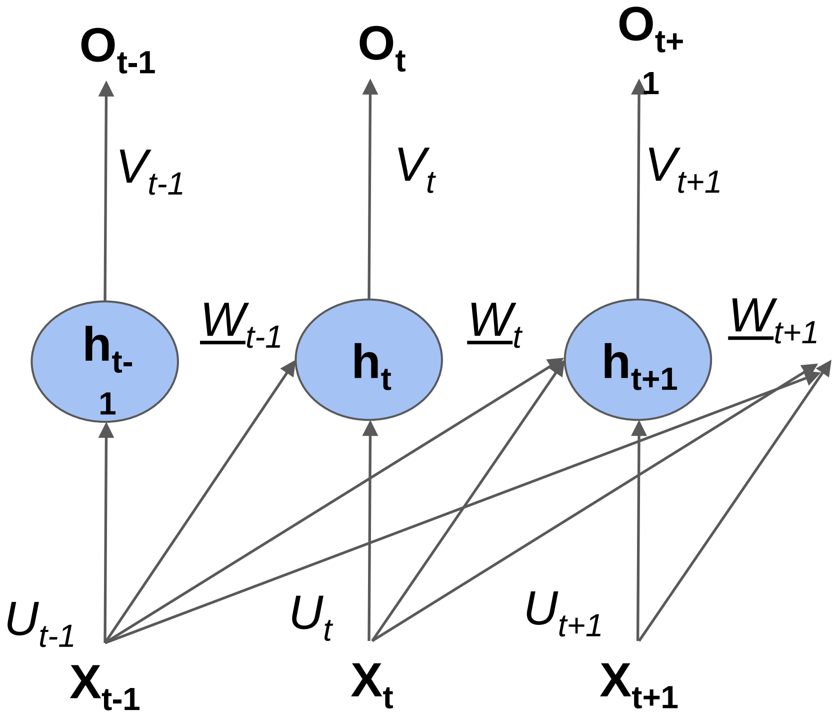


- Handles sequence? Yes ✓
- Models dependency? No ✗
- Variable length sequences? No ✗
- Many parameters to train :(✗

A revised architecture

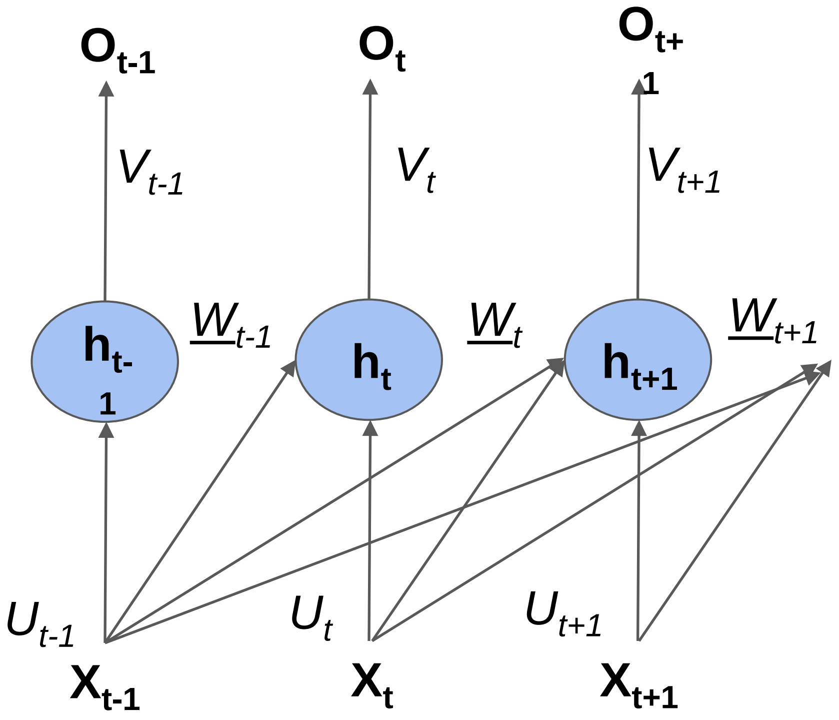


A revised architecture



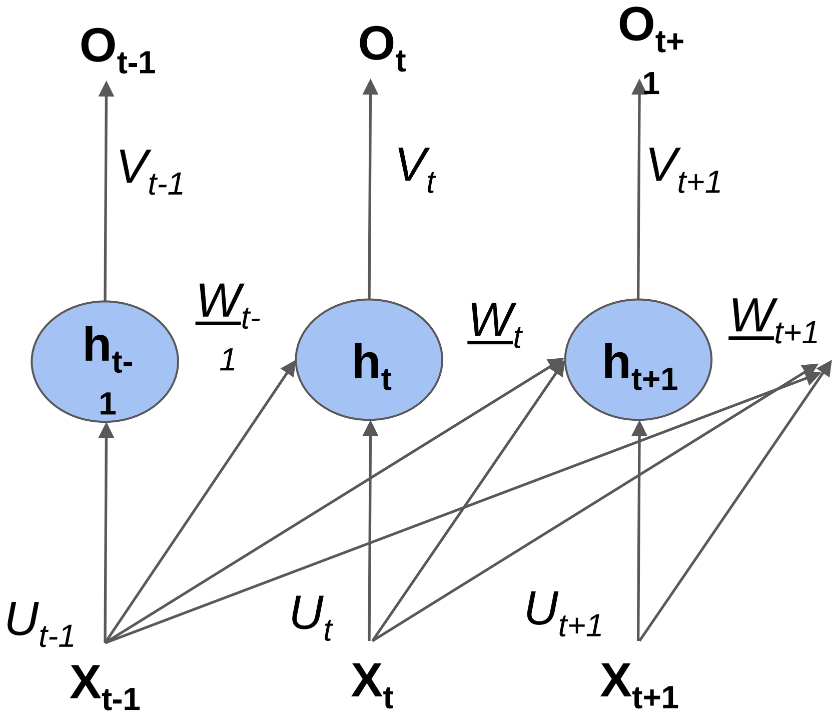
- Handles sequence?

A revised architecture



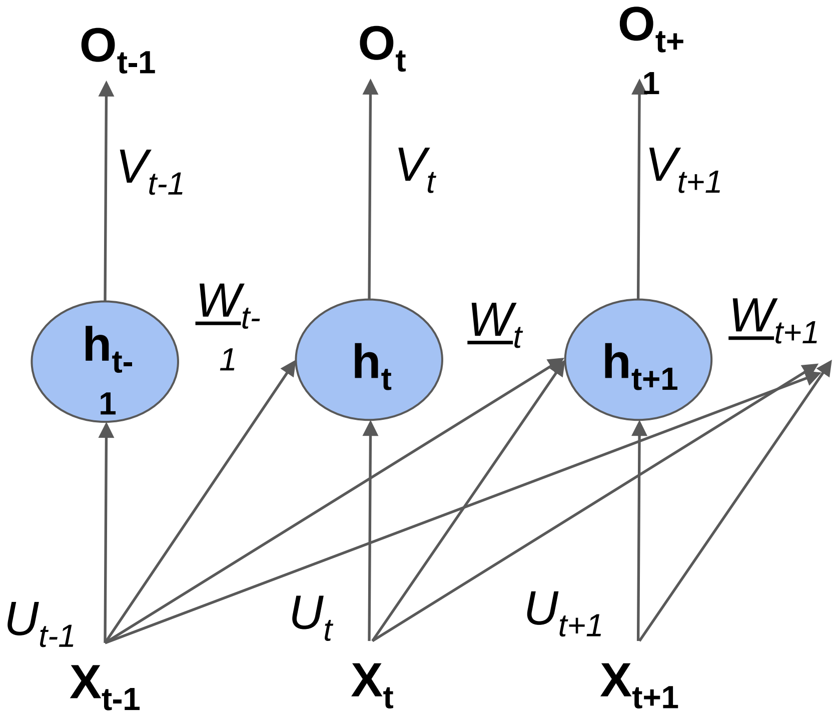
- Handles sequence? Yes ✓

A revised architecture



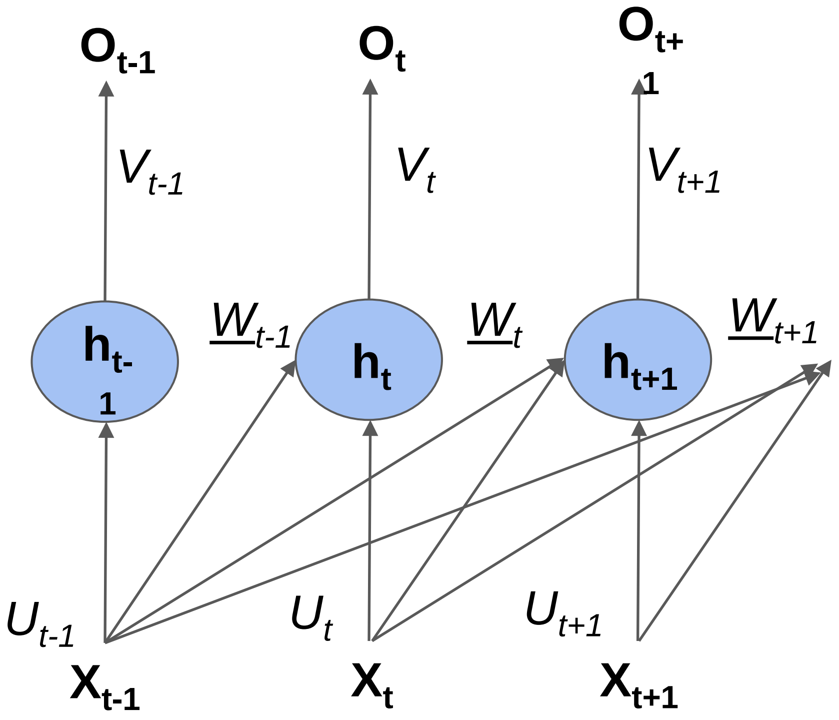
- Handles sequence? Yes ✓
- Models dependency?

A revised architecture



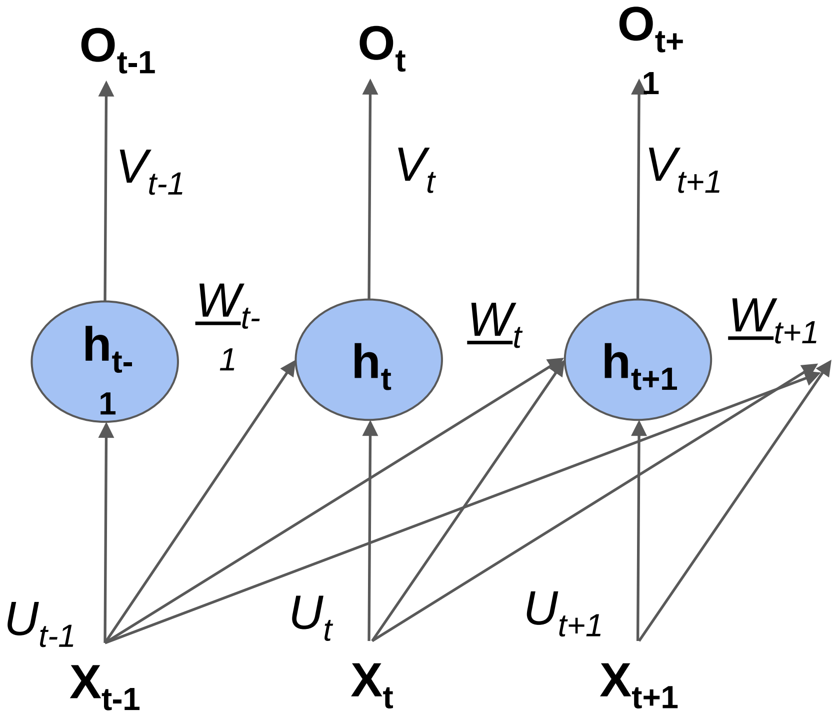
- Handles sequence? Yes ✓
- Models dependency? Yes ✓

A revised architecture



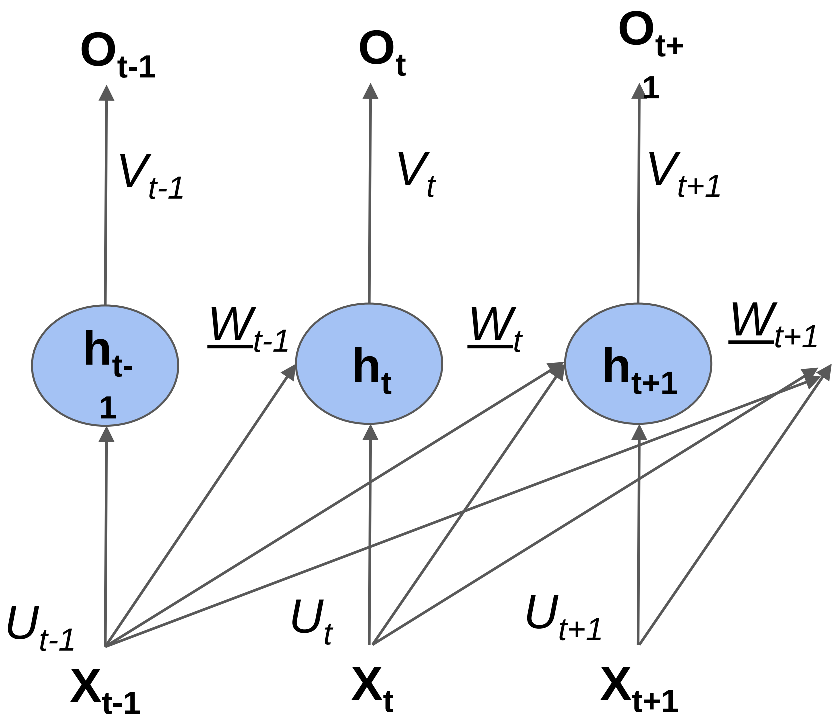
- Handles sequence? Yes ✓
- Models dependency? Yes ✓
- Variable length sequences?

A revised architecture



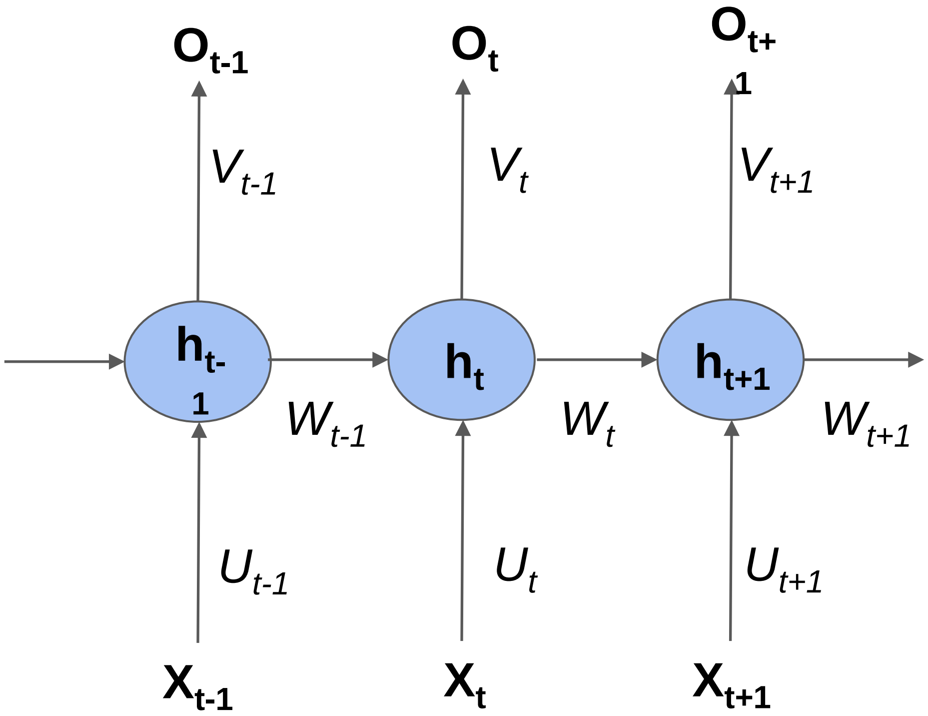
- Handles sequence? Yes ✓
- Models dependency? Yes ✓
- Variable length sequences? No ✗

A revised architecture

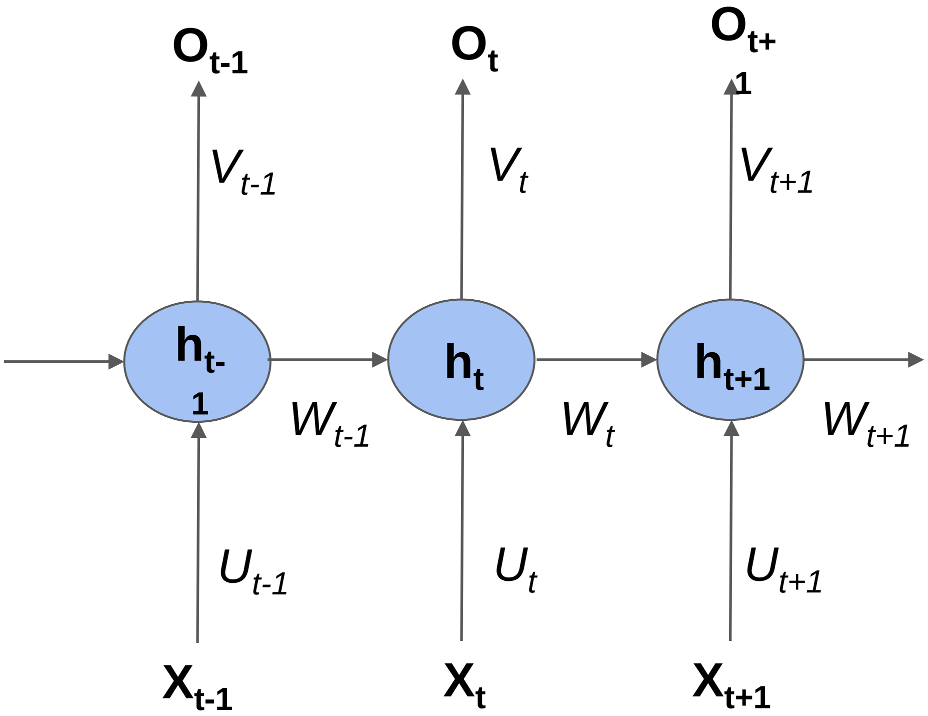


- Handles sequence? Yes ✓
- Models dependency? Yes ✓
- Variable length sequences? No ✗
- Too many parameters to train :(✗

A revised architecture

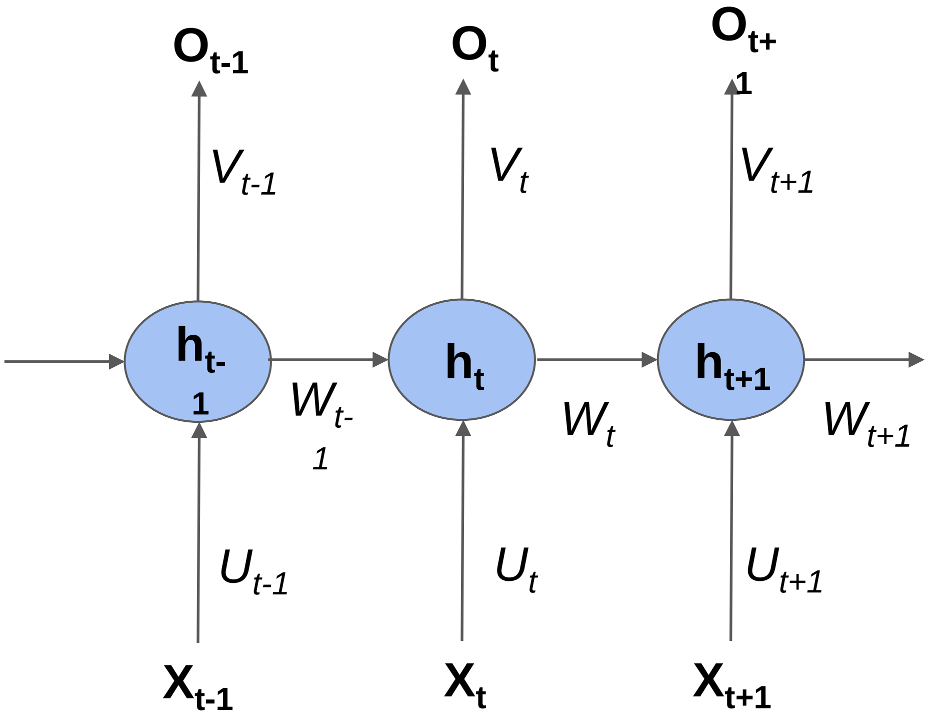


A revised architecture



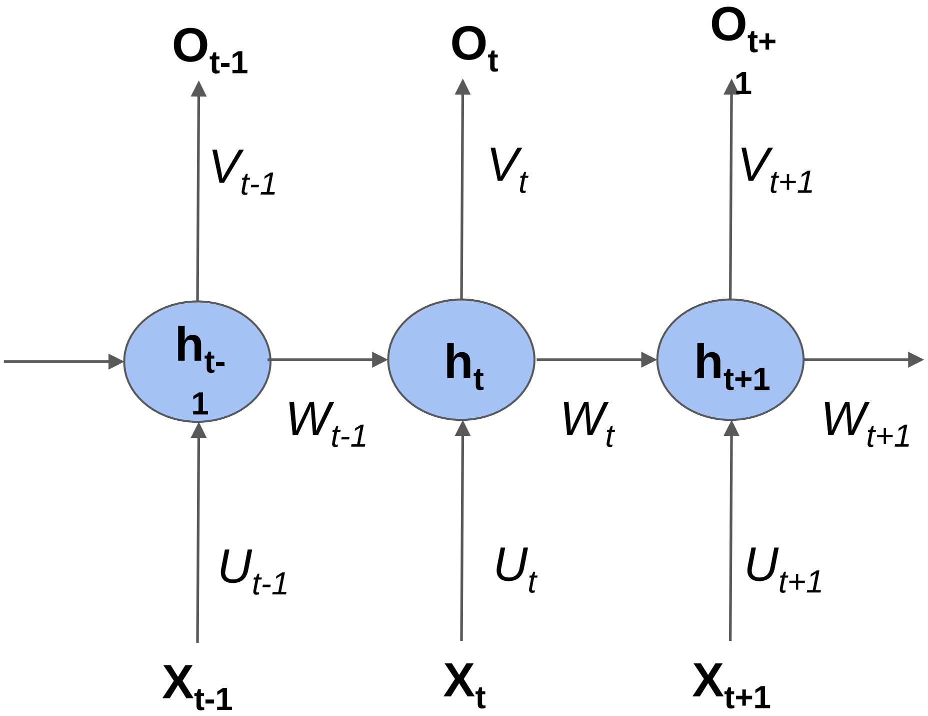
- Handles sequence?

A revised architecture



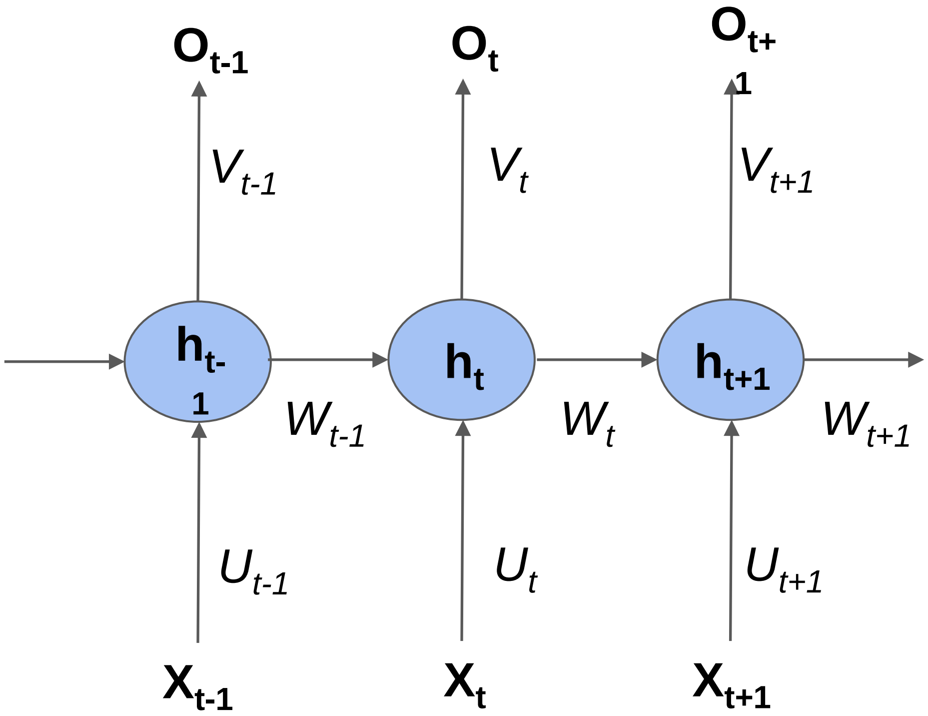
- Handles sequence? Yes ✓

A revised architecture



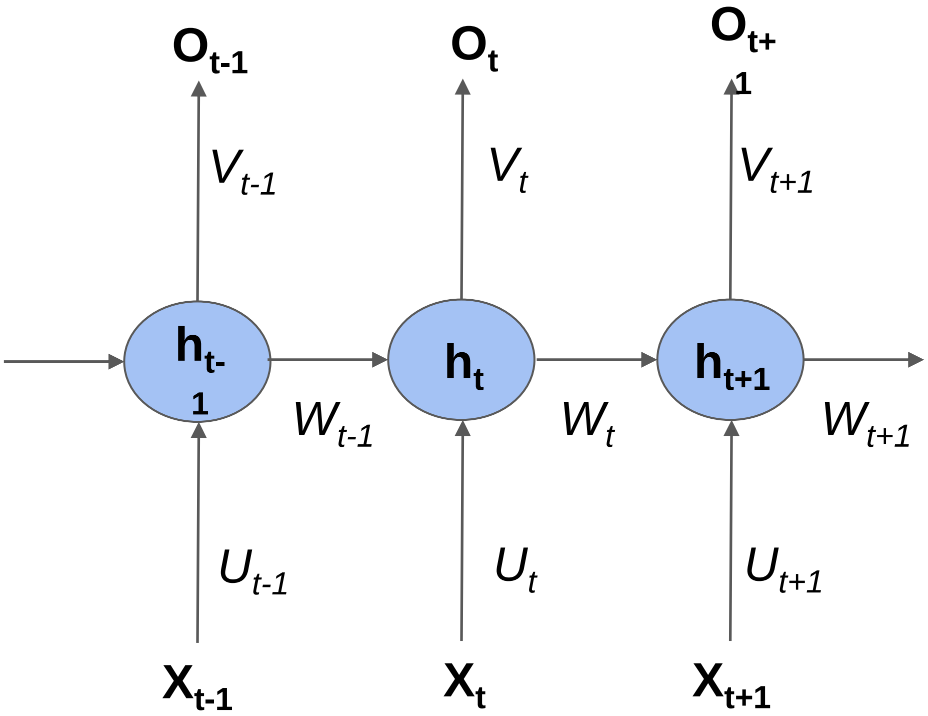
- Handles sequence? Yes ✓
- Models dependency?

A revised architecture



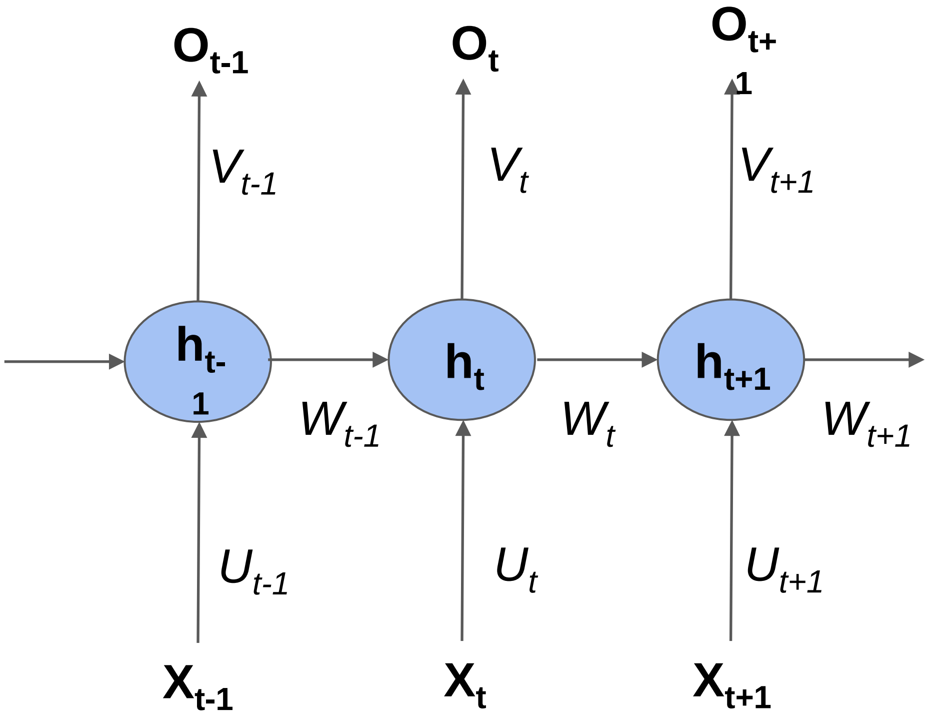
- Handles sequence? Yes ✓
- Models dependency? Yes ✓

A revised architecture



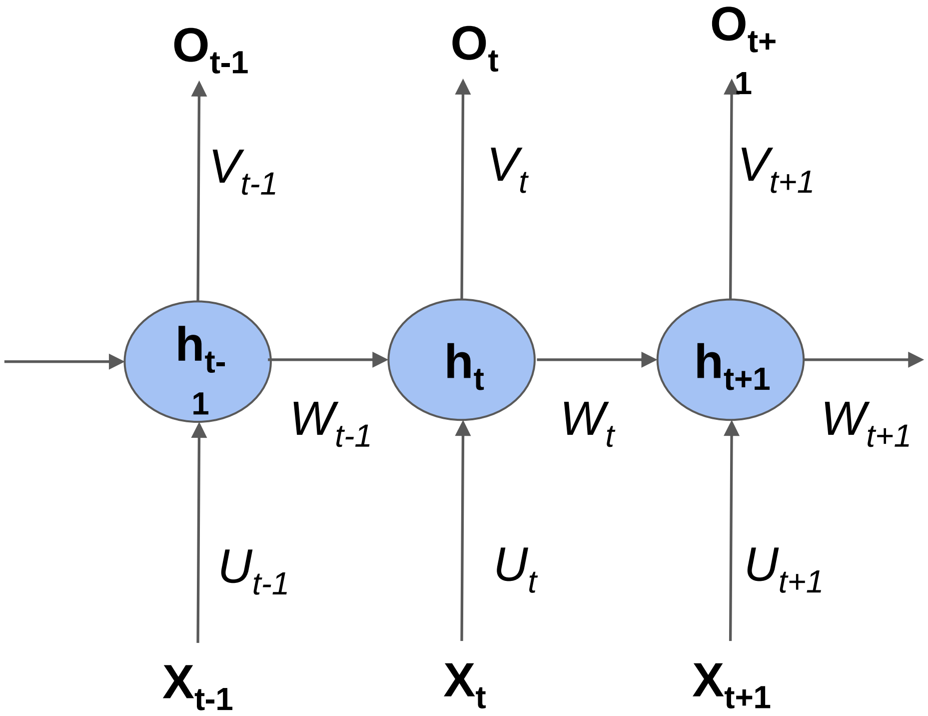
- Handles sequence? Yes ✓
- Models dependency? Yes ✓
- Variable length sequences?

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**

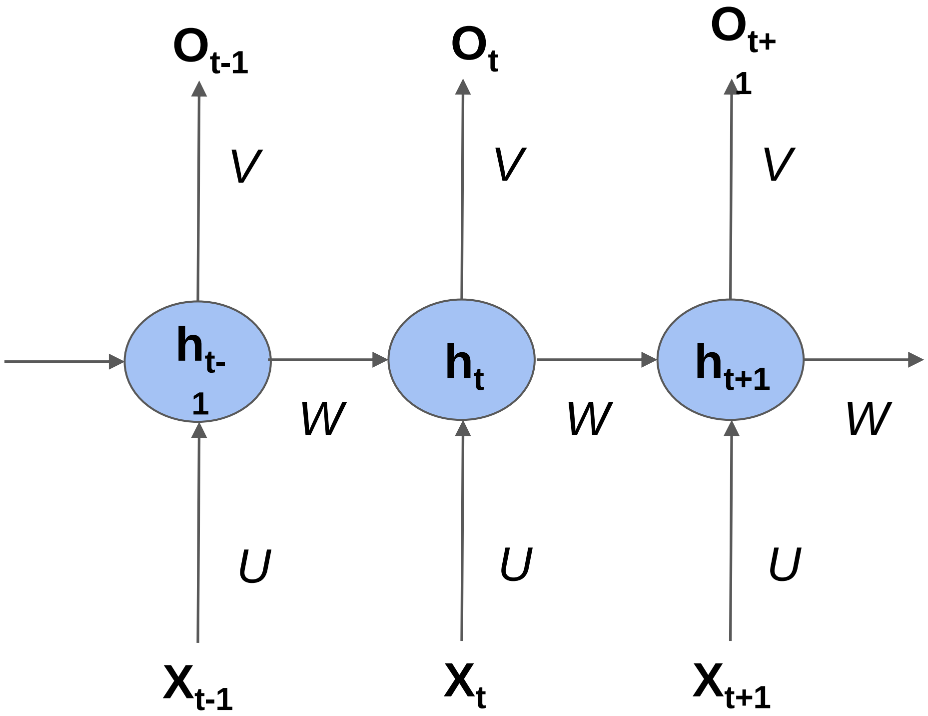
Lessons from CNNs and vanilla networks

- 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

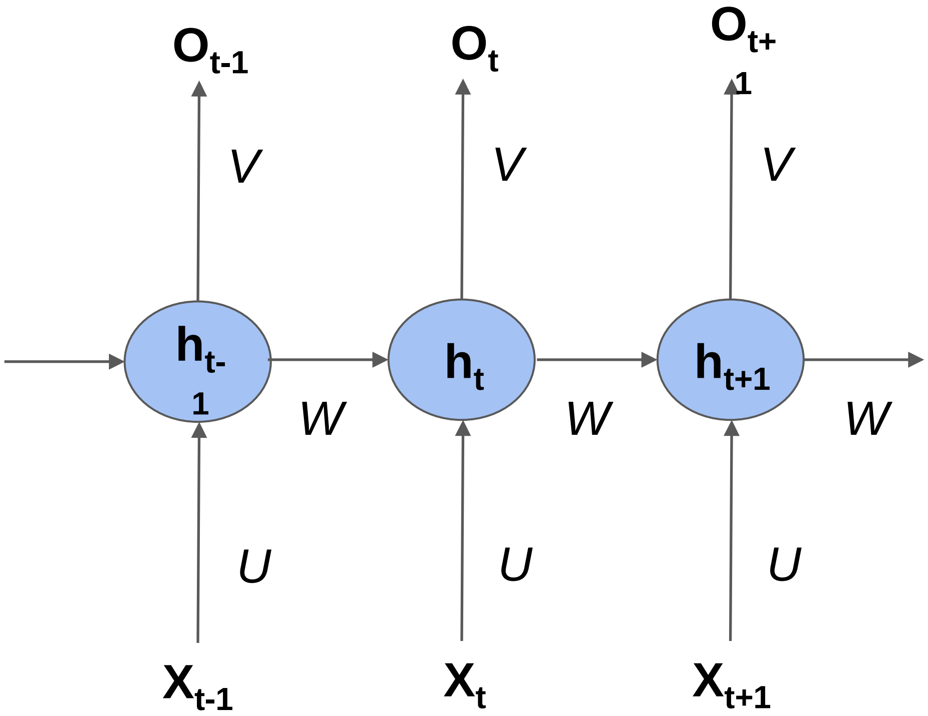
- Use CNNs to solve these problems
- Main idea of a CNN is **parameter sharing**

Can we now modify our network based on these?

A better architecture

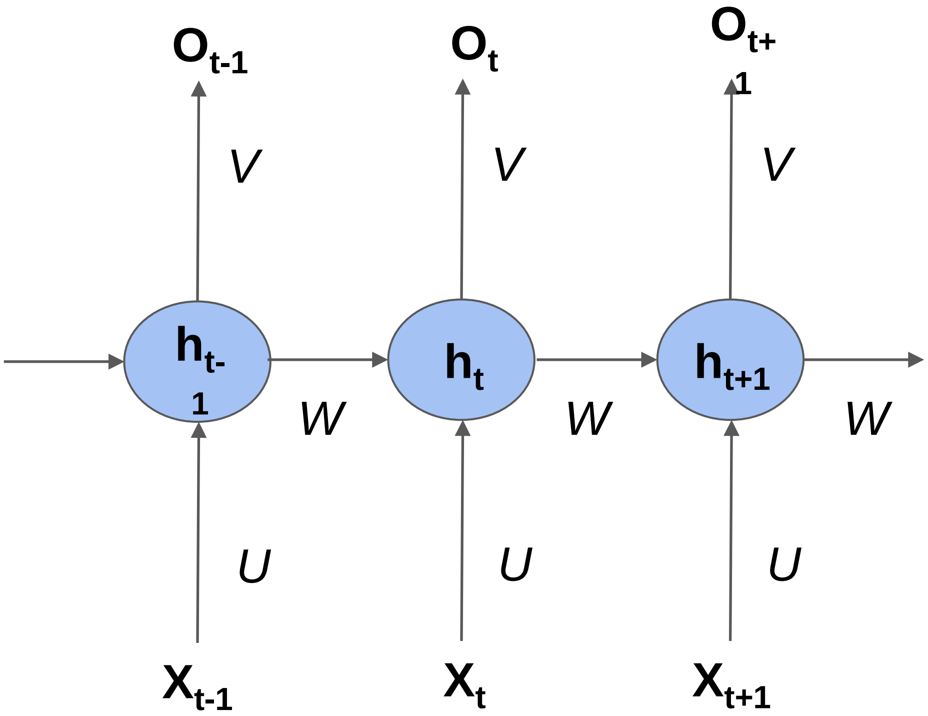


A better architecture



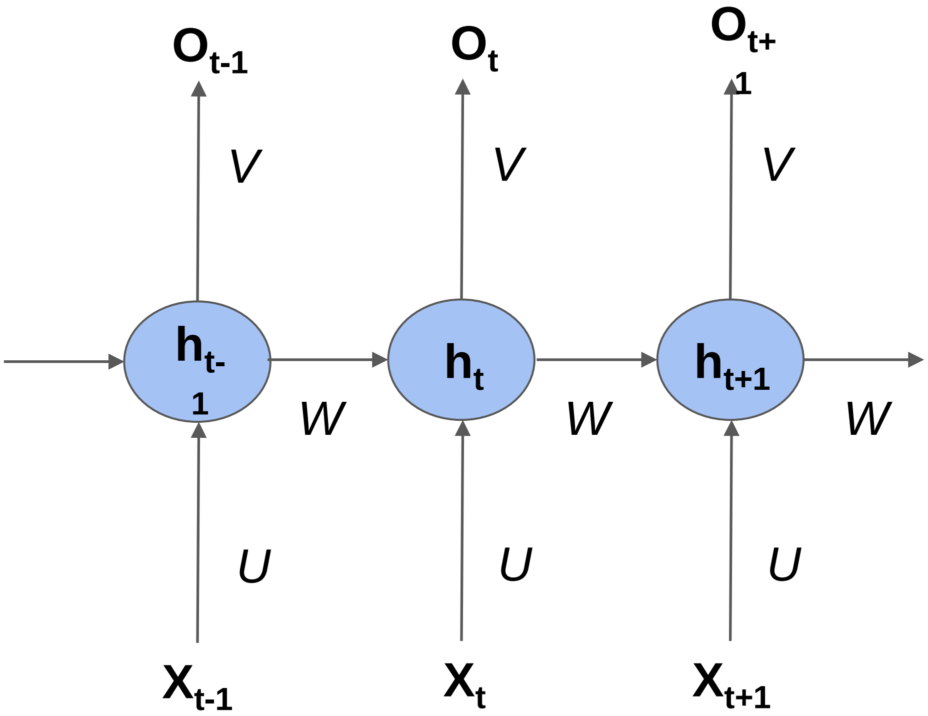
- Handles sequence?

A better architecture



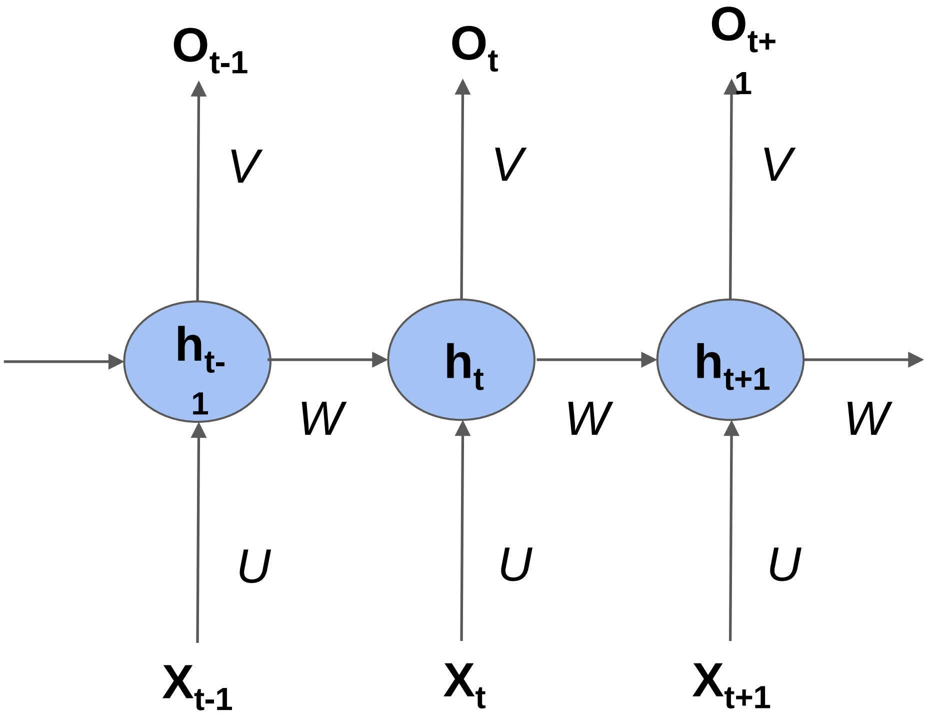
- Handles sequence? Yes ✓

A better architecture



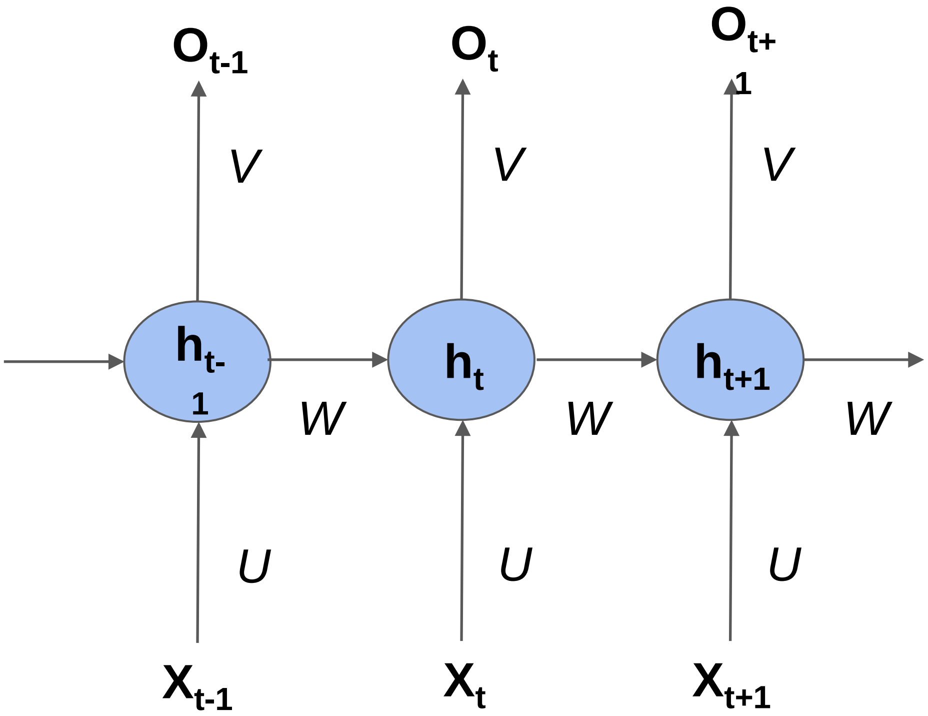
- Handles sequence? Yes ✓
- Models dependency?

A better architecture



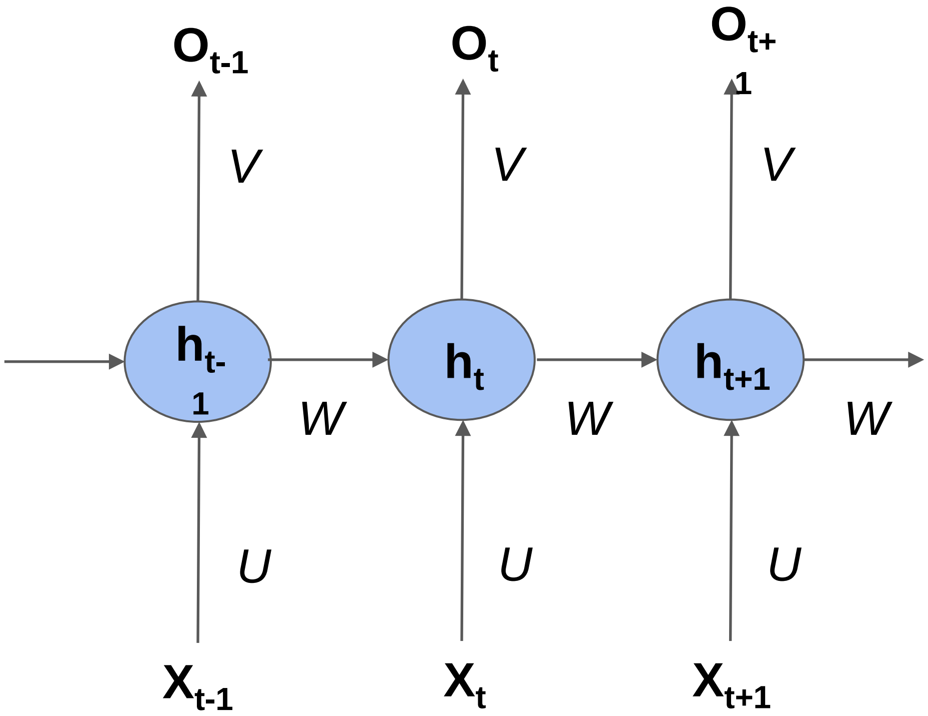
- Handles sequence? Yes ✓
- Models dependency? Yes ✓

A better architecture



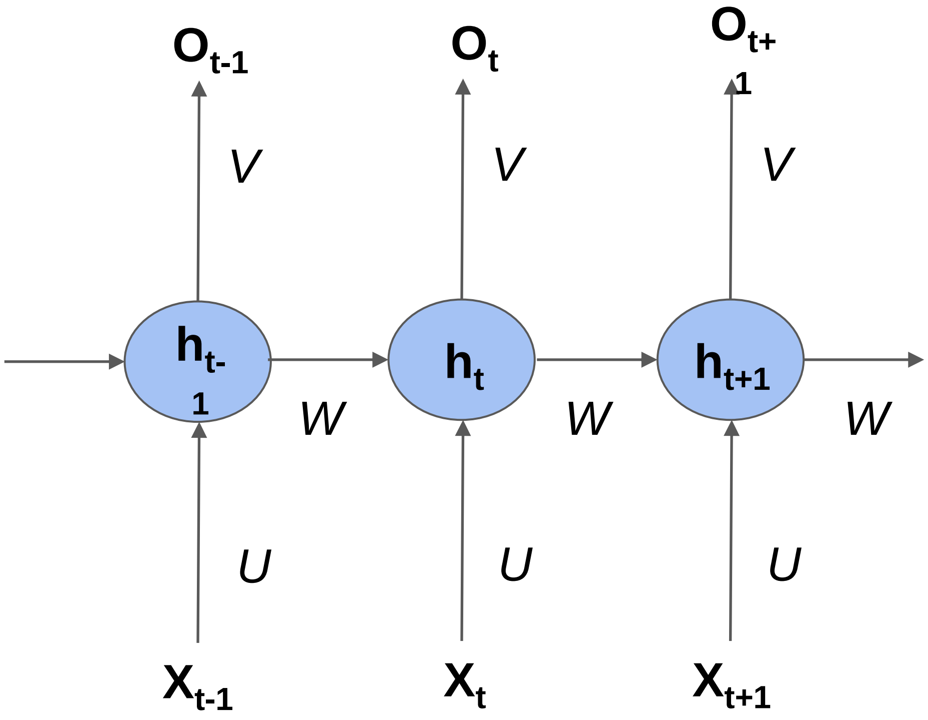
- Handles sequence? Yes ✓
- Models dependency? Yes ✓
- Variable length sequences?

A better architecture



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

A better architecture



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

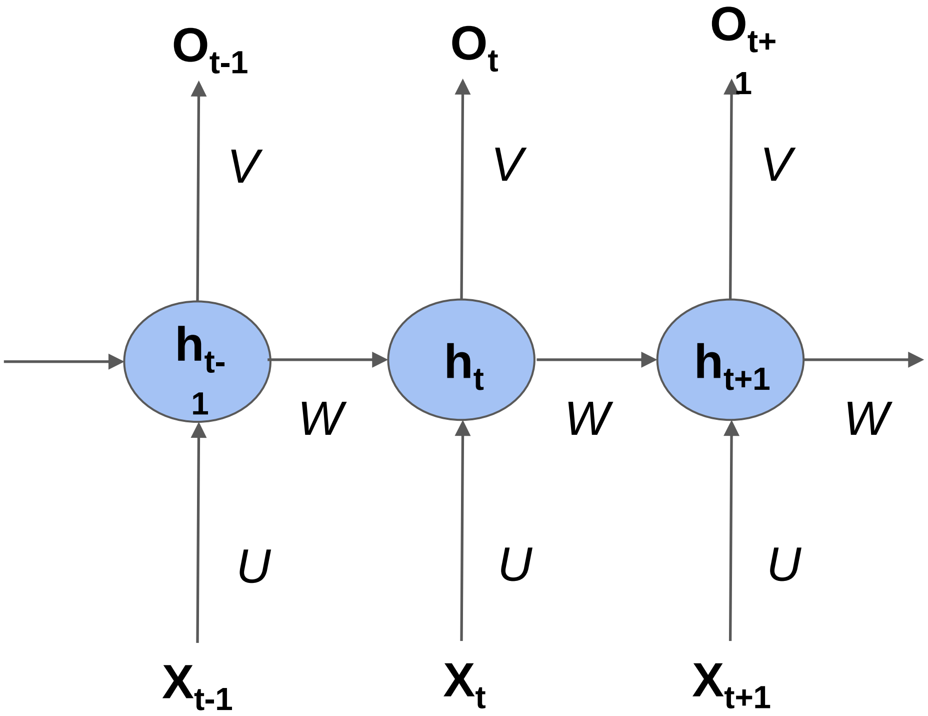
To summarize...

- Recurrent networks model sequences of data

$$\{x_1, x_2, \ldots, x_t, \ldots, x_{n-1}, x_n\} \longrightarrow \{y_1, y_2, \ldots, y_t, \ldots, y_{m-1}, y_m\}$$

- 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

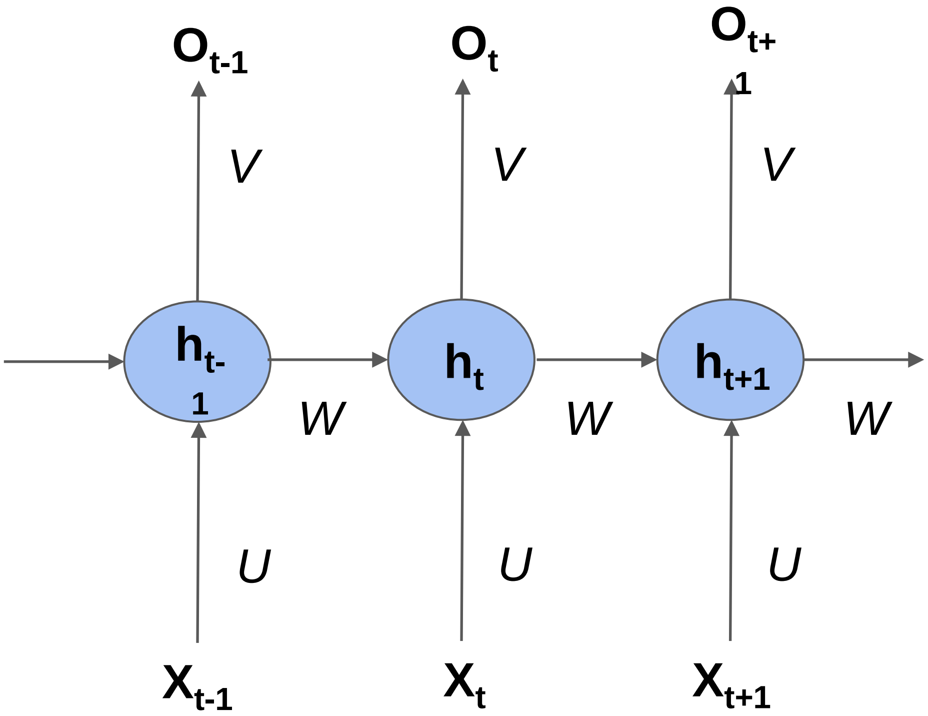
Recurrent networks



- At each time-step *t*,

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

Recurrent networks

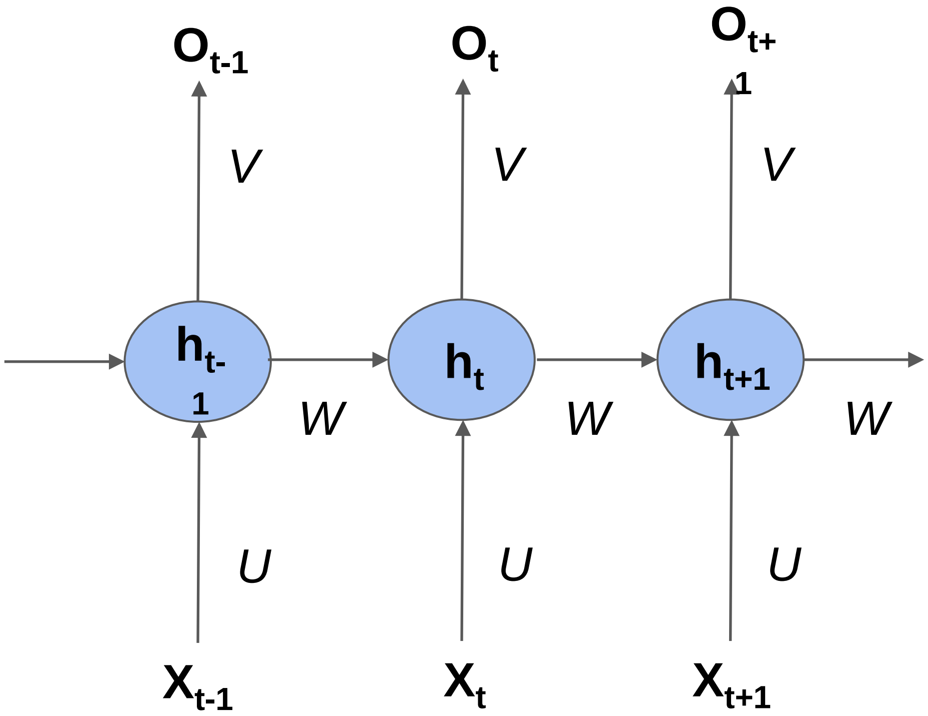


- At each time-step *t*,

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

- Here, ***f*** and ***g*** are activation functions, tanh, sigmoid, ReLU etc., and $\boldsymbol{h_t}$ is called the **hidden state** at time-step ***t*** and ***W, U, V*** are shared weights

Recurrent networks

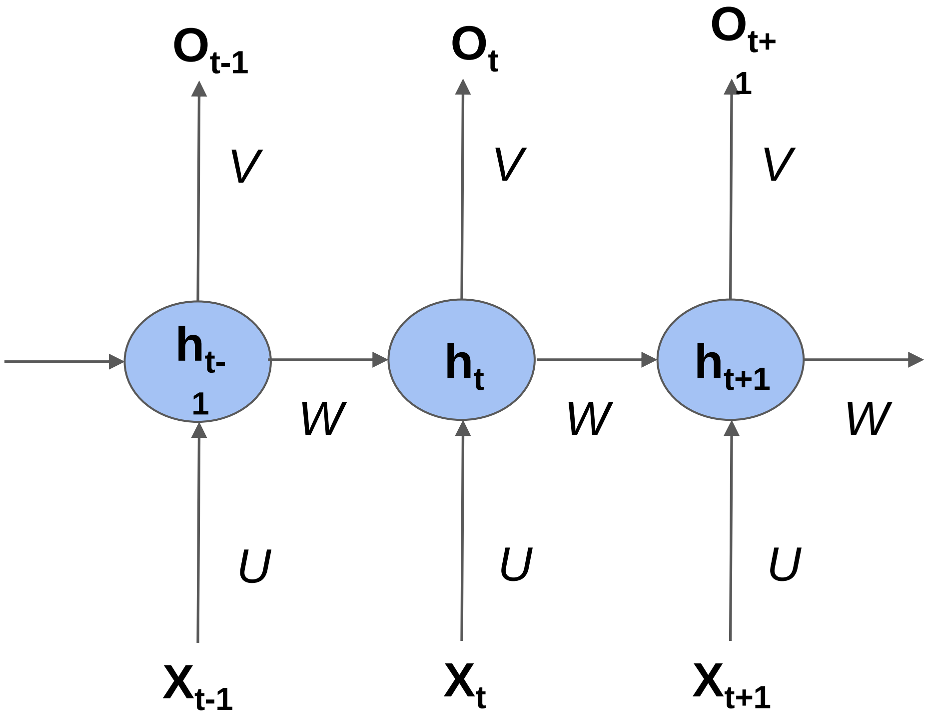


- At each time-step *t*,

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

- Here, ***f*** and ***g*** are activation functions, tanh, sigmoid, ReLU etc., and $\boldsymbol{h_t}$ is called the **hidden state** at time-step ***t*** and ***W, U, V*** are shared weights

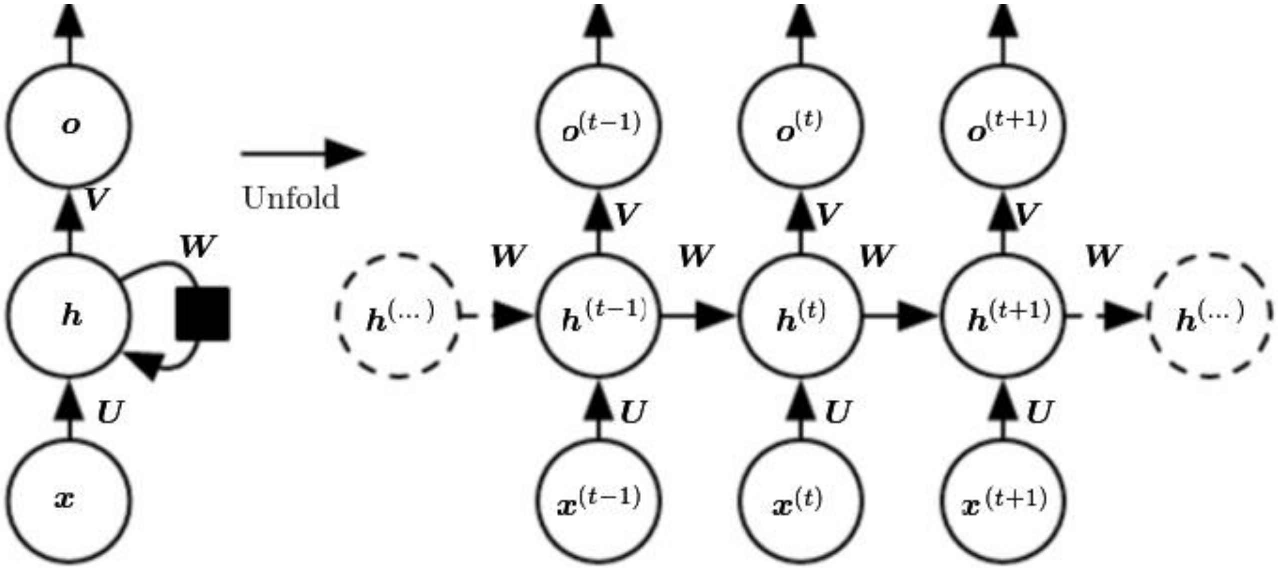
Recurrent networks



- 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

*image courtesy Deep Learning book, Ian, Yoshua, Aaron

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

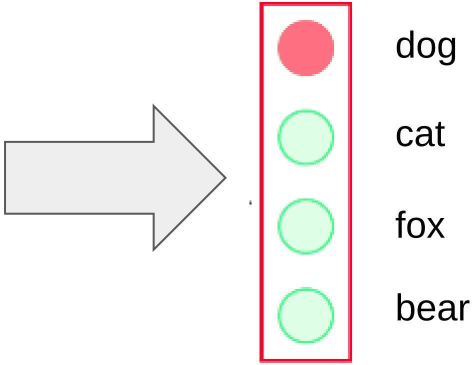
*slide courtesy, Chetan Arora CVIT Summer School

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 → text string, or speech → text

Different flavors of RNNs

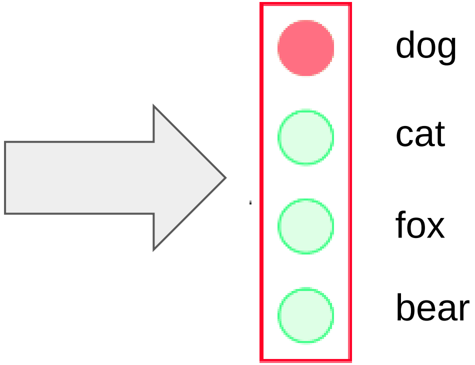
Different flavors of RNNs



Architecture?
RNN? CNN?

*image courtesy, Google images

Different flavors of RNNs



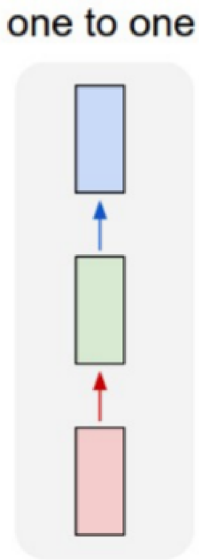


*image courtesy, Google images and Andrej Karpathy blogpost

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

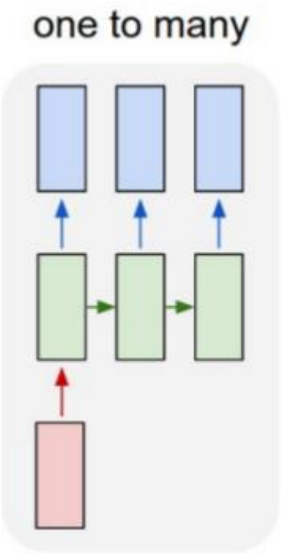
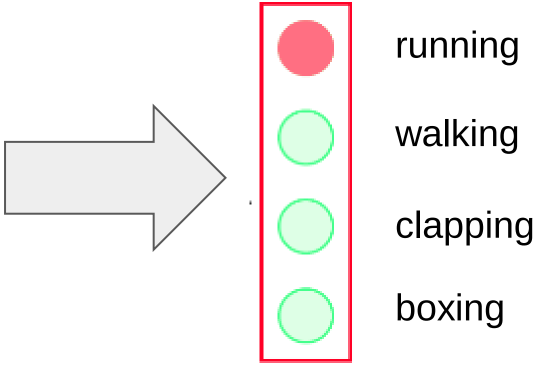


*image courtesy, Andrej Karpathy blogpost

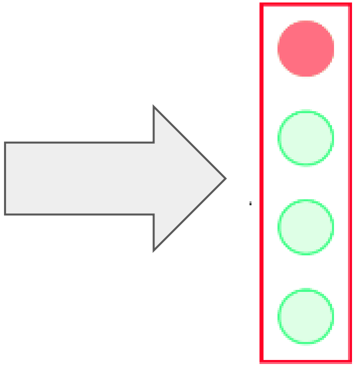
Different flavors of RNNs



Architecture?

*image courtesy, Google images

Different flavors of RNNs



running

walking

clapping

boxing

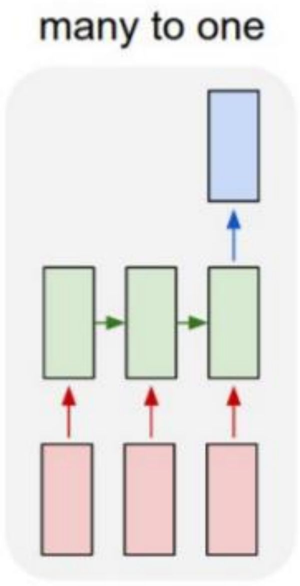


*image courtesy, Google images and Andrej Karpathy blogpost

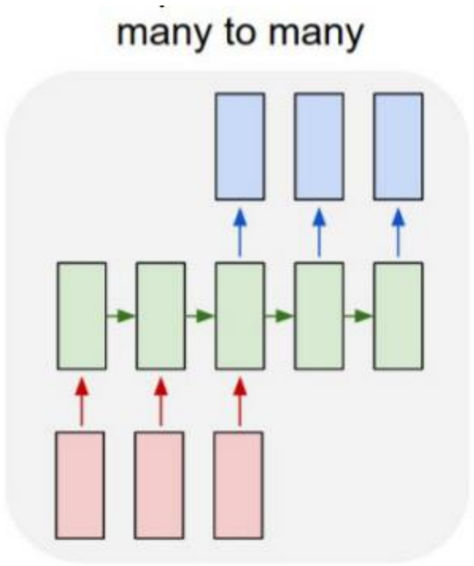
Different flavors of RNNs

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

Architecture?

Different flavors of RNNs

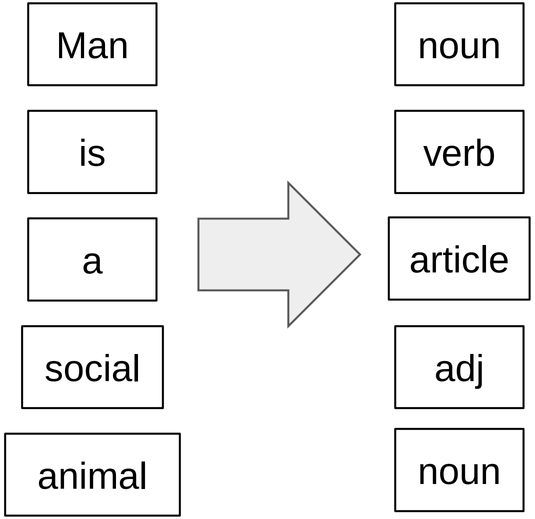
Hello, how are you? ⟷ नमस्ते आप कैसे हैं?



Asynchronous

*image courtesy, Andrej Karpathy blogpost

Different flavors of RNNs



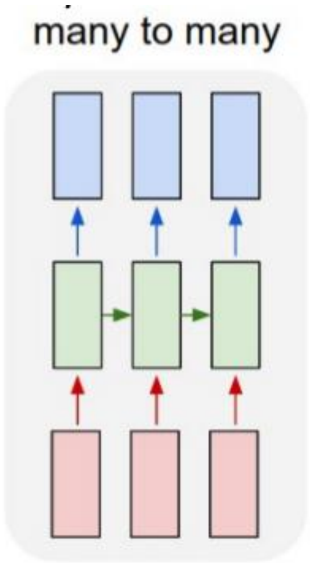
Part-of-speech tagging

Architecture?

Different flavors of RNNs

Man → noun
is → verb
a → article
social → adj
animal → noun

Part-of-speech tagging



Synchronous

*image courtesy, Andrej Karpathy blogpost

Different flavors of RNNs

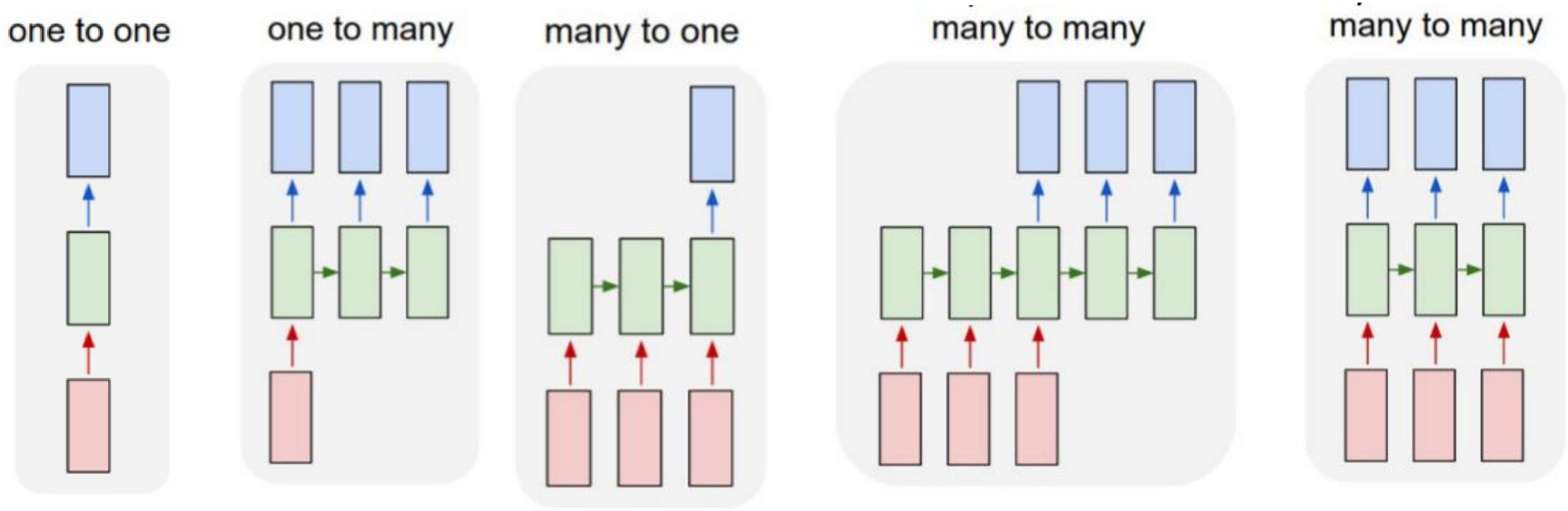
Image classification

Scene text, Image Captioning

Sentiment analysis, Activity recognition from video

Machine translation

Part-of-speech tagging

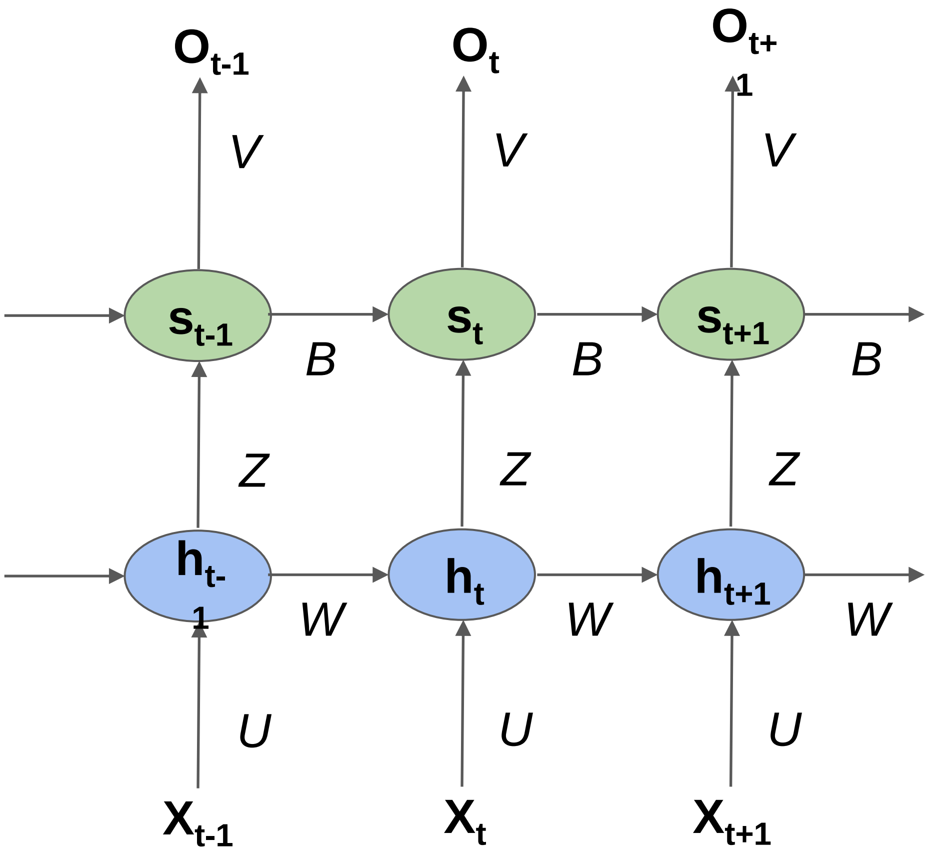


Asynchronous

Synchronous

*image courtesy, Andrej Karpathy blogpost

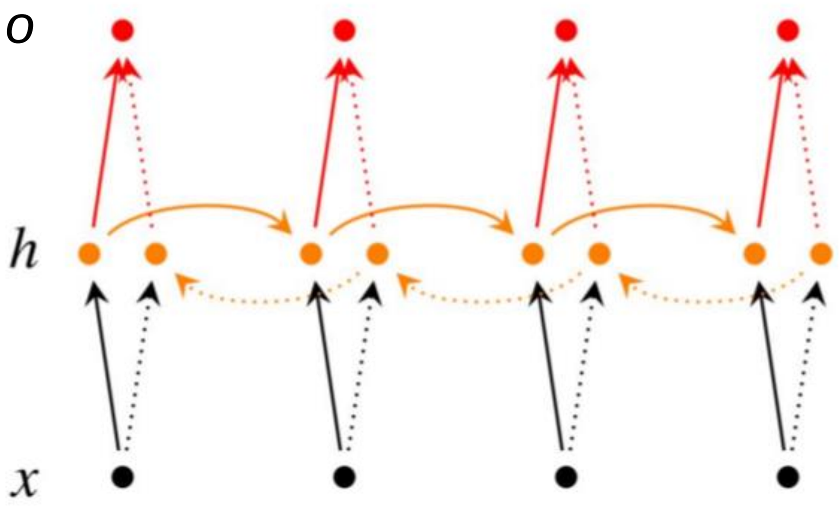
Different flavors of RNNs



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



$$\overrightarrow{h}_t = f\left(\overrightarrow{W}\,\overrightarrow{h}_{t-1} + \overrightarrow{U}x_t\right)$$

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

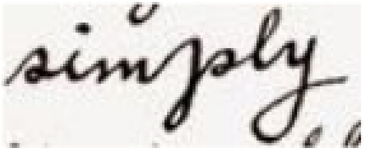
$$o_t = g\left(V\left[\overleftarrow{h}_t; \overrightarrow{h}_t\right]\right)$$

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

*image courtesy Google images

Case study: Scene text recognition

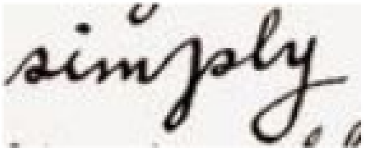
- **Problem:** Extract the text from an image



“simply

Case study: Scene text recognition

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



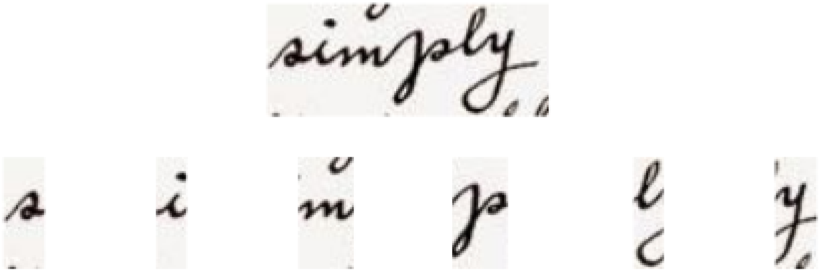
`“simply”`

Can you design an architecture for this?

A simple architecture with character segmentation

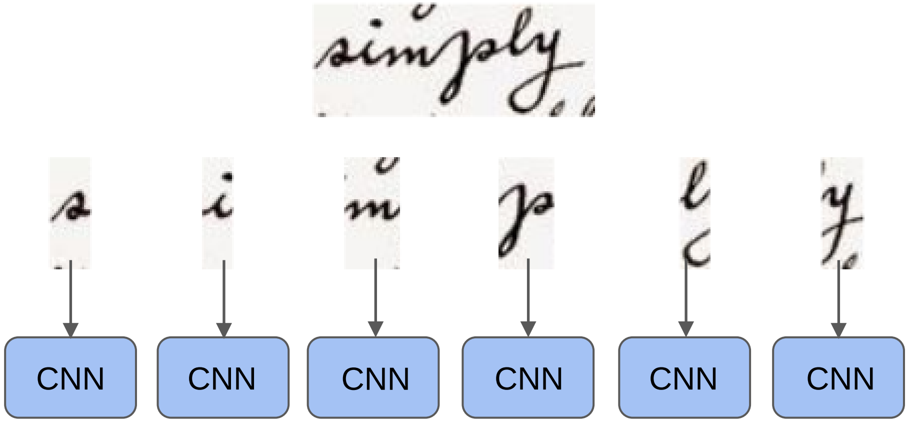
A simple architecture with character segmentation

- Segment the image into characters



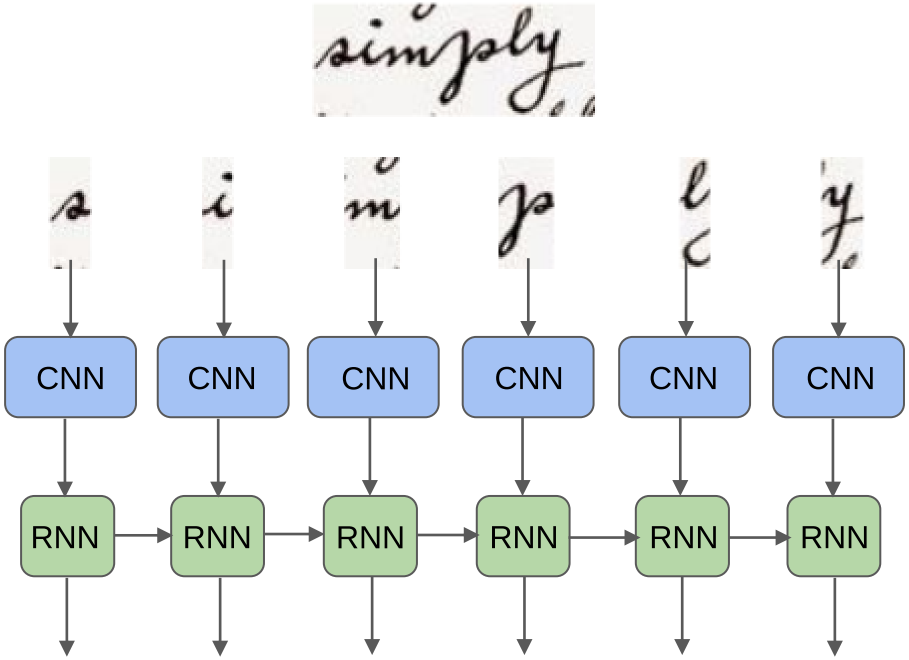
A simple architecture with character segmentation

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



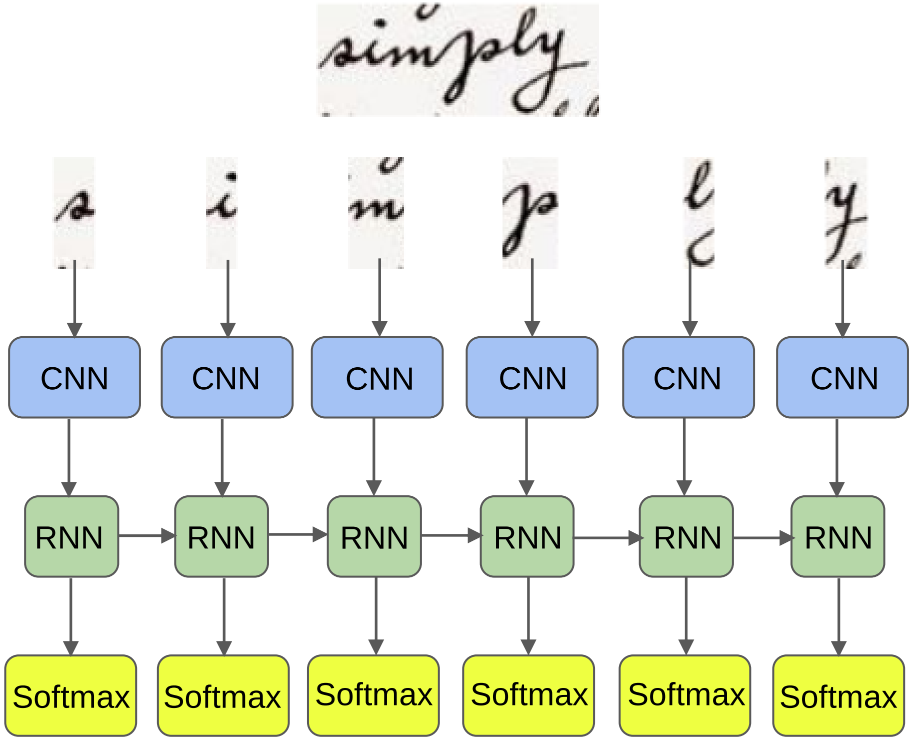
A simple architecture with character segmentation

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



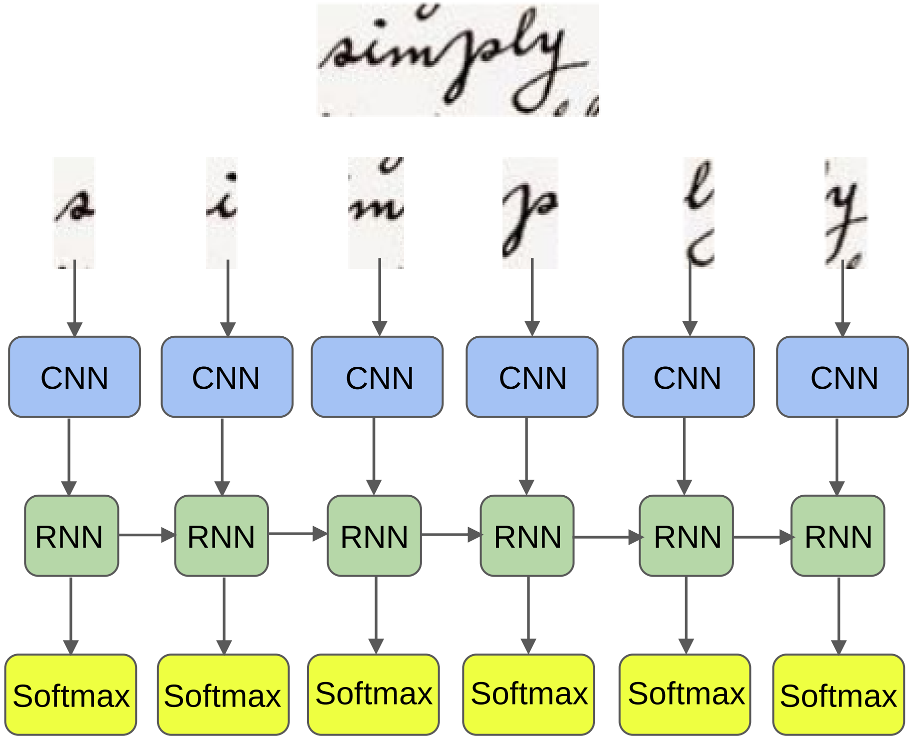
A simple architecture with character segmentation

- 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?

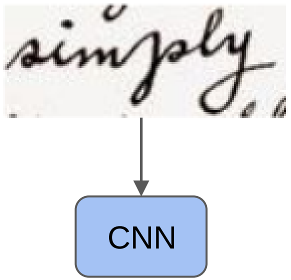


Can we improve this architecture?

Architecture without character segmentation

Architecture without character segmentation

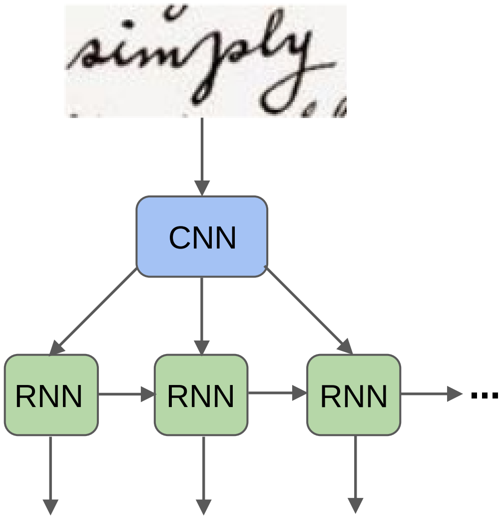
- Extract the features of the image using CNN



Architecture without character segmentation

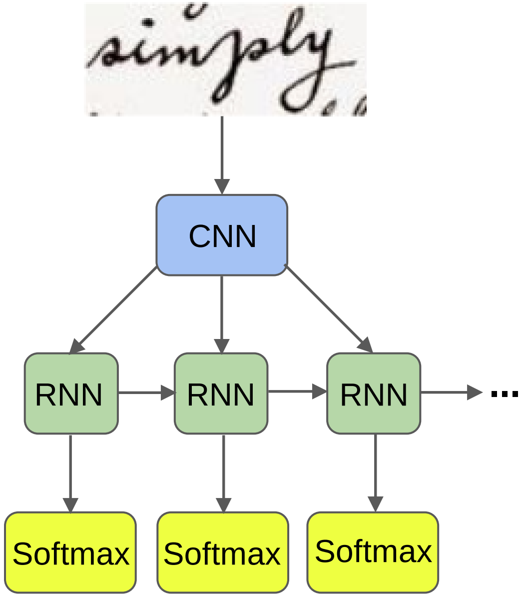
- Extract the features of the image using CNN

- Pass those features as input to RNN at each time-step



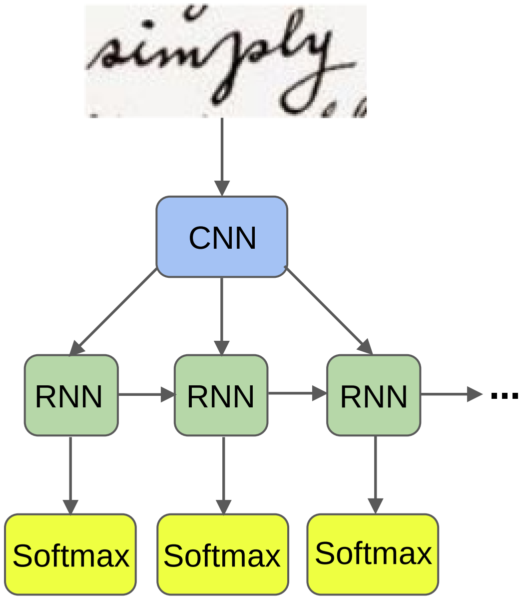
Architecture without character segmentation

- 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,

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

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

$$T = \text{Number of time-steps}$$

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

How to train the network?

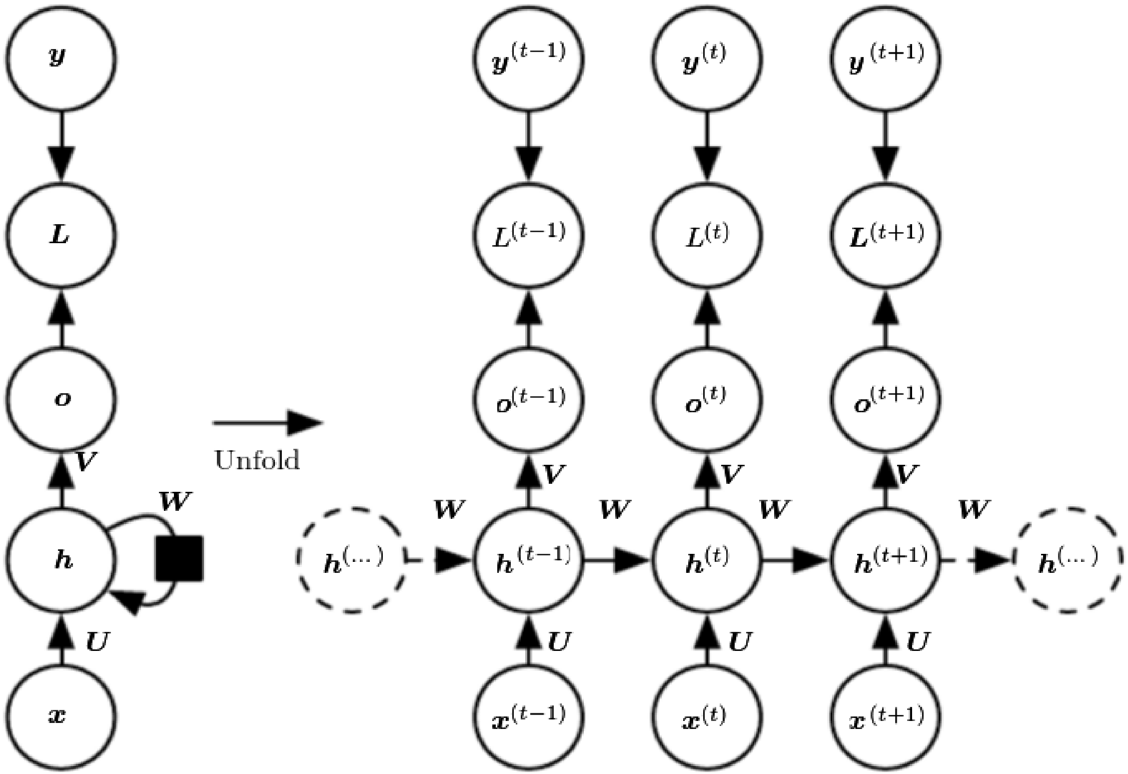
How to train the network?

Back propagation

Training RNNs

- Let’s say the weights are ***U, V, W*** as denoted previously
- Using backpropagation, we need to compute the gradients of the loss with respect to ***U, V, W***
- Let’s see how to do that

Training RNNs



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

*image courtesy Deep Learning book, Ian, Yoshua, Aaron

Training RNNs

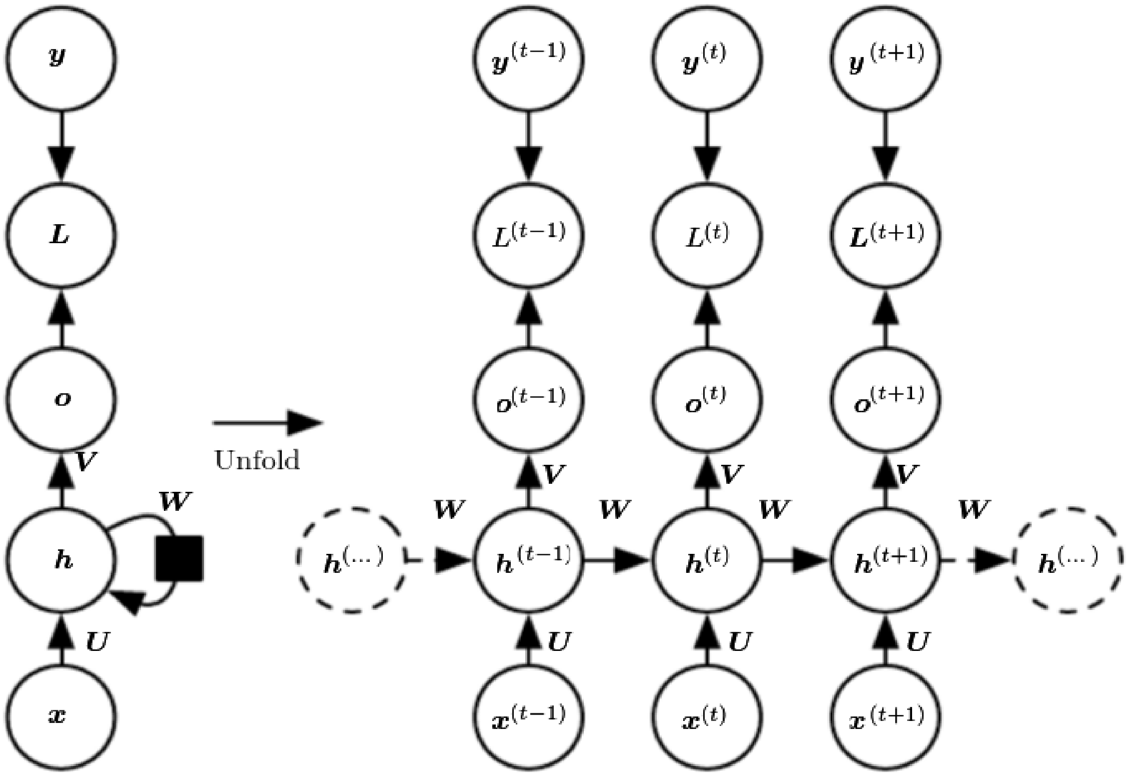


*image courtesy Deep Learning book, Ian, Yoshua, Aaron

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

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

Training RNNs

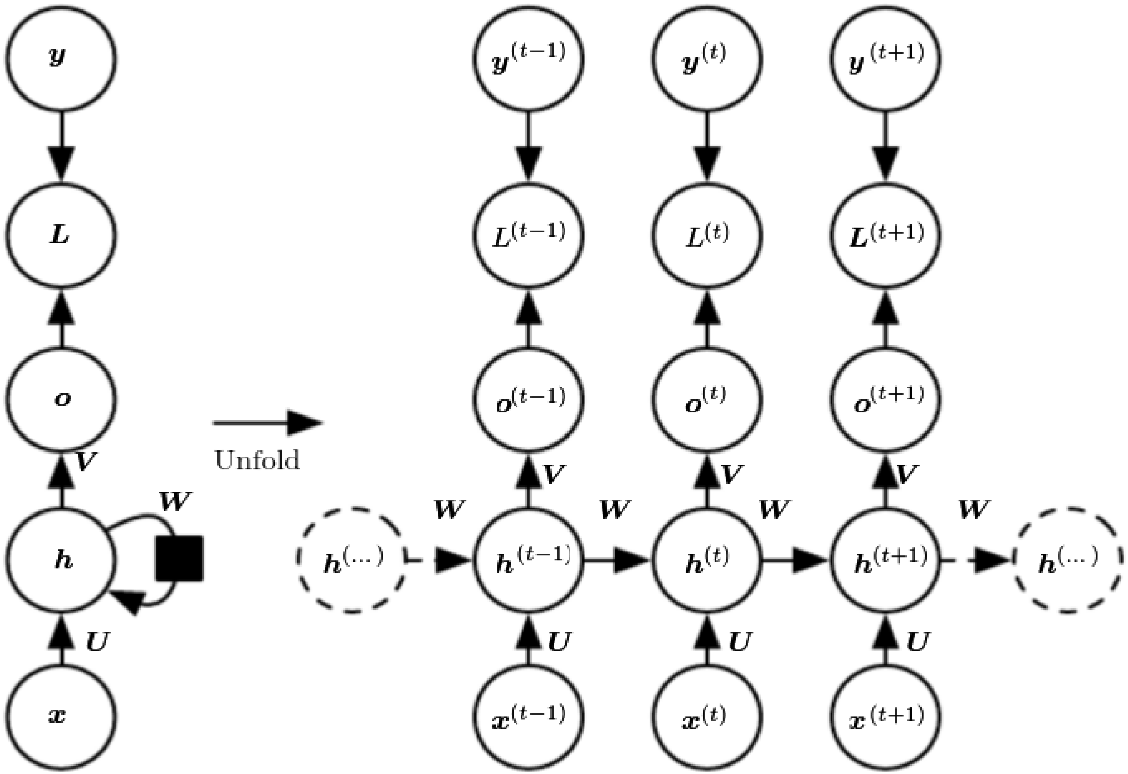


*image courtesy Deep Learning book, Ian, Yoshua, Aaron

- 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

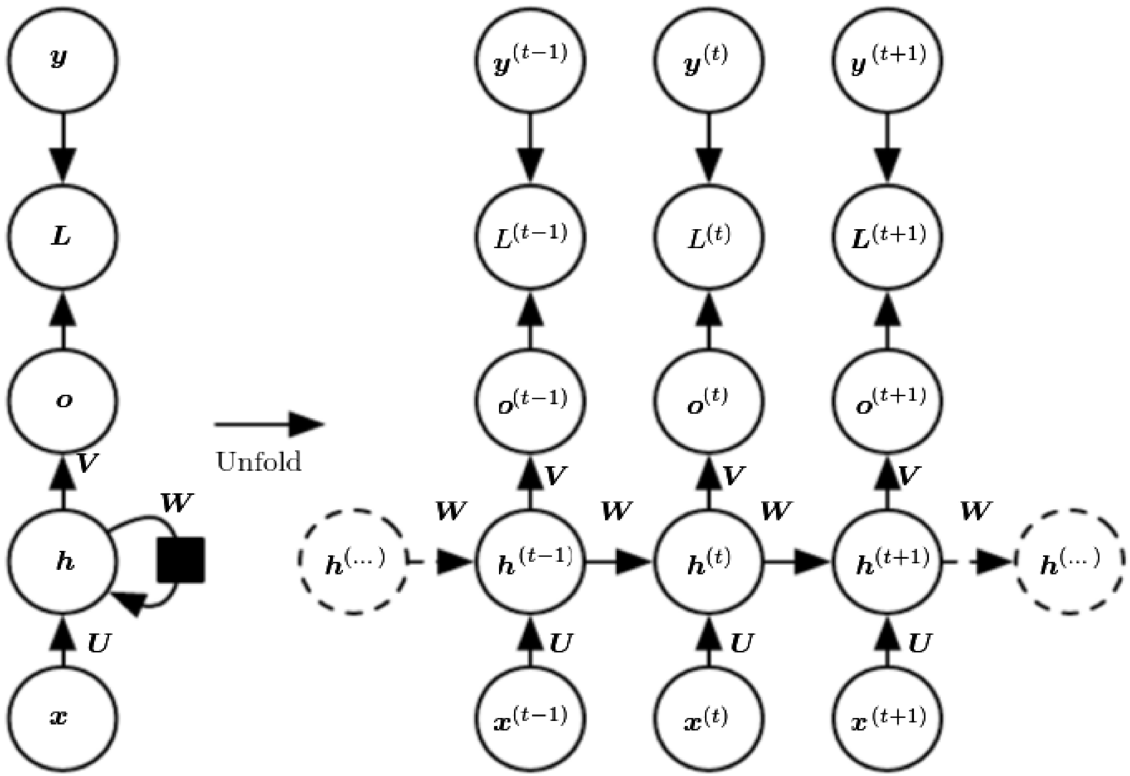


*image courtesy Deep Learning book, Ian, Yoshua, Aaron

- 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

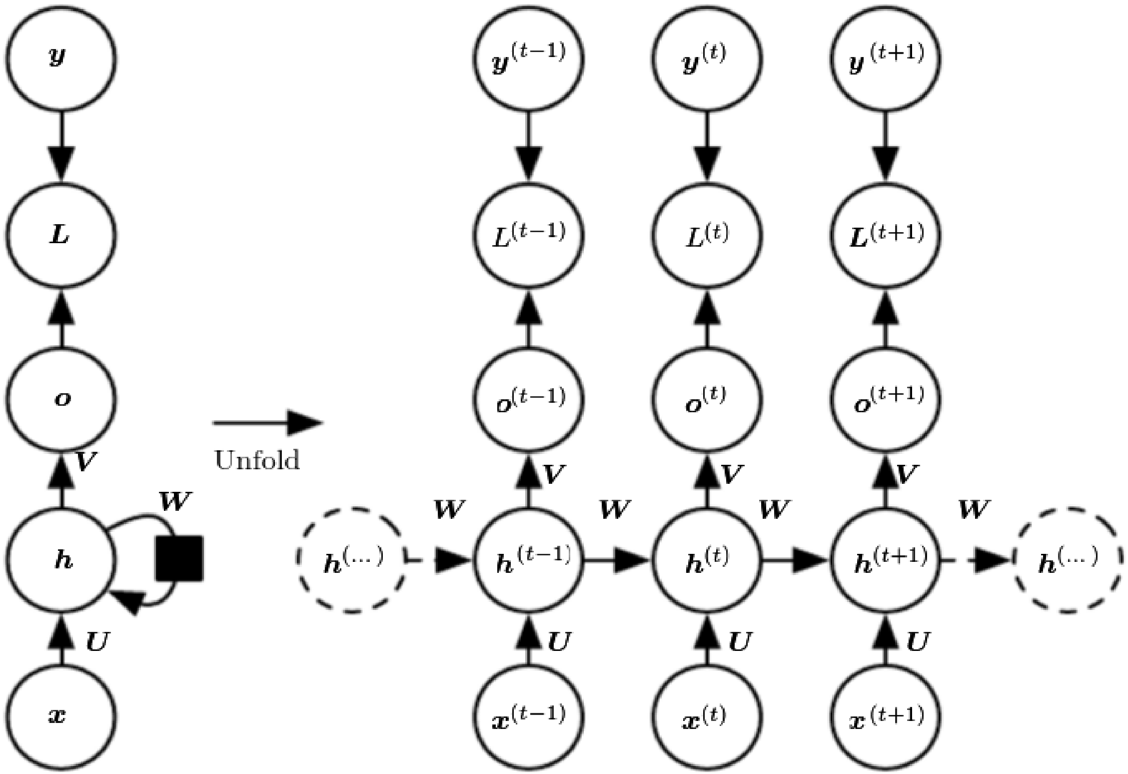


*image courtesy Deep Learning book, Ian, Yoshua, Aaron

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

Training RNNs

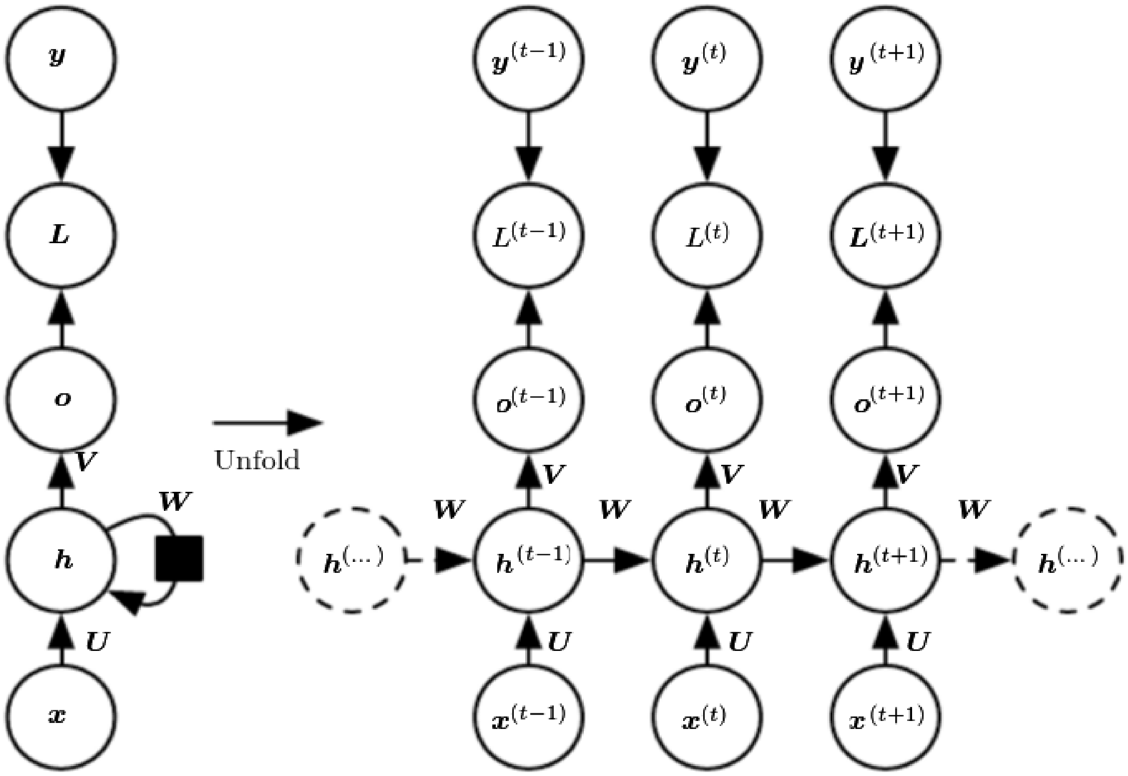


*image courtesy Deep Learning book, Ian, Yoshua, Aaron

- 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

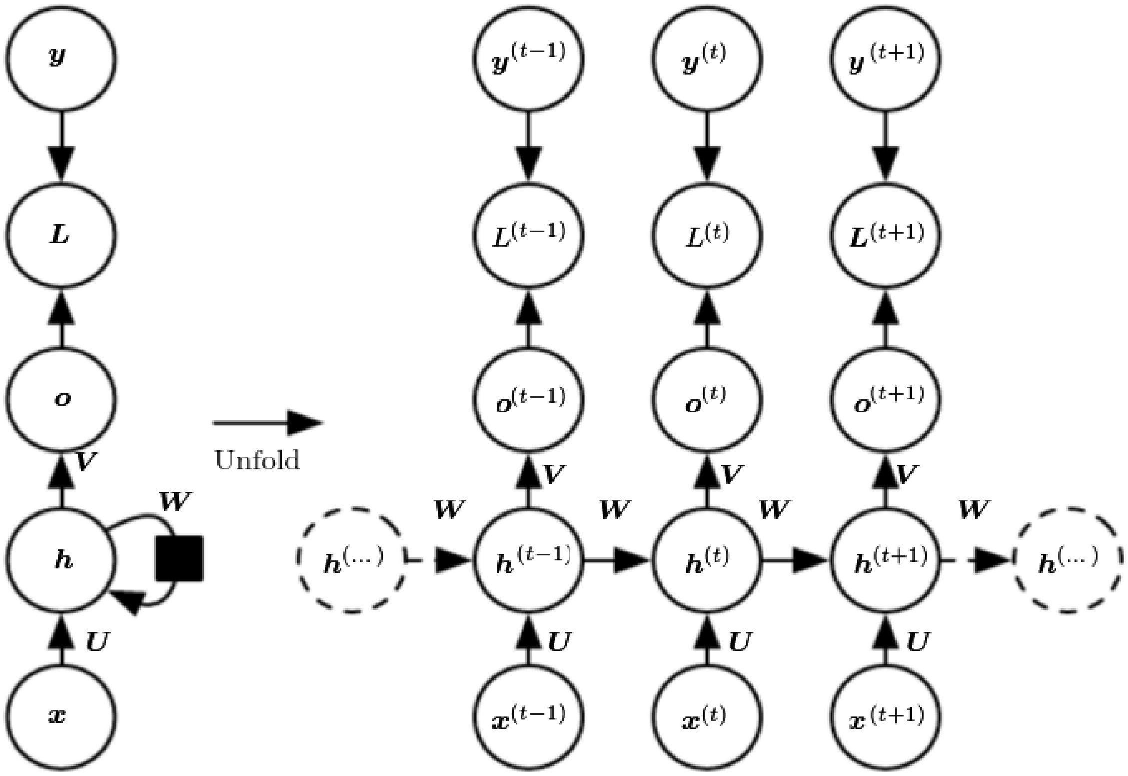


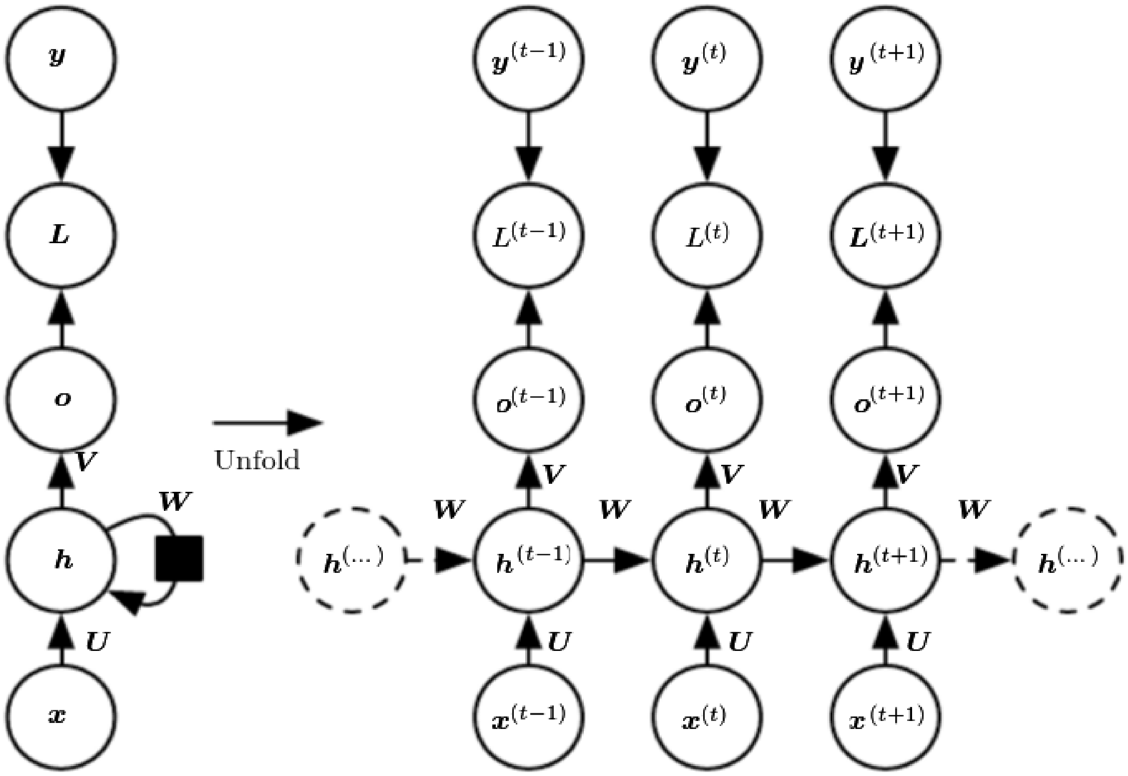
*image courtesy Deep Learning book, Ian, Yoshua, Aaron

- 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 ***W***

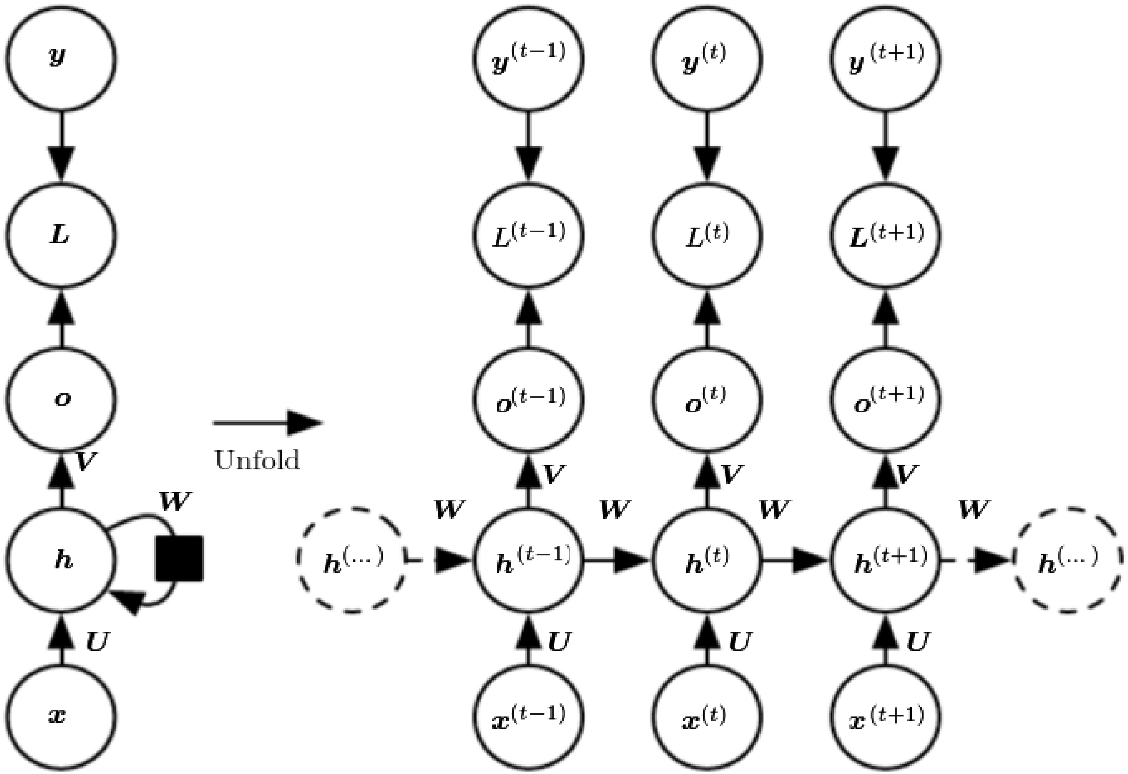
Training RNNs



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

*image courtesy Deep Learning book, Ian, Yoshua, Aaron

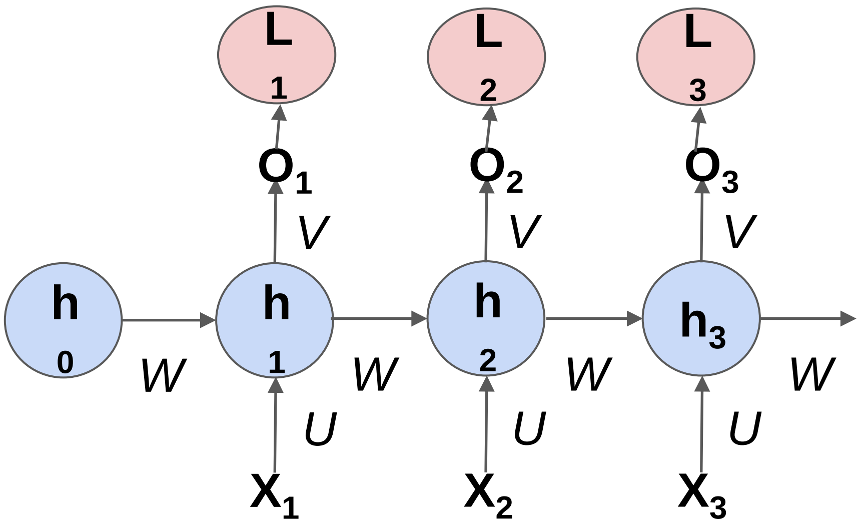
Training RNNs



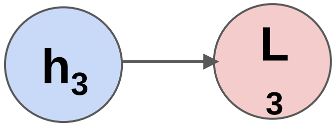
- What are the paths connecting $\mathbb{L}_t\left(\theta\right)$ to ***W*** ?
- Let us analyze this by calculating $\mathbb{L}_3\left(\theta\right)$

*image courtesy Deep Learning book, Ian, Yoshua, Aaron

Training RNNs

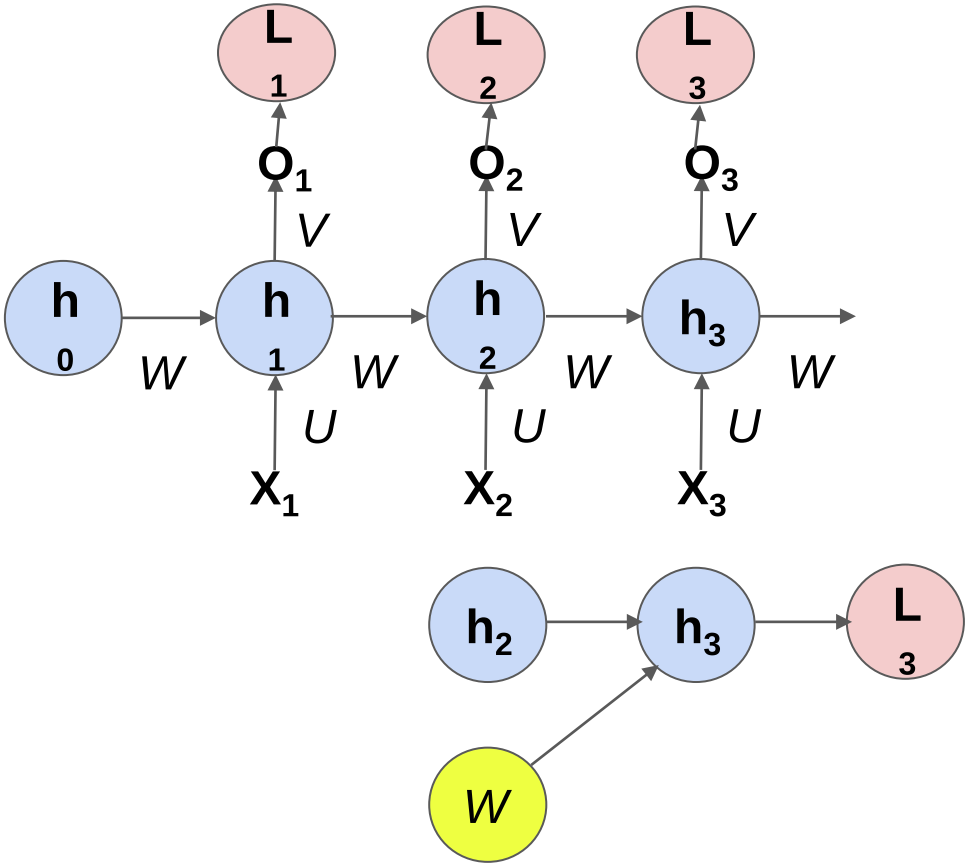


- $\mathbb{L}_3(\theta)$ depends on $\boldsymbol{h_3}$



*slide courtesy Mitesh Khapra, CS7015

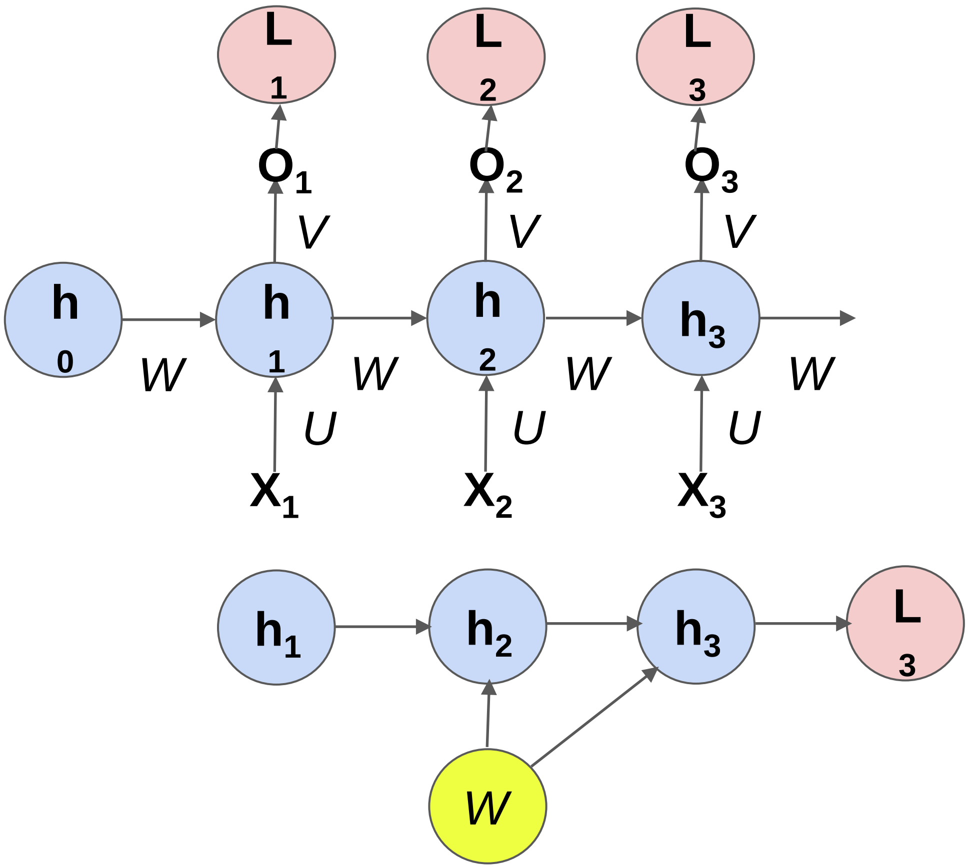
Training RNNs



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

*slide courtesy Mitesh Khapra, CS7015

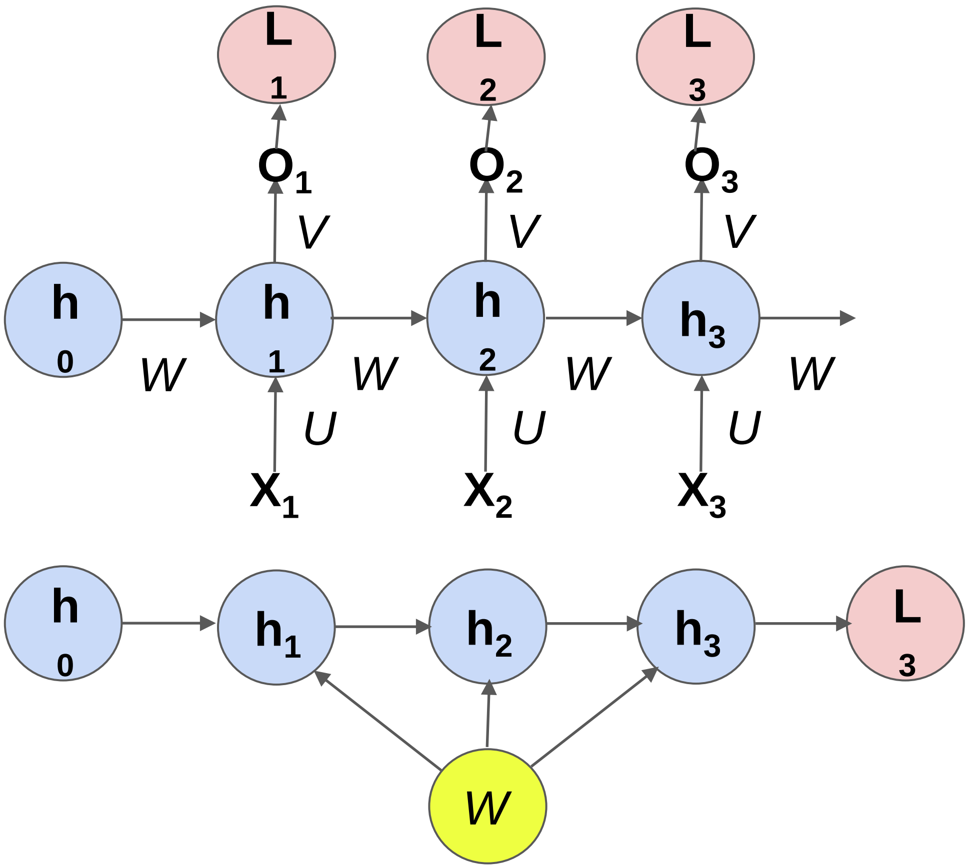
Training RNNs



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

*slide courtesy Mitesh Khapra, CS7015

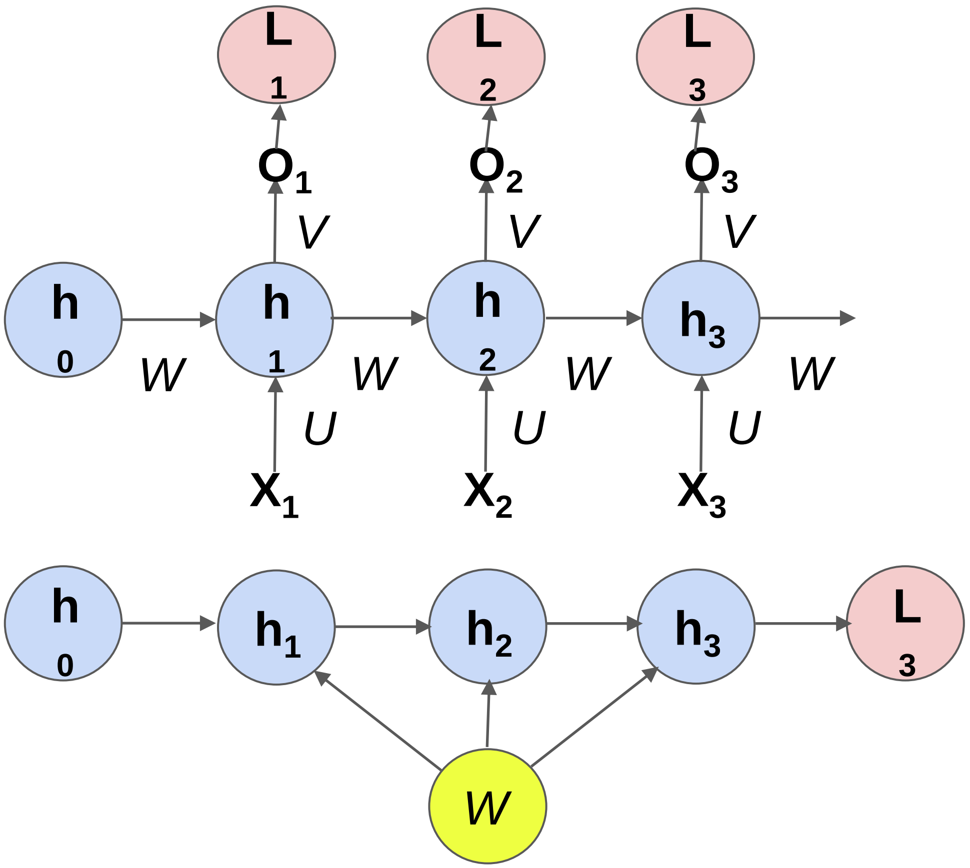
Training RNNs



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

*slide courtesy Mitesh Khapra, CS7015

Training RNNs

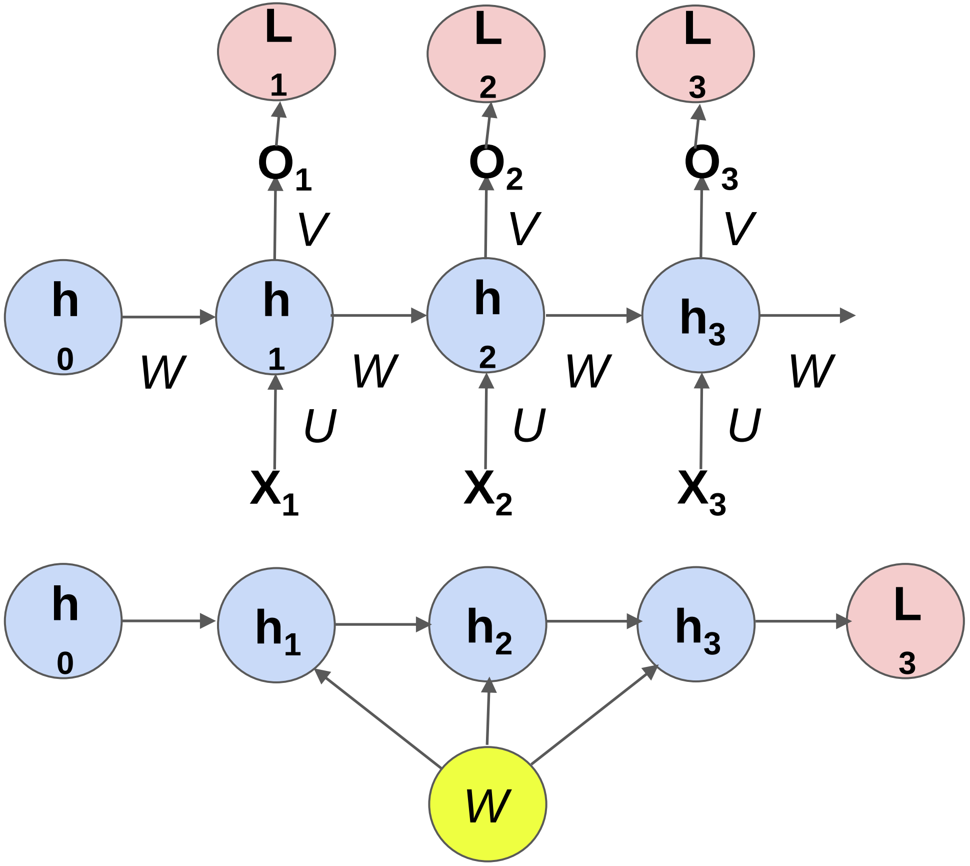


- By back propagation we get,

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

*slide courtesy Mitesh Khapra, CS7015

Training RNNs



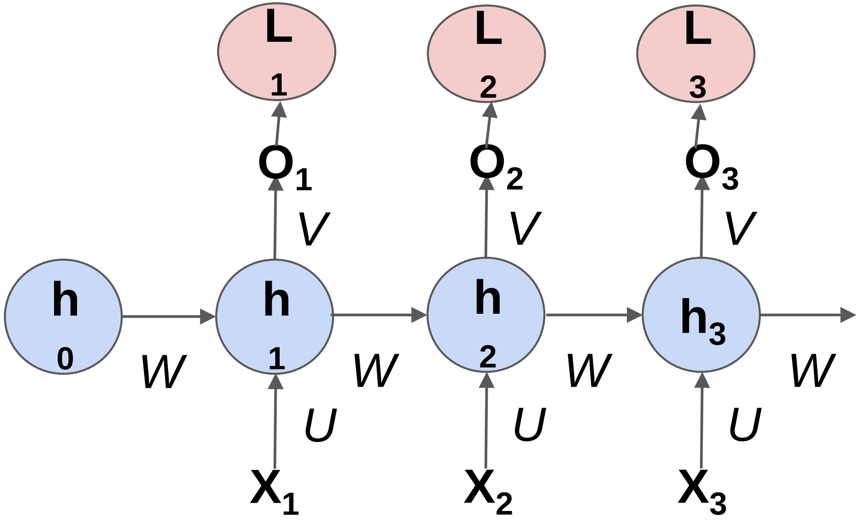
- By back propagation we get,

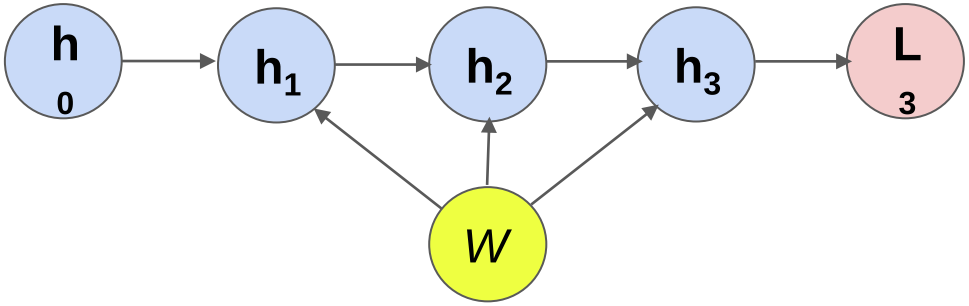
$$\frac{\partial \mathbb{L}_3(\theta)}{\partial W} = \frac{\partial \mathbb{L}_3(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$

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

*slide courtesy Mitesh Khapra, CS7015

Training RNNs





- By back propagation we get,

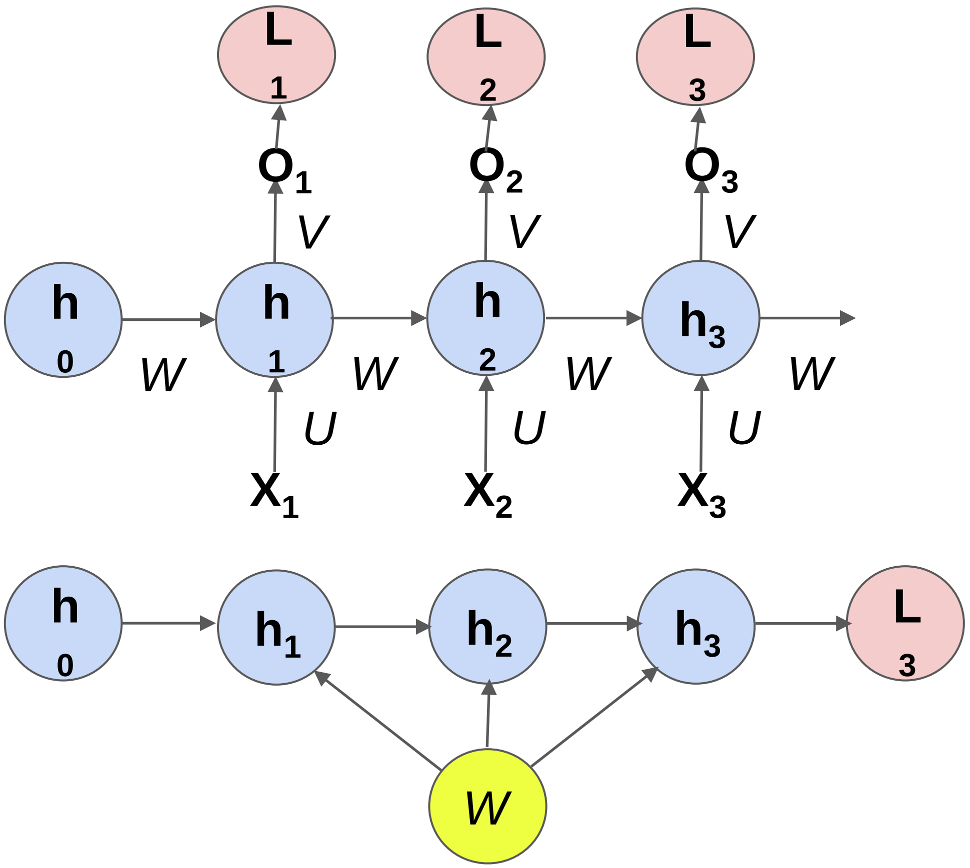
$$\frac{\partial \mathbb{L}_3(\theta)}{\partial W} = \frac{\partial \mathbb{L}_3(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$

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

- But how do we compute, $\frac{\partial h_3}{\partial W}$

*slide courtesy Mitesh Khapra, CS7015

Training RNNs

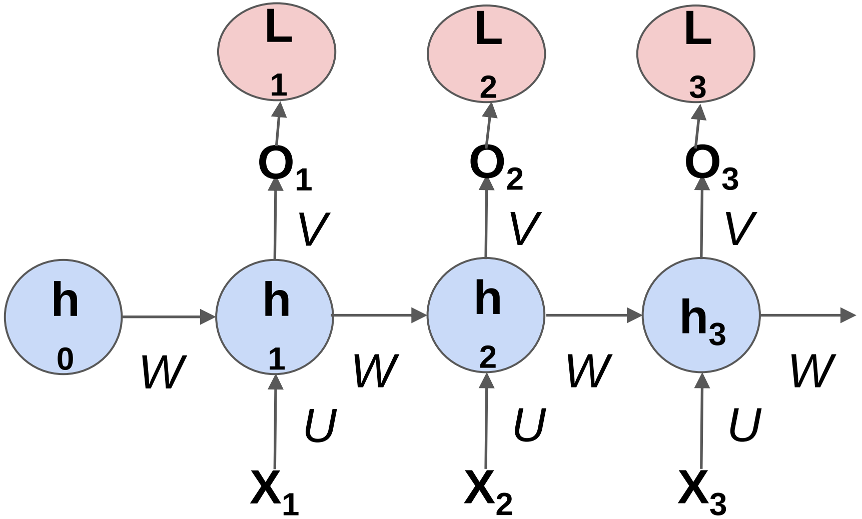


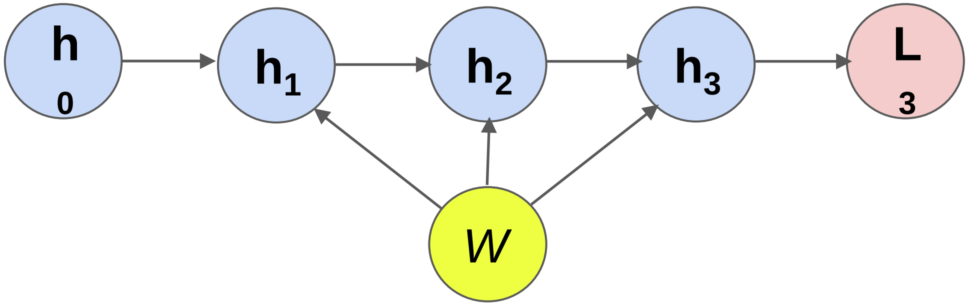
- Recall that,

$$h_t = f\left(W h_{t-1} + U x_t\right)$$
$$h_3 = f\left(W h_2 + U x_3\right)$$

*slide courtesy Mitesh Khapra, CS7015

Training RNNs





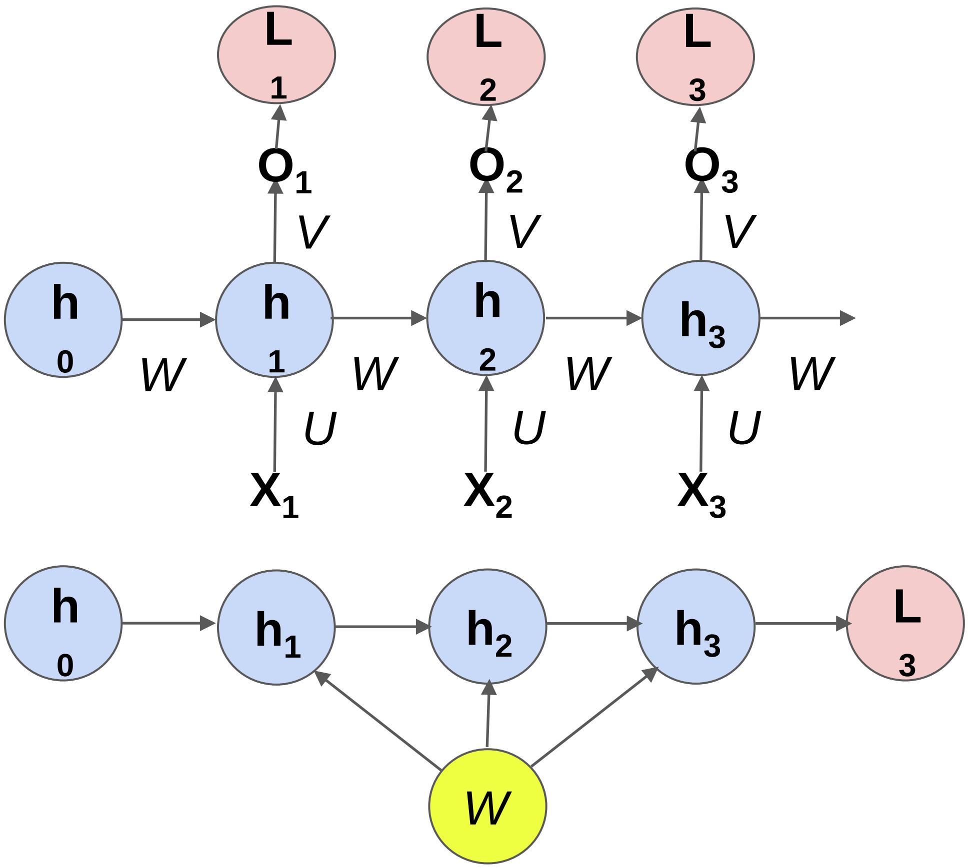
- Recall that,

$$h_t = f\left(Wh_{t-1} + Ux_t\right)$$
$$h_3 = f\left(Wh_2 + Ux_3\right)$$

- Expanding $\frac{\partial h_3}{\partial W}$, we get two parts because $\boldsymbol{h_2}$ also depends on $\boldsymbol{W}$

*slide courtesy Mitesh Khapra, CS7015

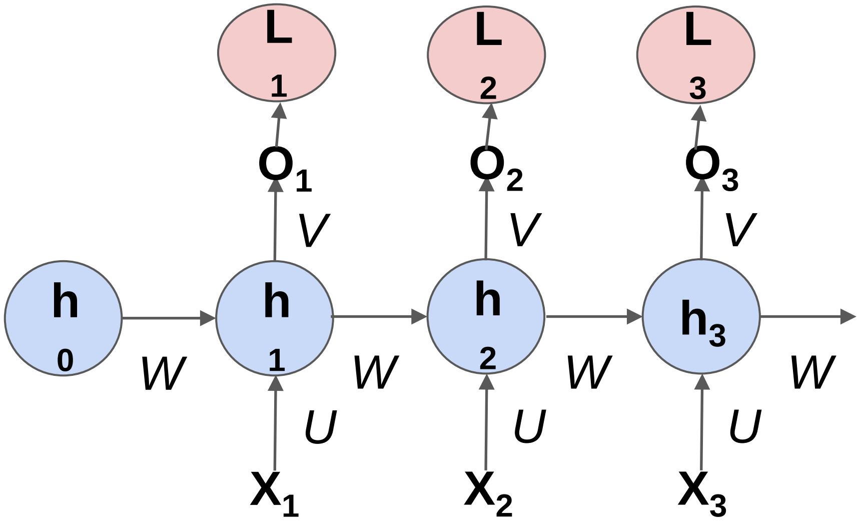
Training RNNs

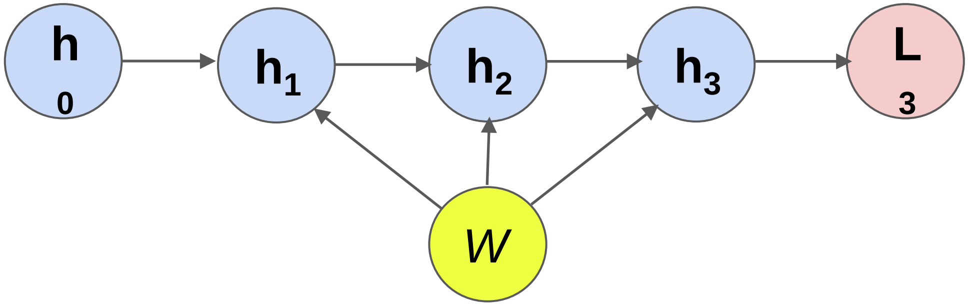


- Recall that,
$$h_t = f\left(W h_{t-1} + U x_t\right)$$
$$h_3 = f\left(W h_2 + U x_3\right)$$
- Expanding $\frac{\partial h_3}{\partial W}$, we get two parts because $\boldsymbol{h_2}$ also depends on $\boldsymbol{W}$
- **Explicit:** $\frac{\partial^+ h_3}{\partial W}$, treating all other inputs as constant

*slide courtesy Mitesh Khapra, CS7015

Training RNNs





- Recall that,

$$h_t = f\left(W h_{t-1} + U x_t\right)$$
$$h_3 = f\left(W h_2 + U x_3\right)$$

- Expanding $\frac{\partial h_3}{\partial W}$, we get two parts because $\boldsymbol{h_2}$ 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 $\boldsymbol{h_3}$ to ***W***

*slide courtesy Mitesh Khapra, CS7015

Training RNNs

- Therefore,

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

*slide courtesy Mitesh Khapra, CS7015

Training RNNs

- Therefore,

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

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

*slide courtesy Mitesh Khapra, CS7015

Training RNNs

- Therefore,

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

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

$$= \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial^+ h_2}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial^+ h_1}{\partial W}$$

(How?)

*slide courtesy Mitesh Khapra, CS7015

Training RNNs

- Therefore,

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

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

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

*slide courtesy Mitesh Khapra, CS7015

Training RNNs

- Therefore,

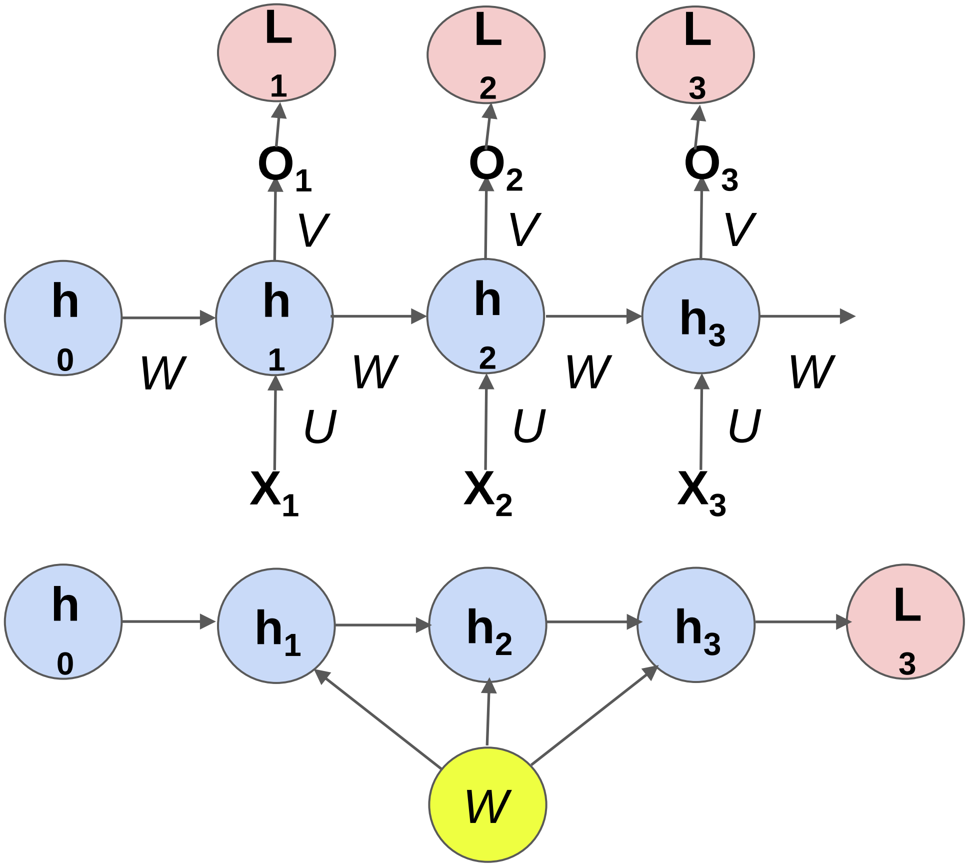
$$\begin{aligned}
\frac{\partial h_3}{\partial W} &= \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W} \\
&= \frac{\partial^+ h_3}{\partial W} + \frac{\partial h_3}{\partial h_2}\left[\frac{\partial^+ h_2}{\partial W} + \frac{\partial h_2}{\partial h_1}\frac{\partial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

- Simplifying,

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

*slide courtesy Mitesh Khapra, CS7015

Training RNNs

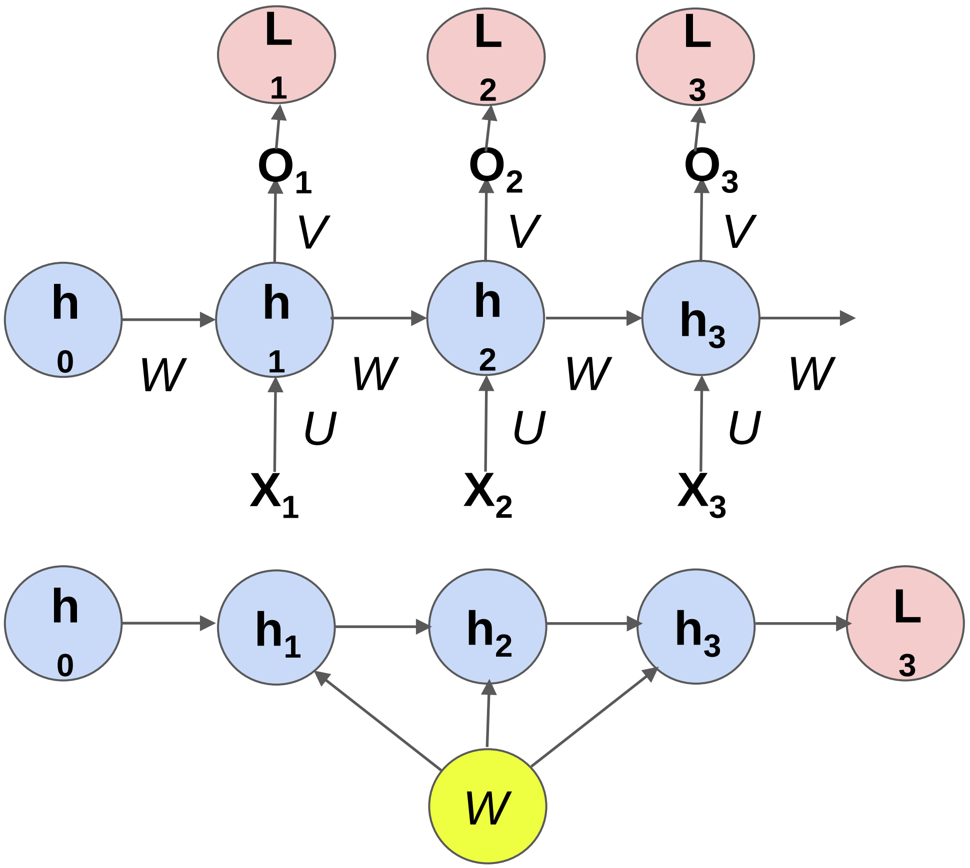


- Finally,

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

*slide courtesy Mitesh Khapra, CS7015

Training RNNs



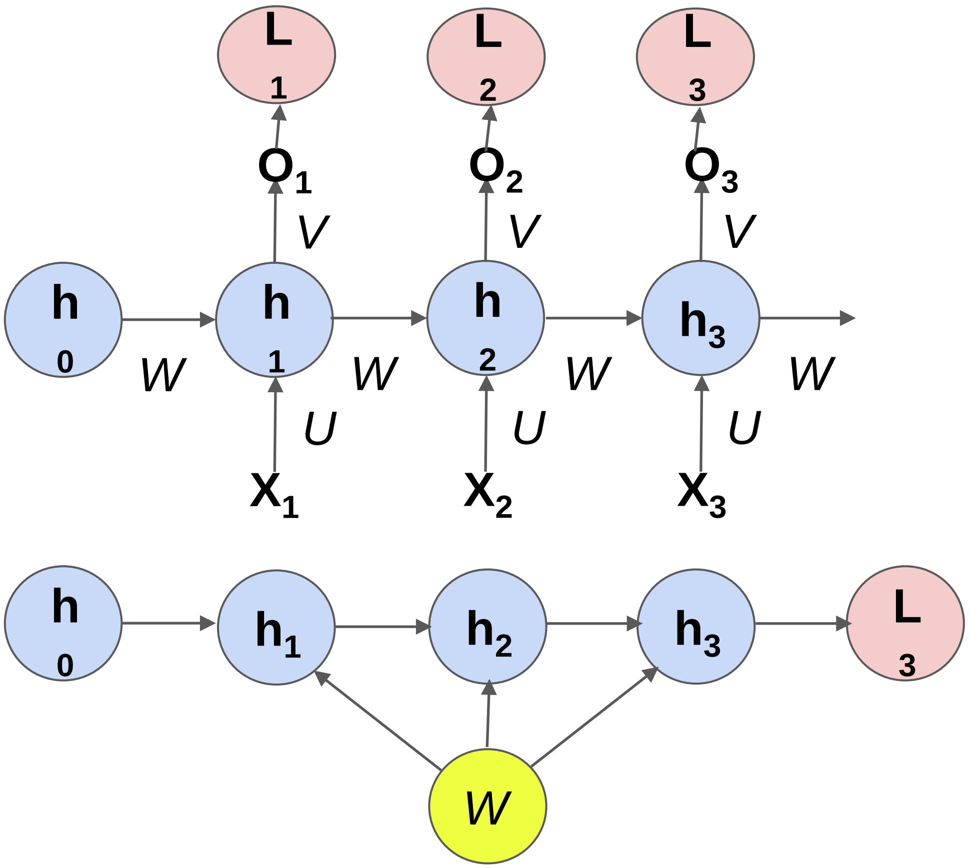
- Finally,

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

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

*slide courtesy Mitesh Khapra, CS7015

Training RNNs



- Finally,

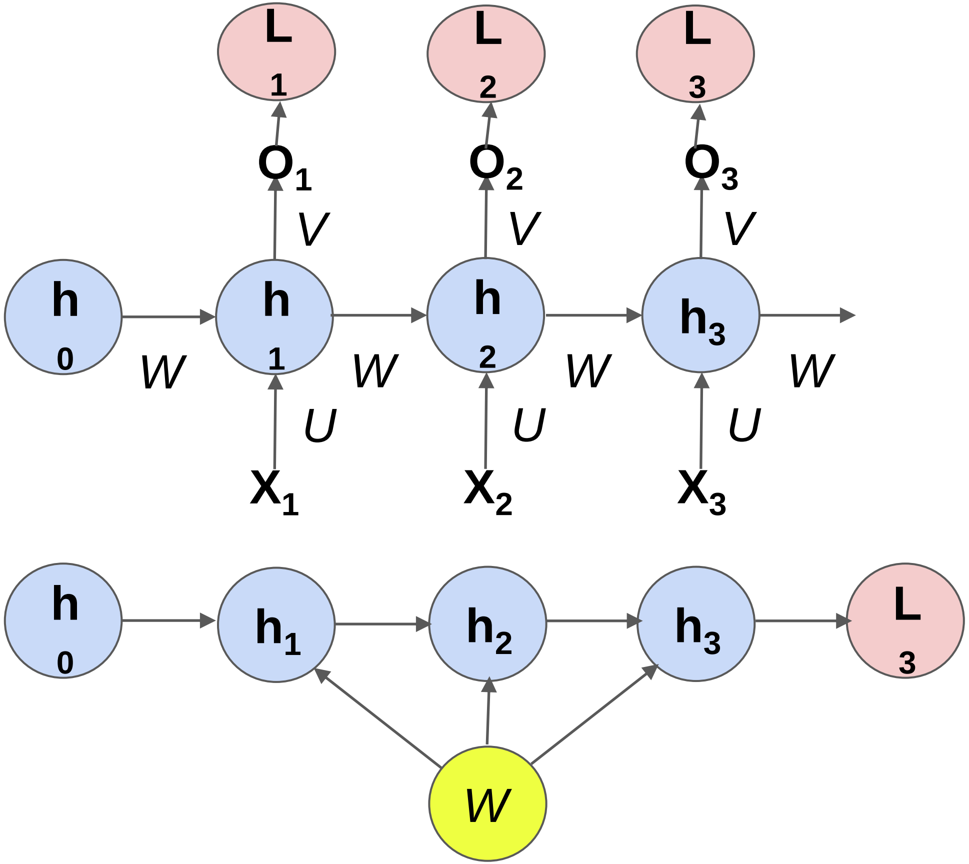
$$\frac{\partial \mathbb{L}_3(\theta)}{\partial W} = \frac{\partial \mathbb{L}_3(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$

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

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

*slide courtesy Mitesh Khapra, CS7015

Training RNNs



- Finally,

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

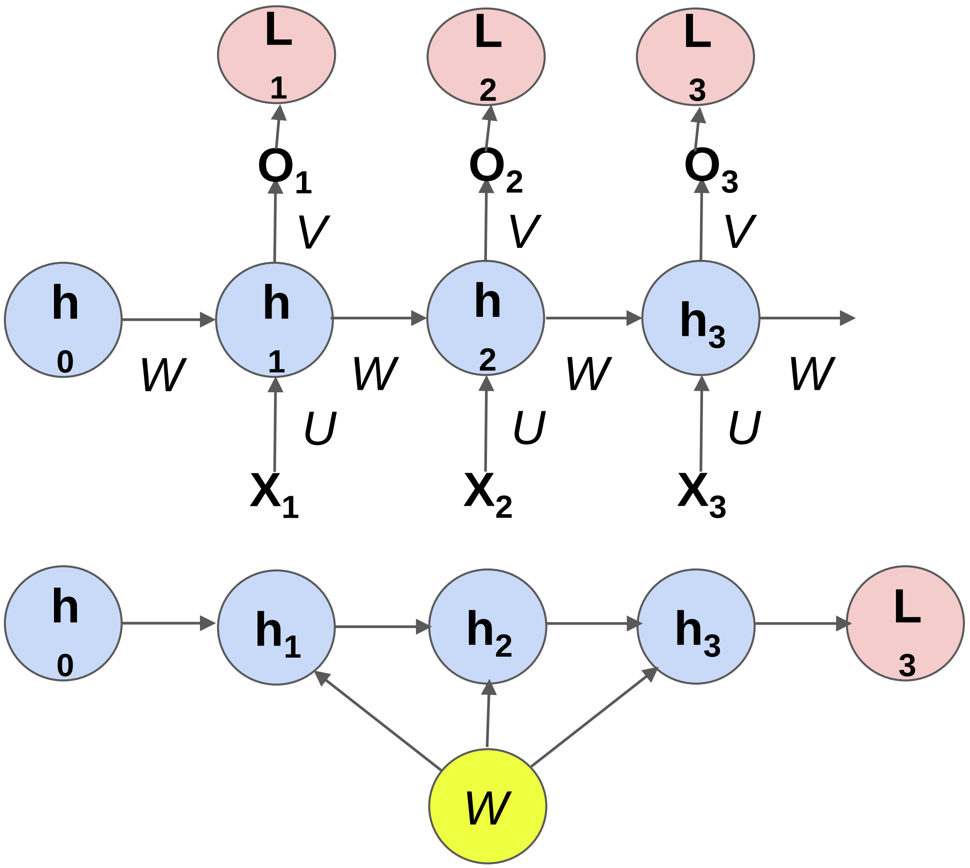
$$\frac{\partial h_3}{\partial W} = \sum_{k=1}^{3} \frac{\partial h_3}{\partial h_k}\frac{\partial^+ h_k}{\partial W}$$

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

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

*slide courtesy Mitesh Khapra, CS7015

Training RNNs



- Finally,

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

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

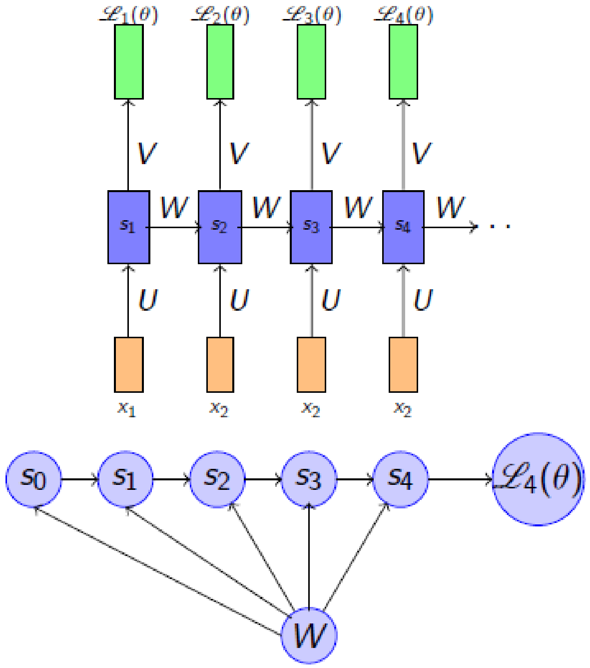
$$\frac{\partial \mathbb{L}_3(\theta)}{\partial W} = \frac{\partial \mathbb{L}_3(\theta)}{\partial h_3}\sum_{k=1}^{3}\frac{\partial h_3}{\partial h_k}\frac{\partial^+ h_k}{\partial W}$$

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

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

*slide courtesy Mitesh Khapra, CS7015

Vanishing Gradients



- Finally we have

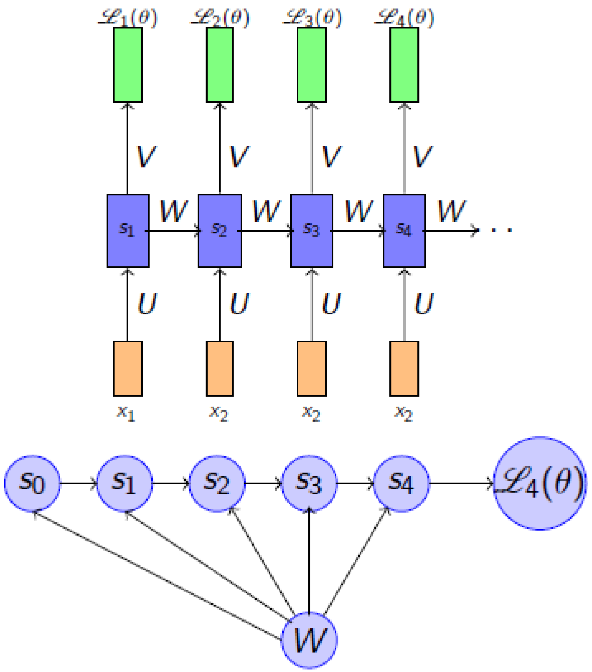
$$\frac{\partial L_4(\theta)}{\partial W} = \frac{\partial L_4 \theta}{\partial s_4} \frac{\partial s_4}{\partial W}$$

$$\frac{\partial s_4}{\partial W} = \sum_{k=1}^{4} \frac{\partial s_4}{\partial s_k} \frac{\partial^+ s_k}{\partial W}$$

$$\therefore \frac{\partial L_t(\theta)}{\partial W} = \frac{\partial L_t(\theta)}{\partial s_t} \sum_{k=1}^{t} \frac{\partial s_t}{\partial s_k} \frac{\partial^+ s_k}{\partial W}$$

- This algorithm is called backpropagation through time (BPTT) as we backpropagate 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

$$\frac{\partial s_t}{\partial s_k} = \frac{\partial s_t}{\partial s_{t-1}} \frac{\partial s_{t-1}}{\partial s_{t-2}} \cdots \frac{\partial s_{k+1}}{\partial s_k}$$
$$= \prod_{j=k}^{t-1} \frac{\partial s_{j+1}}{\partial s_j}$$

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

Vanishing Gradients

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

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \dots a_{jd},]$$
$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \dots \sigma(a_{jd})]$$

$$a_j = Ws_{j-1} + b$$
$$s_j = \sigma(a_j)$$
$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}}$$

$$\frac{\partial s_j}{\partial a_j} = \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}$$

Vanishing Gradients

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

$$a_j = W s_{j-1} + b$$
$$s_j = \sigma(a_j)$$
$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}}$$

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \ldots a_{jd},]$$
$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \ldots \sigma(a_{jd})]$$

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

Vanishing Gradients

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

$$a_j = W s_{j-1} + b$$
$$s_j = \sigma(a_j)$$
$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}}$$

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \ldots a_{jd},]$$
$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \ldots \sigma(a_{jd})]$$

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

$$= \begin{bmatrix} \sigma'(a_{j1}) & 0 & 0 & 0 \\ & & & \\ & & & \end{bmatrix}$$

Vanishing Gradients

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

$$a_j = W s_{j-1} + b$$
$$s_j = \sigma(a_j)$$
$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}}$$

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \ldots a_{jd},]$$
$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \ldots \sigma(a_{jd})]$$

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

Vanishing Gradients

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

$$a_j = Ws_{j-1} + b$$
$$s_j = \sigma(a_j)$$
$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j}\frac{\partial a_j}{\partial s_{j-1}}$$

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \dots a_{jd},]$$
$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \dots \sigma(a_{jd})]$$

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

Vanishing Gradients

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \ldots a_{jd},]$$
$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \ldots \sigma(a_{jd})]$$

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

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

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

Vanishing Gradients

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \ldots a_{jd},]$$
$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \ldots \sigma(a_{jd})]$$

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

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

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

- 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)

Vanishing Gradients

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

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

Vanishing Gradients

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

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

Vanishing Gradients

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

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

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

Vanishing Gradients

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

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

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

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

Vanishing Gradients

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

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

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

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

Vanishing Gradients

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

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

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

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

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

- 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

Implementation Details

$$\underbrace{\frac{\partial L_t(\theta)}{\partial W}}_{\in \mathbb{R}^{d\times d}} = \underbrace{\frac{\partial L_t(\theta)}{\partial s_t}}_{\in \mathbb{R}^{1\times d}} \sum_{k=1}^{t} \underbrace{\frac{\partial s_t}{\partial s_k}}_{\in \mathbb{R}^{d\times d}} \underbrace{\frac{\partial^+ s_k}{\partial W}}_{\in \mathbb{R}^{d\times d\times d}}$$

$$\underbrace{\frac{\partial L_t(\theta)}{\partial W}}_{\in \mathbb{R}^{d\times d}} = \underbrace{\frac{\partial L_t(\theta)}{\partial s_t}}_{\in \mathbb{R}^{1\times d}} \sum_{k=1}^{t} \underbrace{\frac{\partial s_t}{\partial s_k}}_{\in \mathbb{R}^{d\times d}} \underbrace{\frac{\partial^+ s_k}{\partial W}}_{\in \mathbb{R}^{d\times d\times d}}$$

- We know how to compute $\frac{\partial L_t(\theta)}{\partial s_t}$ (derivative of $L_t(\theta)$ (scalar) w.r.t. last hidden layer (vector)) using backpropagation
- We just saw a formula for $\frac{\partial s_t}{\partial s_k}$ (derivative of a vector w.r.t. a vector)
- $\frac{\partial^+ s_k}{\partial W}$ is a tensor $\in \mathbb{R}^{d\times d\times d}$, the derivative of a vector $\in \mathbb{R}^d$ w.r.t. a matrix $\in \mathbb{R}^{d\times d}$
- How do we compute $\frac{\partial^+ s_k}{\partial W}$? Let us see

- We just look at one element of this $\frac{\partial^+ s_k}{\partial W}$ tensor
- $\frac{\partial^+ s_{kp}}{\partial W_{qr}}$ is the (p, q, r)-th element of the 3d tensor

  $a_k = W s_{k-1} + b$

  $s_k = \sigma(a_k)$

$$
\begin{aligned}
a_k &= W s_{k-1} \\
\begin{bmatrix} a_{k1} \\ a_{k2} \\ \vdots \\ a_{kp} \\ \vdots \\ a_{kd} \end{bmatrix} &= \begin{bmatrix} W_{11} & W_{12} & \dots & W_{1d} \\ \vdots & \vdots & \vdots & \vdots \\ W_{p1} & W_{p2} & \dots & W_{pd} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} s_{k-1,1} \\ s_{k-1,2} \\ \vdots \\ s_{k-1,p} \\ \vdots \\ s_{k-1,d} \end{bmatrix} \\
a_{kp} &= \sum_{i=1}^{d} W_{pi} s_{k-1,i} \\
s_{kp} &= \sigma(a_{kp}) \\
\frac{\partial s_{kp}}{\partial W_{qr}} &= \frac{\partial s_{kp}}{\partial a_{kp}} \frac{\partial a_{kp}}{\partial W_{qr}} \\
&= \sigma'(a_{kp}) \frac{\partial a_{kp}}{\partial W_{qr}}
\end{aligned}
$$

$$a_k = Ws_{k-1}$$

$$\begin{bmatrix} a_{k1} \\ a_{k2} \\ \vdots \\ a_{kp} \\ \vdots \\ a_{kd} \end{bmatrix} = \begin{bmatrix} W_{11} & W_{12} & \dots & W_{1d} \\ \vdots & \vdots & \vdots & \vdots \\ W_{p1} & W_{p2} & \dots & W_{pd} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} s_{k-1,1} \\ s_{k-1,2} \\ \vdots \\ s_{k-1,p} \\ \vdots \\ s_{k-1,d} \end{bmatrix}$$

$$a_{kp} = \sum_{i=1}^{d} W_{pi} s_{k-1,i}$$

$$s_{kp} = \sigma(a_{kp})$$

$$\frac{\partial s_{kp}}{\partial W_{qr}} = \frac{\partial s_{kp}}{\partial a_{kp}} \frac{\partial a_{kp}}{\partial W_{qr}}$$

$$= \sigma'(a_{kp}) \frac{\partial a_{kp}}{\partial W_{qr}}$$

$$\frac{\partial a_{kp}}{\partial W_{qr}} = \frac{\partial \sum_{i=1}^{d} W_{pi} s_{k-1,i}}{\partial W_{qr}}$$

$$= s_{k-1,i} \quad \text{if} \quad p = q$$

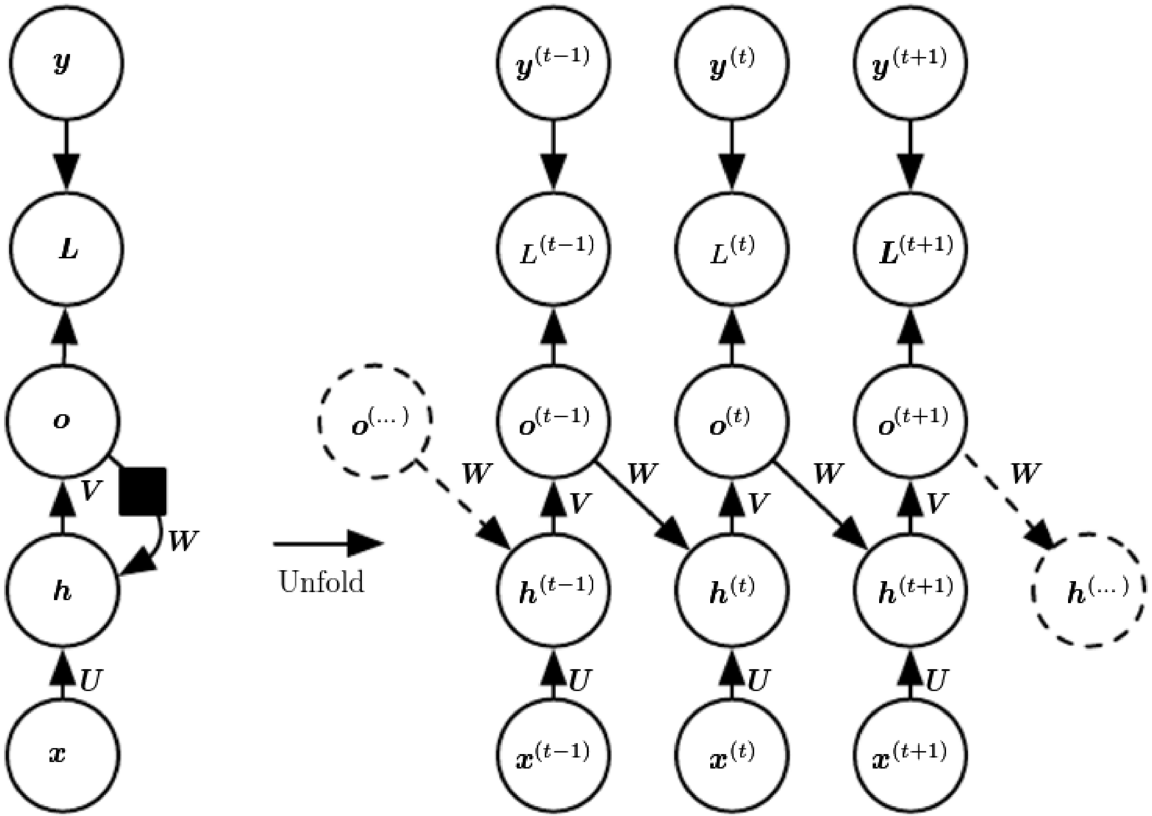
$$= 0 \quad \text{otherwise}$$

$$\frac{\partial s_{kp}}{\partial W_{qr}} = \sigma'(a_{kp}) s_{k-1,r} \quad \text{if} \quad p = q$$

$$= 0 \quad \text{otherwise}$$

Variations of RNNs

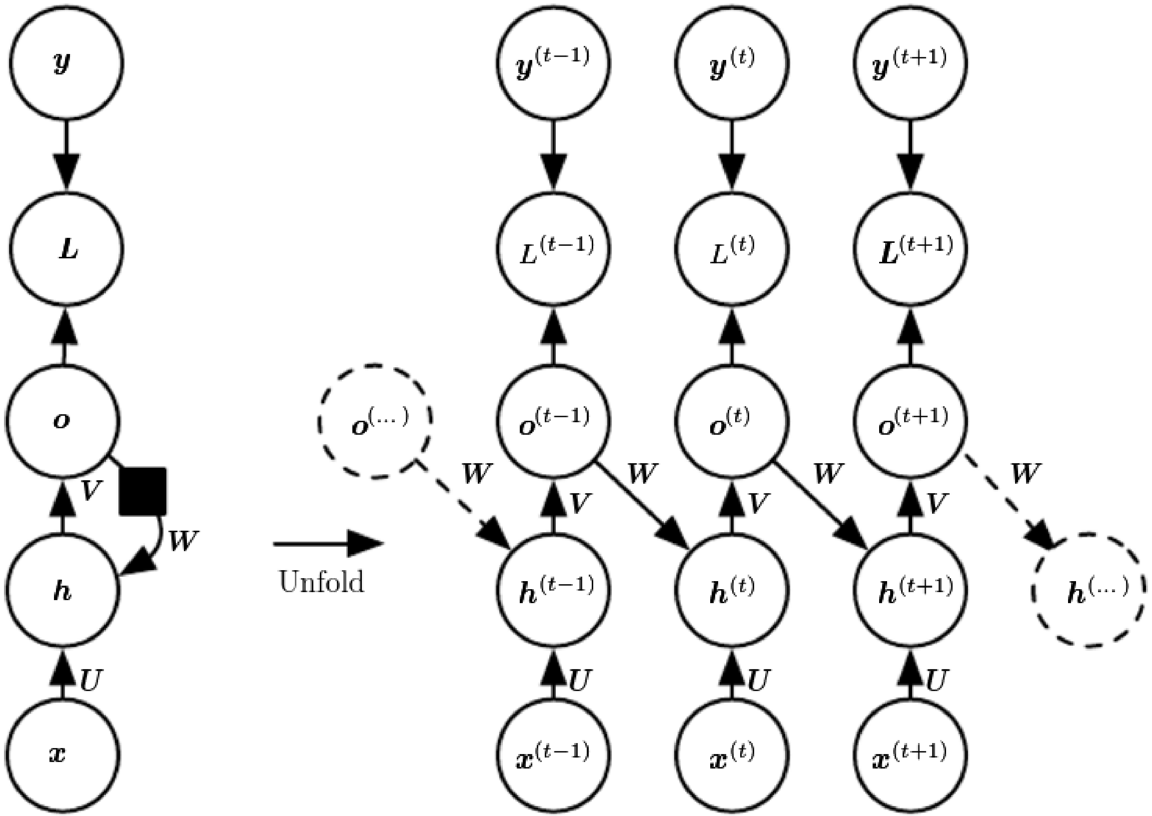
Variations of RNNs (Output recurrence)



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

*image courtesy Deep Learning book, Ian, Yoshua, Aaron

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?

*image courtesy Deep Learning book, Ian, Yoshua, Aaron

Output recurrence (or Teacher forcing)

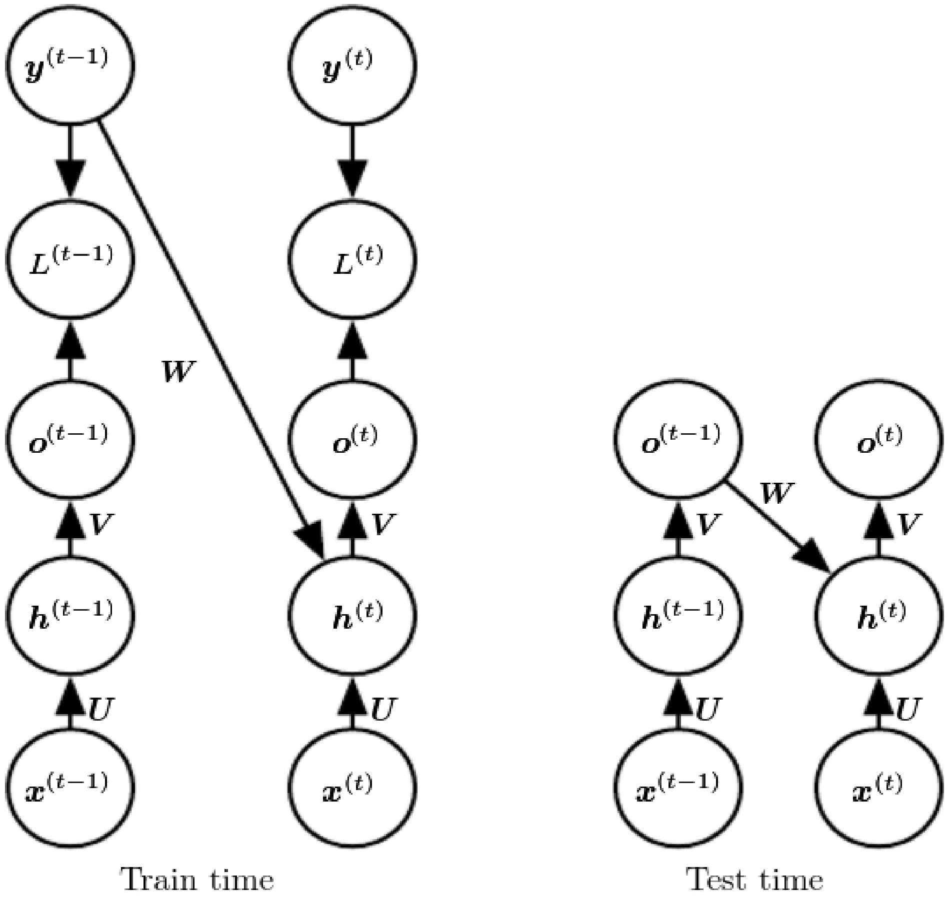
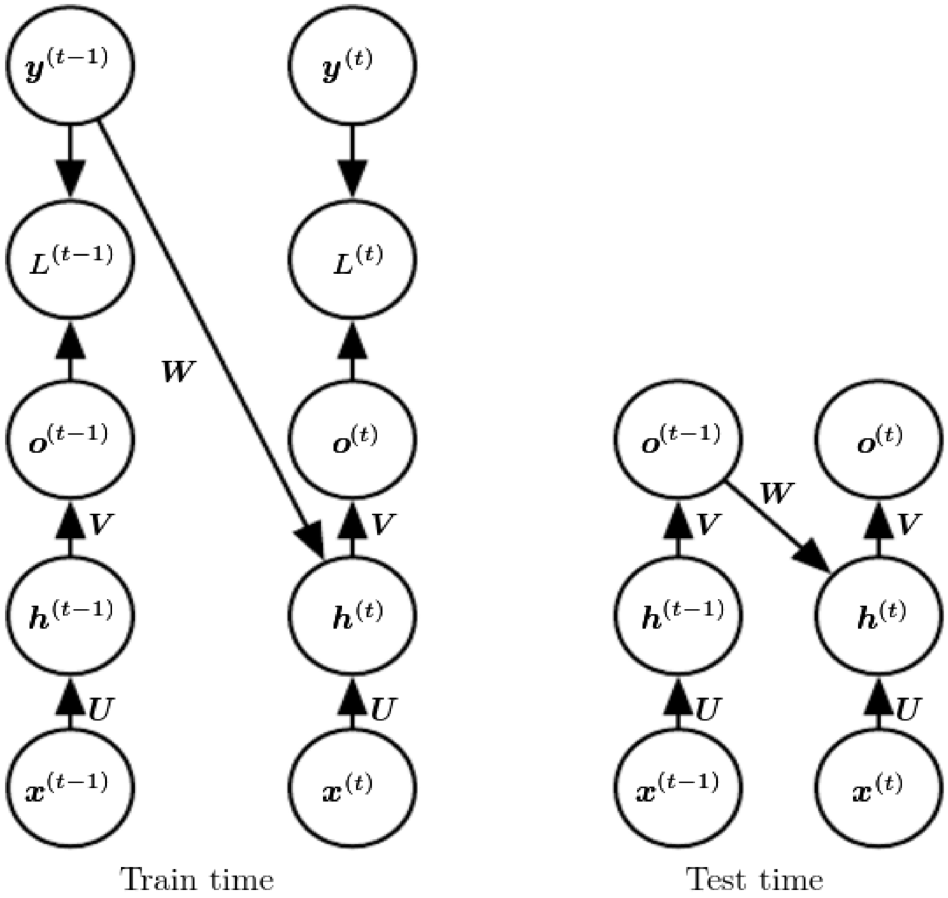


*image courtesy Deep Learning book, Ian, Yoshua, Aaron

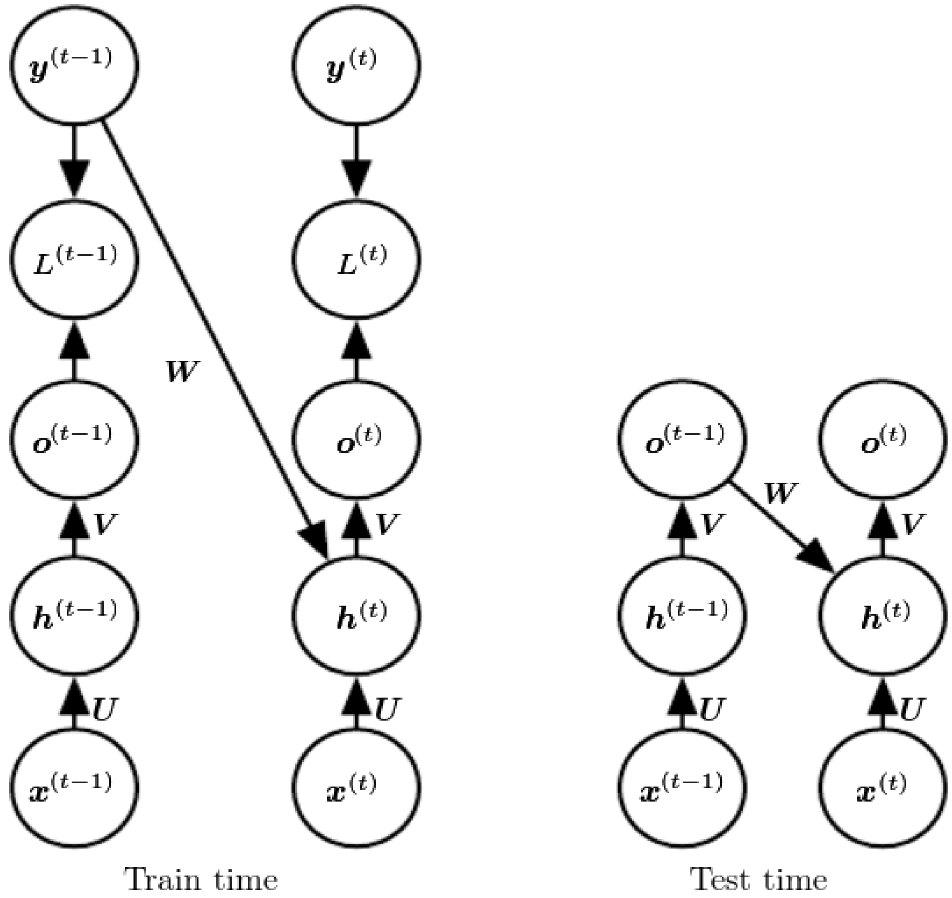
Output recurrence (or Teacher forcing)



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

*image courtesy Deep Learning book, Ian, Yoshua, Aaron

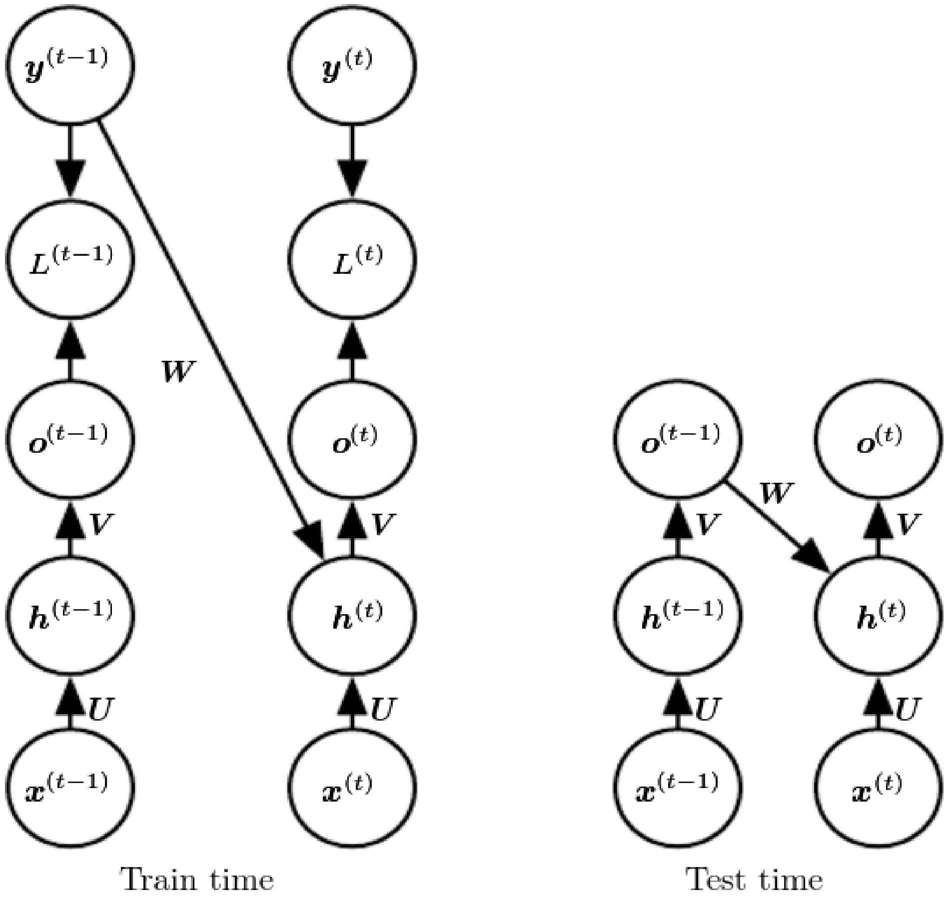
Output recurrence (or Teacher forcing)



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

*image courtesy Deep Learning book, Ian, Yoshua, Aaron

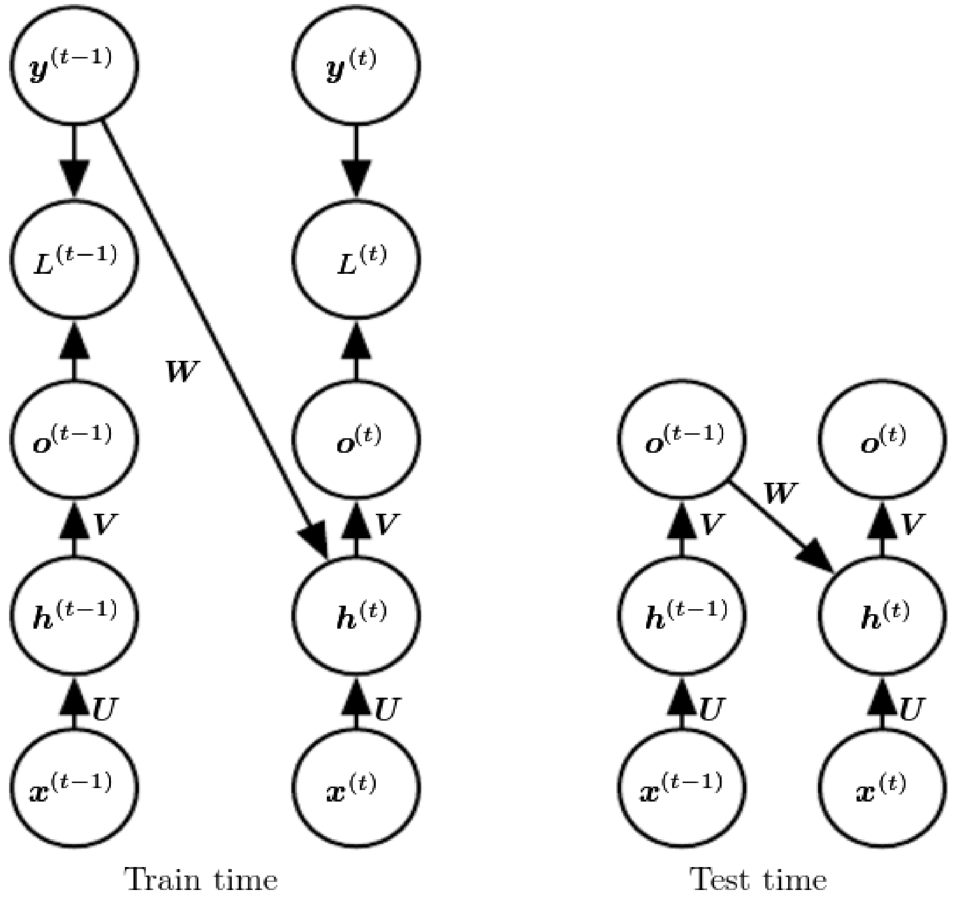
Output recurrence (or Teacher forcing)



- 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

*image courtesy Deep Learning book, Ian, Yoshua, Aaron

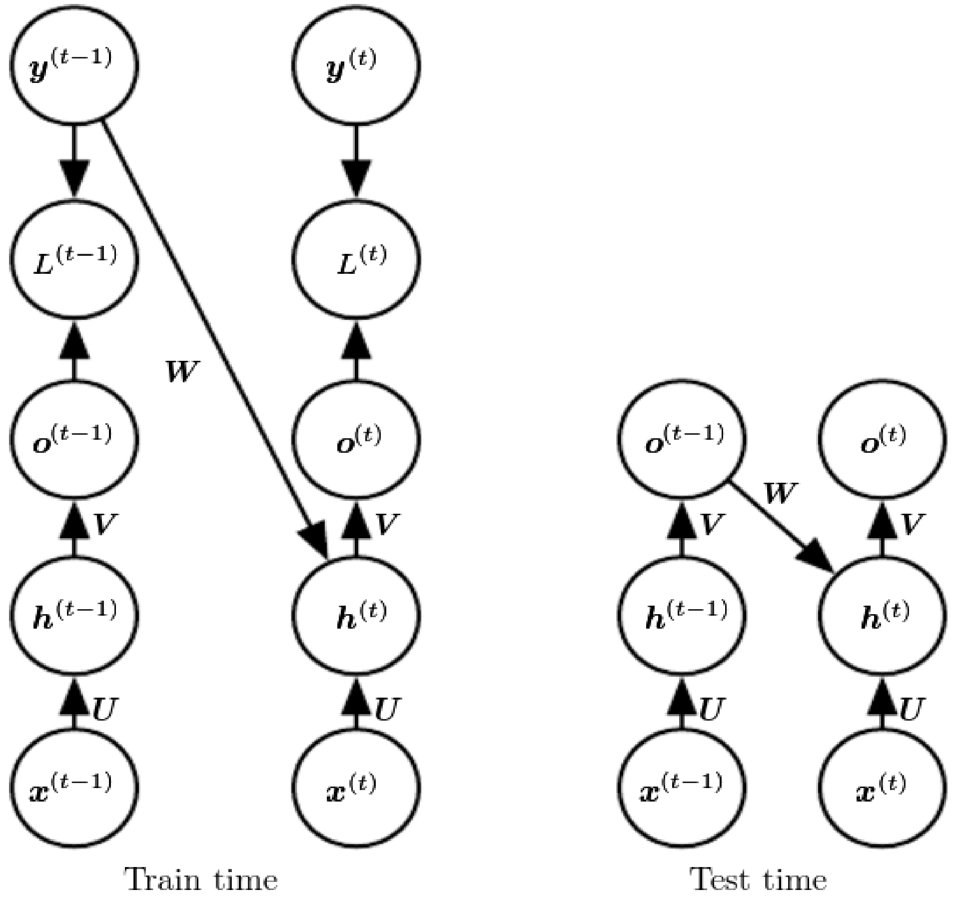
Output recurrence (or Teacher forcing)



- 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

*image courtesy Deep Learning book, Ian, Yoshua, Aaron

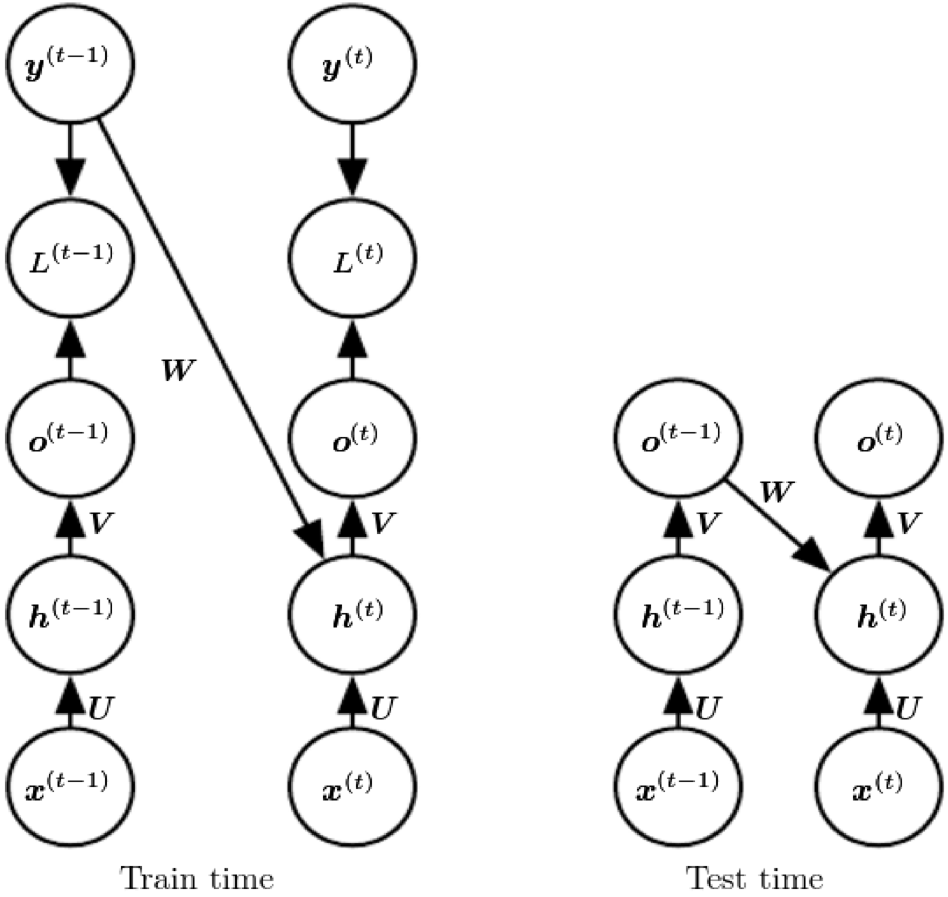
Output recurrence (or Teacher forcing)



- So why do we study it?

*image courtesy Deep Learning book, Ian, Yoshua, Aaron

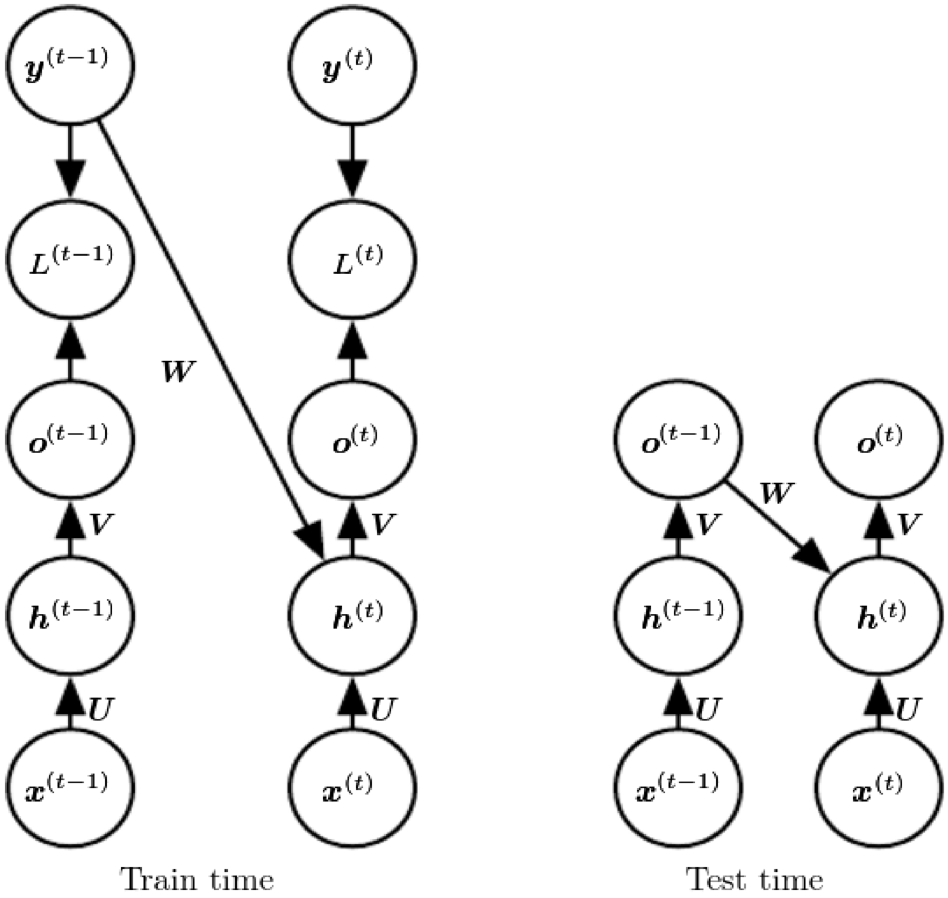
Output recurrence (or Teacher forcing)



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

*image courtesy Deep Learning book, Ian, Yoshua, Aaron

Output recurrence (or Teacher forcing)



- 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?)

*image courtesy Deep Learning book, Ian, Yoshua, Aaron

Next coming up…

(i) Problems with RNNs

(ii) Different types of RNNs

(iii) Approximations in BPTT