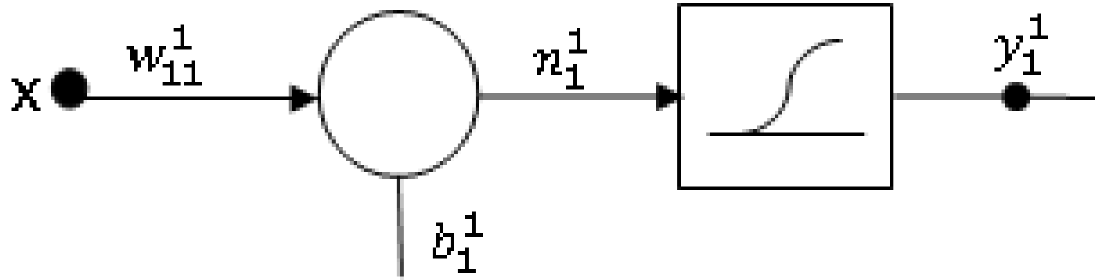
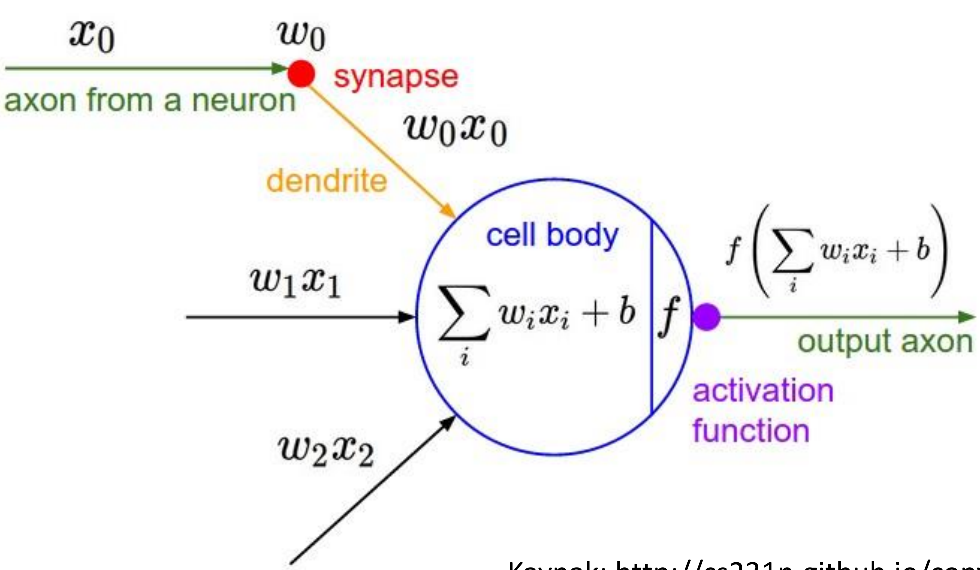
- **Neural Network Basics**
- **Activation functions**

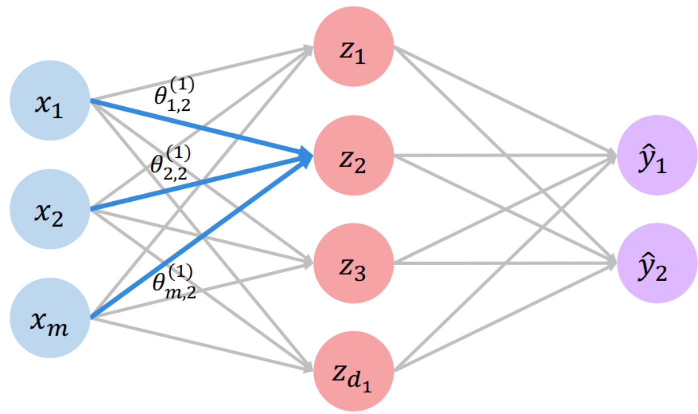
Artificial Neuron





Kaynak: http://cs231n.github.io/convolutional-networks/

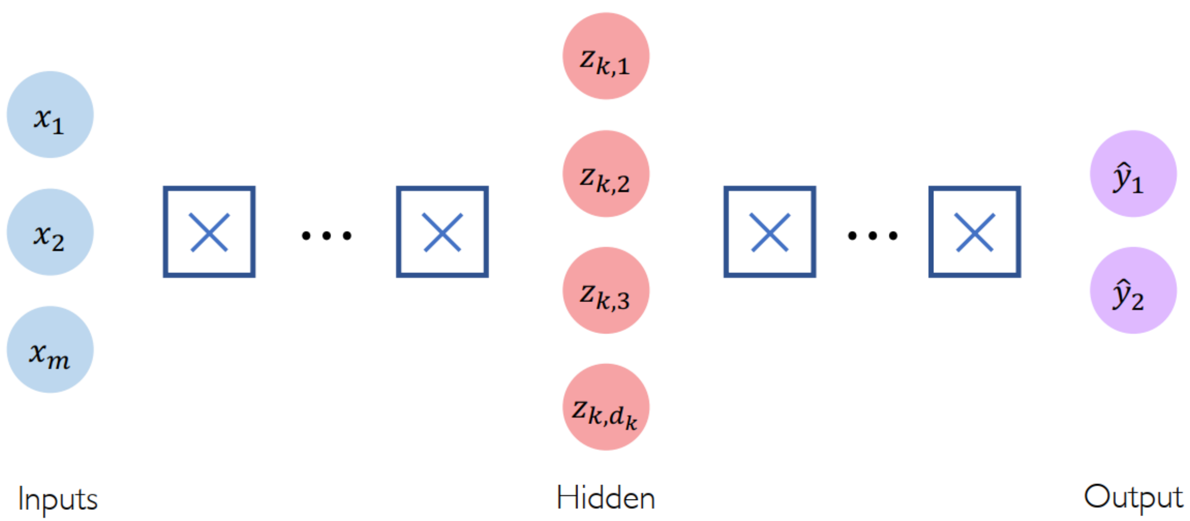
Yapay Sinir Ağları



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

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

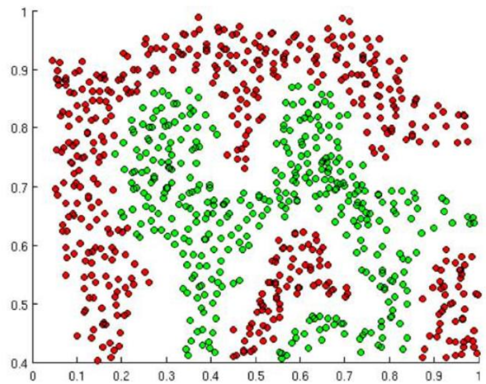
6.S191 Introduction to Deep Learning



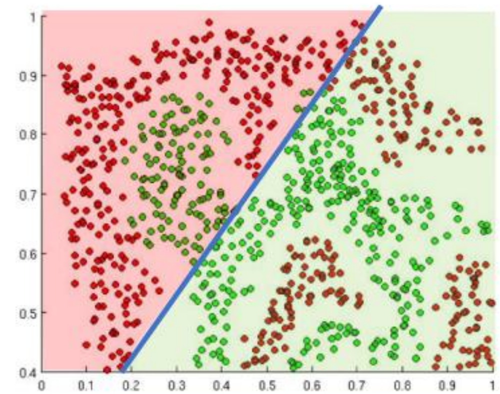
$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j}) \, w_{j,i}^{(k)}$$

Activation functions

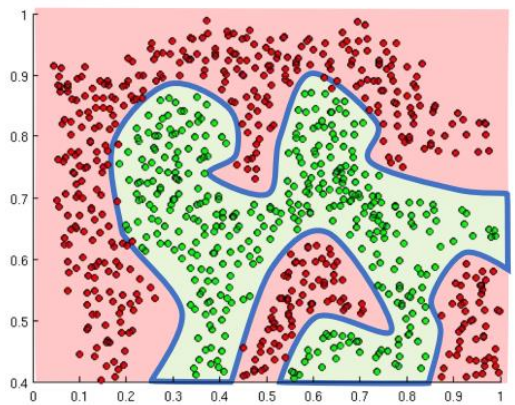
*The purpose of activation functions is to **introduce non-linearities** into the network*



What if we wanted to build a Neural Network to distinguish green vs red points?



Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

Activation functions

linear

```
keras.activations.linear(x)
```

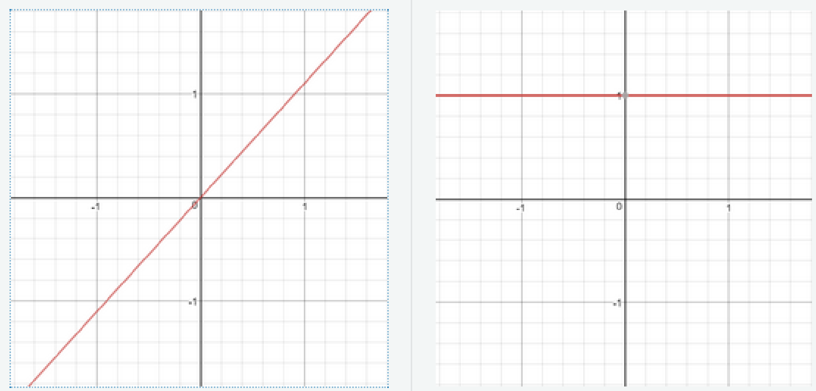
Linear (i.e. identity) activation function.

Arguments

- **x**: Input tensor.

Returns

Input tensor, unchanged.



relu

```
keras.activations.relu(x, alpha=0.0, max_value=None, threshold=0.0)
```

Rectified Linear Unit.

With default values, it returns element-wise `max(x, 0)`.

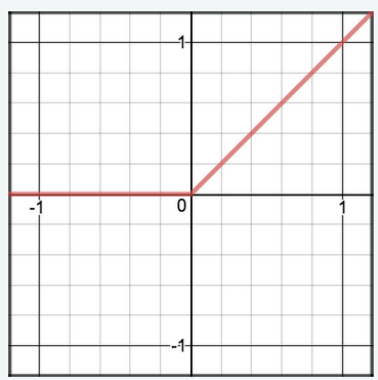
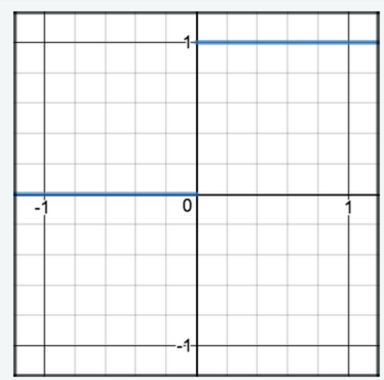
Otherwise, it follows: `f(x) = max_value` for `x >= max_value`, `f(x) = x` for `threshold <= x < max_value`, `f(x) = alpha * (x - threshold)` otherwise.

Arguments

- **x**: Input tensor.
- **alpha**: float. Slope of the negative part. Defaults to zero.
- **max_value**: float. Saturation threshold.
- **threshold**: float. Threshold value for thresholded activation.

Returns

A tensor.

Function	Derivative
$R(z) = \begin{Bmatrix} z & z > 0 \\ 0 & z <= 0 \end{Bmatrix}$	$R'(z) = \begin{Bmatrix} 1 & z > 0 \\ 0 & z < 0 \end{Bmatrix}$
	

Kaynak: https://keras.io/activations/
Kaynak: https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

Activation functions

sigmoid

```
keras.activations.sigmoid(x)
```

Sigmoid activation function.

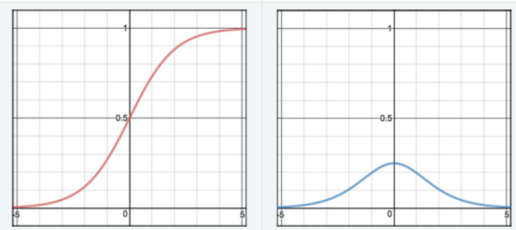
Arguments

- **x**: Input tensor.

Returns

The sigmoid activation: `1 / (1 + exp(-x))`.

Function	Derivative
$S(z) = \frac{1}{1+e^{-z}}$	$S'(z) = S(z) \cdot (1 - S(z))$



tanh

```
keras.activations.tanh(x)
```

Hyperbolic tangent activation function.

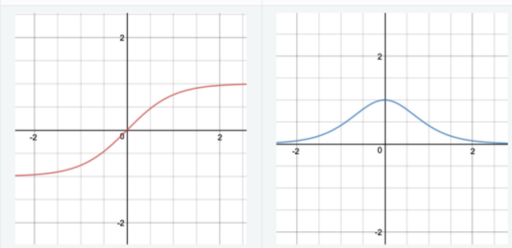
Arguments

- **x**: Input tensor.

Returns

The hyperbolic activation: `tanh(x) = (exp(x) - exp(-x)) / (exp(x) + exp(-x))`

Function	Derivative
$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$tanh'(z) = 1 - tanh(z)^2$



elu

```
keras.activations.elu(x, alpha=1.0)
```

Exponential linear unit.

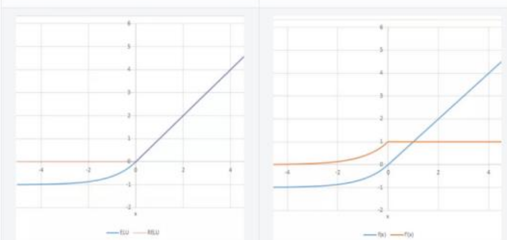
Arguments

- **x**: Input tensor.
- **alpha**: A scalar, slope of negative section.

Returns

The exponential linear activation: `x` if `x > 0` and `alpha * (exp(x)-1)` if `x < 0`.

Function	Derivative
$R(z) = \begin{Bmatrix} z & z > 0 \\ \alpha.(e^z - 1) & z <= 0 \end{Bmatrix}$	$R'(z) = \begin{Bmatrix} 1 & z > 0 \\ \alpha.e^z & z < 0 \end{Bmatrix}$



Kaynak: https://keras.io/activations/
Kaynak: https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

Activation functions

softmax

```
keras.activations.softmax(x, axis=-1)
```

Softmax activation function.

Arguments

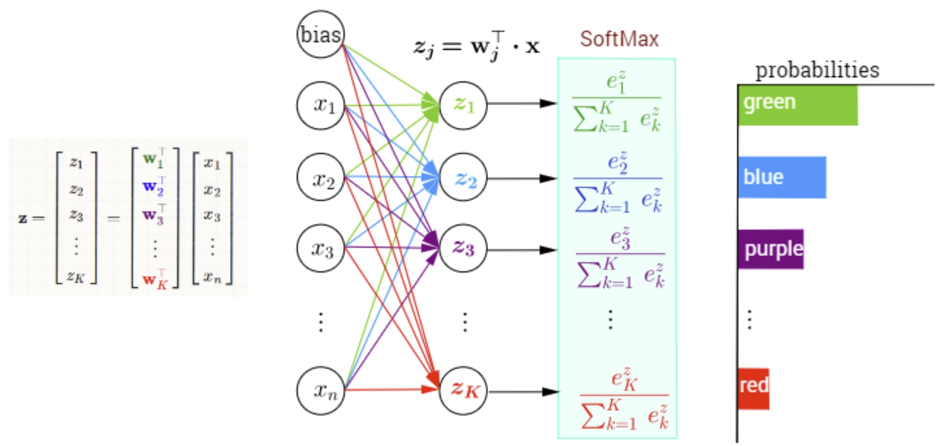
- **x**: Input tensor.
- **axis**: Integer, axis along which the softmax normalization is applied.

Returns

Tensor, output of softmax transformation.

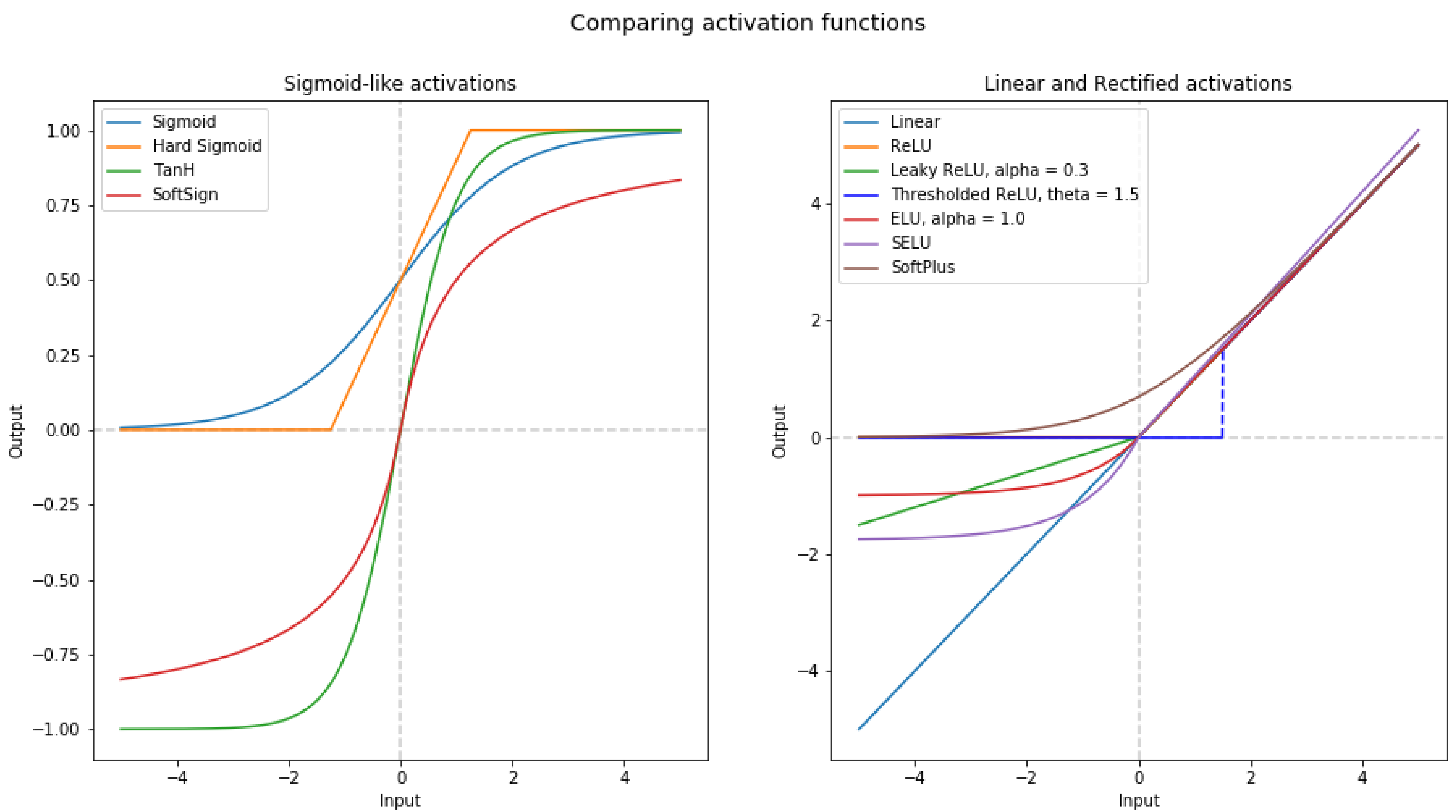
Raises

- **ValueError**: In case `dim(x) == 1`.



Kaynak: https://keras.io/activations/
Kaynak: https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

Activation functions



Kaynak: https://towardsdatascience.com/deep-study-of-a-not-very-deep-neural-network-part-2-activation-functions-fd9bd8d406fc

Example

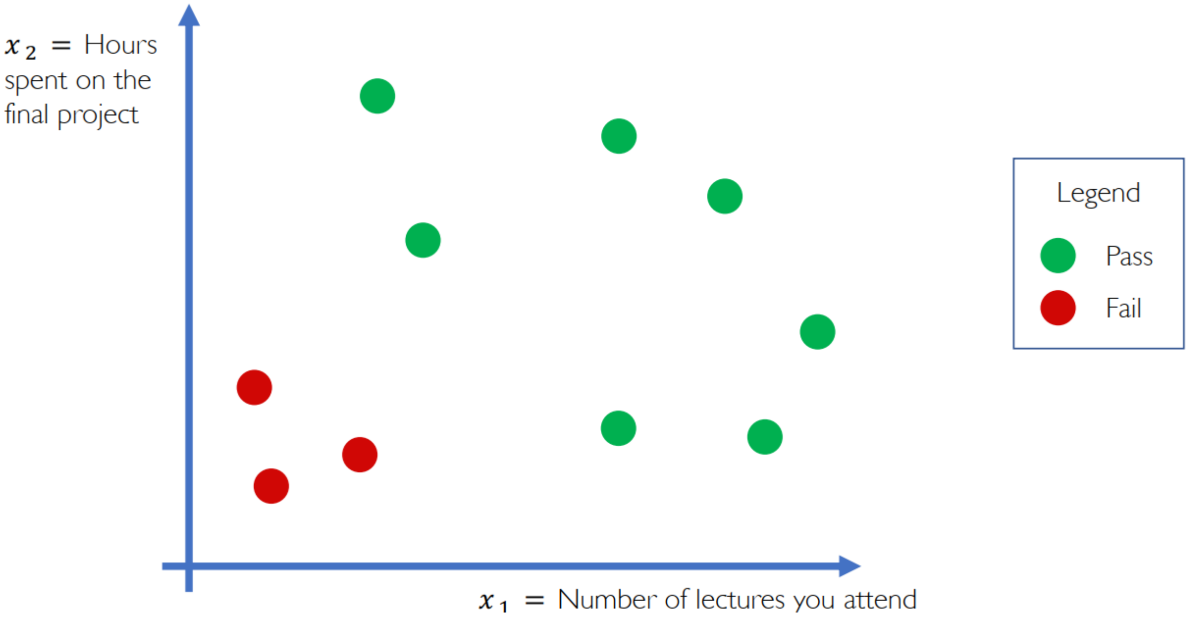
Example Problem

Will I pass this class?

Let's start with a simple two feature model

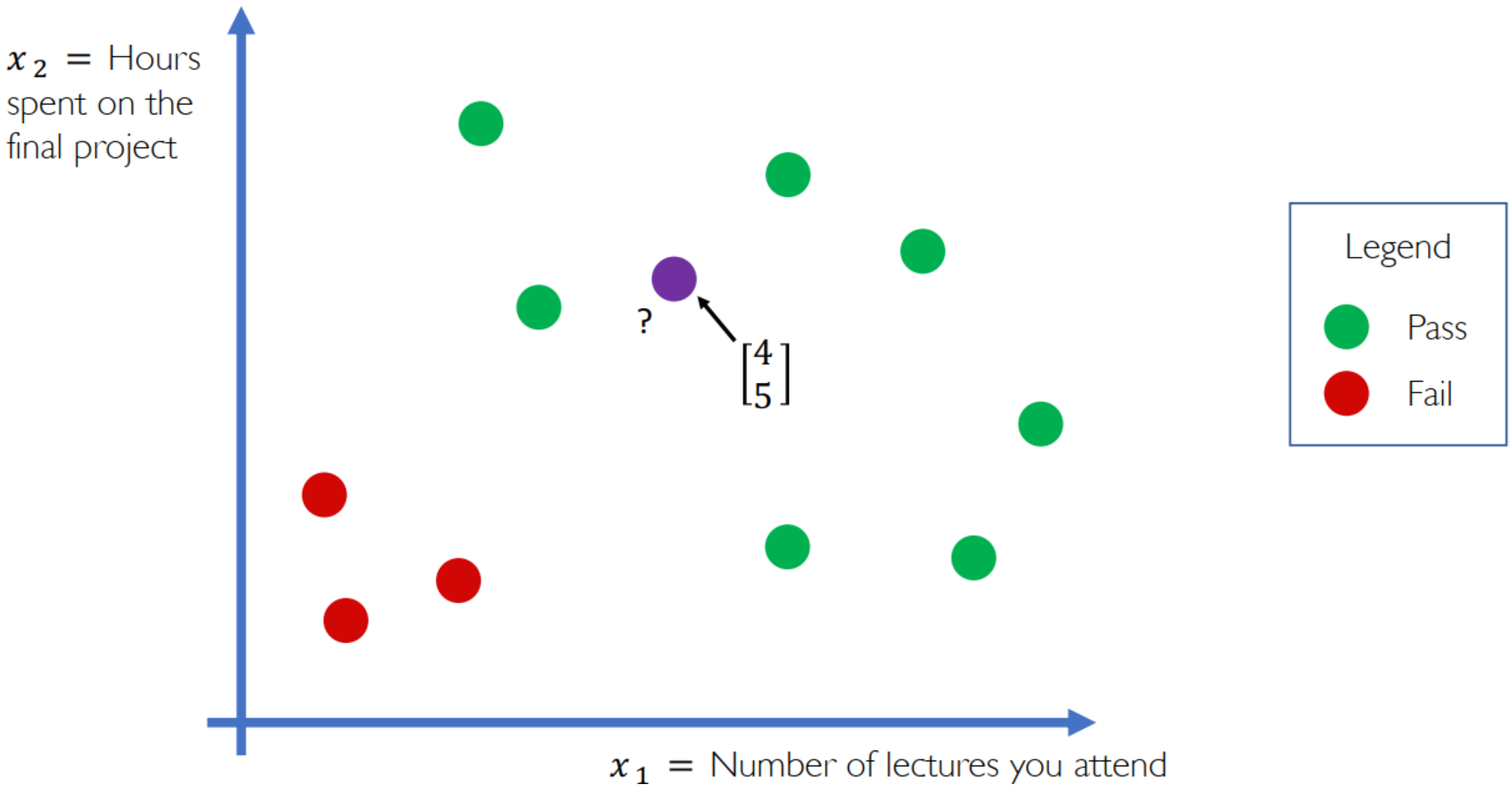
x_1 = Number of lectures you attend

x_2 = Hours spent on the final project

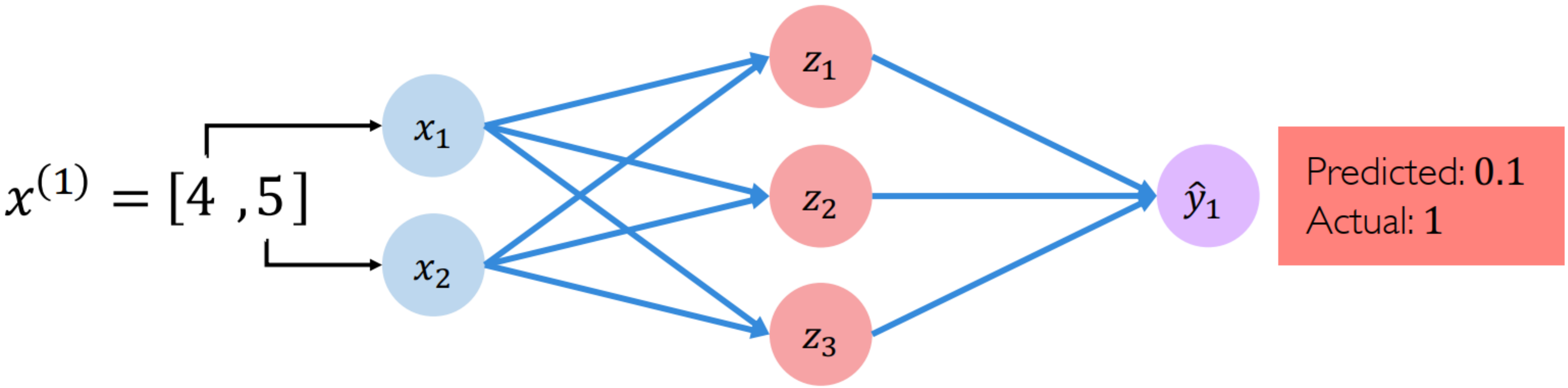


Example

Example Problem: Will I pass this class?



Example



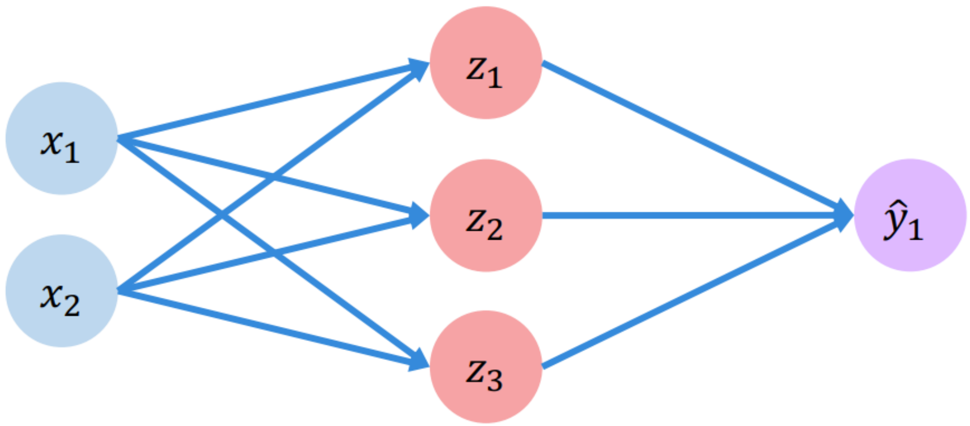
*The **loss** of our network measures the cost incurred from incorrect predictions*

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

Loss function

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

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



$$f(x) = \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \qquad y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

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

Also known as:

- Objective function
- Cost function
- Empirical Risk