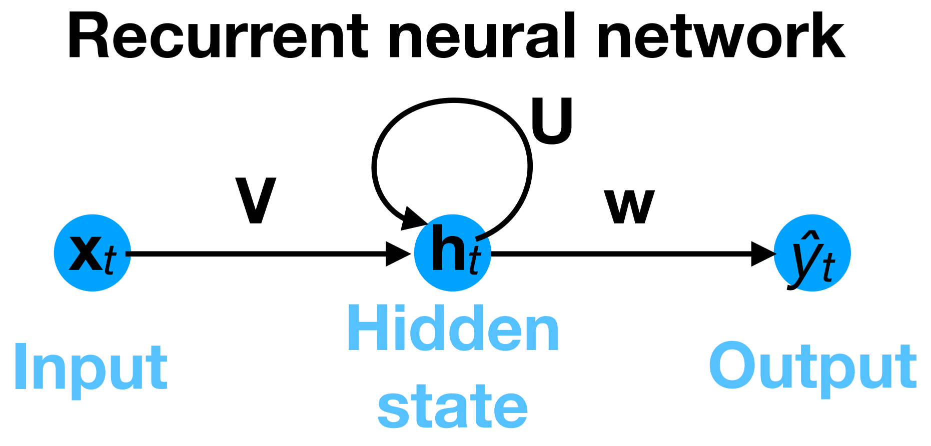
CS/DS 541: Class 13

Jacob Whitehill

Recurrent neural networks (RNNs)

Recurrent neural network

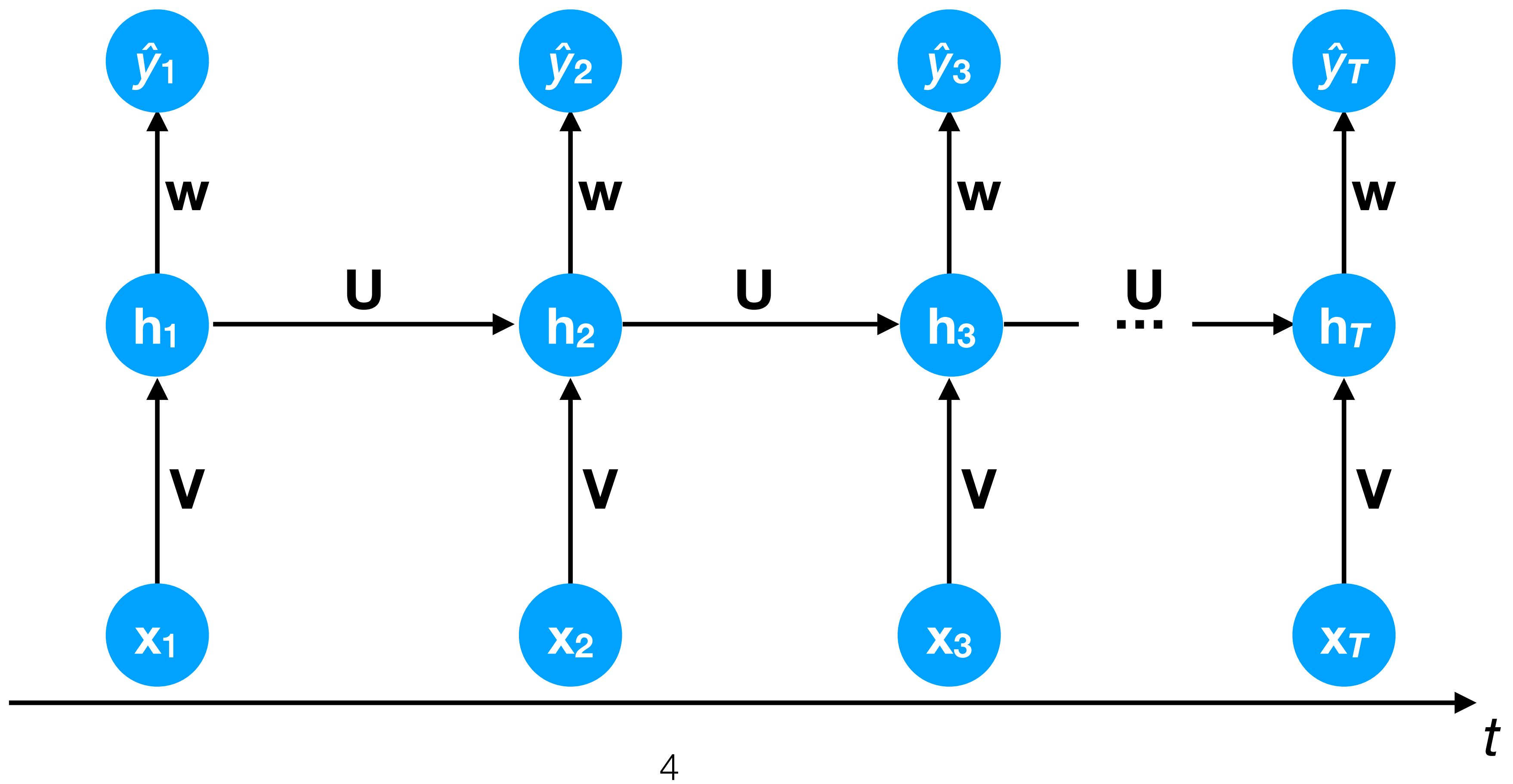
- We can construct a simple **recurrent neural network** (RNN) as follows:



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

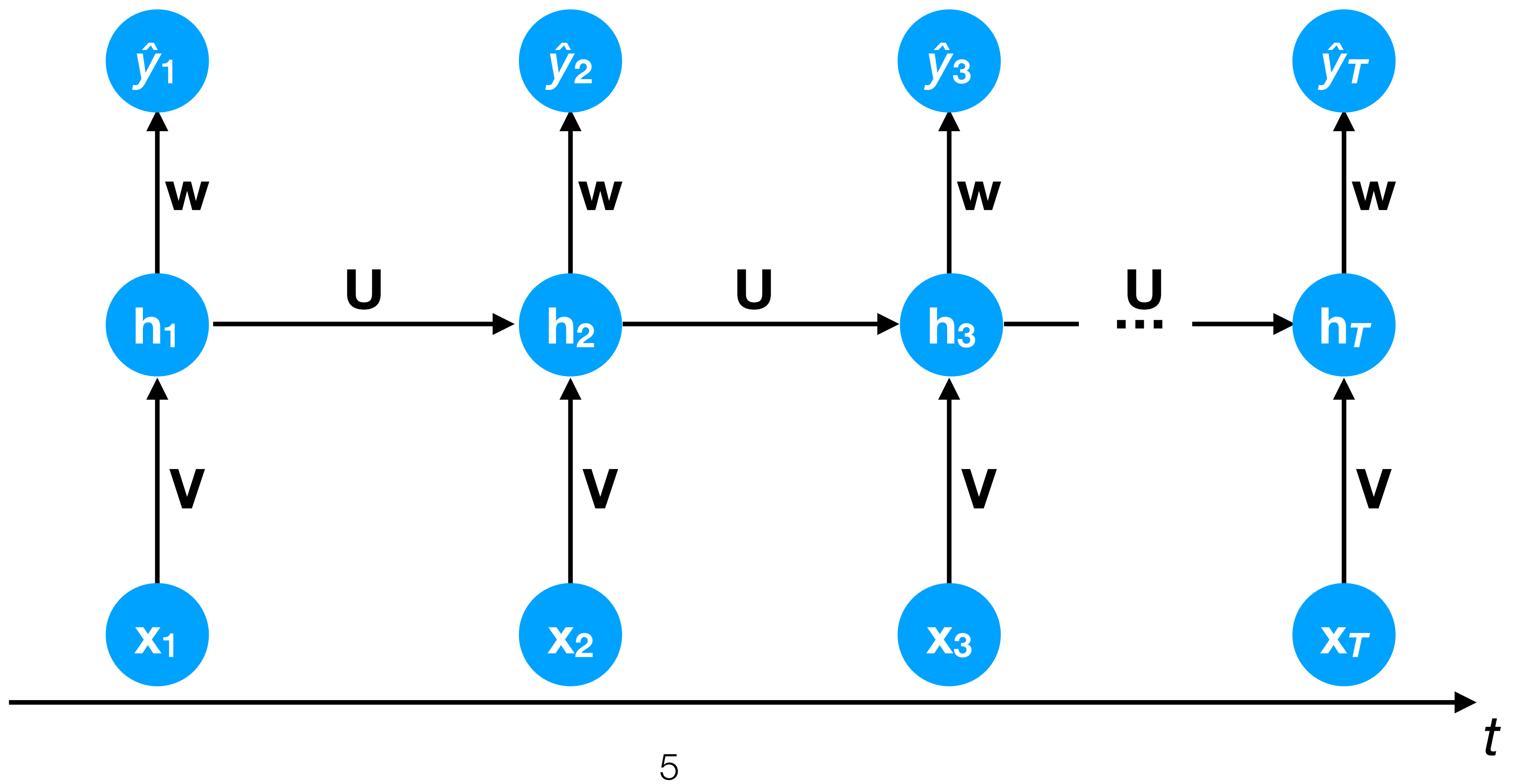
Difficulty in training RNNs

- 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).



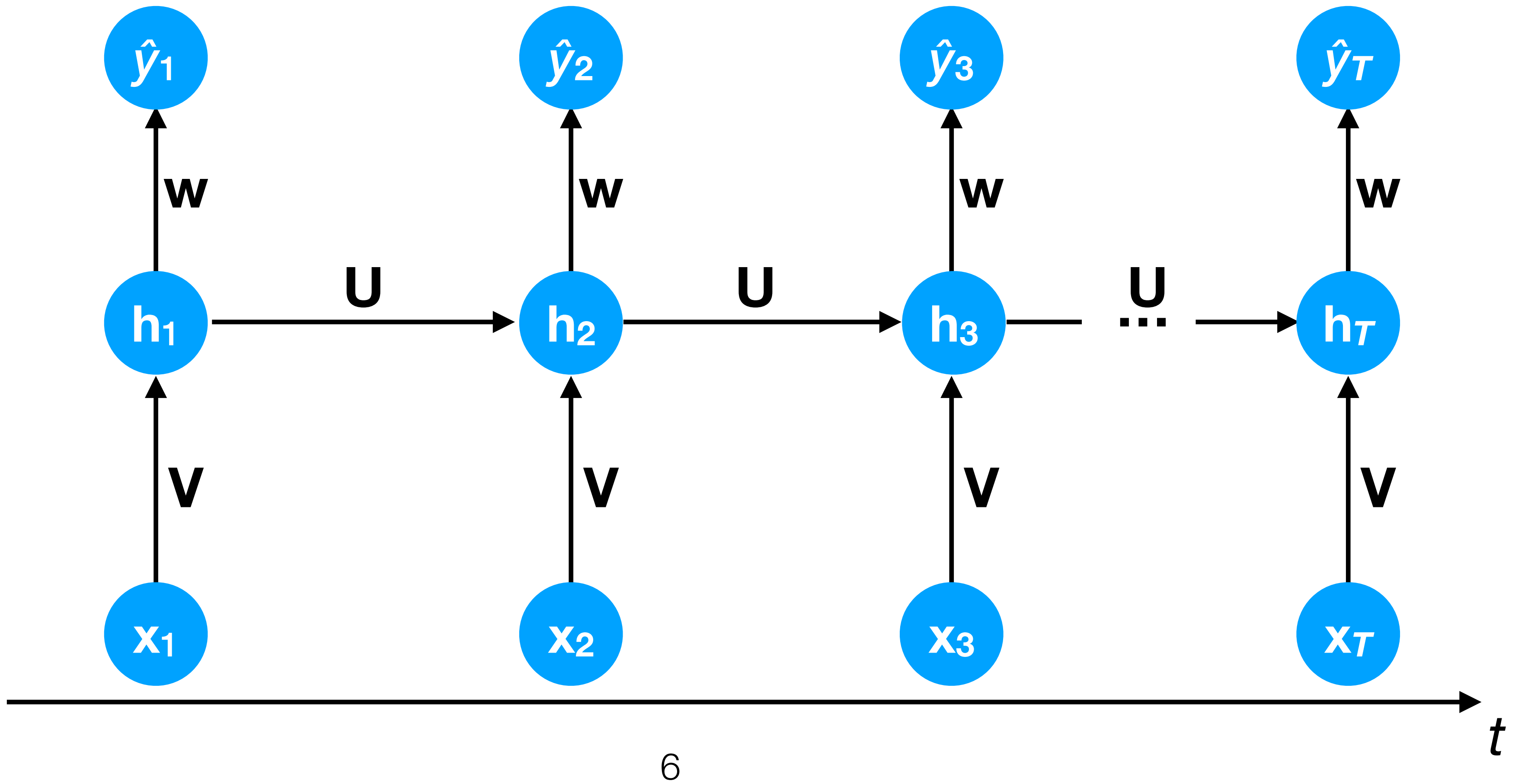
Difficulty in training RNNs

- 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 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).

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.

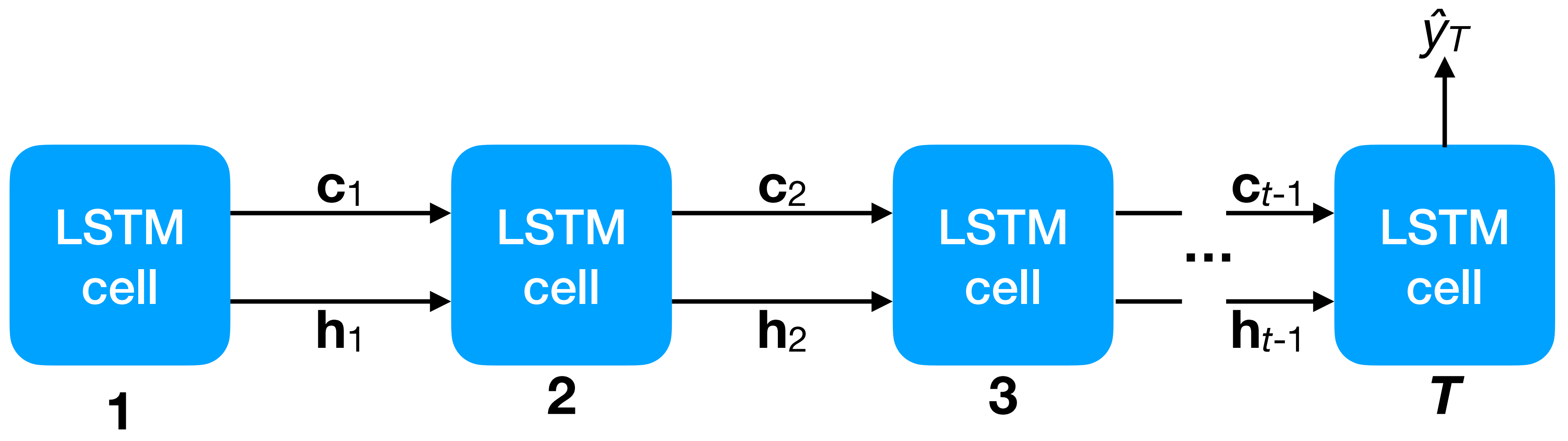
$$
\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_f) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_i) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_o) \\
\tilde{\mathbf{c}}_t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

LSTM

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

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

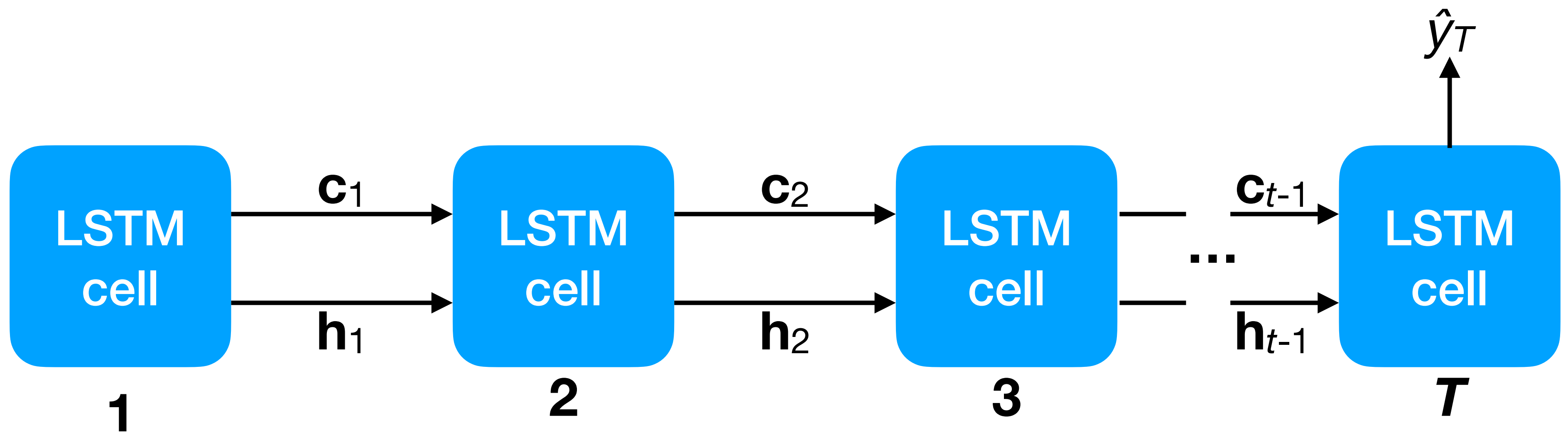
- It tends not to decay due to exponentiated eigenvalues.



LSTM

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

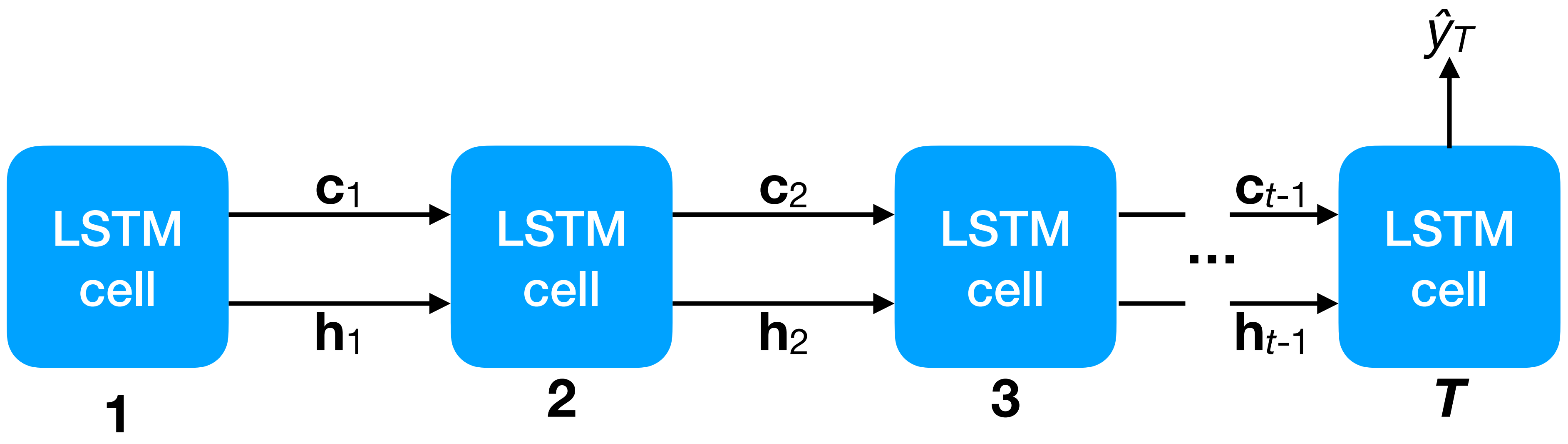
$$\mathbf{c}_t = \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{c}_{t-1}$$



LSTM

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

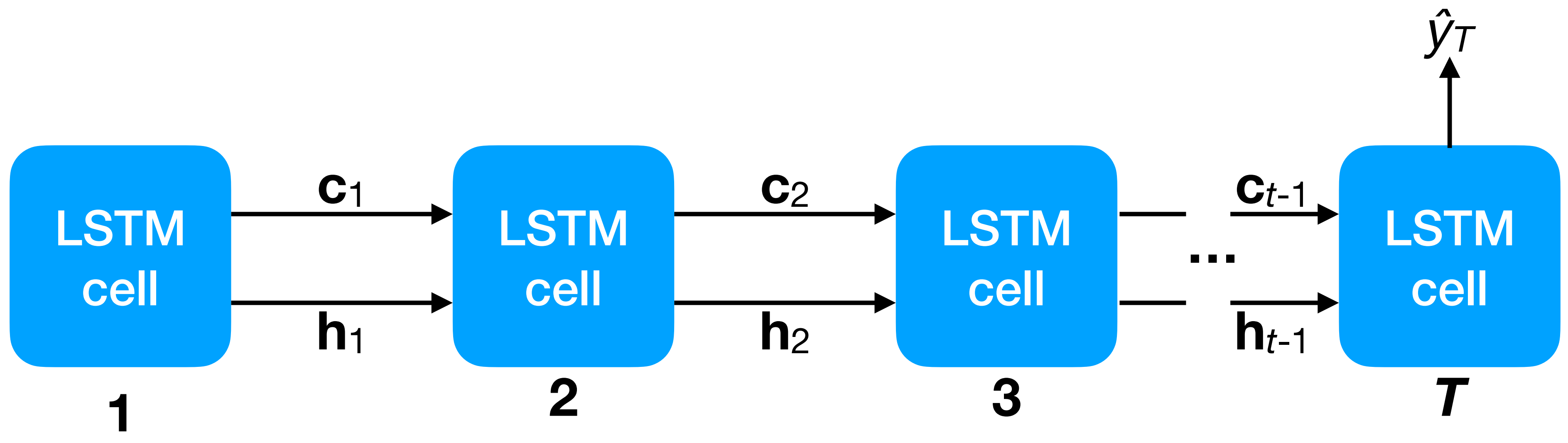
$$\begin{aligned}\mathbf{c}_t &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{c}_{t-1} \\ &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \mathbf{c}_{t-2}\end{aligned}$$



LSTM

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

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



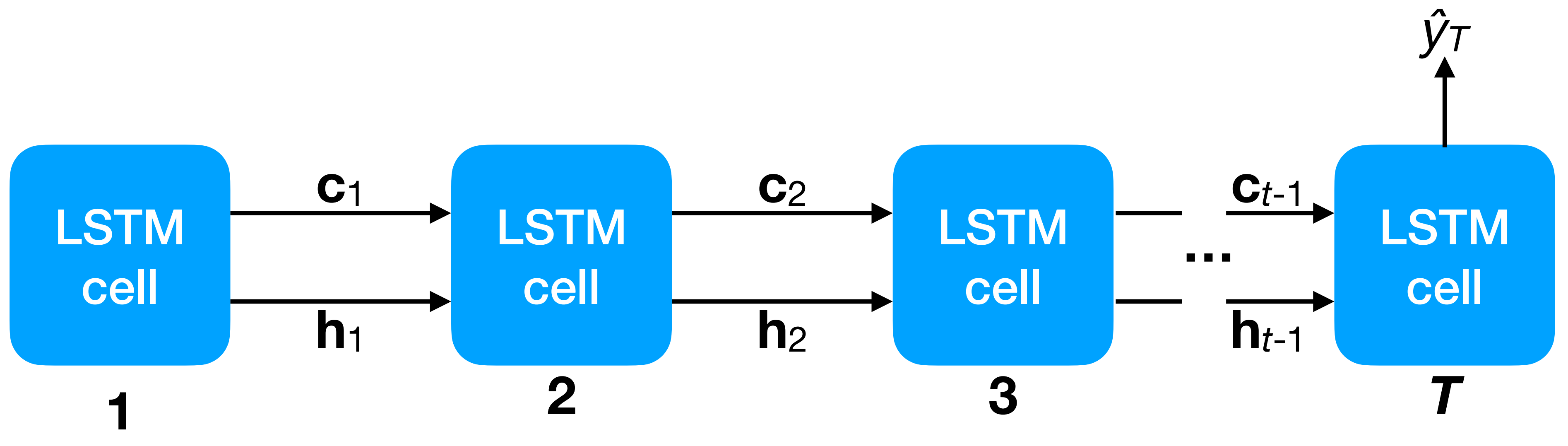
LSTM

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

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

$$\ldots$$

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



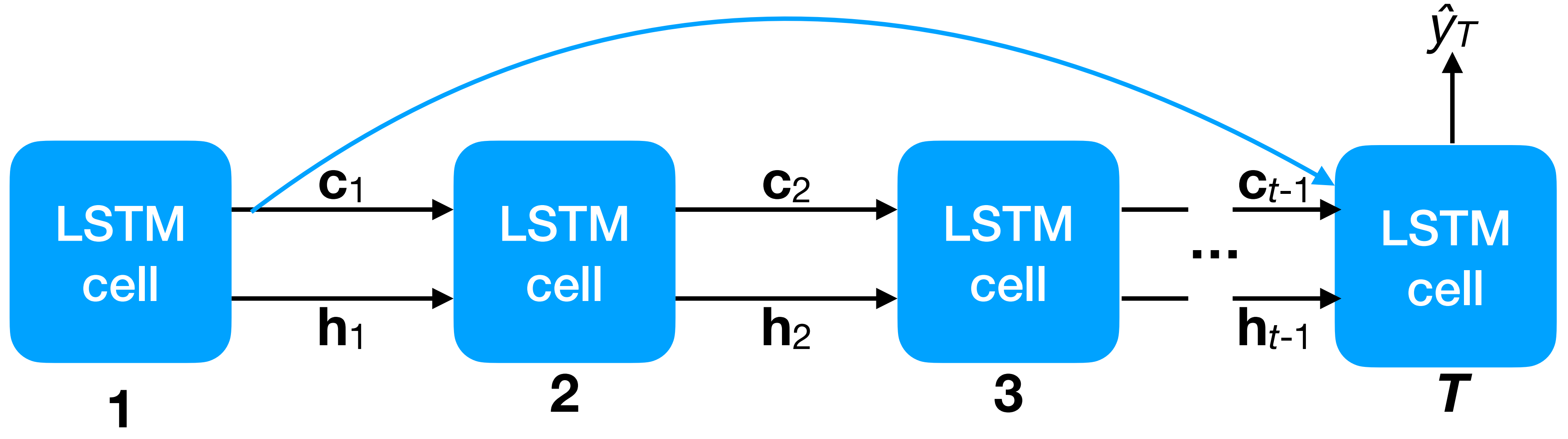
LSTM

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

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

$$\ldots$$

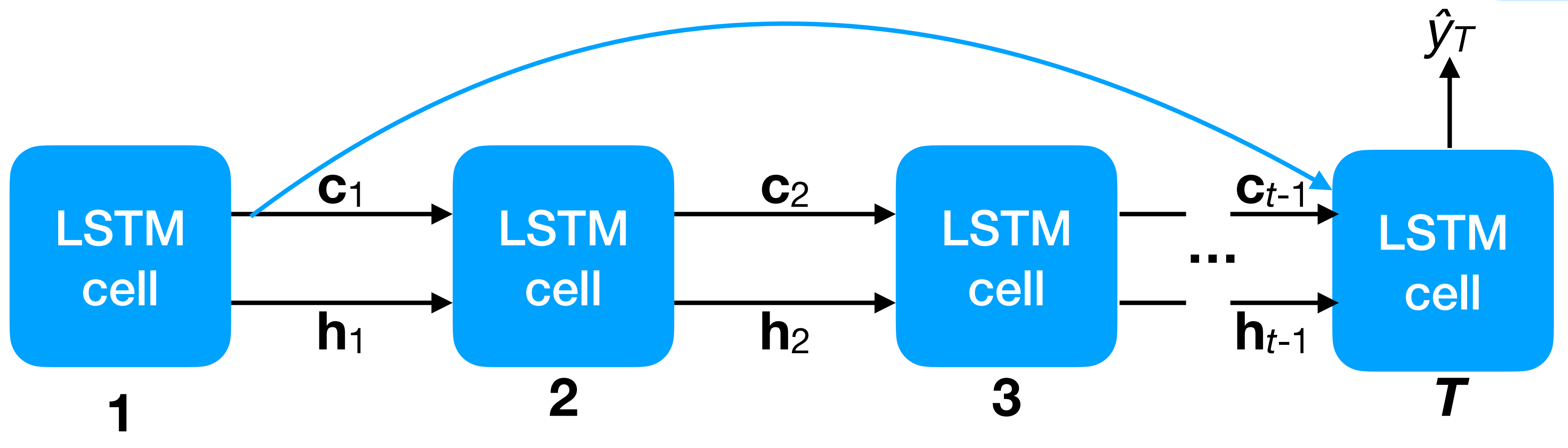
$$\mathbf{c}_t = \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \ldots + \mathbf{i}_2 \odot \tilde{\mathbf{c}}_2 + \mathbf{c}_1$$



LSTM

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

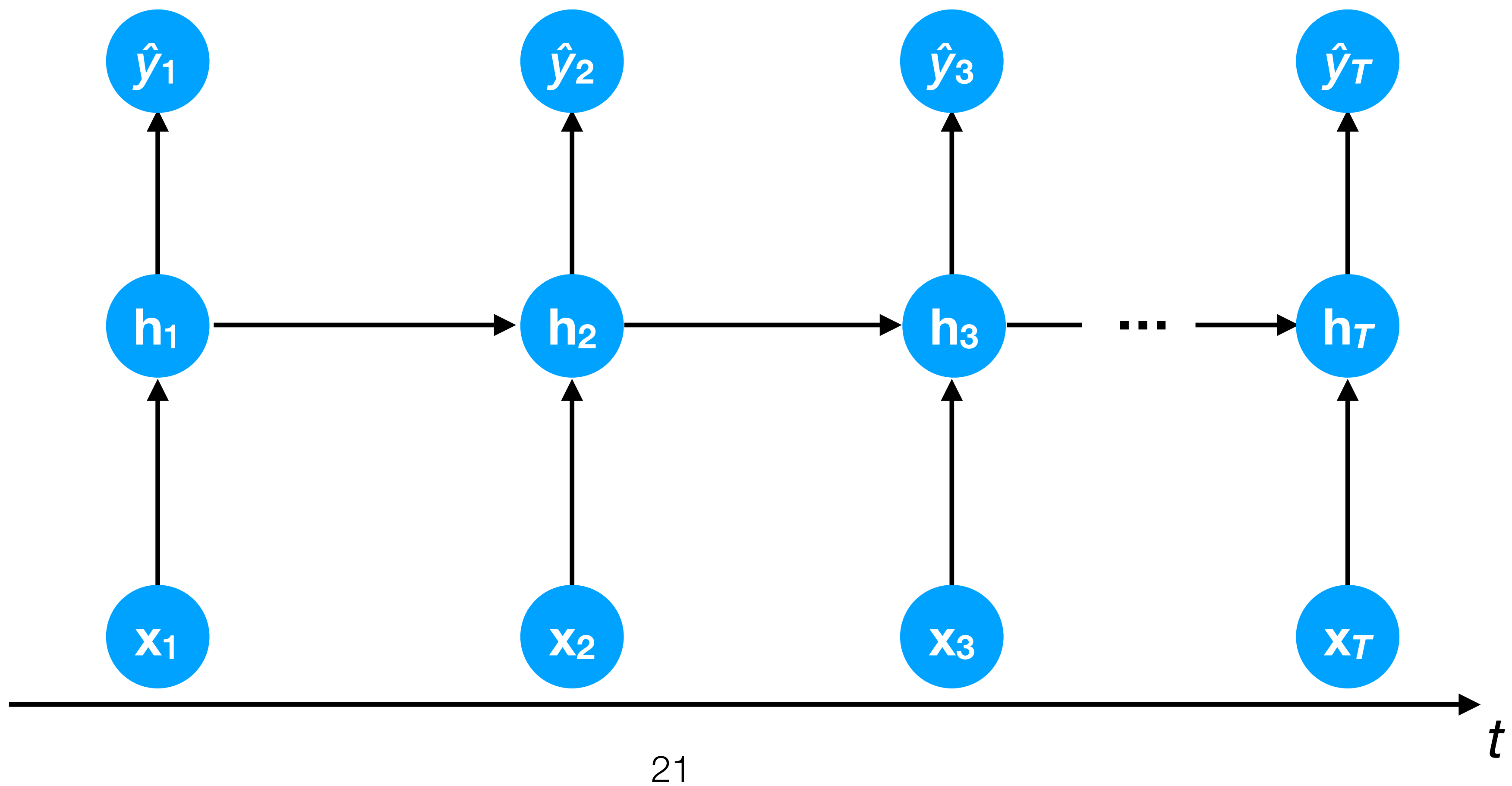
$$\mathbf{c}_t = \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{i}_{t-1} \odot \tilde{\mathbf{c}}_{t-1} + \ldots + \mathbf{i}_2 \odot \tilde{\mathbf{c}}_2 + \mathbf{c}_1$$



Bi-directional RNNs

Bi-directional RNNs

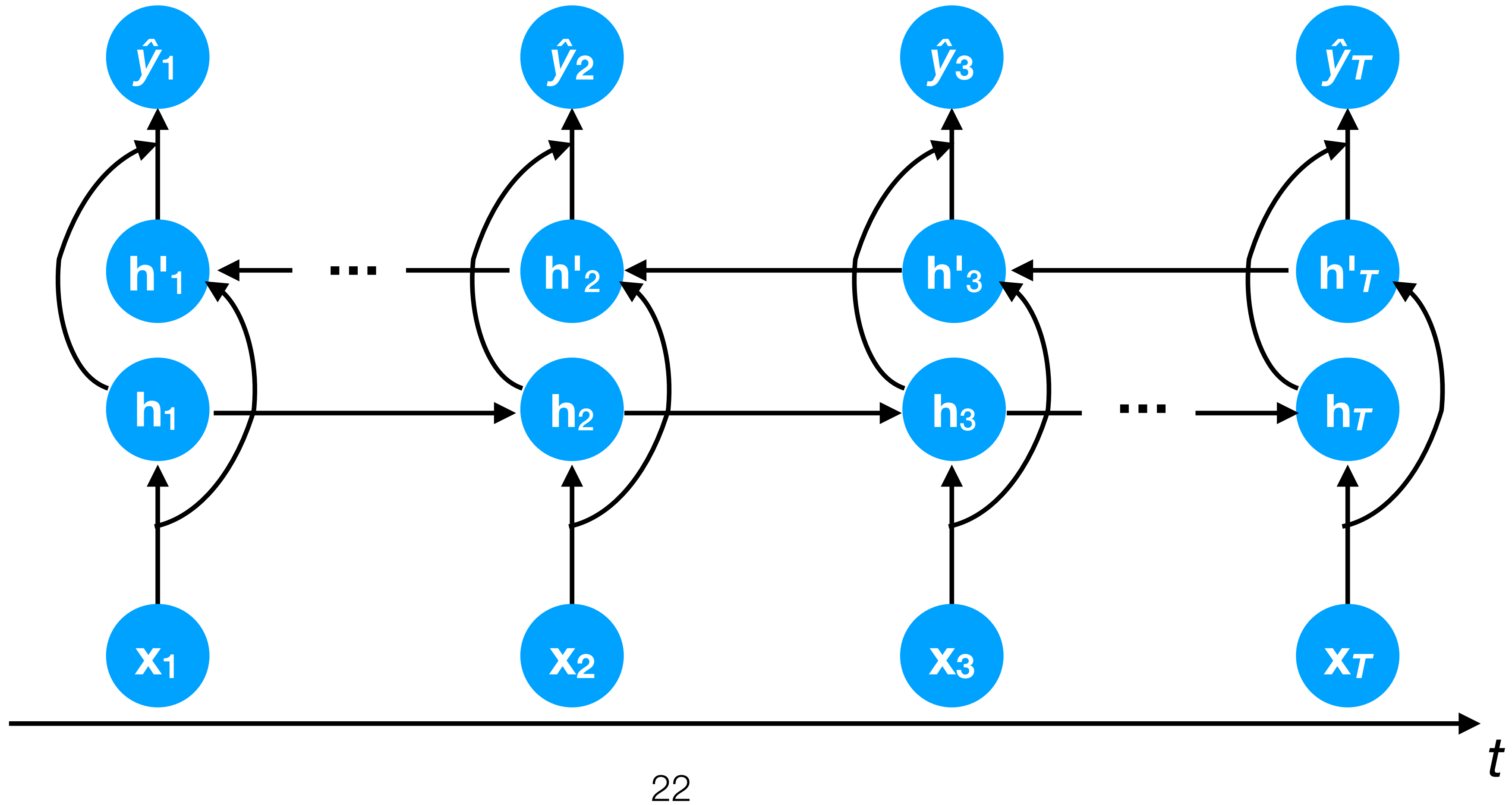
- The RNNs (including LSTMs) we have examined so far are useful when an output $\hat{y}_t$ must be estimated immediately after t timesteps.



Bi-directional RNNs

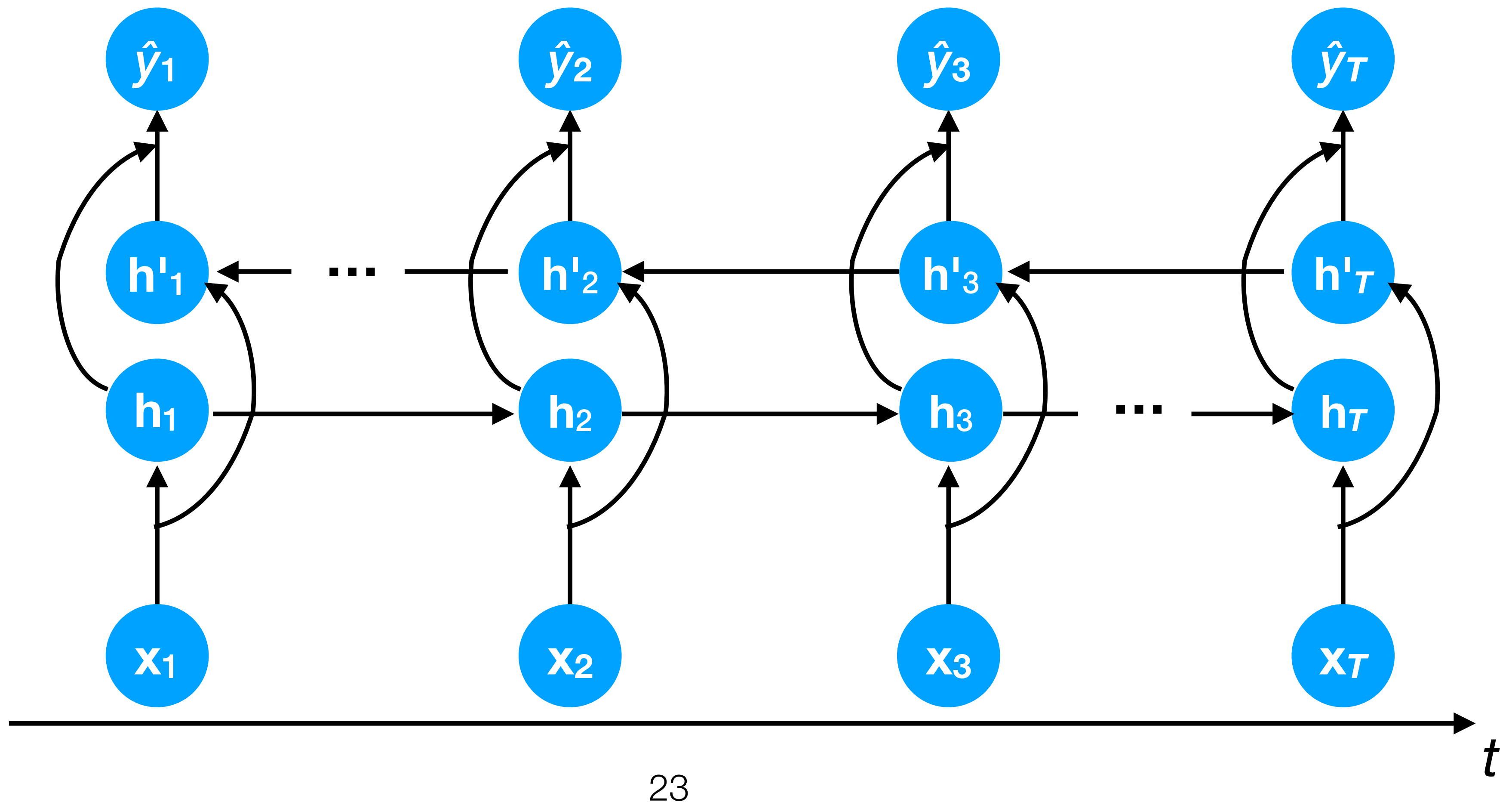
- However, in some settings, we may be able to wait to see the *entire* input sequence before producing any outputs.

- 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 $\mathbf{W}^{(k)}$.

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

- 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 $\varepsilon\alpha < 1$, **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 $\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{n}$.

L_2 regularization ≅ Gaussian noise augmentation

- Instead of a sum, the cost function contains the *expected* L_2 distance between the predictions and target labels.

- We define separate cost functions for the original (**x**) and the noise-augmented ($\tilde{\mathbf{x}}$) inputs w.r.t. weights **w**:

$$f(\mathbf{w}) = \mathbb{E}[(\hat{y} - y)^2] = \mathbb{E}[(\mathbf{x}^\top \mathbf{w} - y)^2]$$
$$\tilde{f}(\mathbf{w}) = \mathbb{E}[(\tilde{\hat{y}} - y)^2] = \mathbb{E}[(\tilde{\mathbf{x}}^\top \mathbf{w} - y)^2]$$

L_2 regularization ≅ Gaussian noise augmentation

- We can then derive that the noise-augmented loss $\tilde{f}$ equals the original loss f, plus an L_2 regularization term:

$$\mathbb{E}[(\tilde{\mathbf{x}}^\top \mathbf{w} - y)^2] = \mathbb{E}[((\mathbf{x} + \mathbf{n})^\top \mathbf{w} - y)^2]$$

L_2 regularization ≊ Gaussian noise augmentation

- We can then derive that the noise-augmented loss $\tilde{f}$ equals the original loss *f*, plus an L_2 regularization term:

$$\begin{aligned}\mathbb{E}[(\tilde{\mathbf{x}}^\top \mathbf{w} - y)^2] &= \mathbb{E}[((\mathbf{x}+\mathbf{n})^\top \mathbf{w} - y)^2] \\ &= \mathbb{E}[(\mathbf{x}^\top \mathbf{w} + \mathbf{n}^\top \mathbf{w} - y)^2]\end{aligned}$$

L_2 regularization ≅ Gaussian noise augmentation

- We can then derive that the noise-augmented loss $\tilde{f}$ equals the original loss f, plus an L_2 regularization term:

$$\begin{aligned}\mathbb{E}[(\tilde{\mathbf{x}}^\top \mathbf{w} - y)^2] &= \mathbb{E}[((\mathbf{x}+\mathbf{n})^\top \mathbf{w} - y)^2] \\ &= \mathbb{E}[(\mathbf{x}^\top \mathbf{w} + \mathbf{n}^\top \mathbf{w} - y)^2] \\ &= \mathbb{E}[((\mathbf{x}^\top \mathbf{w} - y) + \mathbf{n}^\top \mathbf{w})^2]\end{aligned}$$

L_2 regularization ≅ Gaussian noise augmentation

- We can then derive that the noise-augmented loss $\tilde{f}$ equals the original loss f, plus an L_2 regularization term:

$$\begin{aligned}
\mathbb{E}[(\tilde{\mathbf{x}}^\top \mathbf{w} - y)^2] &= \mathbb{E}[((\mathbf{x}+\mathbf{n})^\top \mathbf{w} - y)^2] \\
&= \mathbb{E}[(\mathbf{x}^\top \mathbf{w} + \mathbf{n}^\top \mathbf{w} - y)^2] \\
&= \mathbb{E}[((\mathbf{x}^\top \mathbf{w} - y) + \mathbf{n}^\top \mathbf{w})^2] \\
&= \mathbb{E}[(\mathbf{x}^\top \mathbf{w} - y)^2] + \mathbb{E}[2(\mathbf{x}^\top \mathbf{w} - y)\mathbf{n}^\top \mathbf{w}] + \mathbb{E}[(\mathbf{n}^\top \mathbf{w})^2]
\end{aligned}$$

L_2 regularization ≊ Gaussian noise augmentation

- We can then derive that the noise-augmented loss $\tilde{f}$ equals the original loss f, plus an L_2 regularization term:

$$\begin{aligned}
\mathbb{E}[(\tilde{\mathbf{x}}^\top \mathbf{w} - y)^2] &= \mathbb{E}[((\mathbf{x} + \mathbf{n})^\top \mathbf{w} - y)^2] \\
&= \mathbb{E}[(\mathbf{x}^\top \mathbf{w} + \mathbf{n}^\top \mathbf{w} - y)^2] \\
&= \mathbb{E}[((\mathbf{x}^\top \mathbf{w} - y) + \mathbf{n}^\top \mathbf{w})^2] \\
&= \mathbb{E}[(\mathbf{x}^\top \mathbf{w} - y)^2] + \mathbb{E}[2(\mathbf{x}^\top \mathbf{w} - y)\mathbf{n}^\top \mathbf{w}] + \mathbb{E}[(\mathbf{n}^\top \mathbf{w})^2] \\
&= f(\mathbf{w}) + \mathbb{E}[2(\mathbf{x}^\top \mathbf{w}\mathbf{n}^\top \mathbf{w}) - 2y\mathbf{n}^\top \mathbf{w}] + \mathbb{E}[\mathbf{w}^\top \mathbf{n}\mathbf{n}^\top \mathbf{w}]
\end{aligned}$$

L_2 regularization ≊ Gaussian noise augmentation

- We can then derive that the noise-augmented loss $\tilde{f}$ equals the original loss f, plus an L_2 regularization term:

$$\begin{aligned}
\mathbb{E}[(\tilde{\mathbf{x}}^\top \mathbf{w} - y)^2] &= \mathbb{E}[((\mathbf{x}+\mathbf{n})^\top \mathbf{w} - y)^2] \\
&= \mathbb{E}[(\mathbf{x}^\top \mathbf{w} + \mathbf{n}^\top \mathbf{w} - y)^2] \\
&= \mathbb{E}[((\mathbf{x}^\top \mathbf{w} - y) + \mathbf{n}^\top \mathbf{w})^2] \\
&= \mathbb{E}[(\mathbf{x}^\top \mathbf{w} - y)^2] + \mathbb{E}[2(\mathbf{x}^\top \mathbf{w} - y)\mathbf{n}^\top \mathbf{w}] + \mathbb{E}[(\mathbf{n}^\top \mathbf{w})^2] \\
&= f(\mathbf{w}) + \mathbb{E}[2(\mathbf{x}^\top \mathbf{w}\mathbf{n}^\top \mathbf{w}) - 2y\mathbf{n}^\top \mathbf{w}] + \mathbb{E}[\mathbf{w}^\top \mathbf{n}\mathbf{n}^\top \mathbf{w}] \\
&= f(\mathbf{w}) + 2\mathbb{E}[\mathbf{x}^\top]\mathbf{w}\mathbb{E}[\mathbf{n}^\top]\mathbf{w} - 2\mathbb{E}[y]\mathbb{E}[\mathbf{n}^\top]\mathbf{w} + \mathbf{w}^\top \mathbb{E}[\mathbf{n}\mathbf{n}^\top]\mathbf{w}
\end{aligned}$$

Here we can split the expectation into the product of two expectations.

L_2 regularization ≊ Gaussian noise augmentation

- We can then derive that the noise-augmented loss $\tilde{f}$ equals the original loss *f*, plus an L_2 regularization term:

$$\begin{aligned}
\mathbb{E}[(\tilde{\mathbf{x}}^\top \mathbf{w} - y)^2] &= \mathbb{E}[((\mathbf{x}+\mathbf{n})^\top \mathbf{w} - y)^2] \\
&= \mathbb{E}[(\mathbf{x}^\top \mathbf{w} + \mathbf{n}^\top \mathbf{w} - y)^2] \\
&= \mathbb{E}[((\mathbf{x}^\top \mathbf{w} - y) + \mathbf{n}^\top \mathbf{w})^2] \\
&= \mathbb{E}[(\mathbf{x}^\top \mathbf{w} - y)^2] + \mathbb{E}[2(\mathbf{x}^\top \mathbf{w} - y)\mathbf{n}^\top \mathbf{w}] + \mathbb{E}[(\mathbf{n}^\top \mathbf{w})^2] \\
&= f(\mathbf{w}) + \mathbb{E}[2(\mathbf{x}^\top \mathbf{w}\mathbf{n}^\top \mathbf{w}) - 2y\mathbf{n}^\top \mathbf{w}] + \mathbb{E}[\mathbf{w}^\top \mathbf{n}\mathbf{n}^\top \mathbf{w}] \\
&= f(\mathbf{w}) + 2\mathbb{E}[\mathbf{x}^\top]\mathbf{w}\mathbb{E}[\mathbf{n}^\top]\mathbf{w} - 2\mathbb{E}[y]\mathbb{E}[\mathbf{n}^\top]\mathbf{w} + \mathbf{w}^\top \mathbb{E}[\mathbf{n}\mathbf{n}^\top]\mathbf{w} \\
&= f(\mathbf{w}) + 2\mathbb{E}[\mathbf{x}^\top]\mathbf{w} \cdot 0 \cdot \mathbf{w} - 2\mathbb{E}[y] \cdot 0 \cdot \mathbf{w} + \mathbf{w}^\top \alpha \mathbf{I} \mathbf{w}
\end{aligned}$$

n has 0 mean.

L_2 regularization ≊ Gaussian noise augmentation

- We can then derive that the noise-augmented loss $\tilde{f}$ equals the original loss f, plus an L_2 regularization term:

$$\begin{aligned}
\mathbb{E}[(\tilde{\mathbf{x}}^\top \mathbf{w} - y)^2] &= \mathbb{E}[((\mathbf{x}+\mathbf{n})^\top \mathbf{w} - y)^2] \\
&= \mathbb{E}[(\mathbf{x}^\top \mathbf{w} + \mathbf{n}^\top \mathbf{w} - y)^2] \\
&= \mathbb{E}[((\mathbf{x}^\top \mathbf{w} - y) + \mathbf{n}^\top \mathbf{w})^2] \\
&= \mathbb{E}[(\mathbf{x}^\top \mathbf{w} - y)^2] + \mathbb{E}[2(\mathbf{x}^\top \mathbf{w} - y)\mathbf{n}^\top \mathbf{w}] + \mathbb{E}[(\mathbf{n}^\top \mathbf{w})^2] \\
&= f(\mathbf{w}) + \mathbb{E}[2(\mathbf{x}^\top \mathbf{w}\mathbf{n}^\top \mathbf{w}) - 2y\mathbf{n}^\top \mathbf{w}] + \mathbb{E}[\mathbf{w}^\top \mathbf{n}\mathbf{n}^\top \mathbf{w}] \\
&= f(\mathbf{w}) + 2\mathbb{E}[\mathbf{x}^\top]\mathbf{w}\mathbb{E}[\mathbf{n}^\top]\mathbf{w} - 2\mathbb{E}[y]\mathbb{E}[\mathbf{n}^\top]\mathbf{w} + \mathbf{w}^\top \mathbb{E}[\mathbf{n}\mathbf{n}^\top]\mathbf{w} \\
&= f(\mathbf{w}) + 2\mathbb{E}[\mathbf{x}^\top]\mathbf{w} \cdot 0 \cdot \mathbf{w} - 2\mathbb{E}[y] \cdot 0 \cdot \mathbf{w} + \mathbf{w}^\top \alpha\mathbf{I}\mathbf{w}
\end{aligned}$$

Covariance of n.

L_2 regularization ≅ Gaussian noise augmentation

- We can then derive that the noise-augmented loss $\tilde{f}$ equals the original loss f, plus an L_2 regularization term:

$$\begin{aligned}
\mathbb{E}[(\tilde{\mathbf{x}}^\top \mathbf{w} - y)^2] &= \mathbb{E}[((\mathbf{x}+\mathbf{n})^\top \mathbf{w} - y)^2] \\
&= \mathbb{E}[(\mathbf{x}^\top \mathbf{w} + \mathbf{n}^\top \mathbf{w} - y)^2] \\
&= \mathbb{E}[((\mathbf{x}^\top \mathbf{w} - y) + \mathbf{n}^\top \mathbf{w})^2] \\
&= \mathbb{E}[(\mathbf{x}^\top \mathbf{w} - y)^2] + \mathbb{E}[2(\mathbf{x}^\top \mathbf{w} - y)\mathbf{n}^\top \mathbf{w}] + \mathbb{E}[(\mathbf{n}^\top \mathbf{w})^2] \\
&= f(\mathbf{w}) + \mathbb{E}[2(\mathbf{x}^\top \mathbf{w}\mathbf{n}^\top \mathbf{w}) - 2y\mathbf{n}^\top \mathbf{w}] + \mathbb{E}[\mathbf{w}^\top \mathbf{n}\mathbf{n}^\top \mathbf{w}] \\
&= f(\mathbf{w}) + 2\mathbb{E}[\mathbf{x}^\top]\mathbf{w}\mathbb{E}[\mathbf{n}^\top]\mathbf{w} - 2\mathbb{E}[y]\mathbb{E}[\mathbf{n}^\top]\mathbf{w} + \mathbf{w}^\top \mathbb{E}[\mathbf{n}\mathbf{n}^\top]\mathbf{w} \\
&= f(\mathbf{w}) + 2\mathbb{E}[\mathbf{x}^\top]\mathbf{w} \cdot 0 \cdot \mathbf{w} - 2\mathbb{E}[y] \cdot 0 \cdot \mathbf{w} + \mathbf{w}^\top \alpha \mathbf{I} \mathbf{w} \\
&= f(\mathbf{w}) + \underbrace{\alpha \mathbf{w}^\top \mathbf{w}}_{L_2 \text{ regularization}}
\end{aligned}$$

L_2 regularization ≊ Gaussian noise augmentation

- For non-linear and deep NNs, element-wise Gaussian noise augmentation and L_2 regularization are no longer equivalent, but they may sometimes have similar effects.

Weight sharing

- One of the most powerful methods of regularizing a neural network is to reduce the number of parameters by tying some weight matrices to be the same.

- Prominent cases:

  - CNNs: the same convolution filter is used at *every location*.

  - RNNs (including LSTMs, GRUs, etc): the same weights are used at *every timestep*.

Pre-training

- Both supervised and unsupervised pre-training allow ML practitioners to harness much larger datasets to learn good representations of the inputs.

- For domains with a small number of training data, this can be a powerful regularization technique to prevent overfitting.

Ensembles

- By training multiple predictors and averaging their outputs, we can create an **ensemble**.
- Ensembles are an easy and often effective way of increasing accuracy.
- Useful ensembles require the individual predictors' outputs to have low correlation, i.e., they make different kinds of mistakes on the same inputs.

Ensembles

- Suppose we train an ensemble of n NNs, each of which is unbiased, i.e., $E[\hat{y}_i - y] = E[\varepsilon_i] = 0$, where:
  - y is the target label for a randomly drawn example **x**.
  - $\hat{y}_i$ is the ith NN's prediction for **x**.
  - ε_i is the error of NN i's prediction.
- Let the variance (expected squared error) of NN i be $E[\varepsilon_i^2] = v$.
- Suppose the covariance of predictions between NNs $i \neq j$ is $E[\varepsilon_i \varepsilon_j] = c$.
- The ensemble's prediction on any **x** is: $\frac{1}{n}\sum_{i=1}^{n} \hat{y}_i$

Ensembles

- We can compute the expected squared error of the ensemble:

$$\mathbb{E}\left[\left(\left(\frac{1}{n}\sum_{i=1}^{n}\hat{y}_i\right)-y\right)^2\right]=\mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i-y)\right)^2\right]$$

Ensembles

- We can compute the expected squared error of the ensemble:

$$
\begin{aligned}
\mathbb{E}\left[\left(\left(\frac{1}{n}\sum_{i=1}^{n}\hat{y}_i\right)-y\right)^2\right] &= \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i-y)\right)^2\right] \\
&= \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}\epsilon_i\right)^2\right]
\end{aligned}
$$

Ensembles

- We can compute the expected squared error of the ensemble:

$$
\begin{aligned}
\mathbb{E}\left[\left(\left(\frac{1}{n}\sum_{i=1}^{n}\hat{y}_i\right)-y\right)^2\right] &= \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i-y)\right)^2\right] \\
&= \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}\epsilon_i\right)^2\right] \\
&= \frac{1}{n^2}\mathbb{E}\left[\begin{array}{c} \epsilon_1\epsilon_1+\epsilon_1\epsilon_2+\ldots+\epsilon_1\epsilon_n+ \\ \ldots \\ +\epsilon_n\epsilon_1+\epsilon_n\epsilon_2+\ldots+\epsilon_1\epsilon_n \end{array}\right]
\end{aligned}
$$

Ensembles

- We can compute the expected squared error of the ensemble:

$$
\begin{aligned}
\mathbb{E}\left[\left(\left(\frac{1}{n}\sum_{i=1}^{n}\hat{y}_i\right)-y\right)^2\right] &= \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i-y)\right)^2\right] \\
&= \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}\epsilon_i\right)^2\right] \\
&= \frac{1}{n^2}\mathbb{E}\left[\begin{array}{c} \epsilon_1\epsilon_1+\epsilon_1\epsilon_2+\ldots+\epsilon_1\epsilon_n+ \\ \ldots \\ +\epsilon_n\epsilon_1+\epsilon_n\epsilon_2+\ldots+\epsilon_1\epsilon_n \end{array}\right] \\
&= \frac{1}{n^2}\left[\begin{array}{c} v+c+\ldots+c+ \\ \ldots \\ +c+\ldots+c+v \end{array}\right]
\end{aligned}
$$

Ensembles

- We can compute the expected squared error of the ensemble:

$$
\begin{aligned}
\mathbb{E}\left[\left(\left(\frac{1}{n}\sum_{i=1}^{n}\hat{y}_i\right) - y\right)^2\right] &= \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y)\right)^2\right] \\
&= \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}\epsilon_i\right)^2\right] \\
&= \frac{1}{n^2}\mathbb{E}\left[\begin{array}{c} \epsilon_1\epsilon_1 + \epsilon_1\epsilon_2 + \ldots + \epsilon_1\epsilon_n + \\ \ldots \\ +\epsilon_n\epsilon_1 + \epsilon_n\epsilon_2 + \ldots + \epsilon_1\epsilon_n \end{array}\right] \\
&= \frac{1}{n^2}\left[\begin{array}{c} v + c + \ldots + c + \\ \ldots \\ +c + \ldots + c + v \end{array}\right] \\
&= \frac{nv + (n-1)nc}{n^2}
\end{aligned}
$$

Ensembles

- We can compute the expected squared error of the ensemble:

$$
\begin{aligned}
\mathbb{E}\left[\left(\left(\frac{1}{n}\sum_{i=1}^{n}\hat{y}_i\right)-y\right)^2\right] &= \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i-y)\right)^2\right] \\
&= \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}\epsilon_i\right)^2\right] \\
&= \frac{1}{n^2}\mathbb{E}\left[\begin{array}{c} \epsilon_1\epsilon_1+\epsilon_1\epsilon_2+\ldots+\epsilon_1\epsilon_n+ \\ \ldots \\ +\epsilon_n\epsilon_1+\epsilon_n\epsilon_2+\ldots+\epsilon_1\epsilon_n \end{array}\right] \\
&= \frac{1}{n^2}\left[\begin{array}{c} v+c+\ldots+c+ \\ \ldots \\ +c+\ldots+c+v \end{array}\right] \\
&= \frac{nv+(n-1)nc}{n^2} \\
&= \frac{v}{n}+\frac{n-1}{n}c
\end{aligned}
$$

Ensembles

- If *c*=0 then the ensemble reduces our expected squared error by a factor of *n* — great news!

$$
\begin{aligned}
\mathbb{E}\left[\left(\left(\frac{1}{n}\sum_{i=1}^{n}\hat{y}_i\right)-y\right)^2\right] &= \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i-y)\right)^2\right] \\
&= \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}\epsilon_i\right)^2\right] \\
&= \frac{1}{n^2}\mathbb{E}\begin{bmatrix} \epsilon_1\epsilon_1+\epsilon_1\epsilon_2+\ldots+\epsilon_1\epsilon_n+ \\ \ldots \\ +\epsilon_n\epsilon_1+\epsilon_n\epsilon_2+\ldots+\epsilon_1\epsilon_n \end{bmatrix} \\
&= \frac{1}{n^2}\begin{bmatrix} v+c+\ldots+c+ \\ \ldots \\ +c+\ldots+c+v \end{bmatrix} \\
&= \frac{nv+(n-1)nc}{n^2} \\
&= \frac{v}{n}+\frac{n-1}{n}c
\end{aligned}
$$

Ensembles

- If *c*=*v* then the ensemble is no better than any of the individual predictors.

$$\mathbb{E}\left[\left(\left(\frac{1}{n}\sum_{i=1}^{n}\hat{y}_i\right)-y\right)^2\right] = \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y)\right)^2\right]$$

$$= \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}\epsilon_i\right)^2\right]$$

$$= \frac{1}{n^2}\mathbb{E}\left[\begin{array}{c}\epsilon_1\epsilon_1 + \epsilon_1\epsilon_2 + \ldots + \epsilon_1\epsilon_n + \\ \ldots \\ +\epsilon_n\epsilon_1 + \epsilon_n\epsilon_2 + \ldots + \epsilon_1\epsilon_n\end{array}\right]$$

$$= \frac{1}{n^2}\left[\begin{array}{c}v + c + \ldots + c + \\ \ldots \\ +c + \ldots + c + v\end{array}\right]$$

$$= \frac{nv + (n-1)nc}{n^2}$$

$$= \frac{v}{n} + \frac{n-1}{n}c$$

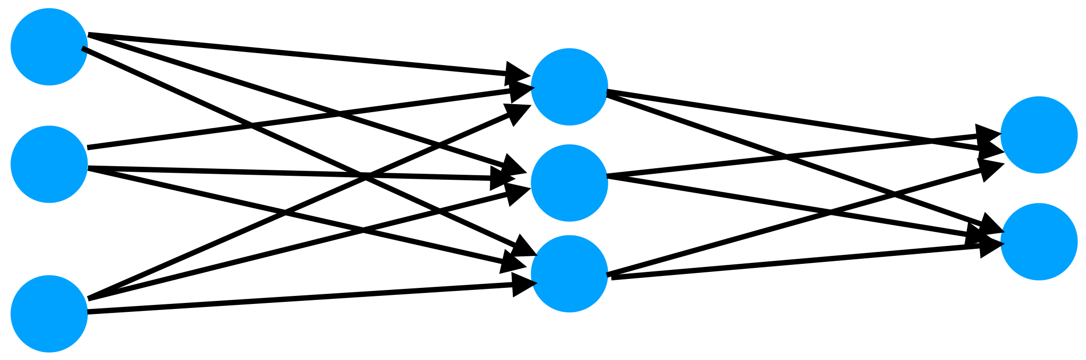
Dropout

- One of the most recently discovered regularization methods is **dropout**, whereby a random set of neurons is removed from the network at each iteration during both forward and backward propagation.

- Surprisingly, this simple method can both help the network to reach a better local minimum *and* prevent it from overfitting.

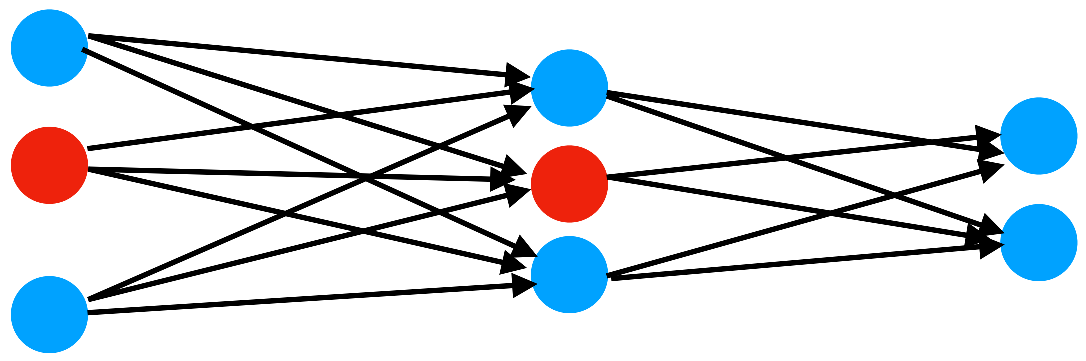
Dropout

- Suppose we are training the NN shown below:



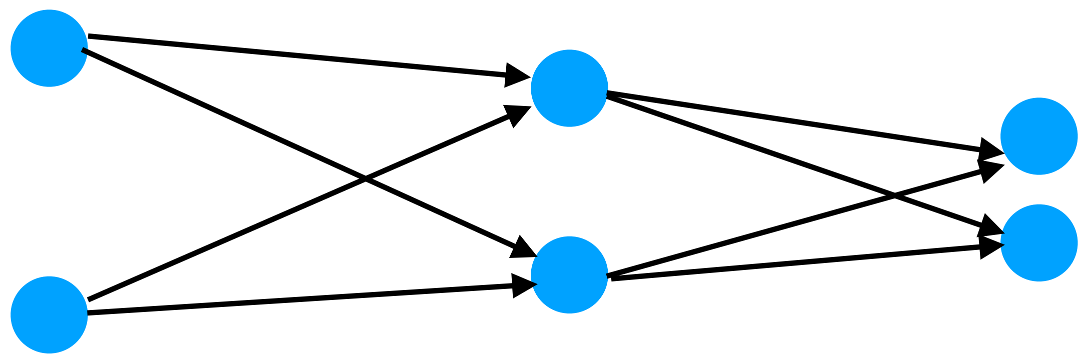
Dropout

- Suppose we are training the NN shown below:
- For each step of SGD, we randomly select (with “keep” probability *p*) some of the input and hidden neurons (*not* the output neurons).



Dropout

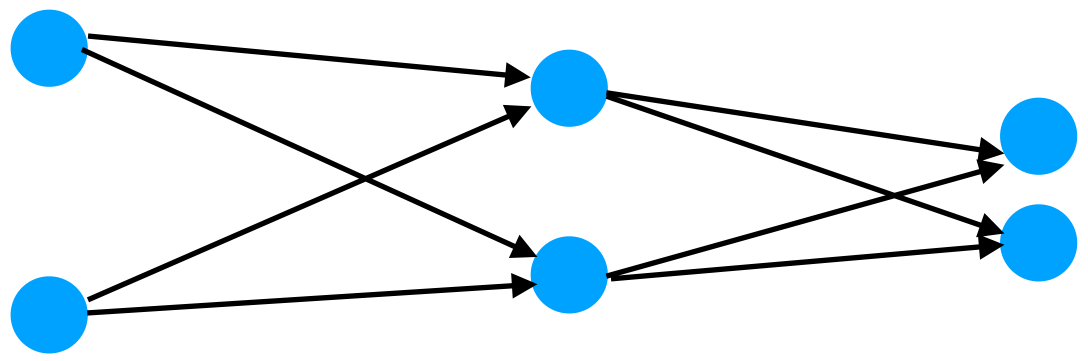
- Suppose we are training the NN shown below:
- We then remove these neurons and perform forward-propagation on the reduced network.



Dropout

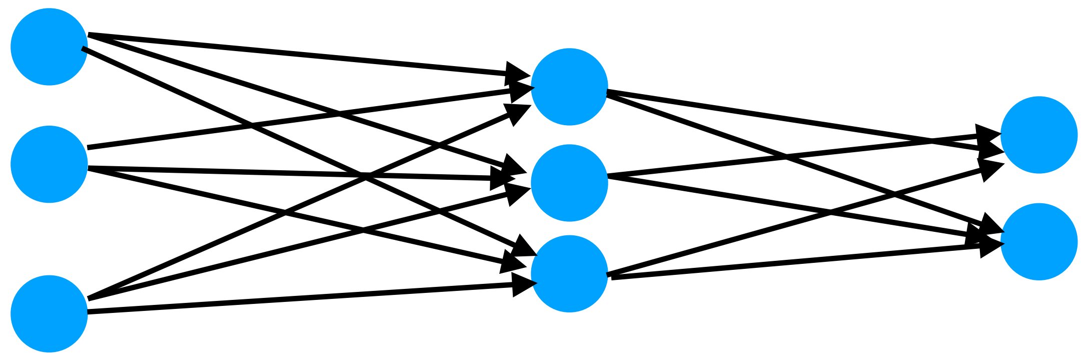
- Suppose we are training the NN shown below:

- During back-propagation, we adjust the weights of *only* those neurons that were retained in the reduced network.



Dropout

- We then replace the neurons we had removed and resume training. (During the next SGD iteration, we will randomly select *another* set of neurons to remove, etc.)

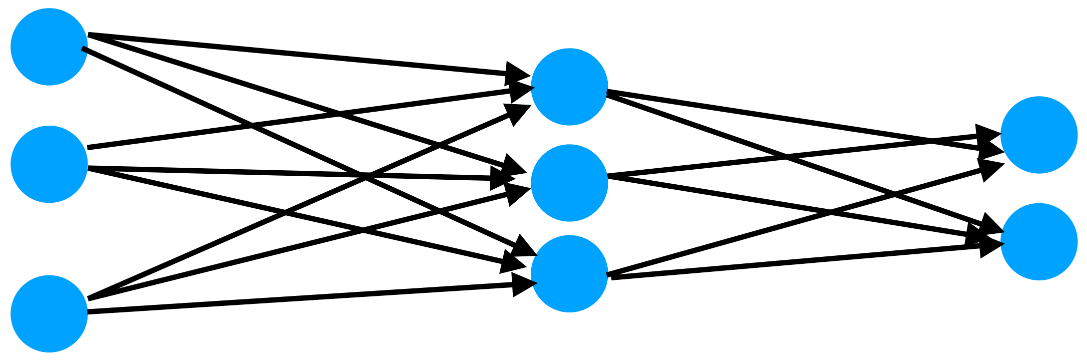


Dropout: example

- Suppose the weights are:

$$\mathbf{W}^{(1)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad \mathbf{W}^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

(For simplicity, assume that $\mathbf{b}^{(1)}=\mathbf{b}^{(2)}=0$.)

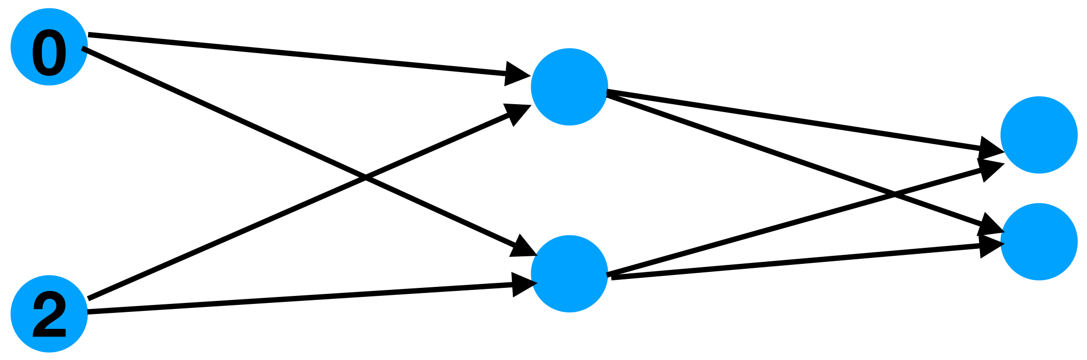


Dropout: example

- Suppose the weights are:

$$\mathbf{W}^{(1)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad \mathbf{W}^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

- If we drop the red neurons, then we will obtain $\hat{\mathbf{y}}=[60, 132]^T$ for the input $x=[0, 1, 2]^T$ during forward-propagation.

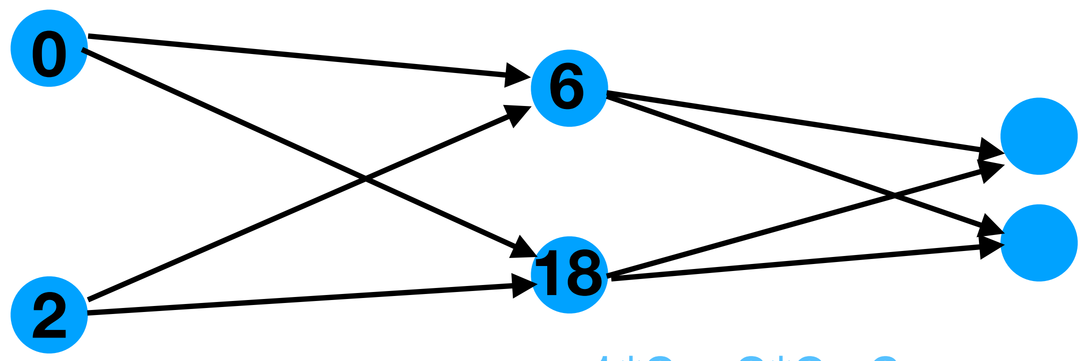


Dropout: example

- Suppose the weights are:

$$\mathbf{W}^{(1)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad \mathbf{W}^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

- If we drop the red neurons, then we will obtain $\hat{\mathbf{y}}=[60, 132]^T$ for the input $x=[0, 1, 2]^T$ during forward-propagation.



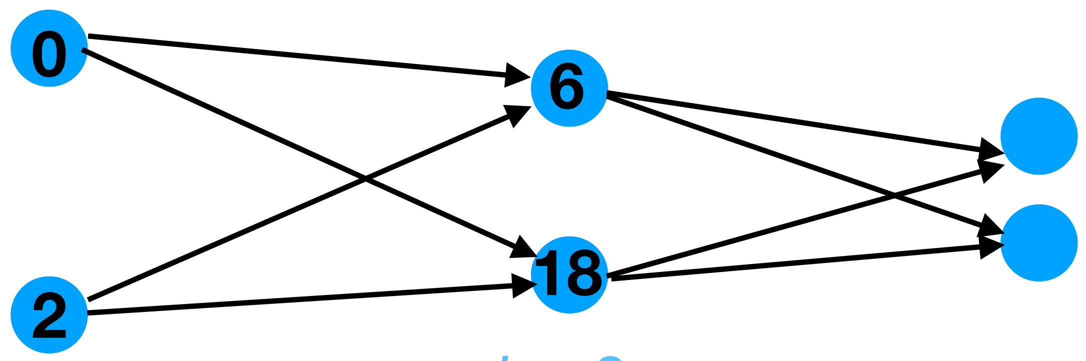
$z_1=1*0+3*2=6$
$z_2=7*0+9*2=18$

Dropout: example

- Suppose the weights are:

$$\mathbf{W}^{(1)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad \mathbf{W}^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

- If we drop the red neurons, then we will obtain $\hat{\mathbf{y}}=[60, 132]^T$ for the input $x=[0, 1, 2]^T$ during forward-propagation.



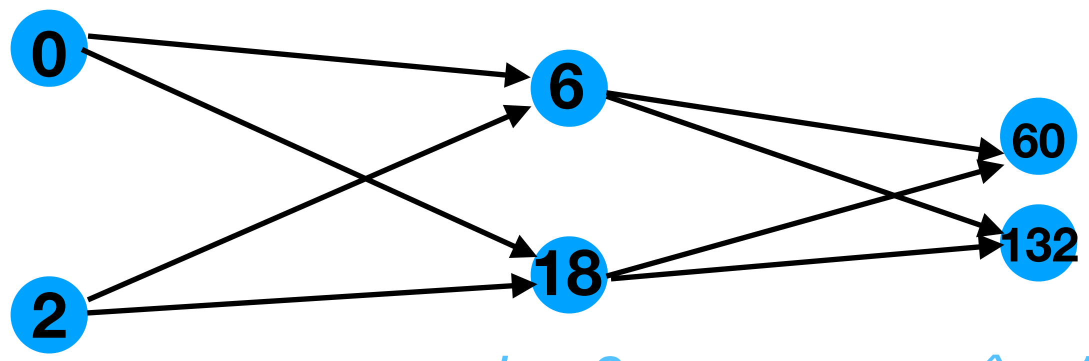
$h_1=6$
$h_2=18$

Dropout: example

- Suppose the weights are:

$$\mathbf{W}^{(1)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad \mathbf{W}^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

- If we drop the red neurons, then we will obtain $\hat{\mathbf{y}}=[60, 132]^T$ for the input $x=[0, 1, 2]^T$ during forward-propagation.



$h_1=6$
$h_2=18$

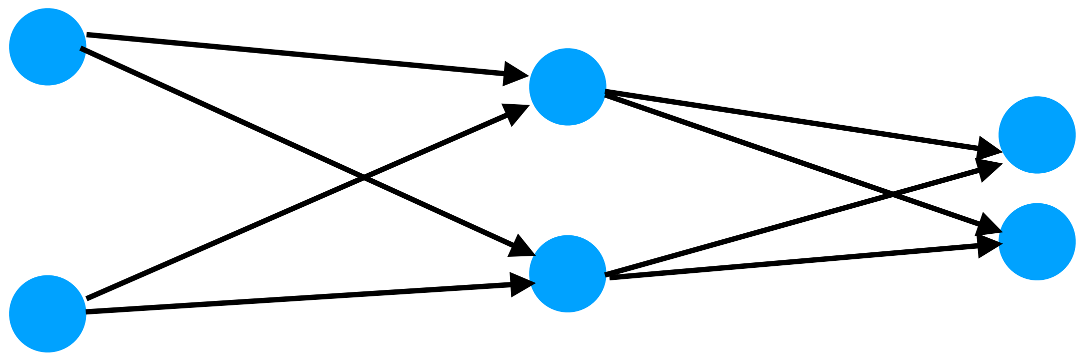
$\hat{y}_1=1*6+3*18=60$
$\hat{y}_2=4*6+6*18=132$

Dropout: example

- Suppose the weights are:

$$\mathbf{W}^{(1)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad \mathbf{W}^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

- During back-propagation, we will update the weights of *only* those neurons that were not removed.



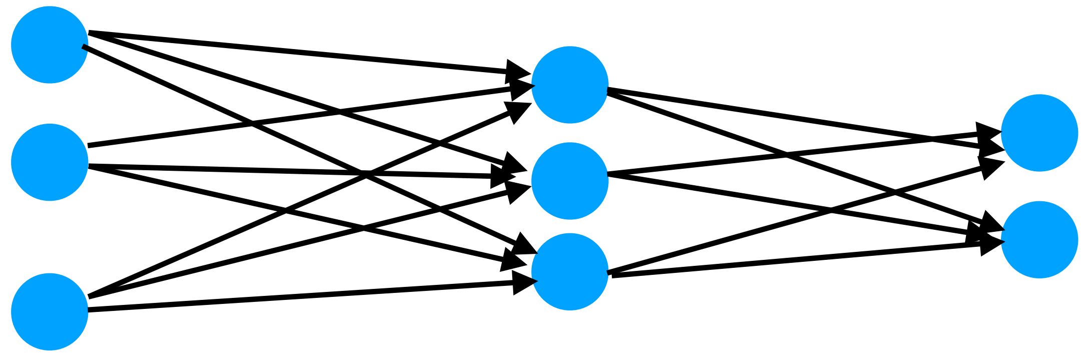
Dropout: why helpful?

- There are two main explanations for why dropout helps improve the accuracy of neural networks:
    - Symmetry breaking & prevention of co-adaptation.
    - Ensemble of many smaller networks.

Symmetry breaking

- When multiple neurons in the hidden layers are highly correlated with each other, the network does not utilize its full capacity.

- Extreme example: all rows of $\mathbf{W}^{(1)}$ are the same:



- Then all hidden units in $\mathbf{h}^{(1)}$ are also the same.

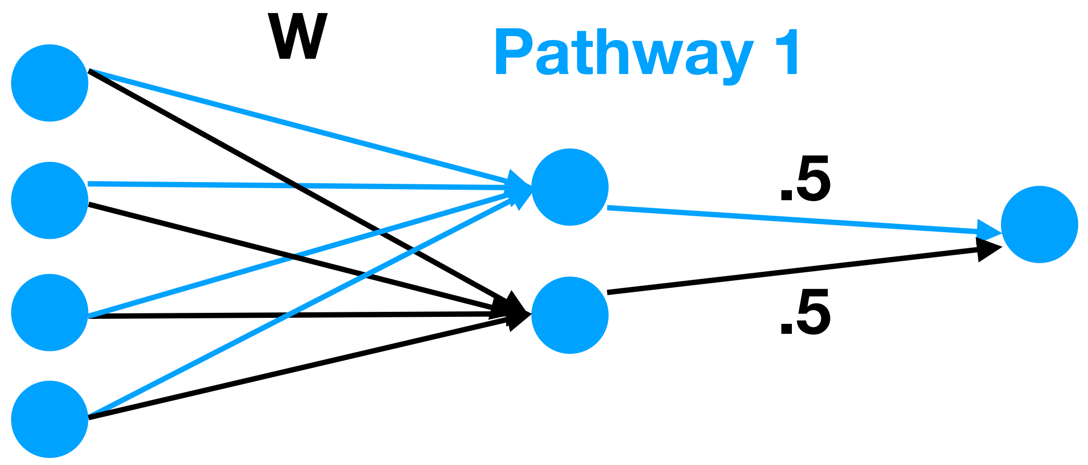
Symmetry breaking

- One reason why we initialize weights randomly is to break symmetry between them, so they learn to produce independent values in the subsequent hidden layer.

- Dropout can also help break symmetry since only some of the elements of each weight matrix are updated during each SGD iteration.

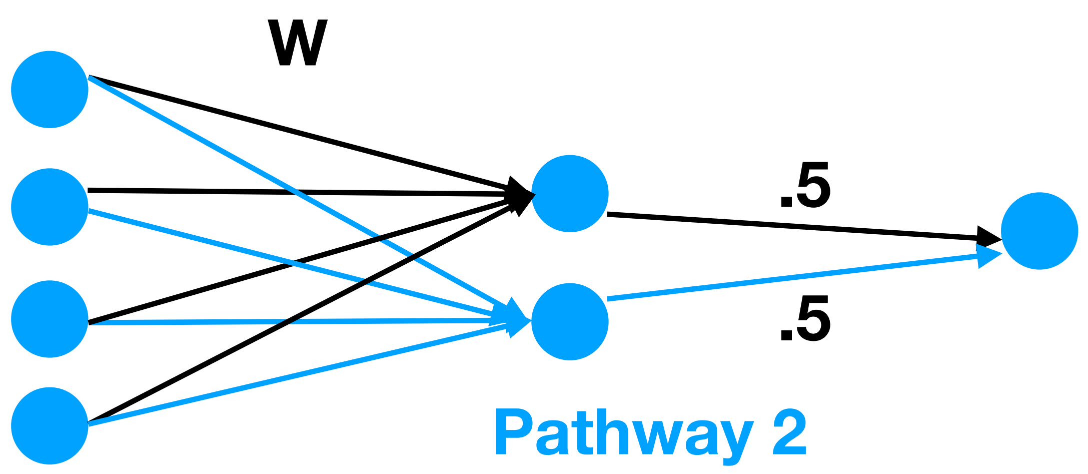
Co-adaptation

- A subtler problem that can occur is that the weights associated with different “pathways” through the NN can adapt to each other in pathological ways.



Co-adaptation

- A subtler problem that can occur is that the weights associated with different "pathways" through the NN can adapt to each other in pathological ways.

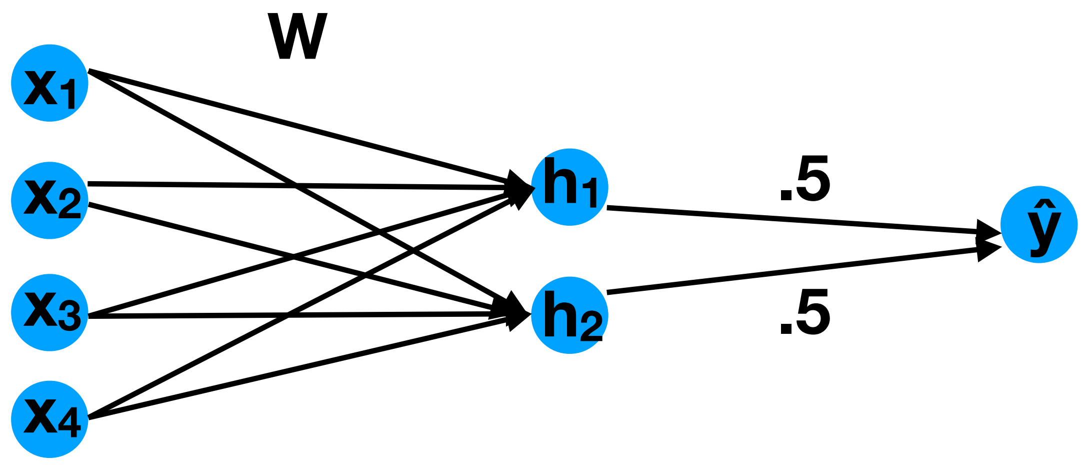


Co-adaptation

- If $\mathbf{W} = \begin{bmatrix} 2 & -3 & 1 & -1 \\ 2 & -3 & -1 & 1 \end{bmatrix}$

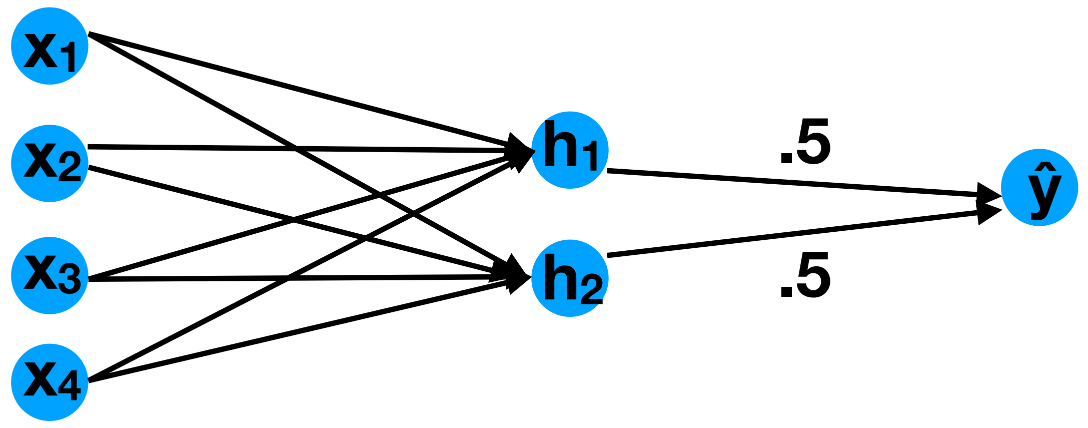
  then $\mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} 2x_1 - 3x_2 + x_3 - x_4 \\ 2x_1 - 3x_2 - x_3 + x_4 \end{bmatrix}$

  and hence $\hat{y} = 2x_1 - 3x_2$.



Co-adaptation

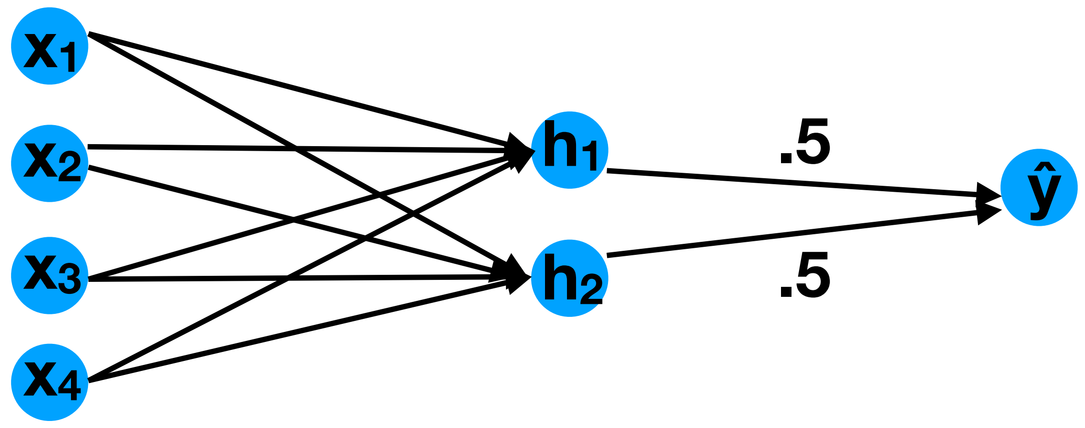
- At this point, neither x_3 nor x_4 have any impact on $\hat{y}$.
- We may be at a local minimum where the last two columns of **W** stay "locked" to effectively delete x_3, x_4.
- This is an example of **weight co-adaptation**; it is often a suboptimal solution.



$$\mathbf{W} = \left[\begin{array}{cccc} 2 & -3 & 1 & -1 \\ 2 & -3 & -1 & 1 \end{array}\right]$$

Co-adaptation

- Instead, we generally want every “pathway” of a NN to give an independently good estimate of *y*.
- Dropout can reduce co-adaptation since each pathway must "stand on its own”.

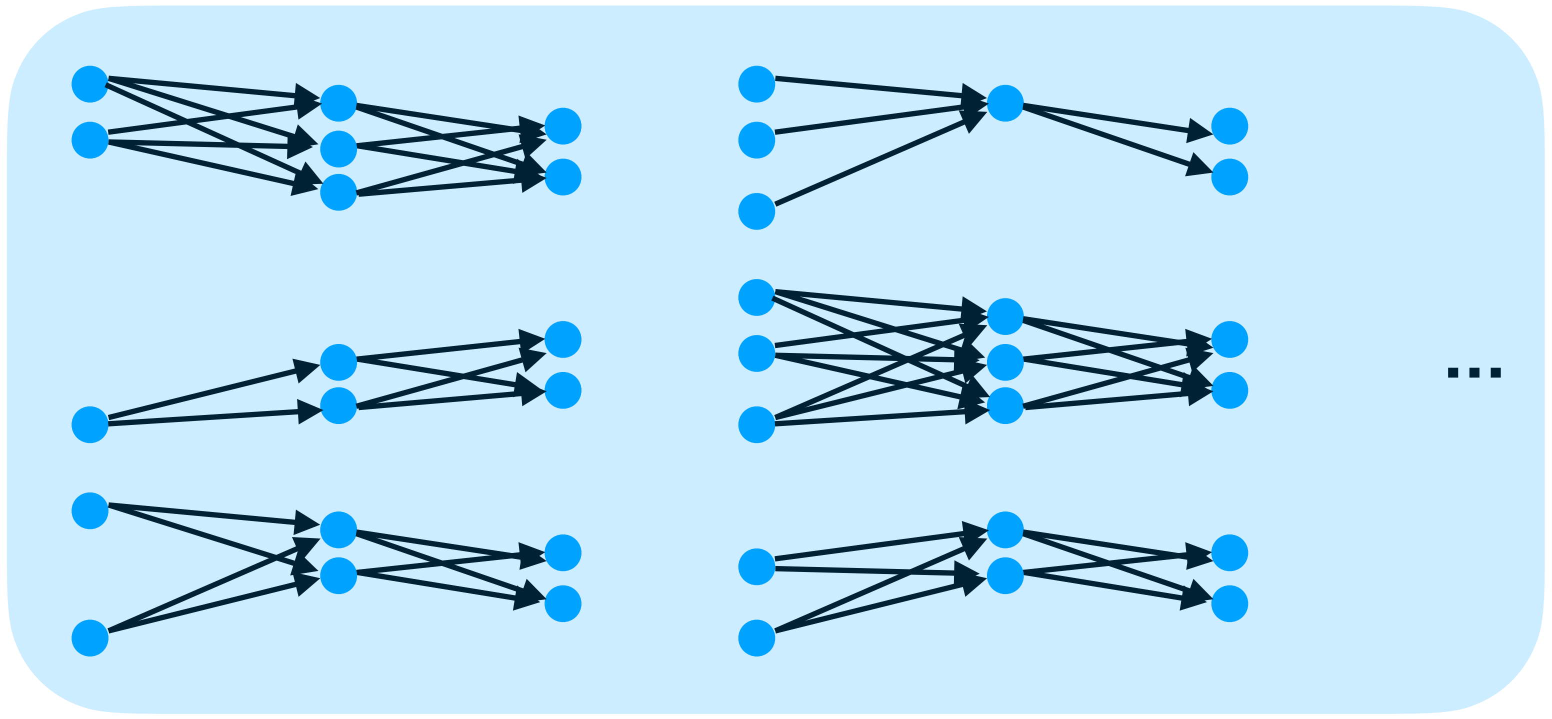


$$\mathbf{W} = \begin{bmatrix} 2 & -3 & 1 & -1 \\ 2 & -3 & -1 & 1 \end{bmatrix}$$

Ensemble of many smaller networks

- Dropout-based NN training can also be seen as approximating a large ensemble of many smaller networks.

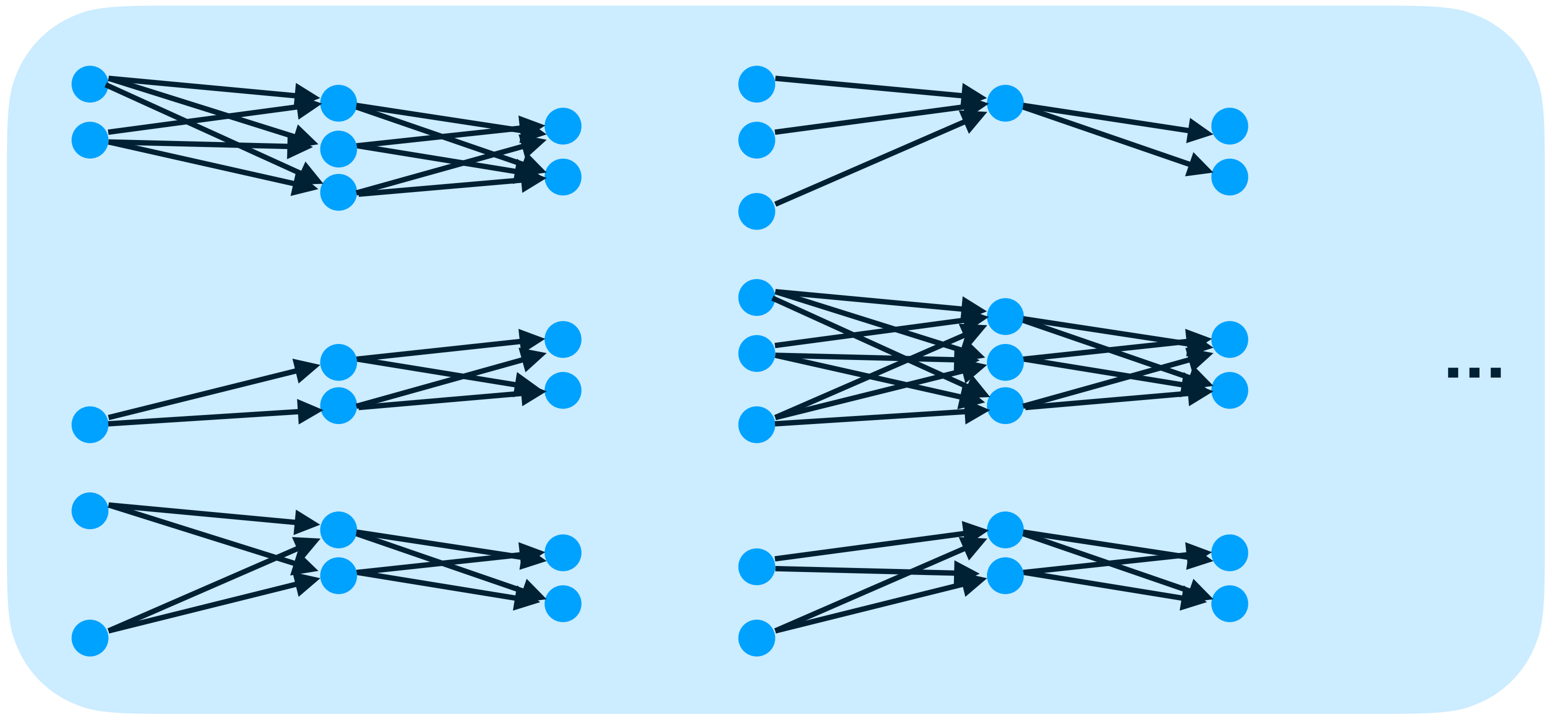
- Each member of the ensemble arises by randomly dropping some of the whole network's neurons:



Ensemble of many networks

Ensemble of many smaller networks

- At the end of SGD training, the final network approximates the average prediction over all members of the ensemble.
- Caveat: each member of the ensemble is constrained to *share the same weights* with all other members.



Ensemble of many networks