



# CS 540 Introduction to Artificial Intelligence

## Neural Networks (II)

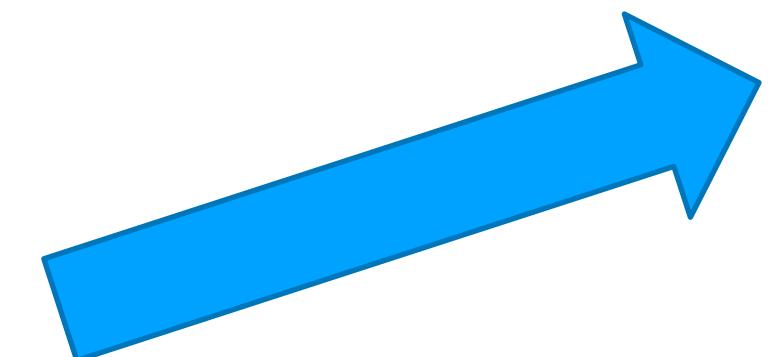
### University of Wisconsin-Madison

**Spring 2023**

# Announcements

- **Homeworks:**
  - None due this week.
- **Midterm Course Evaluation**
- **Class roadmap:**

Last topic  
included on  
midterm!



Tuesday, Mar 7	ML: Neural Networks II
Thursday, Mar 9	Review Lecture; Midterm 5:45-7:15pm
Tuesday, Mar 21	ML: Neural Networks III
Thursday, Mar 23	Deep Learning I

# Today's outline

- Single-layer Perceptron Review
- Multi-layer Perceptron
  - Single output
  - Multiple output
- How to train neural networks
  - Gradient descent

# Review: Perceptron

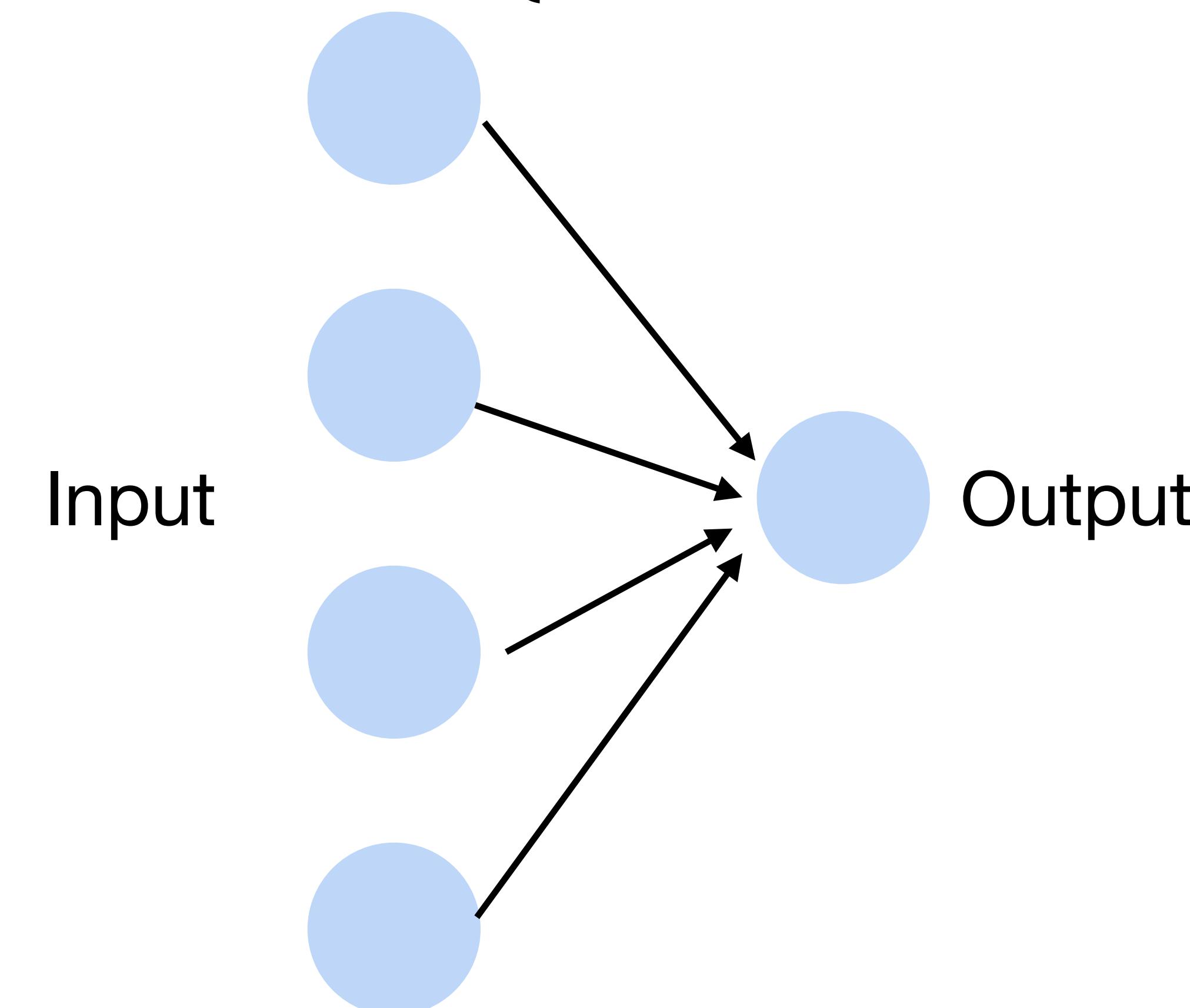
- Given input  $\mathbf{x}$ , weight  $\mathbf{w}$  and bias  $b$ , perceptron outputs:

$$o = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

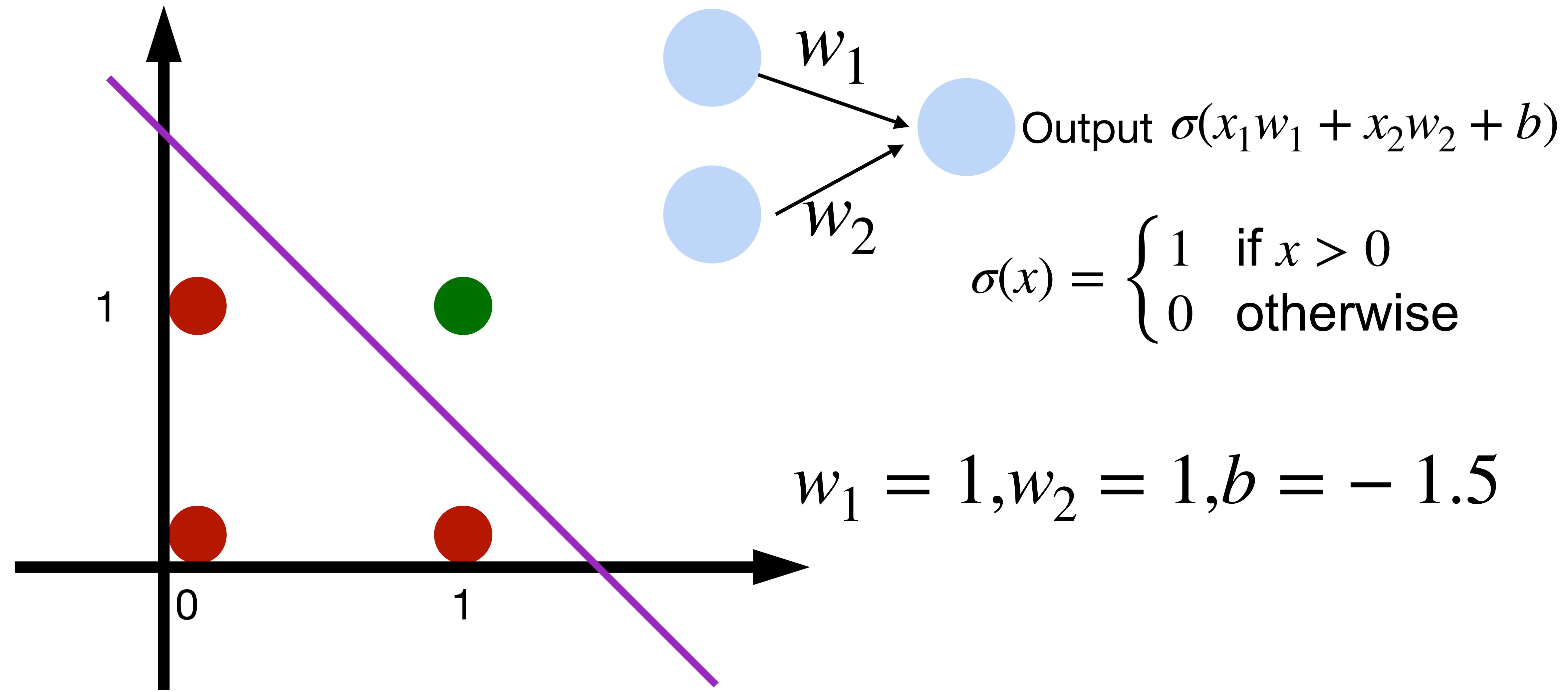
Activation function

Cats vs. dogs?



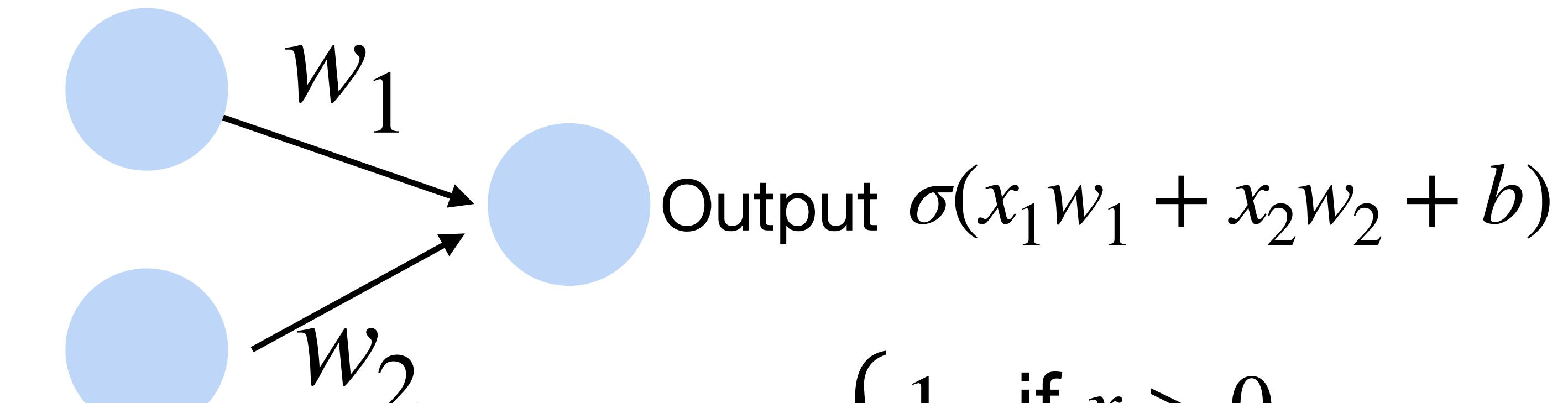
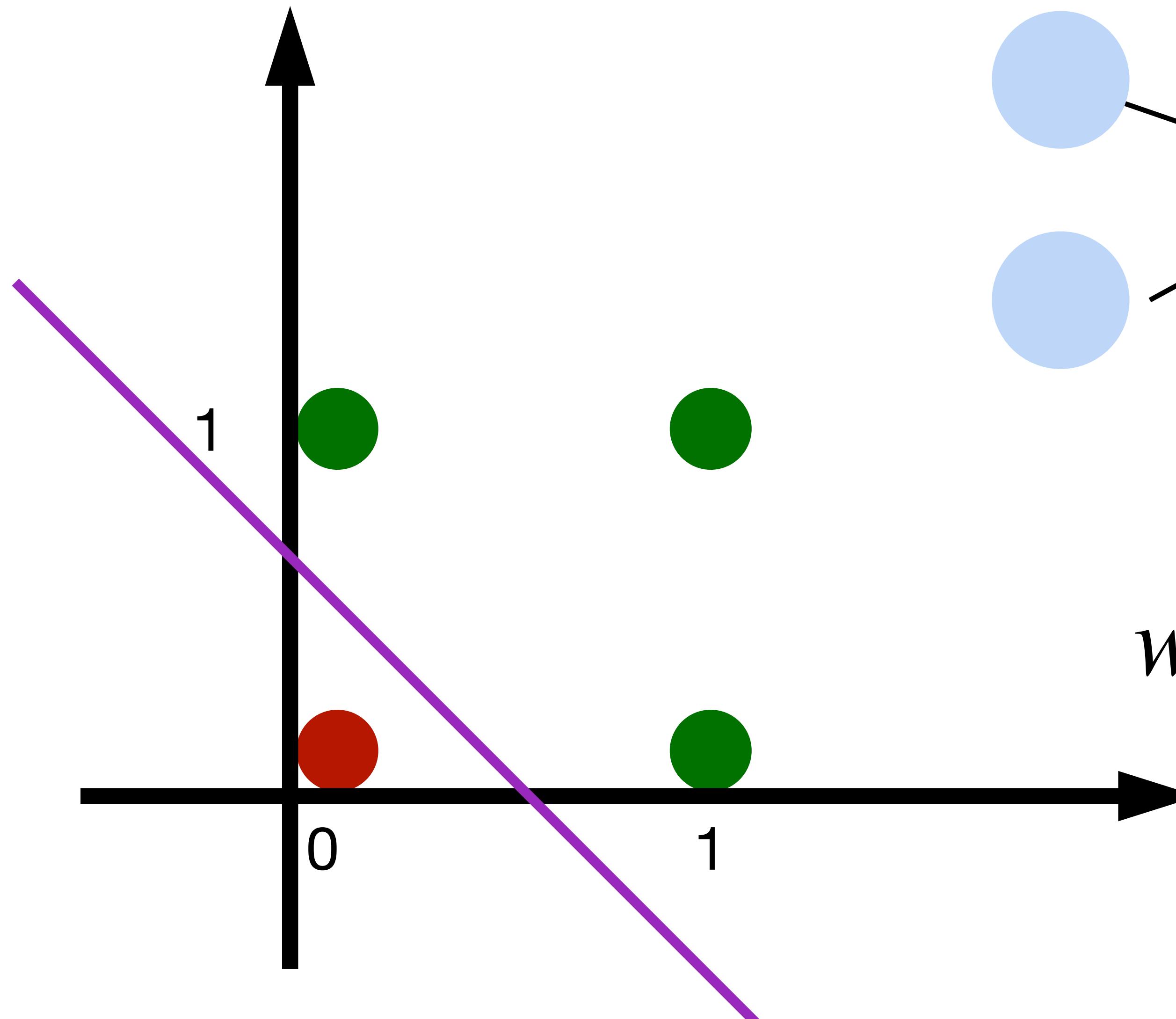
# Learning AND function using perceptron

The perceptron can learn an AND function



# Learning OR function using perceptron

The perceptron can learn an OR function

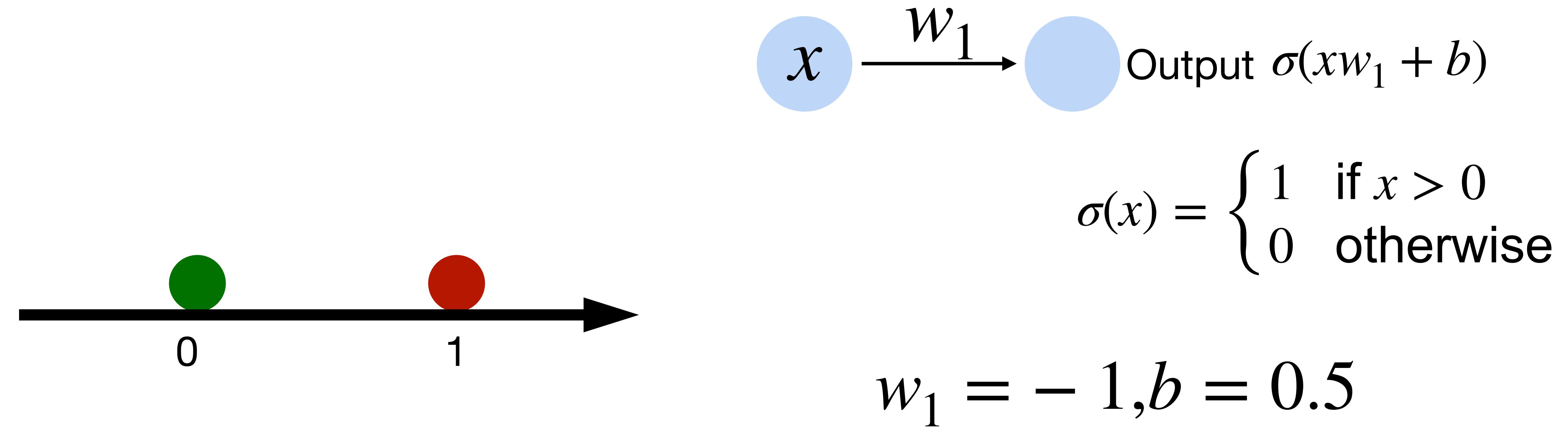


$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$w_1 = 1, w_2 = 1, b = -0.5$$

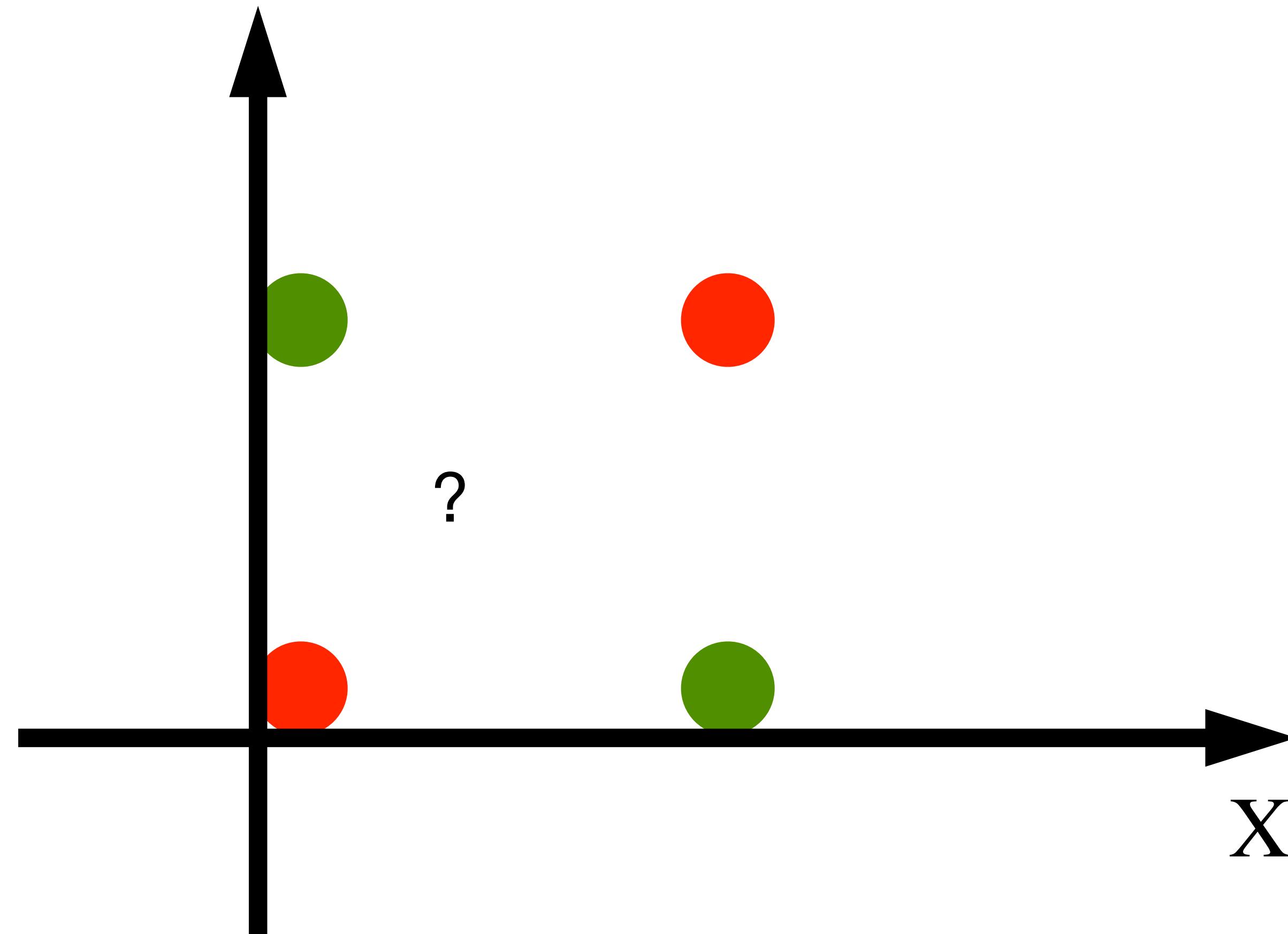
# Learning NOT function using perceptron

The perceptron can learn NOT function (single input)



# The limited power of a single neuron

The perceptron cannot learn an **XOR** function  
(neurons can only generate linear separators)



$$x_1 = 1, x_2 = 1, y = 0$$

$$x_1 = 1, x_2 = 0, y = 1$$

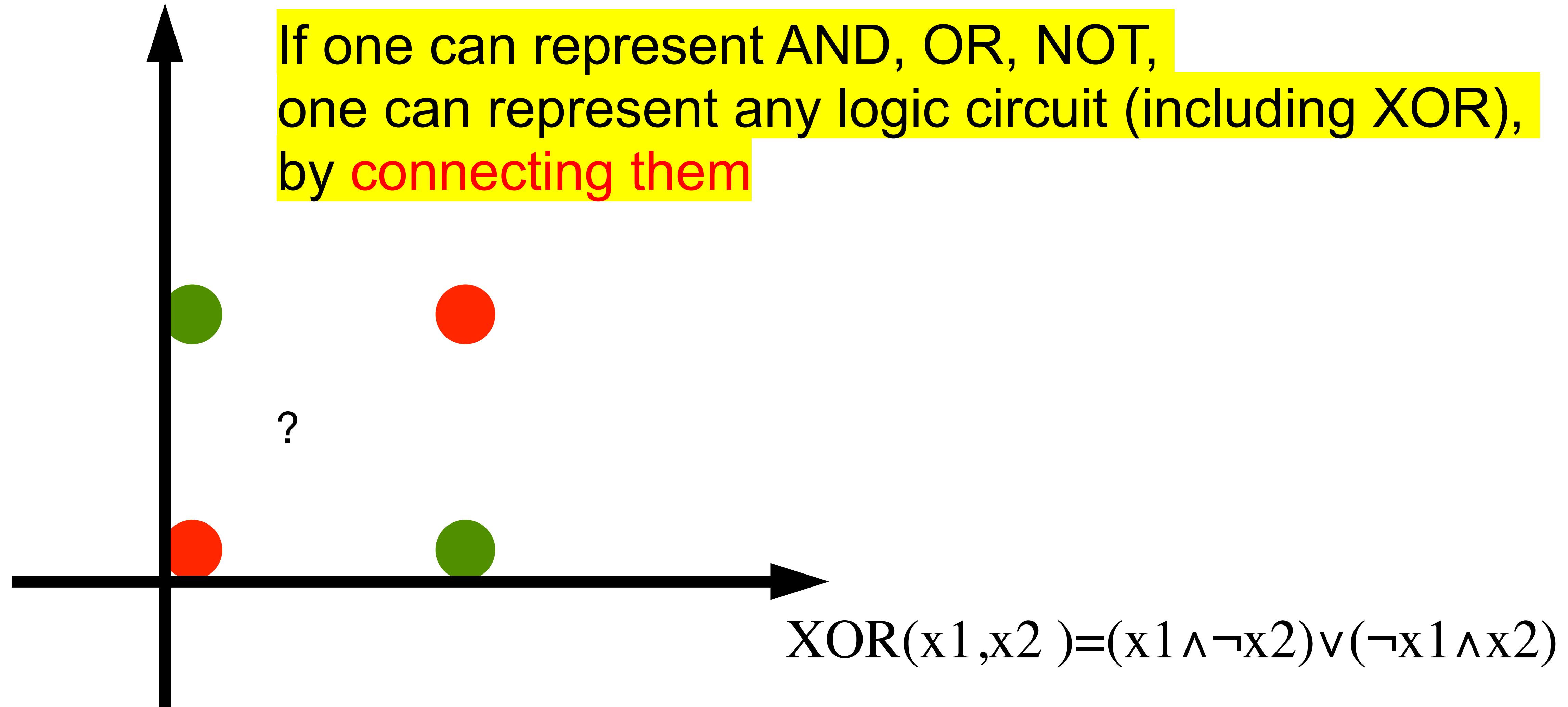
$$x_1 = 0, x_2 = 1, y = 1$$

$$x_1 = 0, x_2 = 0, y = 0$$

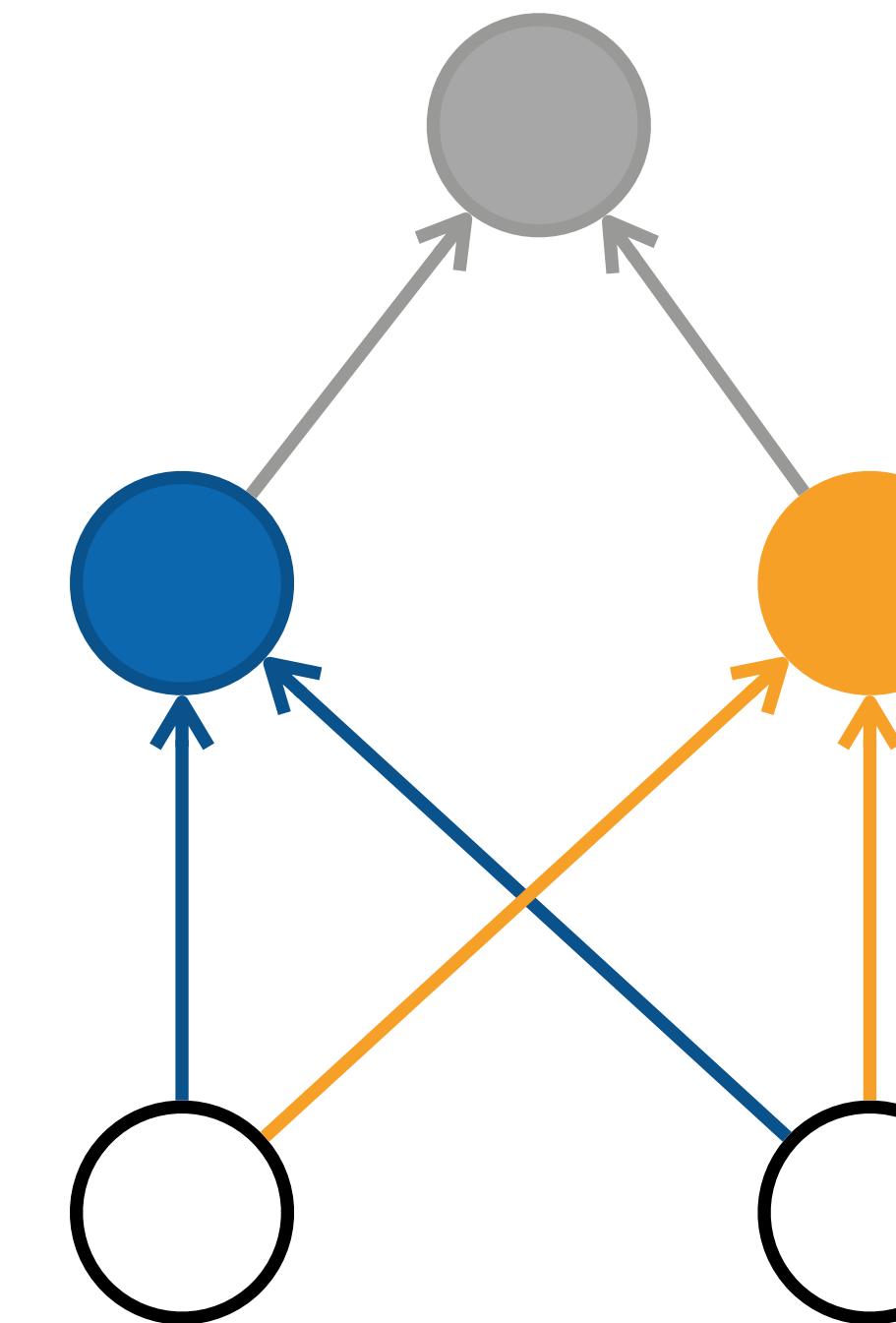
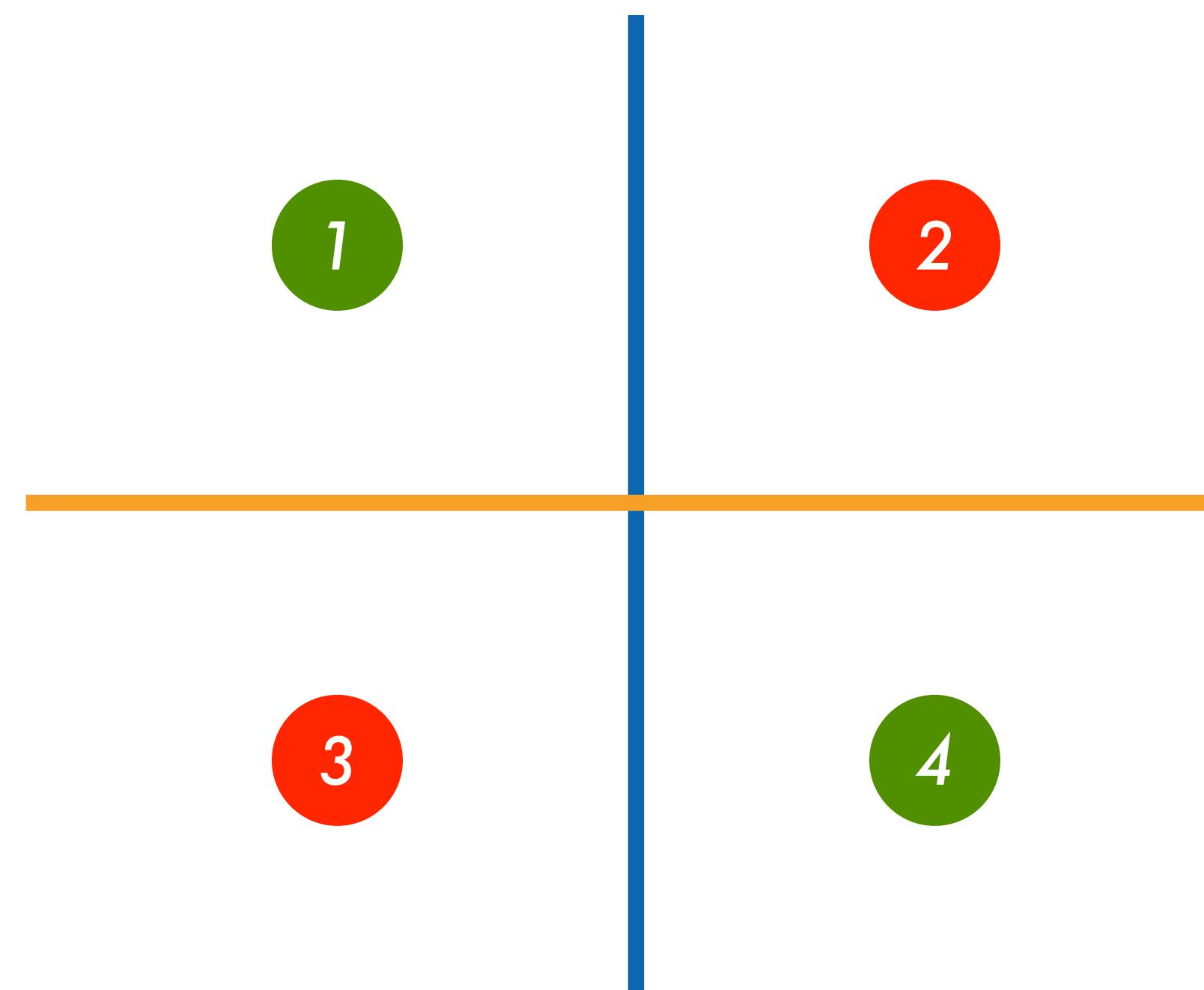
$$\text{XOR}(x_1, x_2) = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$

# The limited power of a single neuron

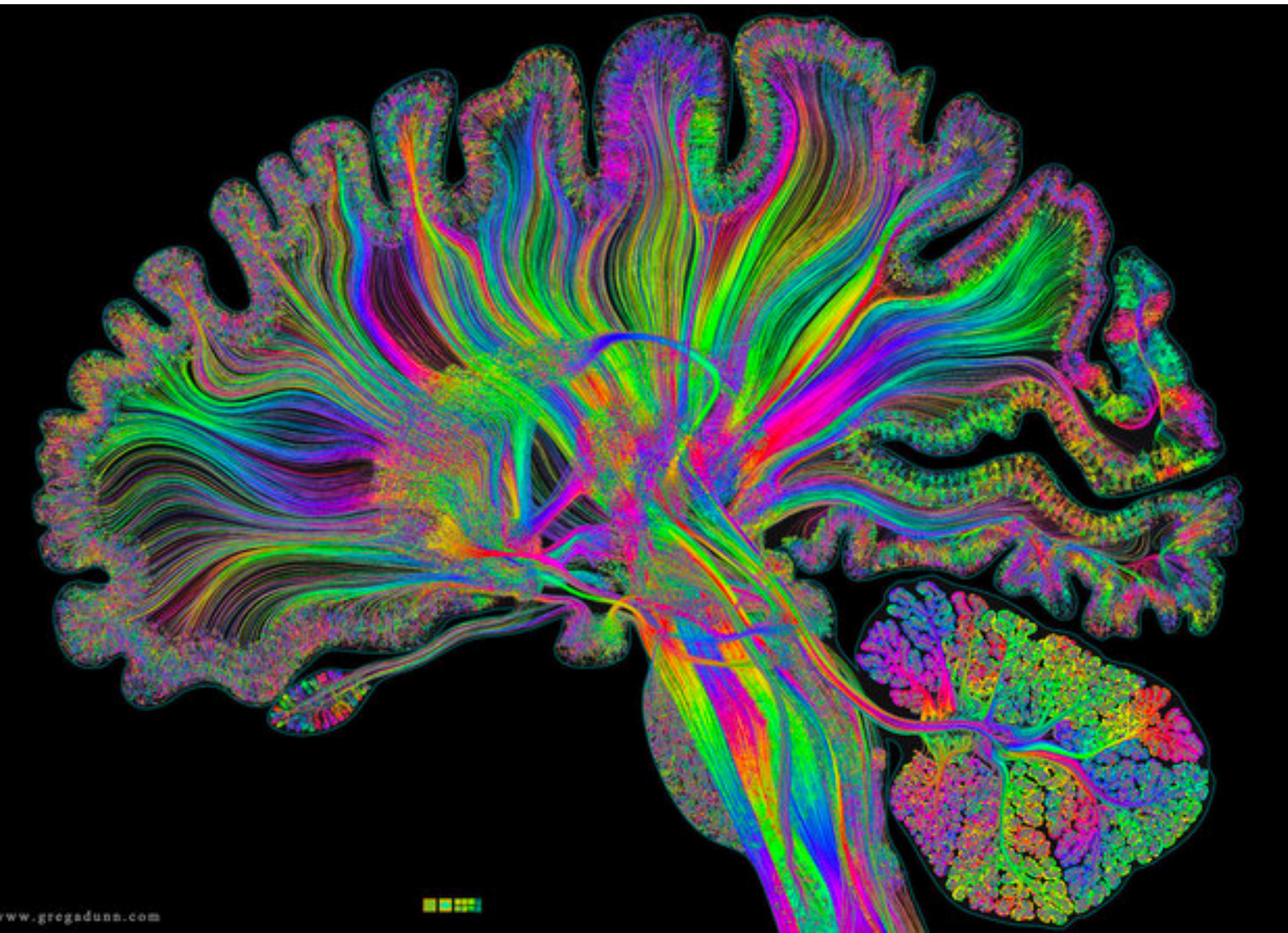
## XOR problem



# Learning XOR

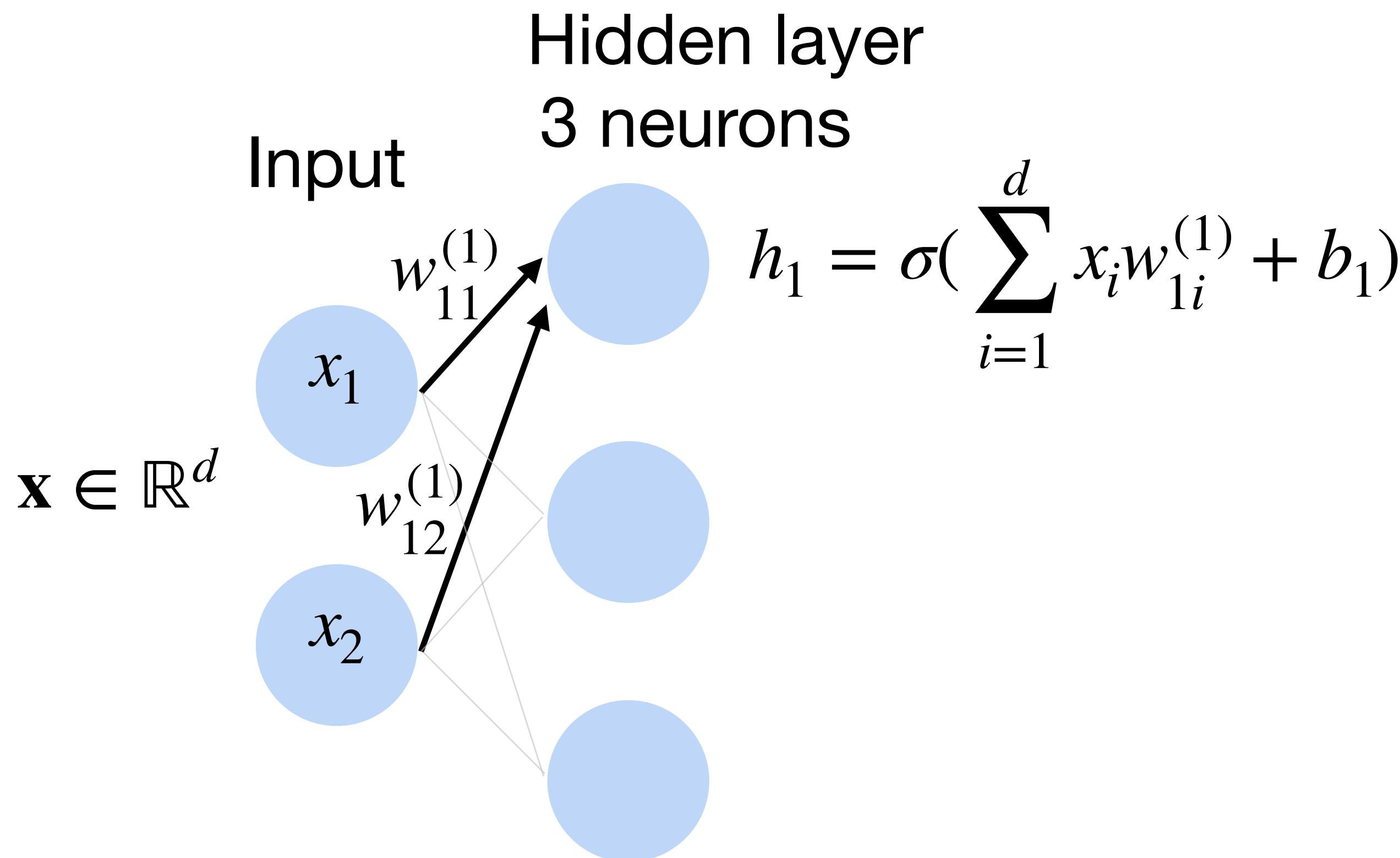


# Multilayer Perceptron



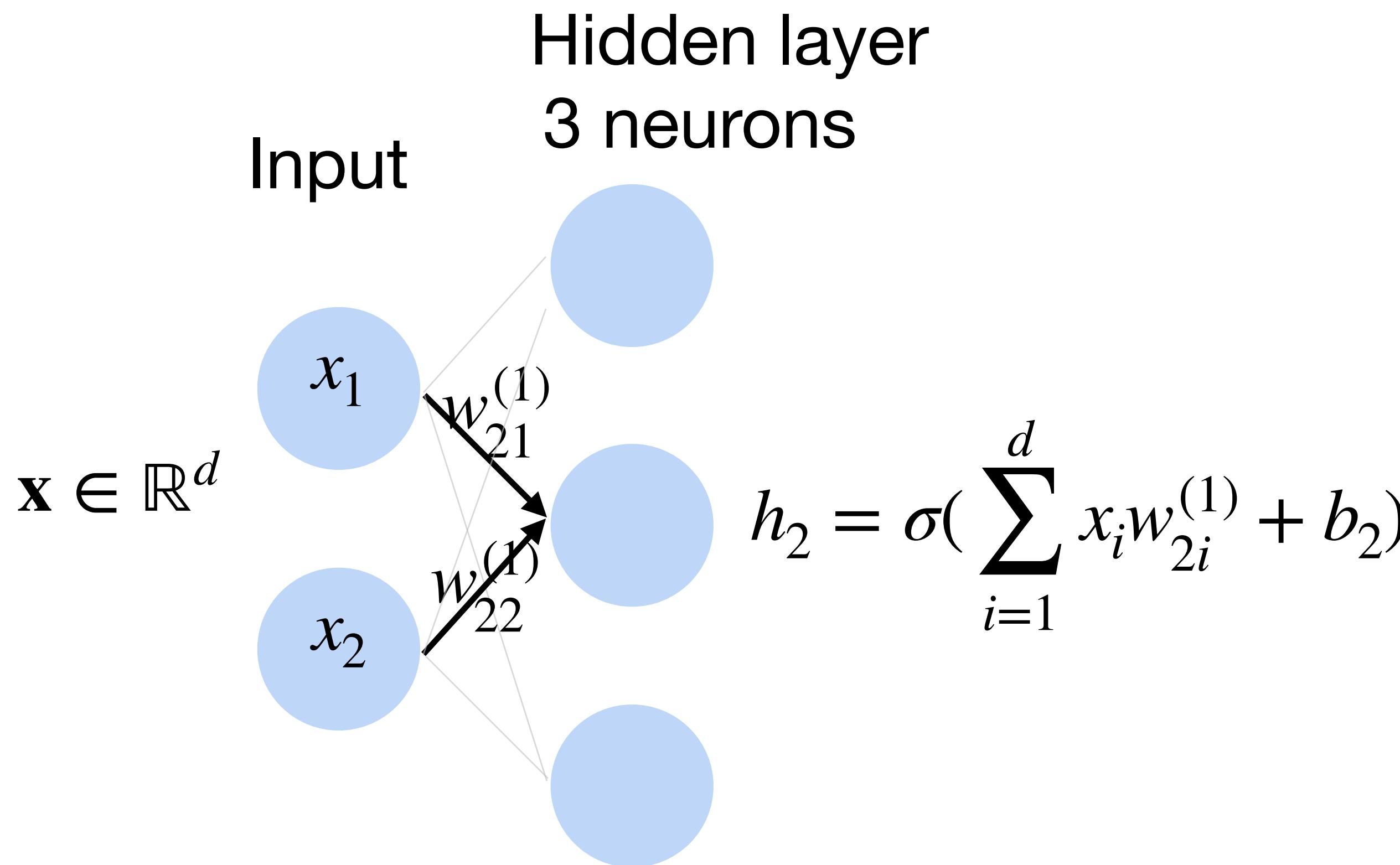
# Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



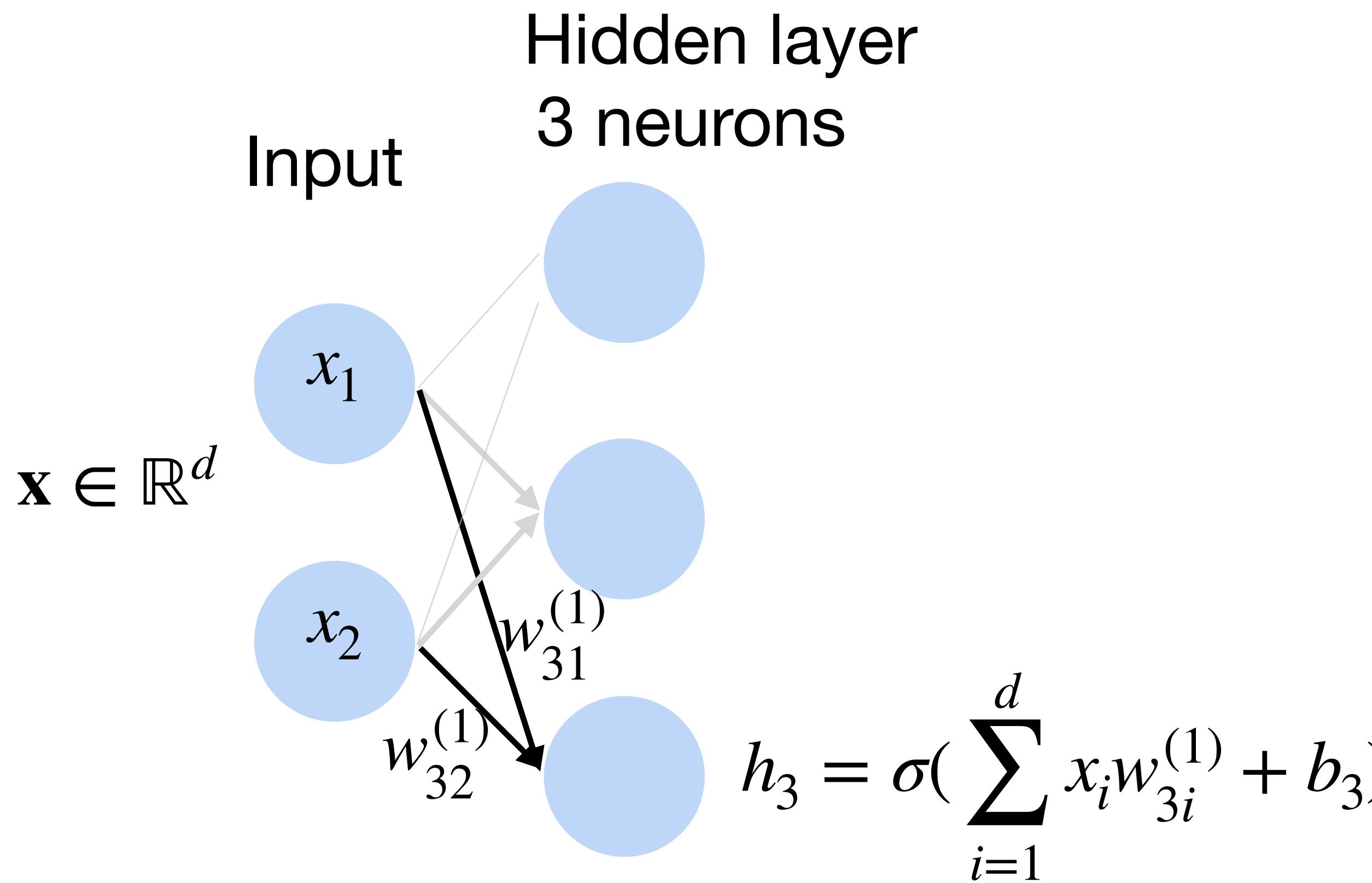
# Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



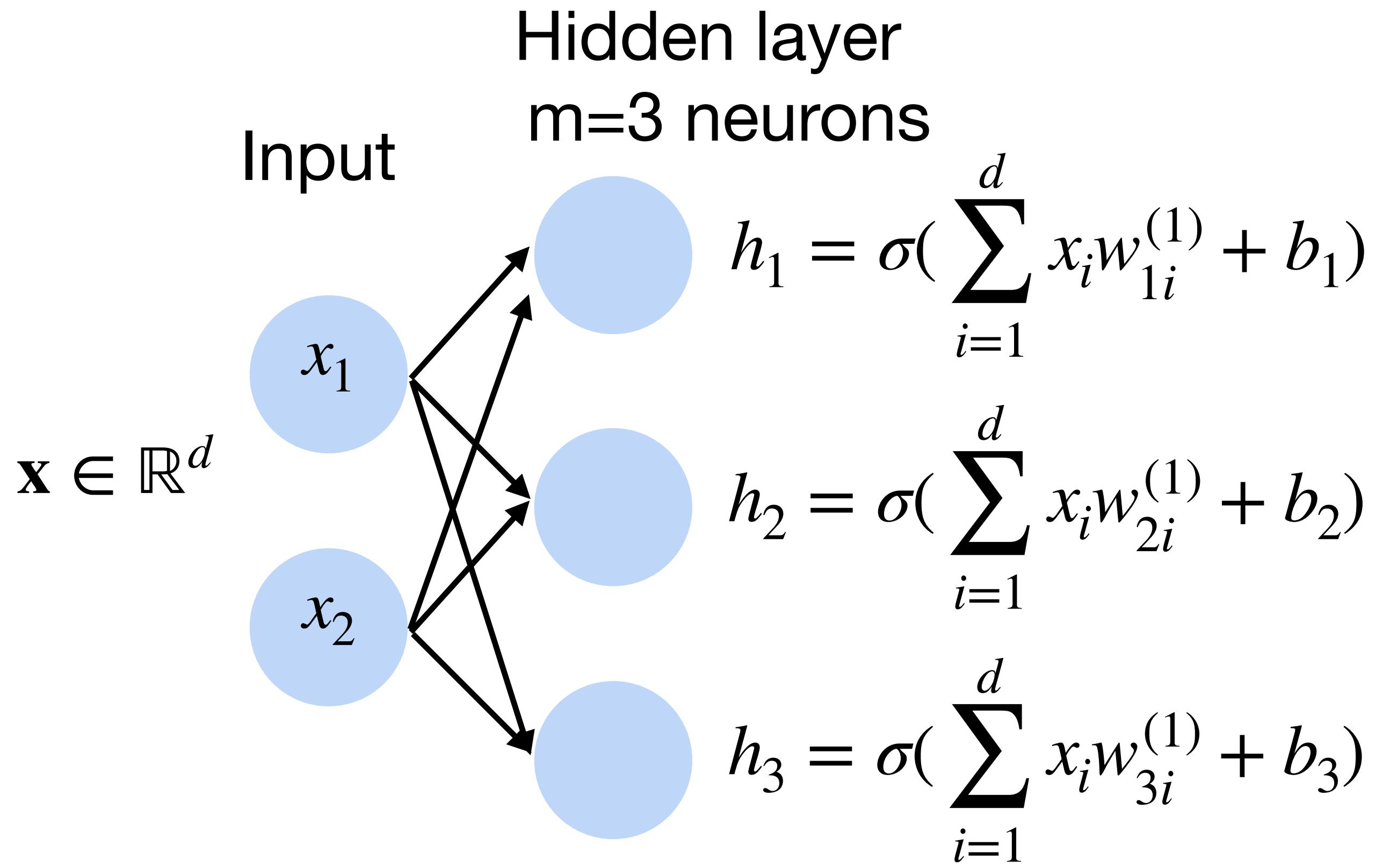
# Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



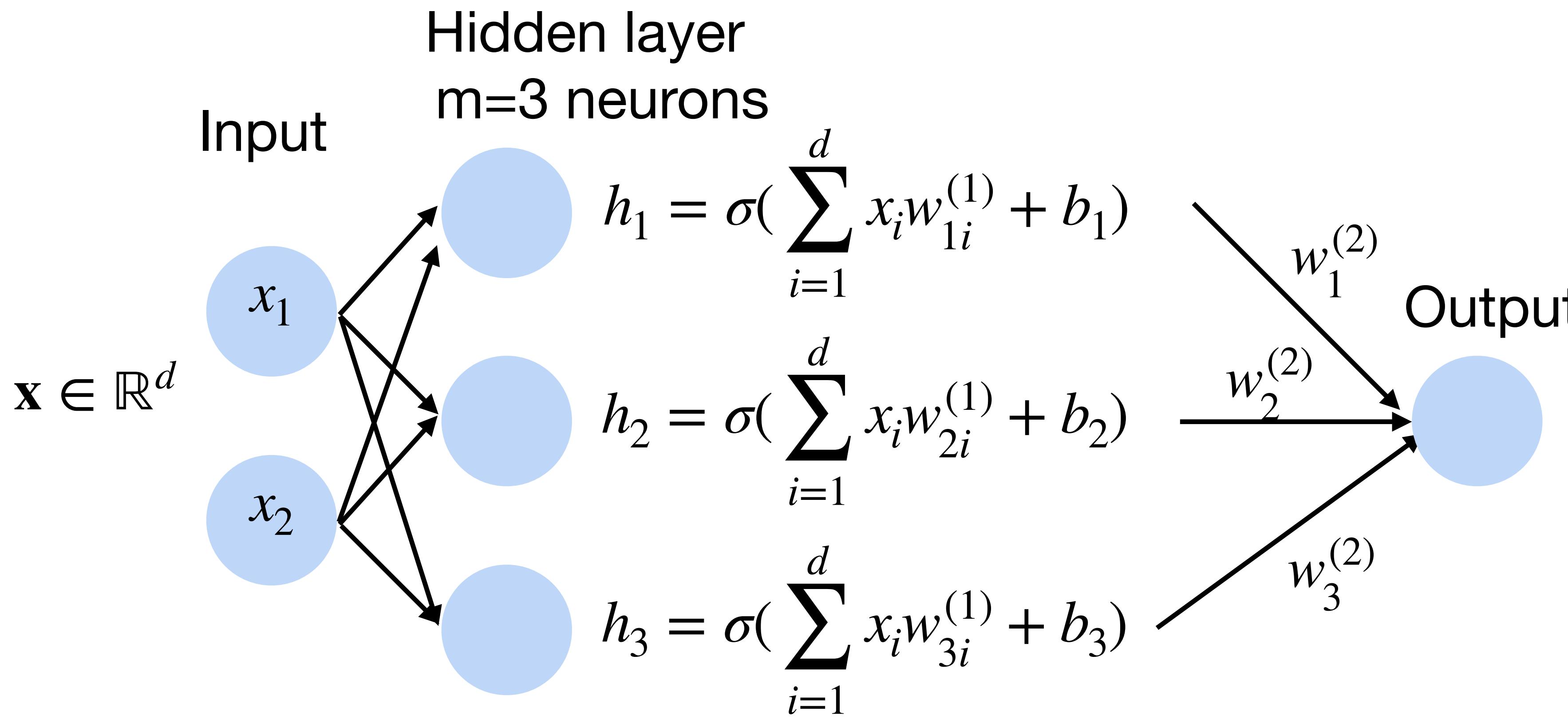
# Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



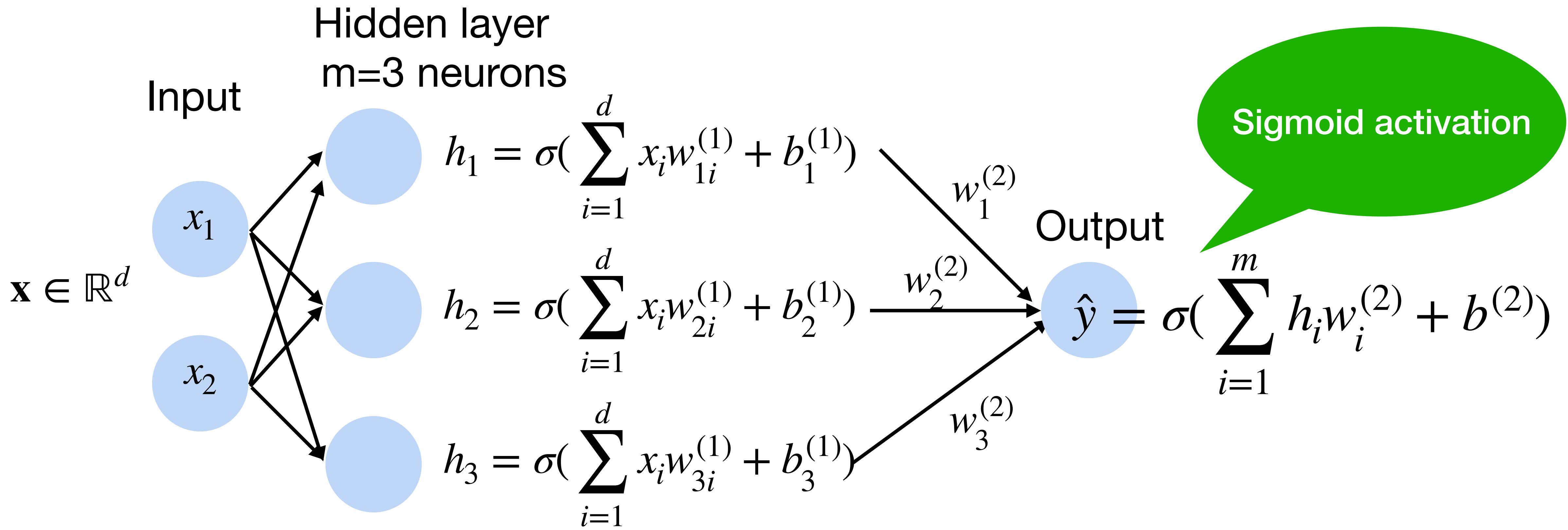
# Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



# Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2

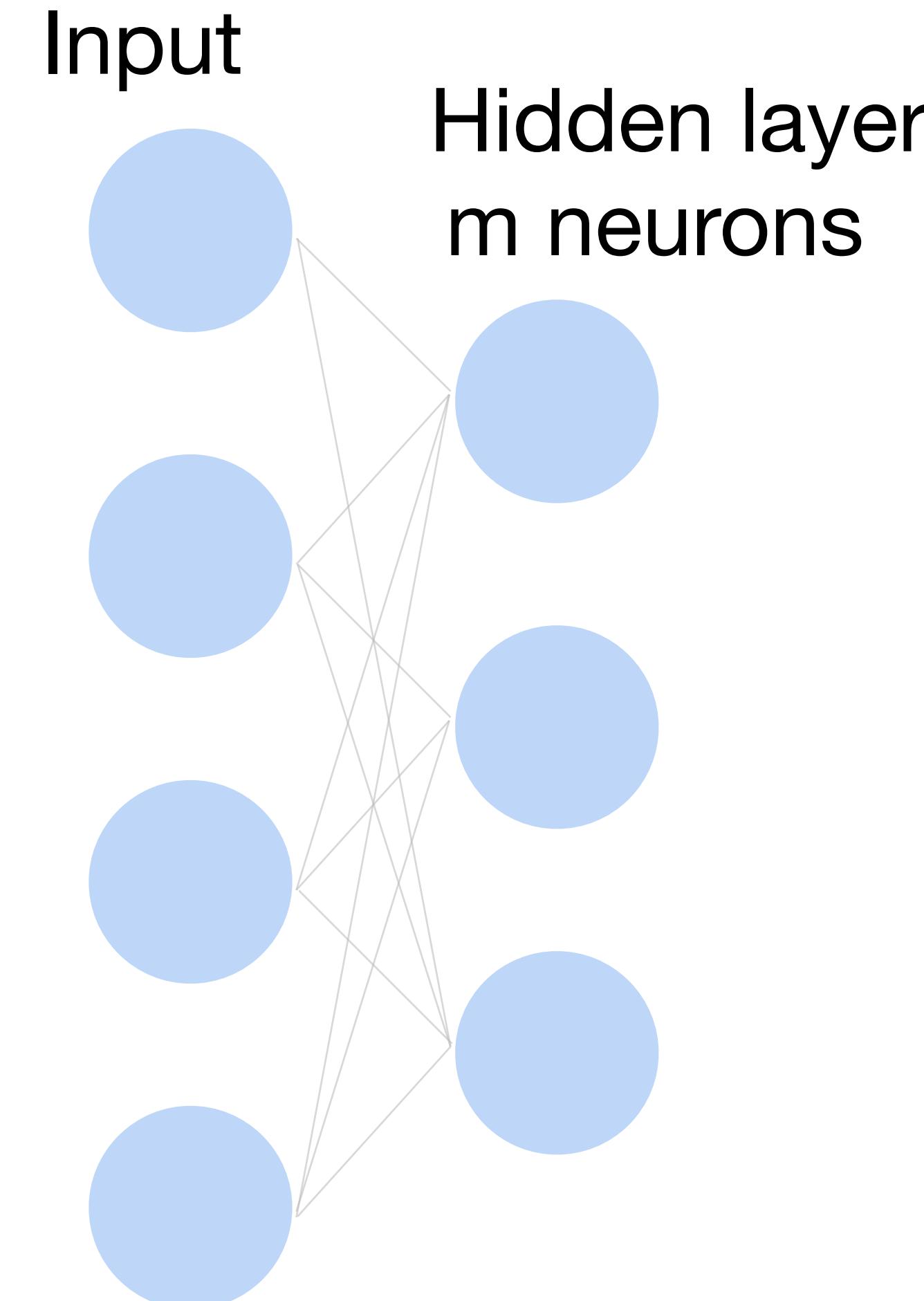


# Multi-layer perceptron: Matrix Notation

- Input  $\mathbf{x} \in \mathbb{R}^d$
- Hidden  $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b}^{(1)} \in \mathbb{R}^m$
- Intermediate output

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

$$\mathbf{h} \in \mathbb{R}^m$$

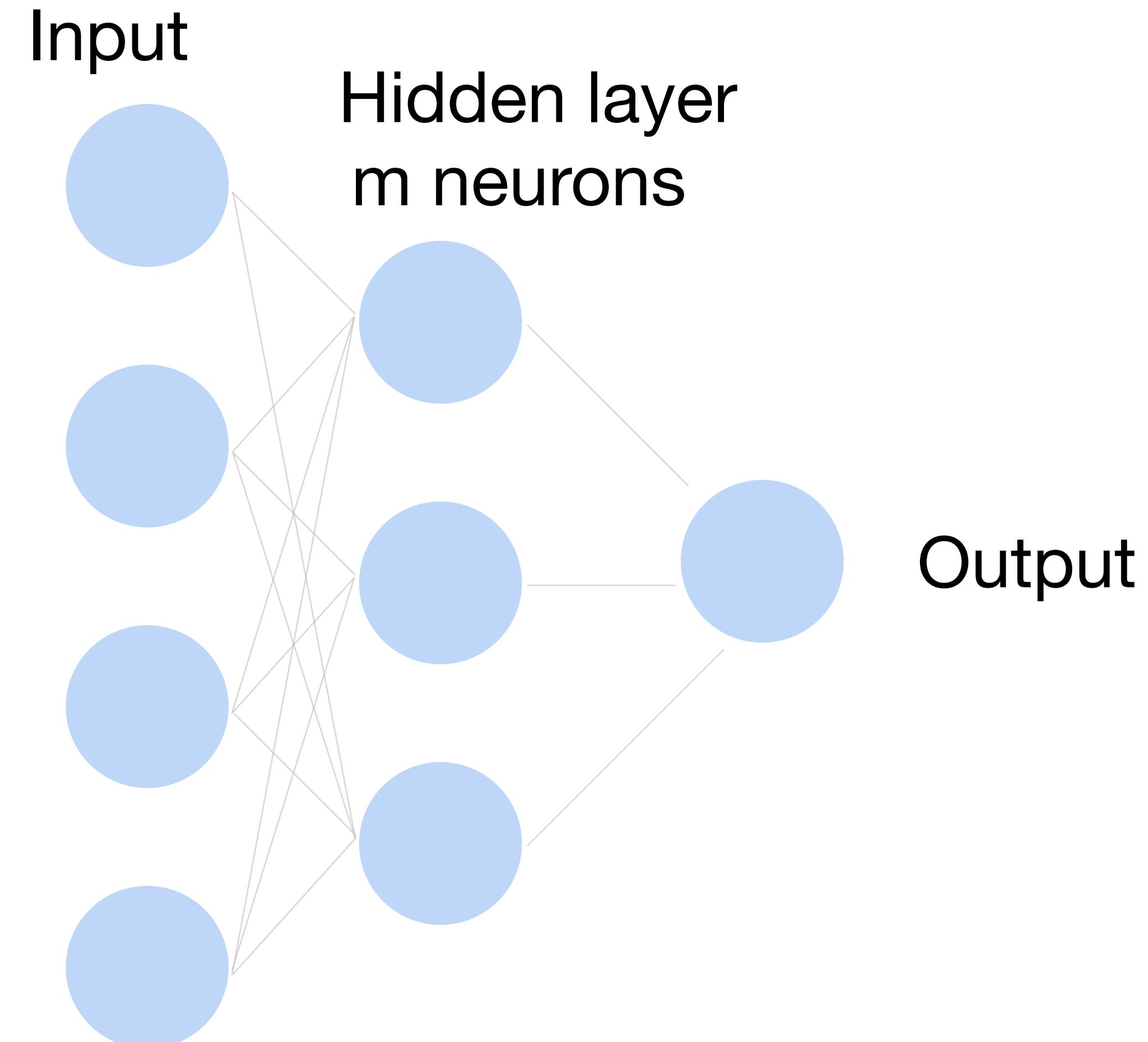


# Multi-layer perceptron: Matrix Notation

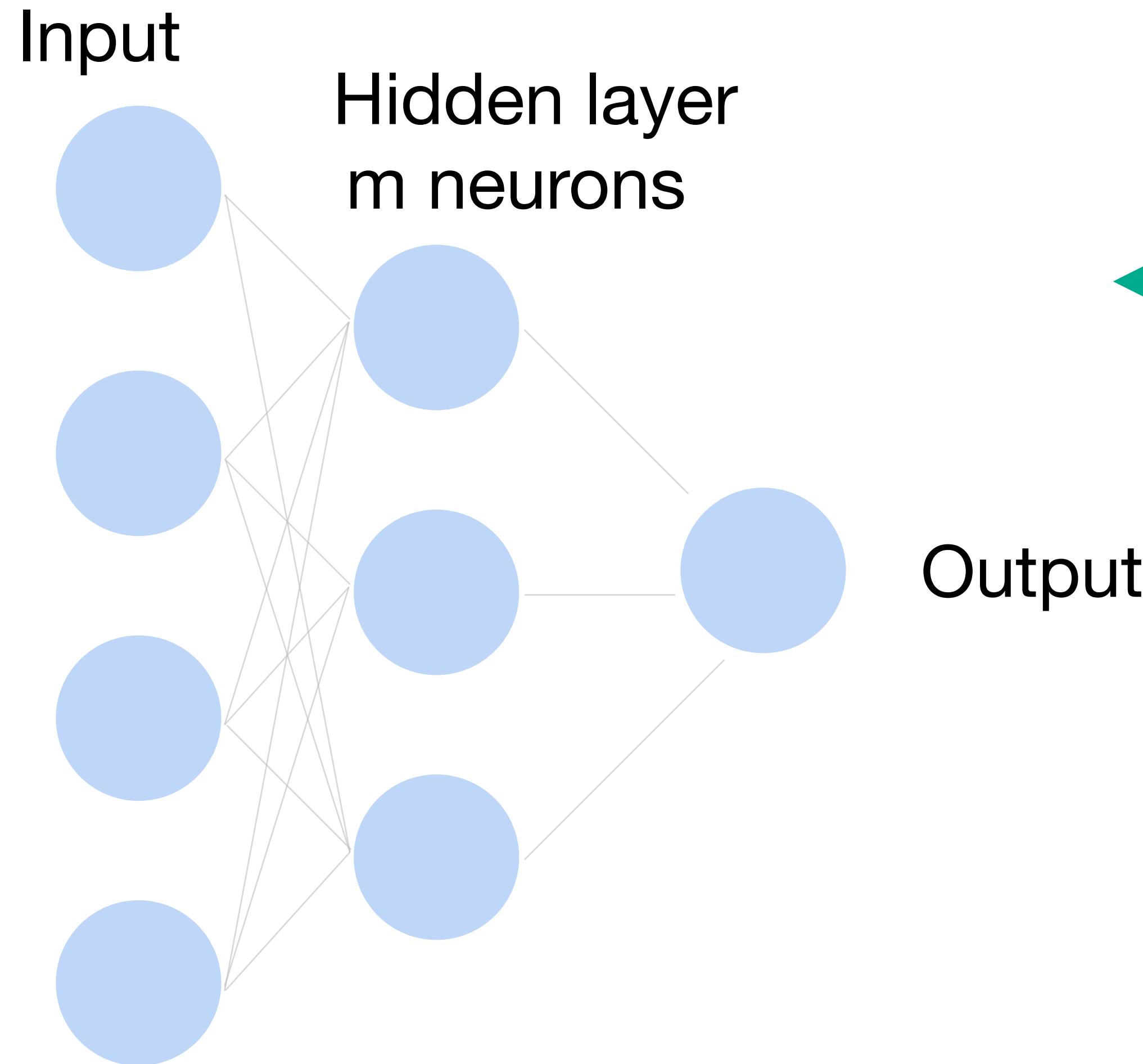
- Input  $\mathbf{x} \in \mathbb{R}^d$
- Hidden  $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b}^{(1)} \in \mathbb{R}^m$
- Intermediate output

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

$$\hat{y} = \sigma(\mathbf{w}^{(2)}\mathbf{h} + \mathbf{b}^{(2)})$$

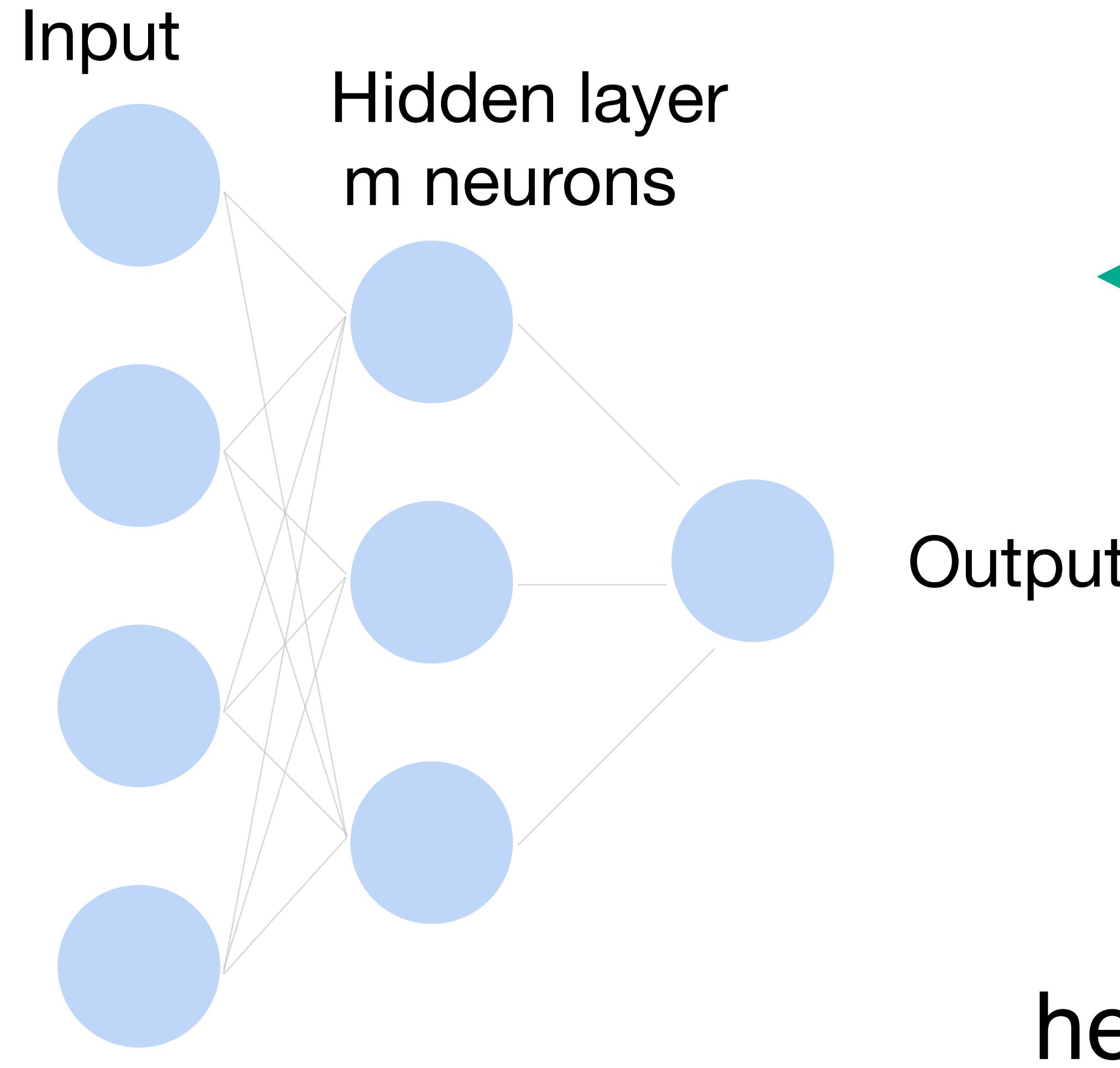


# Multi-layer perceptron



Why do we need a  
nonlinear activation?

# Multi-layer perceptron



Why do we need an a  
nonlinear activation?

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

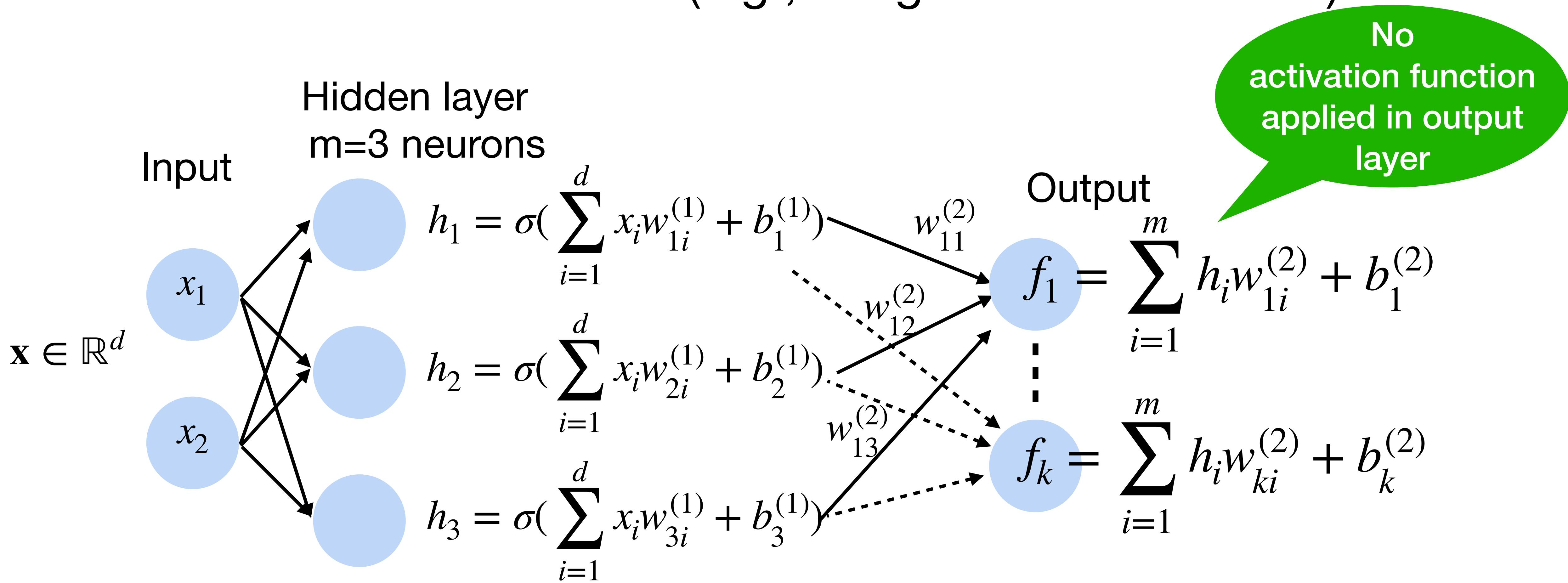
$$f = (\mathbf{w}^{(2)})^T \mathbf{h} + b^{(2)}$$

$$\text{hence } f = (\mathbf{w}^{(2)})^\top \mathbf{W}^{(1)}\mathbf{x} + b'$$

# Neural network for K-way classification

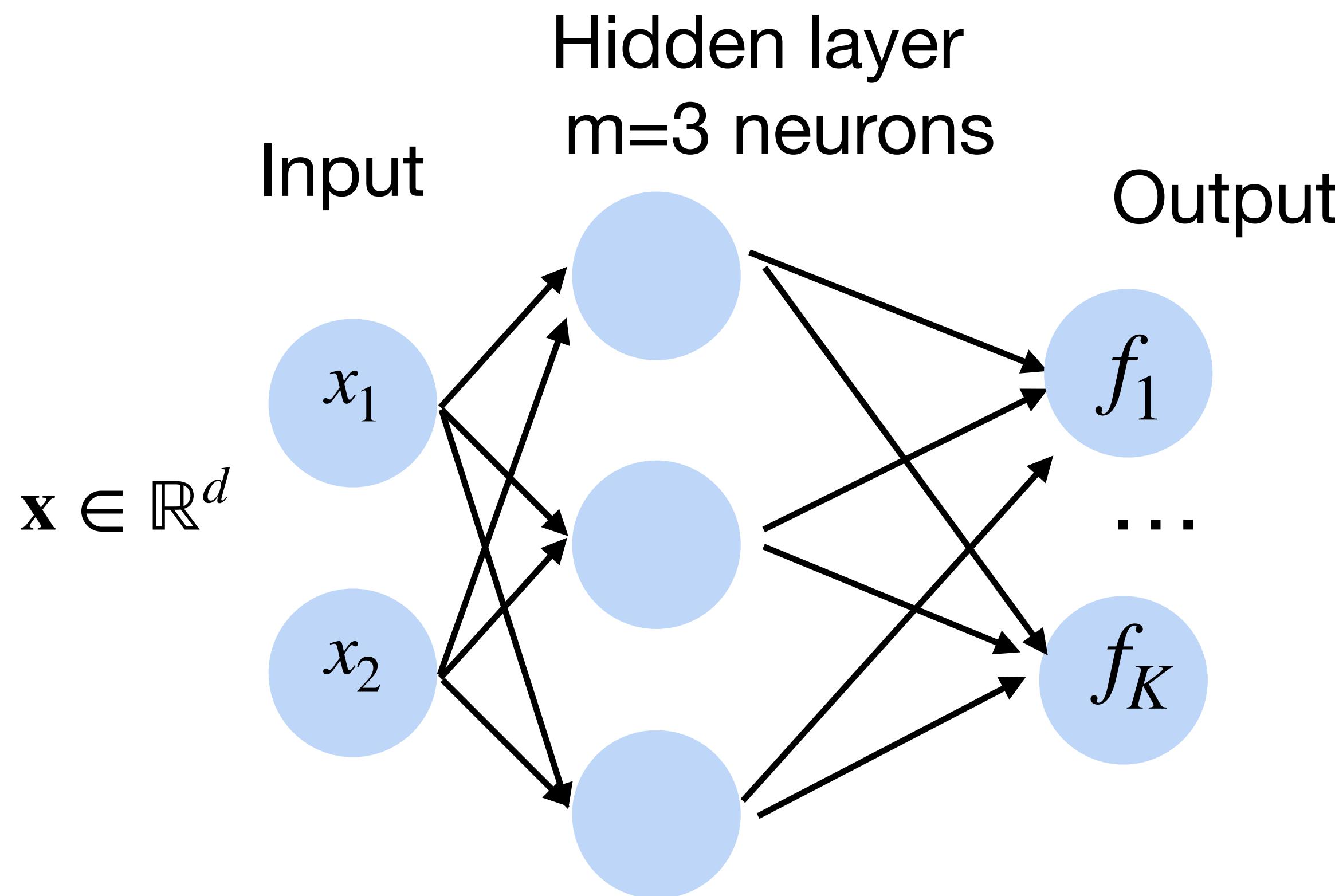
- K outputs in the final layer

**Multi-class classification** (e.g., ImageNet with K=1000)



# Softmax

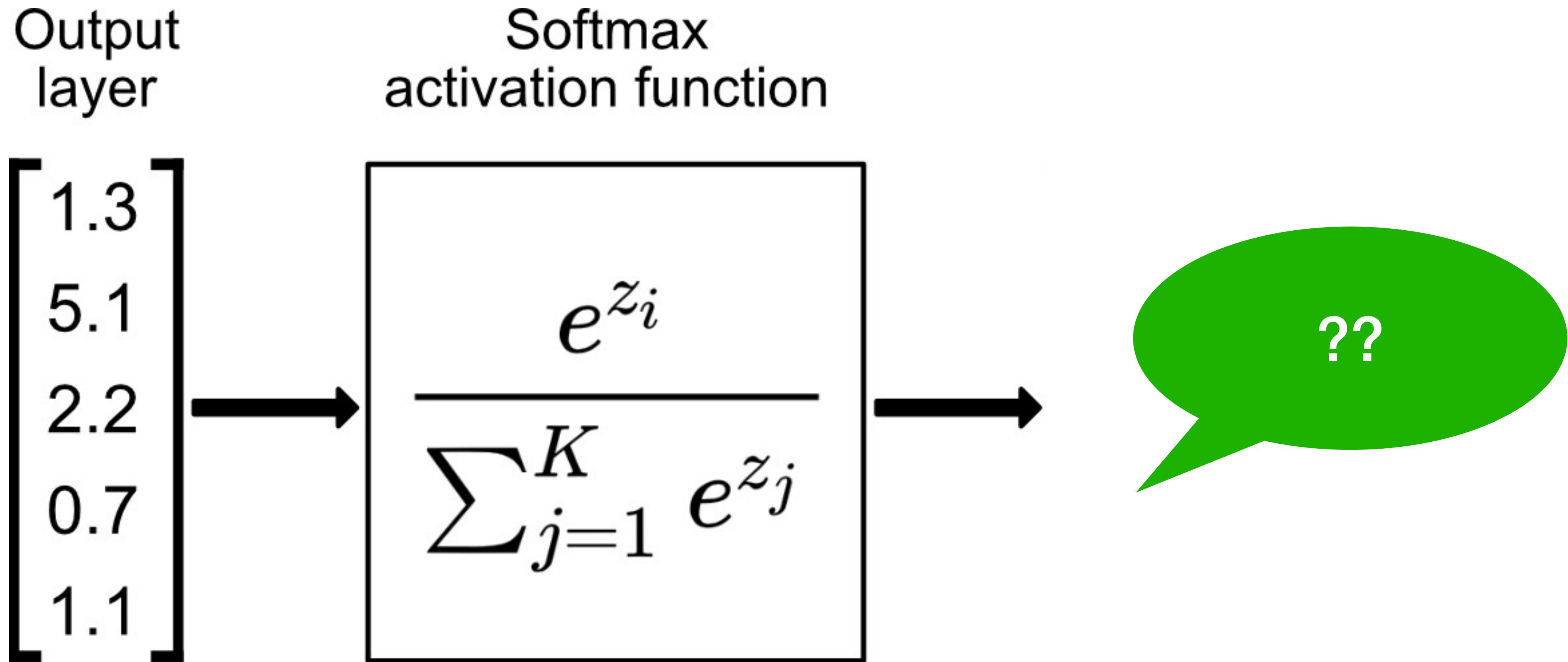
Turns outputs  $f$  into probabilities (sum up to 1 across  $K$  classes)



$$p(y | \mathbf{x}) = \text{softmax}(f) = \frac{\exp(f_y(x))}{\sum_{k=1}^K \exp(f_k(x))}$$

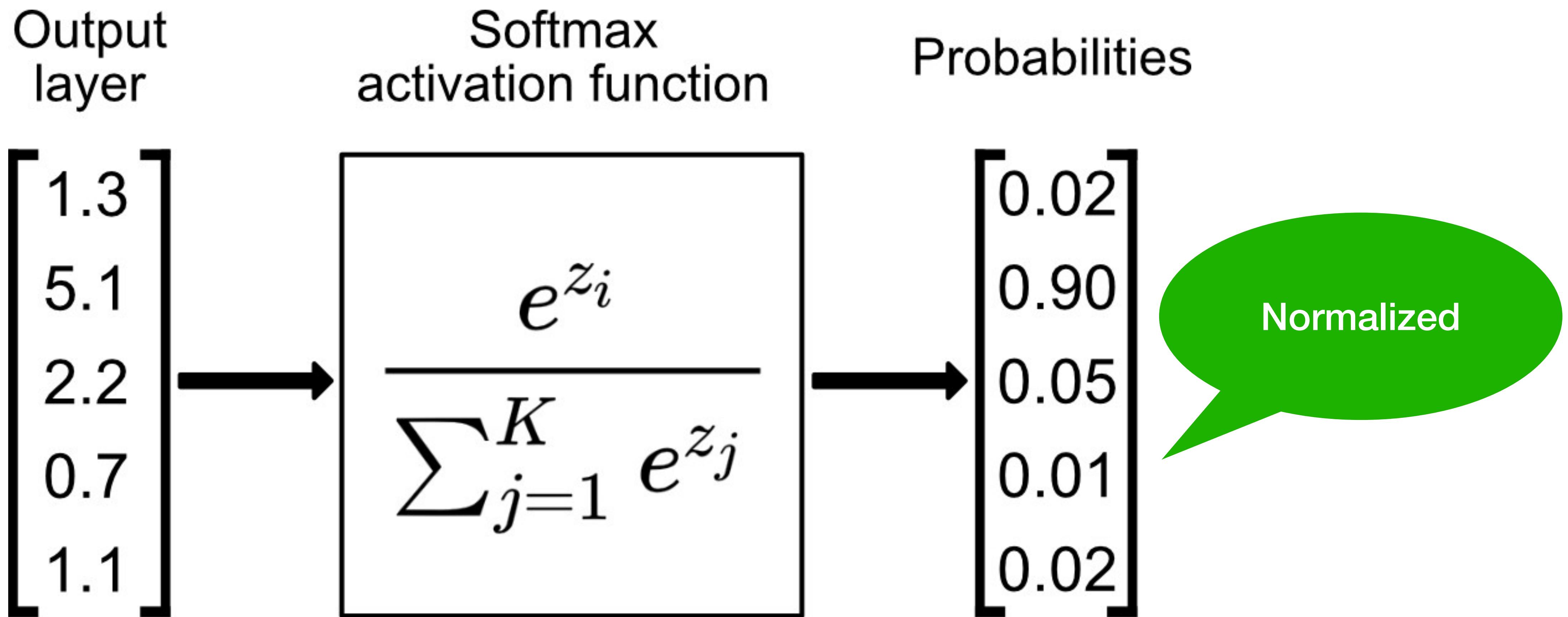
# Softmax

Turns outputs  $f$  into probabilities (sum up to 1 across  $K$  classes)



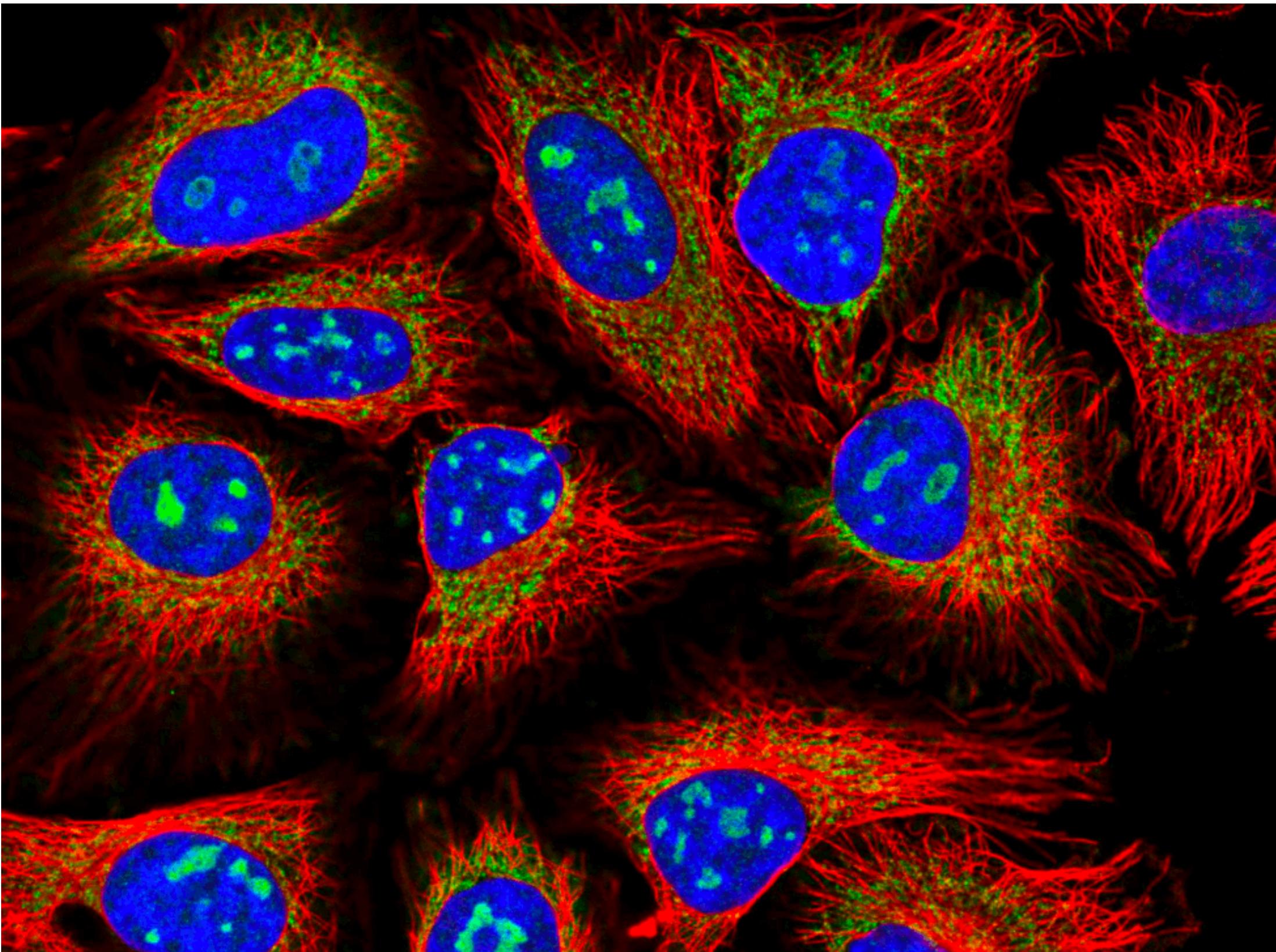
# Softmax

Turns outputs  $f$  into probabilities (sum up to 1 across  $K$  classes)



# Classification Tasks at Kaggle

Classify human protein microscope images into 28 categories

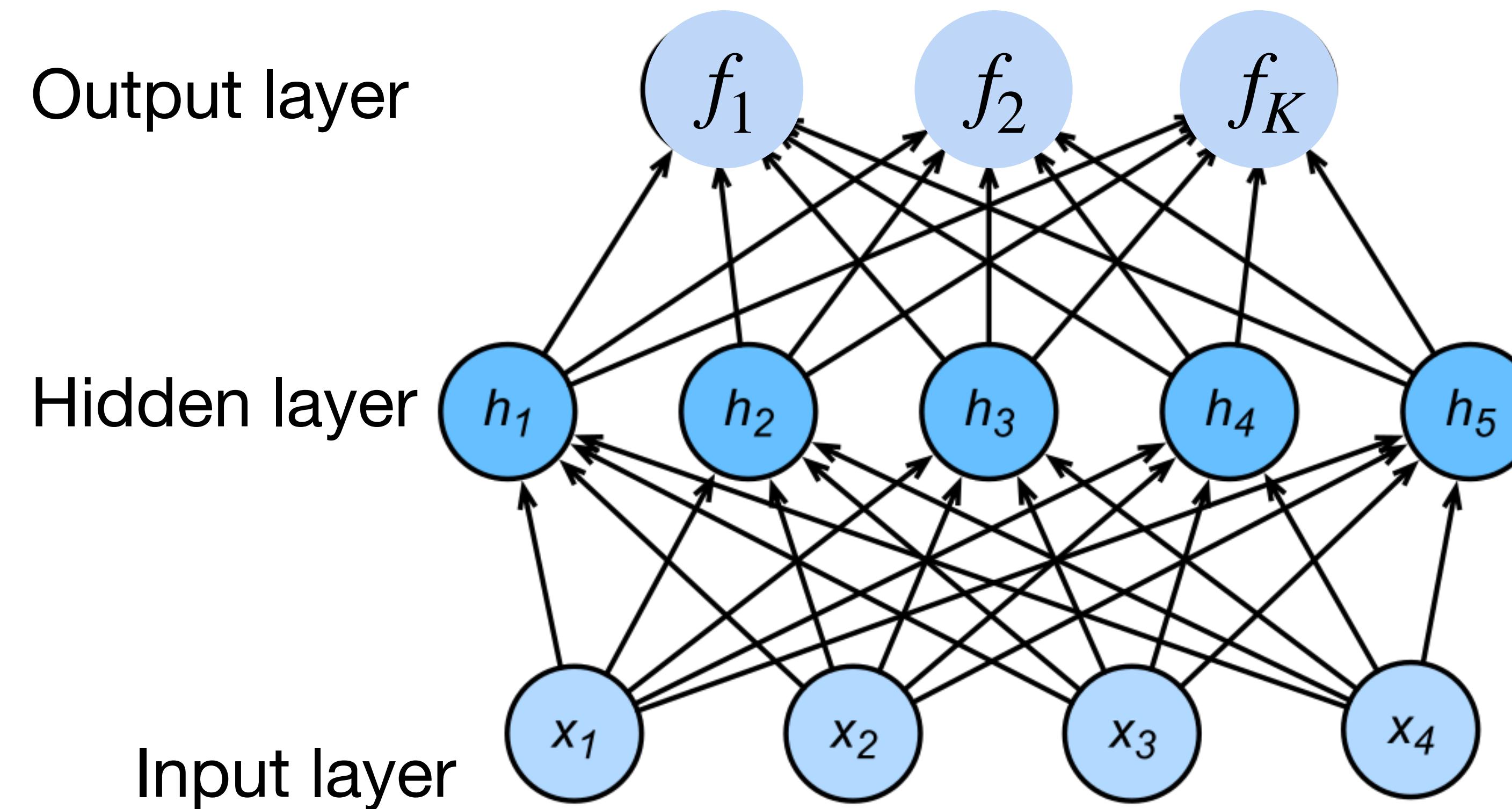


- 0. Nucleoplasm
- 1. Nuclear membrane
- 2. Nucleoli
- 3. Nucleoli fibrillar
- 4. Nuclear speckles
- 5. Nuclear bodies
- 6. Endoplasmic reticu
- 7. Golgi apparatus
- 8. Peroxisomes
- 9. Endosomes
- 10. Lysosomes
- 11. Intermediate fila
- 12. Actin filaments
- 13. Focal adhesion si
- 14. Microtubules
- 15. Microtubule ends
- 16. Cytokinetic brida

<https://www.kaggle.com/c/human-protein-atlas-image-classification>

# More complicated neural networks

$$p_1, p_2, \dots, p_K = \text{softmax}(f_1, f_2, \dots, f_K)$$



# More complicated neural networks

- Input  $\mathbf{x} \in \mathbb{R}^d$

- Hidden  $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b}^{(1)} \in \mathbb{R}^m$

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

$$\mathbf{f} = \sigma(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)})$$

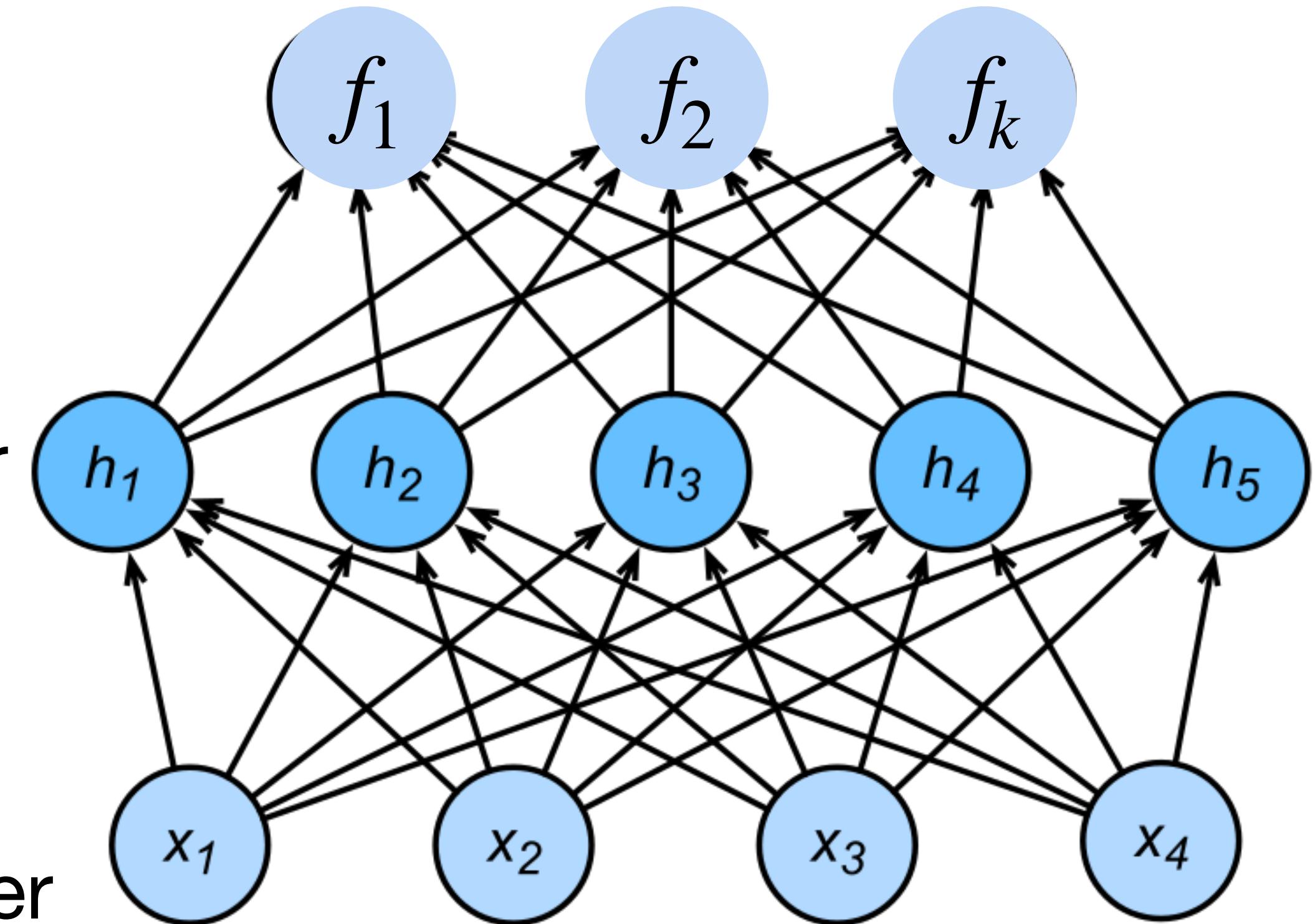
$$\mathbf{p} = \text{softmax}(\mathbf{f})$$

$$p_1, p_2, \dots, p_K = \text{softmax}(f_1, f_2, \dots, f_K)$$

Output layer

Hidden layer

Input layer



# More complicated neural networks: multiple hidden layers

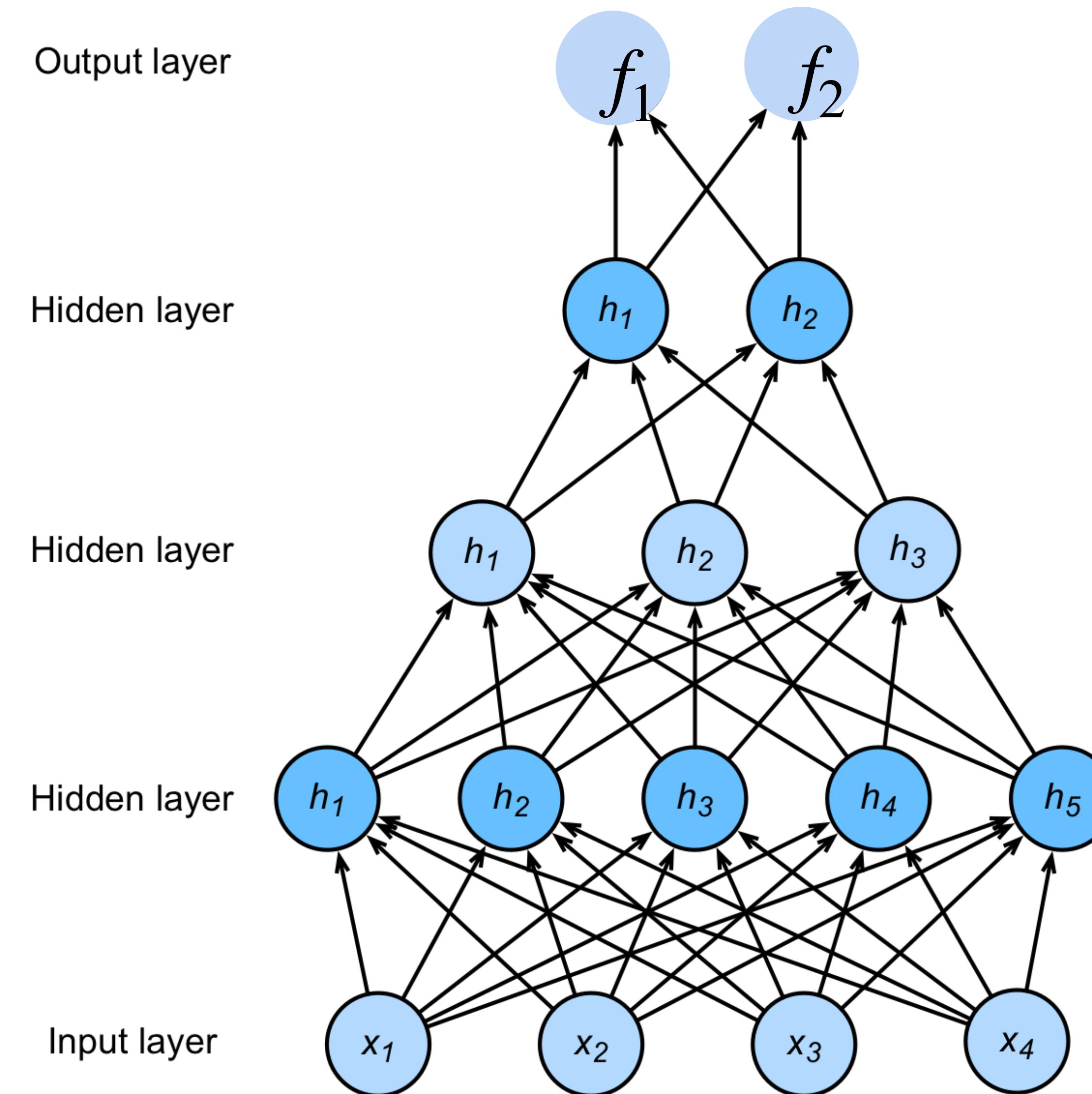
$$\mathbf{h}_1 = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\mathbf{h}_2 = \sigma(\mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)})$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}^{(3)}\mathbf{h}_2 + \mathbf{b}^{(3)})$$

$$\mathbf{f} = \mathbf{W}^{(4)}\mathbf{h}_3 + \mathbf{b}^{(4)}$$

$$\mathbf{p} = \text{softmax}(\mathbf{f})$$



# Quiz Break

Which output function is often used for multi-class classification tasks?

- A Sigmoid function
- B Rectified Linear Unit (ReLU)
- C Softmax function
- D Max function

# Quiz Break

Which output function is often used for multi-class classification tasks?

- A Sigmoid function
- B Rectified Linear Unit (ReLU)
- C Softmax function
- D Max function

# Quiz Break

Suppose you are given a 3-layer multilayer perceptron (2 hidden layers  $h_1$  and  $h_2$  and 1 output layer). All activation functions are sigmoids, and the output layer uses a softmax function. Suppose  $h_1$  has 1024 units and  $h_2$  has 512 units. Given a dataset with 2 input features and 3 unique class labels, how many learnable parameters does the perceptron have in total?

# Quiz Break

Suppose you are given a 3-layer multilayer perceptron (2 hidden layers  $h_1$  and  $h_2$  and 1 output layer). All activation functions are sigmoids, and the output layer uses a softmax function. Suppose  $h_1$  has 1024 units and  $h_2$  has 512 units. Given a dataset with 2 input features and 3 unique class labels, how many learnable parameters does the perceptron have in total?

$$1024 * 2 + 1024 + 512 * 1024 + 512 + 512 * 3 + 3 = 529411$$

# Quiz Break

Consider a three-layer network with **linear Perceptrons** for binary classification. The hidden layer has 3 neurons. Can the network represent a XOR problem?

- a) Yes
- b) No

# Quiz Break

Consider a three-layer network with **linear Perceptrons** for binary classification. The hidden layer has 3 neurons. Can the network represent a XOR problem?

a) Yes

b) No

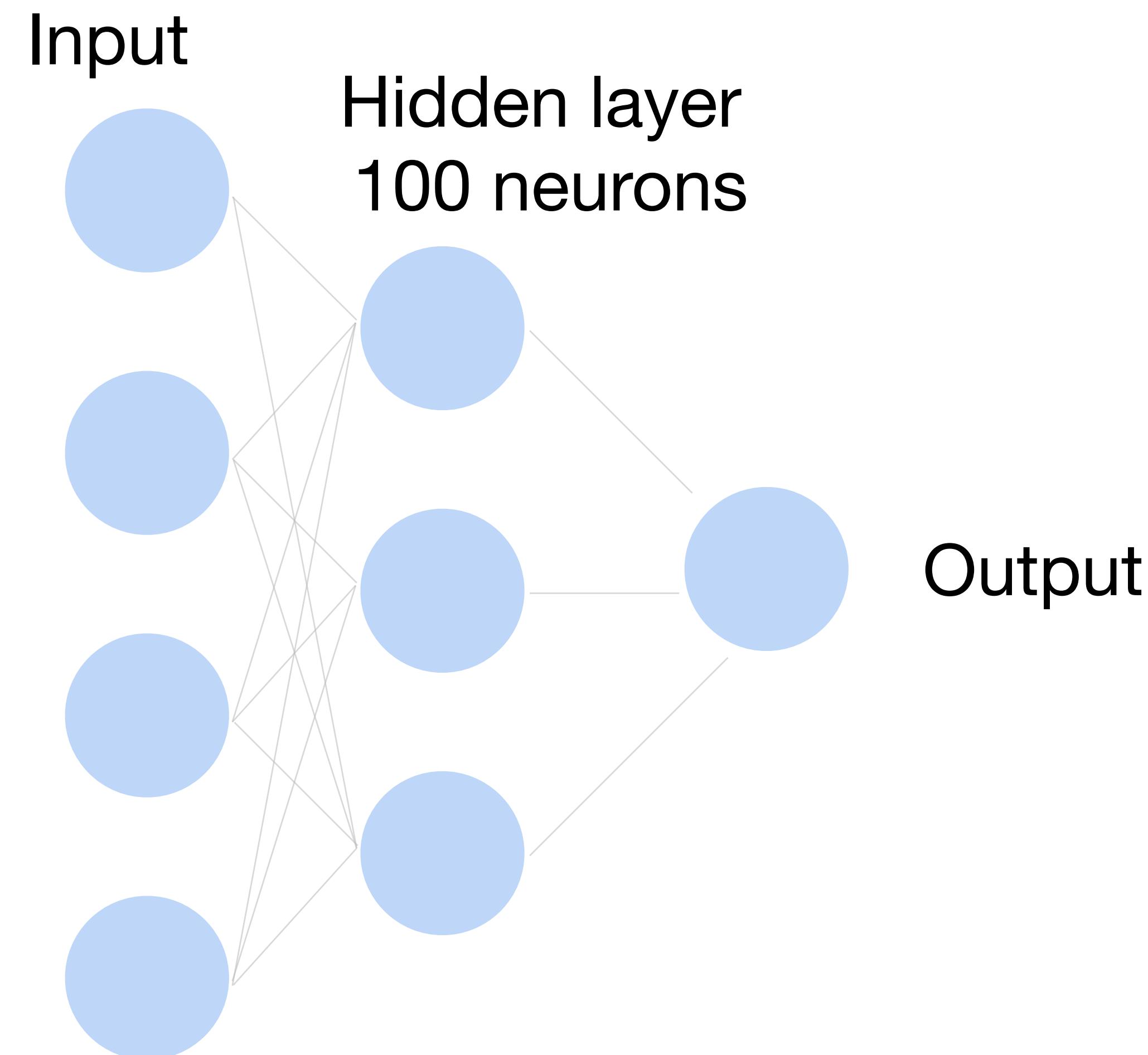


Solution:

A combination of linear Perceptrons is still a linear function.

# How to train a neural network?

**Classify cats vs. dogs**



# How to train a neural network? Binary classification

$x \in \mathbb{R}^d$  One training data point in the training set D

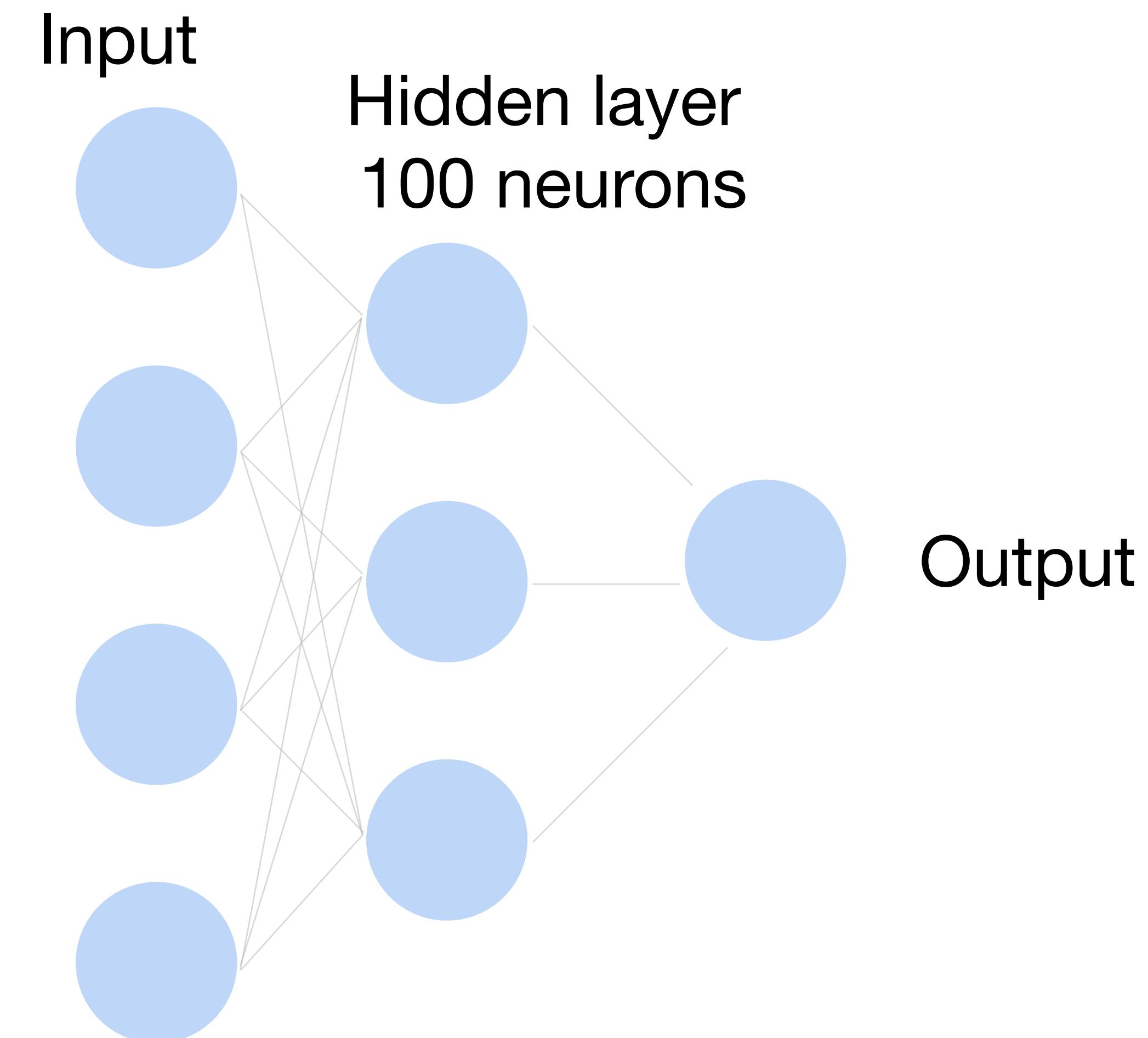
$\hat{y} \in [0,1]$  Model output for example x

(This is a function of all weights W)

y Ground truth label for example x

**Learning by matching the output  
to the label**

We want  $\hat{y} \rightarrow 1$  when  $y = 1$ ,  
and  $\hat{y} \rightarrow 0$  when  $y = 0$

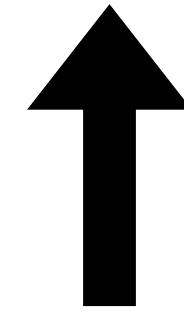


# How to train a neural network? Binary classification

Loss function:  $\frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \ell(\mathbf{x}, y)$

Per-sample loss:

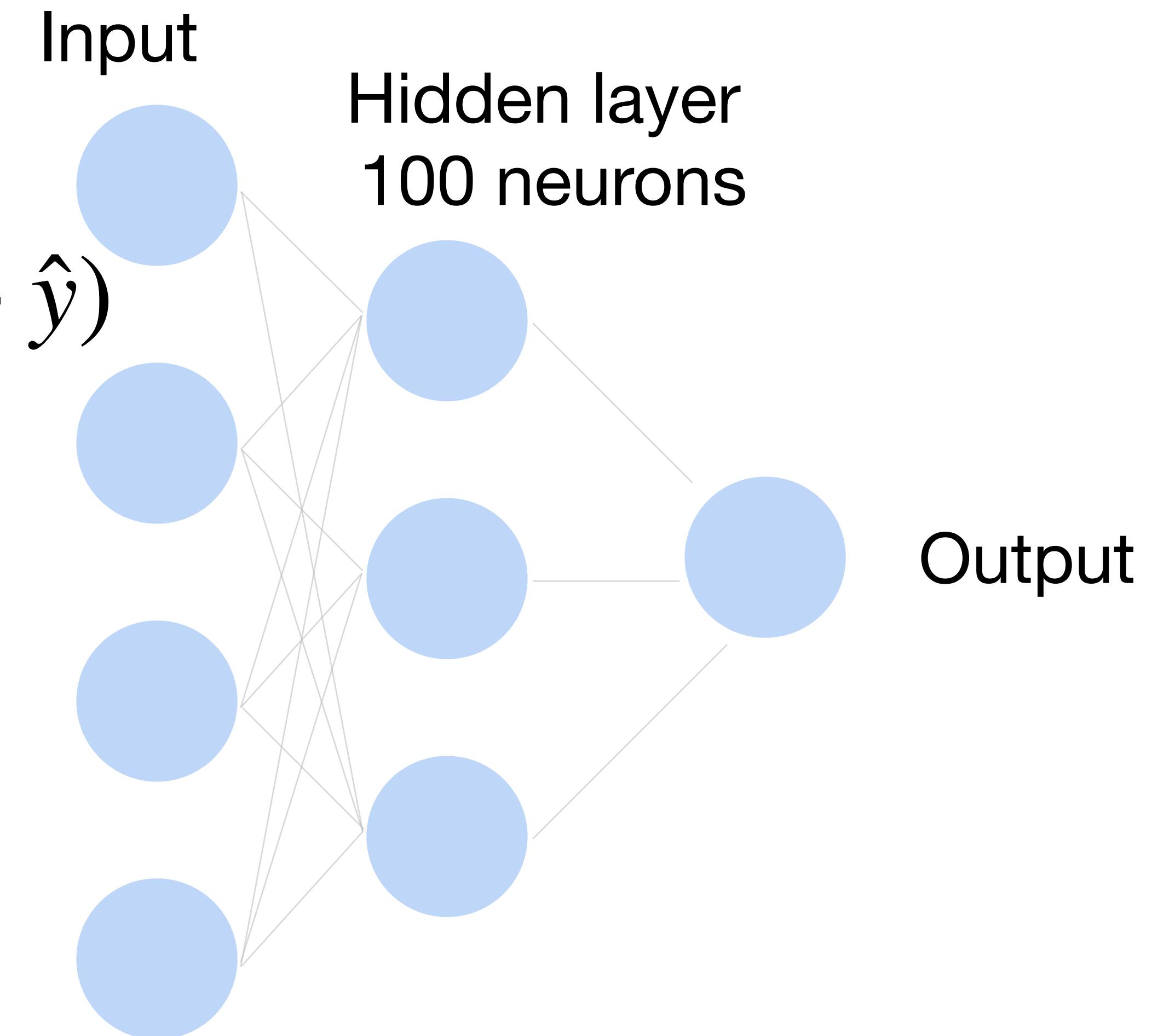
$$\ell(\mathbf{x}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$



Negative log likelihood

Minimizing NLL is equivalent to Max Likelihood Learning (MLE)

Also known as **binary cross-entropy**



# How to train a neural network? Multiclass

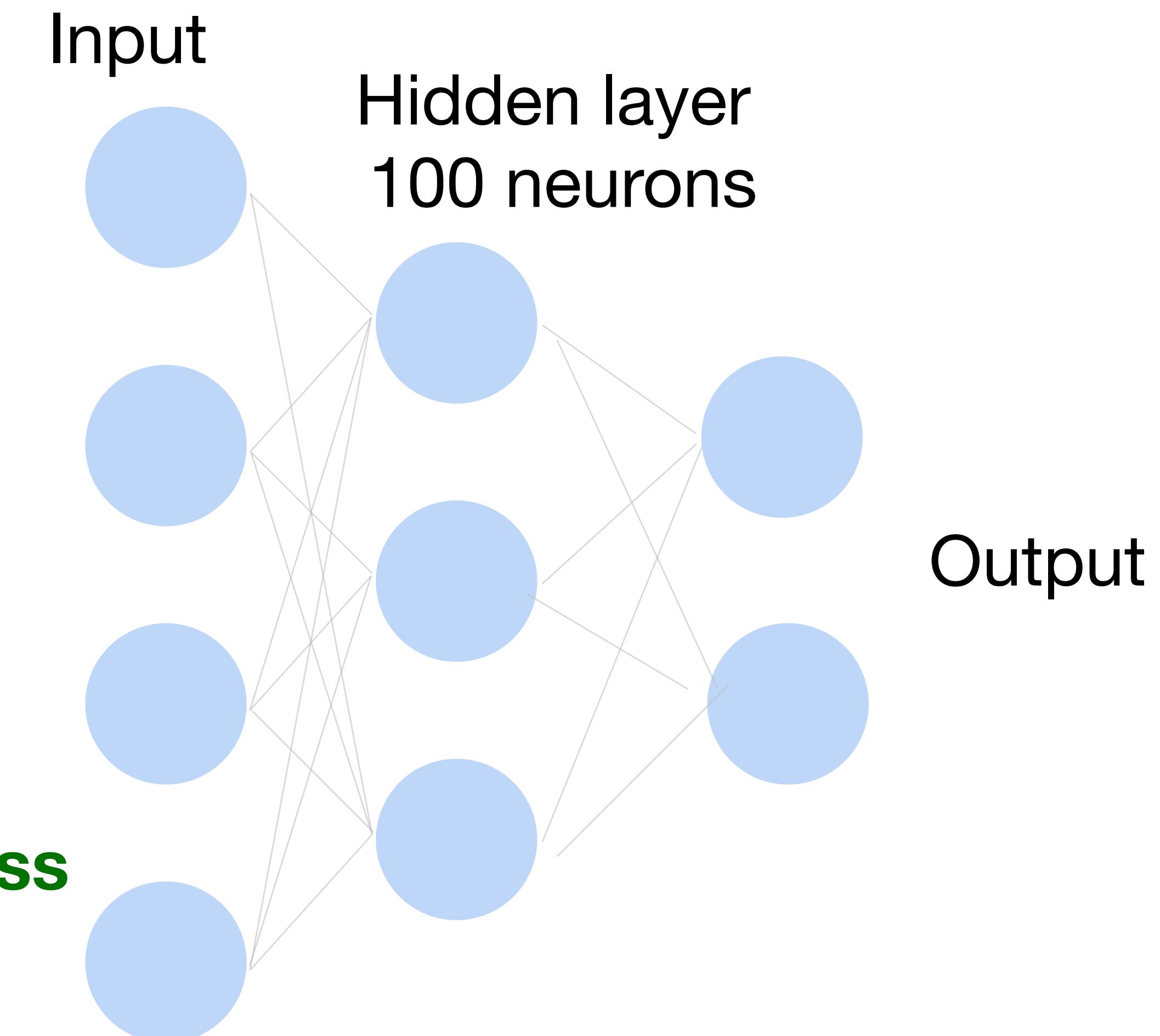
Loss function:  $\frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \ell(\mathbf{x}, y)$

Per-sample loss:

$$\ell(\mathbf{x}, y) = \sum_{k=1}^K -Y_k \log p_k = -\log p_y$$

where  $Y$  is one-hot encoding of  $y$

Also known as **cross-entropy loss**  
or **softmax loss**

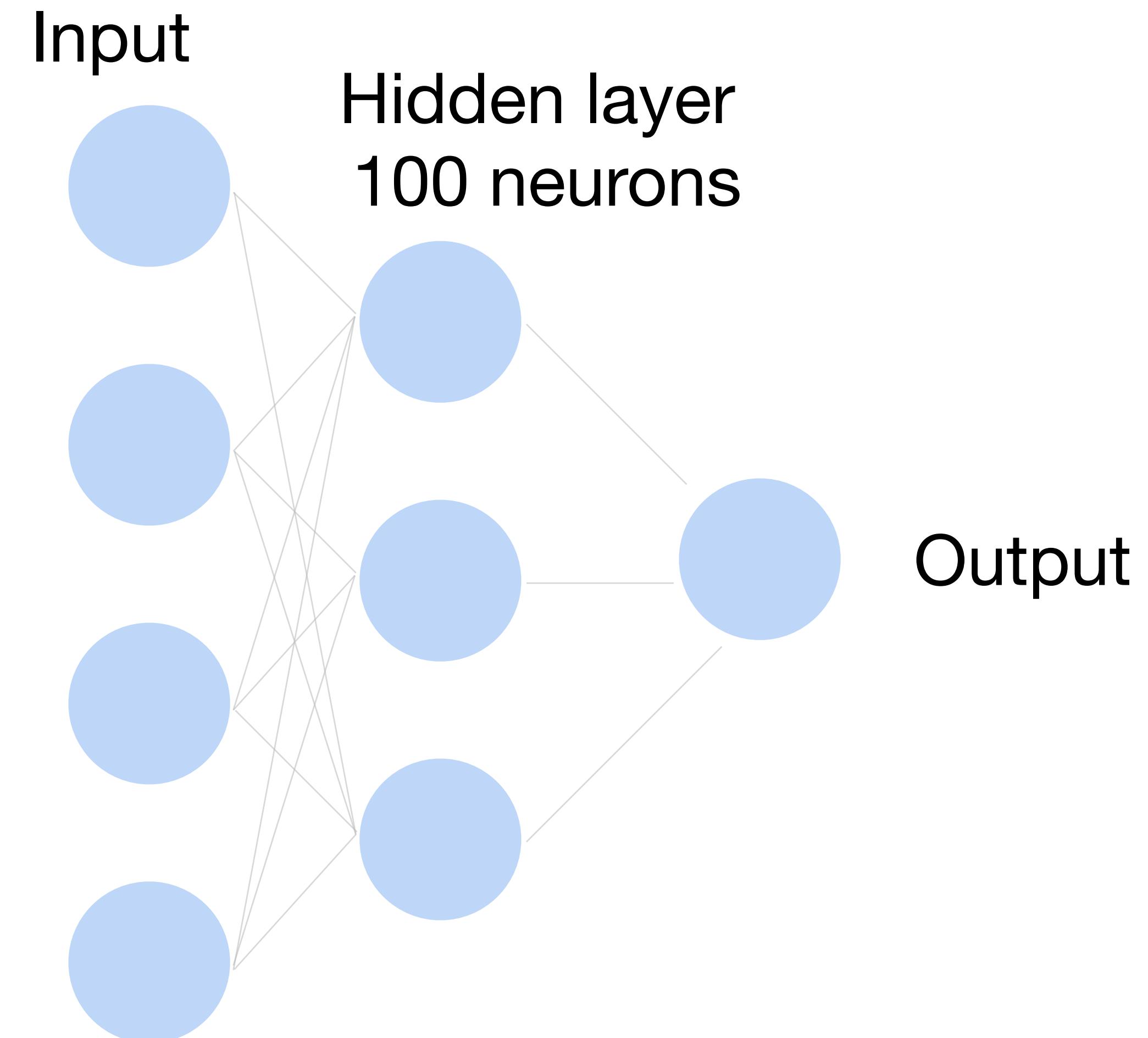


# How to train a neural network?

Update the weights  $W$  to minimize the loss function

$$L = \frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \ell(\mathbf{x}, y)$$

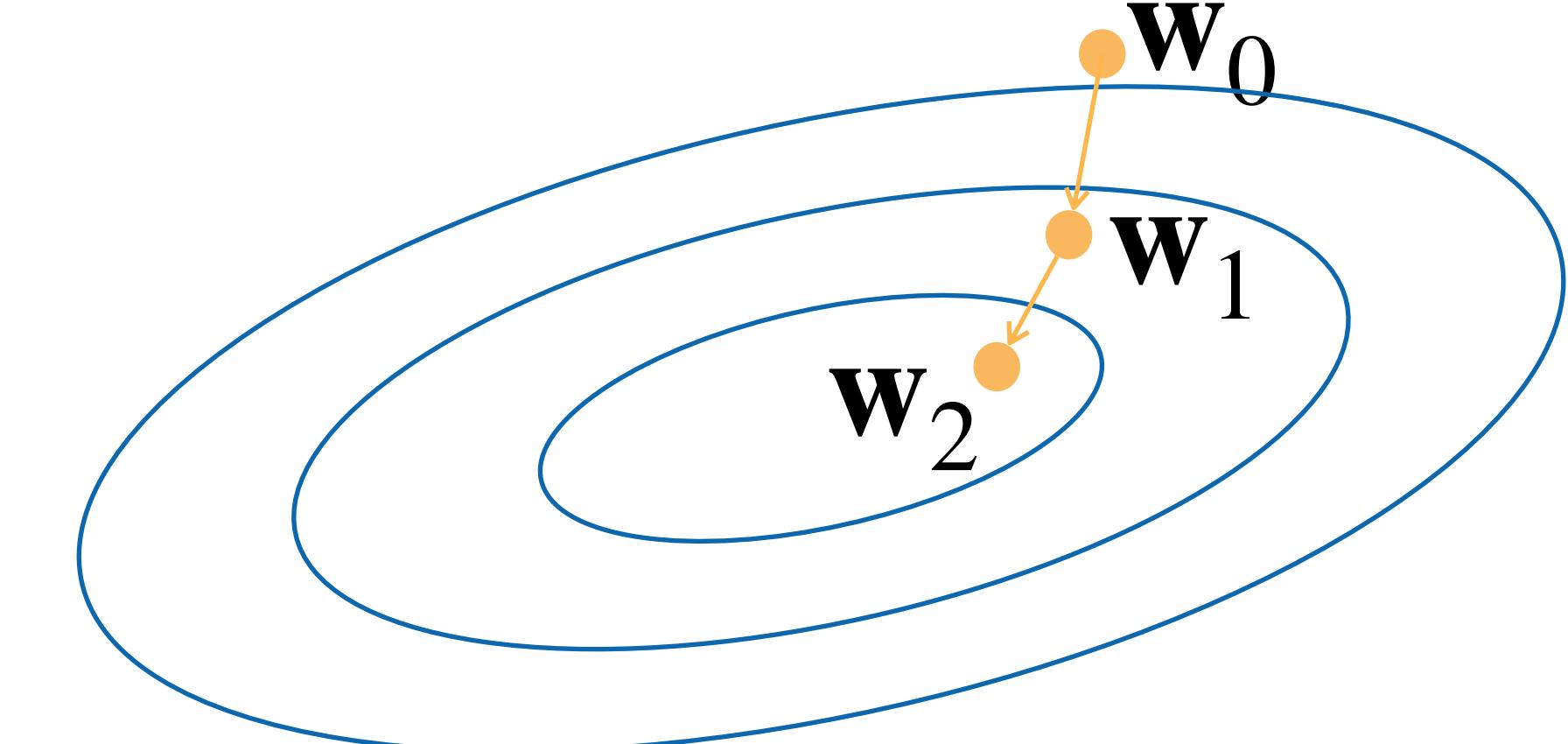
Use gradient descent!



# Gradient Descent

- Choose a learning rate  $\alpha > 0$
- Initialize the model parameters  $w_0$
- For  $t = 1, 2, \dots$ 
  - Update parameters:

$$\begin{aligned} w_t &= w_{t-1} - \alpha \frac{\partial L}{\partial w_{t-1}} \\ &= w_{t-1} - \alpha \frac{1}{|D|} \sum_{(x,y) \in D} \frac{\partial \ell(x, y)}{\partial w_{t-1}} \end{aligned}$$



D can  
be very large.  
Expensive

- Repeat until converges

The gradient w.r.t. all parameters is obtained by concatenating the partial derivatives w.r.t. each parameter

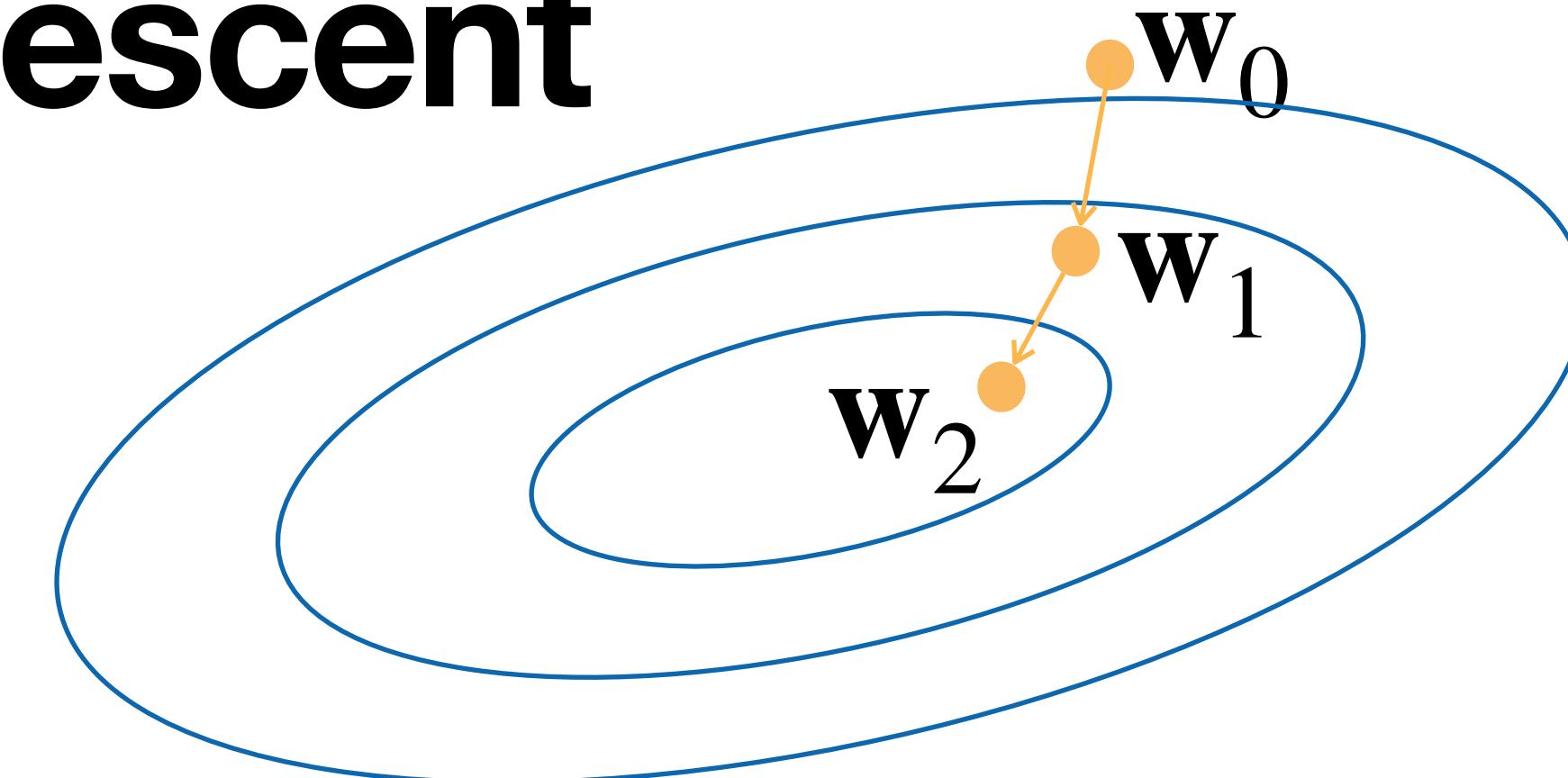
# Minibatch Stochastic Gradient Descent

- Choose a learning rate  $\alpha > 0$
- Initialize the model parameters  $w_0$
- For  $t = 1, 2, \dots$

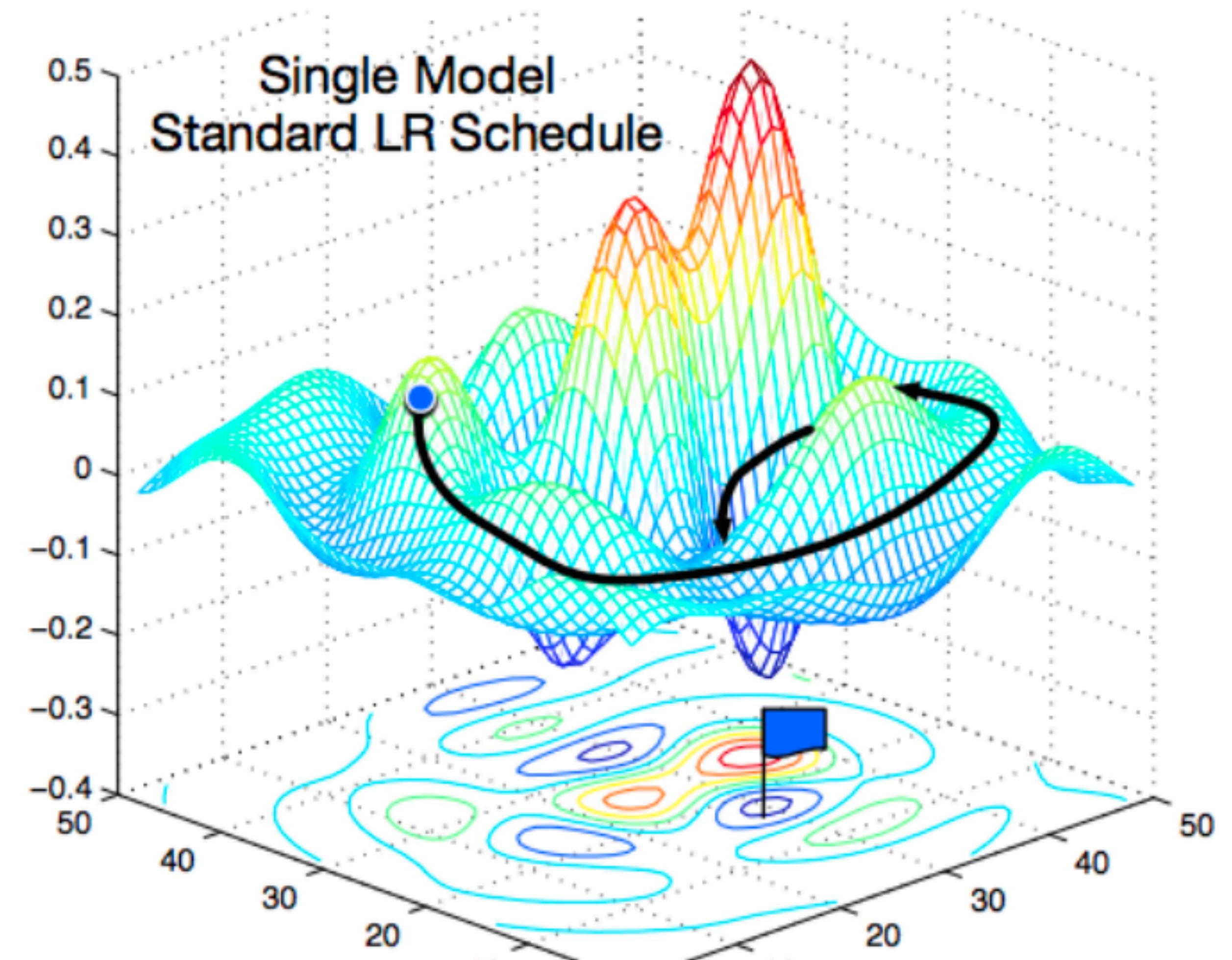
- Randomly sample a subset (mini-batch)  $B \subset D$   
Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \frac{1}{|B|} \sum_{(\mathbf{x}, y) \in B} \frac{\partial \ell(\mathbf{x}, y)}{\partial \mathbf{w}_{t-1}}$$

- Repeat until converges

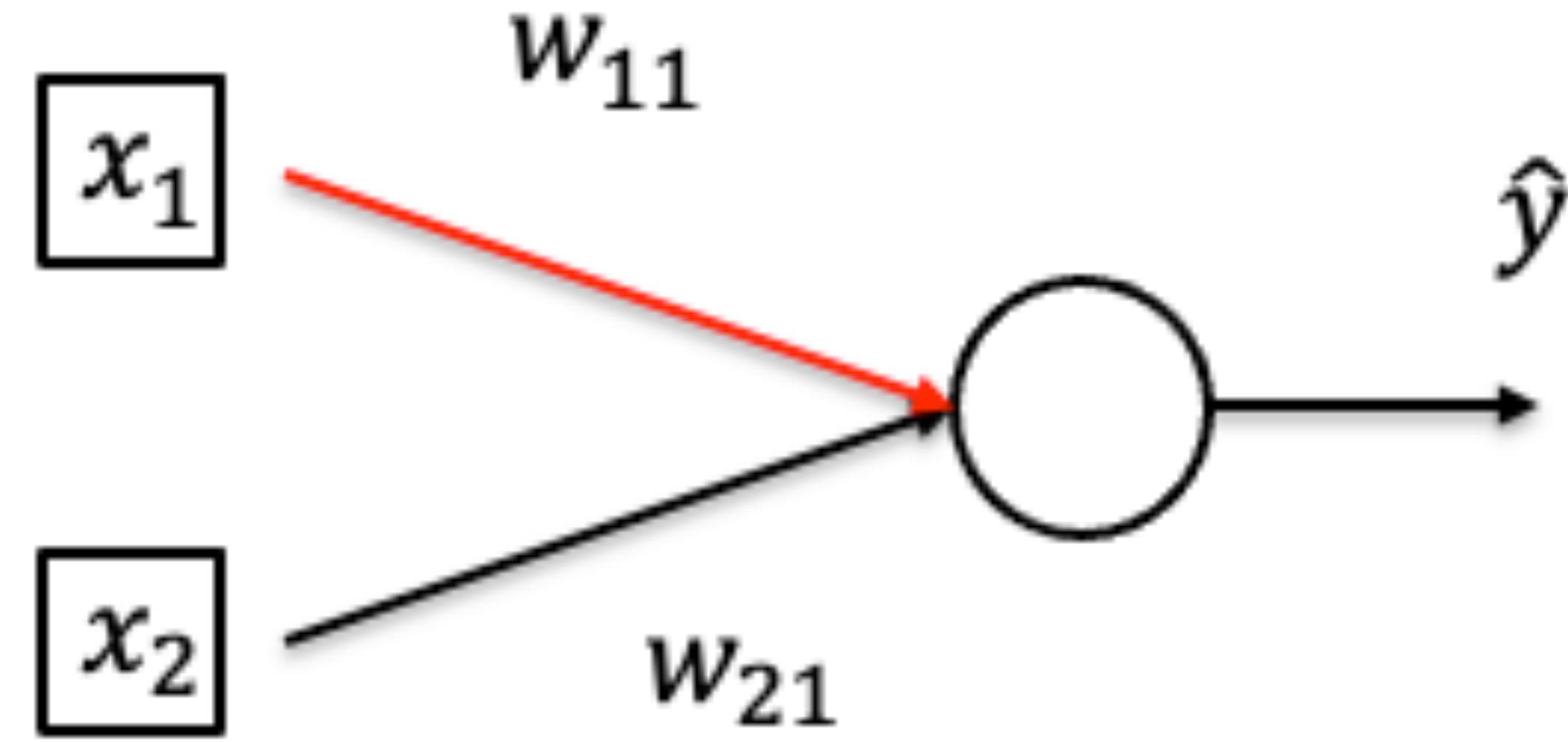


# Non-convex Optimization



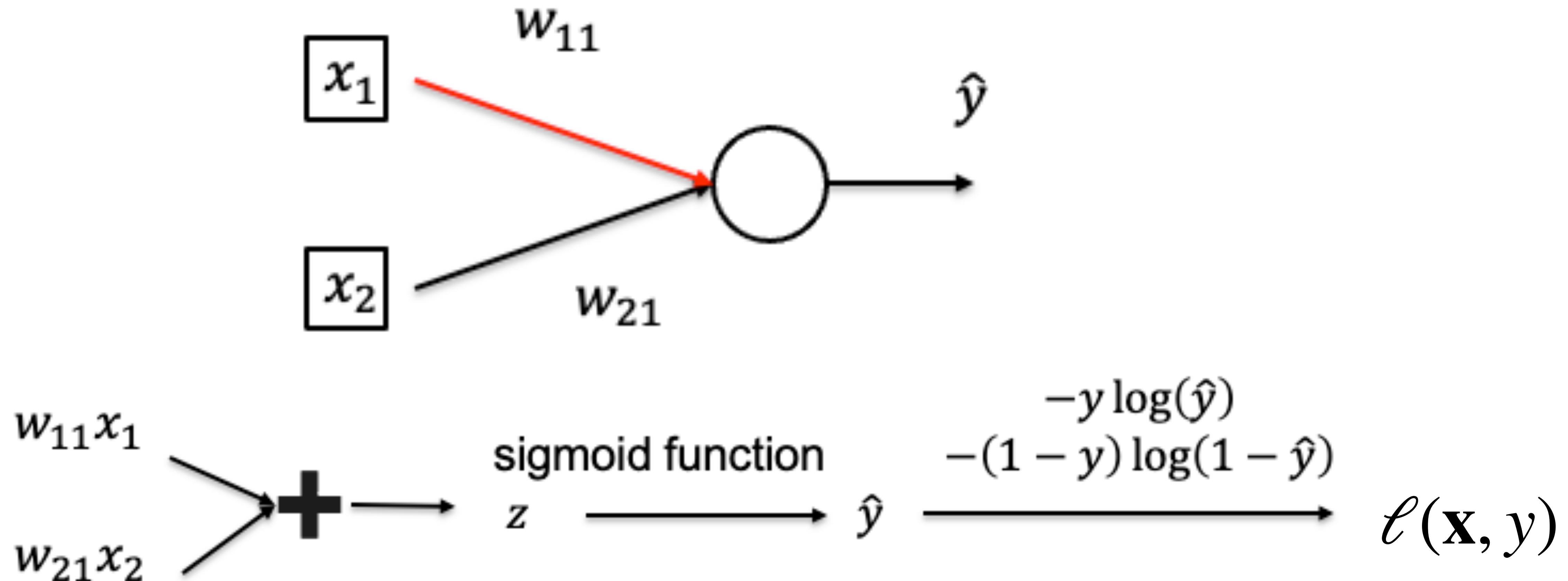
[Gao and Li et al., 2018]

# Calculate Gradient (on one data point)

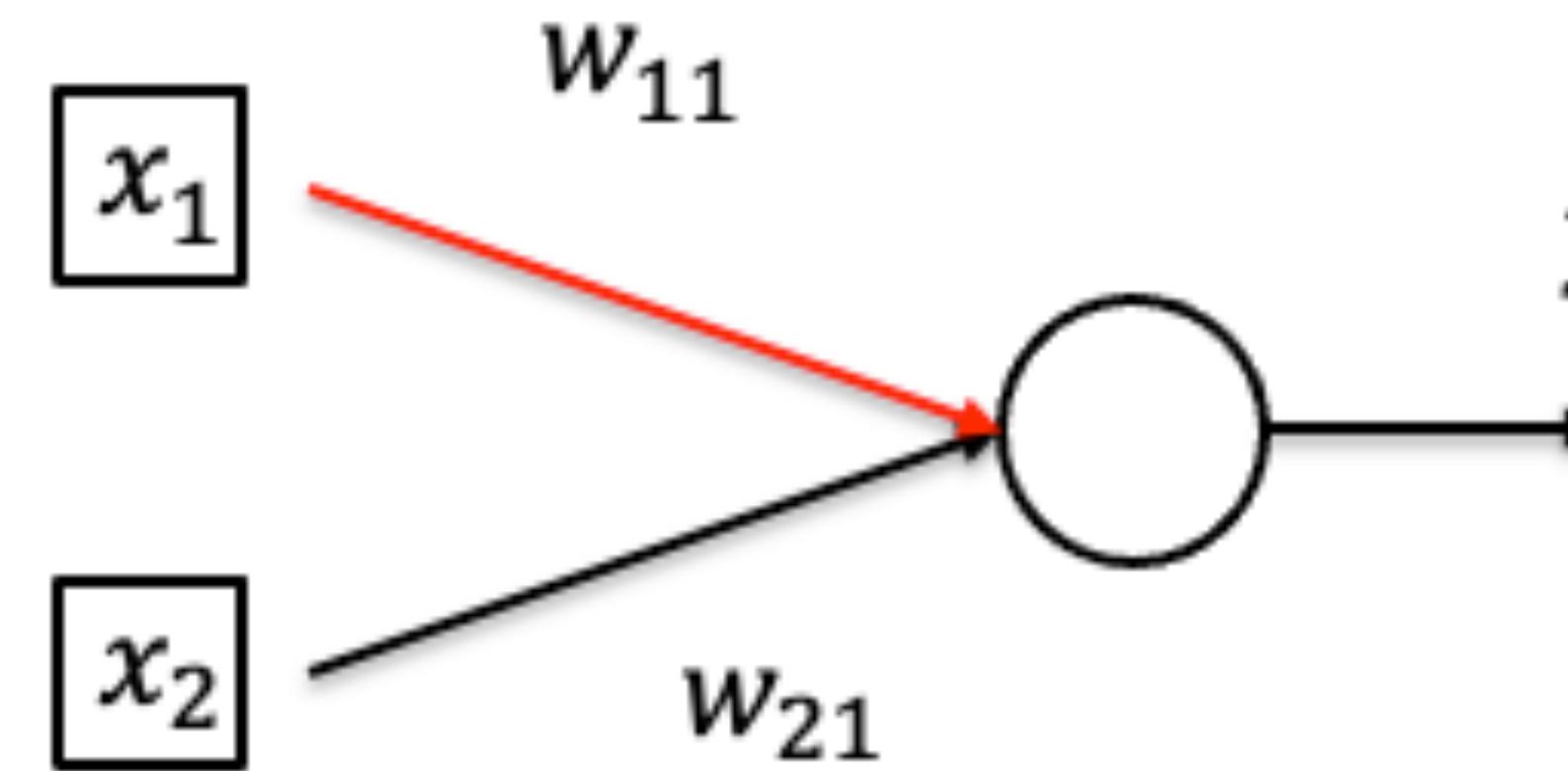


- Want to compute  $\frac{\partial \ell(\mathbf{x}, y)}{\partial w_{11}}$

# Calculate Gradient (on one data point)



# Calculate Gradient (on one data point)



A computational graph illustrating the forward pass and backpropagation for a logistic regression model. The forward pass consists of three main steps:

- Inputs  $w_{11}x_1$  and  $w_{21}x_2$  are summed to produce  $z$ .
- The sigmoid function is applied to  $z$  to produce the prediction  $\hat{y}$ .
- The loss function  $\ell(\mathbf{x}, y)$  is calculated based on  $\hat{y}$  and the target  $y$ .

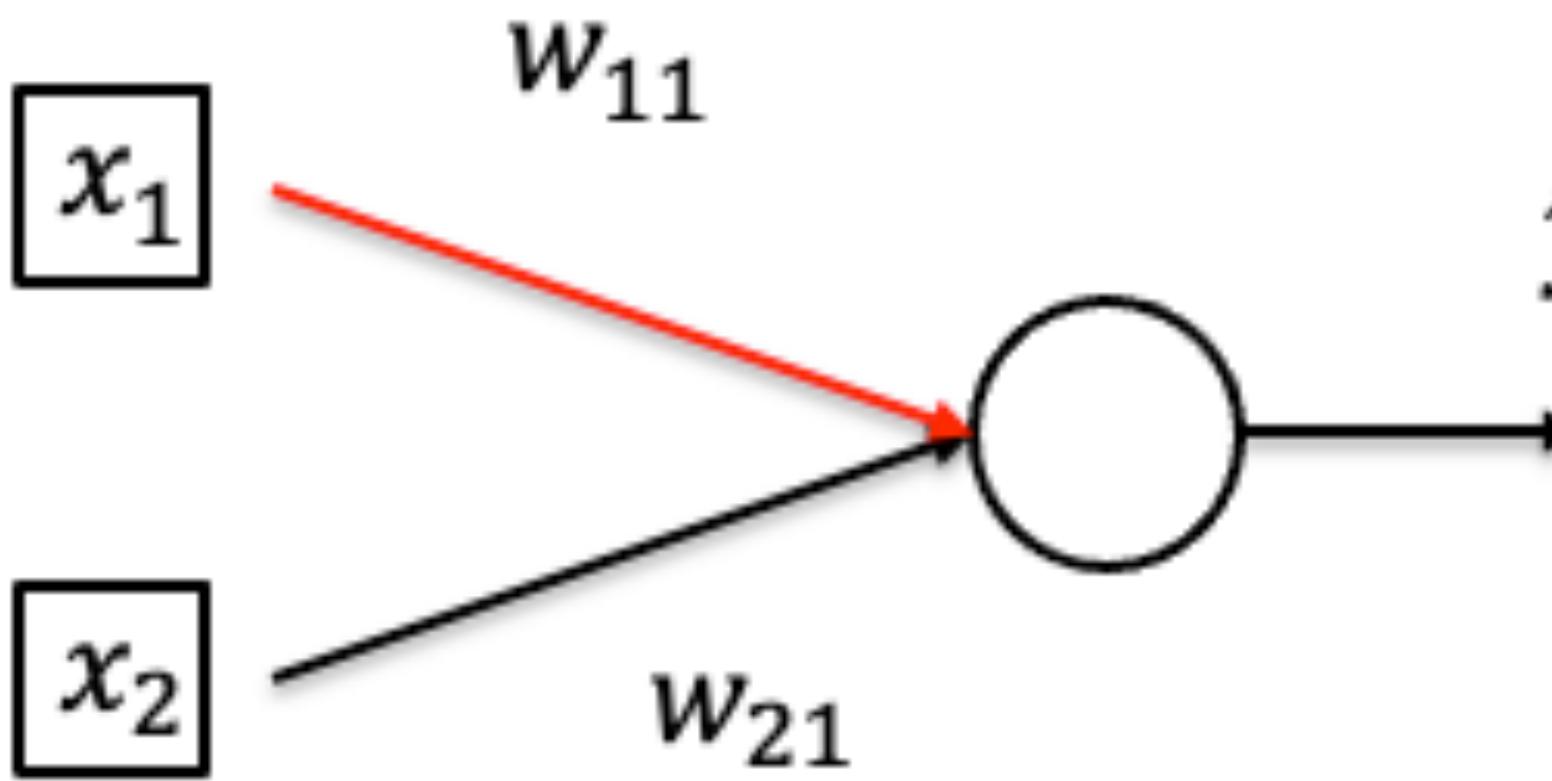
The backpropagation step shows the derivative of the loss with respect to the prediction  $\hat{y}$ :

$$\frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{-y \log(\hat{y})}{-(1 - y) \log(1 - \hat{y})} = \frac{1 - y}{1 - \hat{y}} - \frac{y}{\hat{y}}$$

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_{11}}$$

# Calculate Gradient (on one data point)



A computational graph illustrating the forward pass and backpropagation for a logistic regression model. The forward pass consists of three main steps:

- Inputs  $w_{11}x_1$  and  $w_{21}x_2$  are summed to produce  $z$ .
- The sigmoid function is applied to  $z$  to produce the predicted value  $\hat{y}$ .
- The loss function  $\ell(\mathbf{x}, y)$  is calculated based on  $\hat{y}$  and the target value  $y$ .

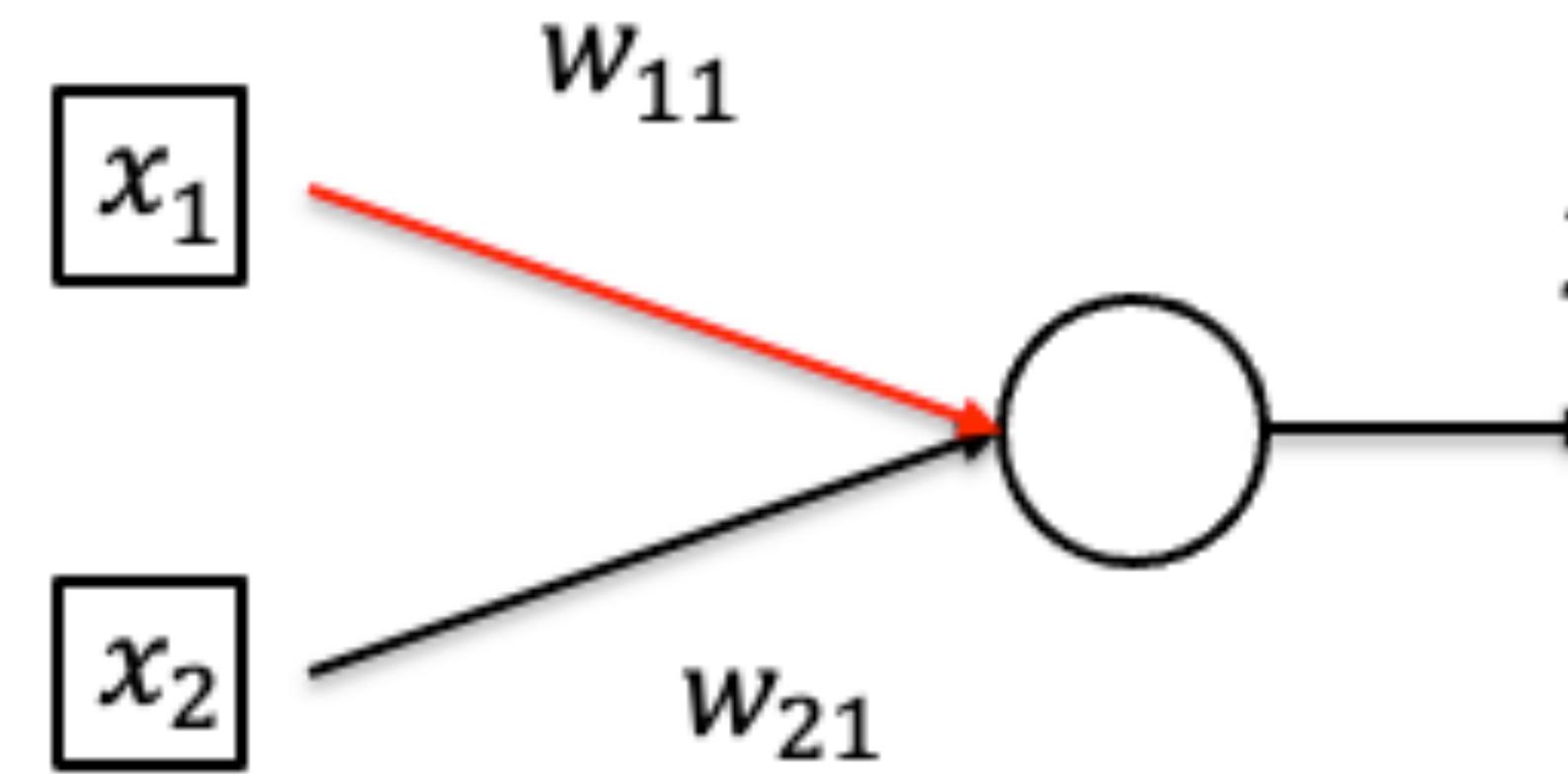
The backpropagation step shows the derivative of the loss function with respect to the predicted value  $\hat{y}$ :

$$\frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{-y \log(\hat{y})}{-(1 - y) \log(1 - \hat{y})} = \frac{1 - y}{1 - \hat{y}} - \frac{y}{\hat{y}}$$

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} x_1$$

# Calculate Gradient (on one data point)



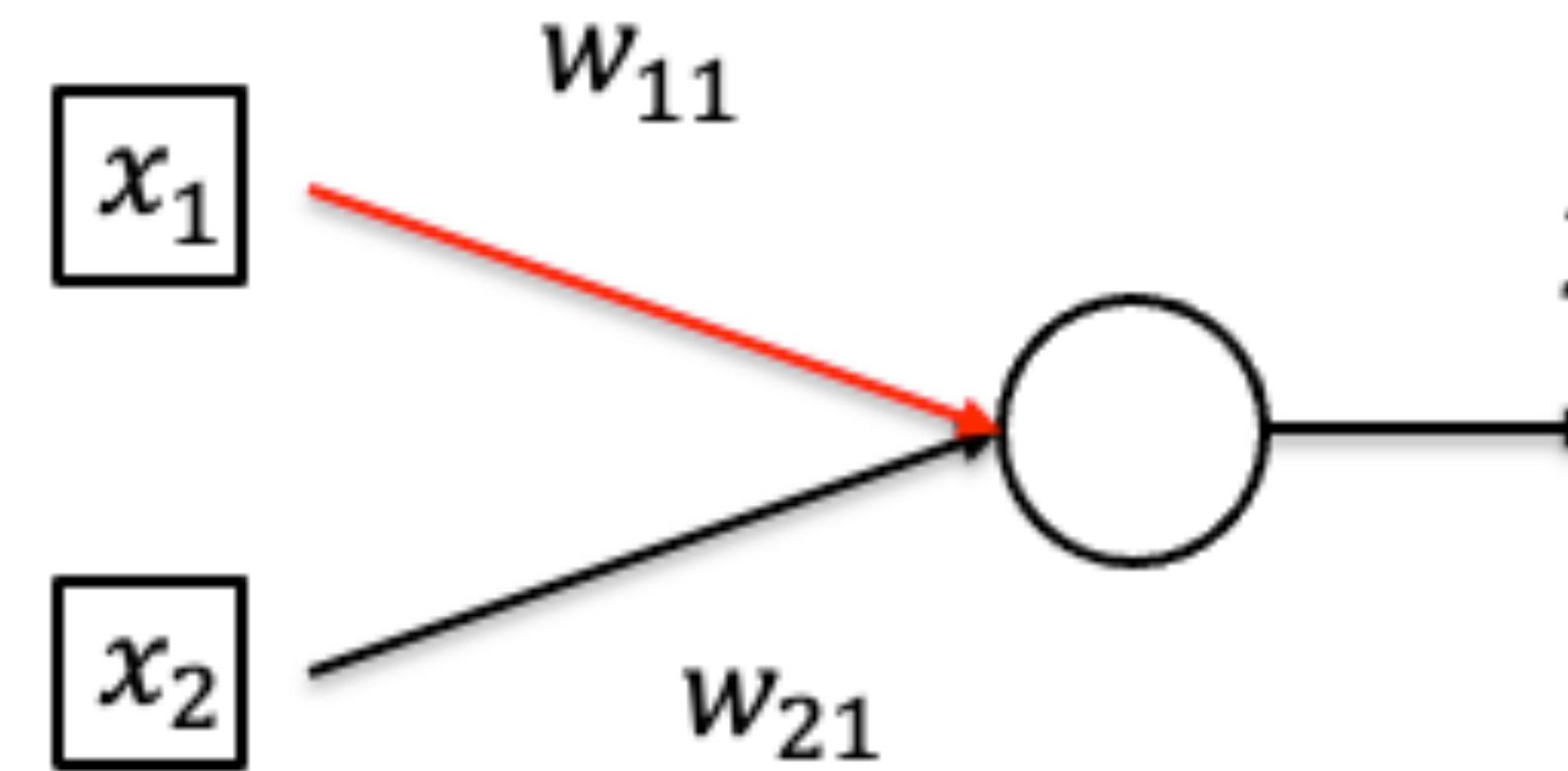
A computational graph illustrating the forward pass and the backward pass for calculating the gradient. The forward pass consists of three main steps: 1) Inputs  $w_{11}x_1$  and  $w_{21}x_2$  are summed at a plus node to produce  $z$ . 2) The sigmoid function maps  $z$  to  $\hat{y}$ . 3) The loss function maps  $\hat{y}$  to the loss value  $\ell(\mathbf{x}, y)$ . The backward pass involves calculating gradients:  $\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$ , and the final gradient term is  $\frac{\partial l}{\partial w_{21}} = \frac{\partial l}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot w_{21}x_2$ .

$$\begin{array}{ccccc} w_{11}x_1 & \xrightarrow{\quad + \quad} & \text{sigmoid function} & \xrightarrow{\quad \frac{-y \log(\hat{y})}{-(1 - \hat{y}) \log(1 - \hat{y})} \quad} & \ell(\mathbf{x}, y) \\ w_{21}x_2 & & z & \longrightarrow & \hat{y} \\ & & & & \frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z)) \end{array}$$

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \cdot \hat{y}(1 - \hat{y})x_1$$

# Calculate Gradient (on one data point)



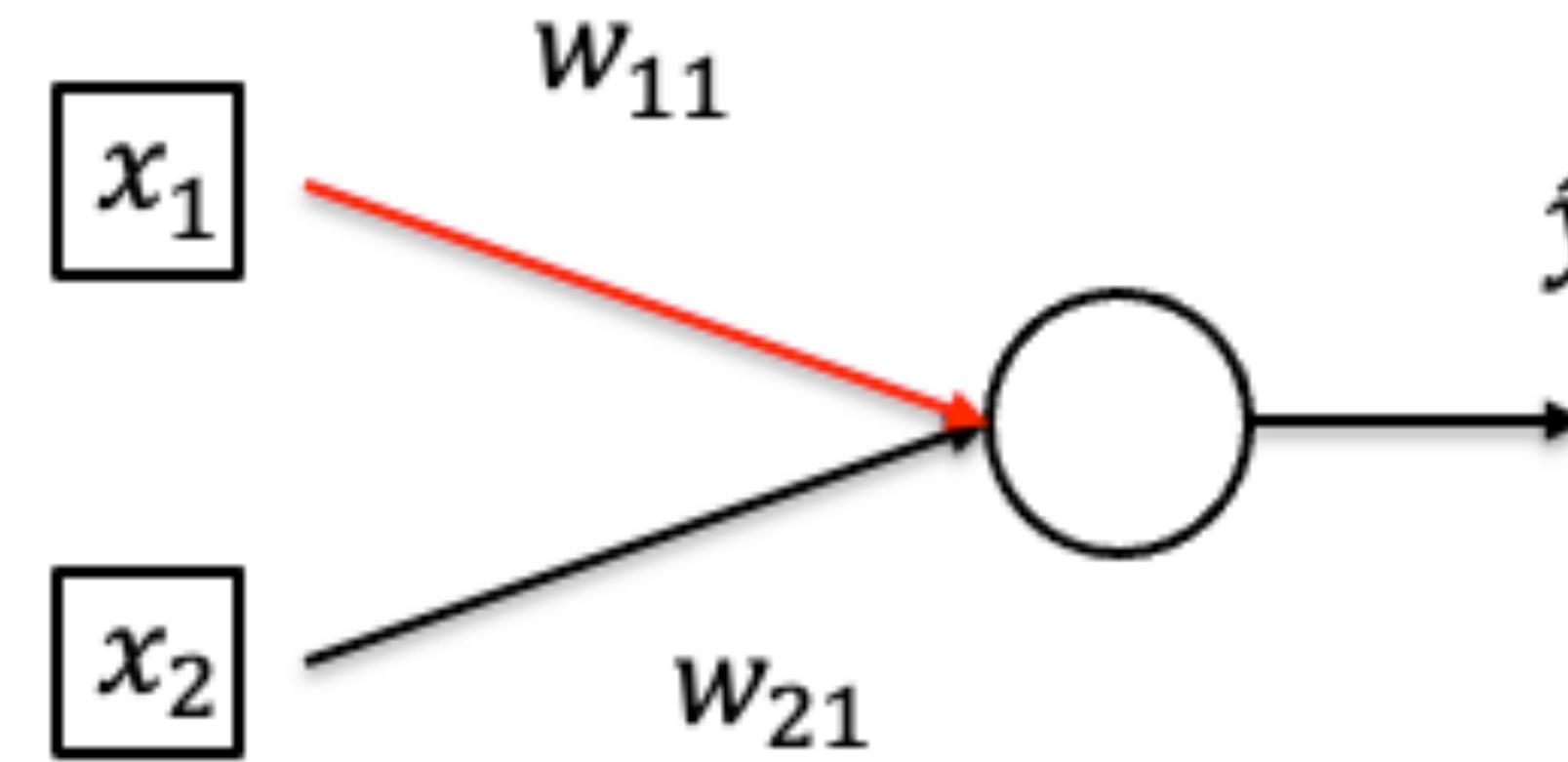
The diagram illustrates the calculation of the loss function  $\ell(\mathbf{x}, y)$  from the inputs  $x_1$  and  $x_2$ . The inputs are multiplied by their respective weights  $w_{11}$  and  $w_{21}$ , then summed. This sum is passed through a sigmoid function to produce the predicted output  $\hat{y}$ . The loss function is then calculated as  $-(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$ .

$$\begin{array}{c} w_{11}x_1 \\ w_{21}x_2 \\ \hline + \end{array} \quad \text{sigmoid function} \quad z \longrightarrow \hat{y} \quad \frac{-y \log(\hat{y})}{-(1 - y) \log(1 - \hat{y})} \longrightarrow \ell(\mathbf{x}, y)$$
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \left( \frac{1 - y}{1 - \hat{y}} - \frac{y}{\hat{y}} \right) \hat{y}(1 - \hat{y})x_1$$

# Calculate Gradient (on one data point)



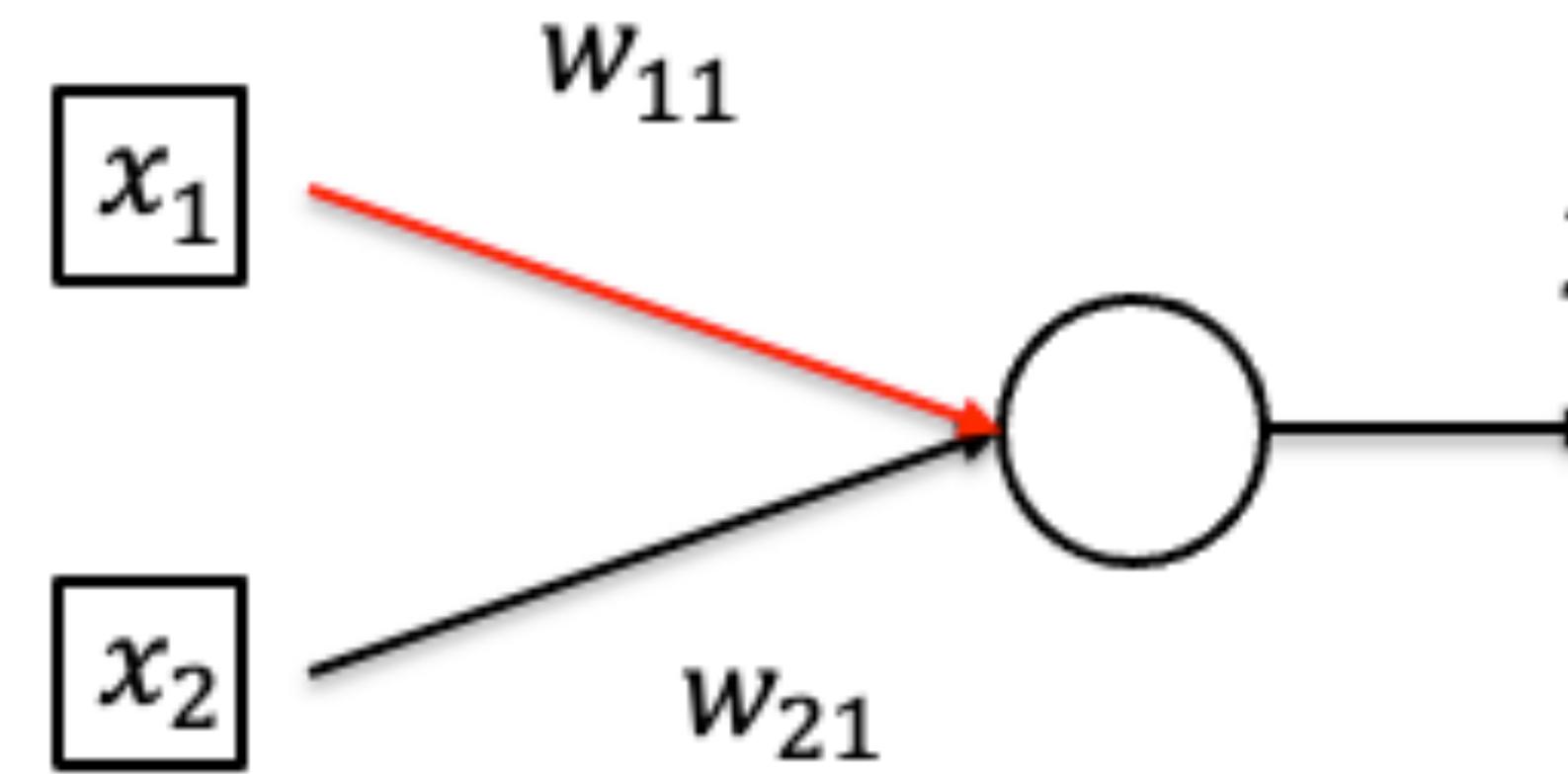
A computational graph illustrating the forward pass and loss calculation:

- Inputs  $w_{11}x_1$  and  $w_{21}x_2$  are summed at a plus node.
- The result of the summation is passed through a "sigmoid function".
- The output of the sigmoid function is  $\hat{y}$ .
- The loss function is calculated as  $-(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$ .
- The derivative of the loss with respect to  $\hat{y}$  is shown as  $\frac{\partial l}{\partial \hat{y}} = -(y/\hat{y}) + ((1 - y)/(1 - \hat{y}))$ .
- The derivative of the sigmoid function with respect to  $z$  is given as  $\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$ .

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = (\hat{y} - y)x_1$$

# Calculate Gradient (on one data point)



A computational graph illustrating the forward pass and gradients for a single data point. The forward pass consists of three main steps:

- Input features  $x_1$  and  $x_2$  are multiplied by weights  $w_{11}$  and  $w_{21}$  respectively, and then summed.
- The result of the summation is passed through a sigmoid function to produce the predicted output  $\hat{y}$ .
- The loss function  $\ell(\mathbf{x}, y)$  is calculated based on the predicted output  $\hat{y}$  and the true target  $y$ .

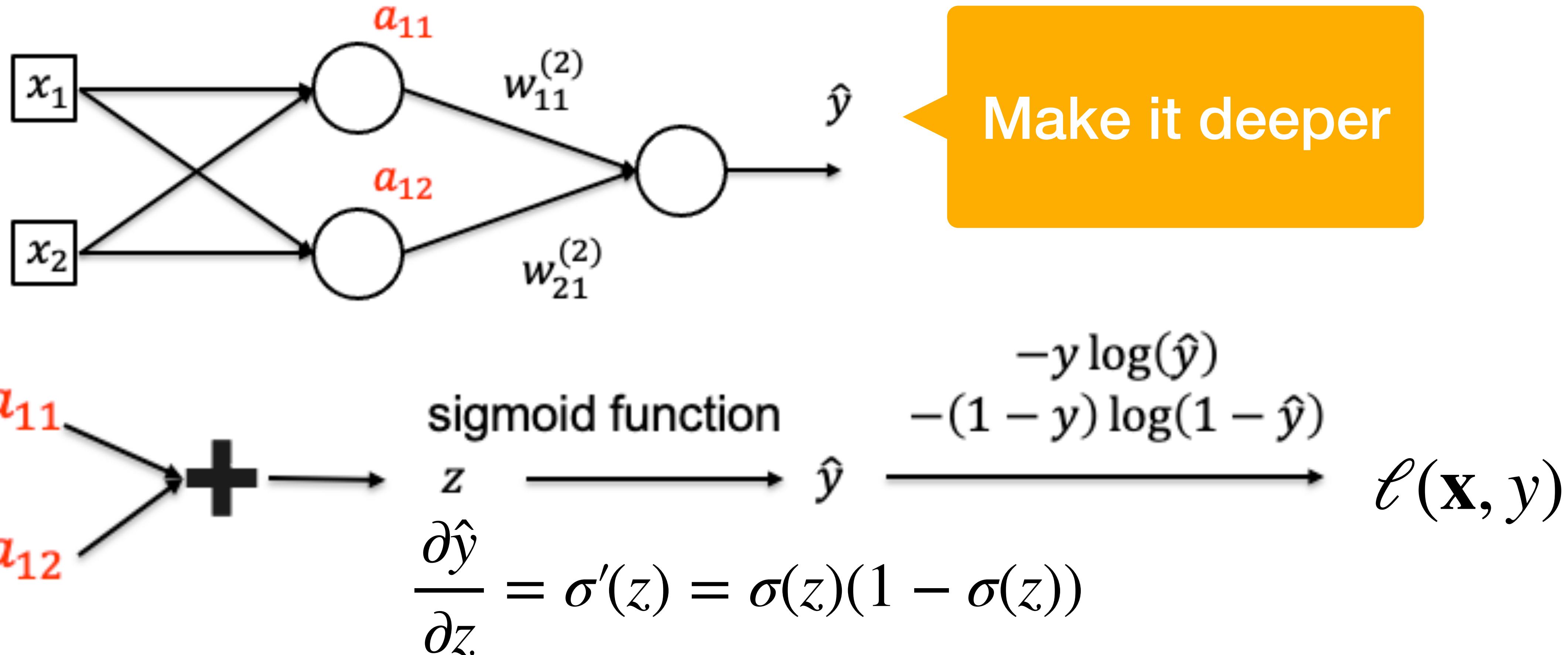
Below the graph, the gradient of the loss function with respect to the input  $x_1$  is derived using the chain rule:

$$\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} w_{11} = (\hat{y} - y)w_{11}$$

- By chain rule:

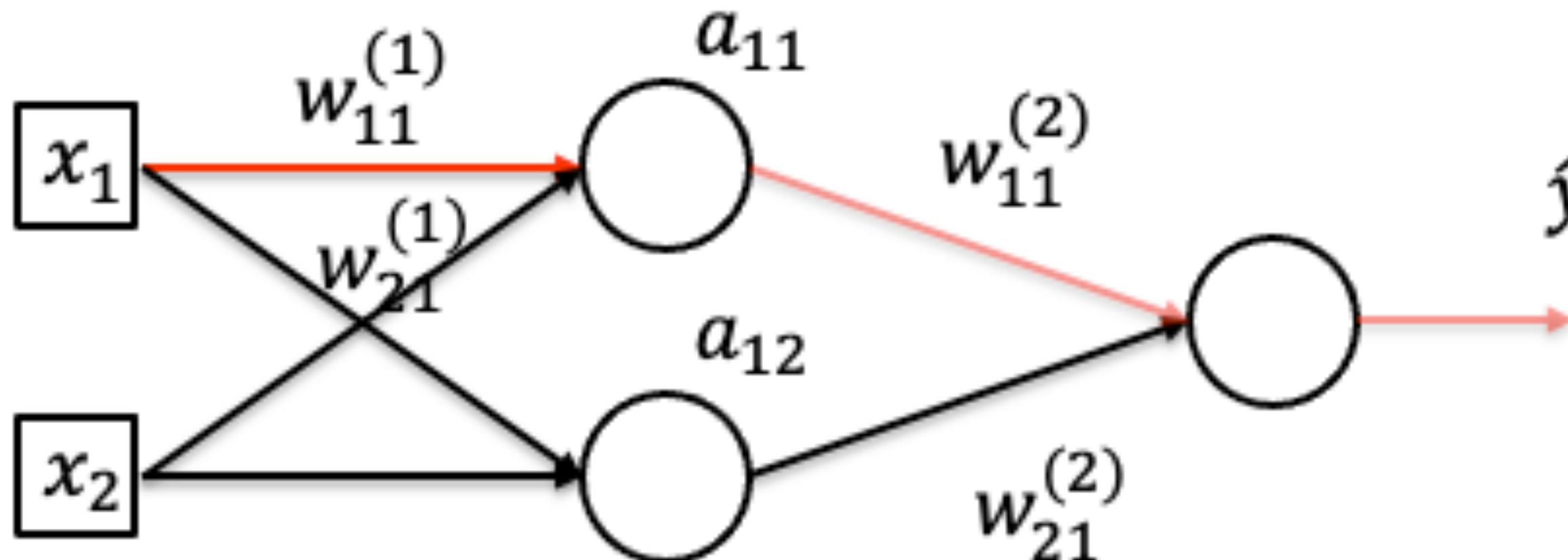
$$\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} w_{11} = (\hat{y} - y)w_{11}$$

# Calculate Gradient (on one data point)



- By chain rule:  $\frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}$ ,  $\frac{\partial l}{\partial a_{12}} = (\hat{y} - y)w_{21}^{(2)}$

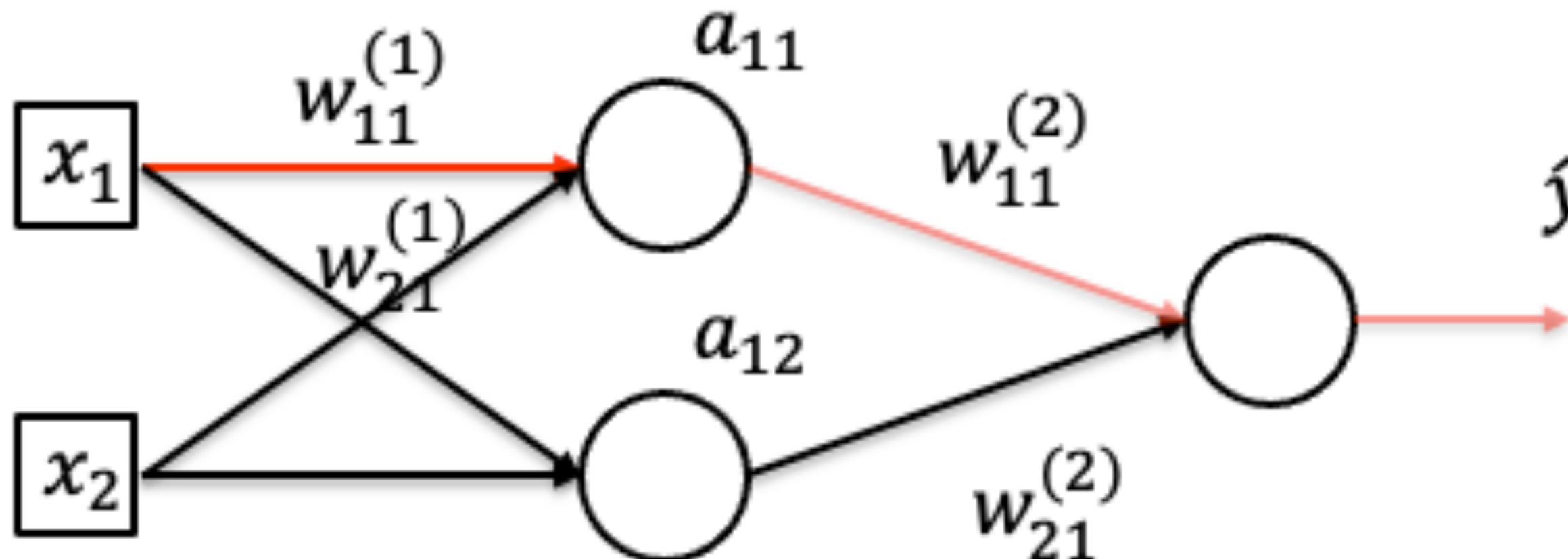
# Calculate Gradient (on one data point)



The diagram illustrates a single layer of a neural network. It starts with two input nodes,  $w_{11}^{(1)}x_1$  and  $w_{21}^{(1)}x_2$ , which converge to a central summation node (a large black plus sign). This results in the weighted sum  $z_{11}$ . The next step is the application of an activation function  $\sigma(z_{11})$ , producing the output  $a_{11}$ . Finally, this output  $a_{11}$  is passed through another function  $l(x, y)$ .

- By chain rule:  $\frac{\partial l}{\partial w_{11}^{(1)}} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial w_{11}^{(1)}} = (\hat{y} - y)w_{11}^{(2)} \frac{\partial a_{11}}{\partial w_{11}^{(1)}}$

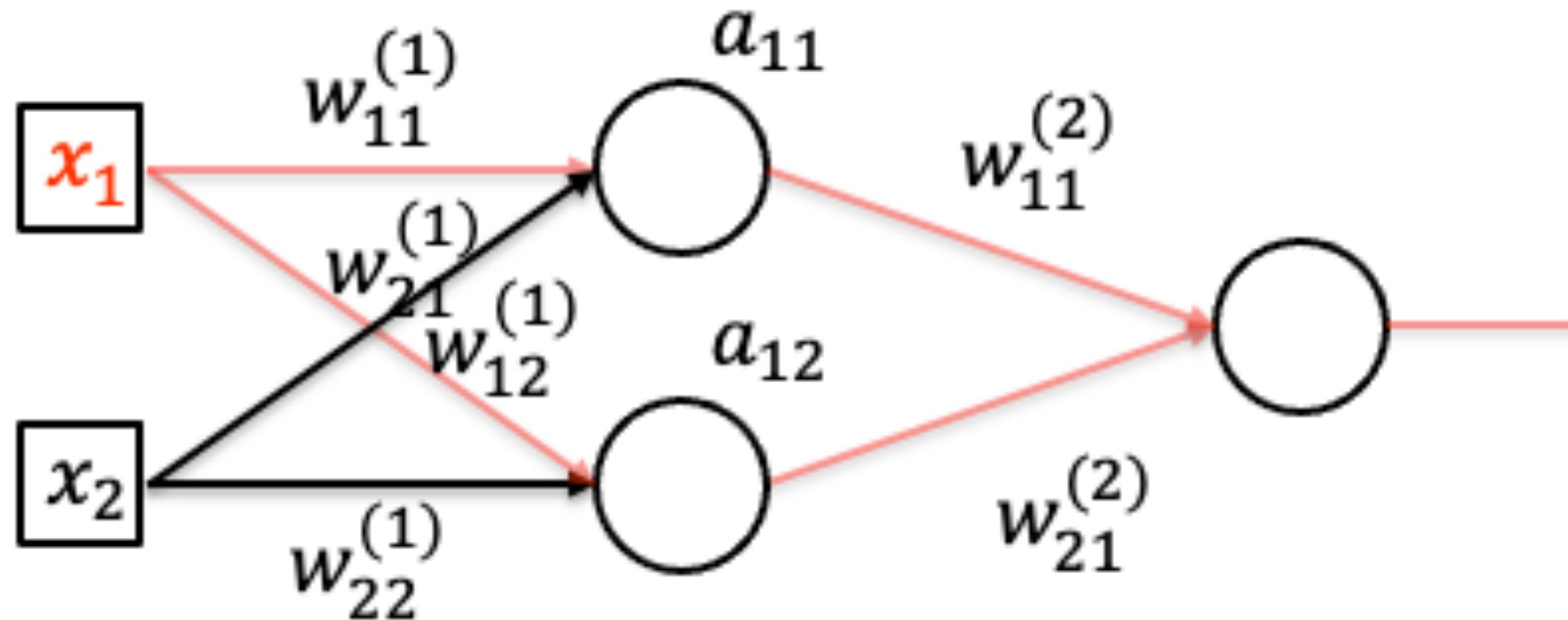
# Calculate Gradient (on one data point)



$$\begin{aligned} & w_{11}^{(1)} x_1 \quad \quad \quad z_{11} \quad \quad \quad a_{11} \quad \quad \quad l(x, y) \\ & w_{21}^{(1)} x_2 \quad \quad \quad \sigma(z_{11}) \quad \quad \quad \frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \quad \quad \quad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y) w_{11}^{(2)} \end{aligned}$$

- By chain rule:  $\frac{\partial l}{\partial w_{11}^{(1)}} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial w_{11}^{(1)}} = (\hat{y} - y) w_{11}^{(2)} a_{11} (1 - a_{11}) x_1$

# Calculate Gradient (on one data point)



$$\begin{aligned} w_{11}^{(1)} x_1 & \quad \quad \quad + \quad \quad \quad \sigma(z_{11}) \quad \quad \quad l(\mathbf{x}, y) \\ w_{21}^{(1)} x_2 & \end{aligned}$$
$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \quad \quad \quad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y) w_{11}^{(2)}$$

- By chain rule:

$$\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial x_1} + \frac{\partial l}{\partial a_{12}} \frac{\partial a_{12}}{\partial x_1}$$

# Quiz Break

Gradient Descent in neural network training computes the \_\_\_\_\_ of a loss function with respect to the model \_\_\_\_\_ until convergence.

- A gradients, parameters
- B parameters, gradients
- C loss, parameters
- D parameters, loss

# Quiz Break

Gradient Descent in neural network training computes the \_\_\_\_\_ of a loss function with respect to the model \_\_\_\_\_ until convergence.

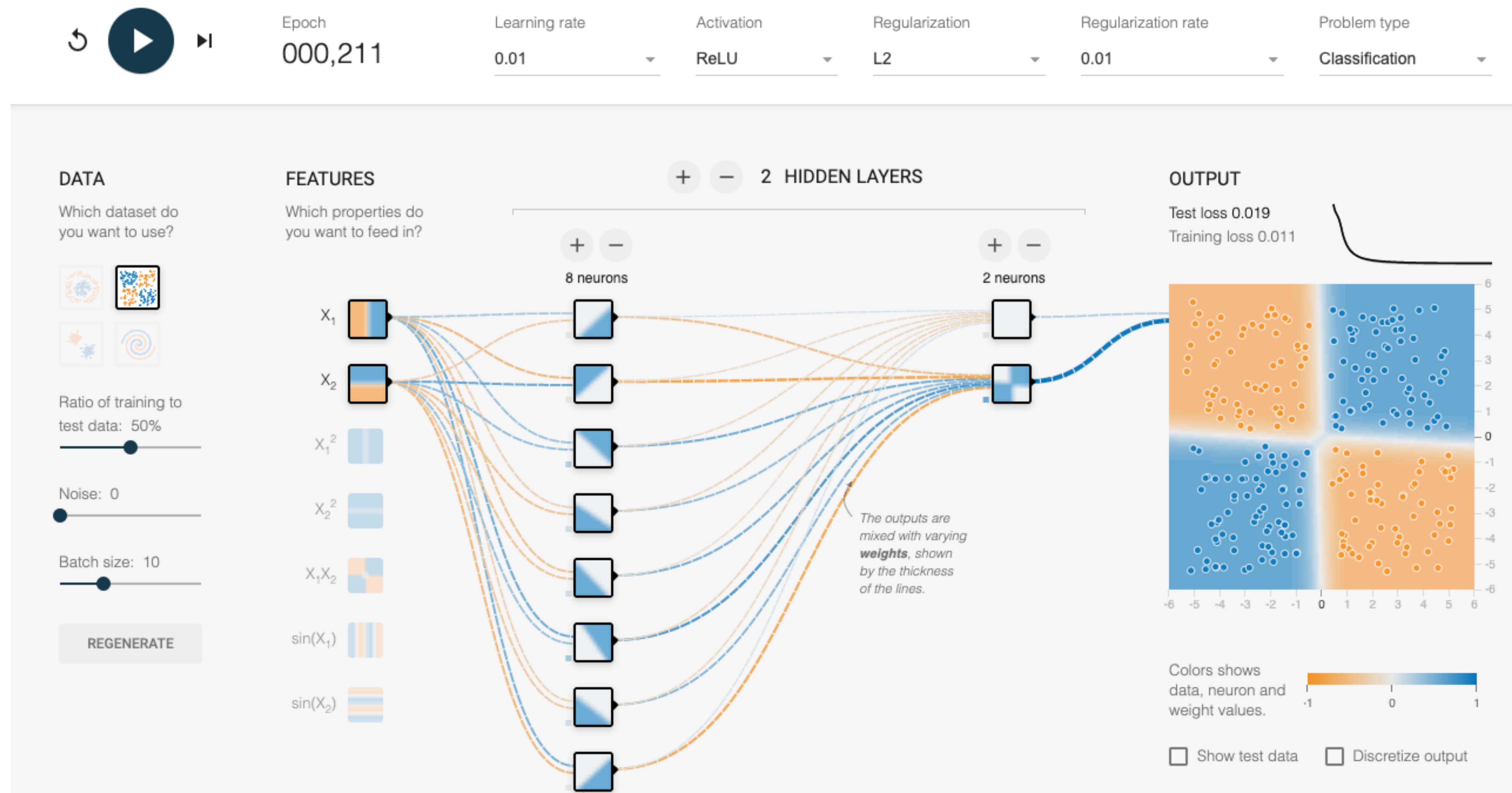
- A gradients, parameters
- B parameters, gradients
- C loss, parameters
- D parameters, loss

# Quiz Break

Suppose you are given a dataset with 1,000,000 images to train with. Which of the following methods is more desirable if training resources are limit but enough accuracy is needed?

- A Gradient Descent
- B Stochastic Gradient Descent
- C Minibatch Stochastic Gradient Descent
- D Computation Graph

# Demo: Learning XOR using neural net



• <https://playground.tensorflow.org/>

# What we've learned today...

- Single-layer Perceptron Review
- Multi-layer Perceptron
  - Single output
  - Multiple output
- How to train neural networks
  - Gradient descent