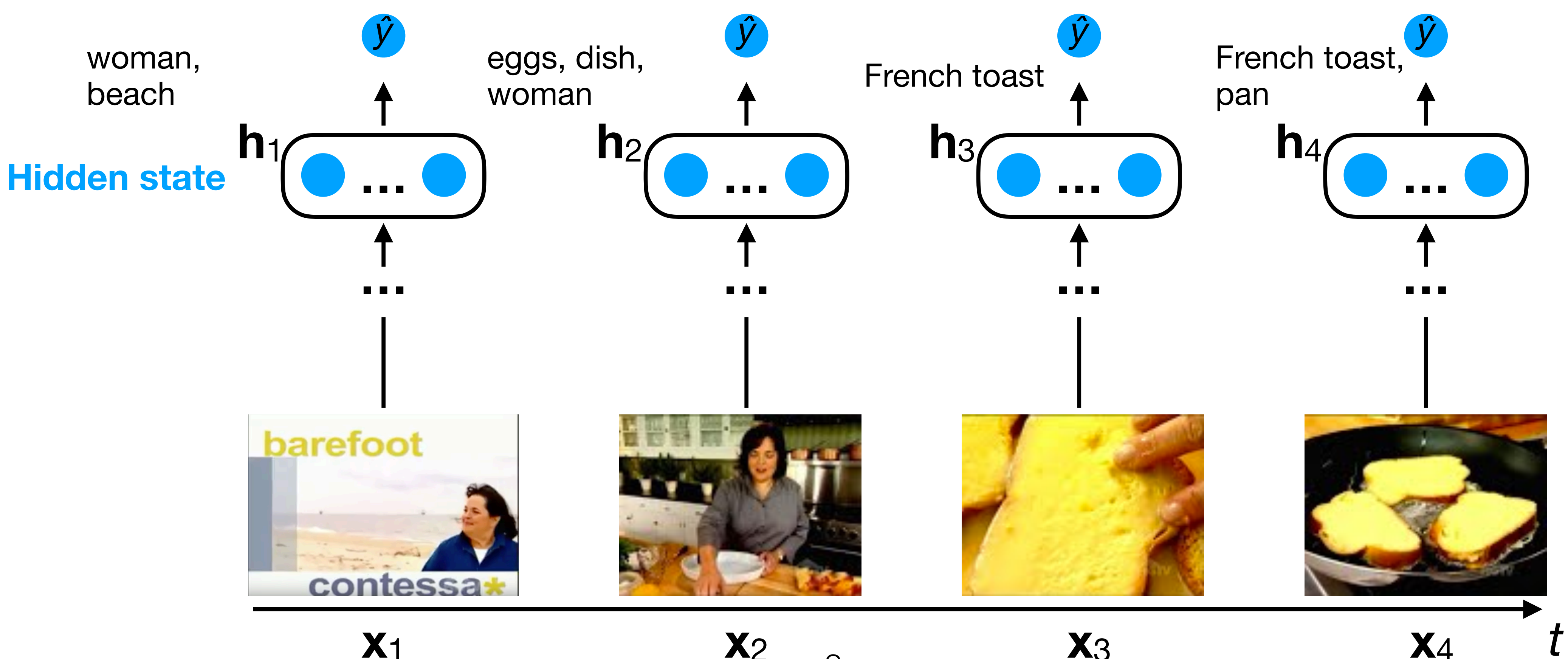
CS/DS 541: Class 12

Jacob Whitehill

Recurrent neural networks (RNNs)

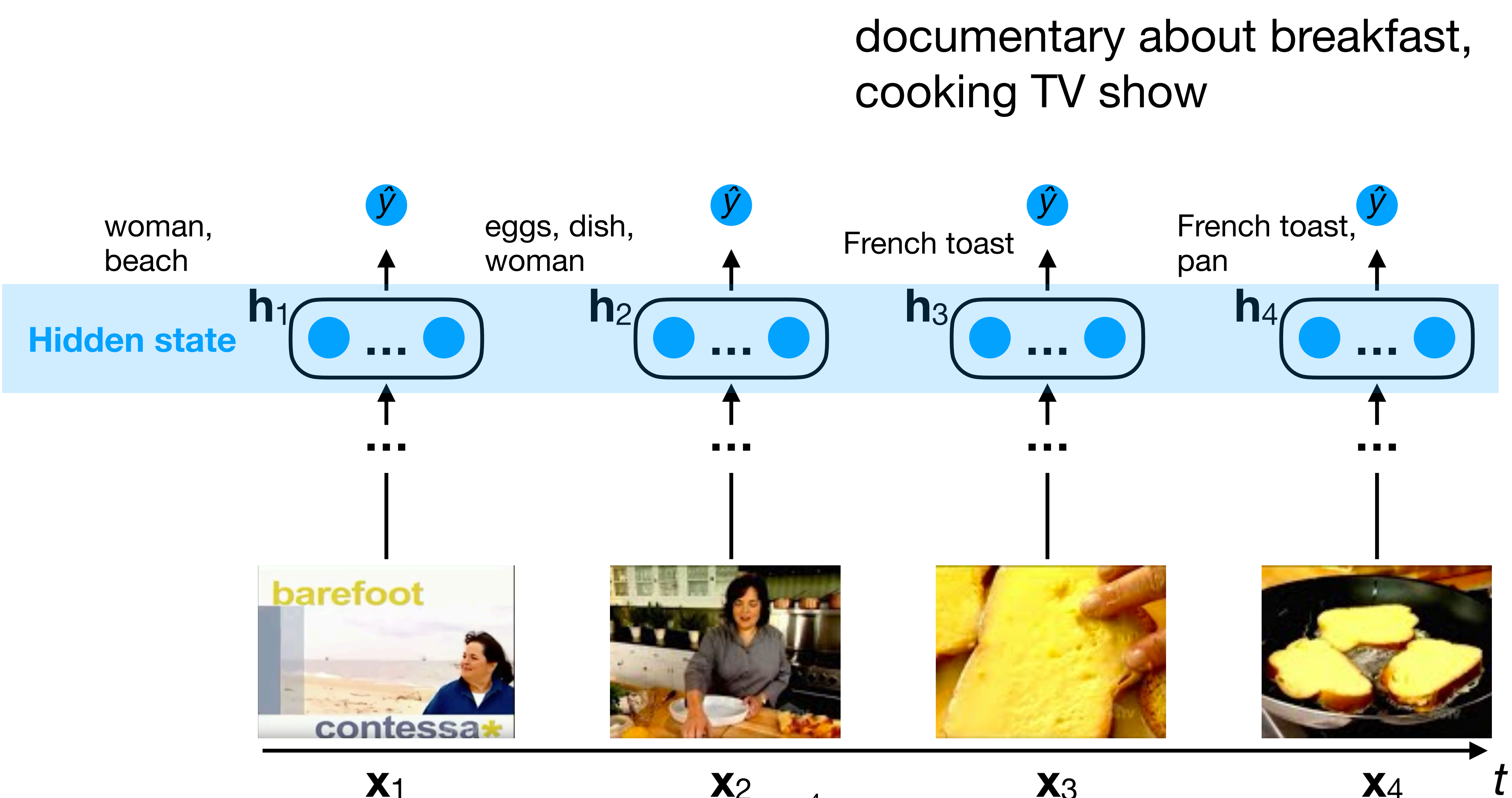
Role of the hidden state

- A CNN applied to each image in the sequence can tell us about the *individual objects* contained in *each frame*.
- Here, the hidden state $\mathbf{h}_t$ of each input $\mathbf{x}_t$ is computed **independently**, i.e.: $\mathbf{h}_t = f(\mathbf{x}_t)$



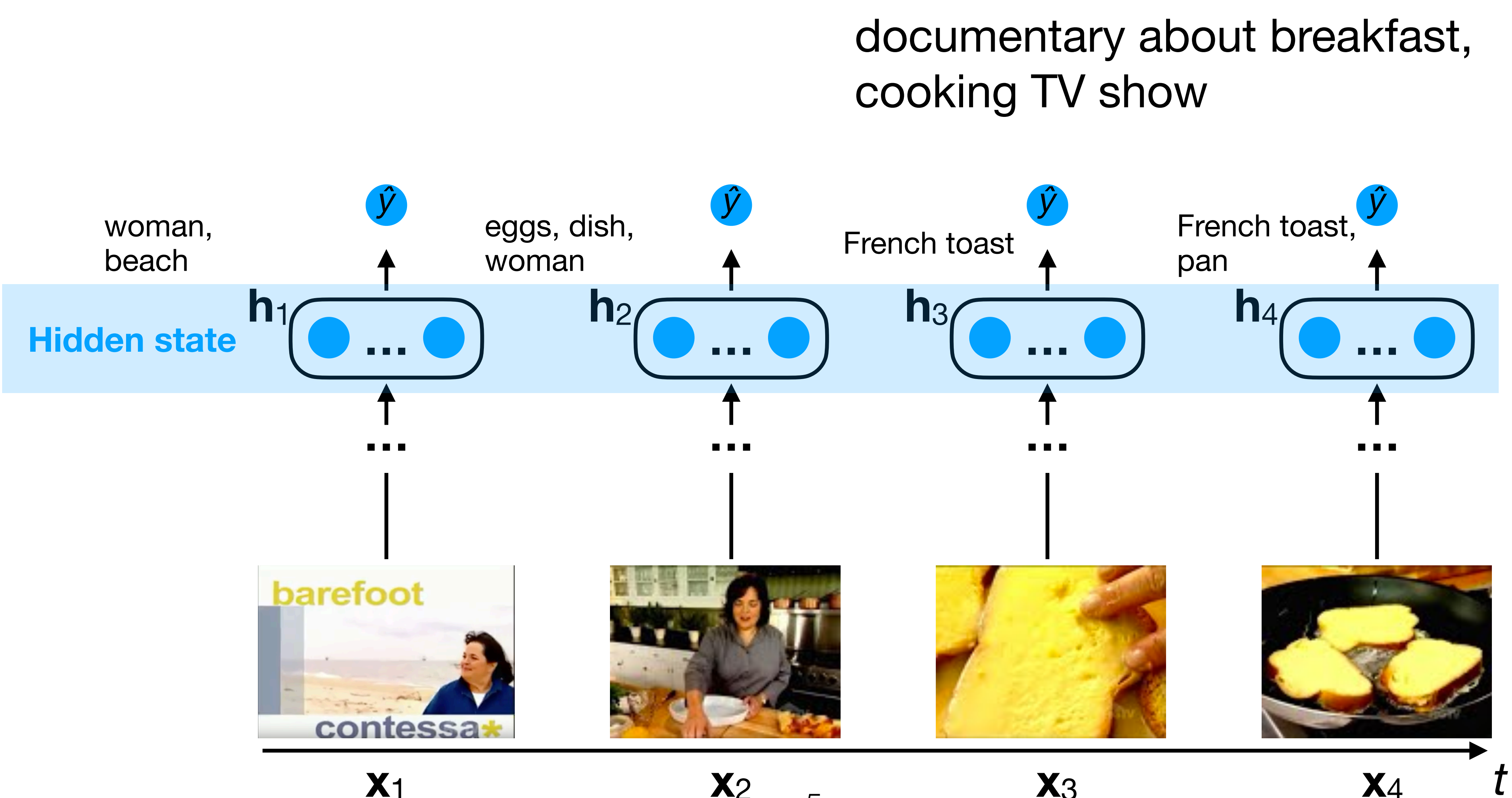
Role of the hidden state

- However, sometimes we need to **aggregate over time** to understand how the individual objects relate to each other:



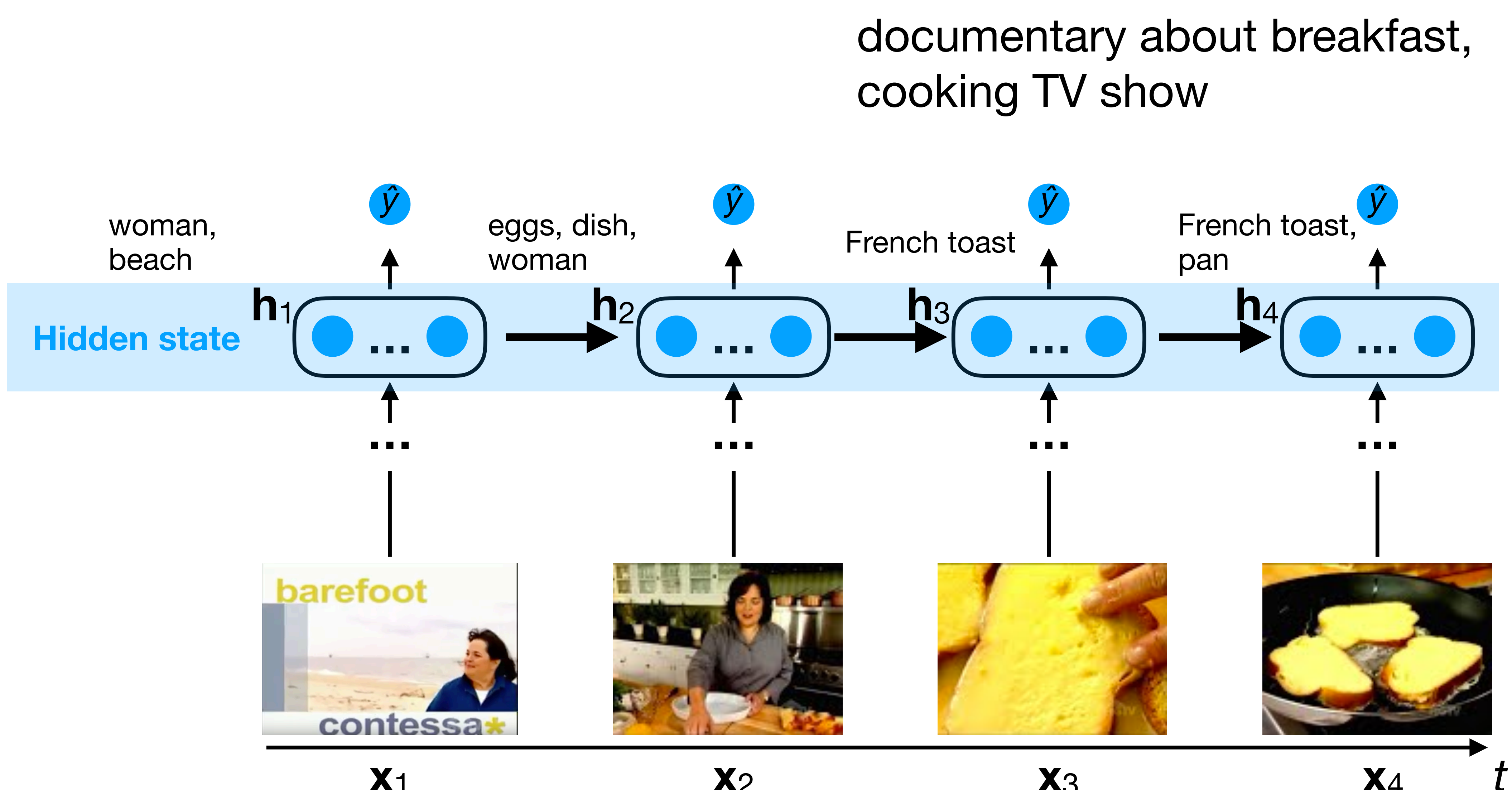
Role of the hidden state

- It’s also possible that past hidden state helps us to infer the current hidden state (e.g., the French toast looks blurry at time t=4 but was very clear at time t=3).



Role of the hidden state

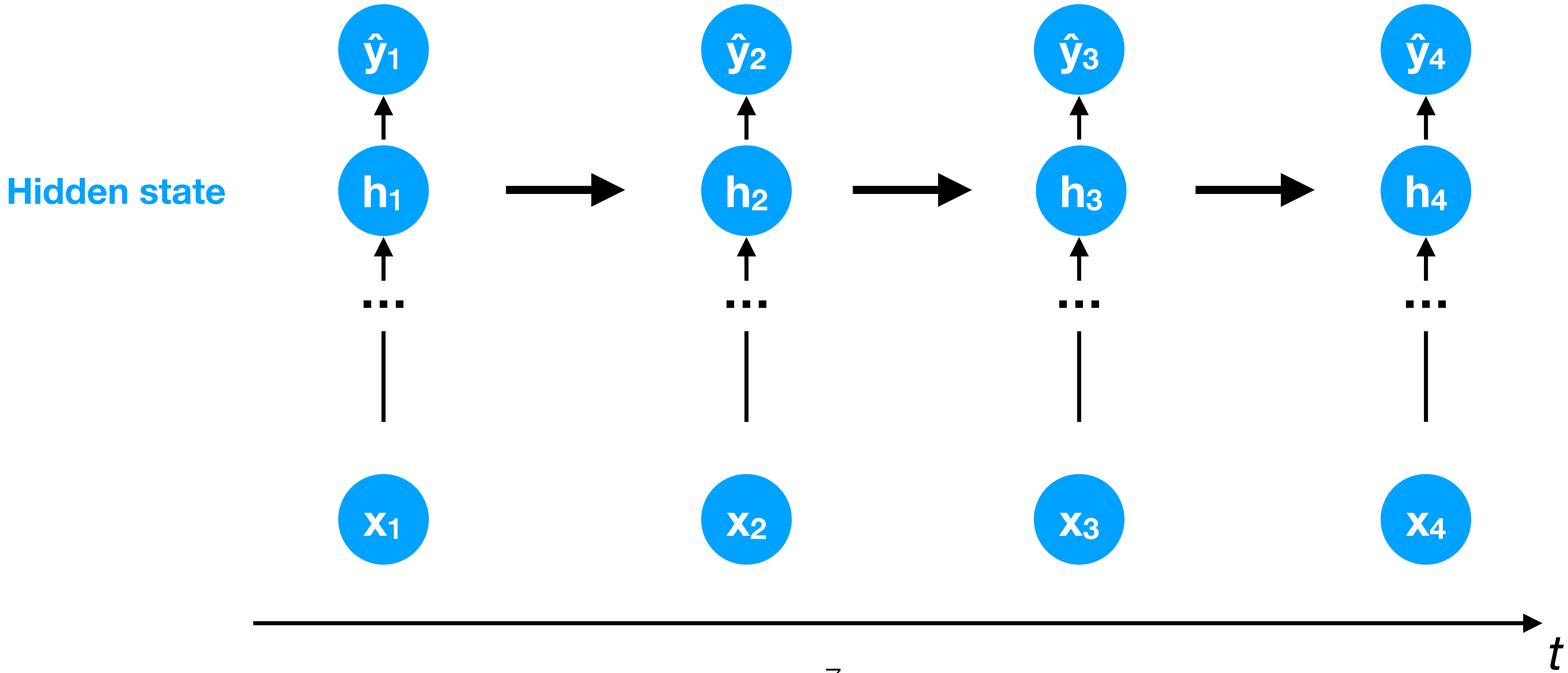
- To accomplish this, we can link the hidden state across the elements of the sequence, i.e.: $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$



Jacob Whitehill, WPI

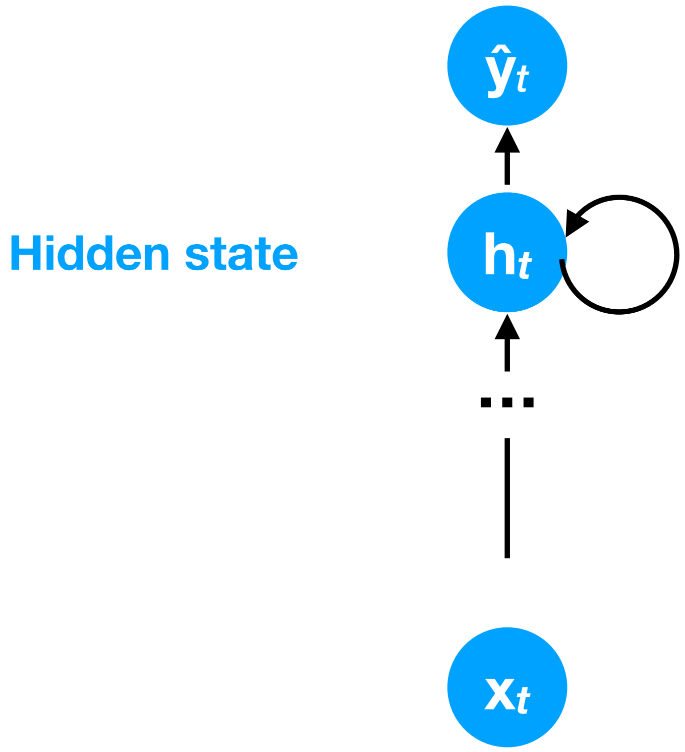
Recurrent neural network

- This is the essence of a recurrent neural network (RNN) — the hidden states at time *t* depend on the hidden states of time *t*-1: $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$

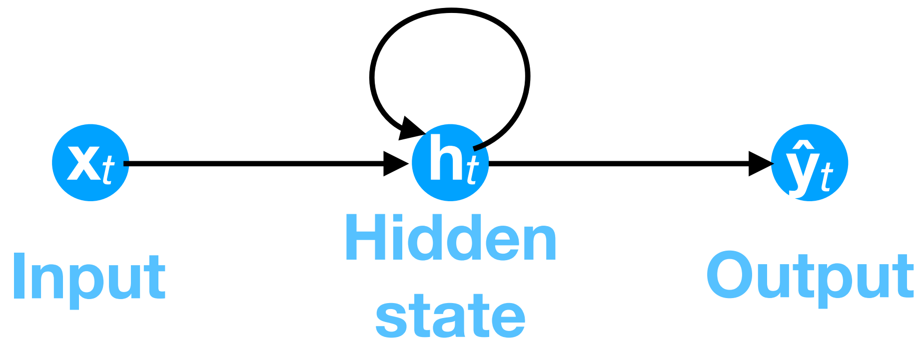


Recurrent neural network

- We sometimes represent the recurrent hidden state as a single node with an arrow to itself:

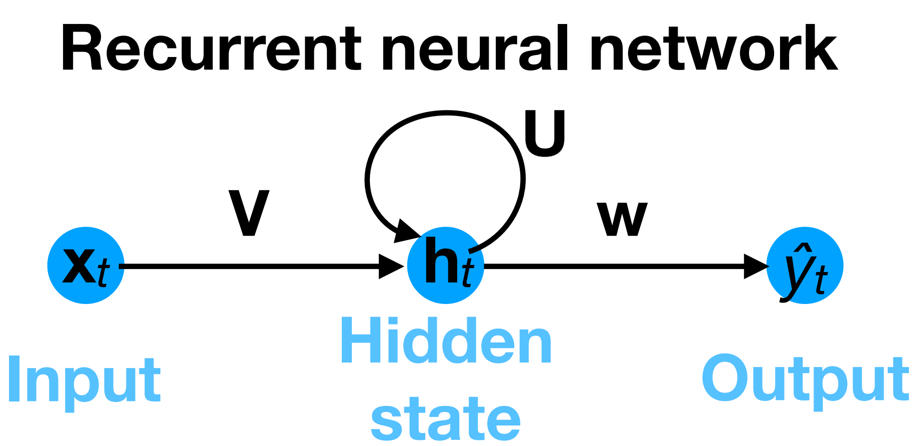


or



Recurrent neural network

- We can construct a simple **recurrent neural network** (RNN) as follows:



$$
\begin{aligned}
\hat{y}_t = g(\mathbf{x}_1, \ldots, \mathbf{x}_t; \mathbf{U}, \mathbf{V}, \mathbf{w}) &= \mathbf{h}_t^\top \mathbf{w} \\
\mathbf{h}_t &= \sigma\left(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{V}\mathbf{x}_t\right)
\end{aligned}
$$

Recurrent neural network

- As the activation σ, we typically use **tanh** instead of the logistic function:
  - Range of tanh: (-1, +1)
  - Range of logistic: (0, 1)
- Reason:
  - We want the hidden state to be able to maintain its value across many time steps (t=1, 2, 3, …).
  - tanh(x) is approximated by the line y=x around x=0.

Recurrent neural network

- Consider a unidimensional hidden state. Let h_1=0.
- If σ=logistic, then:
  - $h_2=\sigma(h_1)=\sigma(0)=0.5$
  - $h_3=\sigma(h_2)=\sigma(0.5)=0.622$
  - $h_4=\sigma(h_3)=\sigma(0.622)=0.651$
  - …

Recurrent neural network

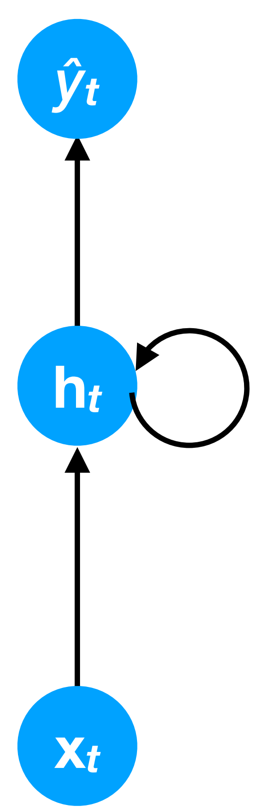
- Consider a unidimensional hidden state. Let h_1=0.
- If σ=tanh, then:
    - $h_2=\sigma(h_1)=\sigma(0)=0$
    - $h_3=\sigma(h_2)=\sigma(0)=0$
    - $h_4=\sigma(h_3)=\sigma(0)=0$
    - …

Recurrent neural network

- The hidden state remains stable for tanh but not for logistic.
- Note that the *gradient* of tanh at 0 is exactly 1 because:
  $\tanh'(x) = 1 - \tanh^2(x)$
  - Learning can still occur!

Training a RNN

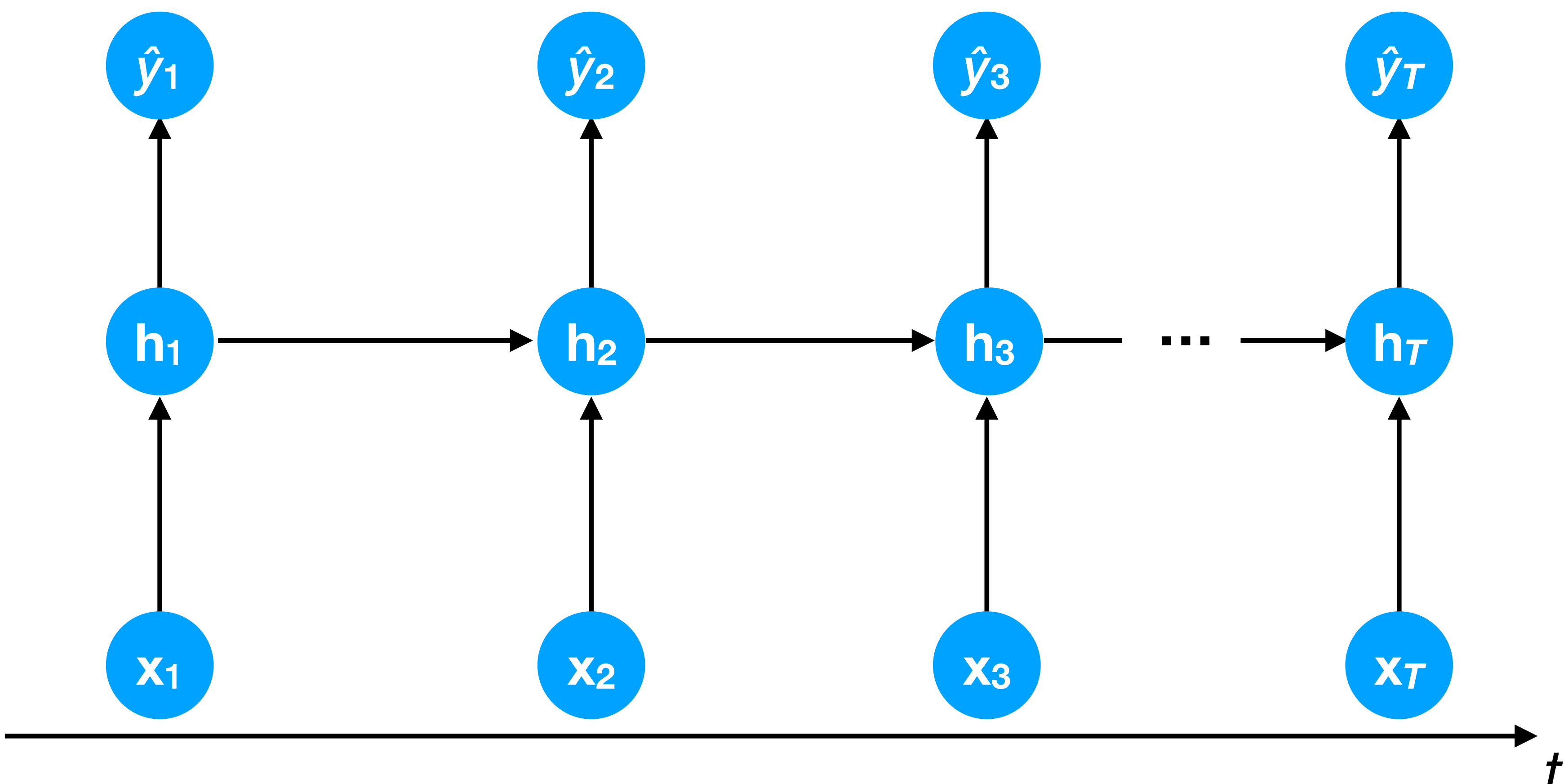
- We can use the same back-propagation procedure to train an RNN as we did for feed-forward neural networks (FFNN).
- For every input sequence ($\mathbf{x}_1$, …, $\mathbf{x}_T$) and labels (y_1, …, y_T), we **unroll** the computational graph over all T time-steps.



Training a RNN

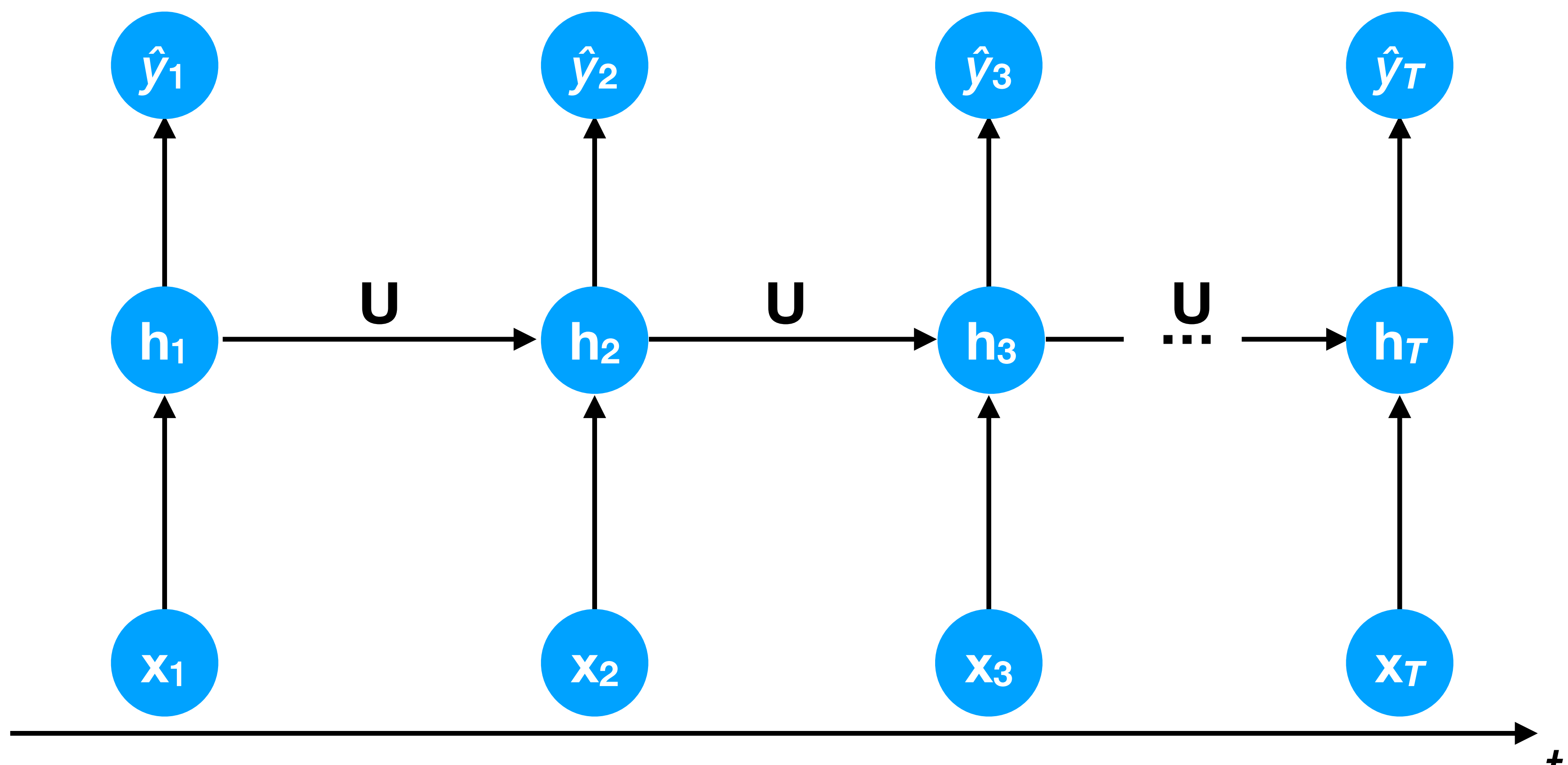
- We can use the same back-propagation procedure to train an RNN as we did for feed-forward neural networks (FFNN).

- For every input sequence ($\mathbf{x}_1$, …, $\mathbf{x}_T$) and labels (y_1, …, y_T), we **unroll** the computational graph over all T time-steps.



Training a RNN

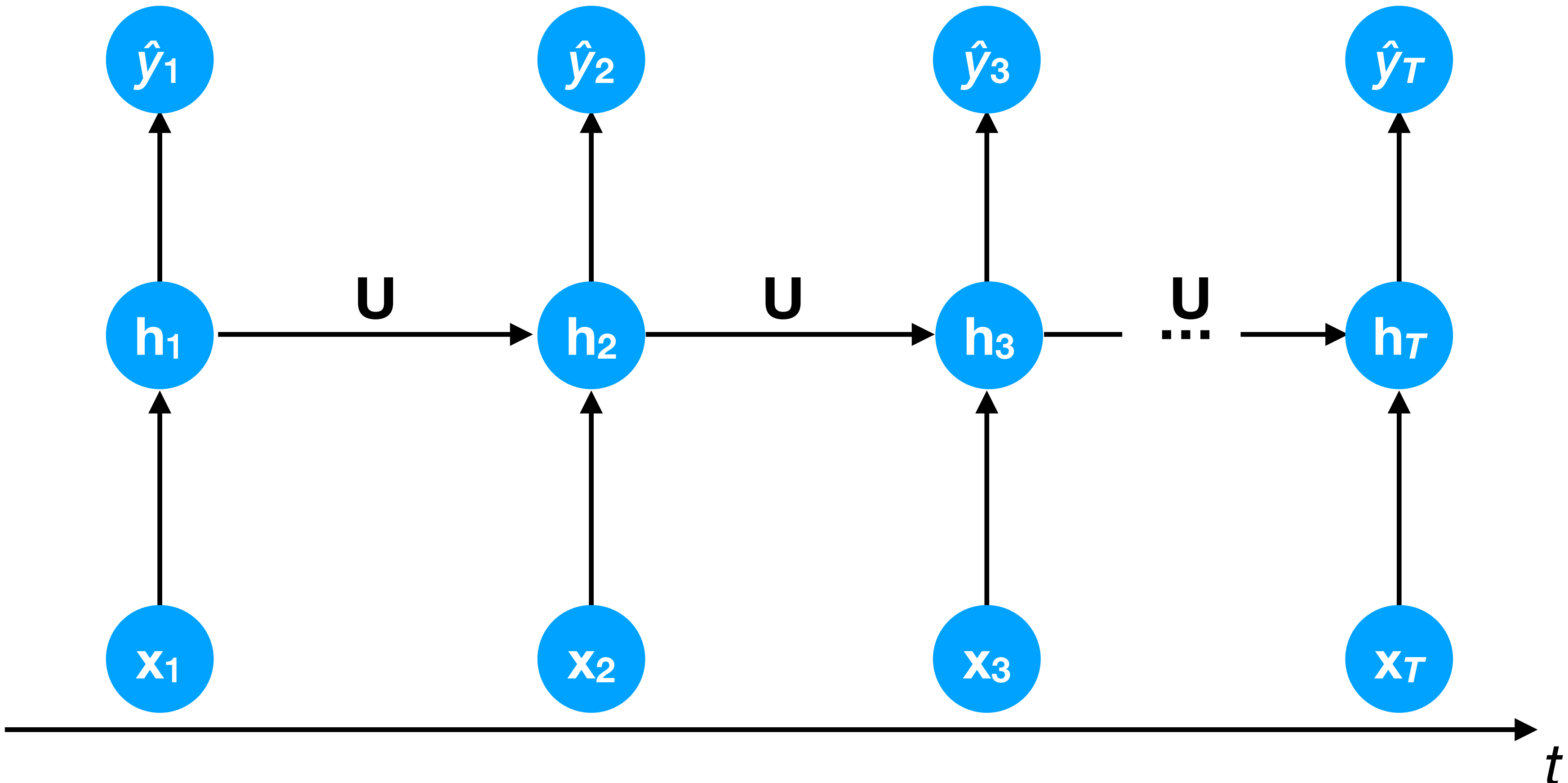
- The same weights (e.g., **U**) are used in all time-steps, and we need to *sum* their contributions to the gradient $\nabla_{\mathbf{U}} J$ of the loss function *J*.



Training a RNN

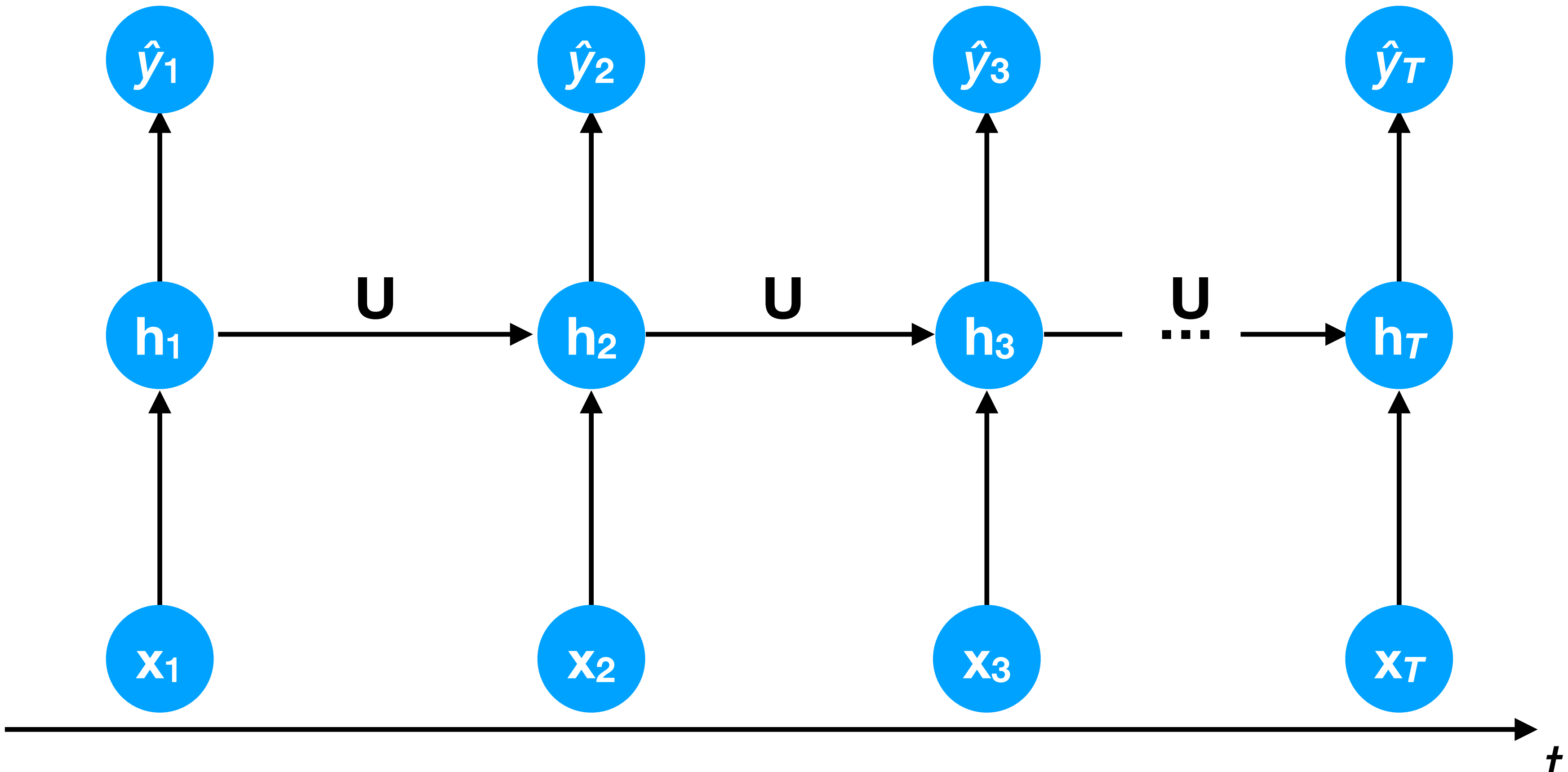
- Moreover, we need to sum the loss values J_t at every timestep t:

$$J(\mathbf{U}) = \sum_{t=1}^{T} J_t(y_t, \hat{y}_t; \mathbf{U})$$



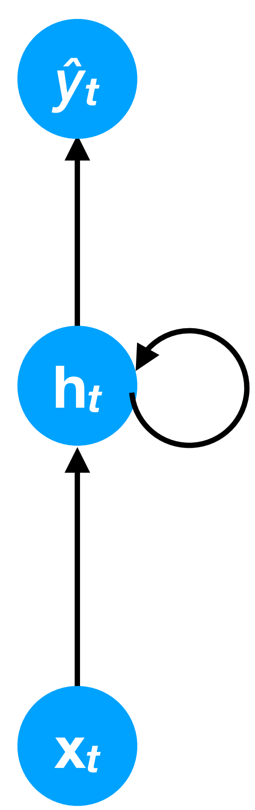
Training a RNN

- This process is called **back-propagation through time** (BPTT).



Depth in RNNs

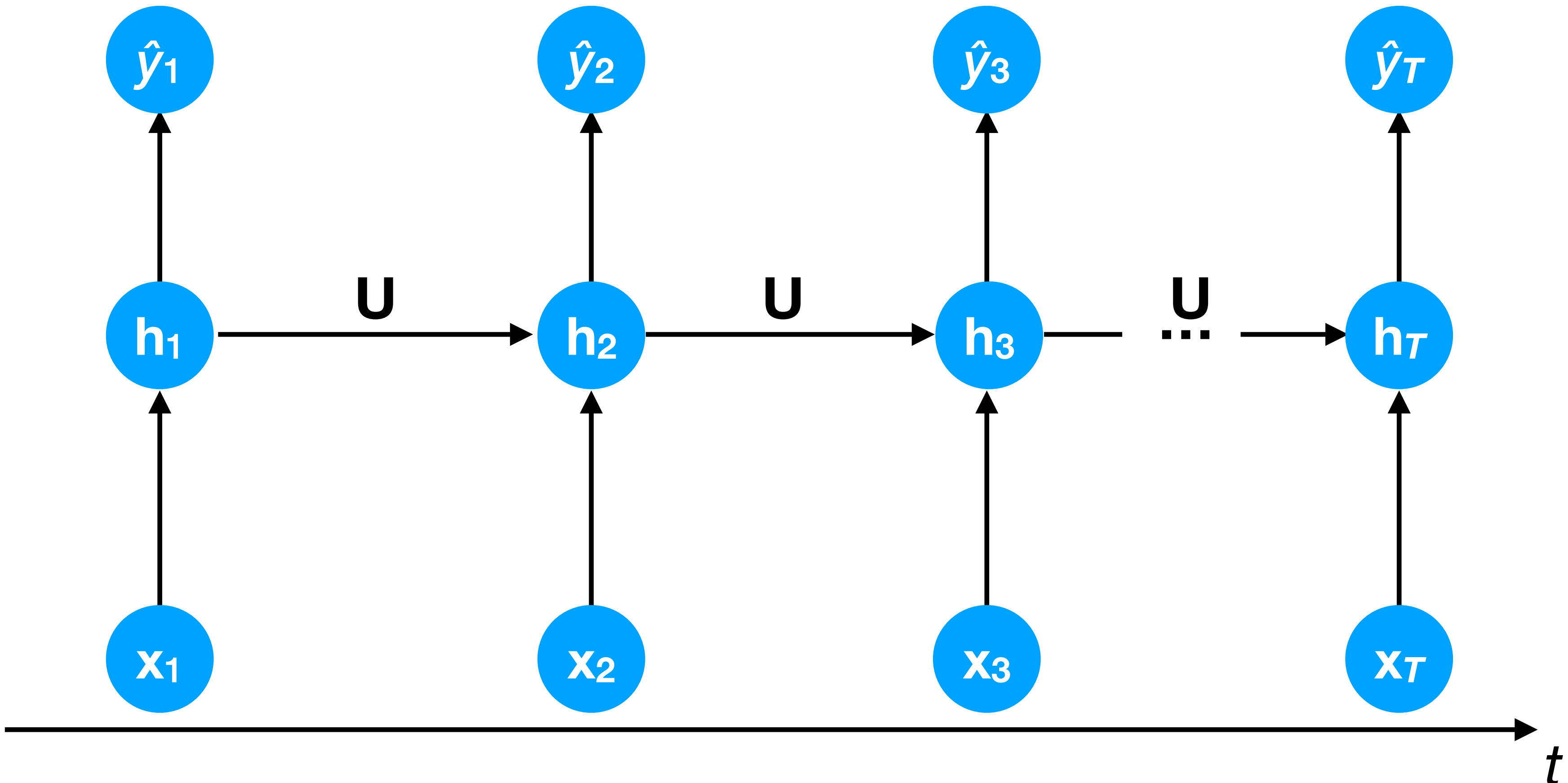
- Note that, in the example RNN below, the “base” network is very simple and has just 1 hidden layer:



Depth in RNNs

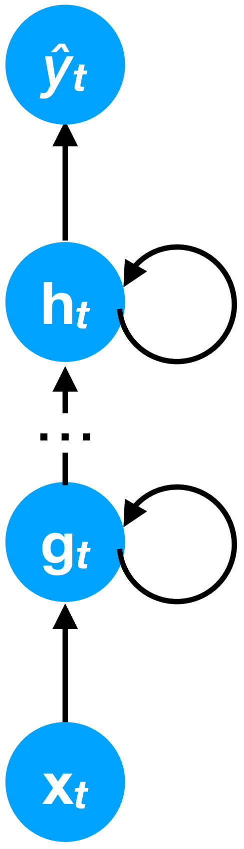
- However, across timesteps, the maximum path-length from any input to any output can be large if *T* is large.

- In this sense, RNNs are deep networks.



Depth in RNNs

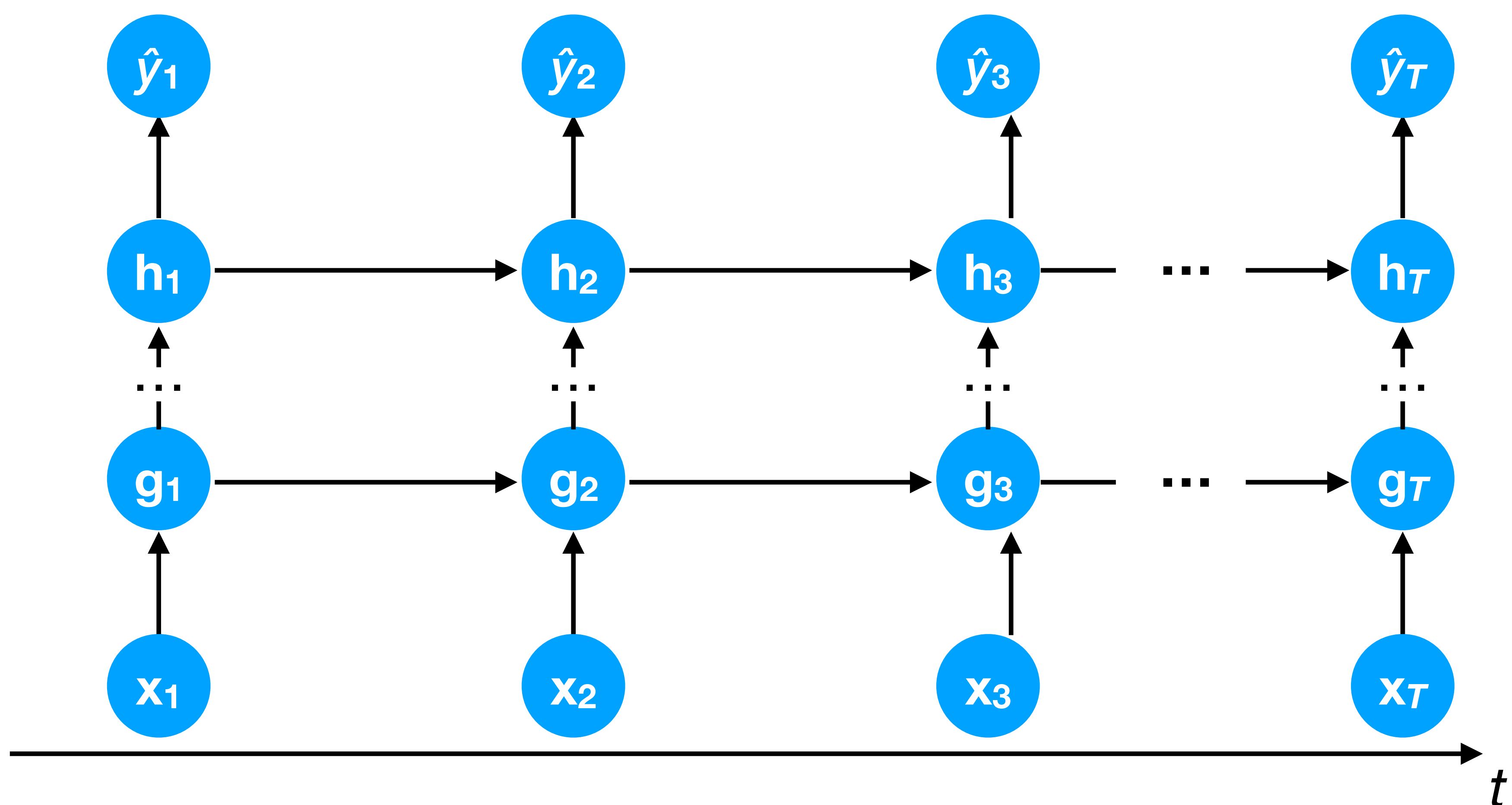
- Note that we can also make the base RNN itself deep:
  - $\mathbf{g}_t$ depends on $\mathbf{x}_t$ and $\mathbf{g}_{t\text{-}1}$.
  - $\mathbf{h}_t$ depends on $\mathbf{g}_t$ and $\mathbf{h}_{t\text{-}1}$.



Jacob Whitehill, WPI

Depth in RNNs

- Note that we can also make the base RNN itself deeper:
  - $\mathbf{g}_t$ depends on $\mathbf{x}_t$ and $\mathbf{g}_{t-1}$.
  - $\mathbf{h}_t$ depends on $\mathbf{g}_t$ and $\mathbf{h}_{t-1}$.

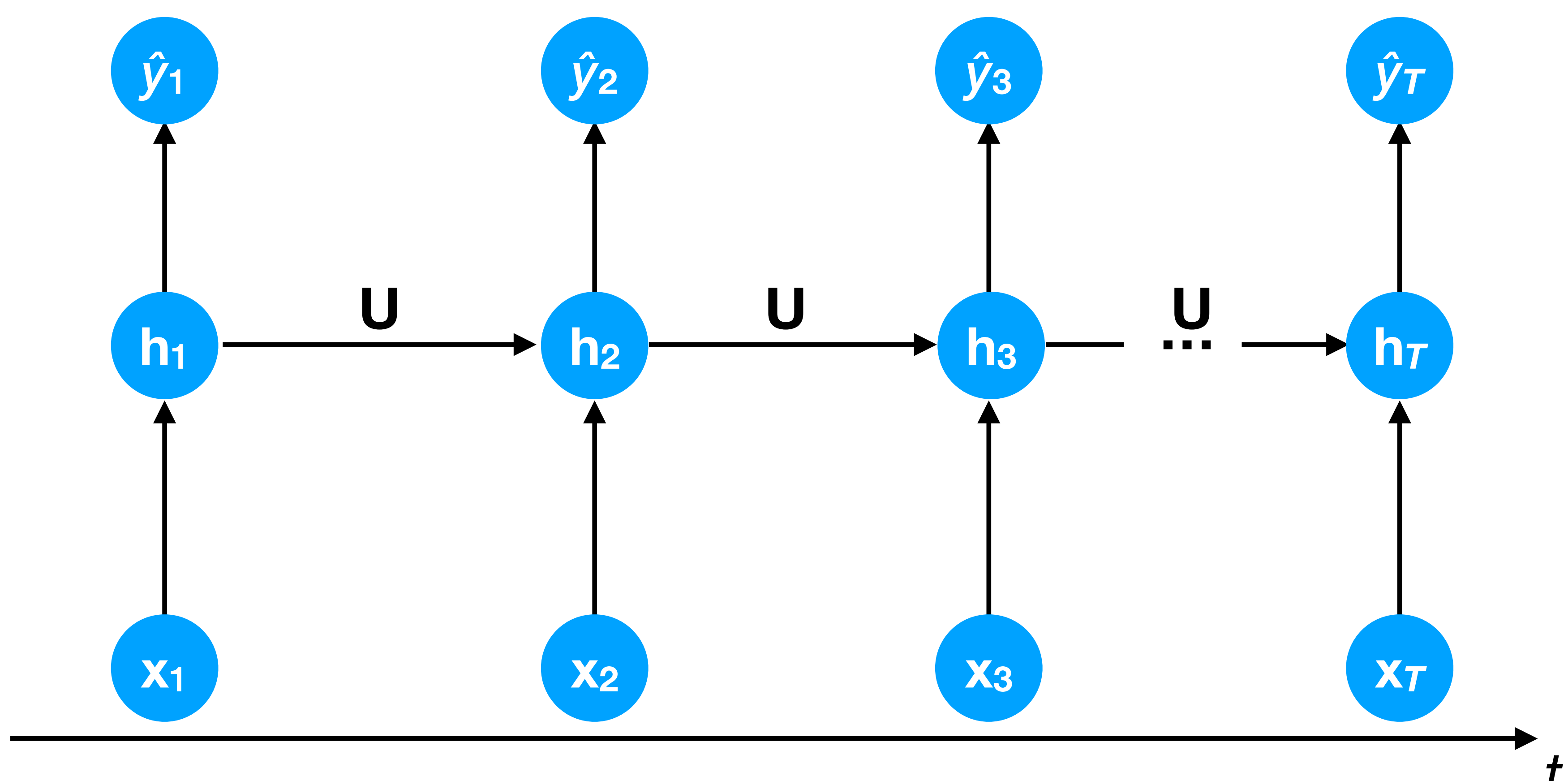


Jacob Whitehill, WPI

Difficulty in training RNNs

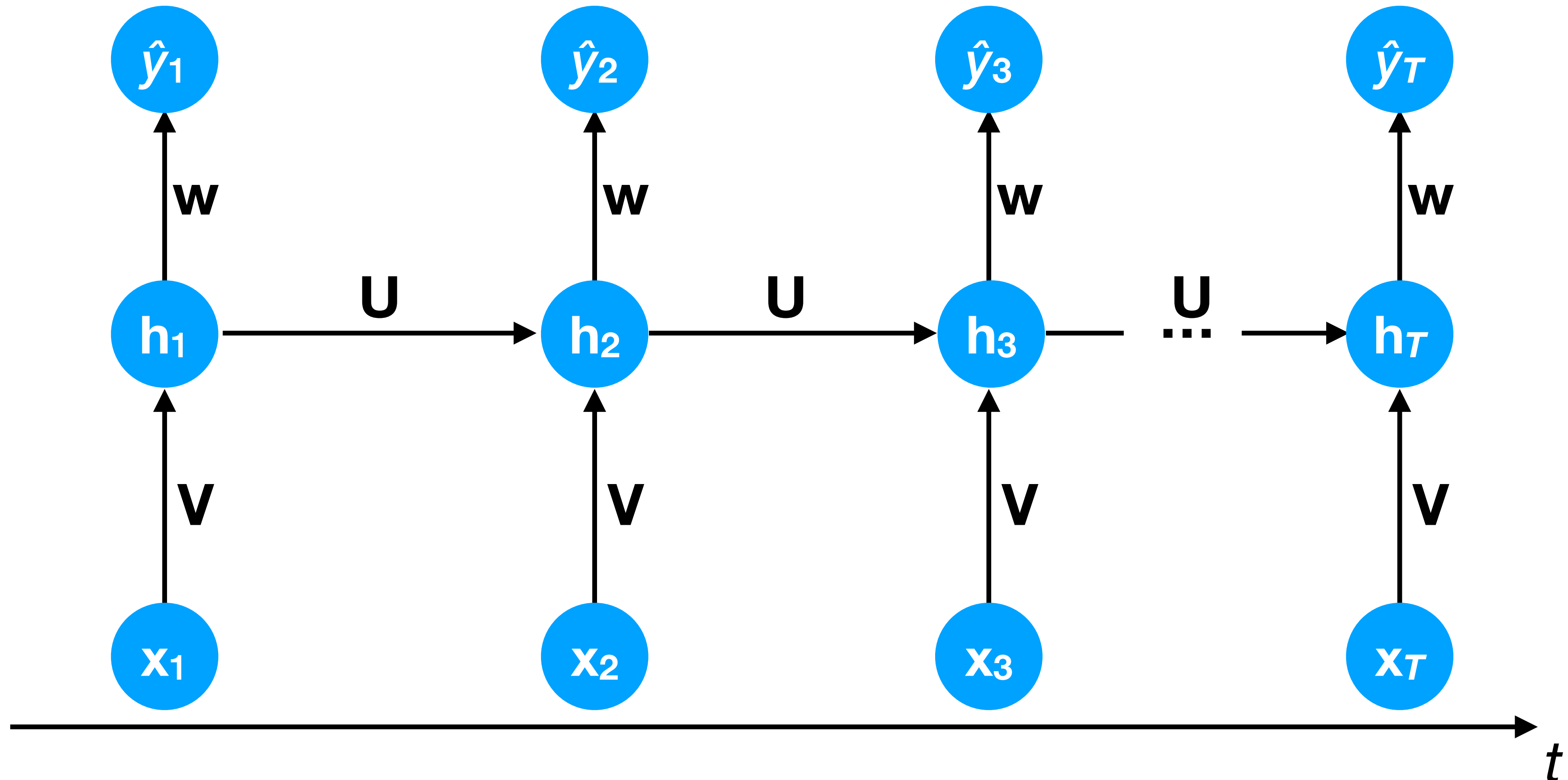
Difficulty in training RNNs

- In their simplest form, RNNs are typically hard to train:
  - The gradients can occasionally become very large (**exploding gradient**), which forces us to use a very small learning rate (which makes training slow).



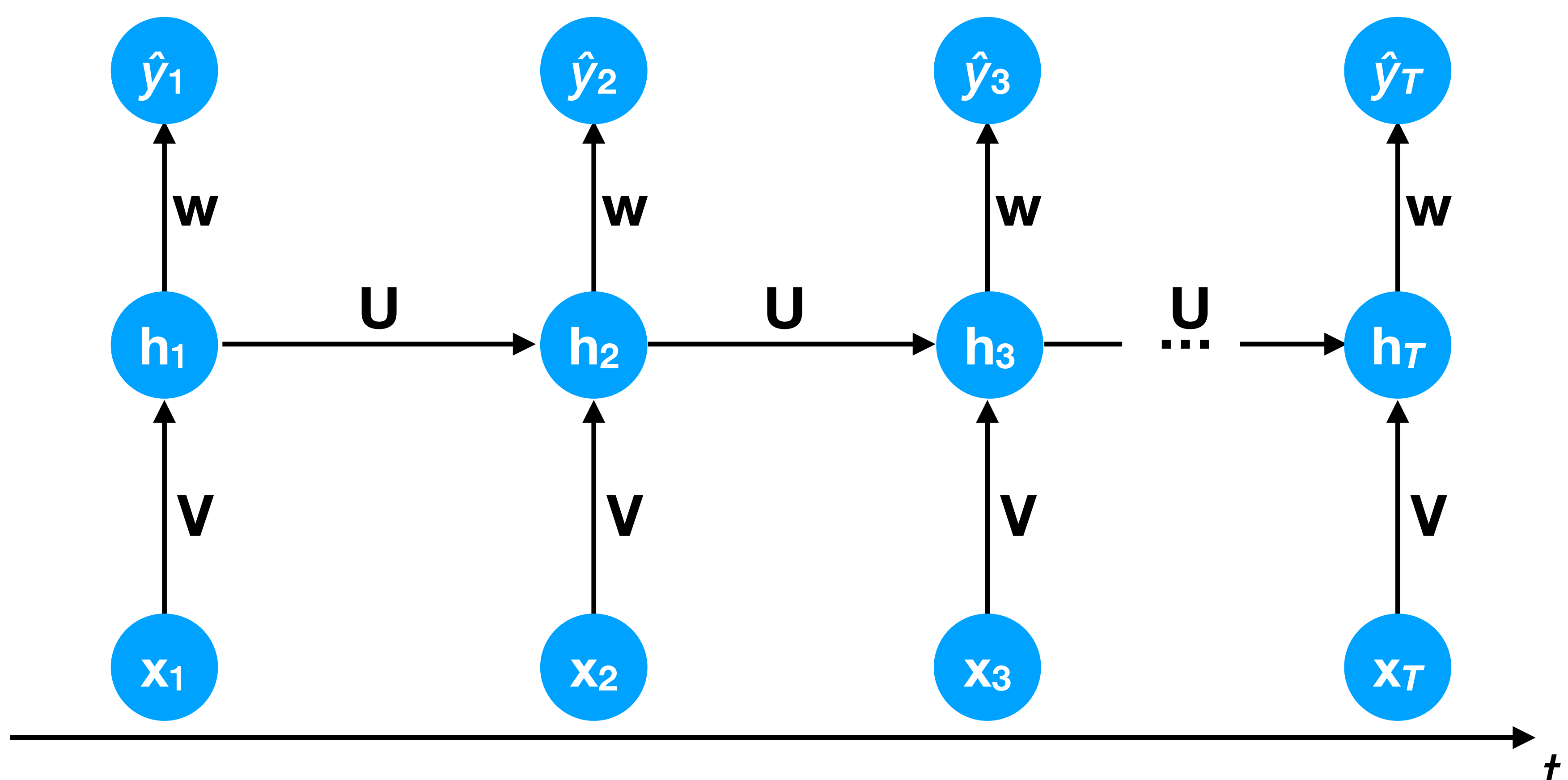
Difficulty in training RNNs

- In their simplest form, RNNs are typically hard to train:
  - The gradients can also become very small (**vanishing gradient**), which also makes learning very slow.



Difficulty in training RNNs

- A related problem is that, if T is large, then information *early* in the input sequence (e.g., $\mathbf{x}_1$) can "get lost" when trying to predict values *late* in the sequence (e.g., $\hat{y}_T$).

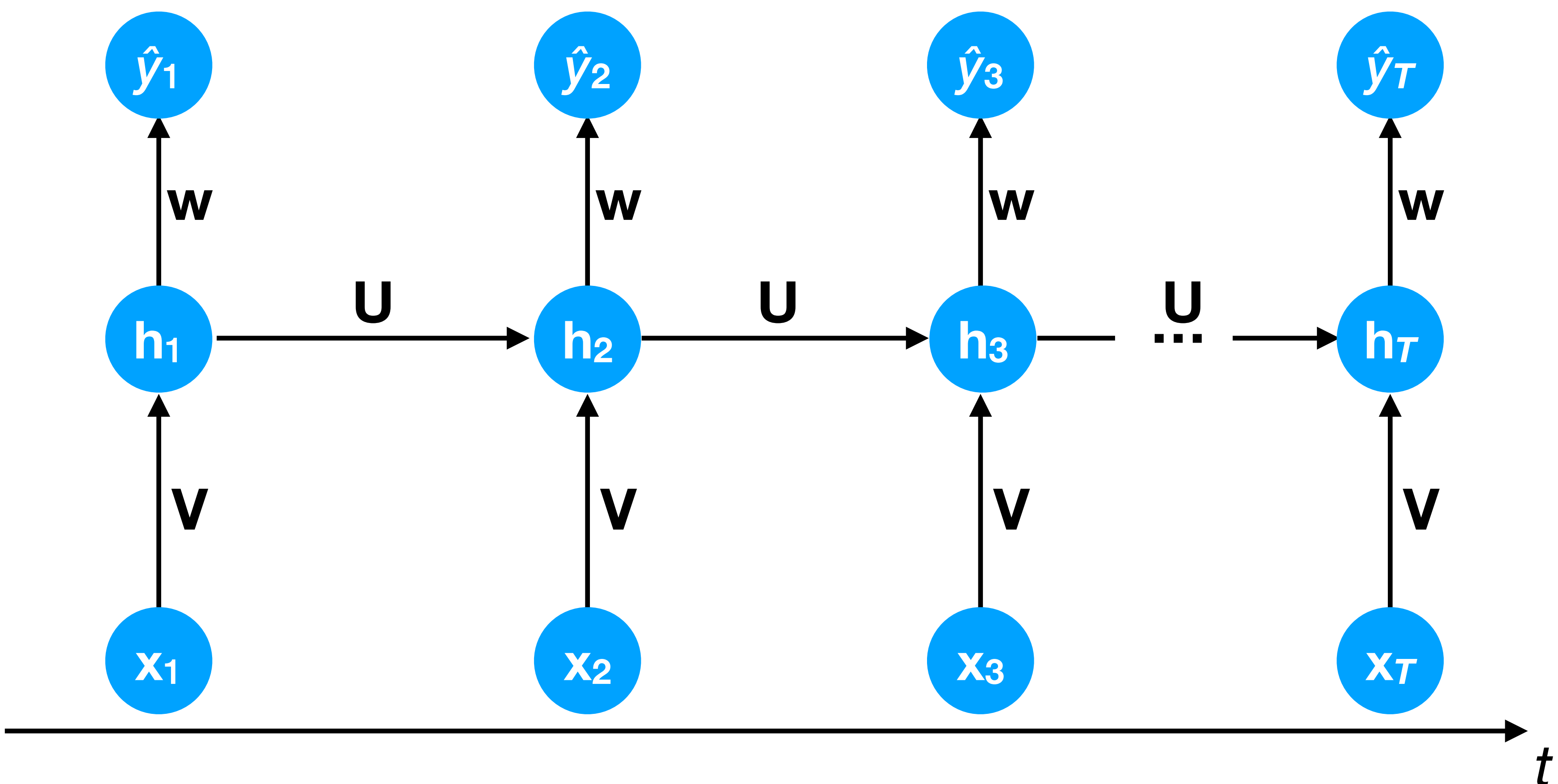


Difficulty in training RNNs

- Why do these problems occur?

- Let's consider a simplistic RNN *without* a non-linear activation function, i.e.:

$$\mathbf{h}_t = \mathbf{U}\mathbf{h}_{t-1} + \mathbf{V}\mathbf{x}_t$$

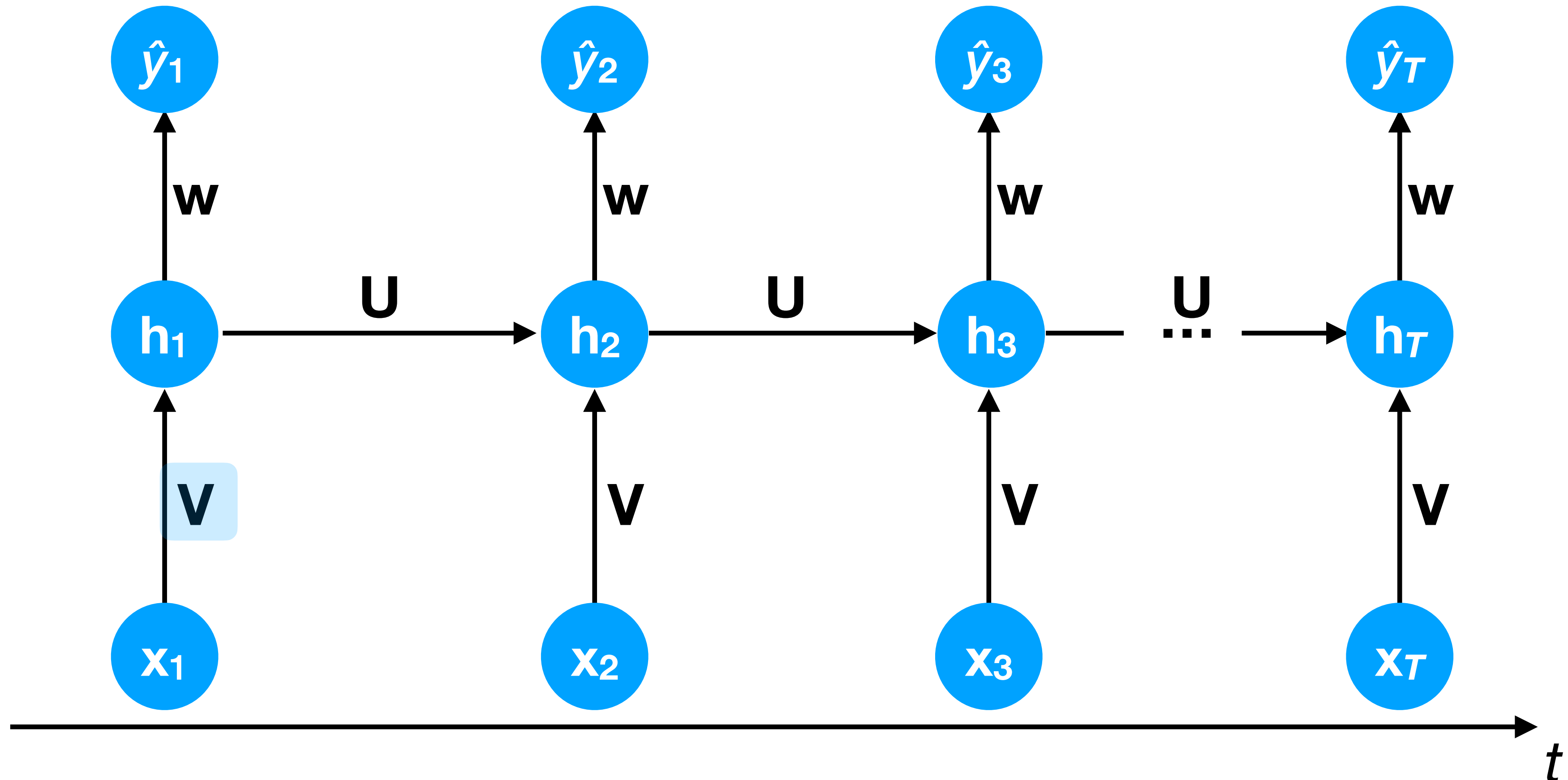


Difficulty in training RNNs

- When we compute the gradient of $J(y_T, \hat{y}_T)$ w.r.t. **V**, we must sum over all occurrences of **V** —including the *first* one:

$$\frac{\partial J_T(\mathbf{V})}{\partial \mathbf{V}} = \frac{\partial J_T(\mathbf{V})}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_{T-1}} \cdots \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{V}} + \text{other terms}$$

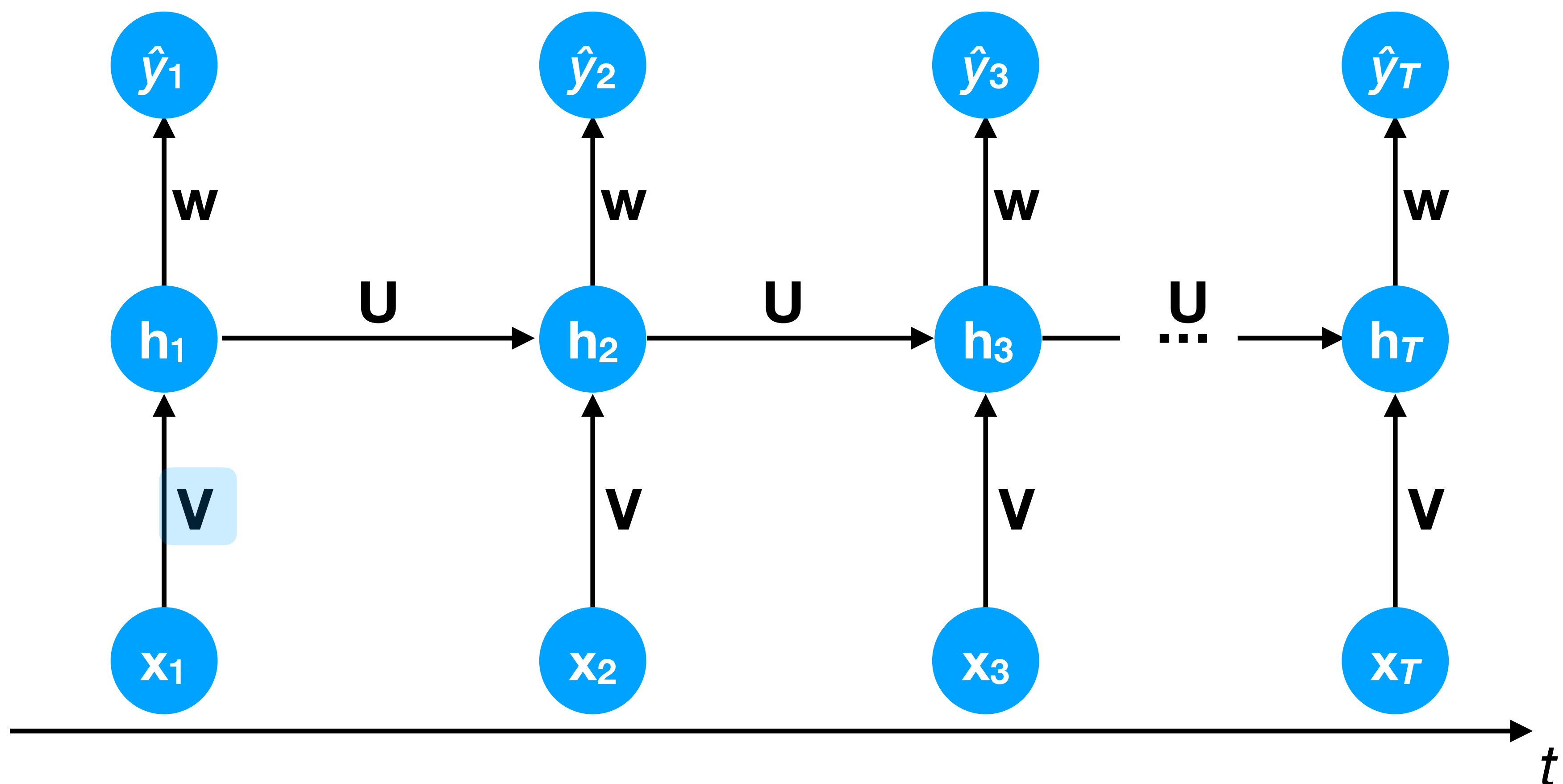
There are "other terms" because V occurs at not just *t*=1.



Difficulty in training RNNs

- When we compute the gradient of $J(y_T, \hat{y}_T)$ w.r.t. **V**, we must sum over all occurrences of **V** —including the *first* one:

$$\frac{\partial J_T(\mathbf{V})}{\partial \mathbf{V}} = \frac{\partial J_T(\mathbf{V})}{\partial \mathbf{h}_T} \mathbf{U}\mathbf{U}\ldots\mathbf{U} \frac{\partial \mathbf{h}_1}{\partial \mathbf{V}}$$



Difficulty in training RNNs

- Note that **U** is square (it maps from one hidden state to another).

- If **U** is diagonalizable, then we can write it as:

$$\mathbf{U} = \mathbf{Q}\mathbf{D}\mathbf{Q}^{-1}$$

  where the columns of **Q** are **U**'s eigenvectors and **D** contains **U**'s the associated eigenvalues.

Difficulty in training RNNs

- Note that **U** is square (it maps from one hidden state to another).

- If **U** is diagonalizable, then we can write it as:

$$\mathbf{U} = \mathbf{Q}\mathbf{D}\mathbf{Q}^{-1}$$

  where the columns of **Q** are **U**'s eigenvectors and **D** contains **U**'s the associated eigenvalues.

- Hence:

$$\overbrace{\mathbf{U}\mathbf{U}\ldots\mathbf{U}}^{\textbf{\textit{k}}\textbf{ terms}} =$$

Difficulty in training RNNs

- Note that **U** is square (it maps from one hidden state to another).

- If **U** is diagonalizable, then we can write it as:

$$\mathbf{U} = \mathbf{Q}\mathbf{D}\mathbf{Q}^{-1}$$

where the columns of **Q** are **U**'s eigenvectors and **D** contains **U**'s the associated eigenvalues.

- Hence:

$$\begin{aligned} \overbrace{\mathbf{U}\mathbf{U}\ldots\mathbf{U}}^{\textbf{\textit{k} terms}} &= \mathbf{Q}\mathbf{D}\mathbf{Q}^{-1}\mathbf{Q}\mathbf{D}\mathbf{Q}^{-1}\ldots\mathbf{Q}\mathbf{D}\mathbf{Q}^{-1} \\ &= \mathbf{Q}\mathbf{D}^k\mathbf{Q}^{-1} \end{aligned}$$

Difficulty in training RNNs

- Plugging back into the gradient:

$$
\begin{aligned}
\frac{\partial J_T(\mathbf{V})}{\partial \mathbf{V}} &= \frac{\partial J_T(\mathbf{V})}{\partial \mathbf{h}_T} \mathbf{U}\mathbf{U} \dots \mathbf{U} \frac{\partial \mathbf{h}_1}{\partial \mathbf{V}} \\
&= \frac{\partial J_T(\mathbf{V})}{\partial \mathbf{h}_T} \mathbf{Q}\mathbf{D}^{T-1}\mathbf{Q}^{-1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{V}} \\
&= \frac{\partial J_T(\mathbf{V})}{\partial \mathbf{h}_T} \mathbf{Q} \begin{bmatrix} \lambda_1^{T-1} & \dots & 0 \\ 0 & \ddots & 0 \\ 0 & \dots & \lambda_m^{T-1} \end{bmatrix} \mathbf{Q}^{-1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{V}}
\end{aligned}
$$

Difficulty in training RNNs

- Plugging back into the gradient:

$$
\begin{aligned}
\frac{\partial J_T(\mathbf{V})}{\partial \mathbf{V}} &= \frac{\partial J_T(\mathbf{V})}{\partial \mathbf{h}_T} \mathbf{U}\mathbf{U} \ldots \mathbf{U} \frac{\partial \mathbf{h}_1}{\partial \mathbf{V}} \\
&= \frac{\partial J_T(\mathbf{V})}{\partial \mathbf{h}_T} \mathbf{Q}\mathbf{D}^{T-1}\mathbf{Q}^{-1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{V}} \\
&= \frac{\partial J_T(\mathbf{V})}{\partial \mathbf{h}_T} \mathbf{Q} \begin{bmatrix} \lambda_1^{T-1} & \ldots & 0 \\ 0 & \ddots & 0 \\ 0 & \ldots & \lambda_m^{T-1} \end{bmatrix} \mathbf{Q}^{-1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{V}}
\end{aligned}
$$

- If $|\lambda_i| < 1$, then $\lambda_i^{T-1} \to 0$ as $T \to \infty$. **Vanishing gradient.**

- If $|\lambda_i| > 1$, then $|\lambda_i^{T-1}| \to \infty$ as $T \to \infty$. **Exploding gradient.**

Difficulty in training RNNs

- Moreover, during forward-propagation:

$$
\begin{aligned}
\hat{y}_T &= \mathbf{w}\mathbf{U}\mathbf{U}\ldots\mathbf{U}\mathbf{x}_1 + \text{other terms} \\
&= \mathbf{w}\mathbf{Q}\mathbf{D}^{T-1}\mathbf{Q}^{-1}\mathbf{x}_1 \\
&= \mathbf{w}\left[\begin{array}{cc} \mathbf{u}_1 & \ldots\mathbf{u}_m \end{array}\right]\left[\begin{array}{ccc} \lambda_1^{T-1} & \ldots & 0 \\ 0 & \ddots & 0 \\ 0 & \ldots & \lambda_m^{T-1} \end{array}\right]\left[\begin{array}{c} \mathbf{u}_1^\top \\ \vdots \\ \mathbf{u}_m^\top \end{array}\right]\mathbf{x}_1
\end{aligned}
$$

where $\mathbf{u}_i$ is the i^{th} eigenvector of $\mathbf{U}$.

- $\hat{y}_T$ loses information from $\mathbf{x}_1$ along direction $\mathbf{u}_i$ unless $|\lambda_i| > 1$.

Difficulty in training deep FFNNs

- Note that the same vanishing/exploding gradient problems can also occur in very deep FFNNs, particularly if the rows of the weight matrices are highly correlated.

Difficulty in training deep FFNNs

- One coping strategy is to **clip** the gradients:
- Suppose we obtain a gradient $\nabla_p J$ for some parameter *p*, and $|\nabla_p J|$ is very large (above some threshold $\boldsymbol{\tau}$).
- Then we can "clip" its value to be:

$$\nabla_p J \left(\frac{\tau}{|\nabla_p J|} \right)$$

Difficulty in training deep FFNNs

- Another strategy for preventing vanishing and exploding gradients is to use **skip connections** (more later)**.**
  - These are used in LSTM and GRU RNNs, as well as ResNet FFNNs.
- Yet another strategy is to restrict **U** to the manifold of **unitary matrices** (i.e., all eigenvalues have magnitude 1; see Helfrich & Ye 2019).

Training RNNs

- Like CNNs, RNNs exhibit an inherently large degree of weight sharing:
  - The **U**, **V** matrices are the same at *all* timesteps.
- See rnn.pdf on Canvas.

Long short-term memory (LSTM) neural networks

LSTMs

- https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs

- Three gates — forget (f), input (i), and output (o).

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_f)$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_i)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_o)$$

LSTMs

- Three gates — forget (f), input (i), and output (o).
- Two state vectors: $\mathbf{h}_t$, $\mathbf{c}_t$.

$$
\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_f) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_i) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_o) \\
\tilde{\mathbf{c}}_t &= \tanh(\mathbf{W}_c[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
$$

LSTMs

- In total, we have 4 weight matrices and 4 bias vectors.

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_f)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_i)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_o)$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_c)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$

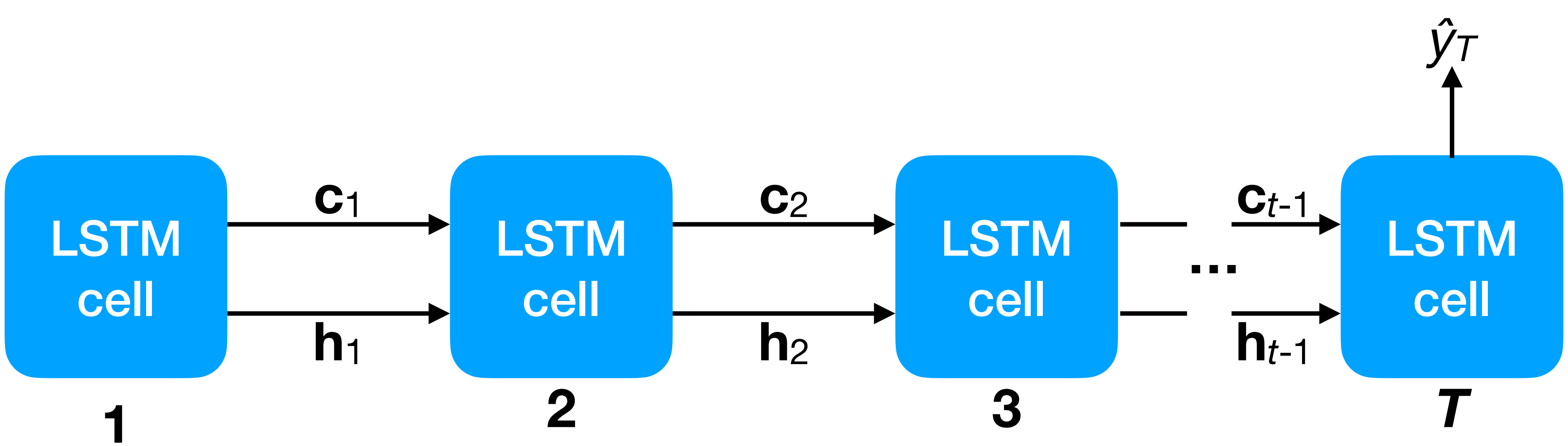
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

LSTM

- The memory cell **c** offers a pathway through the network to preserve information across long time-spans:

$$\mathbf{c}_t = \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{f}_t \odot \mathbf{c}_{t-1}$$

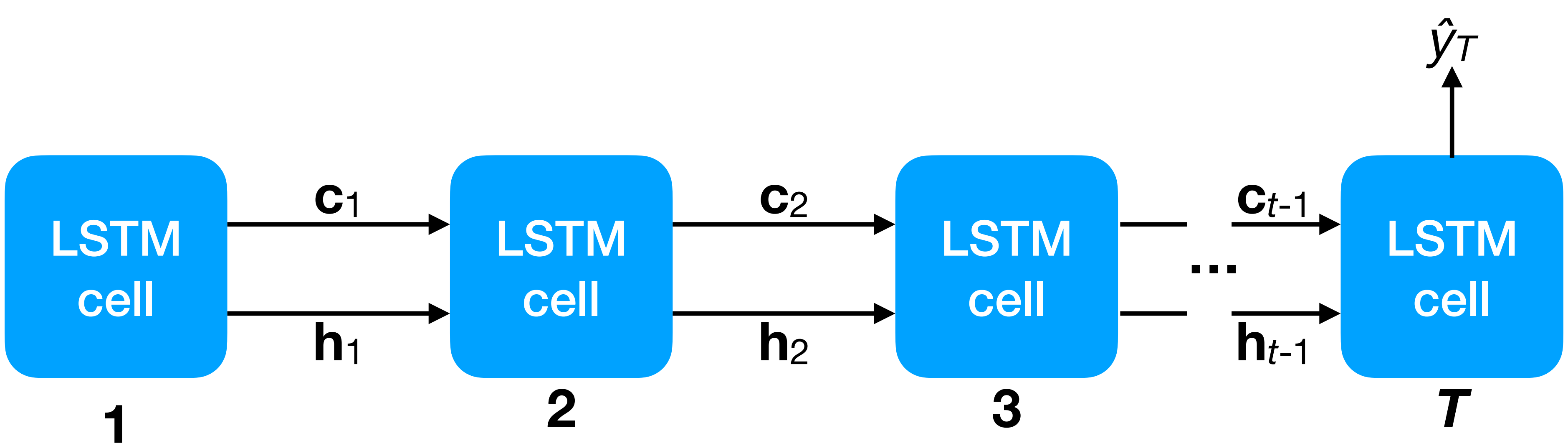
- It tends not to decay due to exponentiated eigenvalues.



LSTM

- If $\mathbf{f}_t$=**1**, then $\mathbf{c}_t$ directly contains information from 1, …, t:

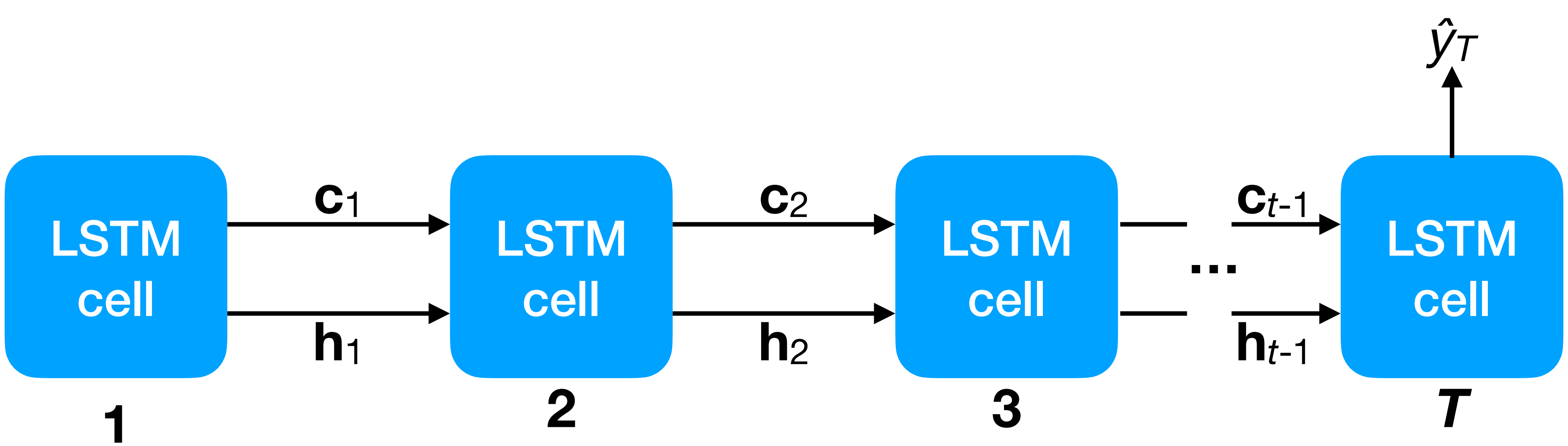
$$\mathbf{c}_t = \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{c}_{t-1}$$



LSTM

- If $\mathbf{f}_t$=**1**, then $\mathbf{c}_t$ directly contains information from 1, …, t:

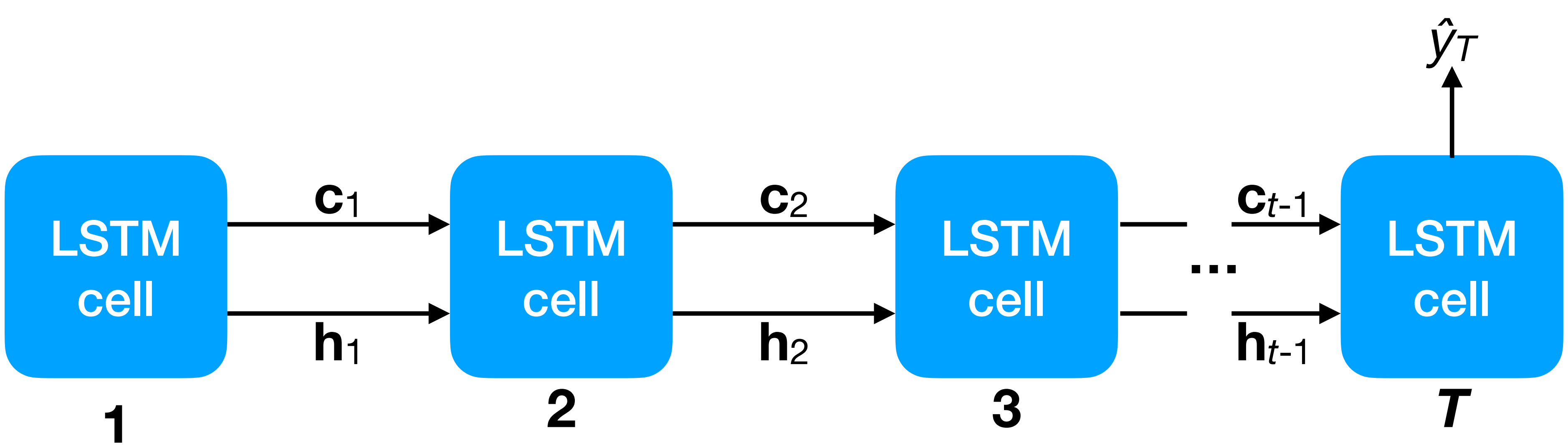
$$
\begin{aligned}
\mathbf{c}_t &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{c}_{t-1} \\
&= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \mathbf{c}_{t-2}
\end{aligned}
$$



LSTM

- If $\mathbf{f}_t$=**1**, then $\mathbf{c}_t$ directly contains information from 1, …, t:

$$\begin{aligned}\mathbf{c}_t &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{c}_{t-1} \\ &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \mathbf{c}_{t-2} \\ &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \mathbf{i}_{t-2} \odot \tilde{\mathbf{c}}_{t-2} + \mathbf{c}_{t-3}\end{aligned}$$



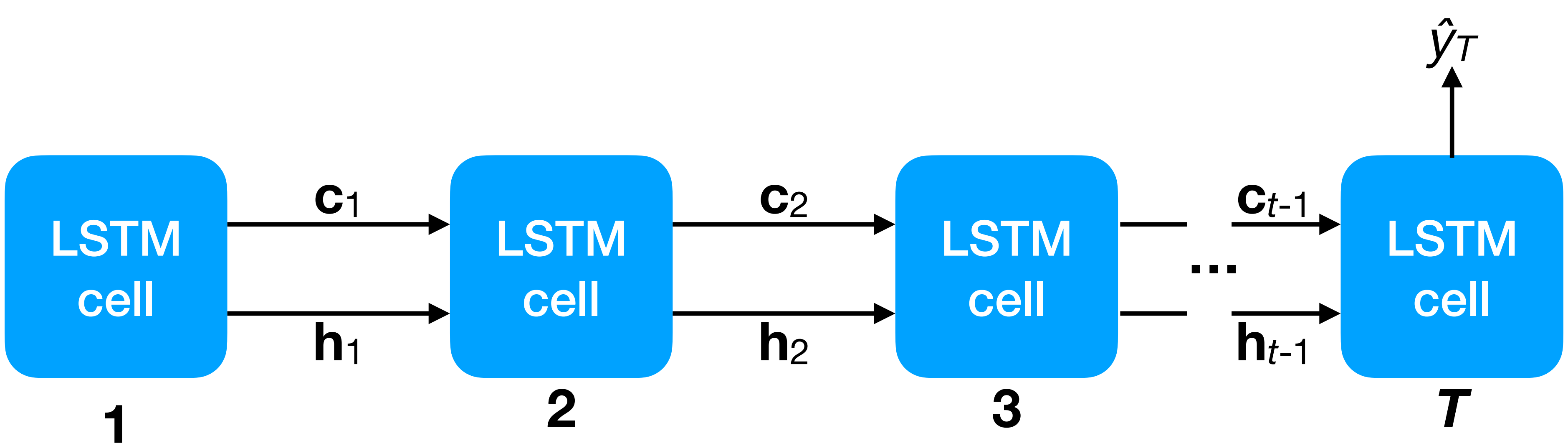
LSTM

- If $\mathbf{f}_t$=1, then $\mathbf{c}_t$ directly contains information from 1, …, t:

$$\begin{aligned}\mathbf{c}_t &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{c}_{t-1} \\ &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \mathbf{c}_{t-2} \\ &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \mathbf{i}_{t-2} \odot \tilde{\mathbf{c}}_{t-2} + \mathbf{c}_{t-3}\end{aligned}$$

$$\ldots$$

$$\mathbf{c}_t = \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \ldots + \mathbf{i}_2 \odot \tilde{\mathbf{c}}_2 + \mathbf{c}_1$$



LSTM

- If $\mathbf{f}_t$=**1**, then $\mathbf{c}_t$ directly contains information from 1, …, t:

$$\begin{aligned}\mathbf{c}_t &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{c}_{t-1} \\ &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \mathbf{c}_{t-2} \\ &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \mathbf{i}_{t-2} \odot \tilde{\mathbf{c}}_{t-2} + \mathbf{c}_{t-3}\end{aligned}$$

$$\ldots$$

$$\mathbf{c}_t = \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \ldots + \mathbf{i}_2 \odot \tilde{\mathbf{c}}_2 + \mathbf{c}_1$$