INF721

2024/2

UFV

Deep Learning

L5: Multilayer Perceptron

Logistics

Announcements

- PA1: Logistic Regression is out!
- There is a holiday next week!

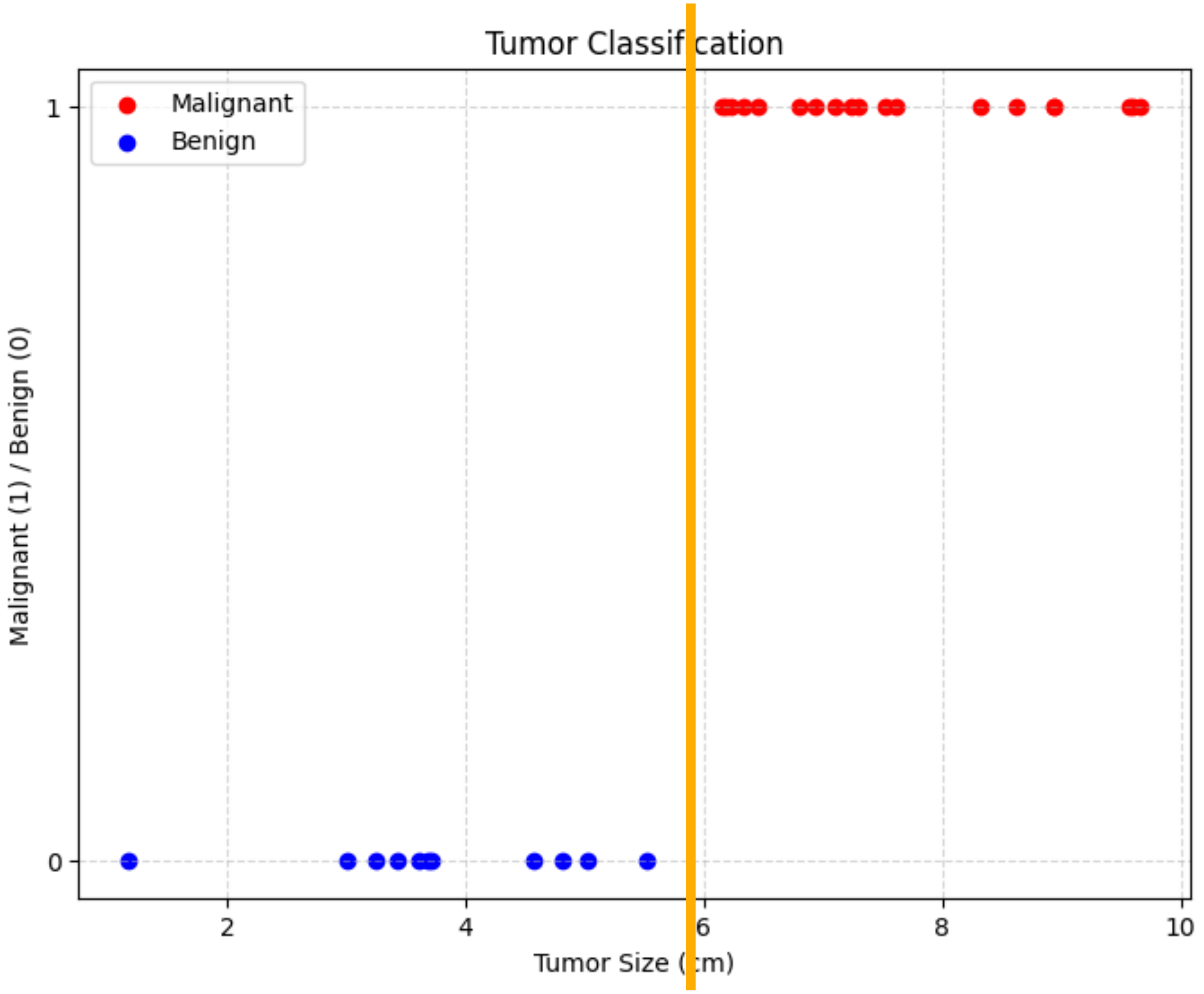
Last Lecture

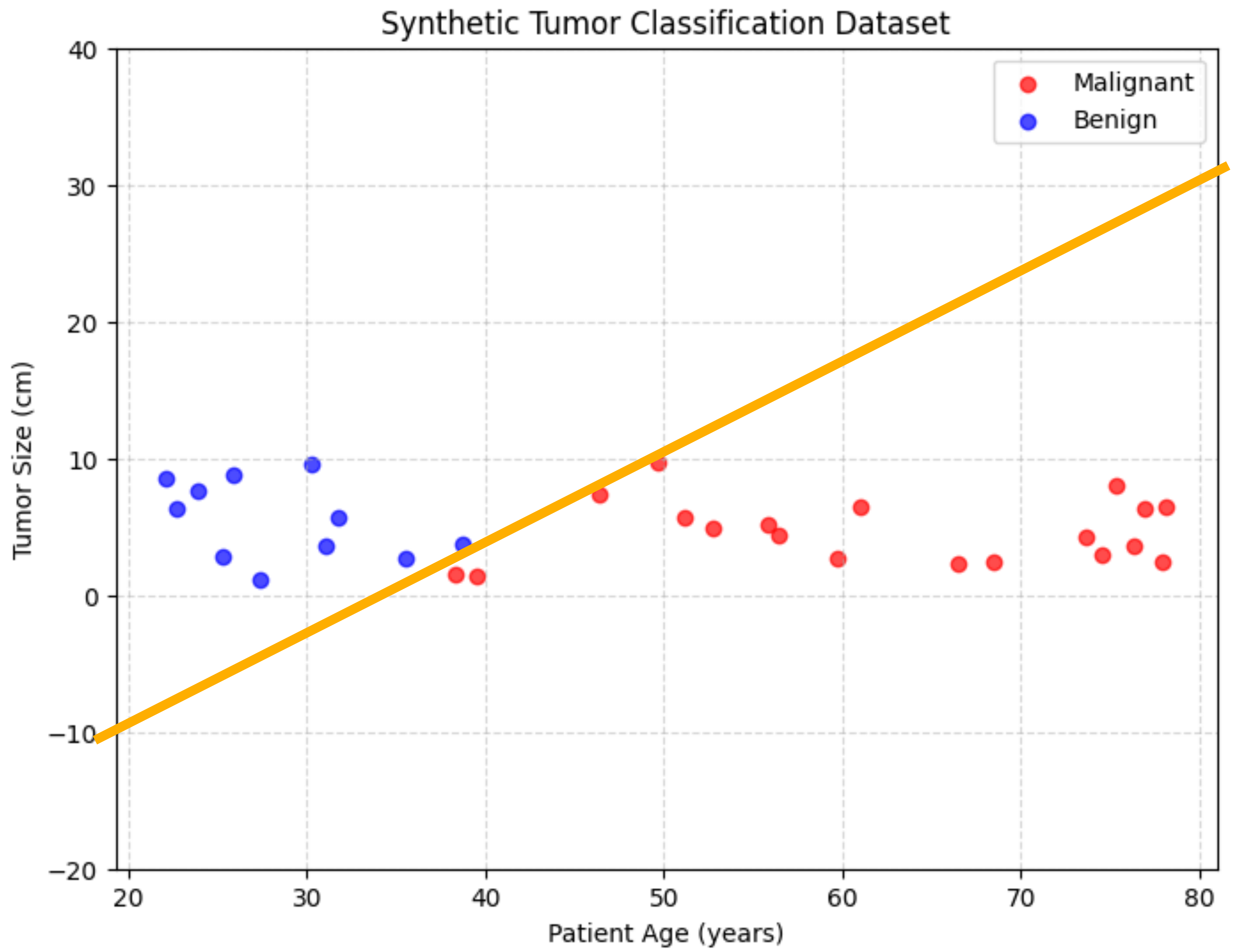
- Linear Regression with Multiple Variables
- Vectorization
- Logistic Regression
  - Sigmoid/Logistic Function
  - Binary Cross-Entropy Loss
  - Gradient Descent for Logistic Regression

Lecture Outline

- Linearly Separable Problems
- The Perceptron
- Linear Models as a Neuron
- Non-linearly Separable Problems
- Multilayer Perceptron
  - Forward Pass
  - Vectorization
- Activation Functions
- Categorical Cross-Entropy Loss

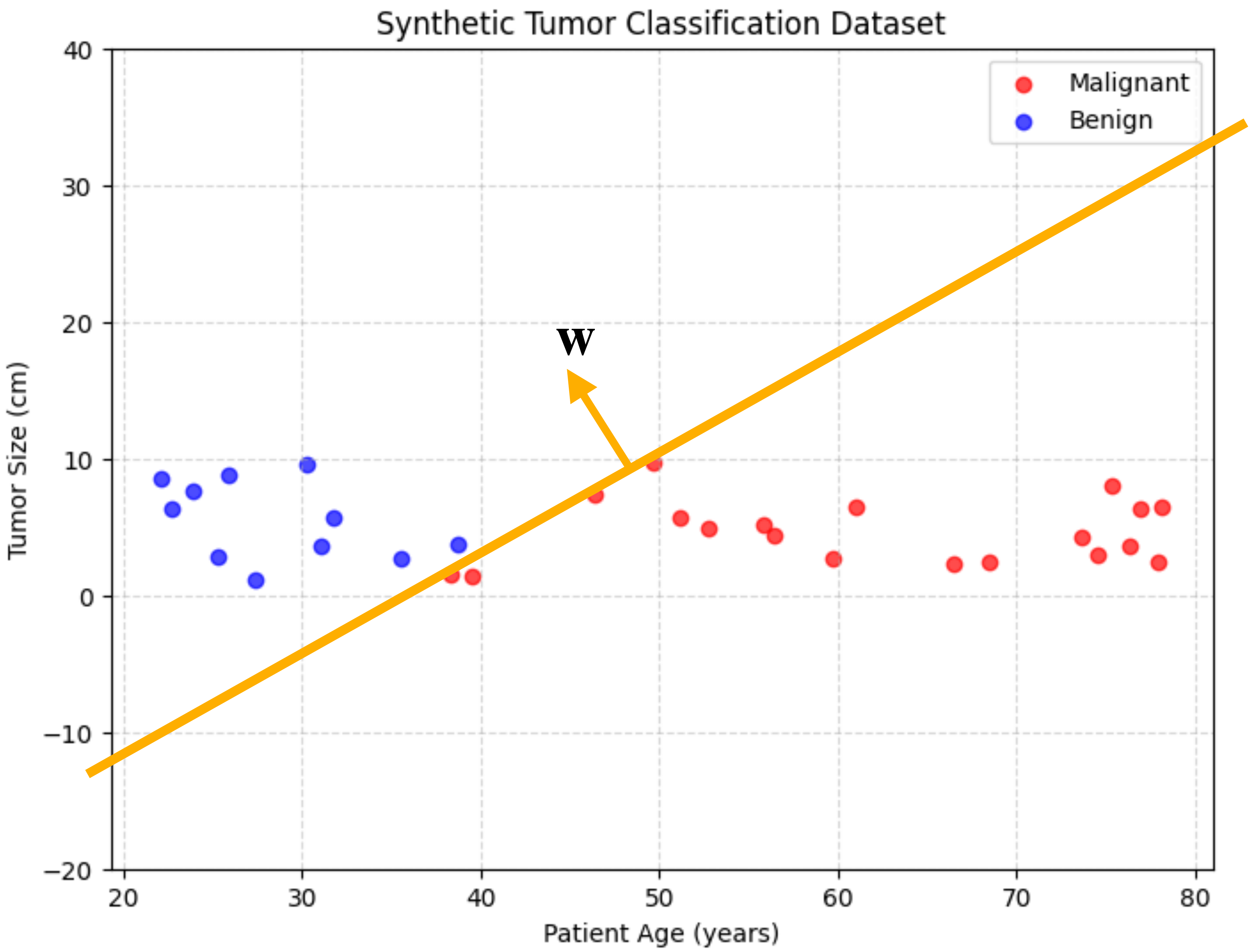
Linearly Separable Problems





UFV

The Perceptron: the first trainable neuron



$$h(\mathbf{x}) = sgn(\mathbf{w} \cdot \mathbf{x} + b) \quad \mathbf{w} = [-0.7, 1] \quad b = 25$$

$$sgn(z) = \begin{cases} +1, & z \geq 0 \\ -1, & z < 0 \end{cases}$$

$$\mathbf{x}^{(1)} = [50, 10]$$

$$h(\mathbf{x}^{(1)}) = sgn(-0.7 \cdot 51 + 1 \cdot 8 + 25) = sgn(-2.7) = -1$$

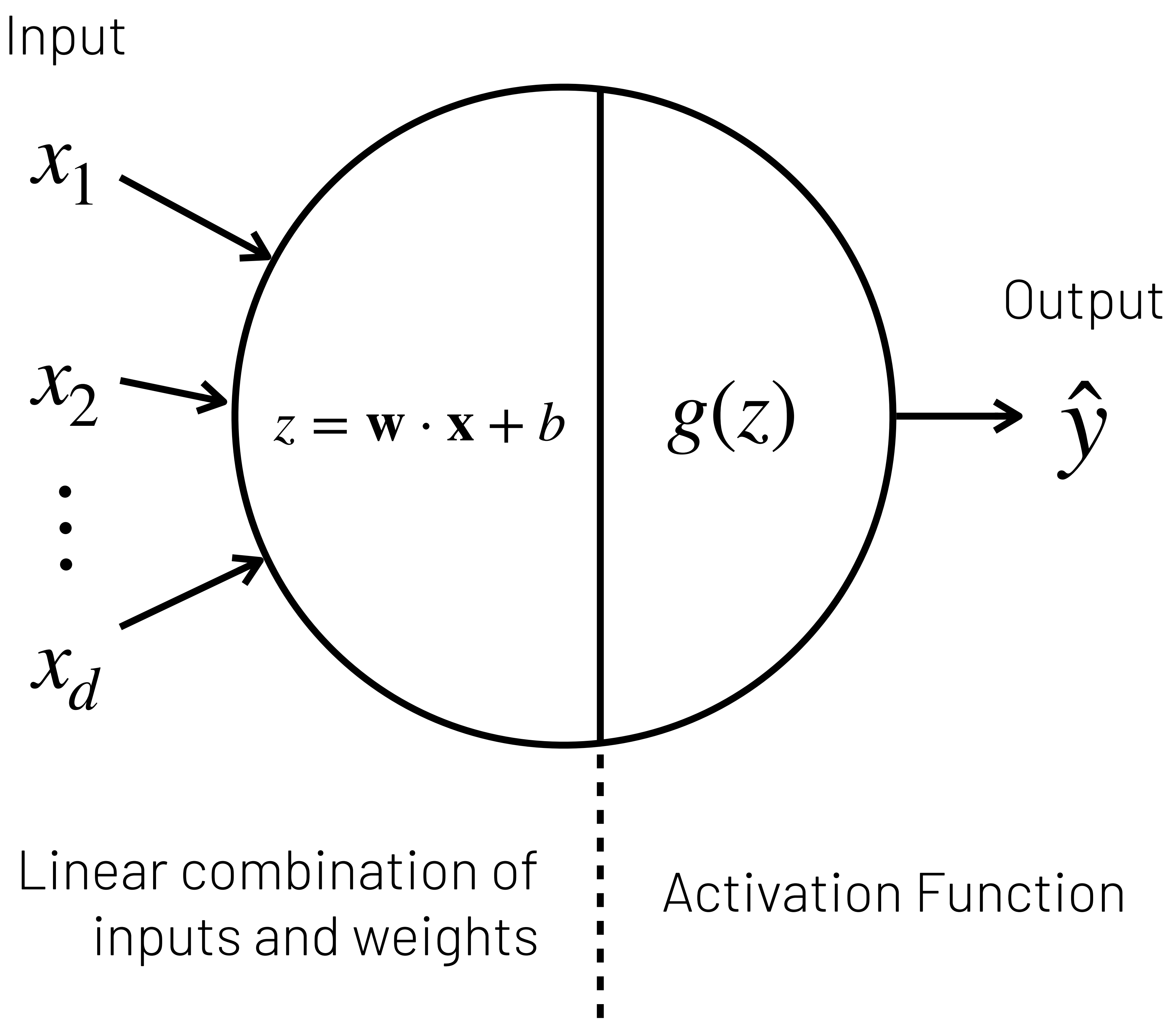
$$\mathbf{x}^{(2)} = [10, 30]$$

$$h(\mathbf{x}^{(2)}) = sgn(-0.7 \cdot 10 + 1 \cdot 30 + 25) = sgn(48) = 1$$

- The Perceptron is not trained with Gradient Descent because the sgn function is not differentiable. Instead, it uses a simple update rule based on misclassifications.

UFV

An Artificial Neuron



A **Neuron** is a computational unit composed of:

1. A linear combination of inputs $\mathbf{x}$ and weights $\mathbf{w}$:
   $z = \mathbf{w} \cdot \mathbf{x} + b$
2. A typically non-linear activation function $g(z)$

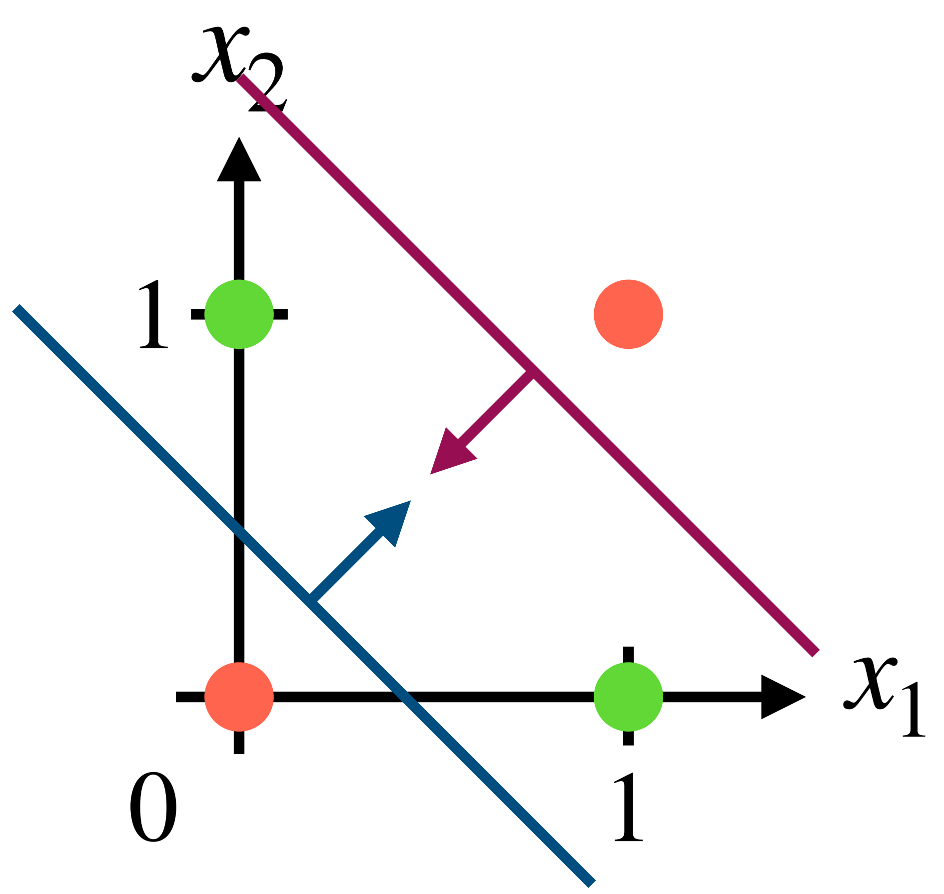
Linear models activation functions:

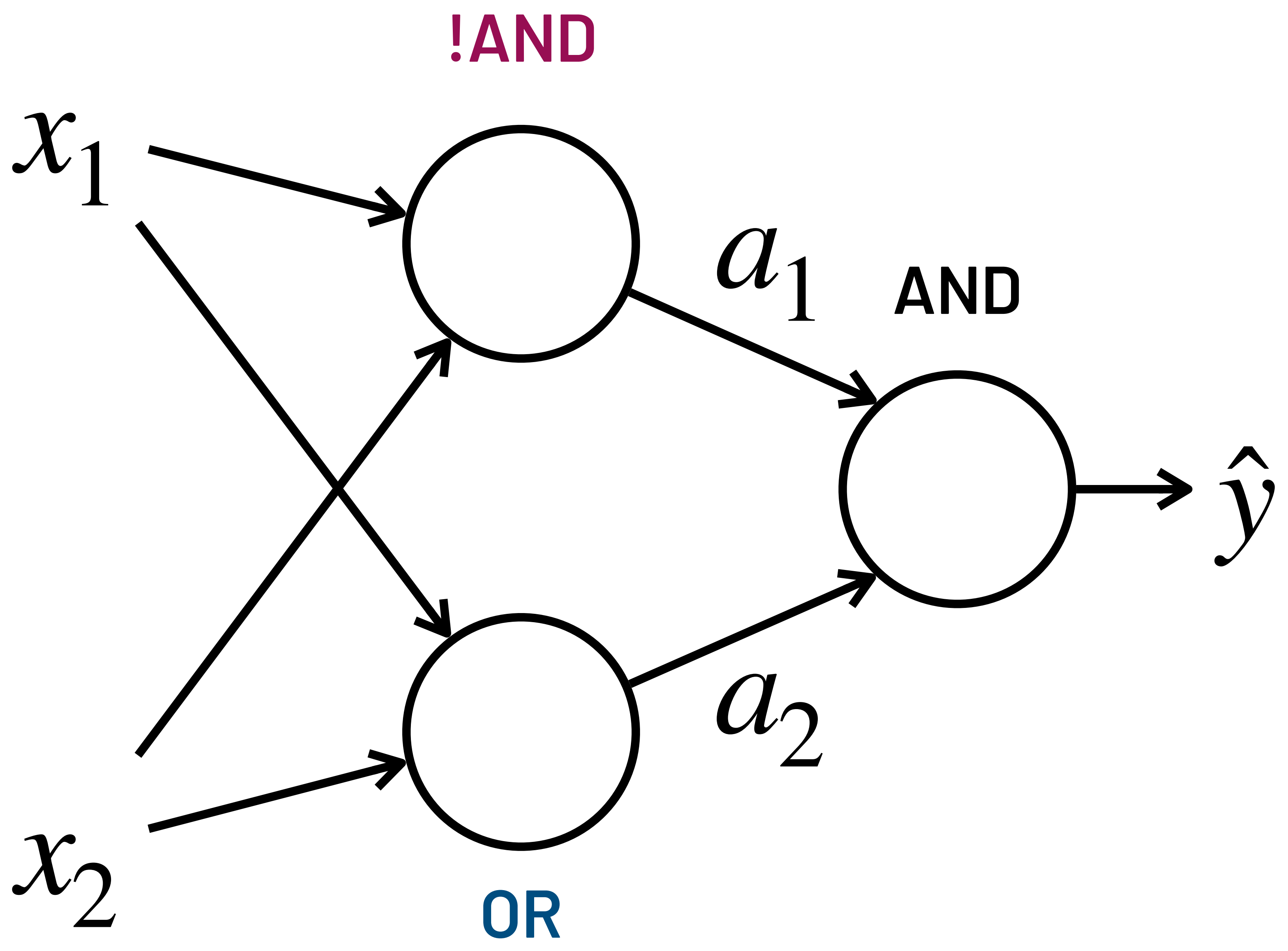
- Linear Regression: $g(z) = z$
- Logistic Regression: $g(z) = \dfrac{1}{(1 + e^{-z})}$
- Perceptron: $g(z) = \begin{cases} 1, & z \geq 0 \\ -1, & z < 0 \end{cases}$

Non-linearly Separable Problems

$$f(x_1, x_2) = x_1 \text{ XOR } x_2$$

$x_2 \backslash x_1$	1	0
0	1	0
1	0	1

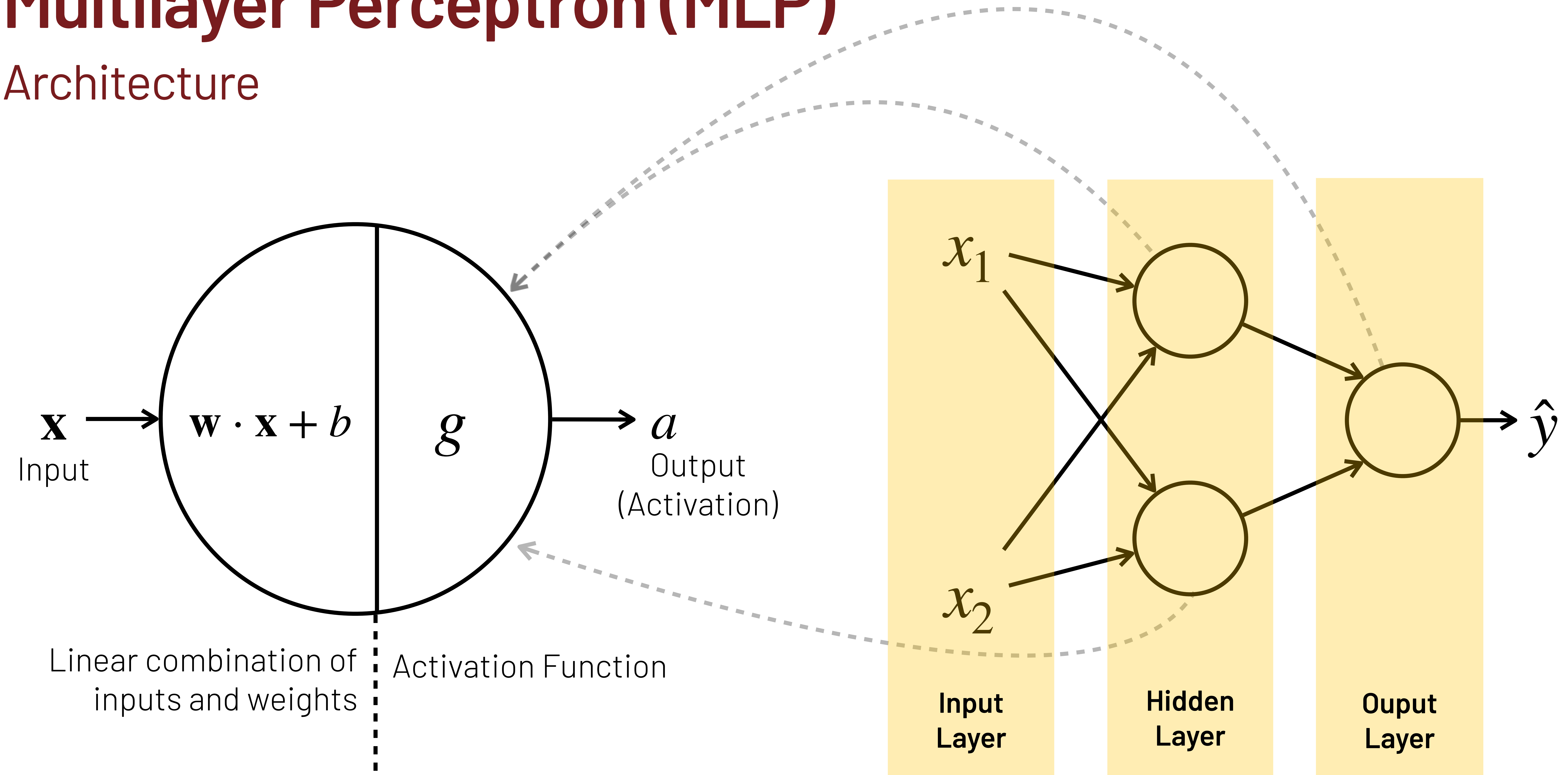




Neural Networks learn new representations $\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$ from inputs data $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, called **latent representations**, that can turn a non-linearly separable problem into linearly separable!

Multilayer Perceptron (MLP)

Architecture



UFV

Forward Pass

For a single input $\mathbf{x}$ $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

$$a_1 = g^{[1]}(w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + b_1^{[1]})$$

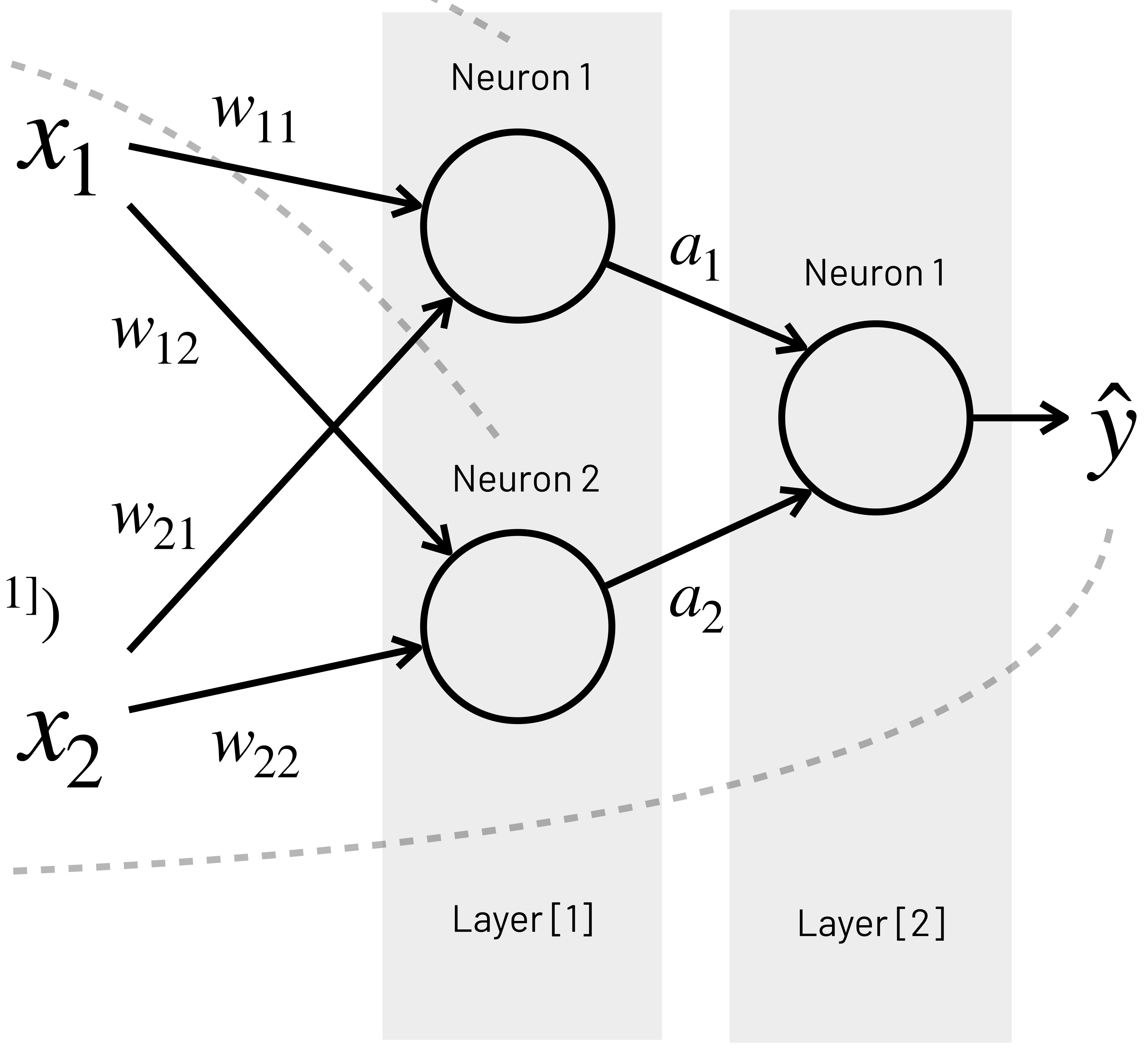
$$a_2 = g^{[1]}(w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + b_2^{[1]})$$

$$\mathbf{a}^{[1]} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = g^{[1]}(\begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + b_1^{[1]} \\ w_{11}^{[1]}x_1 + w_{22}^{[1]}x_2 + b_2^{[1]} \end{bmatrix})$$

$$= g^{[1]}(\begin{bmatrix} w_{11}^{[1]} & w_{21}^{[1]} \\ w_{11}^{[1]} & w_{22}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \end{bmatrix}) = g^{[1]}(W^{[1]}\mathbf{x} + \mathbf{b}^{[1]})$$

$$\hat{y} = g^{[2]}(w_{11}^{[2]}a_1 + w_{21}^{[2]}a_2 + b_1^{[2]})$$

$$\hat{y} = g^{[2]}(\begin{bmatrix} w_{11}^{[2]} & w_{21}^{[2]} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} + b_1^{[2]}) = g^{[2]}(W^{[2]}\mathbf{a} + b_1^{[2]})$$



Forward Pass

For a dataset X with m examples

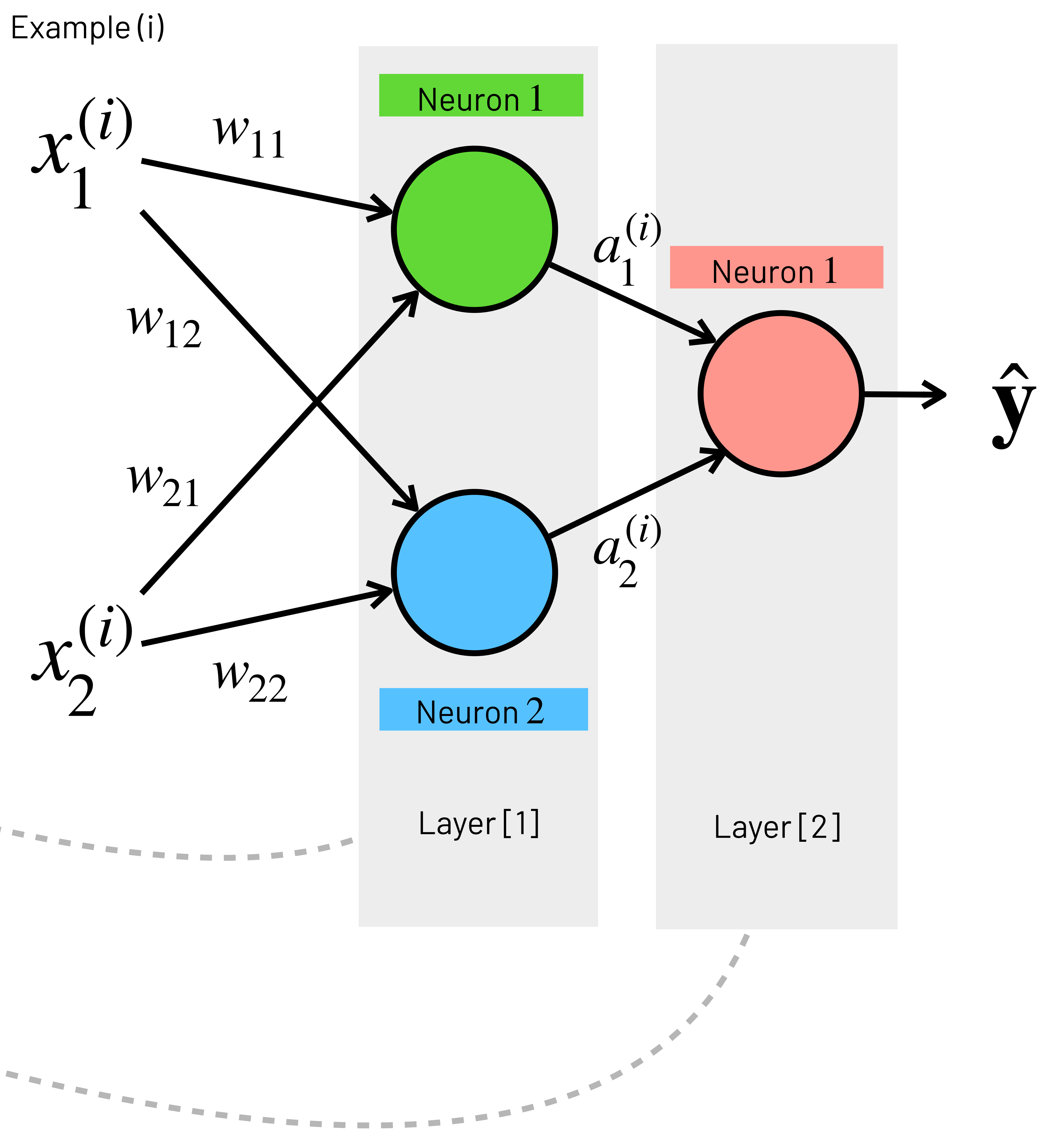
$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(m)} \end{bmatrix}$$

$$W^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{21}^{[1]} \\ w_{12}^{[1]} & w_{22}^{[1]} \end{bmatrix} \quad \mathbf{b}^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \end{bmatrix}$$

$$A^{[1]} = g^{[1]}(W^{[1]}X + \mathbf{b}^{[1]}) = g^{[1]}\left(\begin{bmatrix} a_1^{(1)} & a_1^{(2)} & \dots & a_1^{(m)} \\ a_2^{(1)} & a_2^{(2)} & \dots & a_2^{(m)} \end{bmatrix}\right)$$

$$W^{[2]} = \begin{bmatrix} w_{11}^{[2]} & w_{21}^{[2]} \end{bmatrix}$$

$$\hat{\mathbf{y}} = g^{[2]}(W^{[2]}A^{[1]} + b^{[2]}) = \begin{bmatrix} \hat{y}^{(1)} & \hat{y}^{(2)} & \dots & \hat{y}^{(m)} \end{bmatrix}$$



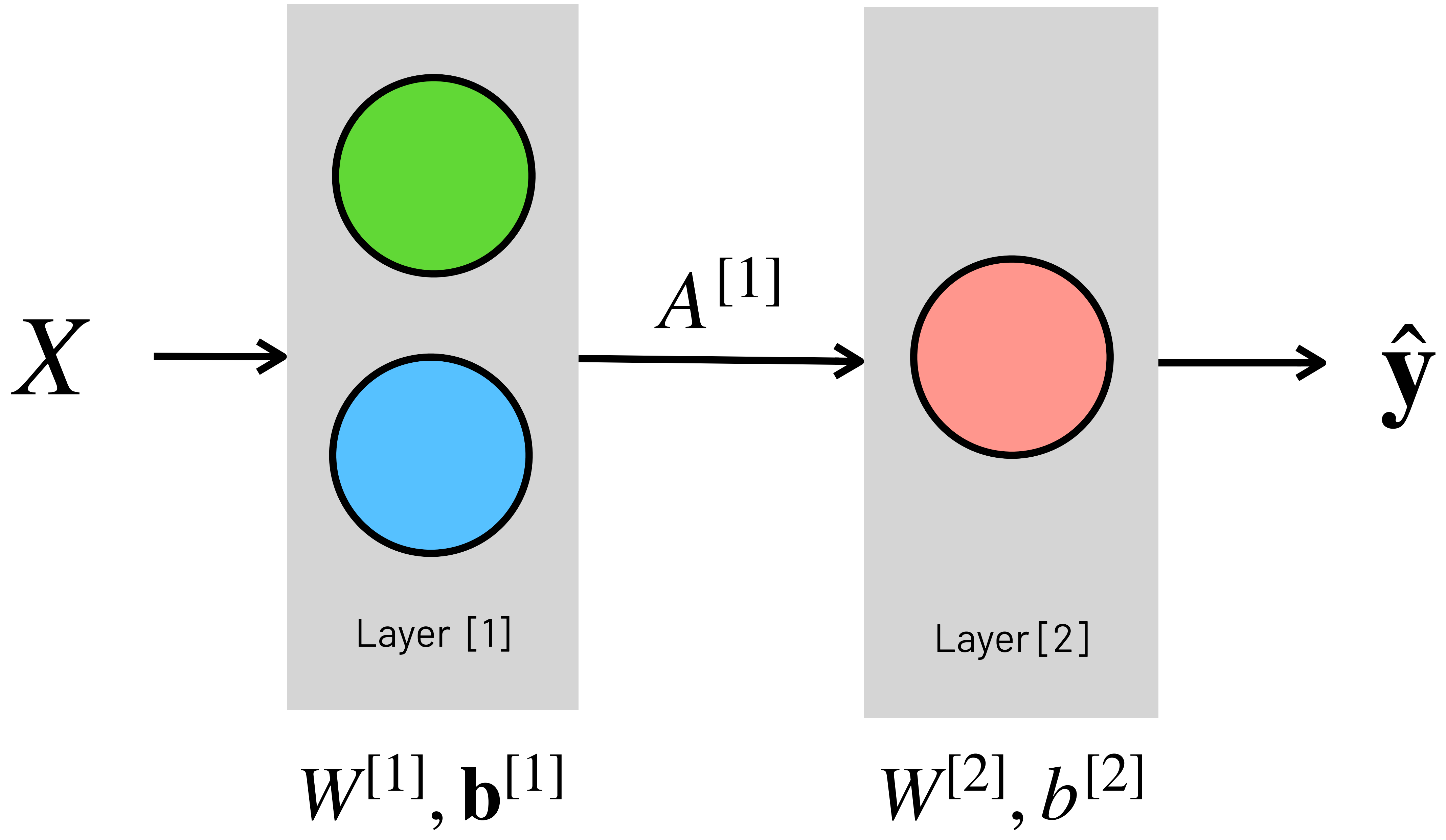
Hypothesis Space

Hypothesis Space H

$$Z^{[1]} = W^{[1]}X + \mathbf{b}^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$\hat{\mathbf{y}} = g^{[2]}(Z^{[2]})$$

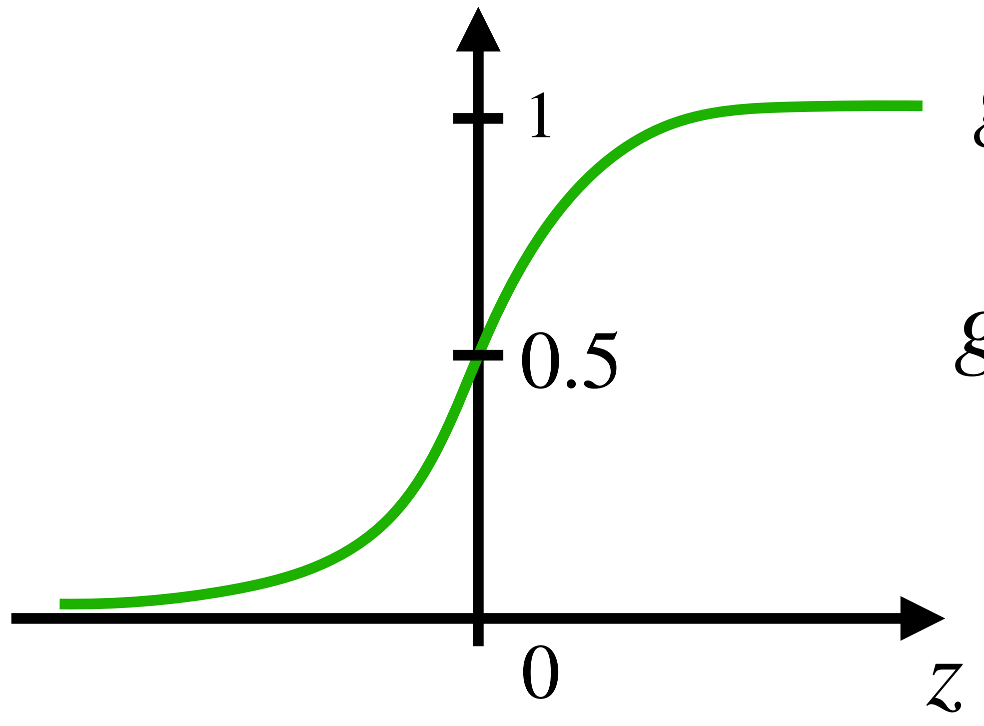
$$\hat{\mathbf{y}} = h(\mathbf{x}) = g^{[2]}(W^{[2]} \cdot g^{[2]}(W^{[1]}X + \mathbf{b}^{[1]}) + b^{[2]}$$
$$h(\mathbf{x}) = g^{[2]}(W^{[2]} \cdot h^{[1]}(X) + b^{[2]})$$

MLPs learn composite functions!



Activation Functions

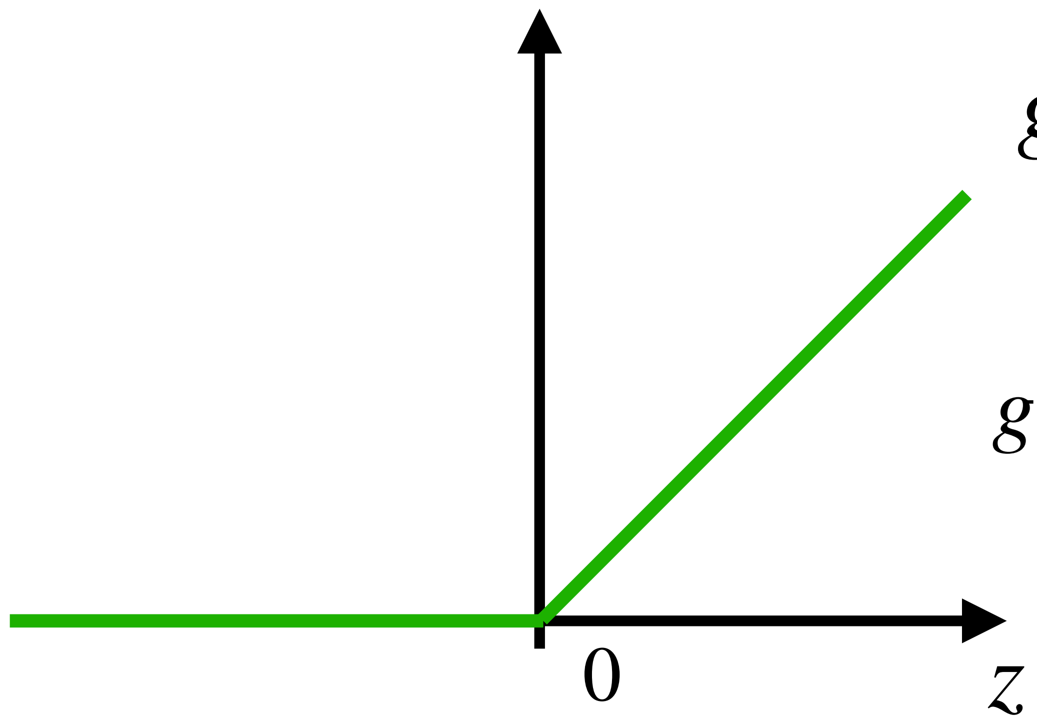
Logistic (sigmoid)



$$g(z) = \frac{1}{(1 + e^{-z})}$$

$$g'(z) = g(z)(1 - g(z))$$

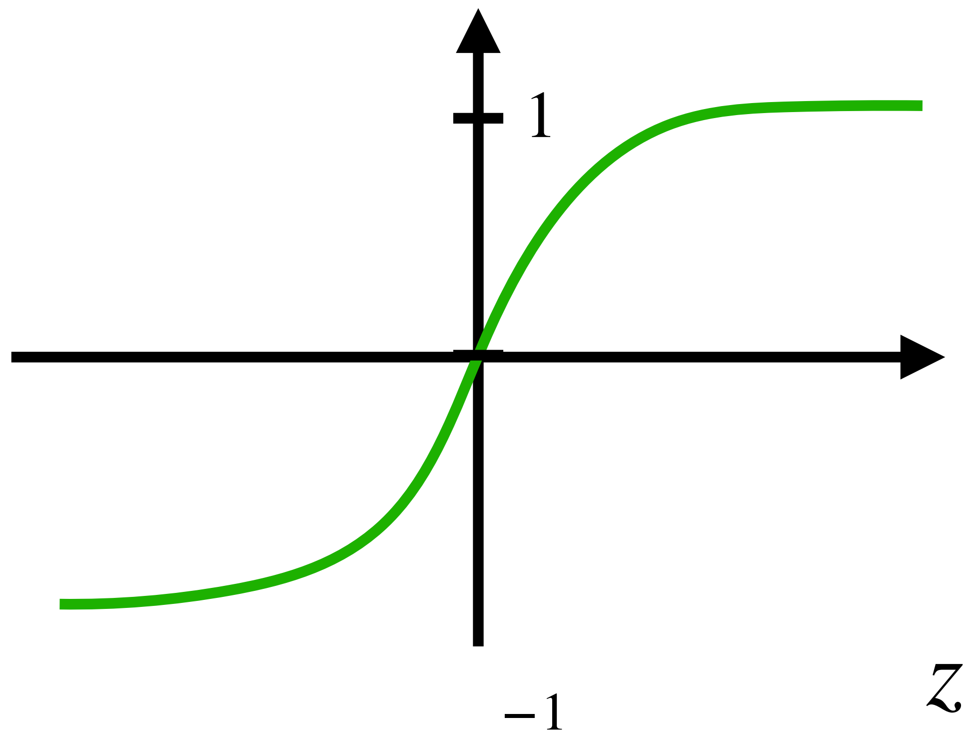
Rectified Linear Unit (ReLU)



$$g(z) = max(0, z)$$

$$g'(z) = \begin{cases} 1 \text{ if } z > 0 \\ 0 \text{ if } z < 0 \\ \nexists, \text{ if } z = 0 \end{cases}$$

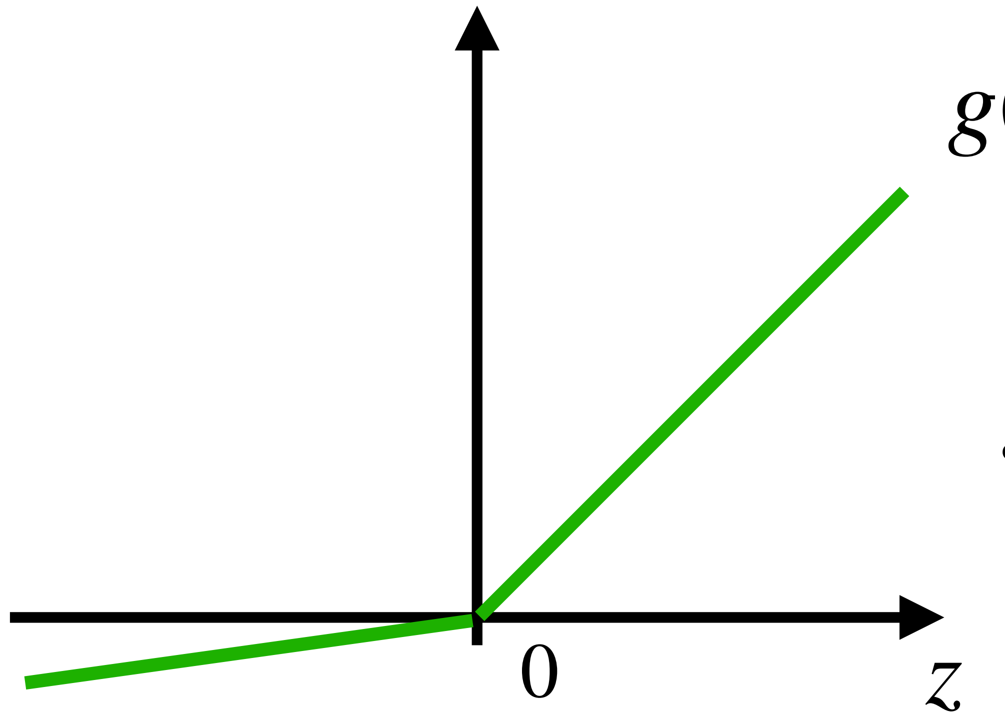
Hyperbolic Tangent



$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Leaky ReLU



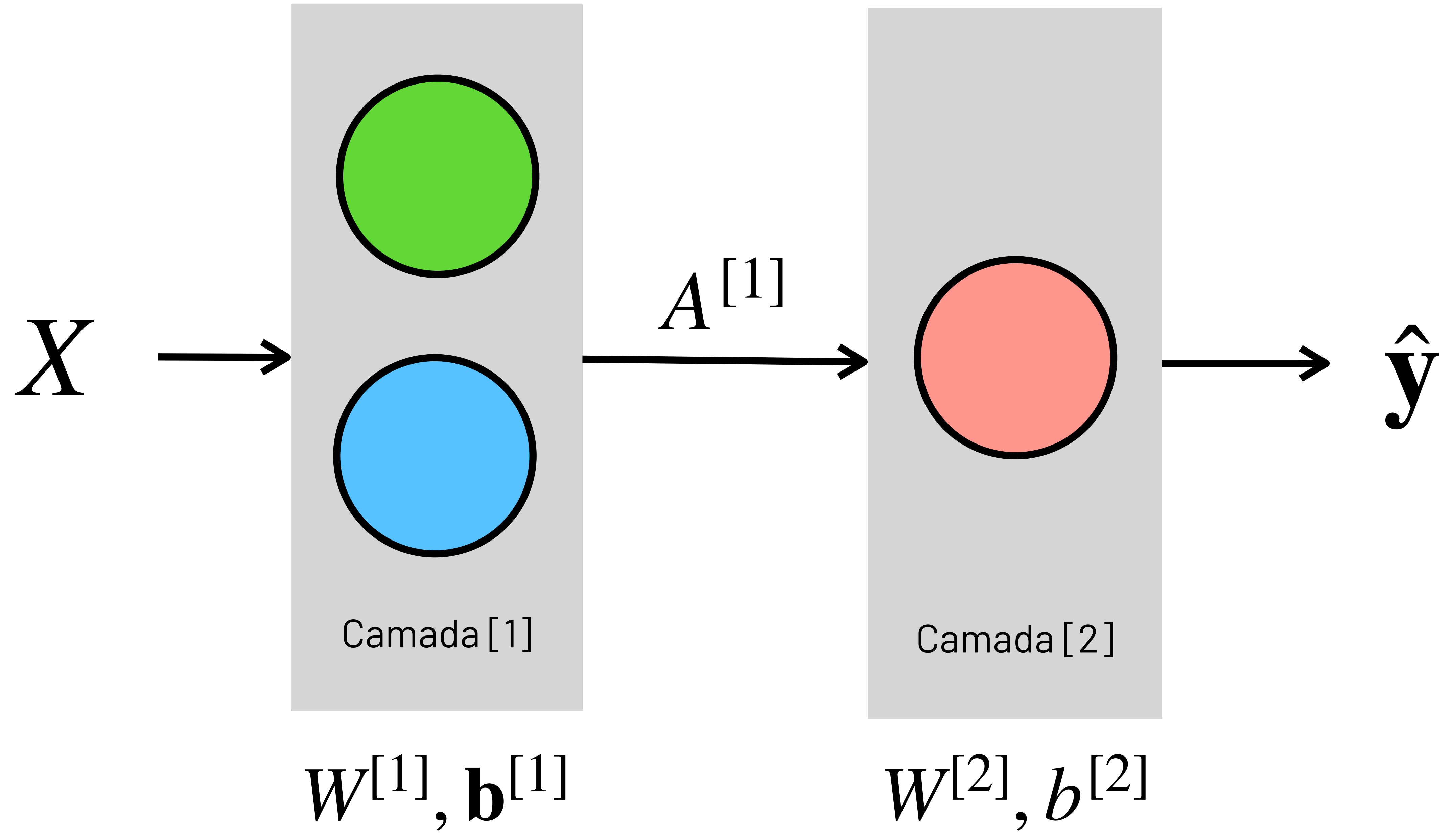
$$g(z) = max(0.01z, z)$$

$$g'(z) = \begin{cases} 1 \text{ if } z > 0 \\ 0.01 \text{ if } z < 0 \\ \nexists, \text{ if } z = 0 \end{cases}$$

UFV

Why do we need non-linear activation functions?

$$Z^{[1]} = W^{[1]}X + \mathbf{b}^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$\hat{\mathbf{y}} = g^{[2]}(Z^{[2]})$$



$$\hat{\mathbf{y}} = h(\mathbf{x}) = g^{[2]}(W^{[2]} \cdot g^{[1]}(W^{[1]} \cdot \mathbf{x} + \mathbf{b}^{[1]}) + b^{[2]})$$
$$h(\mathbf{x}) = W^{[2]} \cdot (W^{[1]} \cdot \mathbf{x} + \mathbf{b}^{[1]}) + b^{[2]}$$
$$h(\mathbf{x}) = \underbrace{(W^{[2]} \cdot W^{[1]})}_{W'} \cdot \mathbf{x} + \underbrace{(W^{[2]} \cdot \mathbf{b}^{[1]}) + b^{[2]}}_{b'}$$

$$h(x) = W' \cdot \mathbf{x} + b'$$

If we use linear activation functions, our hypothesis **will be linear!**

Initializating MLP weights

In Neural Networks with at least 1 hidden layer (MLPs), we need to initialize the weights with random varibales close to zero.

If we initialize the weights with zeros, all neurons in the hidden layers will be equal!

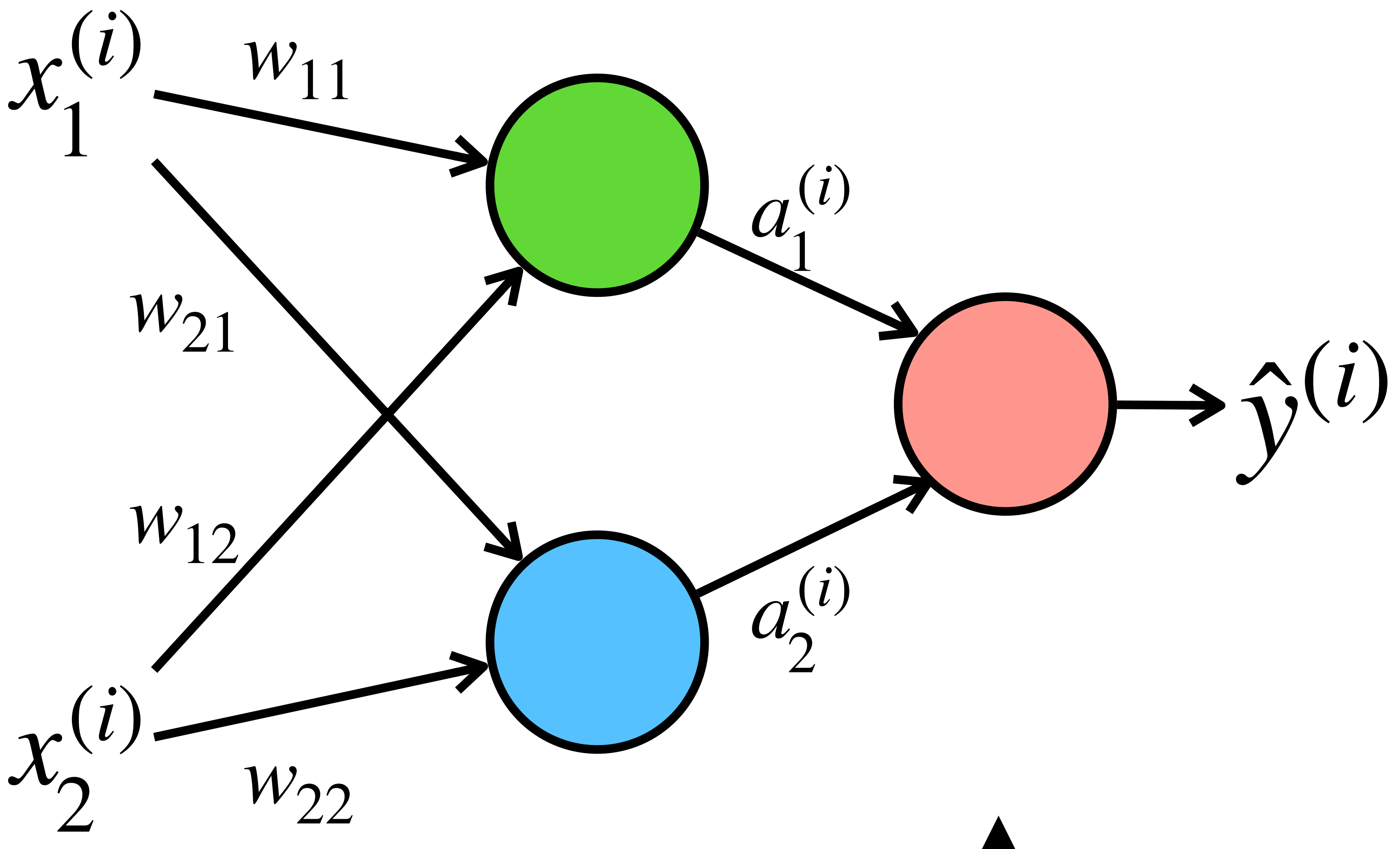
$$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$W^{[2]} = [0 \quad 0] \quad b^{[2]} = 0$$

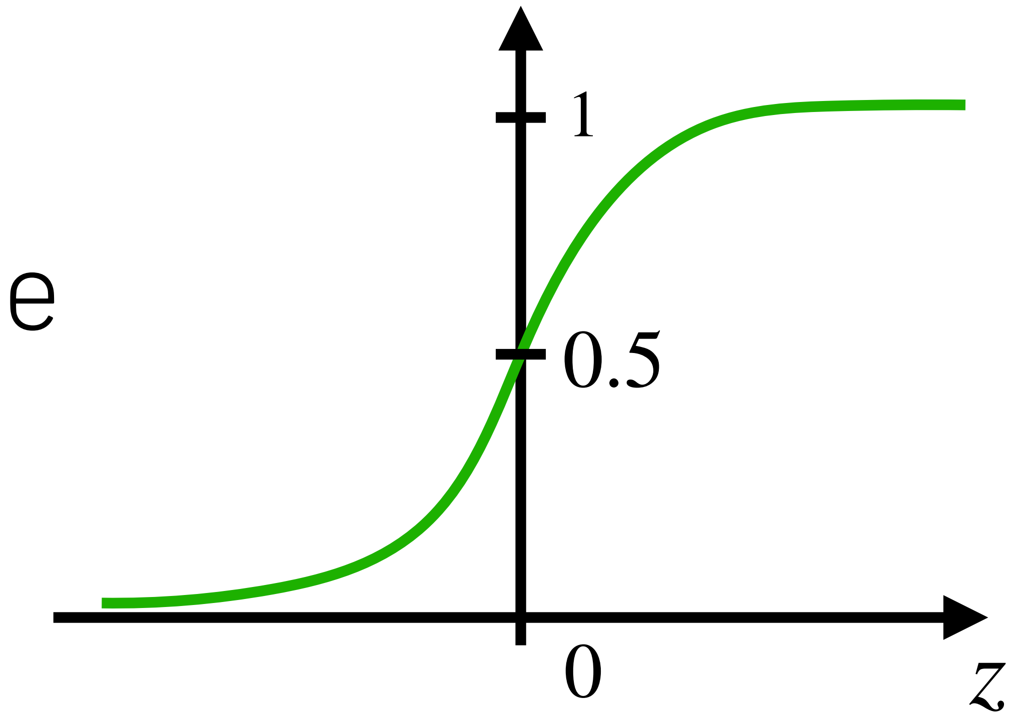
$$\downarrow$$

$$a_1^{(i)} = a_2^{(i)} \longrightarrow dZ_1^{[1]} = dZ_2^{[1]}$$

$$dW = \begin{bmatrix} u & u \\ u & u \end{bmatrix}$$



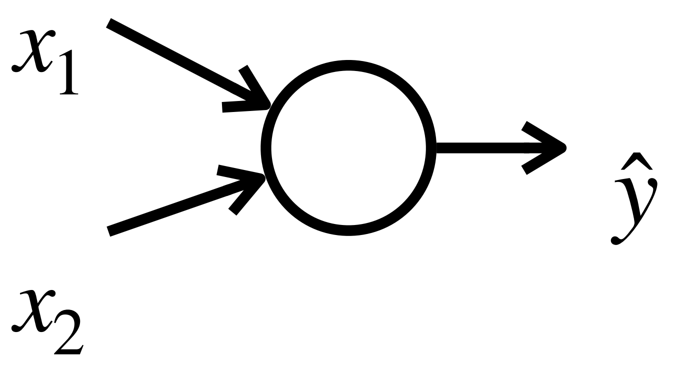
In regions close to zero the gradient is greater!



Deep Neural Networks

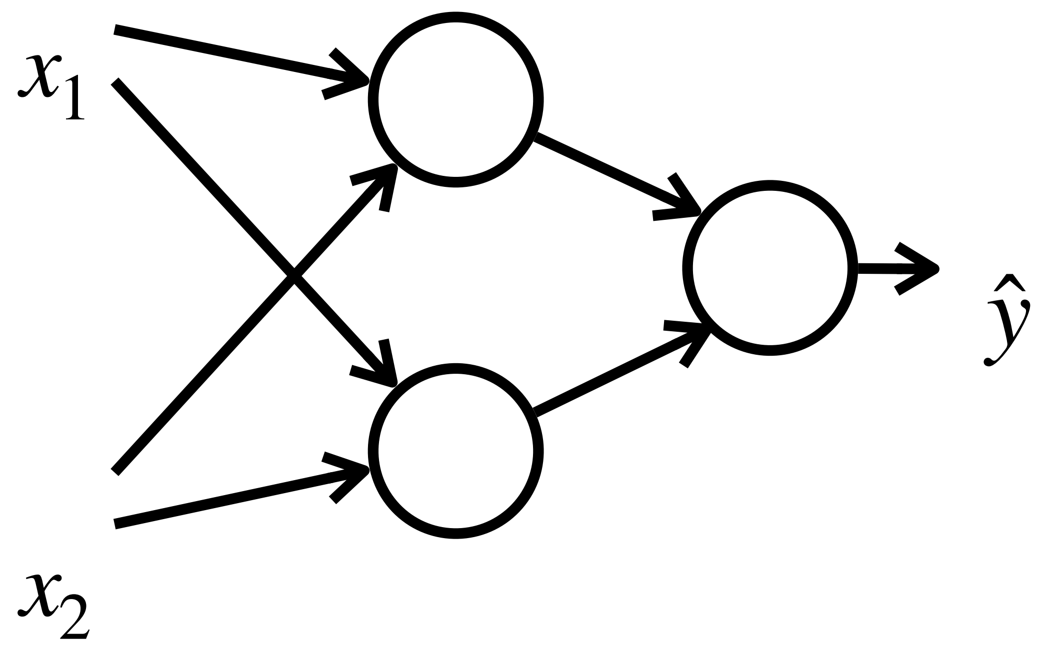
Logistic/Linear Regression

NN with 1 layer (shallow)



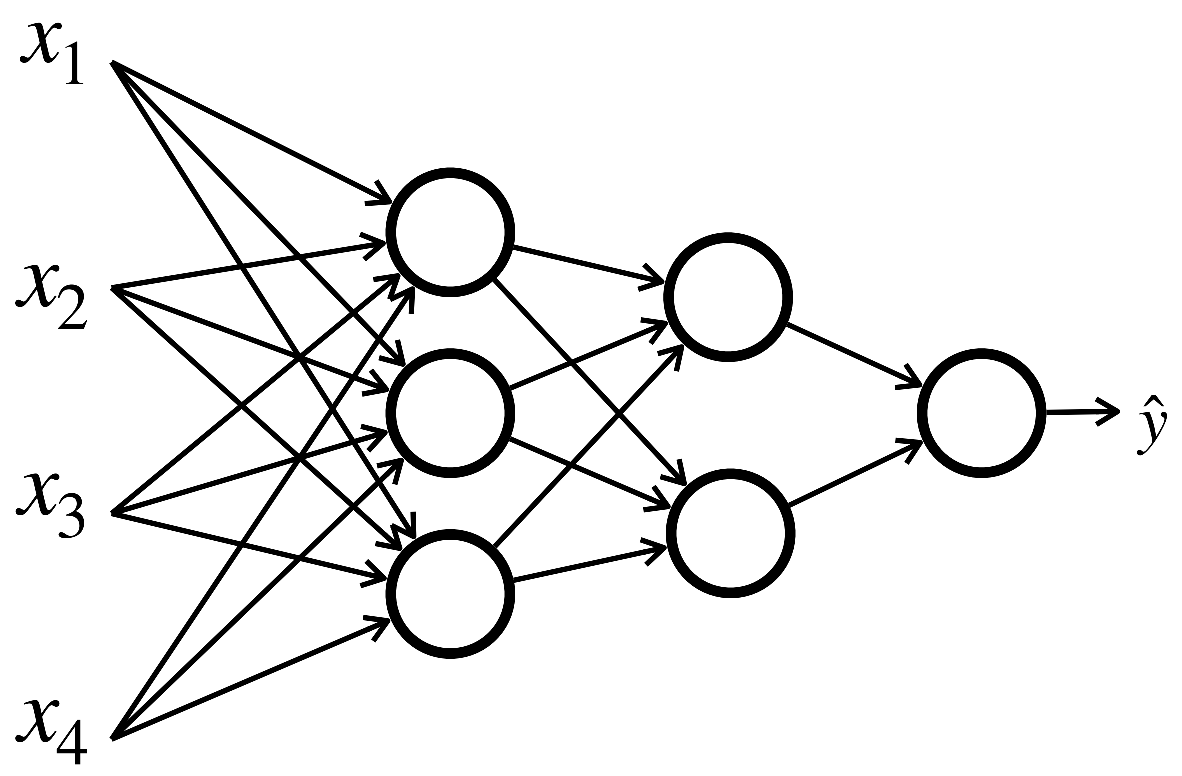
1 hidden layer

NN with 2 layers (shallow)



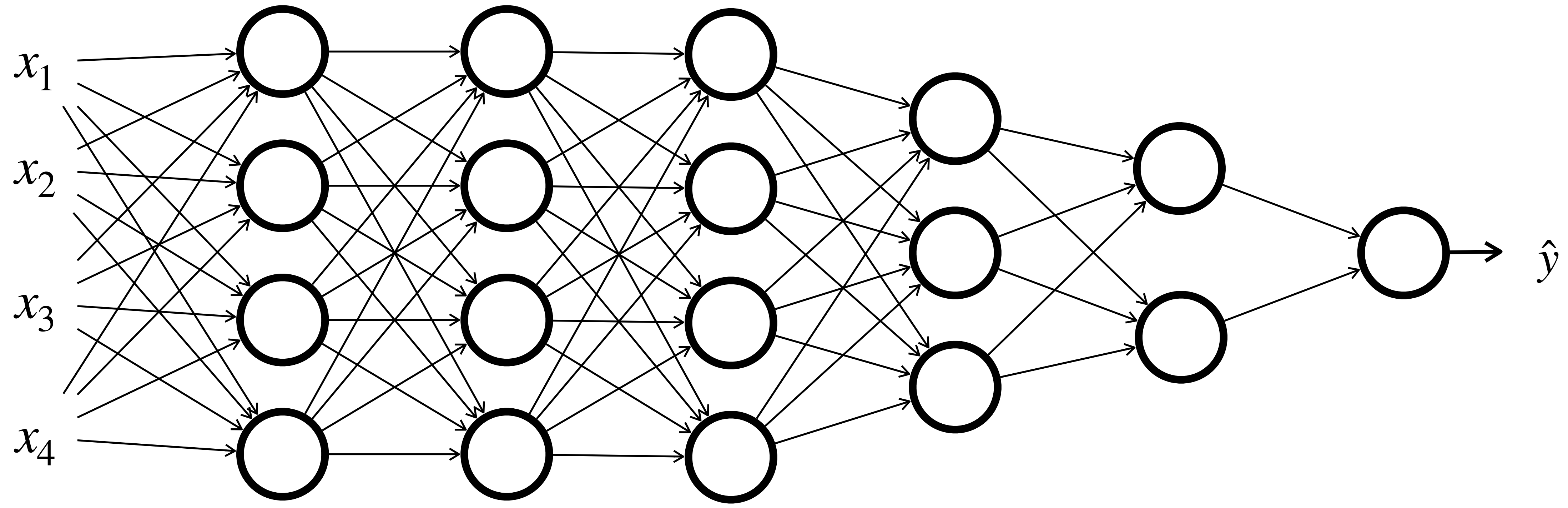
2 hidden layers

NN with 3 layers (shallow)



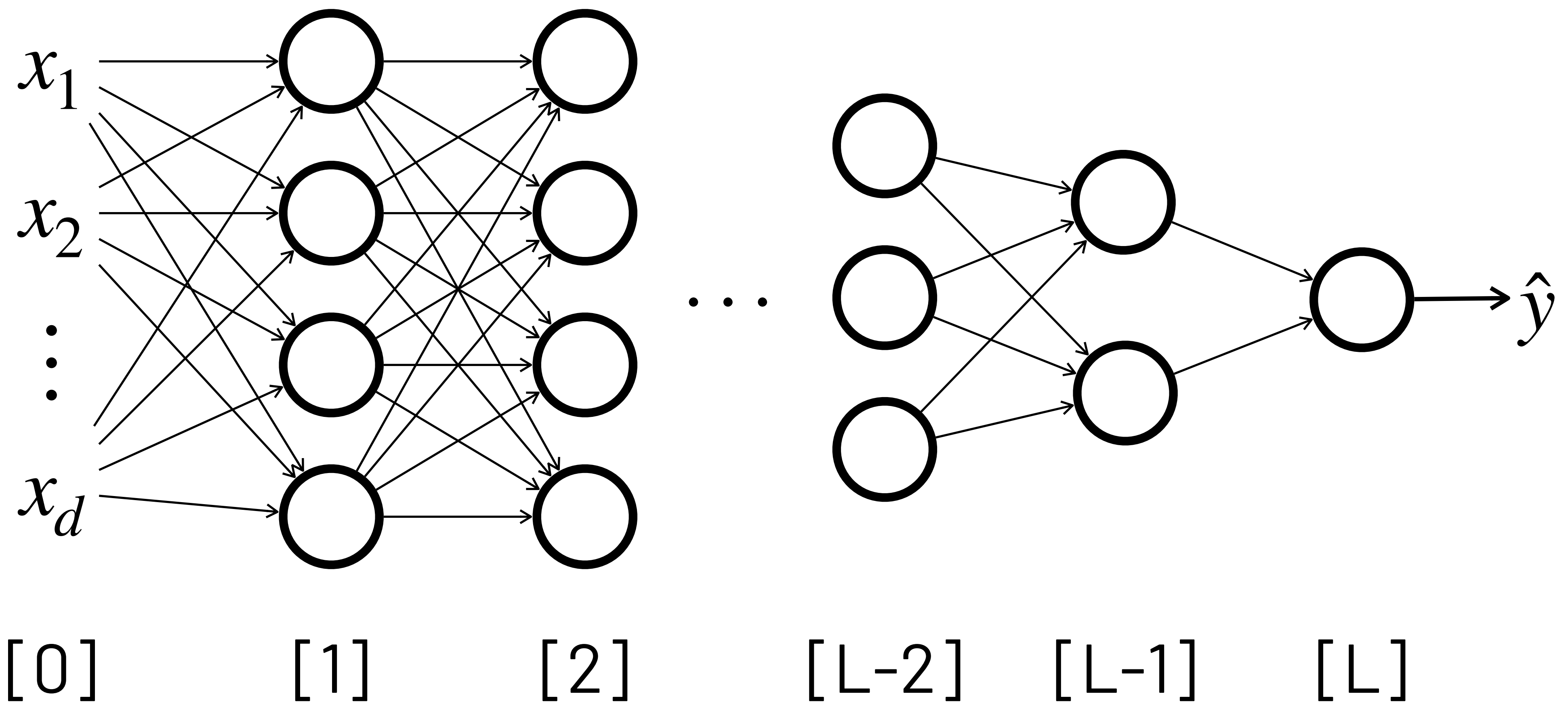
5 hidden layers

NN with 6 layers (deep)



Deep Neural Networks Forward Pass

NN with L layers



For a single example $\mathbf{x}$:

$$\mathbf{z}^{[1]} = W^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$
$$\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$$
$$\mathbf{z}^{[2]} = W^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$
$$\mathbf{a}^{[2]} = g^{[2]}(\mathbf{z}^{[2]})$$
$$\dots$$
$$\mathbf{z}^{[L]} = W^{[L]}\mathbf{a}^{[L-1]} + b^{[L]}$$
$$\hat{y} = g^{[L]}(\mathbf{z}^{[L]})$$

General formulation:

$$\mathbf{z}^{[l]} = W^{[l]}\mathbf{a}^{[l-i]} + \mathbf{b}^{[l]}$$
$$\mathbf{a}^{[l]} = g^{[l]}(\mathbf{z}^{[l]})$$

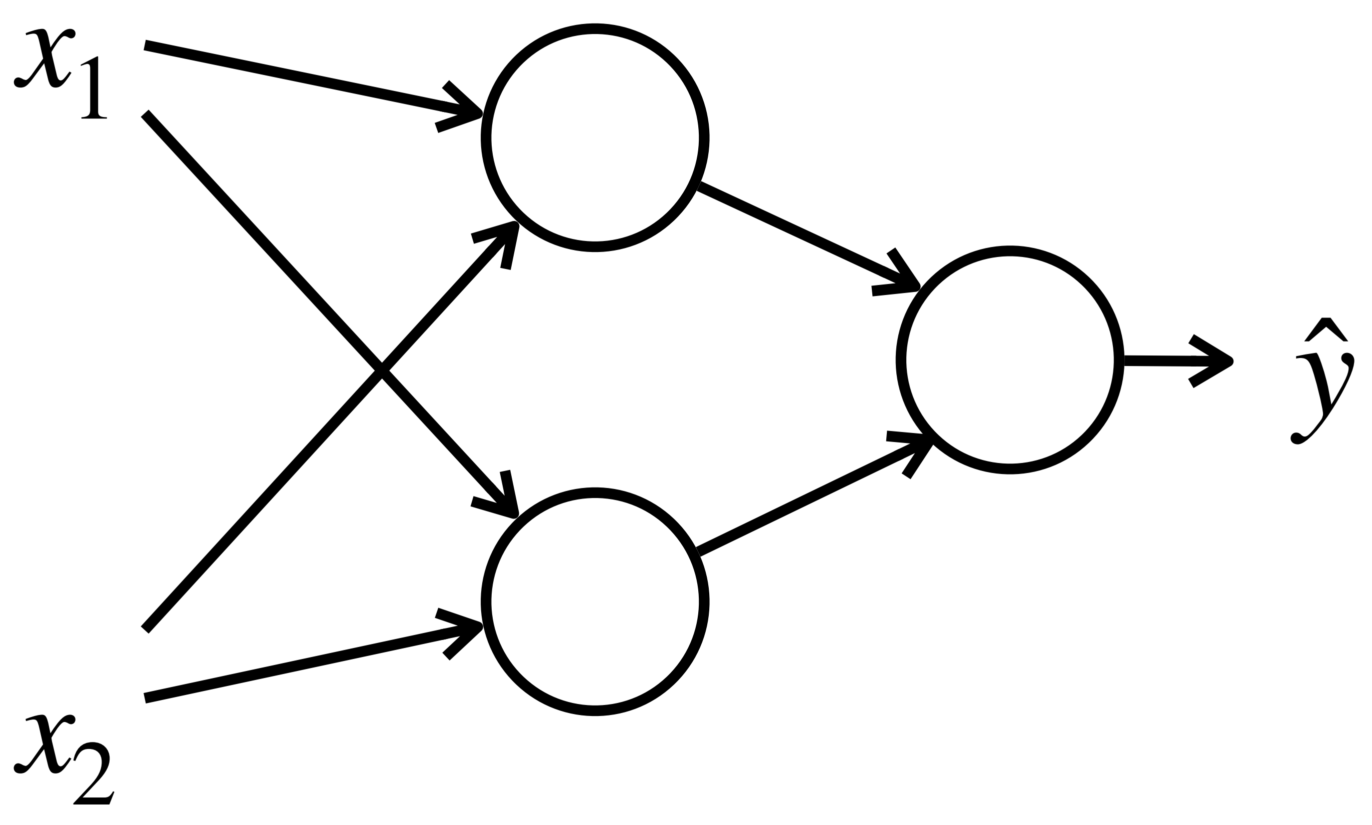
Vectorized

$$Z^{[l]} = W^{[l]}A^{[l-1]} + \mathbf{b}^{[l]}$$
$$A^{[l]} = g^{[l]}(Z^{[l]})$$

$$A^{[0]} = X$$
$$A^{[L]} = \hat{Y}$$

Output Layer with a Single Neuron

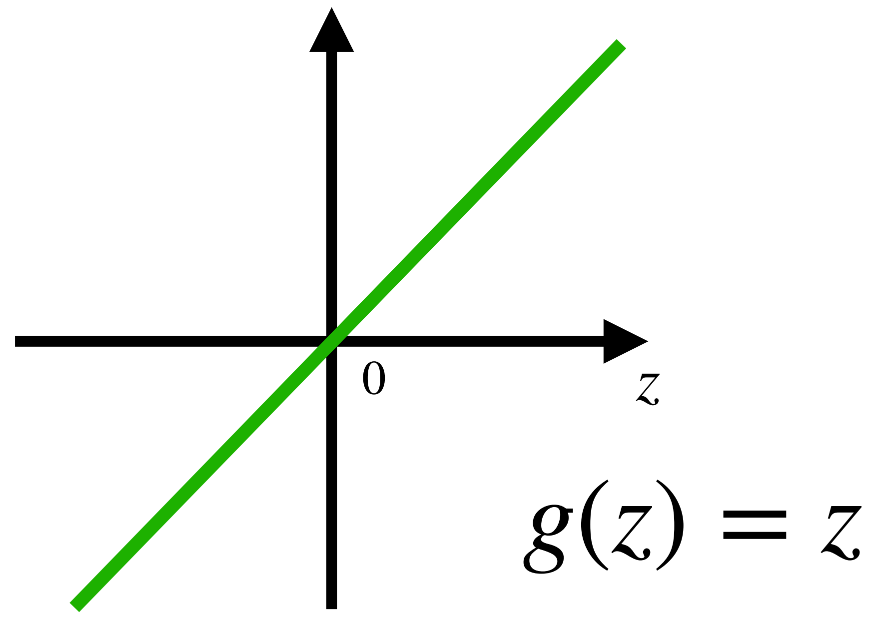
For **Regression and Binary Classification** problems, our Neural Network will have a single neuron in the output layer.



Regression

Linear Activation Function

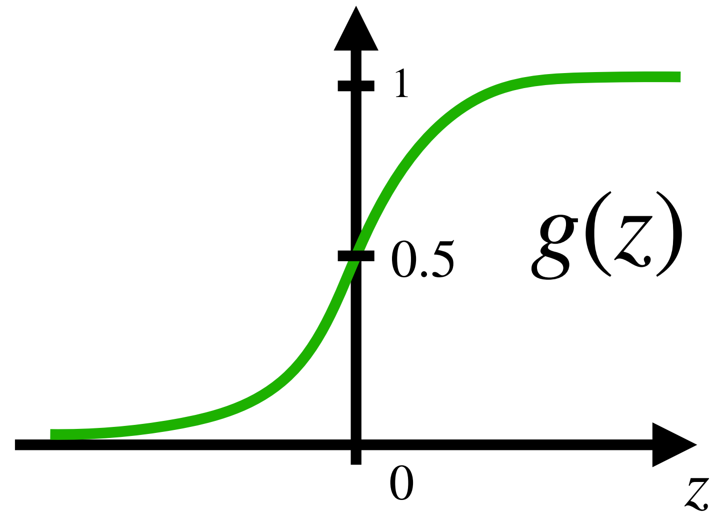
$\hat{y} = 418.7$



Binary Classification

Sidmoid Activation Function

$\hat{y} = P(y = 1 \mid \mathbf{x}) = 0.3$

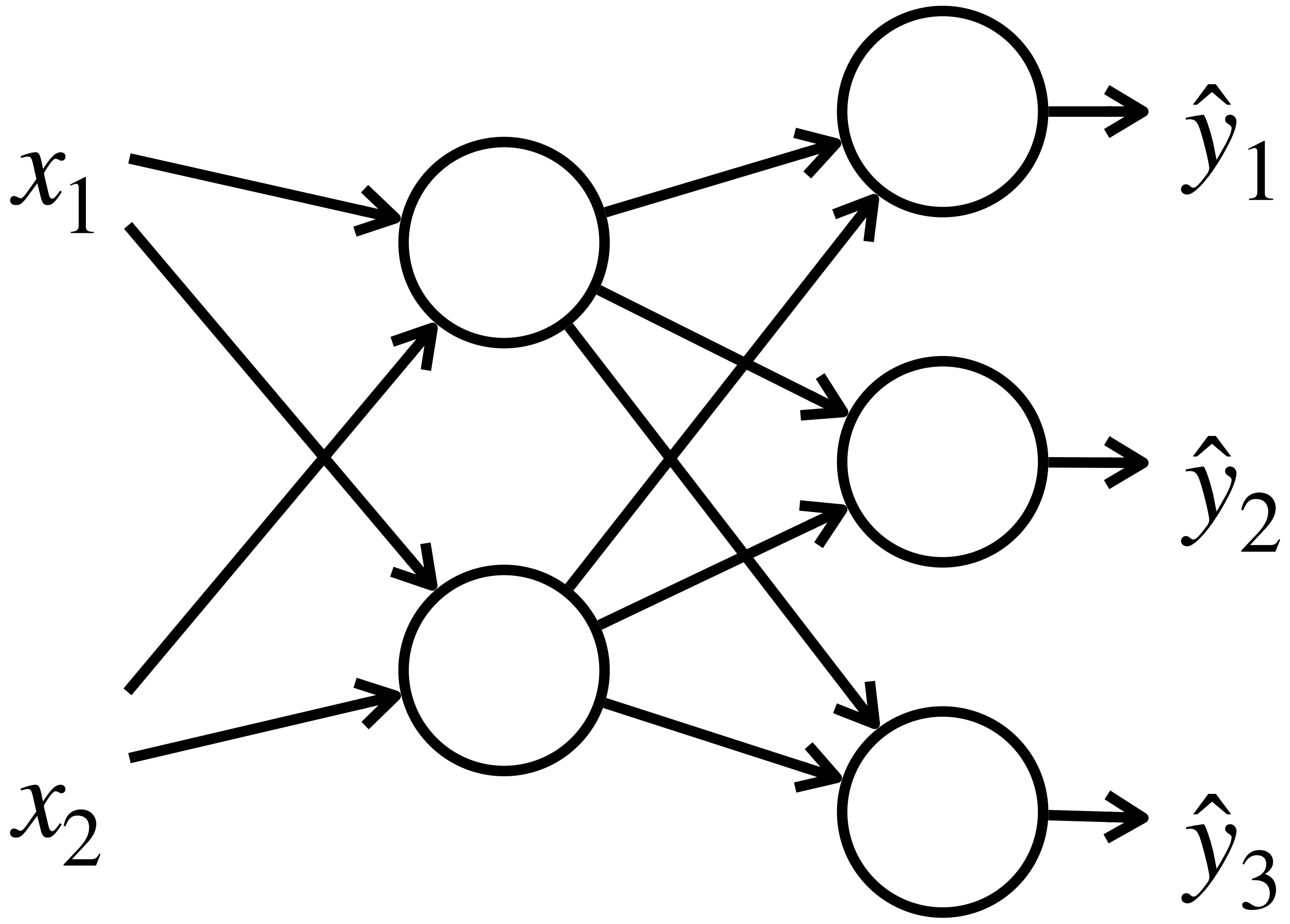


$$g(z) = \frac{1}{(1 + e^{-z})}$$

UFV

Output Layer with Multiple Neurons

For **multiclass classification problems**, the number of neurons in the output layer is equal to the number of classes in the problem and the activation function is called **softmax**.



Multiclass Classification
Softmax Activation Function

$$g(z) = \frac{e^z}{\sum_{j=1}^{C} e_j^z}$$

$$Z^{[2]} = \begin{bmatrix} 5 \\ 2 \\ -1 \end{bmatrix} \qquad e^z = \frac{\begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \end{bmatrix}}{\sum_{j=i}^{C} e_i^z = 156.17} \qquad \hat{y}^{(i)} = \begin{bmatrix} 0.531 \\ 0.238 \\ 0.229 \end{bmatrix} \begin{matrix} \text{Class 1} \\ \text{Class 2} \\ \text{Class 3} \end{matrix}$$

Probability Distribution

Categorial Cross-Entropy Loss Function

For multiclass classification problems, we use the Categorical Cross-Entropy Loss Function, which is a generalization of the BCE Loss:

Binary Cross-Entropy

$$L(h) = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i \log(\hat{y}^{(i)}) + (1 - y^{(i)})\log(1 - \hat{y}^{(i)})\right]$$

- $y^{(i)}$: true label (0 or 1) for example (i)
- $\hat{y}^{(i)}$: predicted probability for example (i)

Example:

- True label $y^{(i)} = 1$
- Predicted probability $\hat{y}^{(i)} = 0.8$

$$L = -[1 * log(0.8) + (1-1) * log(1-0.8)]$$
$$= -[log(0.8)] \approx 0.223$$

Categorical Cross-Entropy

$$L(h) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{c=1}^{C} y_c^{(i)} \log(\hat{y}_c^{(i)})$$

- $y_c^{(i)}$: true label of class c for example (i)
- $\hat{y}_c^{(i)}$: predicted probability of class c for example (i)

Example:

- True labels: $y^{(i)} = [0,1,0]$
- Predicted probabilities: $\hat{y}^{(i)} = [0.1,0.7,0.2]$

$$L = -[0 * log(0.1) + 1 * log(0.7) + 0 * log(0.2)]$$
$$= -[log(0.7)] \approx 0.357$$

Next Lecture

L6: Backpropagation

Algorithm to eficiently compute the gradients of a loss function with respect to the MLP weights