CS 540 Introduction to Artificial Intelligence

Neural Networks (II)

Yingyu Liang
University of Wisconsin-Madison

Oct 21, 2021

Slides created by Sharon Li [modified by Yingyu Liang]

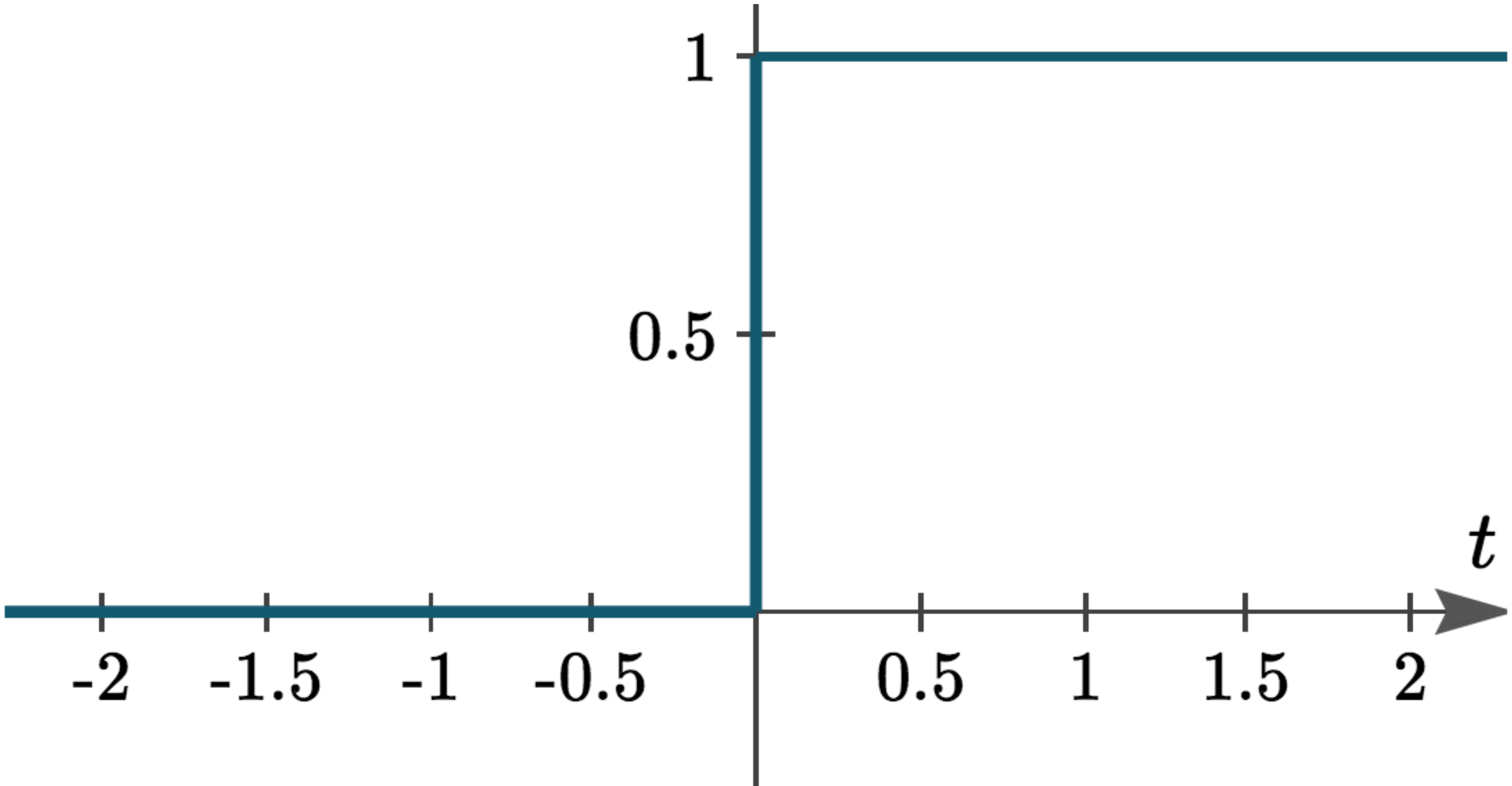
Today’s outline

- 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

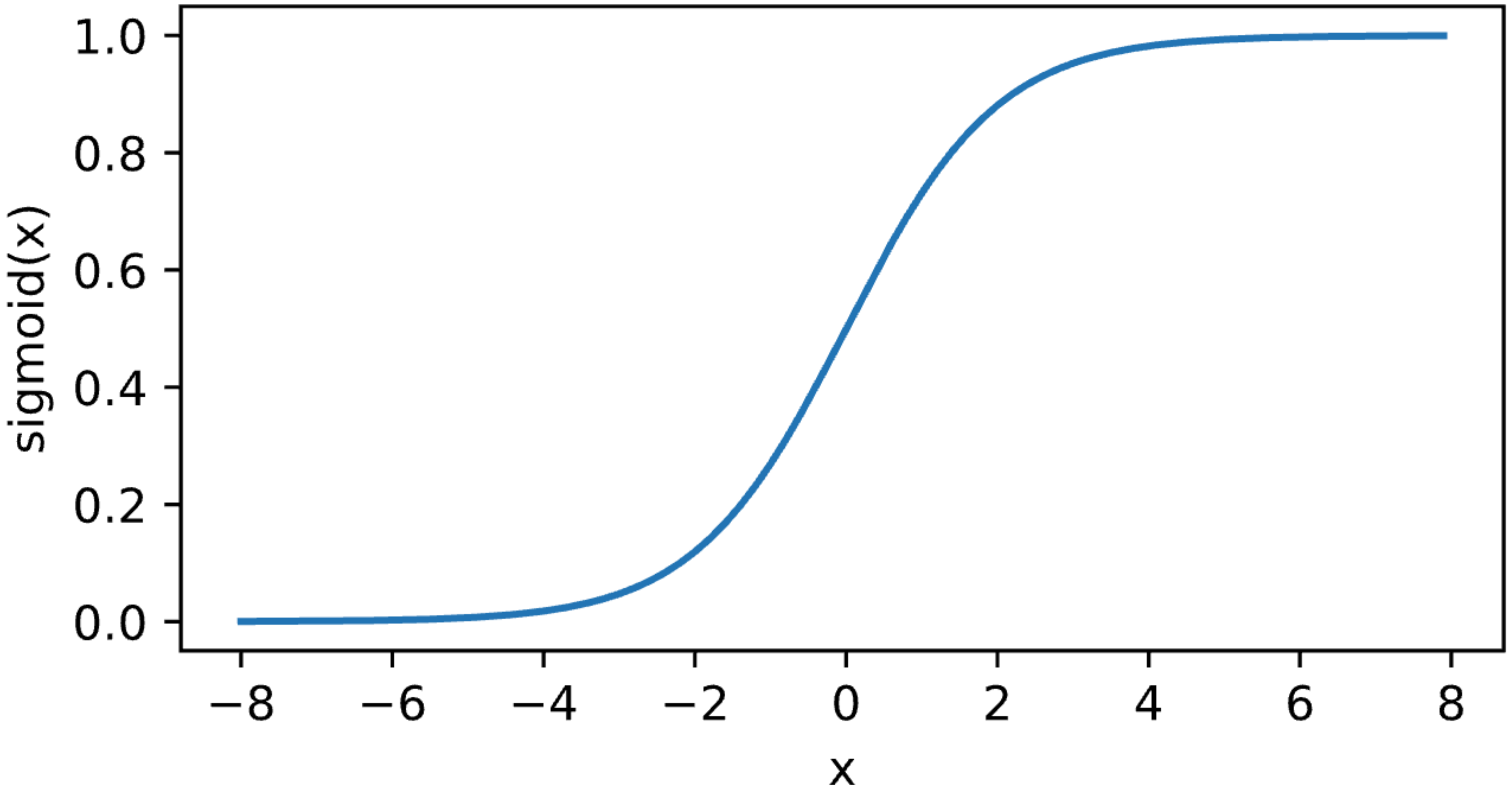
$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$



Sigmoid/Logistic Activation

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

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

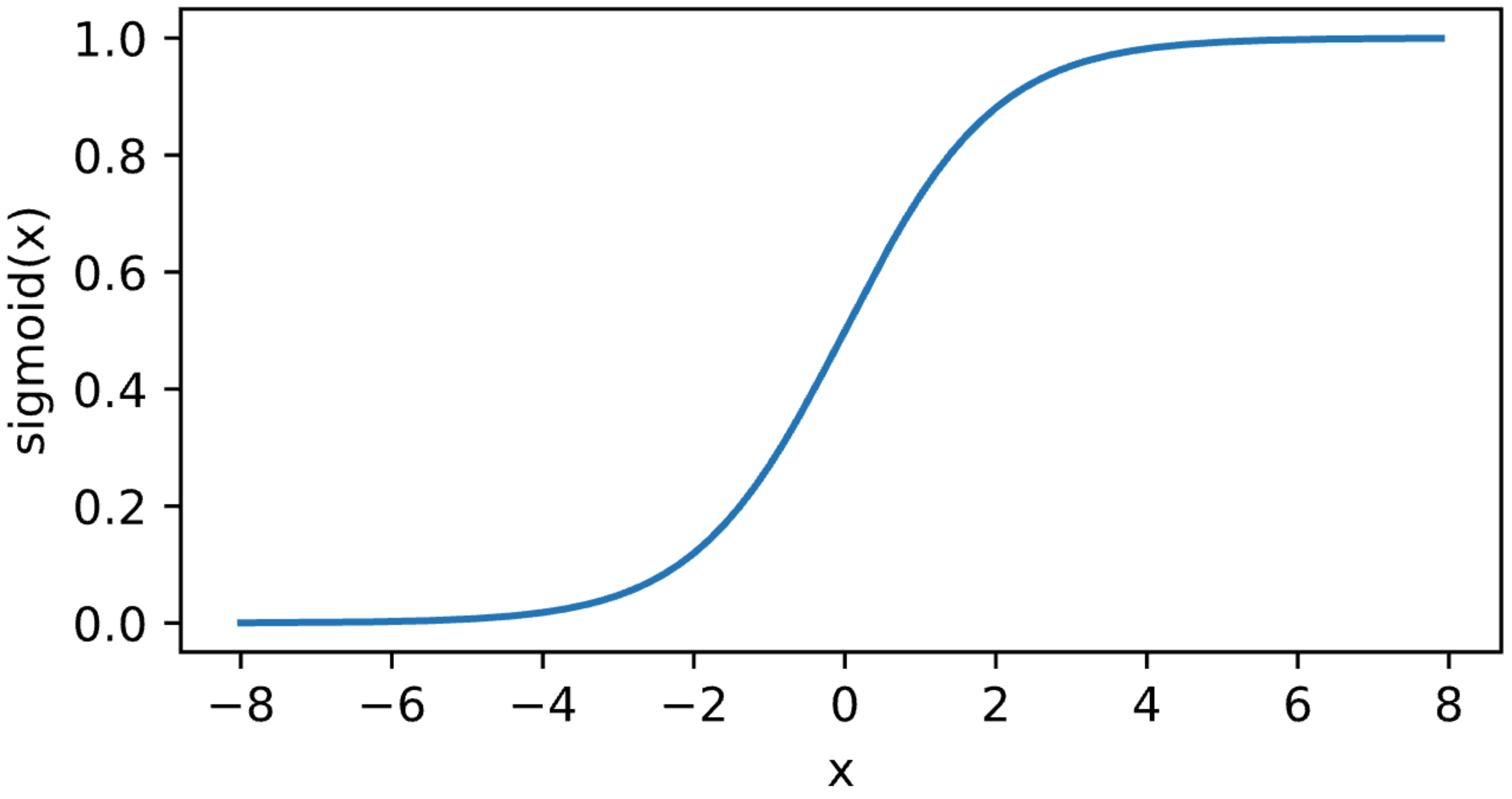


Logistic regression

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

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

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



Logistic regression

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

Training: maximize likelihood estimate (on the conditional probability)

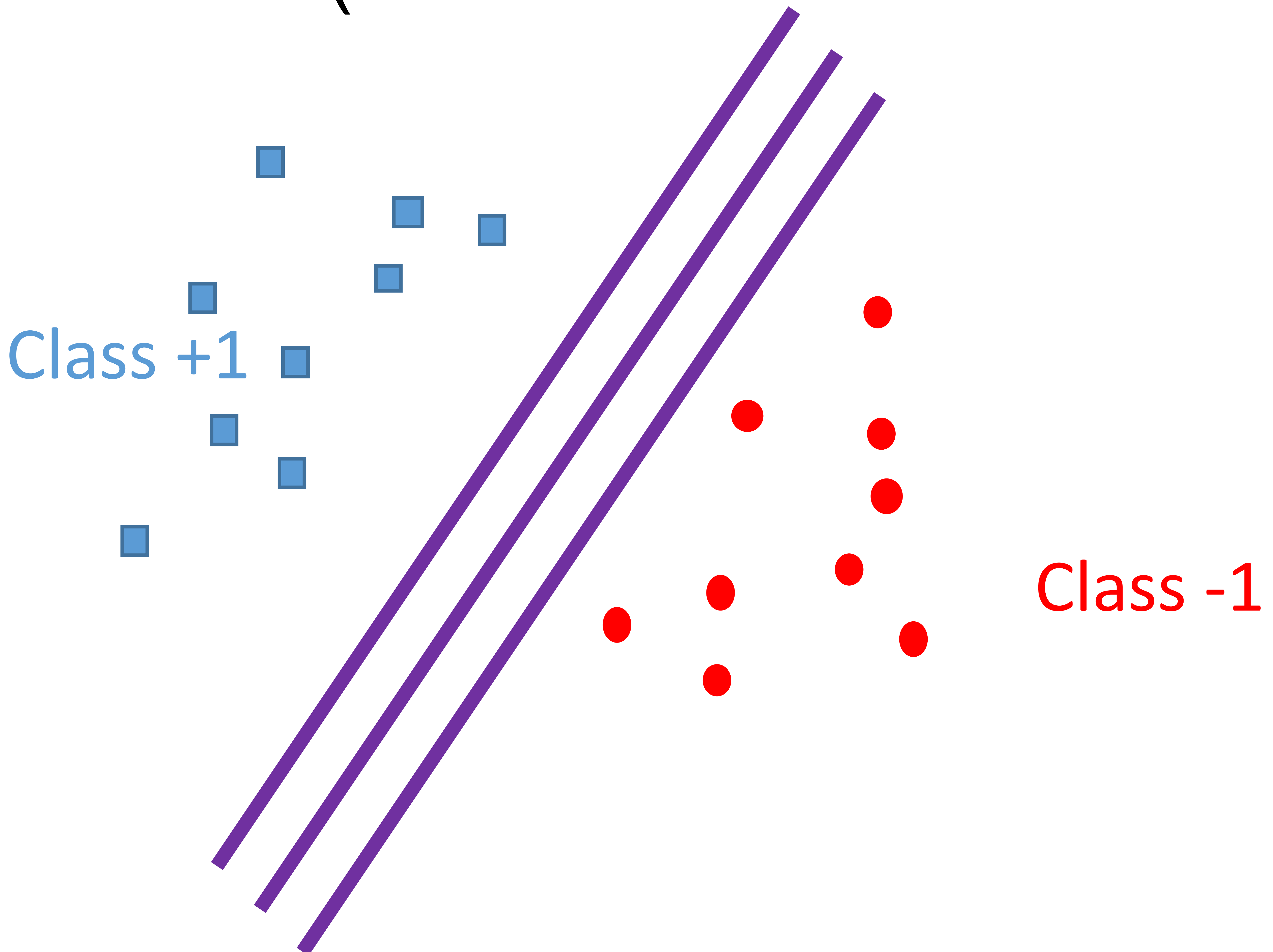
$$\max_{\mathbf{w}} \sum_i \log \frac{1}{1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i)}$$

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)

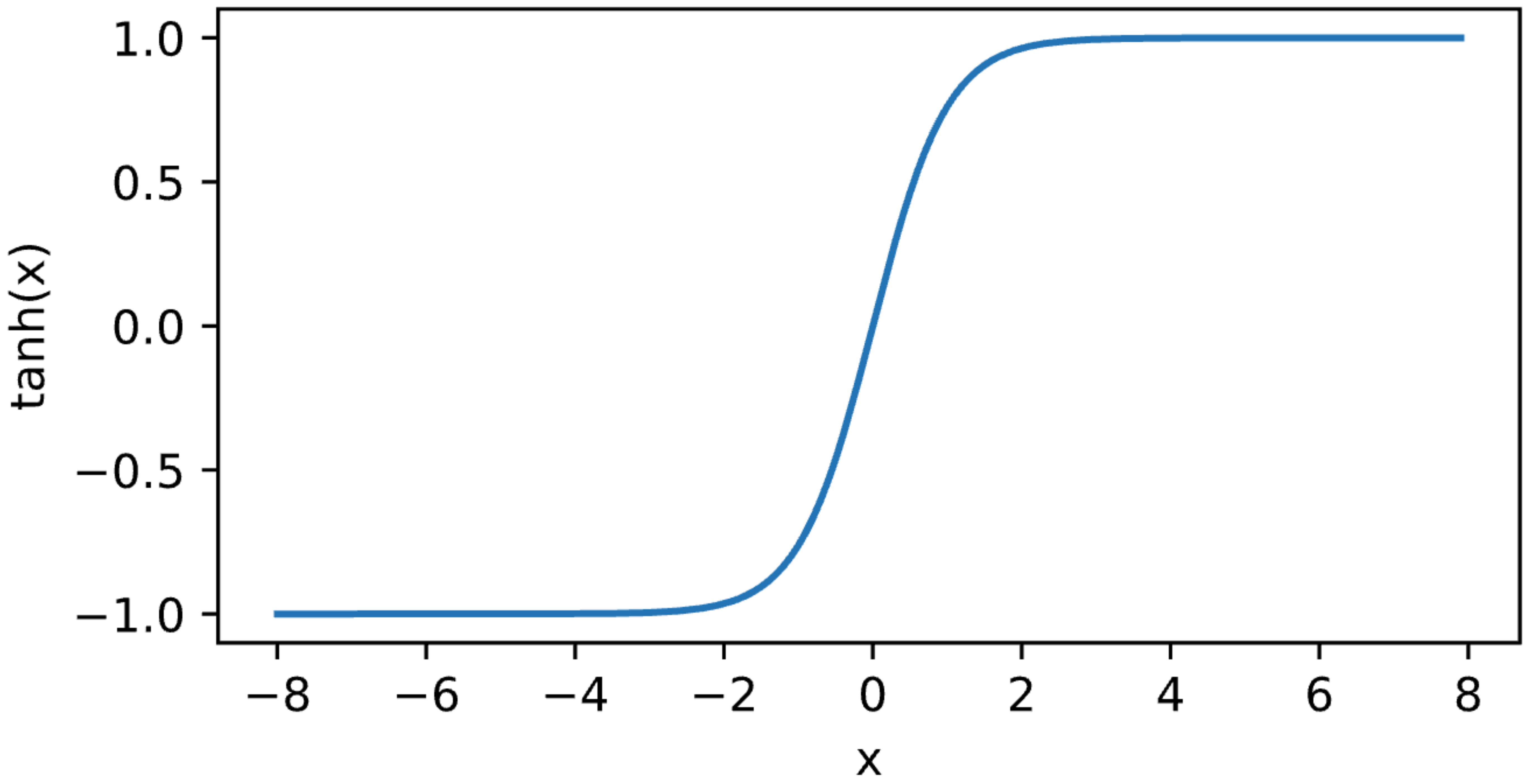
$$\min_{\mathbf{w}} \sum_i -\log \frac{1}{1+\exp(-y_i \mathbf{w}^T \mathbf{x}_i)} + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- Convex optimization
- Solve via (stochastic) gradient descent

Tanh Activation

Map inputs into (-1, 1)

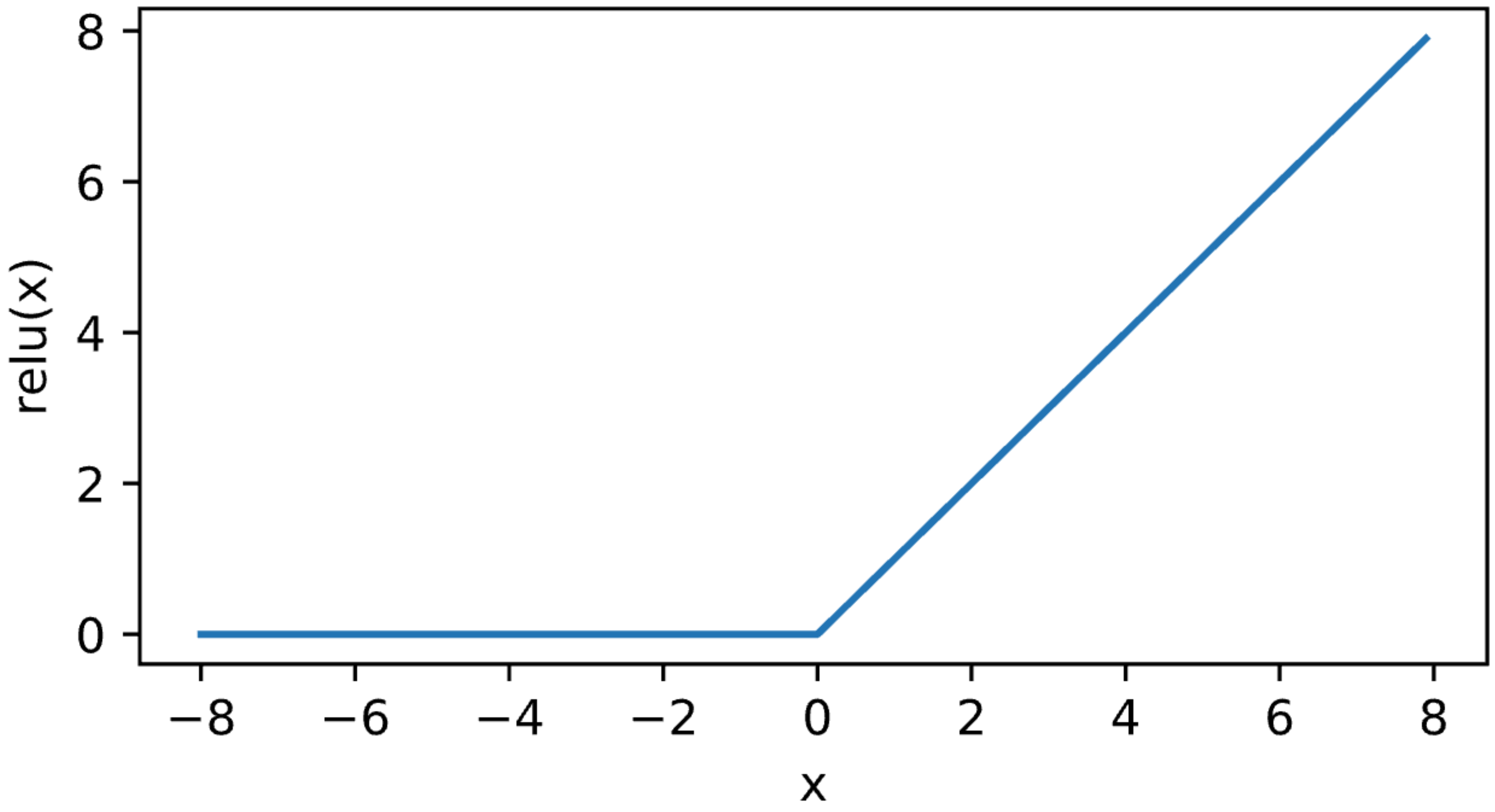
$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$



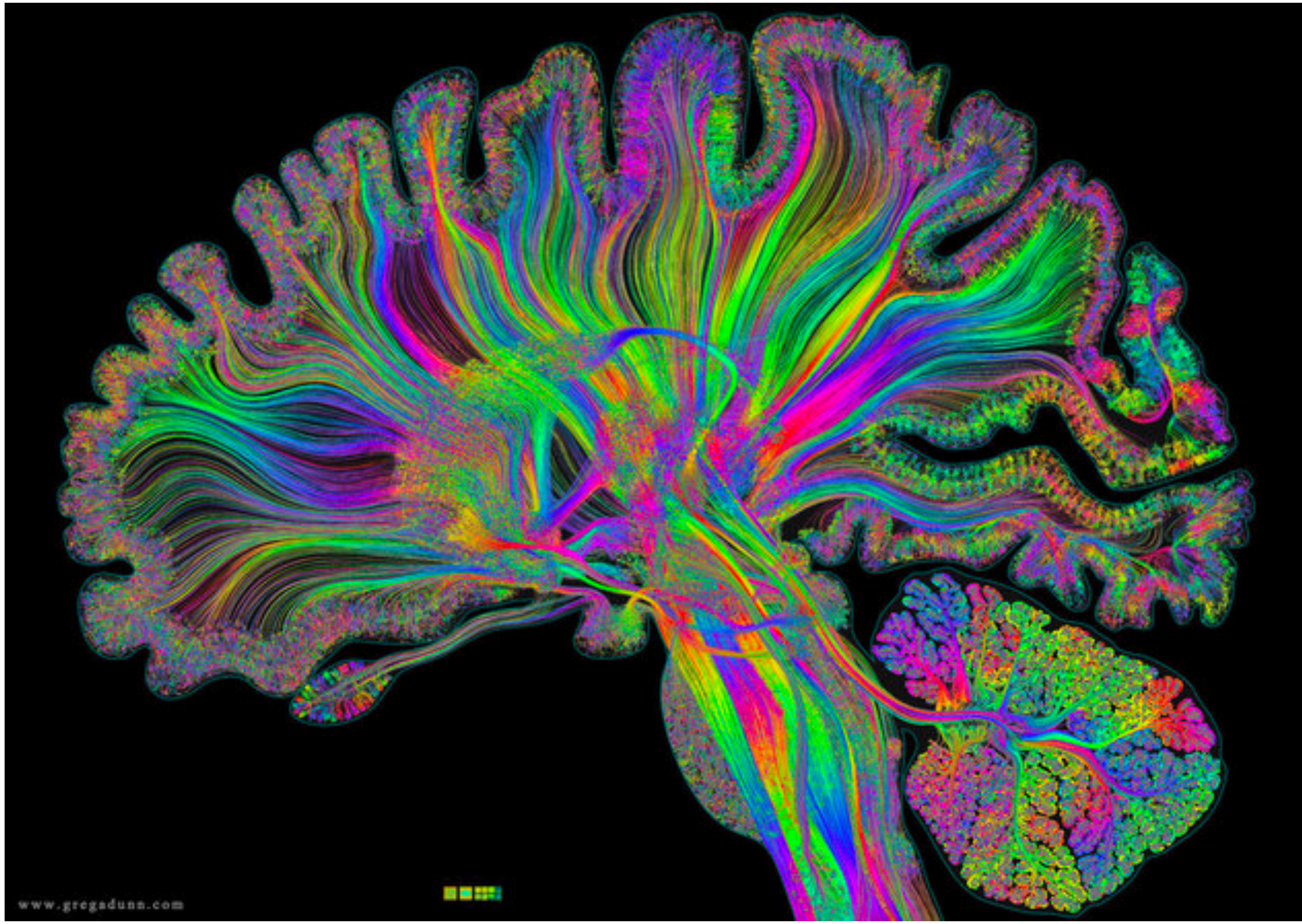
ReLU Activation

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

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

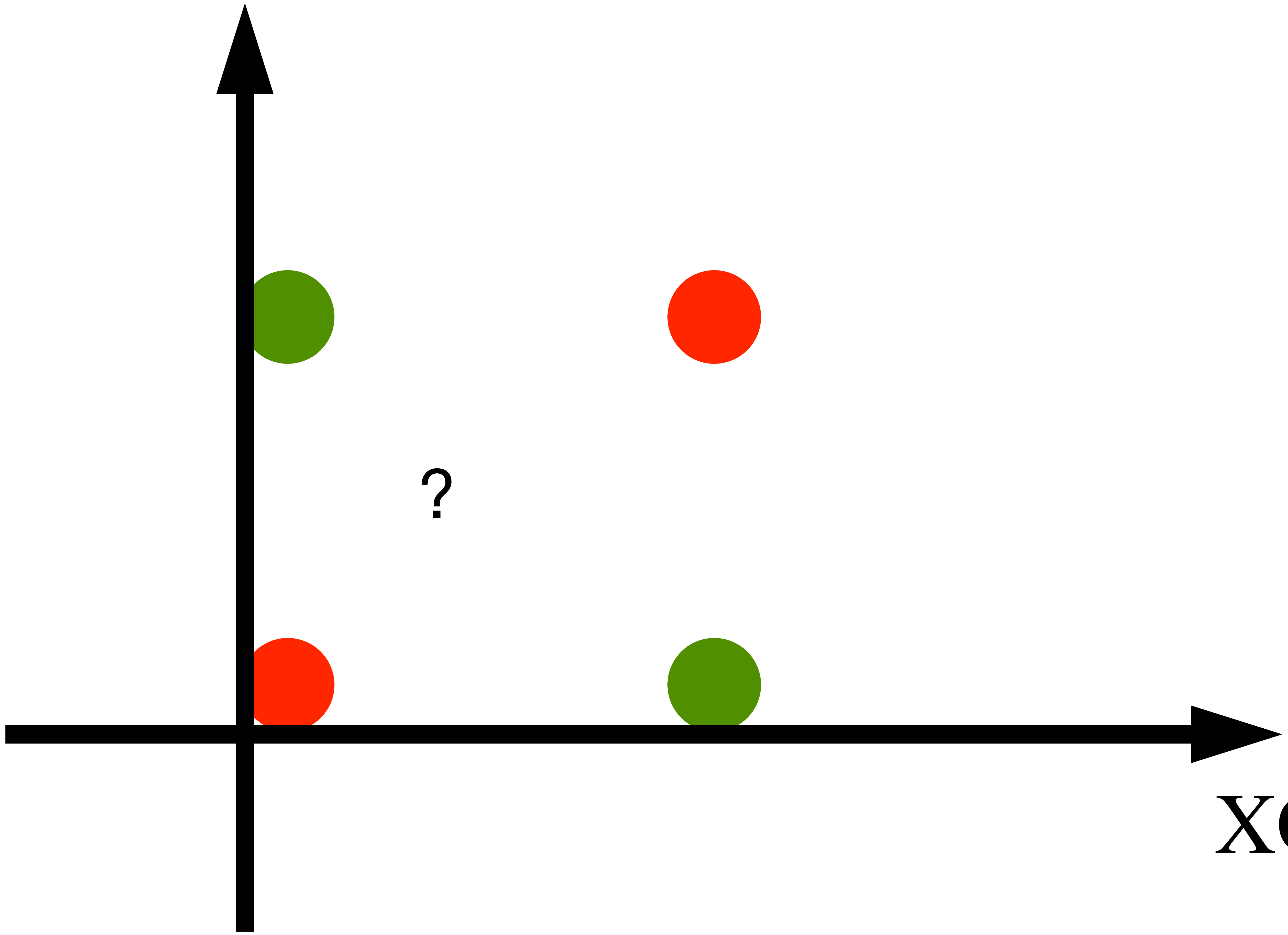


Multilayer Perceptron



The limited power of a single neuron

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



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

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

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

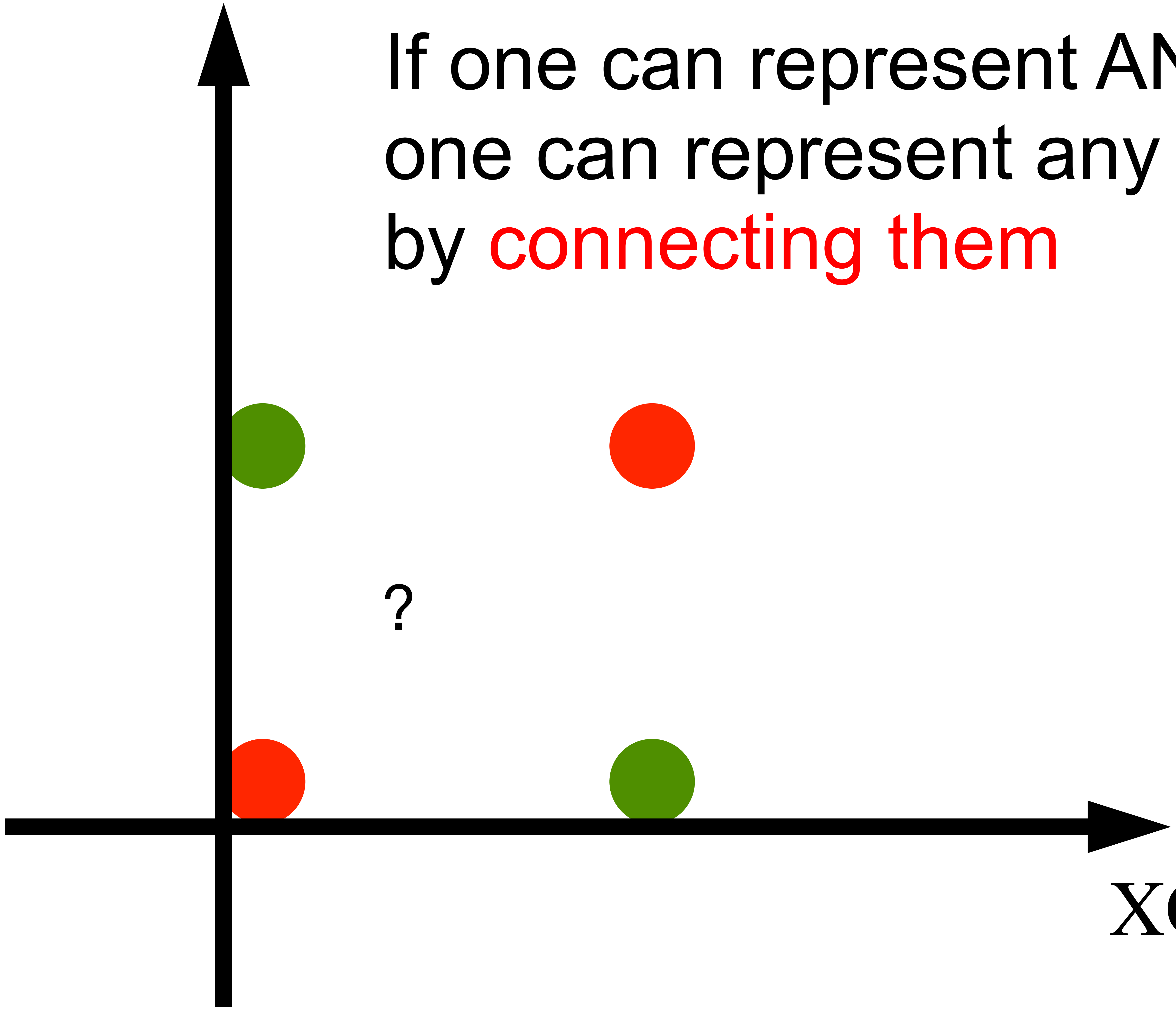
$$x_1 = 0, x_2 = 0, y = 0$$

$$\mathrm{XOR}(x1, x2) = (x1 \land \neg x2) \lor (\neg x1 \land x2)$$

The limited power of a single neuron

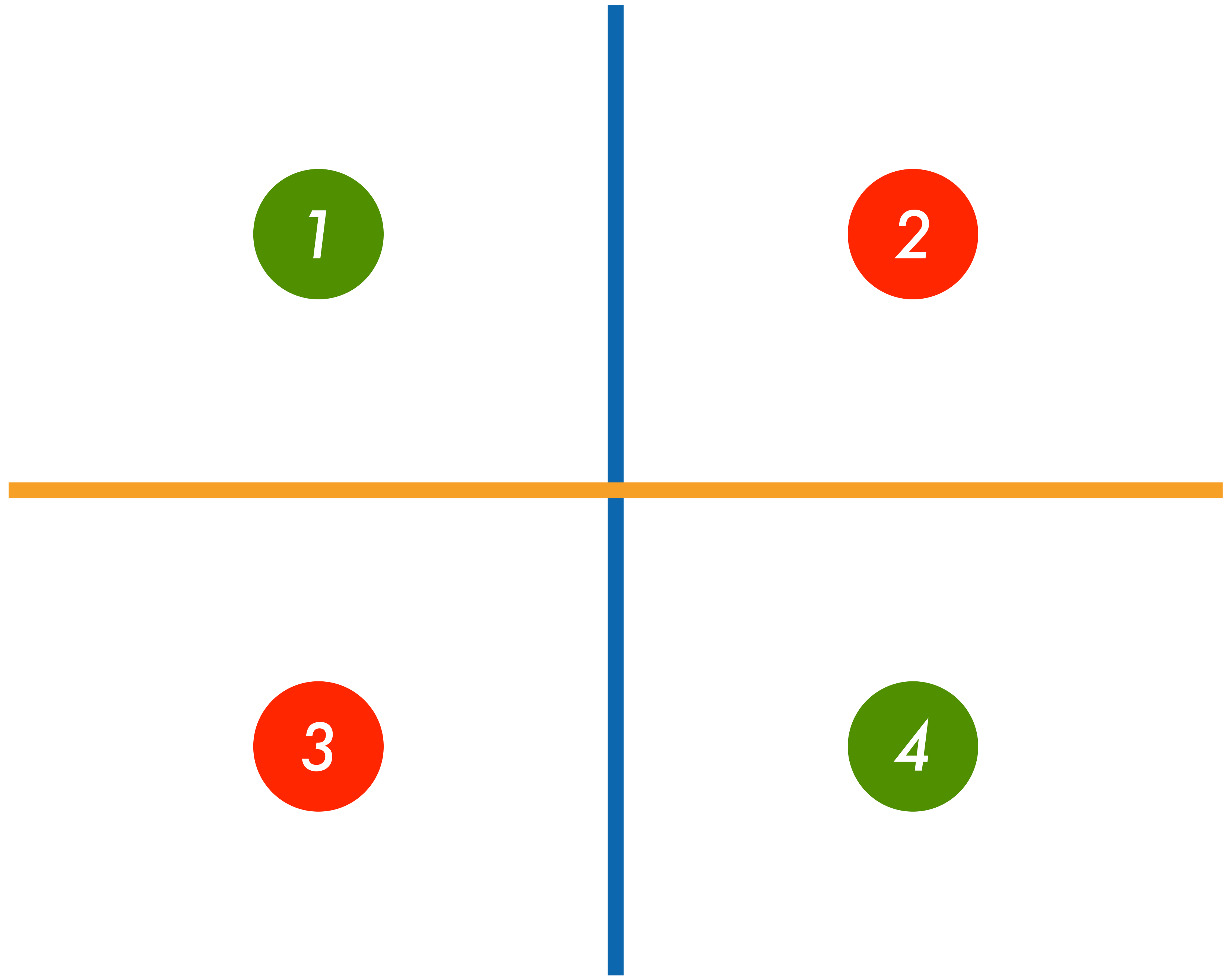
XOR problem

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

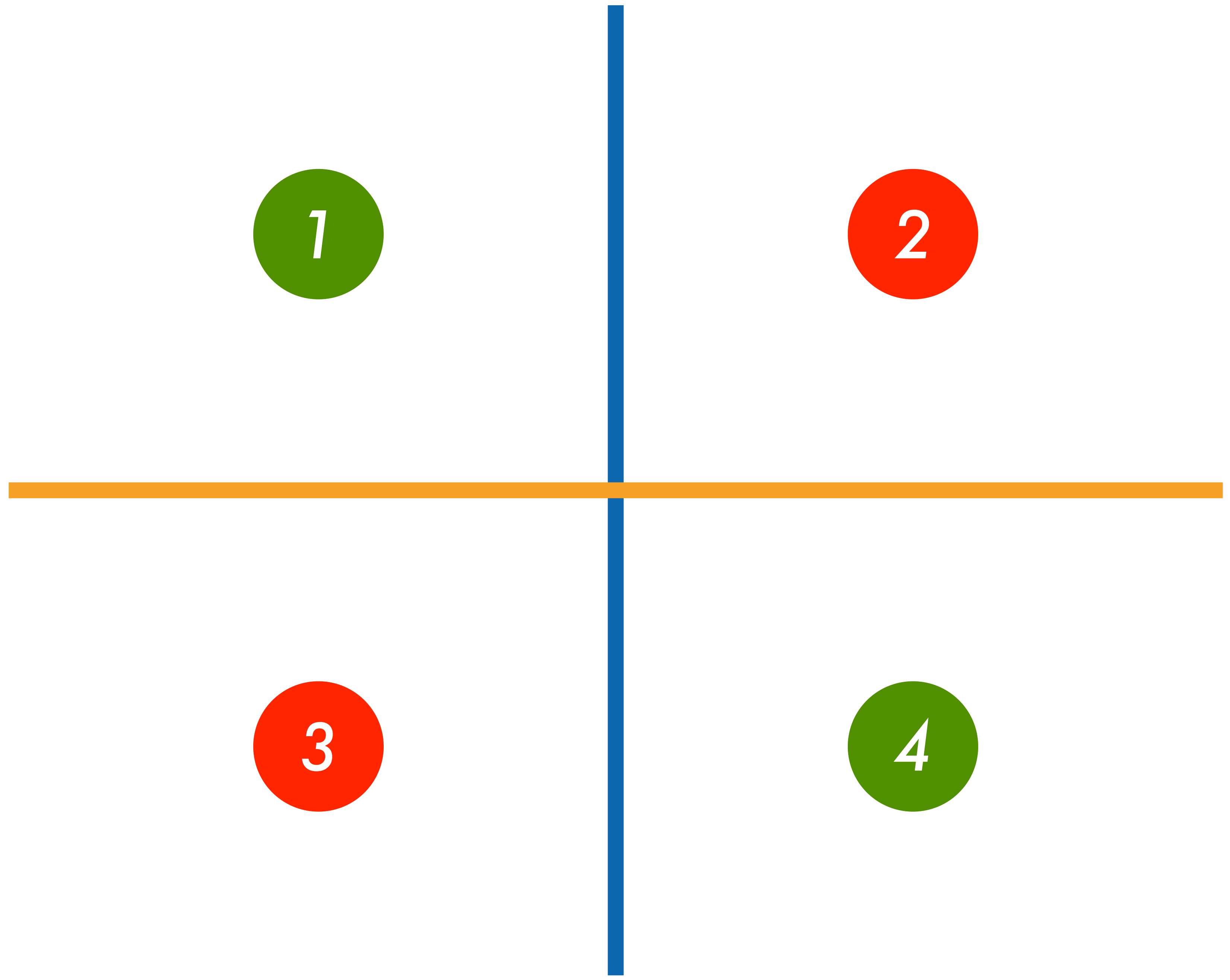


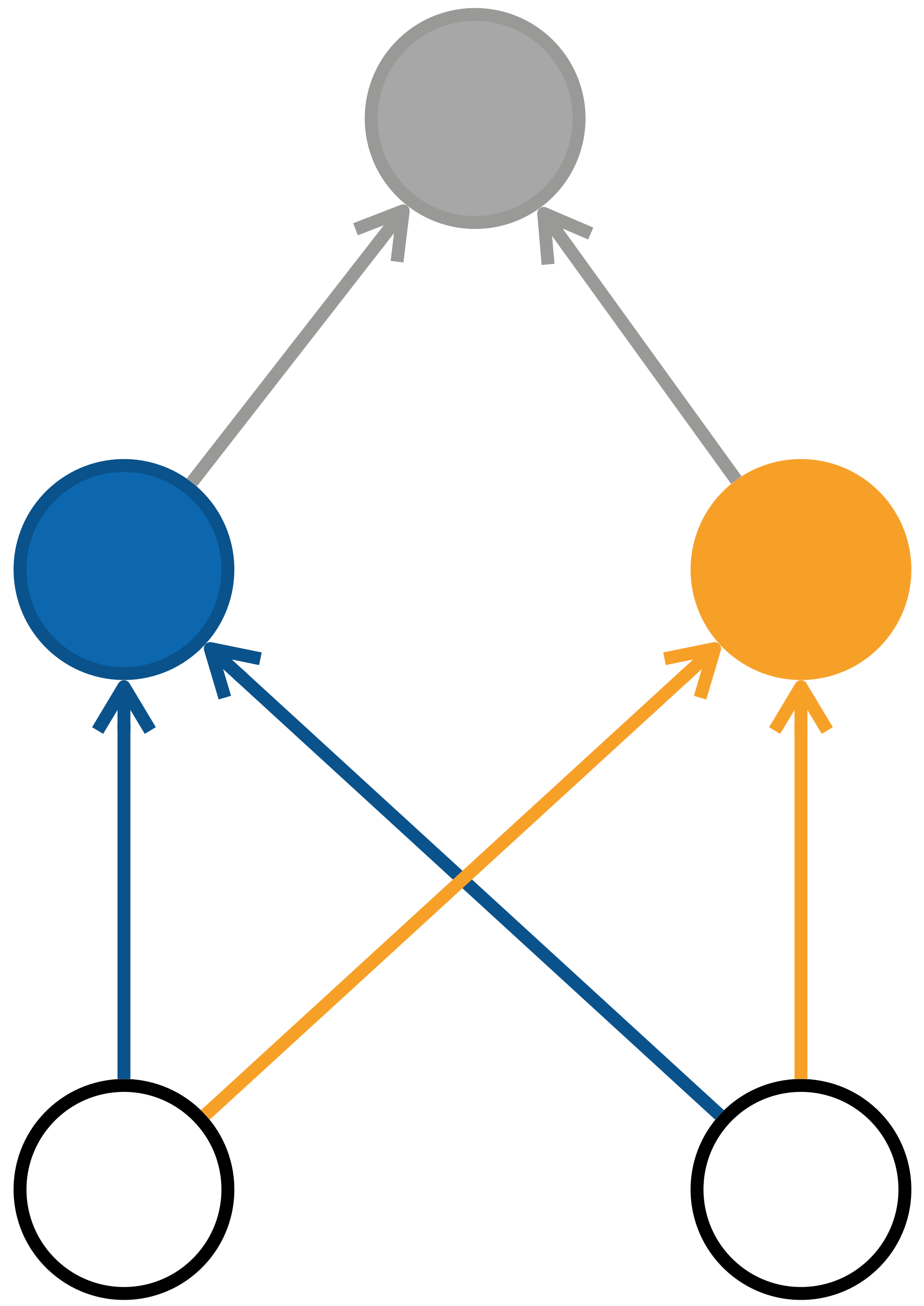
$$XOR(x1,x2) = (x1 \wedge \neg x2) \vee (\neg x1 \wedge x2)$$

Learning XOR



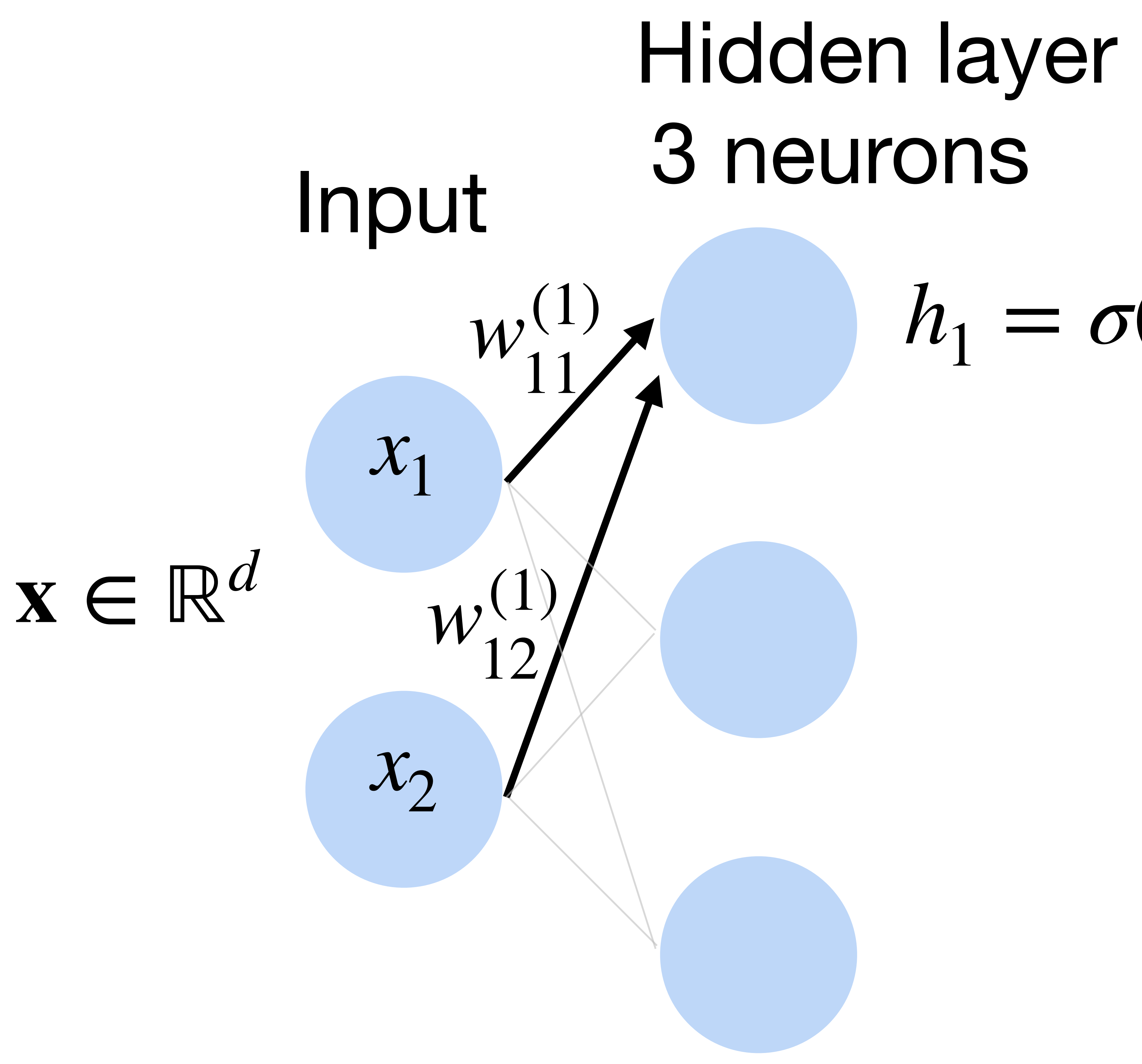
Learning XOR





Multi-layer perceptron: Example

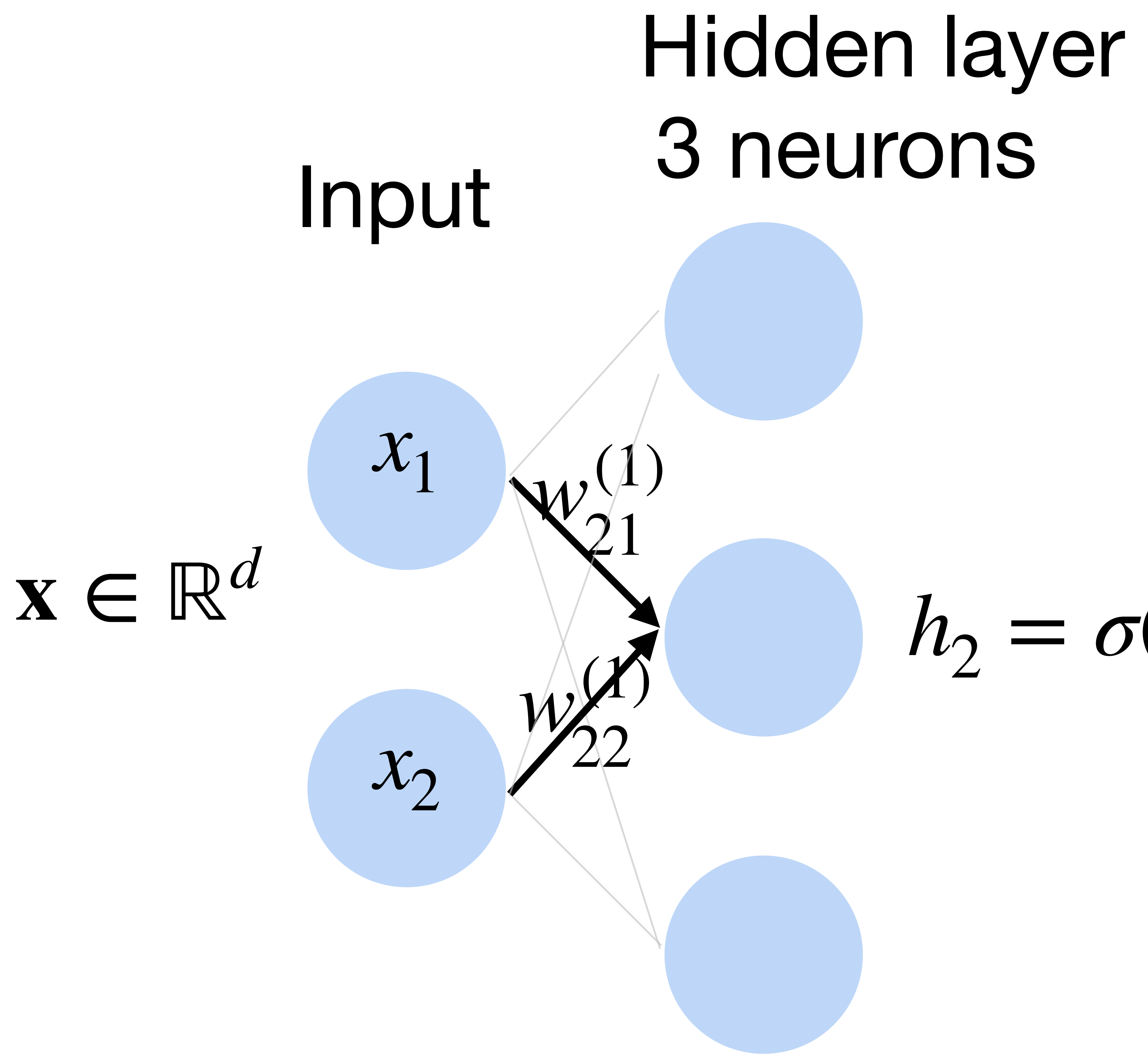
- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



$$h_1 = \sigma(\sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1)$$

Multi-layer perceptron: Example

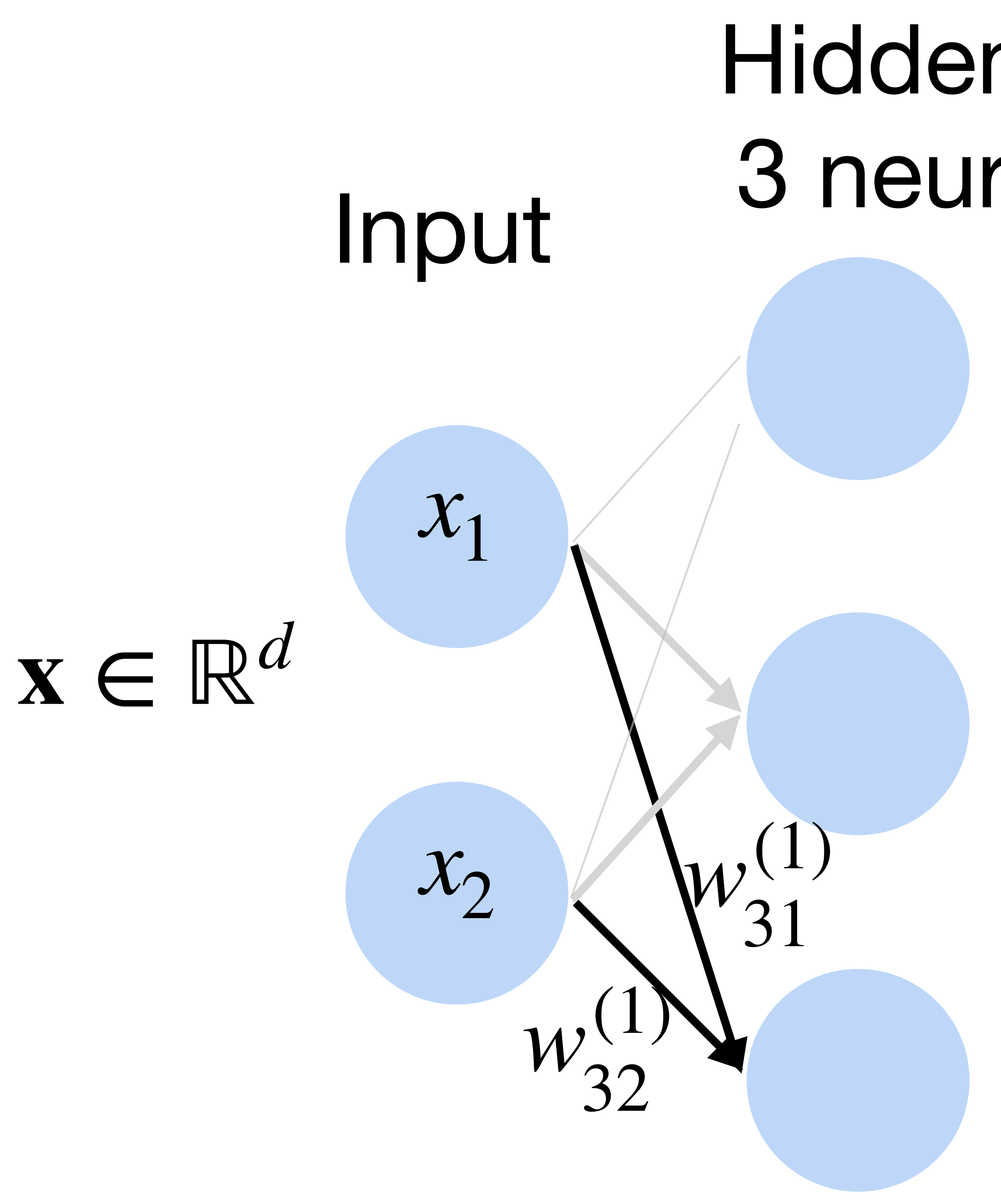
- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



$$h_2 = \sigma\left(\sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2\right)$$

Multi-layer perceptron: Example

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



$$h_3 = \sigma\left(\sum_{i=1}^{d} x_i w_{3i}^{(1)} + b_3\right)$$

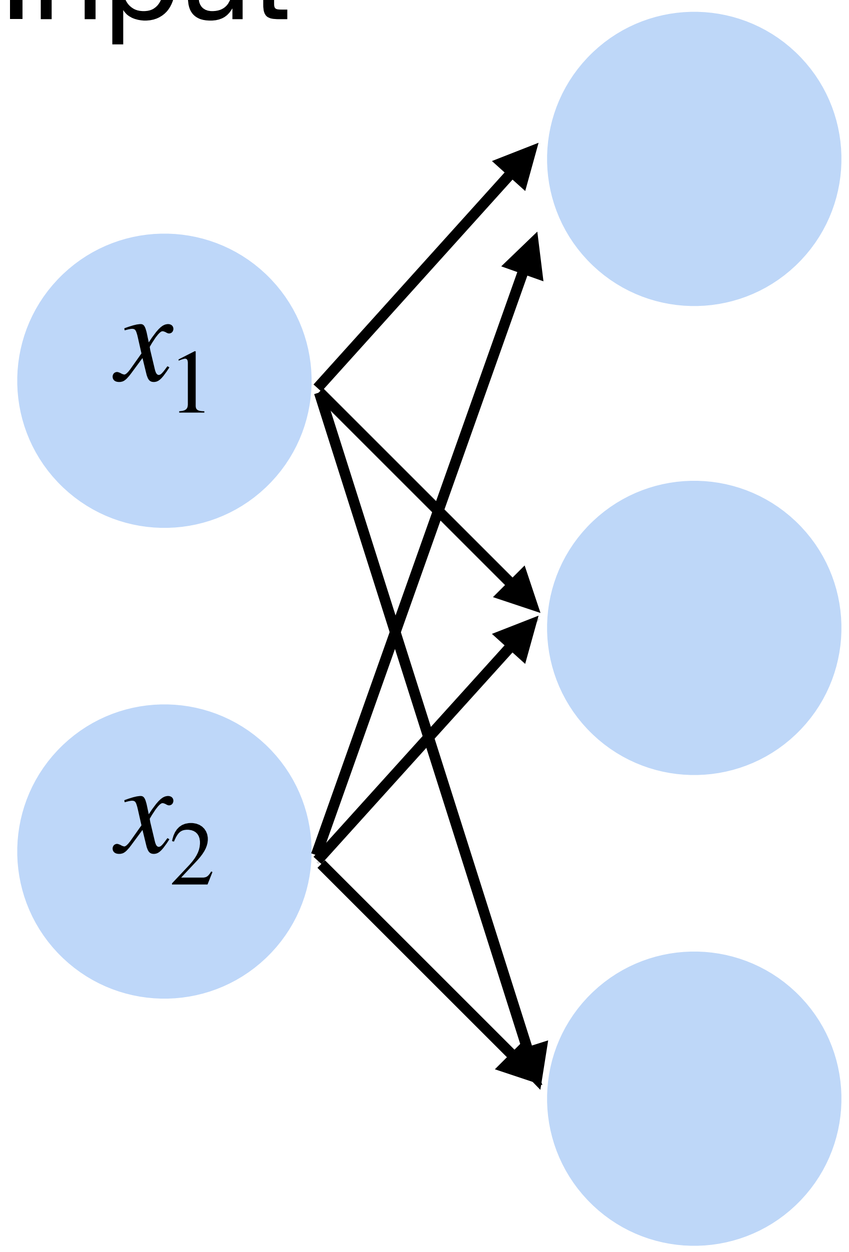
Multi-layer perceptron: Example

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

Hidden layer
m=3 neurons

Input

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



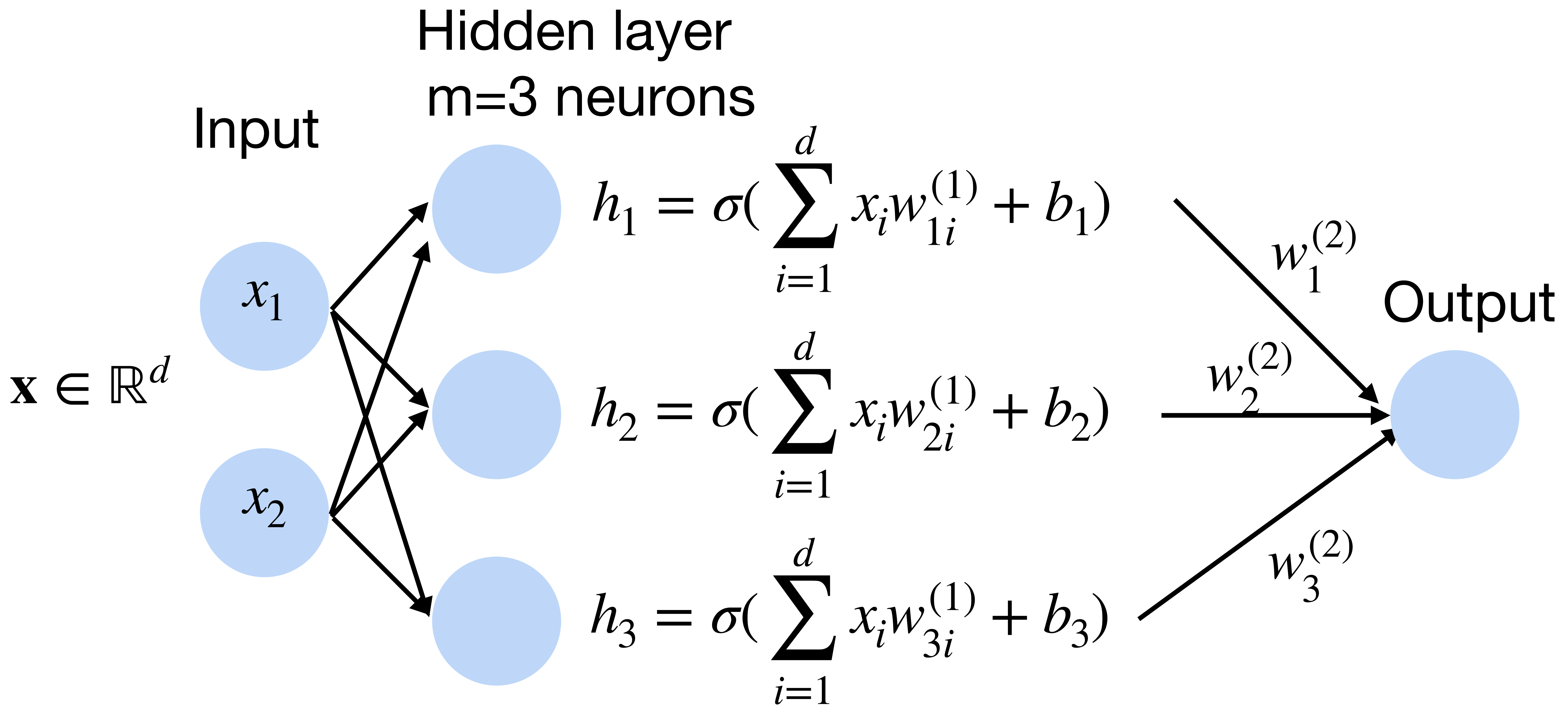
$$h_1 = \sigma\left(\sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1\right)$$

$$h_2 = \sigma\left(\sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2\right)$$

$$h_3 = \sigma\left(\sum_{i=1}^{d} x_i w_{3i}^{(1)} + b_3\right)$$

Multi-layer perceptron: Example

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



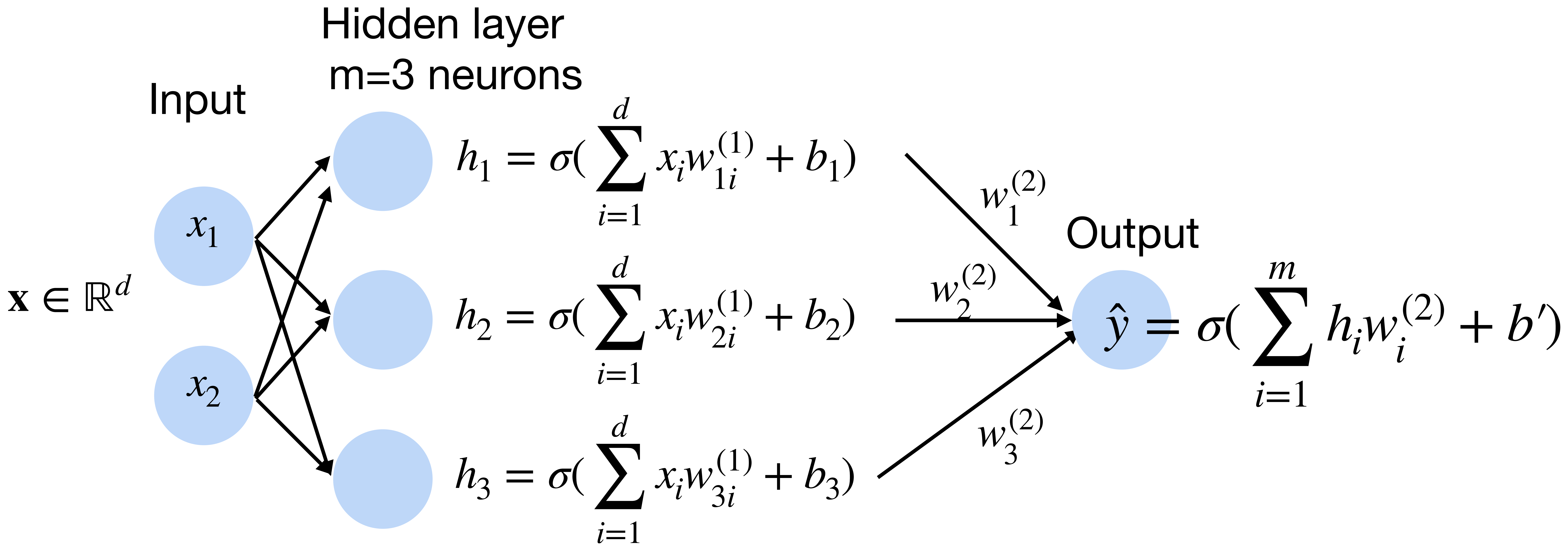
$$h_1 = \sigma(\sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1)$$

$$h_2 = \sigma(\sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2)$$

$$h_3 = \sigma(\sum_{i=1}^{d} x_i w_{3i}^{(1)} + b_3)$$

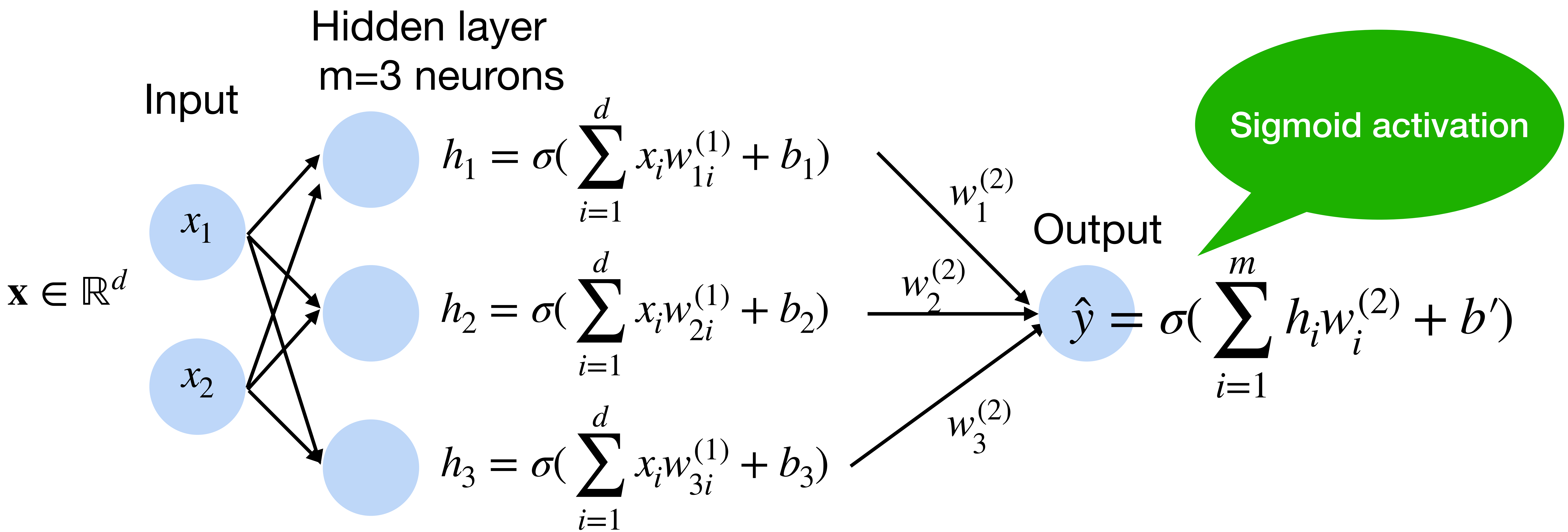
Multi-layer perceptron: Example

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



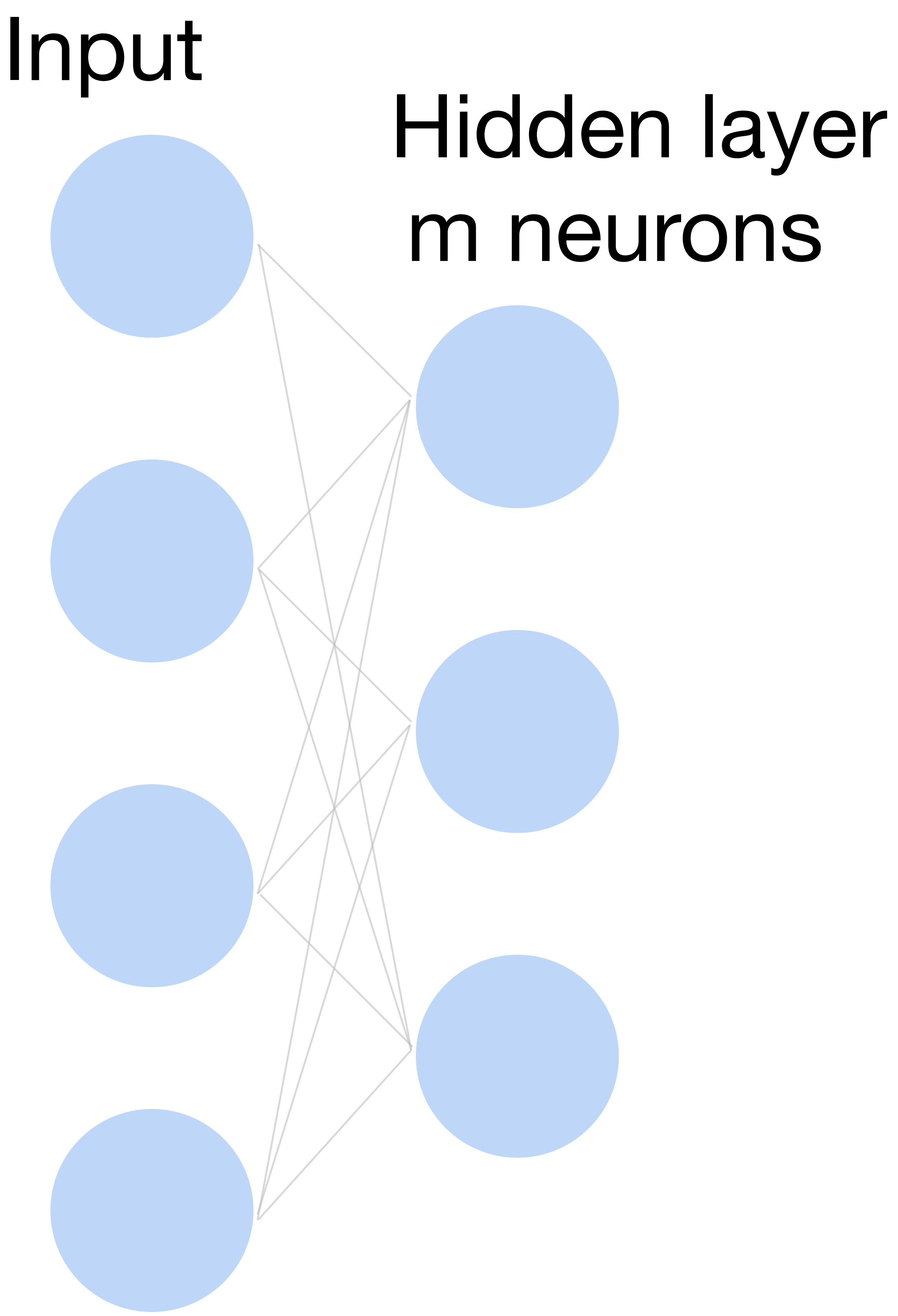
Multi-layer perceptron: Example

- 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

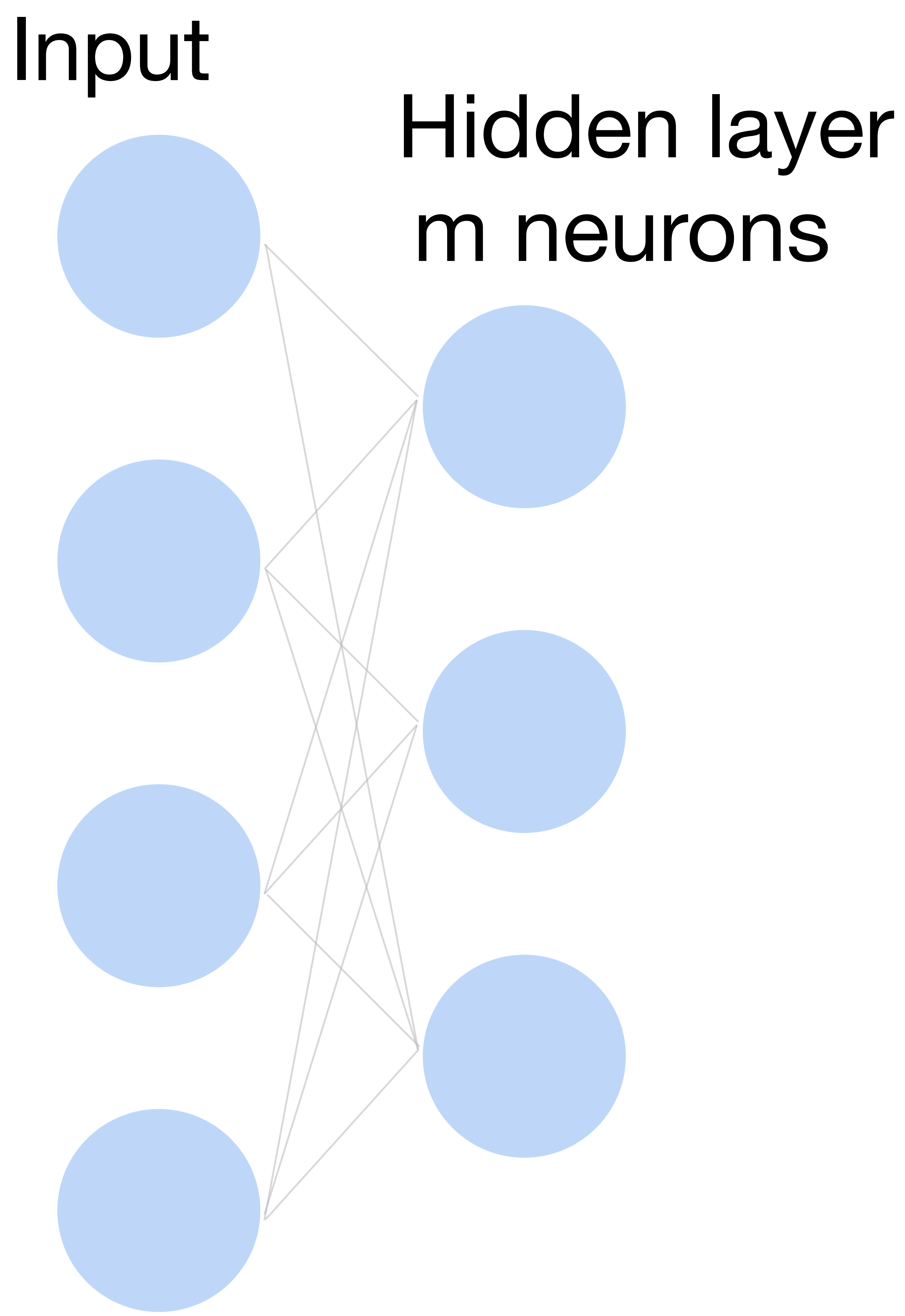


Multi-layer perceptron: Matrix Notation

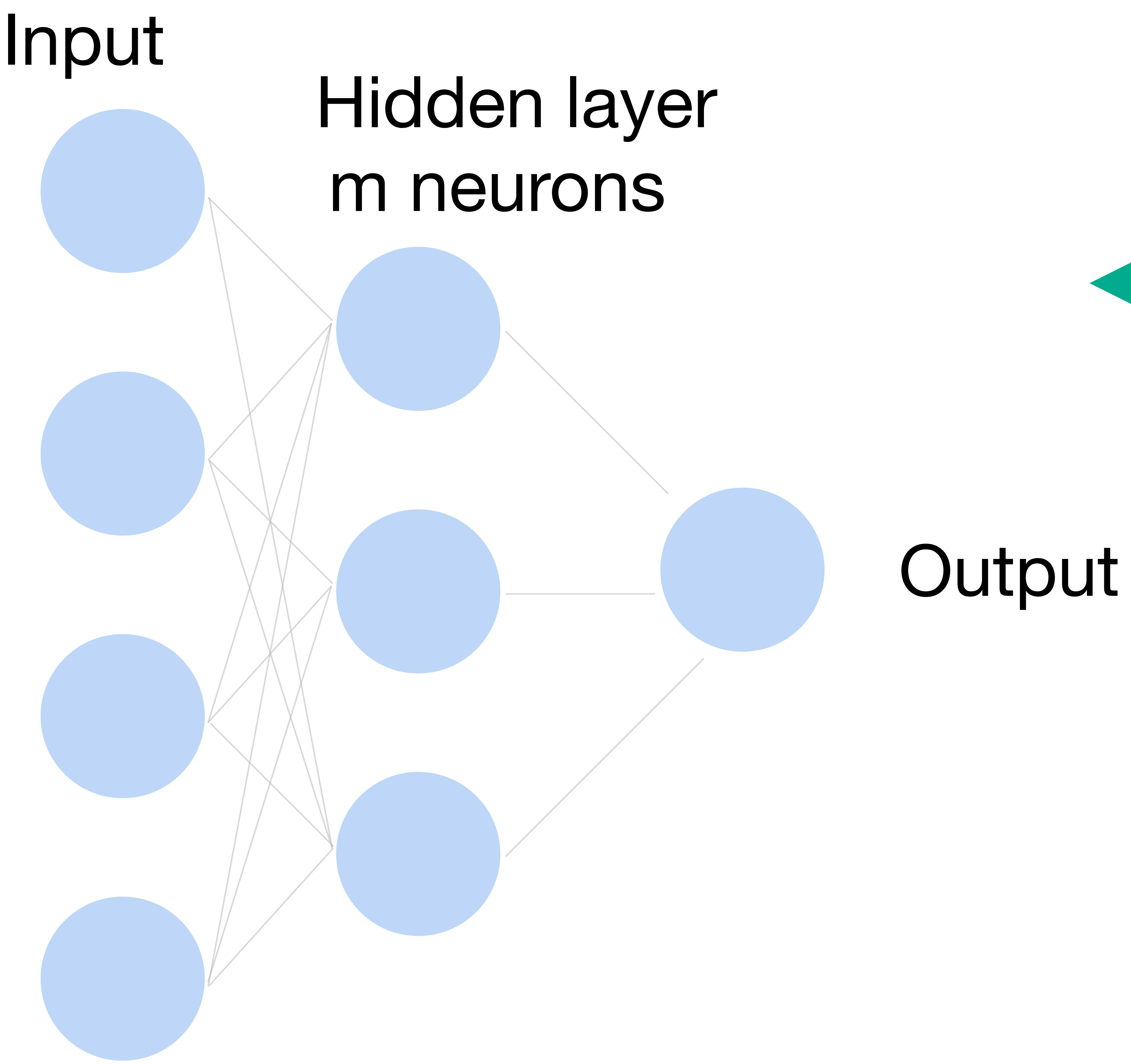
- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$
- Intermediate output

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

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

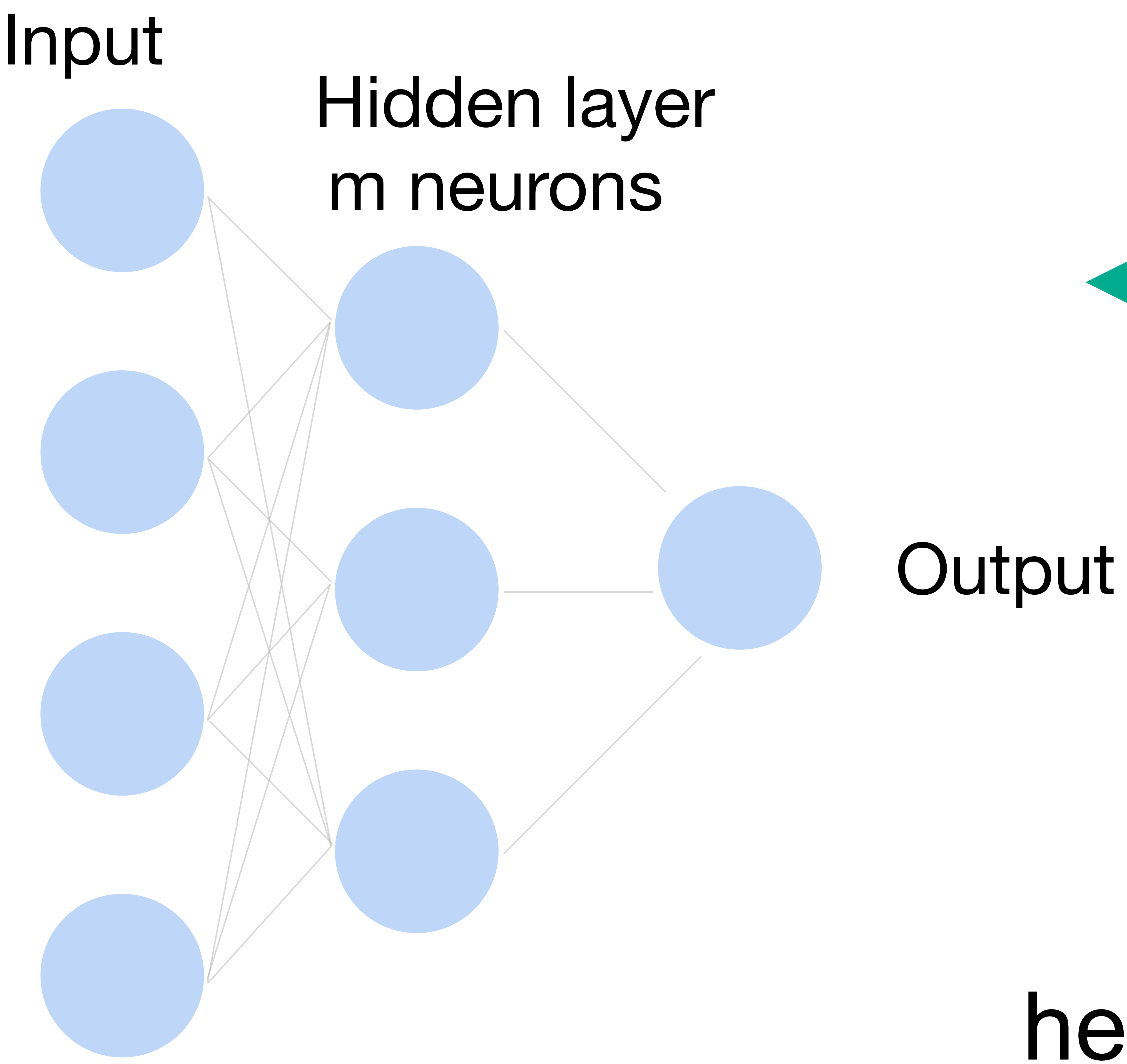


Multi-layer perceptron



Why do we need an a nonlinear activation?

Multi-layer perceptron



Why do we need an a nonlinear activation?

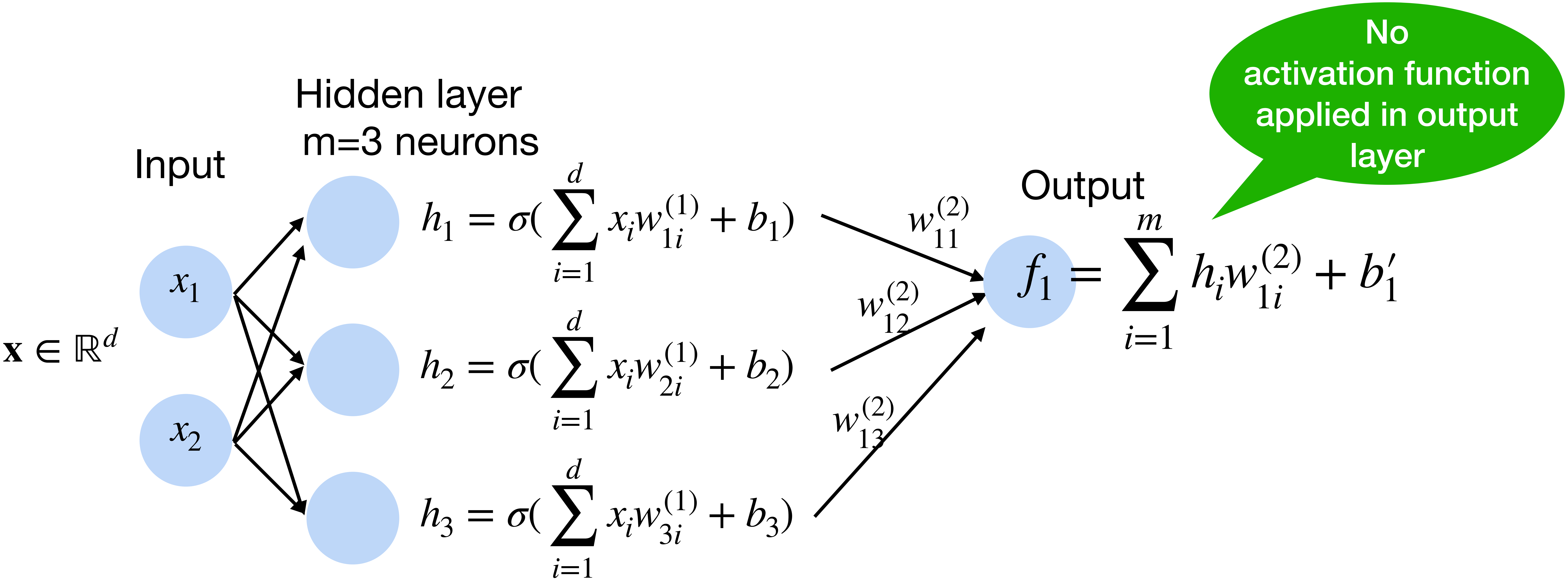
$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

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

hence $f = \mathbf{w}_2^\top \mathbf{W}\mathbf{x} + b'$

Neural network for k-way classification

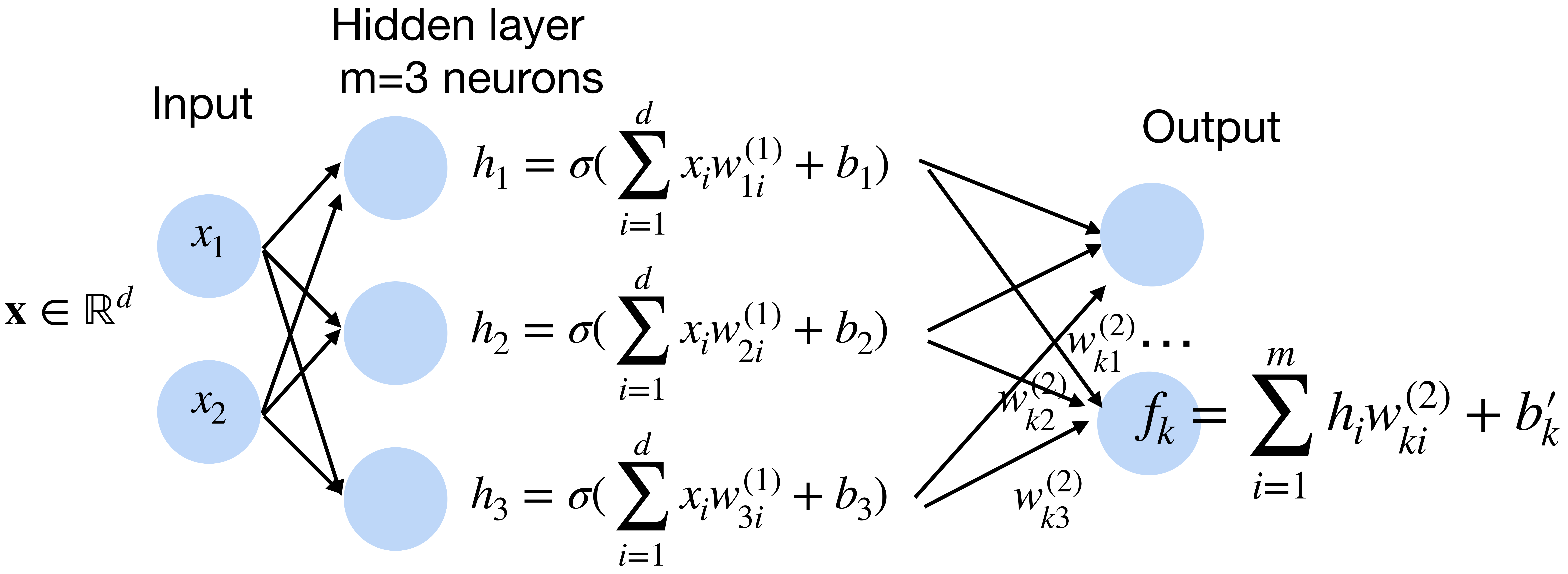
- K outputs in the final layer



Neural network for k-way classification

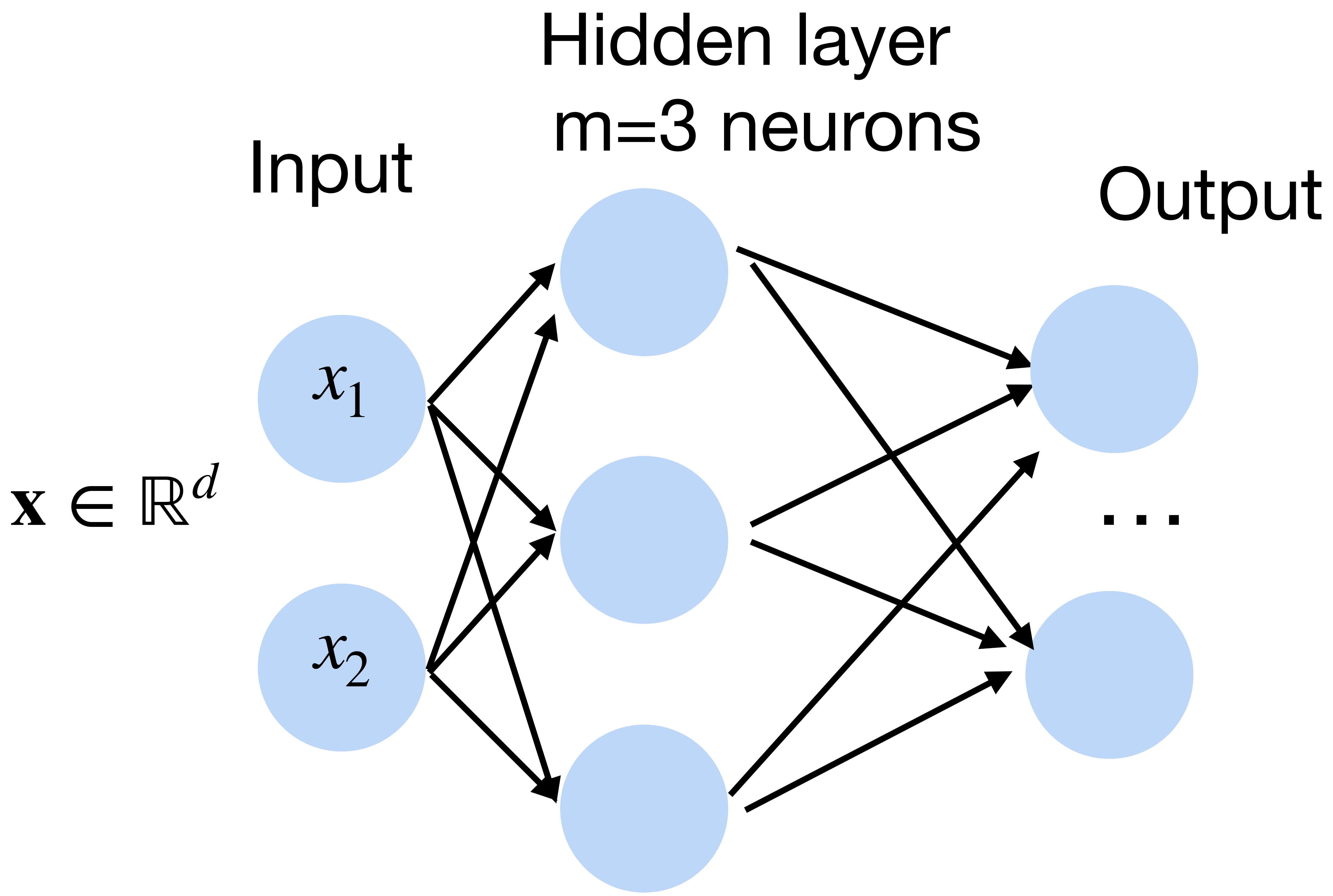
- K outputs units in the final layer

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



Softmax

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

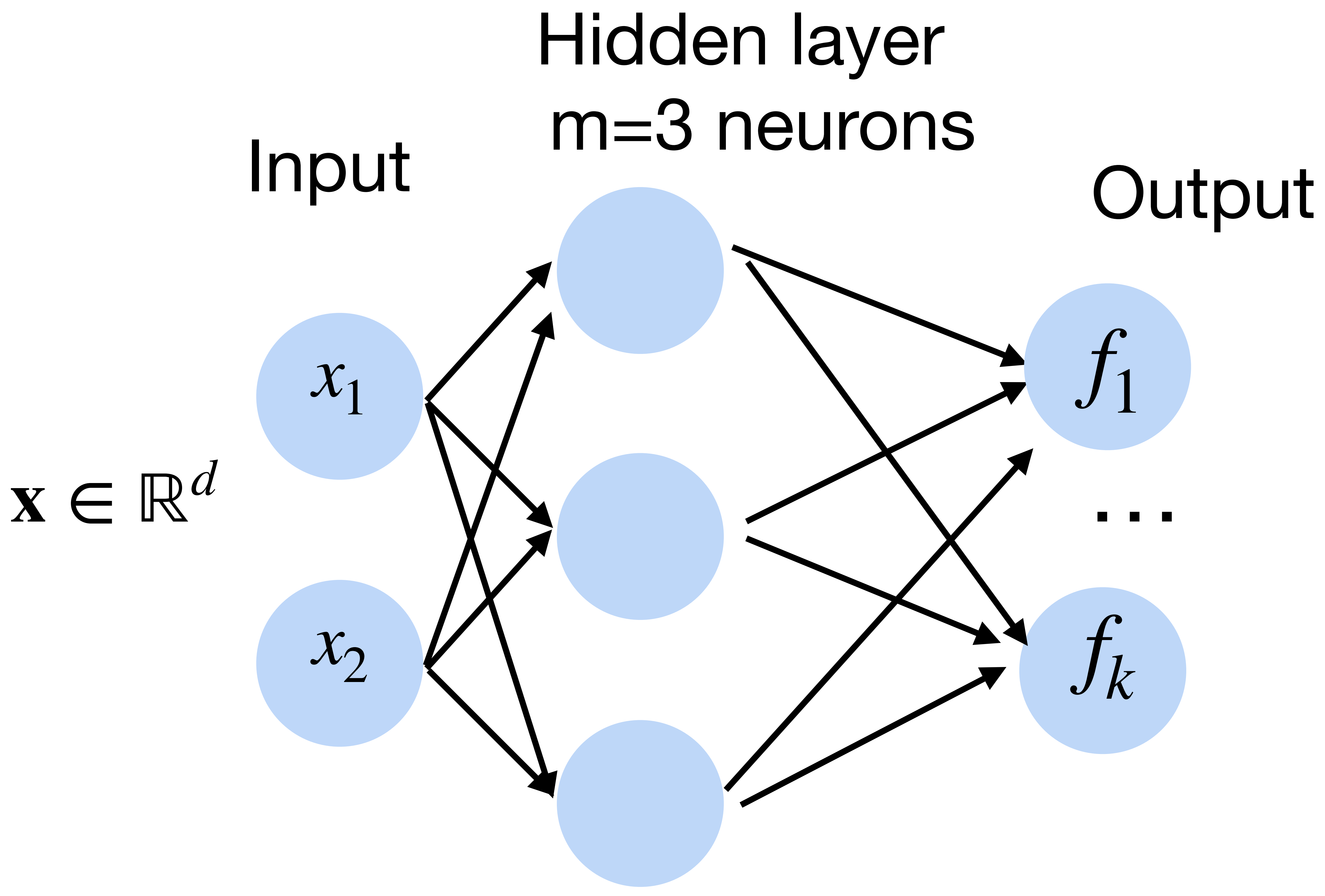


$$p(y \mid \mathbf{x}) = \text{softmax}(f)$$

$$= \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)}$$

Softmax

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



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

$$= \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)}$$

Softmax

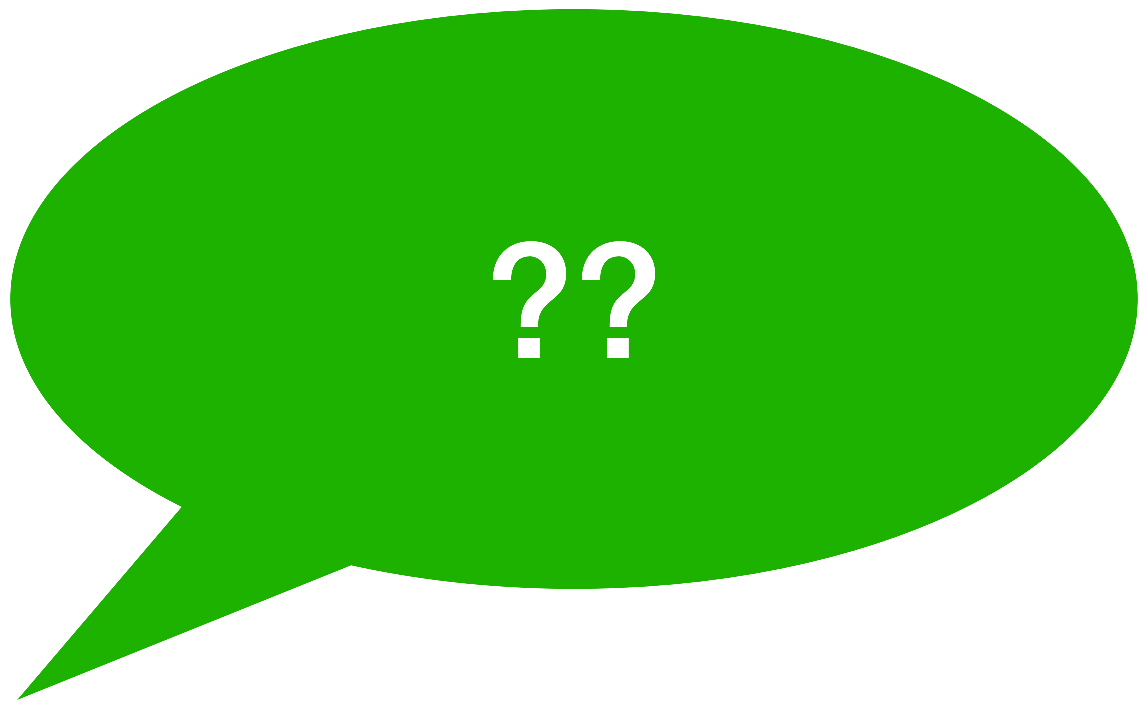
Turns outputs f into probabilities (sum up to 1 across k classes)

Output layer

$$\begin{bmatrix} 1.3 \\ 5.1 \\ 2.2 \\ 0.7 \\ 1.1 \end{bmatrix} \rightarrow$$

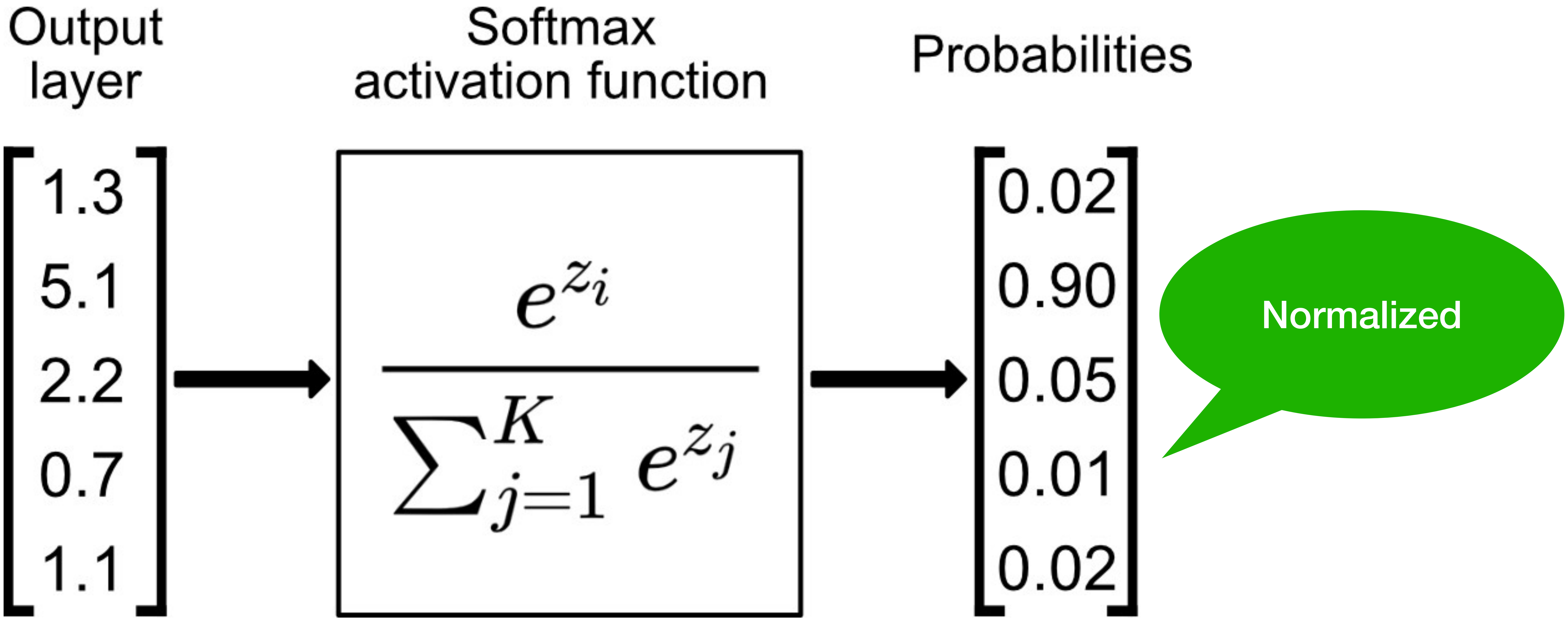
Softmax activation function

$$\boxed{\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}} \rightarrow$$



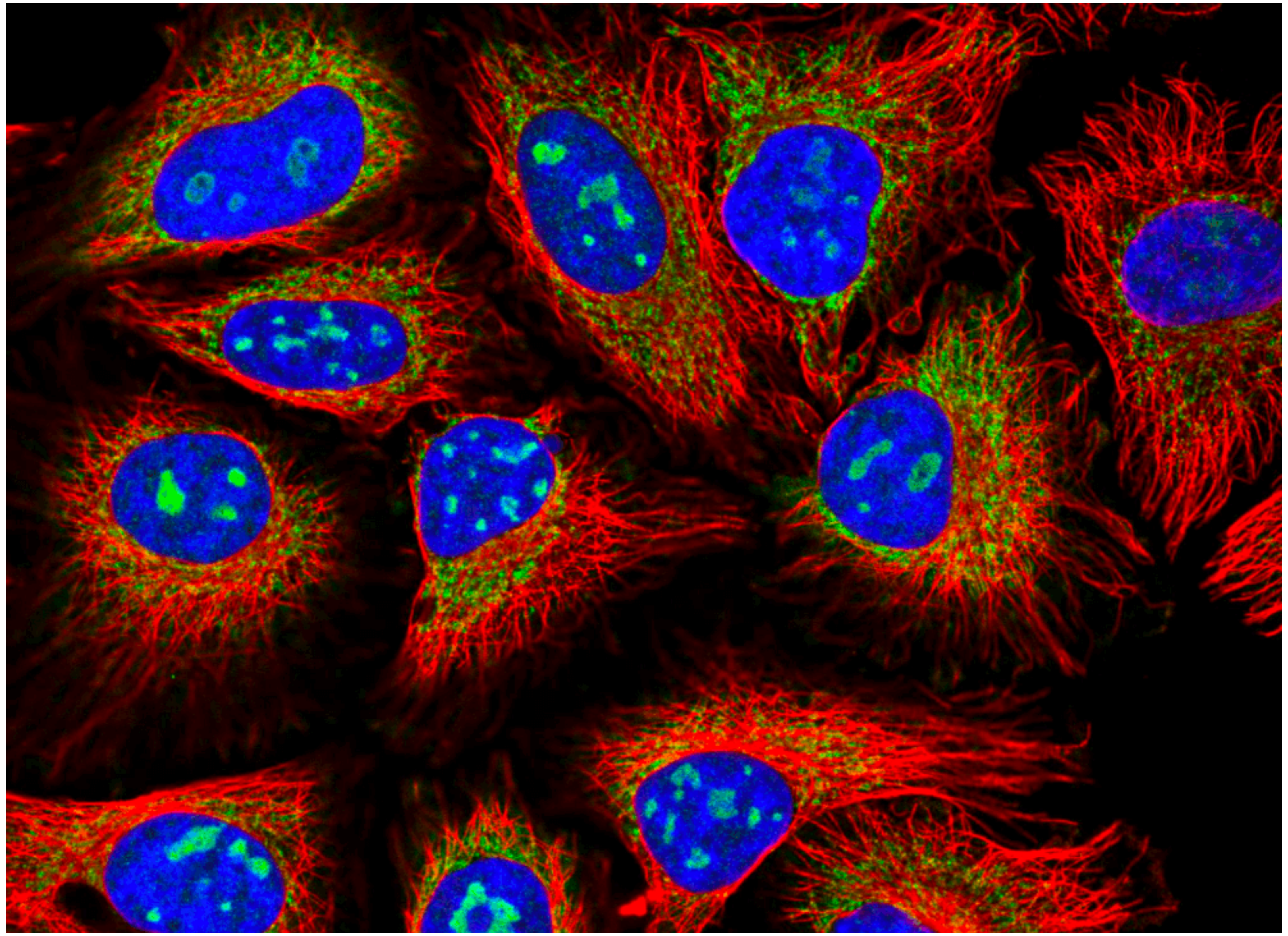
Softmax

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



Classification Tasks at Kaggle

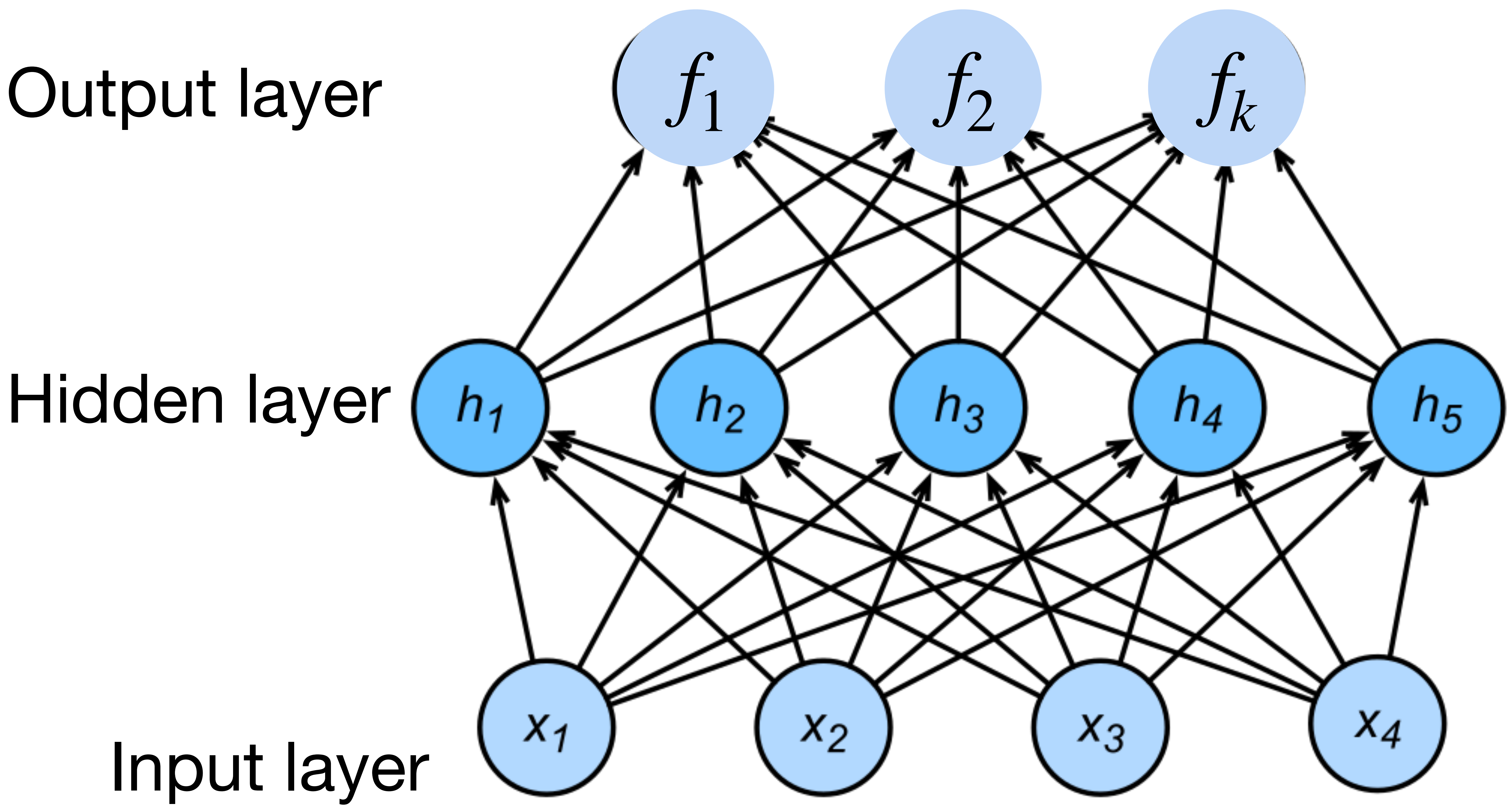
Classify human protein microscope images into 28 categories



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 bridg

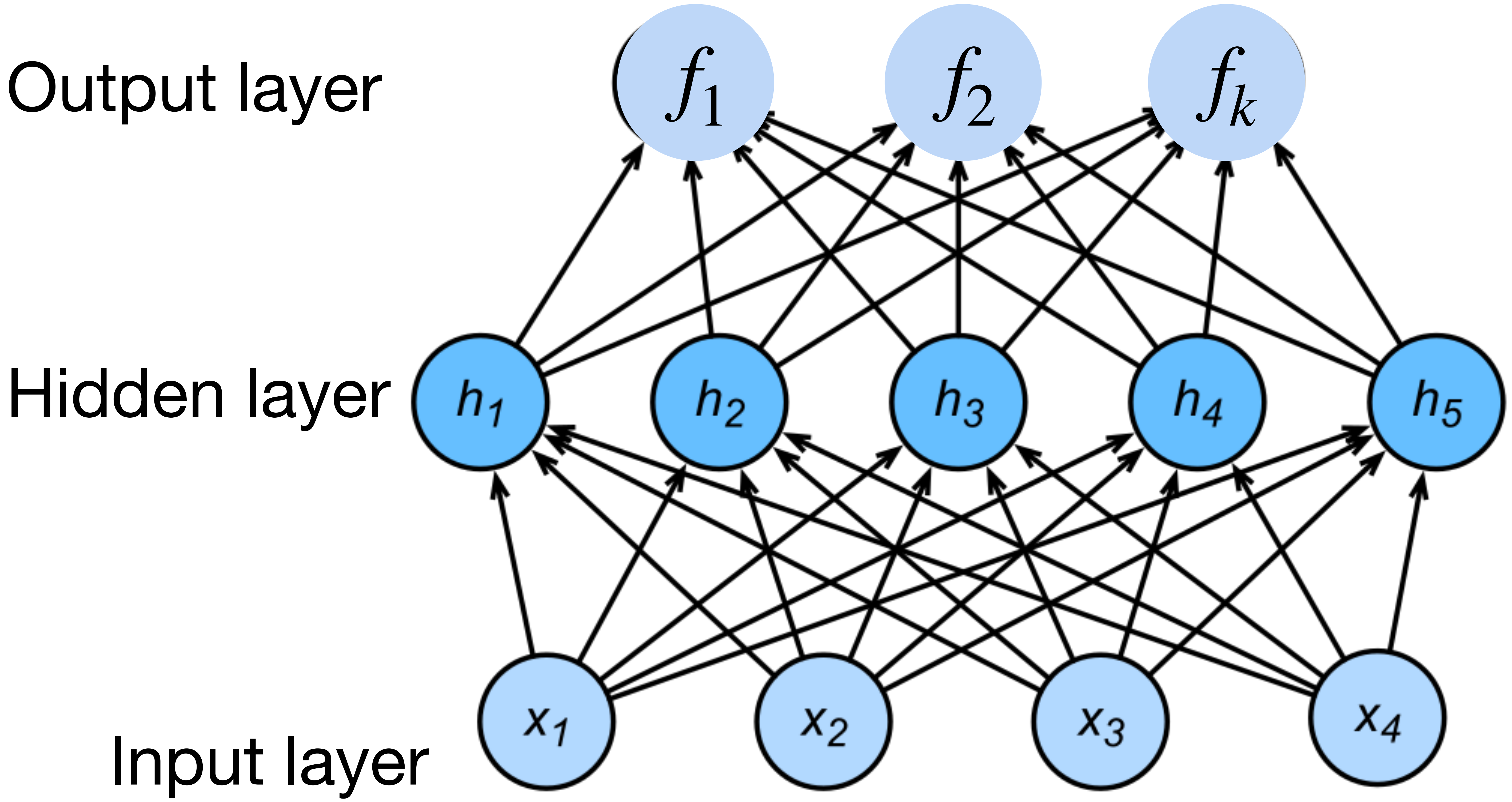
https://www.kaggle.com/c/human-protein-atlas-image-classification

More complicated neural networks



More complicated neural networks

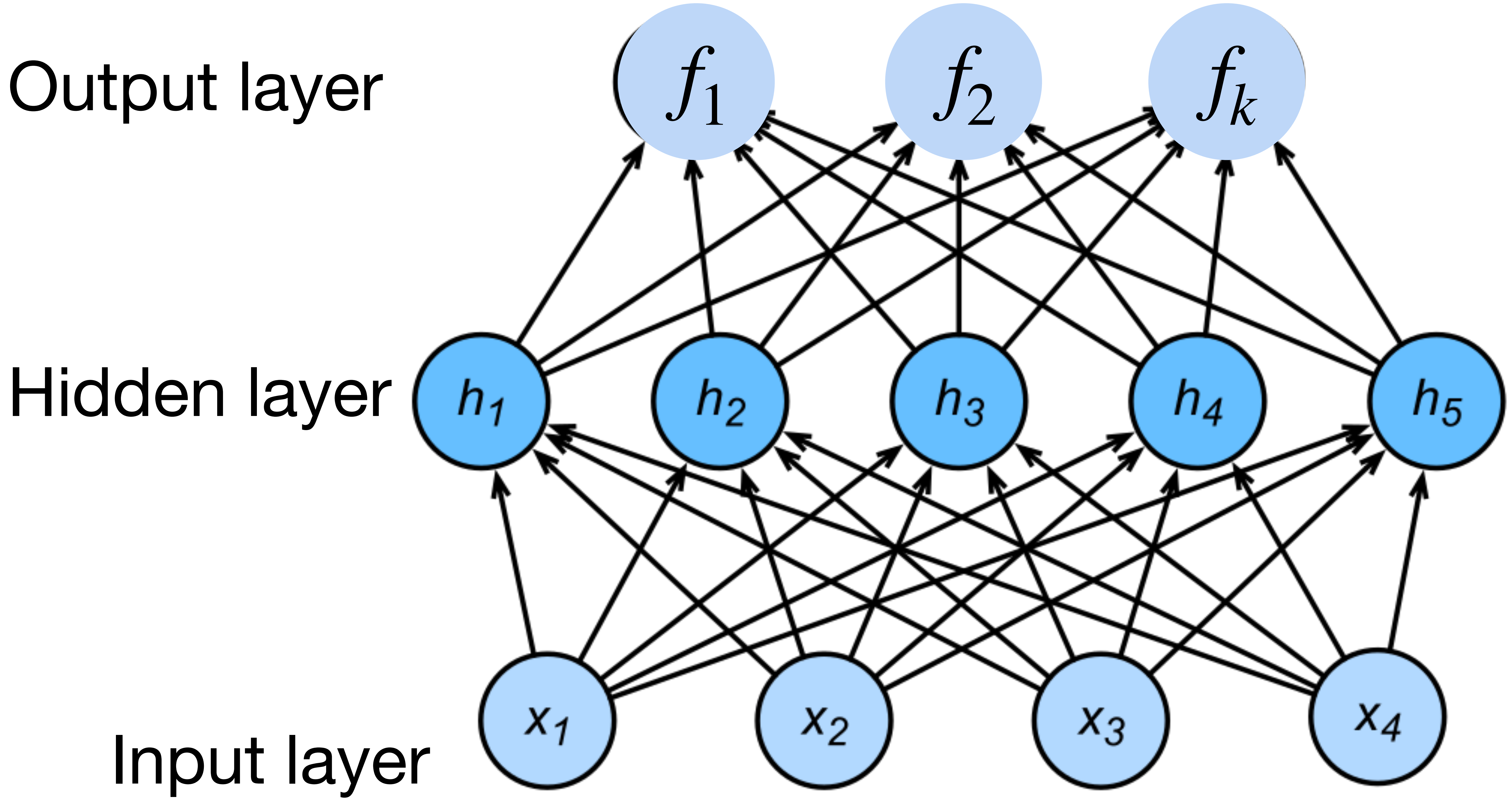
$$y_1, y_2, \ldots, y_k = \text{softmax}(f_1, f_2, \ldots, f_k)$$



More complicated neural networks

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

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b})$$
$$\mathbf{f} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$
$$\mathbf{y} = \text{softmax}(\mathbf{f})$$

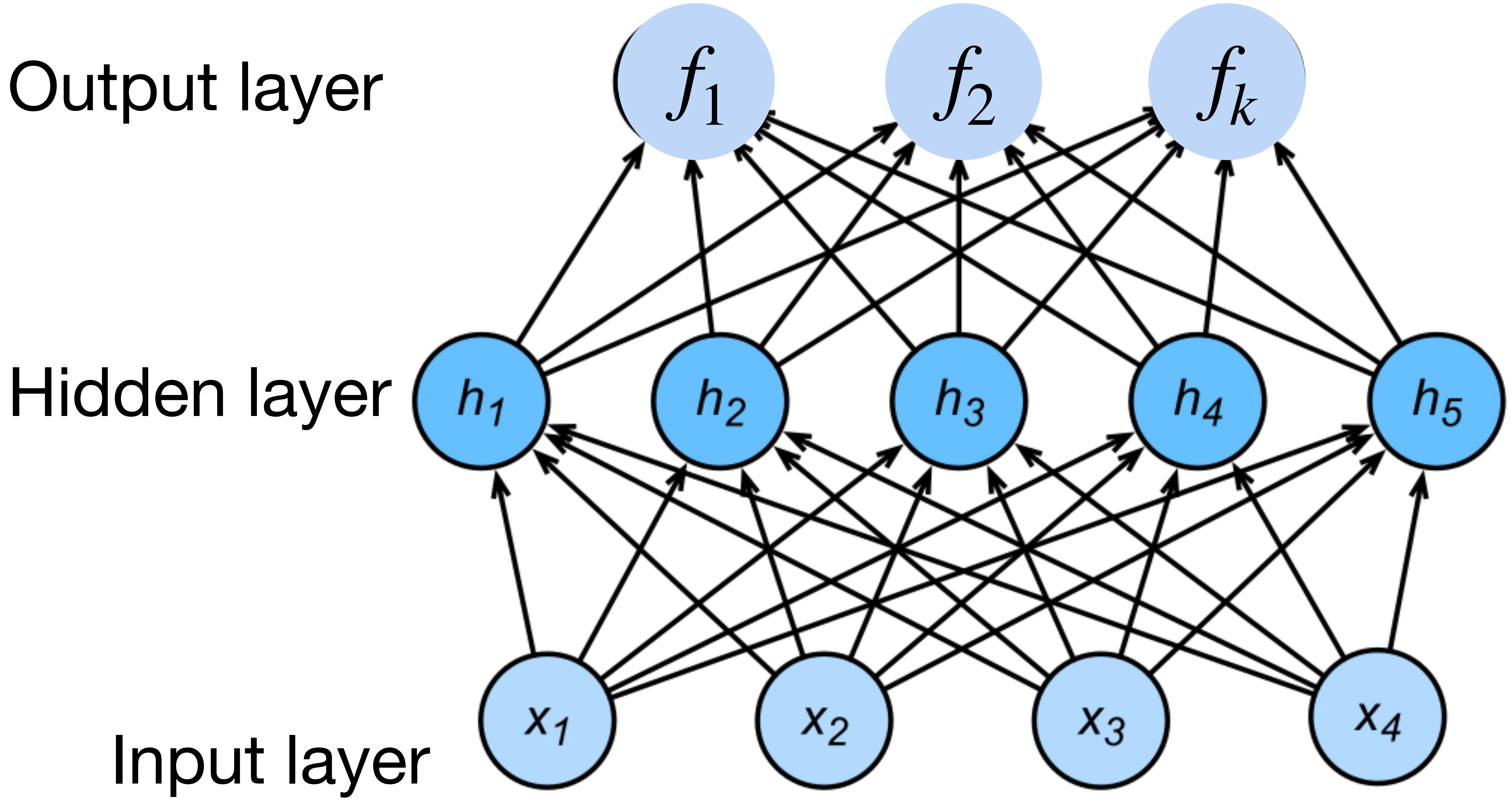


More complicated neural networks

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

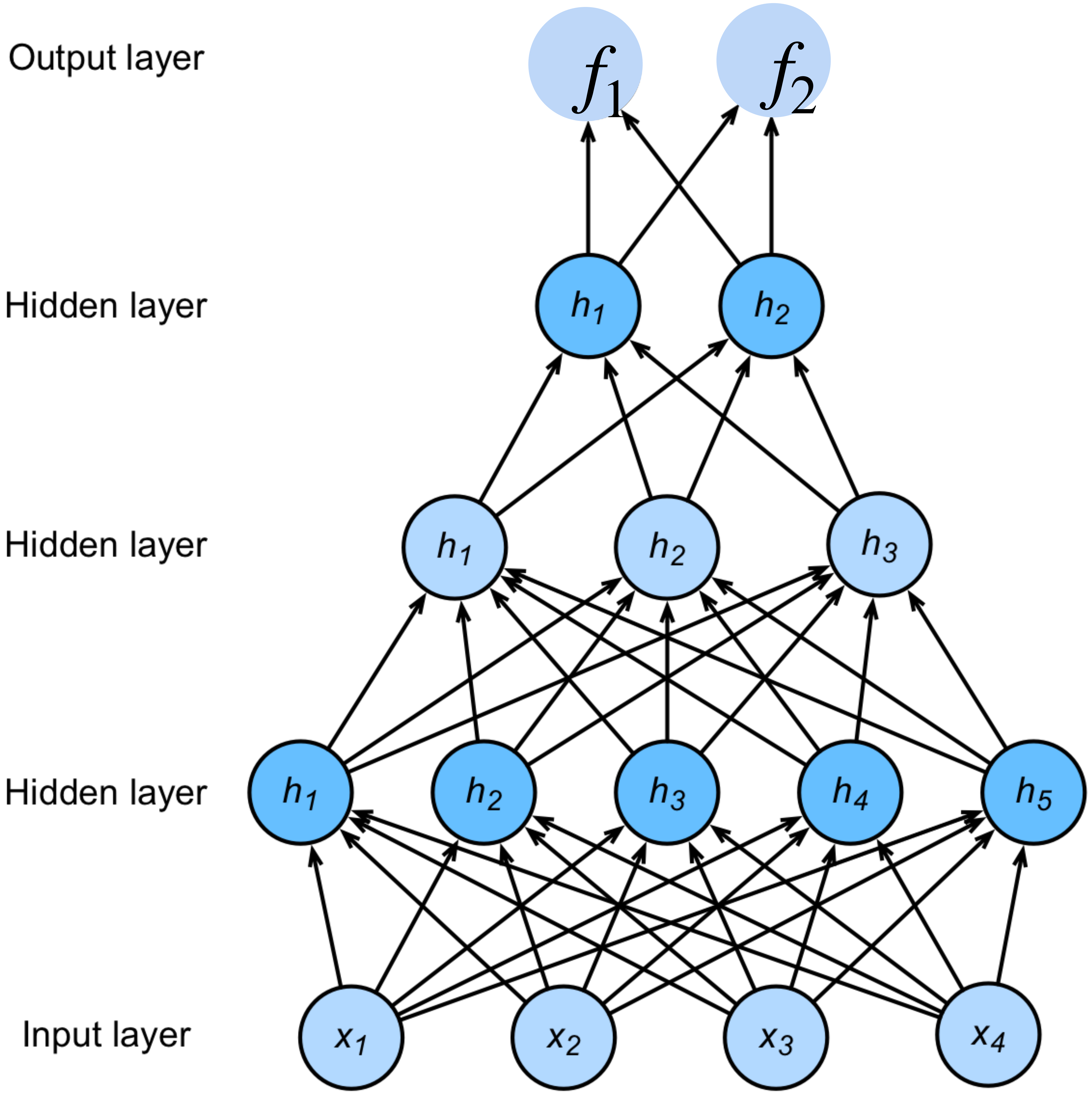
$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b})$$
$$\mathbf{f} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$
$$\mathbf{y} = \text{softmax}(\mathbf{f})$$

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



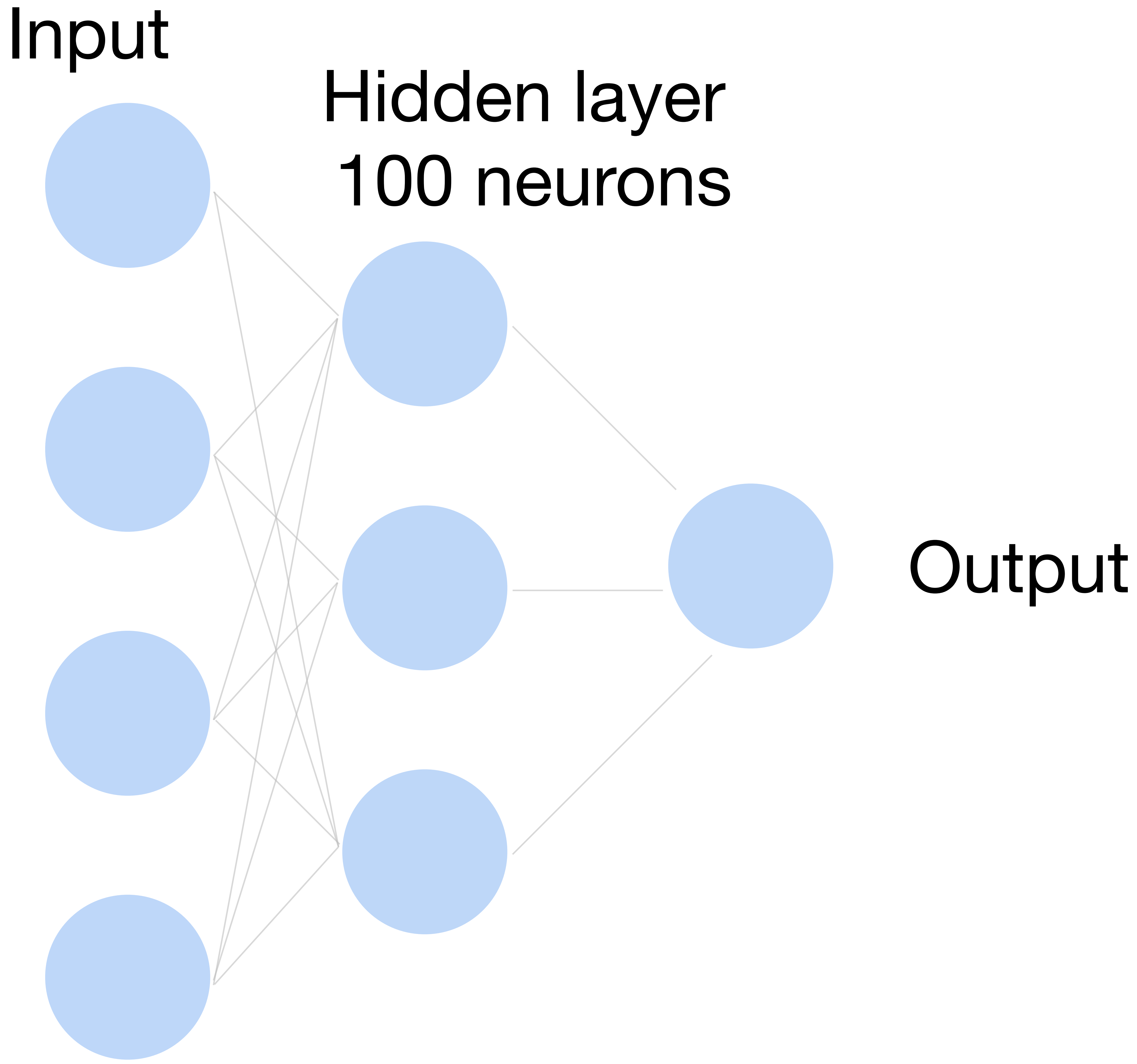
More complicated neural networks: multiple hidden layers

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$
$$\mathbf{h}_3 = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$$
$$\mathbf{f} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$
$$\mathbf{y} = \text{softmax}(\mathbf{f})$$



How to train a neural network?

Classify cats vs. dogs

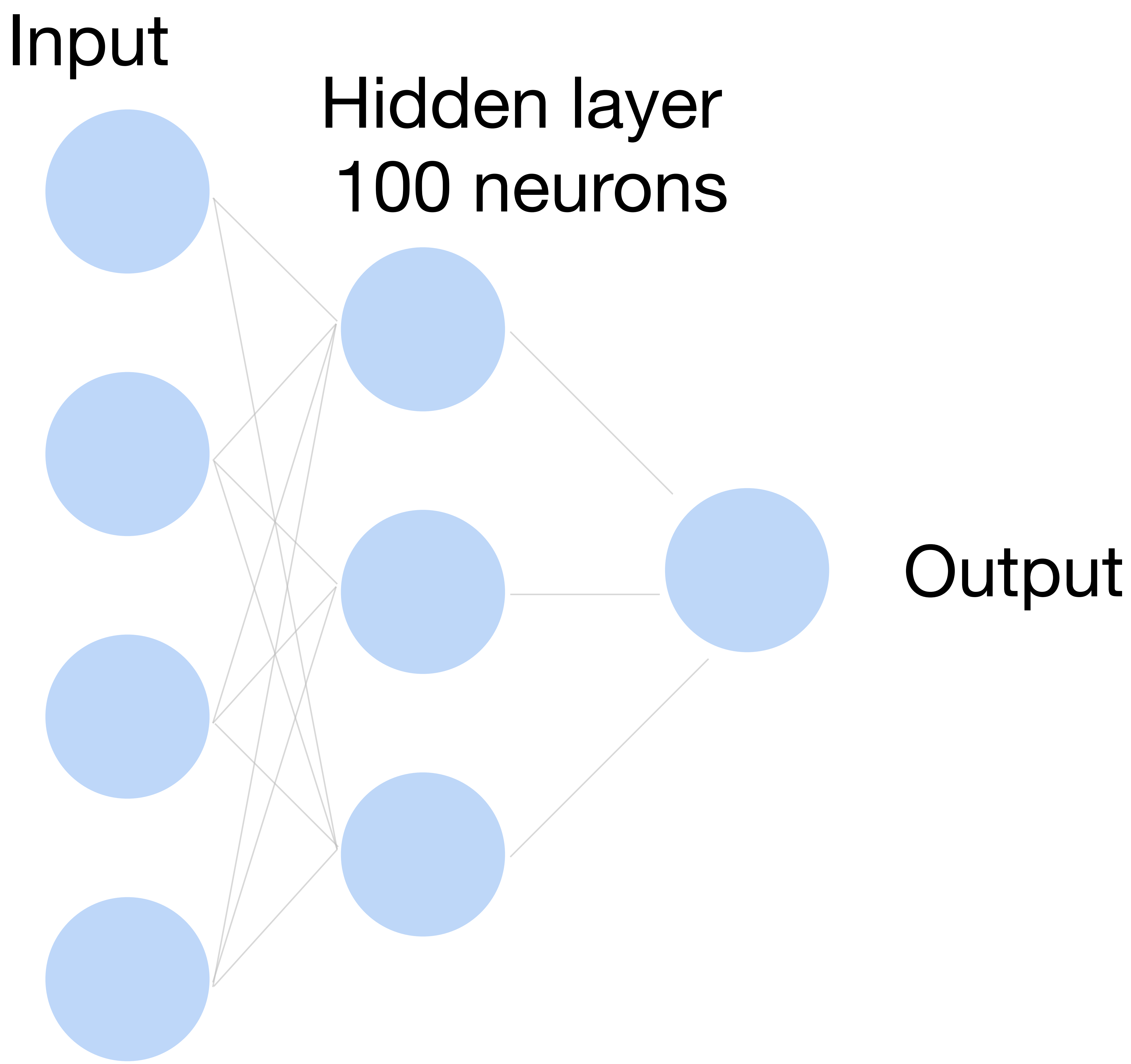


How to train a neural network?

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

$\hat{y}$ Model output for example **x**

y Ground truth label for example **x**

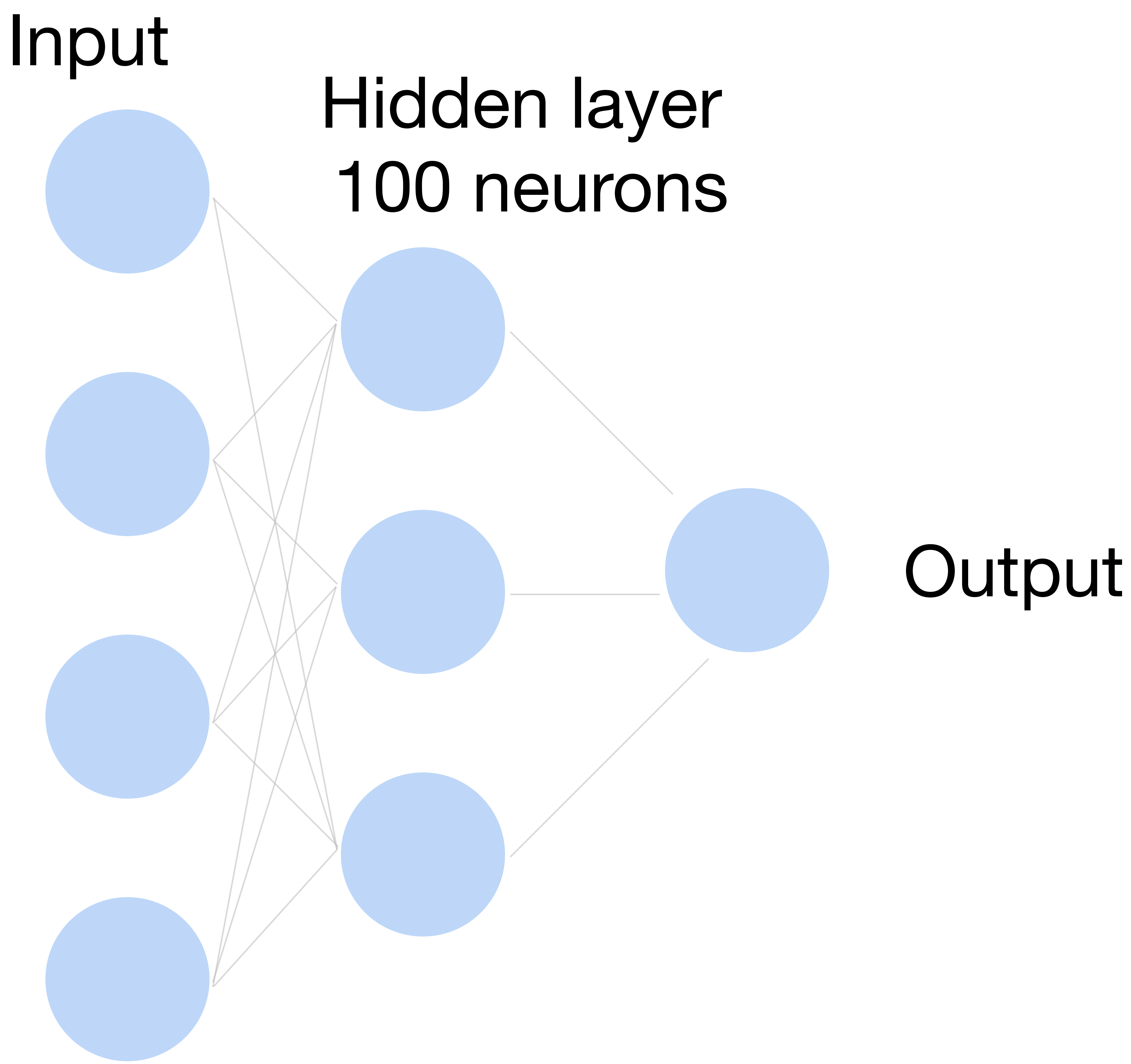


Learning by matching the output to the label

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

How to train a neural network?

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

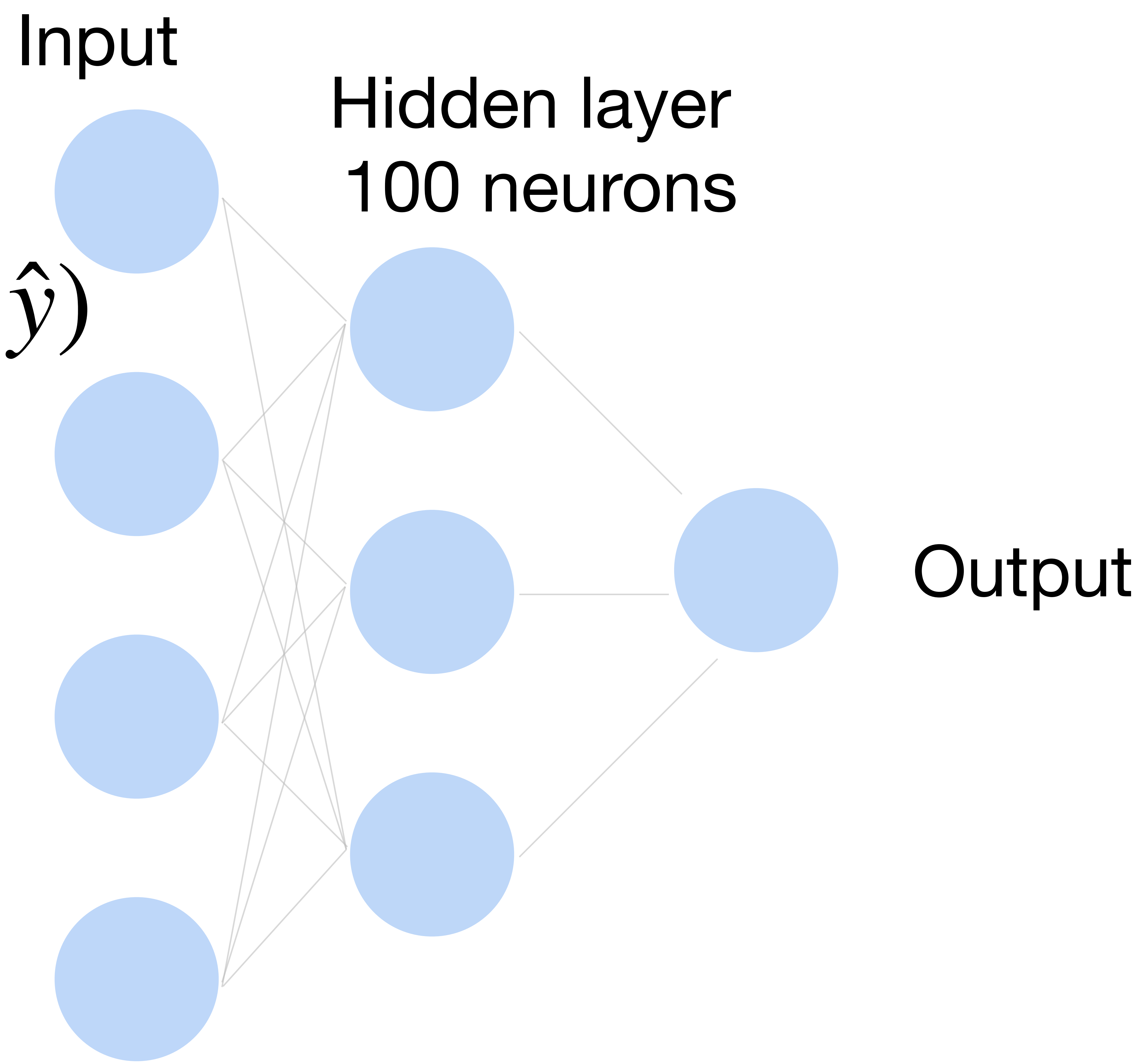


How to train a neural network?

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

Per-sample loss:

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



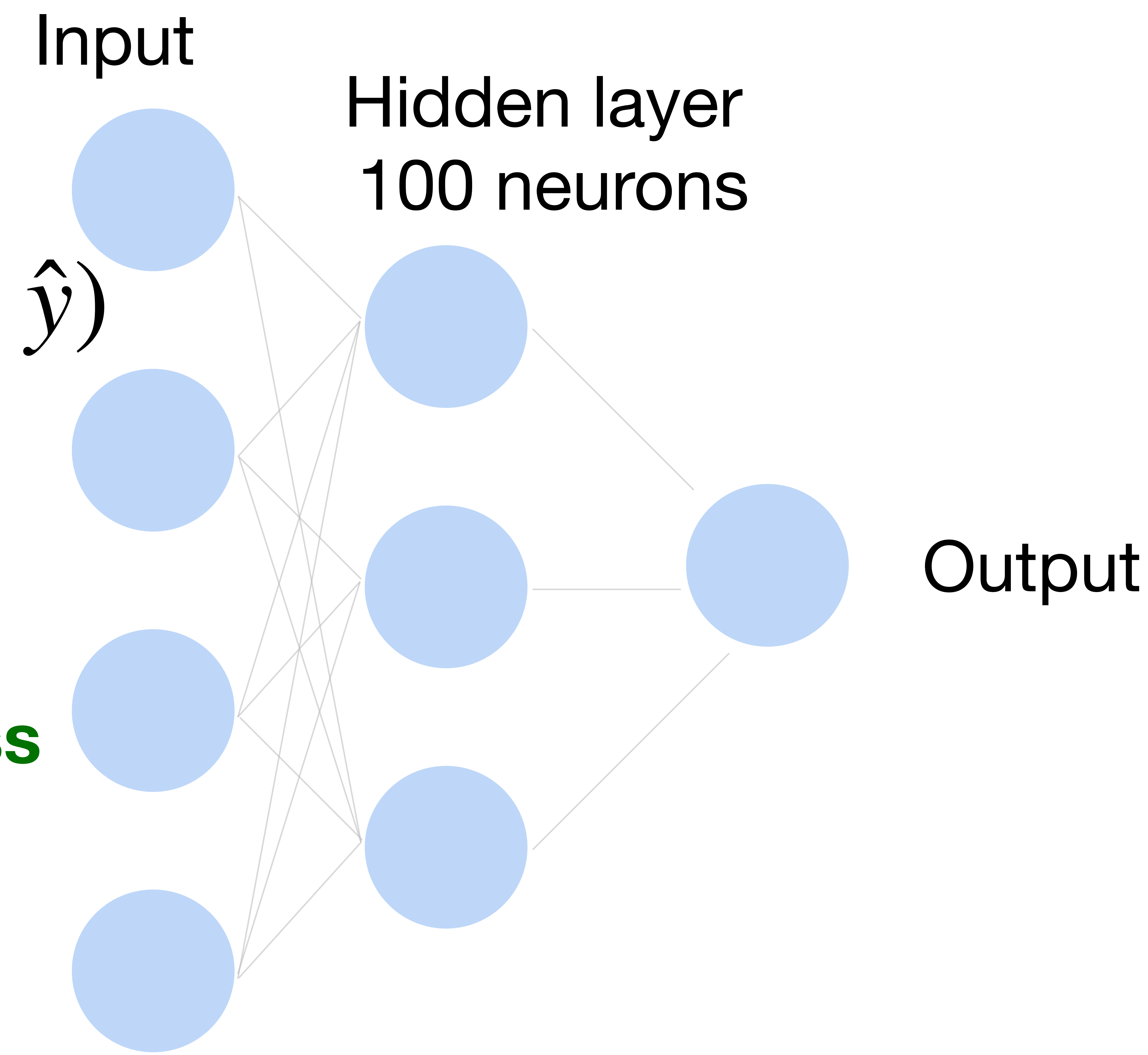
How to train a neural network?

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

Per-sample loss:

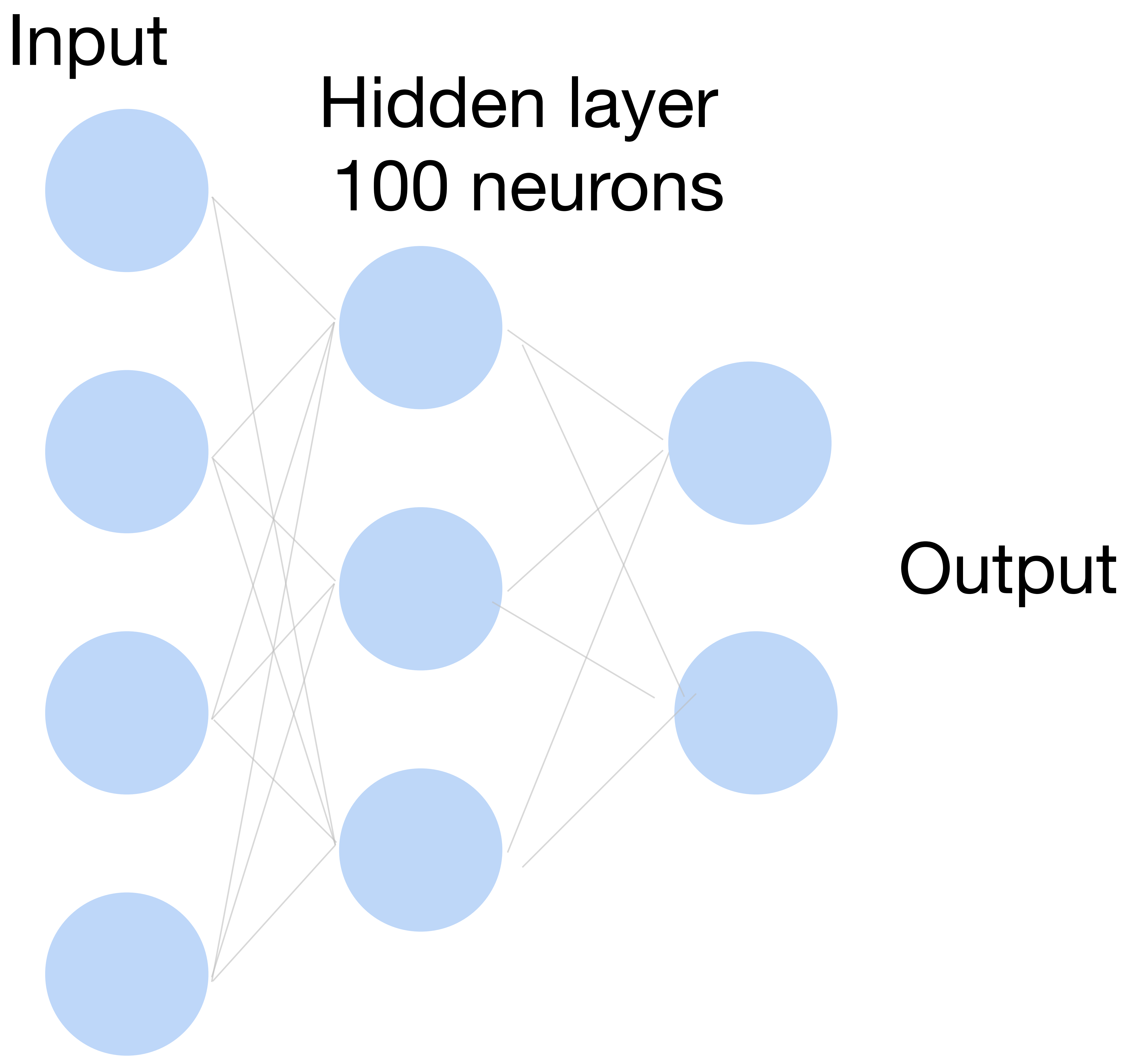
$$\ell(\mathbf{x}, y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

Also known as binary cross-entropy loss



How to train a neural network?

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

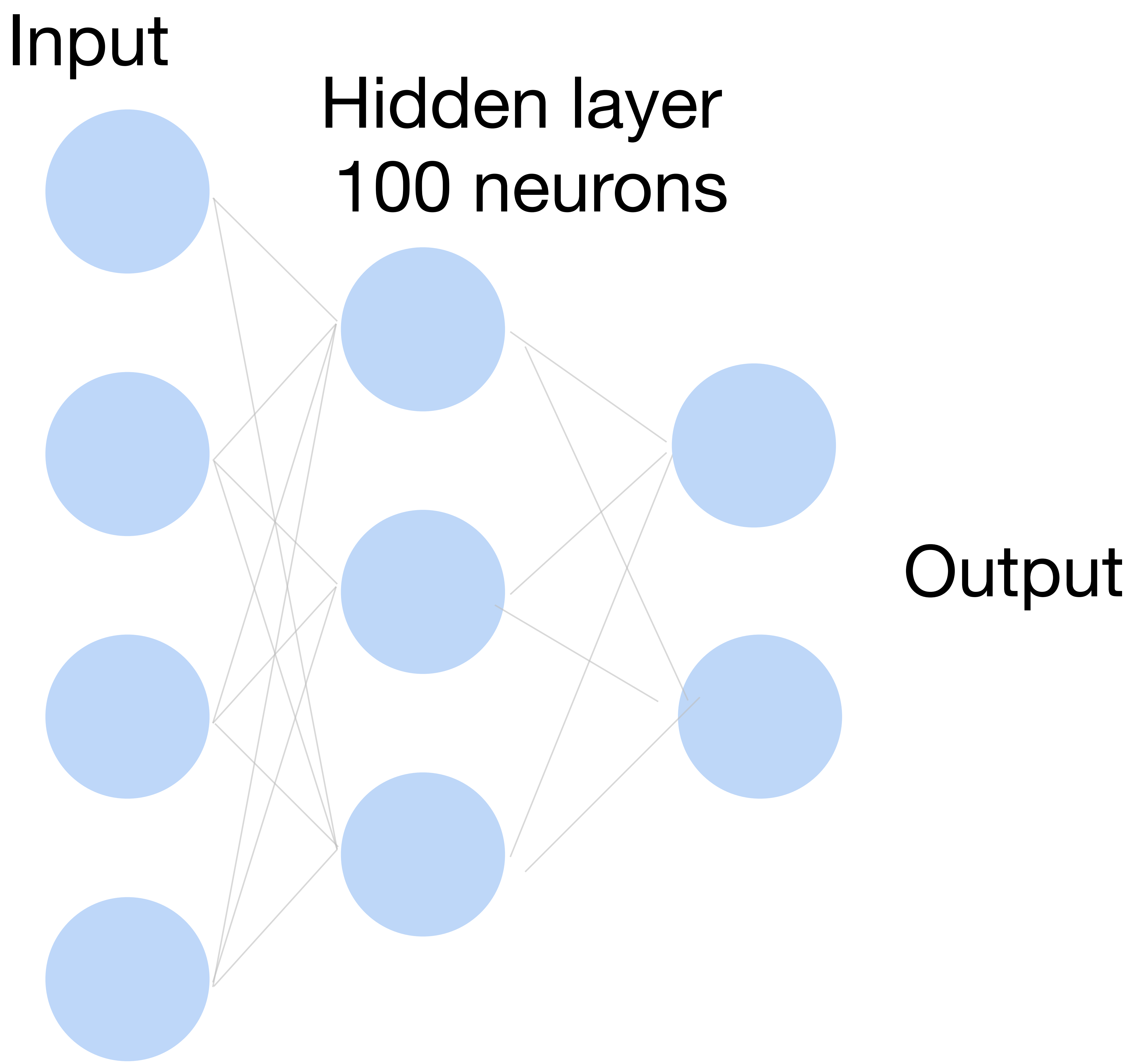


How to train a neural network?

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

Per-sample loss:

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



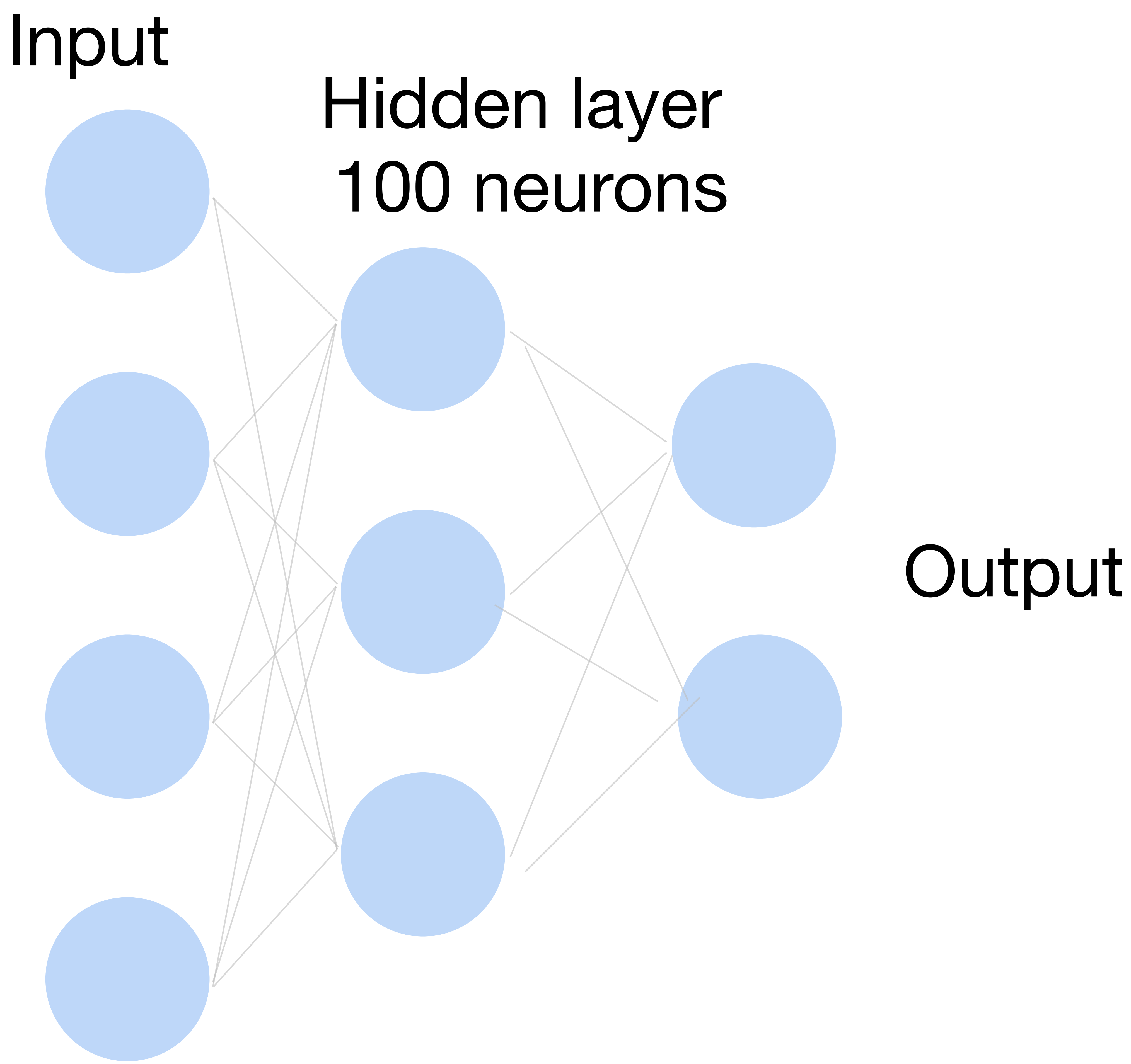
How to train a neural network?

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

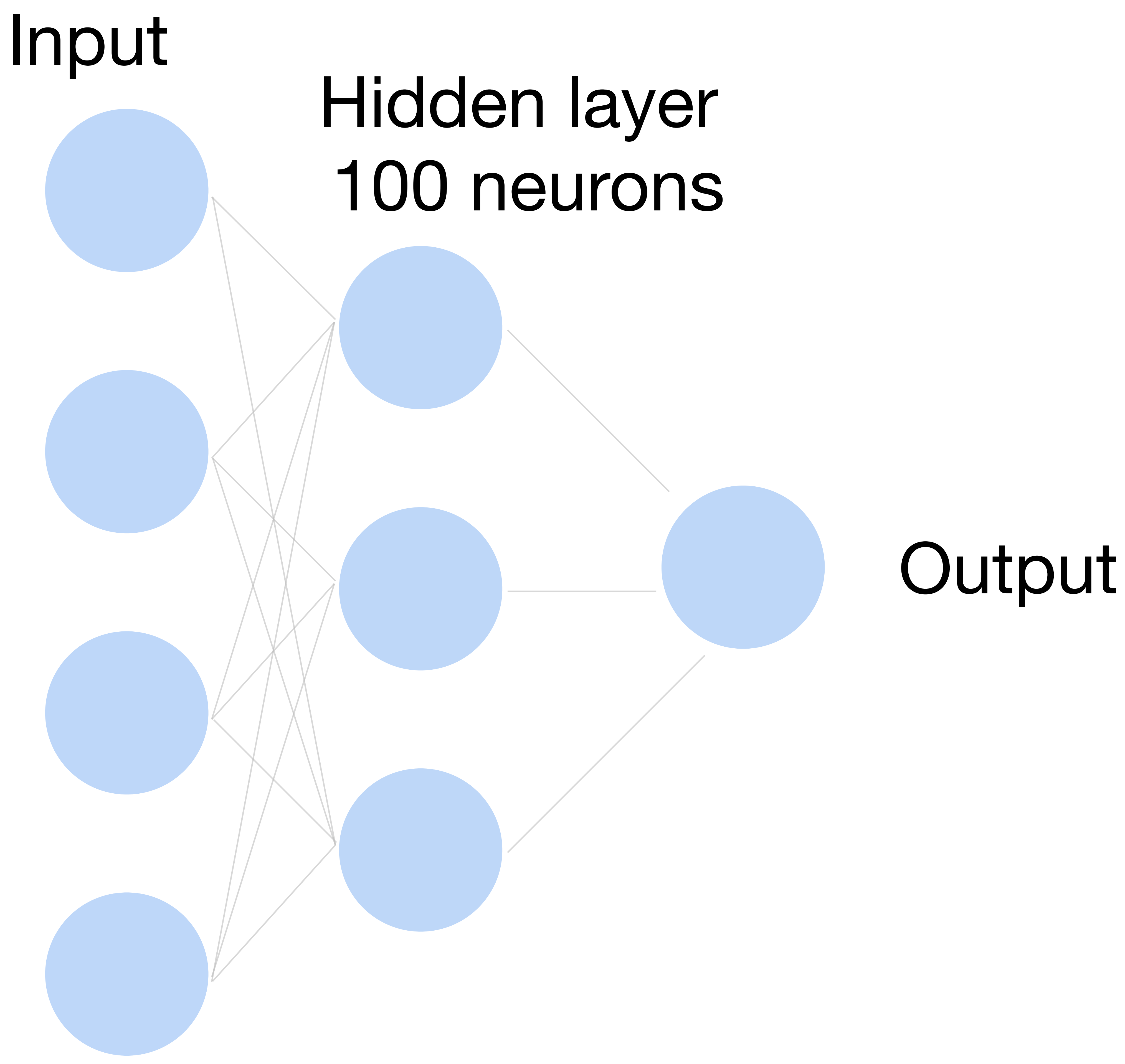


How to train a neural network?

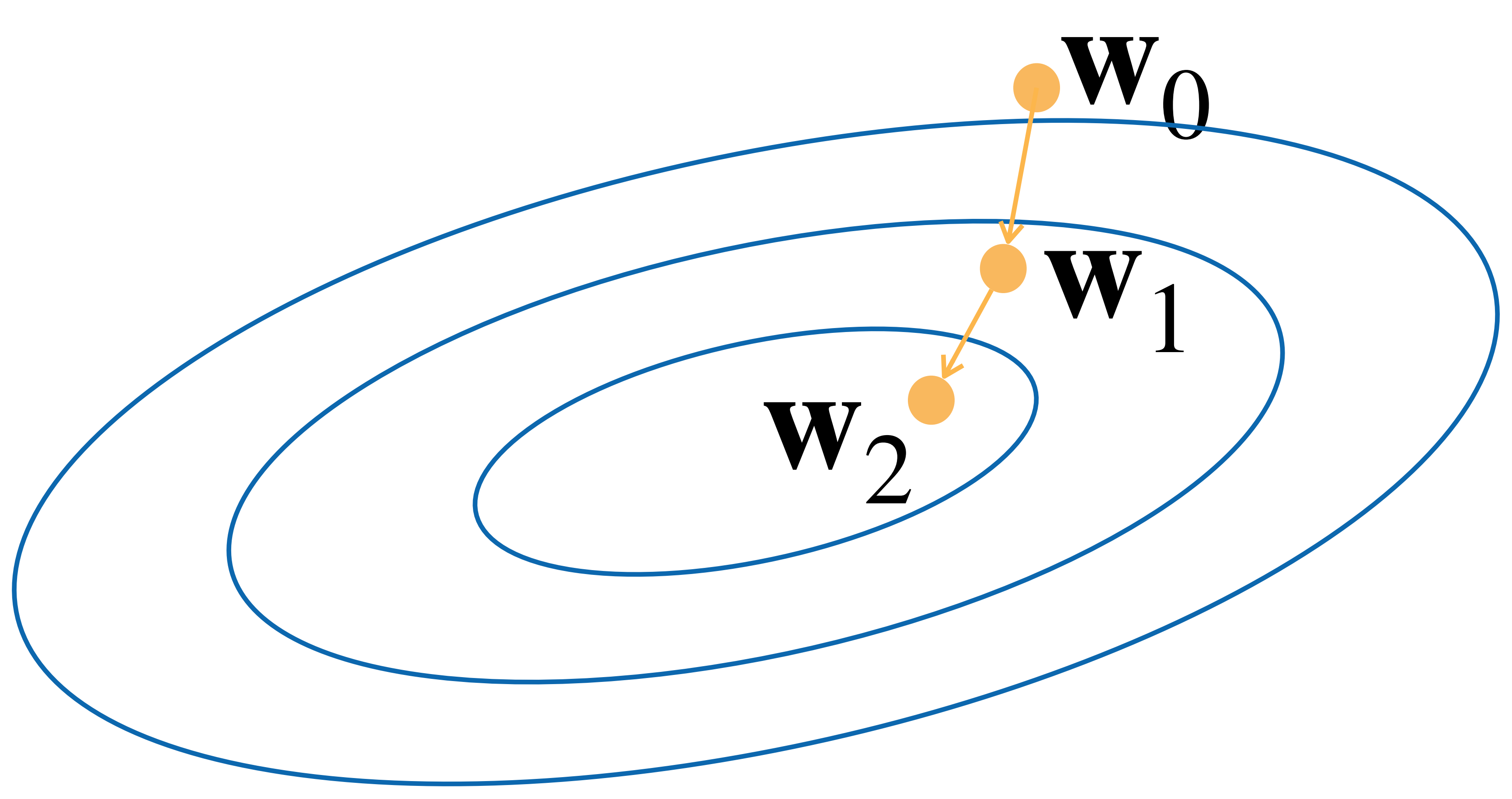
Update the weights W to minimize the loss function

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

Use gradient descent!



Gradient Descent



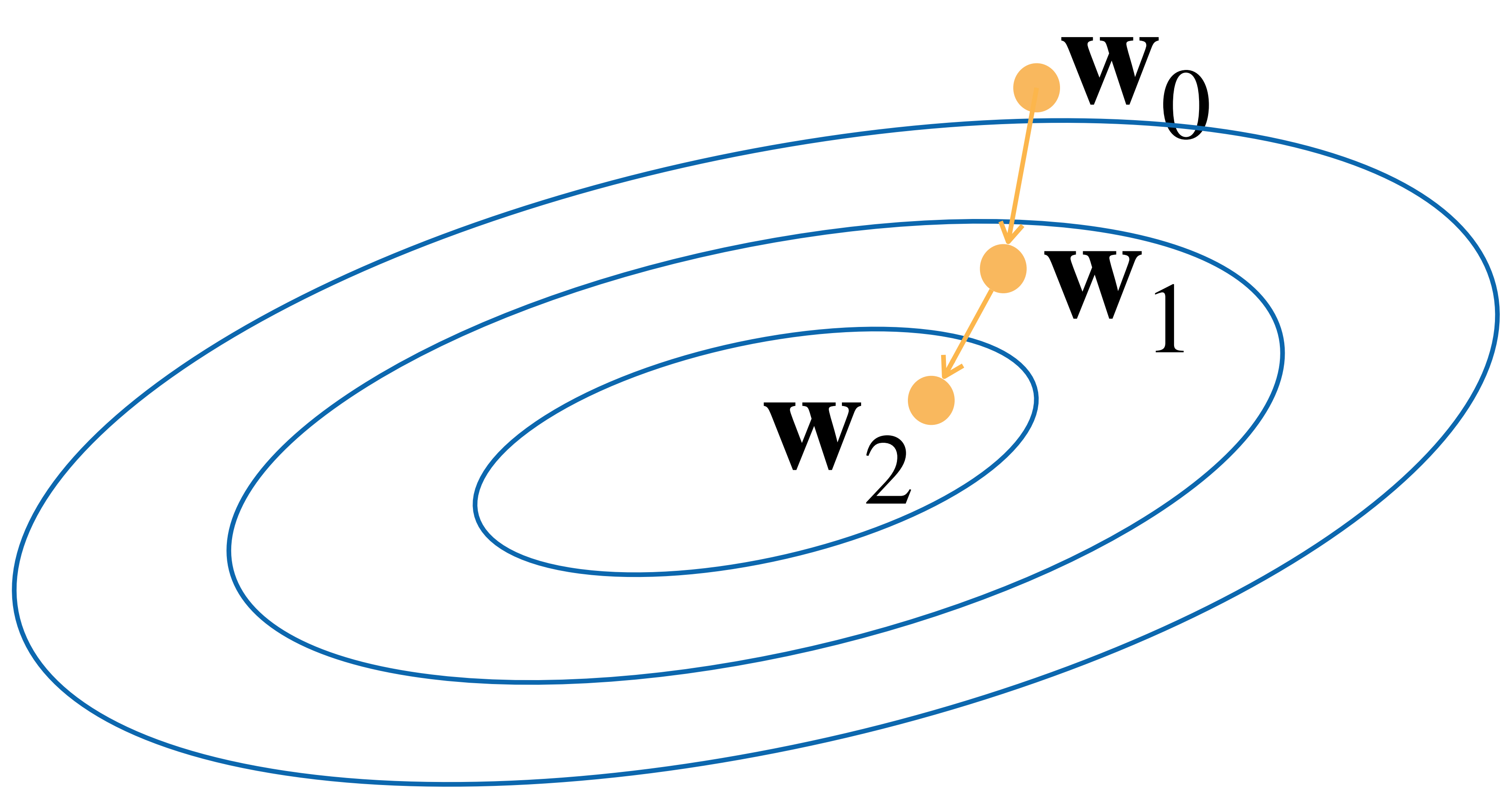
- Choose a learning rate $\alpha > 0$
- Initialize the model parameters w_0
- For t =1,2,…
    - Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \frac{\partial L}{\partial \mathbf{w}_{t-1}}$$

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

- Repeat until converges

Gradient Descent



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

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \frac{\partial L}{\partial \mathbf{w}_{t-1}}$$

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

D can
be very large.
Expensive

- Repeat until converges

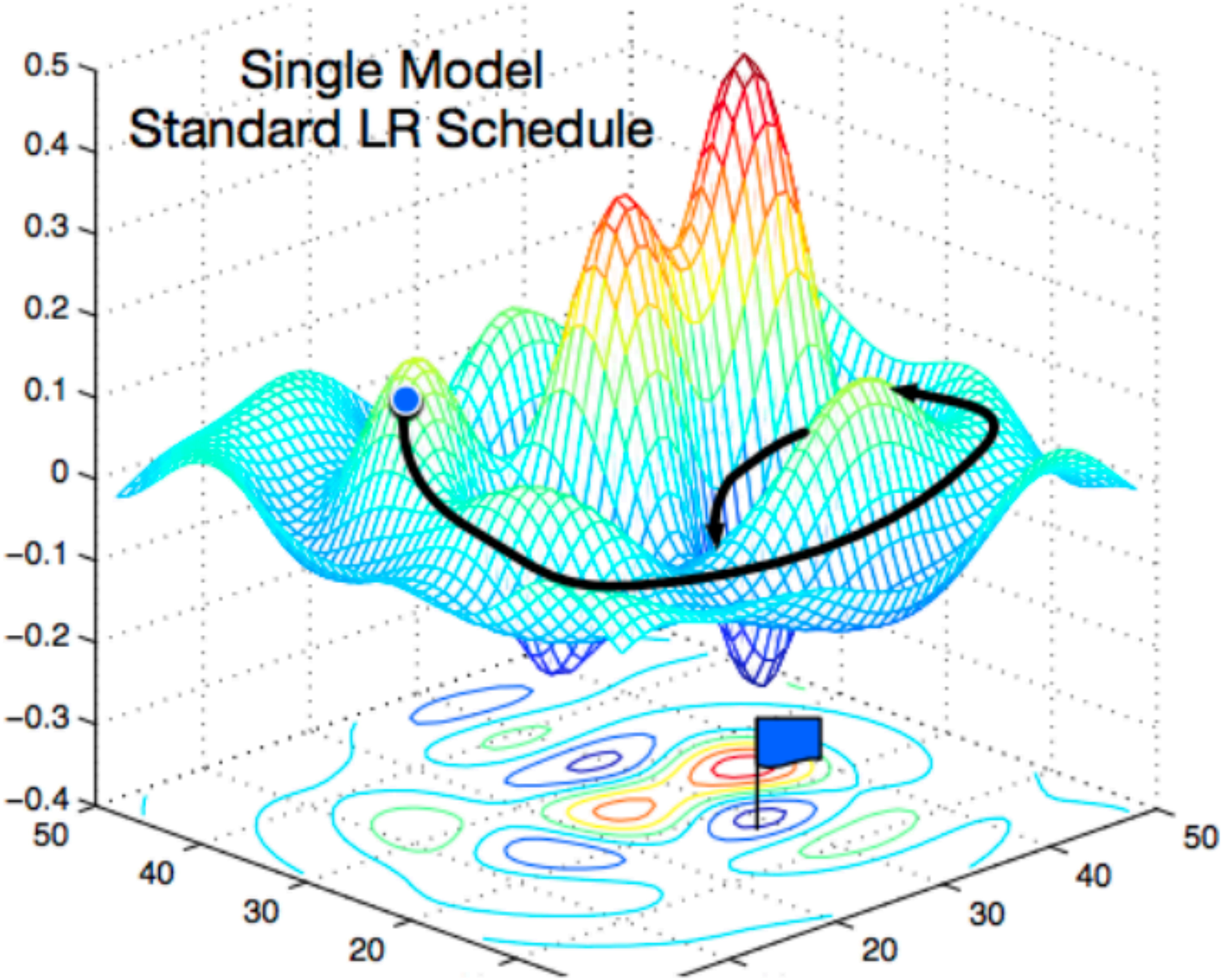
Minibatch Stochastic Gradient Descent

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

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

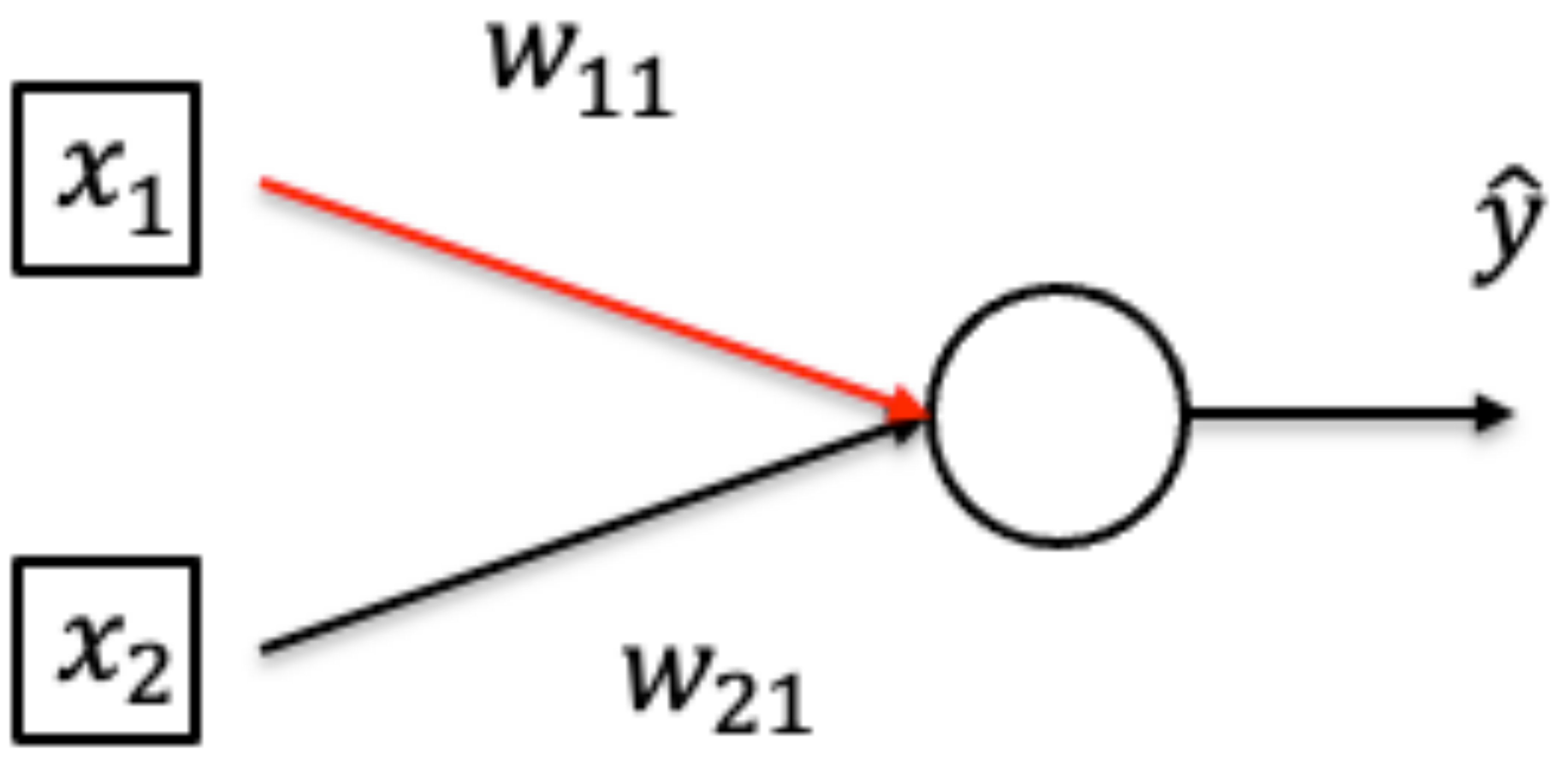
- Repeat until converges

Non-convex Optimization



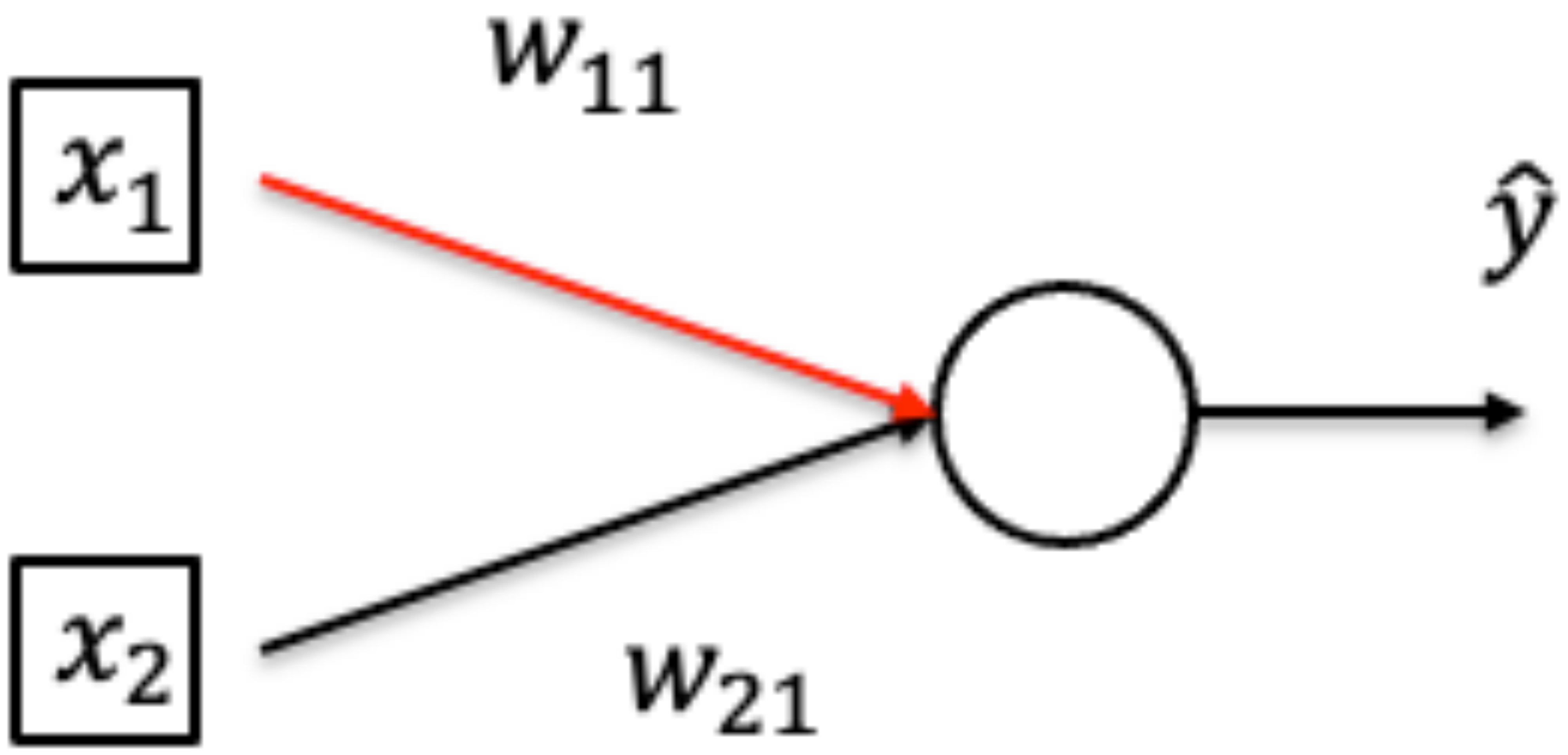
[Gao and Li et al., 2018]

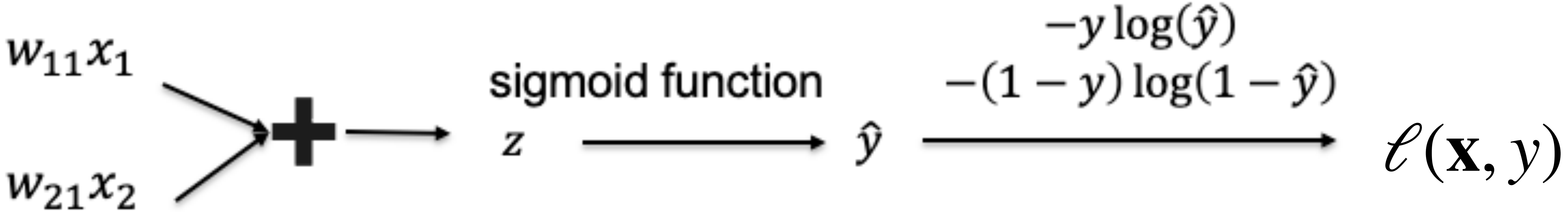
Calculate Gradient (on one data point)



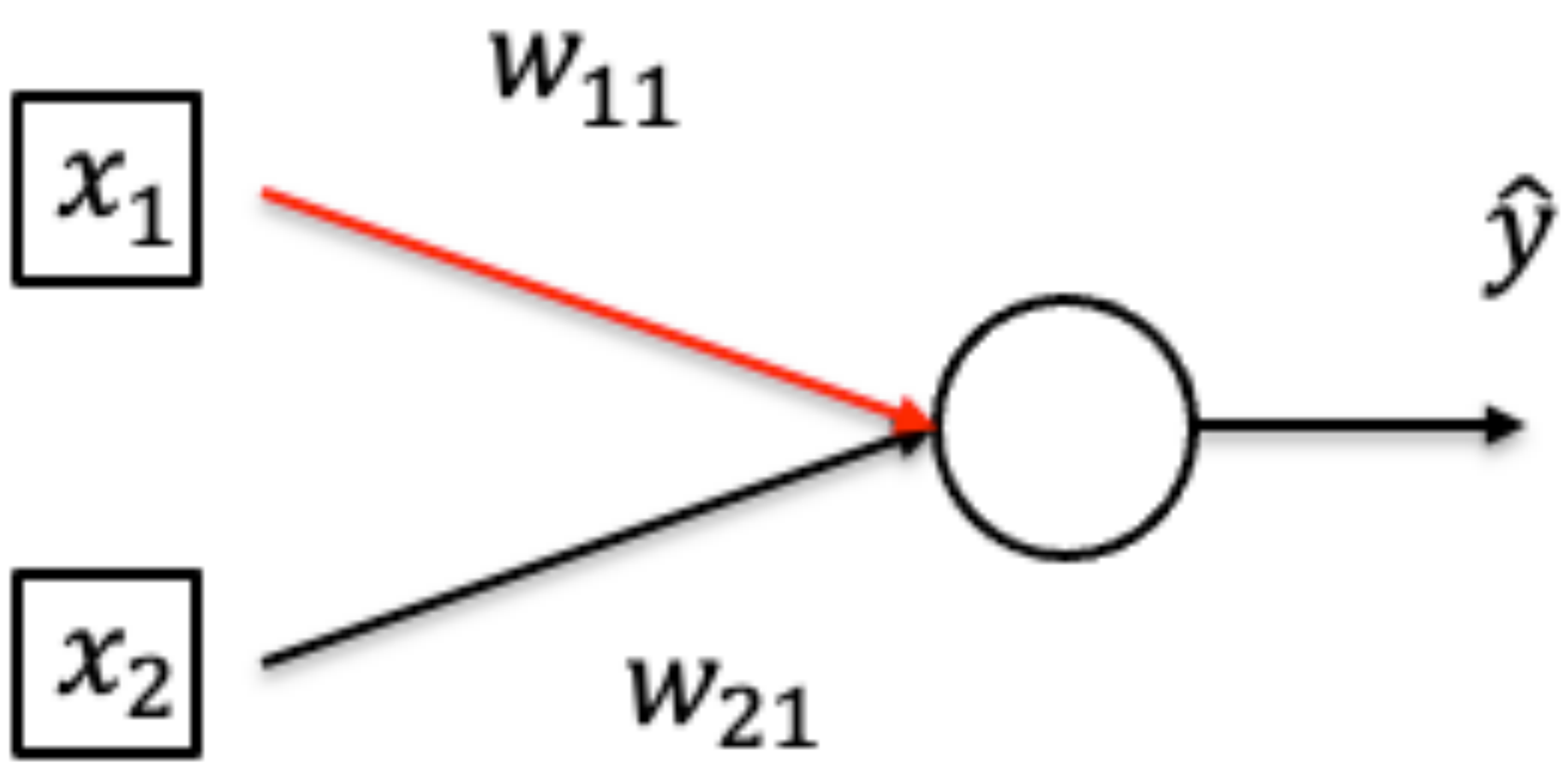
- Want to compute $\dfrac{\partial \ell(\mathbf{x}, y)}{\partial w_{11}}$

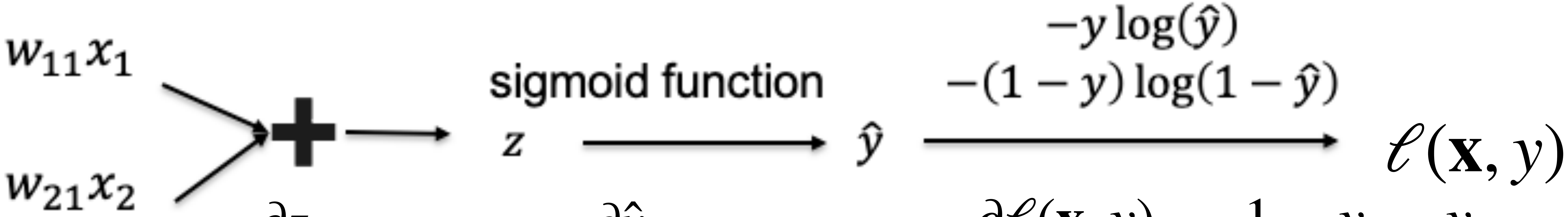
Calculate Gradient (on one data point)





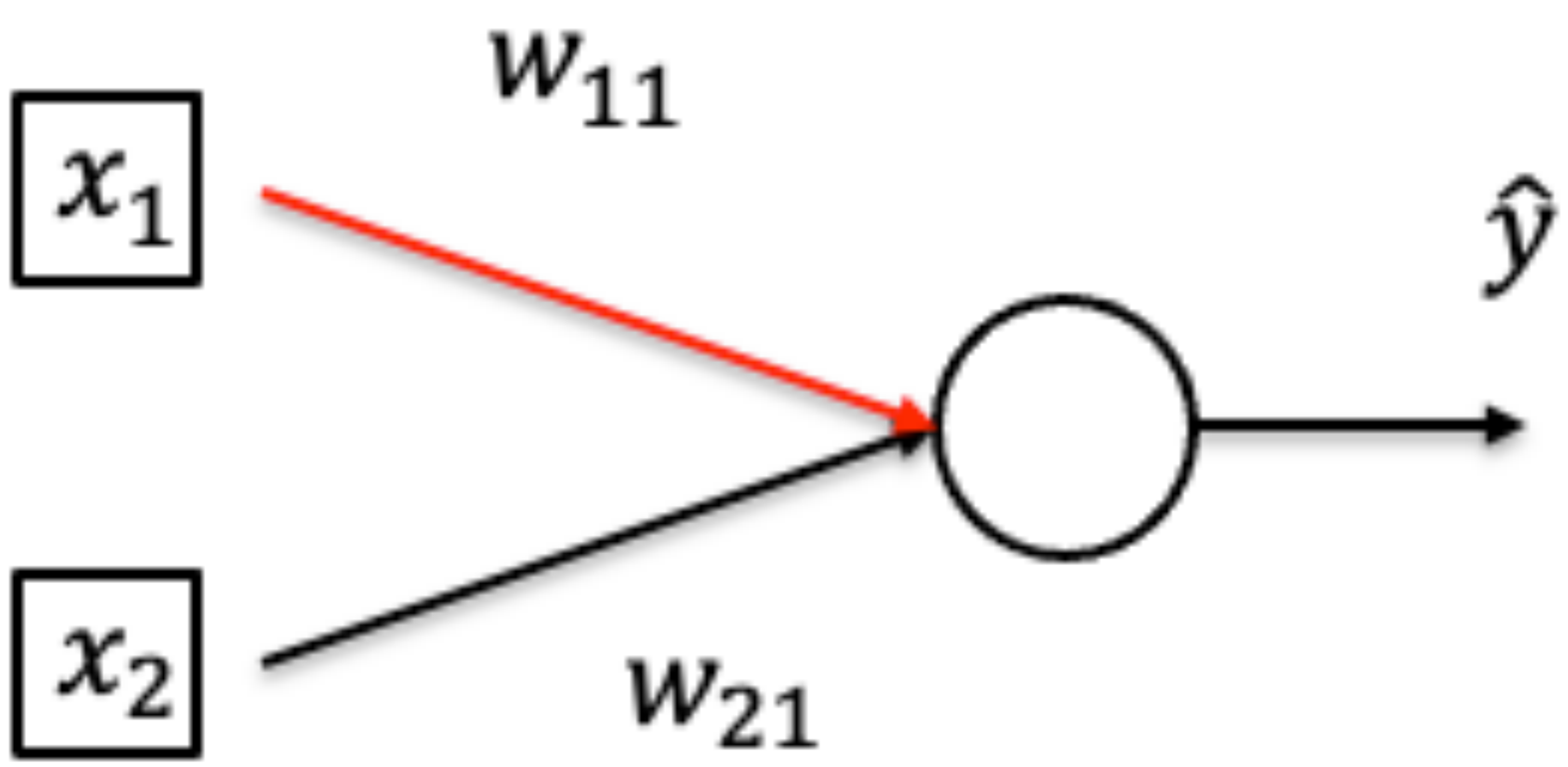
Calculate Gradient (on one data point)

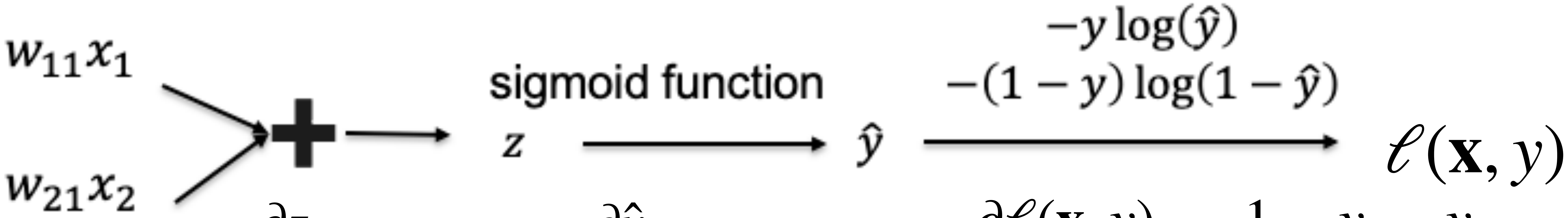




$$\frac{\partial z}{\partial w_{11}} = x_1 \qquad \frac{\partial \hat{y}}{\partial z} = \sigma'(z) \qquad \frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

Calculate Gradient (on one data point)

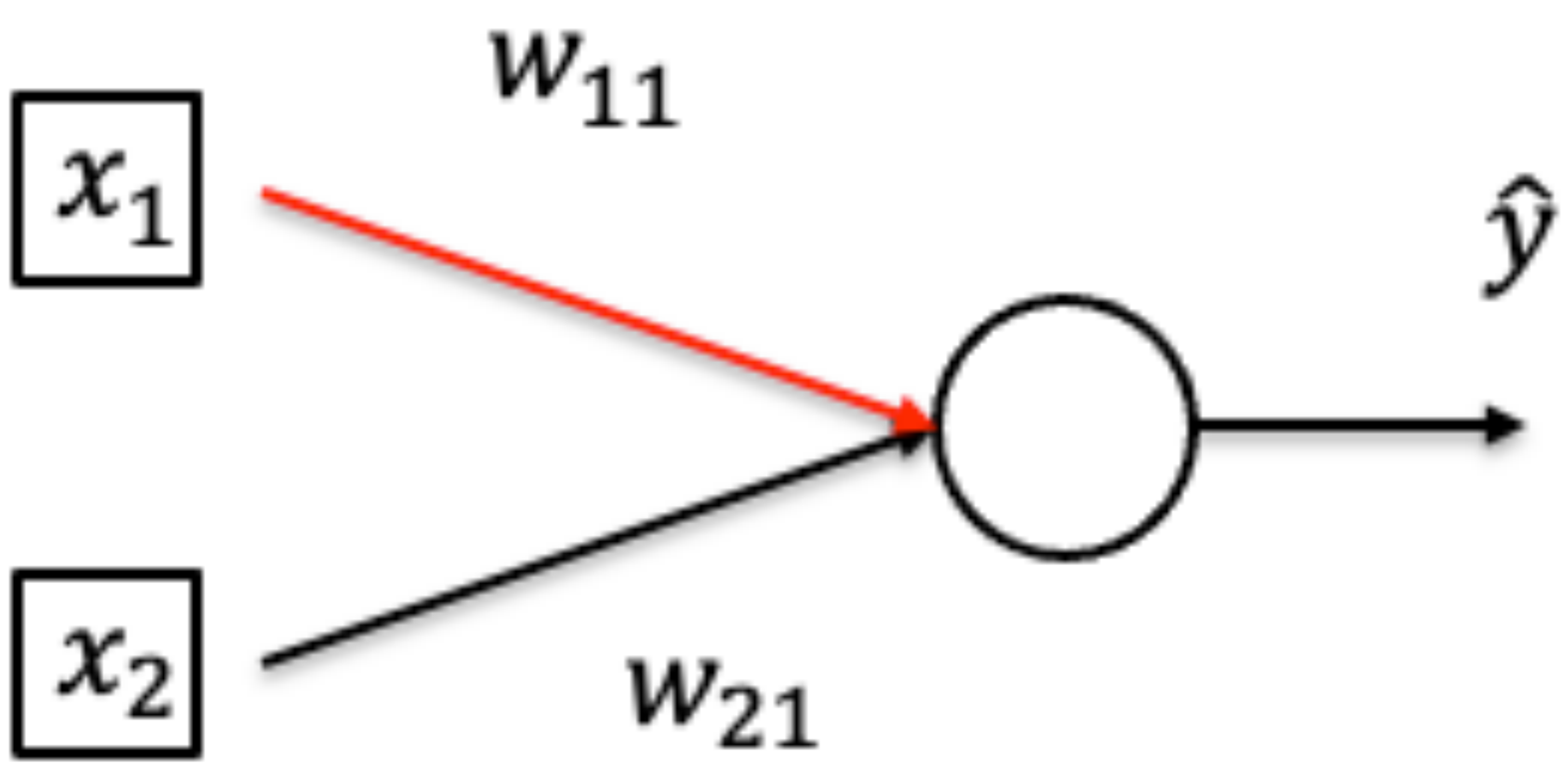


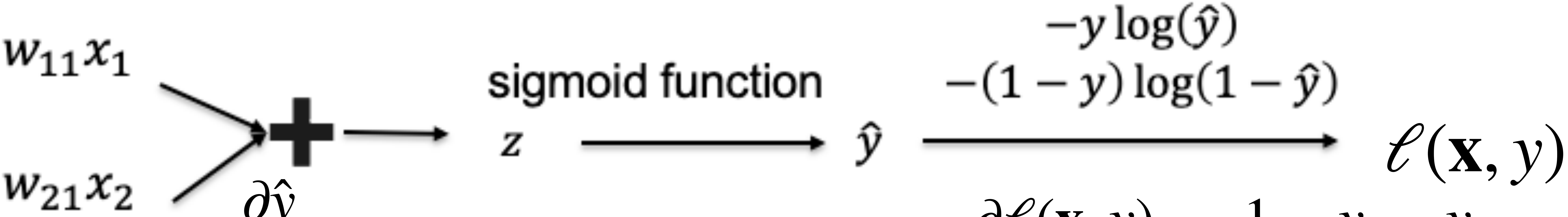


$$\frac{\partial z}{\partial w_{11}} = x_1 \qquad \frac{\partial \hat{y}}{\partial z} = \sigma'(z) \qquad \frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

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

Calculate Gradient (on one data point)

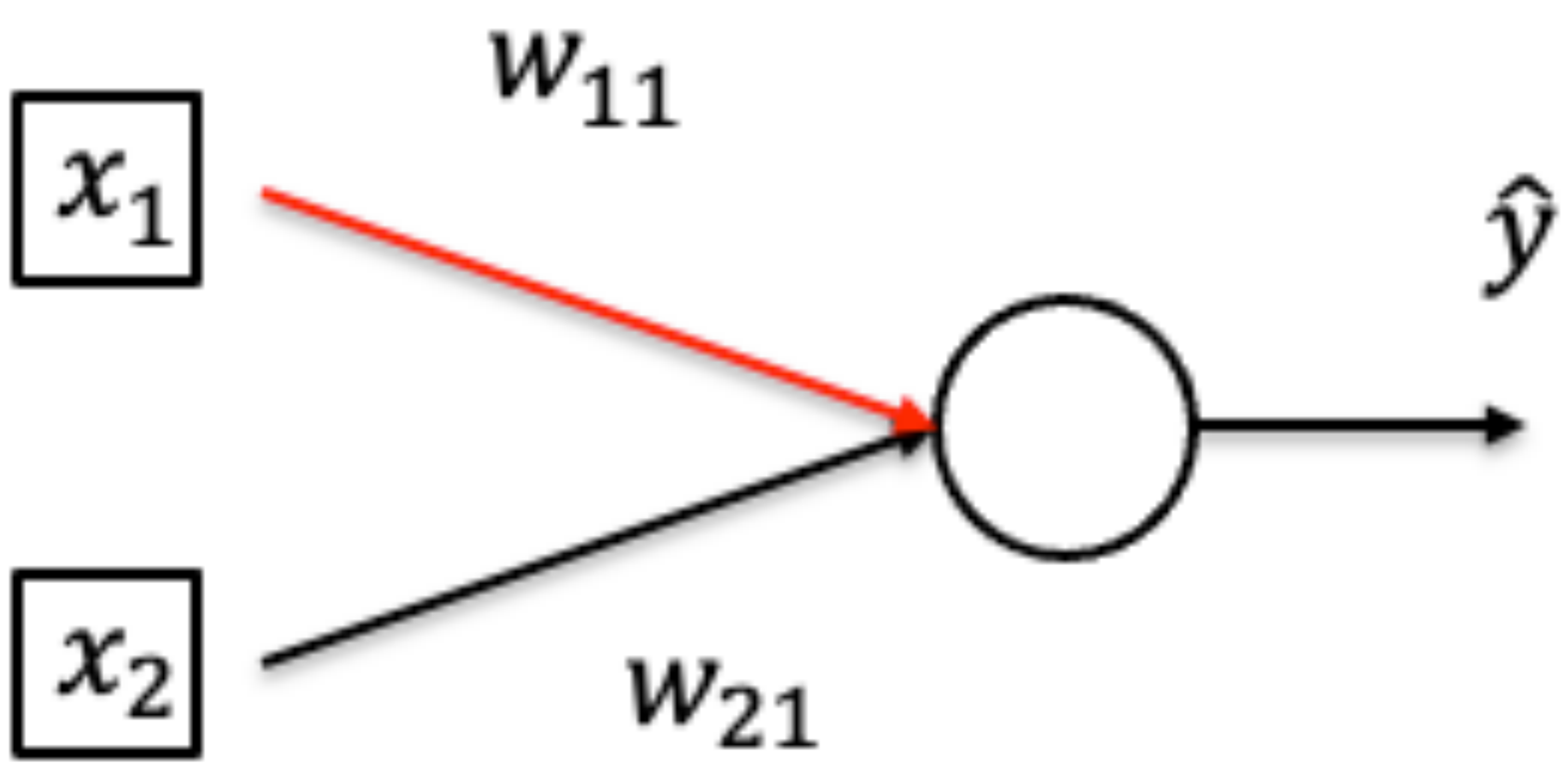


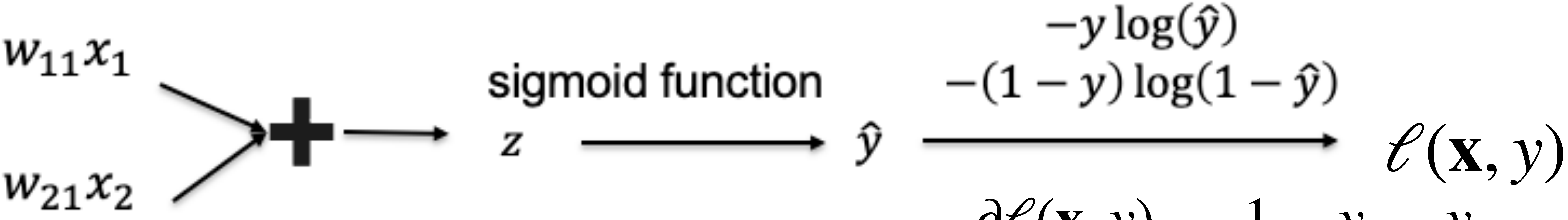


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z)) \qquad \frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

- 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)

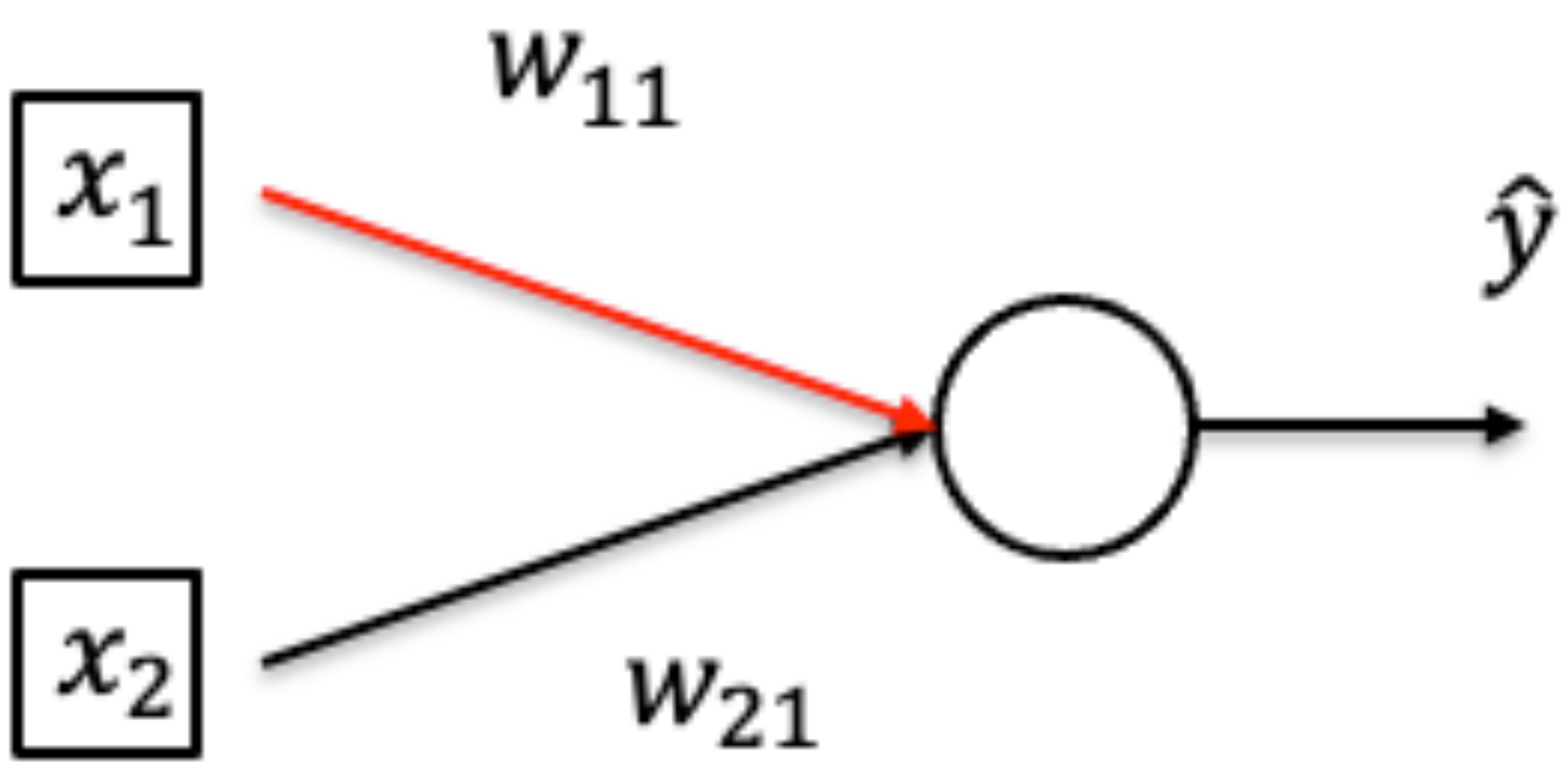


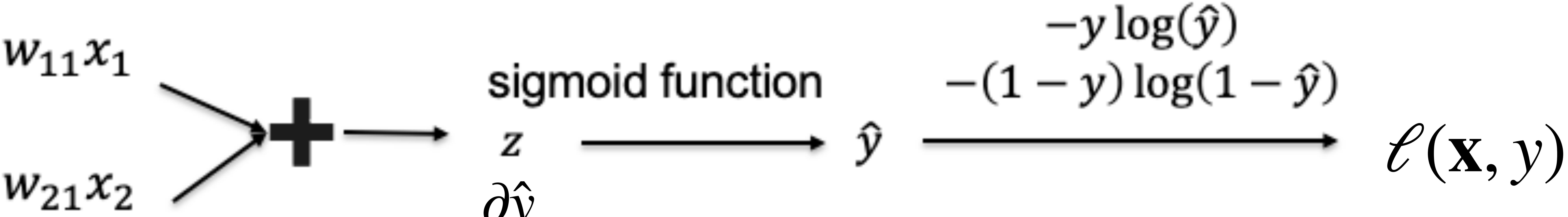


$$\frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

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

Calculate Gradient (on one data point)

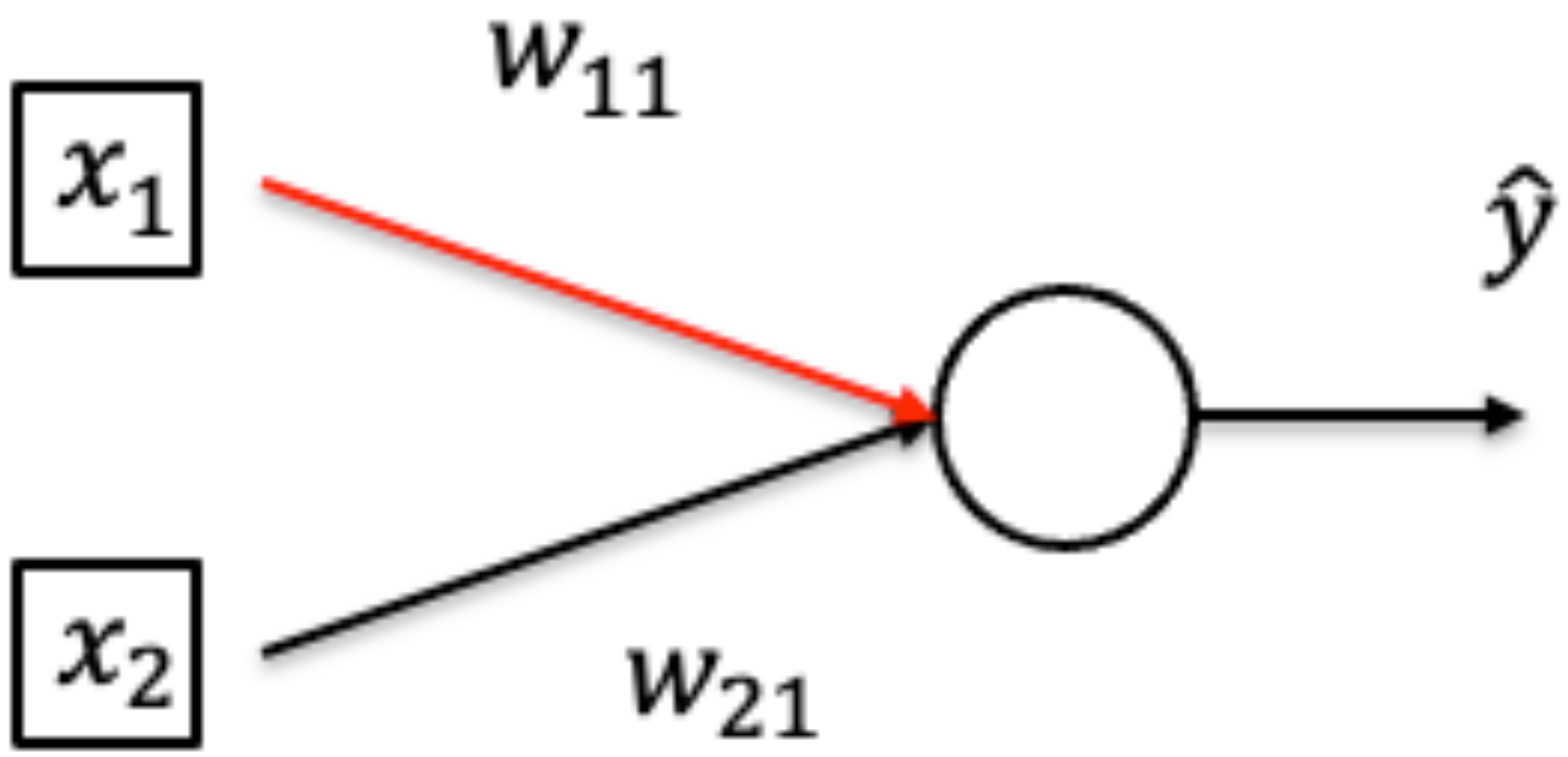


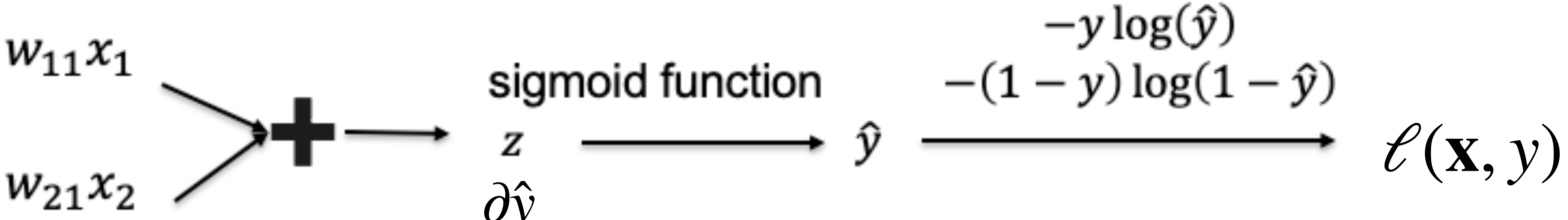


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

- 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)

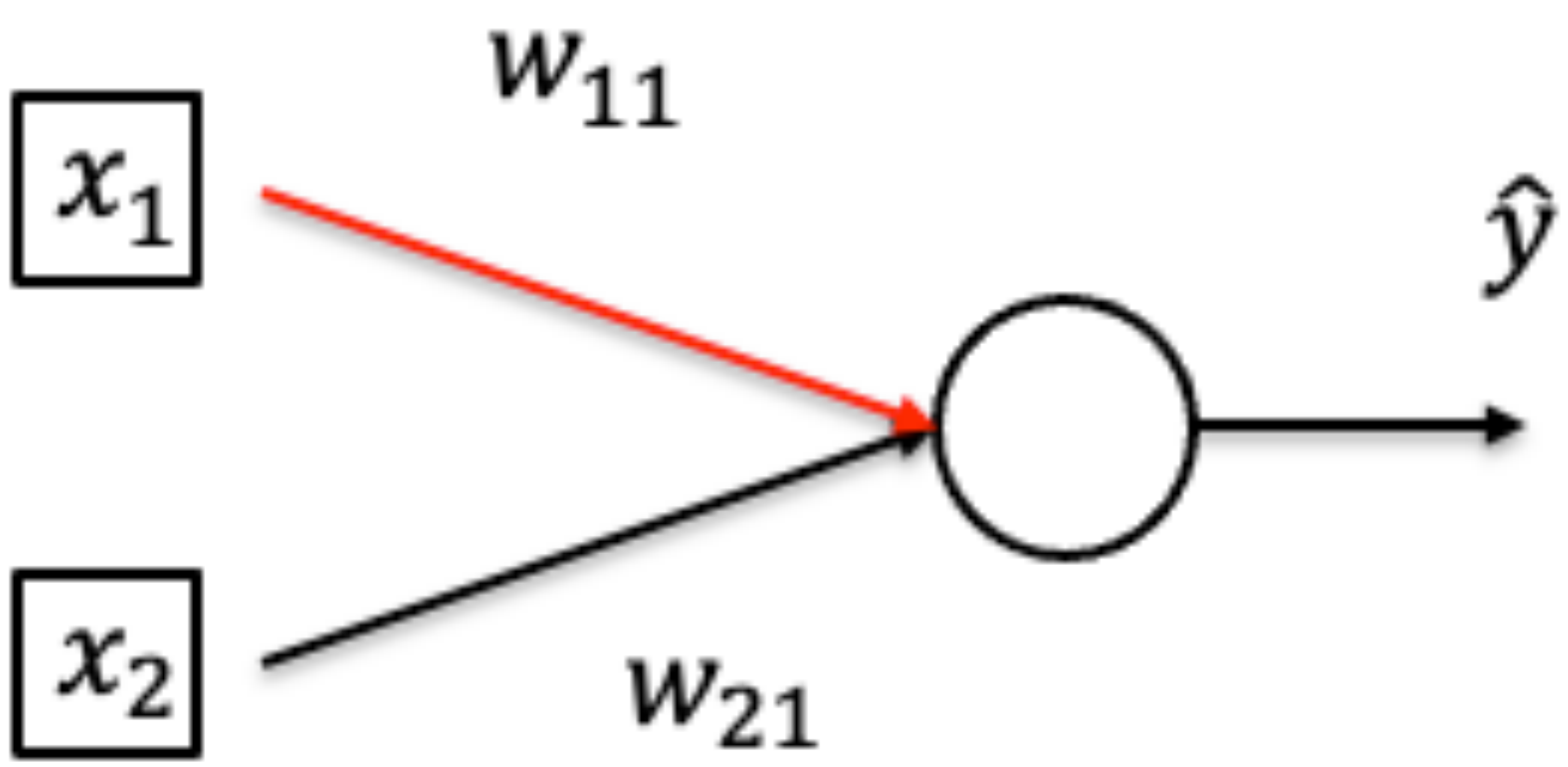


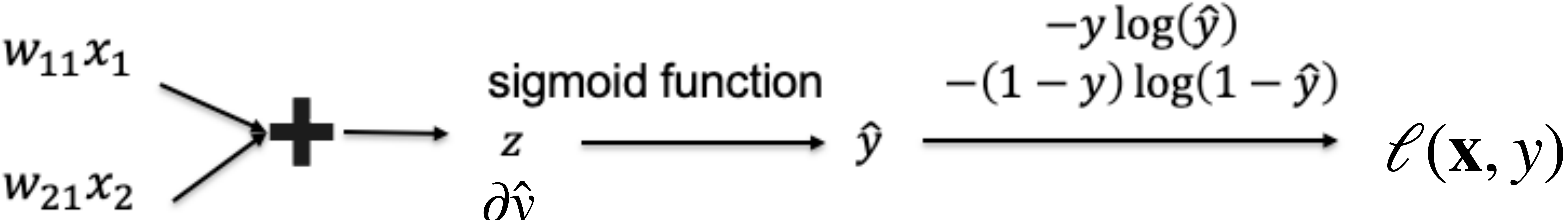


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

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

Calculate Gradient (on one data point)

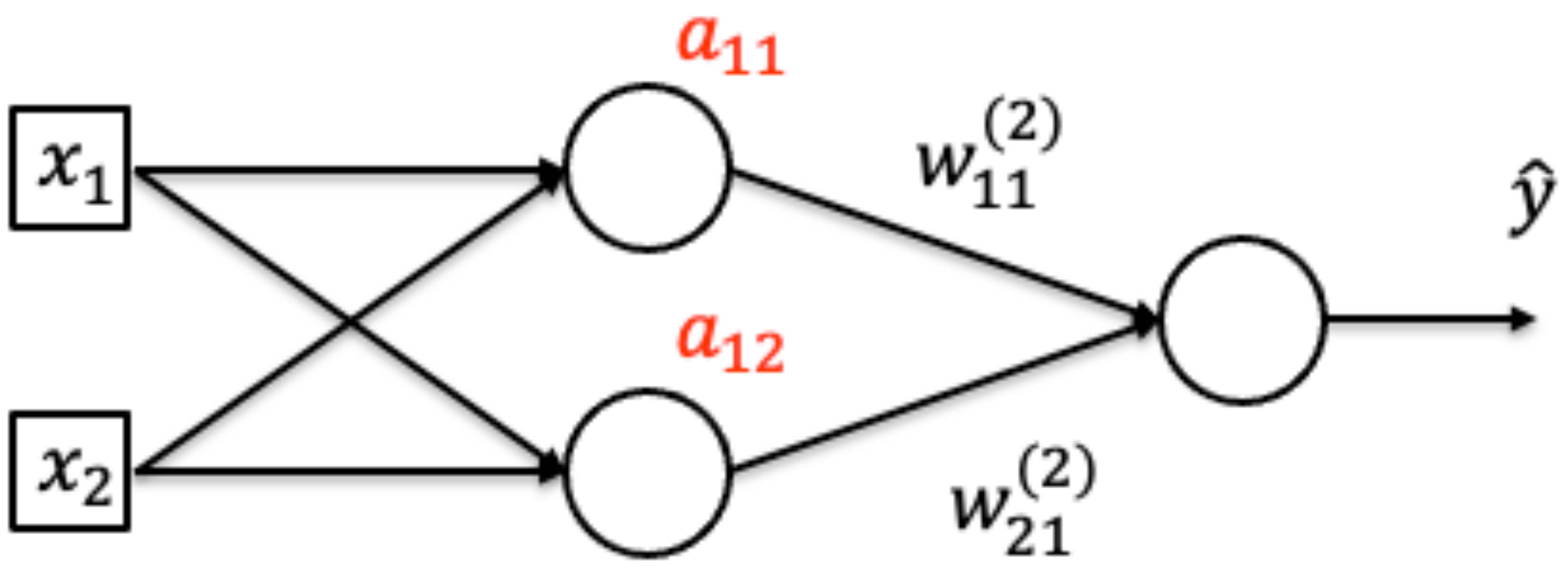




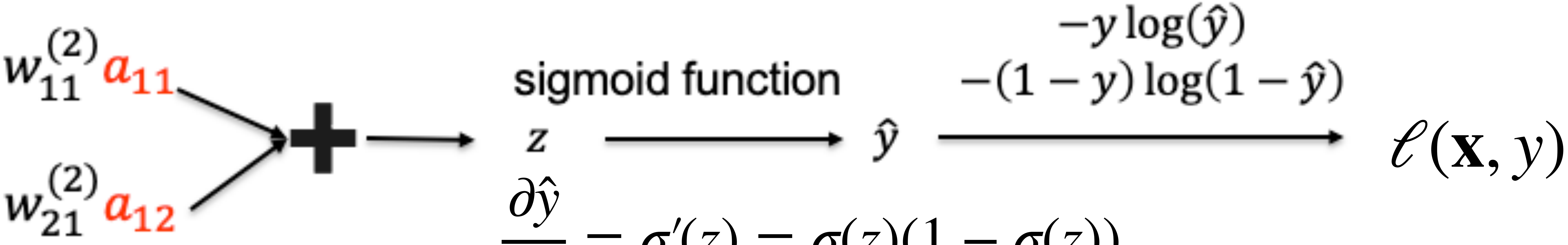
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

- 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)



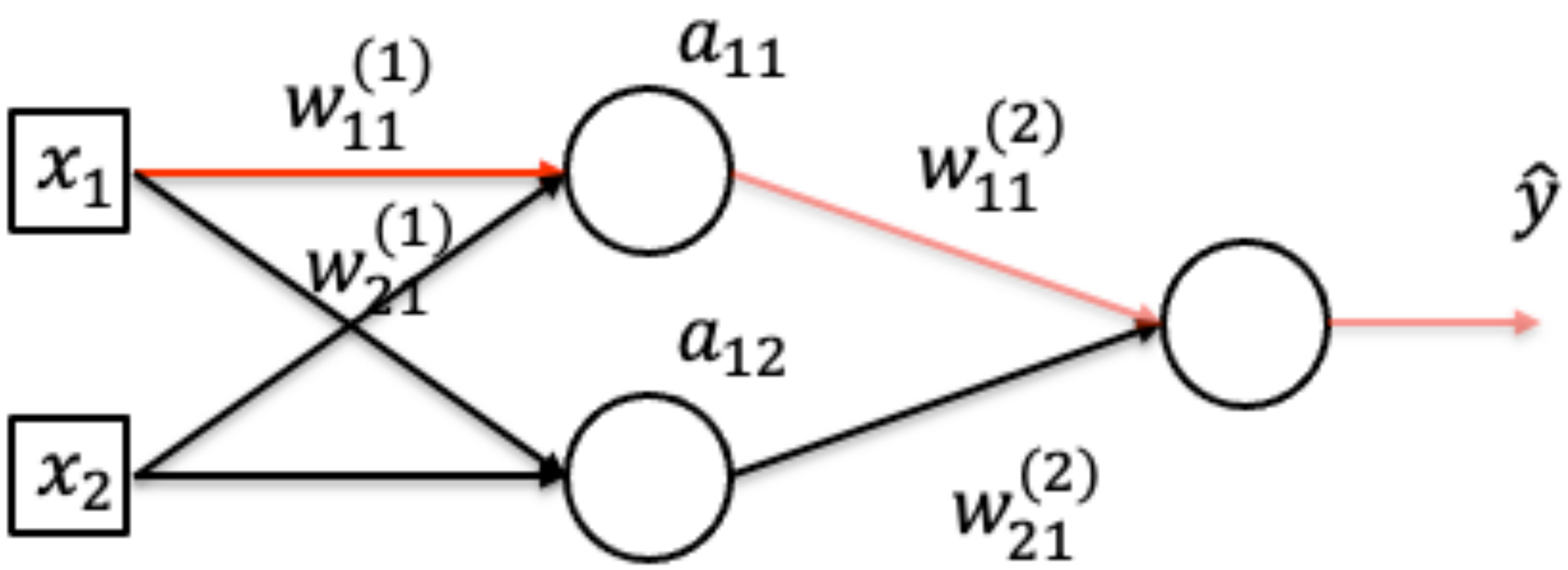
Make it deeper



$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

- 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)

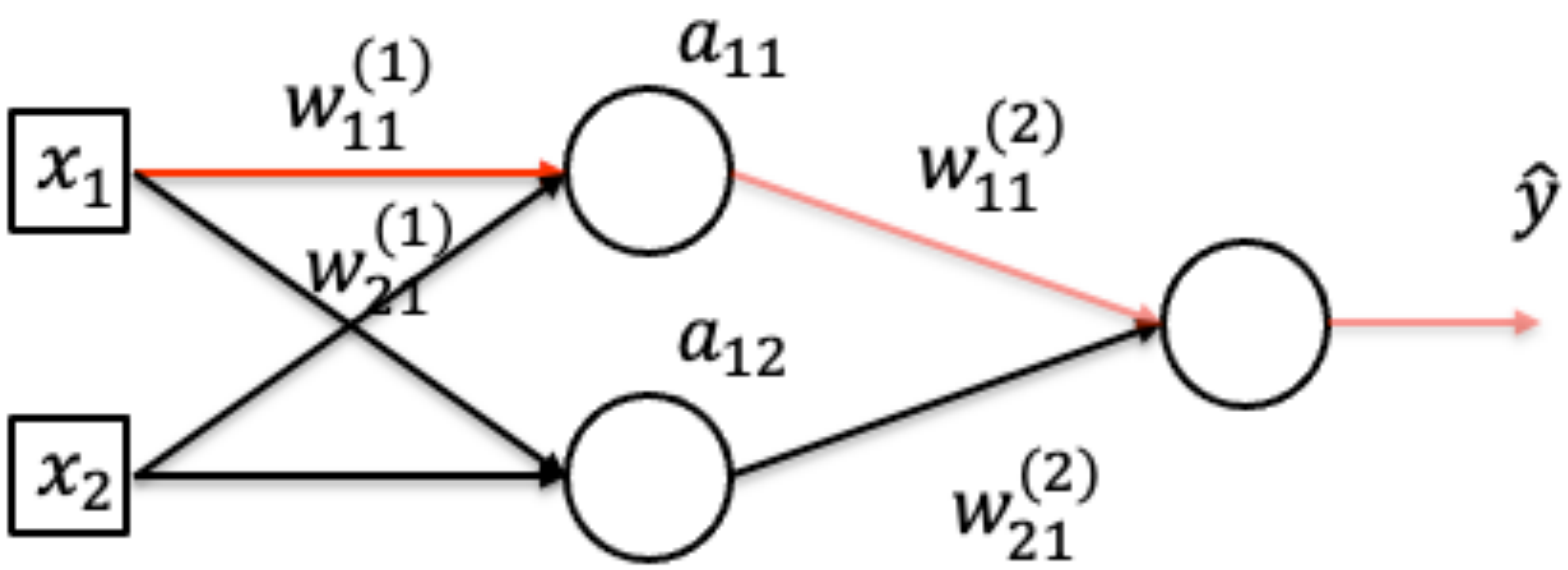


$$w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 \longrightarrow z_{11} \xrightarrow{\sigma(z_{11})} a_{11} \longrightarrow l(\boldsymbol{x}, y)$$

$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \qquad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y) w_{11}^{(2)}$$

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

Calculate Gradient (on one data point)

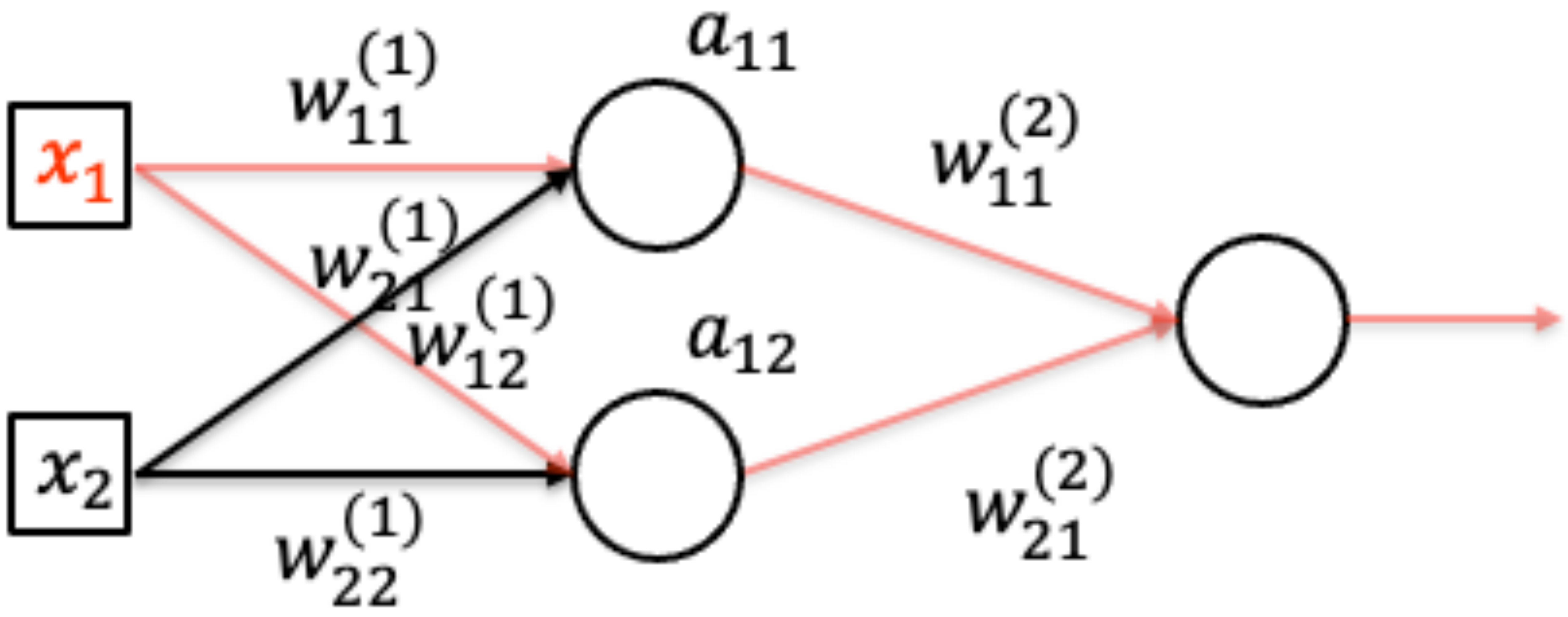


$$\left.\begin{array}{l} w_{11}^{(1)}x_1 \\ w_{21}^{(1)}x_2 \end{array}\right\} + \longrightarrow z_{11} \xrightarrow{\sigma(z_{11})} a_{11} \longrightarrow l(\boldsymbol{x}, y)$$

$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \qquad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}$$

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

Calculate Gradient (on one data point)

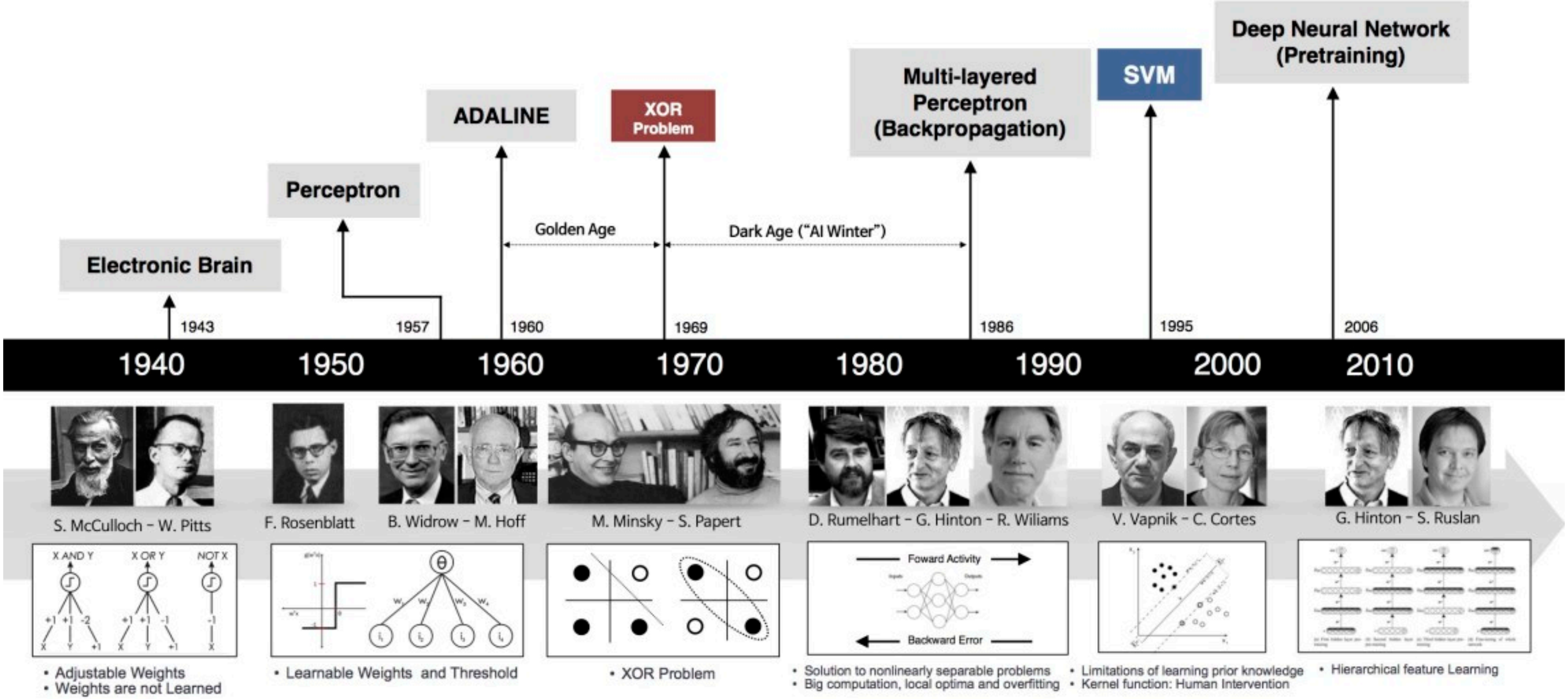


$$w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2 \longrightarrow z_{11} \xrightarrow{\sigma(z_{11})} a_{11} \longrightarrow l(\boldsymbol{x}, y)$$

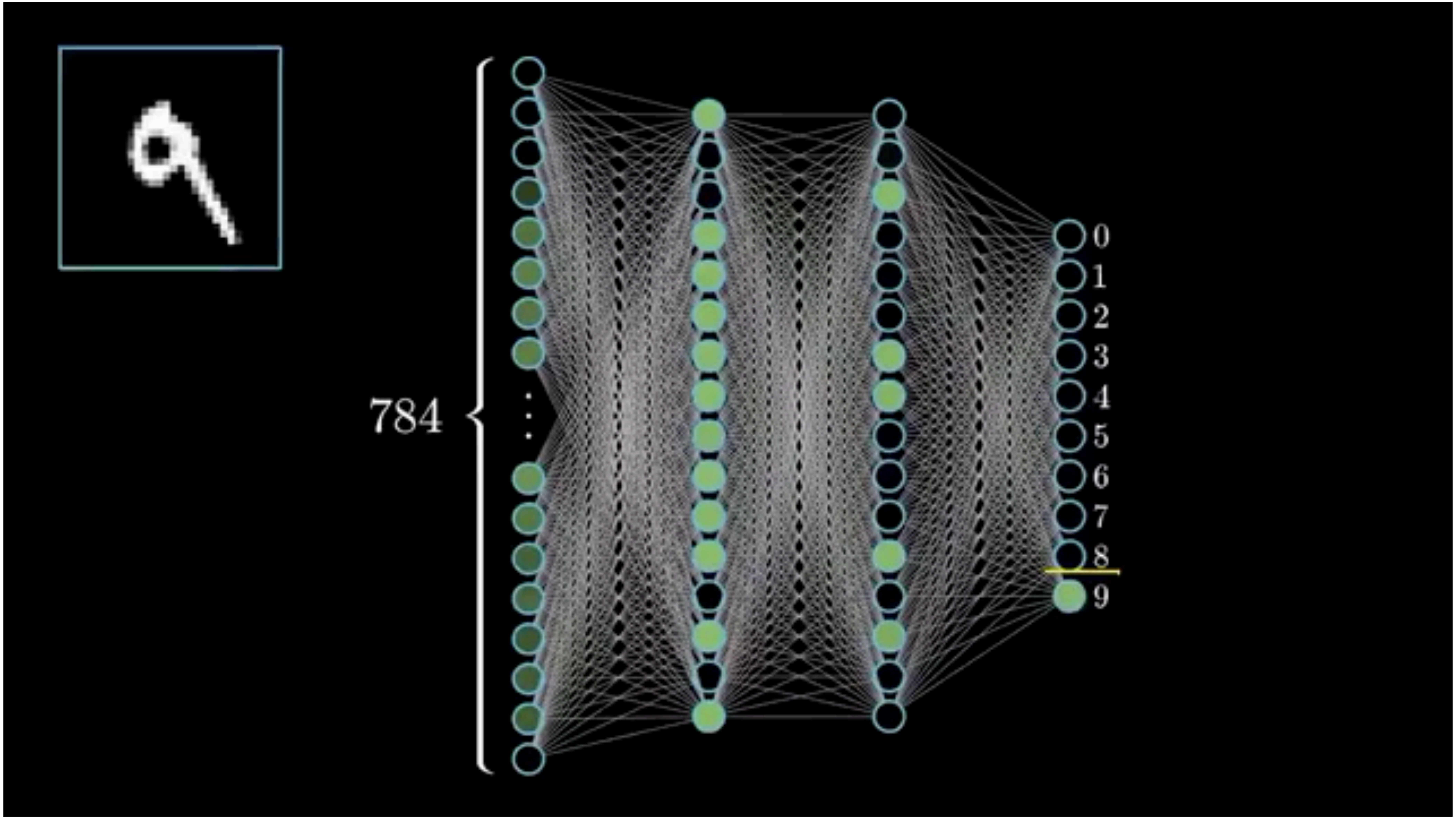
$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \qquad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}$$

- By chain rule: $\dfrac{\partial l}{\partial x_1} = \dfrac{\partial l}{\partial a_{11}}\dfrac{\partial a_{11}}{\partial x_1} + \dfrac{\partial l}{\partial a_{12}}\dfrac{\partial a_{12}}{\partial x_1}$

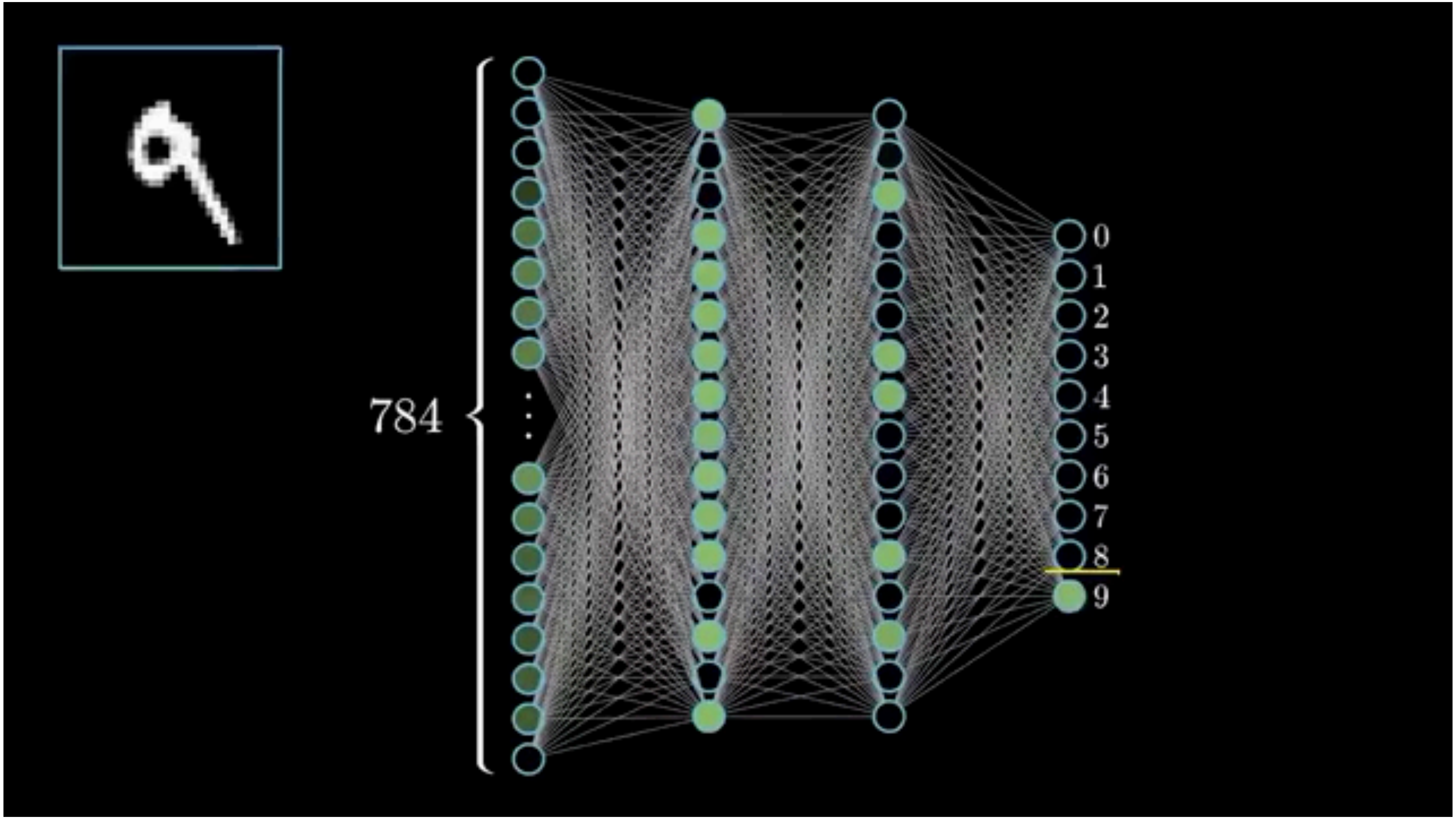
Brief history of neural networks



HW6

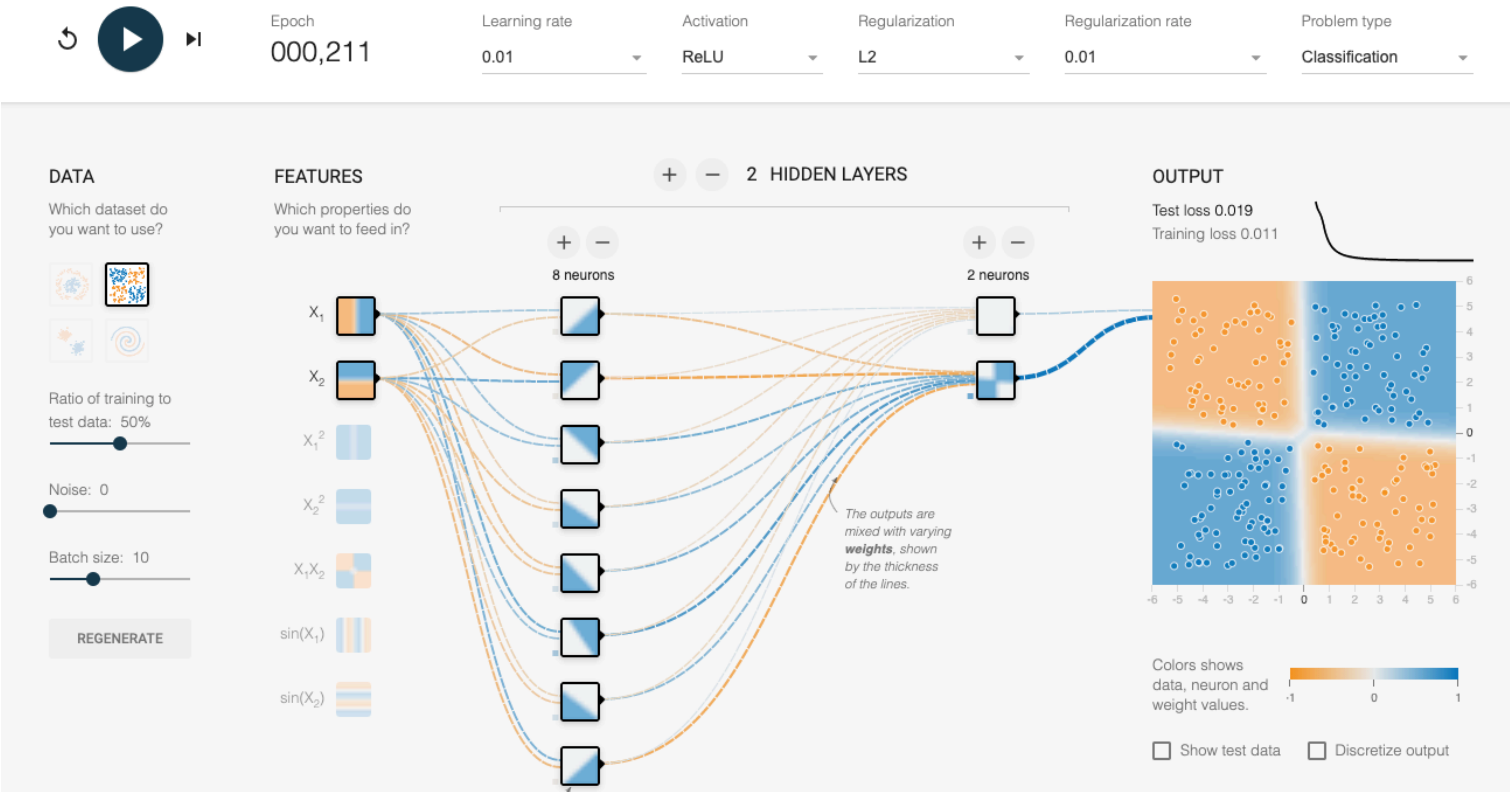


HW6



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!