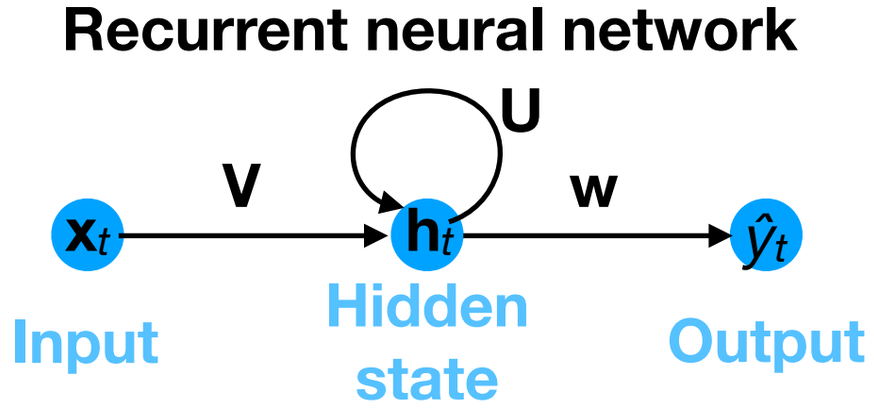
CS/DS 541: Class 13

Jacob Whitehill

Recurrent neural networks (RNNs)

Recurrent neural network

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

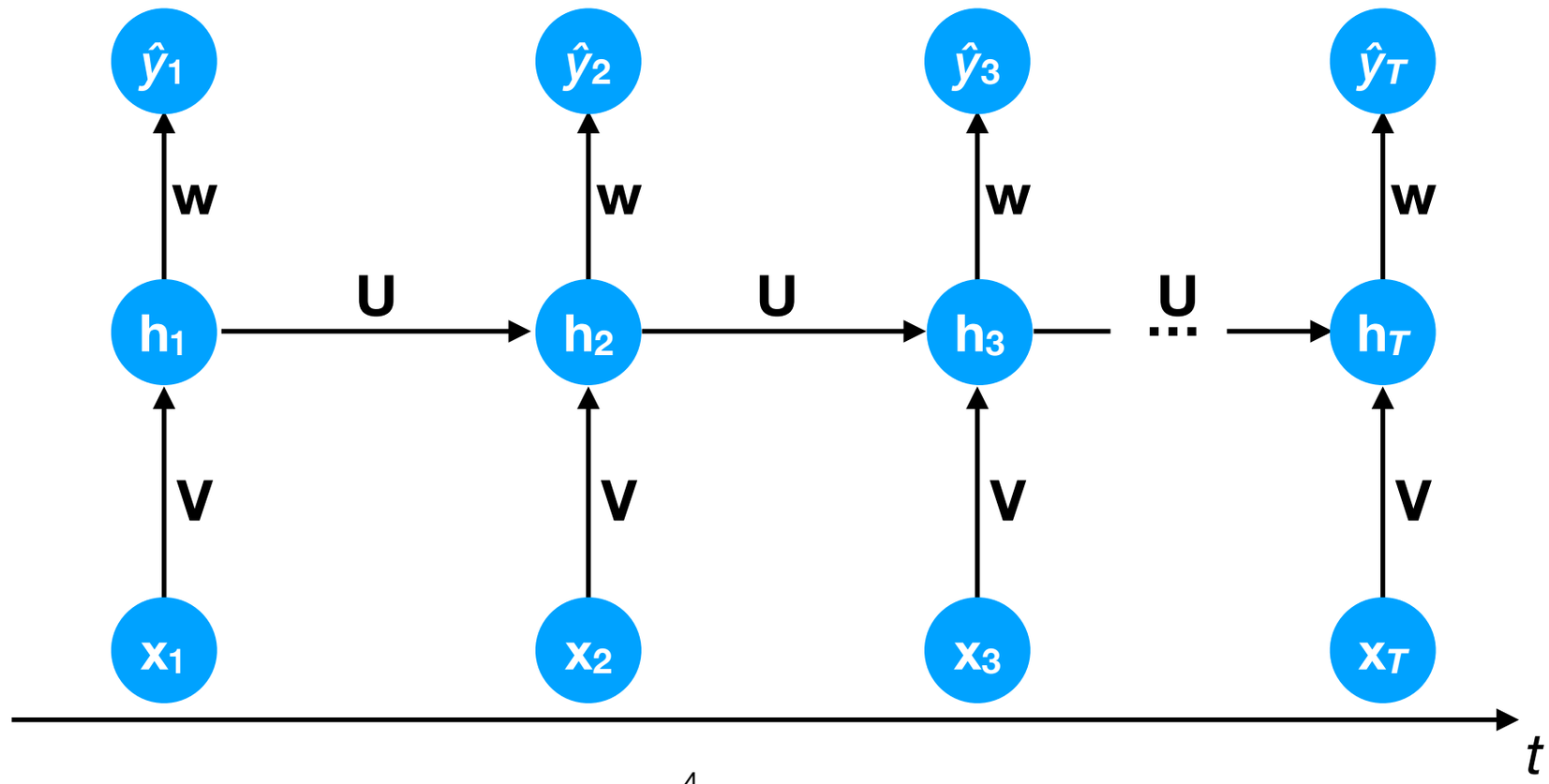


$$\hat{y}_t = g(\mathbf{x}_1, \ldots, \mathbf{x}_t; \mathbf{U}, \mathbf{V}, \mathbf{w}) = \mathbf{h}_t^\top \mathbf{w}$$

$$\mathbf{h}_t = \sigma\left(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{V}\mathbf{x}_t\right)$$

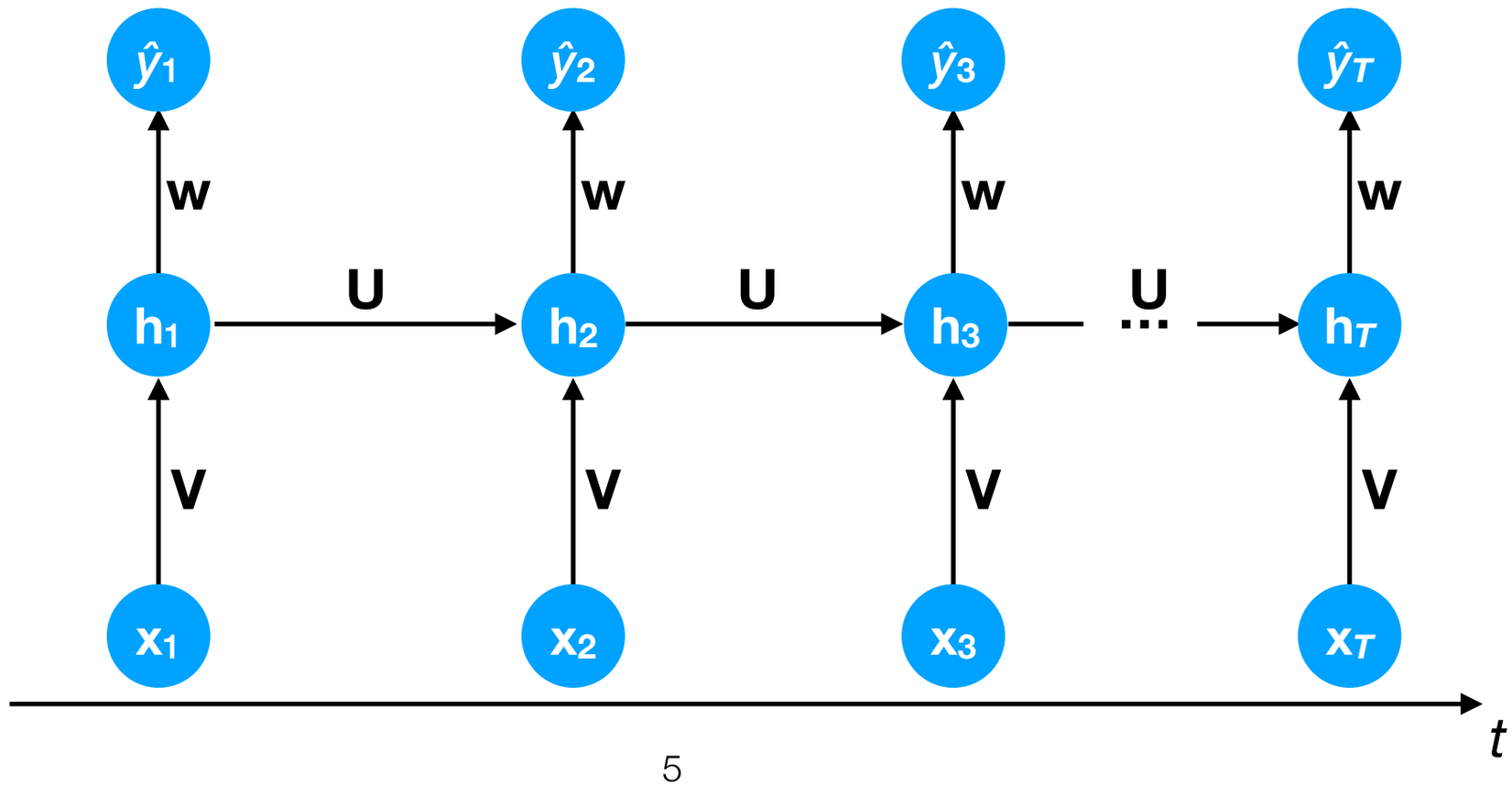
Difficulty in training RNNs

- 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).



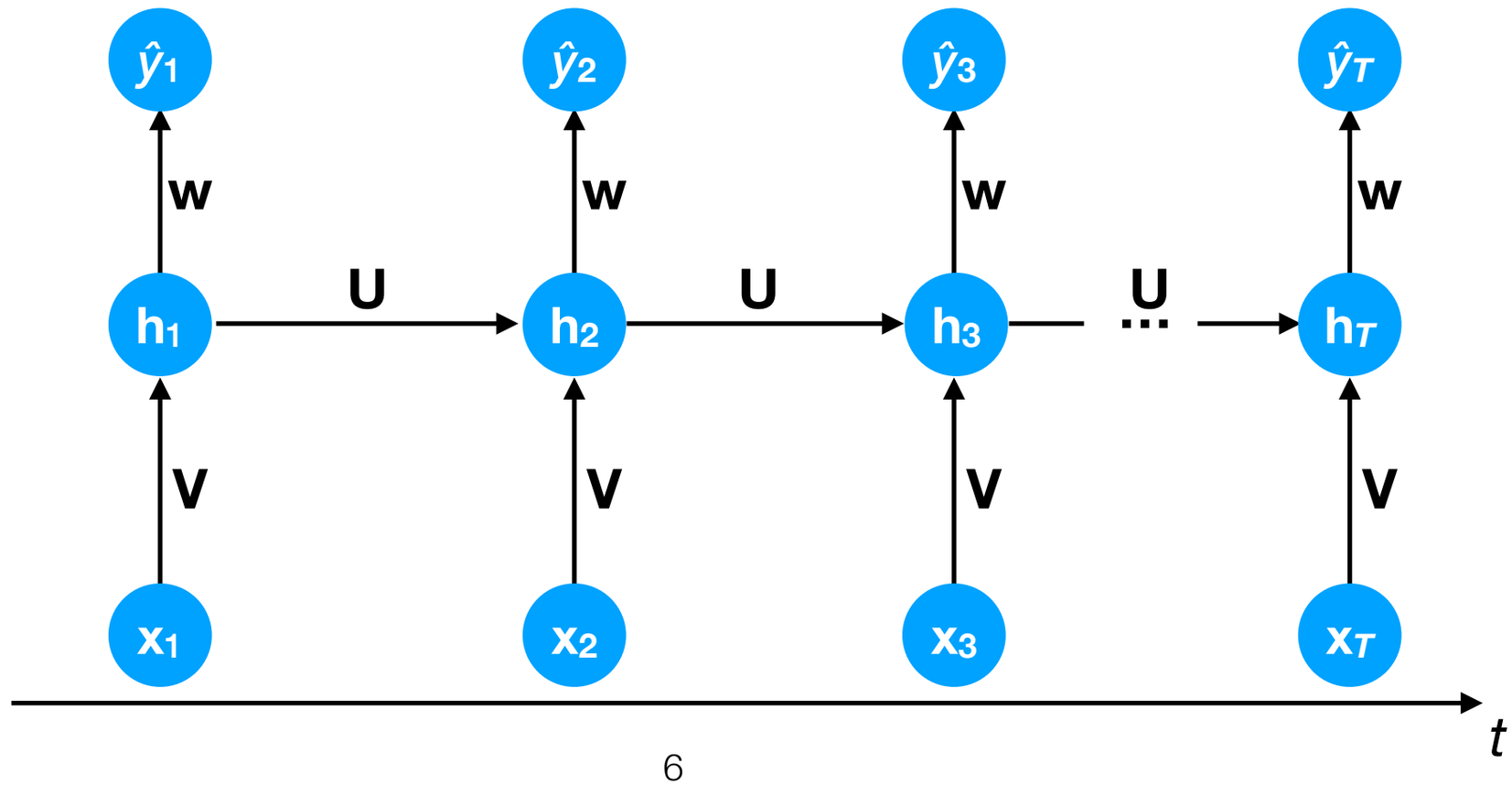
Difficulty in training RNNs

- 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

- In a linear RNN, the forward-propagation yields:

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

  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

- Another strategy (besides clipping gradients) 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).

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.

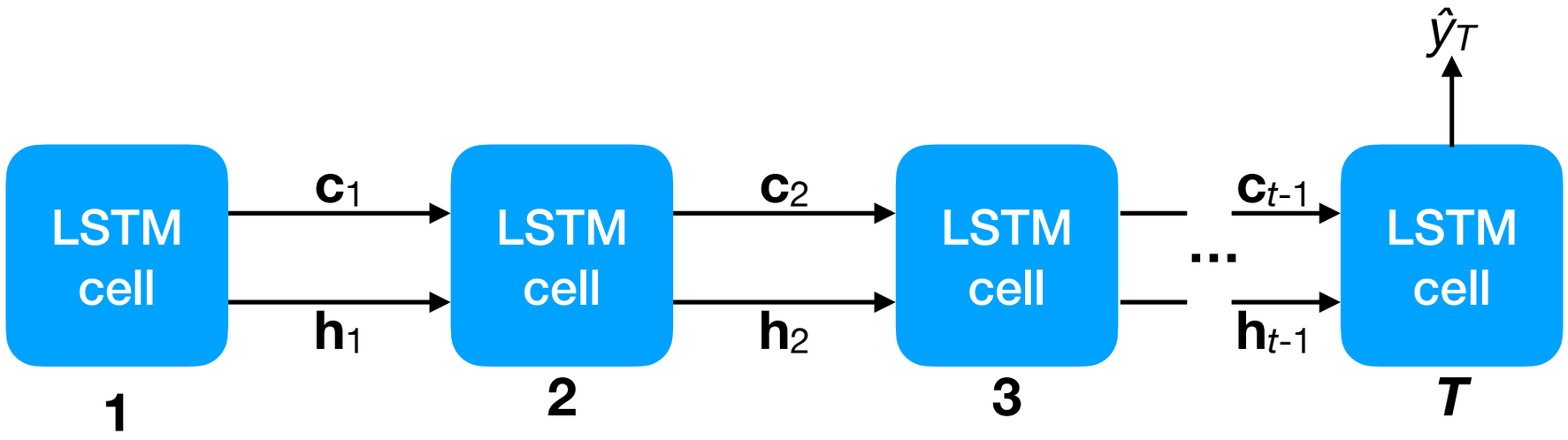
$$
\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_f) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_i) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_o) \\
\tilde{\mathbf{c}}_t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

LSTM

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

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

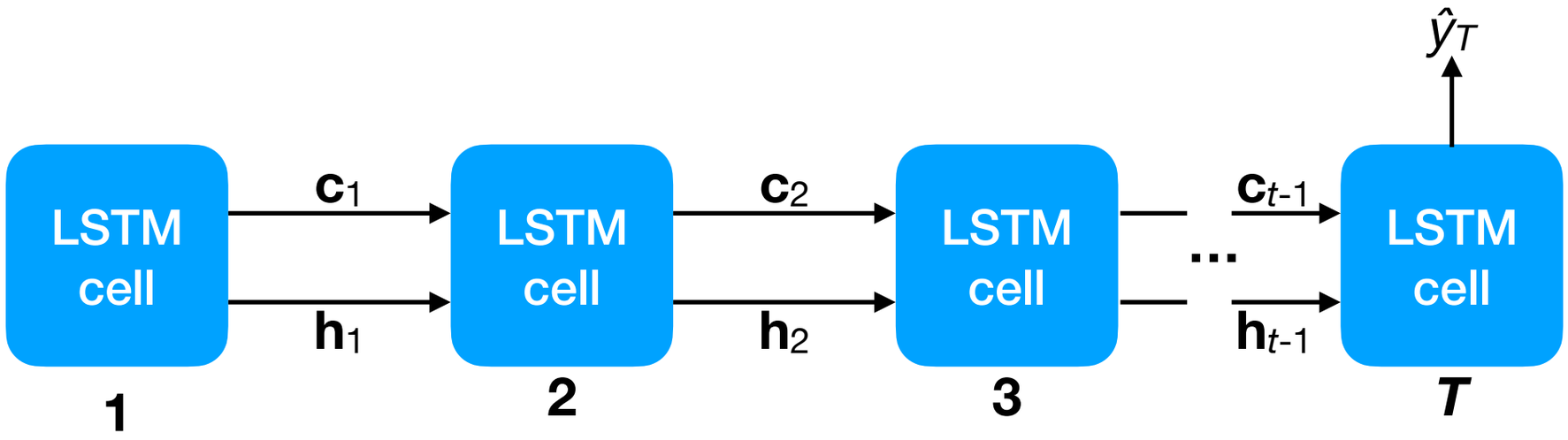
- It tends not to decay due to exponentiated eigenvalues.



LSTM

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

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

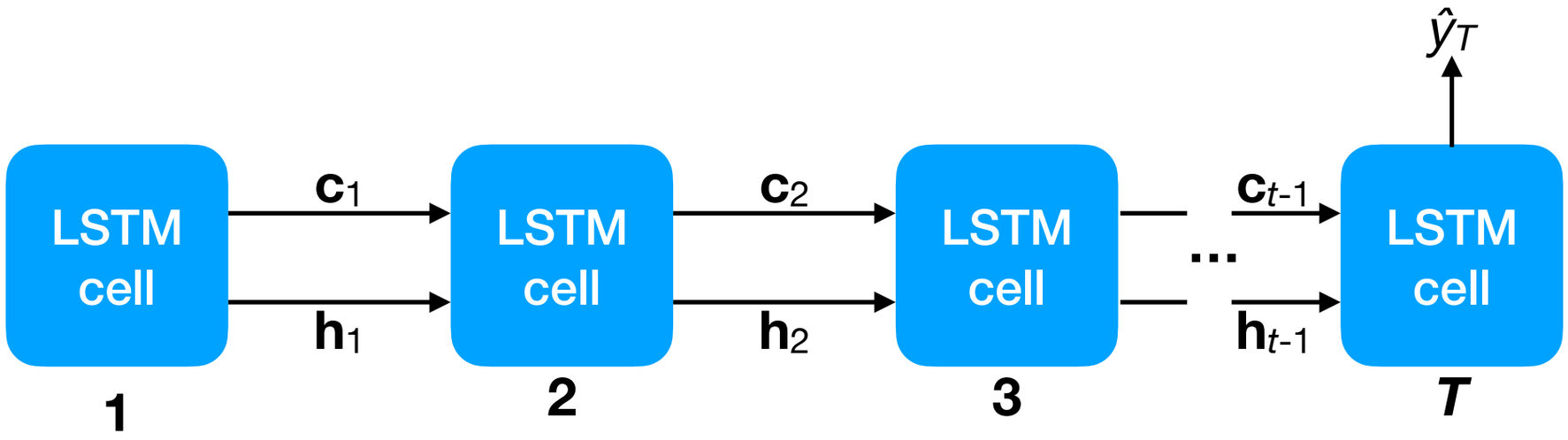


LSTM

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

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

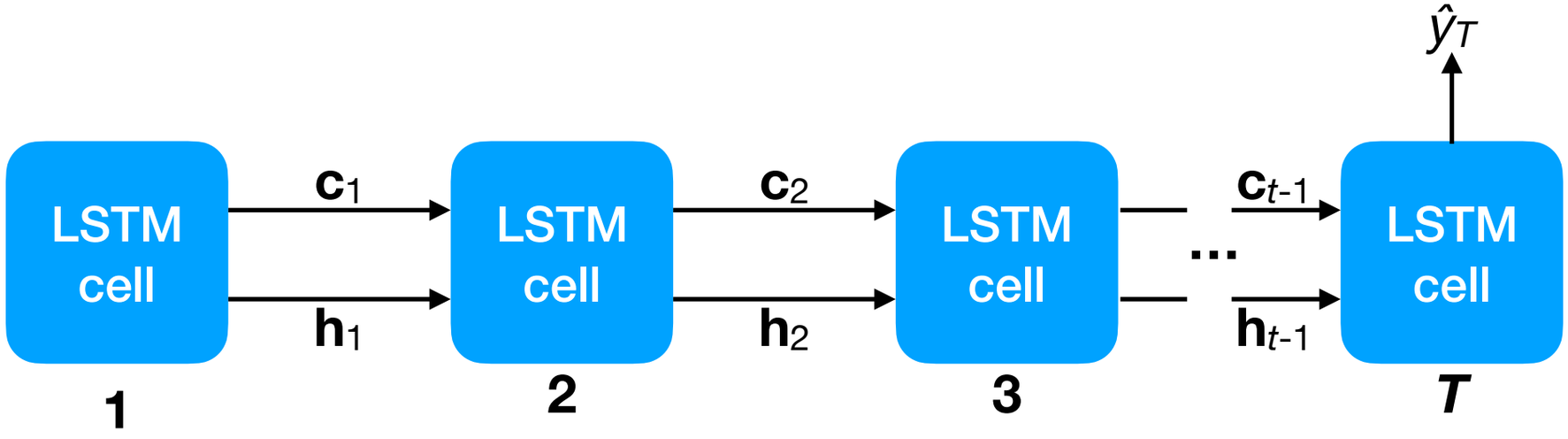
$$= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \mathbf{c}_{t-2}$$



LSTM

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

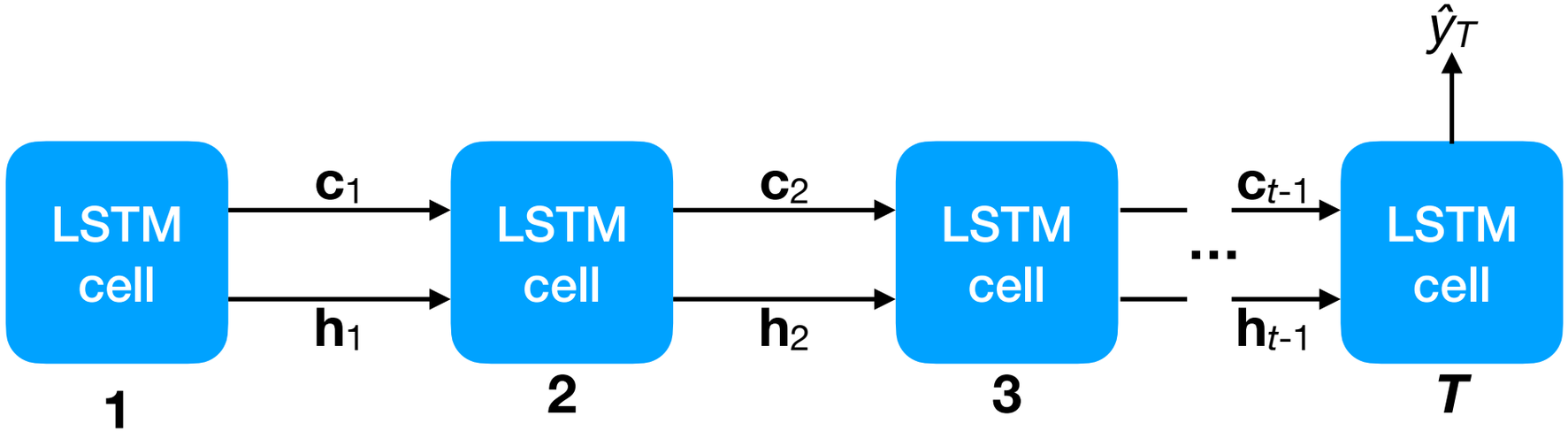
$$\begin{aligned}\mathbf{c}_t &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{c}_{t-1} \\ &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \mathbf{c}_{t-2} \\ &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1}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



LSTM

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

$$\begin{aligned}\mathbf{c}_t &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{c}_{t-1} \\ &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \mathbf{c}_{t-2} \\ &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \mathbf{i}_{t-2} \odot \tilde{\mathbf{c}}_{t-2} + \mathbf{c}_{t-3} \\ &\dots \\ \mathbf{c}_t &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \ldots + \mathbf{i}_2 \odot \tilde{\mathbf{c}}_2 + \mathbf{c}_1\end{aligned}$$



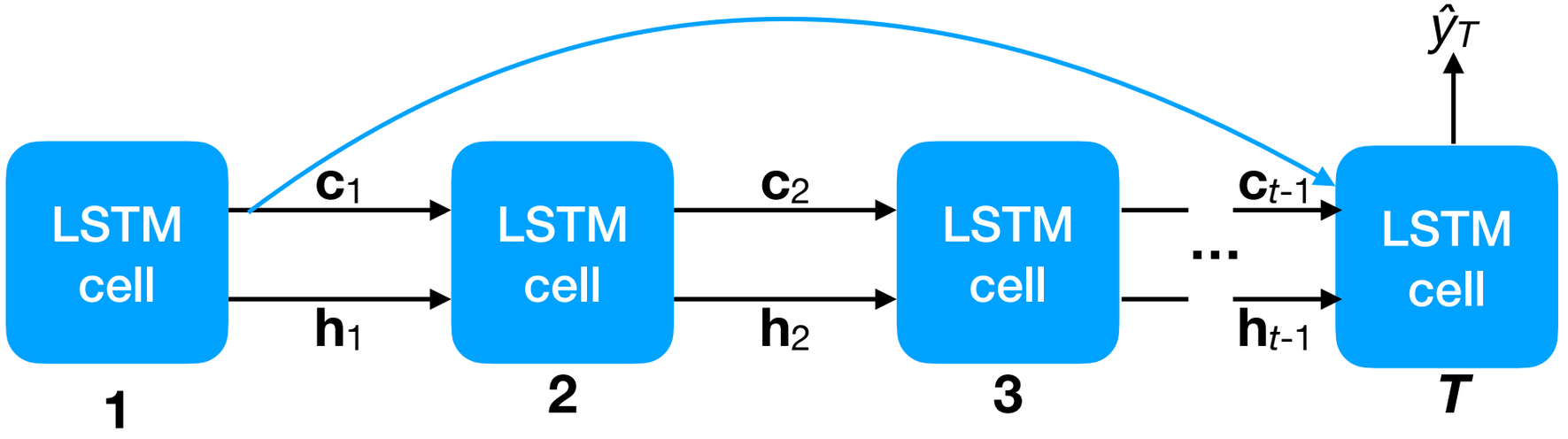
LSTM

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

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

$$\ldots$$

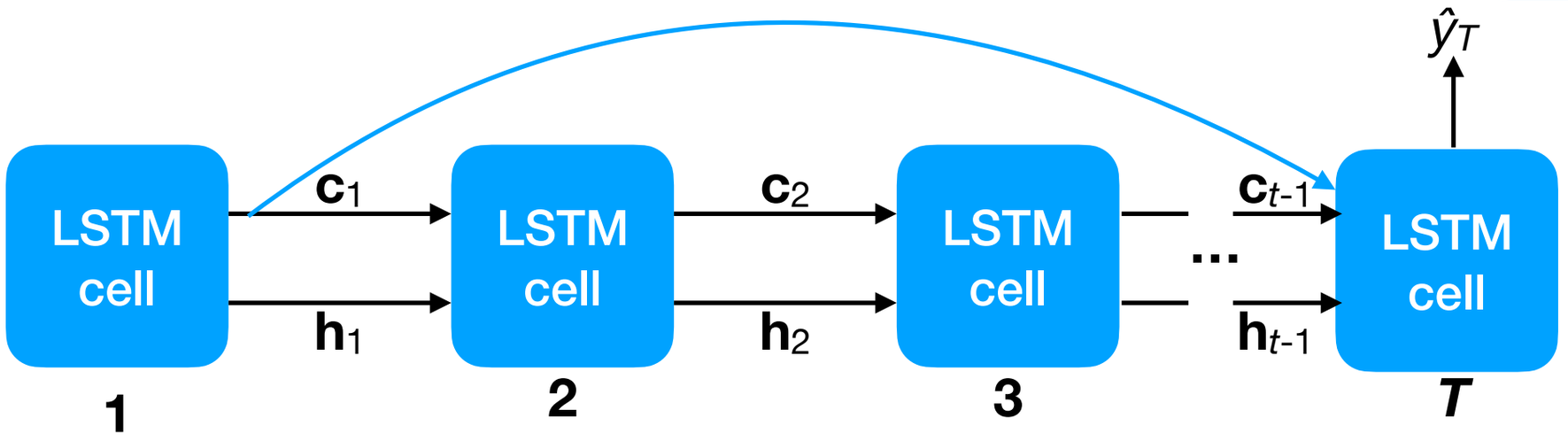
$$\mathbf{c}_t = \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \ldots + \mathbf{i}_2 \odot \tilde{\mathbf{c}}_2 + \mathbf{c}_1$$



LSTM

- This is sometimes called a **skip connection**.

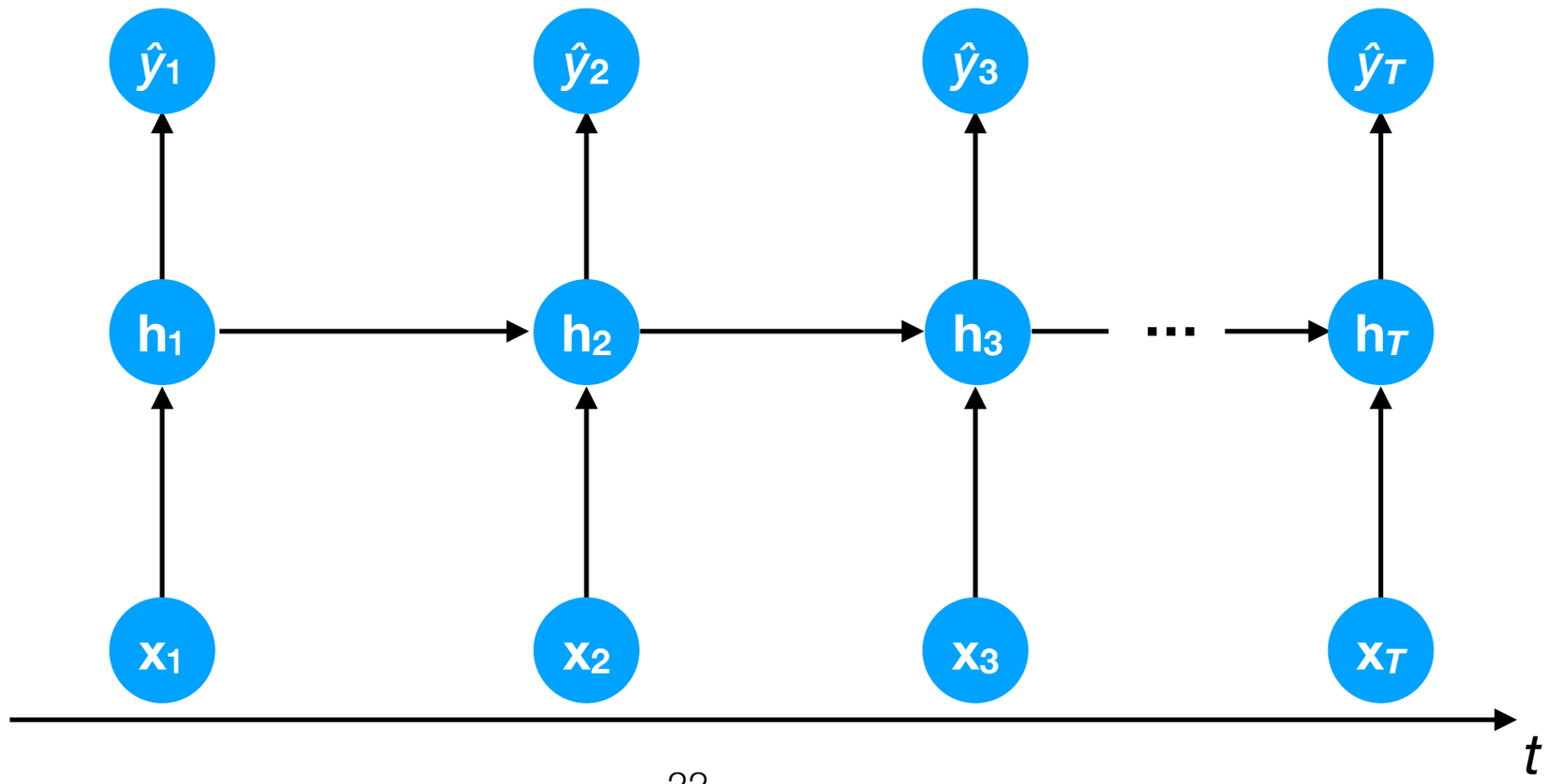
$$\mathbf{c}_t = \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \ldots + \mathbf{i}_2 \odot \tilde{\mathbf{c}}_2 + \mathbf{c}_1$$



Bi-directional RNNs

Bi-directional RNNs

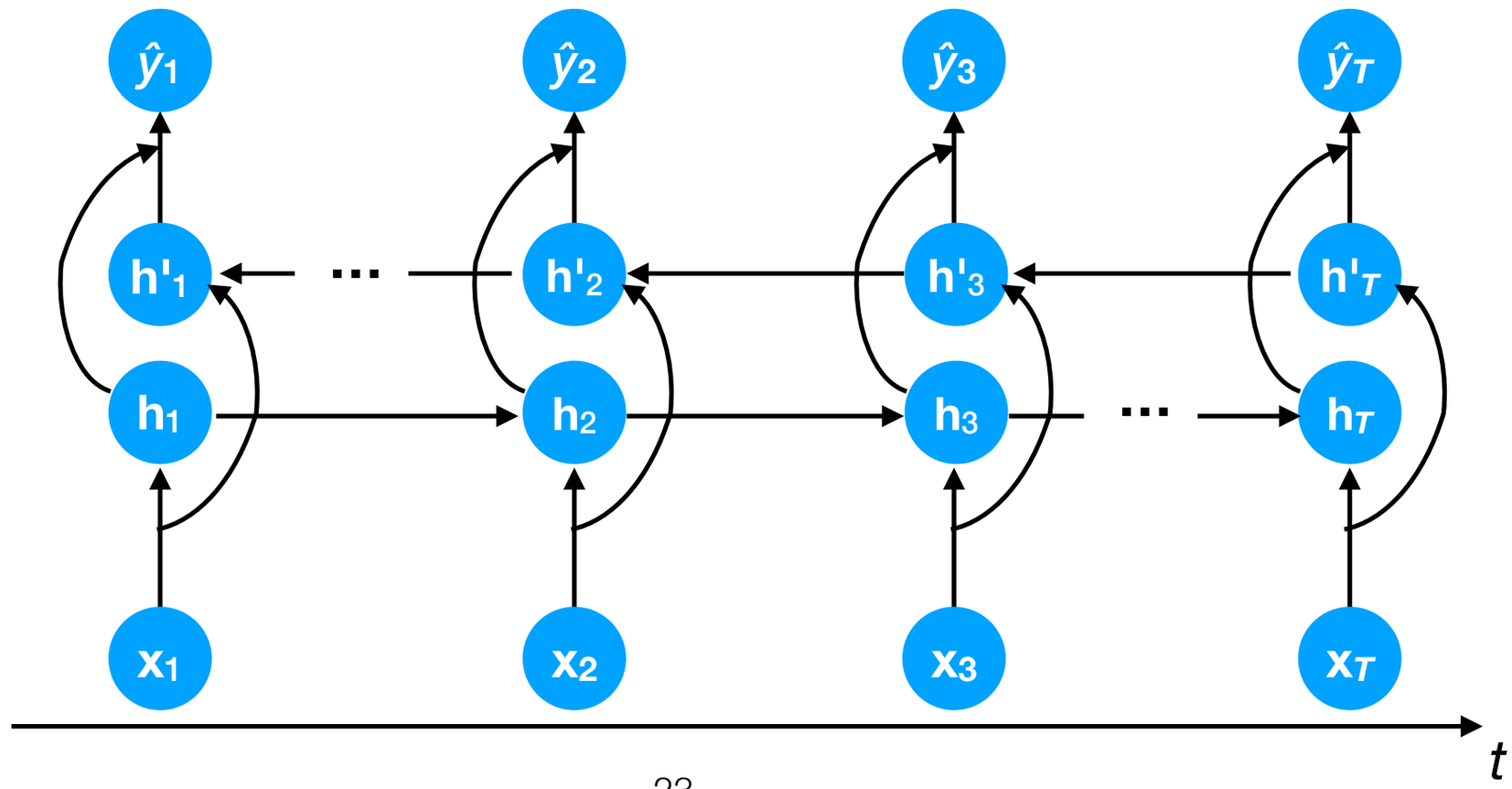
- The RNNs (including LSTMs) we have examined so far are useful when an output $\hat{y}_t$ must be estimated immediately after t timesteps.



Bi-directional RNNs

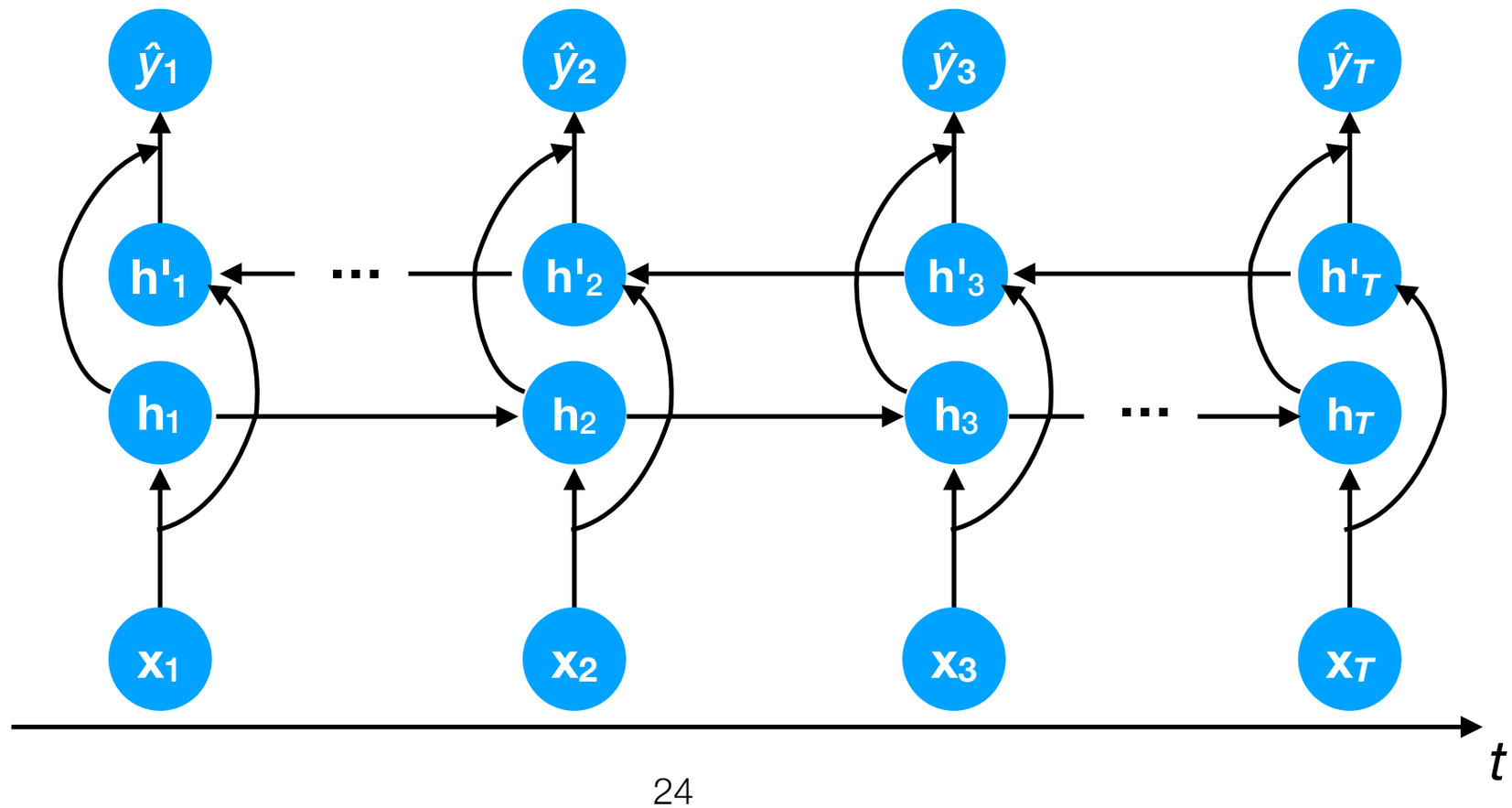
- However, in some settings, we may be able to wait to see the *entire* input sequence before producing any outputs.

- In this case, it can help to harness the entire sequence $x_1, \ldots, x_T$ for each prediction $\hat{y}_t$.



Bi-directional RNNs

- With a bi-directional RNN, each prediction $\hat{y}_t$ is determined by two different hidden representations $\mathbf{h}_t$, $\mathbf{h}'_t$ — one from each direction of processing.



Regularization

Regularization

- We (like in Goodfellow's *Deep Learning*) can define **regularization** as anything that helps to improve generalization performance of a trained ML model.

- Deep learning benefits from many standard techniques (e.g., L_1, L_2 regularization) but also offers some of its own.

L_2 regularization

- We have already seen L_2 regularization, whereby the L_2 norm of each (vectorized) weight matrix is added to the loss, e.g.:

$$f(\hat{\mathbf{y}}, \mathbf{y}; \{\mathbf{W}^{(k)}, \mathbf{b}^{(k)}\}_{k=1}^{l}) = \frac{1}{2}(\hat{\mathbf{y}} - \mathbf{y})^\top(\hat{\mathbf{y}} - \mathbf{y}) + \sum_{k=1}^{l} \frac{\alpha_k}{2} \|\mathbf{W}^{(k)}\|_{\mathrm{Fr}}^2$$

- The Frobenius norm of a matrix is the sum of its **squared** entries; it is equivalent to the L_2 norm of the vectorized matrix. Gradient: $\nabla_{\mathbf{W}} \left(\frac{1}{2} \|\mathbf{W}\|_{\mathrm{Fr}}^2 \right) = \mathbf{W}$
- The L_2 norm encourages *all* the entries in each weight matrix to be small.

L_2 regularization

- We can apply different amounts of regularization to each matrix $\mathbf{W}^{(k)}$.

$$f(\hat{\mathbf{y}}, \mathbf{y}; \{\mathbf{W}^{(k)}, \mathbf{b}^{(k)}\}_{k=1}^{l}) = \frac{1}{2}(\hat{\mathbf{y}} - \mathbf{y})^\top(\hat{\mathbf{y}} - \mathbf{y}) + \sum_{k=1}^{l} \frac{\alpha_k}{2} \|\mathbf{W}^{(k)}\|_{\text{Fr}}^2$$

- Bias terms are typically not regularized because we want them to “shift” the activations as much as needed.

L_1 regularization

- A related technique is L_1 regularization, which penalizes the sum of the **absolute values** of each entry of a weight matrix, e.g.:

$$f(\hat{\mathbf{y}}, \mathbf{y}; \{\mathbf{W}^{(k)}, \mathbf{b}^{(k)}\}_{k=1}^{l}) = \frac{1}{2}(\hat{\mathbf{y}} - \mathbf{y})^\top(\hat{\mathbf{y}} - \mathbf{y}) + \sum_{k=1}^{l} \alpha_k \|\mathbf{W}^{(k)}\|_1$$

- The L_1 norm encourages *some* parameters to be *exactly 0*. This can encourage sparse feature representations. Gradient:

$$\nabla_{\mathbf{W}} \left(\|\mathbf{W}\|_1\right) = \text{sign}(\mathbf{W})$$

- Note that L_1 and L_2 regularization can also be combined.

L_2 regularization = weight decay

- You may sometimes encounter the term **weight decay**, which means that weights tend to "decay" in magnitude during training.

- Weight decay is equivalent to L_2 regularization in SGD:

$$\mathbf{W}^{\text{new}} = \mathbf{W} - \epsilon(\nabla_{\mathbf{W}} f(\mathbf{W}) + \alpha \mathbf{W})$$

L_2 regularization = weight decay

- You may sometimes encounter the term **weight decay**, which means that weights tend to "decay" in magnitude during training.

- Weight decay is equivalent to L_2 regularization in SGD:

$$\begin{aligned}\mathbf{W}^{\text{new}} &= \mathbf{W} - \epsilon(\nabla_{\mathbf{W}} f(\mathbf{W}) + \alpha \mathbf{W}) \\ &= \mathbf{W} - \epsilon \nabla_{\mathbf{W}} f(\mathbf{W}) - \epsilon \alpha \mathbf{W}\end{aligned}$$

L_2 regularization = weight decay

- You may sometimes encounter the term **weight decay**, which means that weights tend to "decay" in magnitude during training.

- Weight decay is equivalent to L_2 regularization in SGD:

$$\begin{aligned}\mathbf{W}^{\text{new}} &= \mathbf{W} - \epsilon(\nabla_{\mathbf{W}} f(\mathbf{W}) + \alpha \mathbf{W}) \\ &= \mathbf{W} - \epsilon \nabla_{\mathbf{W}} f(\mathbf{W}) - \epsilon \alpha \mathbf{W} \\ &= \mathbf{W}(1 - \epsilon\alpha) - \epsilon \nabla_{\mathbf{W}} f(\mathbf{W})\end{aligned}$$

L_2 regularization = weight decay

- You may sometimes encounter the term **weight decay**, which means that weights tend to "decay" in magnitude during training.

- Weight decay is equivalent to L_2 regularization in SGD:

$$\begin{aligned}\mathbf{W}^{\text{new}} &= \mathbf{W} - \epsilon(\nabla_{\mathbf{W}} f(\mathbf{W}) + \alpha \mathbf{W}) \\ &= \mathbf{W} - \epsilon \nabla_{\mathbf{W}} f(\mathbf{W}) - \epsilon \alpha \mathbf{W} \\ &= \mathbf{W}(1 - \epsilon\alpha) - \epsilon \nabla_{\mathbf{W}} f(\mathbf{W})\end{aligned}$$

- For $\varepsilon\alpha < 1$, $\mathbf{W}$ shrinks in length at each iteration.

L_2 regularization ≅ Gaussian noise augmentation

- For 2-layer linear NNs (i.e., linear regression), L_2 regularization is also equivalent to augmenting the training set by adding element-wise Gaussian noise to each input.

- To show this, we will use a probabilistic interpretation.
- Let $\mathbf{x} \in \mathbb{R}^m$ be a randomly drawn training input and (scalar) *y* is its associated label.
- Let $\mathbf{n} \in \mathbb{R}^m,\ \mathbf{n} \sim \mathcal{N}(\mathbf{0}, \alpha\mathbf{I})$ be 0-mean Gaussian noise that is *independent* of **x**.
- Recall that, for any two independent random variables **x** and **n**, we have: $\mathbb{E}[\mathbf{xn}] = \mathbb{E}[\mathbf{x}]\mathbb{E}[\mathbf{n}]$
- Define $\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{n}$.