



Deep Learning Basics

Lecture 9: Recurrent Neural Networks

Princeton University COS 495

Instructor: Yingyu Liang

Introduction

Recurrent neural networks

- Dates back to (Rumelhart *et al.*, 1986)
- A family of neural networks for handling sequential data, which involves variable length inputs or outputs
- Especially, for natural language processing (NLP)

Sequential data

- Each data point: A sequence of vectors $x^{(t)}$, for $1 \leq t \leq \tau$
- Batch data: many sequences with different lengths τ
- Label: can be a scalar, a vector, or even a sequence
- Example
 - Sentiment analysis
 - Machine translation

Example: machine translation

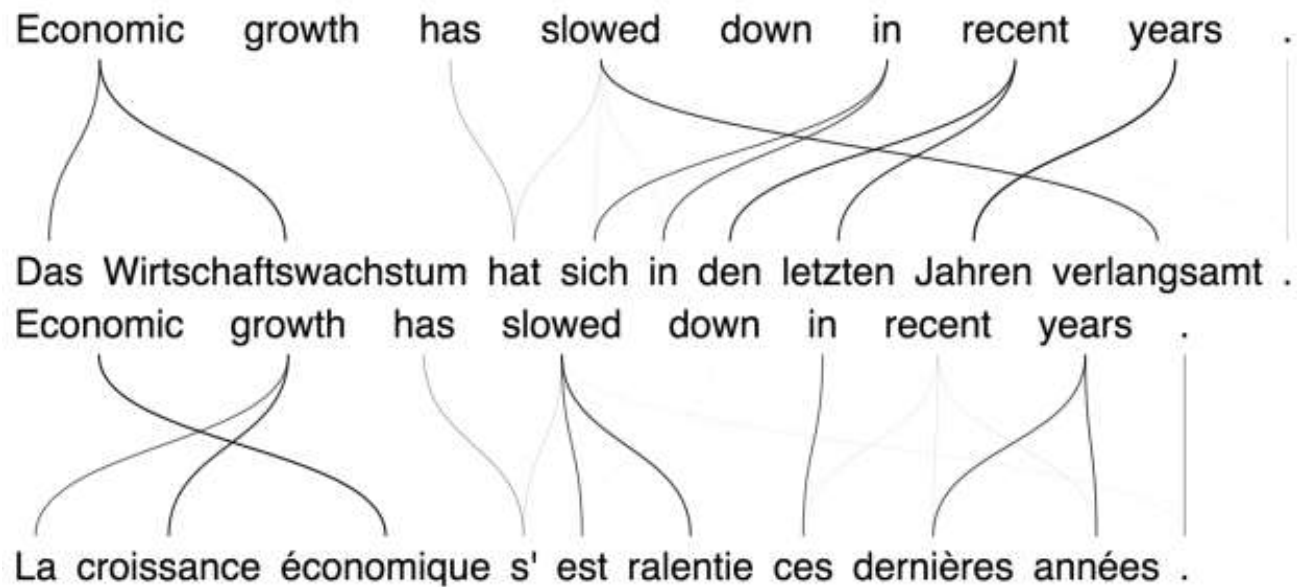


Figure from: devblogs.nvidia.com

More complicated sequential data

- Data point: two dimensional sequences like images
- Label: different type of sequences like text sentences
- Example: image captioning

Image captioning

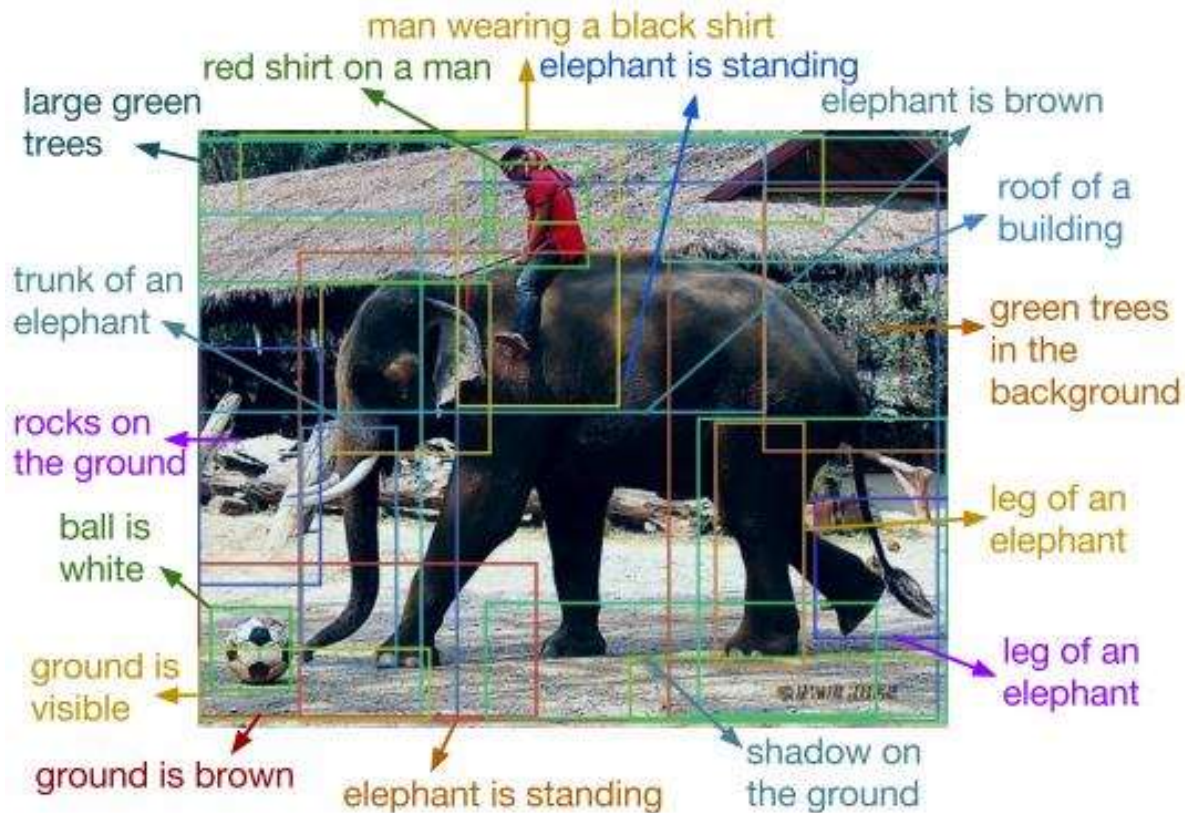
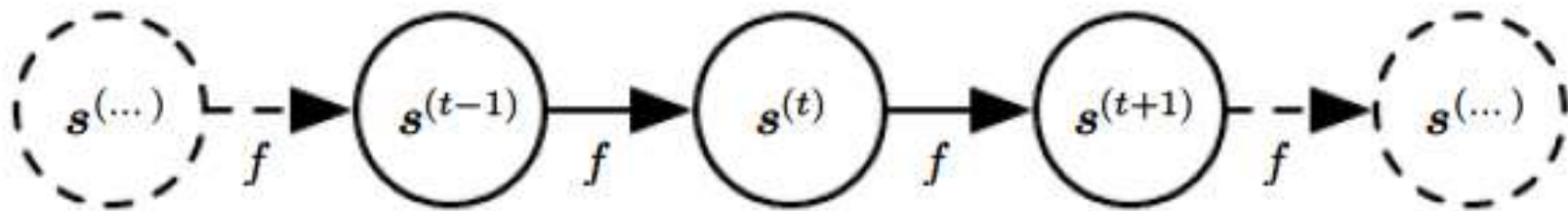


Figure from the paper “DenseCap: Fully Convolutional Localization Networks for Dense Captioning”, by Justin Johnson, Andrej Karpathy, Li Fei-Fei

Computational graphs

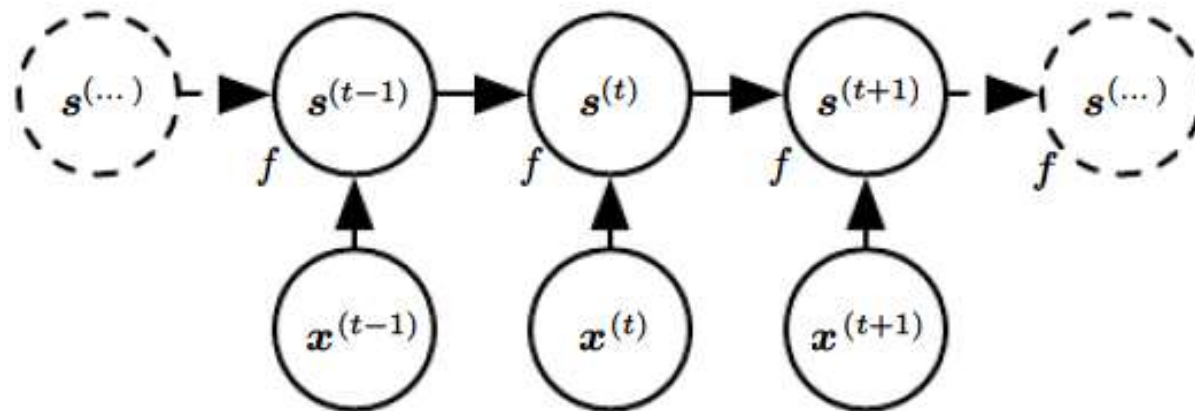
A typical dynamic system



$$s^{(t+1)} = f(s^{(t)}; \theta)$$

Figure from *Deep Learning*,
Goodfellow, Bengio and Courville

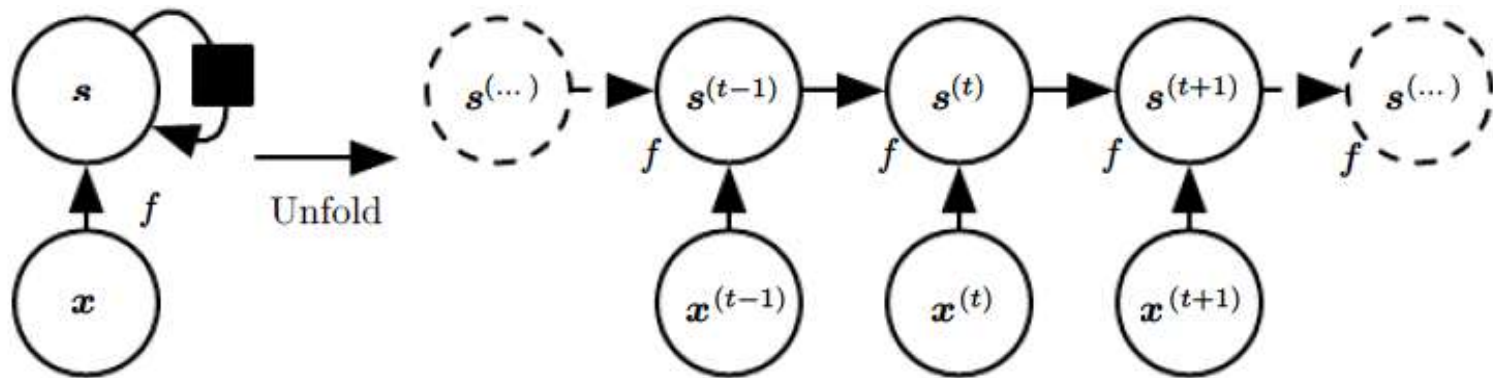
A system driven by external data



$$s^{(t+1)} = f(s^{(t)}, x^{(t+1)}; \theta)$$

Figure from *Deep Learning*,
Goodfellow, Bengio and Courville

Compact view

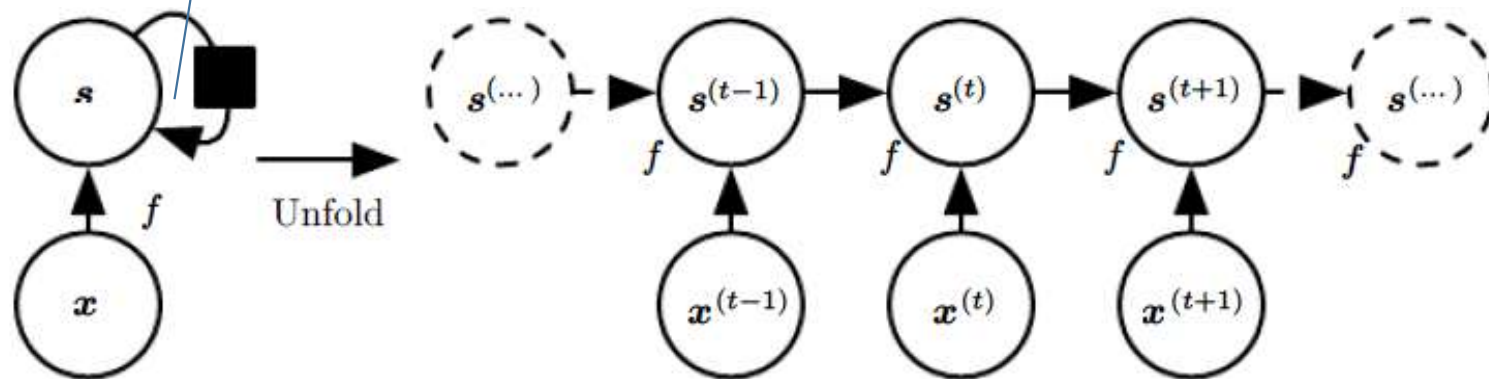


$$s^{(t+1)} = f(s^{(t)}, x^{(t+1)}; \theta)$$

Figure from *Deep Learning*,
Goodfellow, Bengio and Courville

Compact view

square: one step time delay



$$s^{(t+1)} = f(s^{(t)}, x^{(t+1)}; \theta)$$

Key: the same f and θ
for all time steps

Figure from *Deep Learning*,
Goodfellow, Bengio and Courville

Recurrent neural networks (RNN)

Recurrent neural networks

- Use **the same** computational function and parameters across different time steps of the sequence
- Each time step: takes the input entry **and the previous hidden state** to compute the output entry
- Loss: typically computed every time step

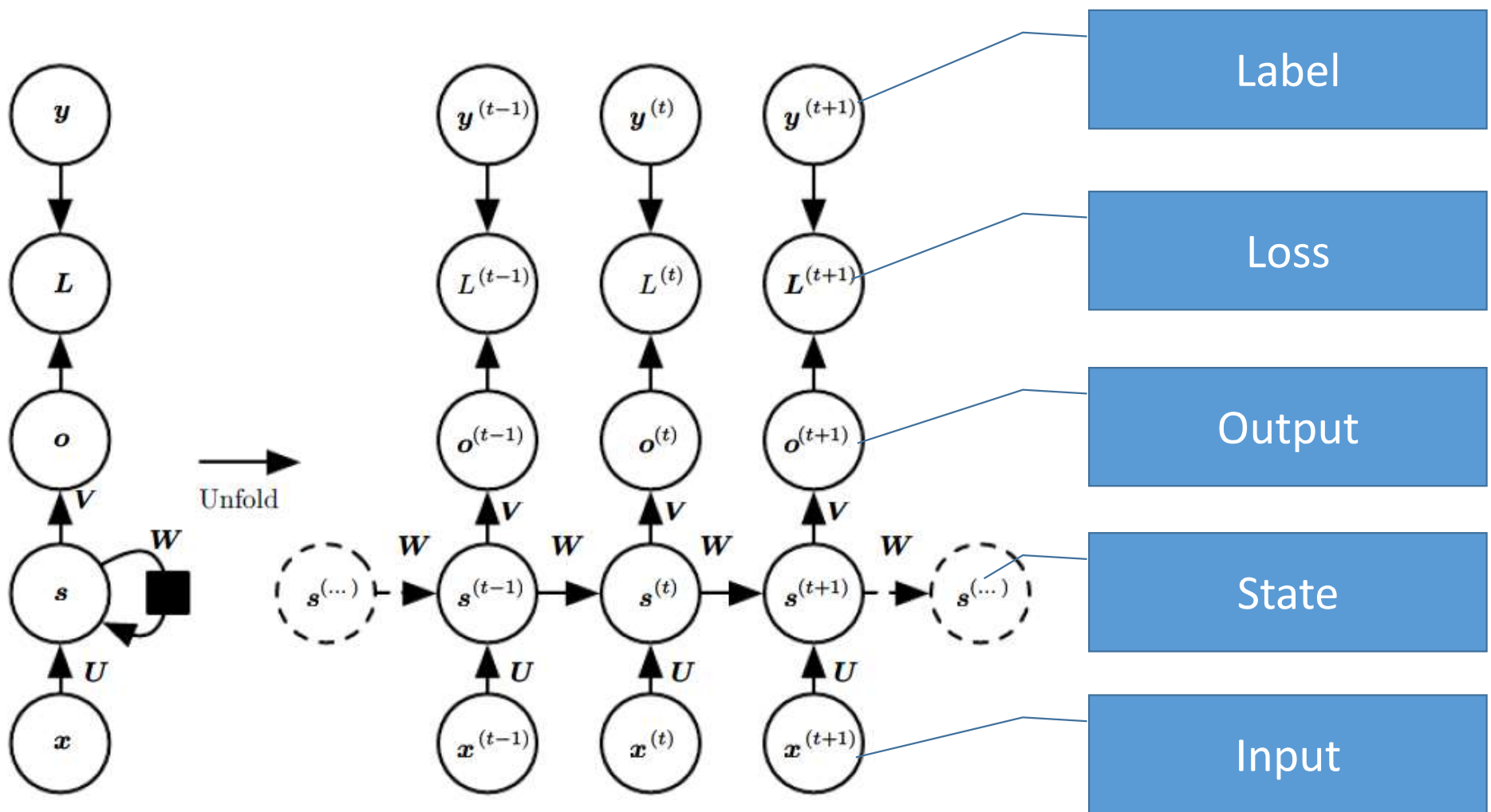
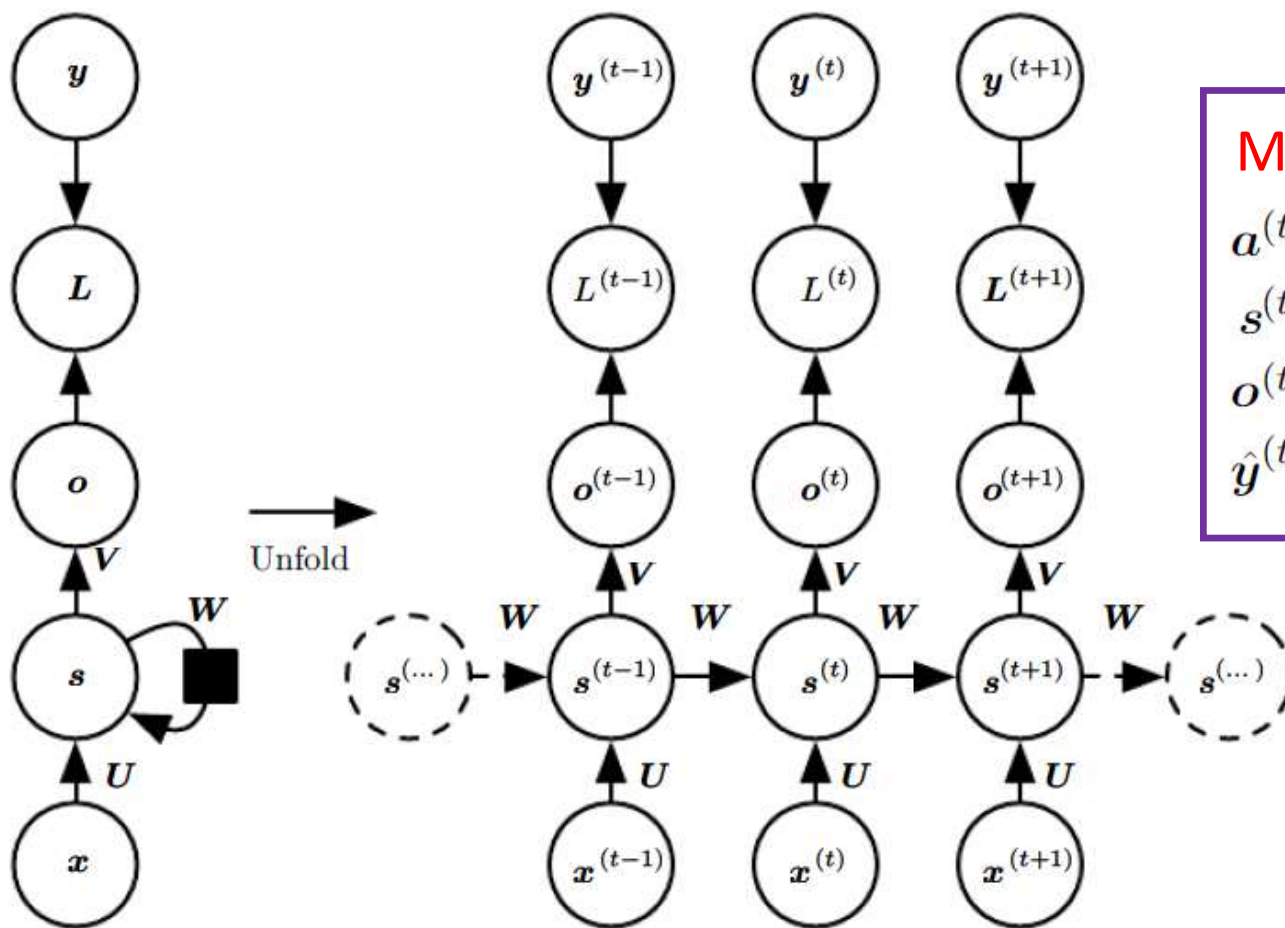


Figure from *Deep Learning*, by Goodfellow, Bengio and Courville



Math formula:

$$a^{(t)} = b + Ws^{(t-1)} + Ux^{(t)}$$

$$s^{(t)} = \tanh(a^{(t)})$$

$$o^{(t)} = c + Vs^{(t)}$$

$$\hat{y}^{(t)} = \text{softmax}(o^{(t)})$$

Figure from *Deep Learning*,
Goodfellow, Bengio and Courville

Advantage

- Hidden state: a lossy summary of the past
- Shared functions and parameters: greatly reduce the **capacity** and good for **generalization** in learning
- Explicitly use the prior knowledge that the sequential data can be processed by in the same way at different time step (e.g., NLP)

Advantage

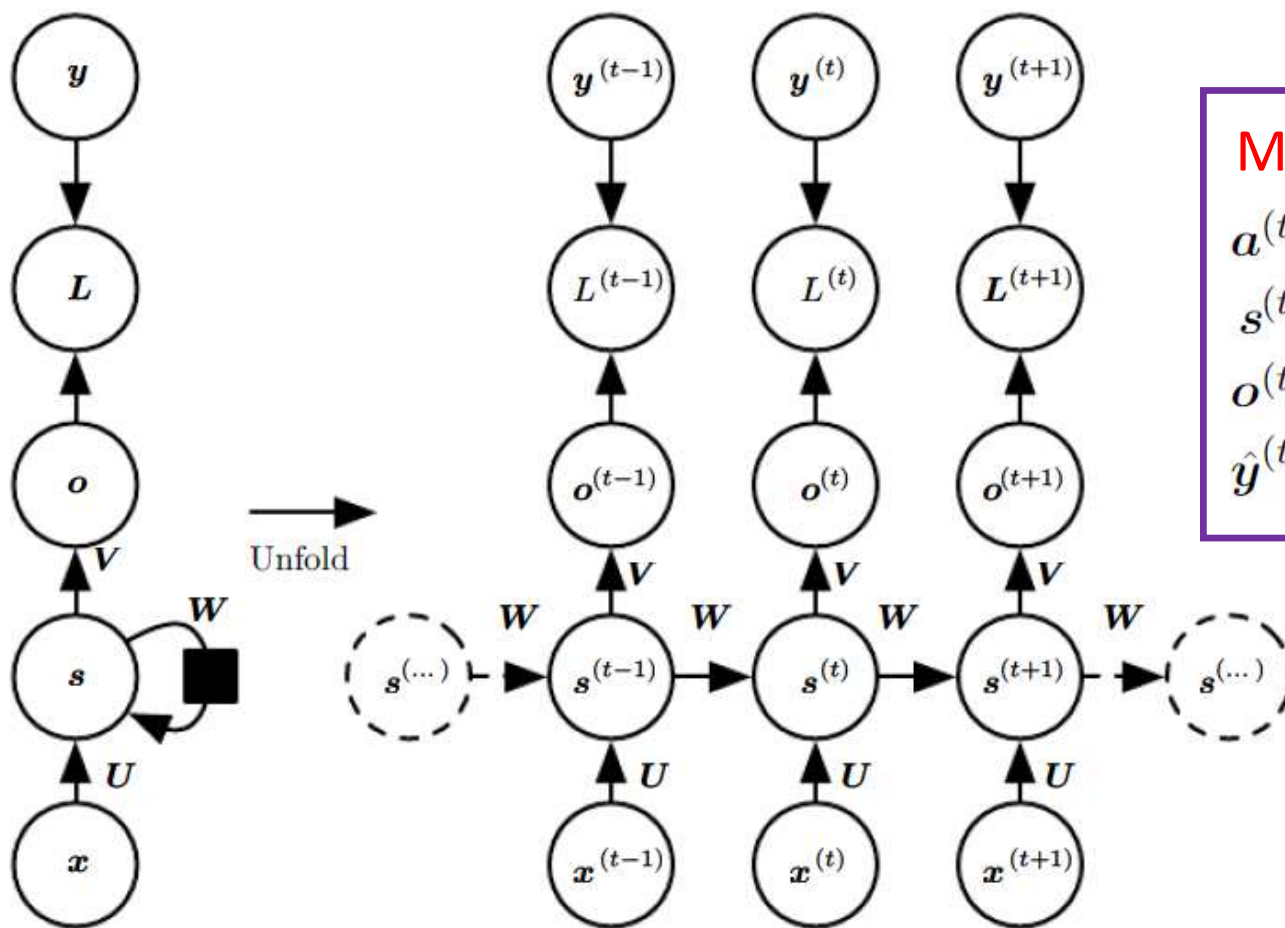
- Hidden state: a lossy summary of the past
- Shared functions and parameters: greatly reduce the capacity and good for **generalization** in learning
- Explicitly use the **prior knowledge** that the sequential data can be processed by in the same way at different time step (e.g., NLP)
- Yet still powerful (actually **universal**): any function computable by a Turing machine can be computed by such a recurrent network of a finite size (see, e.g., Siegelmann and Sontag (1995))

Training RNN

- Principle: unfold the computational graph, and use **backpropagation**
- Called back-propagation through time (BPTT) algorithm
- Can then apply any general-purpose gradient-based techniques

Training RNN

- Principle: unfold the computational graph, and use backpropagation
- Called back-propagation through time (BPTT) algorithm
- Can then apply any general-purpose gradient-based techniques
- Conceptually: first compute the gradients of **the internal nodes**, then compute the gradients of **the parameters**



Math formula:

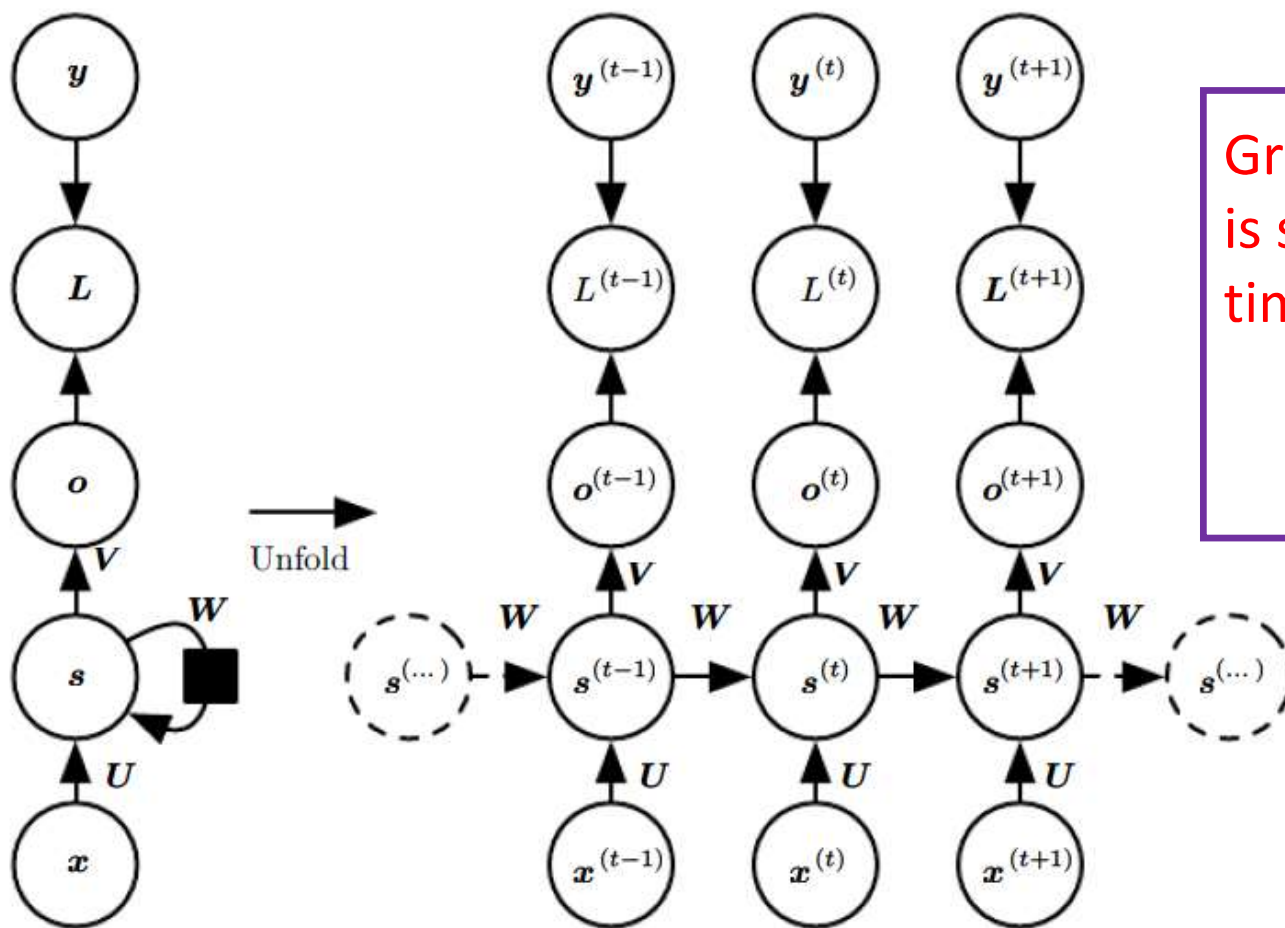
$$a^{(t)} = b + Ws^{(t-1)} + Ux^{(t)}$$

$$s^{(t)} = \tanh(a^{(t)})$$

$$o^{(t)} = c + Vs^{(t)}$$

$$\hat{y}^{(t)} = \text{softmax}(o^{(t)})$$

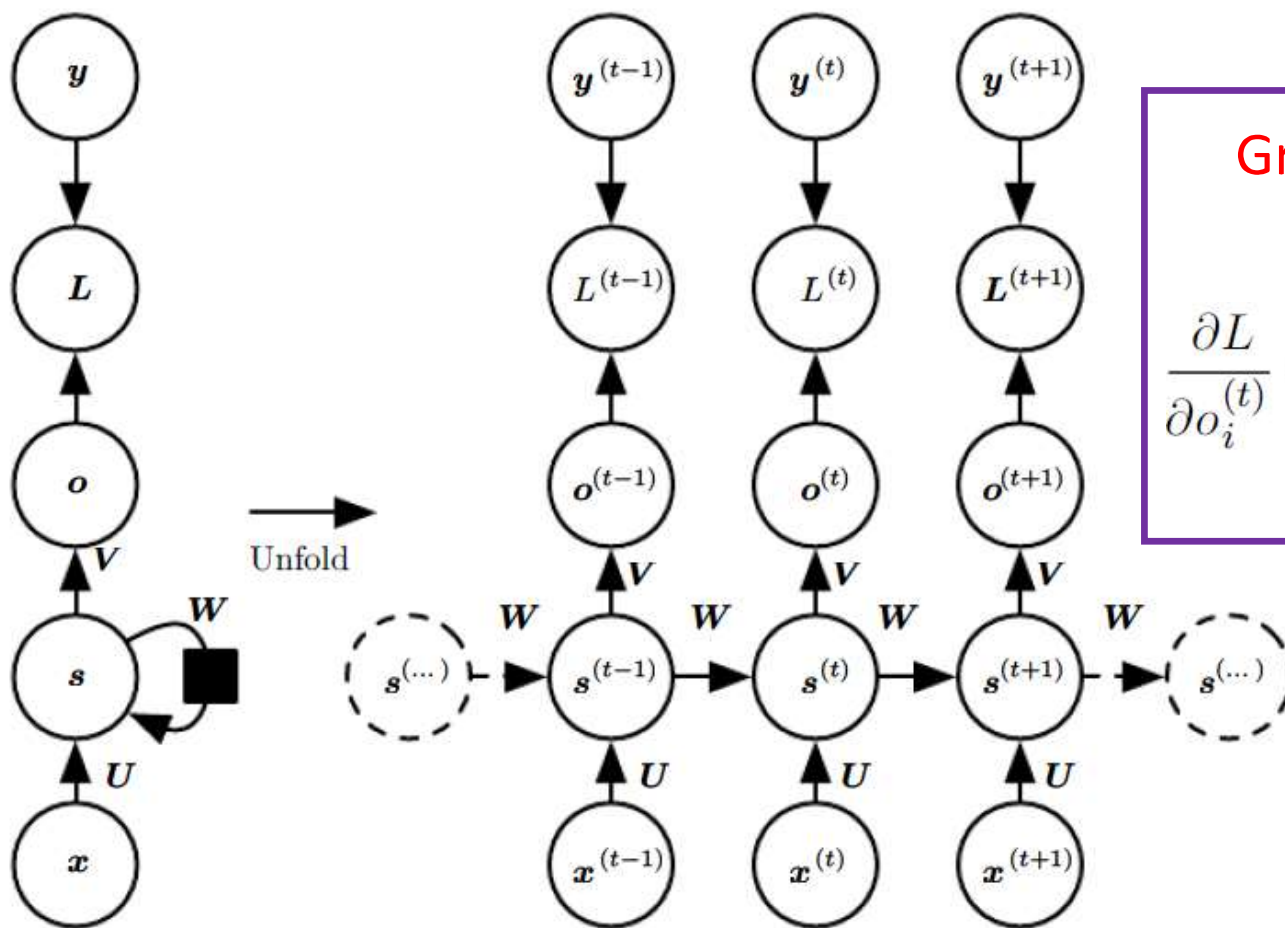
Figure from *Deep Learning*,
Goodfellow, Bengio and Courville



Gradient at $L^{(t)}$: (total loss is sum of those at different time steps)

$$\frac{\partial L}{\partial L^{(t)}} = 1.$$

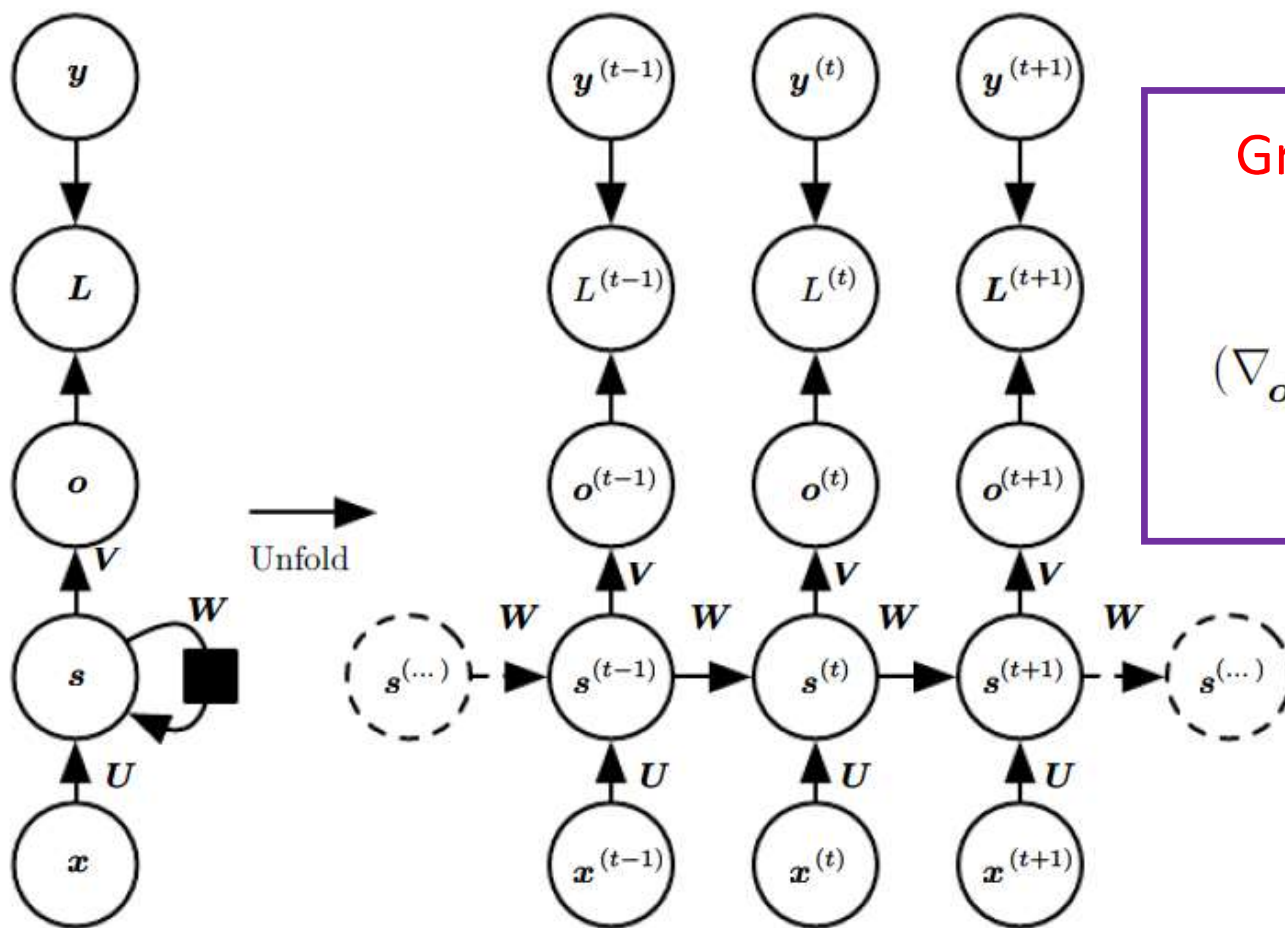
Figure from *Deep Learning*,
Goodfellow, Bengio and Courville



Gradient at $o^{(t)}$:

$$\frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i, y^{(t)}}$$

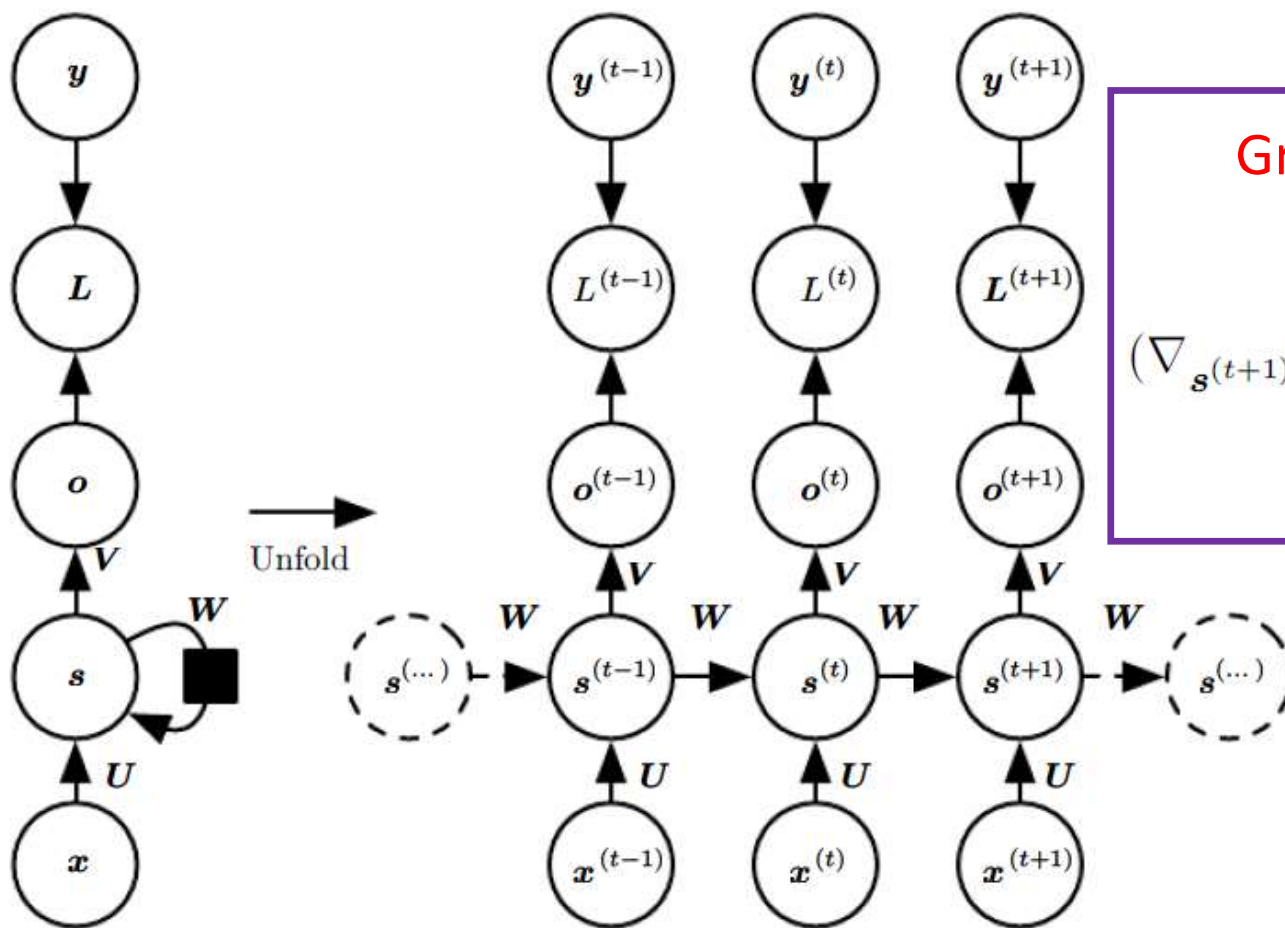
Figure from *Deep Learning*,
Goodfellow, Bengio and Courville



Gradient at $s^{(\tau)}$:

$$(\nabla_{o^{(\tau)}} L) \frac{\partial o^{(\tau)}}{\partial s^{(\tau)}} = (\nabla_{o^{(\tau)}} L) V$$

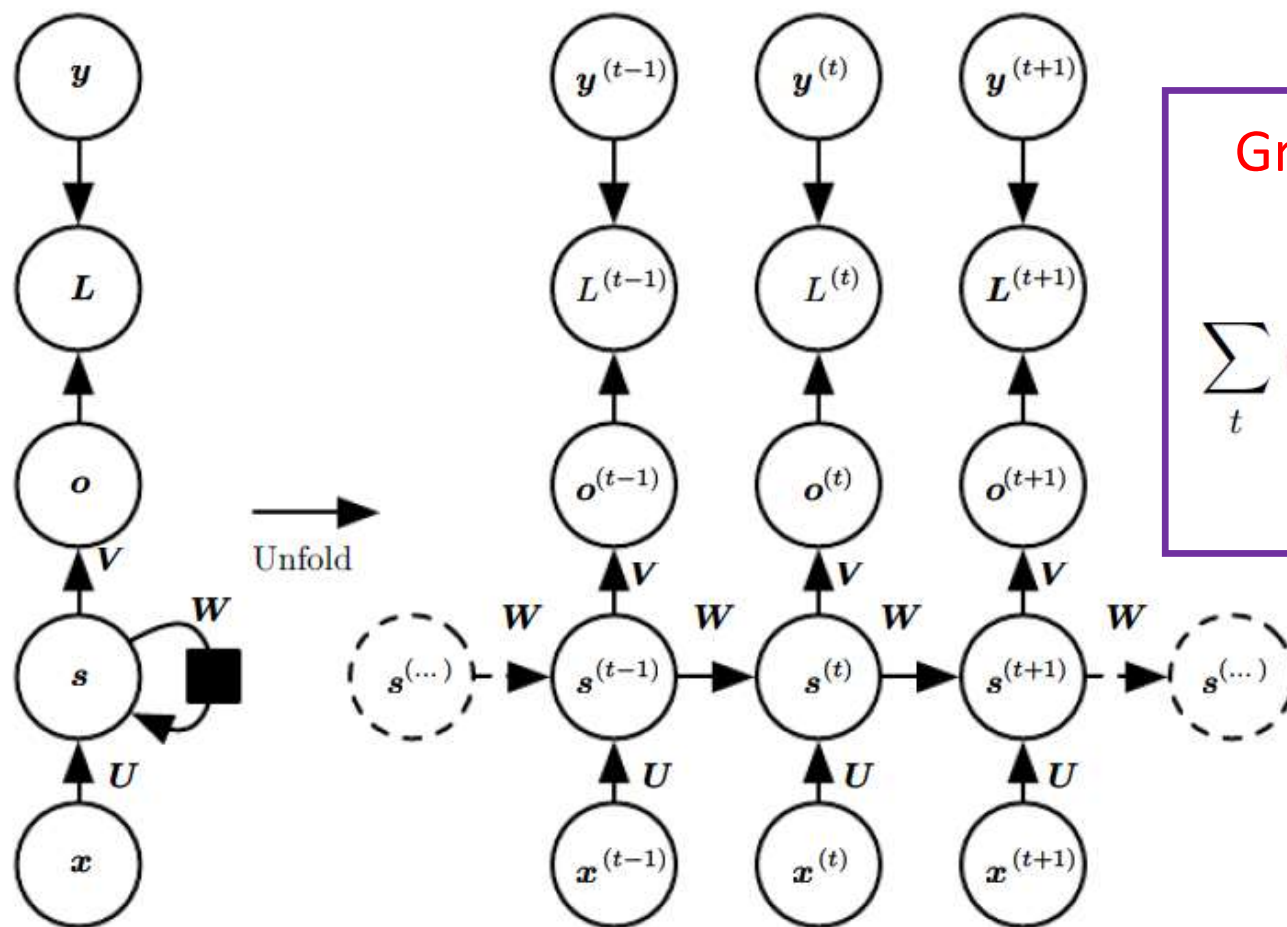
Figure from *Deep Learning*,
Goodfellow, Bengio and Courville



Gradient at $s^{(t)}$:

$$(\nabla_{s^{(t+1)}} L) \frac{\partial s^{(t+1)}}{\partial s^{(t)}} + (\nabla_{o^{(t)}} L) \frac{\partial o^{(t)}}{\partial s^{(t)}}$$

Figure from *Deep Learning*,
Goodfellow, Bengio and Courville



Gradient at parameter V :

$$\sum_t (\nabla_{o^{(t)}} L) \frac{\partial o^{(t)}}{\partial V} = \sum_t (\nabla_{o^{(t)}} L) s^{(t)\top}$$

Figure from *Deep Learning*,
Goodfellow, Bengio and Courville

Variants of RNN

RNN

- Use **the same** computational function and parameters across different time steps of the sequence
- Each time step: takes the input entry **and the previous hidden state** to compute the output entry
- Loss: typically computed every time step
- Many variants
 - Information about the past can be in many other forms
 - Only output at the end of the sequence

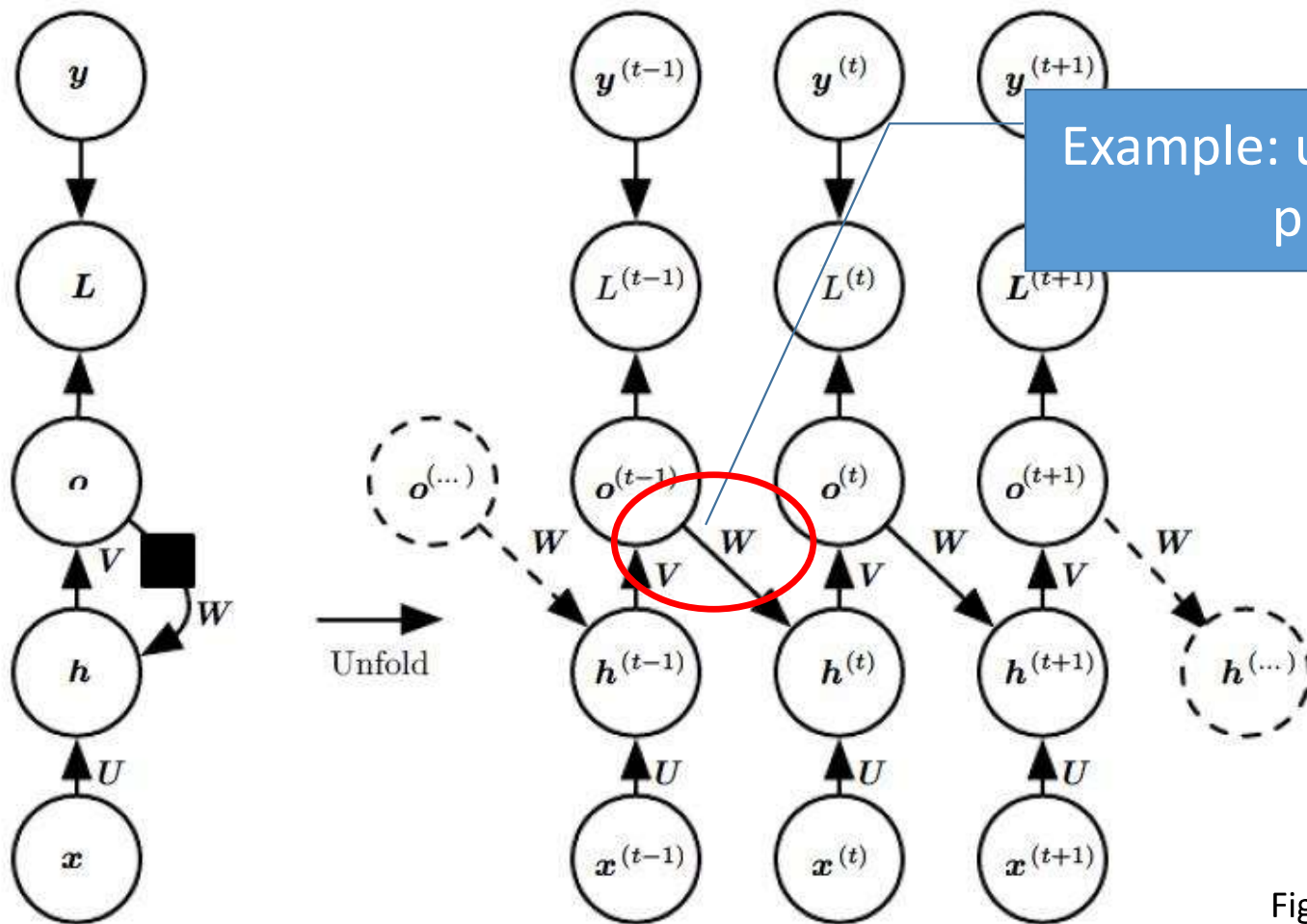


Figure from *Deep Learning*,
Goodfellow, Bengio and Courville

Example: only output at the end

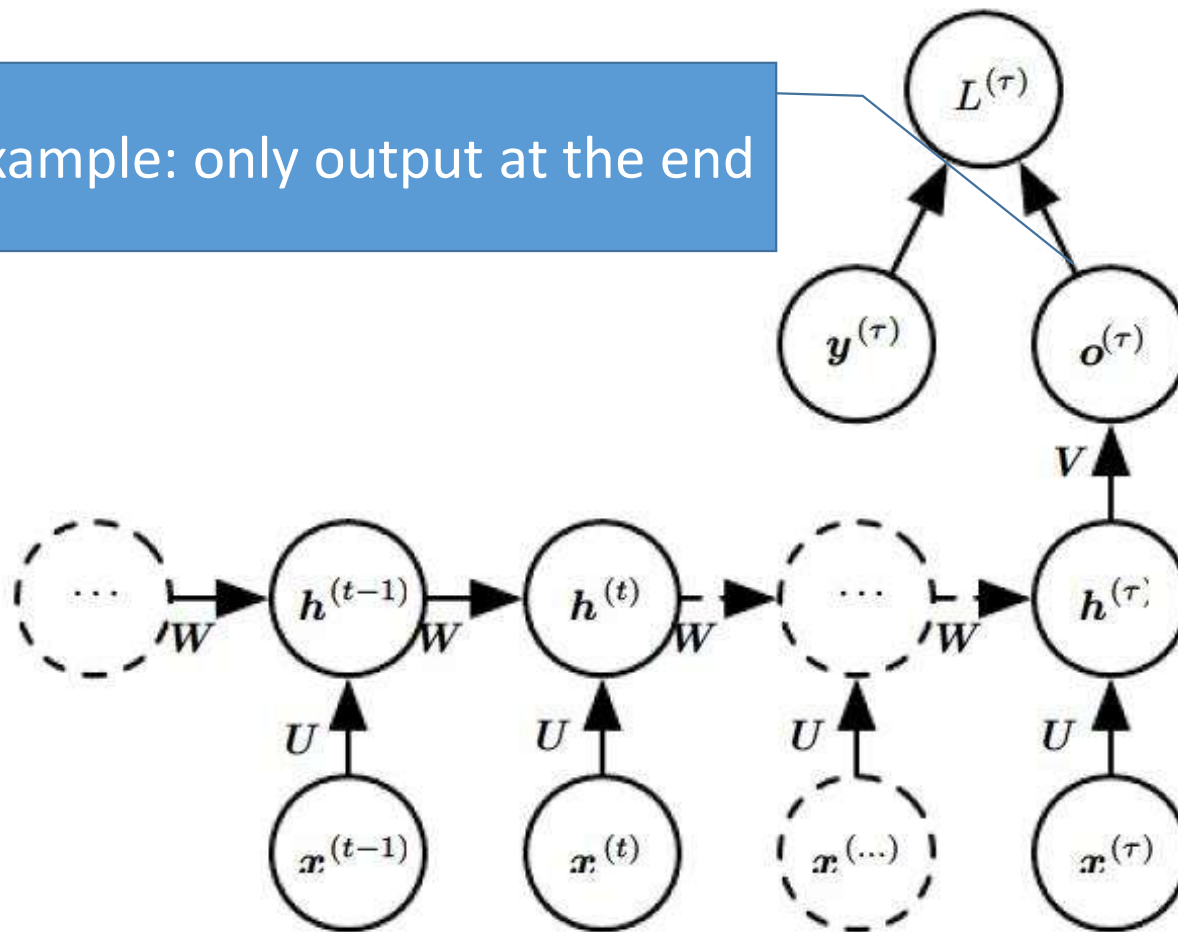


Figure from *Deep Learning*,
Goodfellow, Bengio and Courville

Bidirectional RNNs

- Many applications: output at time t may depend on the whole input sequence
- Example in speech recognition: correct interpretation of the current sound may depend on the next few phonemes, potentially even the next few words
- Bidirectional RNNs are introduced to address this

BiRNNs

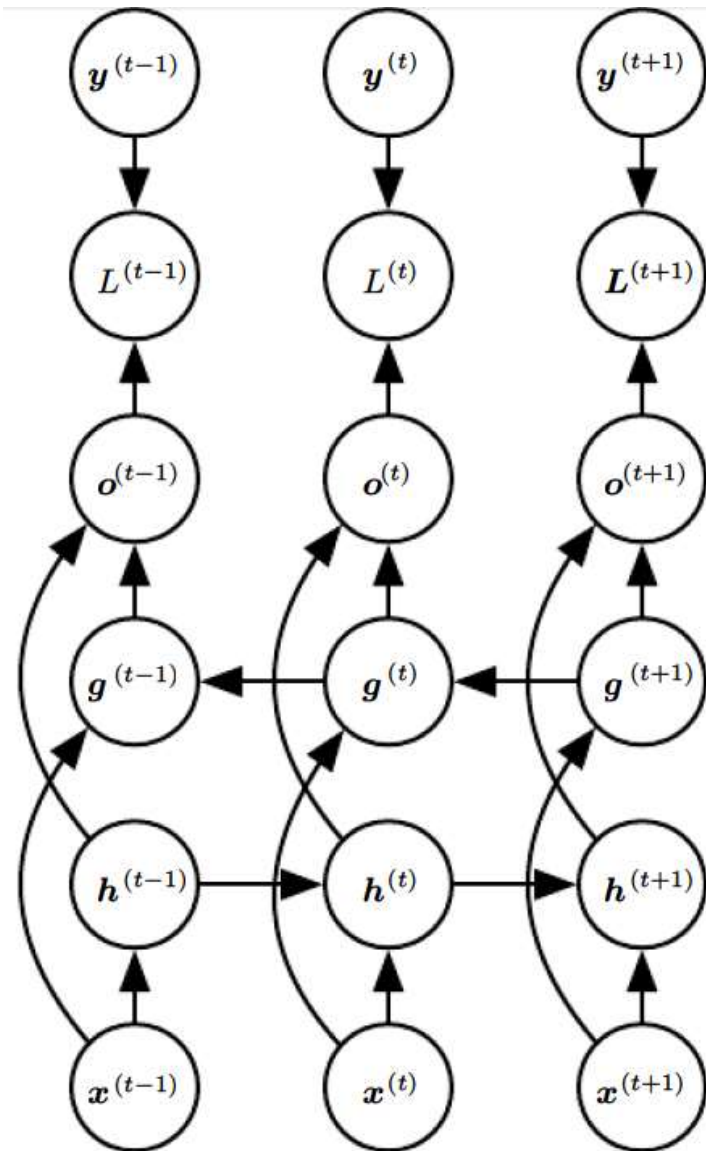


Figure from *Deep Learning*,
Goodfellow, Bengio and Courville

Encoder-decoder RNNs

- RNNs: can map sequence to one vector; or to sequence of same length
- What about mapping sequence to sequence of different length?
- Example: speech recognition, machine translation, question answering, etc

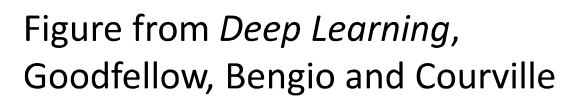


Figure from *Deep Learning*,
Goodfellow, Bengio and Courville