



CS 540 Introduction to Artificial Intelligence Neural Networks (II)

Sharon Yixuan Li
University of Wisconsin-Madison

March 9, 2021

Announcement

- HW3 grade released last Friday (<https://piazza.com/class/kk1k70vbawp3ts?cid=551>)
- HW6 is going out today, due on **Friday March 19**
- Extended deadline of HW6 (due to midterm)

Today's outline

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

Review: Perceptron

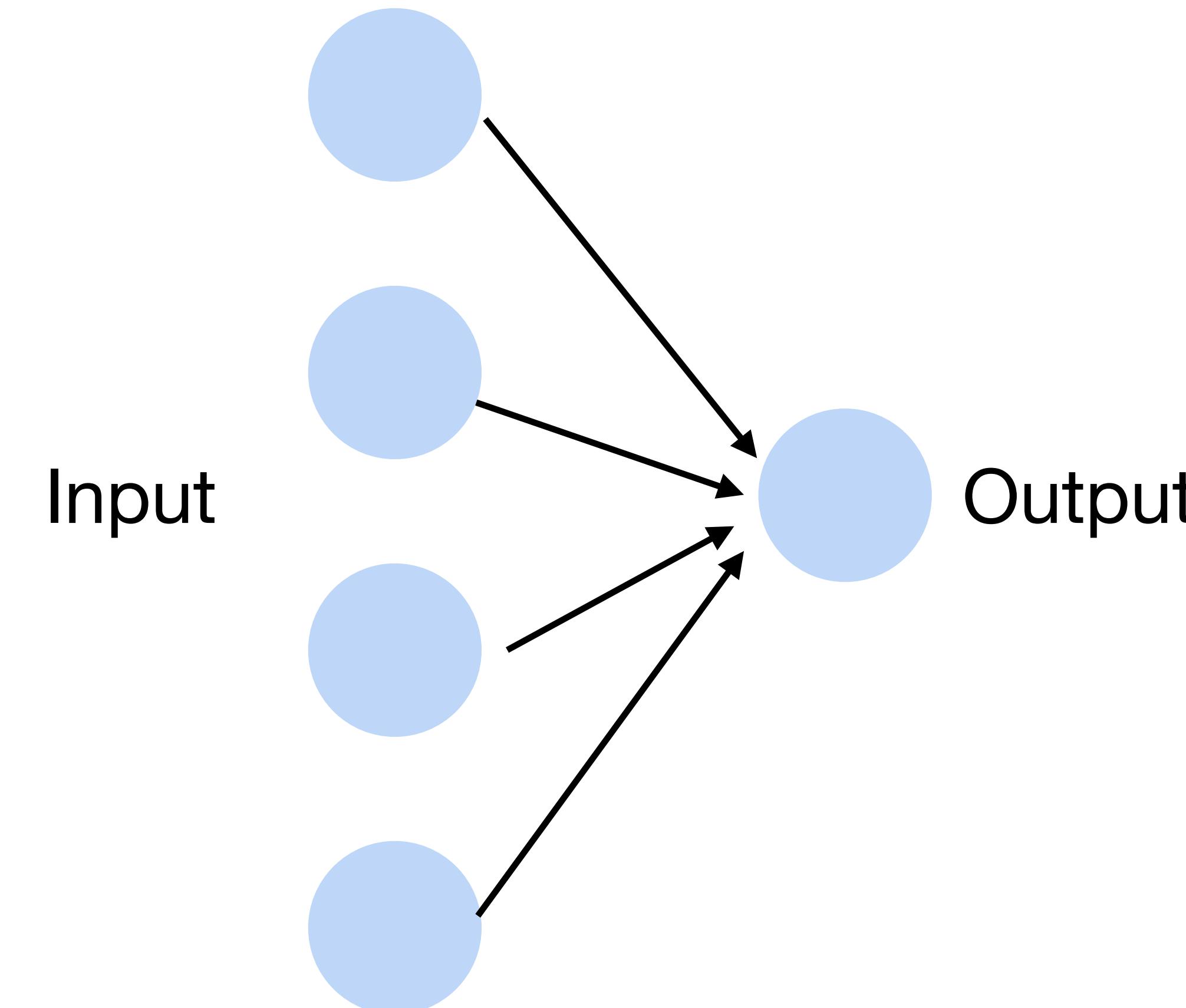
- Given input \mathbf{x} , weight \mathbf{w} and bias b , perceptron outputs:

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

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

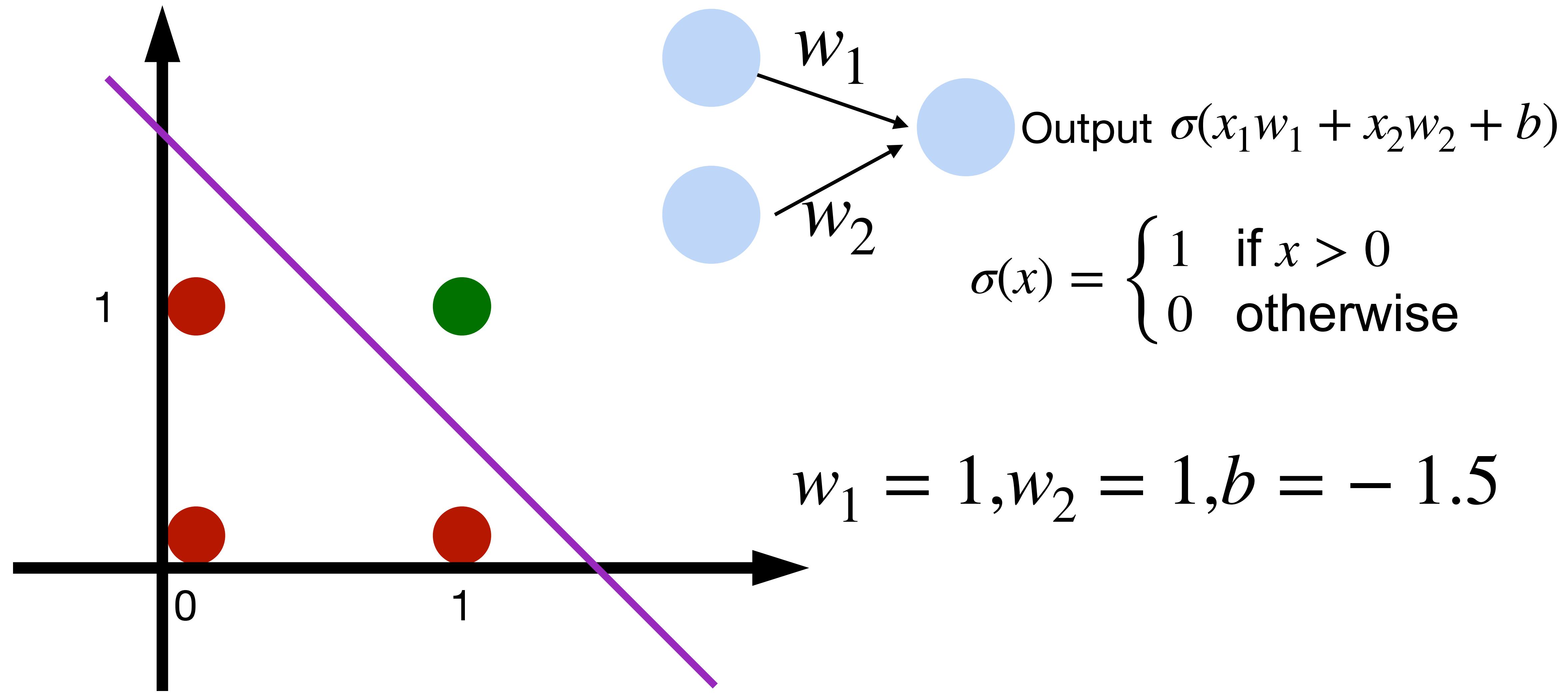
Activation function

Cats vs. dogs?



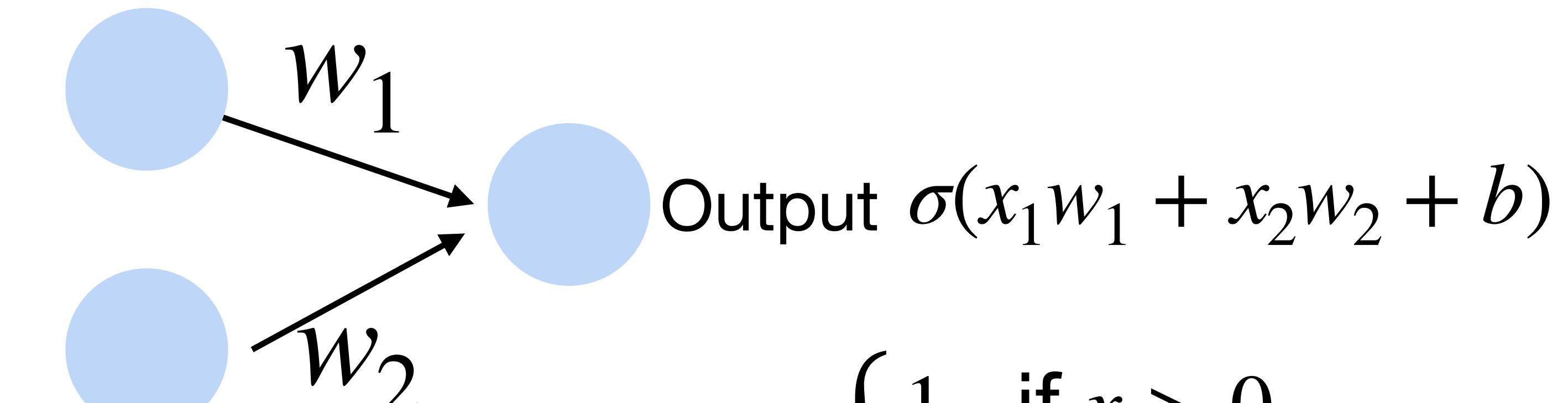
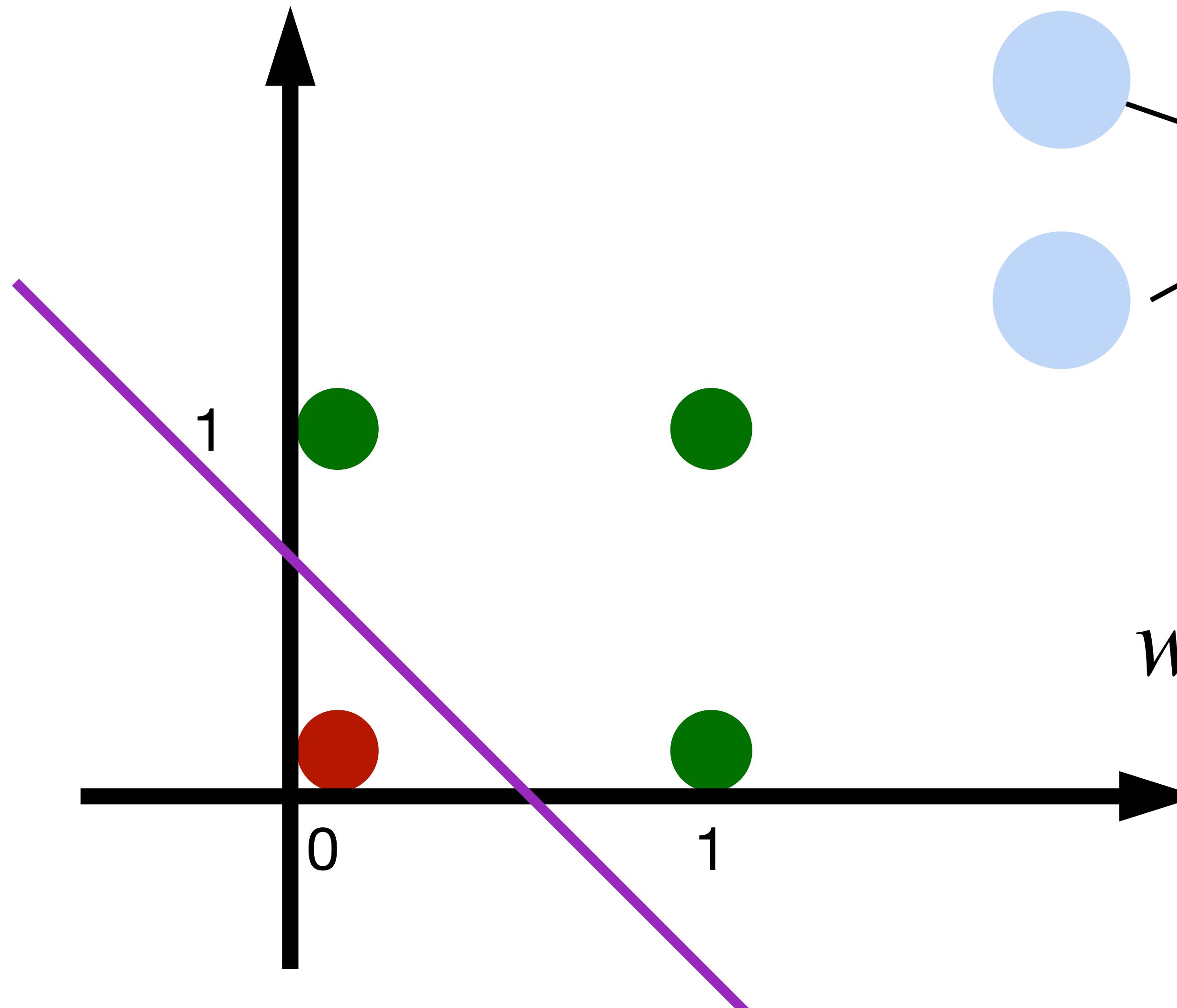
Learning AND function using perceptron

The perceptron can learn an AND function



Learning OR function using perceptron

The perceptron can learn an OR function

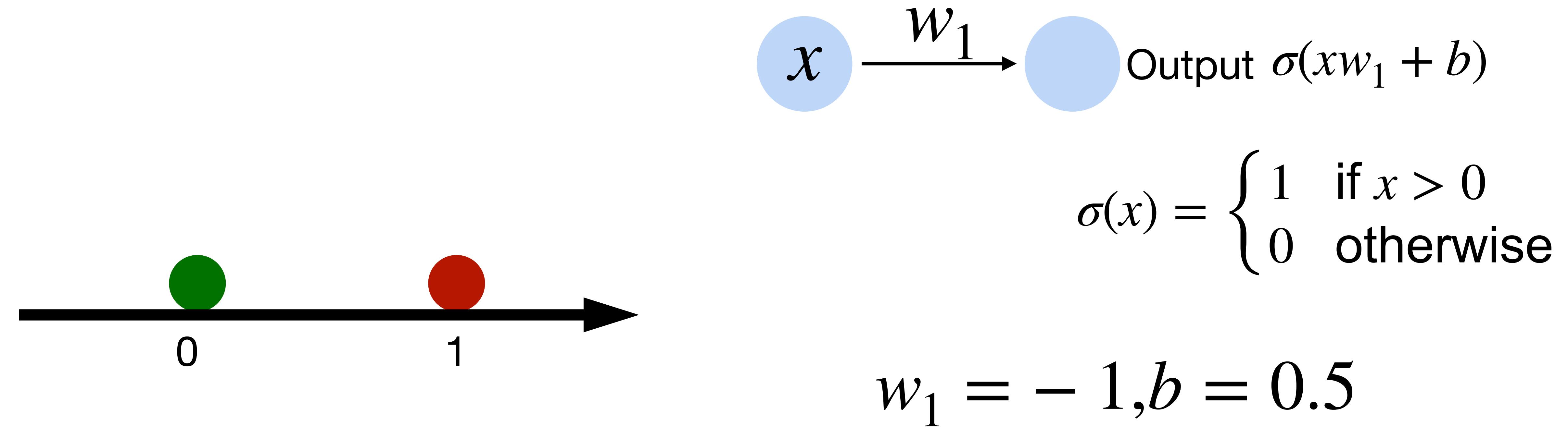


$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

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

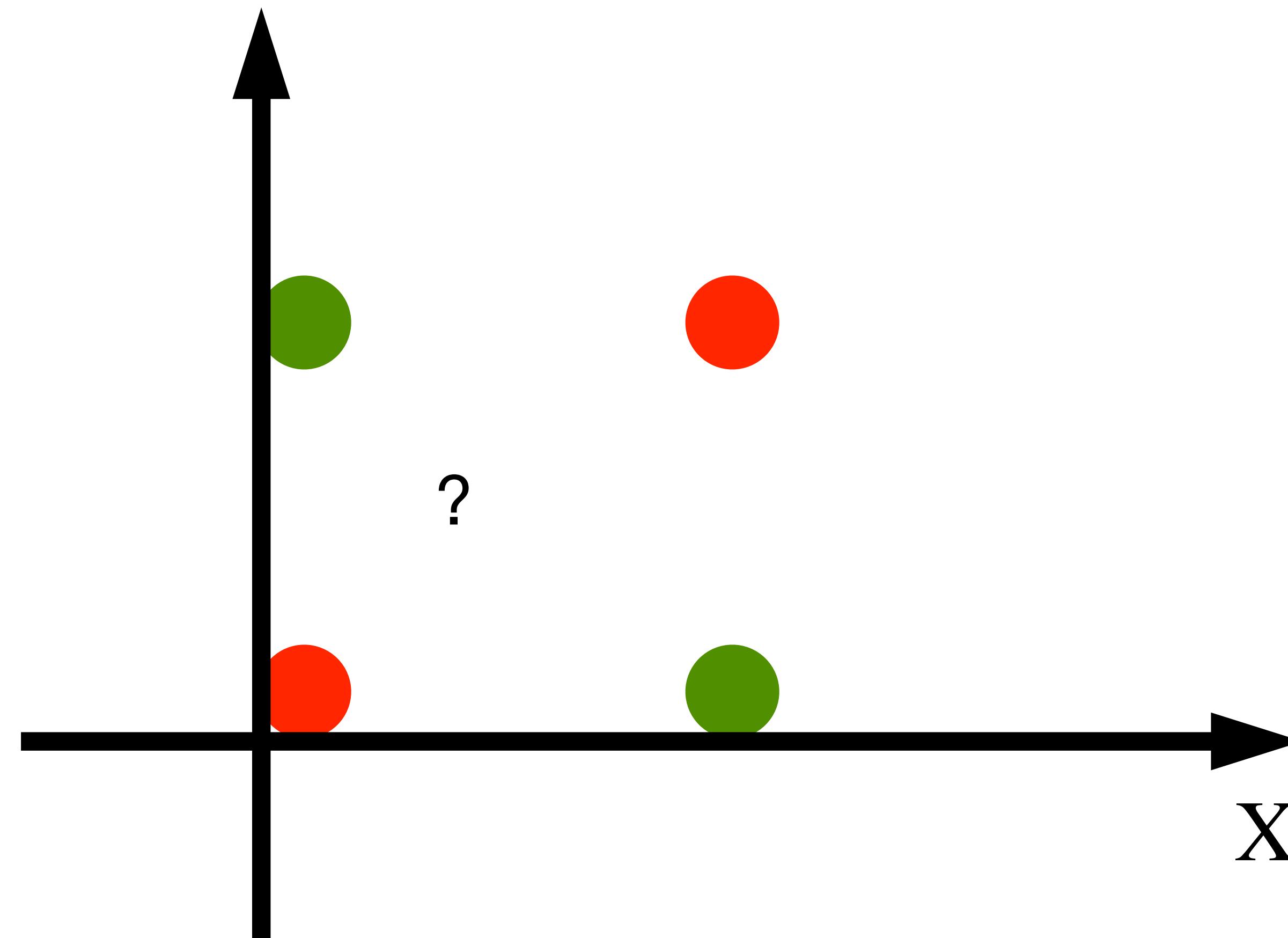
Learning NOT function using perceptron

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)



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

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

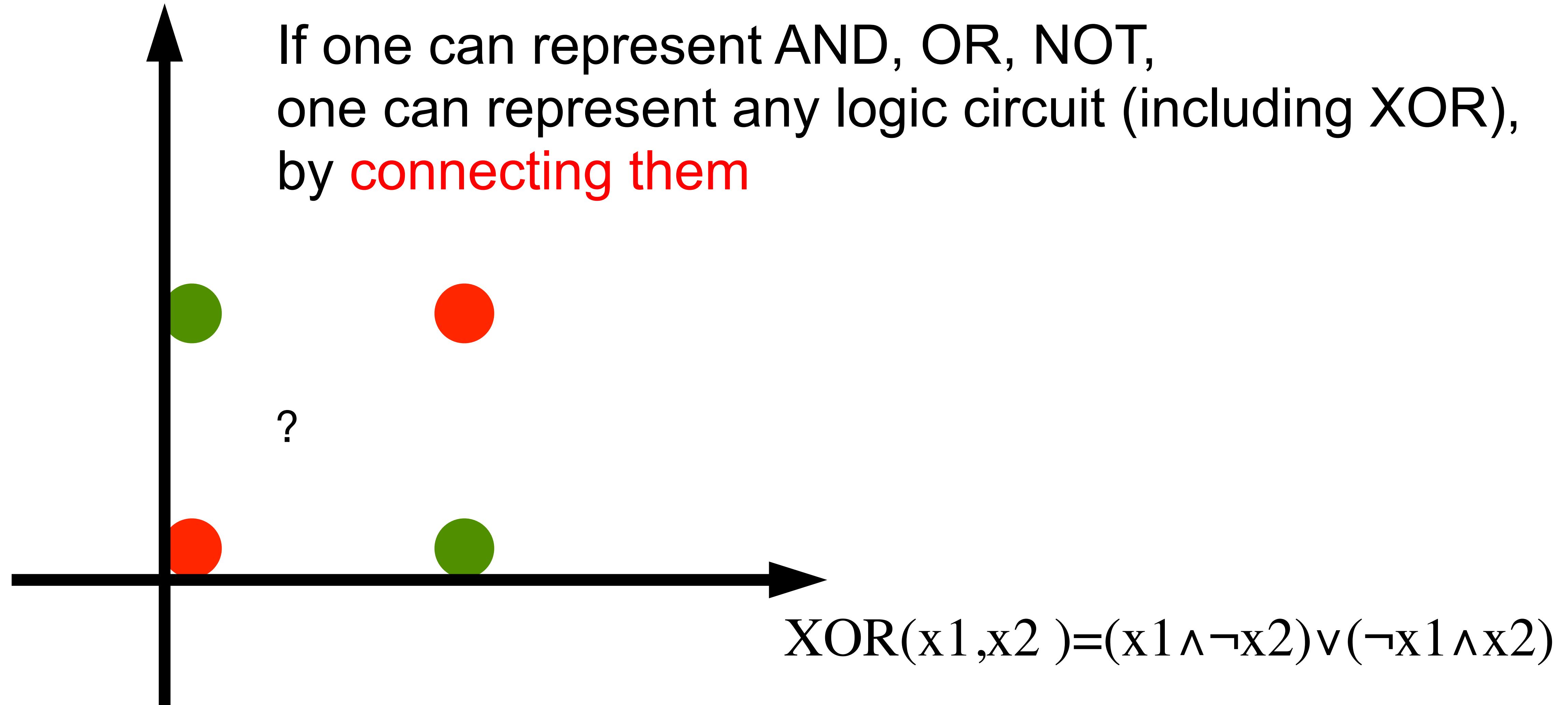
$$x_1 = 0, x_2 = 1, y = 1$$

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

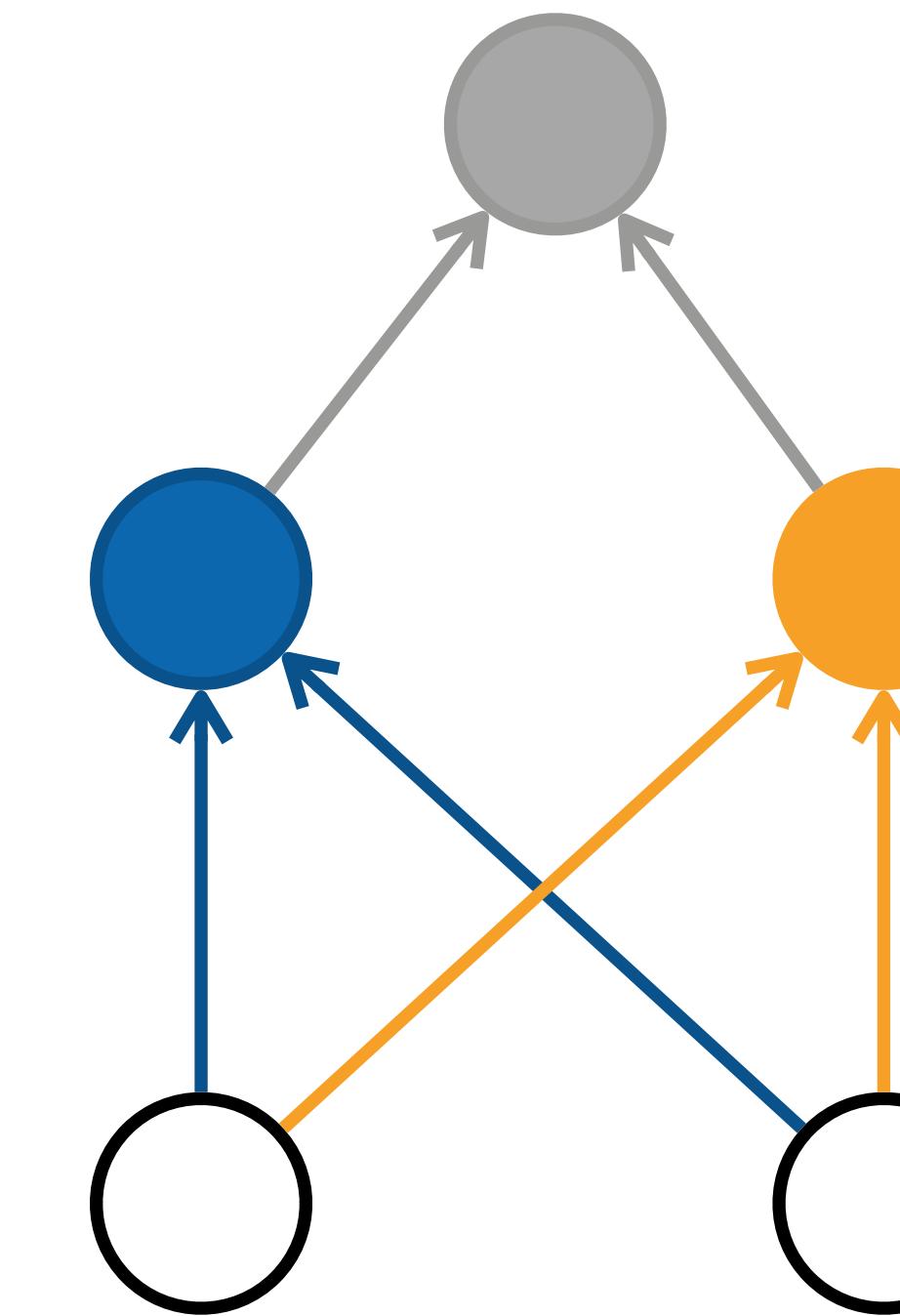
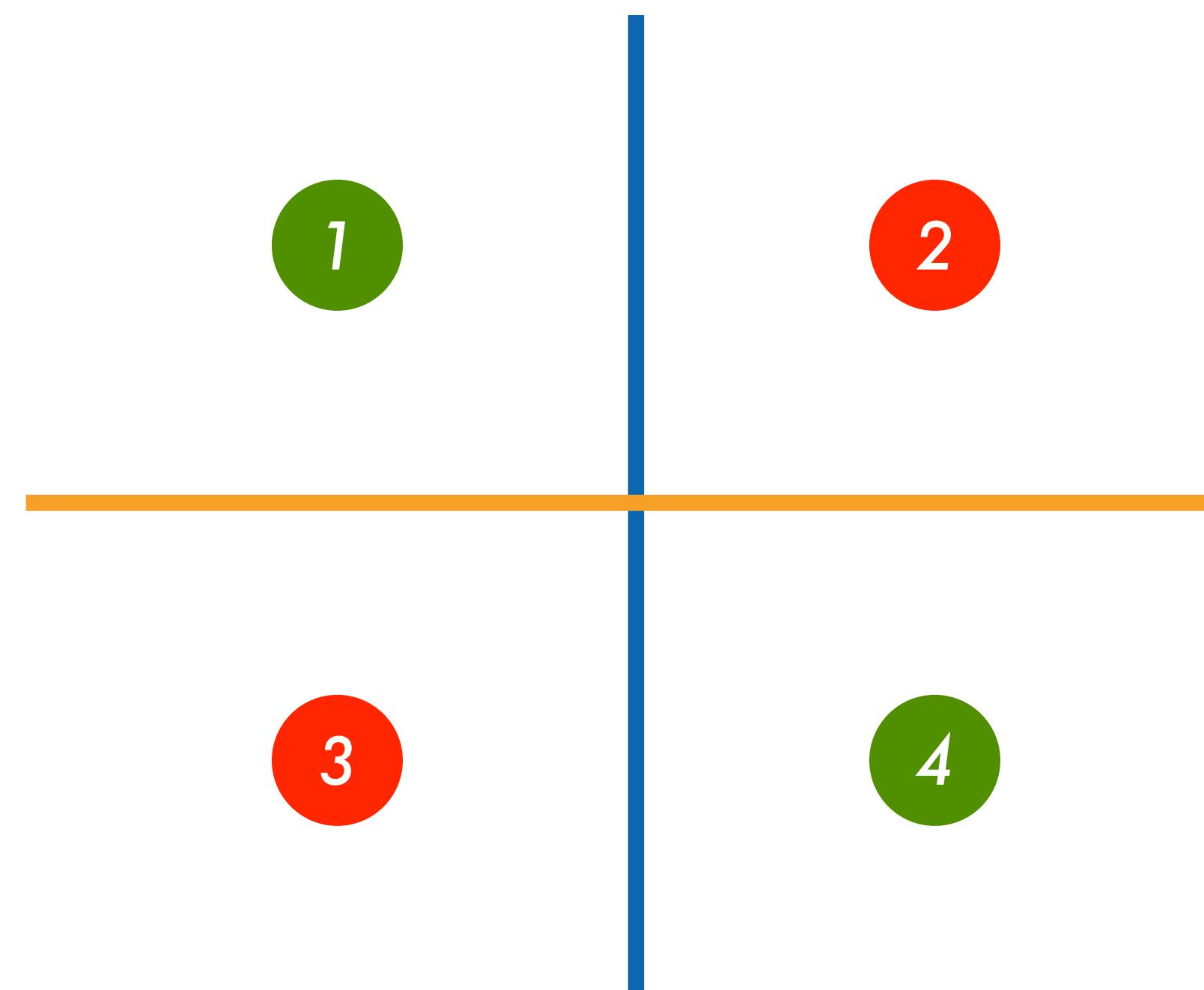
$$\text{XOR}(x_1, x_2) = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$

The limited power of a single neuron

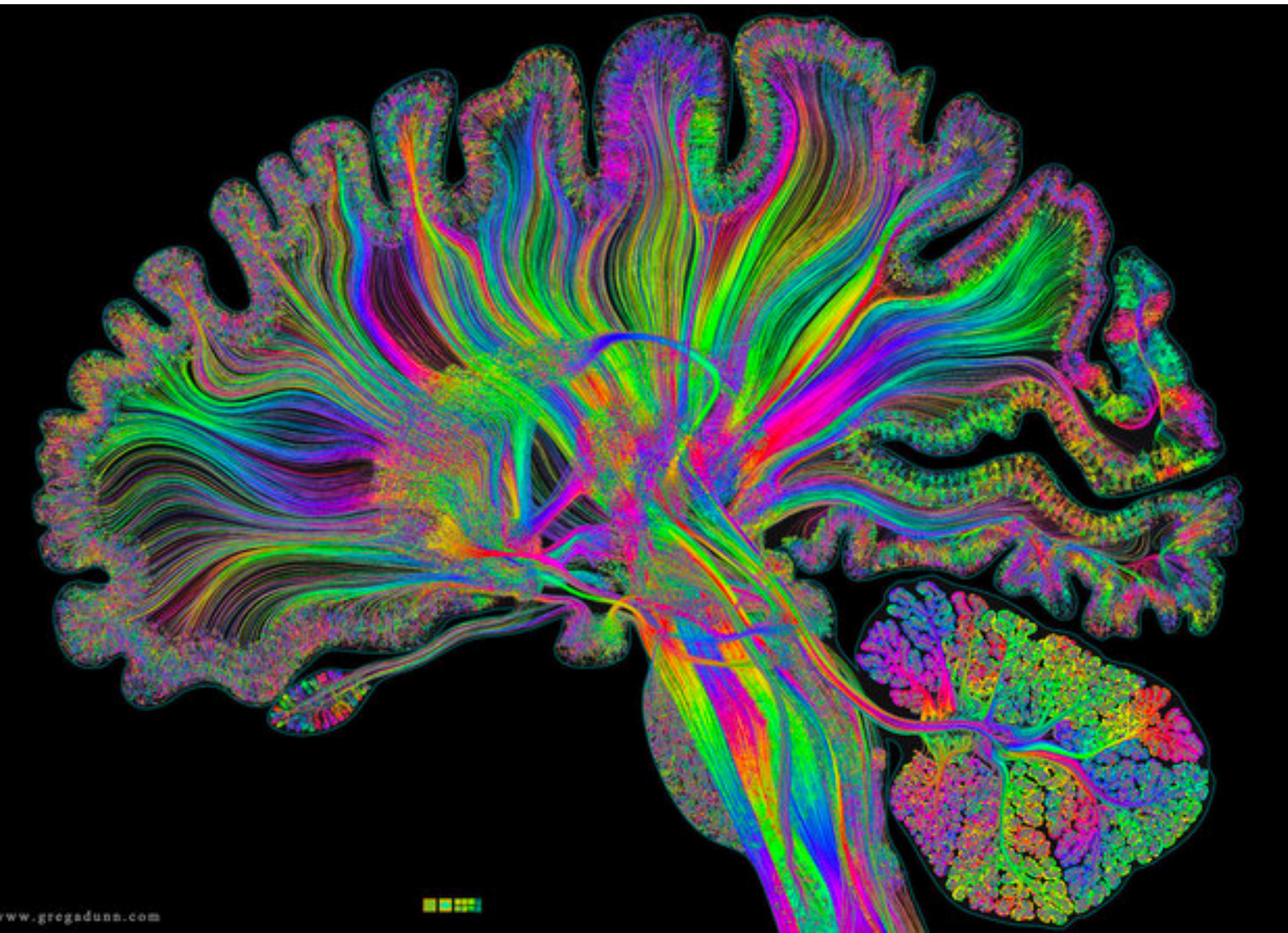
XOR problem



Learning XOR

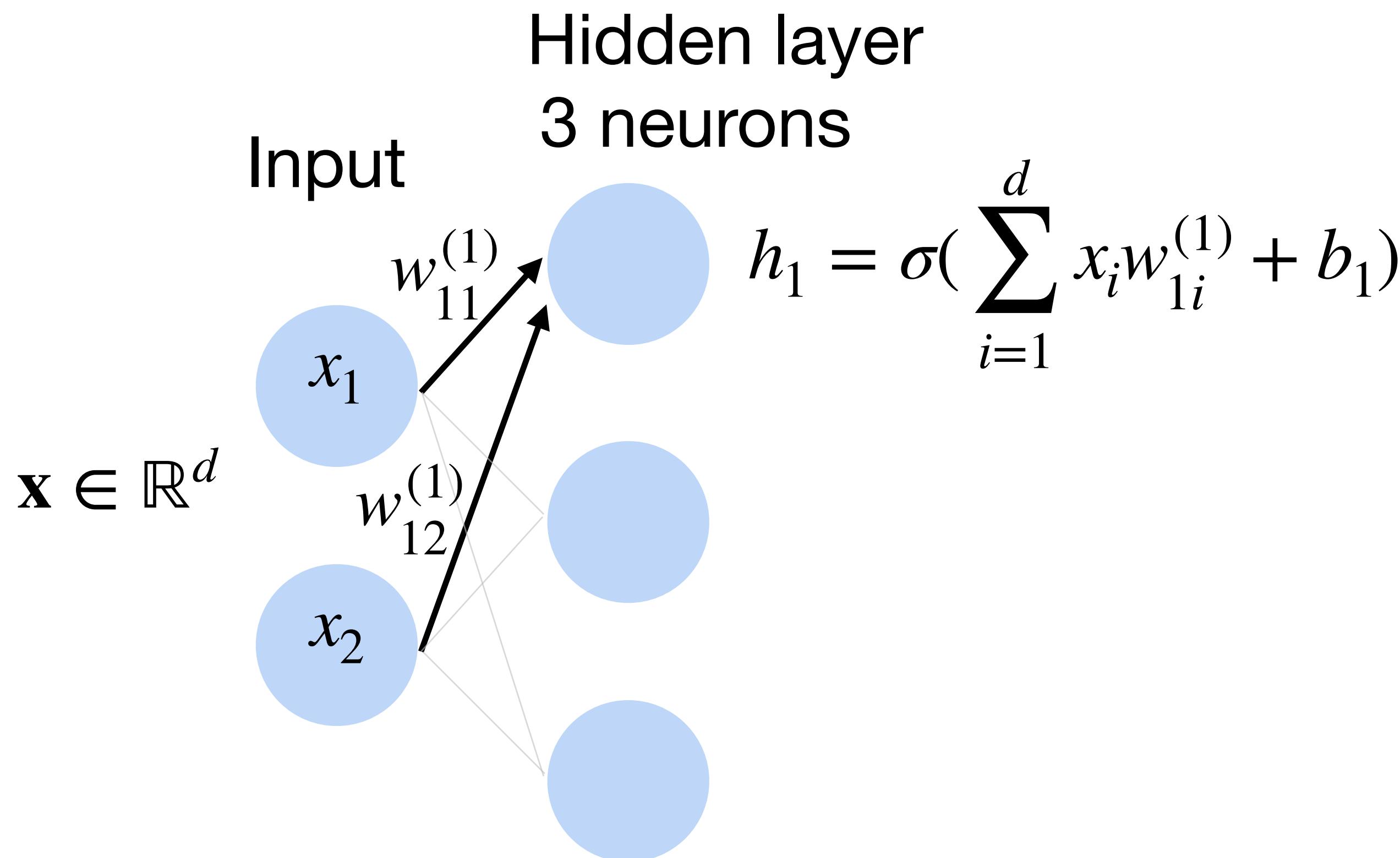


Multilayer Perceptron



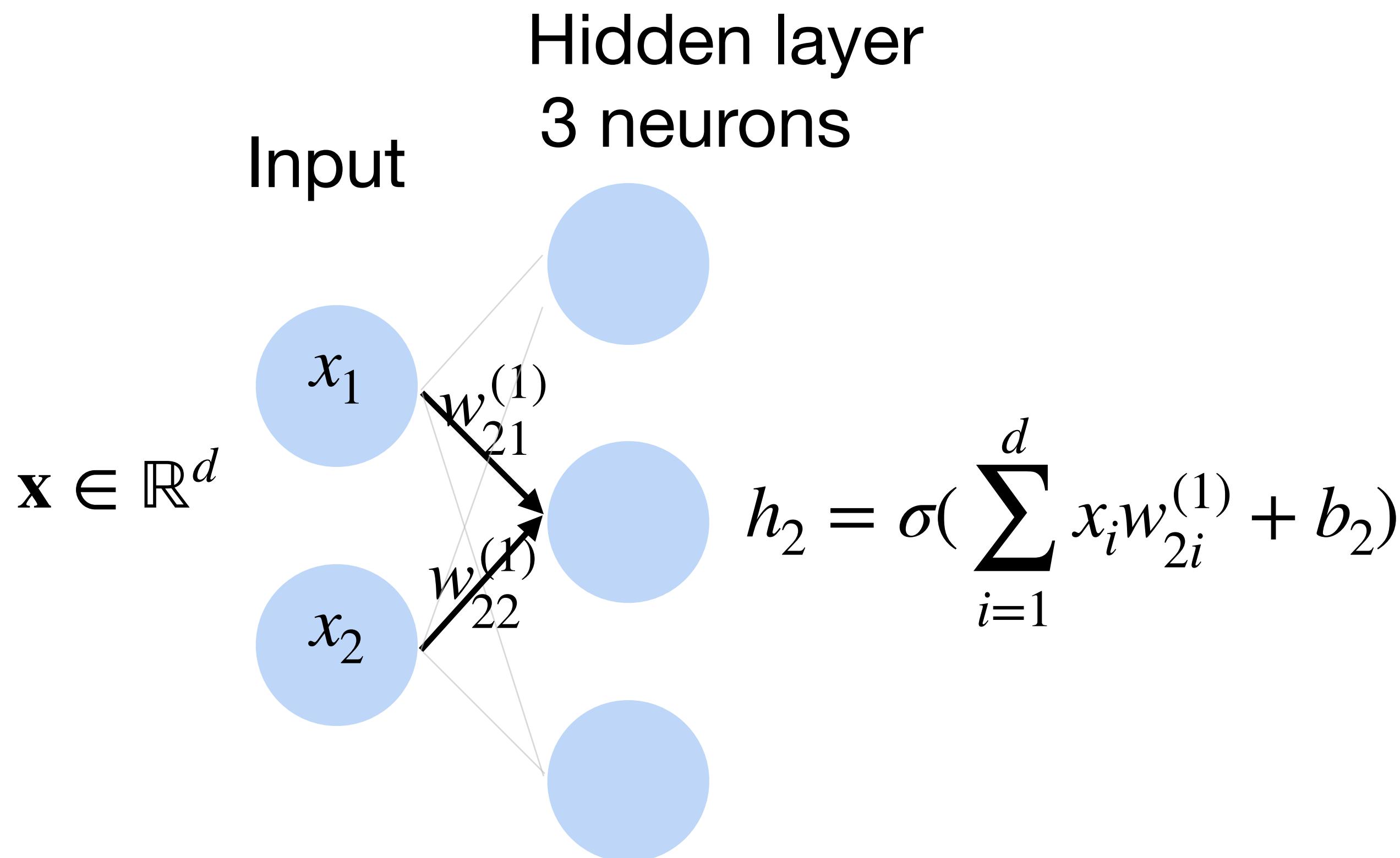
Multi-layer perceptron: Example

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



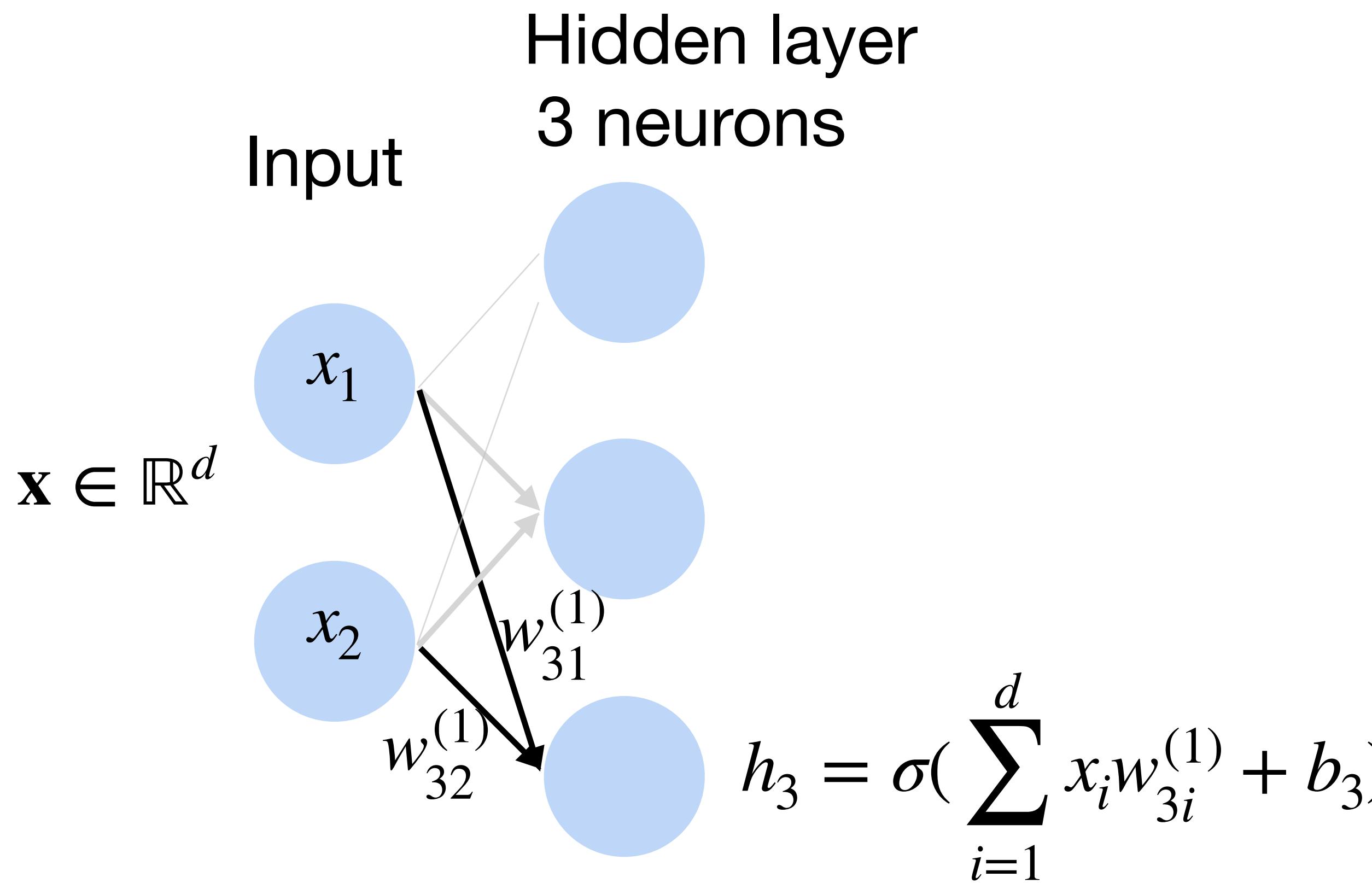
Multi-layer perceptron: Example

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



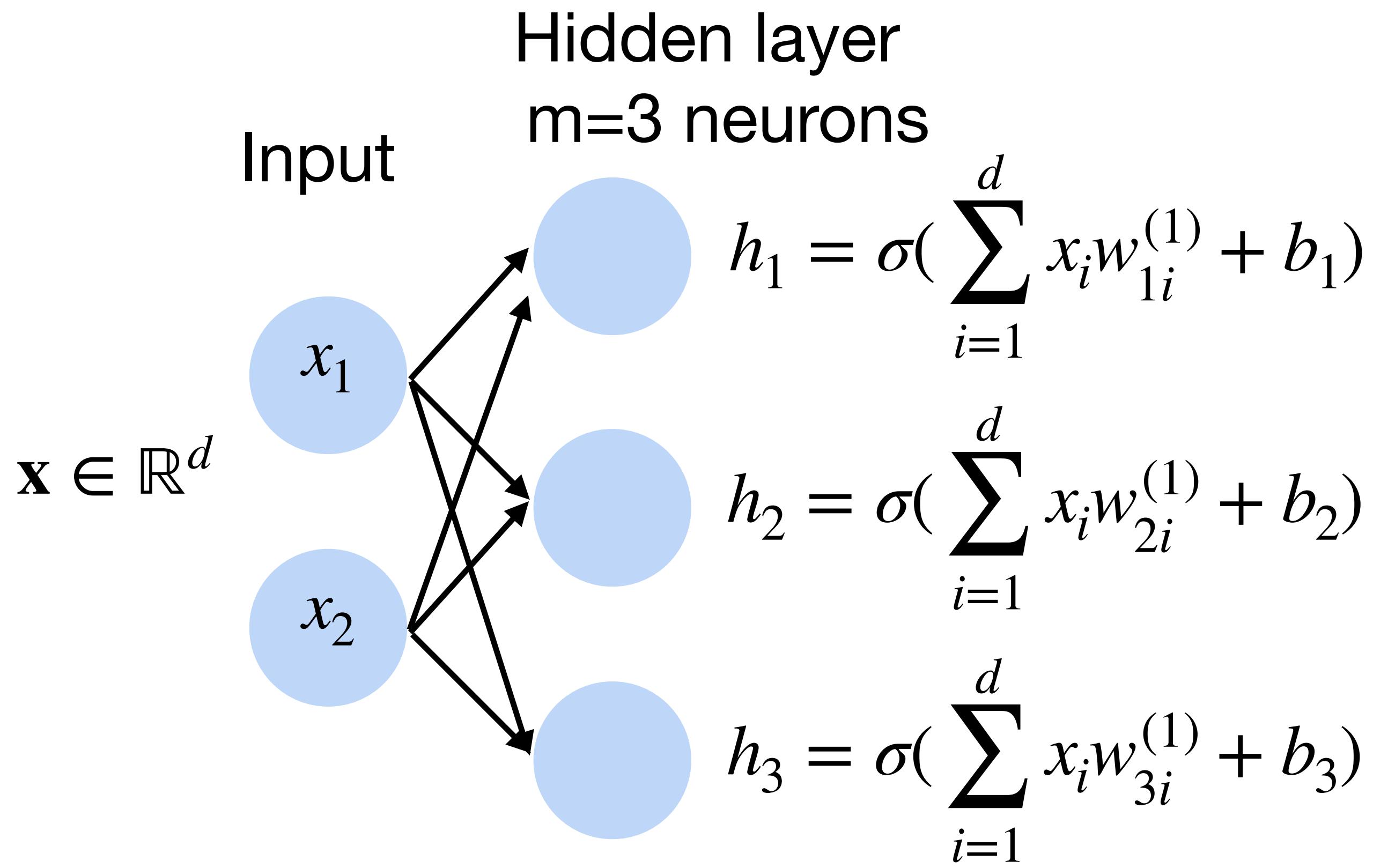
Multi-layer perceptron: Example

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



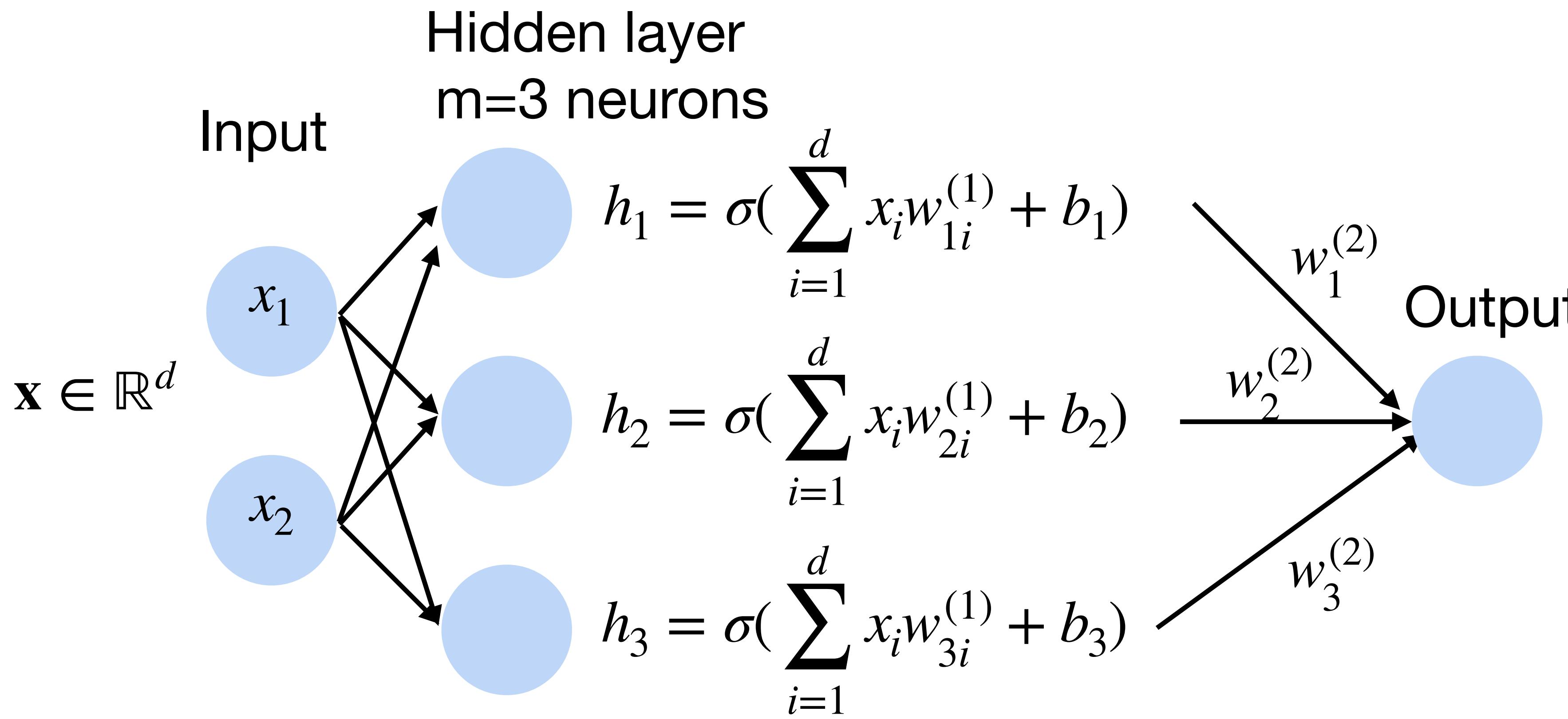
Multi-layer perceptron: Example

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



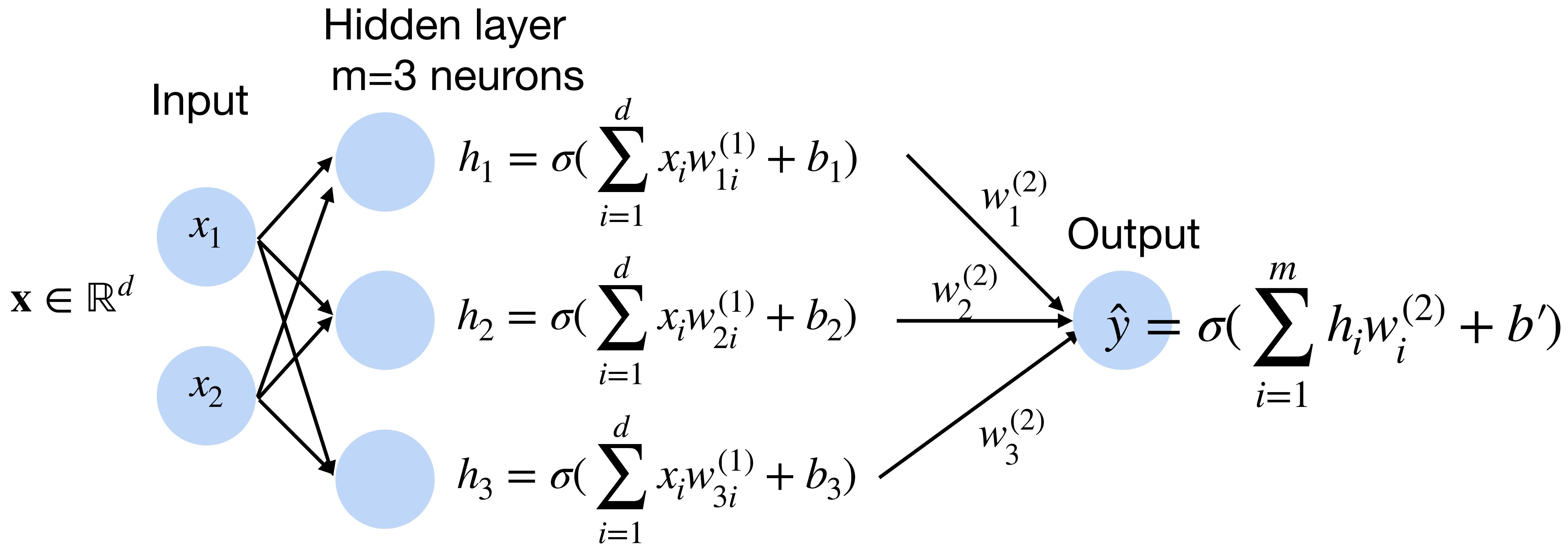
Multi-layer perceptron: Example

- 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

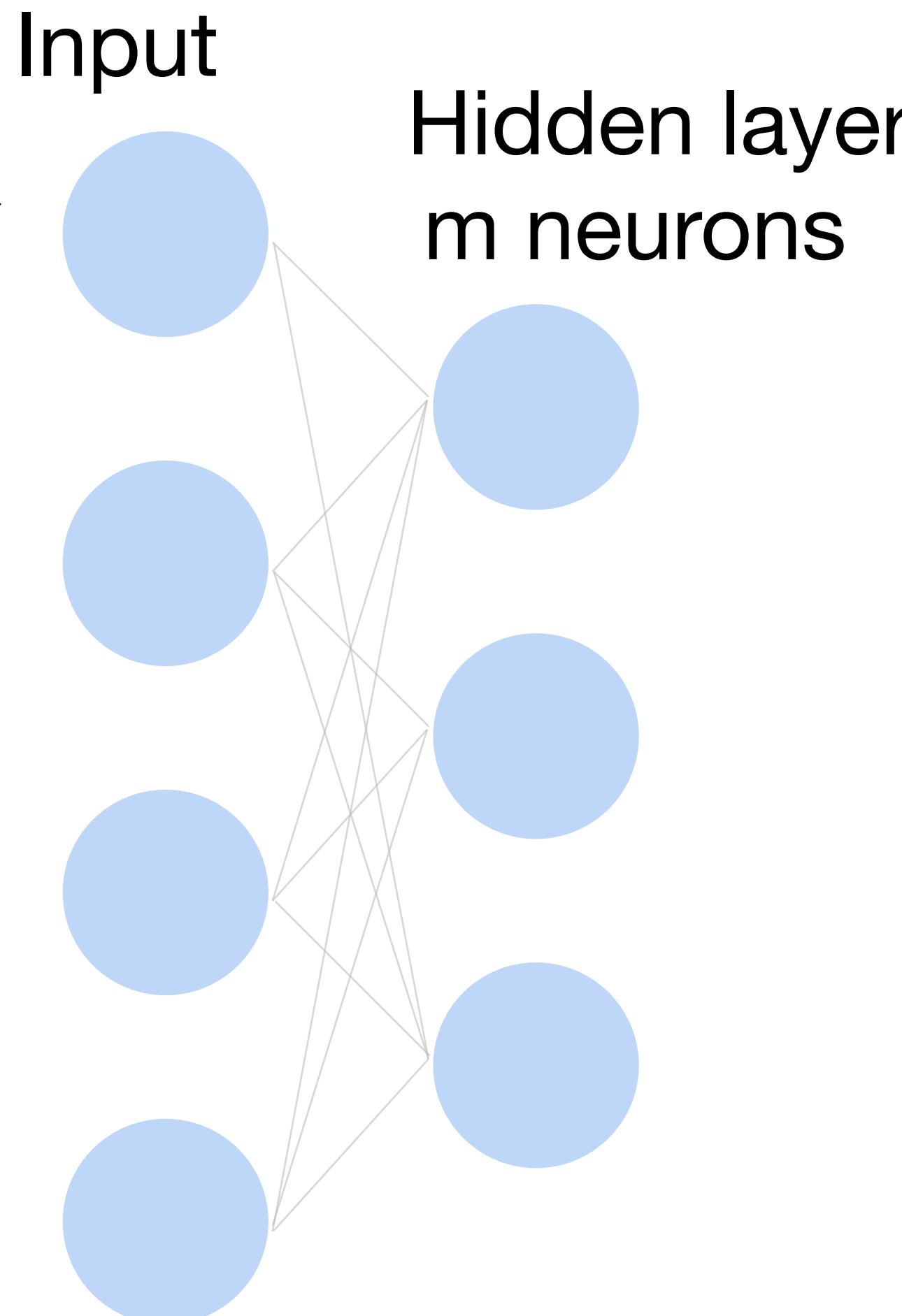


Multi-layer perceptron: Matrix Notation

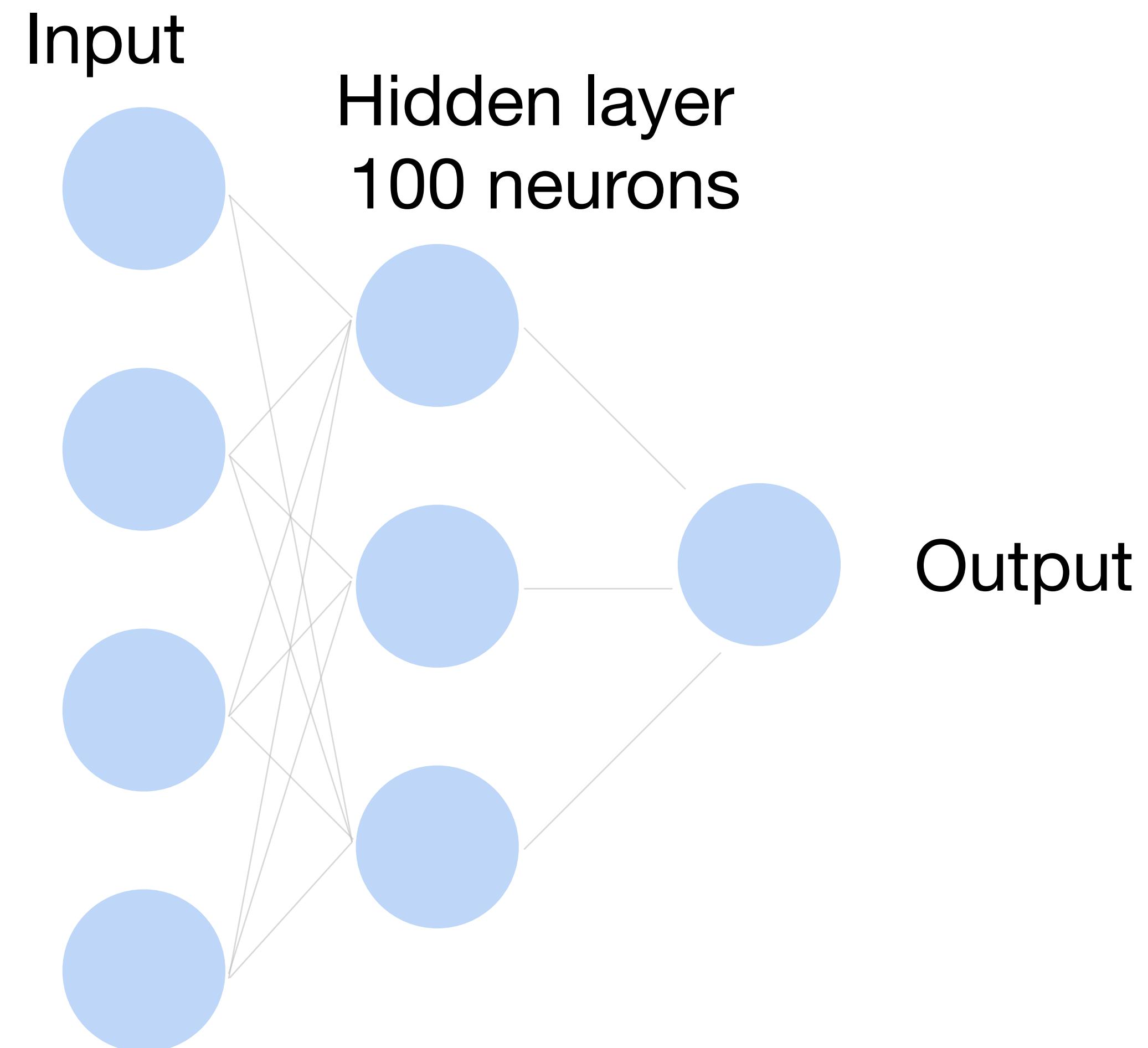
- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$
- Intermediate output

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

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

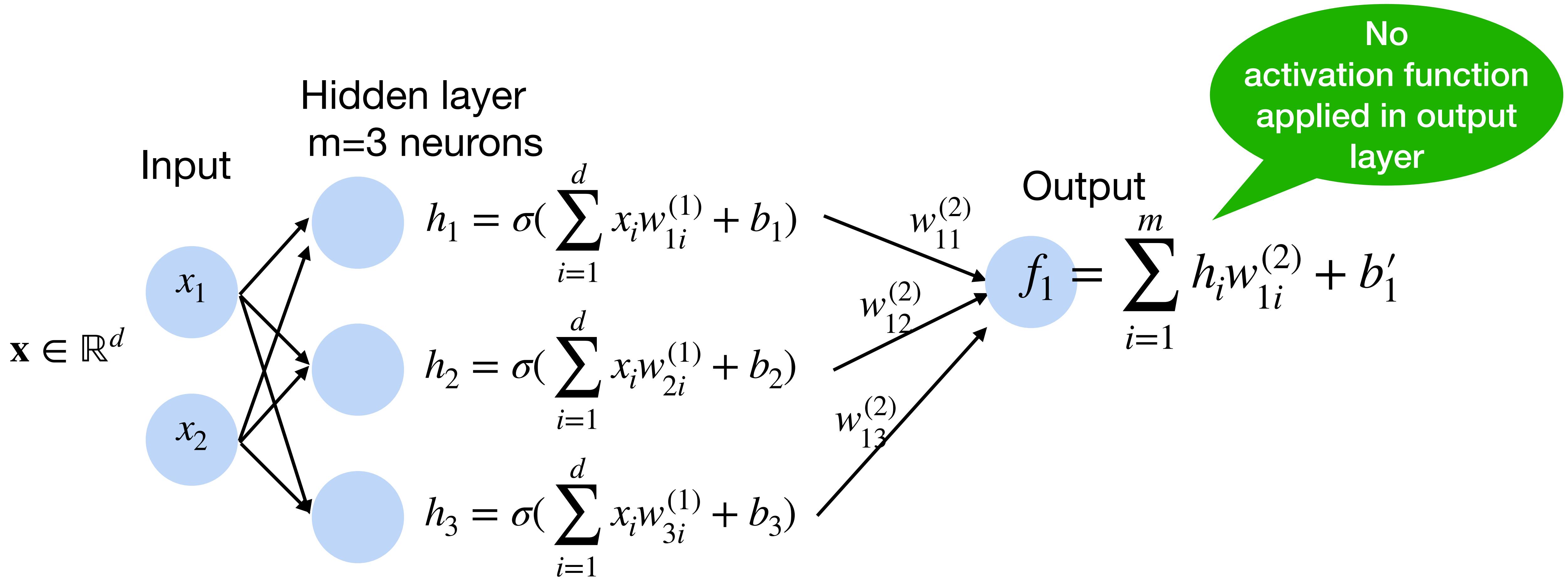


Classify cats vs. dogs



Neural network for k-way classification

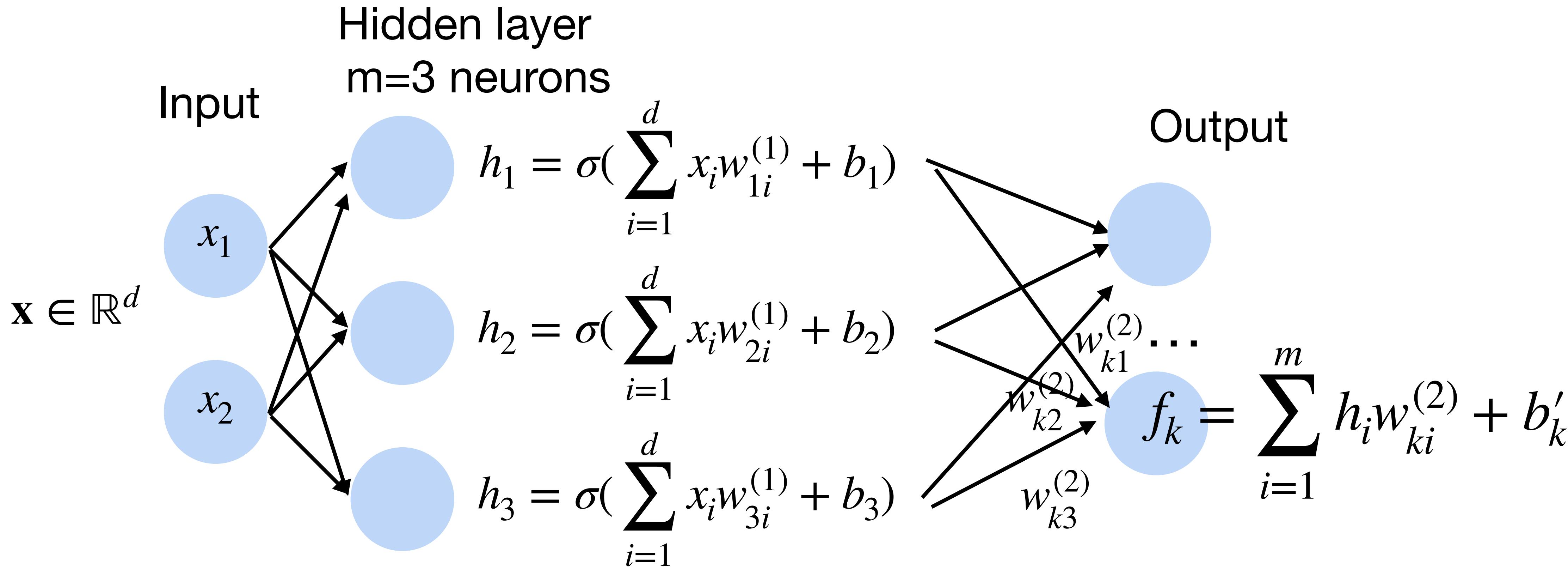
- K outputs in the final layer



Neural network for k-way classification

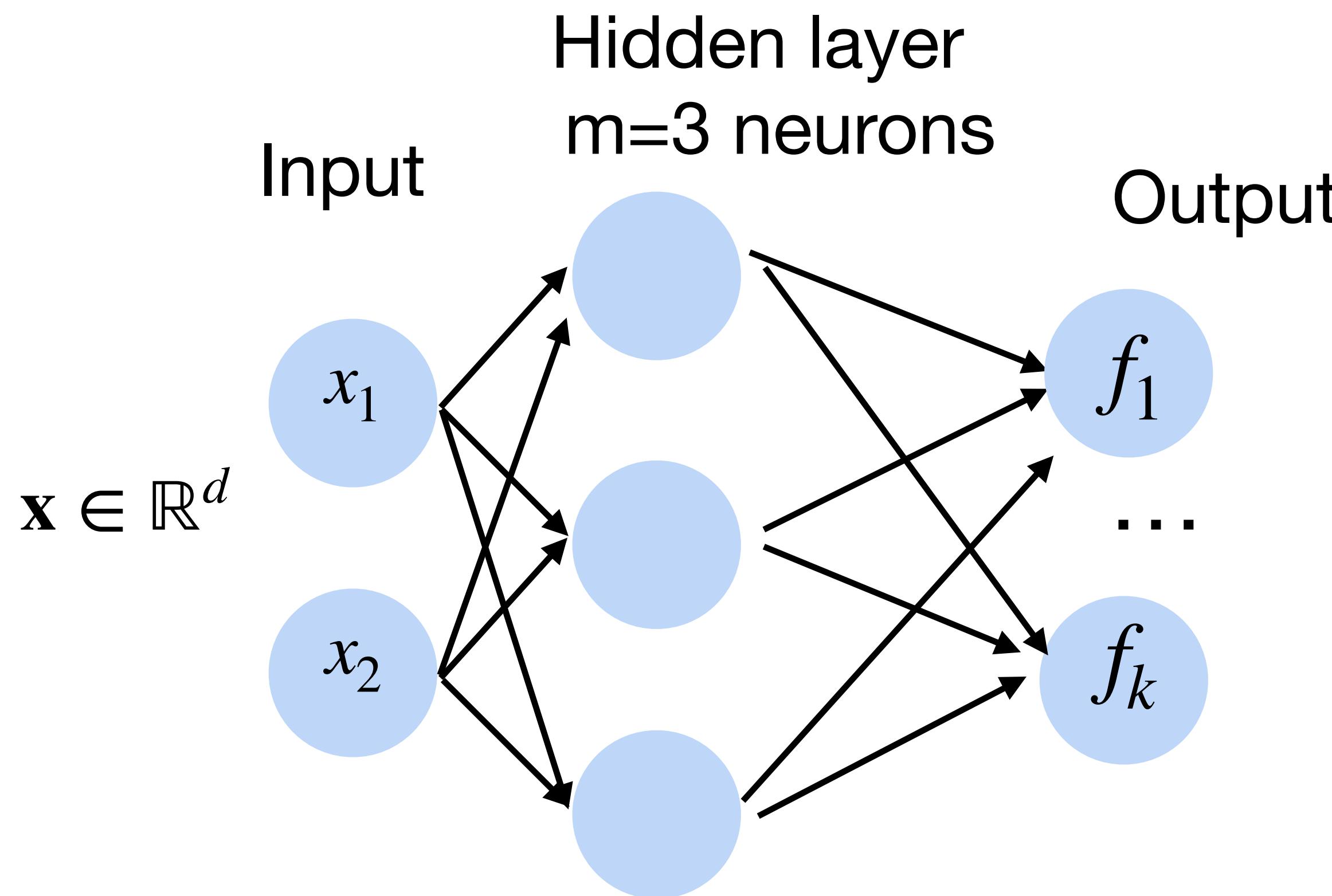
- K outputs units in the final layer

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



Softmax

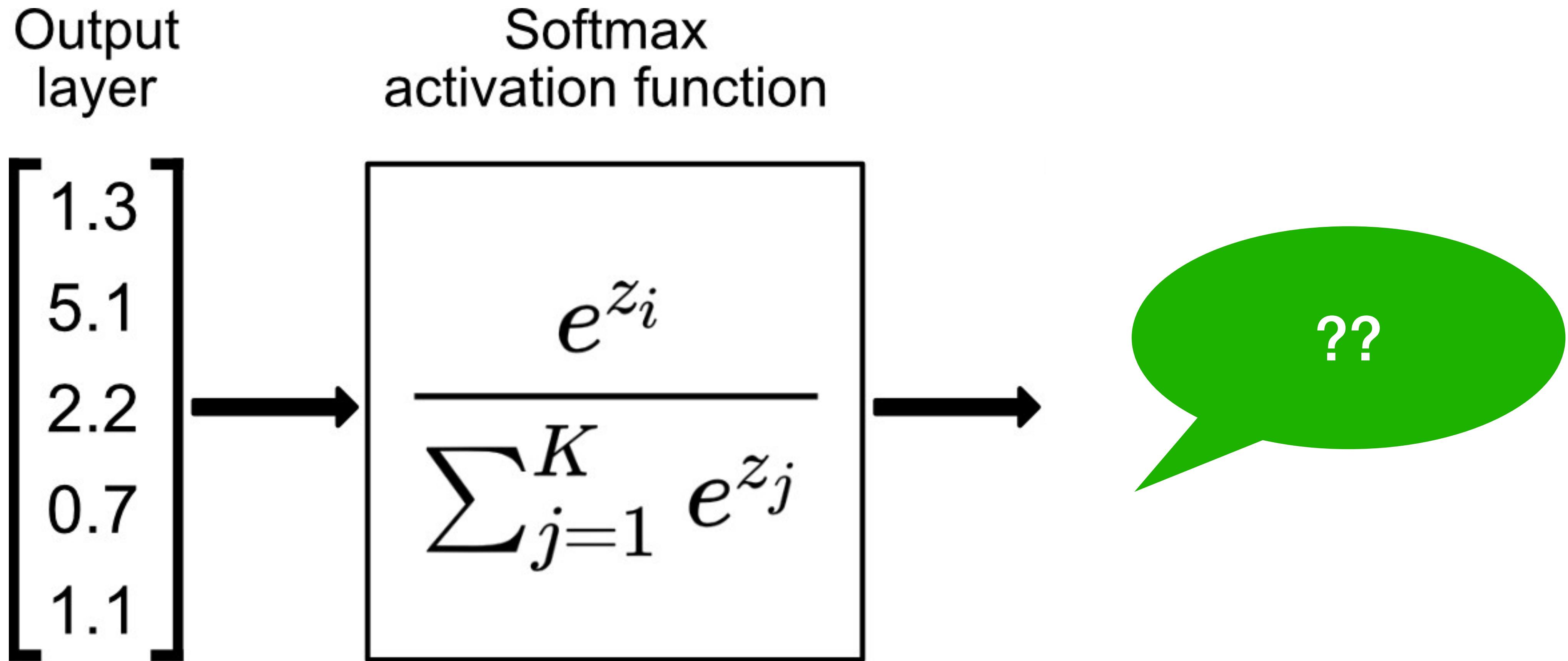
Turns outputs f into probabilities (sum up to 1 across k classes)



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

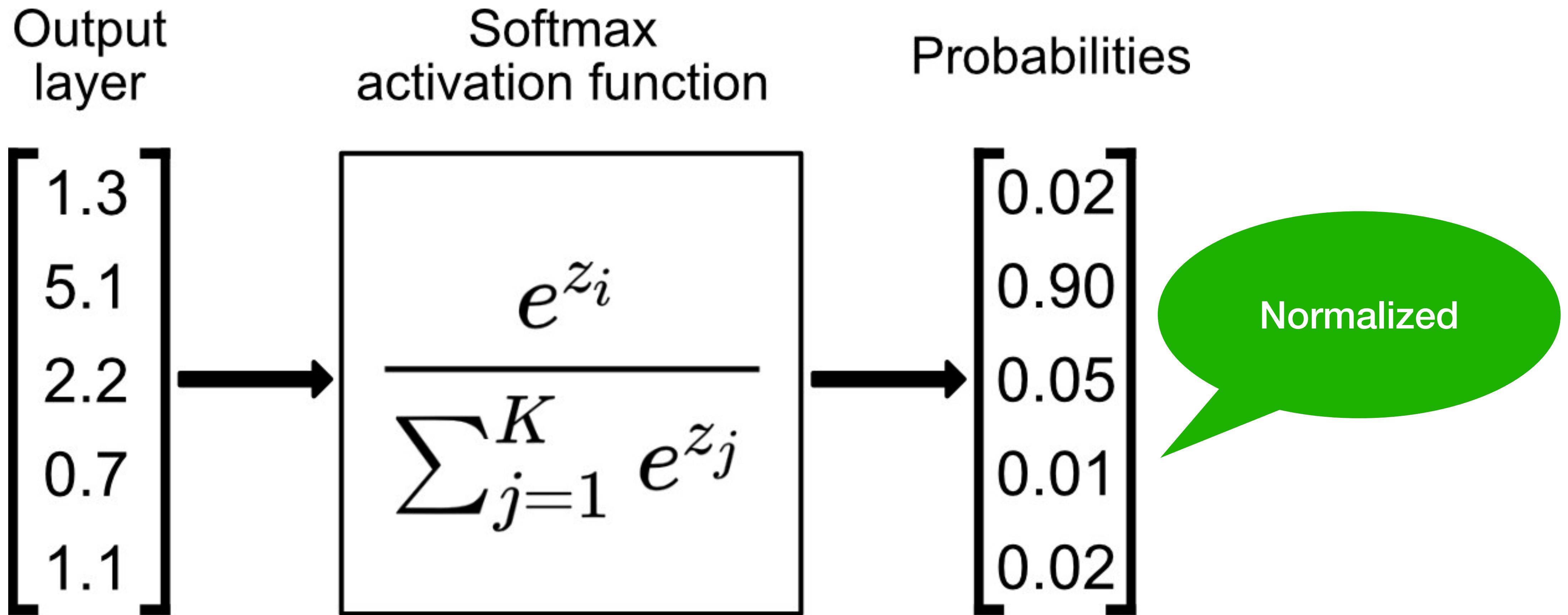
Softmax

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



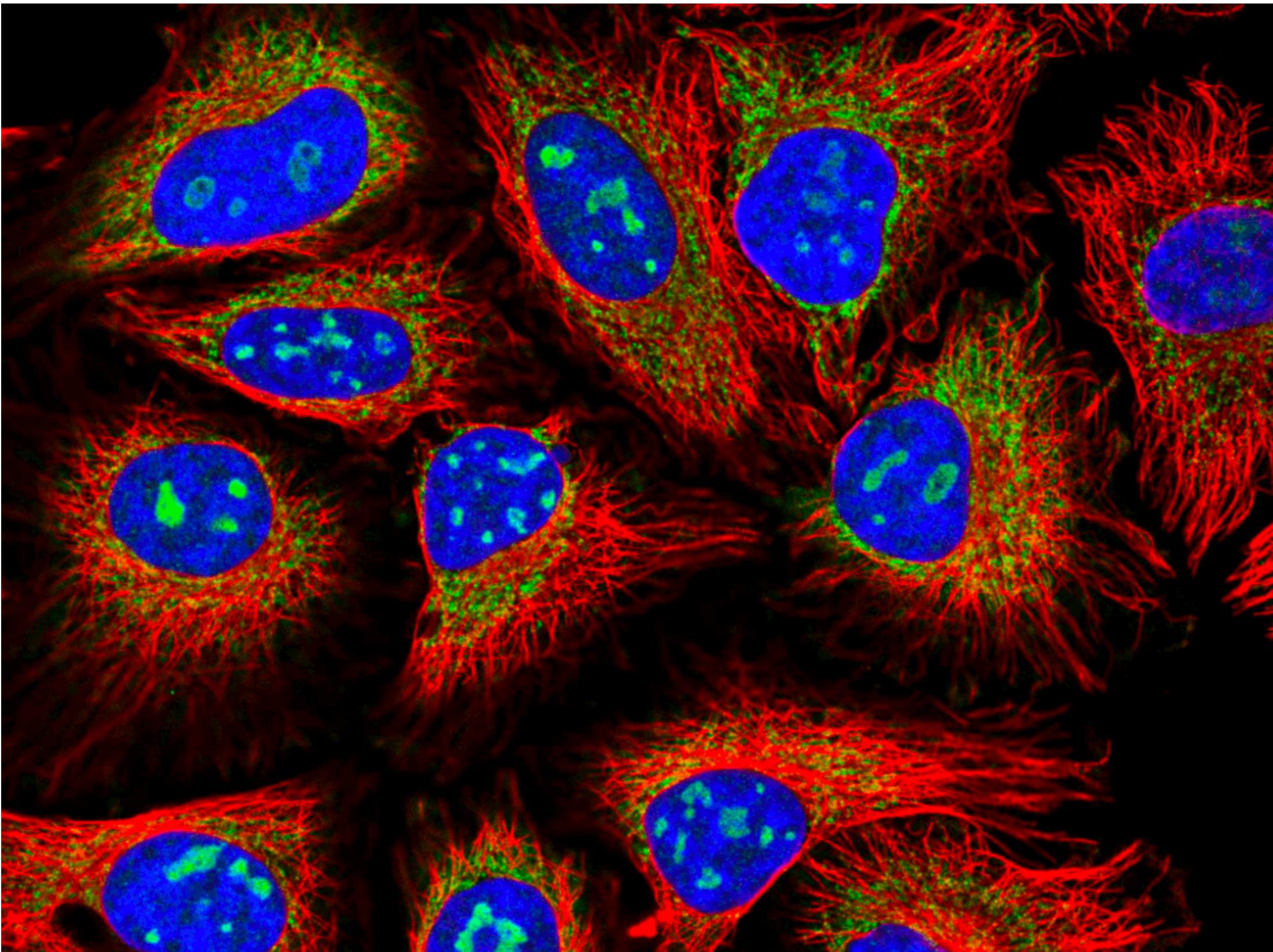
Softmax

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



Classification Tasks at Kaggle

Classify human protein microscope images into 28 categories

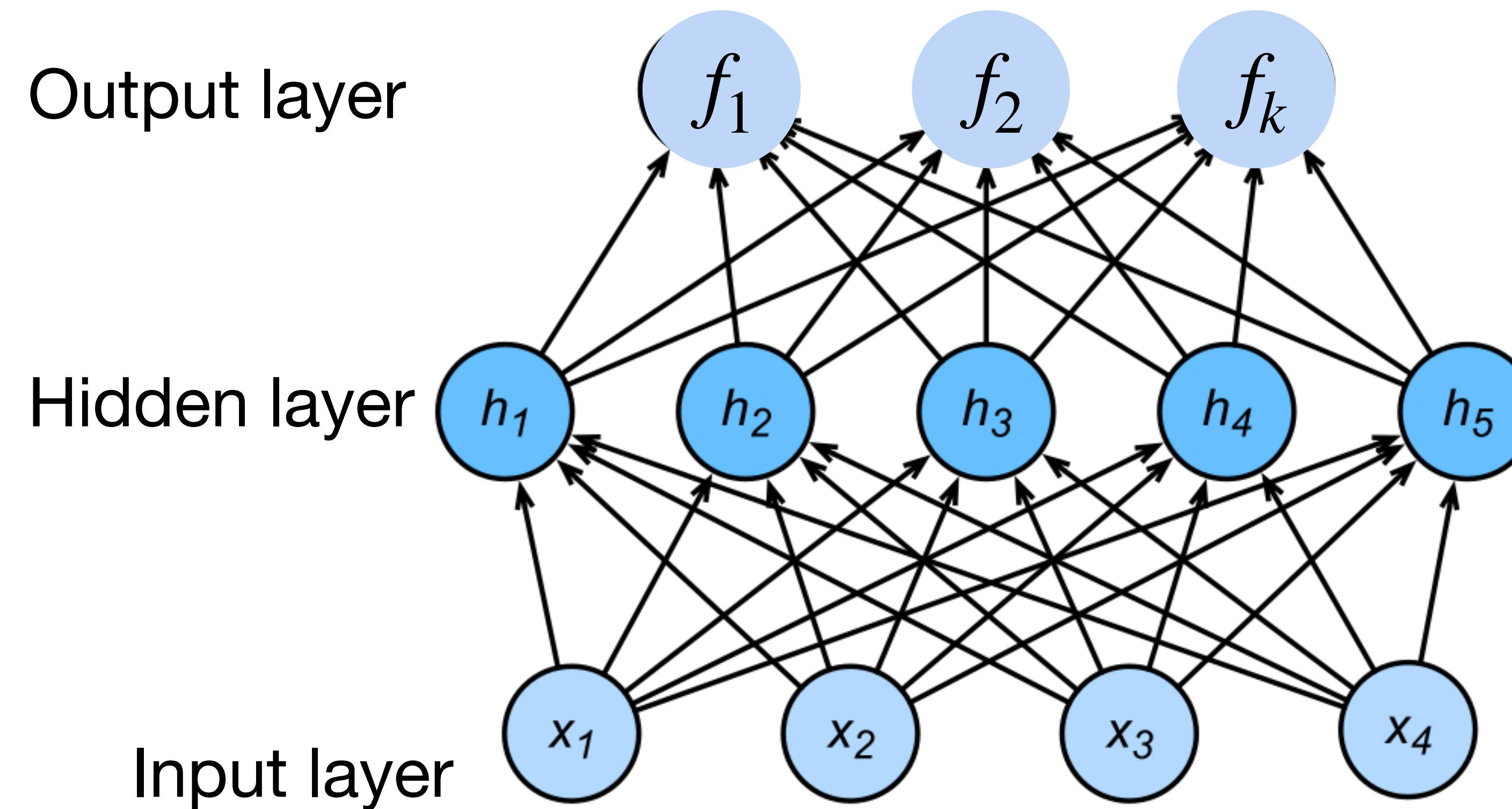


- 0. Nucleoplasm
- 1. Nuclear membrane
- 2. Nucleoli
- 3. Nucleoli fibrillar
- 4. Nuclear speckles
- 5. Nuclear bodies
- 6. Endoplasmic reticu
- 7. Golgi apparatus
- 8. Peroxisomes
- 9. Endosomes
- 10. Lysosomes
- 11. Intermediate fila
- 12. Actin filaments
- 13. Focal adhesion si
- 14. Microtubules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

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

More complicated neural networks

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



More complicated neural networks

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

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

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

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

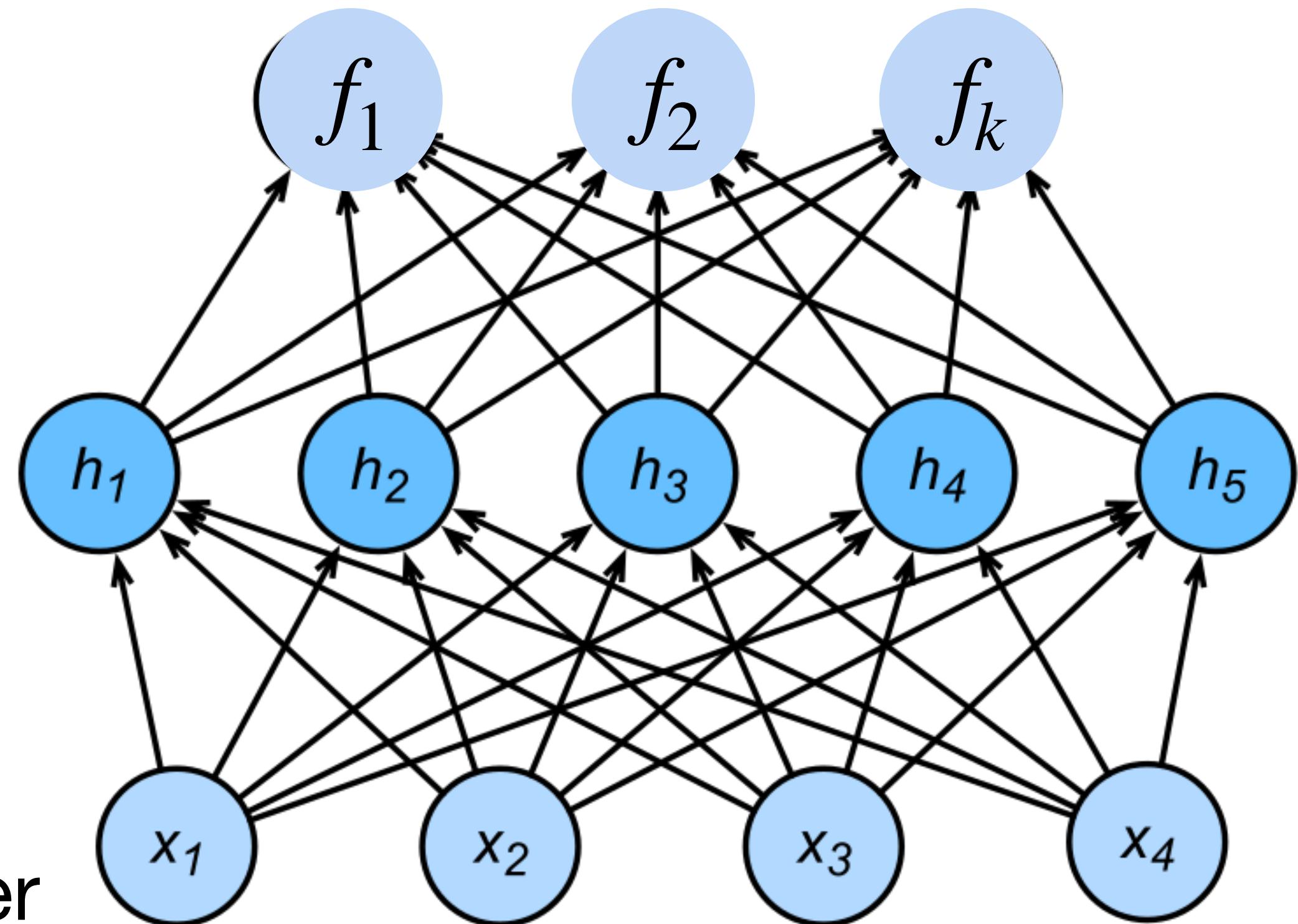
$$\mathbf{y} = \text{softmax}(\mathbf{f})$$

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

Output layer

Hidden layer

Input layer



More complicated neural networks: multiple hidden layers

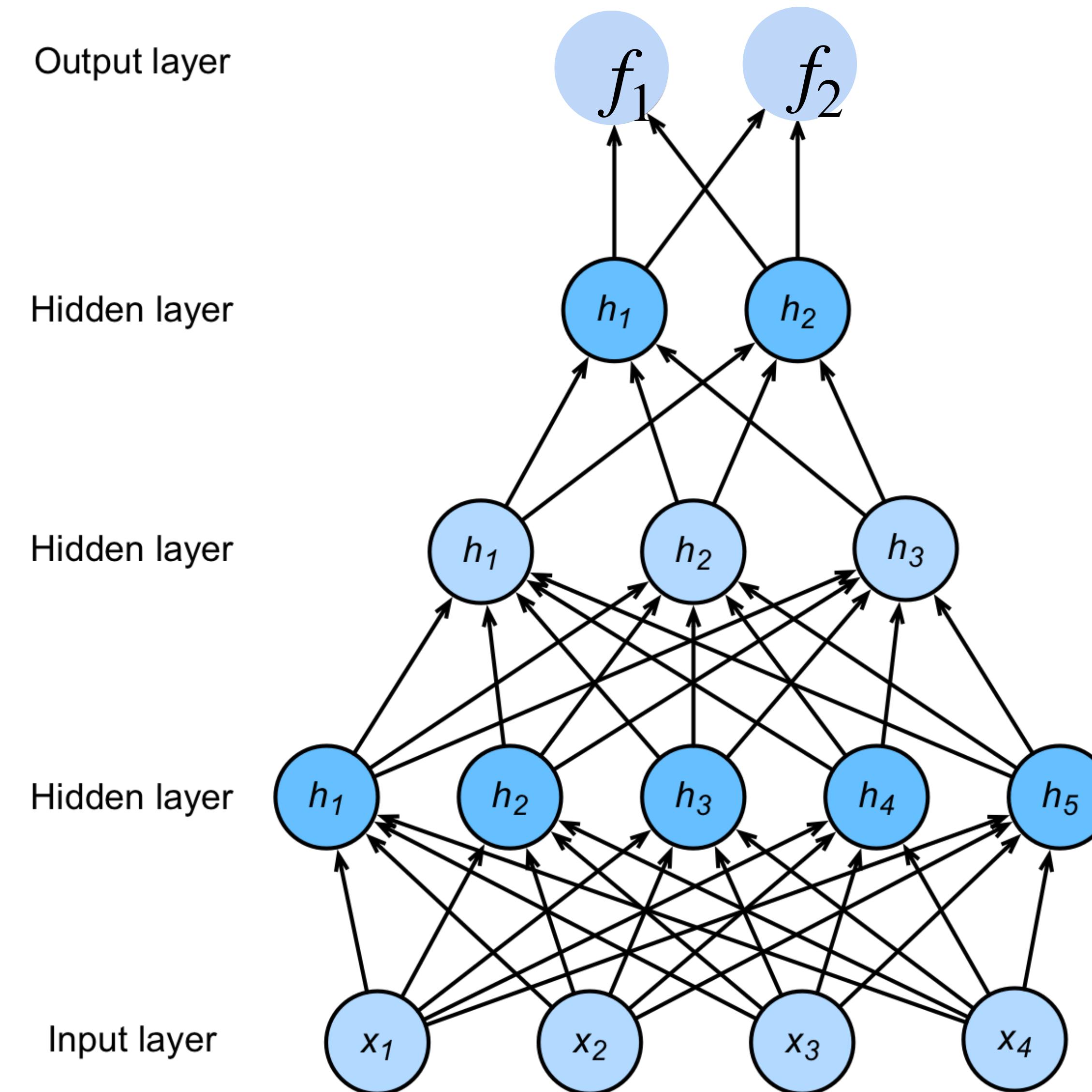
$$\mathbf{h}_1 = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

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

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

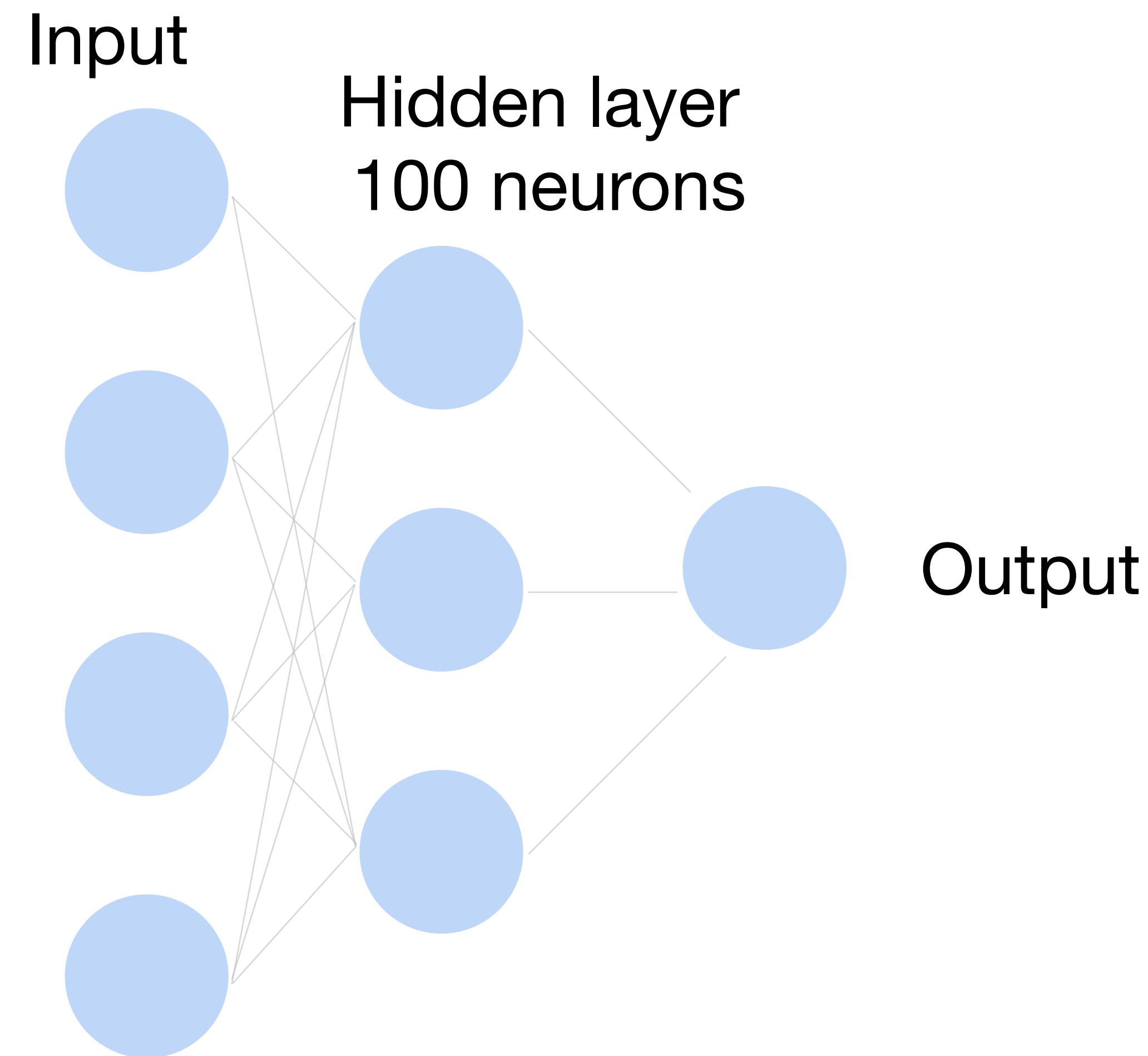
$$\mathbf{f} = \mathbf{W}_4 \mathbf{h}_3 + \mathbf{b}_4$$

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



How to train a neural network?

Classify cats vs. dogs



How to train a neural network?

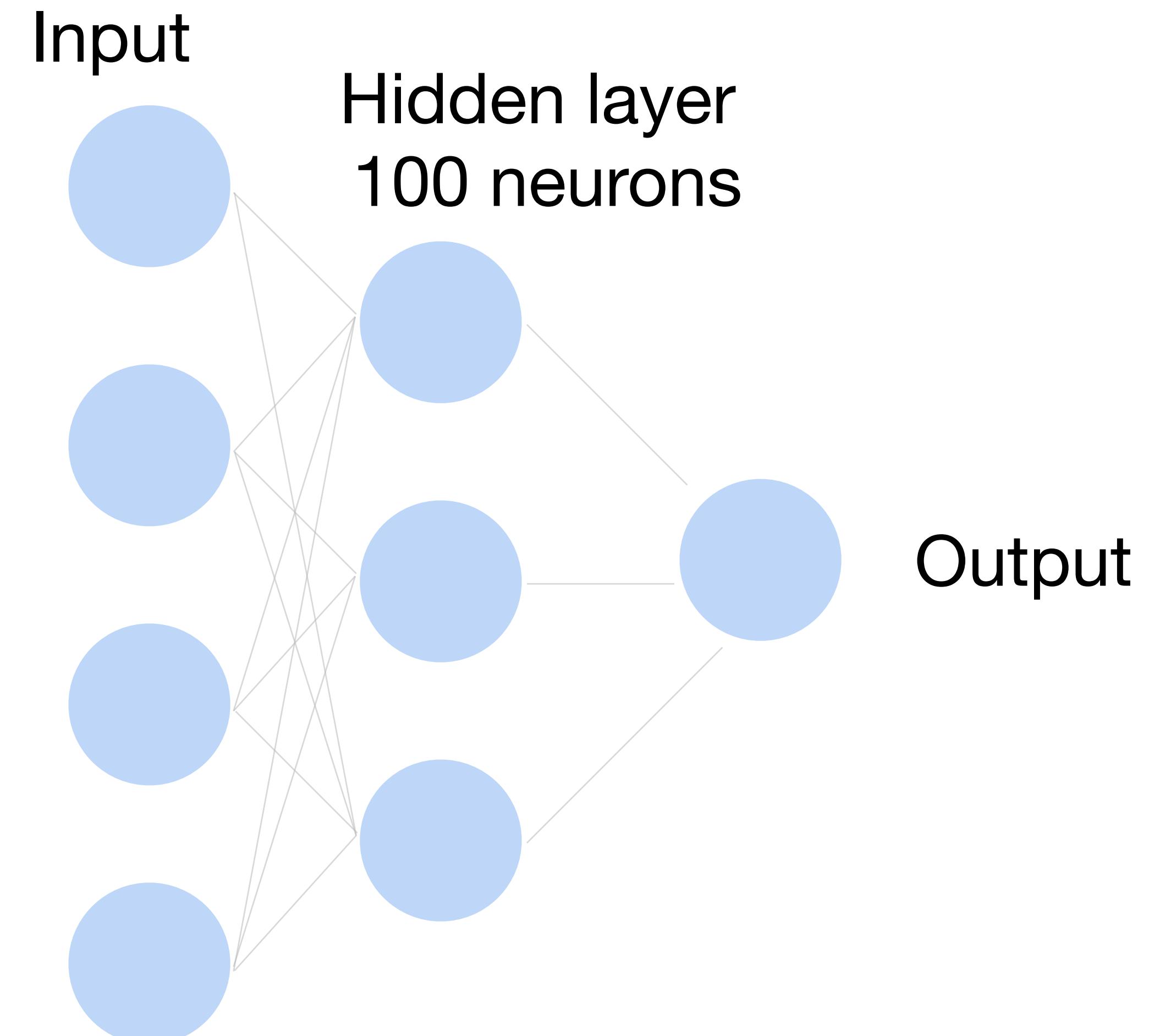
$\mathbf{x} \in \mathbb{R}^d$ One training data point in the training set D

\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 $y = 0$

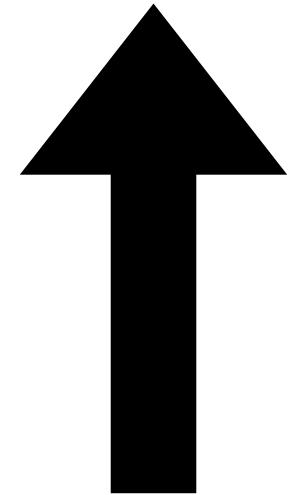


How to train a neural network?

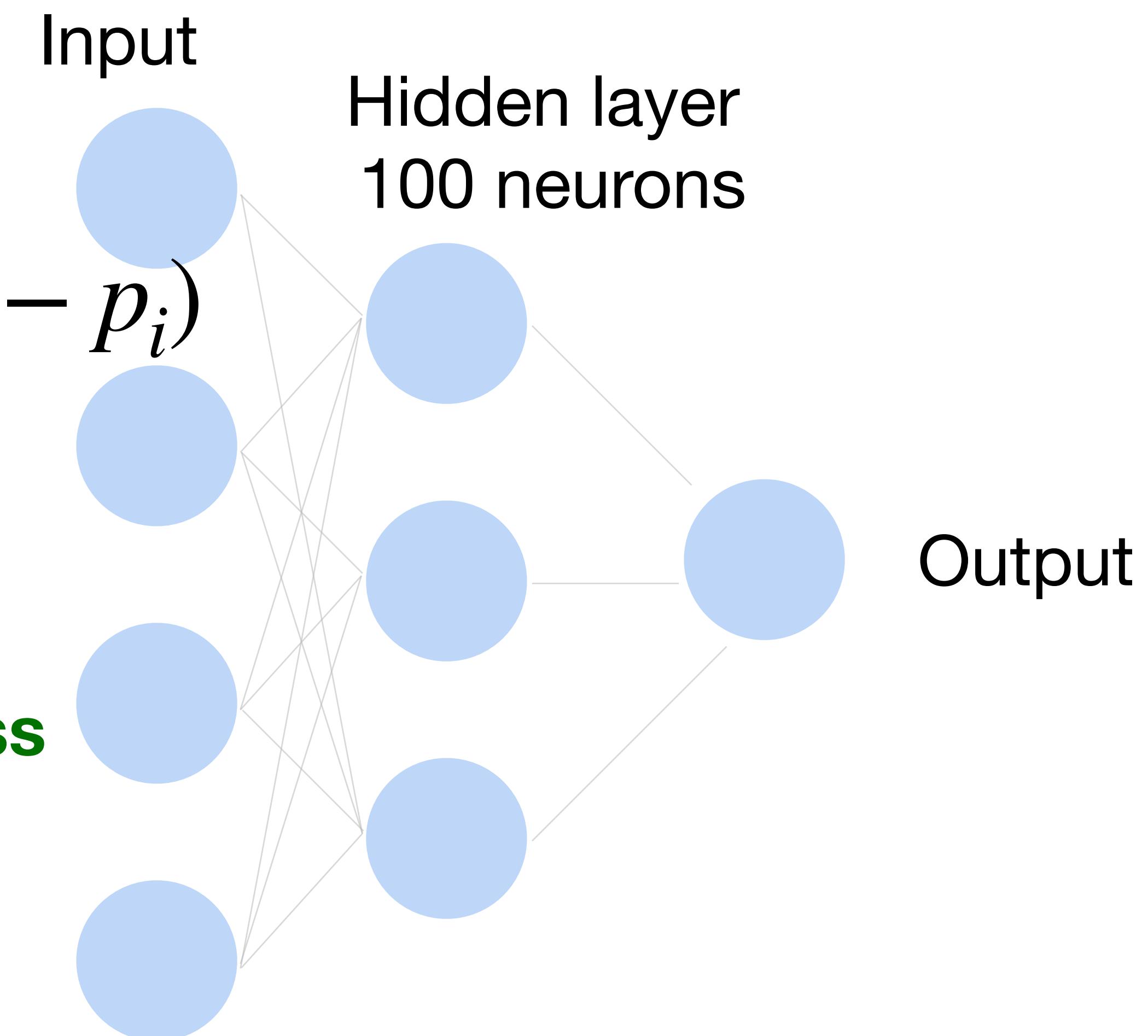
Loss function: $\frac{1}{|D|} \sum_i \ell(\mathbf{x}_i, y_i)$

Per-sample loss:

$$\ell(\mathbf{x}_i, y_i) = -y \log(p_i) + (1 - y) \log(1 - p_i)$$



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

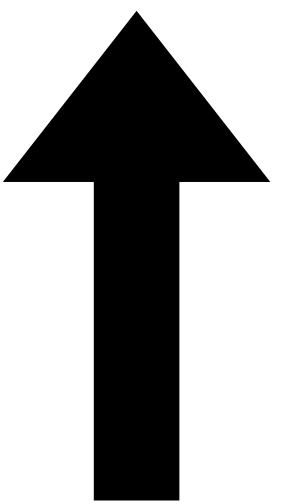


How to train a neural network?

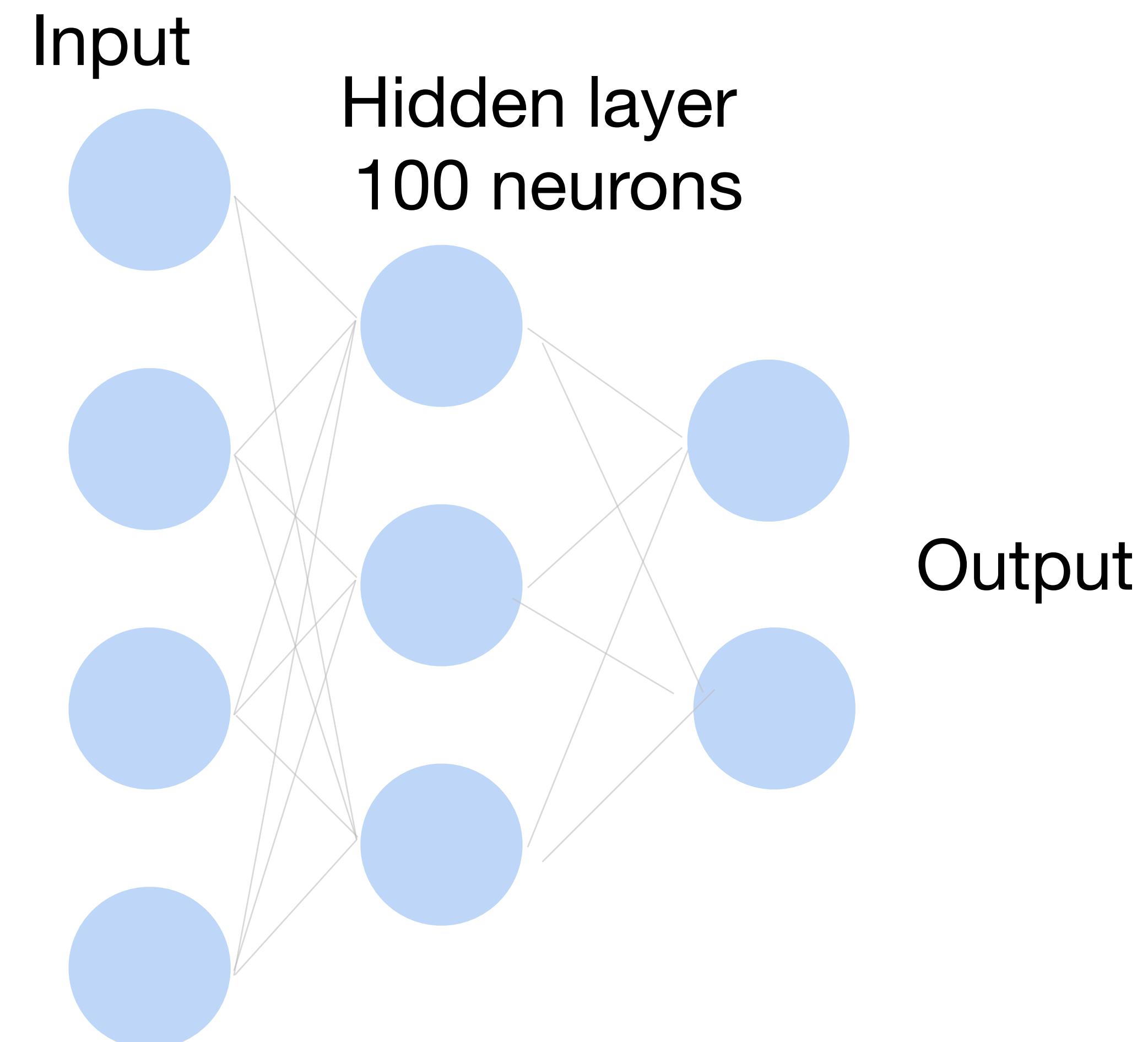
Loss function: $\frac{1}{|D|} \sum_i \ell(\mathbf{x}_i, y_i)$

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**

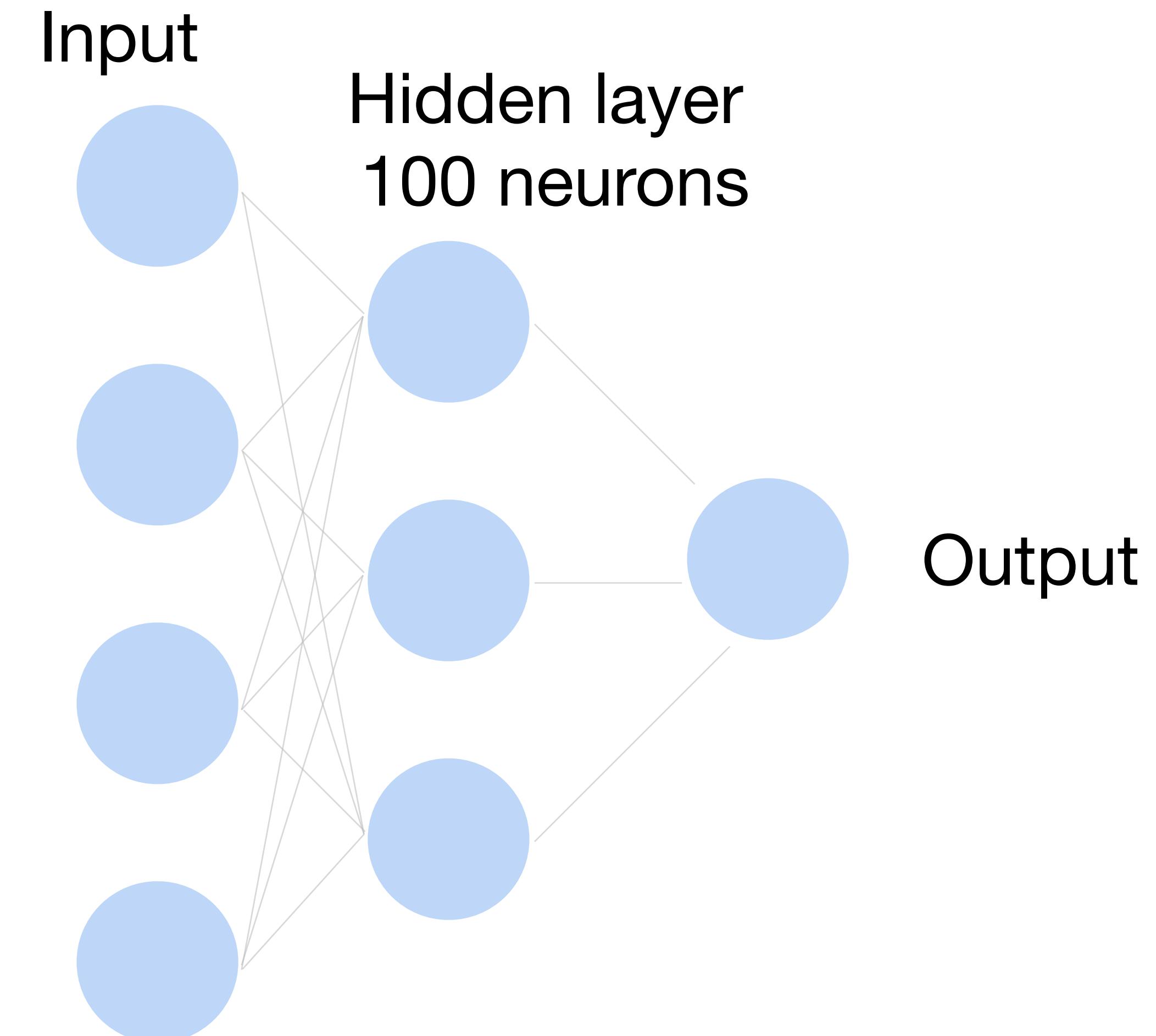


How to train a neural network?

Update the weights W to minimize the loss function

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

Use gradient descent!



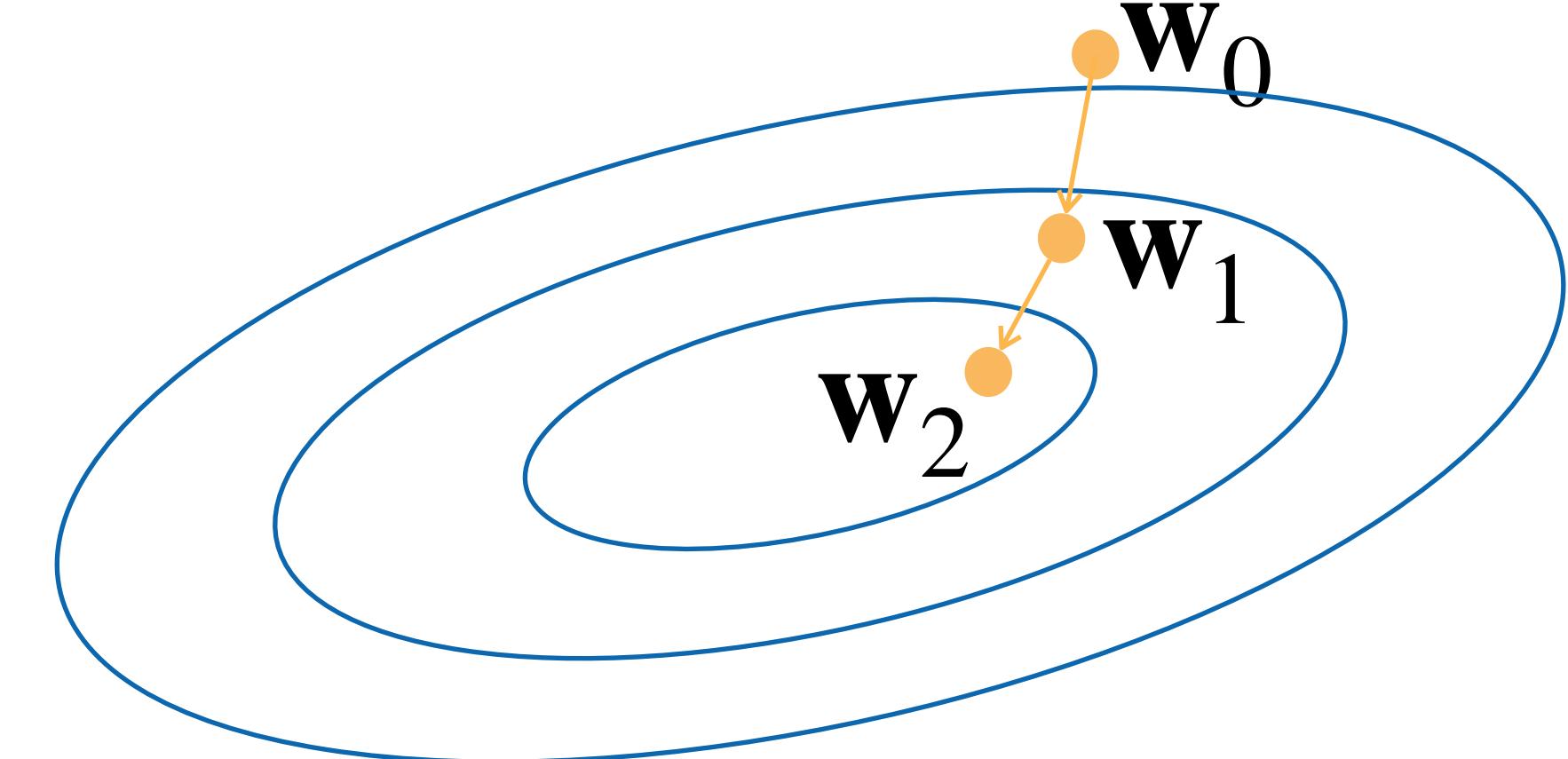
Gradient Descent

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

- Update parameters:

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

D can
be very large.
Expensive



- Repeat until converges

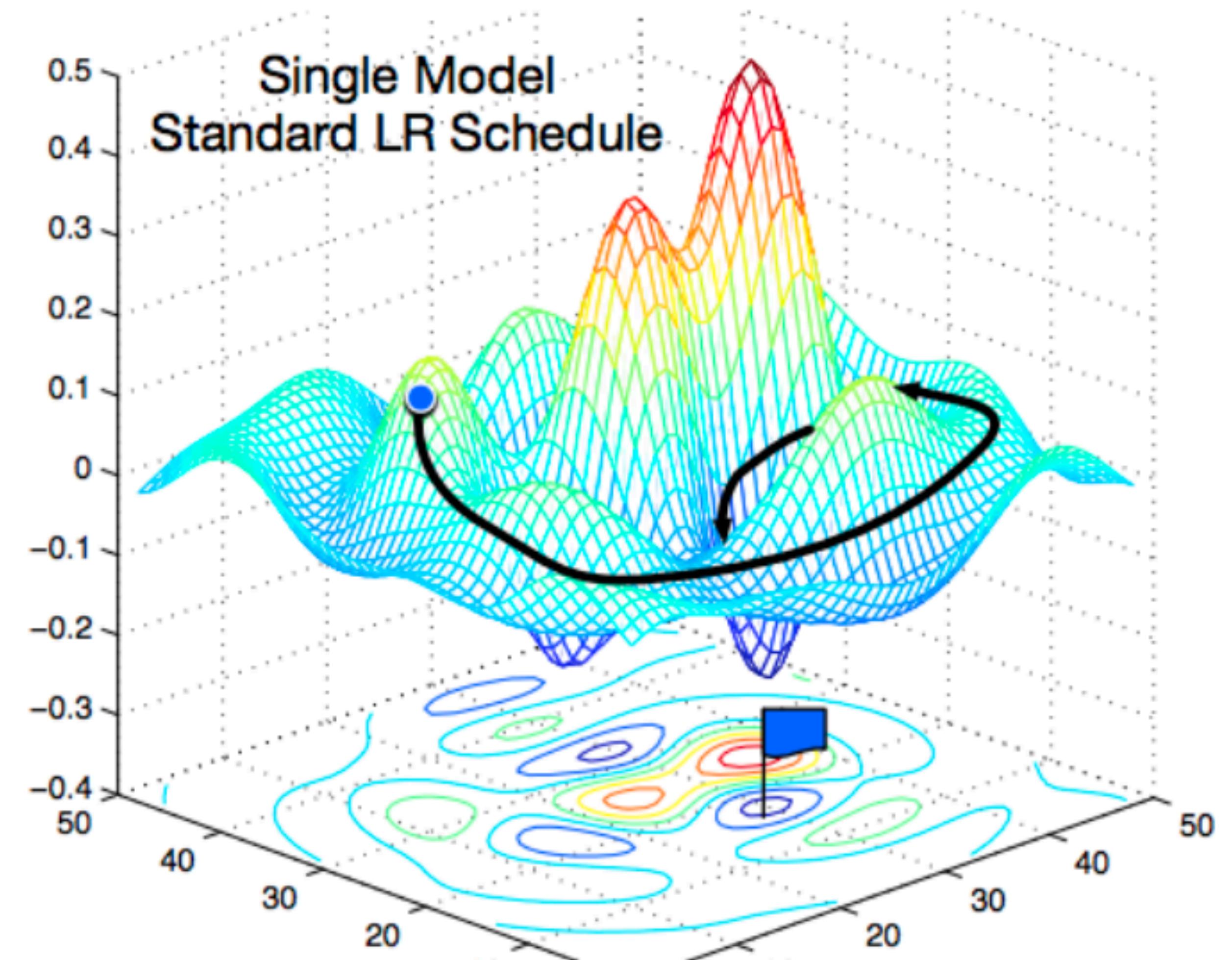
Minibatch Stochastic Gradient Descent

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

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

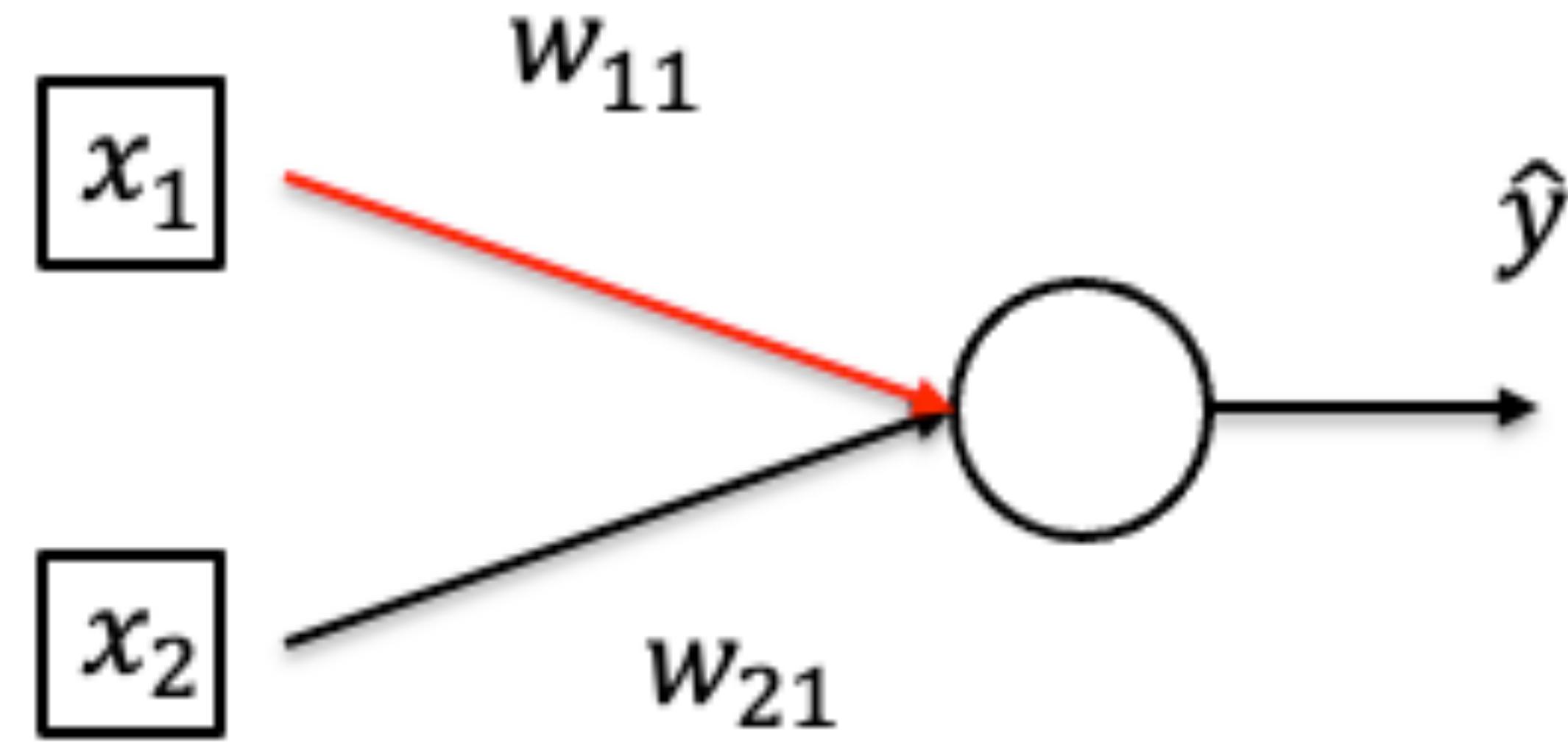
- Repeat until converges

Non-convex Optimization



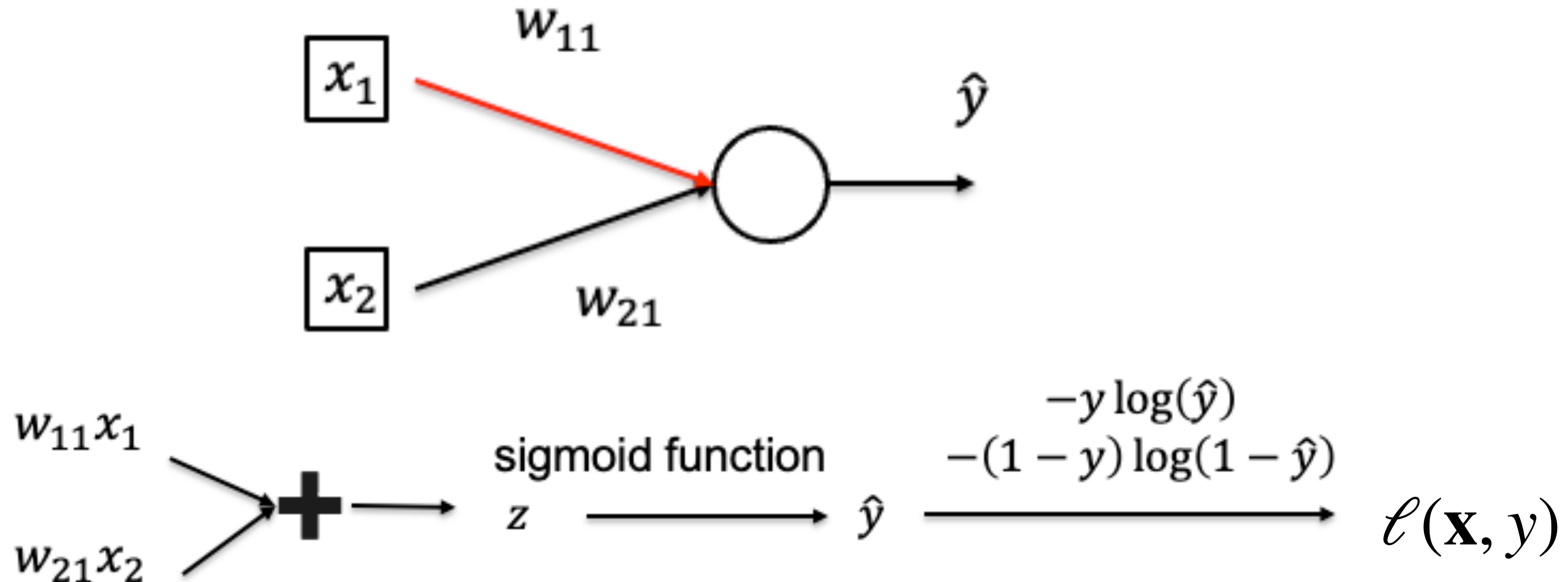
[Gao and Li et al., 2018]

Calculate Gradient (on one data point)

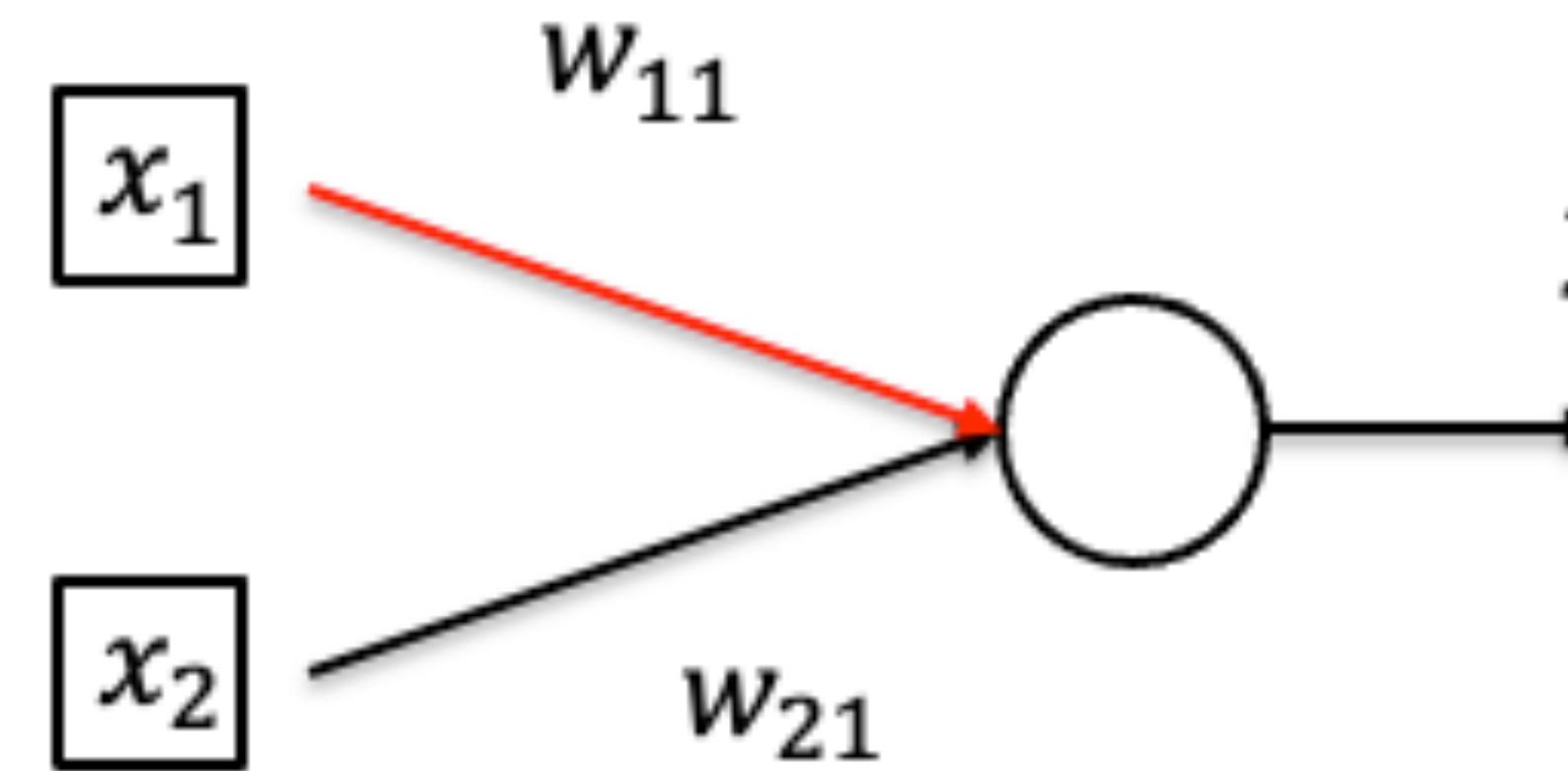


- Want to compute $\frac{\partial \ell(\mathbf{x}, y)}{\partial w_{11}}$

Calculate Gradient (on one data point)



Calculate Gradient (on one data point)

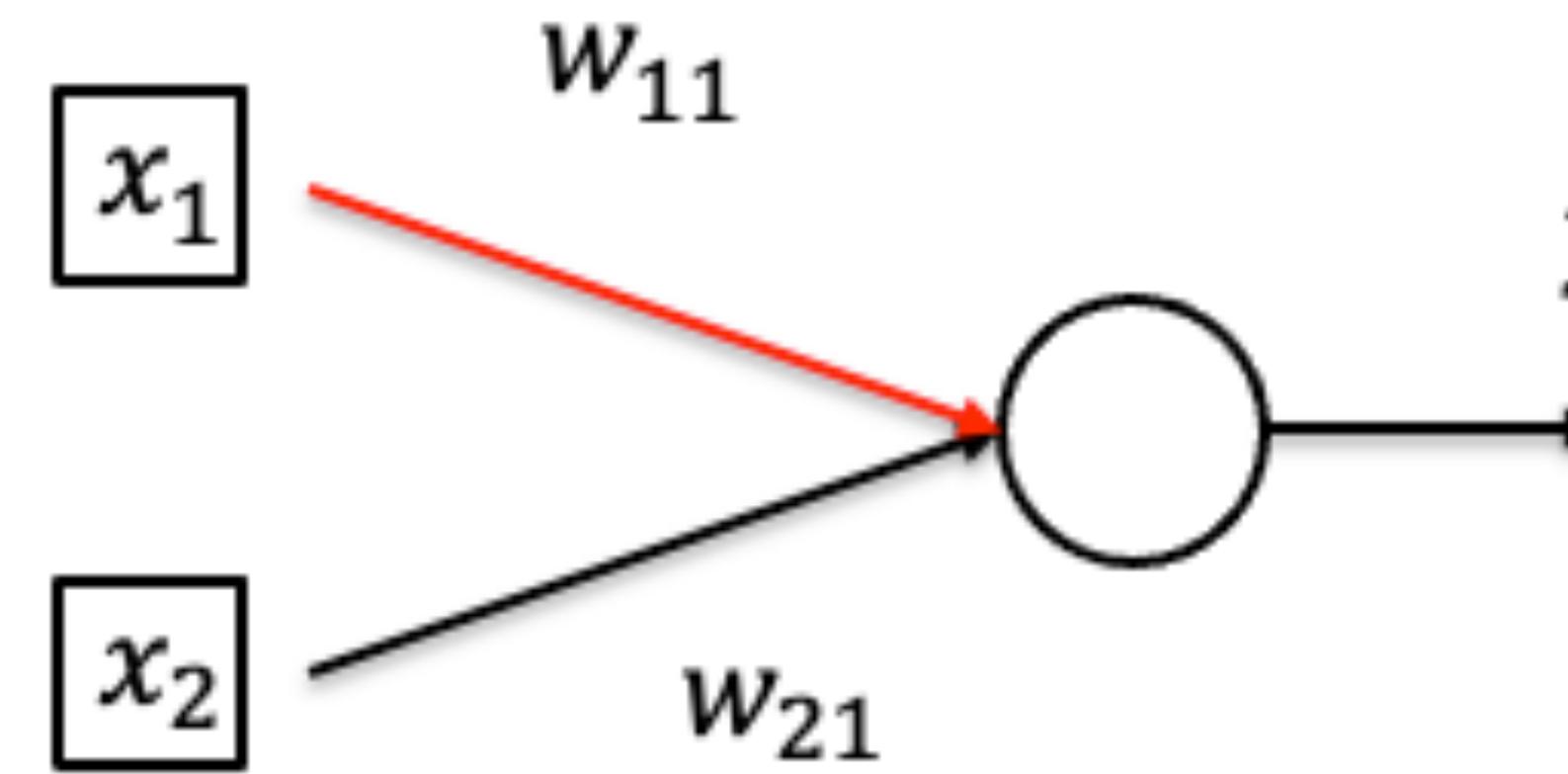


$$\begin{array}{ccccc} w_{11}x_1 & \xrightarrow{\text{+}} & z & \xrightarrow{\text{sigmoid function}} & \hat{y} \\ w_{21}x_2 & & & & \xrightarrow{-y \log(\hat{y})} \\ & & & & -(1 - \hat{y}) \log(1 - \hat{y}) \\ & & & & \ell(\mathbf{x}, y) \\ \frac{\partial \hat{y}}{\partial z} = \sigma'(z) & & & \frac{\partial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

- 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)

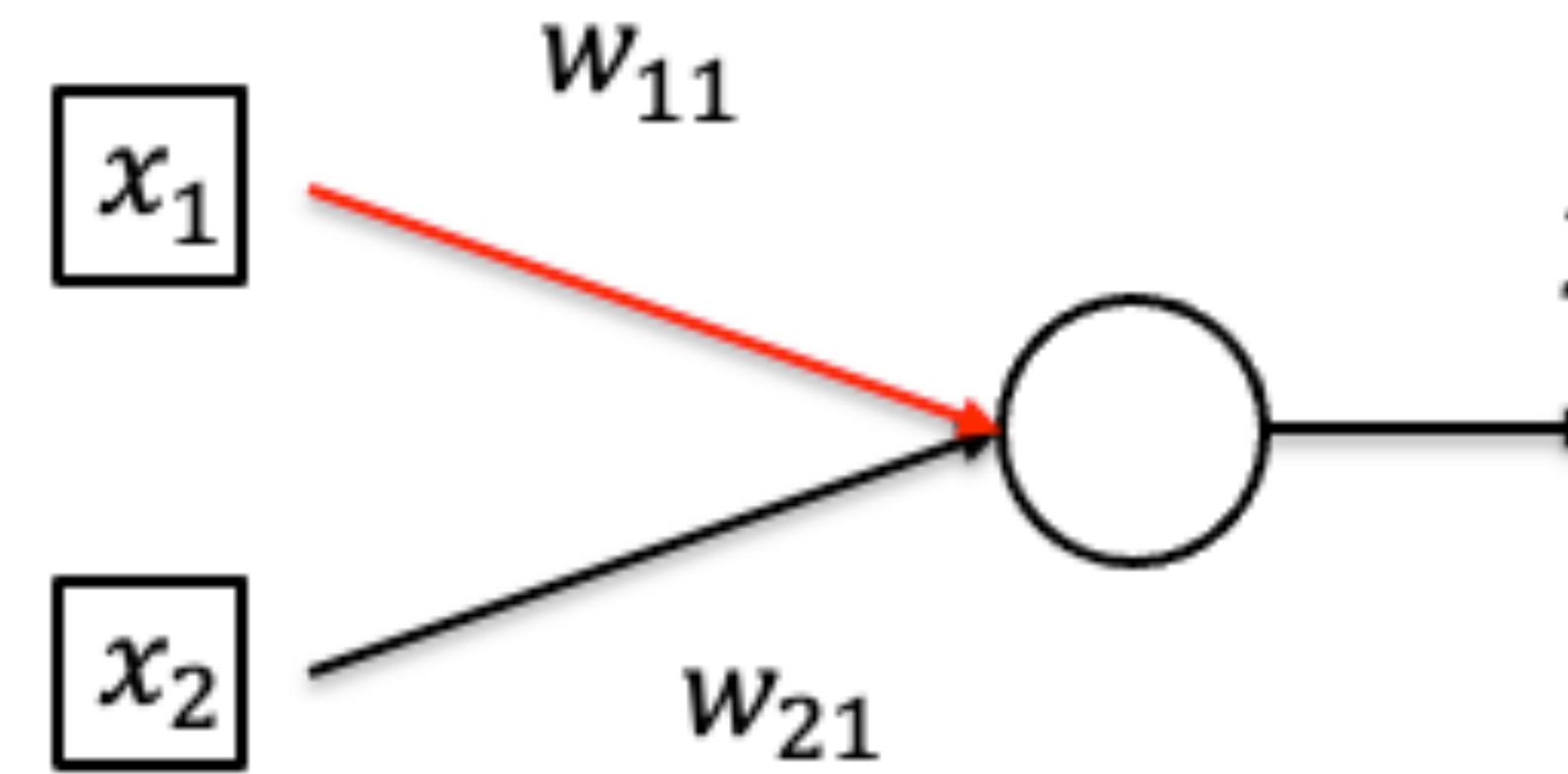


$$\begin{array}{ccccc} w_{11}x_1 & \xrightarrow{\quad + \quad} & z & \xrightarrow{\text{sigmoid function}} & \hat{y} \\ w_{21}x_2 & & & & \xrightarrow{-y \log(\hat{y})} \\ & & & & -(1 - \hat{y}) \log(1 - \hat{y}) \\ & & & & \ell(\mathbf{x}, y) \\ \frac{\partial \hat{y}}{\partial z} = \sigma'(z) & & &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

- By chain rule:

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

Calculate Gradient (on one data point)



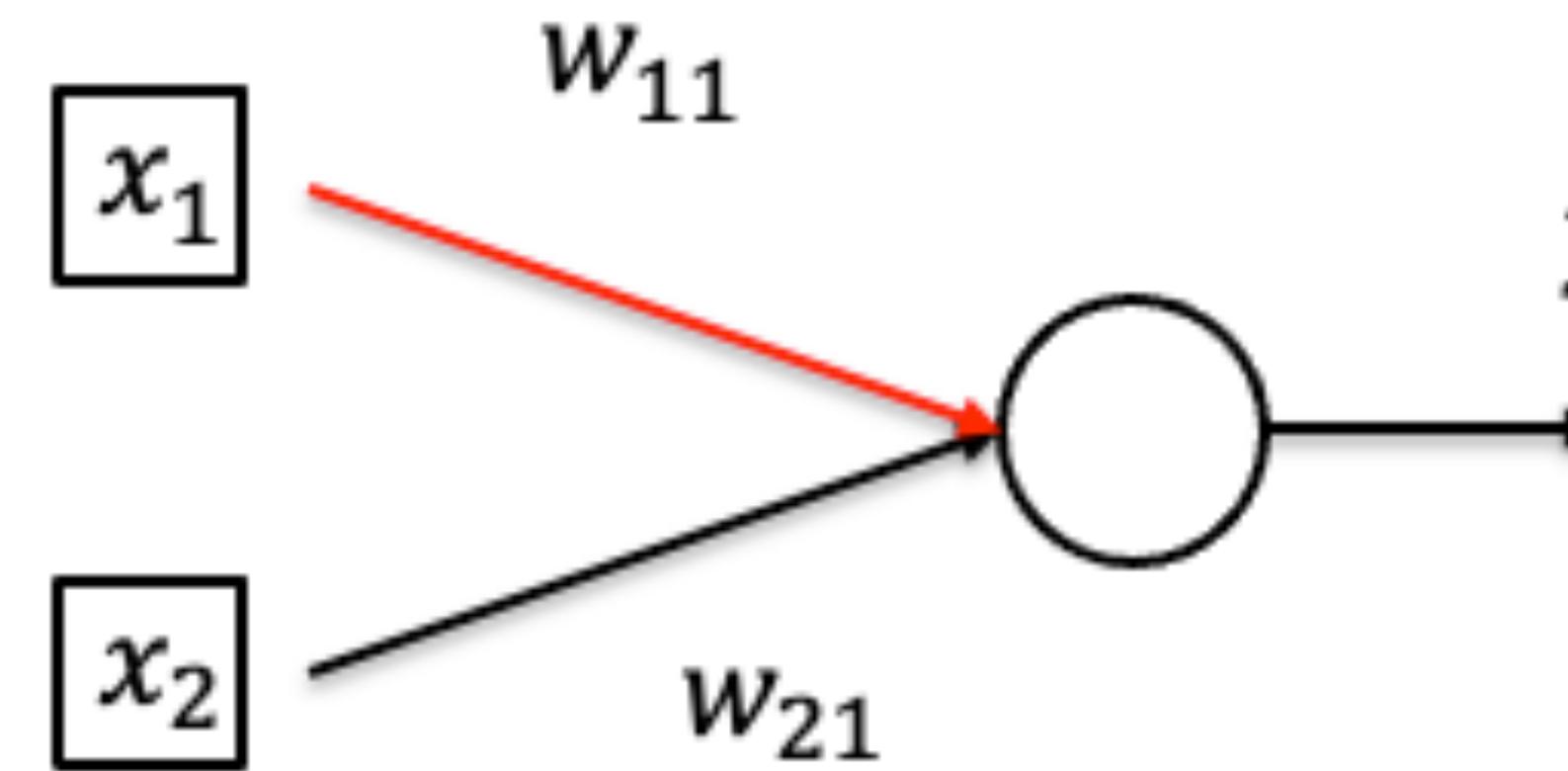
The diagram illustrates the forward pass through a neural network. It starts with two input values, $w_{11}x_1$ and $w_{21}x_2$, which are summed at a plus sign node. The result is z , which is then passed through a "sigmoid function". The output of the sigmoid function is \hat{y} . Finally, the loss function $\ell(\mathbf{x}, y)$ is calculated using the formula $-(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$. Below the diagram, the derivative of the sigmoid function is given as $\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$.

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

- By chain rule:

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

Calculate Gradient (on one data point)



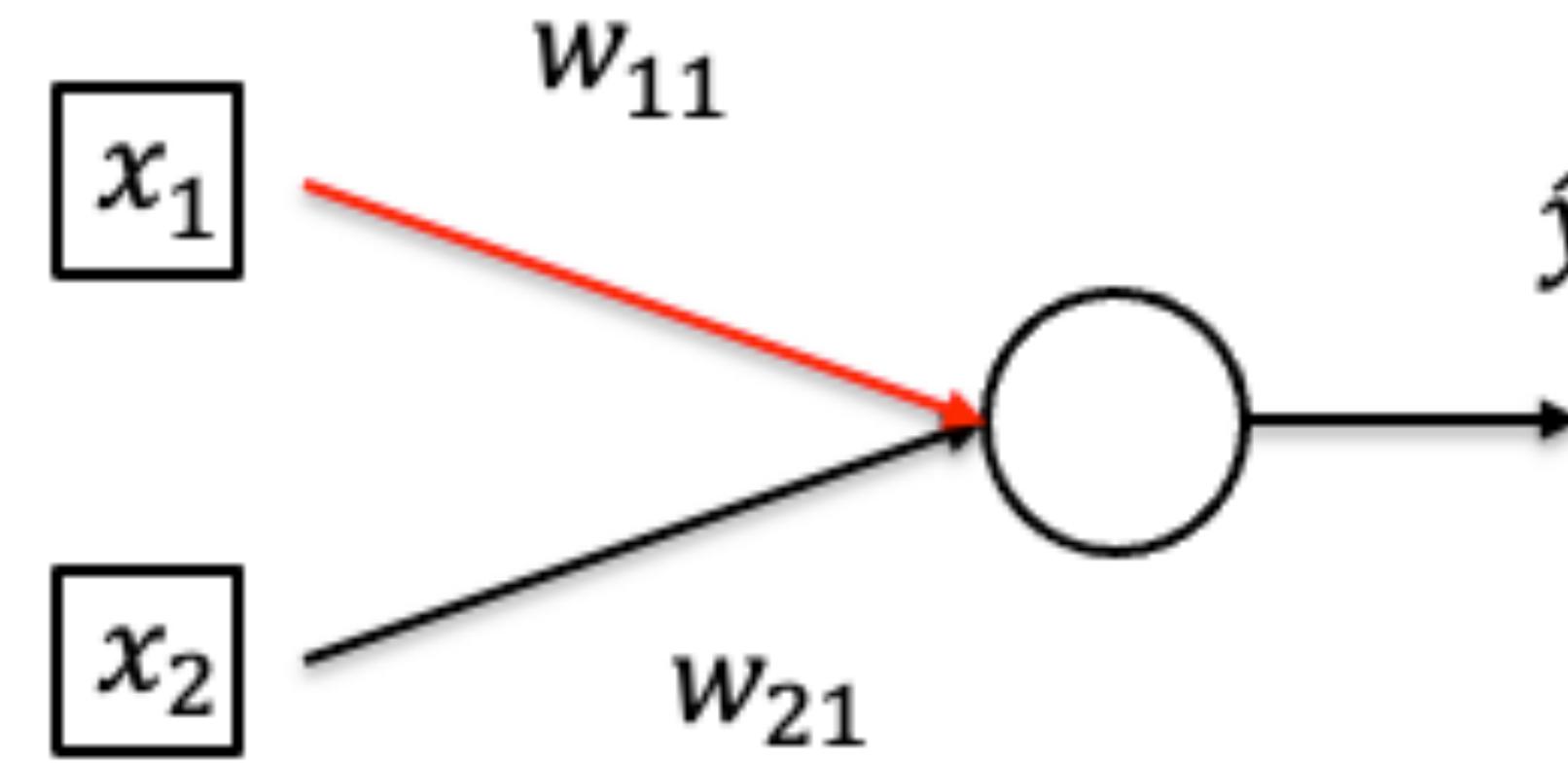
The diagram illustrates the forward pass of a neural network. It starts with inputs $w_{11}x_1$ and $w_{21}x_2$ which are summed at a plus sign. The result is passed through a "sigmoid function" to produce the output \hat{y} . The output \hat{y} is then used in the loss function $\ell(\mathbf{x}, y)$. Below the diagram, the derivative of the sigmoid function is given as $\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$.

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

- 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)



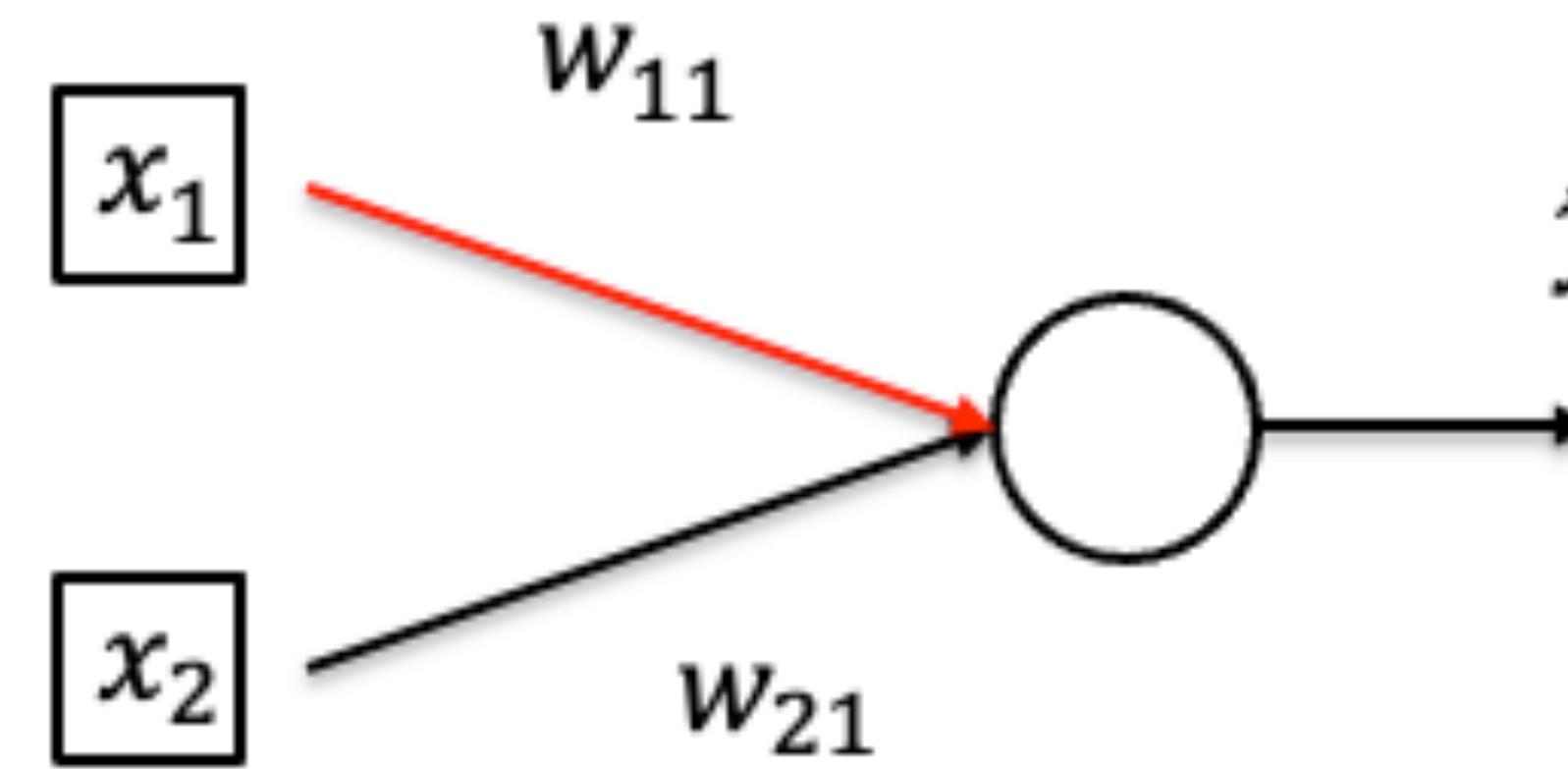
A computational graph illustrating the forward pass through a neural network layer. It starts with two inputs, $w_{11}x_1$ and $w_{21}x_2$, which are summed at a plus node. The result is passed through a sigmoid function to produce the output \hat{y} . Finally, the cross-entropy loss is calculated as $-(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$.

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

- By chain rule:

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

Calculate Gradient (on one data point)



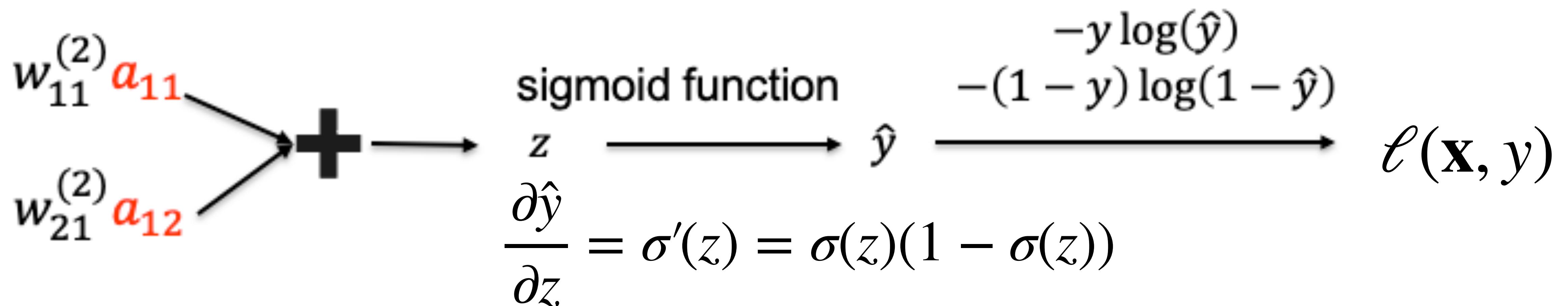
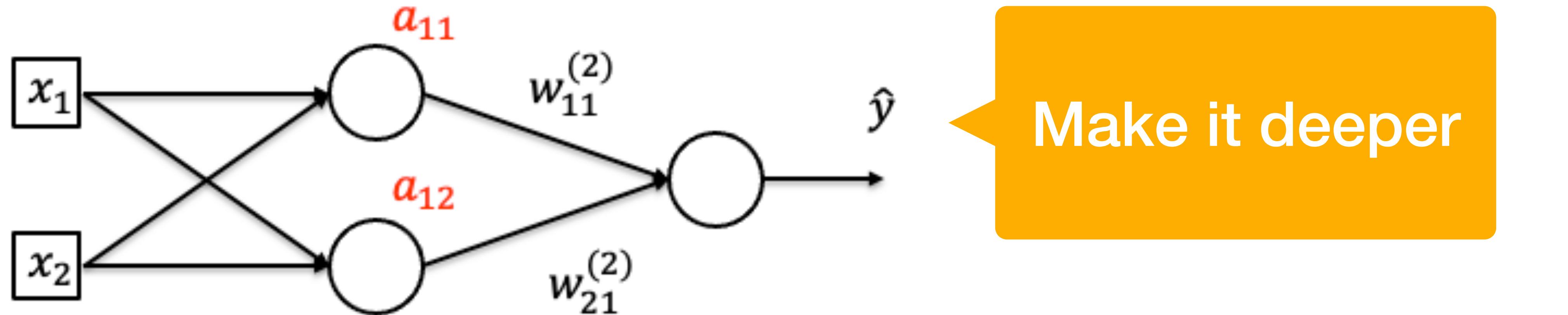
A computational graph illustrating the forward pass and the calculation of gradients. On the left, two inputs $w_{11}x_1$ and $w_{21}x_2$ are summed to produce z . This z is passed through a "sigmoid function" to produce the output \hat{y} . The loss function $\ell(\mathbf{x}, y)$ is then calculated based on \hat{y} . Below the graph, the gradient of the loss with respect to z is given as $\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$.

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

- By chain rule:

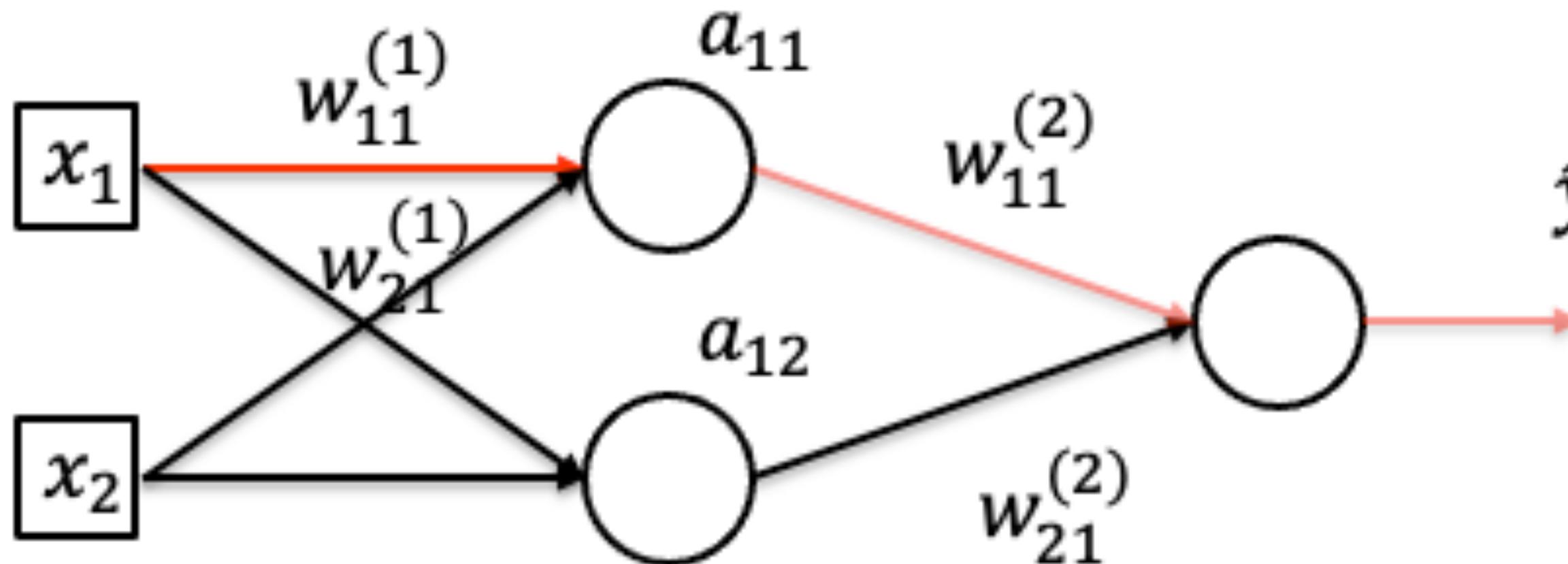
$$\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} w_{11} = (\hat{y} - y)w_{11}$$

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)

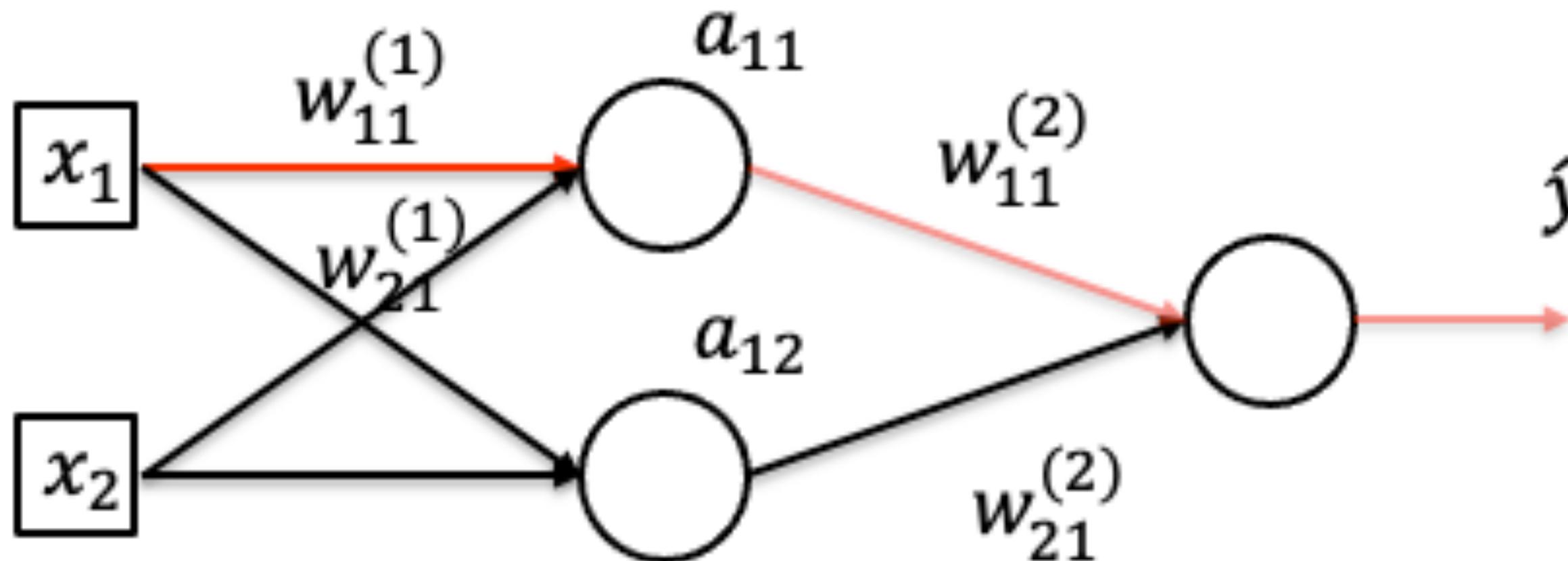


A computational graph illustrating the backpropagation process. The graph shows the flow of values and gradients:

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

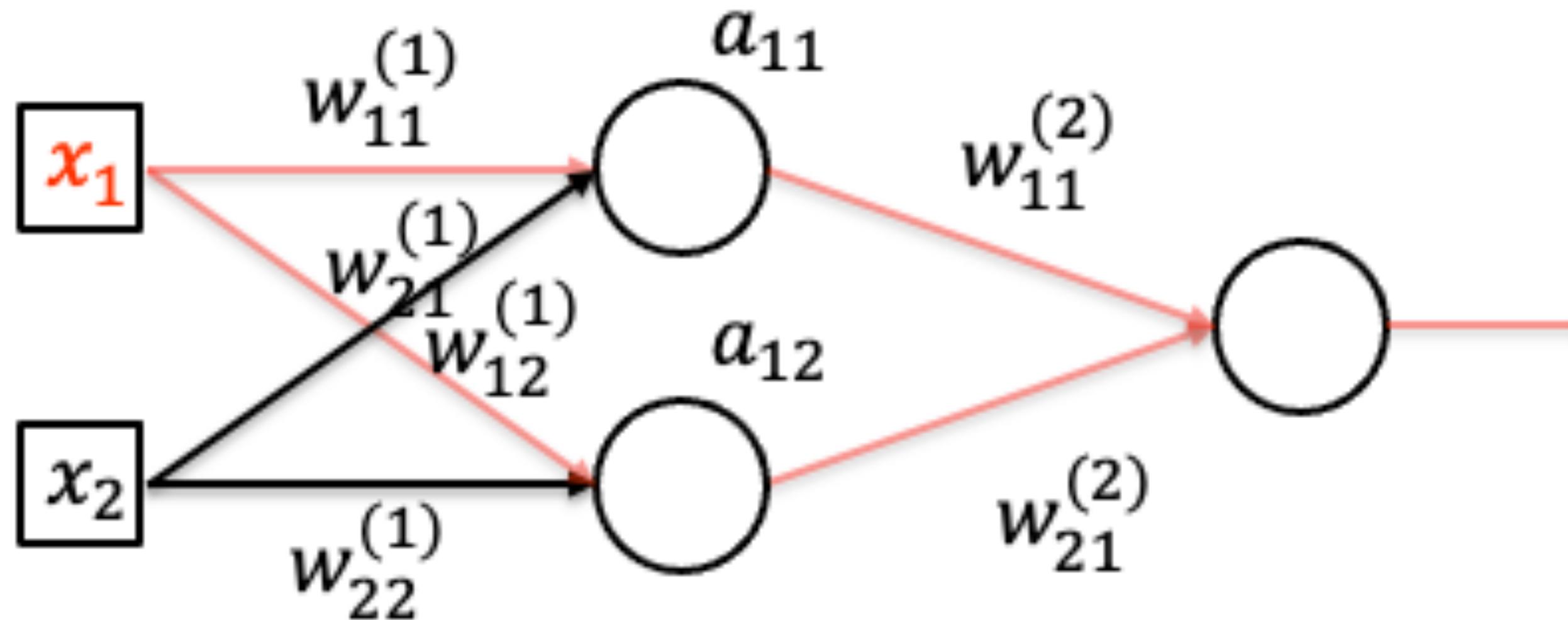
Calculate Gradient (on one data point)



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

- 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)

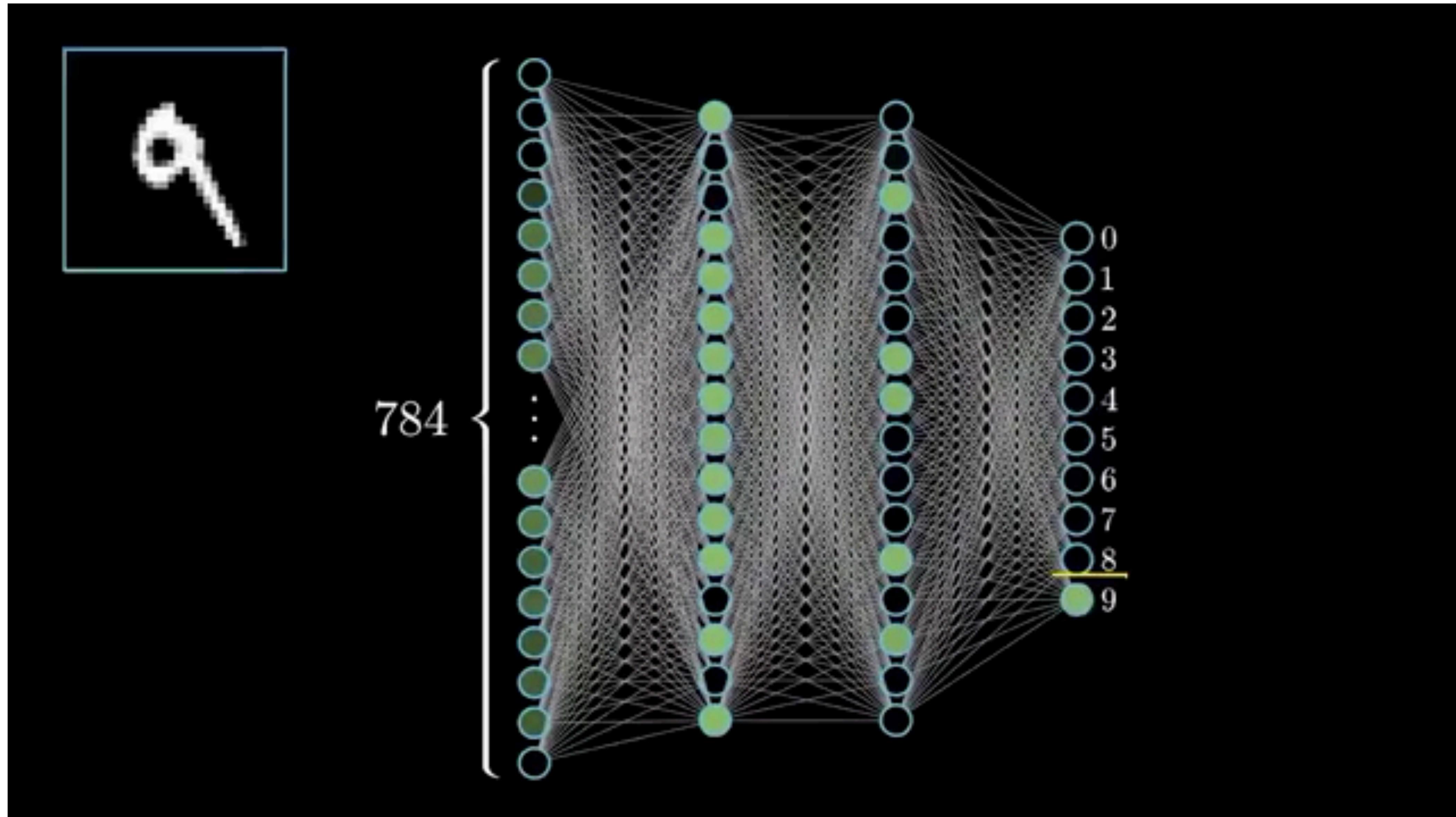


$$\begin{aligned} w_{11}^{(1)} x_1 & \quad \quad \quad + \quad \quad \quad \sigma(z_{11}) \quad \quad \quad l(\mathbf{x}, y) \\ w_{21}^{(1)} x_2 & \end{aligned}$$
$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \quad \quad \quad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y) w_{11}^{(2)}$$

- By chain rule:

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

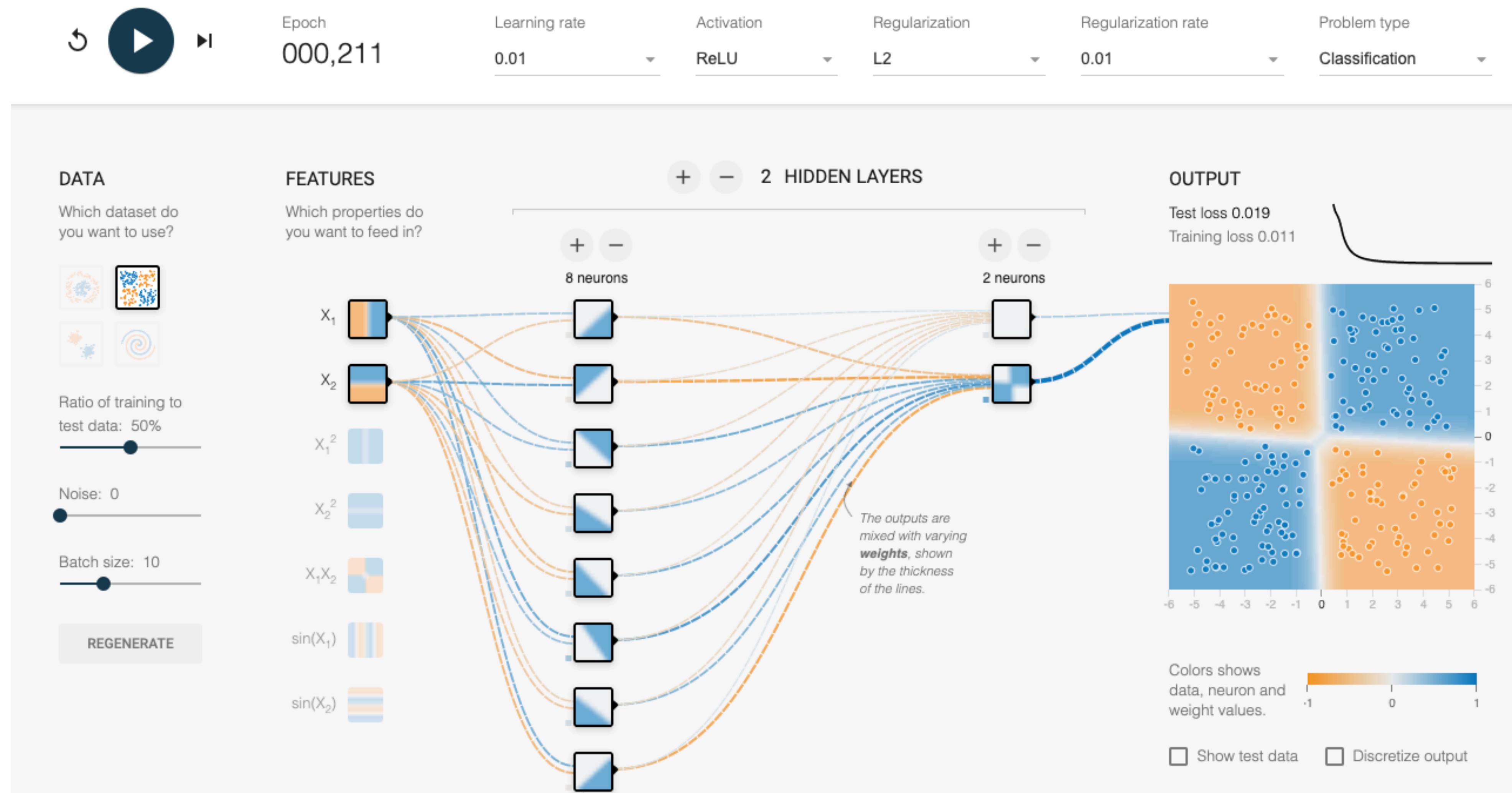
HW6



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!

Based on slides from Xiaojin (Jerry) Zhu, Yingyu Liang and Yin Li (<http://pages.cs.wisc.edu/~jerryzhu/cs540.html>), and Alex Smola: <https://courses.d2l.ai/berkeley-stat-157/units/mlp.html>