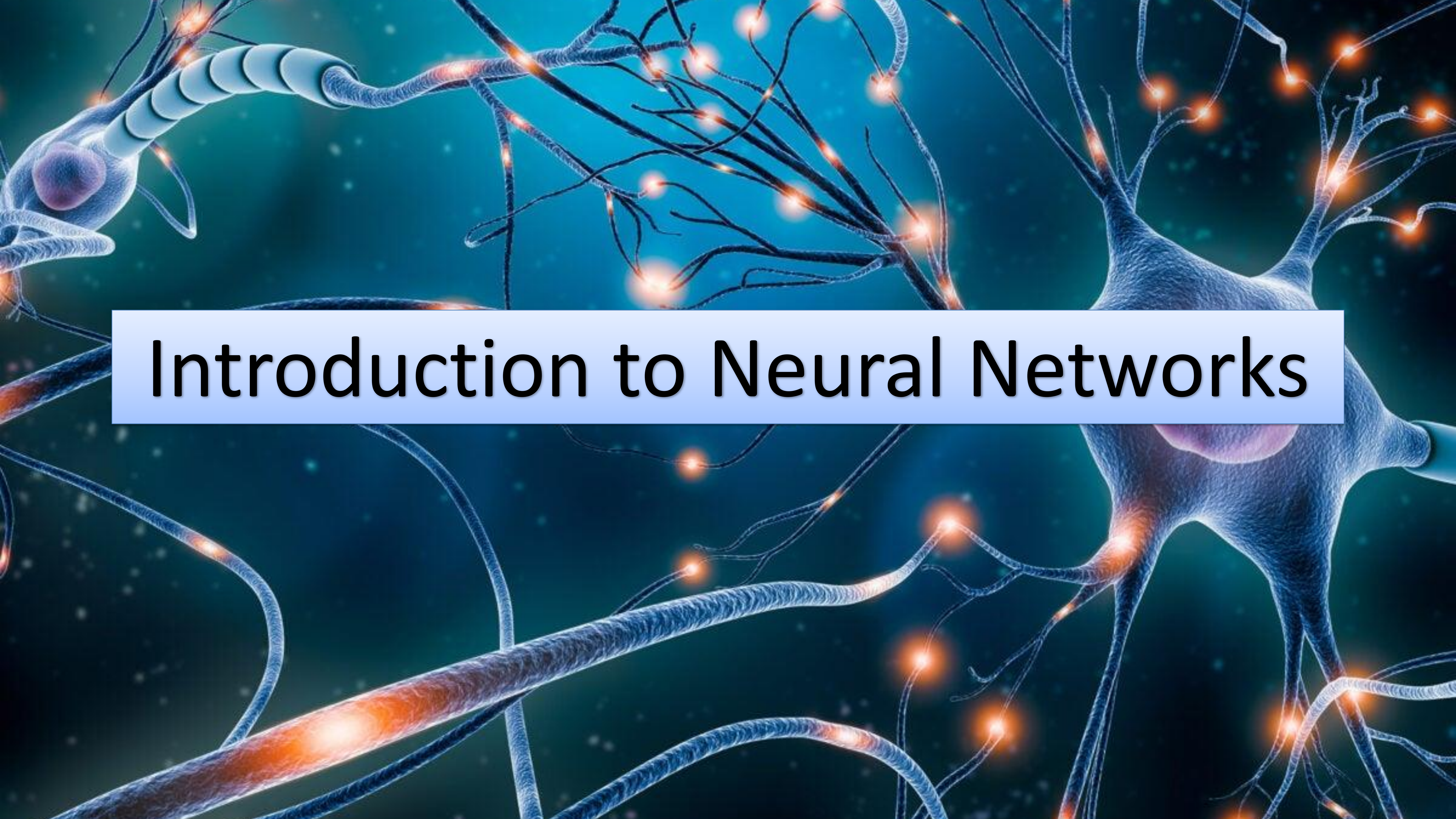
Introduction to Neural Networks



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

MIT 6.S191

Why Deep Learning and Why Now?

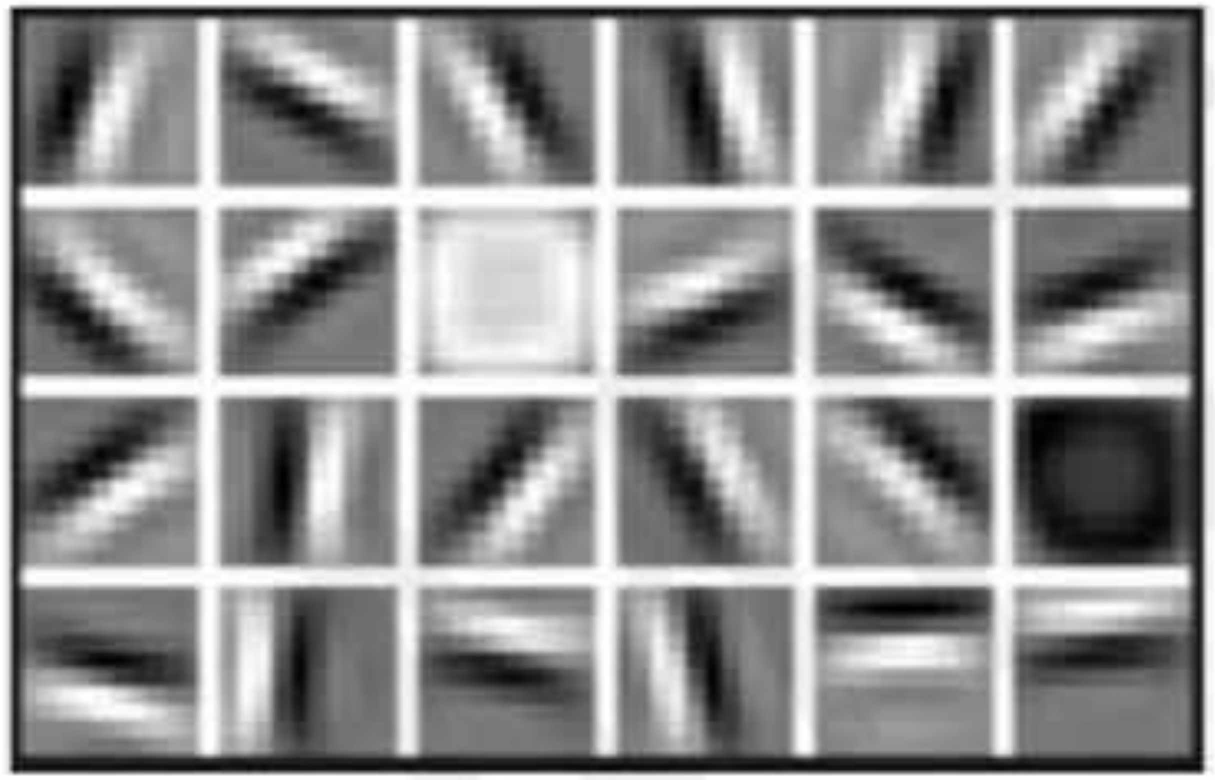
MIT 6.S191

Why Deep Learning?

Hand engineered features are time consuming, brittle, and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



Lines & Edges

Mid Level Features

Eyes & Nose & Ears

High Level Features

Facial Structure

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

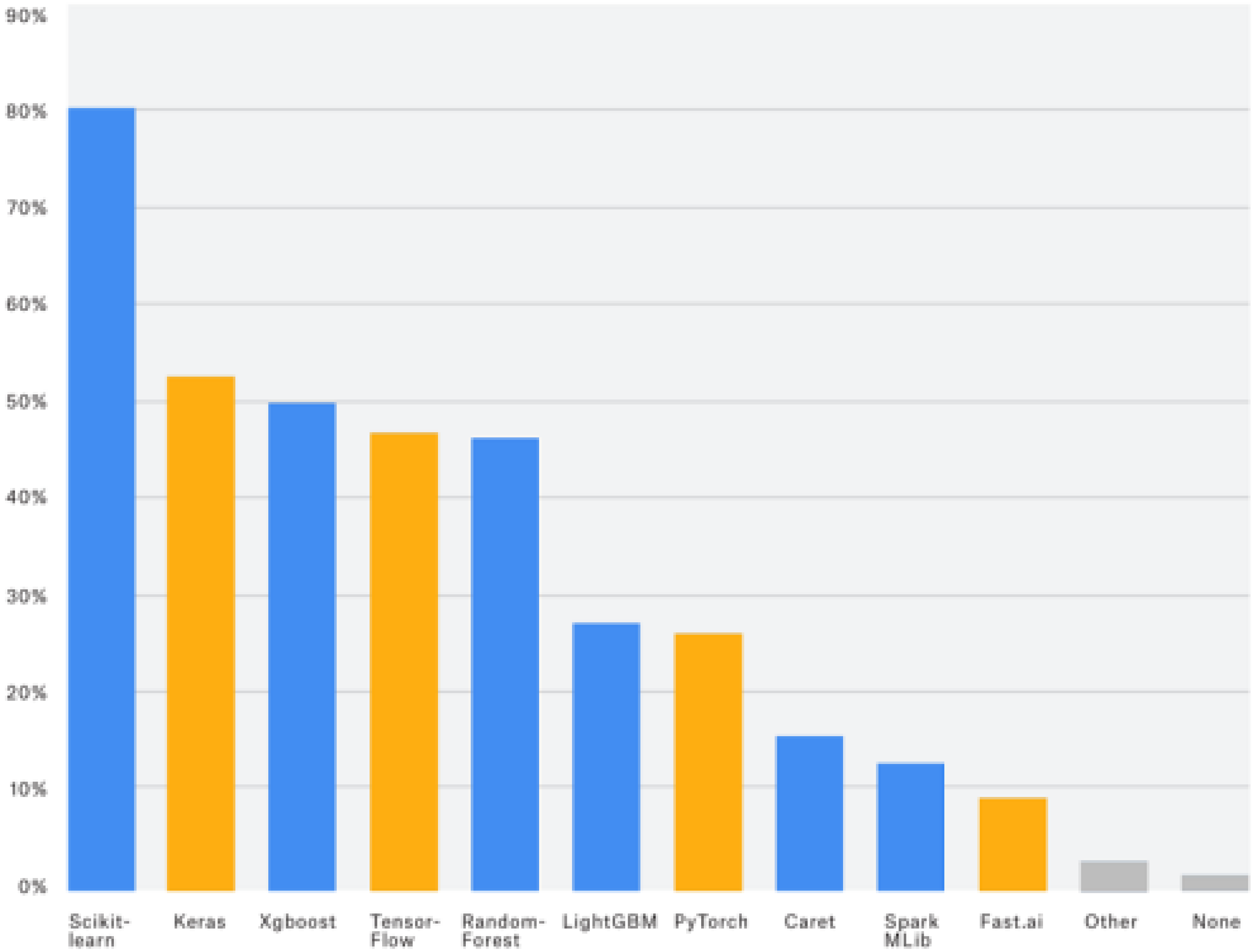
WIKIPEDIA
The Free Encyclopedia

2. Hardware
- Graphics Processing Units (GPUs)
- Massively Parallelizable

3. Software
- Improved Techniques
- New Models
- Toolboxes

TensorFlow

MIT 6.S191



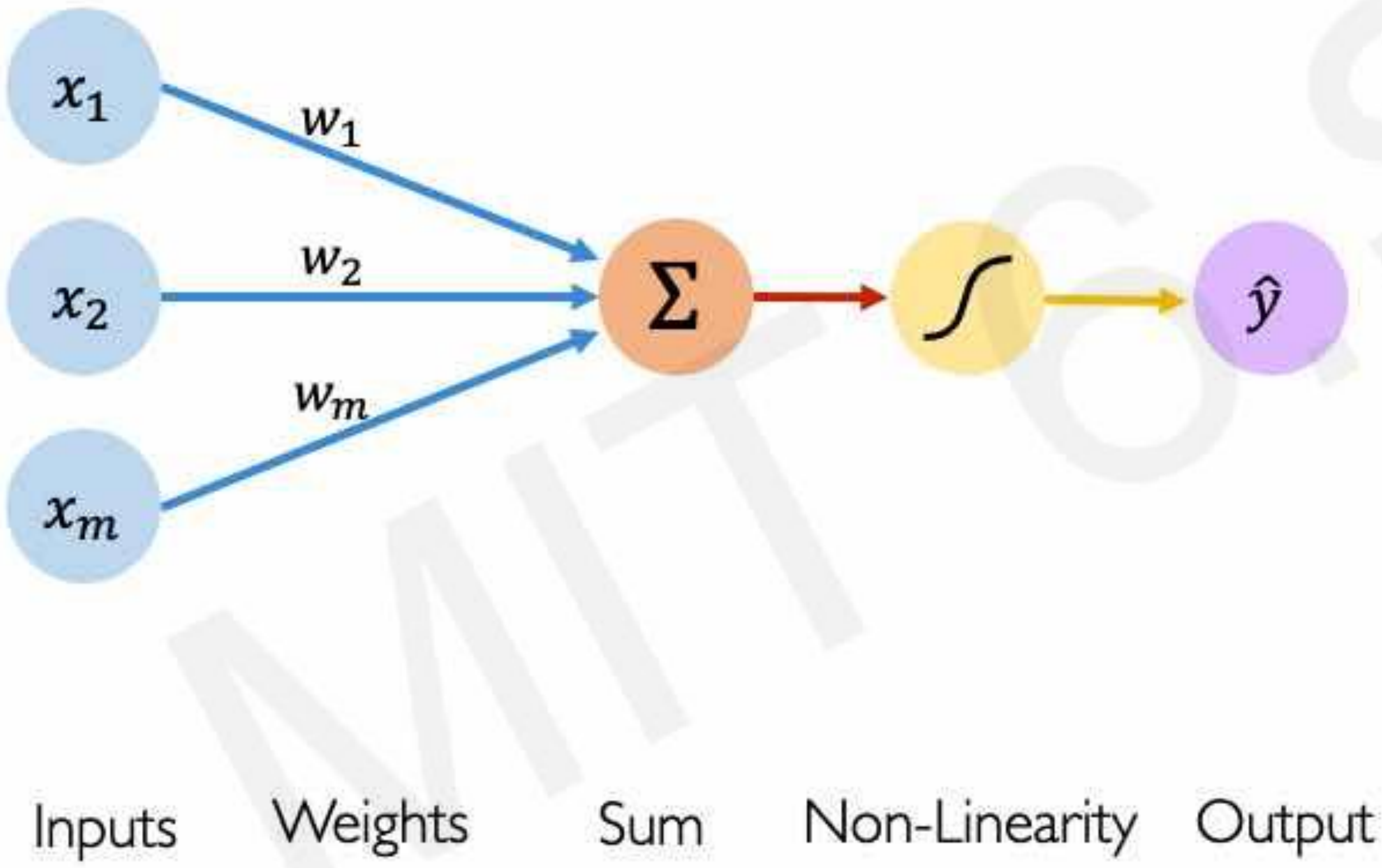
Trends in Deep Learning

The Perceptron

The structural building block of deep learning

MIT 6.S191

The Perceptron: Forward Propagation



Output

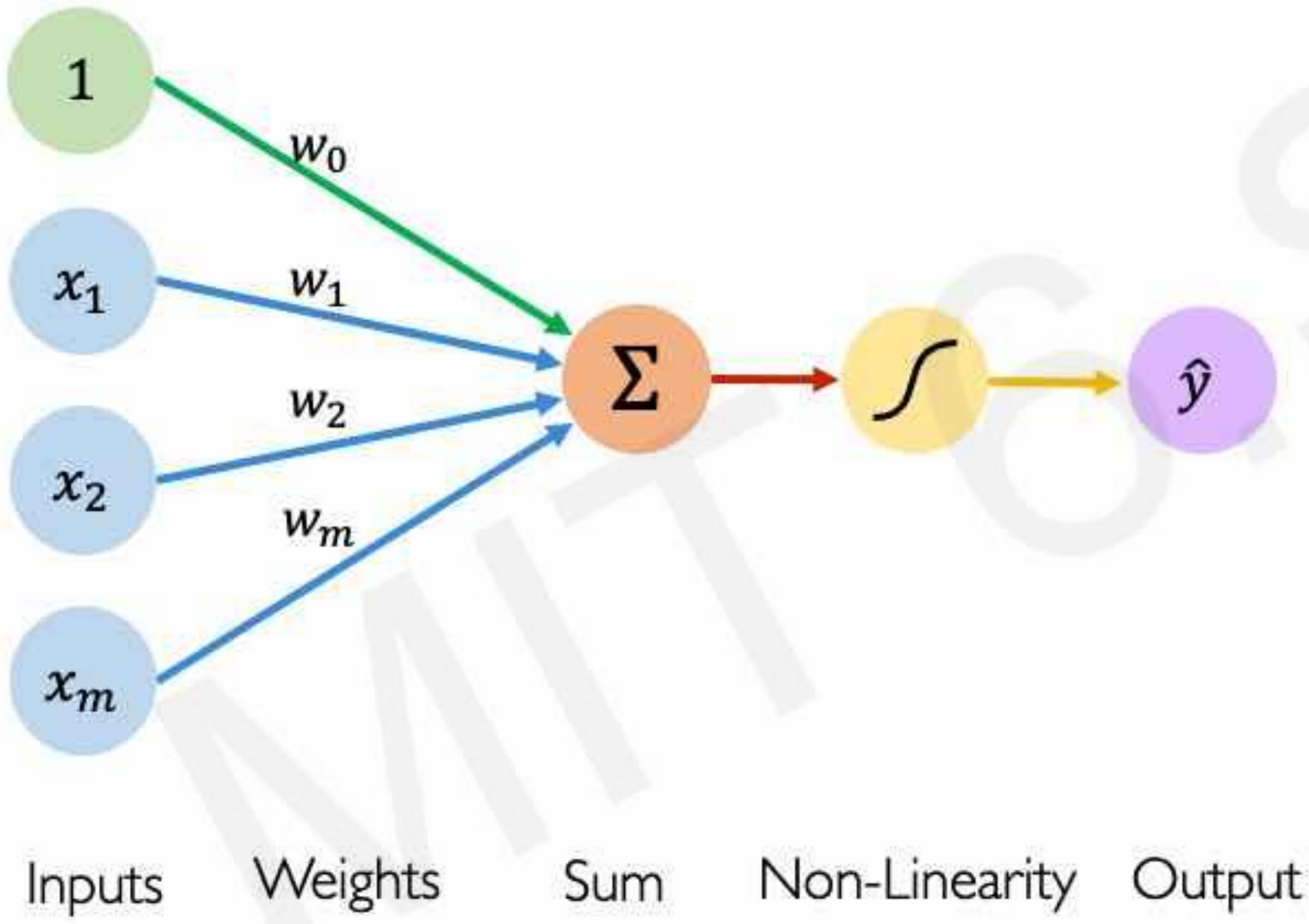
Linear combination of inputs

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

Non-linear activation function

MIT 6.S191

The Perceptron: Forward Propagation



Output

Linear combination of inputs

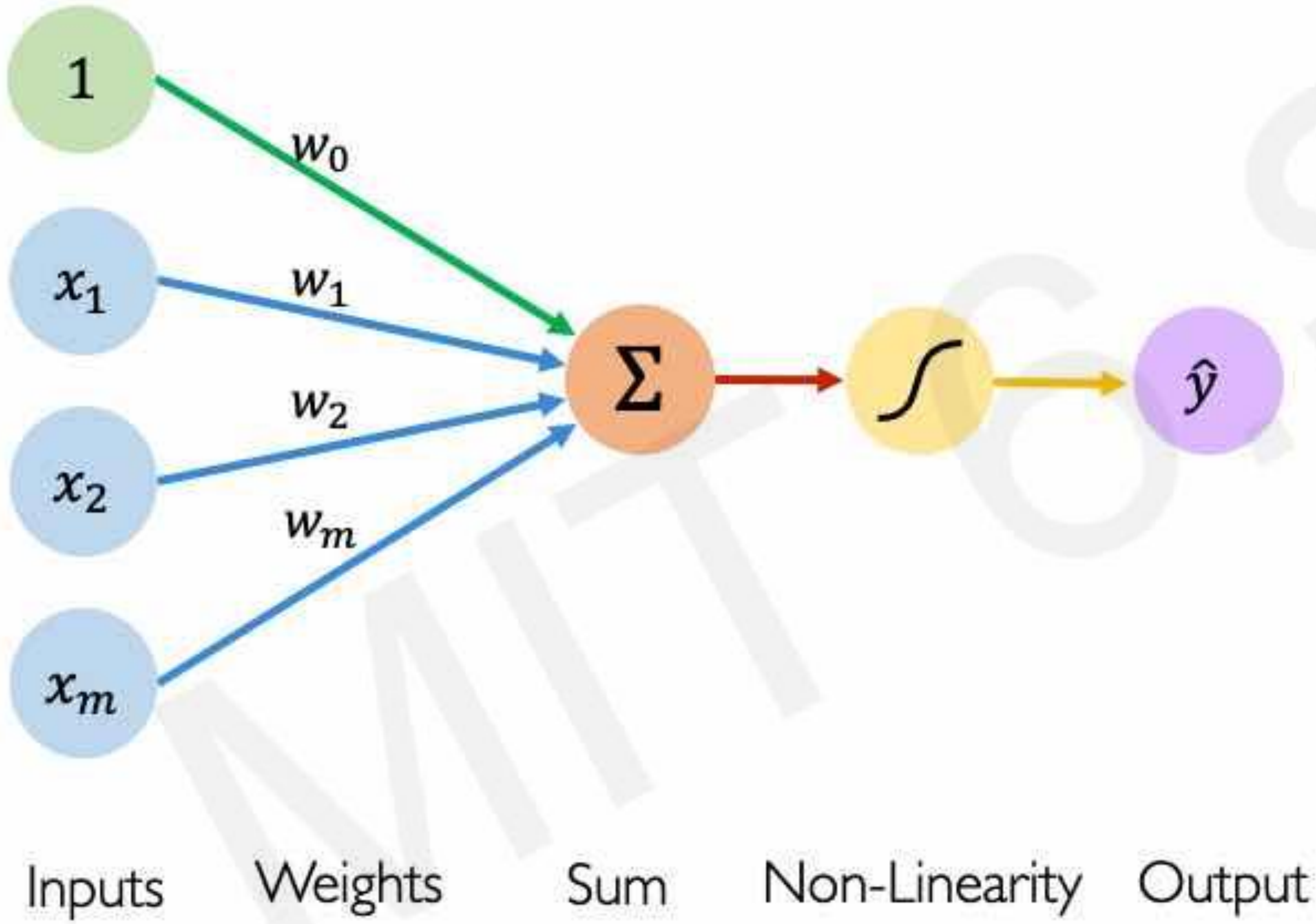
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

Non-linear activation function

Bias

MIT 6.S191

The Perceptron: Forward Propagation



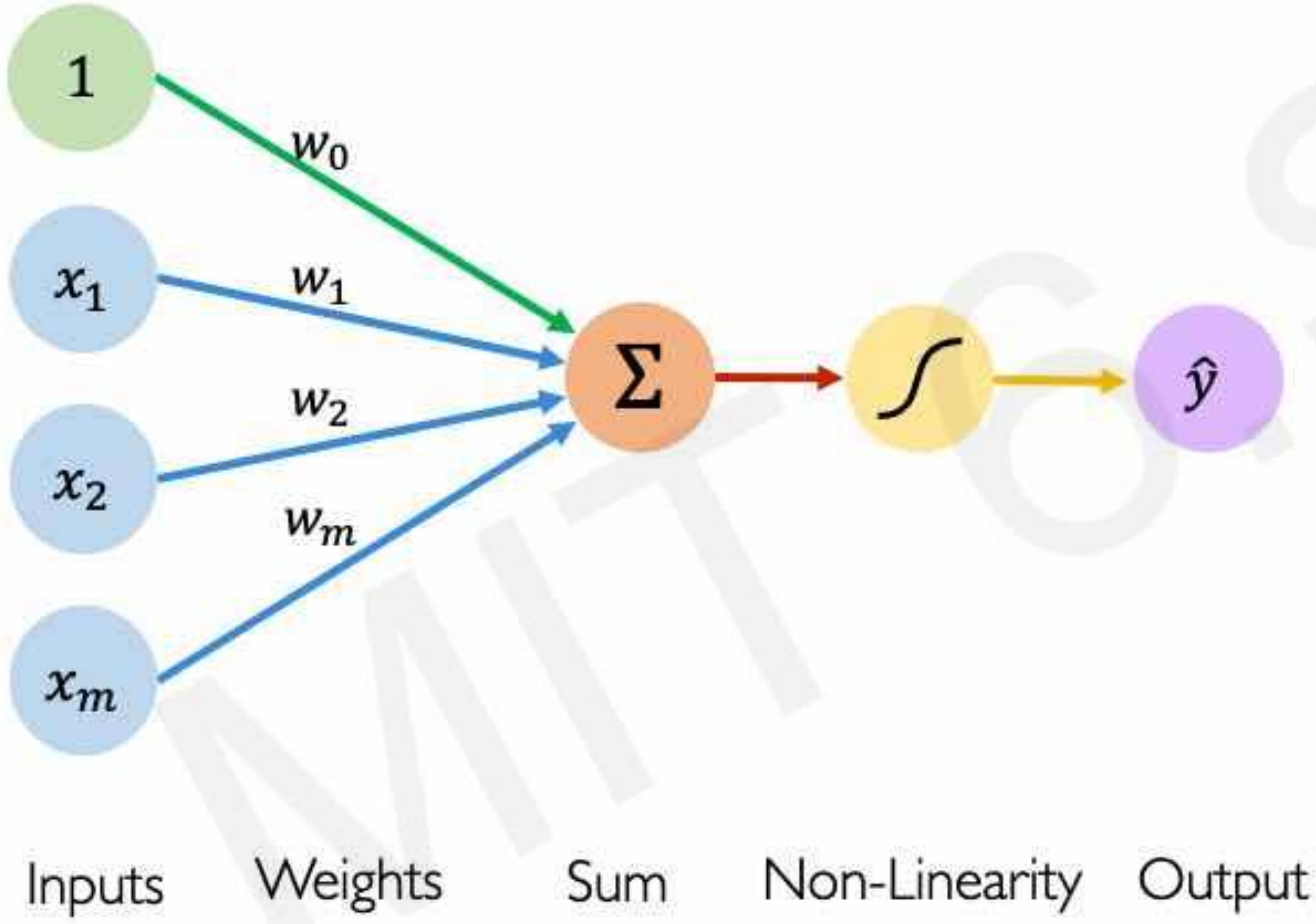
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

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

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

MIT 6.S191

The Perceptron: Forward Propagation

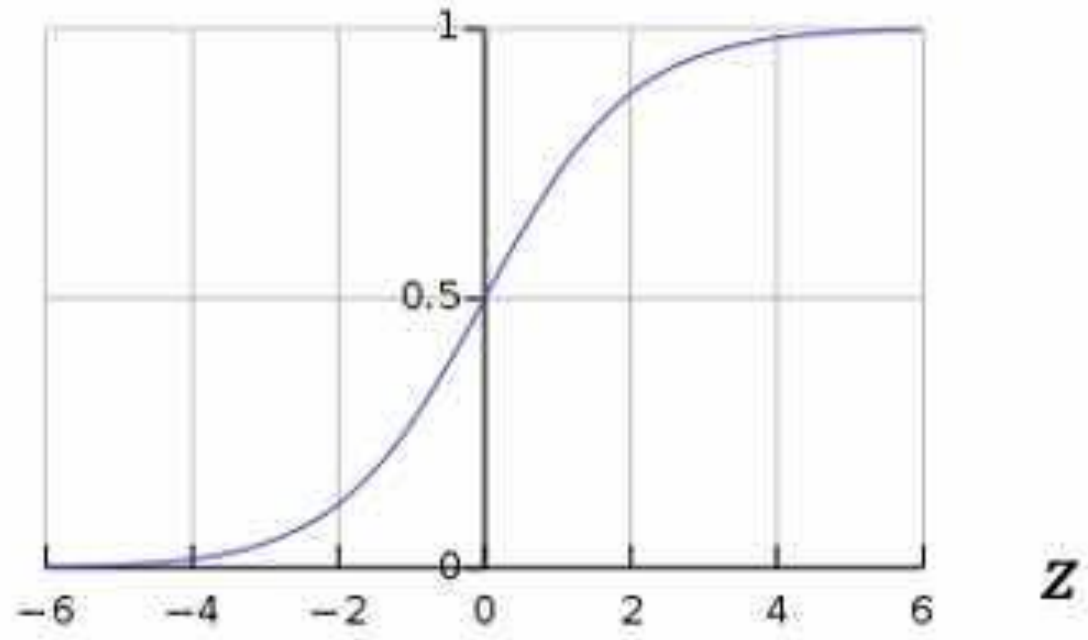


Activation Functions

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

- Example: sigmoid function

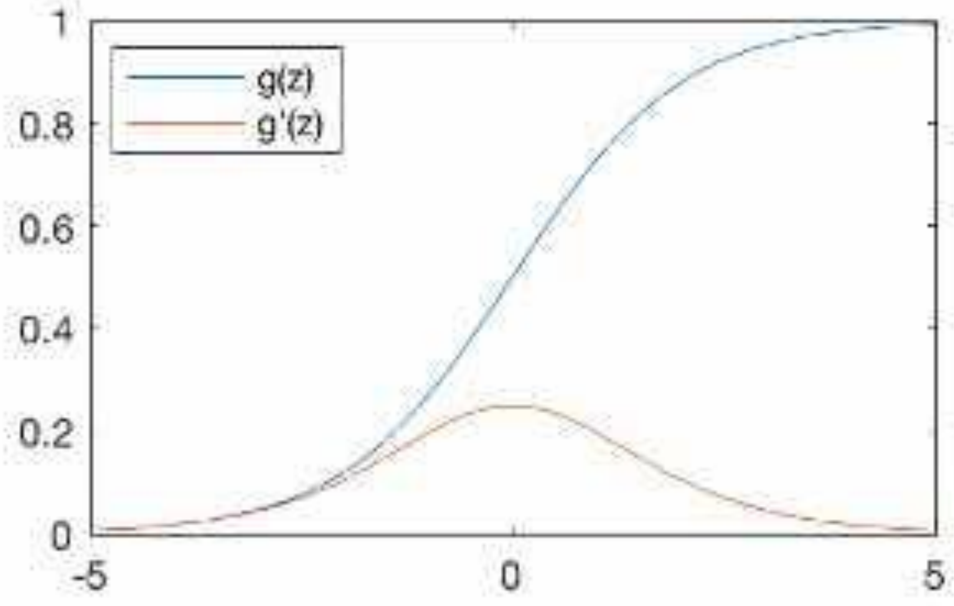
$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



MIT 6.S191

Common Activation Functions

Sigmoid Function

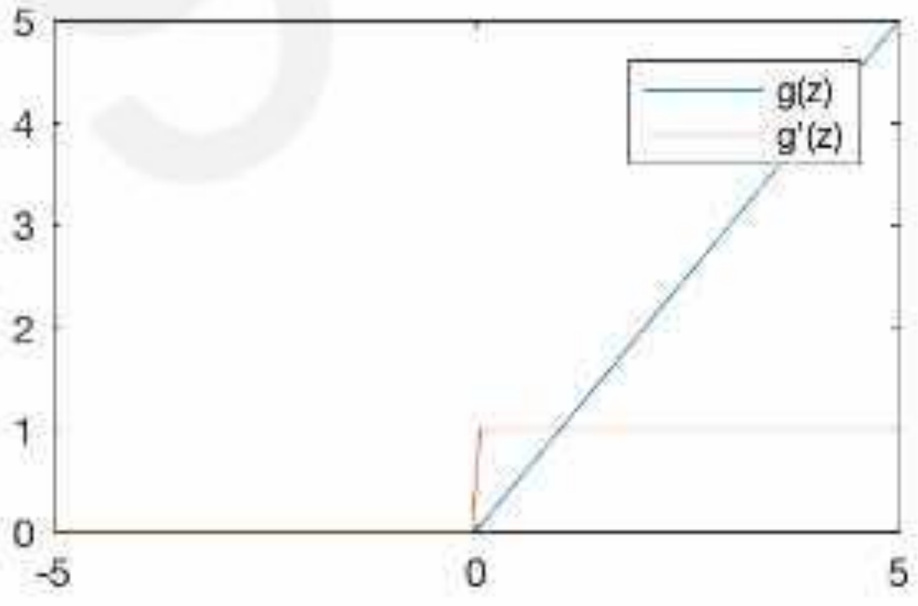


$$g(z) = \frac{1}{1 + e^{-z}}$$

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

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

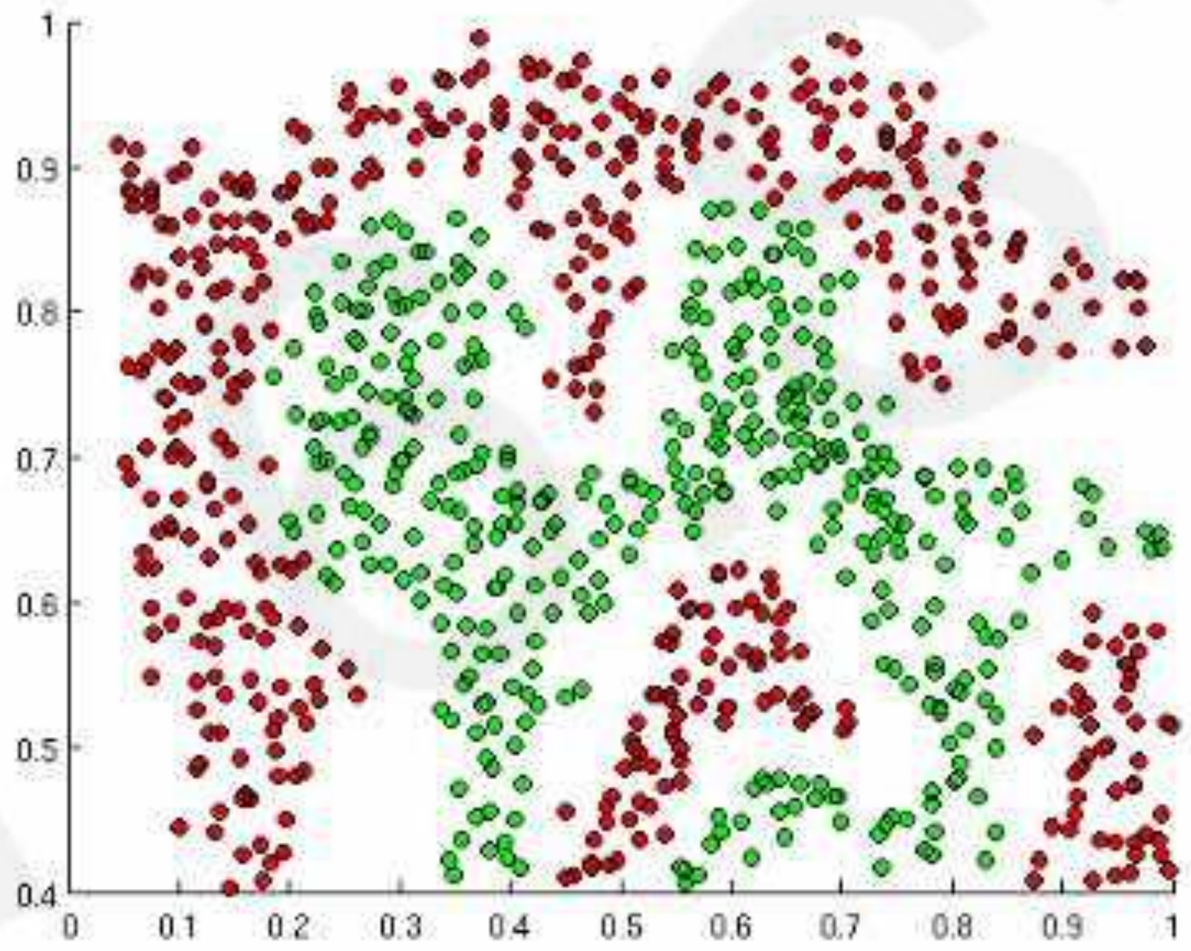
TensorFlow code blocks

NOTE: All activation functions are non-linear

MIT

Importance of Activation Functions

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

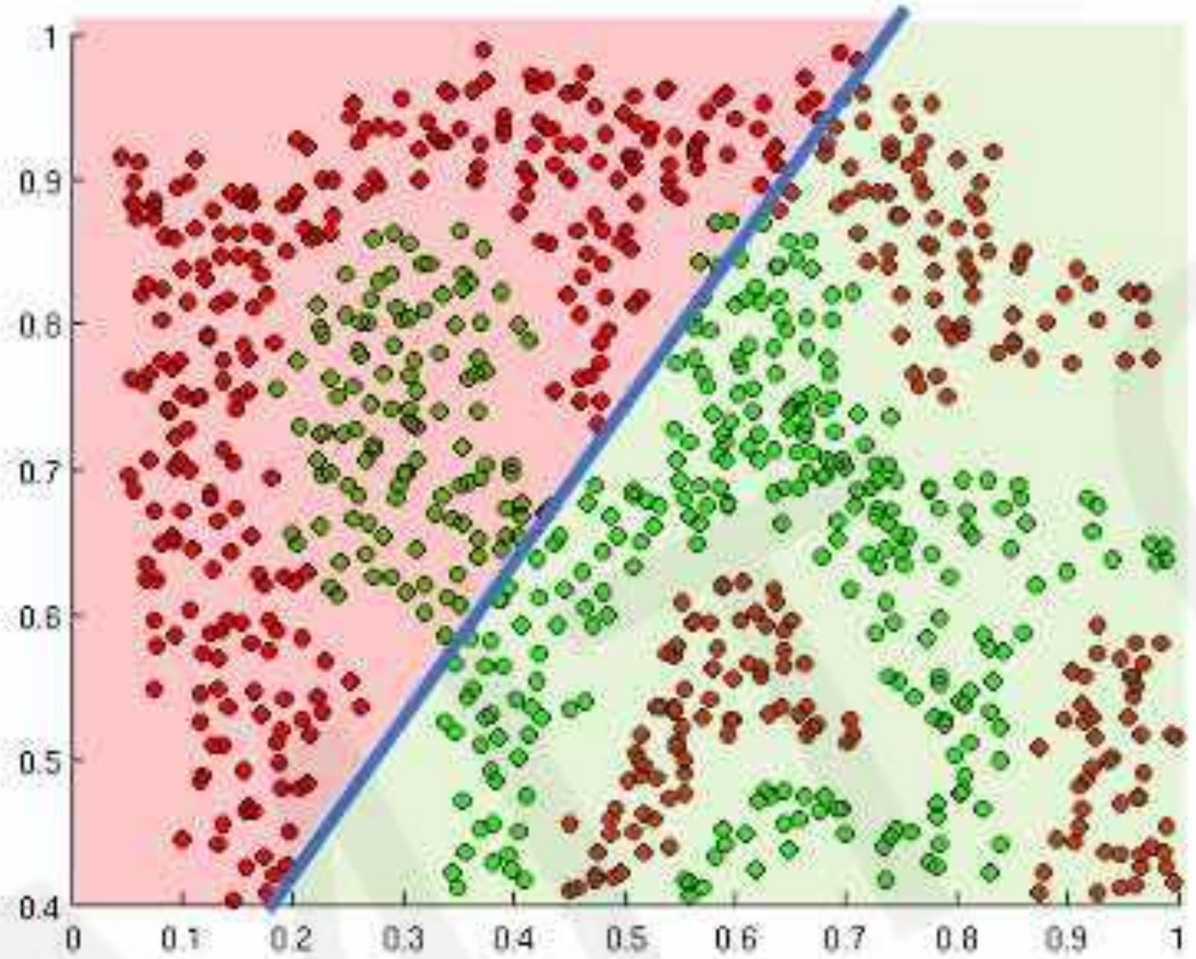


What if we wanted to build a neural network to
distinguish green vs red points?

MIT 6.S191

Importance of Activation Functions

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

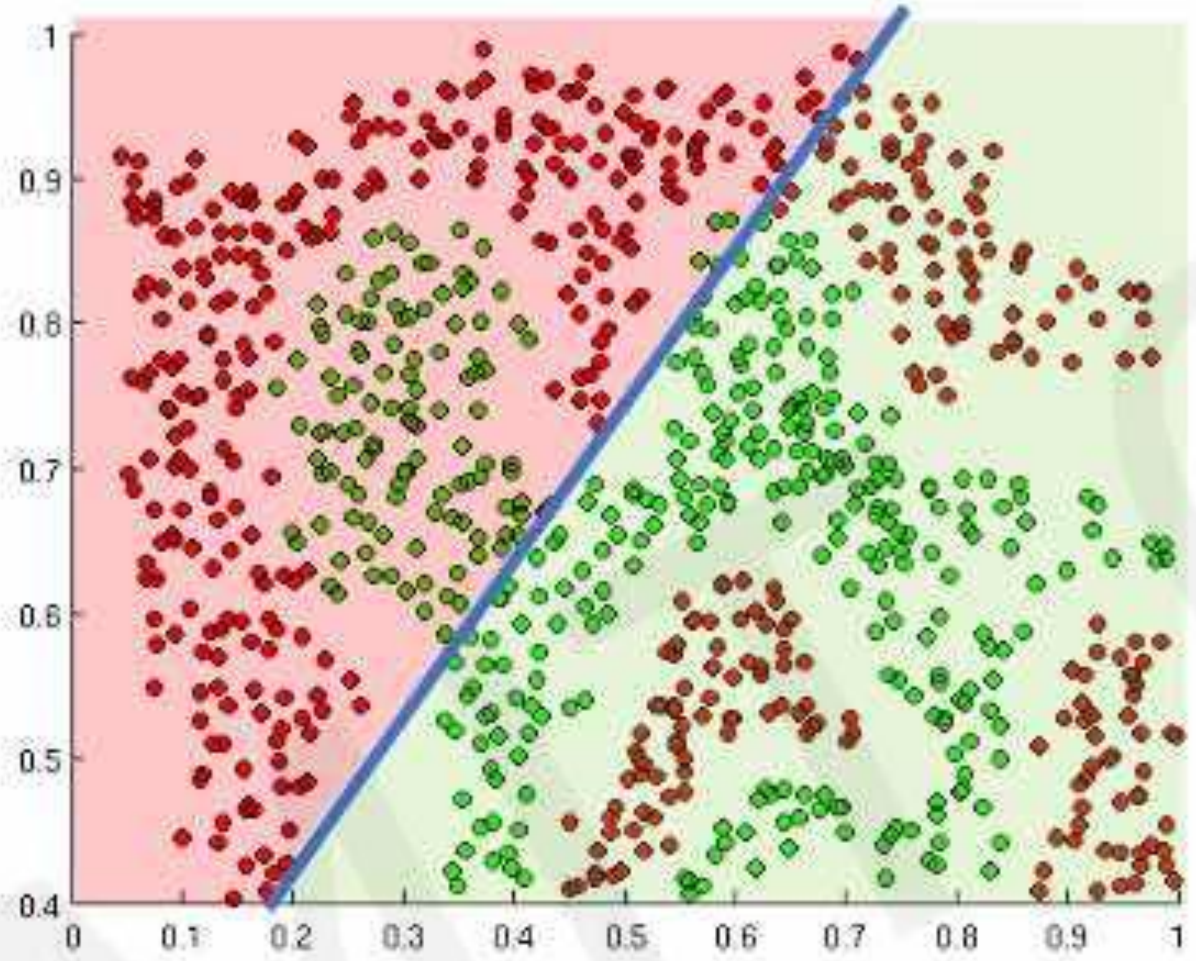


Linear activation functions produce linear decisions no matter the network size

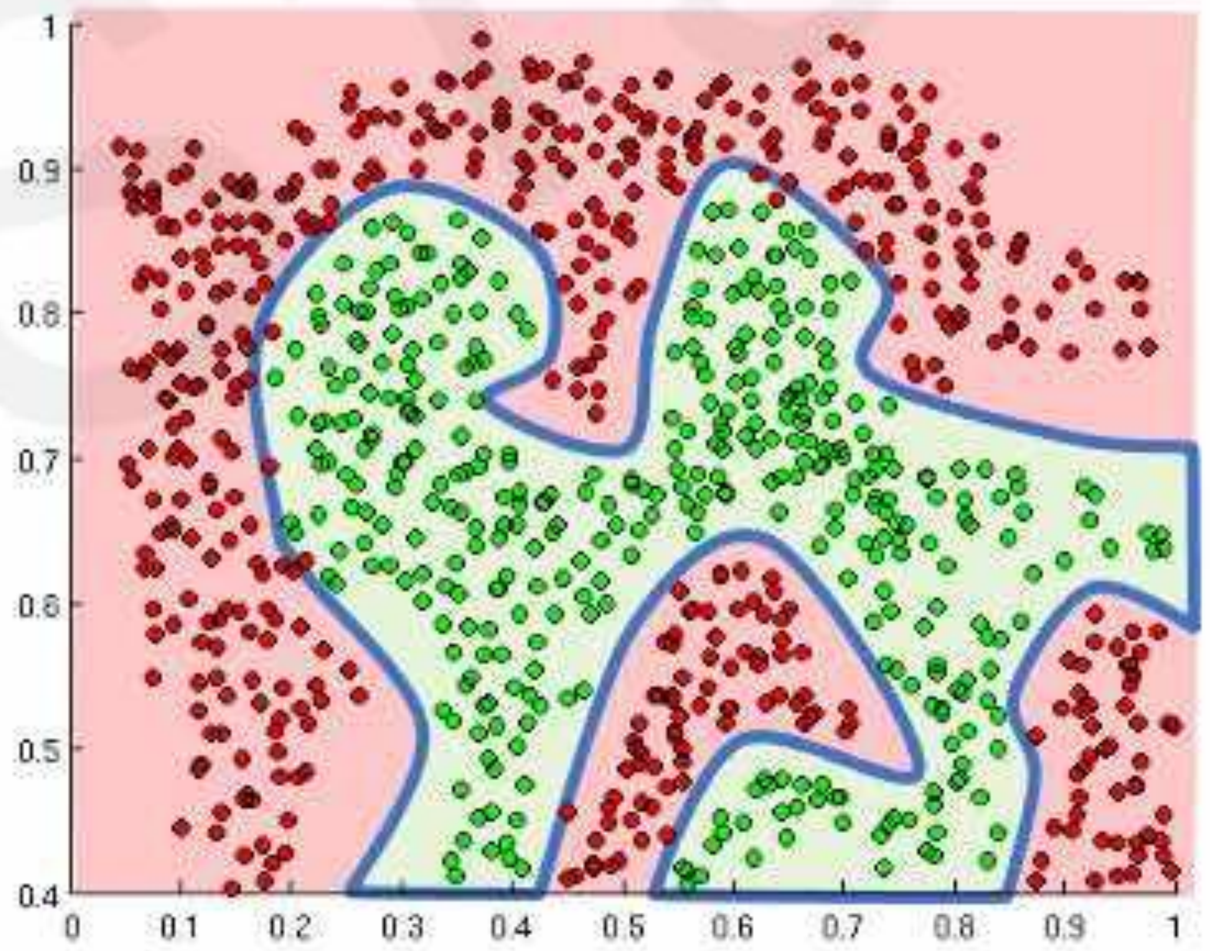
MIT 6.S191

Importance of Activation Functions

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



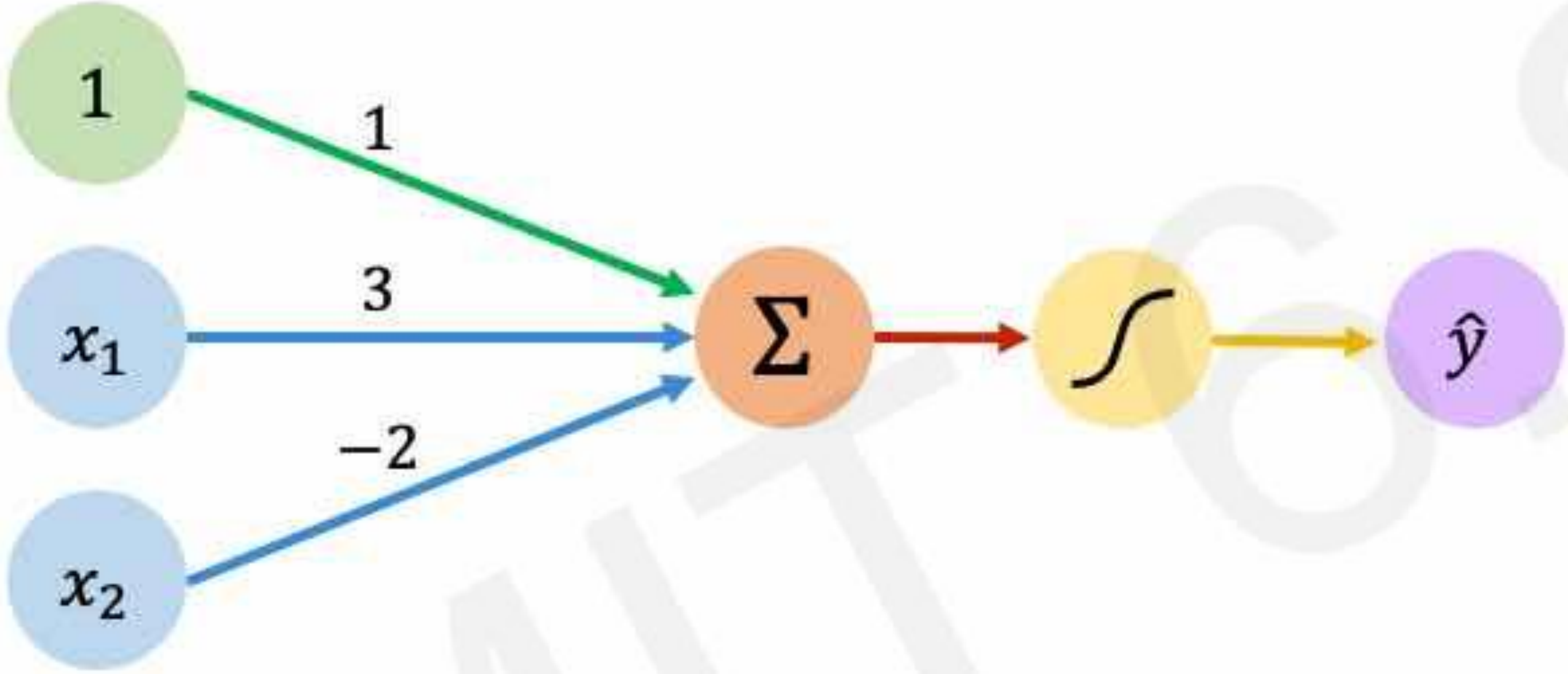
Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

MIT 6.S191

The Perceptron: Example

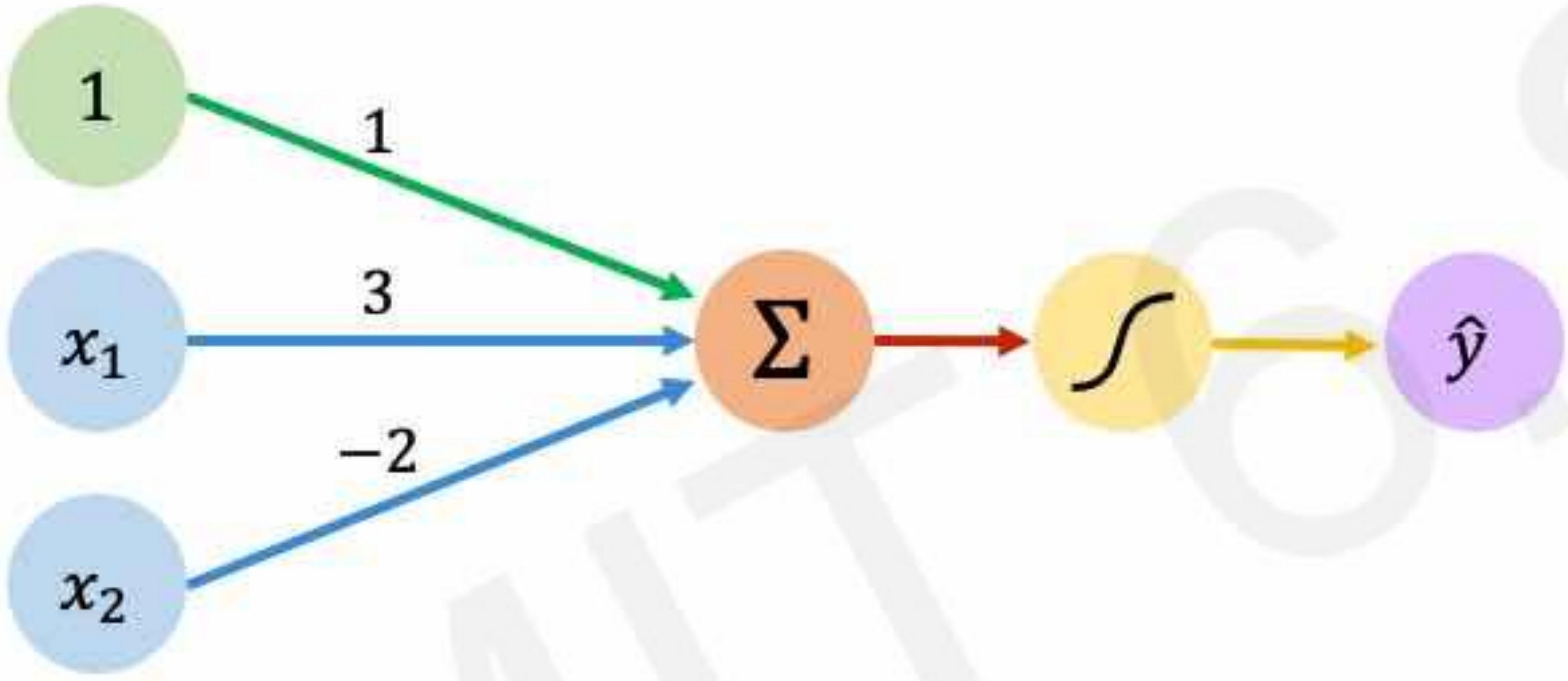


We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

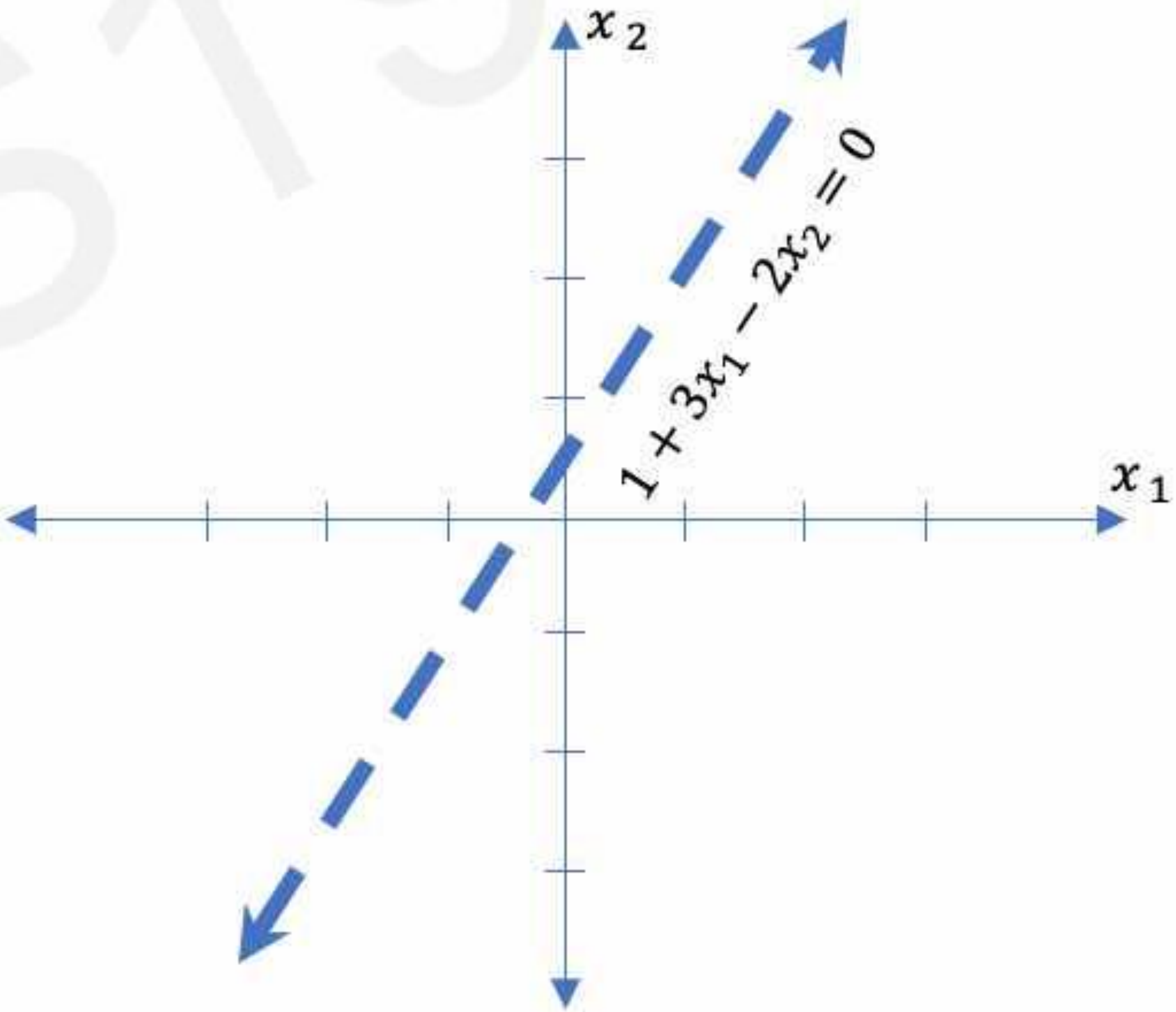
$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$
$$\hat{y} = g\underbrace{(1 + 3x_1 - 2x_2)}_{\text{This is just a line in 2D!}}$$

MIT 6.S191

The Perceptron: Example

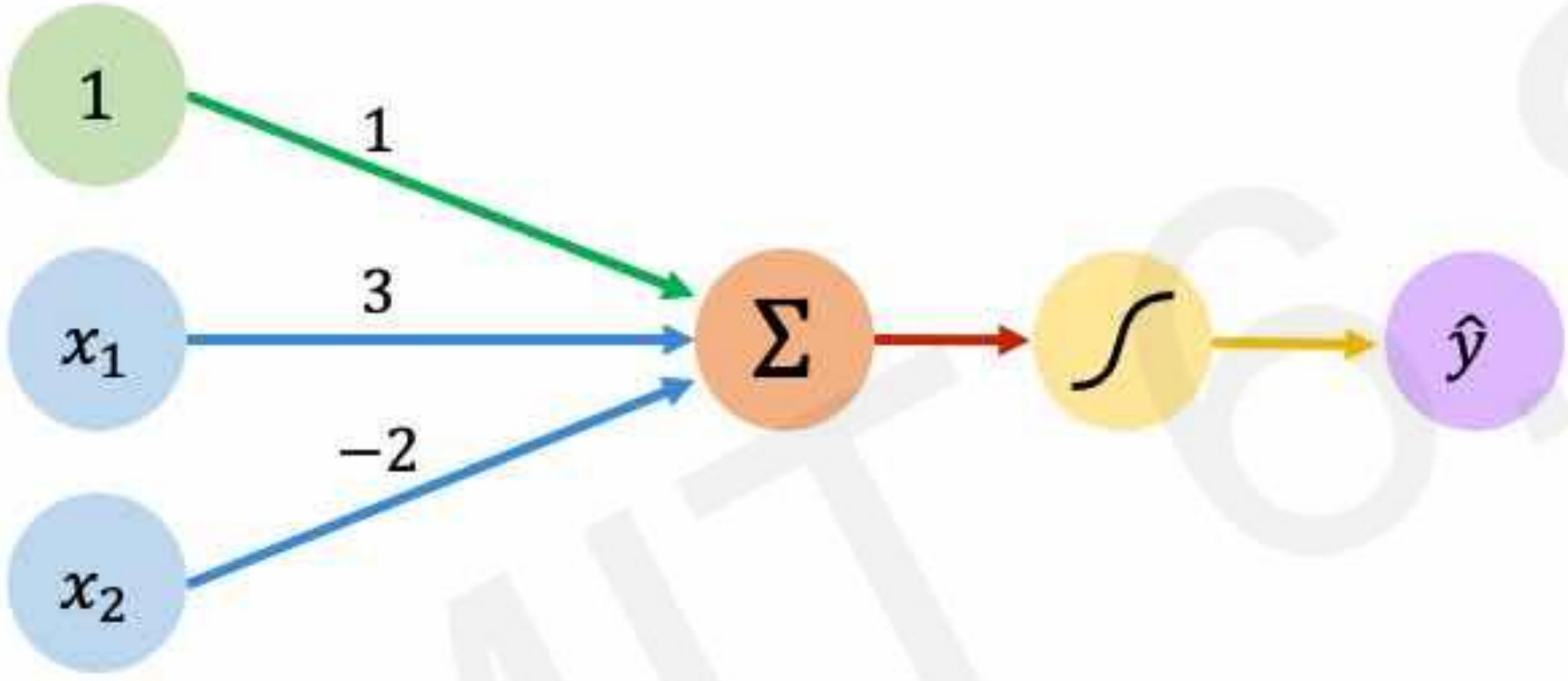


$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



MIT 6.S191

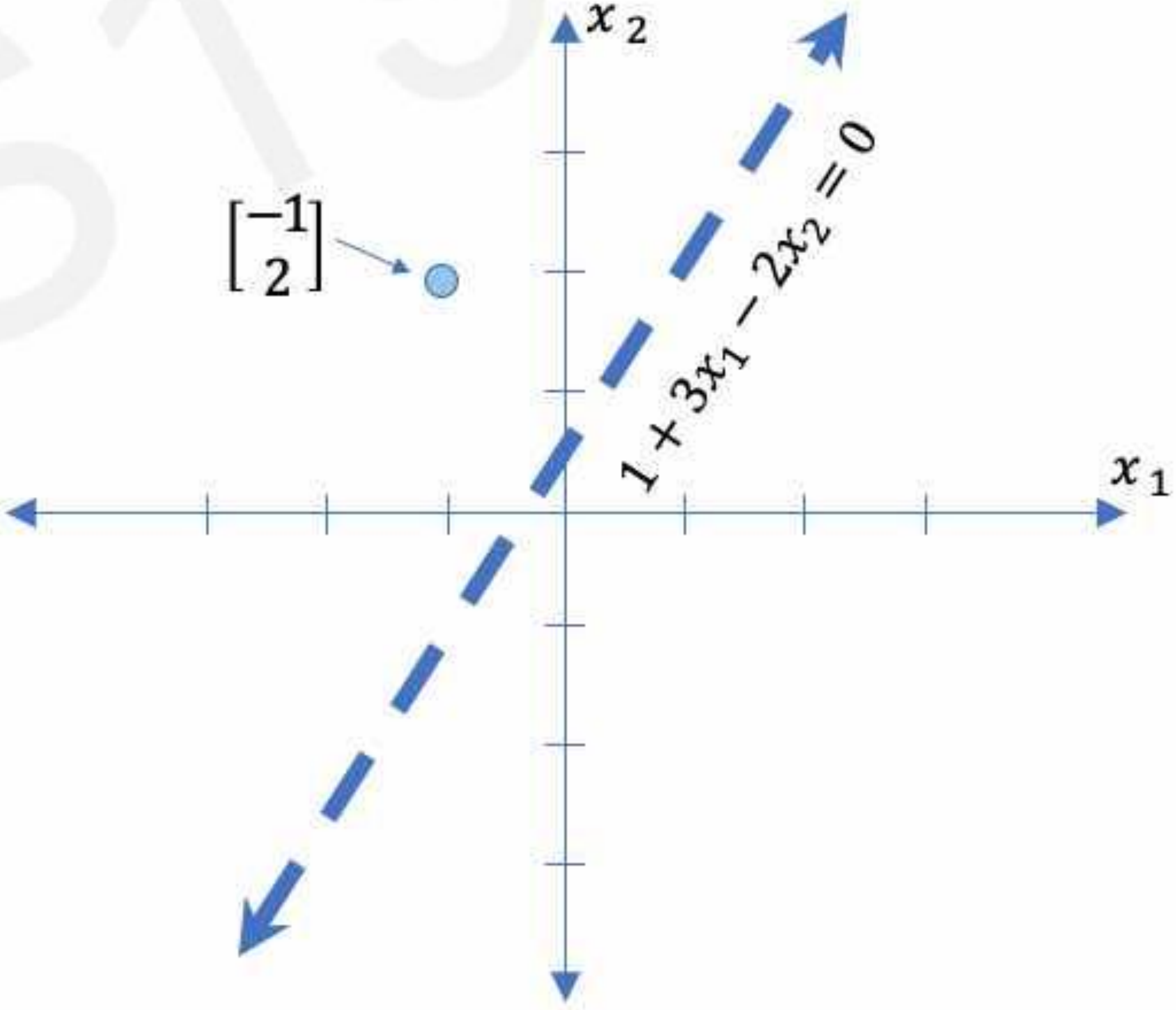
The Perceptron: Example



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

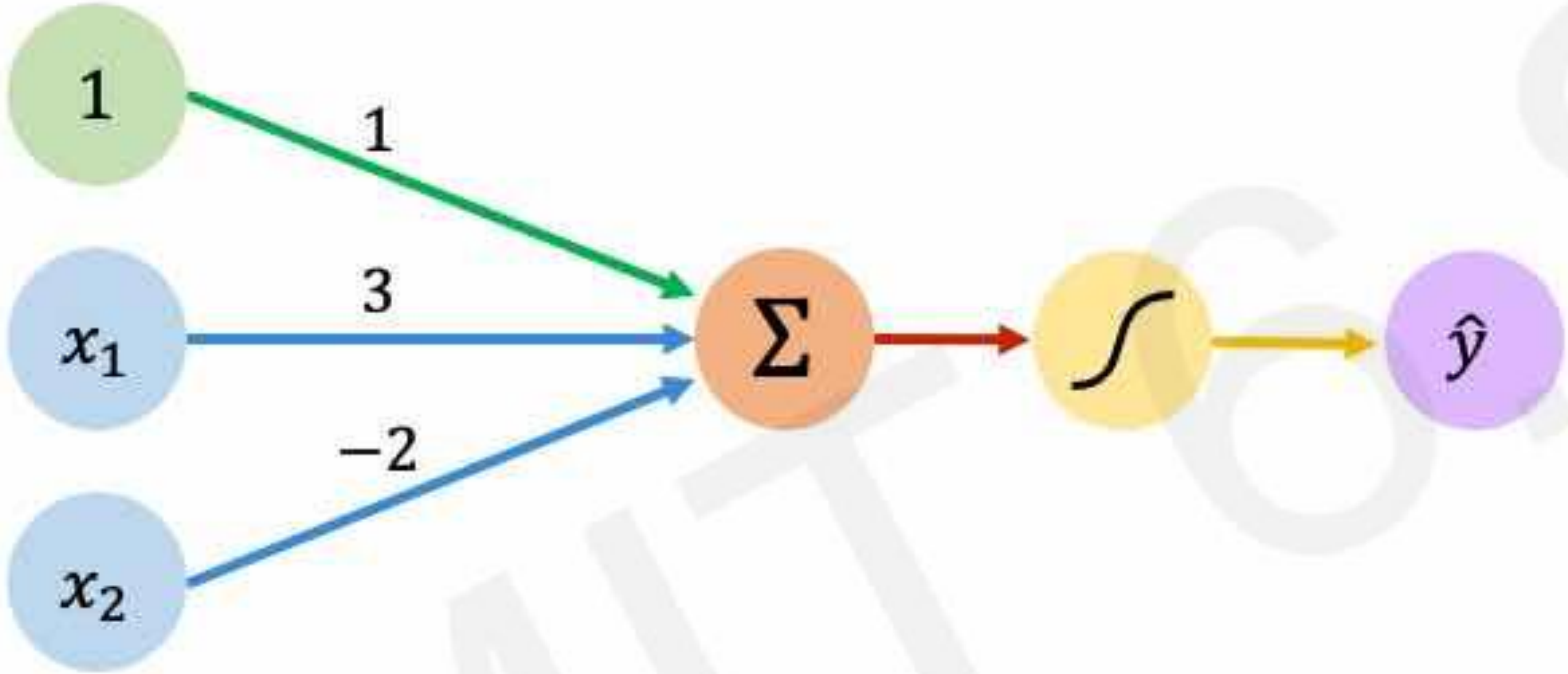
$$\hat{y} = g\big(1 + (3*-1) - (2*2)\big)$$
$$= g(-6) \approx 0.002$$

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

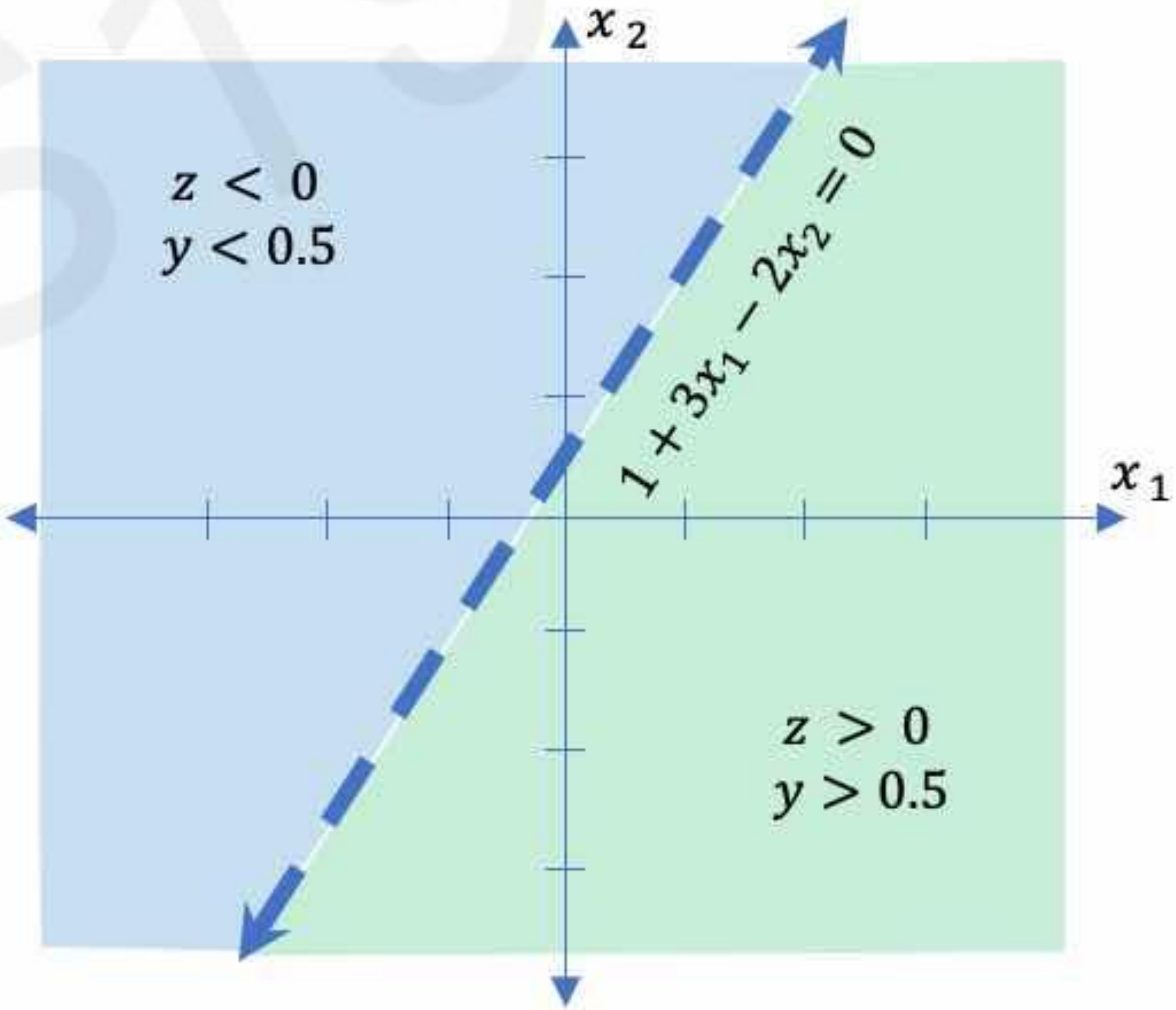


MIT 6.S191

The Perceptron: Example



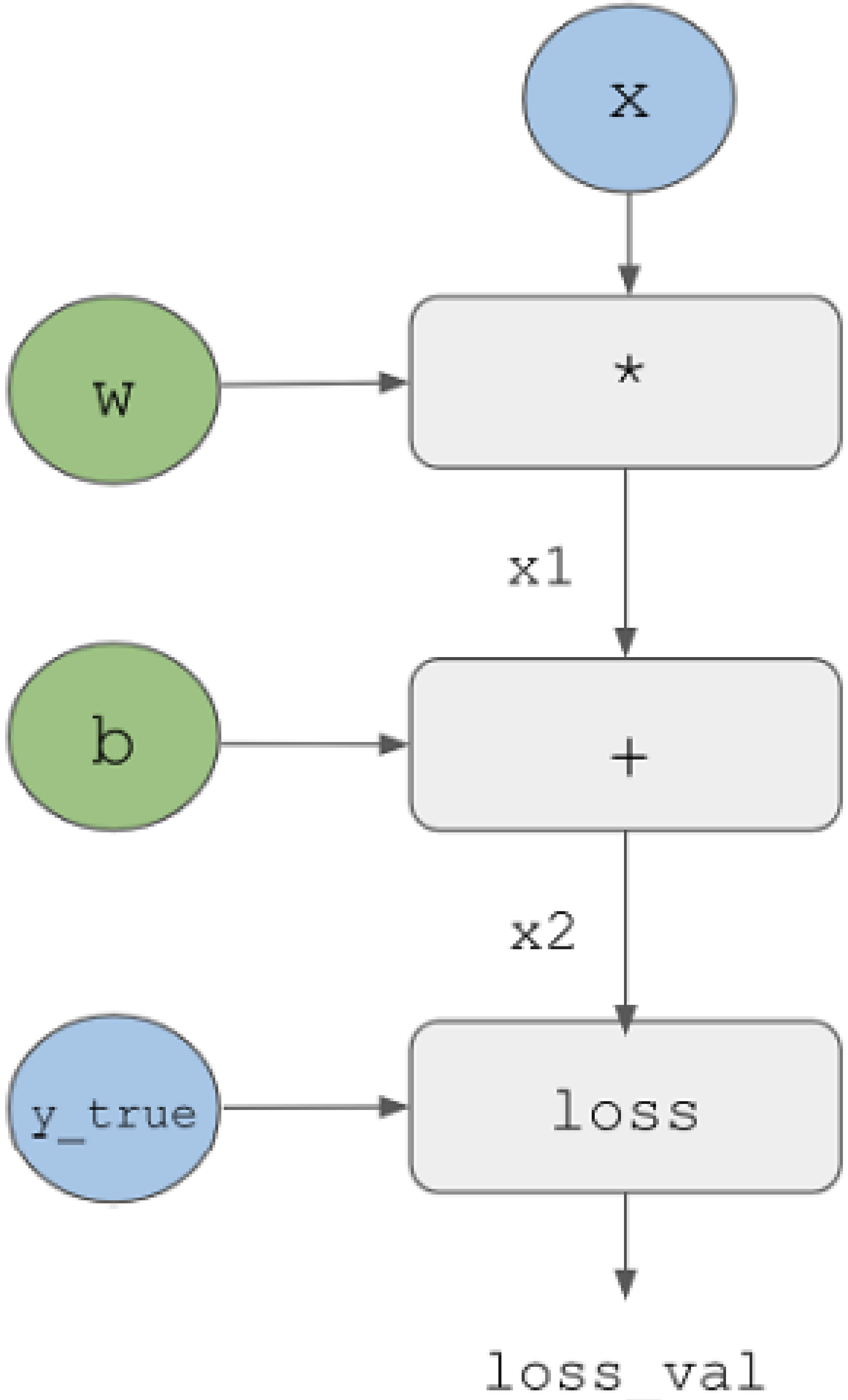
$\hat{y} = g(1 + 3x_1 - 2x_2)$



MIT 6.S191

Building Neural Networks with Perceptrons

MIT 6.S191



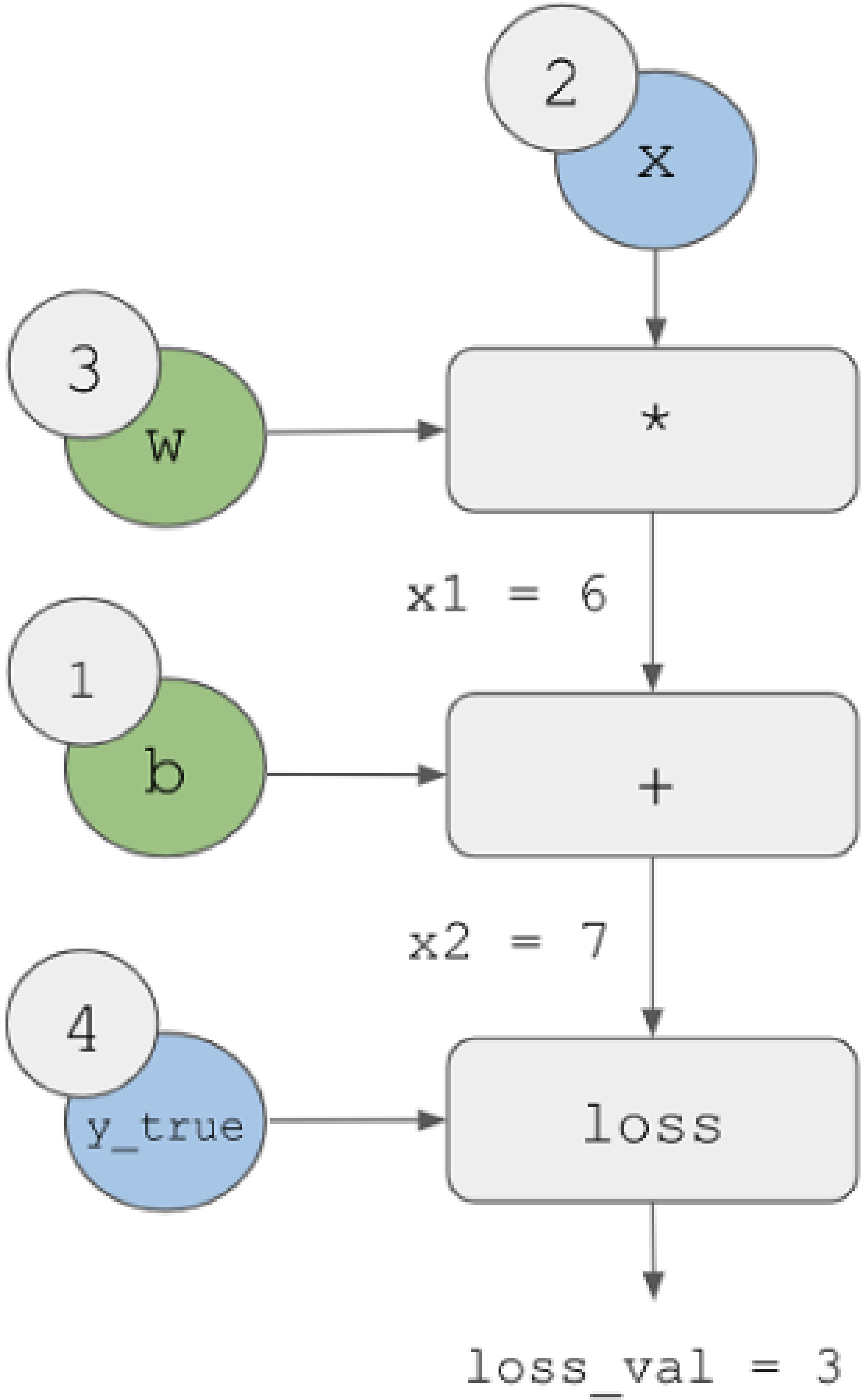


Figure 2.21 Running a forward pass

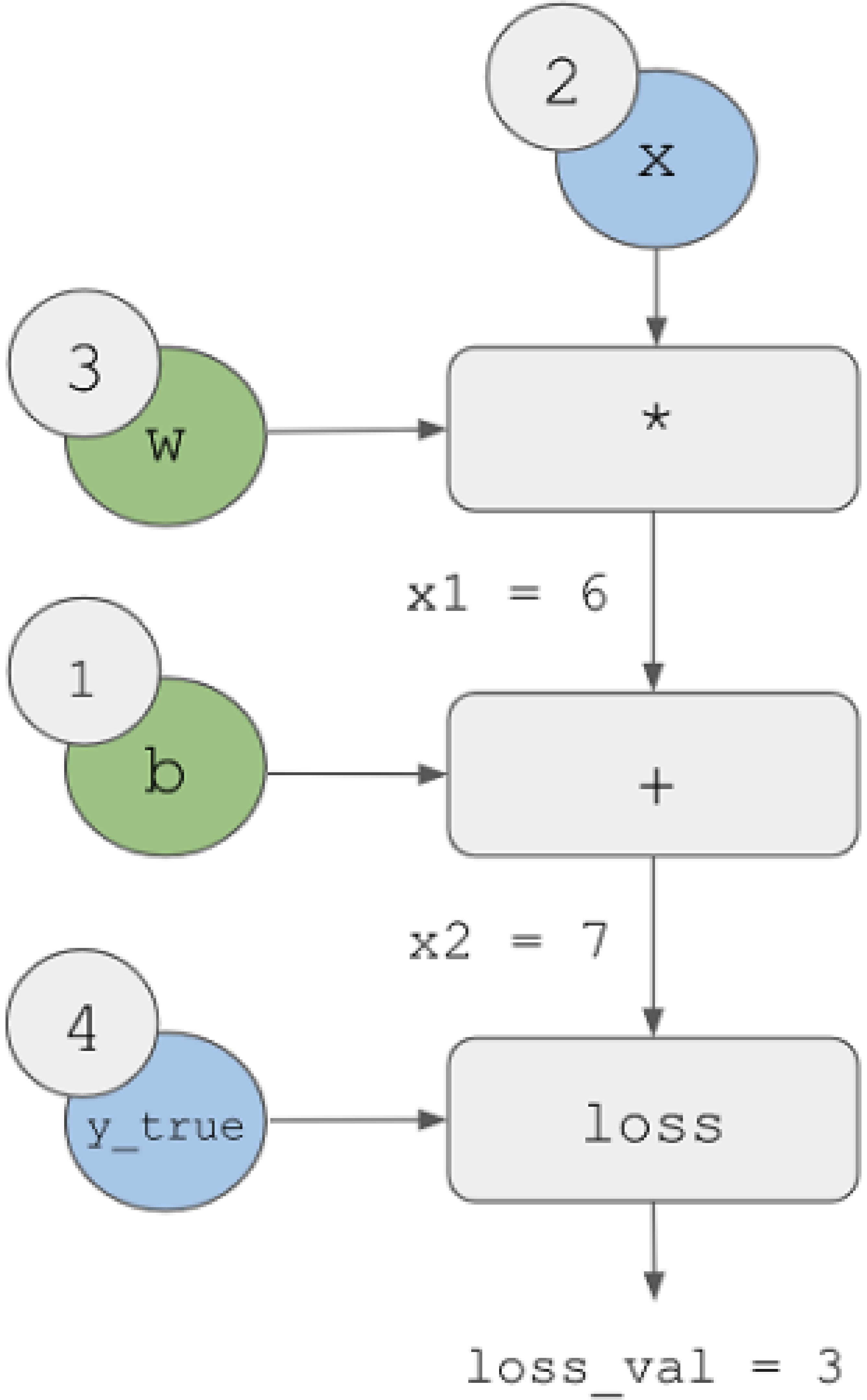


Figure 2.21 Running a forward pass

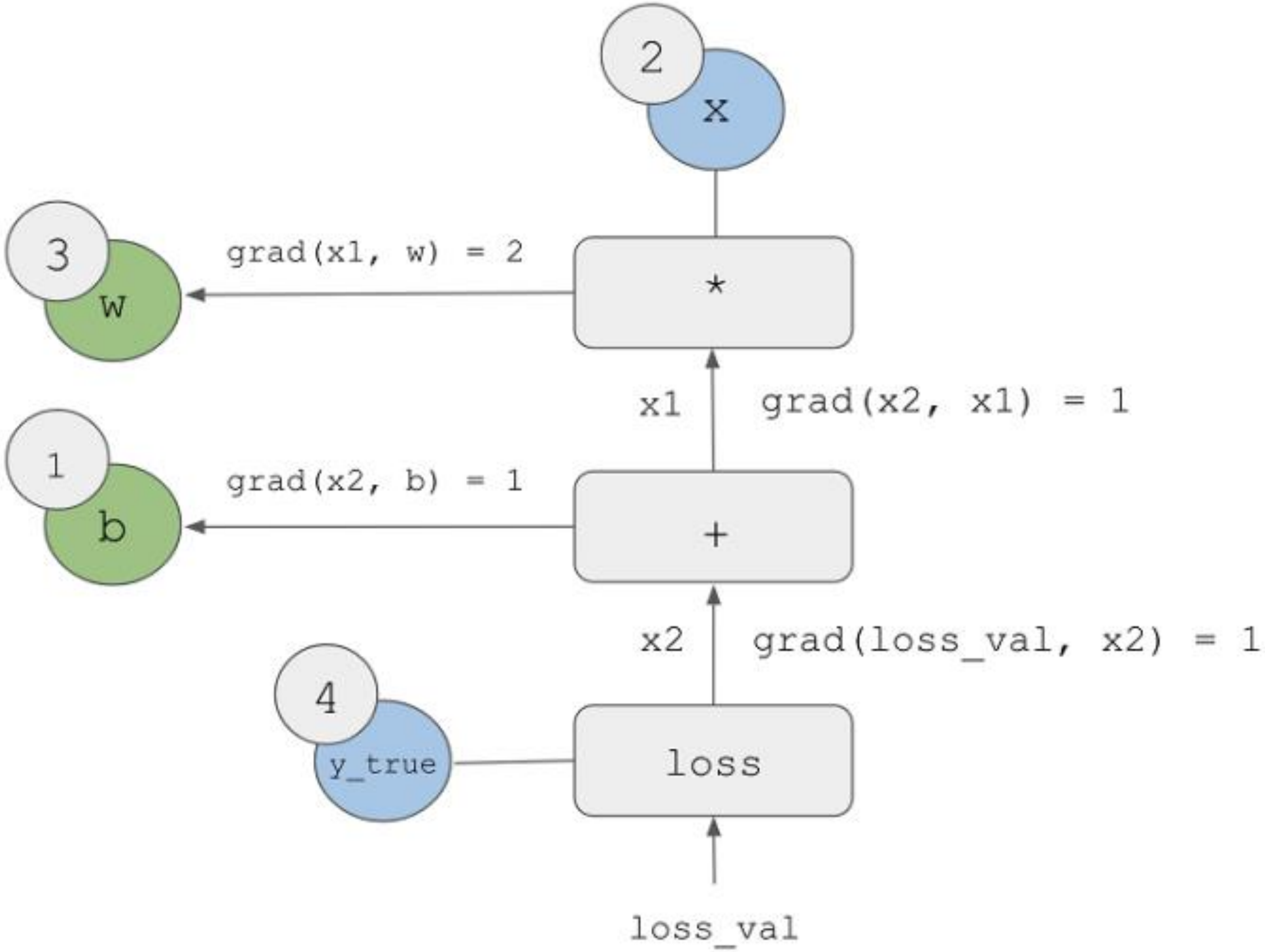
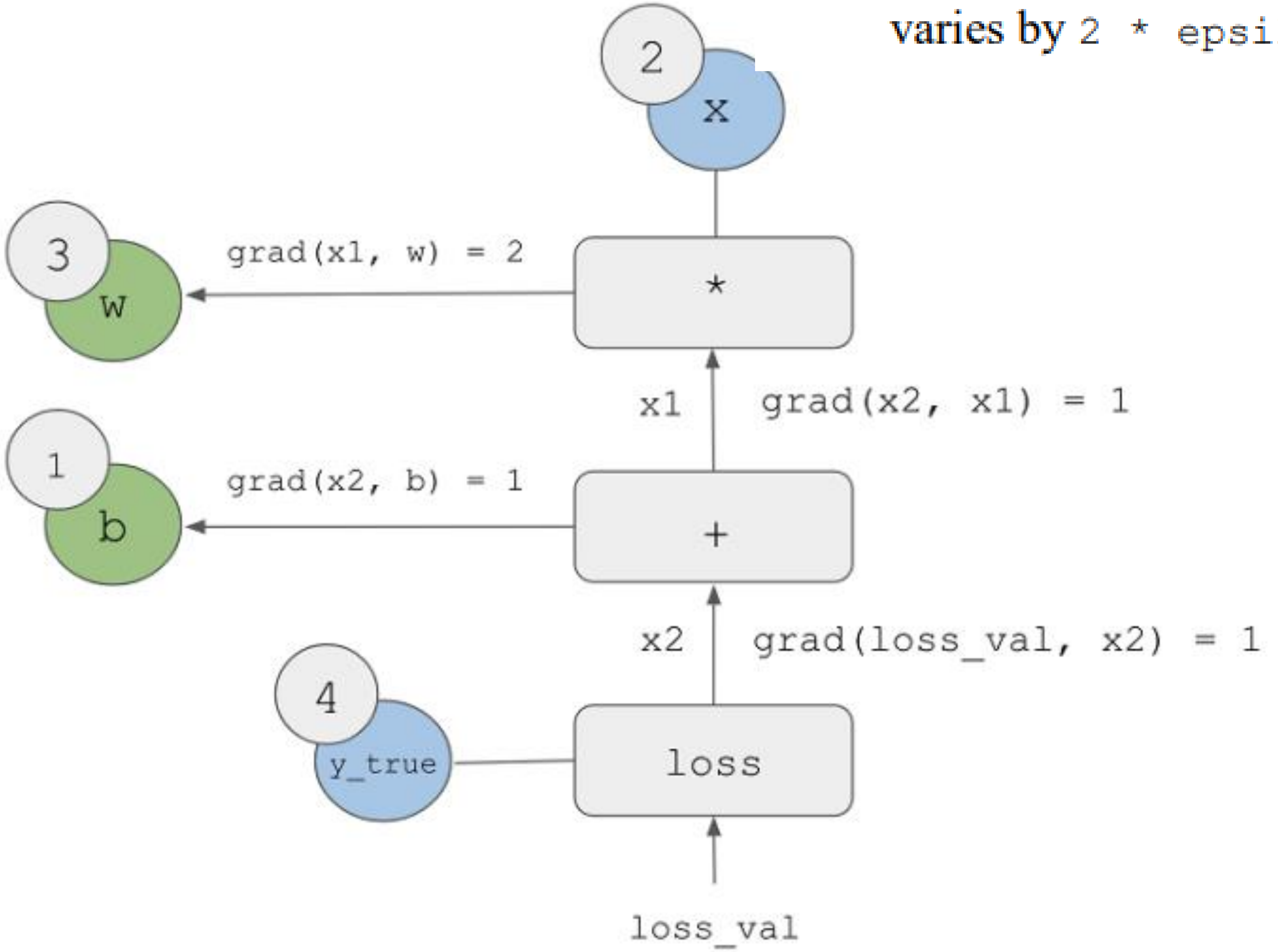


Figure 2.22 Running a backward pass

We have:

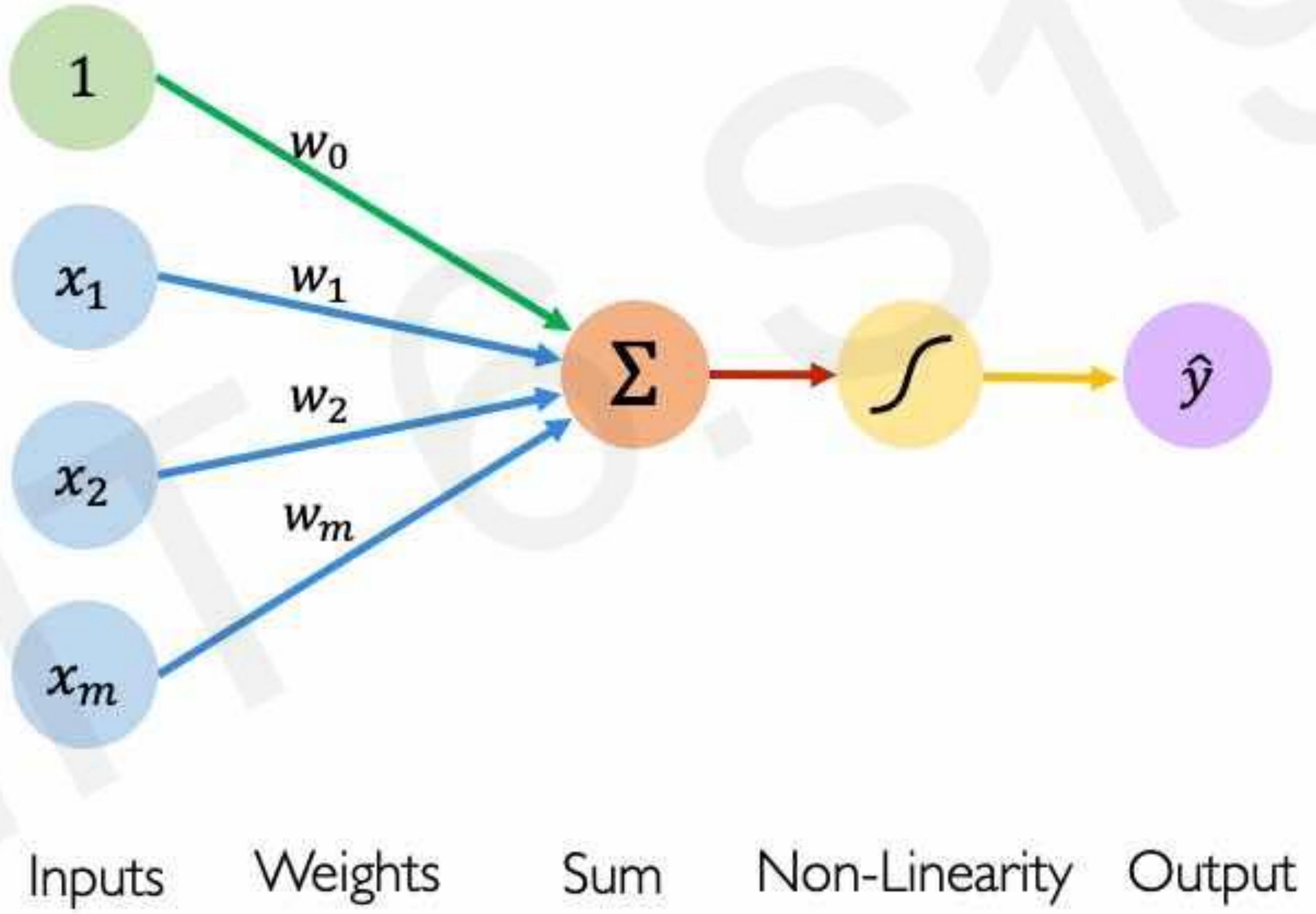
- `grad(loss_val, x2) = 1`, because as `x2` varies by an amount epsilon, `loss_val = abs(4 - x2)` varies by the same amount.
- `grad(x2, x1) = 1`, because as `x1` varies by an amount epsilon, `x2 = x1 + b = x1 + 1` varies by the same amount.
- `grad(x2, b) = 1`, because as `b` varies by an amount epsilon, `x2 = x1 + b = 6 + b` varies by the same amount.
- `grad(x1, w) = 2`, because as `w` varies by an amount epsilon, `x1 = x * w = 2 * w` varies by `2 * epsilon`.



Backward-Pass a.k.a. Backpropagation

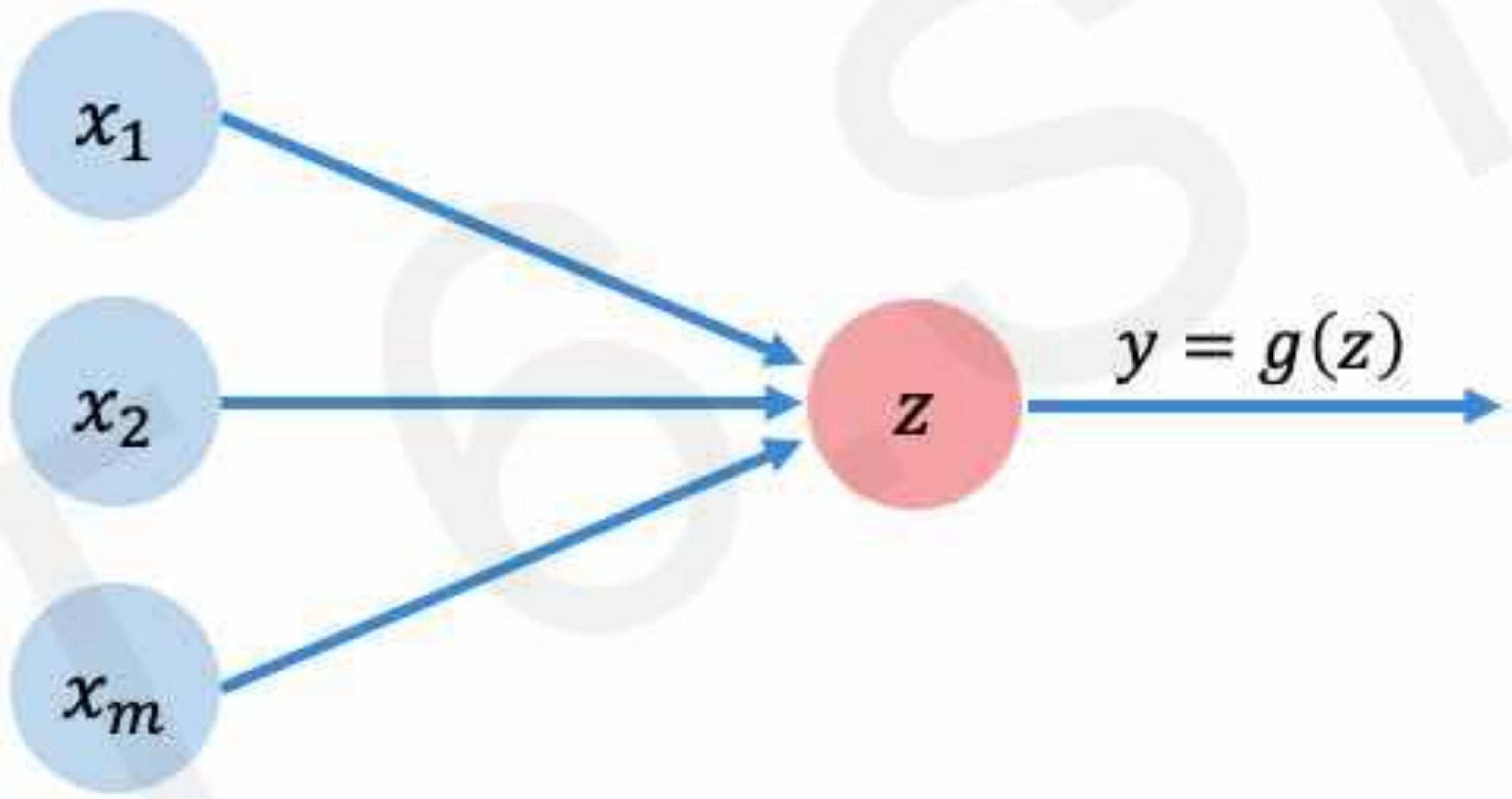
The Perceptron: Simplified

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



MIT 6.S191

The Perceptron: Simplified

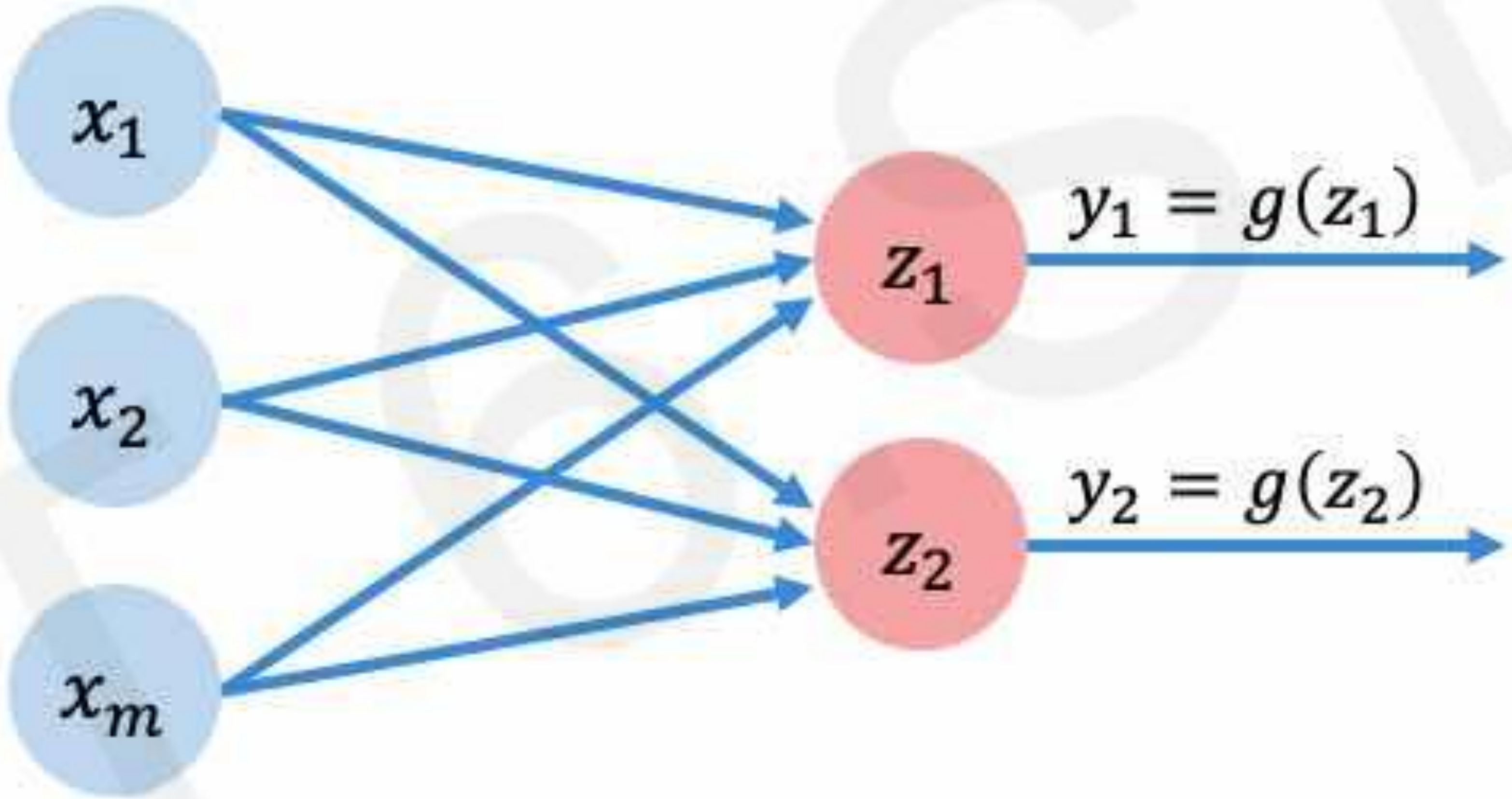


$$z = w_0 + \sum_{j=1}^{m} x_j \, w_j$$

MIT 6.S191

Multi Output Perceptron

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

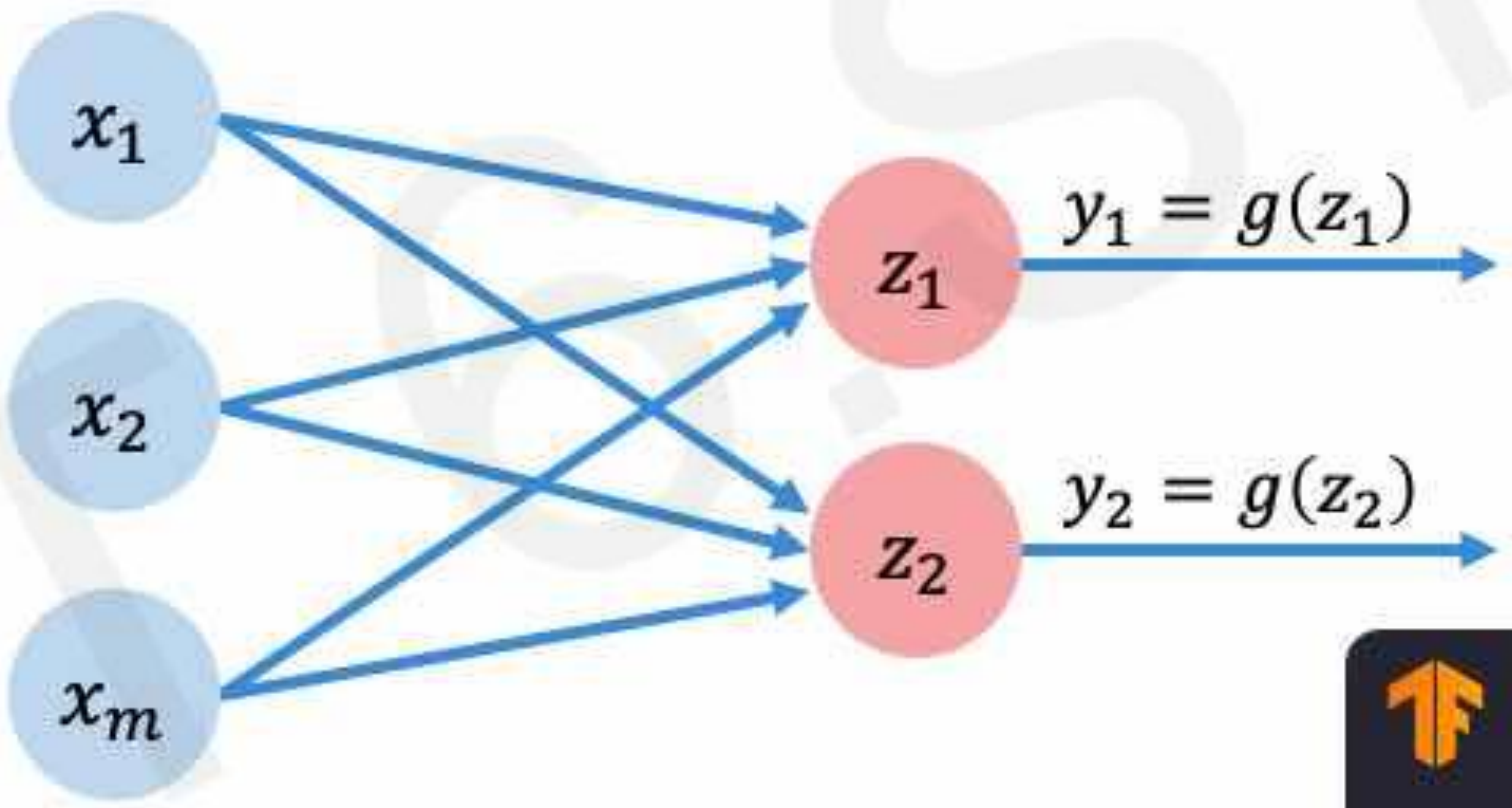


$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

MIT 6.S191

Multi Output Perceptron

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



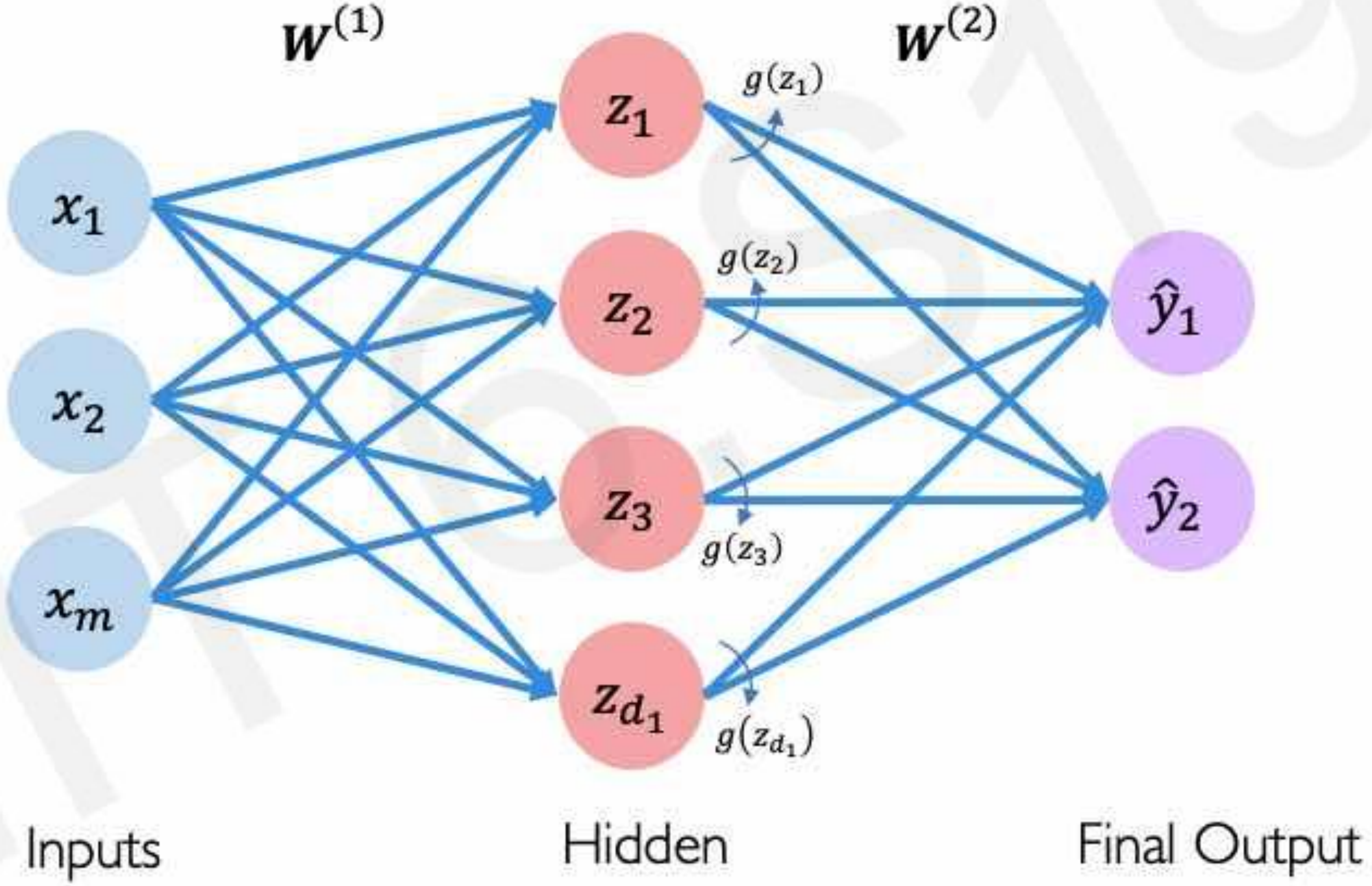

```python
import tensorflow as tf

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

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

MIT 6.S191

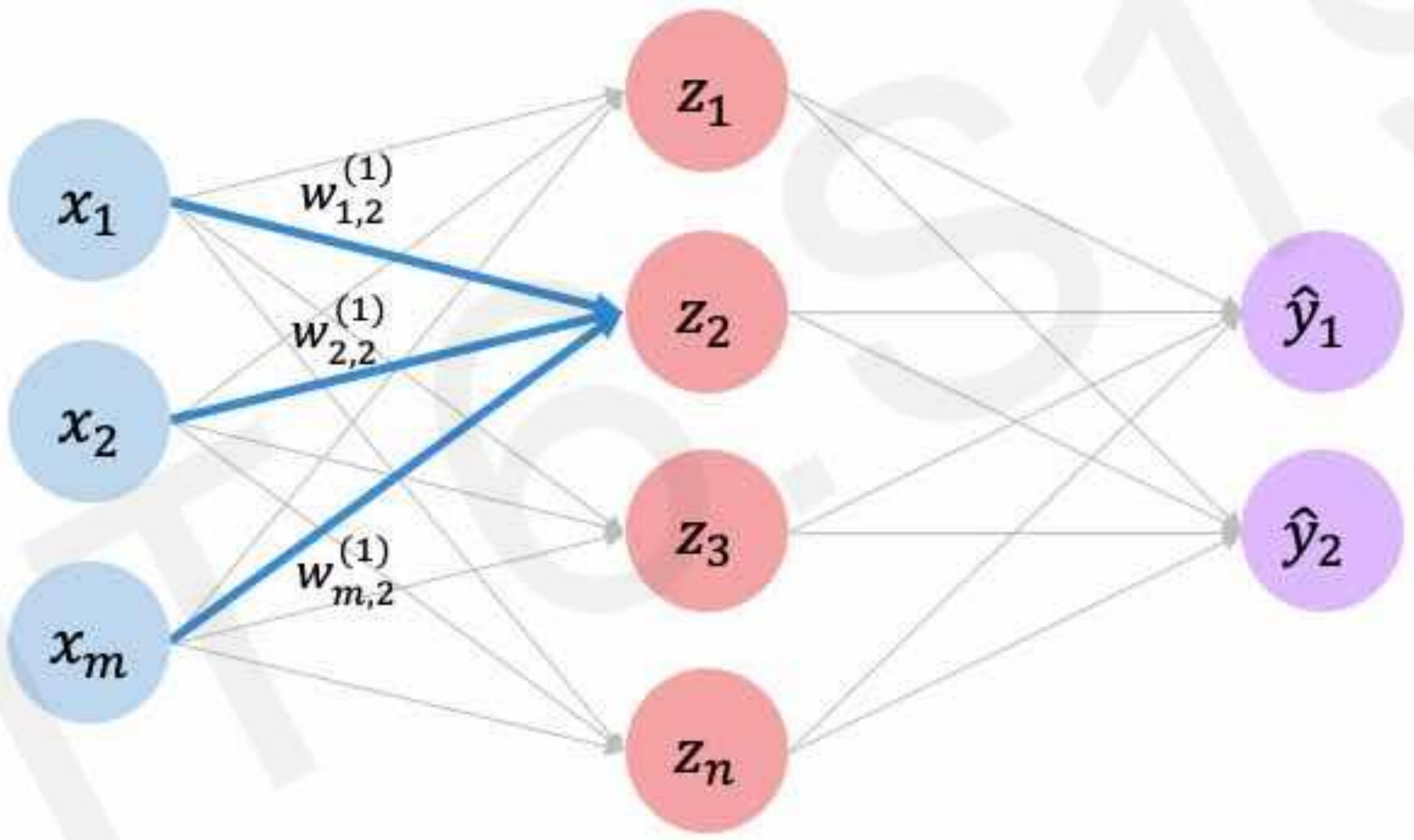
Single Layer Neural Network



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

MIT 6.S191

Single Layer Neural Network



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

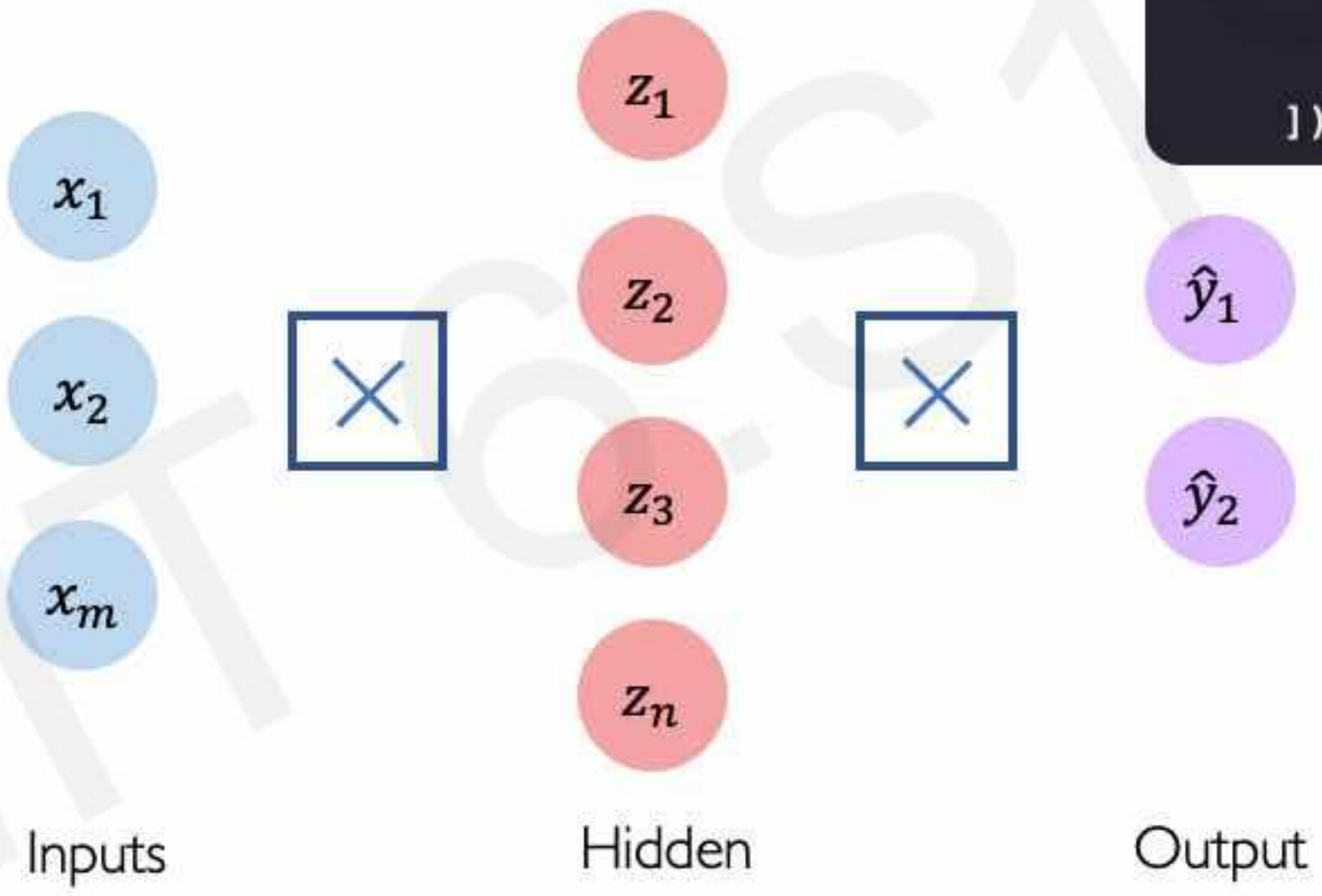
$$= w_{0,2}^{(1)} + x_1\, w_{1,2}^{(1)} + x_2\, w_{2,2}^{(1)} + x_m\, w_{m,2}^{(1)}$$

MIT 6.S191

Multi Output Perceptron

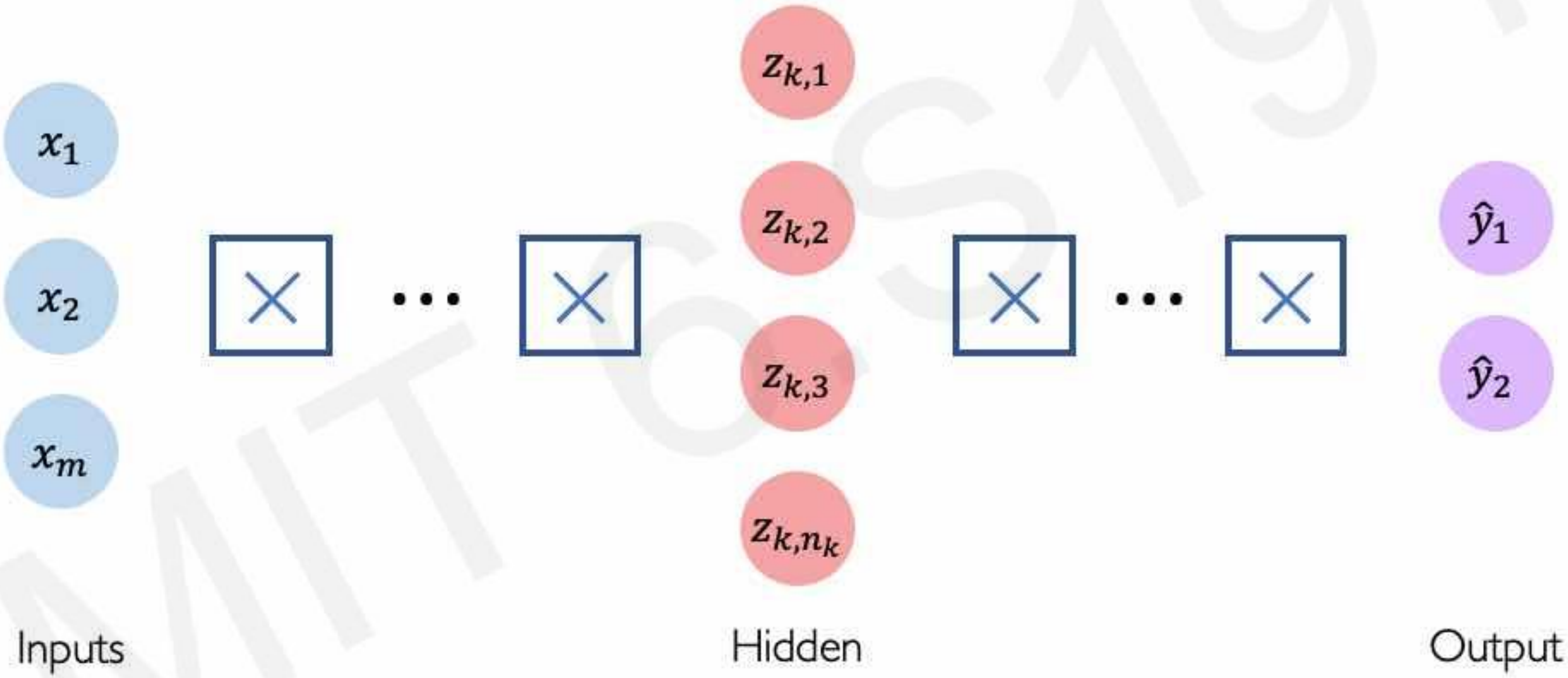
```
import tensorflow as tf

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



MIT 6.S191

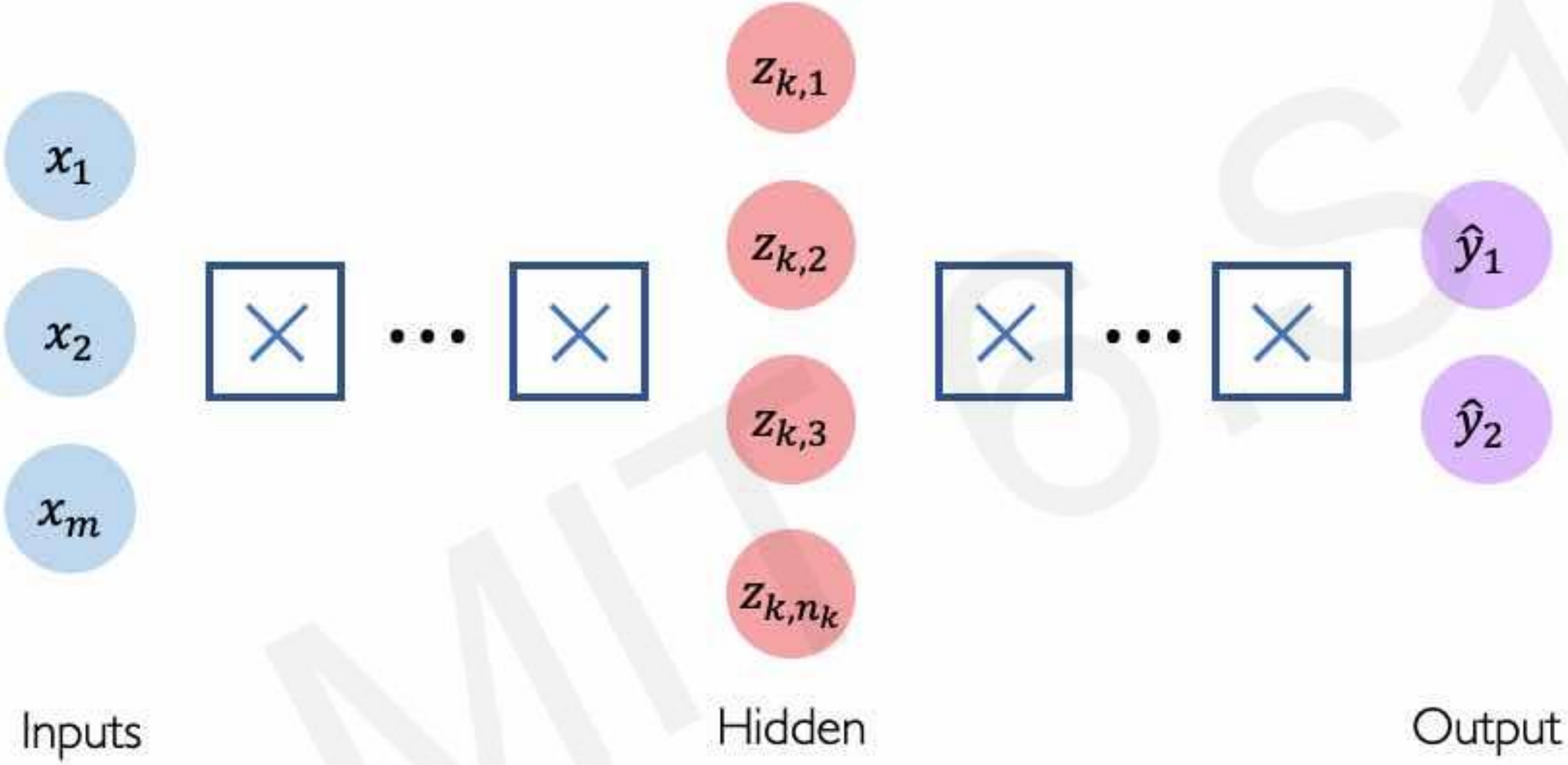
Deep Neural Network



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

MIT 6.S191

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)
])
```

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

MIT 6.S191

Applying Neural Networks

MIT 6.S191

Example Problem

Will I pass this class?

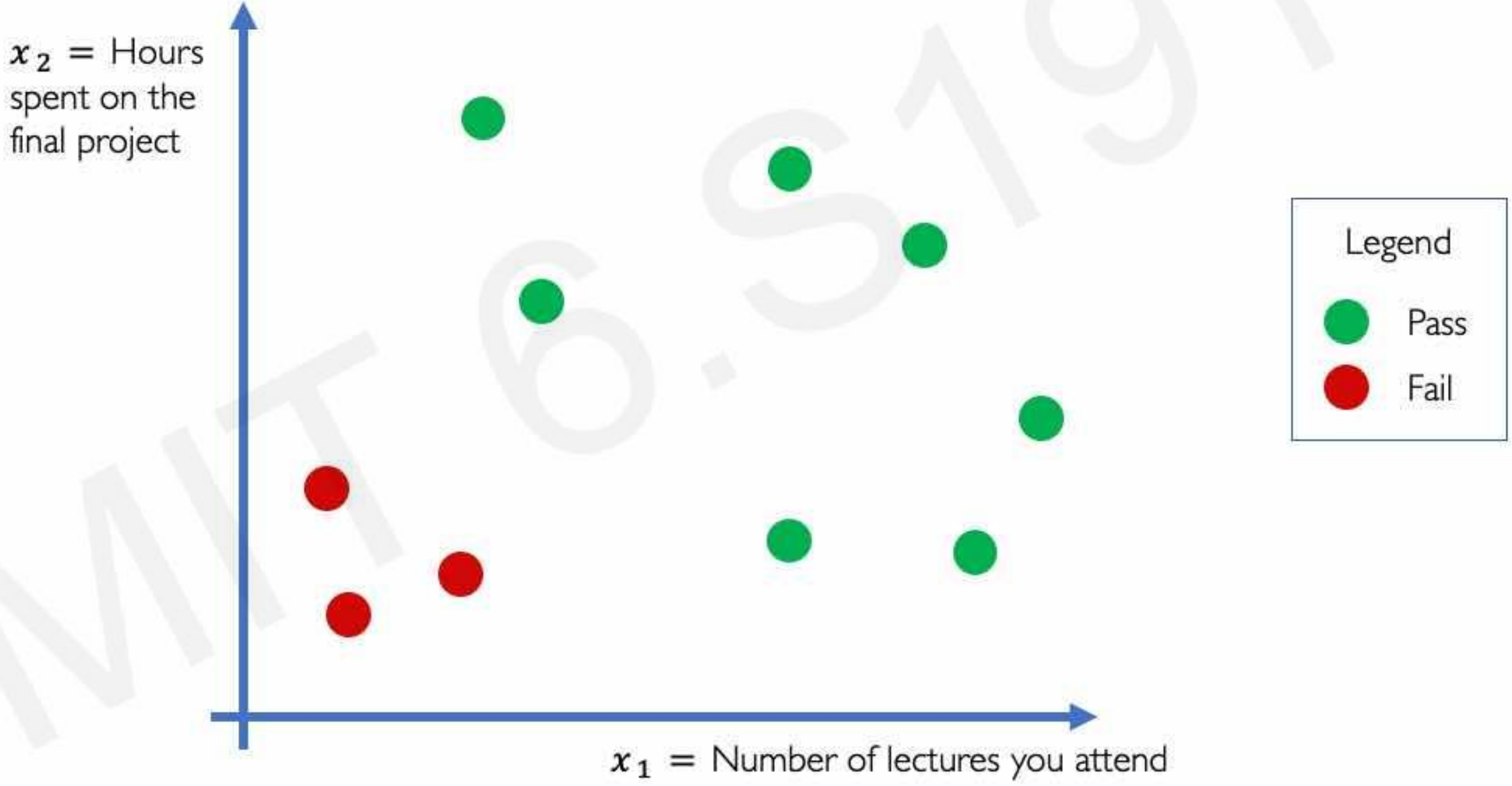
Let's start with a simple two feature model

x_1 = Number of lectures you attend

x_2 = Hours spent on the final project

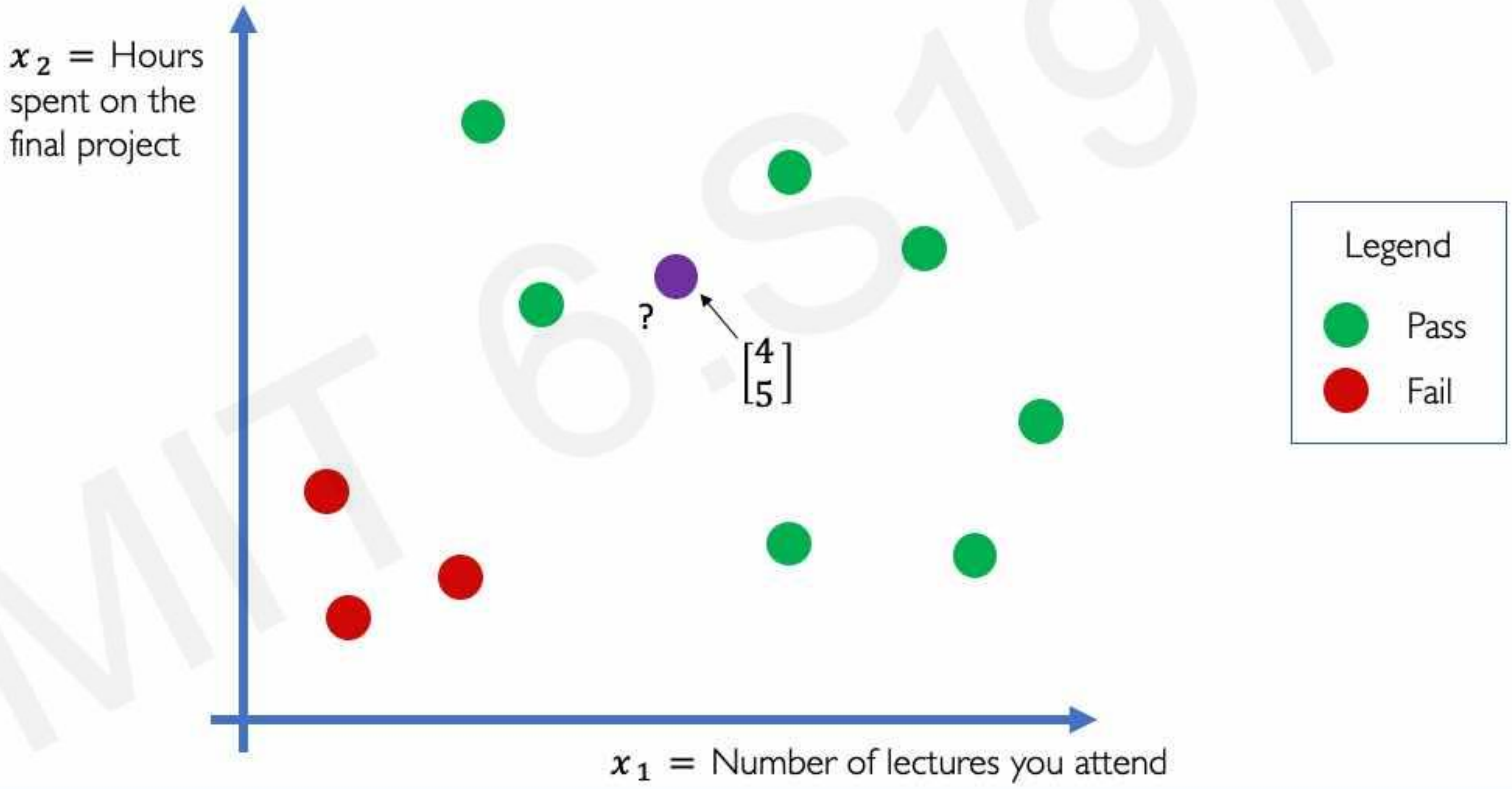
MIT 6.S191

Example Problem: Will I pass this class?



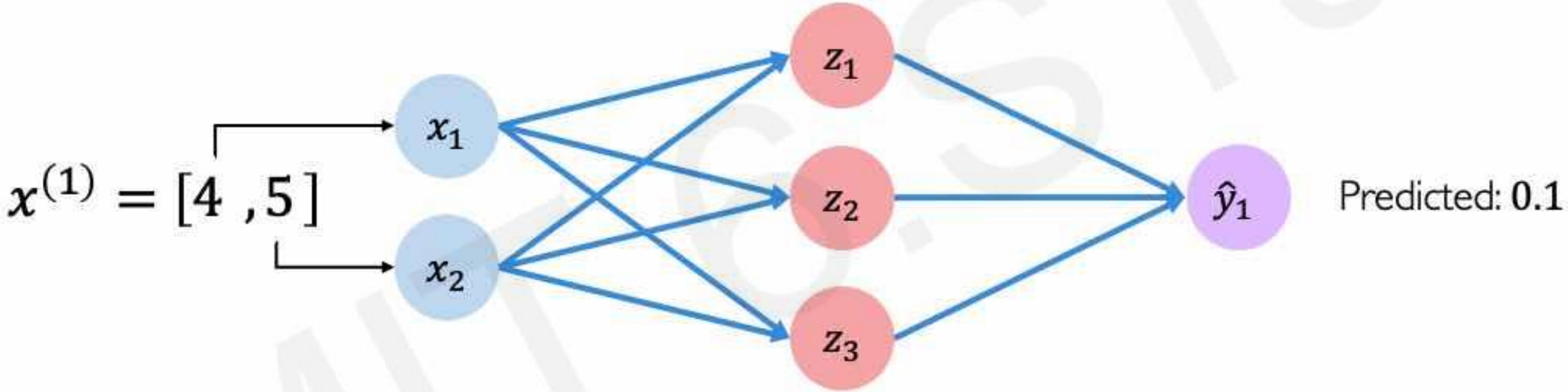
MIT 6.S191

Example Problem: Will I pass this class?



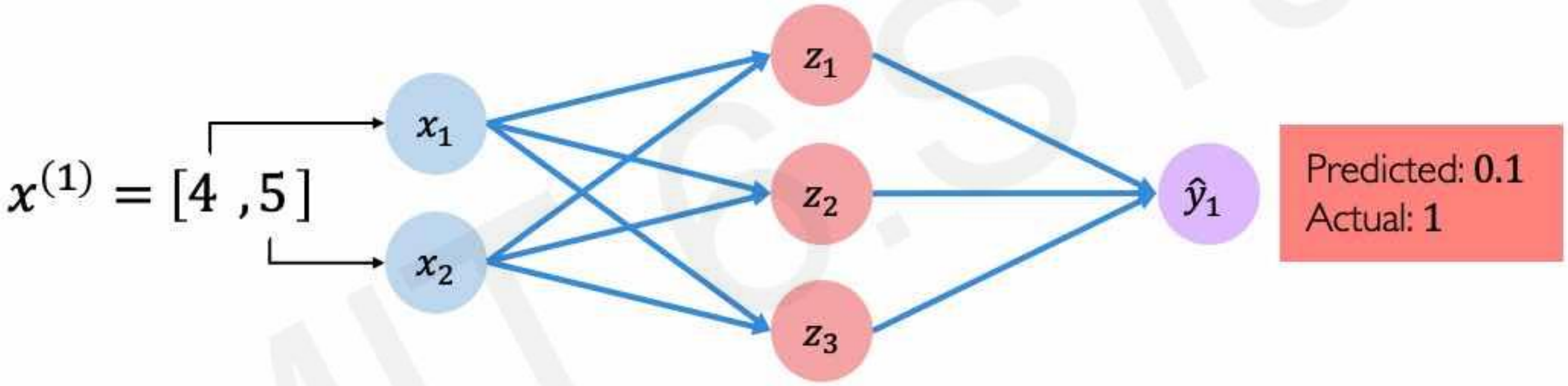
MIT 6.S191

Example Problem: Will I pass this class?



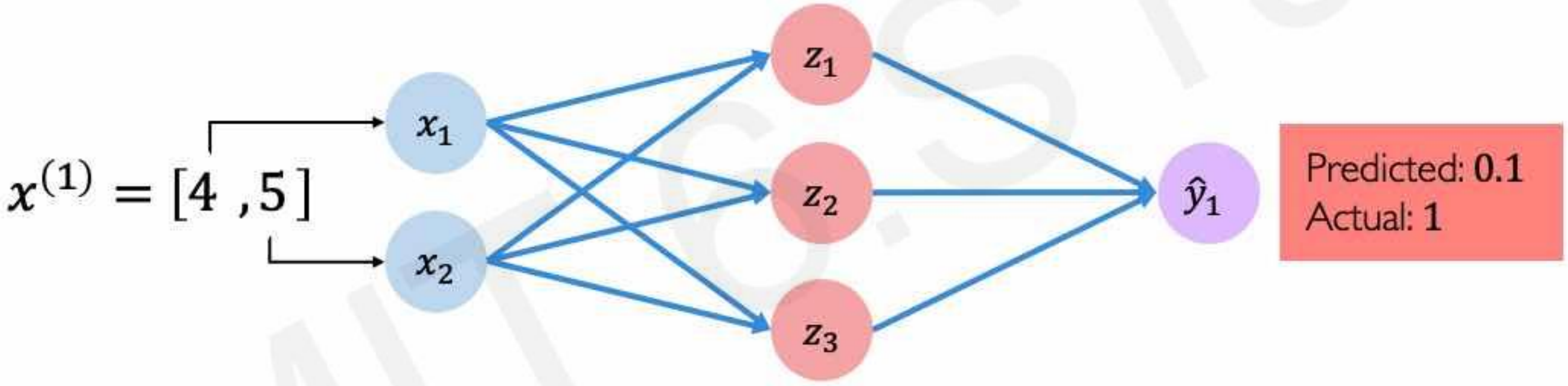
MIT 6.S191

Example Problem: Will I pass this class?



Quantifying Loss

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



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

MIT 6.S191