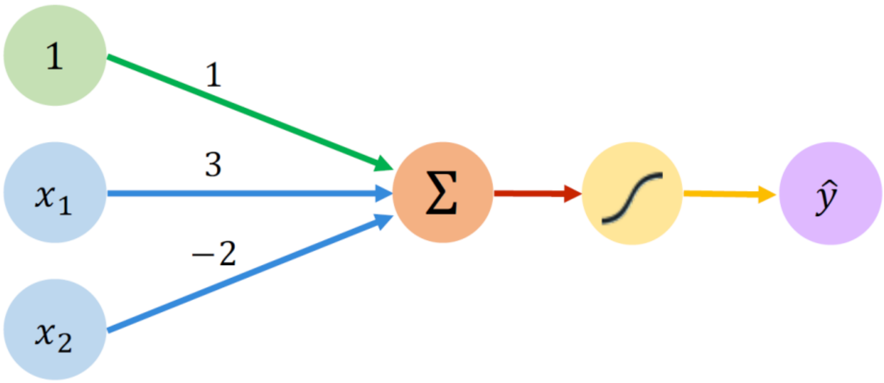
(Artificial) Neural Networks: From Perceptron to MLP

Industrial AI Lab.

Prof. Seungchul Lee

POSTECH

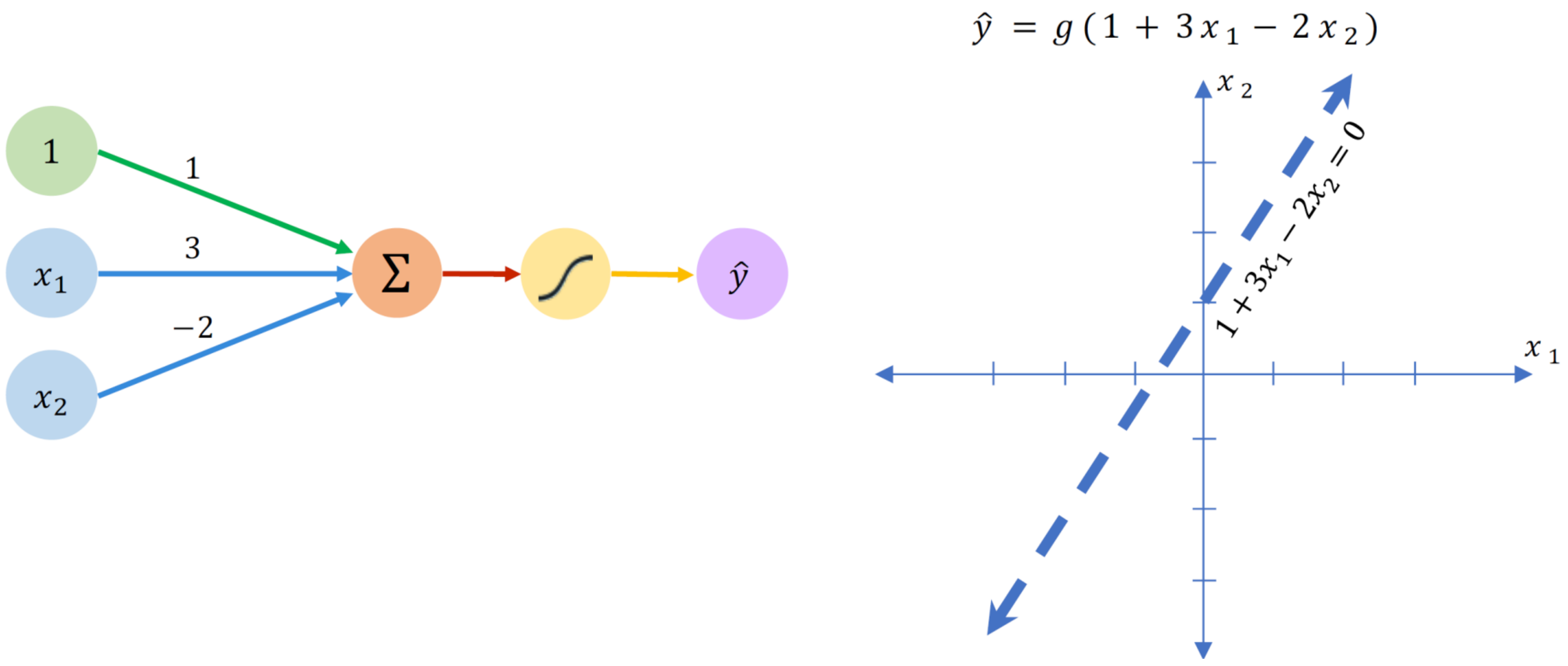
Perceptron: Example



We have: $\theta_0 = 1$ and $\boldsymbol{\theta} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

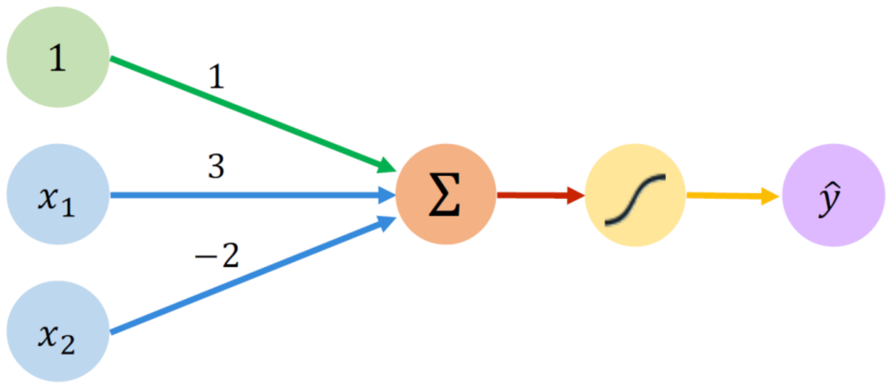
$$\hat{y} = g(\theta_0 + \boldsymbol{X}^T \boldsymbol{\theta})$$
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$
$$\hat{y} = g(\underbrace{1 + 3x_1 - 2x_2}_{\text{This is just a line in 2D!}})$$

Perceptron: Example



Source: 6.S191 Intro. to Deep Learning at MIT

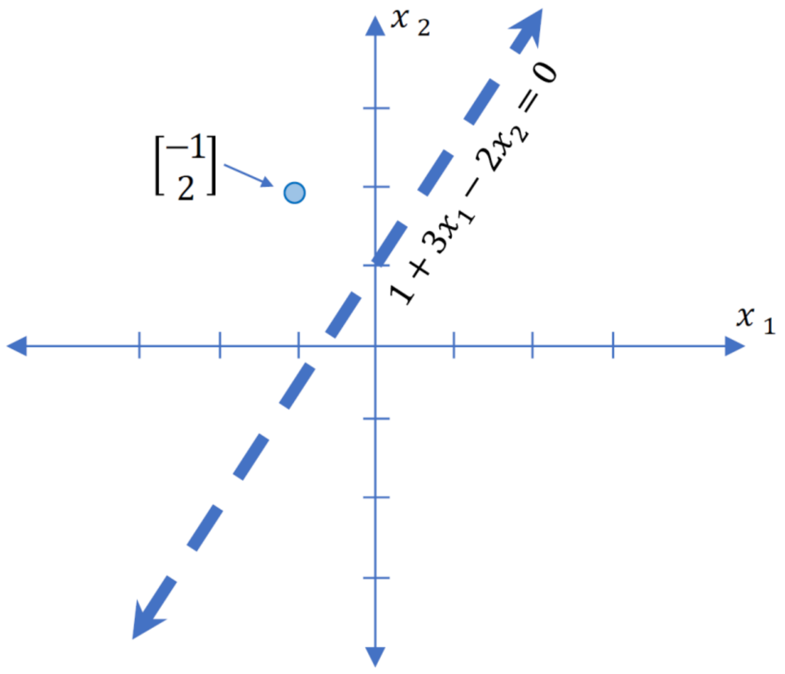
Perceptron: Example



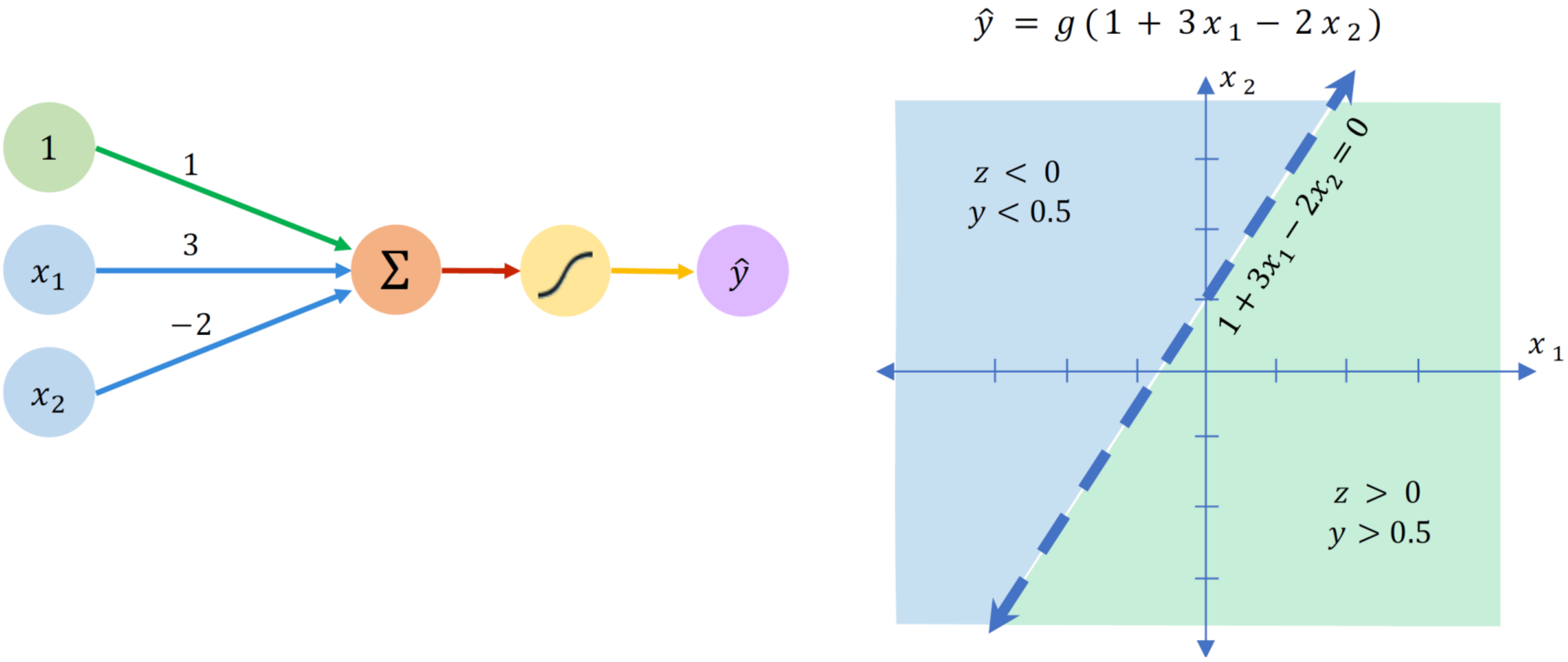
Assume we have input: $\boldsymbol{X} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

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

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

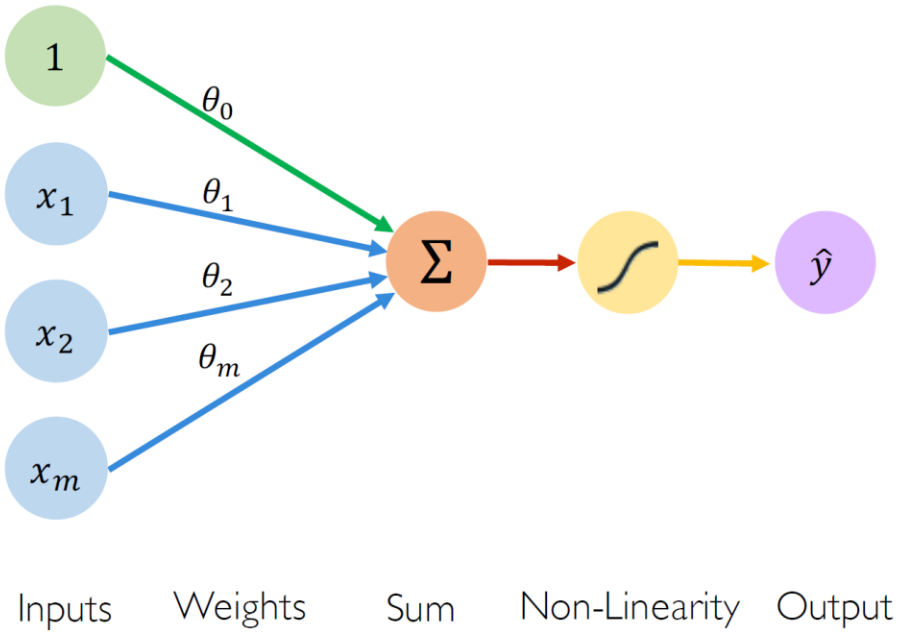


Perceptron: Example



Source: 6.S191 Intro. to Deep Learning at MIT

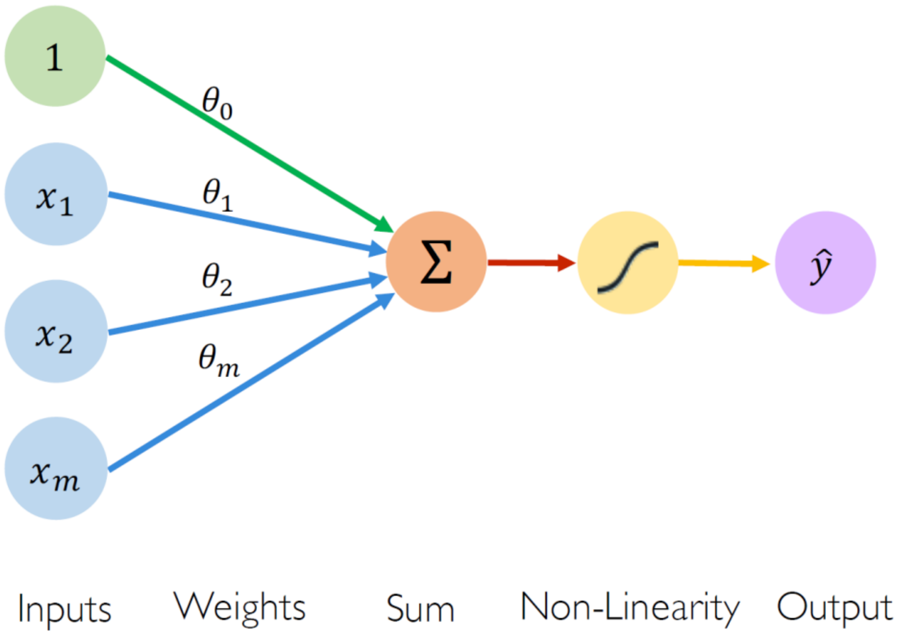
Perceptron: Forward Propagation



$$\overbrace{\hat{y}}^{\text{Output}} = \underbrace{g}_{\text{Non-linear activation function}} \left(\underbrace{\theta_0}_{\text{Bias}} + \overbrace{\sum_{i=1}^{m} x_i \, \theta_i}^{\text{Linear combination of inputs}} \right)$$

Source: 6.S191 Intro. to Deep Learning at MIT

Perceptron: Forward Propagation

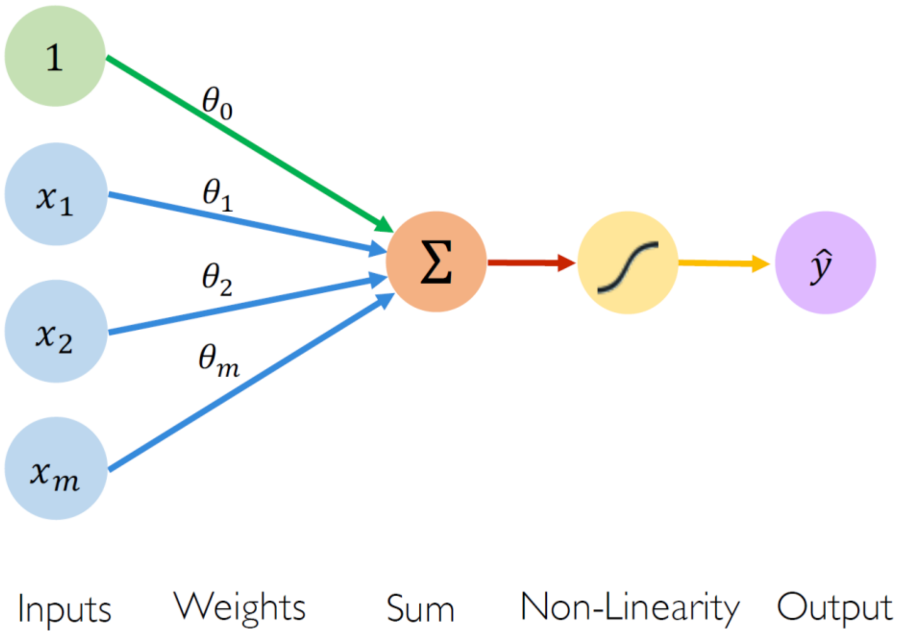


$$\hat{y} = g\left(\theta_0 + \sum_{i=1}^{m} x_i \theta_i\right)$$

$$\hat{y} = g(\theta_0 + \boldsymbol{X}^T \boldsymbol{\theta})$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$

Perceptron: Forward Propagation

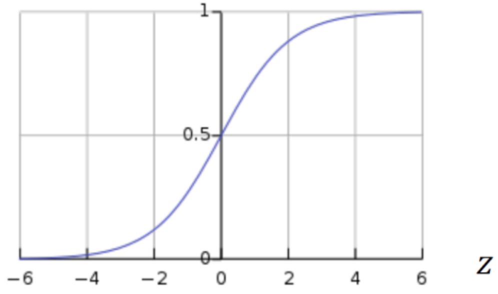


Activation Functions

$$\hat{y} = \boxed{g}\left(\theta_0 + \boldsymbol{X}^T \boldsymbol{\theta} \right)$$

- Example: sigmoid function

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



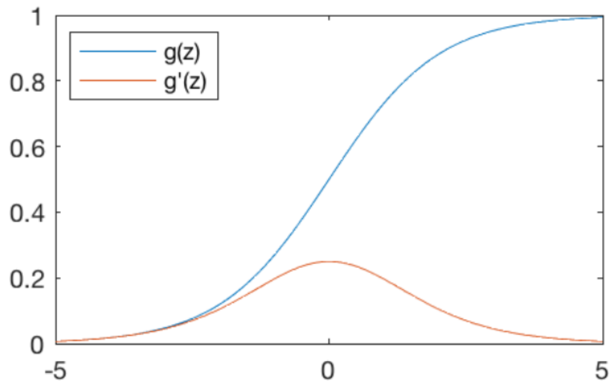
Source: 6.S191 Intro. to Deep Learning at MIT

Common Activation Functions

Discuss later

Sigmoid Function	Hyperbolic Tangent	Rectified Linear Unit (ReLU)

Sigmoid Function

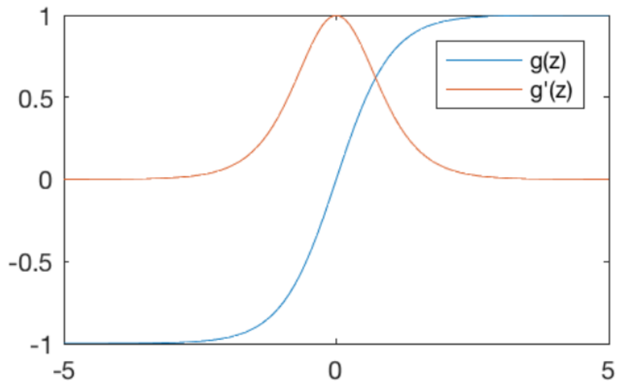


$$g(z) = \frac{1}{1 + e^{-z}}$$

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

`tf.nn.sigmoid(z)`

Hyperbolic Tangent

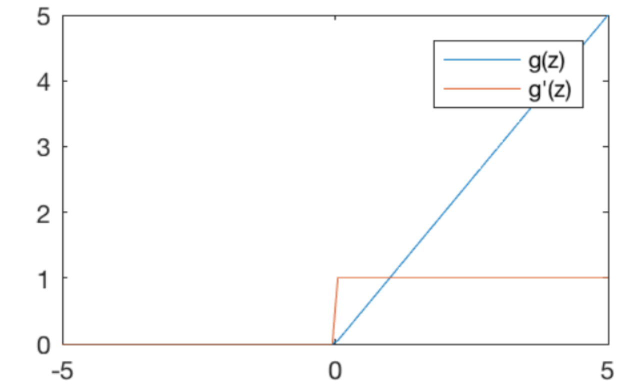


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

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

`tf.nn.tanh(z)`

Rectified Linear Unit (ReLU)



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

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

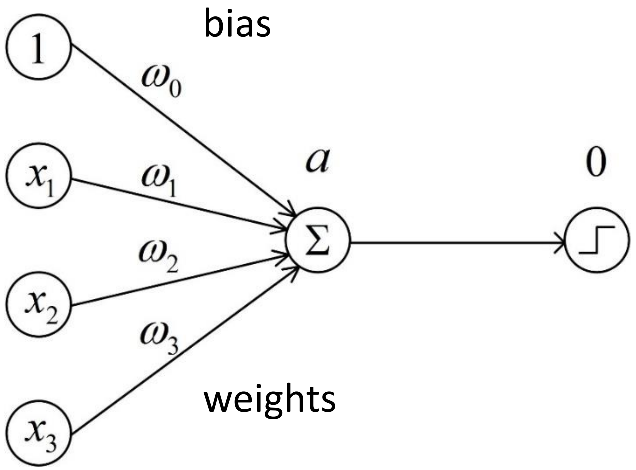
`tf.nn.relu(z)`

Source: 6.S191 Intro. to Deep Learning at MIT

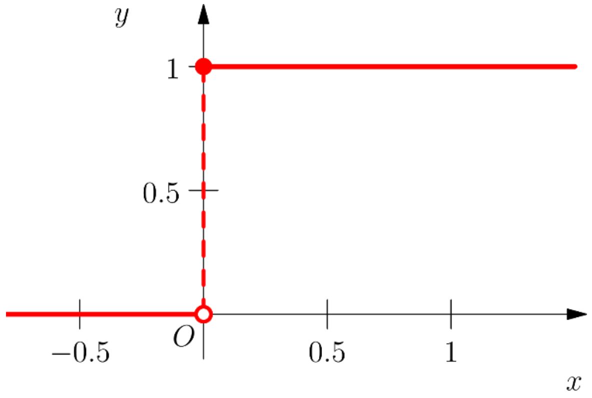
From Perceptron to MLP

Artificial Neural Networks: Perceptron

- Perceptron for $h(\theta)$ or $h(\omega)$
  - Neurons compute the weighted sum of their inputs
  - A neuron is activated or fired when the sum a is positive



$$a = \omega_0 + \omega_1 x_1 + \cdots$$
$$o = \sigma(\omega_0 + \omega_1 x_1 + \cdots)$$

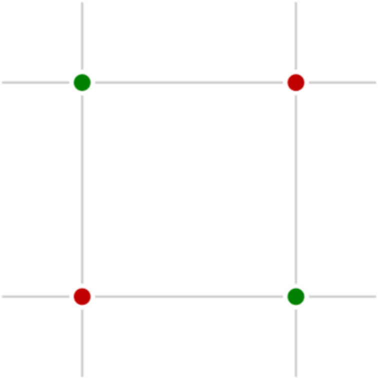


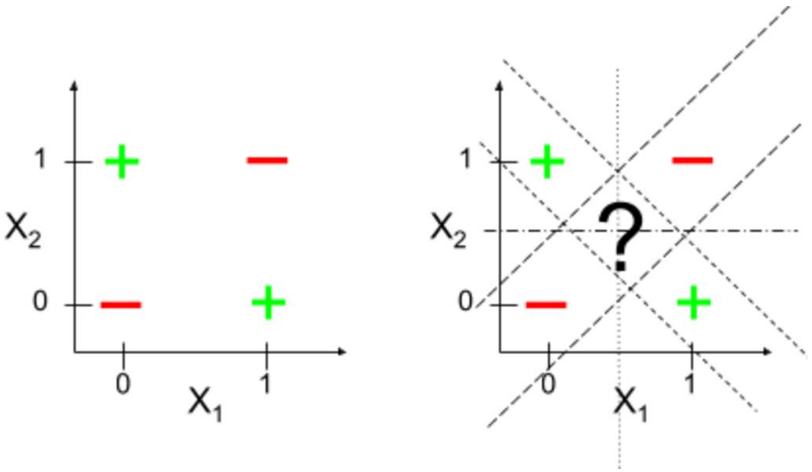
- A step function is not differentiable
- One neuron is often not enough
  - One hyperplane

XOR Problem

- Minsky-Papert Controversy on XOR
  - Not linearly separable
  - Limitation of perceptron

x_1	x_2	x_1 **XOR** x_2
0	0	0
0	1	1
1	0	1
1	1	0

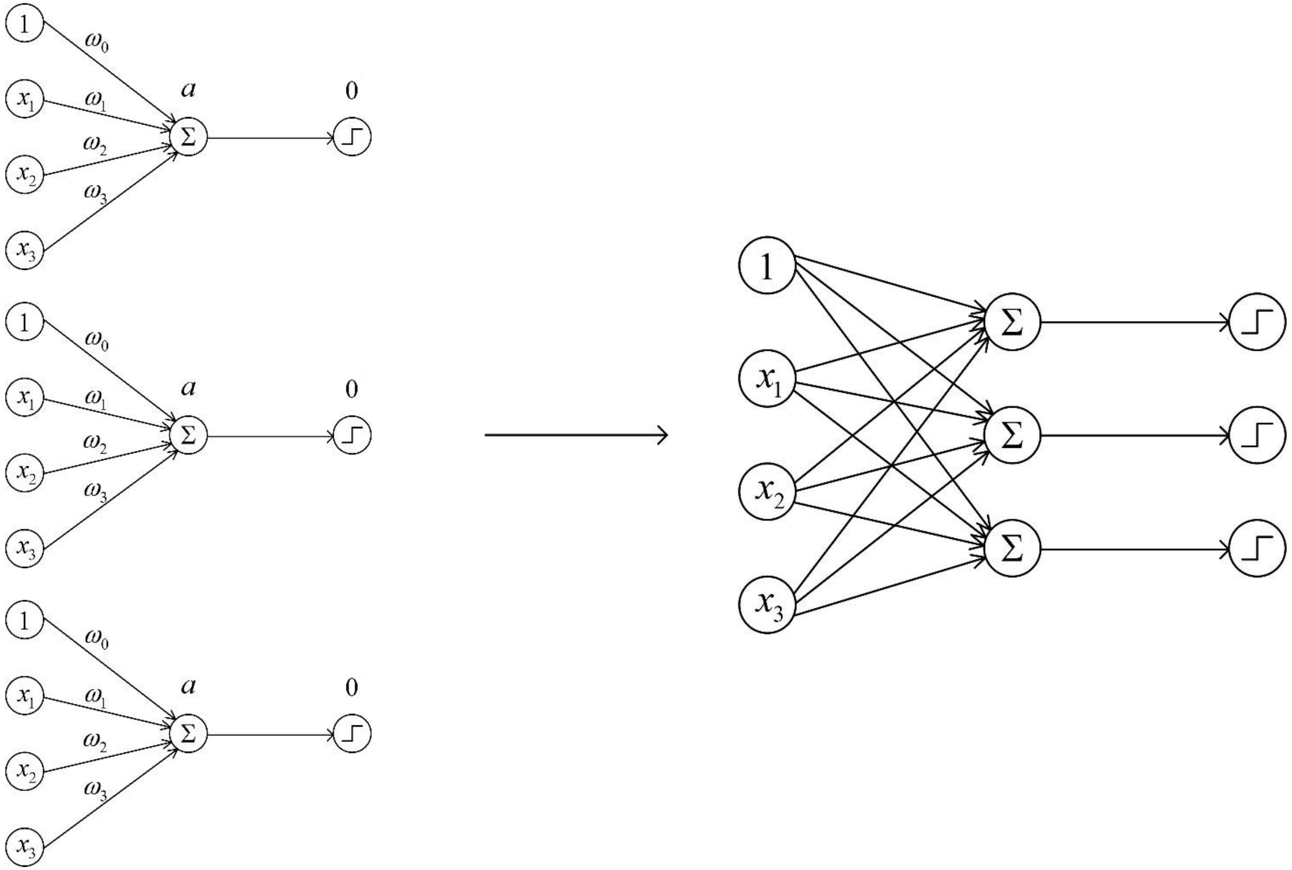




- Single neuron = one linear classification boundary

Artificial Neural Networks: MLP

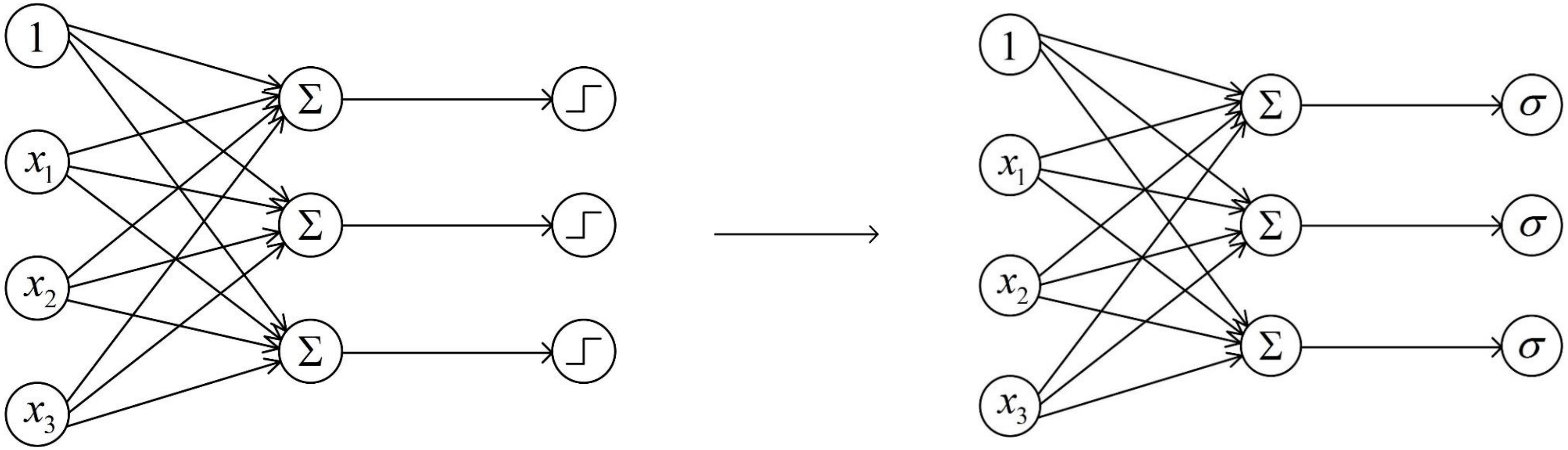
- Multi-layer Perceptron (MLP) = Artificial Neural Networks (ANN)
  – Multi neurons = multiple linear classification boundaries

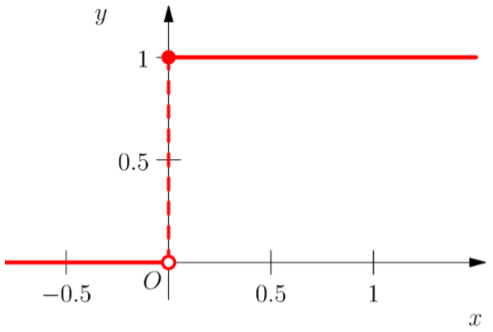


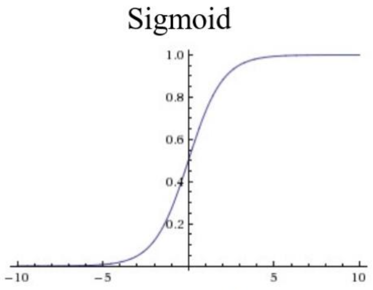
POSTECH

Artificial Neural Networks: Activation Function

- Differentiable nonlinear activation function



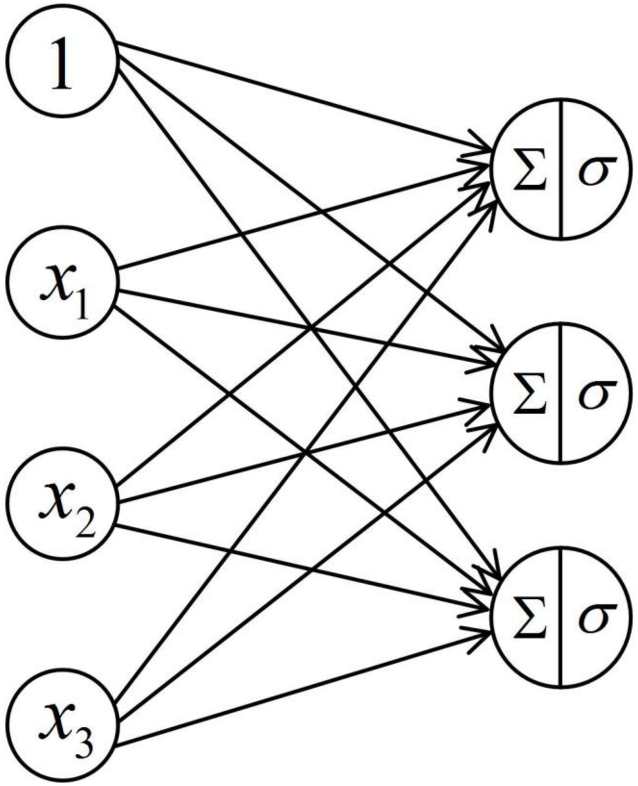




$$g(x) = \frac{1}{1 + e^{-x}}$$

Artificial Neural Networks

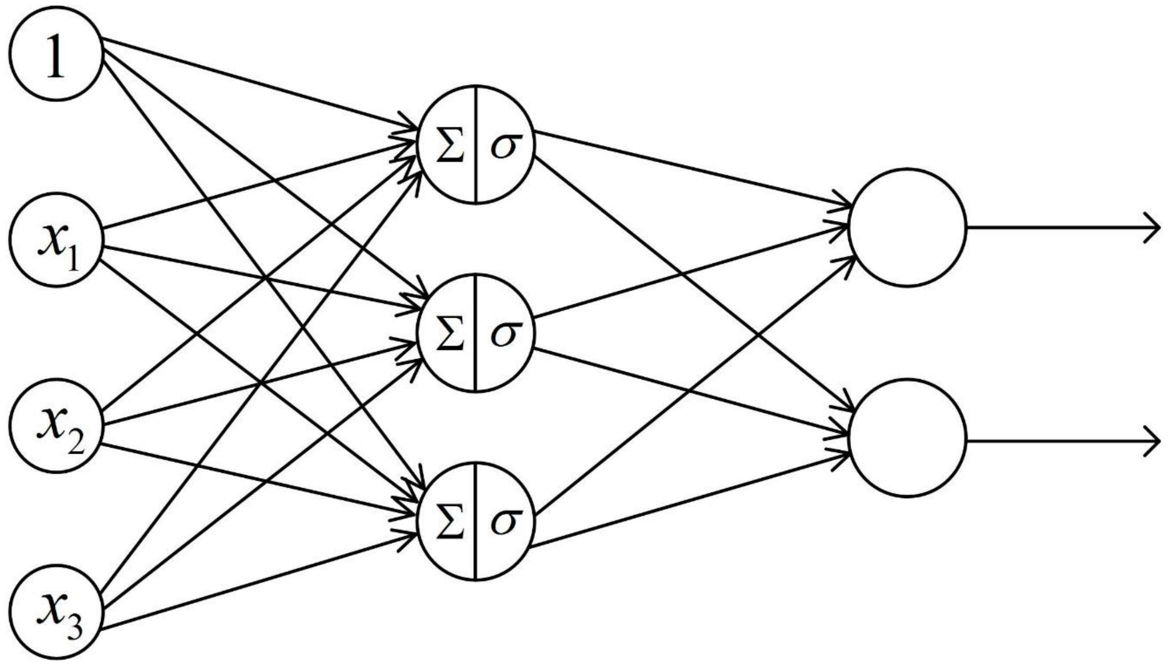
- In a compact representation



3 hyperplanes

Artificial Neural Networks

- Multi-layer perceptron
  - Features of features
  - Mapping of mappings

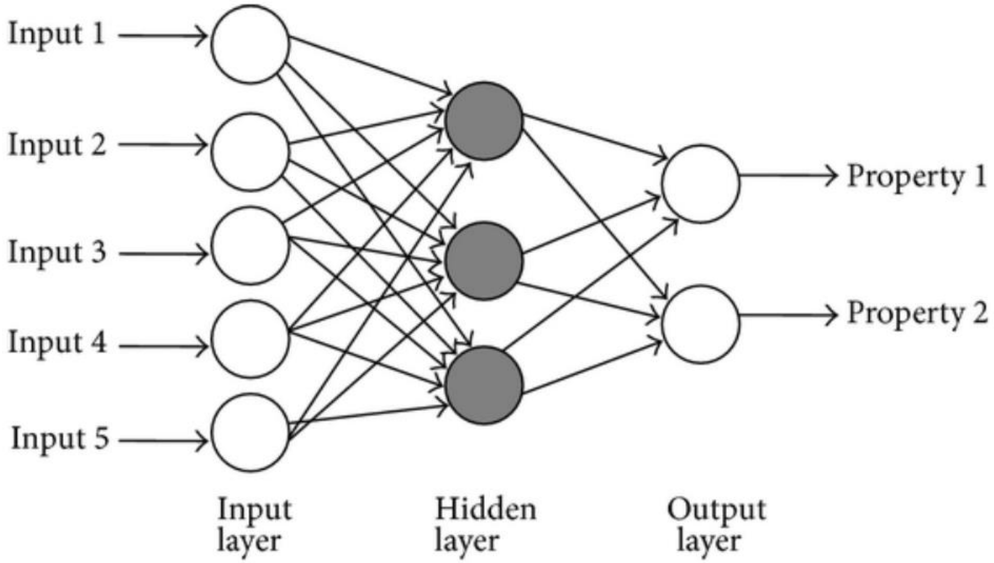


ANN: Architecture

- A single layer is not enough to be able to represent complex relationship between input and output

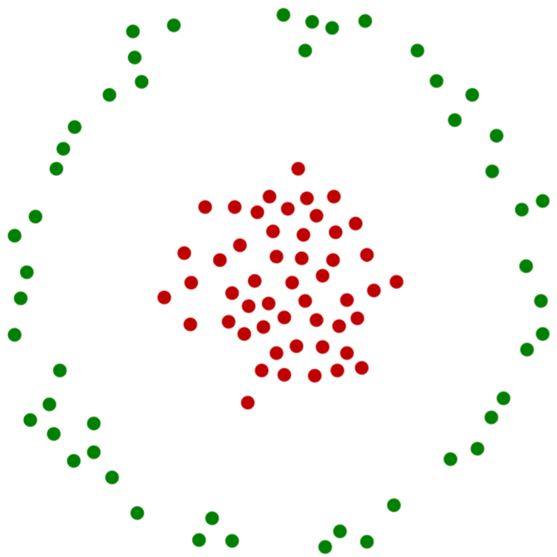
$\Longrightarrow$ perceptron with many layers and units

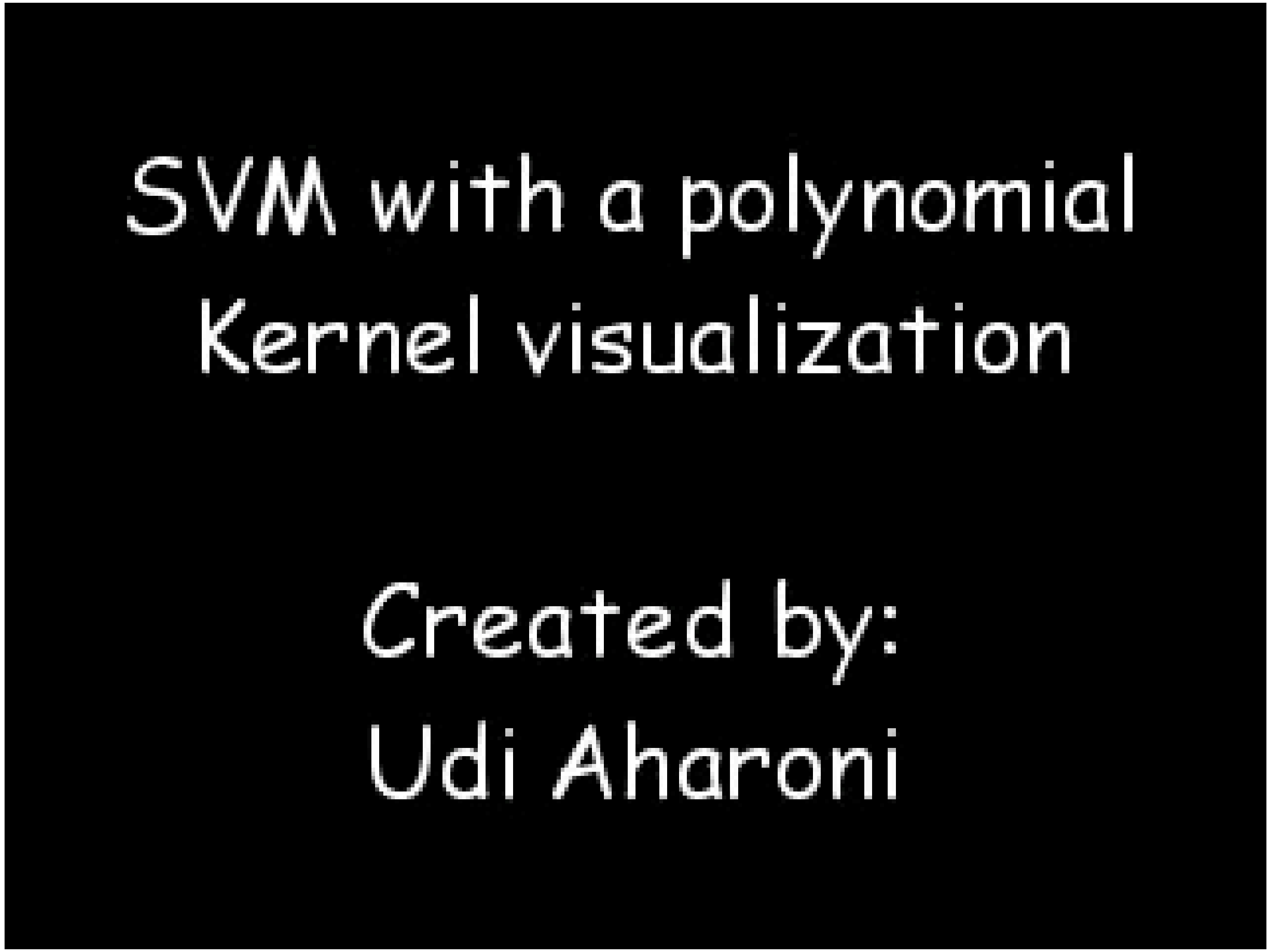
$$o_2 = \sigma_2\left(\theta_2^T o_1 + b_2\right) = \sigma_2\left(\theta_2^T \sigma_1\left(\theta_1^T x + b_1\right) + b_2\right)$$



Another Perspective:
ANN as Kernel Learning

Nonlinear Classification



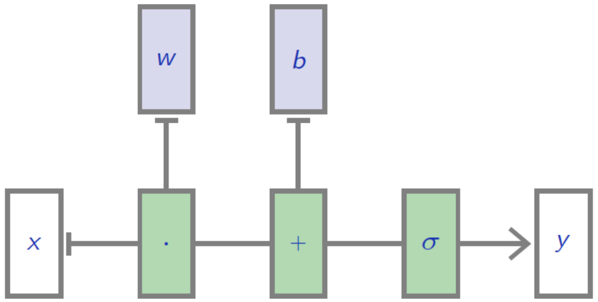


https://www.youtube.com/watch?v=3liCbRZPrZA

Neuron

- We can represent this “neuron” as follows:

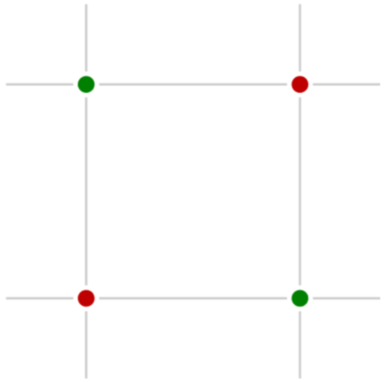
$$f(x) = \sigma(w \cdot x + b).$$



Source: Dr. Francois Fleuret at EPFL

XOR Problem

- The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be linearly separable.



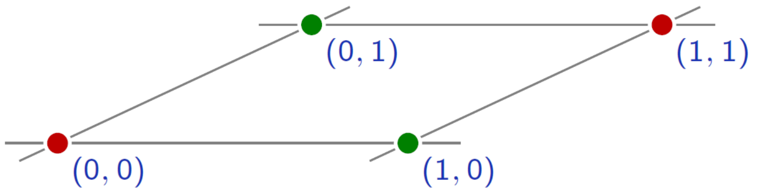
"xor"

Source: Dr. Francois Fleuret at EPFL

Nonlinear Mapping

- The XOR example can be solved by pre-processing the data to make the two populations linearly separable.

$$\Phi : (x_u, x_v) \mapsto (x_u, x_v, x_u x_v).$$

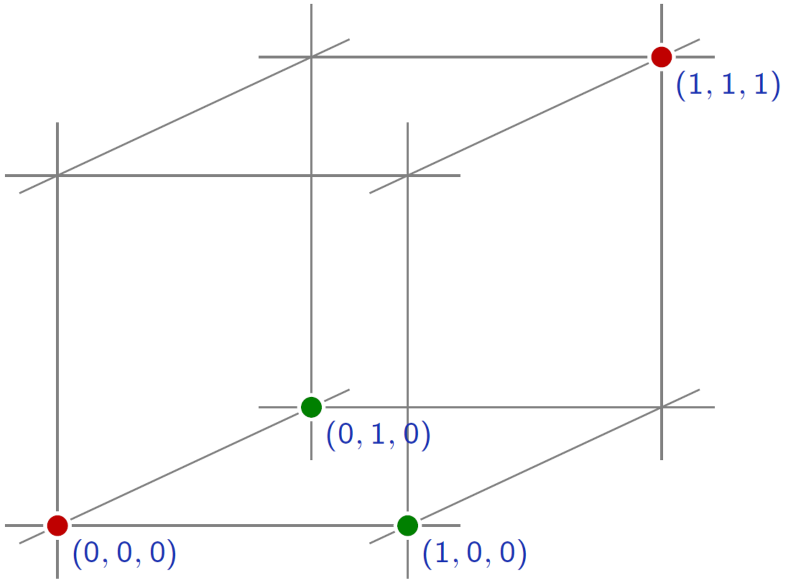


Source: Dr. Francois Fleuret at EPFL

Nonlinear Mapping

- The XOR example can be solved by pre-processing the data to make the two populations linearly separable.

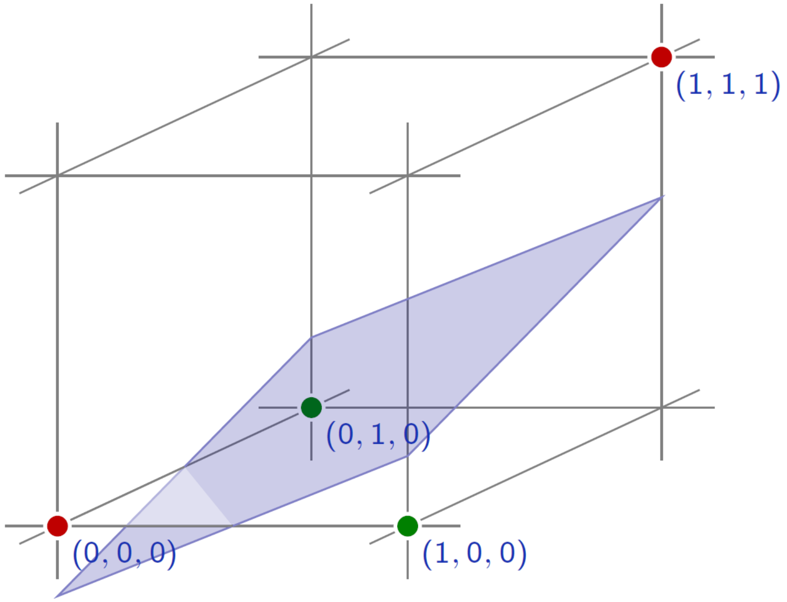
$$\Phi : (x_u, x_v) \mapsto (x_u, x_v, x_u x_v).$$



Nonlinear Mapping

- The XOR example can be solved by pre-processing the data to make the two populations linearly separable.

$$\Phi : (x_u, x_v) \mapsto (x_u, x_v, x_u x_v).$$



Source: Dr. Francois Fleuret at EPFL

Kernel

- Often we want to capture nonlinear patterns in the data
  - nonlinear regression: input and output relationship may not be linear
  - nonlinear classification: classes may note be separable by a linear boundary

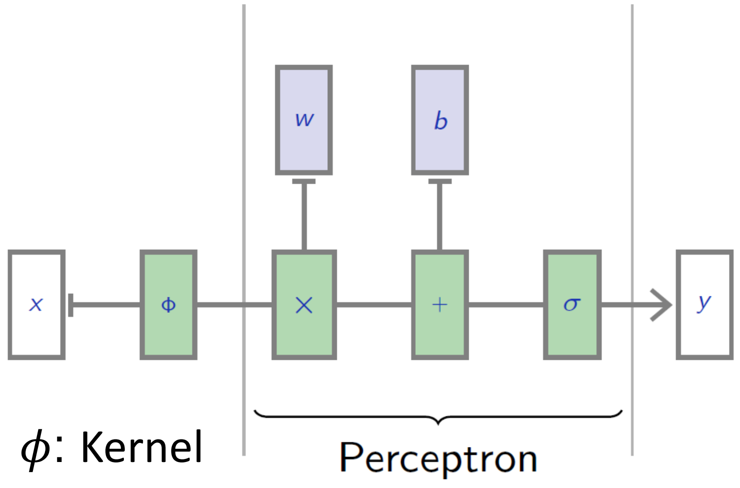
- Linear models (e.g. linear regression, linear SVM) are not just rich enough
  - by mapping data to higher dimensions where it exhibits linear patterns
  - apply the linear model in the new input feature space
  - mapping = changing the feature representation

- Kernels: make linear model work in nonlinear settings

Kernel + Neuron

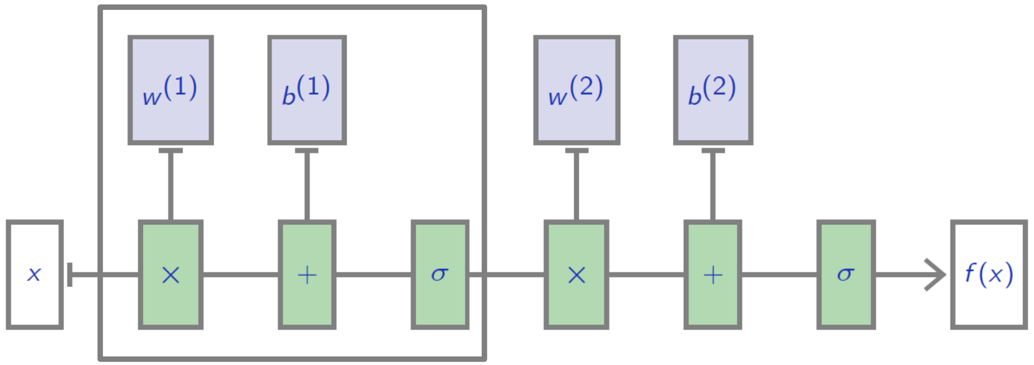
- Nonlinear mapping + neuron

$$\Phi : (x_u, x_v) \mapsto (x_u, x_v, x_u x_v).$$



Neuron + Neuron

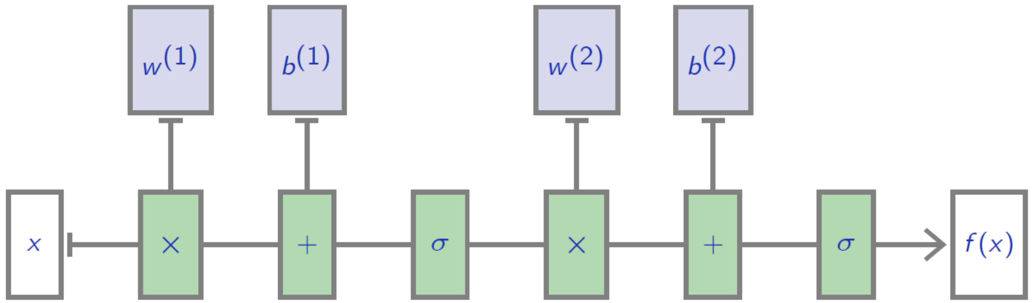
- Nonlinear mapping can be represented by another neurons



ϕ: Kernel

- Nonlinear Kernel
  - Nonlinear activation functions

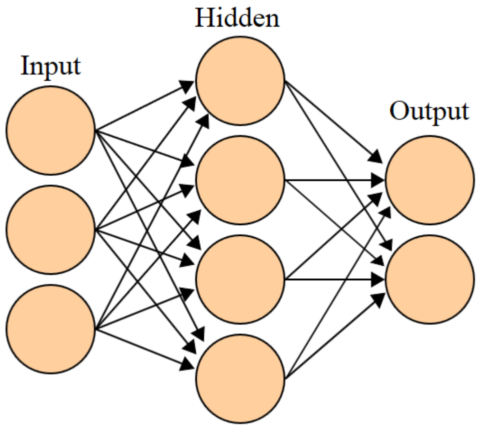
- Nonlinear mapping can be represented by another neurons
- We can generalize an MLP



Summary

- Universal function approximator
- Universal function classifier

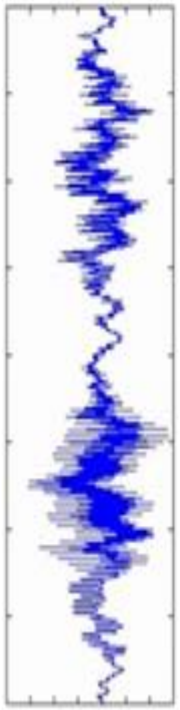
- Parameterized

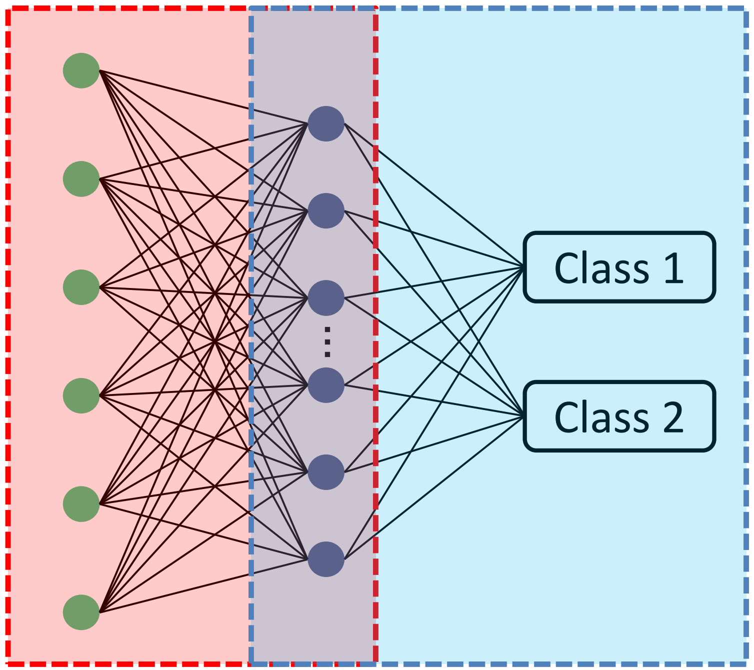


$$x \longrightarrow \boxed{\hat{y} = f_{\omega_1,\cdots,\omega_k}(x)} \longrightarrow y$$

Artificial Neural Networks

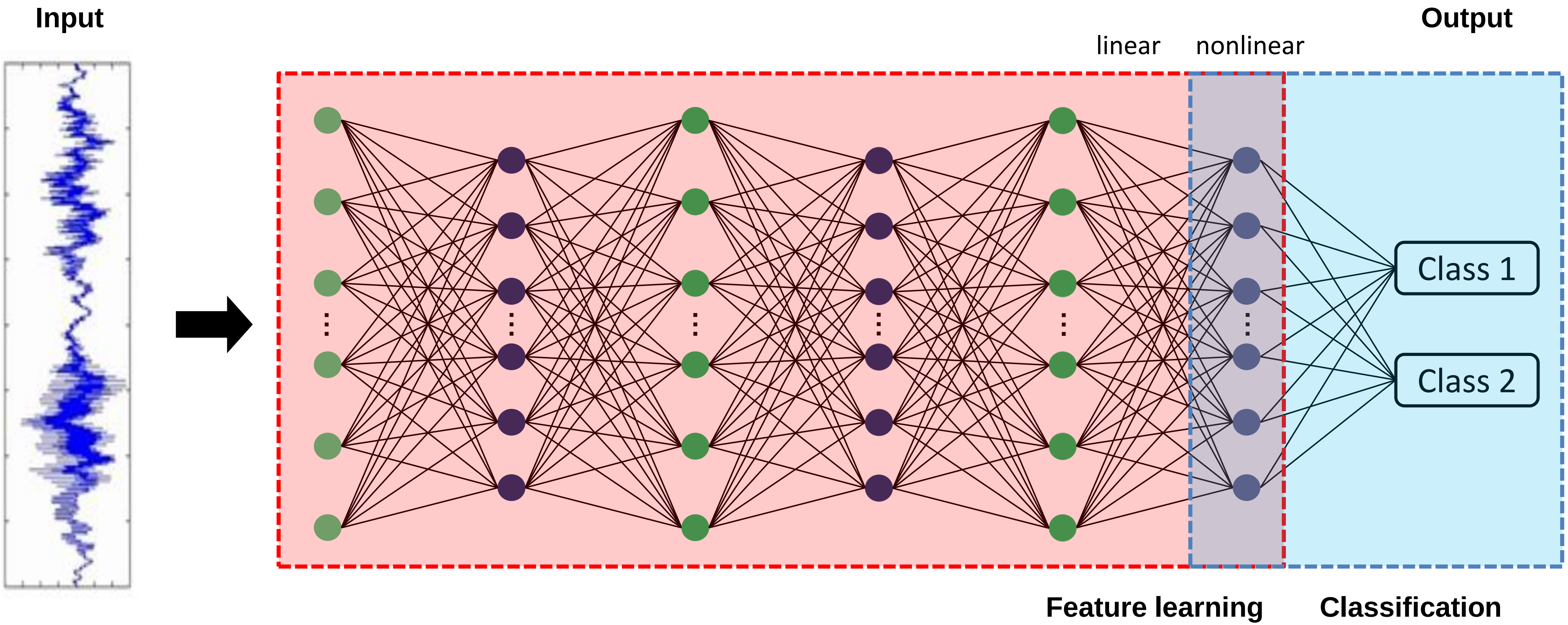
- Complex/Nonlinear universal function approximator
  - Linearly connected networks
  - Simple nonlinear neurons





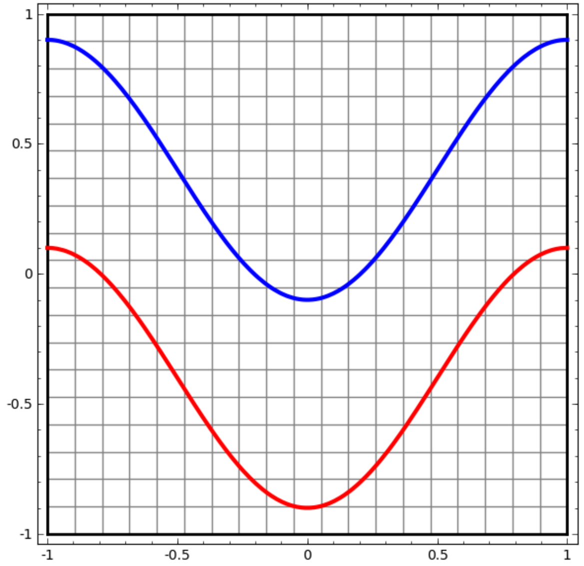
Deep Artificial Neural Networks

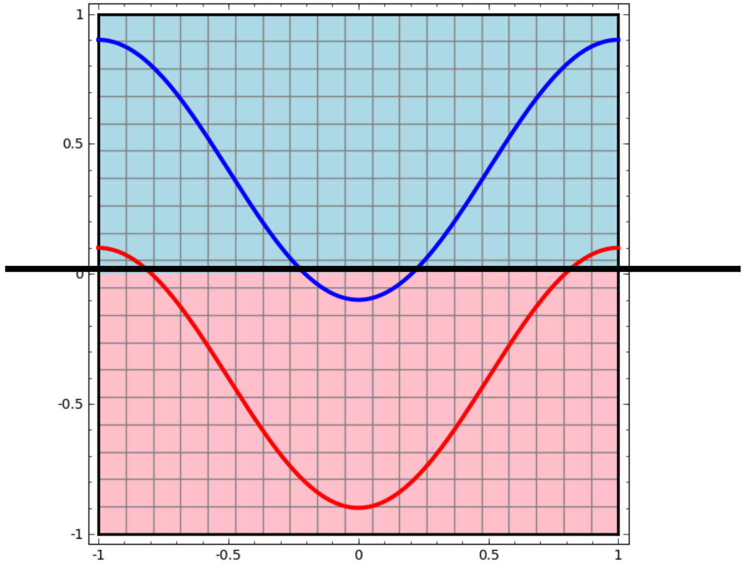
- Complex/Nonlinear universal function approximator
  - Linearly connected networks
  - Simple nonlinear neurons



Example: Linear Classifier

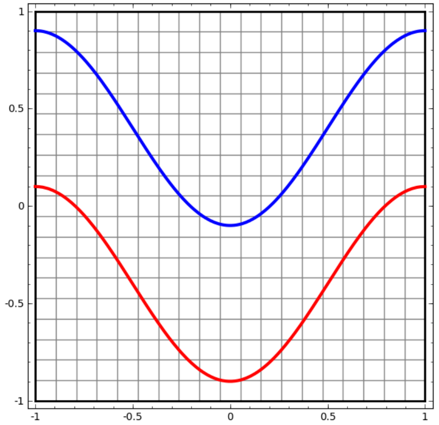
- Perceptron tries to separate the two classes of data by dividing them with a line

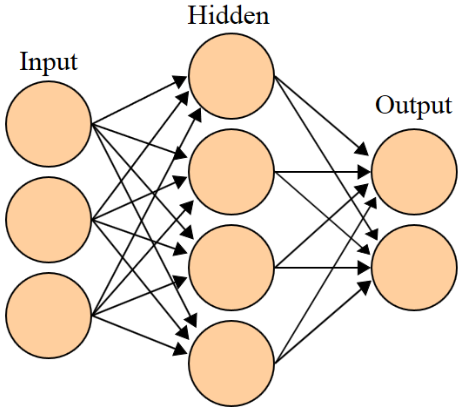


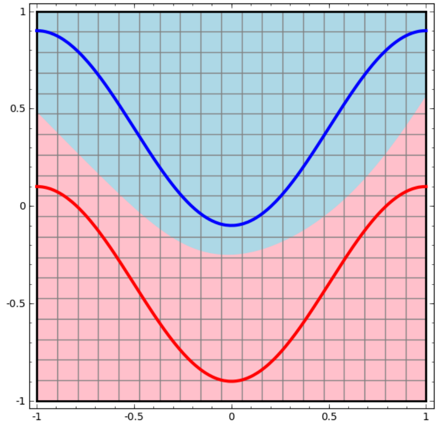


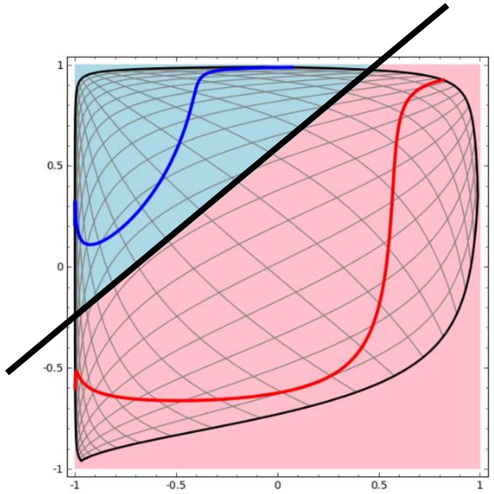
Example: Neural Networks

- The hidden layer learns a representation so that the data gets linearly separable

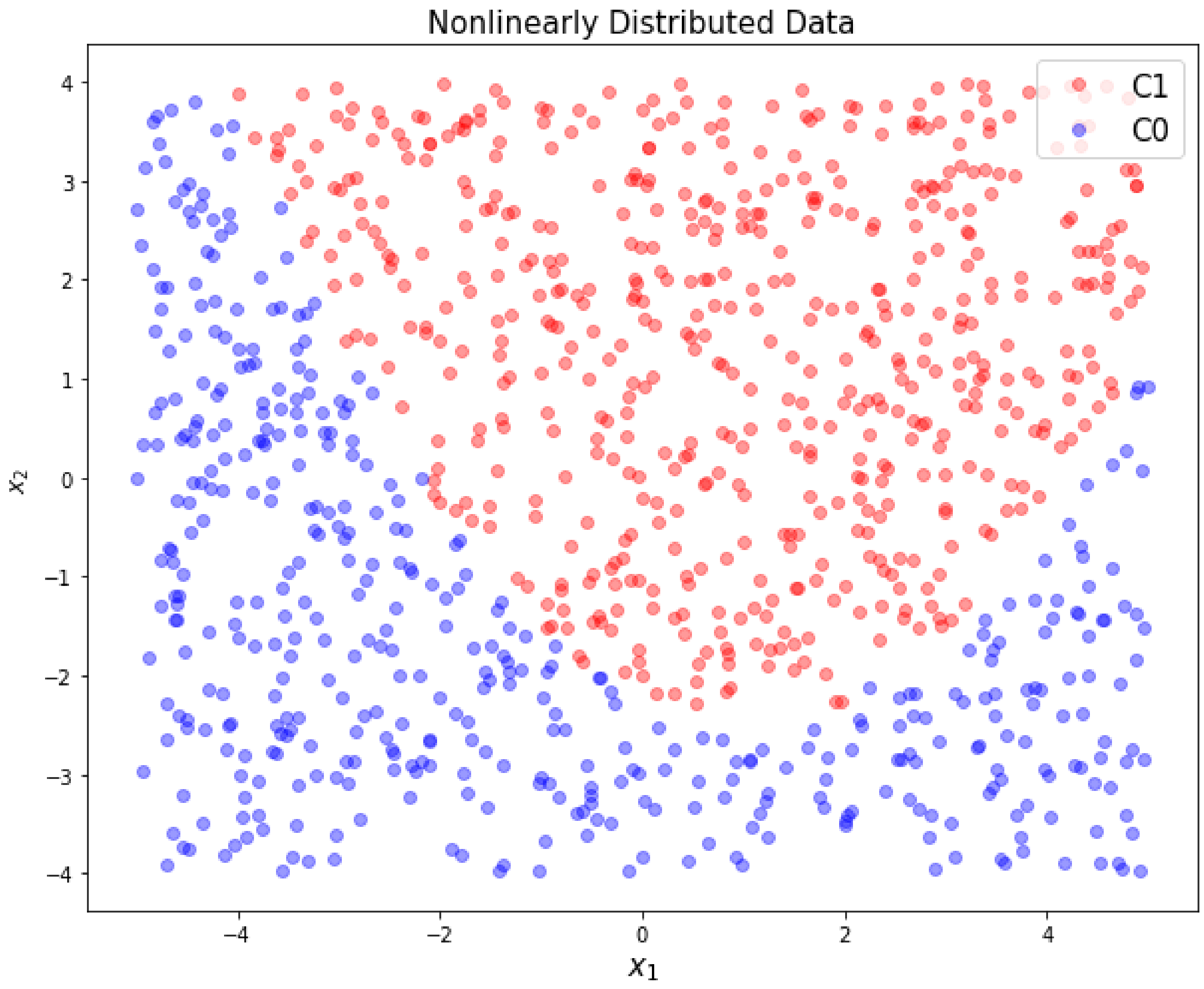




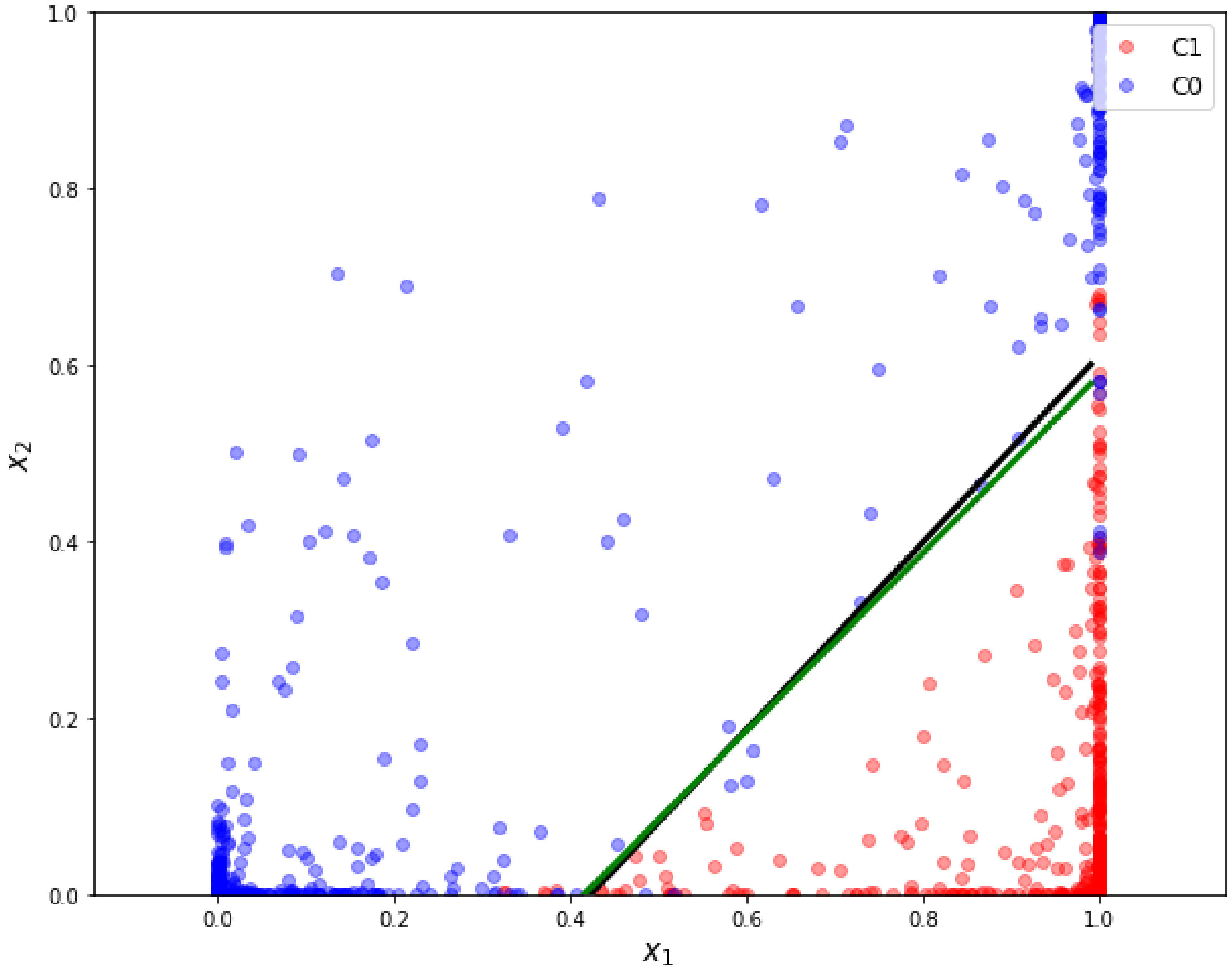




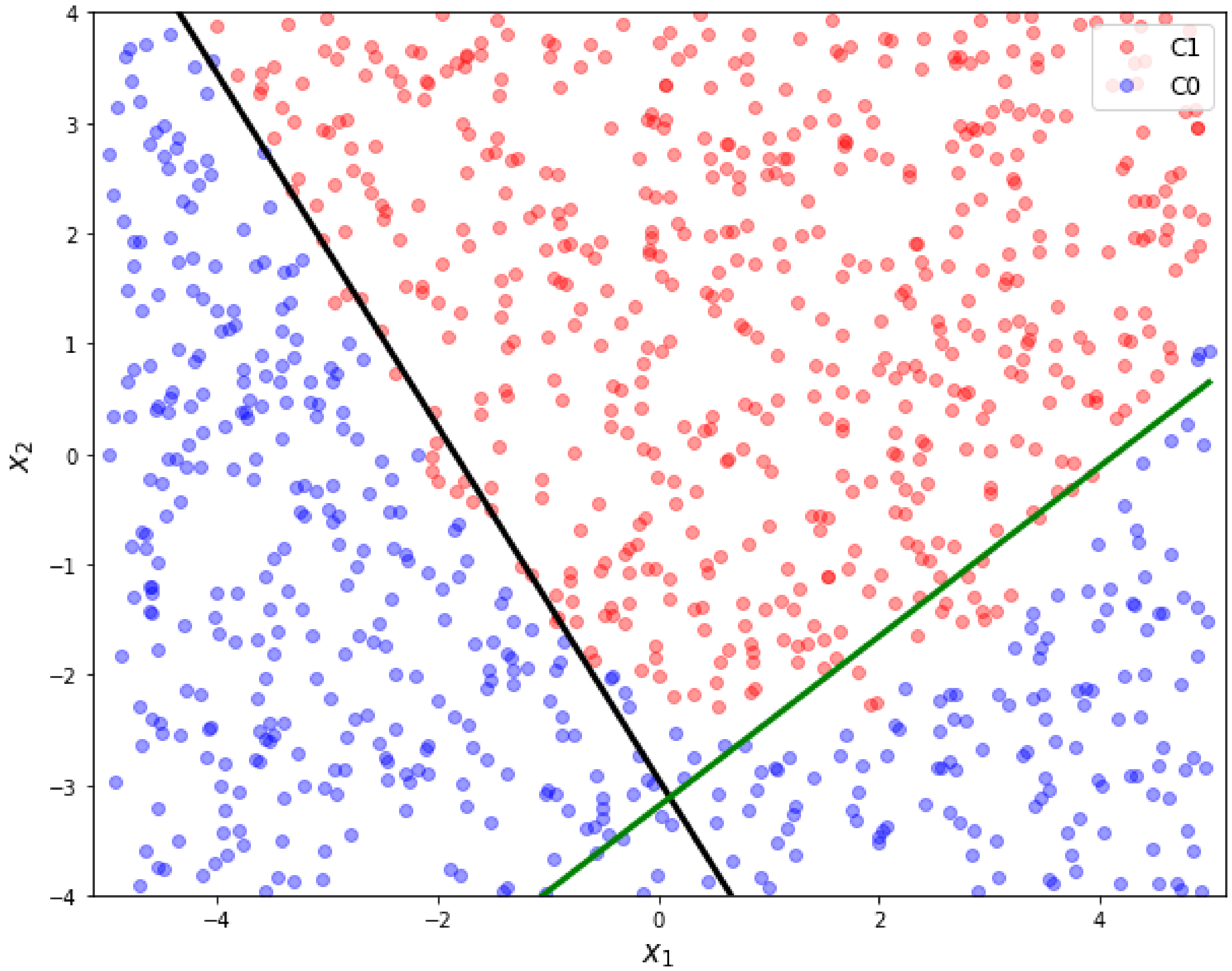
Nonlinearly Distributed Data



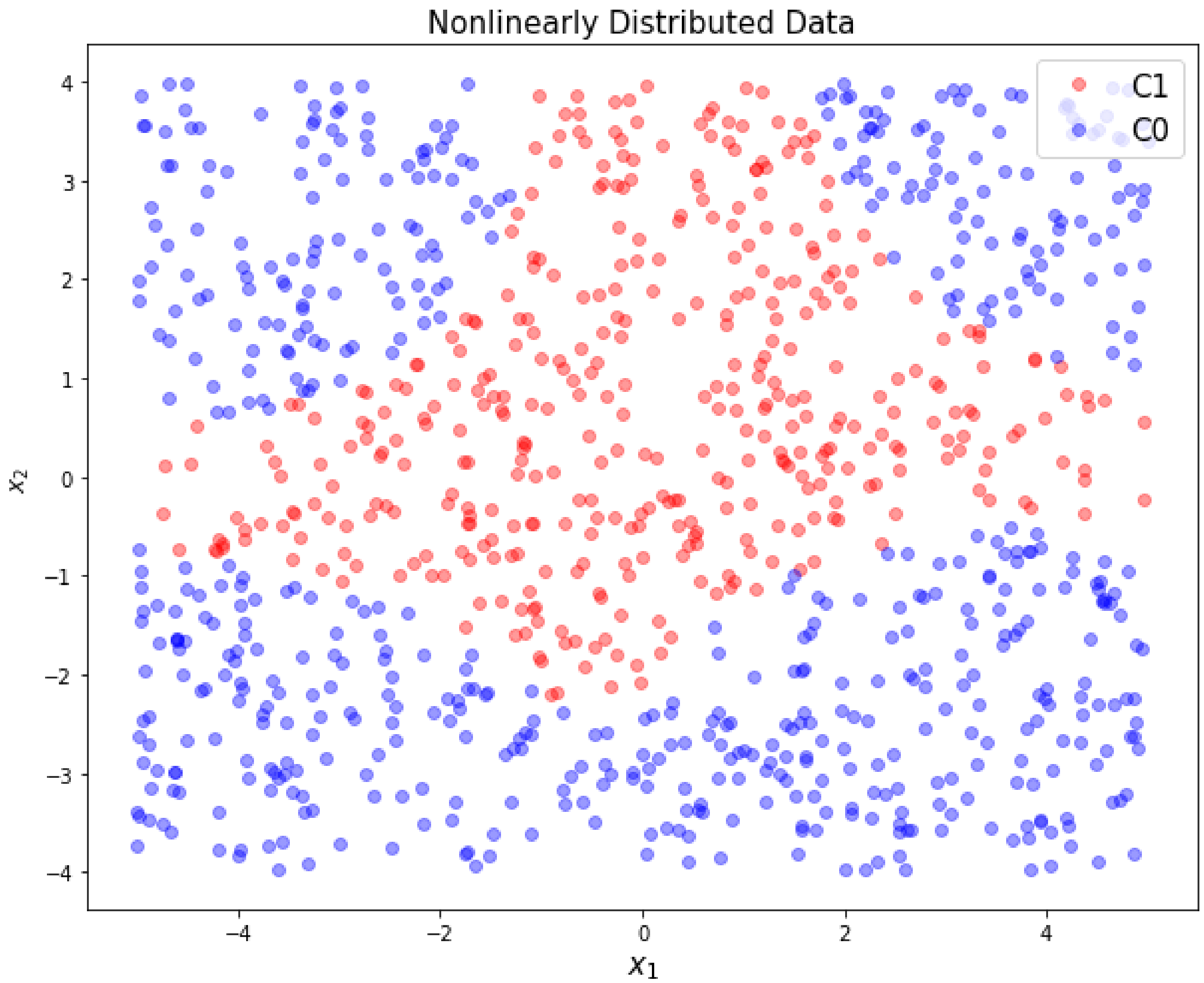
Multi Layers



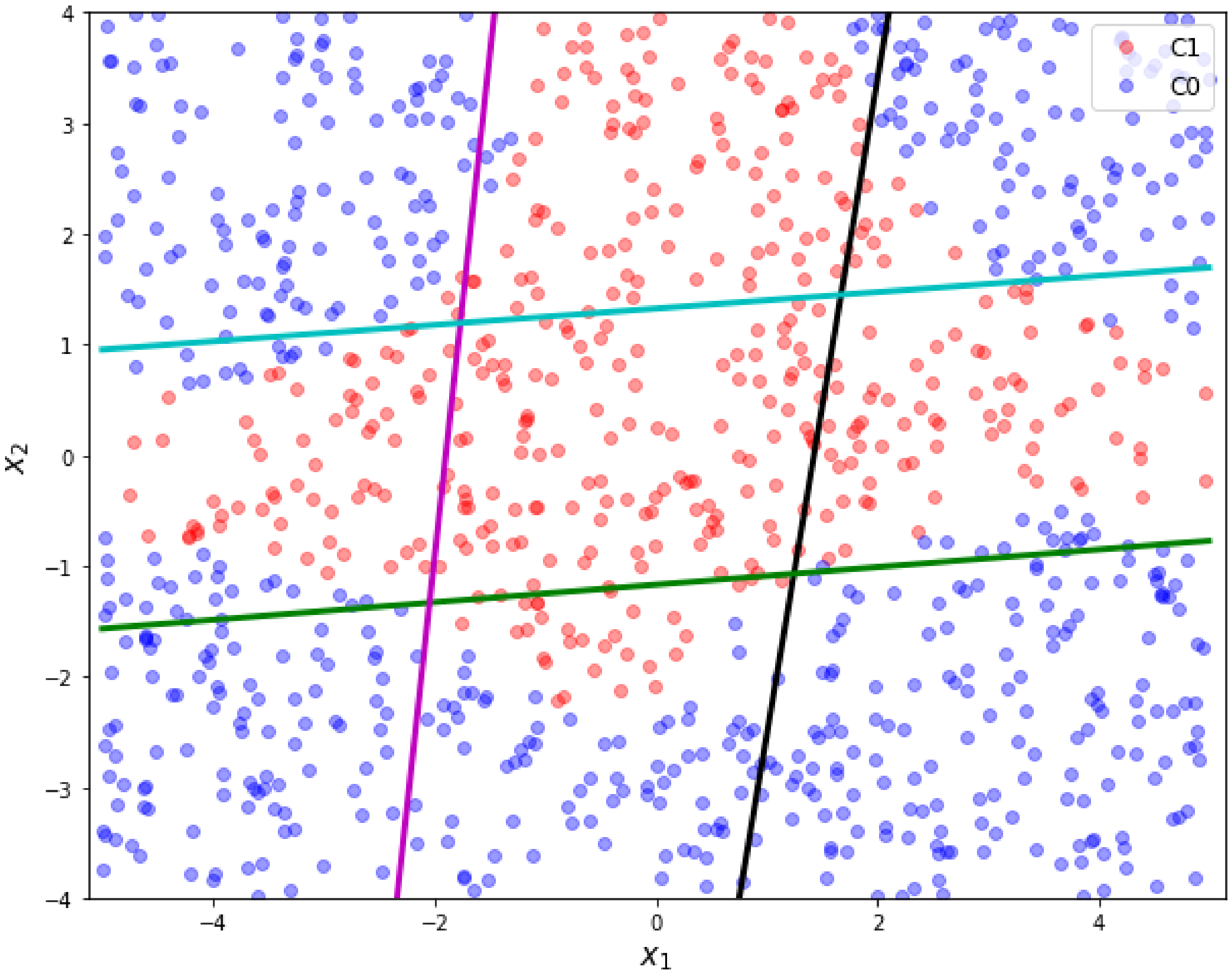
Multi Layers



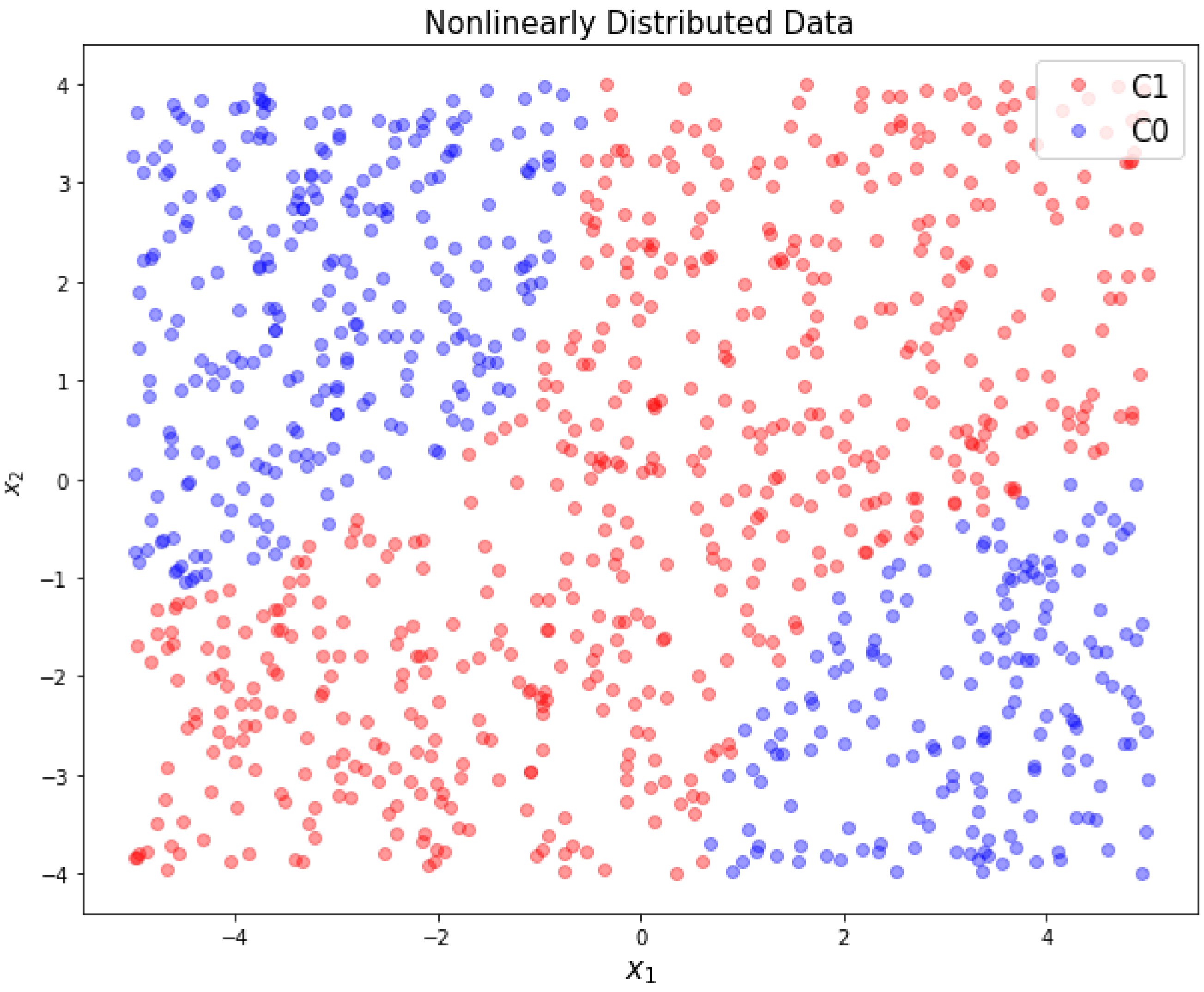
Nonlinearly Distributed Data



Multi Layers



Nonlinearly Distributed Data



Multi Layers