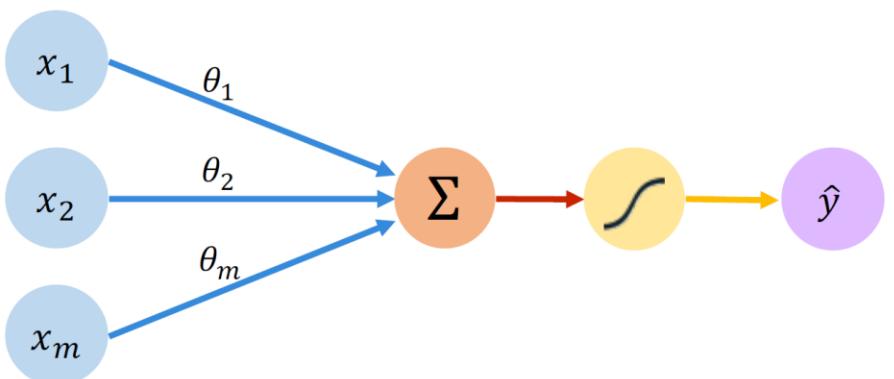




(Artificial) Neural Networks

Industrial AI Lab.
Prof. Seungchul Lee

Perceptron: Forward Propagation



Inputs Weights Sum Non-Linearity Output

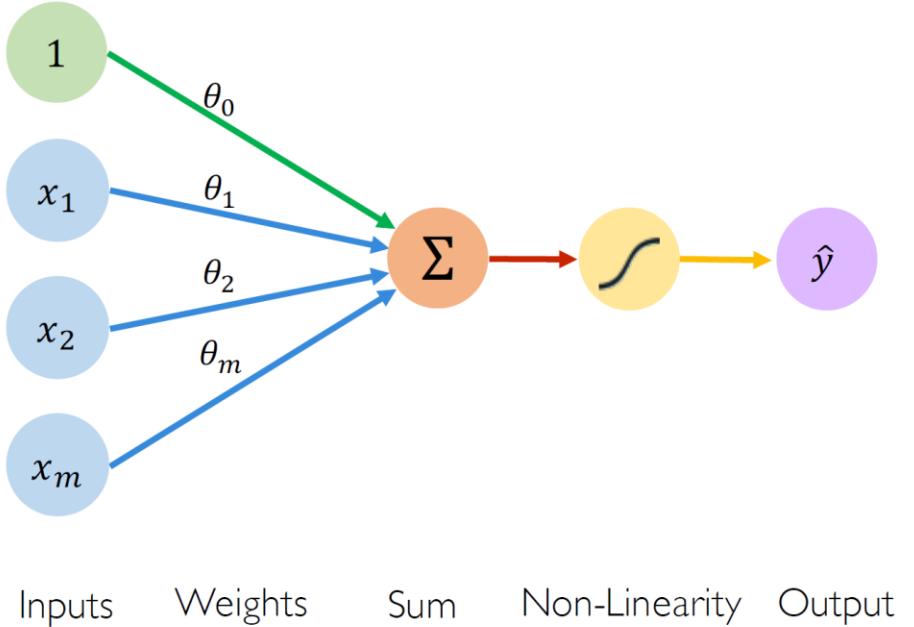
Output

Linear combination
of inputs

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

Non-linear
activation function

Perceptron: Forward Propagation



Linear combination of inputs

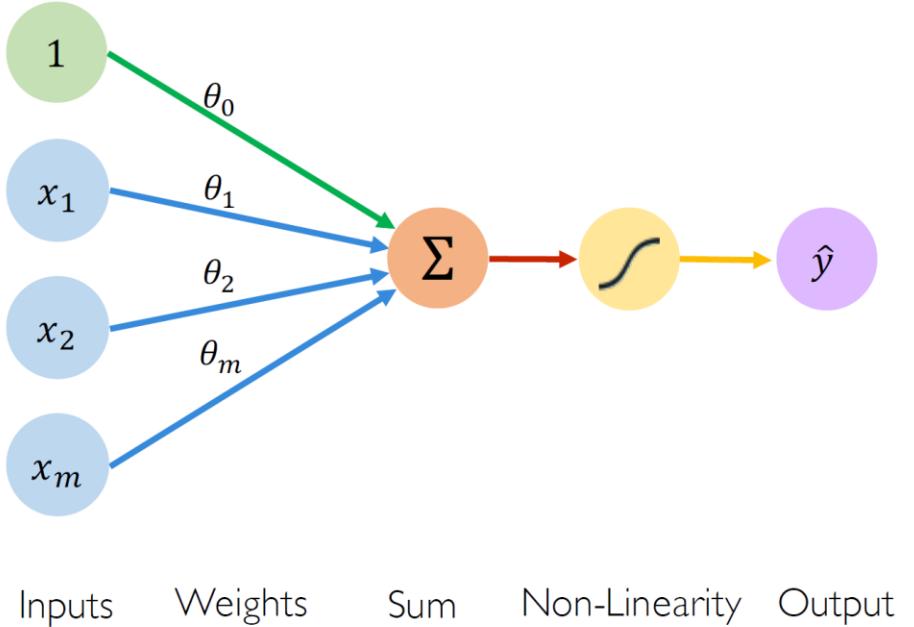
Output

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

Non-linear activation function

Bias

Perceptron: Forward Propagation

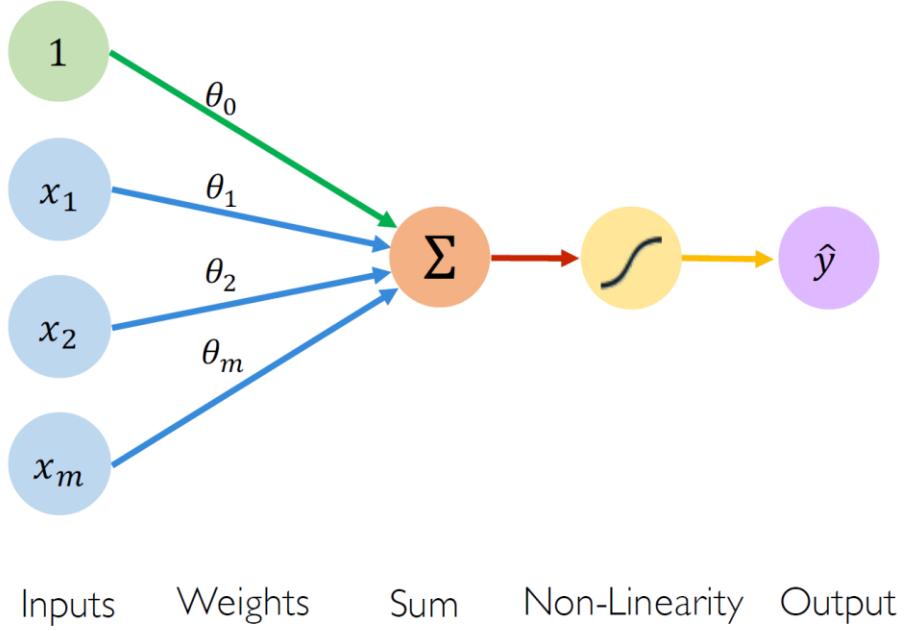


$$\hat{y} = g \left(\theta_0 + \sum_{i=1}^m x_i \theta_i \right)$$

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

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

Perceptron: Forward Propagation

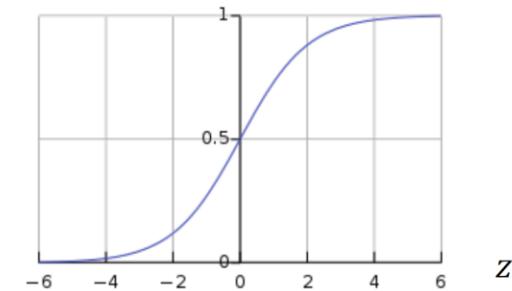


Activation Functions

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

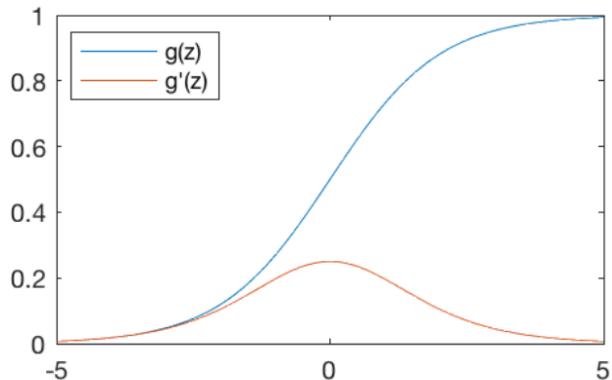
- Example: sigmoid function

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



Common Activation Functions

Sigmoid Function

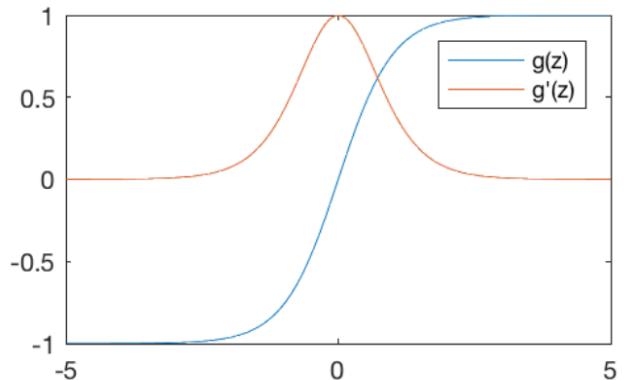


$$g(z) = \frac{1}{1 + e^{-z}}$$

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

 `tf.nn.sigmoid(z)`

Hyperbolic Tangent

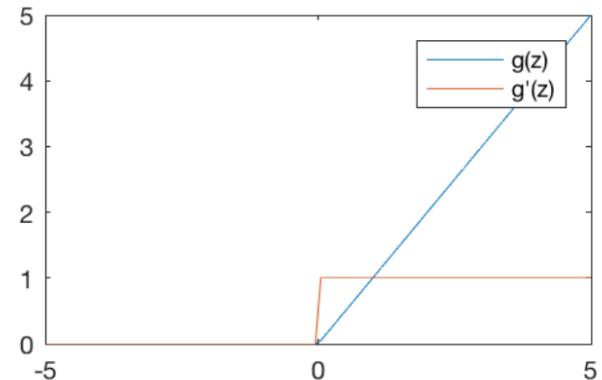


$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

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

 `tf.nn.tanh(z)`

Rectified Linear Unit (ReLU)



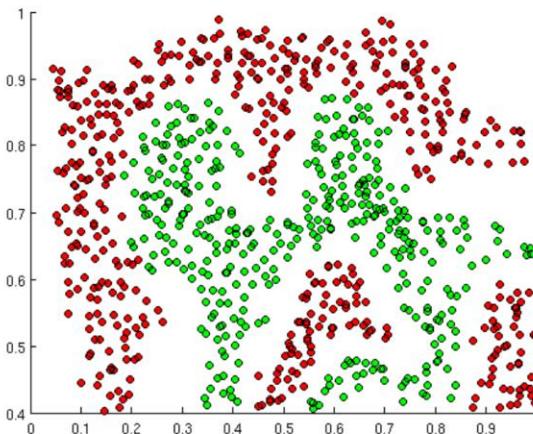
$$g(z) = \max(0, z)$$

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

 `tf.nn.relu(z)`

Importance of Activation Functions

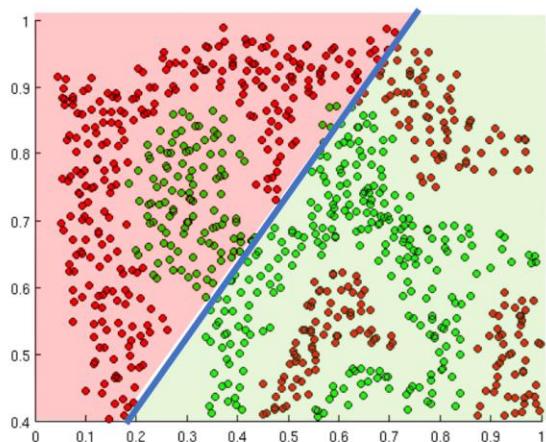
The purpose of activation functions is to **introduce non-linearities** into the network



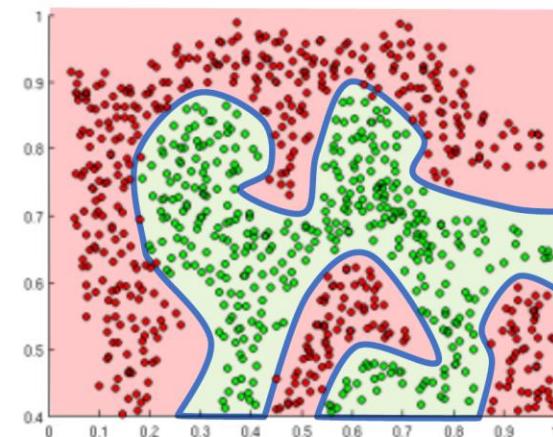
What if we wanted to build a Neural Network to
distinguish green vs red points?

Importance of Activation Functions

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

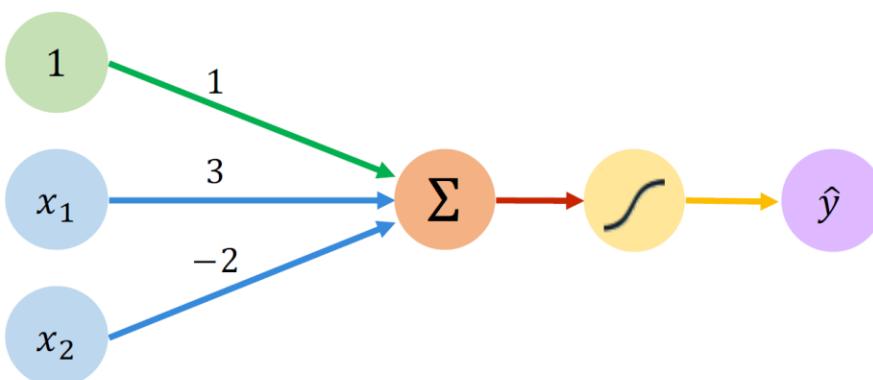


Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

Perceptron: Example

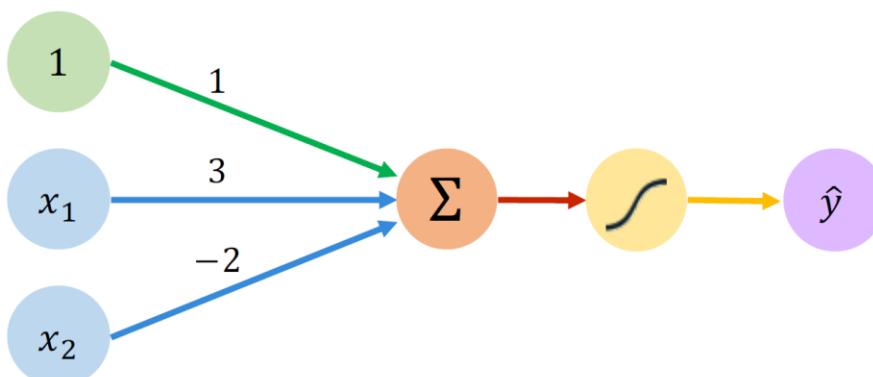


We have: $\theta_0 = 1$ and $\boldsymbol{\theta} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

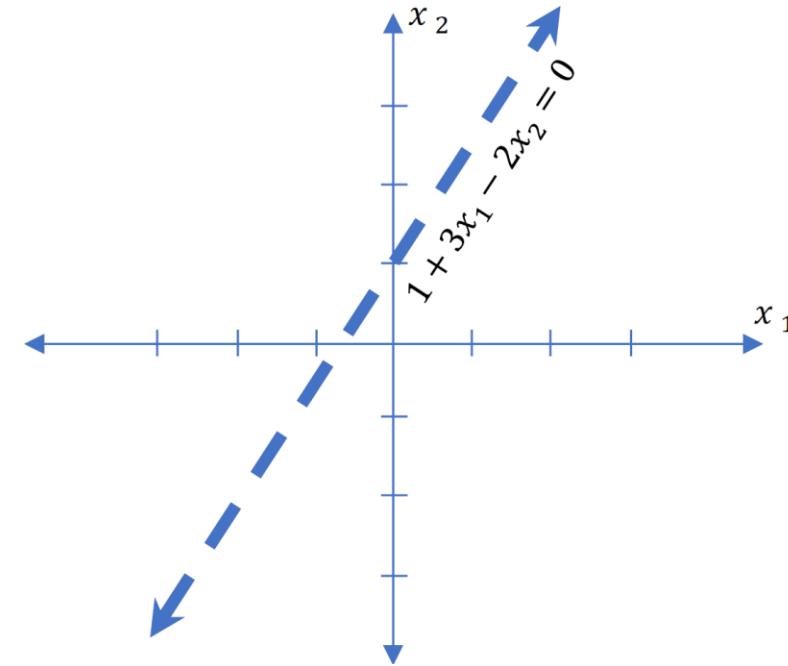
$$\begin{aligned}\hat{y} &= g(\theta_0 + \mathbf{X}^T \boldsymbol{\theta}) \\ &= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \\ \hat{y} &= g\left(1 + 3x_1 - 2x_2\right)\end{aligned}$$

This is just a line in 2D!

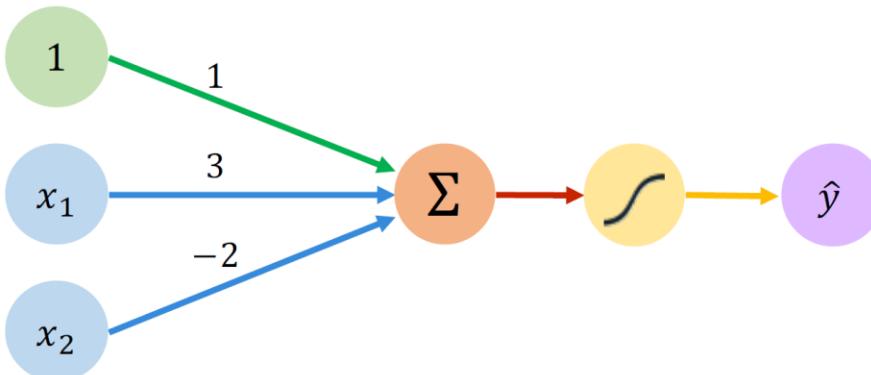
Perceptron: Example



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



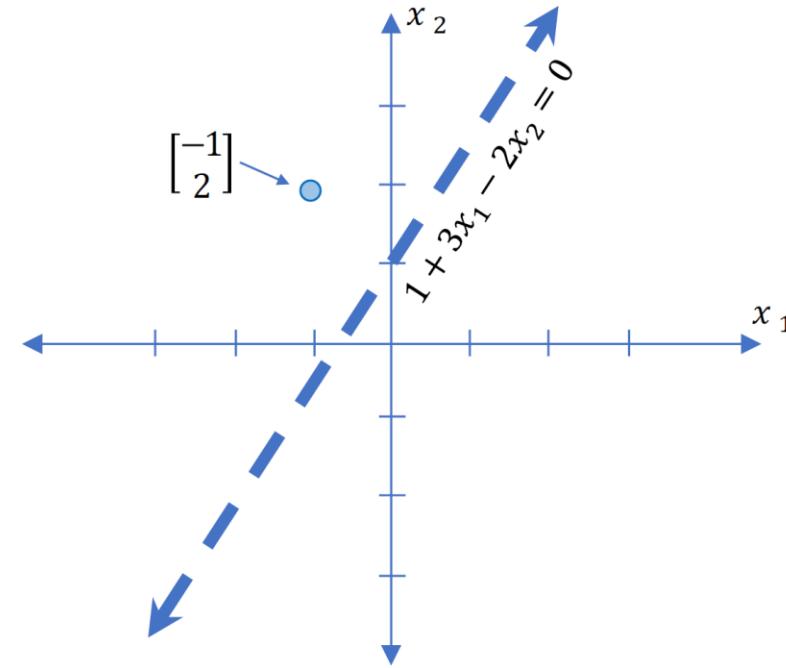
Perceptron: Example



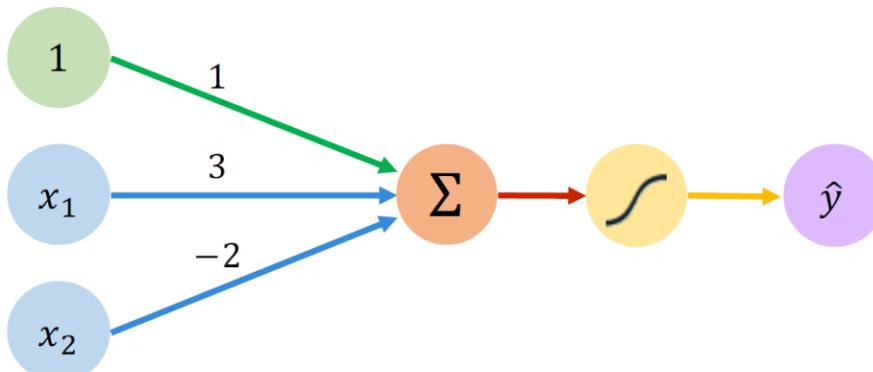
Assume we have input: $X = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

$$\begin{aligned}\hat{y} &= g(1 + (3 * -1) - (2 * 2)) \\ &= g(-6) \approx 0.002\end{aligned}$$

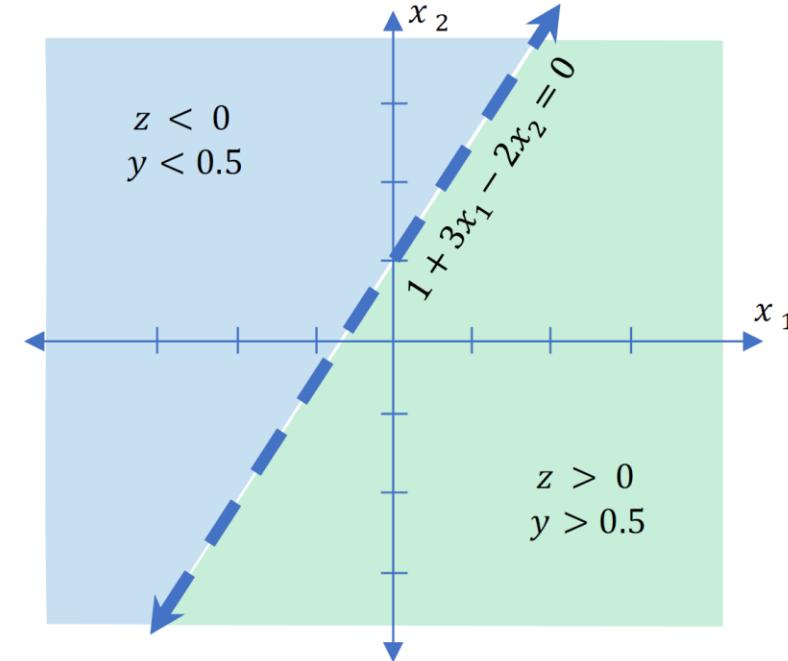
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



Perceptron: Example

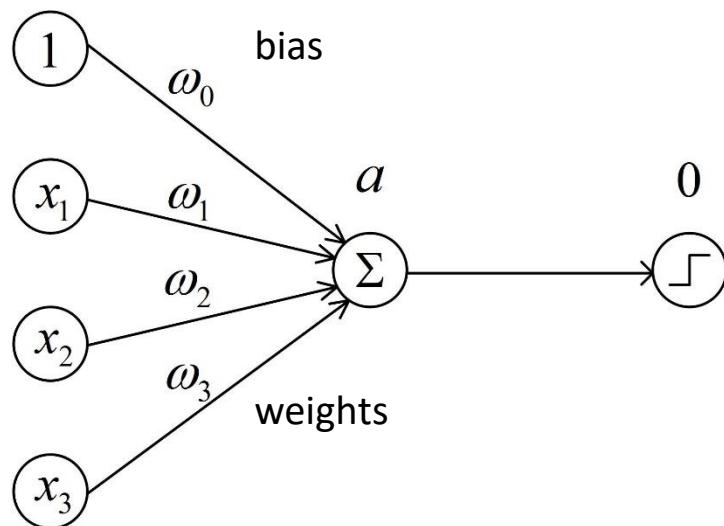


$$\hat{y} = g(1 + 3x_1 - 2x_2)$$

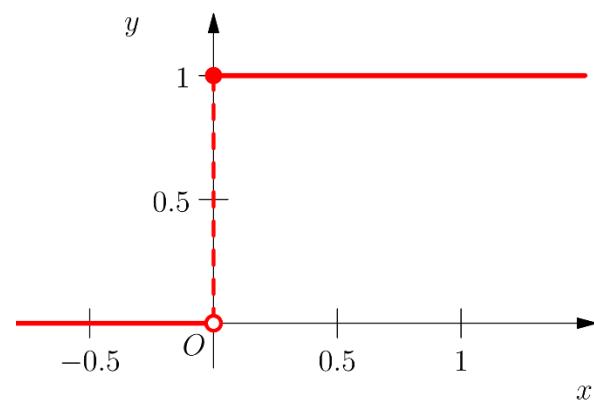


Artificial Neural Networks: Perceptron

- 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



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

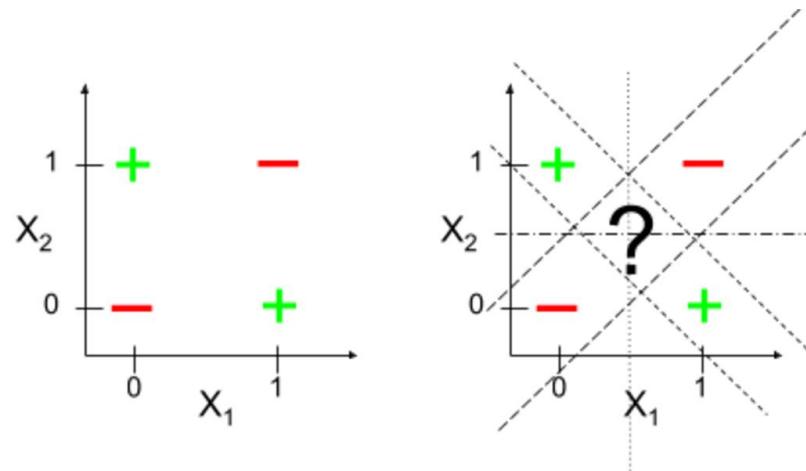


- A step function is not differentiable
- One layer is often not enough

XOR Problem

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

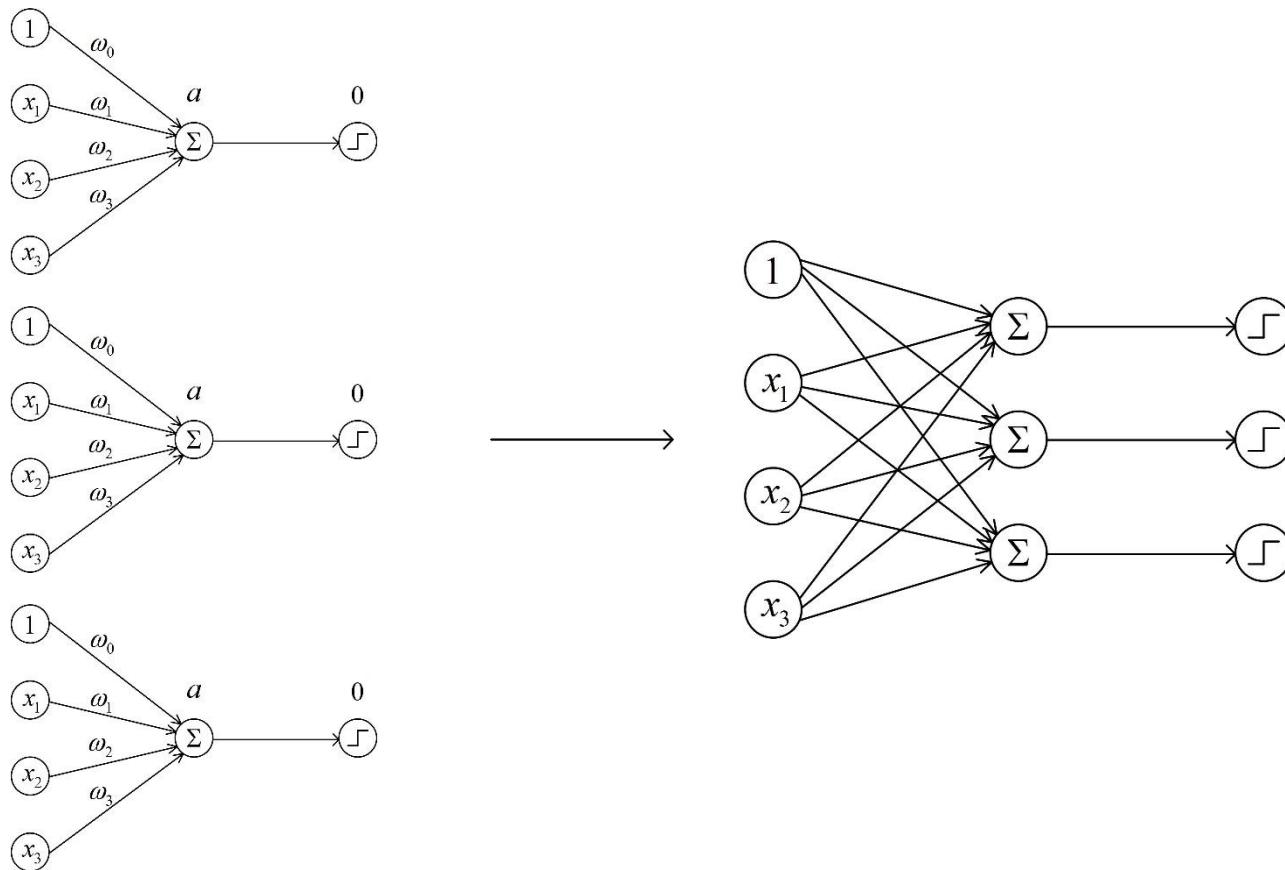
x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0



- Single neuron = one linear classification boundary

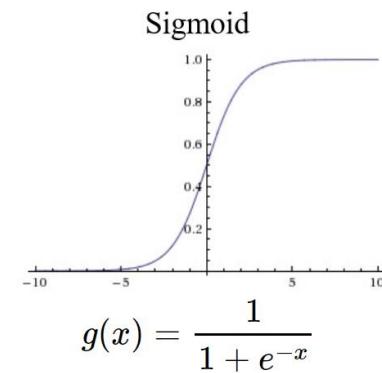
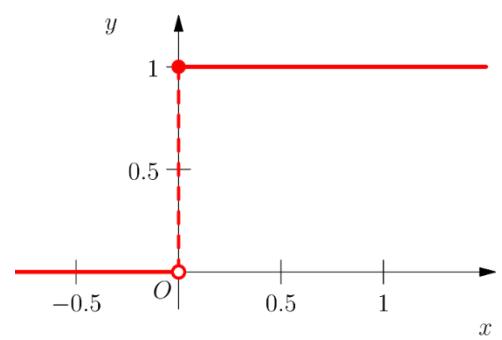
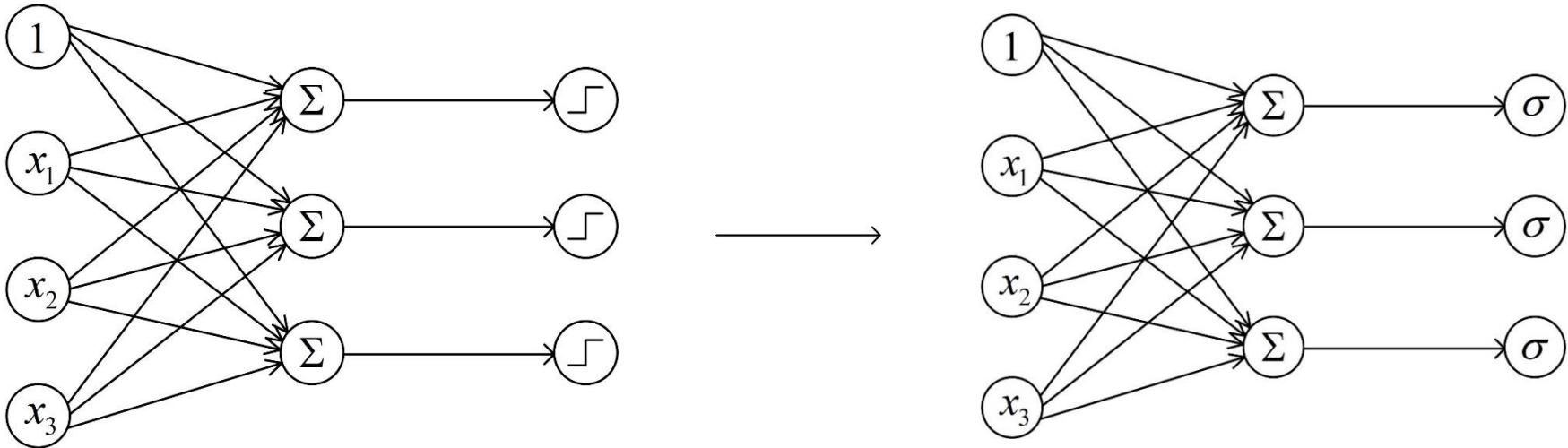
Artificial Neural Networks: MLP

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



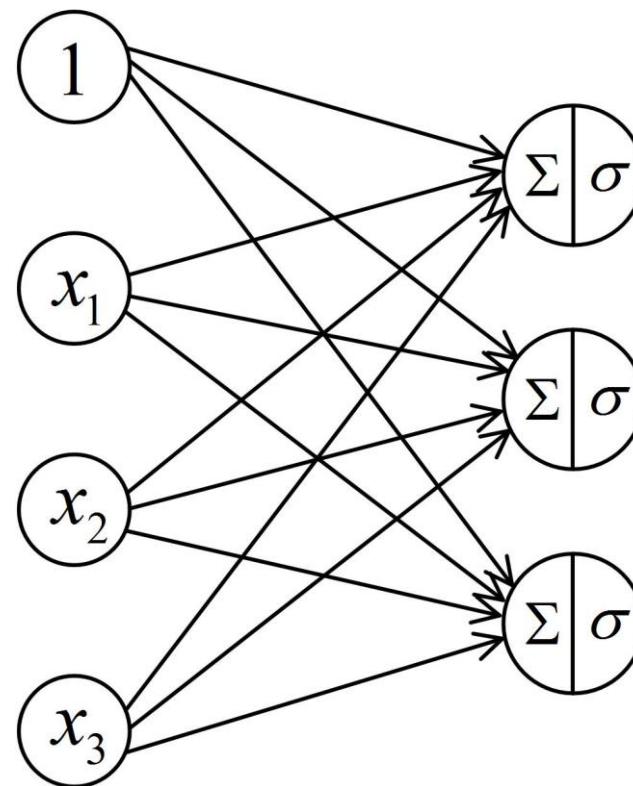
Artificial Neural Networks: Activation Func.

- Differentiable non-linear activation function



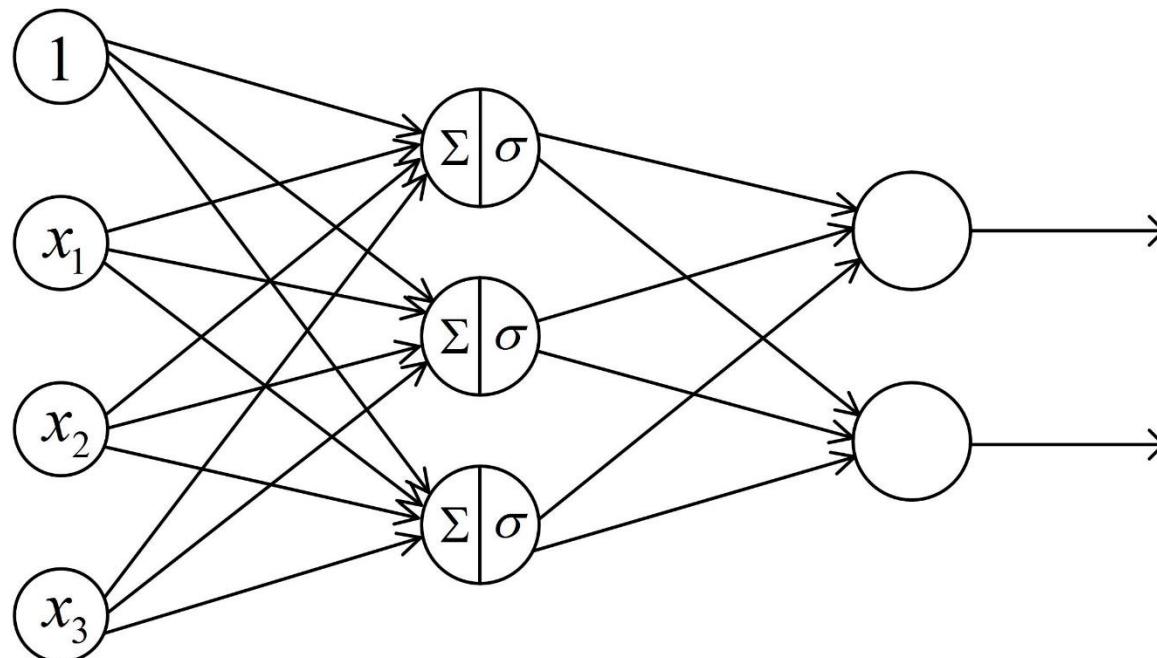
Artificial Neural Networks

- In a compact representation



Artificial Neural Networks

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



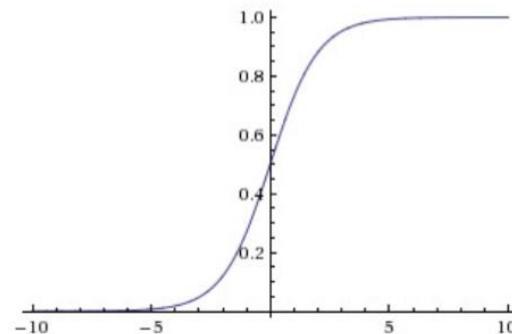
ANN: Transformation

- Affine (or linear) transformation and nonlinear activation layer (notations are mixed:
 $g = \sigma, \omega = \theta, \omega_0 = b$)

$$o(x) = g(\theta^T x + b)$$

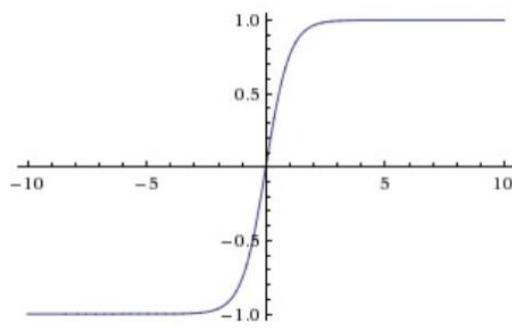
- Nonlinear activation functions ($g = \sigma$)

Sigmoid



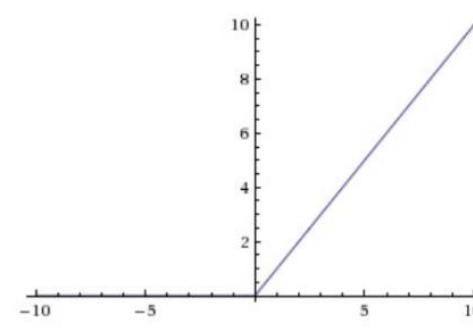
$$g(x) = \frac{1}{1 + e^{-x}}$$

tanh



$$g(x) = \tanh(x)$$

Rectified Linear Unit

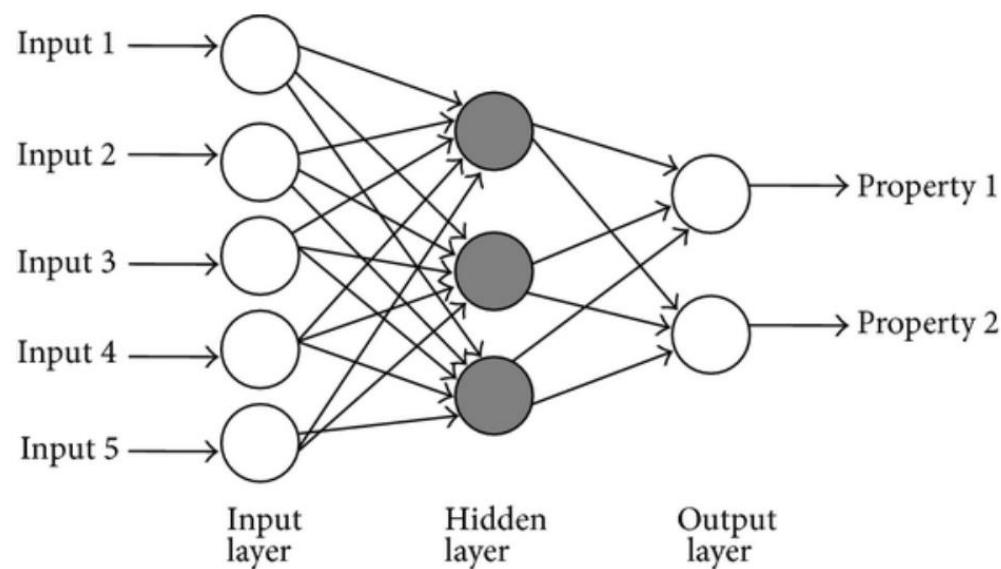


$$g(x) = \max(0, x)$$

ANN: Architecture

- A single layer is not enough to be able to represent complex relationship between input and output
⇒ perceptron with many layers and units

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



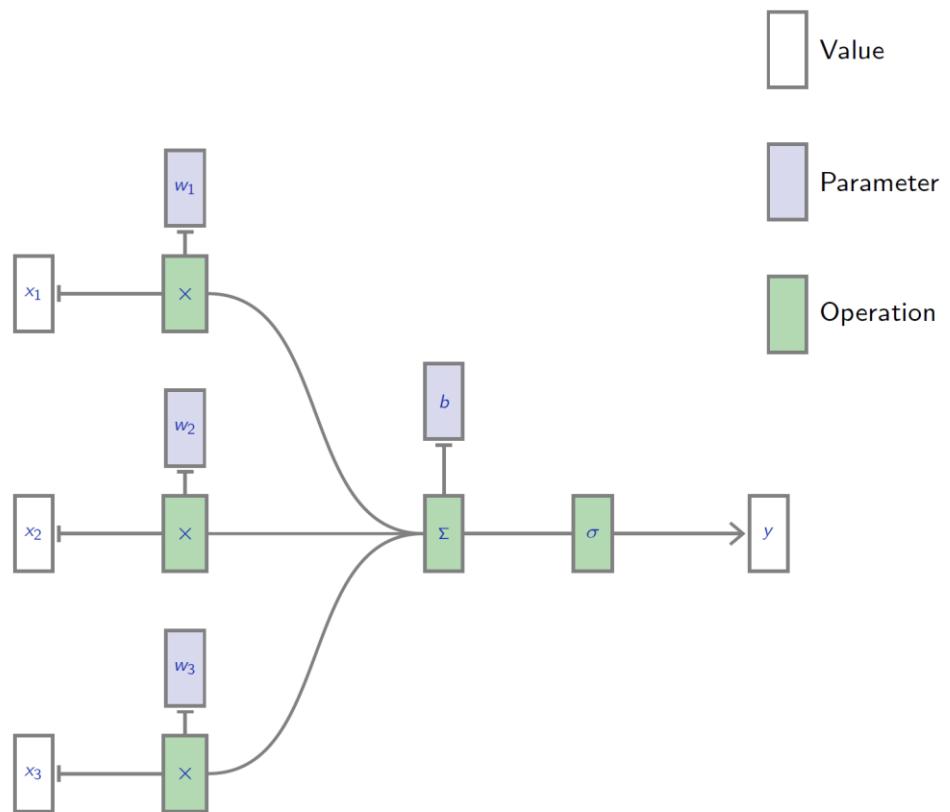
- The perceptron classification rule boils down to

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

- For neural networks, the function σ that follows a linear operator is called the activation function.

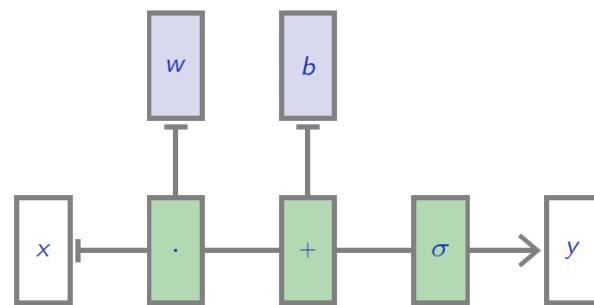
- We can also use tensor operations, as in

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

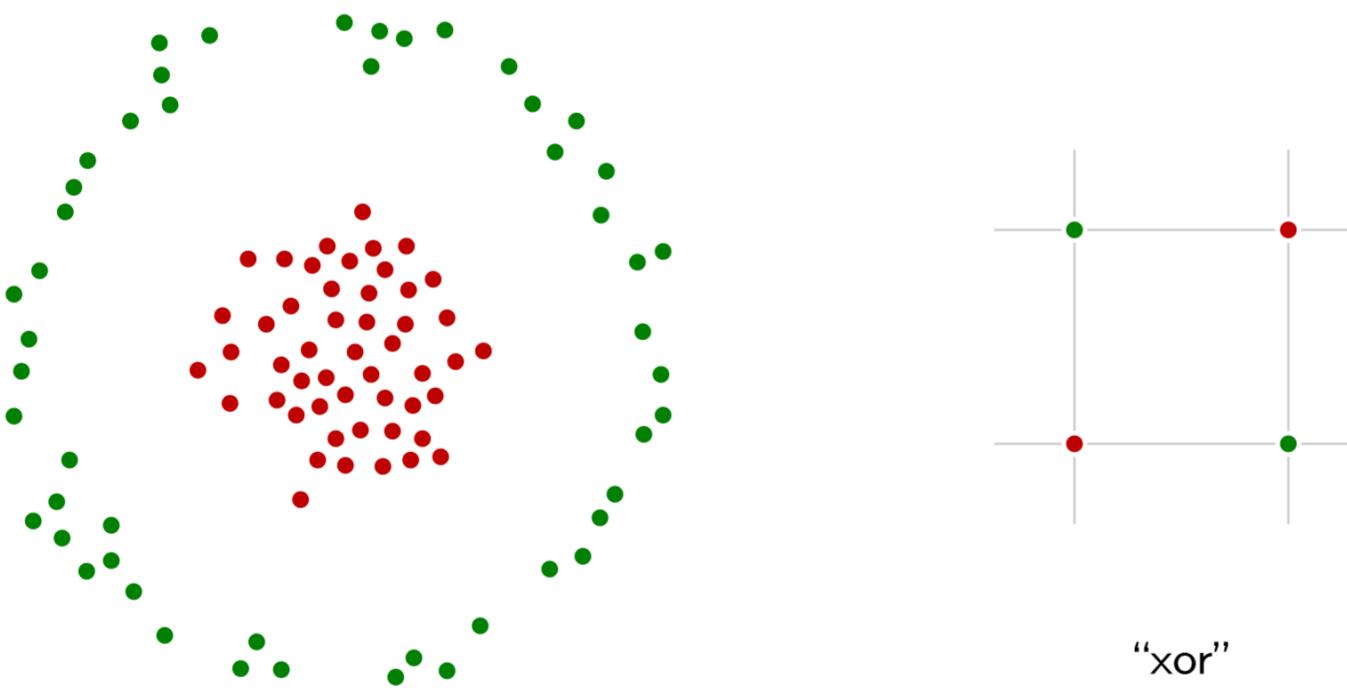


- We can represent this “neuron” as follows:

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

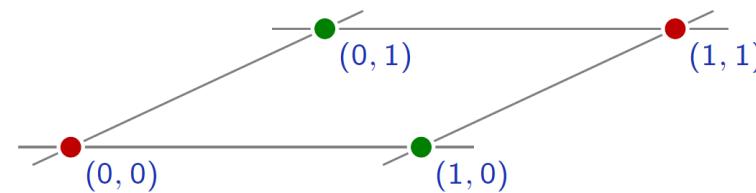


- The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be linearly separable.



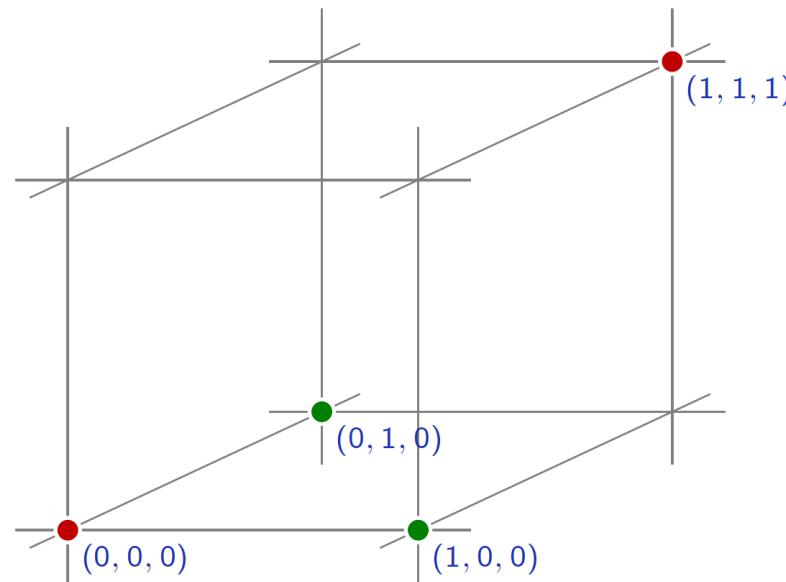
- The xor example can be solved by pre-processing the data to make the two populations linearly separable.

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



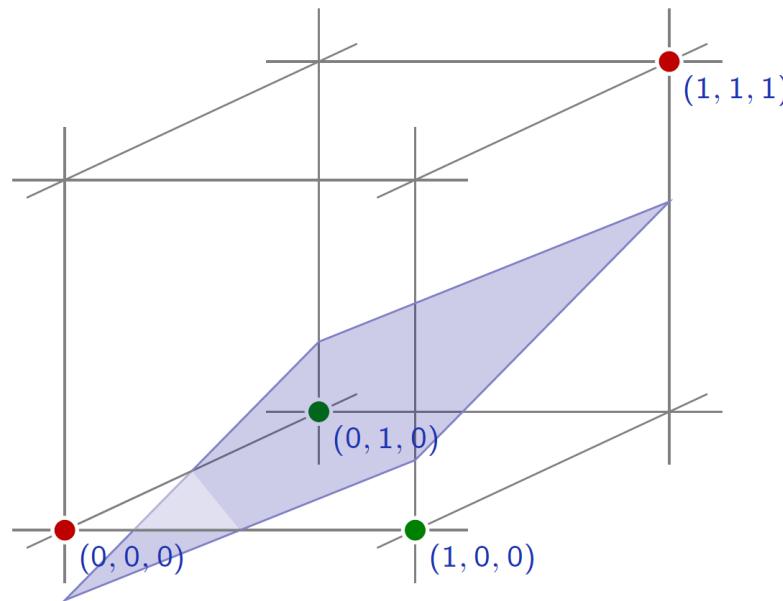
- The xor example can be solved by pre-processing the data to make the two populations linearly separable.

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



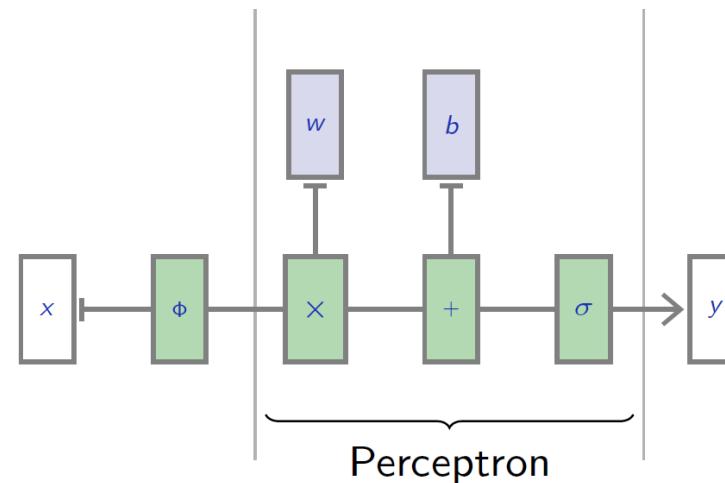
- The xor example can be solved by pre-processing the data to make the two populations linearly separable.

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

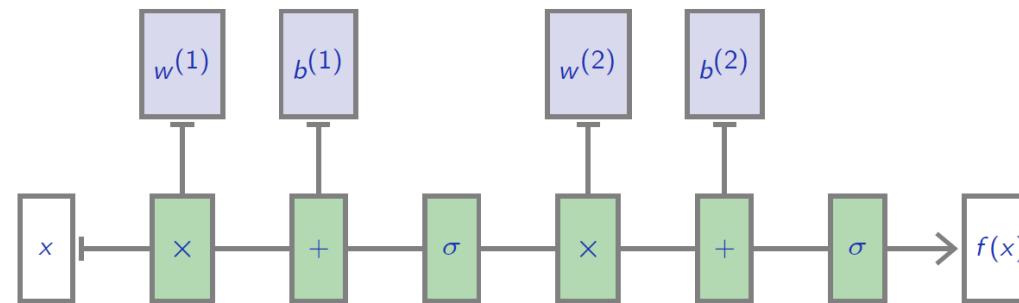


- Nonlinear mapping + neuron

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

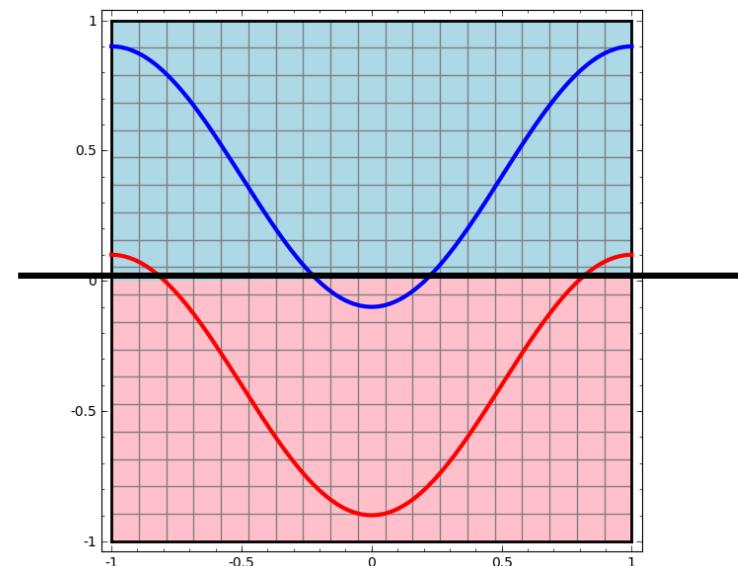
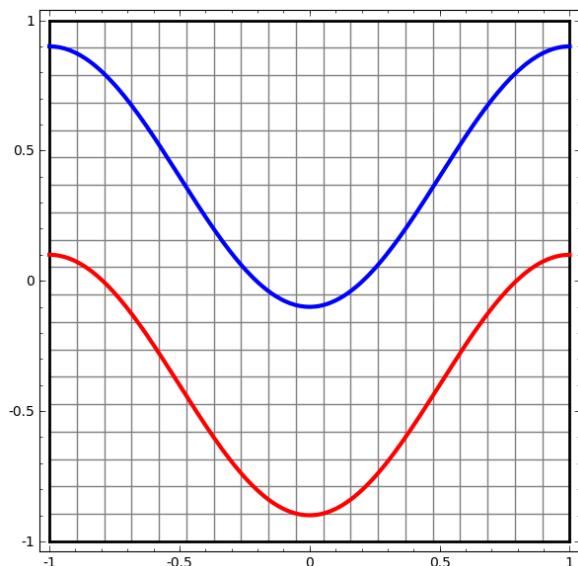


- Nonlinear mapping can be represented by another neurons
- We can generalize an MLP



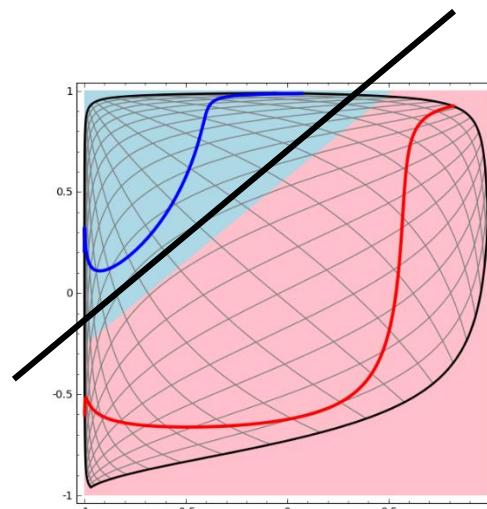
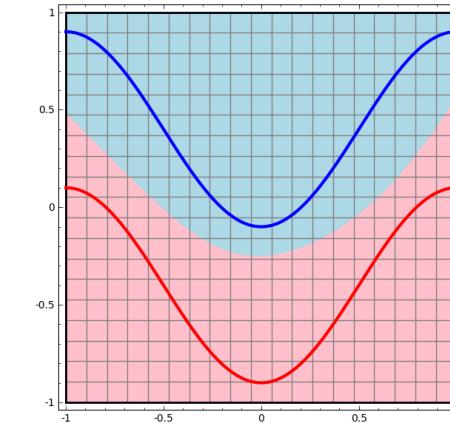
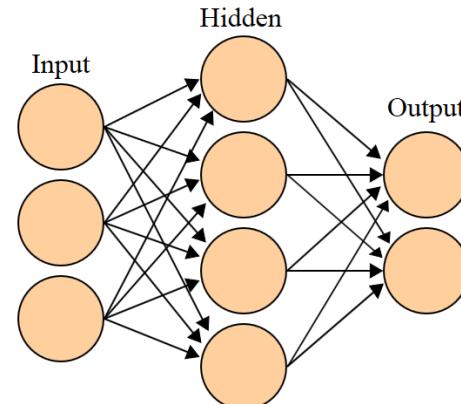
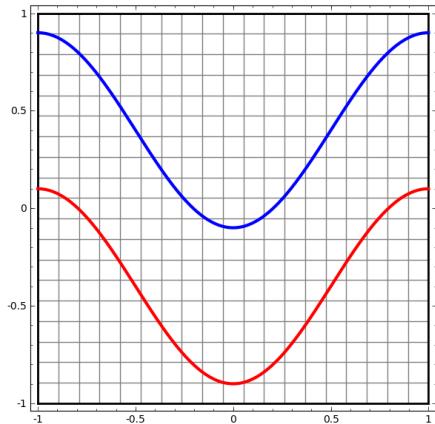
Linear Classifier

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



Neural Networks

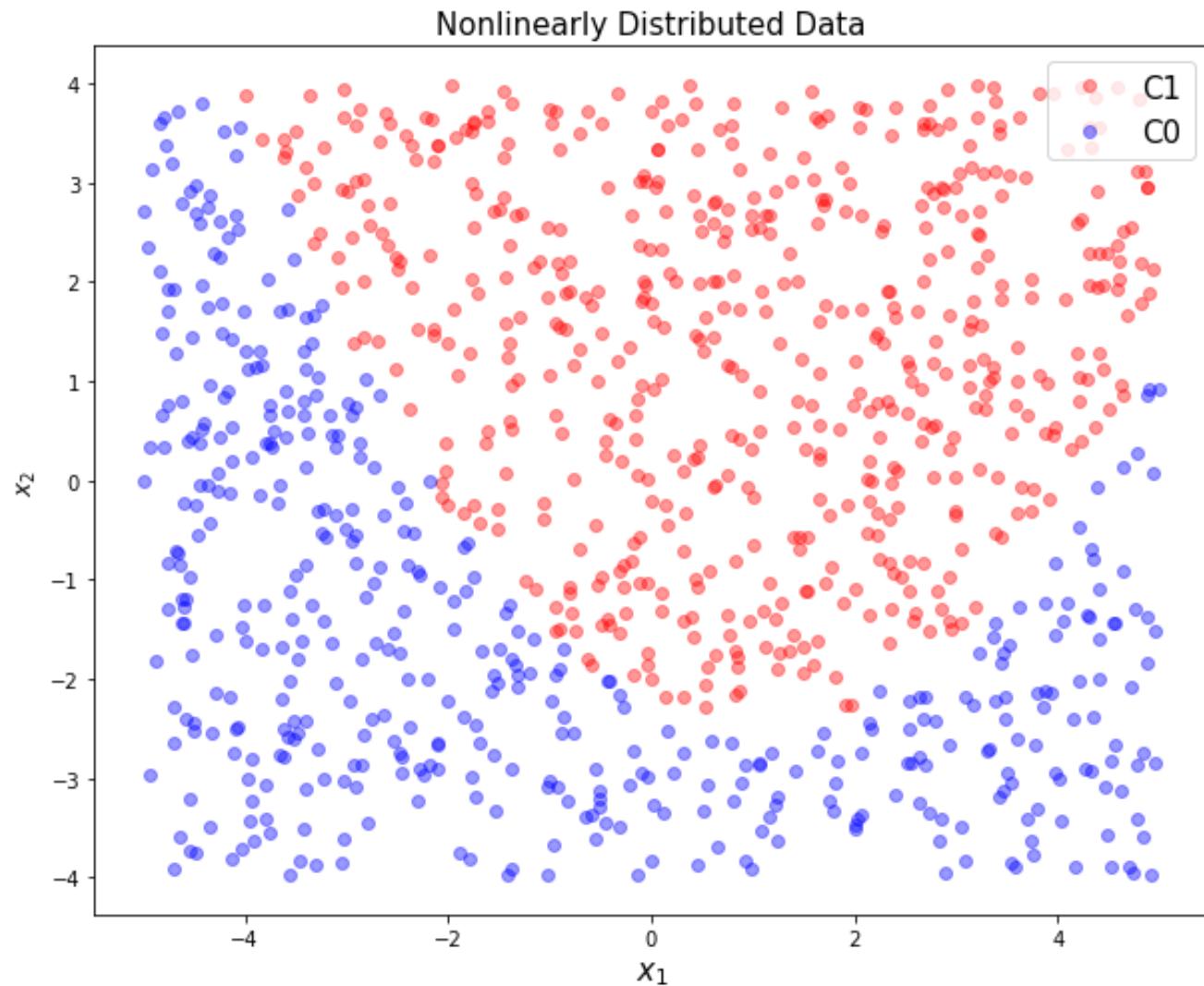
- The hidden layer learns a representation so that the data is linearly separable



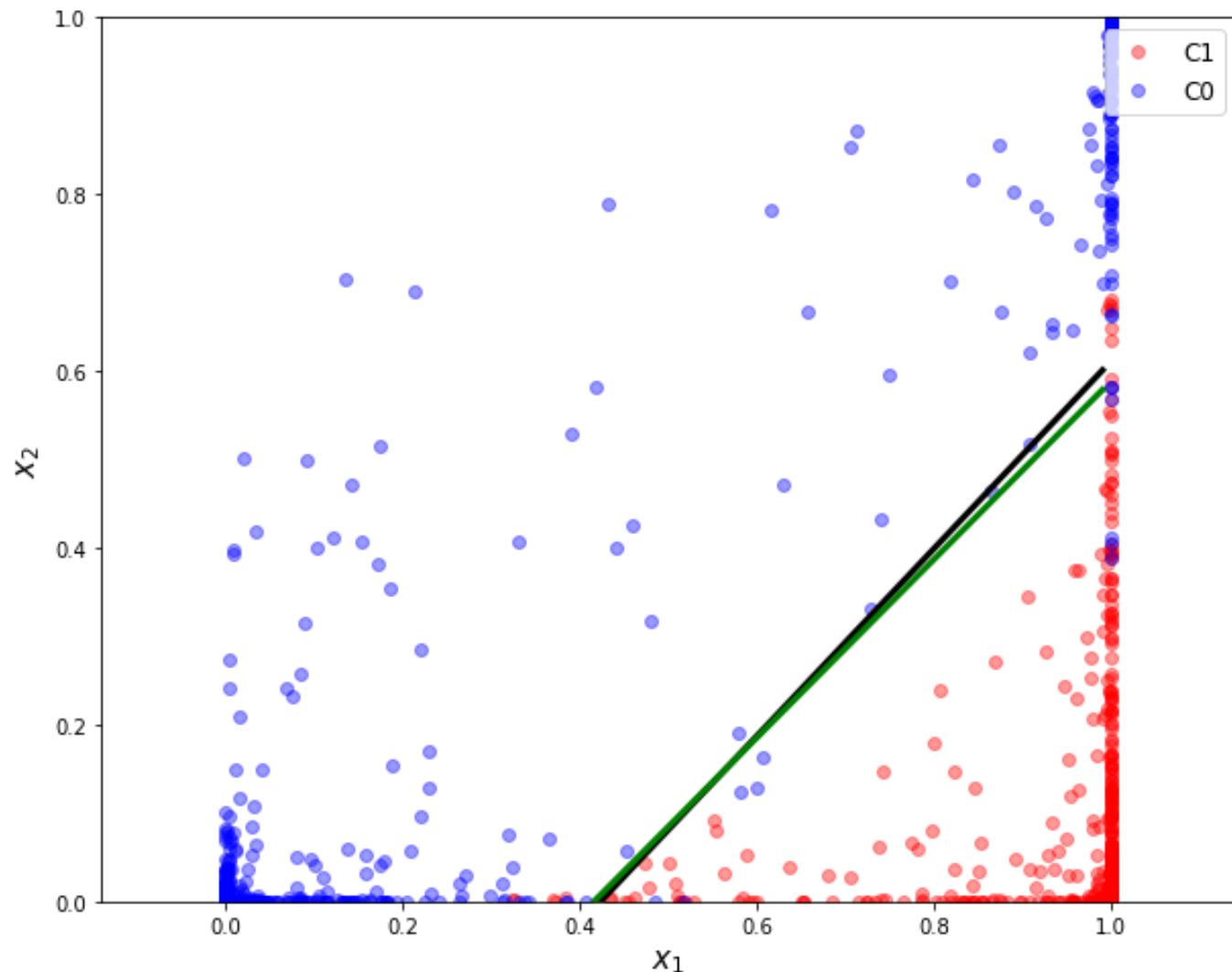
Understanding a Network's Behavior

- Understanding what is happening in a deep architectures after training is complex and the tools we have at our disposal are limited.
- We can look at
 - the network's parameters, filters as images,
 - internal activations as images,
 - distributions of activations on a population of samples,
 - derivatives of the response(s) wrt the input,
 - maximum-response synthetic samples,
 - adversarial samples.

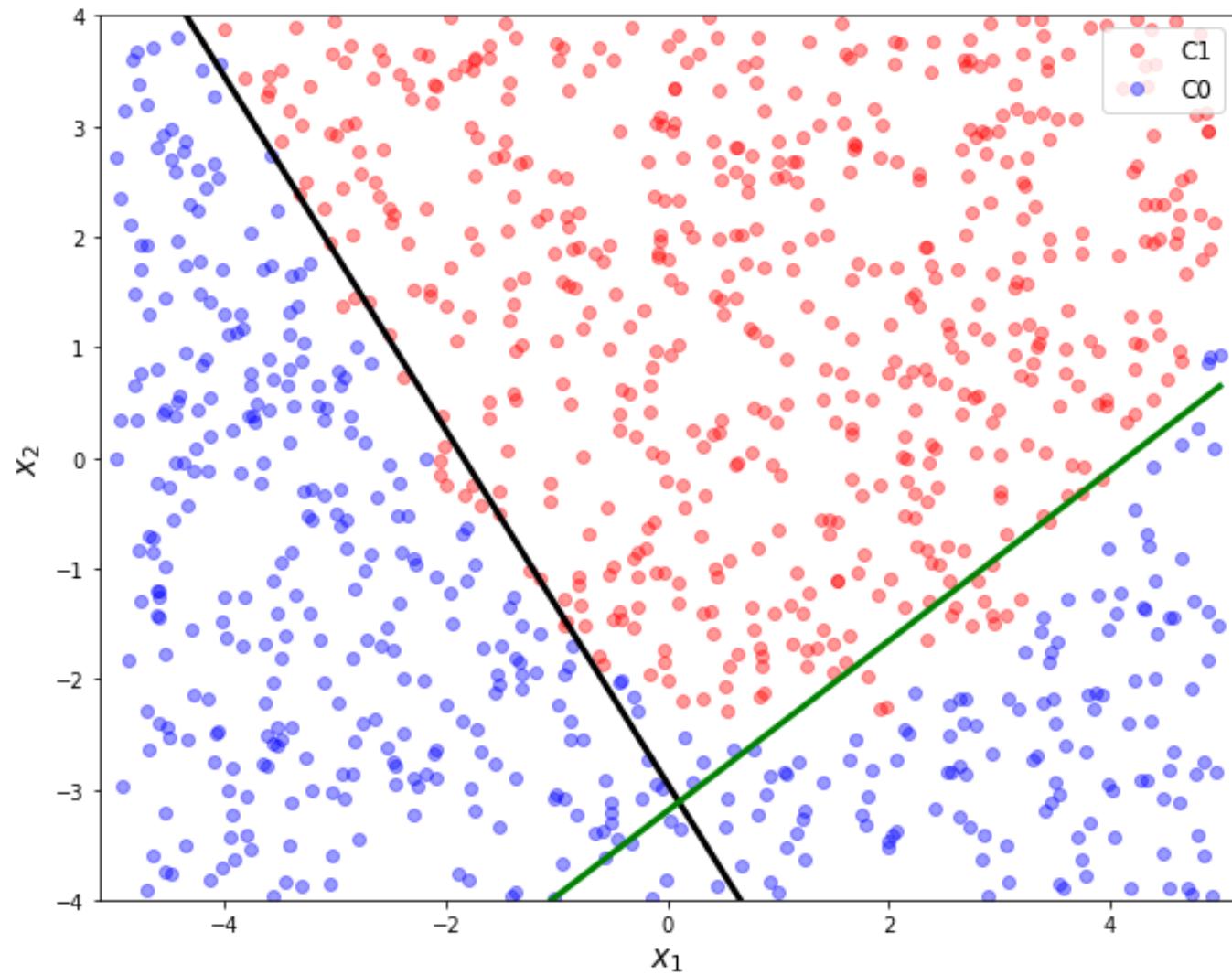
Nonlinearly Distributed Data



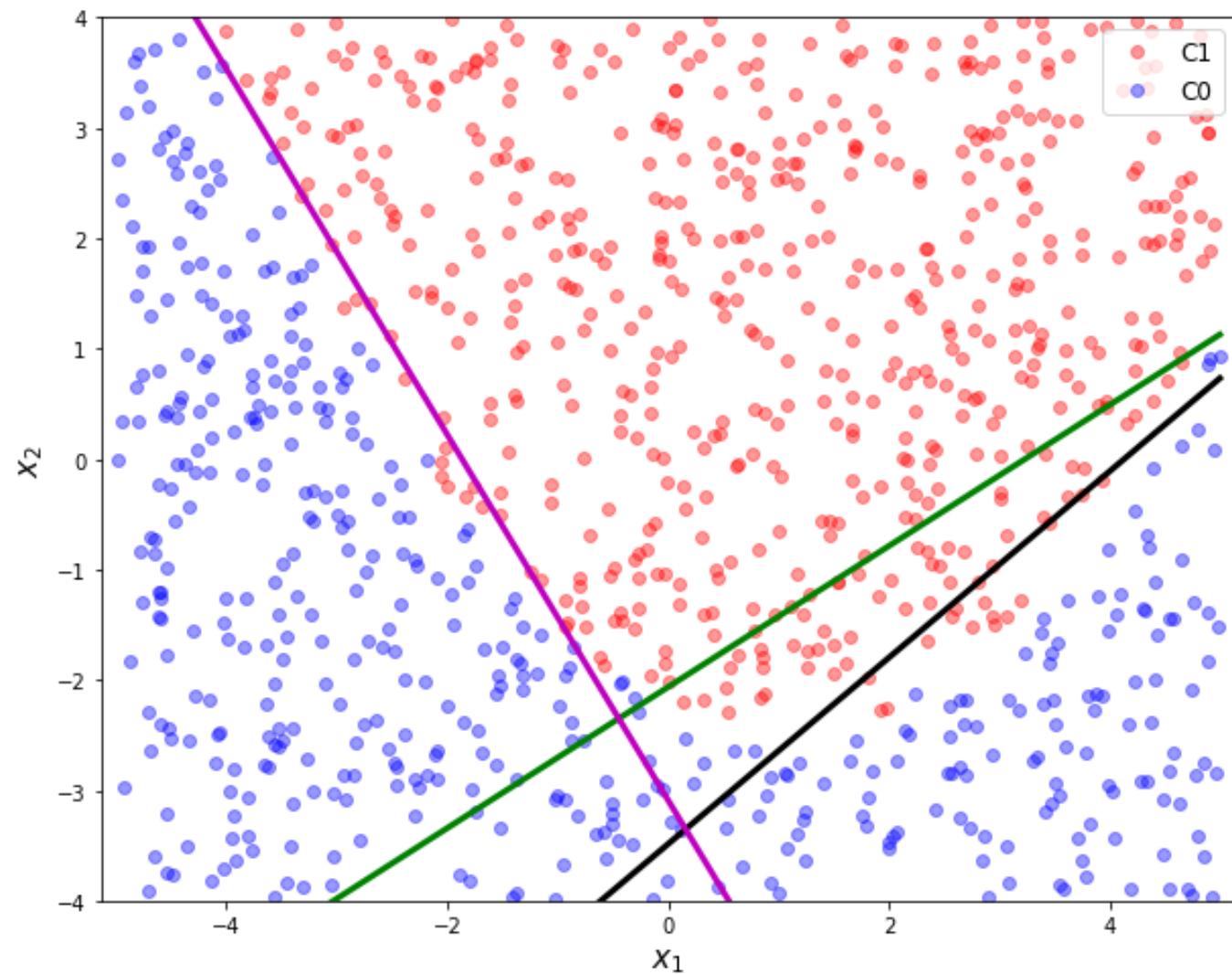
Multi Layers



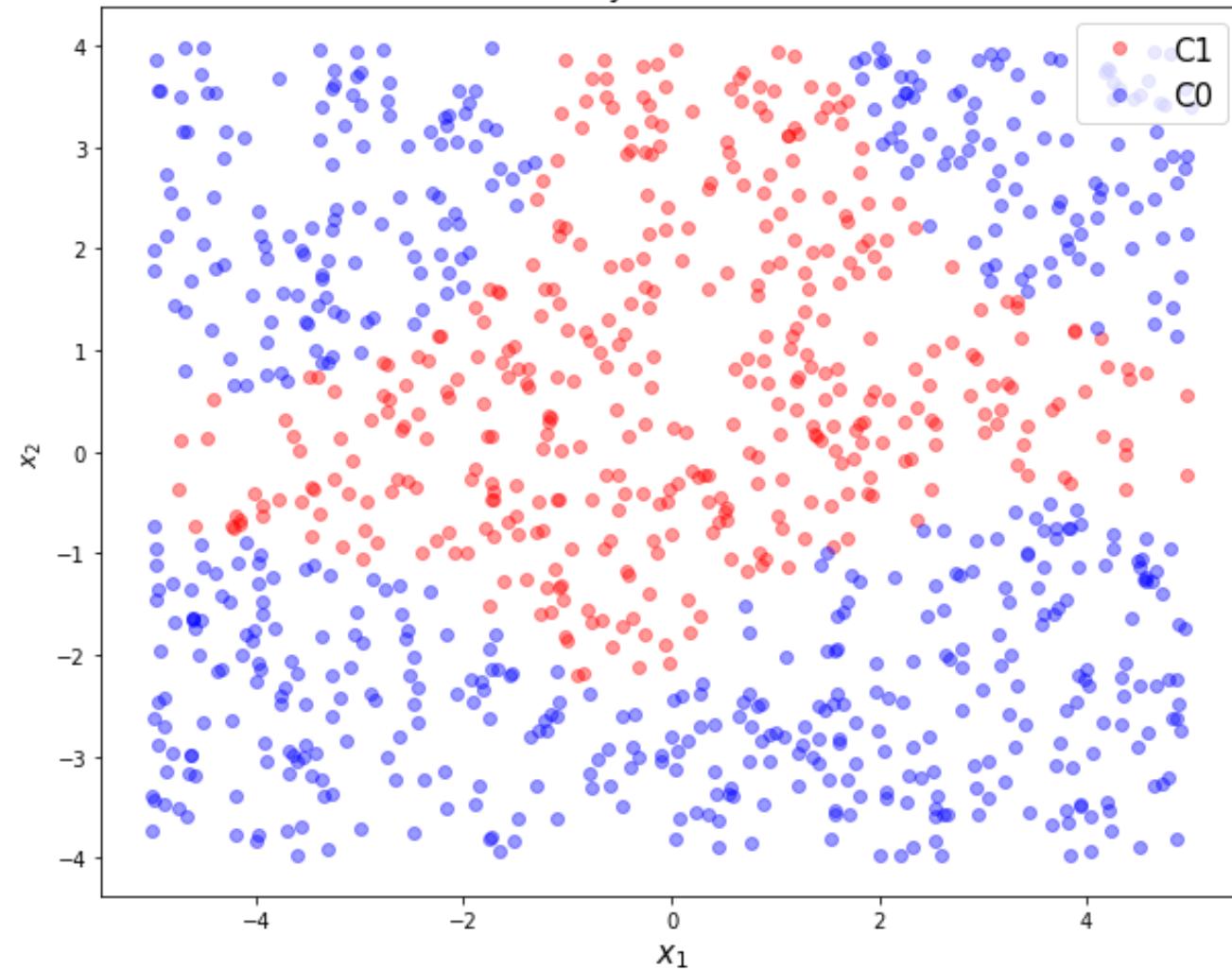
Multi Layers

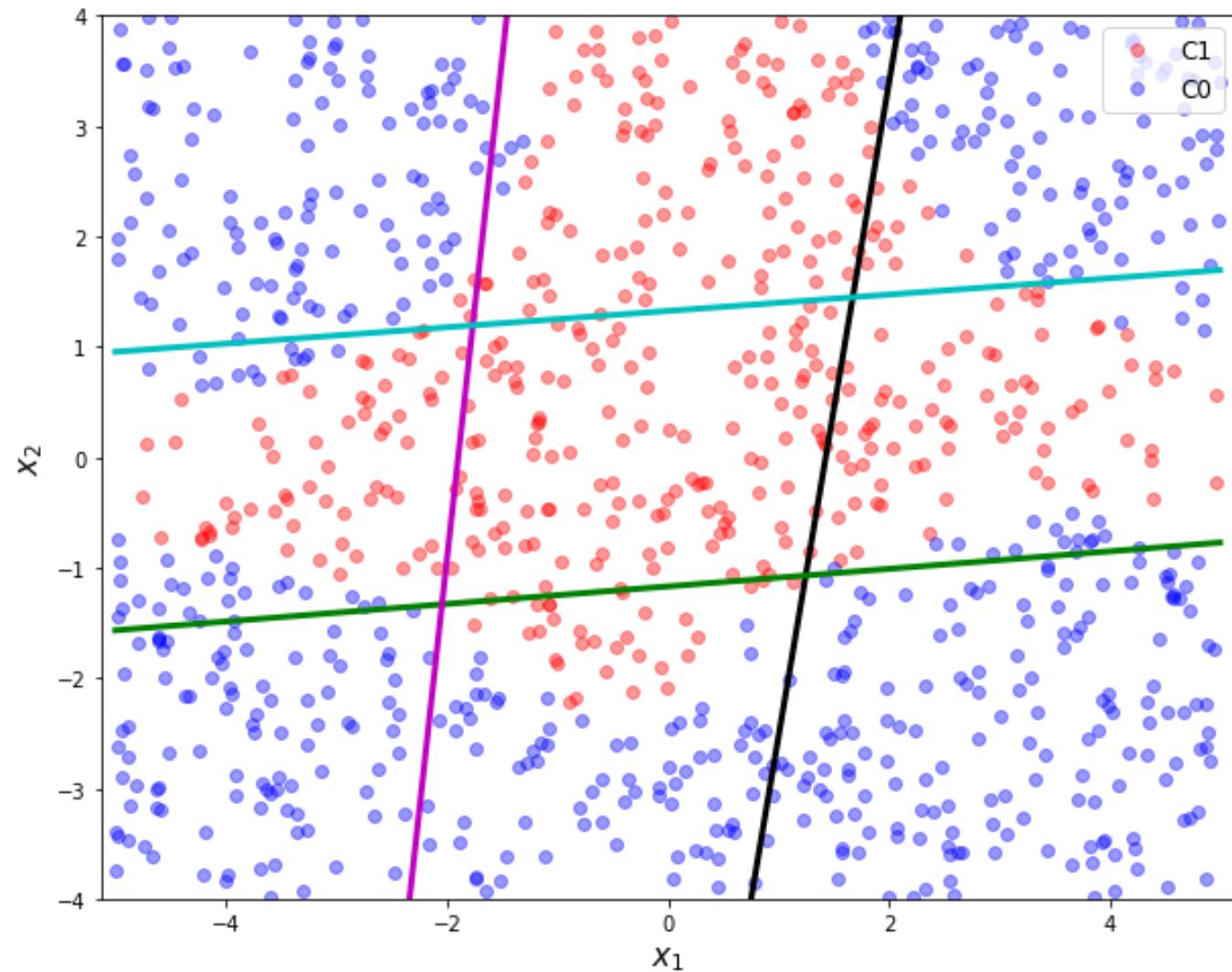


Multi Neurons

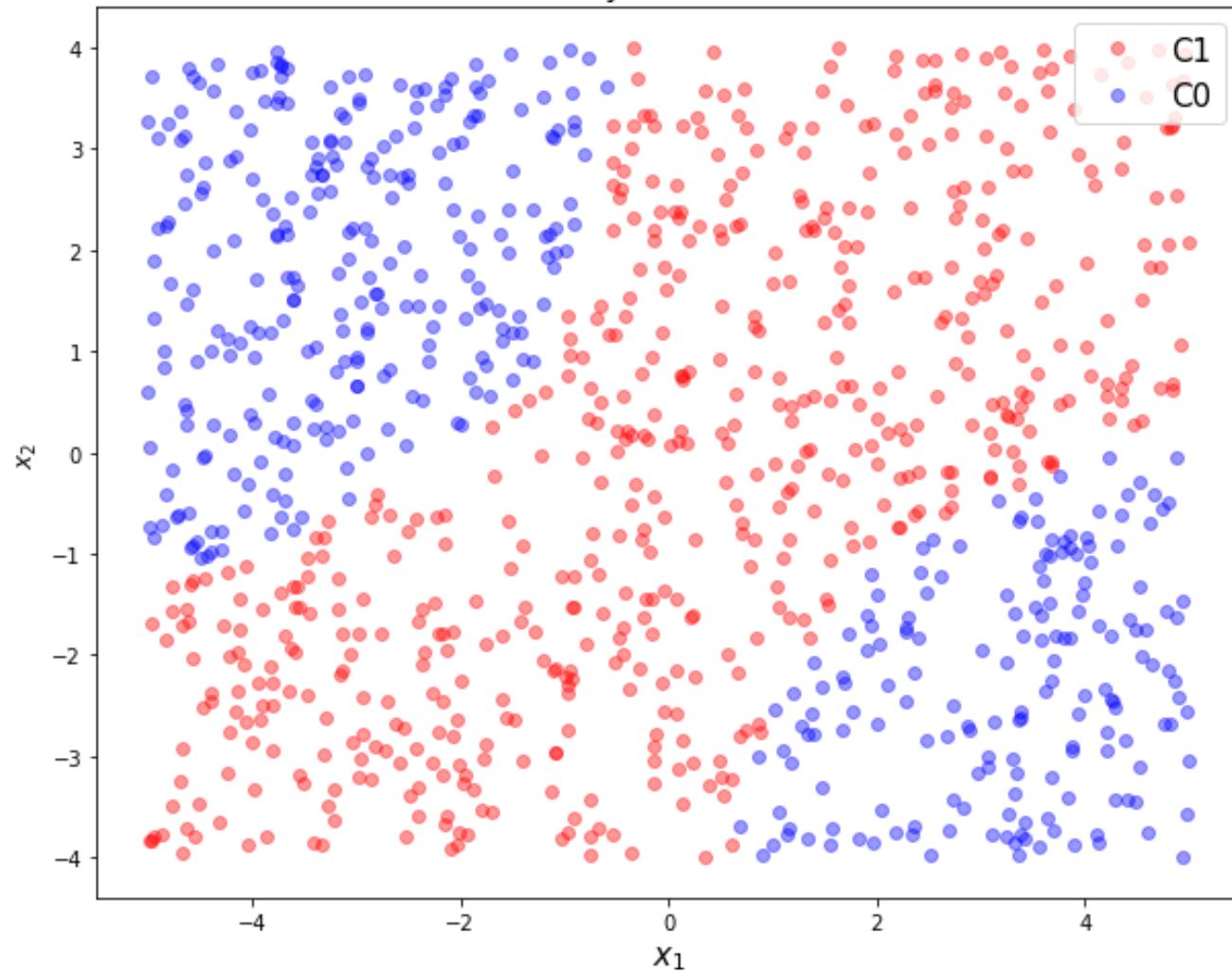


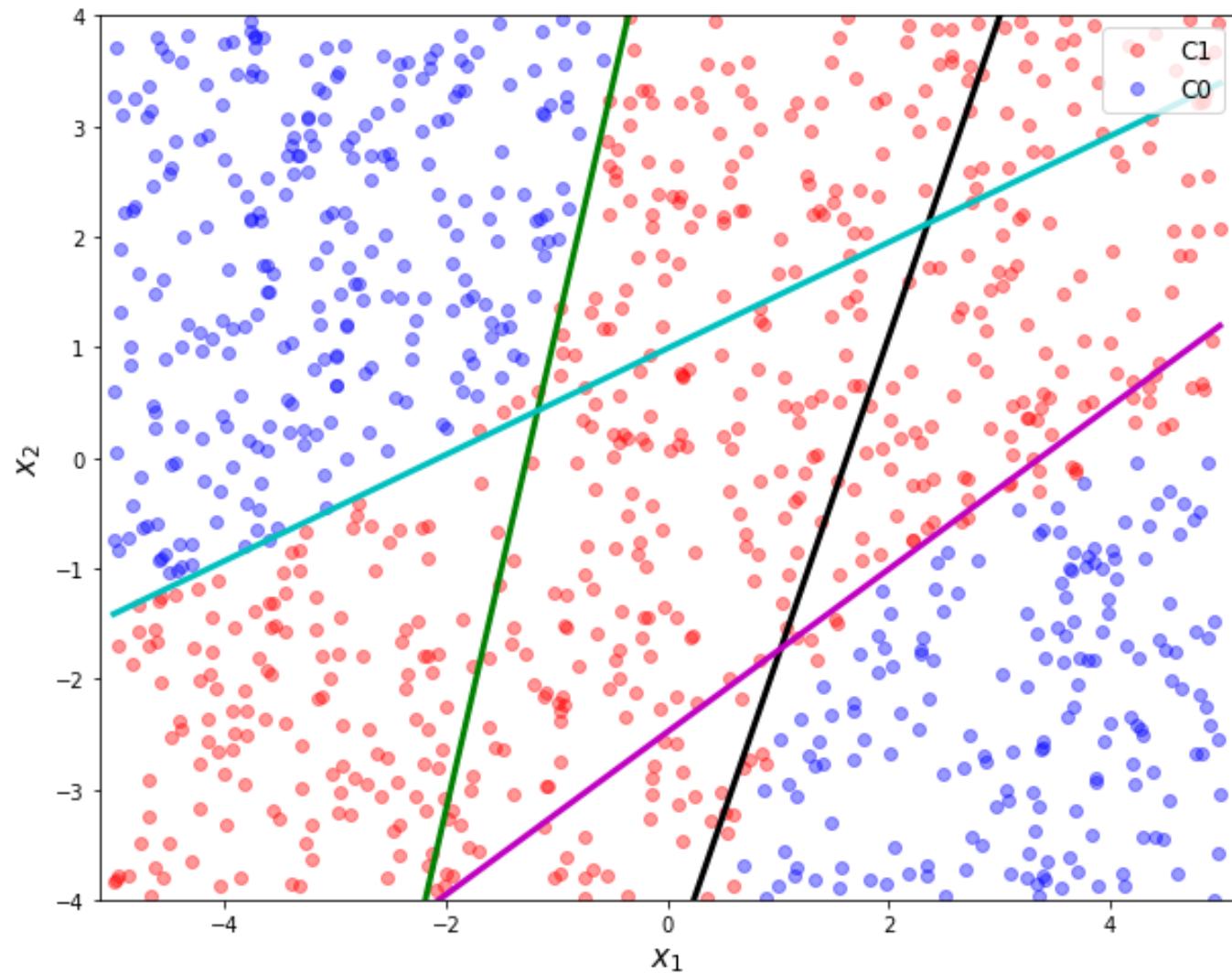
Nonlinearly Distributed Data





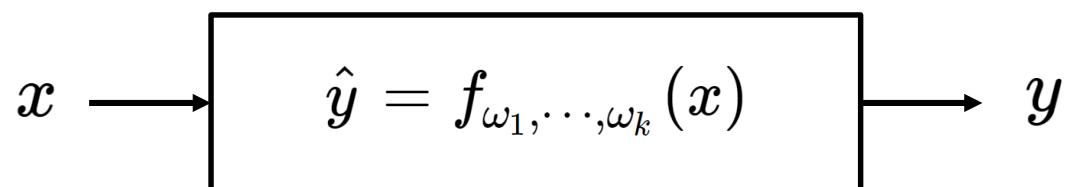
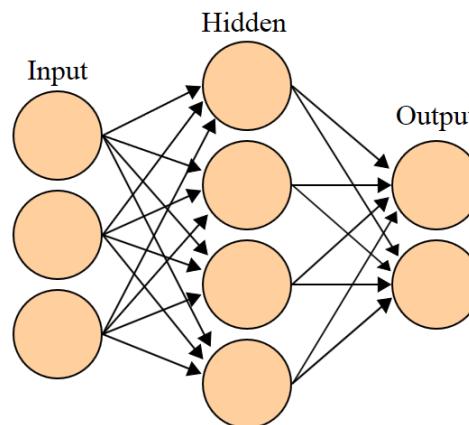
Nonlinearly Distributed Data



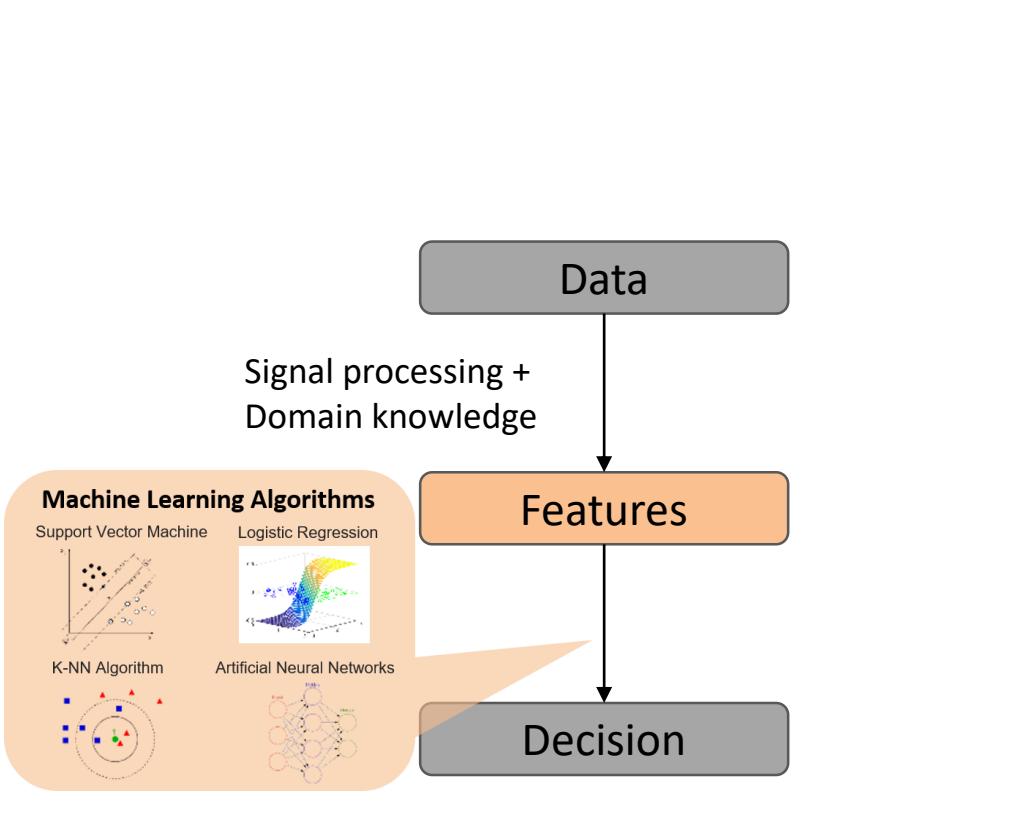


Summary

- Learning weights and biases from data using gradient descent

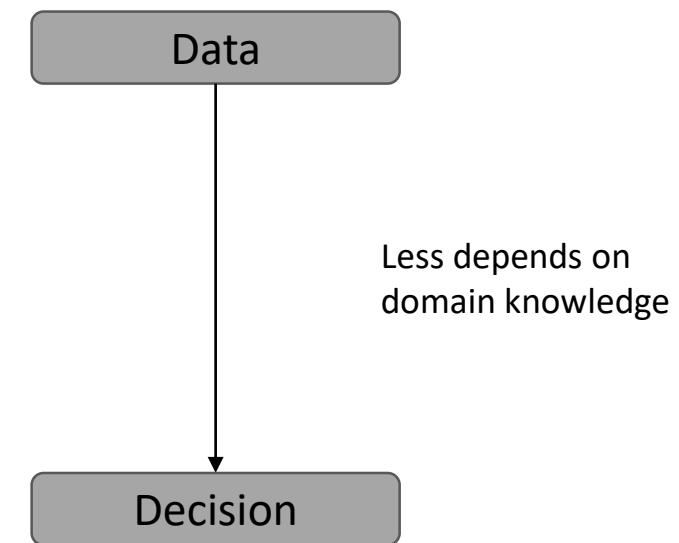
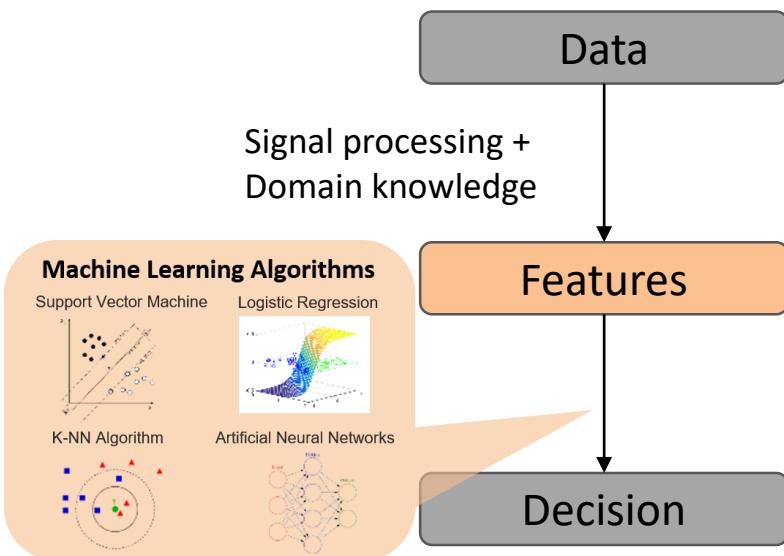


Machine Learning



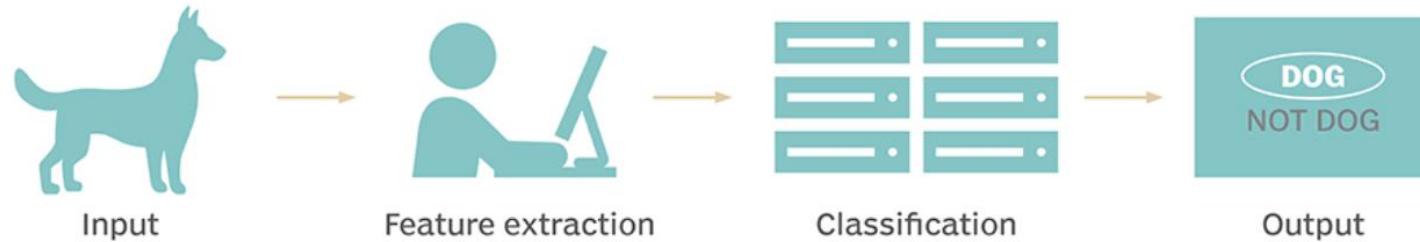
Machine Learning

Deep Learning

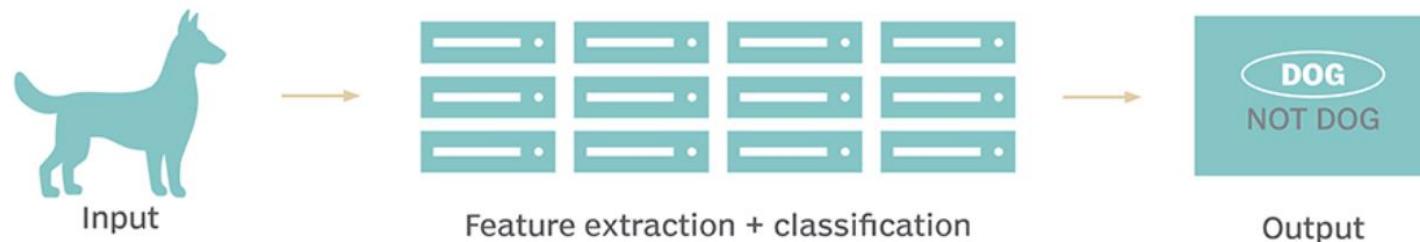


Recall Supervised Learning Setup

TRADITIONAL MACHINE LEARNING



DEEP LEARNING

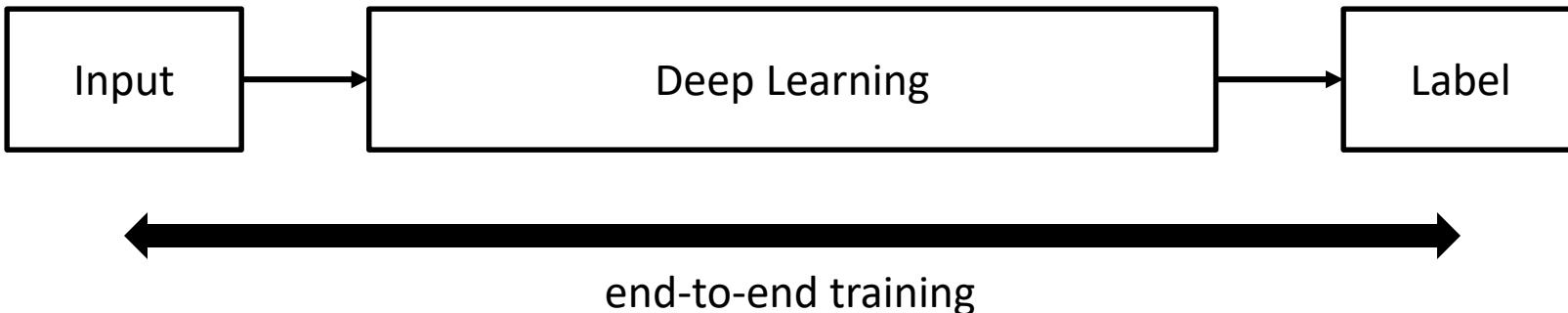


Machine Learning and Deep Learning

- Machine Learning



- Deep supervised learning

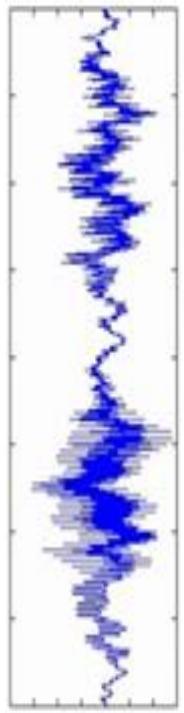


Artificial Neural Networks

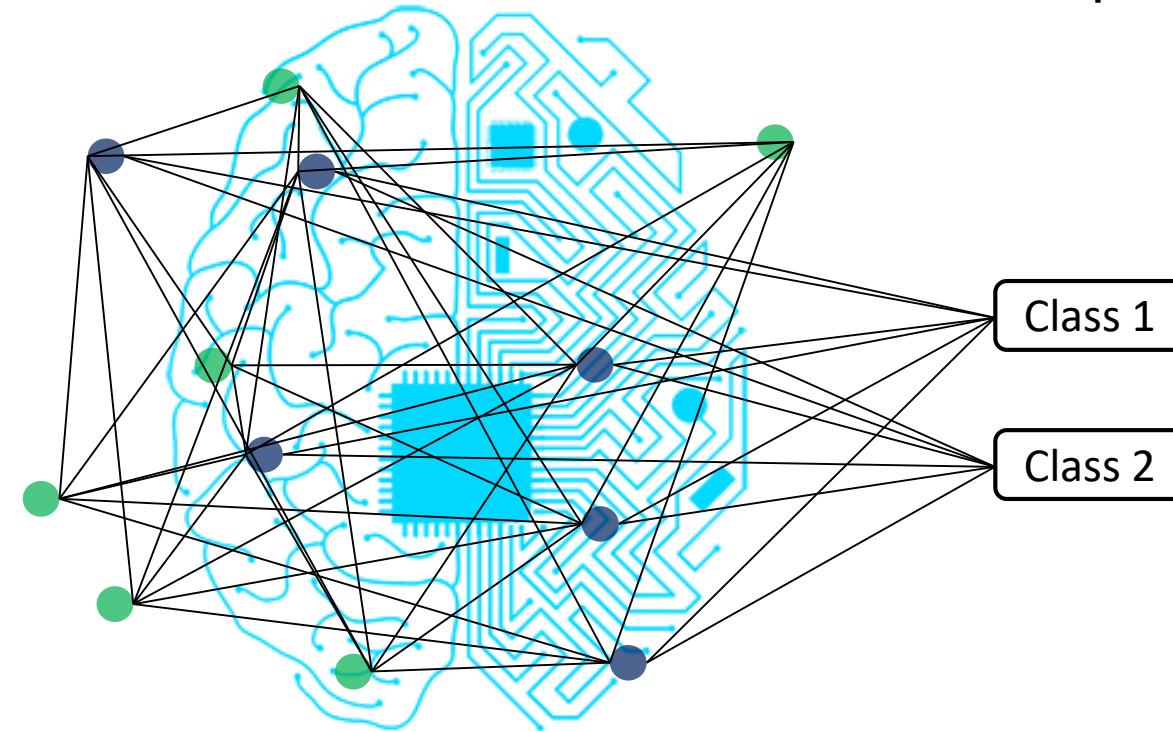
- Complex/Nonlinear universal function approximator
 - Linearly connected networks
 - Simple nonlinear neurons



Input

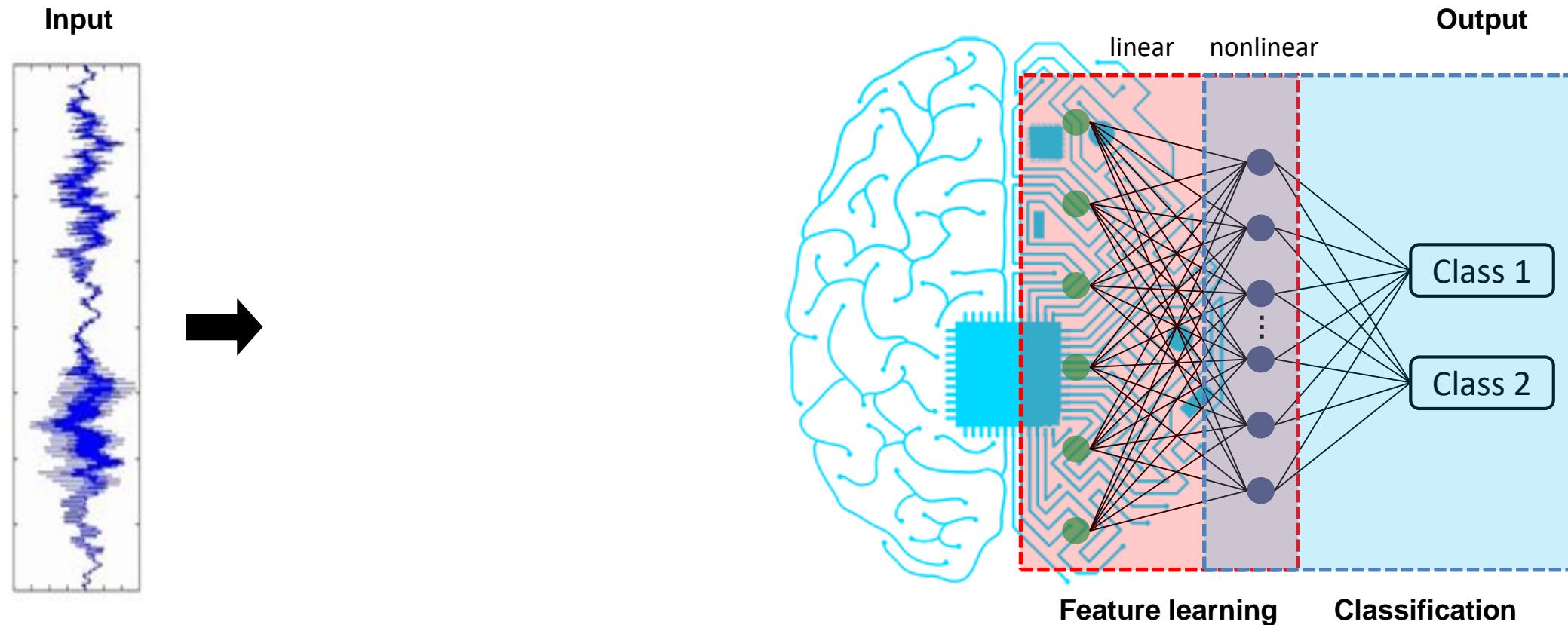


Output



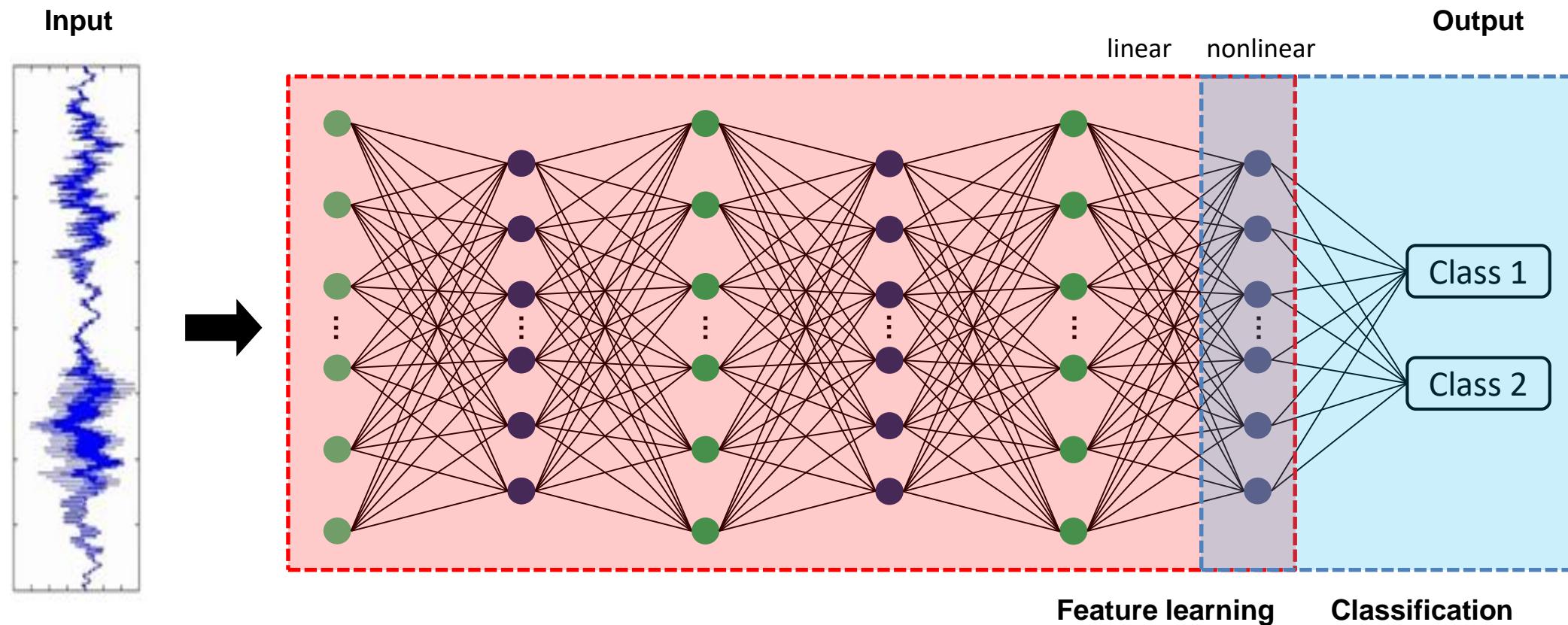
Artificial Neural Networks

- 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



Training: Backpropagation

Training Neural Networks: Optimization

- Learning or estimating weights and biases of multi-layer perceptron from training data
- 3 key components
 - objective function $f(\cdot)$
 - decision variable or unknown θ
 - constraints $g(\cdot)$
- In mathematical expression

$$\begin{aligned} \min_{\theta} \quad & f(\theta) \\ \text{subject to} \quad & g_i(\theta) \leq 0, \quad i = 1, \dots, m \end{aligned}$$

Training Neural Networks: Loss Function

- Measures error between target values and predictions

$$\min_{\theta} \sum_{i=1}^m \ell \left(h_{\theta} \left(x^{(i)} \right), y^{(i)} \right)$$

- Example
 - Squared loss (for regression):

$$\frac{1}{N} \sum_{i=1}^N \left(h_{\theta} \left(x^{(i)} \right) - y^{(i)} \right)^2$$

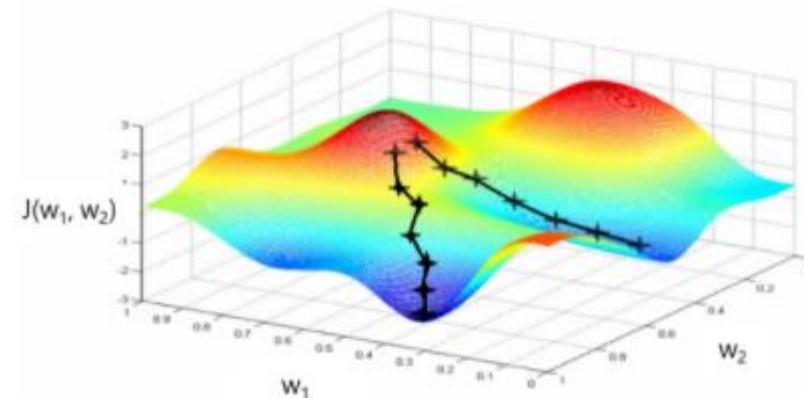
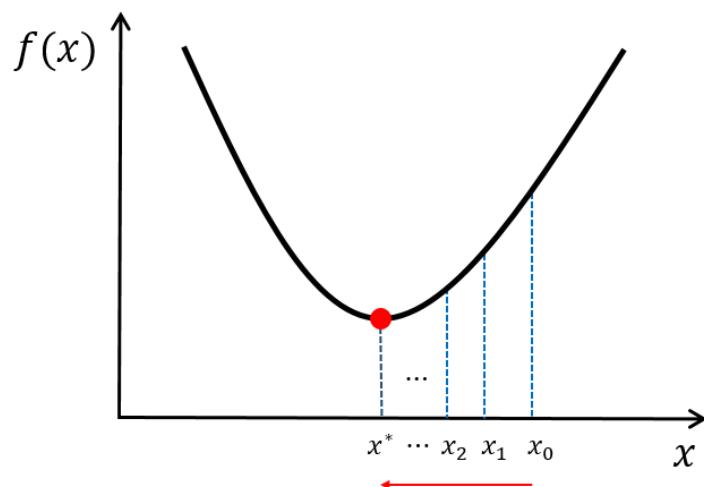
- Cross entropy (for classification):

$$-\frac{1}{N} \sum_{i=1}^N y^{(i)} \log \left(h_{\theta} \left(x^{(i)} \right) \right) + \left(1 - y^{(i)} \right) \log \left(1 - h_{\theta} \left(x^{(i)} \right) \right)$$

Training Neural Networks: Gradient Descent

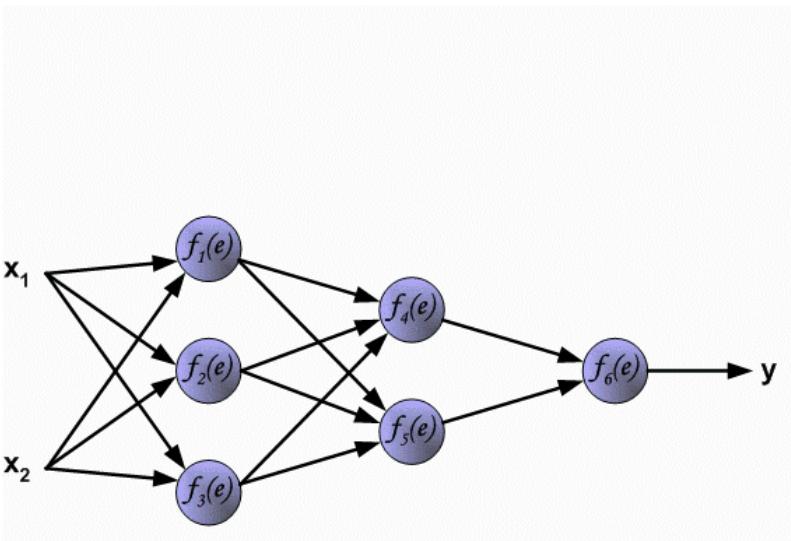
- Negative gradients points directly downhill of the cost function
- We can decrease the cost by moving in the direction of the negative gradient (α is a learning rate)

$$\theta := \theta - \alpha \nabla_{\theta} \left(h_{\theta} \left(x^{(i)} \right), y^{(i)} \right)$$



Training Neural Networks: Backpropagation Learning

- Forward propagation
 - the initial information propagates up to the hidden units at each layer and finally produces output
- Backpropagation
 - allows the information from the cost to flow backwards through the network in order to compute the gradients



Backpropagation

- Chain Rule
 - Computing the derivative of the composition of functions

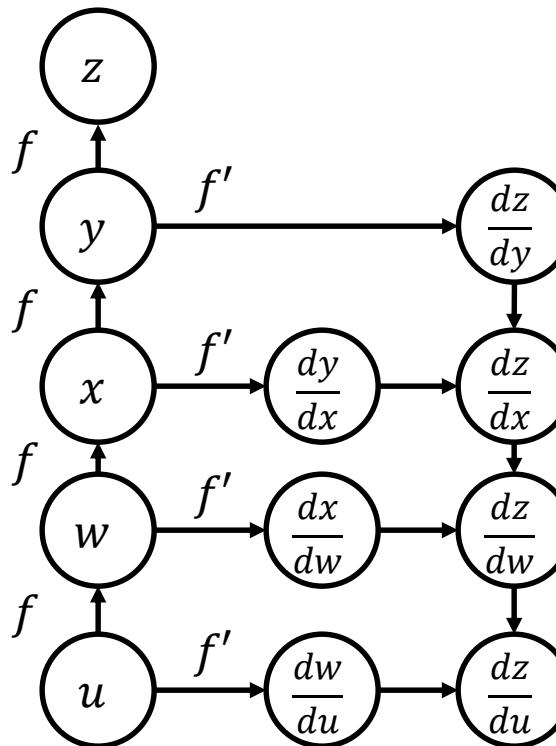
- $f(g(x))' = f'(g(x))g'(x)$

- $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$

- $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \right) \cdot \frac{dx}{dw}$

- $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw} \right) \cdot \frac{dw}{du}$

- Backpropagation
 - Update weights recursively



Backpropagation

- Chain Rule
 - Computing the derivative of the composition of functions

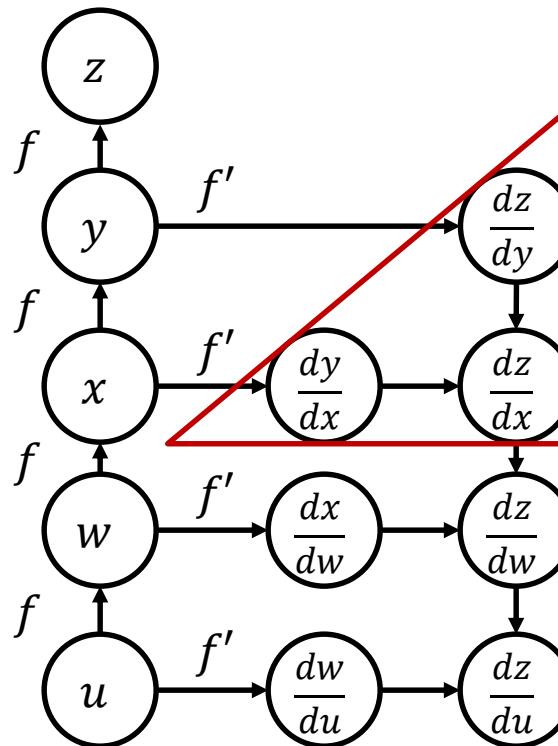
- $f(g(x))' = f'(g(x))g'(x)$

- $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$

- $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \right) \cdot \frac{dx}{dw}$

- $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw} \right) \cdot \frac{dw}{du}$

- Backpropagation
 - Update weights recursively



Backpropagation

- Chain Rule
 - Computing the derivative of the composition of functions

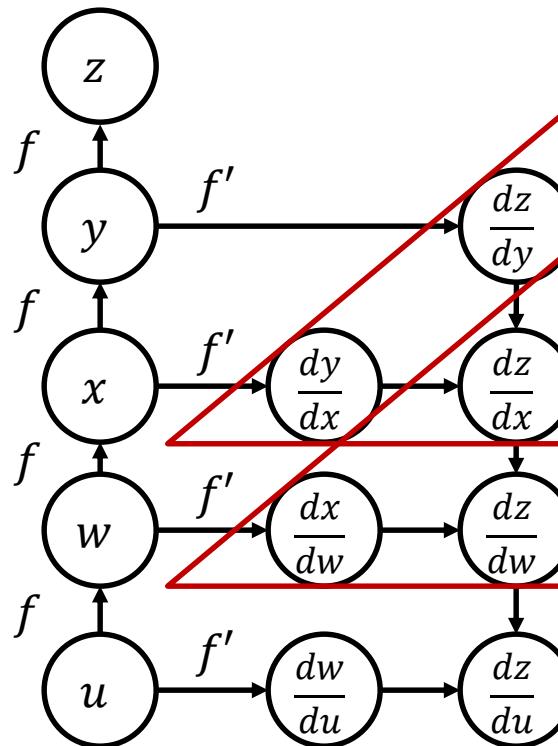
- $f(g(x))' = f'(g(x))g'(x)$

- $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$

- $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \right) \cdot \frac{dx}{dw}$

- $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw} \right) \cdot \frac{dw}{du}$

- Backpropagation
 - Update weights recursively



Backpropagation

- Chain Rule
 - Computing the derivative of the composition of functions

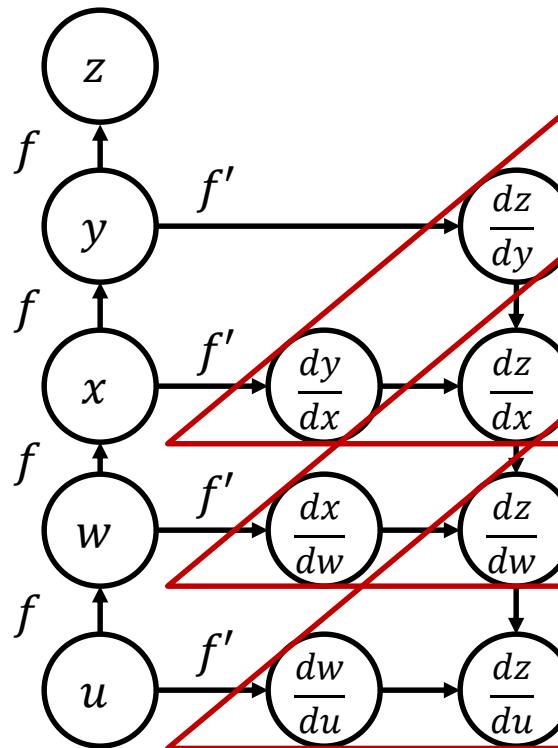
- $f(g(x))' = f'(g(x))g'(x)$

- $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$

- $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \right) \cdot \frac{dx}{dw}$

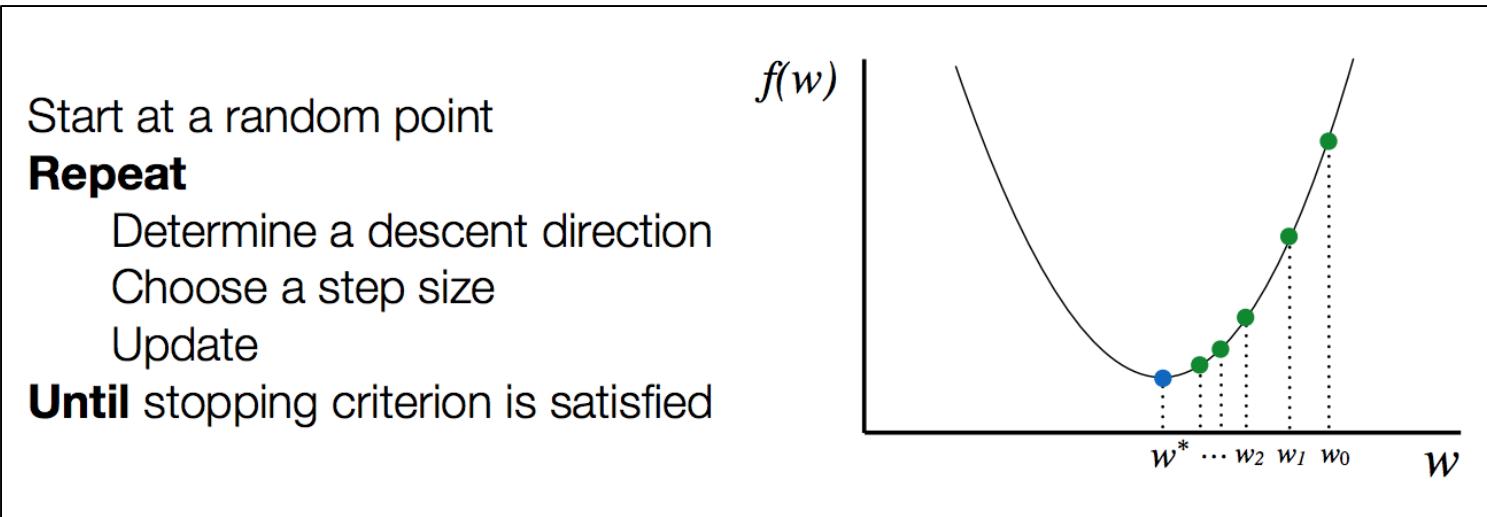
- $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw} \right) \cdot \frac{dw}{du}$

- Backpropagation
 - Update weights recursively



Training Neural Networks

- Optimization procedure

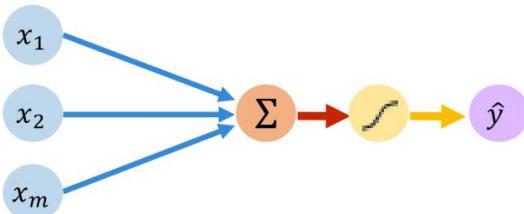


- It is not easy to numerically compute gradients in network in general.
 - The good news: people have already done all the "hard work" of developing numerical solvers (or libraries)
 - There are a wide range of tools

Core Foundation Review

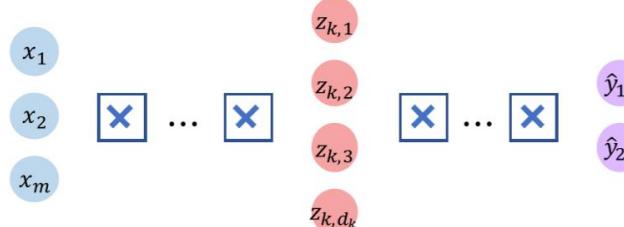
The Perceptron

- Structural building blocks
- Nonlinear activation functions



Neural Networks

- Stacking Perceptrons to form neural networks
- Optimization through backpropagation



Training in Practice

- Adaptive learning
- Batching
- Regularization

