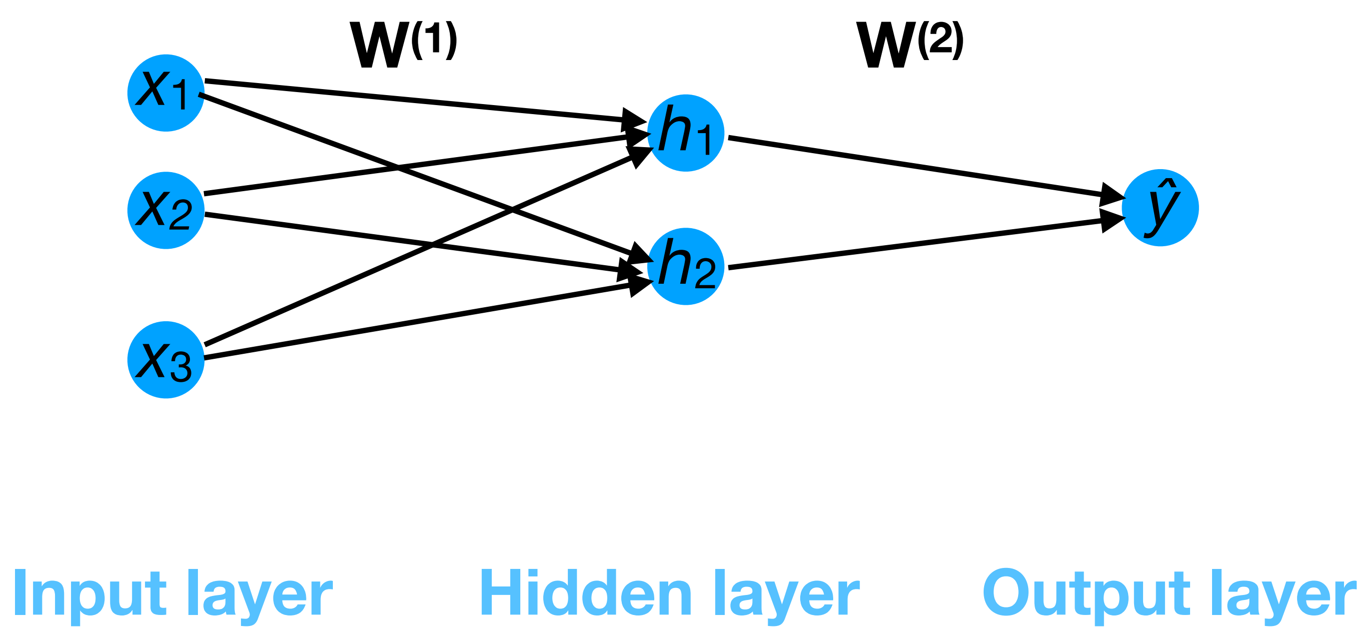
CS 453X: Class 19

Jacob Whitehill

More on neural networks

Feed-forward NN

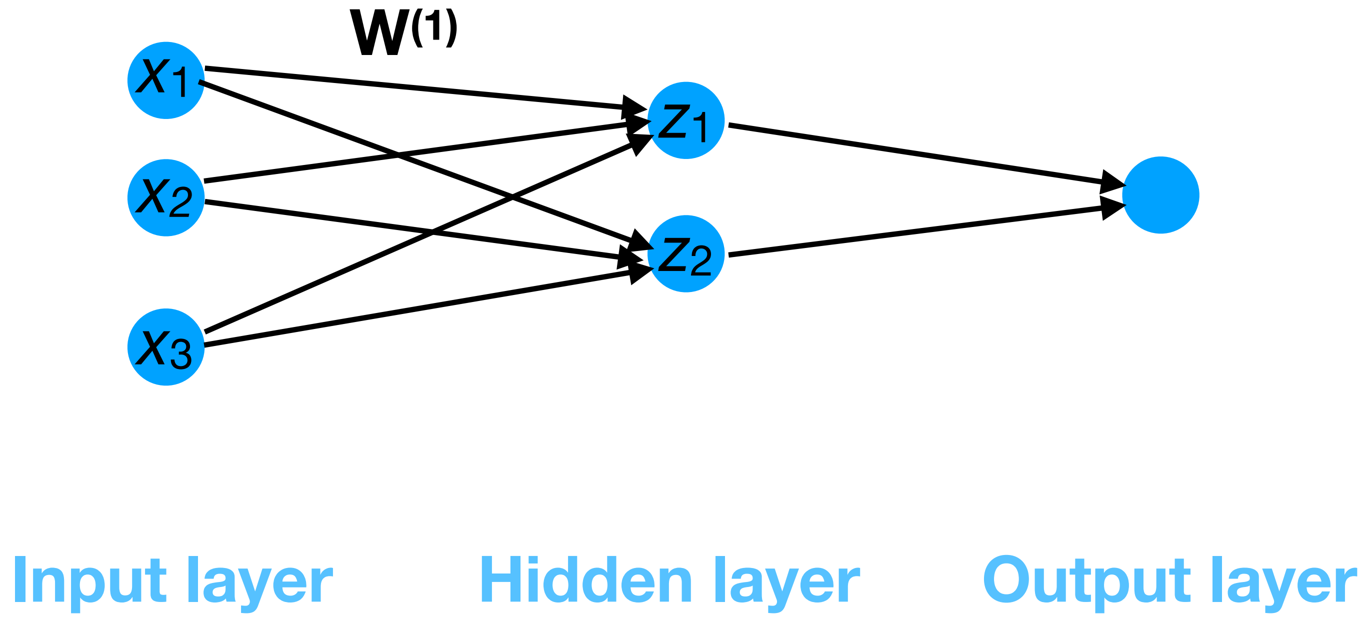
- Neural networks can have multiple neurons per layer.
- Between each adjacent pair of layers (input-hidden and hidden-output), there is a *matrix* of (synaptic) weights:



Feed-forward NN

- We can compute the pre-activation values **z** of the hidden layer as:

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

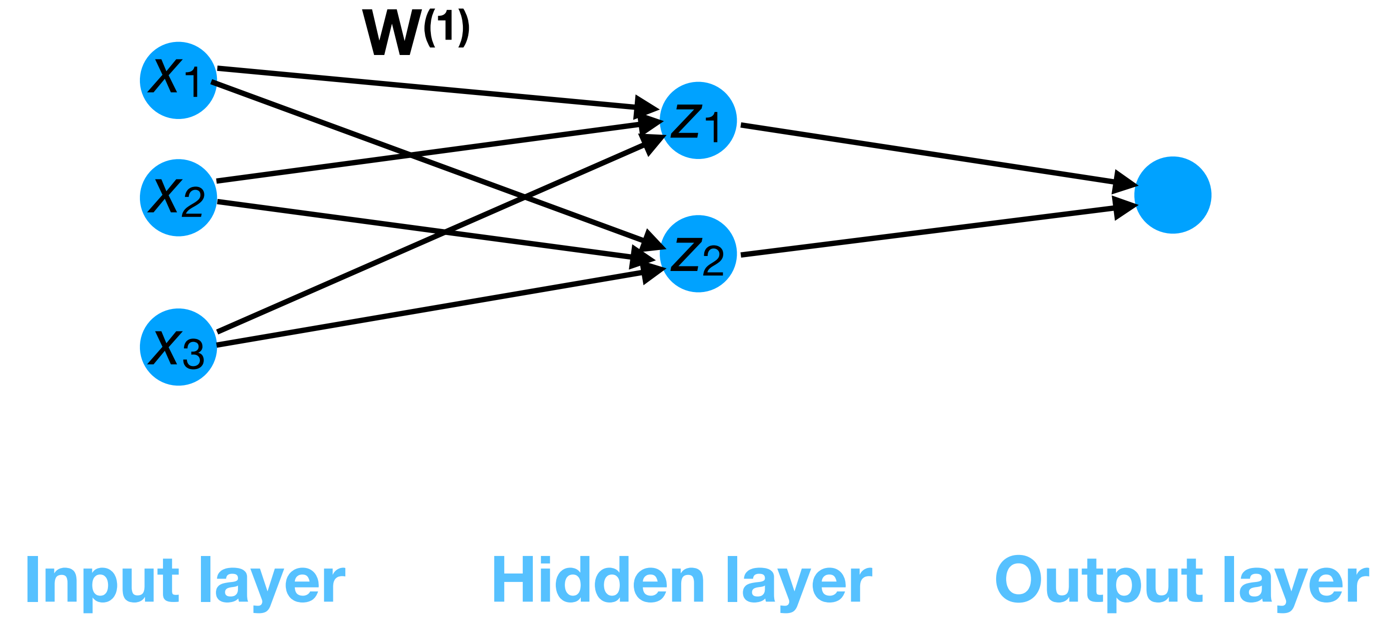


Feed-forward NN

- We can compute the pre-activation values $\mathbf{z}$ of the hidden layer as:

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

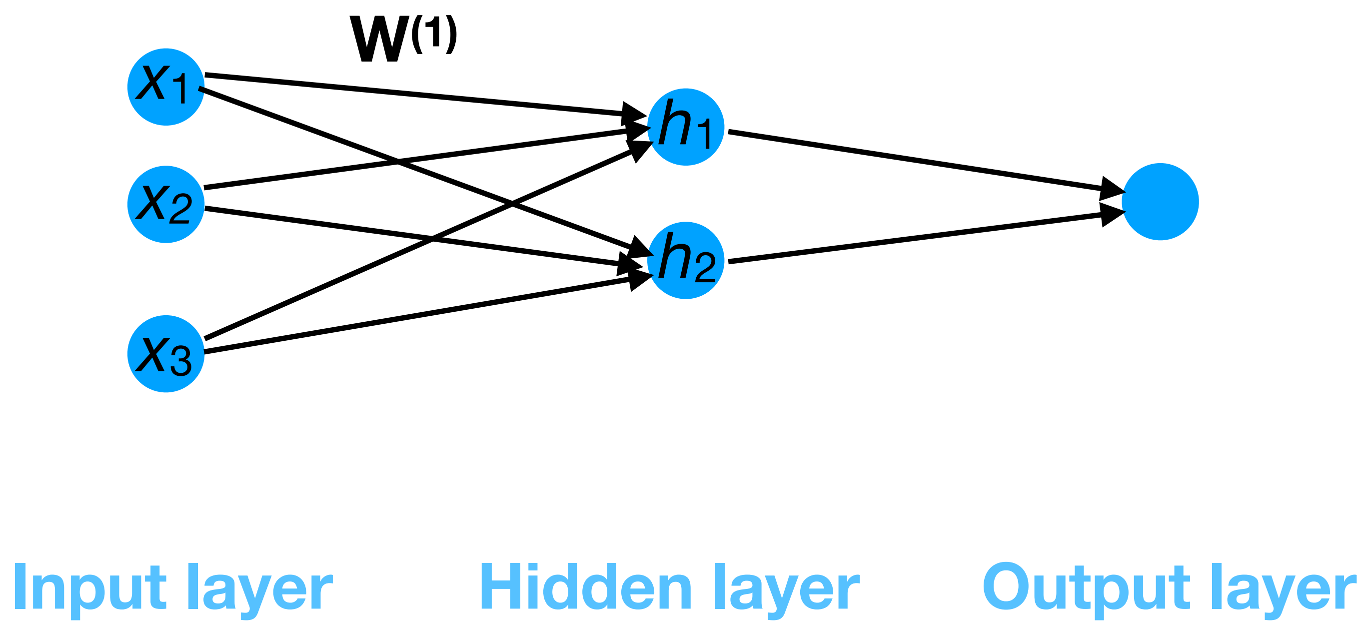
$\mathbf{W}^{(1)}$ is 2 x 3.



Feed-forward NN

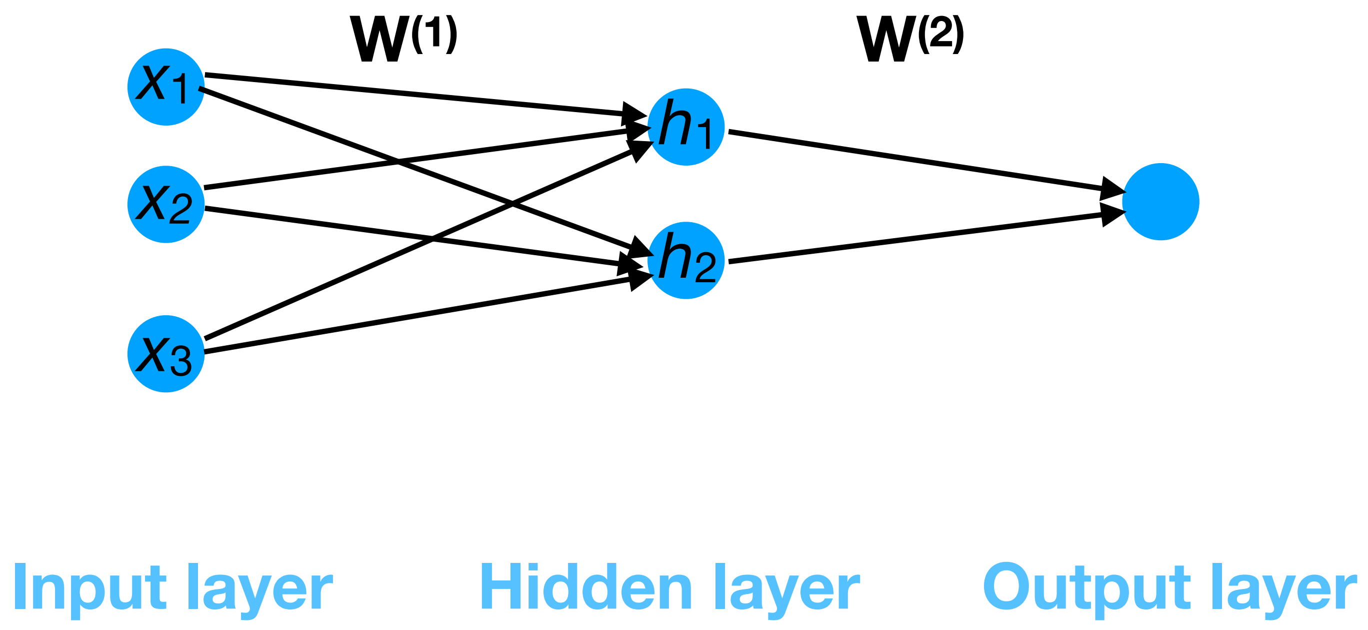
- We can then pass **z** to the activation function σ and compute the hidden neuron values **element-wise**, i.e.:

$$\mathbf{h}_j = \sigma(\mathbf{z}_j)$$



Feed-forward NN

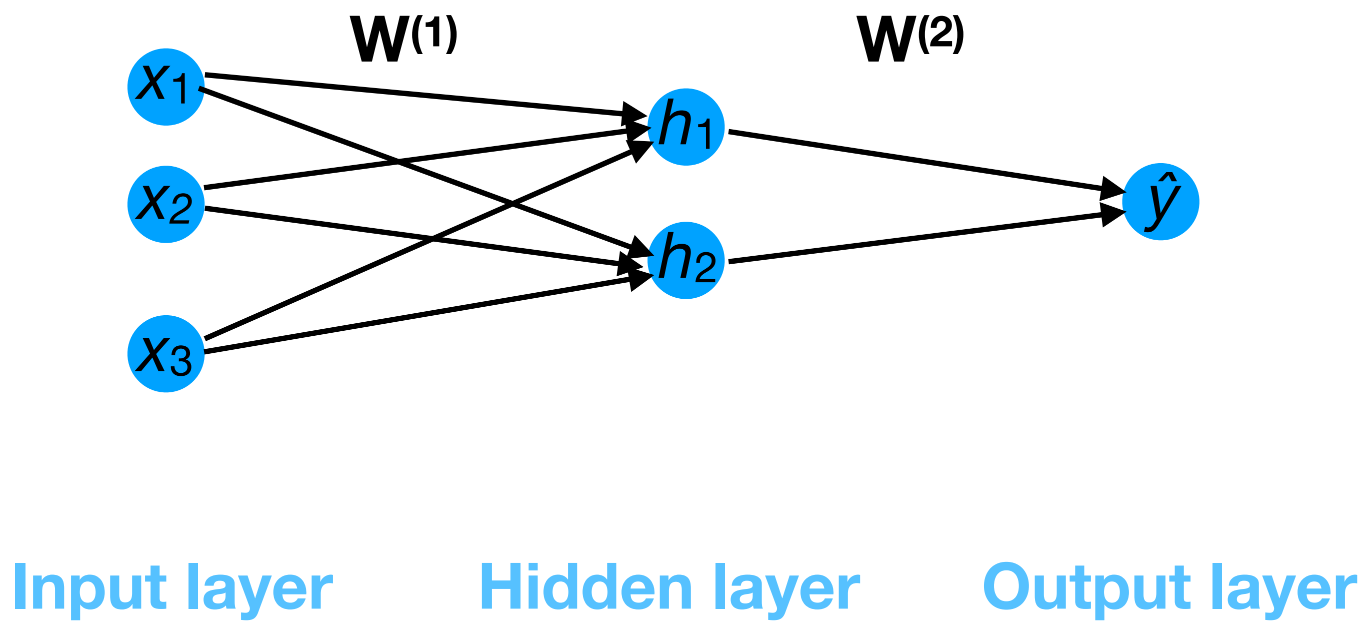
- Next, we pass **h** to the next layer…



Feed-forward NN

- …and compute the product:

$$\hat{\mathbf{y}} = \mathbf{W}^{(2)}\mathbf{h}$$

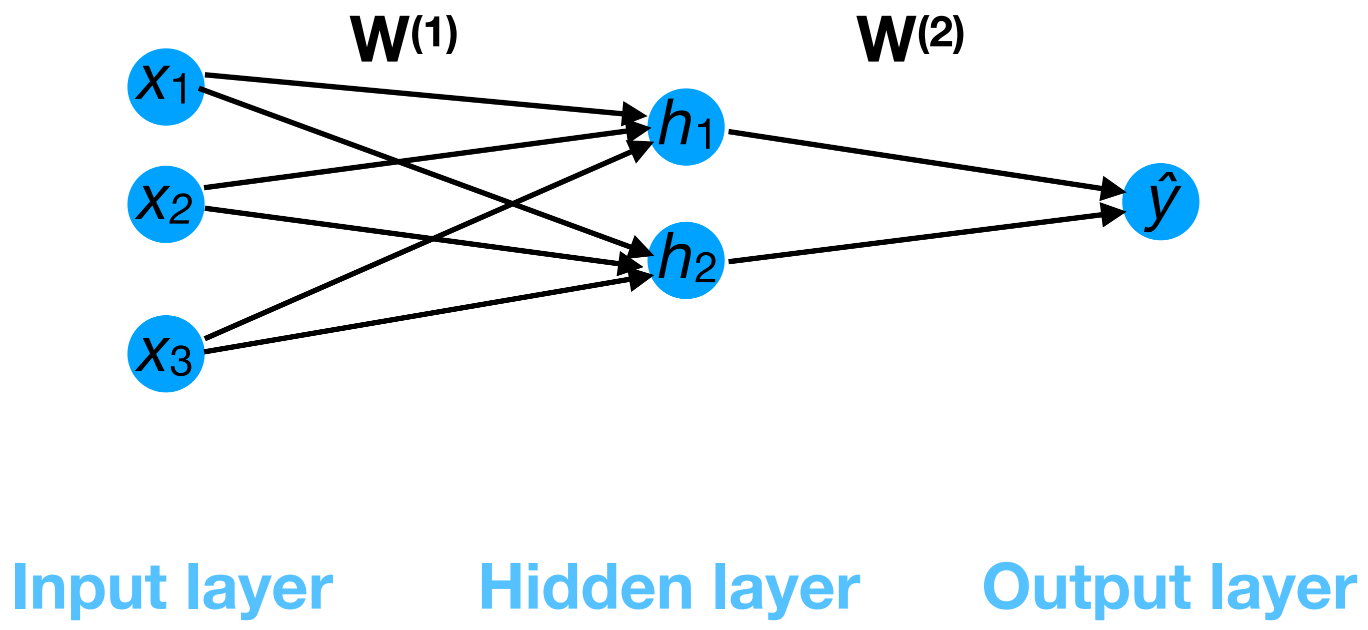


Feed-forward NN

- …and compute the product:

$$\hat{\mathbf{y}} = \mathbf{W}^{(2)}\mathbf{h}$$

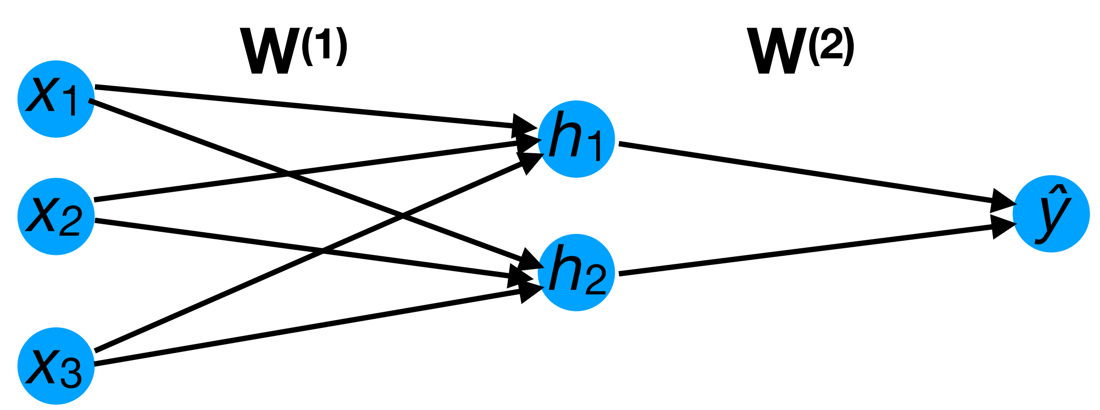
$\mathbf{W}^{(2)}$ is 1 x 2.



Feed-forward NN

- Note that the final layer could also have an activation function (if we wanted one), e.g.:

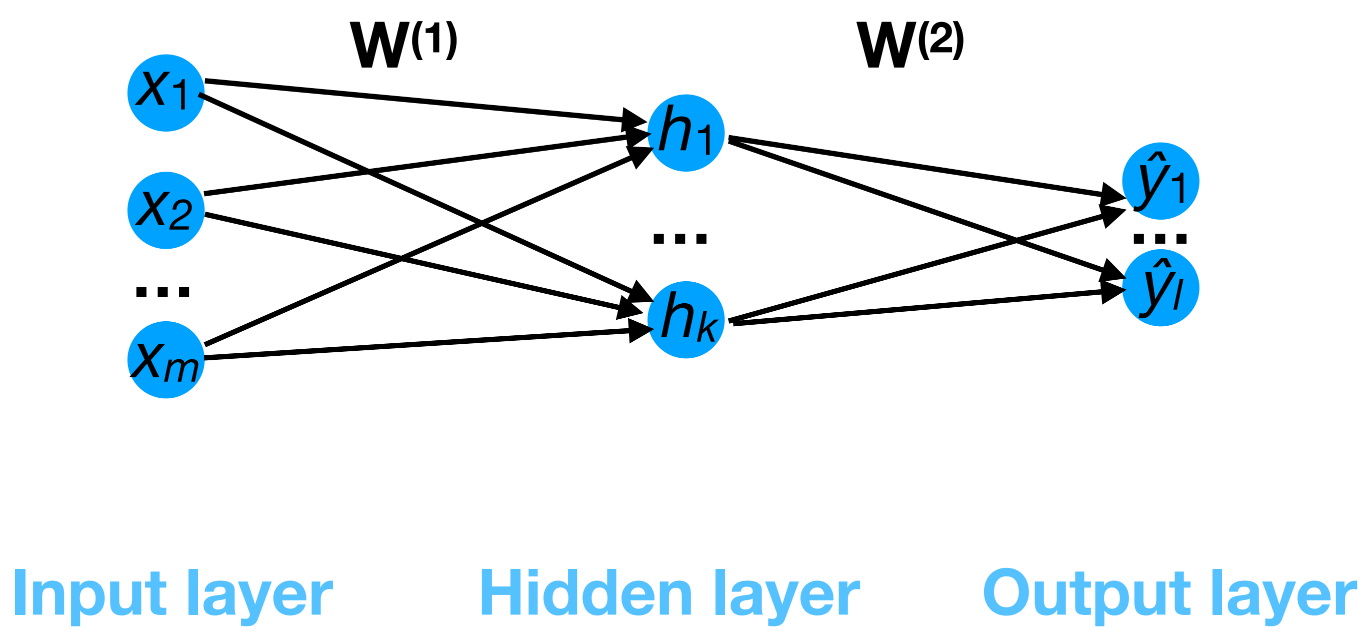
$$\hat{\mathbf{y}} = \sigma\left(\mathbf{W}^{(2)}\mathbf{h}\right)$$



Input layer Hidden layer Output layer

Multiple output neurons

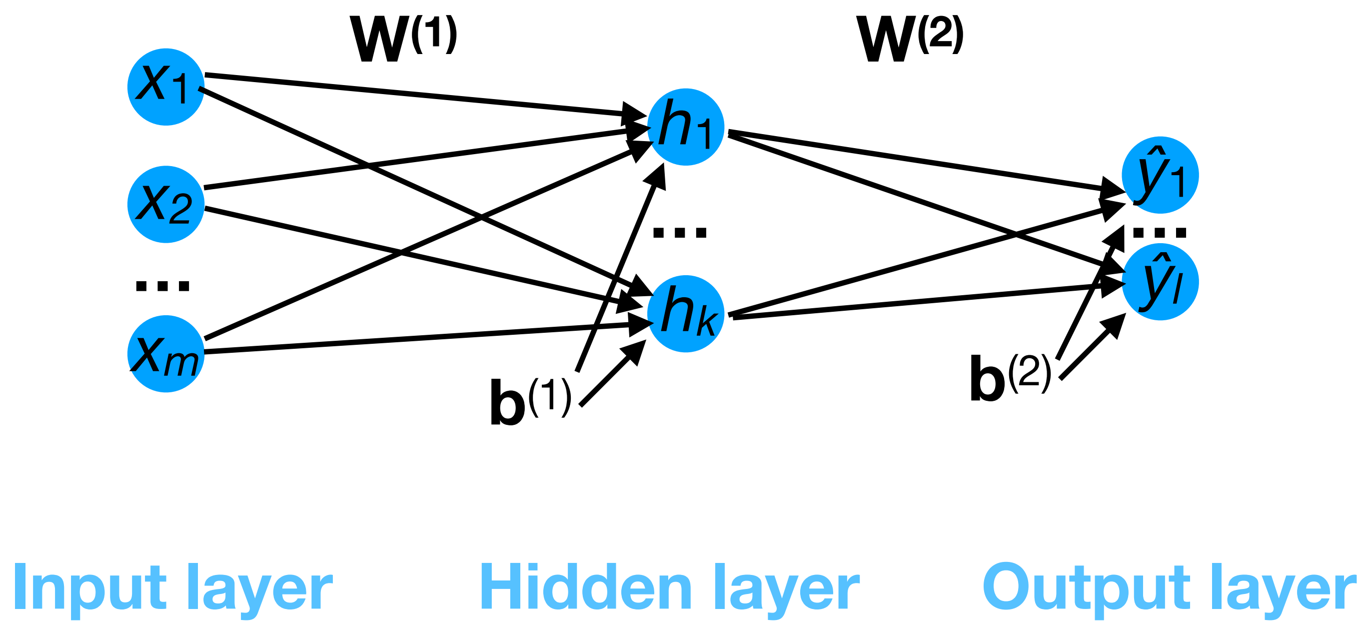
- We can also have a NN with multiple output neurons, e.g.:



Bias terms

- We typically include a **bias term** for every neuron, so that the layers' values are computed as:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \qquad \textbf{and} \qquad \hat{\mathbf{y}} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$



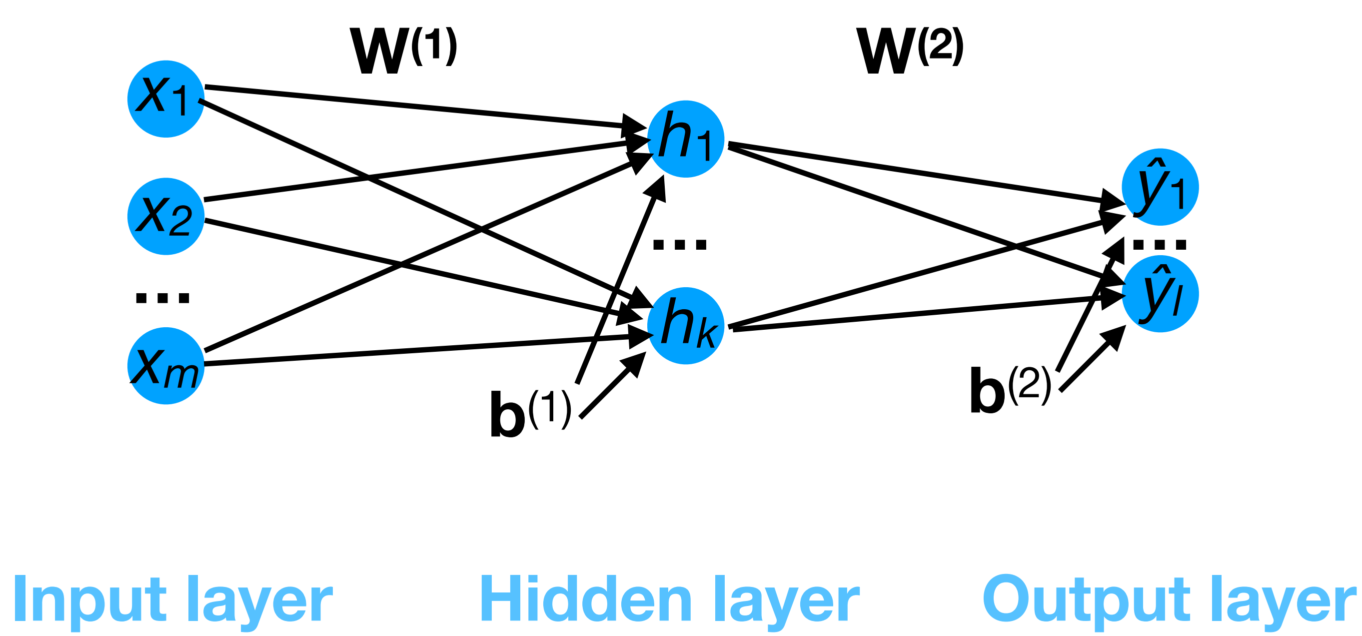
Exercise: bias terms

- What will **ŷ** be for **x** = [1 0 -2]ᵀ,

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

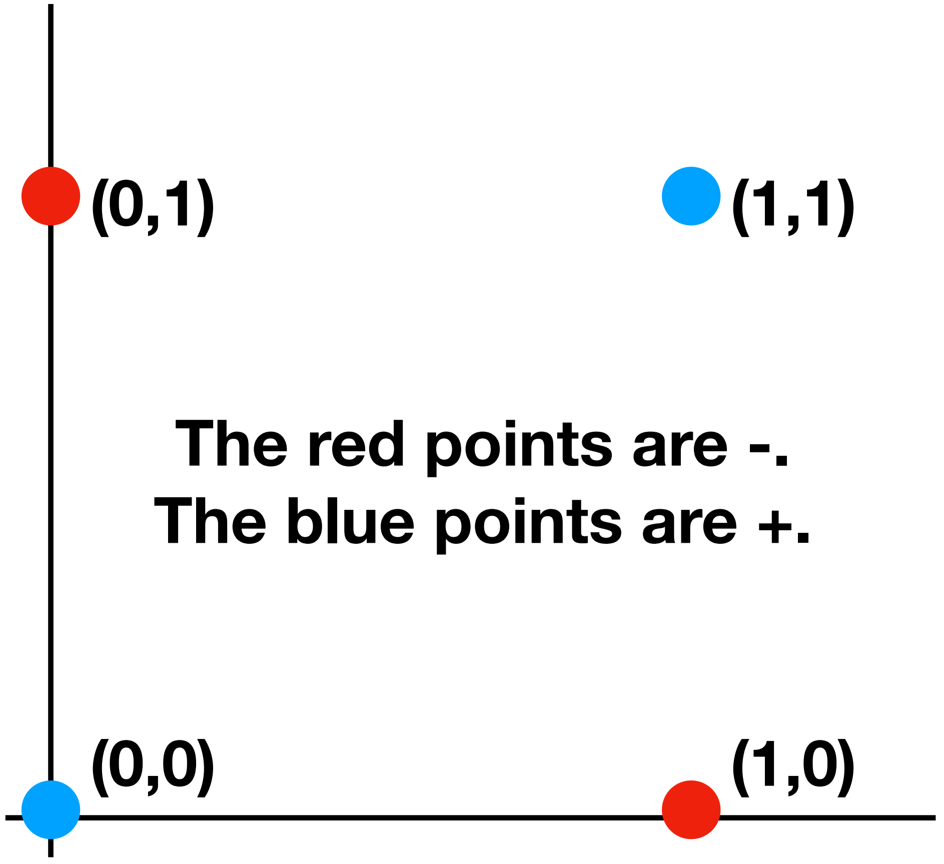
$$\mathbf{W}^{(1)} = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 2 & 3 \end{bmatrix} \quad \mathbf{b}^{(1)} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad \mathbf{b}^{(2)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$



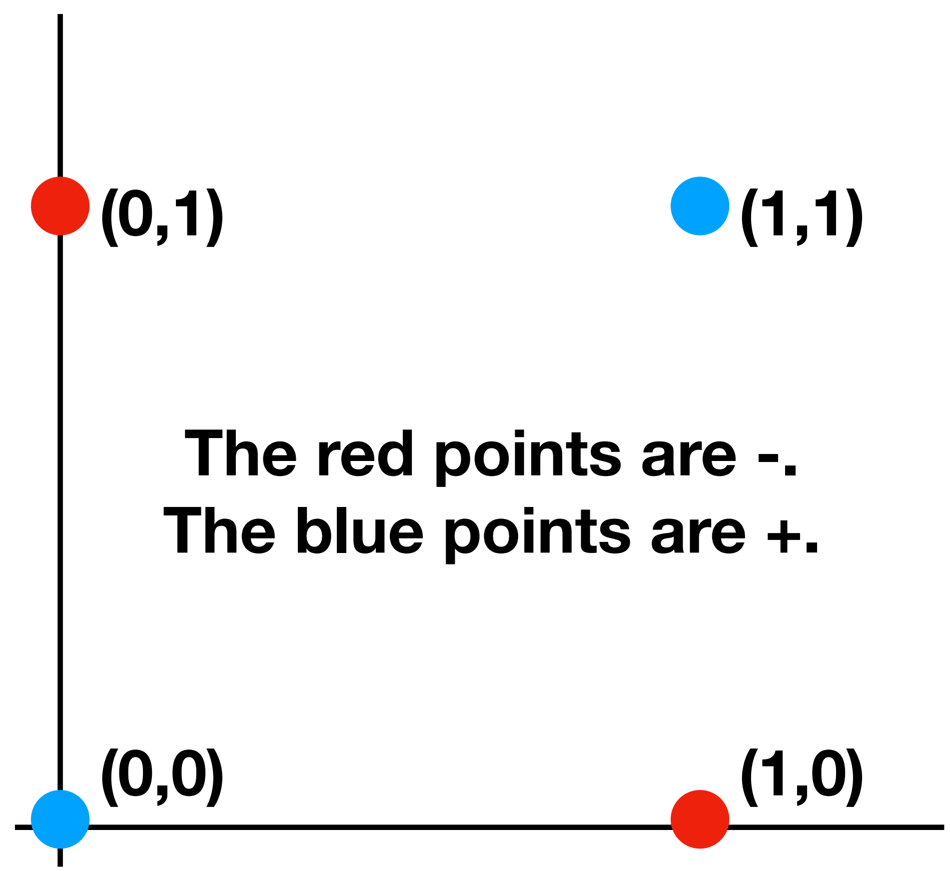
XOR problem

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



XOR problem

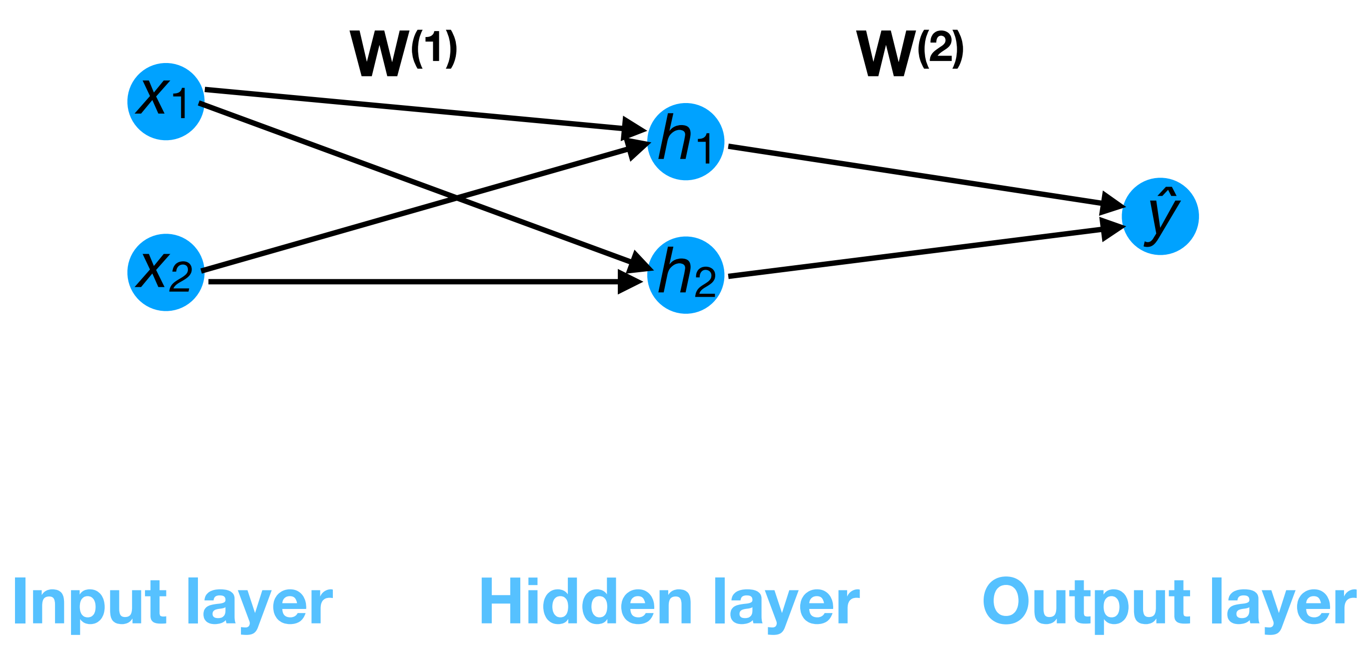
- Recall that no linear decision boundary (e.g., linear SVM, linear regression) can solve the XOR problem.



- Let’s see how using a hidden layer can help us solve it…

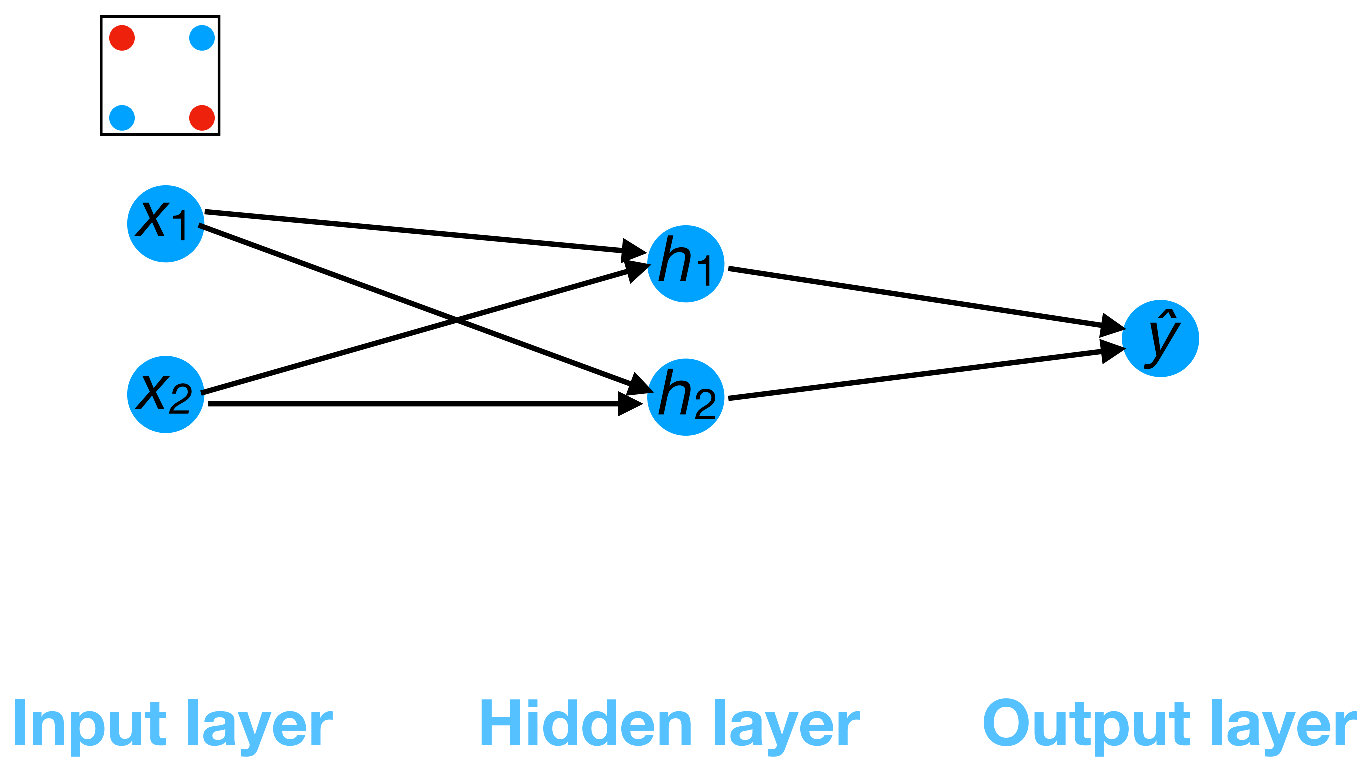
XOR problem

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



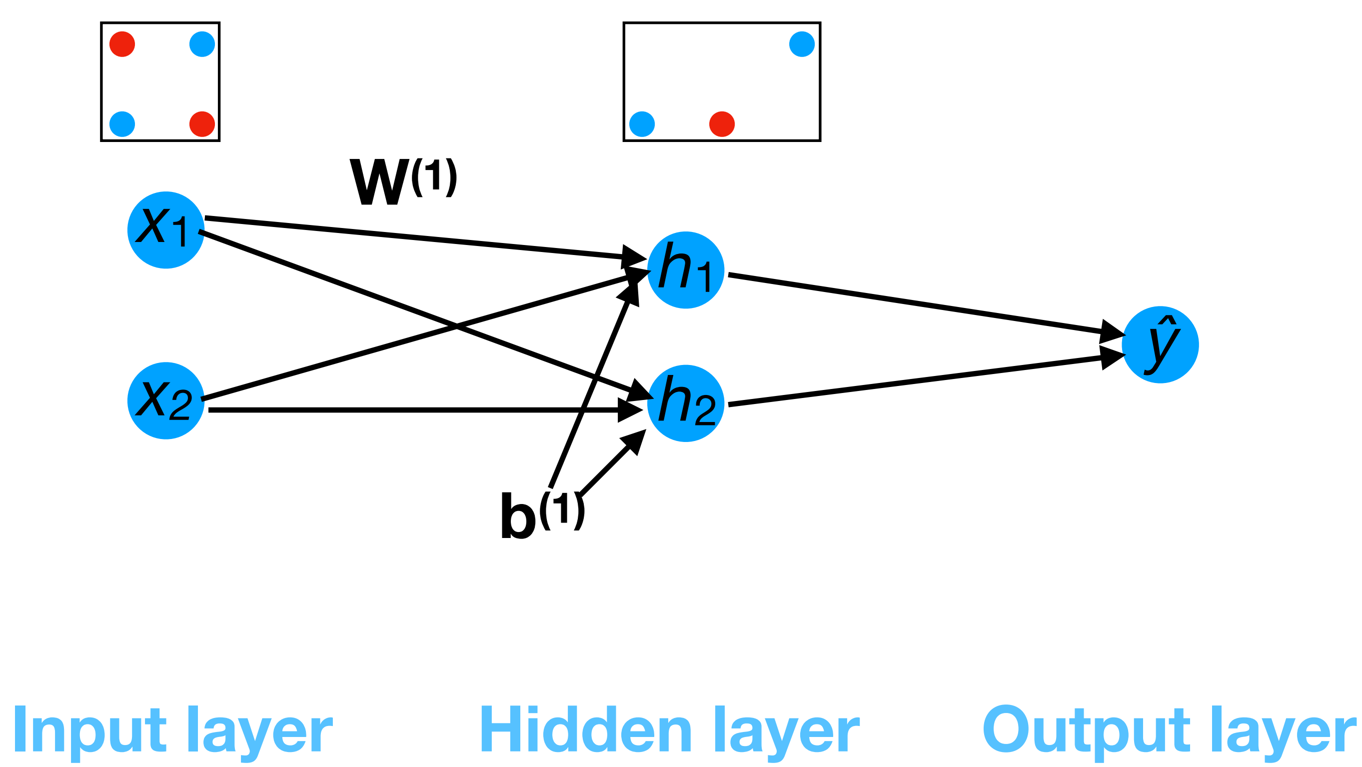
XOR problem

- Here's how a 3-layer NN with a non-linear (ReLU) activation function can solve it:



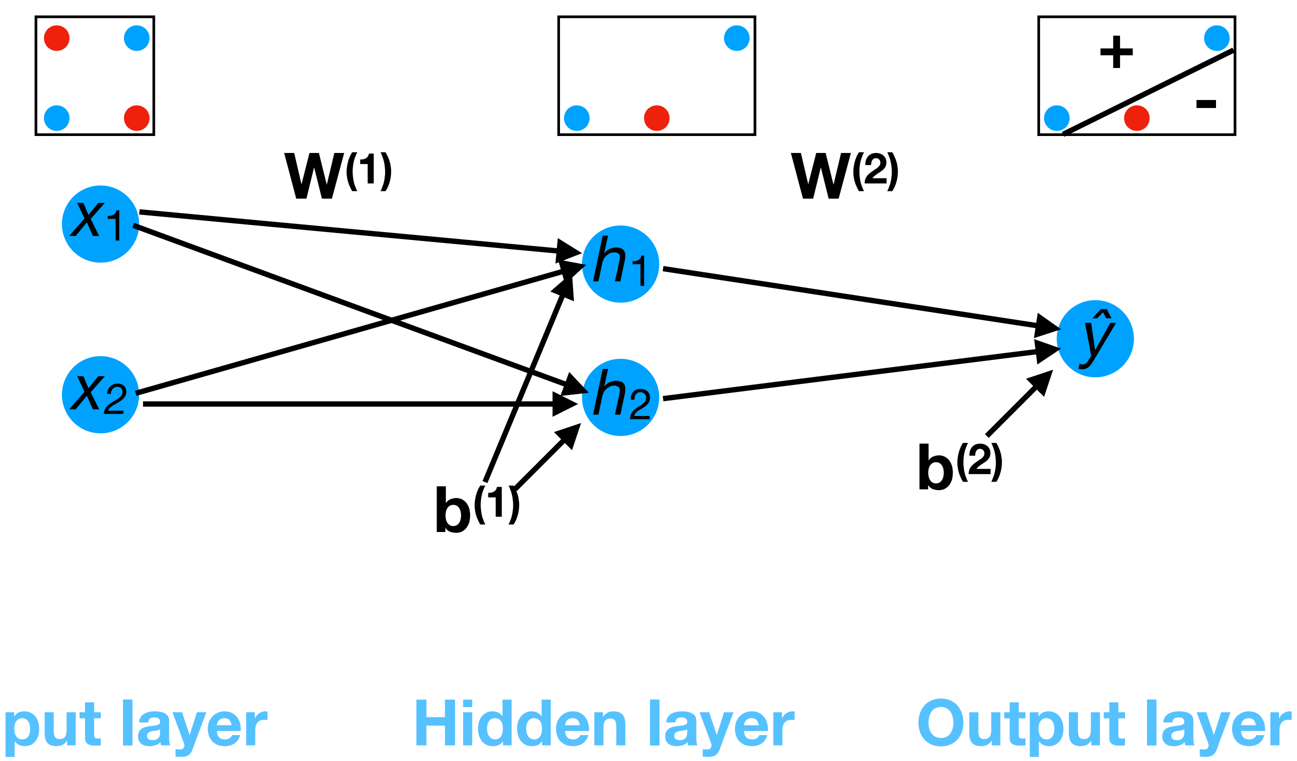
XOR problem

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



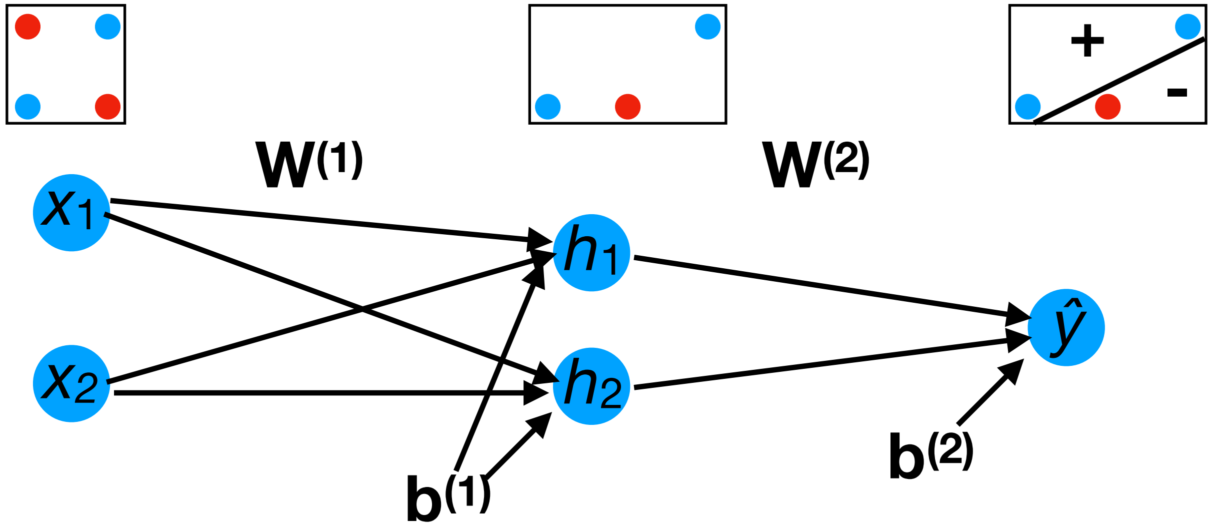
XOR problem

- Here's how a 3-layer NN with a non-linear (ReLU) activation function can solve it:
  - $\mathbf{W}^{(1)}$, $\mathbf{b}^{(1)}$ 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.



XOR problem

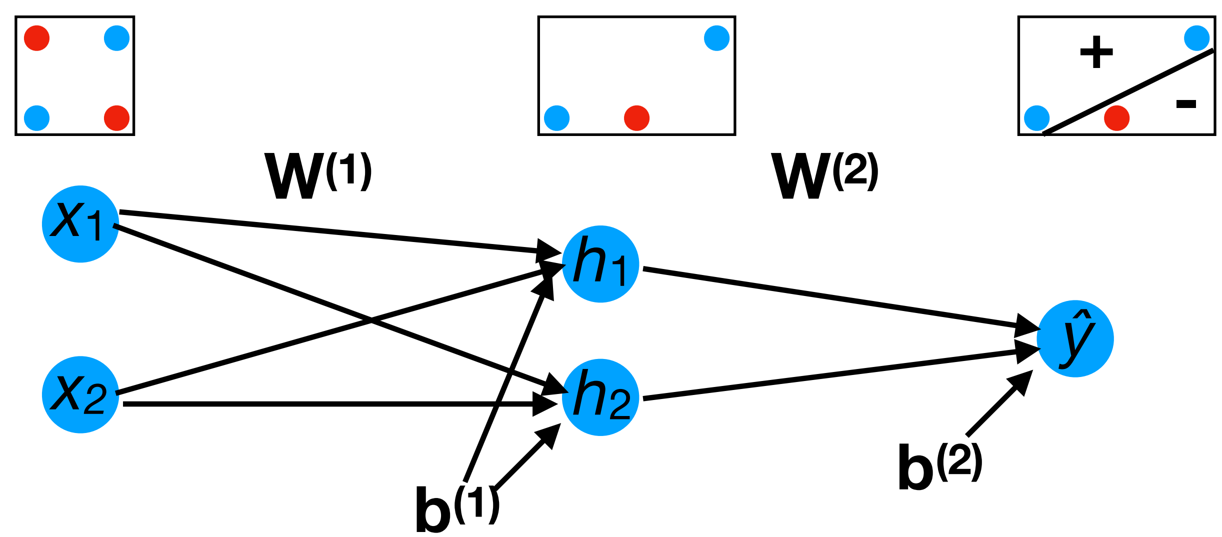
- Here's how a 3-layer NN with a non-linear (ReLU) activation function can solve it:
  - $\mathbf{W}^{(1)}$, $\mathbf{b}^{(1)}$ will "collapse" the two - data points onto one point in the "hidden" 2-D space.
  - Since the + and - data are now linearly separated, $\mathbf{W}^{(2)}$, $\mathbf{b}^{(2)}$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



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

XOR problem

- Here's how a 3-layer NN with a non-linear (ReLU) activation function can solve it:
  - **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.)

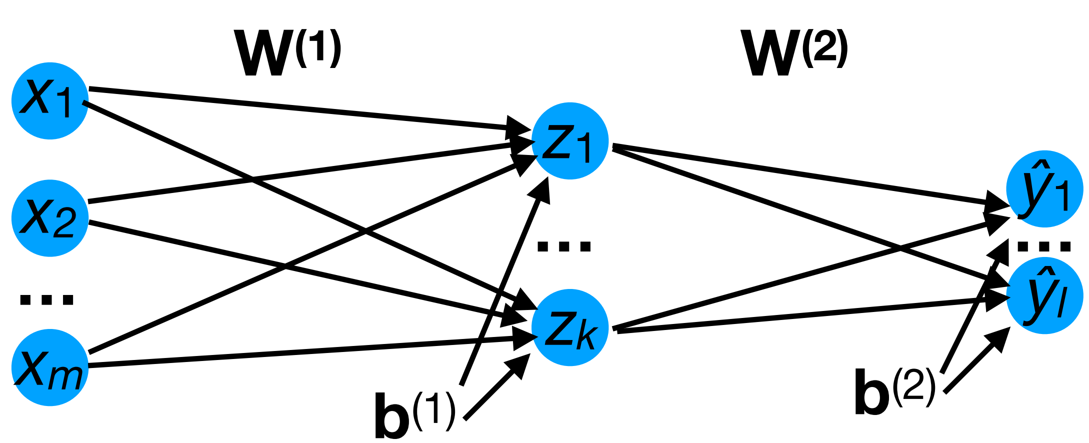
XOR problem

- Note that the ability of the 3-layer NN to solve the XOR problem relies crucially on the non-linear ReLU activation function.
  - Note that 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

- Suppose we define a 3-layer NN **without** non-linearity:

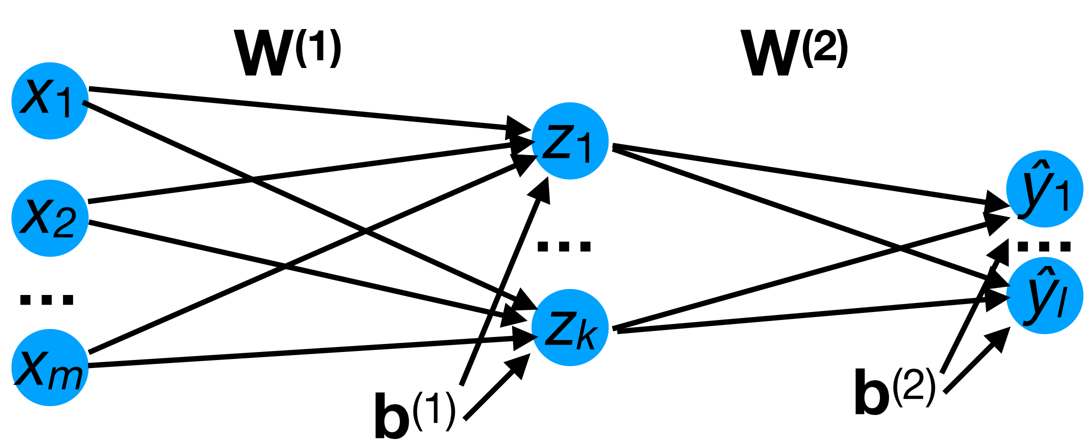
$$g(\mathbf{x}) \quad = \quad \mathbf{W}^{(2)}\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}$$



Crucial role of non-linearity

- Then we can simplify *g* to be:

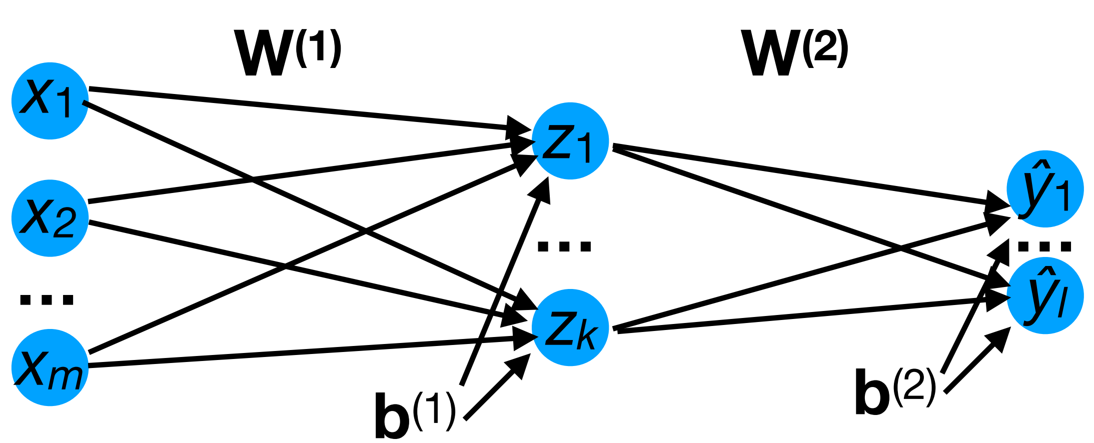
$$g(\mathbf{x}) \quad = \quad \mathbf{W}^{(2)}\left(\mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)}\right)+\mathbf{b}^{(2)}$$



Crucial role of non-linearity

- Then we can simplify g to be:

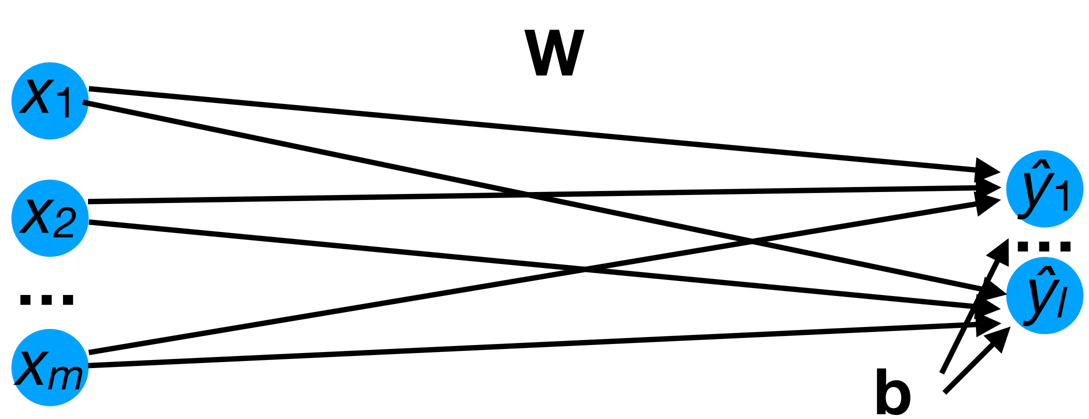
$$
\begin{aligned}
g(\mathbf{x}) &= \mathbf{W}^{(2)}\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)} \\
&= \left(\mathbf{W}^{(2)}\mathbf{W}^{(1)}\right)\mathbf{x} + \mathbf{W}^{(2)}\mathbf{b}^{(1)} + \mathbf{b}^{(2)}
\end{aligned}
$$



Crucial role of non-linearity

- Then we can simplify g to be:

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



Training neural networks

Training neural networks

- While training neural networks by hand is (arguably) fun, it is completely impractical except for toy examples.
- How can we find good values for the weights and bias terms automatically?

Training neural networks

- While training neural networks by hand is (arguably) fun, it is completely impractical except for toy examples.
- How can we find good values for the weights and bias terms automatically?
  - Gradient descent.

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:

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

Gradient descent: XOR problem

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

- Let's first define:

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

- Then we can define *g* so that:

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

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}):

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

Gradient descent: XOR problem

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

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

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

$$\frac{\partial f_{\text{MSE}}}{\partial p} = \frac{1}{2}\sum_{i=1}^{n}\frac{\partial f_{\text{MSE}}}{\partial \hat{y}^{(i)}}\frac{\hat{y}^{(i)}}{\partial p}$$

where:

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

Gradient descent: XOR problem

- Now we just have to differentiate $\hat{y} = g(\mathbf{x})$ 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 $\hat{y} = g(\mathbf{x})$ 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 $\hat{y} = g(\mathbf{x})$ 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 $\hat{y} = g(\mathbf{x})$ 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 $\hat{y}$ = g(**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**

Gradient descent: XOR problem

- We also need to derive the other partial derivatives:

$$\frac{\partial \hat{y}}{\partial w_1}, \ldots \frac{\partial \hat{y}}{\partial w_6}, \frac{\partial \hat{y}}{\partial b_1}, \ldots \frac{\partial \hat{y}}{\partial w_3}$$