CS 540 Introduction to Artificial Intelligence

Neural Networks (II)

University of Wisconsin-Madison

Spring 2022

Today’s outline

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

Review: Perceptron

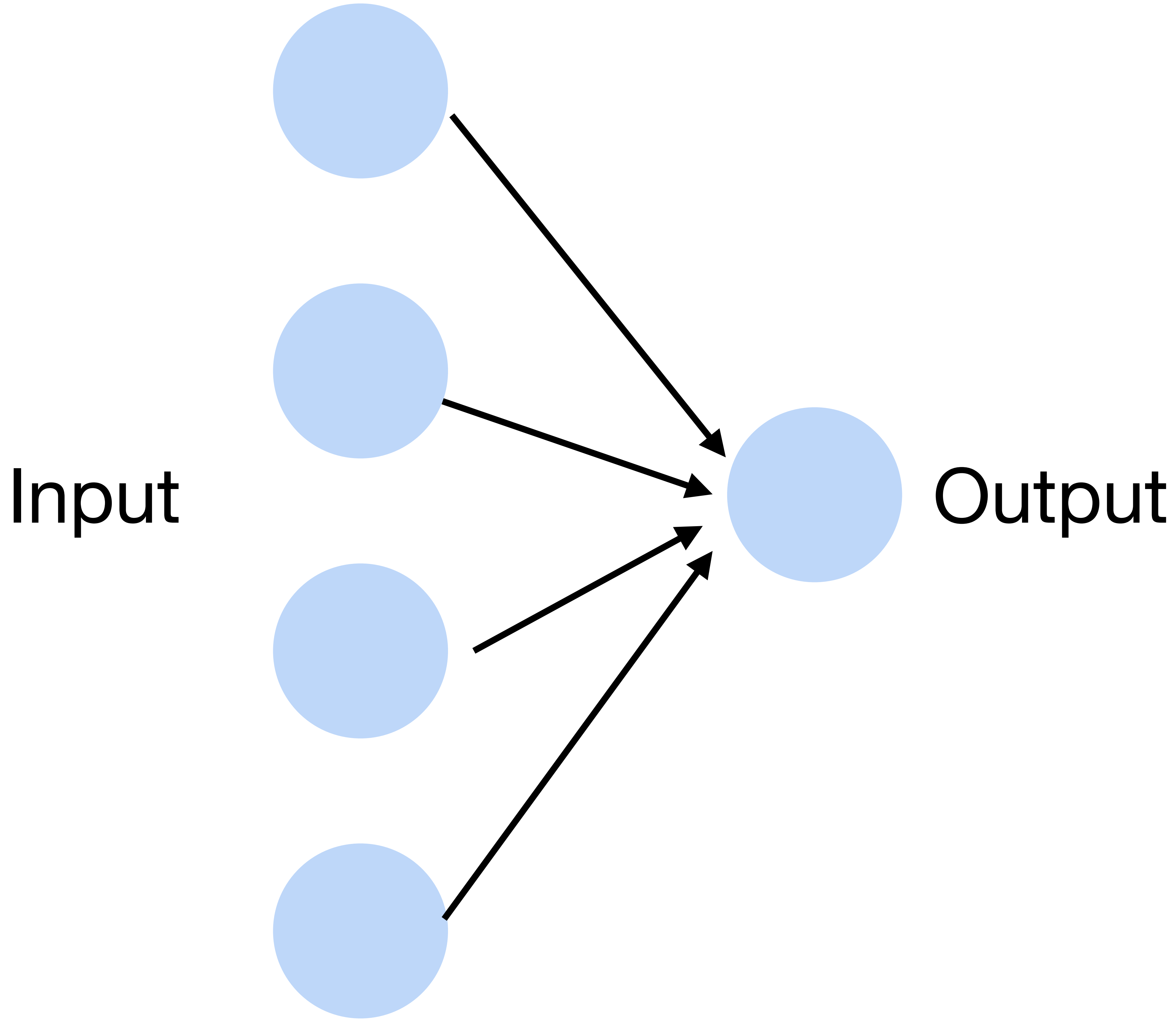
- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias b, perceptron outputs:

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

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

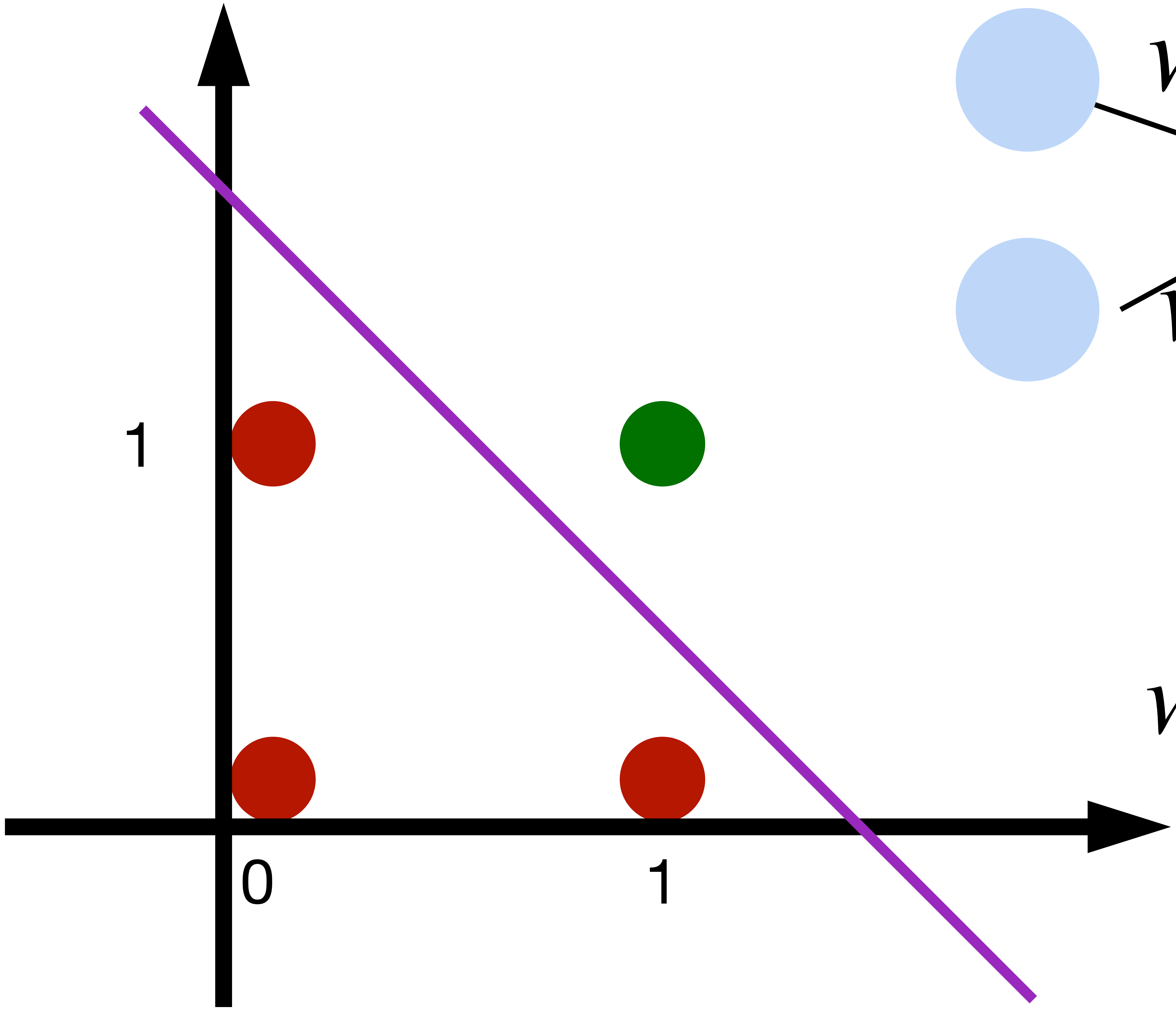
Activation function

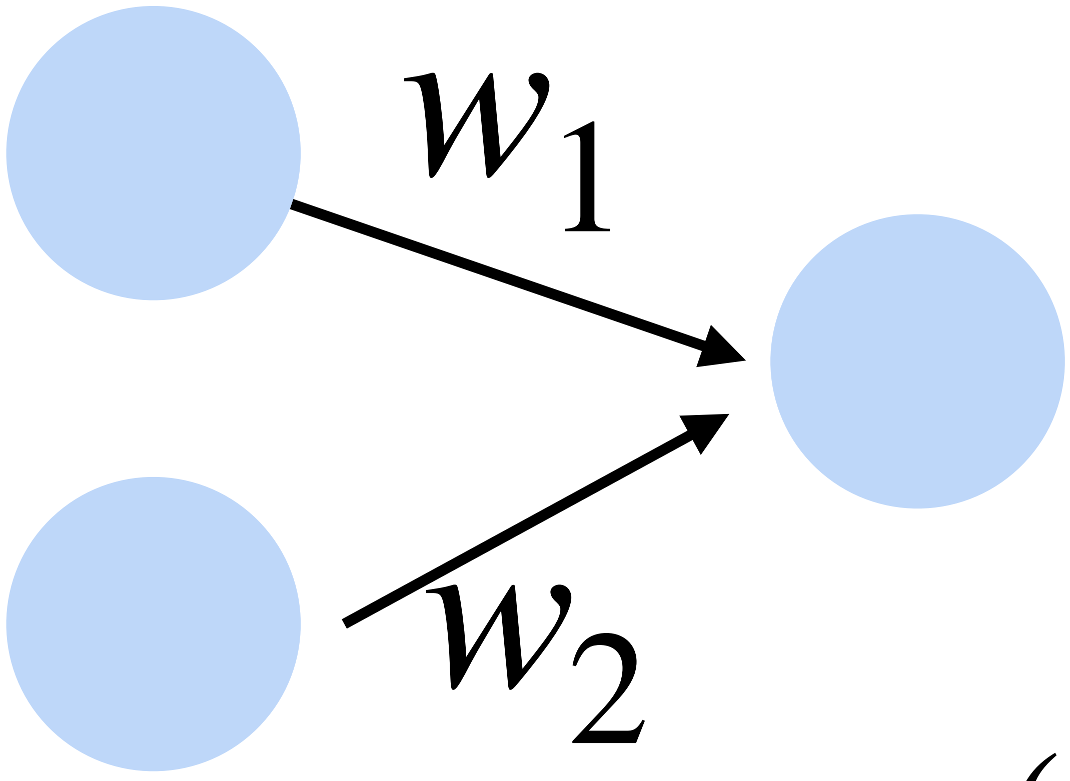
Cats vs. dogs?



Learning AND function using perceptron

The perceptron can learn an AND function





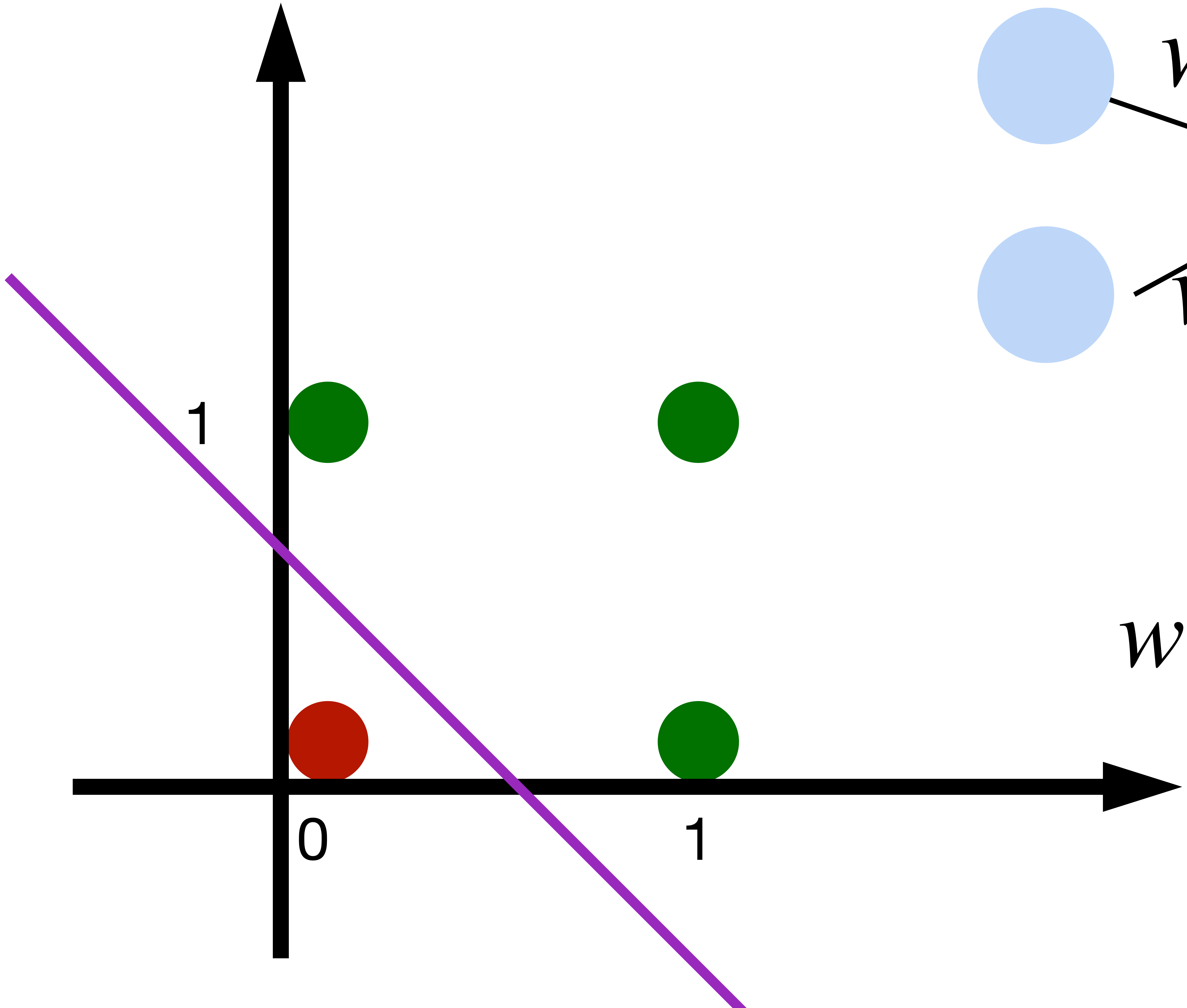
Output $\sigma(x_1 w_1 + x_2 w_2 + b)$

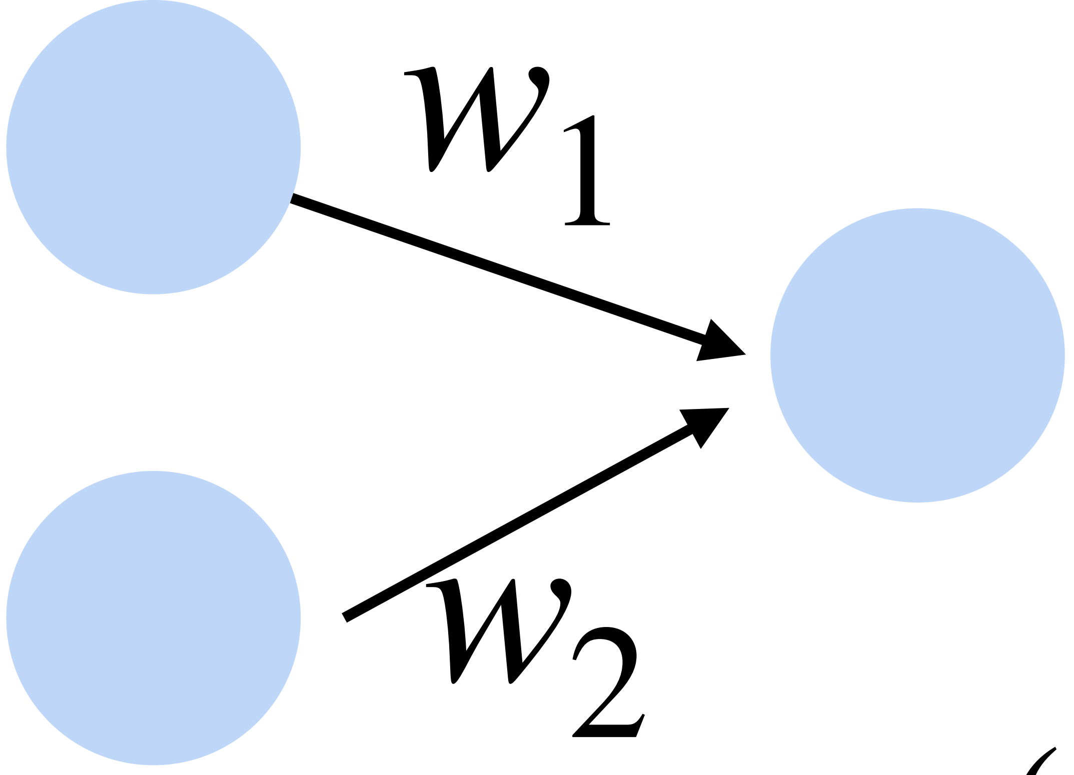
$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$w_1 = 1, w_2 = 1, b = -1.5$$

Learning OR function using perceptron

The perceptron can learn an OR function





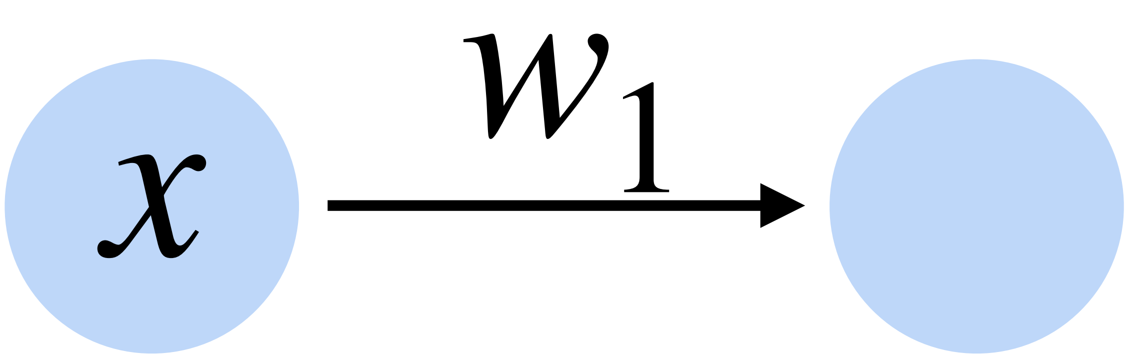
Output $\sigma(x_1 w_1 + x_2 w_2 + b)$

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

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

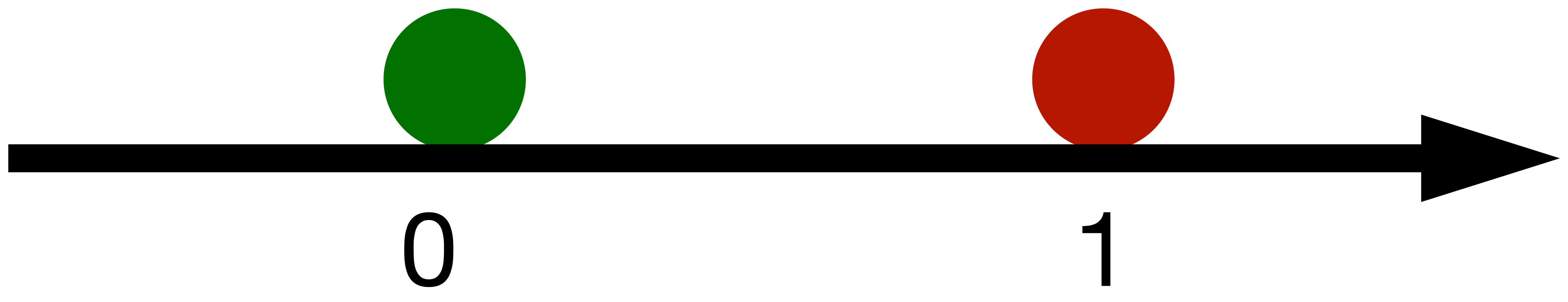
Learning NOT function using perceptron

The perceptron can learn NOT function (single input)



Output $\sigma(xw_1 + b)$

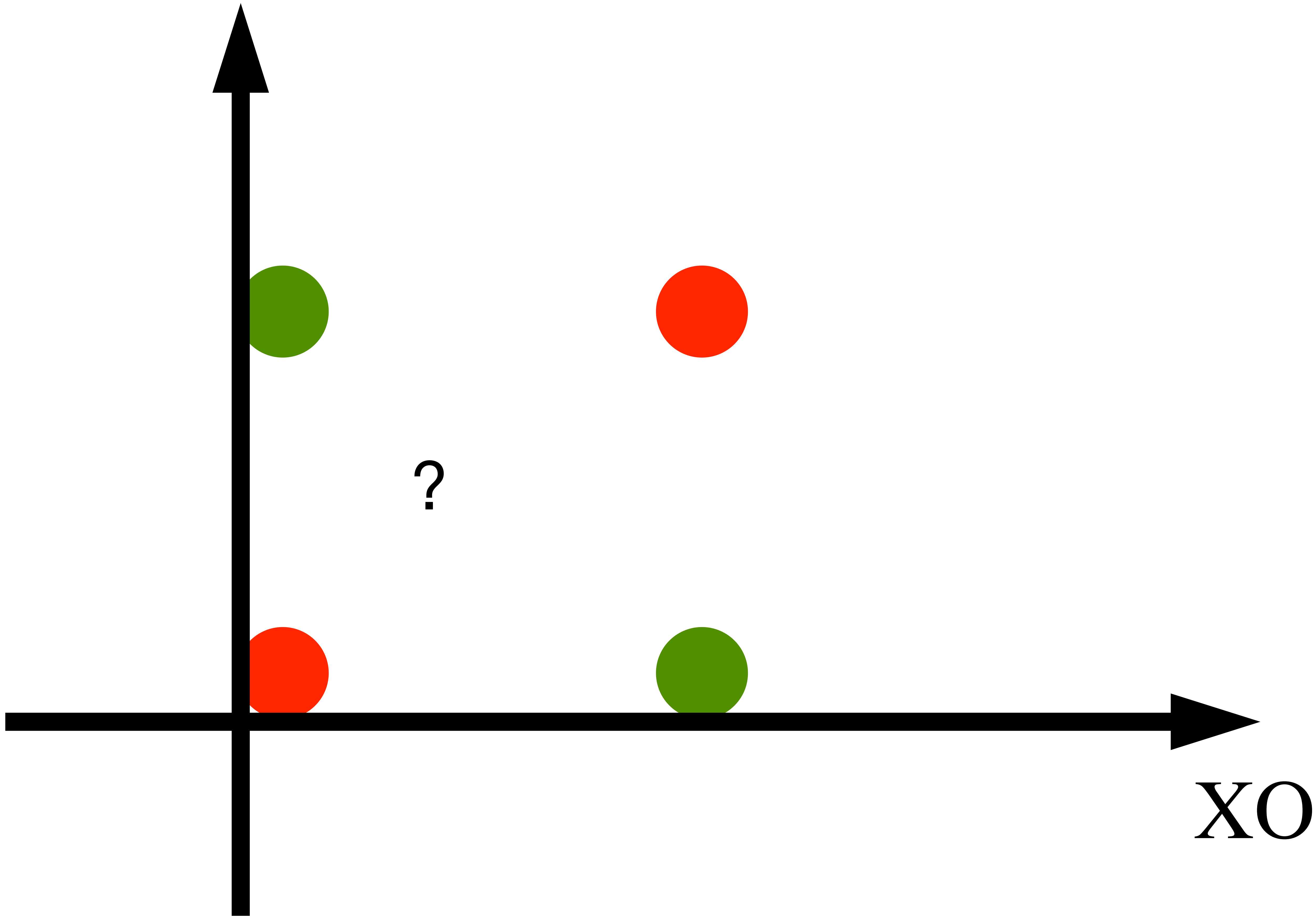
$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$



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

The limited power of a single neuron

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



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

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

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

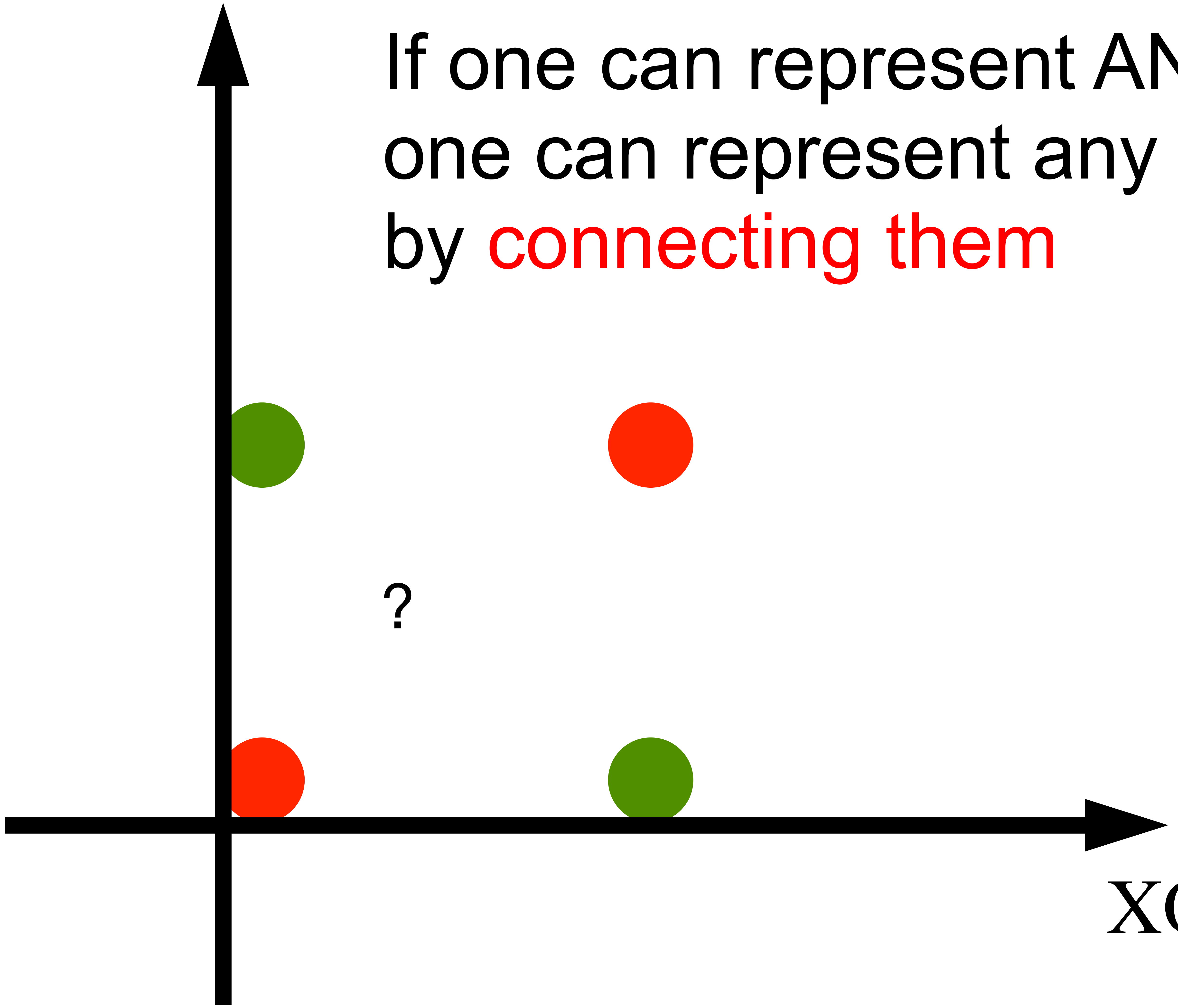
$$x_1 = 0, x_2 = 0, y = 0$$

XOR(x1,x2)=(x1∧¬x2)∨(¬x1∧x2)

The limited power of a single neuron

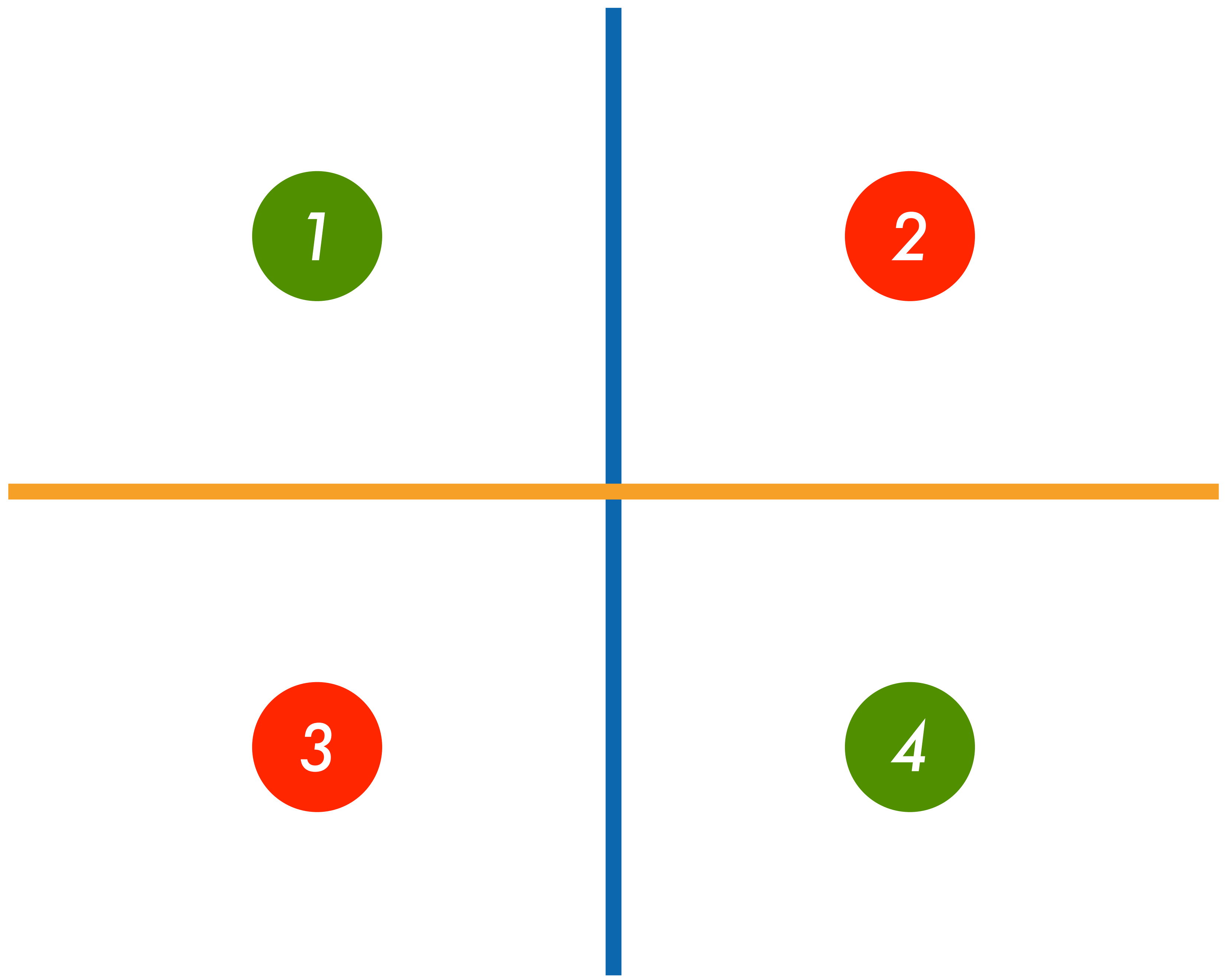
XOR problem

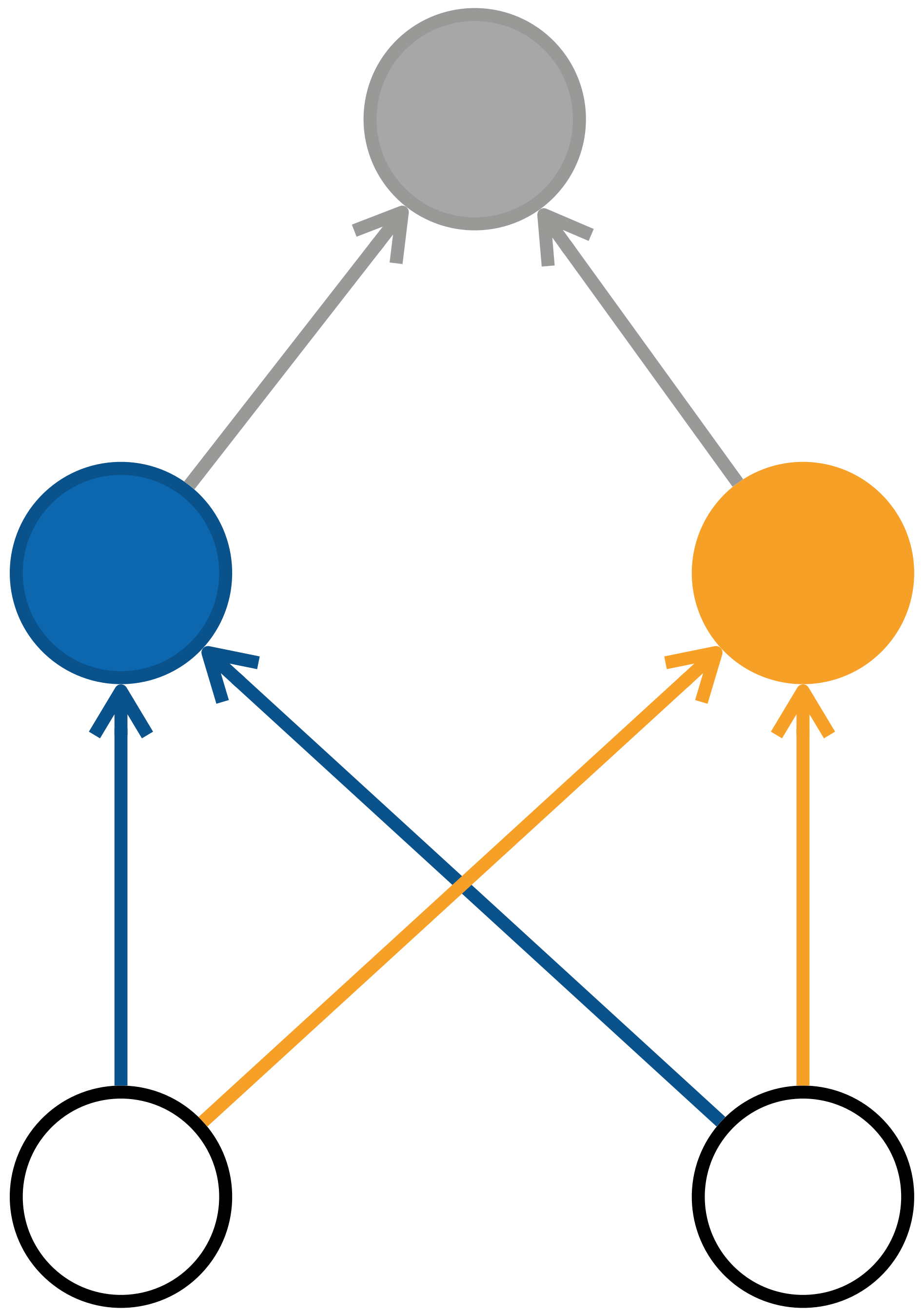
If one can represent AND, OR, NOT,
one can represent any logic circuit (including XOR),
by connecting them



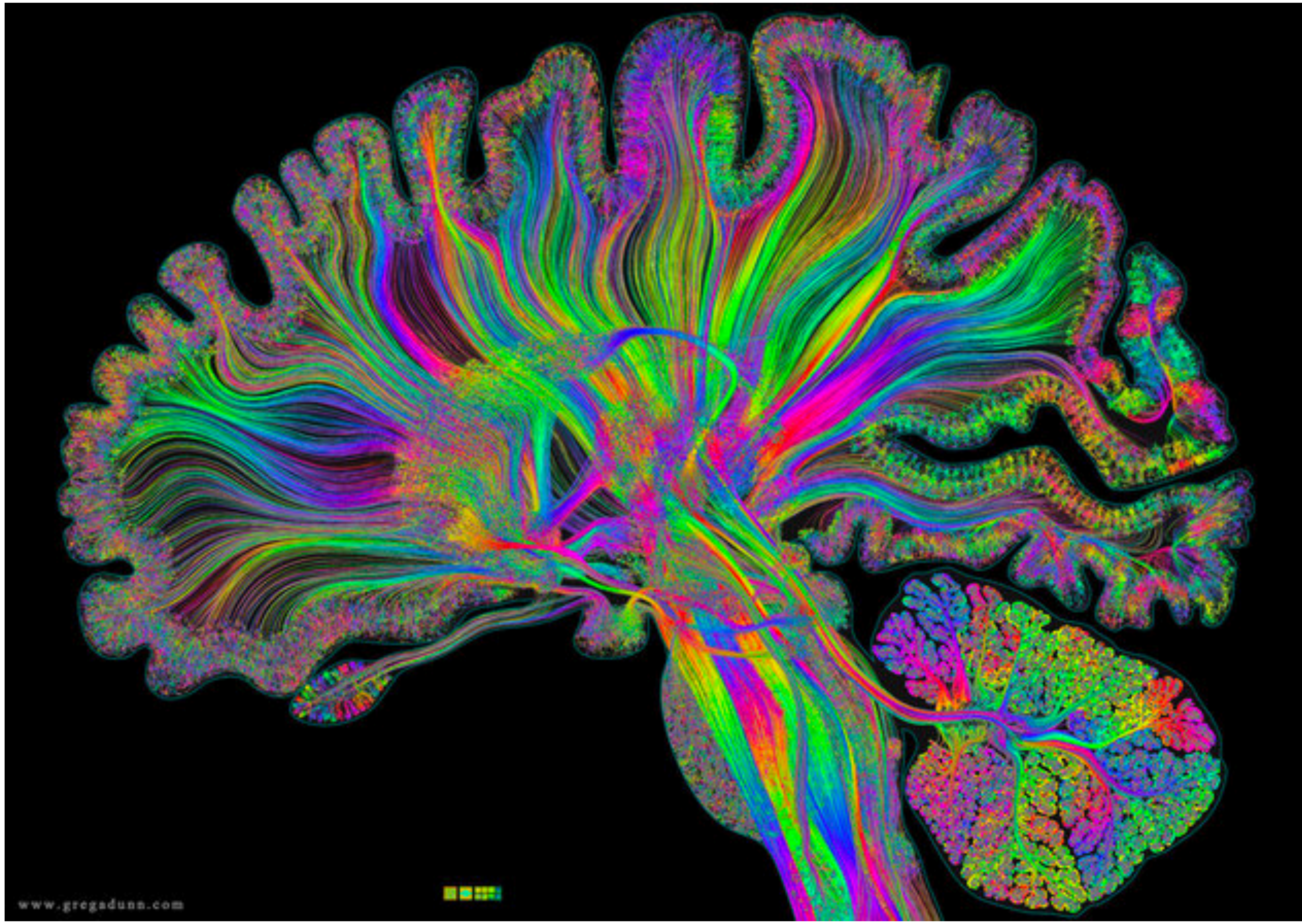
XOR(x1,x2)=(x1∧¬x2)∨(¬x1∧x2)

Learning XOR



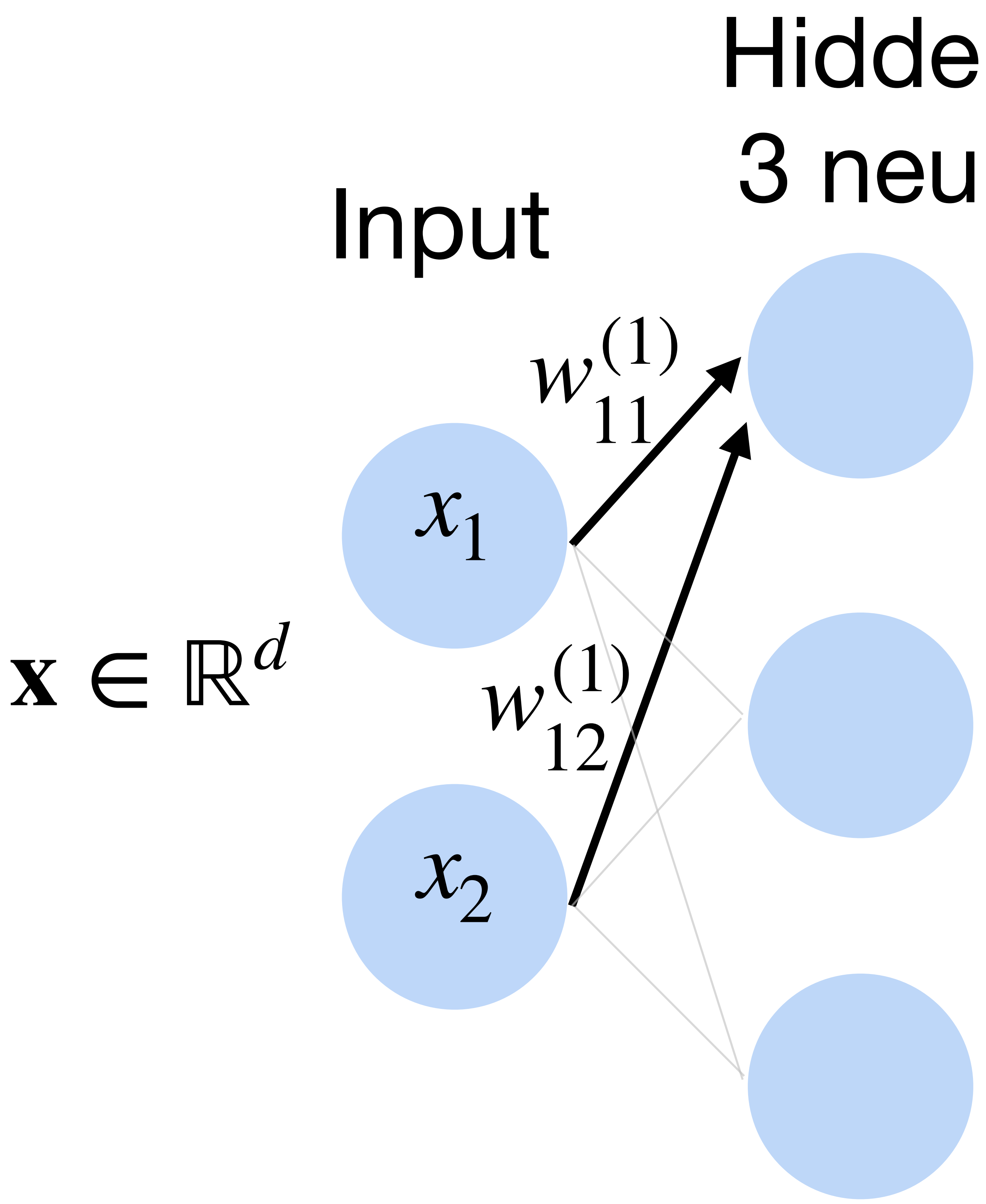


Multilayer Perceptron



Multi-layer perceptron: Example

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



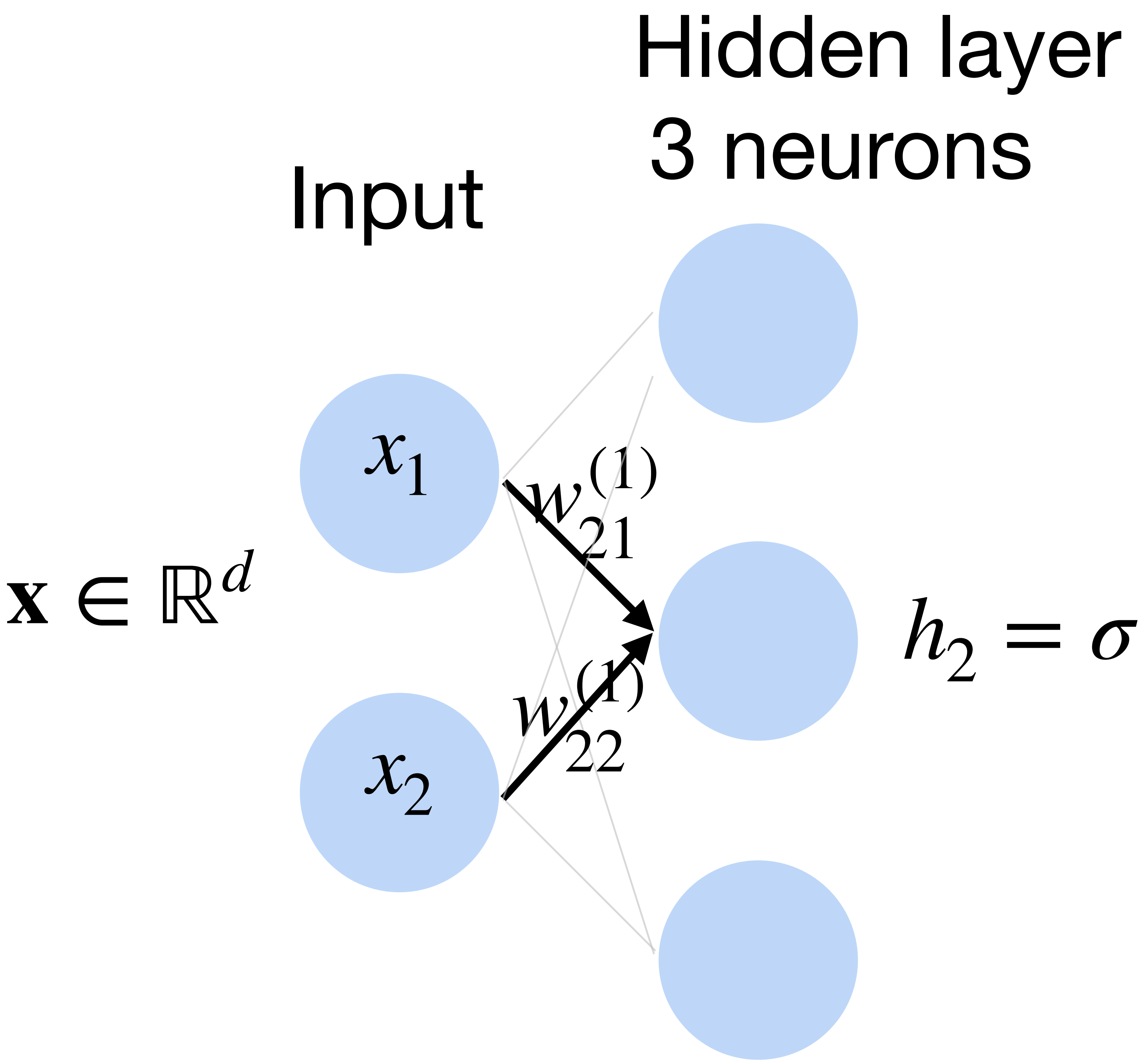
Hidden layer
3 neurons

Input

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

Multi-layer perceptron: Example

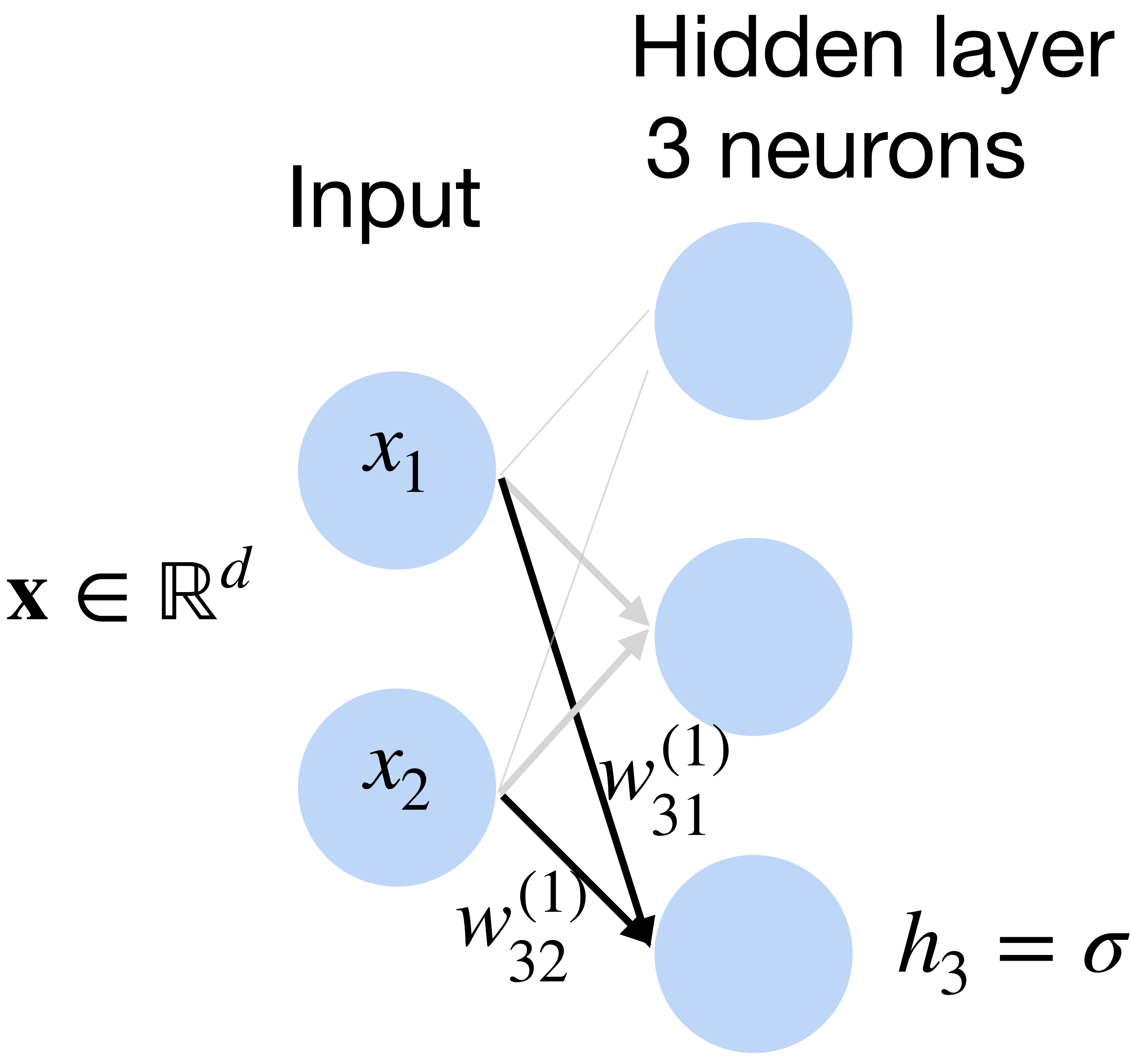
- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



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

Multi-layer perceptron: Example

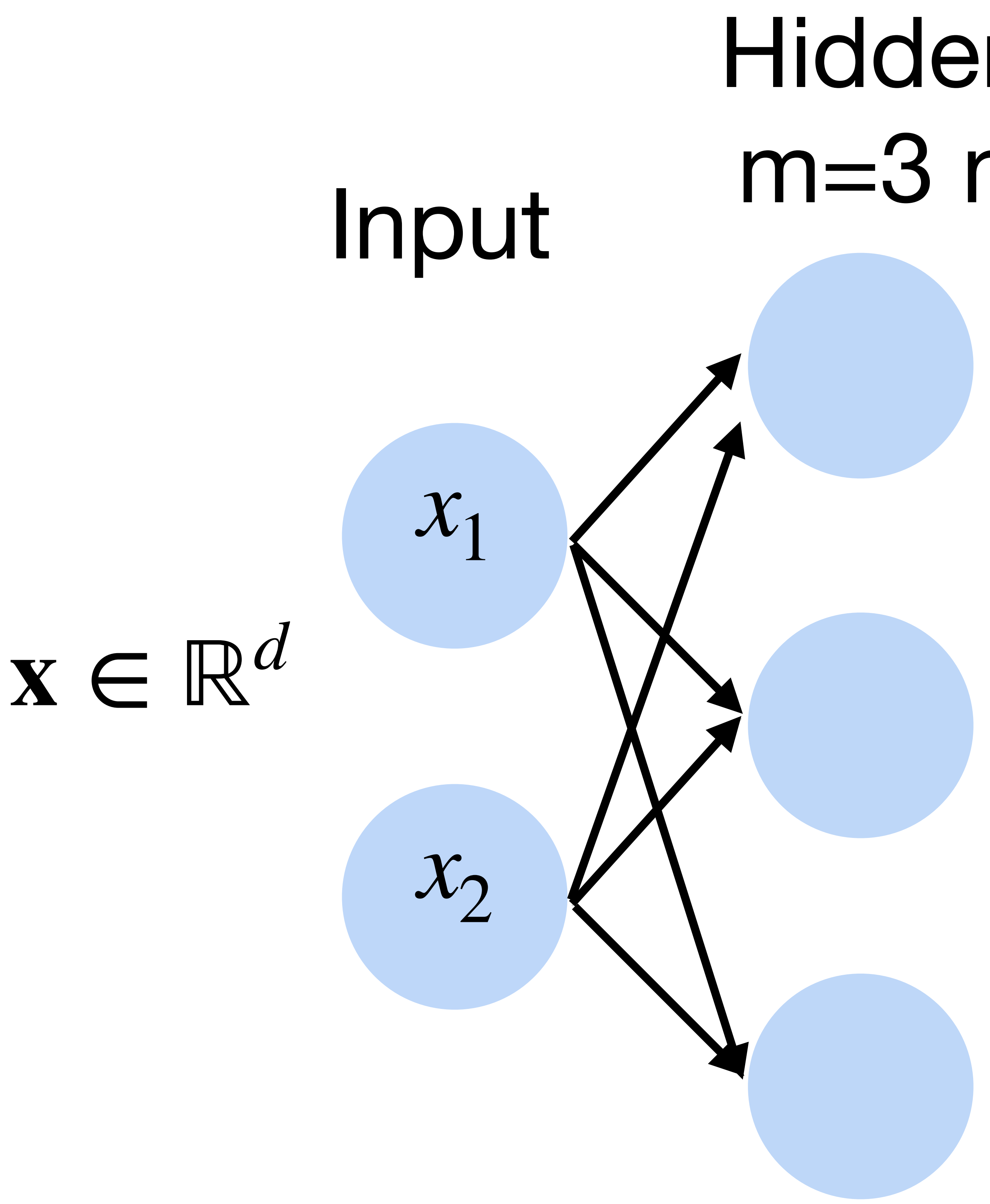
- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



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

Multi-layer perceptron: Example

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



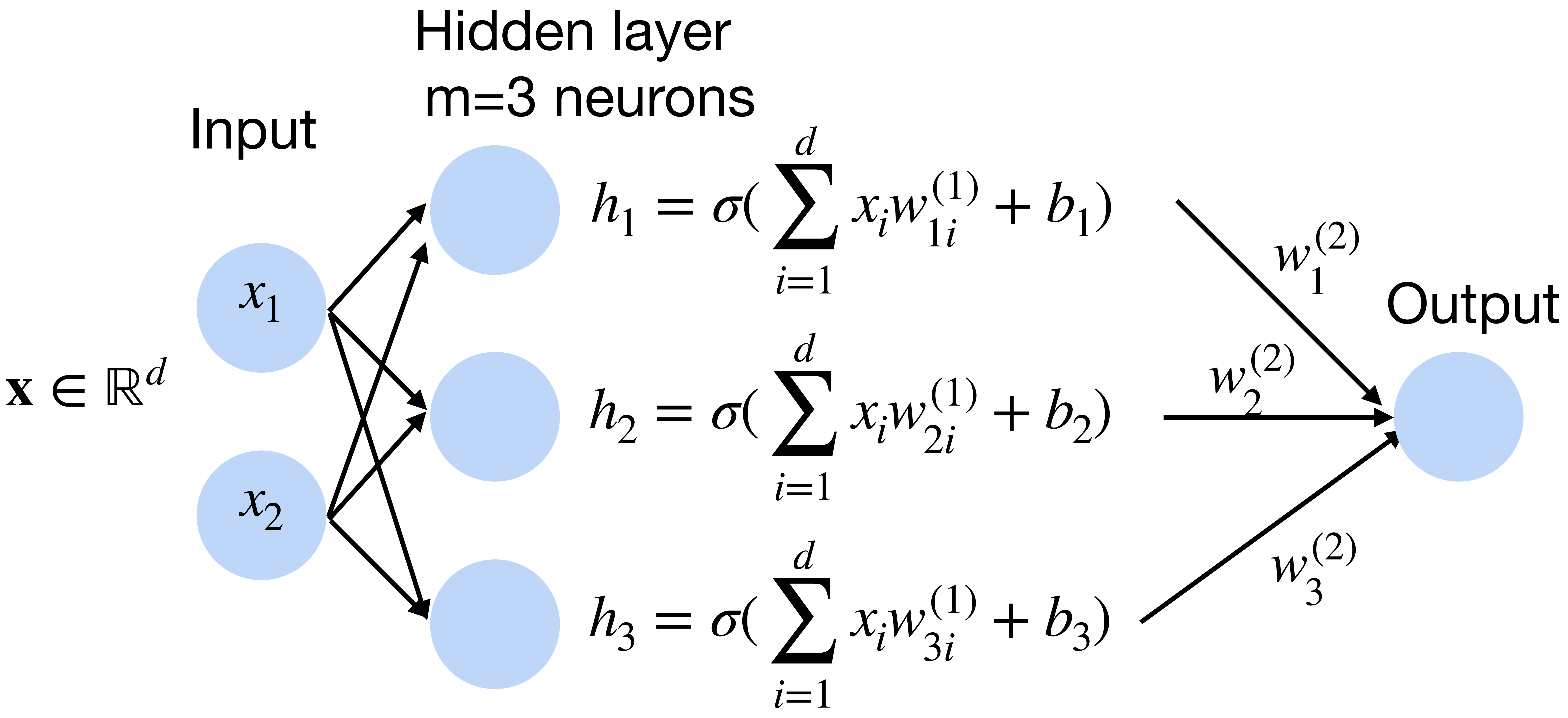
$$h_1 = \sigma(\sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1)$$

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

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

Multi-layer perceptron: Example

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



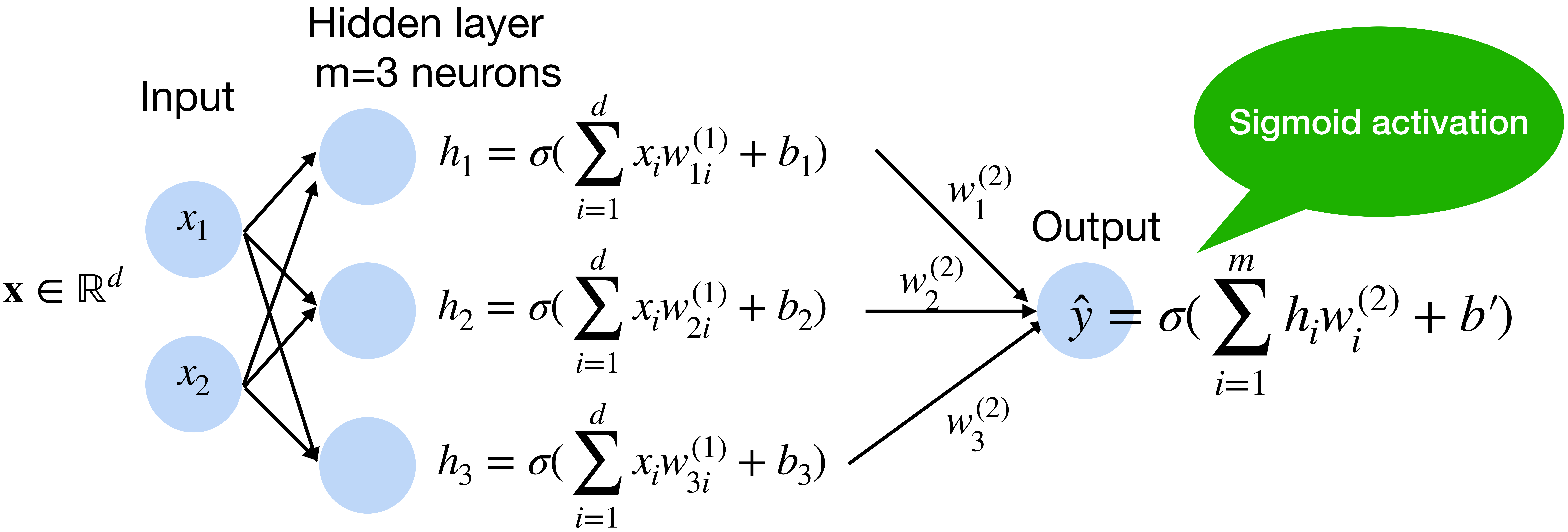
$$h_1 = \sigma(\sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1)$$

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

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

Multi-layer perceptron: Example

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



Input

Hidden layer
m=3 neurons

Output

Sigmoid activation

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

x_1 x_2

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

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

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

$w_1^{(2)}$ $w_2^{(2)}$ $w_3^{(2)}$

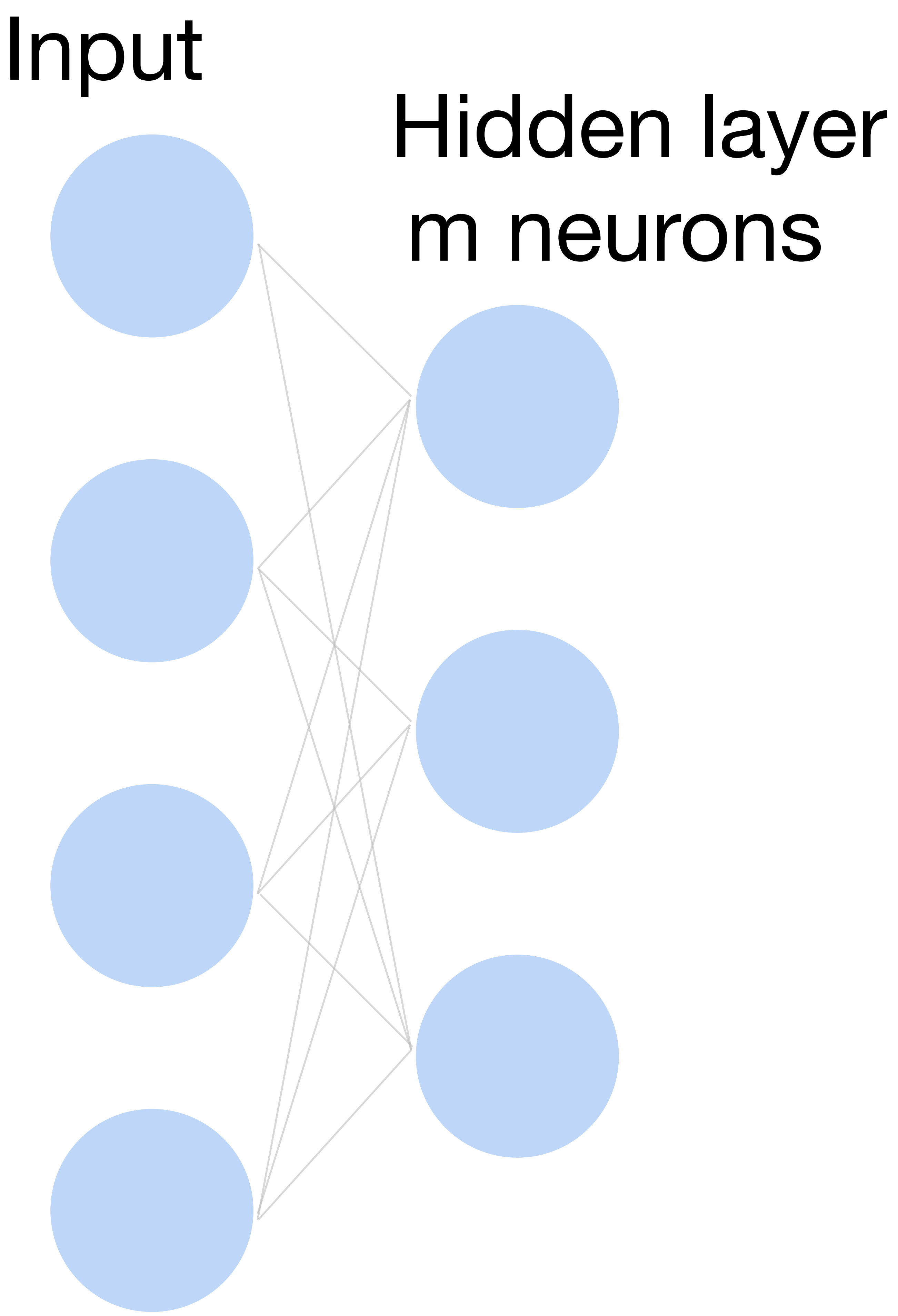
$$\hat{y} = \sigma(\sum_{i=1}^{m} h_i w_i^{(2)} + b')$$

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

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

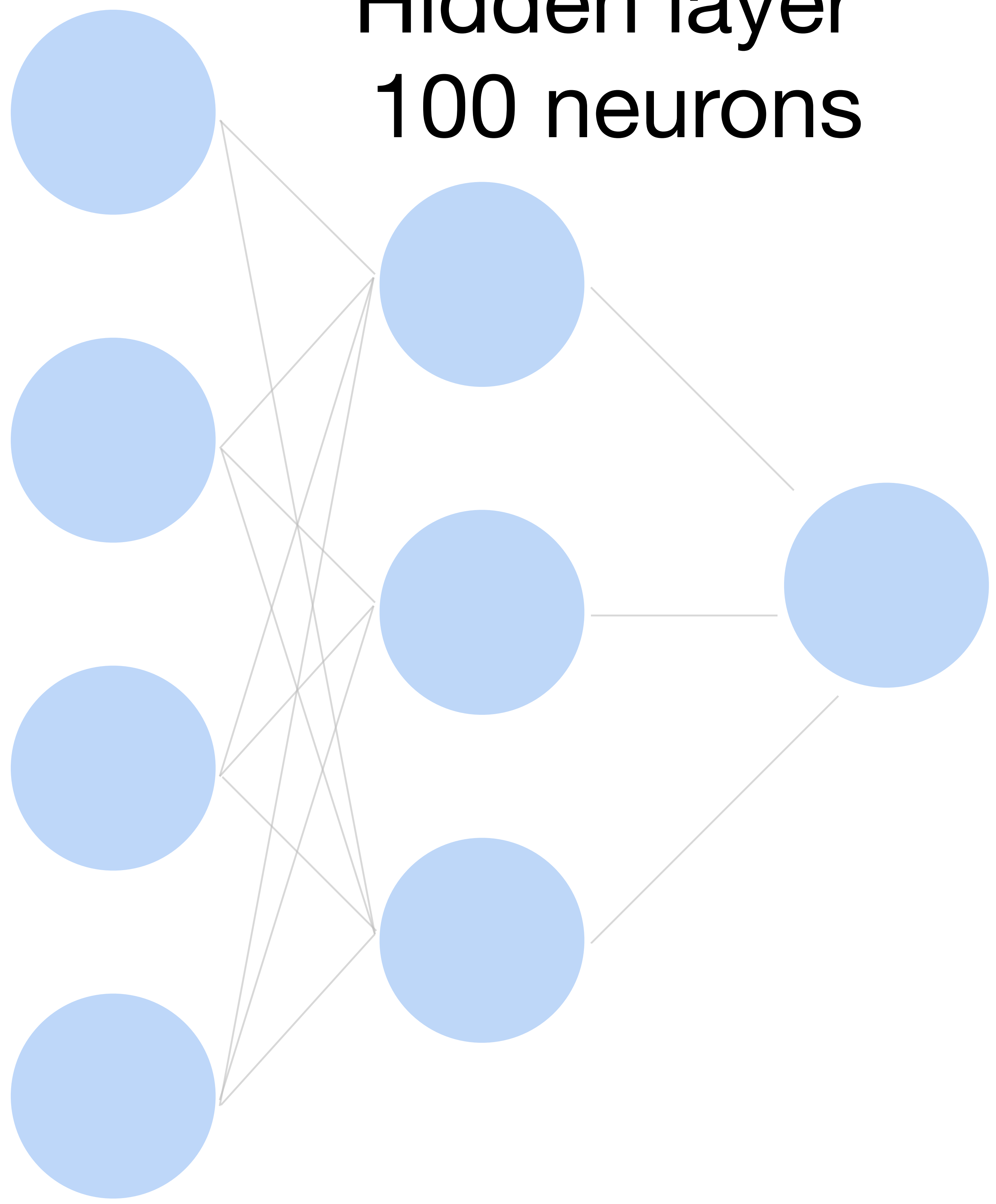
$$\mathbf{h} \in \mathbb{R}^m$$



Classify cats vs. dogs

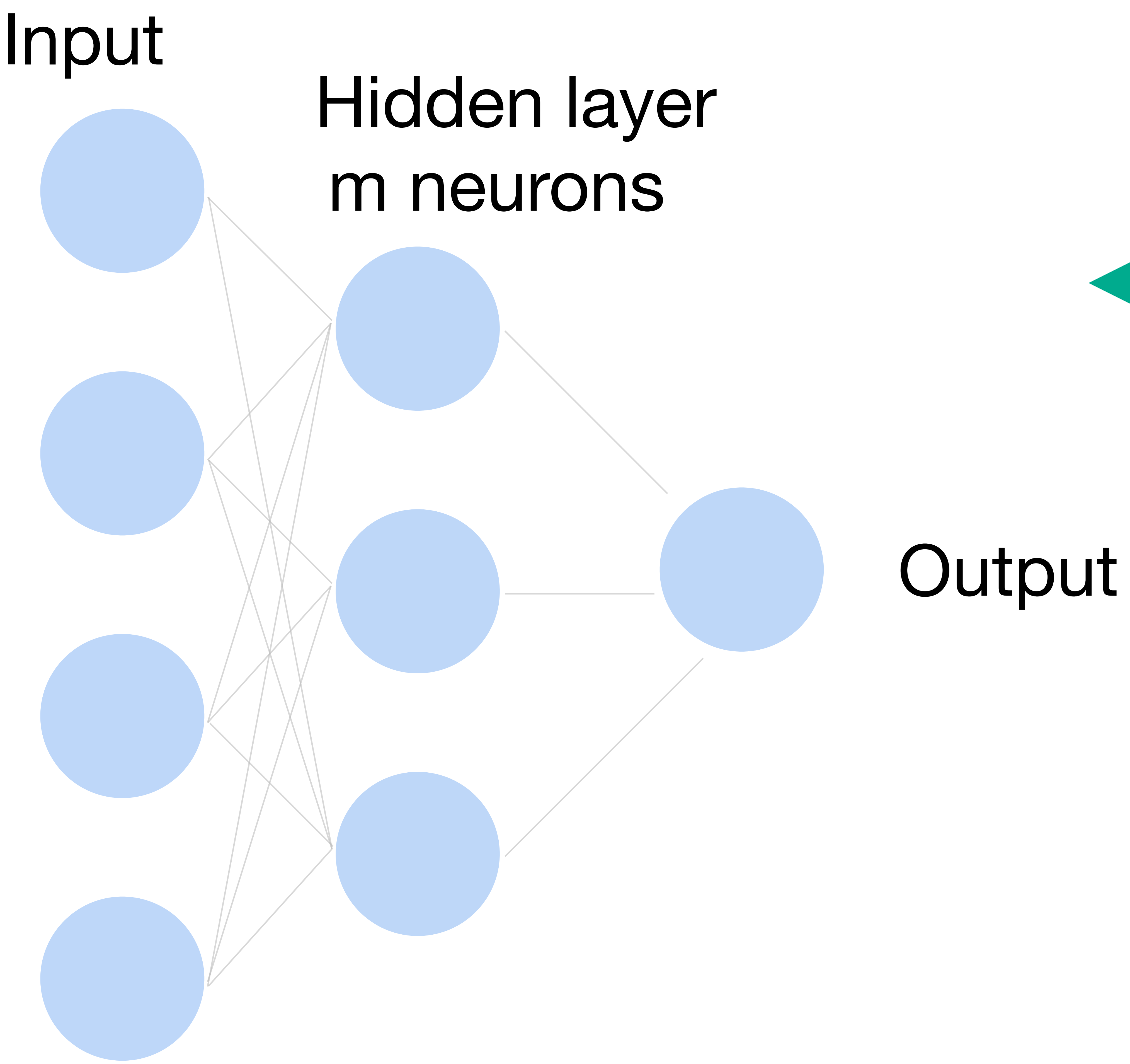
Input

Hidden layer
100 neurons



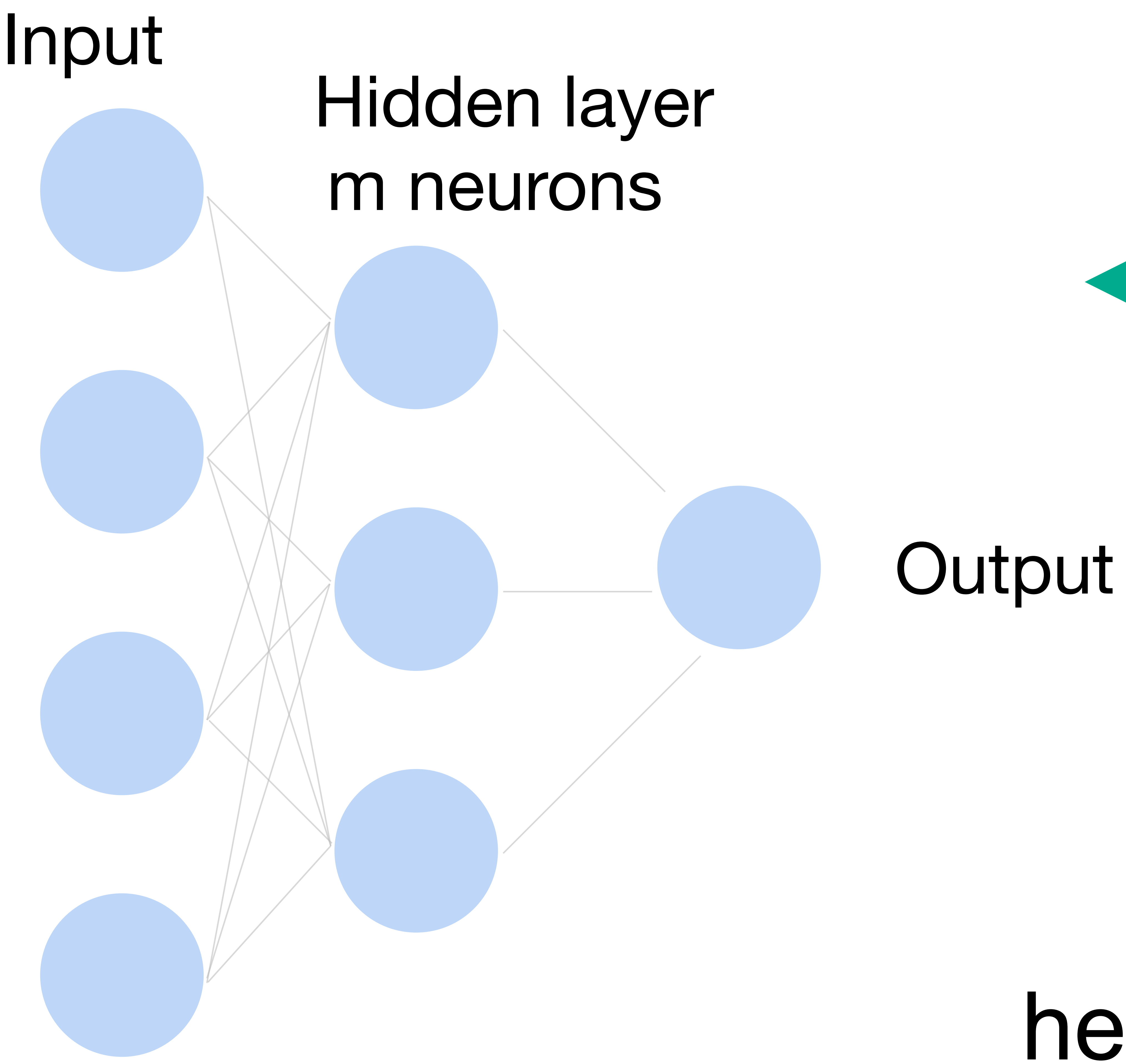
Output

Multi-layer perceptron



Why do we need an a nonlinear activation?

Multi-layer perceptron



Why do we need an a nonlinear activation?

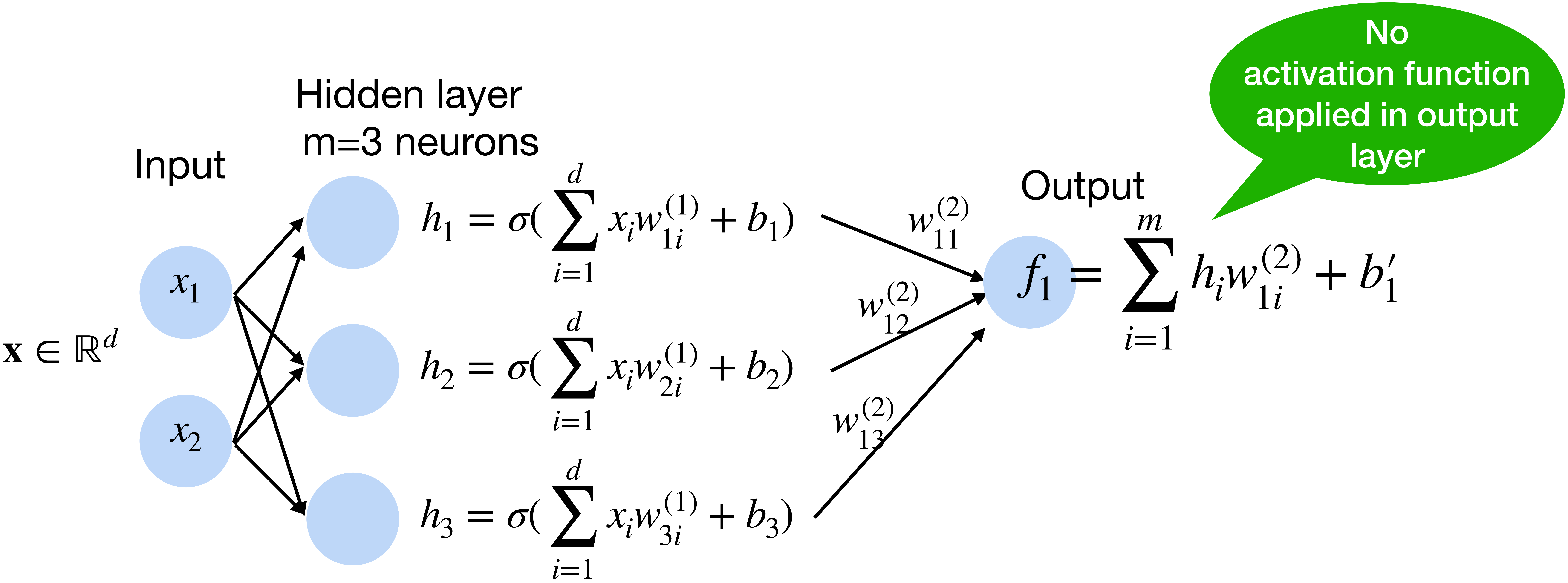
$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

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

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

Neural network for k-way classification

- K outputs in the final layer



Hidden layer
m=3 neurons

Input

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

x_1

x_2

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

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

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

$w_{11}^{(2)}$

$w_{12}^{(2)}$

$w_{13}^{(2)}$

Output

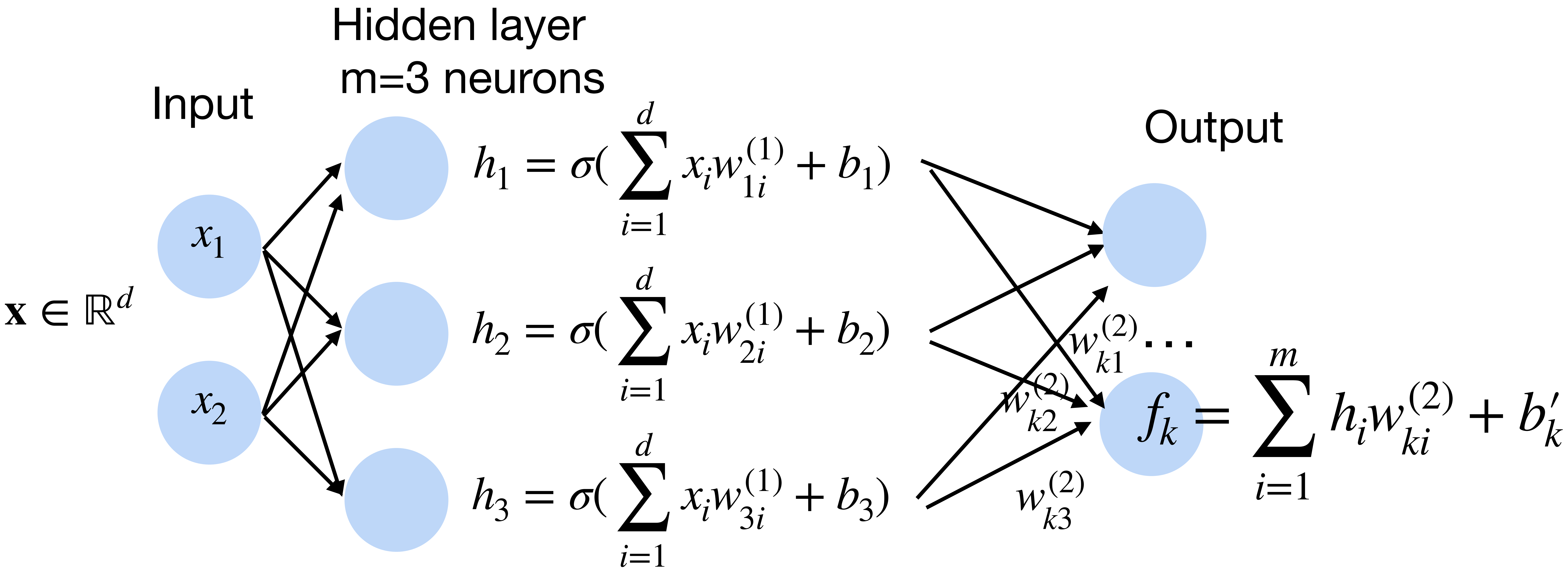
$$f_1 = \sum_{i=1}^{m} h_i w_{1i}^{(2)} + b_1'$$

No
activation function
applied in output
layer

Neural network for k-way classification

- K outputs units in the final layer

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



Hidden layer
m=3 neurons

Input

Output

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

x_1

x_2

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

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

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

$w_{k1}^{(2)}$ …

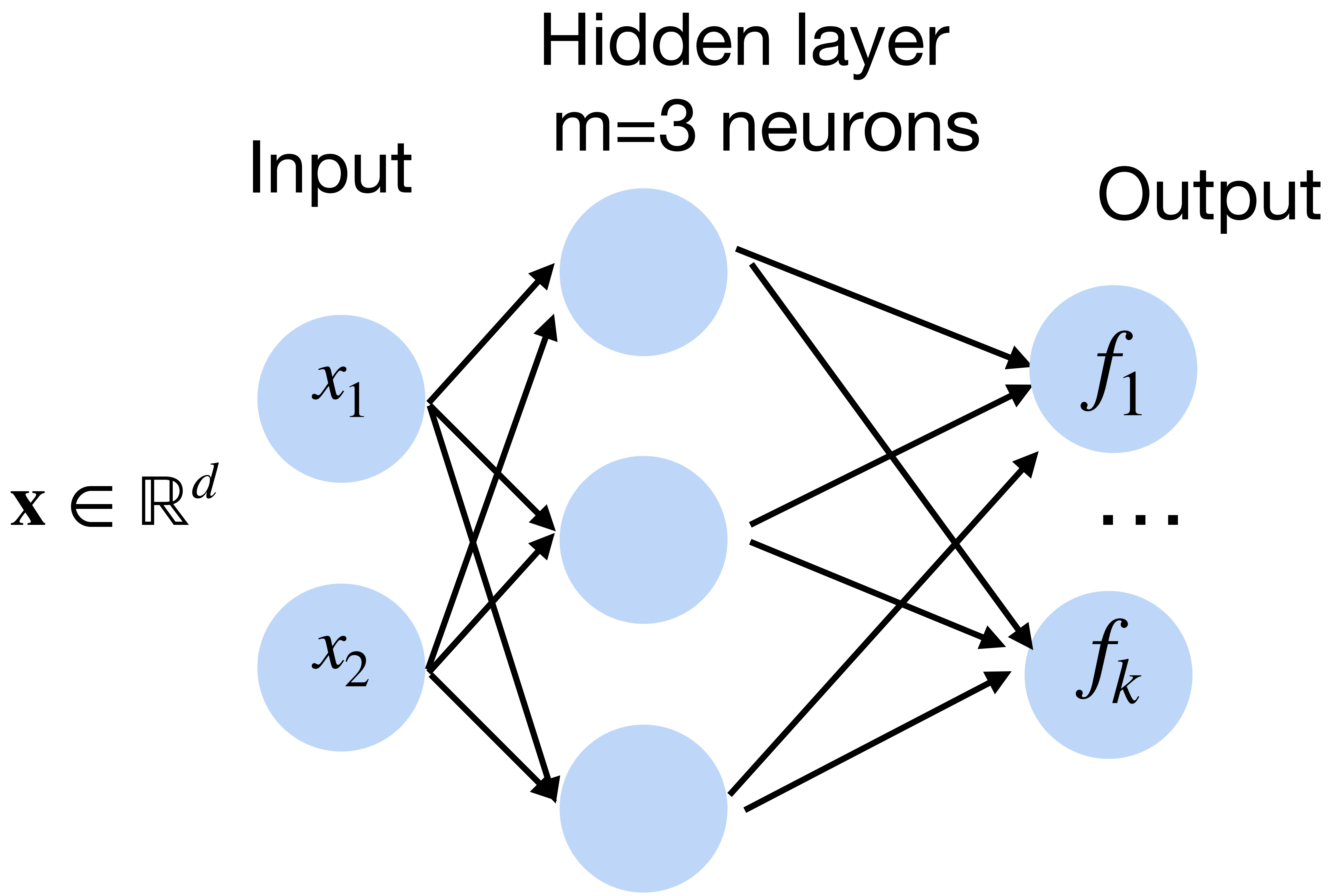
$w_{k2}^{(2)}$

$w_{k3}^{(2)}$

$$f_k = \sum_{i=1}^{m} h_i w_{ki}^{(2)} + b'_k$$

Softmax

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

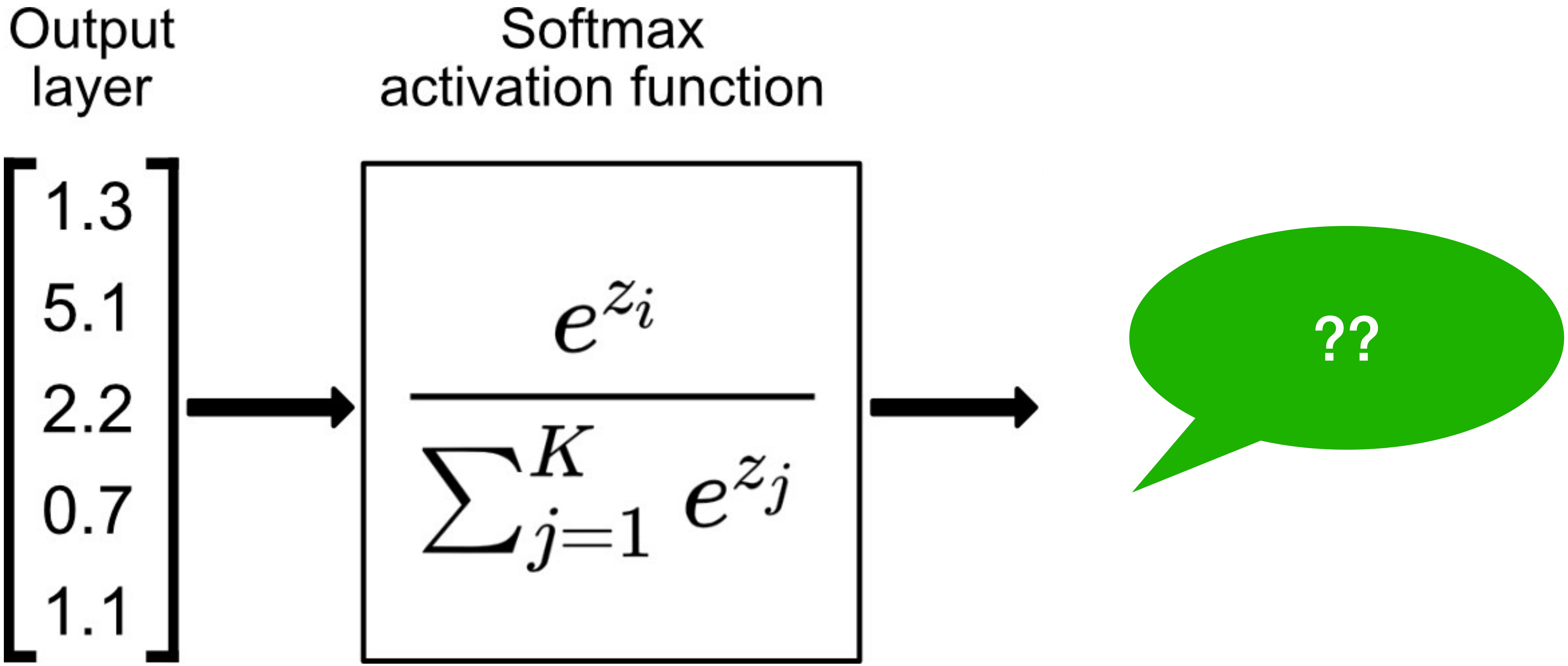


$$p(y \mid \mathbf{x}) = \text{softmax}(f)$$

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

Softmax

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



Output layer

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

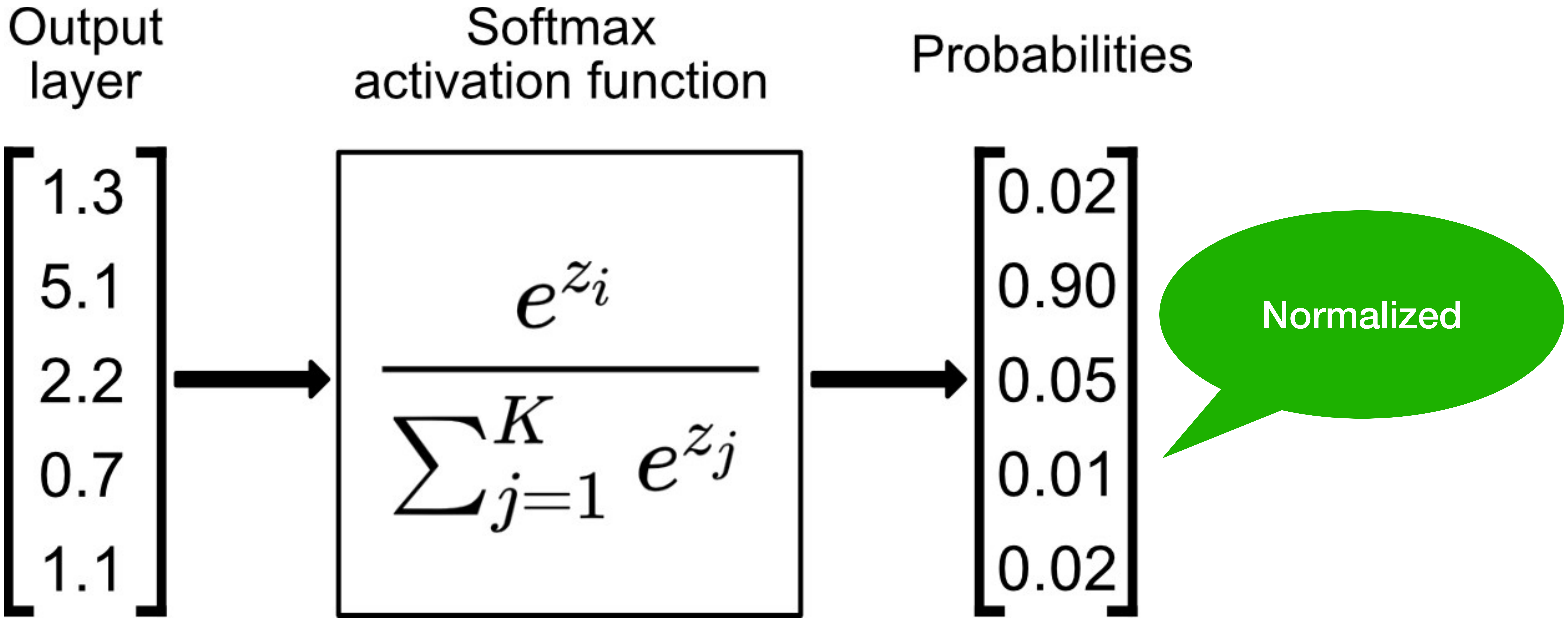
Softmax activation function

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

??

Softmax

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



Output layer

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

Softmax activation function

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

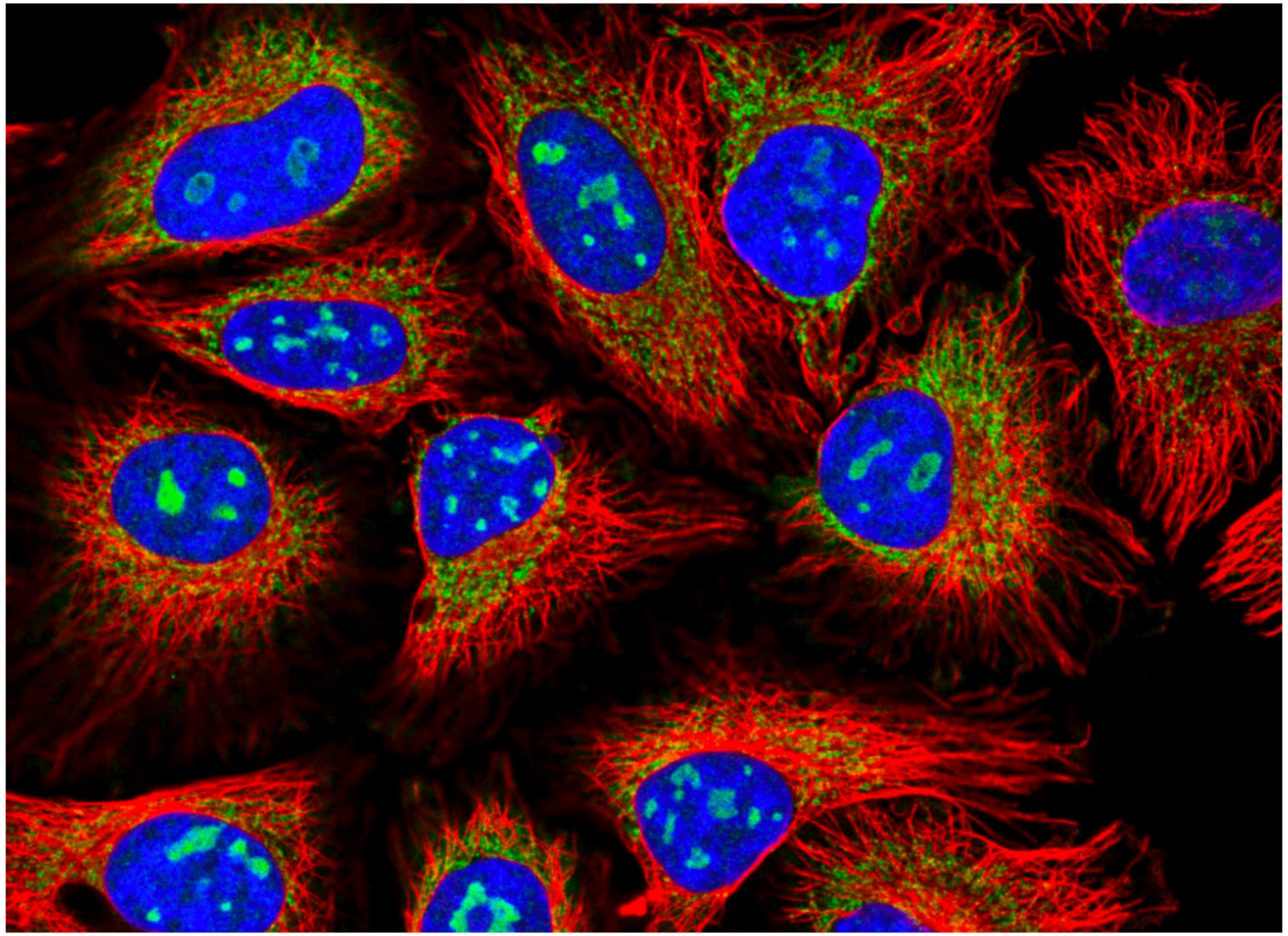
Probabilities

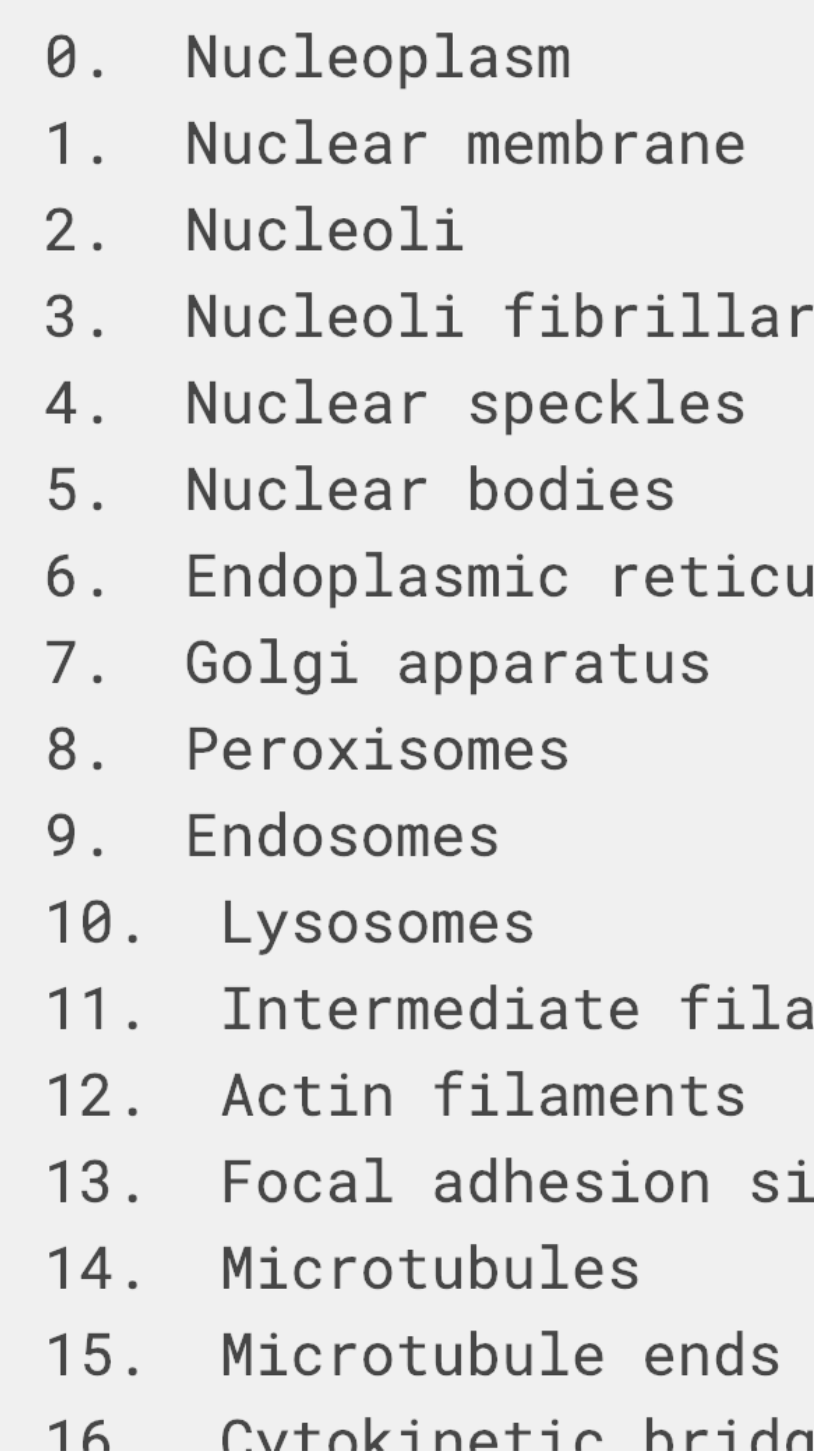
$$\begin{bmatrix} 0.02 \\ 0.90 \\ 0.05 \\ 0.01 \\ 0.02 \end{bmatrix}$$

Normalized

Classification Tasks at Kaggle

Classify human protein microscope images into 28 categories



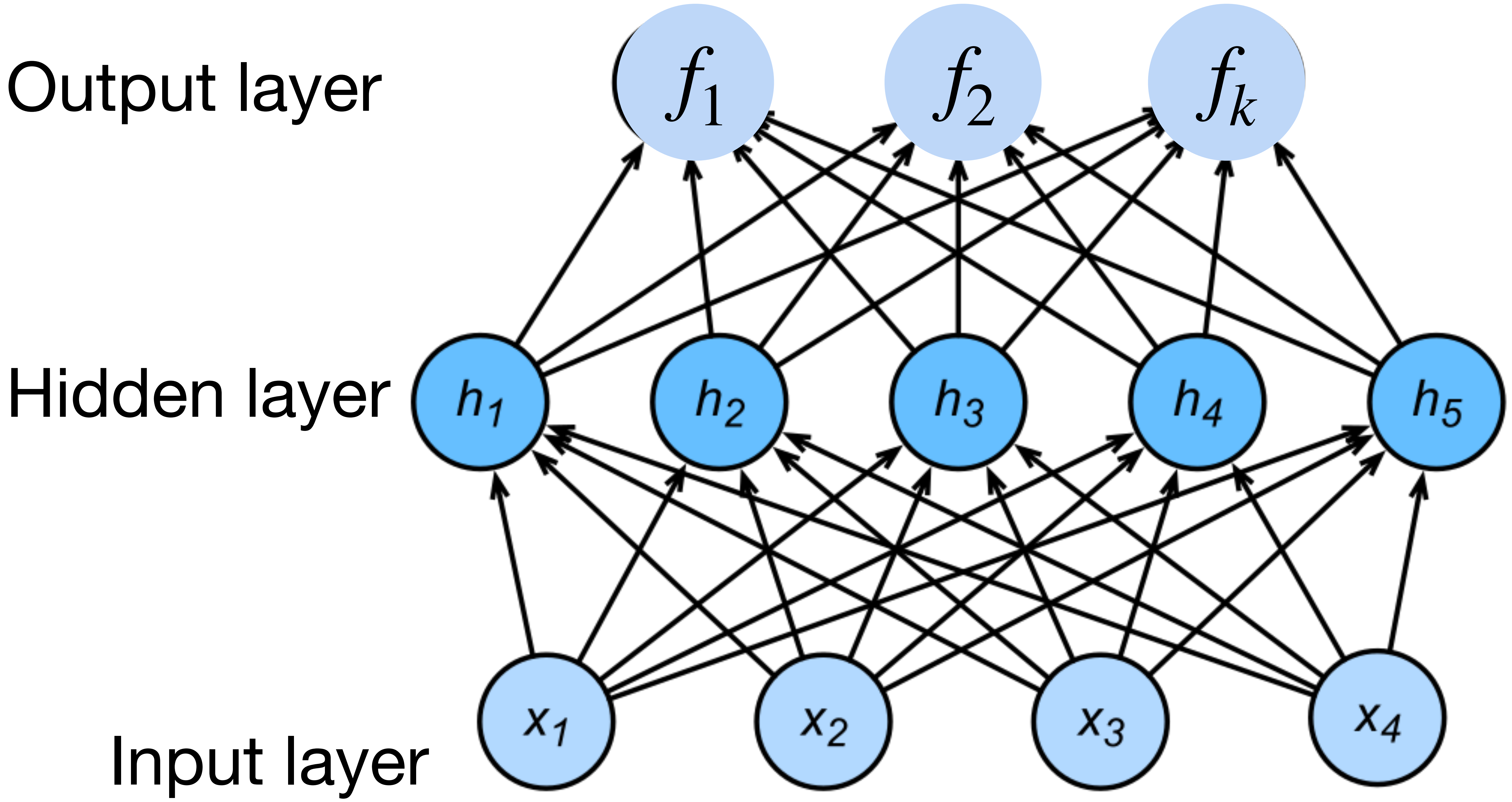


```
0.  Nucleoplasm
1.  Nuclear membrane
2.  Nucleoli
3.  Nucleoli fibrillar
4.  Nuclear speckles
5.  Nuclear bodies
6.  Endoplasmic reticu
7.  Golgi apparatus
8.  Peroxisomes
9.  Endosomes
10.  Lysosomes
11.  Intermediate fila
12.  Actin filaments
13.  Focal adhesion si
14.  Microtubules
15.  Microtubule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

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

More complicated neural networks

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



More complicated neural networks

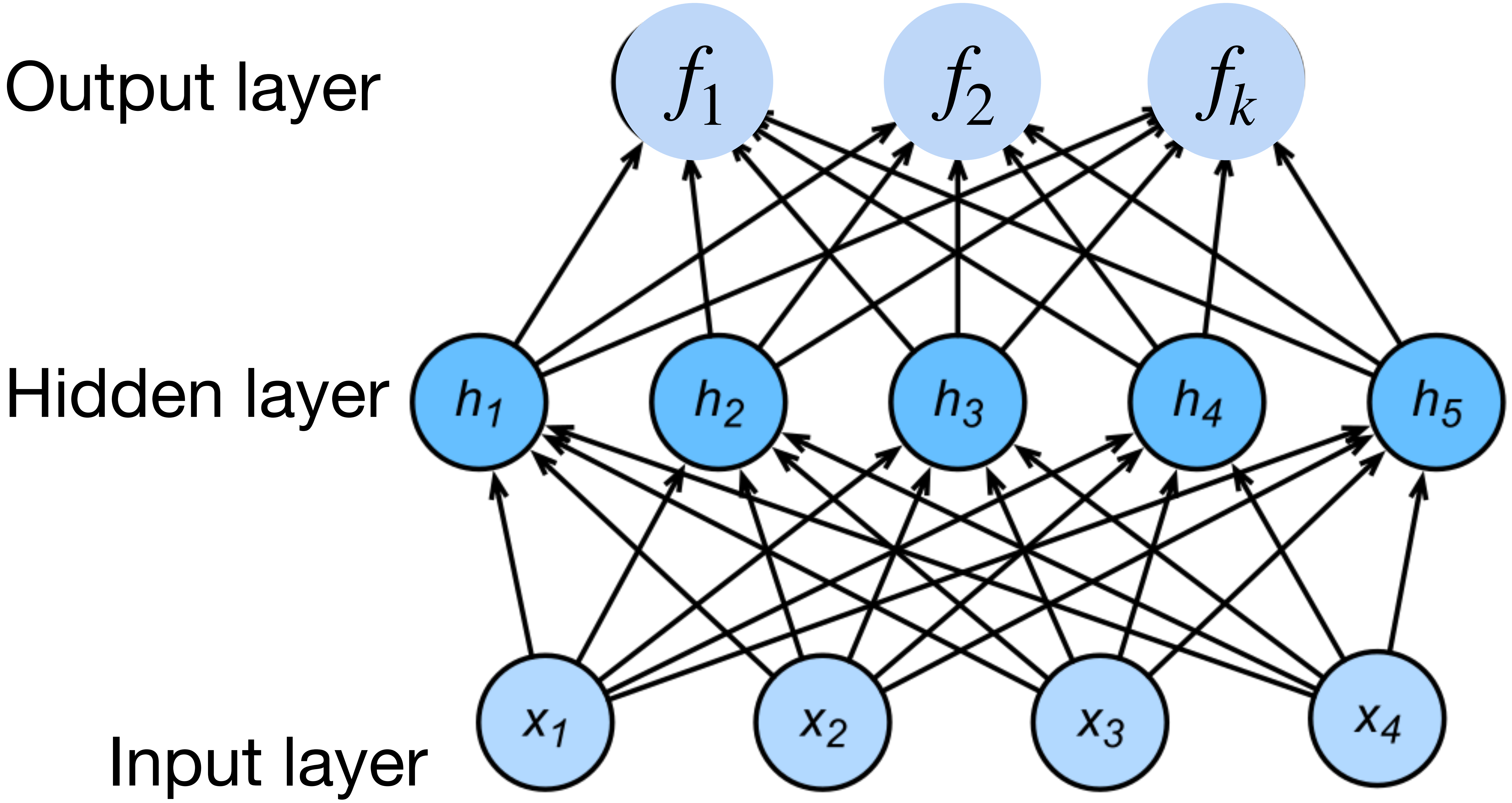
- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$

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

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

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

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



More complicated neural networks: multiple hidden layers

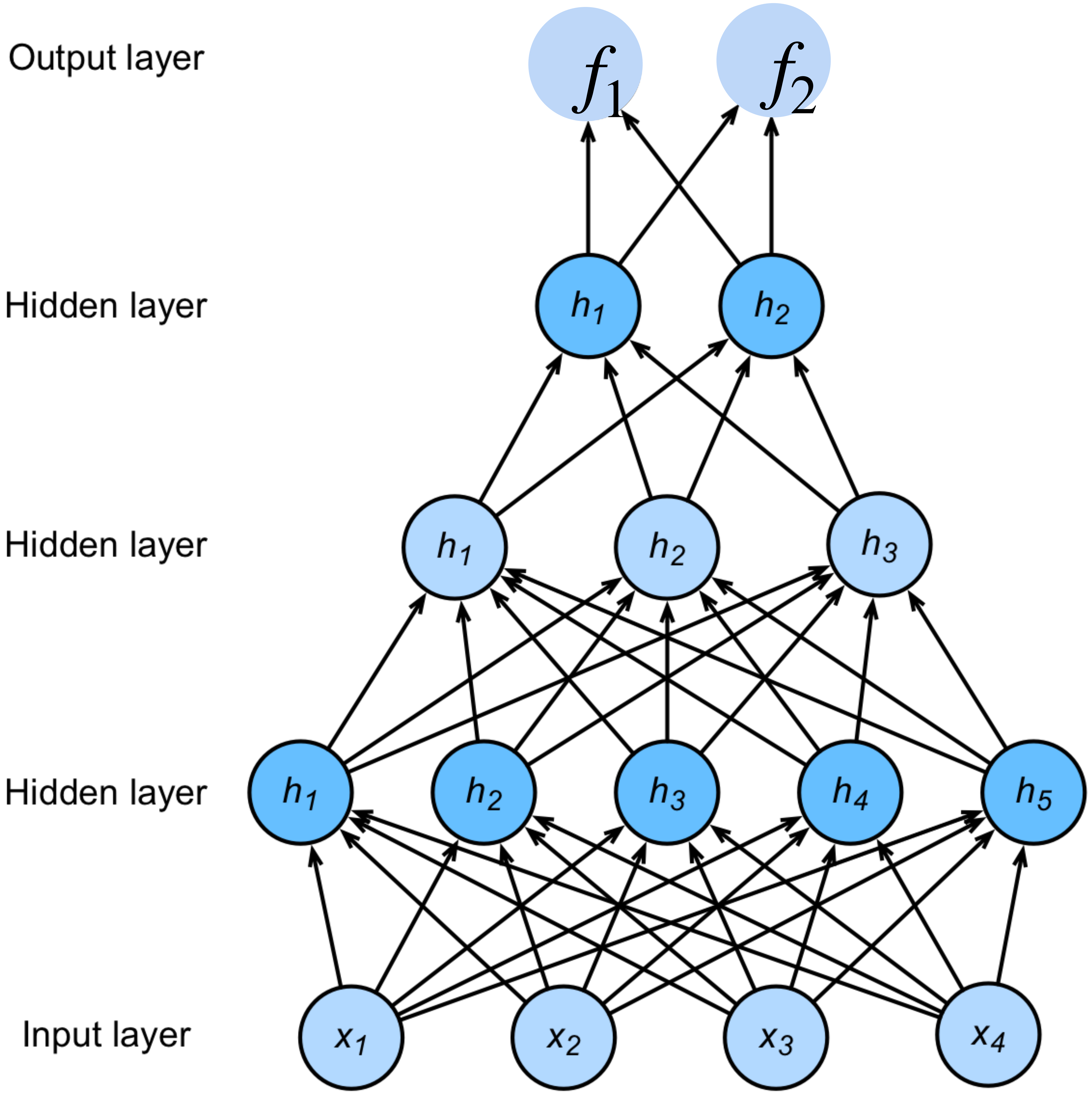
$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

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

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

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

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



Quiz Break

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

A Sigmoid function

B Rectified Linear Unit (ReLU)

C Softmax function

D Max function

Quiz Break

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

A Sigmoid function

B Rectified Linear Unit (ReLU)

[✓ C Softmax function]

D Max function

Quiz Break

Suppose you are given a 3-layer multilayer perceptron (2 hidden layers h1 and h2 and 1 output layer). All activation functions are sigmoids, and the output layer uses a softmax function. Suppose h1 has 1024 units and h2 has 512 units. Given a dataset with 2 input features and 3 unique class labels, how many learnable parameters does the perceptron have in total?

Quiz Break

Suppose you are given a 3-layer multilayer perceptron (2 hidden layers h1 and h2 and 1 output layer). All activation functions are sigmoids, and the output layer uses a softmax function. Suppose h1 has 1024 units and h2 has 512 units. Given a dataset with 2 input features and 3 unique class labels, how many learnable parameters does the perceptron have in total?

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

Quiz Break

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

a)Yes

b)No

Quiz Break

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

a)Yes

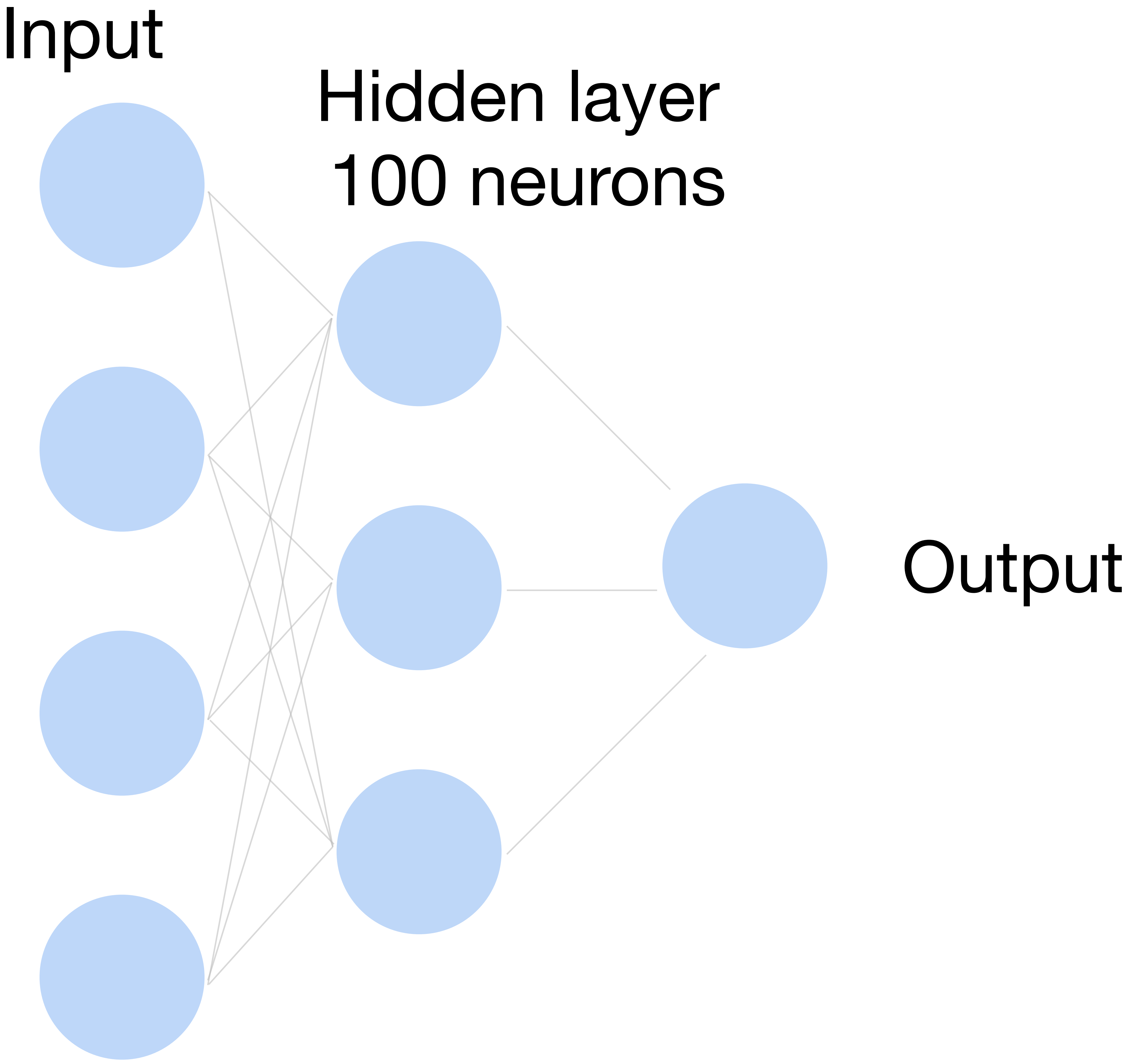
[✓ b)No]

Solution:

A combination of linear Perceptrons is still a linear function.

How to train a neural network?

Classify cats vs. dogs



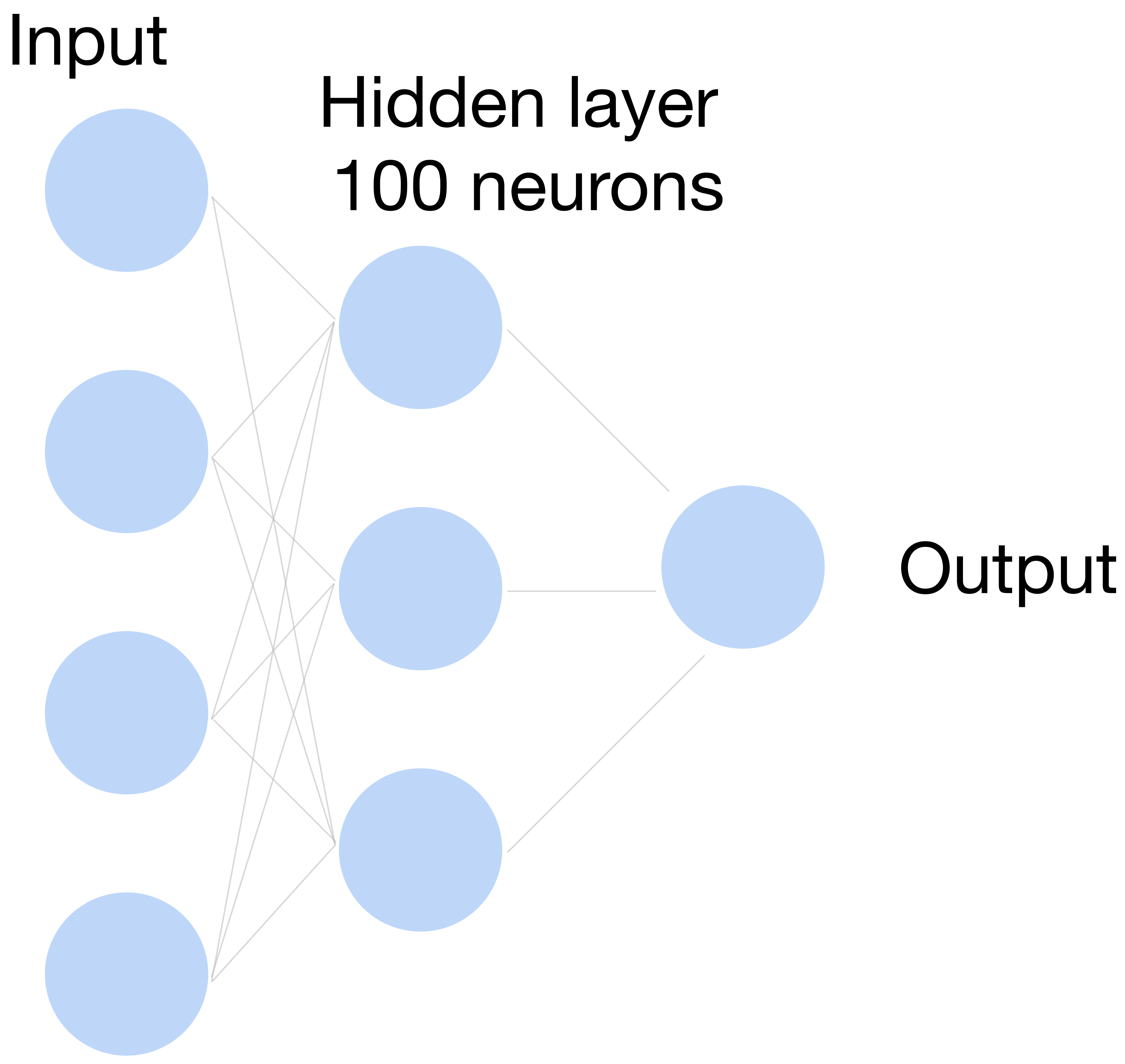
How to train a neural network?

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

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

(This is a function of all weights W)

y Ground truth label for example **x**



Learning by matching the output to the label

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

How to train a neural network?

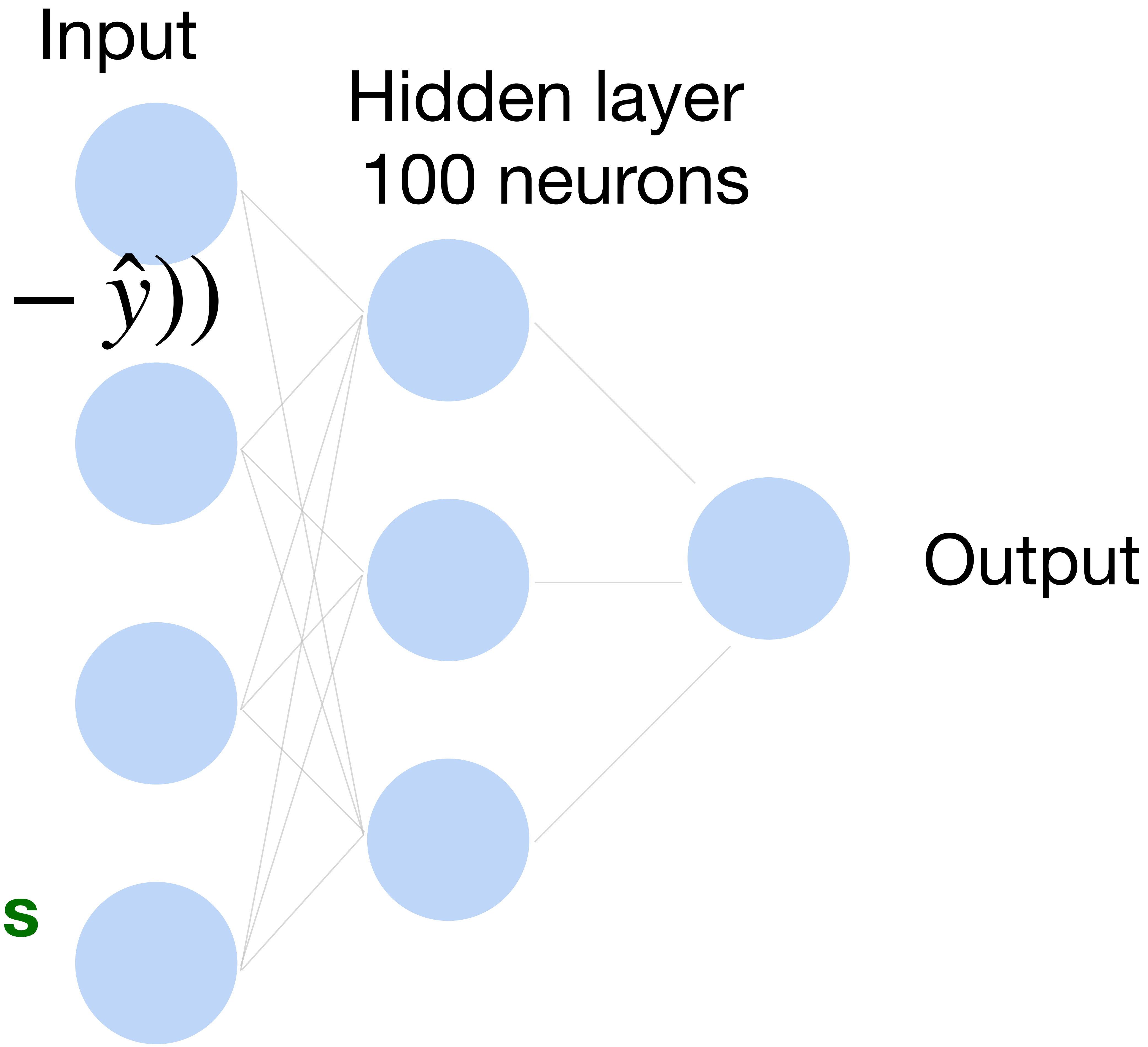
Loss function: $$\frac{1}{|D|}\sum_{i} \ell(\mathbf{x}_i, y_i)$$

Per-sample loss:

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

Negative log likelihood

Also known as binary cross-entropy loss

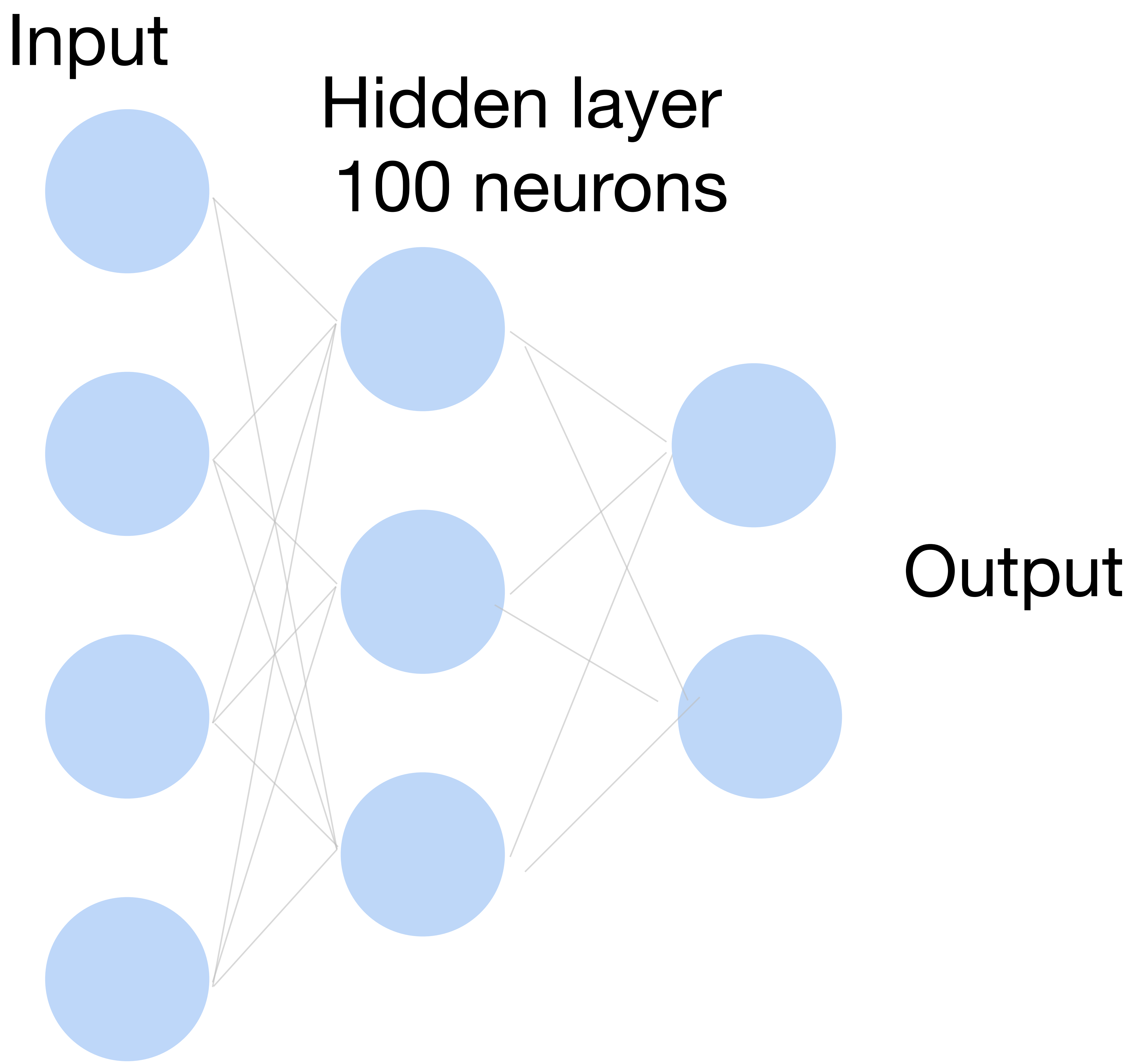


How to train a neural network?

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

Per-sample loss:

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

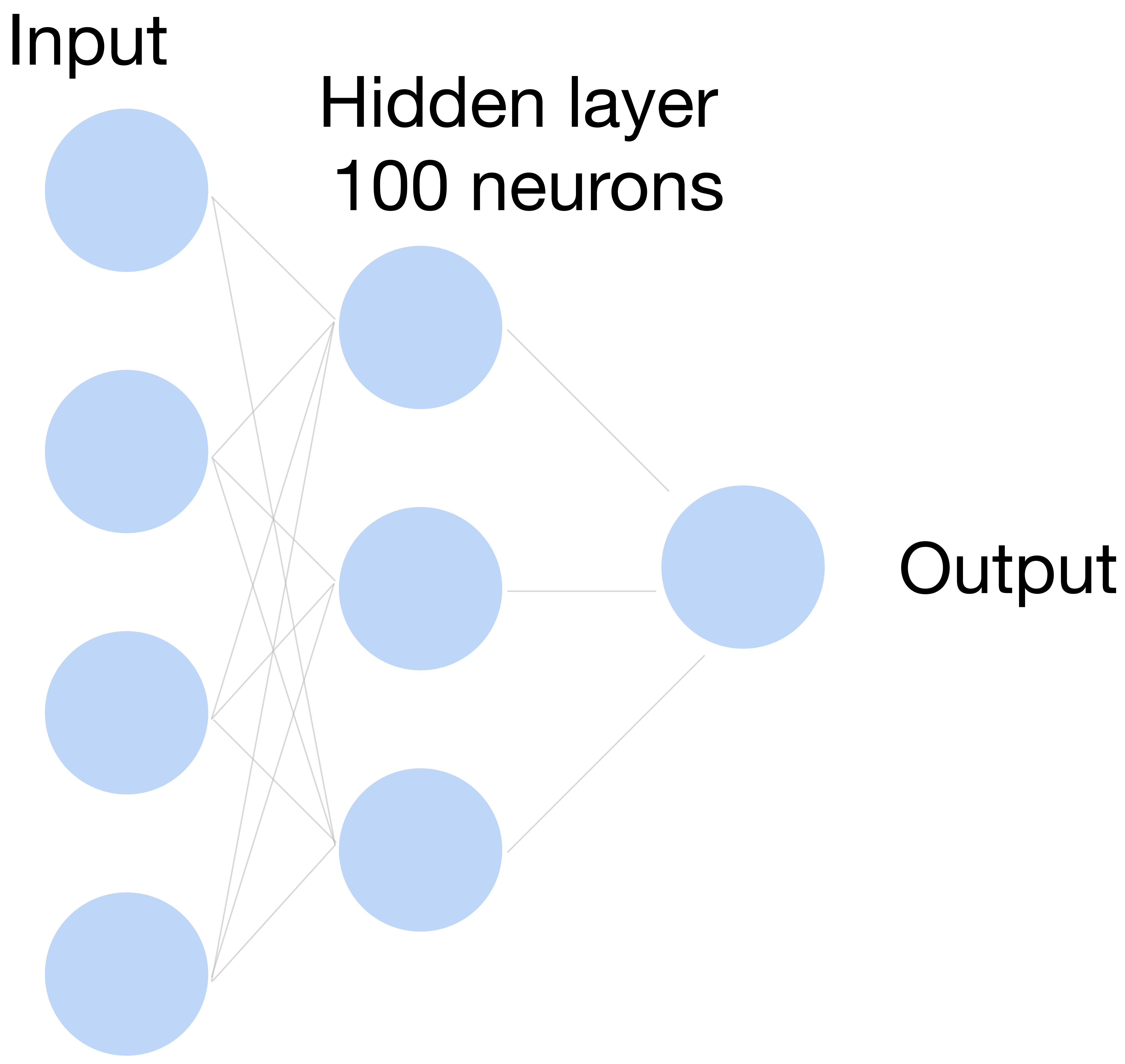


How to train a neural network?

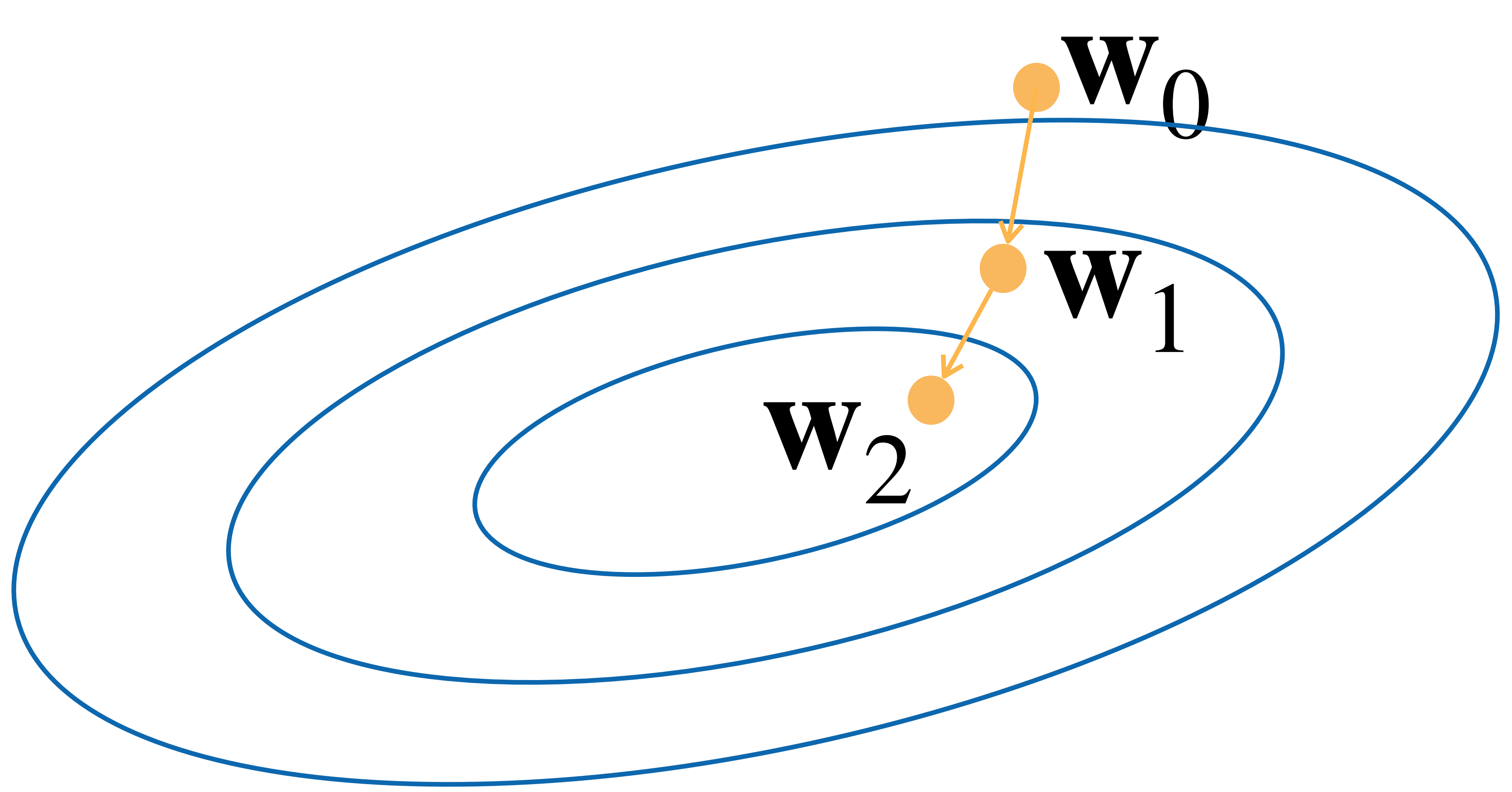
Update the weights W to minimize the loss function

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

Use gradient descent!



Gradient Descent



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

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

D can be very large. Expensive

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

- Repeat until converges

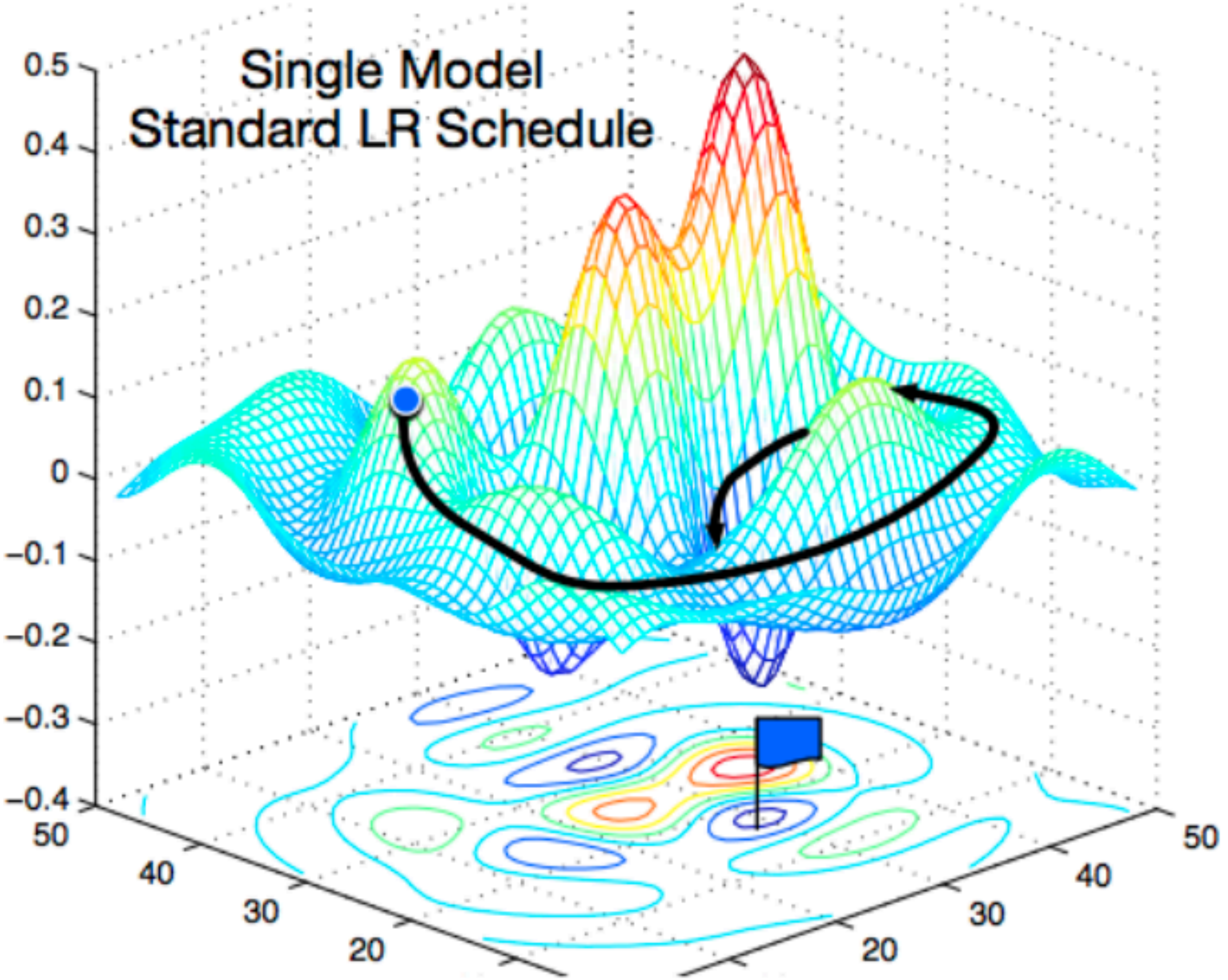
Minibatch Stochastic Gradient Descent

- 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:

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

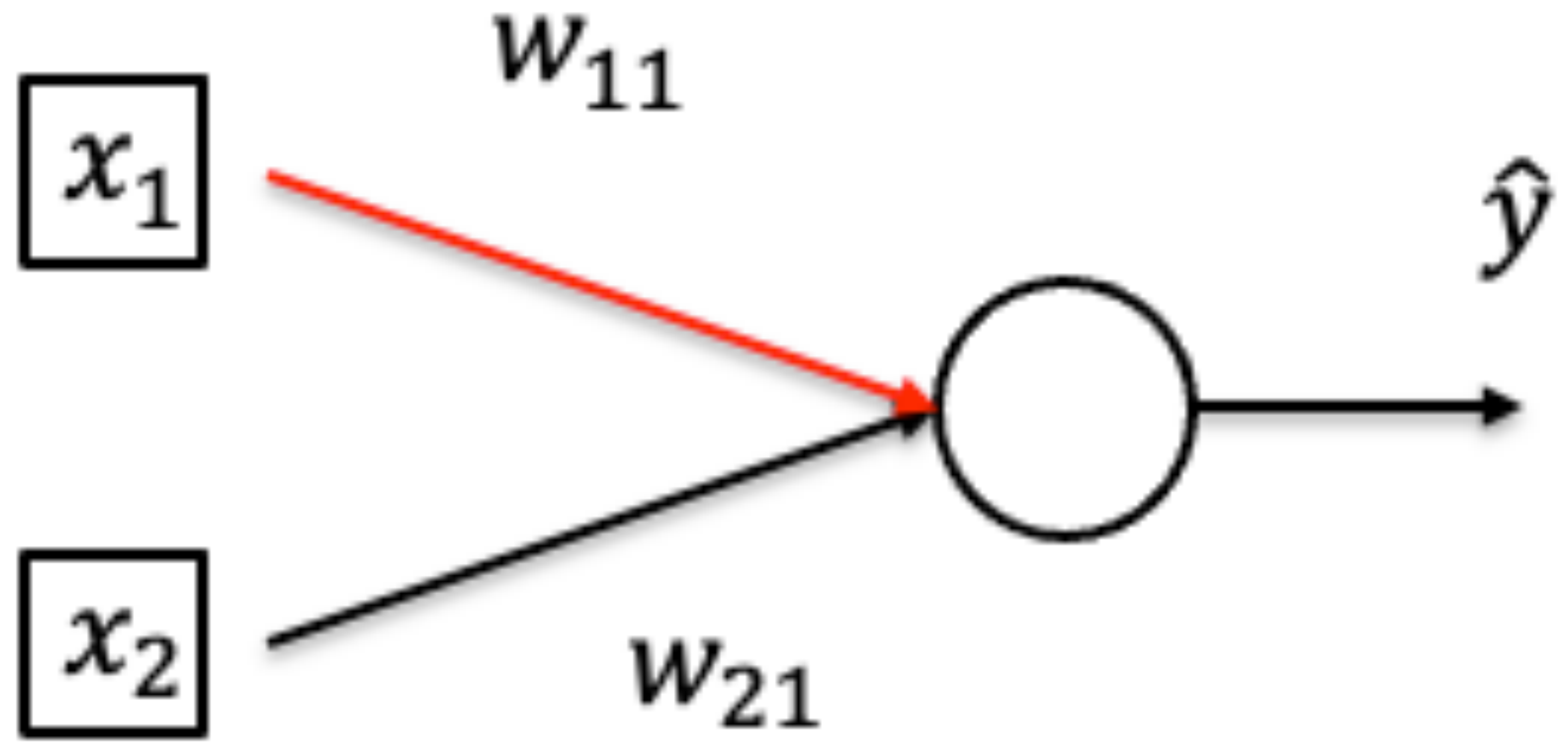
- Repeat until converges

Non-convex Optimization



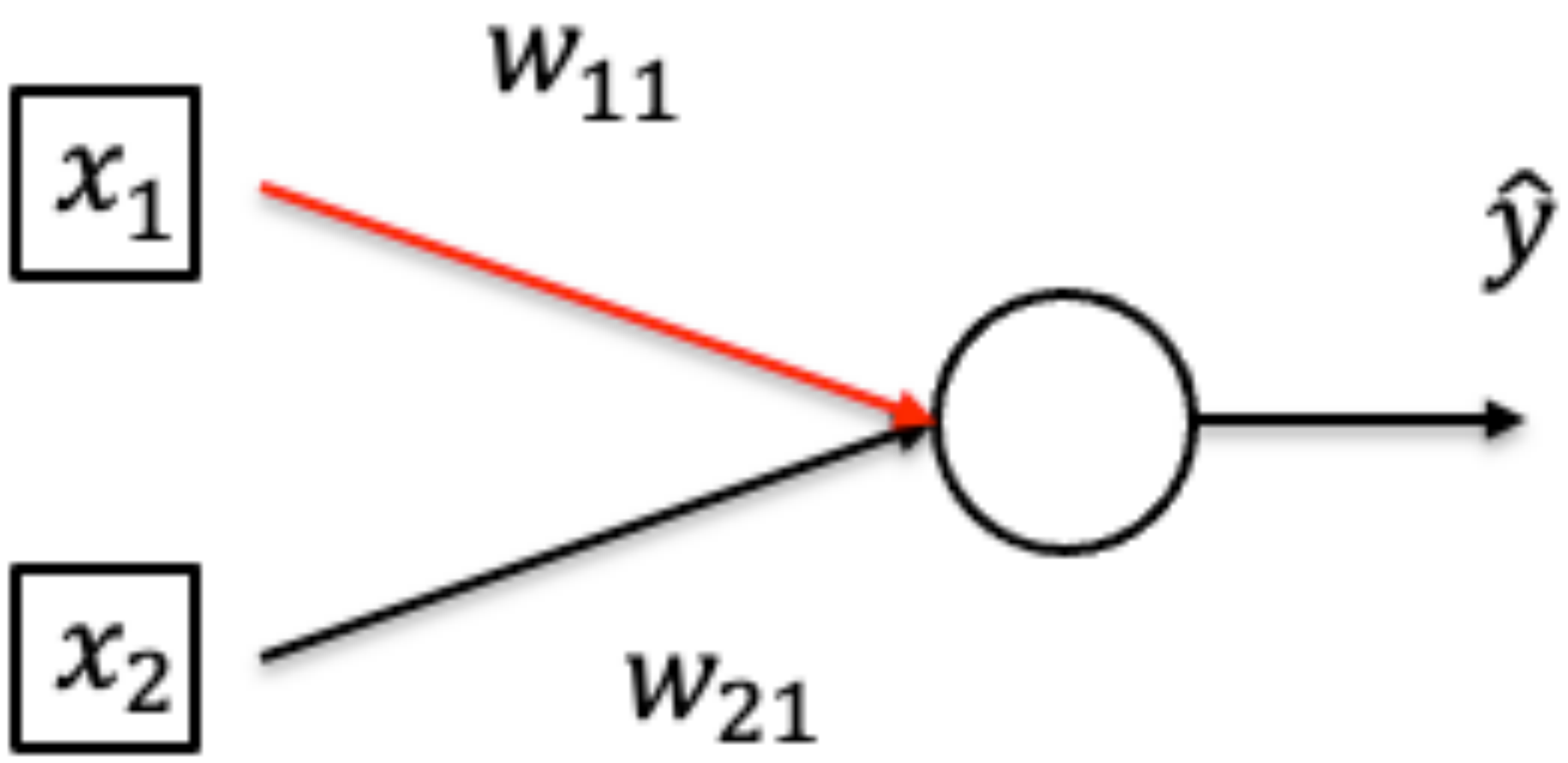
[Gao and Li et al., 2018]

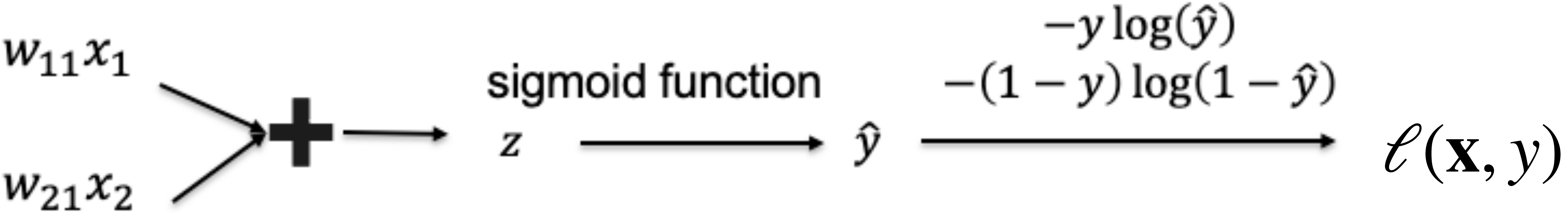
Calculate Gradient (on one data point)



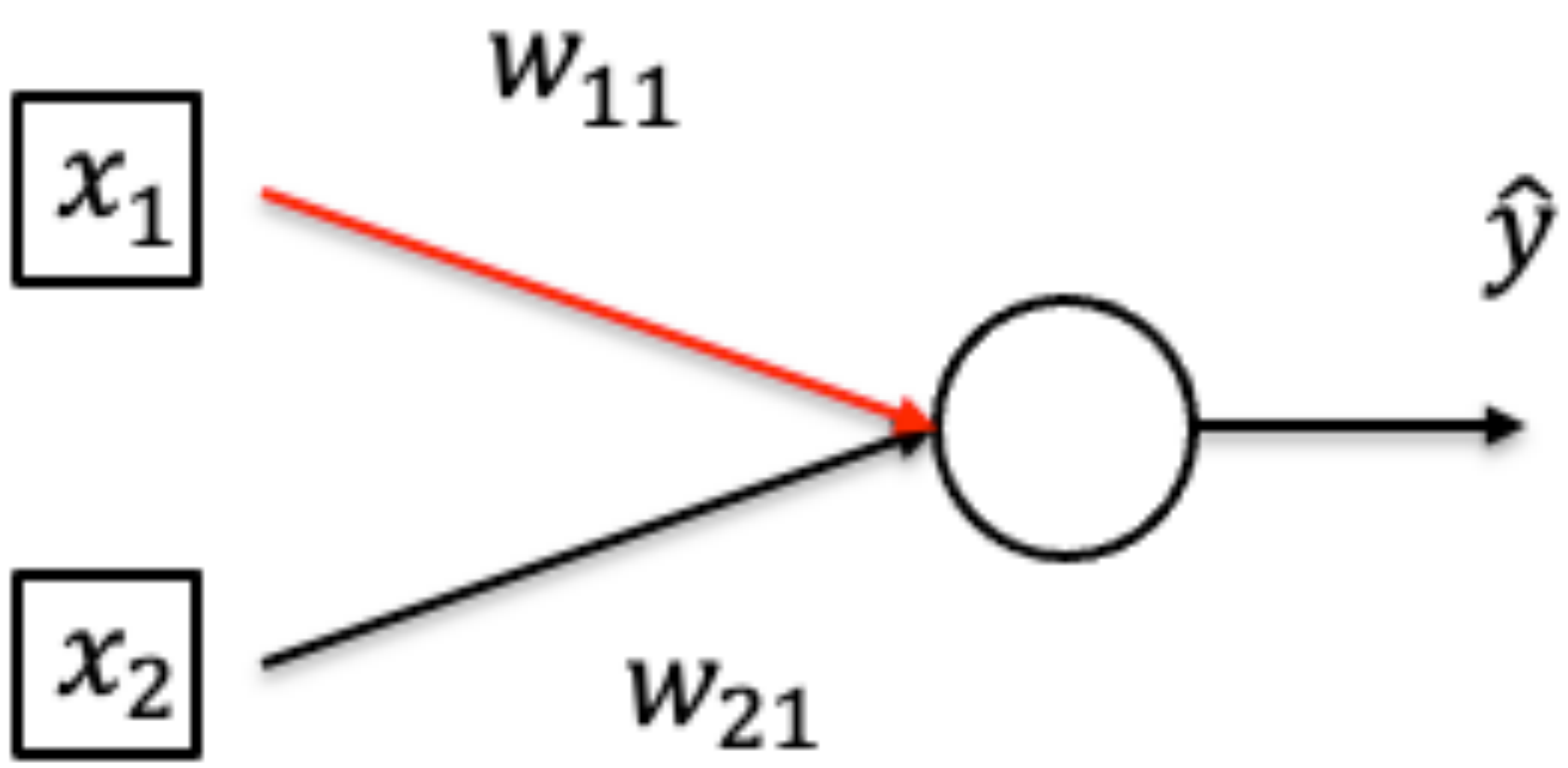
- Want to compute $\dfrac{\partial \ell(\mathbf{x}, y)}{\partial w_{11}}$

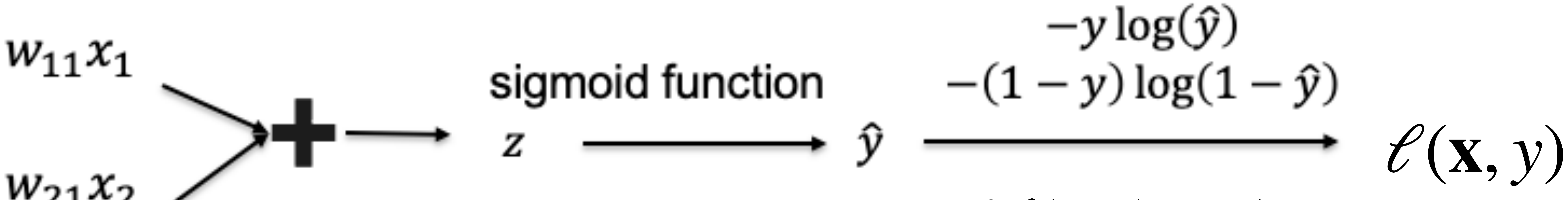
Calculate Gradient (on one data point)





Calculate Gradient (on one data point)

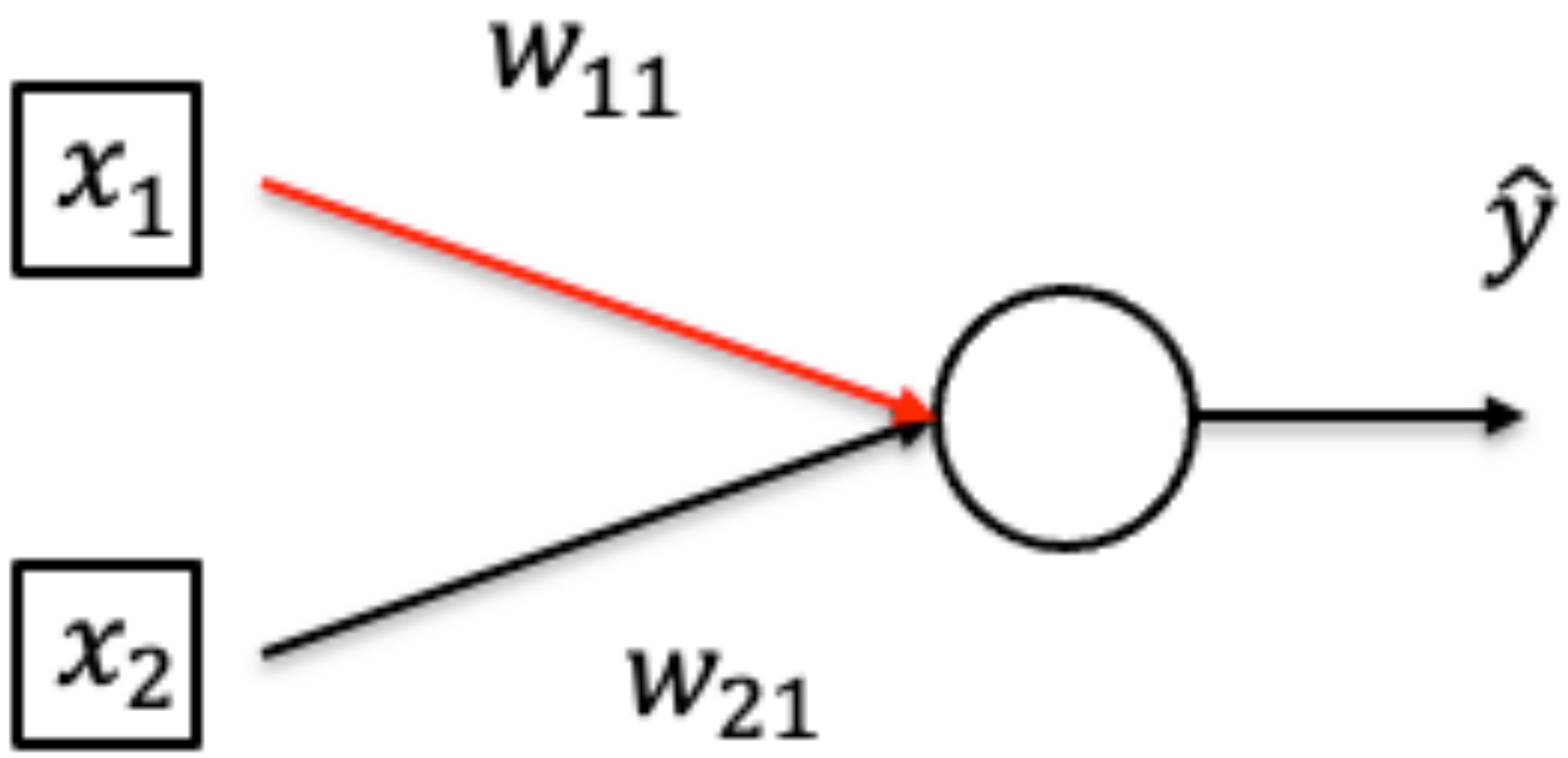


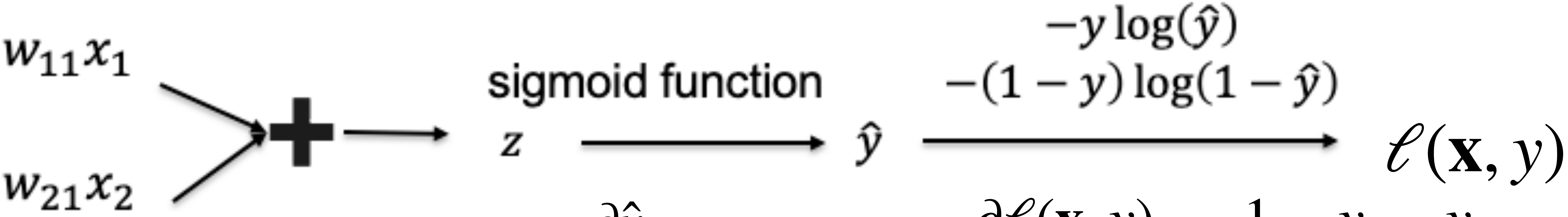


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) \qquad \frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

- 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)

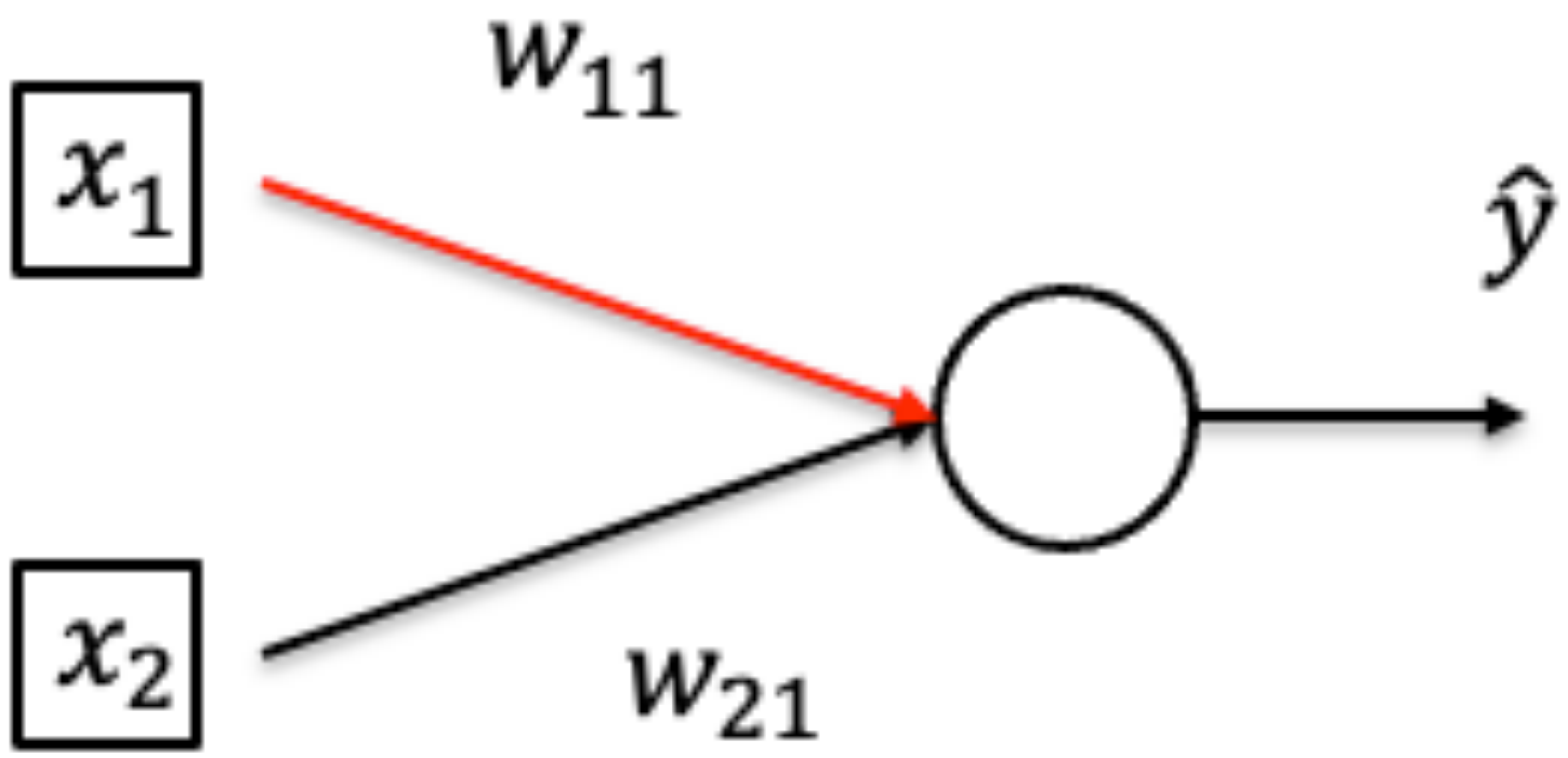


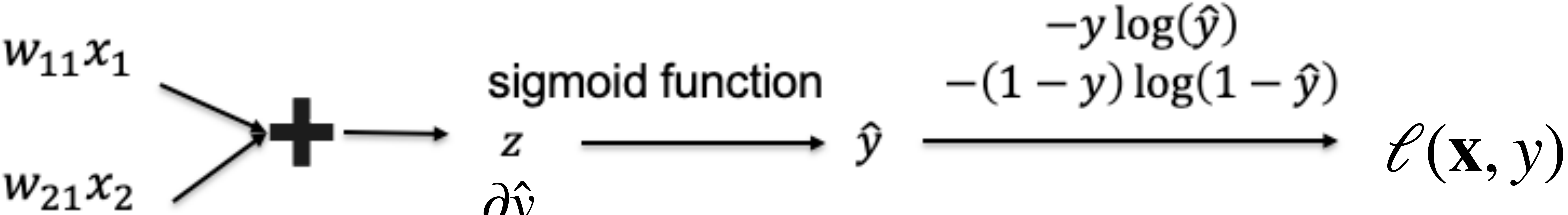


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) \qquad \frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

- By chain rule: $\dfrac{\partial l}{\partial w_{11}} = \dfrac{\partial l}{\partial \hat{y}} \dfrac{\partial \hat{y}}{\partial z} x_1$

Calculate Gradient (on one data point)

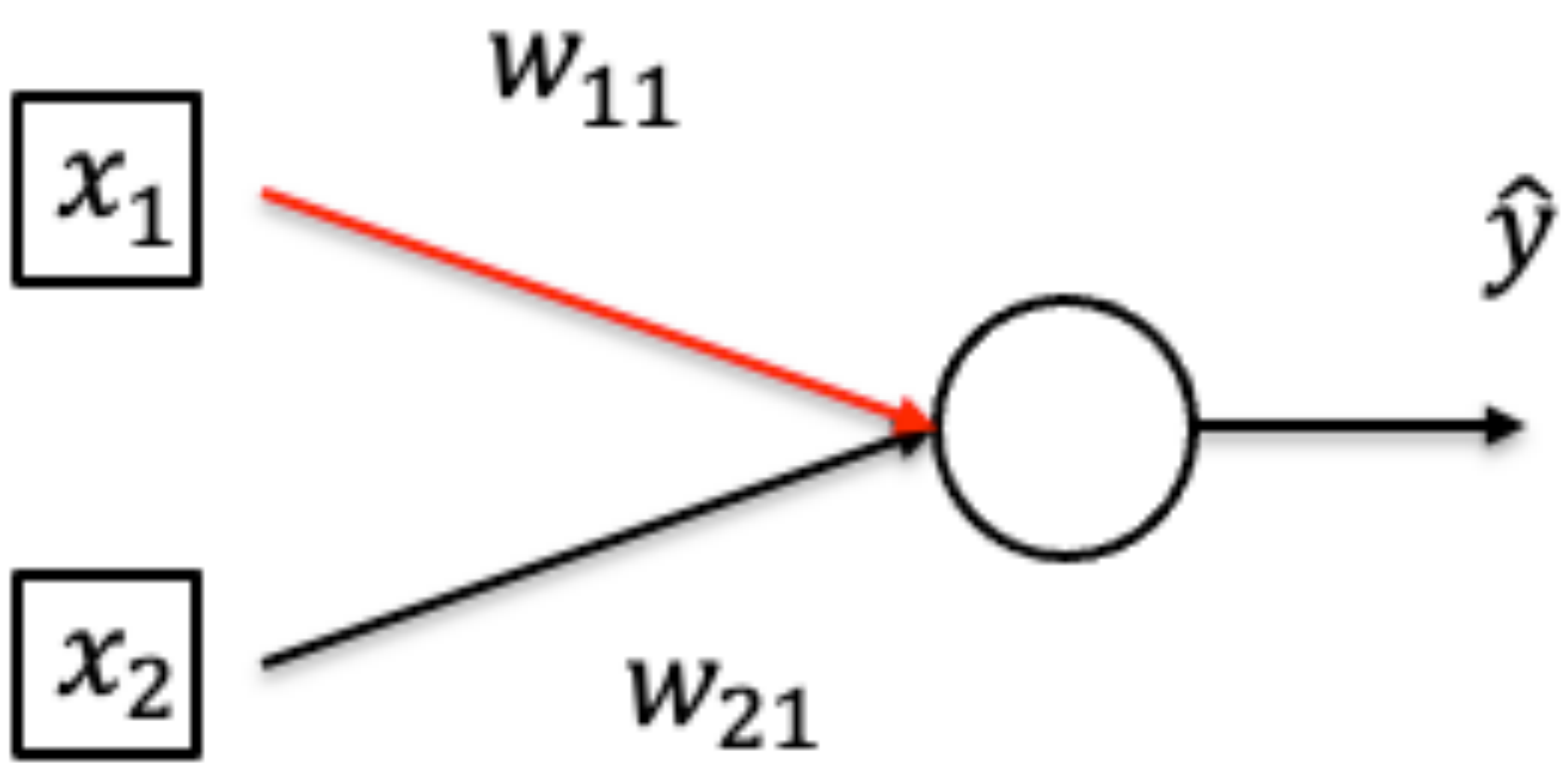


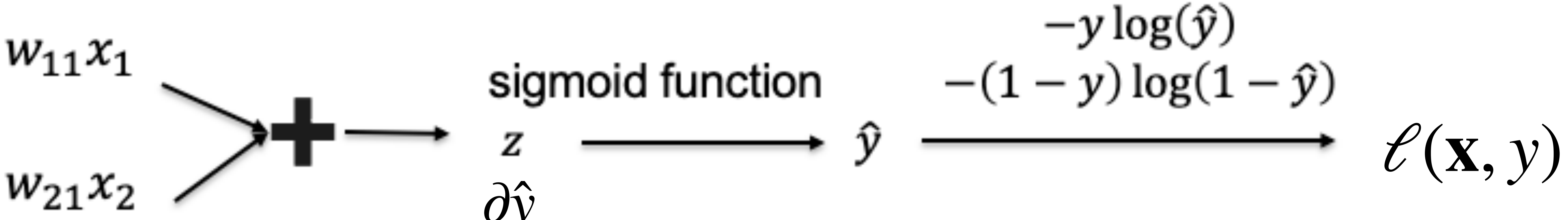


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

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

Calculate Gradient (on one data point)

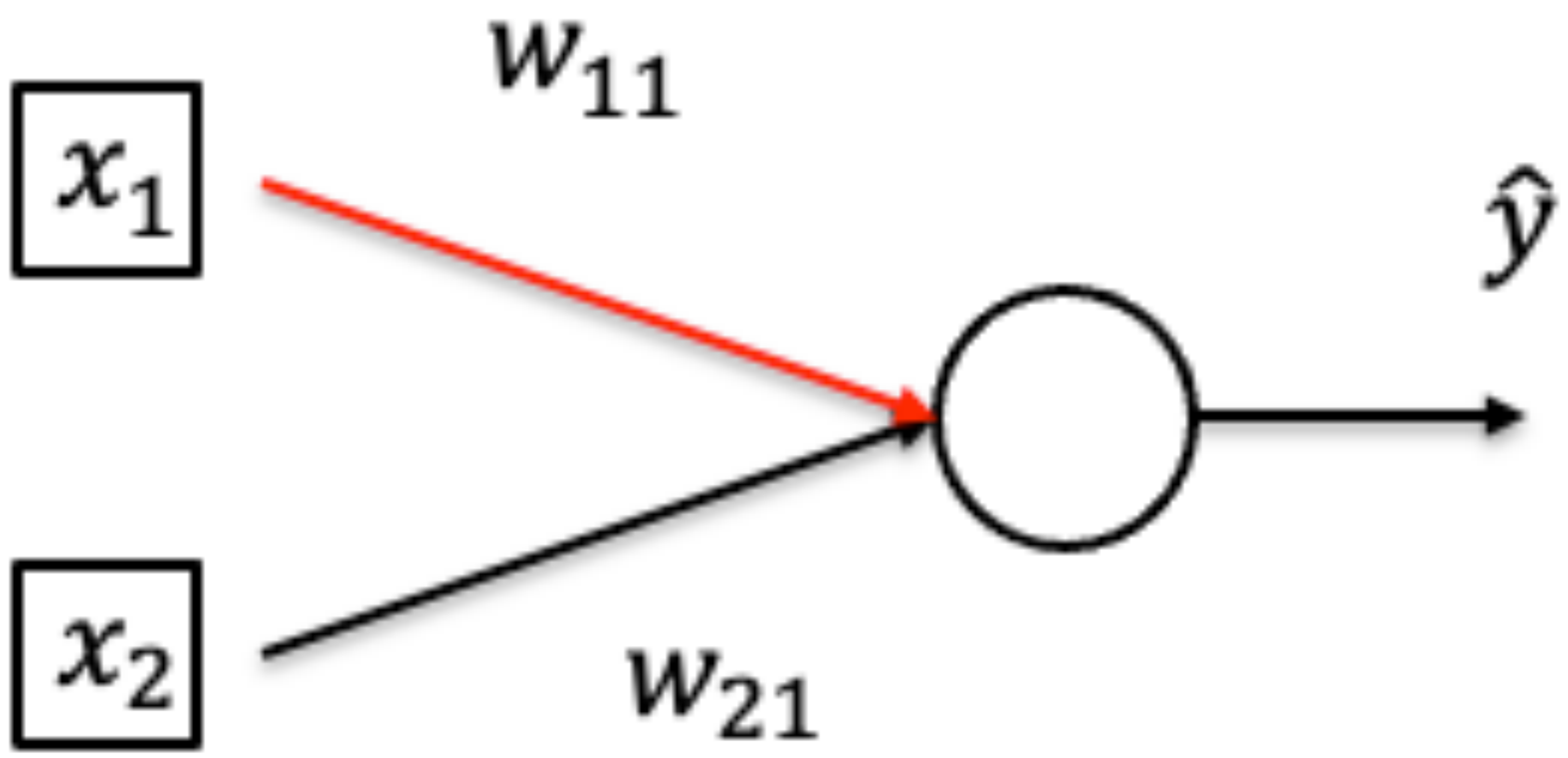


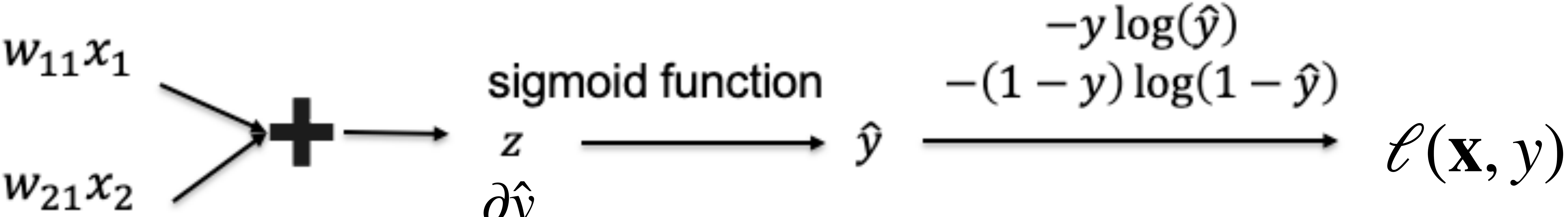


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

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

Calculate Gradient (on one data point)

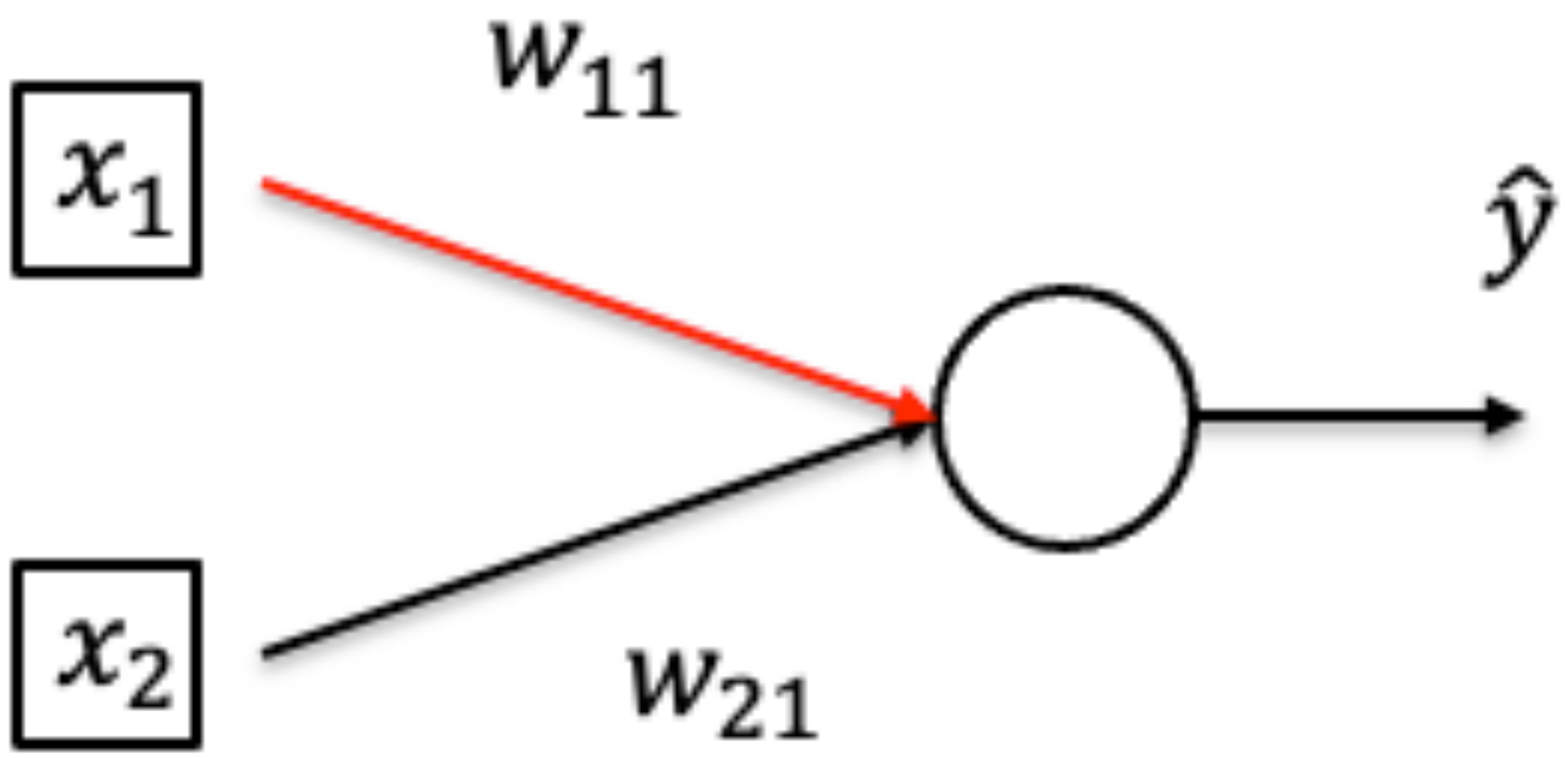


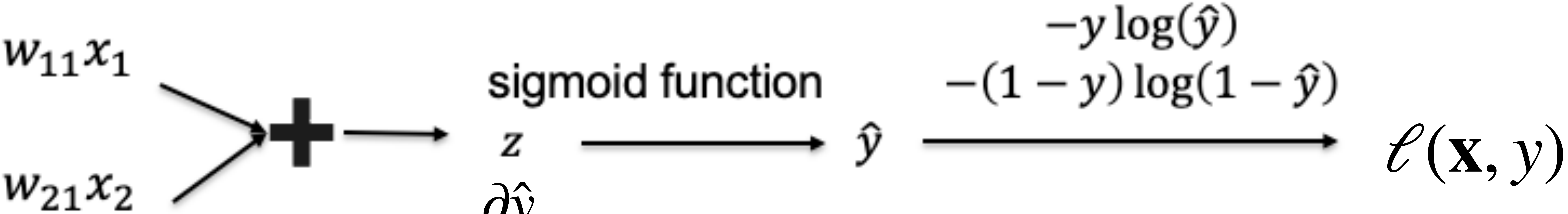


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

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

Calculate Gradient (on one data point)

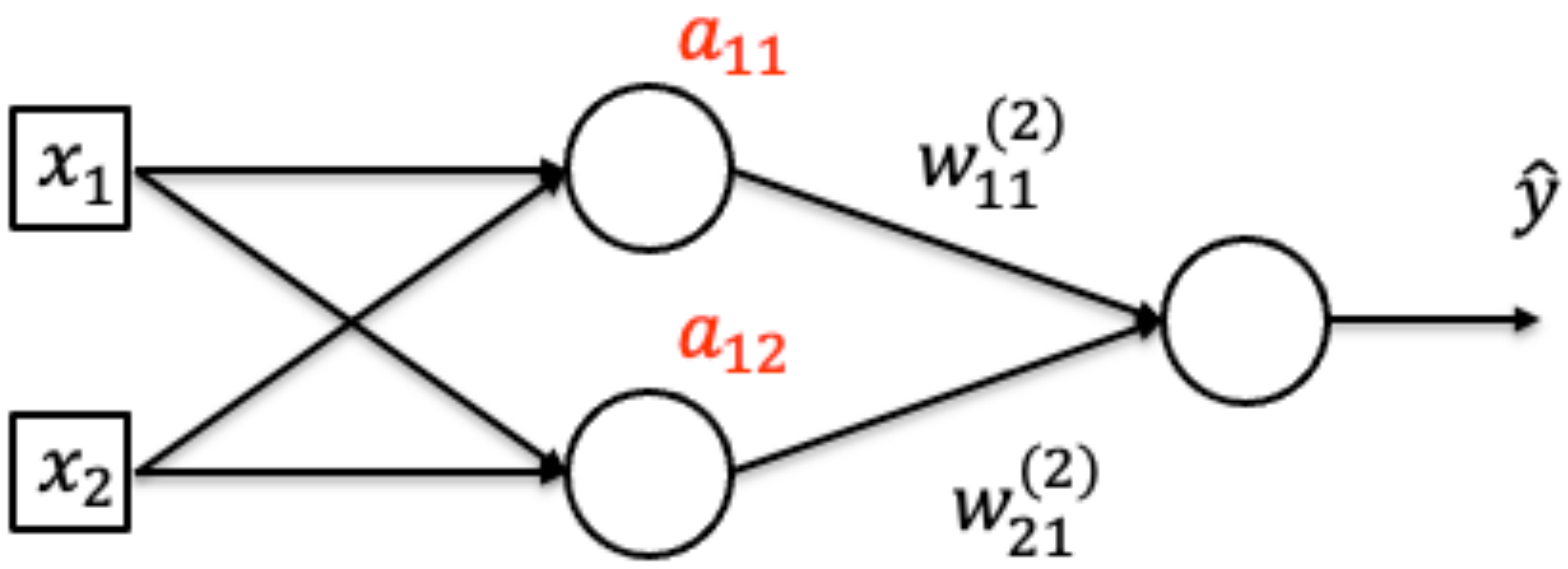




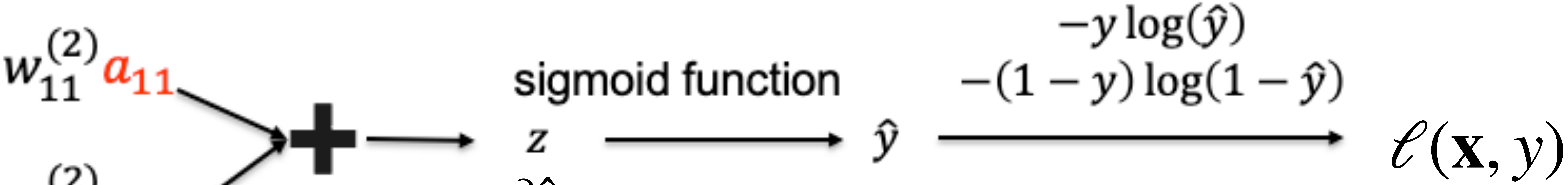
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

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

Calculate Gradient (on one data point)



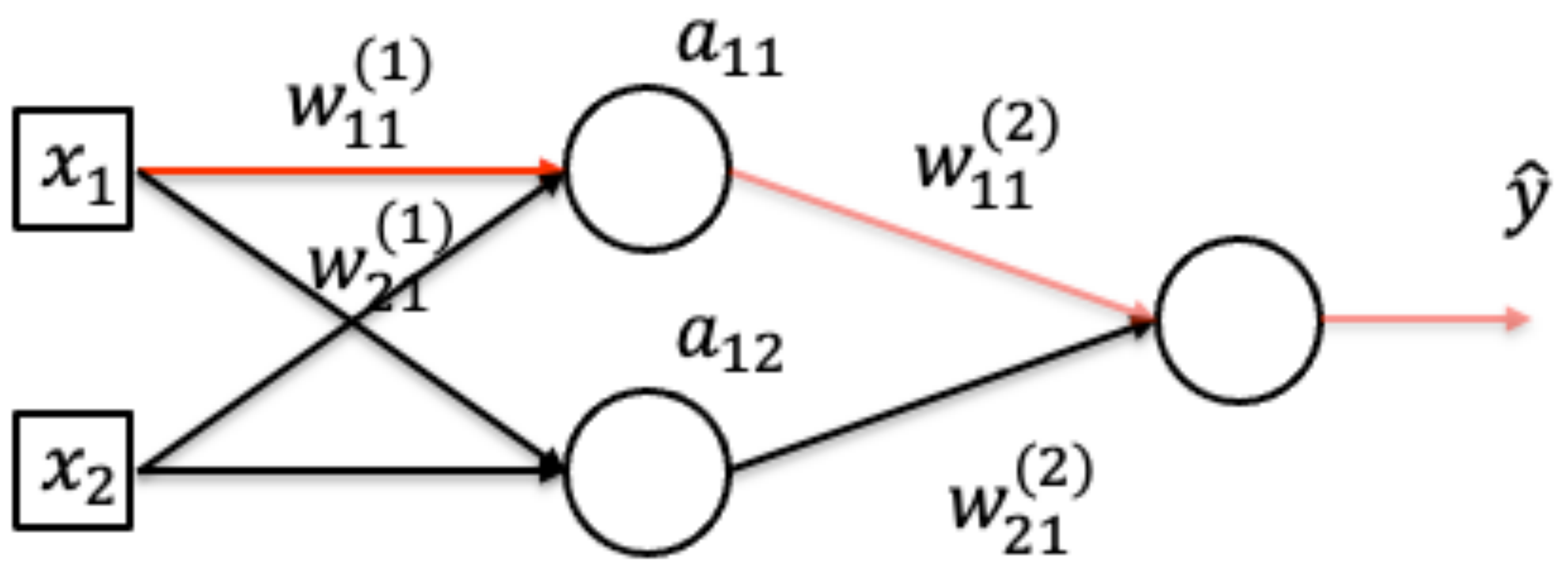
Make it deeper



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

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

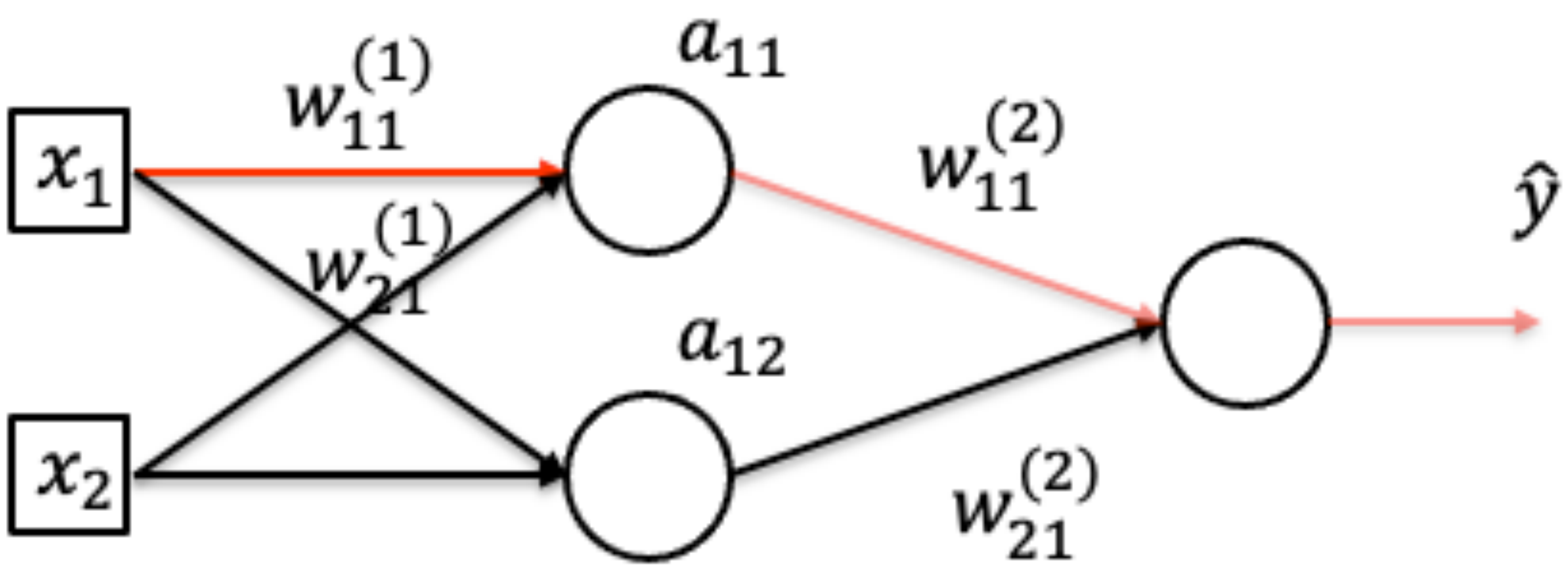
Calculate Gradient (on one data point)



$$\begin{tikzcd}
w_{11}^{(1)}x_1 \arrow[dr] & & & & \\
 & + \arrow[r] & z_{11} \arrow[r, "\sigma(z_{11})", "\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11})"'] & a_{11} \arrow[r, "\frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}"'] & l(\boldsymbol{x}, y) \\
w_{21}^{(1)}x_2 \arrow[ur] & & & &
\end{tikzcd}$$

- 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)

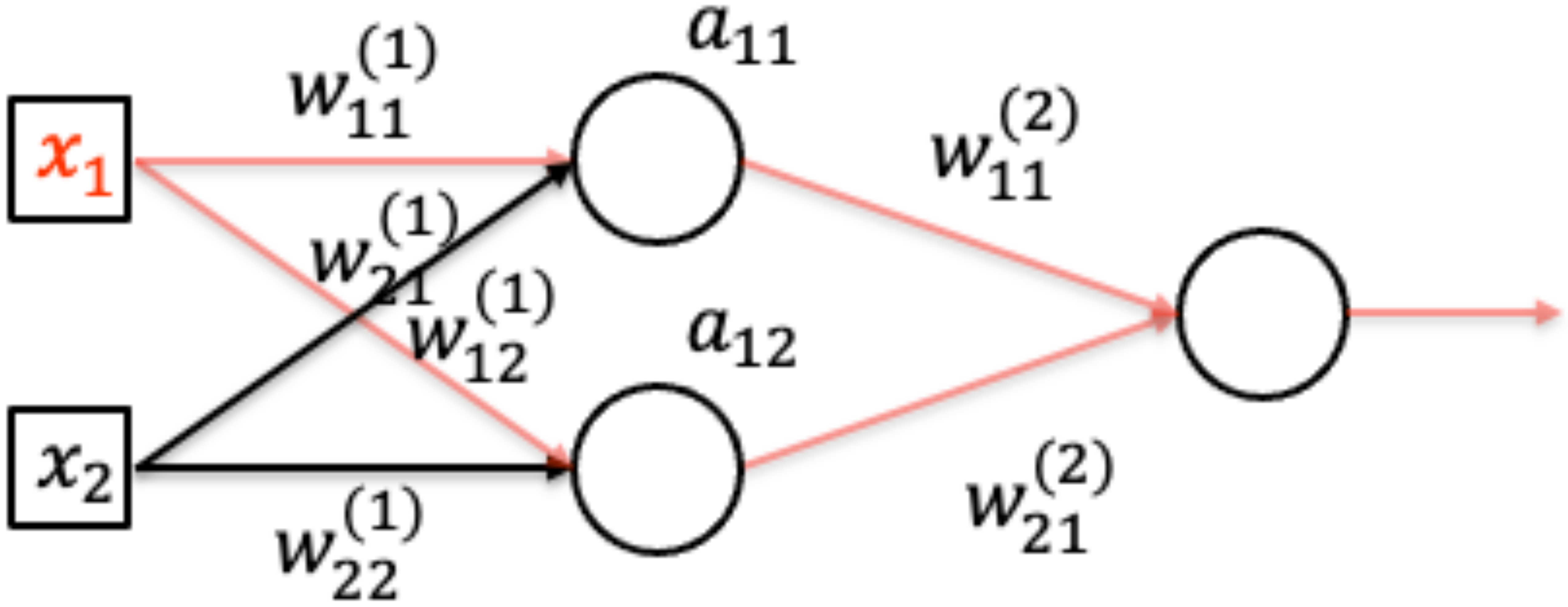


$$\begin{tikzcd} w_{11}^{(1)}x_1 \arrow[dr] & & & & \\ & + \arrow[r] & z_{11} \arrow[r, "\sigma(z_{11})"] & a_{11} \arrow[r] & l(\boldsymbol{x}, y) \\ w_{21}^{(1)}x_2 \arrow[ur] & & & & \end{tikzcd}$$

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

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

Calculate Gradient (on one data point)



$$\begin{tikzcd} w_{11}^{(1)}x_1 \arrow[dr] & & & & \\ & + \arrow[r] & z_{11} \arrow[r, "\sigma(z_{11})", "\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11})"'] & a_{11} \arrow[r, "\frac{\partial l}{\partial a_{11}} = (\hat{y}-y)w_{11}^{(2)}"'] & l(\boldsymbol{x},y) \\ w_{21}^{(1)}x_2 \arrow[ur] & & & & \end{tikzcd}$$

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

Quiz Break

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

A gradients, parameters

B parameters, gradients

C loss, parameters

D parameters, loss

Quiz Break

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

[✓ A gradients, parameters]

B parameters, gradients

C loss, parameters

D parameters, loss

Quiz Break

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

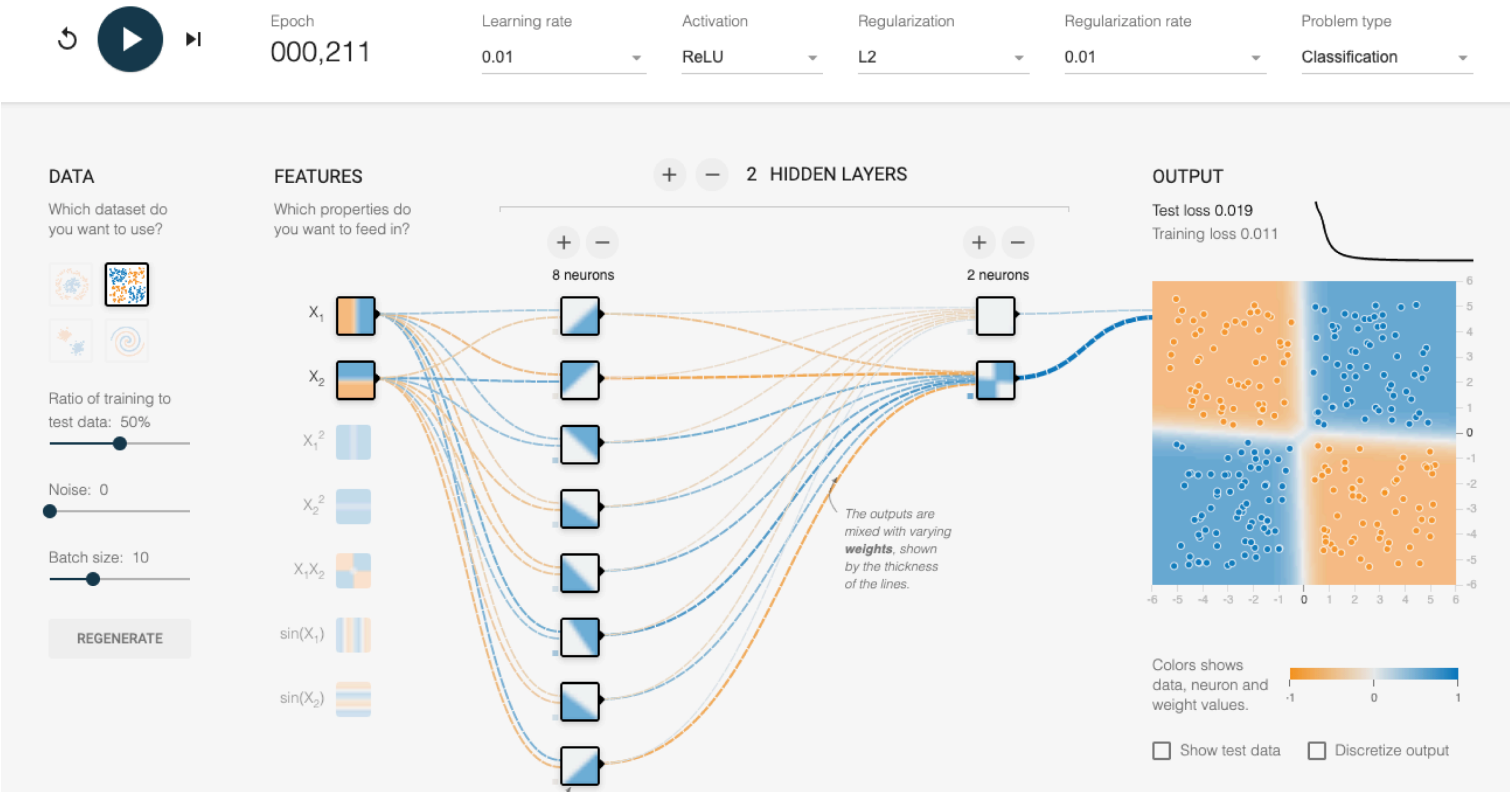
A Gradient Descent

B Stochastic Gradient Descent

[✓ C Minibatch Stochastic Gradient Descent]

D Computation Graph

Demo: Learning XOR using neural net



Epoch 000,211

Learning rate: 0.01

Activation: ReLU

Regularization: L2

Regularization rate: 0.01

Problem type: Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

FEATURES

Which properties do you want to feed in?

2 HIDDEN LAYERS

8 neurons

2 neurons

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

OUTPUT

Test loss 0.019

Training loss 0.011

Colors shows data, neuron and weight values.

□ Show test data □ Discretize output

- 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