

# Chapter 10

## Sequence Modeling: Recurrent Neural Networks

## Recurrent Neural Networks (RNN)

---

- A family of neural networks specialized for processing sequential data

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(\tau)}$$

with hidden units  $\mathbf{h}^{(t)}$  forming a dynamical system

$$\mathbf{h}^{(t)} = f_{\theta}(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)})$$

- This recurrent relation, when unfolded, leads to

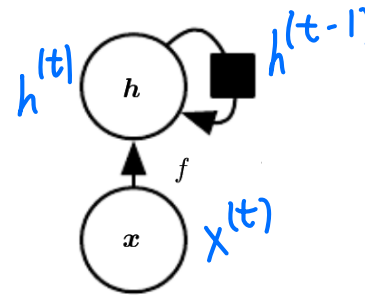
$$\begin{aligned}\mathbf{h}^{(t)} &= f_{\theta}(f_{\theta}(\mathbf{h}^{(t-2)}, \mathbf{x}^{(t-1)}), \mathbf{x}^{(t)}) \\ &= f_{\theta}(f_{\theta}(\dots f_{\theta}(\mathbf{h}^{(0)}, \mathbf{x}^{(1)}), \mathbf{x}^{(2)}), \dots, \mathbf{x}^{(t-1)}), \mathbf{x}^{(t)}) \\ &= g^{(t)}(\mathbf{h}^{(0)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)})\end{aligned}$$

- $\mathbf{h}^{(t)}$  summarizes the past inputs up to  $t$  with a fixed length vector

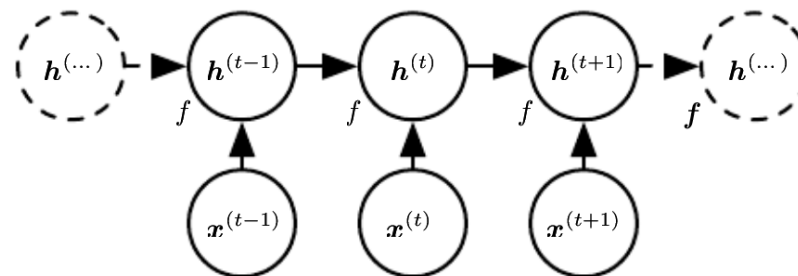
- The RNN is to learn a single shared model  $f_{\theta}$  instead of separate models  $g^{(t)}$  at different time steps
- This enables generalization to any sequence length, and requires far fewer parameters
- ★ However, it also limits the model capacity; for example, the predicted relationship between the previous time step and a current time step is independent of  $t$  if  $h^{(t-1)} = h^{(t'-1)}$  and  $x^{(t)} = x^{(t')}$  then  $f(h^{(t-1)}, x^{(t)}) = f(h^{(t'-1)}, x^{(t')})$
- Theoretically, when wired properly, the RNN is able to simulate procedures achievable by a Turing machine

## Circuit Diagrams and Graph Unrolling

- **Circuit diagrams** – a succinct description of computations with nodes representing components <sup>簡明的</sup> that might exist in physical implementations

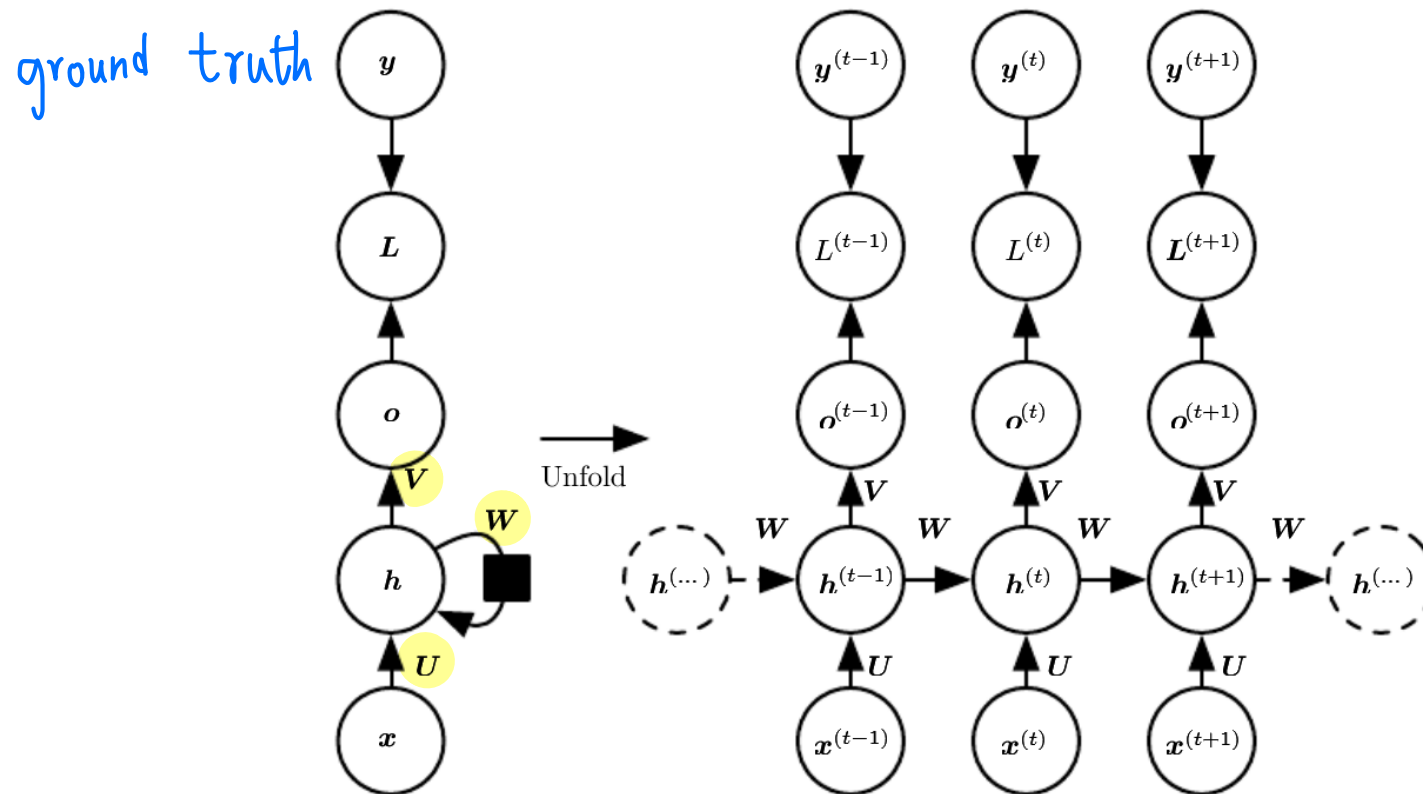


- **Unfolded computational graphs** – an explicit description of computations with nodes indicating variables at each time step



## Design Pattern I

- An output at each time step with recurrent hidden unit connections



- Essentially, it implements a sequence-to-sequence mapping

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(\tau)} \rightarrow \mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(\tau)}$$

- Forward propagation (with initial  $\mathbf{h}^{(0)}$ )

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)},$$


$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}),$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)},$$

where  $\mathbf{U}, \mathbf{W}, \mathbf{V}$  correspond respectively to connections for

- Input-to-hidden units ( $\mathbf{U}$ )
- Hidden-to-hidden units ( $\mathbf{W}$ )
- Hidden-to-output units ( $\mathbf{V}$ )

and  $\mathbf{b}, \mathbf{c}$  are biases

- Why use tanh instead of sigmoid for activation? 

- When the target  $\mathbf{y}^{(t)}$  is a **discrete** (multinoulli) variable, a **softmax** is applied to  $\mathbf{o}^{(t)}$  to obtain

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

- This  $\hat{\mathbf{y}}^{(t)}$  serves as the model prediction for the empirical conditional probability of  $\mathbf{y}^{(t)}$  given  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}$

$$p_{\text{model}}(\mathbf{y}^{(t)} | \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}) = \prod_i \left( \hat{y}_i^{(t)} \right)^{\mathbf{1}(y_i^{(t)}=1)}$$

- During training, the total loss is the sum of losses over all time steps

$$L(\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}\}, \{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(t)}\}) = \sum_t L^{(t)}$$

where the **loss**  $L^{(t)}$  at time step  $t$  is given by

$$L^{(t)} = -\log p_{\text{model}}(\mathbf{y}^{(t)} | \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)})$$

- This network architecture corresponds to a conditional distribution  $p_{\text{model}}(\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(\tau)} | \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(\tau)})$  that factorizes as

$$\begin{aligned}
 & p_{\text{model}}(\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(\tau)} | \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(\tau)}) \\
 &= \prod_t p_{\text{model}}(\mathbf{y}^{(t)} | \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}) \quad \gamma_{\leq t} \\
 &= \prod_t p_{\text{model}}(\mathbf{y}^{(t)} | \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(\tau)}) \\
 & \quad \mathbf{x}^{(t+1)} \sim \mathbf{x}^{(\tau)} \text{ 不影響}
 \end{aligned}$$

- This suggests that  $\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(\tau)}$  are modeled to be **conditionally independent** given  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(\tau)}$

$$p(x_1, x_2, x_3) = p(x_1) p(x_2 | x_1) \cdot p(x_3 | x_2, x_1)$$



## Back-Propagation Through Time (BPTT)

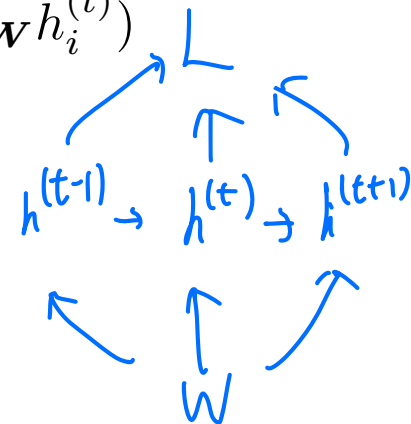
- BPTT is merely **back-propagation applied to unrolled graphs**
- To obtain the gradient on parameter nodes, the gradient on their immediate child (downstream) nodes have to be evaluated first
- As an example, to compute  $\nabla_{\mathbf{W}} L$ , we observe that
  - The immediate child nodes of  $\mathbf{W}$  are all  $\mathbf{h}^{(t)}$ 's, and
  - The chain rule for tensors<sup>a</sup> can be applied to arrive at

$$\nabla_{\mathbf{W}} L = \sum_t \sum_i \left( \frac{\partial L}{\partial h_i^{(t)}} \right) (\nabla_{\mathbf{W}} h_i^{(t)})$$

<sup>a</sup>

$$\mathbf{X}_{m \times n} \xrightarrow{g(\mathbf{X})} \mathbf{Y}_{s \times k} \xrightarrow{f(\mathbf{Y})} z_{1 \times 1}$$

$$\nabla_{\mathbf{X}} z = \sum_j \left( \frac{\partial z}{\partial Y_j} \right) \nabla_{\mathbf{X}} Y_j,$$



- To complete the evaluation, we need to know further  $\nabla_{\mathbf{h}^{(t)}} L$ , which can be evaluated using the same chain rule as

$$\begin{aligned}\nabla_{\mathbf{h}^{(t)}} L &= \left( \frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}} \right)^T (\nabla_{\mathbf{h}^{(t+1)}} L) + \left( \frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}} \right)^T (\nabla_{\mathbf{o}^{(t)}} L) \\ &= \mathbf{W}^T \mathbf{H}^{(t+1)} (\nabla_{\mathbf{h}^{(t+1)}} L) + \mathbf{V}^T (\nabla_{\mathbf{o}^{(t)}} L)\end{aligned}$$

where

$$\begin{aligned}\mathbf{H}^{(t+1)} &= \left( \frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{a}^{(t+1)}} \right)^T \\ &= \begin{bmatrix} 1 - (h_1^{(t+1)})^2 & 0 & \dots & 0 \\ 0 & 1 - (h_2^{(t+1)})^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 - (h_n^{(t+1)})^2 \end{bmatrix} \\ \nabla_{\mathbf{o}^{(t)}} L &= \hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}\end{aligned}$$

and

$$\nabla_{\mathbf{h}^{(\tau)}} L = \mathbf{V}^T (\nabla_{\mathbf{o}^{(\tau)}} L) = \mathbf{V}^T (\hat{\mathbf{y}}^{(\tau)} - \mathbf{y}^{(\tau)})$$

- In matrix form,  $\nabla_{\mathbf{W}} L$  is given as

$$\nabla_{\mathbf{W}} L = \sum_t \mathbf{H}^{(t)} (\nabla_{\mathbf{h}^{(t)}} L) \mathbf{h}^{(t-1)T}$$

- The gradient on the remaining parameters can be obtained similarly

$$\nabla_{\mathbf{U}} L = \sum_t \mathbf{H}^{(t)} (\nabla_{\mathbf{h}^{(t)}} L) \mathbf{x}^{(t)T}$$

$$\nabla_{\mathbf{V}} L = \sum_t (\nabla_{\mathbf{o}^{(t)}} L) \mathbf{h}^{(t)T}$$

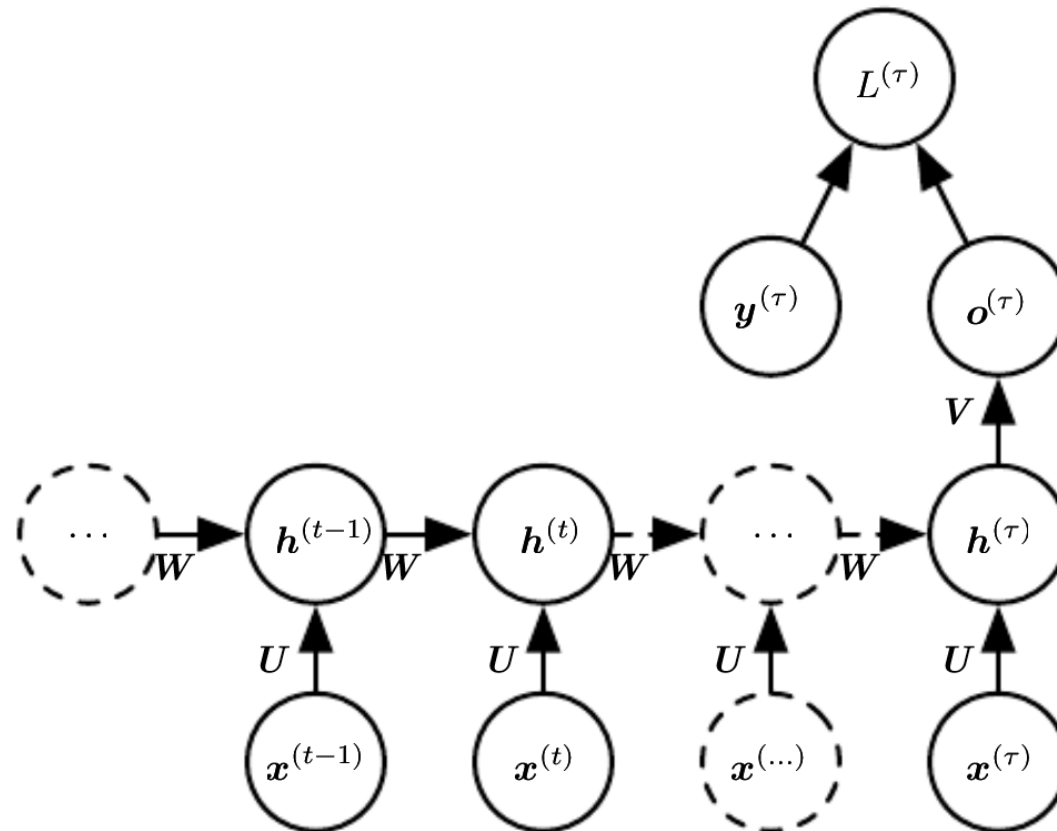
$$\nabla_{\mathbf{b}} L = \sum_t \mathbf{H}^{(t)} (\nabla_{\mathbf{h}^{(t)}} L)$$

$$\nabla_{\mathbf{c}} L = \sum_t \nabla_{\mathbf{o}^{(t)}} L$$

- The runtime and memory cost for BPTT are both  $\mathcal{O}(\tau)$

## Design Pattern II

- Networks with hidden unit connections that produce a single output for an entire sequence



## Vanishing and Exploding Gradients

- Gradients propagate over many stages tend to either vanish or explode
- For example, the gradient  $\nabla_{\mathbf{h}^{(t)}} L$  in pattern II is seen to follow

$$\begin{aligned}\nabla_{\mathbf{h}^{(t)}} L &= \mathbf{W}^T \mathbf{H}^{(t+1)} (\nabla_{\mathbf{h}^{(t+1)}} L) \\ &= \underbrace{(\mathbf{W}^T \mathbf{H}^{(t+1)})}_{\mathbf{M}}^{(\tau-t)} (\nabla_{\mathbf{h}^{(\tau)}} L) \\ &= \mathbf{Q} \mathbf{\Lambda}^{(\tau-t)} \mathbf{Q}^T (\nabla_{\mathbf{h}^{(\tau)}} L)\end{aligned}$$

where  $\mathbf{M}$  is assumed to be the same at every time step (as is the case without activation) and have an eigendecomposition  $\mathbf{M} = \mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^T$

- The term  $\mathbf{\Lambda}^{(\tau-t)}$  causes the eigenvalues with magnitude smaller than one to decay to zero and eigenvalues with magnitude greater than one to explode (when  $t \ll \tau$ )

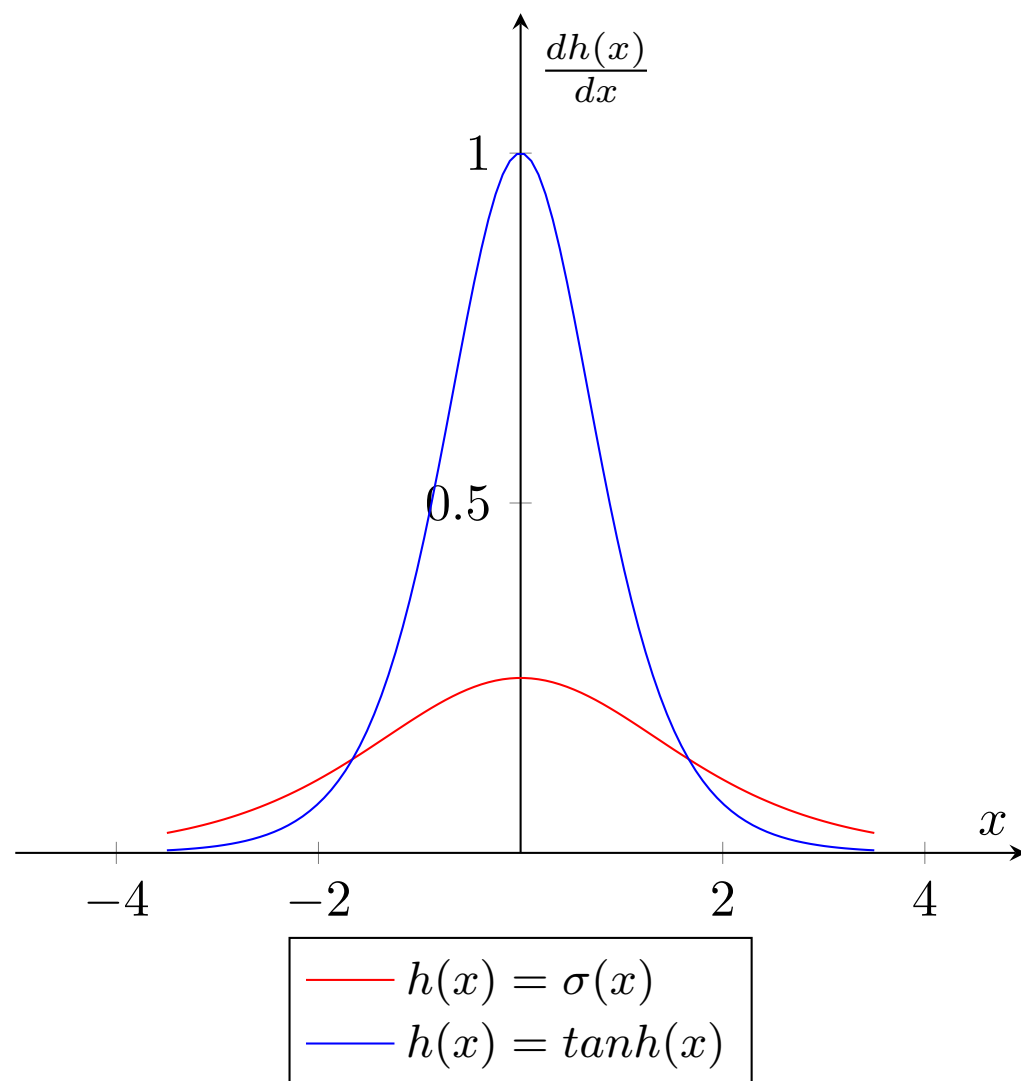
- When gradients vanish quickly, it becomes difficult to learn long-term dependencies; as an example, recall that

$$\nabla_{\mathbf{W}} L = \sum_t \mathbf{H}^{(t)} (\nabla_{\mathbf{h}^{(t)}} L) \mathbf{h}^{(t-1)T}$$

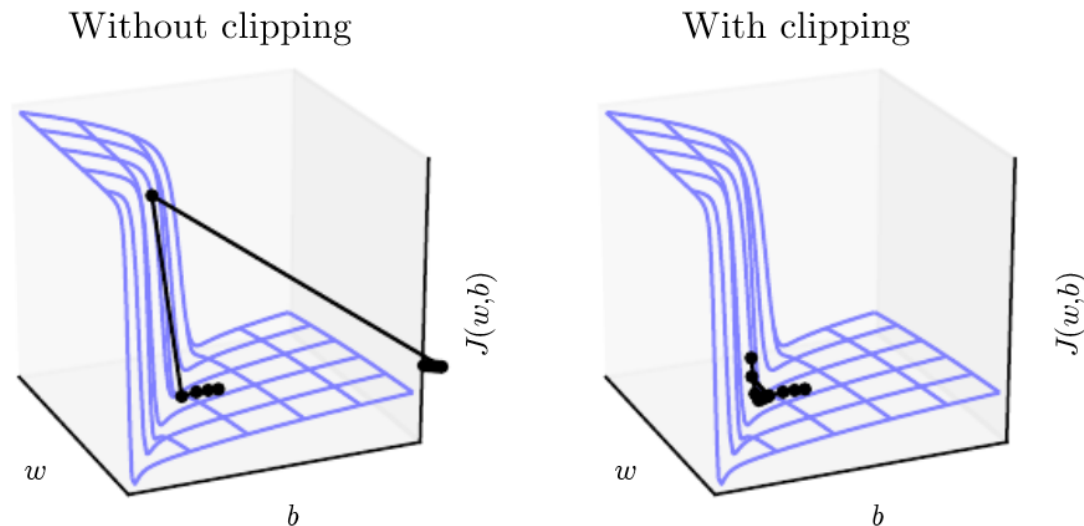
$$\nabla_{\mathbf{U}} L = \sum_t \mathbf{H}^{(t)} (\nabla_{\mathbf{h}^{(t)}} L) \mathbf{x}^{(t)T}$$

where the distant past inputs  $\mathbf{x}^{(t)}$  and hidden units  $\mathbf{h}^{(t)}$  with  $t \ll \tau$  are seen to contribute little to learning  $\mathbf{W}$  and  $\mathbf{U}$

- From the perspective of forward propagation, an input  $\mathbf{x}^{(t)}$ ,  $t \ll \tau$  may have been attenuated significantly before it reaches the output  $\mathbf{o}^{(\tau)}$  due to the repeated multiplication with  $\mathbf{HW}$  纖細的, 消滅的
- **Sidetracking**: The reason for choosing tanh as activation has to do with  $\mathbf{H}$ , of which the diagonal entries are derivatives of  $h(x)$ ; gradients may vanish much quicker if sigmoid is used 使轉移話題



- When gradients explode, the gradient-based training could throw the parameters far into a region where the objective becomes larger



- One simple solution is to clip the gradient before the parameter update when a threshold on its norm is exceeded; that is, if  $\|g\| > v$

$$g \leftarrow \frac{gv}{\|g\|}$$



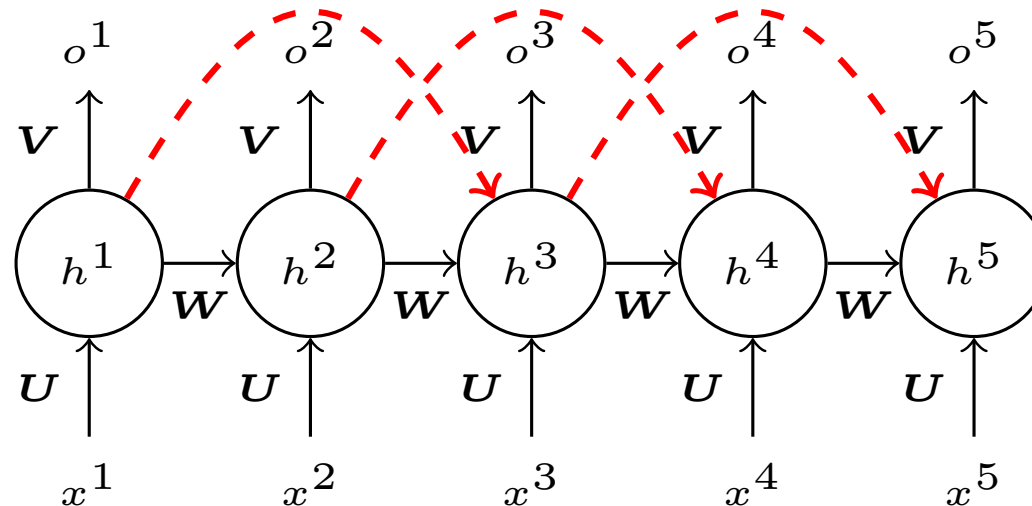
- Note that while the activation function  $\tanh$  helps to ensure forward propagation has bounded dynamics (i.e. hidden units will not explode), it is possible for backward propagation to remain unbounded (recall that  $\mathbf{H}$  can be  $\mathbf{I}$ )

## Learning Long-Term Dependencies

- **Echo state networks**

- Set recurrent weights  $W$  to ensure spectral radius of Jacobian  $\approx 1$
- Learn output weights  $V$  only

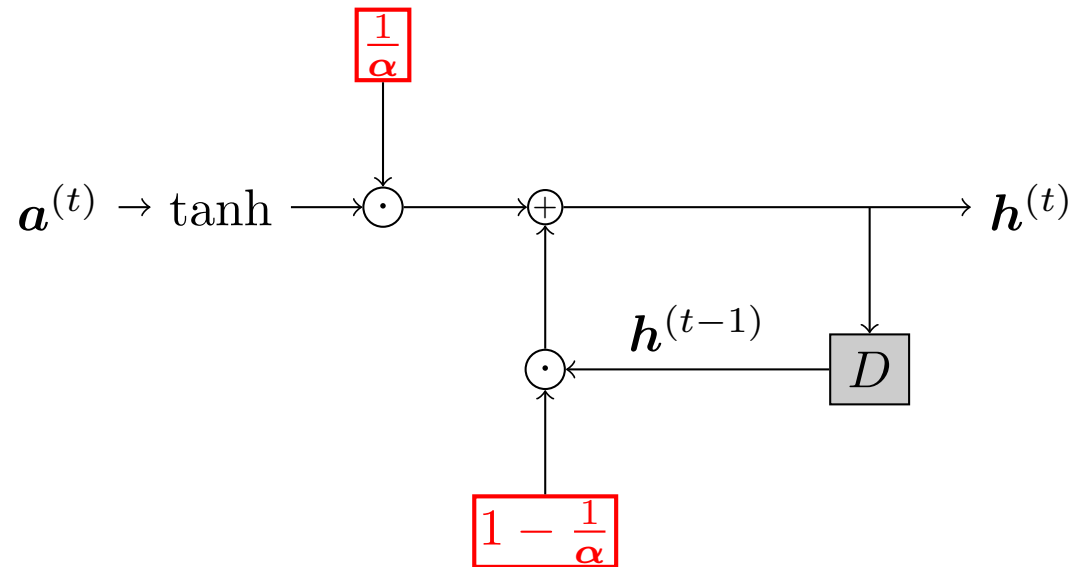
- **Skip connections** – connections from variables in the distant past



Gradients diminish exponentially as a function of  $\tau/d$

- **Leaky units** – introducing an integrator in the hidden unit

$$\mathbf{h}^{(t)} = \left(1 - \frac{1}{\alpha}\right) \odot \mathbf{h}^{(t-1)} + \frac{1}{\alpha} \odot \tanh(\mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}), \quad 1 \leq \alpha_i < \infty$$



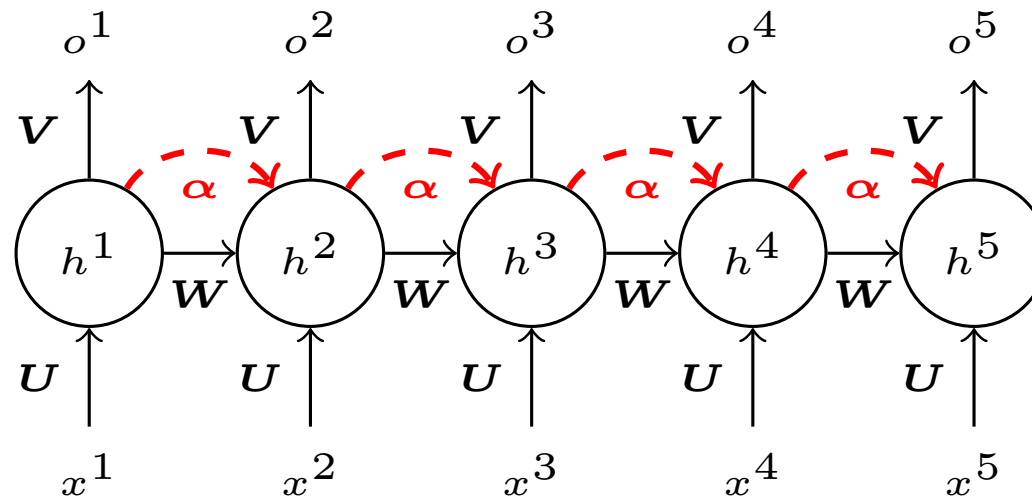
- The new content  $\mathbf{a}^{(t)}$  is given by

$$\mathbf{a}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}$$

- The conventional RNN does not have this extra inner loop

$$\mathbf{a}^{(t)} \rightarrow \tanh \longrightarrow \mathbf{h}^{(t)}$$

- $\alpha$  determines the **time scale** of integration ( $\alpha_i \uparrow$ , **scale**  $\uparrow$ )
  - $\alpha_i = 1$  : Ordinary RNN
  - $\alpha_i > 1$  :  $\mathbf{x}^{(t)}$ 's flow longer; gradients propagate more easily

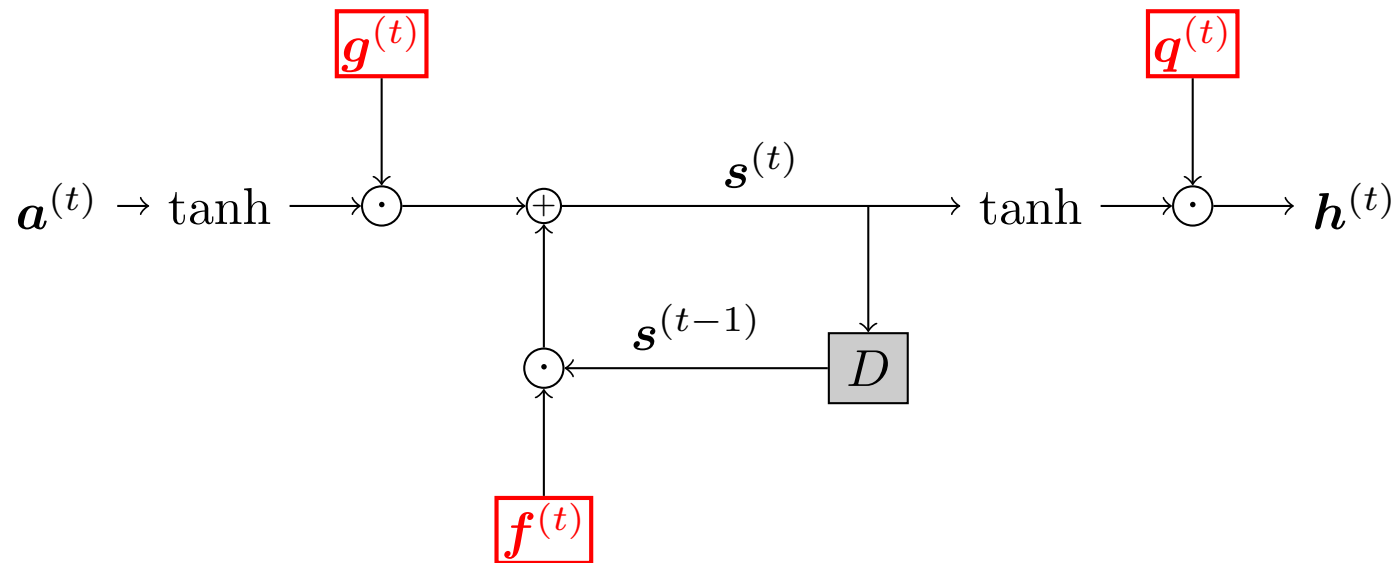


- The Jacobian  $(\partial \mathbf{h}^{(t+1)} / \partial \mathbf{h}^{(t)})^T$  approximates a diagonally dominant matrix when  $\alpha_i$ 's are large enough, suggesting gradients can back propagate more easily

$$\nabla_{\mathbf{h}^{(t)}} L = \left( \frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}} \right)^T (\nabla_{\mathbf{h}^{(t+1)}} L)$$

## Long Short-Term Memory (LSTM)

- To change the time scale of integration dynamically by introducing programable gates ( $g^{(t)}$ ,  $f^{(t)}$ ,  $q^{(t)}$ ) that are conditioned on context

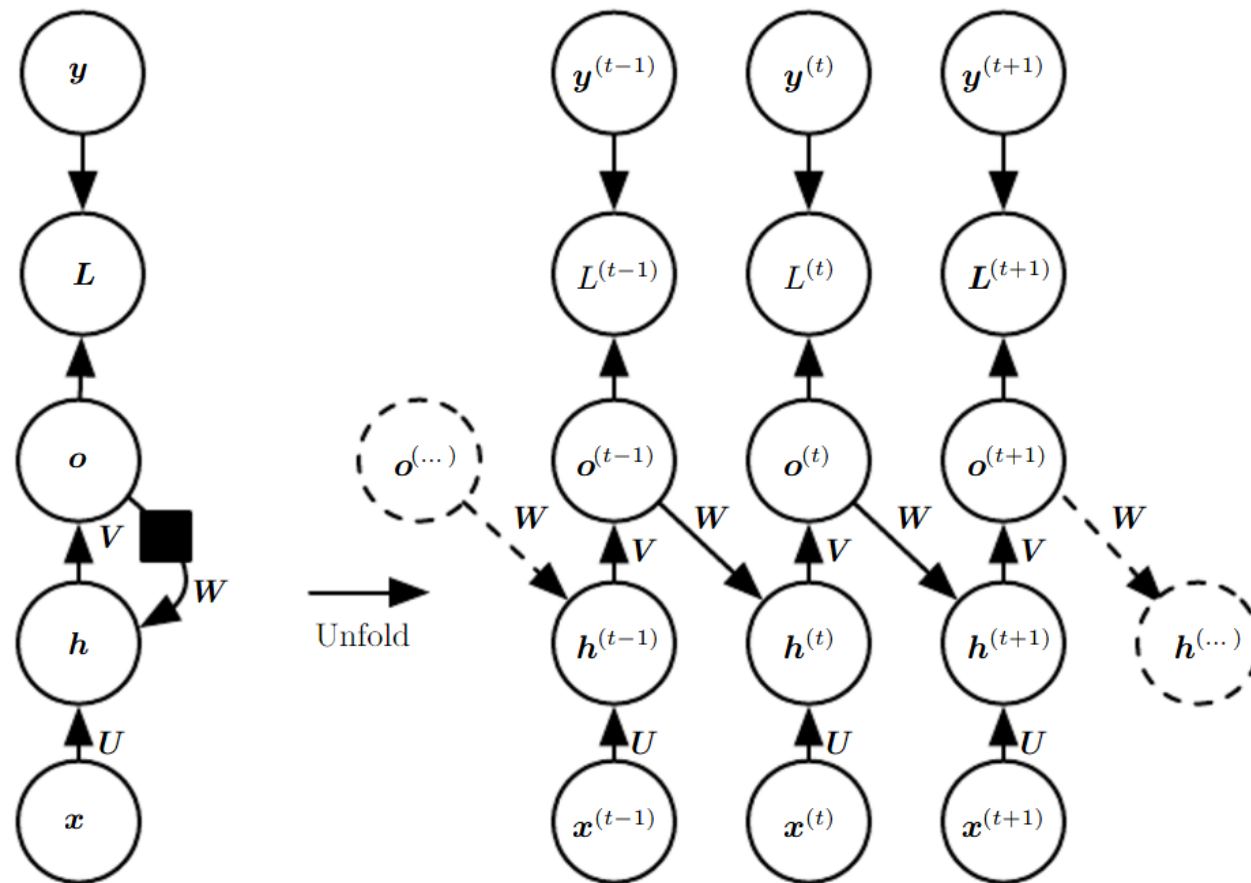


- The gating context at time  $t$  refers collectively to  $\{x^{(t)}, h^{(t-1)}\}$  and may include other inputs

- Memory state:  $\mathbf{s}^{(t)}$
  - Input gate:  $\mathbf{g}^{(t)} = \sigma(\mathbf{U}^g \mathbf{x}^{(t)} + \mathbf{W}^g \mathbf{h}^{(t-1)})$
  - Output gate:  $\mathbf{q}^{(t)} = \sigma(\mathbf{U}^o \mathbf{x}^{(t)} + \mathbf{W}^o \mathbf{h}^{(t-1)})$
  - Forget gate:  $\mathbf{f}^{(t)} = \sigma(\mathbf{U}^f \mathbf{x}^{(t)} + \mathbf{W}^f \mathbf{h}^{(t-1)})$
  - New content:  $\mathbf{a}^{(t)} = \mathbf{U} \mathbf{x}^{(t)} + \mathbf{W} \mathbf{h}^{(t-1)}$
  - Memory update:  $\mathbf{s}^{(t)} = \mathbf{f}^{(t)} \odot \mathbf{s}^{(t-1)} + \mathbf{g}^{(t)} \odot \tanh(\mathbf{a}^{(t)})$
  - Hidden unit update:  $\mathbf{h}^{(t)} = \mathbf{q}^{(t)} \odot \tanh(\mathbf{s}^{(t)})$
  - Output unit update:  $\mathbf{o}^{(t)} = \mathbf{V} \mathbf{h}^{(t)}$
- The design of LSTM is justified by the fact that there could be subsequences of varying statistics in a main sequence
  - Many variants of LSTM are available (study by yourself)

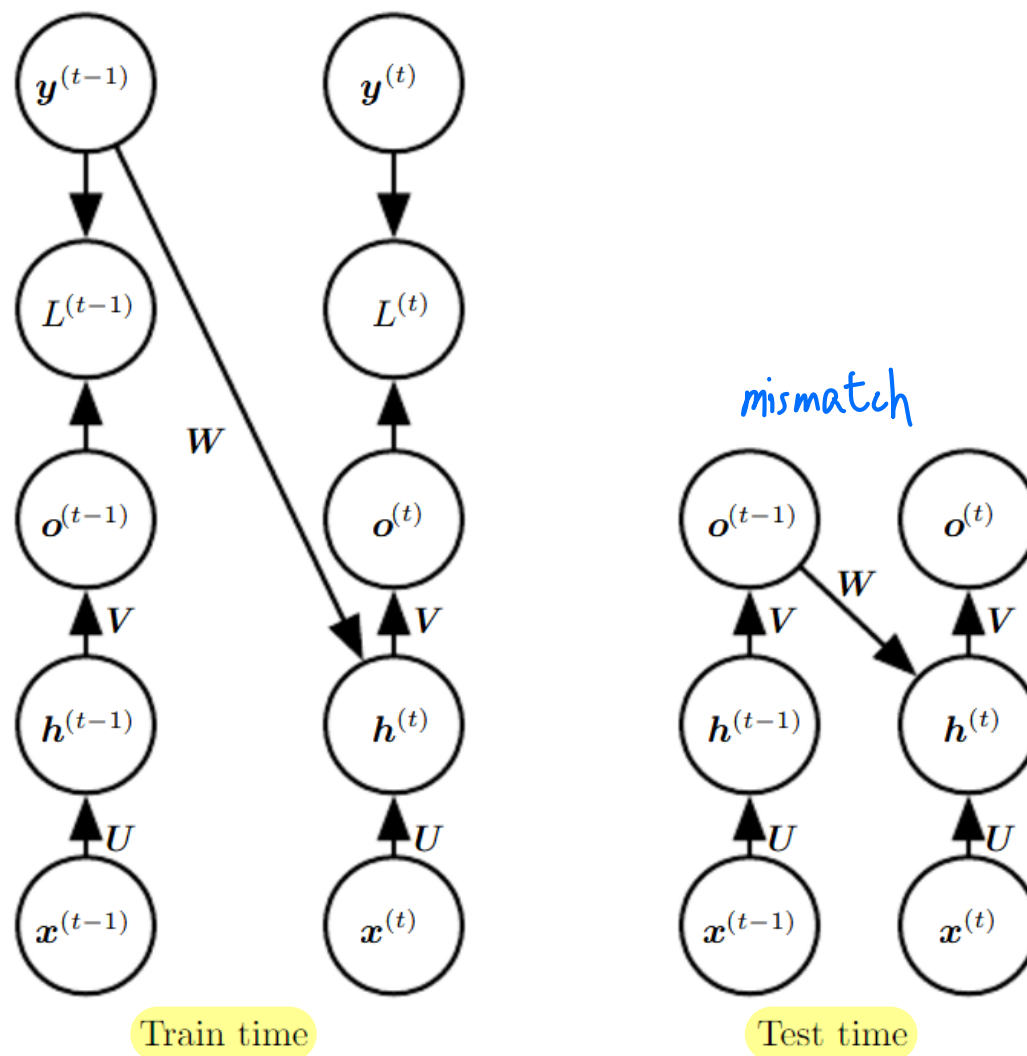
## Design Pattern III

- Networks with only **output recurrence**





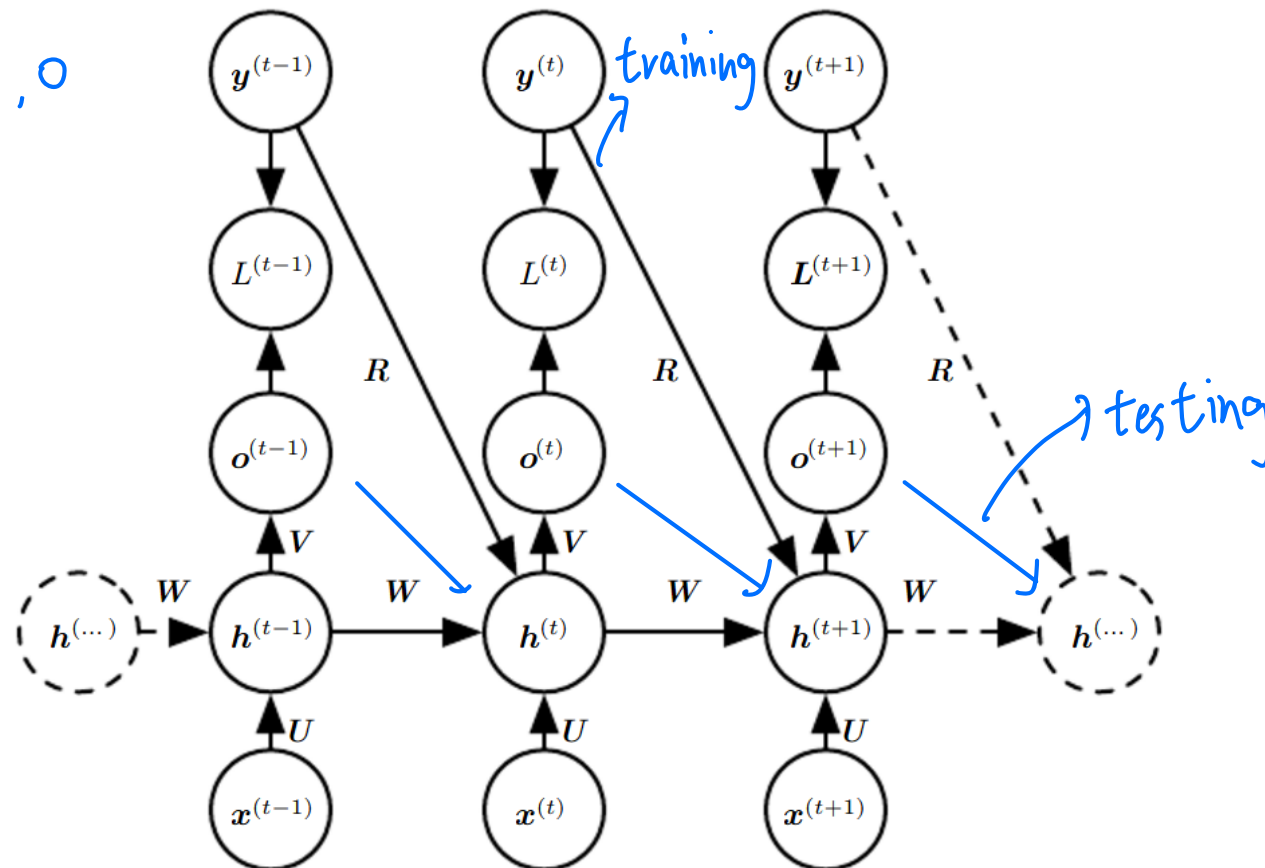
- Such networks are less powerful as the outputs  $\mathbf{o}^{(t)}$ 's are forced to approximate the targets  $\mathbf{y}^{(t)}$ 's while having to convey a good summary about the past
- They however shed lights on training networks with output recurrence
- One effective way for training this type of networks is **teacher forcing**, which feeds correct targets  $\mathbf{y}^{(t)}$  into  $\mathbf{h}^{(t+1)}$  during training, allowing the gradient for each time step to be computed in isolation
- **Open-loop issue**: inputs seen at training and test time are different (mismatch)
- To resolve this, it is common to train with both teacher-forced inputs and free-running inputs (generated by the output-to-input paths), or randomly choose between them



## Design Pattern IV

- Networks with both output recurrence and hidden unit connections

混合使用  $y, o$



- The training objective becomes to maximize

$$p_{\text{model}}(\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(\tau)} | \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(\tau)})$$

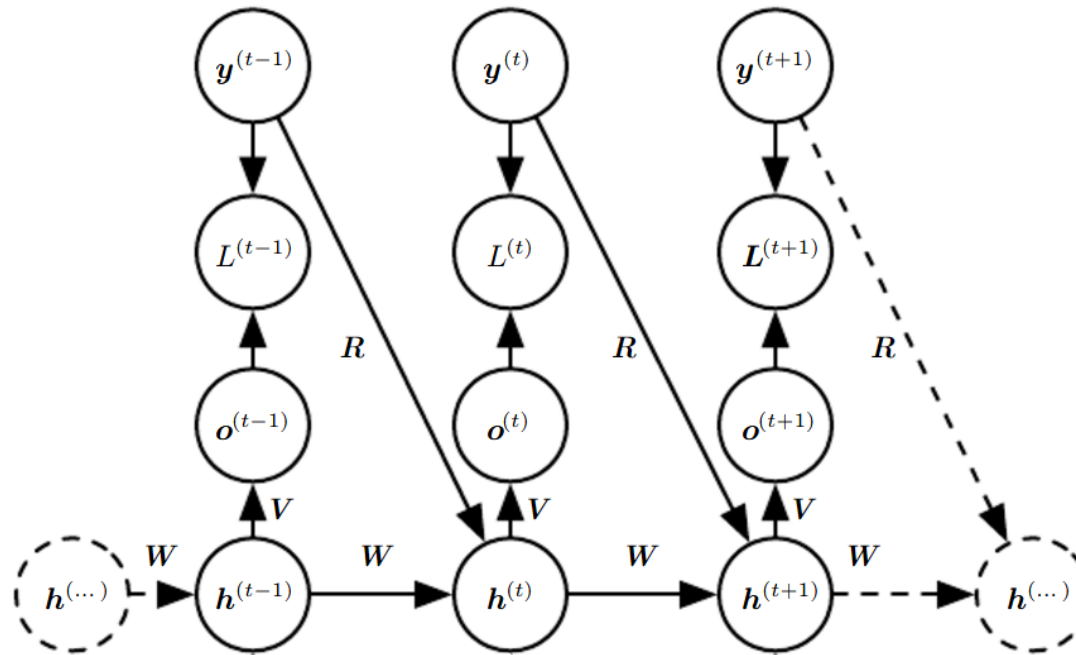
$$= \prod_{t=1}^{\tau} p_{\text{model}}(\mathbf{y}^{(t)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}, \mathbf{y}^{(1)}, \dots, \mathbf{y}^{(t-1)})$$

where  $\mathbf{y}^{(t)}$ 's are modeled to be **conditionally dependent** (cf. Pattern I)

- This type of networks allows modeling an arbitrary distribution over the  $\mathbf{y}$  sequence given the  $\mathbf{x}$  sequence of the same length, whether  $\mathbf{y}^{(t)}$ 's are conditionally independent or dependent
- One may as well broadcast <sup>the process of generating textual description of an image</sup> one single vector  $\mathbf{x}$  as  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(\tau)}$ , as is often seen in **image captioning**; in such tasks,  $\mathbf{x}$  may denote the feature vector of an image and  $\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(\tau)}$  are its caption 圖片說明文字

$$p(y_1, y_2, \dots, y_n) = \prod_t p(y_t | y_{<t}) \quad \text{autoregressive}$$

- ★ One can also turn the network into a **generative** model for unsupervised learning by omitting all the  $x^{(t)}$ 's



- Essentially, this corresponds to learning a model that factorizes as

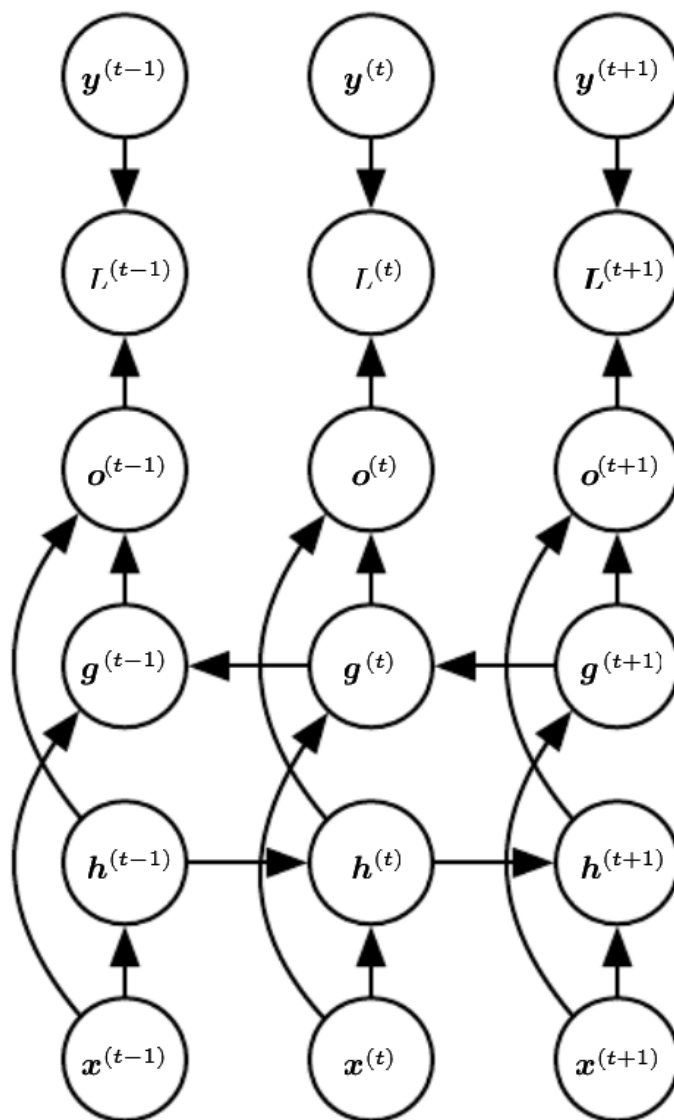
$$p_{\text{model}}(\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(\tau)}) = \prod_{t=1}^{\tau} p_{\text{model}}(\mathbf{y}^{(t)} | \mathbf{y}^{(t-1)}, \mathbf{y}^{(t-2)}, \dots, \mathbf{y}^{(1)})$$

*autoregressive*

## Bidirectional RNN

---

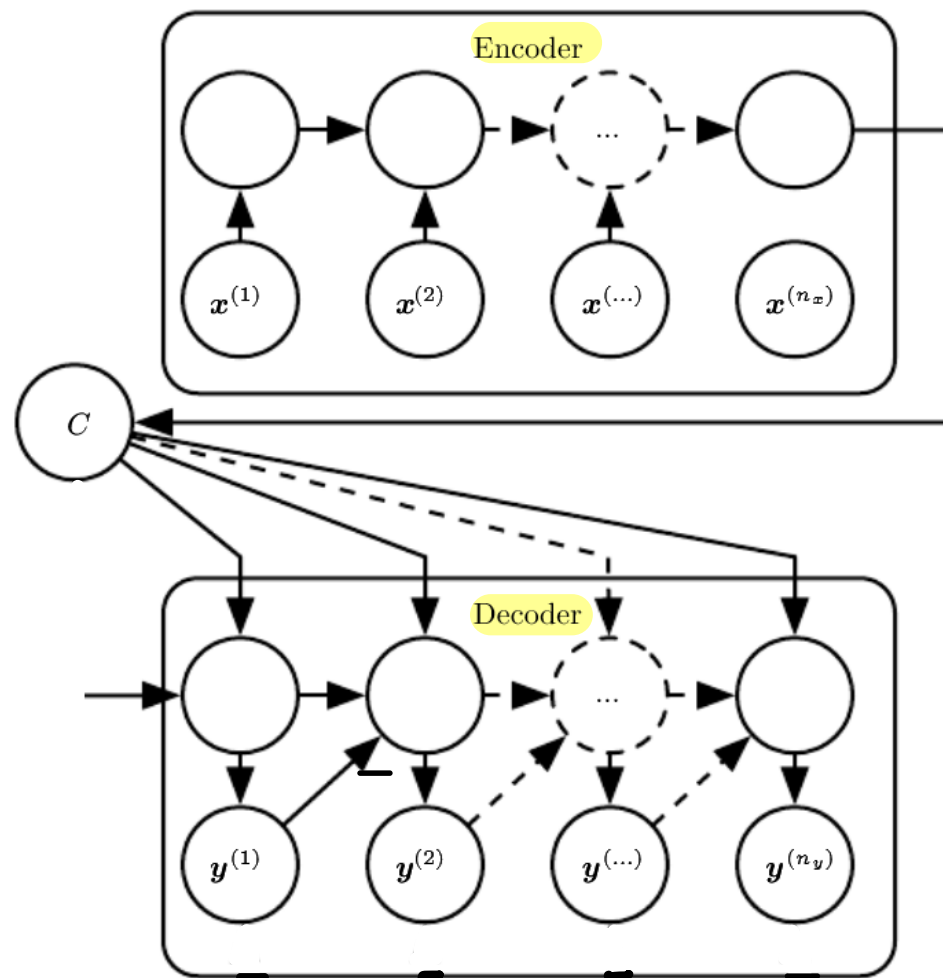
- RNN has a causal structure: the state  $\mathbf{h}^{(t)}$  at time  $t$  captures only information from the past  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t-1)}$  and the present  $\mathbf{x}^{(t)}$  inputs
- In many applications, it may be necessary to output a prediction  $\mathbf{y}^{(t)}$  that depend on the entire sequence
- Bidirectional RNN combines two RNNs, one moving forward in time and the other moving backward, to capture information from both the past and future
- This however requires the entire sequence be buffered
- The notion can readily be extended to 2-D signals, e.g. images



## Sequence-to-Sequence Networks

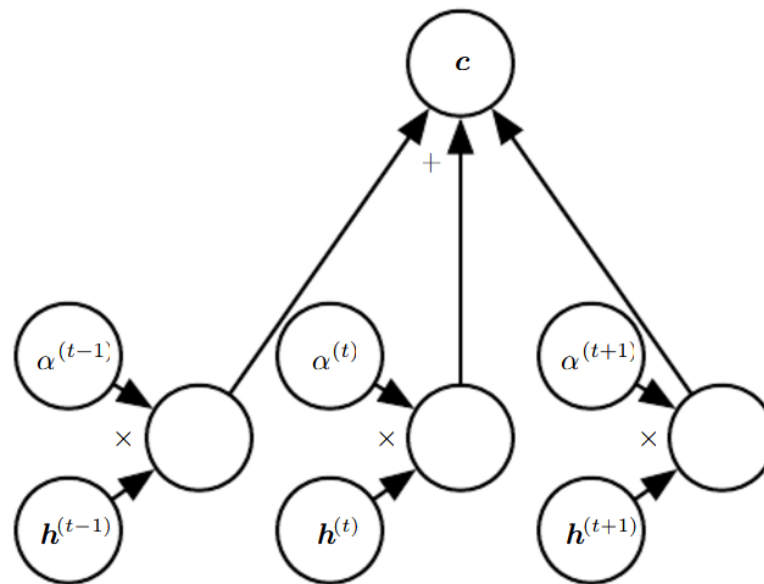
- In some applications, e.g. machine translation and question answering, it may be necessary to map an input sequence to an output sequence that is **not of the same length**
- The encoder network encodes the input sequence and emits a context  $C$  for the decoder network to generate the output sequence 發出, 射出, 散發
- The context  $C$  can be a function of the last hidden unit or a summary of different hidden units by introducing an **attention mechanism**





## Attention Mechanisms

- A weighted average of information with weights  $\alpha^{(t)}$ 's (gate signals) produced by the model itself



- The information to be averaged could be **hidden units** of a network or **raw input data**
- The **weights** are usually produced by applying a **softmax** function to

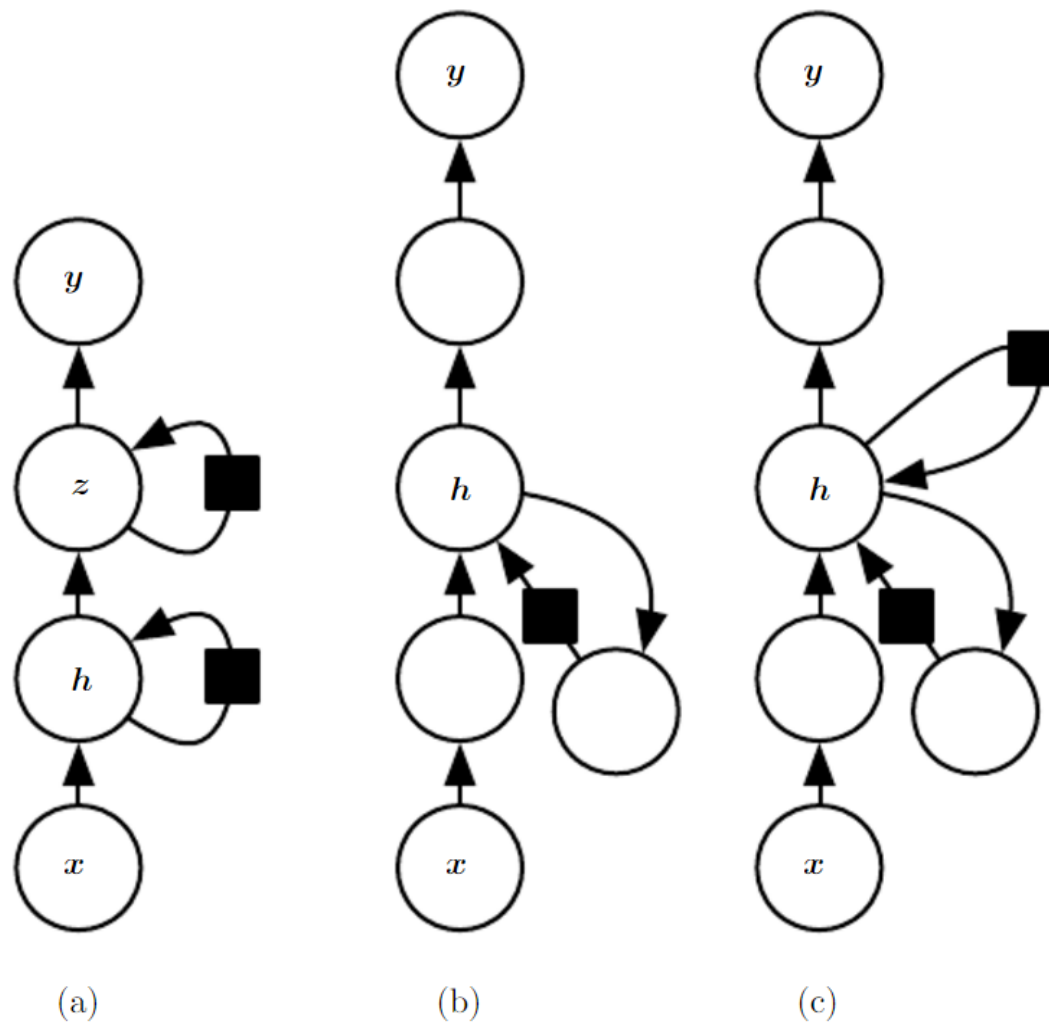
some relevance scores emitted somewhere in the model

- The **soft weighting** is more expensive than direct indexing but is differentiable, making it trainable with gradient-based algorithms

## Deep Recurrent Networks

---

- RNN can be made deep in many ways
- Recurrent hidden units can be organized in a layered manner
- MLP can be introduced in the input-to-hidden, hidden-to-hidden, and hidden-to-output parts
- In doing so, skip connections may be necessary to mitigate the gradient vanishing and exploding problem



## Review

---

- Design patterns of RNN and their probability models
- Gradient vanishing and exploding problem
- Long short-term memory
- Back-propagation through time (BPTT)
- Training with output recurrence
- Attention mechanisms