



CS 540 Introduction to Artificial Intelligence

Neural Networks II

University of Wisconsin-Madison
Spring 2026 Sections 1 & 2

Announcements

- **Homework:**

- HW5 due **Friday Oct 16 at 11:59 PM**
- No homework next week due to midterm.

- Midterm course evaluations due tomorrow

- Class roadmap:

Neural Networks II
Neural Networks III
Deep Learning I

Machine
Learning

Midterm Information

- . **Time: Oct 23rd 5:45-7:15 PM**
- . **Location:** Humanities Building Room 2340
- . **Format:** MCQ
- . **Cheat sheet:** a handwritten single piece of paper, front and back
- . **Calculator:** optional, if it doesn't have an Internet connection
- . **Bring:** your WISC ID, pencil (No 2 or softer), your 1-sheet notes.

Today's outline

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

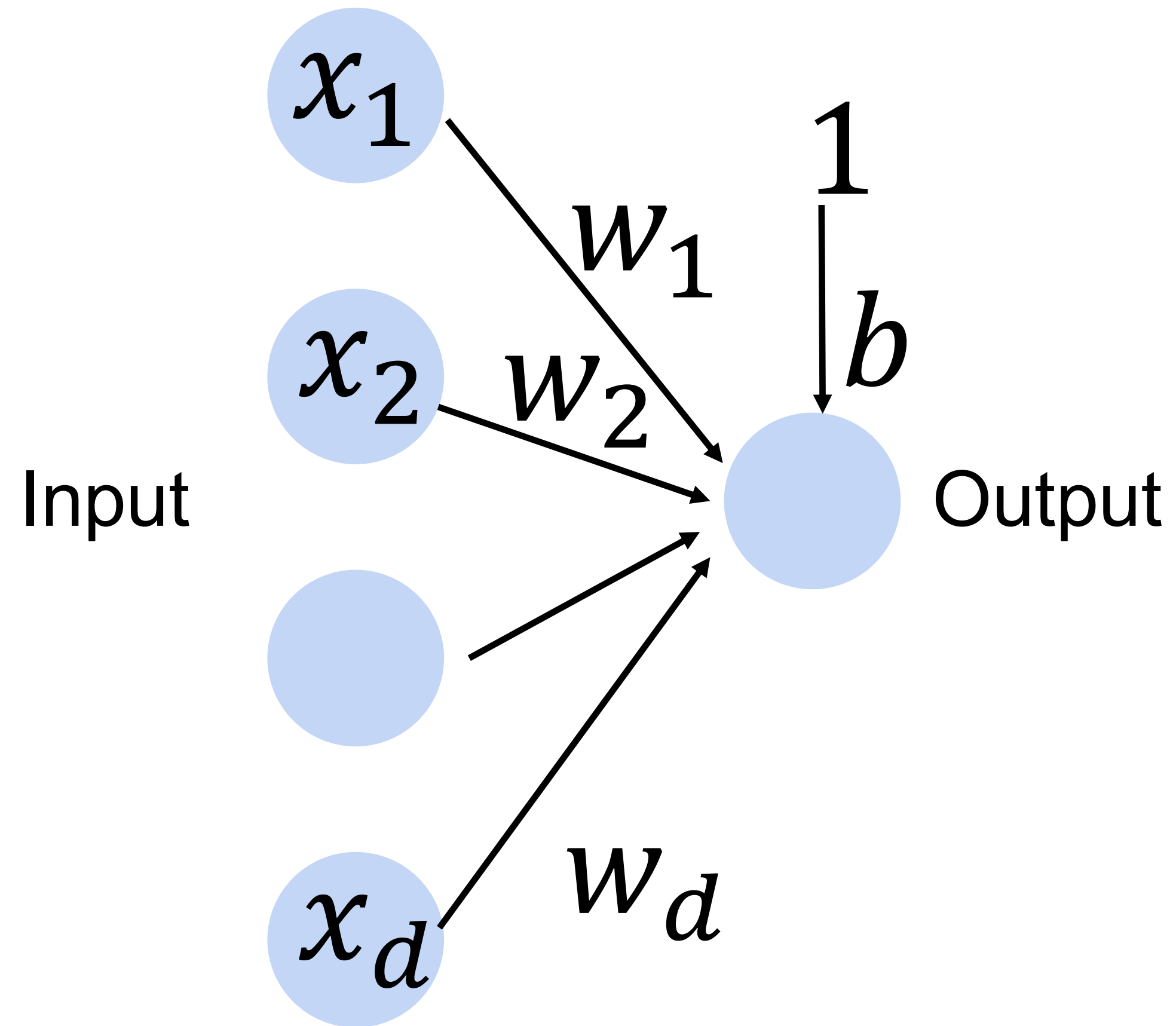
Review: The Perceptron

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

$$\hat{y} = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$
$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{otherwise} \end{cases}$$

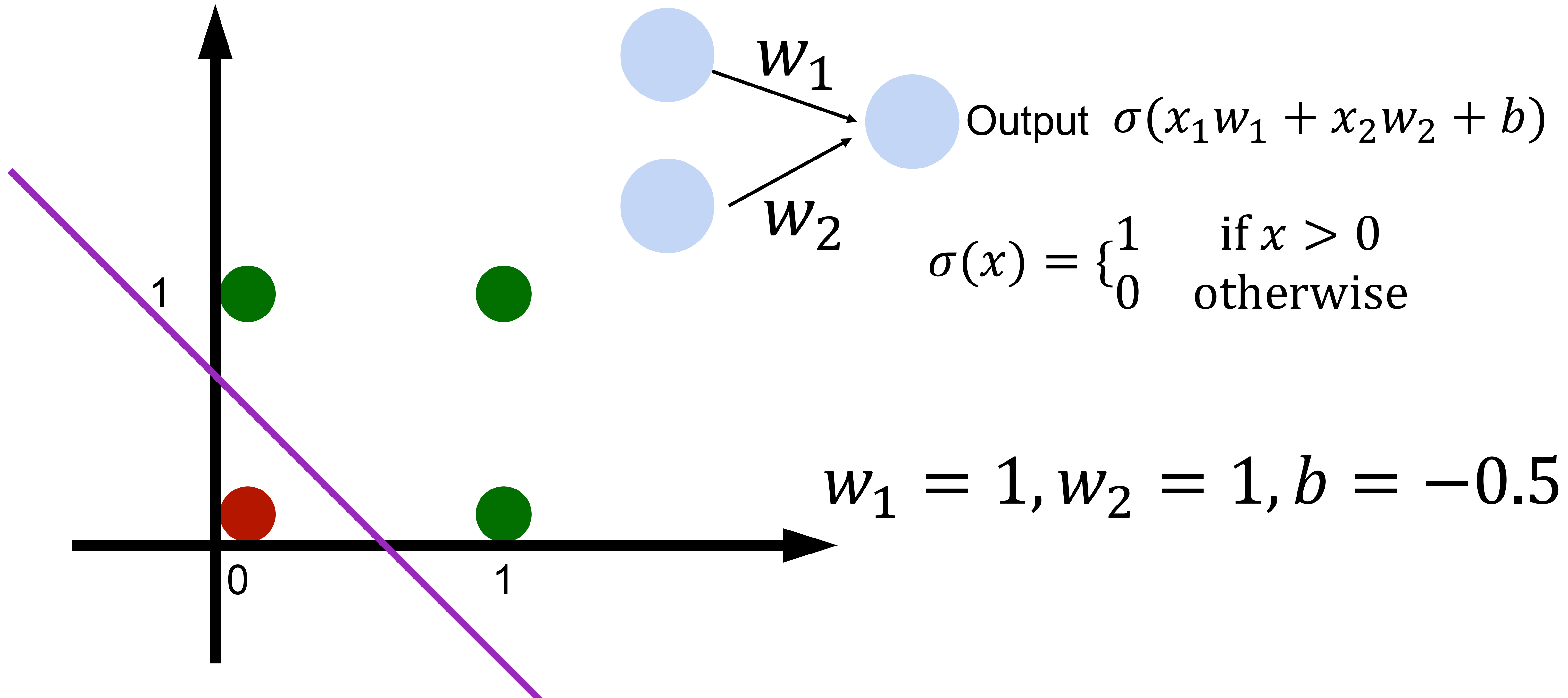
**activation
function**

Cats vs. dogs?



Learning OR function using perceptron

The perceptron can learn an OR function



Learning NOT function using perceptron

The perceptron can learn NOT function (single input)



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

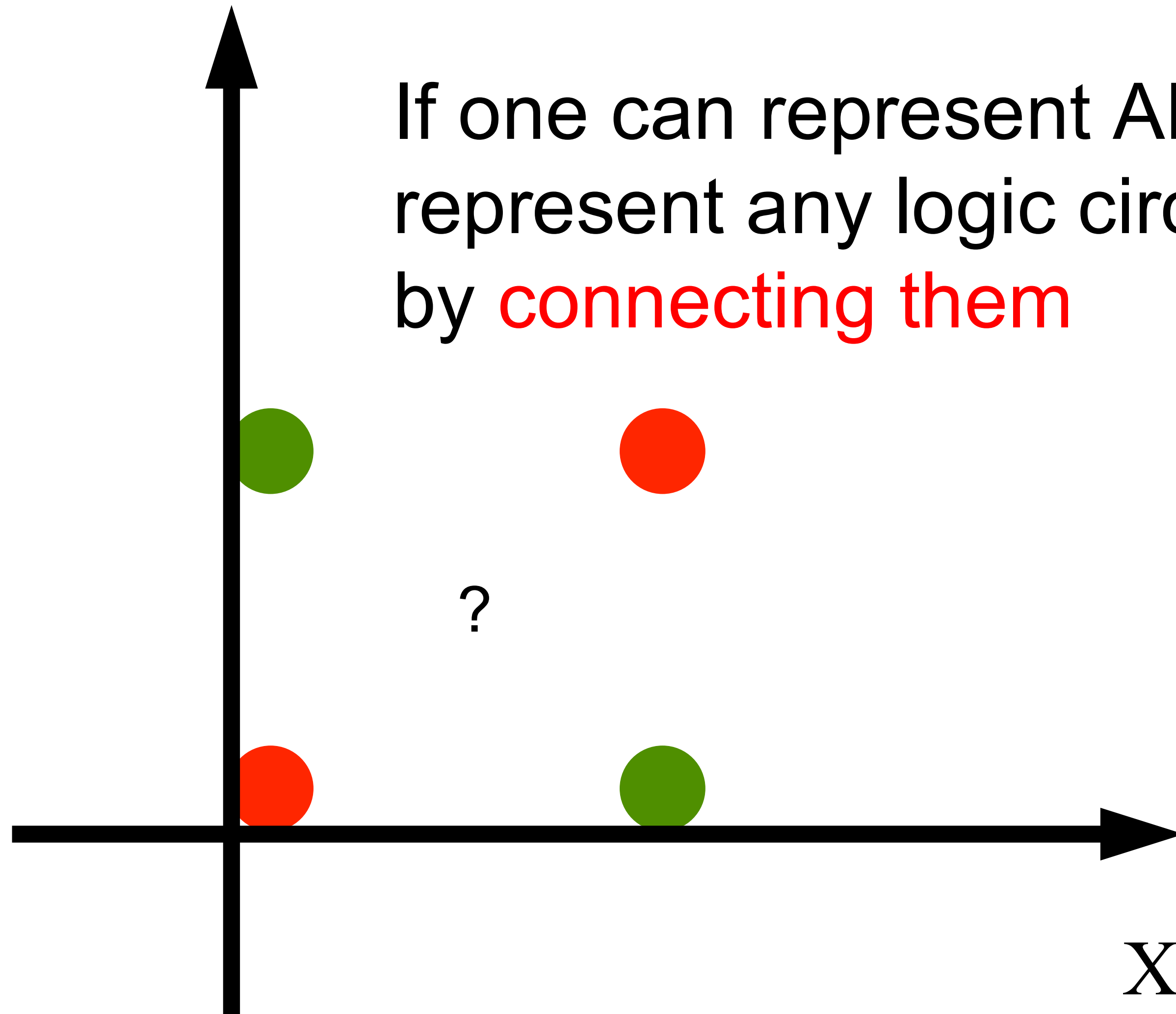


$$w_1 = -1, b = 0.5$$

The limited power of a single neuron

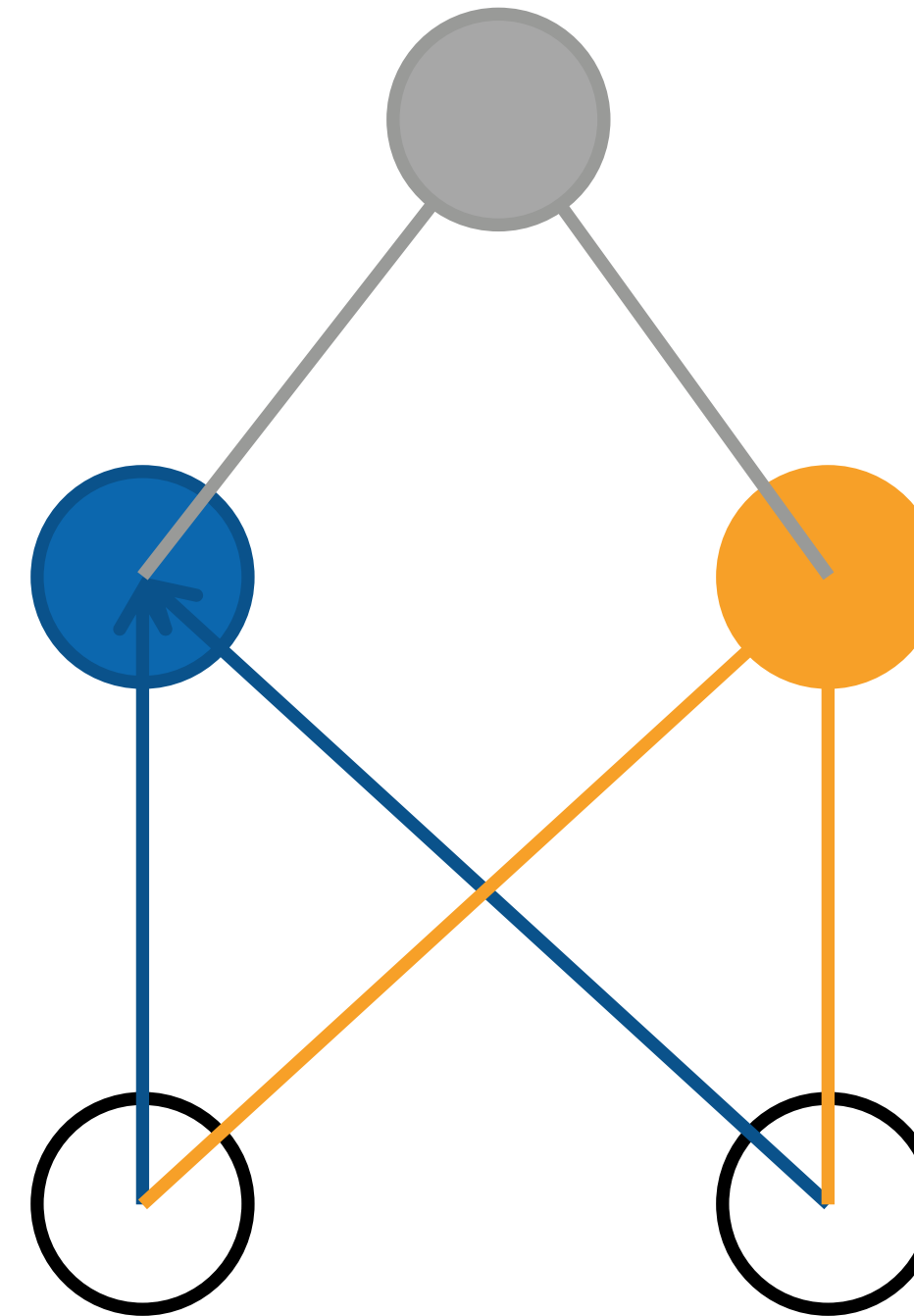
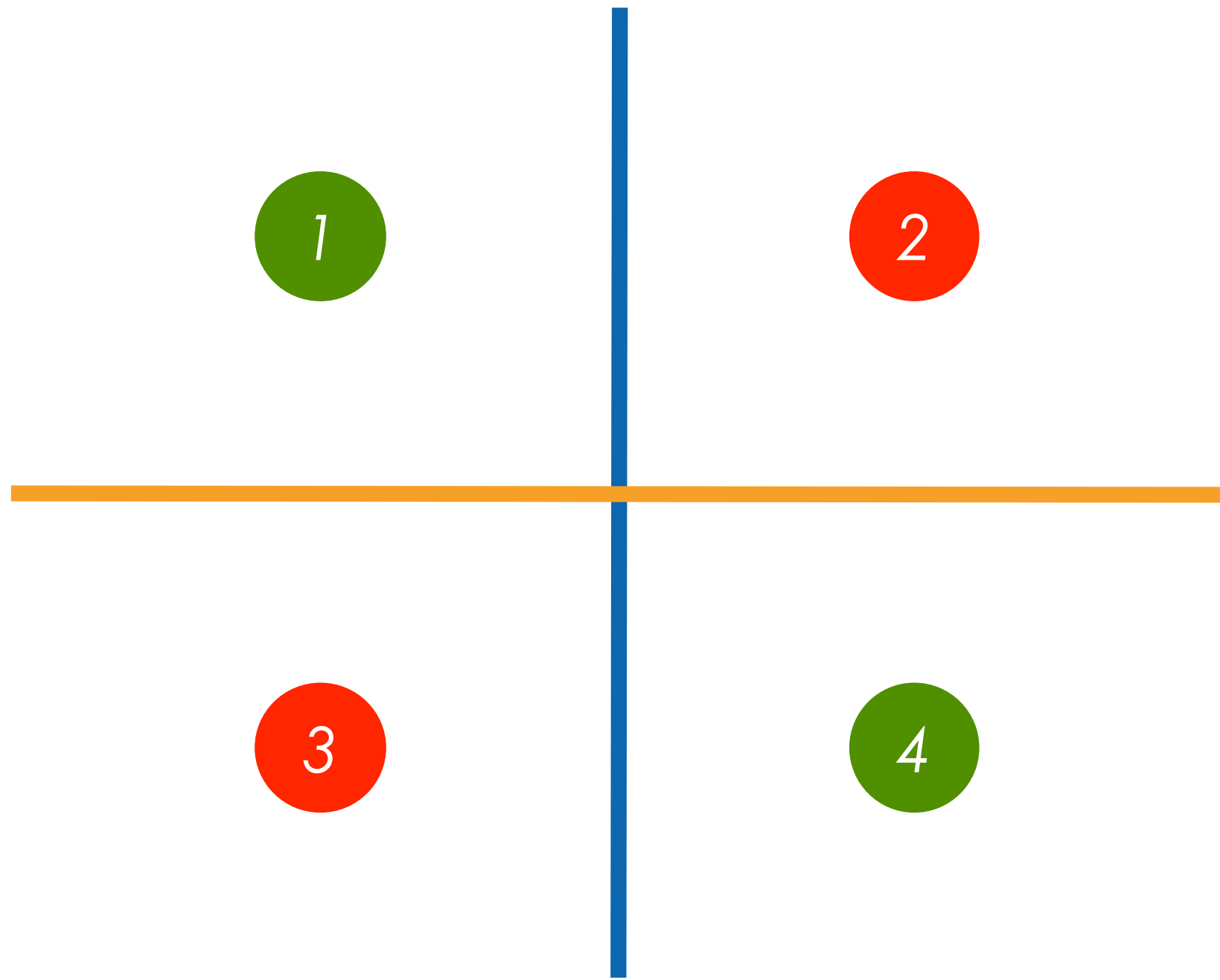
XOR problem

If one can represent AND, OR, NOT, one can represent any logic circuit (including XOR),
by **connecting them**



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

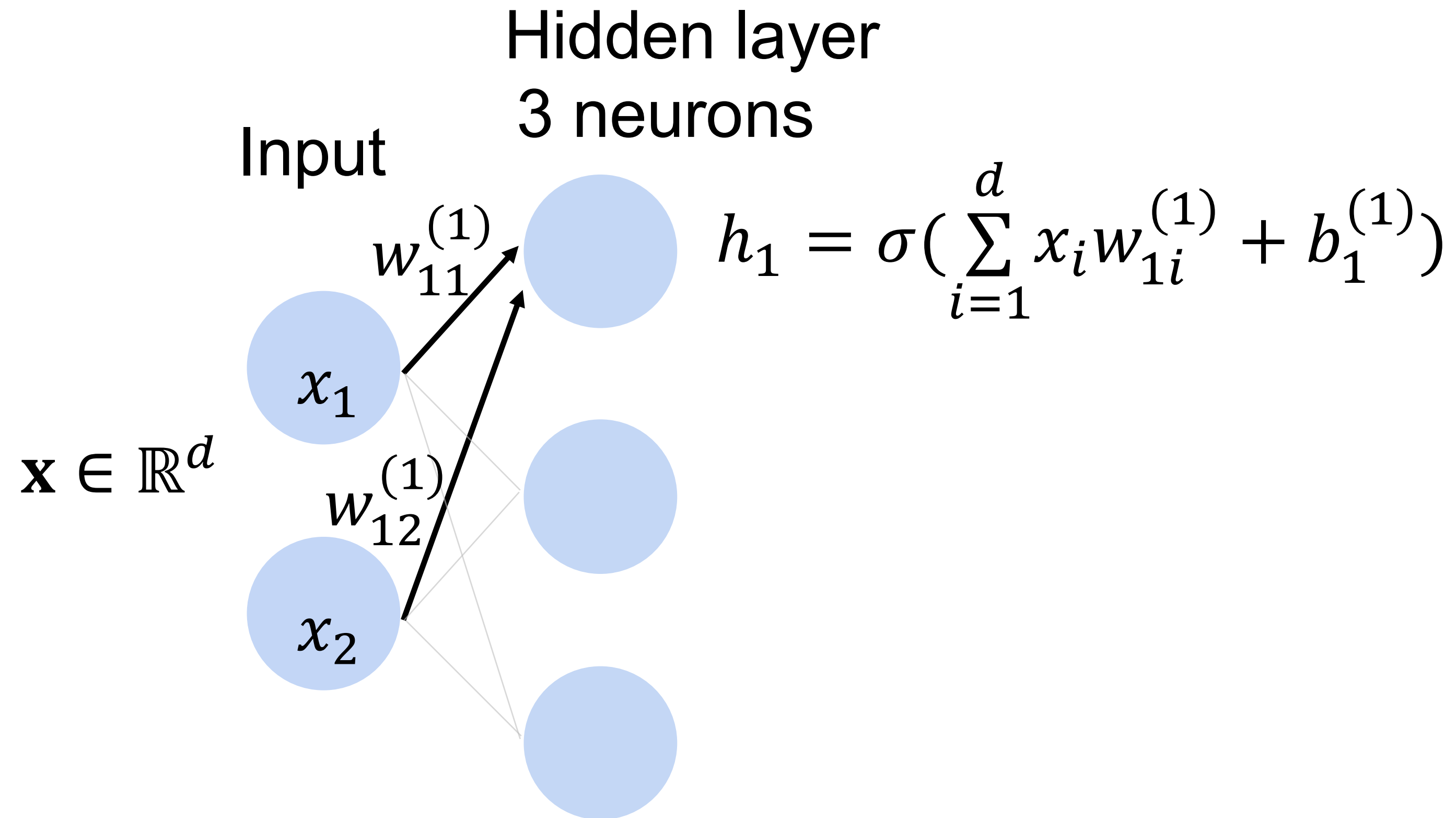
Learning XOR



This motivates using **multi-layer** perceptrons.

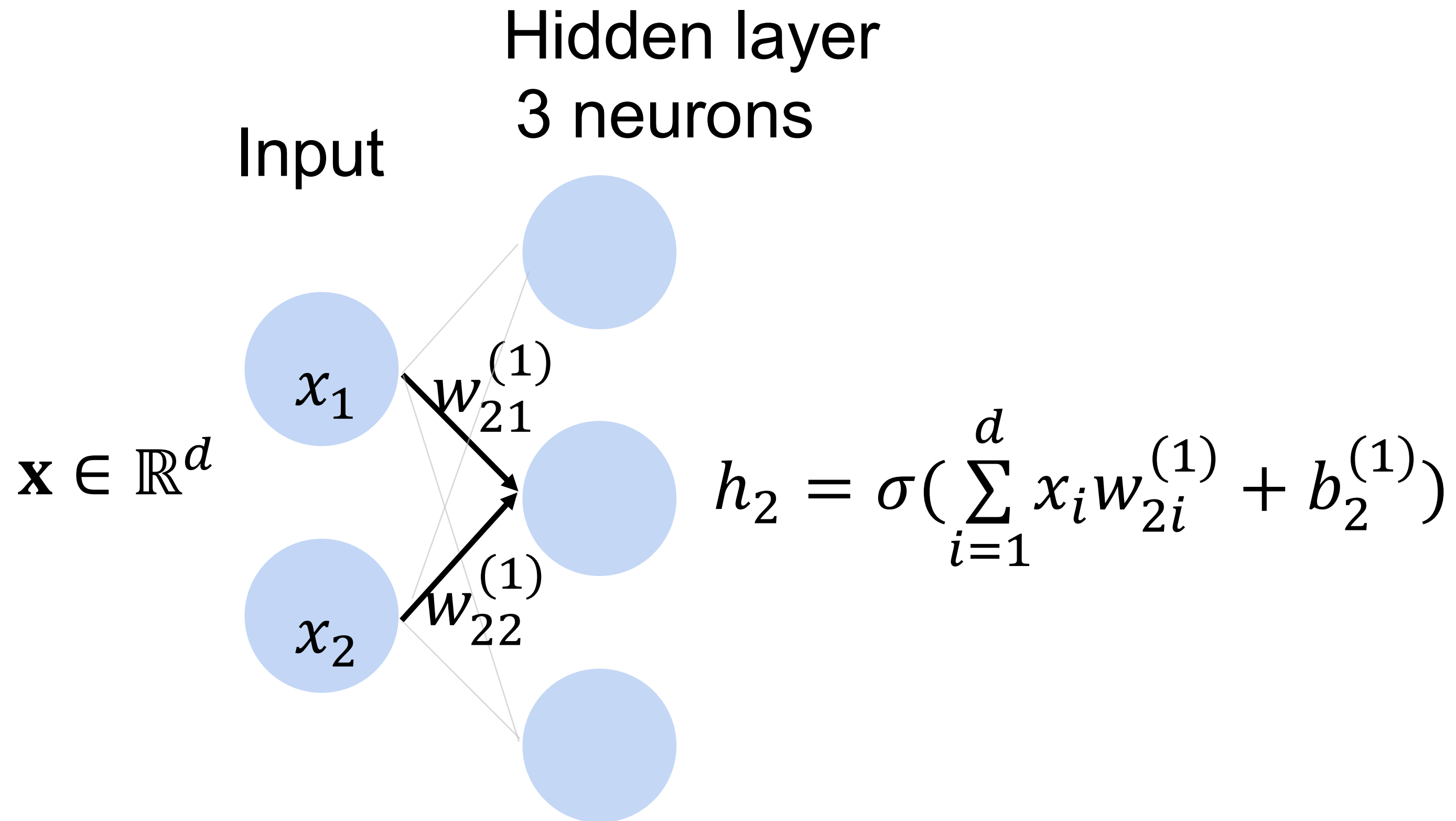
Multi-layer perceptron: Example

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



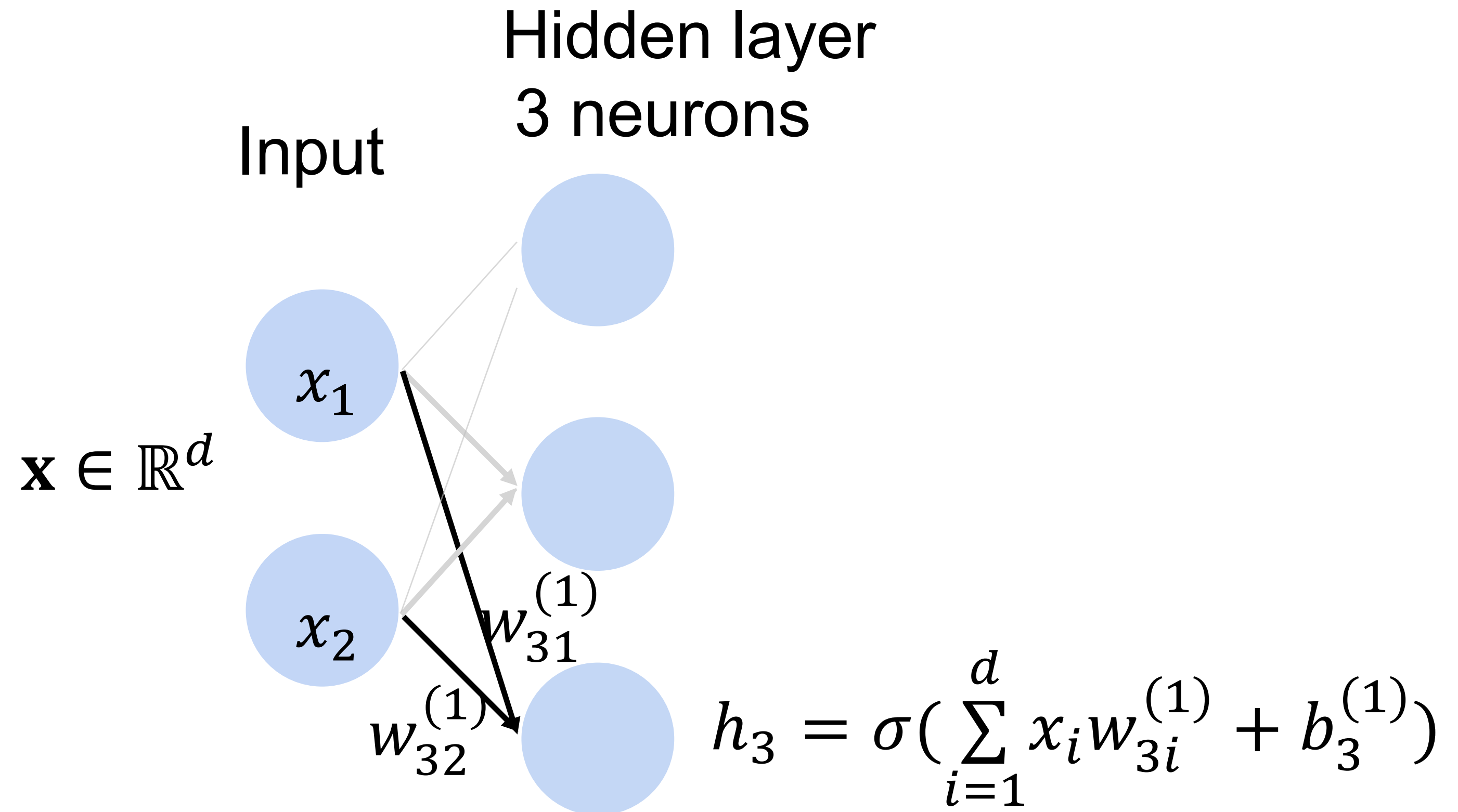
Multi-layer perceptron: Example

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



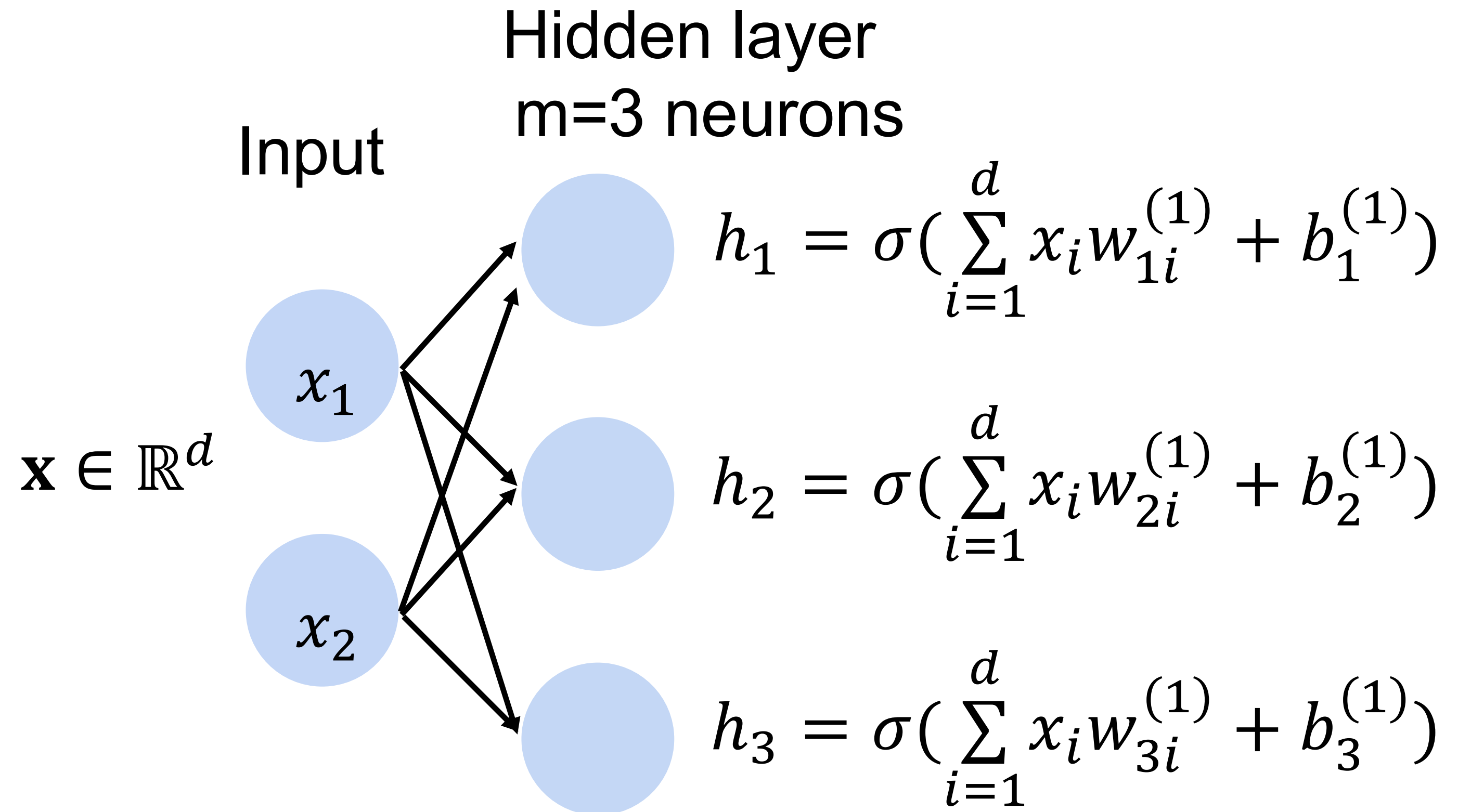
Multi-layer perceptron: Example

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



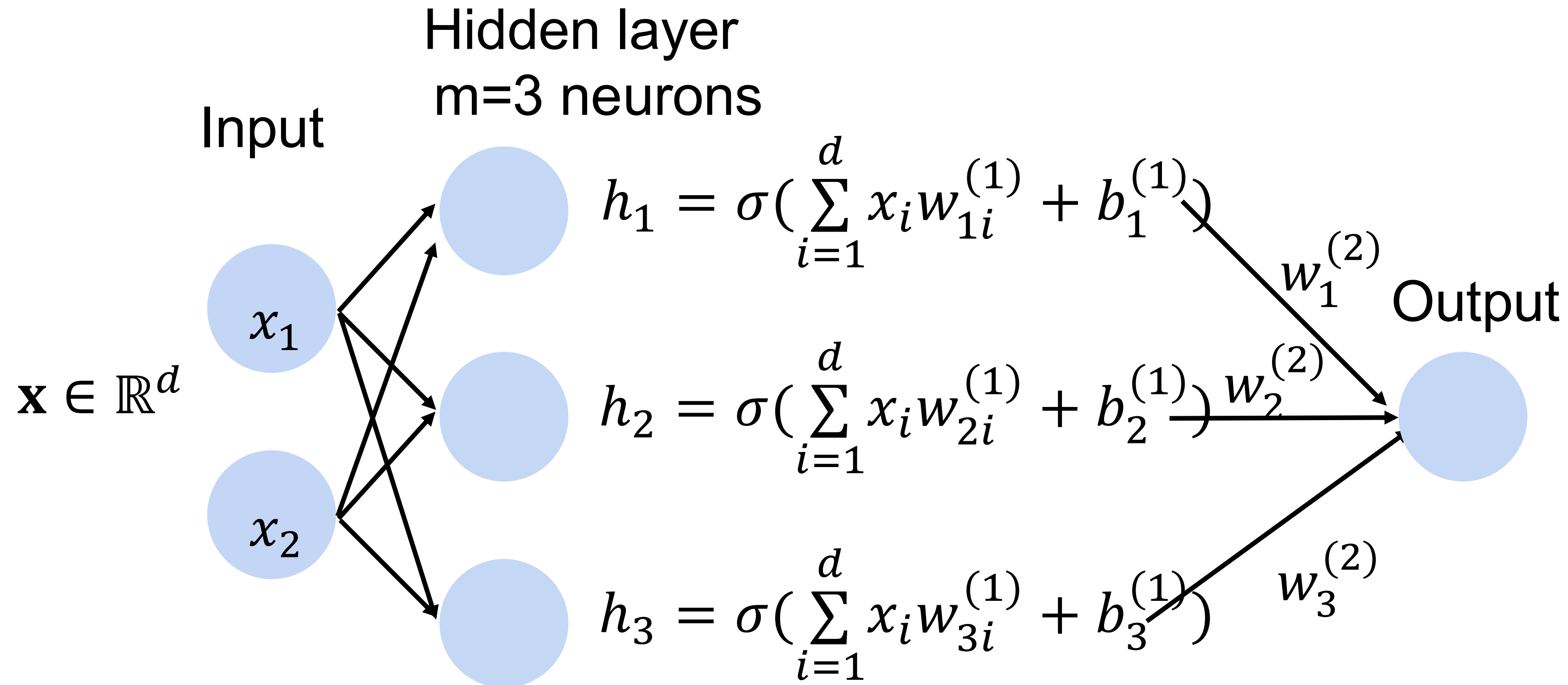
Multi-layer perceptron: Example

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



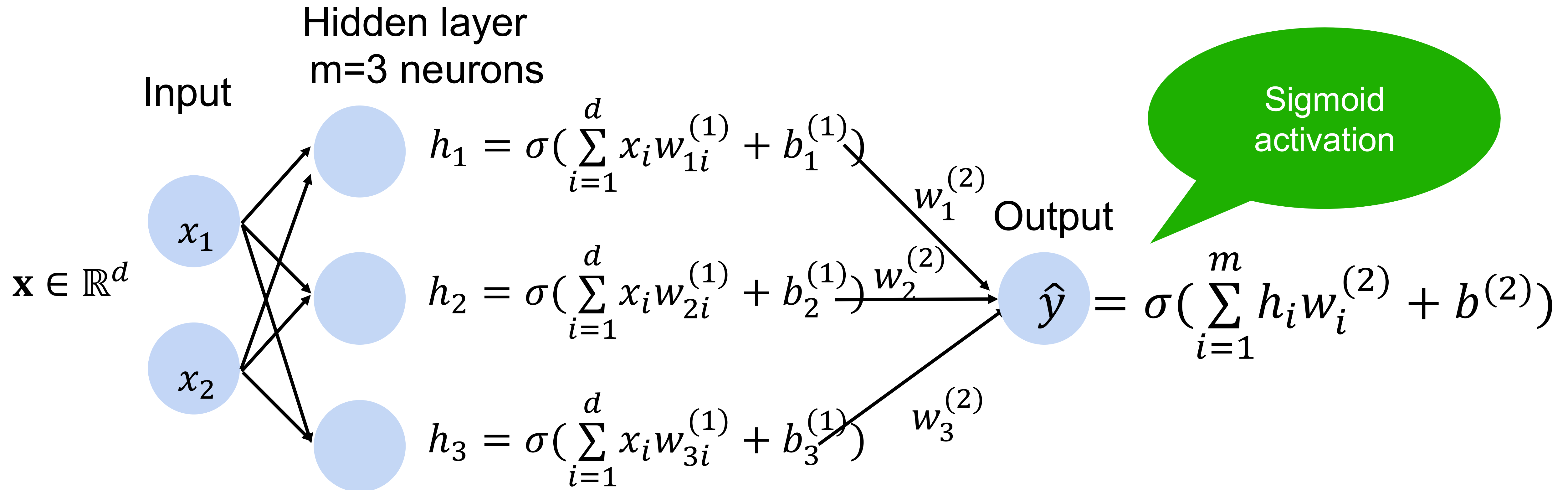
Multi-layer perceptron: Example

- 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

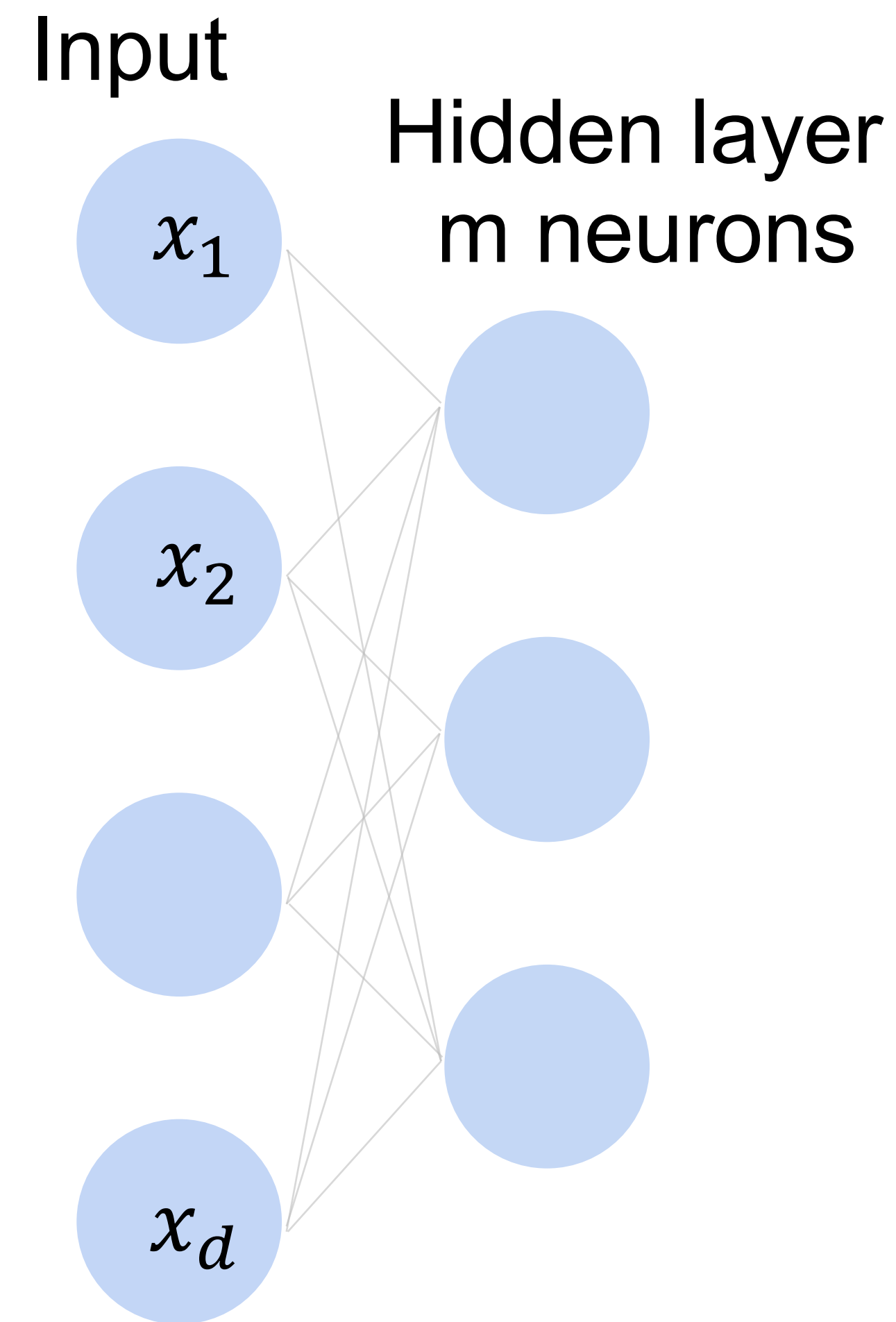


Multi-layer perceptron: Matrix Notation

- 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

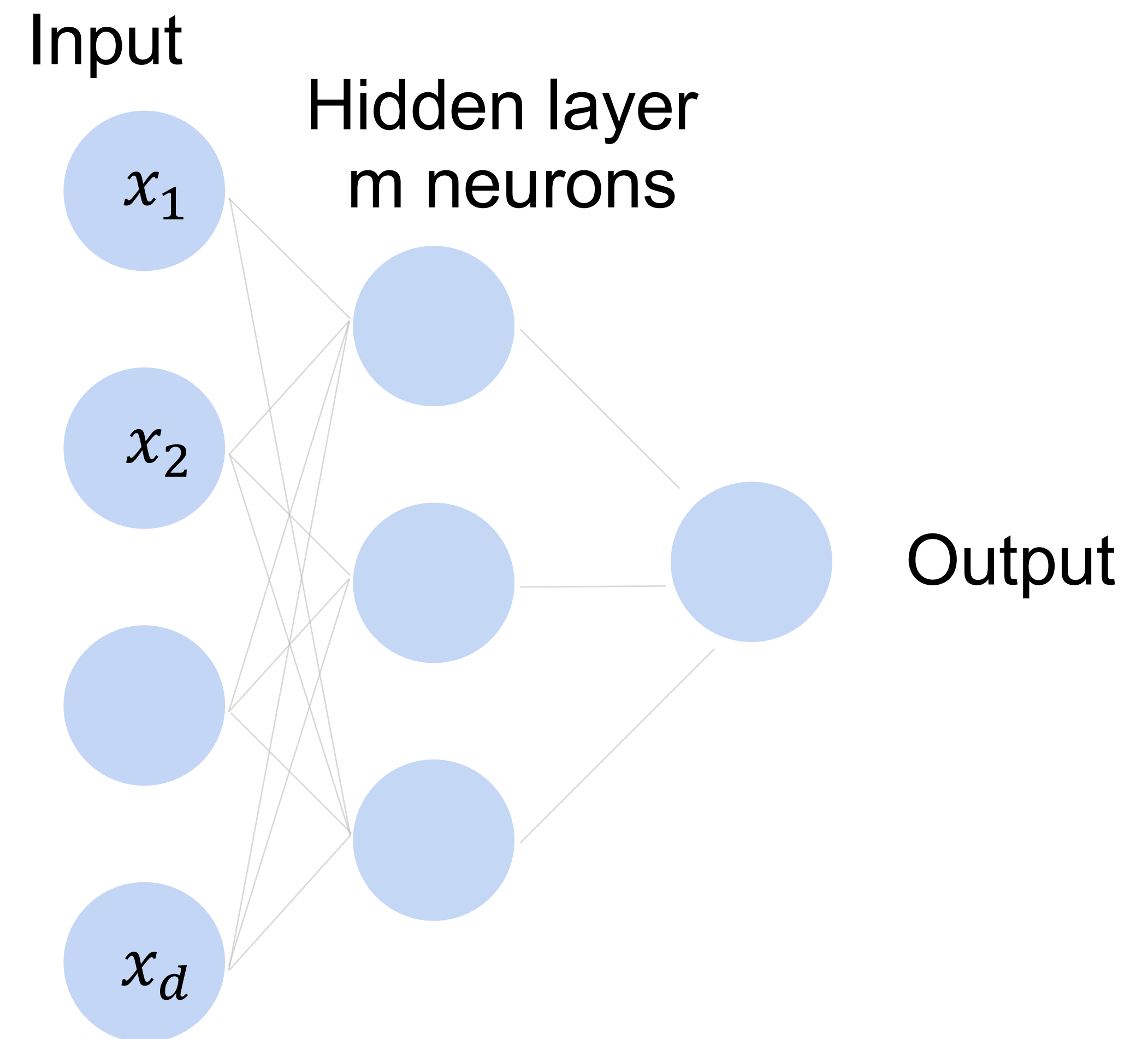
$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

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

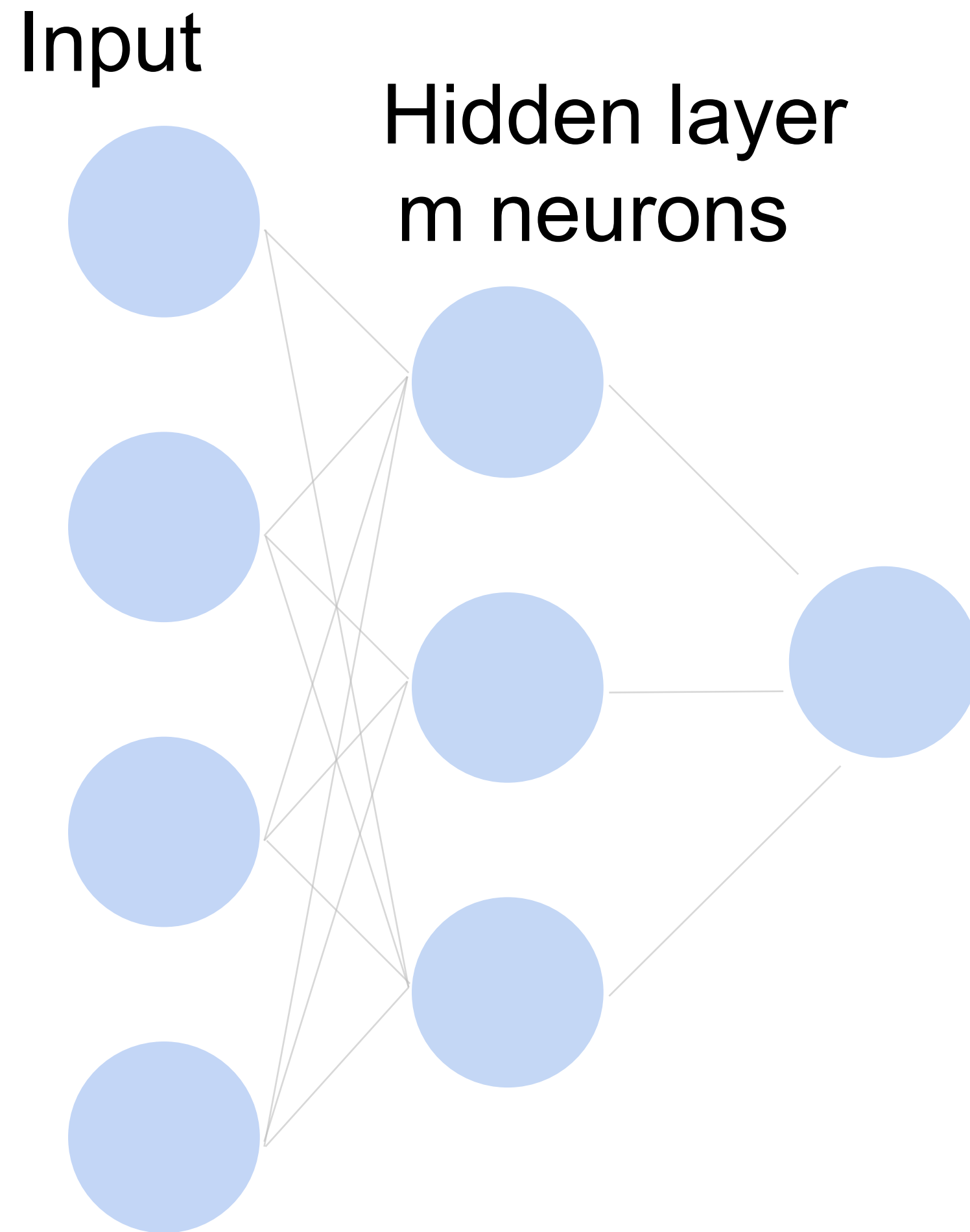


Multi-layer perceptron: Matrix Notation

- 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
$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
- Final output
$$\hat{y} = \sigma(\mathbf{w}^{(2)}\mathbf{h} + \mathbf{b}^{(2)})$$



Multi-layer perceptron



Why do we need a nonlinear activation?

Output

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

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

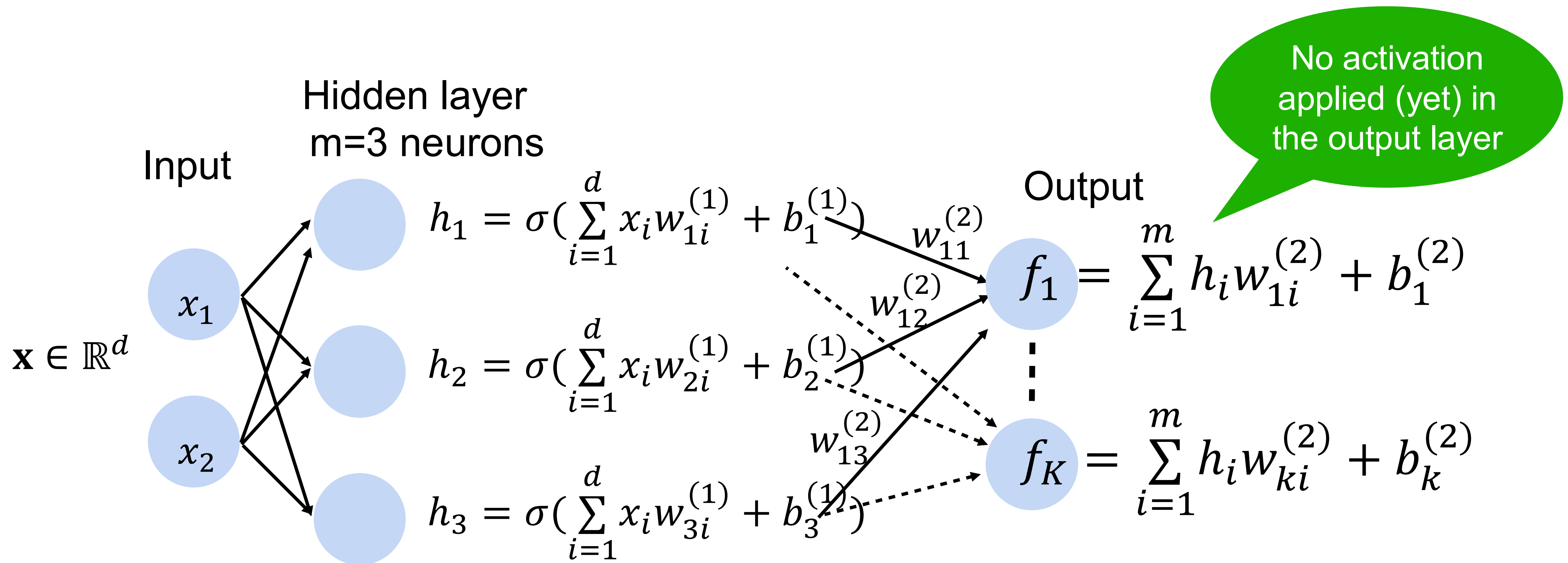
$$\rightarrow f = (\mathbf{w}^{(2)})^\top \mathbf{W}^{(1)}\mathbf{x} + (\mathbf{w}^{(2)})^\top \mathbf{b}^{(1)} + b^{(2)}$$

$$\rightarrow f = \mathbf{w}^\top \mathbf{x} + b$$

Neural network for K-way classification

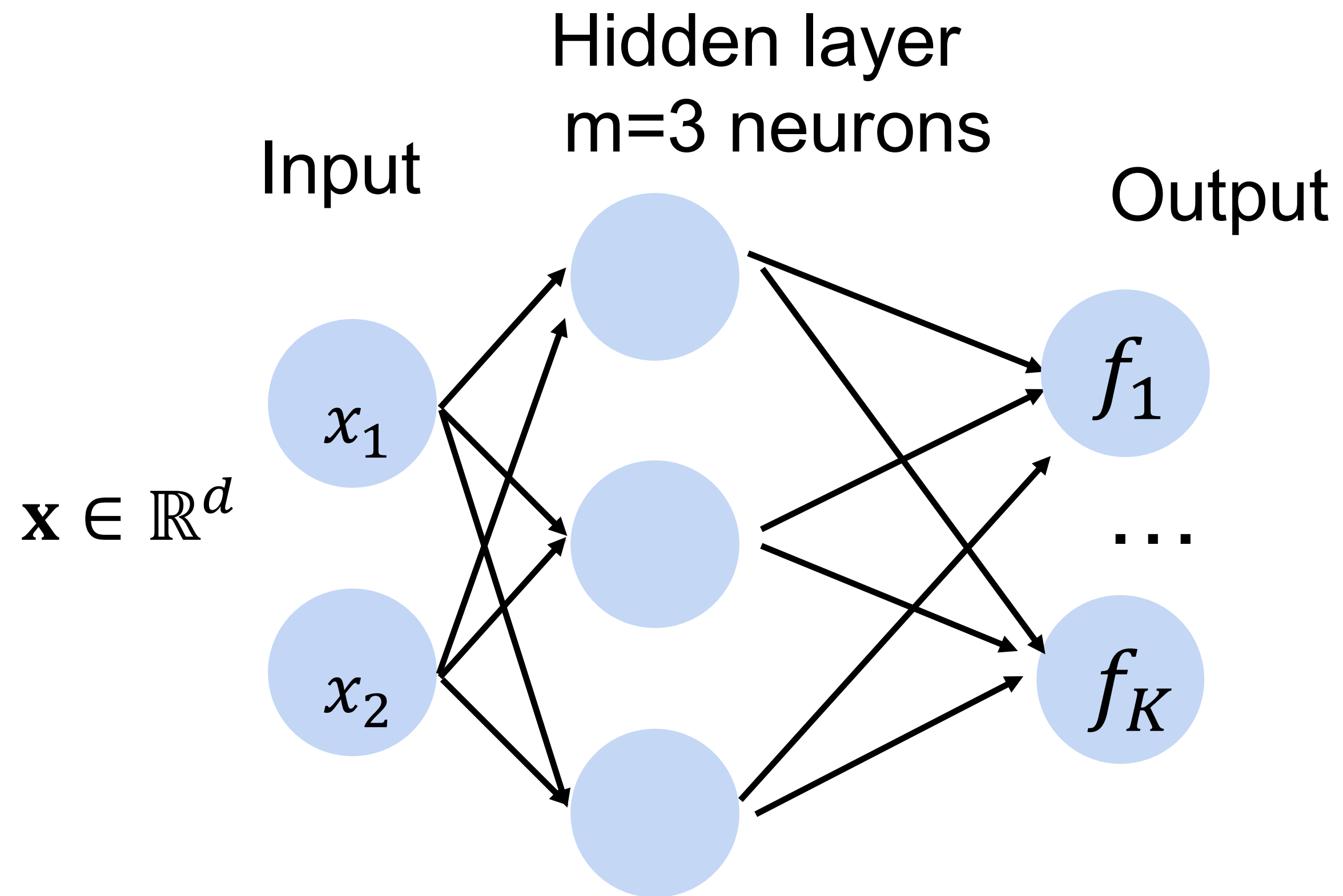
K outputs in the final layer

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



Softmax

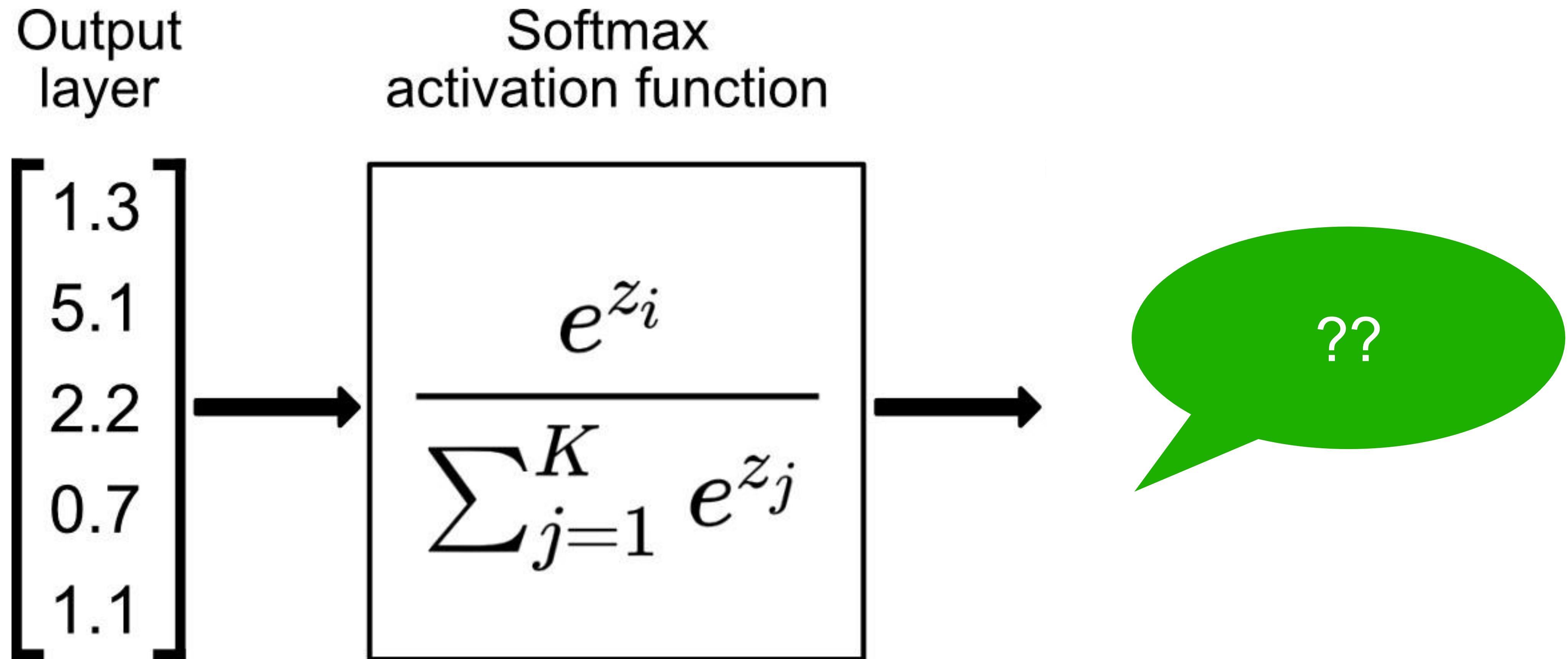
Turns outputs f into probabilities (sum up to 1 across K classes)



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

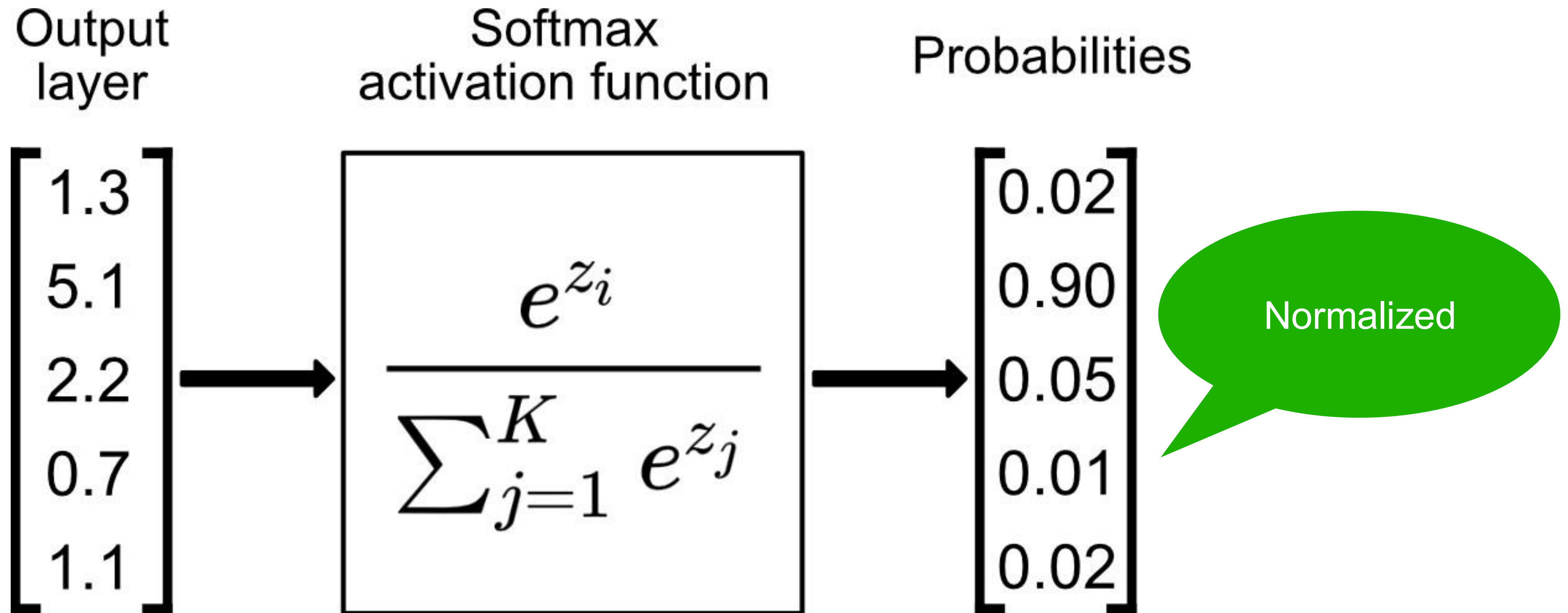
Softmax

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)



Writing in matrix form

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

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

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

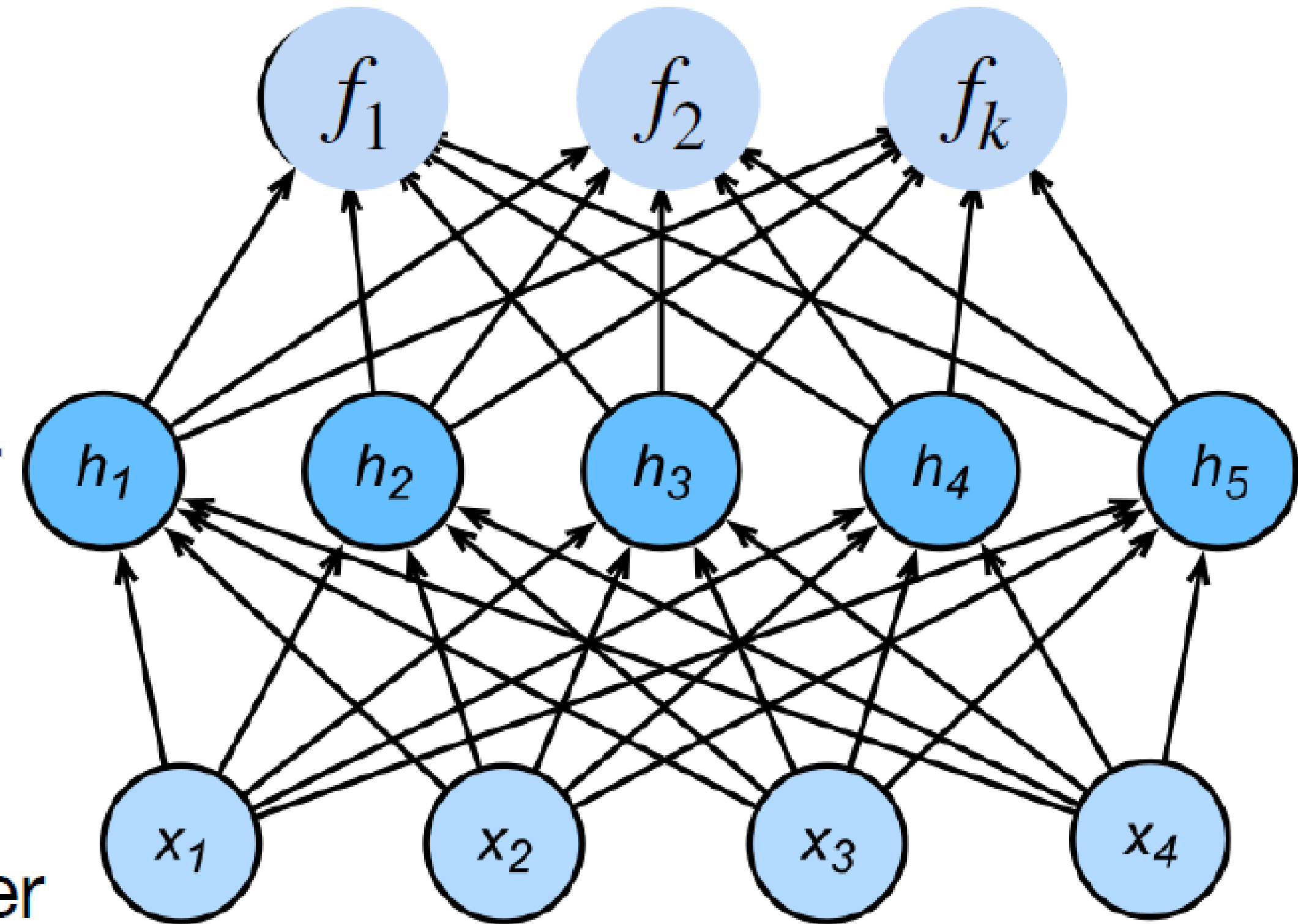
$$\mathbf{f} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

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

Output layer

Hidden layer

Input layer



More complicated neural networks: multiple hidden layers

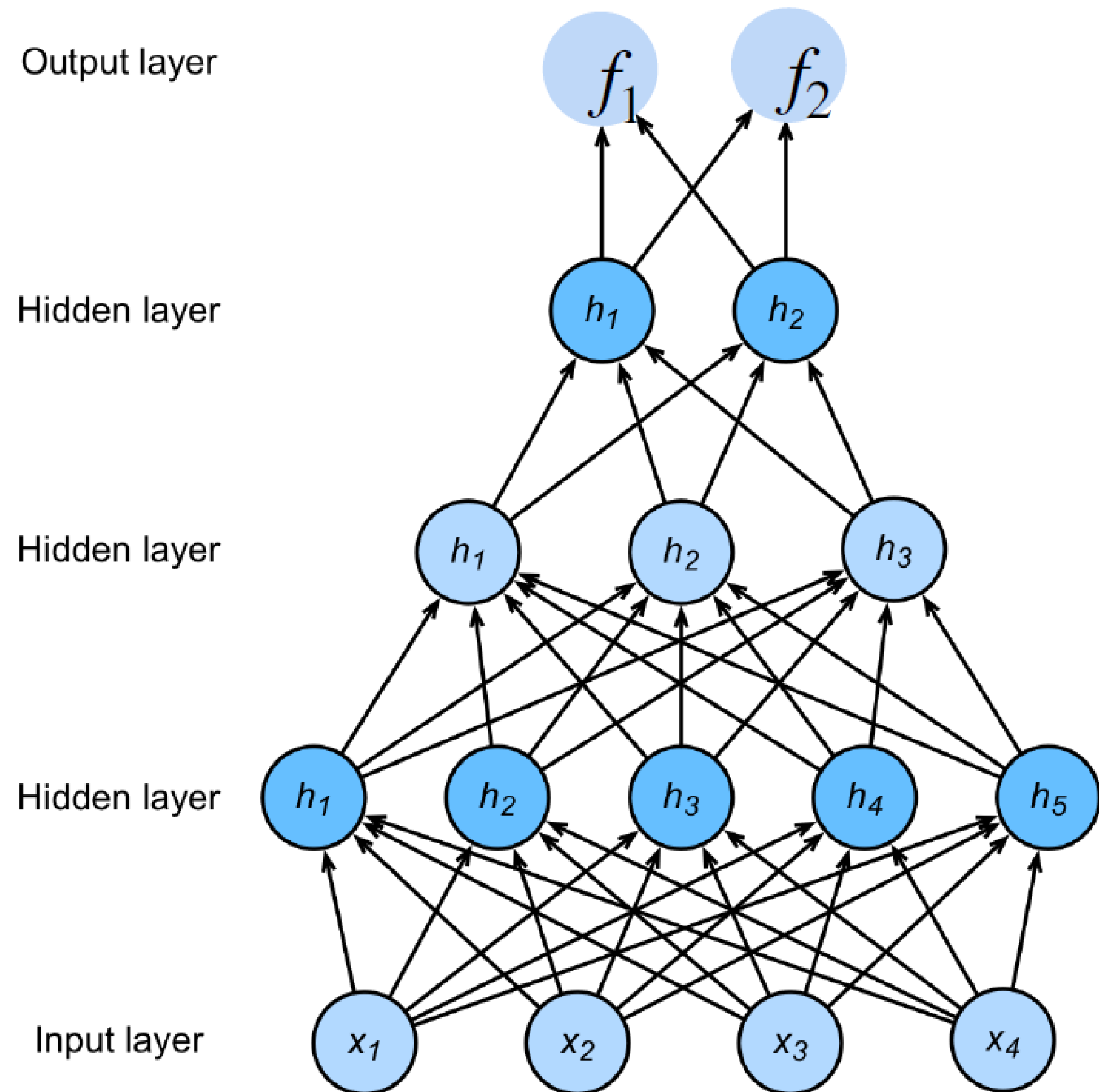
$$\mathbf{h}_1 = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

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

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

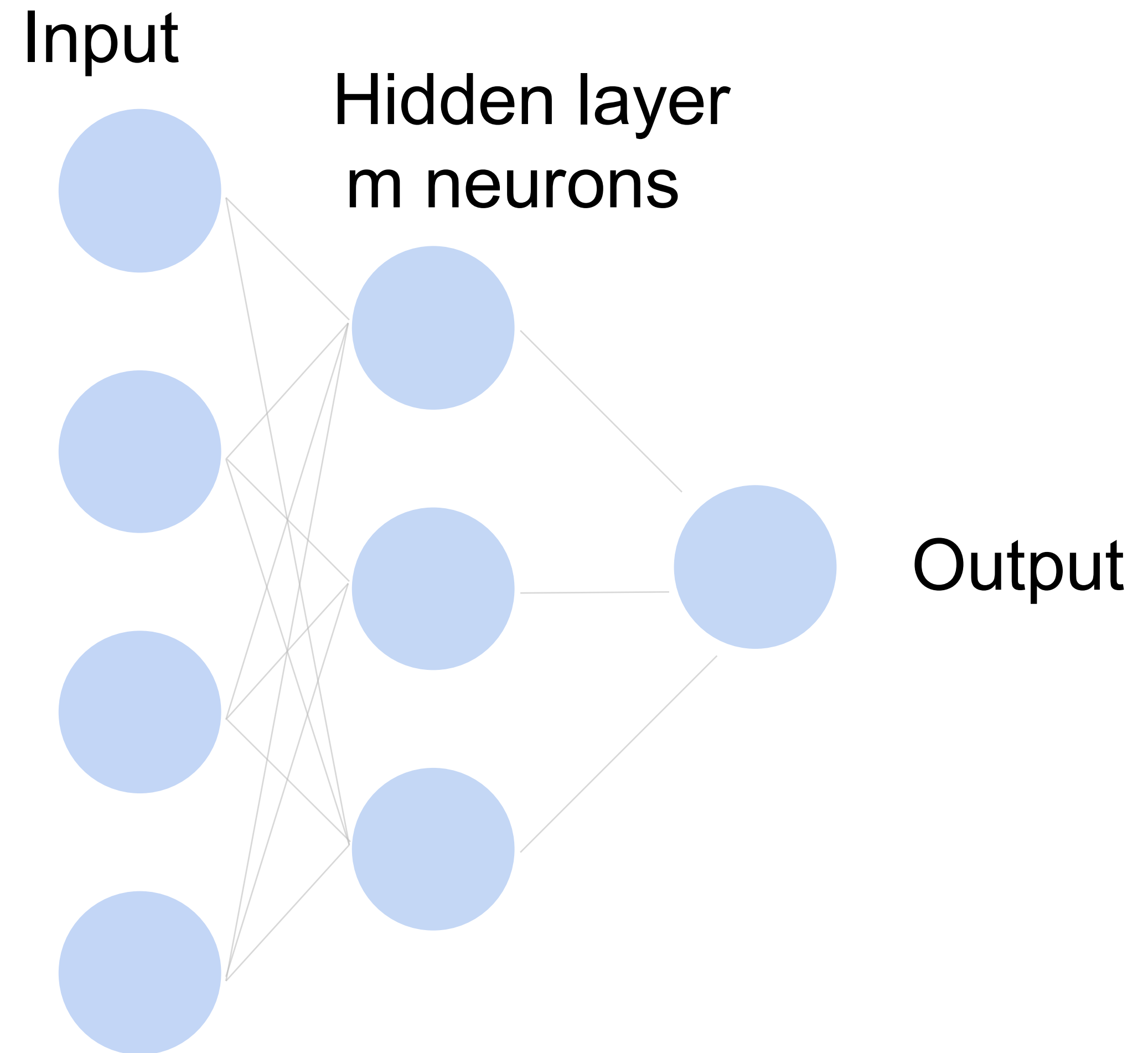
$$\mathbf{f} = \mathbf{W}^{(4)}\mathbf{h}_3 + \mathbf{b}^{(4)}$$

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



How to train a neural network?

Classify cats vs. dogs



Recall loss for logistic regression (derived from MLE)

- Conditional log likelihood: $\ell(\theta; X, Y) = \log P_{\theta}(X, Y) = \log \prod_{i=1}^n p_{\theta}(y_i | x_i)$
- MLE is: $\max_{\theta} \ell(\theta; X, Y) = \min_{\theta} - \sum_{i=1}^n \log p_{\theta}(y_i | x_i)$
- Let us write this as

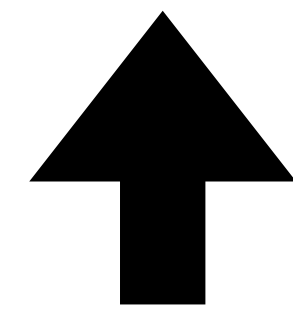
$$\begin{aligned} \min_{\theta} \sum_{i=1}^n - \left(\mathbb{1}(y_i = 1) \log p_{\theta}(y_i = 1 | x_i) + \mathbb{1}(y_i = 0) \log p_{\theta}(y_i = 0 | x_i) \right) \\ = \min_{\theta} \frac{1}{n} \sum_{i=1}^n \left(- \sum_{j \in \{0,1\}} \mathbb{1}(y_i = j) \log p_{\theta}(y_i = j | x_i) \right) \end{aligned}$$

Cross entropy loss: K-way classification

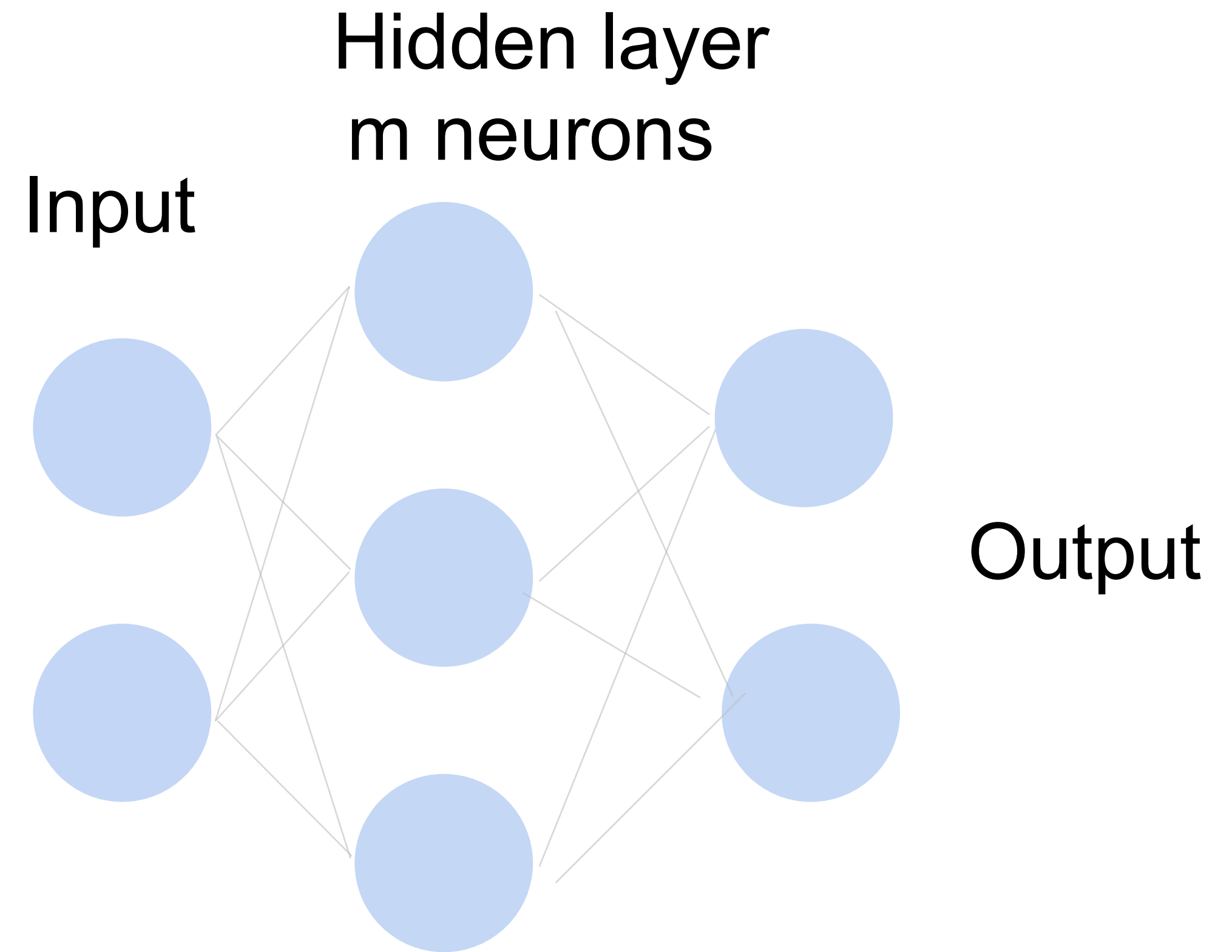
Loss function: $\frac{1}{|D|} \sum_i \ell(\mathbf{x}_i, y_i)$

Represent y_i 's as one-hot vectors.
Loss on sample i when softmax
outputs p_j :

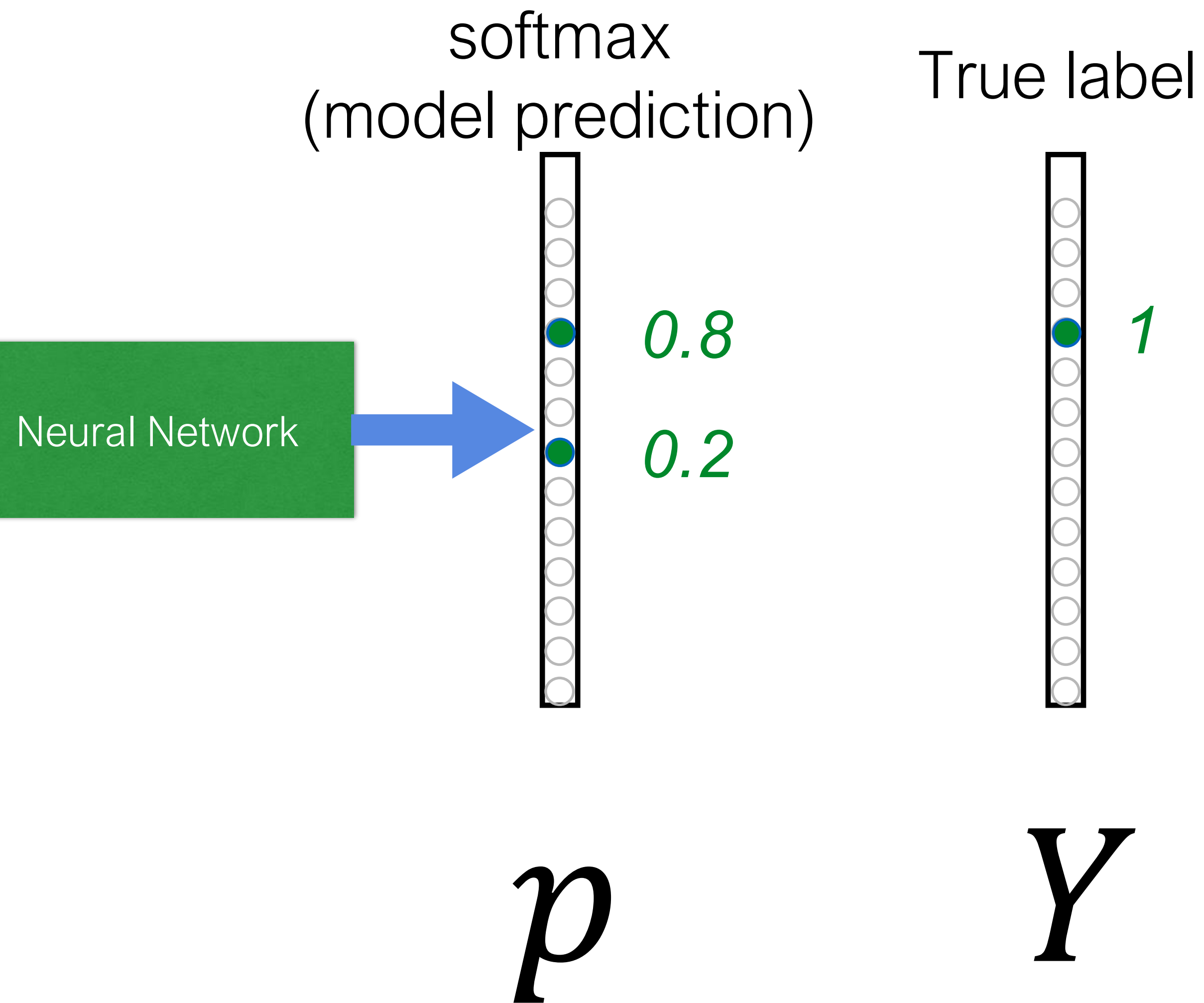
$$\ell(\mathbf{x}, y) = \sum_{j=1}^K -y_{i,j} \log p_j$$



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



Cross-Entropy Loss



$$L_{CE} = \sum_j -y_j \log(p_j)$$

$$= -\log(0.8)$$

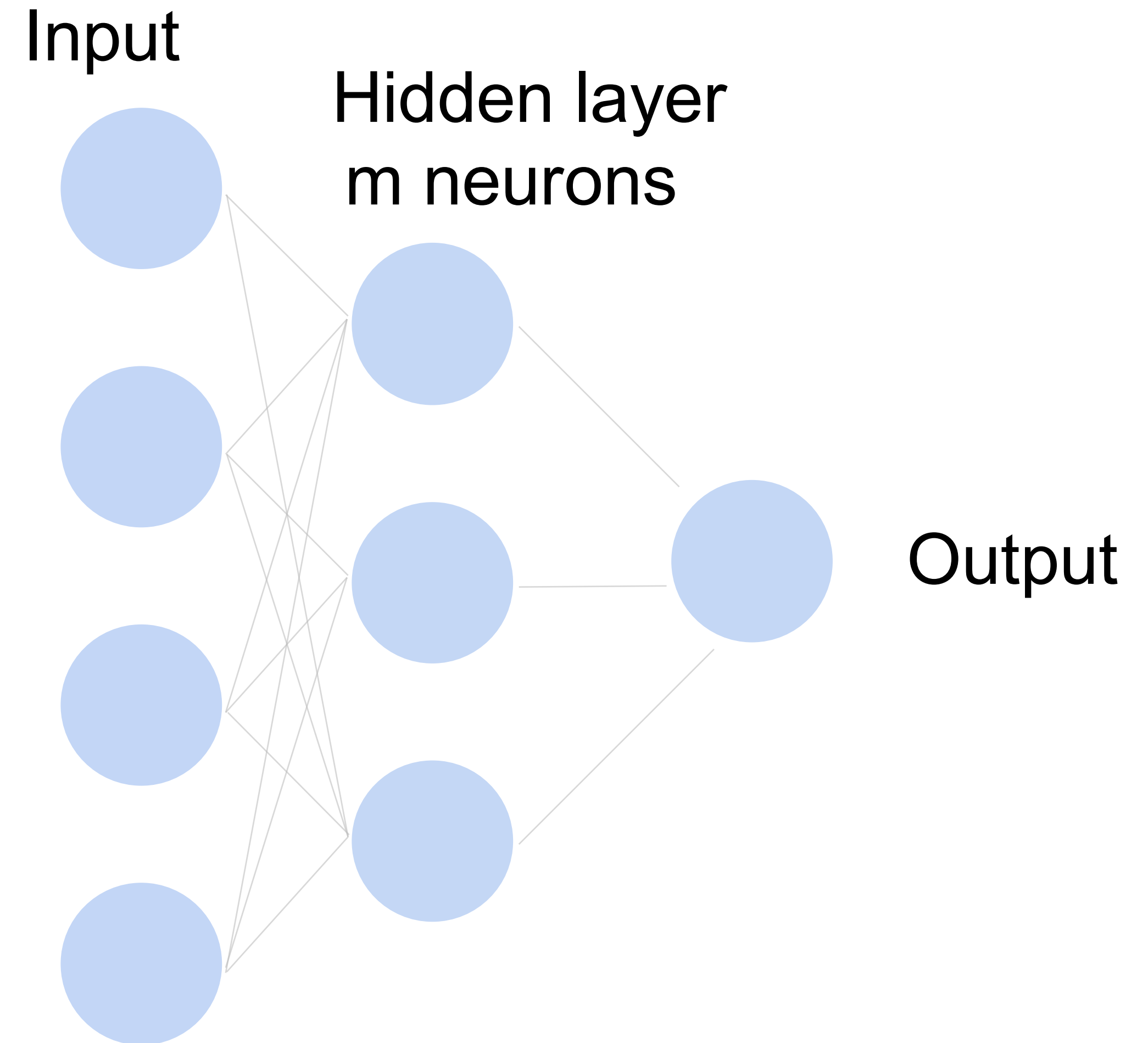
Goal: push $\mathbf{p}$ and $\mathbf{Y}$ to be identical

How to train a neural network?

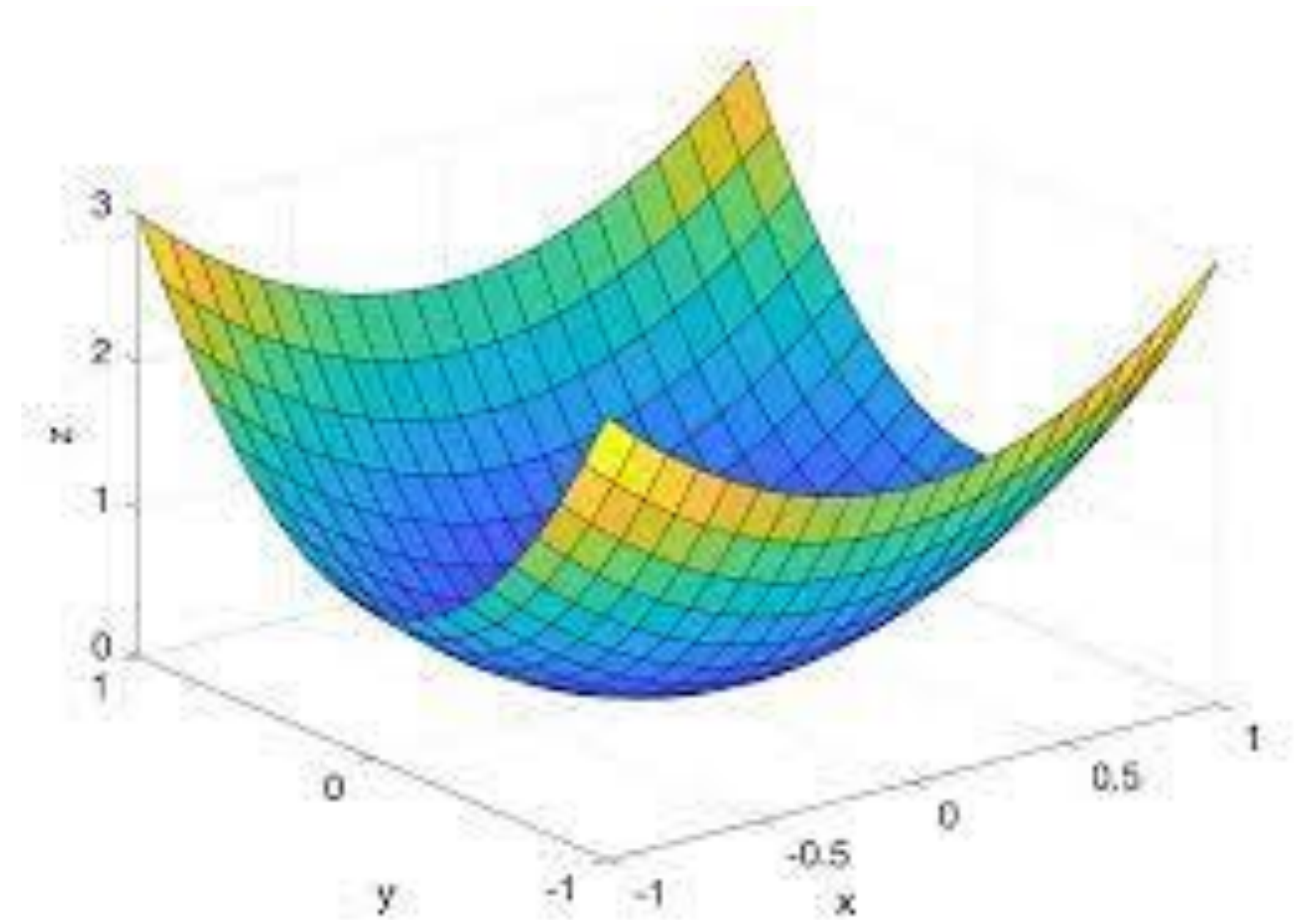
Update the weights W to minimize the loss function

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

Use gradient descent!



Detour: Multivariate Calculus

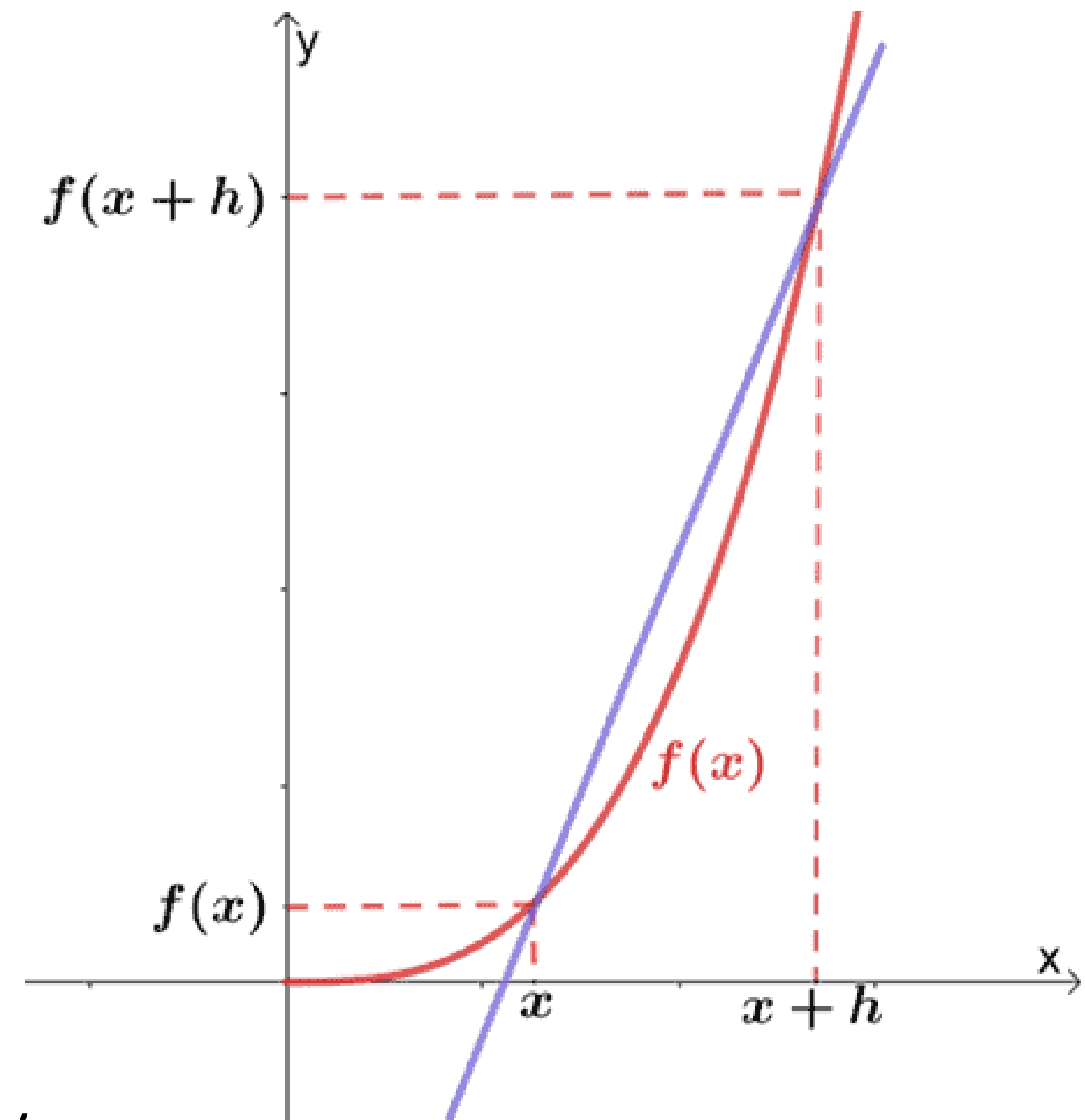


<https://machinelearningmastery.com/a-gentle-introduction-to-multivariate-calculus/>

Derivatives of functions of single variables

- Given function $f(x)$, we would like to find the slope of the tangent line at any point x_0 .
- Why? Tells us how fast $f(x)$ is increasing / decreasing at x_0 .

- $$\frac{df}{dx} = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$



Derivatives of functions of single variables

- For many functions $f(x)$ we have simple rules to find the derivative.
- If $f(x) = cx^2$ then $\frac{df}{dx} = 2cx$
 - Or more generally, if $f(x) = cx^p$ then $\frac{df}{dx} = cpx^{p-1}$
- If $f(x) = \log x$ then $\frac{df}{dx} = \frac{1}{x}$
- If $f(x) = c$ then $\frac{df}{dx} = 0$

More Complex Functions

- Derivation rules can be applied hierarchically to find derivatives of more complex functions.
- **Sum rule:** If $f(x) = h(x) + g(x)$ then $\frac{df}{dx} = \frac{dh}{dx} + \frac{dg}{dx}$
- **Product rule:** If $f(x) = h(x) \cdot g(x)$ then $\frac{df}{dx} = h(x) \frac{dg}{dx} + g(x) \frac{dh}{dx}$
- **Chain rule:** If $f(x) = h(g(x))$ then $\frac{df}{dx} = \frac{dh}{dg} \frac{dg}{dx}$
- **More complex chain rule:** if $f(x) = f_1(f_2(\cdots f_n(x) \cdots))$ then $\frac{df}{dx} = \frac{df_1}{df_2} \frac{df_2}{df_3} \cdots \frac{df_n}{dx}$

Derivatives of functions of multiple variables

- Generalize derivative to functions of form $f(x_1, x_2, \dots, x_n)$.
- The partial derivative $\frac{\partial f}{\partial x_1}$ tells us how fast f increases / decreases as we increase the value of x_1 .
- For each variable x_i we have a partial derivative $\frac{\partial f}{\partial x_i}$.
- The **gradient** is the vector of all of these partial derivatives.
 - $\frac{df}{d\mathbf{x}} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right]$

Computing Partial Derivatives

- Rules from single-variable calculus directly extend to multivariate calculus with one small change.
- When computing $\frac{\partial f}{\partial x_i}$, treat all other variables as constants.
- Examples:
 - If $f(x_1, x_2) = \log x_1 + \log x_2$ then $\frac{\partial f}{\partial x_1} = \frac{1}{x_1}$
 - If $f(x_1, x_2) = x_1(x_1 + x_2)$ then $\frac{\partial f}{\partial x_2} = x_1$

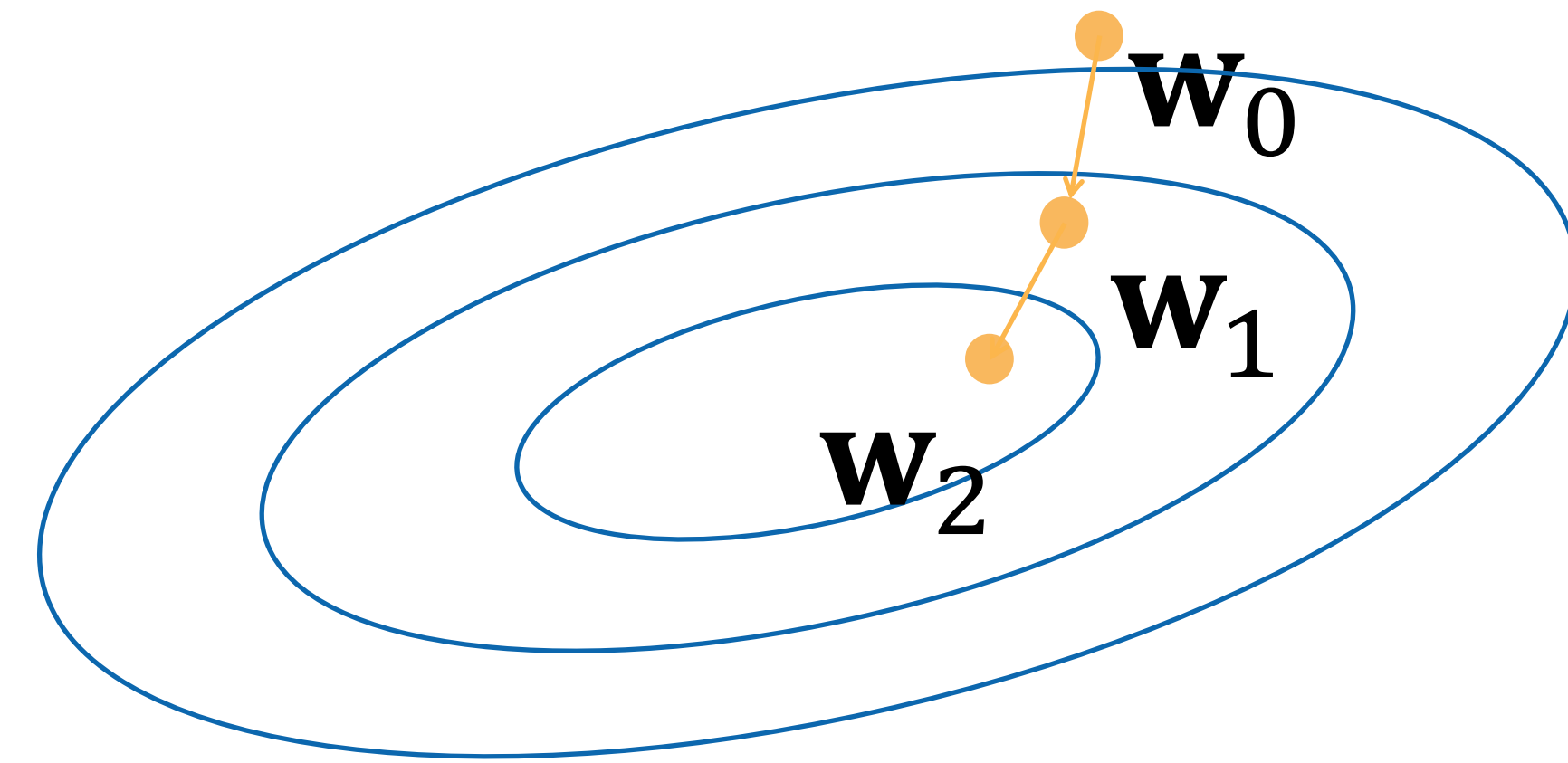
Gradient Descent

- Choose a learning rate $\eta > 0$
- Initialize the model parameters $\mathbf{w}_0$
- For $t = 1, 2, \dots$
 - Update parameters:

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

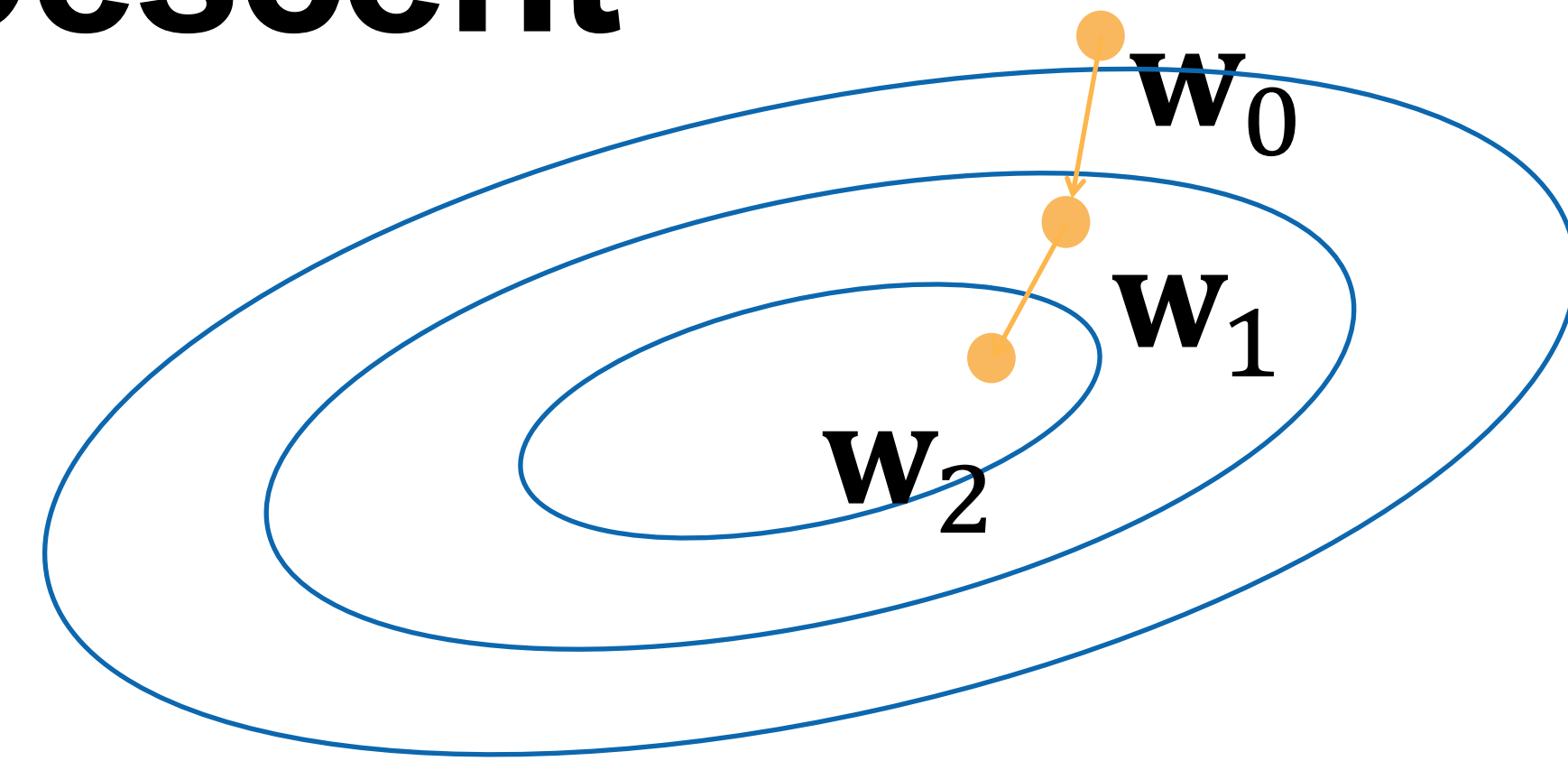
D can be very large. Expensive per iteration

- Repeat until convergence



Minibatch Stochastic Gradient Descent

- Choose a learning rate $\eta > 0$
- Initialize the model parameters $\mathbf{w}_0$
- For $t = 1, 2, \dots$

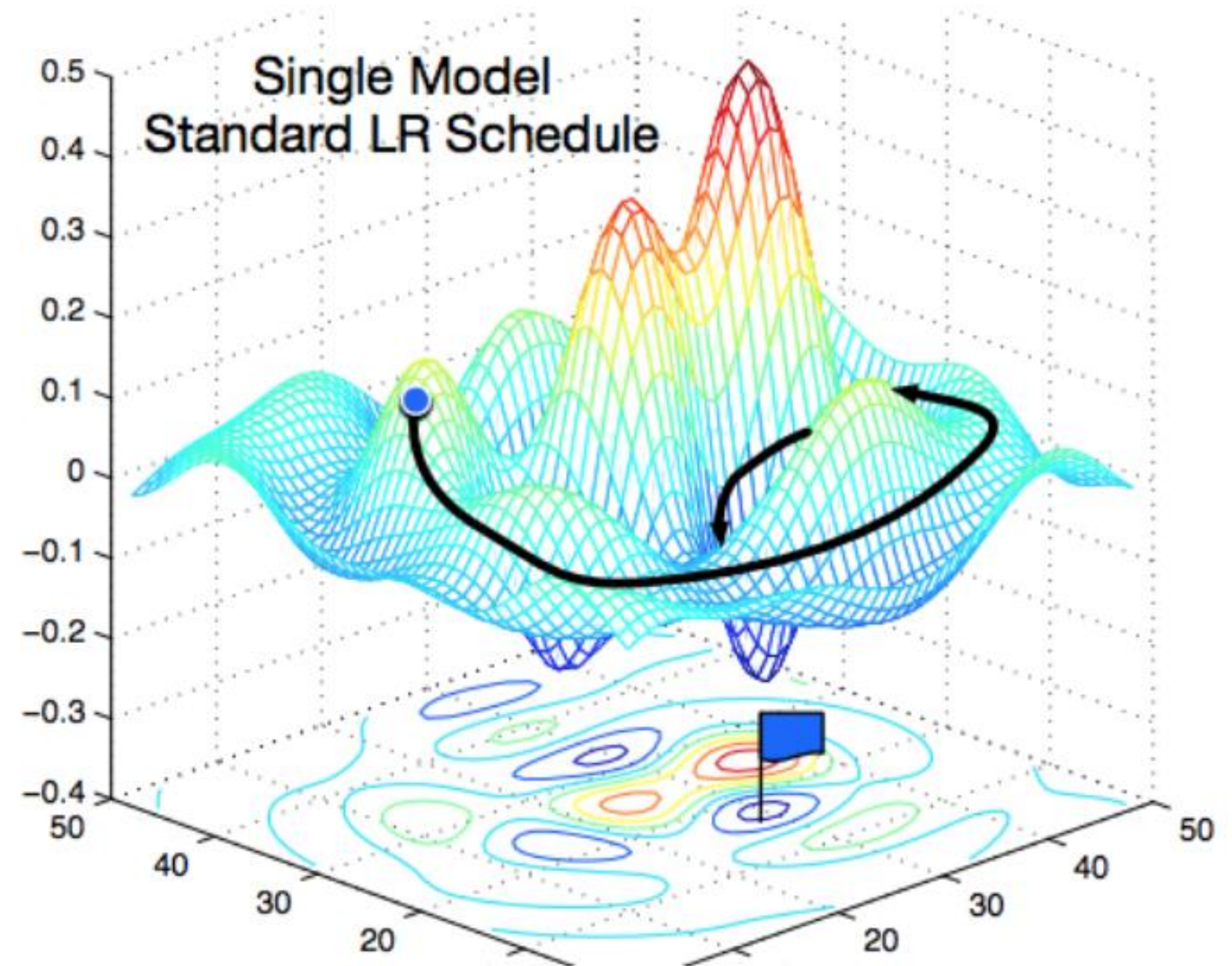


- Randomly sample a subset (mini-batch) $B \subset D$
- Update parameters:

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

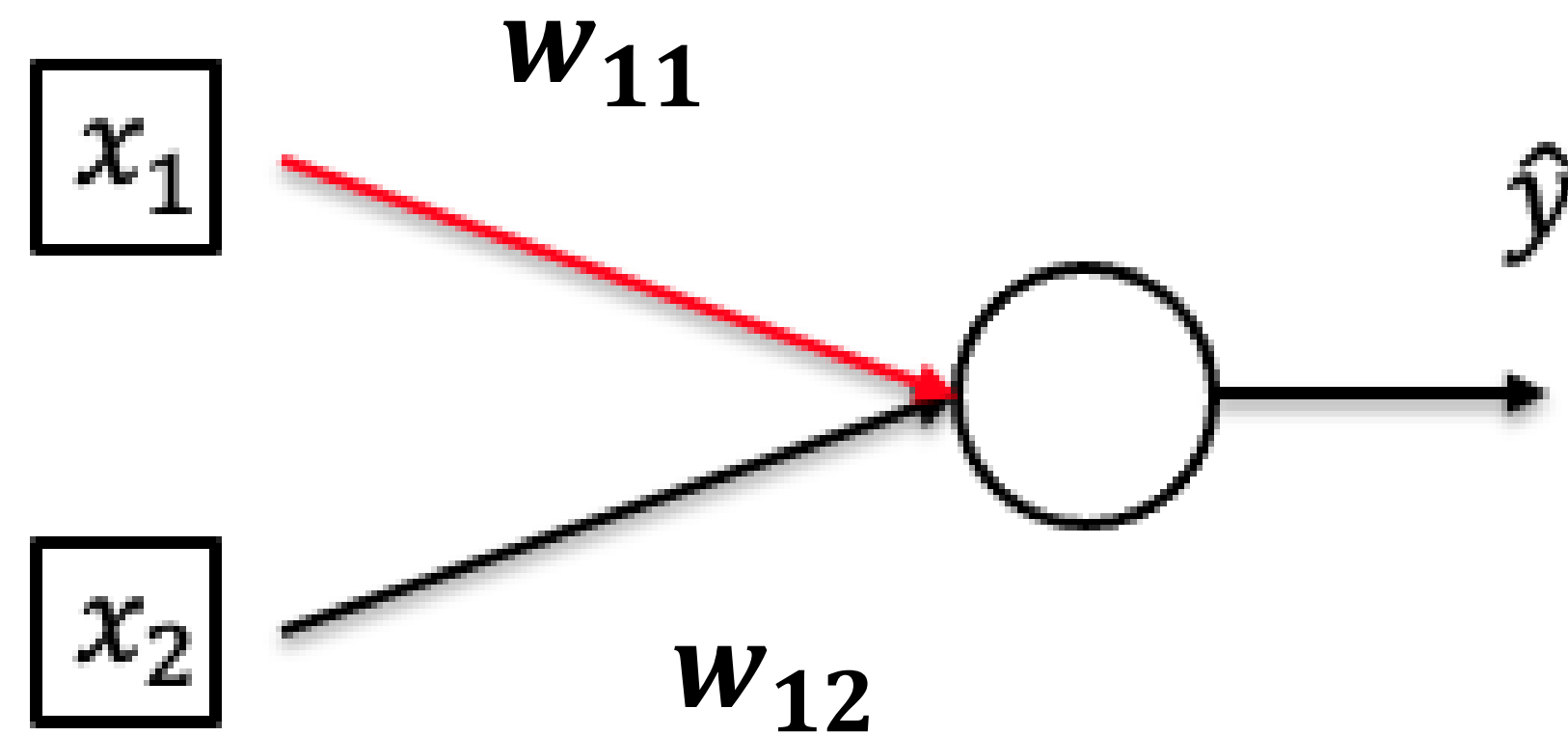
- Repeat until convergence

Computing gradients in a neural network



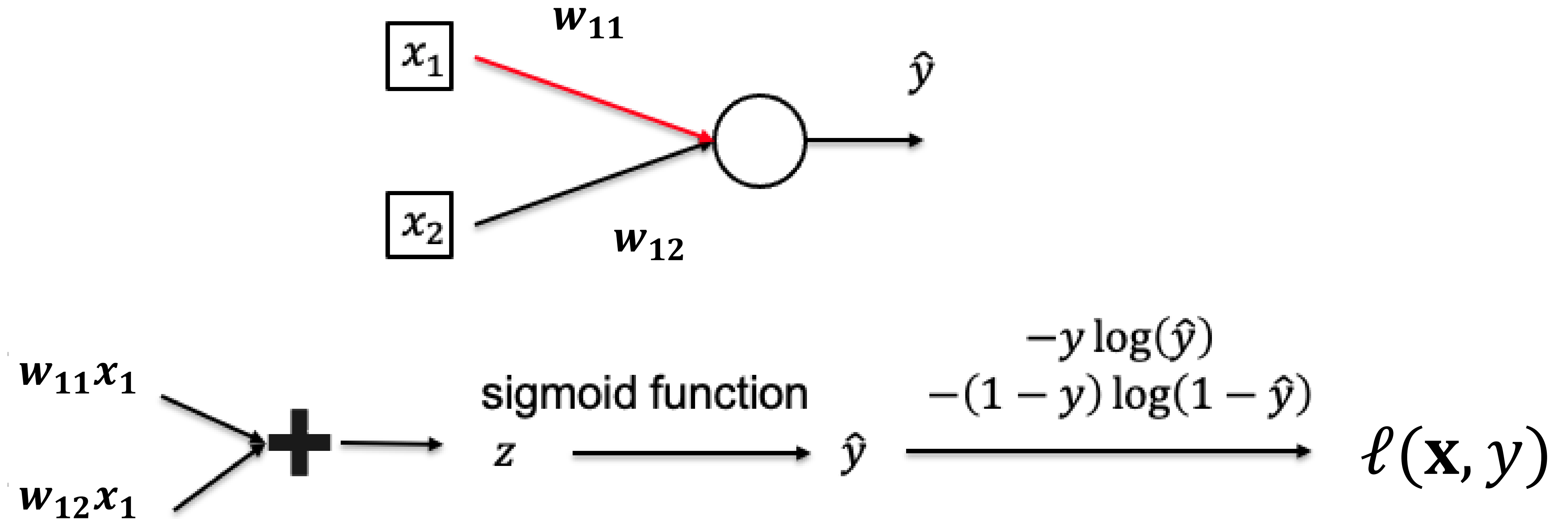
[Gao and Li et al., 2018]

Calculate Gradient (on one data point)



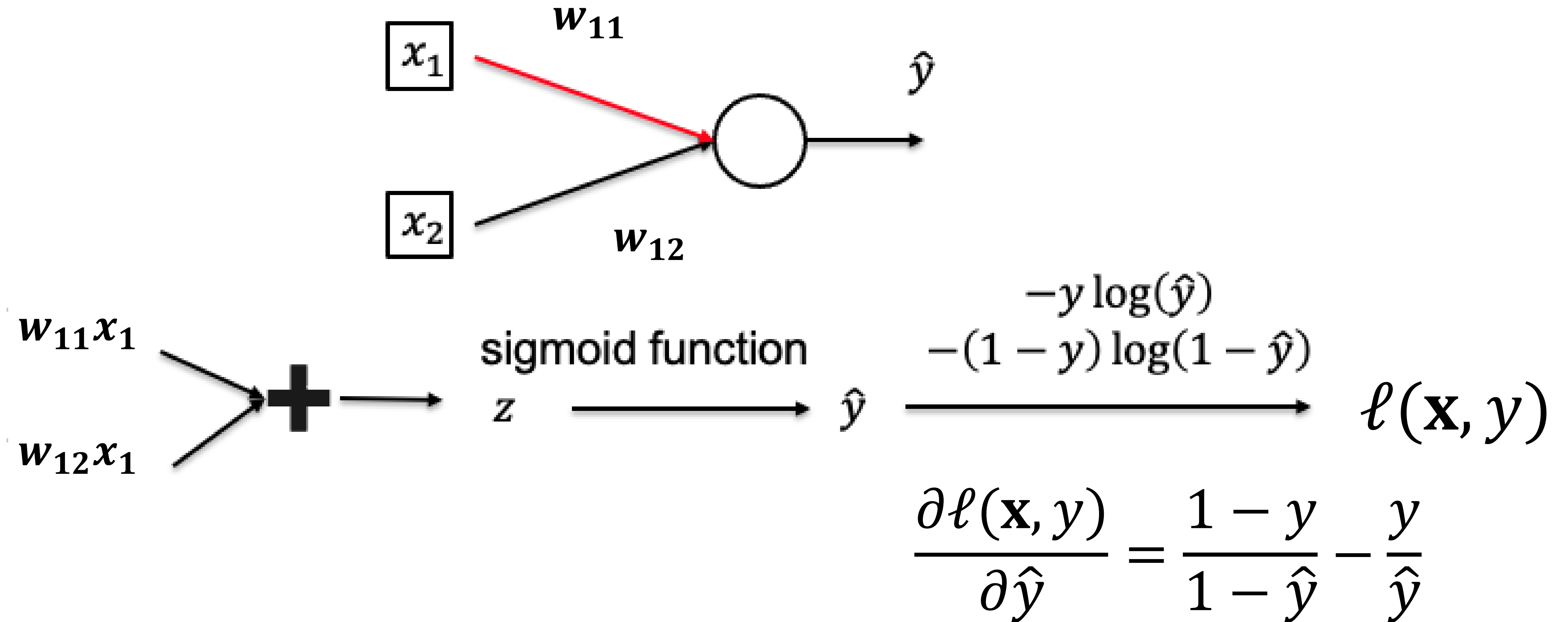
- Want to compute $\frac{\partial \ell(\mathbf{x}, y)}{\partial w_{11}}$
- Data point: $((x_1, x_2), y)$

Calculate Gradient (on one data point)



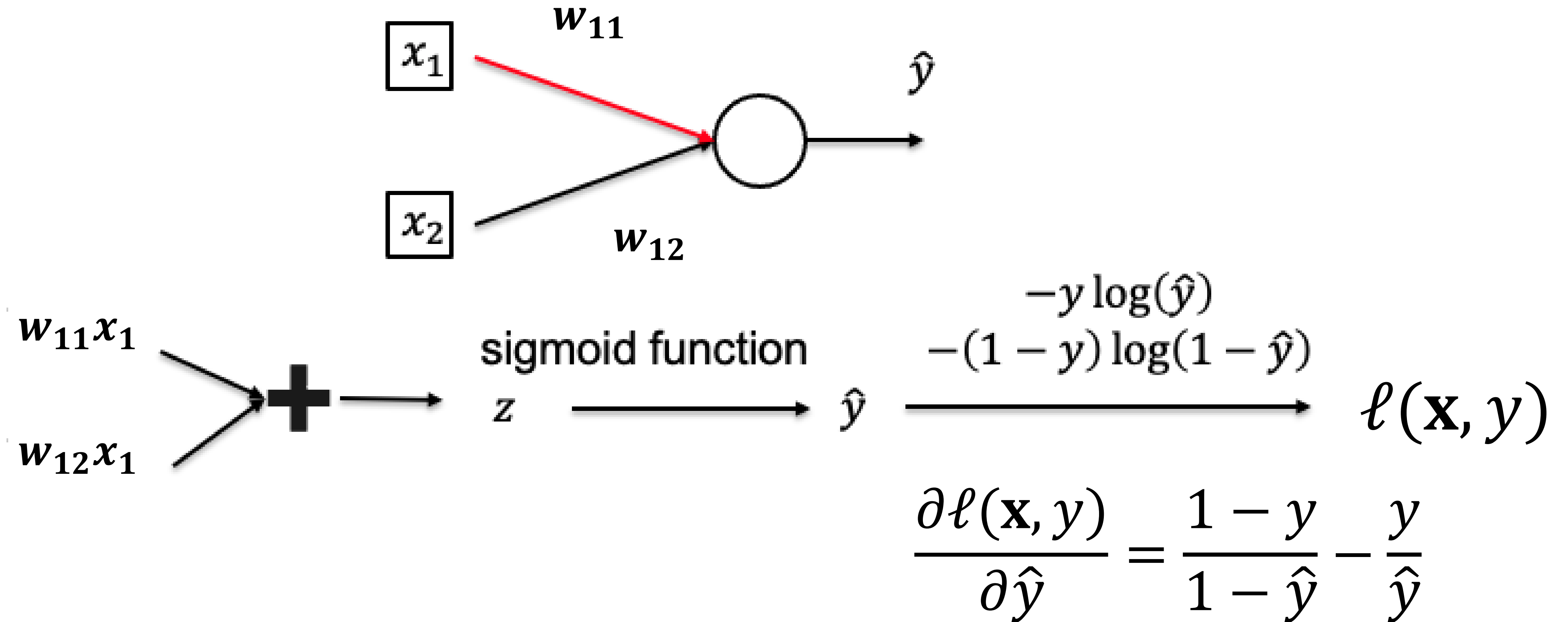
Use chain rule!

Calculate Gradient (on one data point)



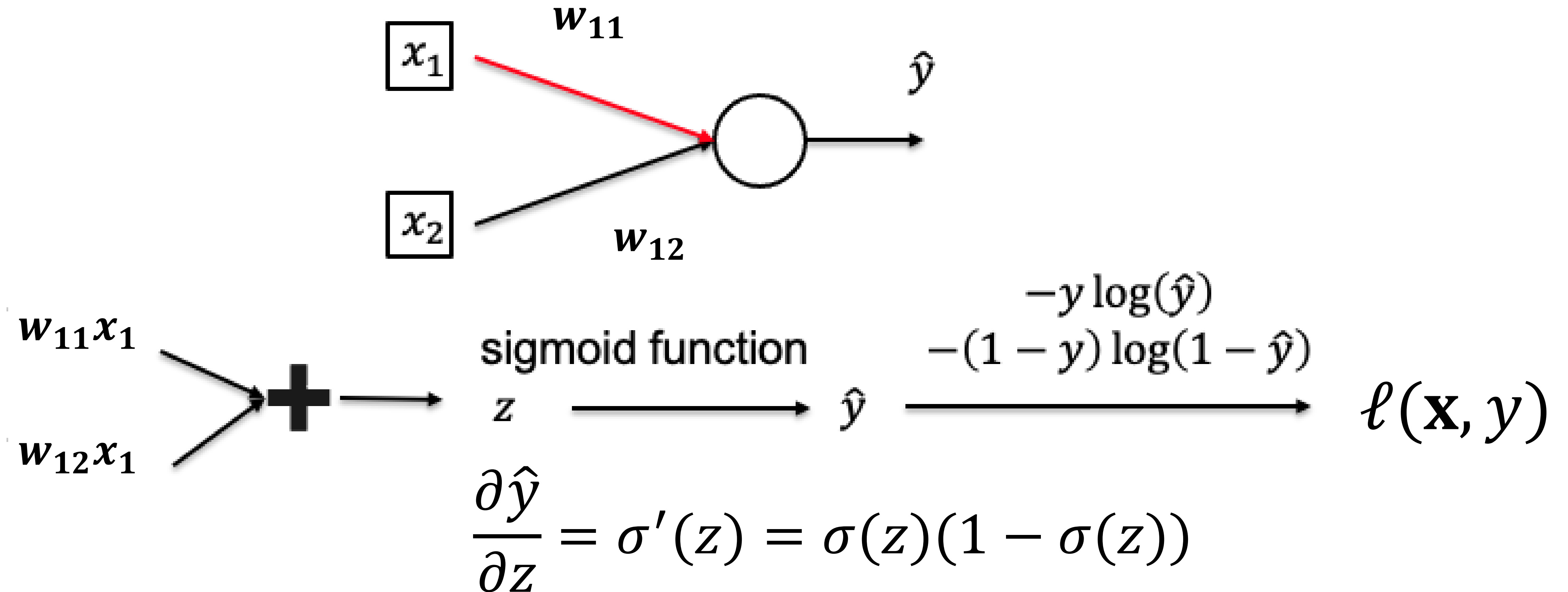
- By chain rule:
$$\frac{\partial \ell}{\partial w_{11}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_{11}}$$

Calculate Gradient (on one data point)



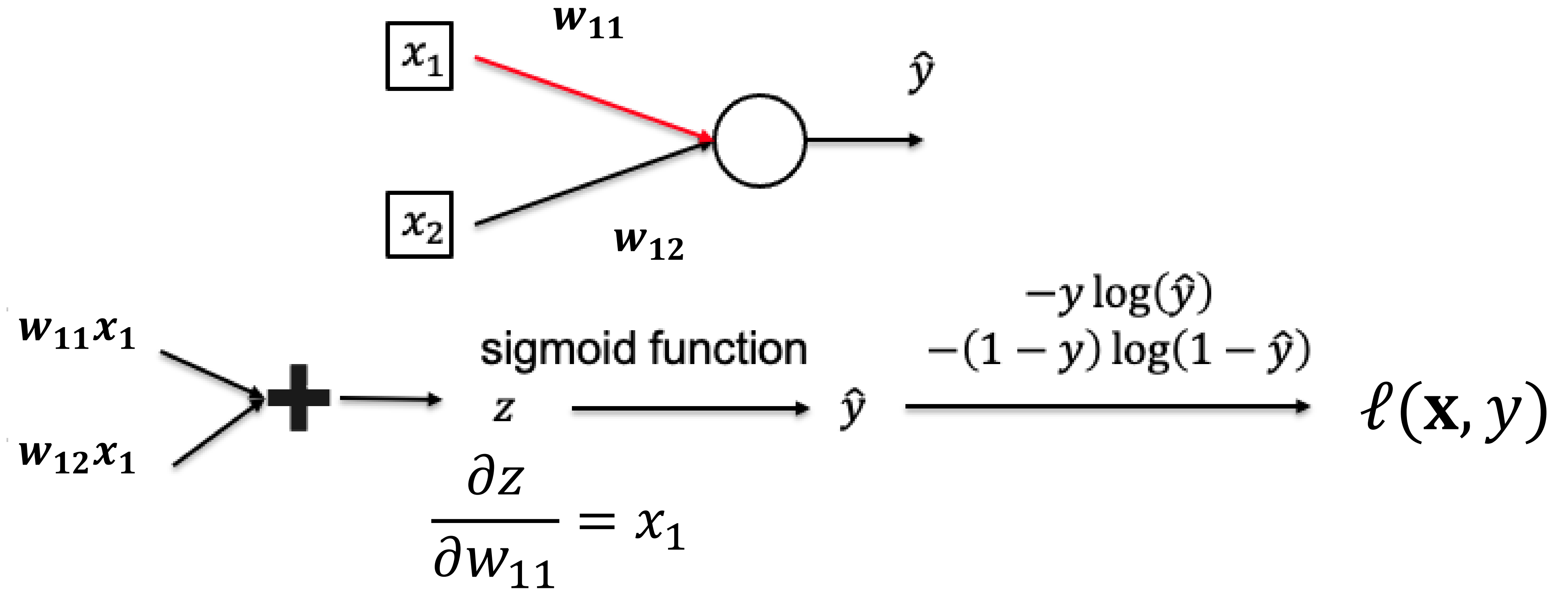
- By chain rule:
$$\frac{\partial \ell}{\partial w_{11}} = \left(\frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}} \right) \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_{11}}$$

Calculate Gradient (on one data point)



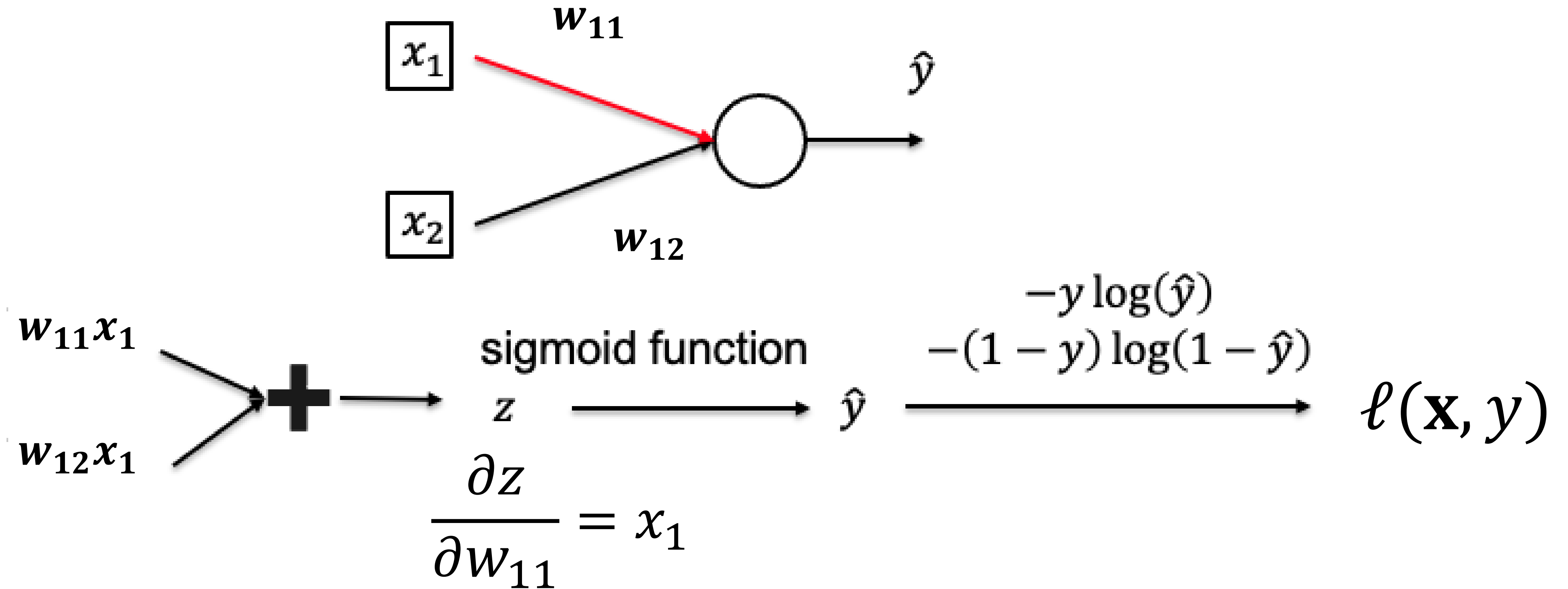
- By chain rule:
$$\frac{\partial \ell}{\partial w_{11}} = \left(\frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}} \right) (1 - \hat{y}) \frac{\partial z}{\partial w_{11}}$$

Calculate Gradient (on one data point)



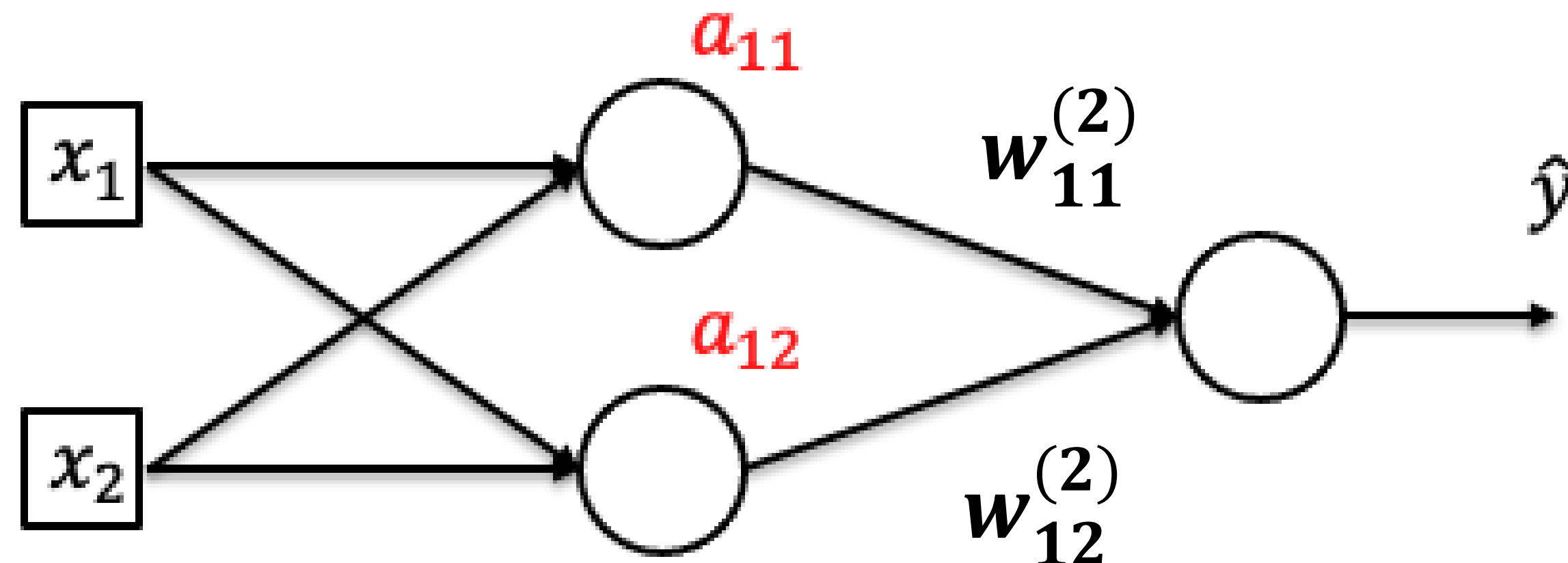
- By chain rule:
$$\frac{\partial l}{\partial w_{11}} = \left(\frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}} \right) (1 - \hat{y}) x_1$$

Calculate Gradient (on one data point)

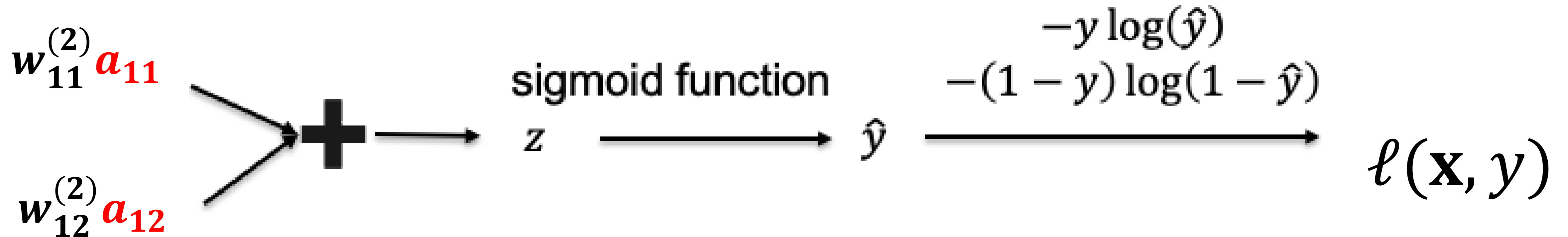


- By chain rule: $\frac{\partial l}{\partial w_{11}} = (\hat{y} - y)x_1$

Calculate Gradient (on one data point)

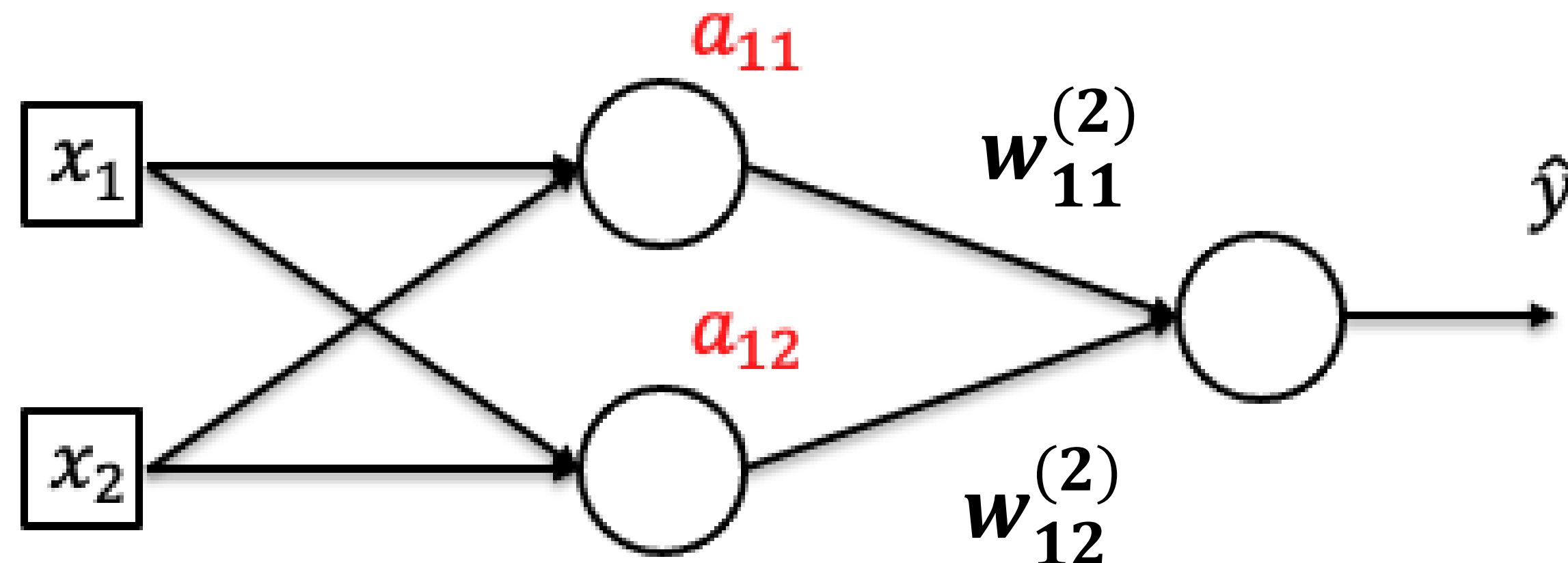


Make it deeper

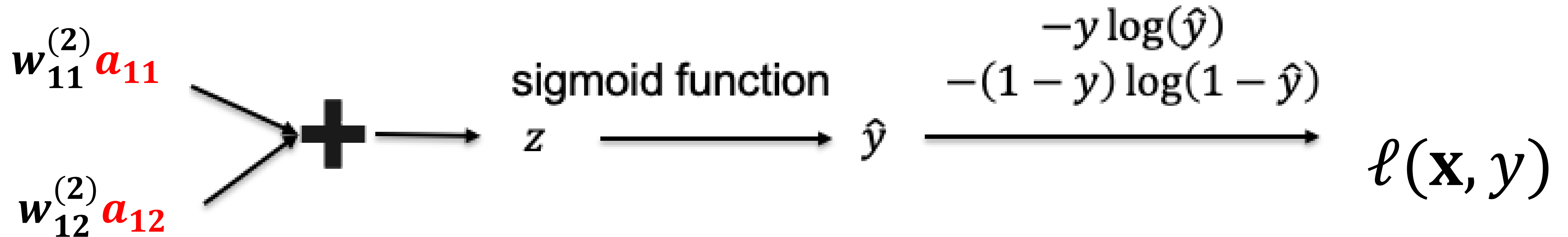


- By chain rule: $\frac{\partial l}{\partial a_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial a_{11}} = (\hat{y} - y) w_{11}^{(2)}$

Calculate Gradient (on one data point)

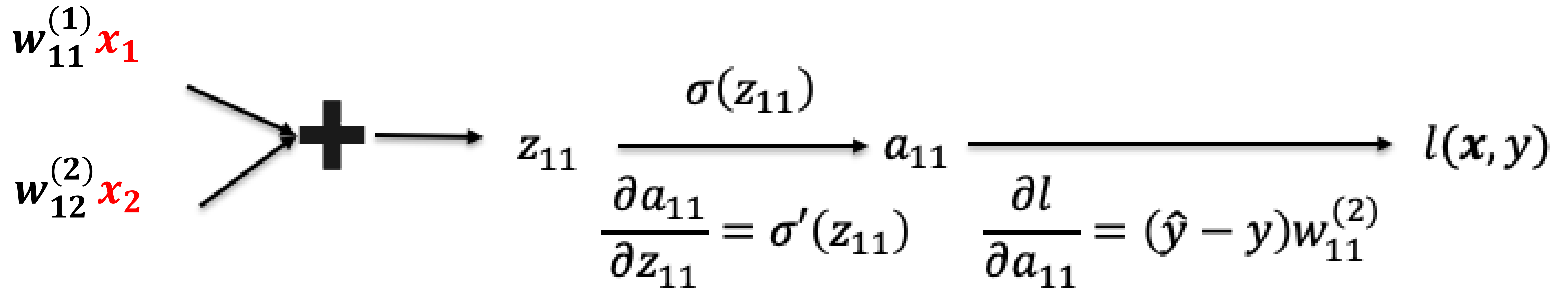
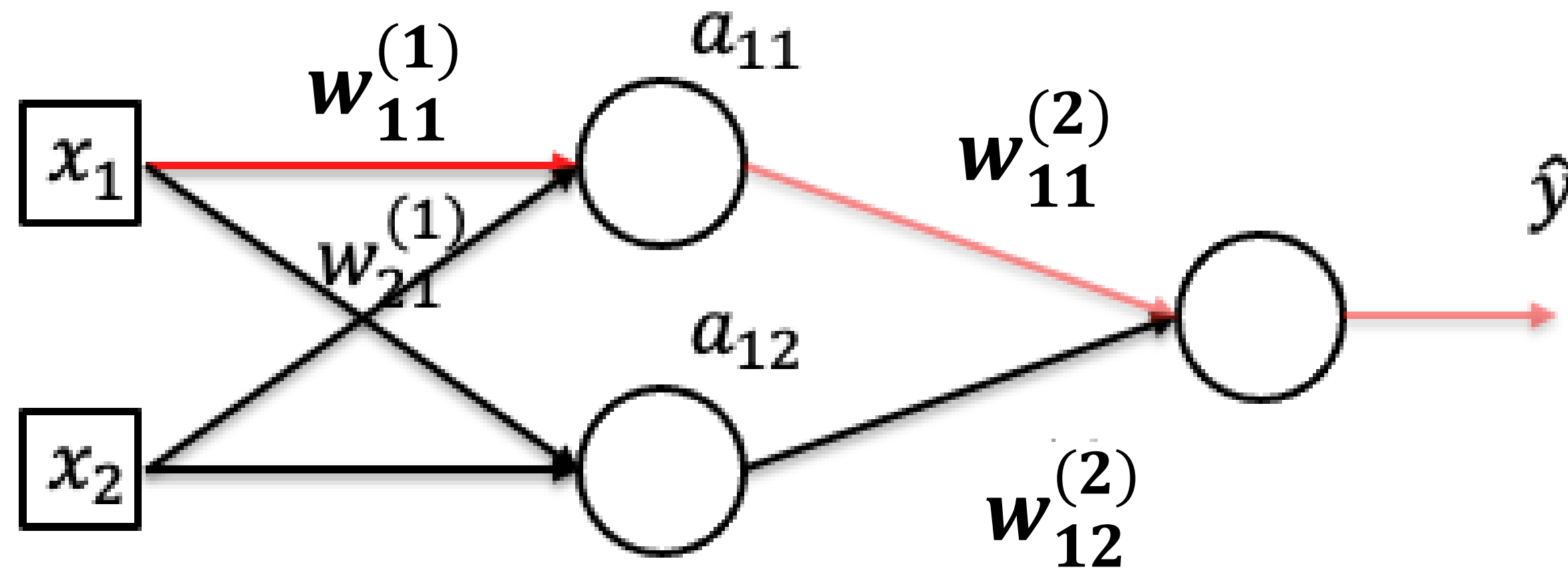


Make it deeper



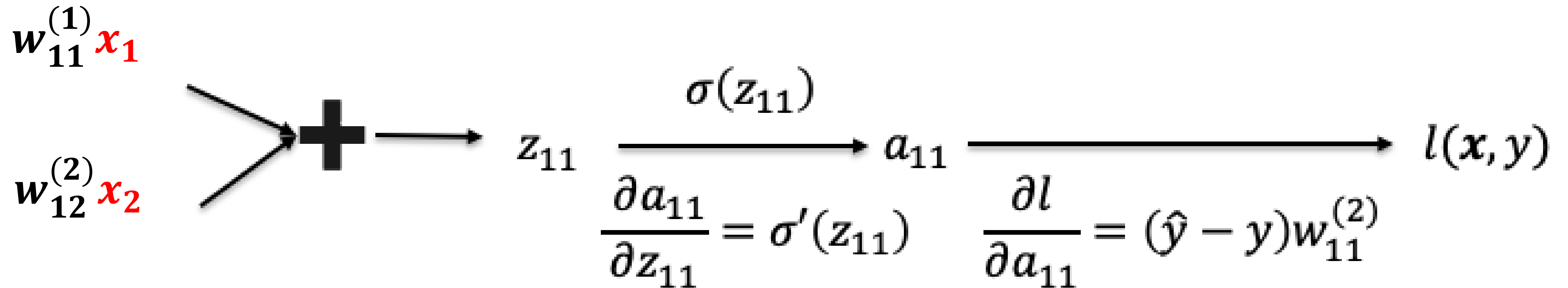
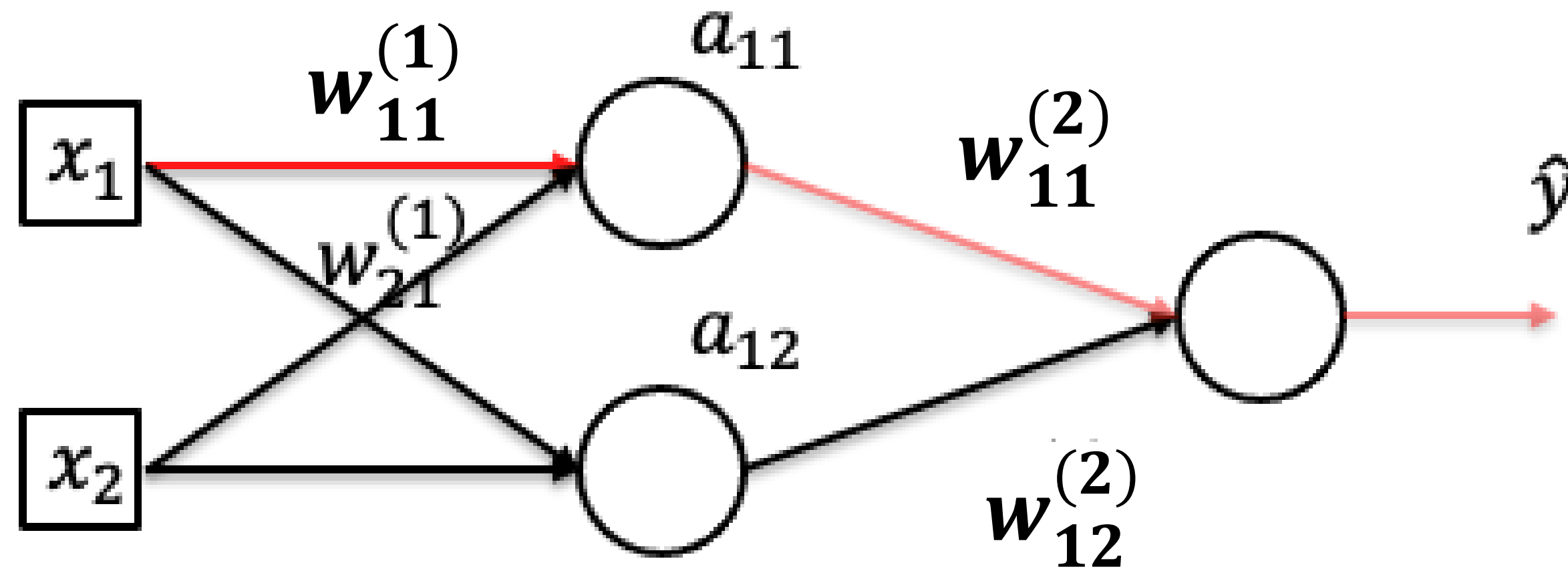
- By chain rule: $\frac{\partial l}{\partial a_{12}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial a_{12}} = (\hat{y} - y) w_{12}^{(2)}$

Calculate Gradient (on one data point)



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

Calculate Gradient (on one data point)



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

Auto Differentiation



Auto Differentiation (AD)

- In practice, back-propagation is handled by automatic differentiation
- AD evaluates gradients of a function specified by a program at given values
- AD is distinct from

```
In[1]:= D[4 x3 + x2 + 3, x]
```

```
Out[1]= 2 x + 12 x2
```

- Symbolic differentiation

- Numerical differentiation $\frac{\partial f(x)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$

What we've learned today...

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

Suggested reading

- Textbook: Artificial Intelligence: A Modern Approach (4th edition). Stuart Russell and Peter Norvig. Pearson, 2020.
 - Sections 21 Introduction, 21.1, 21.2, 21.4 ,21.5