CS/DS 541: Class 7

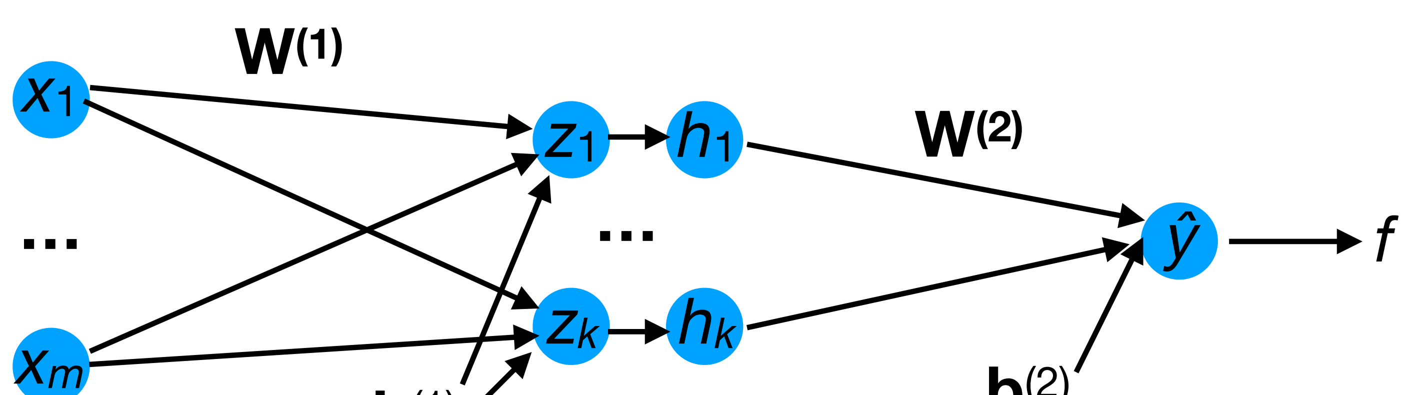
Jacob Whitehill

Training neural networks

Training neural networks

- To train arbitrarily deep NNs, we use the same strategy as we did for linear regression and softmax regression:
  - For each parameter *p*, estimate how the loss function changes as *p* changes, i.e., compute:

$$\nabla_p f(\mathbf{X}, \mathbf{y}; \mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots)$$



Training neural networks

- To train arbitrarily deep NNs, we use the same strategy as we did for linear regression and softmax regression:
  - For each parameter *p*, estimate how the loss function changes as *p* changes, i.e., compute:

$$\nabla_p f(\mathbf{X}, \mathbf{y}; \mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots)$$

  - Update the parameter by moving it slightly opposite the gradient:

$$p^{\text{new}} \leftarrow p^{\text{old}} - \epsilon \nabla_p f(\mathbf{X}, \mathbf{y}; \ldots p^{\text{old}} \ldots)$$

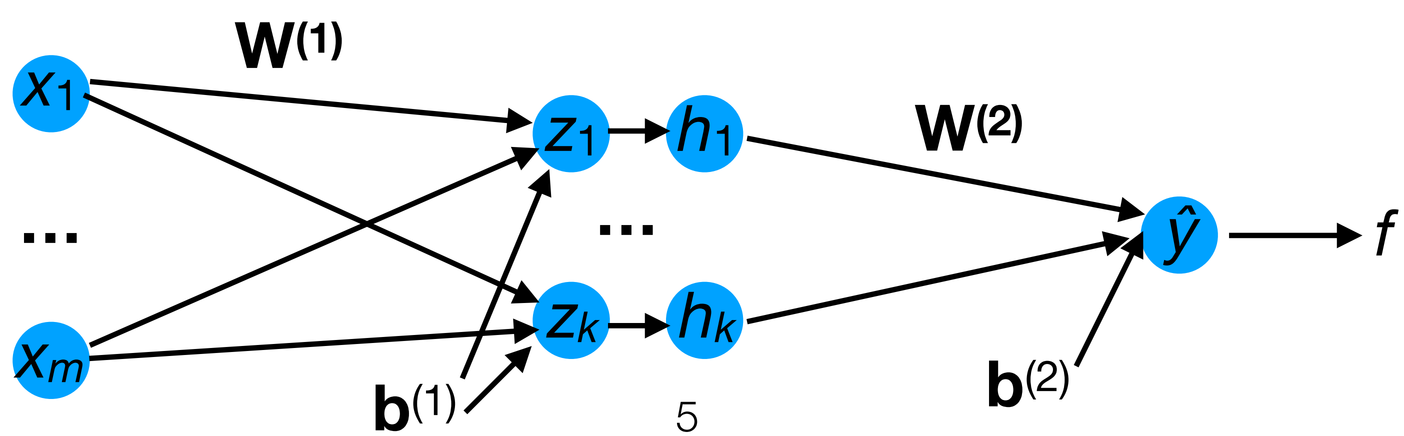
Training neural networks

- Neural networks represent deep compositions of functions.

- Hence, to compute each gradient, we use the chain rule of multivariate calculus, i.e.:

$$\frac{\partial(f \circ g \circ h)}{\partial \mathbf{x}} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial h} \frac{\partial h}{\partial \mathbf{x}}$$

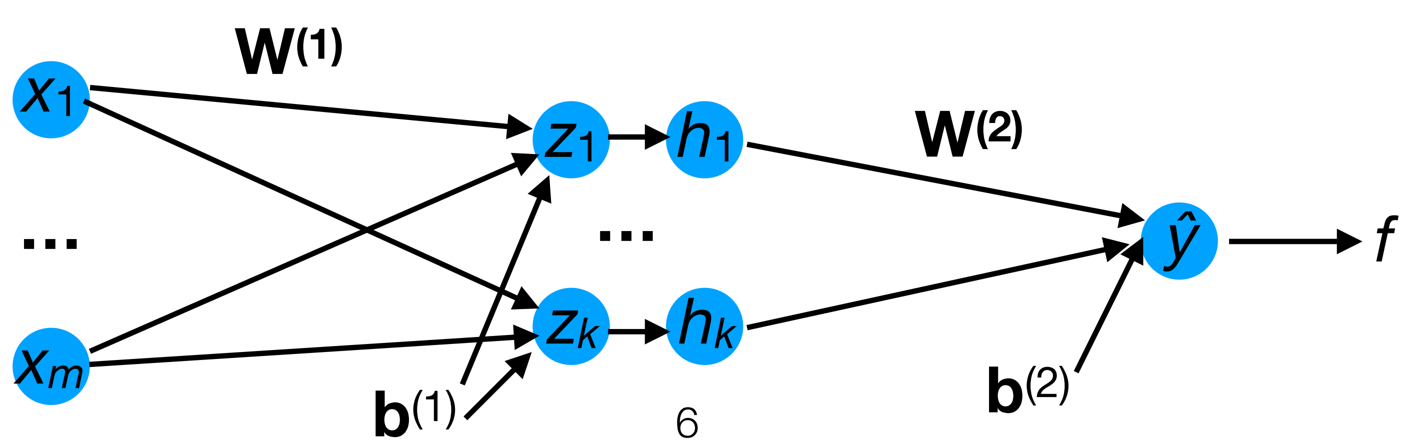
- This means we must compute multiple Jacobian matrices and multiple them together.



Computing the gradients

- To train the NN below, we need to compute the gradients:

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

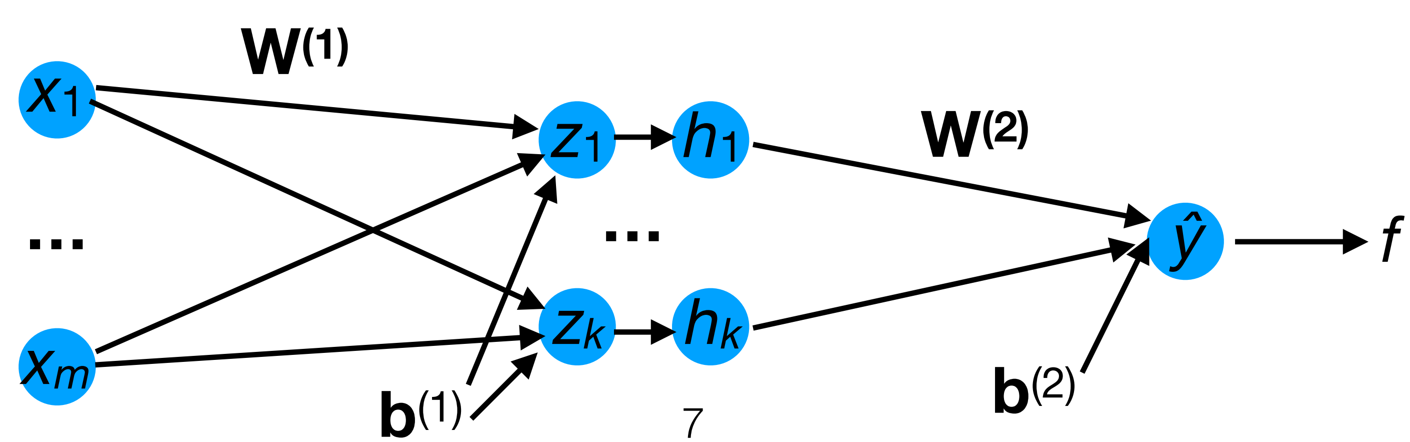


Computing the gradients

- Note that the gradient terms share some computation:

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

Redundant computation

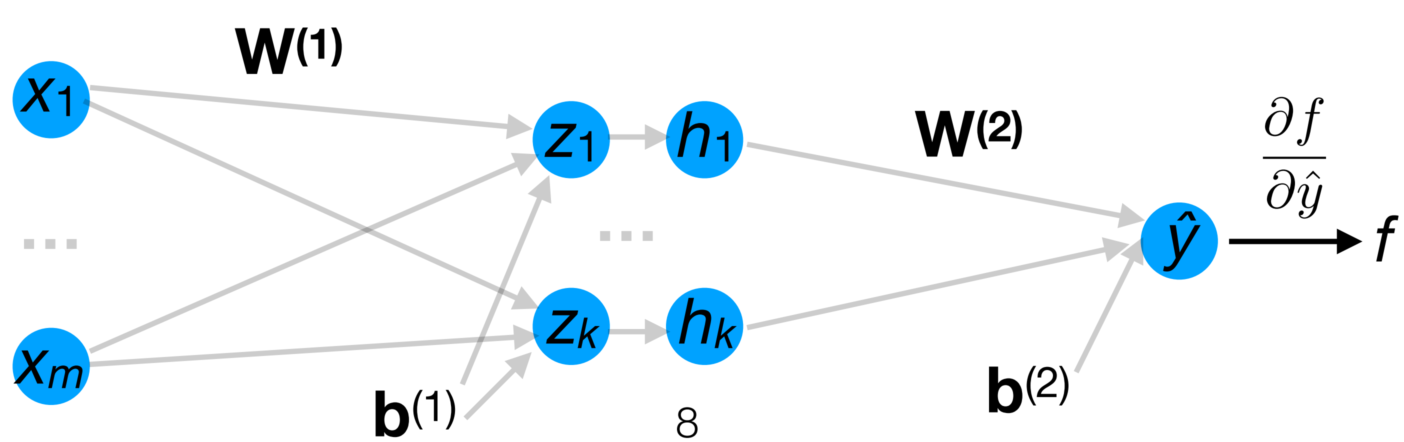


Jacob Whitehill, WPI

Computing the gradients

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

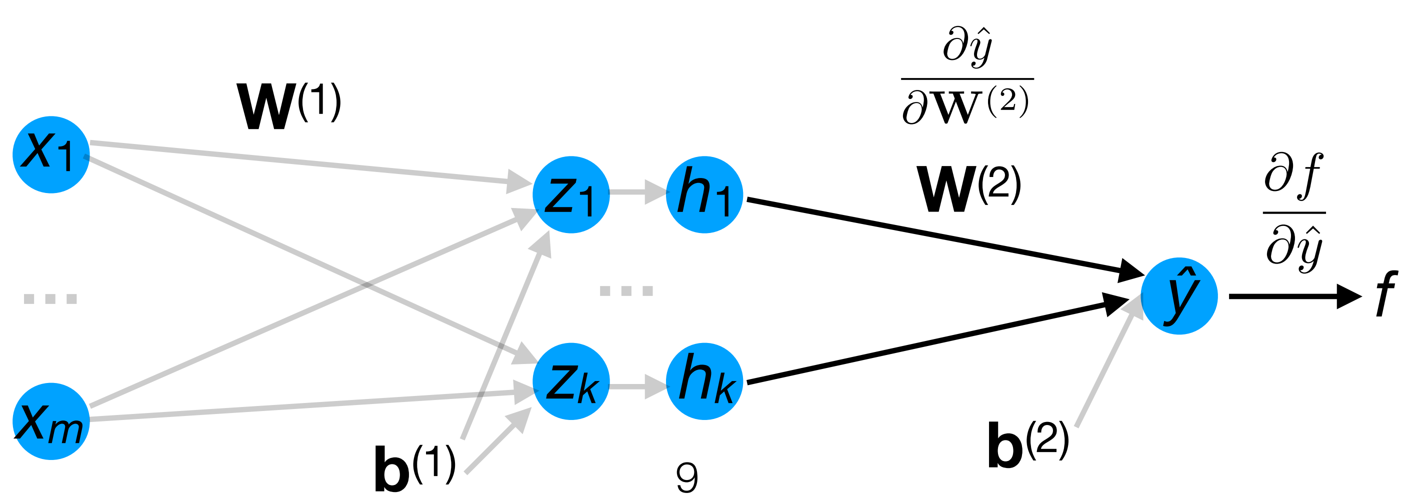
$$\frac{\partial f}{\partial \mathbf{W}^{(2)}} = \frac{\partial f}{\partial \hat{y}}$$



Computing the gradients

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

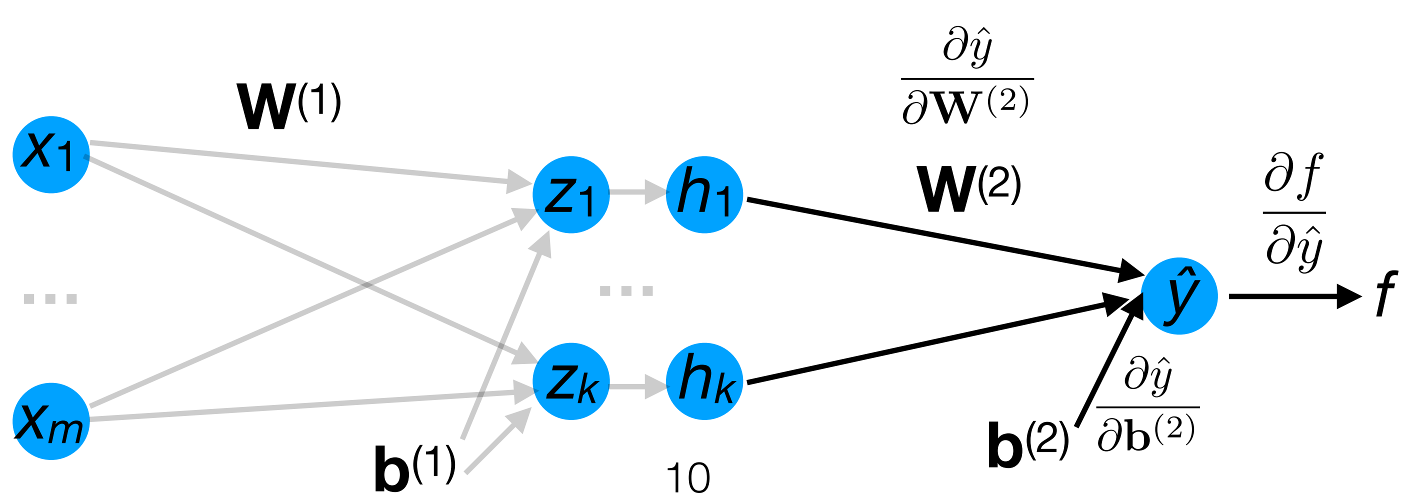
$$\frac{\partial f}{\partial \mathbf{W}^{(2)}} = \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}}$$



Computing the gradients

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

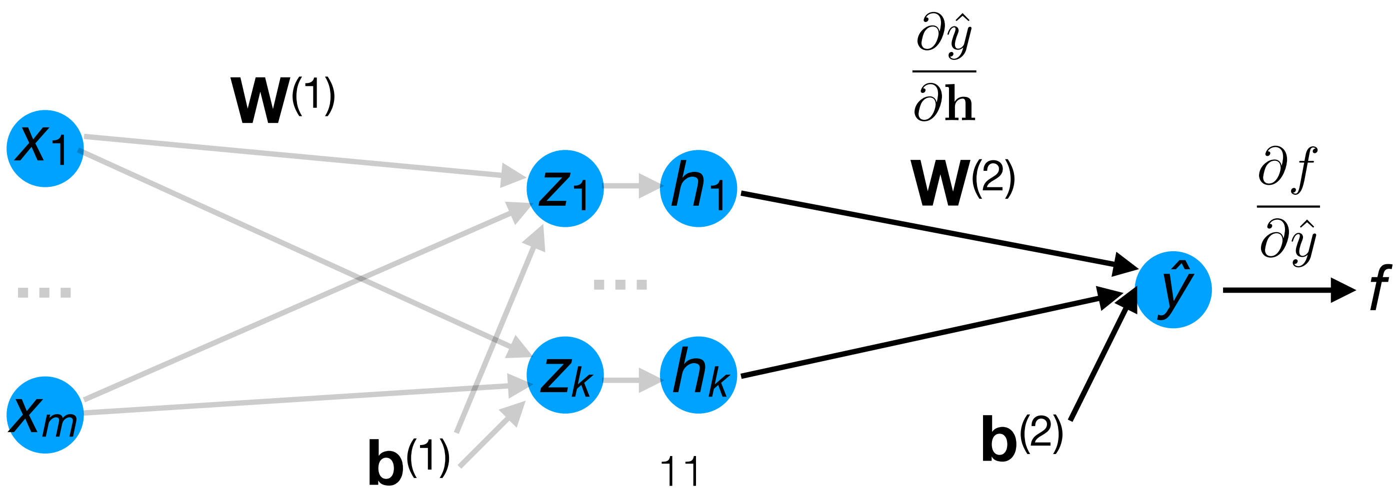
$$\begin{aligned}\frac{\partial f}{\partial \mathbf{W}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}} \\ \frac{\partial f}{\partial \mathbf{b}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}}\end{aligned}$$



Computing the gradients

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

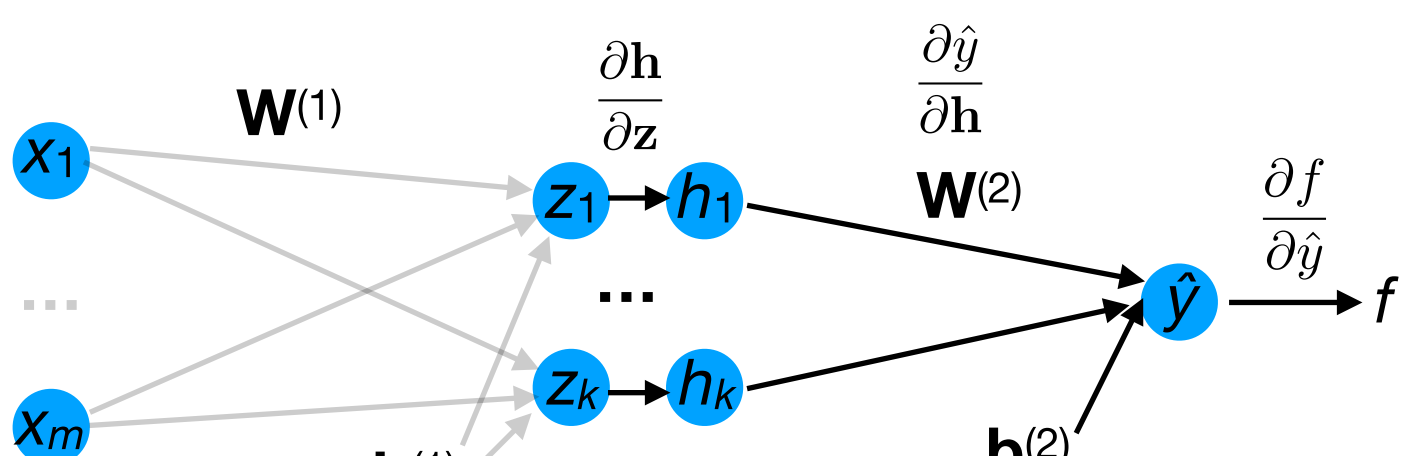
$$\frac{\partial f}{\partial \mathbf{W}^{(2)}} = \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}}$$
$$\frac{\partial f}{\partial \mathbf{b}^{(2)}} = \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}}$$
$$\frac{\partial f}{\partial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



Computing the gradients

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

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

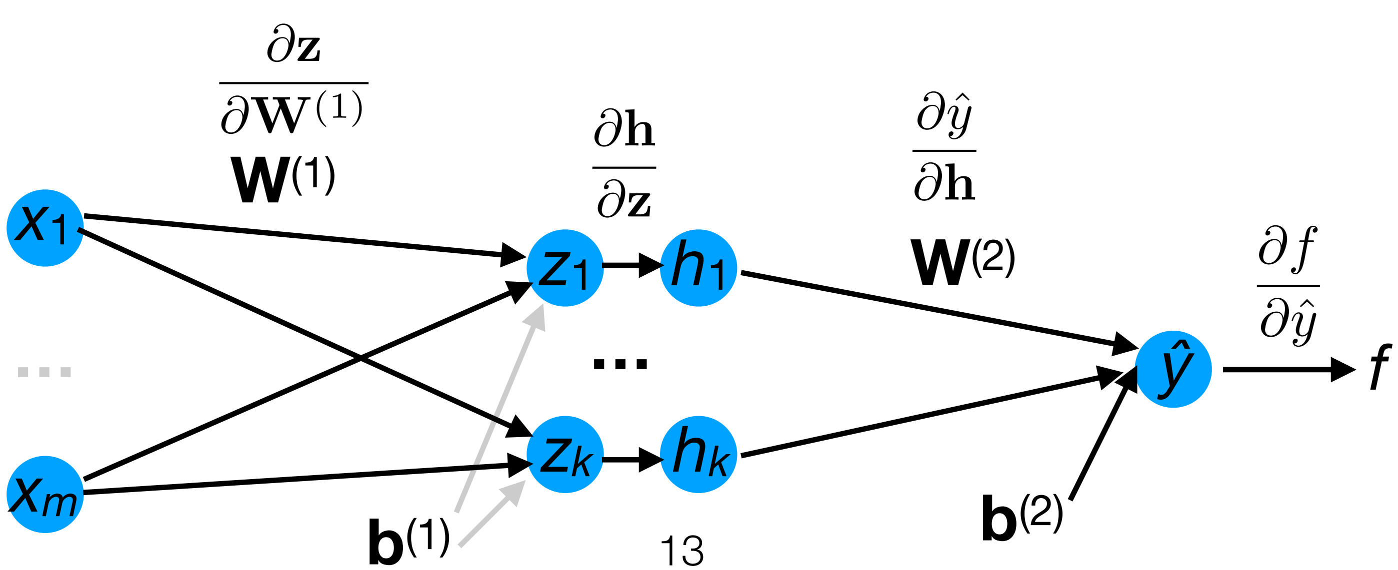


Jacob Whitehill, WPI

Computing the gradients

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

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

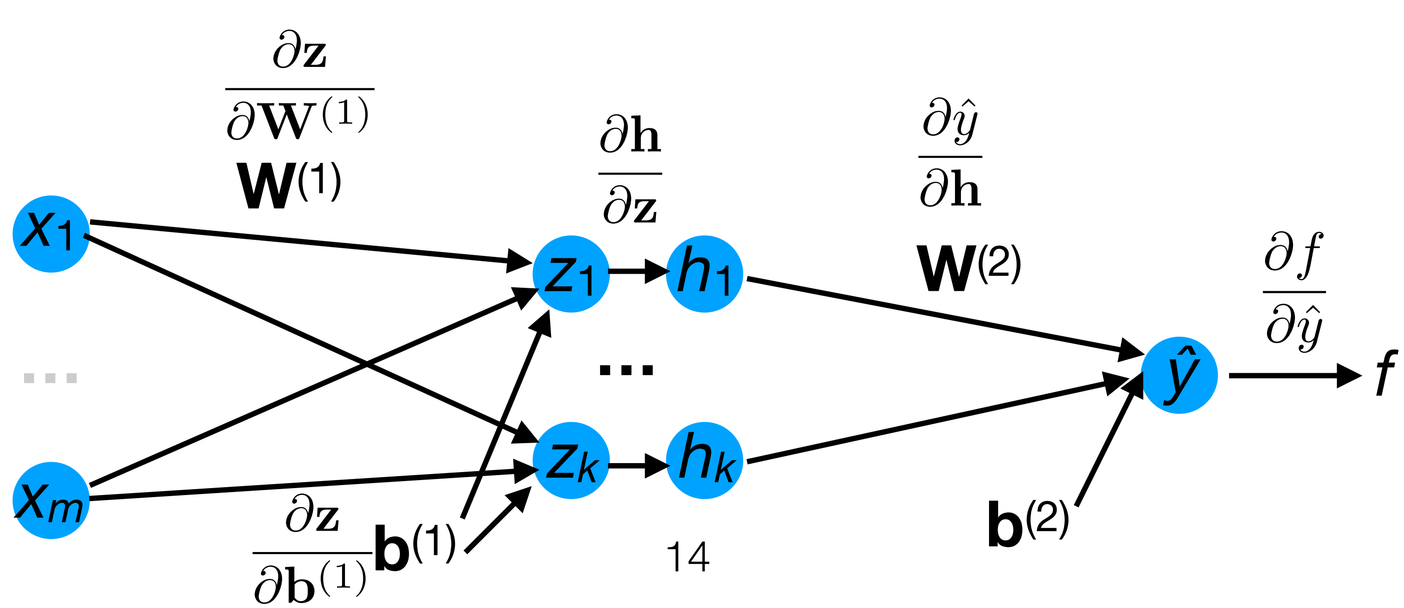


Jacob Whitehill, WPI

Computing the gradients

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

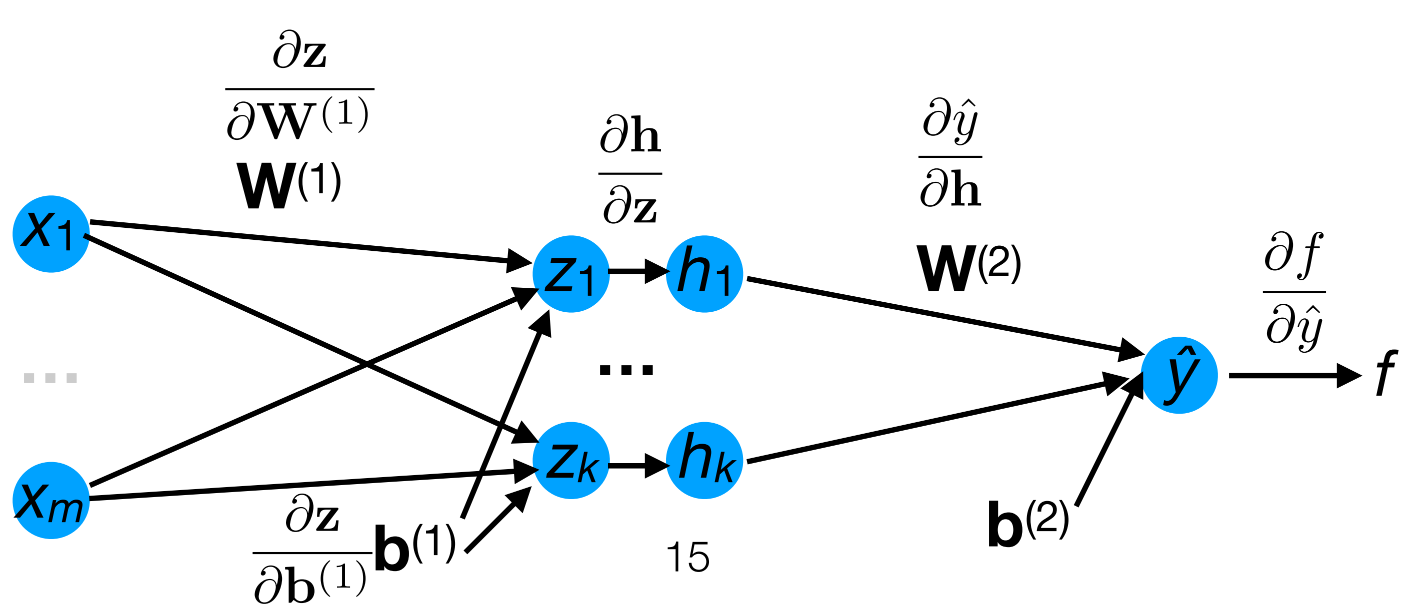
$$
\begin{aligned}
\frac{\partial f}{\partial \mathbf{W}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}} \\
\frac{\partial f}{\partial \mathbf{b}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}} \\
\frac{\partial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



Jacob Whitehill, WPI

Computing the gradients

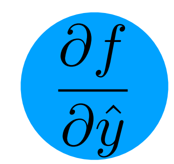
- This process is known as **backwards propagation ("backprop"):**
  - It produces the gradient terms of all the weight matrices and bias vectors.
  - It requires first conducting forward propagation.

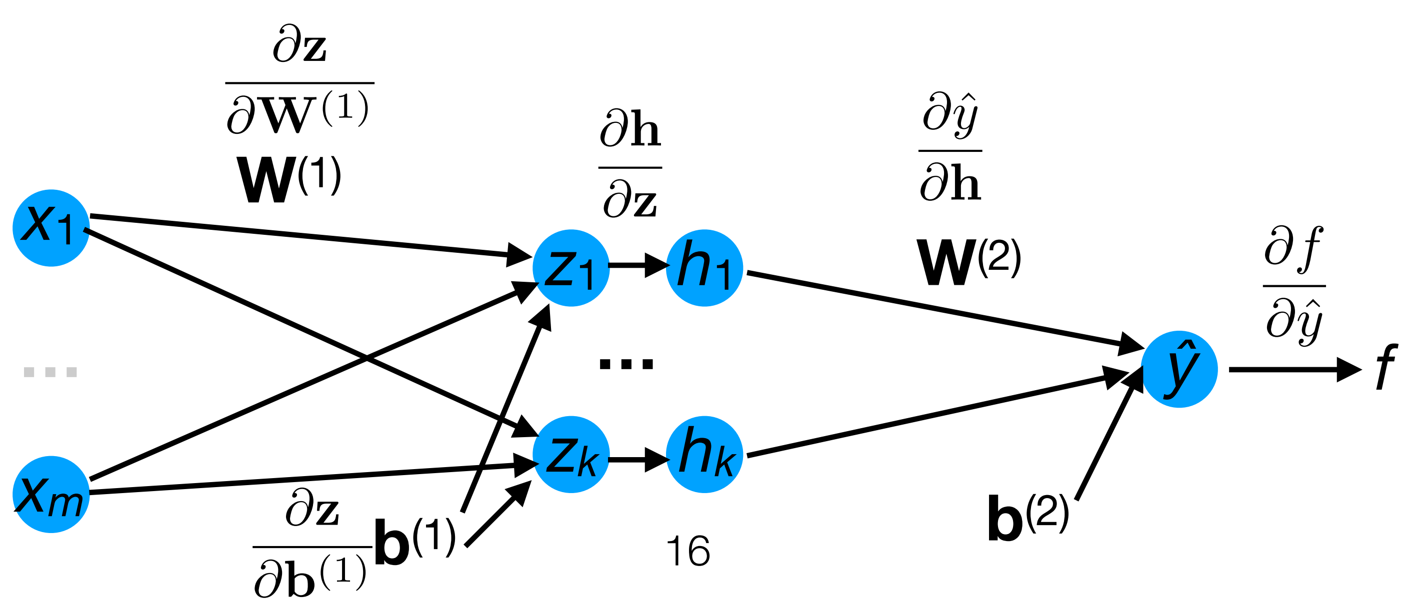


Computing the gradients

Forward propagation

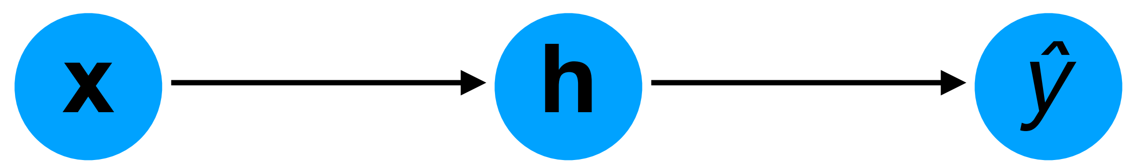
Backward propagation



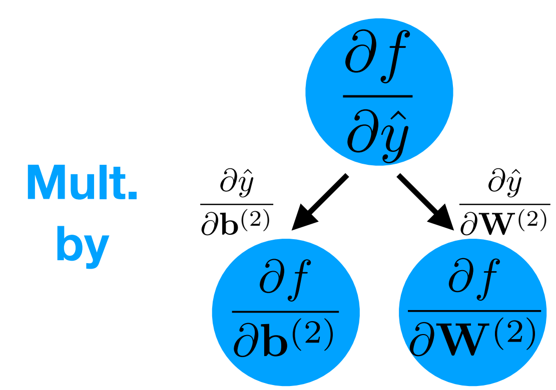


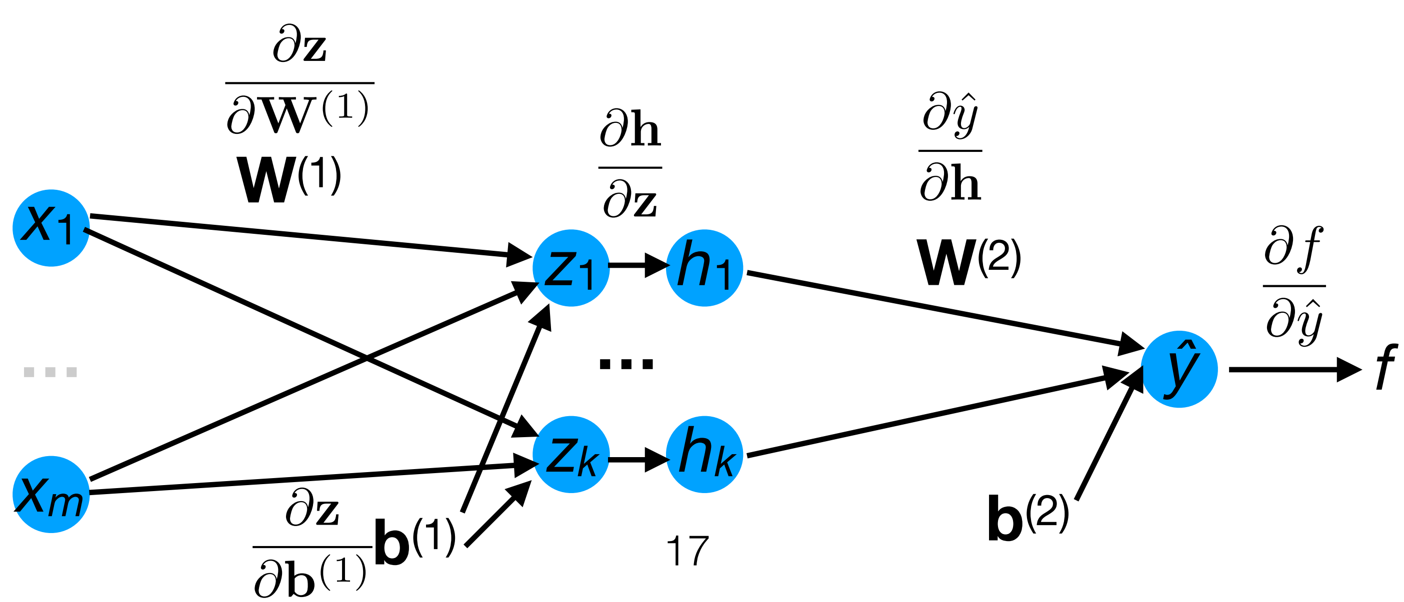
Computing the gradients

Forward propagation

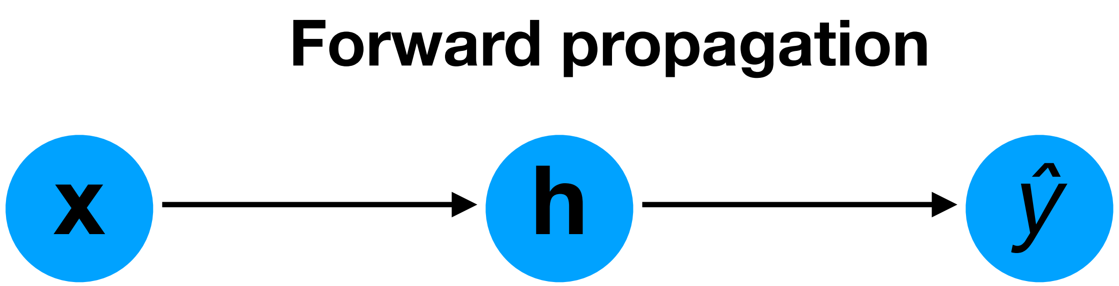


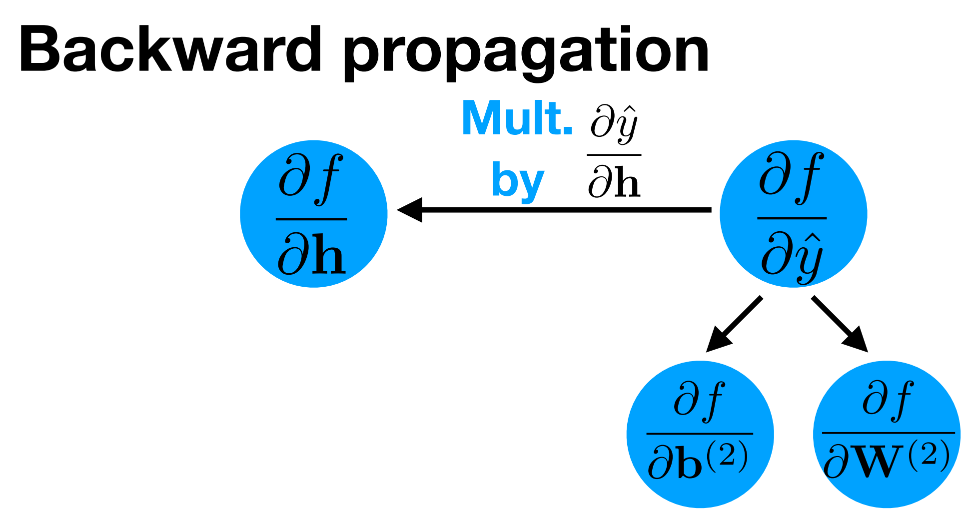
Backward propagation

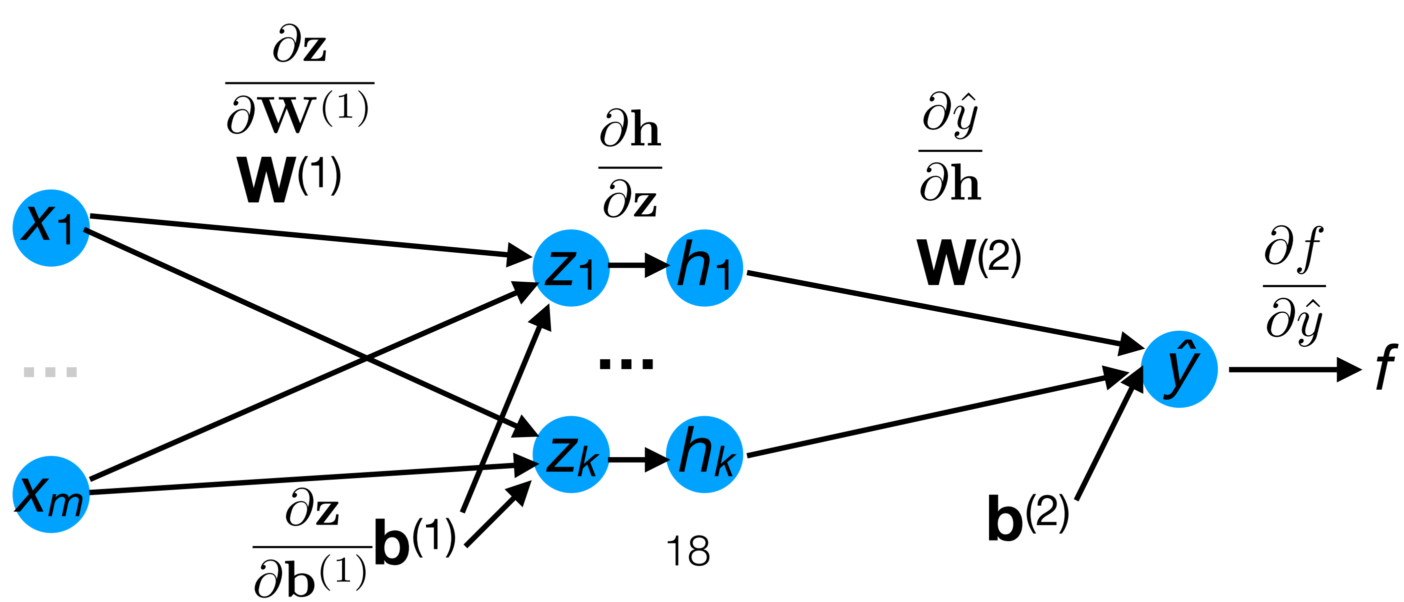




Computing the gradients



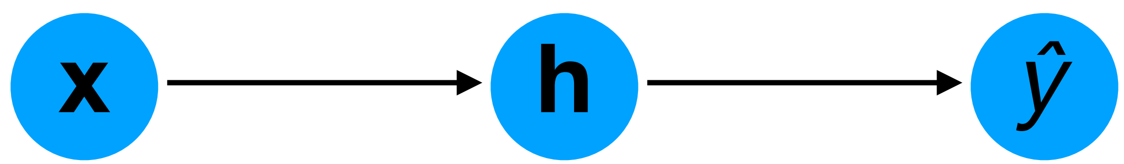




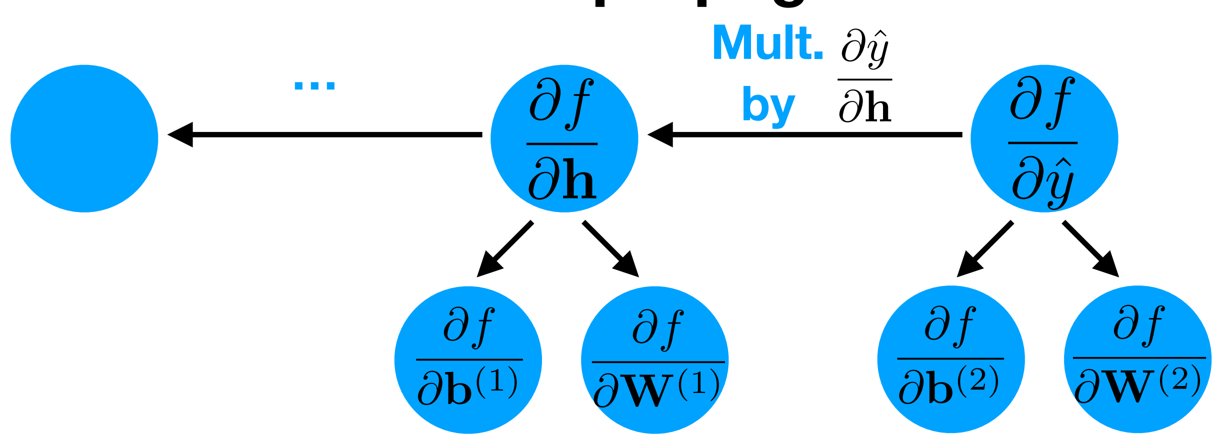
Jacob Whitehill, WPI

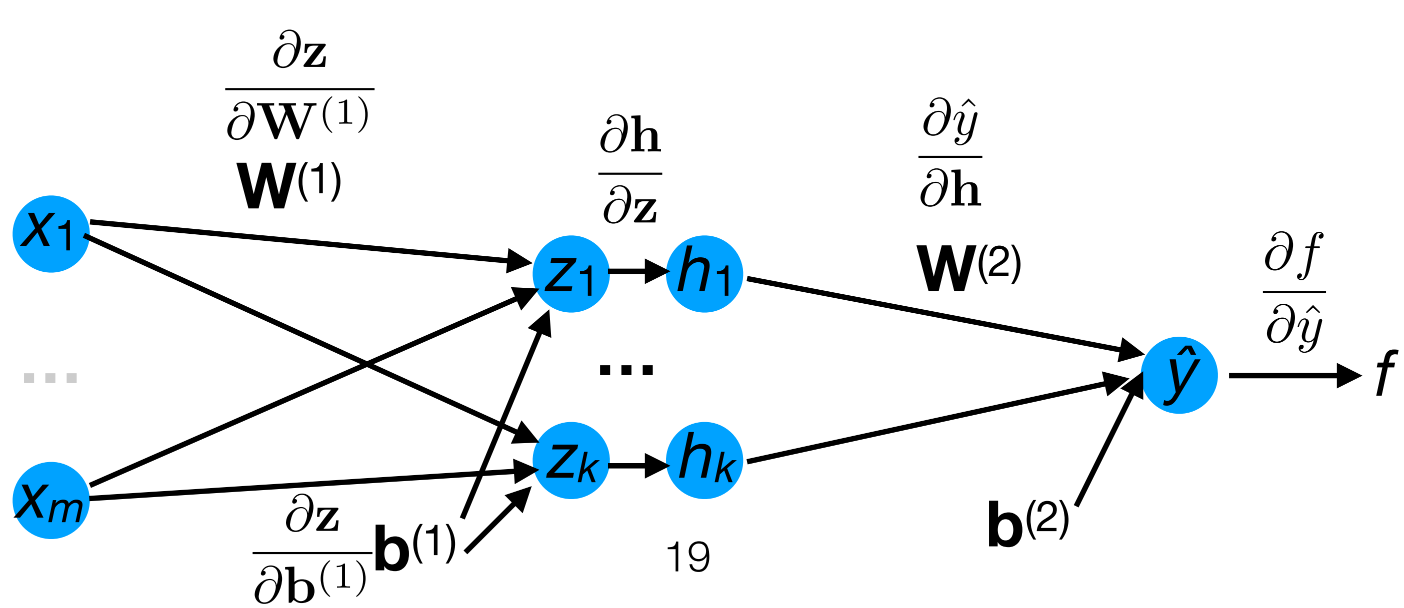
Computing the gradients

Forward propagation



Backward propagation





Computing the gradients

- Applying the chain rule is equivalent to multiplying a sequence of Jacobian matrices.
- However, for the vast majority of NN designs, it turns out that we can simplify this process analytically.
- After simplifying, we obtain the following algorithm for training arbitrarily deep FFNNs…

Computing the gradients

Algorithm 6.4 Backward computation for the deep neural network of algorithm 6.3, which uses, in addition to the input $\boldsymbol{x}$, a target $\boldsymbol{y}$. This computation yields the gradients on the activations $\boldsymbol{a}^{(k)}$ for each layer k, starting from the output layer and going backwards to the first hidden layer. From these gradients, which can be interpreted as an indication of how each layer's output should change to reduce error, one can obtain the gradient on the parameters of each layer. The gradients on weights and biases can be immediately used as part of a stochastic gradient update (performing the update right after the gradients have been computed) or used with other gradient-based optimization methods.

After the forward computation, compute the gradient on the output layer:
$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$
for $k = l, l-1, \dots, 1$ **do**
  Convert the gradient on the layer's output into a gradient on the pre-nonlinearity activation (element-wise multiplication if f is element-wise):

  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$
  Compute gradients on weights and biases (including the regularization term, where needed):
  $\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$
  $\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$
  Propagate the gradients w.r.t. the next lower-level hidden layer's activations:
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top}\, \boldsymbol{g}$
end for

http://www.deeplearningbook.org/contents/mlp.html

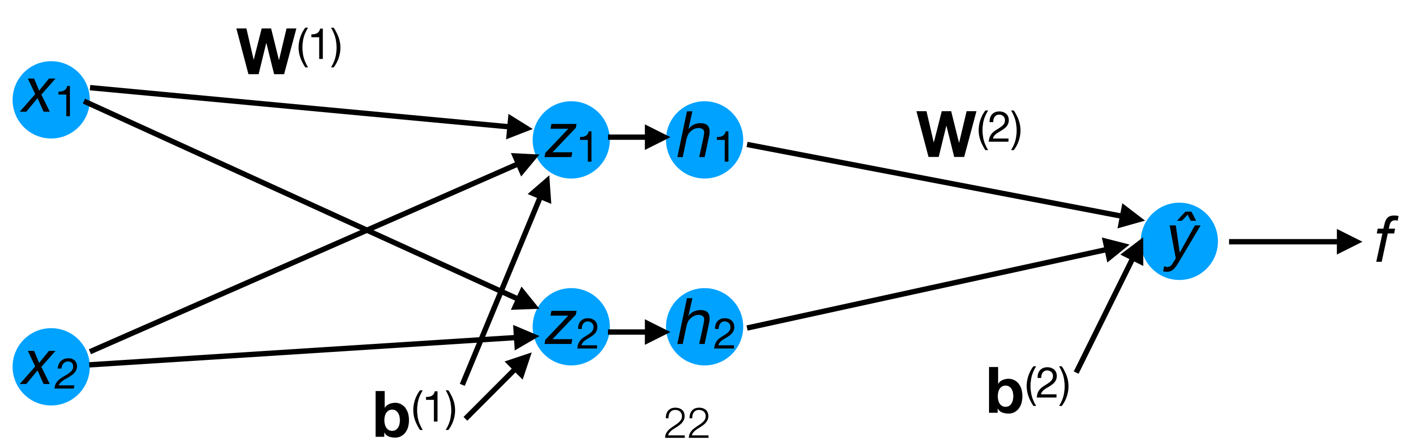
Computing the gradients

- Where do these come from?

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

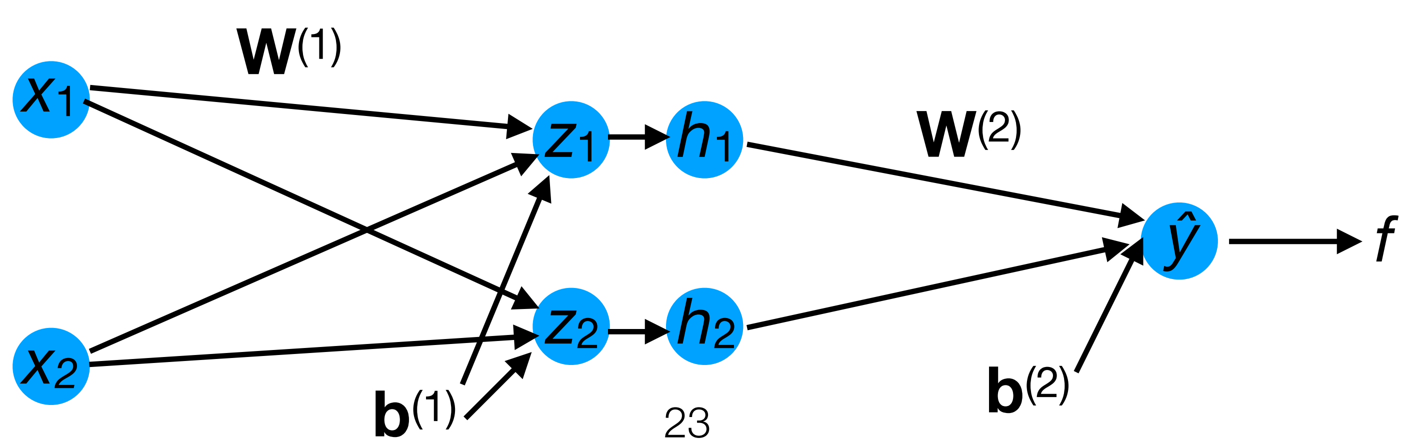
where

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



Exercise

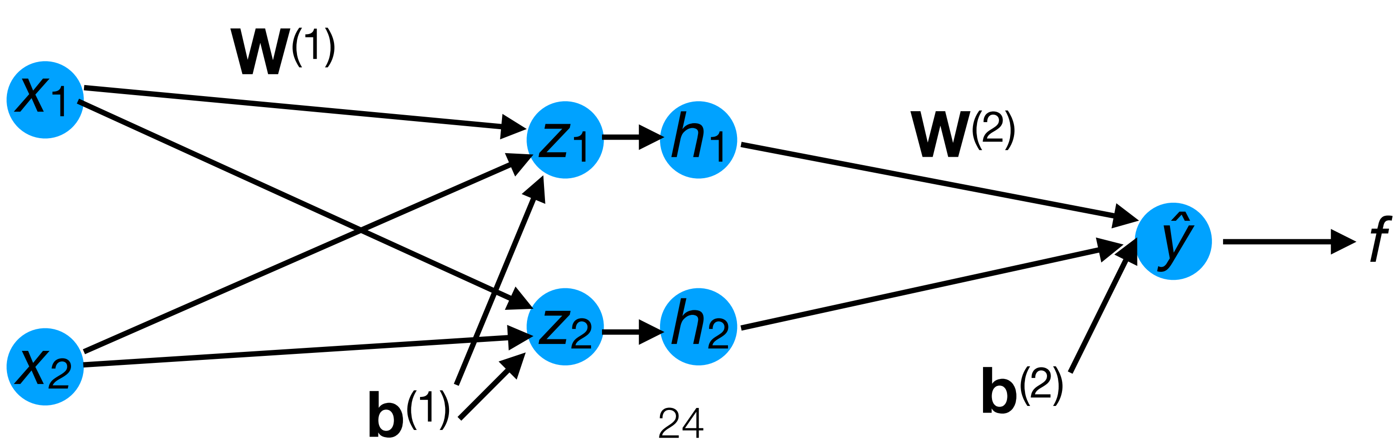
- Let's first focus on the derivative of **h** w.r.t. **z**.
- Let relu' be the gradient of relu w.r.t. its input z (which is 1 if $z > 0$ and 0 if $z <= 0$).
- Derive the Jacobian matrix for d**h**/d**z**:



Exercise

- Let's first focus on the derivative of **h** w.r.t. **z**.
- Let relu' be the gradient of relu w.r.t. its input *z* (which is 1 if *z* > 0 and 0 if *z* <= 0).
- Derive the Jacobian matrix for d**h**/d**z**:

$$\frac{\partial \mathbf{h}}{\partial \mathbf{z}} = \left[\begin{array}{cc} \mathrm{relu}'(z_1) & 0 \\ 0 & \mathrm{relu}'(z_2) \end{array} \right]$$

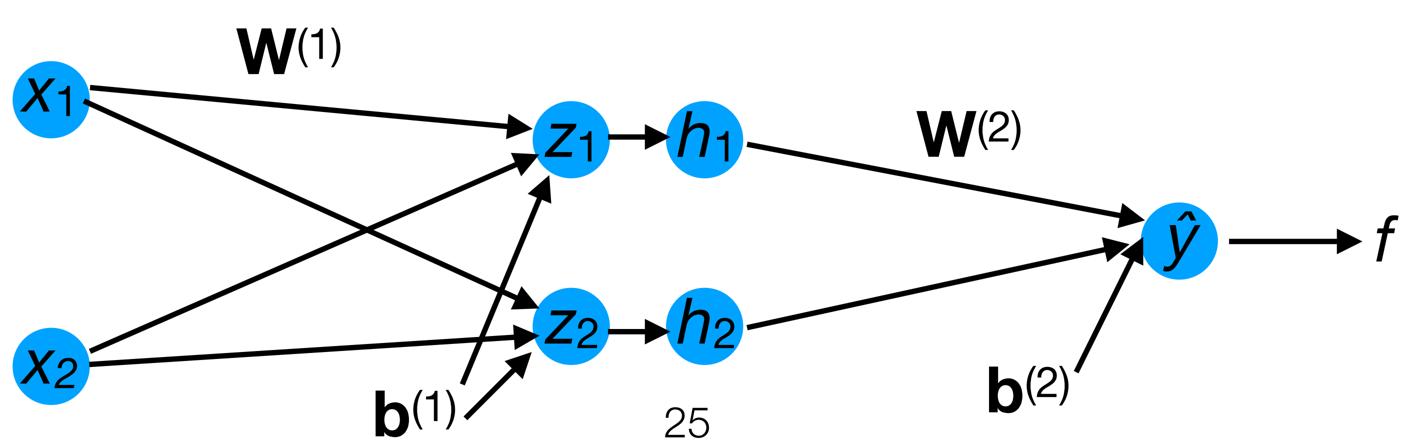


Exercise

- Let's next look at the derivative of $\mathbf{z}$ w.r.t. $\mathbf{W}^{(1)}$, where:

$$\mathbf{W}^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} \end{bmatrix}$$

- Derive the Jacobian matrix for $d\mathbf{z}/d\text{vec}\mathbf{W}^{(1)}$:



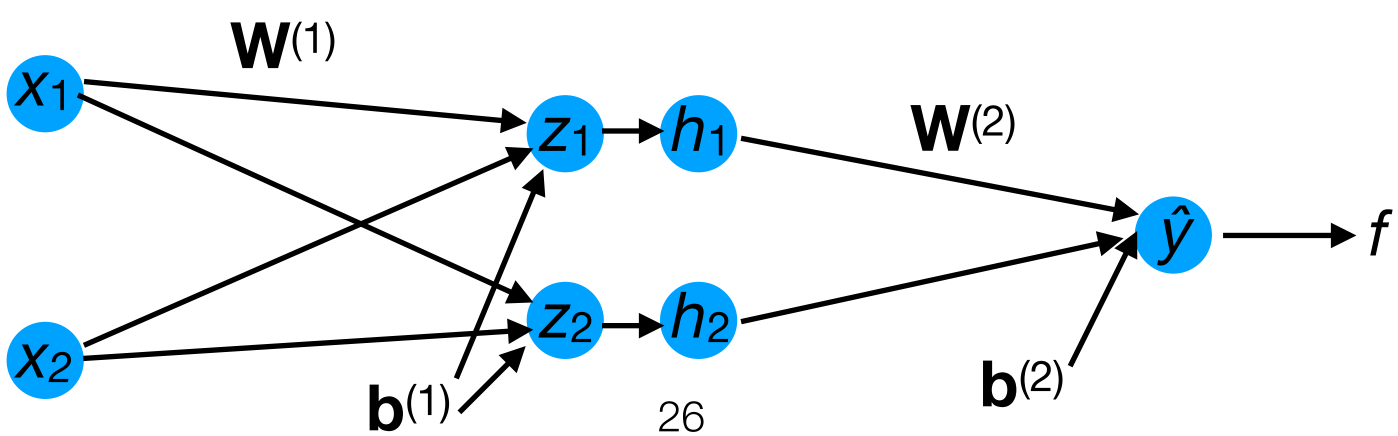
Exercise

- Let's next look at the derivative of **z** w.r.t. $\mathbf{W}^{(1)}$, where:

$$\mathbf{W}^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} \end{bmatrix}$$

- Derive the Jacobian matrix for d**z**/dvec$\mathbf{W}^{(1)}$:

$$\frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}^{(1)}]} = \left[\qquad \qquad \qquad \qquad \qquad \qquad \right]$$



Exercise

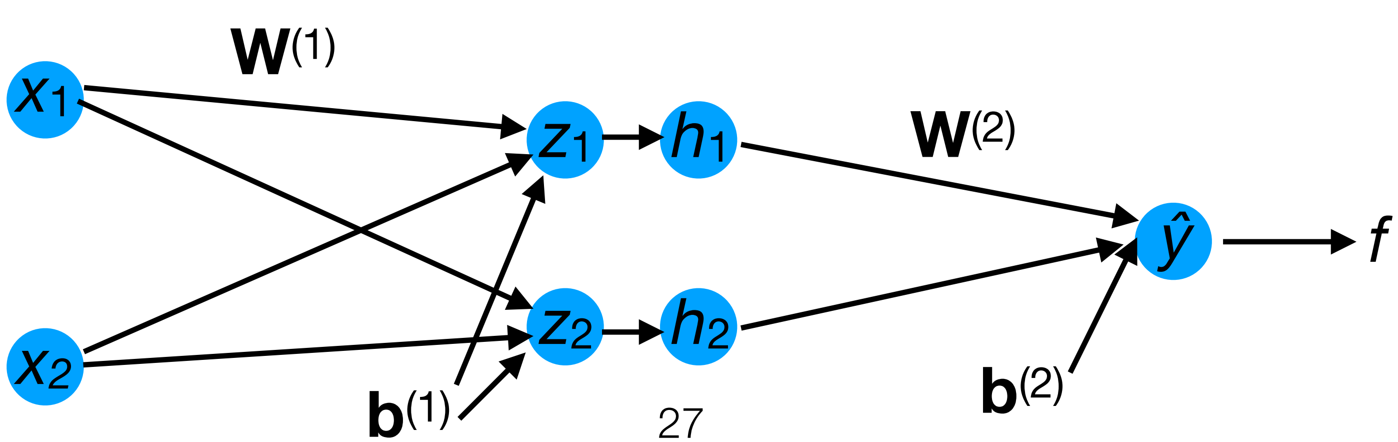
- Let's next look at the derivative of **z** w.r.t. $\mathbf{W}^{(1)}$, where:

$$\mathbf{W}^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} \end{bmatrix}$$

- Derive the Jacobian matrix for d**z**/dvec$\mathbf{W}^{(1)}$:

$$\frac{\partial \mathbf{z}}{\partial \mathrm{vec}[\mathbf{W}^{(1)}]} = \begin{bmatrix} \frac{\partial z_1}{\partial W_{11}^{(1)}} & \frac{\partial z_1}{\partial W_{12}^{(1)}} & \frac{\partial z_1}{\partial W_{21}^{(1)}} & \frac{\partial z_1}{\partial W_{22}^{(1)}} \\ \frac{\partial z_2}{\partial W_{11}^{(1)}} & \frac{\partial z_2}{\partial W_{12}^{(1)}} & \frac{\partial z_2}{\partial W_{21}^{(1)}} & \frac{\partial z_2}{\partial W_{22}^{(1)}} \end{bmatrix}$$

$$= \begin{bmatrix} x_1 & x_2 & 0 & 0 \\ 0 & 0 & x_1 & x_2 \end{bmatrix}$$



Computing the gradients

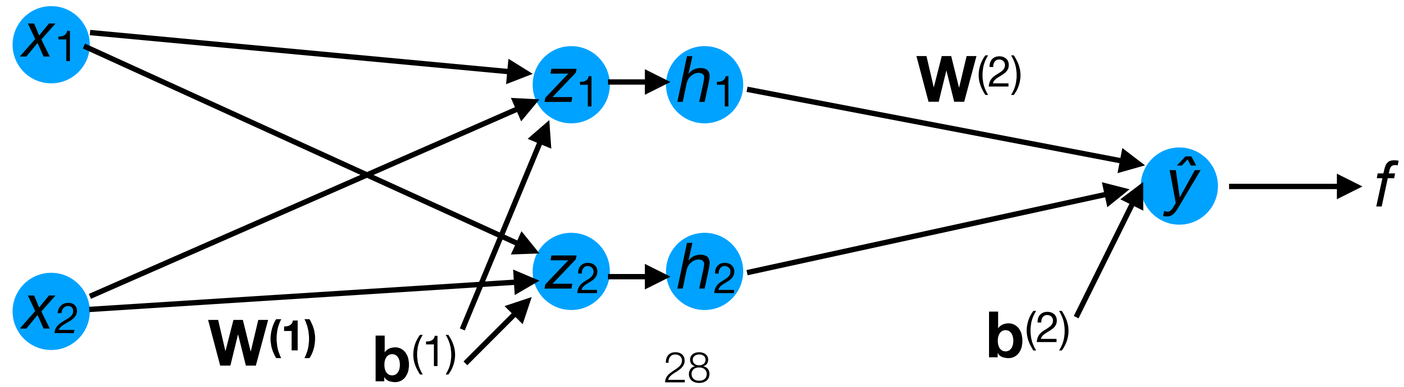
- Let's derive each gradient term in turn (for n=1):

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

- How does f depend on $\hat{y}$?

$$f(\mathbf{x}; \mathbf{W}^{(2)}) = \frac{1}{2}(\hat{y} - y)^2$$

$$\implies \frac{\partial f}{\partial \hat{y}} = (\hat{y} - y)$$



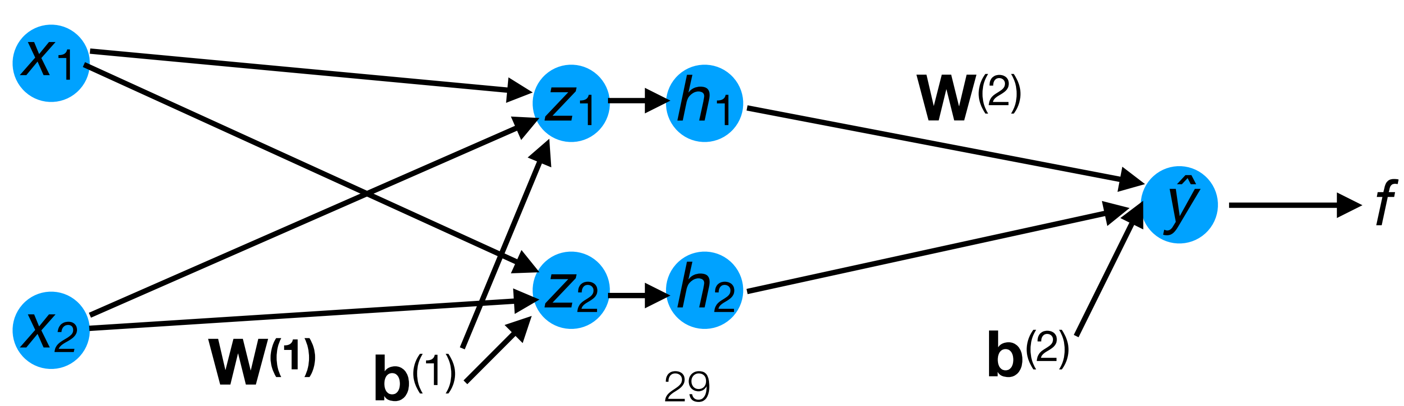
Computing the gradients

- Let's derive each gradient term in turn (for n=1):

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

- How does $\hat{y}$ depend on $\mathbf{W}^{(2)}$?

$$\begin{aligned} \hat{y} &= \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)} \\ &= W_1^{(2)} h_1 + W_2^{(2)} h_2 + b^{(2)} \\ \implies \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}} &= \left[\begin{array}{cc} h_1 & h_2 \end{array} \right] \\ &= \mathbf{h}^\top \end{aligned}$$



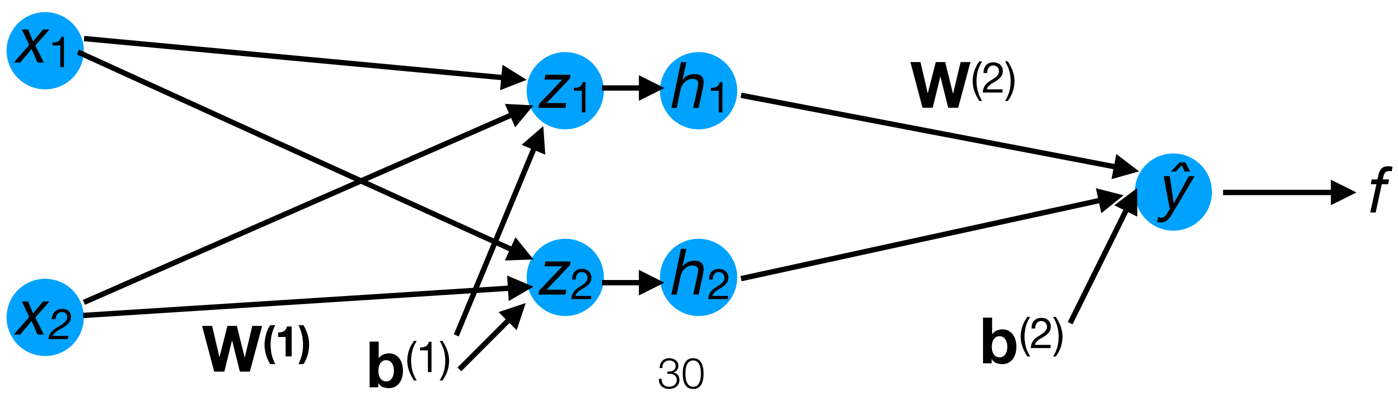
Computing the gradients

- Let's derive each gradient term in turn (for n=1):

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

- How does $\hat{y}$ depend on $\mathbf{W}^{(2)}$?

$$\begin{aligned} \hat{y} &= \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)} \\ &= W_1^{(2)} h_1 + W_2^{(2)} h_2 + b^{(2)} \\ \implies \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}} &= \left[\begin{array}{cc} h_1 & h_2 \end{array}\right] \\ &= \mathbf{h}^\top \end{aligned}$$



Jacob Whitehill, WPI

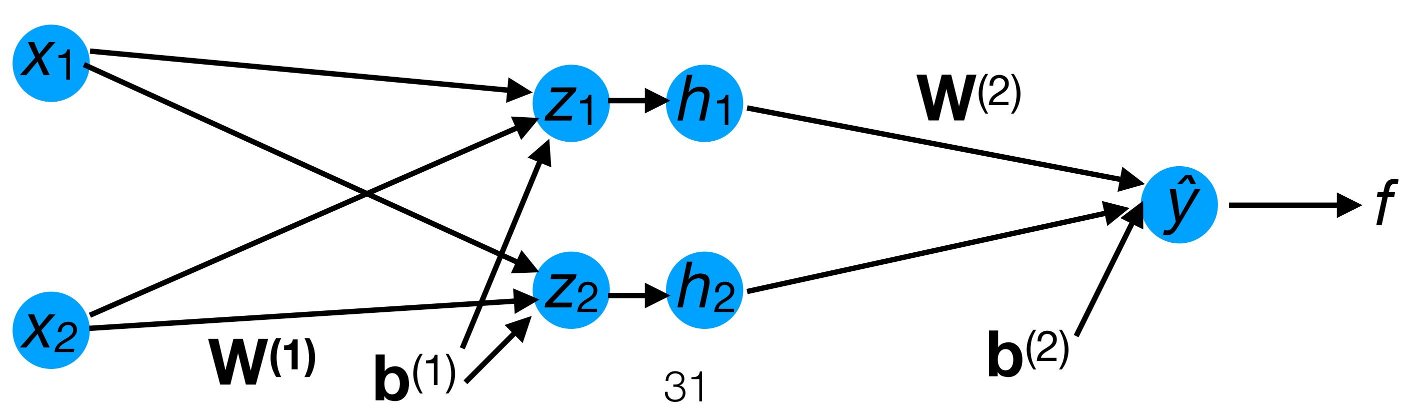
Computing the gradients

- Let's derive each gradient term in turn (for *n*=1):

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

- How does $\hat{y}$ depend on $\mathbf{b}^{(2)}$?

$$\begin{aligned} \hat{y} &= \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)} \\ &= W_1^{(2)} h_1 + W_2^{(2)} h_2 + b^{(2)} \\ \Longrightarrow \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}} &= 1 \end{aligned}$$



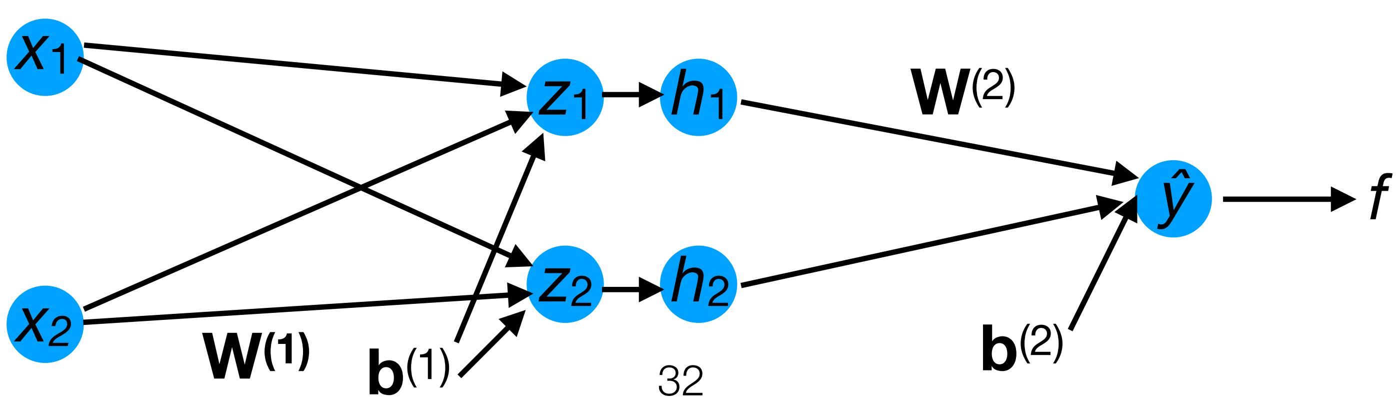
Computing the gradients

- Let's derive each gradient term in turn (for *n*=1):

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

- How does $\hat{y}$ depend on $\mathbf{b}^{(2)}$?

$$\begin{aligned} \hat{y} &= \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)} \\ &= W_1^{(2)} h_1 + W_2^{(2)} h_2 + b^{(2)} \\ \Longrightarrow \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}} &= 1 \end{aligned}$$



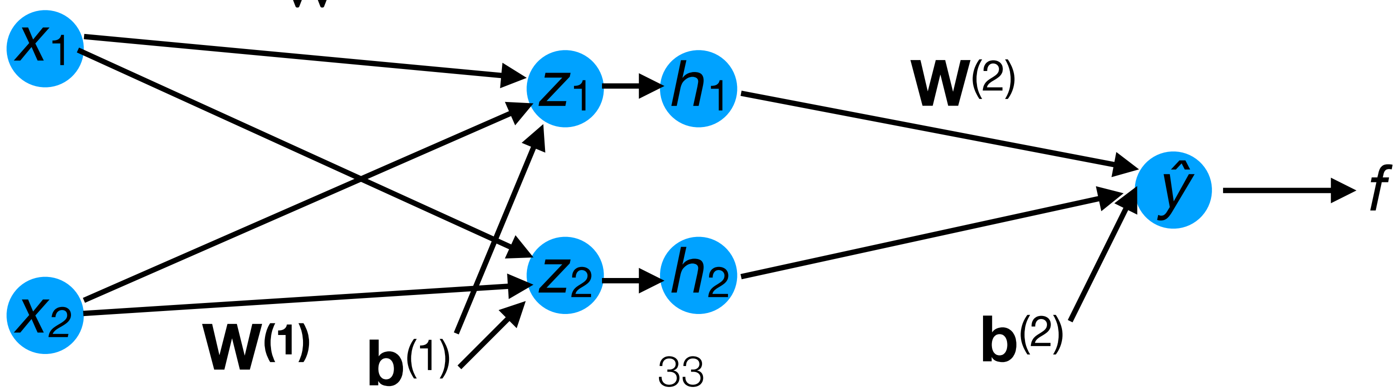
Computing the gradients

- Let's derive each gradient term in turn (for n=1):

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

- How does $\hat{y}$ depend on **h**?

$$\begin{aligned}
\hat{y} &= \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)} \\
&= W_1^{(2)} h_1 + W_2^{(2)} h_2 + b^{(2)} \\
\Longrightarrow \frac{\partial \hat{y}}{\partial \mathbf{h}} &= \left[\begin{array}{cc} \frac{\partial \hat{y}}{\partial h_1} & \frac{\partial \hat{y}}{\partial h_2} \end{array} \right] \\
&= \left[\begin{array}{cc} W_1^{(2)} & W_2^{(2)} \end{array} \right] \\
&= \mathbf{W}^{(2)}
\end{aligned}$$



Jacob Whitehill, WPI

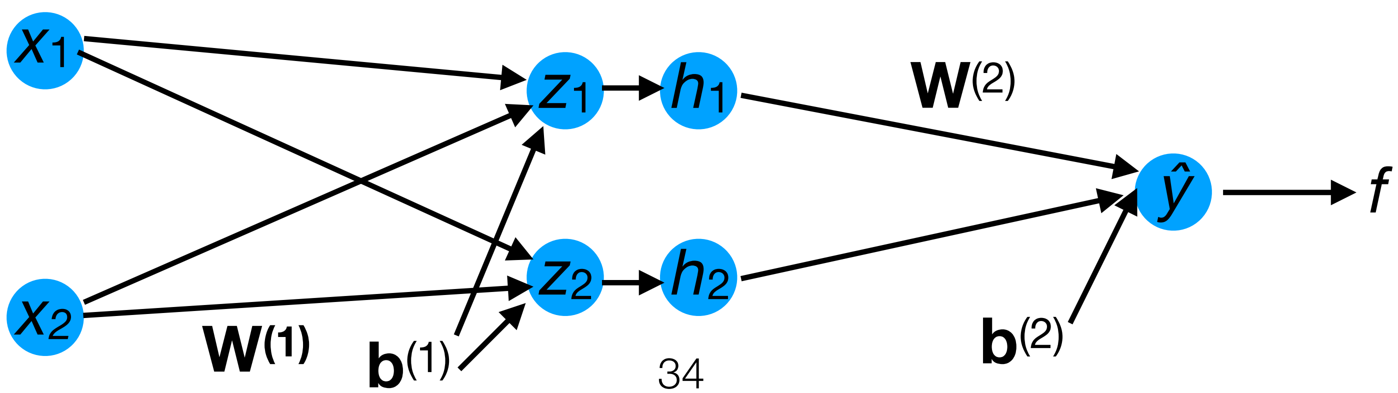
Computing the gradients

- Let's derive each gradient term in turn (for *n*=1):

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

- How does **h** depend on **z**?

$$\mathbf{h} = \begin{bmatrix} \mathrm{relu}(\mathbf{z}_1) \\ \mathrm{relu}(\mathbf{z}_2) \end{bmatrix}$$



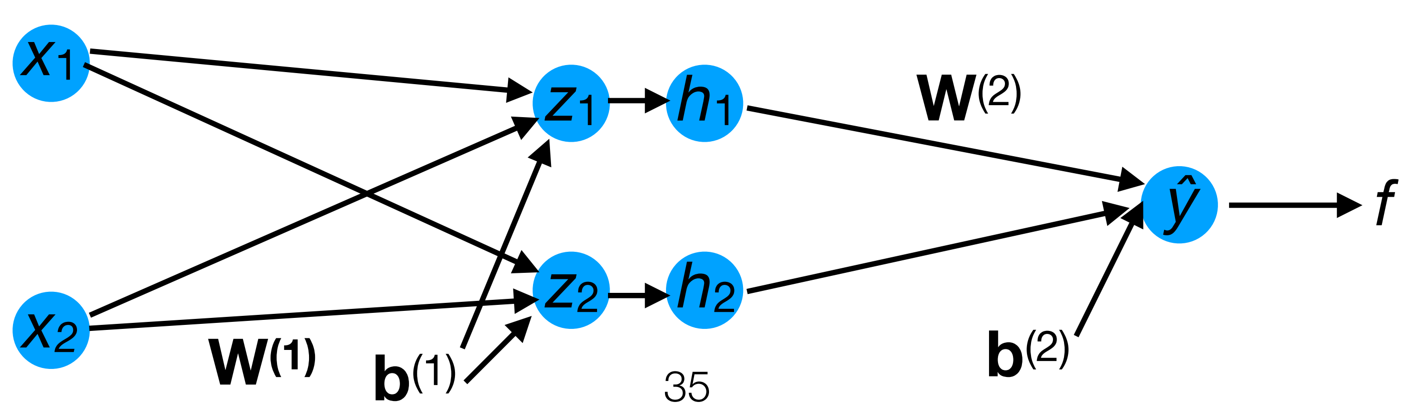
Computing the gradients

- Let's derive each gradient term in turn (for *n*=1):

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

- How does **h** depend on **z**?

$$\begin{aligned} \mathbf{h} &= \begin{bmatrix} \mathrm{relu}(\mathbf{z}_1) \\ \mathrm{relu}(\mathbf{z}_2) \end{bmatrix} \\ \Longrightarrow \frac{\partial \mathbf{h}}{\partial \mathbf{z}} &= \begin{bmatrix} \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1} & \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_2} \\ \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_1} & \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2} \end{bmatrix} \end{aligned}$$



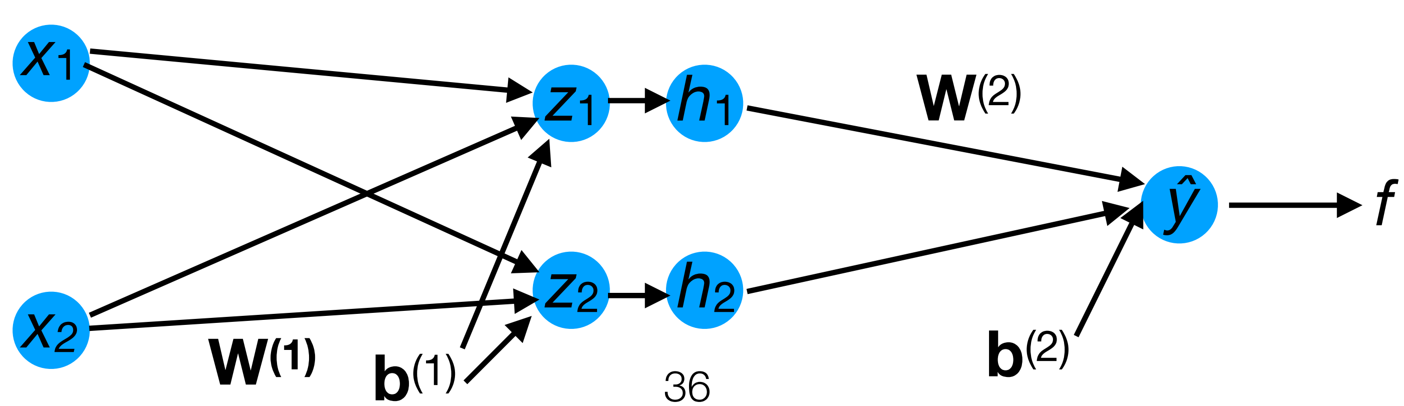
Computing the gradients

- Let’s derive each gradient term in turn (for *n*=1):

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

- How does **h** depend on **z**?

$$\begin{aligned} \mathbf{h} &= \begin{bmatrix} \mathrm{relu}(\mathbf{z}_1) \\ \mathrm{relu}(\mathbf{z}_2) \end{bmatrix} \\ \Longrightarrow \frac{\partial \mathbf{h}}{\partial \mathbf{z}} &= \begin{bmatrix} \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1} & \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_2} \\ \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_1} & \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2} \end{bmatrix} \\ &= \begin{bmatrix} \mathrm{relu}'(\mathbf{z}_1) & 0 \\ 0 & \mathrm{relu}'(\mathbf{z}_2) \end{bmatrix} \end{aligned}$$



Jacob Whitehill, WPI

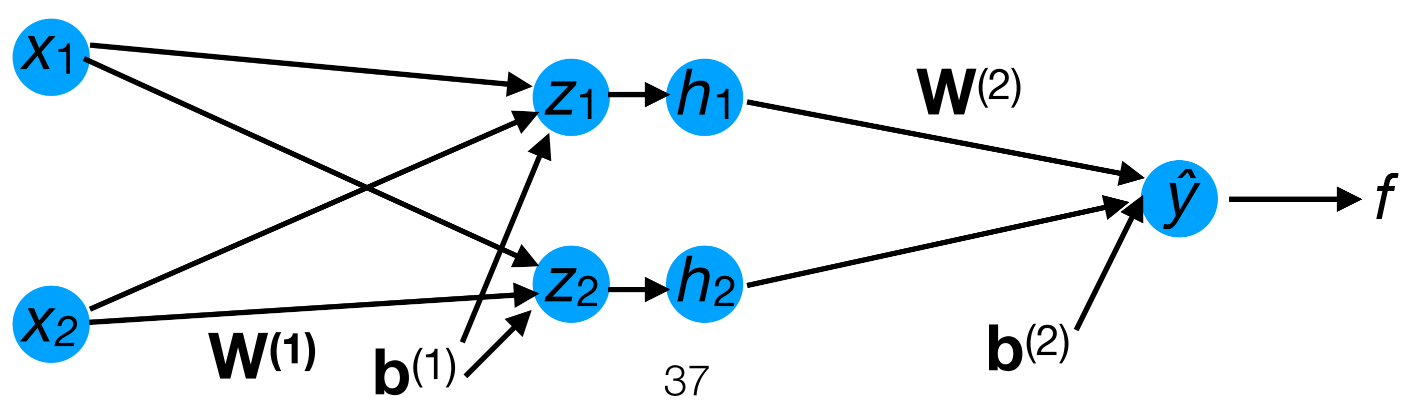
Computing the gradients

- Let's derive each gradient term in turn (for n=1):

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

- How does **z** depend on $\mathbf{W}^{(1)}$?

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$



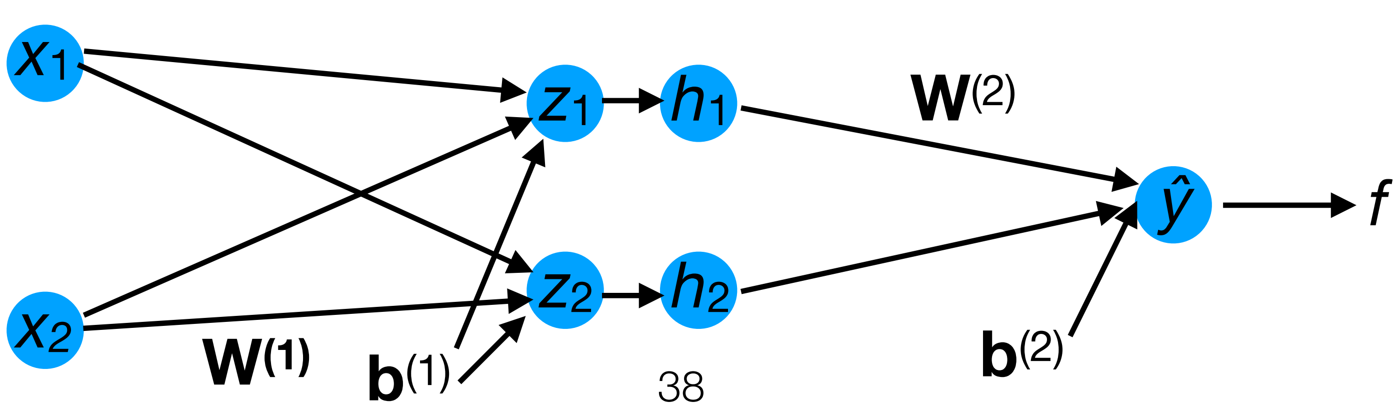
Computing the gradients

- Let's derive each gradient term in turn (for n=1):

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

- How does **z** depend on **W**$^{(1)}$?

$$\begin{aligned} \mathbf{z} &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\ \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} &= \begin{bmatrix} W_1^{(1)} & W_2^{(1)} \\ W_3^{(1)} & W_4^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} \end{aligned}$$



Jacob Whitehill, WPI

Computing the gradients

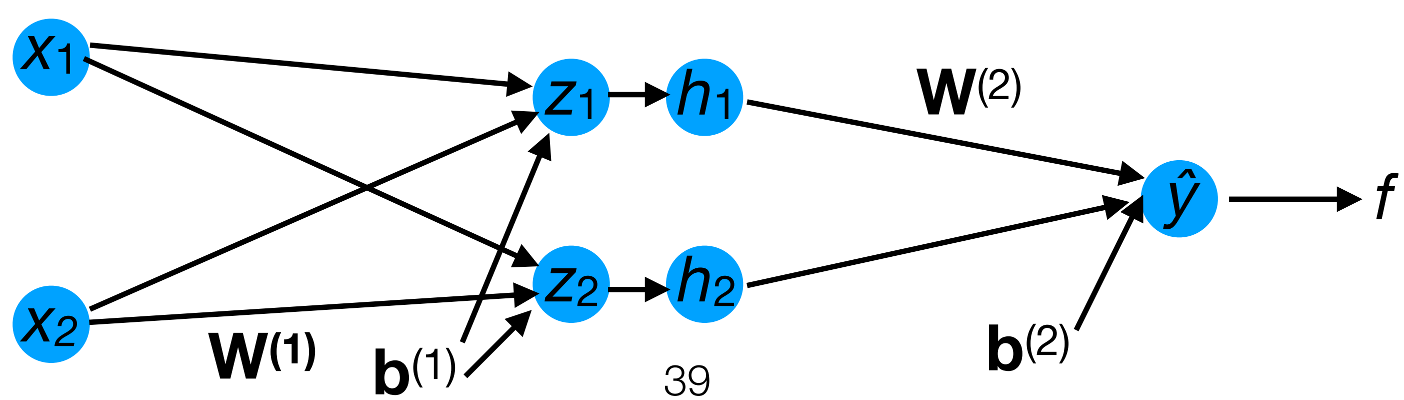
- Let's derive each gradient term in turn (for n=1):

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

- How does **z** depend on $\mathbf{W}^{(1)}$?

$$\begin{aligned}
\mathbf{z} &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\
\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} &= \begin{bmatrix} W_1^{(1)} & W_2^{(1)} \\ W_3^{(1)} & W_4^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} \\
\Longrightarrow \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}} &= \begin{bmatrix} \frac{\partial z_1}{\partial W_1^{(1)}} & \frac{\partial z_1}{\partial W_2^{(1)}} & \frac{\partial z_1}{\partial W_3^{(1)}} & \frac{\partial z_1}{\partial W_4^{(1)}} \\ \frac{\partial z_2}{\partial W_1^{(1)}} & \frac{\partial z_2}{\partial W_2^{(1)}} & \frac{\partial z_2}{\partial W_3^{(1)}} & \frac{\partial z_2}{\partial W_4^{(1)}} \end{bmatrix} \\
&= \begin{bmatrix} x_1 & x_2 & 0 & 0 \\ 0 & 0 & x_1 & x_2 \end{bmatrix}
\end{aligned}$$

For Jacobian matrix, we have to treat $\mathbf{W}^{(1)}$ as if it were a *vector*.



Jacob Whitehill, WPI

Analytical simplification

- We can now finally derive the gradient update for $\mathbf{W}^{(1)}$:

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

Analytical simplification

- We can now finally derive the gradient update for $\mathbf{W}^{(1)}$:

$$\begin{aligned}\frac{\partial f}{\partial \mathbf{W}^{(1)}} &= \frac{\partial f}{\partial \hat{\mathbf{y}}}\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}}\frac{\partial \mathbf{h}}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}} \\ &= (\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)} \begin{bmatrix} \mathrm{relu}'(\mathbf{z}_1) & 0 \\ 0 & \mathrm{relu}'(\mathbf{z}_2) \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & 0 & 0 \\ 0 & 0 & \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix}\end{aligned}$$

Analytical simplification

- We can now finally derive the gradient update for $\mathbf{W}^{(1)}$:

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

$$= (\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)} \boxed{\begin{bmatrix} \mathrm{relu}'(\mathbf{z}_1) & 0 \\ 0 & \mathrm{relu}'(\mathbf{z}_2) \end{bmatrix}} \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & 0 & 0 \\ 0 & 0 & \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix}$$

$$= \left(\left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)} \right) \boxed{\odot \begin{bmatrix} \mathrm{relu}'(\mathbf{z}_1) & \mathrm{relu}'(\mathbf{z}_2) \end{bmatrix}} \right) \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & 0 & 0 \\ 0 & 0 & \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix}$$

since multiplying by a diagonal matrix is equivalent to element-wise (Hadamard) product.

Analytical simplification

- We can now finally derive the gradient update for **W**(1):

$$
\begin{aligned}
\frac{\partial f}{\partial \mathbf{W}^{(1)}} &= \frac{\partial f}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}} \\
&= (\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)} \begin{bmatrix} \mathrm{relu}'(\mathbf{z}_1) & 0 \\ 0 & \mathrm{relu}'(\mathbf{z}_2) \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & 0 & 0 \\ 0 & 0 & \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix} \\
&= \left(\left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)} \right) \odot \begin{bmatrix} \mathrm{relu}'(\mathbf{z}_1) & \mathrm{relu}'(\mathbf{z}_2) \end{bmatrix} \right) \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & 0 & 0 \\ 0 & 0 & \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{g}_1 & \mathbf{g}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & 0 & 0 \\ 0 & 0 & \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix}
\end{aligned}
$$

To simplify notation, let's define a new vector that equals the first few terms.

Jacob Whitehill, WPI

Analytical simplification

- We can now finally derive the gradient update for $\mathbf{W}^{(1)}$:

$$
\begin{aligned}
\frac{\partial f}{\partial \mathbf{W}^{(1)}} &= \frac{\partial f}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}} \\
&= (\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)} \begin{bmatrix} \text{relu}'(\mathbf{z}_1) & 0 \\ 0 & \text{relu}'(\mathbf{z}_2) \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & 0 & 0 \\ 0 & 0 & \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix} \\
&= \left(\left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)} \right) \odot \begin{bmatrix} \text{relu}'(\mathbf{z}_1) & \text{relu}'(\mathbf{z}_2) \end{bmatrix} \right) \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & 0 & 0 \\ 0 & 0 & \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{g}_1 & \mathbf{g}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & 0 & 0 \\ 0 & 0 & \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{g}_1 \mathbf{x}_1 & \mathbf{g}_1 \mathbf{x}_2 & \mathbf{g}_2 \mathbf{x}_1 & \mathbf{g}_2 \mathbf{x}_2 \end{bmatrix}
\end{aligned}
$$

Jacob Whitehill, WPI

Analytical simplification

- We can now finally derive the gradient update for $\mathbf{W}^{(1)}$:

$$
\begin{aligned}
\frac{\partial f}{\partial \mathbf{W}^{(1)}} &= \frac{\partial f}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}} \\
&= (\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)} \begin{bmatrix} \mathrm{relu}'(\mathbf{z}_1) & 0 \\ 0 & \mathrm{relu}'(\mathbf{z}_2) \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & 0 & 0 \\ 0 & 0 & \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix} \\
&= \left(\left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)} \right) \odot \begin{bmatrix} \mathrm{relu}'(\mathbf{z}_1) & \mathrm{relu}'(\mathbf{z}_2) \end{bmatrix} \right) \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & 0 & 0 \\ 0 & 0 & \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{g}_1 & \mathbf{g}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & 0 & 0 \\ 0 & 0 & \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{g}_1\mathbf{x}_1 & \mathbf{g}_1\mathbf{x}_2 & \mathbf{g}_2\mathbf{x}_1 & \mathbf{g}_2\mathbf{x}_2 \end{bmatrix} \\
\Longrightarrow \nabla_{\mathbf{W}^{(1)}} f &= \mathbf{g}\mathbf{x}^\top
\end{aligned}
$$

Outer product

Gradient check

Gradient check

- When manually deriving gradient terms, it is very useful to verify their correctness using **numerical differentiation.**

- We can approximate the true gradient of a function *f* using **finite differences**:

$$
\begin{aligned}
\frac{\partial f}{\partial x}(x) &= \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \\
&\approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \text{ for small } \Delta x
\end{aligned}
$$

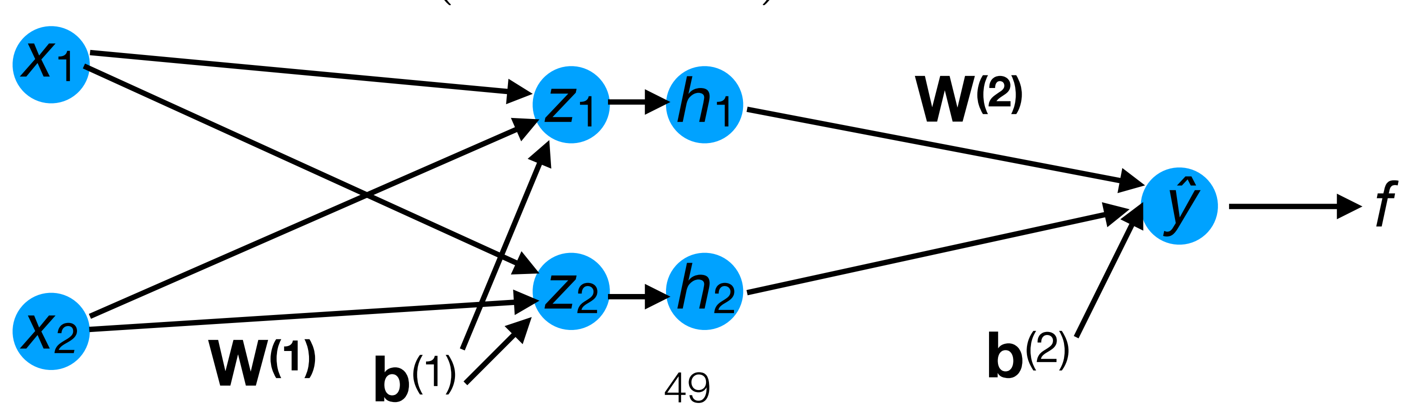
Weight initialization

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?

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

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



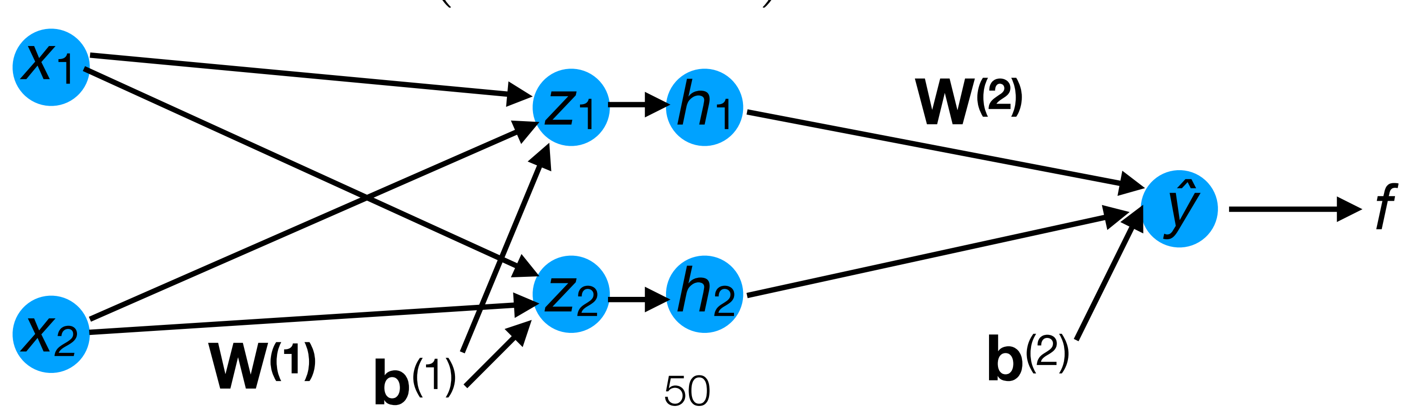
Weight initialization: example

- 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.

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

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



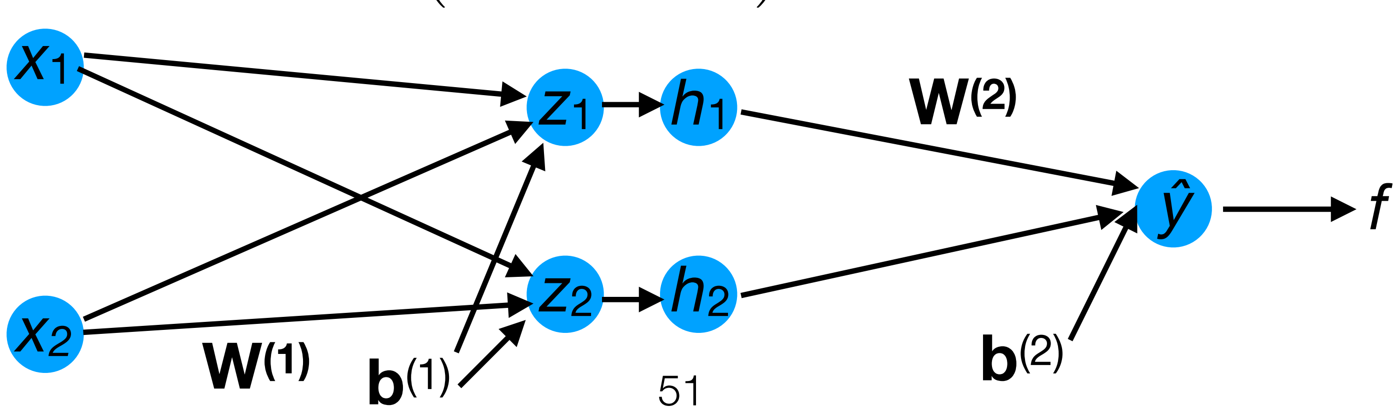
Weight initialization: example

- 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:

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

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



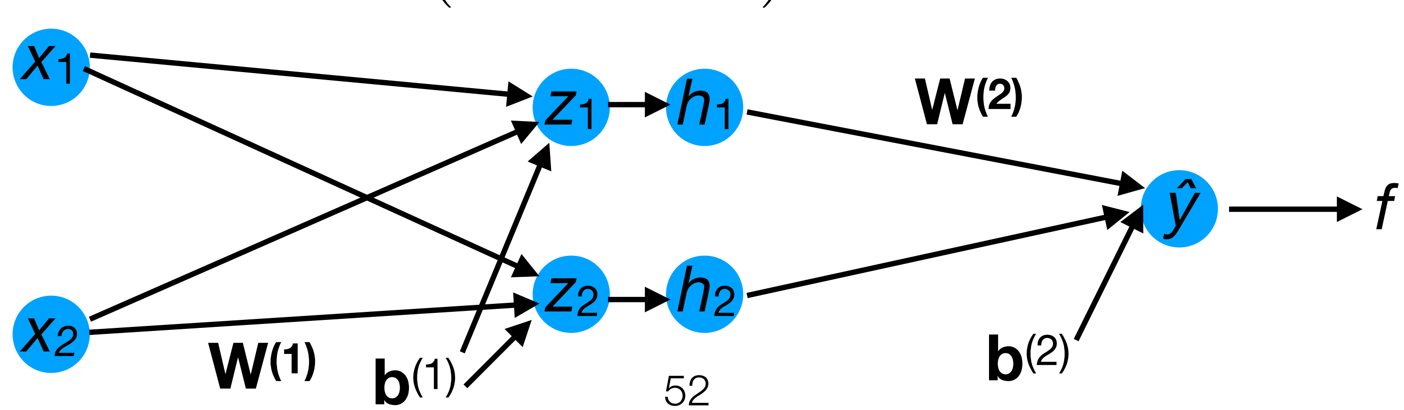
Weight initialization: example

- 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

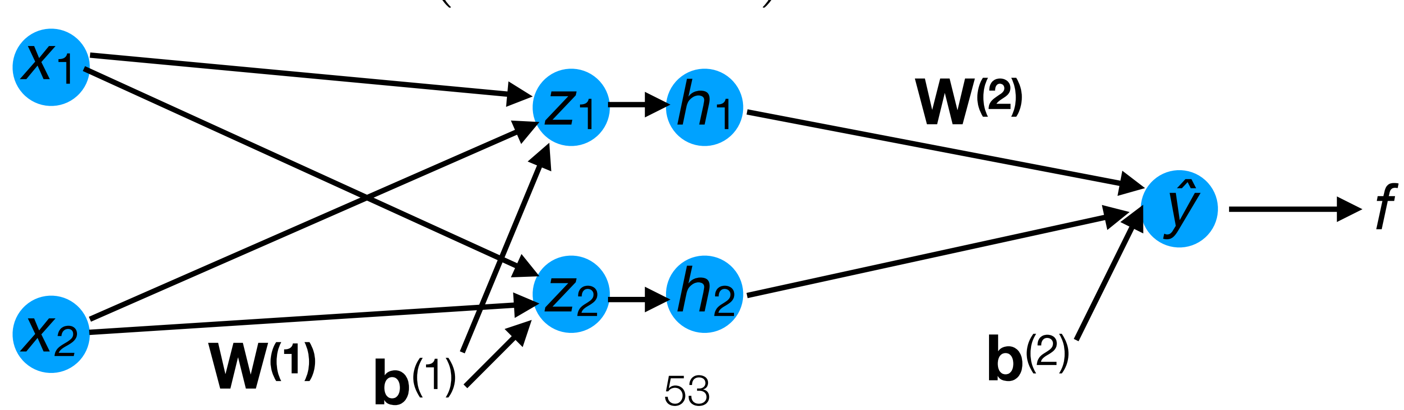
- Because the gradients w.r.t. $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$, and $\mathbf{b}^{(1)}$ are all 0, they will *never change*.

- Only $\mathbf{b}^{(2)}$ will change (to the mean of the target values y).

During backwards propagation, we have:

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

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

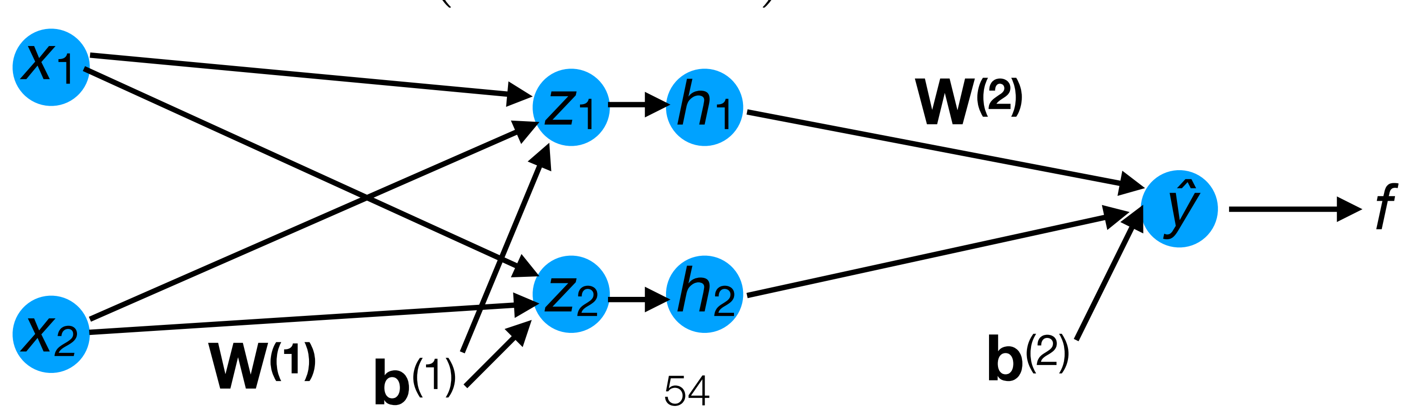


Weight initialization: exercise 1

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

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

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



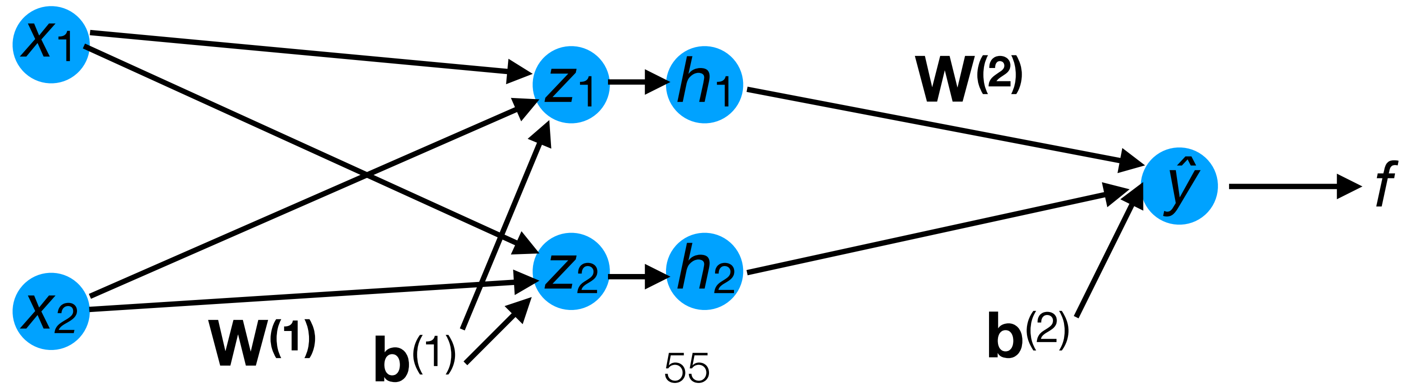
Jacob Whitehill, WPI

Weight initialization: exercise 1

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

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

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



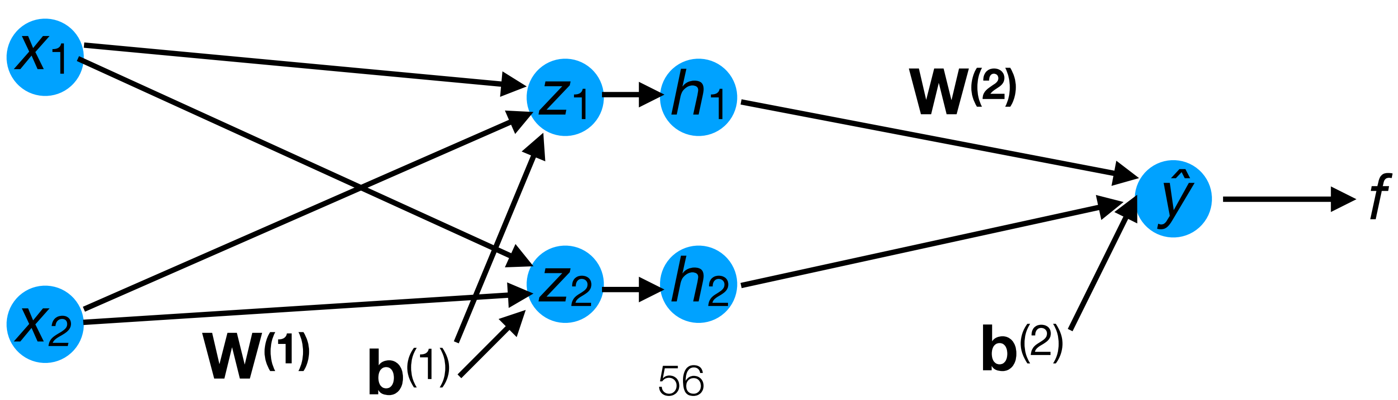
Jacob Whitehill, WPI

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).

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

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

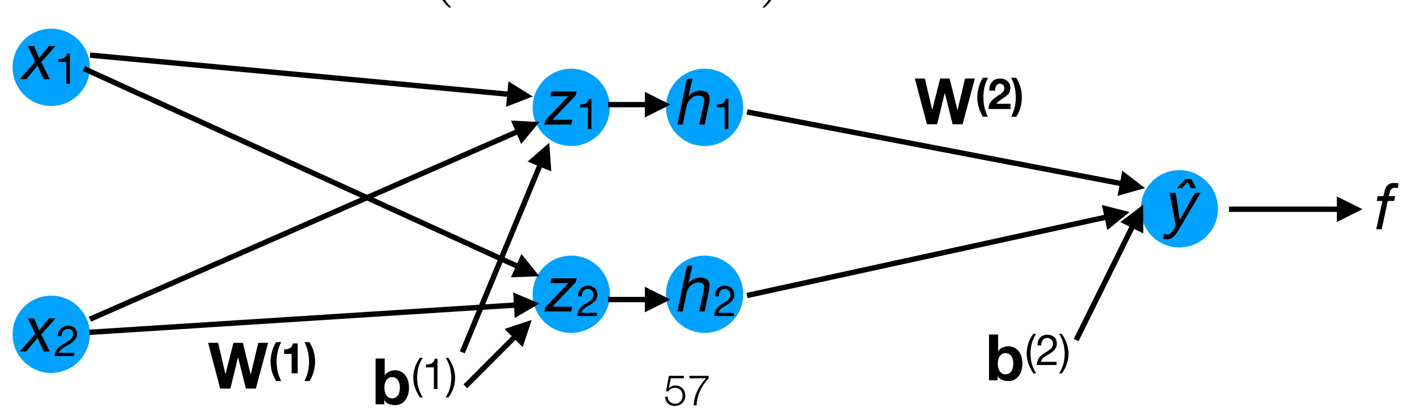


Weight initialization: exercise 2

- 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?

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

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

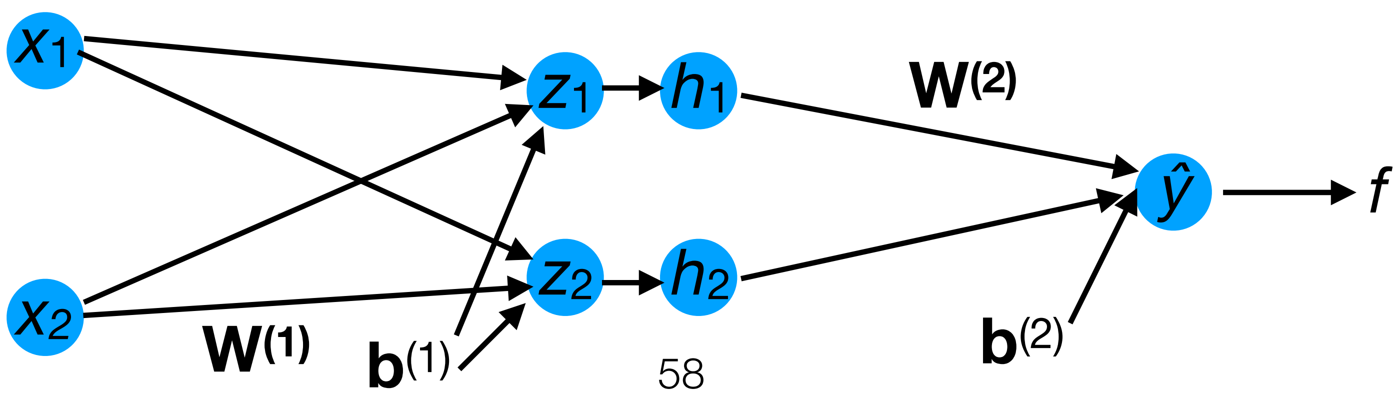


Weight initialization: exercise 2

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

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

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



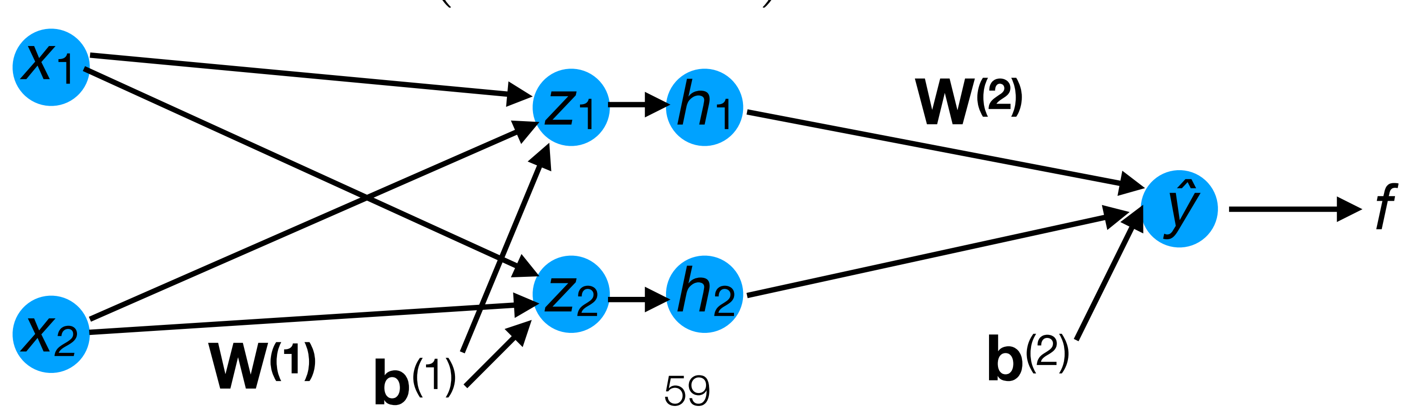
Jacob Whitehill, WPI

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.

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

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

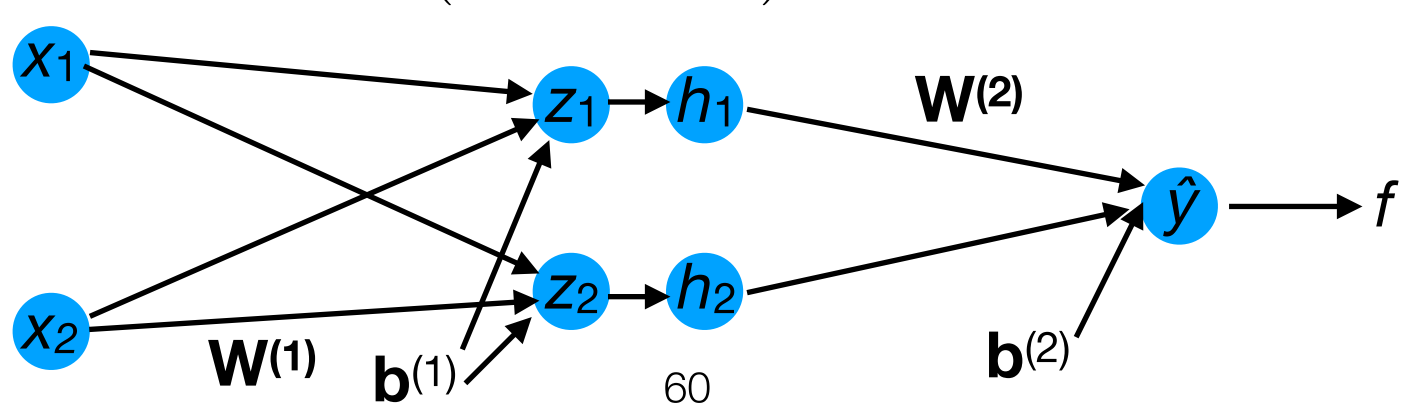


Weight initialization: exercise 2

- 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).

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

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

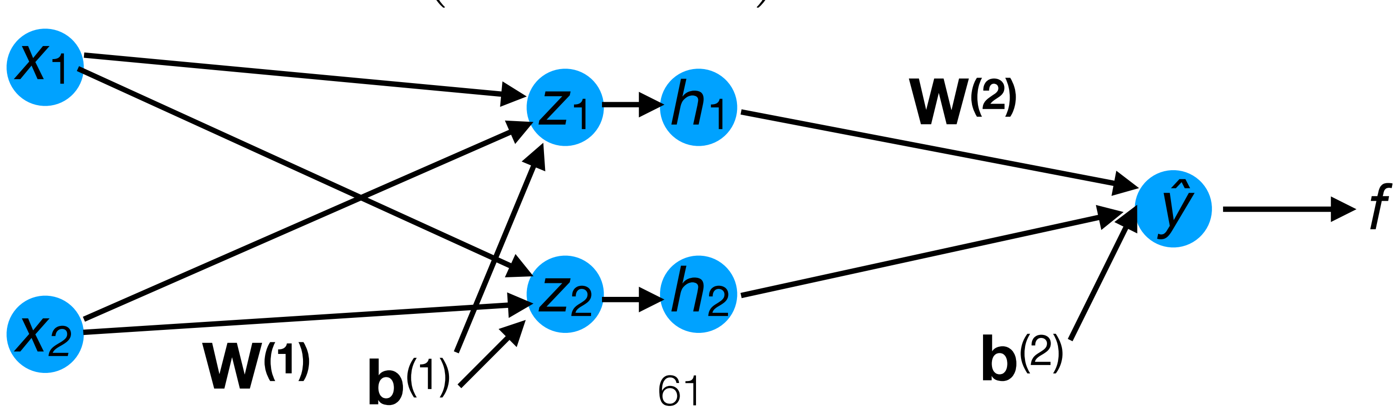


Weight initialization: exercise 3

- Suppose that each weight matrix & bias vector consists of the same row *repeated many times*.
- What will happen during SGD?

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

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



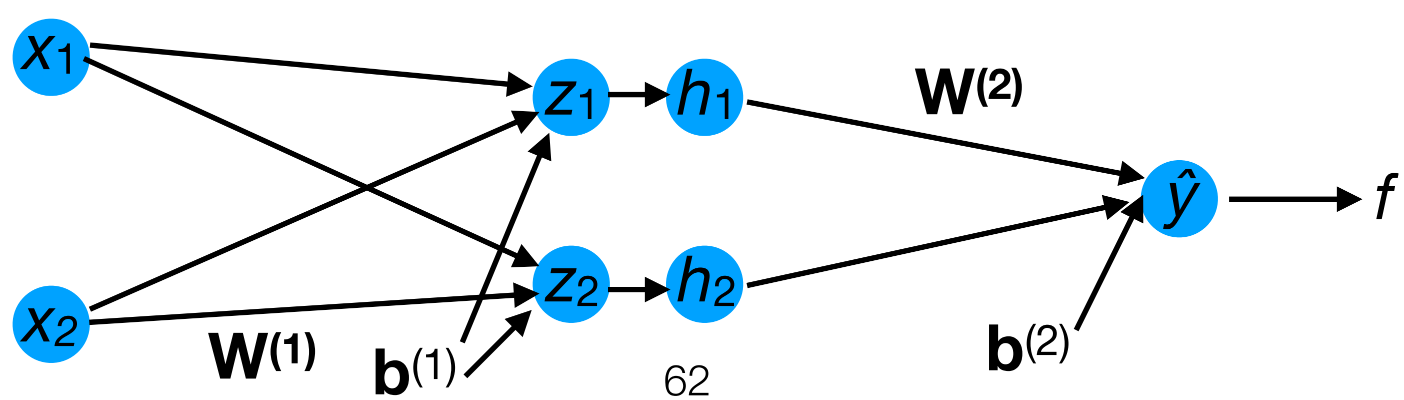
Jacob Whitehill, WPI

Weight initialization: exercise 3

- In this case, every node of **h** (and **z**) has the *same value*.
- Therefore, the gradient update to each row of $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ has the *same value*.
- The NN is performing redundant computation — although it has 2 hidden units, it might as well just have 1!

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

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



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 Gaussian with deviation $1/\sqrt{\text{cols}(\mathbf{W}^{(j)})}$.
    - Within certain NNs, helps to ensure that the gradients are usually non-zero.

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 Gaussian with deviation $1/\sqrt{\text{cols}(\mathbf{W}^{(j)})}$.
    - Within certain NNs, helps to ensure that the gradients are usually non-zero.
  - *Optional*: orthogonalize the rows of $\mathbf{W}^{(j)}$ to reduce correlation between different units of the pre-activation layer $\mathbf{z}^{(j)}$.

Why “deep” learning?

Why “deep” learning?

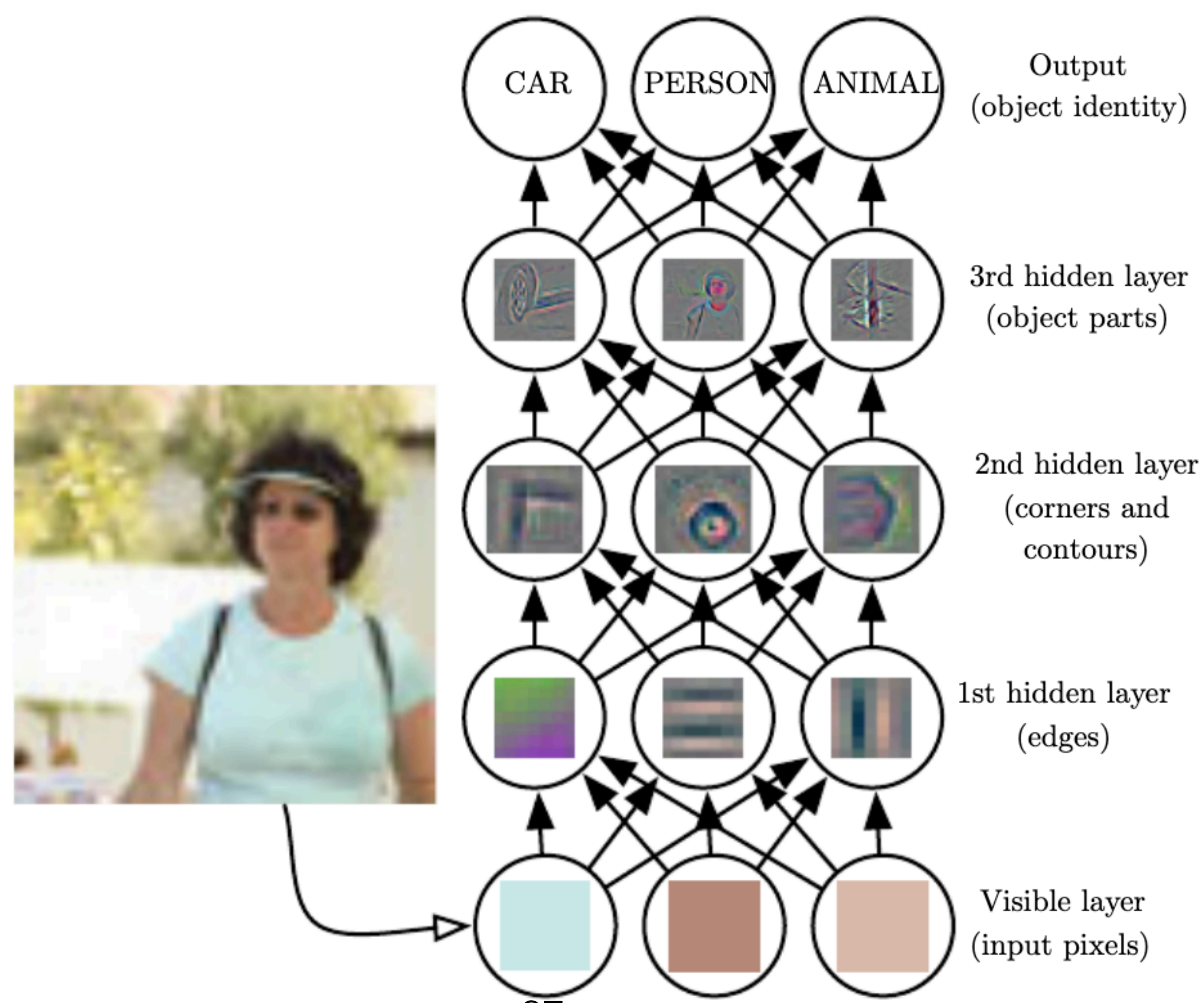
- One of the reasons for the resurgence of NNs around 2010 is that increasing the depth is very powerful.

- There is some theory, and an abundance of empirical results, that deeper networks are more powerful (can represent more complicated functions) and more accurate than shallow networks.

- Why?

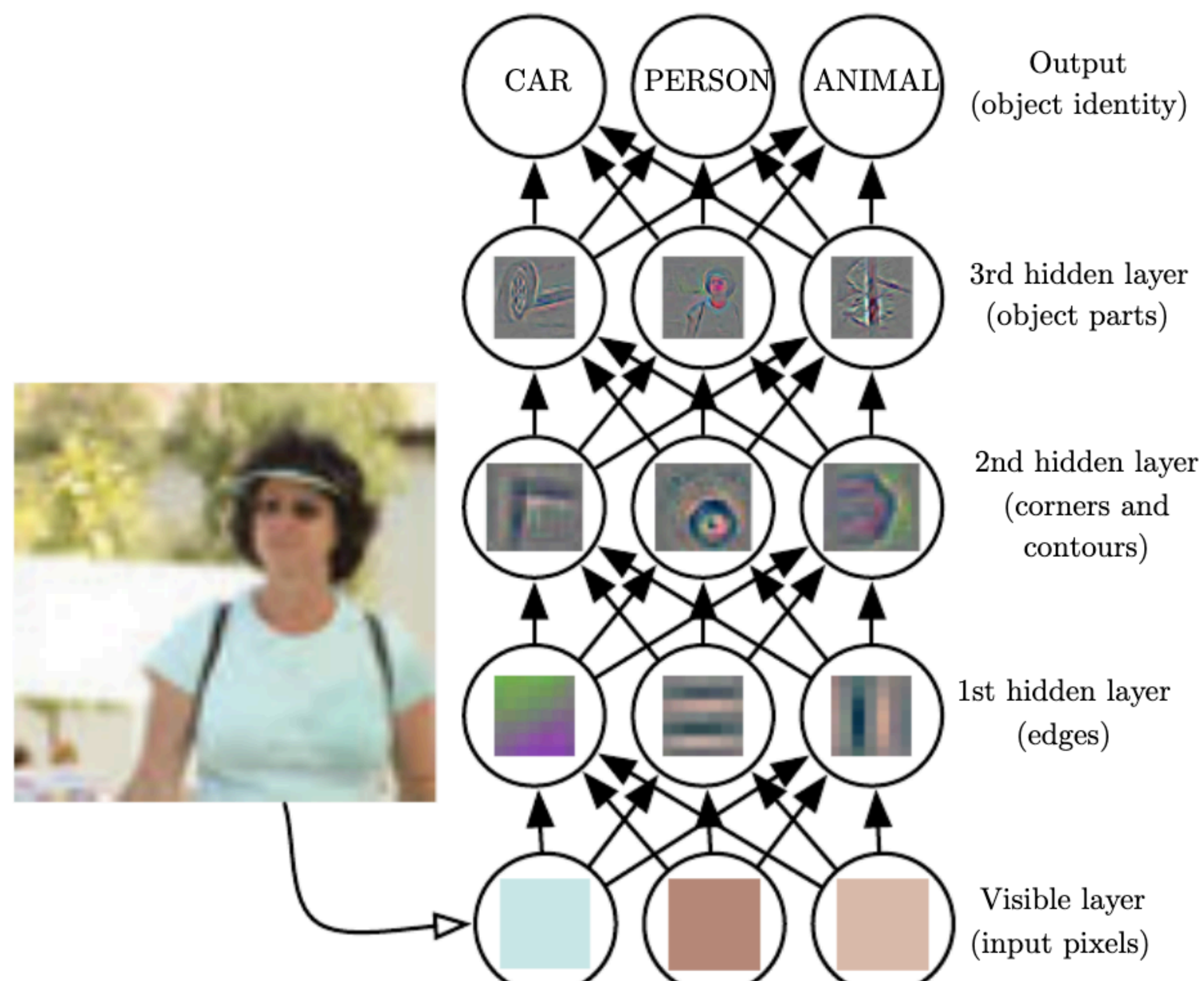
Why "deep" learning?

- One explanation for the success of deep NNs is that each layer can represent **increasingly abstract representations** of the input (from Zeiler & Fergus 2014):



Why "deep" learning?

- The hidden units can represent the content of the input in a compact way.



Demo

- As a simple illustration, let's consider two feed-forward NNs with the **same number of weights (60)**:
  - 3 layers with dimensions: (2, 20, 1)
  - 6 layers with dimensions: (2, 4, 4, 4, 4, 1)
- If we initialize their weights randomly, and use a non-linear activation function (`abs` in this case), then what kinds of functions can they represent?

Difficulty in deep learning

- Training very deep neural networks has traditionally been very difficult for two main reasons:
  - **Vanishing gradients**: gradients of weights tends to 0 as depth increases. => no learning
  - **Exploding gradient**: gradients of weights tends to infinity as depth increases. => requires small learning rate.

Difficulty in deep learning

- Since ~2008, DL researchers have:
    - Harnessed faster hardware to achieve good accuracy more quickly.
    - Found better architectures and training algorithms that prevent the gradients from behaving “badly”.

Practical suggestions

Tackle the problem based on your past experience

- Think of the machine learning problem you have solved that is *most similar* to the one you are tackling now, e.g.:
    - Same task (object detection, semantic segmentation, speaker diarization, etc).
    - Same dataset (MNIST, ImageNet, Faces in the Wild, etc.)
- How did you solve it?
    - NN design
    - Training procedures
    - Hyperparameters
- Start with the approach you used before, make sensible adjustments, and start training.

Start small

- Debug your code on a smaller version of the problem:
  - Subset of data (e.g., just 10K images instead of 50K)
  - Subset of classes (e.g., just 2 MNIST classes instead of 10)
- Advantages:
  - Less time for initialization.
  - Less time for training.
  - Fewer variables to examine during forward/back-propagation

Start simple

- Until you gain confidence & experience, train a simple model first:
  - They’re often faster to train and easier to debug than more powerful models.

Start simple

- Until you gain confidence & experience, train a simple model first:
  - They're often faster to train and easier to debug than more powerful models.
- Make sure your model's accuracy is above chance:
  - Take the prior class probabilities into account! (If the classes are 90/10, then the baseline rate for guessing the majority class is 90%.)
  - Make sure your model is not always predicting the dominant class.

But not too simple

- Sometimes a complex neural network is necessary to solve a problem with high accuracy.
- For instance, a 3-layer NN may not have enough representational power to analyze complex images.
- You might need to use a standard architecture (e.g., ResNet) — but use one of the simpler versions (e.g., 50 instead of 100 layers).

Start small & simple

- Try to find a model (and hyper-parameters) whose training loss decreases *smoothly*.
- Afterwards, increase the size of the training set and model complexity.

Regularization

- If there is a large divergence between training accuracy and testing accuracy (i.e., overfitting), then try regularizing the model:

    - Increasing L_1, L_2 regularization strength.

    - Adding/increasing dropout (for NNs).

    - Reducing number of training epochs (for NNs).

    - Synthesizing more training examples with label-preserving transformations (geometric & noise-based).

Hyper-parameter optimization

- Try a variety of hyper-parameters:
  - Find a few values manually that seem to work; use these as a guide to pick a reasonable range (e.g., for learning rate, 1e-5 to 1e0, spaced logarithmically).
  - Be disciplined about optimizing parameters on a validation set, not the test set!

Hyper-parameter optimization

- Try a variety of hyper-parameters:
  - When you have intuition, then it's sometimes worthwhile to watch the loss evolve over time.
  - When you do not have intuition, then automate the process and get on with your life — do not succumb to the temptation to watch learning curves like a movie.

Normalization

- It can be helpful to put every feature onto the same scale.
- In particular, the scale can interact with the L_2 regularization strength.

Normalization: example

- Suppose you are predicting tomorrow's temperature based on (1) today's temperature and (2) wind speed.
- Suppose we measure temperature in Kelvin and wind speed in km/h.
- Suppose the optimal weights w_1, w_2 for these two features, for L_2-regularized linear regression, are 1 and 2, i.e.:
    - $\hat{y} = w_1 t + w_2 s$ (t = today's temp, s = today's wind speed)
      $\hat{y} = 1*t + 2*s$

Normalization: example

- Now, suppose we change the units for wind speed from km/h to m/s.
    - E.g., 18 km/h = 5 m/s **Numerical values reduced by 3.6x**
- If we don't adjust our model weights w_1, w_2, then our predictions will be wrong:
    - $\hat{y} = 1 * t + 2 * s$
      $\hat{y}(4, 18) = 4 + 36 = 40$ **km/h**
      $\hat{y}(4, 5) = 4 + 10 = 14$ **m/s**

Normalization: example

- Because the numerical values of the wind speed were reduced by factor of 3.6, the corresponding weight w_2 must compensate by increasing by 3.6x, i.e.:

  - $\hat{y} = w_1 t + \tilde{w}_2 s$ (t = today's temp, s = today's wind speed)
    $\hat{y} = 1 * t + 3.6 * 2 * s$

- Without regularization, the training procedure (e.g., minimize f_{MSE}) will account for the change-of-scale seamlessly, i.e.:

$$\arg\min_{\tilde{w}_2} f_{\mathrm{MSE}}^{\mathrm{m/s}}(\cdot) \quad \mathbf{= 3.6 *} \quad \arg\min_{w_2} f_{\mathrm{MSE}}^{\mathrm{km/h}}(\cdot)$$

Normalization: example

- But with L_2 regularization, the issue is more complicated:

$$\arg\min_{\tilde{w_2}} \left[f_{\text{MSE}}^{\text{km/h}}(\cdot) + \frac{1}{2} w_2^2 \right]$$

- The regularization term "discourages" w_2 from growing too big:
  - When we rescale from km/h to m/s, the L_2 term prevents the weight w_2 from compensating exactly.

Normalization: recommendations

- For features in a finite range, try rescaling to [0,1] or [-1,1].
- For features in infinite range, try subtracting the mean and dividing by standard deviation (so that the distribution has zero-mean and unit standard deviation).

Jacob Whitehill, WPI