Neural Network Part 4: Recurrent Neural Networks

Yingyu Liang
Computer Sciences 760
Fall 2017

http://pages.cs.wisc.edu/~yliang/cs760/

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Matt Gormley, Elad Hazan, Tom Dietterich, Pedro Domingos, and Geoffrey Hinton.

Goals for the lecture

you should understand the following concepts

- sequential data
- computational graph
- recurrent neural networks (RNN) and the advantage
- training recurrent neural networks
- bidirectional RNNs
- encoder-decoder RNNs

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

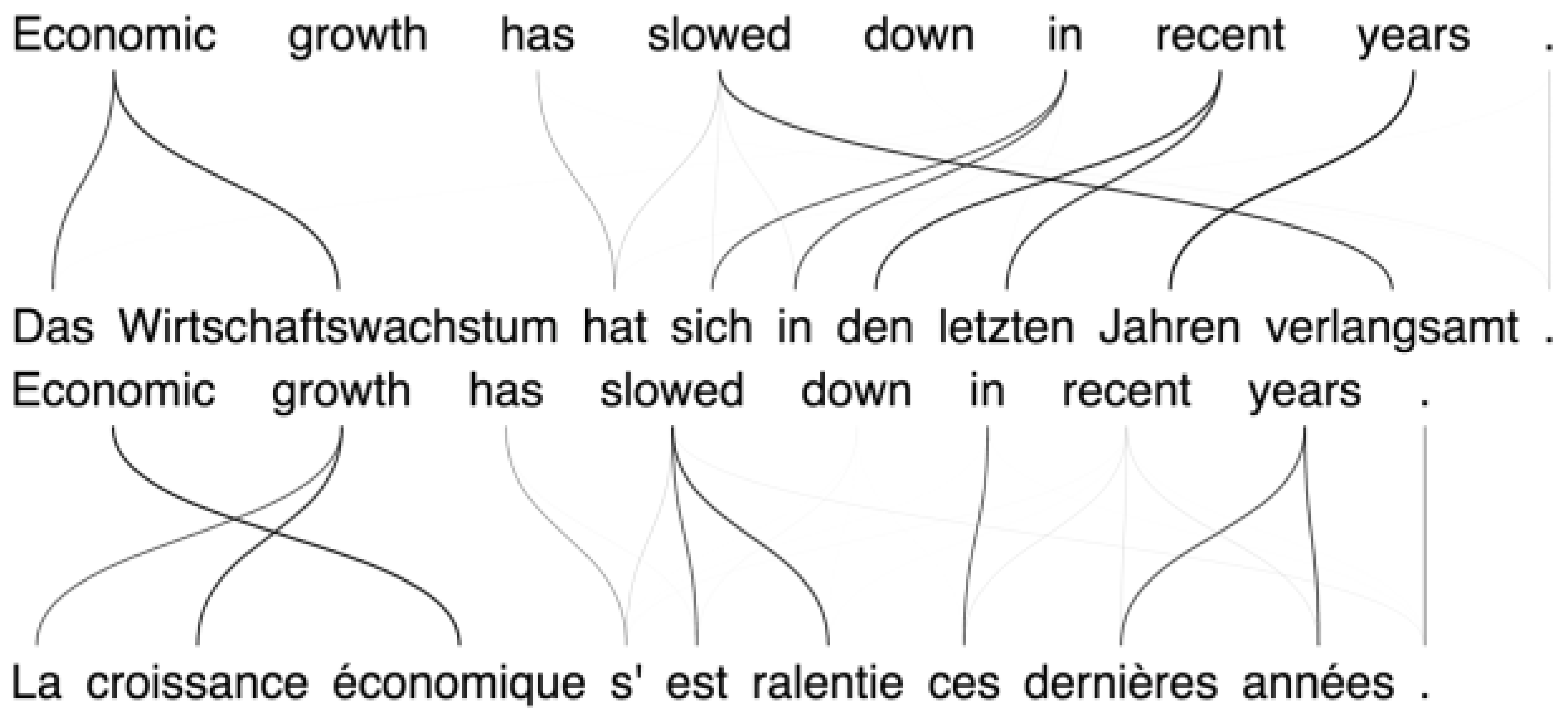
- Especially, for natural language processing (NLP)

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



Economic growth has slowed down in recent years .
Das Wirtschaftswachstum hat sich in den letzten Jahren verlangsamt .
Economic growth has slowed down in recent years .
La croissance économique s' est ralentie ces dernières années .

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

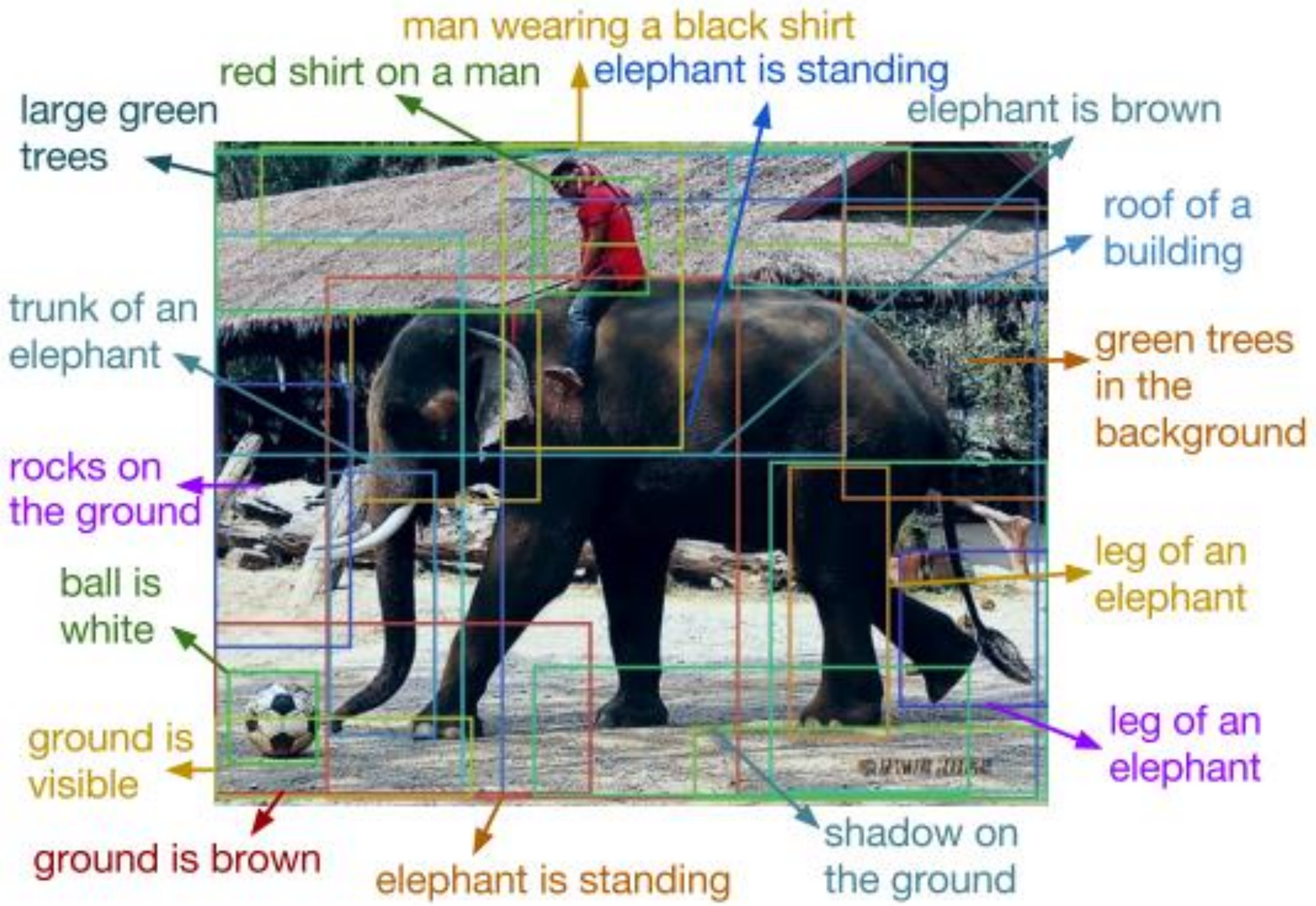
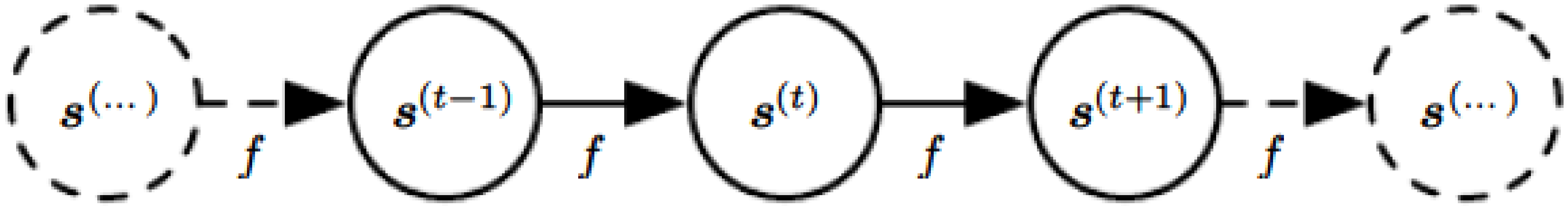


Figure from the paper "DenseCap: Fully Convolutional Localization Networks for Dense Captioning", by Justin Johnson, Andrej Karpathy, Li Fei-Fei

Computational graphs

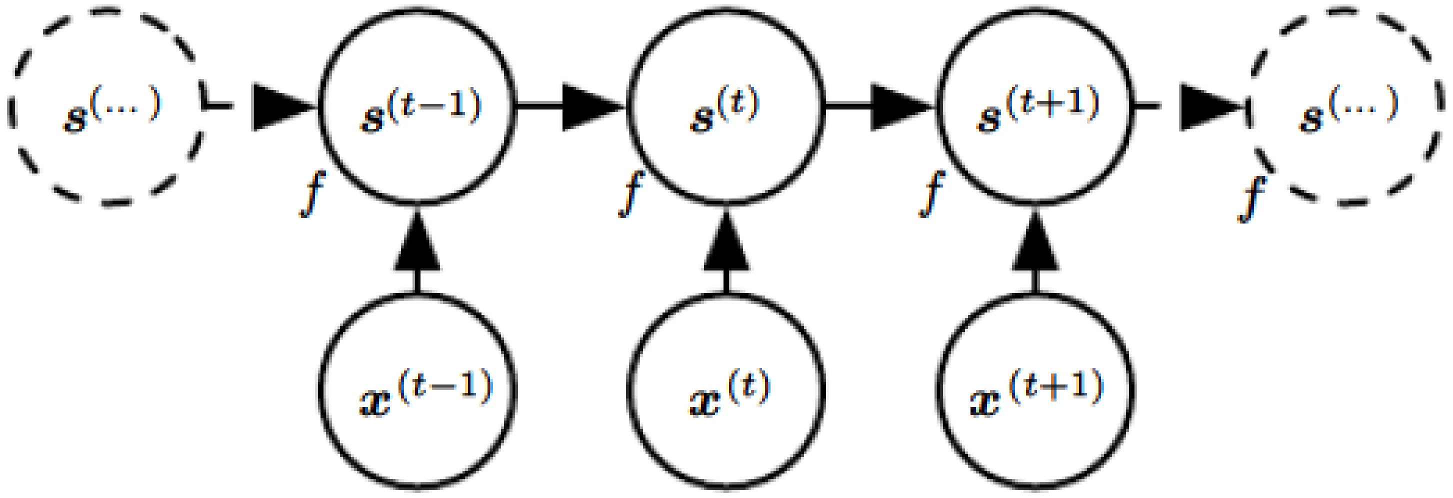
A typical dynamic system



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

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

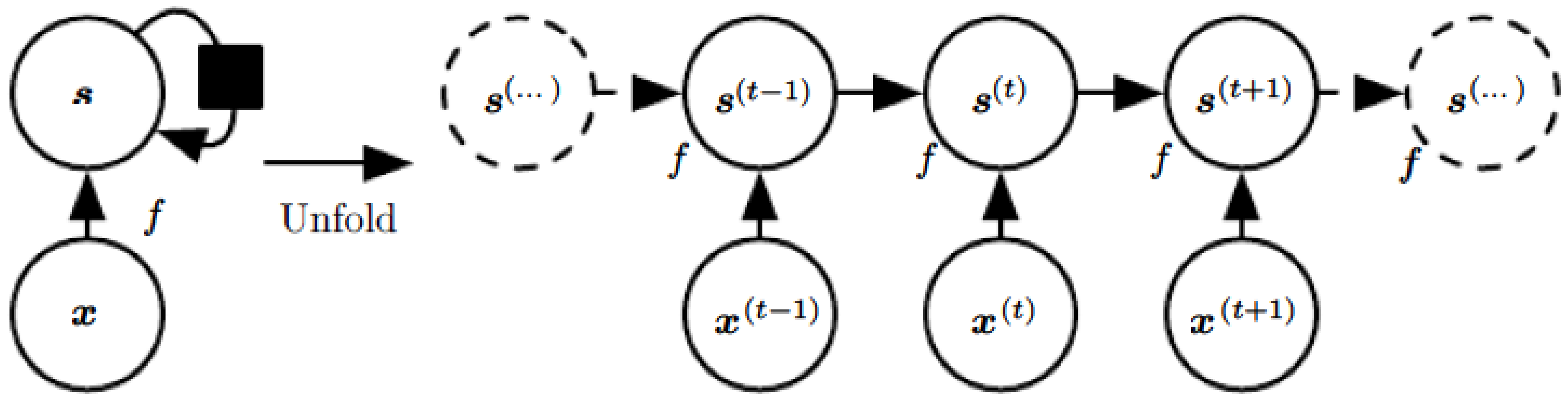
A system driven by external data



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

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

Compact view

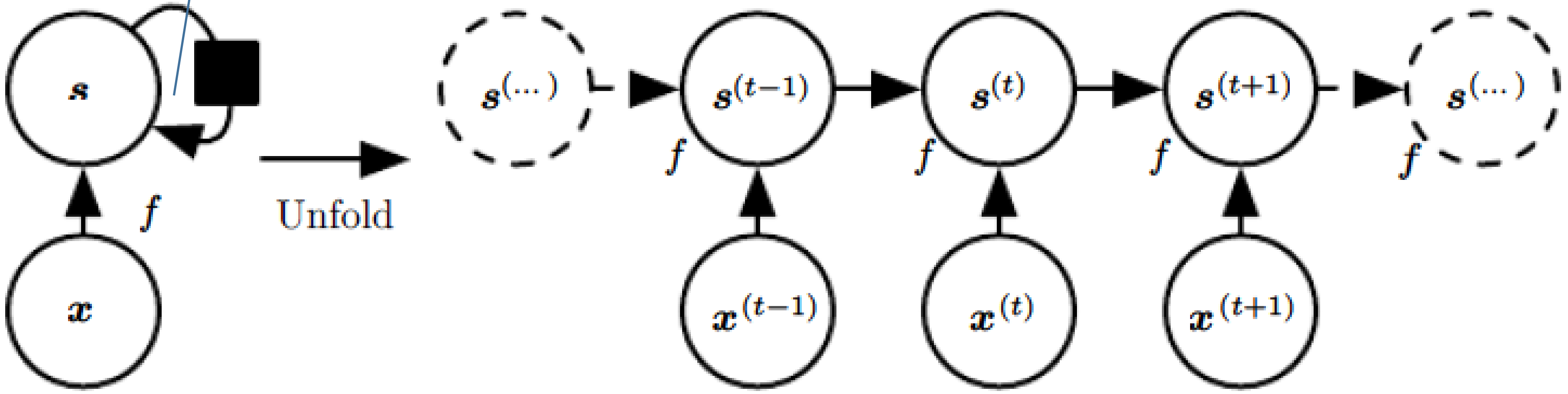


$$s^{(t+1)} = f(s^{(t)}, x^{(t+1)}; \theta)$$

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

square: one step time delay

Compact view



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

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 at every time step

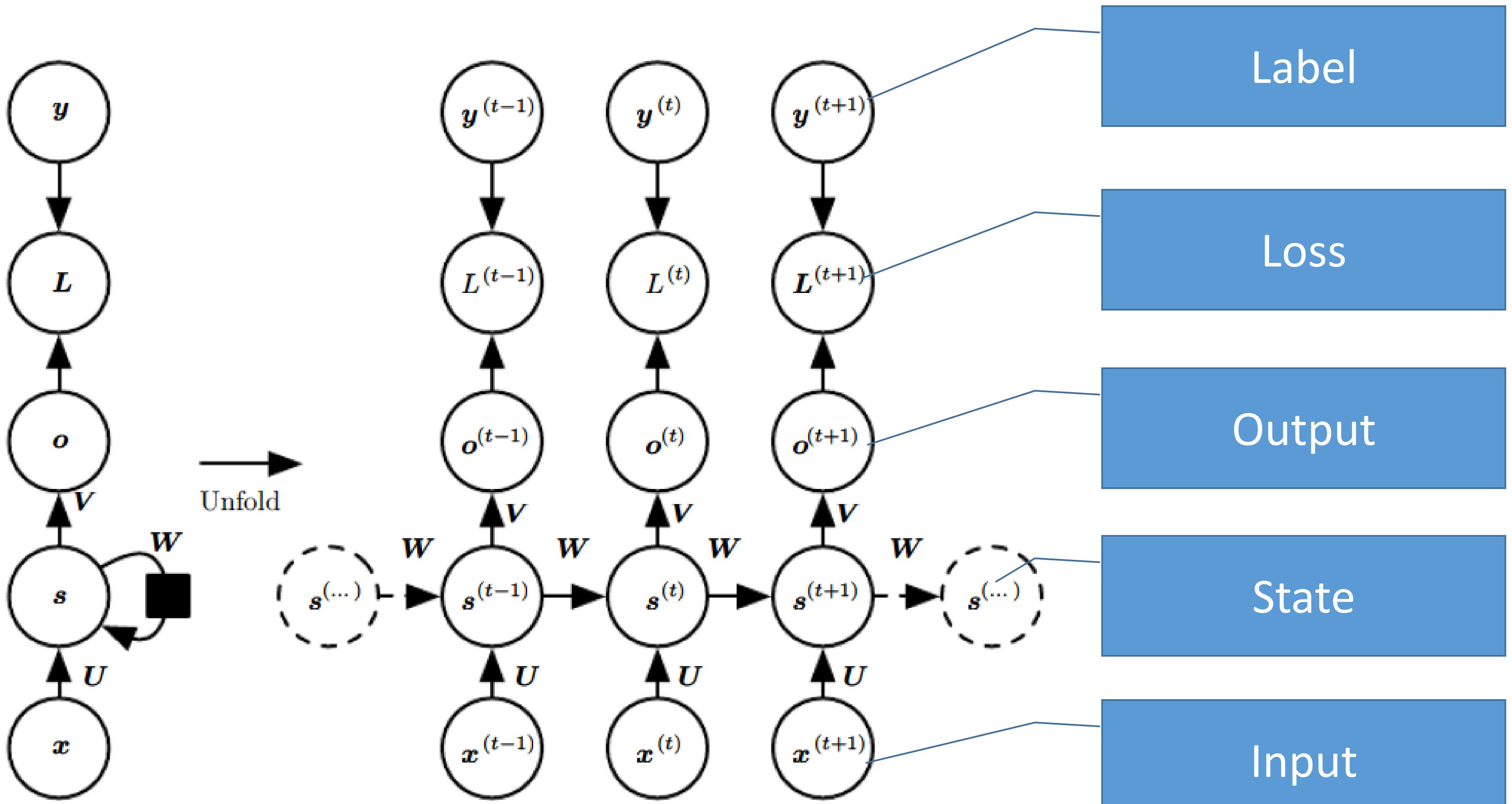
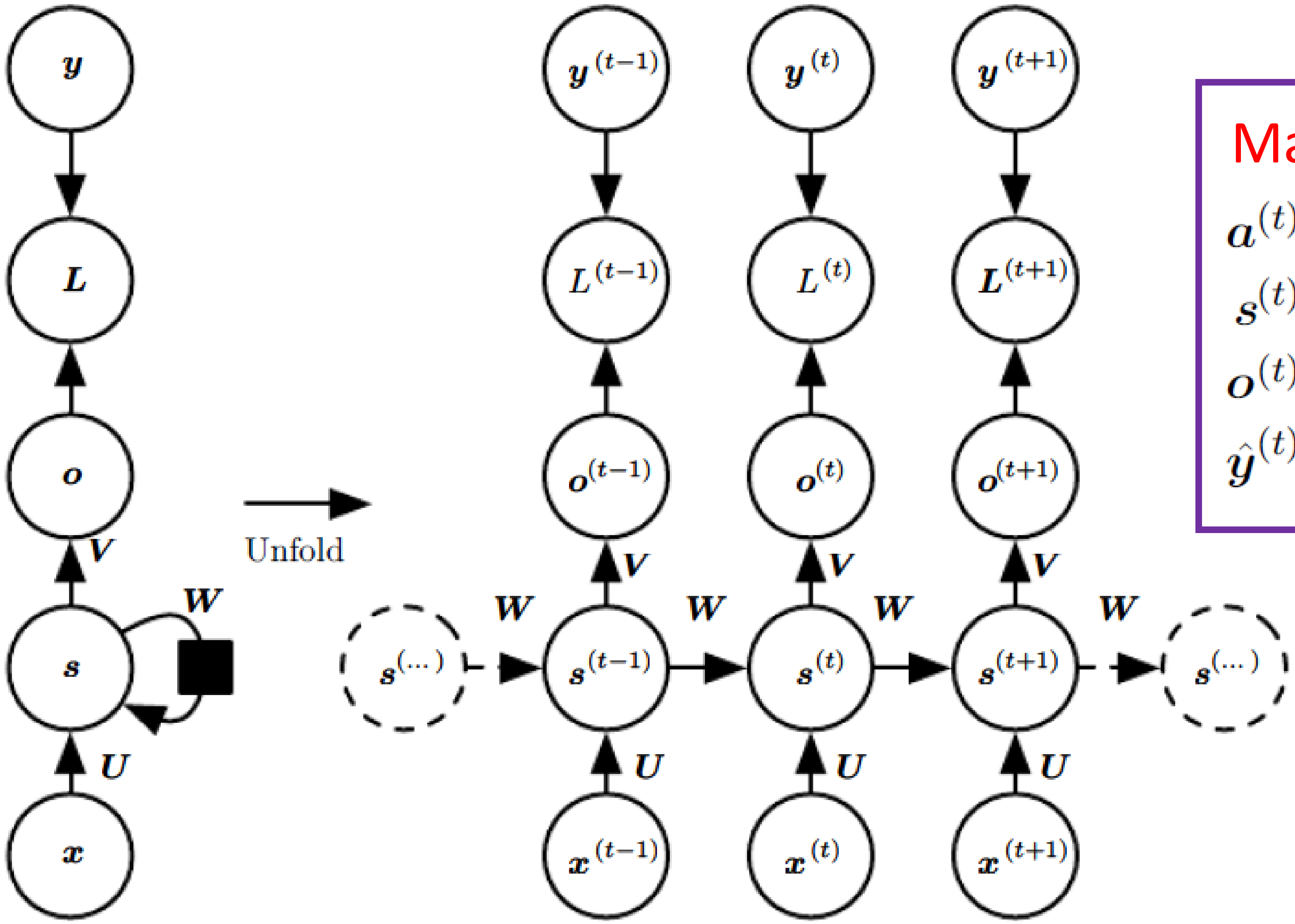


Figure from *Deep Learning*, by Goodfellow, Bengio and Courville



Math formula:

$$\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{s}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{s}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{s}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}$$

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))

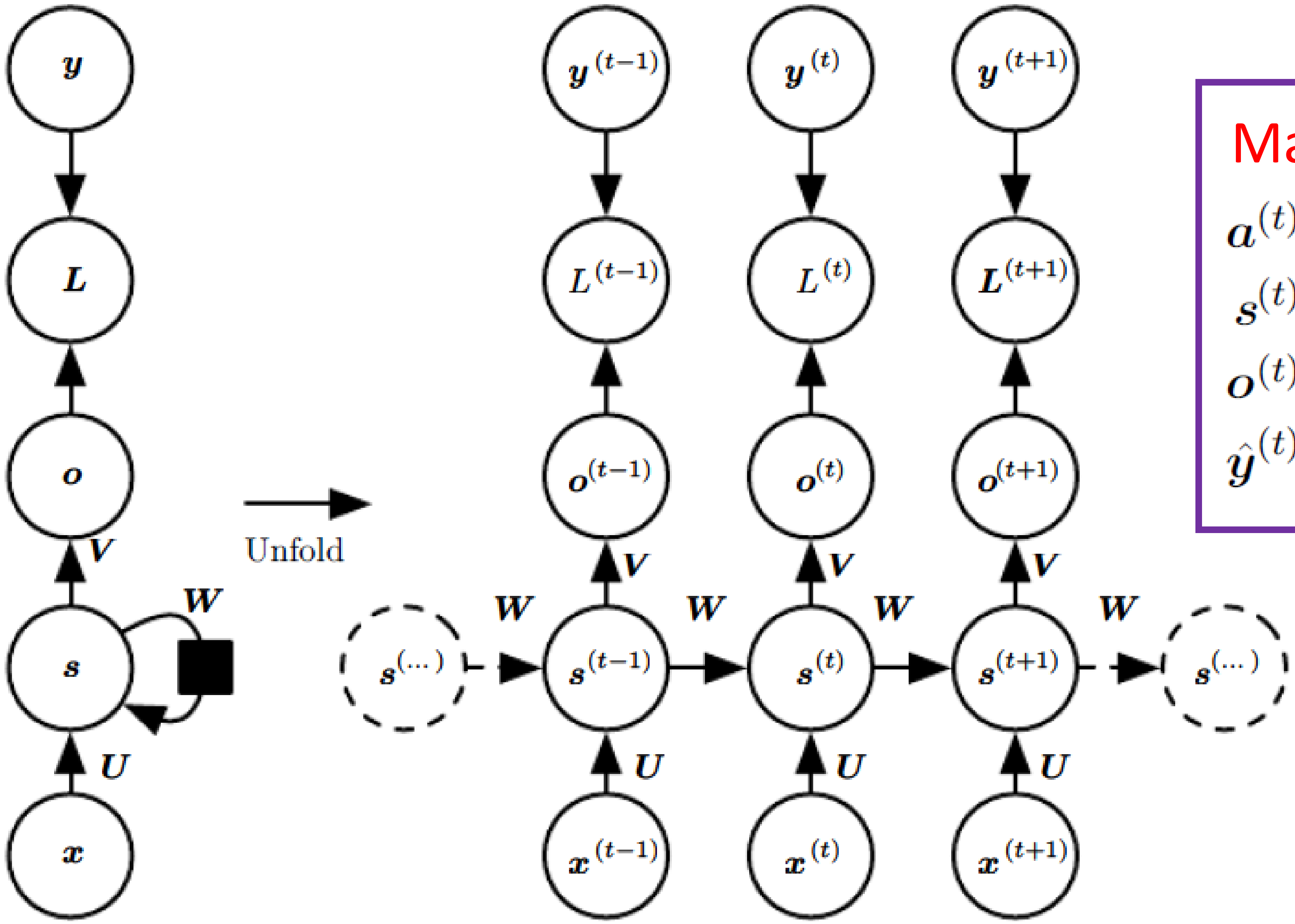
Training RNN

- 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

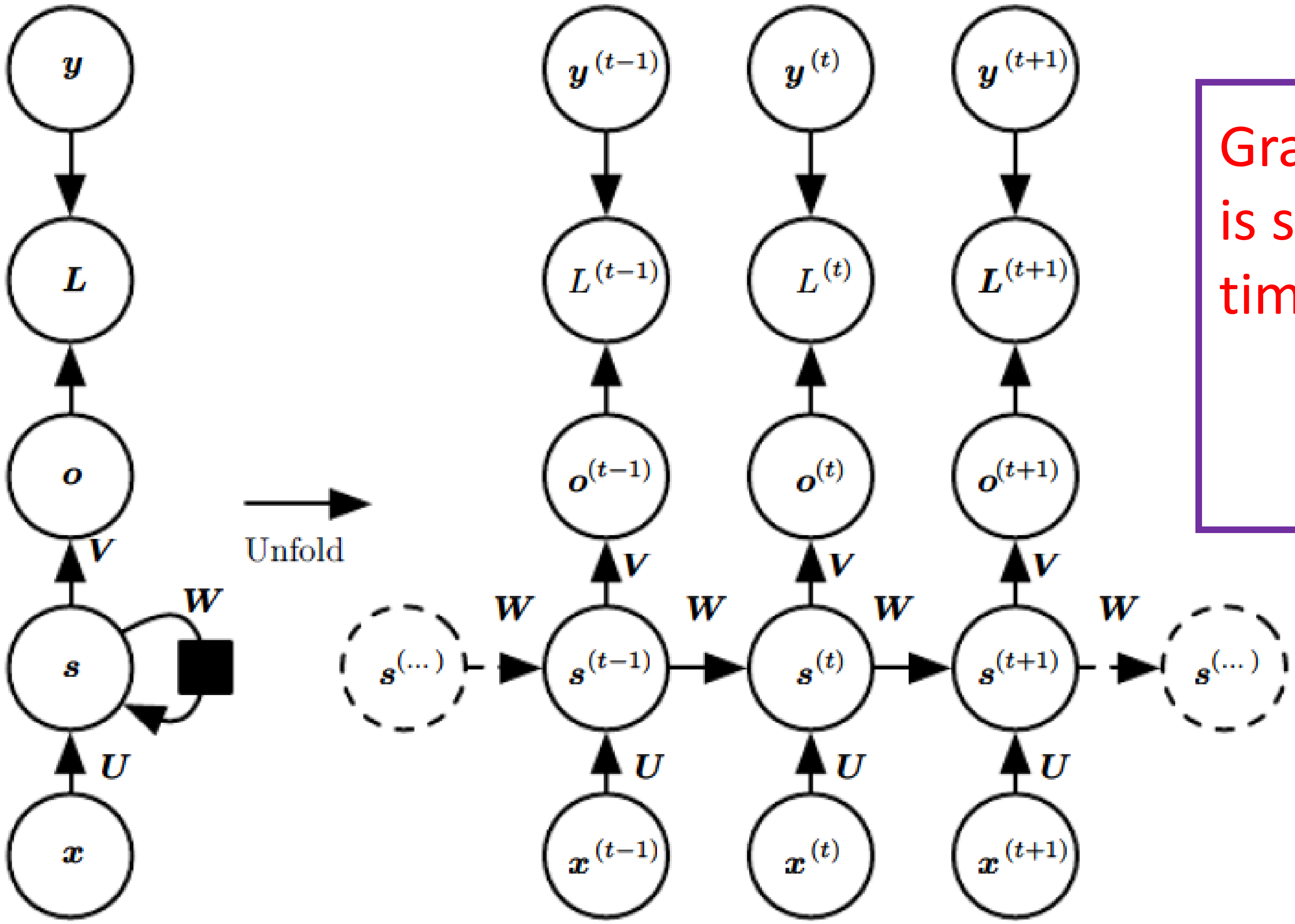
- Conceptually: first compute the gradients of the internal nodes, then compute the gradients of the parameters



Math formula:

$$
\begin{aligned}
a^{(t)} &= b + W s^{(t-1)} + U x^{(t)} \\
s^{(t)} &= \tanh(a^{(t)}) \\
o^{(t)} &= c + V s^{(t)} \\
\hat{y}^{(t)} &= \mathrm{softmax}(o^{(t)})
\end{aligned}
$$

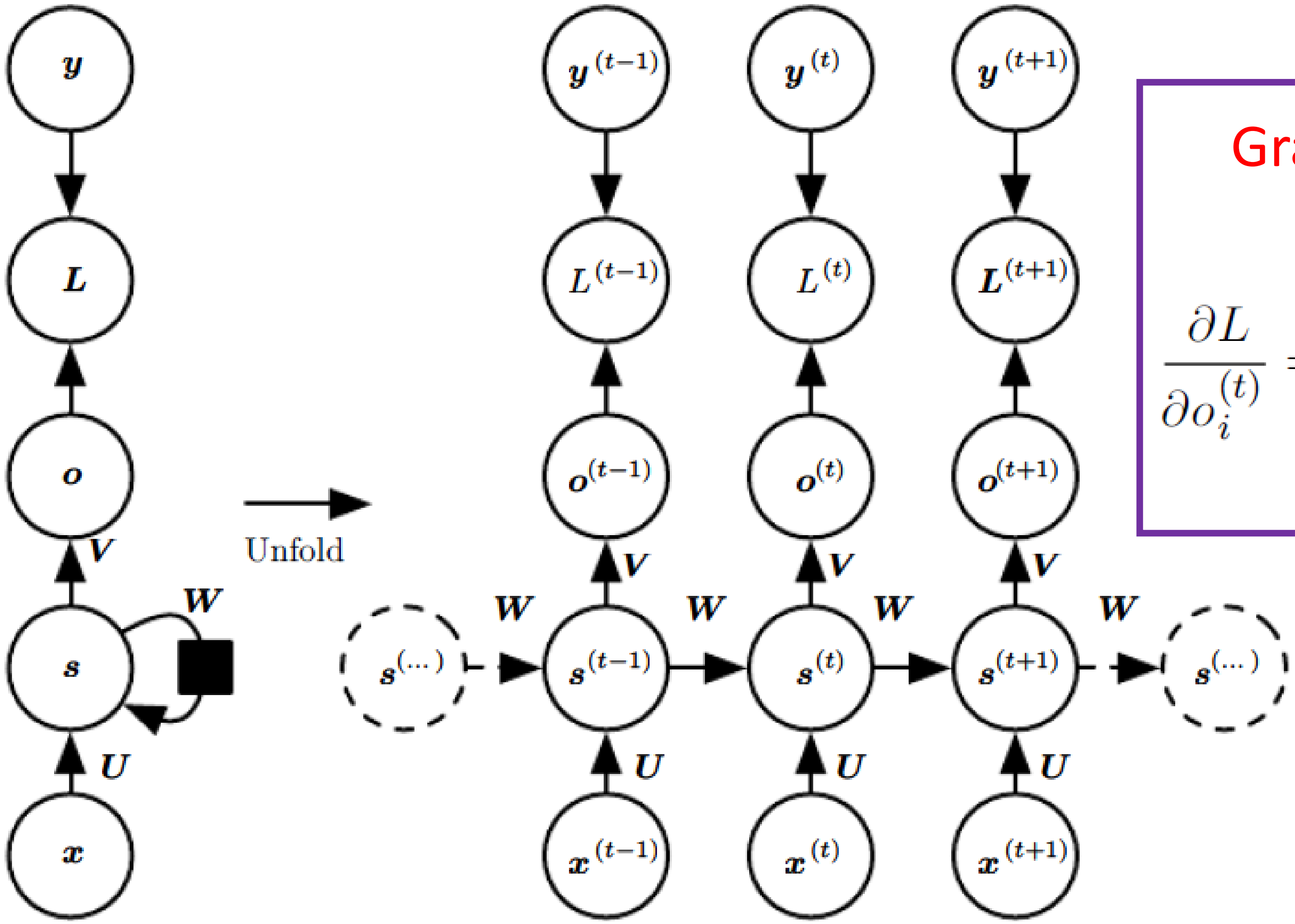
Figure from *Deep Learning*, Goodfellow, Bengio and Courville



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

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

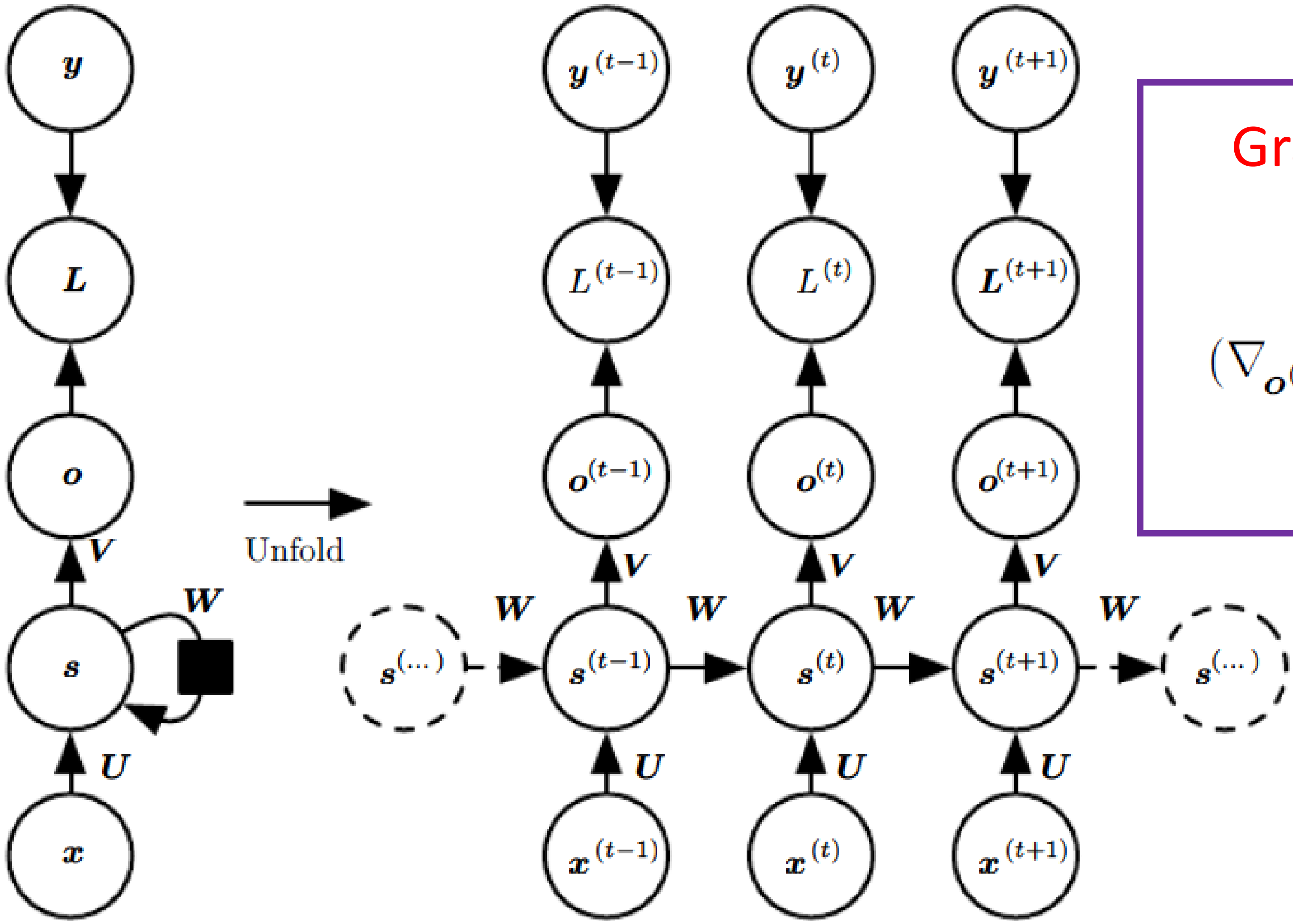
Figure from *Deep Learning*, Goodfellow, Bengio and Courville



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

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

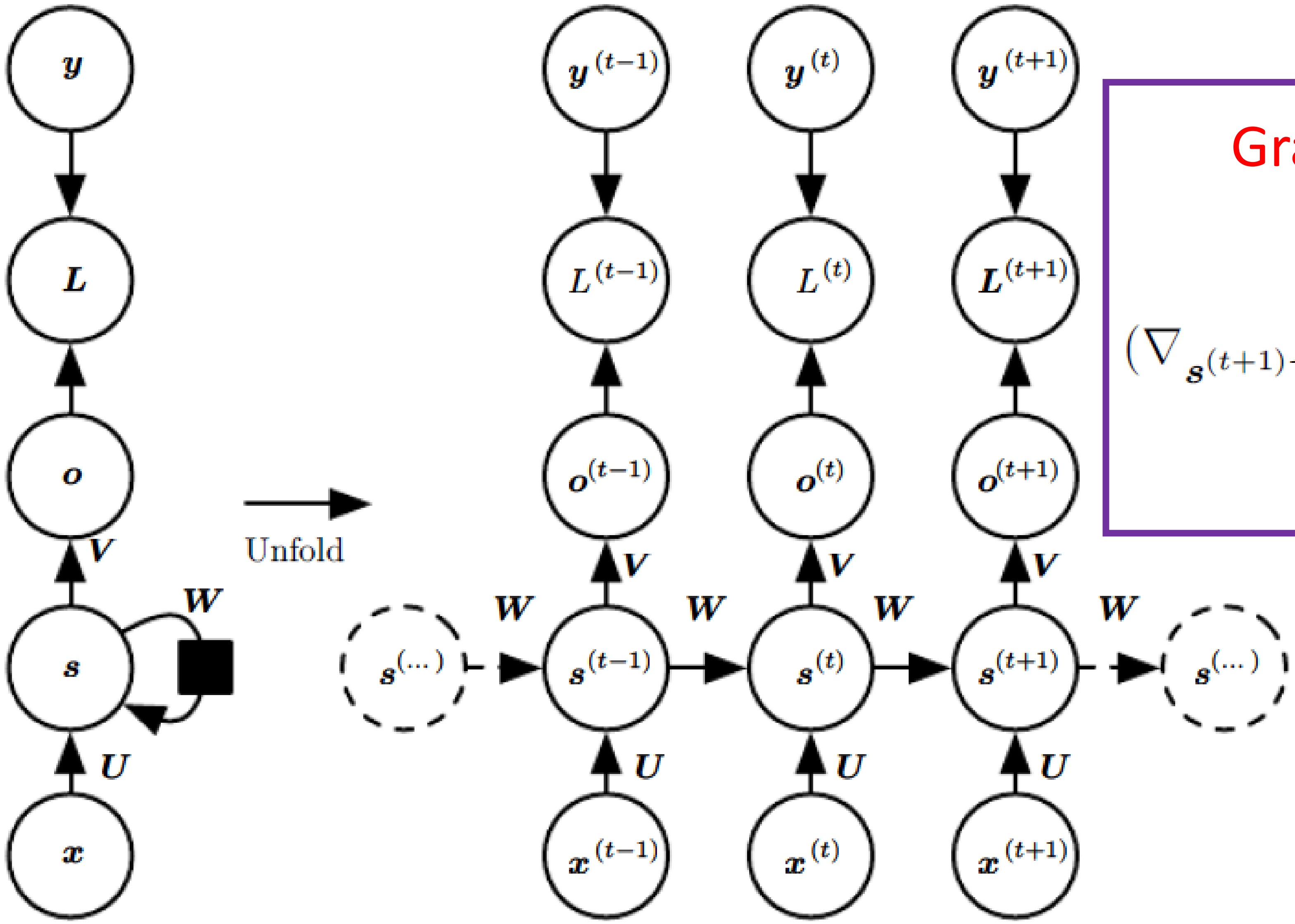
Figure from *Deep Learning*, Goodfellow, Bengio and Courville



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

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

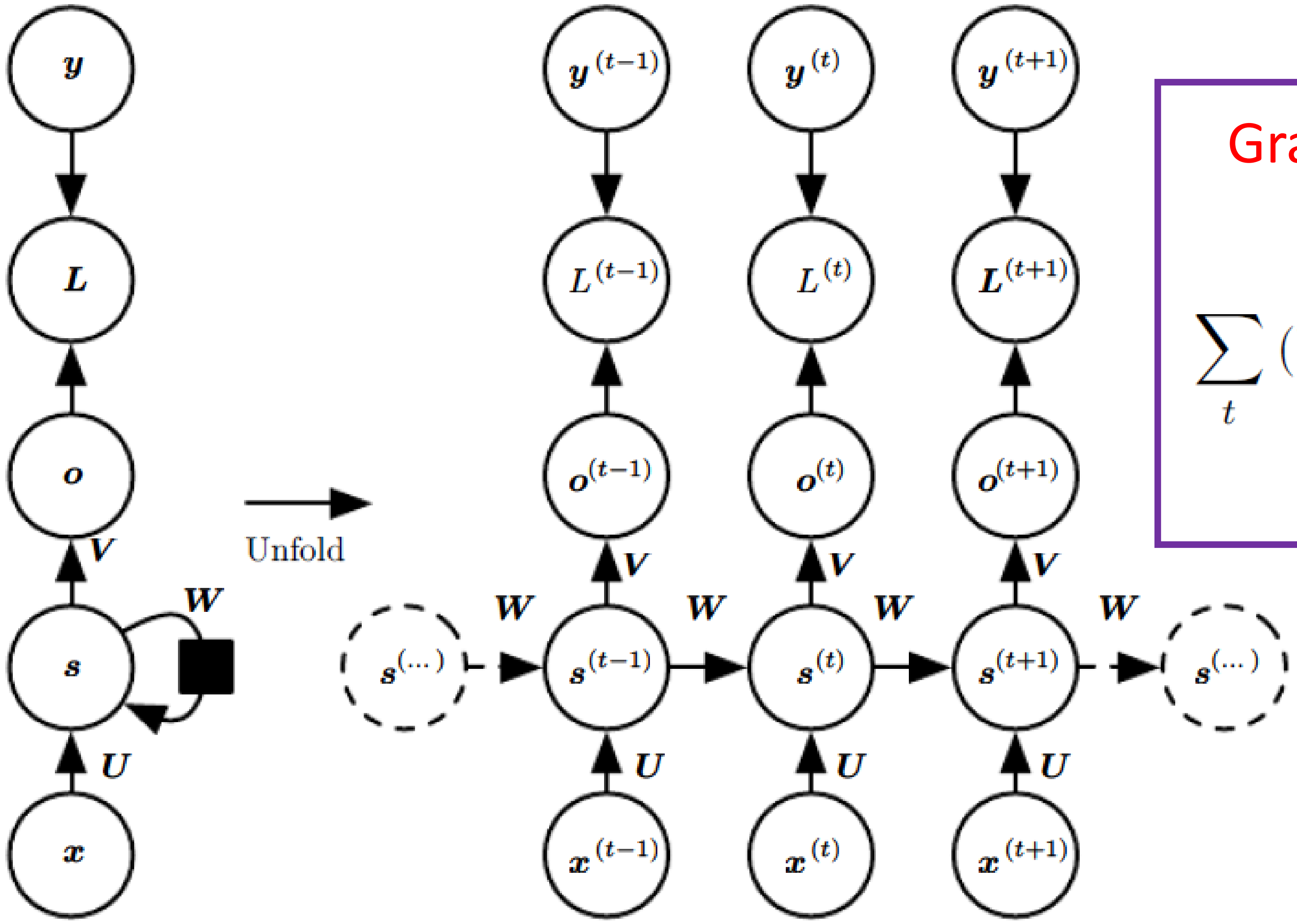
Figure from *Deep Learning*, Goodfellow, Bengio and Courville



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

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

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



Gradient at parameter V:

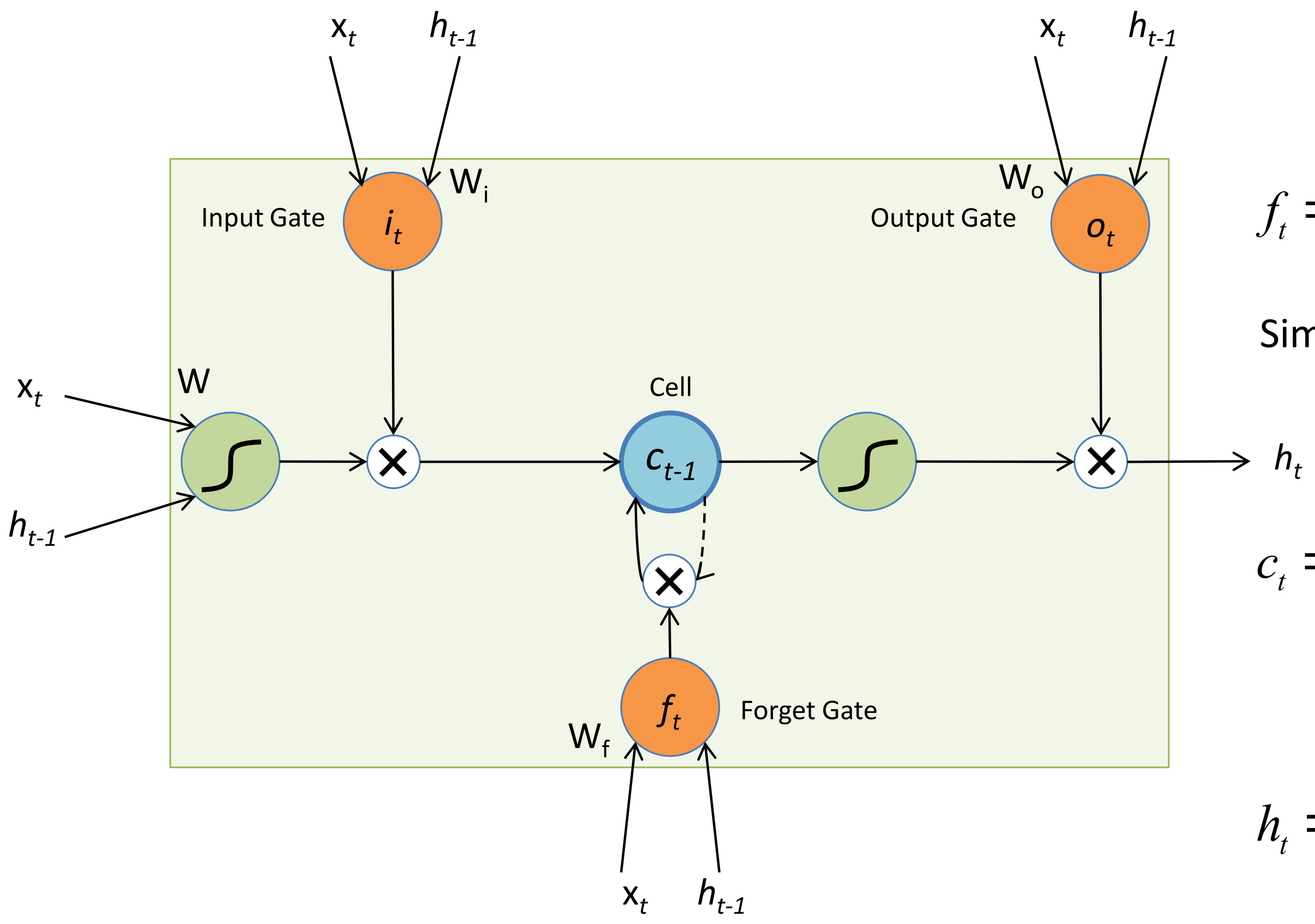
$$\sum_t \left(\nabla_{\boldsymbol{o}^{(t)}} L\right) \frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{V}} = \sum_t \left(\nabla_{\boldsymbol{o}^{(t)}} L\right) \boldsymbol{s}^{(t)\top}$$

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

The problem of exploding/vanishing gradient

- What happens to the magnitude of the gradients as we backpropagate through many layers?
  - If the weights are small, the gradients shrink exponentially.
  - If the weights are big the gradients grow exponentially.
- Typical feed-forward neural nets can cope with these exponential effects because they only have a few hidden layers.
- In an RNN trained on long sequences (*e.g.* 100 time steps) the gradients can easily explode or vanish.
  - We can avoid this by initializing the weights very carefully.
- Even with good initial weights, its very hard to detect that the current target output depends on an input from many time-steps ago.
  - So RNNs have difficulty dealing with long-range dependencies.

The Popular LSTM Cell



$$f_t = s\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

Similarly for i_t, o_t

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$h_t = o_t \text{ Ä } \tanh c_t$$

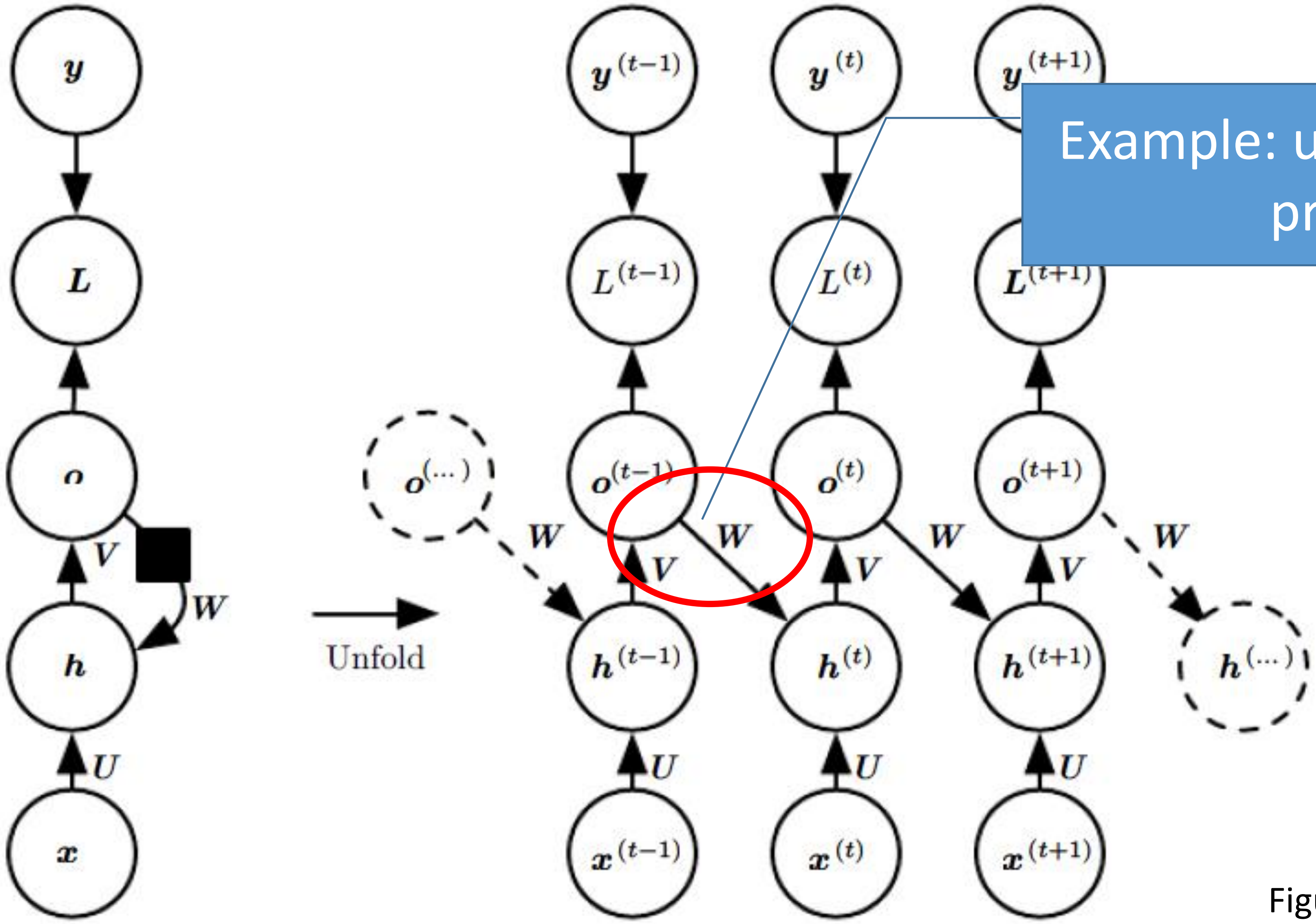
* Dashed line indicates time-lag

Some Other 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



Example: use the output at the previous step

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

Example: only output at the end

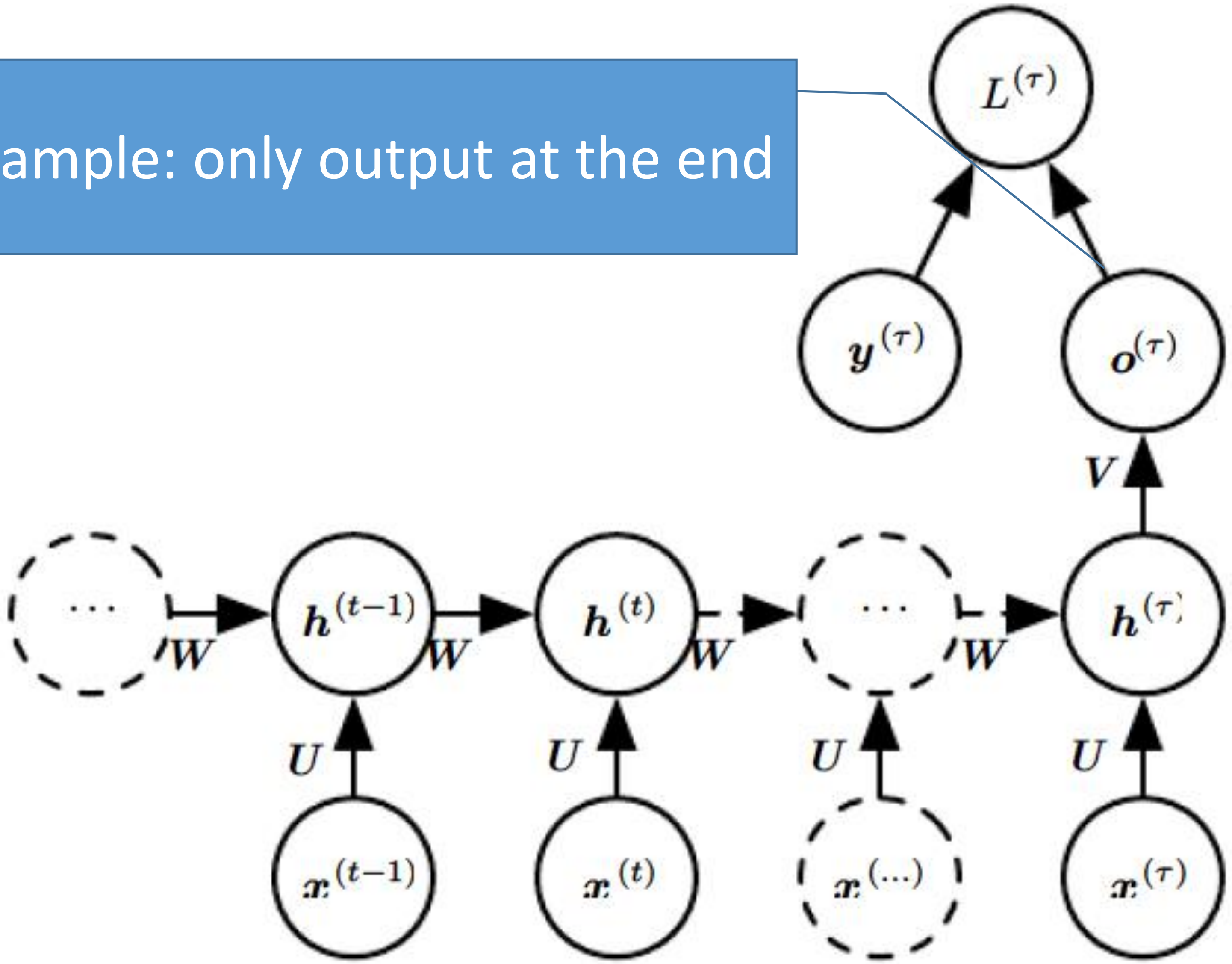


Figure from *Deep Learning*, Goodfellow, Bengio and Courville

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

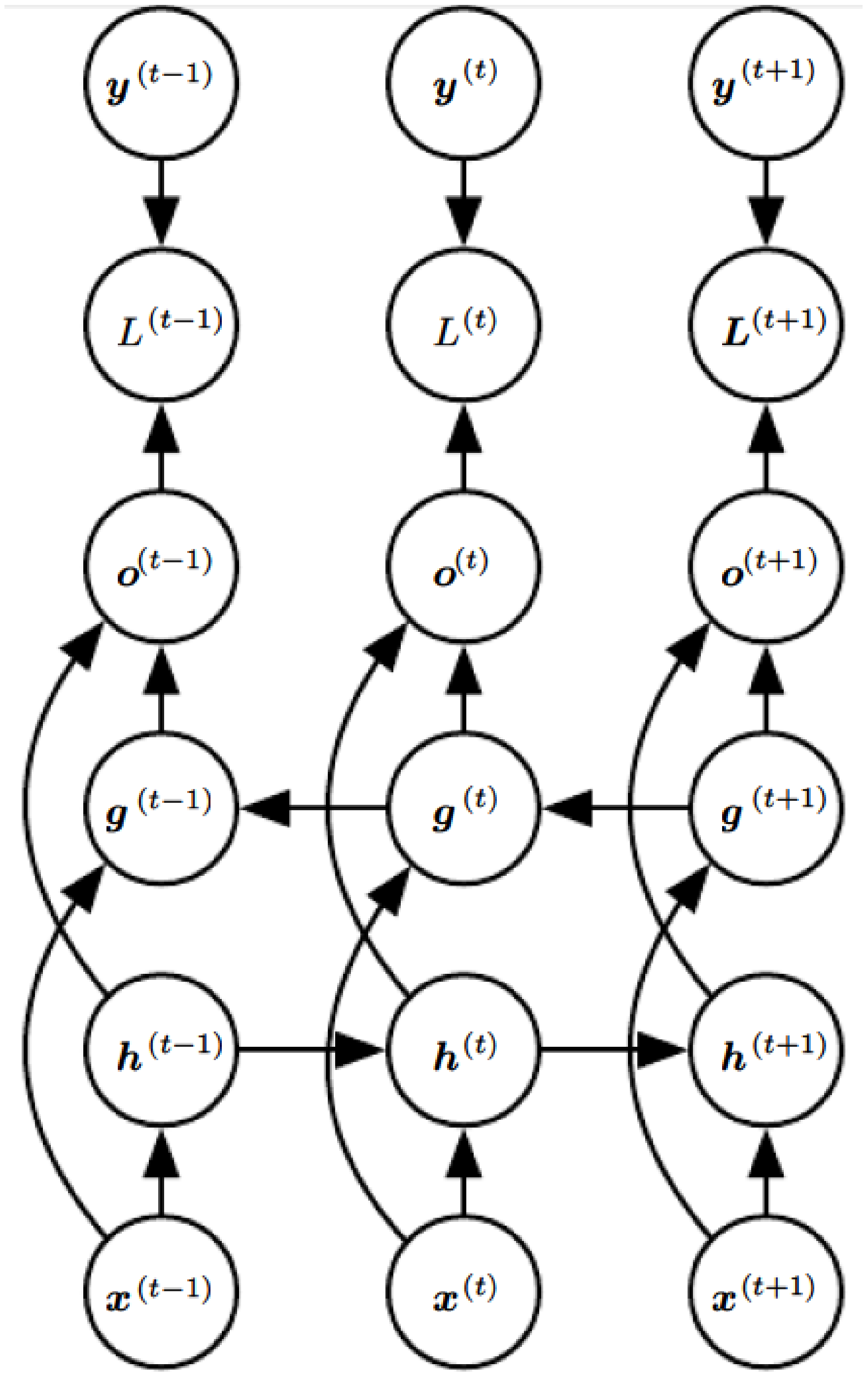


Figure from *Deep Learning*, Goodfellow, Bengio and Courville

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

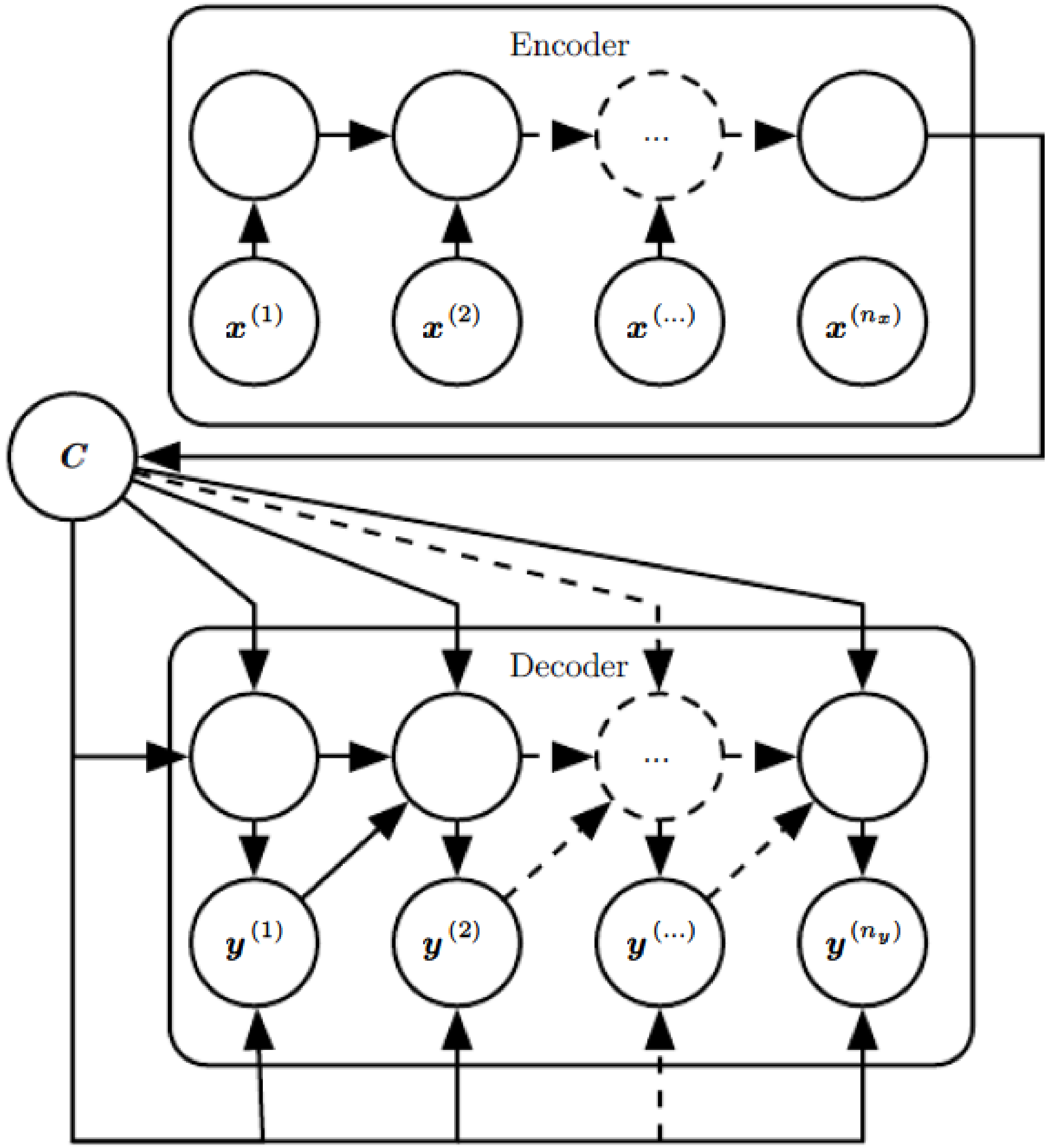


Figure from *Deep Learning*, Goodfellow, Bengio and Courville