Computational Intelligence

Samaneh Hosseini

Isfahan University of Technology

Outline

- Building Neural Networks with Perceptrons
  - Multi Output Perceptron
  - Neural Network with 1 Hidden Layer
  - Deep Neural Network
- Applying Neural Networks
  - Binary Classification
  - Regression
  - Cross Entropy cost function
  - Mean Squared error cost function
- Logistic Regression

Building Neural Networks with Perceptrons

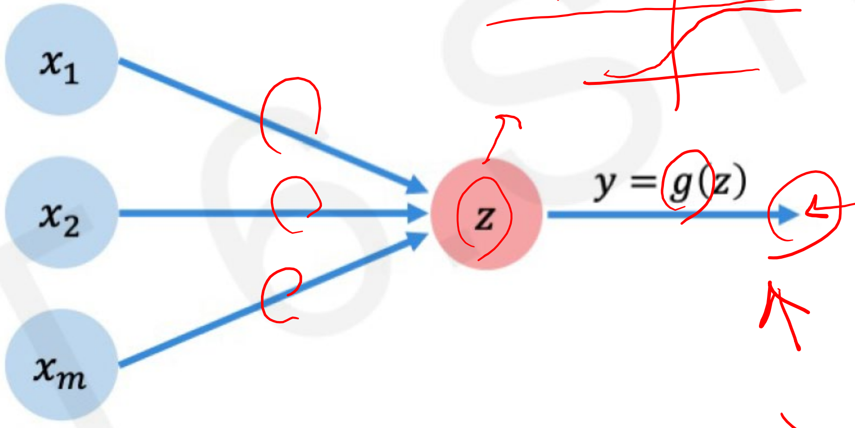
Multi Output Perceptron

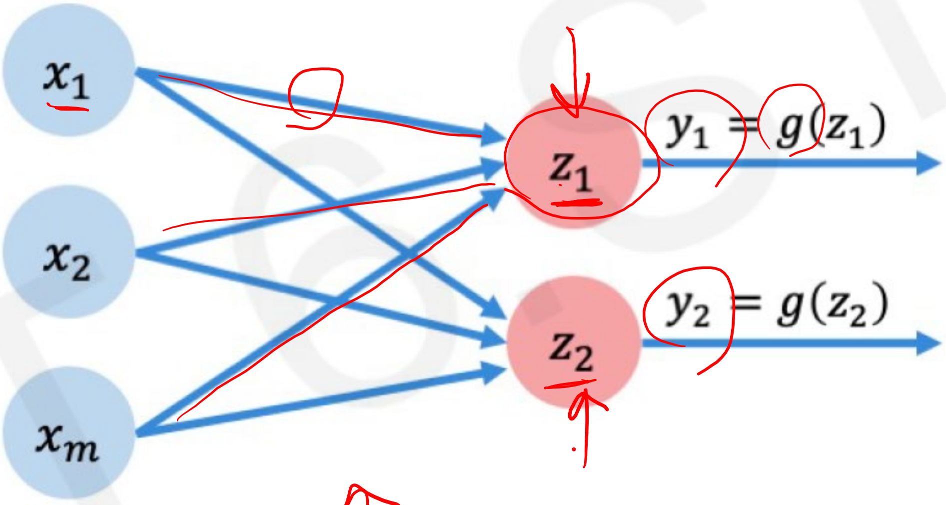
$c((ax+b))+d$

$c(ax)+cb+d$

1_ تعداد لایه ها

2_ تغییر activation Function





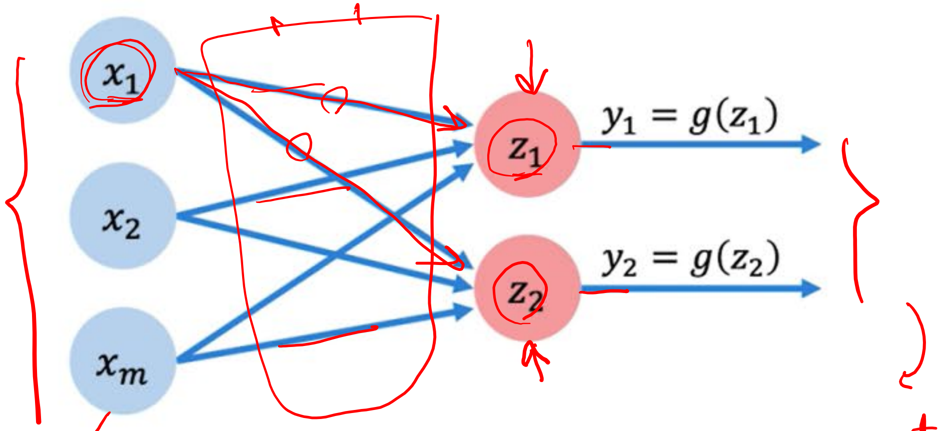
Perceptron : یک لایه از نورون ها

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

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

Multi Output Perceptron

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



m = input_dim

output_Dim = 2

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

Dense layer from scratch

```
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])
```

Dense layer from scratch

```
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])
```

Dense layer from scratch

```
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
```

ضرب برداری

1 x input_dim

input_dim x output_dim

= 1 x output_dim

1x output_dim

Dense layer from scratch

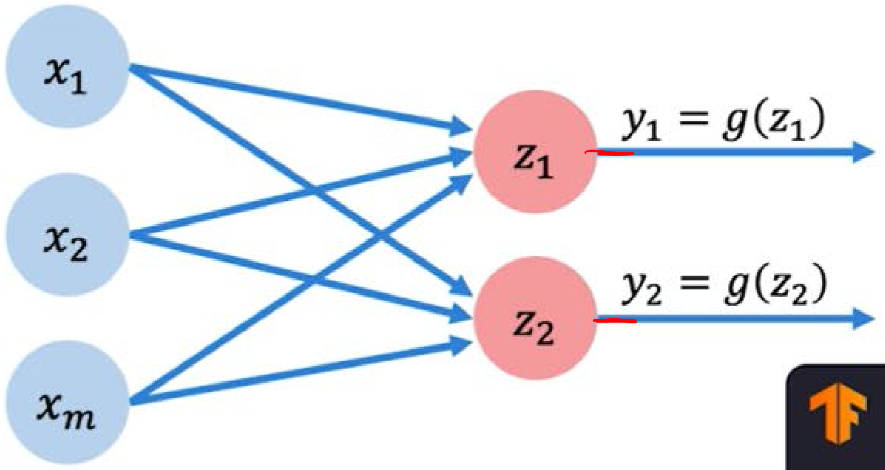
```
class MyDenseLayer(tf.keras.layers.Layer):
  def __init__(self, input_dim, output_dim):
    super(MyDenseLayer, self).__init__()

    # 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
```

Dense layer from scratch

```
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

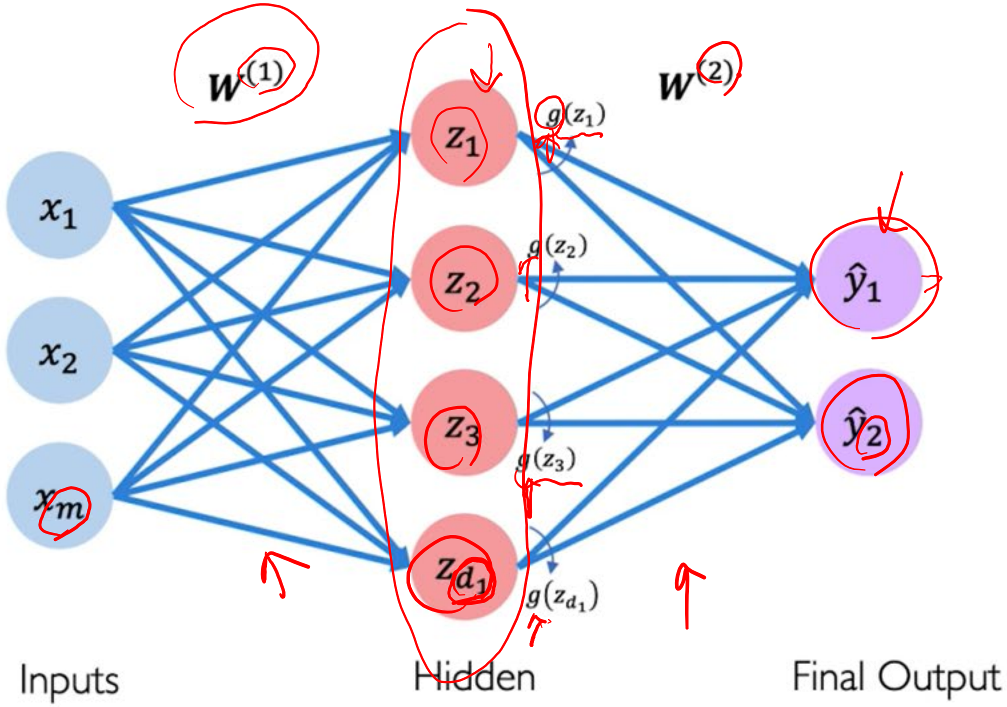
Because all inputs are densely connected to all outputs, these layers are called **Dense** layers




```python
import tensorflow as tf

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

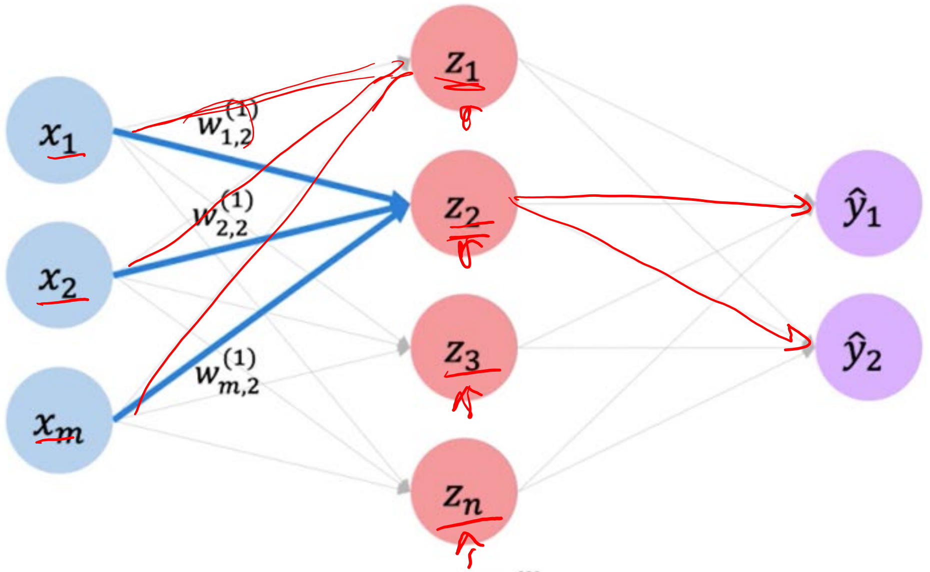
$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

Neural Network with 1 Hidden Layer



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

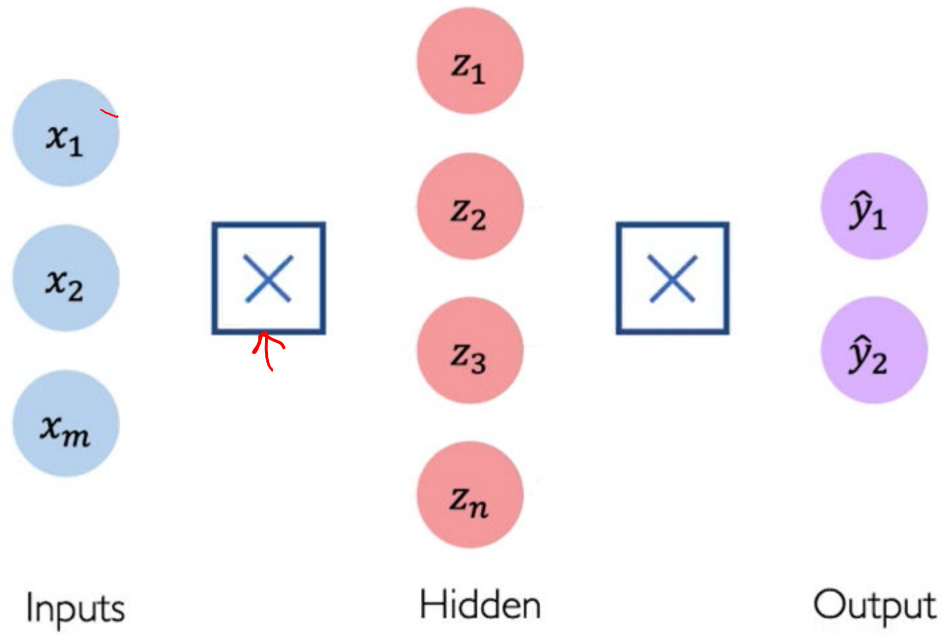
Neural Network with 1 Hidden Layer



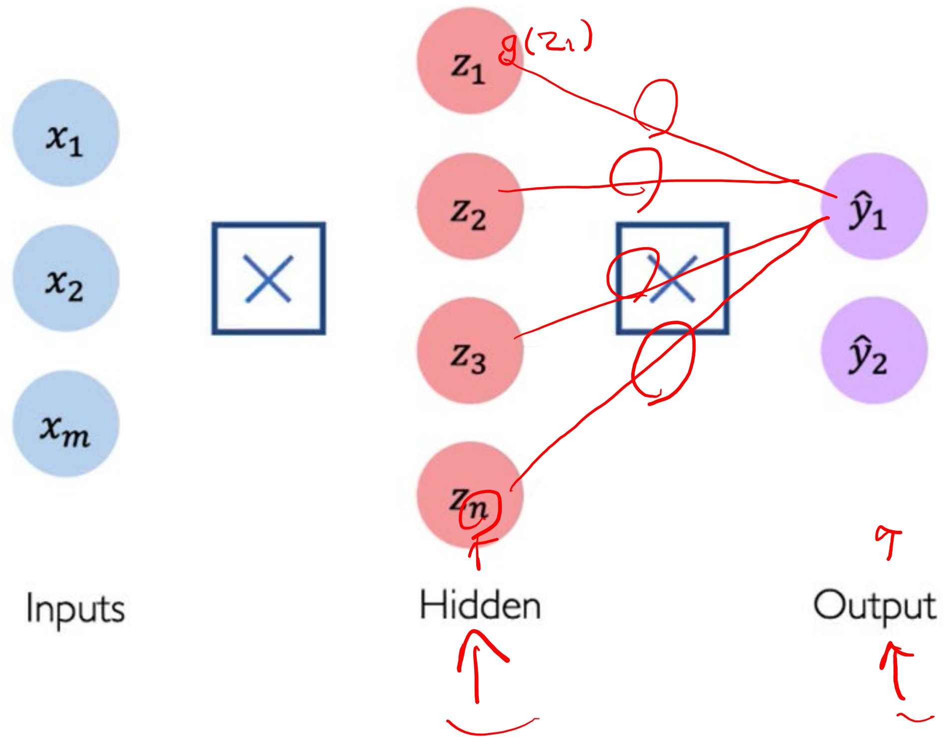
$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

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

Neural Network with 1 Hidden Layer

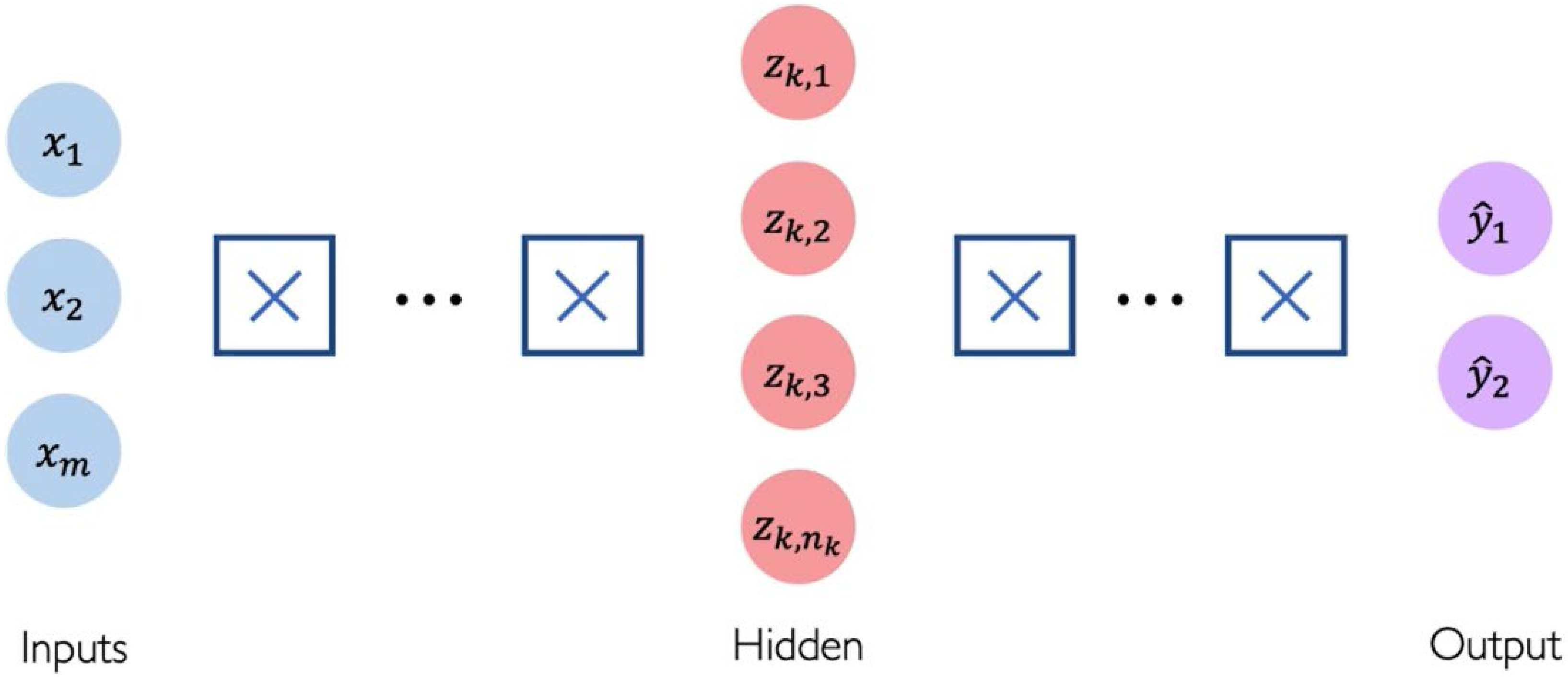


Neural Network with 1 Hidden Layer



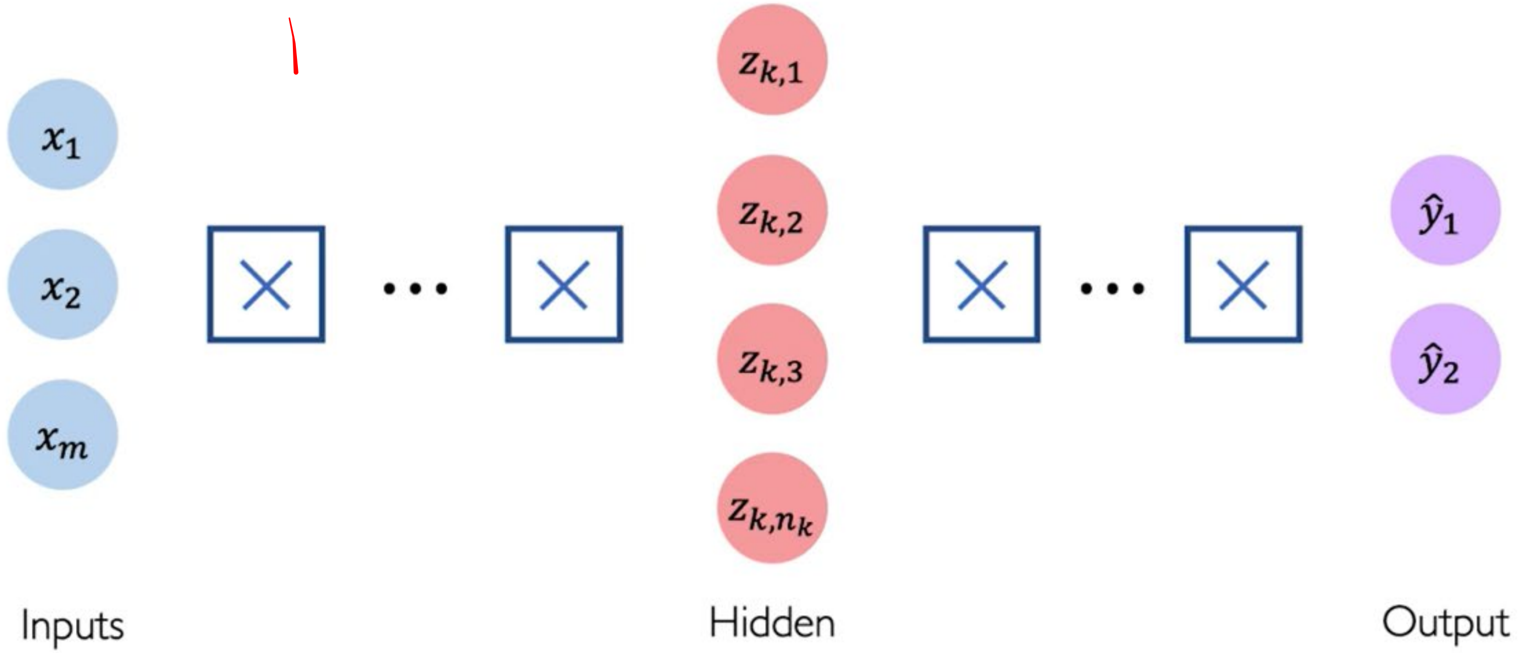

```python
import tensorflow as tf

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

Two types of supervised learning problems

- Classification: target features are discrete.
- Regression: target features are continuous.

Example Supervised Learning Problem

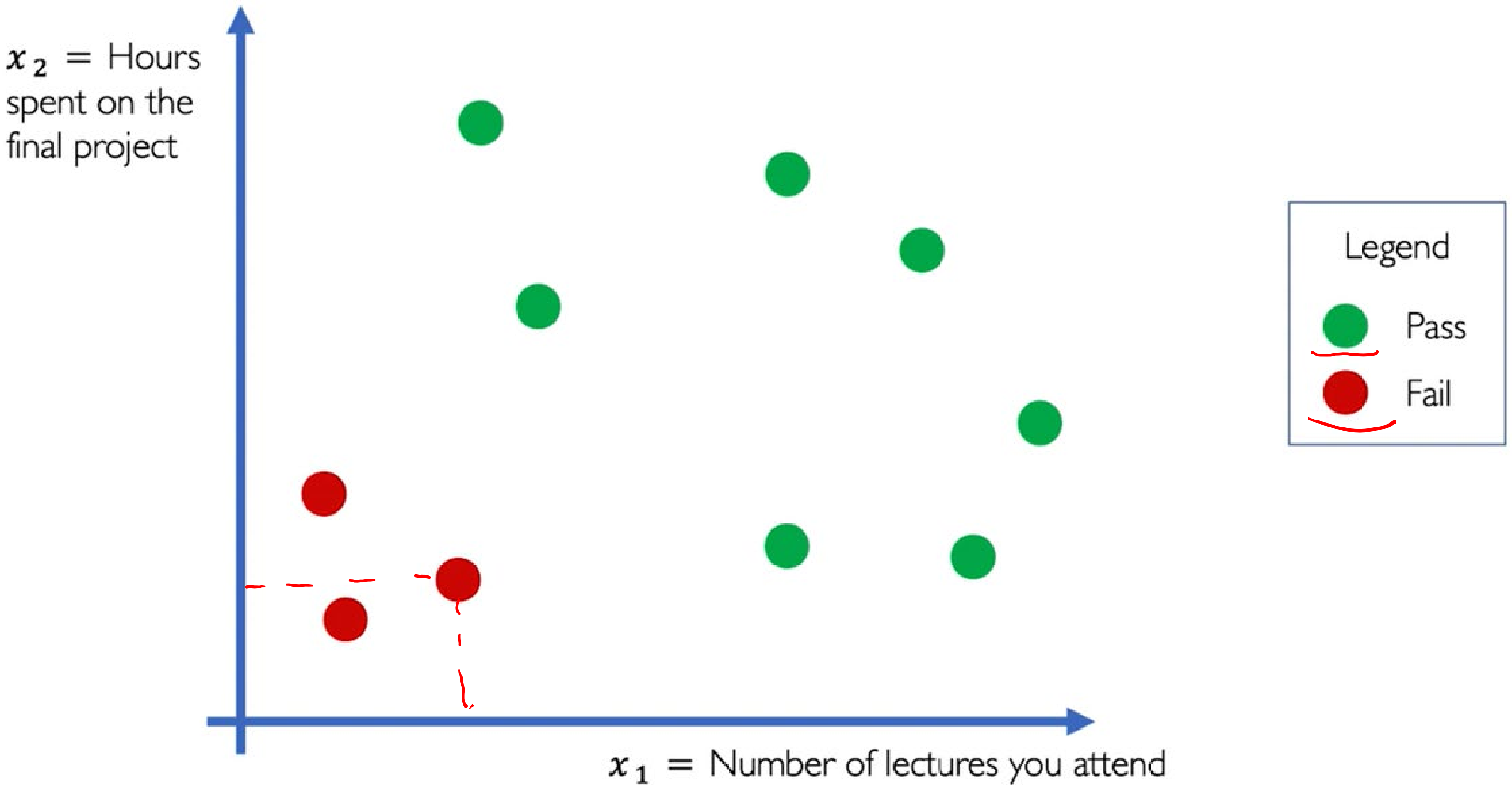
Will I pass this class?

Let's start with a simple two feature model

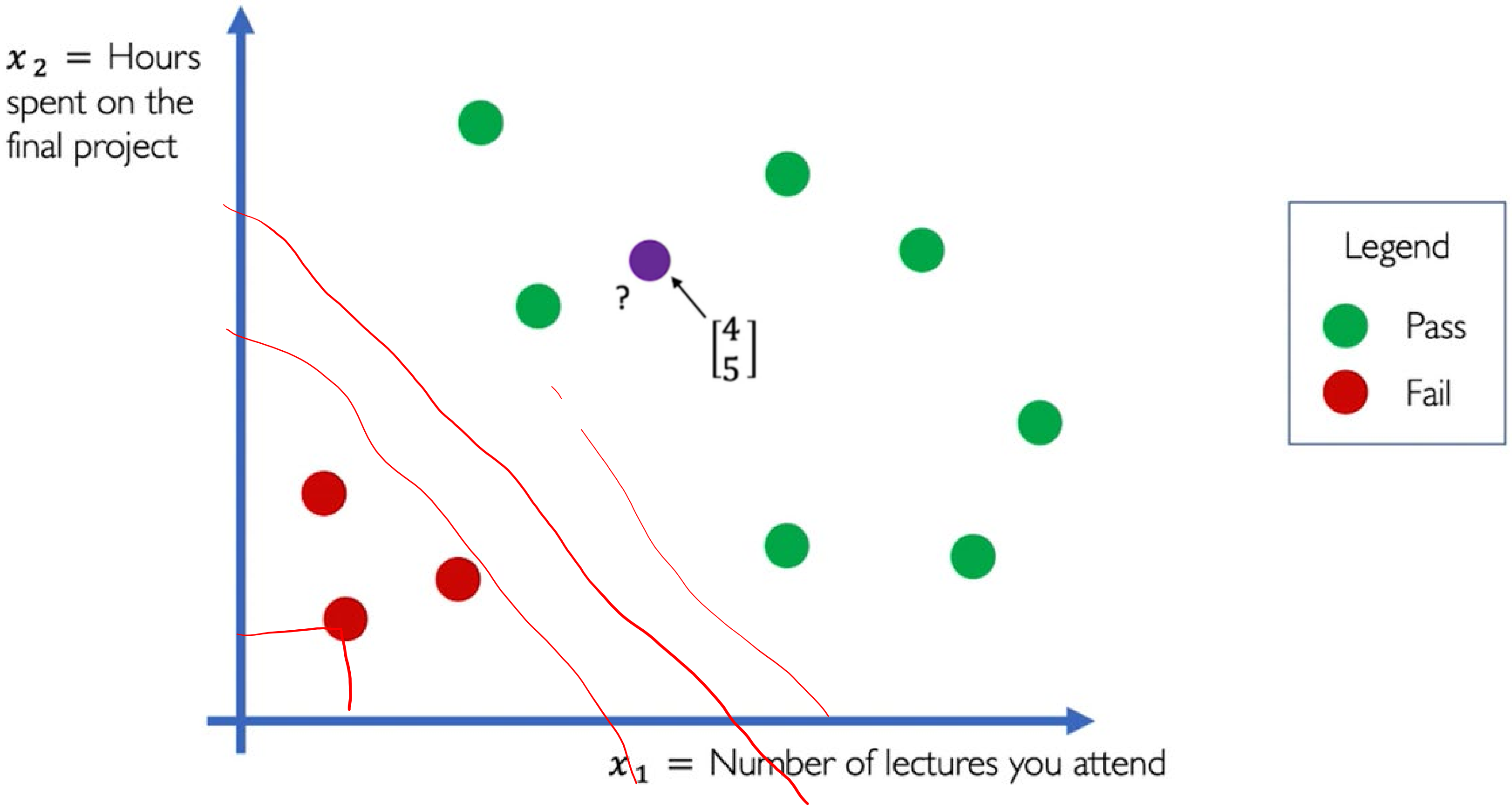
x_1 = Number of lectures you attend

x_2 = Hours spent on the final project

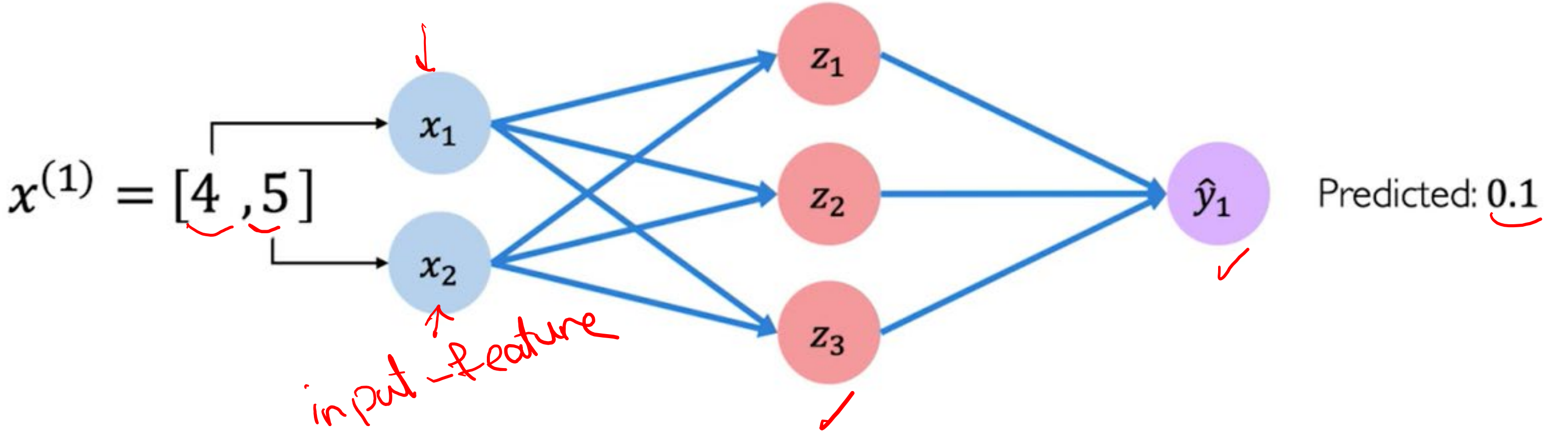
Example Problem: Will I Pass This Class?



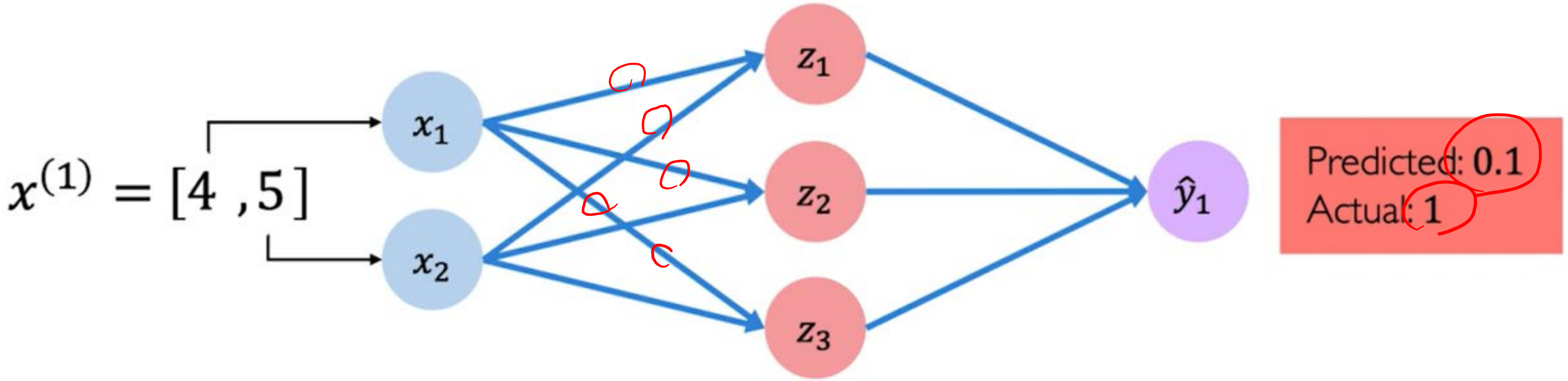
Example Problem: Will I Pass This Class?



Example Problem: Will I Pass This Class?

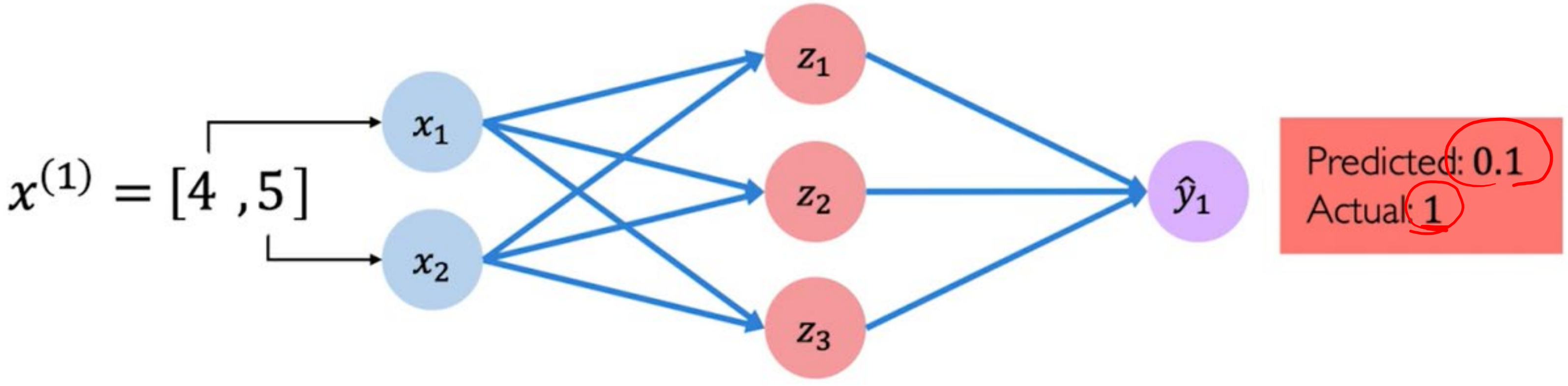


Example Problem: Will I Pass This Class?



Quantifying Loss

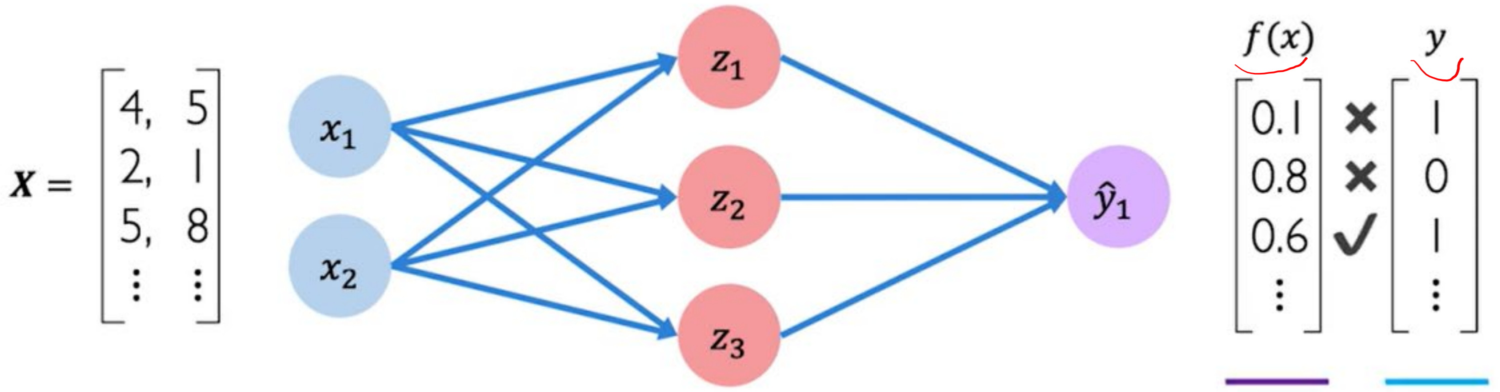
*The **loss** of our network measures the cost incurred from incorrect predictions*



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

Empirical Loss

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



Also known as:
- Objective function
- Cost function
- Empirical Risk

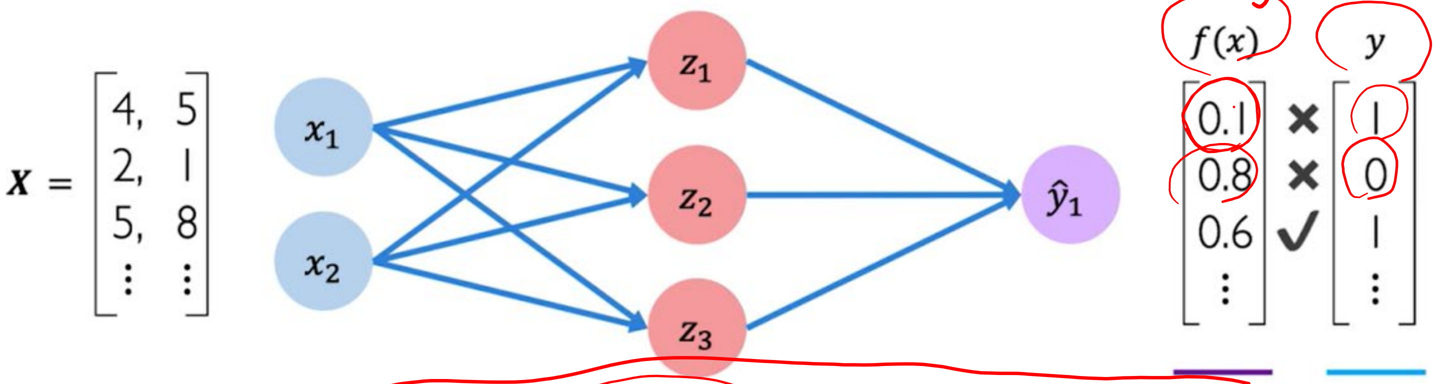
$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\big(\underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}},\underbrace{y^{(i)}}_{\text{Actual}}\big)$$

Binary Cross Entropy Loss / Log Loss

When the number of classes is 2, *Binary Classification*

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

$-\left(y \log \hat{y} + (1-y) \log (1-\hat{y})\right)$

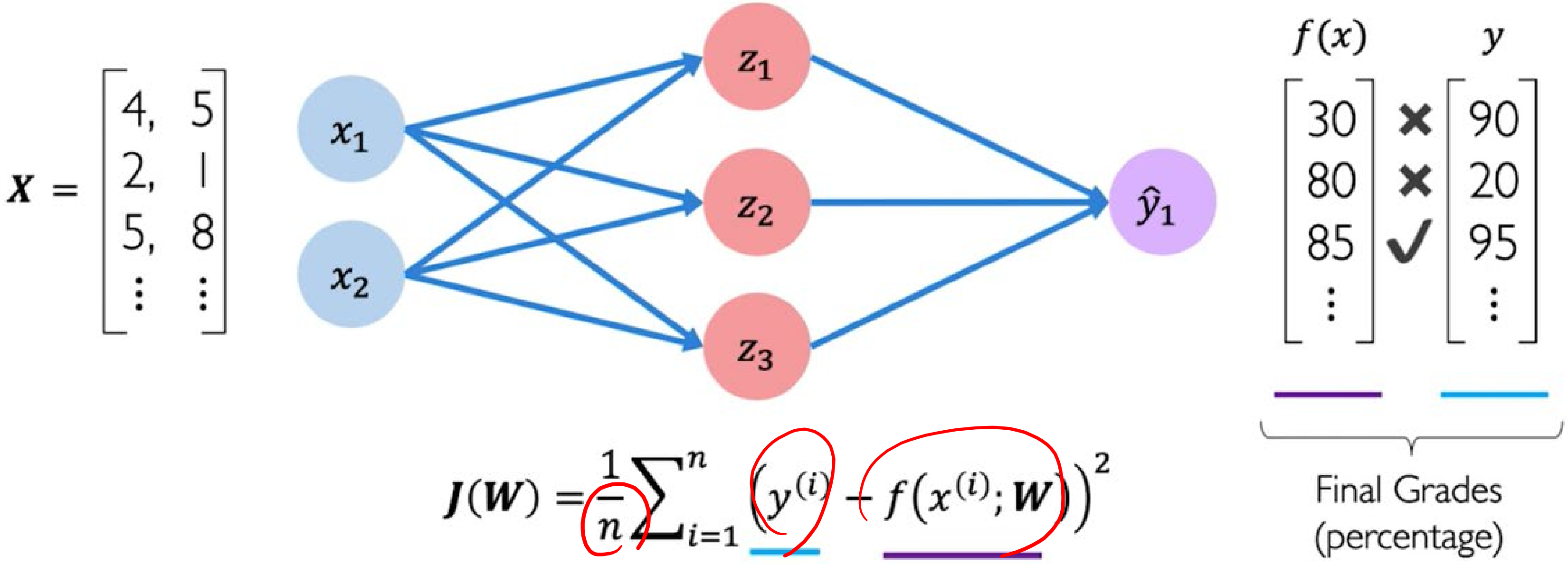


$$J(\boldsymbol{W}) = -\frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log\Big(\underbrace{f\big(x^{(i)}; \boldsymbol{W}\big)}_{\text{Predicted}}\Big) + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log\Big(1 - \underbrace{f\big(x^{(i)}; \boldsymbol{W}\big)}_{\text{Predicted}}\Big)$$

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

Mean Squared Error Loss: Regression Problem

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



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

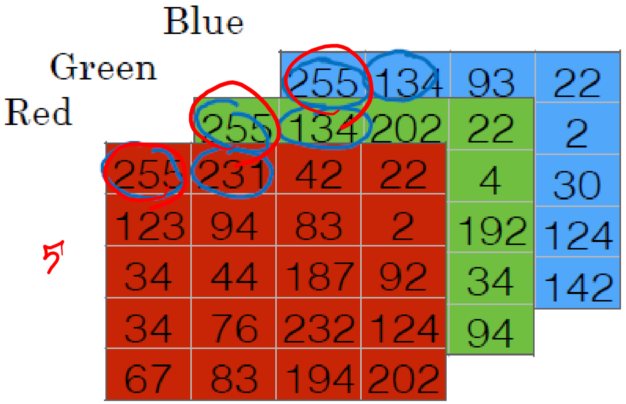
```
loss = tf.reduce_mean( tf.square(tf.subtract(y, predicted)) )
loss = tf.keras.losses.MSE( y, predicted )
```

Example: Binary Classification

64

64

1 (cat) vs 0 (non cat)



$$X = \begin{bmatrix} 255 \\ 231 \\ 42 \\ 22 \\ 123 \\ \vdots \\ 255 \\ \vdots \\ 255 \\ 134 \\ \vdots \end{bmatrix}$$

$64 \times 64 \times 3 = 12288$

$n_x = 12288$

$X \rightarrow$ model $\rightarrow y$

1 cat

0 non cat

Notation

(x, y) $\quad x \in \mathbb{R}^{n_x}$ $\quad y \in \{0, 1\}$

m traing example

$M = M_{train}$

$M_{test} = \#$ test example

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \Big\updownarrow n_x$$

$n_x \times m$

$X.\text{shape} = (n_x, m)$

$$Y = \begin{bmatrix} y^{(1)} & \cdots & y^{(m)} \end{bmatrix}_{1 \times m}$$

$Y \in \mathbb{R}^{1 \times m}$

$Y.\text{shape} = (1, m)$

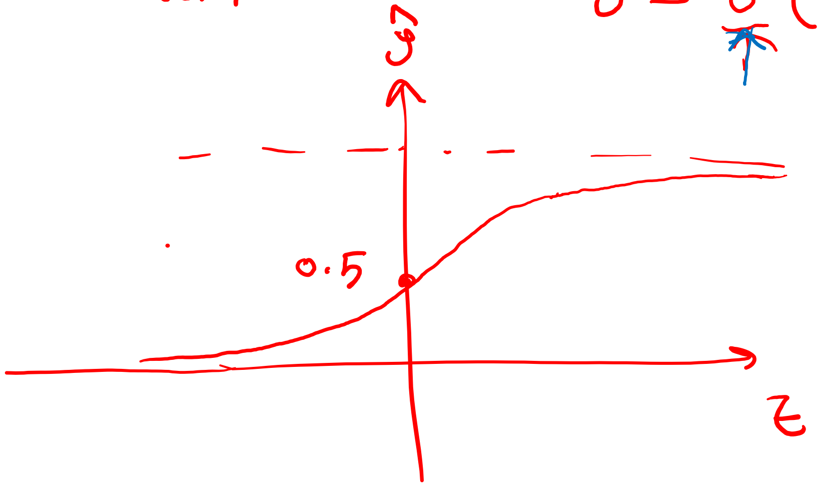
Logistic Regression

Given x want $\hat{y} = P(y=1 \mid x)$

$x \in \mathbb{R}^{n_x}$ $0 \leq \hat{y} \leq 1$

→ parameter $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

output $\hat{y} = \sigma(\underbrace{w^T x + b}_{z})$



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

if z large $\sigma(z) = \frac{1}{1+0} \approx 1$

if z large negative

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+\infty} \approx 0$$

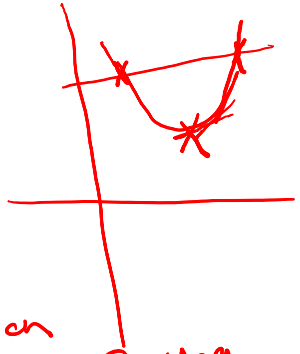
Logistic Regression cost function

$\hat{y}^{(i)} = \sigma\left(w^T x^{(i)} + b\right)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$

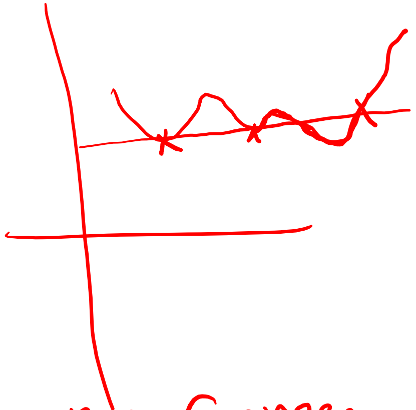
Given $(x^{(1)}, y^{(1)}),\dots,(x^{(m)}, y^{(m)})$, want $\hat{y}^{(i)} \approx y^{(i)}$.

Loss (error) function:

$\mathcal{L}(\hat{y}, y) = -\left(y \log \hat{y} + (1-y)\log(1-\hat{y})\right)$ ← Log function



Convex



non-Convex

if $y = 1$ (1) $\mathcal{L}(\hat{y}, y) = -\log \hat{y} \to \log(\hat{y})$ large $\to \hat{y}$ large $\Rightarrow \hat{y} \to 1$

if $y = 0$ (2) $-\log(1-\hat{y}) \Rightarrow \log(1-\hat{y})$ large $\to 1-\hat{y}$ large $\Rightarrow \hat{y}$ small

Cost function $J(w,b) = \frac{1}{m}\sum_{i=0}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=0}^{m}\left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$

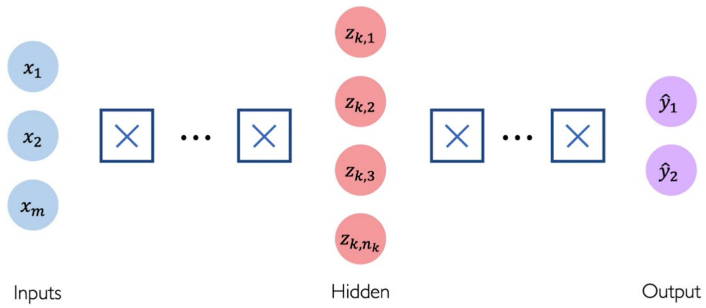
$\hat{y} \to 0$

Core Foundation Review

Deep NN

Empirical Loss

Logistic Regression



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