CS/DS 541: Class 14

Jacob Whitehill

Regularization

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

- Let's consider just a 2-layer (linear regression) NN.

- Instead of the sample mean, the loss is the *expected* squared distance between the predictions & target labels.

- We define separate loss 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:

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

L_2 regularization

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 i^{th} NN's prediction for **x**.
  - ε_i is the error of NN i's prediction for **x**.
- 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:

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

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

Ensembles

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

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

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

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

Ensembles

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

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

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

$$= \frac{1}{n^2}\mathbb{E}\begin{bmatrix} \epsilon_1\epsilon_1+\epsilon_1\epsilon_2+\ldots+\epsilon_1\epsilon_n+ \\ \ldots \\ +\epsilon_n\epsilon_1+\epsilon_n\epsilon_2+\ldots+\epsilon_1\epsilon_n \end{bmatrix}$$

$$= \frac{1}{n^2}\begin{bmatrix} v+c+\ldots+c+ \\ \ldots \\ +c+\ldots+c+v \end{bmatrix}$$

Ensembles

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

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

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

$$= \frac{1}{n^2}\mathbb{E}\begin{bmatrix} \epsilon_1\epsilon_1+\epsilon_1\epsilon_2+\ldots+\epsilon_1\epsilon_n+ \\ \ldots \\ +\epsilon_n\epsilon_1+\epsilon_n\epsilon_2+\ldots+\epsilon_1\epsilon_n \end{bmatrix}$$

$$= \frac{1}{n^2}\begin{bmatrix} v+c+\ldots+c+ \\ \ldots \\ +c+\ldots+c+v \end{bmatrix}$$

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

Ensembles

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

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

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

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

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

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

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

Ensembles

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

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

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

$$= \frac{1}{n^2}\mathbb{E}\begin{bmatrix}\epsilon_1\epsilon_1+\epsilon_1\epsilon_2+\ldots+\epsilon_1\epsilon_n+\\ \ldots \\ +\epsilon_n\epsilon_1+\epsilon_n\epsilon_2+\ldots+\epsilon_1\epsilon_n\end{bmatrix}$$

$$= \frac{1}{n^2}\begin{bmatrix}v+c+\ldots+c+\\ \ldots \\ +c+\ldots+c+v\end{bmatrix}$$

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

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

Ensembles

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

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

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

$$=\frac{1}{n^2}\mathbb{E}\begin{bmatrix}\epsilon_1\epsilon_1+\epsilon_1\epsilon_2+\ldots+\epsilon_1\epsilon_n+\\ \ldots\\ +\epsilon_n\epsilon_1+\epsilon_n\epsilon_2+\ldots+\epsilon_1\epsilon_n\end{bmatrix}$$

$$=\frac{1}{n^2}\begin{bmatrix}v+c+\ldots+c+\\ \ldots\\ +c+\ldots+c+v\end{bmatrix}$$

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

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

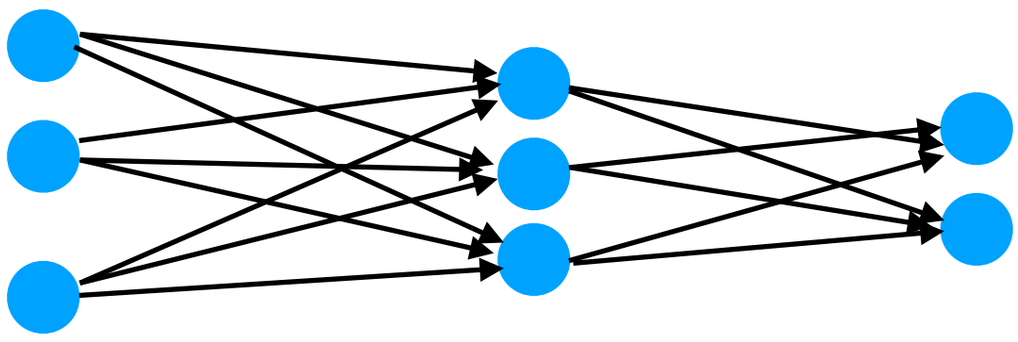
Dropout

- 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

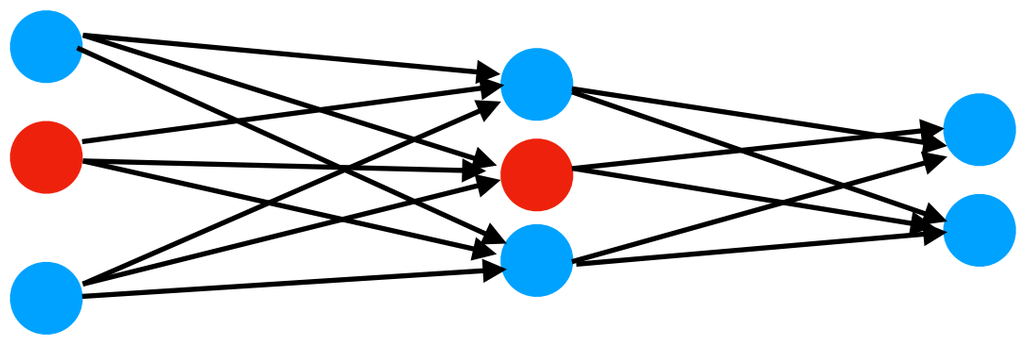
- Suppose we are training the NN shown below:



Dropout

- Suppose we are training the NN shown below:

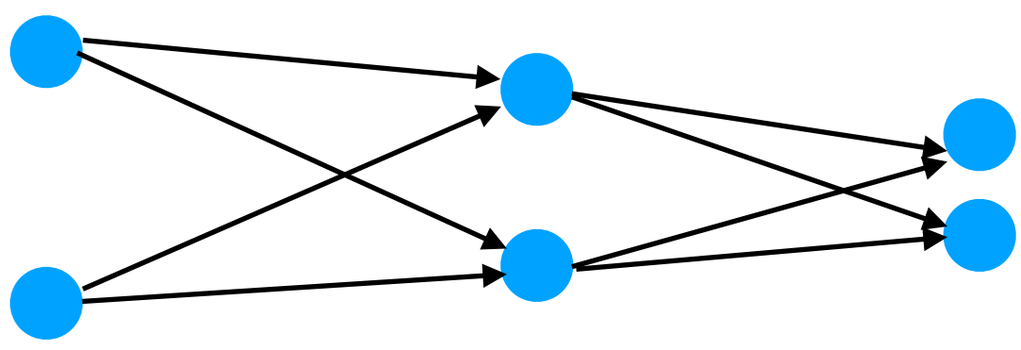
- For each step of SGD, we randomly select (with “keep” probability *p*) some of the input and hidden neurons (*not* the output neurons).



Dropout

- Suppose we are training the NN shown below:

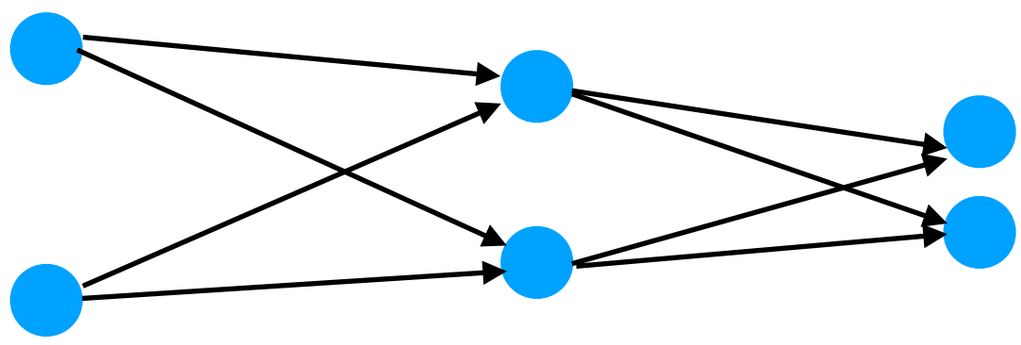
- We then remove these neurons and perform forward-propagation on the reduced network.



Dropout

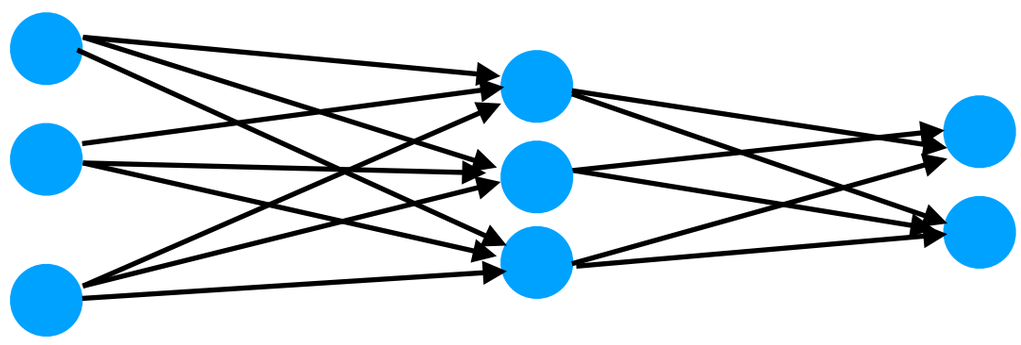
- Suppose we are training the NN shown below:

- 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.)

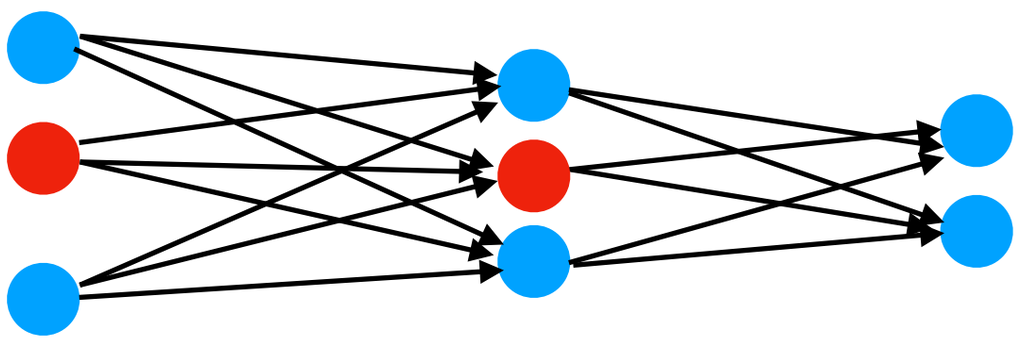


Dropout: example

- 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 $\mathbf{b}^{(1)}=\mathbf{b}^{(2)}=0$.)

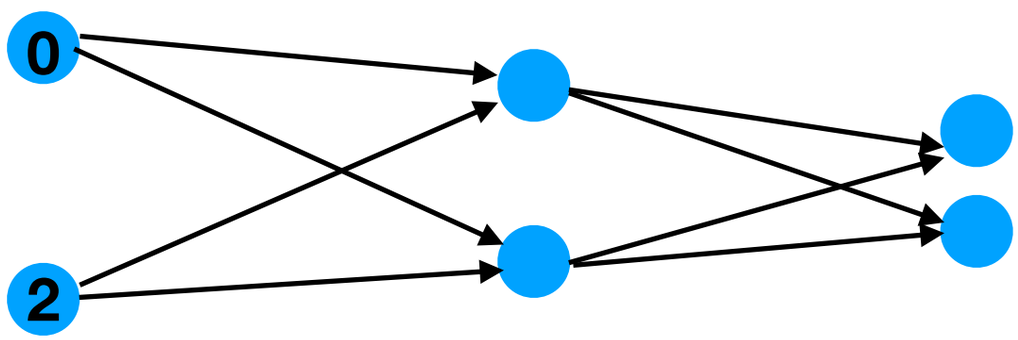


Dropout: example

- 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.

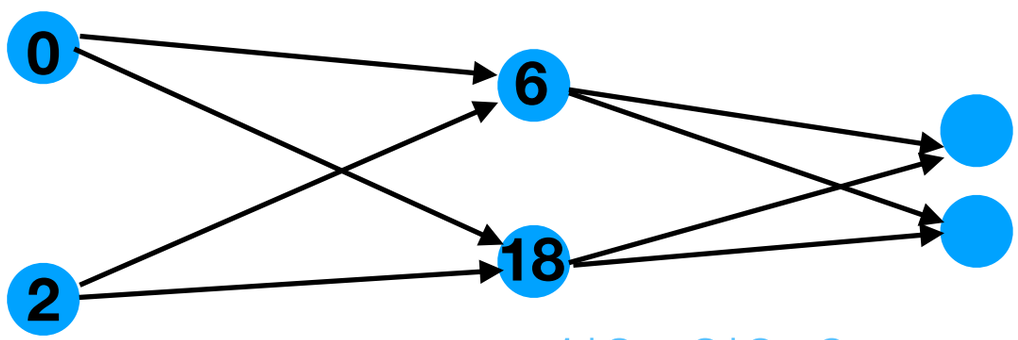


Dropout: example

- 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.

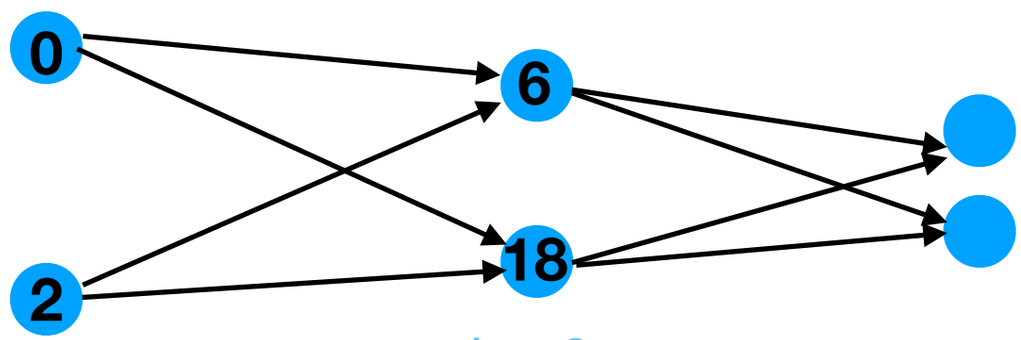


Dropout: example

- 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.

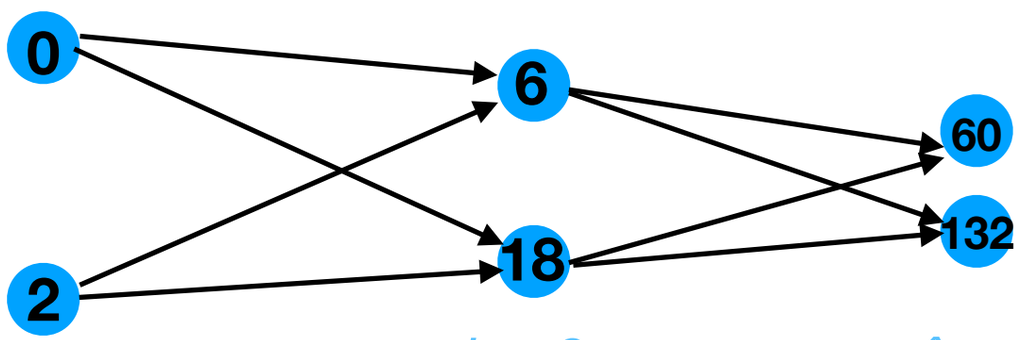


Dropout: example

- 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.

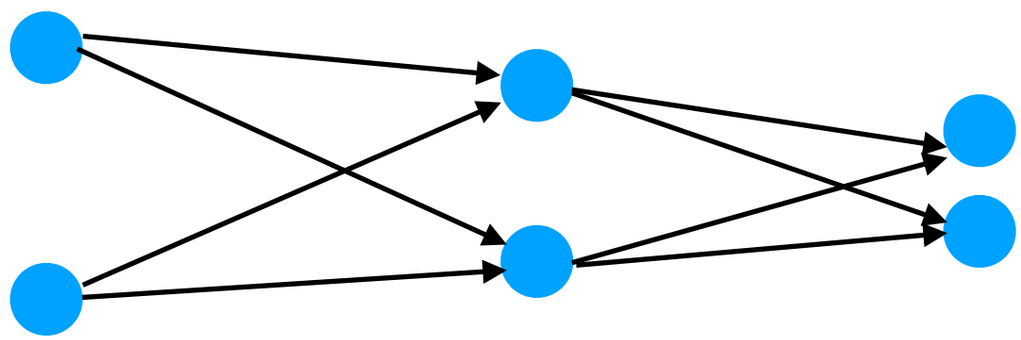


Dropout: example

- 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?

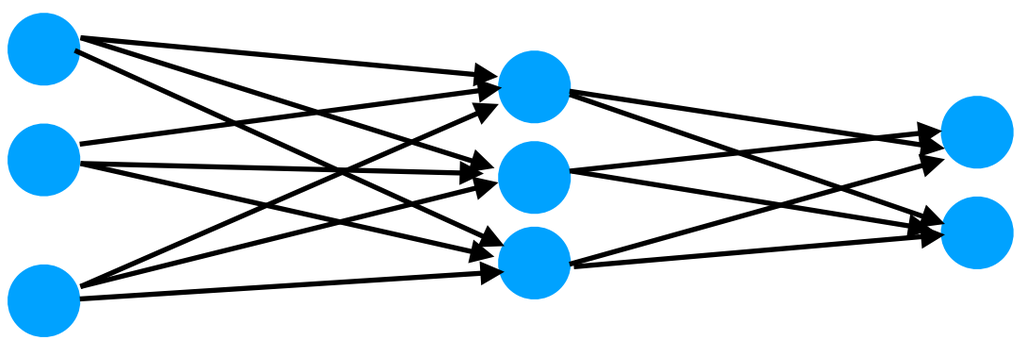
- There are two main explanations for why dropout helps improve the accuracy of neural networks:

- 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.

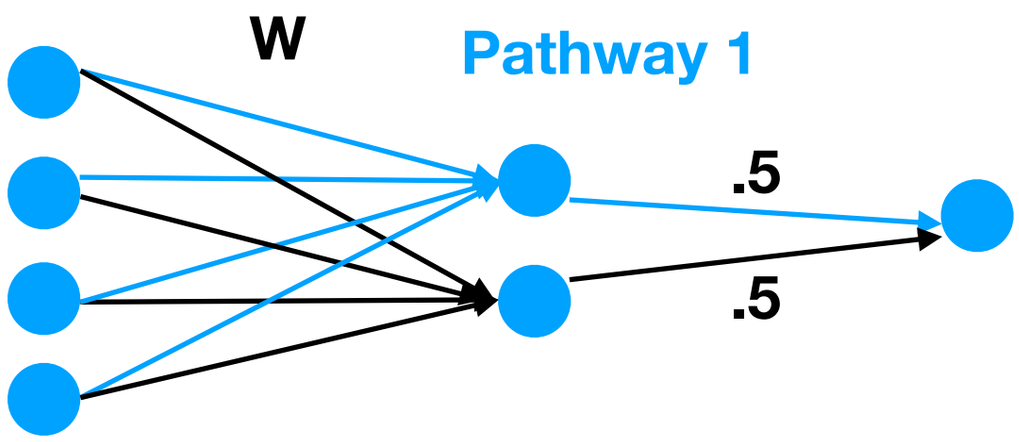
Symmetry breaking

- 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.

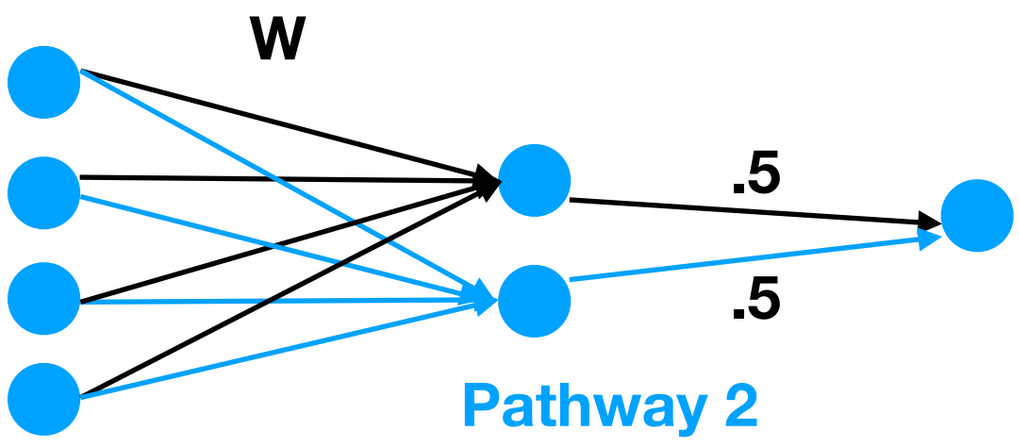
Co-adaptation

- 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.

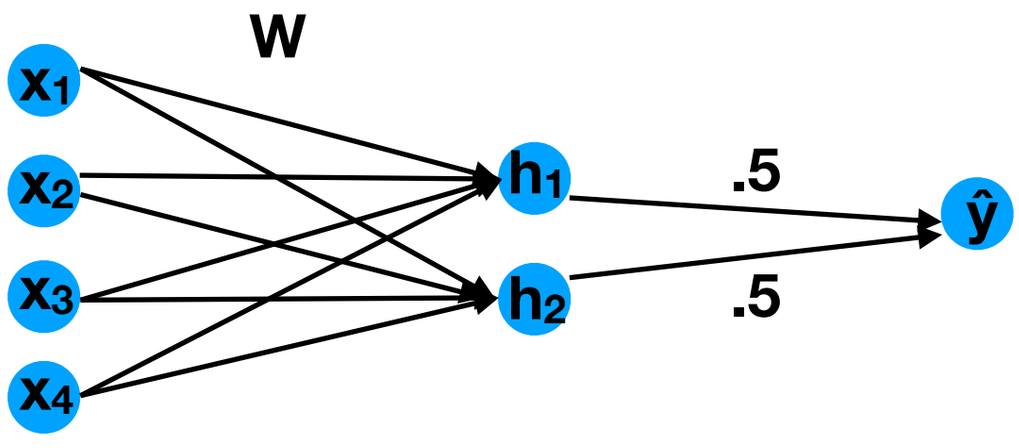


Co-adaptation

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

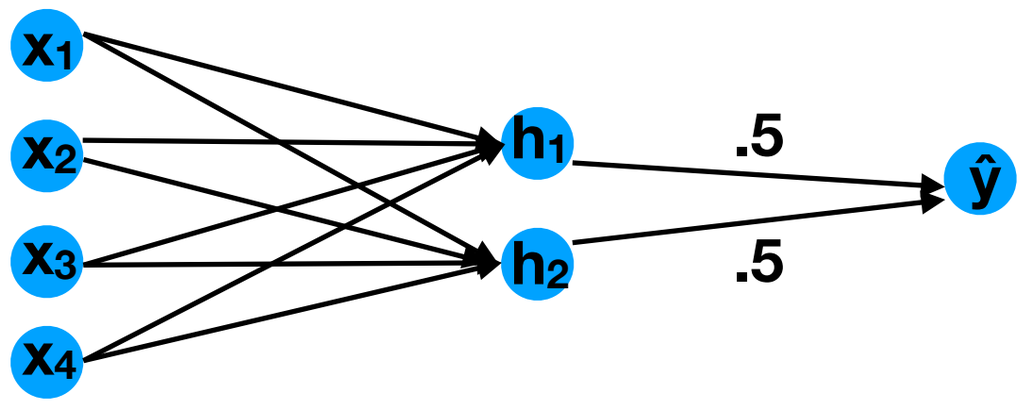
  then $\mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} 2x_1 - 3x_2 + x_3 - x_4 \\ 2x_1 - 3x_2 - x_3 + x_4 \end{bmatrix}$

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



Co-adaptation

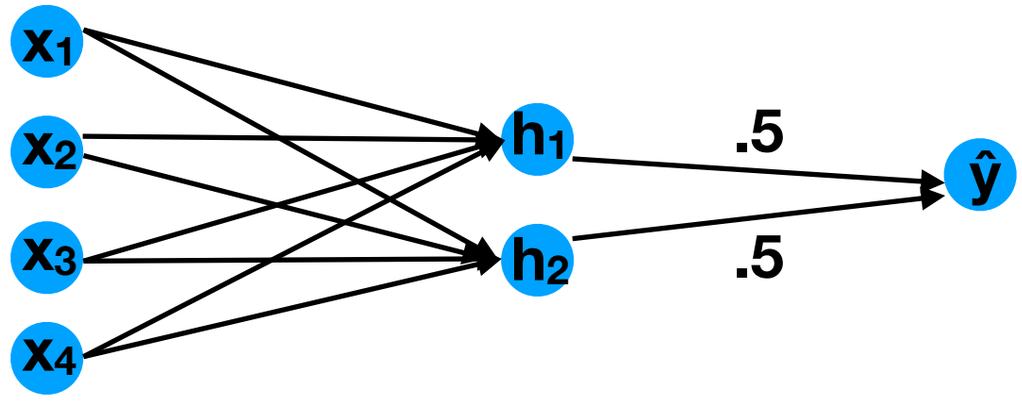
- At this point, neither x_3 nor x_4 have any impact on $\hat{y}$.
- We may be at a local minimum where the last two columns of **W** stay "locked" to effectively delete x_3, x_4.
- This is an example of **weight co-adaptation**; it is often a suboptimal solution.



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

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”.

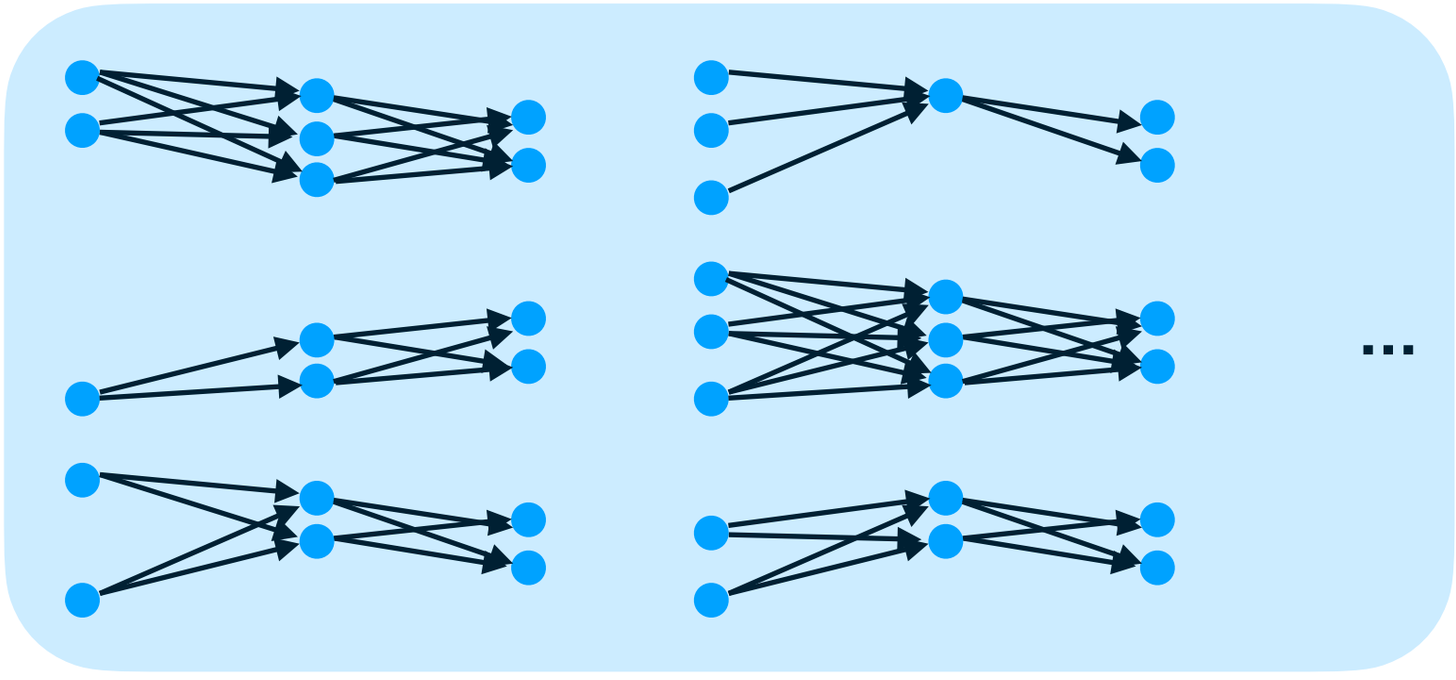


$$\mathbf{W} = \begin{bmatrix} 2 & -3 & 1 & -1 \\ 2 & -3 & -1 & 1 \end{bmatrix}$$

Ensemble of many smaller networks

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

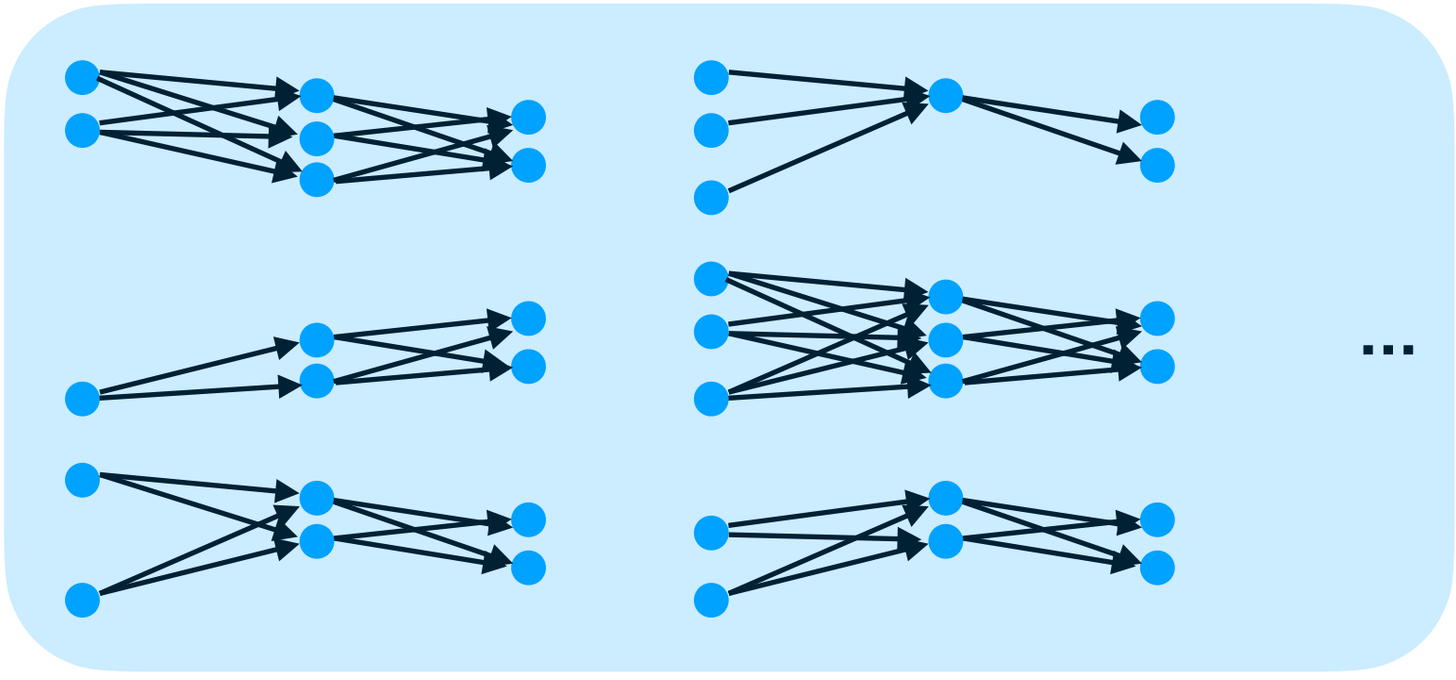
- Each member of the ensemble arises by randomly dropping some of the whole network's neurons:



Ensemble of many 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