

CS/DS 541: Class 6

Jacob Whitehill

Jacobian matrices & the chain rule of multivariate calculus

Jacobian matrices

- Consider a function f that takes a vector as input *and produces a vector as output*, e.g.:

$$f \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} 2x_1 + x_2 \\ \pi x_2 \\ -x_1/x_2 \end{bmatrix}$$

- What is the set of f 's partial derivatives?

Jacobian matrices

- Consider a function f that takes a vector as input *and produces a vector as output*, e.g.:

$$f \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} 2x_1 + x_2 \\ \pi x_2 \\ -x_1/x_2 \end{bmatrix}$$

- What is the set of f 's partial derivatives?

$$\begin{array}{ll} \frac{\partial f_1}{\partial x_1} = 2 & \frac{\partial f_1}{\partial x_2} = 1 \\ \frac{\partial f_2}{\partial x_1} = 0 & \frac{\partial f_2}{\partial x_2} = \pi \\ \frac{\partial f_3}{\partial x_1} = -1/x_2 & \frac{\partial f_3}{\partial x_2} = x_1/x_2^2 \end{array}$$

Jacobian matrices

- Consider a function f that takes a vector as input *and produces a vector as output*, e.g.:

$$f \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} 2x_1 + x_2 \\ \pi x_2 \\ -x_1/x_2 \end{bmatrix}$$

- What is the set of f 's partial derivatives?

$$\begin{bmatrix} 2 & 1 \\ 0 & \pi \\ -1/x_2 & x_1/x_2^2 \end{bmatrix}$$

Jacobian matrix

Jacobian matrices

- For any function $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$, we can define the **Jacobian matrix** of all partial derivatives:

Columns are the *inputs* to f .

$$\frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_1}{\partial \mathbf{x}_m} \\ & \ddots & \\ \frac{\partial f_n}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_n}{\partial \mathbf{x}_m} \end{bmatrix}$$

Row are the *outputs* of f .

Chain rule of multivariate calculus

- Suppose $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ and $g : \mathbb{R}^k \rightarrow \mathbb{R}^m$, so that $f \circ g : \mathbb{R}^k \rightarrow \mathbb{R}^n$

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

Jacobian of f .

Chain rule of multivariate calculus

- Suppose $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ and $g : \mathbb{R}^k \rightarrow \mathbb{R}^m$, so that $f \circ g : \mathbb{R}^k \rightarrow \mathbb{R}^n$

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

Jacobian of g .

Chain rule of multivariate calculus

- Suppose $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ and $g : \mathbb{R}^k \rightarrow \mathbb{R}^m$, so that $f \circ g : \mathbb{R}^k \rightarrow \mathbb{R}^n$

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

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

Chain rule of multivariate calculus: example

- Suppose:

$$f \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) = \begin{bmatrix} x_1 - 2x_2 + x_3/4 \end{bmatrix} \quad g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} x_1 + 2x_2 \\ x_2 \\ -x_1 + 3 \end{bmatrix}$$

Chain rule of multivariate calculus: example

- Suppose:

$$f \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) = \begin{bmatrix} x_1 - 2x_2 + x_3/4 \end{bmatrix} \quad g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} x_1 + 2x_2 \\ x_2 \\ -x_1 + 3 \end{bmatrix}$$

- What is $\frac{\partial(f \circ g)}{\partial \mathbf{x}}$? Two alternative methods:

1. Substitute g into f and differentiate.

2. Apply chain rule.

Chain rule of multivariate calculus: example

$$f \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) = \begin{bmatrix} x_1 - 2x_2 + x_3/4 \end{bmatrix} \quad g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} x_1 + 2x_2 \\ x_2 \\ -x_1 + 3 \end{bmatrix}$$

- Substitution:

$$f \circ g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = f \left(g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) \right)$$

Chain rule of multivariate calculus: example

$$f \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) = \begin{bmatrix} x_1 - 2x_2 + x_3/4 \end{bmatrix} \quad g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} x_1 + 2x_2 \\ x_2 \\ -x_1 + 3 \end{bmatrix}$$

- Substitution:

$$f \circ g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = f \left(g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) \right) = x_1 + 2x_2 - 2(x_2) + (-x_1 + 3)/4$$

Chain rule of multivariate calculus: example

$$f \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) = \begin{bmatrix} x_1 - 2x_2 + x_3/4 \end{bmatrix} \quad g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} x_1 + 2x_2 \\ x_2 \\ -x_1 + 3 \end{bmatrix}$$

- Substitution:

$$\begin{aligned} f \circ g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) &= f \left(g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) \right) = x_1 + 2x_2 - 2(x_2) + (-x_1 + 3)/4 \\ &= \frac{3}{4}(x_1 + 1) \end{aligned}$$

Chain rule of multivariate calculus: example

$$f \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) = \begin{bmatrix} x_1 - 2x_2 + x_3/4 \end{bmatrix} \quad g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} x_1 + 2x_2 \\ x_2 \\ -x_1 + 3 \end{bmatrix}$$

- Substitution:

$$\begin{aligned} f \circ g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) &= f \left(g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) \right) = x_1 + 2x_2 - 2(x_2) + (-x_1 + 3)/4 \\ &= \frac{3}{4}(x_1 + 1) \\ \implies \frac{\partial(f \circ g)}{\partial \mathbf{x}} &= \begin{bmatrix} \frac{3}{4} & 0 \end{bmatrix} \end{aligned}$$

Chain rule of multivariate calculus: example

$$f \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) = \begin{bmatrix} x_1 - 2x_2 + x_3/4 \end{bmatrix} \quad g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} x_1 + 2x_2 \\ x_2 \\ -x_1 + 3 \end{bmatrix}$$

- Chain rule:

$$\frac{\partial f}{\partial g} = \begin{bmatrix} ? \end{bmatrix} \quad \mathbf{1 \times 3}$$

Chain rule of multivariate calculus: example

$$f \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) = \begin{bmatrix} x_1 - 2x_2 + x_3/4 \end{bmatrix} \quad g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} x_1 + 2x_2 \\ x_2 \\ -x_1 + 3 \end{bmatrix}$$

- Chain rule:

$$\frac{\partial f}{\partial g} = \begin{bmatrix} 1 & -2 & \frac{1}{4} \end{bmatrix} \quad \textcolor{blue}{1 \times 3}$$

Chain rule of multivariate calculus: example

$$f \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) = \begin{bmatrix} x_1 - 2x_2 + x_3/4 \end{bmatrix} \quad g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} x_1 + 2x_2 \\ x_2 \\ -x_1 + 3 \end{bmatrix}$$

- Chain rule:

$$\frac{\partial f}{\partial g} = \begin{bmatrix} 1 & -2 & \frac{1}{4} \end{bmatrix} \quad \text{1 x 3}$$

$$\frac{\partial g}{\partial \mathbf{x}} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ -1 & 0 \end{bmatrix} \quad \text{3 x 2}$$

Chain rule of multivariate calculus: example

$$f \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) = \begin{bmatrix} x_1 - 2x_2 + x_3/4 \end{bmatrix} \quad g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} x_1 + 2x_2 \\ x_2 \\ -x_1 + 3 \end{bmatrix}$$

- Chain rule:

$$\frac{\partial f}{\partial g} = \begin{bmatrix} 1 & -2 & \frac{1}{4} \end{bmatrix} \quad \text{1 x 3}$$

$$\frac{\partial g}{\partial \mathbf{x}} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ -1 & 0 \end{bmatrix} \quad \text{3 x 2}$$

$$\Rightarrow \frac{\partial f \circ g}{\partial \mathbf{x}} = \quad \text{1 x 2}$$

Chain rule of multivariate calculus: example

$$f \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) = \begin{bmatrix} x_1 - 2x_2 + x_3/4 \end{bmatrix} \quad g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} x_1 + 2x_2 \\ x_2 \\ -x_1 + 3 \end{bmatrix}$$

- Chain rule:

$$\begin{aligned} \frac{\partial f}{\partial g} &= \begin{bmatrix} 1 & -2 & \frac{1}{4} \end{bmatrix} \quad \text{1 x 3} \\ \frac{\partial g}{\partial \mathbf{x}} &= \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ -1 & 0 \end{bmatrix} \quad \text{3 x 2} \\ \implies \frac{\partial f \circ g}{\partial \mathbf{x}} &= \begin{bmatrix} 1 & -2 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ -1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} \frac{3}{4} & 0 \end{bmatrix} \end{aligned}$$

Chain rule of multivariate calculus

- The chain rule generalizes to arbitrarily deep compositions.
- Suppose $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$, $g : \mathbb{R}^k \rightarrow \mathbb{R}^m$, and $h : \mathbb{R}^l \rightarrow \mathbb{R}^k$.
- Then $\frac{\partial(f \circ g \circ h)}{\partial \mathbf{x}} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial h} \frac{\partial h}{\partial \mathbf{x}}$

$$f: R^m \rightarrow R:$$

Jacobian versus gradient

- Note, for a real-valued function $f : \mathbb{R}^m \rightarrow \mathbb{R}$, the Jacobian matrix is the transpose of the gradient.

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{x}_1} \\ \vdots \\ \frac{\partial f}{\partial \mathbf{x}_m} \end{bmatrix} \quad \frac{\partial f}{\partial \mathbf{x}} = \left[\begin{array}{ccc} \frac{\partial f}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f}{\partial \mathbf{x}_m} \end{array} \right]$$

Gradient **Jacobian**

Jacobian matrices

- Note that we sometimes examine the *same* function from different perspectives, e.g.:

$$f(\mathbf{x}; \mathbf{w}) = f(x, y; a, b) = 2ax + b/y$$

Jacobian matrices

- Note that we sometimes examine the *same* function from different perspectives, e.g.:

$$f(\mathbf{x}; \mathbf{w}) = f(x, y; a, b) = 2ax + b/y$$

- We can consider x, y to be the variables and a, b to be constants.

$$\frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} 2a & -b/y^2 \end{bmatrix}$$

Jacobian matrices

- Note that we sometimes examine the *same* function from different perspectives, e.g.:

$$f(\mathbf{x}; \mathbf{w}) = f(x, y; a, b) = 2ax + b/y$$

- We can consider a, b to be the variables and x, y to be constants.

$$\frac{\partial f}{\partial \mathbf{w}} = [\begin{array}{cc} 2x & 1/y \end{array}]$$

Vectorizing a matrix

- Sometimes we need to differentiate a *vector*-valued function f w.r.t. a *matrix* of its parameters, e.g.:

$$f(\mathbf{x}; \mathbf{W}) = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- But how? The Jacobian can only be 2D.

Vectorizing a matrix

- Sometimes we need to differentiate a *vector*-valued function f w.r.t. a *matrix* of its parameters, e.g.:

$$f(\mathbf{x}; \mathbf{W}) = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 \\ w_3x_1 + w_4x_2 \end{bmatrix}$$

- But how? The Jacobian can only be 2D.
- We could treat this as a 3-D tensor. However, in practice it's easier to **vectorize** matrix \mathbf{W} to be a vector and compute:

$$\frac{\partial f}{\partial \text{vec}[\mathbf{W}]}$$

Vectorizing a matrix

$$f(\mathbf{x}; \mathbf{W}) = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\frac{\partial f}{\partial \text{vec}[\mathbf{W}]} = \begin{bmatrix} \frac{\partial f_1}{w_1} & \frac{\partial f_1}{w_2} & \frac{\partial f_1}{w_3} & \frac{\partial f_1}{w_4} \\ \frac{\partial f_2}{w_1} & \frac{\partial f_2}{w_2} & \frac{\partial f_2}{w_3} & \frac{\partial f_2}{w_4} \end{bmatrix}$$

Vectorizing a matrix

$$f(\mathbf{x}; \mathbf{W}) = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\begin{aligned}\frac{\partial f}{\partial \text{vec}[\mathbf{W}]} &= \begin{bmatrix} \frac{\partial f_1}{w_1} & \frac{\partial f_1}{w_2} & \frac{\partial f_1}{w_3} & \frac{\partial f_1}{w_4} \\ \frac{\partial f_2}{w_1} & \frac{\partial f_2}{w_2} & \frac{\partial f_2}{w_3} & \frac{\partial f_2}{w_4} \end{bmatrix} \\ &= \begin{bmatrix} x_1 & x_2 & 0 & 0 \\ 0 & 0 & x_3 & x_4 \end{bmatrix}\end{aligned}$$

Matrix derivative of a vector-valued function

Suppose a function J depends on \mathbf{z} , where $\mathbf{z} = \mathbf{W}\mathbf{x}$, $\mathbf{x} \in \mathbb{R}^m$, and $\mathbf{W} \in \mathbb{R}^{n \times m}$. Then to compute the derivative of J w.r.t. \mathbf{W} , we can use the chain rule:

$$\frac{\partial J}{\partial \mathbf{W}} = \frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}}$$

!y

Matrix derivative of a vector-valued function

Suppose a function J depends on \mathbf{z} , where $\mathbf{z} = \mathbf{W}\mathbf{x}$, $\mathbf{x} \in \mathbb{R}^m$, and $\mathbf{W} \in \mathbb{R}^{n \times m}$. Then to compute the derivative of J w.r.t. \mathbf{W} , we can use the chain rule:

$$\frac{\partial J}{\partial \mathbf{W}} = \frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}}$$

But how do we compute $\frac{\partial \mathbf{z}}{\partial \mathbf{W}}$ when \mathbf{W} is a matrix? We can *vectorize* \mathbf{W} by concatenating all its elements into one long vector, i.e.,

$$\text{vec}[\mathbf{W}] = [W_{11} \ \dots W_{1m} \ \dots \ W_{n1} \ \dots \ W_{nm}]$$

Matrix derivative of a vector-valued function

Suppose a function J depends on \mathbf{z} , where $\mathbf{z} = \mathbf{W}\mathbf{x}$, $\mathbf{x} \in \mathbb{R}^m$, and $\mathbf{W} \in \mathbb{R}^{n \times m}$. Then to compute the derivative of J w.r.t. \mathbf{W} , we can use the chain rule:

$$\frac{\partial J}{\partial \mathbf{W}} = \frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}}$$

But how do we compute $\frac{\partial \mathbf{z}}{\partial \mathbf{W}}$ when \mathbf{W} is a matrix? We can *vectorize* \mathbf{W} by concatenating all its elements into one long vector, i.e.,

$$\text{vec}[\mathbf{W}] = [W_{11} \dots W_{1m} \dots W_{n1} \dots W_{nm}]$$

Then we can compute:

$$\frac{\partial J}{\partial \text{vec}[\mathbf{W}]} = \frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]}$$

Matrix derivative of a vector-valued function

Suppose a function J depends on \mathbf{z} , where $\mathbf{z} = \mathbf{W}\mathbf{x}$, $\mathbf{x} \in \mathbb{R}^m$, and $\mathbf{W} \in \mathbb{R}^{n \times m}$. Then to compute the derivative of J w.r.t. \mathbf{W} , we can use the chain rule:

$$\frac{\partial J}{\partial \mathbf{W}} = \frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}}$$

But how do we compute $\frac{\partial \mathbf{z}}{\partial \mathbf{W}}$ when \mathbf{W} is a matrix? We can *vectorize* \mathbf{W} by concatenating all its elements into one long vector, i.e.,

$$\text{vec}[\mathbf{W}] = [W_{11} \dots W_{1m} \dots W_{n1} \dots W_{nm}]$$

Then we can compute:

$$\frac{\partial J}{\partial \text{vec}[\mathbf{W}]} = \frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]}$$

After we obtain $\frac{\partial J}{\partial \text{vec}[\mathbf{W}]}$, we then *unvectorize* it to obtain $\nabla_{\mathbf{W}} J \in \mathbb{R}^{n \times m}$.

Matrix derivative of a vector-valued function

$$\frac{\partial J}{\partial \mathbf{z}} \doteq \mathbf{p}^\top \quad \text{where } \mathbf{p} \in \mathbb{R}^n$$

Matrix derivative of a vector-valued function

$$\frac{\partial J}{\partial \mathbf{z}} \doteq \mathbf{p}^\top$$
$$\frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]} = \begin{bmatrix} \frac{\partial z_1}{\partial W_{11}} & \cdots & \frac{\partial z_1}{\partial W_{1m}} & \cdots & \frac{\partial z_1}{\partial W_{n1}} & \cdots & \frac{\partial z_1}{\partial W_{nm}} \\ \vdots & & \ddots & & & \cdots & \\ \frac{\partial z_n}{\partial W_{11}} & \cdots & \frac{\partial z_n}{\partial W_{1m}} & \cdots & \frac{\partial z_n}{\partial W_{n1}} & \cdots & \frac{\partial z_n}{\partial W_{nm}} \end{bmatrix}$$

Matrix derivative of a vector-valued function

$$\frac{\partial J}{\partial \mathbf{z}} \doteq \mathbf{p}^\top$$
$$\frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]} = \begin{bmatrix} \frac{\partial z_1}{\partial W_{11}} & \cdots & \frac{\partial z_1}{\partial W_{1m}} & \cdots & \frac{\partial z_1}{\partial W_{n1}} & \cdots & \frac{\partial z_1}{\partial W_{nm}} \\ \vdots & & \ddots & & \ddots & & \vdots \\ \frac{\partial z_n}{\partial W_{11}} & \cdots & \frac{\partial z_n}{\partial W_{1m}} & \cdots & \frac{\partial z_n}{\partial W_{n1}} & \cdots & \frac{\partial z_n}{\partial W_{nm}} \end{bmatrix}$$

How does the *first element* of \mathbf{z} depend on the *first row* of \mathbf{W} ?

$$\begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} W_{11} & \dots & W_{1m} \\ \vdots & & \vdots \\ W_{n1} & \dots & W_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

Matrix derivative of a vector-valued function

$$\frac{\partial J}{\partial \mathbf{z}} \doteq \mathbf{p}^\top$$
$$\frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]} = \begin{bmatrix} \frac{\partial z_1}{\partial W_{11}} & \cdots & \frac{\partial z_1}{\partial W_{1m}} & \cdots & \frac{\partial z_1}{\partial W_{n1}} & \cdots & \frac{\partial z_1}{\partial W_{nm}} \\ \vdots & & \ddots & & & & \cdots \\ \frac{\partial z_n}{\partial W_{11}} & \cdots & \frac{\partial z_n}{\partial W_{1m}} & \cdots & \frac{\partial z_n}{\partial W_{n1}} & \cdots & \frac{\partial z_n}{\partial W_{nm}} \end{bmatrix}$$

How does the *first element* of \mathbf{z} depend on the *last row* of \mathbf{W} ?

$$\begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} W_{11} & \dots & W_{1m} \\ & \vdots & \\ W_{n1} & \dots & W_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

Matrix derivative of a vector-valued function

$$\frac{\partial J}{\partial \mathbf{z}} \doteq \mathbf{p}^\top$$
$$\frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]} = \begin{bmatrix} \frac{\partial z_1}{\partial W_{11}} & \cdots & \frac{\partial z_1}{\partial W_{1m}} & \cdots & \frac{\partial z_1}{\partial W_{n1}} & \cdots & \frac{\partial z_1}{\partial W_{nm}} \\ \vdots & & \ddots & & \cdots & & \cdots \\ \frac{\partial z_n}{\partial W_{11}} & \cdots & \frac{\partial z_n}{\partial W_{1m}} & \cdots & \frac{\partial z_n}{\partial W_{n1}} & \cdots & \frac{\partial z_n}{\partial W_{nm}} \end{bmatrix}$$

How does the *last element* of \mathbf{z} depend on the *first row* of \mathbf{W} ?

$$\begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} W_{11} & \dots & W_{1m} \\ \vdots & & \vdots \\ W_{n1} & \dots & W_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

Matrix derivative of a vector-valued function

$$\frac{\partial J}{\partial \mathbf{z}} \doteq \mathbf{p}^\top$$
$$\frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]} = \begin{bmatrix} \frac{\partial z_1}{\partial W_{11}} & \cdots & \frac{\partial z_1}{\partial W_{1m}} & \cdots & \frac{\partial z_1}{\partial W_{n1}} & \cdots & \frac{\partial z_1}{\partial W_{nm}} \\ \vdots & & \ddots & & & & \\ \frac{\partial z_n}{\partial W_{11}} & \cdots & \frac{\partial z_n}{\partial W_{1m}} & \cdots & \frac{\partial z_n}{\partial W_{n1}} & \cdots & \frac{\partial z_n}{\partial W_{nm}} \end{bmatrix}$$

How does the *last element* of \mathbf{z} depend on the *last row* of \mathbf{W} ?

$$\begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} W_{11} & \dots & W_{1m} \\ & \vdots & \\ W_{n1} & \dots & W_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

Matrix derivative of a vector-valued function

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{z}} &\doteq \mathbf{p}^\top \\ \frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]} &= \left[\begin{array}{cccccc} \frac{\partial z_1}{\partial W_{11}} & \cdots & \frac{\partial z_1}{\partial W_{1m}} & \cdots & \frac{\partial z_1}{\partial W_{n1}} & \cdots & \frac{\partial z_1}{\partial W_{nm}} \\ \vdots & & \ddots & & \cdots & & \end{array} \right] \\ &= \left[\begin{array}{cccccc} x_1 & \dots & x_m & \dots & 0 & \dots & 0 \\ \vdots & & \ddots & & \dots & & \\ 0 & \dots & 0 & \dots & x_1 & \dots & x_m \end{array} \right]\end{aligned}$$

Matrix derivative of a vector-valued function

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{z}} &\doteq \mathbf{p}^\top \\ \frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]} &= \left[\begin{array}{cccccc} \frac{\partial z_1}{\partial W_{11}} & \cdots & \frac{\partial z_1}{\partial W_{1m}} & \cdots & \frac{\partial z_1}{\partial W_{n1}} & \cdots & \frac{\partial z_1}{\partial W_{nm}} \\ \vdots & & \ddots & & \cdots & & \end{array} \right] \\ &= \left[\begin{array}{cccccc} x_1 & \dots & x_m & \dots & 0 & \dots & 0 \\ \vdots & & \ddots & & \dots & & \end{array} \right] \\ &= \left[\begin{array}{ccc} \mathbf{x}^\top & \dots & \mathbf{0}^\top \\ \vdots & \ddots & \dots \\ \mathbf{0}^\top & \dots & \mathbf{x}^\top \end{array} \right]\end{aligned}$$

Matrix derivative of a vector-valued function

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{z}} &\doteq \mathbf{p}^\top \\ \frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]} &= \left[\begin{array}{cccccc} \frac{\partial z_1}{\partial W_{11}} & \cdots & \frac{\partial z_1}{\partial W_{1m}} & \cdots & \frac{\partial z_1}{\partial W_{n1}} & \cdots & \frac{\partial z_1}{\partial W_{nm}} \\ \vdots & & \ddots & & \cdots & & \end{array} \right] \\ &= \left[\begin{array}{cccccc} x_1 & \dots & x_m & \dots & 0 & \dots & 0 \\ \vdots & & \ddots & & \dots & & \end{array} \right] \\ &= \left[\begin{array}{ccc} \mathbf{x}^\top & \dots & \mathbf{0}^\top \\ \vdots & \ddots & \dots \\ \mathbf{0}^\top & \dots & \mathbf{x}^\top \end{array} \right] \\ \frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]} &= \mathbf{p}^\top \left[\begin{array}{ccc} \mathbf{x}^\top & \dots & \mathbf{0}^\top \\ \vdots & \ddots & \dots \\ \mathbf{0}^\top & \dots & \mathbf{x}^\top \end{array} \right]\end{aligned}$$

Matrix derivative of a vector-valued function

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{z}} &\doteq \mathbf{p}^\top \\ \frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]} &= \left[\begin{array}{cccccc} \frac{\partial z_1}{\partial W_{11}} & \cdots & \frac{\partial z_1}{\partial W_{1m}} & \cdots & \frac{\partial z_1}{\partial W_{n1}} & \cdots & \frac{\partial z_1}{\partial W_{nm}} \\ \vdots & & \ddots & & \cdots & & \end{array} \right] \\ &= \left[\begin{array}{cccccc} x_1 & \dots & x_m & \dots & 0 & \dots & 0 \\ \vdots & & \ddots & & \dots & & \end{array} \right] \\ &= \left[\begin{array}{ccc} \mathbf{x}^\top & \dots & \mathbf{0}^\top \\ \vdots & \ddots & \dots \\ \mathbf{0}^\top & \dots & \mathbf{x}^\top \end{array} \right] \\ \frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]} &= \mathbf{p}^\top \left[\begin{array}{ccc} \mathbf{x}^\top & \dots & \mathbf{0}^\top \\ \vdots & \ddots & \dots \\ \mathbf{0}^\top & \dots & \mathbf{x}^\top \end{array} \right] \\ &= [p_1 \mathbf{x}^\top \dots p_n \mathbf{x}^\top]\end{aligned}$$

Matrix derivative of a vector-valued function

Now we unvectorize the result:

$$\nabla_{\mathbf{W}} J \doteq \frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]} = \begin{bmatrix} p_1 \mathbf{x}^\top \\ \vdots \\ p_n \mathbf{x}^\top \end{bmatrix}$$

Matrix derivative of a vector-valued function

Now we unvectorize the result:

$$\begin{aligned}\nabla_{\mathbf{W}} J \doteq \frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \text{vec}[\mathbf{W}]} &= \begin{bmatrix} p_1 \mathbf{x}^\top \\ \vdots \\ p_n \mathbf{x}^\top \end{bmatrix} \\ &= \mathbf{p} \mathbf{x}^\top\end{aligned}$$

Latent variable models

Latent variable models

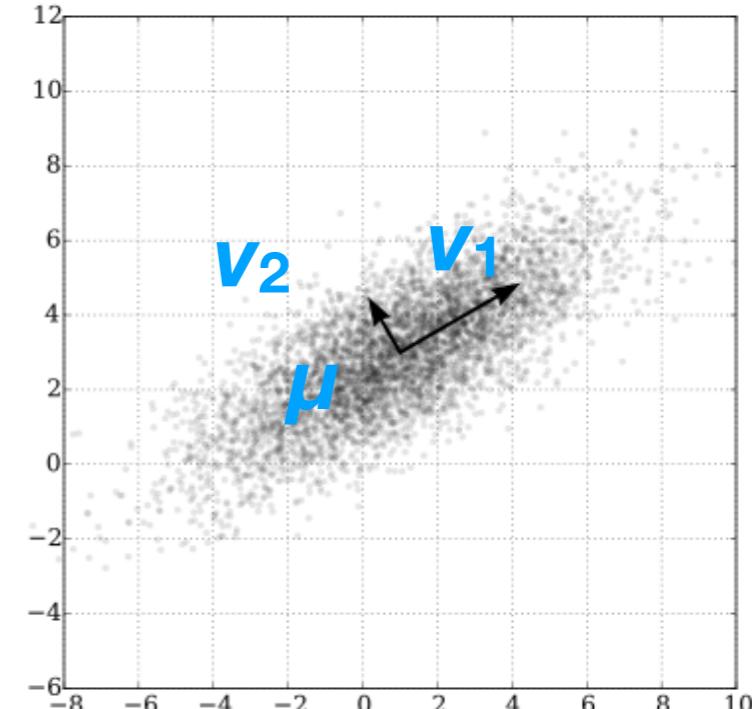
- Latent variable models (LVM) are **unsupervised** ML models.
- They posit that the observed data $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^n$ can be explained by a set of low-dimensional **latent** (unobserved) variables $\{\mathbf{h}^{(i)}\}_{i=1}^n$.
- Examples:
 - MNIST:
 - GENKI4K:

Latent variable models

- Latent variable models (LVM) are **unsupervised** ML models.
- They posit that the observed data $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^n$ can be explained by a set of low-dimensional **latent** (unobserved) variables $\{\mathbf{h}^{(i)}\}_{i=1}^n$.
- Examples:
 - MNIST: each image is “generated” by the digit (0-9) and a thickness (0-5) latent “code”.
 - GENKI4K: each face image is “generated” by the gender (0-1) and smile intensity (0-1).

Latent variable models

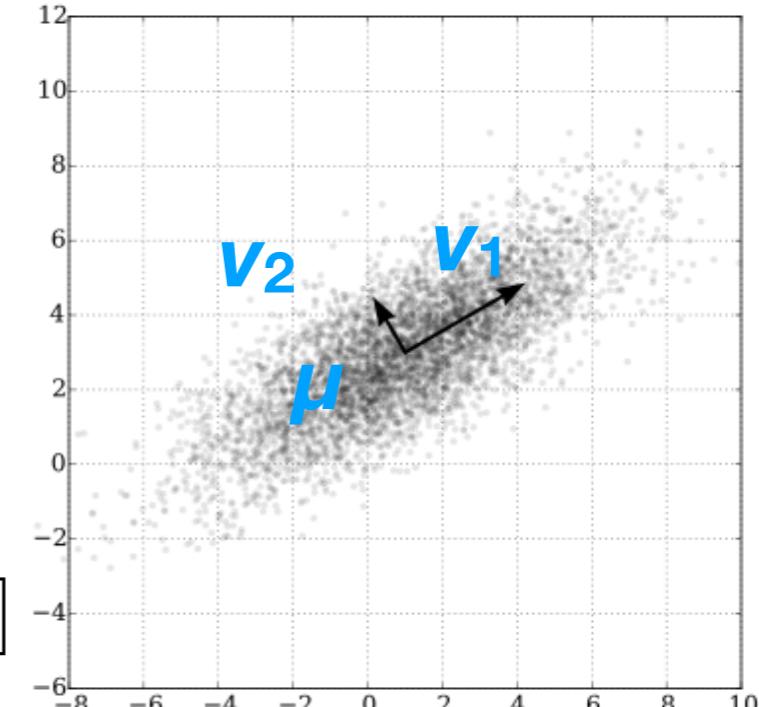
- Arguably the simplest LVM is **principal component analysis** (PCA).
- Recall that, for a dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^n$ with mean $\boldsymbol{\mu}$, PCA defines the set of k principal directions along which the variance of the projected data is maximized:



Latent variable models

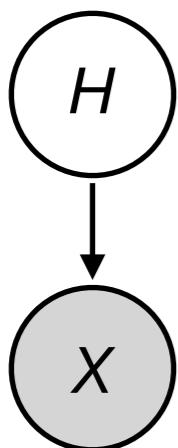
- Arguably the simplest LVM is **principal component analysis** (PCA).
- Recall that, for a dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^n$ with mean μ , PCA defines the set of k principal directions along which the variance of the projected data is maximized:
- Vector $\mathbf{v}^{(k)}$ has the same direction as the eigenvector of the auto-covariance matrix of \mathcal{D} with the k th largest associated eigenvalue.

$$\text{Cov}(\mathbf{X}, \mathbf{X}) = \mathbb{E}[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^\top]$$



Principal component analysis (PCA)

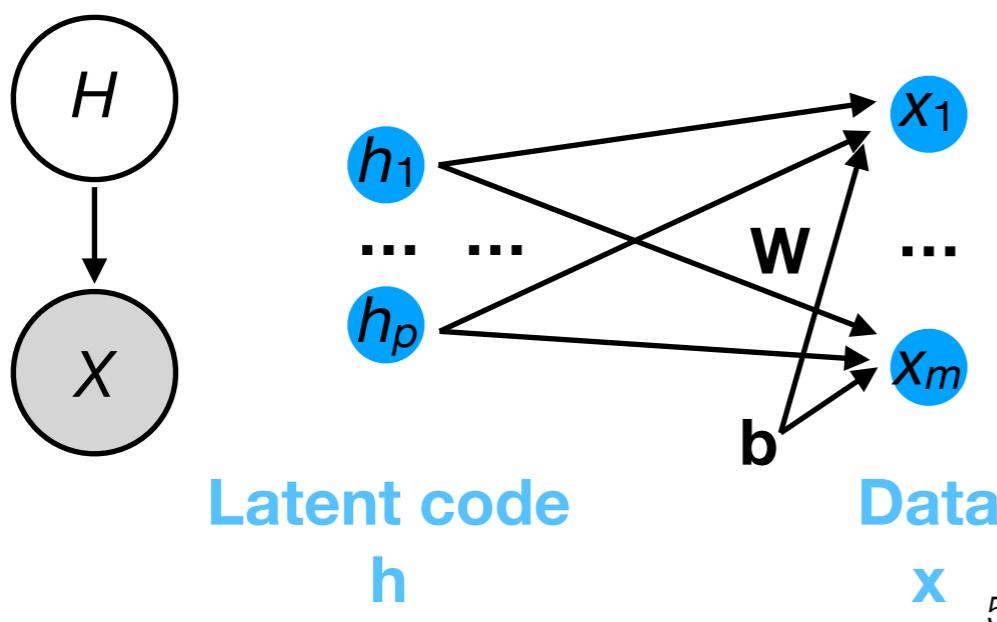
- However, PCA also has a **probabilistic** interpretation as an LVM.
- Let us assume that every data point $\mathbf{x} \in \mathbb{R}^m$ is actually generated from a lower-dimensional latent variable (or **code**) $\mathbf{h} \in \mathbb{R}^p$, where $p < m$.



Principal component analysis (PCA)

- In particular, we assume that each \mathbf{x} is approximately linear in \mathbf{h} , i.e.:

$$\mathbf{x} \approx \mathbf{W}\mathbf{h} + \mathbf{b}$$



Principal component analysis (PCA)

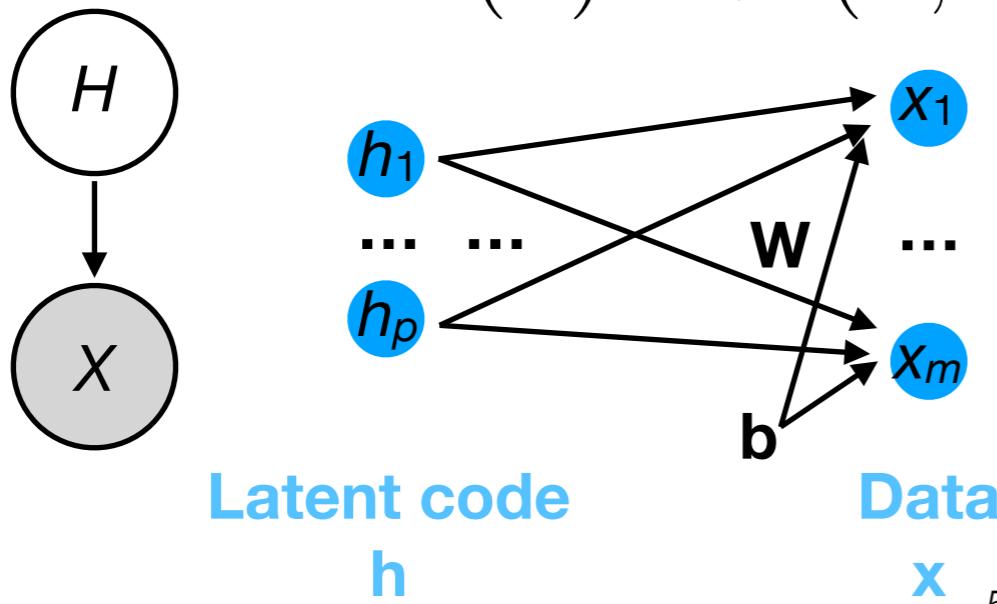
- In particular, we assume that each \mathbf{x} is approximately linear in \mathbf{h} , i.e.:

$$\mathbf{x} \approx \mathbf{W}\mathbf{h} + \mathbf{b}$$

- We can represent this probabilistically as:

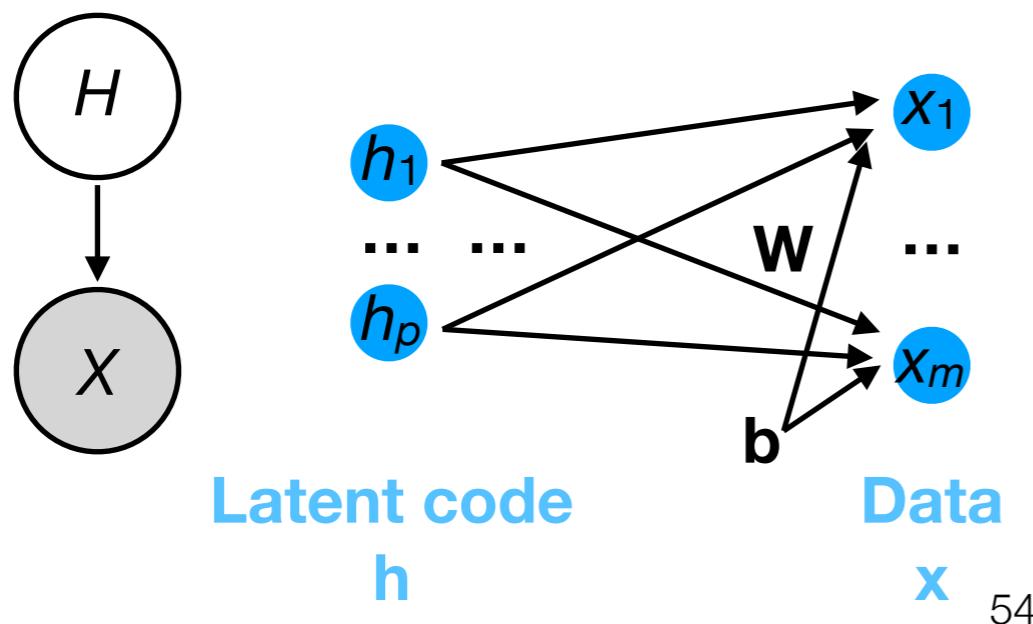
$$P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b}) = \mathcal{N}(\mathbf{x}; \mathbf{W}\mathbf{h} + \mathbf{b}, \sigma^2 \mathbf{I})$$

$$P(\mathbf{h}) = \mathcal{N}(\mathbf{h}; \mathbf{0}, \mathbf{I})$$



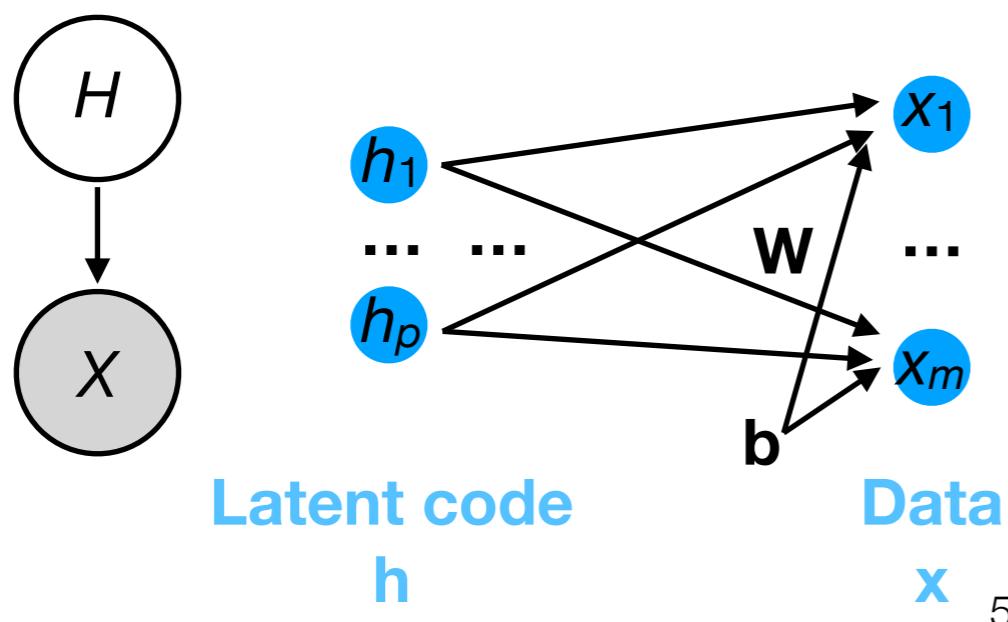
Principal component analysis (PCA)

- Given a set of training data \mathcal{D} , we can optimize \mathbf{W} and \mathbf{b} using maximum-likelihood estimation (MLE).



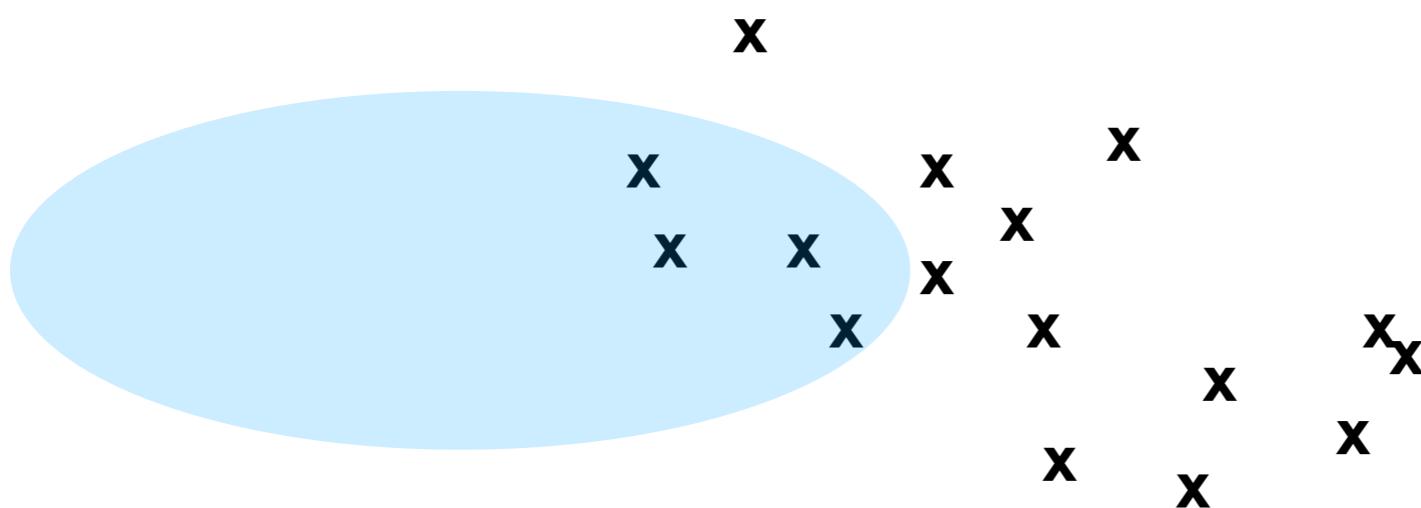
Principal component analysis (PCA)

- The MLE for \mathbf{b} is the mean of the data $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^n$.
- The MLE for \mathbf{W} is the same set of principal eigenvectors from standard PCA (but weighted by their eigenvalues).
- The weights ensure that $P(\mathbf{h})$ is 0-mean with \mathbf{I} -covariance.



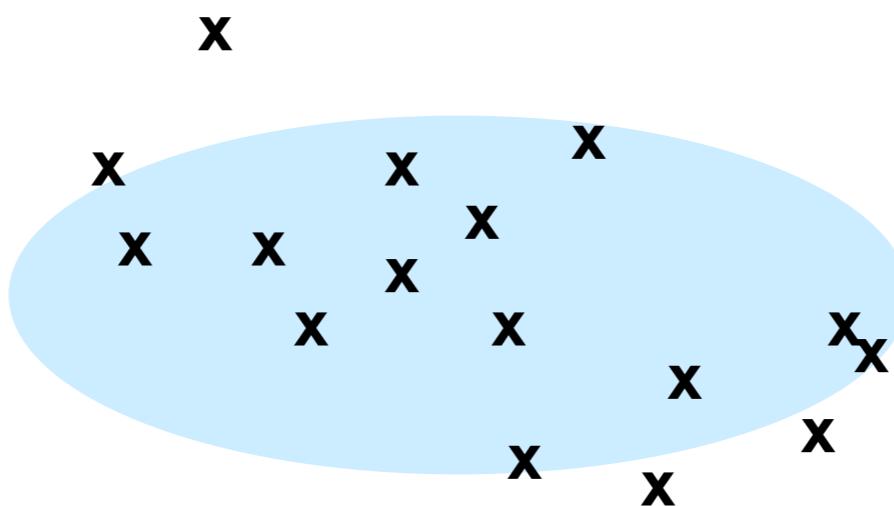
MLE for a multivariate Gaussian distribution

- Suppose we have the following dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^n$.
- Which Gaussian (with mean μ and covariance Σ) fits these data better?



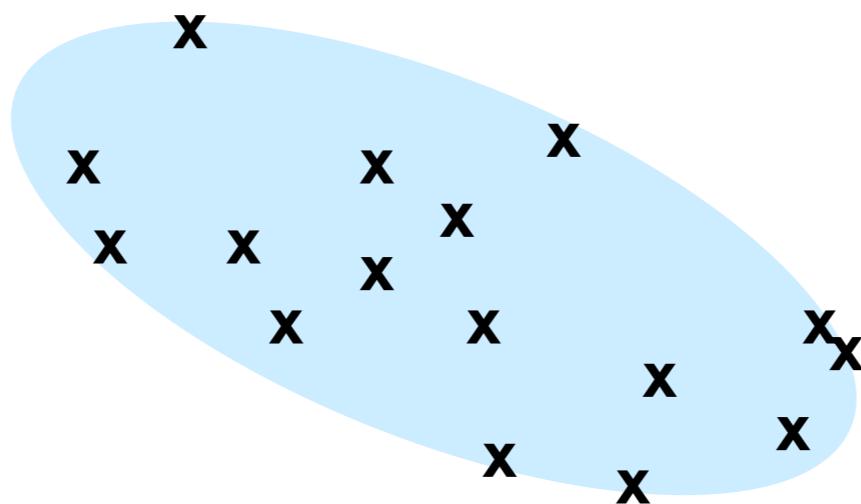
MLE for a multivariate Gaussian distribution

- Suppose we have the following dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^n$.
- Which Gaussian (with mean μ and covariance Σ) fits these data better?



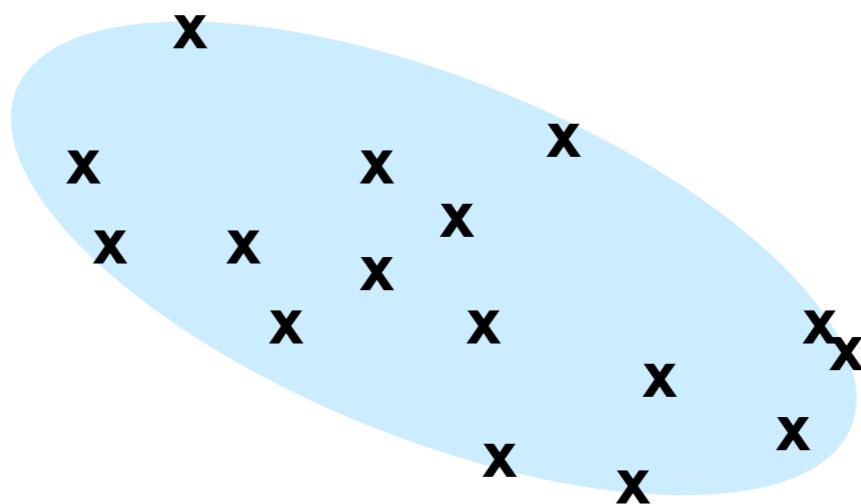
MLE for a multivariate Gaussian distribution

- Suppose we have the following dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^n$.
- Which Gaussian (with mean μ and covariance Σ) fits these data better?



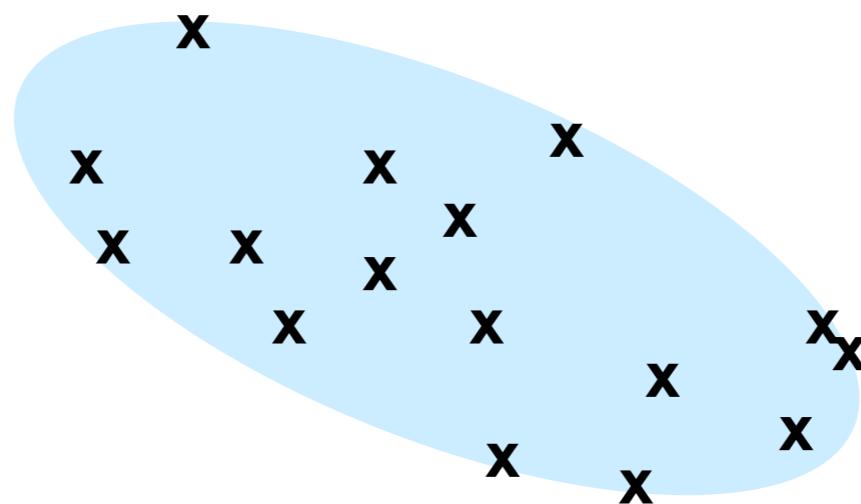
MLE for a multivariate Gaussian distribution

- For each data point \mathbf{x} , we can calculate the Gaussian likelihood:
$$\frac{1}{\sqrt{(2\pi)^m \det \Sigma}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$$
- Due to conditional independence, the joint distribution $P(\mathcal{D} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ factors as the product over all n points.



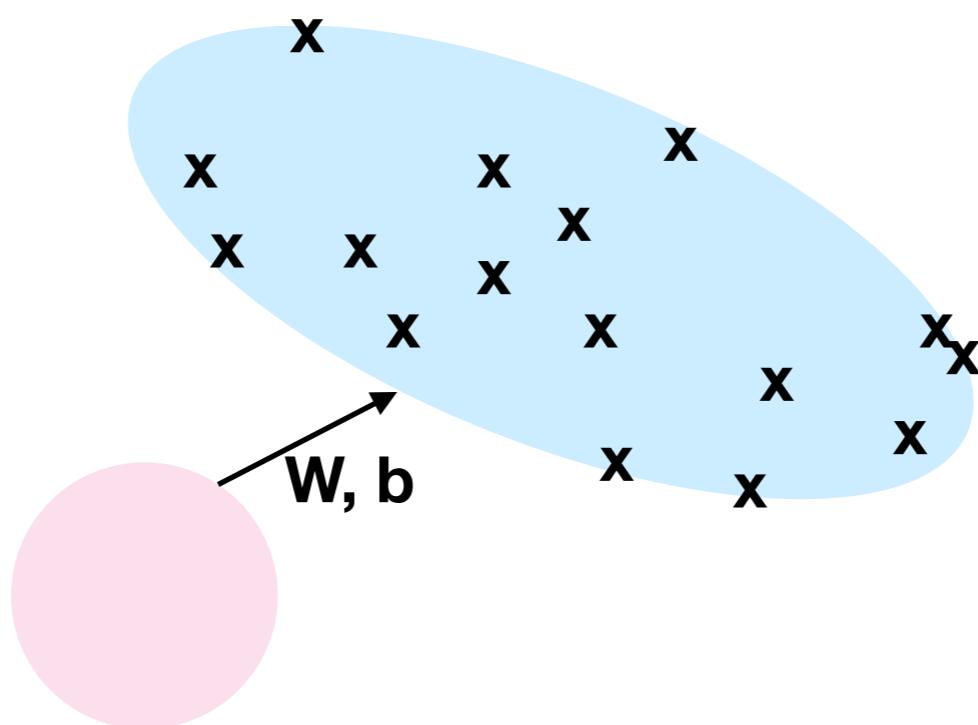
MLE for a multivariate Gaussian distribution

- We can use differential calculus to maximize the (log) likelihood w.r.t. the parameters μ and Σ .



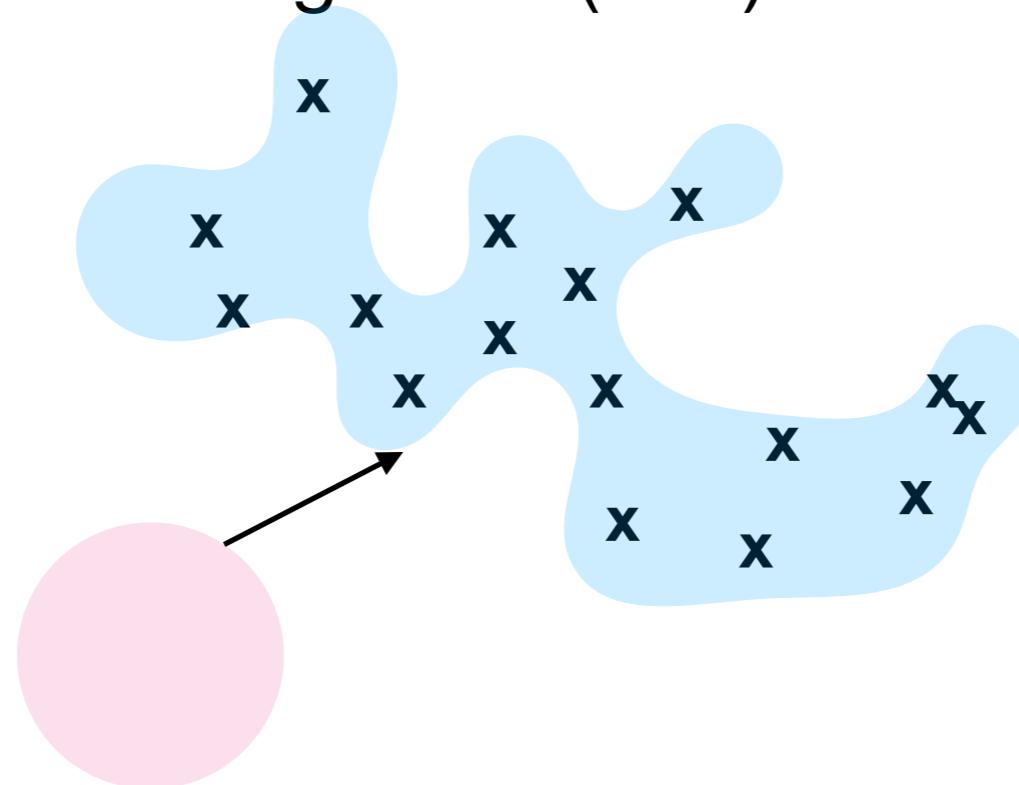
MLE for a multivariate Gaussian distribution

- With PCA as an LVM, we instead identify the matrix \mathbf{W} and vector \mathbf{b} that transforms a **low-dimensional**, 0-mean, and \mathbf{I} -covariance Gaussian into a **high-dimensional** Gaussian that fits the data \mathcal{D} .



MLE for a multivariate Gaussian distribution

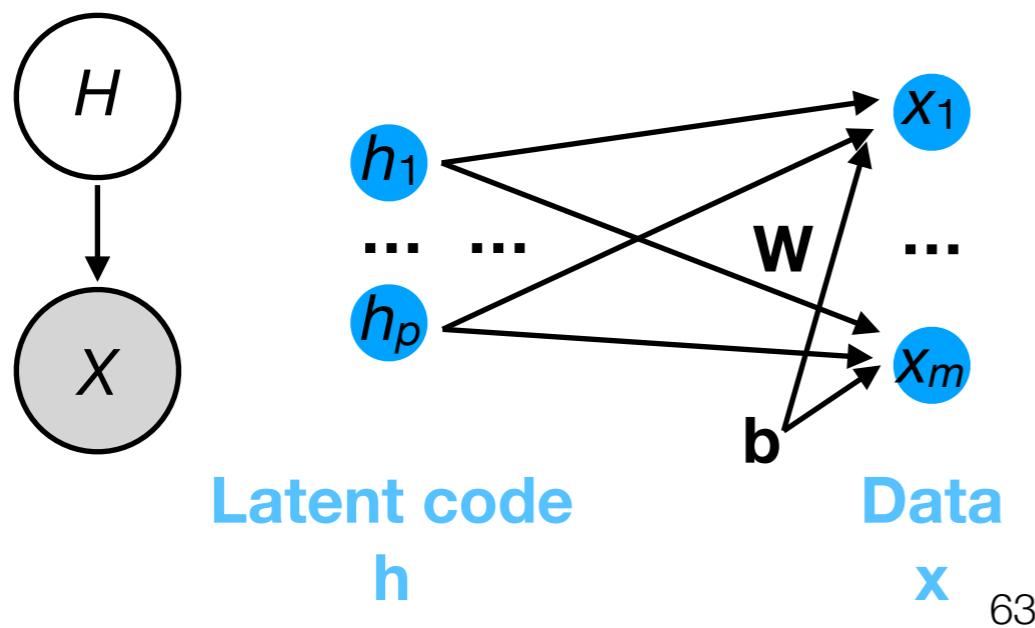
- With PCA, this transformation is linear, which limits the representational power of the LVM.
- More powerful LVMs include variational auto-encoders (VAEs) and normalizing flows (NFs).



Principal component analysis (PCA)

- This probabilistic PCA model allows us to **generate** novel examples:
 1. Sample from the prior distribution over \mathbf{h} .

$$P(\mathbf{h}) = \mathcal{N}(\mathbf{h}; \mathbf{0}, \mathbf{I})$$



Principal component analysis (PCA)

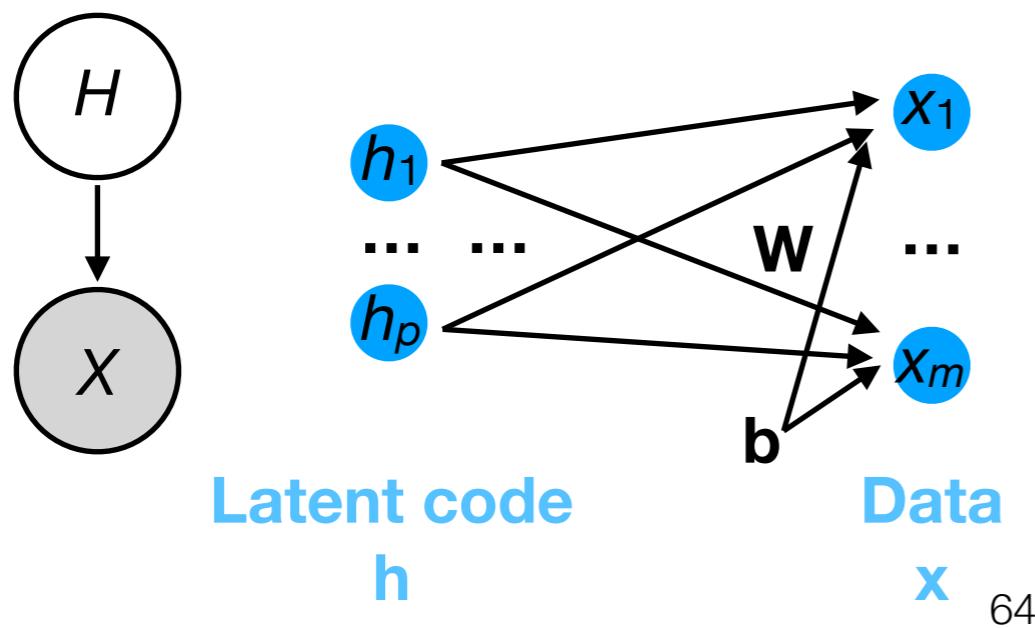
- This probabilistic PCA model allows us to **generate** novel examples:

1. Sample from the prior distribution over \mathbf{h} .

$$P(\mathbf{h}) = \mathcal{N}(\mathbf{h}; \mathbf{0}, \mathbf{I})$$

2. Sample from the conditional distribution of $\mathbf{x} | \mathbf{h}$.

$$P(\mathbf{x} | \mathbf{h}, \mathbf{W}, \mathbf{b}) = \mathcal{N}(\mathbf{x}; \mathbf{Wh} + \mathbf{b}, \sigma^2 \mathbf{I})$$



Principal component analysis (PCA)

- Demo of probabilistic PCA on MNIST.
- Demo of probabilistic PCA on GENKI4K.

Principal component analysis (PCA)

- Demo of probabilistic PCA on MNIST.
- Demo of probabilistic PCA on GENKI4K.
- Overall, the results are not very good.
 - Problem: the generative model is **linear** — easy to train and sample, but too simplistic.
 - Later on, we will see more powerful deep latent variable models (variational auto-encoders (VAE)) and other generative neural networks (generative-adversarial networks (GAN)).

Principal component analysis (PCA)

- What if you want to know the latent code \mathbf{h} for a given \mathbf{x} , i.e., $P(\mathbf{h} \mid \mathbf{x})$? We can use Bayes' rule:

$$P(\mathbf{h} \mid \mathbf{x}, \mathbf{W}, \mathbf{b}) = \frac{P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h} \mid \mathbf{W}, \mathbf{b})}{P(\mathbf{x} \mid \mathbf{W}, \mathbf{b})}$$

Principal component analysis (PCA)

- What if you want to know the latent code \mathbf{h} for a given \mathbf{x} , i.e., $P(\mathbf{h} \mid \mathbf{x})$? We can use Bayes' rule:

$$\begin{aligned} P(\mathbf{h} \mid \mathbf{x}, \mathbf{W}, \mathbf{b}) &= \frac{P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h} \mid \mathbf{W}, \mathbf{b})}{P(\mathbf{x} \mid \mathbf{W}, \mathbf{b})} \\ &= \frac{P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h} \mid \mathbf{W}, \mathbf{b})}{\int d\mathbf{h}P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h})} \\ &= \frac{1}{Z(\mathbf{x})}P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h} \mid \mathbf{W}, \mathbf{b}) \end{aligned}$$

Principal component analysis (PCA)

- What if you want to know the latent code \mathbf{h} for a given \mathbf{x} , i.e., $P(\mathbf{h} \mid \mathbf{x})$? We can use Bayes' rule:

$$\begin{aligned} P(\mathbf{h} \mid \mathbf{x}, \mathbf{W}, \mathbf{b}) &= \frac{P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h} \mid \mathbf{W}, \mathbf{b})}{P(\mathbf{x} \mid \mathbf{W}, \mathbf{b})} \\ &= \frac{P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h} \mid \mathbf{W}, \mathbf{b})}{\int d\mathbf{h} P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h})} \\ &= \frac{1}{Z(\mathbf{x})} P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h} \mid \mathbf{W}, \mathbf{b}) \end{aligned}$$

Here, Z is called the normalization constant.

Principal component analysis (PCA)

- In general, the integral in Z is **intractable**. However, for PCA it is a Gaussian and can be computed in closed form.

$$\begin{aligned} P(\mathbf{h} \mid \mathbf{x}, \mathbf{W}, \mathbf{b}) &= \frac{P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h} \mid \mathbf{W}, \mathbf{b})}{P(\mathbf{x} \mid \mathbf{W}, \mathbf{b})} \\ &= \frac{P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h} \mid \mathbf{W}, \mathbf{b})}{\int d\mathbf{h}P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h})} \\ &= \frac{1}{Z(\mathbf{x})}P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h} \mid \mathbf{W}, \mathbf{b}) \end{aligned}$$

Here, Z is called the normalization constant.

Principal component analysis (PCA)

- For more complex LVMs, we can use techniques such as **variational inference** to compute Z approximately.

$$\begin{aligned} P(\mathbf{h} \mid \mathbf{x}, \mathbf{W}, \mathbf{b}) &= \frac{P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h} \mid \mathbf{W}, \mathbf{b})}{P(\mathbf{x} \mid \mathbf{W}, \mathbf{b})} \\ &= \frac{P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h} \mid \mathbf{W}, \mathbf{b})}{\int d\mathbf{h}P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h})} \\ &= \frac{1}{Z(\mathbf{x})}P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b})P(\mathbf{h} \mid \mathbf{W}, \mathbf{b}) \end{aligned}$$

Here, Z is called the normalization constant.

≥3-layer NNs

Limitations of linear NNs

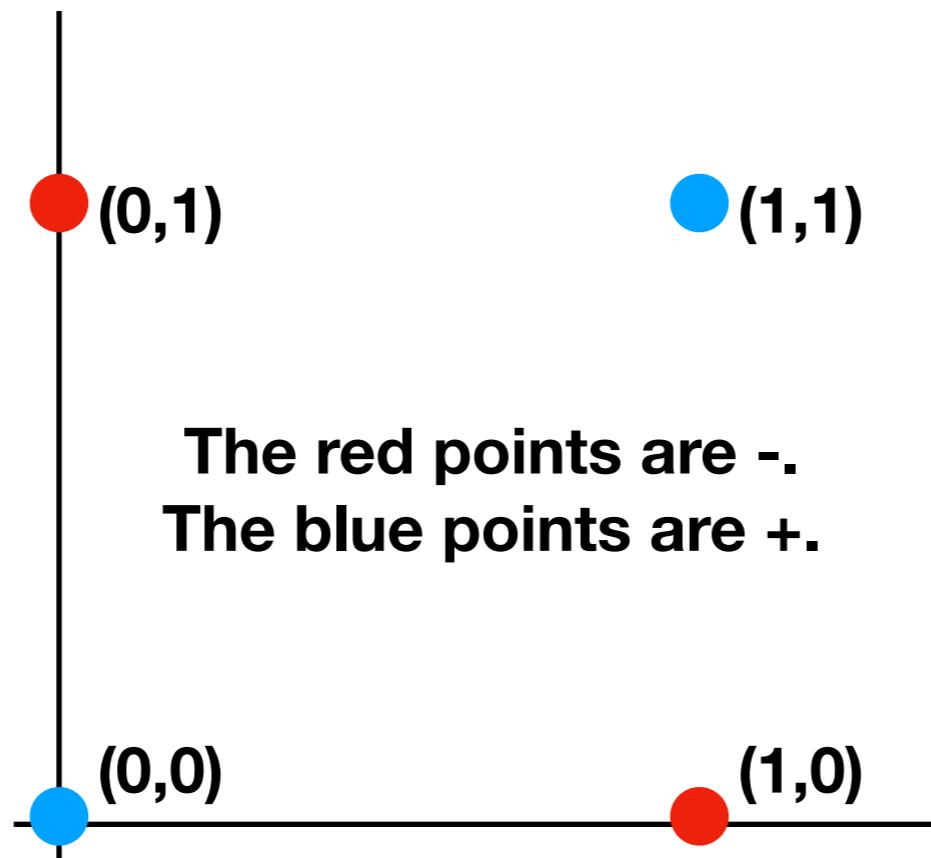
- Linear 2-layer NNs are very convenient to optimize.
- But they are very limited in the functions they can represent.

Limitations of linear NNs

- As illustrated in the homework, no linear NN can solve the XOR problem with accuracy better than chance.
- In order to solve this problem (and many other, more important ones), we need to **transform** the input space into a more useful feature space.

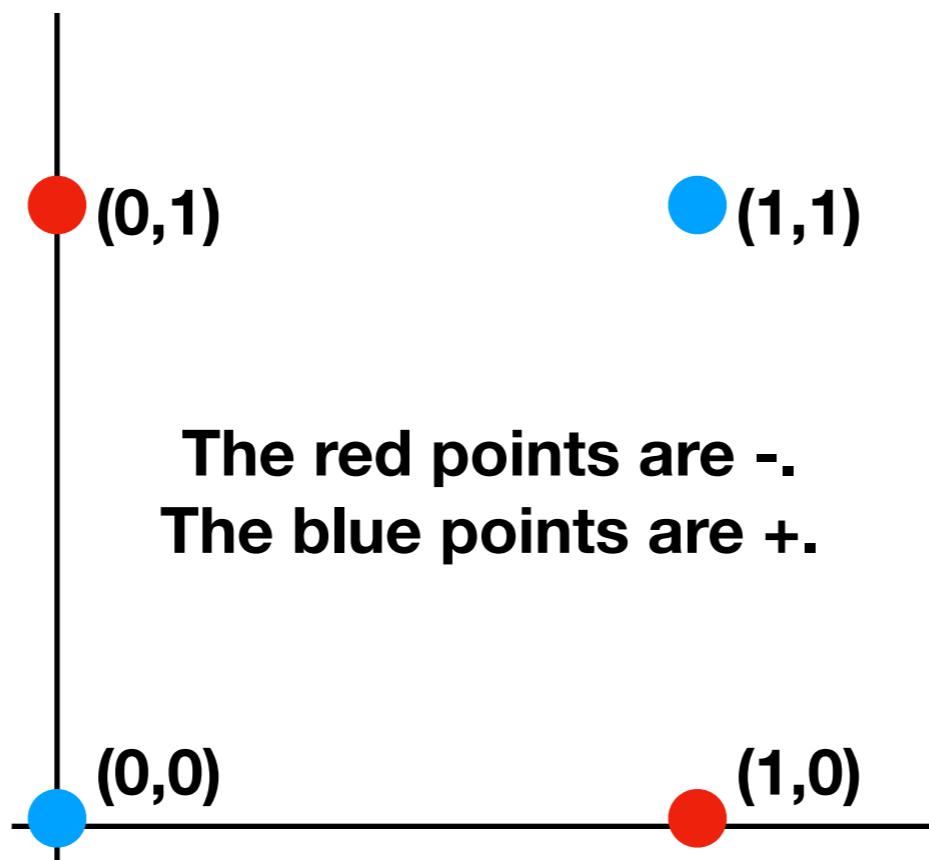
XOR problem

- Recall that no linear decision boundary (e.g., linear SVM, linear regression) can solve the XOR problem.



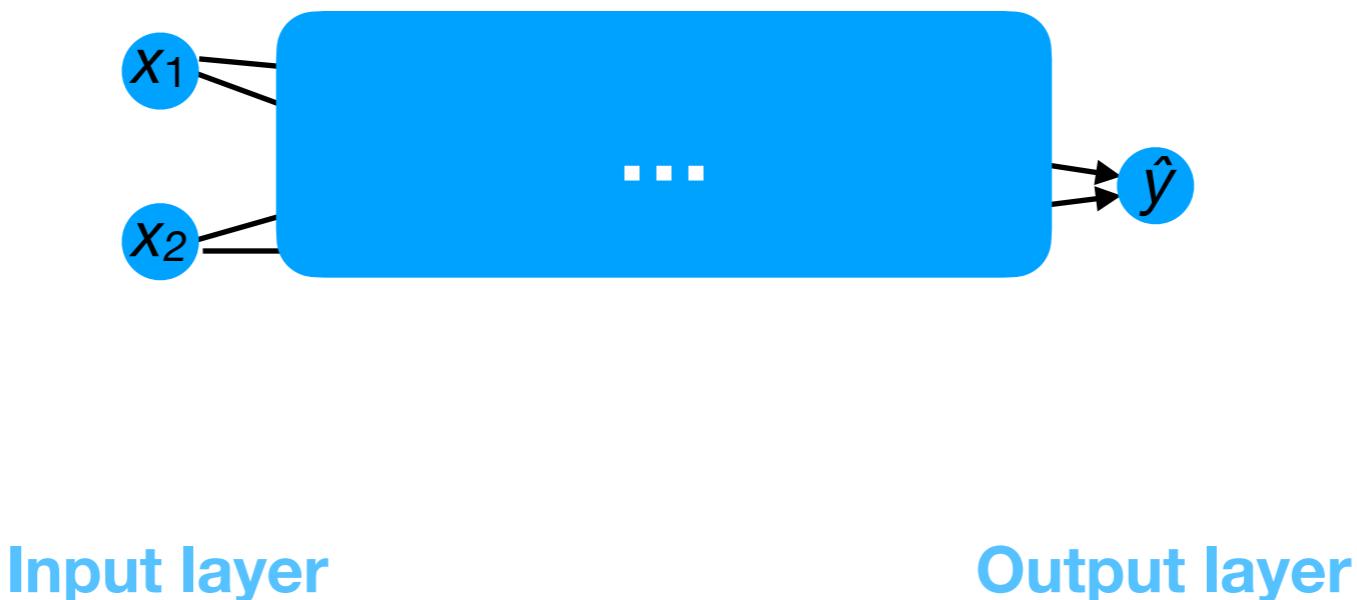
XOR problem

- Let's see how a 3-layer NN can solve this problem...



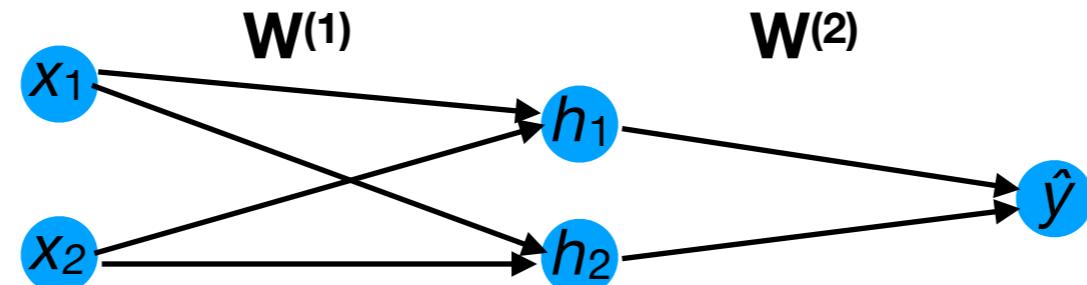
XOR problem

- We want to use a NN to define a function g such that:
 - $g(0,0) = 0$
 - $g(0,1) = 1$
 - $g(1,0) = 1$
 - $g(1,1) = 0$



XOR problem

- We will use a 3-layer NN:
 - 1 input layer
 - 1 hidden layer
 - 1 output layer



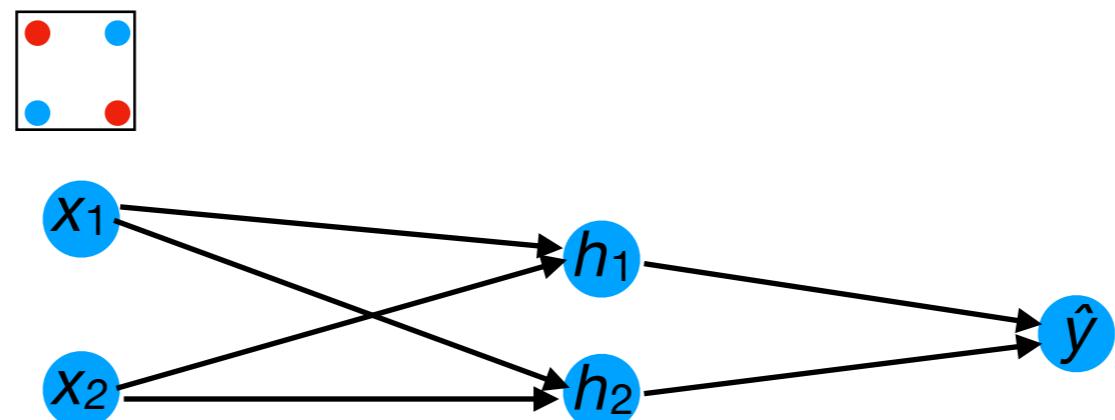
Input layer

Hidden layer

Output layer

XOR problem

- Here's how a 3-layer NN can solve it:



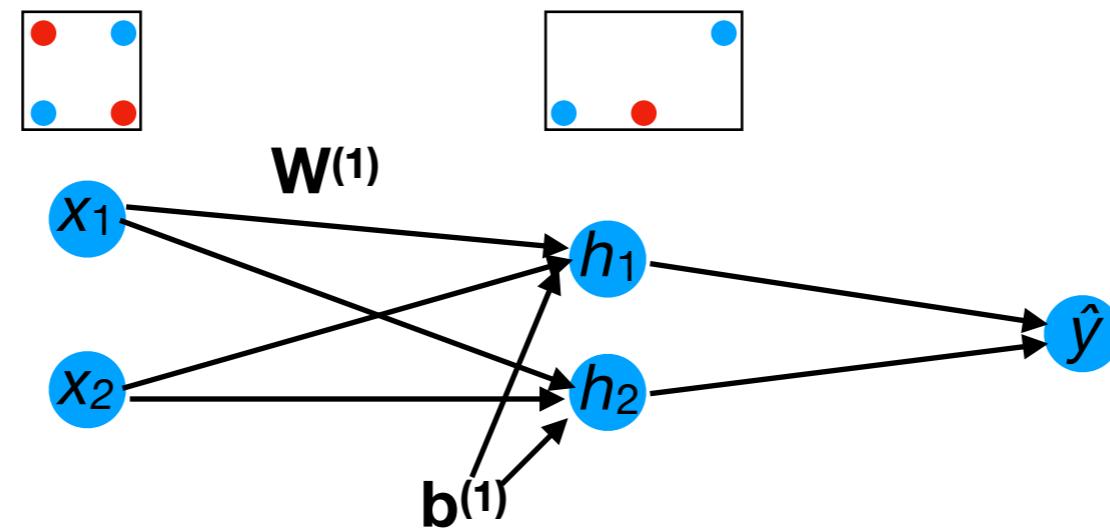
Input layer

Hidden layer

Output layer

XOR problem

- Here's how a 3-layer NN can solve it:
 - $\mathbf{W}^{(1)}, \mathbf{b}^{(1)}$ will “collapse” the two negative data points onto one point in the “hidden” 2-D space.



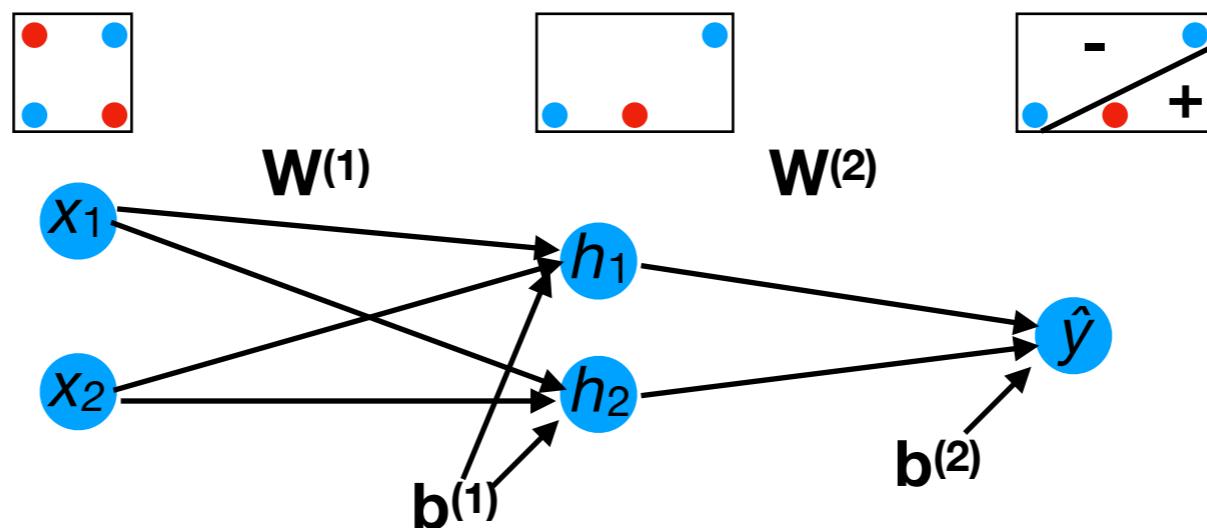
Input layer

Hidden layer

Output layer

XOR problem

- Here's how a 3-layer NN can solve it:
 - $\mathbf{W}^{(1)}, \mathbf{b}^{(1)}$ will “collapse” the two negative data points onto one point in the “hidden” 2-D space.
 - Since the + and - data are now linearly separated, $\mathbf{W}^{(2)}, \mathbf{b}^{(2)}$ can easily distinguish the two classes.



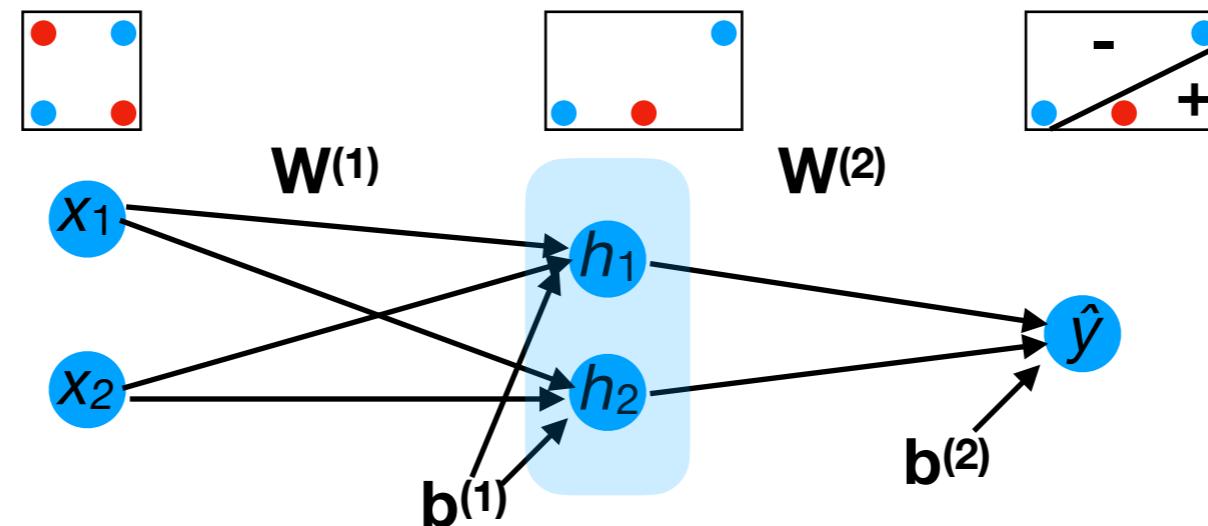
Input layer

Hidden layer

Output layer

XOR problem

- To achieve the collapse, we apply a **non-linear activation function** in the hidden layer.



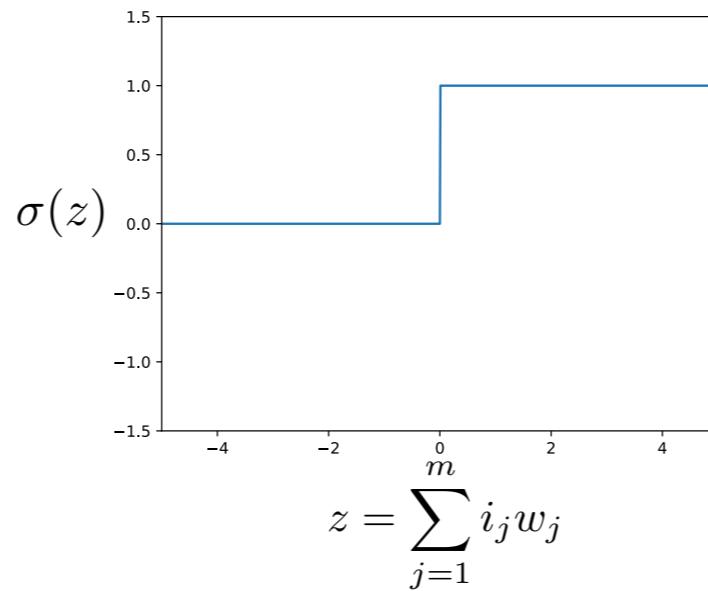
Input layer

Hidden layer

Output layer

Activation functions

- One of the oldest non-linear activation functions, σ , is the **Heaviside step function**, which was used in the original Perceptron neural network (Rosenblatt 1957).

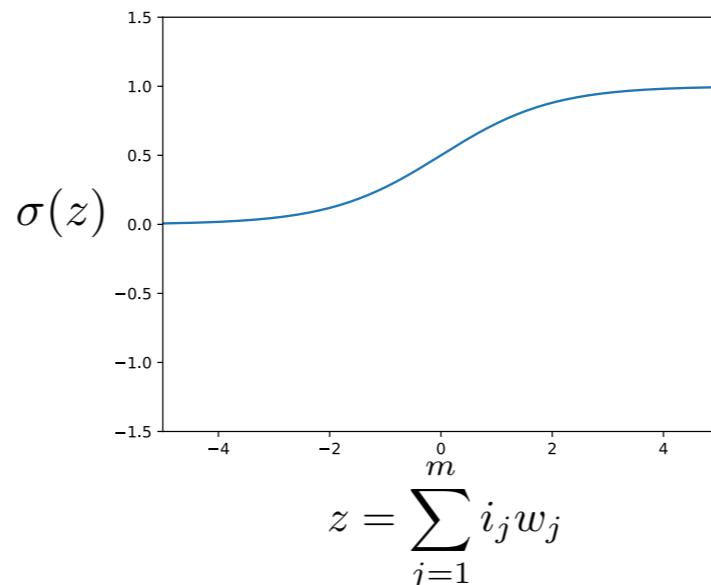


This function directly models the all-or-none firing rule of biological neural networks.

Activation functions

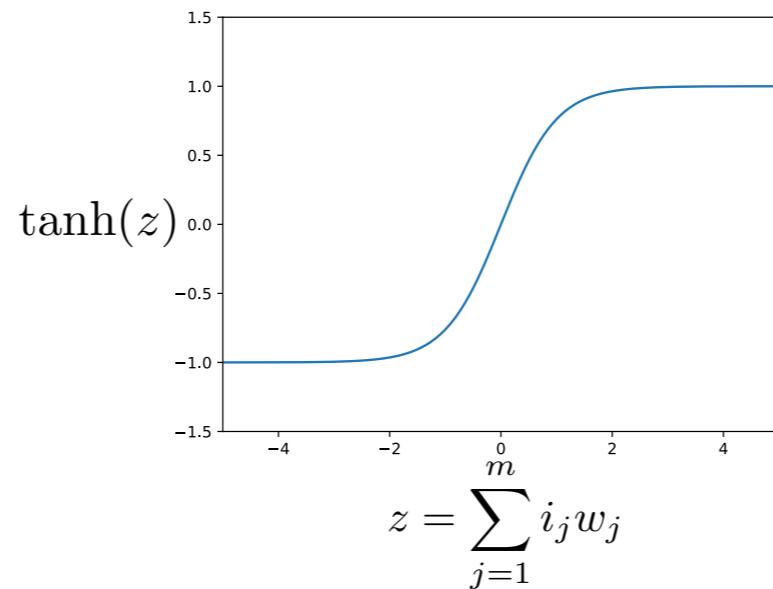
- Because the Heaviside step function is non-differentiable at 0, it was largely replaced by either a **logistic sigmoid**:

$$\sigma(z) = \frac{1}{1 + \exp -z}$$



Activation functions

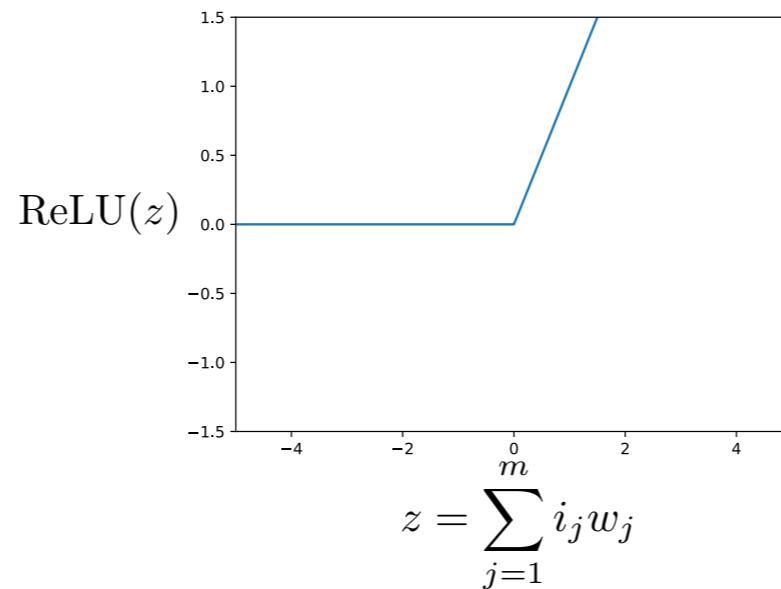
- ...or sometimes with hyperbolic tangent:



Activation functions

- Since 2012, most modern NNs have used a **rectified linear unit** as the activation function, known as **ReLU**:

$$\text{ReLU}(z) = \max\{0, z\}$$

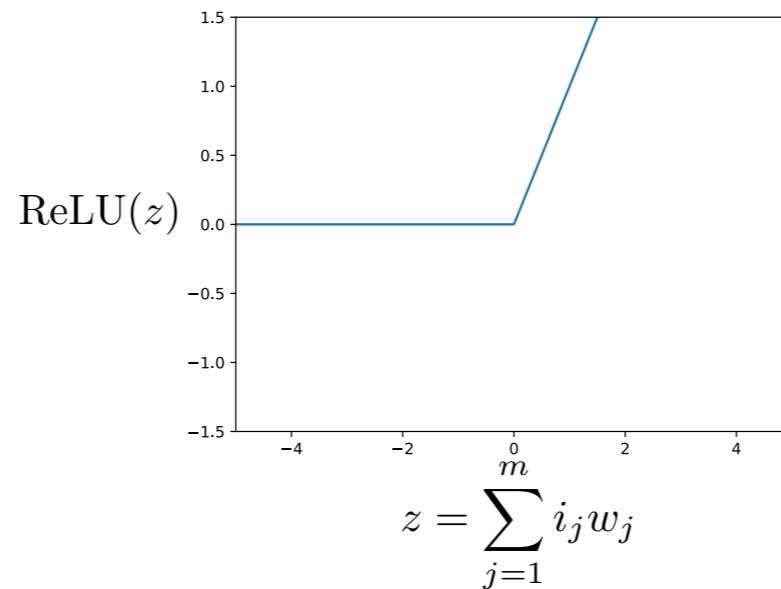


Compared to sigmoids, ReLU's gradient is **unattenuated** for half of the real line.

Activation functions

- Since 2012, most modern NNs have used a **rectified linear unit** as the activation function, known as **ReLU**:

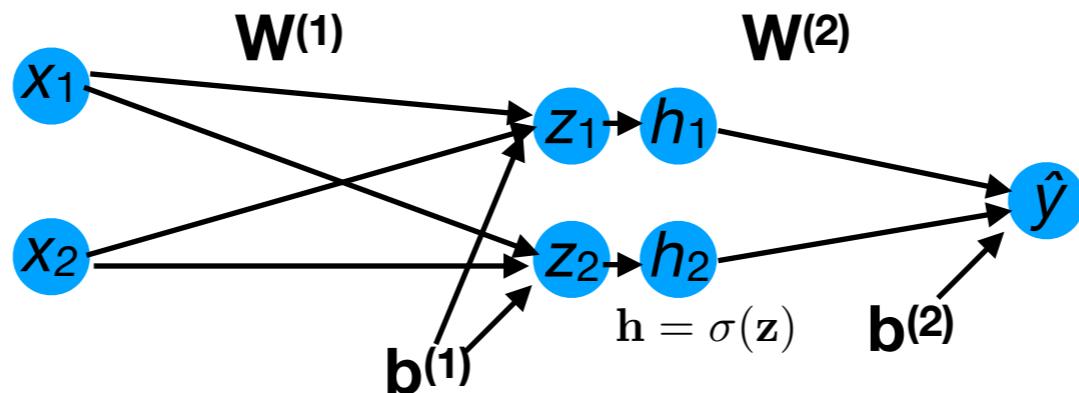
$$\text{ReLU}(z) = \max\{0, z\}$$



Surprisingly, the non-differentiability at 0 doesn't matter so much (since the gradient is often non-zero).

XOR problem

- Even though this is called a 3-layer NN, it is sometimes helpful to represent the **pre-activation** units \mathbf{z} and **hidden** units \mathbf{h} separately:



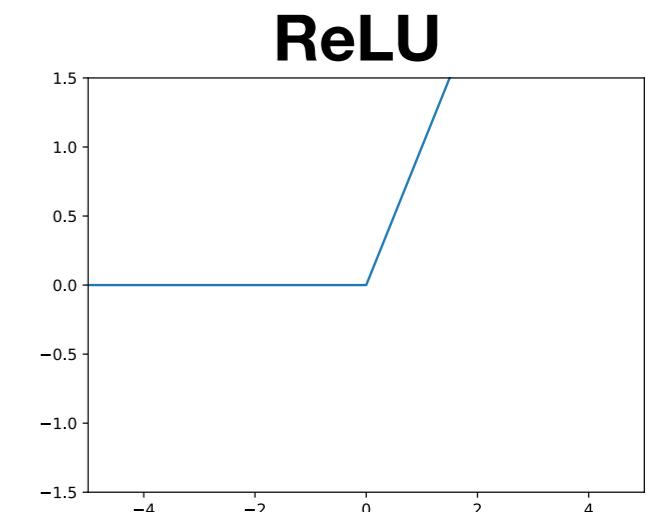
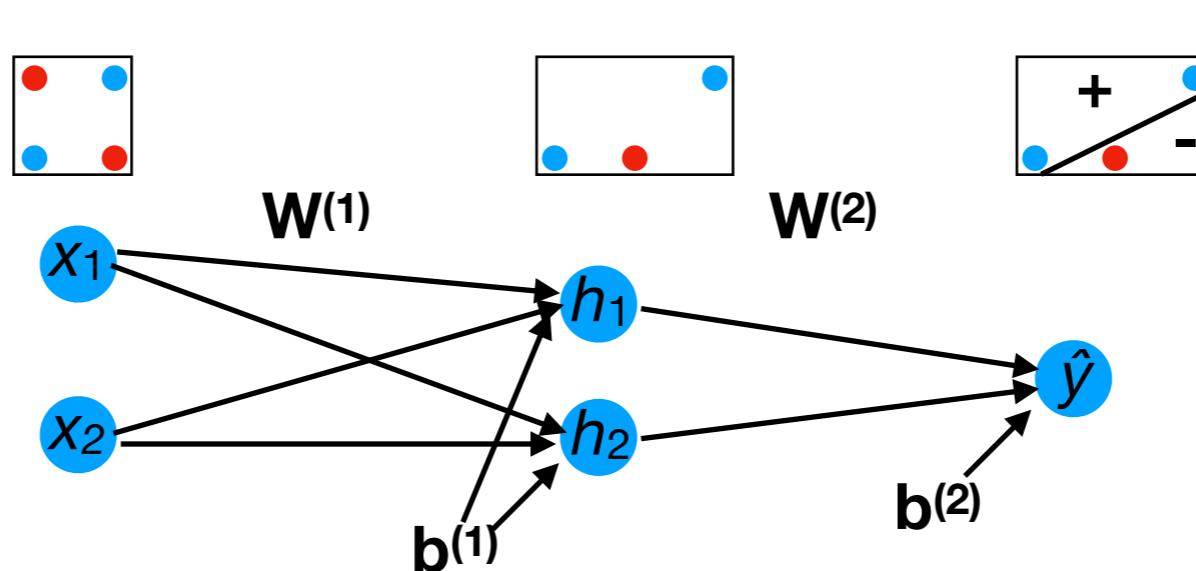
Input layer

Hidden layer

Output layer

XOR problem

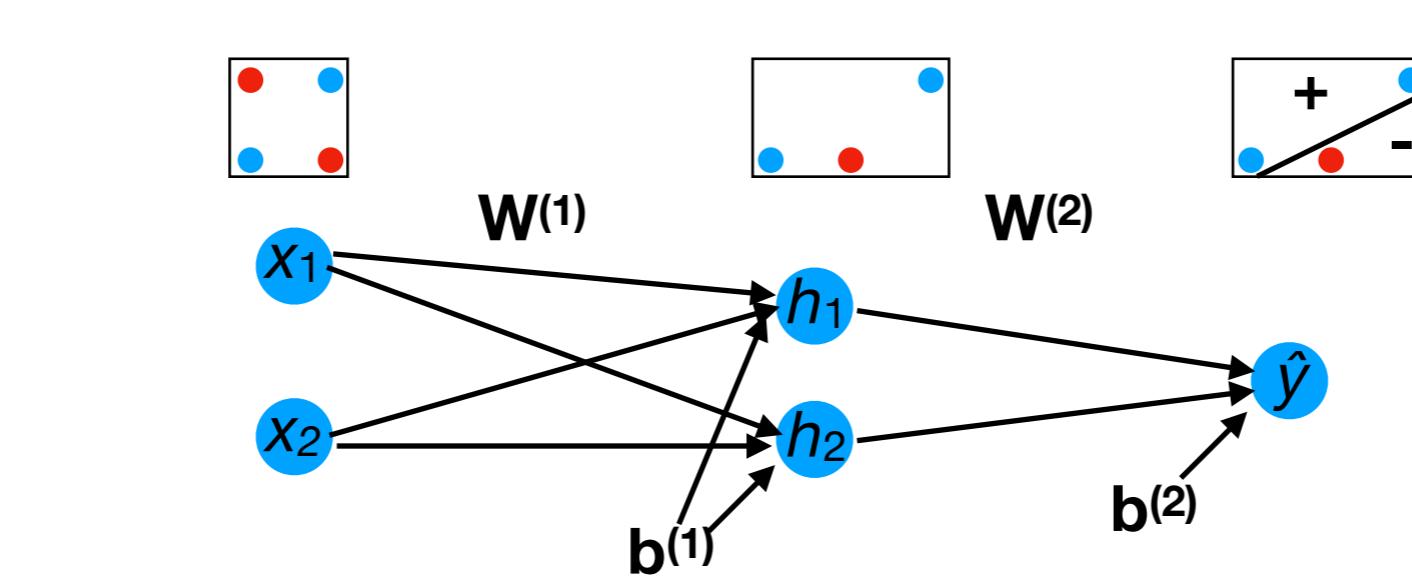
- Using ReLU in the hidden layer:
 - $\mathbf{W}^{(1)}, \mathbf{b}^{(1)}$ will “collapse” the two - data points onto one point in the “hidden” 2-D space.
 - Since the + and - data are now linearly separated, $\mathbf{W}^{(2)}, \mathbf{b}^{(2)}$ can easily distinguish the two classes.



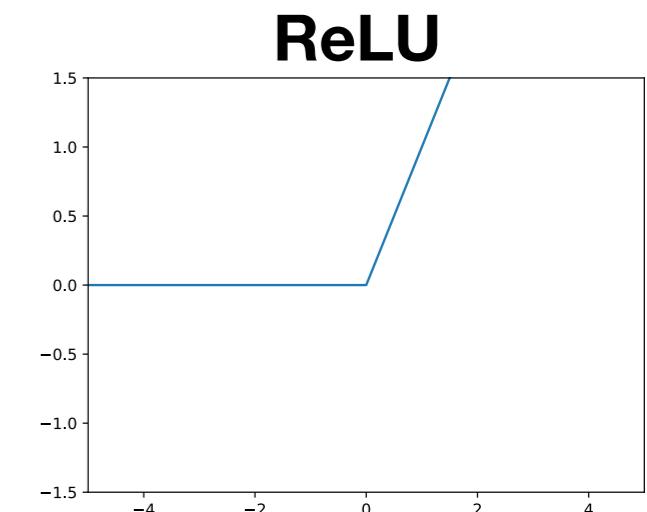
What values for $\mathbf{W}^{(1)}, \mathbf{b}^{(1)}$ will make the 4 data points linearly separable?
(Hint: set $\mathbf{b} = [-1 -1]^T$; \mathbf{W} contains only 1s and 2s.)

XOR problem

- Using ReLU in the hidden layer:
 - $\mathbf{W}^{(1)}, \mathbf{b}^{(1)}$ will “collapse” the two - data points onto one point in the “hidden” 2-D space.
 - Since the + and - data are now linearly separated, $\mathbf{W}^{(2)}, \mathbf{b}^{(2)}$ can easily distinguish the two classes.



$$\mathbf{W}^{(1)} = \begin{bmatrix} 2 & 2 \\ 1 & 1 \end{bmatrix} \quad \mathbf{b}^{(1)} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$



(There are other solutions as well.)

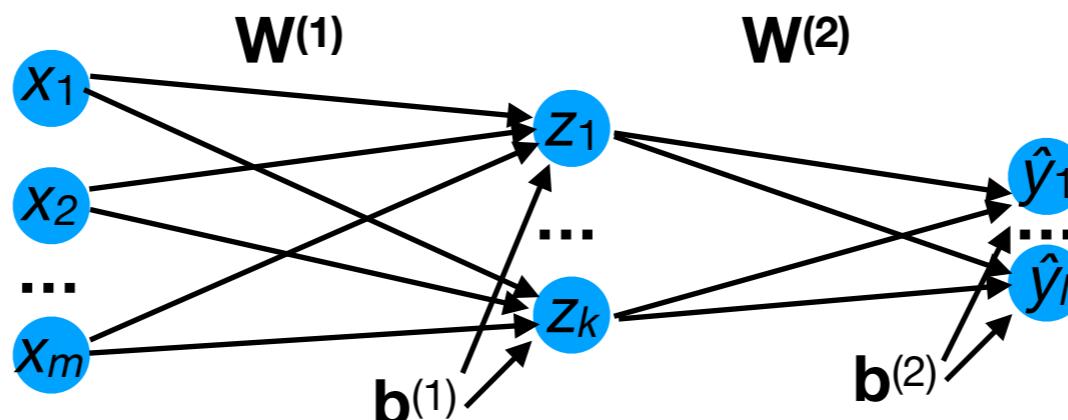
XOR problem

- Note that the ability of the 3-layer NN to solve the XOR problem relies crucially on the non-linear ReLU activation function.
 - Other non-linear functions would also work.
- Without non-linearity, a multi-layer NN is no more powerful than a 2-layer network!

Crucial role of non-linearity

- To see why, suppose we define a 3-layer NN **without** non-linearity:

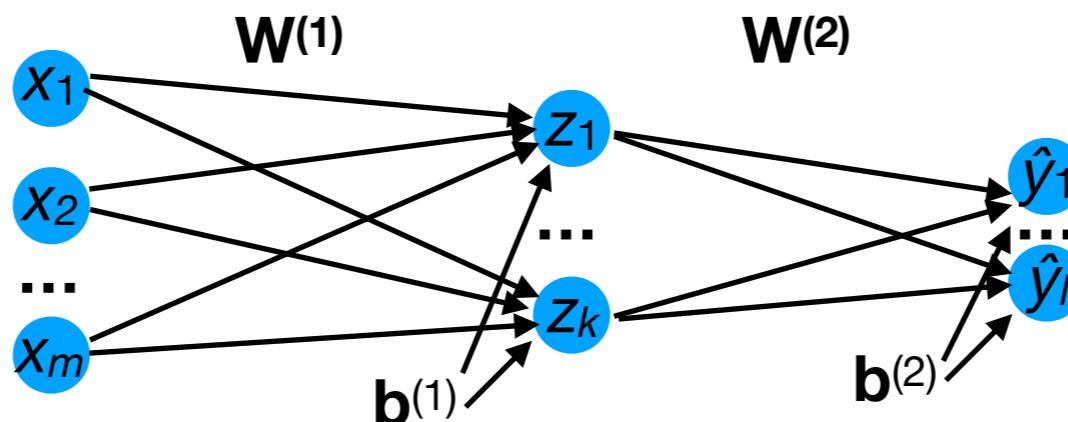
$$g(\mathbf{x}) = \mathbf{W}^{(2)} \left(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \right) + \mathbf{b}^{(2)}$$



Crucial role of non-linearity

- Then we can simplify g to be:

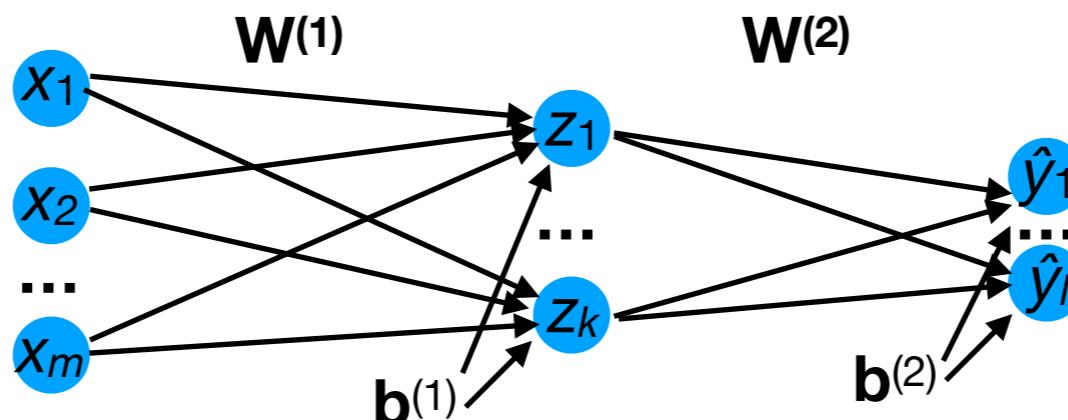
$$g(\mathbf{x}) = \mathbf{W}^{(2)} \left(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \right) + \mathbf{b}^{(2)}$$



Crucial role of non-linearity

- Then we can simplify g to be:

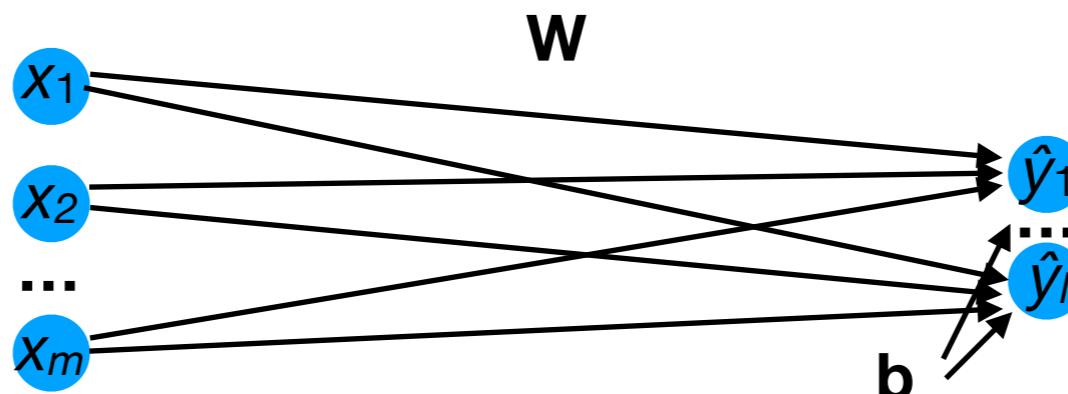
$$\begin{aligned} g(\mathbf{x}) &= \mathbf{W}^{(2)} \left(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \right) + \mathbf{b}^{(2)} \\ &= \left(\mathbf{W}^{(2)} \mathbf{W}^{(1)} \right) \mathbf{x} + \mathbf{W}^{(2)} \mathbf{b}^{(1)} + \mathbf{b}^{(2)} \end{aligned}$$



Crucial role of non-linearity

- Then we can simplify g to be:

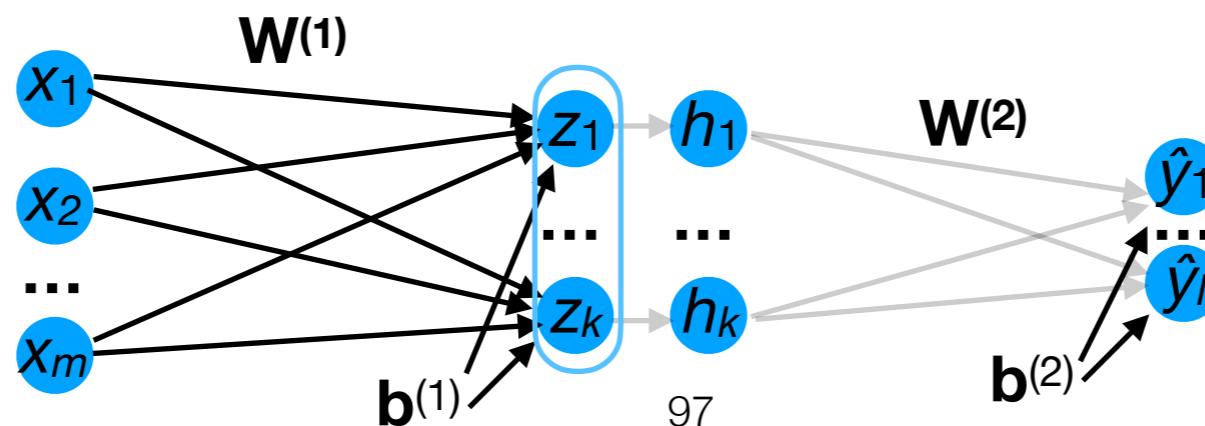
$$\begin{aligned} g(\mathbf{x}) &= \mathbf{W}^{(2)} \left(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \right) + \mathbf{b}^{(2)} \\ &= \left(\mathbf{W}^{(2)} \mathbf{W}^{(1)} \right) \mathbf{x} + \mathbf{W}^{(2)} \mathbf{b}^{(1)} + \mathbf{b}^{(2)} \\ &= \mathbf{Wx} + \mathbf{b} \end{aligned}$$



Feed-forward neural networks

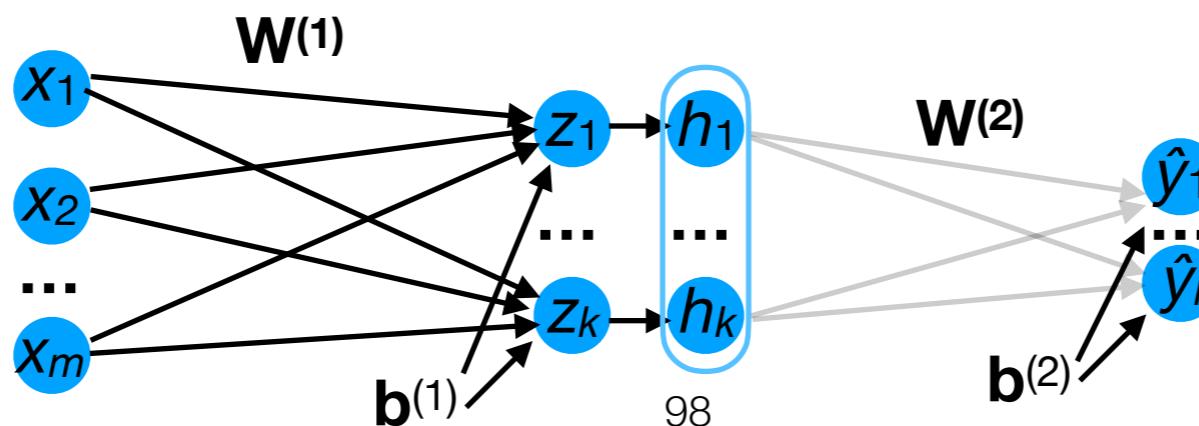
Forward propagation

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



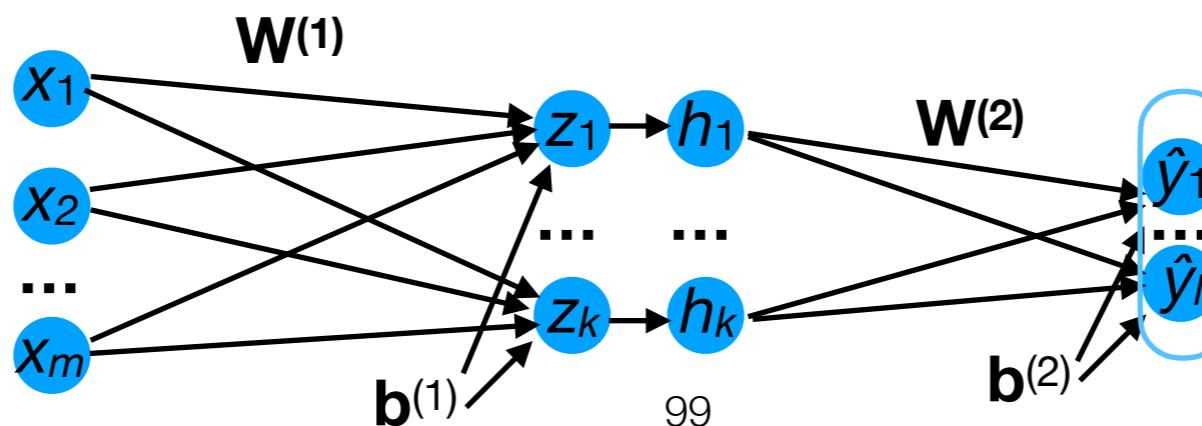
Forward propagation

- 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 $\mathbf{h} = \sigma(\mathbf{z})$.



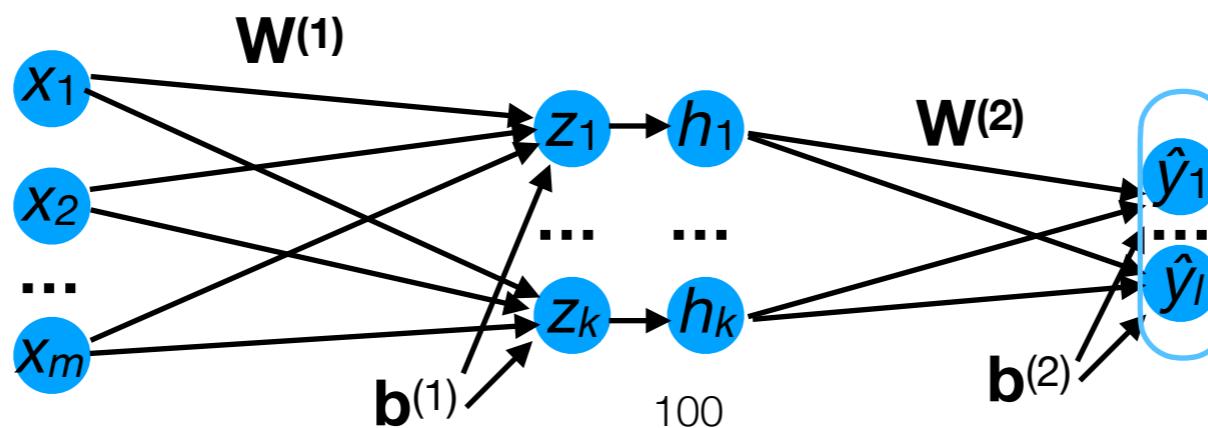
Forward propagation

- 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 $\hat{\mathbf{y}}$.



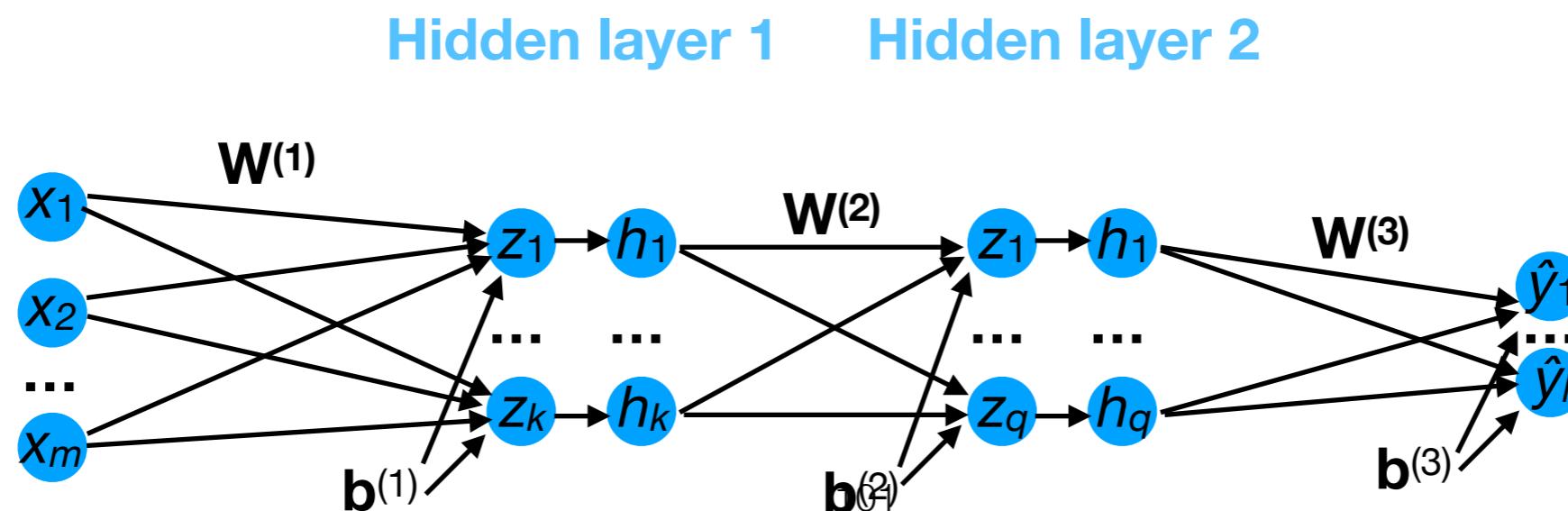
Forward propagation

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



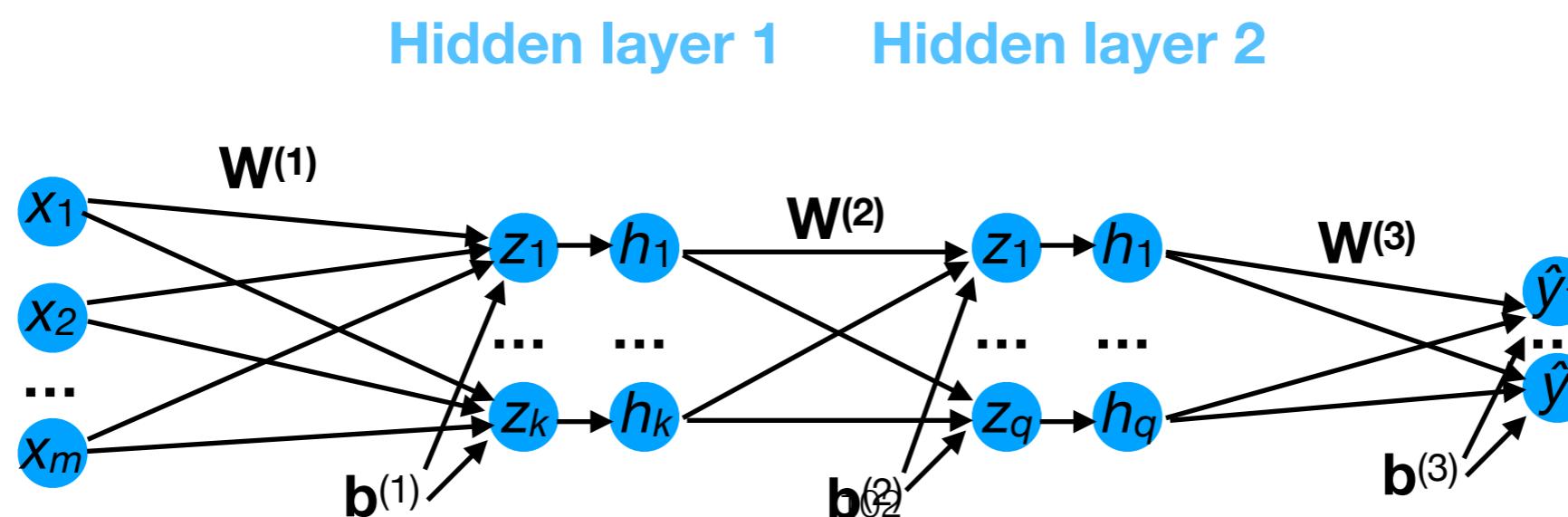
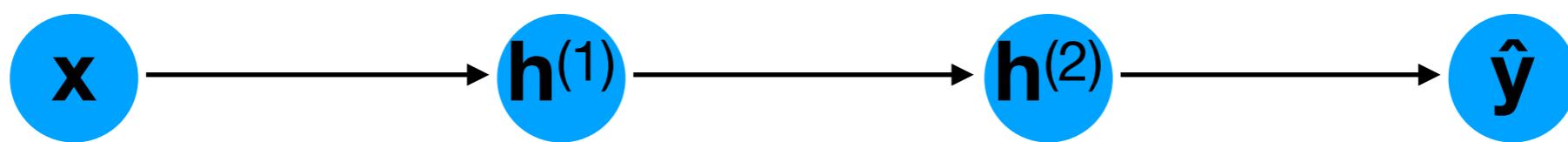
Forward propagation

- This process is known as **forward propagation**.
 - It can be applied to a NN of arbitrary depth; in the NN below, it would produce $\mathbf{z}^{(1)}$, $\mathbf{h}^{(1)}$, $\mathbf{z}^{(2)}$, $\mathbf{h}^{(2)}$, and $\hat{\mathbf{y}}$.



Forward propagation

- Forward propagation results in a **computational graph** representing the key intermediary values of the NN:



Training neural networks

- Except on toy problems (e.g., XOR), training neural networks by hand is completely impractical.
- How can we find good values for the weights and bias terms automatically?

Training neural networks

- Except on toy problems (e.g., XOR), training neural networks by hand is completely impractical.
- How can we find good values for the weights and bias terms automatically?
 - Gradient descent.

Training neural networks

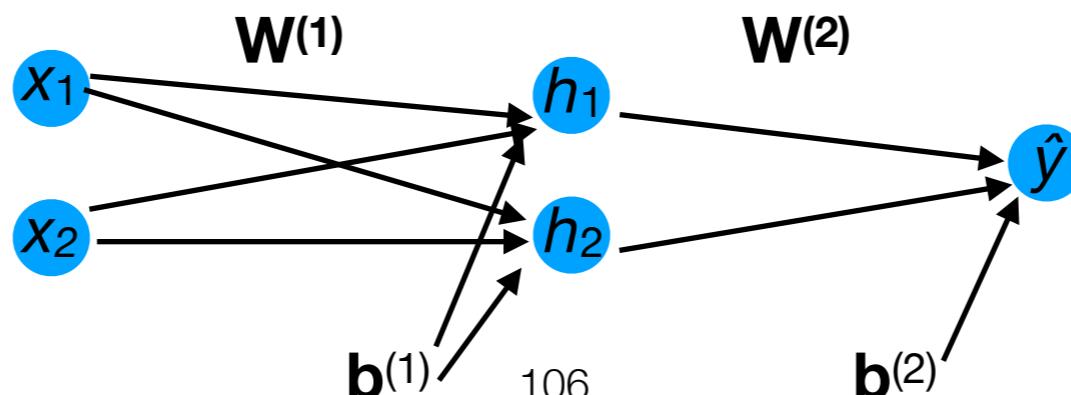
Gradient descent: XOR problem

- Here is how we can conduct gradient descent for the XOR problem...
- Let's first define:

$$\mathbf{W}^{(1)} = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix}, \mathbf{W}^{(2)} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix}, \mathbf{b}^{(1)} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \mathbf{b}^{(2)} = [b_3]$$

- Then we can define g so that:

$$\hat{y} = g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \mathbf{W}^{(2)} \sigma \left(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \right) + \mathbf{b}^{(2)}$$



Gradient descent: XOR problem

- Here is how we can conduct gradient descent for the XOR problem...
- Let's first define:

$$\mathbf{W}^{(1)} = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix}, \mathbf{W}^{(2)} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix}, \mathbf{b}^{(1)} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \mathbf{b}^{(2)} = [b_3]$$

- Then we can define g so that:

$$\begin{aligned}\hat{y} = g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) &= \mathbf{W}^{(2)} \sigma \left(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \right) + \mathbf{b}^{(2)} \\ &= \begin{bmatrix} w_5 \\ w_6 \end{bmatrix}^T \sigma \left(\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right) + b_3\end{aligned}$$

Gradient descent: XOR problem

- Here is how we can conduct gradient descent for the XOR problem...
- Let's first define:

$$\mathbf{W}^{(1)} = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix}, \mathbf{W}^{(2)} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix}, \mathbf{b}^{(1)} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \mathbf{b}^{(2)} = [b_3]$$

- Then we can define g so that:

$$\begin{aligned}\hat{y} = g \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) &= \mathbf{W}^{(2)} \sigma \left(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \right) + \mathbf{b}^{(2)} \\ &= \begin{bmatrix} w_5 \\ w_6 \end{bmatrix}^T \sigma \left(\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right) + b_3 \\ &= w_5 \sigma(w_1 x_1 + w_2 x_2 + b_1) + w_6 \sigma(w_3 x_1 + w_4 x_2 + b_2) + b_3\end{aligned}$$

Gradient descent: XOR problem

- From \hat{y} , we can compute the cost (f_{MSE}):

$$f_{\text{MSE}}(\hat{y}; w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3) = \frac{1}{2n} \sum_{i=1}^n (\hat{y}^{(i)} - y^{(i)})^2$$

Gradient descent: XOR problem

- From \hat{y} , we can compute the cost (f_{MSE}):

$$f_{\text{MSE}}(\hat{y}; w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3) = \frac{1}{2n} \sum_{i=1}^n (\hat{y}^{(i)} - y^{(i)})^2$$

- We then calculate the derivative of f_{MSE} w.r.t. each parameter p using the chain rule as:

$$\frac{\partial f_{\text{MSE}}}{\partial p} = \frac{1}{n} \sum_{i=1}^n (\hat{y}^{(i)} - y^{(i)}) \frac{\partial \hat{y}^{(i)}}{\partial p}$$

Gradient descent: XOR problem

- Now we just have to differentiate each $\hat{y} = g(\mathbf{x})$ w.r.t each parameter p :; e.g.:

$$\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial}{\partial w_1} [w_5\sigma(w_1x_1 + w_2x_2 + b_1) + w_6\sigma(w_3x_1 + w_4x_2 + b_2) + b_3]$$

Gradient descent: XOR problem

- Now we just have to differentiate each $\hat{y} = g(\mathbf{x})$ w.r.t each parameter p : e.g.:

$$\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial}{\partial w_1} [w_5\sigma(w_1x_1 + w_2x_2 + b_1) + w_6\sigma(w_3x_1 + w_4x_2 + b_2) + b_3]$$

This is the only term that depends on w_1 . It has the form: $c^*\sigma(z)$ where z is a function of w_1 . Recall that:

$$\begin{aligned}\frac{\partial}{\partial w_1} [c\sigma(z)] &= c\frac{\partial\sigma}{\partial z}(z)\frac{\partial z}{\partial w_1} \\ &= c\sigma'(z)\frac{\partial z}{\partial w_1}\end{aligned}$$

Gradient descent: XOR problem

- Now we just have to differentiate each $\hat{y} = g(\mathbf{x})$ w.r.t each parameter p :; e.g.:

$$\begin{aligned}\frac{\partial \hat{y}}{\partial w_1} &= \frac{\partial}{\partial w_1} [w_5\sigma(w_1x_1 + w_2x_2 + b_1) + w_6\sigma(w_3x_1 + w_4x_2 + b_2) + b_3] \\ &= w_5\end{aligned}$$

Gradient descent: XOR problem

- Now we just have to differentiate each $\hat{y} = g(\mathbf{x})$ w.r.t each parameter p :; e.g.:

$$\begin{aligned}\frac{\partial \hat{y}}{\partial w_1} &= \frac{\partial}{\partial w_1} [w_5\sigma(w_1x_1 + w_2x_2 + b_1) + w_6\sigma(w_3x_1 + w_4x_2 + b_2) + b_3] \\ &= w_5\sigma'(w_1x_1 + w_2x_2 + b_1)\end{aligned}$$

Gradient descent: XOR problem

- Now we just have to differentiate each $\hat{y} = g(\mathbf{x})$ w.r.t each parameter p : , e.g.:

$$\begin{aligned}\frac{\partial \hat{y}}{\partial w_1} &= \frac{\partial}{\partial w_1} [w_5\sigma(w_1x_1 + w_2x_2 + b_1) + w_6\sigma(w_3x_1 + w_4x_2 + b_2) + b_3] \\ &= w_5\sigma'(w_1x_1 + w_2x_2 + b_1)x_1\end{aligned}$$

Gradient descent: XOR problem

- Now we just have to differentiate each $\hat{y} = g(\mathbf{x})$ w.r.t each parameter p : e.g.:

$$\begin{aligned}\frac{\partial \hat{y}}{\partial w_1} &= \frac{\partial}{\partial w_1} [w_5\sigma(w_1x_1 + w_2x_2 + b_1) + w_6\sigma(w_3x_1 + w_4x_2 + b_2) + b_3] \\ &= w_5\sigma'(w_1x_1 + w_2x_2 + b_1)x_1\end{aligned}$$

where:

$$\sigma'(z) = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{if } z > 0 \end{cases} \quad \text{for ReLU}$$

Gradient descent: XOR problem

- Hence:

$$\frac{\partial \hat{y}}{\partial w_1} = \begin{cases} 0 & \text{if } w_1x_1 + w_2x_2 + b_1 \leq 0 \\ w_5x_1 & \text{if } w_1x_1 + w_2x_2 + b_1 > 0 \end{cases}$$

since:

$$\sigma'(z) = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{if } z > 0 \end{cases} \quad \text{for ReLU}$$

Exercise

- Find:

$$\frac{\partial \hat{y}}{\partial b_2} = \frac{\partial}{\partial b_2} [w_5\sigma(w_1x_1 + w_2x_2 + b_1) + w_6\sigma(w_3x_1 + w_4x_2 + b_2) + b_3]$$

Exercise

- Find:

$$\begin{aligned}\frac{\partial \hat{y}}{\partial b_2} &= \frac{\partial}{\partial b_2} [w_5\sigma(w_1x_1 + w_2x_2 + b_1) + w_6\sigma(w_3x_1 + w_4x_2 + b_2) + b_3] \\ &= w_6\sigma'(w_3x_1 + w_4x_2 + b_2) \\ &= \begin{cases} 0 & \text{if } w_3x_1 + w_4x_2 + b_2 \leq 0 \\ w_6 & \text{if } w_3x_1 + w_4x_2 + b_2 > 0 \end{cases}\end{aligned}$$

Gradient descent: XOR problem

- We also need to derive the other partial derivatives:

$$\frac{\partial \hat{y}}{\partial w_1}, \dots, \frac{\partial \hat{y}}{\partial w_6}, \frac{\partial \hat{y}}{\partial b_1}, \dots, \frac{\partial \hat{y}}{\partial b_3}$$

- Based on these, we can compute the gradient terms:

$$\frac{\partial f_{\text{MSE}}}{\partial w_1} = \frac{\partial f_{\text{MSE}}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_1}$$

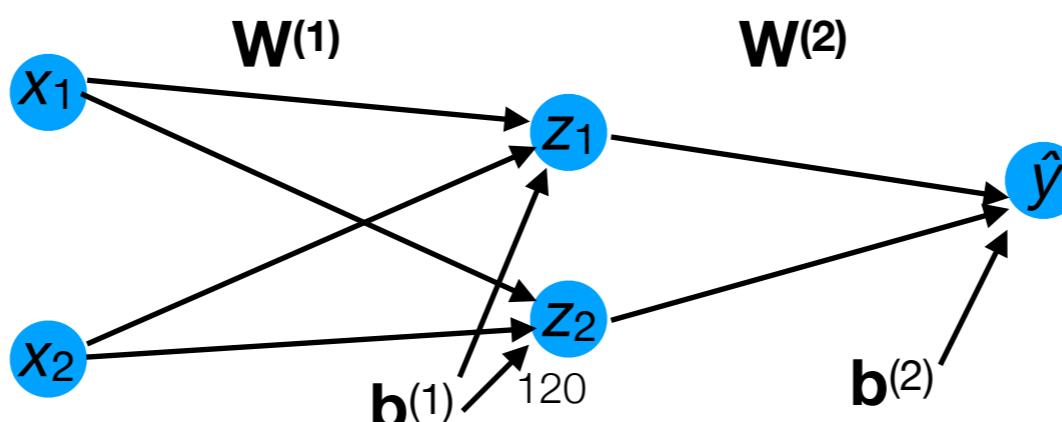
...

$$\frac{\partial f_{\text{MSE}}}{\partial b_1} = \frac{\partial f_{\text{MSE}}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_1}$$

...

$$\frac{\partial f_{\text{MSE}}}{\partial w_6} = \frac{\partial f_{\text{MSE}}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_6}$$

$$\frac{\partial f_{\text{MSE}}}{\partial b_3} = \frac{\partial f_{\text{MSE}}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_3}$$



Gradient descent: XOR problem

- Given the gradients, we can conduct gradient descent:
 - For each epoch:
 - For each minibatch:
 - Compute all 9 partial derivatives (summed over all examples in the minibatch):
 - Update w_1 : $w_1 \leftarrow w_1 - \epsilon \frac{\partial f_{\text{MSE}}}{\partial w_1}$
 - ...
 - Update w_6 : $w_6 \leftarrow w_6 - \epsilon \frac{\partial f_{\text{MSE}}}{\partial w_6}$
 - Update b_1 : $b_1 \leftarrow b_1 - \epsilon \frac{\partial f_{\text{MSE}}}{\partial b_1}$
 - ...
 - Update b_3 : $b_3 \leftarrow b_3 - \epsilon \frac{\partial f_{\text{MSE}}}{\partial b_3}$

Gradient descent: (any 3-layer NN)

- It is more compact and efficient to represent and compute these gradients more abstractly:

$$\nabla_{\mathbf{W}^{(1)}} f_{\text{MSE}}$$

$$\nabla_{\mathbf{W}^{(2)}} f_{\text{MSE}}$$

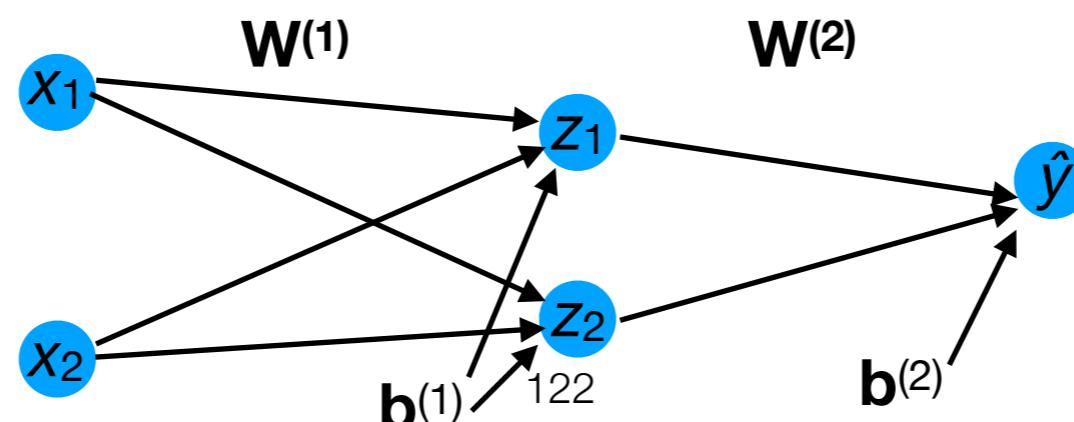
$$\nabla_{\mathbf{b}^{(1)}} f_{\text{MSE}}$$

$$\nabla_{\mathbf{b}^{(2)}} f_{\text{MSE}}$$

where

$$\nabla_{\mathbf{W}^{(1)}} f_{\text{MSE}} = \begin{bmatrix} \frac{\partial f_{\text{MSE}}}{\partial w_1} & \frac{\partial f_{\text{MSE}}}{\partial w_2} \\ \frac{\partial f_{\text{MSE}}}{\partial w_3} & \frac{\partial f_{\text{MSE}}}{\partial w_4} \end{bmatrix}$$

etc.



Gradient descent (any 3-layer NN)

- Given the gradients, we can conduct gradient descent:
 - For each epoch:
 - For each minibatch:
 - Compute all gradient terms (summed over all examples in the minibatch):
 - Update $\mathbf{W}^{(1)}$: $\mathbf{W}^{(1)} \leftarrow \mathbf{W}^{(1)} - \epsilon \nabla_{\mathbf{W}^{(1)}} f_{\text{MSE}}$

Update $\mathbf{W}^{(2)}$: $\mathbf{W}^{(2)} \leftarrow \mathbf{W}^{(2)} - \epsilon \nabla_{\mathbf{W}^{(2)}} f_{\text{MSE}}$

Update $\mathbf{b}^{(1)}$: $\mathbf{b}^{(1)} \leftarrow \mathbf{b}^{(1)} - \epsilon \nabla_{\mathbf{b}^{(1)}} f_{\text{MSE}}$

Update $\mathbf{b}^{(2)}$: $\mathbf{b}^{(2)} \leftarrow \mathbf{b}^{(2)} - \epsilon \nabla_{\mathbf{b}^{(2)}} f_{\text{MSE}}$

Deriving the gradient terms

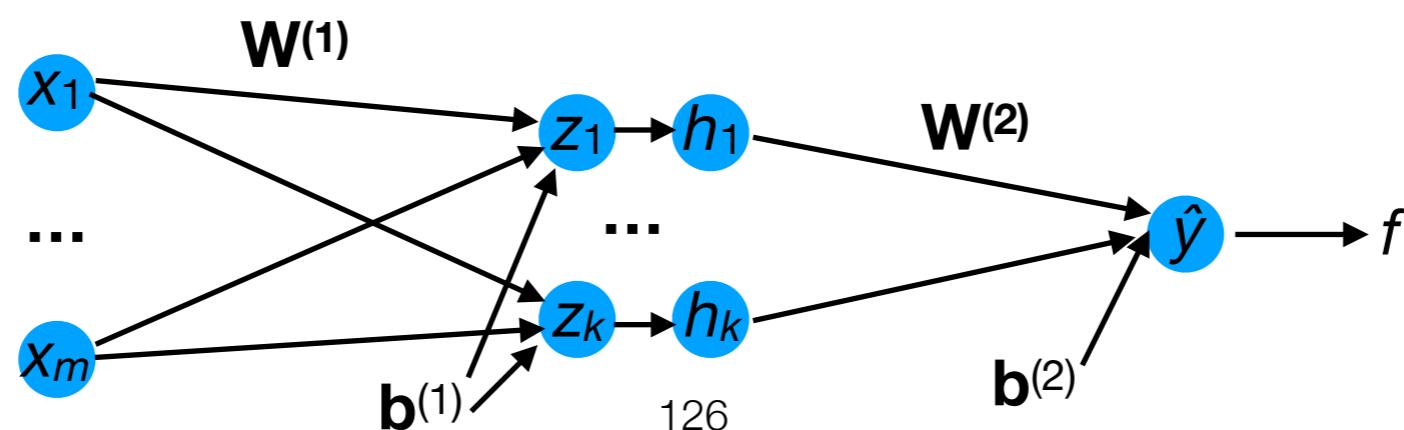
- We can largely *automate* the process of computing each gradient term $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \dots, \mathbf{W}^{(d)}$ (for d layers).
- This procedure is enabled by the **chain rule of multivariate calculus**.

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:

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



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:
$$\nabla_p f(\mathbf{X}, \mathbf{y}; \mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots)$$
 - Update the parameter by moving it slightly opposite the gradient:

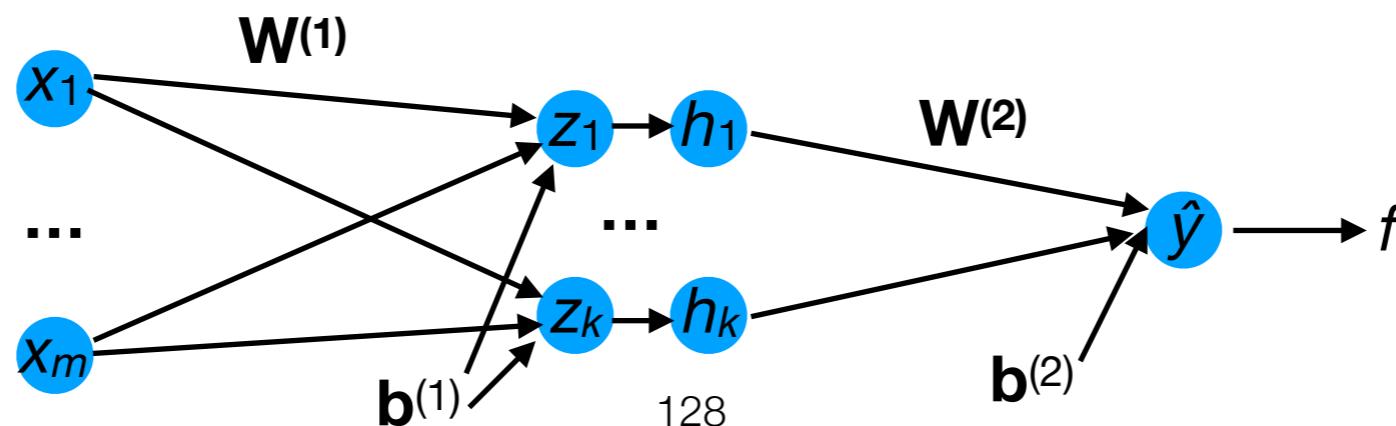
$$p^{\text{new}} \leftarrow p^{\text{old}} - \epsilon \nabla_p f(\mathbf{X}, \mathbf{y}; \dots, p^{\text{old}}, \dots)$$

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.:

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

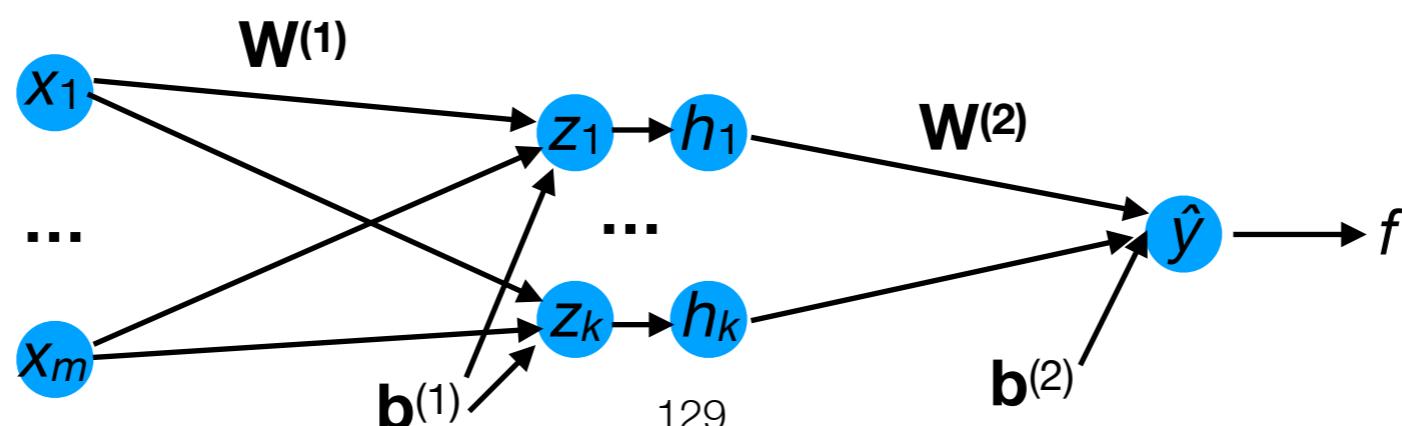
- This means we must compute multiple Jacobian matrices and multiply them together.



Computing the gradients

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

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

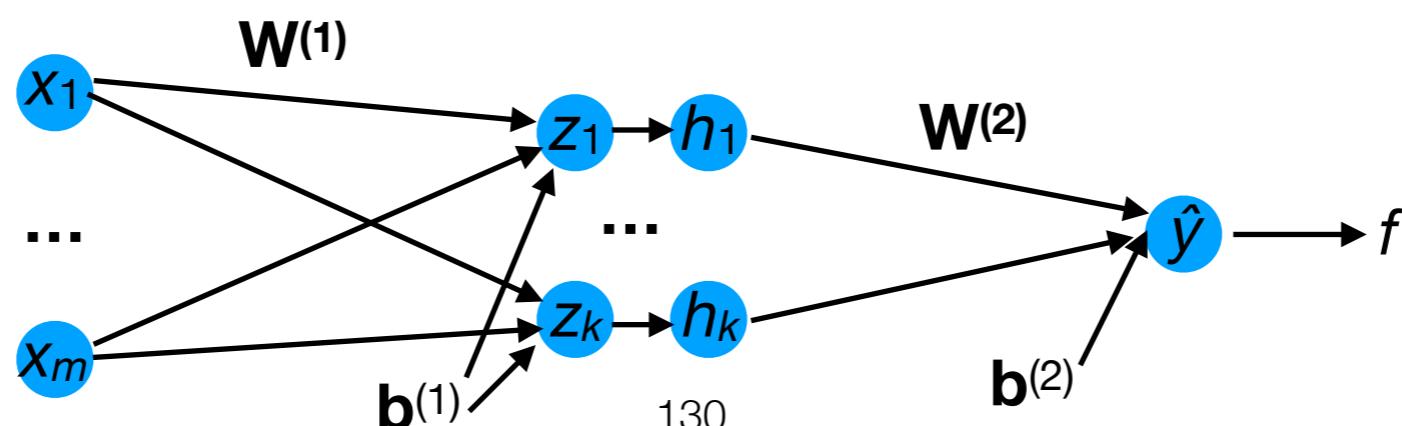


Computing the gradients

- Note that the gradient terms share some computation:

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

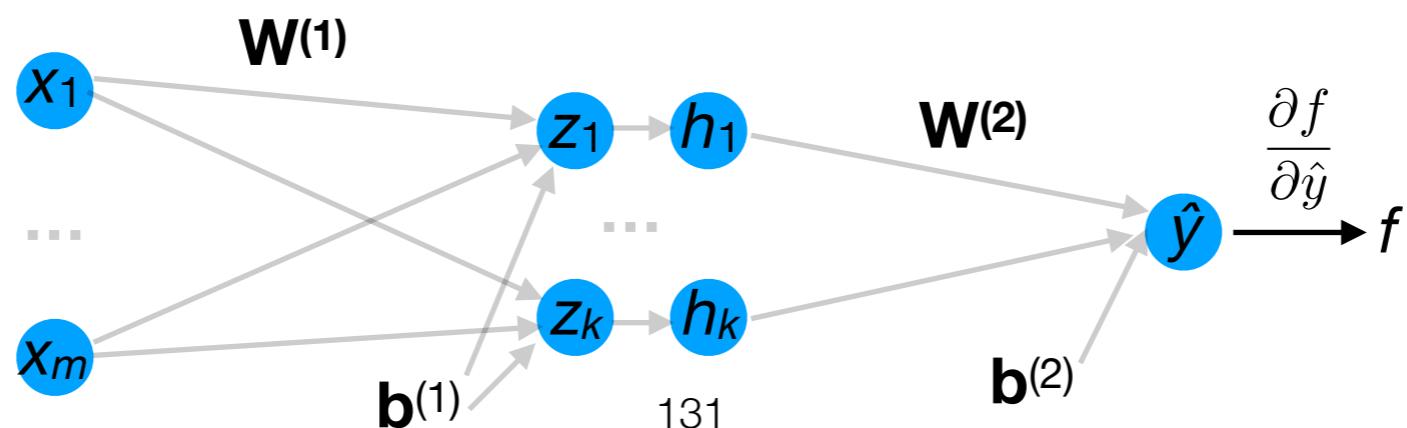
**Redundant
computation**



Computing the gradients

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

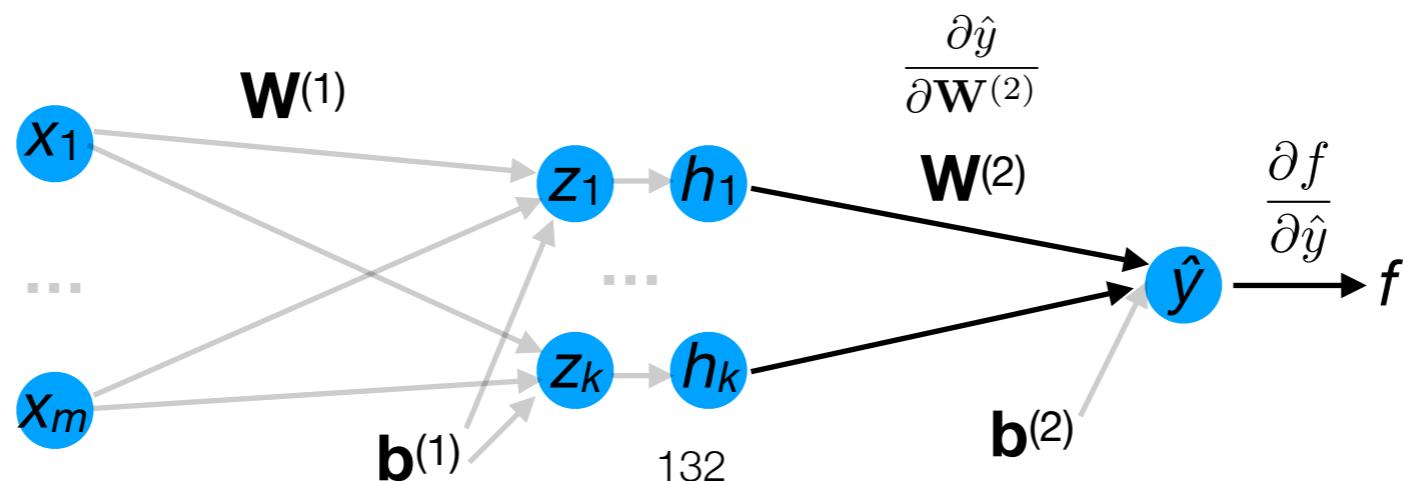
$$\frac{\partial f}{\partial \mathbf{W}^{(2)}} = \frac{\partial f}{\partial \hat{y}}.$$



Computing the gradients

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

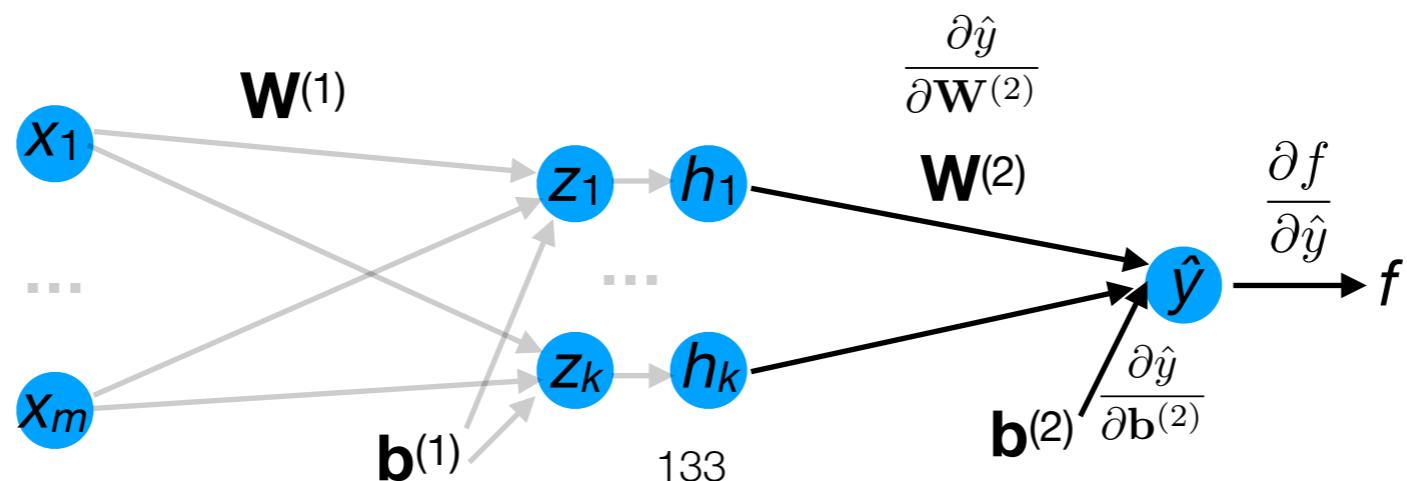
$$\frac{\partial f}{\partial \mathbf{W}^{(2)}} = \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}}$$



Computing the gradients

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

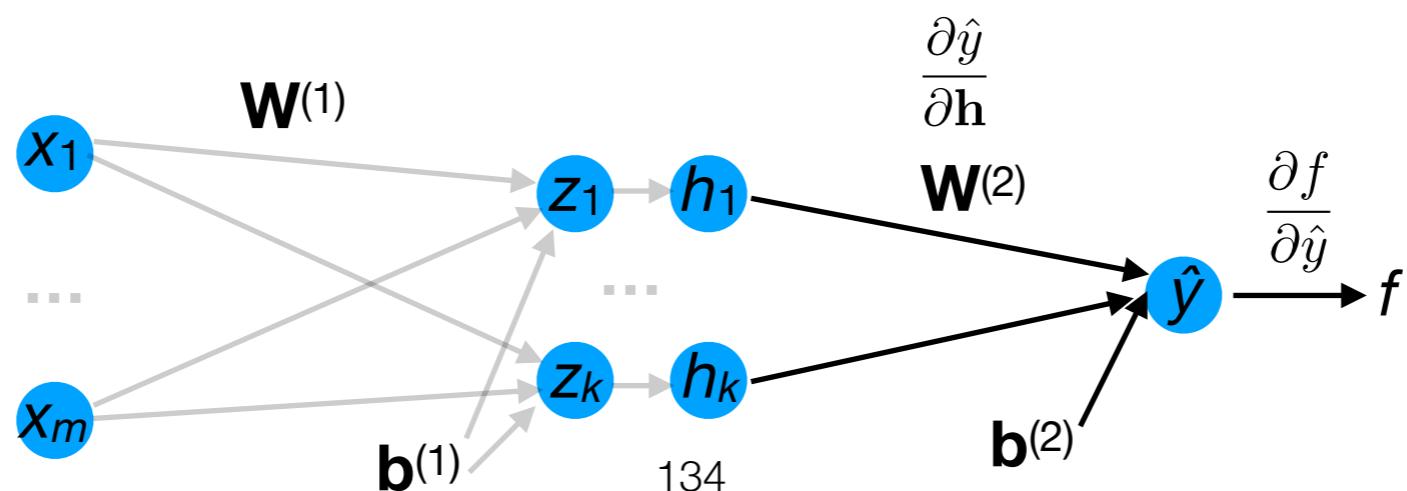
$$\begin{aligned}\frac{\partial f}{\partial \mathbf{W}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}} \\ \frac{\partial f}{\partial \mathbf{b}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}}\end{aligned}$$



Computing the gradients

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

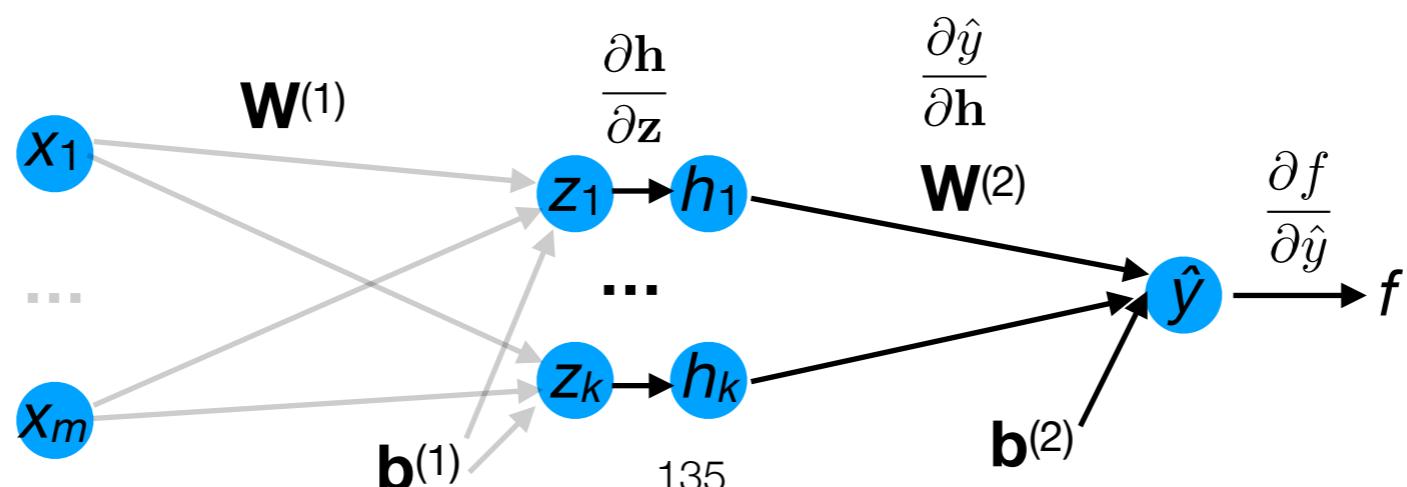
$$\begin{aligned}\frac{\partial f}{\partial \mathbf{W}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}} \\ \frac{\partial f}{\partial \mathbf{b}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}} \\ \frac{\partial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



Computing the gradients

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

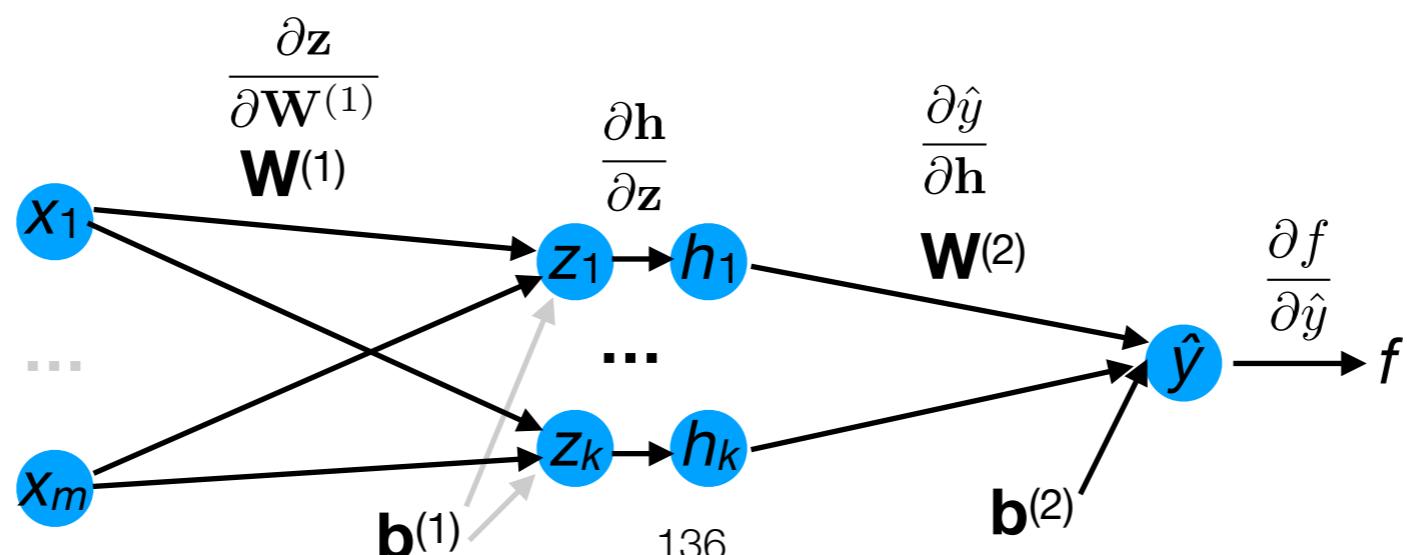
$$\begin{aligned}\frac{\partial f}{\partial \mathbf{W}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}} \\ \frac{\partial f}{\partial \mathbf{b}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}} \\ \frac{\partial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



Computing the gradients

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

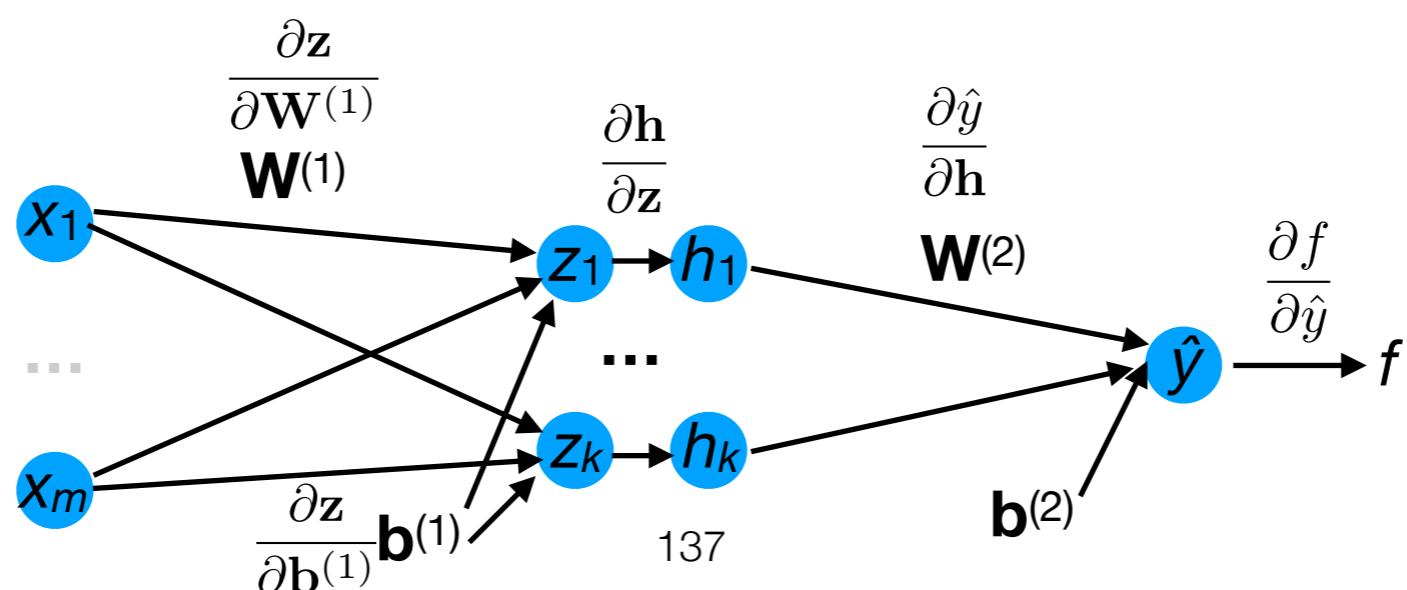
$$\begin{aligned}\frac{\partial f}{\partial \mathbf{W}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}} \\ \frac{\partial f}{\partial \mathbf{b}^{(2)}} &= \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{b}^{(2)}} \\ \frac{\partial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



Computing the gradients

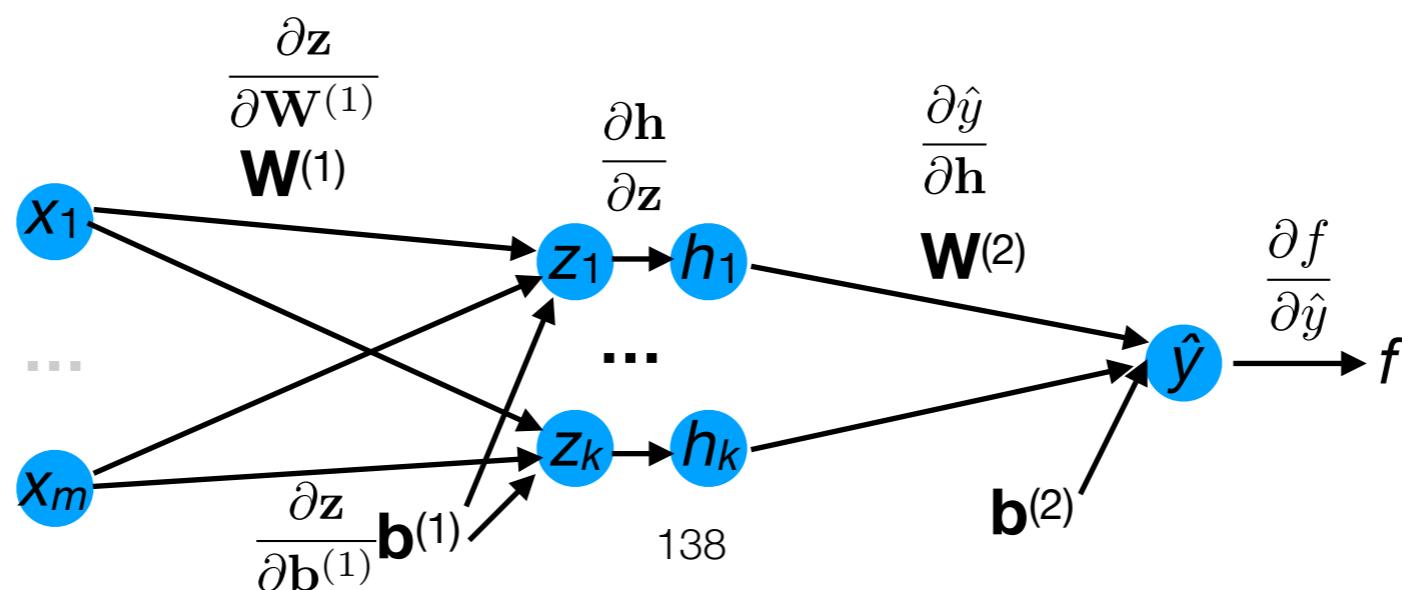
- Here's how we can compute all these *efficiently*:

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



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.



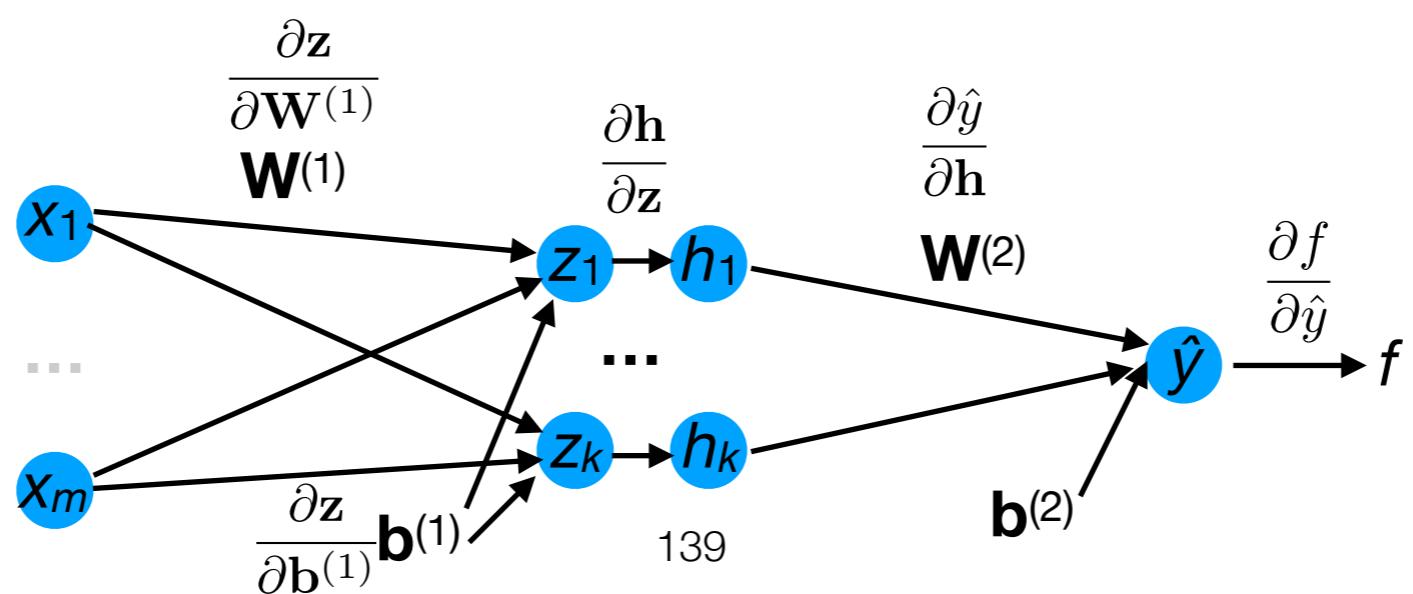
Computing the gradients

Forward propagation



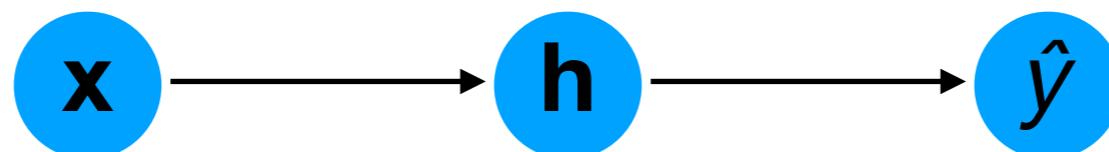
Backward propagation

$$\frac{\partial f}{\partial \hat{y}}$$

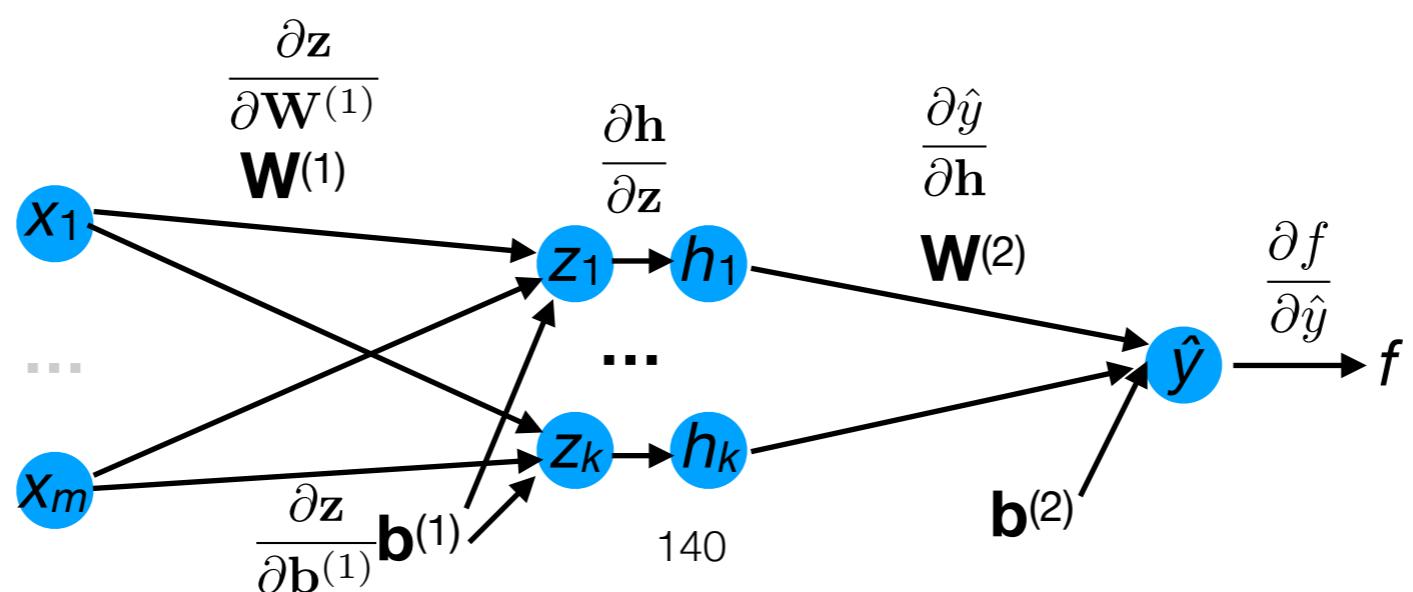
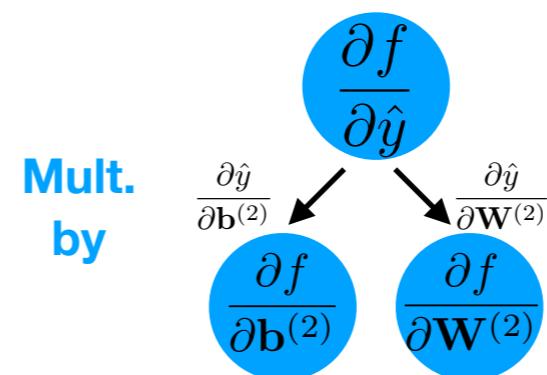


Computing the gradients

Forward propagation



Backward propagation

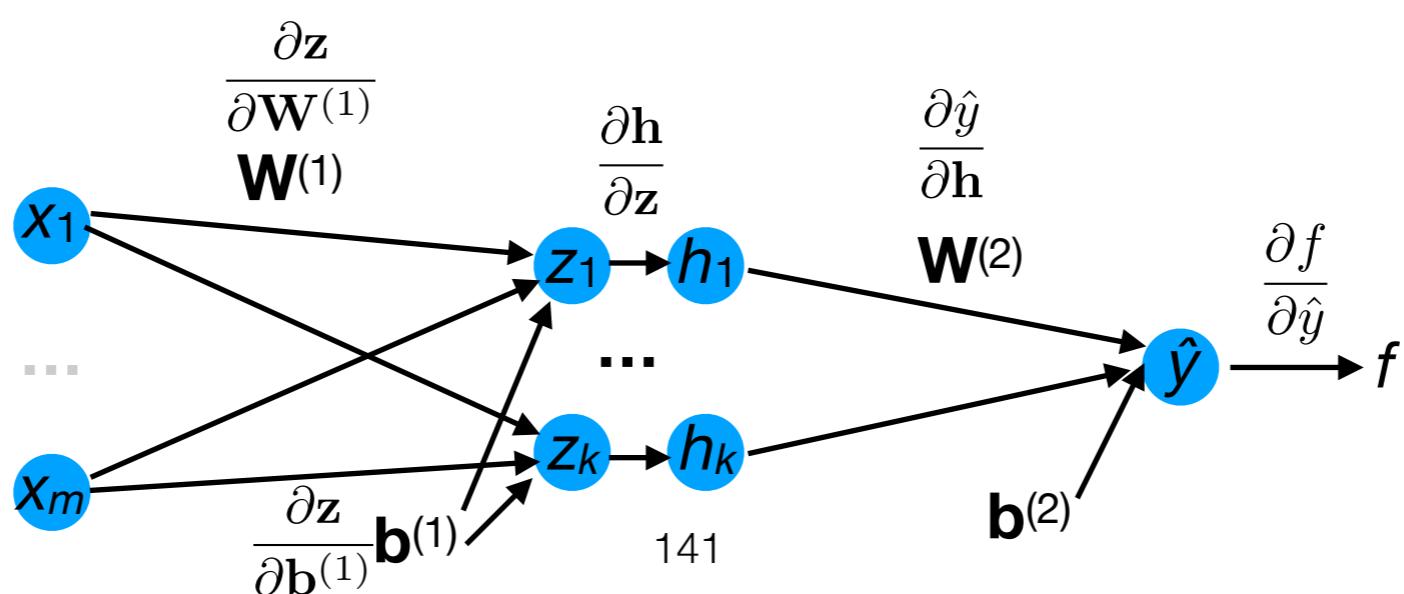
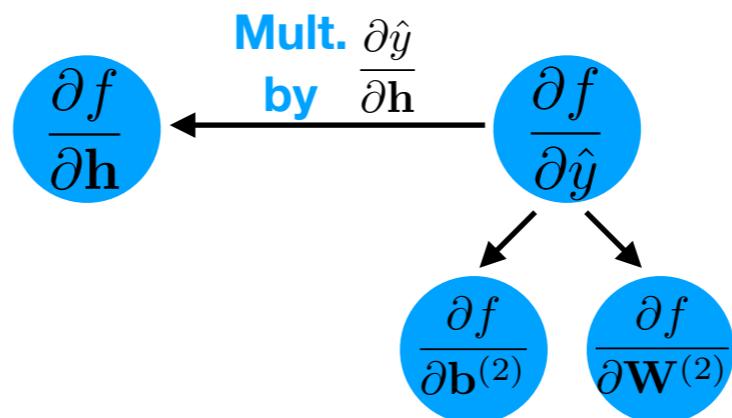


Computing the gradients

Forward propagation



Backward propagation

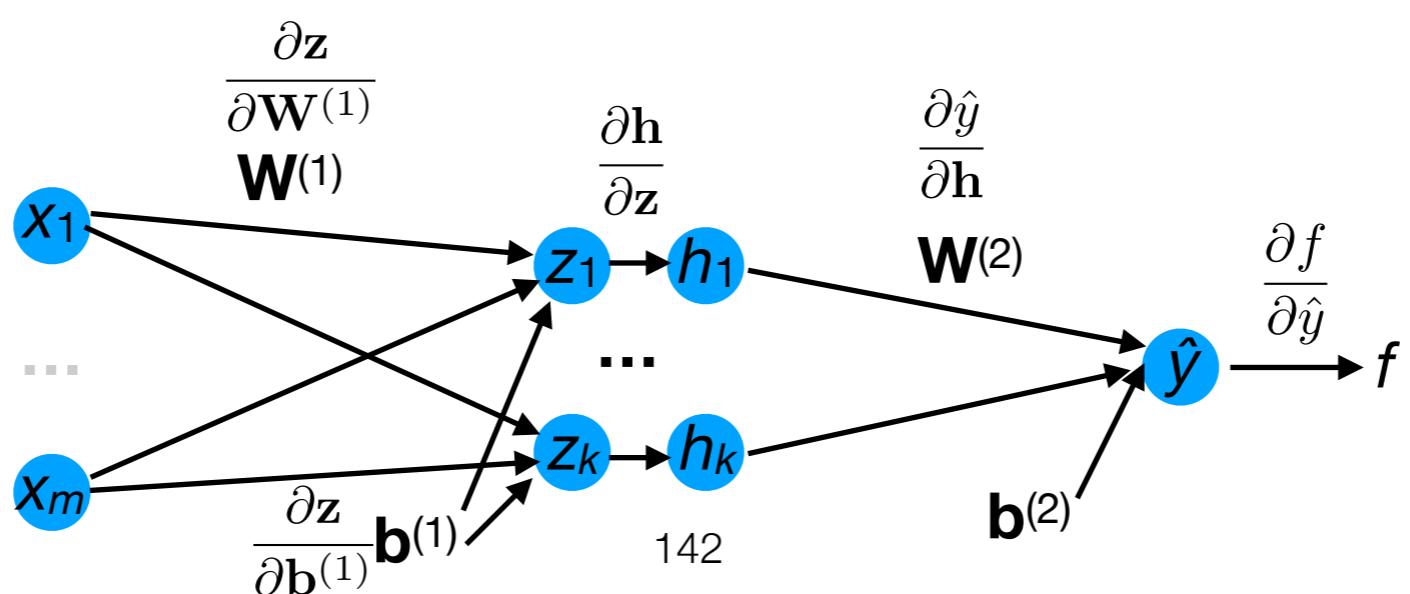
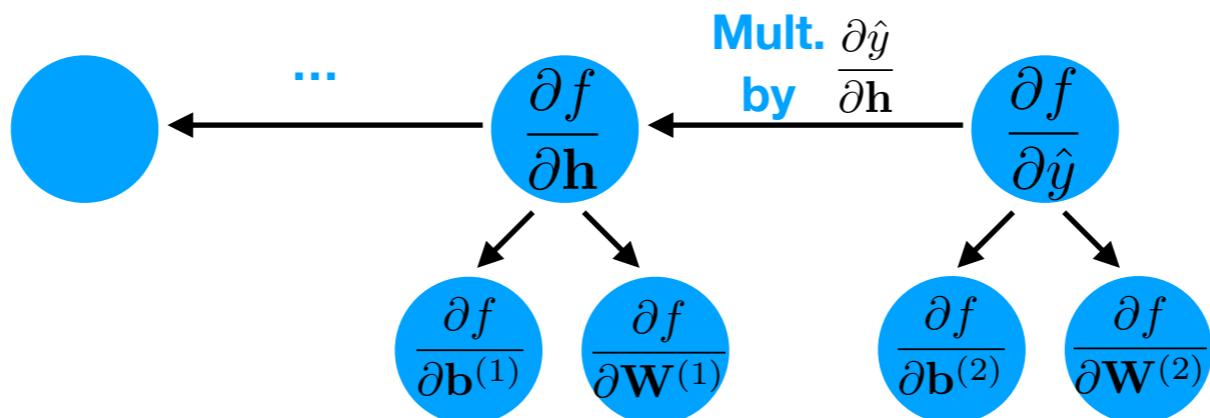


Computing the gradients

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 \mathbf{x} , a target \mathbf{y} . This computation yields the gradients on the activations $\mathbf{a}^{(k)}$ 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:

```
 $\mathbf{g} \leftarrow \nabla_{\hat{\mathbf{y}}} J = \nabla_{\hat{\mathbf{y}}} L(\hat{\mathbf{y}}, \mathbf{y})$ 
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):

$$\mathbf{g} \leftarrow \nabla_{\mathbf{a}^{(k)}} J = \mathbf{g} \odot f'(\mathbf{a}^{(k)})$$

Compute gradients on weights and biases (including the regularization term, where needed):

$$\begin{aligned}\nabla_{\mathbf{b}^{(k)}} J &= \mathbf{g} + \lambda \nabla_{\mathbf{b}^{(k)}} \Omega(\theta) \\ \nabla_{\mathbf{W}^{(k)}} J &= \mathbf{g} \mathbf{h}^{(k-1)\top} + \lambda \nabla_{\mathbf{W}^{(k)}} \Omega(\theta)\end{aligned}$$

Propagate the gradients w.r.t. the next lower-level hidden layer's activations:

$$\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(k-1)}} J = \mathbf{W}^{(k)\top} \mathbf{g}$$

```
end for
```

Computing the gradients

- Where do these come from?

$$\nabla_{\mathbf{W}^{(2)}} f_{\text{CE}} = (\hat{\mathbf{y}} - \mathbf{y}) \mathbf{h}^{(1)^\top}$$

$$\nabla_{\mathbf{b}^{(2)}} f_{\text{CE}} = (\hat{\mathbf{y}} - \mathbf{y})$$

$$\nabla_{\mathbf{W}^{(1)}} f_{\text{CE}} = \mathbf{g} \mathbf{x}^\top$$

$$\nabla_{\mathbf{b}^{(1)}} f_{\text{CE}} = \mathbf{g}$$

where

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

