Universidad de Guadalajara

Thompson Rivers University Seminar

Introduction to Machine Learning

Dr. Carlos Villaseñor

Content

Day 3 (Dr. Carlos Villaseñor)

- Introduction to neural networks
  - Biological neuron
  - Artificial neuron
  - Perceptron algorithm

Rest

  - Gradient Descent
  - Linear Neurons and Logistic neurons (MSE and BCE)
  - One-layer Network (Softmax and CCE)
  - How to model the last layer of a net

Rest

  - Multi-layer Perceptron
  - Keras/TensorFlow framework

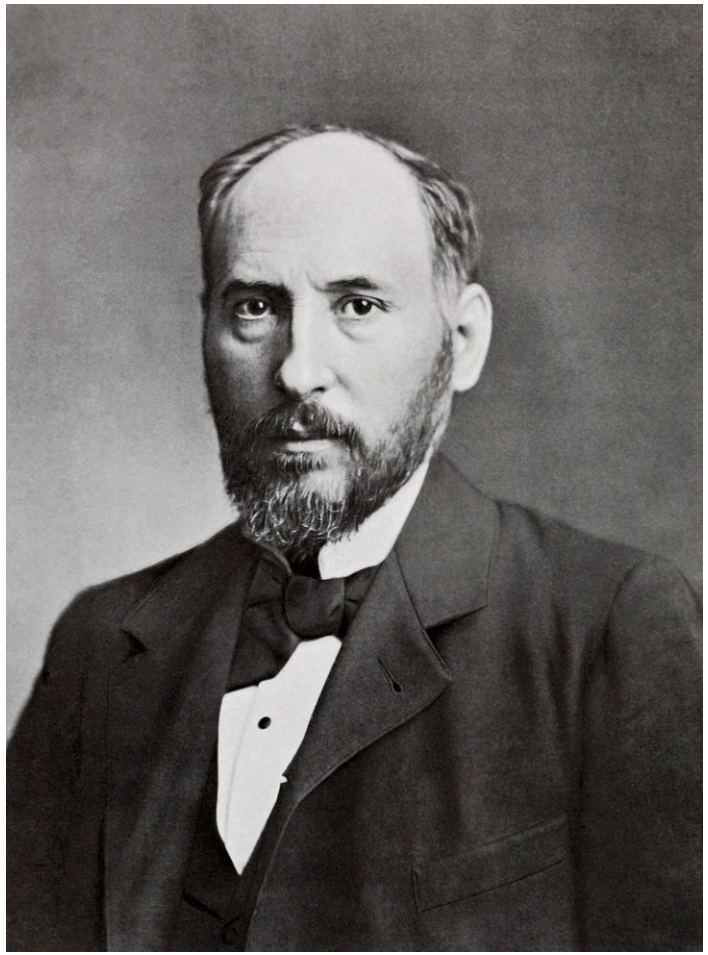
Course repository

https://github.com/Dr-Carlos-Villasenor/TRSeminar.git

The Biological Neuron

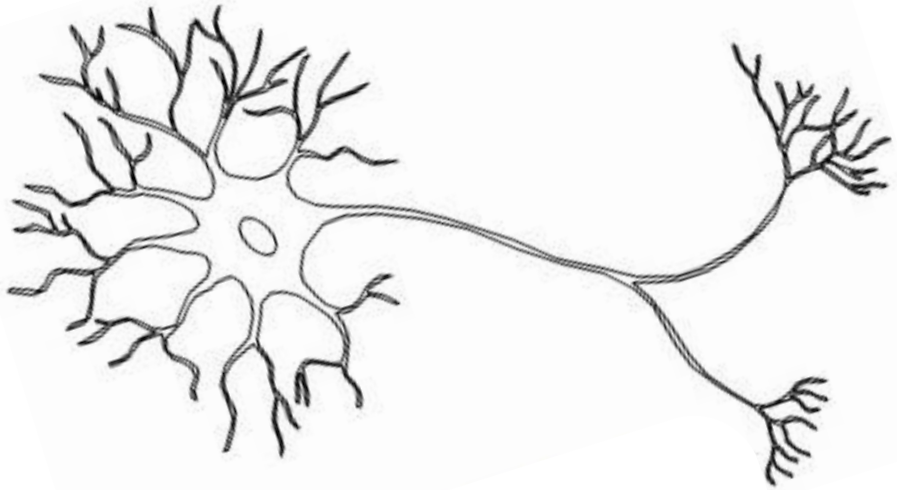
- In 1906, Santiago Ramón y Cajal won the Nobel Prize in Medicine in recognition of his work on the structure of the nervous system.

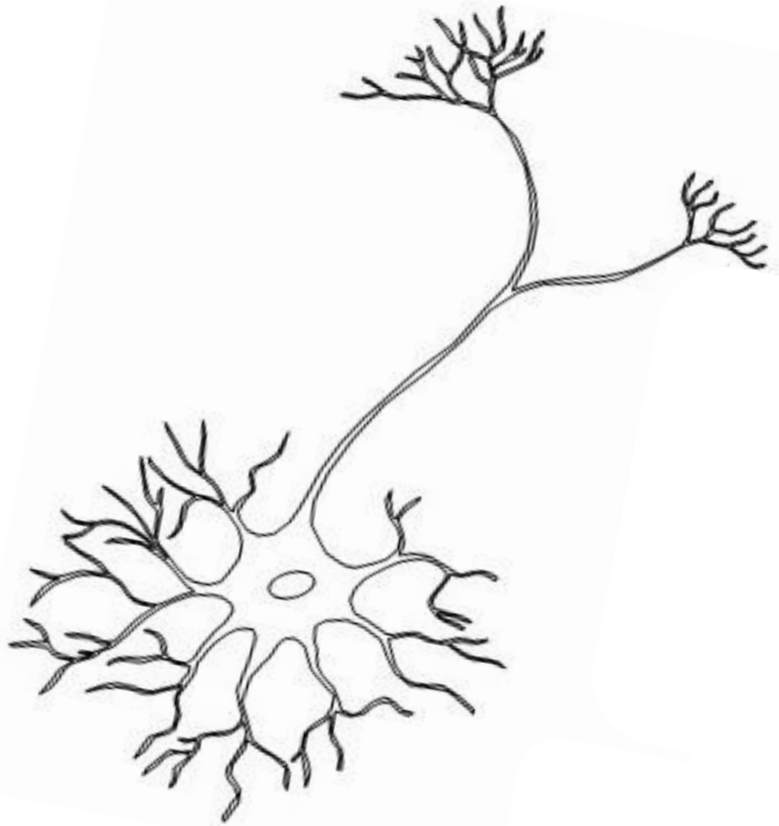
- He described neurons as information-processing units that connect and form dynamic networks to fulfill all the necessary functions.



Ramón, S. (1906). Cajal. *The Nobel Prize in Physiology or Medicine*.

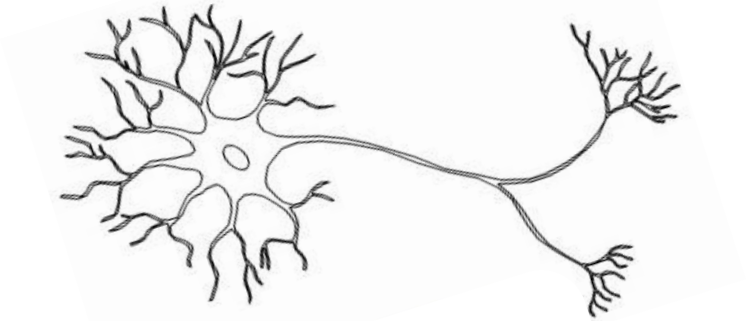
The Biological Neuron

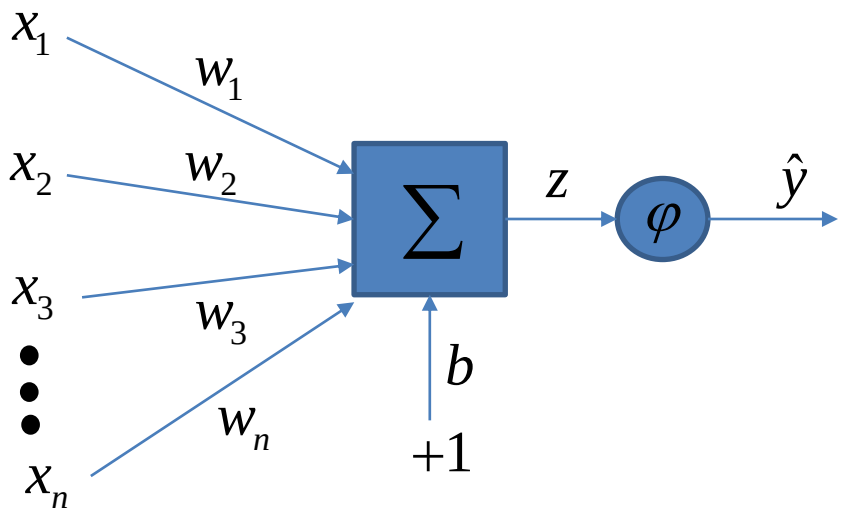




The artificial neuron

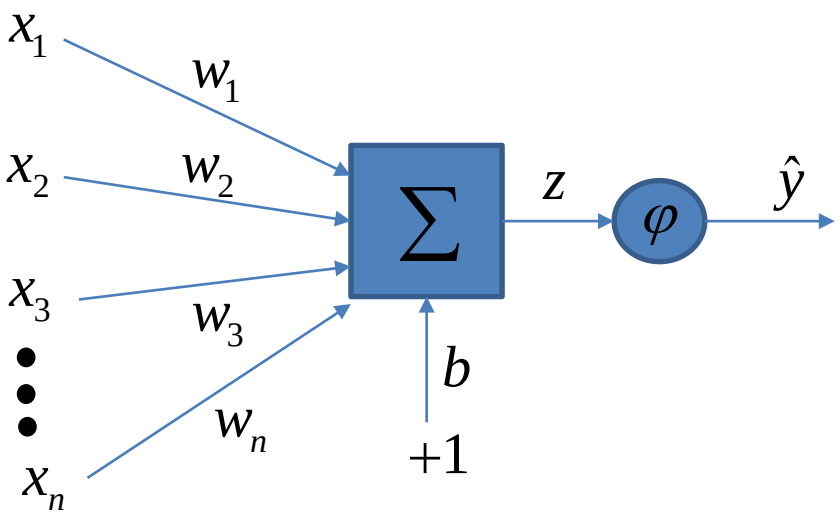
In 1943, McCulluch and Pitts developed the first mathematical model of a neuron





McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115-133.

The artificial neuron



$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

$$\hat{y} = \varphi(z)$$

$$\varphi(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115-133.

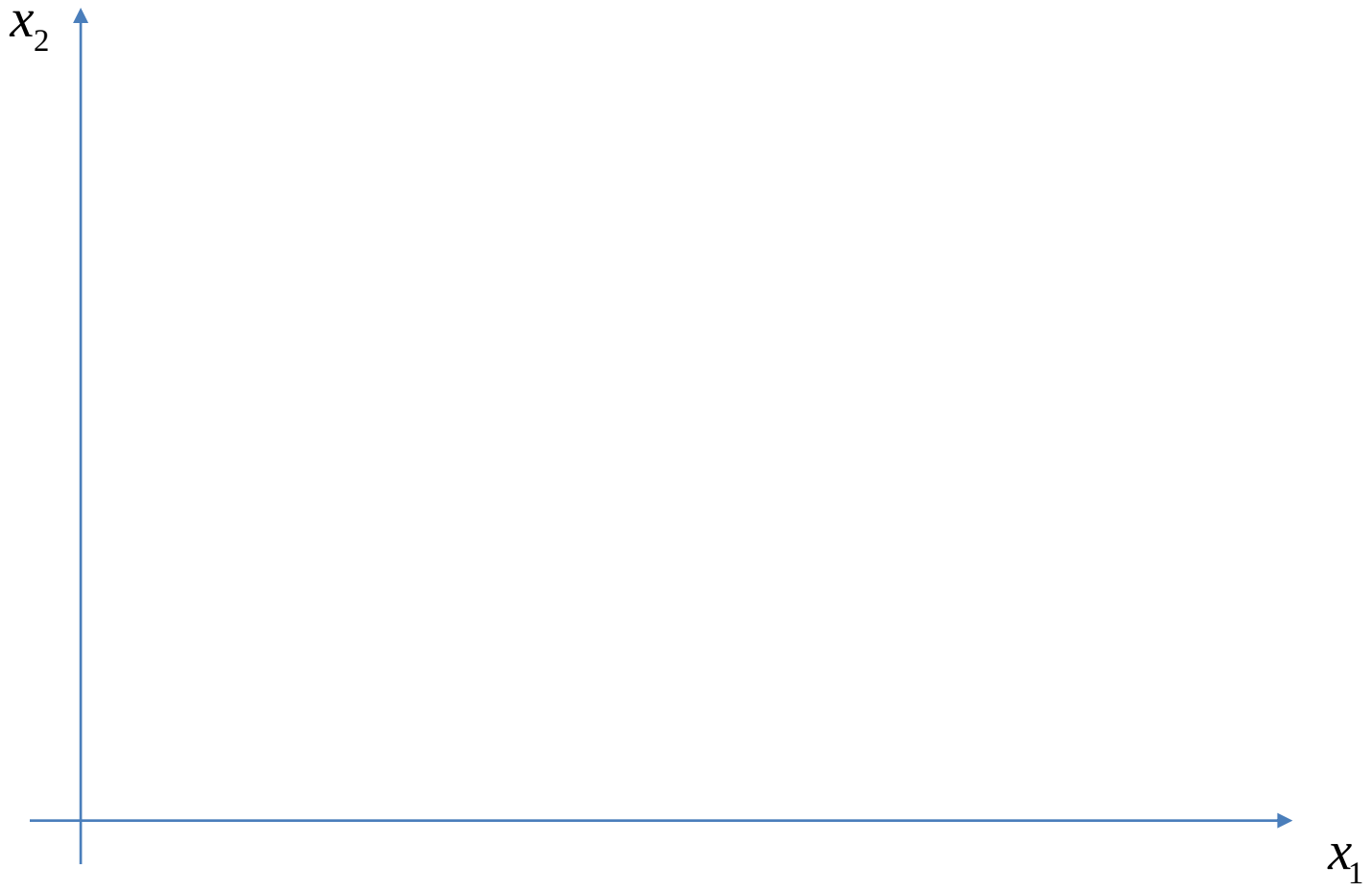
The artificial neuron

$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(p)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(p)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \cdots & x_n^{(p)} \end{bmatrix}$$

$$\hat{Y} = \begin{bmatrix} \hat{y}^{(1)} & \hat{y}^{(2)} & \cdots & \hat{y}^{(p)} \end{bmatrix}$$

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115-133.

The artificial neuron



Realizar actividad L01.2

The Perceptron algorithm

Data: $\left\{\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \cdots, \left(x^{(p)}, y^{(p)}\right)\right\}$ $\quad x^{(i)} \in \mathbb{R}^n, y^{(i)} \in \{0,1\}$

$$\text{For e} \in \{1, 2, \cdots, \text{epochs}\}$$

$$\text{For } i \in \{1, 2, \cdots, p\}$$

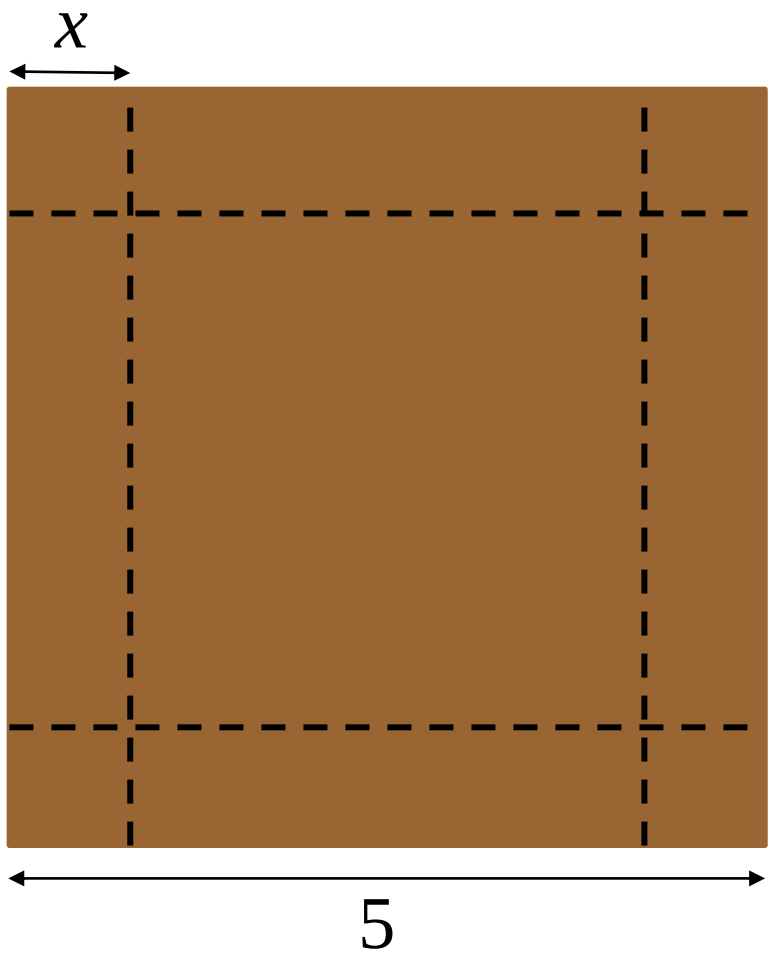
$$\hat{y} = \varphi\left(w^T x^{(i)} + b\right)$$

$$w \leftarrow w + \eta\left(y^{(i)} - \hat{y}\right) x^{(i)}$$

$$b \leftarrow b + \eta\left(y^{(i)} - \hat{y}\right)$$

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.

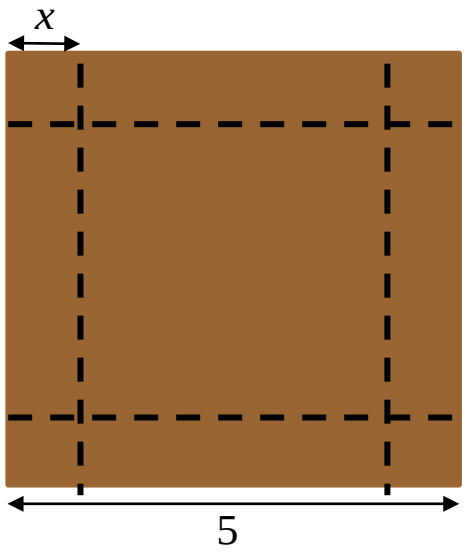
Optimization

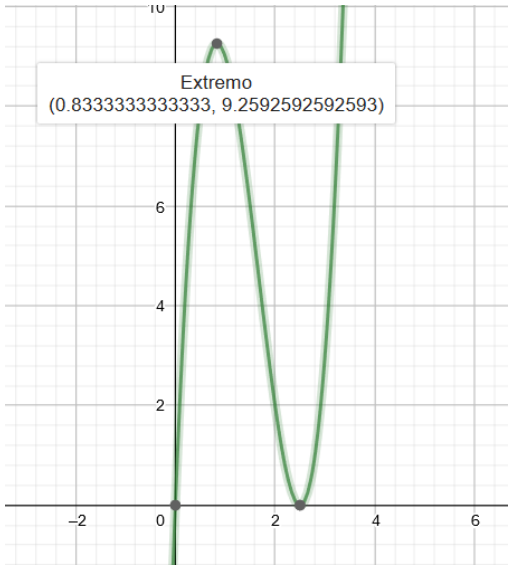


$$\max_{x} f(x)$$

x

Optimization





$$f(x) = x(5-2x)^2$$

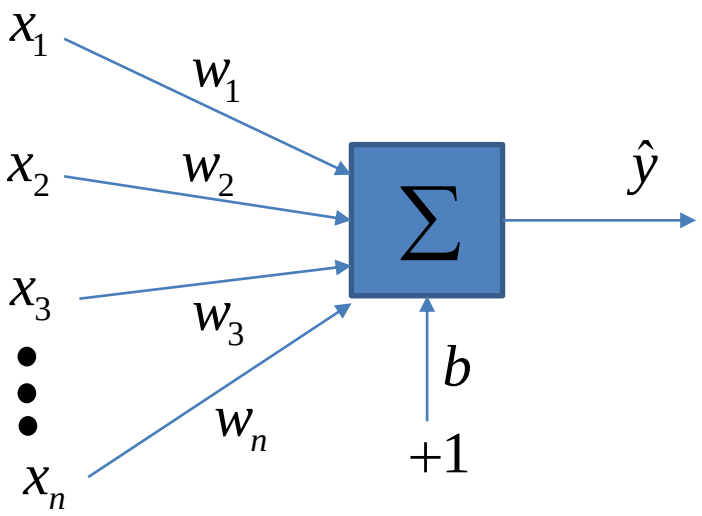
$$f(x) = 4x^3 - 20x^2 + 25x$$

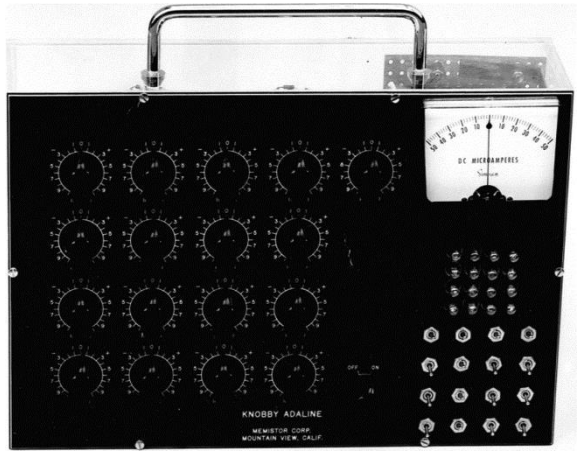
$$\frac{d}{dx} f(x) = 0$$

$$\frac{d}{dx} f(x) = 12x^2 - 40x + 25 = 0$$

$$x_{1,2} = \frac{5}{2}, \frac{5}{6}$$

Linear Neuron



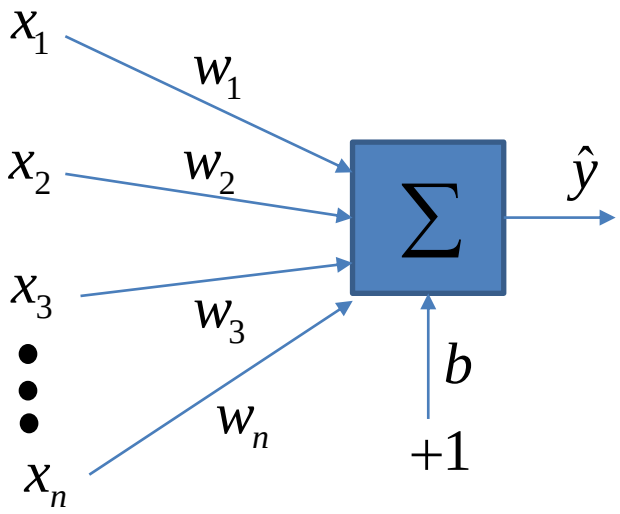


$$\hat{y} = w^T x + b \qquad x^{(i)} \in \mathbb{R}^n, y^{(i)} \in \mathbb{R}$$

Widrow, B. (1960). *Adaptive "adaline" Neuron Using Chemical "memistors.".*

Widrow, B. (1962). Generalization and information storage in network of ADALINE 'neurons'. *Self-organizing systems-1962*, 435-462.

Linear Neuron



$$\hat{y} = w^T x + b$$

Online Optimization Problem:

$$\min_{w,b} \frac{1}{2}\left(y - \hat{y}\right)^2$$

Global optimization problem (for the entire batch of data):

$$\min_{w,b} \frac{1}{2p}\sum_{i=1}^{p}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

$$D=\left\{\left(x^{(1)}, y^{(1)}\right),\left(x^{(2)}, y^{(2)}\right),\cdots,\left(x^{(p)}, y^{(p)}\right)\right\}$$

$$x^{(i)} \in \mathbb{R}^n, y^{(i)} \in \mathbb{R}$$

Gradient Descent (SGD)

$$\frac{\delta}{\delta w_i} L(y, \hat{y}) = -(y - \hat{y}) x_i$$

$$\nabla_w L(y, \hat{y}) = \begin{bmatrix} \frac{\delta}{\delta w_1} \\ \frac{\delta}{\delta w_2} \\ \vdots \\ \frac{\delta}{\delta w_n} \end{bmatrix} L(y, \hat{y}) = \begin{bmatrix} -(y - \hat{y}) x_1 \\ -(y - \hat{y}) x_2 \\ \vdots \\ -(y - \hat{y}) x_n \end{bmatrix} = -(y - \hat{y}) \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = -(y - \hat{y}) x$$

Propagate Multiple Entries

$$D=\left\{\left(x^{(1)},y^{(1)}\right),\left(x^{(2)},y^{(2)}\right),\cdots,\left(x^{(p)},y^{(p)}\right)\right\}$$

$$X=\begin{bmatrix} | & | & | & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(p)} \\ | & | & | & | \end{bmatrix}\in\mathbb{R}^{n\times p}$$

$$Y=\begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(p)} \end{bmatrix}\in\mathbb{R}^{1\times p}$$

$$\hat{y}=\begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix}\begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}+b$$

Broadcasting

$$\hat{y} = \begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix} \begin{bmatrix} | & | & | & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(p)} \\ | & | & | & | \end{bmatrix} + b$$

For greater comfort, let's redefine w

$$w = \begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix} \in \mathbb{R}^{1 \times n}$$

$$\underset{1 \times p}{\hat{Y}} = \underset{1 \times n}{w} \; \underset{n \times p}{X} + \underset{1 \times 1}{b}$$

Batch Descending Gradient (BGD)

$$E_2 = \frac{1}{2p}\sum_{i=1}^{p}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

$$X = \begin{bmatrix} | & | & | & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(p)} \\ | & | & | & | \end{bmatrix} \in \mathbb{R}^{n\times p}$$

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(p)} \end{bmatrix} \in \mathbb{R}^{1\times p}$$

$$\nabla_w E_2 \qquad \frac{\partial}{\partial b} E_2$$

Batch Descending Gradient (BGD)

$$E_2 = \frac{1}{2p}\sum_{i=1}^{p}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 \qquad \hat{y}^{(i)} = wx^{(i)} + b$$

$$\nabla_w E_2 = \nabla_w \left[\frac{1}{2p}\sum_{i=1}^{p}\left(y^{(i)} - \hat{y}^{(i)}\right)^2\right]$$

$$\nabla_w E_2 = \frac{1}{p}\sum_{i=1}^{p}\left[\nabla_w \frac{1}{2}\left(y^{(i)} - \hat{y}^{(i)}\right)^2\right]$$

$$\nabla_w E_2 = \frac{1}{p}\sum_{i=1}^{p}\left[-\left(y^{(i)} - \hat{y}^{(i)}\right)x^{(i)}\right]$$

$$\nabla_w E_2 = -\frac{1}{p}\sum_{i=1}^{p}\left[\left(y^{(i)} - \hat{y}^{(i)}\right)x^{(i)}\right]$$

Batch Descending Gradient (BGD)

For $e \in \{1, 2, \cdots, \text{epochs}\}$

$$\hat{Y} = wX + b$$

$$w \leftarrow w + \frac{\eta}{p}\left(Y - \hat{Y}\right)X^T$$

$$b \leftarrow b + \frac{\eta}{p}\sum_{i=1}^{p}\left(y^{(i)} - \hat{y}^{(i)}\right)$$

$$w \in \mathbb{R}^{1\times n}$$

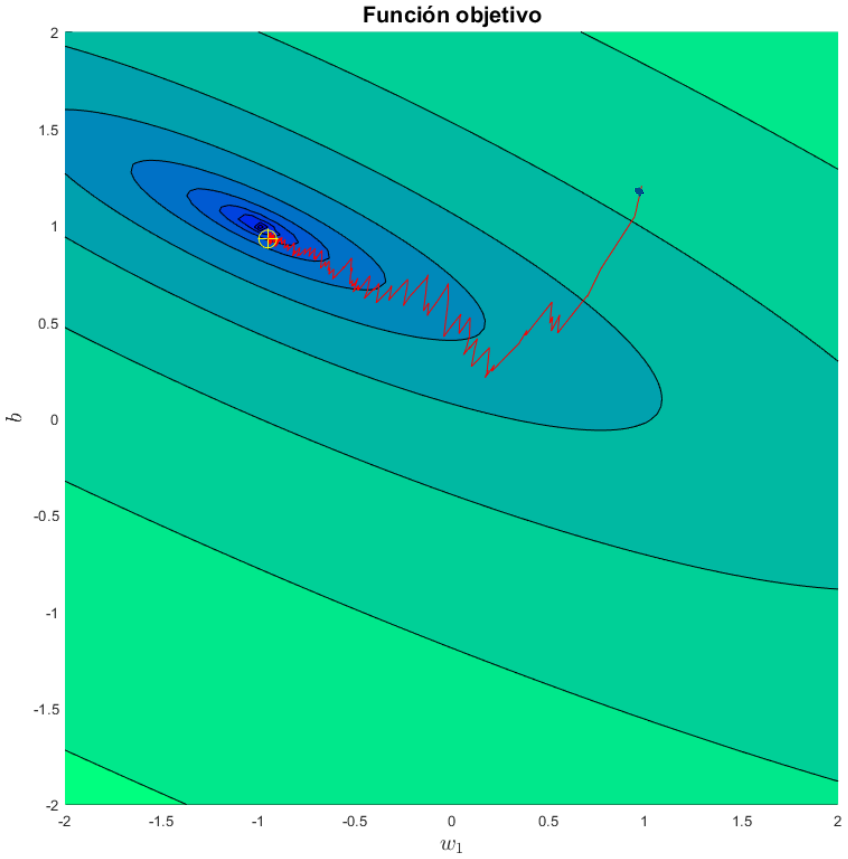
$$X \in \mathbb{R}^{n\times p}$$

$$Y, \hat{Y} \in \mathbb{R}^{1\times p}$$

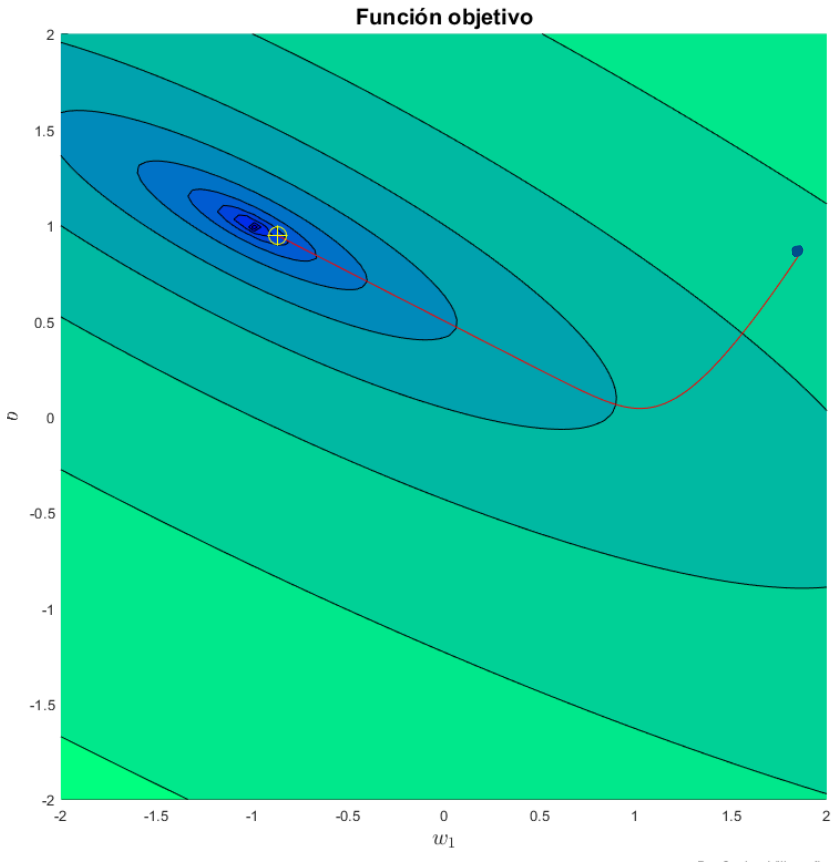
$$b, \eta \in \mathbb{R}$$

Mini-Batch Descending Gradient (mBGD)

SGD

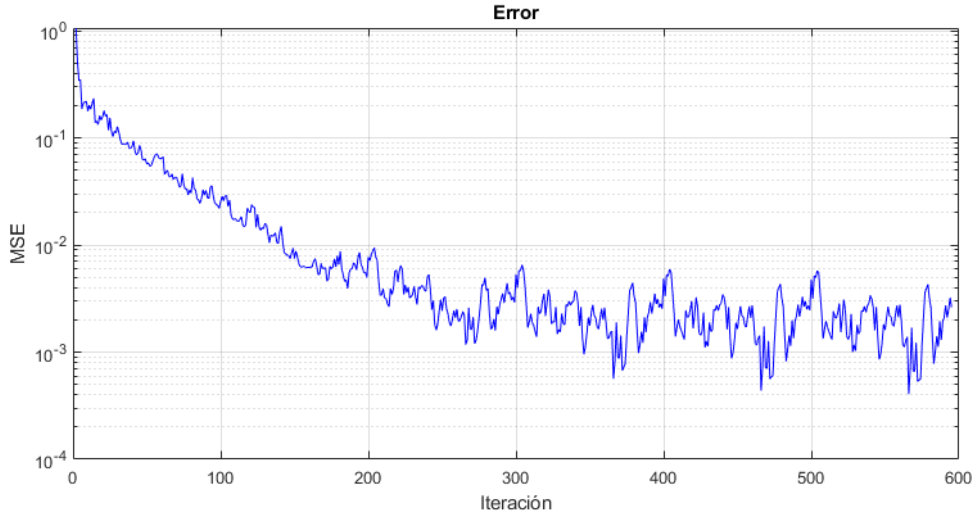


BGD

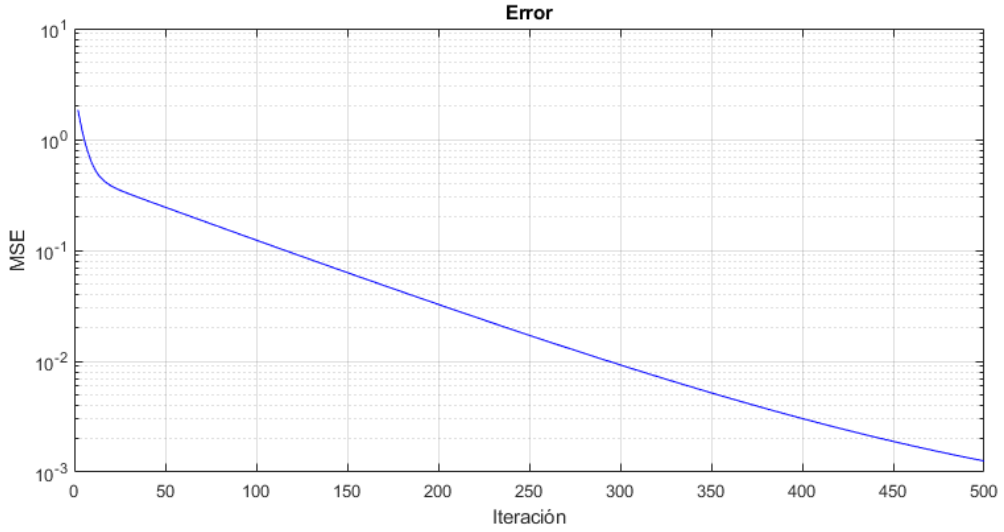


Mini-Batch Descending Gradient (mBGD)

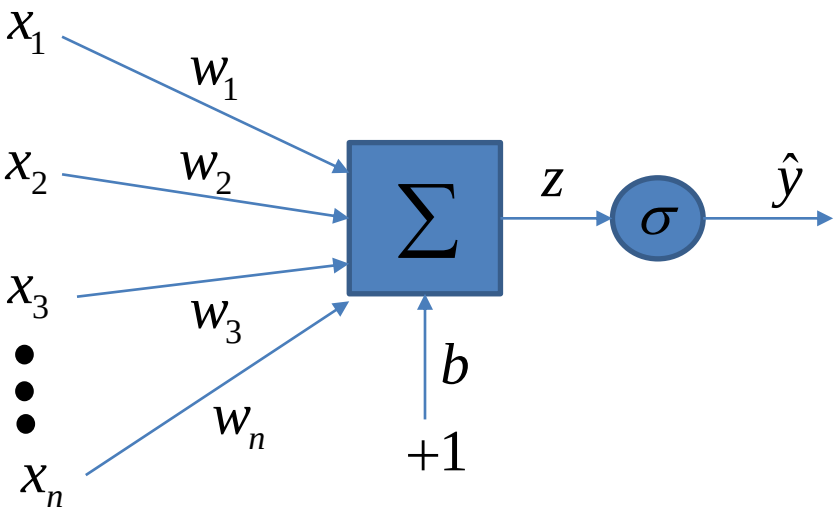
SGD



BGD

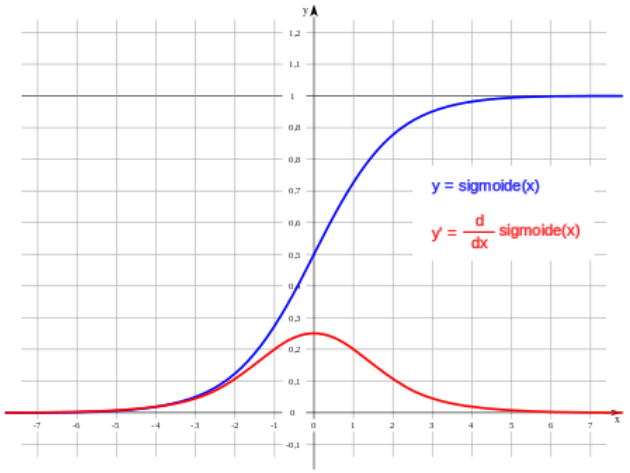


Sigmoid Neuron



$$z = w^T x + b$$

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



Binary Cross Entropy

Optimization Problem

$$\min_{w,b} -\frac{1}{2}\left[y\log(\hat{y}) + (1-y)\log(1-\hat{y}) \right]$$

Learning algorithm

$$w \leftarrow w - \eta \nabla_w L(y,\hat{y})$$

$$b \leftarrow b - \eta \frac{\delta}{\delta b} L(y,\hat{y})$$

Update equations

$$w \leftarrow w + \eta (y - \hat{y}) x$$

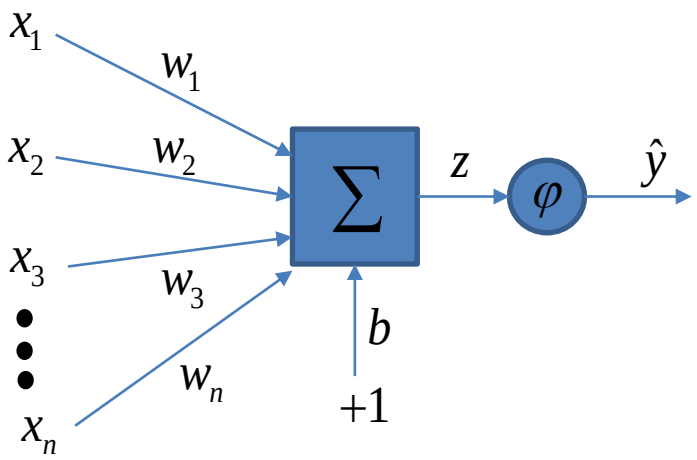
$$b \leftarrow b + \eta (y - \hat{y})$$

$$z = w^T x + b$$

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

Delta Rule

- Suppose $\varphi(z)$ is a differentiable function then



$$\hat{y} = \varphi\left(w^T x + b\right)$$

$$\min_{w,b} \frac{1}{2}\left(y - \hat{y}\right)^2$$

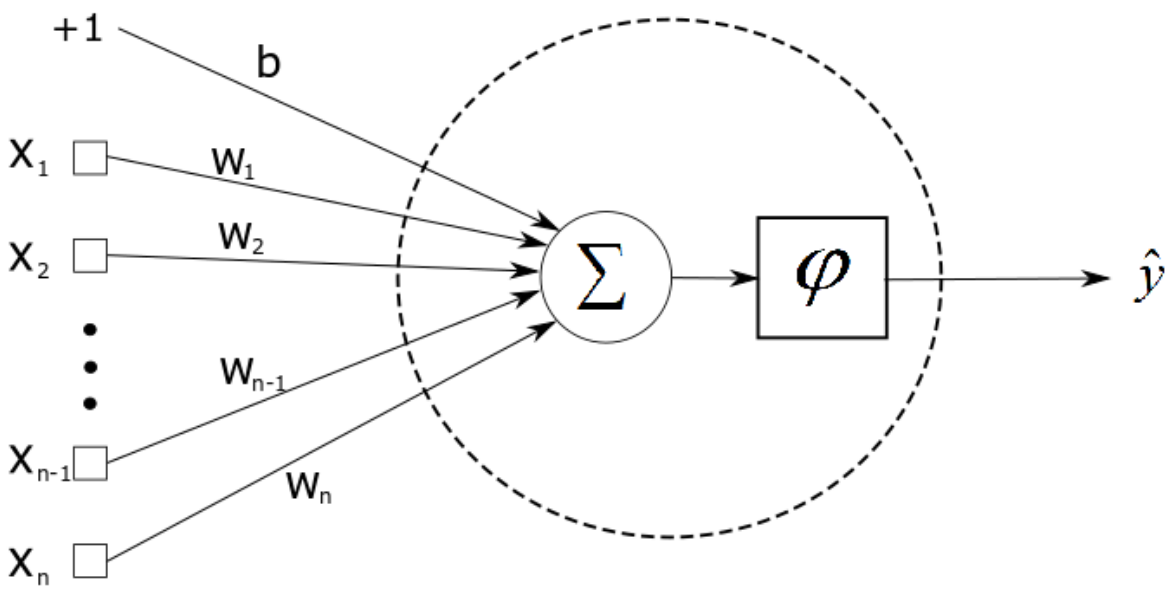
$$w \leftarrow w + \eta\left(y - \hat{y}\right)\frac{d\varphi(z)}{dz}x$$

$$b \leftarrow b + \eta\left(y - \hat{y}\right)\frac{d\varphi(z)}{dz}$$

Delta Rule

Function name	Function	Derivative
Linear or identity	z	1
Sigmoid	$\frac{1}{1+e^{-z}}$	$\hat{y}(1-\hat{y})$
Hyperbolic tangent	$\tanh(z)$	$(1+\hat{y})(1-\hat{y})$
Rectified Linear Unit (ReLU)	$\max(0,z)$	$\begin{cases} 1 & z>0 \\ 0 & z\leq 0 \end{cases}$

One-Layer Network

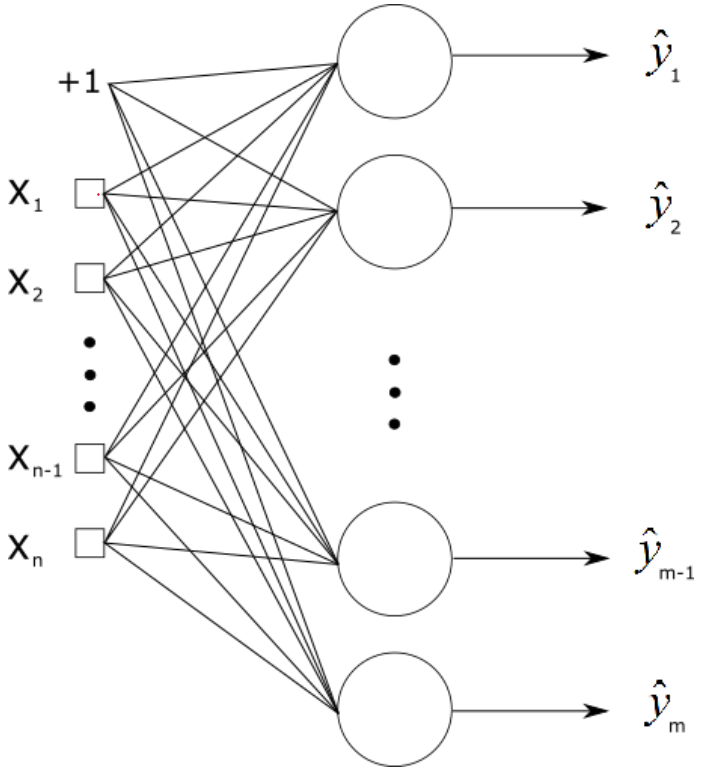


$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n \qquad w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \in \mathbb{R}^n$$

$$b \in \mathbb{R}$$

$$\hat{y} = \varphi\left(w^T x + b\right) = \varphi\left(\begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + b \right)$$

One-Layer Network



$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n \qquad \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} \in \mathbb{R}^m \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \in \mathbb{R}^m$$

$$w = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

One-Layer Network

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} = \varphi\left(\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} \right)$$

$$z = wx + b$$

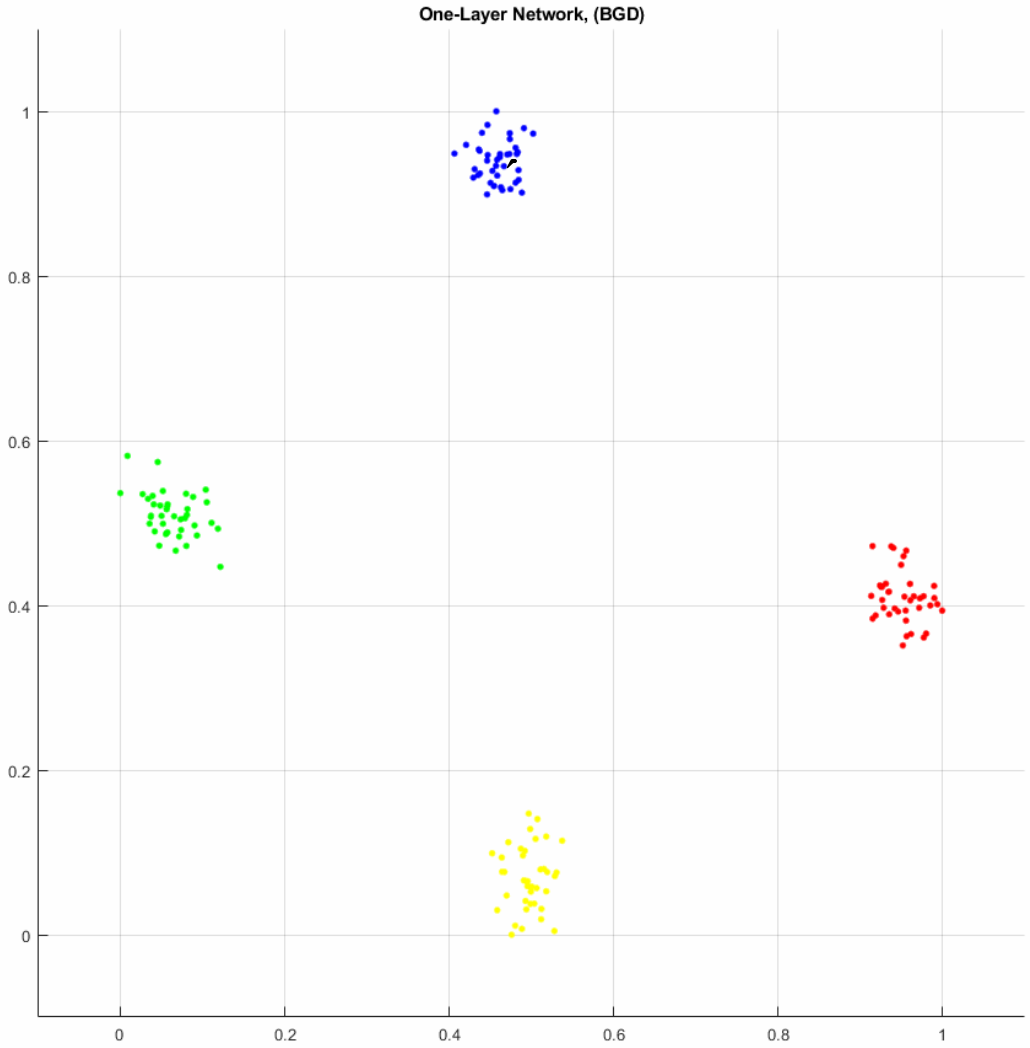
$$y = \varphi(z)$$

One-Layer Network

$$\hat{Y} = \varphi\left(\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(p)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(p)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \cdots & x_n^{(p)} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}\right)$$

$$\hat{Y} = \begin{bmatrix} \hat{y}_1^{(1)} & \hat{y}_1^{(2)} & \cdots & \hat{y}_1^{(p)} \\ \hat{y}_2^{(1)} & \hat{y}_2^{(2)} & \cdots & \hat{y}_2^{(p)} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_m^{(1)} & \hat{y}_m^{(2)} & \cdots & \hat{y}_m^{(p)} \end{bmatrix}$$

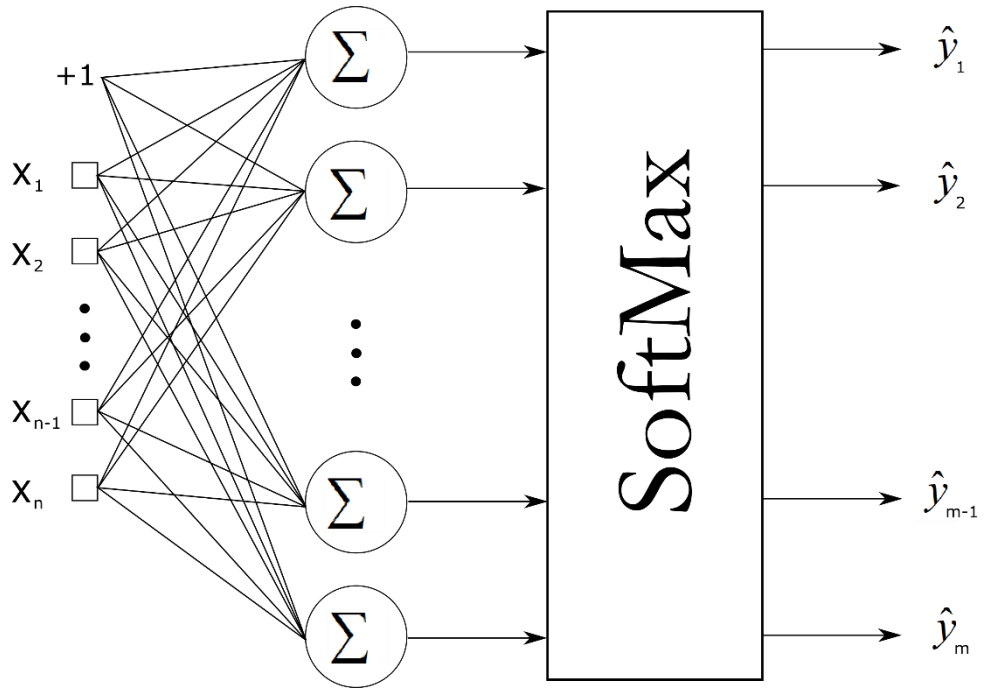
One vs All



SoftMax (Categorical Cross Entropy)

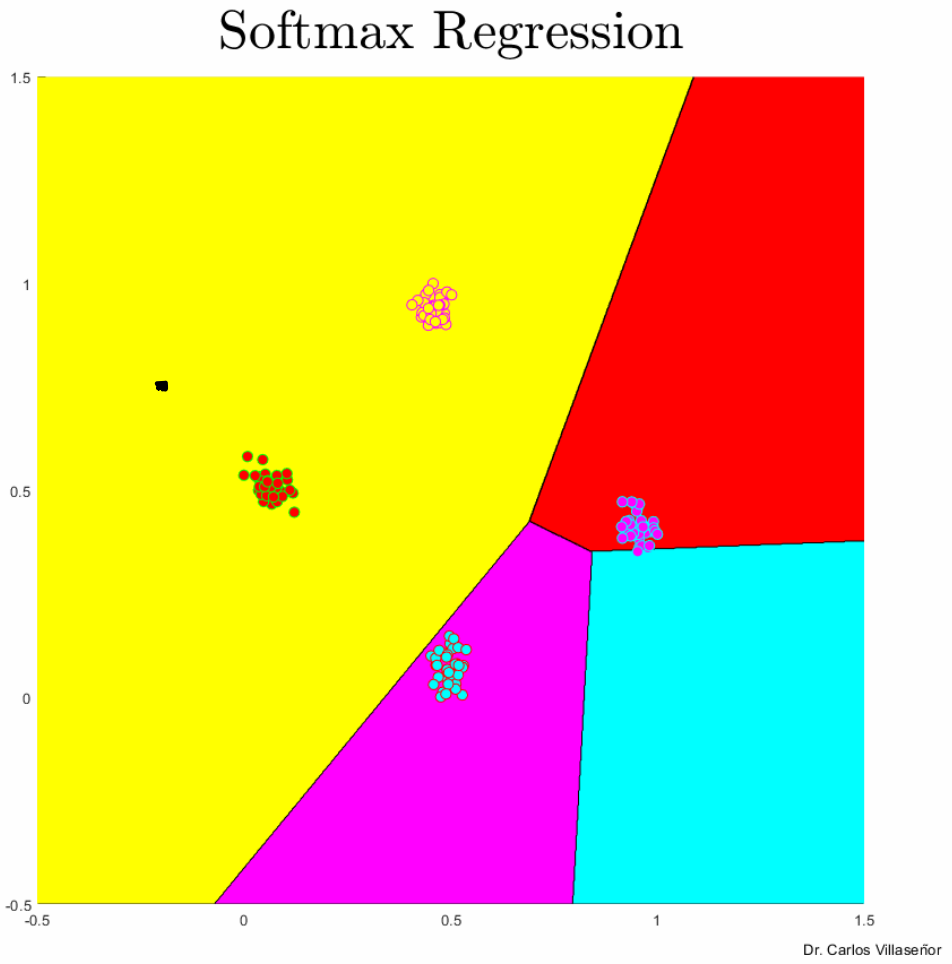
$$E = -\sum_{j=1}^{m} y_j \log\left(\hat{y}_j\right)$$

$$E = -\frac{1}{p}\sum_{i=1}^{p}\sum_{j=1}^{m} y_j^{(i)} \log\left(\hat{y}_j^{(i)}\right)$$

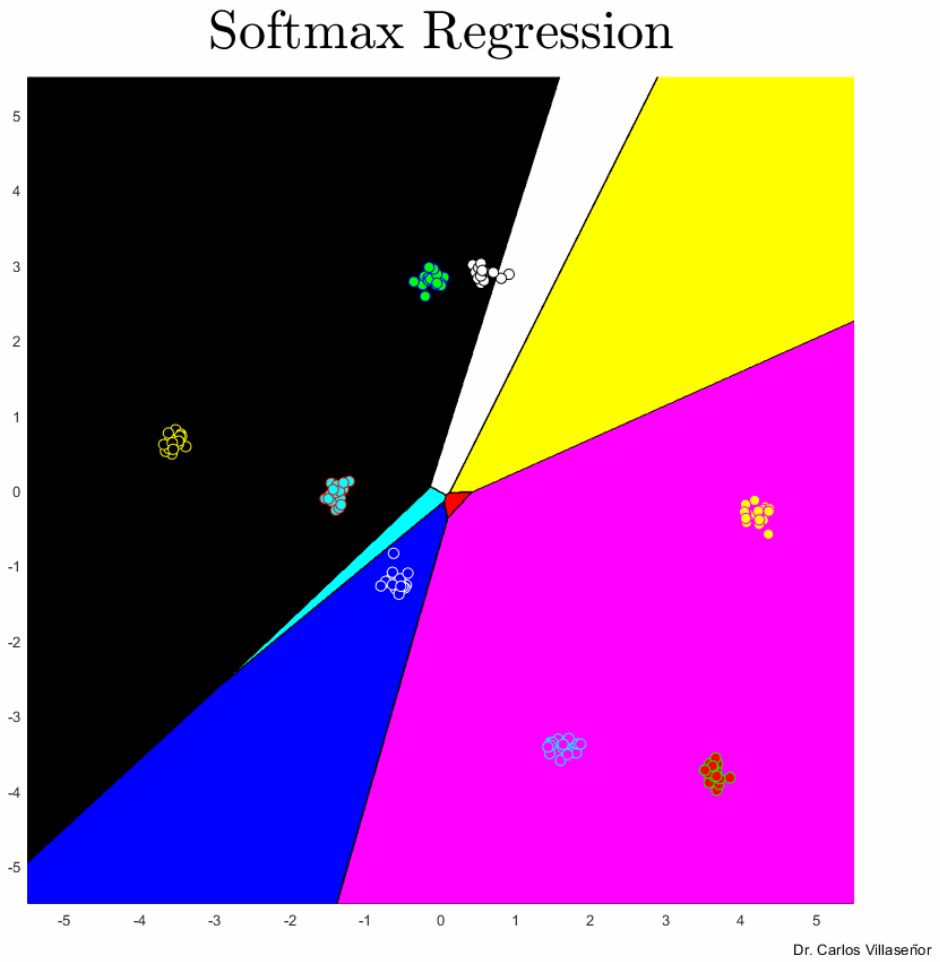


$$S(z)_j = \frac{e^{z_j}}{\sum_{i=1}^{m} e^{z_i}}$$

Softmax Layer



Softmax Layer

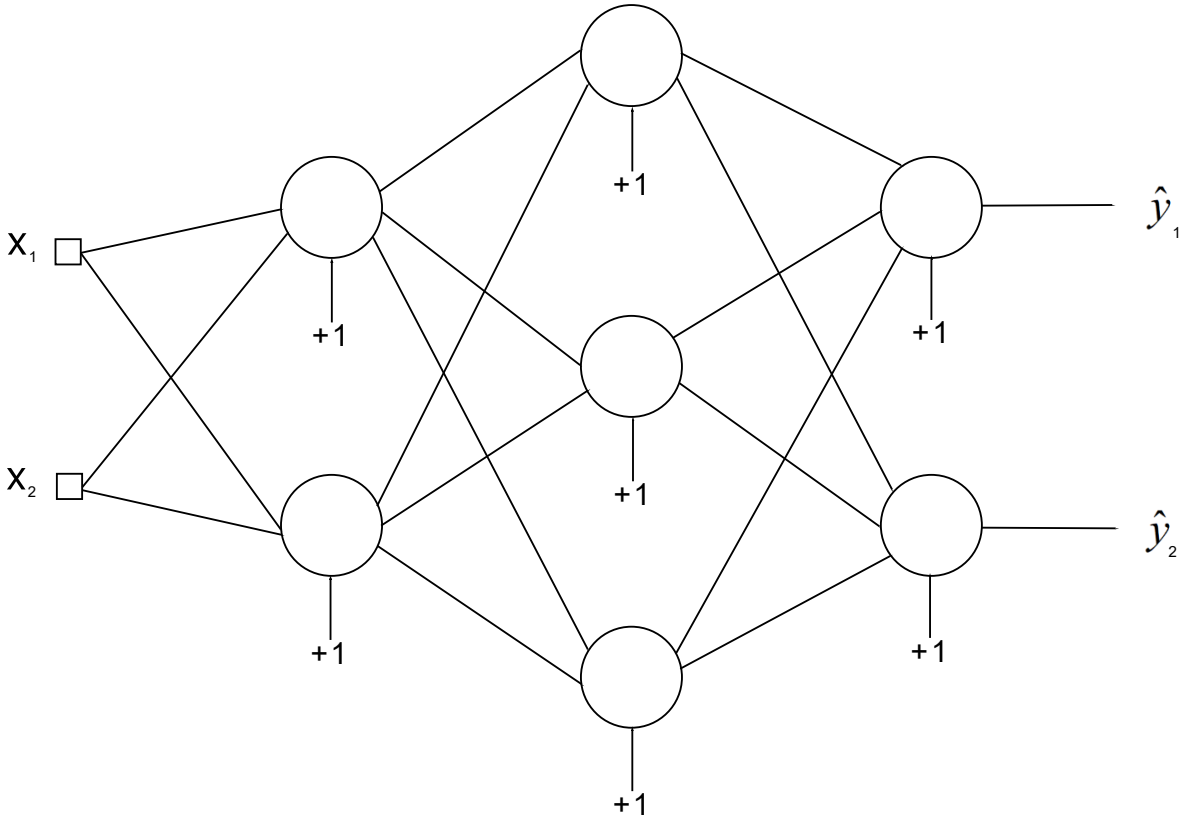


Summary

Problem	Activation function	Loss Function
Regression	Linear	Mean square error(MSE)
Multi-label classification	Sigmoid	Binary Cross Entropy Error(BCE)
One-winner qualification	SoftMax	Categorical cross-entropy error(CCE)

Dense Network

Enumeration of layers and neurons:



$L =$

$n_1 =$

$n_0 =$

$n_L =$

n_l : Number of neurons or inputs in the layer number "l"

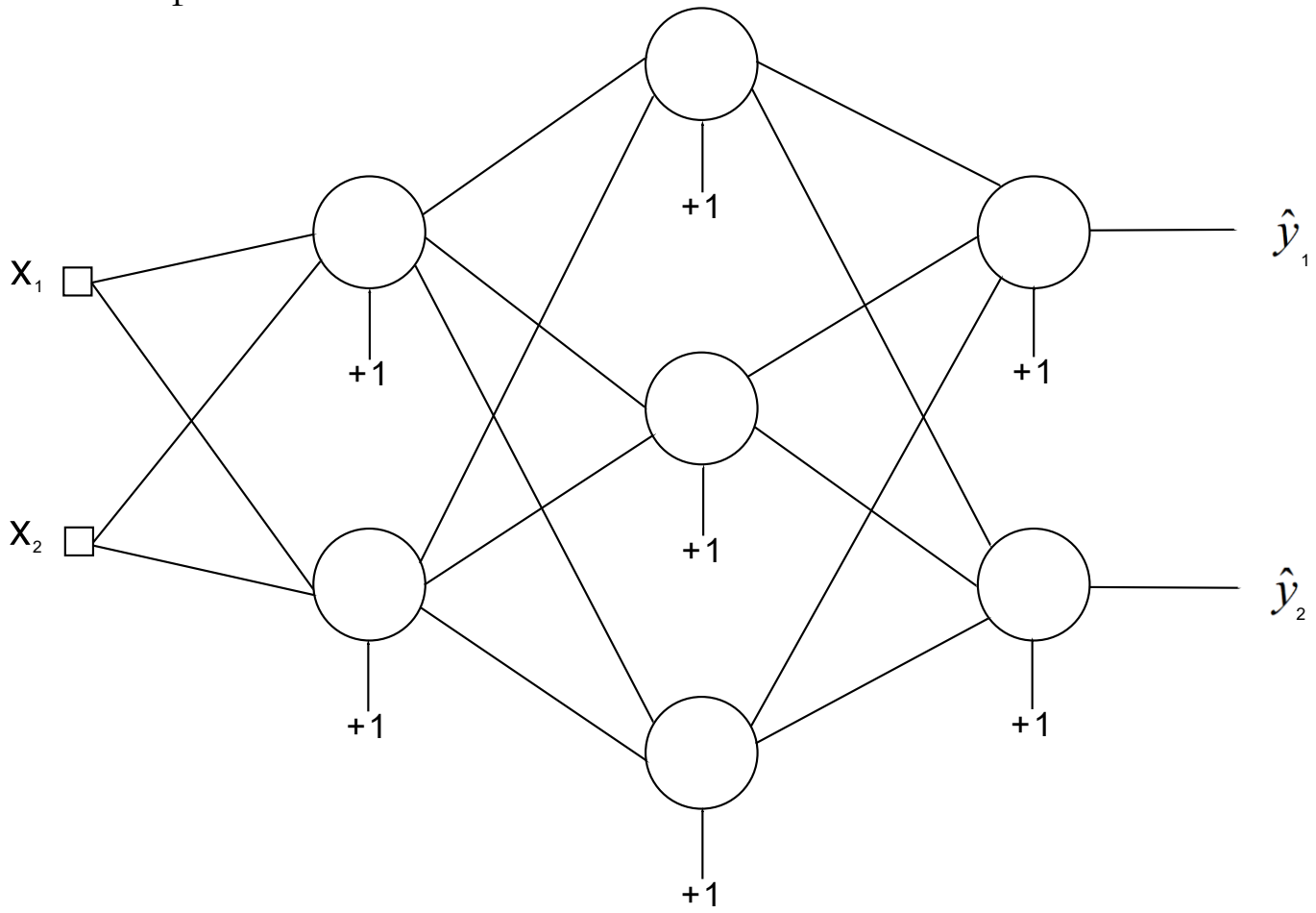
L : Last Layer

Dense Network

$$w^{[l]} = \begin{bmatrix} w_{1,1}^{[l]} & w_{1,2}^{[l]} & \cdots & w_{1,n_{l-1}}^{[l]} \\ w_{2,1}^{[l]} & w_{2,2}^{[l]} & \cdots & w_{2,n_{l-1}}^{[l]} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_l,1}^{[l]} & w_{n_l,2}^{[l]} & \cdots & w_{n_l,n_{l-1}}^{[l]} \end{bmatrix} \in \mathbb{R}^{n_l \times n_{l-1}} \qquad b^{[l]} = \begin{bmatrix} b_1^{[l]} \\ b_2^{[l]} \\ \vdots \\ b_{n_l}^{[l]} \end{bmatrix} \in \mathbb{R}^{n_l}$$

Dense Network

Inputs and outputs:



$a_2^{[0]}$

$a_2^{[1]}$

$a_3^{[2]}$

$a_1^{[3]}$

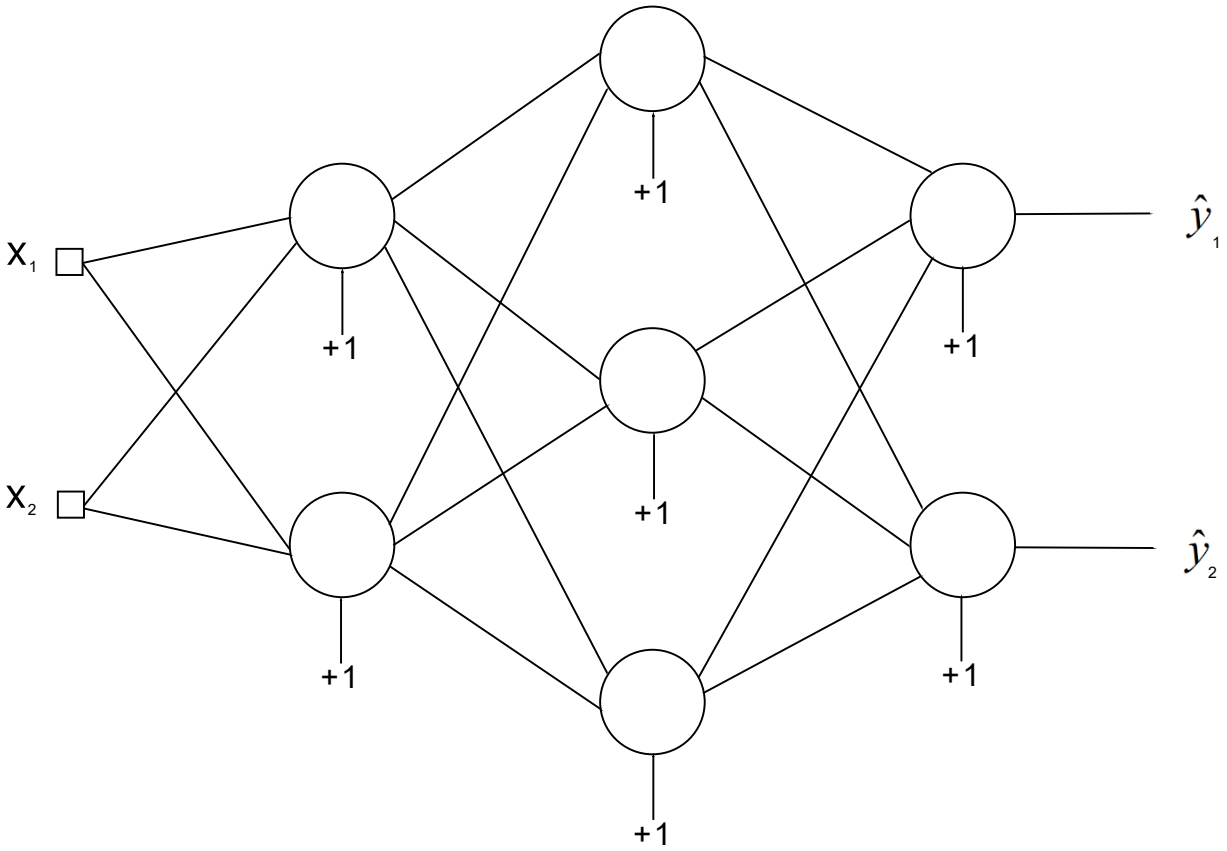
$a_i^{[l]}$: Output of the "l" layer of the "i" neuron

$$a_i^{[L]} = \hat{y}_i$$

$$a_i^{[0]} = x_i$$

$$a^{[l]} = \begin{bmatrix} a_1^{[l]} \\ a_2^{[l]} \\ \vdots \\ a_{n_l}^{[l]} \end{bmatrix} \in \mathbb{R}^{n_l}$$

Propagation



$$a^{[0]} = x$$

$$z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$$

$$a^{[1]} = \varphi^{[1]}\left(z^{[1]}\right)$$

$$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[2]} = \varphi^{[2]}\left(z^{[2]}\right)$$

$$z^{[3]} = w^{[3]} a^{[2]} + b^{[3]}$$

$$a^{[3]} = \varphi^{[3]}\left(z^{[3]}\right)$$

$$\hat{y} = a^{[3]}$$

Propagation

$$a^{[0]} = x$$

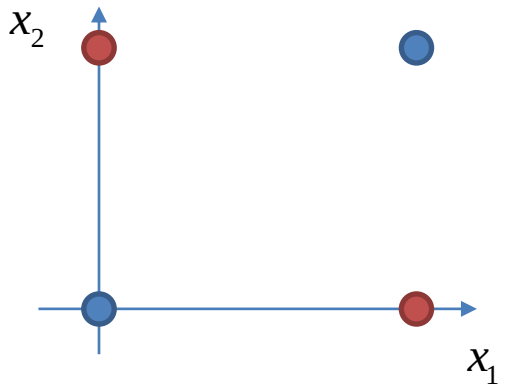
$$\text{foreach } l \in \{1, \cdots, L\}$$

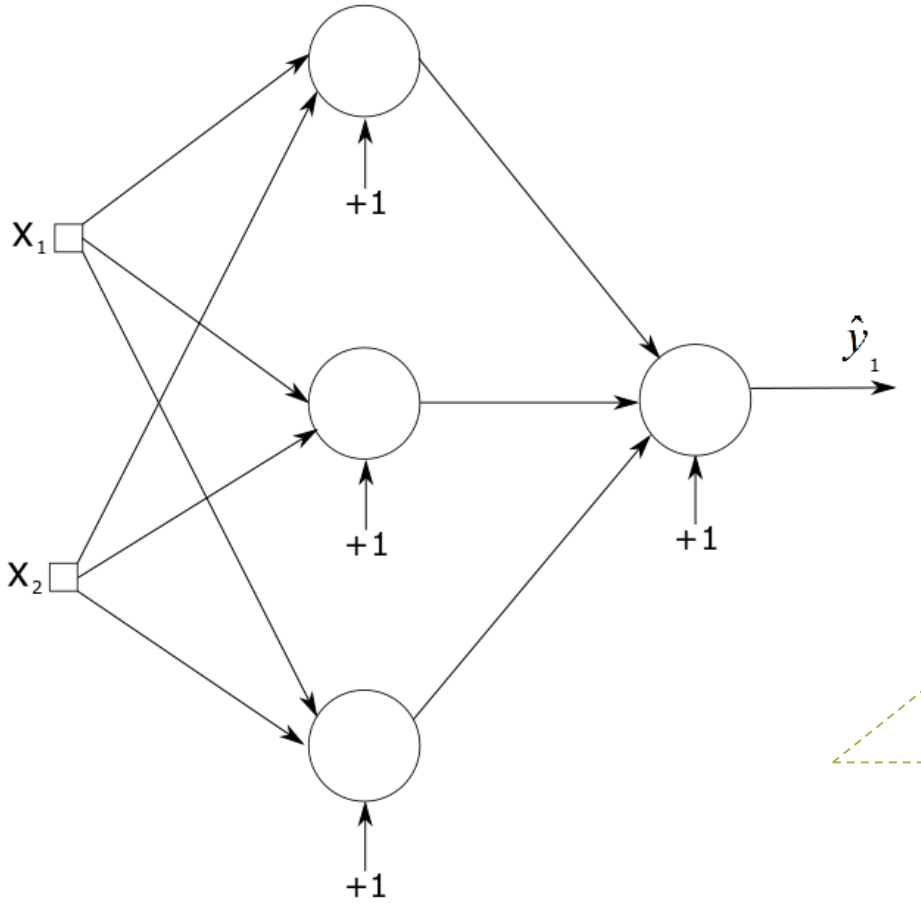
$$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$$

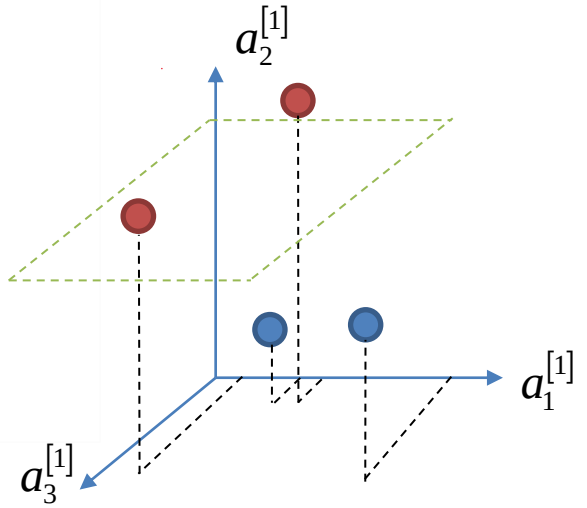
$$\mathrm{a}^{[l]} = \varphi^{[l]}\left(z^{[l]}\right)$$

$$\hat{y} = a^{[L]}$$

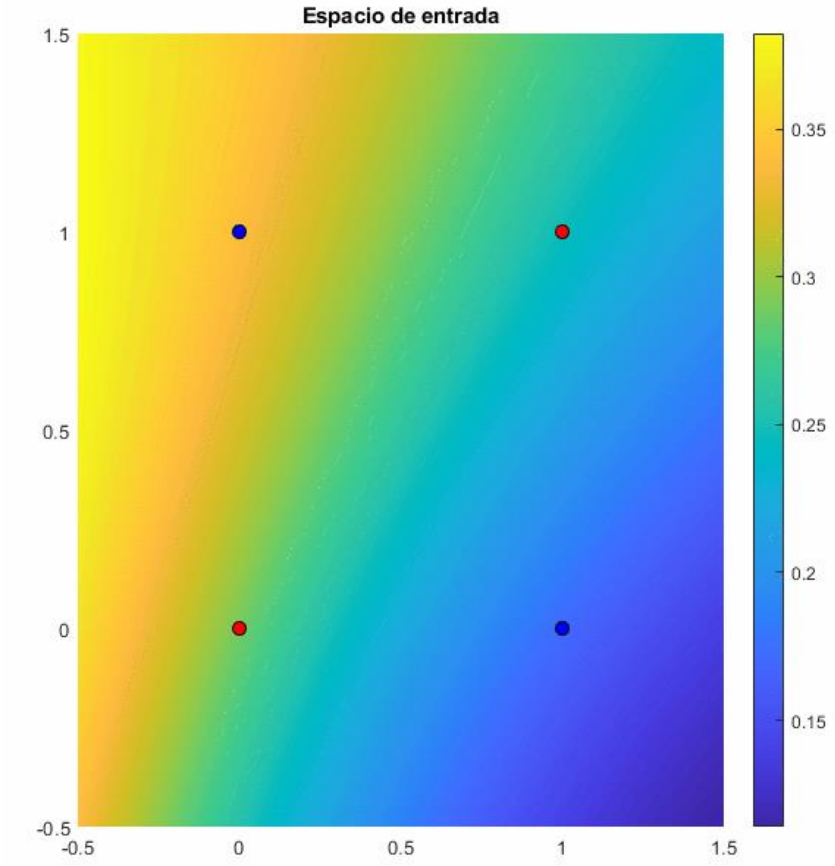
Geometric Representation

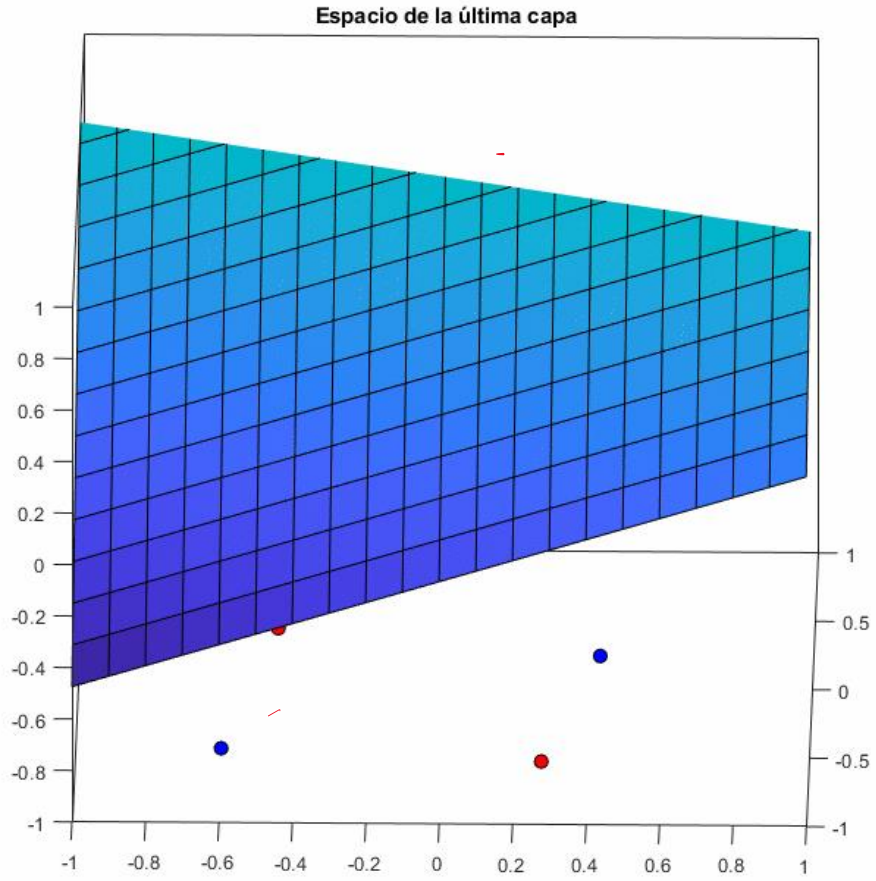




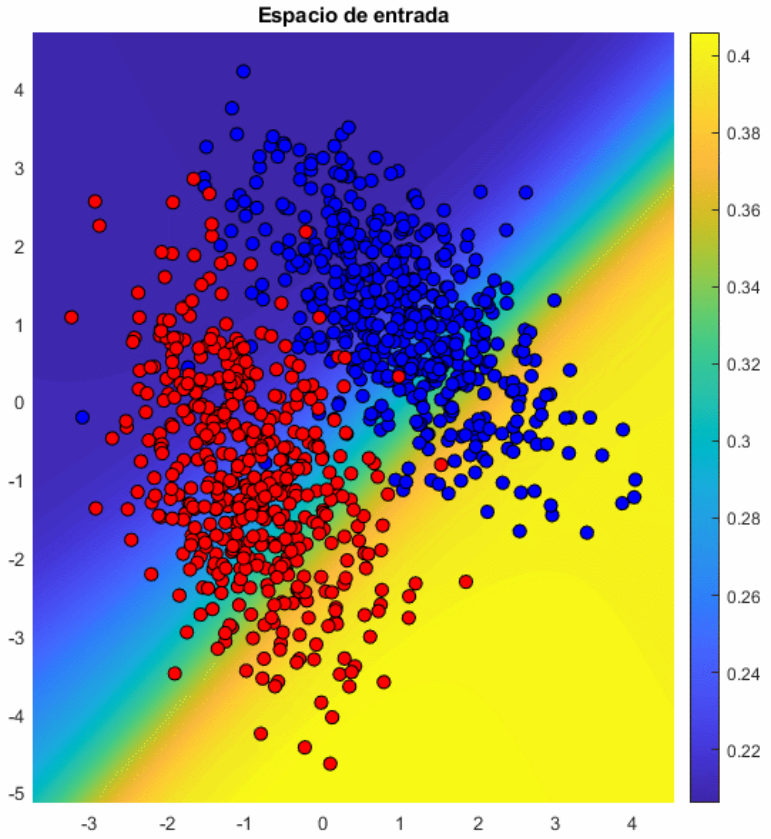


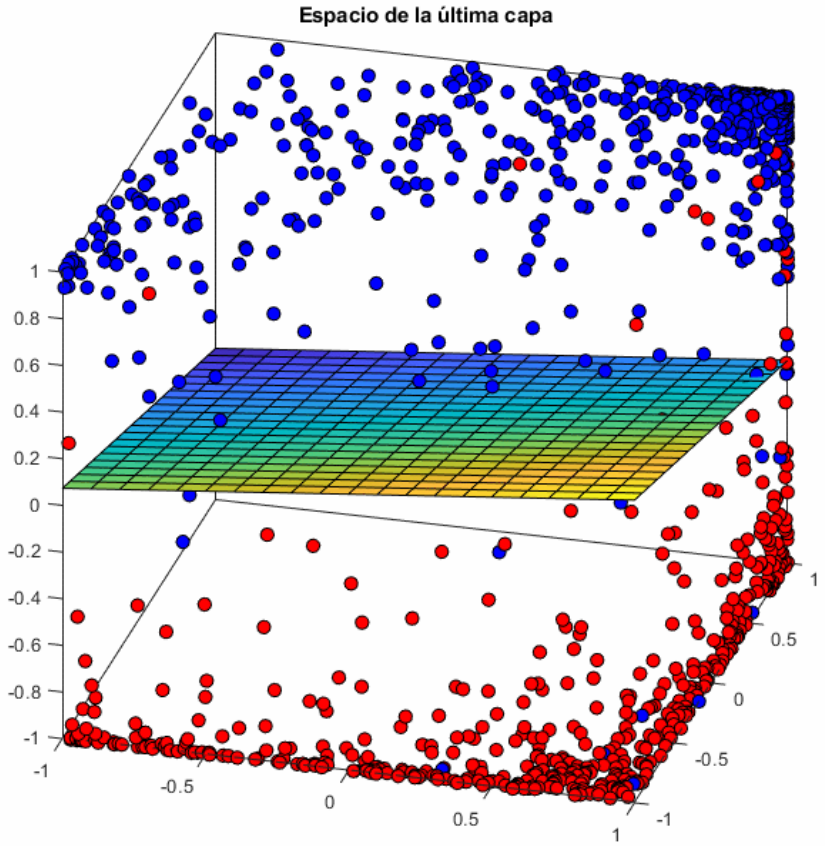
Geometric Representation



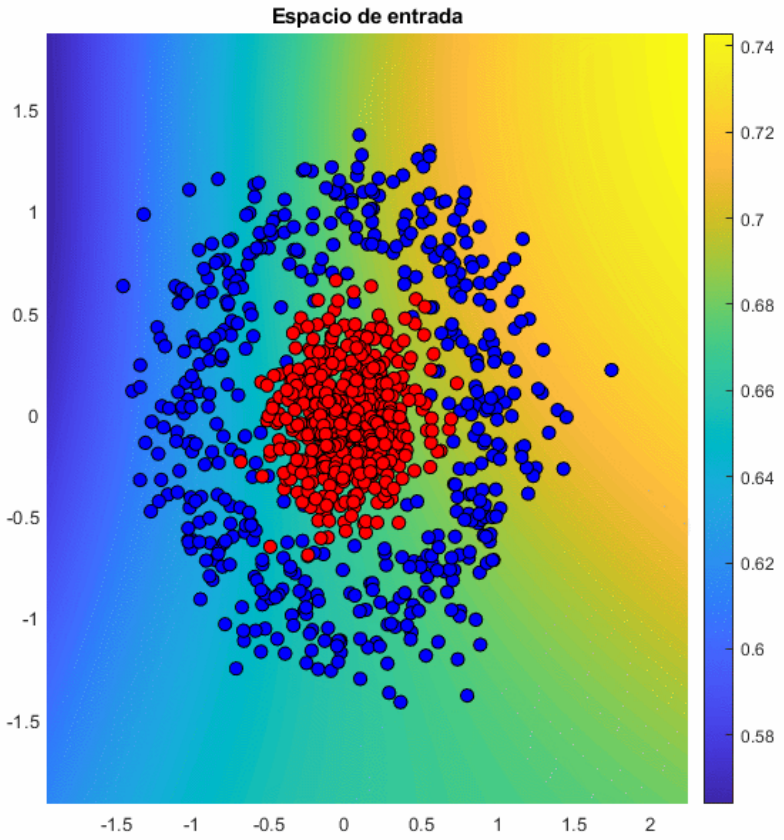


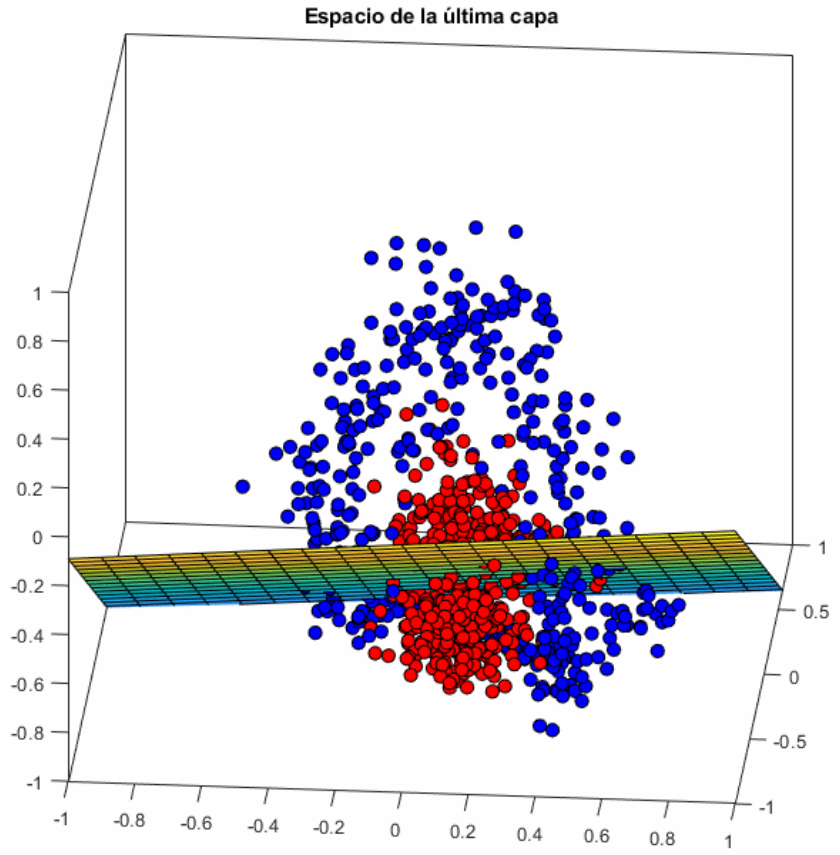
Geometric Representation



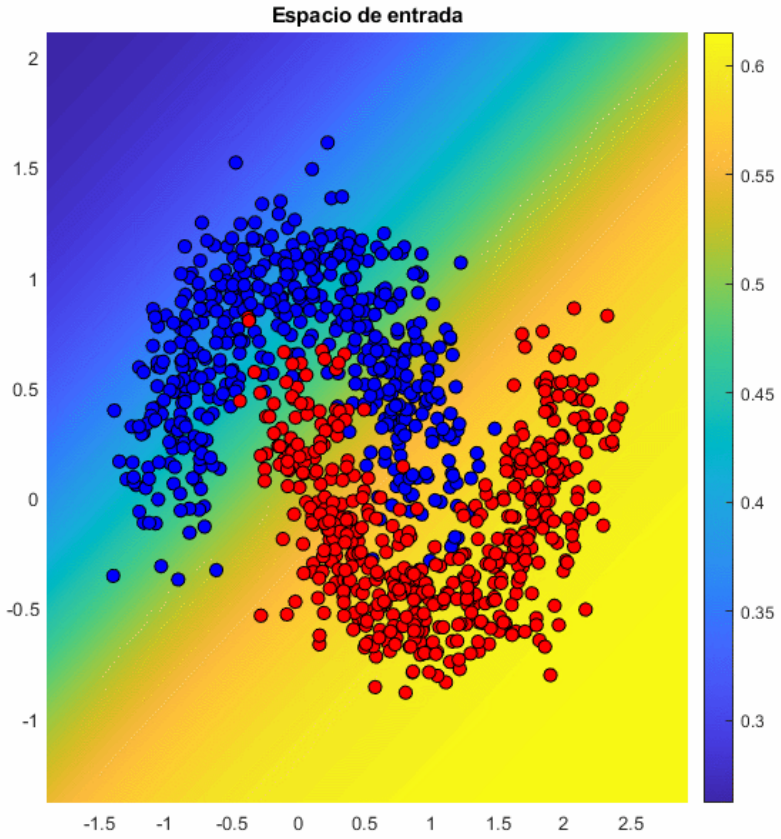


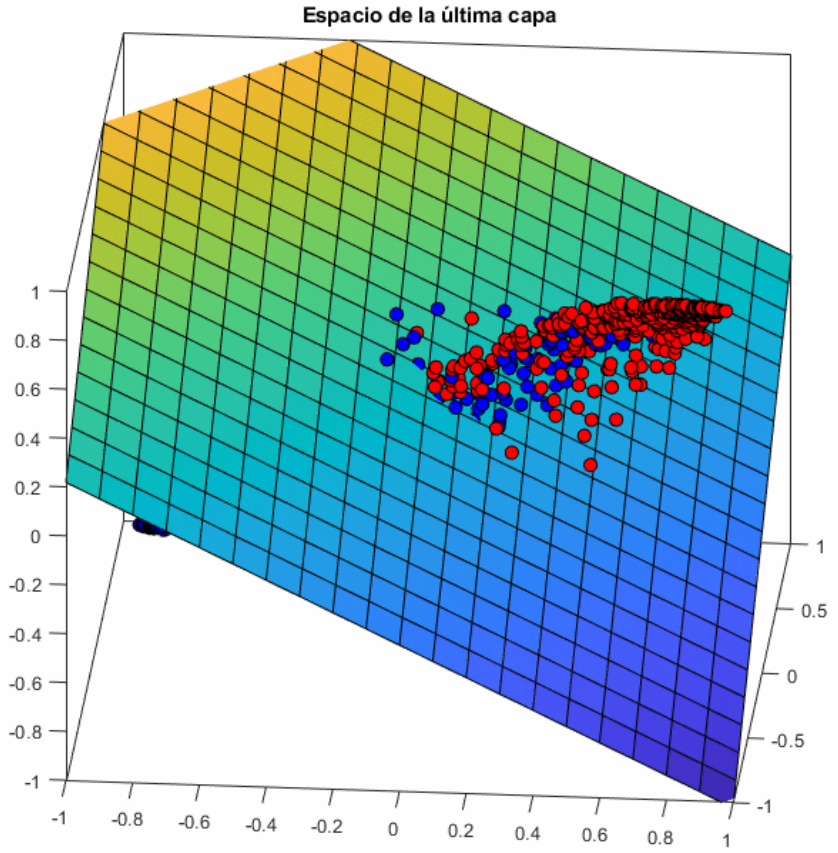
Geometric Representation





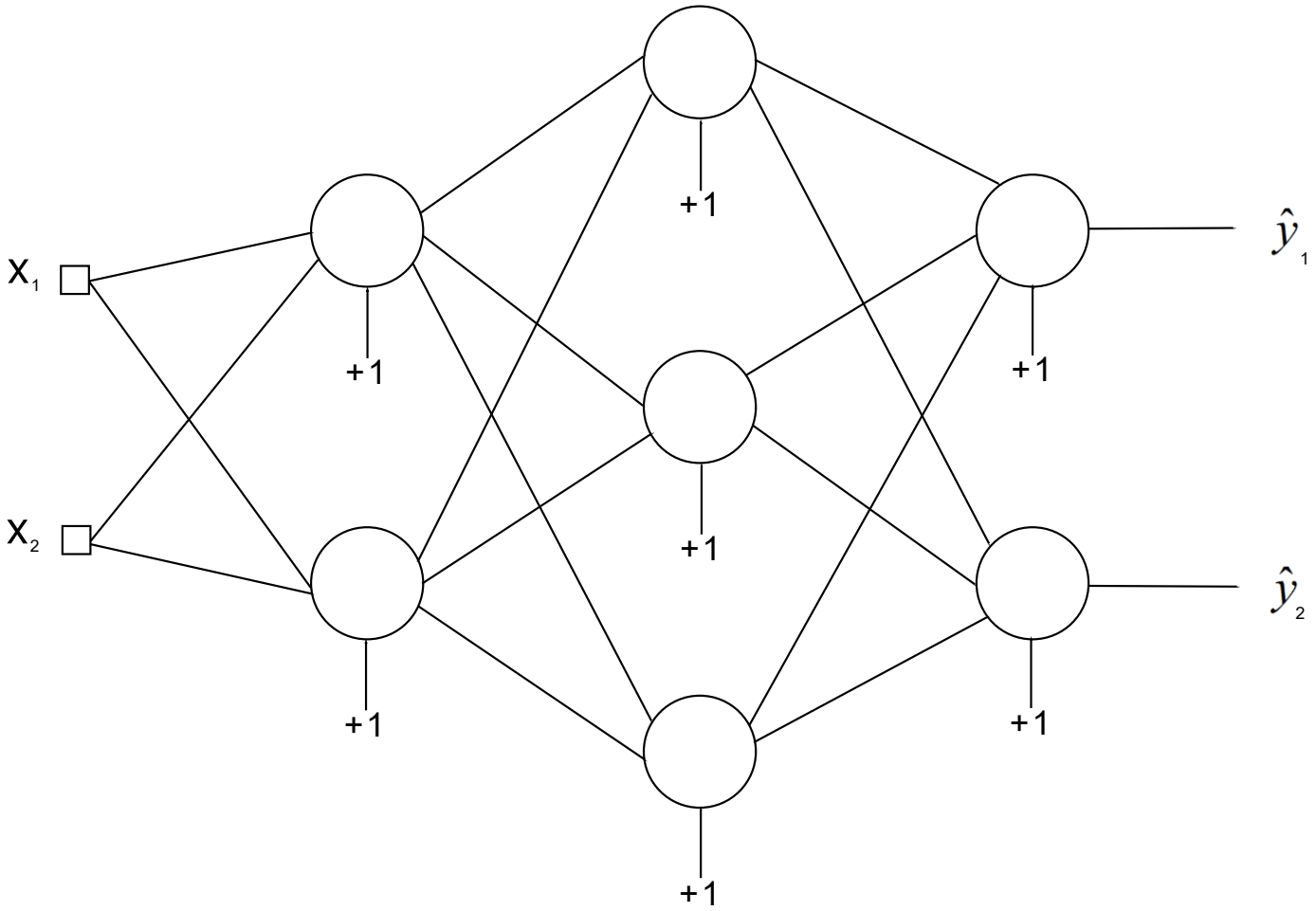
Geometric Representation





Backpropagation

$$da_i^{[L]} = \frac{d\varphi_i^{[L]}}{dz}$$



$\delta_i^{[l]}$: The local gradient of the "i" neuron in the "l" layer

$$\delta_i^{[L]} = \left(y_i - a_i^{[L]}\right) da_i^{[L]}$$

Backpropagation

- Let's take a specific example:

$$\delta_1^{[2]} = \left[w_{1,1}^{[3]} \delta_1^{[3]} + w_{2,1}^{[3]} \delta_2^{[3]} \right] da_1^{[2]}$$

$$\delta_2^{[1]} = \left[w_{1,2}^{[2]} \delta_1^{[2]} + w_{2,2}^{[2]} \delta_2^{[2]} + w_{3,2}^{[2]} \delta_3^{[2]} \right] da_2^{[1]}$$

Backpropagation

- We finally come to these equations

$$\delta^{[l]} = \begin{cases} \left(y - a^{[l]}\right) \odot da^{[l]} & \text{Si} \quad l = L \\ \left[w^{T[l+1]} \delta^{[l+1]}\right] \odot da^{[l]} & \text{Si} \quad 0 < l < L \end{cases}$$

$$w^{[l]} \leftarrow w^{[l]} + \eta \delta^{[l]} a^{T[l-1]}$$

$$b^{[l]} \leftarrow b^{[l]} + \eta \delta^{[l]}$$

Training Algorithm

$$a^{[0]} = x$$

foreach $l \in \{1, \cdots, L\}$

$$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$$

$$a^{[l]}, da^{[l]} = \varphi^{[l]}\left(z^{[l]}\right)$$

foreach $l \in \{L, \cdots, 1\}$

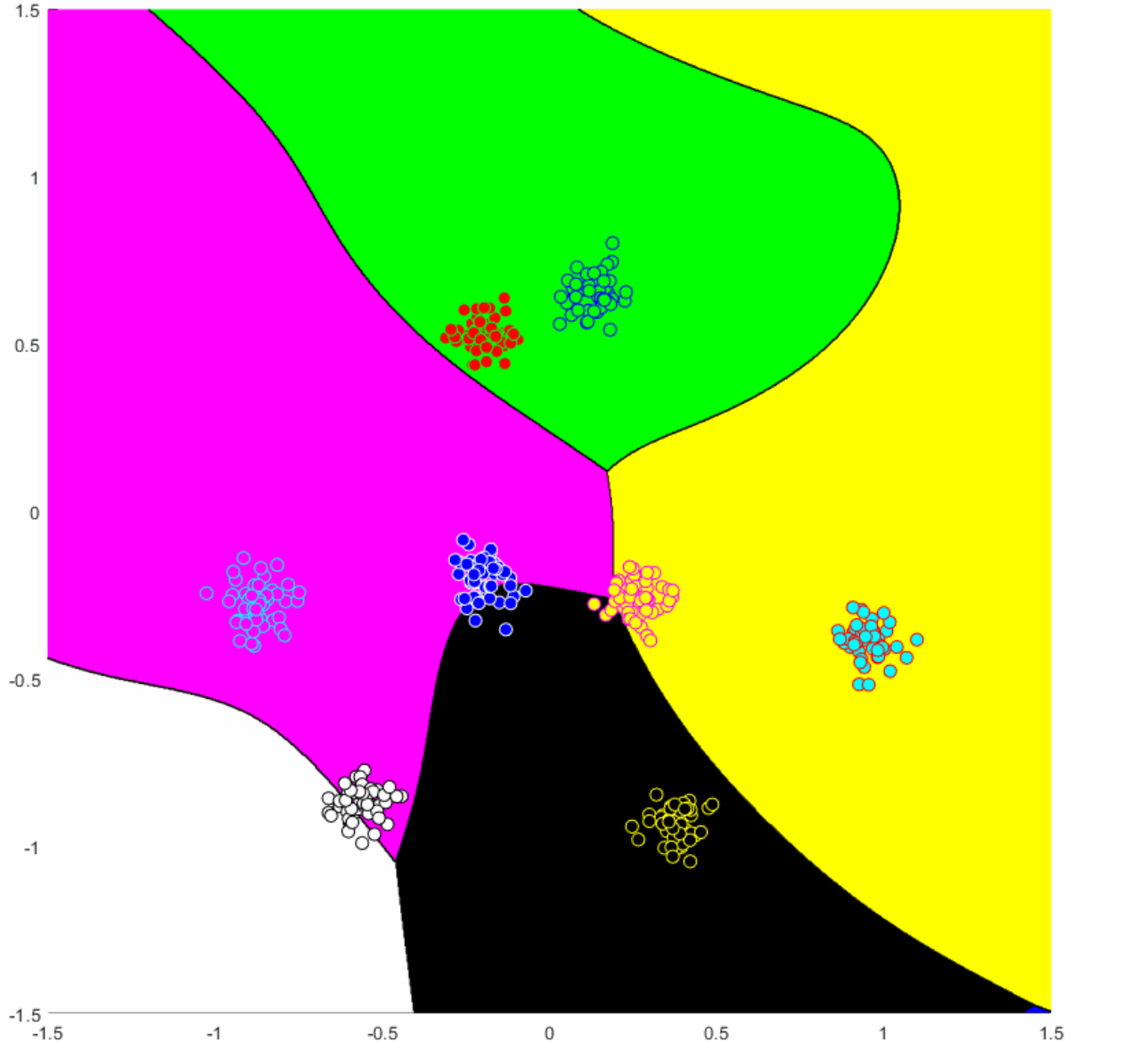
$$\delta^{[l]} = \begin{cases} \left(y - a^{[l]}\right) \odot da^{[l]} & \text{Si} \quad l = L \\ \left[w^{T[l+1]} \delta^{[l+1]}\right] \odot da^{[l]} & \text{Si} \quad 0 < l < L \end{cases}$$

foreach $l \in \{1, \cdots, L\}$

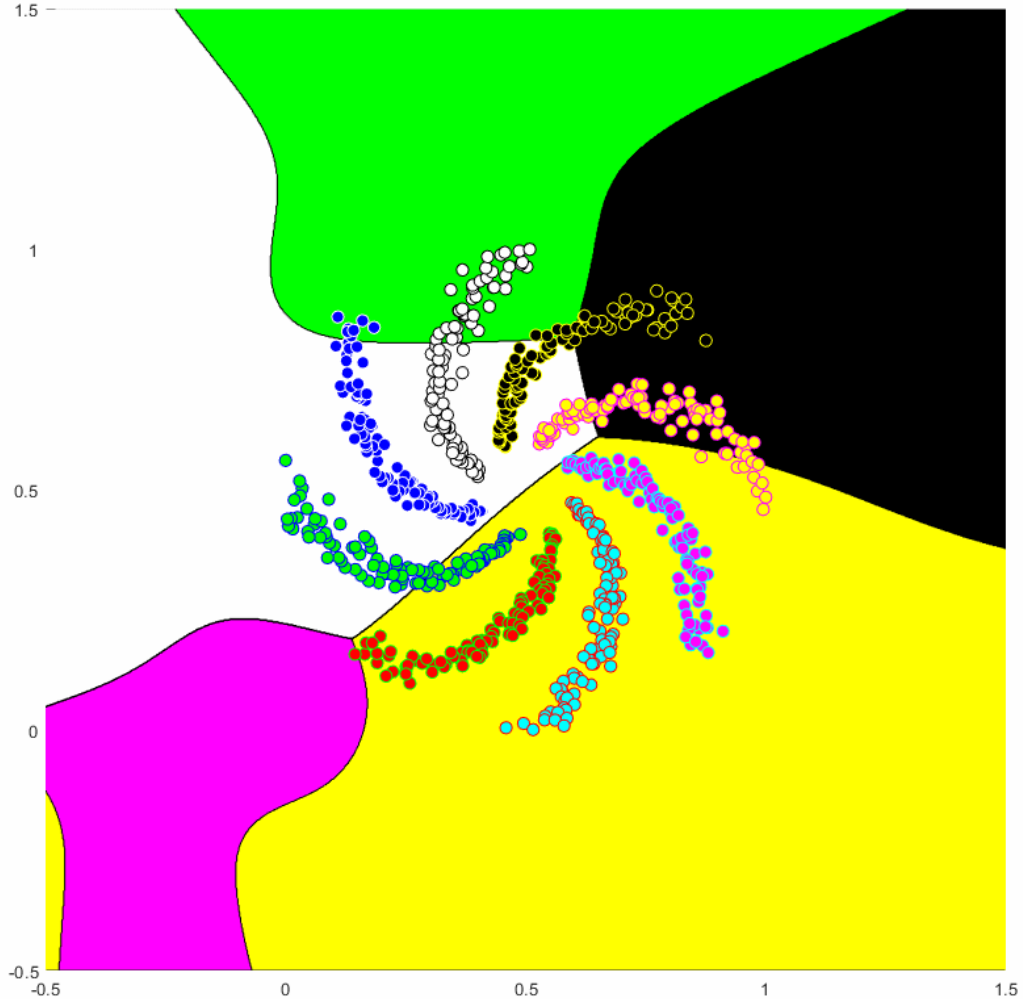
$$w^{[l]} \leftarrow w^{[l]} + \eta \delta^{[l]} a^{T[l-1]}$$

$$b^{[l]} \leftarrow b^{[l]} + \eta \delta^{[l]}$$

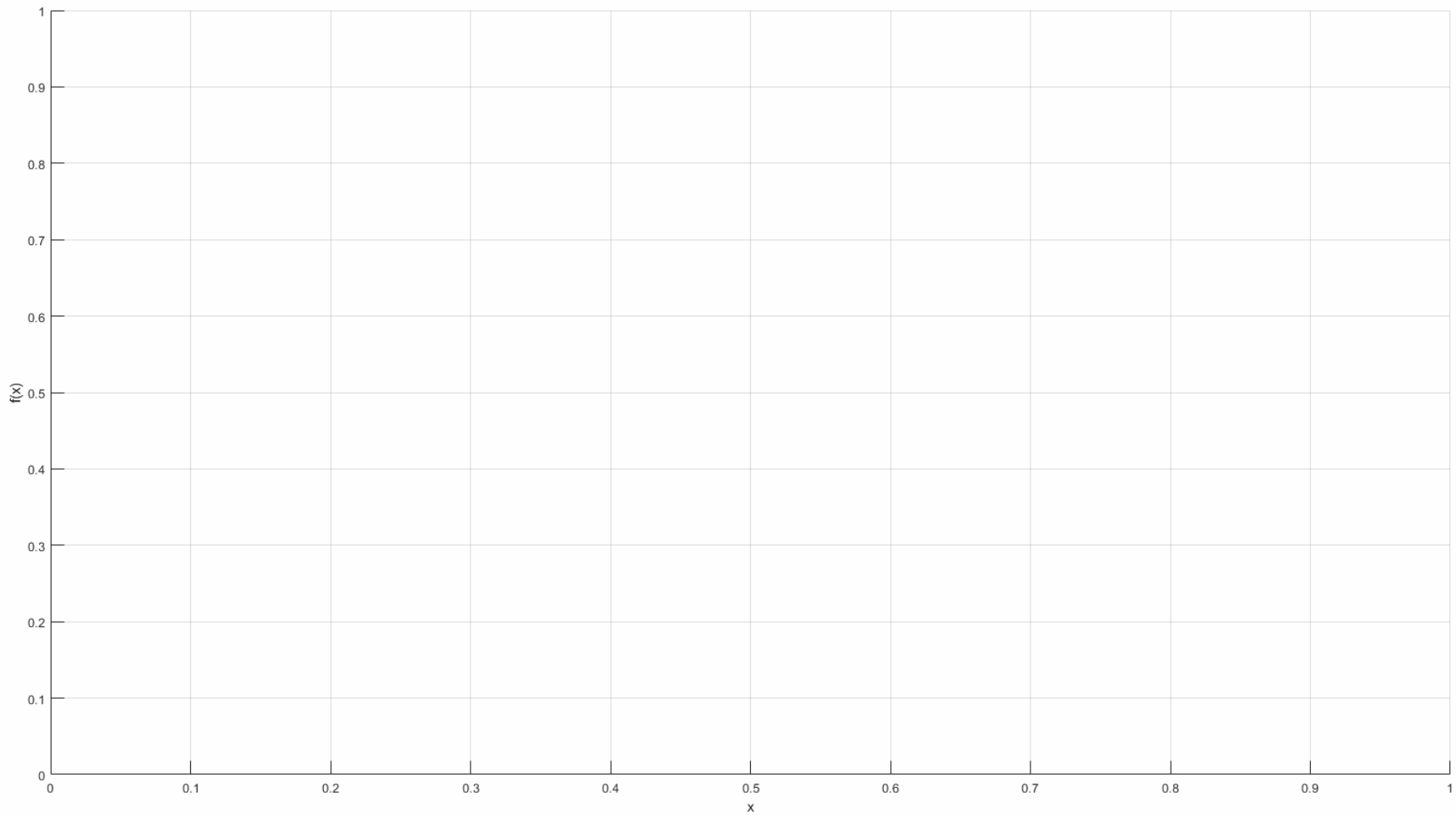
MLP with Softmax

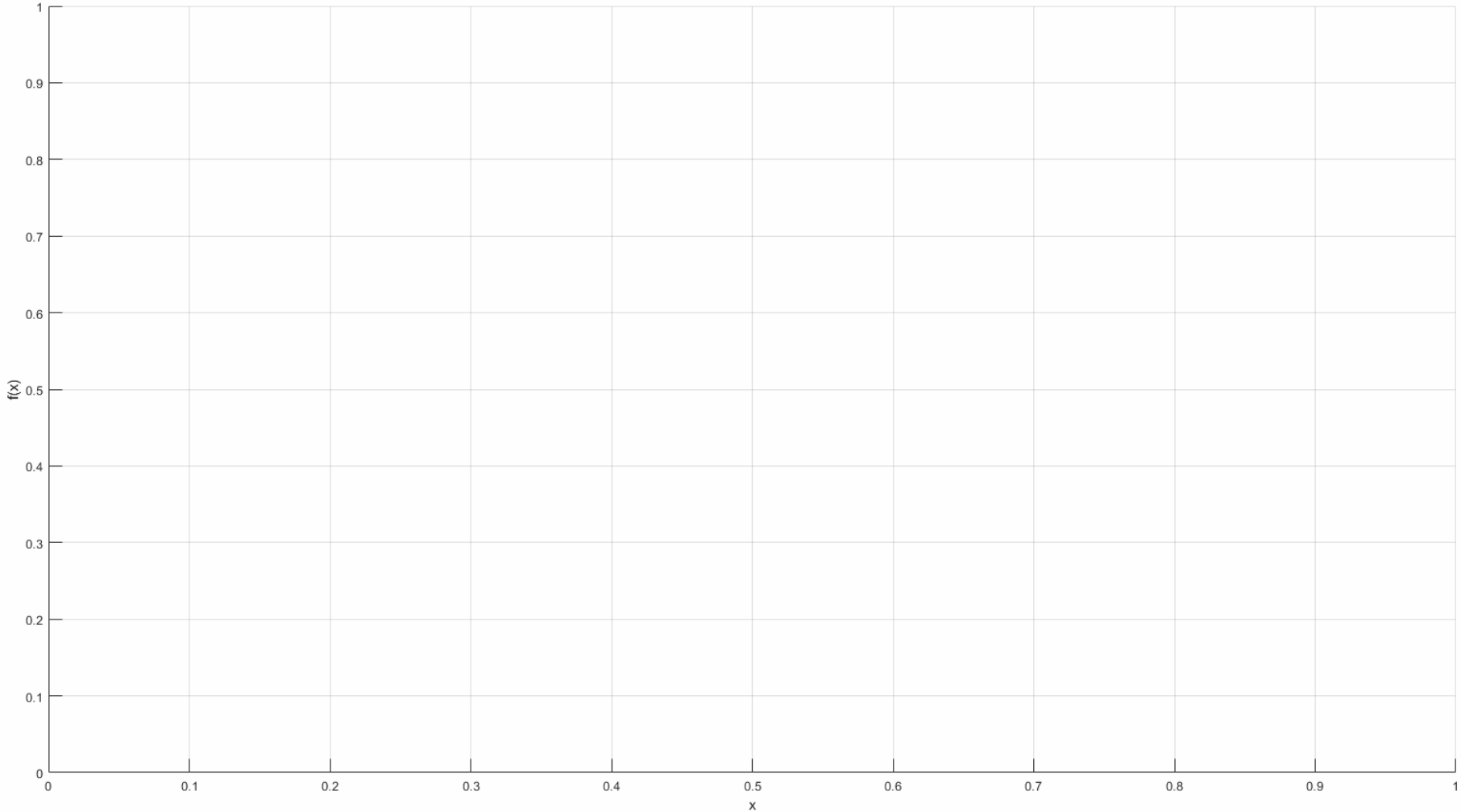


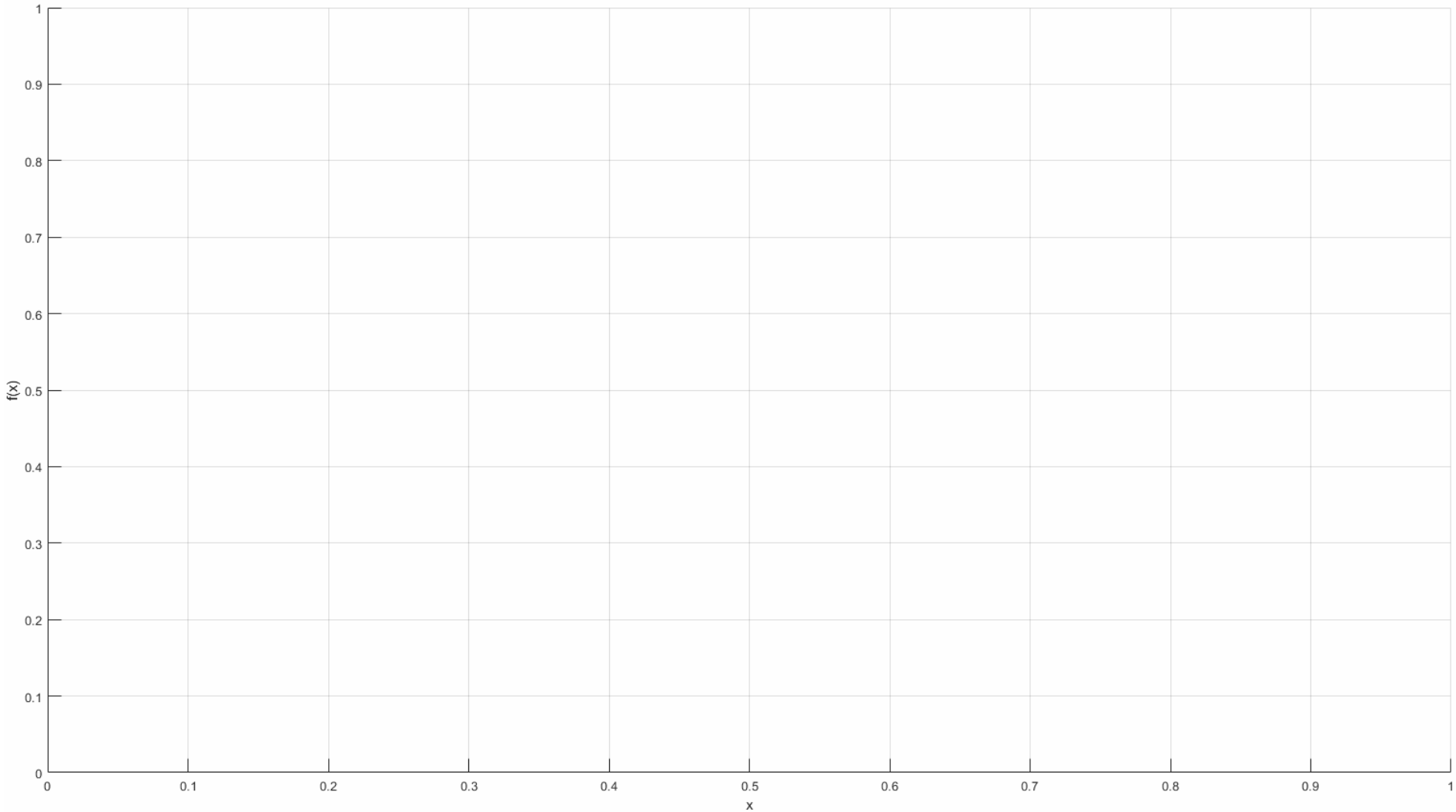
MLP with Softmax



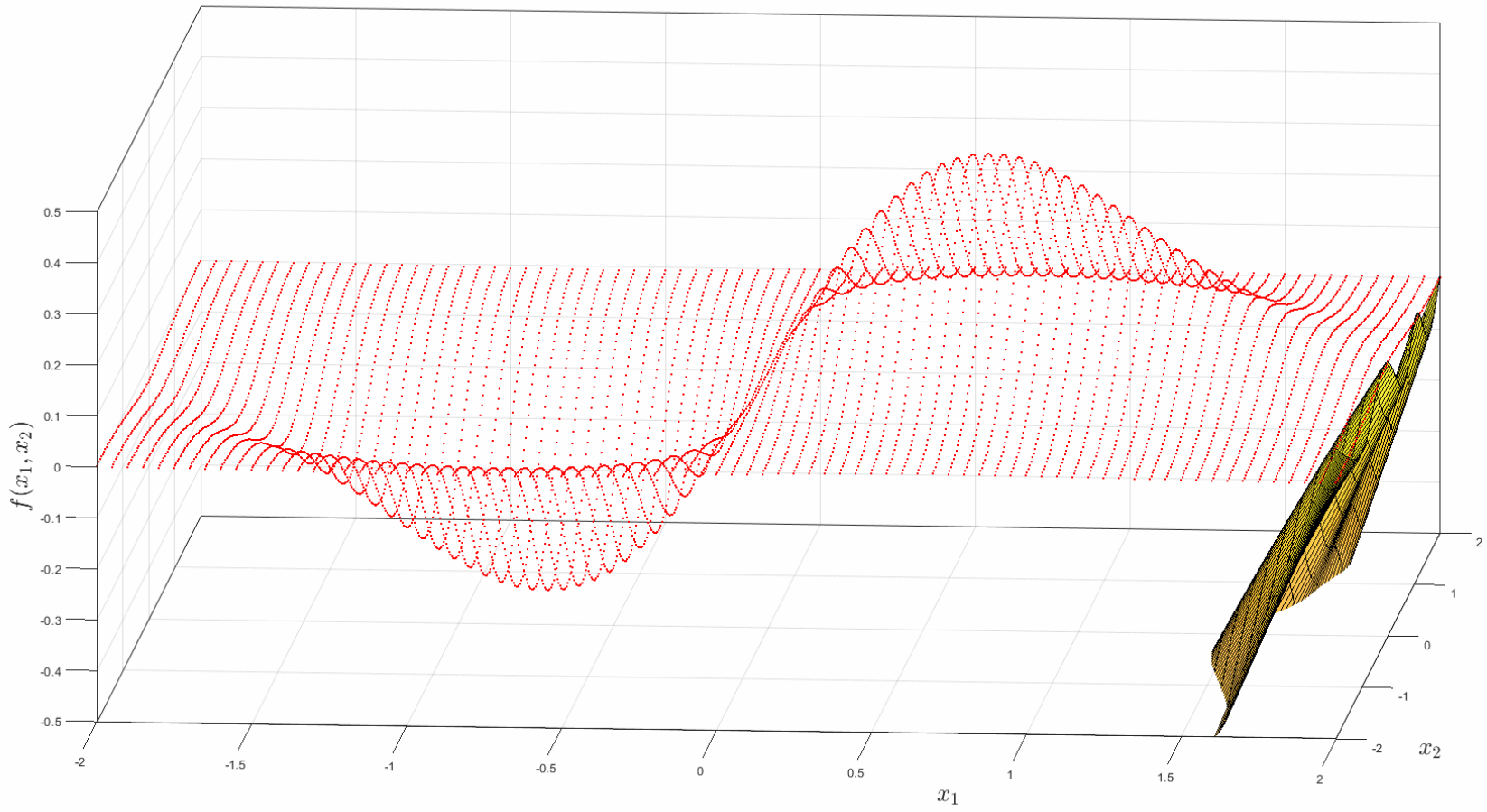
Dr. Carlos Villaseñor


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
f(x)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
x

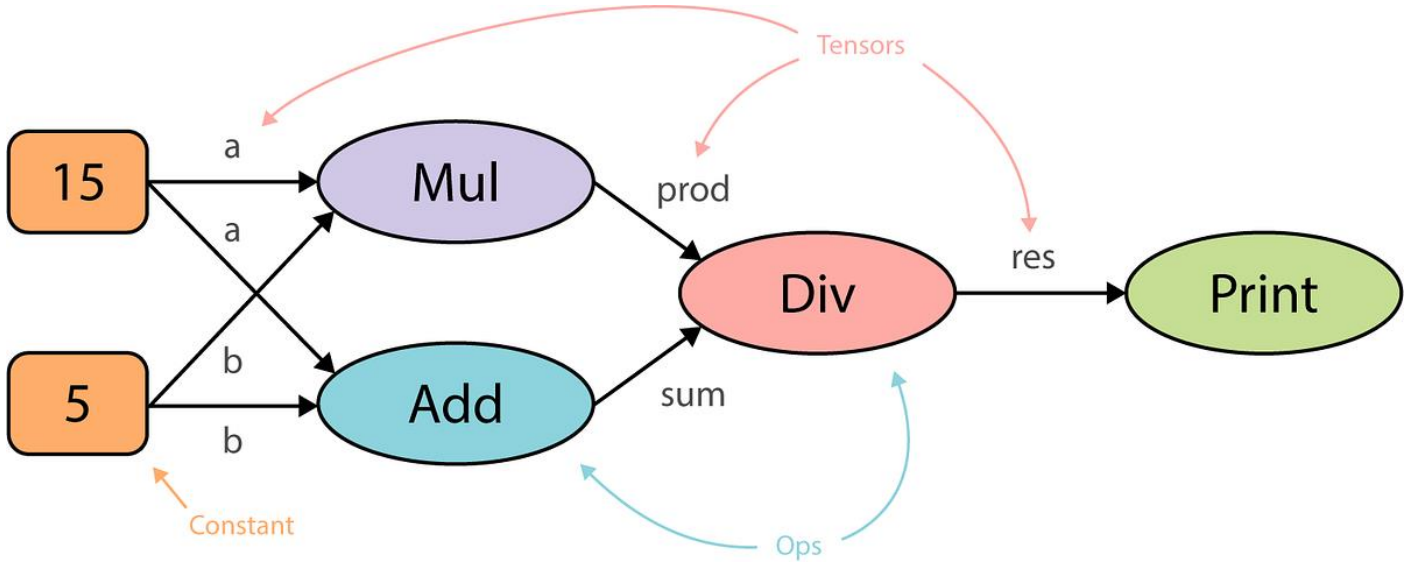

1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
f(x)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
x

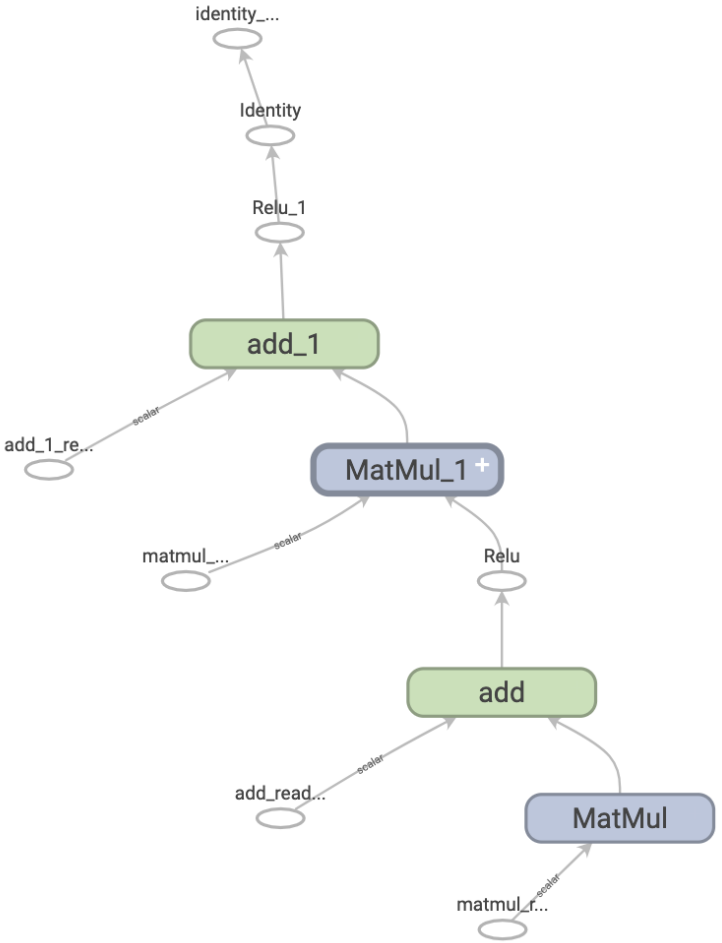

1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
f(x)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
x

Backpropagation for Regression

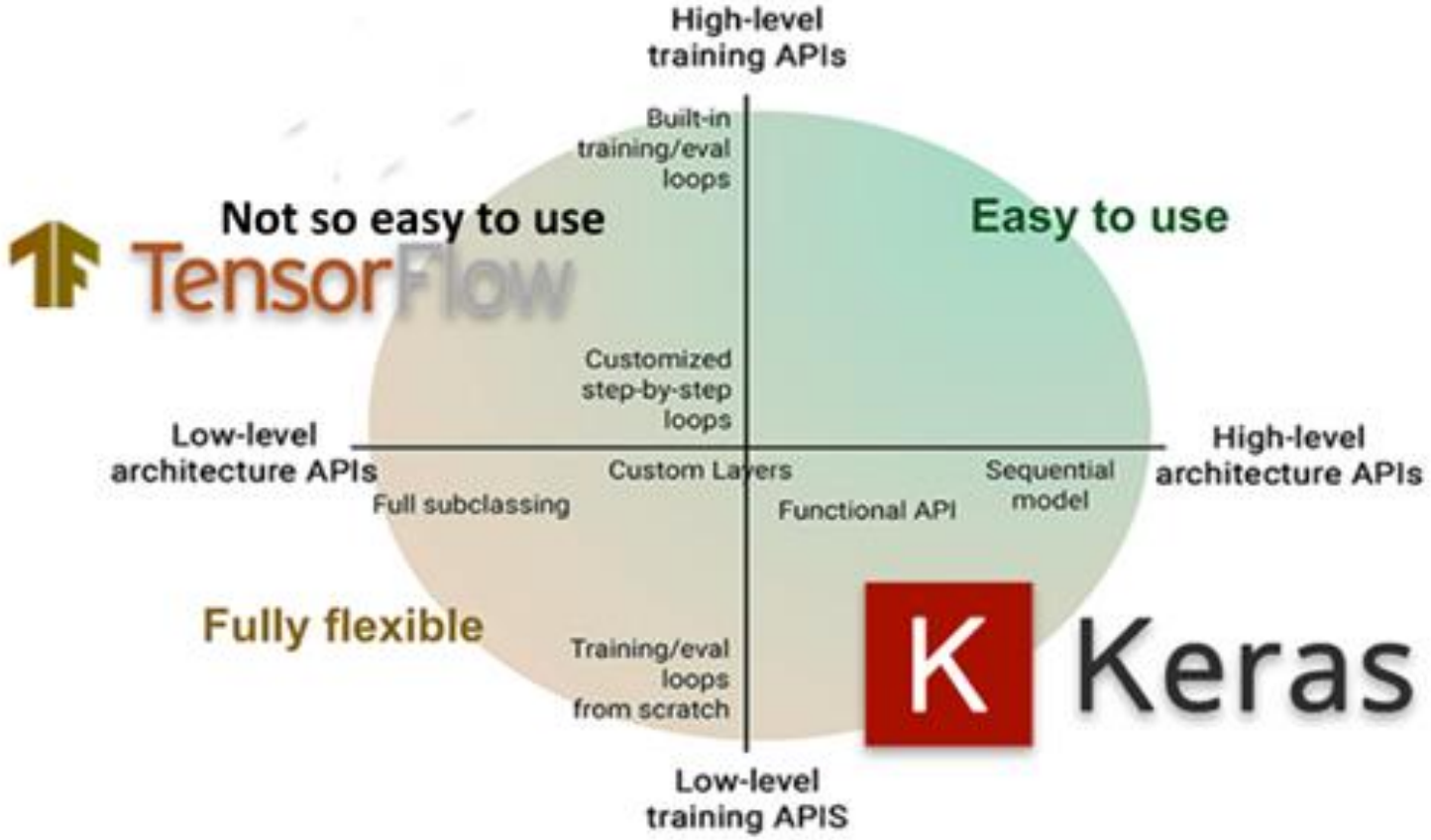


Keras/Tensorflow





Keras/Tensorflow



Keras/Tensorflow

	Keras	TensorFlow	PyTorch
Level of API	high-level API[1]	Both high & low level APIs	Lower-level API[2]
Speed	Slow	High	High
Architecture	Simple, more readable and concise	Not very easy to use	Complex[3]
Debugging	No need to debug	Difficult to debugging	Good debugging capabilities
Dataset Compatibility	Slow & Small	Fast speed & large	Fast speed & large datasets
Popularity Rank	1	2	3
Uniqueness	Multiple back-end support	Object Detection Functionality	Flexibility & Short Training Duration
Created By	Not a library on its own	Created by Google	Created by Facebook[4]
Ease of use	User-friendly	Incomprehensive API	Integrated with Python language
Computational graphs used	Static graphs	Static graphs	Dynamic computation graphs[5]