CS 4342: Class 22

Jacob Whitehill

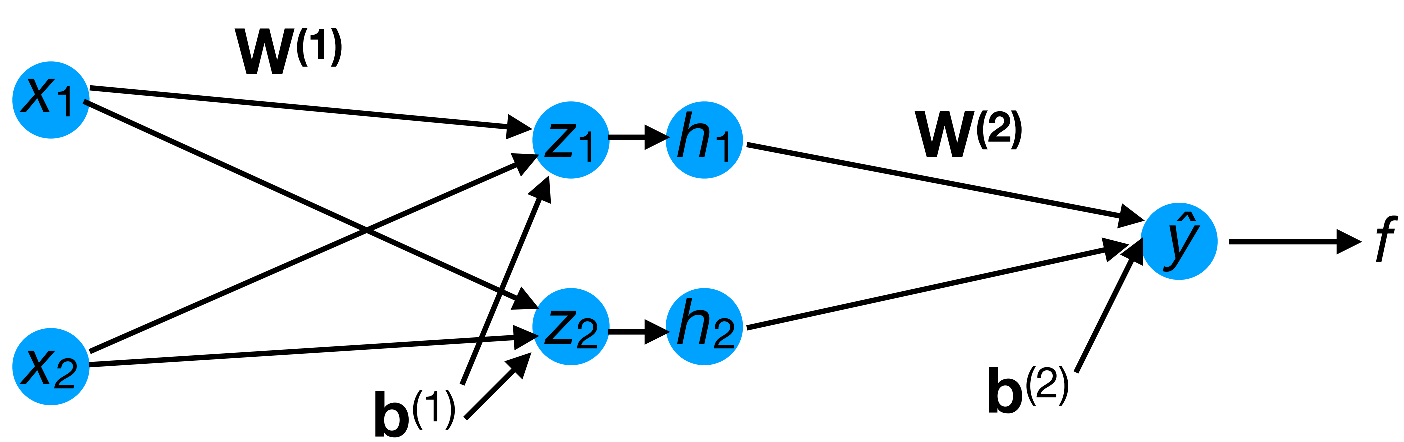
Computing the gradients

- Where do these come from?

$$
\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) {\mathbf{h}^{(1)}}^\top \\
\nabla_{\mathbf{b}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) \\
\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}\mathbf{x}^\top \\
\nabla_{\mathbf{b}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}
\end{aligned}
$$

where

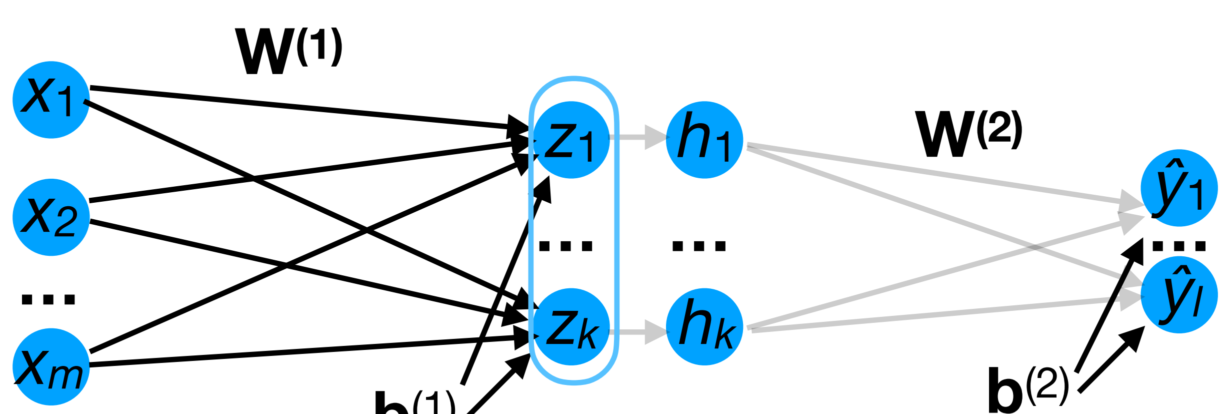
$$
\mathbf{g}^\top = \left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)} \right) \odot \mathrm{relu}'({\mathbf{z}^{(1)}}^\top)
$$



Forwards and backwards propagation

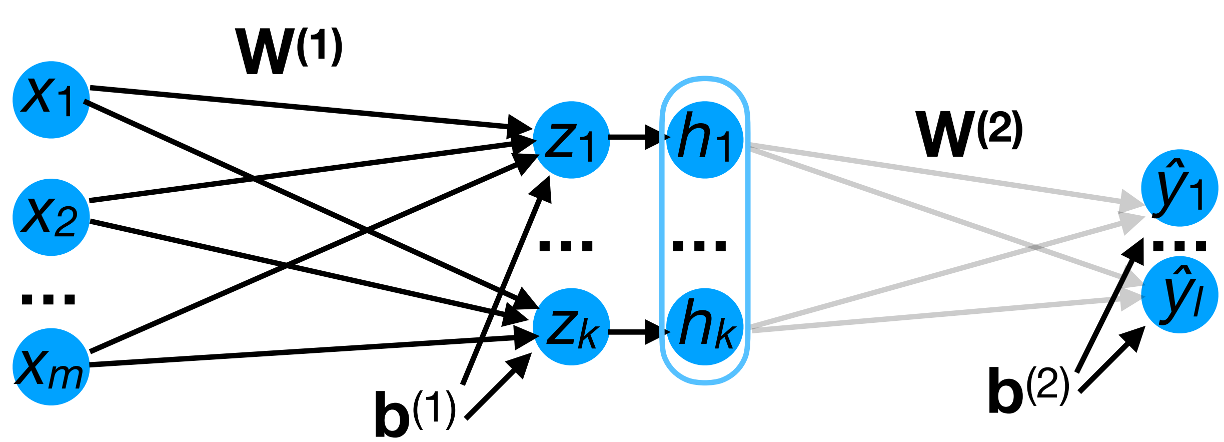
Computing the gradients

- Consider the 3-layer NN below:
  - From **x**, **W**$^{(1)}$, and **b**$^{(1)}$, we can compute **z**.



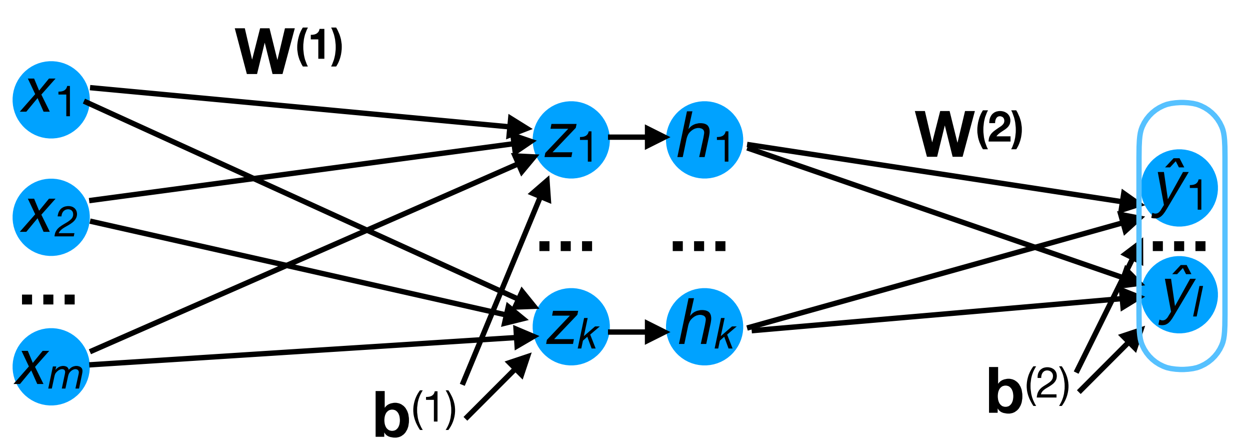
Computing the gradients

- Consider the 3-layer NN below:
  - From **x**, $\mathbf{W}^{(1)}$, and $\mathbf{b}^{(1)}$, we can compute **z**.
  - From **z** and σ, we can compute $\mathbf{h} = \sigma(\mathbf{z})$.



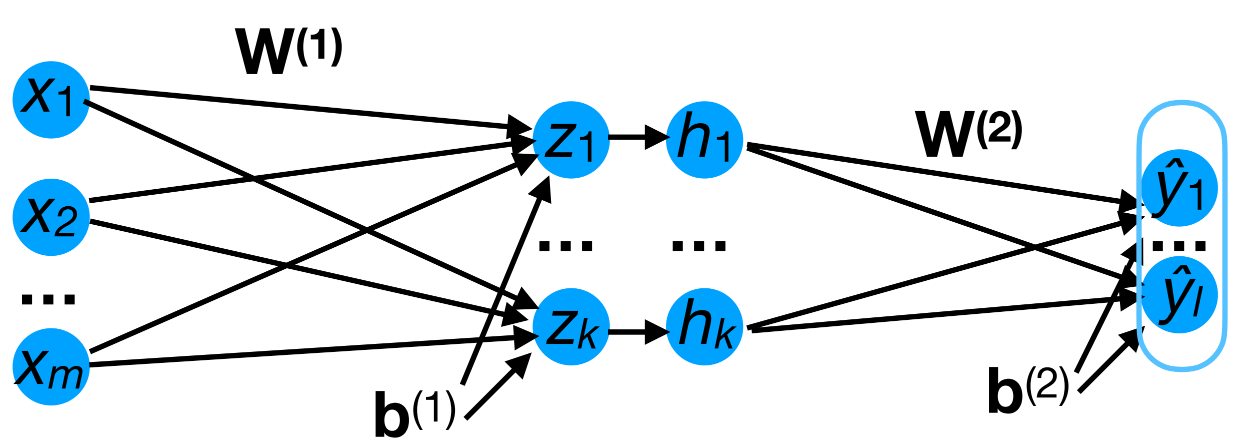
Computing the gradients

- Consider the 3-layer NN below:
  - From $\mathbf{x}$, $\mathbf{W}^{(1)}$, and $\mathbf{b}^{(1)}$, we can compute $\mathbf{z}$.
  - From $\mathbf{z}$ and σ, we can compute $\mathbf{h} = \sigma(\mathbf{z})$.
  - From $\mathbf{h}$, $\mathbf{W}^{(2)}$, and $\mathbf{b}^{(2)}$, we can compute $\hat{\mathbf{y}}$.



Computing the gradients

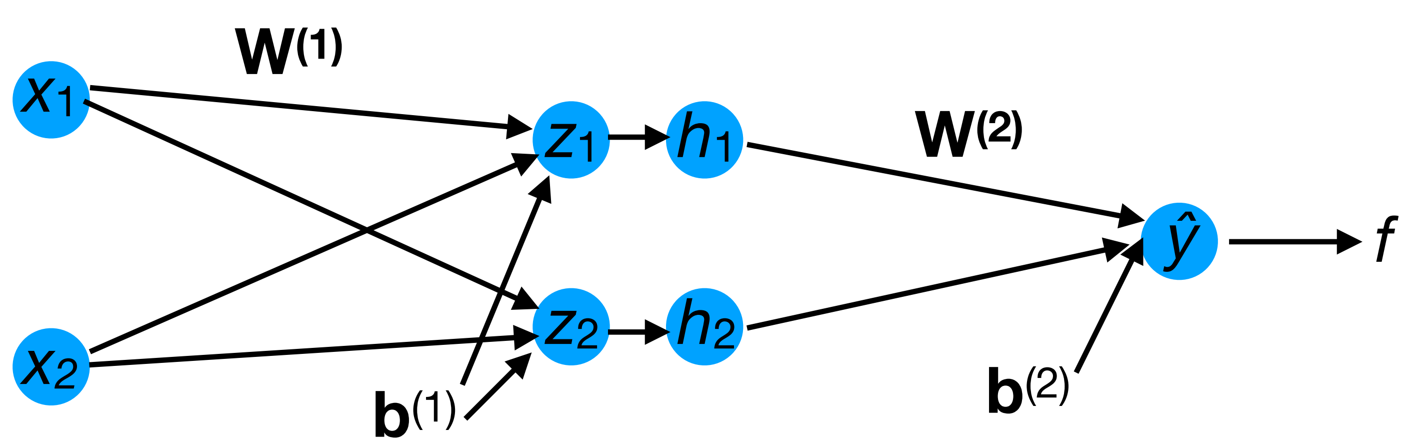
- This process is known as **forward propagation**.
  - It produces all the intermediary (**h**, **z**) and final ($\hat{\mathbf{y}}$) network outputs.



Computing the gradients

- Now, let's look at how to compute each gradient term:

$$
\begin{aligned}
\frac{\partial f}{\partial \mathbf{W}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}} \\
\frac{\partial f}{\partial \mathbf{b}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}} \\
\frac{\partial f}{\partial \mathbf{W}^{(1)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}} \\
\frac{\partial f}{\partial \mathbf{b}^{(1)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}^{(1)}}
\end{aligned}
$$

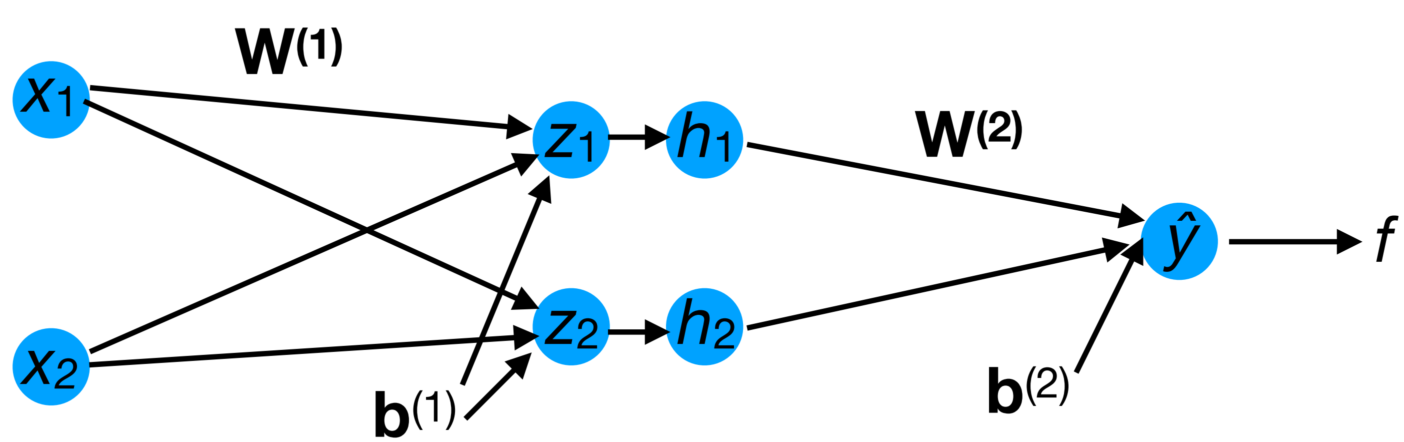


Computing the gradients

- Now, let's look at how to compute each gradient term:

$$
\begin{aligned}
\frac{\partial f}{\partial \mathbf{W}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}} \\
\frac{\partial f}{\partial \mathbf{b}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}} \\
\frac{\partial f}{\partial \mathbf{W}^{(1)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}} \\
\frac{\partial f}{\partial \mathbf{b}^{(1)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}^{(1)}}
\end{aligned}
$$

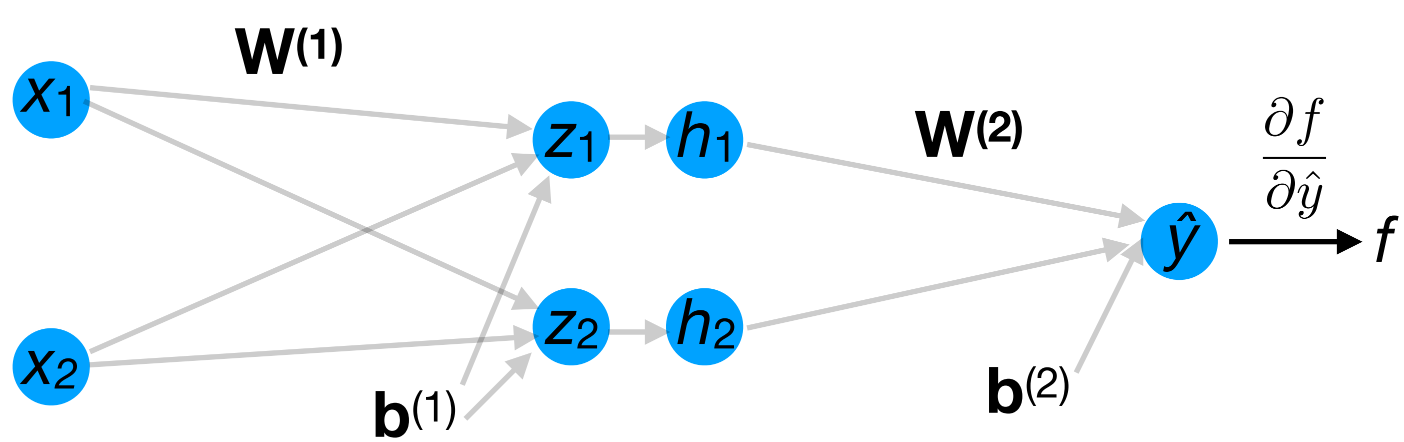
Redundant computation



Computing the gradients

- Here's how we can compute all these *efficiently*:

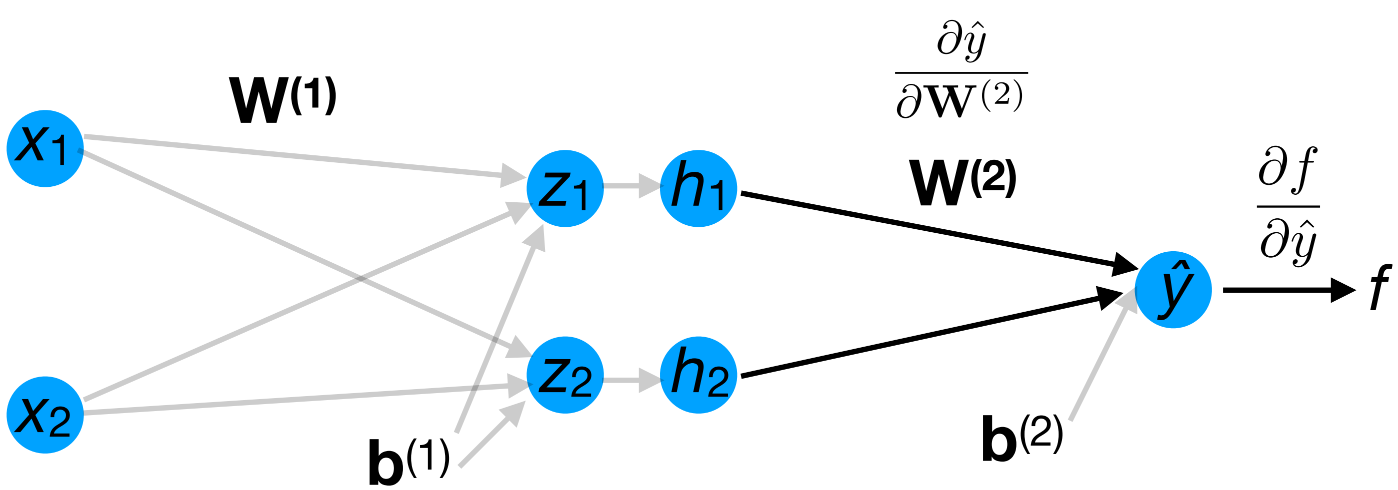
$$\frac{\partial f}{\partial \mathbf{W}^{(2)}} = \frac{\partial f}{\partial \hat{y}}$$



Computing the gradients

- Here's how we can compute all these *efficiently*:

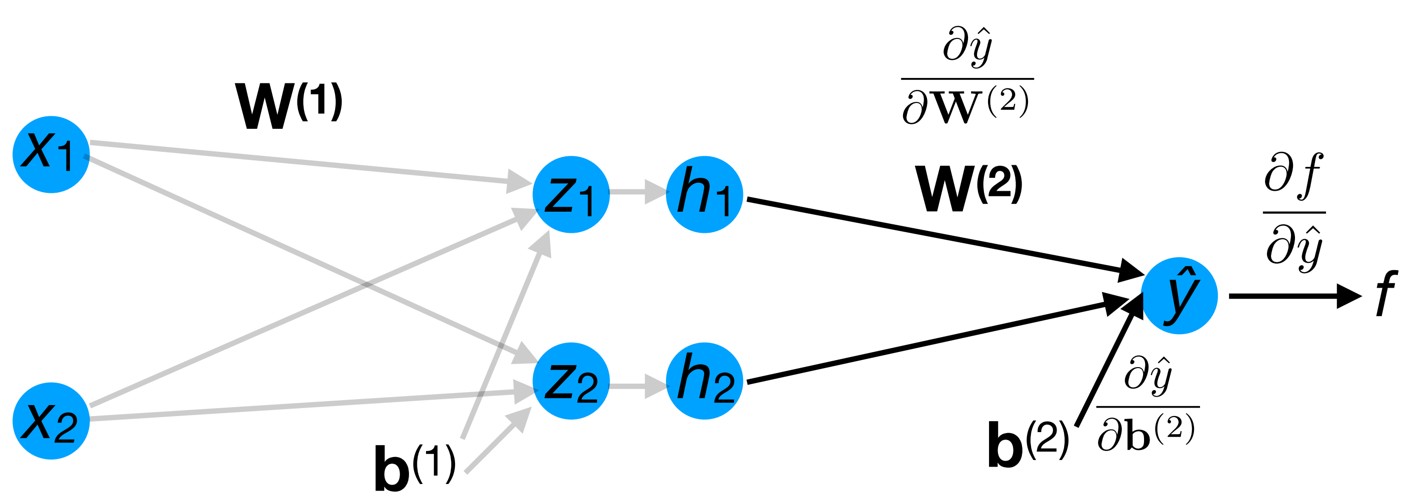
$$\frac{\partial f}{\partial \mathbf{W}^{(2)}} = \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}}$$



Computing the gradients

- Here's how we can compute all these *efficiently*:

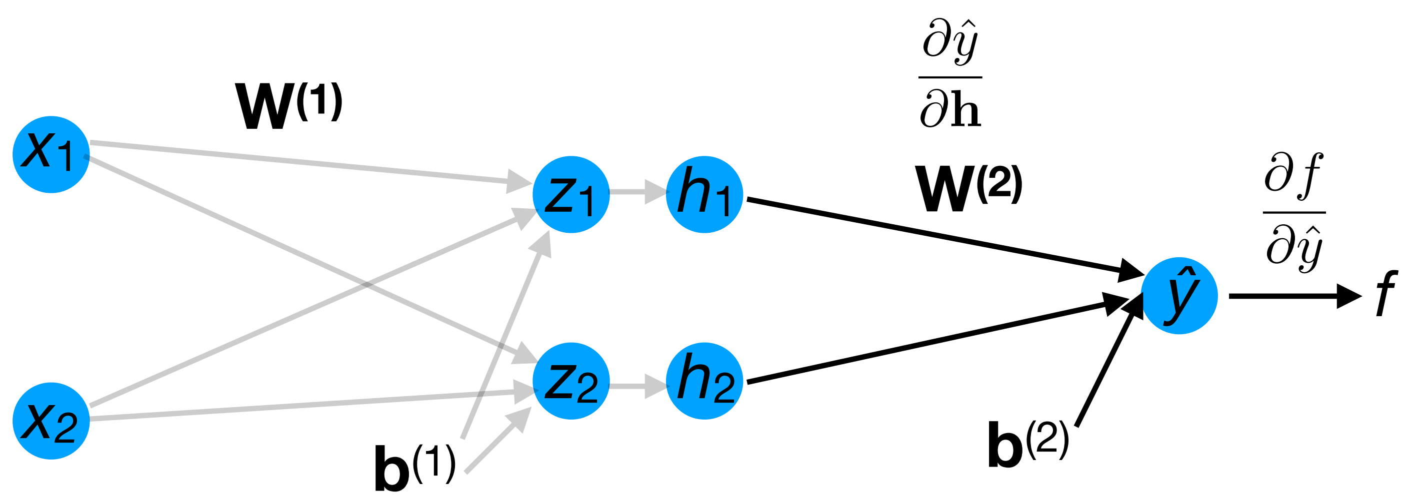
$$\frac{\partial f}{\partial \mathbf{W}^{(2)}} = \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}}$$
$$\frac{\partial f}{\partial \mathbf{b}^{(2)}} = \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}}$$



Computing the gradients

- Here's how we can compute all these *efficiently*:

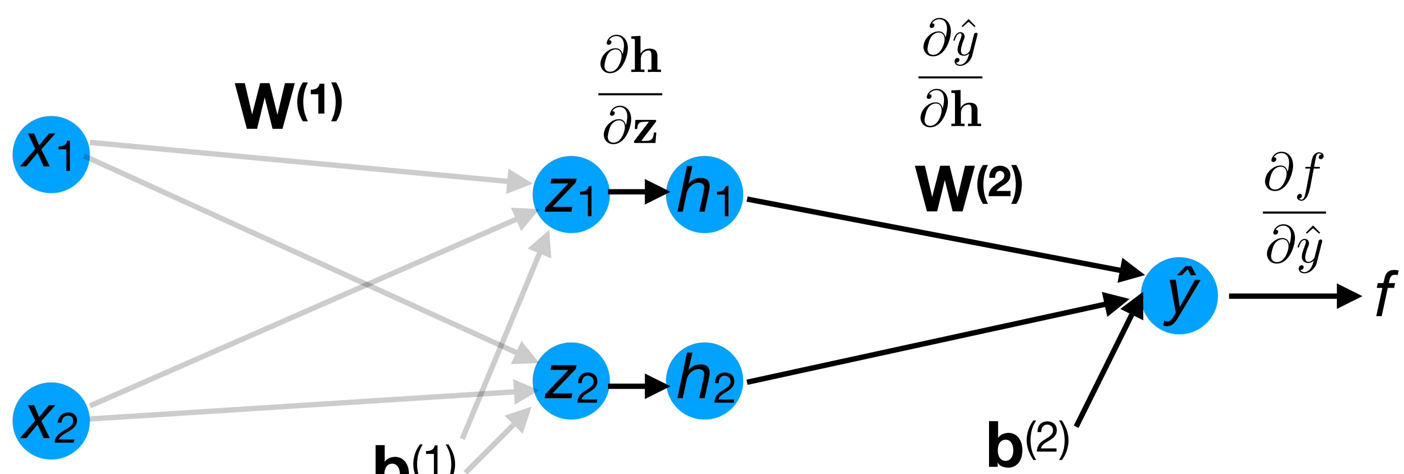
$$\frac{\partial f}{\partial \mathbf{W}^{(2)}} = \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}}$$
$$\frac{\partial f}{\partial \mathbf{b}^{(2)}} = \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}}$$
$$\frac{\partial f}{\partial \mathbf{W}^{(1)}} = \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{h}}$$



Computing the gradients

- Here's how we can compute all these *efficiently*:

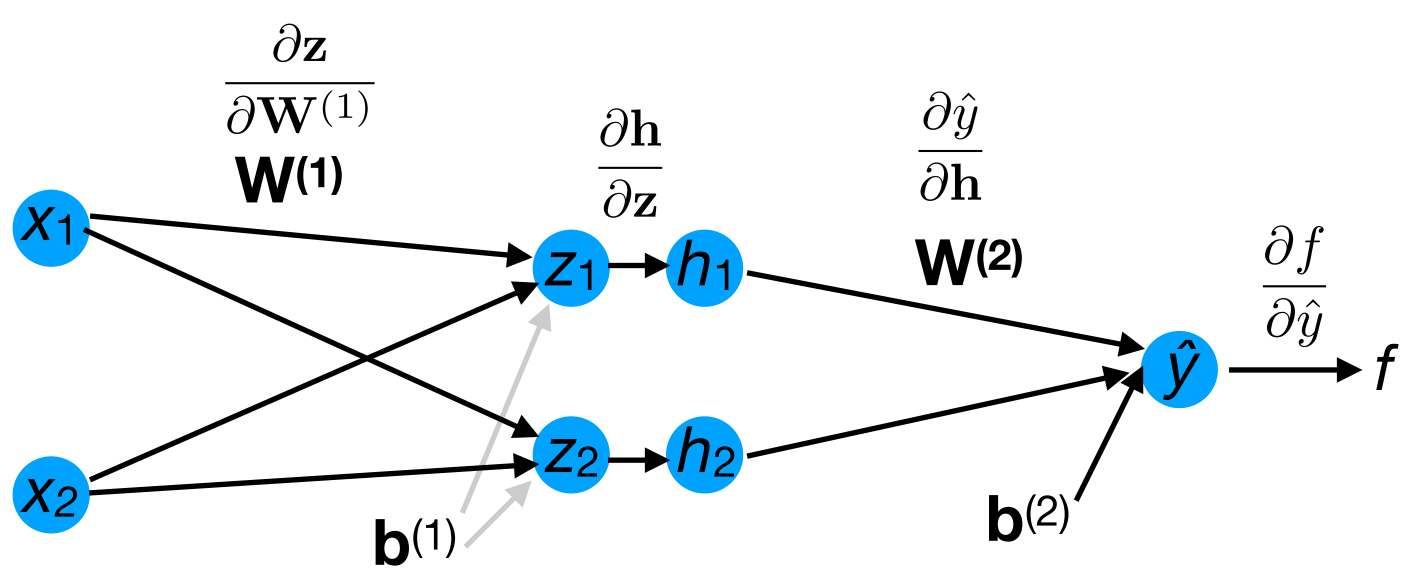
$$
\begin{aligned}
\frac{\partial f}{\partial \mathbf{W}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}} \\
\frac{\partial f}{\partial \mathbf{b}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}} \\
\frac{\partial f}{\partial \mathbf{W}^{(1)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}}
\end{aligned}
$$



Computing the gradients

- Here's how we can compute all these *efficiently*:

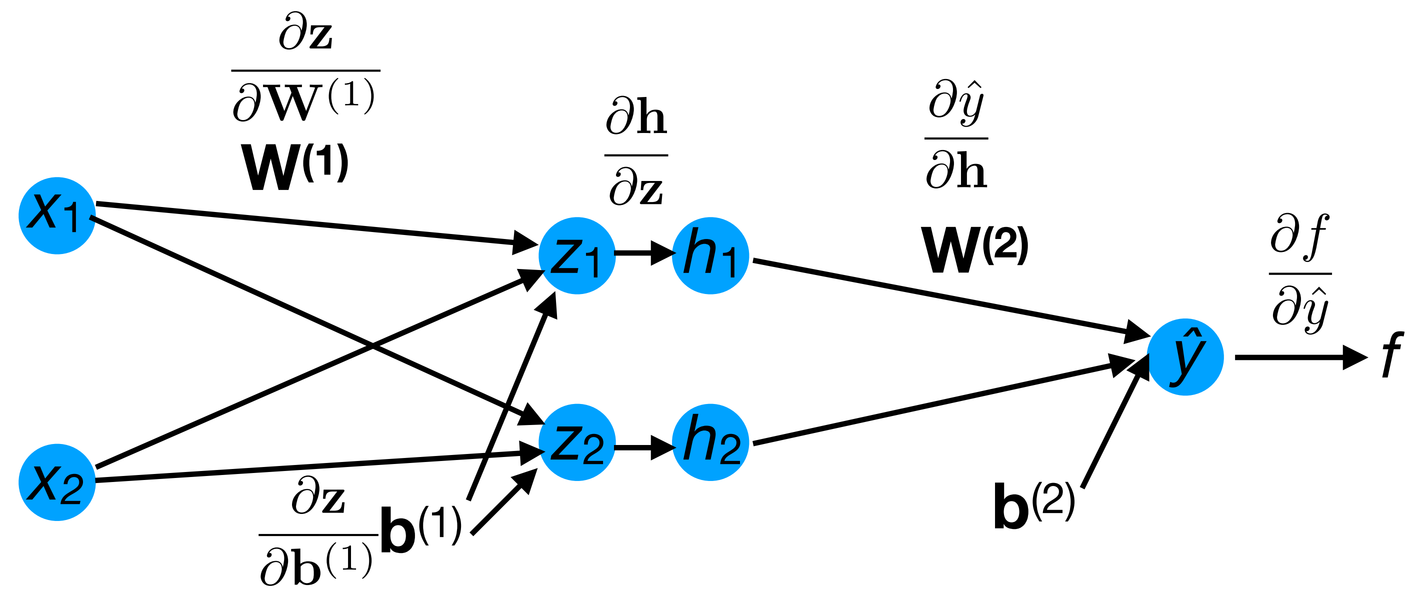
$$\begin{aligned}
\frac{\partial f}{\partial \mathbf{W}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}} \\
\frac{\partial f}{\partial \mathbf{b}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}} \\
\frac{\partial f}{\partial \mathbf{W}^{(1)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}}
\end{aligned}$$



Computing the gradients

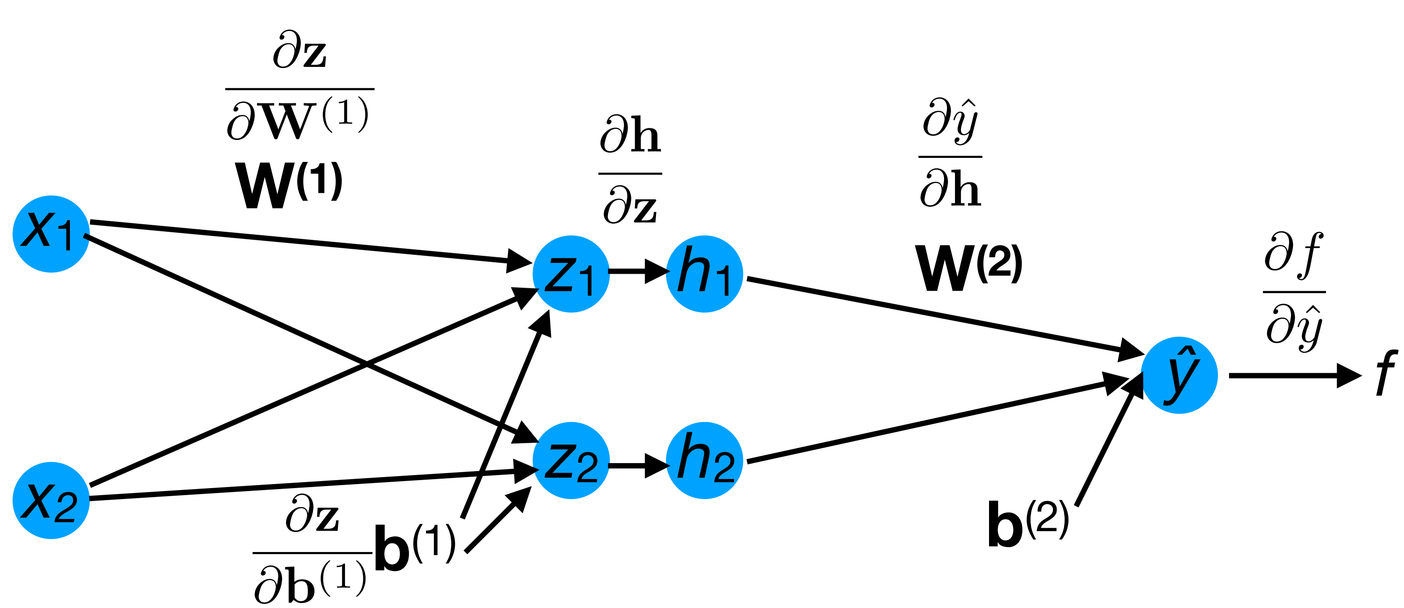
- Here's how we can compute all these *efficiently*:

$$
\begin{aligned}
\frac{\partial f}{\partial \mathbf{W}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}} \\
\frac{\partial f}{\partial \mathbf{b}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}} \\
\frac{\partial f}{\partial \mathbf{W}^{(1)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}} \\
\frac{\partial f}{\partial \mathbf{b}^{(1)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}^{(1)}}
\end{aligned}
$$



Computing the gradients

- This process is known as **backwards propagation ("backprop"):**
  - It produces the gradient terms of all the weight matrices and bias vectors.
  - It requires first conducting forward propagation.



Weight initialization

NNs and convexity

- Neural networks are a non-convex ML model.
- Hence, the values you use to initialize the weights and bias terms can make a big difference on the accuracy of the network.
  - The network might end up in a worse local minimum (or saddle points) of the cost function.

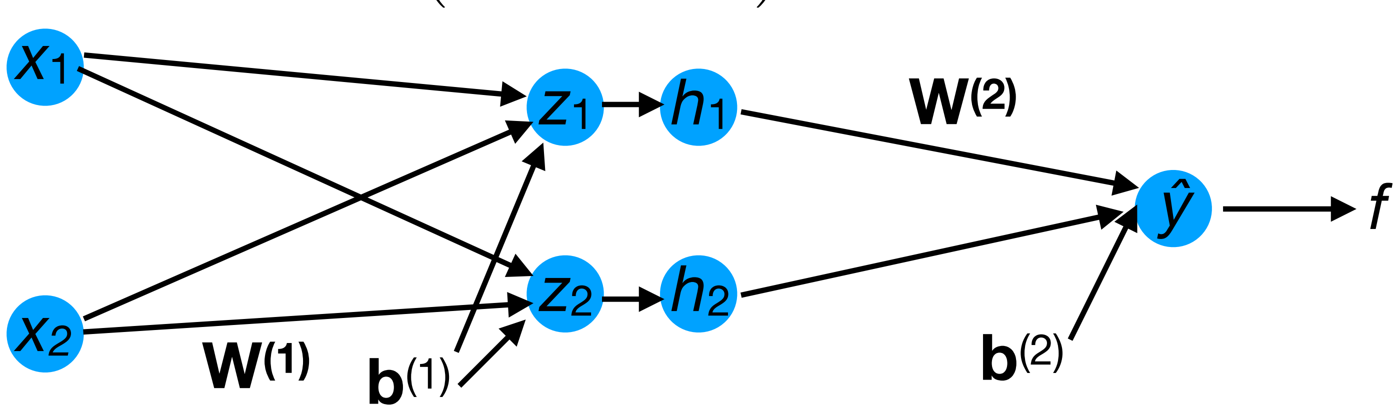
Weight initialization: example

- Suppose we initialize all the weights and bias terms of a 3-layer NN to be 0.

- What will happen during SGD?

$$
\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}){\mathbf{h}^{(1)}}^\top \\
\nabla_{\mathbf{b}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) \\
\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}\mathbf{x}^\top \\
\nabla_{\mathbf{b}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}
\end{aligned}
$$

$$
\mathbf{g}^\top = \left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)}\right) \odot \mathrm{relu}'({\mathbf{z}^{(1)}}^\top)
$$



Weight initialization: example

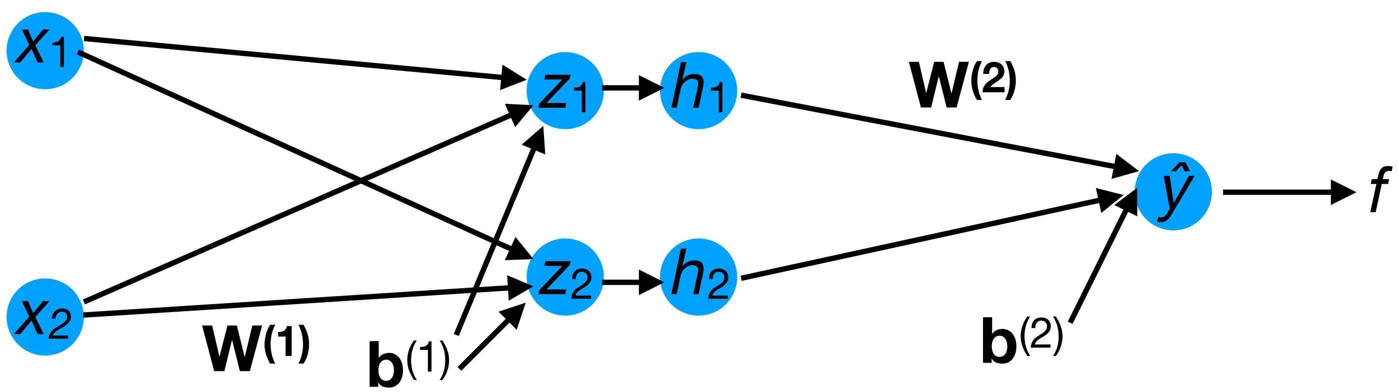
- Suppose we initialize all the weights and bias terms of a 3-layer NN to be 0.

- What will happen during SGD?

During forwards propagation, z and h will be 0. Hence, ŷ will also be 0.

$$
\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}){\mathbf{h}^{(1)}}^\top \\
\nabla_{\mathbf{b}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) \\
\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}\mathbf{x}^\top \\
\nabla_{\mathbf{b}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}
\end{aligned}
$$

$$\mathbf{g}^\top = \left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)}\right) \odot \mathrm{relu}'({\mathbf{z}^{(1)}}^\top)$$



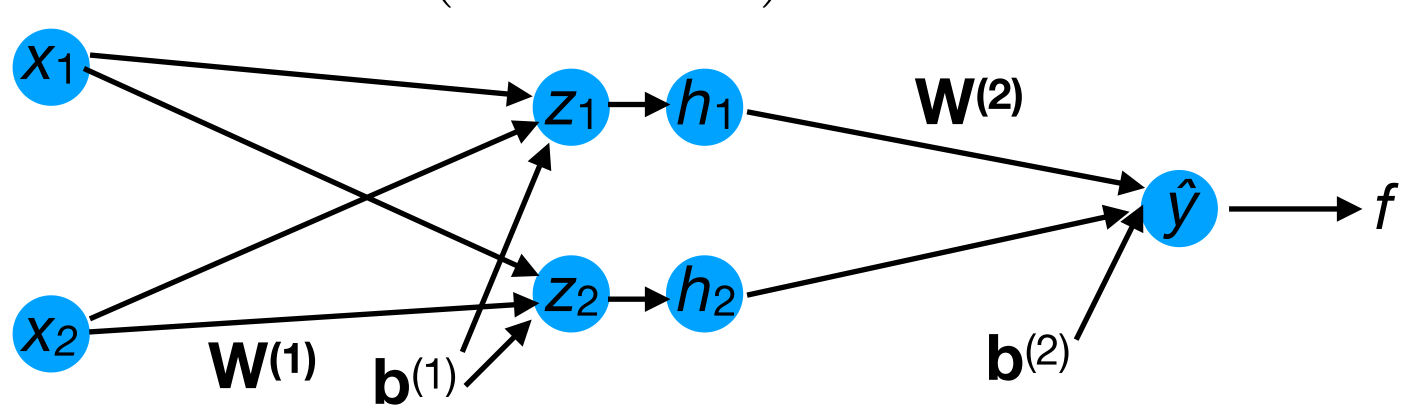
Weight initialization: example

- Suppose we initialize all the weights and bias terms of a 3-layer NN to be 0.
- What will happen during SGD?

During backwards propagation, we have:

$$\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\text{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}){\mathbf{h}^{(1)}}^\top \\
\nabla_{\mathbf{b}^{(2)}} f_{\text{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) \\
\nabla_{\mathbf{W}^{(1)}} f_{\text{CE}} &= \mathbf{g}\mathbf{x}^\top \\
\nabla_{\mathbf{b}^{(1)}} f_{\text{CE}} &= \mathbf{g}
\end{aligned}$$

$$\mathbf{g}^\top = \left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)}\right) \odot \text{relu}'({\mathbf{z}^{(1)}}^\top)$$



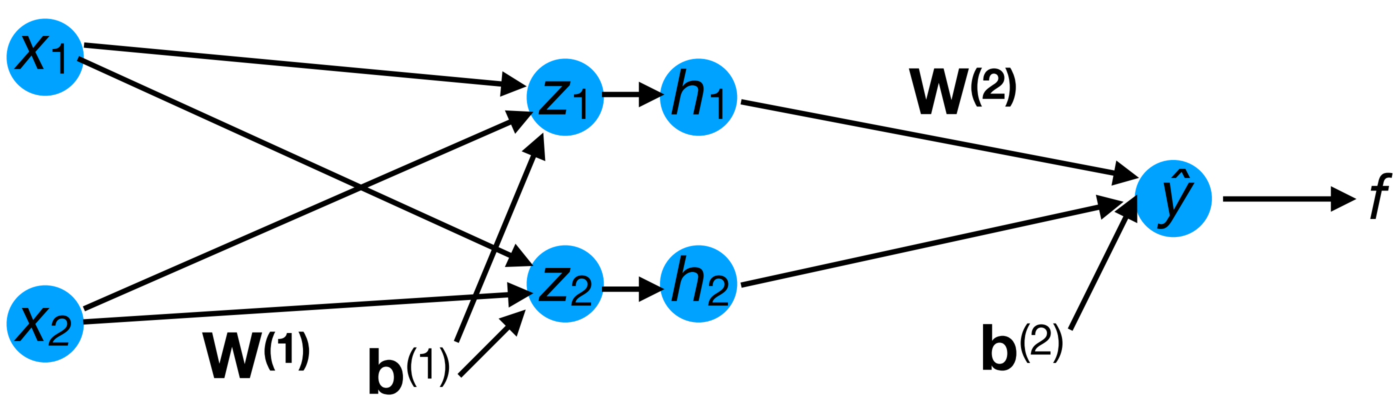
Weight initialization: example

- Suppose we initialize all the weights and bias terms of a 3-layer NN to be 0.
- What will happen during SGD?

During backwards propagation, we have:

$$\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y})\mathbf{h}^{(1)\top} \quad \mathbf{0} \\
\nabla_{\mathbf{b}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) \\
\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}\mathbf{x}^\top \quad \mathbf{0} \\
\nabla_{\mathbf{b}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g} \quad \mathbf{0}
\end{aligned}$$

$$\mathbf{g}^\top = \left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)}\right) \odot \mathrm{relu}'(\mathbf{z}^{(1)\top}) \quad \mathbf{0}$$



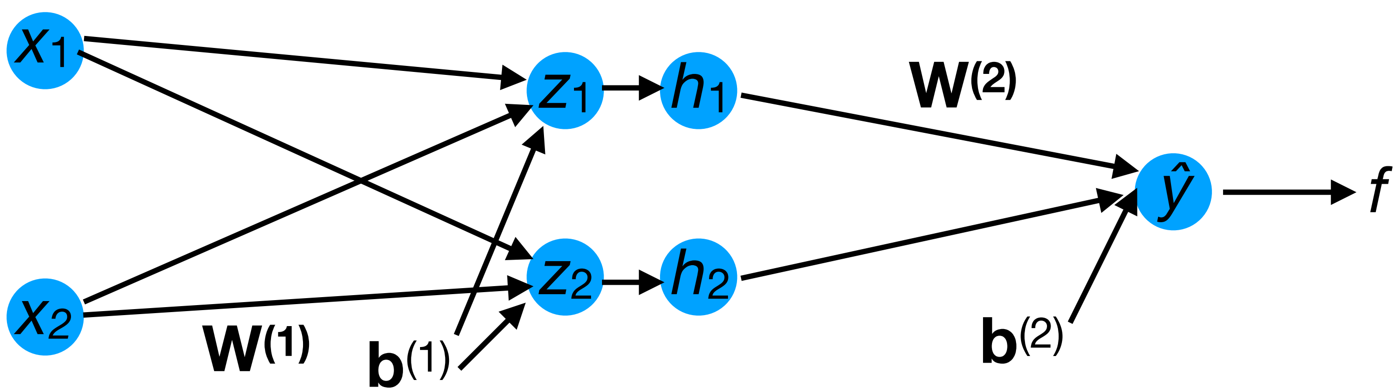
Weight initialization: example

- Because the gradients w.r.t. $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$, and $\mathbf{b}^{(1)}$ are all 0, they will *never change*.
- Only $\mathbf{b}^{(2)}$ will change (to the mean of the target values y).

During backwards propagation, we have:

$$
\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) {\mathbf{h}^{(1)}}^\top \quad \mathbf{0} \\
\nabla_{\mathbf{b}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) \\
\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}\mathbf{x}^\top \quad \mathbf{0} \\
\nabla_{\mathbf{b}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g} \quad \mathbf{0}
\end{aligned}
$$

$$
\mathbf{g}^\top = \left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)} \right) \odot \mathrm{relu}'({\mathbf{z}^{(1)}}^\top) \quad \mathbf{0}
$$

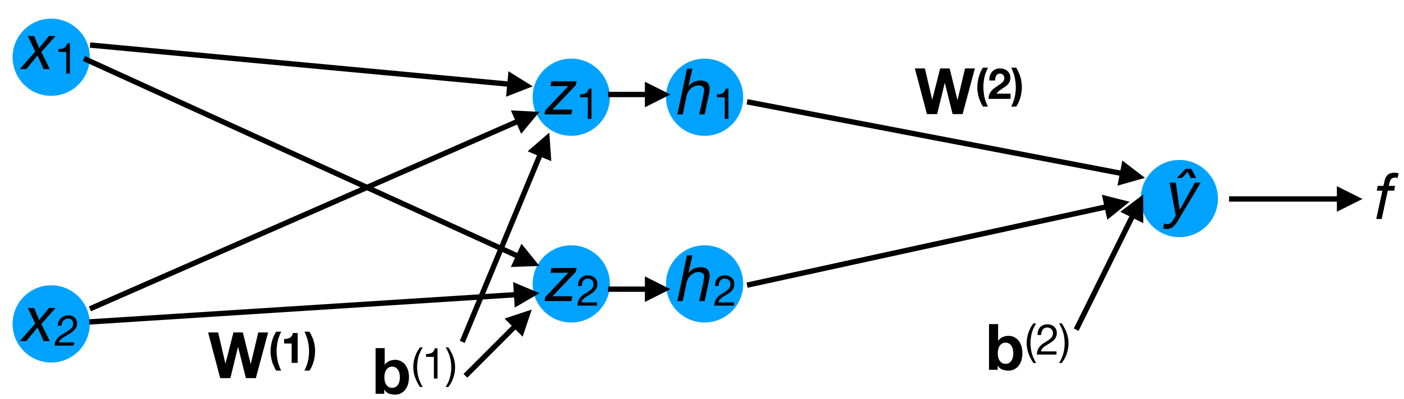


Weight initialization: exercise 1

- Suppose we initialize $\mathbf{W}^{(1)}=\mathbf{b}^{(1)}=0$, but $\mathbf{W}^{(2)}$, $\mathbf{b}^{(2)}$ are non-zero.
- Assume relu'(0)=0.
- What will happen during SGD?

$$
\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y})\mathbf{h}^{(1)^\top} \\
\nabla_{\mathbf{b}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) \\
\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}\mathbf{x}^\top \\
\nabla_{\mathbf{b}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}
\end{aligned}
$$

$$
\mathbf{g}^\top = \left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)}\right) \odot \mathrm{relu}'(\mathbf{z}^{(1)^\top})
$$

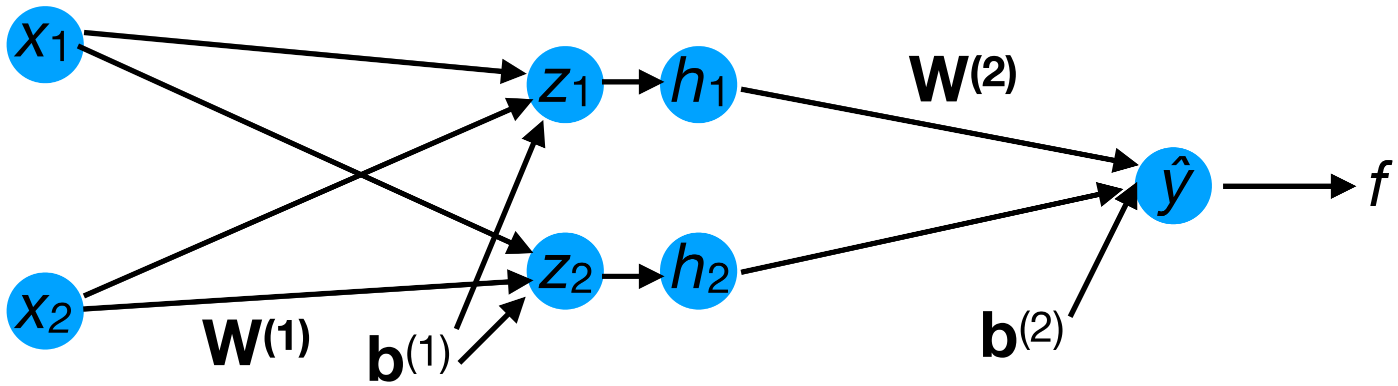


Weight initialization: exercise 1

- Since $\mathbf{W}^{(1)}=\mathbf{b}^{(1)}=0$, then $\mathbf{z}^{(1)}=\mathbf{h}^{(1)}=0$. Hence, $\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} = 0$.

$$
\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y})\mathbf{h}^{(1)^\top} \quad \mathbf{0} \\
\nabla_{\mathbf{b}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) \\
\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}\mathbf{x}^\top \\
\nabla_{\mathbf{b}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}
\end{aligned}
$$

$$
\mathbf{g}^\top = \left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)}\right) \odot \mathrm{relu}'(\mathbf{z}^{(1)^\top})
$$

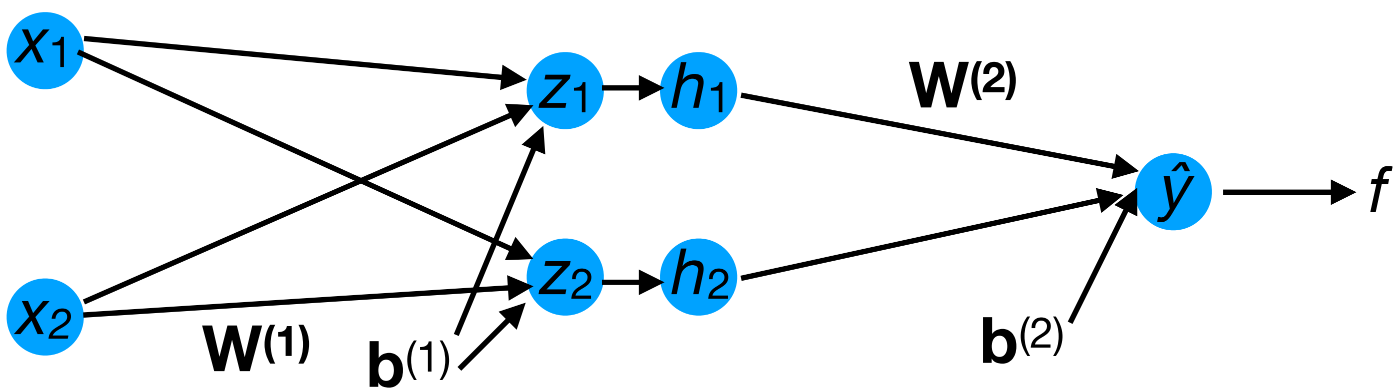


Weight initialization: exercise 1

- Since $\mathbf{W}^{(1)}=\mathbf{b}^{(1)}=0$, then $\mathbf{z}^{(1)}=\mathbf{h}^{(1)}=0$. Hence, $\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} = 0$.
- Since relu'(0) = 0, then $\mathbf{g}=0$. Hence, gradients w.r.t. $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ are 0.
- Only $\mathbf{b}^{(2)}$ can change (so that $\hat{y}$ approaches mean of y).

$$
\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) {\mathbf{h}^{(1)}}^\top \quad 0 \\
\nabla_{\mathbf{b}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) \\
\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}\mathbf{x}^\top \quad 0 \\
\nabla_{\mathbf{b}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g} \quad 0
\end{aligned}
$$

$$
\mathbf{g}^\top = \left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)} \right) \odot \mathrm{relu}'({\mathbf{z}^{(1)}}^\top) \quad 0
$$

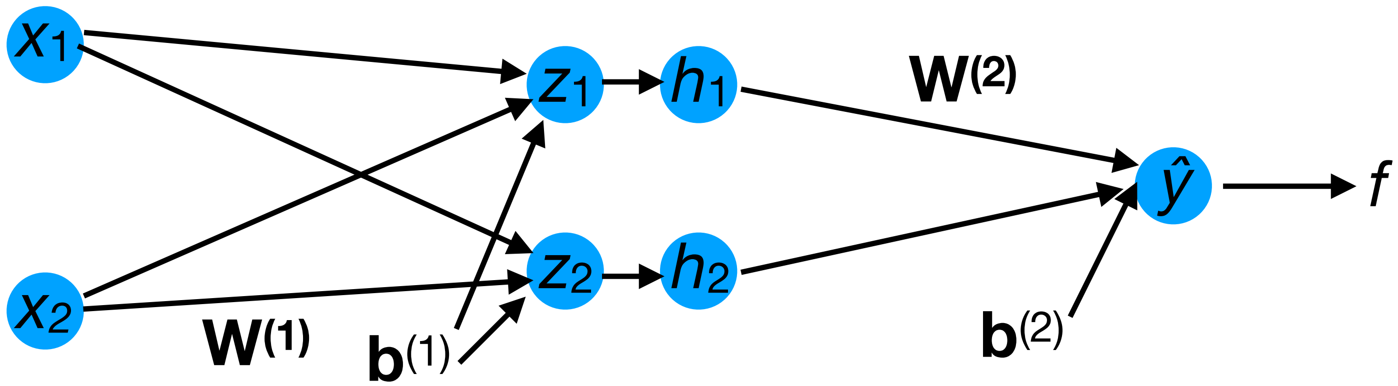


Weight initialization: exercise 2

- Suppose we initialize $\mathbf{W}^{(2)}=\mathbf{b}^{(2)}=0$, but $\mathbf{W}^{(1)}$, $\mathbf{b}^{(1)}$ are non-zero.
- What will happen during SGD?

$$
\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y})\mathbf{h}^{(1)\top} \\
\nabla_{\mathbf{b}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) \\
\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}\mathbf{x}^\top \\
\nabla_{\mathbf{b}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}
\end{aligned}
$$

$$
\mathbf{g}^\top = \left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)}\right) \odot \mathrm{relu}'(\mathbf{z}^{(1)\top})
$$

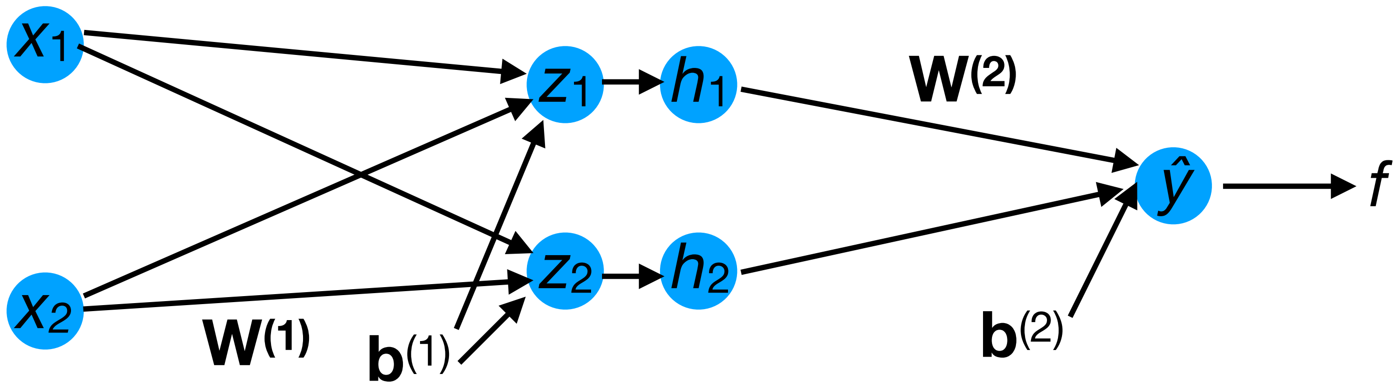


Weight initialization: exercise 2

- Since $\mathbf{W}^{(2)}=0$, then $\mathbf{g}=0$. Hence, $\nabla_{\mathbf{W}^{(1)}} f_{\text{CE}}, \nabla_{\mathbf{b}^{(1)}} f_{\text{CE}} = 0$.

$$
\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\text{CE}} &= (\hat{\mathbf{y}} - \mathbf{y})\mathbf{h}^{(1)^\top} \\
\nabla_{\mathbf{b}^{(2)}} f_{\text{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) \\
\nabla_{\mathbf{W}^{(1)}} f_{\text{CE}} &= \mathbf{g}\mathbf{x}^\top \\
\nabla_{\mathbf{b}^{(1)}} f_{\text{CE}} &= \mathbf{g}
\end{aligned}
$$

$$\mathbf{g}^\top = \left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)} \right) \odot \text{relu}'(\mathbf{z}^{(1)^\top})$$

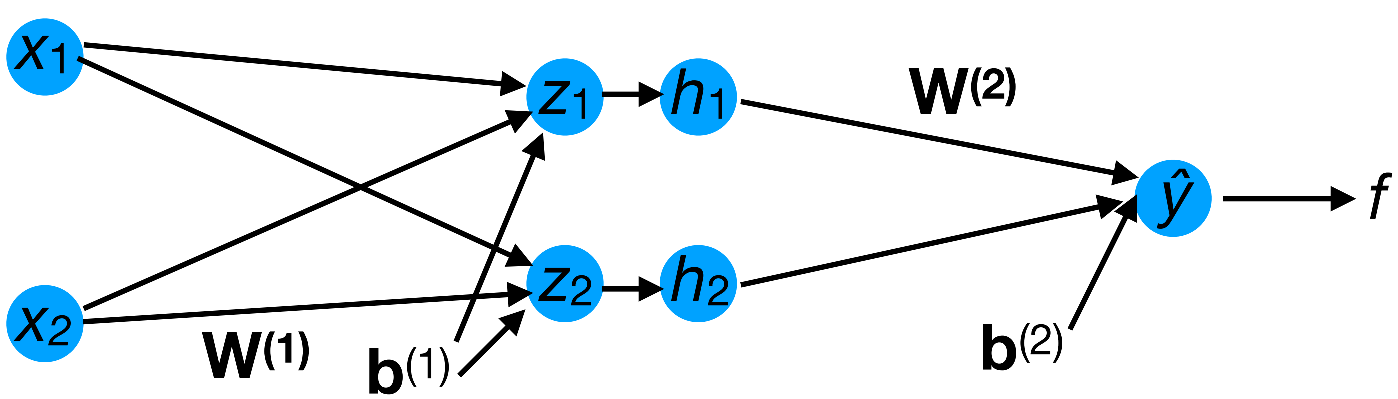


Weight initialization: exercise 2

- Since $\mathbf{W}^{(2)}$=0, then $\mathbf{g}$=0. Hence, $\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}}, \nabla_{\mathbf{b}^{(1)}} f_{\mathrm{CE}}$ = 0.
- However, $\mathbf{h}$ is non-zero. Hence, $\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}}$ is nonzero => $\mathbf{W}^{(2)}$ will change.

$$
\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}){\mathbf{h}^{(1)}}^\top \\
\nabla_{\mathbf{b}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) \\
\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}\mathbf{x}^\top \\
\nabla_{\mathbf{b}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}
\end{aligned}
$$

$$
\mathbf{g}^\top = \left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)}\right) \odot \mathrm{relu}'({\mathbf{z}^{(1)}}^\top)
$$

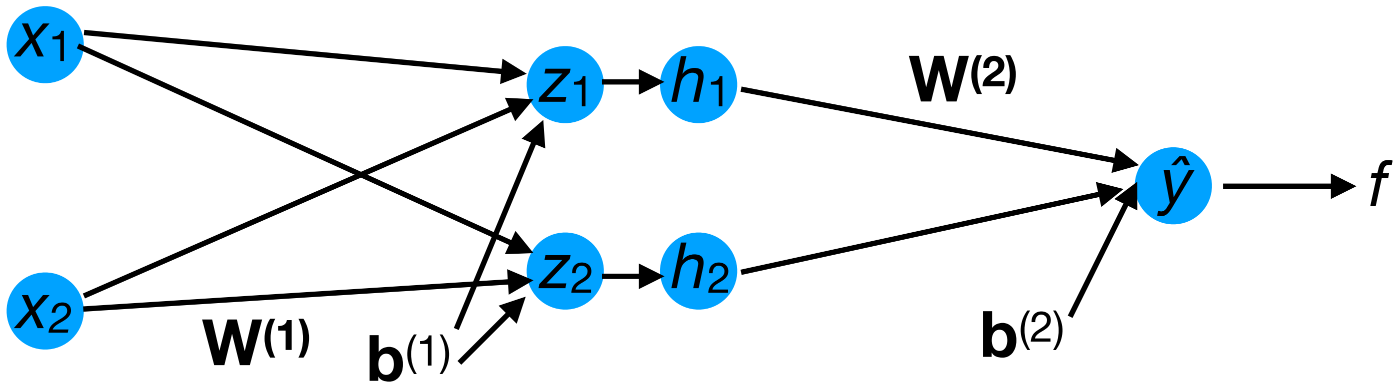


Weight initialization: exercise 2

- Since $\mathbf{W}^{(2)}$=0, then $\mathbf{g}$=0. Hence, $\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}}, \nabla_{\mathbf{b}^{(1)}} f_{\mathrm{CE}}$ = 0.
- However, $\mathbf{h}$ is non-zero. Hence, $\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}}$ is nonzero => $\mathbf{W}^{(2)}$ will change.
- During the *next* gradient update, $\mathbf{g}$ is non-zero => $\mathbf{W}^{(1)}$, $\mathbf{b}^{(1)}$ will change.
- In summary: this initialization does not severely inhibit the network's performance (though initializing $\mathbf{W}^{(2)}$, $\mathbf{b}^{(2)}$ to 0 is still not recommended).

$$\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}){\mathbf{h}^{(1)}}^\top \\
\nabla_{\mathbf{b}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) \\
\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}\mathbf{x}^\top \\
\nabla_{\mathbf{b}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}
\end{aligned}$$

$$\mathbf{g}^\top = \left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)} \right) \odot \mathrm{relu}'({\mathbf{z}^{(1)}}^\top)$$



Weight initialization methods

- There are various different methods of initializing the weights of a neural network.
- One common approach:
  - For weight matrix $\mathbf{W}^{(j)}$, sample each component from a 0-mean probability distribution with standard deviation of $1/\sqrt{\text{cols}(\mathbf{W}^{(j)})}$.
    - Within certain NNs, this can help to ensure that the gradients are usually non-zero.

L_1, L_2 Regularization

Regularization

- **Regularization** is any means to help a machine learning model to generalize better to data not used for training.

- Regularization is particularly important with neural networks since they are so powerful and therefore prone to overfitting.

L_2 regularization in NNs

- To prevent the weight matrices from growing too big, we can apply an L_2 regularization term to each matrix by augmenting the cross-entropy loss:

$$f_{\mathrm{CE}}(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{10}\mathbf{y}_k^{(i)}\log\hat{\mathbf{y}}_k^{(i)} + \frac{1}{2}\|\mathbf{W}^{(1)}\|_{\mathrm{Fr}}^2 + \frac{1}{2}\|\mathbf{W}^{(2)}\|_{\mathrm{Fr}}^2$$

- Here, $|\mathbf{W}|_{\mathrm{Fr}}{}^2$ means the squared **Frobenius norm** of **W**.
  - It's just the sum of squares of all the elements of **W**.

L_2 regularization in NNs

- This results in a modified gradient for each weight matrix:

$$
\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}){\mathbf{h}^{(1)}}^\top + \mathbf{W}^{(2)} \\
\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}\mathbf{x}^\top + \mathbf{W}^{(1)}
\end{aligned}
$$

L_1 regularization in NNs

- Alternatively, we can use L_1 regularization, which encourages entries of the weight matrix to be exactly 0:

$$f_{\mathrm{CE}}(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{10}\mathbf{y}_k^{(i)} \log \hat{\mathbf{y}}_k^{(i)} + |\mathbf{W}^{(1)}| + |\mathbf{W}^{(2)}|$$

- Here, $|\mathbf{W}|$ means the sum of the absolute values of each element of $\mathbf{W}$.

L_1 regularization in NNs

- This results in the following gradient terms:

$$
\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) {\mathbf{h}^{(1)}}^\top + \mathrm{sign}\left(\mathbf{W}^{(2)}\right) \\
\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}\mathbf{x}^\top + \mathrm{sign}\left(\mathbf{W}^{(1)}\right)
\end{aligned}
$$

$L_1 + L_2$ regularization

- We can also combine both kinds of regularization with different strengths for each weight matrix:

$$
\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\mathrm{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) {\mathbf{h}^{(1)}}^\top + \alpha^{(2)} \mathbf{W}^{(2)} + \beta^{(2)} \mathrm{sign}\left(\mathbf{W}^{(2)}\right) \\
\nabla_{\mathbf{W}^{(1)}} f_{\mathrm{CE}} &= \mathbf{g}\mathbf{x}^\top + \alpha^{(1)} \mathbf{W}^{(1)} + \beta^{(1)} \mathrm{sign}\left(\mathbf{W}^{(1)}\right)
\end{aligned}
$$

- This results in several hyperparameters, all of which should be optimized on a separate validation set (*not* the test set).

Dropout

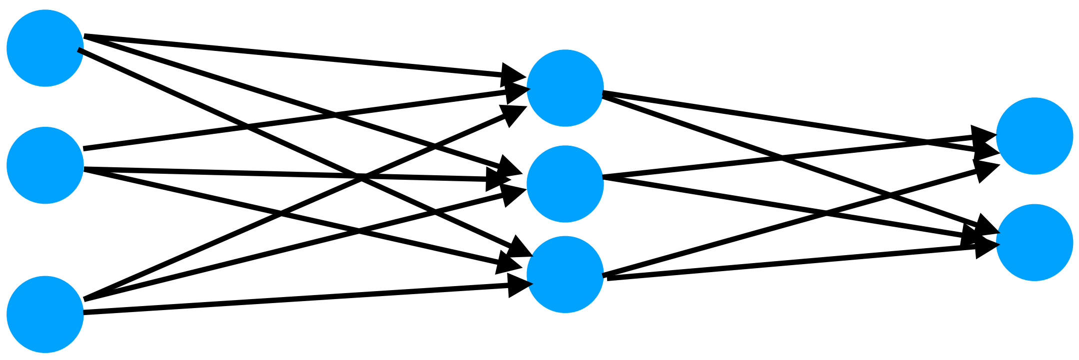
Dropout

- One of the more recently discovered regularization methods is **dropout**, whereby a random set of neurons is removed (and later replaced) from the network for each gradient update.

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

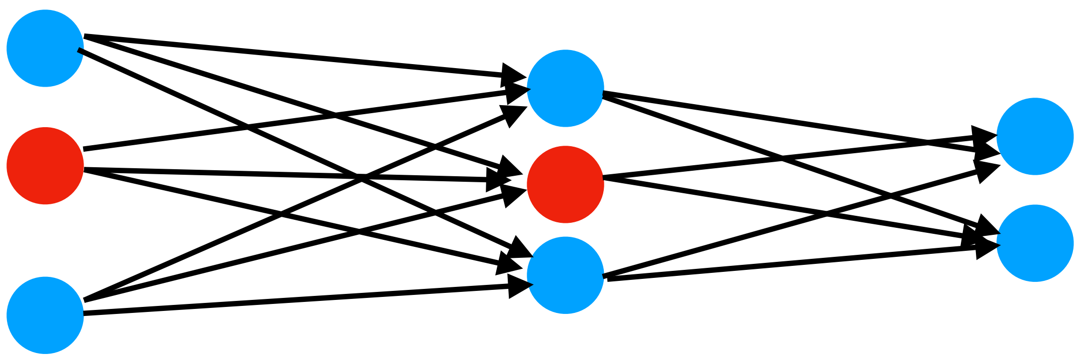
Dropout

- Suppose we are training the NN shown below:



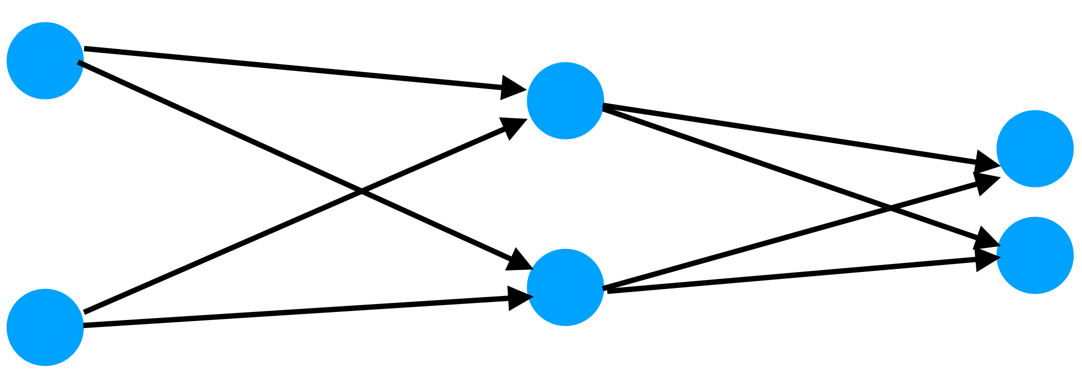
Dropout

- 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

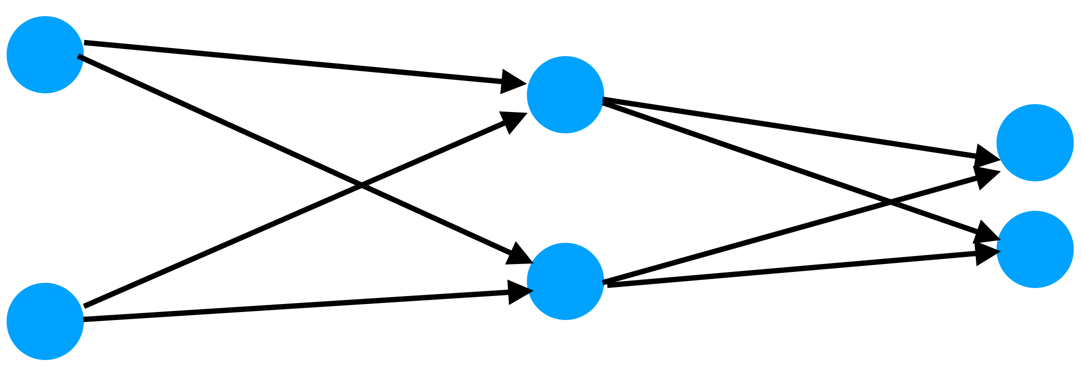
- Suppose we are training the NN shown below:
- We then remove these neurons and perform forward-propagation on the reduced network.



Dropout

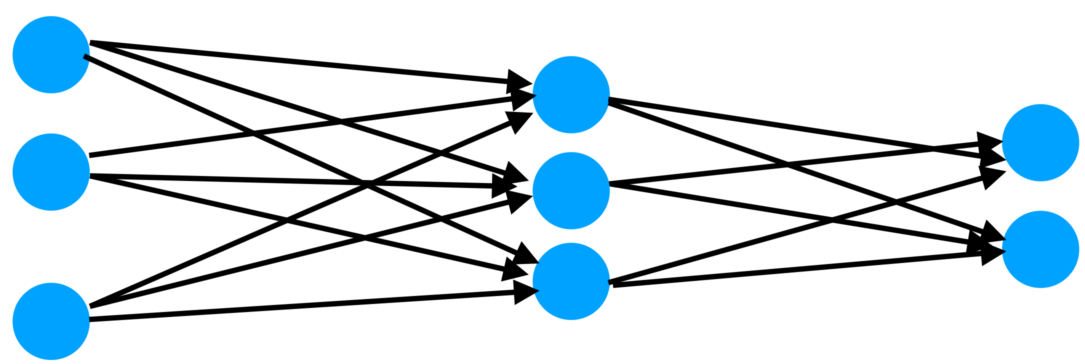
- Suppose we are training the NN shown below:

- 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.)

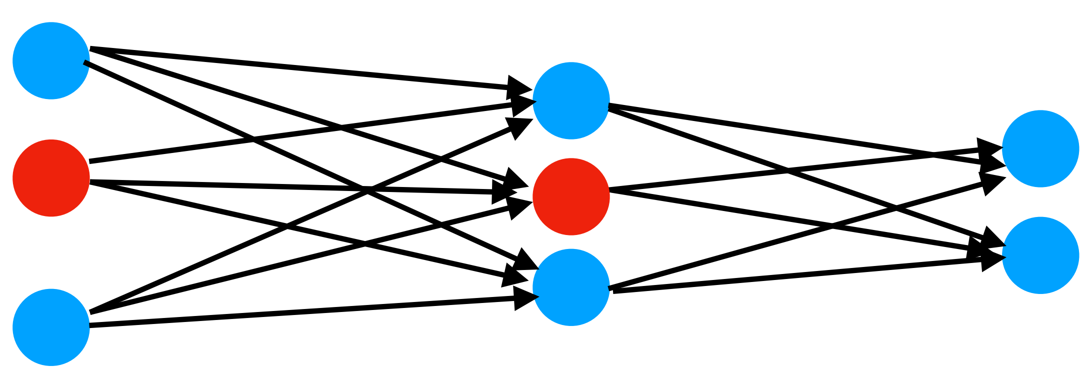


Dropout: example

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

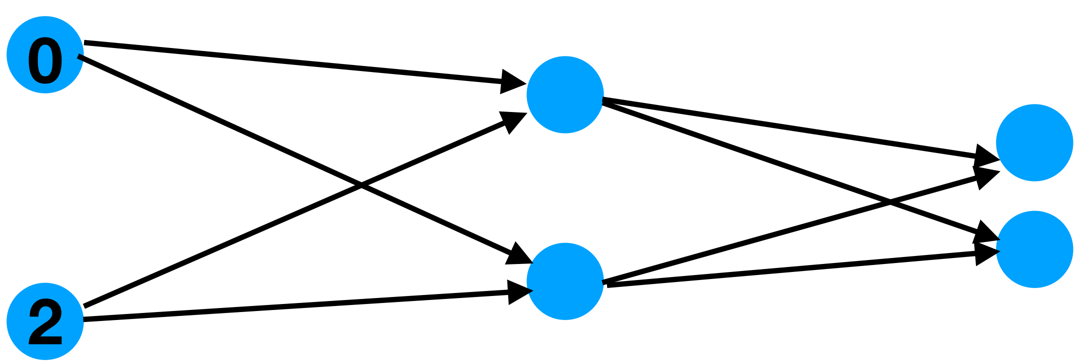


Dropout: example

- 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.

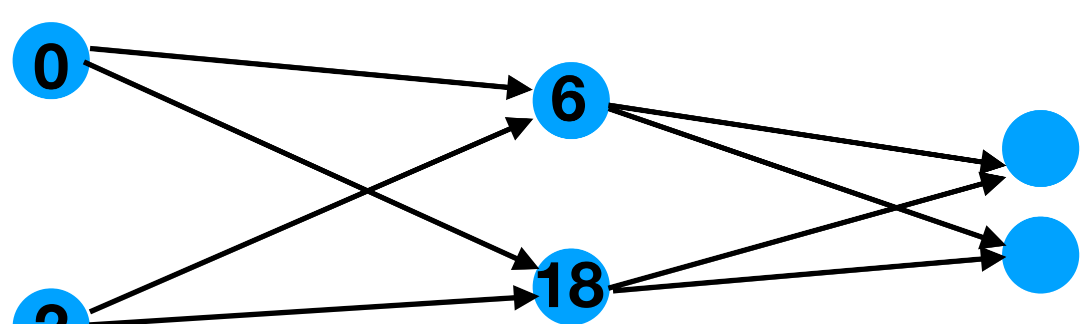


Dropout: example

- 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.



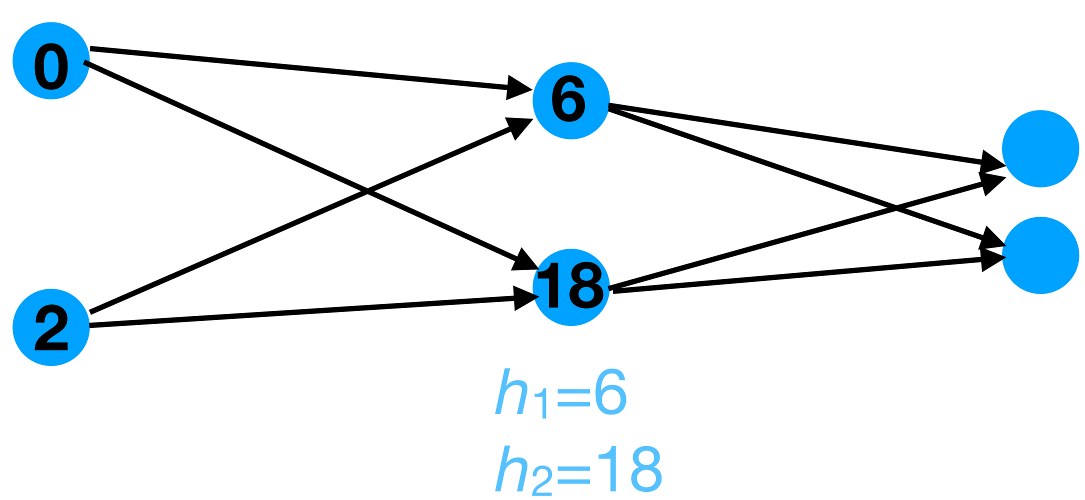
$z_1=1*0 +3*2=6$
$z_2=7*0 +9*2=18$

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.

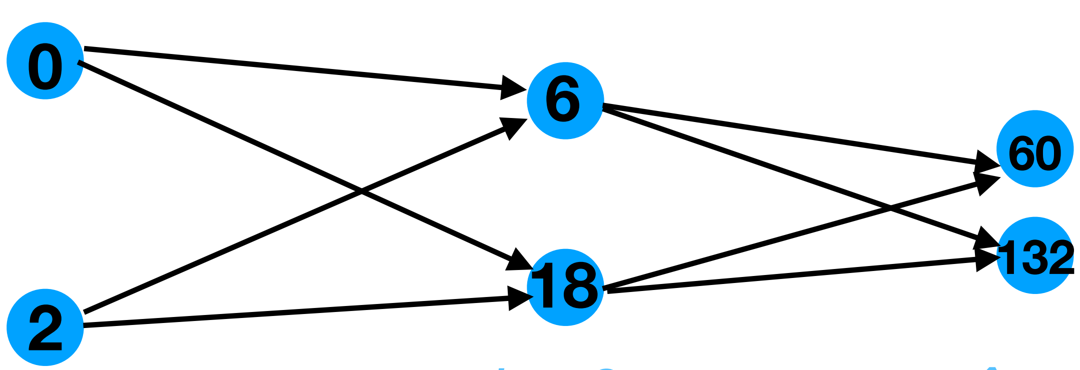


Dropout: example

- 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.



$h_1=6$
$h_2=18$

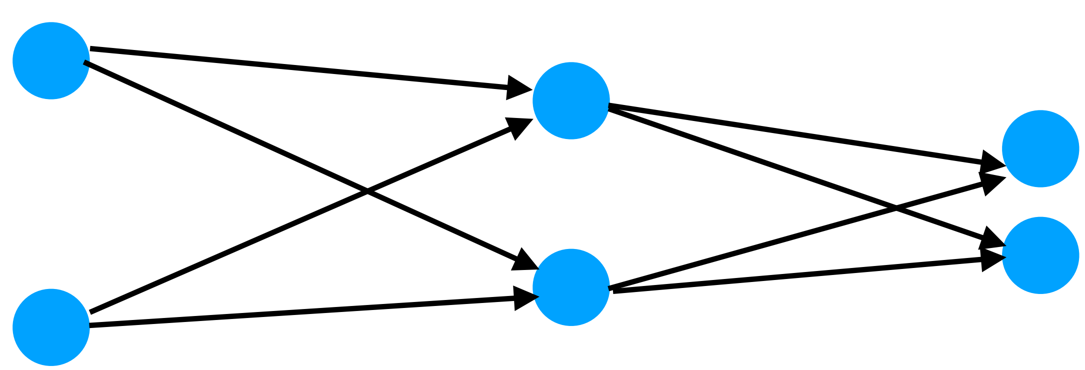
$\hat{y}_1=1*6+3*18=60$
$\hat{y}_2=4*6+6*18=132$

Dropout: example

- Suppose the weights are:

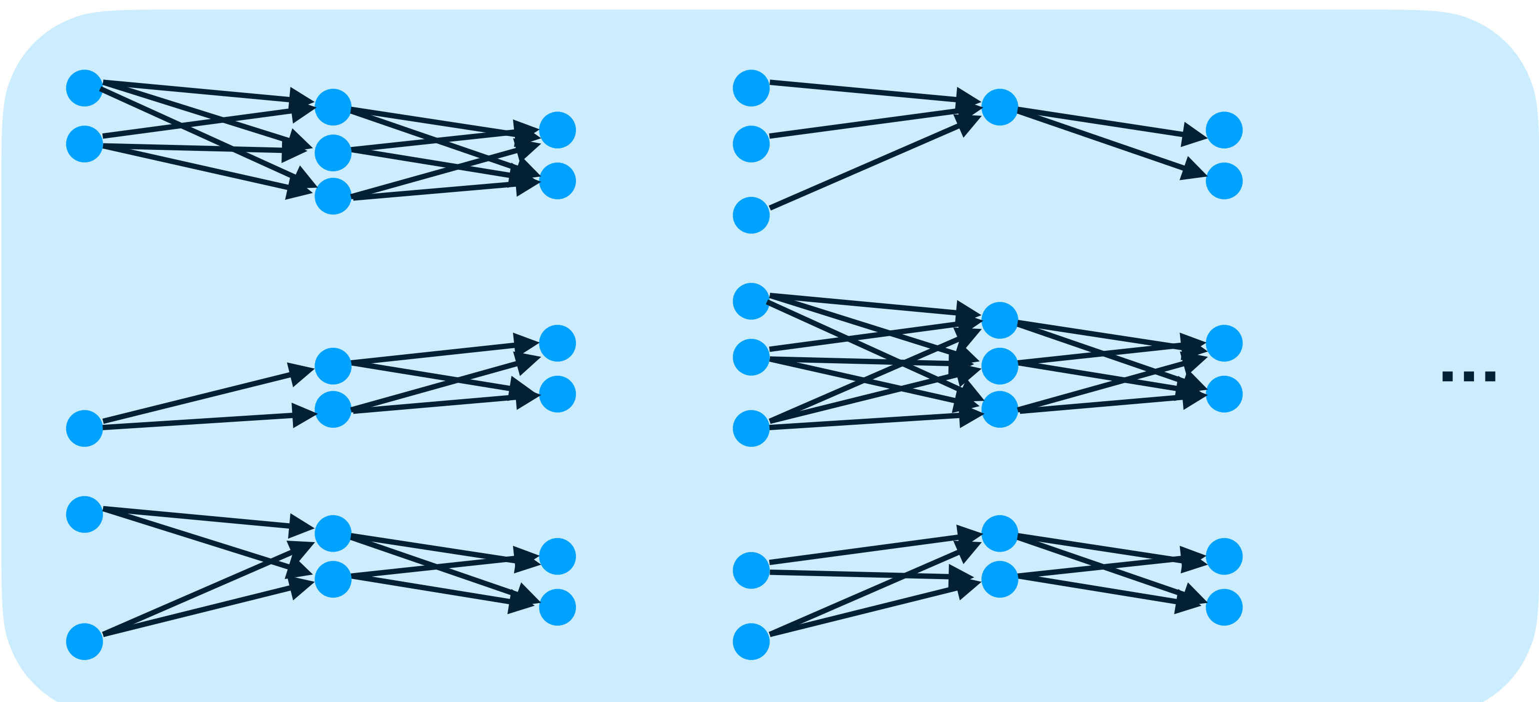
$$\mathbf{W}^{(1)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad \mathbf{W}^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

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



Ensemble of many smaller networks

- Dropout-based NN training can 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