Introduction to Deep Learning

Alexander Amini

MIT 6.S191

January 24, 2022

What is Deep Learning?

Artificial Intelligence

Any technique that enables computers to mimic human behavior

Machine Learning

Ability to learn without explicitly being programmed

Deep Learning

Extract patterns from data using neural networks

Why Now?

1952	Stochastic Gradient Descent
1958	Perceptron • Learnable Weights
⋮	
1986	Backpropagation • Multi-Layer Perceptron
1995	Deep Convolutional NN • Digit Recognition
⋮	

Neural Networks date back decades, so why the resurgence?

1. Big Data

- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

2. Hardware

- Graphics Processing Units (GPUs)
- Massively Parallelizable

3. Software

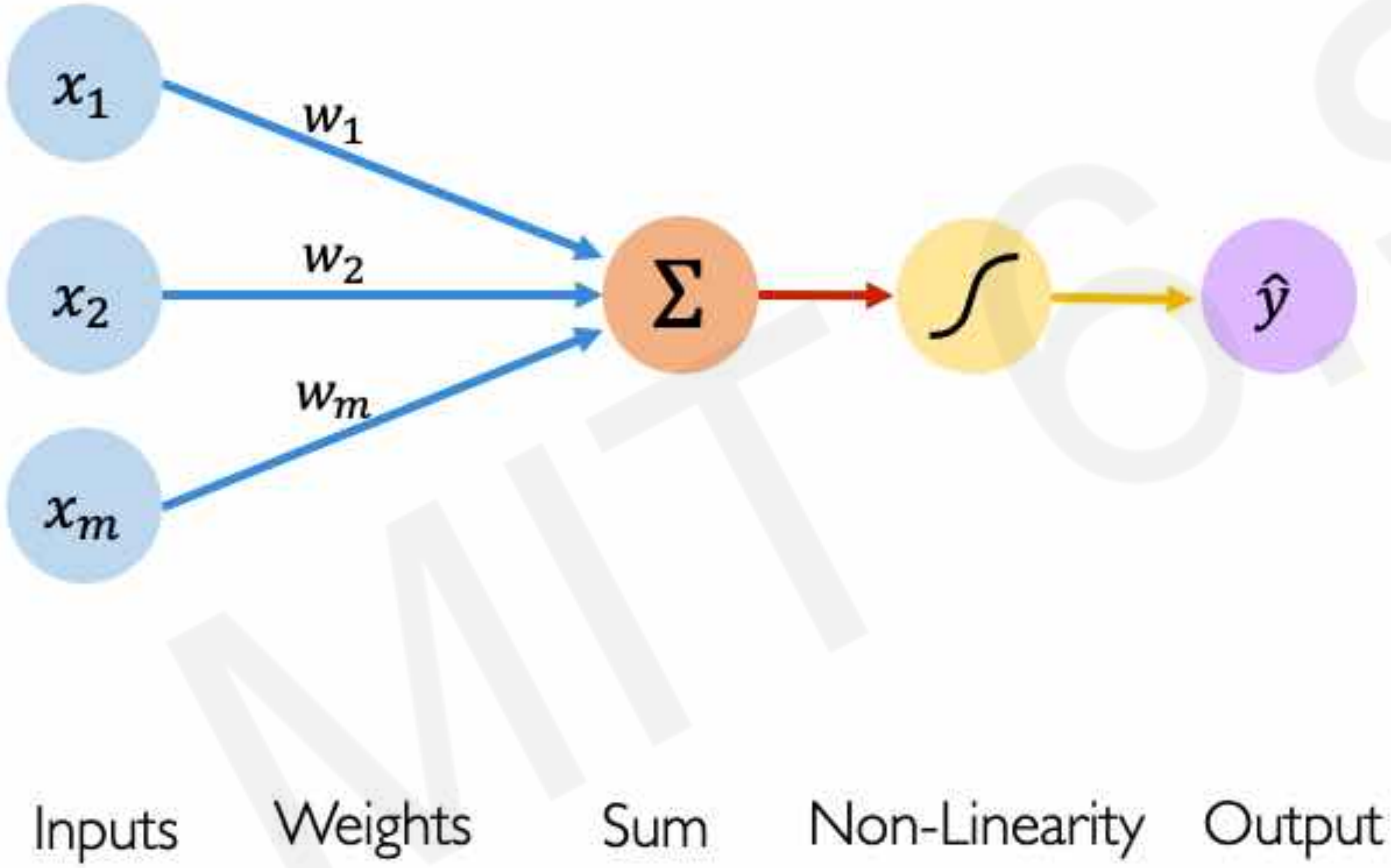
- Improved Techniques
- New Models
- Toolboxes

TensorFlow

The Perceptron

The structural building block of deep learning

The Perceptron: Forward Propagation



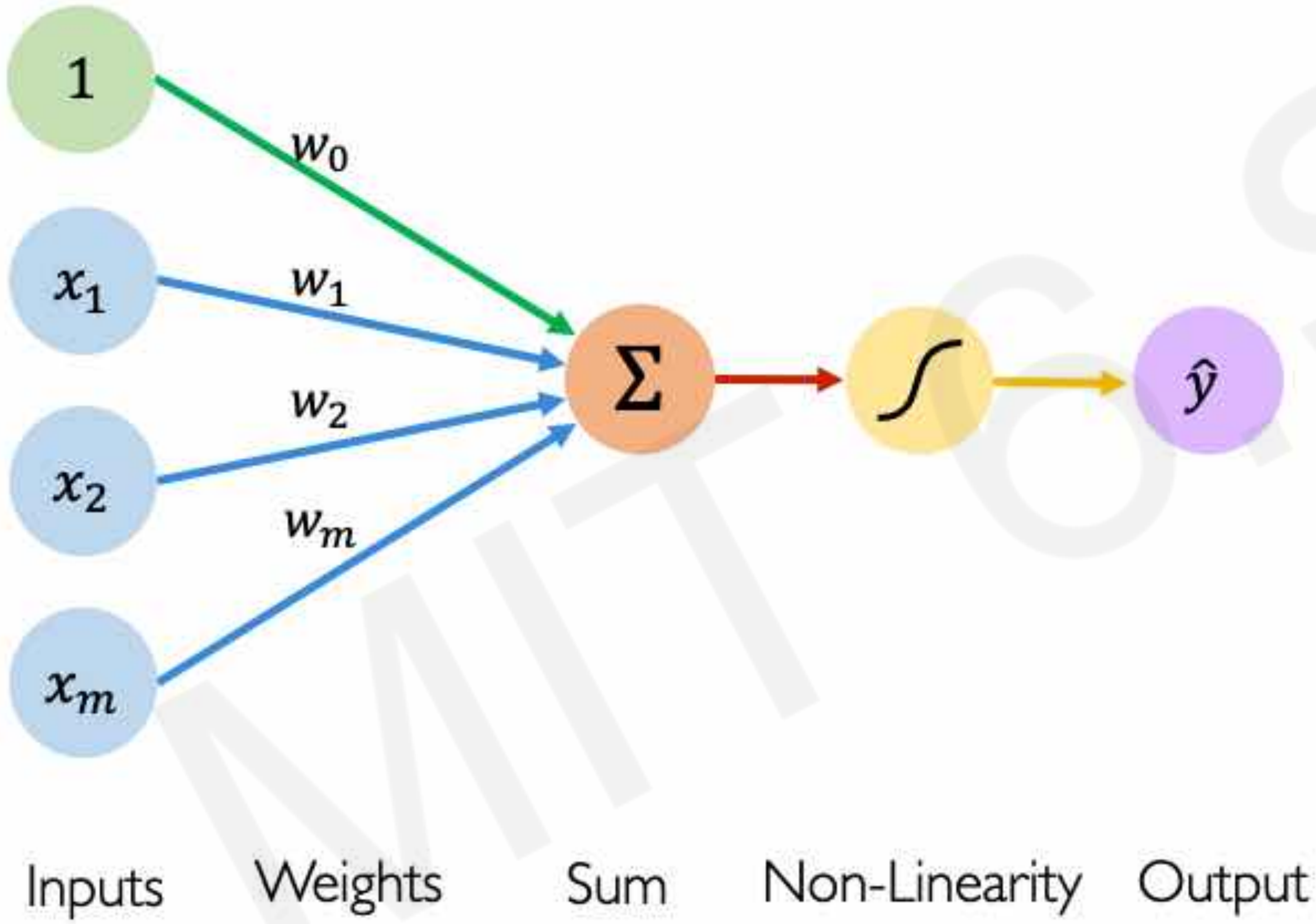
Output

Linear combination of inputs

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

Non-linear activation function

The Perceptron: Forward Propagation



Output

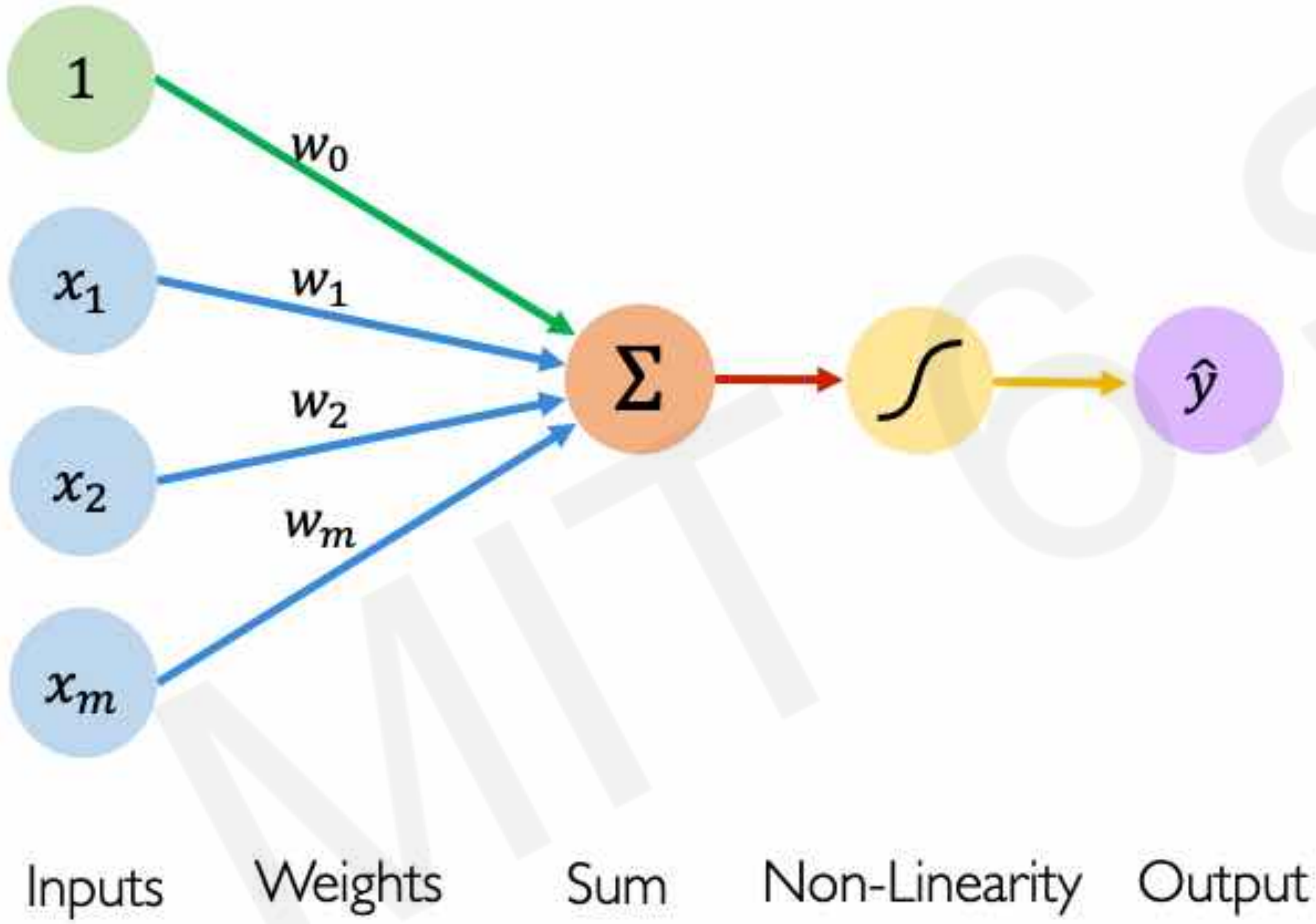
Linear combination of inputs

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

Non-linear activation function

Bias

The Perceptron: Forward Propagation

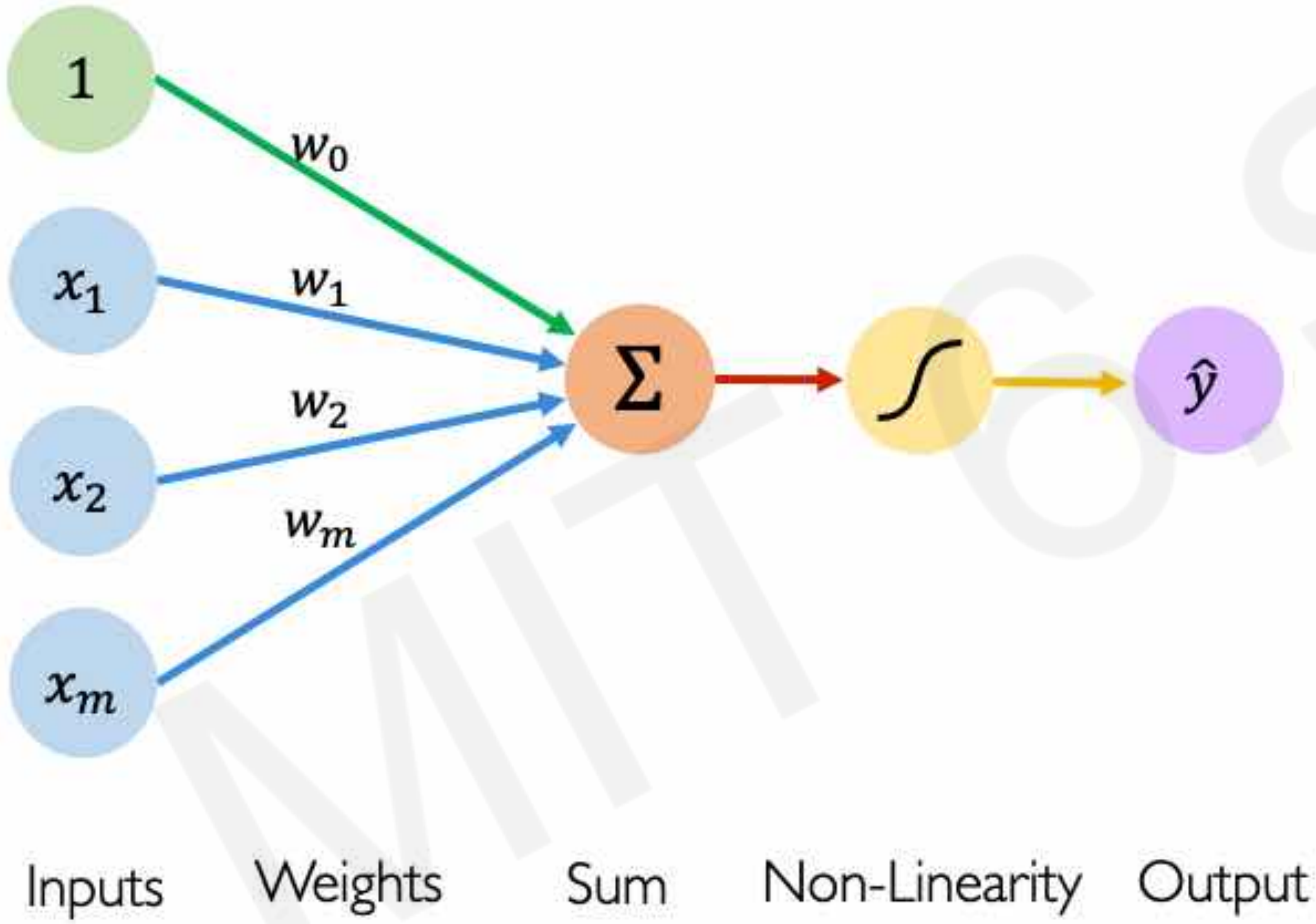


$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

$$\hat{y} = g\,(\, w_0 + \boldsymbol{X}^T \boldsymbol{W} \,)$$

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

The Perceptron: Forward Propagation

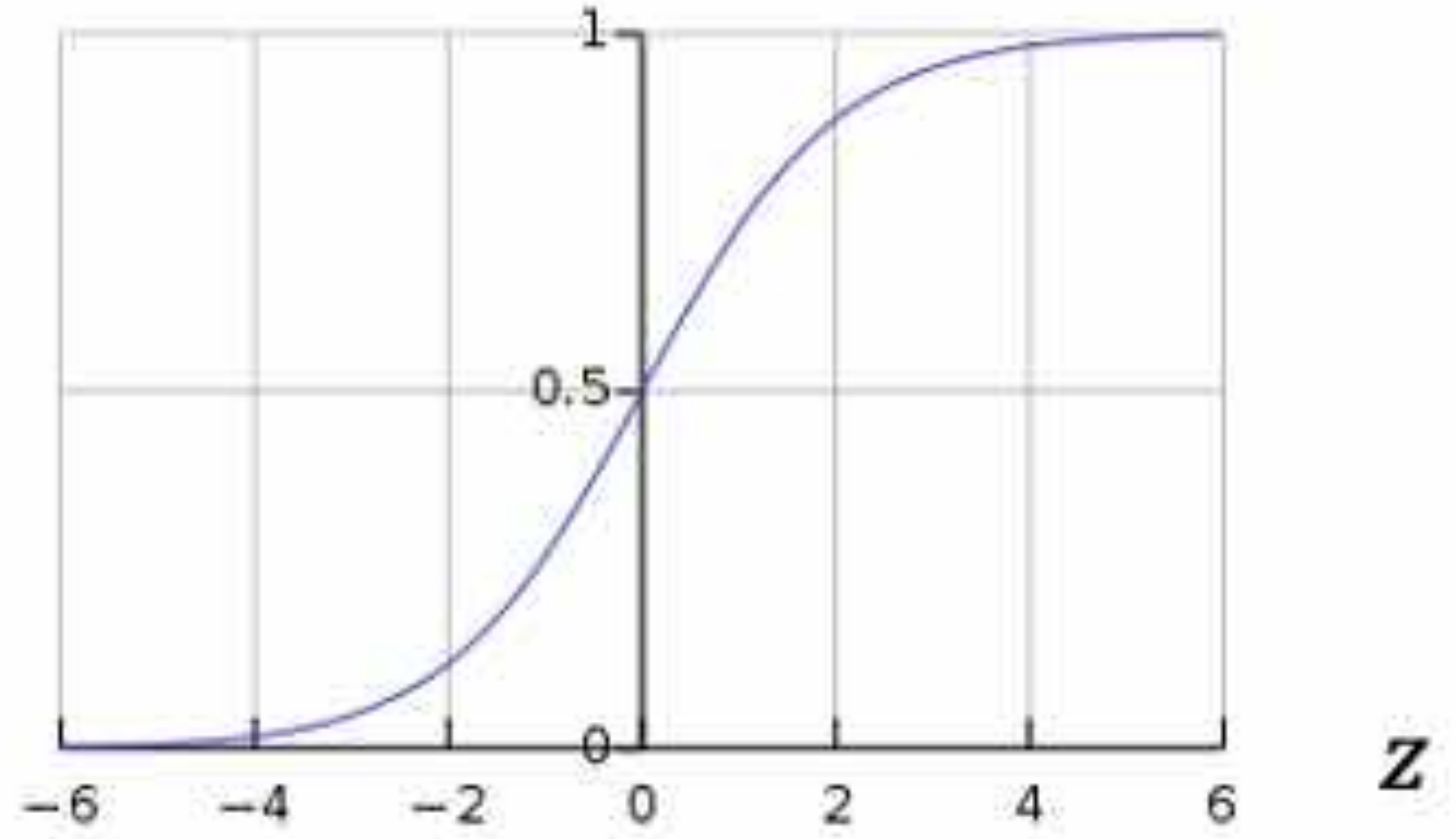


Activation Functions

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

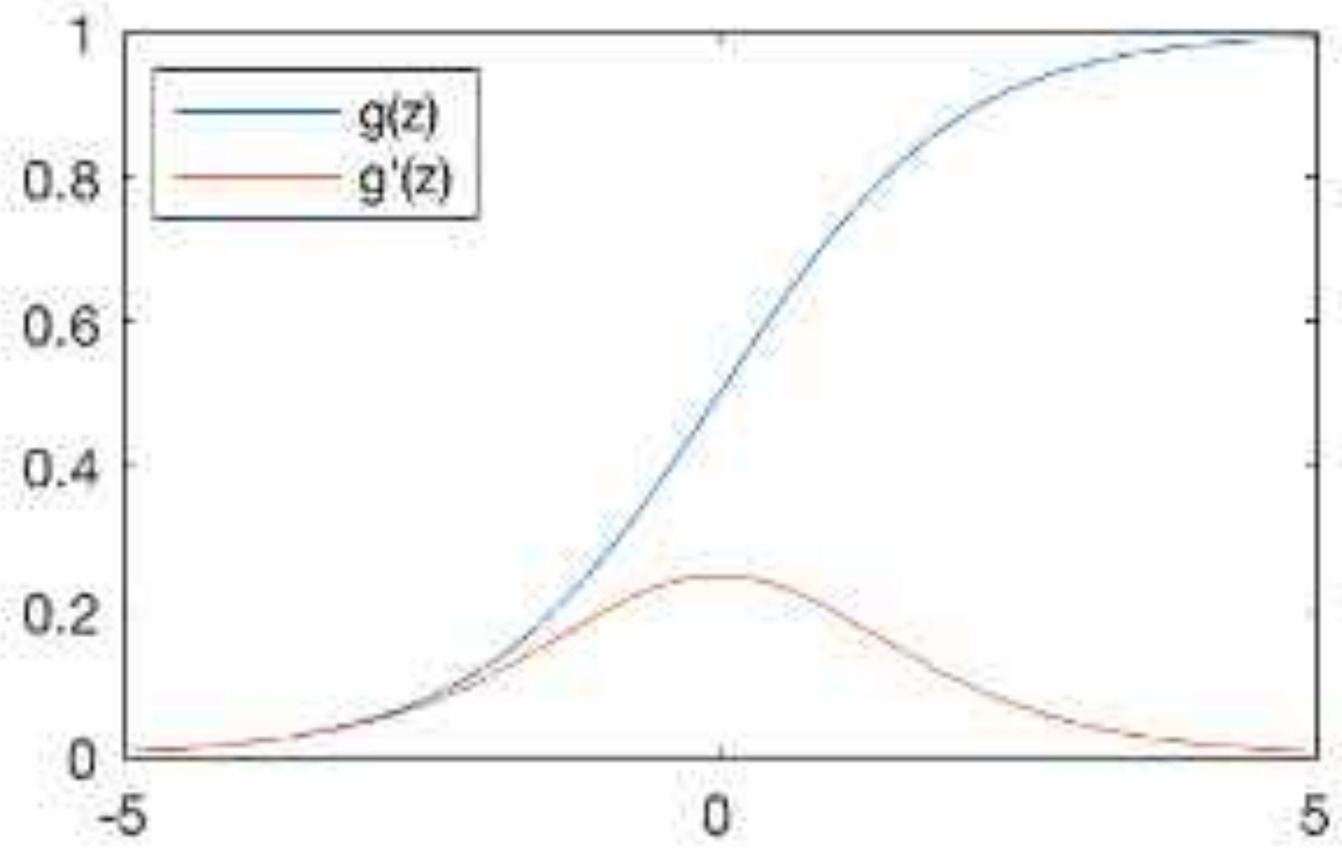
- Example: sigmoid function

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



Common Activation Functions

Sigmoid Function

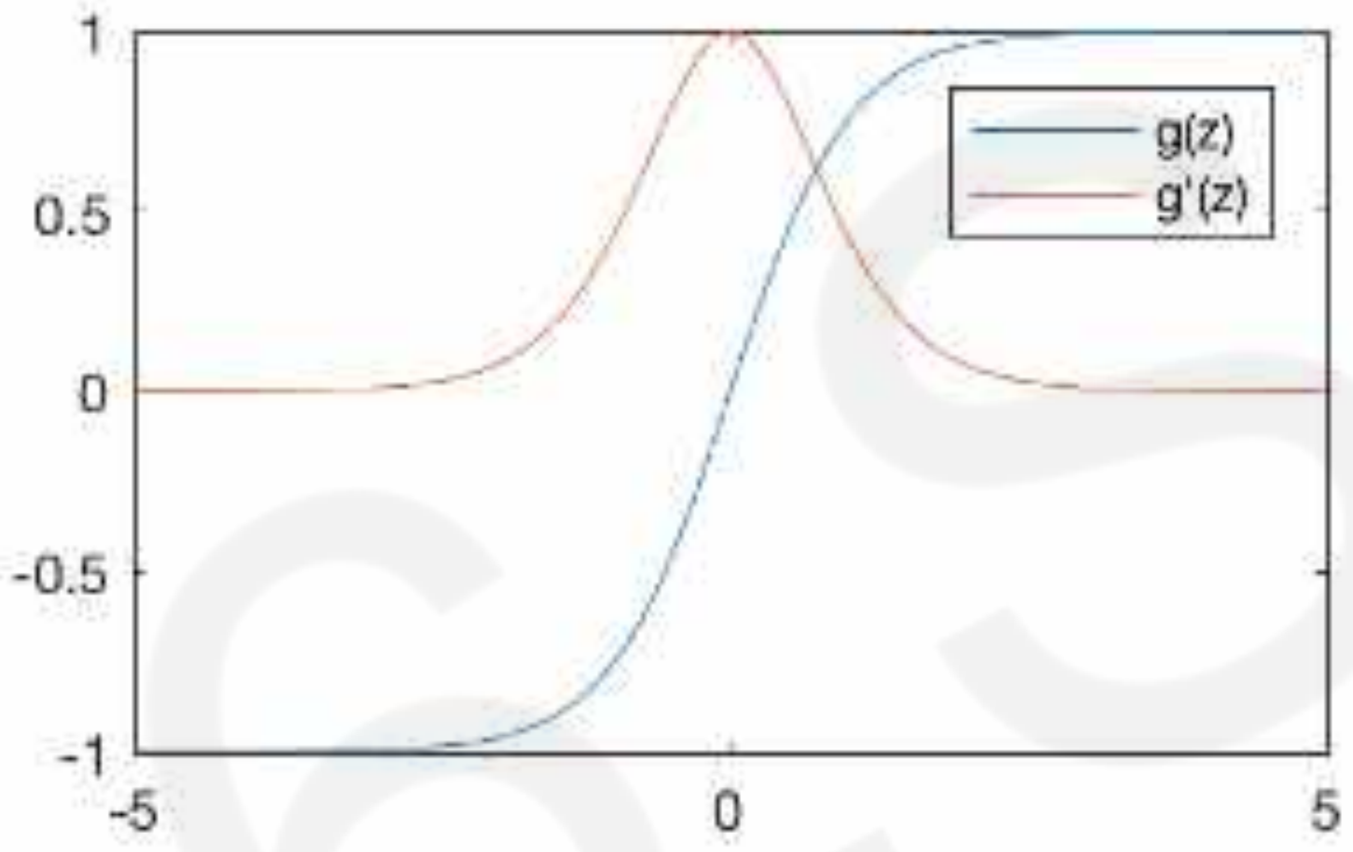


$$g(z) = \frac{1}{1 + e^{-z}}$$

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

```
tf.math.sigmoid(z)
```

Hyperbolic Tangent

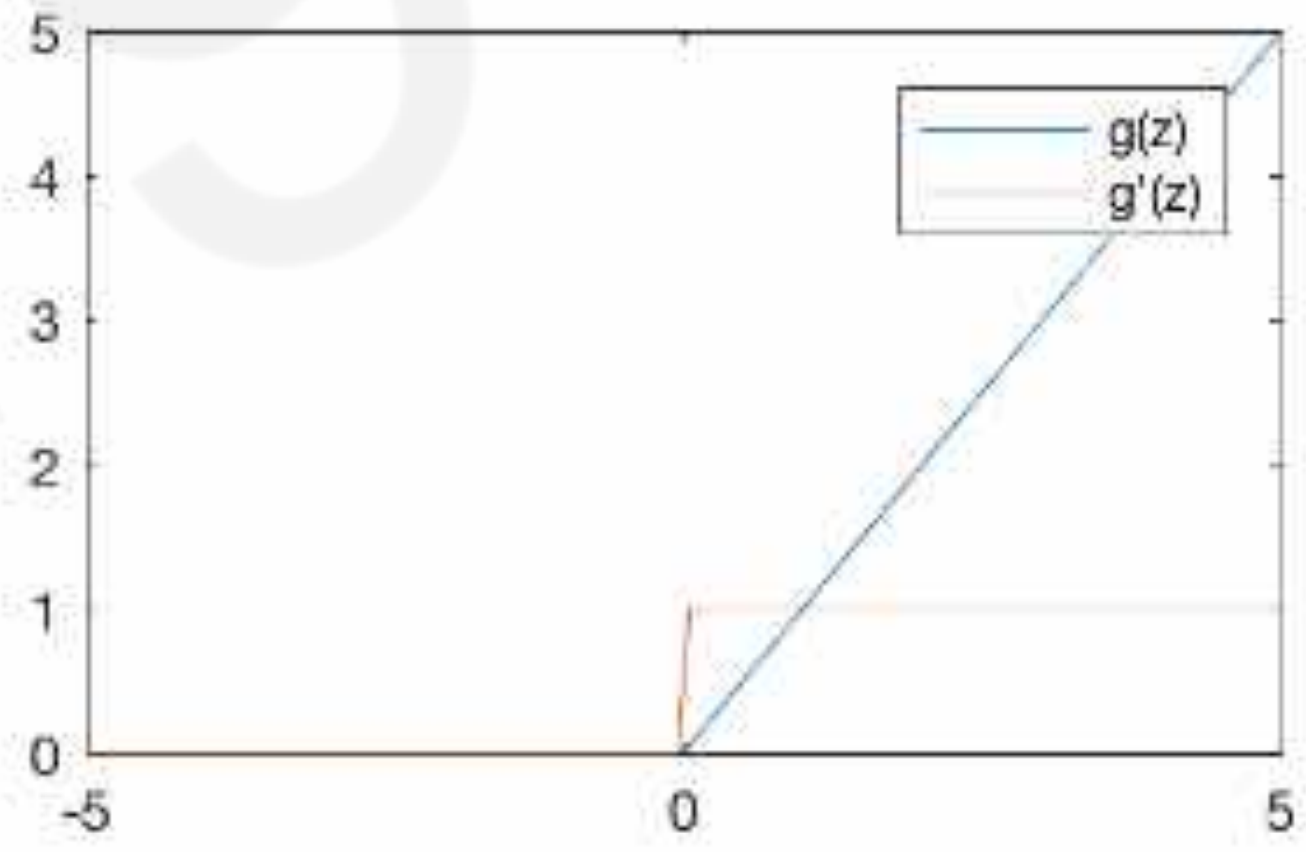


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

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

```
tf.math.tanh(z)
```

Rectified Linear Unit (ReLU)



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

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

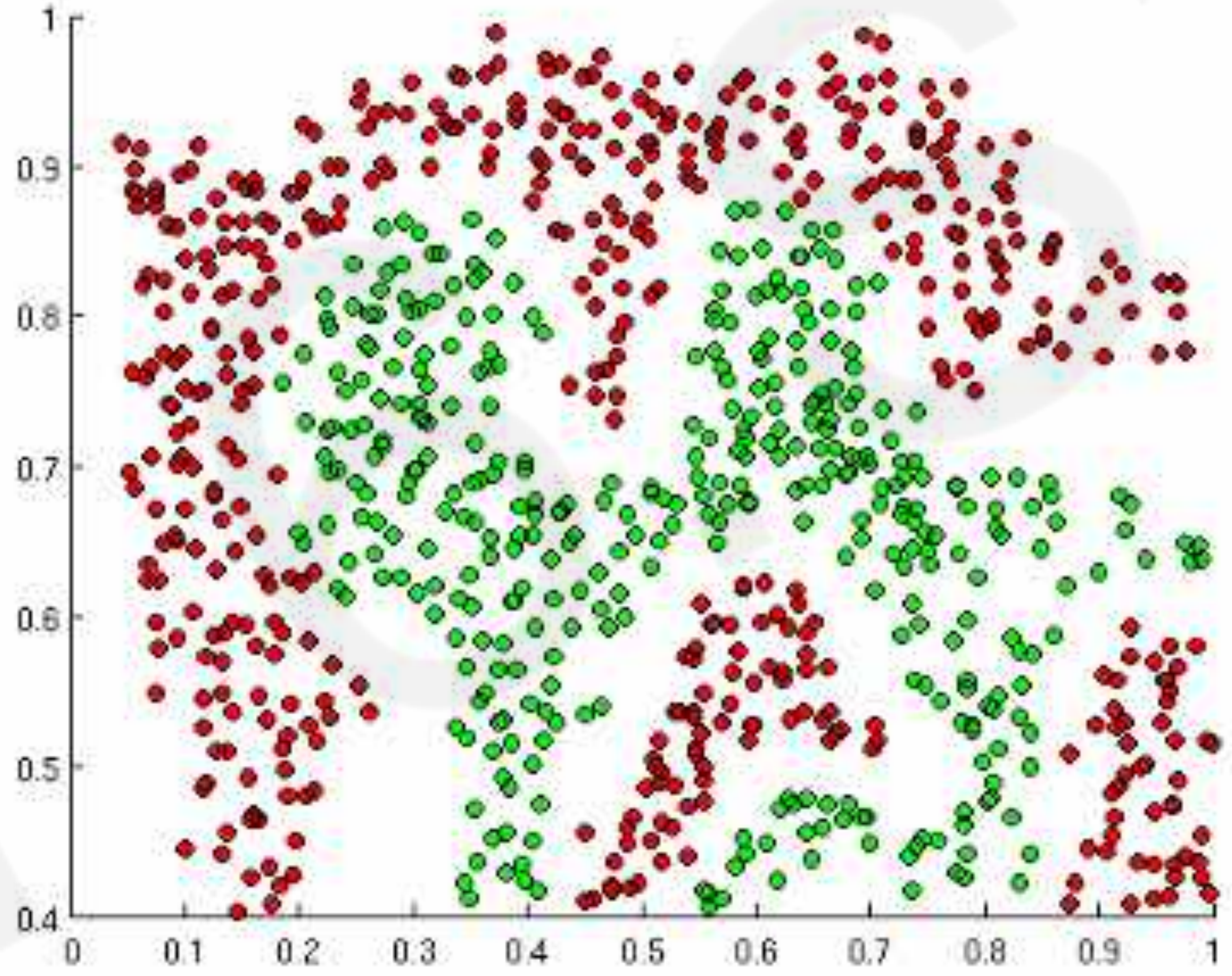
```
tf.nn.relu(z)
```

TensorFlow code blocks

NOTE: All activation functions are non-linear

Importance of Activation Functions

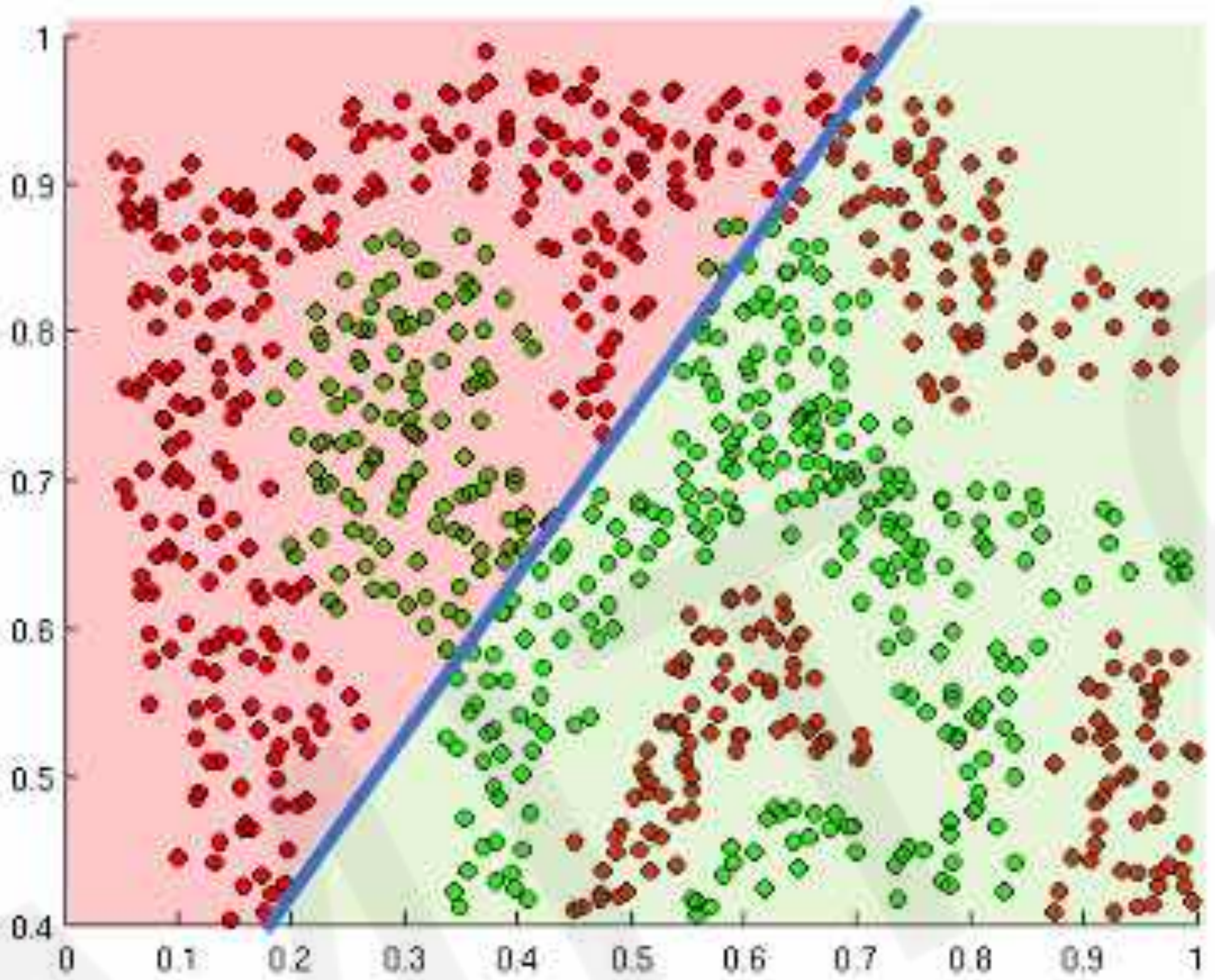
The purpose of activation functions is to ***introduce non-linearities*** *into the network*



What if we wanted to build a neural network to distinguish green vs red points?

Importance of Activation Functions

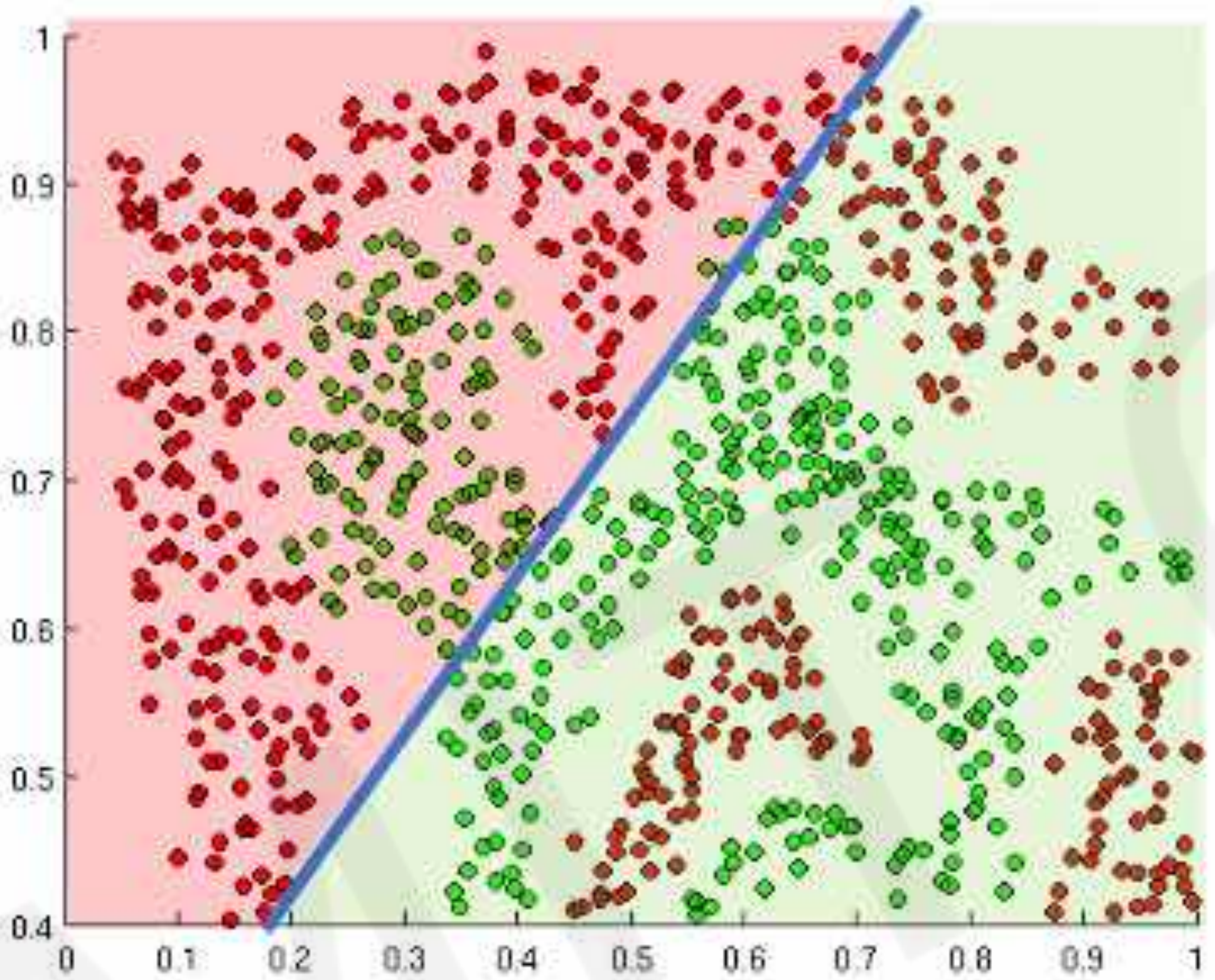
*The purpose of activation functions is to **introduce non-linearities** into the network*



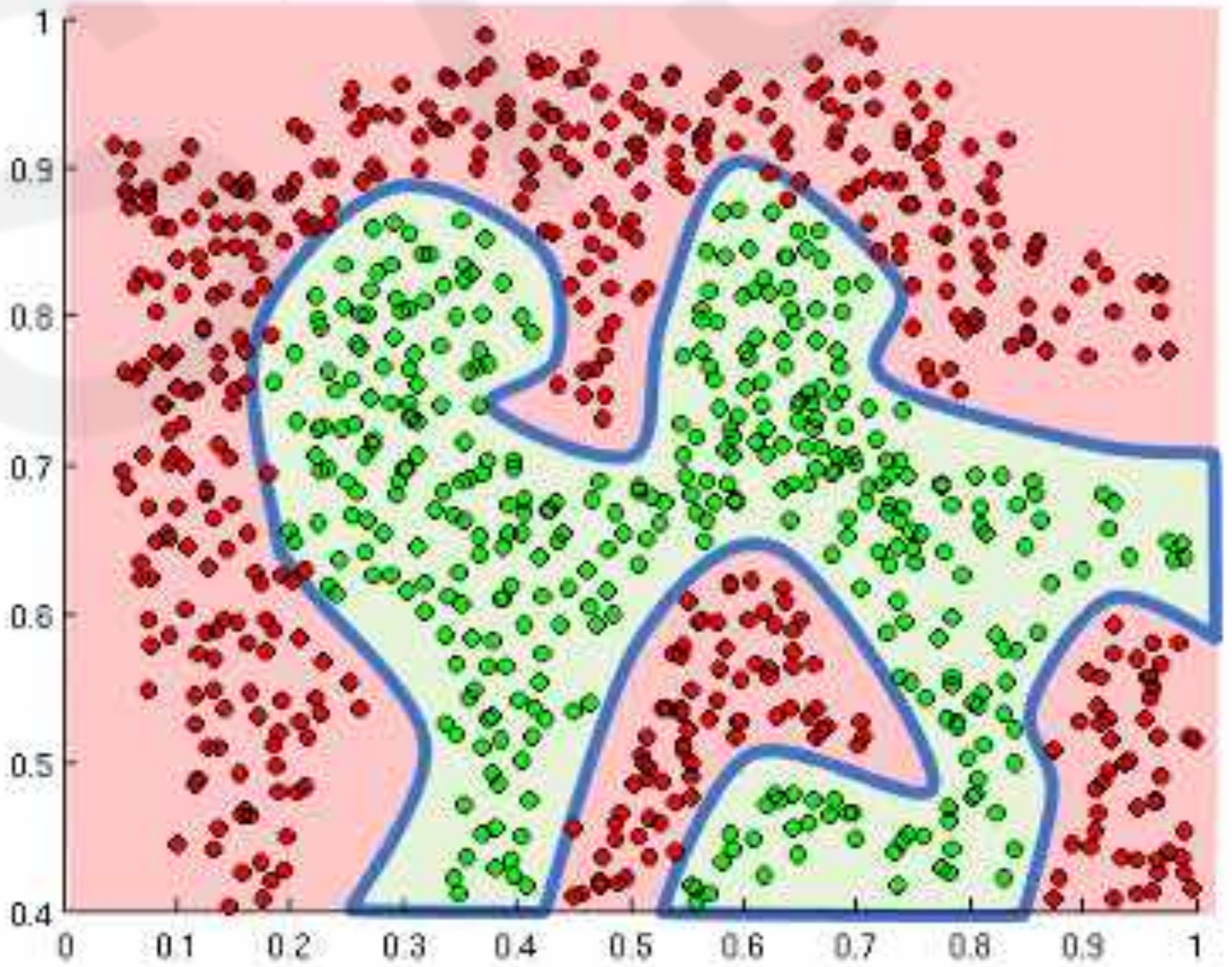
Linear activation functions produce linear decisions no matter the network size

Importance of Activation Functions

*The purpose of activation functions is to **introduce non-linearities** into the network*

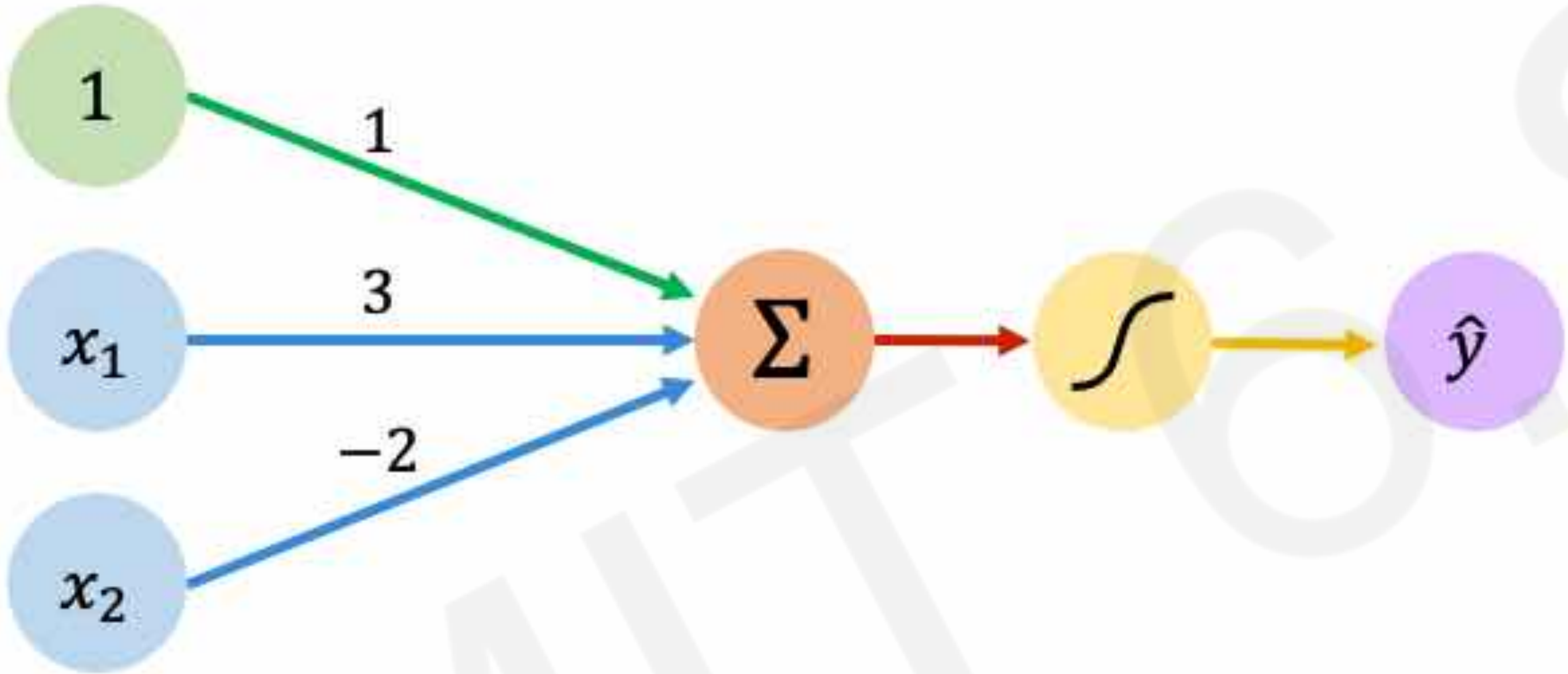


Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

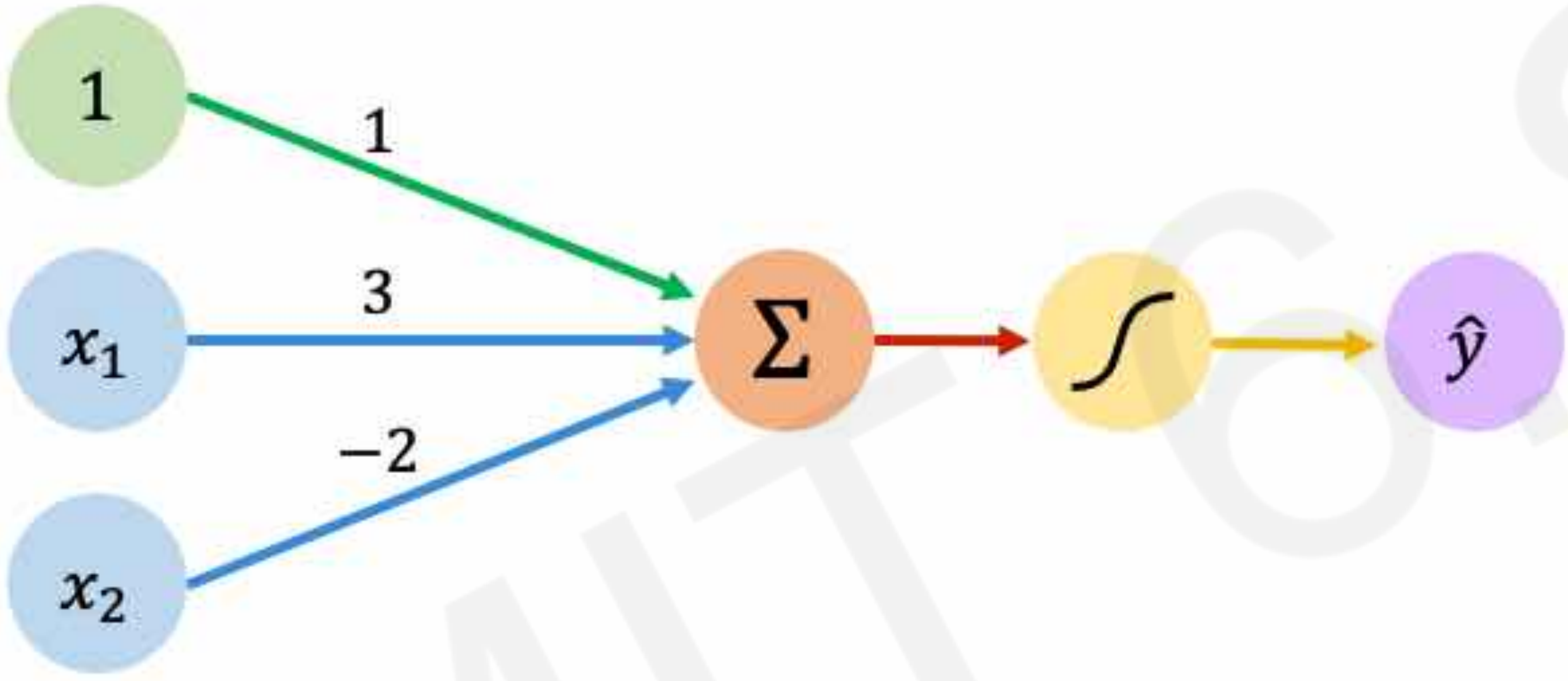
The Perceptron: Example



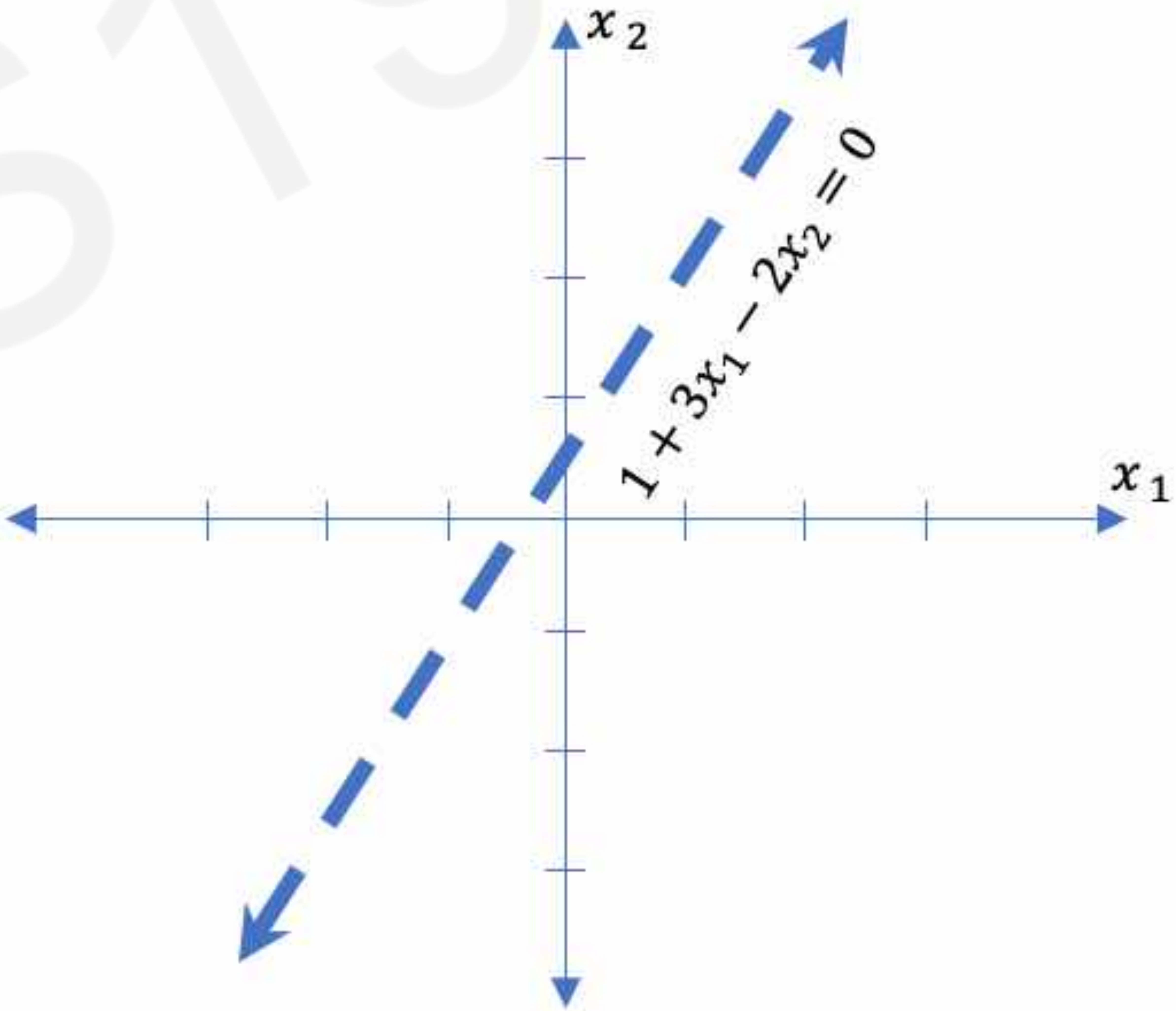
We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\begin{aligned}
\hat{y} &= g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right) \\
&= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix} \right) \\
\hat{y} &= g(\underbrace{1 + 3x_1 - 2x_2}_{\text{This is just a line in 2D!}})
\end{aligned}$$

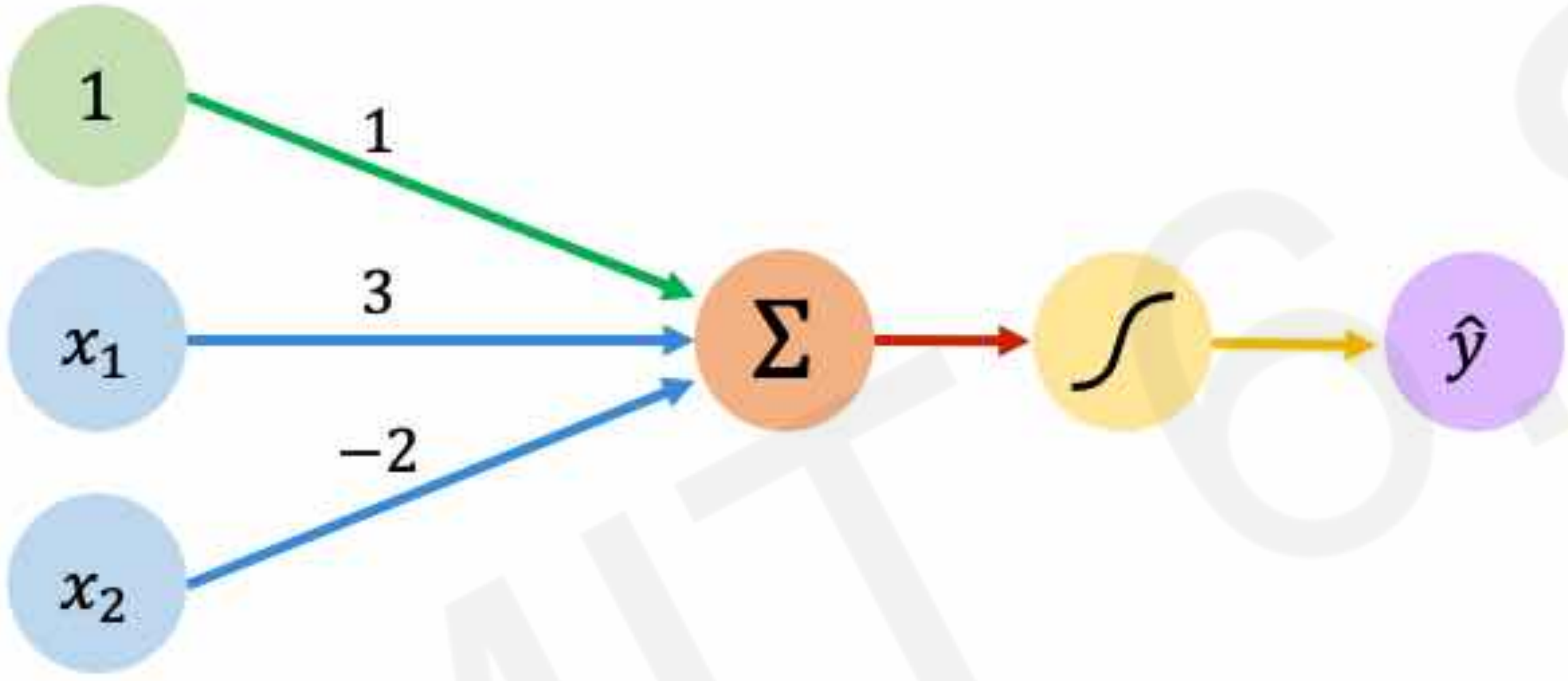
The Perceptron: Example



$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



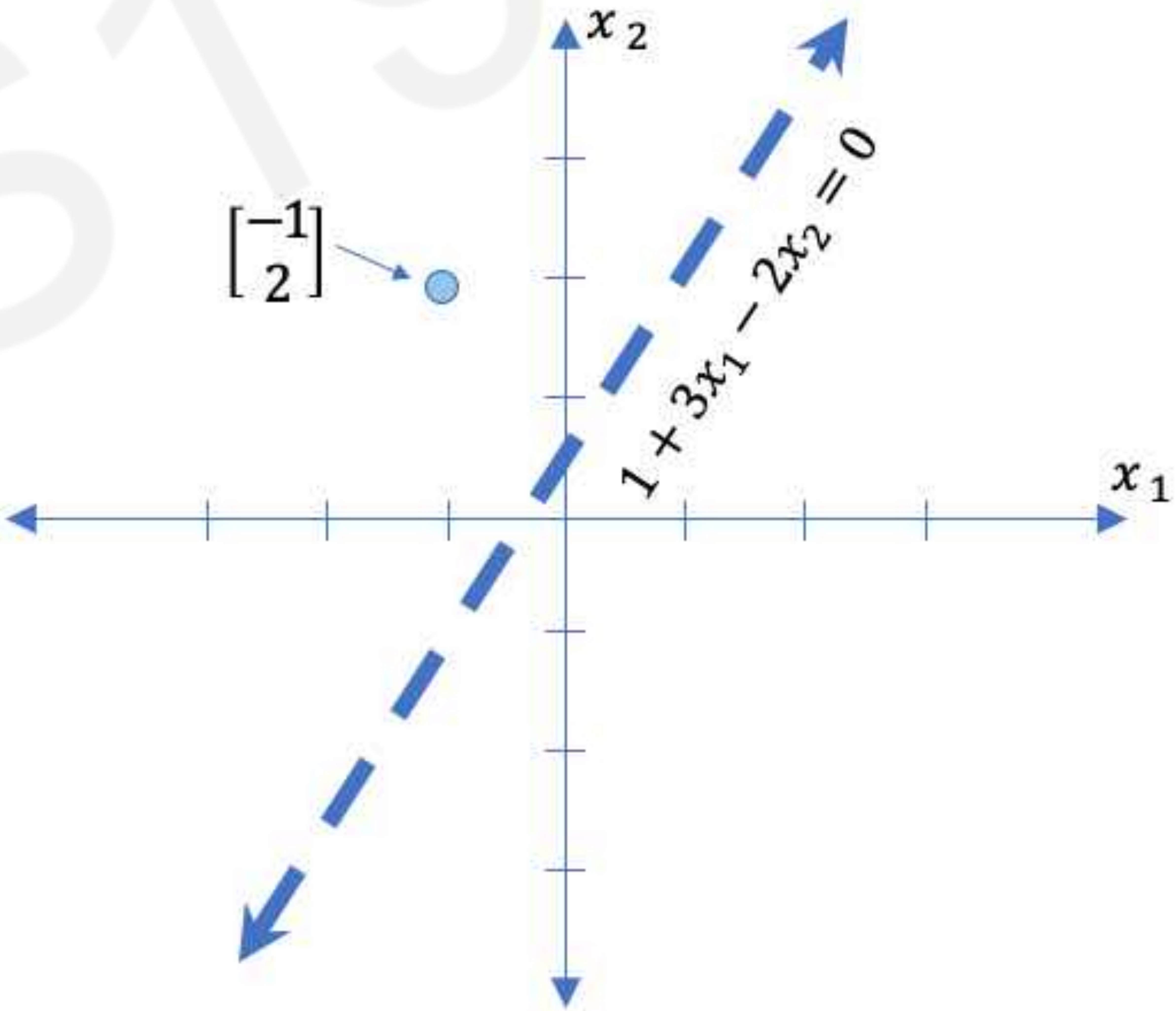
The Perceptron: Example



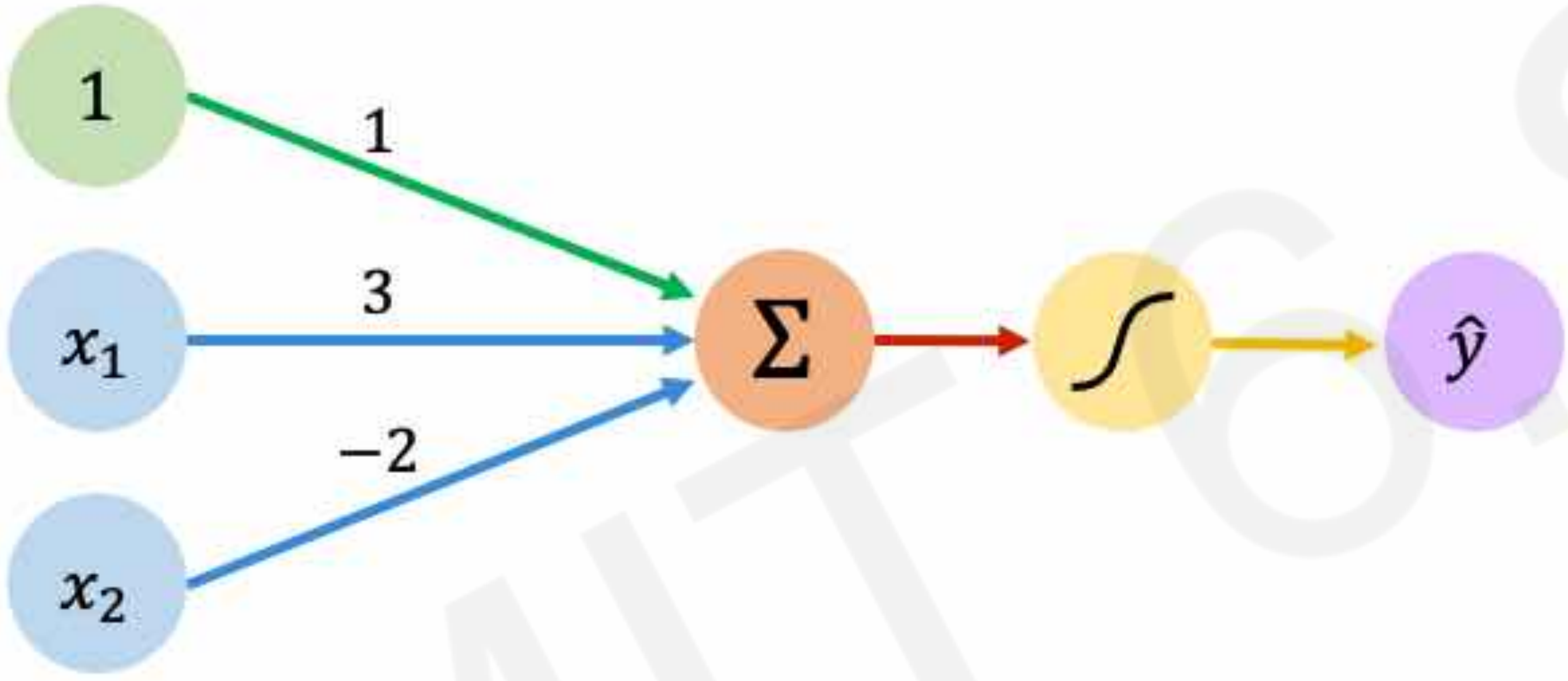
Assume we have input: $\boldsymbol{X} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

$$\hat{y} = g\big(1 + (3 * -1) - (2 * 2)\big)$$
$$= g(-6) \approx 0.002$$

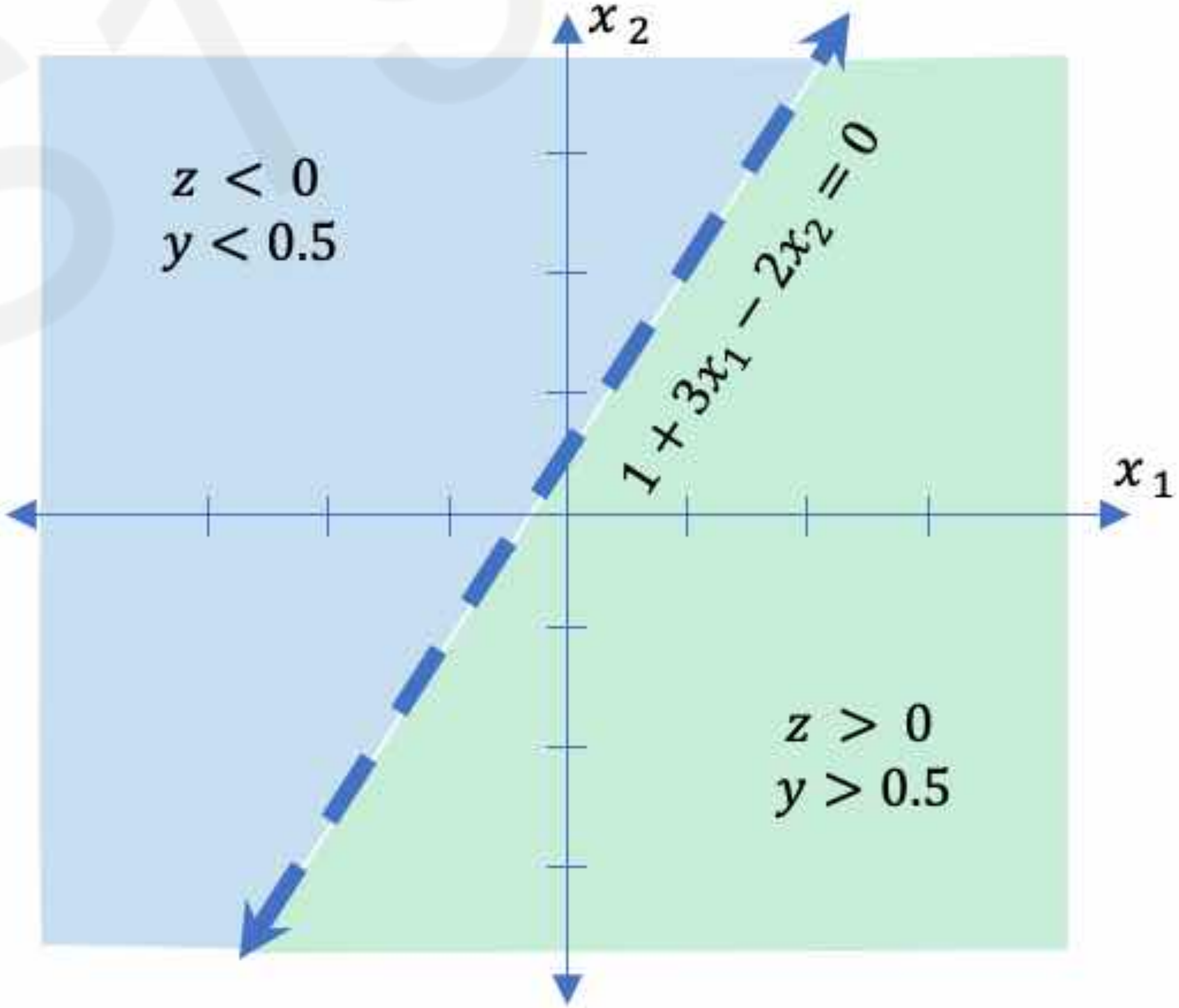
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



The Perceptron: Example



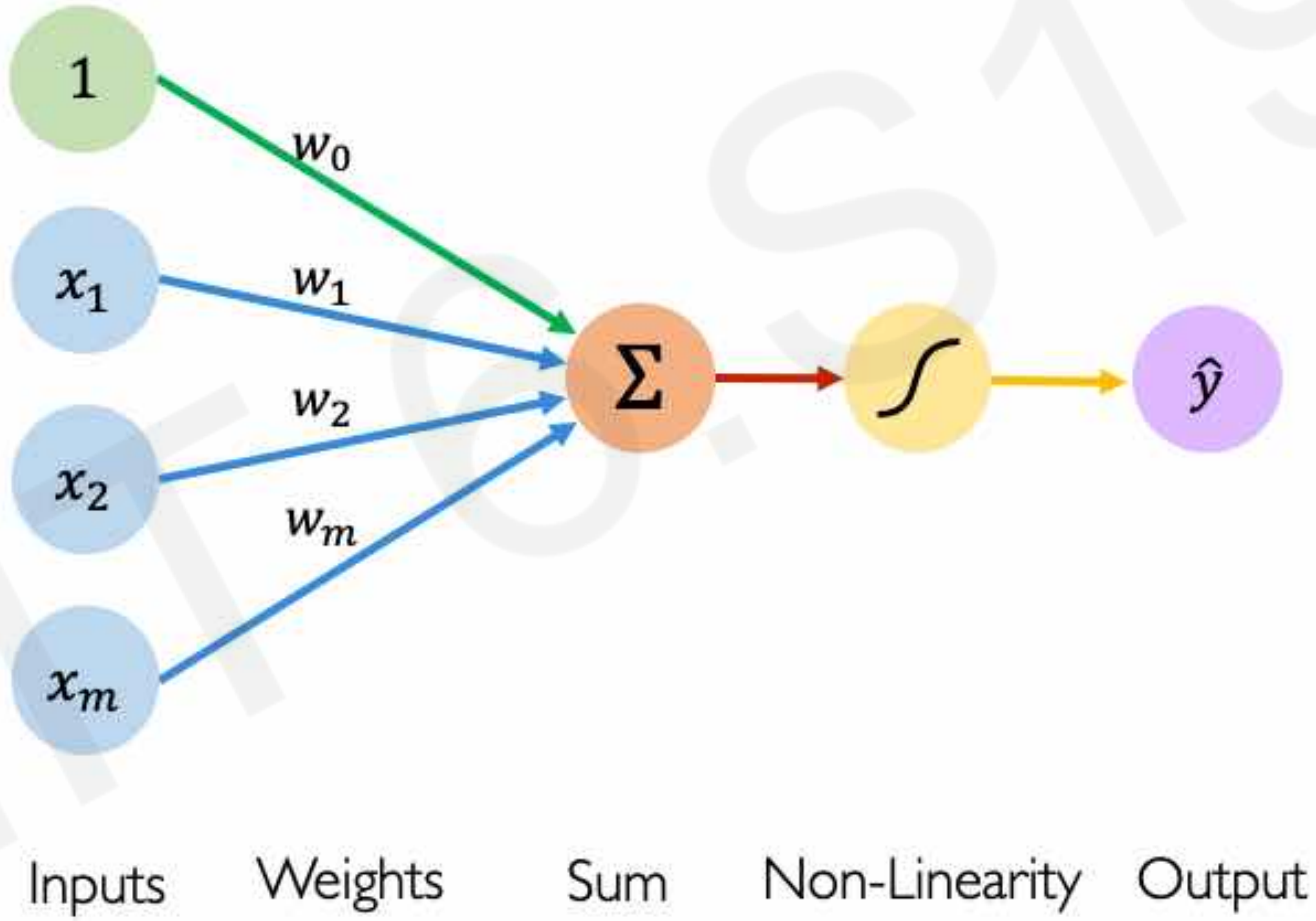
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



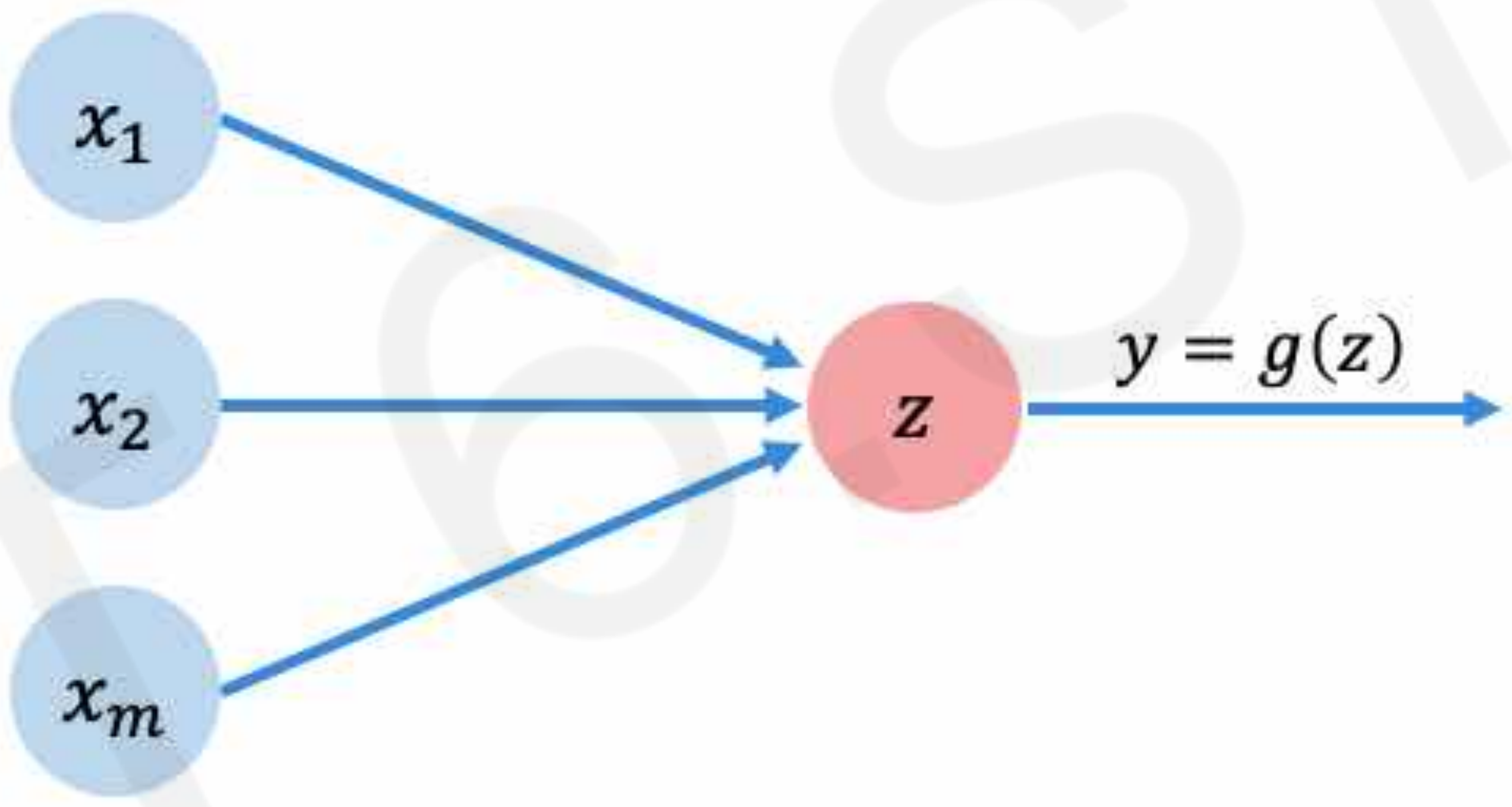
Building Neural Networks with Perceptrons

The Perceptron: Simplified

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



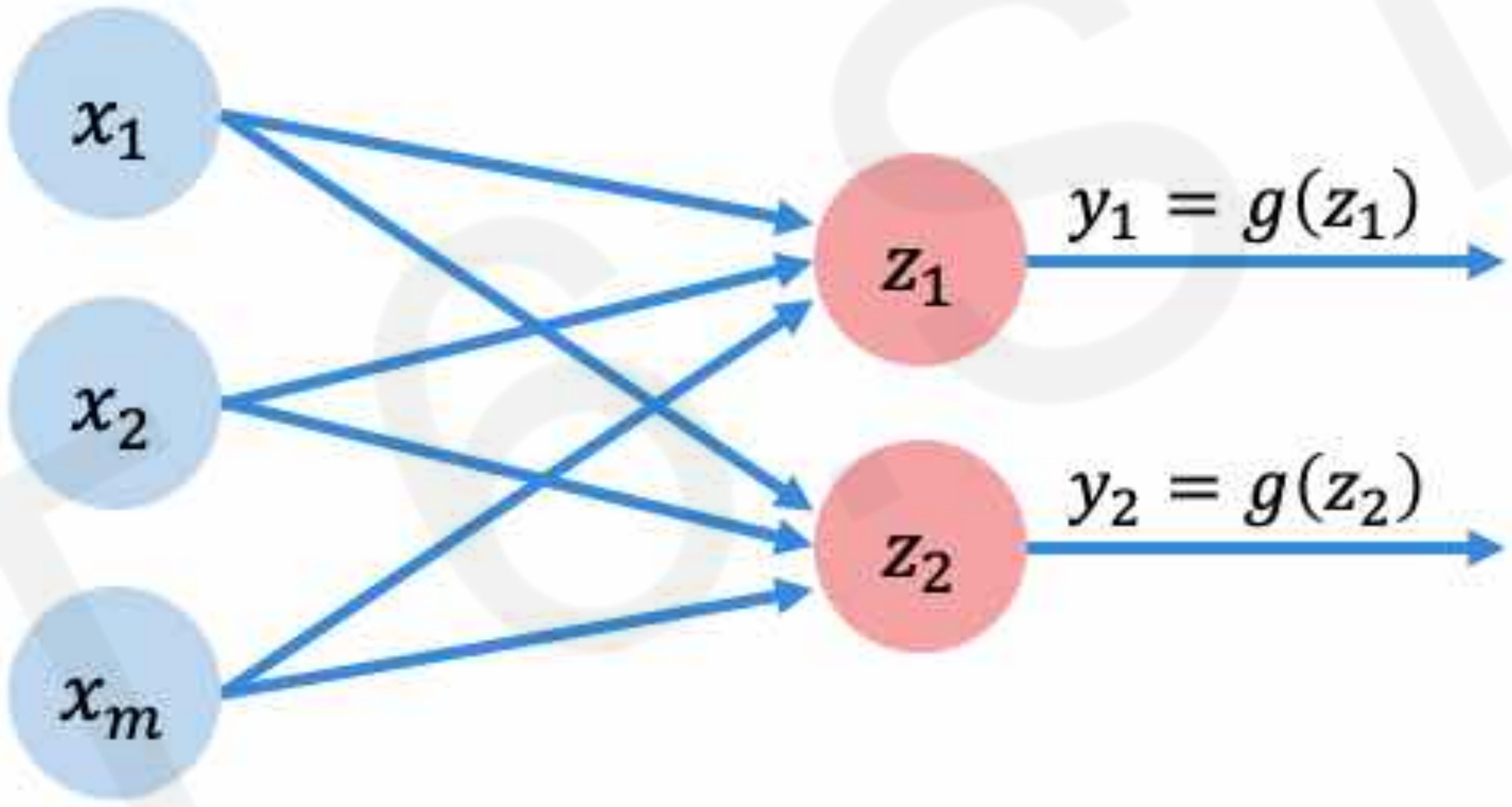
The Perceptron: Simplified



$$z = w_0 + \sum_{j=1}^{m} x_j \, w_j$$

Multi Output Perceptron

Because all inputs are densely connected to all outputs, these layers are called **Dense** layers



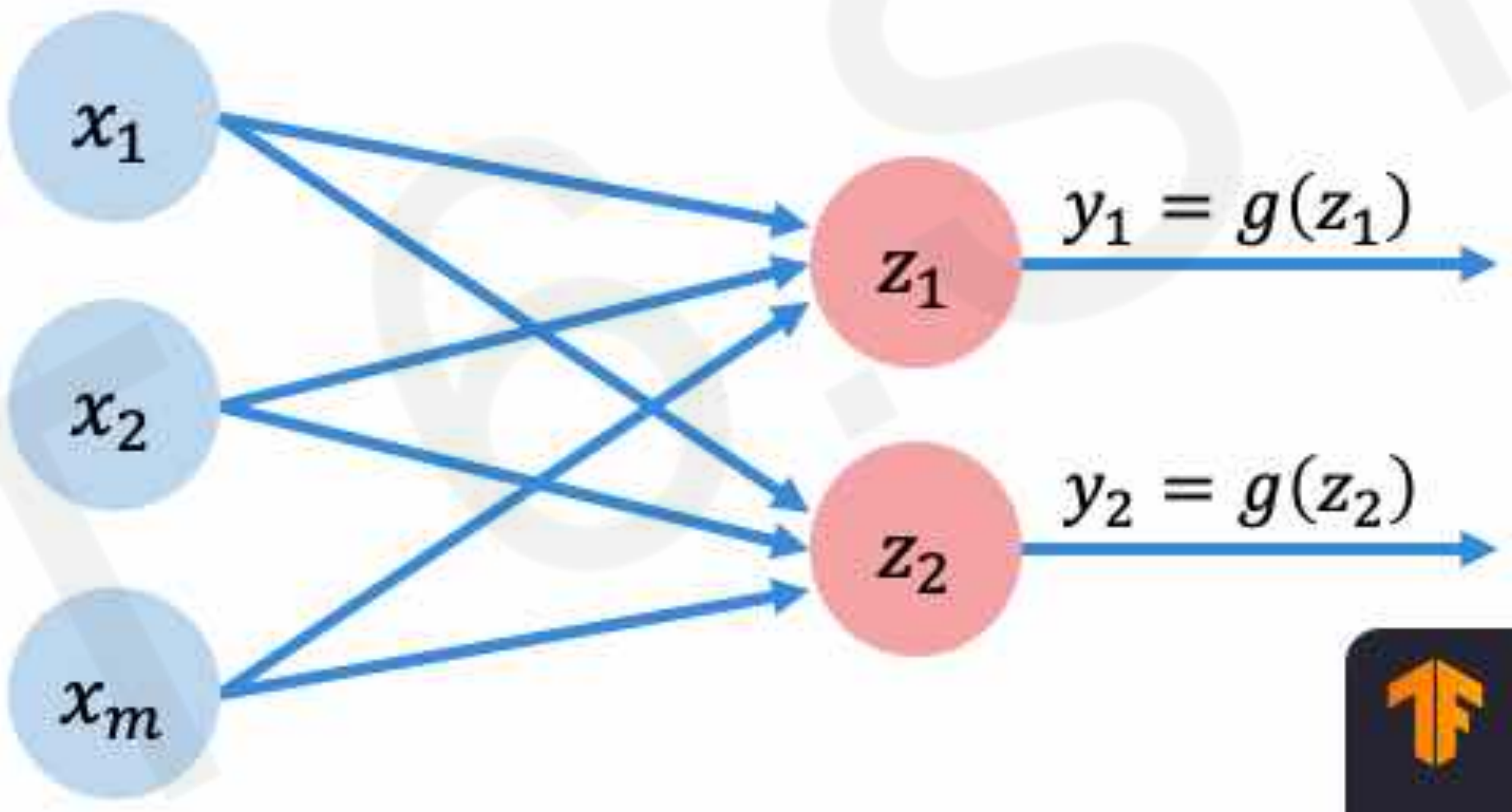
$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

Dense layer from scratch

```python
class MyDenseLayer(tf.keras.layers.Layer):
  def __init__(self, input_dim, output_dim):
    super(MyDenseLayer, self).__init__()

    # Initialize weights and bias
    self.W = self.add_weight([input_dim, output_dim])
    self.b = self.add_weight([1, output_dim])

  def call(self, inputs):
    # Forward propagate the inputs
    z = tf.matmul(inputs, self.W) + self.b

    # Feed through a non-linear activation
    output = tf.math.sigmoid(z)

    return output
```

Multi Output Perceptron

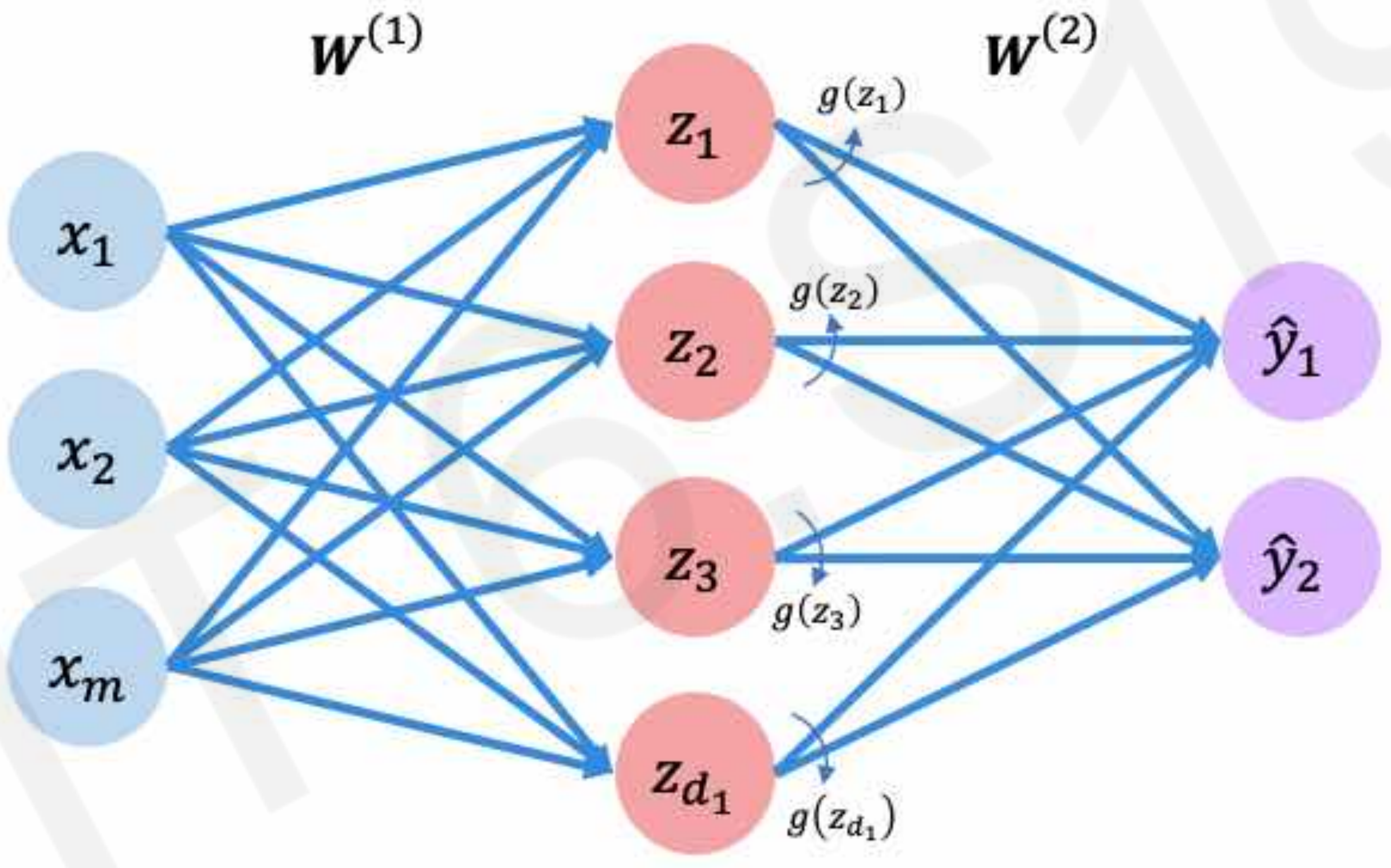
Because all inputs are densely connected to all outputs, these layers are called **Dense** layers




```python
import tensorflow as tf

layer = tf.keras.layers.Dense(
    units=2)
```

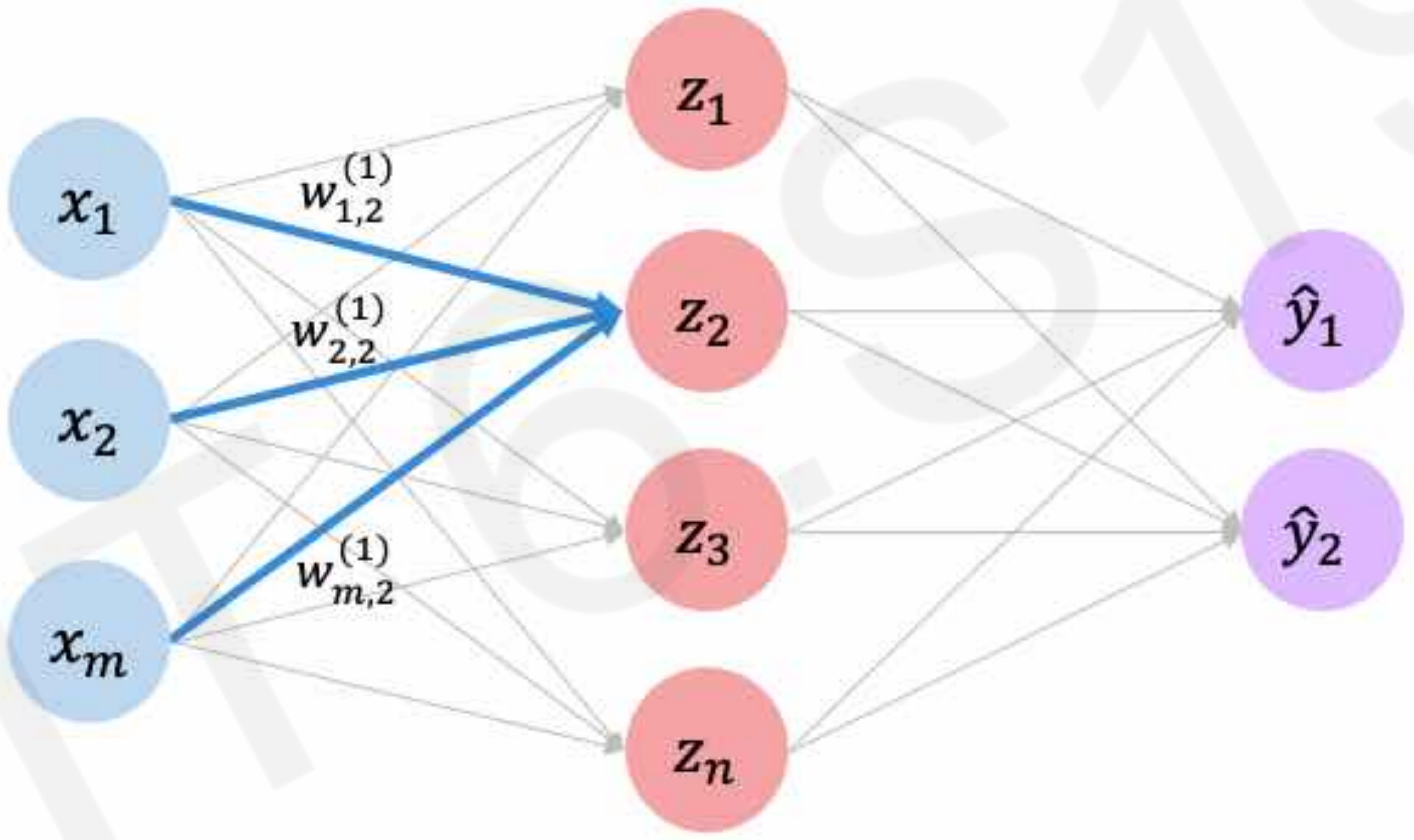
$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

Single Layer Neural Network



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j) \, w_{j,i}^{(2)}\right)$$

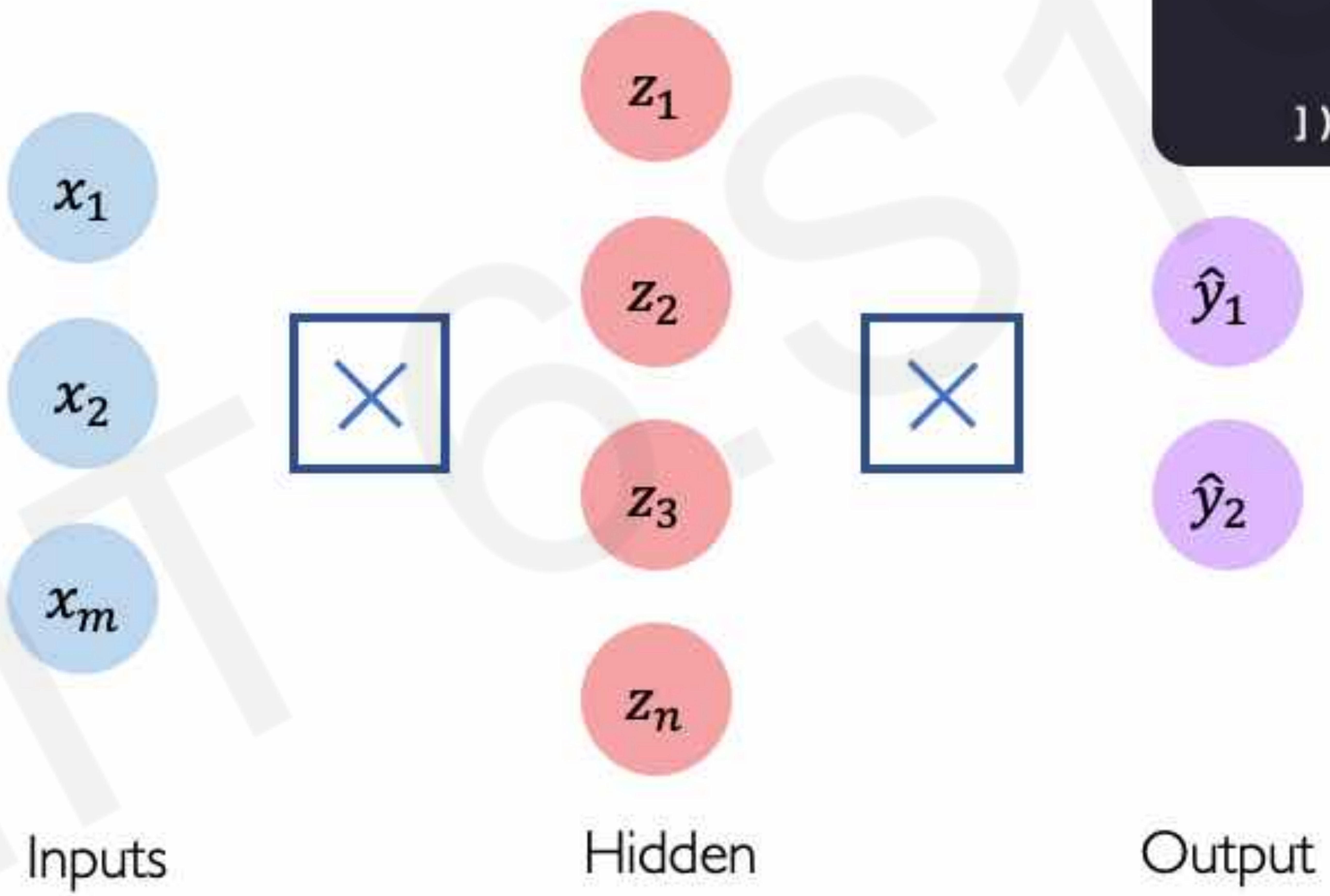
Single Layer Neural Network



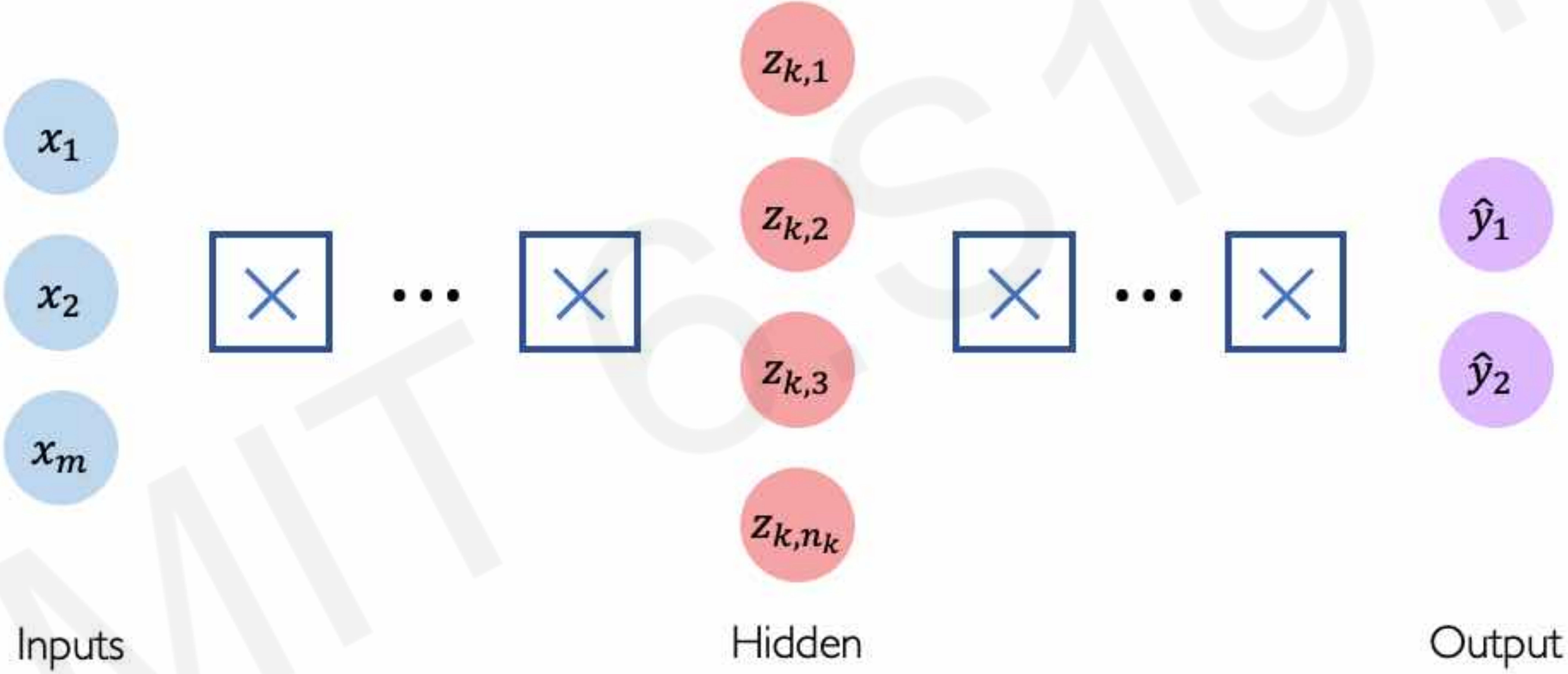
$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j\, w_{j,2}^{(1)}$$
$$= w_{0,2}^{(1)} + x_1\, w_{1,2}^{(1)} + x_2\, w_{2,2}^{(1)} + x_m\, w_{m,2}^{(1)}$$

Multi Output Perceptron

```python
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(n),
    tf.keras.layers.Dense(2)
])
```

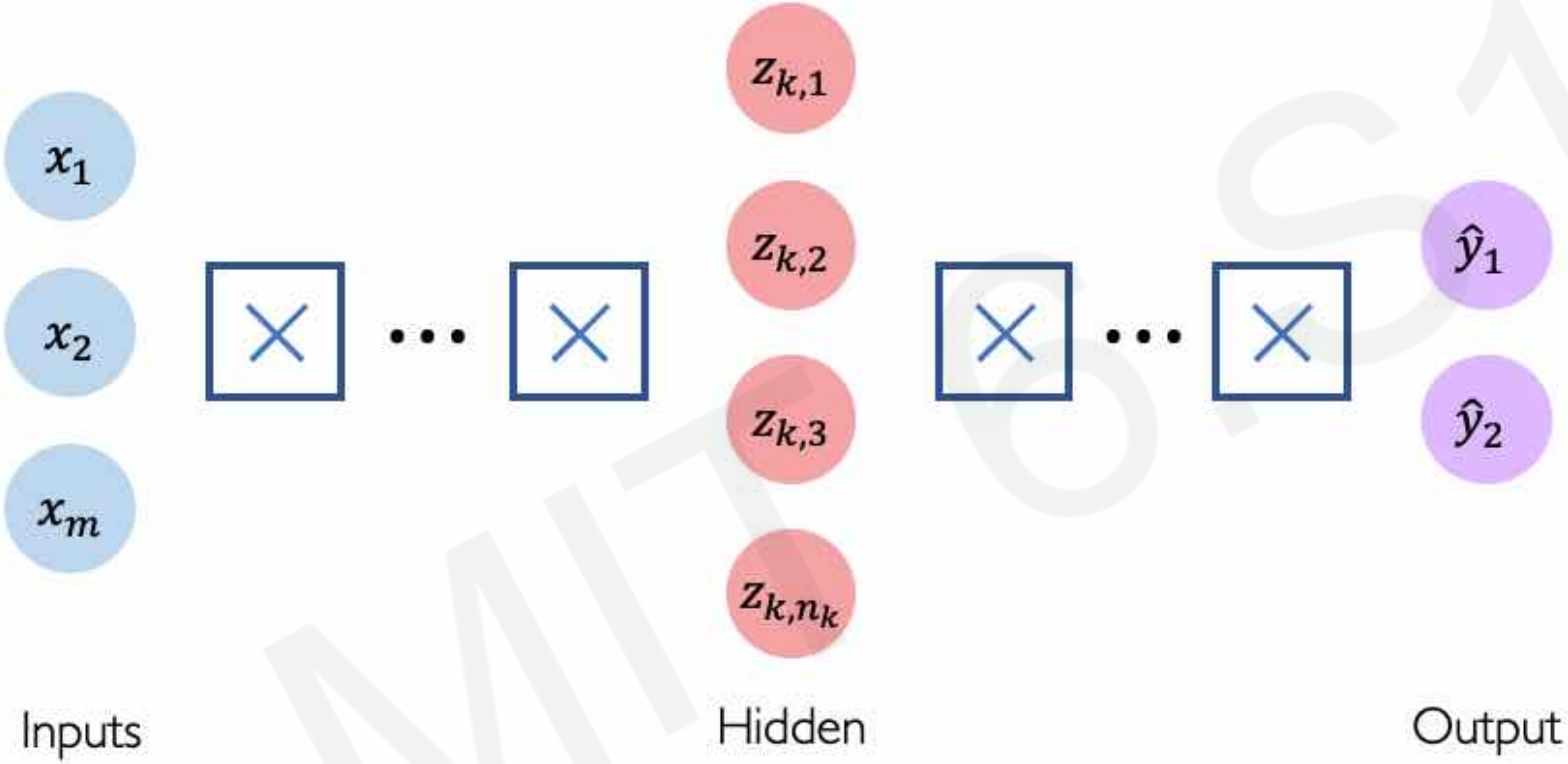


Deep Neural Network



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Deep Neural Network




```python
import tensorflow as tf

model = tf.keras.Sequential([
  tf.keras.layers.Dense(n₁),
  tf.keras.layers.Dense(n₂),
  ⋮
  tf.keras.layers.Dense(2)
])
```

$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Applying Neural Networks

Example Problem

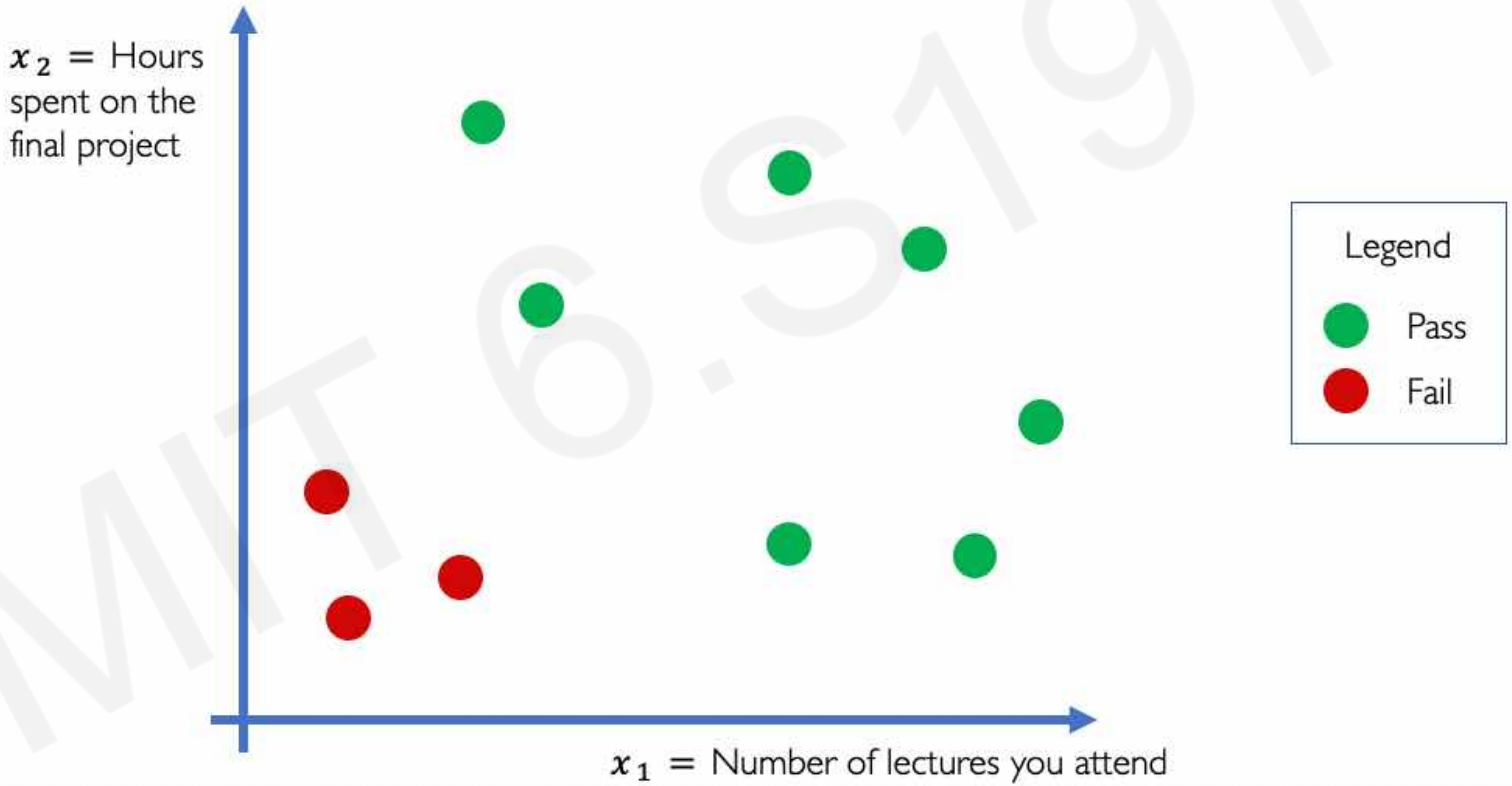
Will I pass this class?

Let's start with a simple two feature model

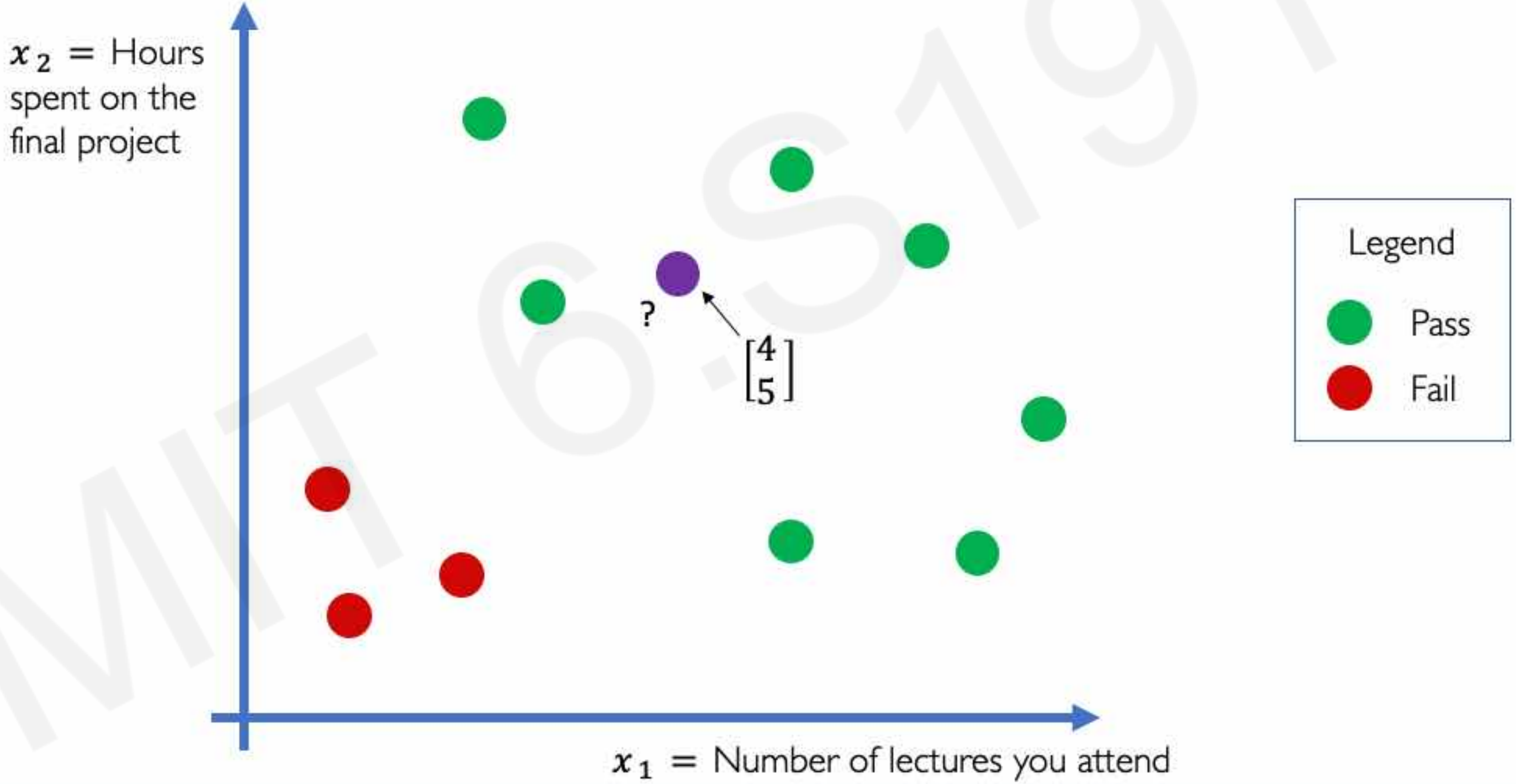
x_1 = Number of lectures you attend

x_2 = Hours spent on the final project

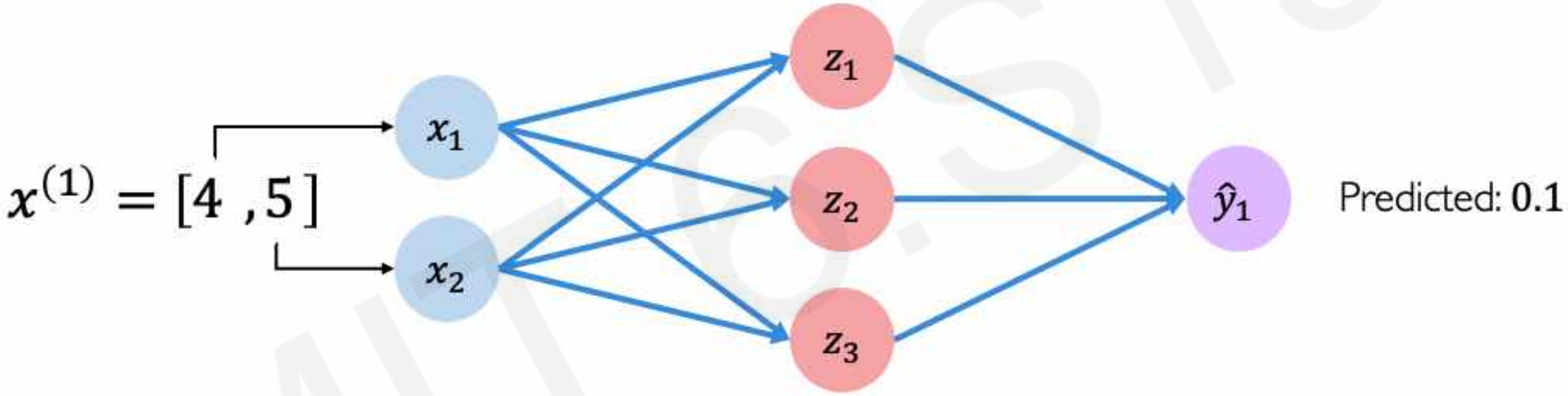
Example Problem: Will I pass this class?



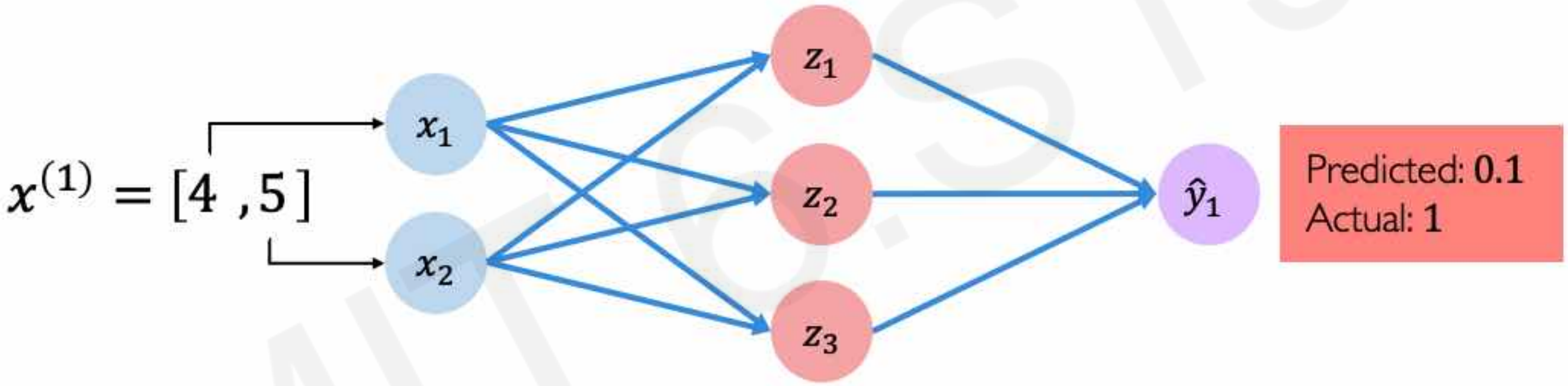
Example Problem: Will I pass this class?



Example Problem: Will I pass this class?

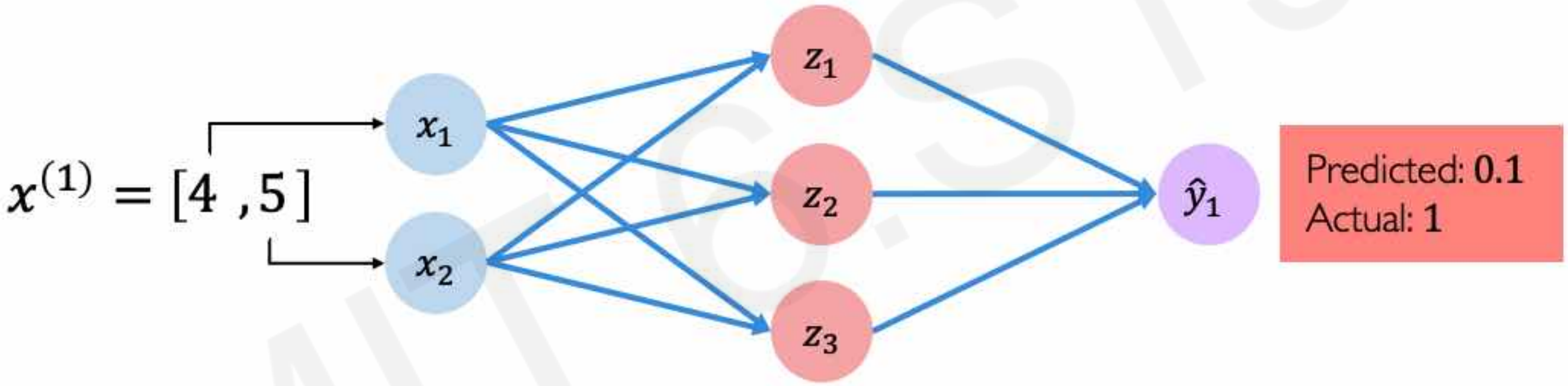


Example Problem: Will I pass this class?



Quantifying Loss

*The **loss** of our network measures the cost incurred from incorrect predictions*

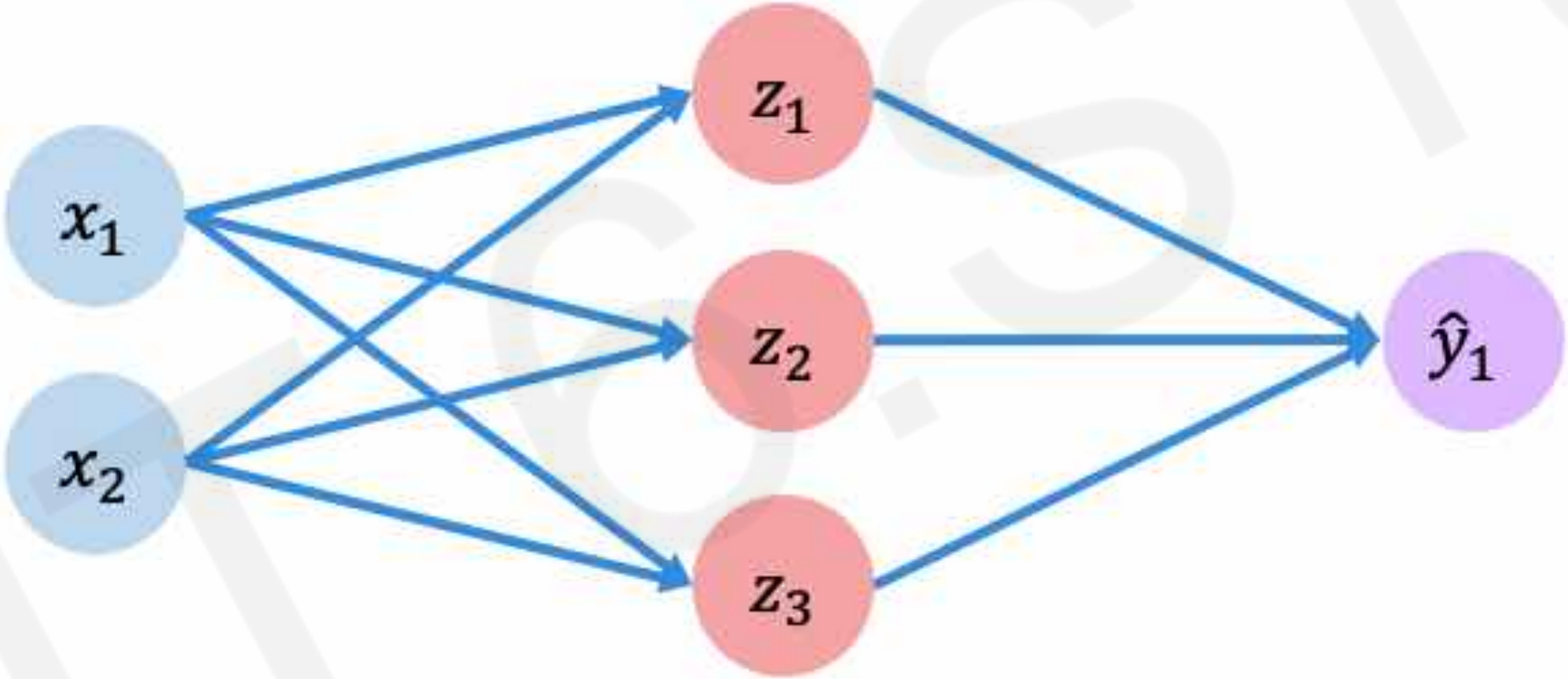


$$\mathcal{L}\left(\underbrace{f\left(x^{(i)};\boldsymbol{W}\right)}_{\text{Predicted}},\underbrace{y^{(i)}}_{\text{Actual}}\right)$$

Empirical Loss

*The **empirical loss** measures the total loss over our entire dataset*

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\begin{array}{ccc} f(x) & & y \\ \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} & \begin{matrix} \times \\ \times \\ \checkmark \\ \end{matrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \end{array}$$

$$\boldsymbol{J}(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\big(\underbrace{f\big(x^{(i)}; \boldsymbol{W}\big)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}\big)$$

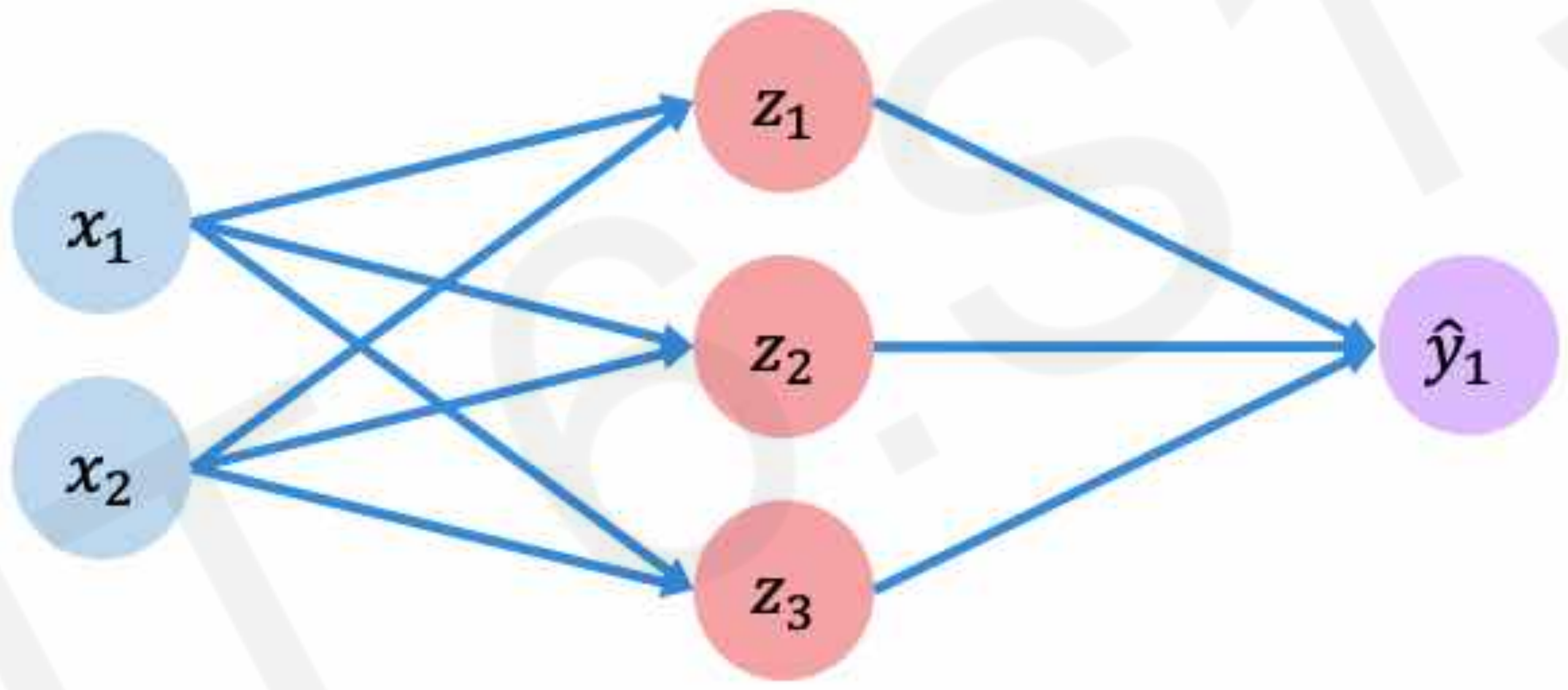
Also known as:

- Objective function
- Cost function
- Empirical Risk

Binary Cross Entropy Loss

Cross entropy loss *can be used with models that output a probability between 0 and 1*

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\begin{array}{ccc} f(x) & & y \\ \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} & \begin{array}{c} \times \\ \times \\ \checkmark \\ \end{array} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \end{array}$$

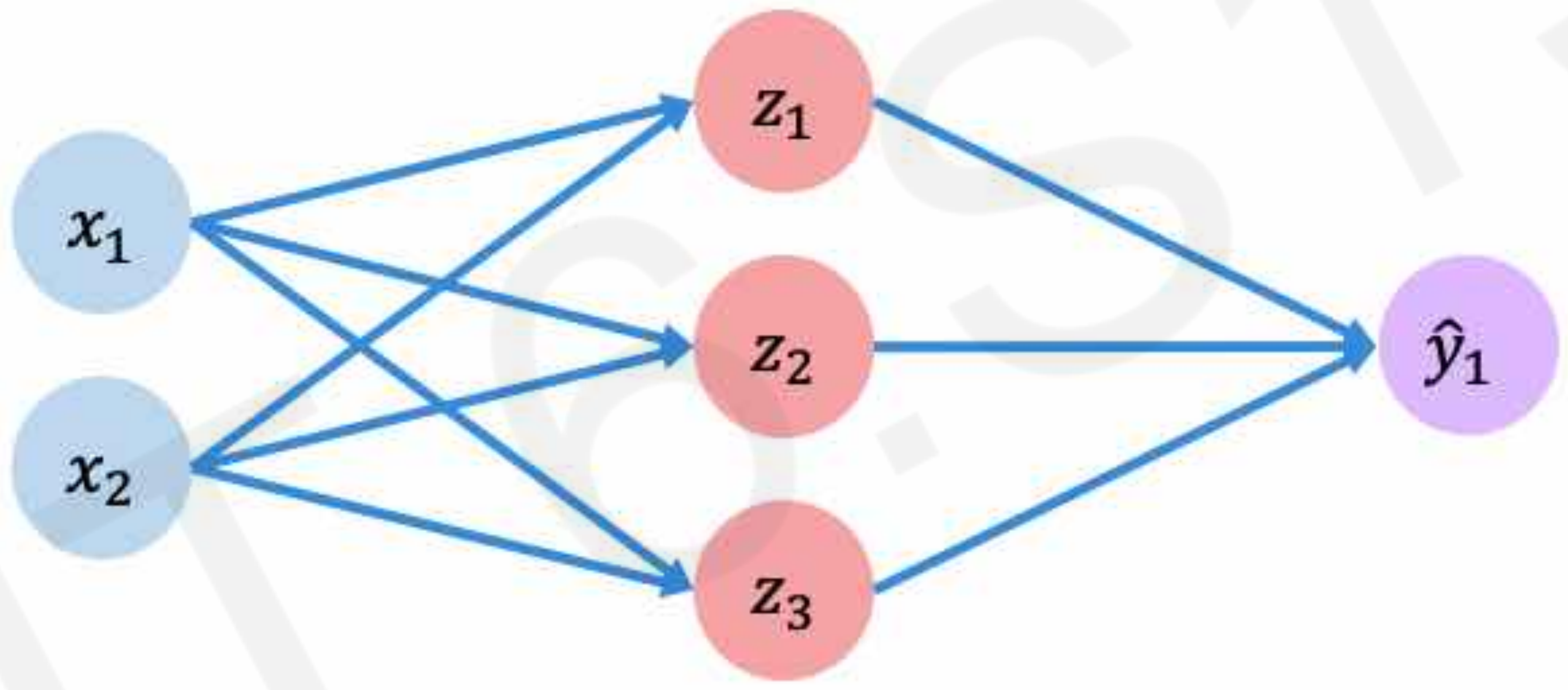
$$J(\boldsymbol{W}) = -\frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log\Big(\underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big) + (1-\underbrace{y^{(i)}}_{\text{Actual}}) \log\Big(1-\underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big)$$

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(y, predicted) )
```

Mean Squared Error Loss

Mean squared error loss *can be used with regression models that output continuous real numbers*

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



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

Final Grades (percentage)

$$\boldsymbol{J}(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\Big(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big)^2$$

```
loss = tf.reduce_mean( tf.square(tf.subtract(y, predicted)) )
loss = tf.keras.losses.MSE( y, predicted )
```

6.S191: Introduction to Deep Learning

Lab 1: Introduction to TensorFlow and Music Generation with RNNs

Link to download labs:
http://introtodeeplearning.com#schedule

1. Open the lab in Google Colab
2. Start executing code blocks and filling in the #TODOs
3. Need help? Come to the class Gather.Town or 10-250!