



CS 540 Introduction to Artificial Intelligence Neural Networks (II)

Yingyu Liang

University of Wisconsin-Madison

Oct 21, 2021

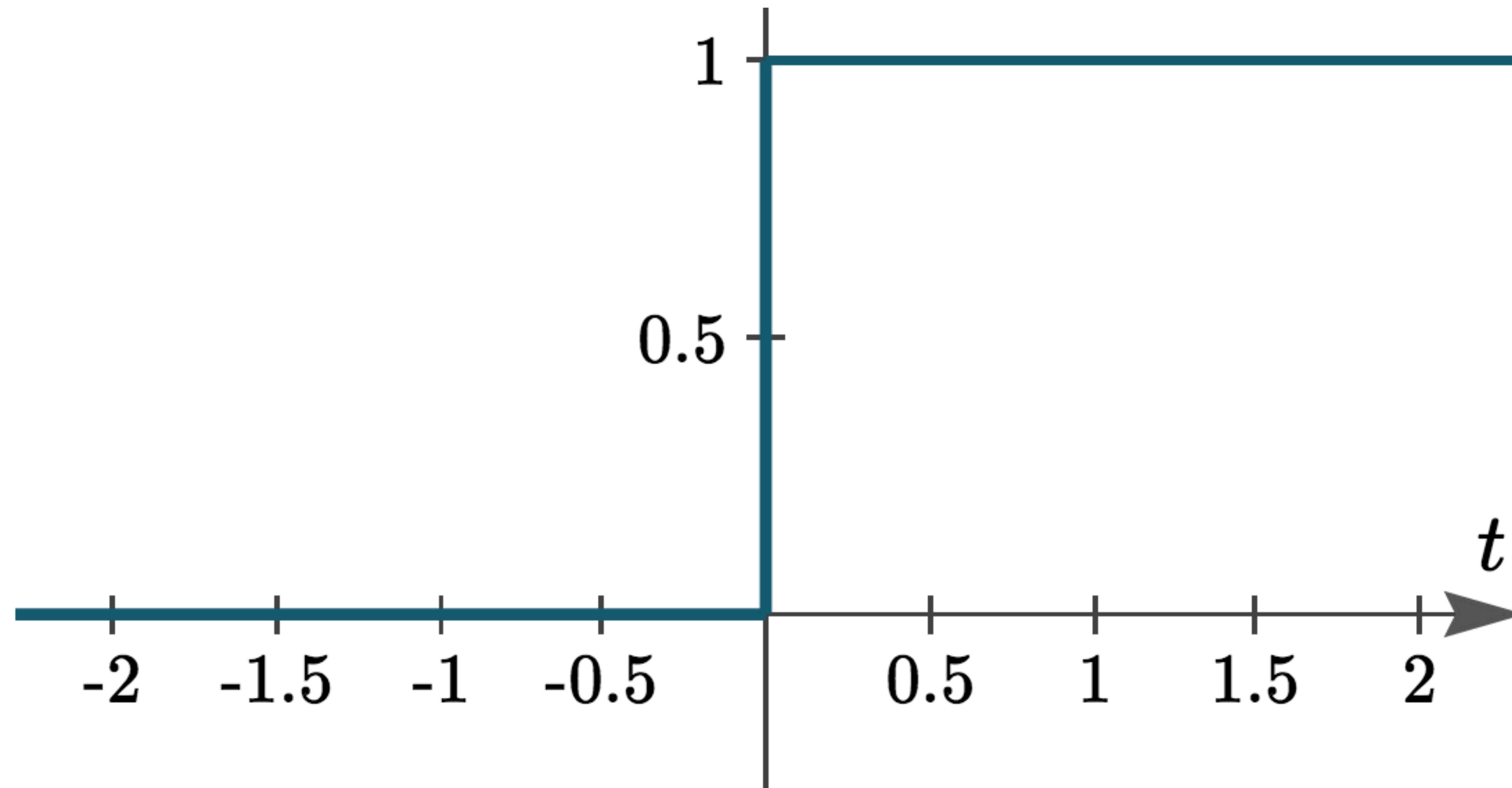
Today's outline

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

Step Function activation

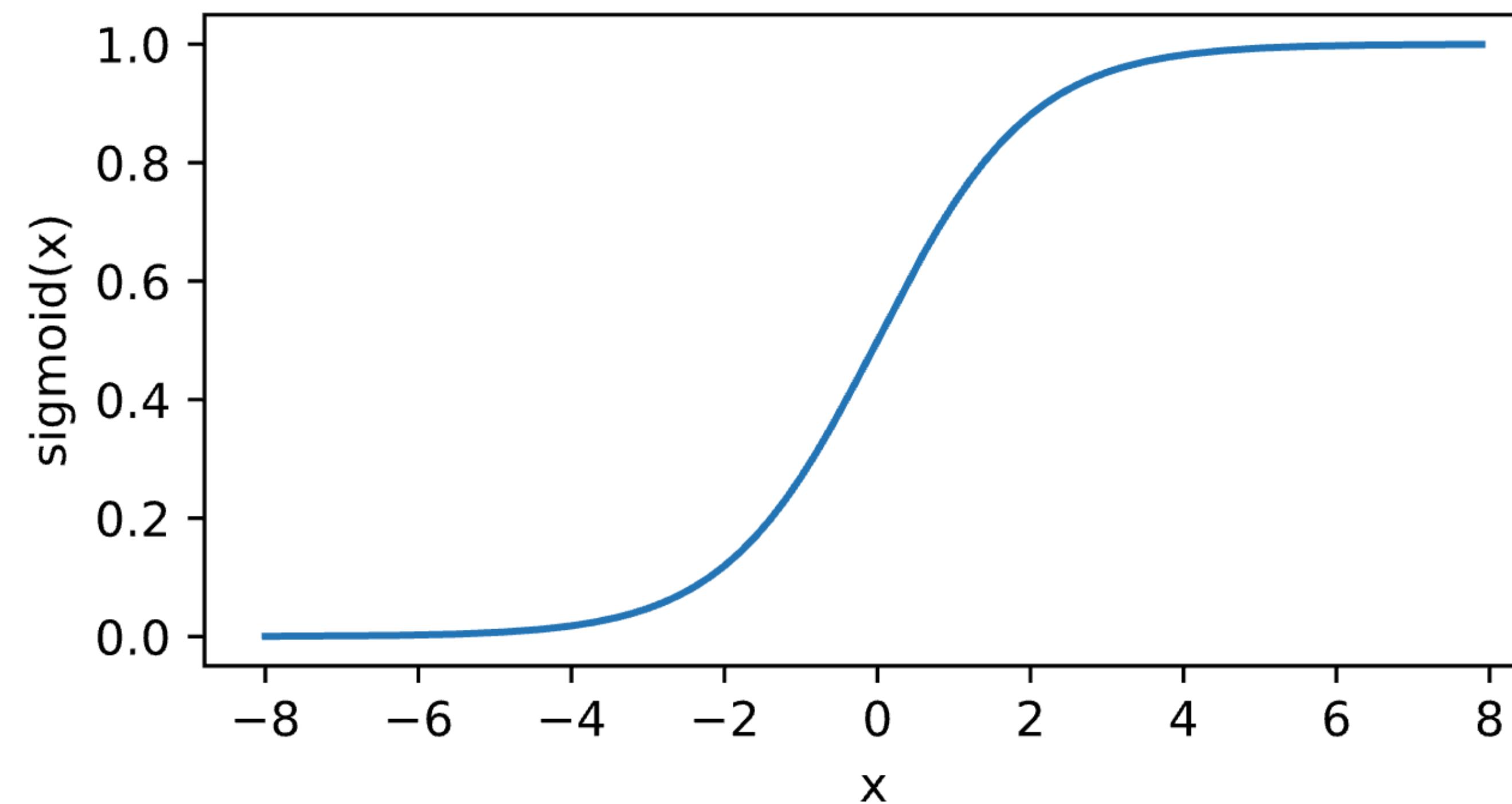
Step function is discontinuous, which cannot be used for gradient descent

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



Sigmoid/Logistic Activation

Map input into $[0, 1]$, a **soft** version of $\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$

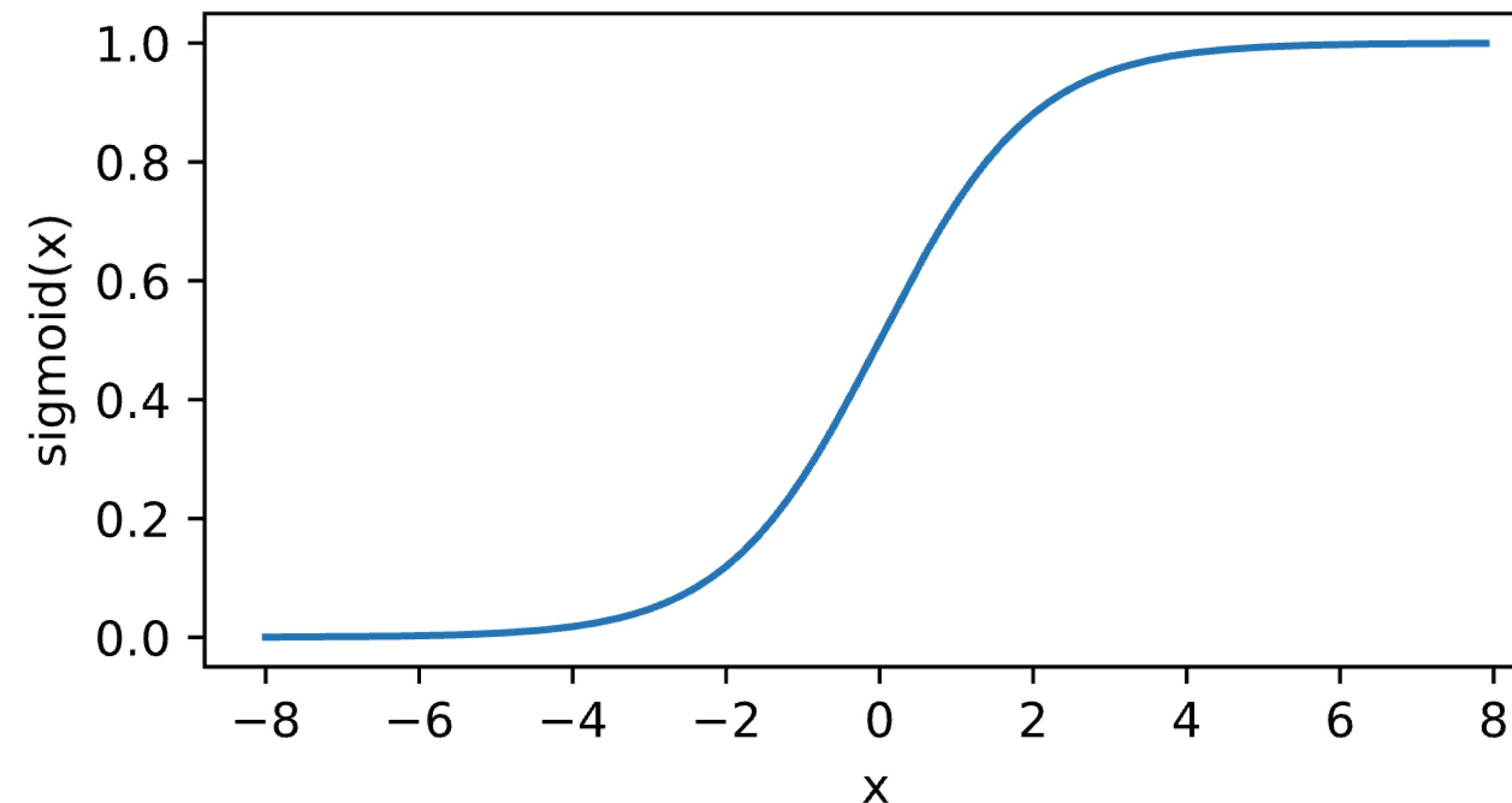
$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$


Logistic regression

$\mathbf{x} \in \mathbb{R}^d, y = \{-1, +1\}$

$$p(y = 1 | \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

$$p(y = -1 | \mathbf{x}) = 1 - \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(\mathbf{w}^T \mathbf{x})}$$



Logistic regression

Given: $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$

Training: maximize likelihood estimate (on the conditional probability)

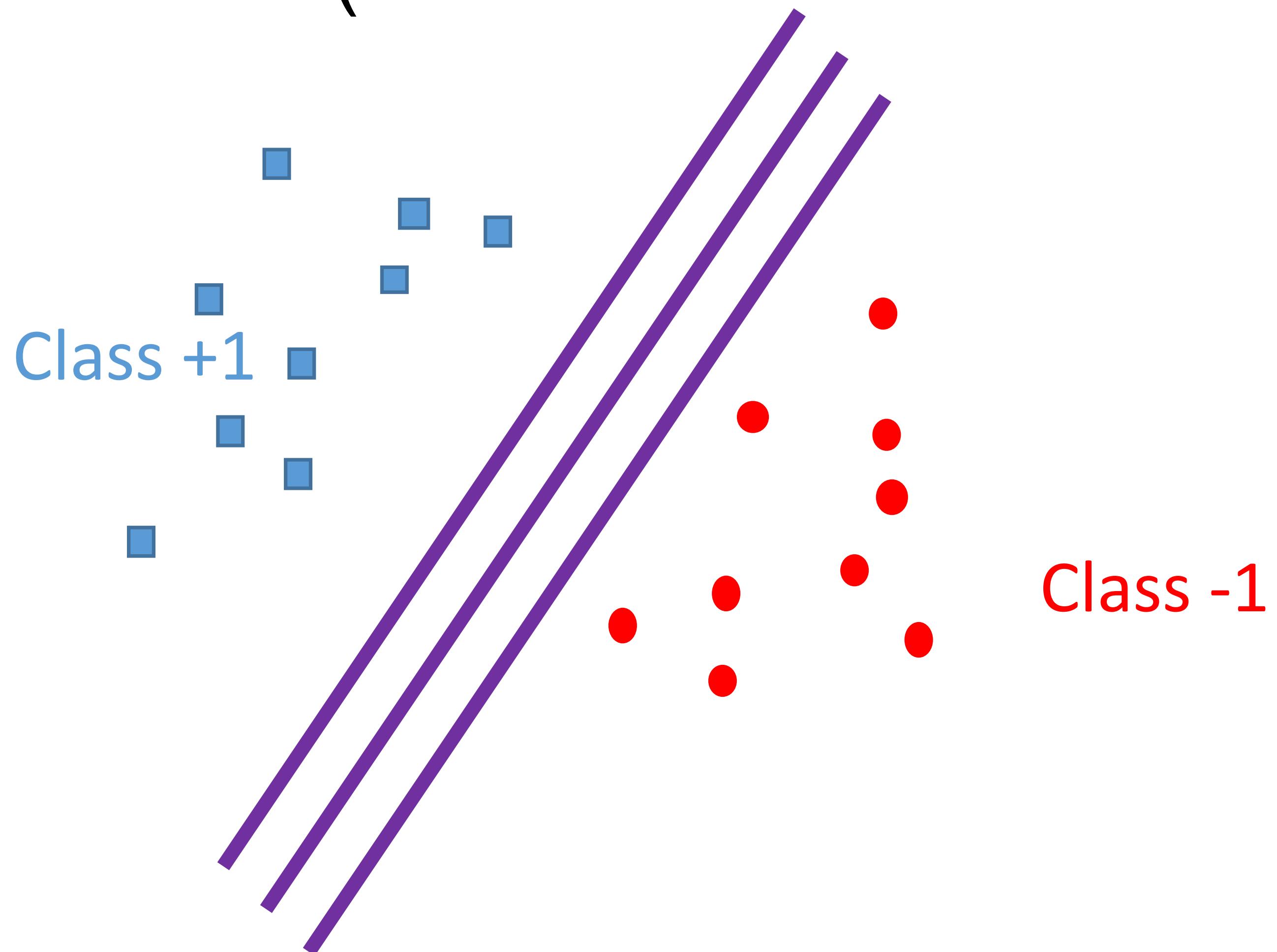
$$\max_{\mathbf{w}} \sum_i \log \frac{1}{1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i)}$$

Logistic regression

Given: $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$

Training: maximize likelihood estimate (on the conditional probability)

When training data is linearly separable, many solutions



Logistic regression

Given: $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$

Training: maximum A posteriori (MAP)

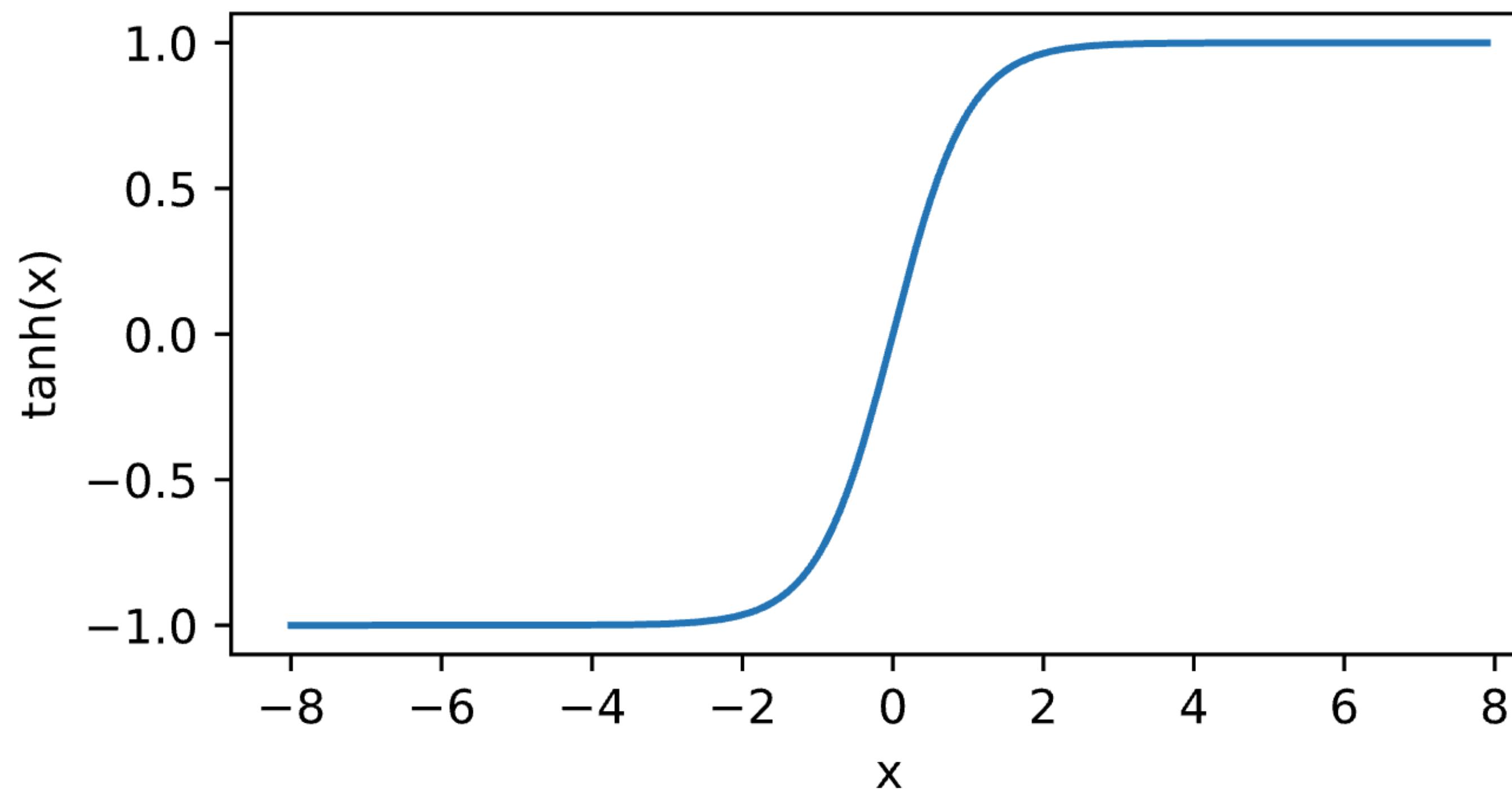
$$\min_{\mathbf{w}} \sum_i -\log \frac{1}{1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i)} + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

- Convex optimization
- Solve via (stochastic) gradient descent

Tanh Activation

Map inputs into (-1, 1)

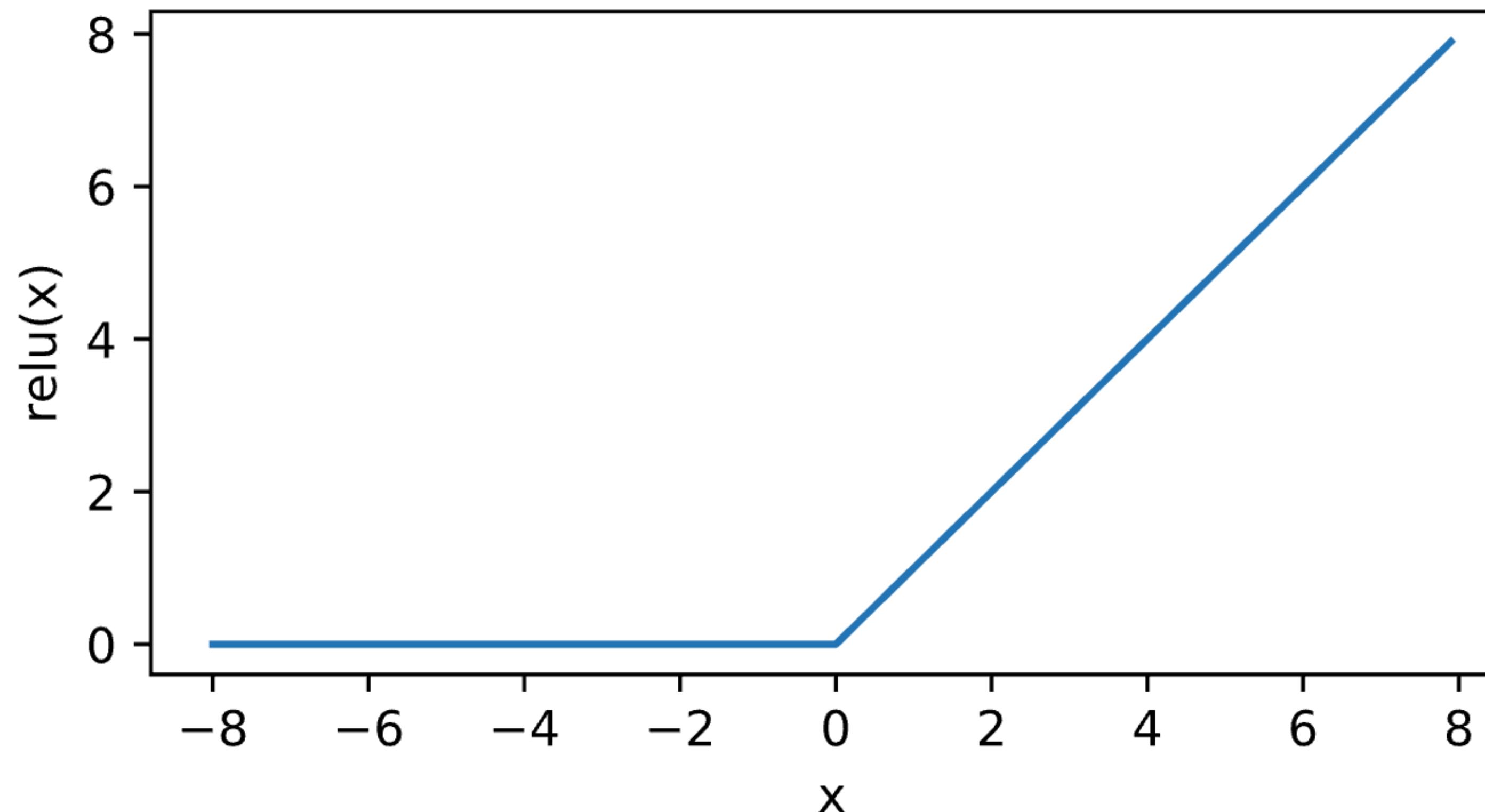
$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$



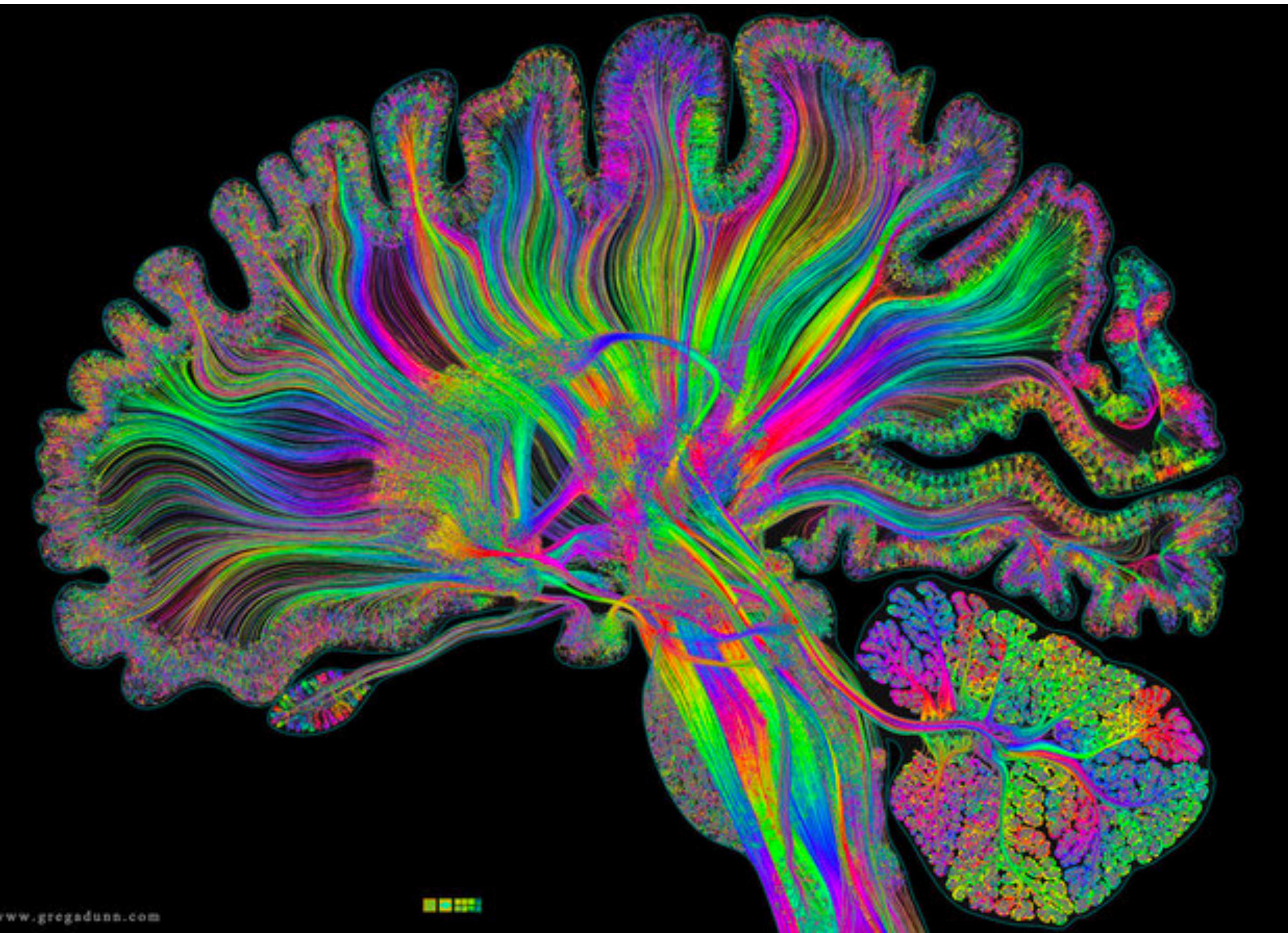
ReLU Activation

ReLU: rectified linear unit (commonly used in modern neural networks)

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

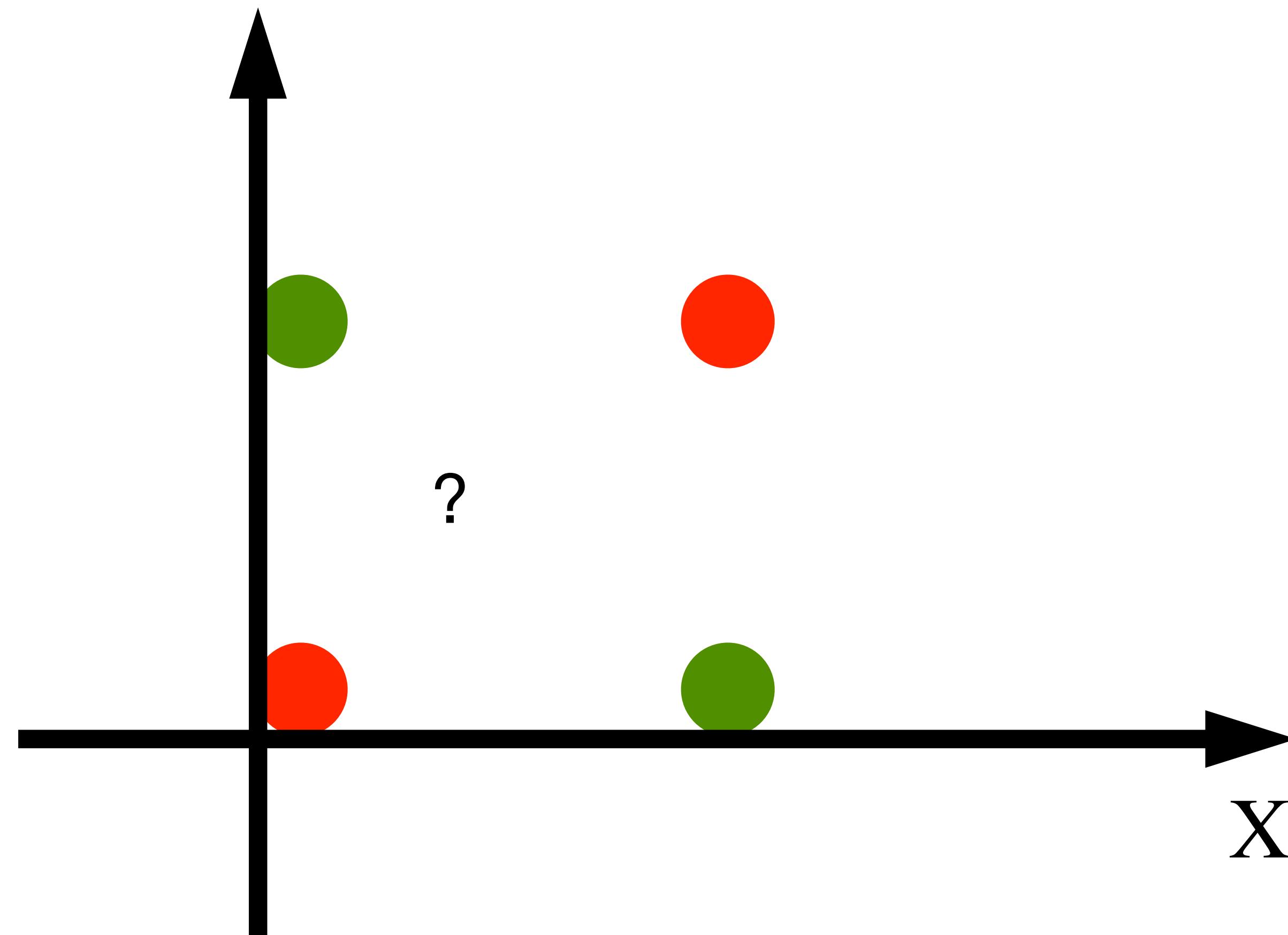


Multilayer Perceptron



The limited power of a single neuron

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



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

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

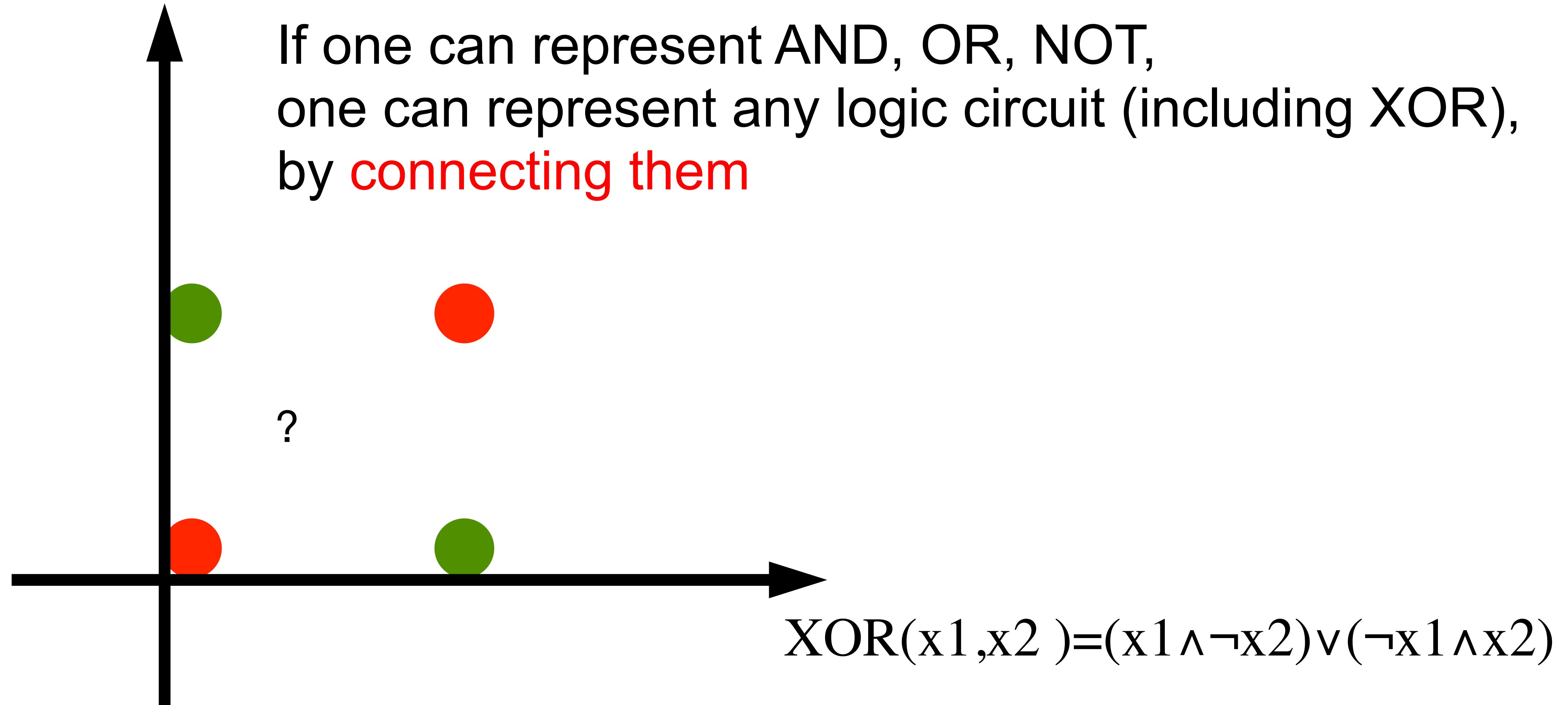
$$x_1 = 0, x_2 = 1, y = 1$$

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

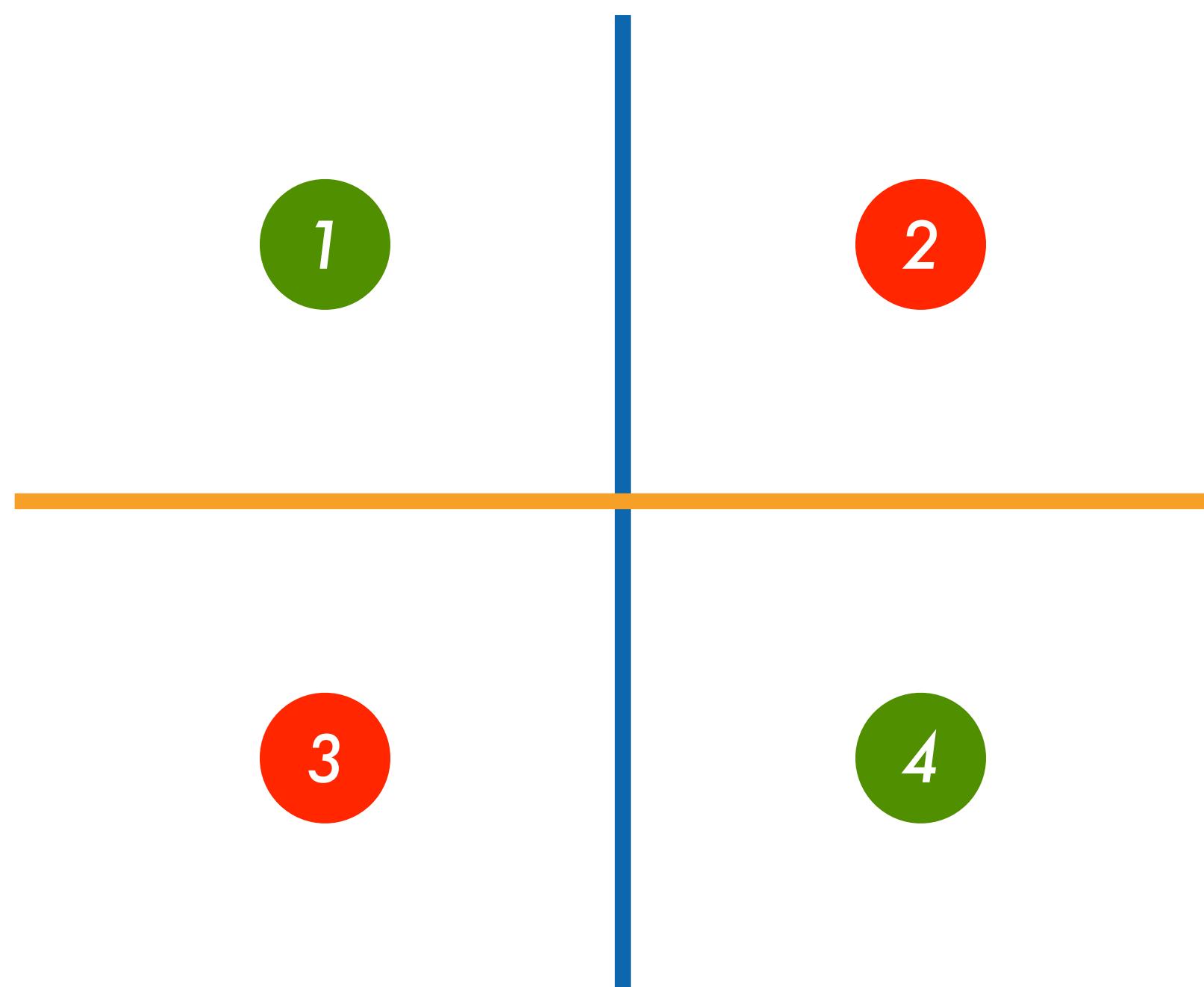
$$\text{XOR}(x_1, x_2) = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$

The limited power of a single neuron

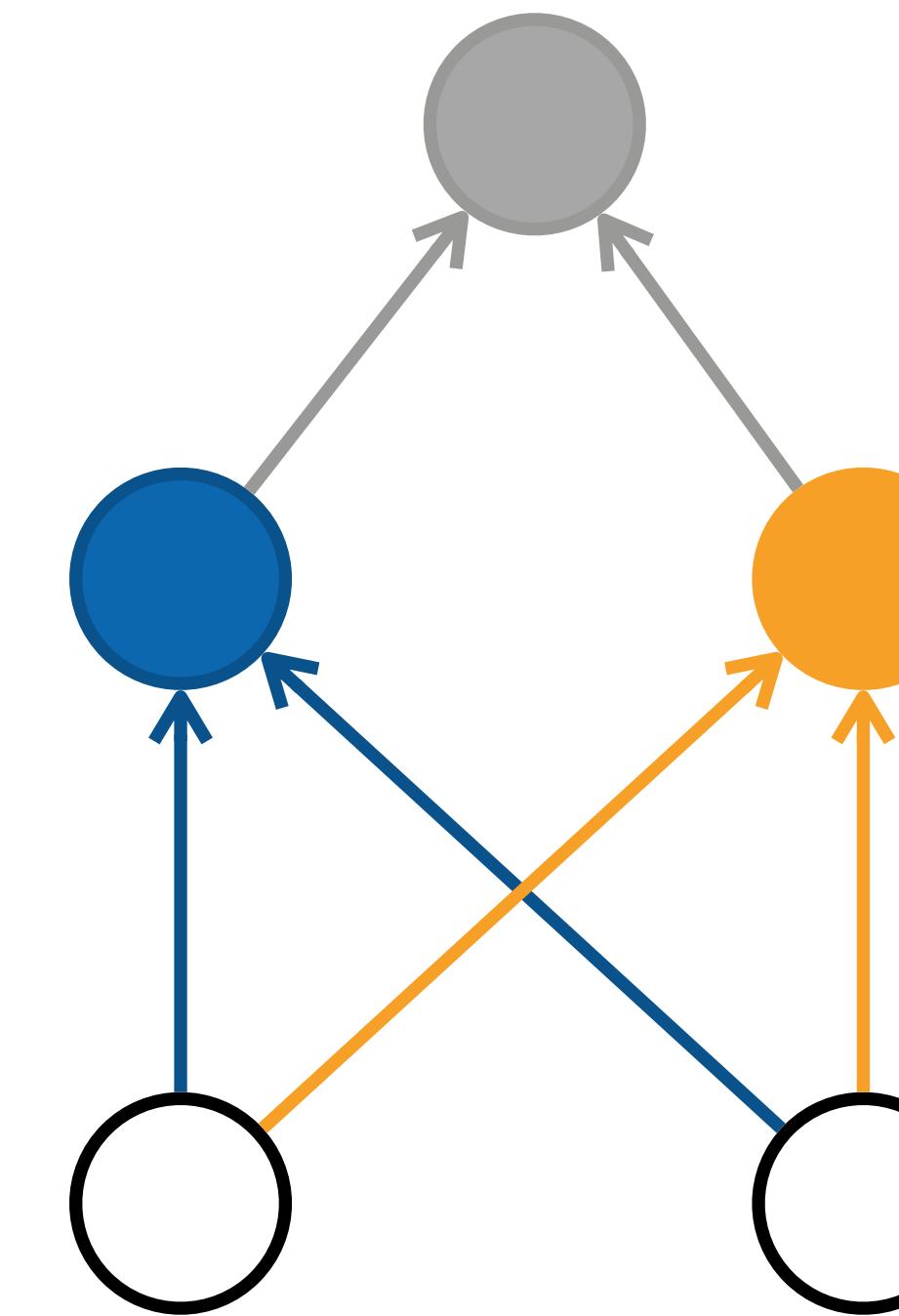
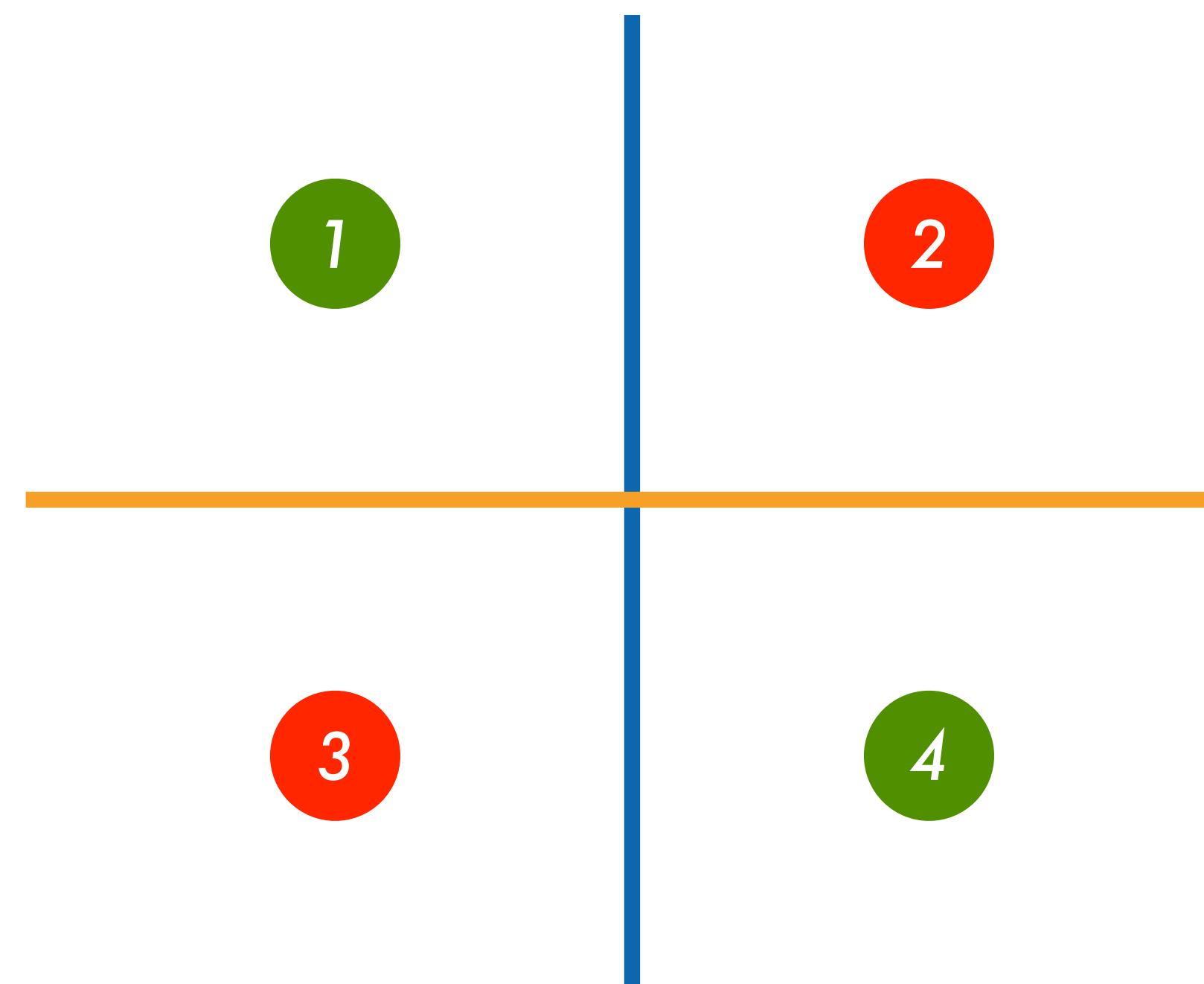
XOR problem



Learning XOR

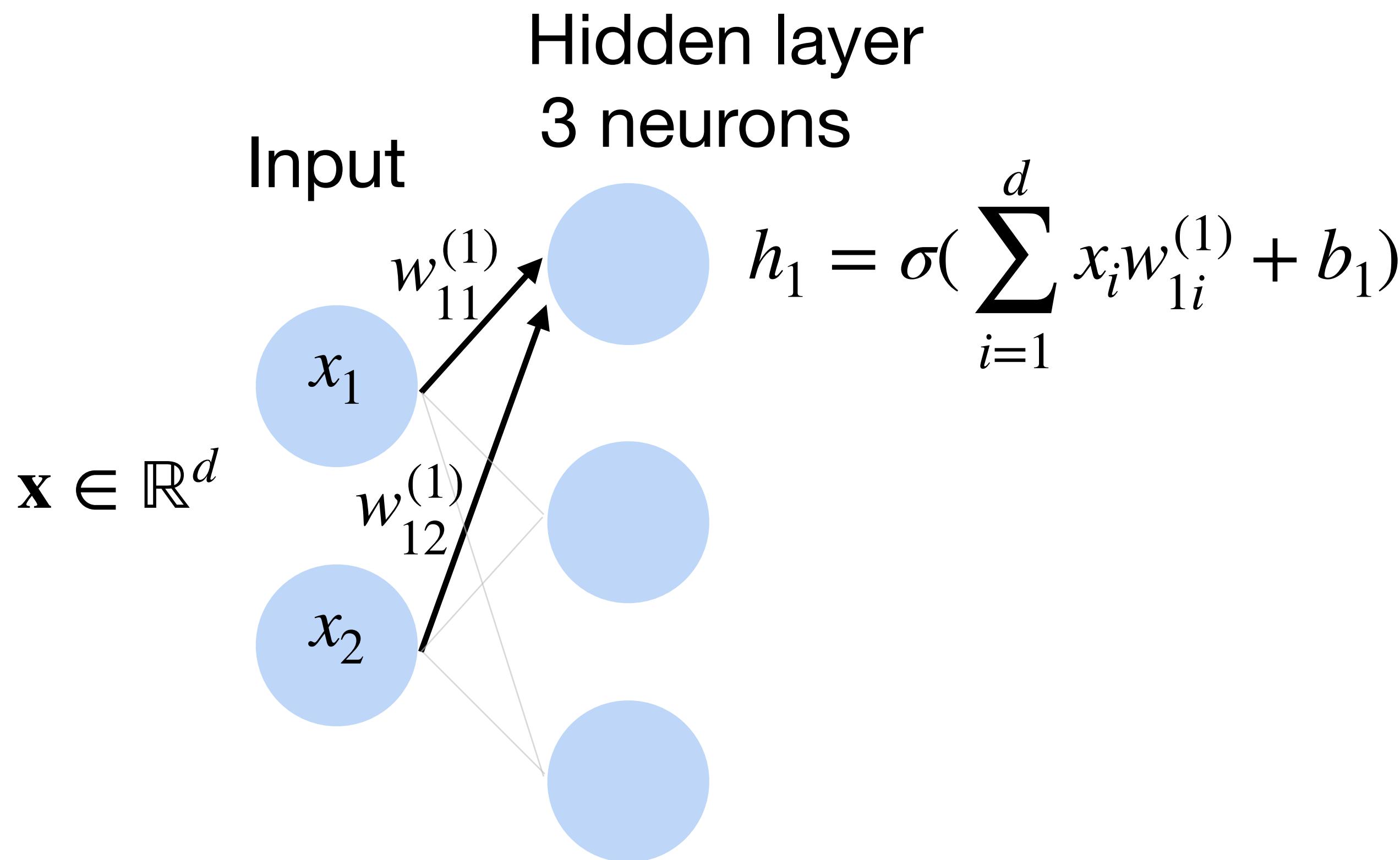


Learning XOR



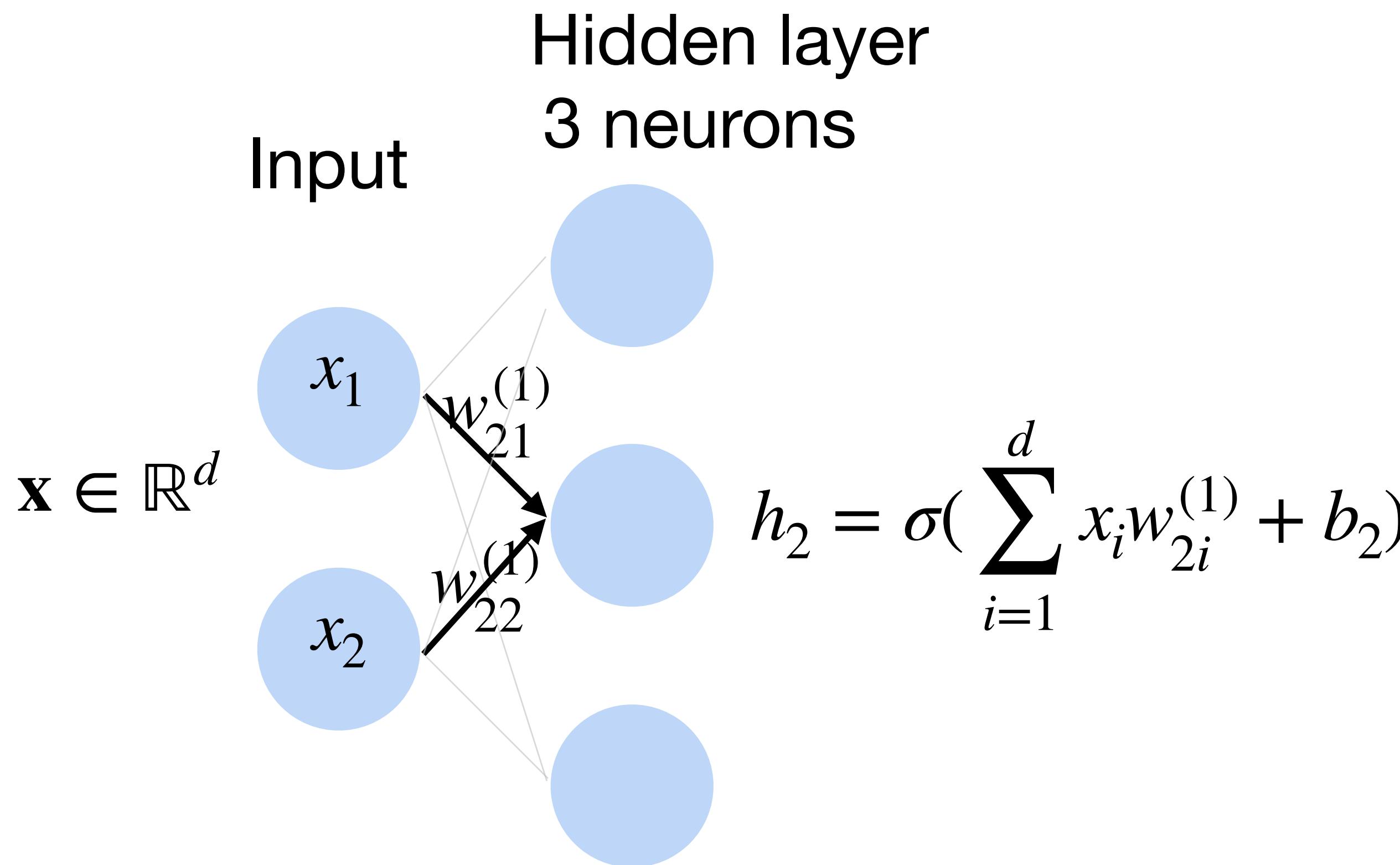
Multi-layer perceptron: Example

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



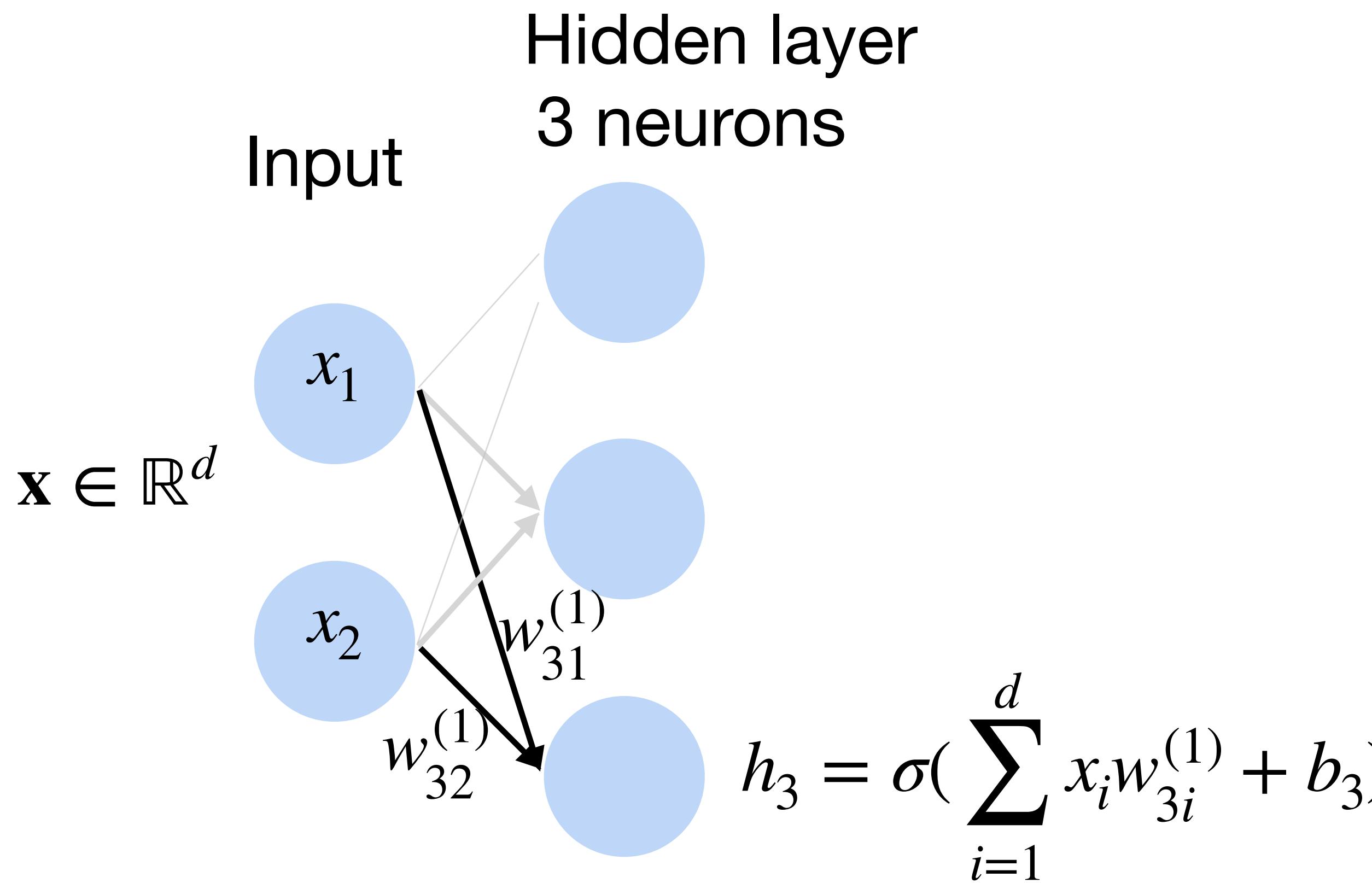
Multi-layer perceptron: Example

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



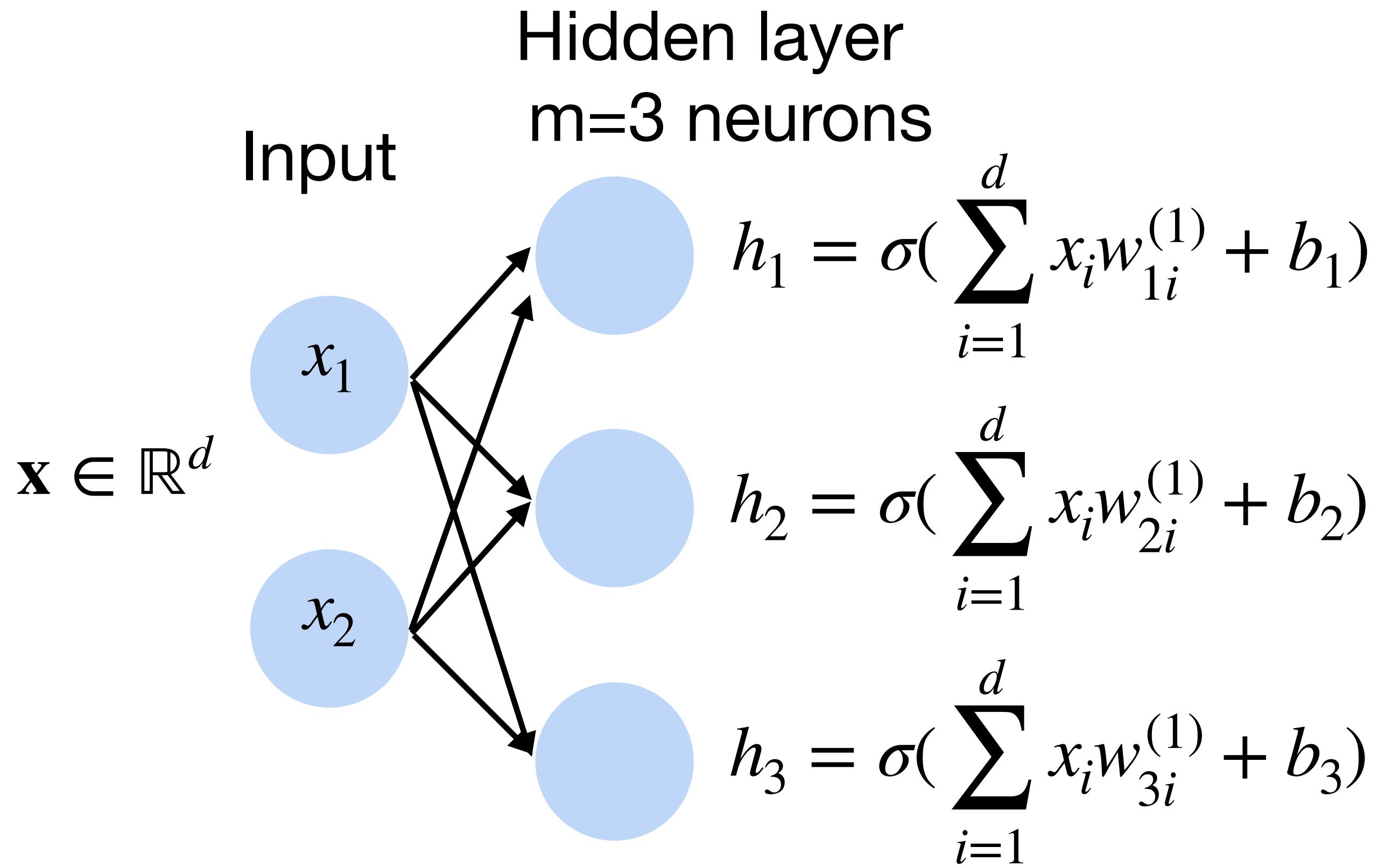
Multi-layer perceptron: Example

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



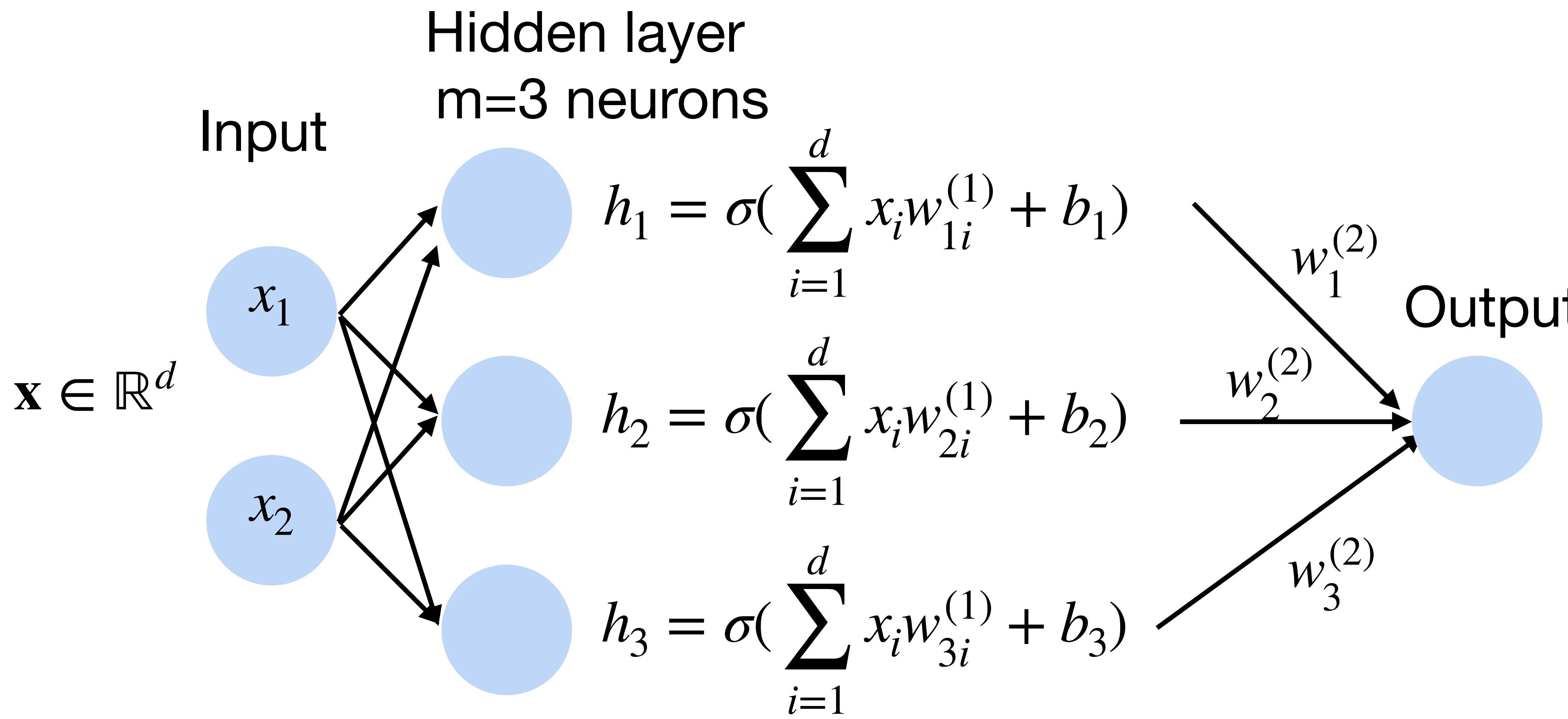
Multi-layer perceptron: Example

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



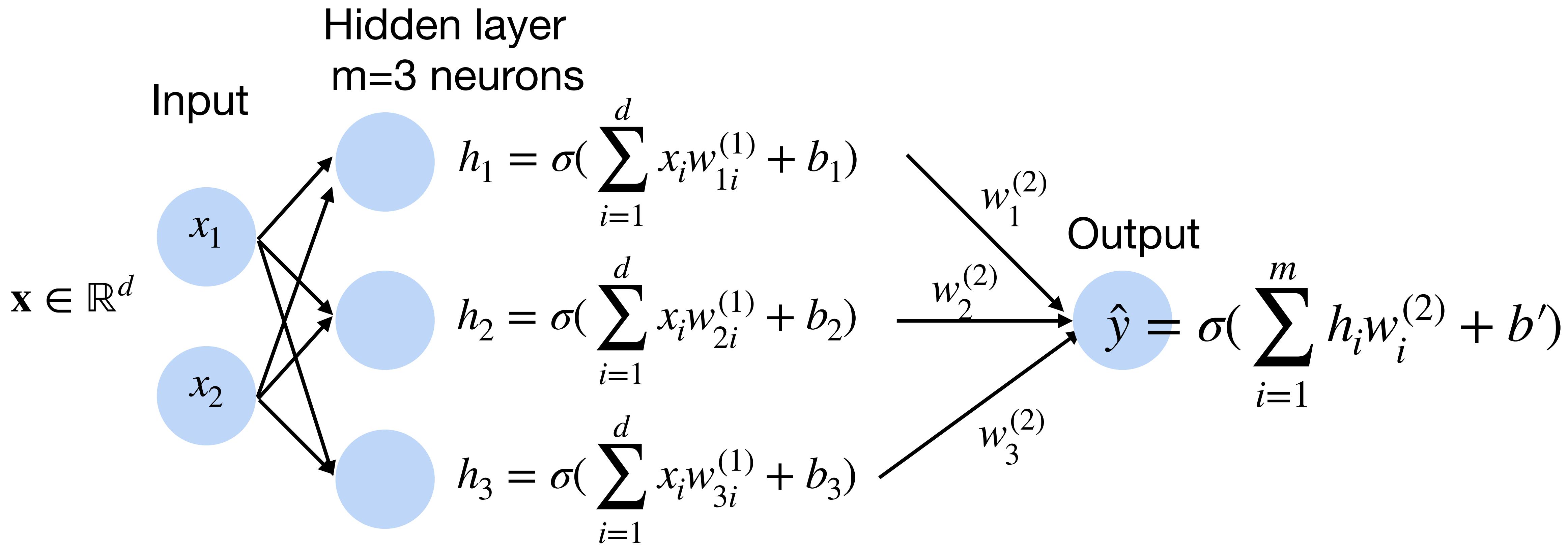
Multi-layer perceptron: Example

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



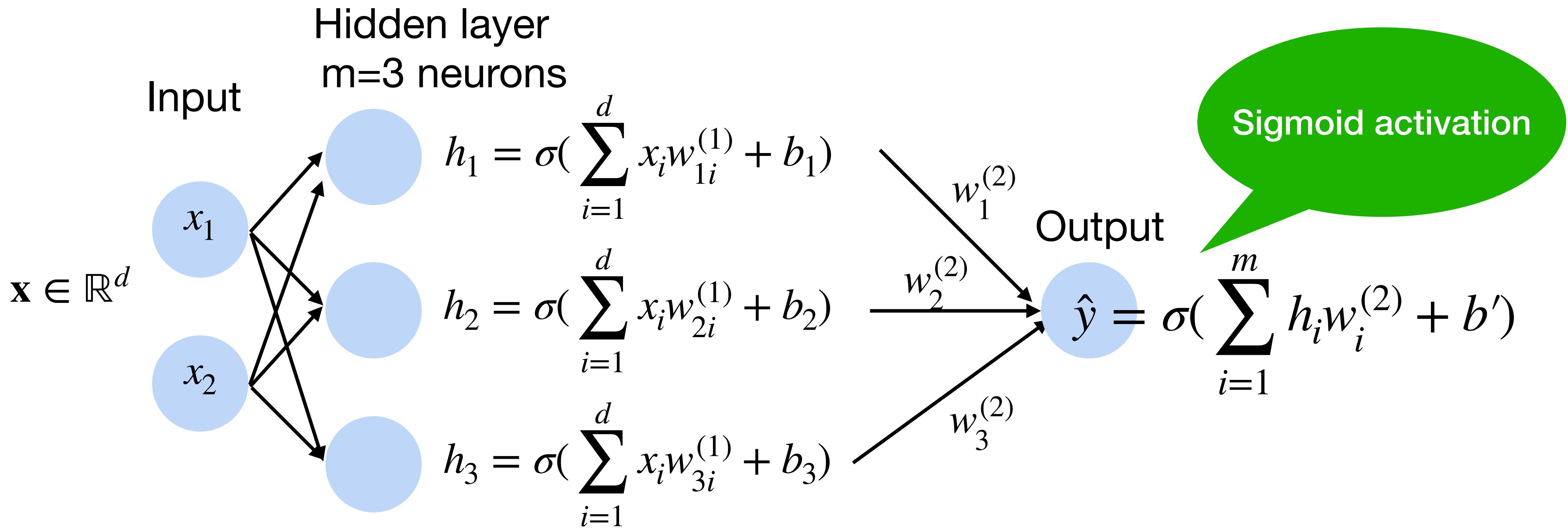
Multi-layer perceptron: Example

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



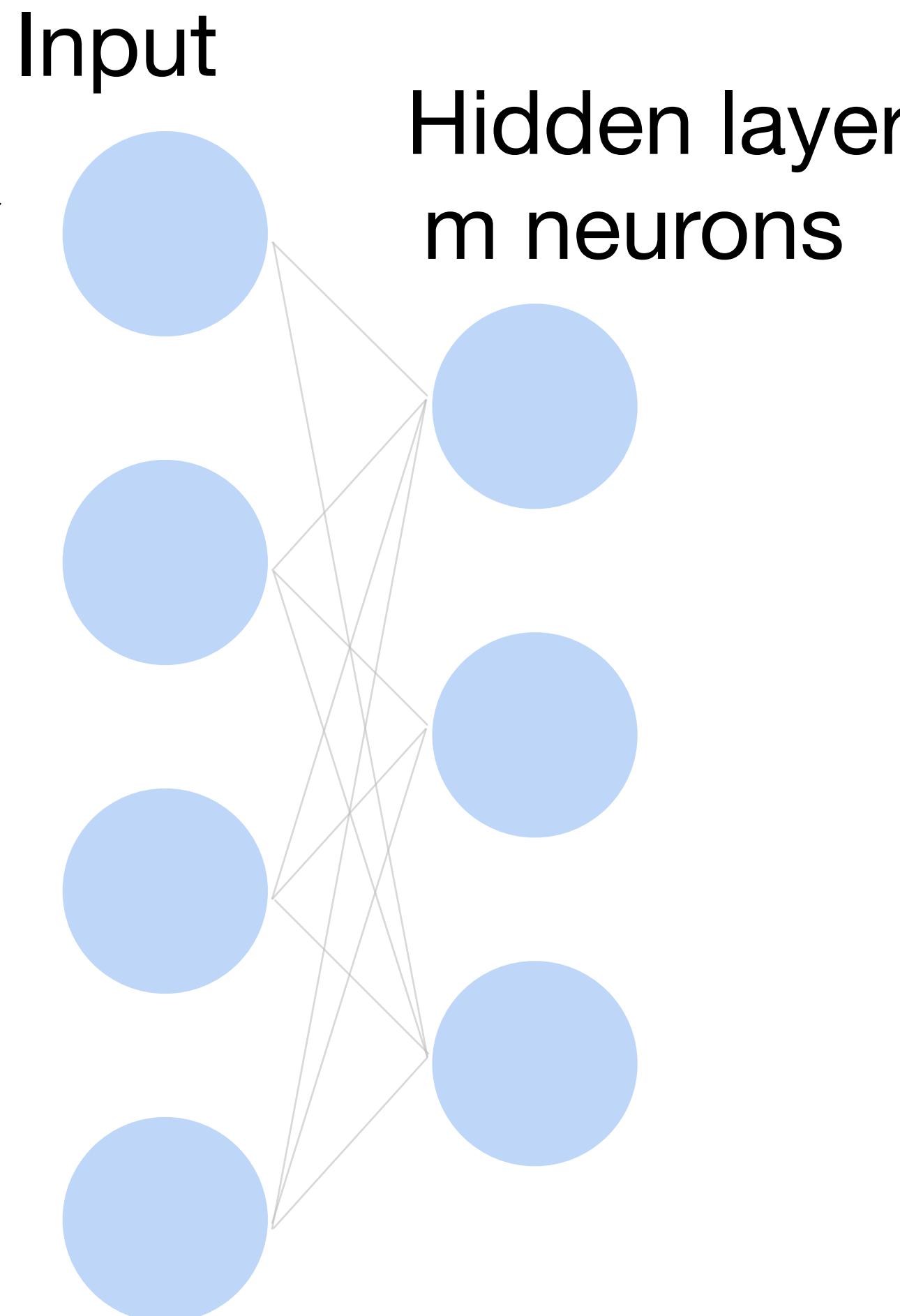
Multi-layer perceptron: Example

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

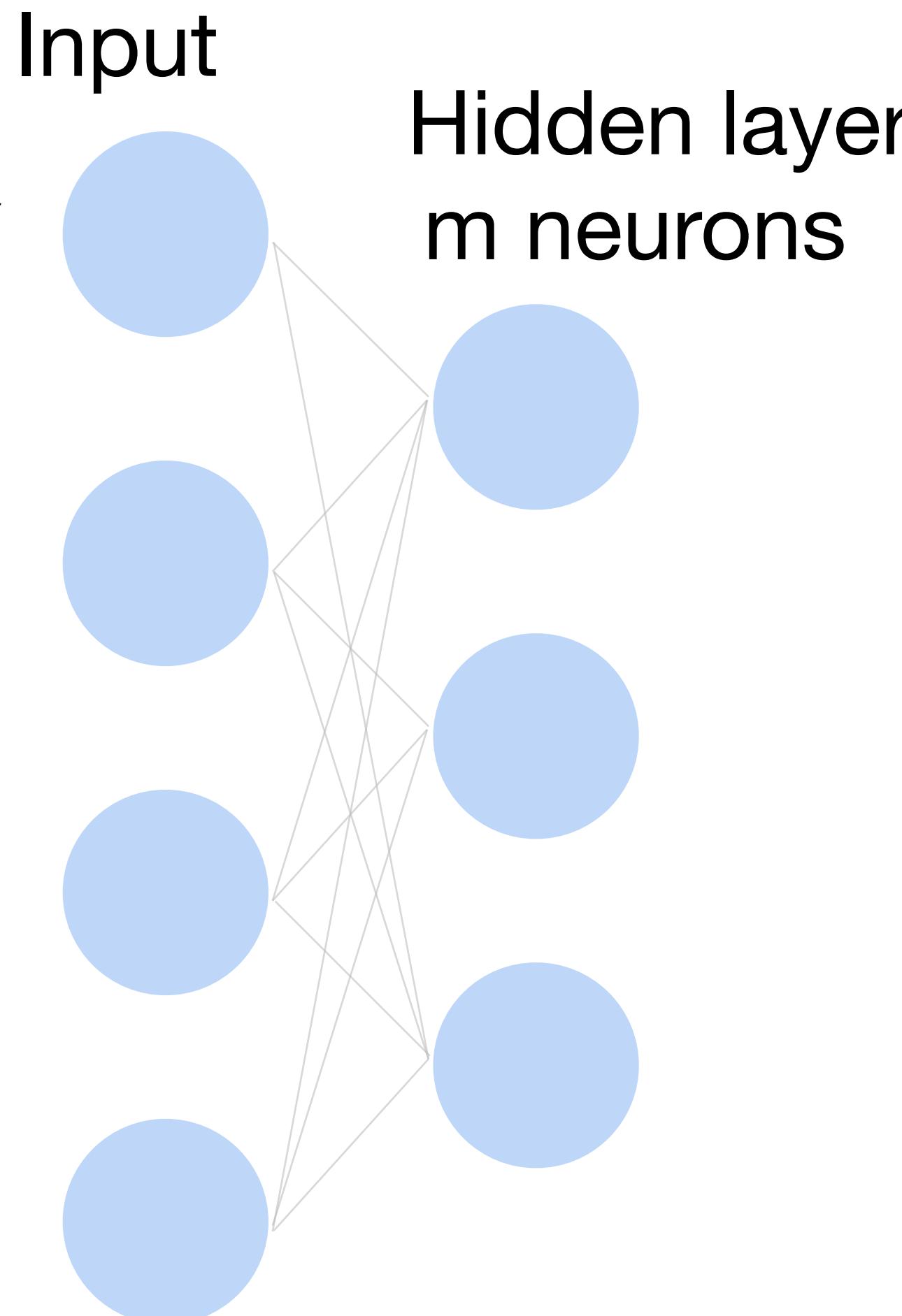


Multi-layer perceptron: Matrix Notation

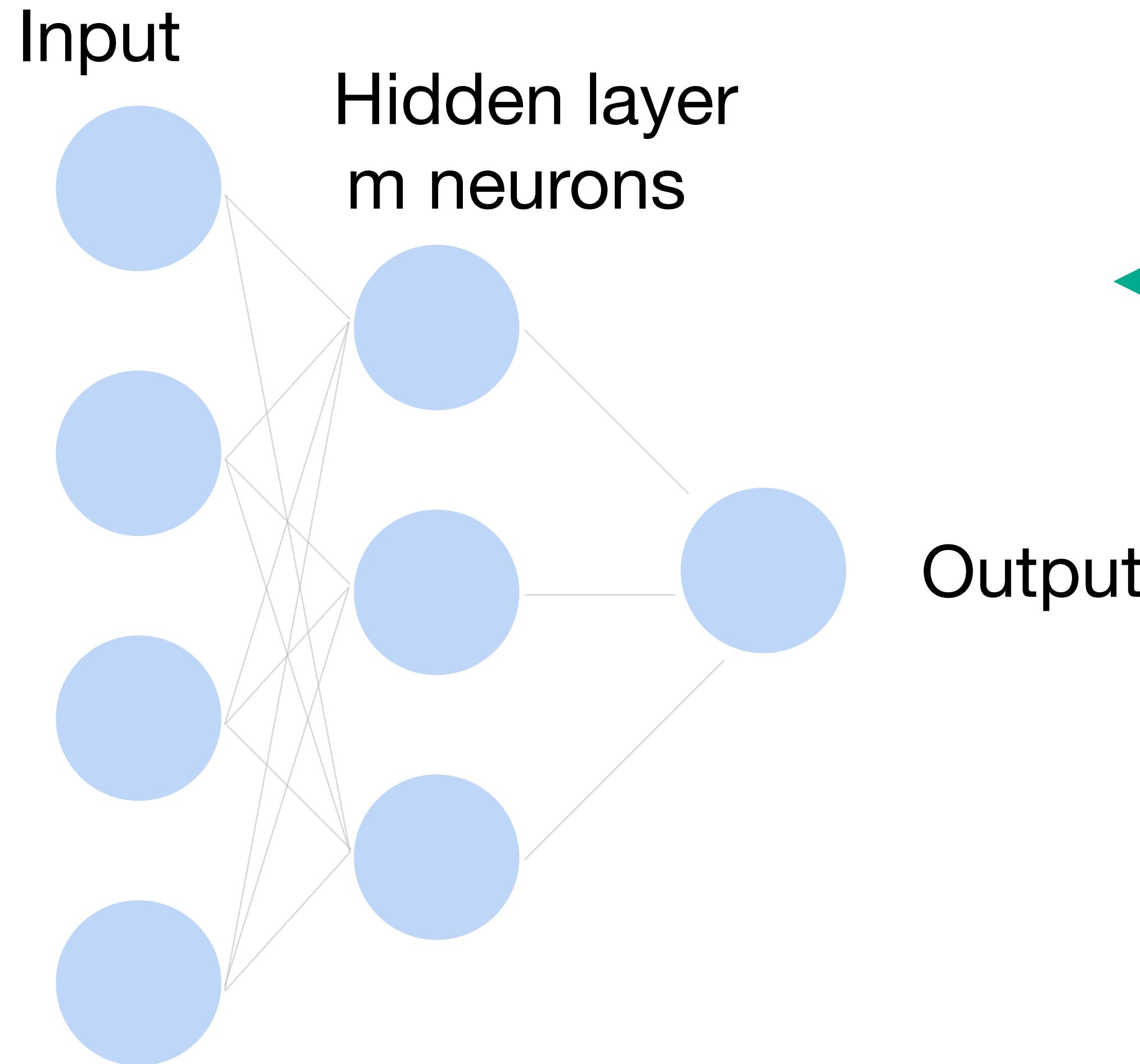
- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$
- Intermediate output

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

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

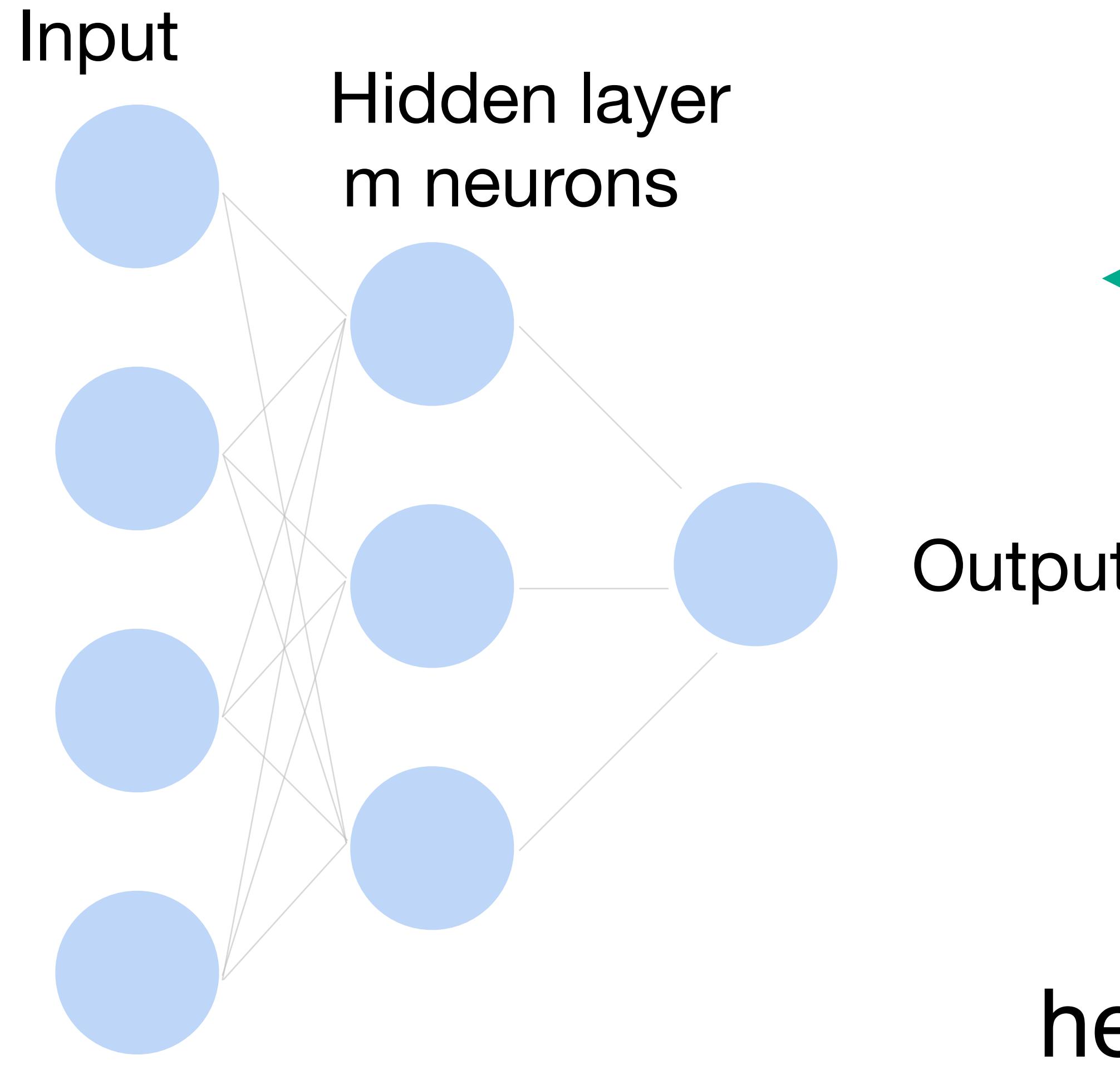


Multi-layer perceptron



Why do we need an a
nonlinear activation?

Multi-layer perceptron



Why do we need an a
nonlinear activation?

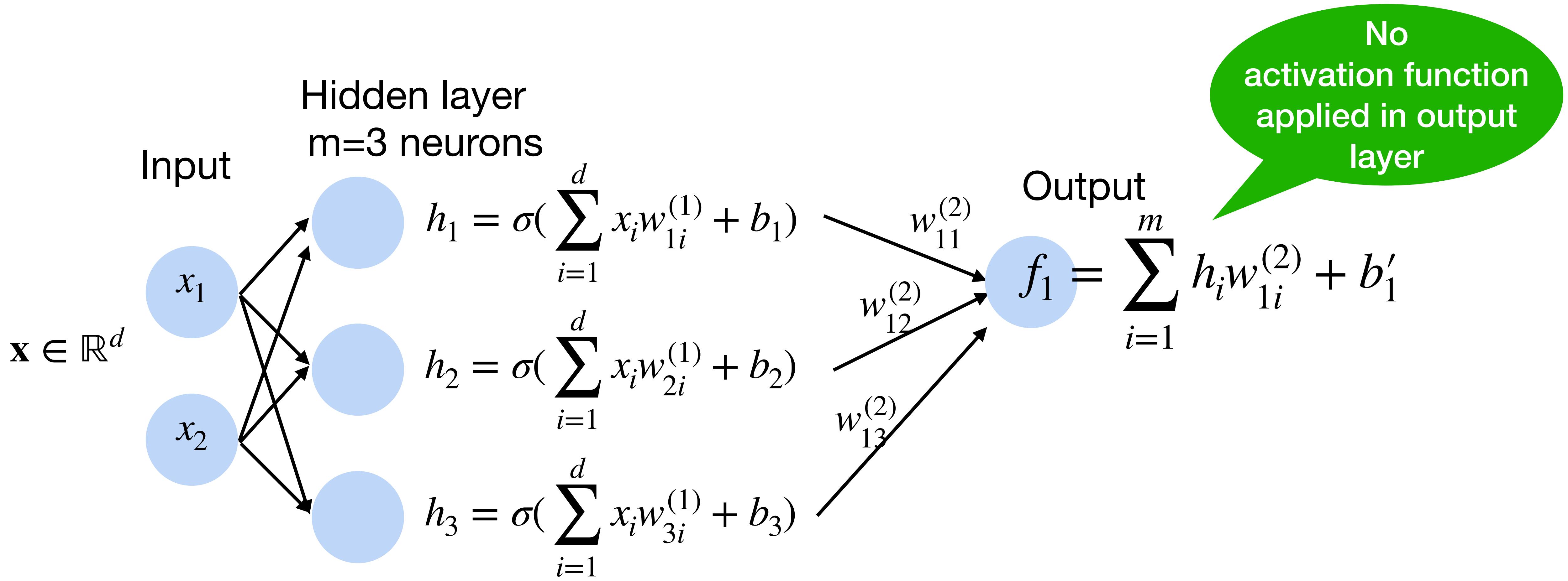
$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$f = \mathbf{w}_2^T \mathbf{h} + b_2$$

$$\text{hence } f = \mathbf{w}_2^T \mathbf{W}\mathbf{x} + b'$$

Neural network for k-way classification

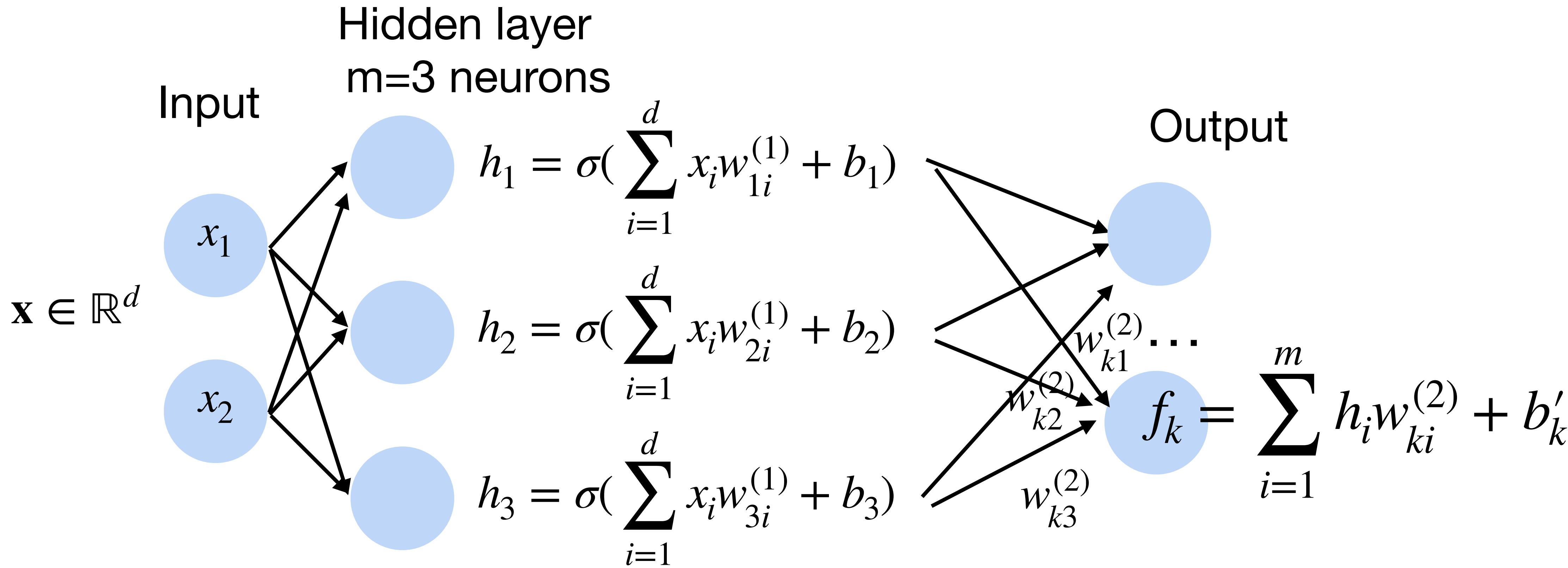
- K outputs in the final layer



Neural network for k-way classification

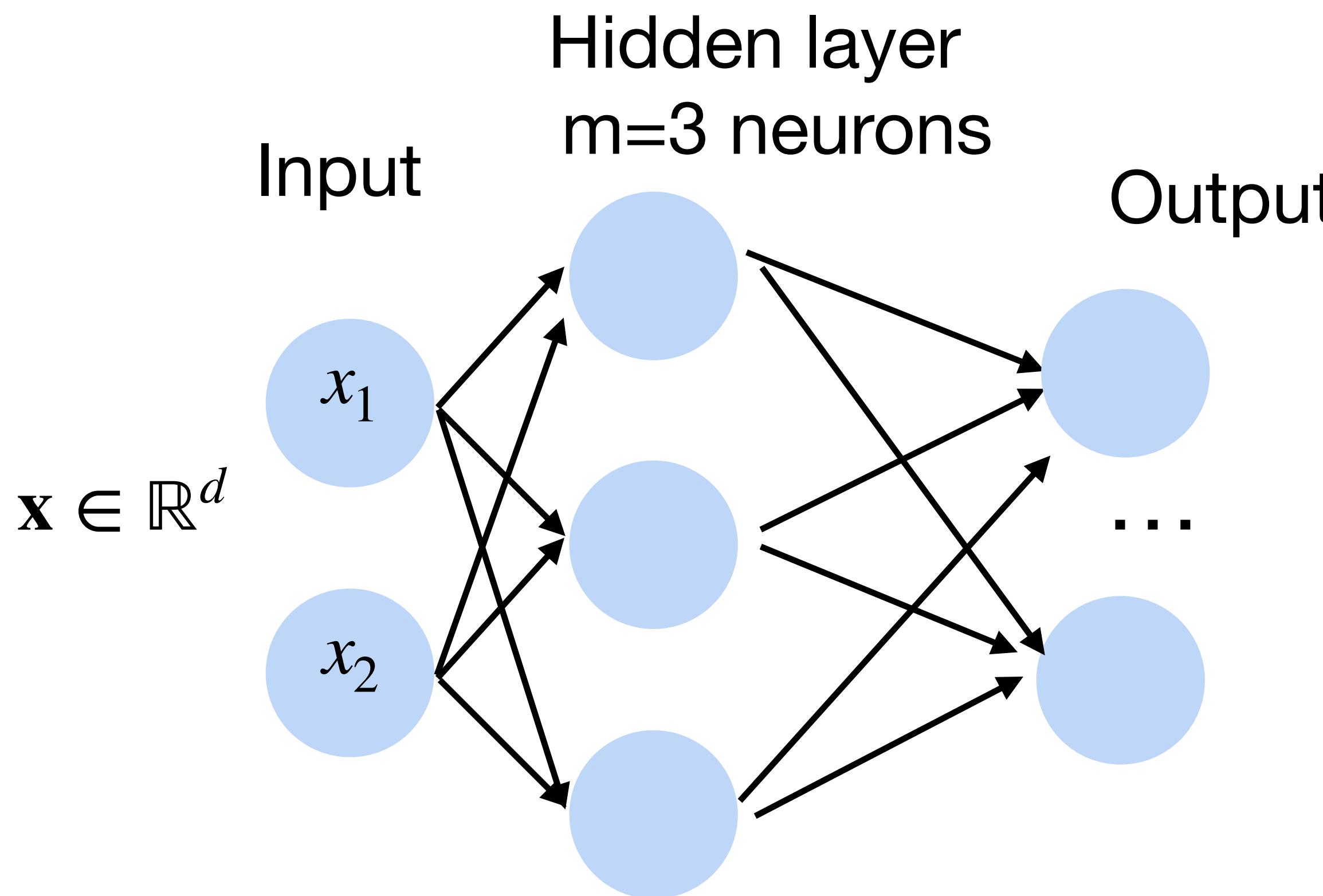
- K outputs units in the final layer

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



Softmax

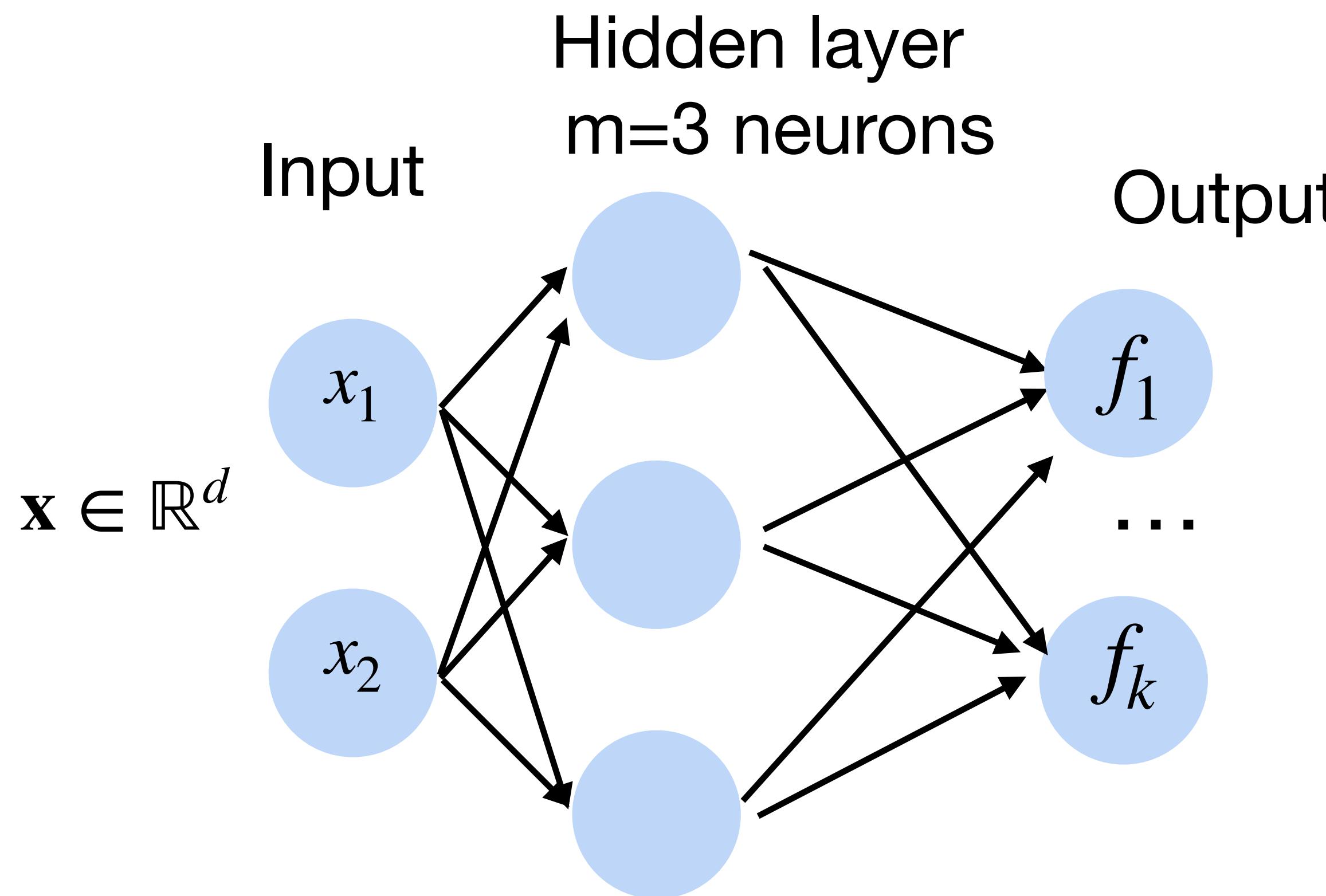
Turns outputs f into probabilities (sum up to 1 across k classes)



$$p(y | \mathbf{x}) = \text{softmax}(f)$$
$$= \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)}$$

Softmax

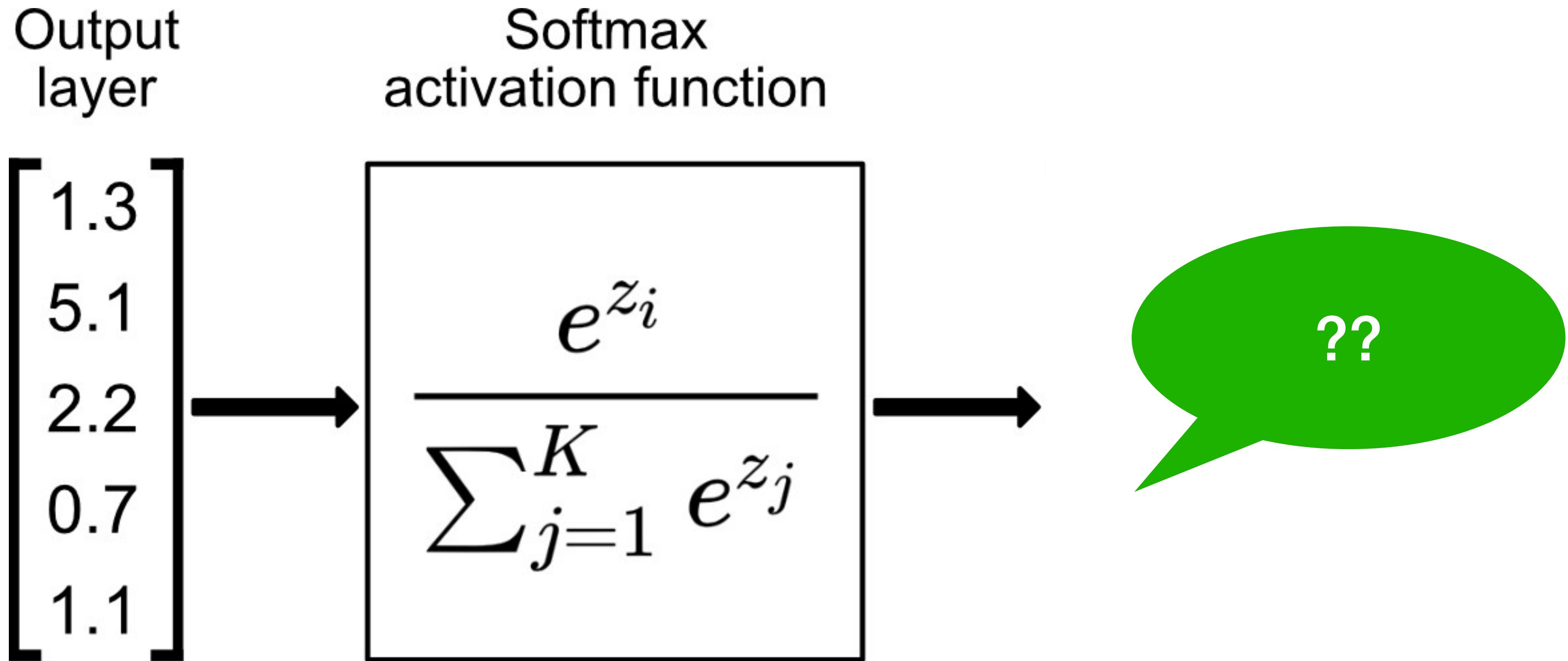
Turns outputs f into probabilities (sum up to 1 across k classes)



$$p(y | \mathbf{x}) = \text{softmax}(f) = \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)}$$

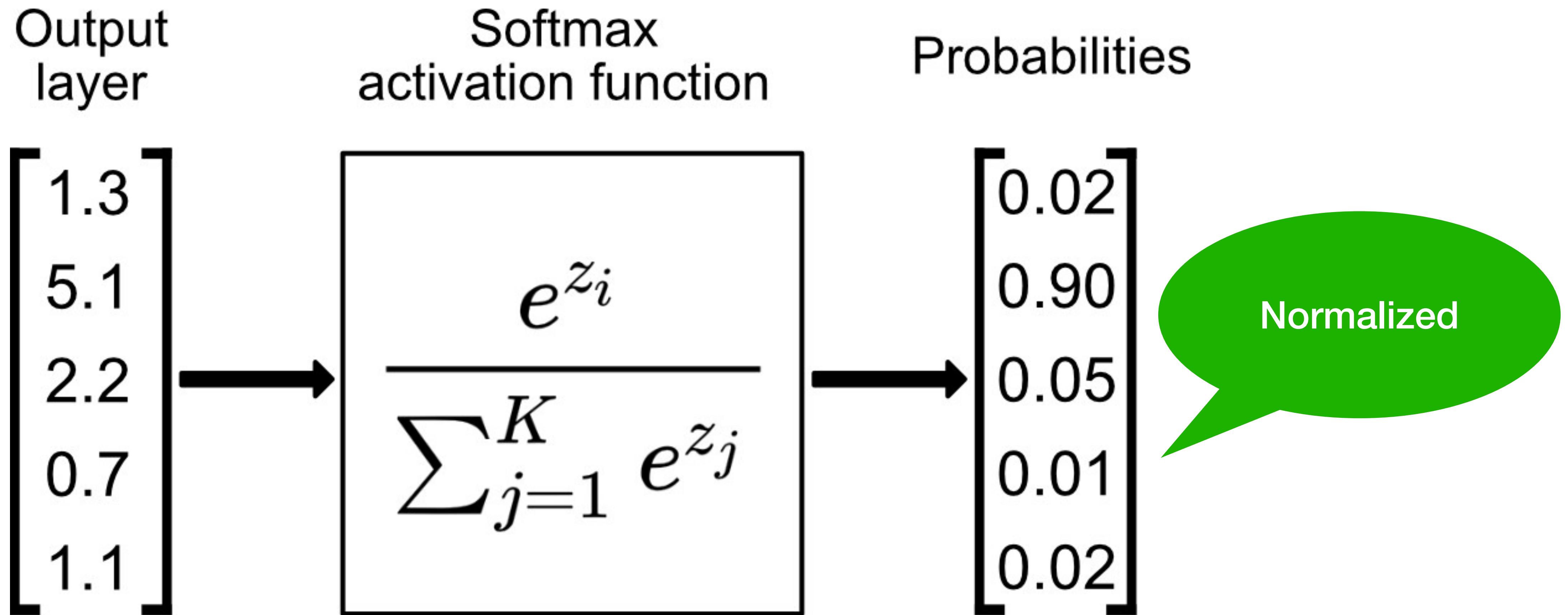
Softmax

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



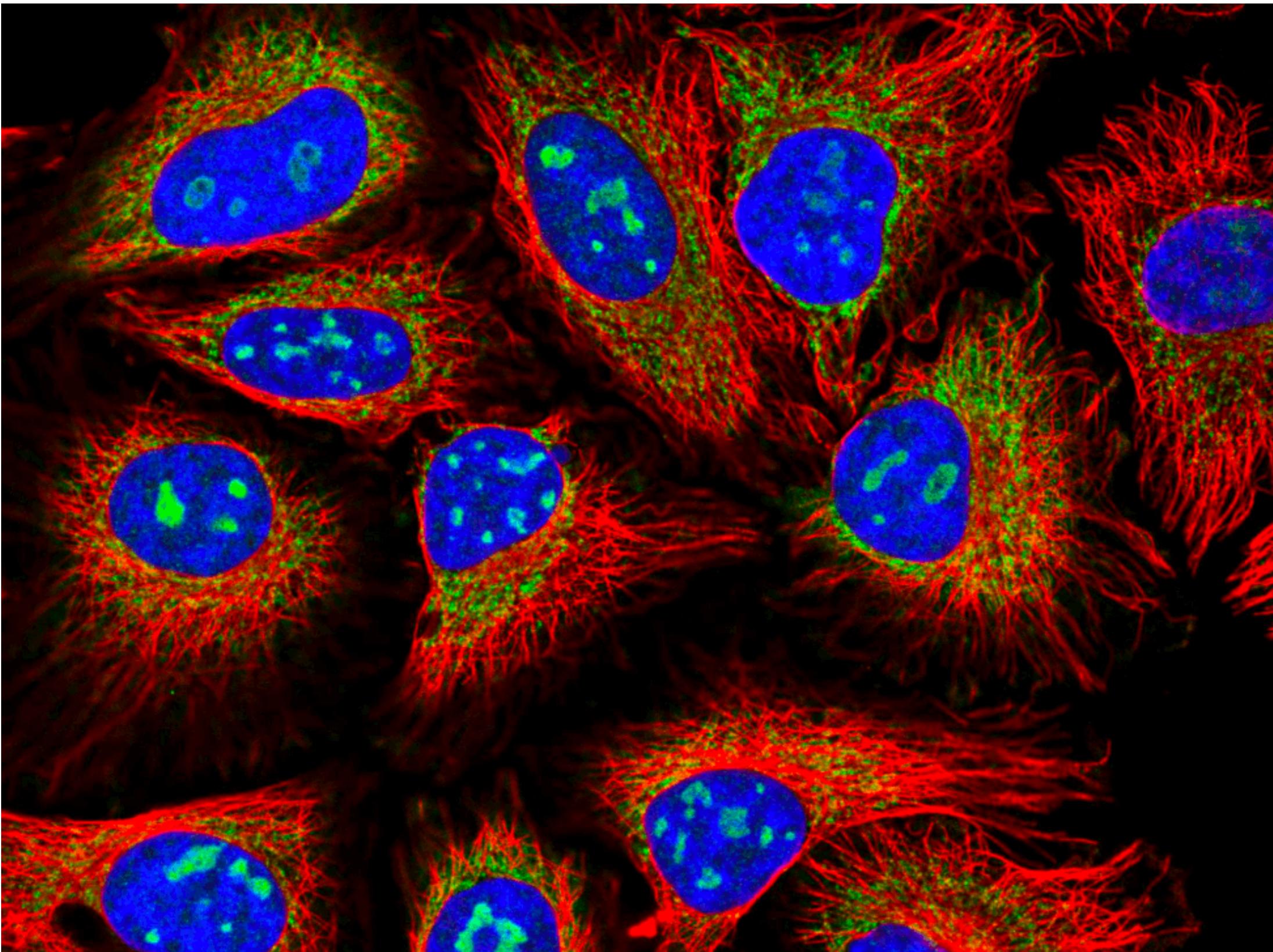
Softmax

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



Classification Tasks at Kaggle

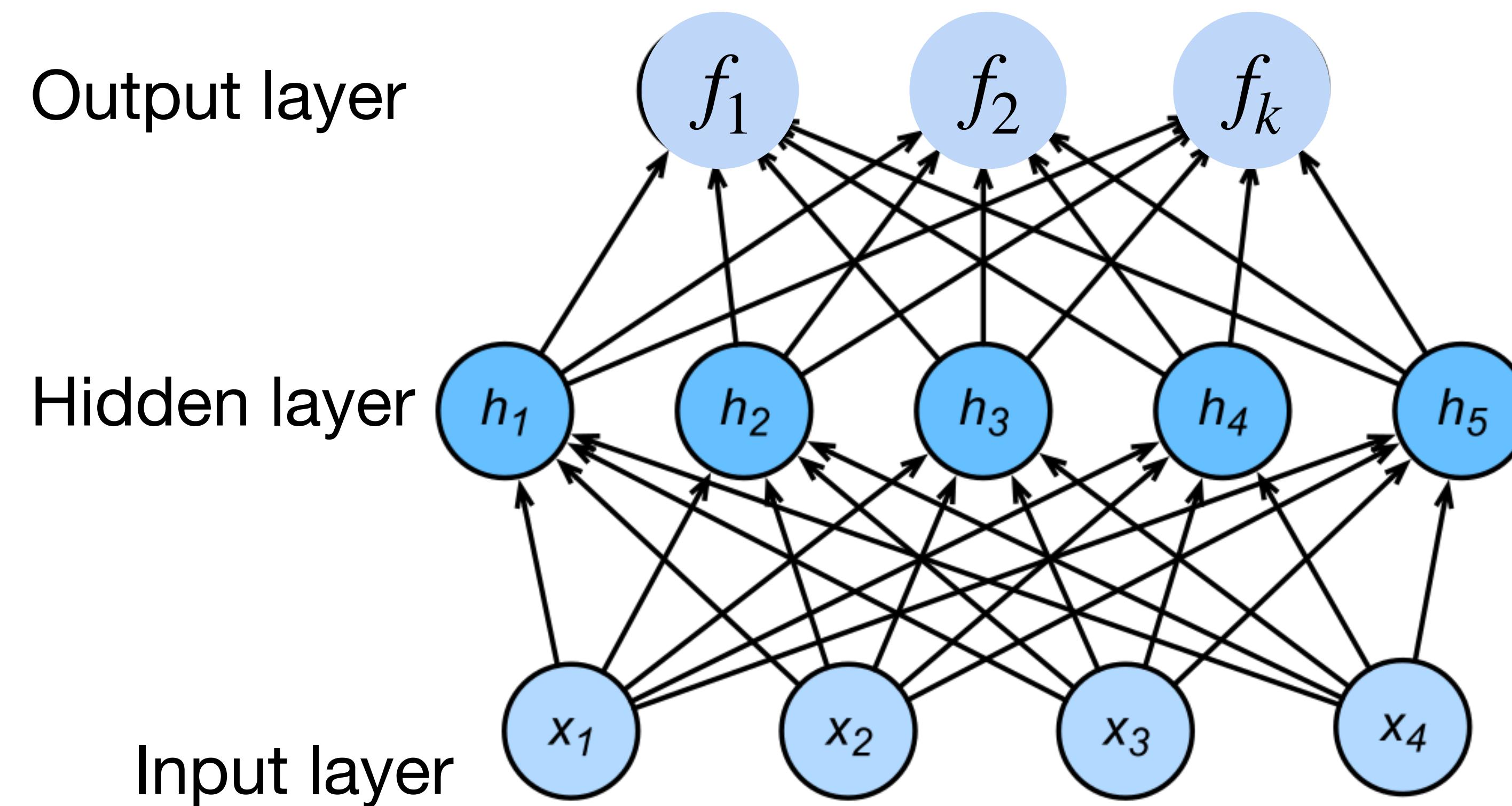
Classify human protein microscope images into 28 categories



- 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

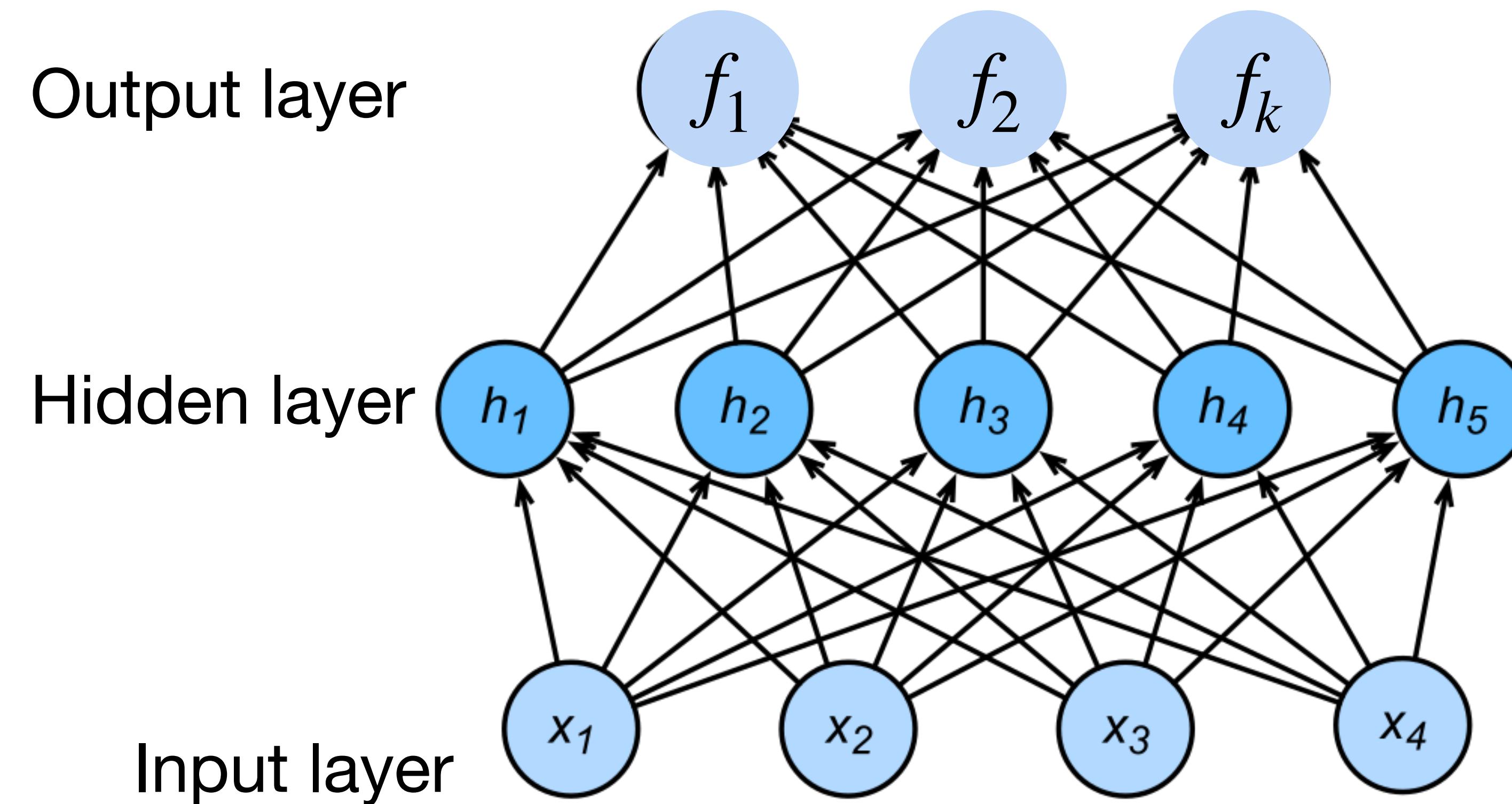
<https://www.kaggle.com/c/human-protein-atlas-image-classification>

More complicated neural networks



More complicated neural networks

$$y_1, y_2, \dots, y_k = \text{softmax}(f_1, f_2, \dots, f_k)$$



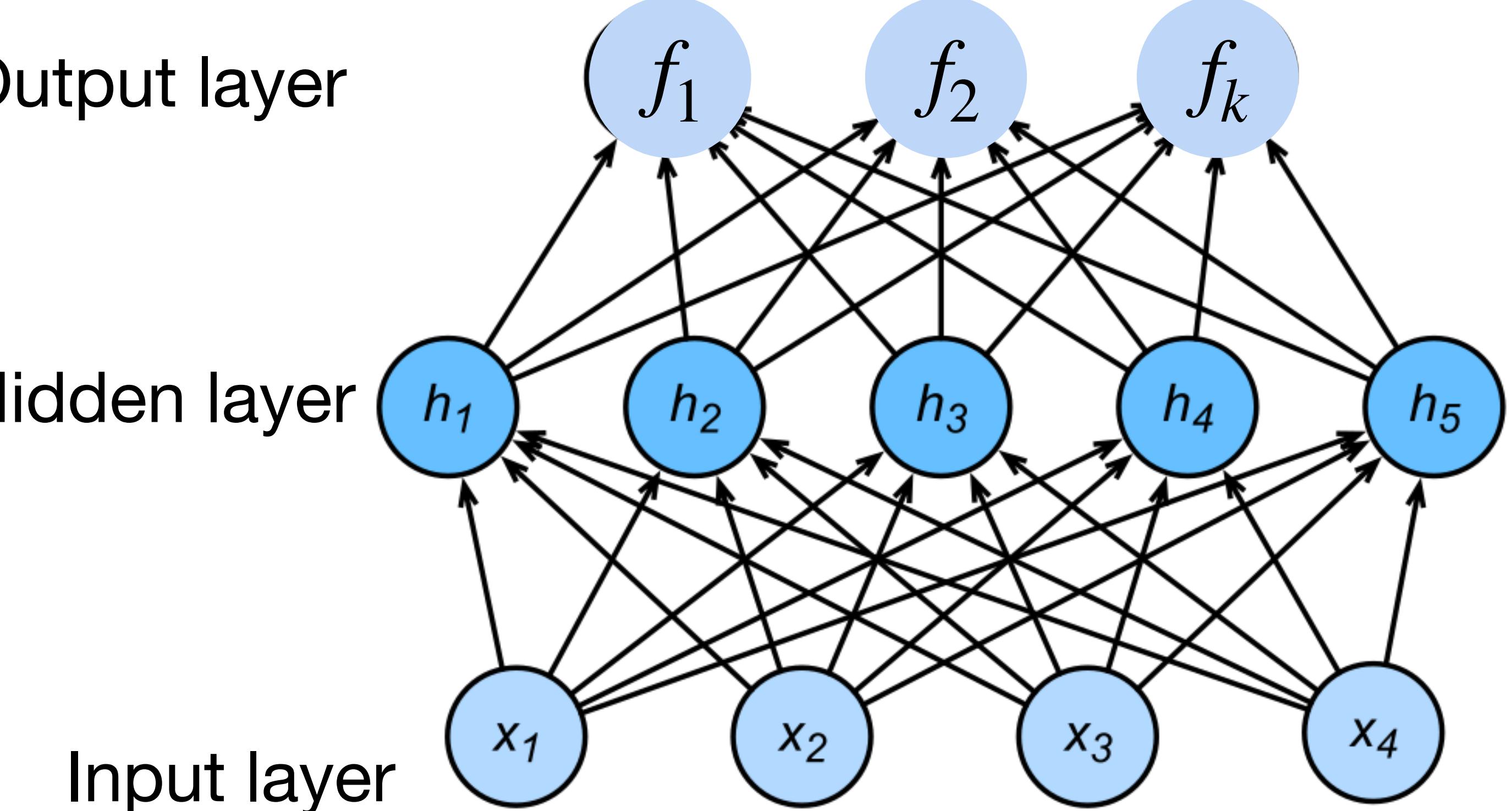
More complicated neural networks

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

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

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

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



More complicated neural networks

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

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

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

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

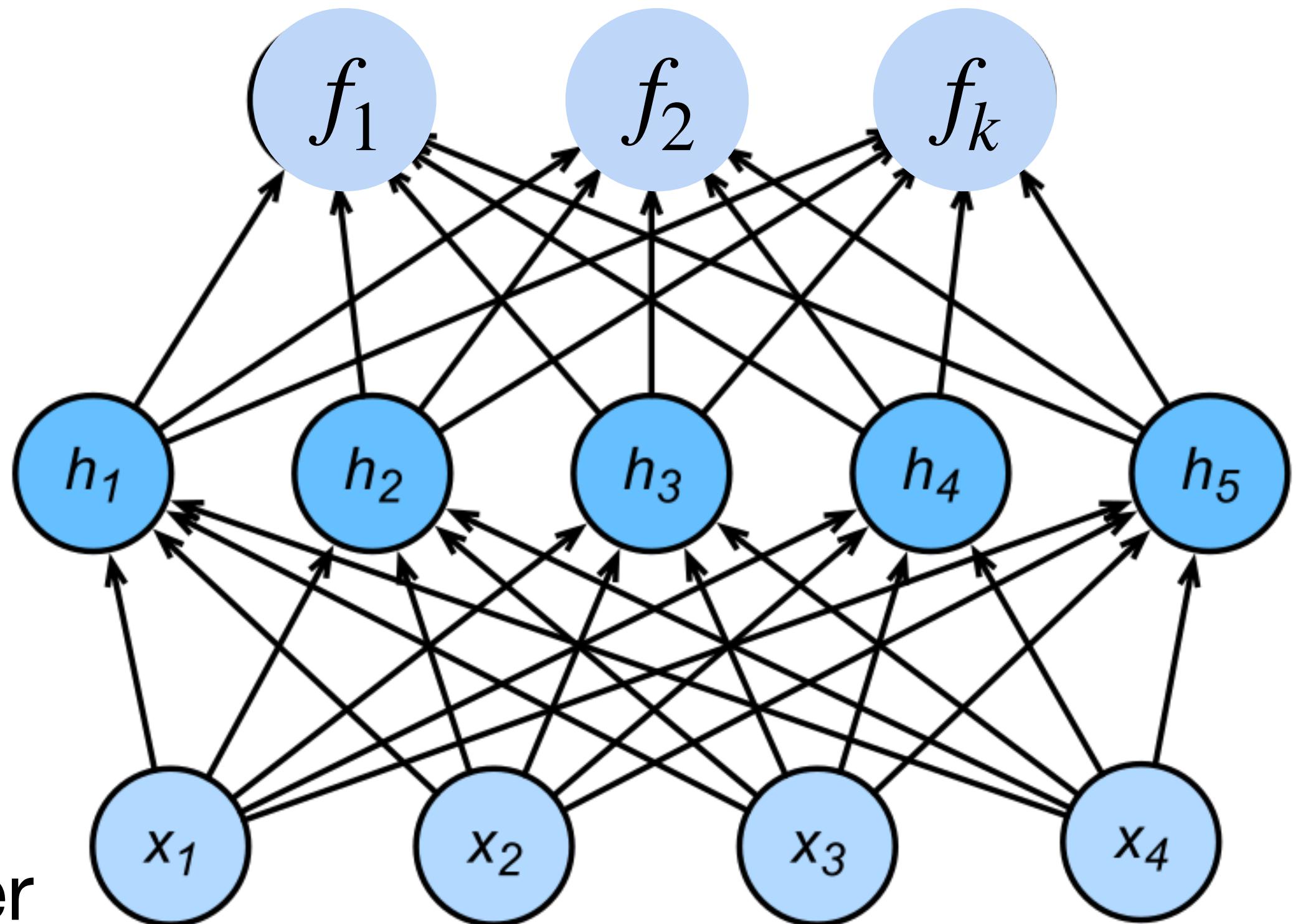
$$\mathbf{y} = \text{softmax}(\mathbf{f})$$

$$y_1, y_2, \dots, y_k = \text{softmax}(f_1, f_2, \dots, f_k)$$

Output layer

Hidden layer

Input layer



More complicated neural networks: multiple hidden layers

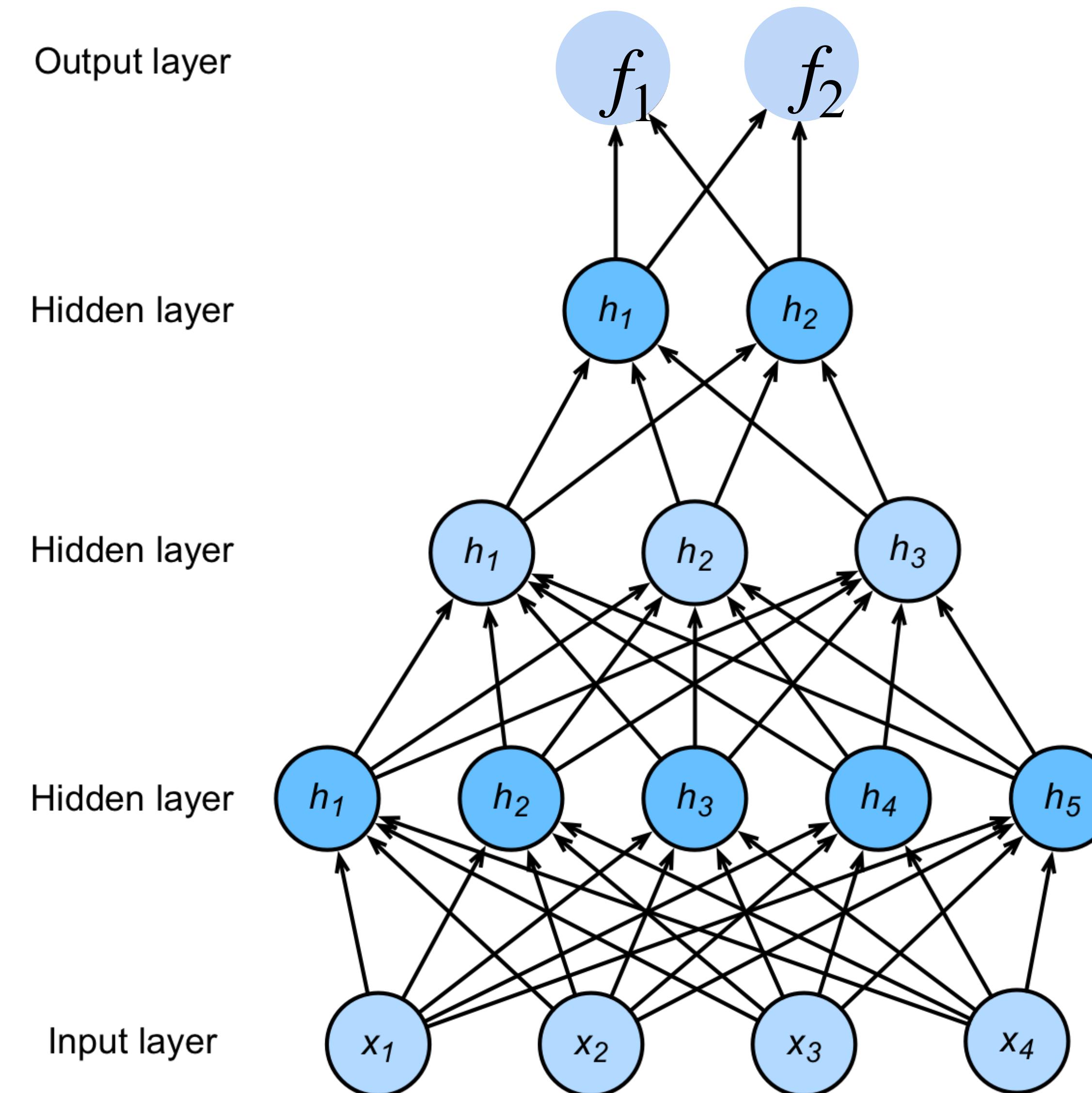
$$\mathbf{h}_1 = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

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

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

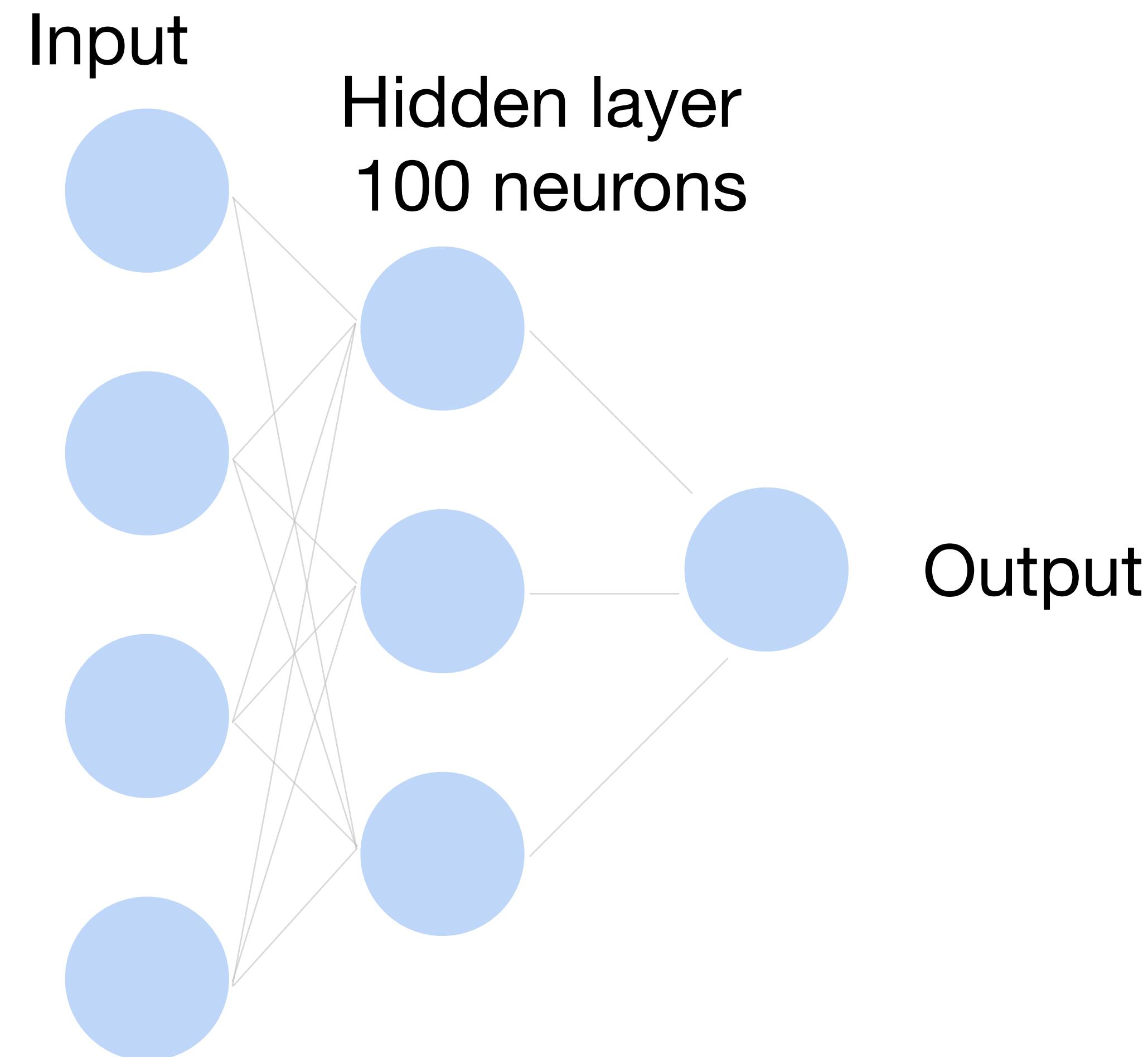
$$\mathbf{f} = \mathbf{W}_4 \mathbf{h}_3 + \mathbf{b}_4$$

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



How to train a neural network?

Classify cats vs. dogs



How to train a neural network?

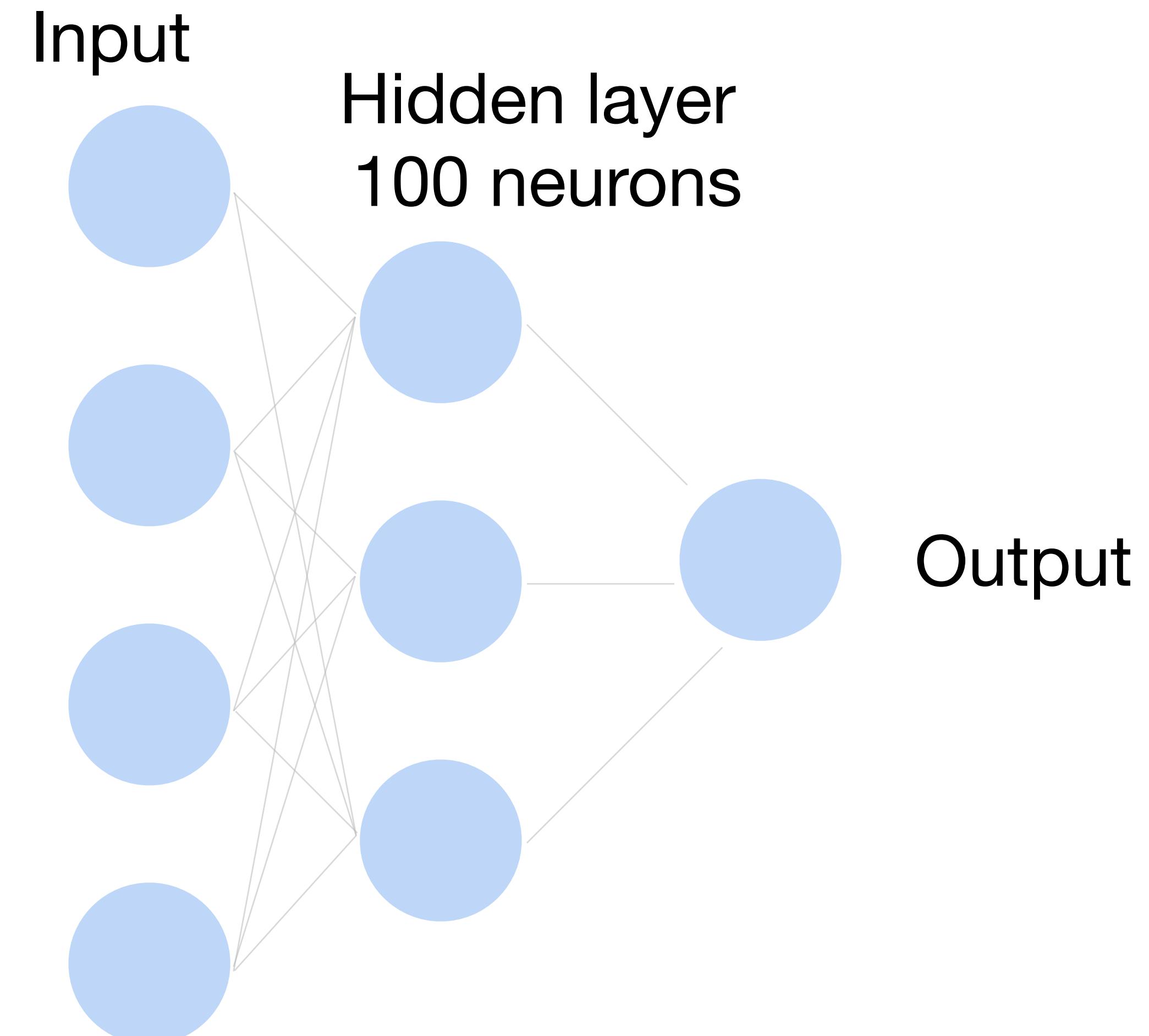
$\mathbf{x} \in \mathbb{R}^d$ One training data point in the training set D

\hat{y} Model output for example \mathbf{x}

y Ground truth label for example \mathbf{x}

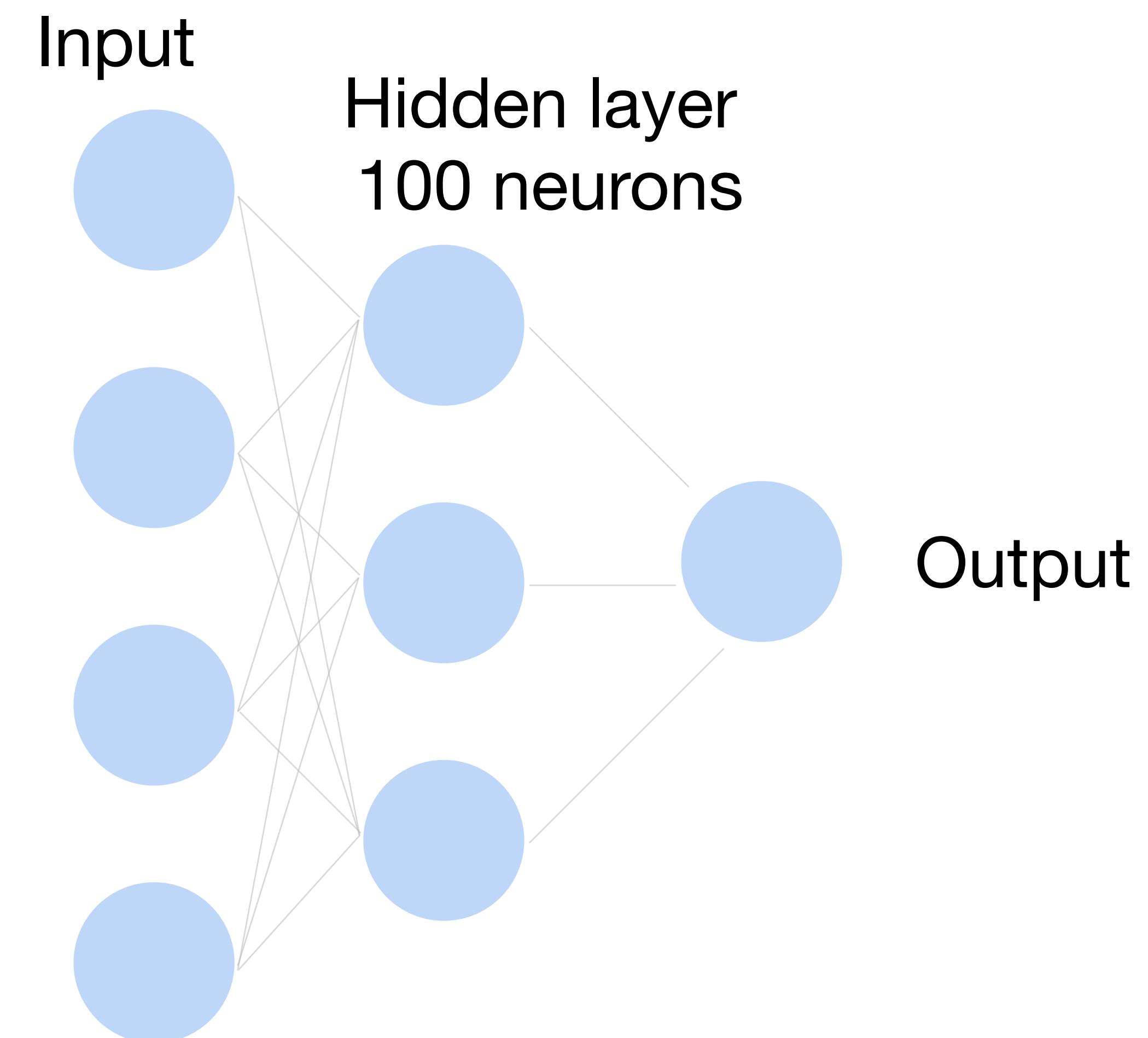
**Learning by matching the output
to the label**

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



How to train a neural network?

Loss function: $\frac{1}{|D|} \sum_i \ell(\mathbf{x}_i, y_i)$

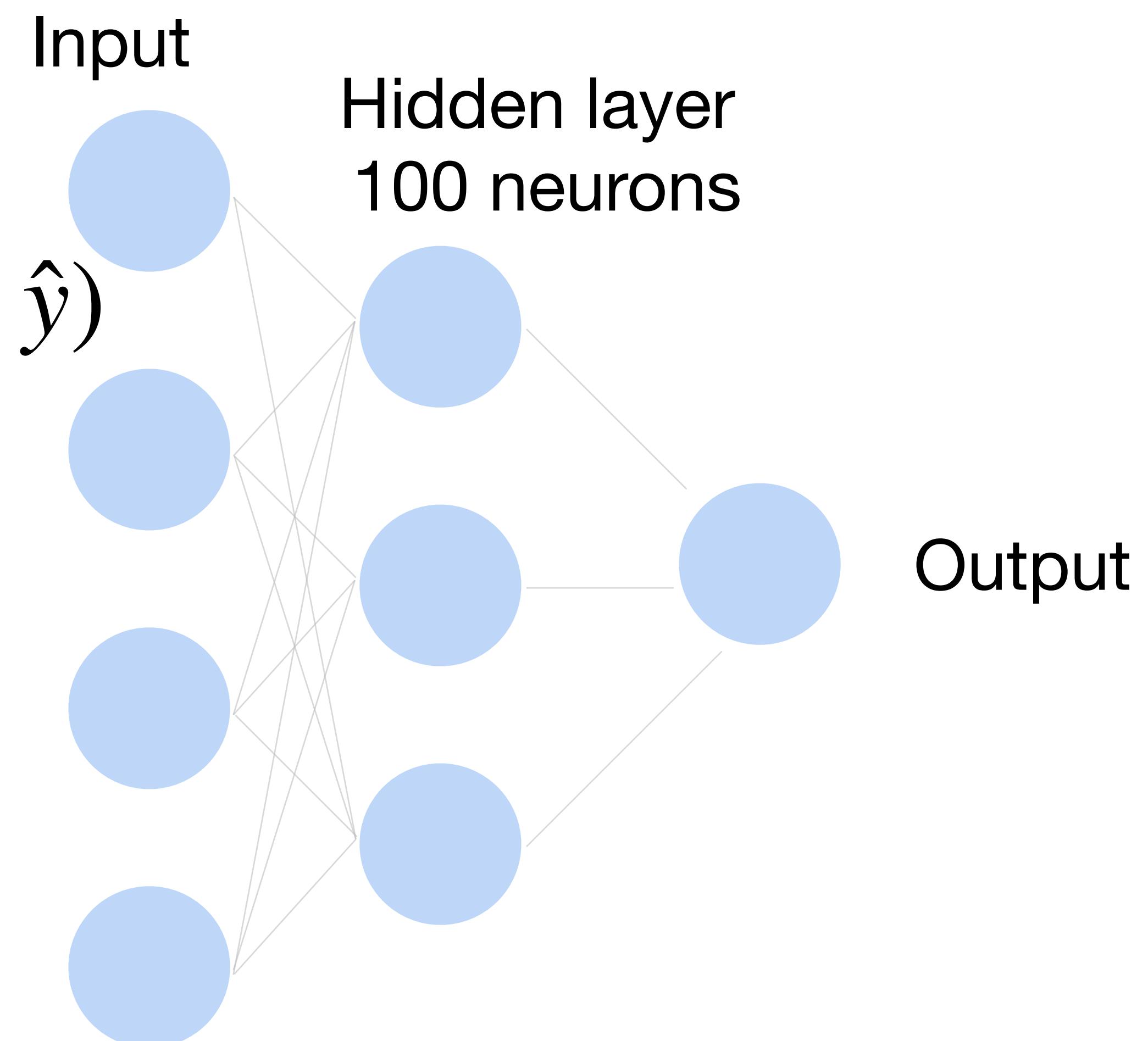


How to train a neural network?

Loss function: $\frac{1}{|D|} \sum_i \ell(\mathbf{x}_i, y_i)$

Per-sample loss:

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

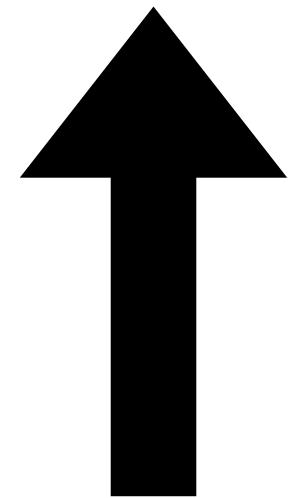


How to train a neural network?

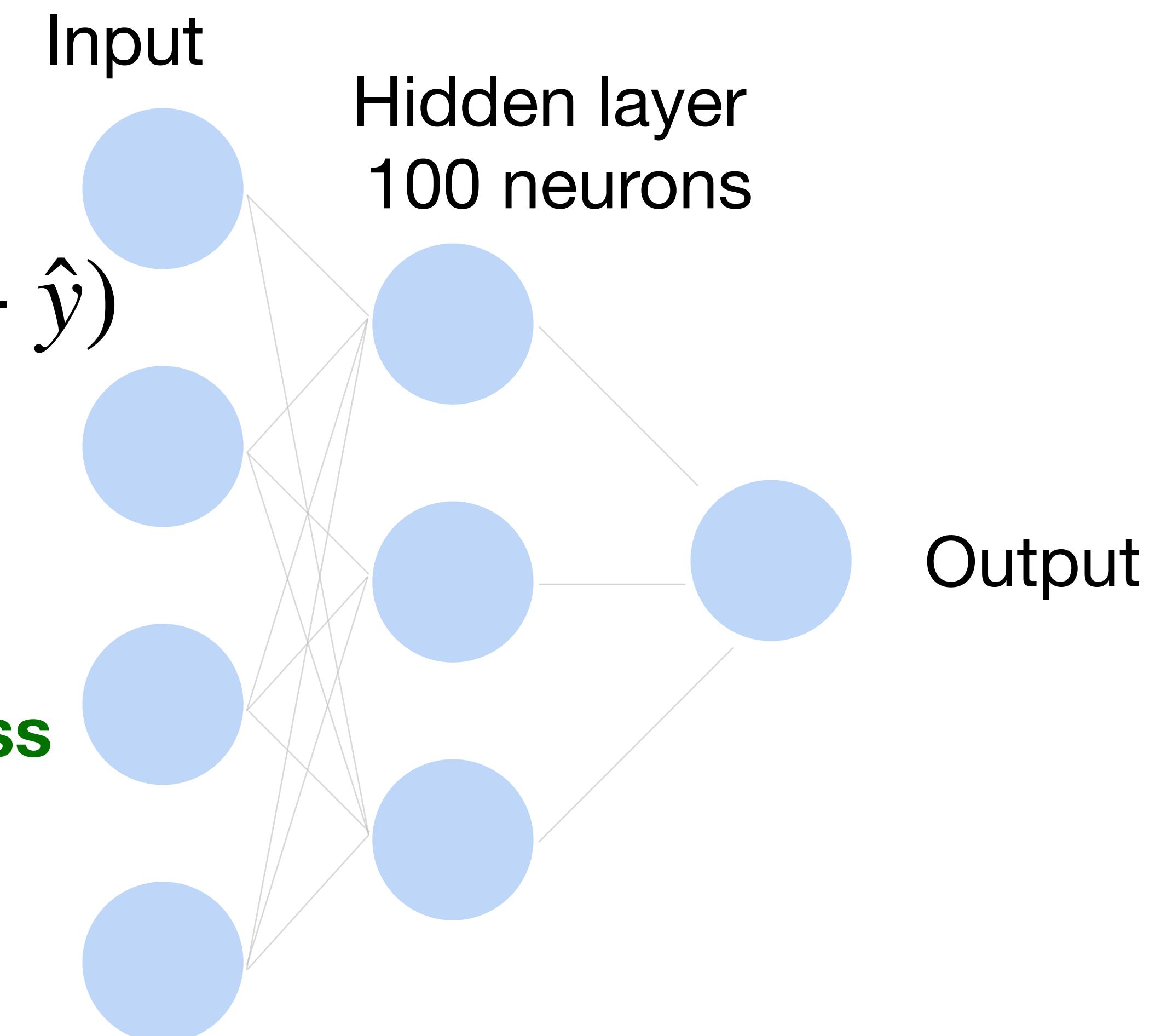
Loss function: $\frac{1}{|D|} \sum_i \ell(\mathbf{x}_i, y_i)$

Per-sample loss:

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

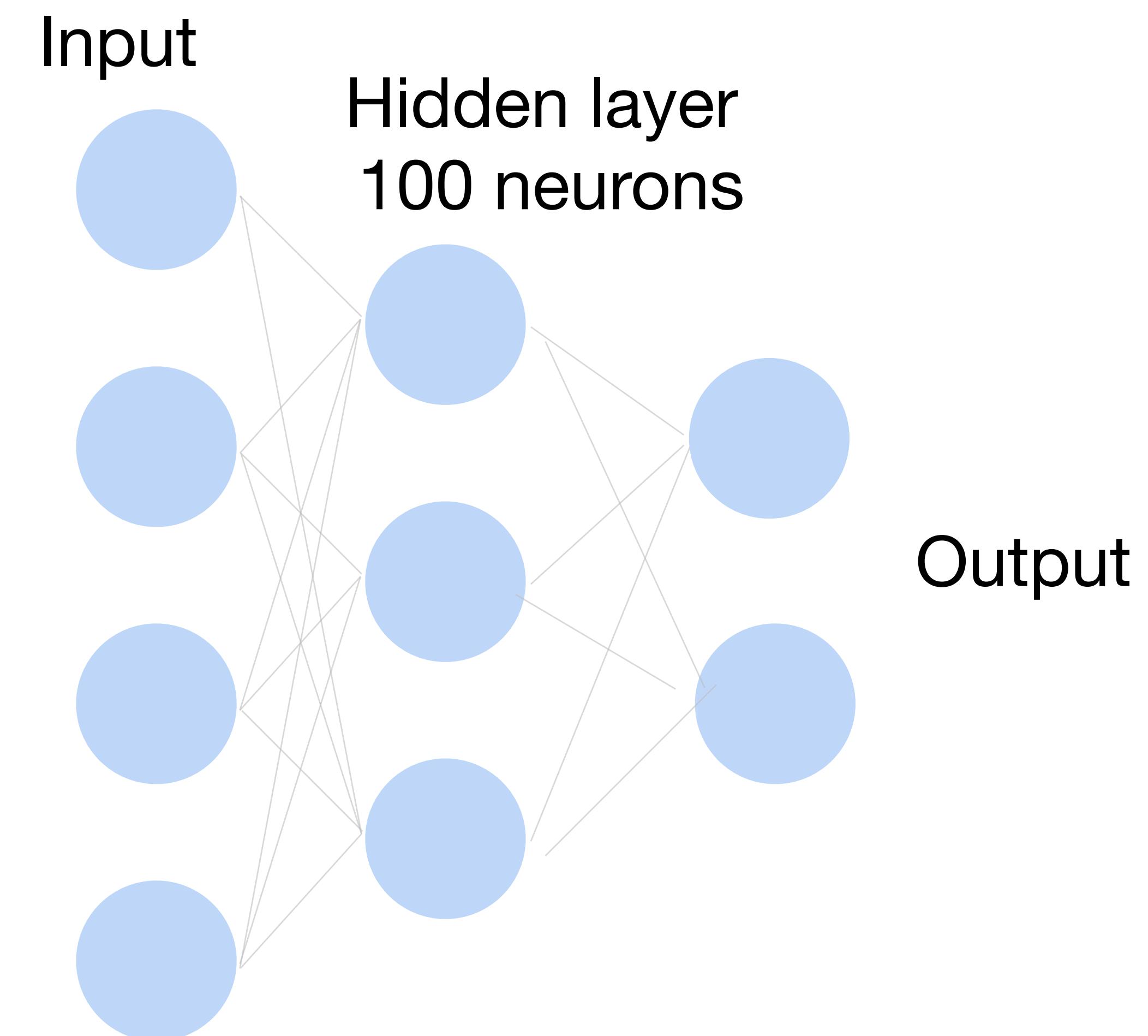


Also known as **binary cross-entropy loss**



How to train a neural network?

Loss function: $\frac{1}{|D|} \sum_i \ell(\mathbf{x}_i, y_i)$

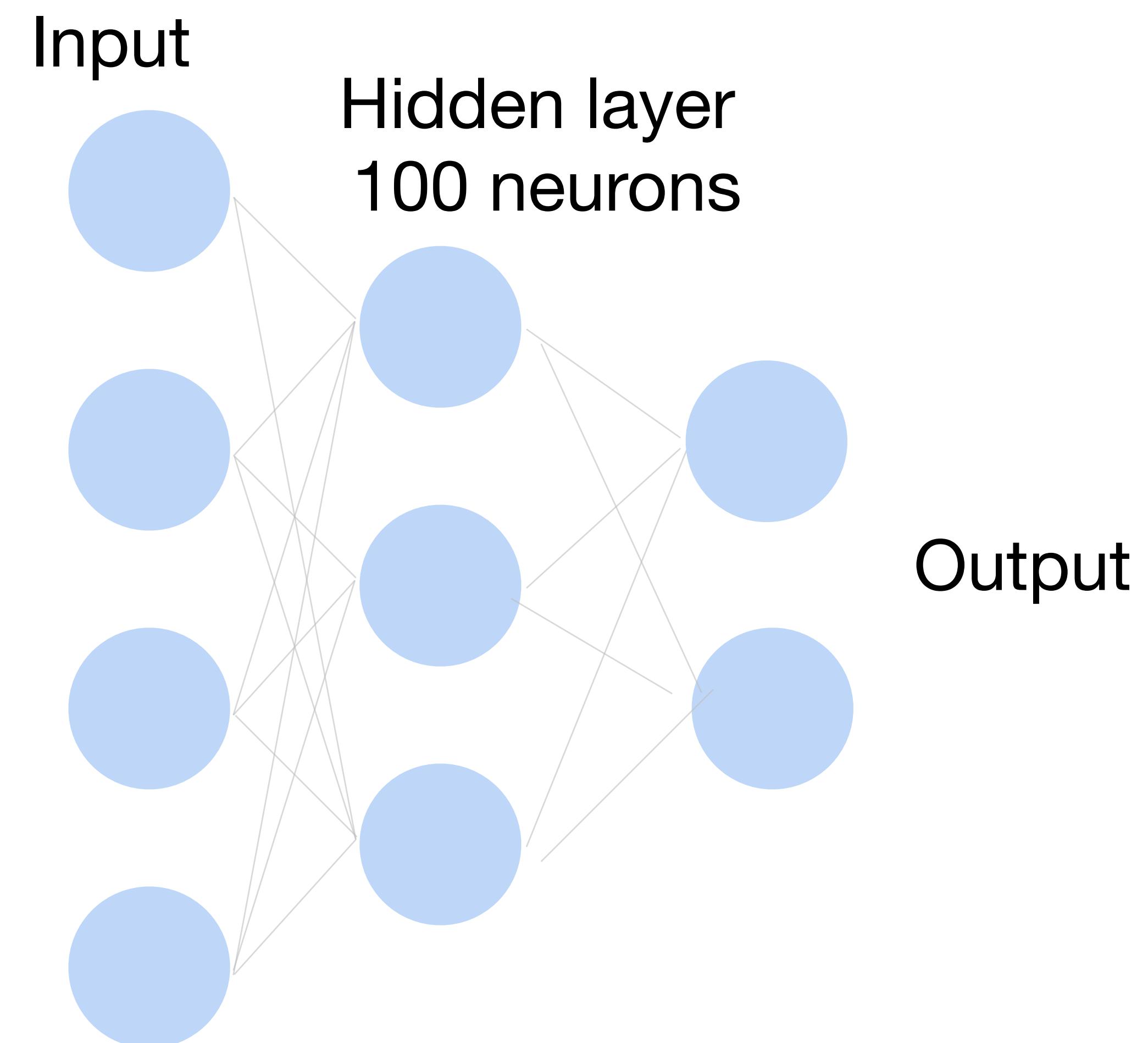


How to train a neural network?

Loss function: $\frac{1}{|D|} \sum_i \ell(\mathbf{x}_i, y_i)$

Per-sample loss:

$$\ell(\mathbf{x}, y) = \sum_{j=1}^K -y_j \log p_j$$

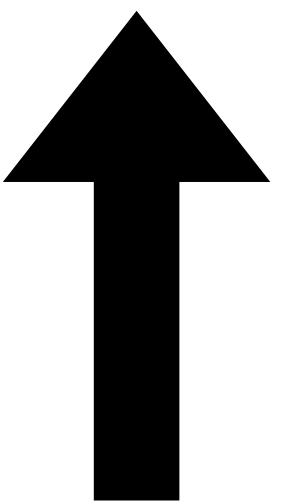


How to train a neural network?

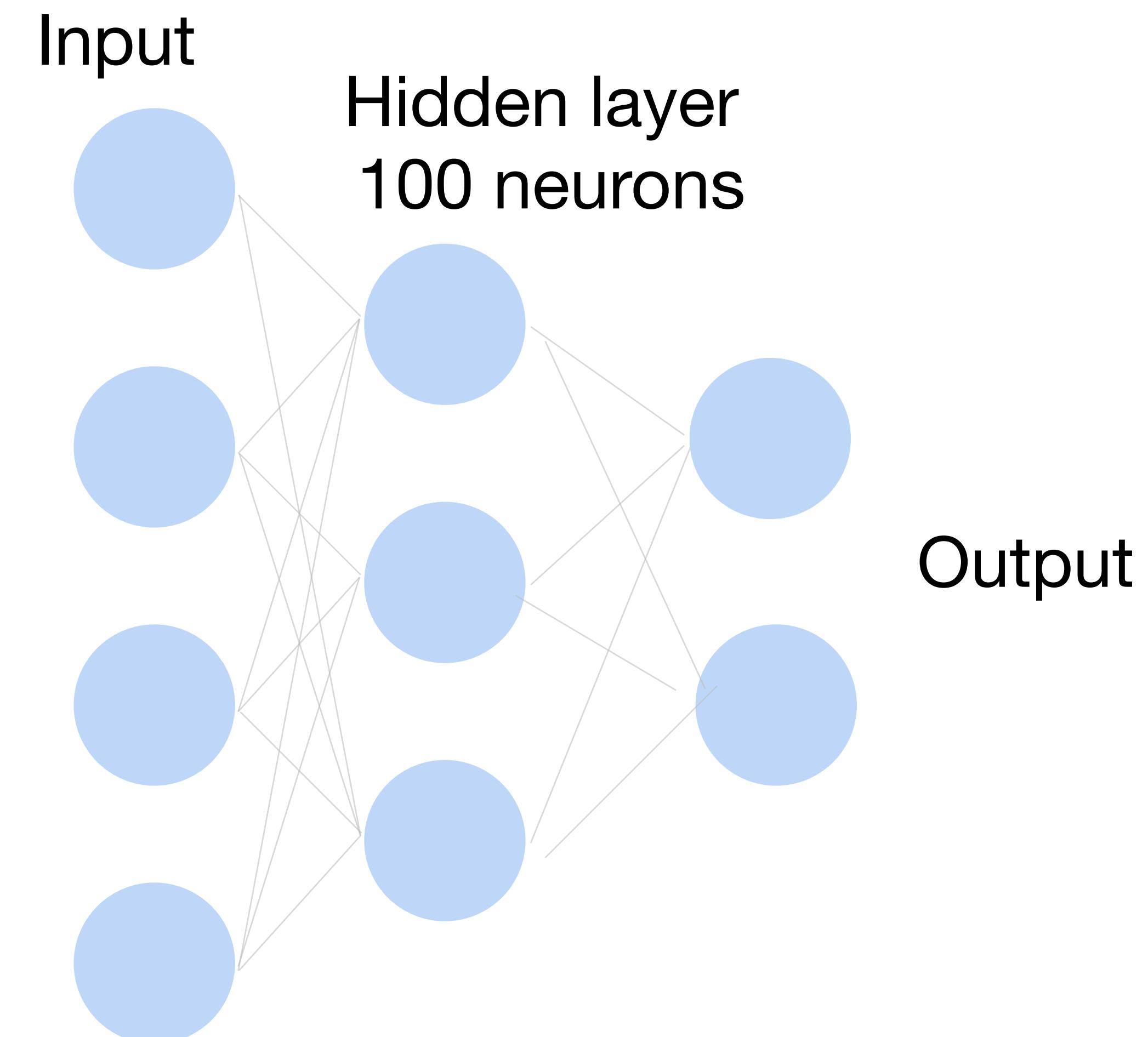
Loss function: $\frac{1}{|D|} \sum_i \ell(\mathbf{x}_i, y_i)$

Per-sample loss:

$$\ell(\mathbf{x}, y) = \sum_{j=1}^K -y_j \log p_j$$



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

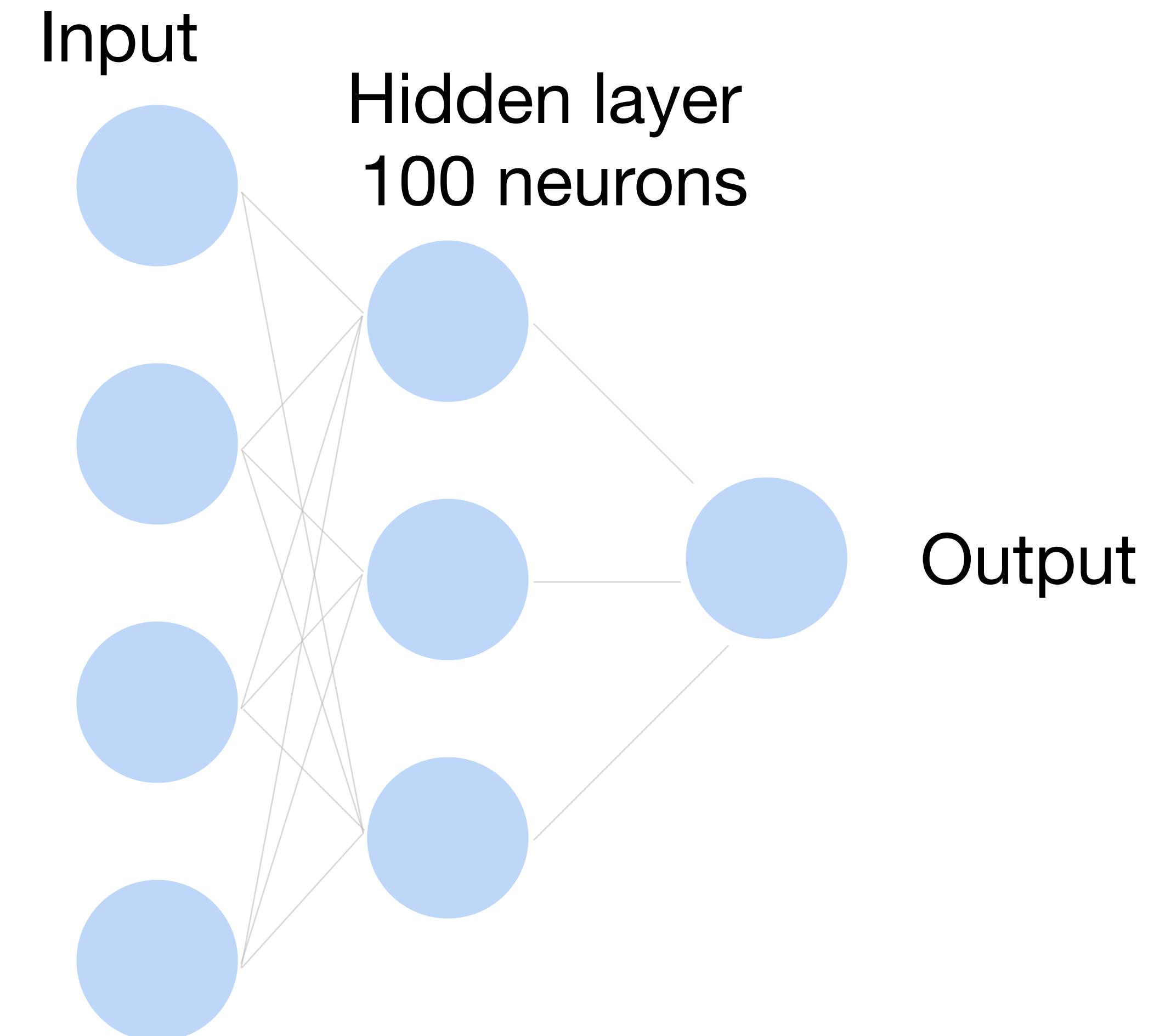


How to train a neural network?

Update the weights W to minimize the loss function

$$L = \frac{1}{|D|} \sum_i \ell(\mathbf{x}_i, y_i)$$

Use gradient descent!



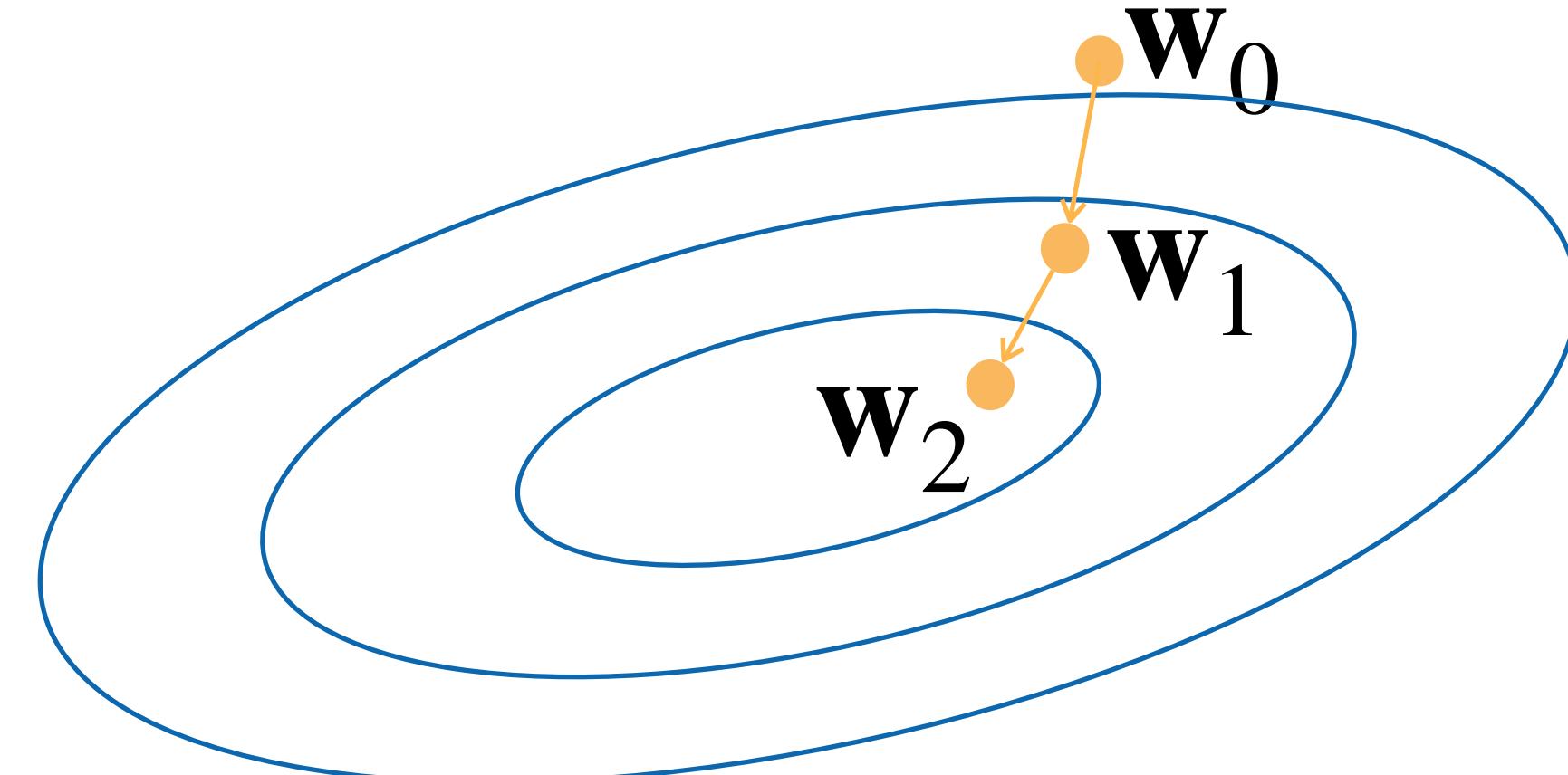
Gradient Descent

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

- Update parameters:

$$\begin{aligned} \mathbf{w}_t &= \mathbf{w}_{t-1} - \alpha \frac{\partial L}{\partial \mathbf{w}_{t-1}} \\ &= \mathbf{w}_{t-1} - \alpha \frac{1}{|D|} \sum_{\mathbf{x} \in D} \frac{\partial \ell(\mathbf{x}_i, y_i)}{\partial \mathbf{w}_{t-1}} \end{aligned}$$

- Repeat until converges

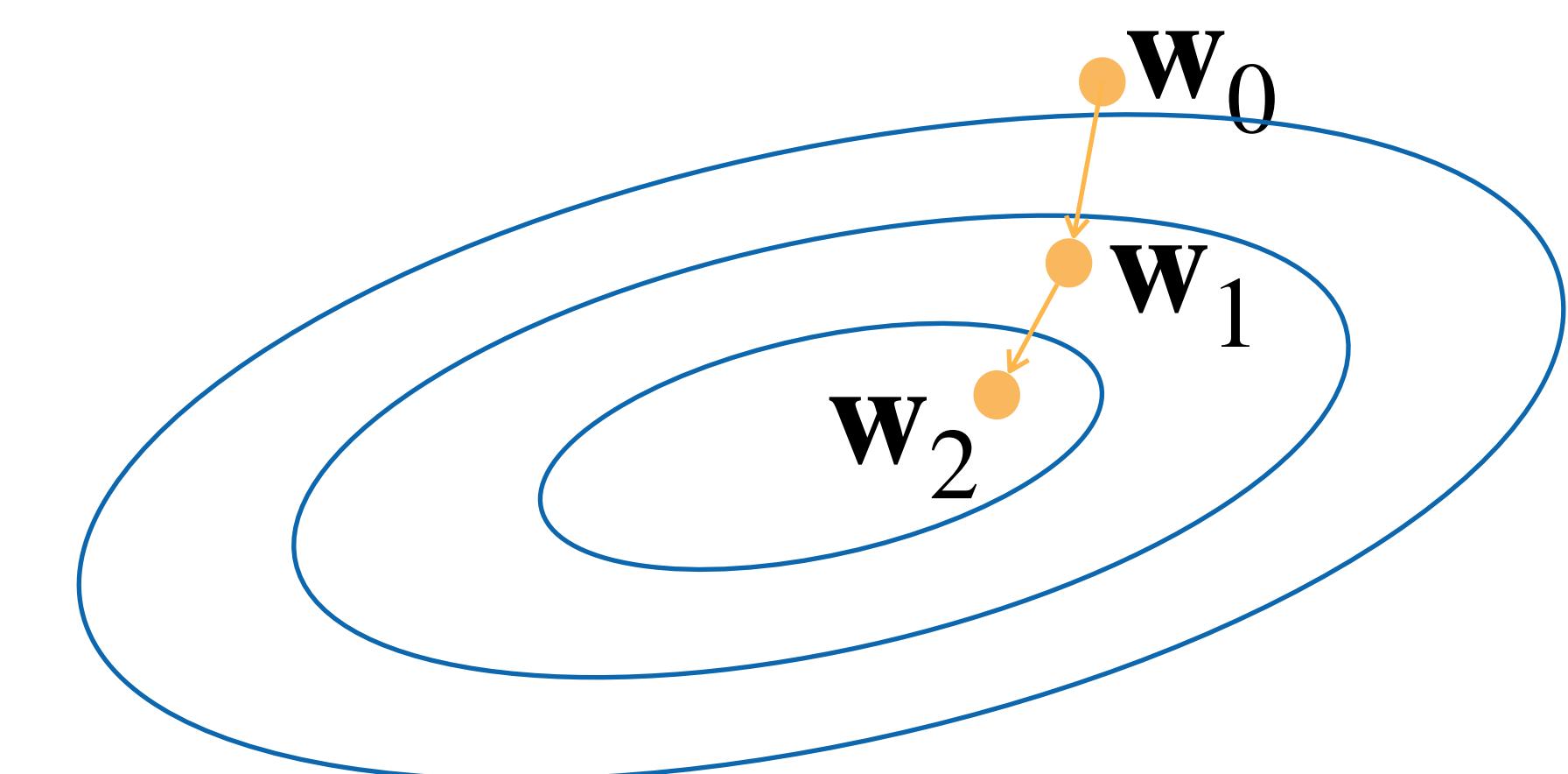


Gradient Descent

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

- Update parameters:

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



D can
be very large.
Expensive

- Repeat until converges

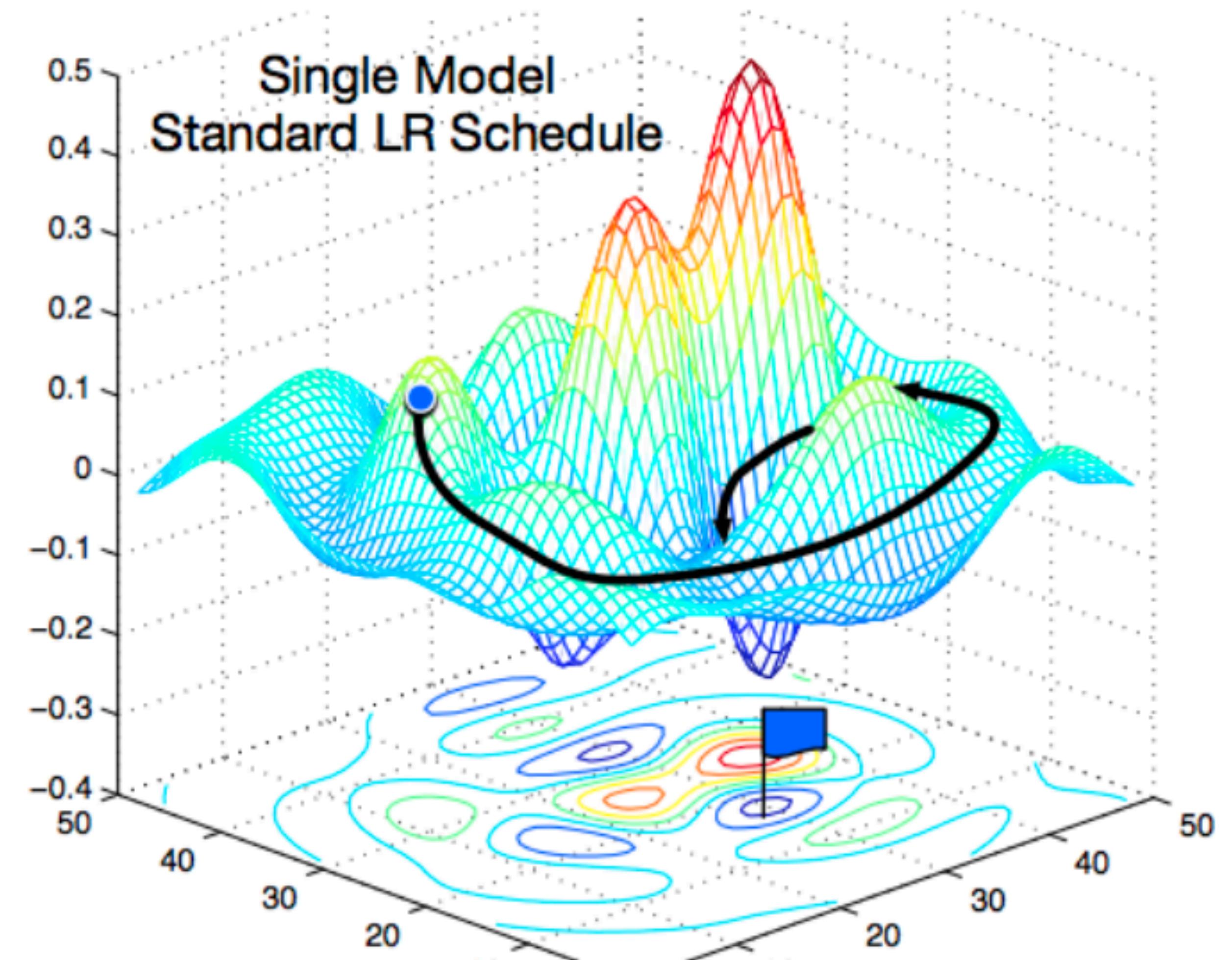
Minibatch Stochastic Gradient Descent

- Choose a learning rate $\alpha > 0$
- Initialize the model parameters w_0
- For $t = 1, 2, \dots$
 - **Randomly sample a subset (mini-batch) $\hat{D} \in D$**
Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \frac{1}{|\hat{D}|} \sum_{\mathbf{x} \in \hat{D}} \frac{\partial \ell(\mathbf{x}_i, y_i)}{\partial \mathbf{w}_{t-1}}$$

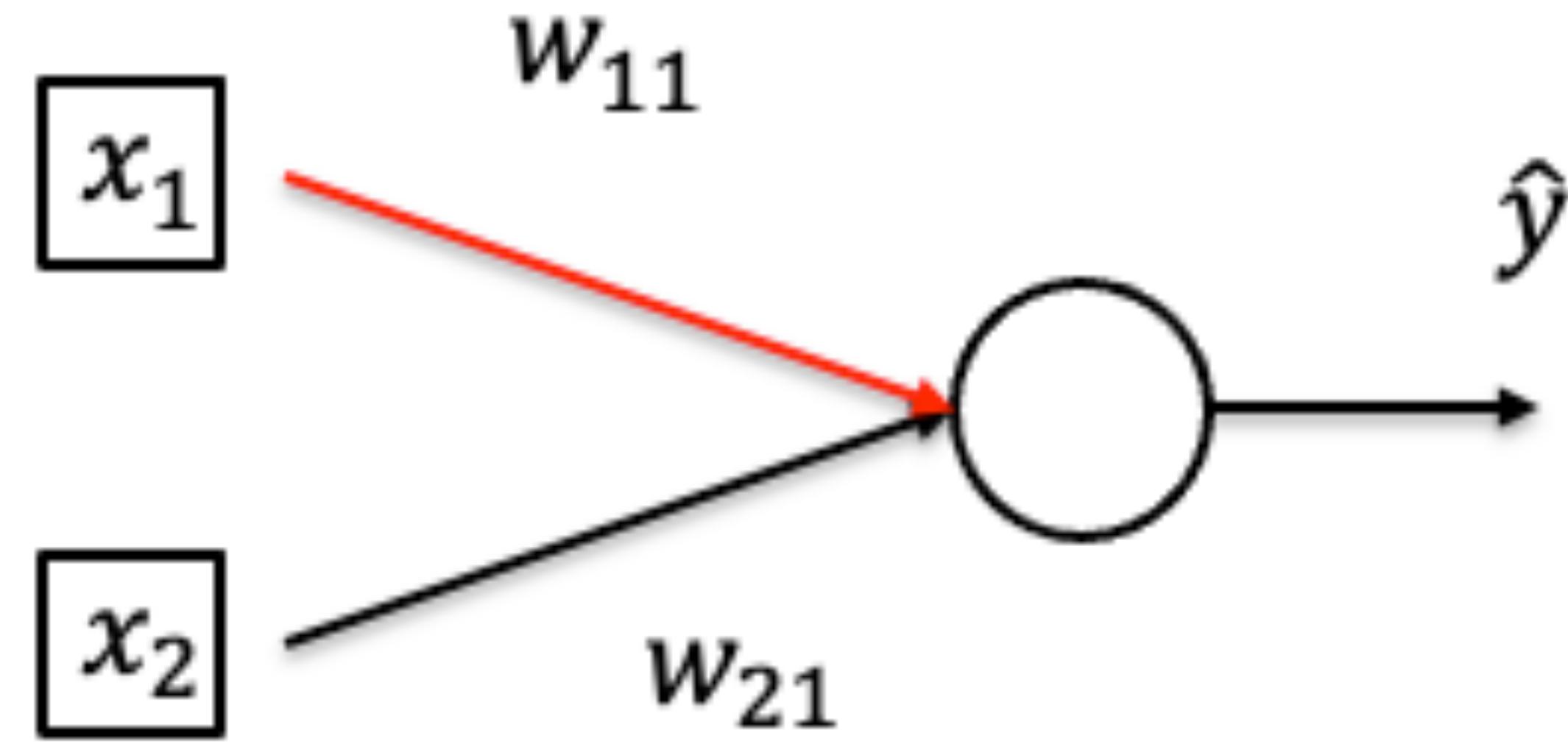
- Repeat until converges

Non-convex Optimization



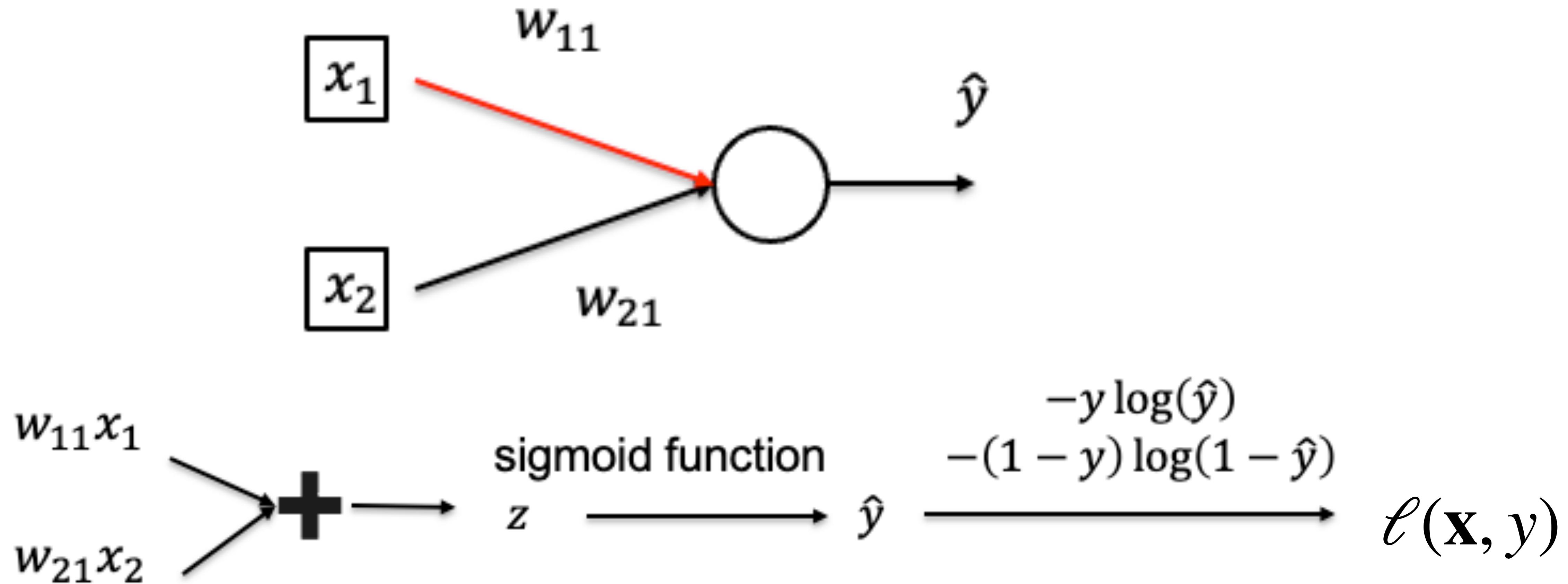
[Gao and Li et al., 2018]

Calculate Gradient (on one data point)

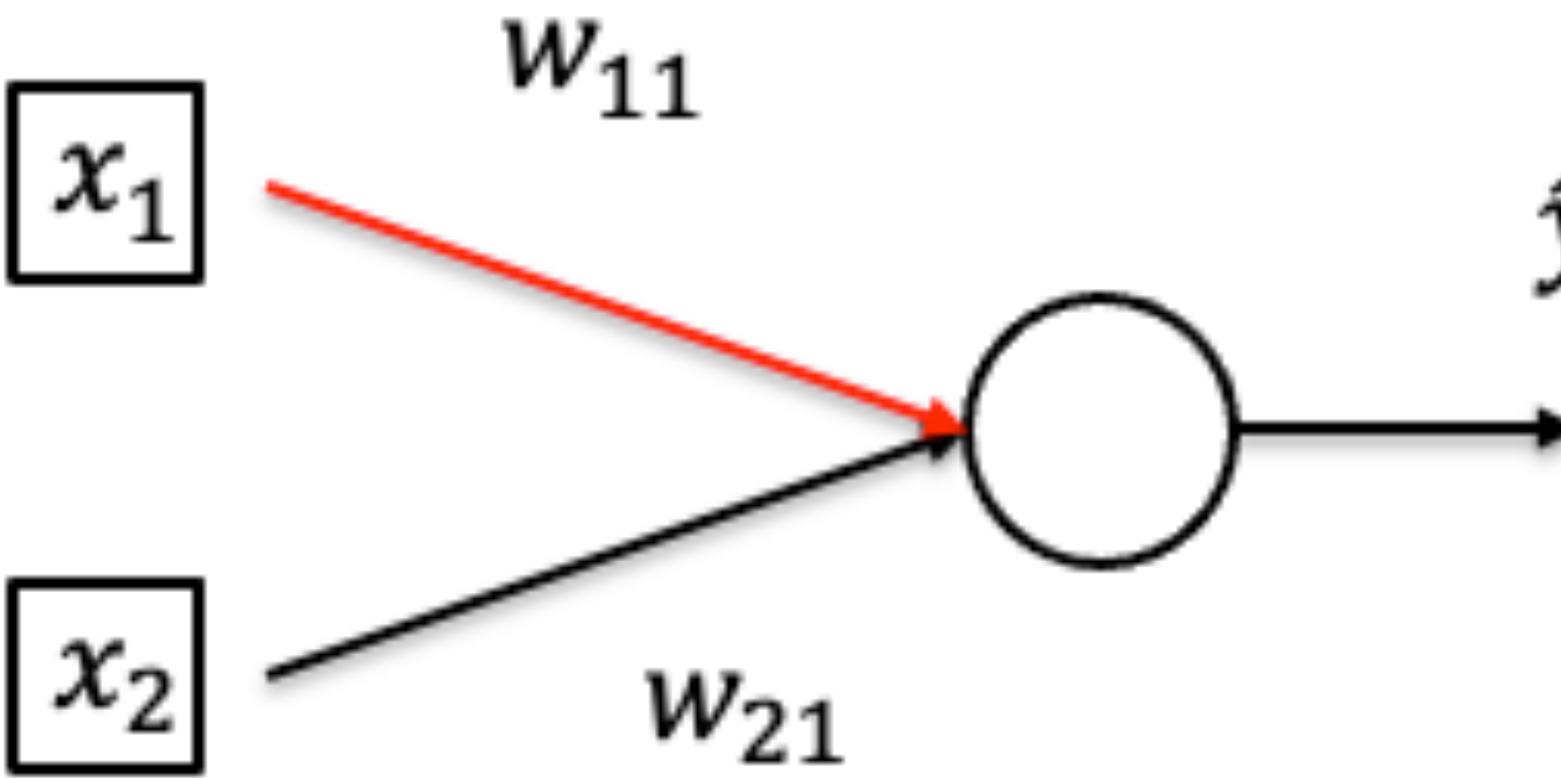


- Want to compute $\frac{\partial \ell(\mathbf{x}, y)}{\partial w_{11}}$

Calculate Gradient (on one data point)

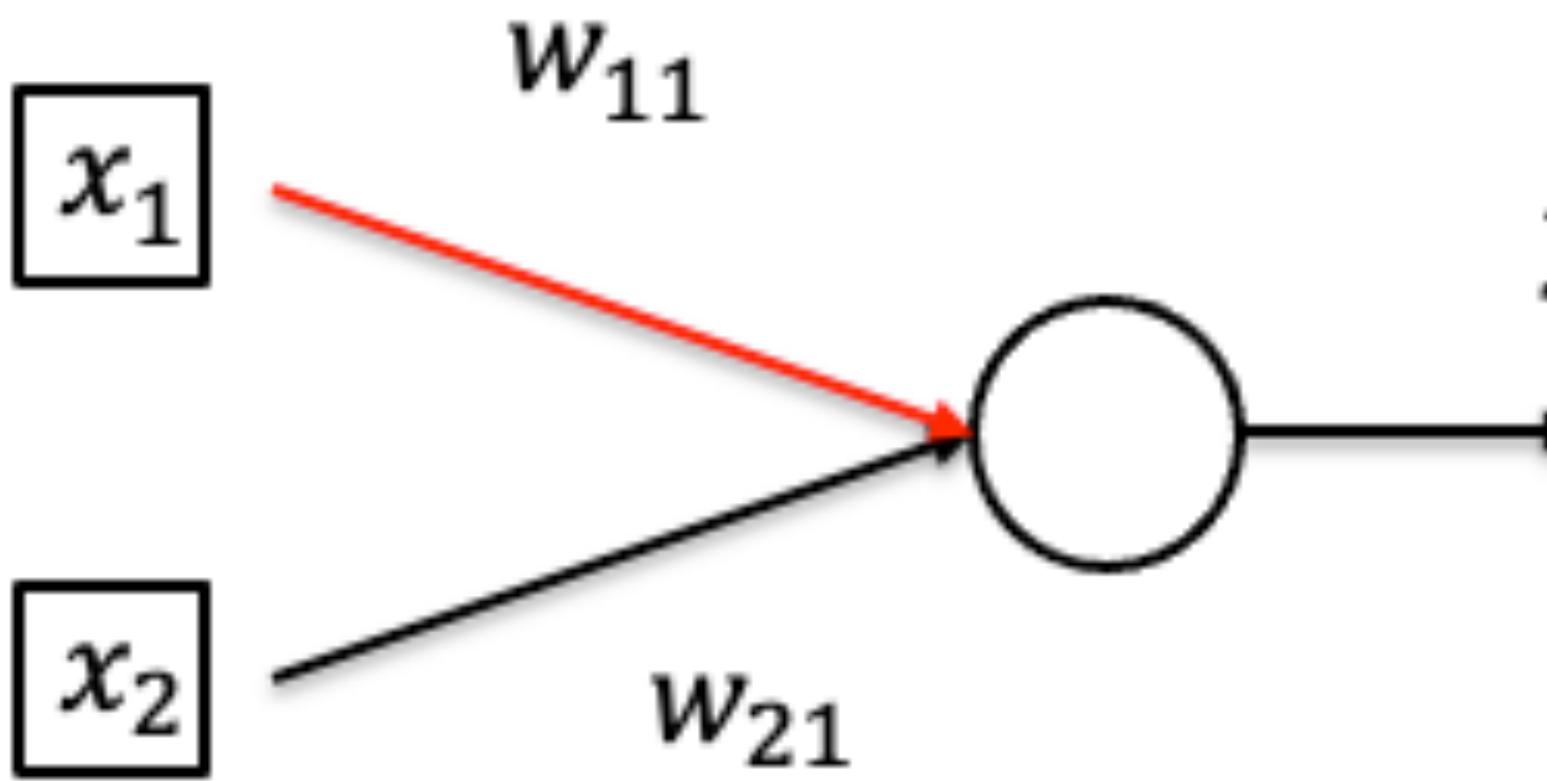


Calculate Gradient (on one data point)



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

Calculate Gradient (on one data point)

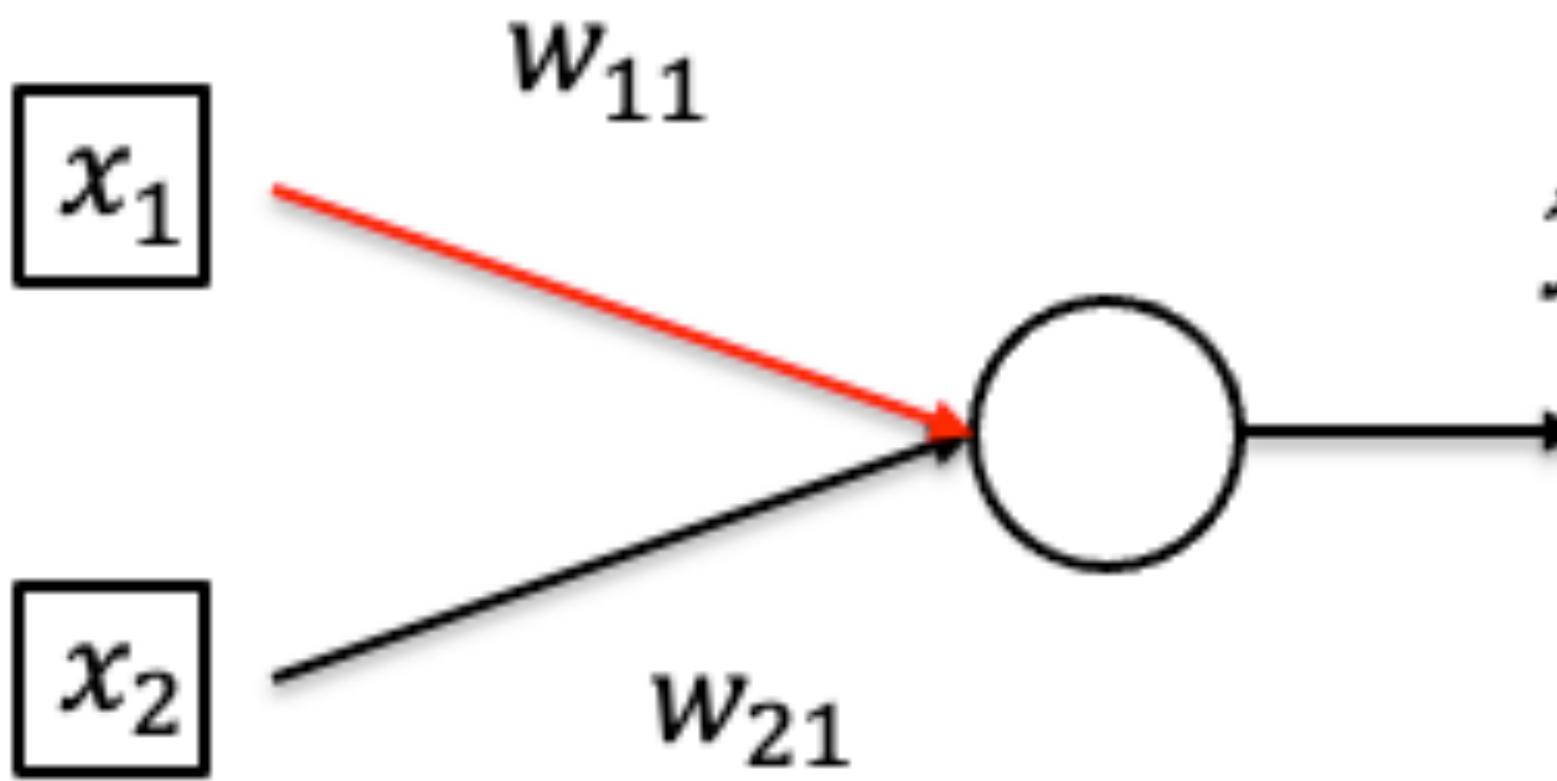


$$\begin{array}{ccccc} w_{11}x_1 & \xrightarrow{\text{+}} & z & \xrightarrow{\text{sigmoid function}} & \hat{y} \\ w_{21}x_2 & & & & \xrightarrow{-y \log(\hat{y})} \\ \frac{\partial z}{\partial w_{11}} = x_1 & & \frac{\partial \hat{y}}{\partial z} = \sigma'(z) & & \frac{\partial \ell(\mathbf{x}, y)}{\partial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

- 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)

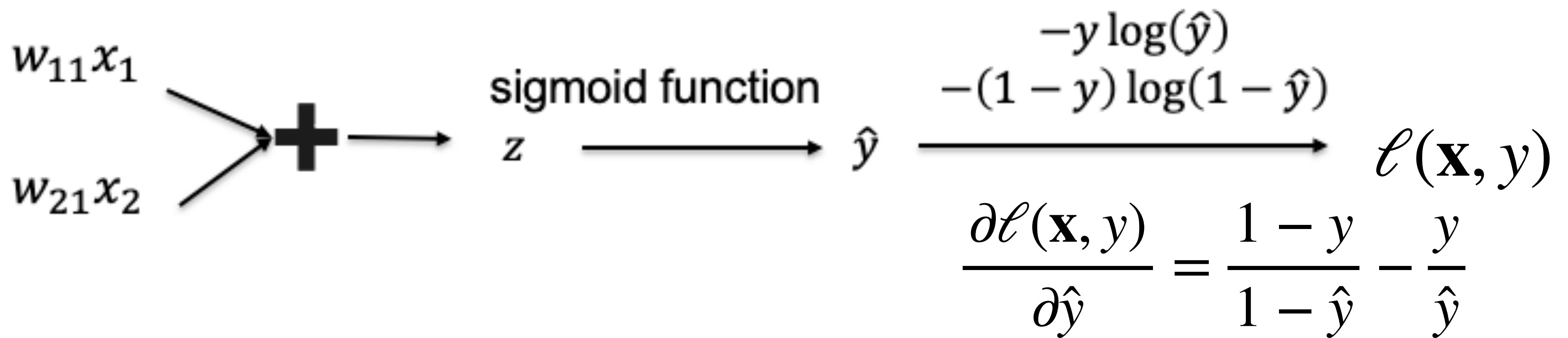
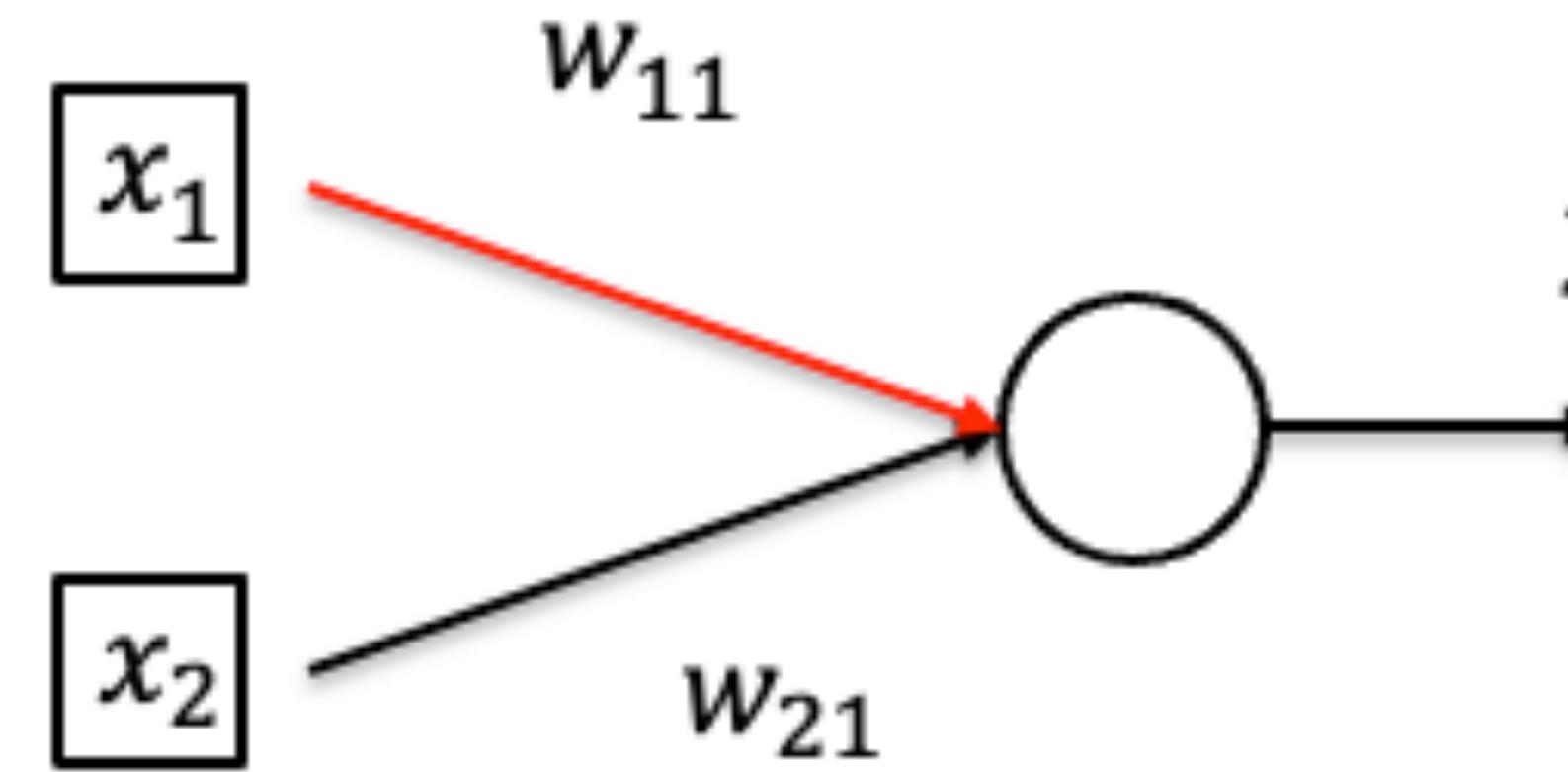


$$\begin{array}{c} w_{11}x_1 \\ w_{21}x_2 \end{array} \rightarrow \begin{array}{c} + \\ \text{sigmoid function} \end{array} \rightarrow z \rightarrow \hat{y} \rightarrow \ell(\mathbf{x}, y)$$
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z)) \quad \frac{-y \log(\hat{y})}{-(1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

- By chain rule:

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

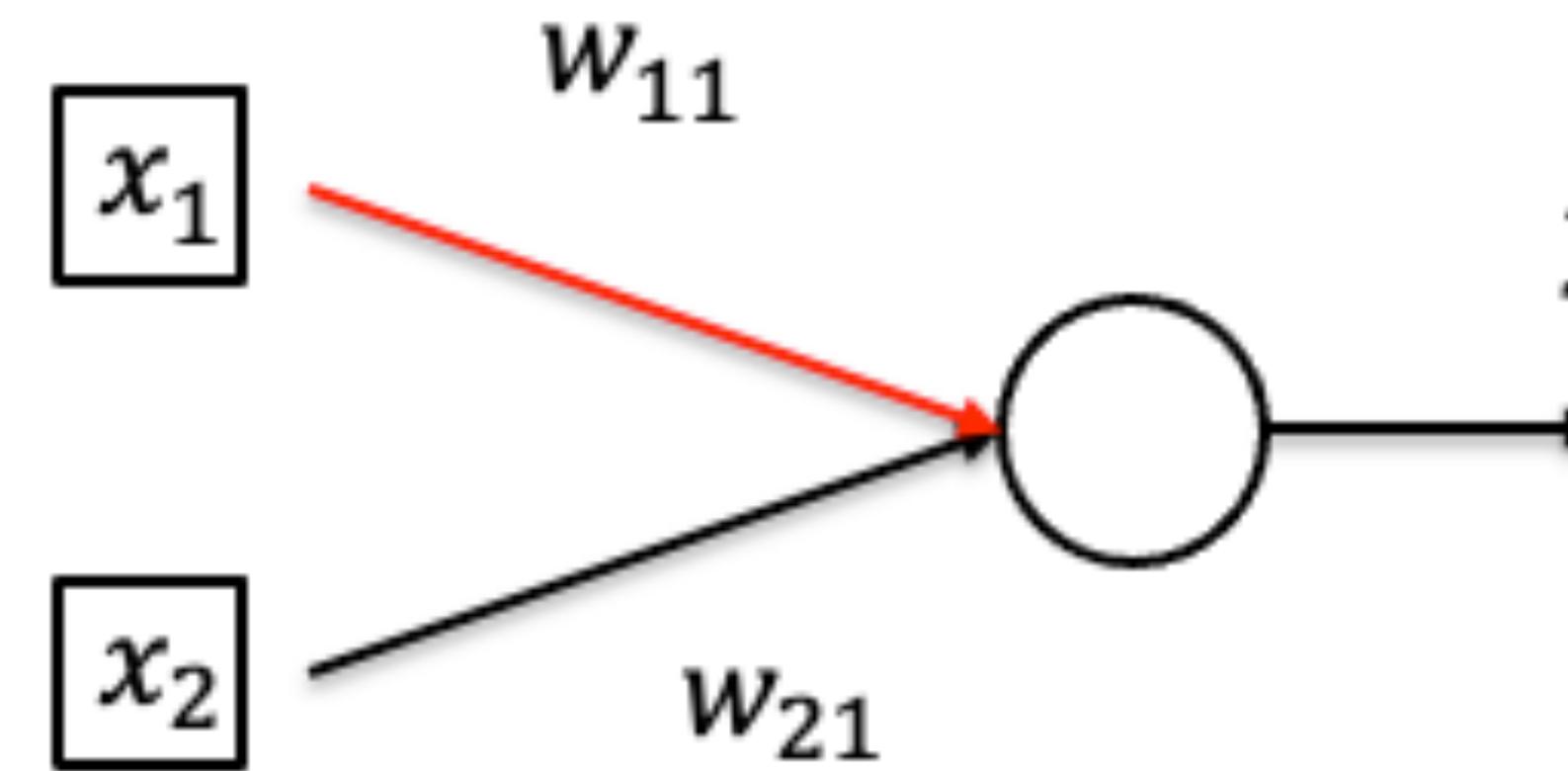
Calculate Gradient (on one data point)



- By chain rule:

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

Calculate Gradient (on one data point)



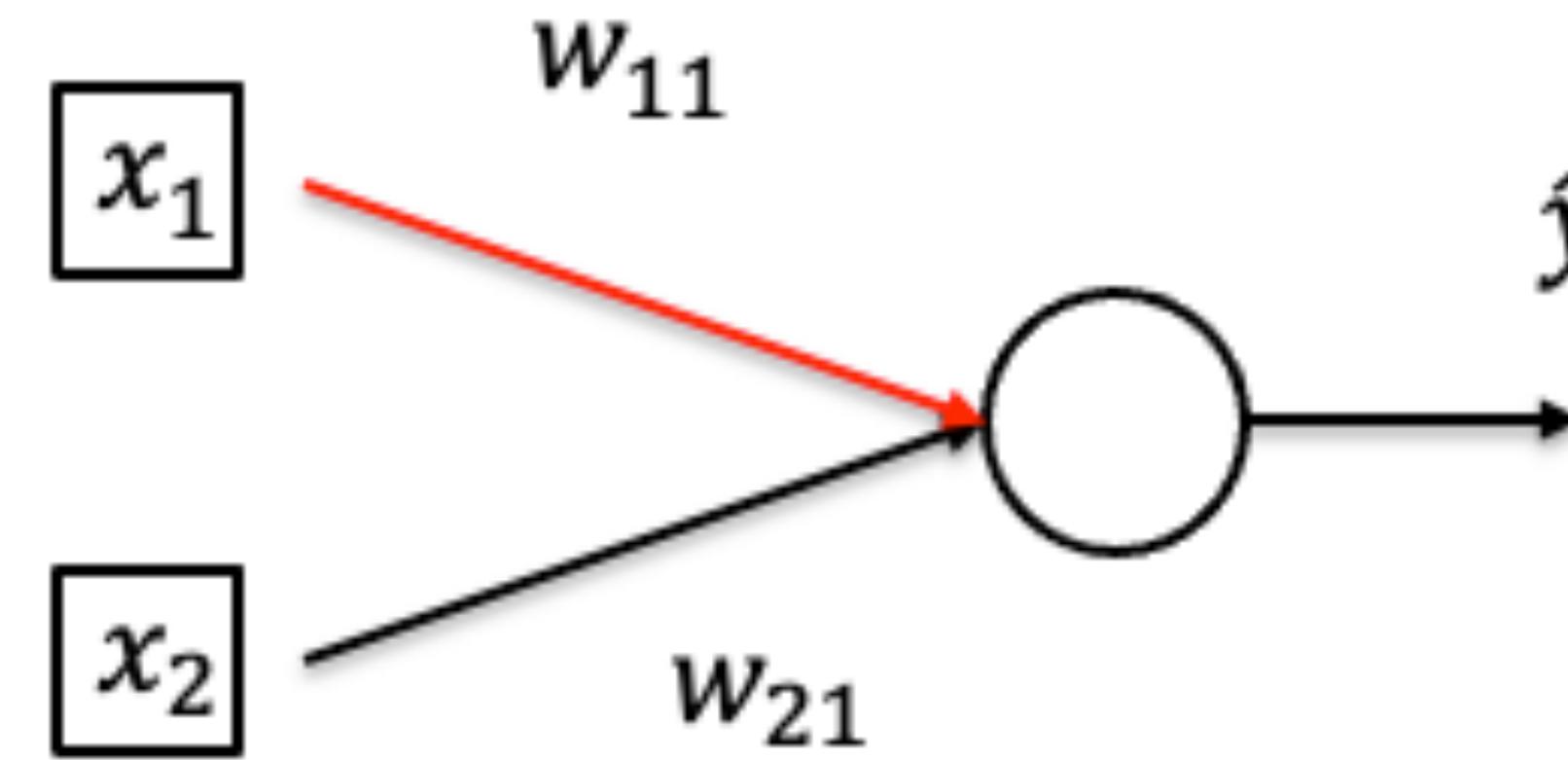
The diagram illustrates the calculation of the loss function $\ell(\mathbf{x}, y)$ from the inputs x_1 and x_2 . The inputs are multiplied by their respective weights w_{11} and w_{21} , then summed. This sum is passed through a sigmoid function to produce the predicted output \hat{y} . The loss function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



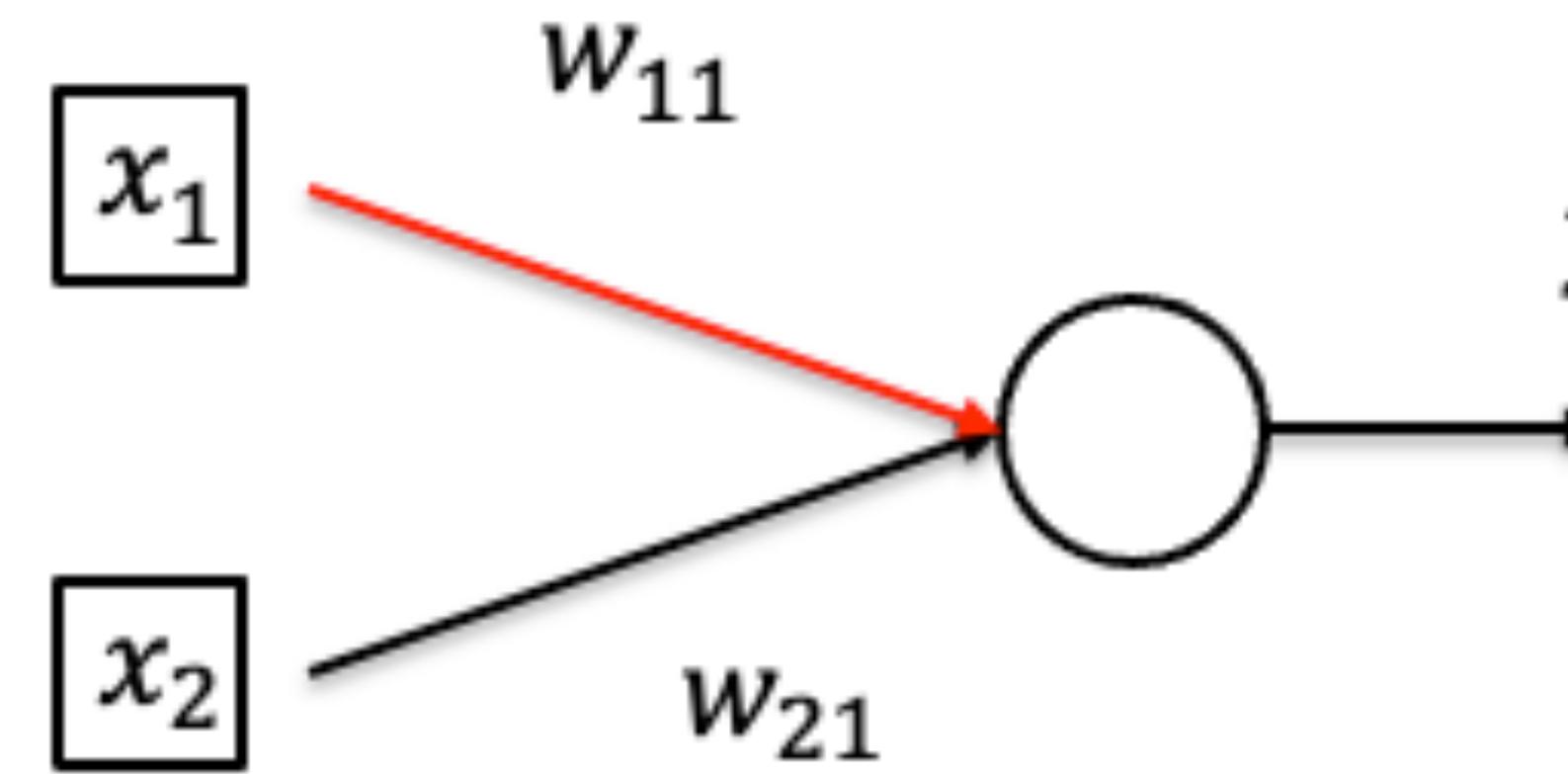
A computational graph illustrating the forward pass and loss calculation:

- 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 .

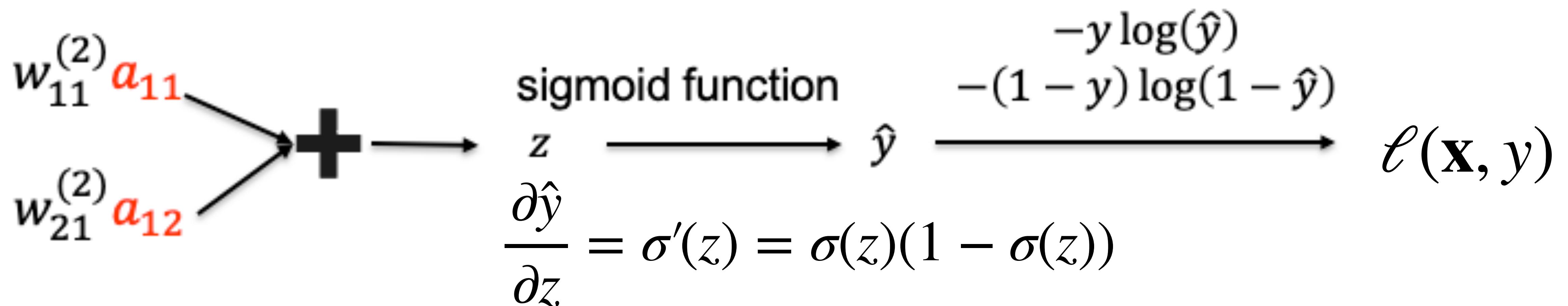
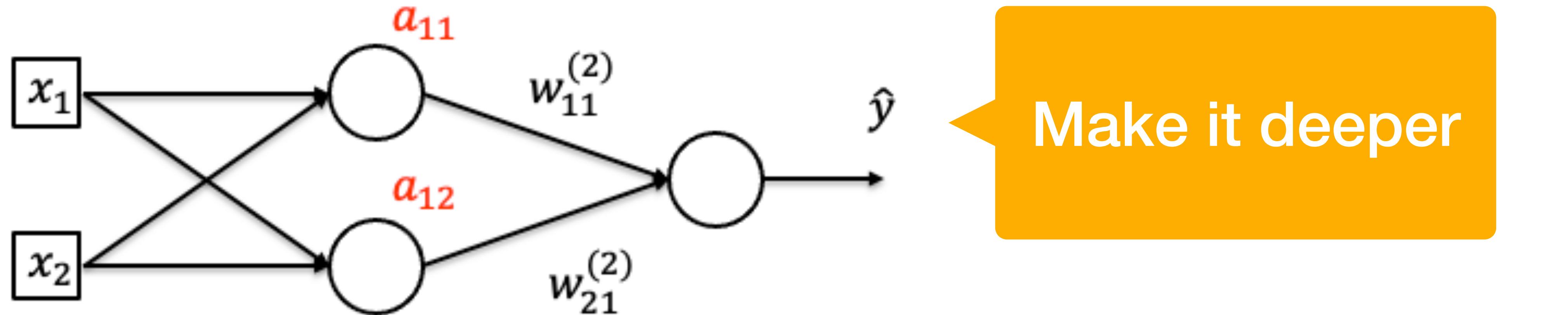
Below the graph, the gradient of the loss function with respect to the input x_1 is derived using the chain rule:

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

- By chain rule:

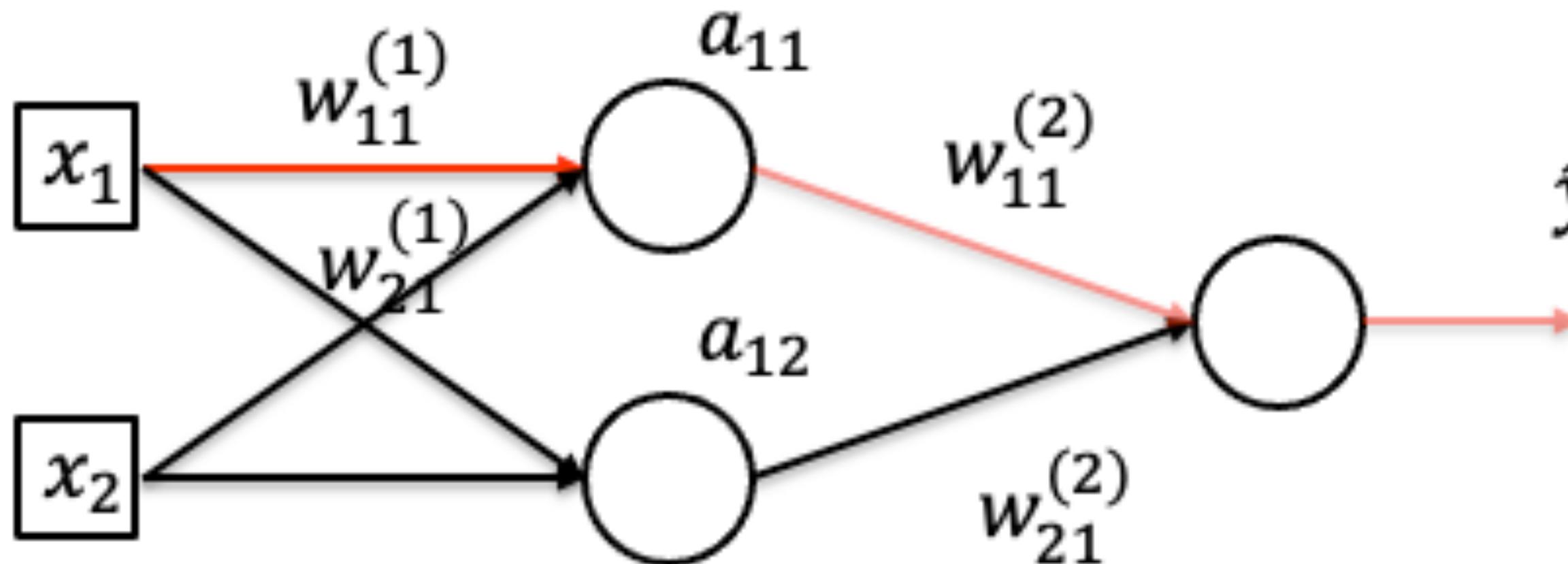
$$\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} w_{11} = (\hat{y} - y)w_{11}$$

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)

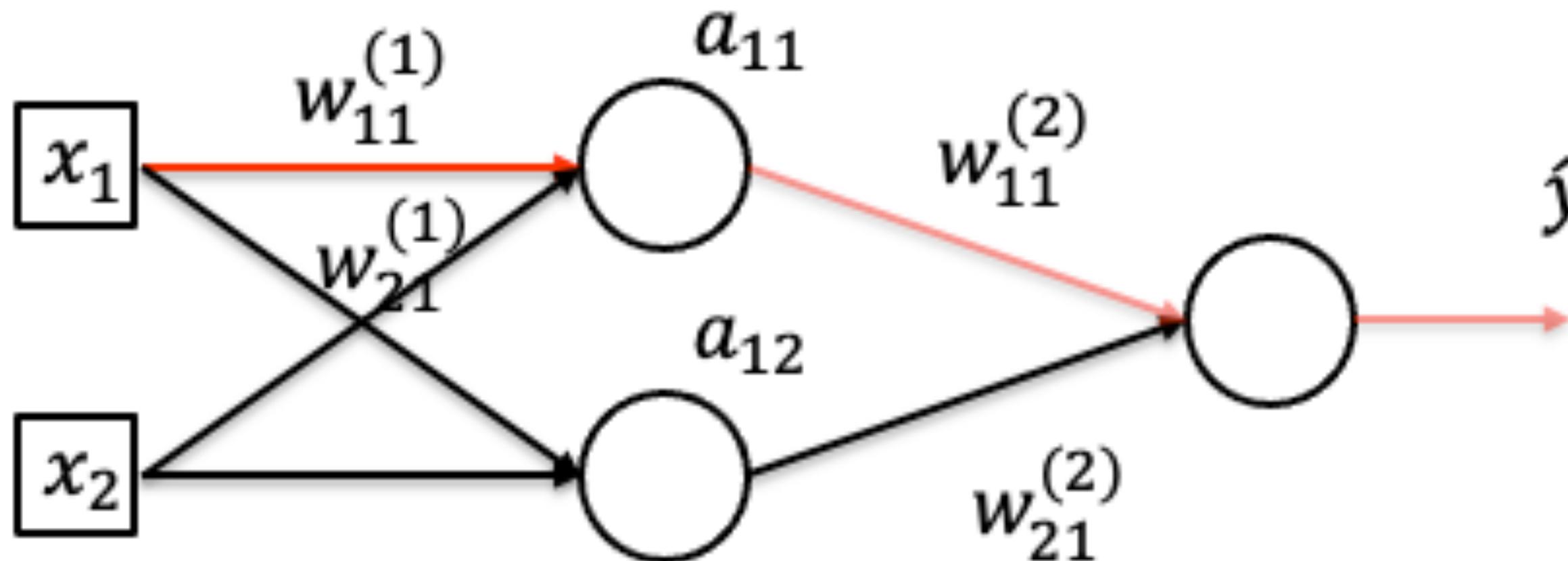


A computational graph illustrating the backpropagation process. The graph shows the flow of values and gradients:

- The input values $w_{11}^{(1)}x_1$ and $w_{21}^{(1)}x_2$ are summed at a black addition node to produce the pre-activation value z_{11} .
- The activation function $\sigma(z_{11})$ is applied to z_{11} to produce the hidden unit output a_{11} .
- The error gradient $\frac{\partial l}{\partial a_{11}}$ is calculated using the chain rule: $\frac{\partial l}{\partial a_{11}} = \sigma'(z_{11}) \cdot \frac{\partial l}{\partial z_{11}}$.
- The final error gradient is $\frac{\partial l}{\partial w_{11}^{(1)}} = (\hat{y} - y)w_{11}^{(2)} \frac{\partial a_{11}}{\partial w_{11}^{(1)}}$.

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

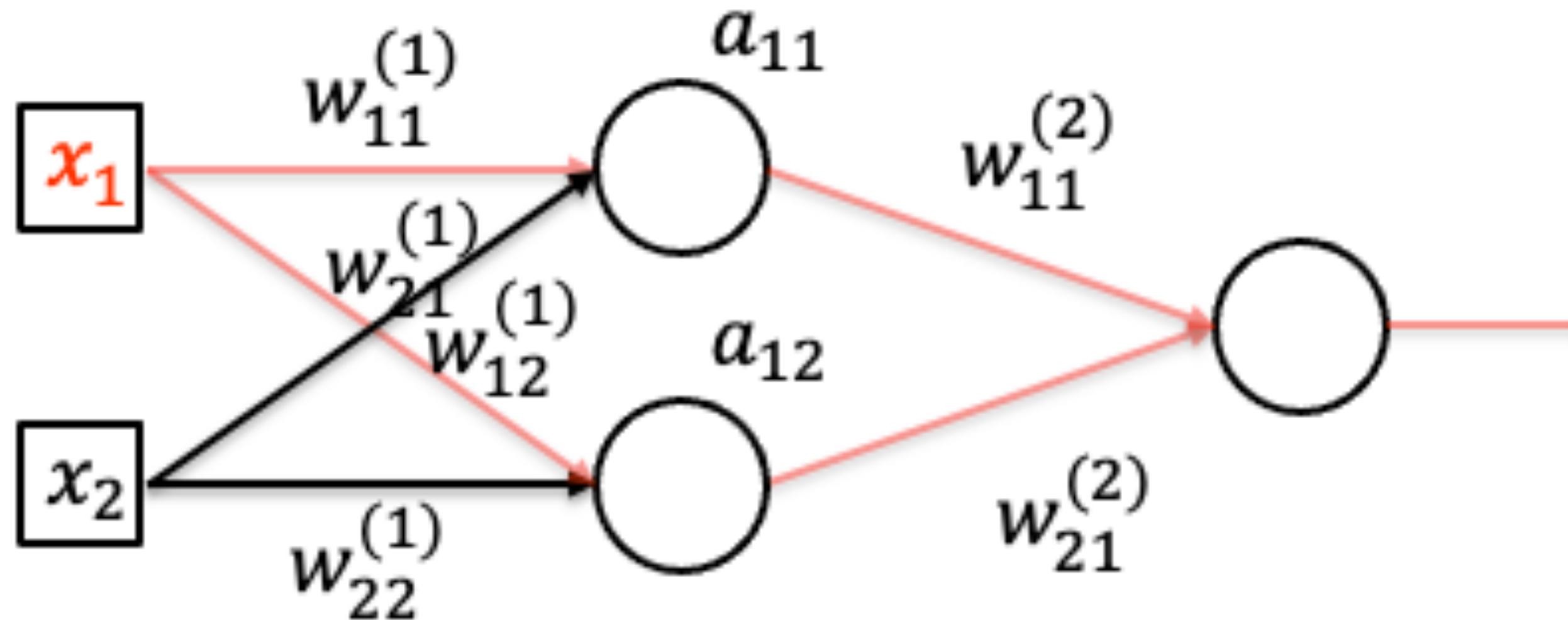
Calculate Gradient (on one data point)



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

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

Calculate Gradient (on one data point)

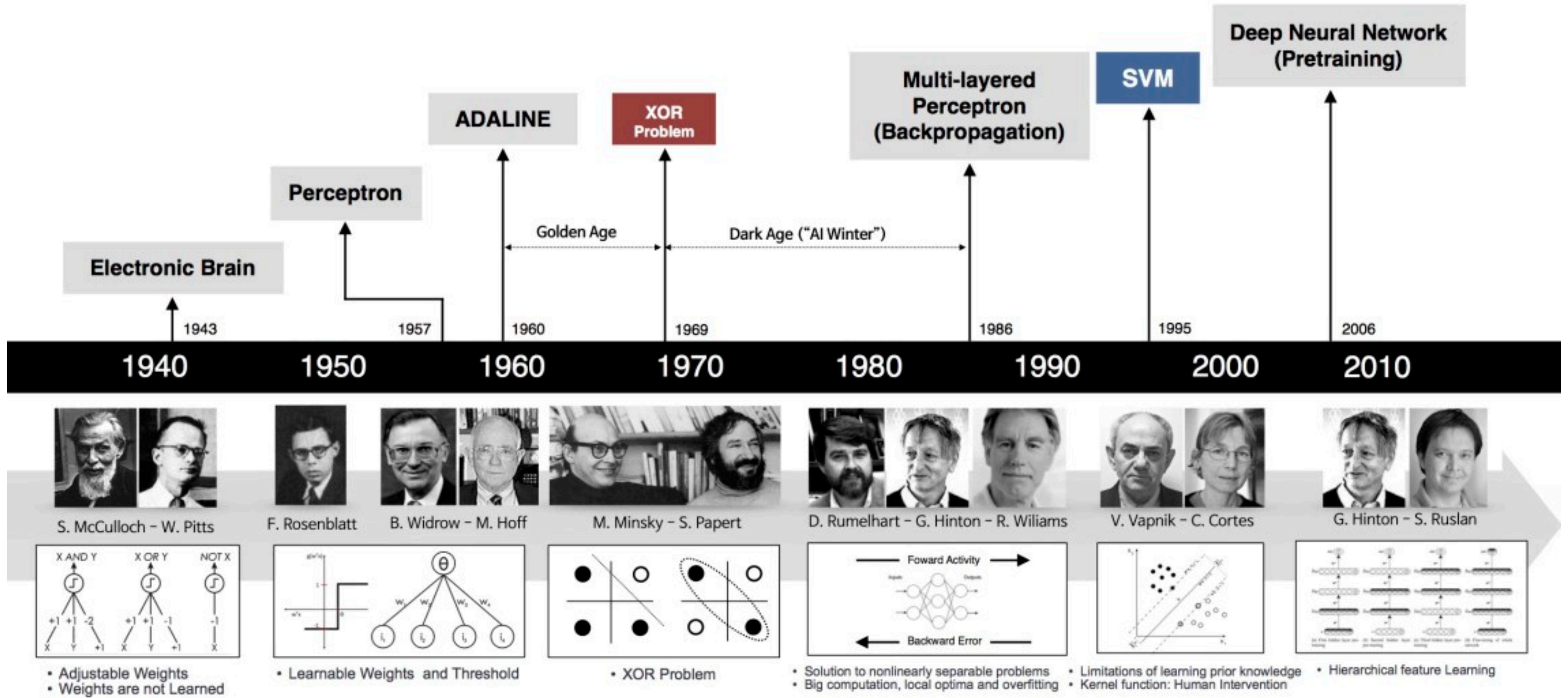


$$\begin{aligned} w_{11}^{(1)} x_1 & \quad \quad \quad + \quad \quad \quad \sigma(z_{11}) \quad \quad \quad l(\mathbf{x}, y) \\ w_{21}^{(1)} x_2 & \end{aligned}$$
$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \quad \quad \quad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y) w_{11}^{(2)}$$

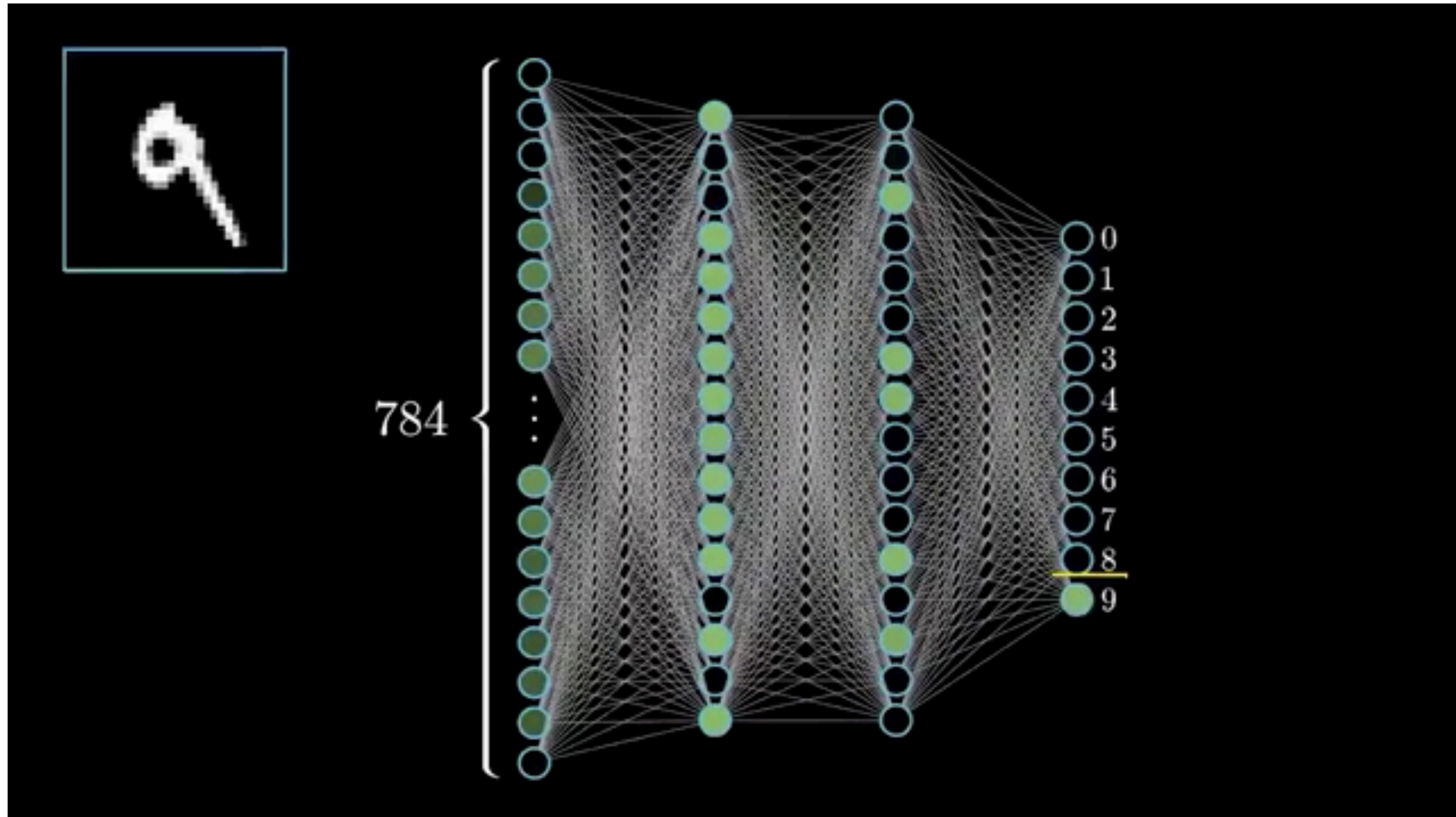
- By chain rule:

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

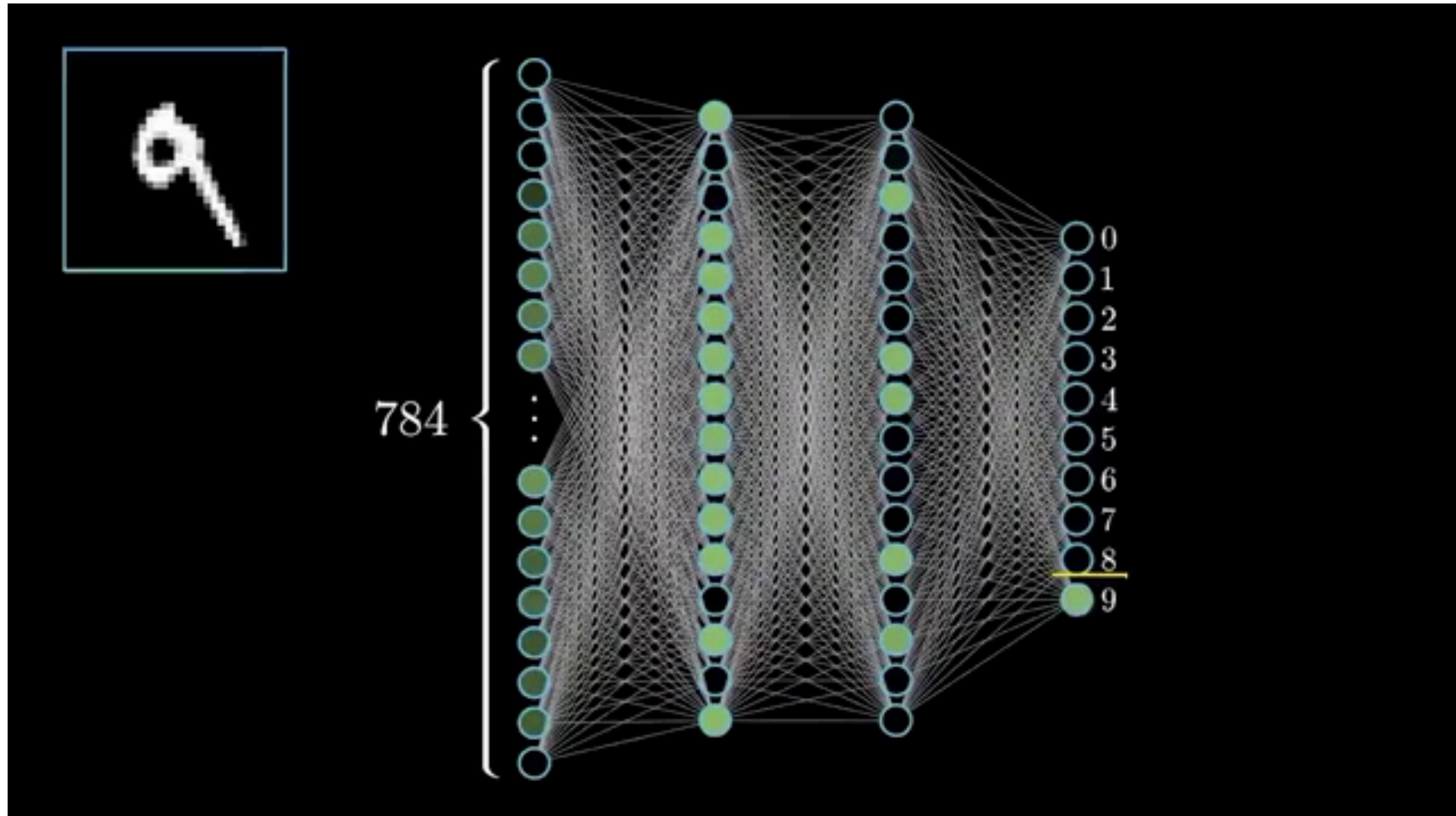
Brief history of neural networks



HW6



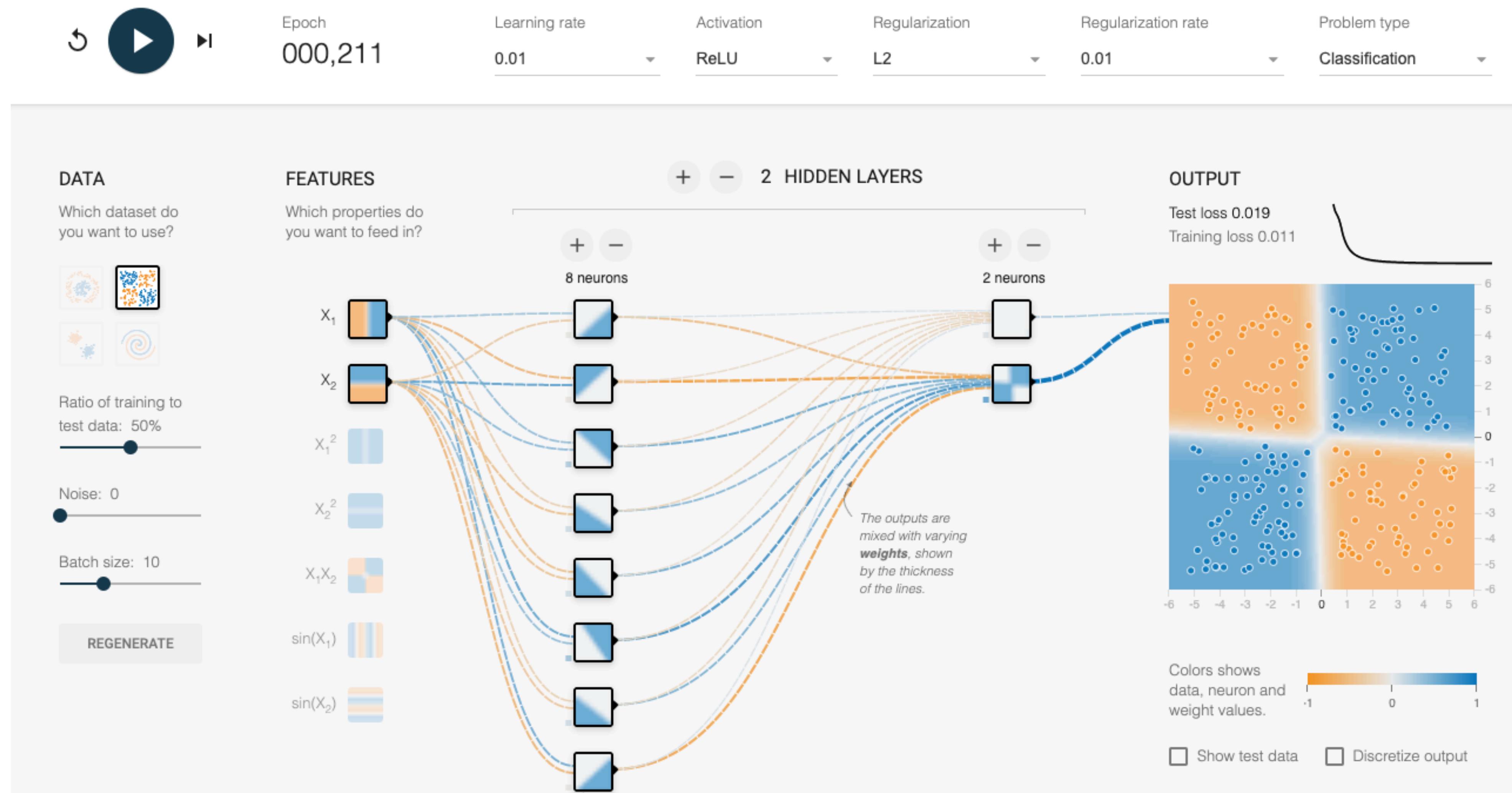
HW6



HW6 (working with MNIST dataset)



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



Thanks!