CS/DS 541: Class 5

Jacob Whitehill

More on latent variable models (LVMs)

Principal component analysis (PCA)

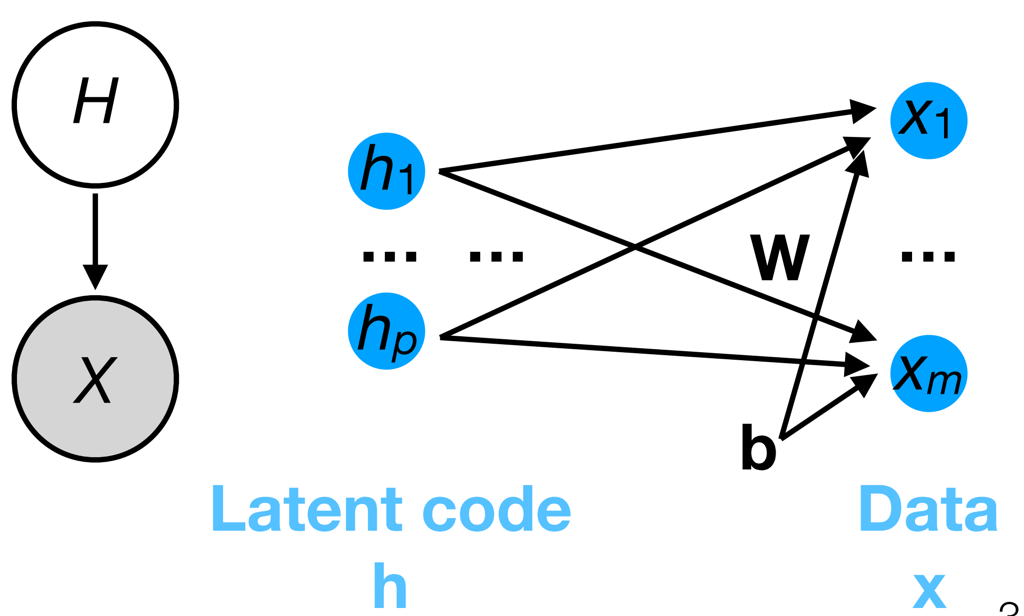
- Probabilistic PCA is an LVM that allows us to **generate** novel examples:

1. Sample from the prior distribution over **h**.

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

2. Sample from the conditional distribution of **x** | **h**.

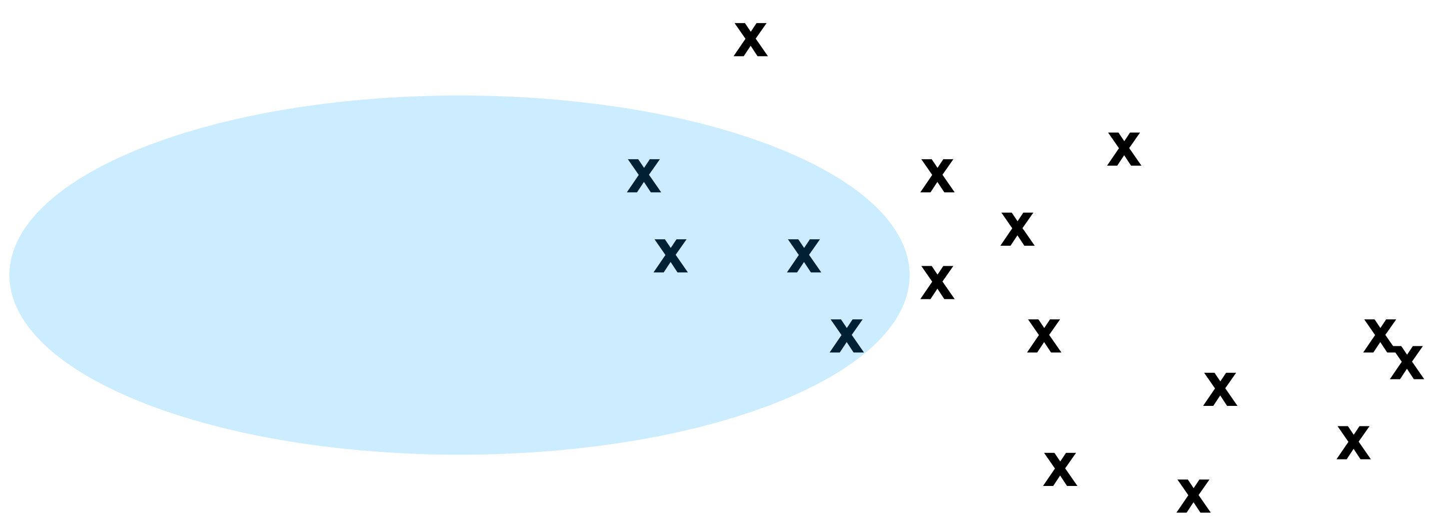
$$P(\mathbf{x} \mid \mathbf{h}, \mathbf{W}, \mathbf{b}) = \mathcal{N}(\mathbf{x}; \mathbf{W}\mathbf{h} + \mathbf{b}, \sigma^2 \mathbf{I})$$



MLE for a multivariate Gaussian distribution

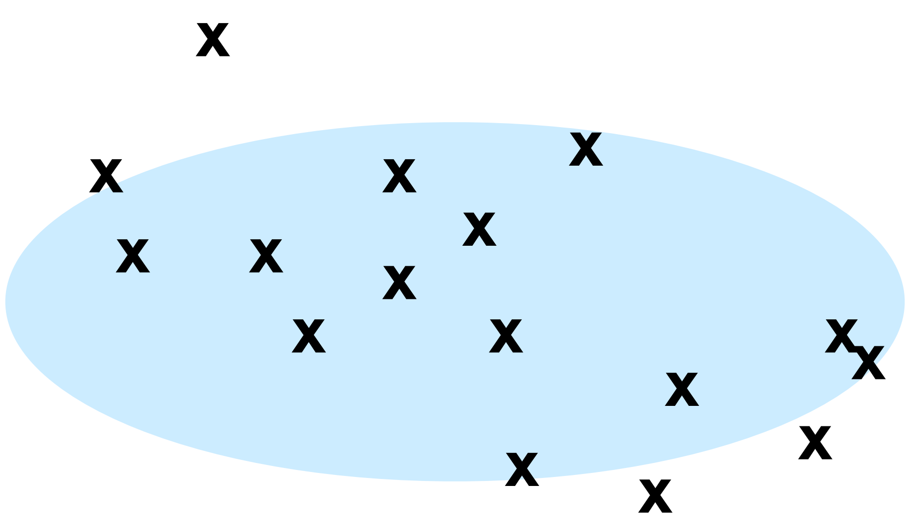
- Suppose we have the following dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{n}$.

- Which Gaussian (with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$) fits these data better?



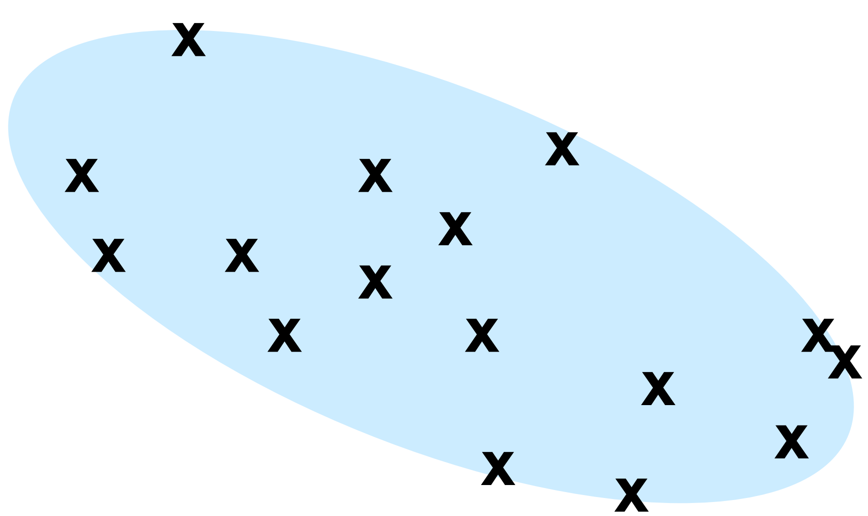
MLE for a multivariate Gaussian distribution

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



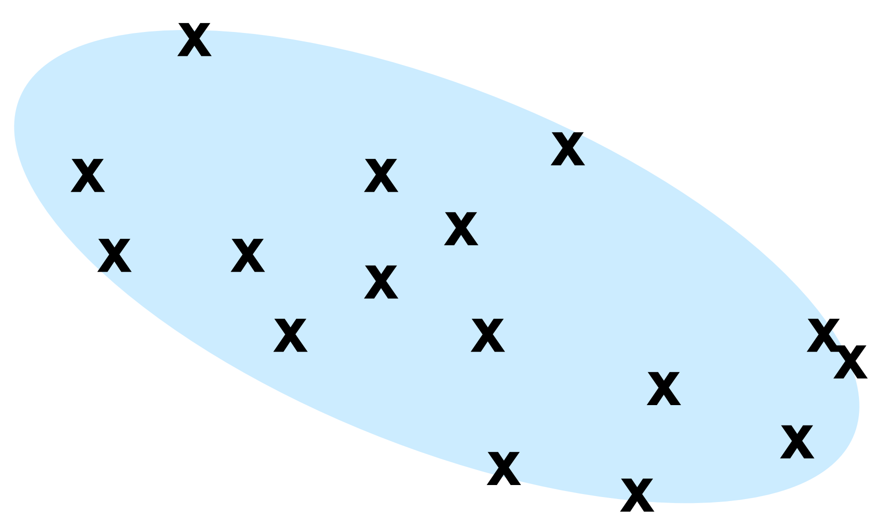
MLE for a multivariate Gaussian distribution

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



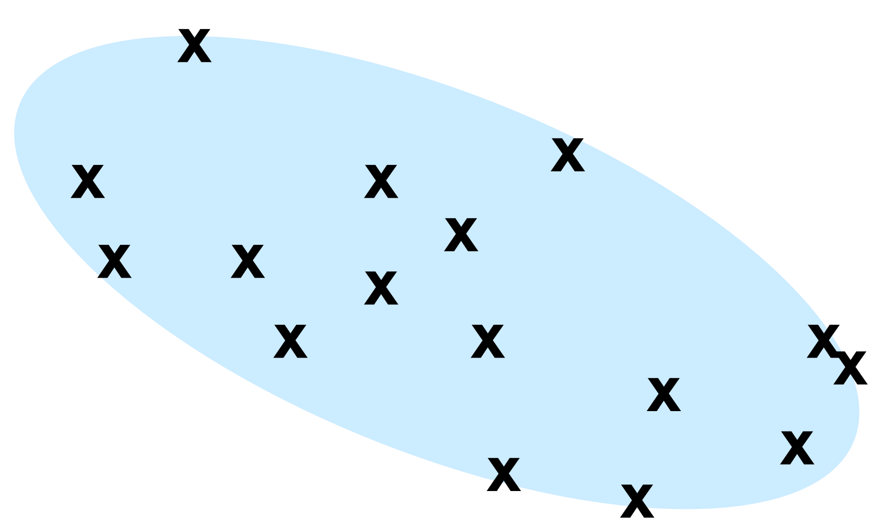
MLE for a multivariate Gaussian distribution

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



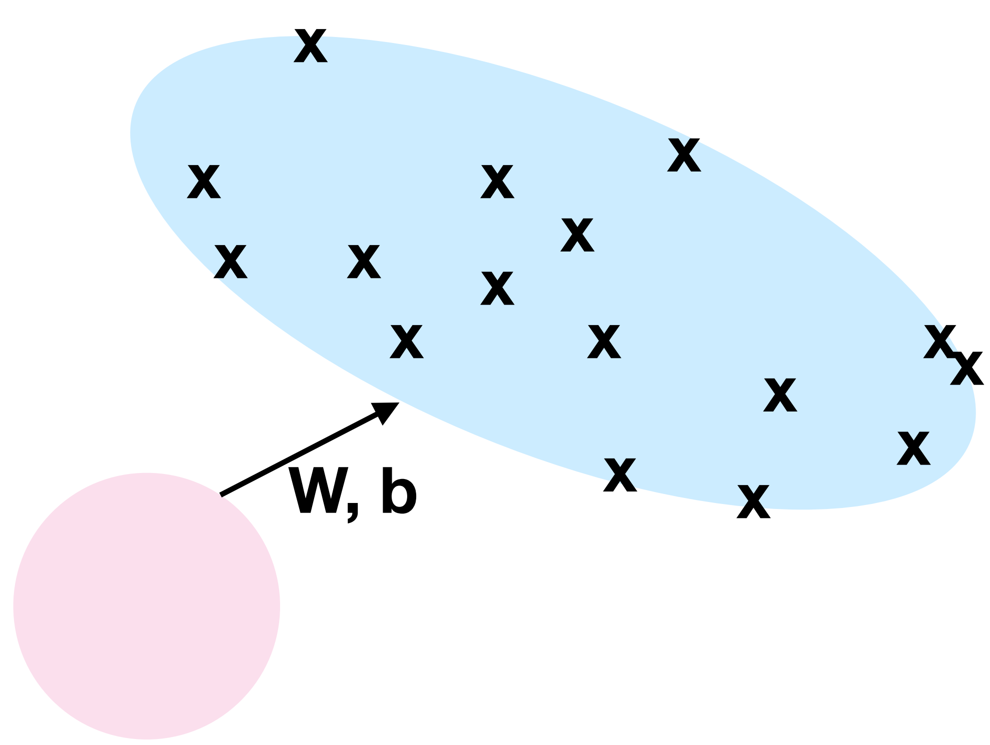
MLE for a multivariate Gaussian distribution

- We can use differential calculus to maximize the (log) likelihood w.r.t. the parameters $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$.



MLE for a multivariate Gaussian distribution

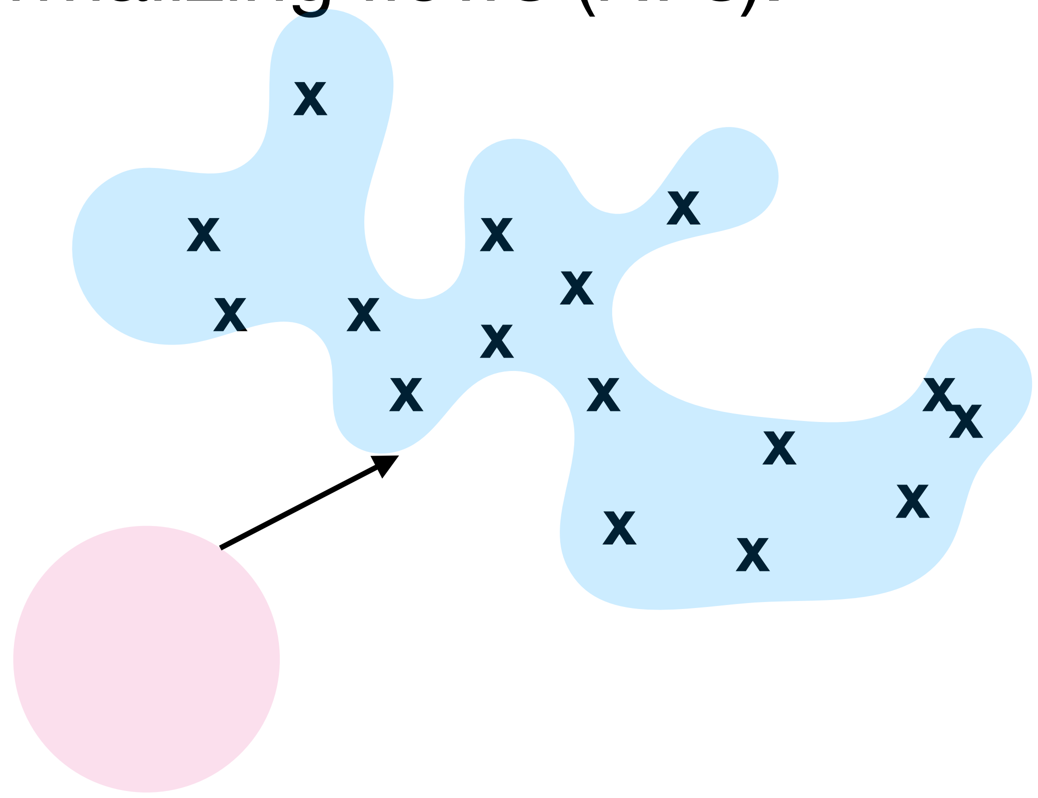
- With PCA as an LVM, we instead identify the matrix **W** and vector **b** that transforms a low-dimensional, 0-mean, and **I**-covariance Gaussian into a high-dimensional Gaussian that fits the data $\mathcal{D}$.



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).



≥3-layer NNs

Limitations of linear NNs

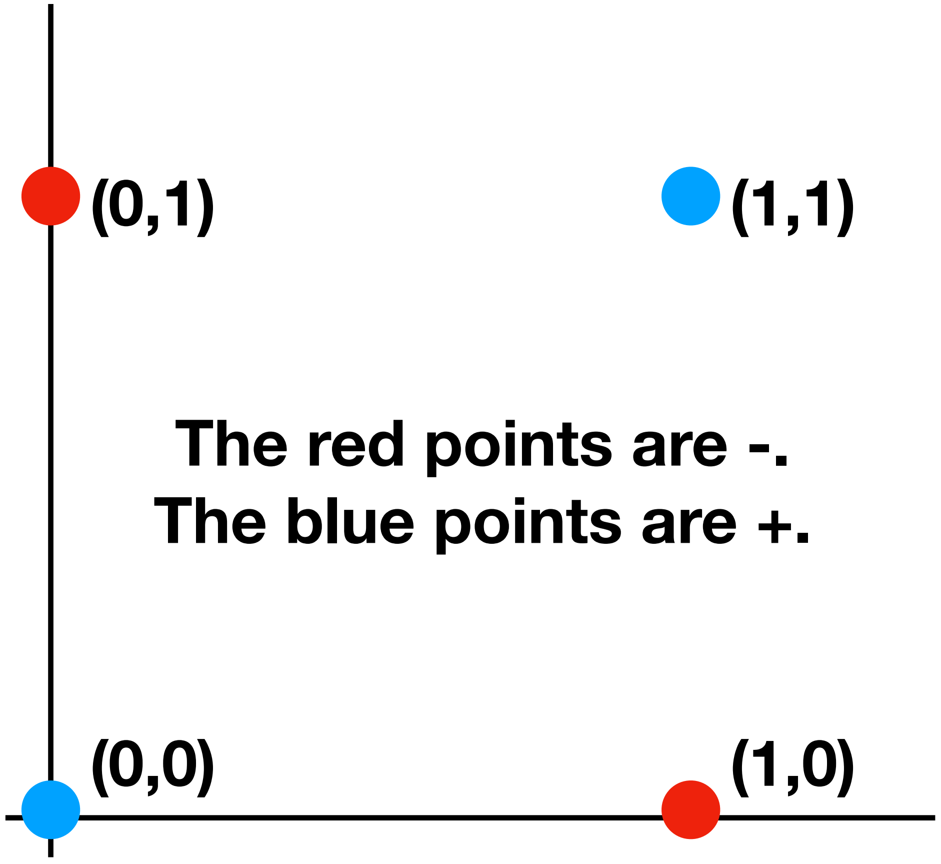
- Linear 2-layer NNs are very convenient to optimize.
- But they are very limited in the functions they can represent.

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.

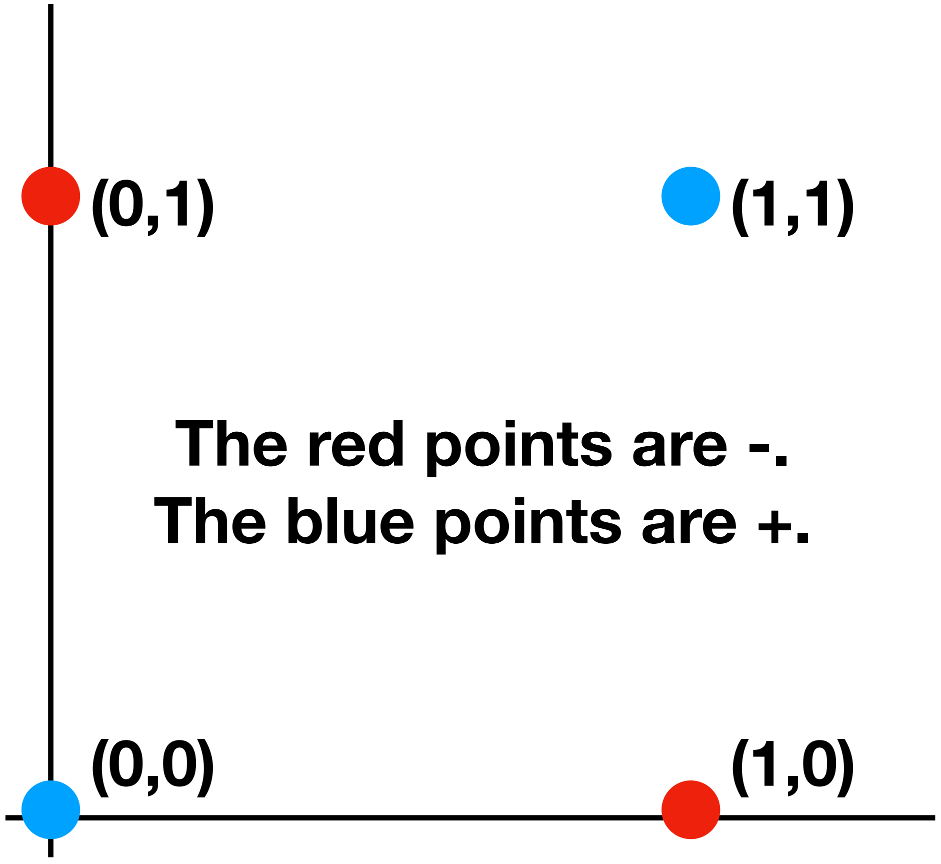
XOR problem

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



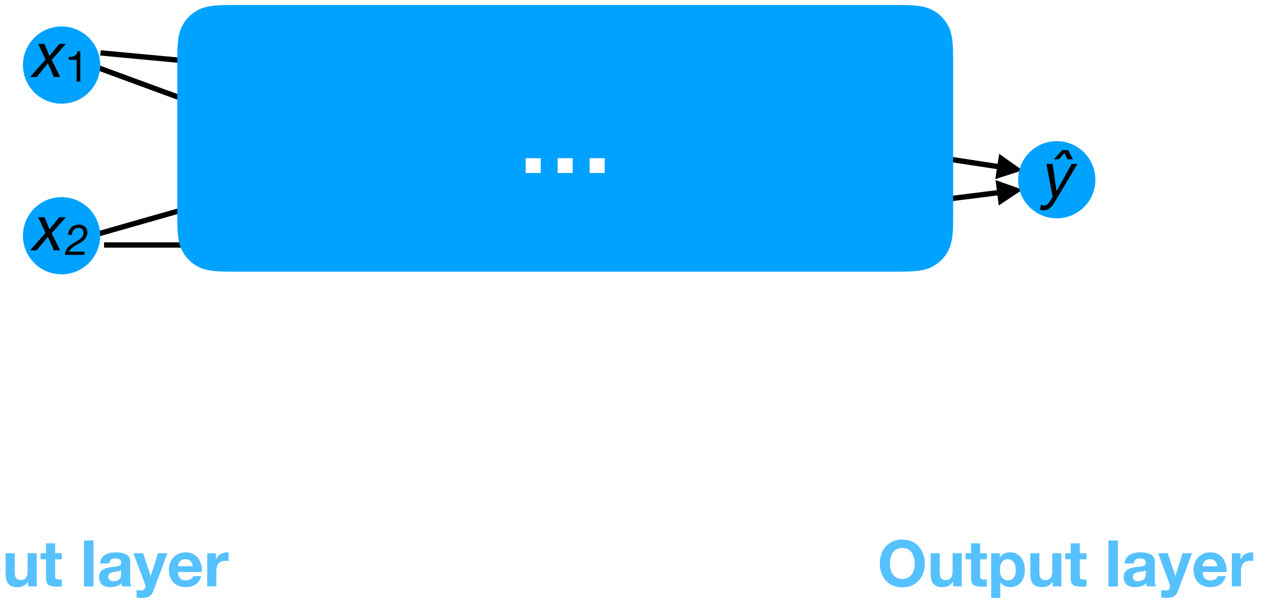
XOR problem

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



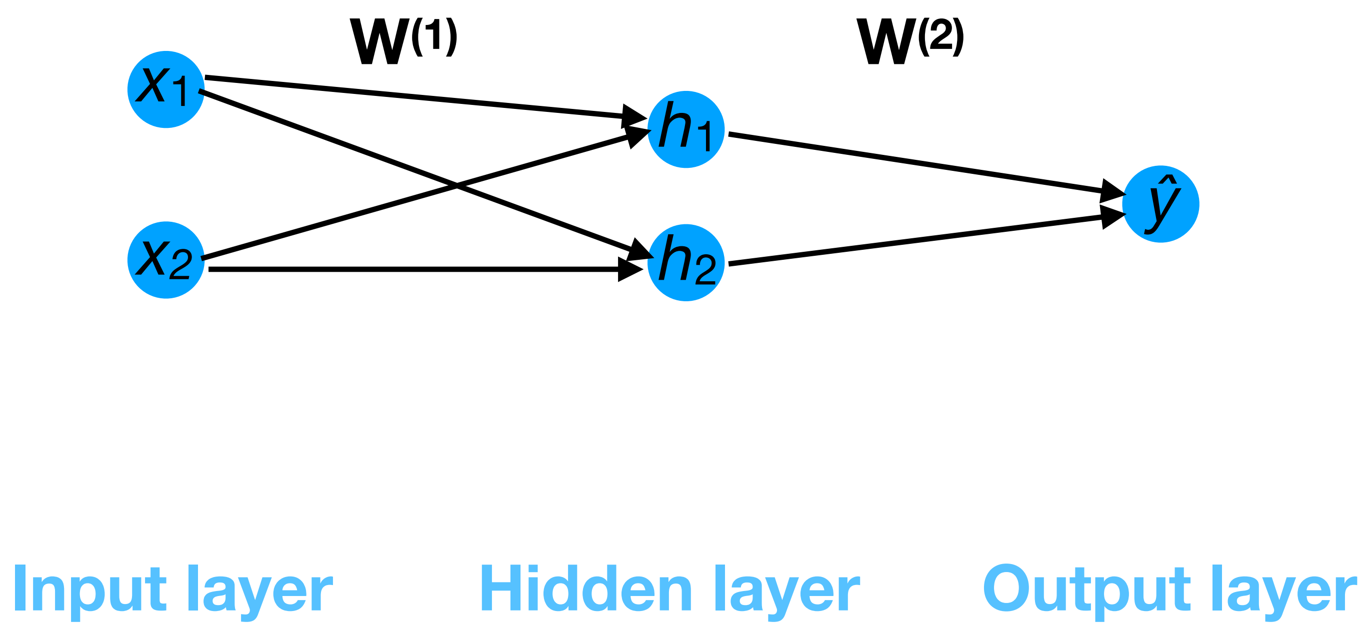
XOR problem

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



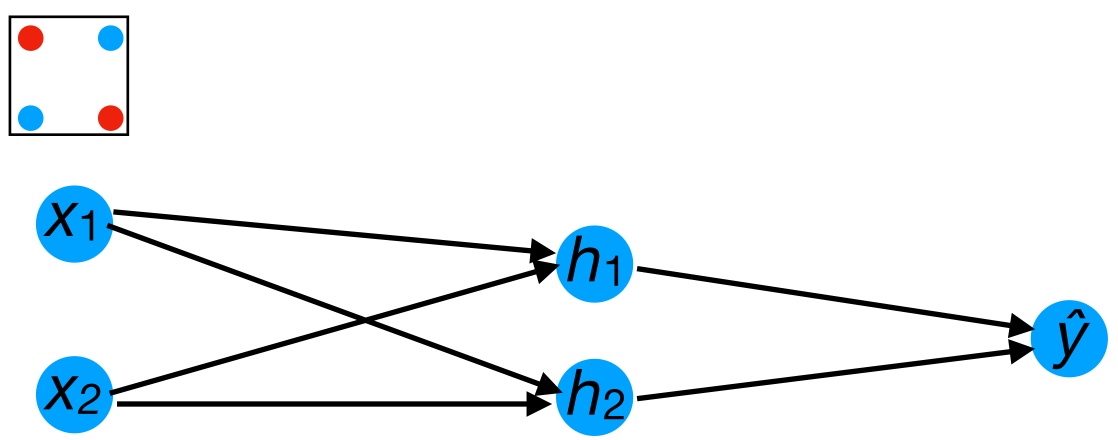
XOR problem

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



XOR problem

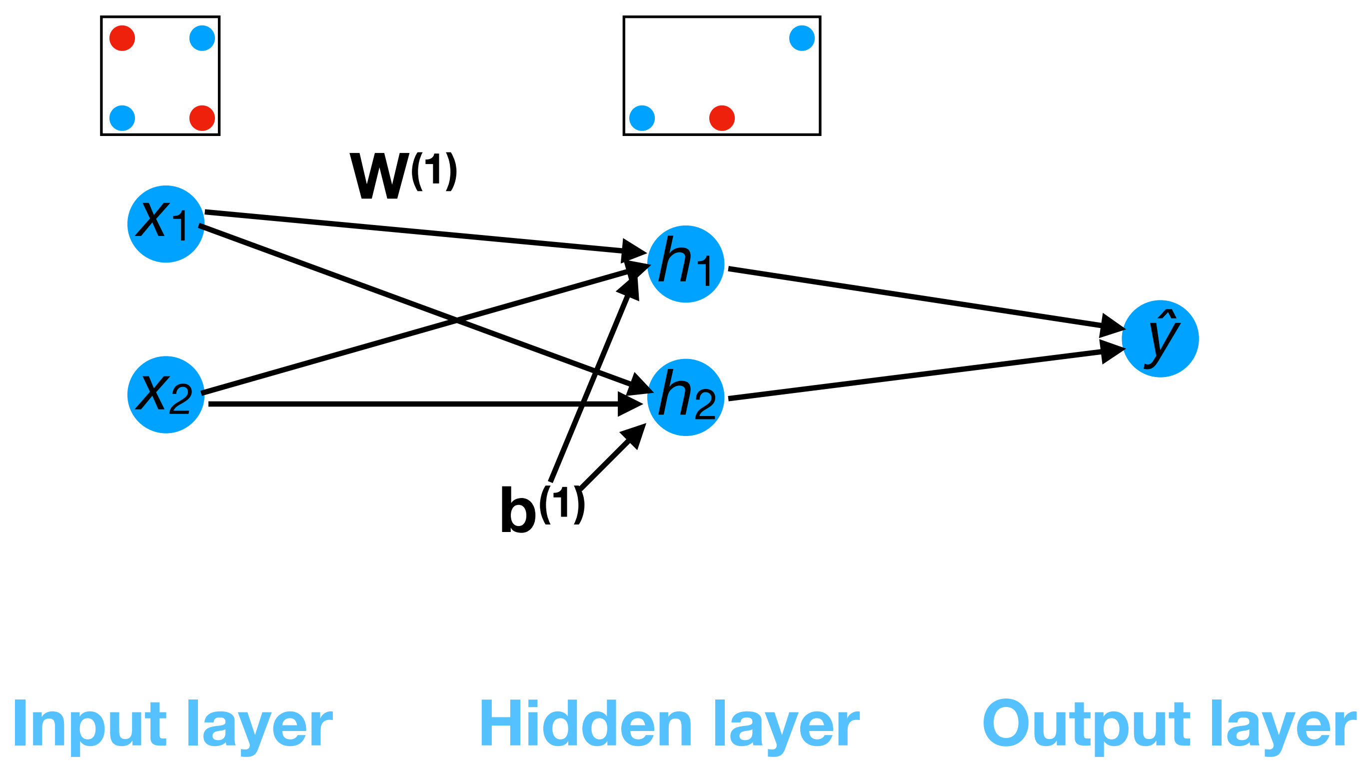
- Here’s how a 3-layer NN can solve it:



Input layer Hidden layer Output layer

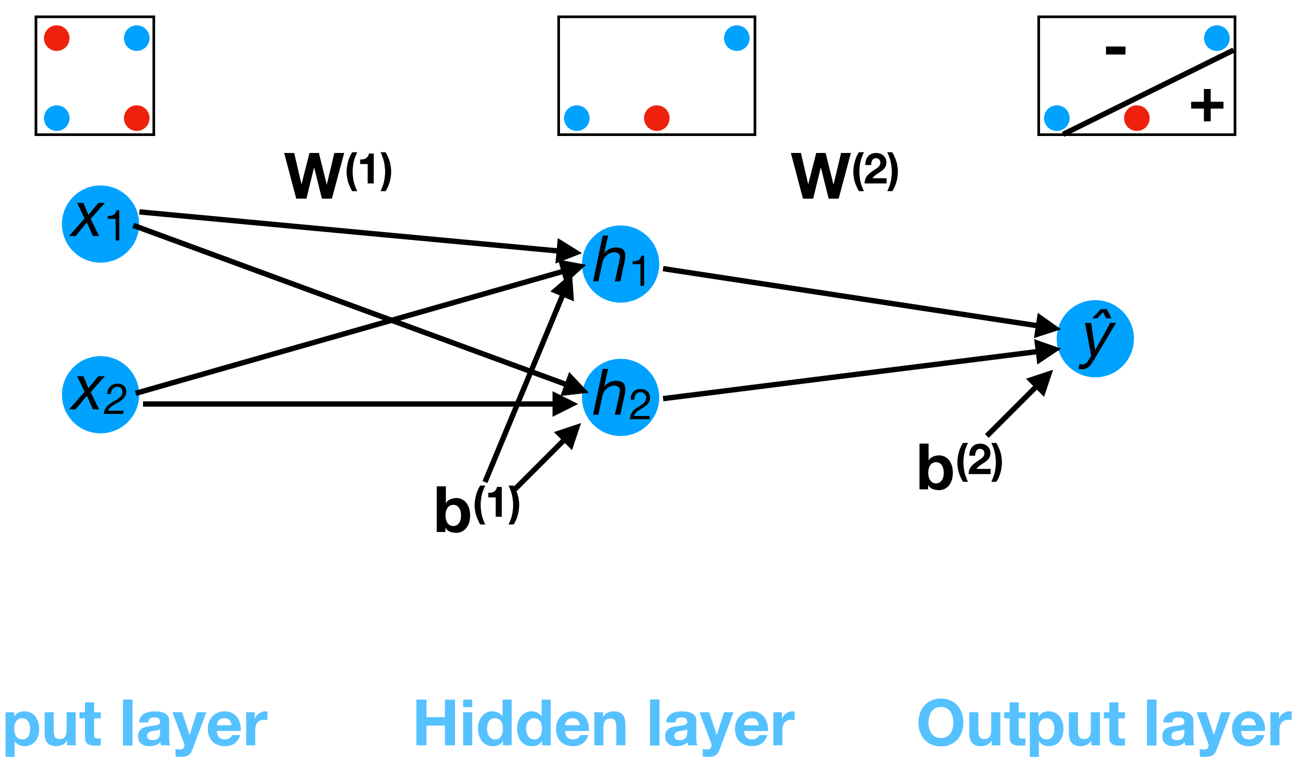
XOR problem

- 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.



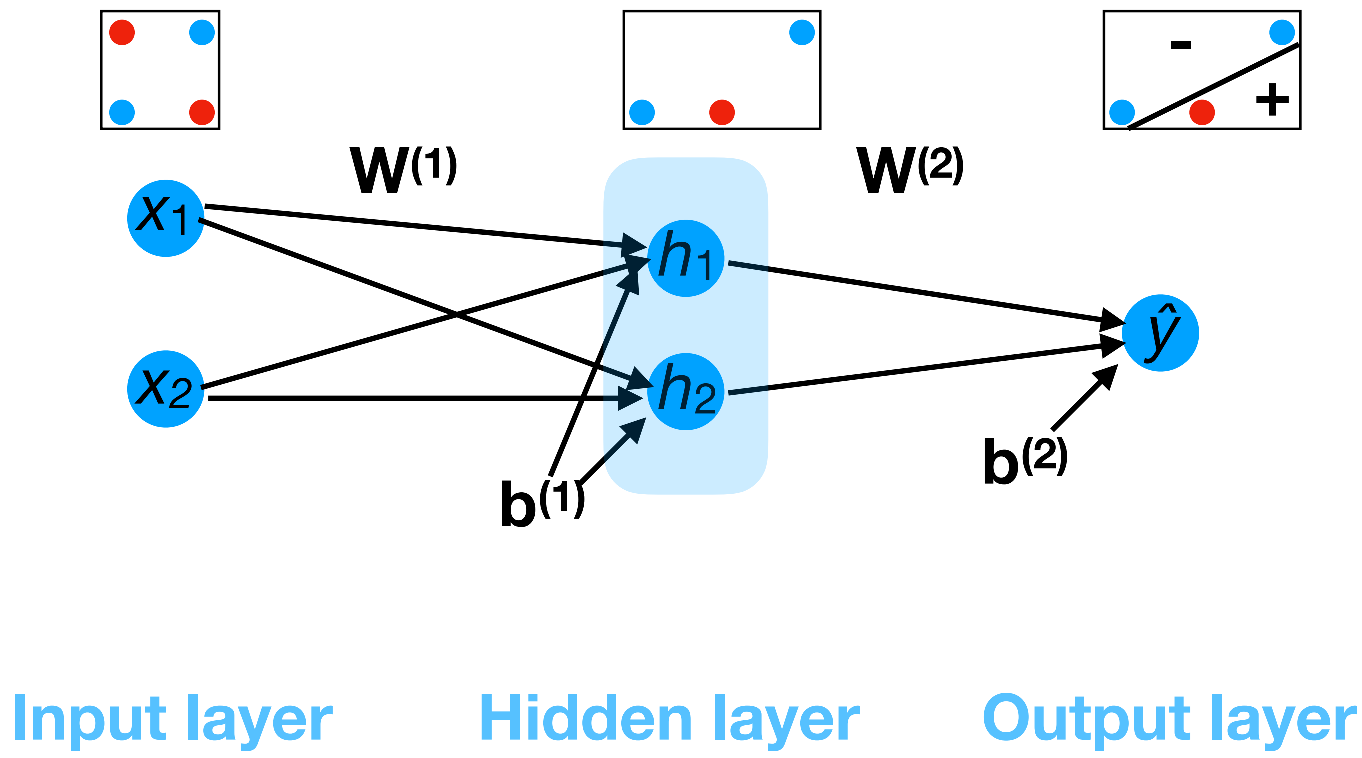
XOR problem

- 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.



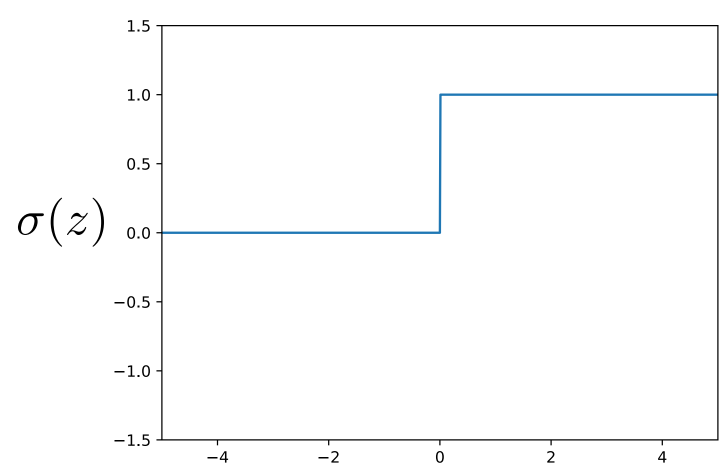
XOR problem

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



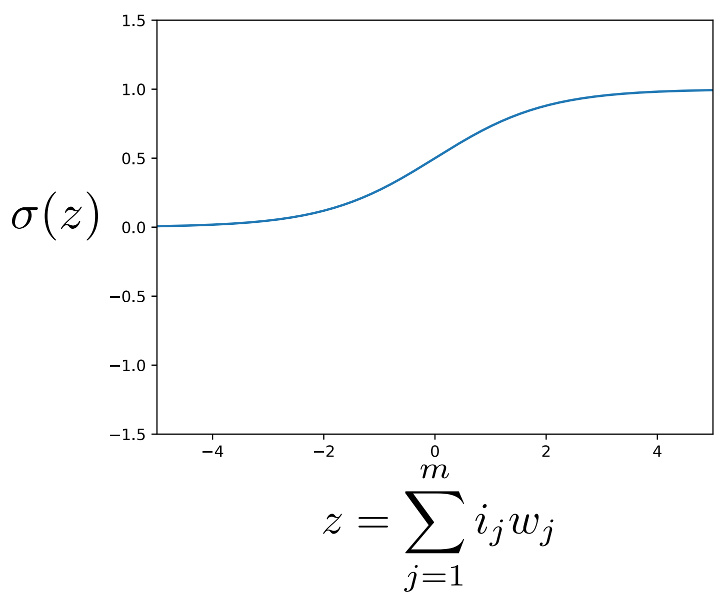
$$z = \sum_{j=1}^{m} i_j w_j$$

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

Activation functions

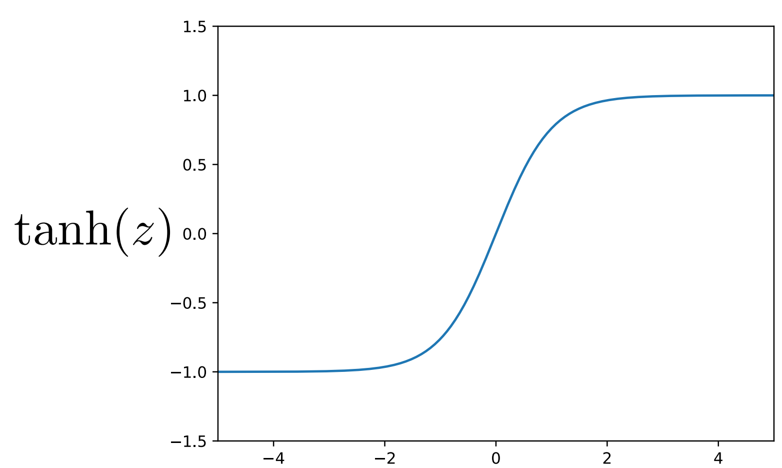
- Because the Heaviside step function is non-differentiable at 0, it was largely replaced by either a **logistic sigmoid**:

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



Activation functions

- …or sometimes with hyperbolic tangent:

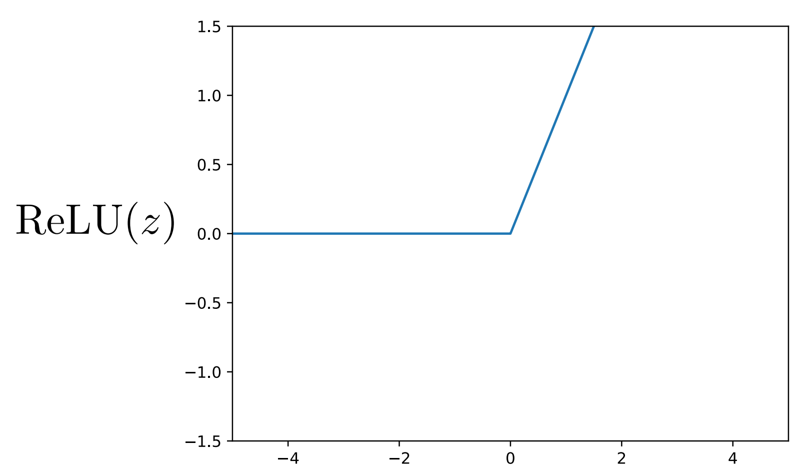


$$z = \sum_{j=1}^{m} i_j w_j$$

Activation functions

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

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



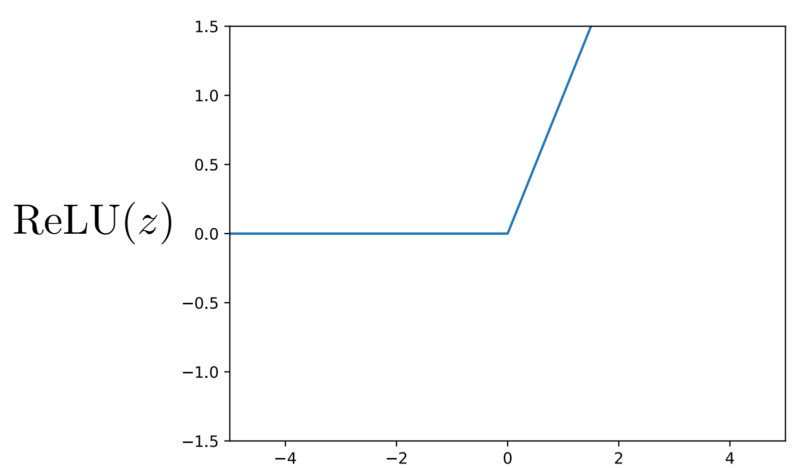
$$z = \sum_{j=1}^{m} i_j w_j$$

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**:

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

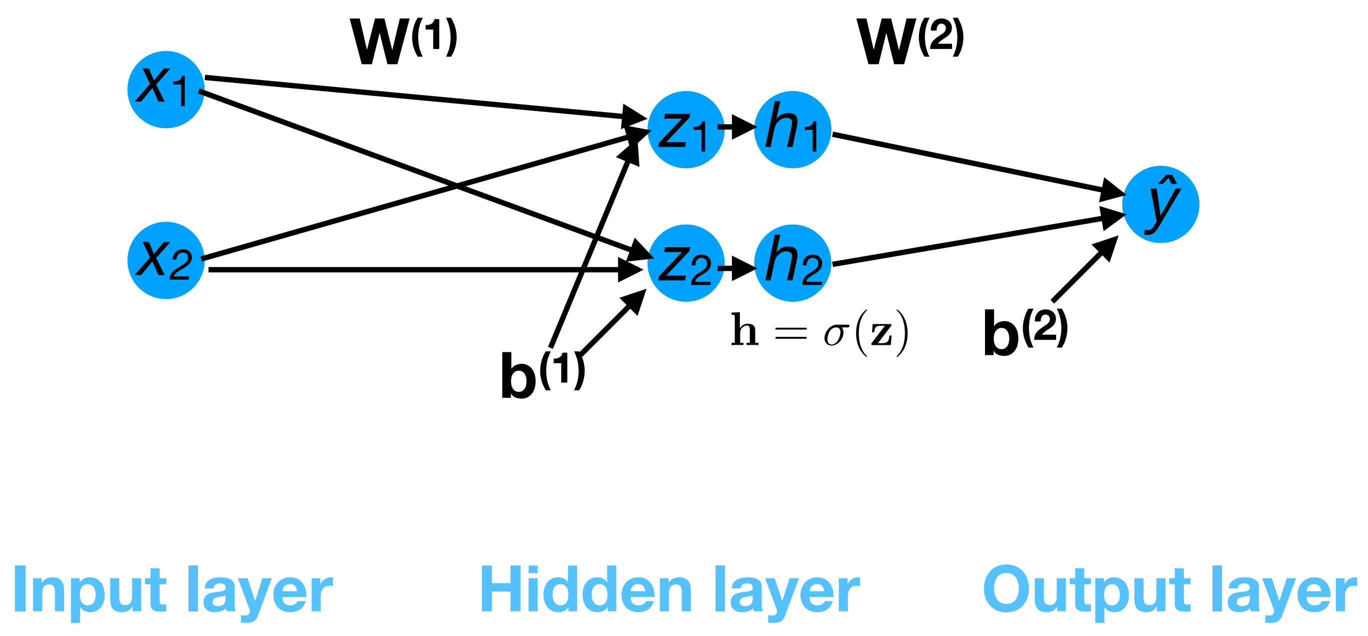


$$z = \sum_{j=1}^{m} i_j w_j$$

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

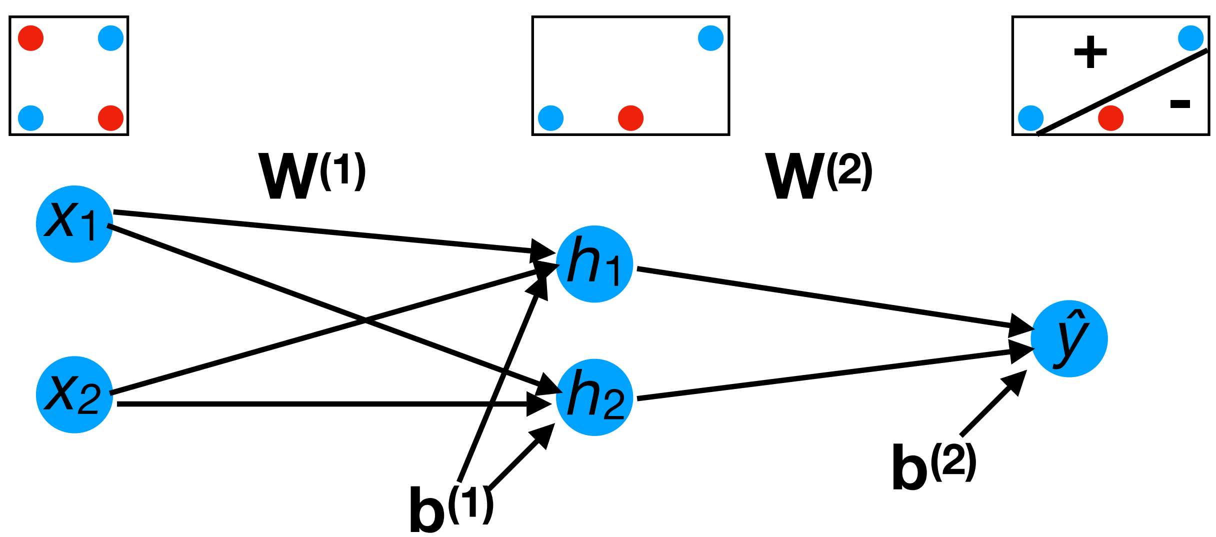
XOR problem

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

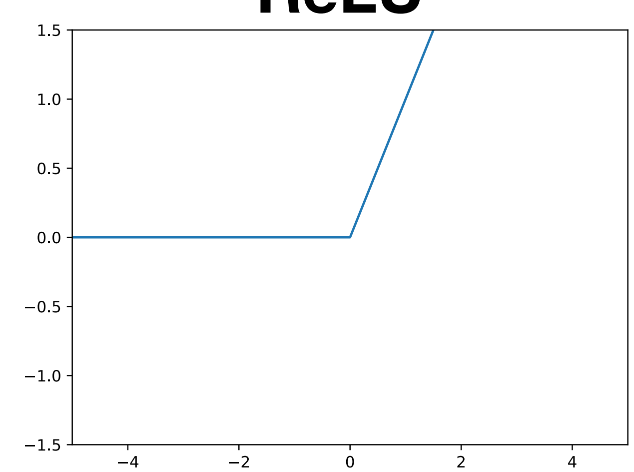


XOR problem

- Using ReLU in the hidden layer:
  - **W**$^{(1)}$, **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, **W**$^{(2)}$, **b**$^{(2)}$ can easily distinguish the two classes.



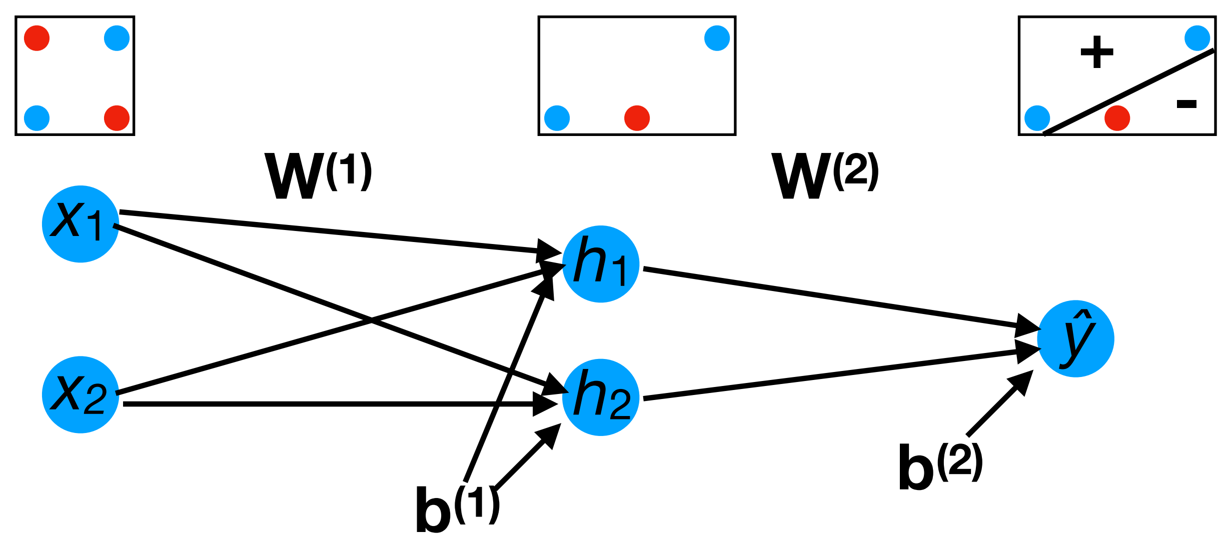
ReLU

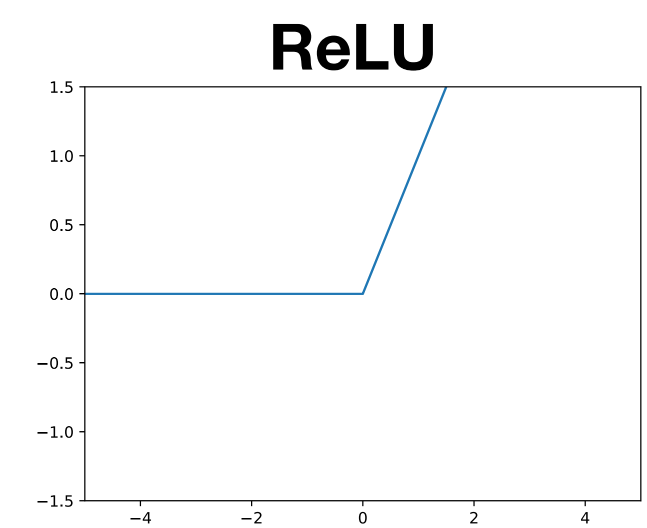


What values for W$^{(1)}$, b$^{(1)}$ will make the 4 data points linearly separable?
(Hint: set b = [-1 -1]^T; W contains only 1s and 2s.)

XOR problem

- Using ReLU in the hidden layer:
  - **W**$^{(1)}$, **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, **W**$^{(2)}$, **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.)

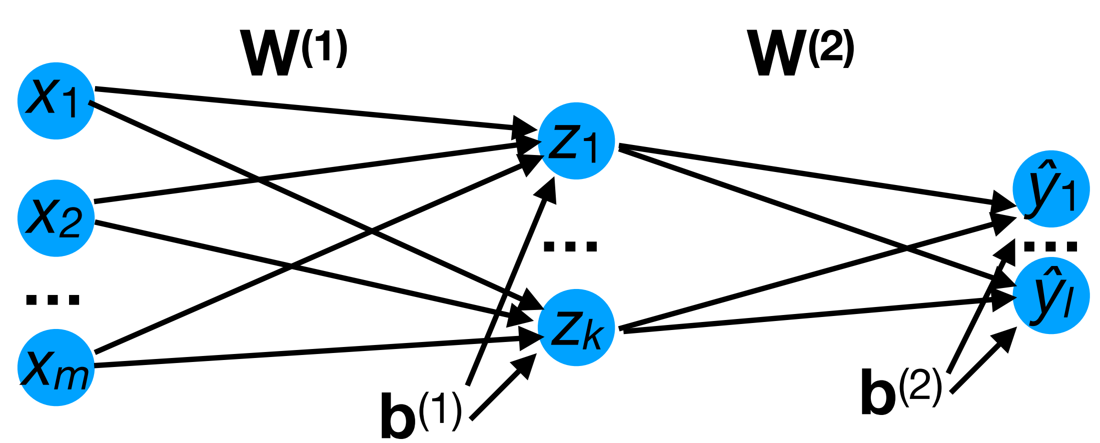
XOR problem

- 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:

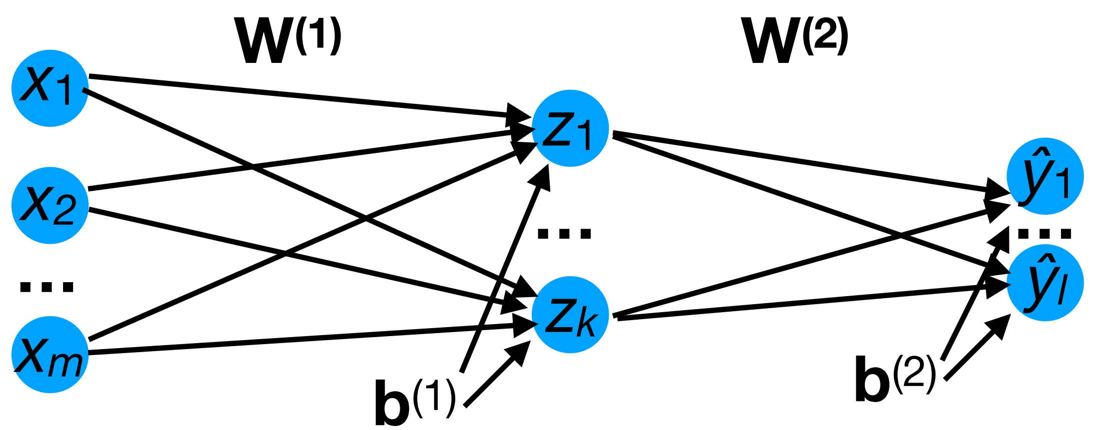
$$g(\mathbf{x}) \quad = \quad \mathbf{W}^{(2)}\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}$$



Crucial role of non-linearity

- Then we can simplify *g* to be:

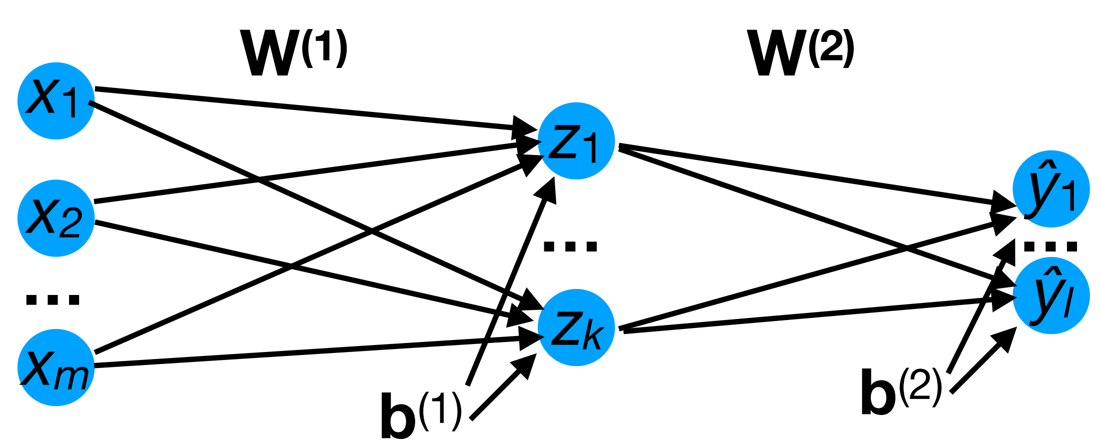
$$g(\mathbf{x}) \quad = \quad \mathbf{W}^{(2)}\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}$$



Crucial role of non-linearity

- Then we can simplify g to be:

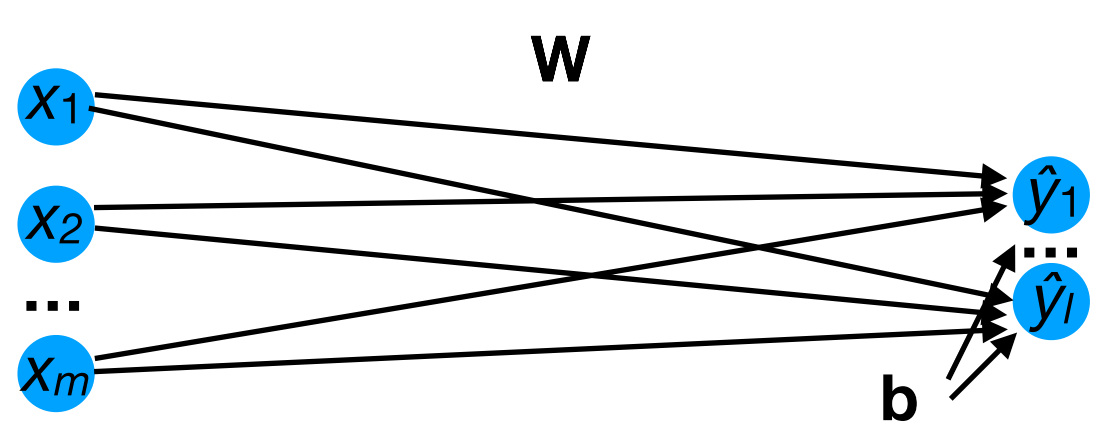
$$\begin{aligned} g(\mathbf{x}) &= \mathbf{W}^{(2)}\left(\mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)}\right)+\mathbf{b}^{(2)} \\ &= \left(\mathbf{W}^{(2)}\mathbf{W}^{(1)}\right)\mathbf{x}+\mathbf{W}^{(2)}\mathbf{b}^{(1)}+\mathbf{b}^{(2)} \end{aligned}$$



Crucial role of non-linearity

- Then we can simplify g to be:

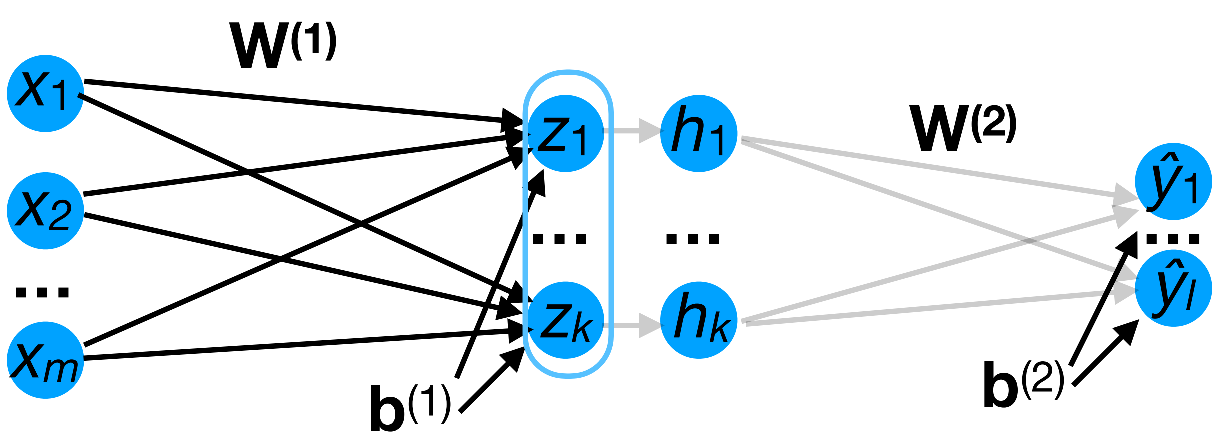
$$
\begin{aligned}
g(\mathbf{x}) &= \mathbf{W}^{(2)}\left(\mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)}\right)+\mathbf{b}^{(2)} \\
&= \left(\mathbf{W}^{(2)}\mathbf{W}^{(1)}\right)\mathbf{x}+\mathbf{W}^{(2)}\mathbf{b}^{(1)}+\mathbf{b}^{(2)} \\
&= \mathbf{W}\mathbf{x}+\mathbf{b}
\end{aligned}
$$



Feed-forward neural networks

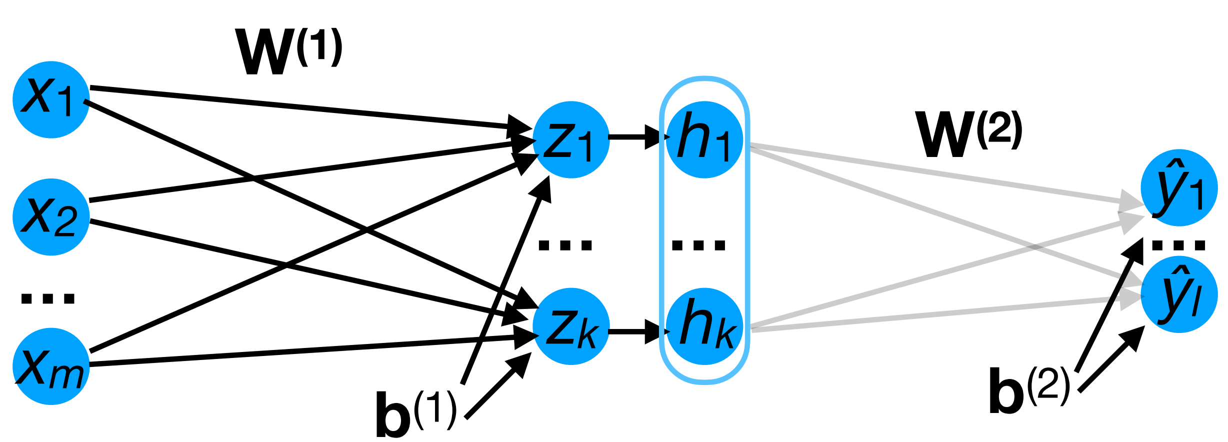
Forward propagation

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



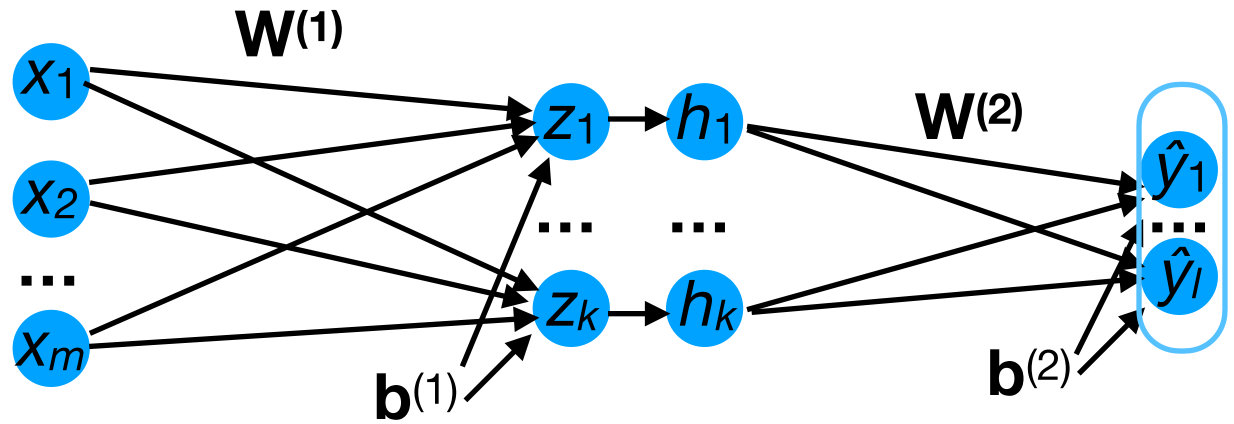
Forward propagation

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



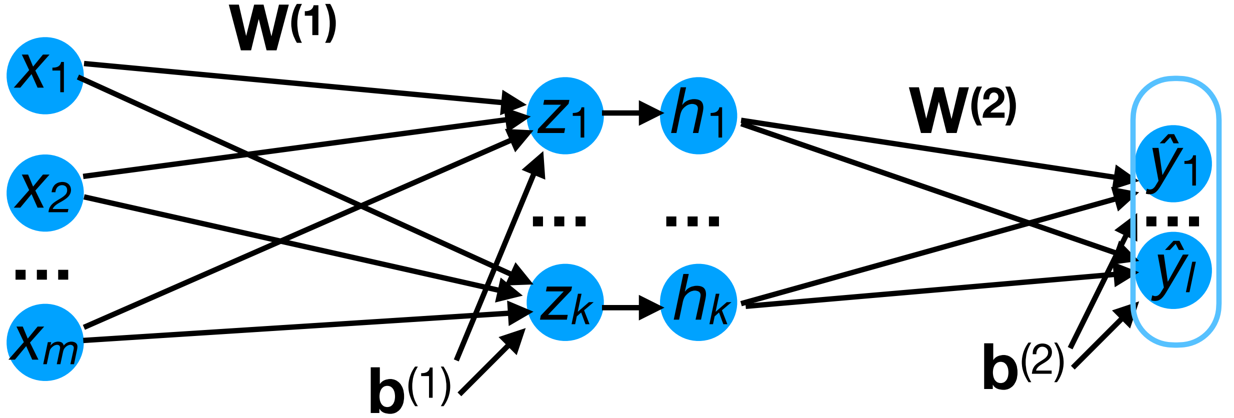
Forward propagation

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



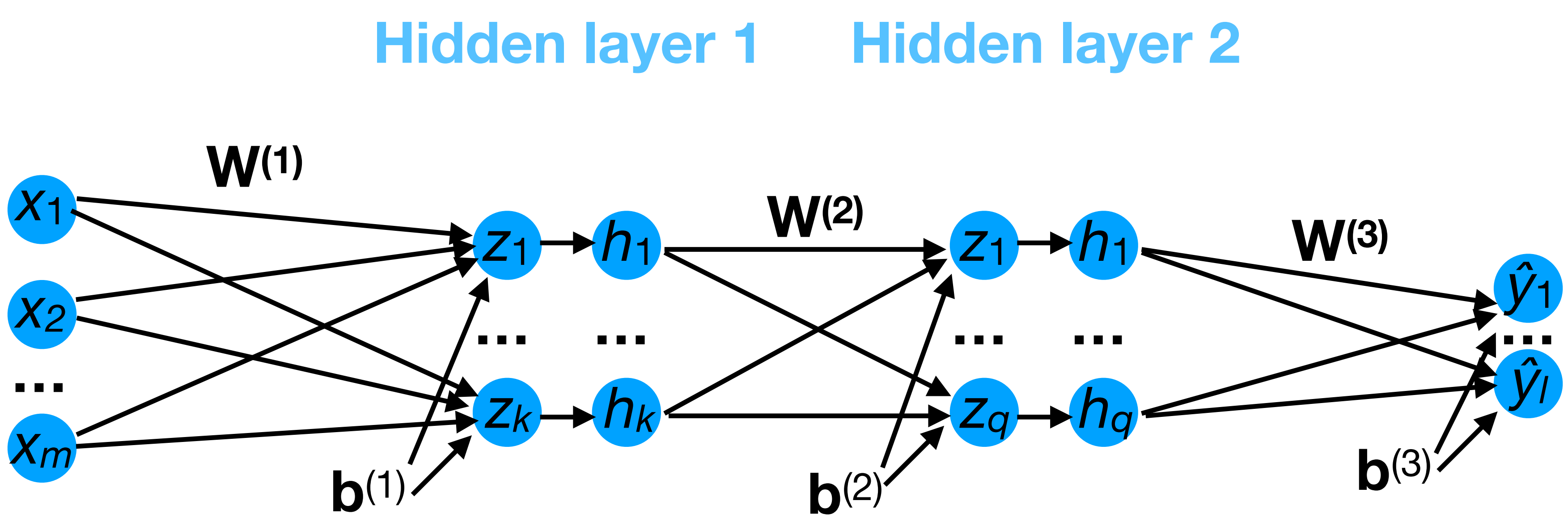
Forward propagation

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



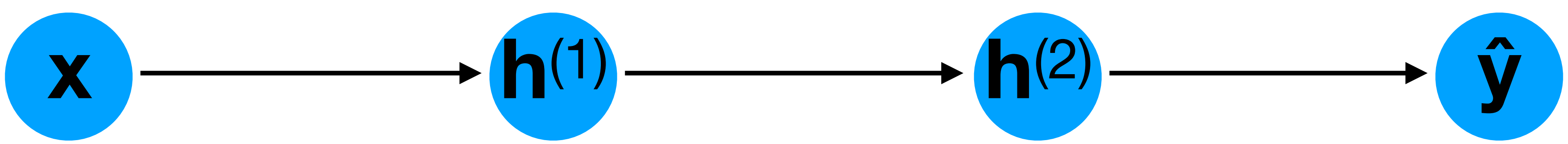
Forward propagation

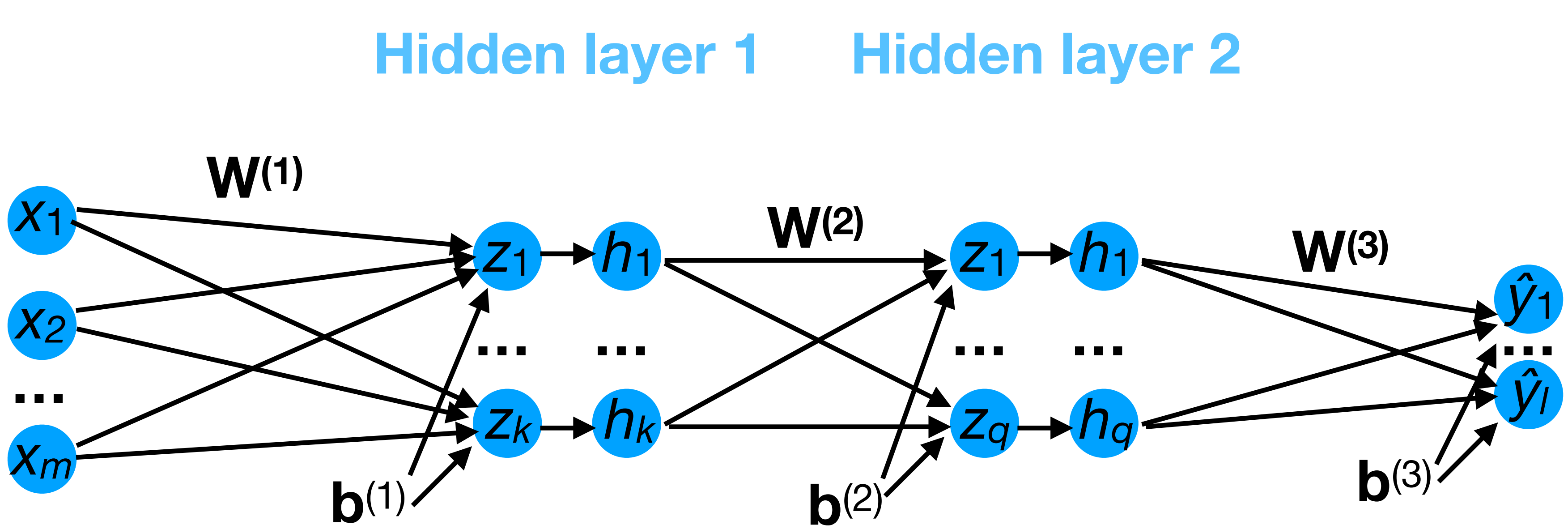
- This process is known as **forward propagation**.
  - It can be applied a NN of arbitrary depth; in the NN below, it would produce $\mathbf{z}^{(1)}$, $\mathbf{h}^{(1)}$, $\mathbf{z}^{(2)}$, $\mathbf{h}^{(2)}$, and $\hat{\mathbf{y}}$.



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

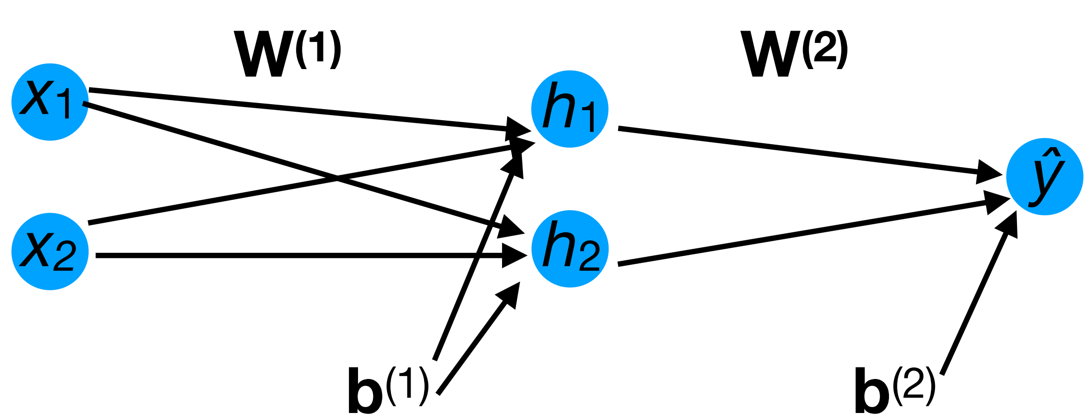
- Here is how we can conduct gradient descent for the XOR problem…

- Let's first define:

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

- 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:

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

- Then we can define *g* so that:

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

Gradient descent: XOR problem

- Here is how we can conduct gradient descent for the XOR problem…

- Let's first define:

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

- Then we can define *g* so that:

$$\begin{aligned} \hat{y} = g\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) &= \mathbf{W}^{(2)}\sigma\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)} \\ &= \begin{bmatrix} w_5 \\ w_6 \end{bmatrix}^{\mathsf{T}} \sigma\left(\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\right) + b_3 \\ &= w_5\sigma(w_1x_1 + w_2x_2 + b_1) + w_6\sigma(w_3x_1 + w_4x_2 + b_2) + b_3 \end{aligned}$$

Gradient descent: XOR problem

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

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

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 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.:

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

Gradient descent: XOR problem

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

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

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:

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

Gradient descent: XOR problem

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

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

Gradient descent: XOR problem

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

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

Gradient descent: XOR problem

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

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

Gradient descent: XOR problem

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

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

where:

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

Gradient descent: XOR problem

- Hence:

$$\frac{\partial \hat{y}}{\partial w_1} = \begin{cases} 0 & \text{if} \quad w_1 x_1 + w_2 x_2 + b_1 \leq 0 \\ w_5 x_1 & \text{if} \quad w_1 x_1 + w_2 x_2 + b_1 > 0 \end{cases}$$

since:

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

Exercise

- Find:

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

Exercise

- Find:

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

Gradient descent: XOR problem

- We also need to derive the other partial derivatives:

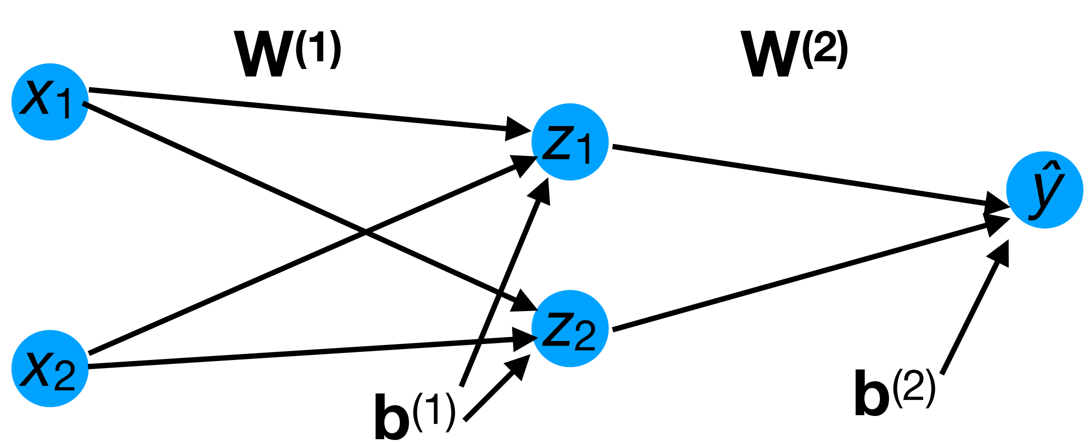
$$\frac{\partial \hat{y}}{\partial w_1}, \ldots \frac{\partial \hat{y}}{\partial w_6}, \frac{\partial \hat{y}}{\partial b_1}, \ldots \frac{\partial \hat{y}}{\partial b_3}$$

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

$$\frac{\partial f_{\mathrm{MSE}}}{\partial w_1} = \frac{\partial f_{\mathrm{MSE}}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_1} \qquad \frac{\partial f_{\mathrm{MSE}}}{\partial b_1} = \frac{\partial f_{\mathrm{MSE}}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_1}$$

... **...**

$$\frac{\partial f_{\mathrm{MSE}}}{\partial w_6} = \frac{\partial f_{\mathrm{MSE}}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_6} \qquad \frac{\partial f_{\mathrm{MSE}}}{\partial b_3} = \frac{\partial f_{\mathrm{MSE}}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_3}$$



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: $w_1 \leftarrow w_1 - \epsilon \frac{\partial f_{\mathrm{MSE}}}{\partial w_1}$
        …
        Update w_6: $w_6 \leftarrow w_6 - \epsilon \frac{\partial f_{\mathrm{MSE}}}{\partial w_6}$

        Update b_1: $b_1 \leftarrow b_1 - \epsilon \frac{\partial f_{\mathrm{MSE}}}{\partial b_1}$
        …
        Update b_3: $b_3 \leftarrow b_3 - \epsilon \frac{\partial f_{\mathrm{MSE}}}{\partial b_3}$

Gradient descent: (any 3-layer NN)

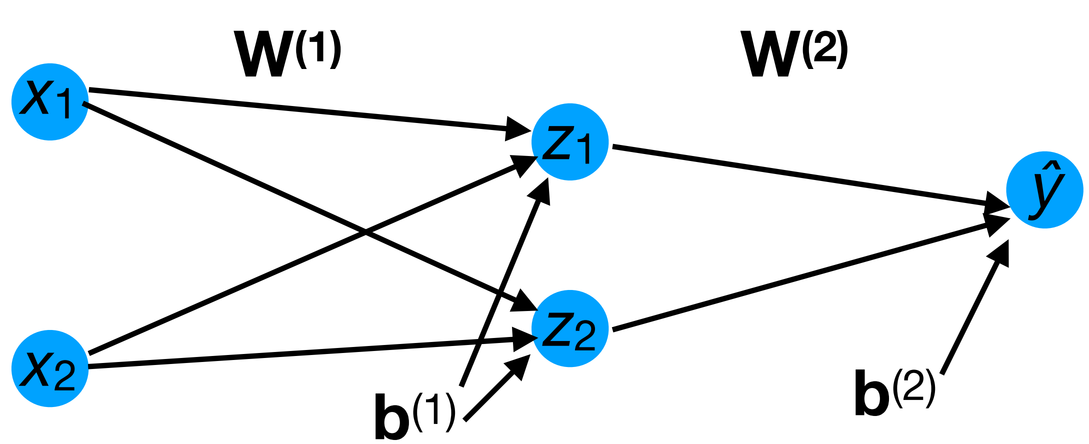
- It is more compact and efficient to represent and compute these gradients more abstractly:

$$\nabla_{\mathbf{W}^{(1)}} f_{\text{MSE}}$$
$$\nabla_{\mathbf{W}^{(2)}} f_{\text{MSE}}$$
$$\nabla_{\mathbf{b}^{(1)}} f_{\text{MSE}}$$
$$\nabla_{\mathbf{b}^{(2)}} f_{\text{MSE}}$$

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 **W**(1): $\mathbf{W}^{(1)} \leftarrow \mathbf{W}^{(1)} - \epsilon \nabla_{\mathbf{W}^{(1)}} f_{\mathrm{MSE}}$

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

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

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

Deriving the gradient terms

- We can largely *automate* the process of computing each gradient term $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$, …, $\mathbf{W}^{(d)}$ (for d layers).

- This procedure is enabled by the **chain rule of multivariate calculus**…

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.:

$$f\left(\left[\begin{array}{c} x_1 \\ x_2 \end{array}\right]\right) = \left[\begin{array}{c} 2x_1 + x_2 \\ \pi x_2 \\ -x_1/x_2 \end{array}\right]$$

- 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.:

$$f\left(\left[\begin{array}{c} x_1 \\ x_2 \end{array}\right]\right) = \left[\begin{array}{c} 2x_1 + x_2 \\ \pi x_2 \\ -x_1/x_2 \end{array}\right]$$

- 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 $g : \mathbb{R}^k \rightarrow \mathbb{R}^m$.
- 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 $g : \mathbb{R}^k \rightarrow \mathbb{R}^m$.
- 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 $f : \mathbb{R}^m \to \mathbb{R}^n$ and $g : \mathbb{R}^k \to \mathbb{R}^m$.
- Then $\frac{\partial (f \circ g)}{\partial \mathbf{x}} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial \mathbf{x}}$
- Note that the order matters!, i.e.:

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

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:

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

- 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

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

- Substitution:

$$f \circ g\left(\left[\begin{array}{c} x_1 \\ x_2 \end{array}\right]\right) \quad = \quad f\left(g\left(\left[\begin{array}{c} x_1 \\ x_2 \end{array}\right]\right)\right)$$

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:

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

Chain rule of multivariate calculus: example

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

- Substitution:

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

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:

$$\frac{\partial f}{\partial g} = \begin{bmatrix} \boxed{?} \end{bmatrix}$$

1 x 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:

$$\frac{\partial f}{\partial g} = \begin{bmatrix} 1 & -2 & \frac{1}{4} \end{bmatrix}$$ **1 x 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:

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

1 x 3

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

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:

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

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

$$\Longrightarrow \frac{\partial f \circ g}{\partial \mathbf{x}} =$$ **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:

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

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

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

$$= \begin{bmatrix} \frac{3}{4} & 0 \end{bmatrix}$$

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.

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{x}_1} \\ \vdots \\ \frac{\partial f}{\partial \mathbf{x}_m} \end{bmatrix}$$

Gradient

$$\frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{x}_1} & \dots & \frac{\partial f}{\partial \mathbf{x}_m} \end{bmatrix}$$

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.

$$\frac{\partial f}{\partial \mathbf{x}} = \left[\begin{array}{cc} 2a & -b/y^2 \end{array} \right]$$

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.

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

Vectorizing a matrix

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

$$f(\mathbf{x}; \mathbf{W}) = \left[\begin{array}{cc} w_1 & w_2 \\ w_3 & w_4 \end{array} \right] \left[\begin{array}{c} x_1 \\ x_2 \end{array} \right]$$

- 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.:

$$f(\mathbf{x}; \mathbf{W}) = \left[\begin{array}{cc} w_1 & w_2 \\ w_3 & w_4 \end{array} \right] \left[\begin{array}{c} x_1 \\ x_2 \end{array} \right] = \left[\begin{array}{c} w_1 x_1 + w_2 x_2 \\ w_3 x_1 + w_4 x_2 \end{array} \right]$$

- 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 **W** to be a vector…

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.,

$$\text{vec}[\mathbf{W}] = \left[\begin{array}{ccccc} W_{11} & \ldots W_{1m} & \ldots & W_{n1} & \ldots & W_{nm} \end{array} \right]$$

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.,

$$\text{vec}[\mathbf{W}] = \left[\begin{array}{ccccc} W_{11} & \ldots W_{1m} & \ldots & W_{n1} & \ldots & W_{nm} \end{array} \right]$$

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.,

$$\mathrm{vec}[\mathbf{W}] = \left[\begin{array}{ccccc} W_{11} & \ldots W_{1m} & \ldots & W_{n1} & \ldots & W_{nm} \end{array} \right]$$

Then we can compute:

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

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

Matrix derivative of a vector-valued function

$$\frac{\partial J}{\partial \mathbf{z}} \doteq \mathbf{p}^\top$$

Matrix derivative of a vector-valued function

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

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 z depend on the *first row* of W?

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

Matrix derivative of a vector-valued function

$$\frac{\partial J}{\partial \mathbf{z}} \doteq \mathbf{p}^\top$$

$$\frac{\partial \mathbf{z}}{\partial \mathrm{vec}[\mathbf{W}]} = \begin{bmatrix} \frac{\partial z_1}{\partial W_{11}} & \cdots & \frac{\partial z_1}{\partial W_{1m}} & \cdots & \boxed{\frac{\partial z_1}{\partial W_{n1}} \quad \cdots \quad \frac{\partial z_1}{\partial W_{nm}}} \\ & \vdots & & \ddots & \cdots \\ \frac{\partial z_n}{\partial W_{11}} & \cdots & \frac{\partial z_n}{\partial W_{1m}} & \cdots & \frac{\partial z_n}{\partial W_{n1}} \quad \cdots \quad \frac{\partial z_n}{\partial W_{nm}} \end{bmatrix}$$

How does the *first element* of z depend on the *last row* of W?

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

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 *last element* of z depend on the *first row* of W?

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

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 *last element* of z depend on the *last row* of W?

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

Matrix derivative of a vector-valued function

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

Matrix derivative of a vector-valued function

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

Matrix derivative of a vector-valued function

$$
\begin{aligned}
\frac{\partial J}{\partial \mathbf{z}} &\doteq \mathbf{p}^\top \\
\frac{\partial \mathbf{z}}{\partial \mathrm{vec}[\mathbf{W}]} &= \begin{bmatrix} \frac{\partial z_1}{\partial W_{11}} & \dots & \frac{\partial z_1}{\partial W_{1m}} & \dots & \frac{\partial z_1}{\partial W_{n1}} & \dots & \frac{\partial z_1}{\partial W_{nm}} \\ & \vdots & & \ddots & & \dots & \\ \frac{\partial z_n}{\partial W_{11}} & \dots & \frac{\partial z_n}{\partial W_{1m}} & \dots & \frac{\partial z_n}{\partial W_{n1}} & \dots & \frac{\partial z_n}{\partial W_{nm}} \end{bmatrix} \\
&= \begin{bmatrix} x_1 & \dots & x_m & \dots & 0 & \dots & 0 \\ & \vdots & & \ddots & & \dots & \\ 0 & \dots & 0 & \dots & x_1 & \dots & x_m \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{x}^\top & \dots & \mathbf{0}^\top \\ \vdots & \ddots & \dots \\ \mathbf{0}^\top & \dots & \mathbf{x}^\top \end{bmatrix} \\
\frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathrm{vec}[\mathbf{W}]} &= \mathbf{p}^\top \begin{bmatrix} \mathbf{x}^\top & \dots & \mathbf{0}^\top \\ \vdots & \ddots & \dots \\ \mathbf{0}^\top & \dots & \mathbf{x}^\top \end{bmatrix}
\end{aligned}
$$

Matrix derivative of a vector-valued function

$$
\begin{aligned}
\frac{\partial J}{\partial \mathbf{z}} &\doteq \mathbf{p}^\top \\
\frac{\partial \mathbf{z}}{\partial \mathrm{vec}[\mathbf{W}]} &= \begin{bmatrix} \frac{\partial z_1}{\partial W_{11}} & \dots & \frac{\partial z_1}{\partial W_{1m}} & \dots & \frac{\partial z_1}{\partial W_{n1}} & \dots & \frac{\partial z_1}{\partial W_{nm}} \\ & \vdots & & \ddots & & \dots & \\ \frac{\partial z_n}{\partial W_{11}} & \dots & \frac{\partial z_n}{\partial W_{1m}} & \dots & \frac{\partial z_n}{\partial W_{n1}} & \dots & \frac{\partial z_n}{\partial W_{nm}} \end{bmatrix} \\
&= \begin{bmatrix} x_1 & \dots & x_m & \dots & 0 & \dots & 0 \\ & \vdots & & \ddots & & \dots & \\ 0 & \dots & 0 & \dots & x_1 & \dots & x_m \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{x}^\top & \dots & \mathbf{0}^\top \\ \vdots & \ddots & \dots \\ \mathbf{0}^\top & \dots & \mathbf{x}^\top \end{bmatrix} \\
\frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathrm{vec}[\mathbf{W}]} &= \mathbf{p}^\top \begin{bmatrix} \mathbf{x}^\top & \dots & \mathbf{0}^\top \\ \vdots & \ddots & \dots \\ \mathbf{0}^\top & \dots & \mathbf{x}^\top \end{bmatrix} \\
&= \begin{bmatrix} p_1 \mathbf{x}^\top & \dots p_n \mathbf{x}^\top \end{bmatrix}
\end{aligned}
$$

Matrix derivative of a vector-valued function

Now we unvectorize the result:

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

Matrix derivative of a vector-valued function

Now we unvectorize the result:

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

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**:

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