Sequence Modeling: Recurrent and Recursive Nets

Lecture slides for Chapter 10 of *Deep Learning*
www.deeplearningbook.org
Ian Goodfellow
2016-09-27
Adapted by m.n. for CMPS 392

RNN

- RNNs are a family of neural networks for processing sequential data
- Recurrent networks can scale to much longer sequences than would be practical for networks without sequence-based specialization.
- Most recurrent networks can also process sequences of variable length.
- Based on parameter sharing
  - If we had separate parameters for each value of the time index,
    - we could not generalize to sequence lengths not seen during training,
    - nor share statistical strength across different sequence lengths,
    - and across different positions in time.

Example

- Consider the two sentences “I went to Nepal in 2009” and “In 2009, I went to Nepal.”
- How a machine learning can extract the year information?
  - A traditional fully connected feedforward network would have separate parameters for each input feature,
  - so it would need to learn all of the rules of the language separately at each position in the sentence.
- By comparison, a recurrent neural network shares the same weights across several time steps.

RNN vs. 1D convolutional

- The output of convolution is a sequence where each member of the output is a function of a small number of neighboring members of the input.
- Recurrent networks share parameters in a different way.
  - Each member of the output is a function of the previous members of the output
  - Each member of the output is produced using the same update rule applied to the previous outputs.
  - This recurrent formulation results in the sharing of parameters through a very deep computational graph.

Classical Dynamical Systems

$$\boldsymbol{s}^{(t)} = f\left(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta}\right)$$

$$\boldsymbol{s}^{(3)} = f\left(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}\right) = f\left(f\left(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}\right); \boldsymbol{\theta}\right)$$

$s^{(t)}$: state of the system

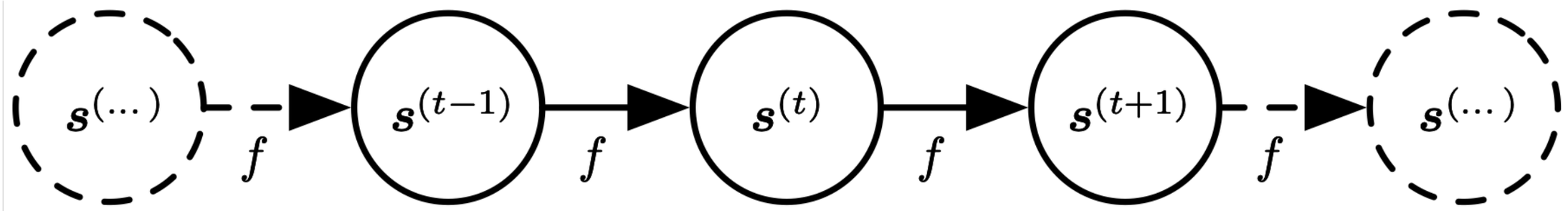


Figure 10.1

Other example: consider a dynamical system driven by an external signal $\boldsymbol{x}^{(t)}$:

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

Unfolding Computation Graphs

$$\boldsymbol{h}^{(t)} = f\left(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}\right)$$
$$= g^{(t)}\left(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}\right)$$

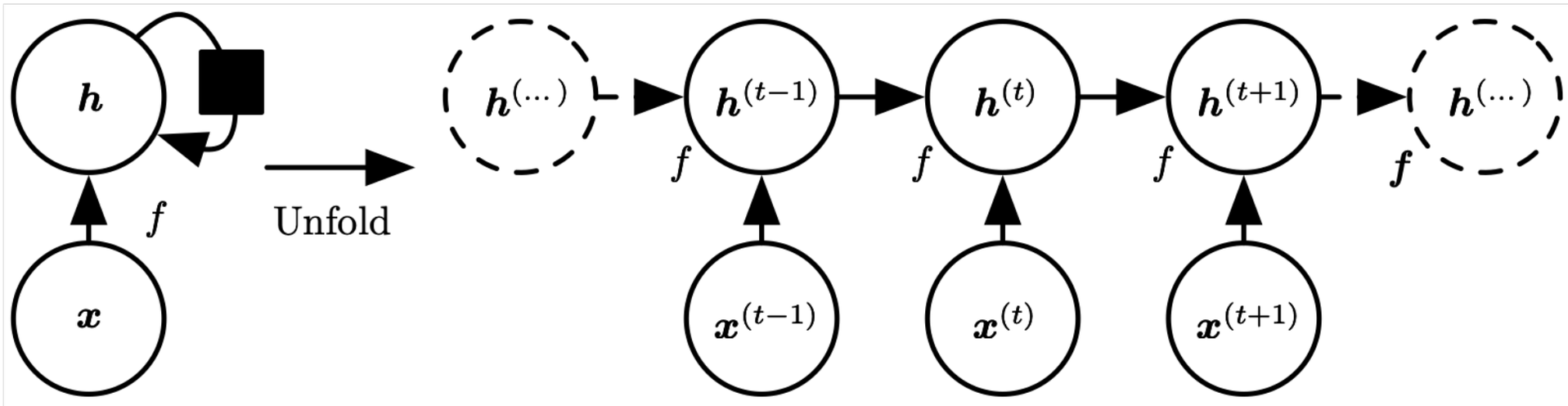
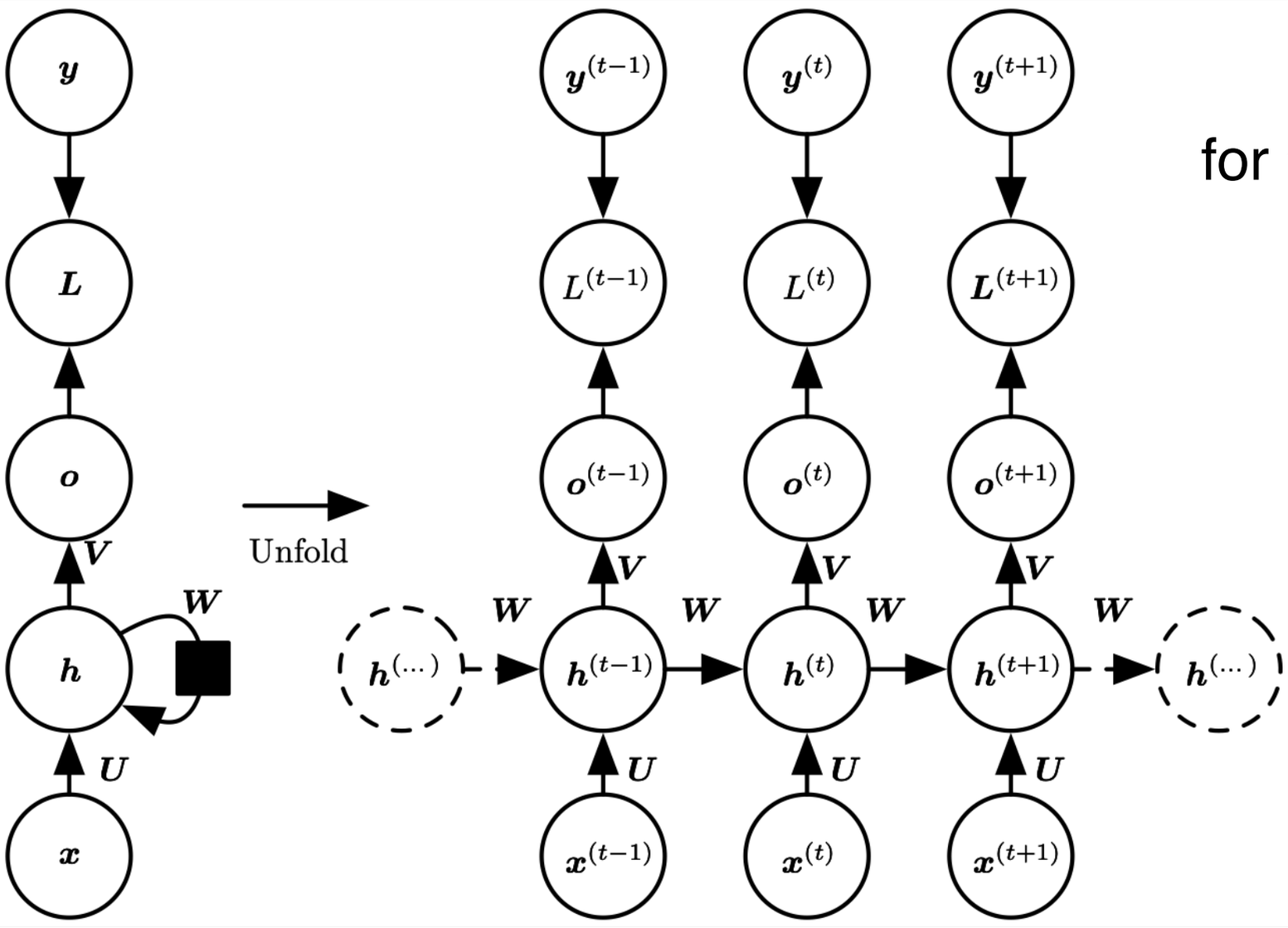


Figure 10.2

The network typically learns to use $\boldsymbol{h}^{(t)}$ as a kind of lossy summary of the task-relevant aspects of the past sequence of inputs up to t

This summary is necessarily lossy, since it maps an arbitrary length sequence $\boldsymbol{x}^{\boldsymbol{t}}, \boldsymbol{x}^{\boldsymbol{t-1}}, \boldsymbol{x}^{\boldsymbol{t-2}}, \ldots, \boldsymbol{x}^{\boldsymbol{2}}, \boldsymbol{x}^{\boldsymbol{1}}$ to a fixed length vector $\boldsymbol{h}^{t}$

Recurrent Hidden Units



for $t = 1, \dots, \tau$:

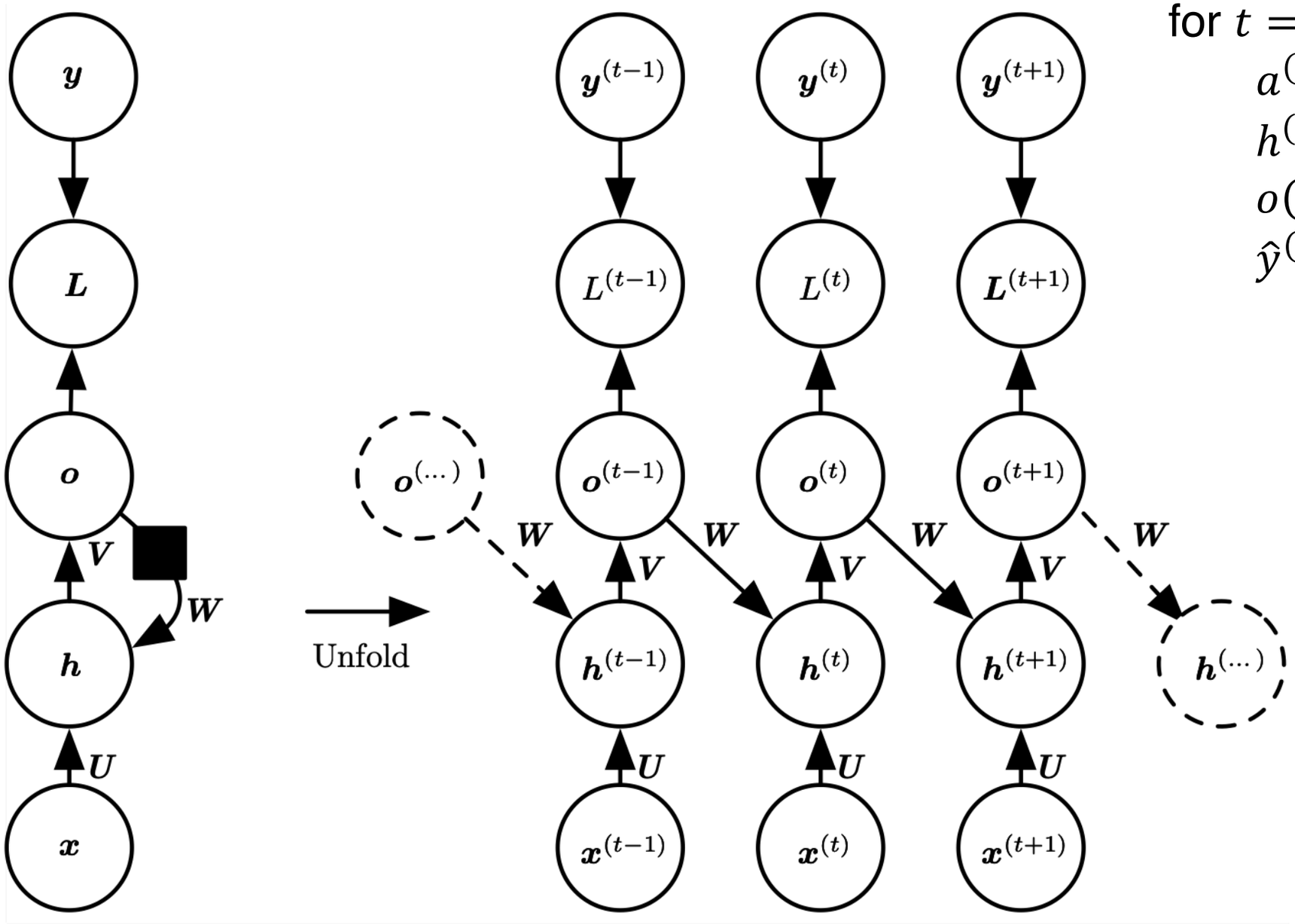
$$a^{(t)} = b + W h^{(t-1)} + U x^{(t)}$$
$$h^{(t)} = \tanh\left(a^{(t)}\right)$$
$$o(t) = c + V\, h(t)$$
$$\hat{y}^{(t)} = \text{softmax}\left(o^{(t)}\right)$$

U: input-to-hidden
V: hidden-to-output
W: hidden-to-hidden

$$L\left(\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(\tau)}\}, \{y^{(1)}, y^{(2)}, \dots, y^{(\tau)}\}\right)$$
$$= \sum_t L(t) = -\sum_t \log p_{model}(y^{(t)} | \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(t)}\}$$

back-propagation through time or BPTT

Recurrence through only the Output (less powerful, easier to train)



for $t = 1, \dots, \tau$:

$$a^{(t)} = b + W o^{(t-1)} + U x^{(t)}$$
$$h^{(t)} = \tanh\left(a^{(t)}\right)$$
$$o(t) = c + V\, h(t)$$
$$\hat{y}^{(t)} = \text{softmax}\left(o^{(t)}\right)$$

Figure 10.4

Sequence Input, Single Output

for $t = 1, \ldots, \tau$:

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

$$h^{(t)} = \tanh\left(a^{(t)}\right)$$

$$o(\tau) = c + V\,h(\tau)$$

$$\hat{y}^{(\tau)} = \text{softmax}\left(o^{(\tau)}\right)$$

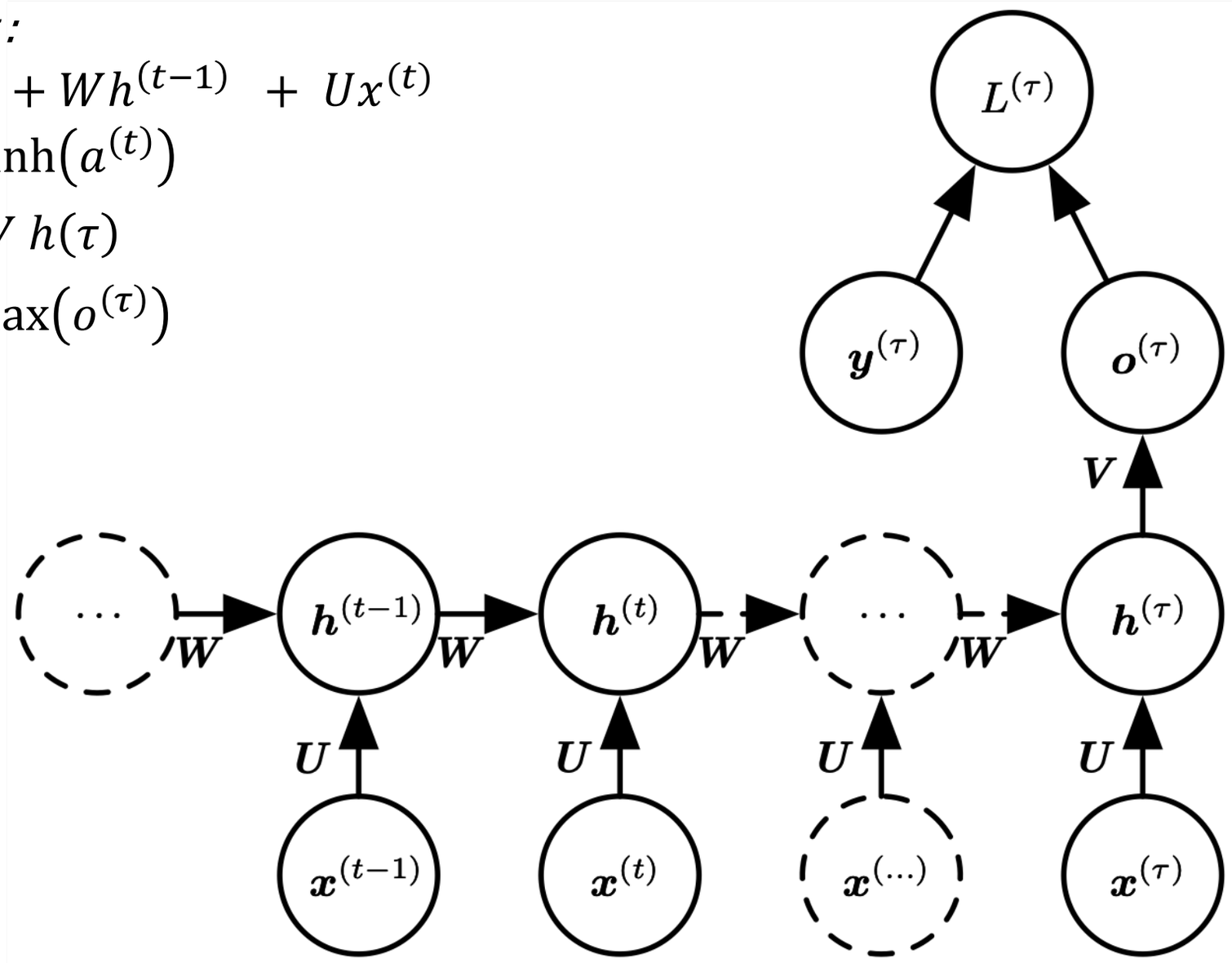
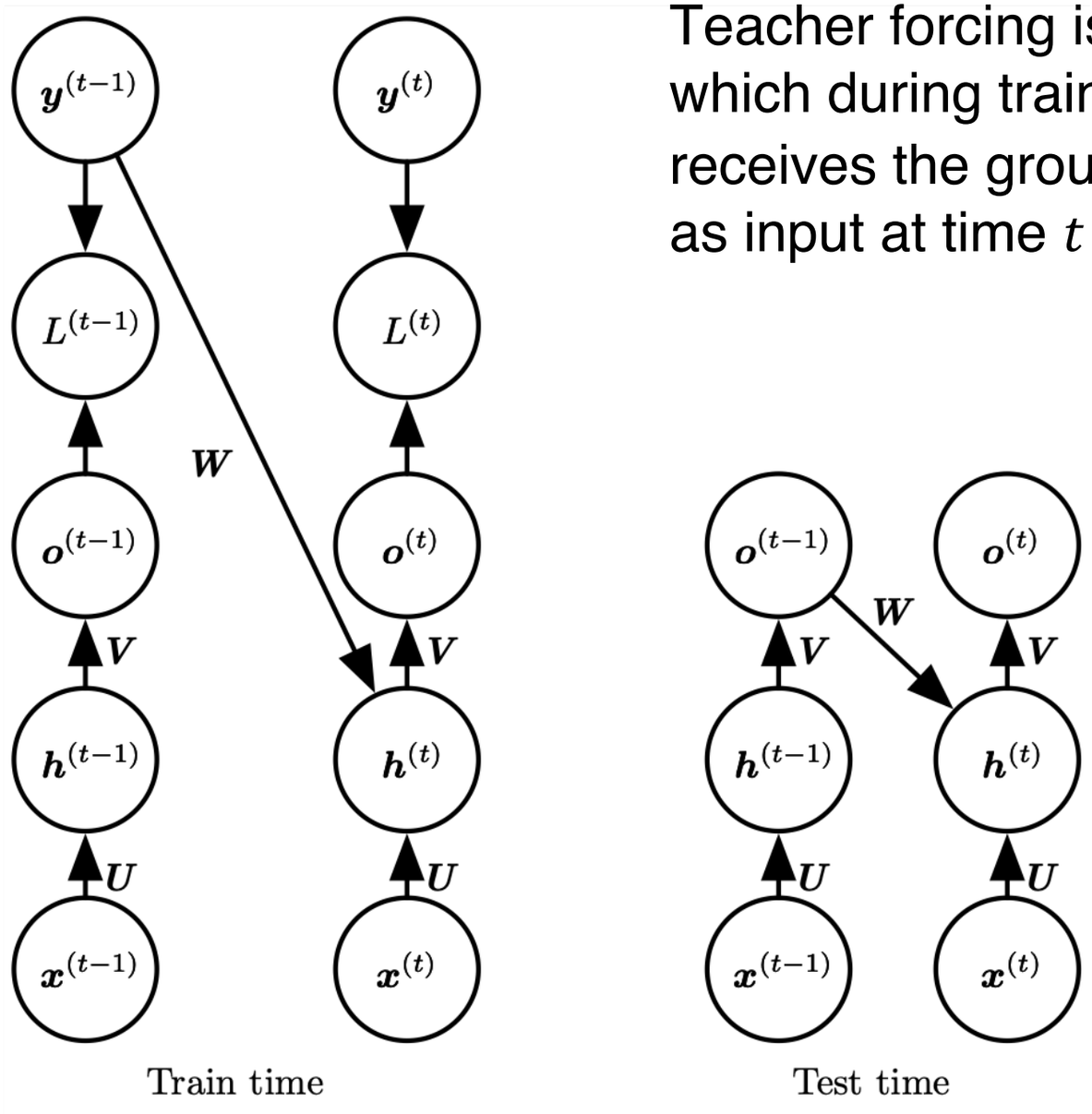


Figure 10.5

Teacher Forcing



Teacher forcing is a procedure in which during training the model receives the ground truth output $y^{(t)}$ as input at time $t + 1$.

Figure 10.6

Computing the gradient

- $L^{(t)} = -\sum p_i \log softmax\left(o_i^{(t)}\right) = -\sum p_i \log \hat{y}_i^{(t)}$
- $\frac{\partial L^{(t)}}{\partial o_i^{(t)}} = -\frac{\partial}{\partial o_i^{(t)}}\left(\sum_{j\neq i} p_j \log \hat{y}_j^{(t)} + p_i \log \hat{y}_i^{(t)}\right)$
  - $\frac{\partial}{\partial o_i^{(t)}} \log \hat{y}_i^{(t)} = 1 - \hat{y}_i^{(t)}$
  - $\frac{\partial}{\partial o_i^{(t)}} \sum_{j\neq i} p_j \log \hat{y}_j^{(t)} = \frac{\partial}{\partial o_i^{(t)}}\left(\sum_{j\neq i} p_j o_j^{(t)} + \sum_{j\neq i} p_j \log \sum_k \exp\left(o_k^{(t)}\right)\right) =$
    $\sum_{j\neq i} p_j \, \hat{y}_i^{(t)} = \hat{y}_i^{(t)} \sum_{j\neq i} p_j = \hat{y}_i^{(t)}(1 - p_i)$
- $\frac{\partial L^{(t)}}{\partial o_i^{(t)}} = -p_i\left(1 - \hat{y}_i^{(t)}\right) - (1 - p_i)\hat{y}_i^{(t)} = -p_i + \hat{y}_i^{(t)}$
- $\frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}}$

$$\nabla_{\boldsymbol{o}^{(t)}} L^{(t)} = \hat{\boldsymbol{y}}^{(t)} - \mathbf{1}_{y^{(t)}}$$

Computing the gradient

for $t = 1, \ldots, \tau$:

$$a^{(t)} = b + W h^{(t-1)} + U x^{(t)}$$
$$h^{(t)} = \tanh\left(a^{(t)}\right)$$
$$o^{(t)} = c + V h^{(t)}$$
$$\hat{y}^{(t)} = \text{softmax}\left(o^{(t)}\right)$$

- $\nabla_{\boldsymbol{h}^{(\tau)}} L = V^T \nabla_{\boldsymbol{o}^{(\tau)}} L$
- $\nabla_{\boldsymbol{a}^{(\tau)}} L = \left(\frac{\partial \boldsymbol{h}^{(\tau)}}{\partial \boldsymbol{a}^{(\tau)}}\right)^T \nabla_{\boldsymbol{h}^{(\tau)}} L = diag\left(1 - h^{(\tau)^2}\right) \nabla_{\boldsymbol{h}^{(\tau)}} L$
- $\nabla_{\boldsymbol{h}^{(\tau-1)}} L = W^T \nabla_{\boldsymbol{a}^{(\tau)}} L + V^T \nabla_{\boldsymbol{o}^{(\tau-1)}} L$

$$\nabla_{\boldsymbol{h}^{(\tau-1)}} L = W^T diag\left(1 - h^{(\tau)^2}\right) \nabla_{\boldsymbol{h}^{(\tau)}} L + V^T \nabla_{\boldsymbol{o}^{(\tau-1)}} L$$

- Valid for any $t < \tau$

$$\nabla_{\boldsymbol{h}^{(t)}} L = W^T diag\left(1 - h^{(t+1)^2}\right) \nabla_{\boldsymbol{h}^{(t+1)}} L + V^T \nabla_{\boldsymbol{o}^{(t)}} L$$

Gradients on the parameter nodes

for $t = 1, \ldots, \tau$:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$
$$h^{(t)} = \tanh\left(a^{(t)}\right)$$
$$o^{(t)} = c + V h^{(t)}$$
$$\hat{y}^{(t)} = \text{softmax}\left(o^{(t)}\right)$$

- $\nabla_c L = \sum_t \nabla_{o^{(t)}} L$
- $\nabla_V L = \sum_t \left(\nabla_{o^{(t)}} L\right) h^{(t)^T}$
- $\nabla_b L = \sum_t \nabla_{a^{(t)}} L = \sum_t diag\left(1 - h^{(t)^2}\right) \nabla_{\boldsymbol{h}^{(t)}} L$
- $\nabla_W L = \sum_t \left(\nabla_{a^{(t)}} L\right) h^{(t-1)^T} = \sum_t \left(diag\left(1 - h^{(t)^2}\right) \nabla_{\boldsymbol{h}^{(t)}} L\right) h^{(t-1)^T}$
- $\nabla_U L = \sum_t \left(\nabla_{a^{(t)}} L\right) x^{(t)^T} = \sum_t \left(diag\left(1 - h^{(t)^2}\right) \nabla_{\boldsymbol{h}^{(t)}} L\right) x^{(t)^T}$

Fully Connected Graphical Model

One way to interpret an RNN as a graphical model is to view the RNN as defining a graphical model whose structure is the complete graph:

Here we consider a sequence of random variables $y^{(t)}$ with no inputs x.

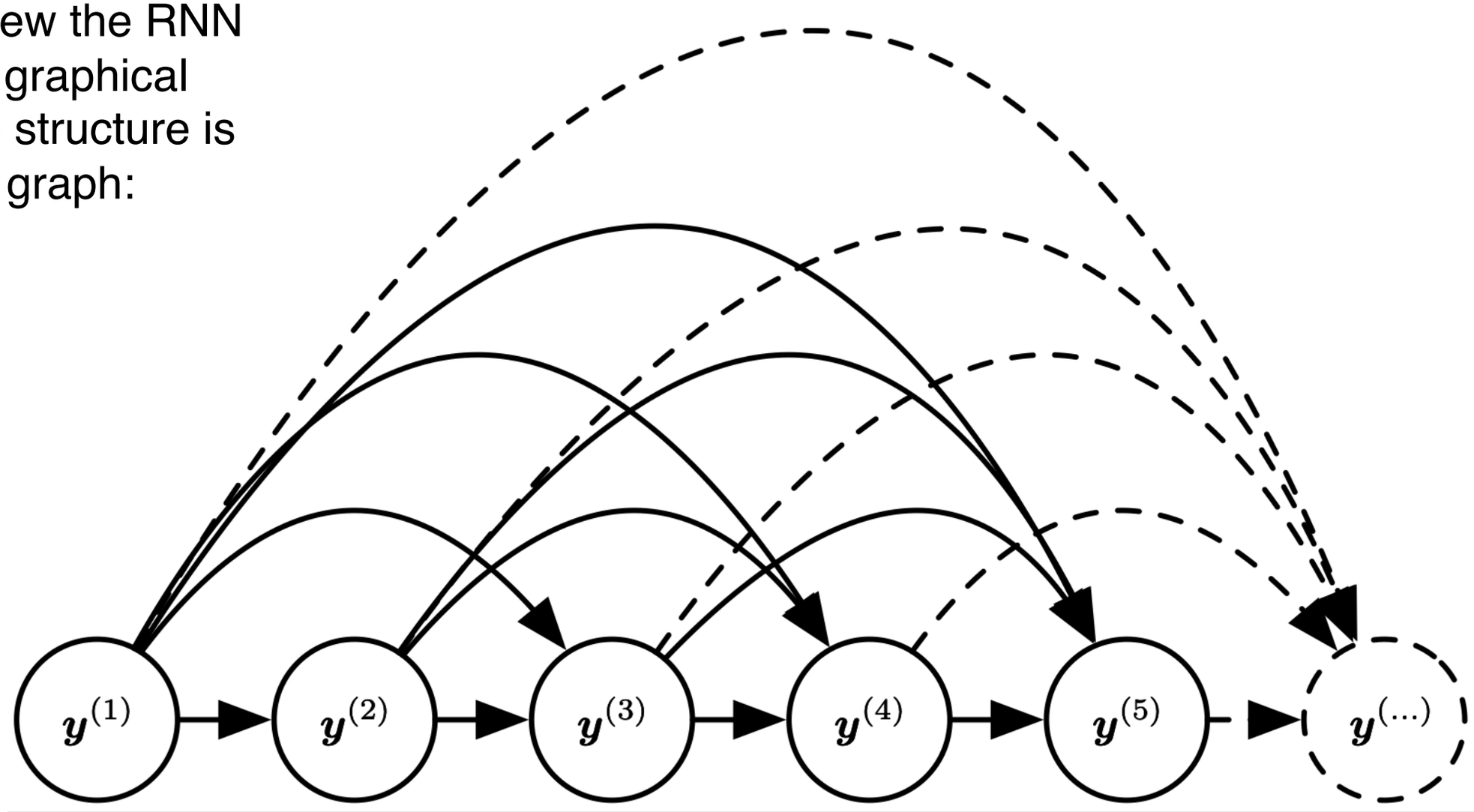


Figure 10.7

RNN Graphical Model

Incorporating the $h^{(t)}$ nodes in the graphical model decouples the past and the future, acting as an intermediate quantity between them.

RNNs obtain the same full connectivity but efficient parametrization

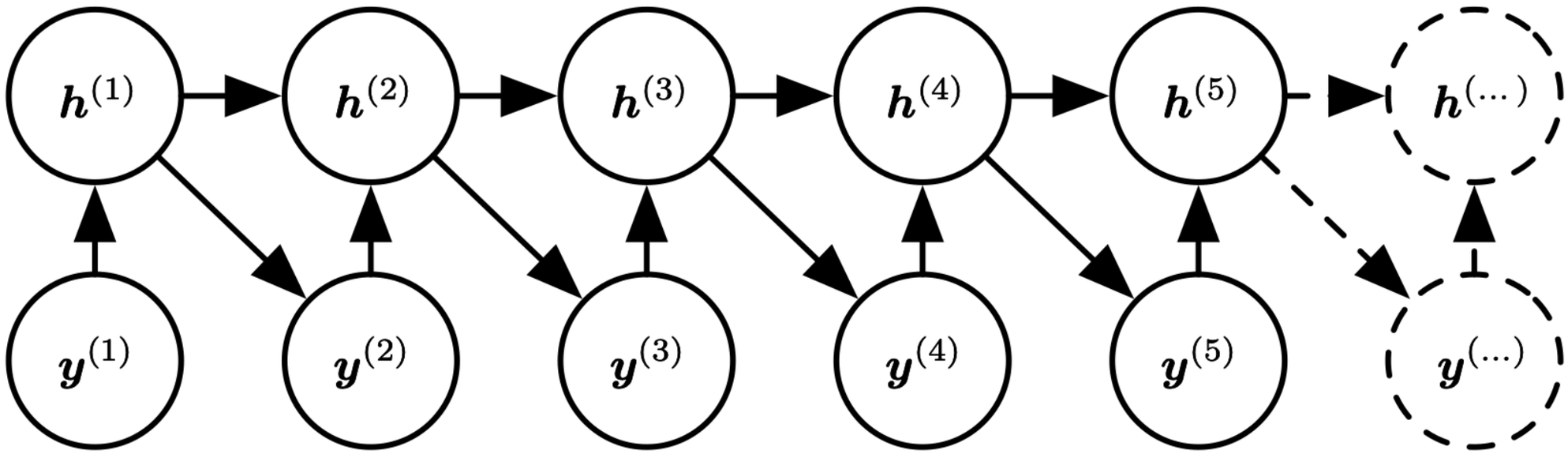


Figure 10.8

(The conditional distributions for the hidden units are deterministic)

RNN assumptions

- The price recurrent networks pay for their reduced number of parameters is that optimizing the parameters may be difficult.
- The parameter sharing used in recurrent networks relies on the assumption that the same parameters can be used for different time steps.
- The assumption is that the conditional probability distribution over the variables at time $t + 1$ given the variables at time t is stationary
- The RNN must have some mechanism for determining the length of the sequence:
  - A special symbol corresponding to the end of a sequence
  - an extra Bernoulli output to the model that represents the decision to either continue generation or halt generation at each time step.
  - add an extra output to the model that predicts the sequence length τ itself, or $\tau - t$, the number of remaining steps.

Vector to Sequence

- An RNN that maps a fixed-length vector x into a distribution over sequences Y.
- This RNN is appropriate for tasks such as image captioning, where a single image is used as input to a model that then produces a sequence of words describing the image.
- Each element $y^{(t)}$ of the observed output sequence serves both as input (for the current time step) and, during training, as target (for the previous time step).

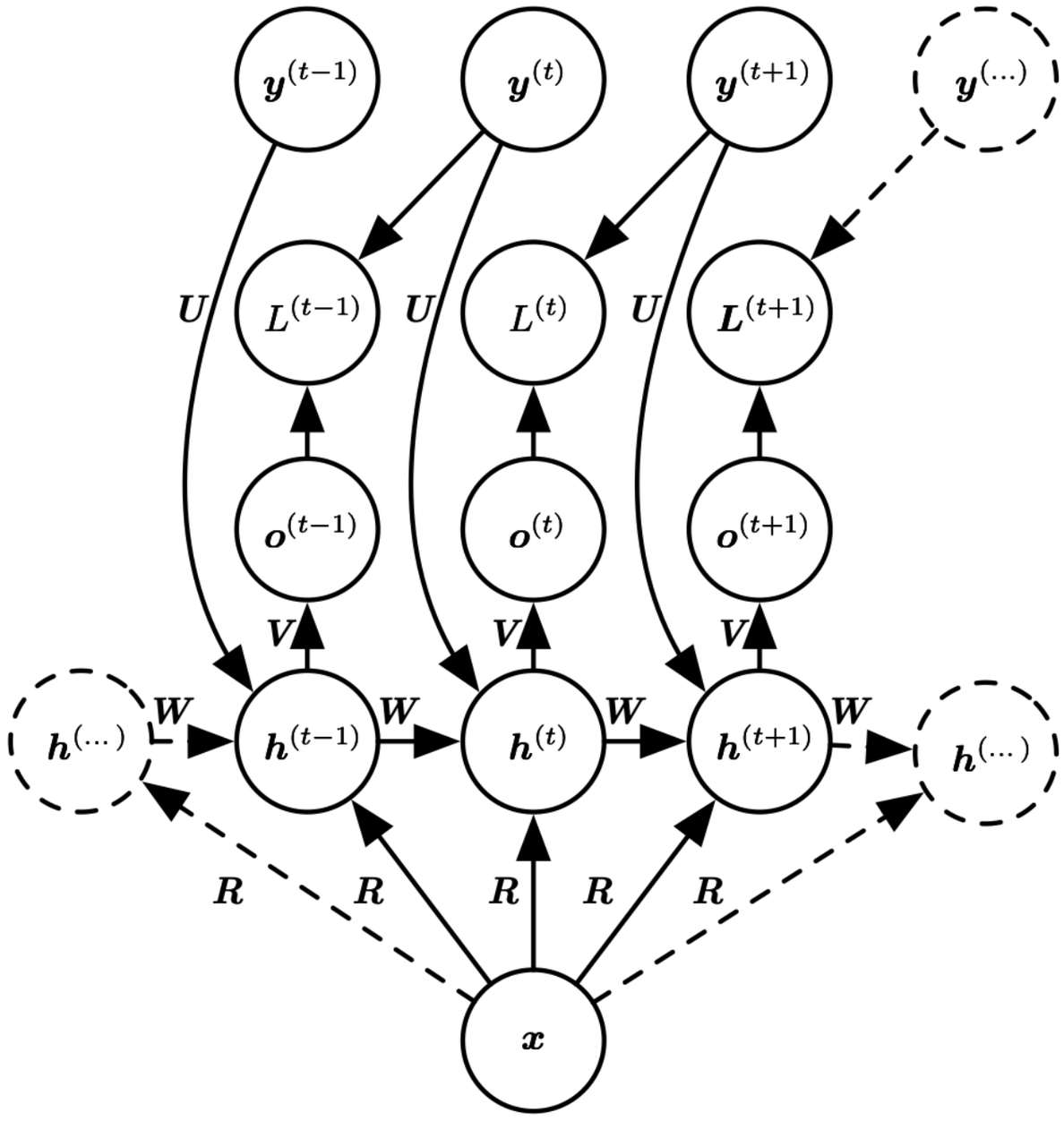


Figure 10.9

Hidden and Output Recurrence

- A conditional recurrent neural network mapping a variable-length sequence of x values into a distribution over sequences of y values of the same length.

- This RNN contains connections from the previous output to the current state.

- These connections allow this RNN to model an arbitrary distribution over sequences of y given sequences of x of the same length.

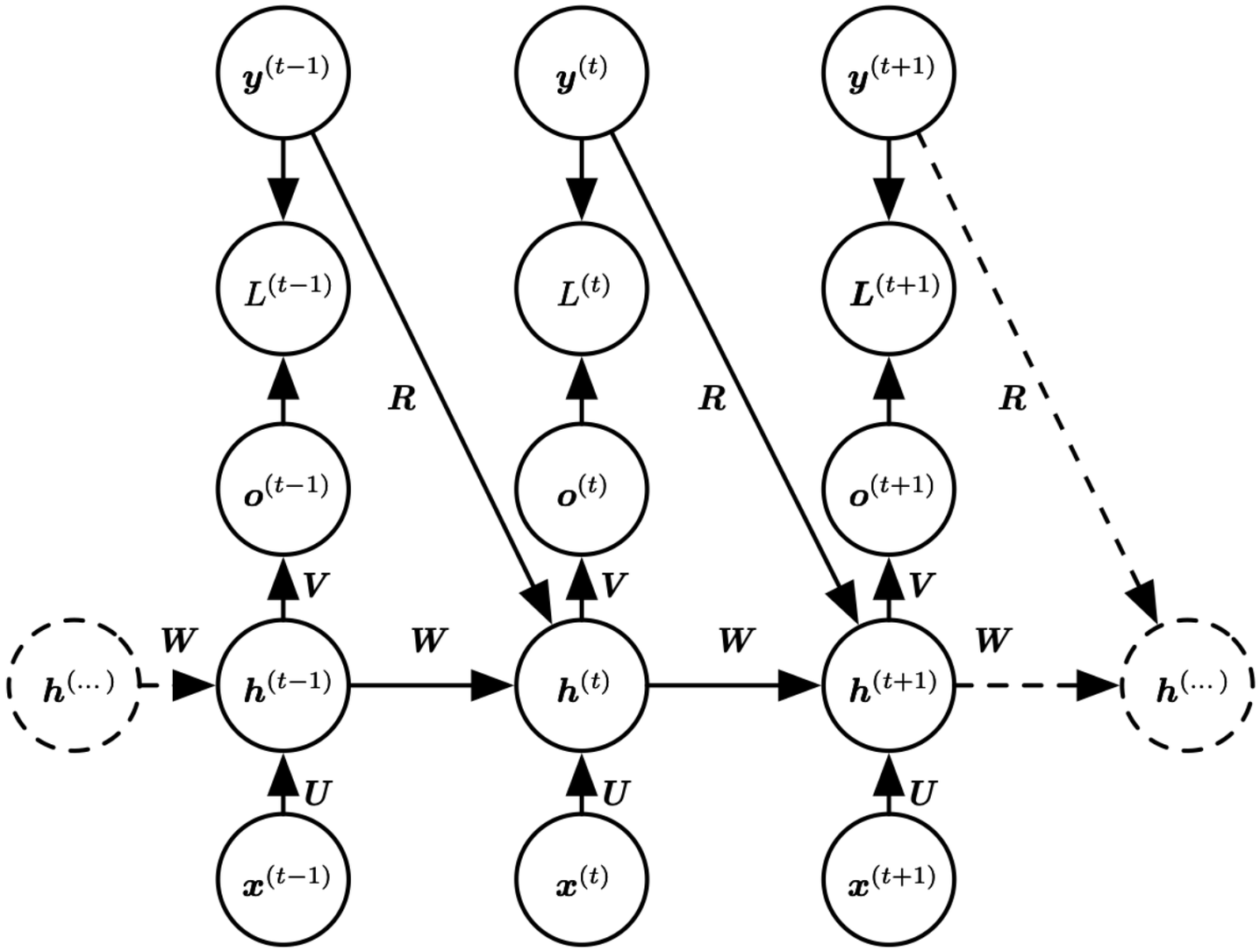


Figure 10.10

The output values are not forced to be conditionally independent in this model.

Bidirectional RNN

- We want to output a prediction of $y^{(t)}$ which may depend on the whole input sequence.
  - For example, in speech recognition, the correct interpretation of the current sound as a phoneme may depend on the next few phonemes because of co-articulation and potentially may even depend on the next few words because of the linguistic dependencies between nearby words:
    - if there are two interpretations of the current word that are both acoustically plausible, we may have to look far into the future (and the past) to disambiguate them.
  - This is also true of handwriting recognition and many other sequence-to-sequence learning tasks

Bidirectional RNN

- Computation of a typical bidirectional recurrent neural network, meant to learn to map input sequences x to target sequences y, with loss $L^{(t)}$ at each step t.
- The h recurrence propagates information forward in time (towards the right) while the g recurrence propagates information backward in time (towards the left).
- Thus at each point t, the output units $o^{(t)}$ can benefit from a relevant summary of the past in its $h^{(t)}$ input and from a relevant summary of the future in its $g^{(t)}$ input.

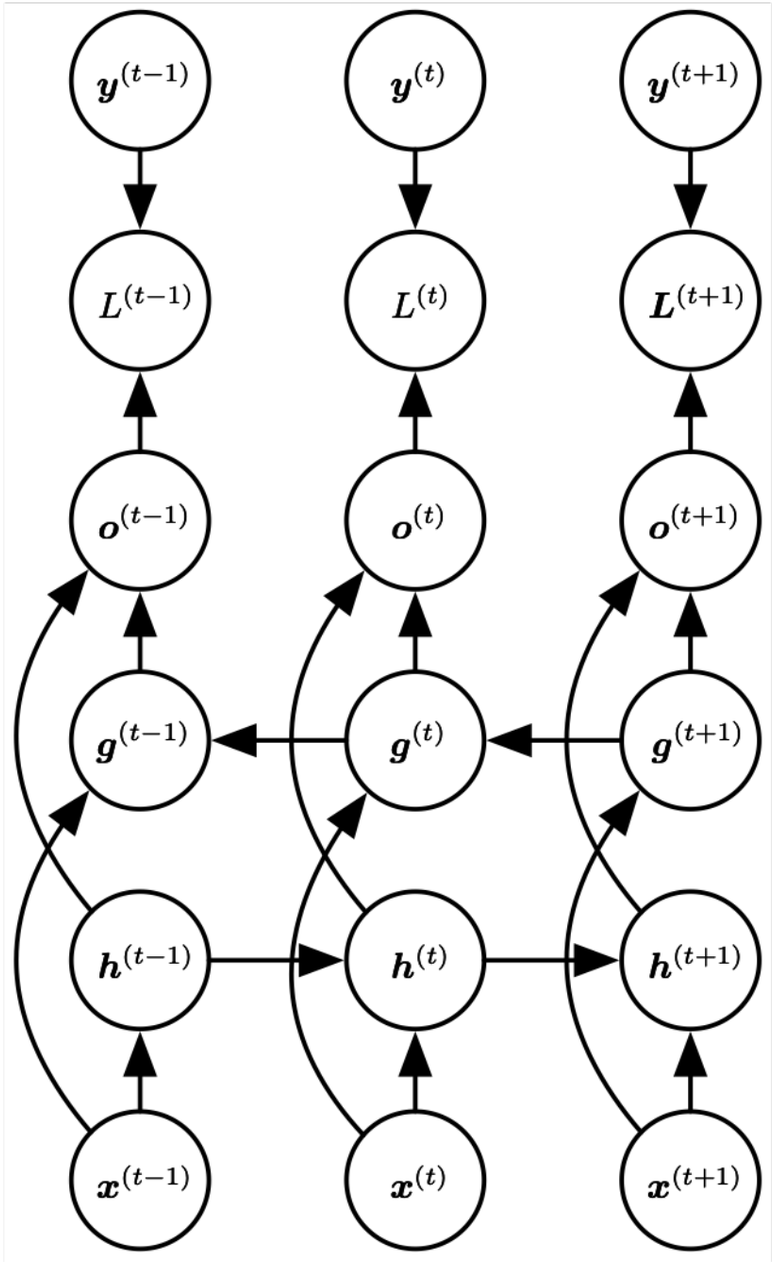


Figure 10.11

Sequence to Sequence Architecture

- RNN can be trained to map an input sequence to an output sequence which is not necessarily of the same length.

- This comes up in many applications, such as speech recognition, machine translation or question answering.

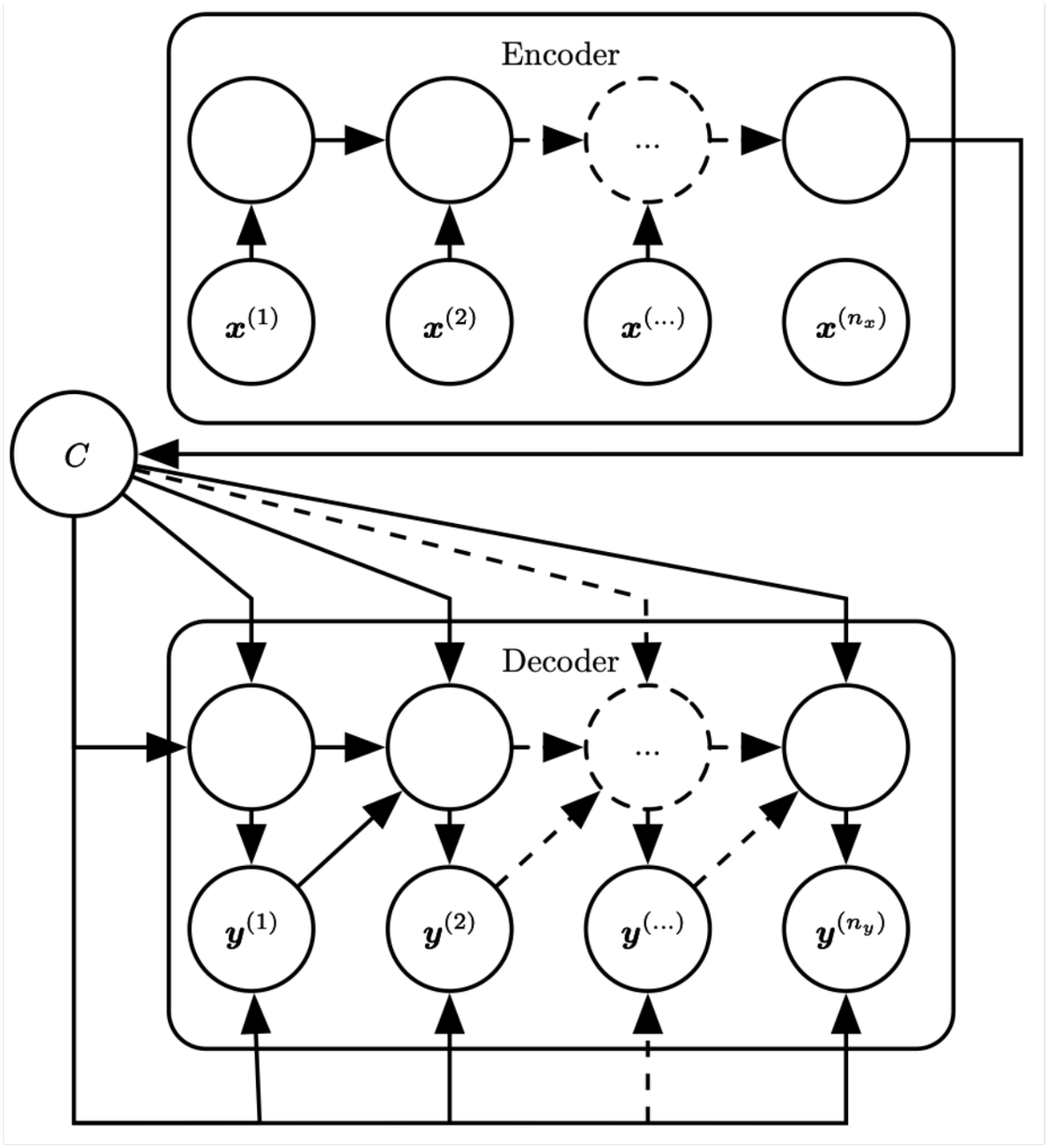


Figure 10.12

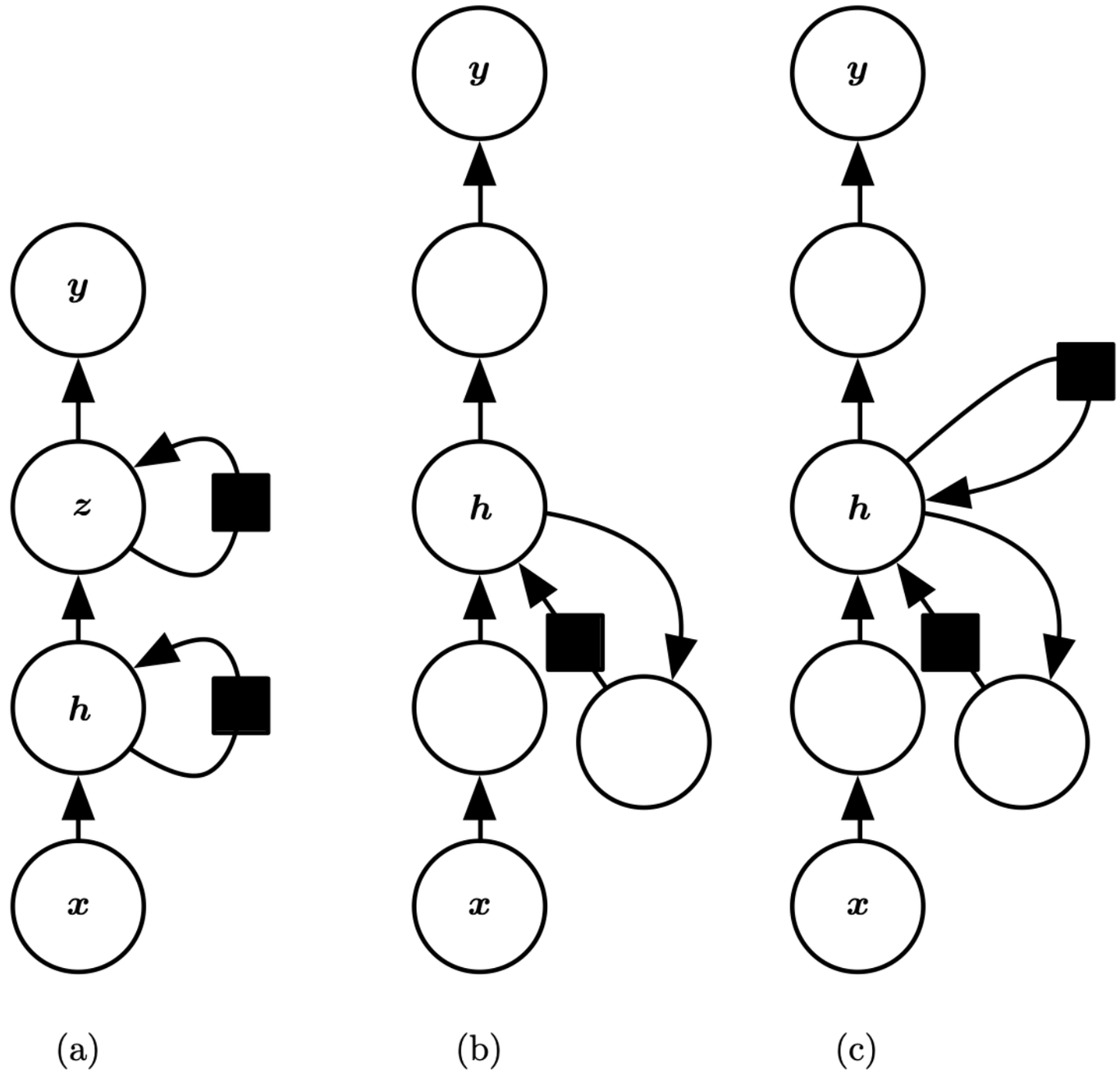
Deep RNNs

A recurrent neural network can be made deep in many ways.

- (a) The hidden recurrent state can be broken down into groups organized hierarchically.

- (b) Deeper computation (e.g., an MLP) can be introduced in the input-to-hidden, hidden-to-hidden and hidden-to-output parts. This may lengthen the shortest path linking different time steps.

- (c)The path-lengthening effect can be mitigated by introducing skip connections.

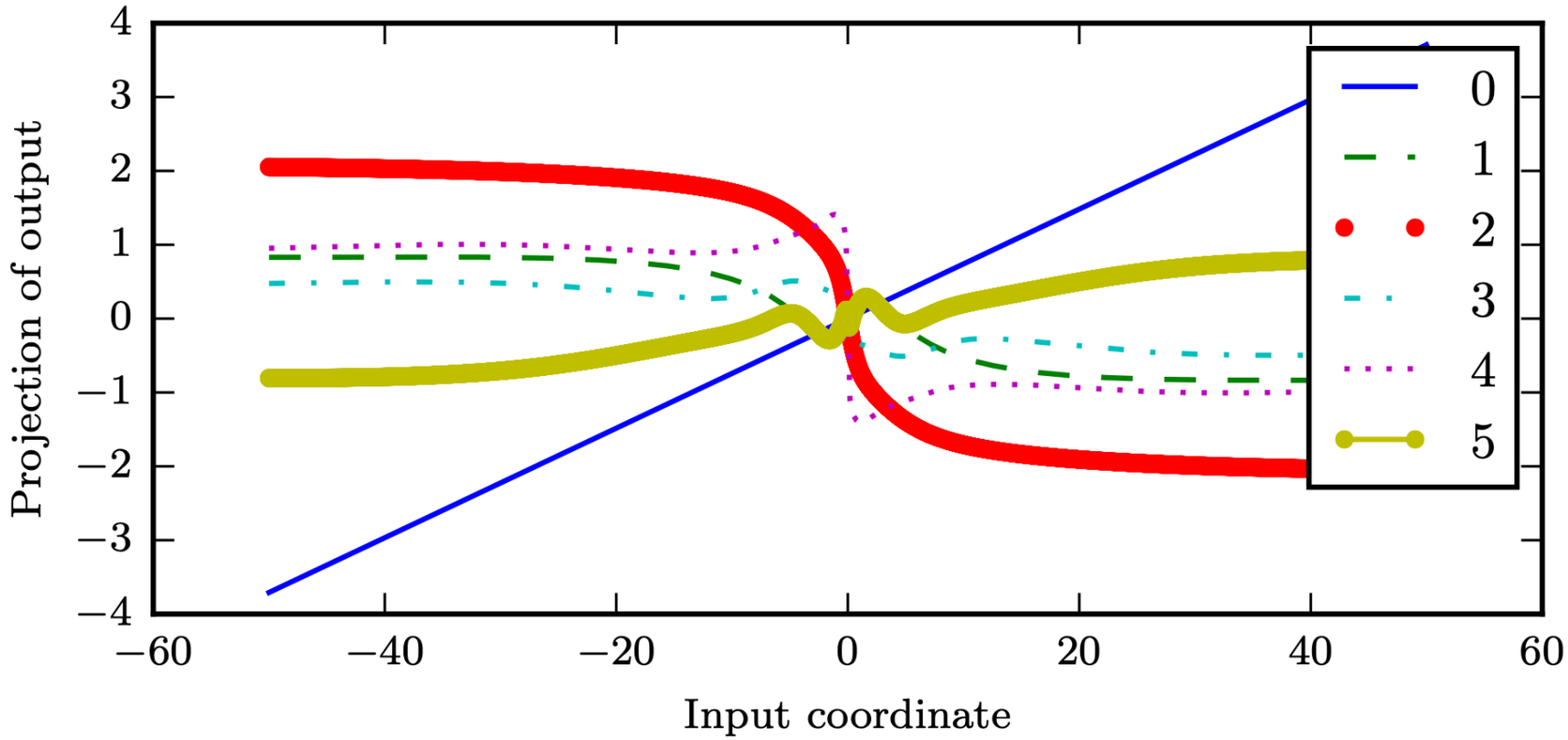


The challenge of long-term dependencies
Repeated Function Composition

- When composing many nonlinear functions (like the linear-tanh layer shown here), the result is highly nonlinear.

- the use of a squashing nonlinearity such as tanh can cause the recurrent dynamics to become bounded.

$$h^{(t)} = W^T h^{(t-1)}$$
$$h^{(t)} = (W^t)^T h^{(0)}$$
$$W = Q\Lambda Q^T$$
$$h^{(t)} = Q^T \Lambda^T Q h^{(0)}$$



- The x-axis is the coordinate of the initial state along a random direction in the 100-dimensional space.
- The y-axis is a linear projection of a 100-dimensional hidden state down to a single dimension
- We can thus view this plot as a linear cross-section of a high-dimensional function.
- The plots show the function after each time step, or equivalently, after each number of times the transition function has been composed.

Gated RNNs

- The most effective sequence models used in practical applications are called gated RNNs.
- These include the long short-term memory (LSTM) and networks based on the gated recurrent unit (GRU)
  - Create paths through time that have derivatives that neither vanish nor explode
  - *Accumulate* information such as evidence for a particular feature or category,
  - *Forget* the old state and start over.

LSTM

"LSTM cells" have an internal recurrence (a self-loop), in addition to the outer recurrence of the RNN

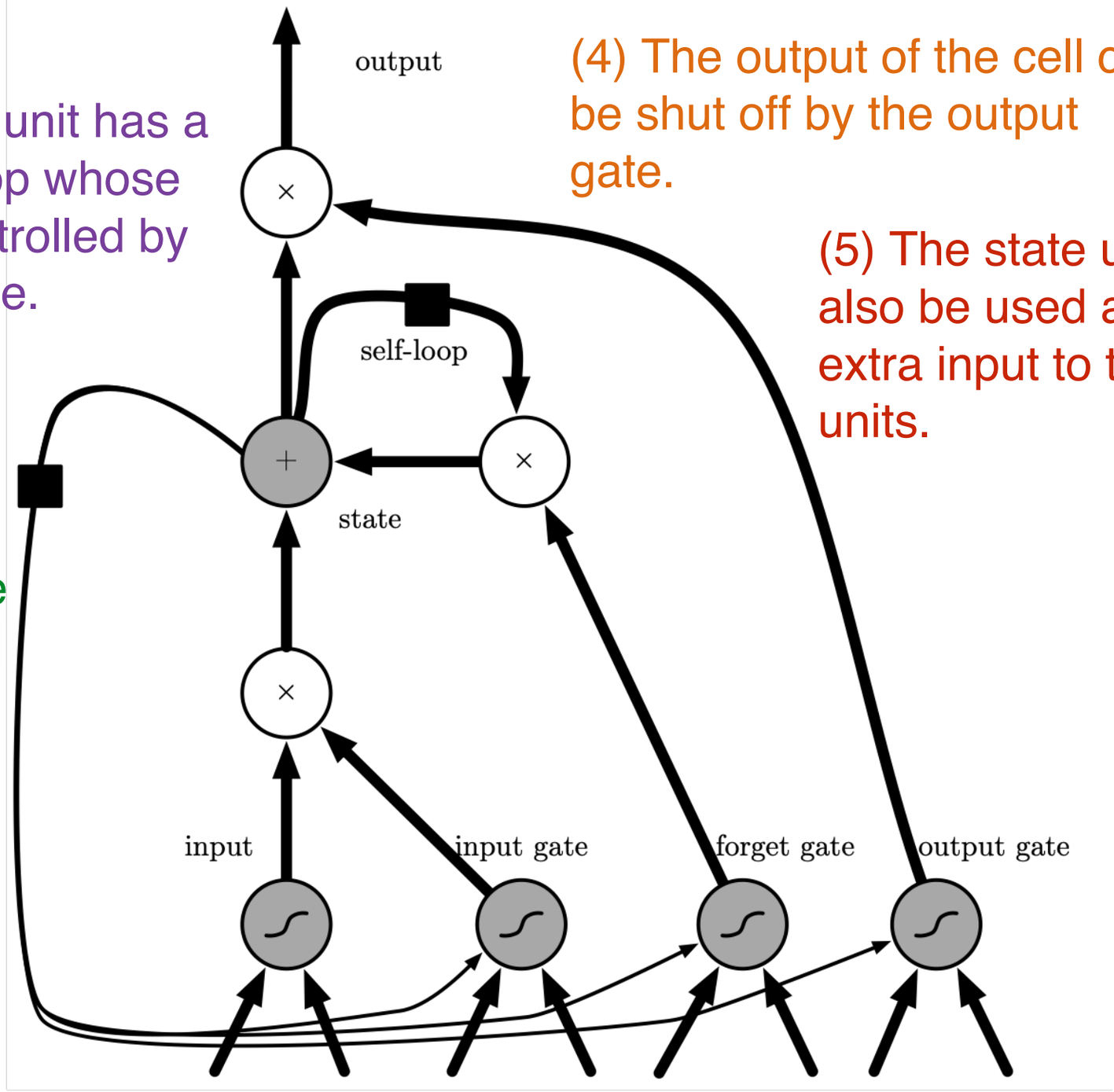
(3) The state unit has a linear self-loop whose weight is controlled by the forget gate.

(4) The output of the cell can be shut off by the output gate.

(5) The state unit can also be used as an extra input to the gating units.

(2) Its value can be accumulated into the state if the sigmoidal input gate allows it.

(1) An input feature is computed with a regular artificial neuron unit



All the gating units have a sigmoid nonlinearity, while the input unit can have any squashing nonlinearity

Figure 10.16

LSTM

Forget gate, Cell i, time step t:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$

State, Cell i, time step t:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right)$$

External input gate, Cell i, time step t:

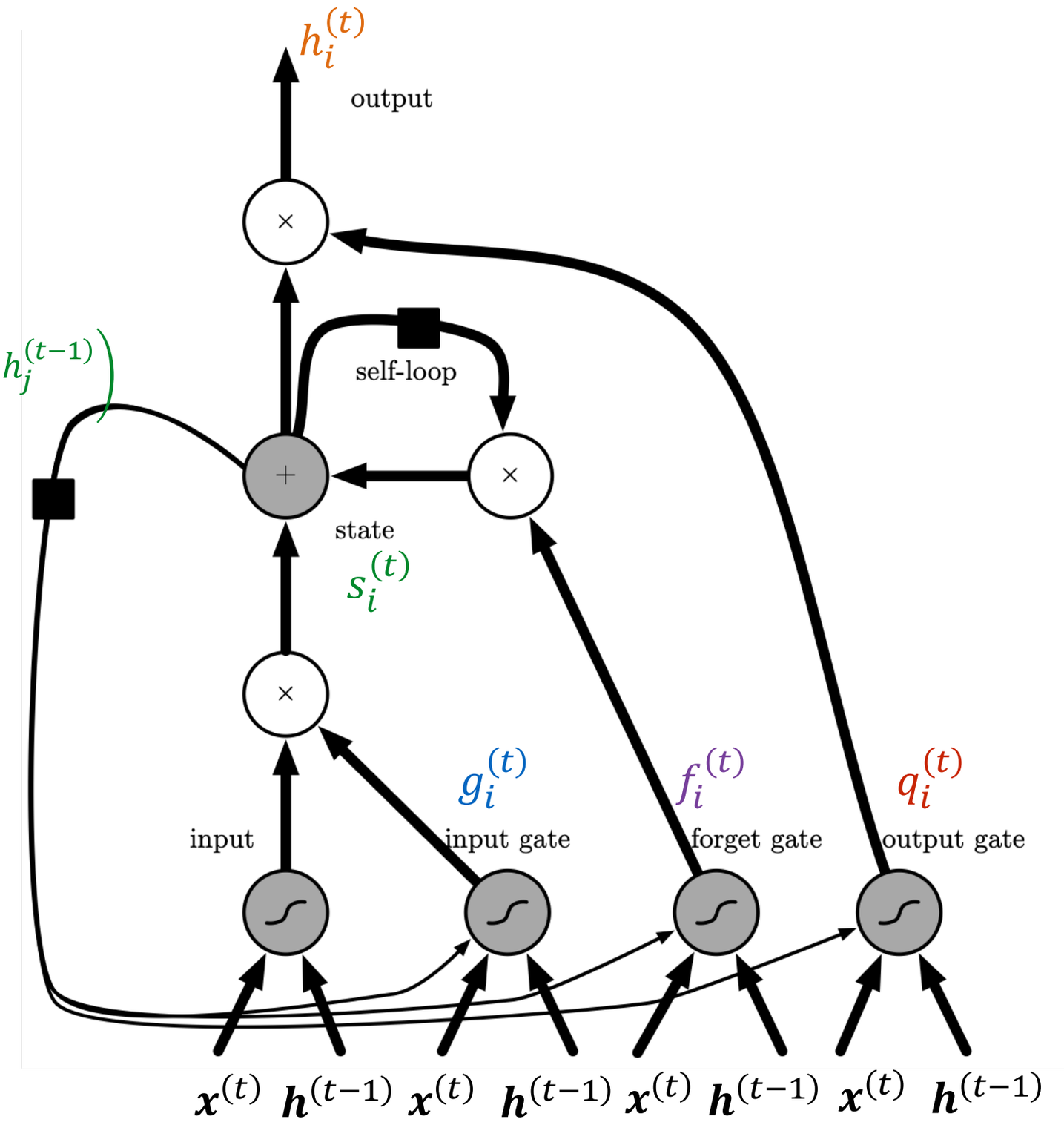
$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right)$$

Output, cell i, time step t:

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}$$

Output gate, cell I, time step t:

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right)$$



LSTM networks

- LSTM networks have been shown to learn long–term dependencies more easily than the simple recurrent architectures,
  - first on artificial data sets designed for testing the ability to learn long–term dependencies
  - then on challenging sequence processing tasks where state–of–the–art performance was obtained

GRU

- The main difference with the LSTM is that a single gating unit simultaneously controls the forgetting factor and the decision to update the state unit.

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + \left(1 - u_i^{(t-1)}\right) \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)}\right)$$

- Update gate:

$$u_i^{(t)} = \sigma\left(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)}\right)$$

- Reset gate:

$$r_i^{(t)} = \sigma\left(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)}\right)$$

Watch

- https://www.youtube.com/watch?v=ZVN14xYm7JA&feature=youtu.be

Farewell

This lecture concludes our CMPS 392

- But it does not conclude your journey with deep learning.

Congratulations