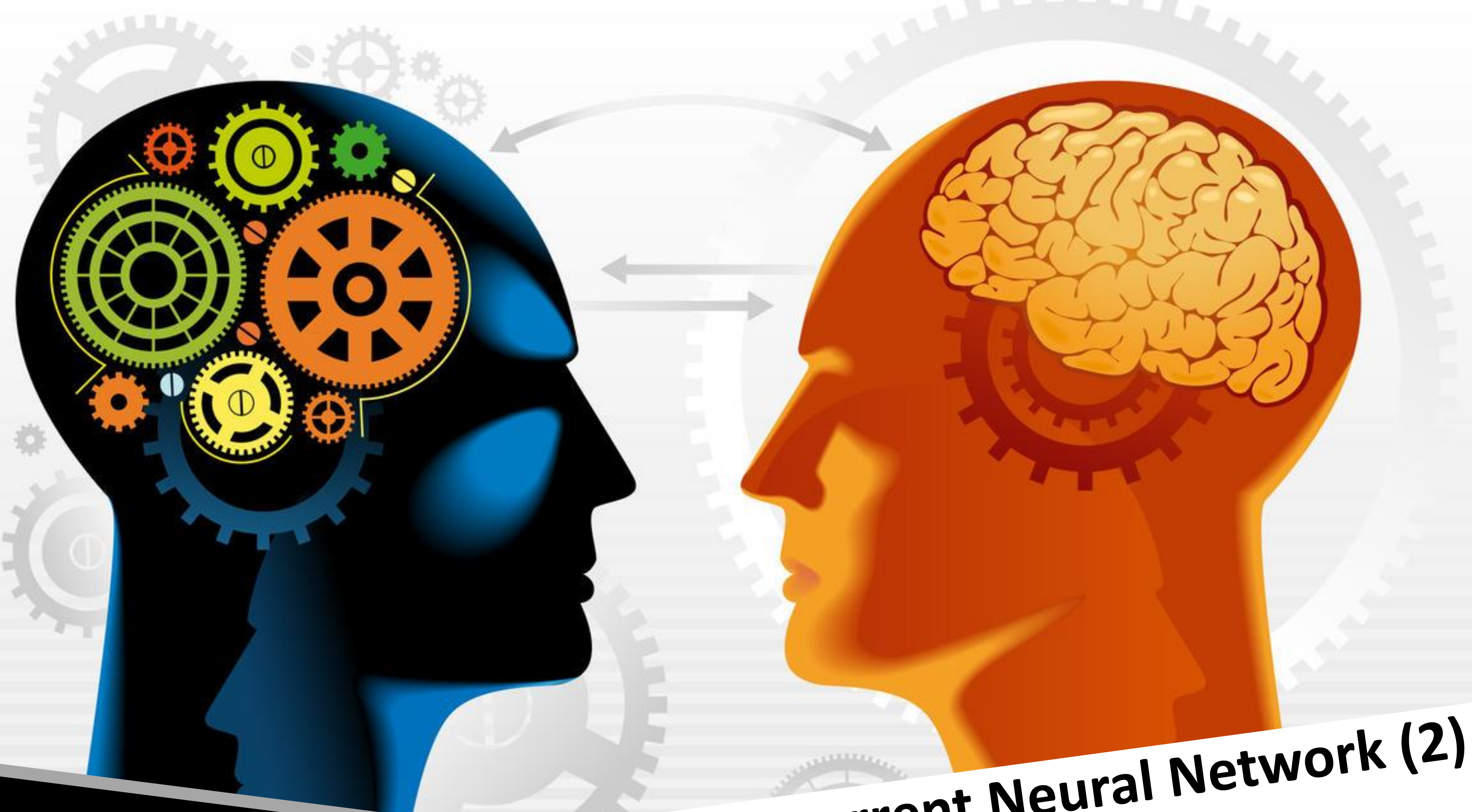


Recurrent Neural Network (2)

Nov 10th, 2016

Applied Deep Learning

YUN-NUNG (VIVIAN) CHEN

WWW.CSIE.NTU.EDU.TW/~YVCHEN/F105-ADL

National Taiwan University

Slide credit from Hung-Yi Lee & Richard Socher

Review

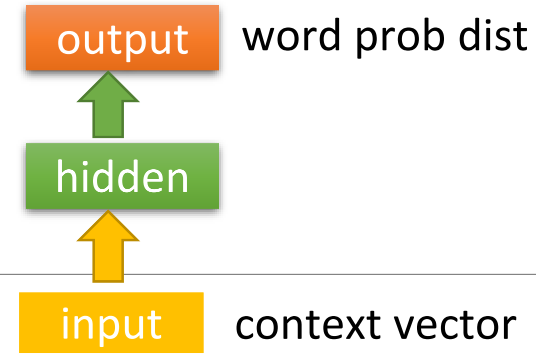
Recurrent Neural Network

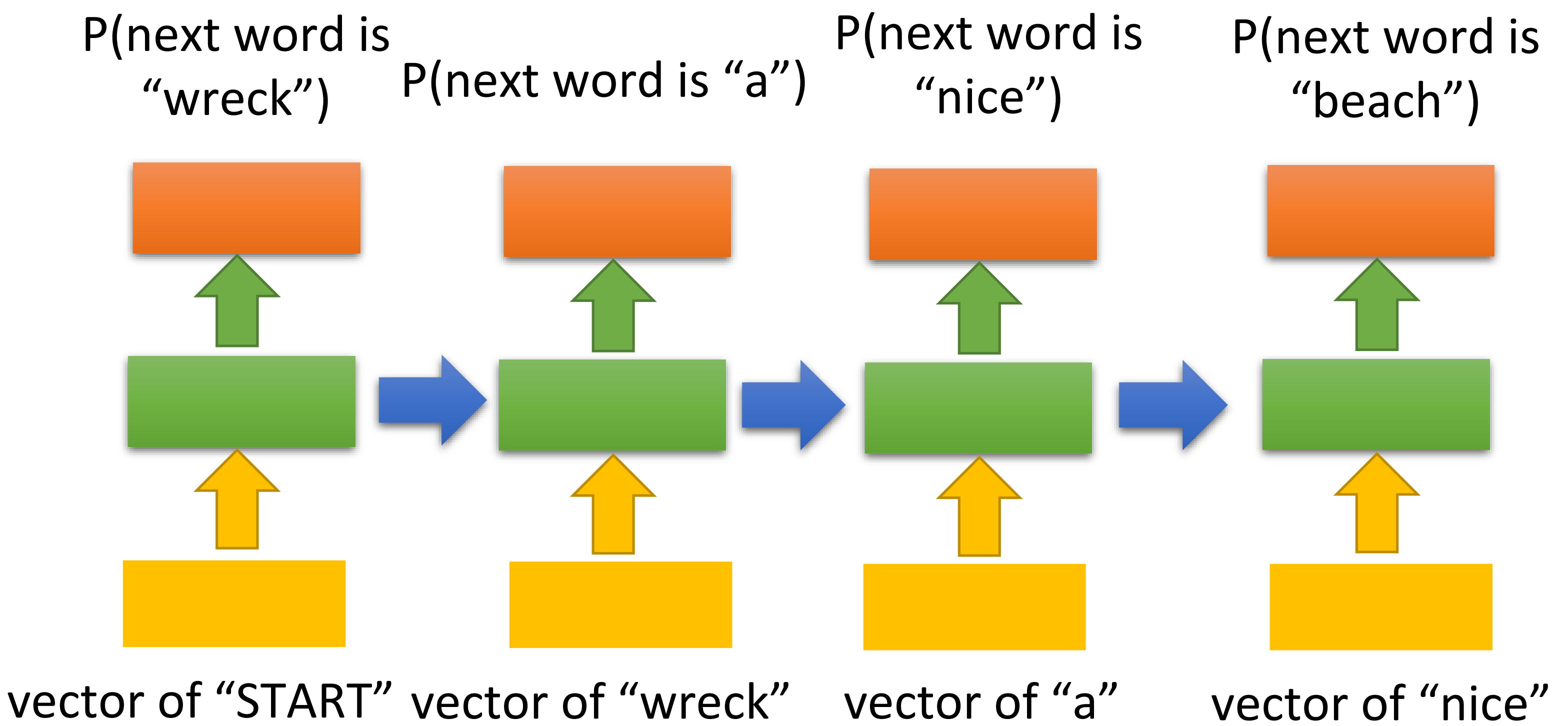
Recurrent Neural Network

Idea: condition the neural network on all previous words and tie the weights at each time step

Assumption: temporal information matters

RNN Language Modeling





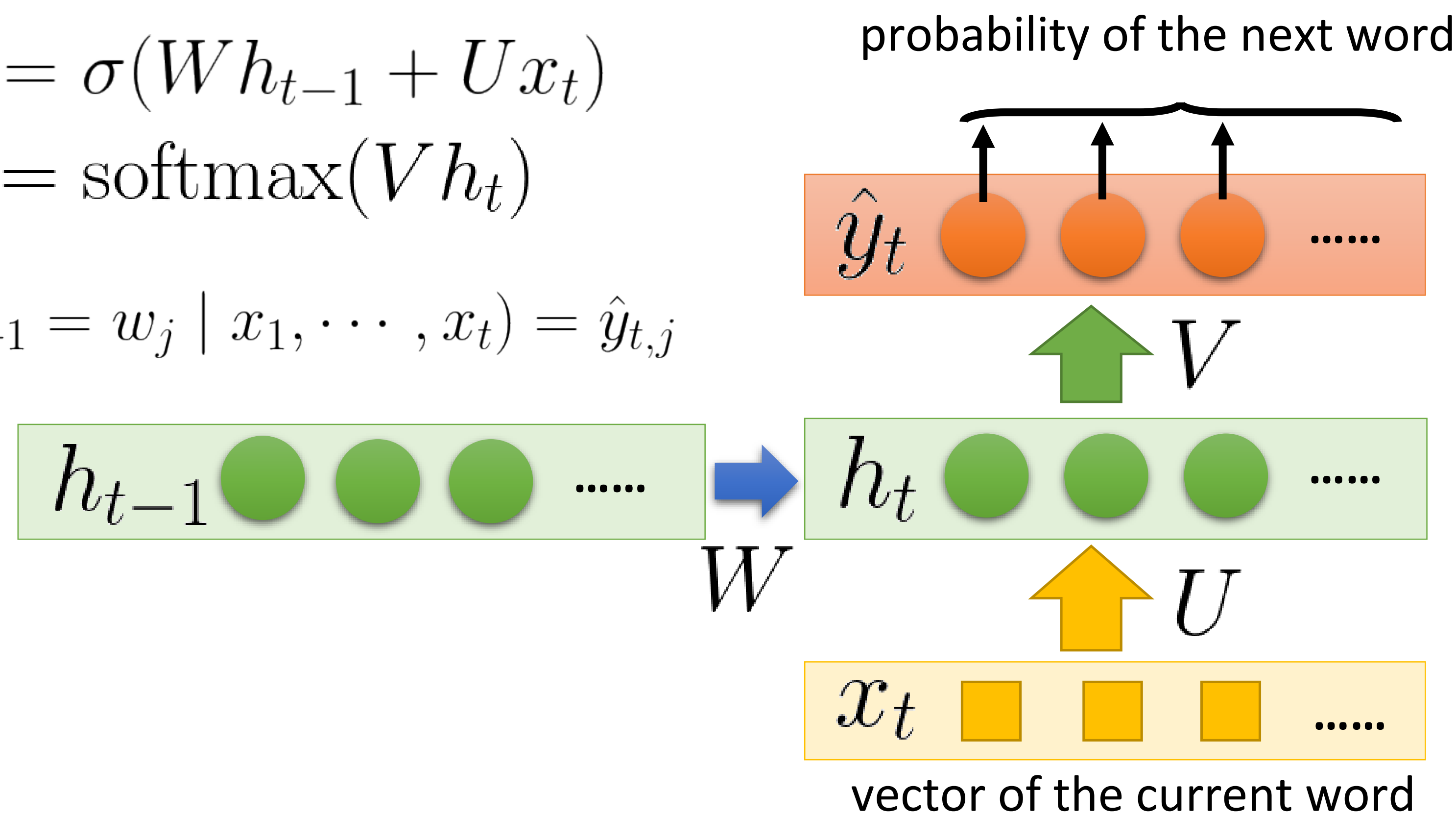
Idea: pass the information from the previous hidden layer to leverage all contexts

RNNLM Formulation

At each time step,

$$h_t = \sigma(W h_{t-1} + U x_t)$$
$$\hat{y}_t = \mathrm{softmax}(V h_t)$$

$$P(x_{t+1} = w_j \mid x_1, \cdots, x_t) = \hat{y}_{t,j}$$

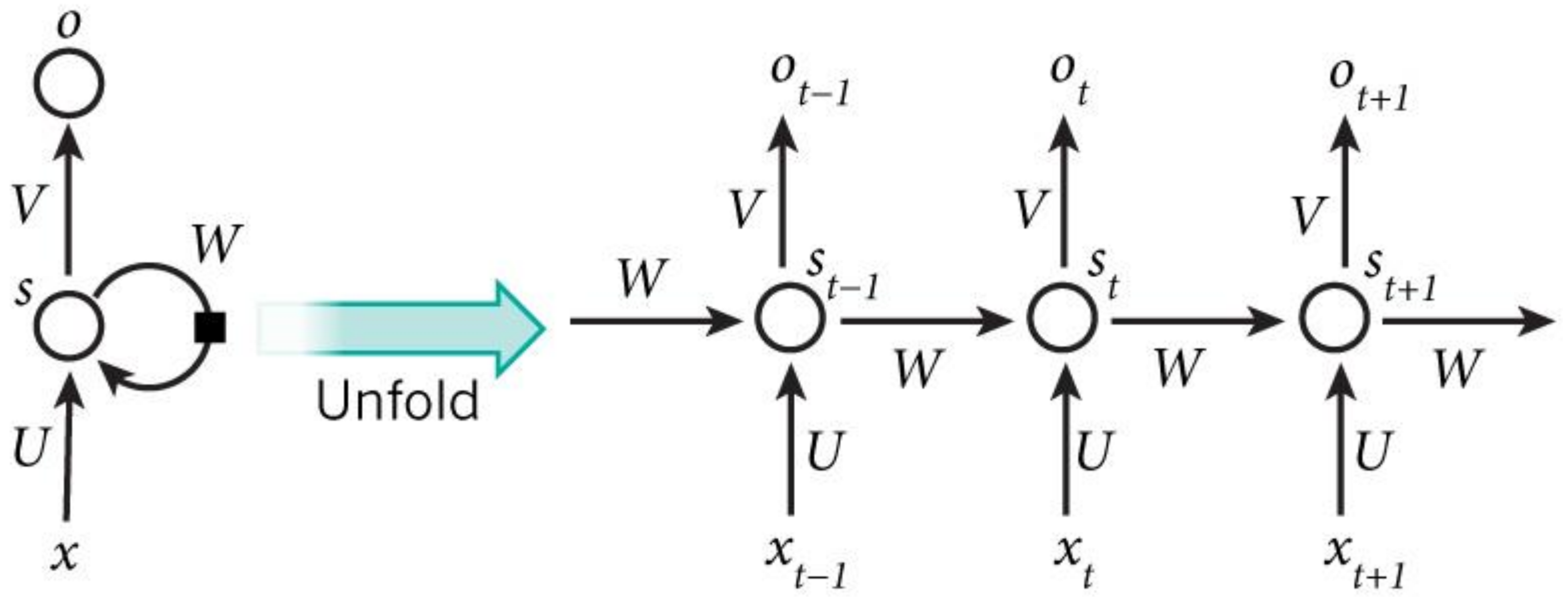


Recurrent Neural Network Definition

$$s_t = \sigma(W s_{t-1} + U x_t)$$

$\sigma(\cdot)$: tanh, ReLU

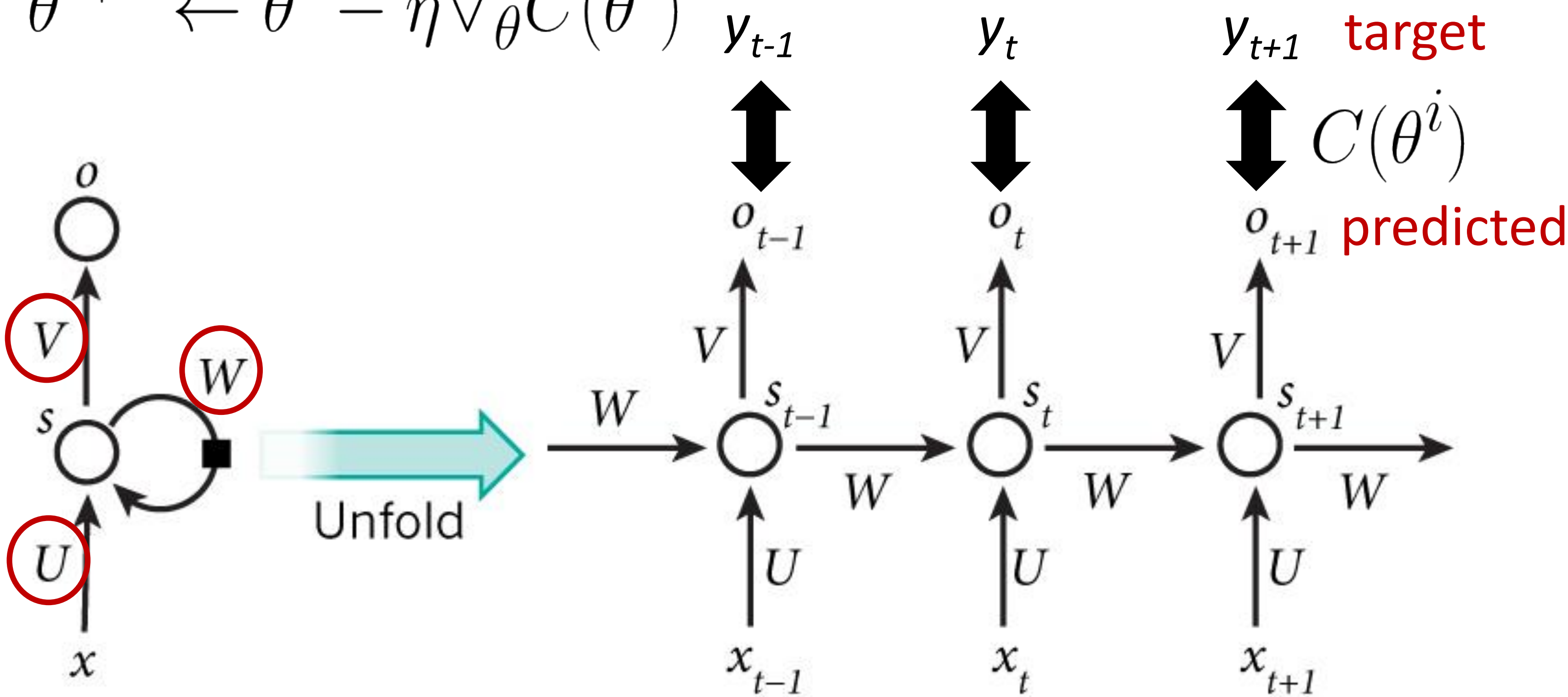
$$o_t = \mathrm{softmax}(V s_t)$$



Model Training

All model parameters $\theta = \{U, V, W\}$ can be updated by

$$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_\theta C(\theta^i)$$



Outline

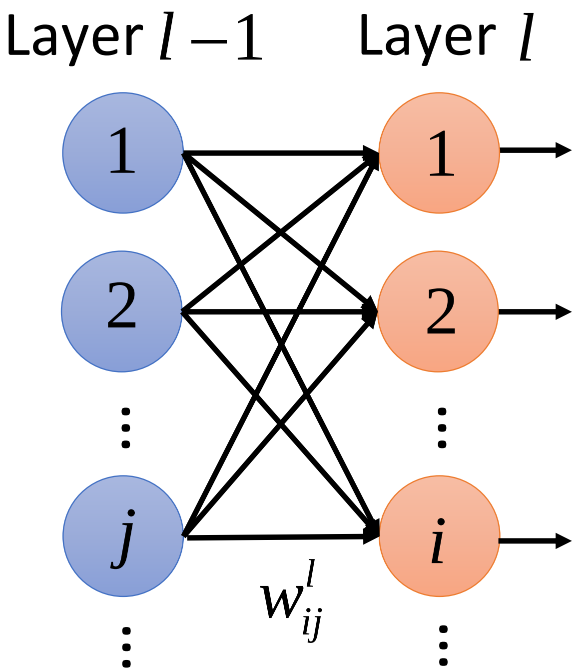
Language Modeling
- N-gram Language Model
- Feed-Forward Neural Language Model
- Recurrent Neural Network Language Model (RNNLM)

Recurrent Neural Network
- Definition
- **Training via Backpropagation through Time (BPTT)**
- Training Issue

Applications
- Sequential Input
- Sequential Output
  - Aligned Sequential Pairs (Tagging)
  - Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)

Backpropagation

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \boxed{\frac{\partial C(\theta)}{\partial z_i^l}}\,\boxed{\frac{\partial z_i^l}{\partial w_{ij}^l}}$$



δ_i^l Error signal

$$\begin{cases} a_j^{l-1} & l > 1 \\ x_j & l = 1 \end{cases}$$

Backward Pass

$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$
$$\delta^{L-1} = \sigma'(z^{L-1}) \odot (W^L)^T \delta^L$$
$$\vdots$$
$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$
$$\vdots$$

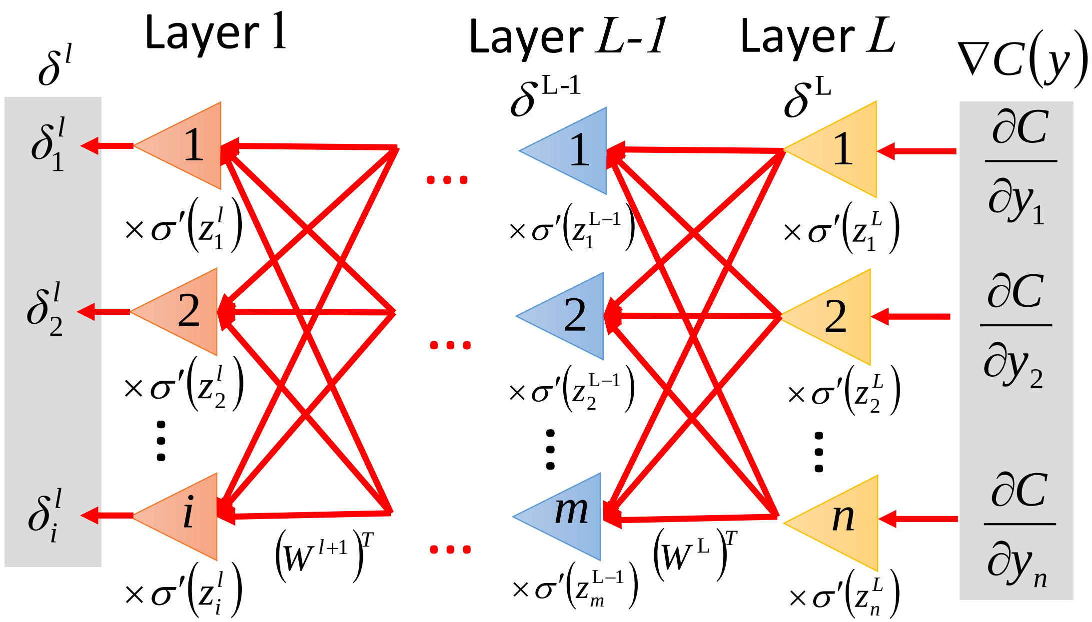
Forward Pass

$$z^1 = W^1 x + b^1$$
$$a^1 = \sigma(z^1)$$
$$\vdots$$
$$z^l = W^l a^{l-1} + b^l$$
$$a^l = \sigma(z^l)$$
$$\vdots$$

Backpropagation

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \boxed{\frac{\partial C(\theta)}{\partial z_i^l}} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

δ_i^l Error signal

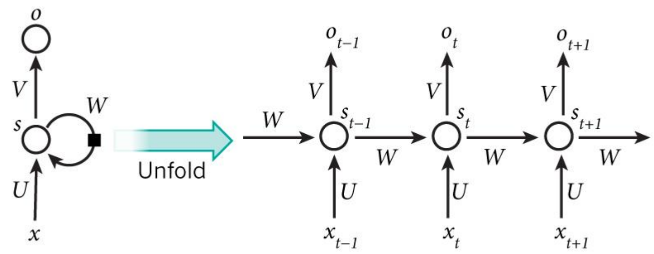


Backward Pass

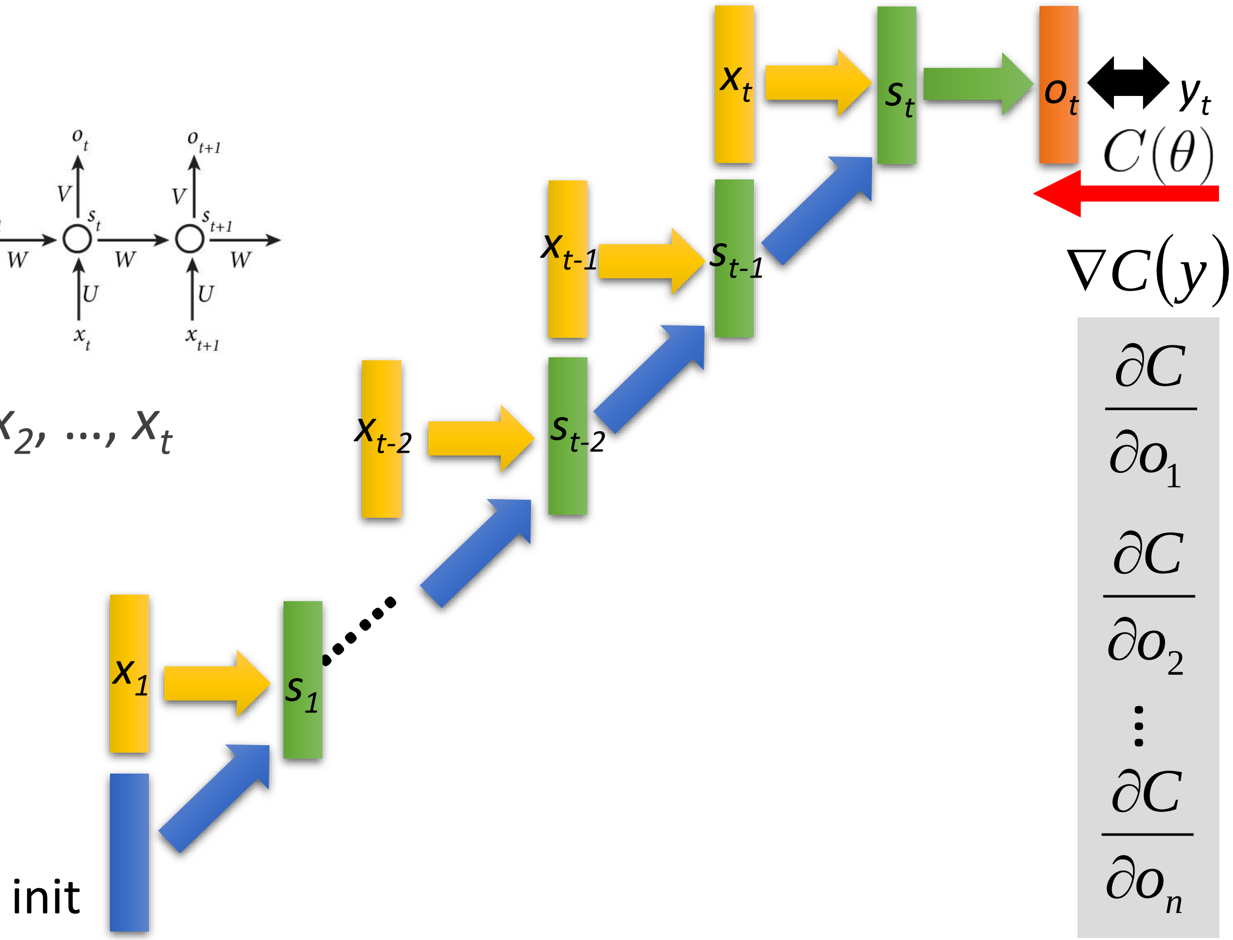
$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$
$$\delta^{L-1} = \sigma'(z^{L-1}) \odot (W^L)^T \delta^L$$
$$\vdots$$
$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$
$$\vdots$$

Backpropagation through Time (BPTT)

Unfold

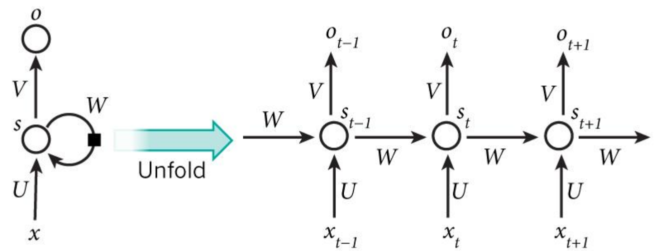


- Input: init, x_1, x_2, …, x_t
- Output: o_t
- Target: y_t

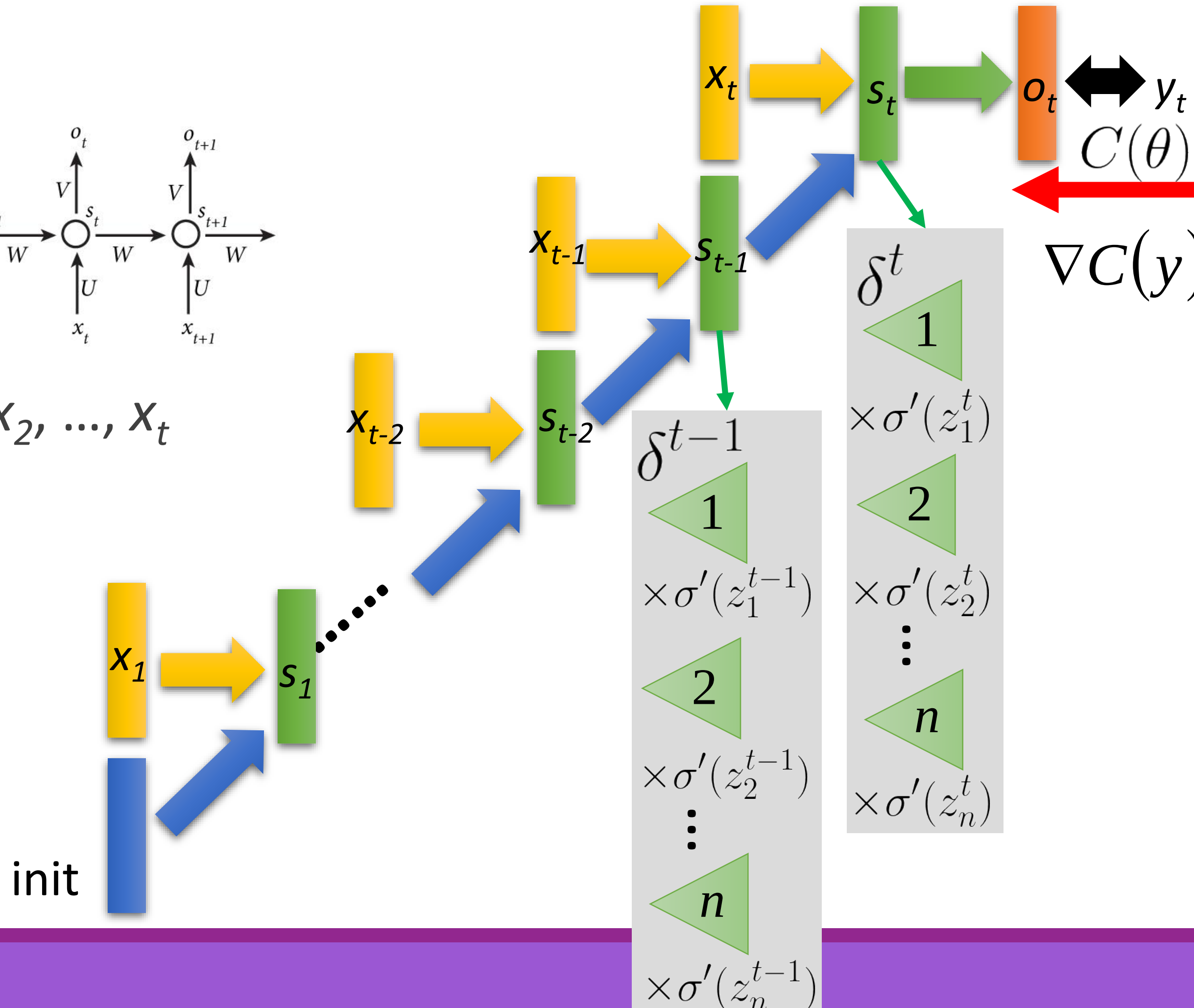


Backpropagation through Time (BPTT)

Unfold

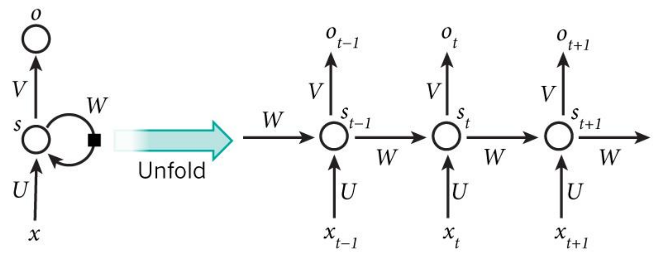


- Input: init, x_1, x_2, …, x_t
- Output: o_t
- Target: y_t

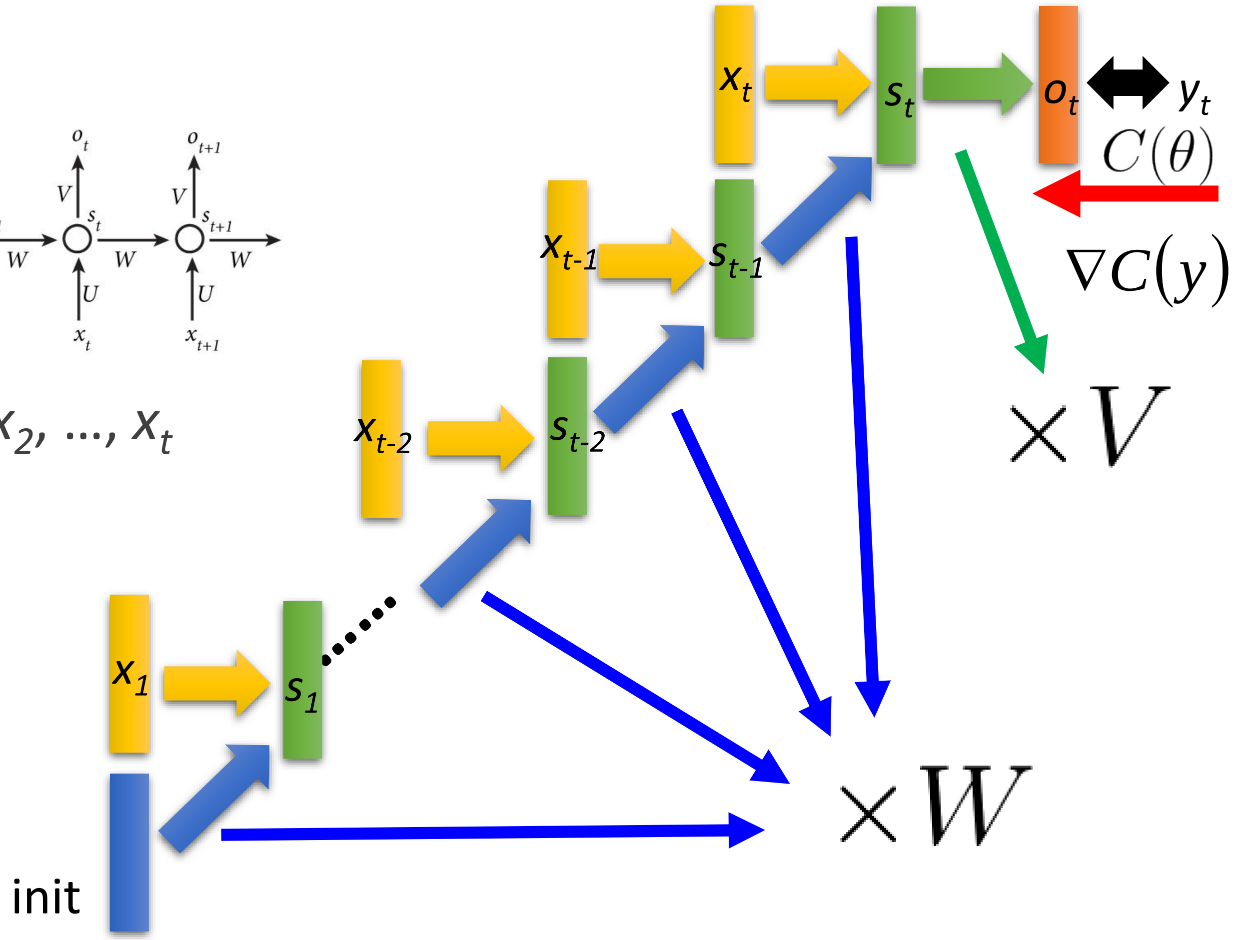


Backpropagation through Time (BPTT)

Unfold

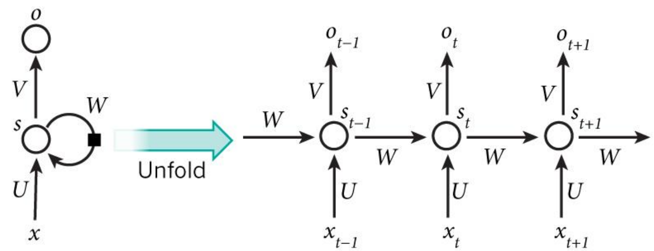


- Input: init, x_1, x_2, …, x_t
- Output: o_t
- Target: y_t

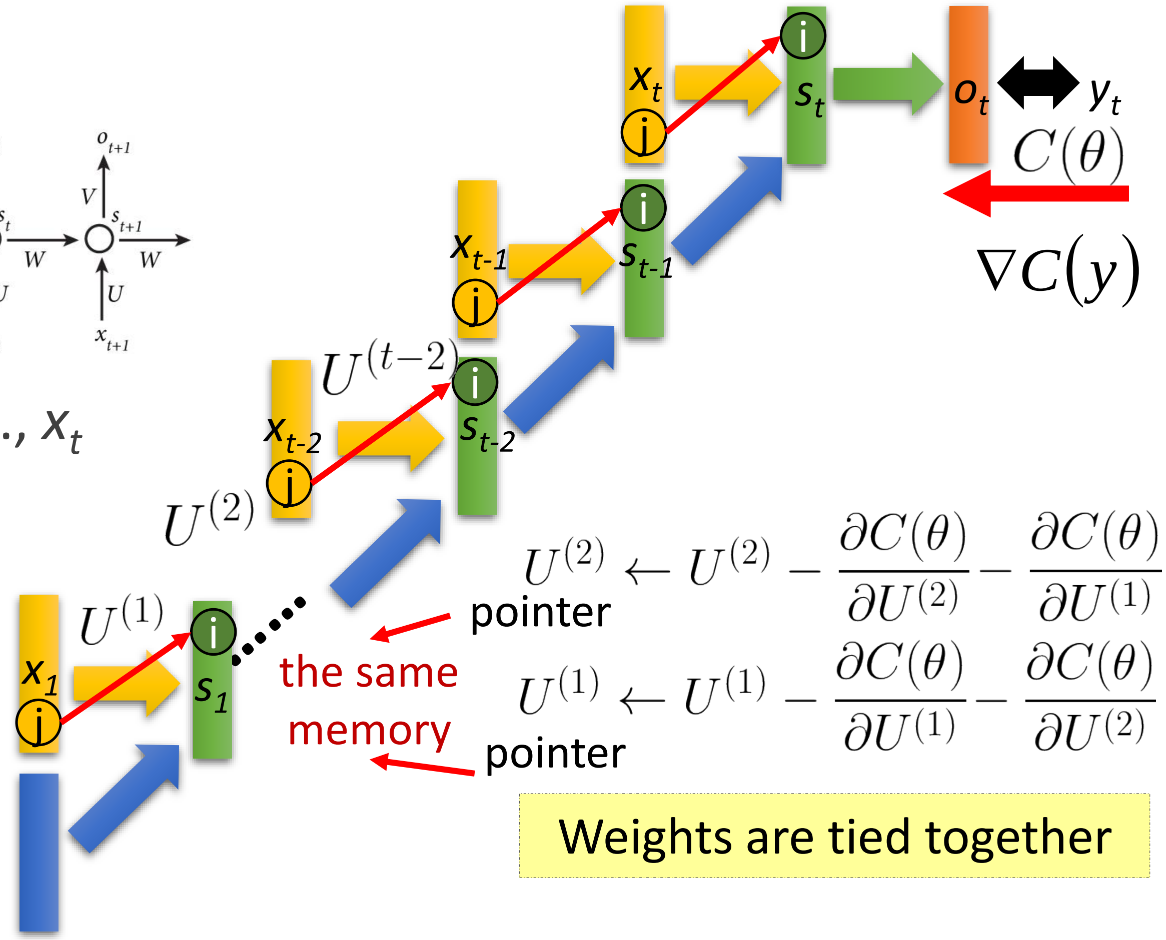


Backpropagation through Time (BPTT)

Unfold



- Input: init, x_1, x_2, …, x_t
- Output: o_t
- Target: y_t



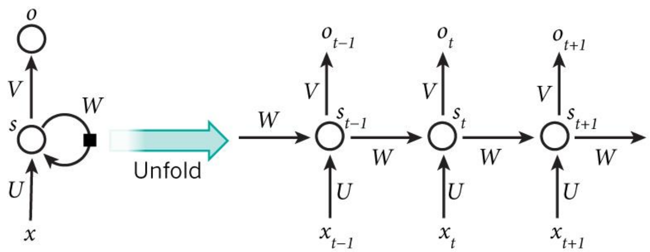
$$U^{(2)} \leftarrow U^{(2)} - \frac{\partial C(\theta)}{\partial U^{(2)}} - \frac{\partial C(\theta)}{\partial U^{(1)}}$$

$$U^{(1)} \leftarrow U^{(1)} - \frac{\partial C(\theta)}{\partial U^{(1)}} - \frac{\partial C(\theta)}{\partial U^{(2)}}$$

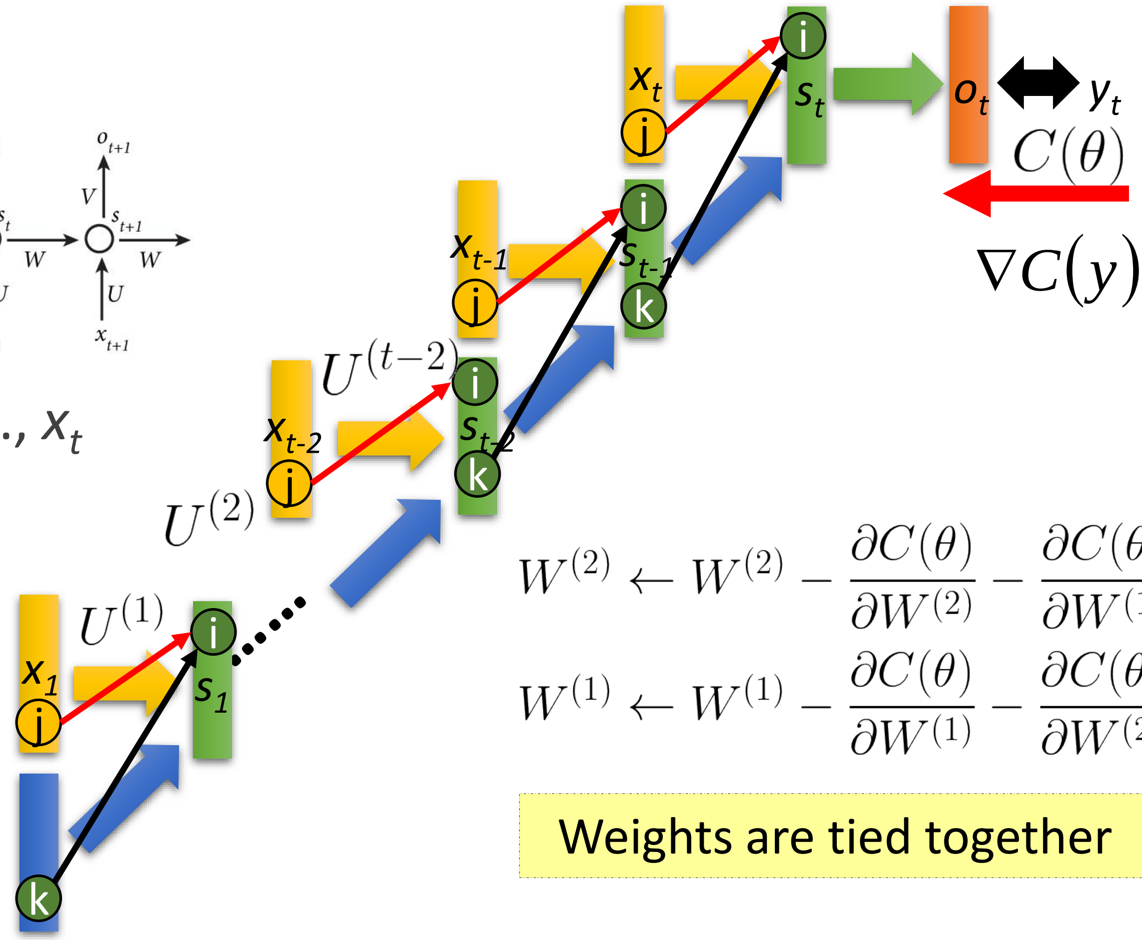
Weights are tied together

Backpropagation through Time (BPTT)

Unfold



- Input: init, $x_1, x_2, \ldots, x_t$
- Output: o_t
- Target: y_t



$$W^{(2)} \leftarrow W^{(2)} - \frac{\partial C(\theta)}{\partial W^{(2)}} - \frac{\partial C(\theta)}{\partial W^{(1)}}$$

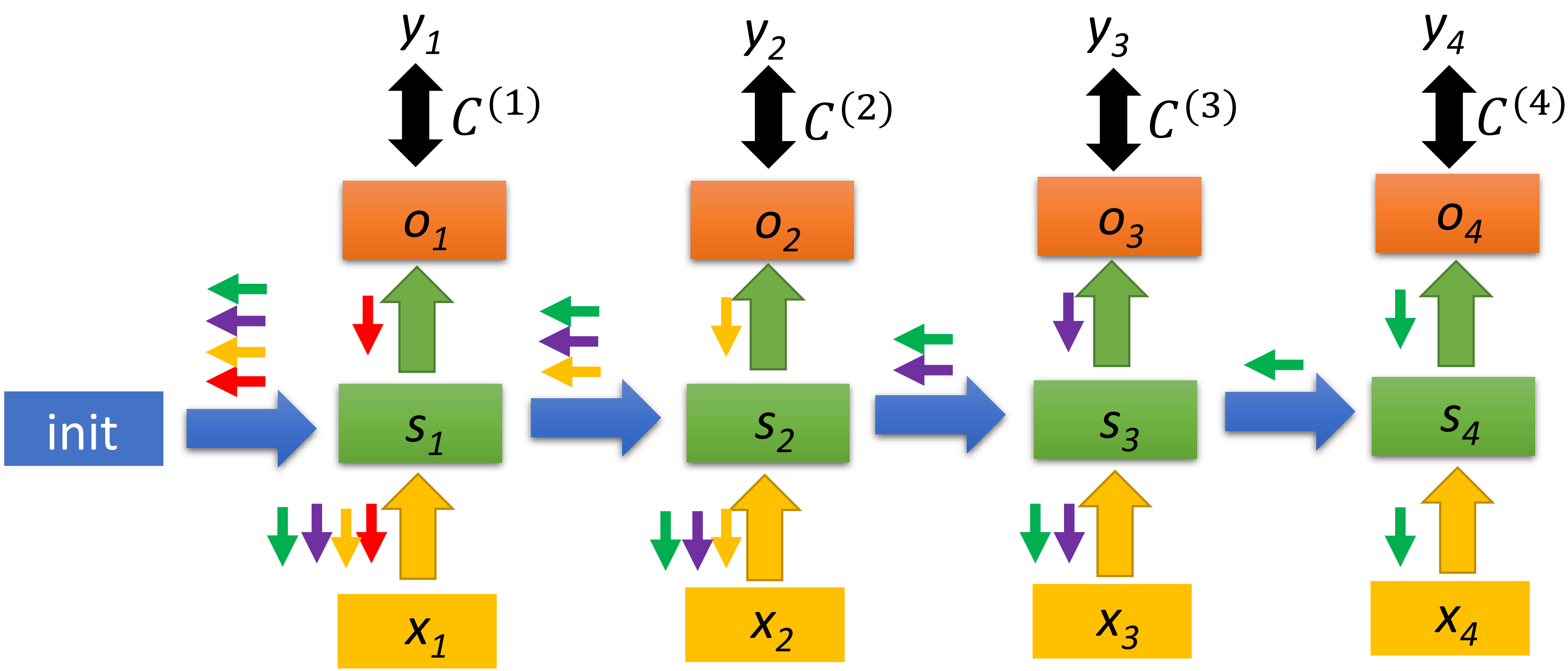
$$W^{(1)} \leftarrow W^{(1)} - \frac{\partial C(\theta)}{\partial W^{(1)}} - \frac{\partial C(\theta)}{\partial W^{(2)}}$$

Weights are tied together

BPTT

Forward Pass: Compute s_1, s_2, s_3, s_4 ……

Backward Pass: For $C^{(4)}$ For $C^{(3)}$ For $C^{(2)}$ For $C^{(1)}$



RNN Training Issue

The gradient is a product of Jacobian matrices, each associated with a step in the forward computation

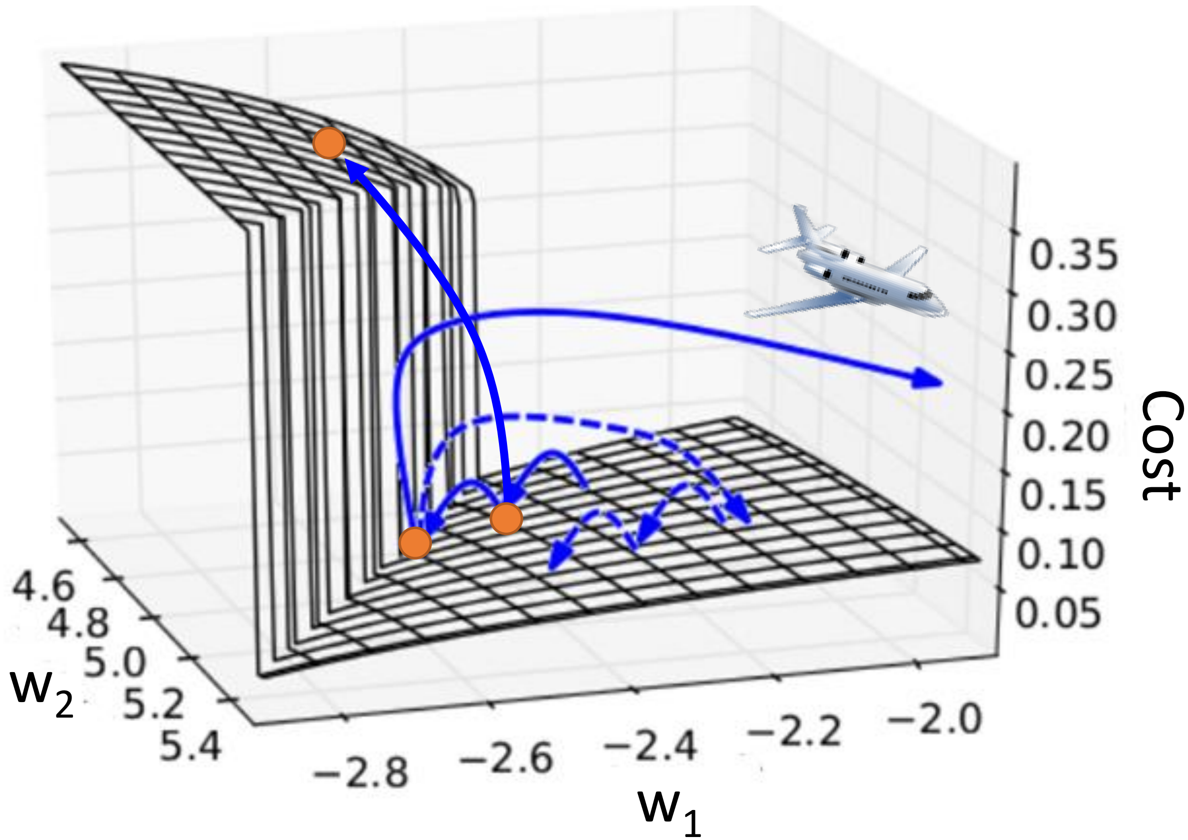
Multiply the same matrix at each time step during backprop

$$\delta^l = \boxed{\sigma'(z^l) \odot (W^{l+1})^T} \delta^{l+1}$$

The gradient becomes very small or very large quickly
→ **vanishing or exploding gradient**

Bengio et al., "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. of Neural Networks*, 1994. [link]
Pascanu et al., "On the difficulty of training recurrent neural networks," in *ICML*, 2013. [link]

Rough Error Surface

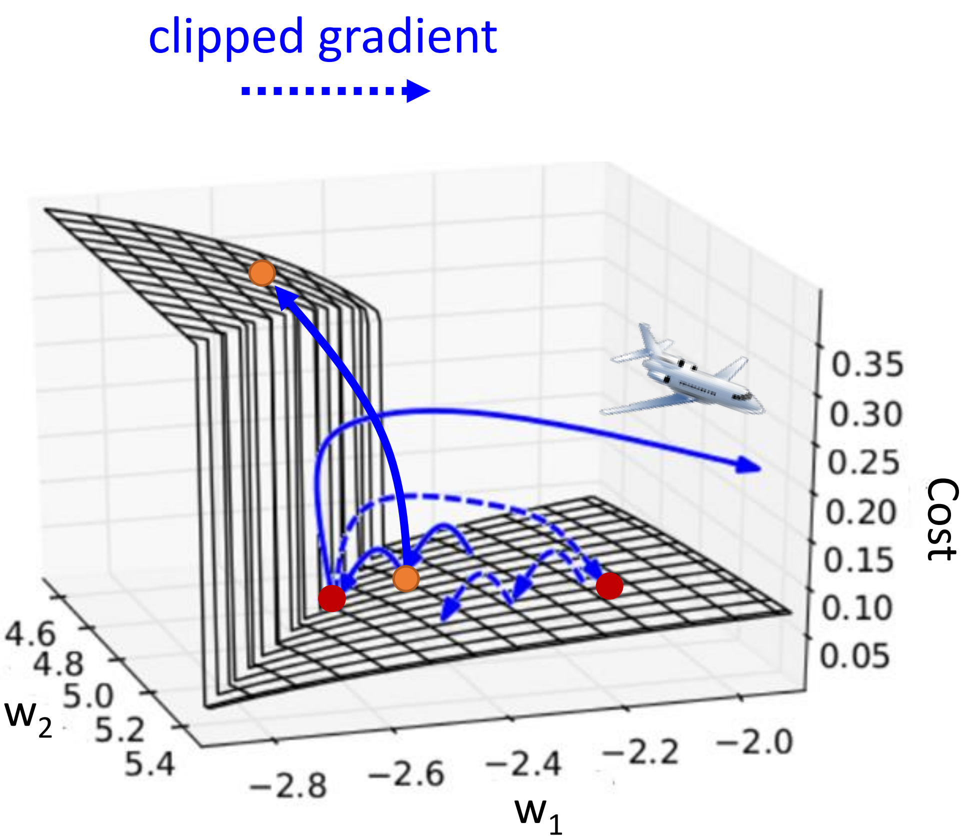


The error surface is either very flat or very steep

Possible Solutions

Recurrent Neural Network

Exploding Gradient: Clipping



Idea: control the gradient value to avoid exploding

Algorithm 1 Pseudo-code for norm clipping

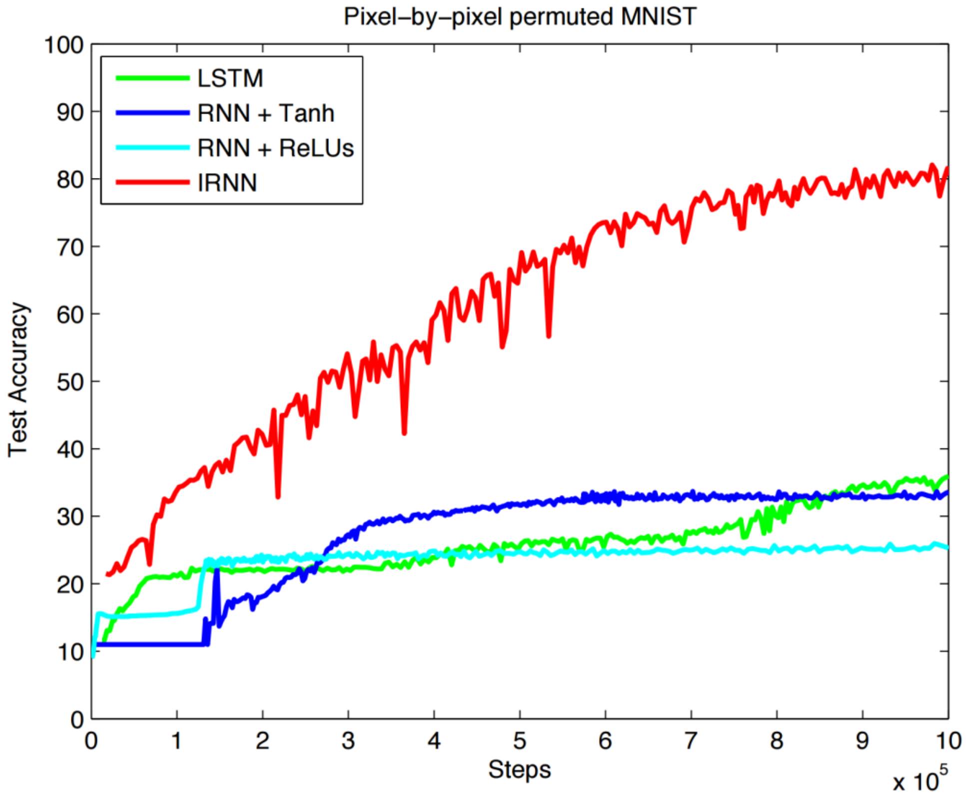
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

Parameter setting: values from half to ten times the average can still yield convergence

Vanishing Gradient: Initialization + ReLU

IRNN

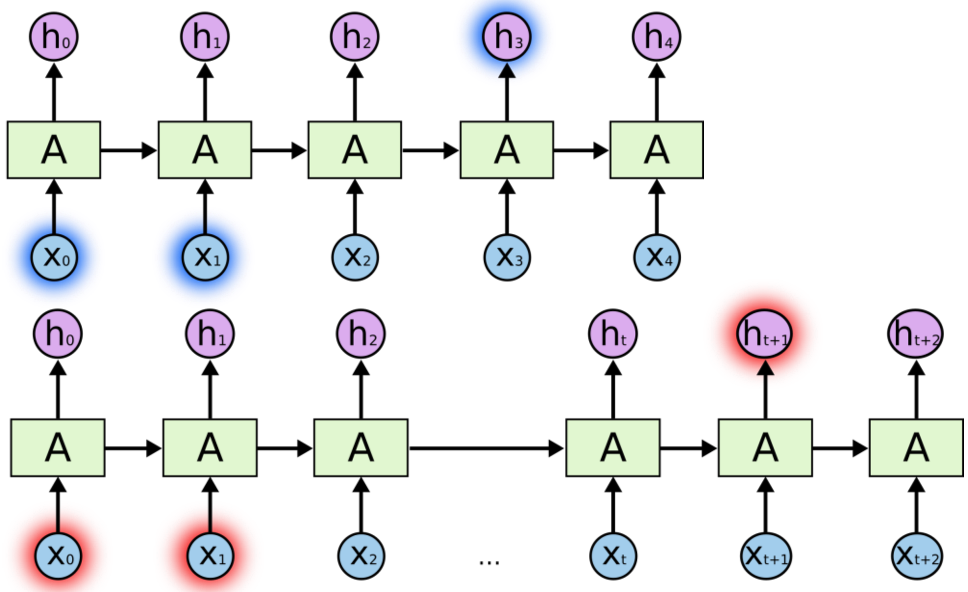
- initialize all W as identity matrix I
- use ReLU for activation functions



Vanishing Gradient: Gating Mechanism

RNN models temporal sequence information

- can handle “long-term dependencies” in theory



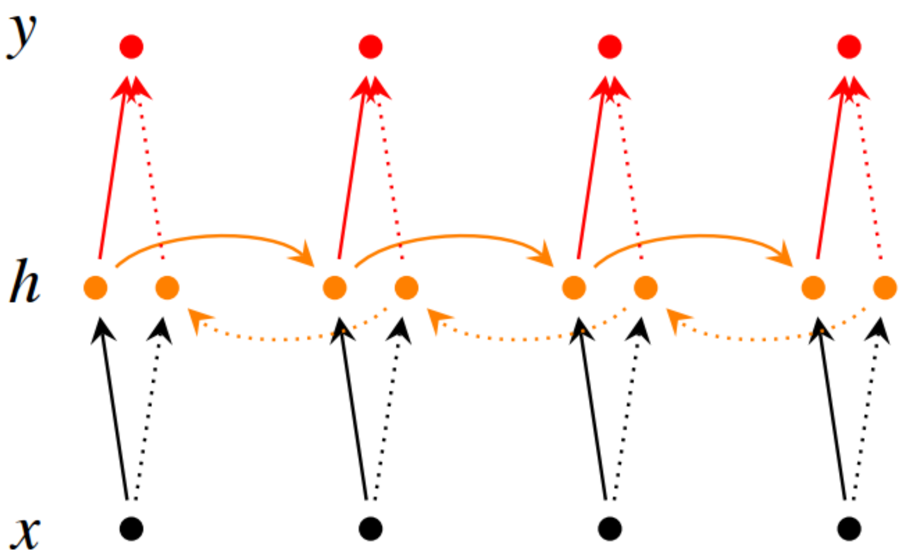
“I grew up in France…
I speak fluent ***French***.”

Issue: RNN cannot handle such “long-term dependencies” in practice due to vanishing gradient
→ apply the gating mechanism to directly encode the long-distance information

Extension

Recurrent Neural Network

Bidirectional RNN



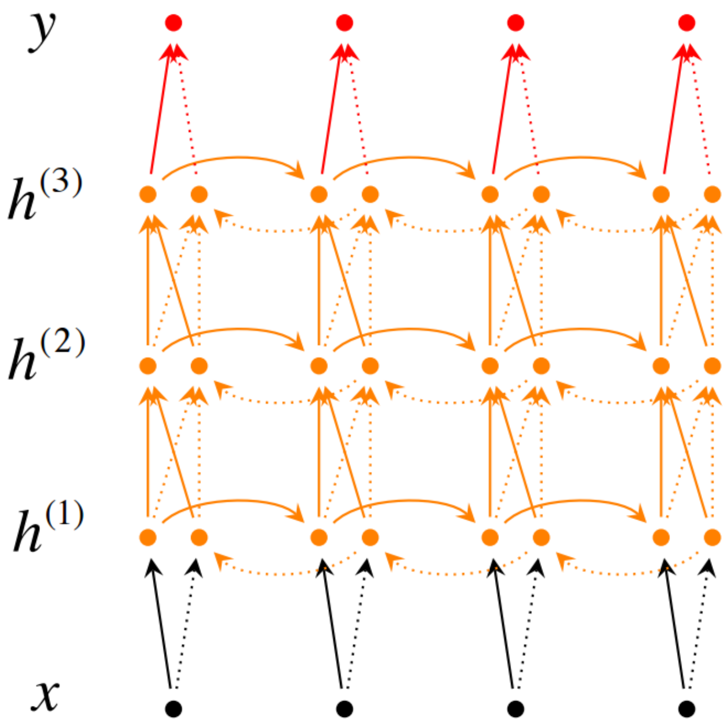
$$\overrightarrow{h}_t = f(\overrightarrow{W} x_t + \overrightarrow{V}\overrightarrow{h}_{t-1} + \overrightarrow{b})$$

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

$$y_t = g(U[\overrightarrow{h}_t;\overleftarrow{h}_t] + c)$$

$h = [\overrightarrow{h}; \overleftarrow{h}]$ represents (summarizes) the past and future around a single token

Deep Bidirectional RNN



$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

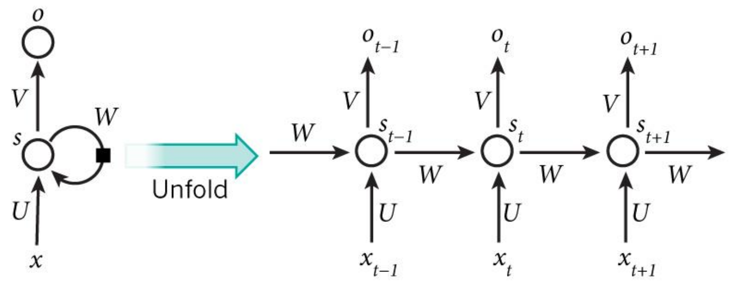
Each memory layer passes an intermediate representation to the next

Concluding Remarks

Recurrent Neural Networks

- Definition

$$s_t = \sigma(W s_{t-1} + U x_t)$$
$$o_t = \mathrm{softmax}(V s_t)$$



- Issue: Vanishing/Exploding Gradient
- Solution:
  - Exploding Gradient: Clipping
  - Vanishing Gradient: Initialization, ReLU, Gated RNNs

Extension

- Bidirectional
- Deep RNN