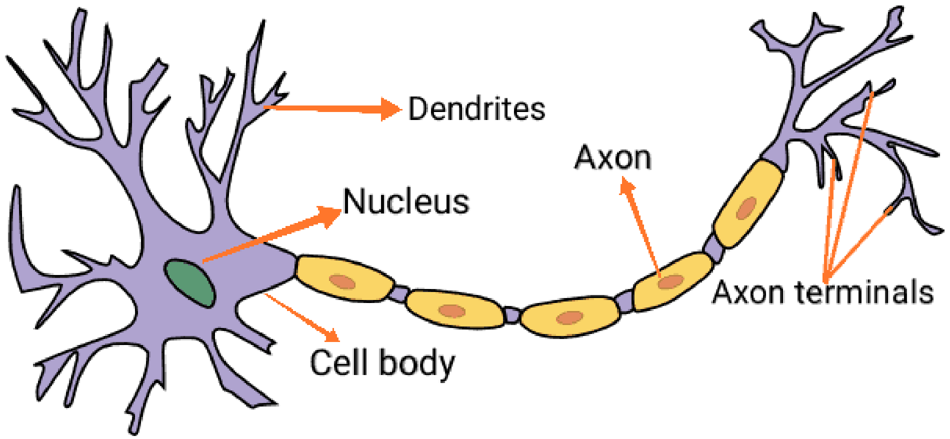
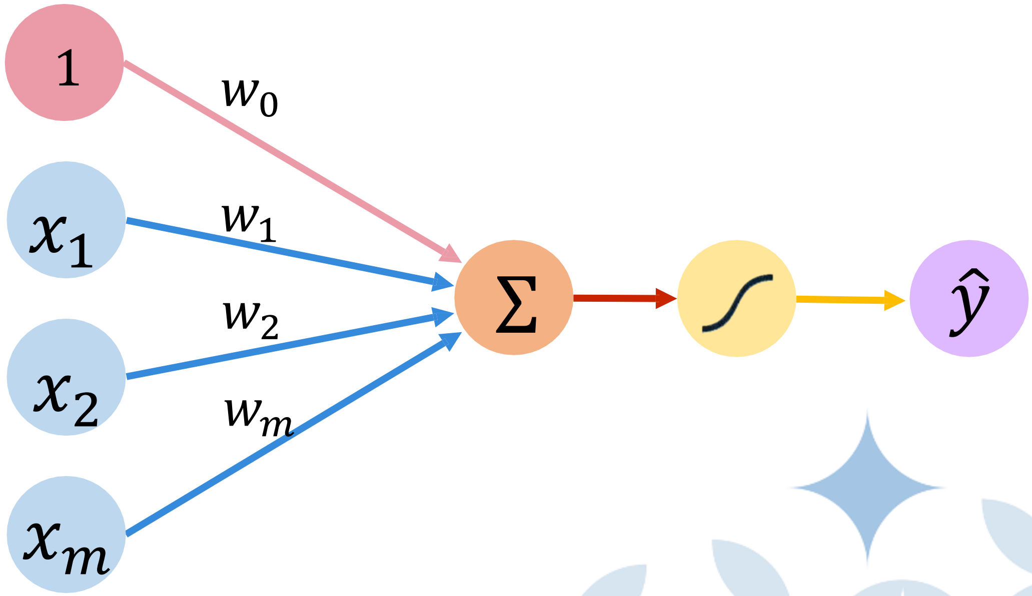
Artificial Neural Networks

Qiang Sun
University of Toronto

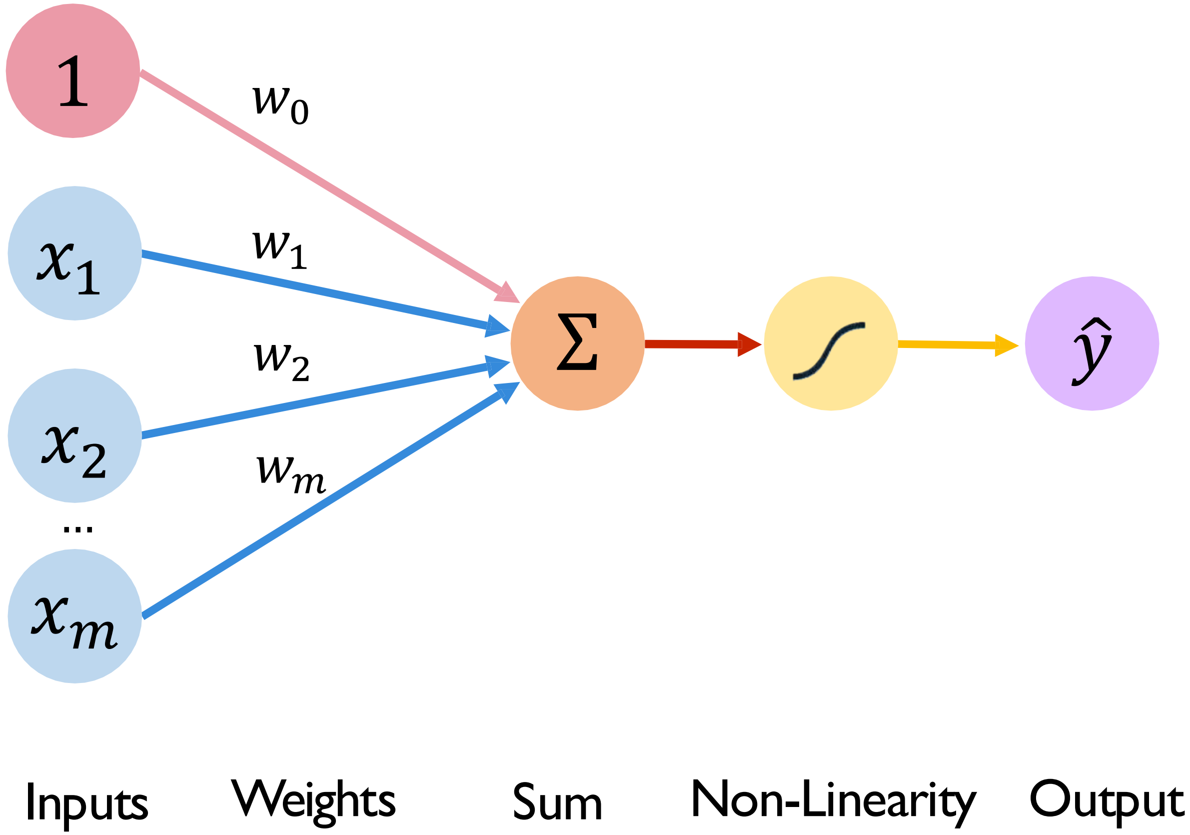
Artificial Neural Network

- A network of artificial neurons that mimics real biological neural networks
  - Each has a body, axon, and many dendrites
  - A neuron can fire or rest
  - A threshold to activate (fire)
- Perceptron (Frank Rosenblatt, 1956)
  - Fundamental building block of deep learning
  - What is Perceptron?
  - How is it defined?
  - Build deep NN from a Perceptron





The Perceptron – A single Neuron



Output

Linear combination of inputs

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

Non-linear activation function

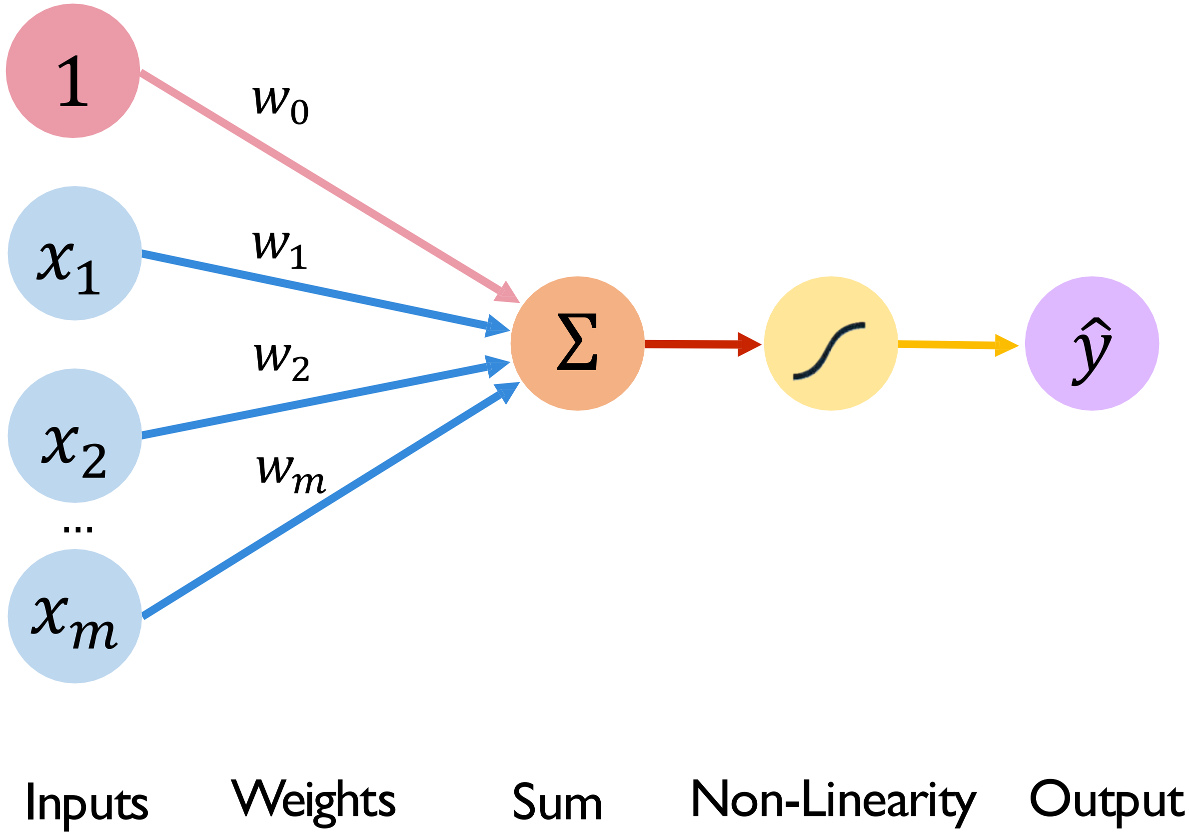
Bias

Binary Classification:

$(x_1, \dots, x_m)$ feature of a data

$\hat{y} \in [0,1]$: probability belongs to the positive class

The Perceptron

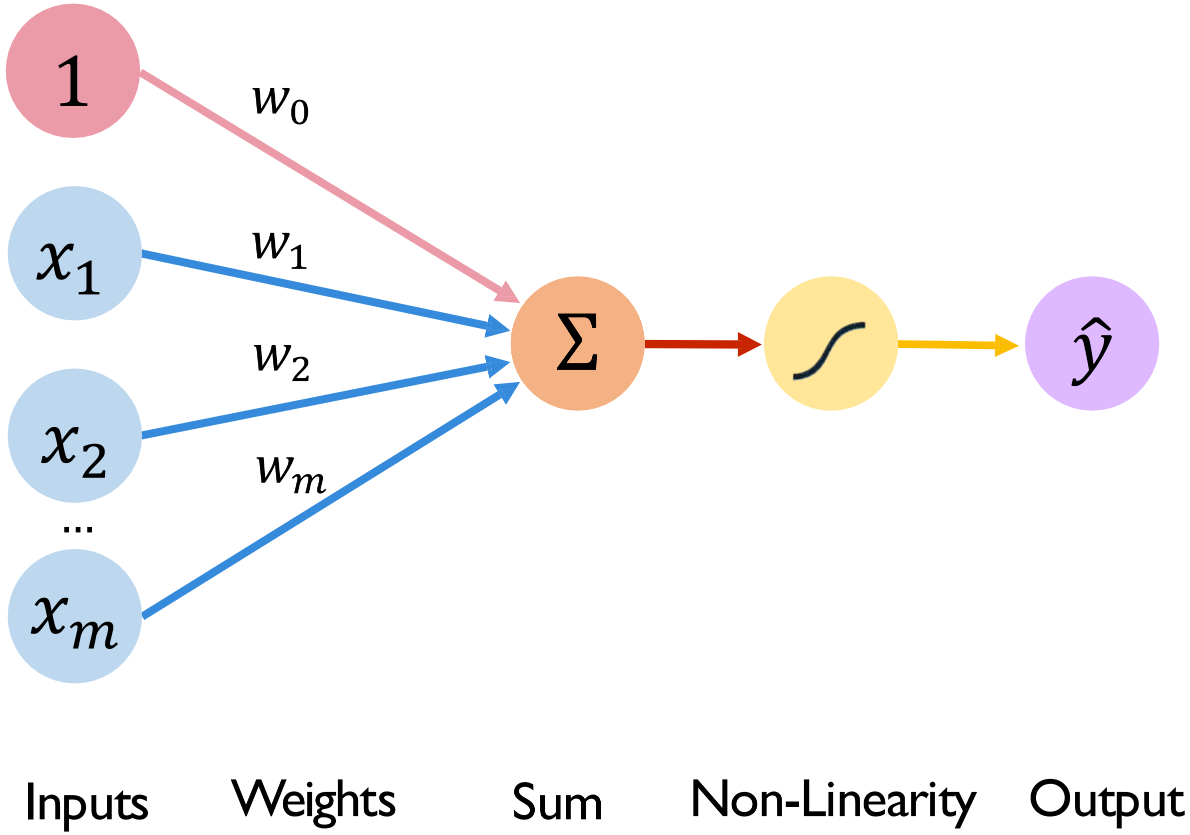


$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

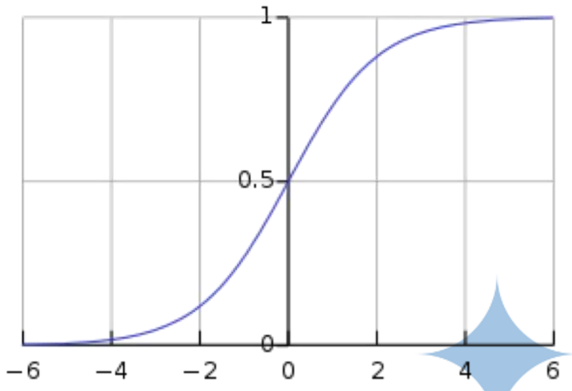
$$\hat{y} = g\left(w_0 + X^{\top} W\right)$$

$$\text{where: } \boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \text{ and } \boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

The Perceptron



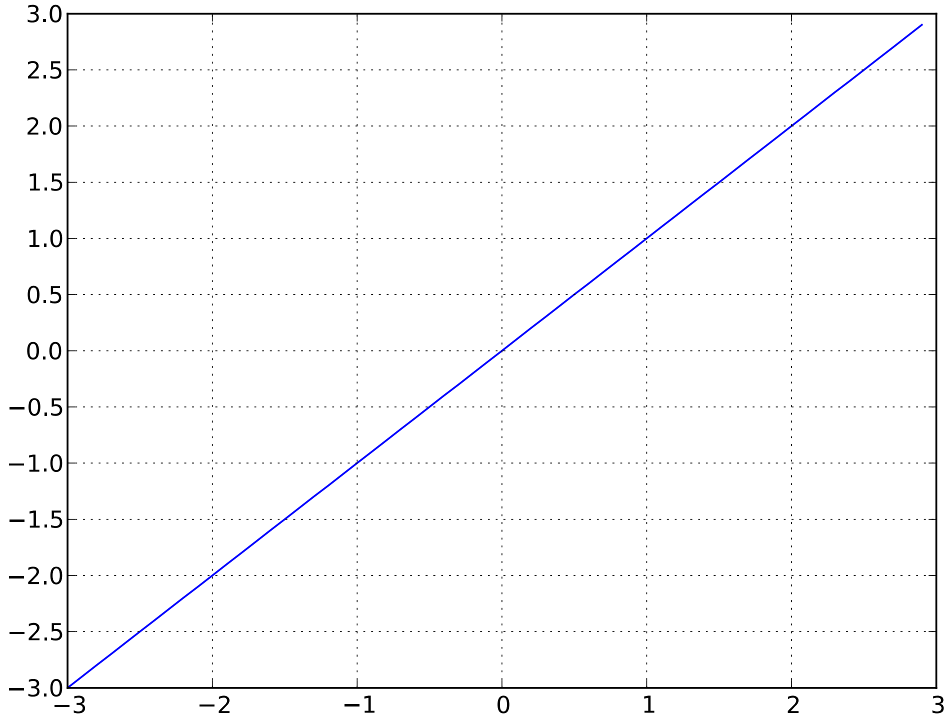
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$



Activation Functions

Linear activation function:

- $g(a) = a$
- Correspond to linear regression



Non-linear Activation Functions

Sigmoid activation function:

- Squashes the neuron's output between 0 and 1

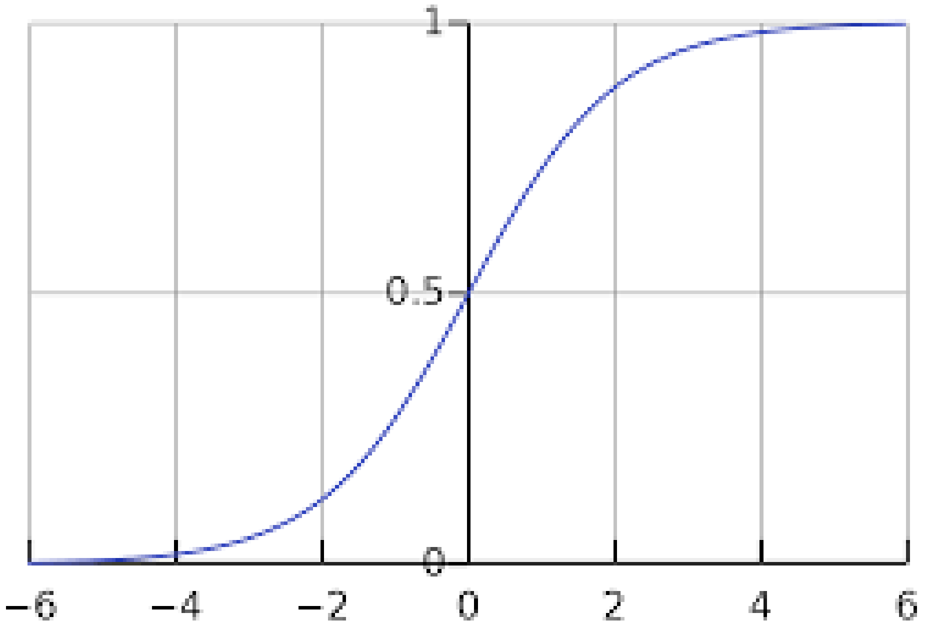
$$g(a) = \sigma(a) = \frac{\exp(a)}{1 + \exp(a)} = \frac{1}{1 + \exp(-a)}$$

- Always positive and strictly increasing

- Naturally suitable for probability

  - Logistic Regression

$$\hat{y} = P(Y = 1|X) = \frac{\exp(w_0 + \sum_i w_i X_i)}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

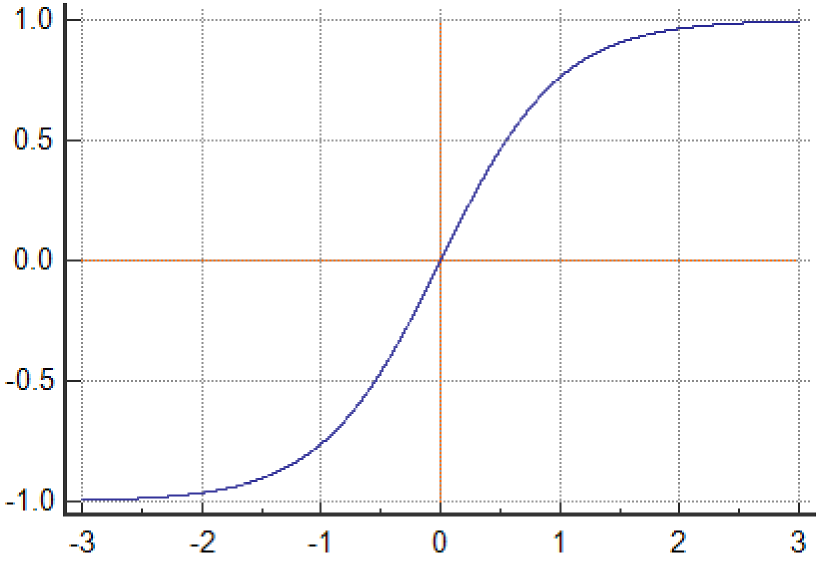


Non-linear Activation Functions

Hyperbolic tangent ("tanh") activation function:

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

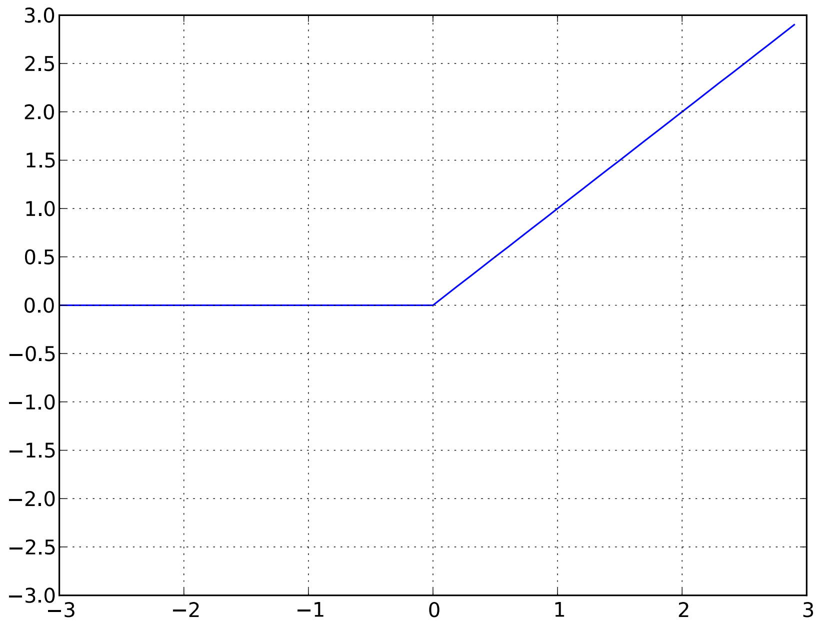
- Squashes the neuron's output between -1 and 1
- Strictly Increasing



Non-linear Activation Functions

Rectified linear Unit (ReLU), activation function:

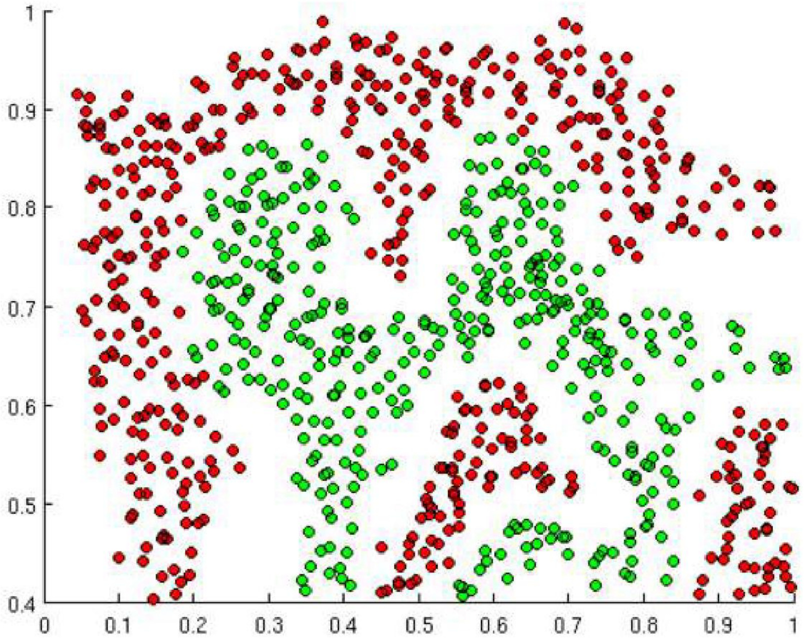
- Rectified linear Unit (ReLU), activation function:
  - Nair and Hinton (2010)
  - Bounded below by 0 (always non-negative)
  - Not upper bounded
  - Not smooth
  - other variants of ReLU (Leaky ReLU)



https://adl1995.github.io/an-overview-of-activation-functions-used-in-neural-networks.html

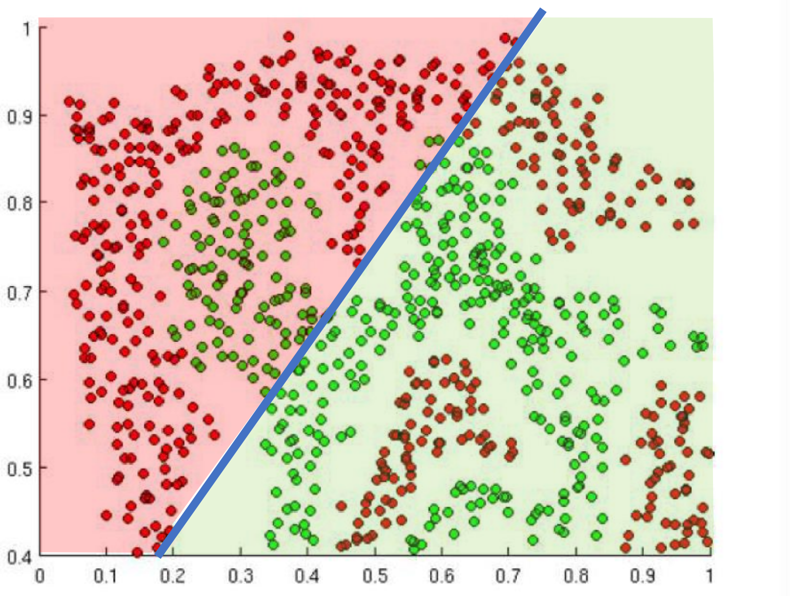
Importance of Activation Function

Non-linearity

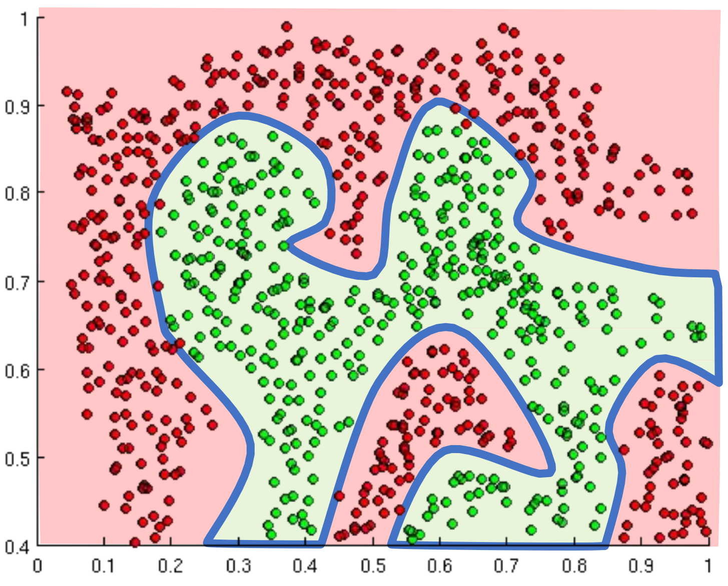


We want to build a Neural Network to distinguish green vs red points

More complex neural networks

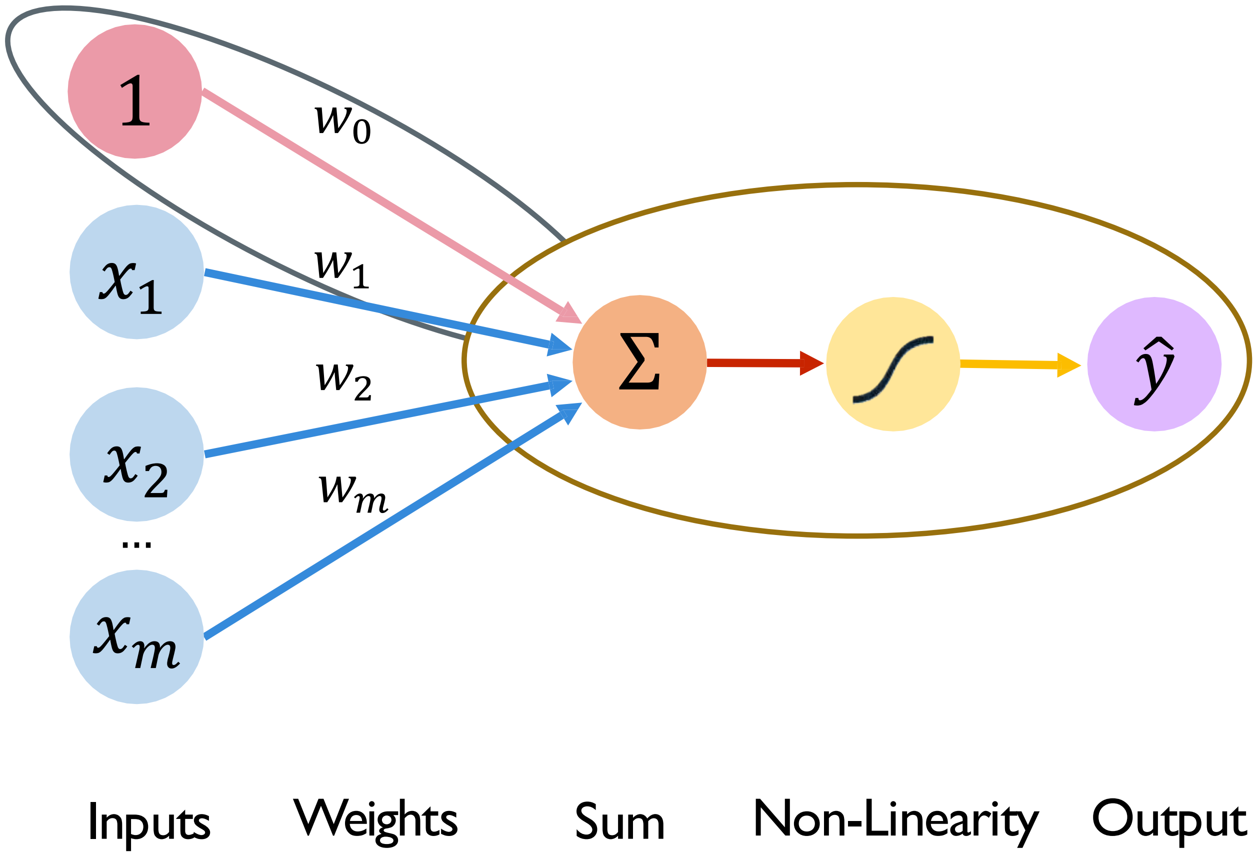


Linear activation function produce
linear decision boundaries

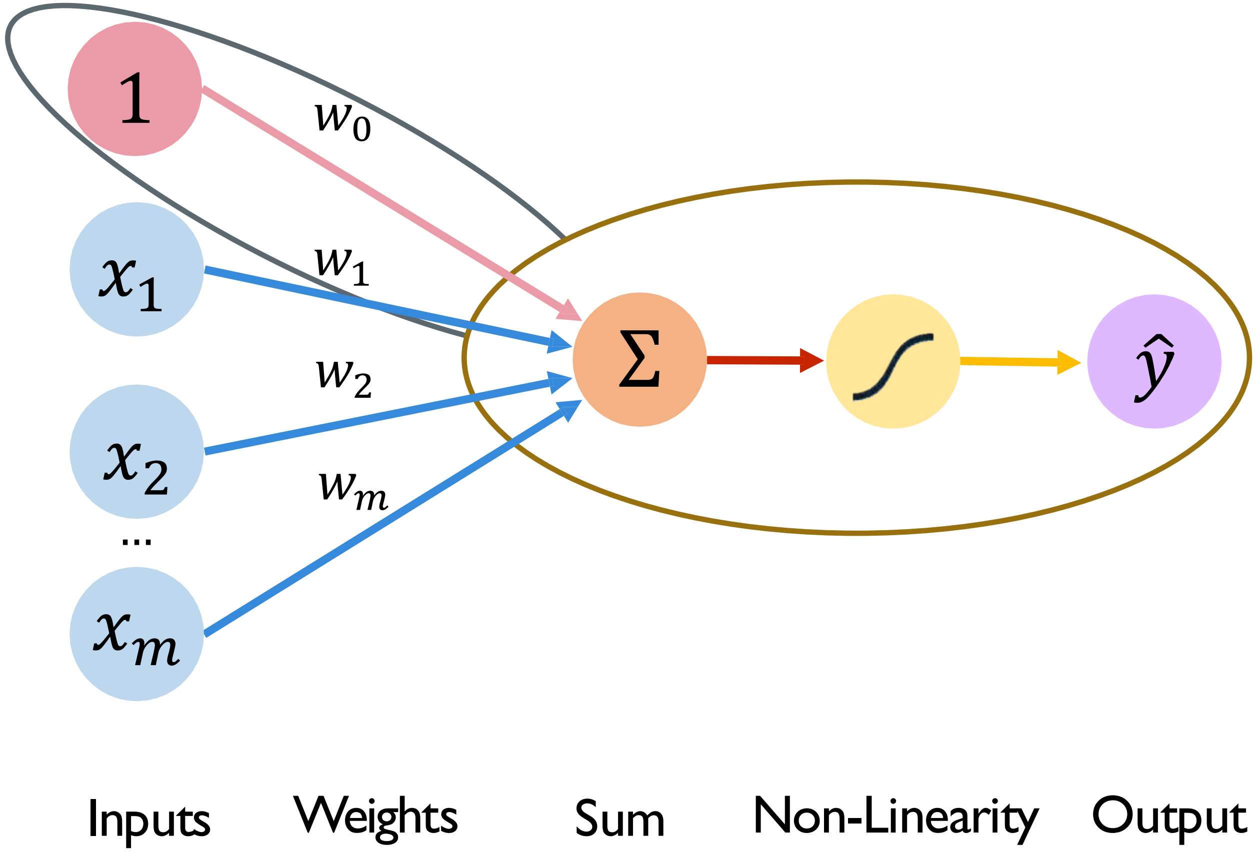


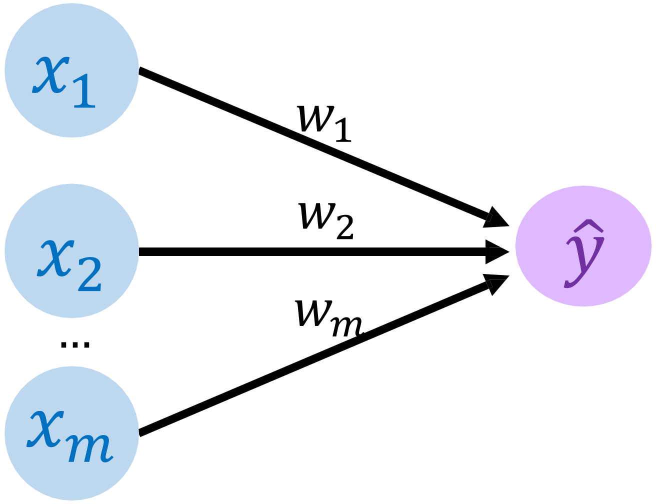
Non-linearity allows us to
approximate complex functions

The Perceptron: Simplified



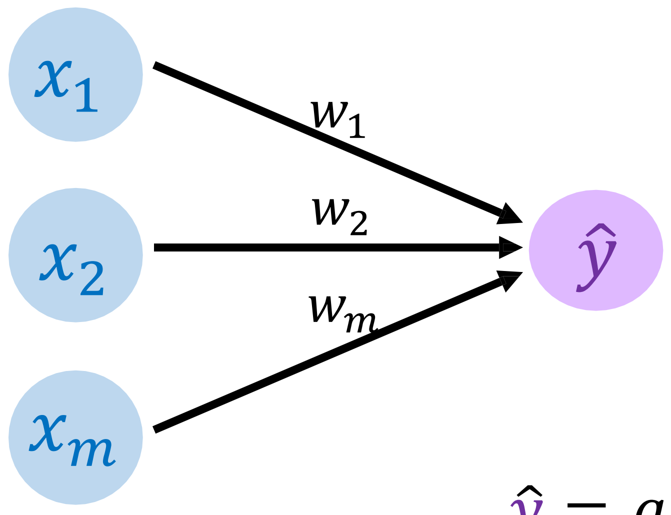
The Perceptron: Simplified





$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

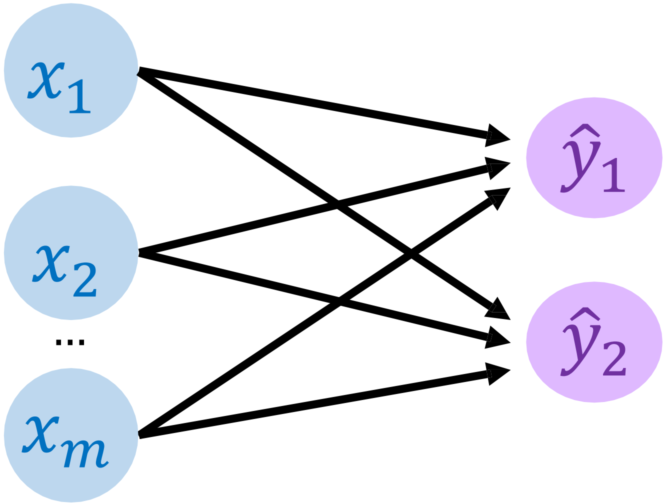
Single Output Perceptron



$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

What if we have multi-output $\hat{y}_1$ and $\hat{y}_2$?

Multi Output Perceptron

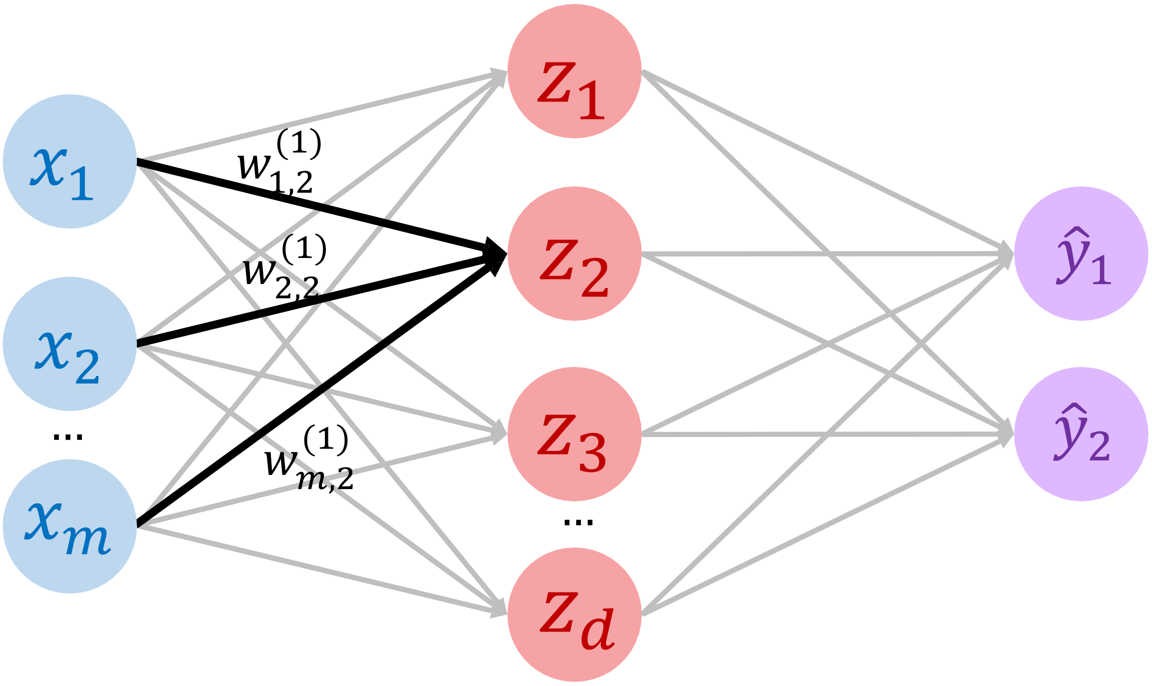


$$= g\left(w_{0,1} + \sum_{i=1}^{m} x_i w_{i,1}\right)$$

$$= g\left(w_{0,2} + \sum_{i=1}^{m} x_i w_{i,2}\right)$$

Output $\hat{y}_j = g\left(w_{0,1} + \sum_{i=1}^{m} x_i w_{i,j}\right)$

Extend to Single (Hidden) Layer Neural Network

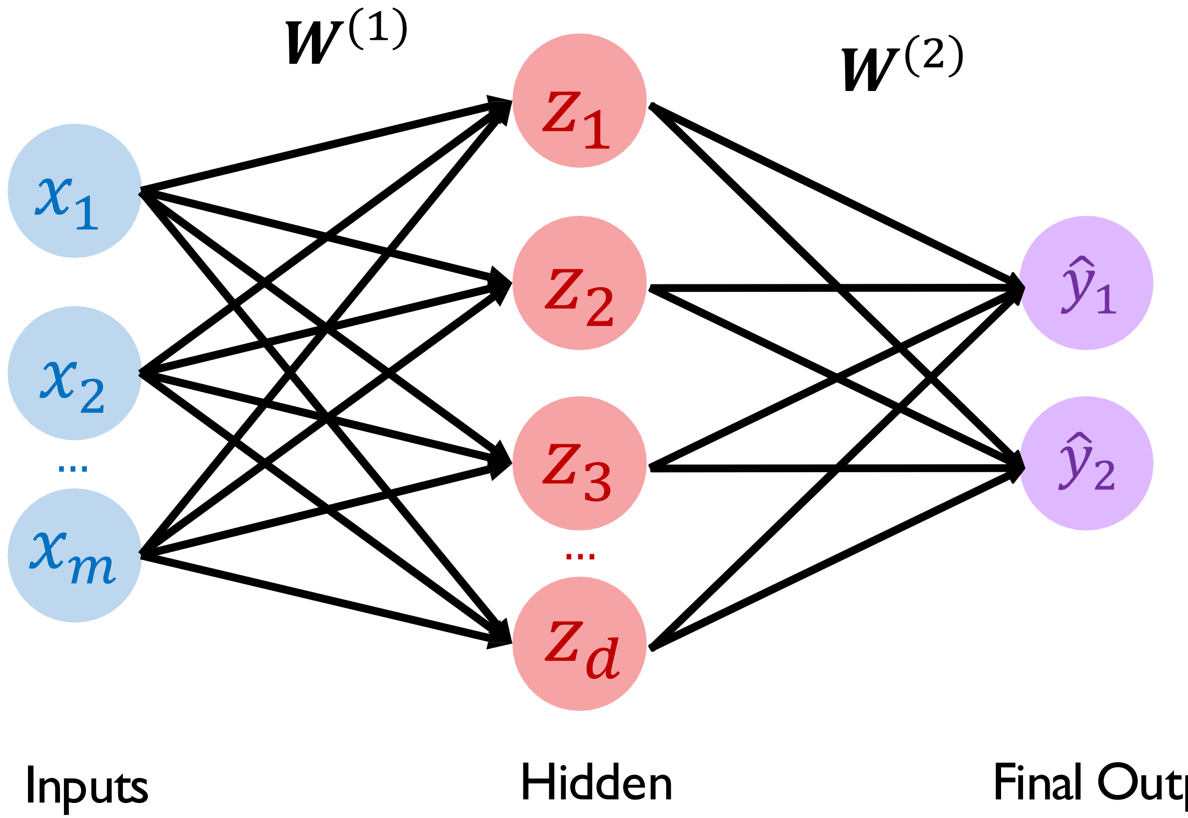


$$z_2 = g(w_{0,2}^{(1)} + \sum_{i=1}^{m} x_i w_{i,2}^{(1)})$$

$$= g(w_{0,2}^{(1)} + x_1 w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + \cdots + x_m w_{m,2}^{(1)})$$

(1) Indicates the 1st layer

Single (Hidden) Layer Neural Network



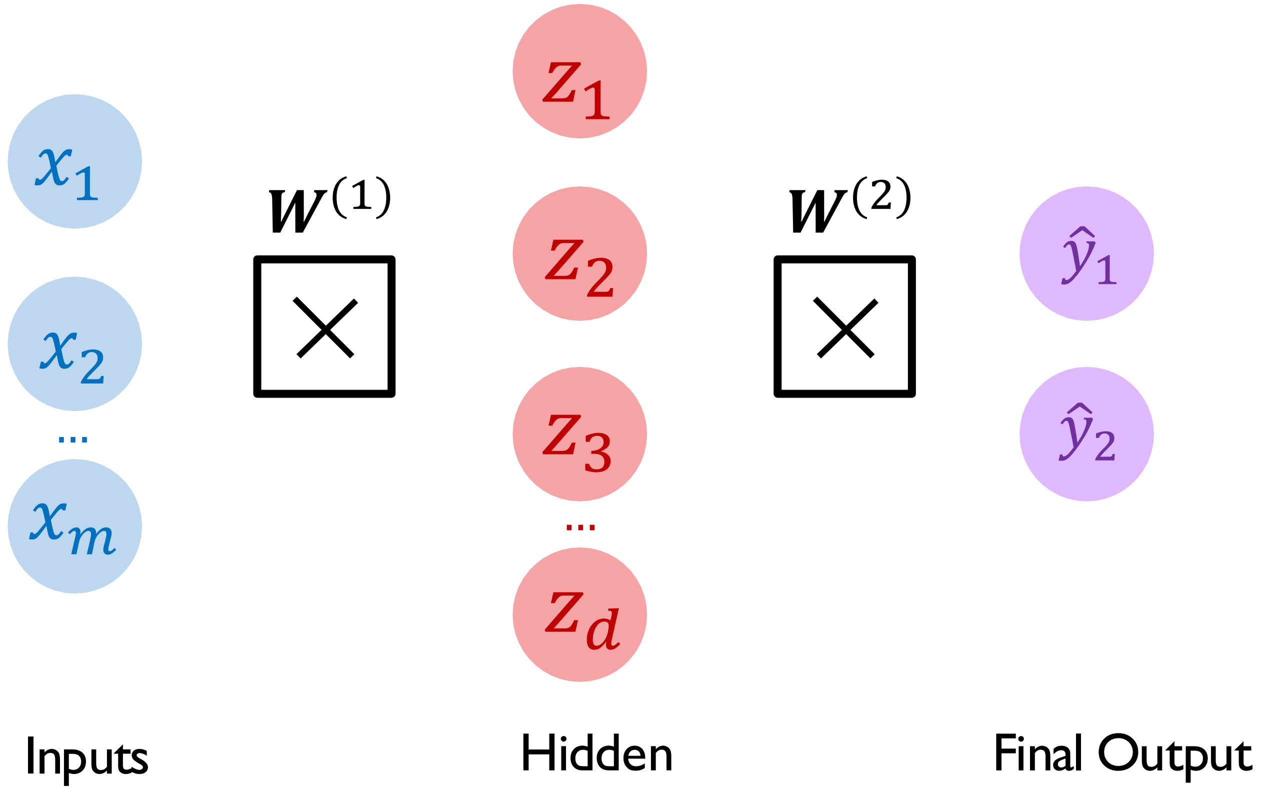
Parameters to be learned

$$W = (W^{(1)}, W^{(2)})$$

$$z_j = g(w_{0,j}^{(1)} + \sum_{i=1}^{m} x_i w_{i,j}^{(1)})$$

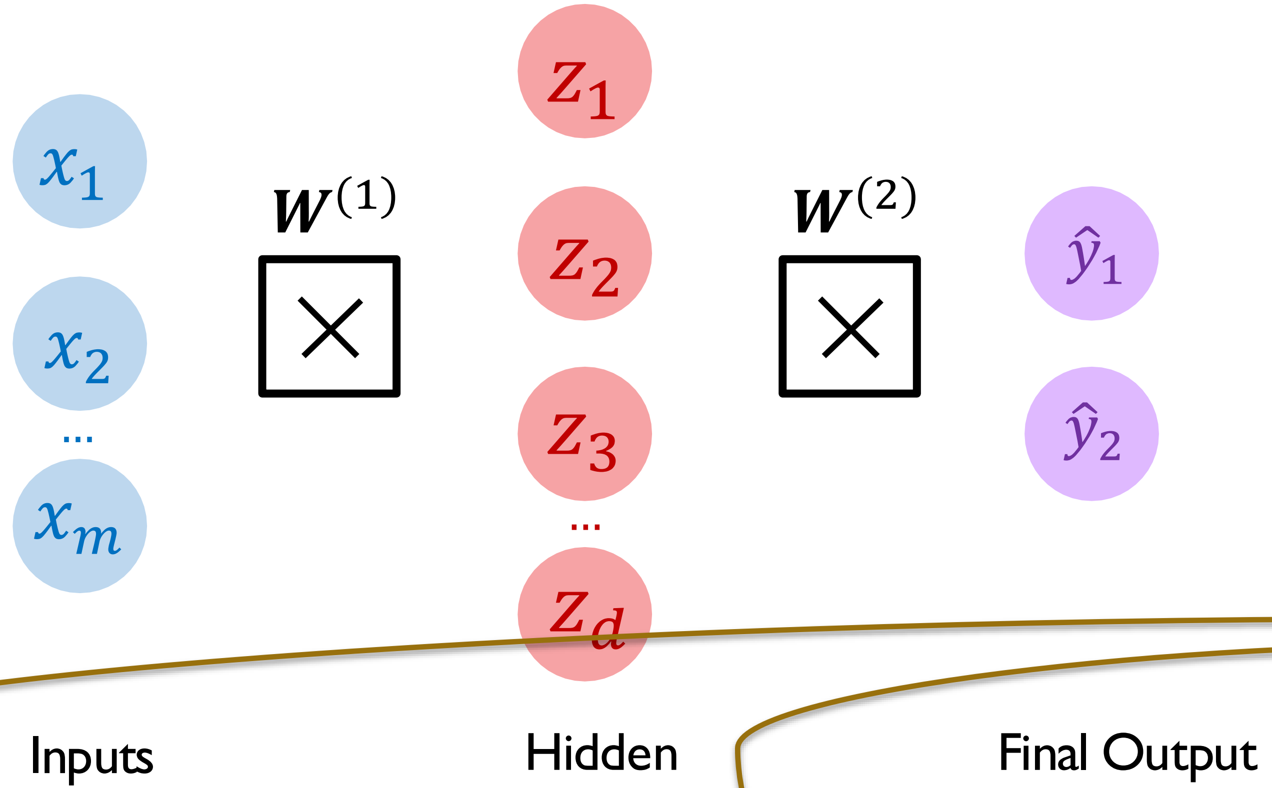
$$\hat{y}_k = g(w_{0,k}^{(2)} + \sum_{j=1}^{d} z_j w_{j,k}^{(2)})$$

Single (Hidden) Layer Neural Network



$$z_j = g\left(w_{0,j}^{(1)} + \sum_{i=1}^{m} x_i w_{i,j}^{(1)}\right) \qquad \hat{y}_k = g\left(w_{0,k}^{(2)} + \sum_{j=1}^{d} z_j w_{j,k}^{(2)}\right)$$

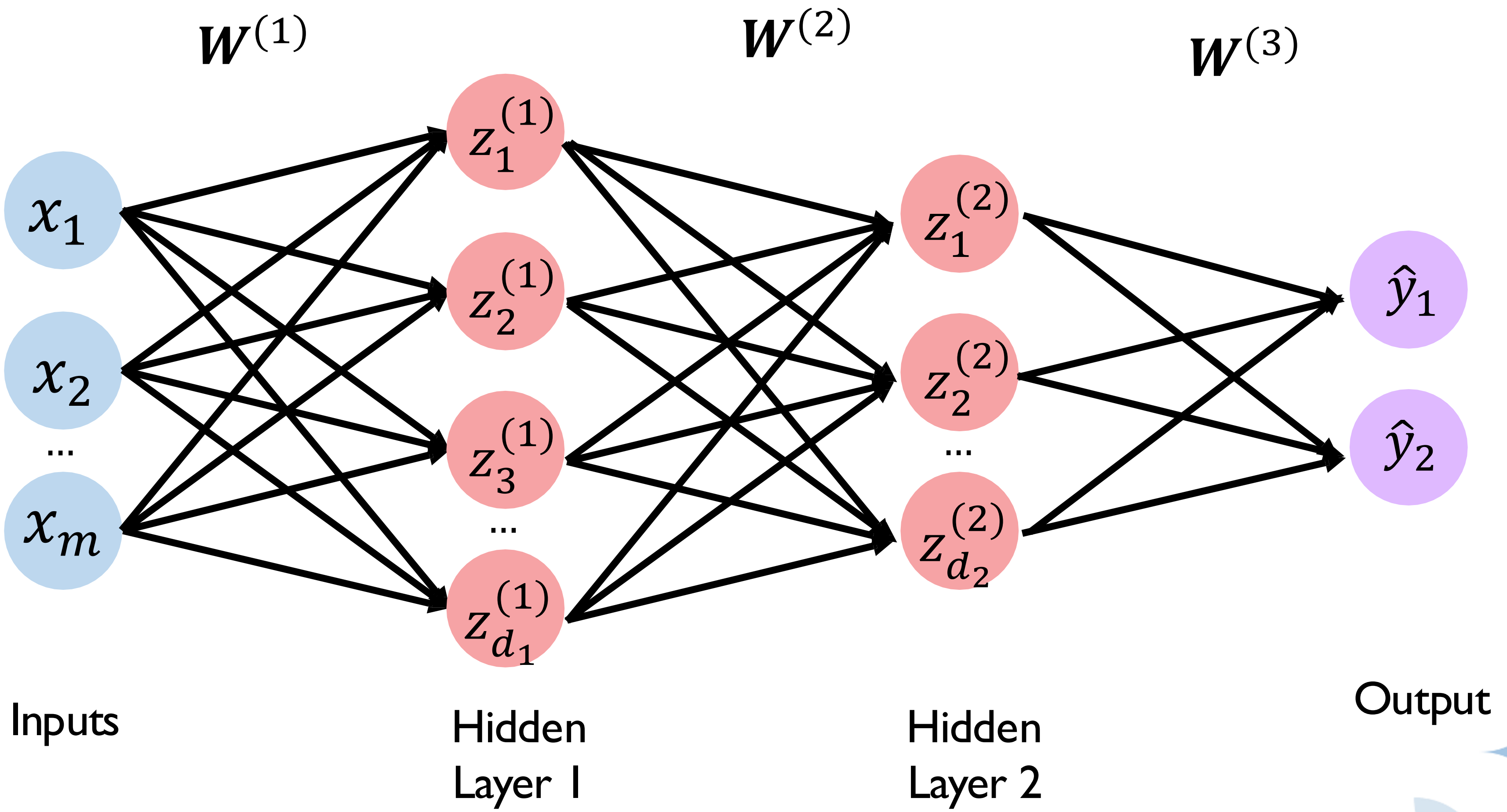
Single (Hidden) Layer Neural Network



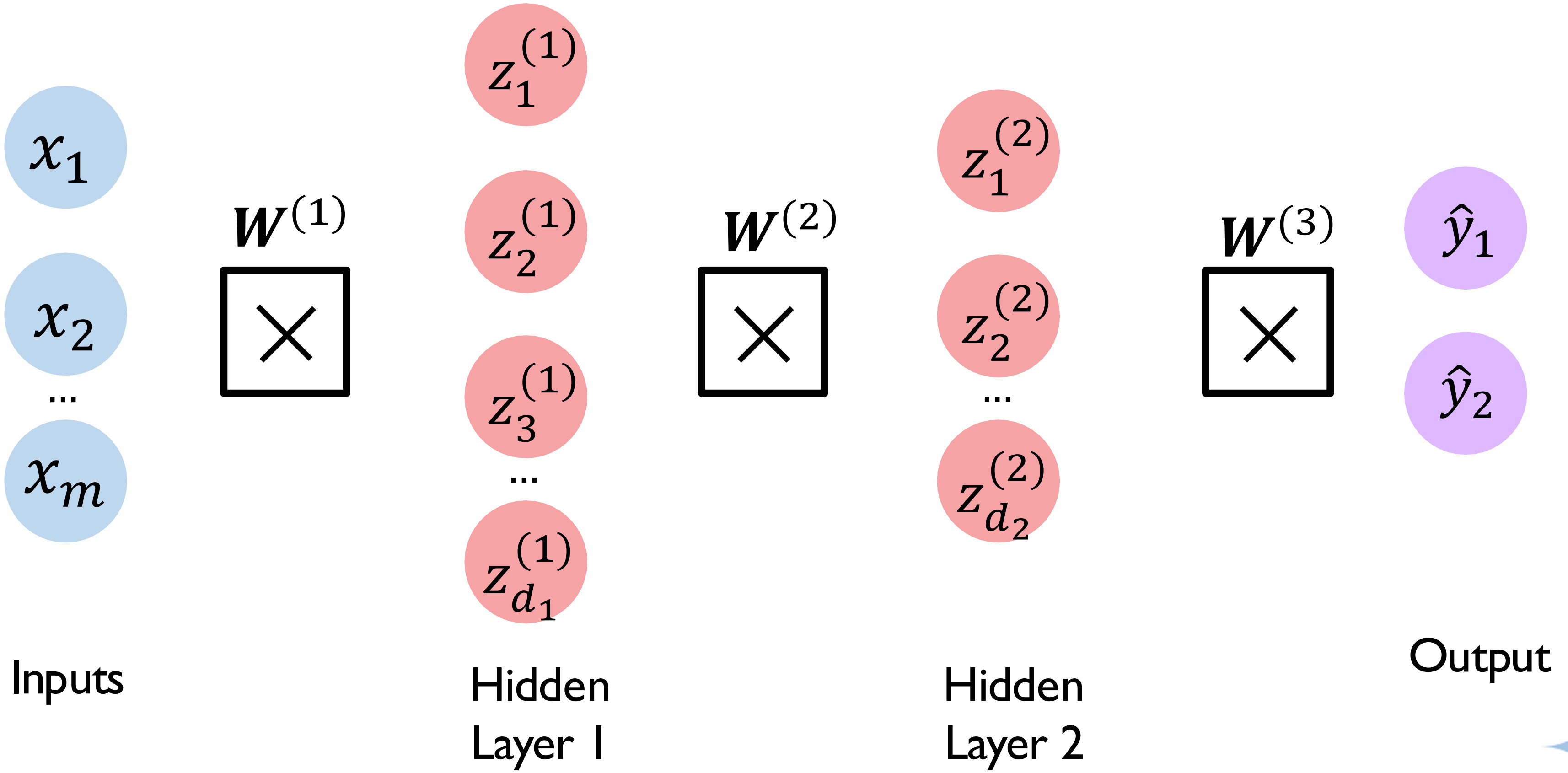
Activation functions
can be different
in different layers

$$z_j = g\left(w_{0,j}^{(1)} + \sum_{i=1}^{m} x_i w_{i,j}^{(1)}\right) \qquad \hat{y}_k = g\left(w_{0,k}^{(2)} + \sum_{j=1}^{d} z_j w_{j,k}^{(2)}\right)$$

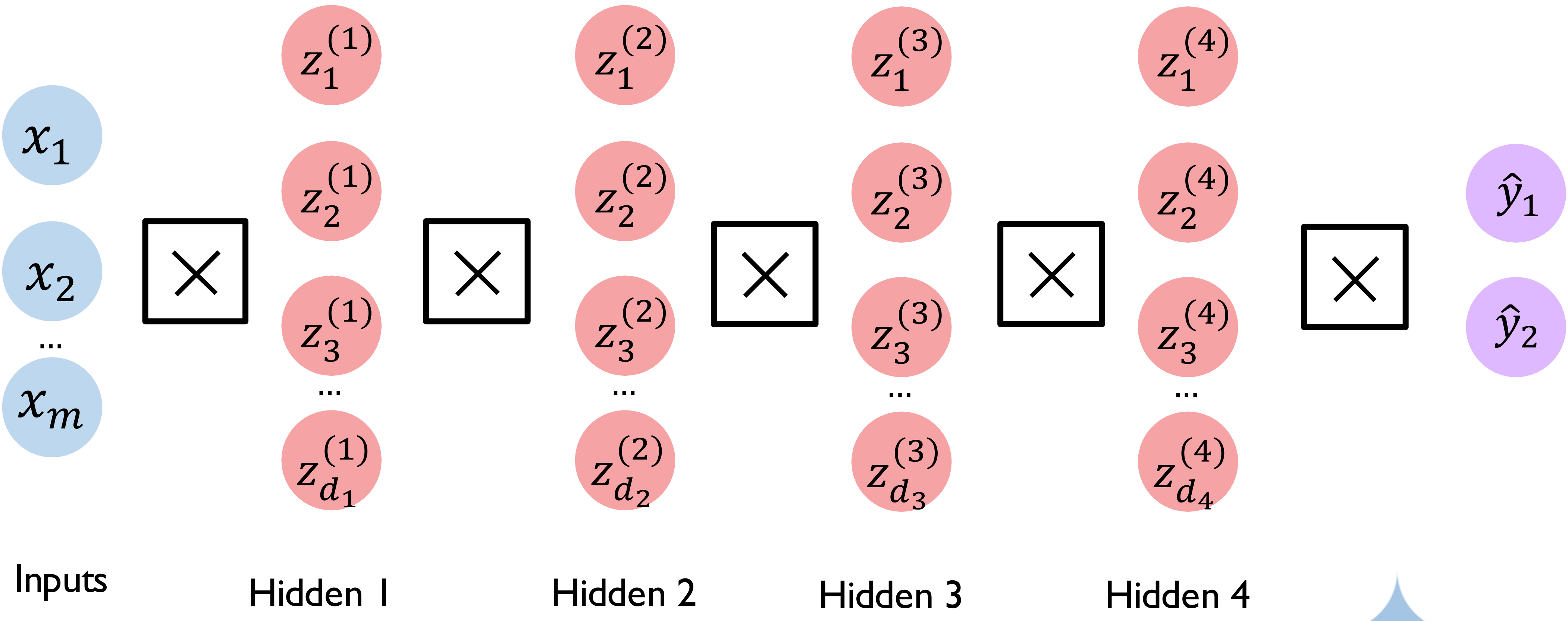
Two-Layer Neural Network



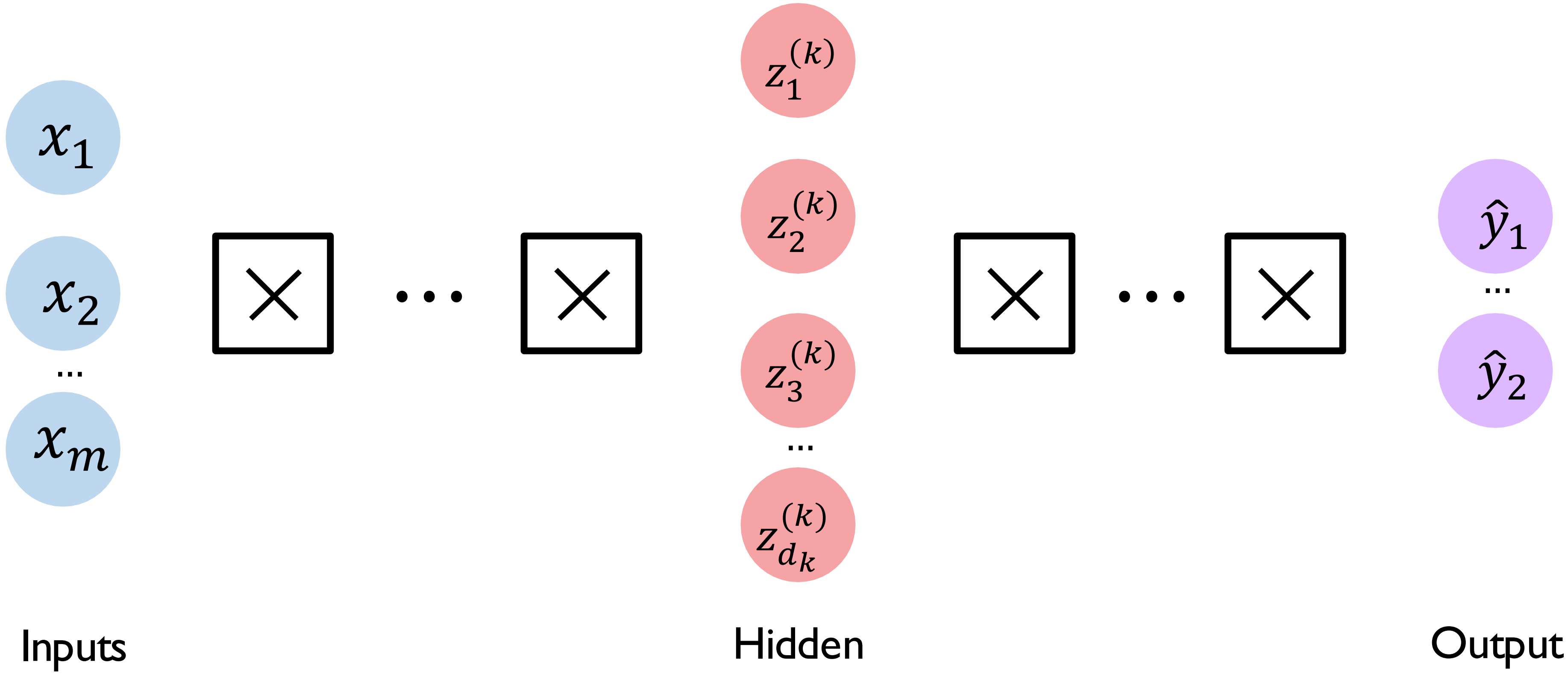
Two-Layer Neural Network



Multi-Layer (Deep) Neural Network

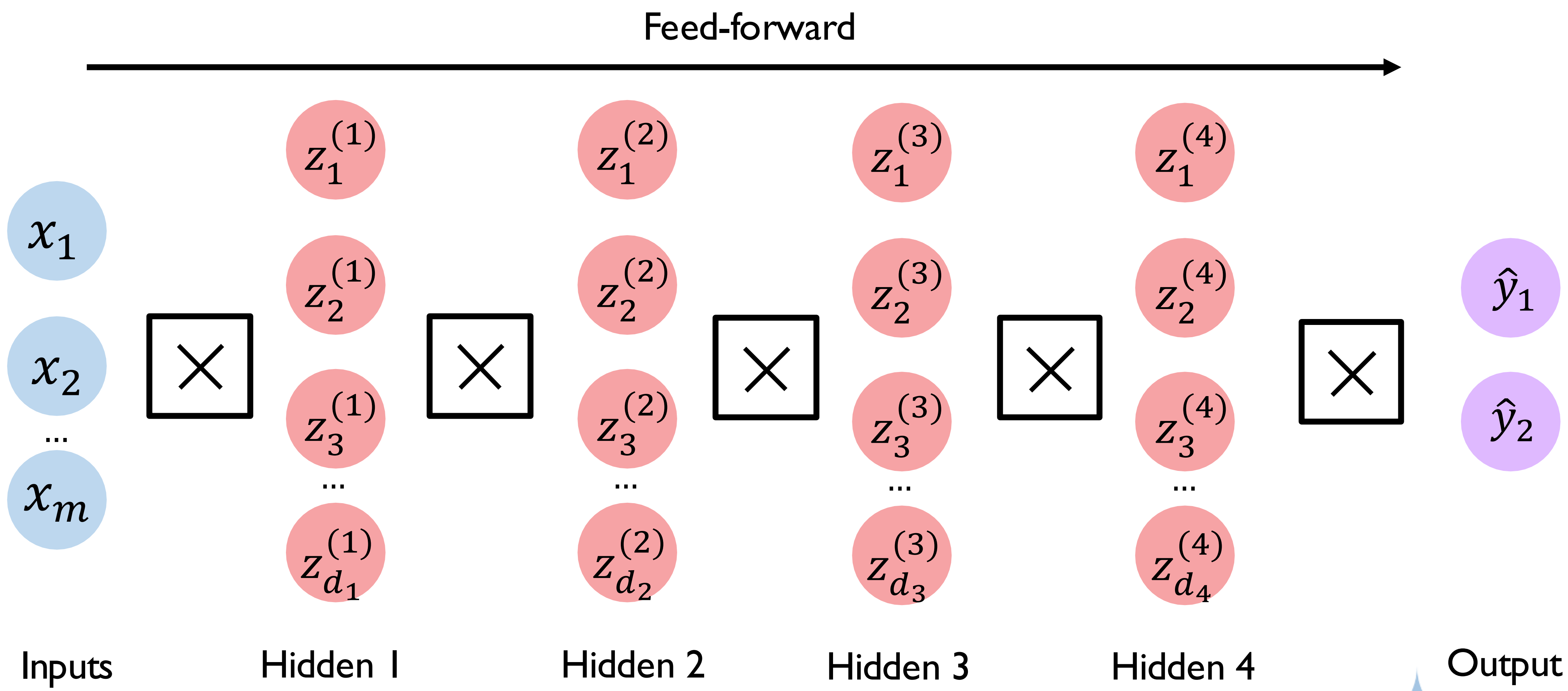


Deep Neural Network (DNN)



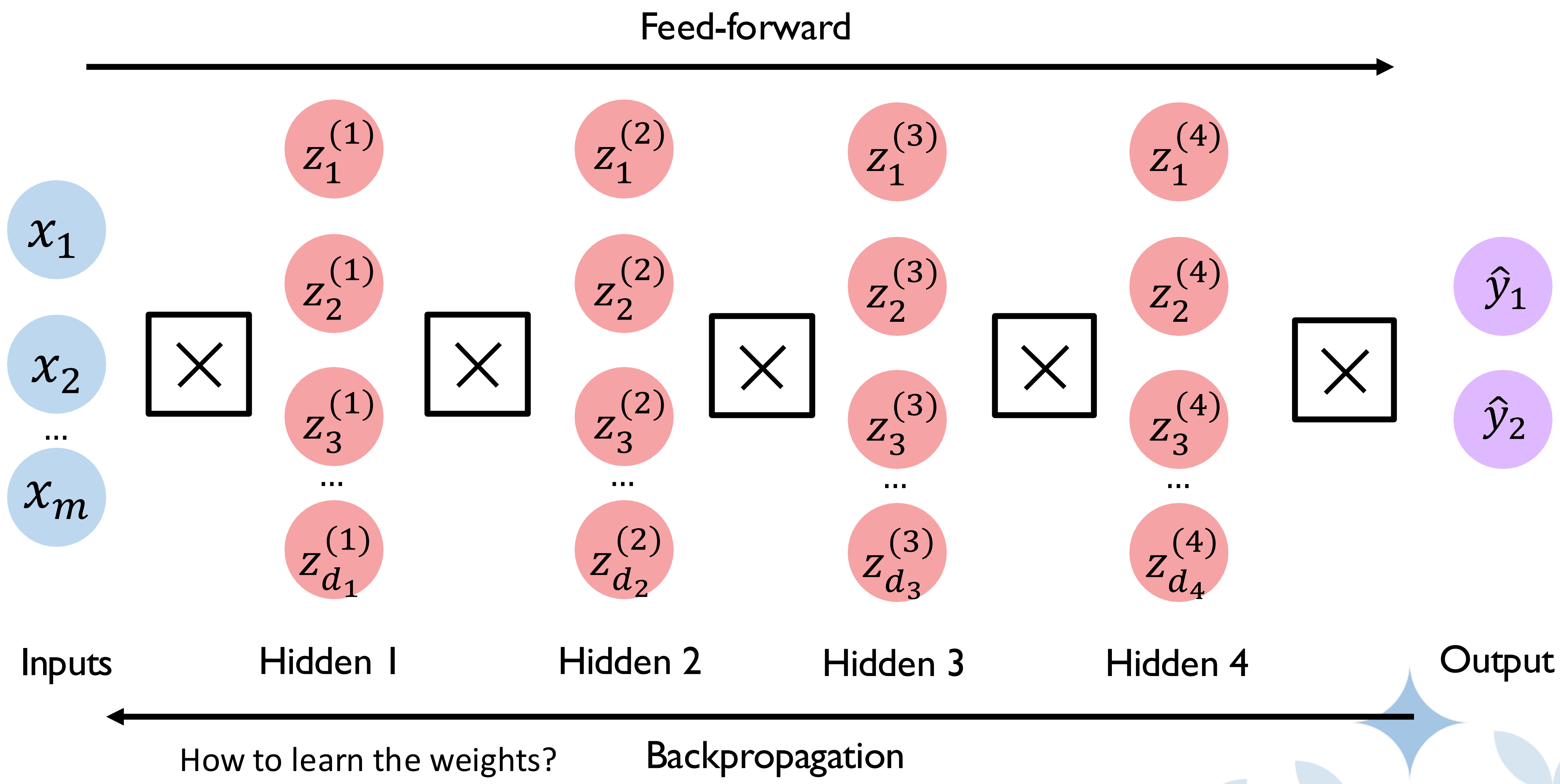
$$z_j^{(k)} = g(w_{0,j}^{(k)} + \sum_{i=1}^{d_{k-1}} z_i^{(k-1)} w_{i,j}^{(k)})$$

Feed-forward Networks and Backpropagation



How to learn the weights?

Feed-forward Networks and Backpropagation



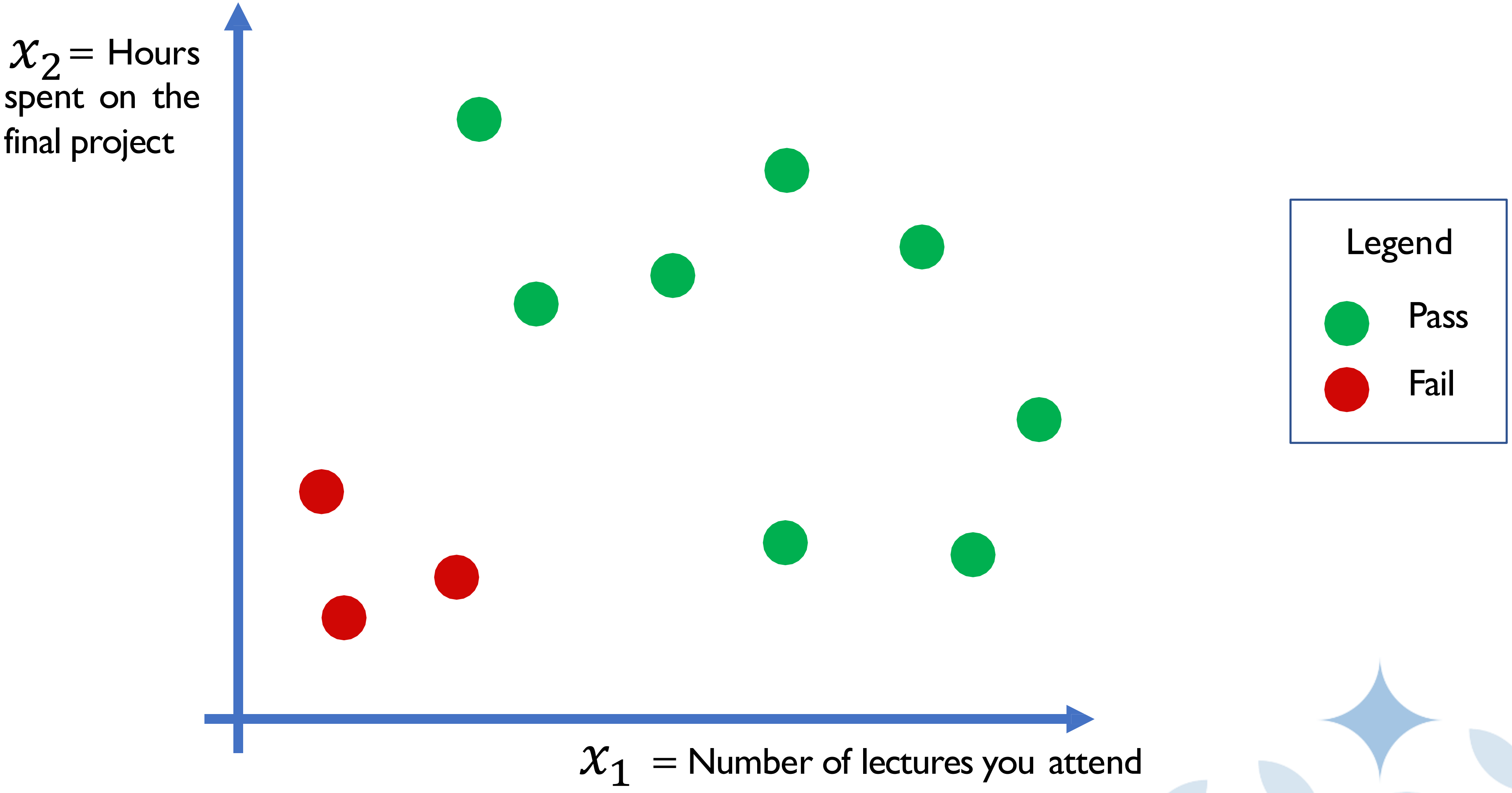
Example Problem

Will I pass the class?

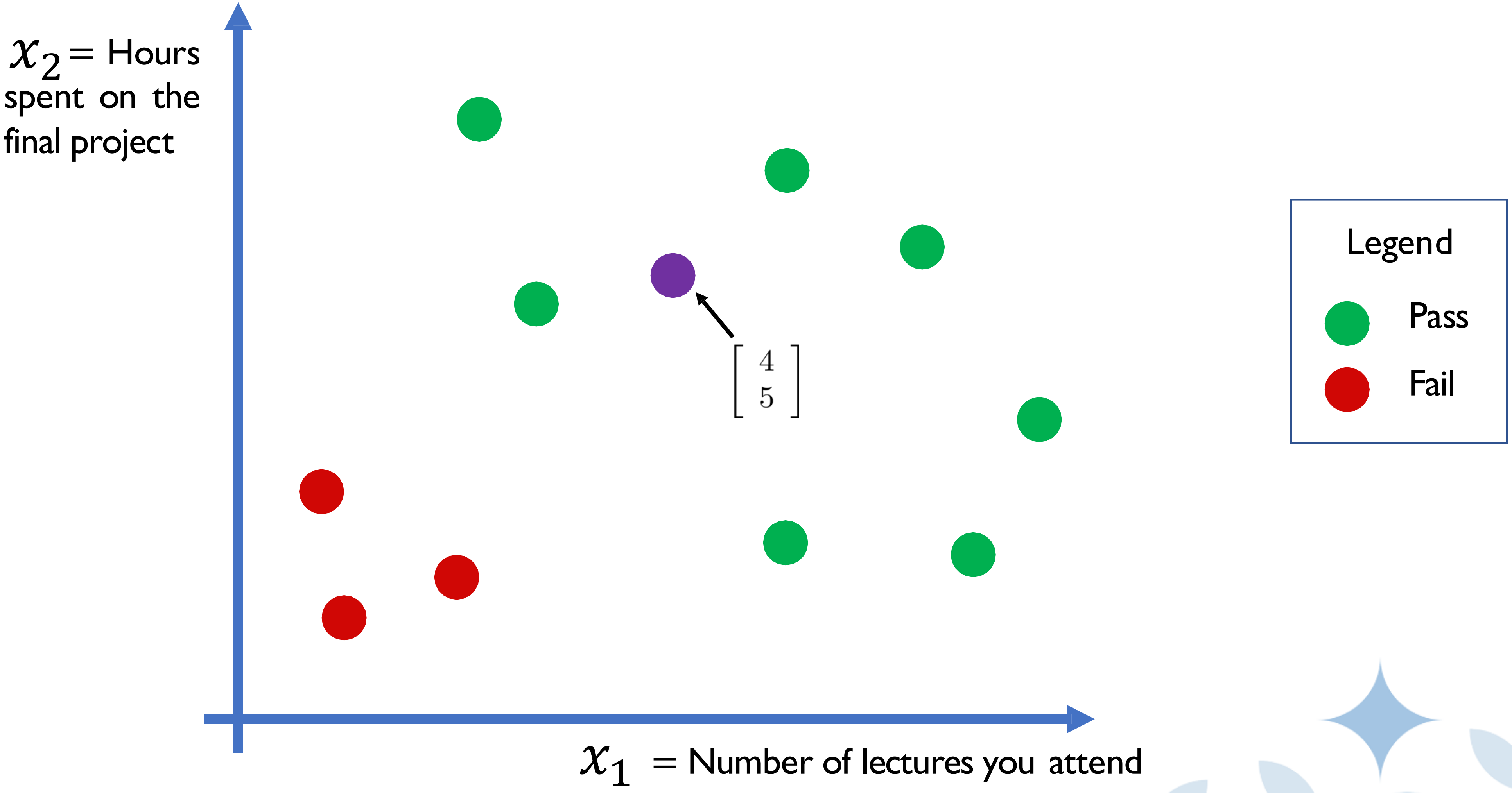
– Let's start with a simple two-feature model

- x_1 = Number of lectures you attend
- x_2 = Hours spent on the final project

Example Problem: Will I pass the class?



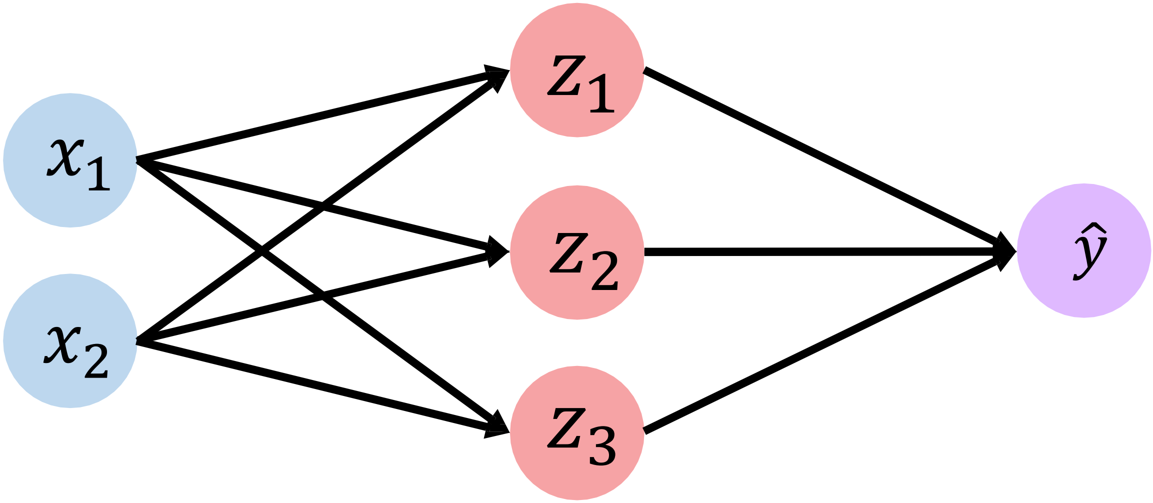
Example Problem: Will I pass the class?



Initialization

Initialize the network to realize the first feed-forward pass

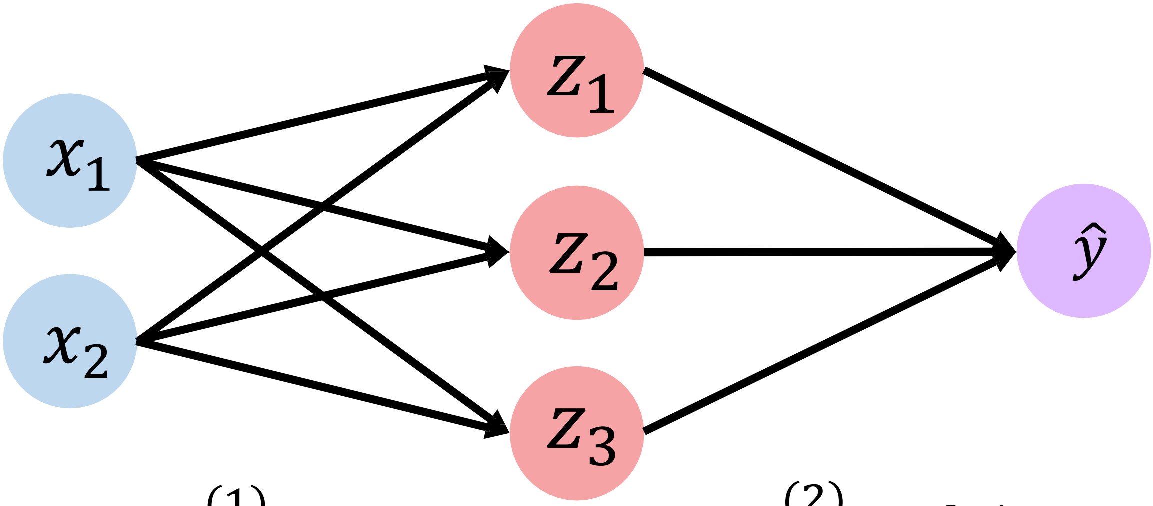
- Step 1: Initialize $W = (W^{(1)}, W^{(2)})$



Initialization

Initialize the network for the first feed-forward pass

- Step 1: Initialize $W = (W^{(1)}, W^{(2)})$



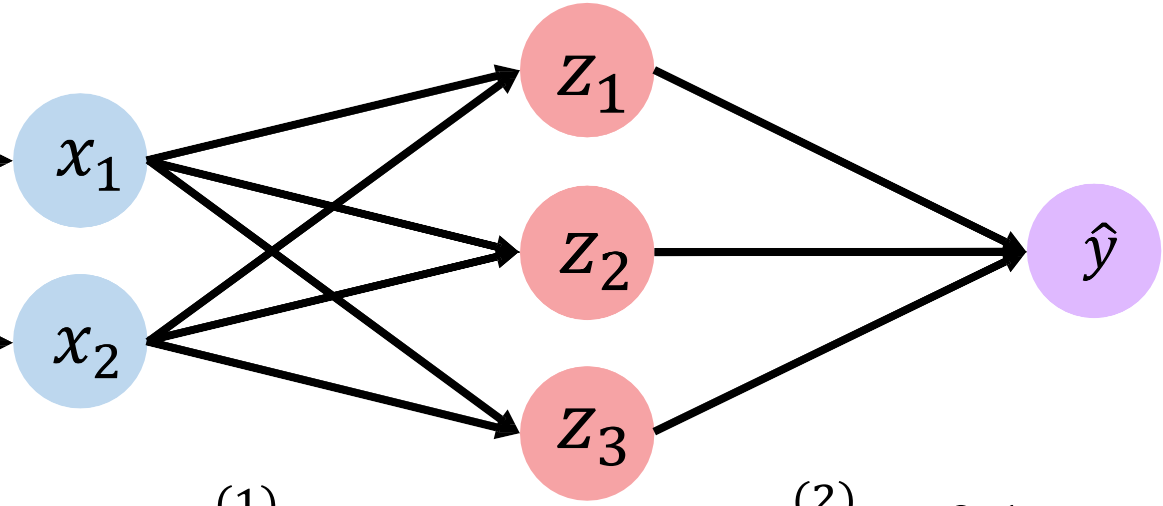
$w_{i,j}^{(1)} = 0.1$ $w_{i,j}^{(2)} = 0.1$

  – For example, set all weights to 0.1

Feed-forward

Data Point:

$x^{[1]} = [4\,,5\,]$



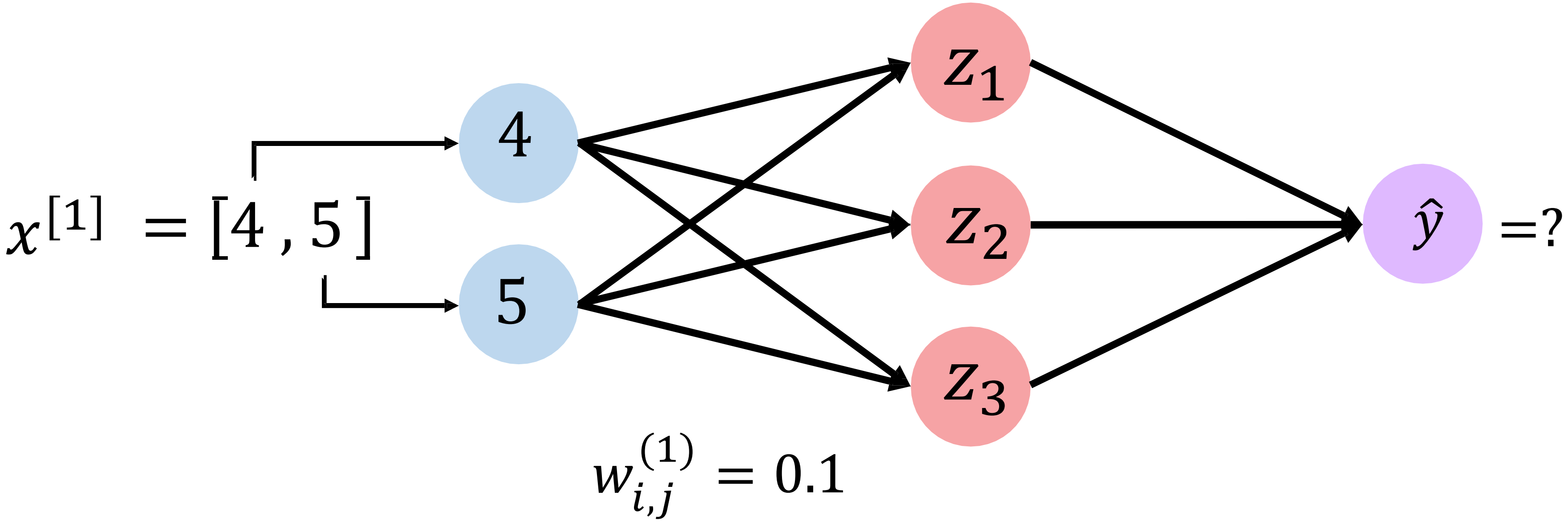
$=?$

$w_{i,j}^{(1)} = 0.1$

$w_{i,j}^{(2)} = 0.1$

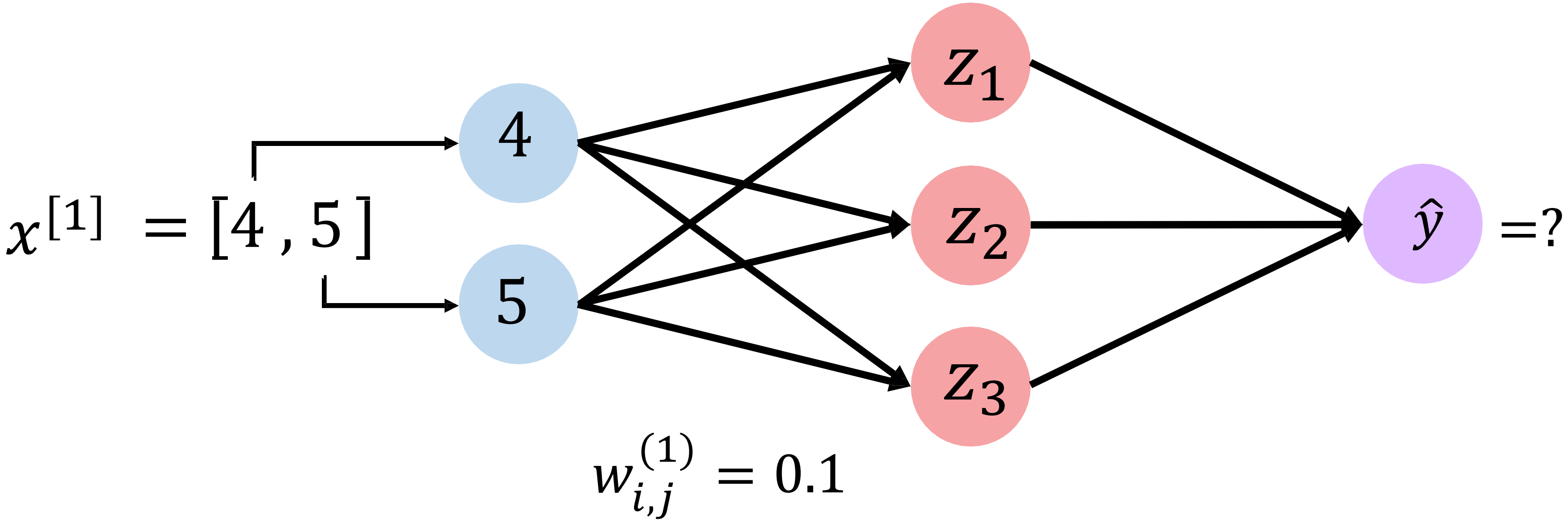
Feed-forward

$$z_1 = g\left(w_{0,1}^{(0)} + x_1 w_{1,1}^{(1)} + x_2 w_{2,1}^{(1)}\right)$$



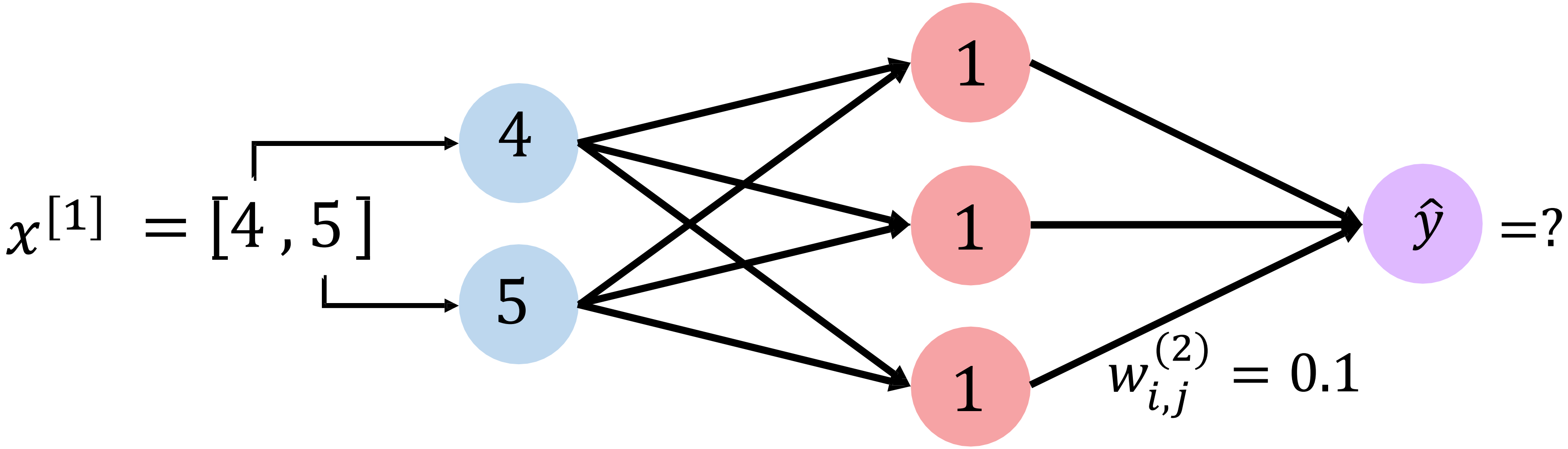
Feed-forward

$$z_1 = g\left(w_{0,1}^{(0)} + x_1 w_{1,1}^{(1)} + x_2 w_{2,1}^{(1)}\right)$$
$$= \mathrm{ReLU}(0.1 + 4 \times 0.1 + 5 \times 0.1) = \mathrm{ReLU}(1) = 1.$$



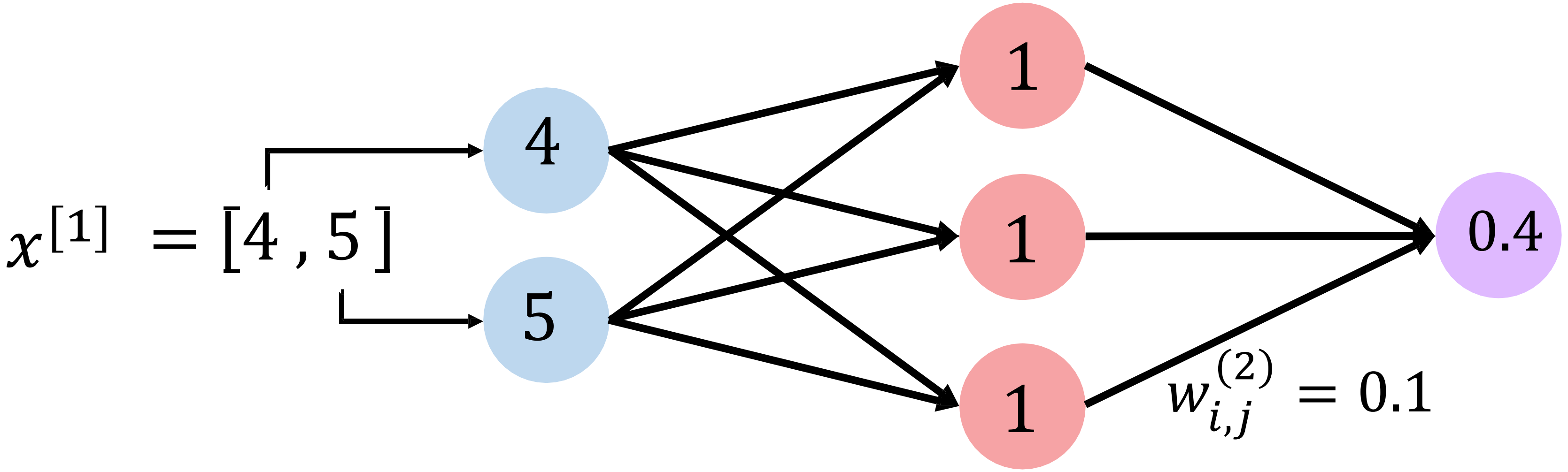
Feed-forward

$$z_1 = g\left(w_{0,1}^{(0)} + x_1 w_{1,1}^{(1)} + x_2 w_{2,1}^{(1)}\right)$$
$$= \mathrm{ReLU}(0.1 + 4 \times 0.1 + 5 \times 0.1) = \mathrm{ReLU}(1) = 1.$$

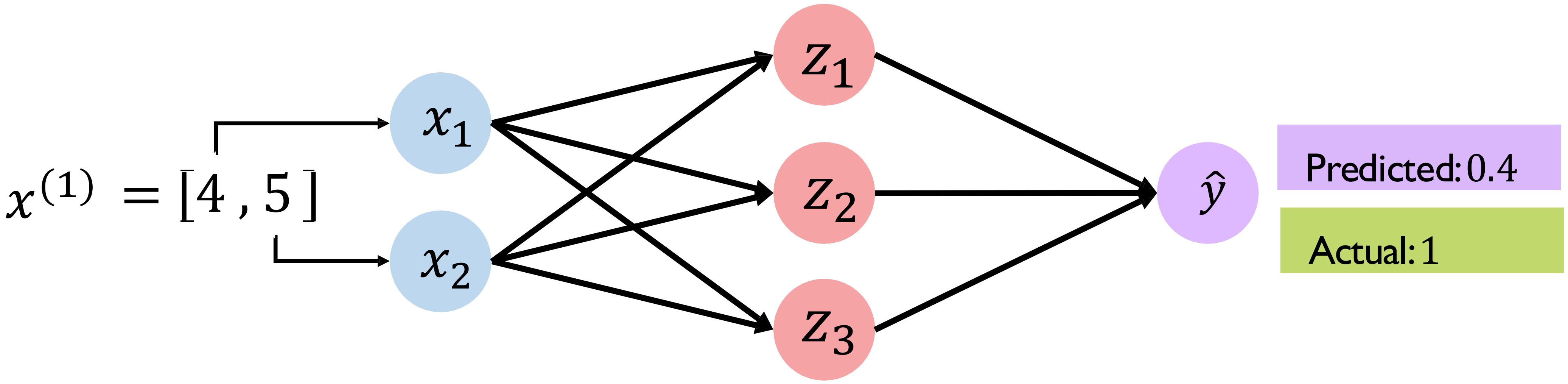


$$\hat{y} = g\left(w_{0,1}^{(2)} + z_1 w_{1,1}^{(2)} + z_2 w_{2,1}^{(2)} + z_3 w_{3,1}^{(2)}\right)$$
$$= \mathrm{ReLU}(0.4) = 0.4$$

Feed-forward

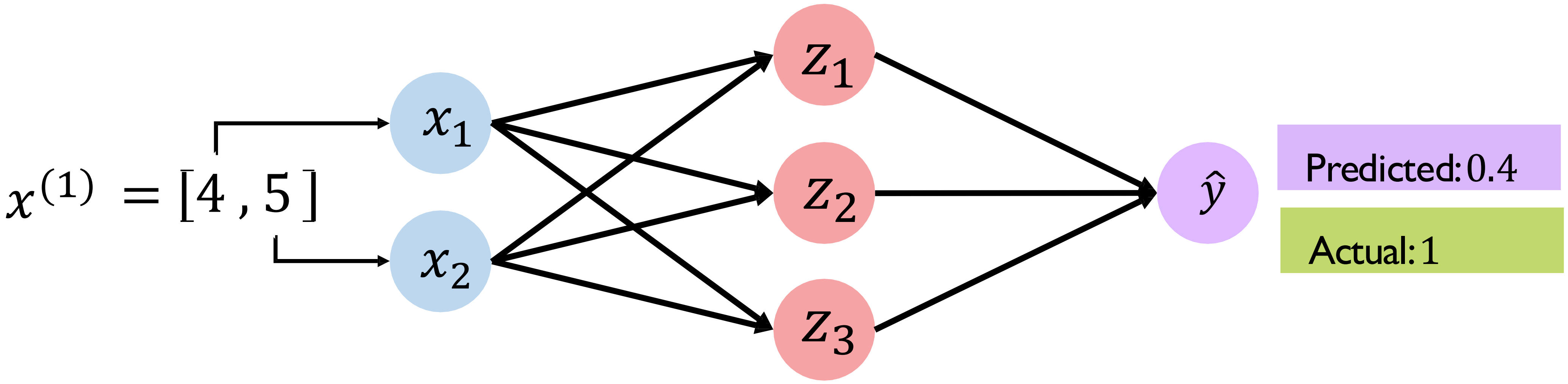


Model Assessment - Quantifying Loss



$$loss(\underbrace{f(X^{[1]}; W)}_{\text{Predicted}}, \underbrace{Y^{[1]}}_{\text{Actual}})$$

Model Assessment - Quantifying Loss

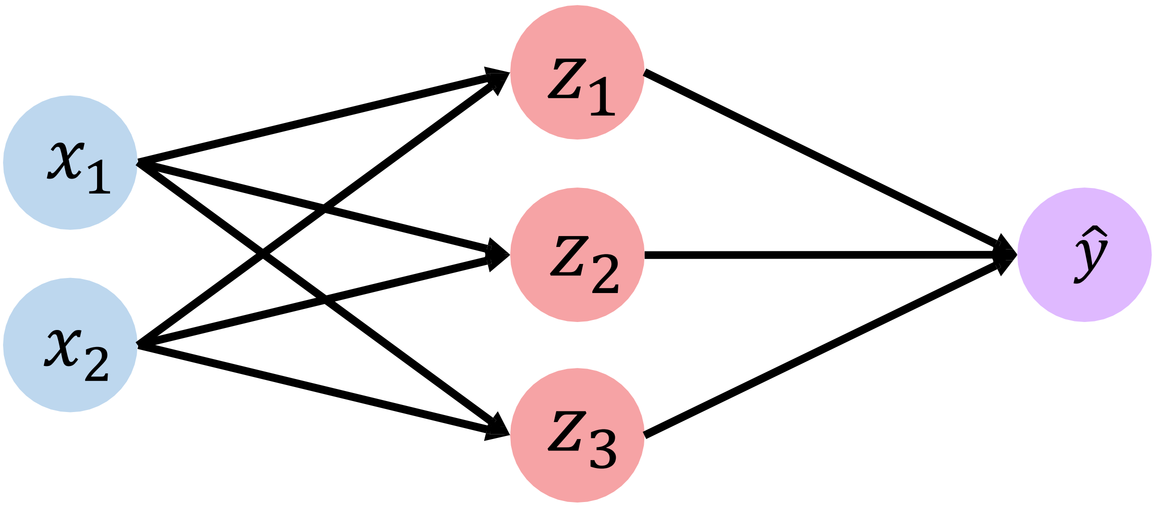


$$loss(\underbrace{f(X^{[1]}; W)}_{\text{Predicted}}, \underbrace{Y^{[1]}}_{\text{Actual}})$$

Model Assessment – Empirical Risk

The empirical Risk measures the total loss over the training dataset

$$X = \begin{bmatrix} 4, & 5 \\ 2, & 2 \\ 6, & 8 \\ \vdots & \vdots \end{bmatrix}$$



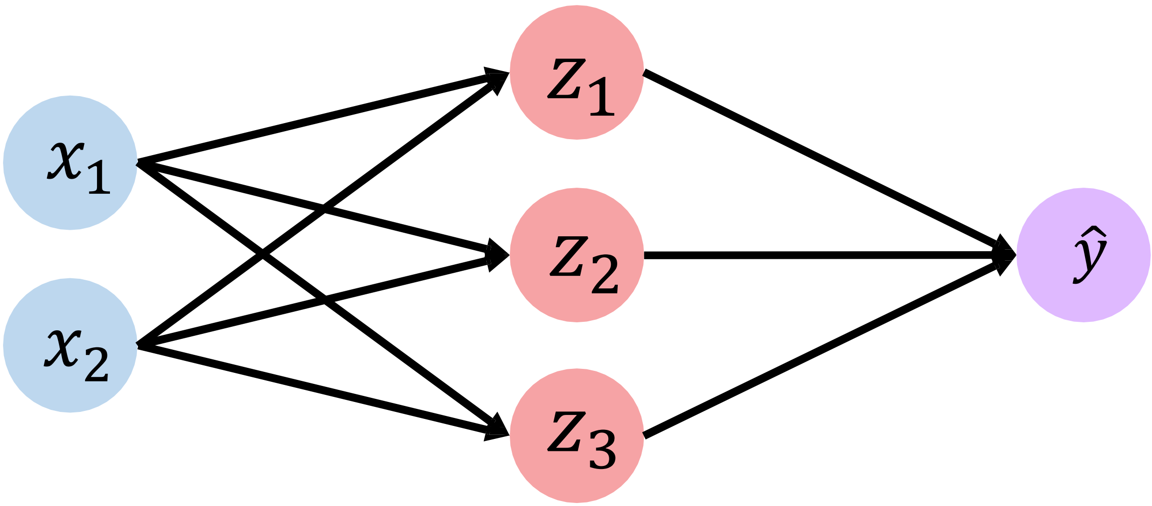
$$\begin{array}{cc} f(x) & y \\ \underline{\begin{bmatrix} 0.4 \\ 0.25 \\ 0.55 \\ \vdots \end{bmatrix}} & \underline{\begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}} \end{array}$$

$$\min \quad J(W) = \frac{1}{n}\sum_{i=1}^{n} loss(\underbrace{f(X^{[i]};W)}_{\text{Predicted}},\ \underbrace{Y^{[i]}}_{\text{Actual}})$$

Model Assessment – Empirical Risk

The empirical Risk measures the total loss over the training dataset

$$X = \begin{bmatrix} 4, & 5 \\ 2, & 2 \\ 6, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\underbrace{\begin{bmatrix} 0.4 \\ 0.25 \\ 0.55 \\ \vdots \end{bmatrix}}_{f(x)} \quad \underbrace{\begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}}_{y}$$

Also known as:

- Objective function
- Cost function

$$\min \quad J(W) = \frac{1}{n}\sum_{i=1}^{n} loss(\underbrace{f(X^{[i]};W)}_{\text{Predicted}},\ \underbrace{Y^{[i]}}_{\text{Actual}})$$

Quantifying Loss – Binary Classification

- 0/1 Loss: $loss(Y, f(X)) = 1_{Y \neq f(X)}$
  - $loss = 0$ when correctly classified, 1 when misclassified
  - Intuitive but hard to train

$$\min \quad J(W) = \frac{1}{n} \sum_{i=1}^{n} loss(\underbrace{f(X^{[i]}; W)}_{\text{Predicted}},\ \underbrace{Y^{[i]}}_{\text{Actual}})$$

Quantifying Loss – Binary Classification

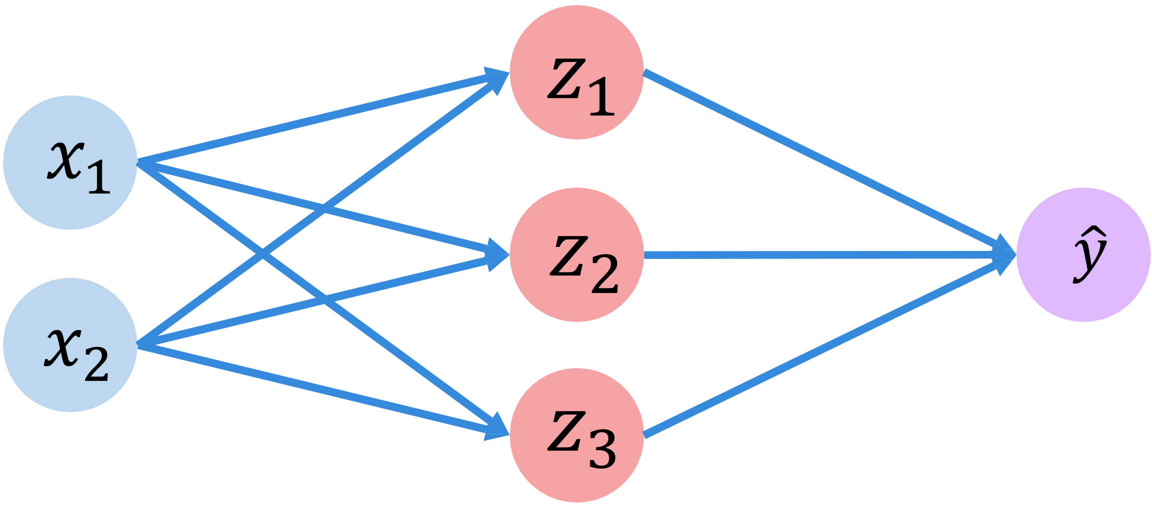
- 0/1 Loss: $loss(Y, f(X)) = 1_{Y \neq f(X)}$
  - $loss = 0$ when correctly classified, 1 when misclassified
  - Intuitive but hard to train
- Binary Cross Entropy Loss: $loss(Y, f(X)) = -Y \log f(X) - (1 - Y) \log(1 - f(X))$
  - measure of the difference between two probability distributions
  - Can be used when output a probability from 0 to 1

$$\min \quad J(W) = \frac{1}{n} \sum_{i=1}^{n} loss(\underbrace{f(X^{[i]}; W)}_{\text{Predicted}}, \ \underbrace{Y^{[i]}}_{\text{Actual}})$$

Classification: Binary Cross Entropy Loss

Using Binary Cross Entropy Loss for binary classification

$$x = \begin{bmatrix} 4, & 5 \\ 2, & 2 \\ 6, & 8 \\ \vdots & \vdots \end{bmatrix}$$

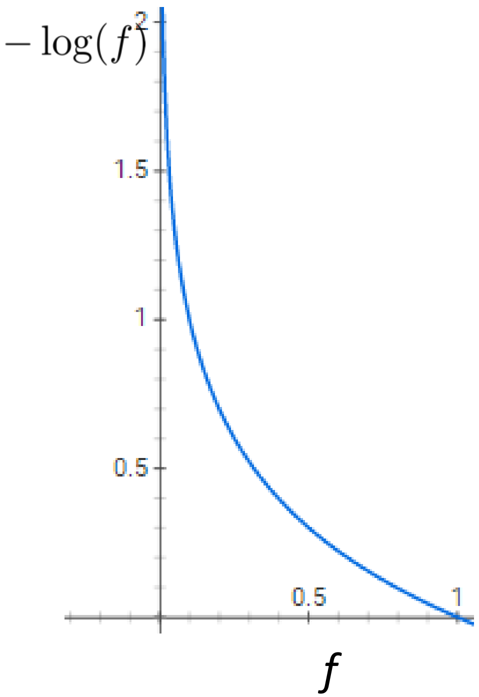


$$\underbrace{\begin{bmatrix} 0.4 \\ 0.25 \\ 0.55 \\ \vdots \end{bmatrix}}_{f(x)} \quad \underbrace{\begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}}_{y}$$

$$\min \quad J(W) = \frac{1}{n}\sum_{i=1}^{n} -\underbrace{Y^{[i]}}_{\text{Actual}} \log \underbrace{f\left(X^{[i]}; W\right)}_{\text{Predicted}} - \underbrace{\left(1 - Y^{[i]}\right)}_{\text{Actual}} \log\underbrace{\left(1 - f\left(X^{[i]}; W\right)\right)}_{\text{Predicted}}$$

Classification: Binary Cross Entropy Loss

True label
$Y^{[i]} = 1$



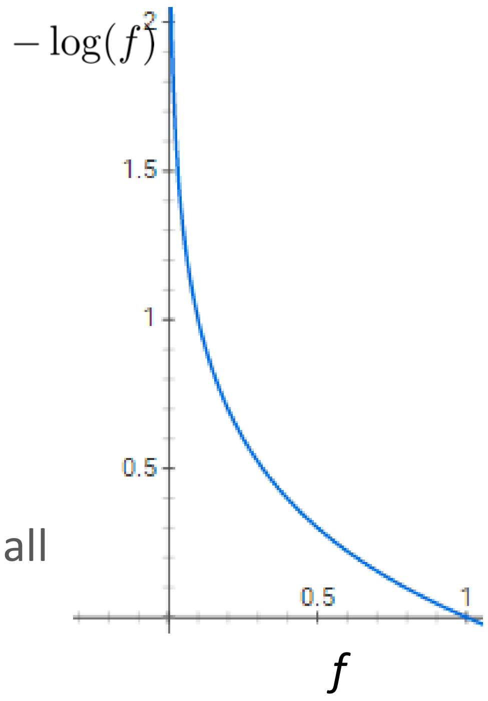
f

$$\min \quad J(W) = \frac{1}{n}\sum_{i=1}^{n} -\underbrace{Y^{[i]}}_{\text{Actual}} \log \underbrace{f\left(X^{[i]}; W\right)}_{\text{Predicted}} - \left(1 - \underbrace{Y^{[i]}}_{\text{Actual}}\right) \log(\underbrace{1 - f\left(X^{[i]}; W\right)}_{\text{Predicted}})$$

Classification: Binary Cross Entropy Loss

True label
$Y^{[i]} = 1$

Hope $-\log(f)$ to be small such that f is close to 1

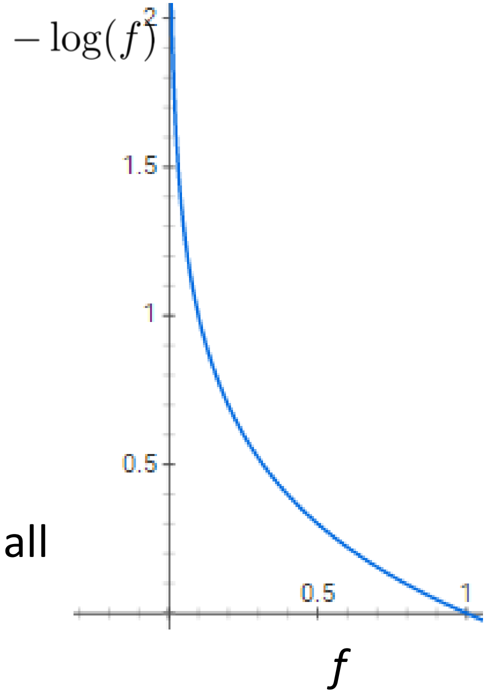


$$\min \quad J(W) = \frac{1}{n}\sum_{i=1}^{n} -\underbrace{Y^{[i]}}_{\text{Actual}} \log \underbrace{f\left(X^{[i]}; W\right)}_{\text{Predicted}} - \underbrace{\left(1 - Y^{[i]}\right)}_{\text{Actual}} \log\underbrace{\left(1 - f\left(X^{[i]}; W\right)\right)}_{\text{Predicted}}$$

Classification: Binary Cross Entropy Loss

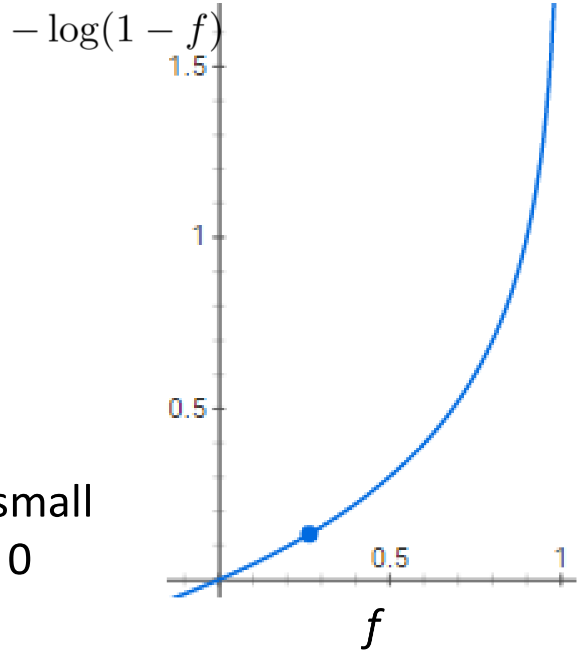
True label
$Y^{[i]} = 1$

Hope $-\log(f)$ to be small such that f is close to 1



True label
$Y^{[i]} = 0$

Hope $-\log(1-f)$ to be small such that f to be close to 0



$$\min \quad J(W) = \frac{1}{n}\sum_{i=1}^{n} -\underbrace{Y^{[i]}}_{\text{Actual}} \log \underbrace{f\left(X^{[i]};W\right)}_{\text{Predicted}} - \left(1-\underbrace{Y^{[i]}}_{\text{Actual}}\right)\log\left(\underbrace{1-f\left(X^{[i]};W\right)}_{\text{Predicted}}\right)$$

Classification: Binary Cross Entropy Loss

- Binary Cross Entropy Loss

$$loss = -Y \log f(X;W) - (1-Y)\log(1-f(X;W))$$

- Vectorize True Label as (Y_1, Y_0): $Y_1 = Y$ and $Y_0 = 1 - Y$

- Prediction: $\hat{Y}_1 = f(X;W)$ versus $\hat{Y}_0 = 1 - f(X;W)$

$$loss = -Y_1 \log \hat{Y}_1 - Y_0 \log \hat{Y}_0$$

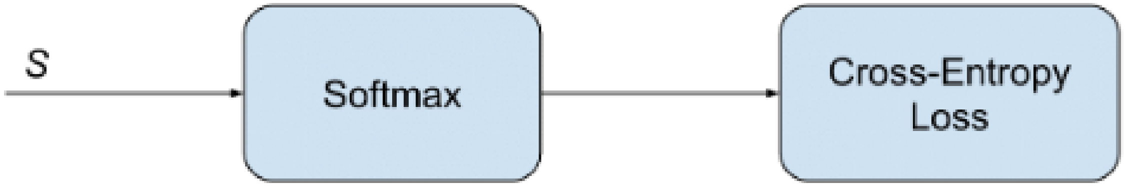
  – Note that $Y_1 + Y_0 = 1$ and $\hat{Y}_1 + \hat{Y}_0 = 1$

- Extend the definition Multi-Class?

Classification: Multi-class Cross Entropy Loss

- Binary Cross-entropy $\qquad loss = -Y_1 \log \hat{Y}_1 - Y_0 \log \hat{Y}_0$
- K Classes: $\qquad loss = -\sum_{k=1,\ldots,K} Y_k \log\left(\hat{Y}_k\right)$
- Vectorize $Y = (Y_1, Y_2, Y_3, \ldots)^{\top}$: $\qquad loss = -Y^T \log\left(\hat{Y}\right)$

- The true label vector $\sum_k y_k = 1$. How to make sure $\sum_k \hat{y}_k = 1$?
  - Apply a Softmax operator to the output
  - Use it as the activation function of the last layer
  - When no hidden layer: logistic regression

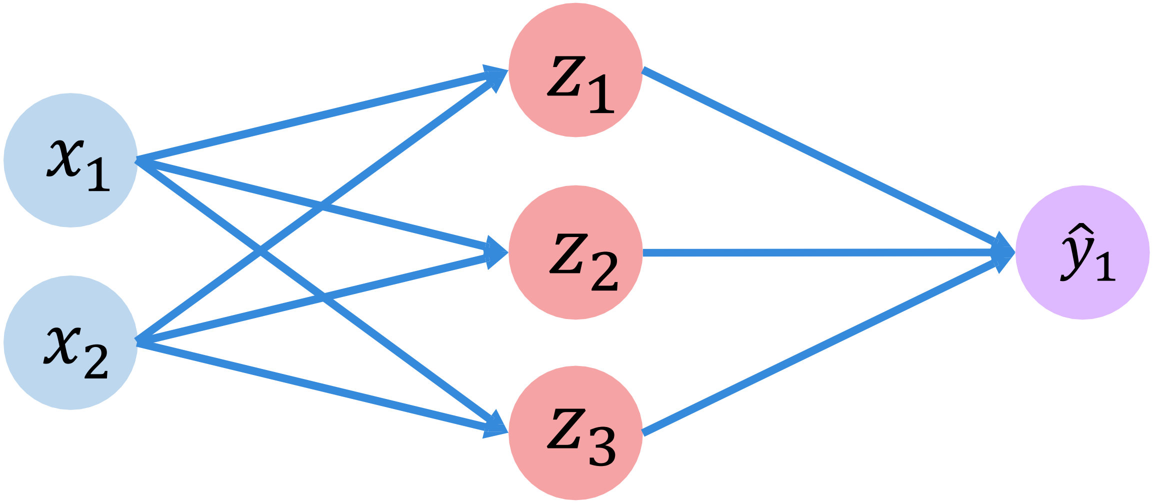


$$\hat{y}_k = \frac{\exp(s_k)}{\sum \exp(s_i)}$$

Regression: Mean Squared Error Loss

Used with regression models that output continuous real numbers

$$x = \begin{bmatrix} 4, & 5 \\ 2, & 2 \\ 6, & 8 \\ \vdots & \vdots \end{bmatrix}$$



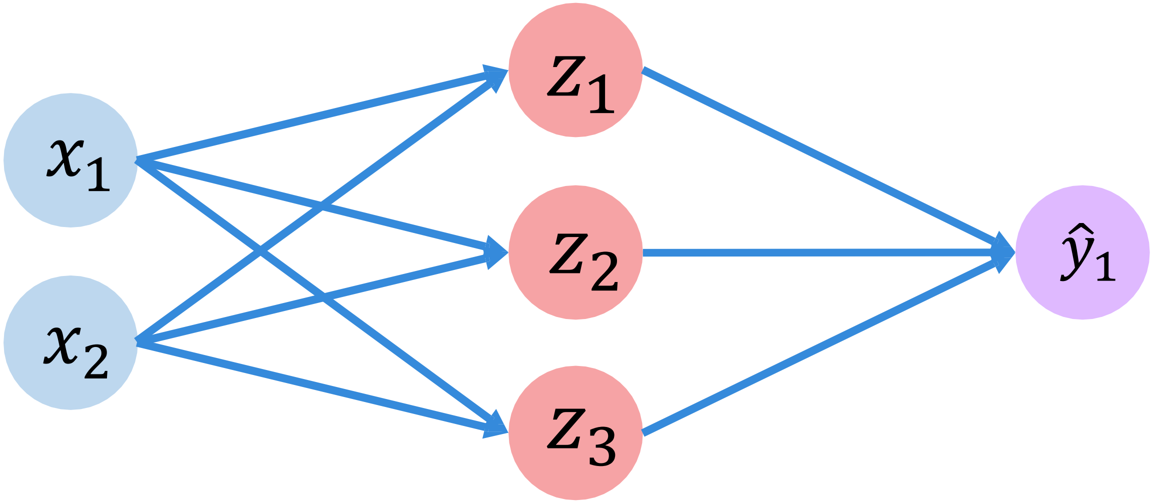
$$\underbrace{\overset{f(x)}{\begin{bmatrix} 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix}} \quad \overset{y}{\begin{bmatrix} 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix}}}_{\text{Final Grades (percentage)}}$$

$$\min \quad J(W) = \frac{1}{n} \sum_{i=1}^{n} loss\big(\underbrace{f(X^{[i]}; W)}_{\text{Predicted}},\ \underbrace{Y^{[i]}}_{\text{Actual}} \big)$$

Regression: Mean Squared Error Loss

Used with regression models that output continuous real numbers

$$x = \begin{bmatrix} 4, & 5 \\ 2, & 2 \\ 6, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\begin{array}{cc} f(x) & y \\ \begin{bmatrix} 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix} & \begin{bmatrix} 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix} \end{array}$$

Final Grades (percentage)

$$\min \quad J(W) = \frac{1}{n}\sum_{i=1}^{n} \Big(\underbrace{f\big(X^{[i]}; W\big)}_{\text{Predicted}} - \underbrace{Y^{[i]}}_{\text{Actual}} \Big)^2$$

Loss Optimization

We want to find the network weights that achieve the lowest loss

$$W^* = \underset{W}{\operatorname{argmin}} J(W) = \frac{1}{n}\sum_{i=1}^{n} loss(\, f(X^{[i]}; W),\ Y^{[i]})$$

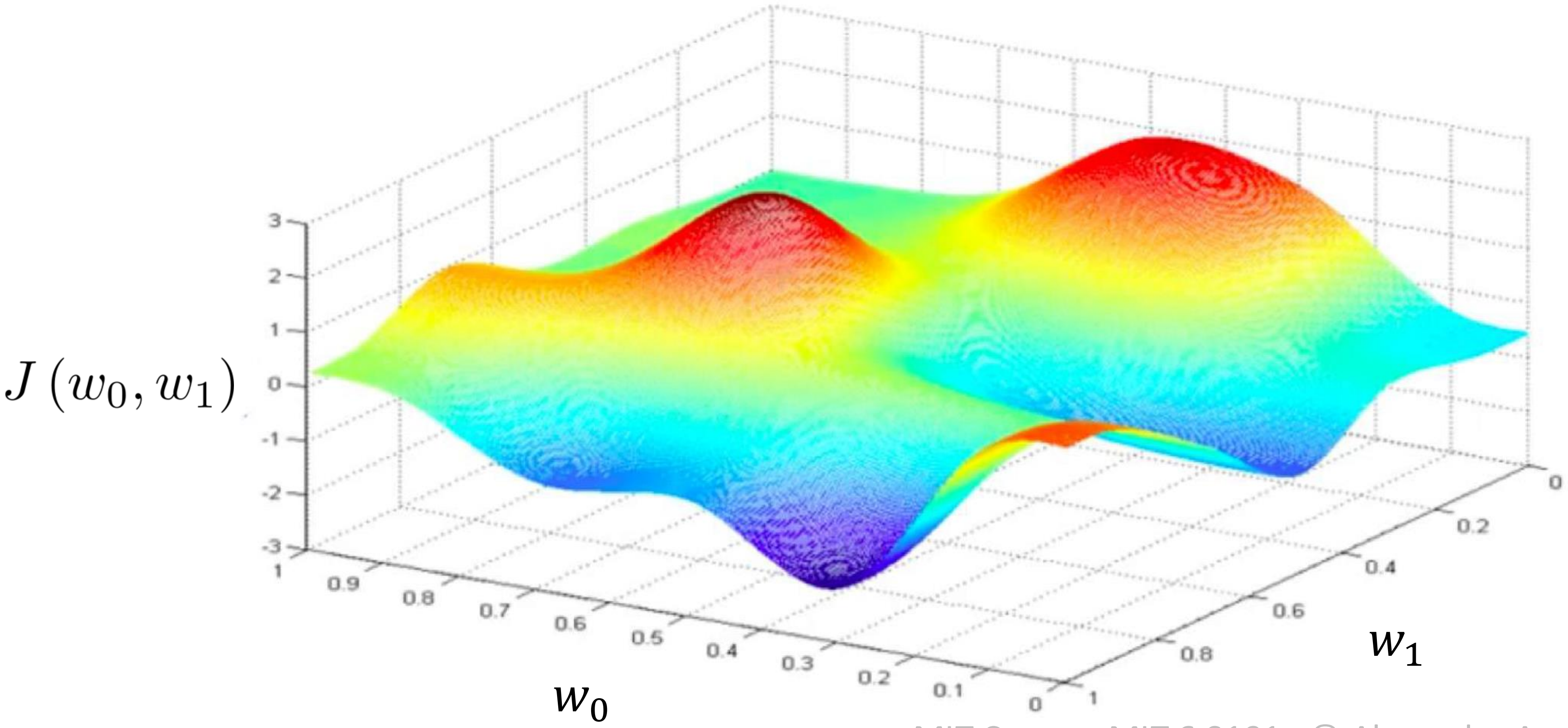
$$\boldsymbol{W} = \left\{\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \cdots\right\}$$

Remember:

Our loss is a function of the network weights!

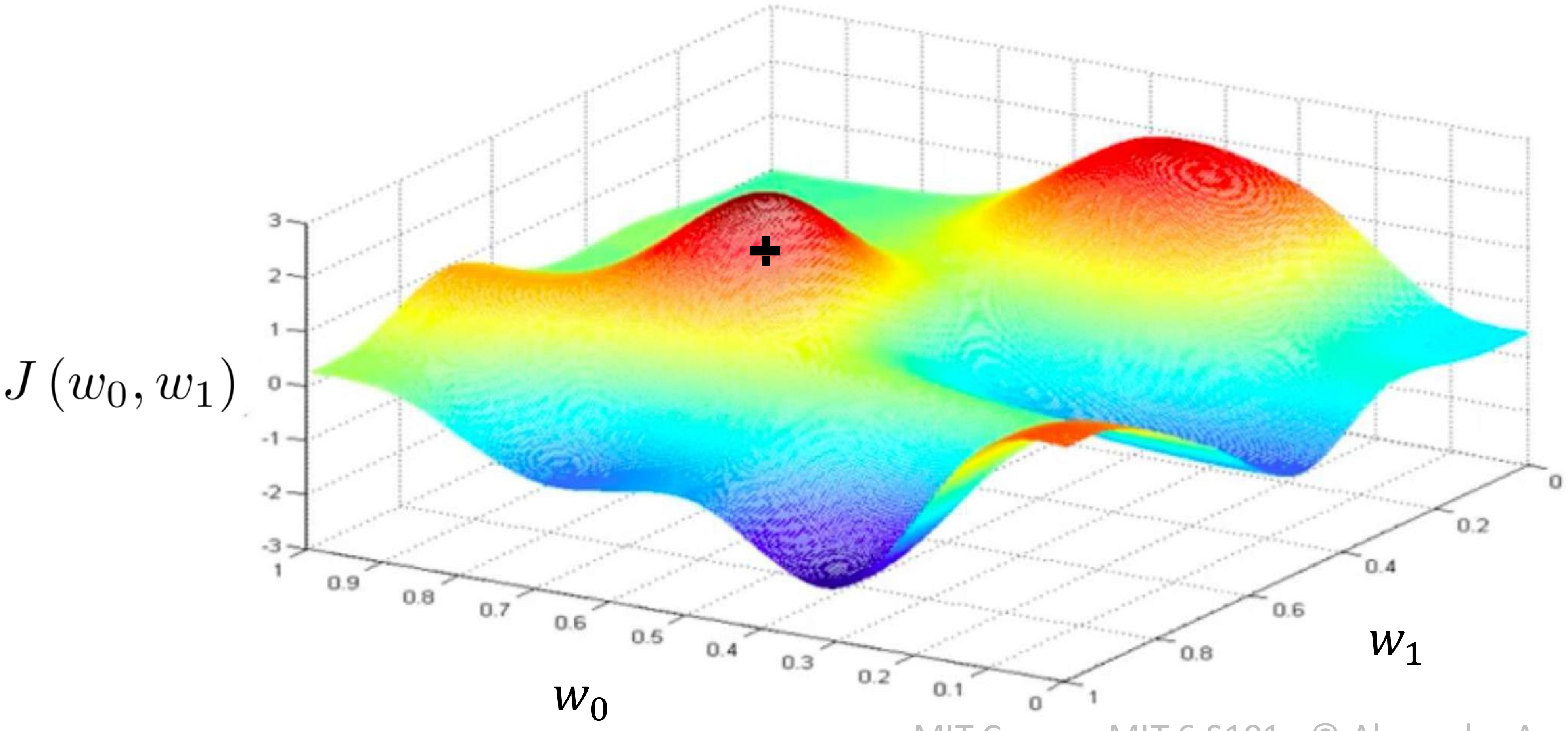
Loss Optimization - Gradient Descent

$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$



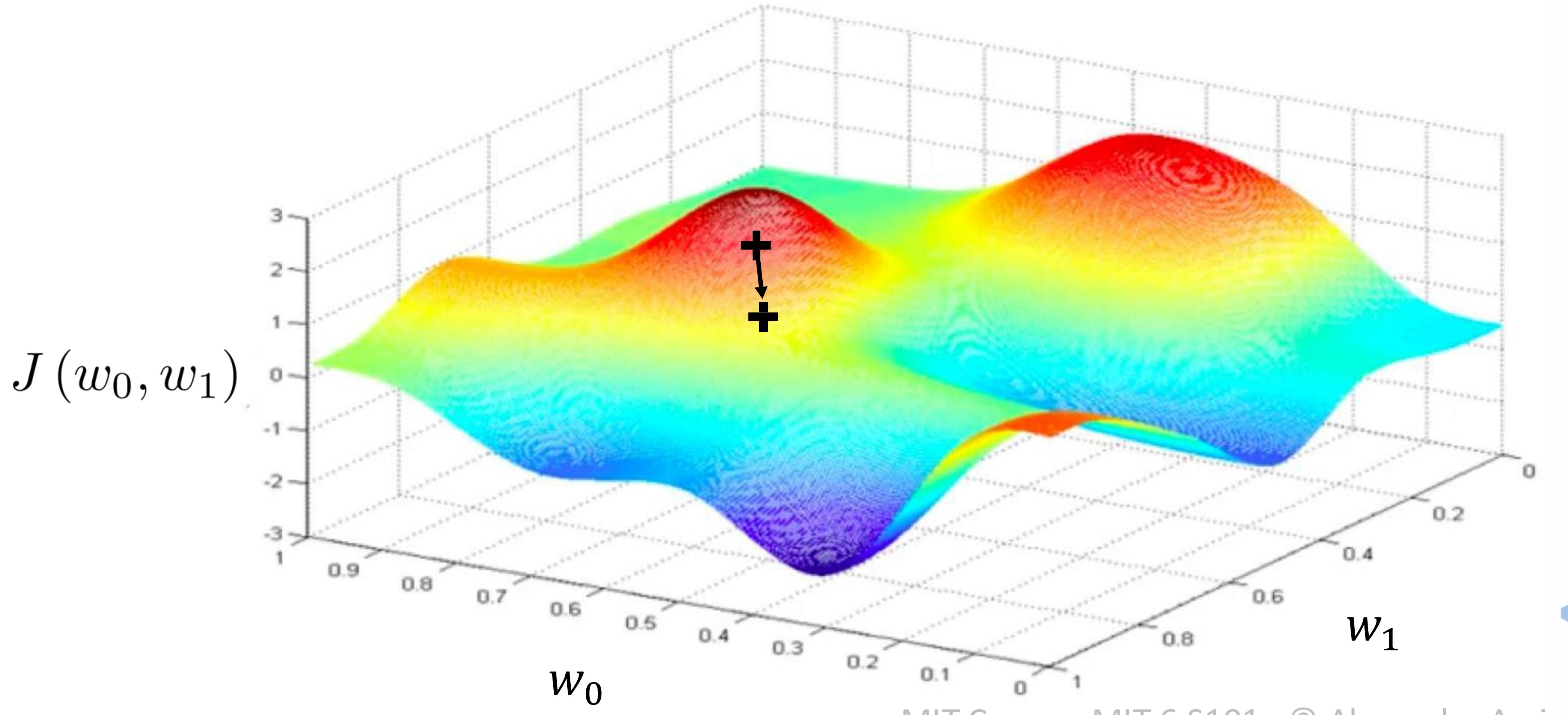
Loss Optimization - Gradient Descent

Randomly pick an initial (w_0, w_1)



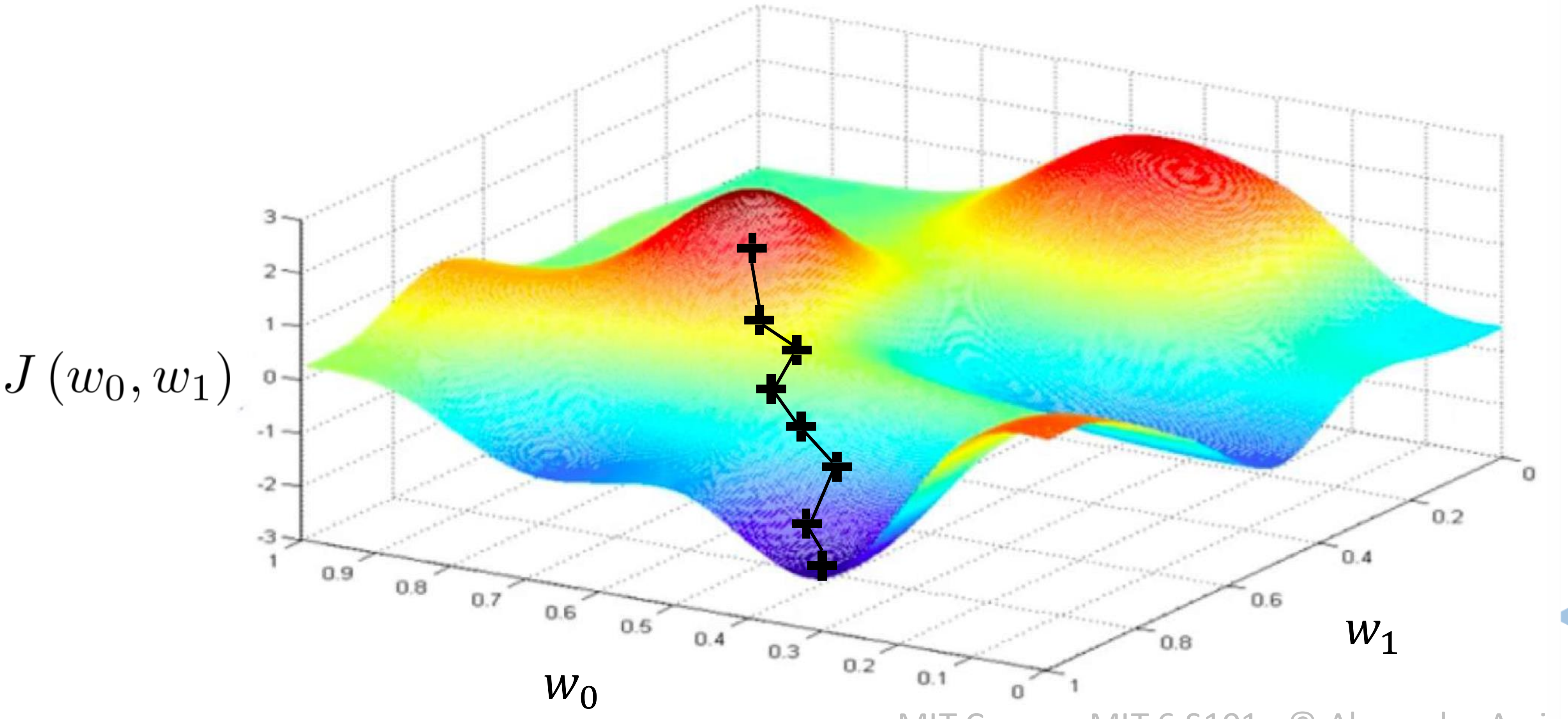
Loss Optimization - Gradient Descent

Compute gradient and move towards negative gradient direction



Loss Optimization - Gradient Descent

Repeat until convergence



Loss Optimization - Gradient Descent

1. Initialize weights
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $W^{t+1} \leftarrow W^t - \eta \frac{\partial J(W)}{\partial W^\top}$
5. Return weights

Loss Optimization - Gradient Descent

1. Initialize weights

Question 1: How to set initial weights?

2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $W^{t+1} \leftarrow W^{t} - \eta \frac{\partial J(W)}{\partial W^{\top}}$
5. Return weights

Loss Optimization - Gradient Descent

1. Initialize weights
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(W)}{\partial W}}$
4. Update weights, $W^{t+1} \leftarrow W^{t} - \eta \frac{\partial J(W)}{\partial W^{\top}}$
5. Return weights

Question 1: How to set initial weights?

Question 2: How to compute gradient?

Loss Optimization - Gradient Descent

1. Initialize weights
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(W)}{\partial W}}$
4. Update weights, $W^{t+1} \leftarrow W^t - \boxed{\eta} \frac{\partial J(W)}{\partial W^\top}$
5. Return weights

Question 1: How to set initial weights?

Question 2: How to compute gradient?

Question 3: How to choose the stepsize?

η: Stepsize (Learning Rates)

Loss Optimization - Gradient Descent

1. Initialize weights
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(W)}{\partial W}}$
4. Update weights, $W^{t+1} \leftarrow W^{t} - \boxed{\eta} \frac{\partial J(W)}{\partial W^{\top}}$
5. Return weights

Question 1: How to set initial weights?

Question 4: When to stop?

Question 2: How to compute gradient?

Question 3: How to choose the stepsize?

η: Stepsize (Learning Rates)

Questions?