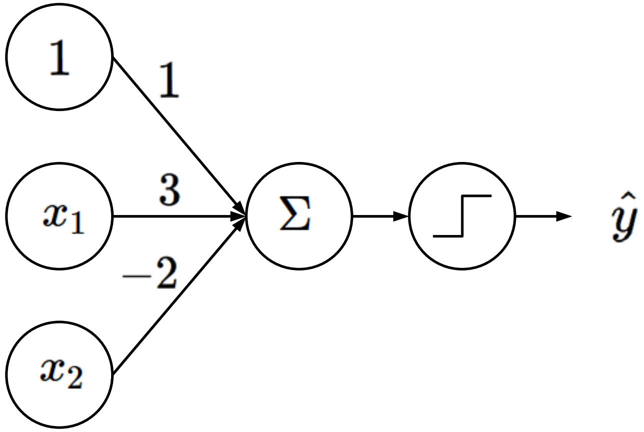
(Artificial) Neural Networks: From Perceptron to MLP

Industrial AI Lab.

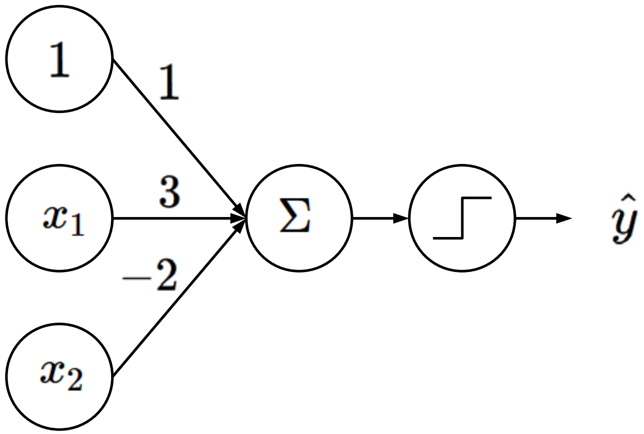
Prof. Seungchul Lee

Perceptron: Example

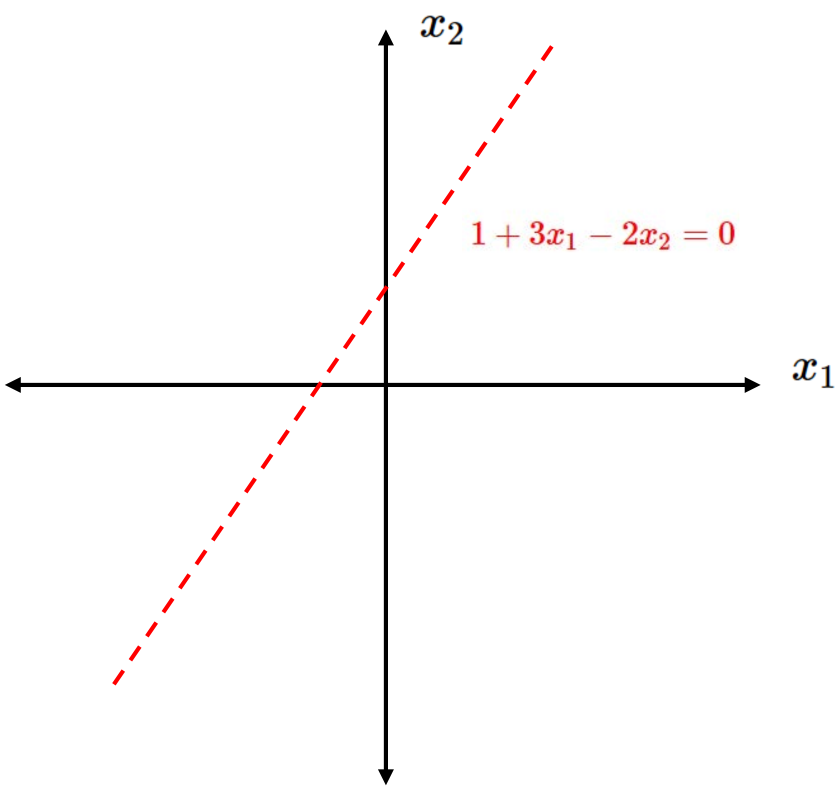


$$
\begin{aligned}
\hat{y} &= g\left(\omega_0 + X^T\omega\right) \\
&= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \\
&= g\left(1 + 3x_1 - 2x_2\right)
\end{aligned}
$$

Perceptron: Example

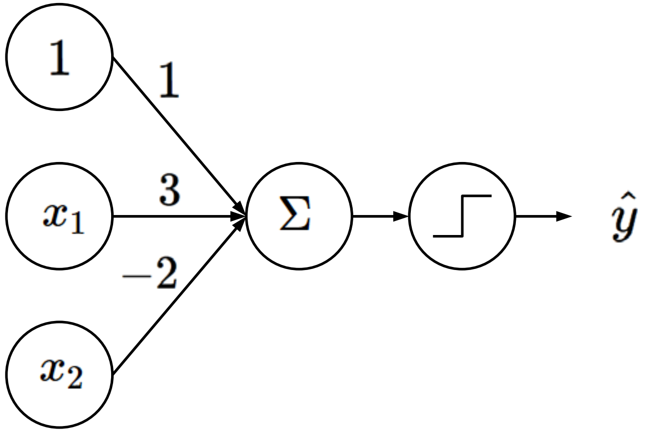


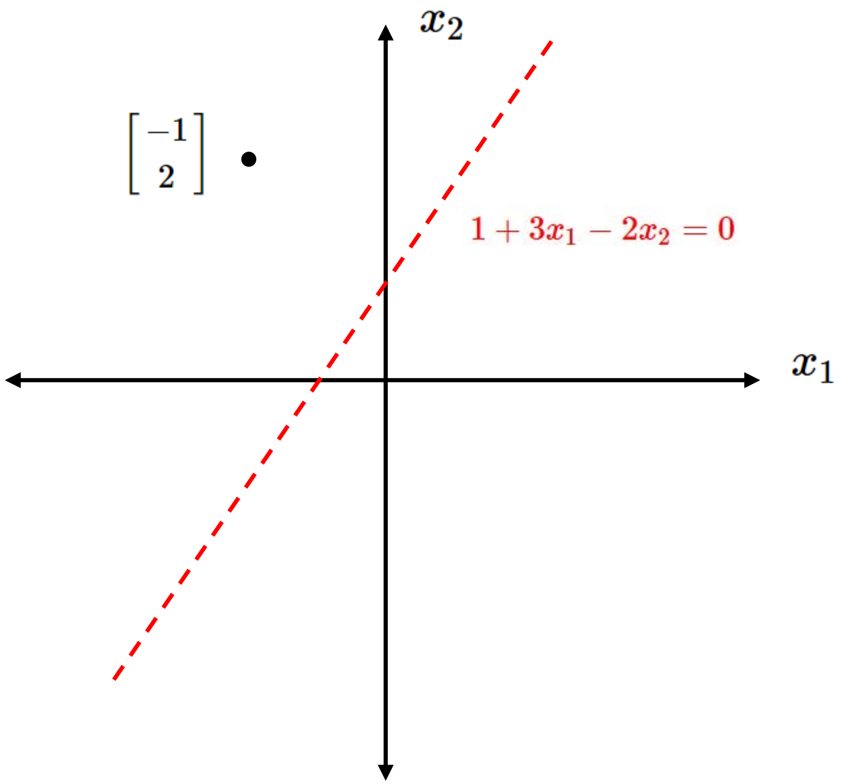
$$\hat{y} = g\left(1 + 3x_1 - 2x_2\right)$$



Perceptron: Example

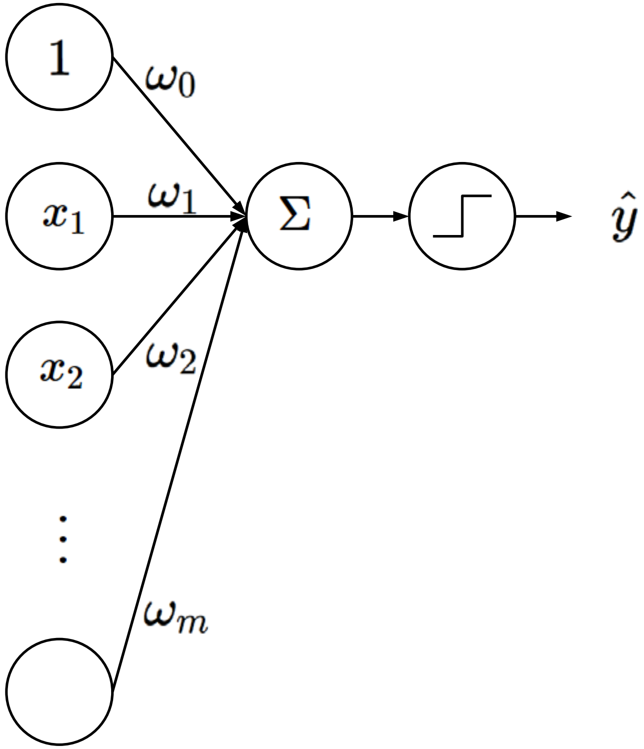
$$\hat{y} = g\left(1 + 3x_1 - 2x_2\right)$$





$$\hat{y} = g\left(1 + 3 \times (-1) - 2 \times 2\right) = g(-6) = -1$$

Perceptron: Forward Propagation



$$\hat{y} = g\left(\omega_0 + X^T \omega\right)$$

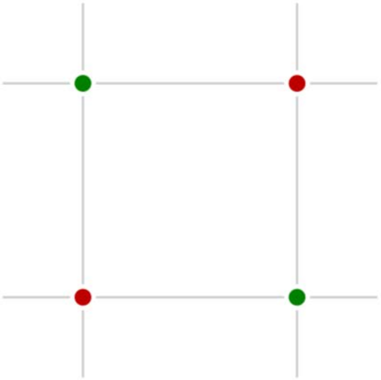
$$= g\left(\omega_0 + \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}^T \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_m \end{bmatrix}\right)$$

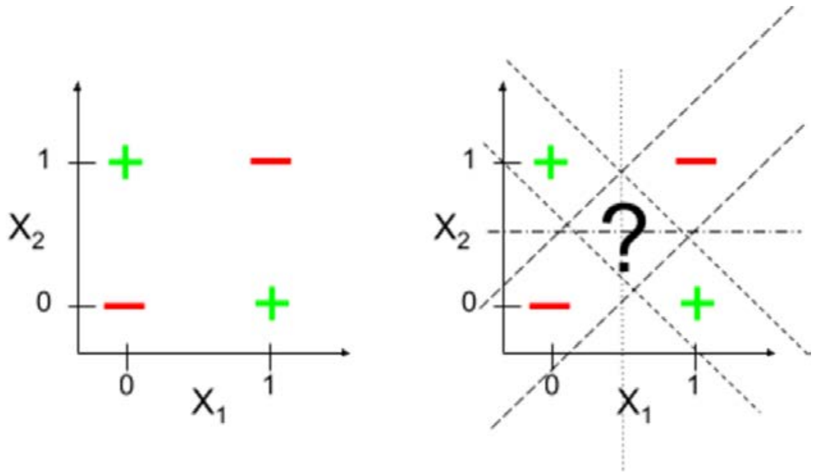
From Perceptron to MLP

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

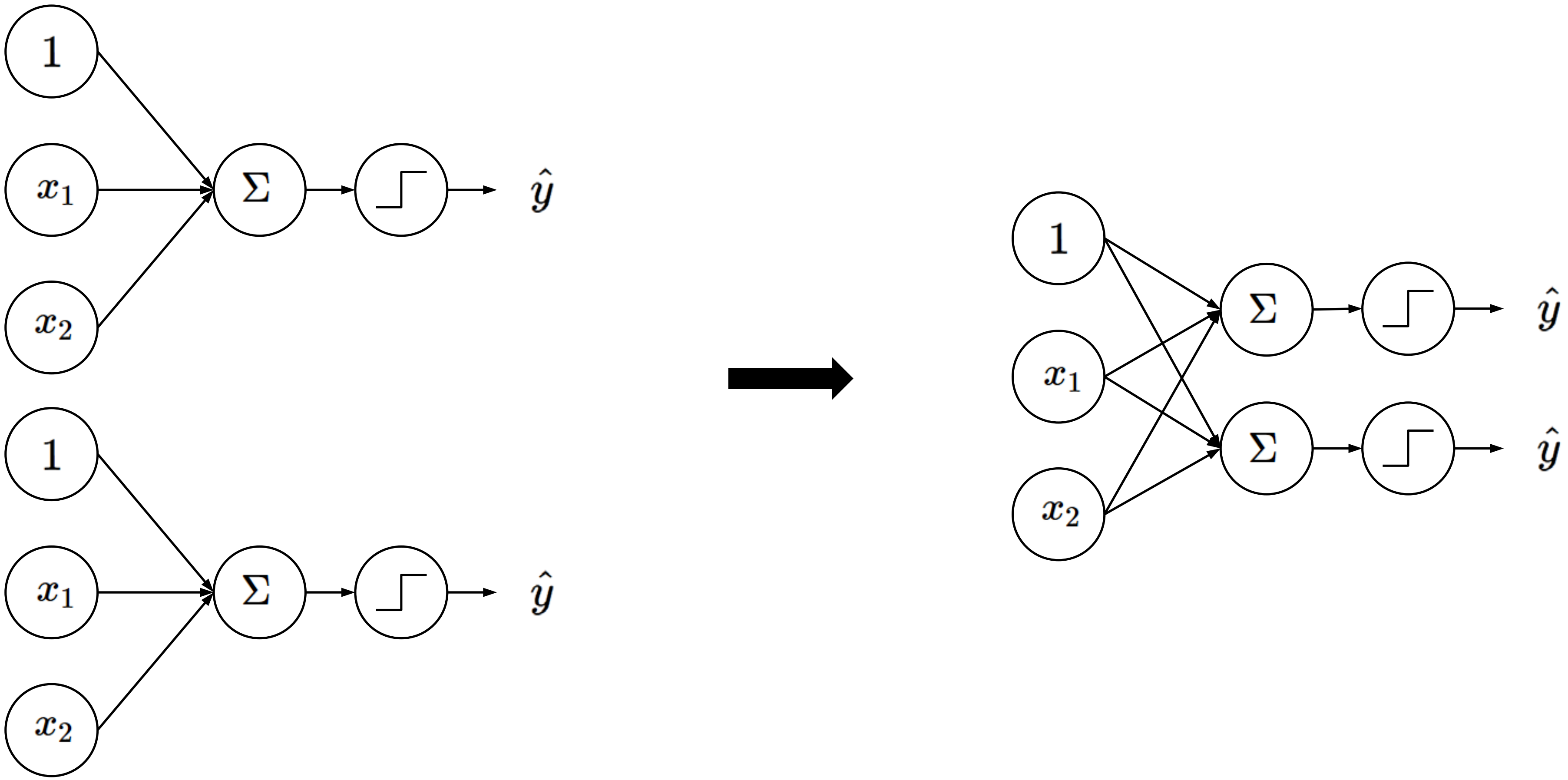




- Single neuron = one linear classification boundary

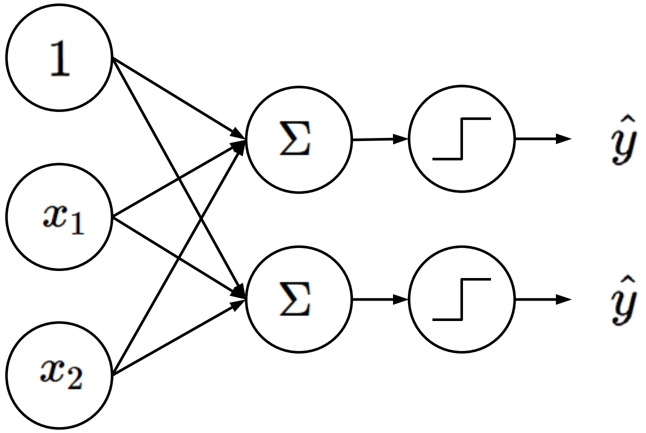
Artificial Neural Networks: MLP

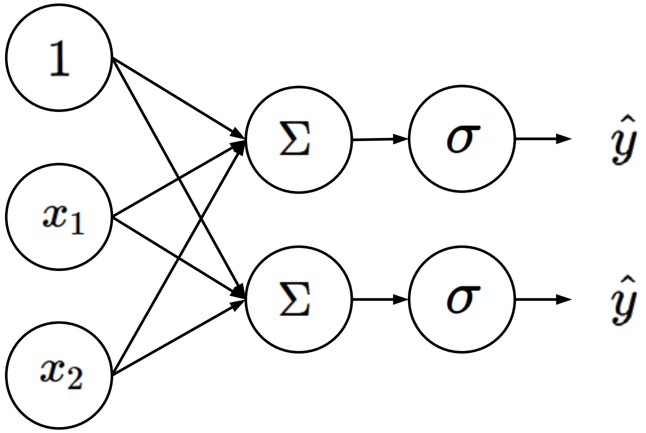
- Multi-layer Perceptron (MLP) = Artificial Neural Networks (ANN)
  - Multi neurons = multiple linear classification boundaries

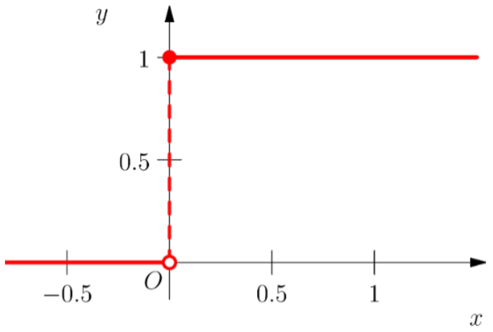


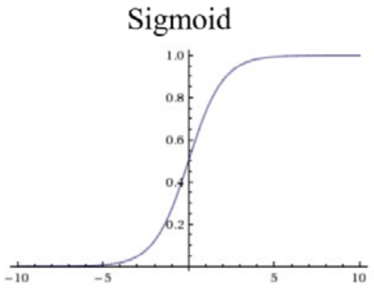
Artificial Neural Networks: Activation Function

- Differentiable nonlinear activation function





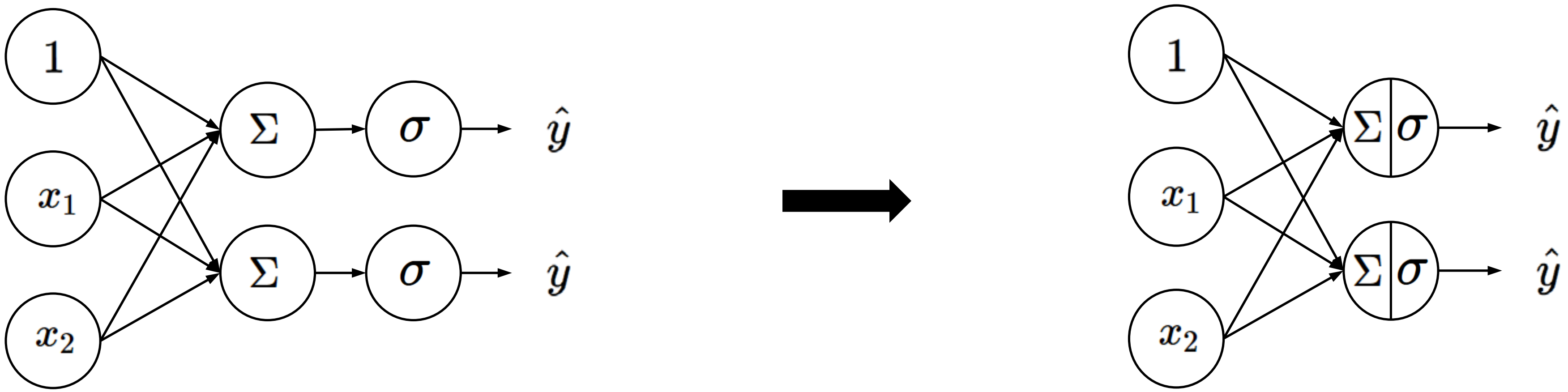




$$g(x) = \frac{1}{1 + e^{-x}}$$

Artificial Neural Networks

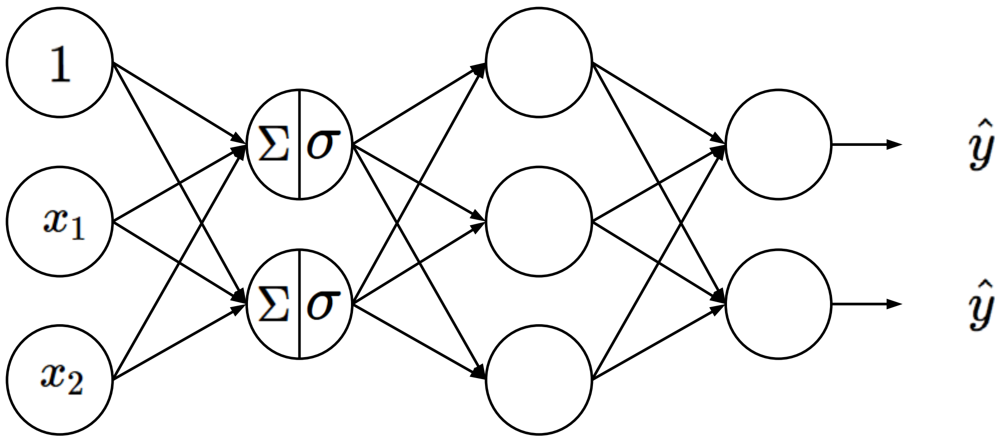
- In a compact representation



Artificial Neural Networks

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

  $\Longrightarrow$ perceptron with many layers and units



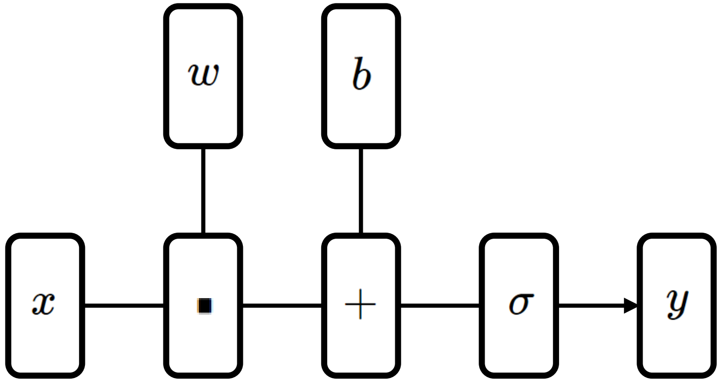
- Multi-layer perceptron
  - Features of features
  - Mapping of mappings

Another Perspective:
ANN as Kernel Learning

Neuron

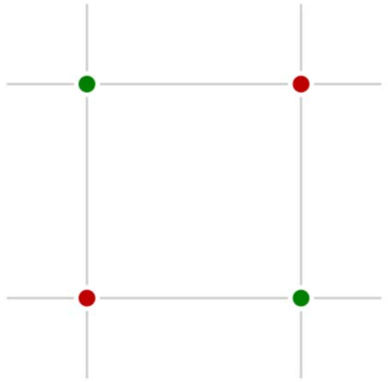
- We can represent this "neuron" as follows:

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



XOR Problem

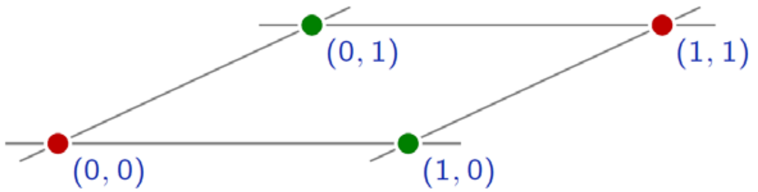
- The main weakness of linear predictors is their lack of capacity.
- For classification, the populations have to be linearly separable.



“xor”

Nonlinear Mapping

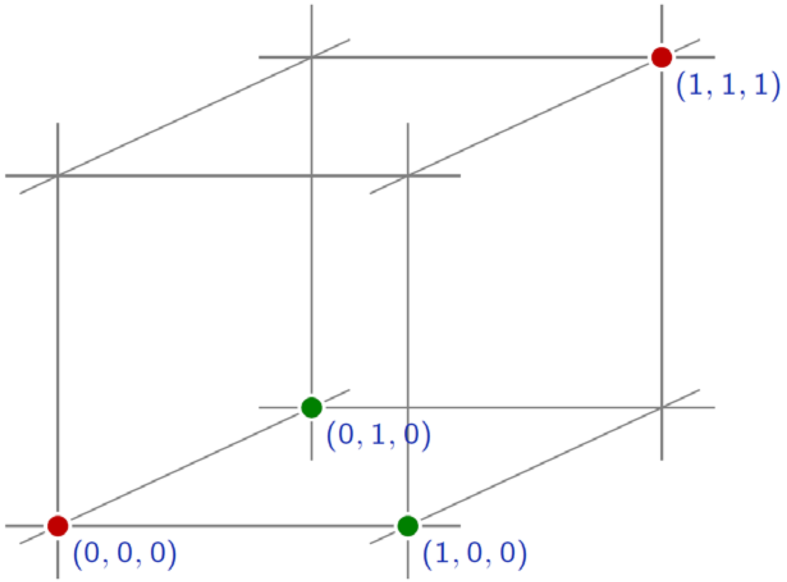
- The XOR example can be solved by pre-processing the data to make the two populations linearly separable.



Nonlinear Mapping

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

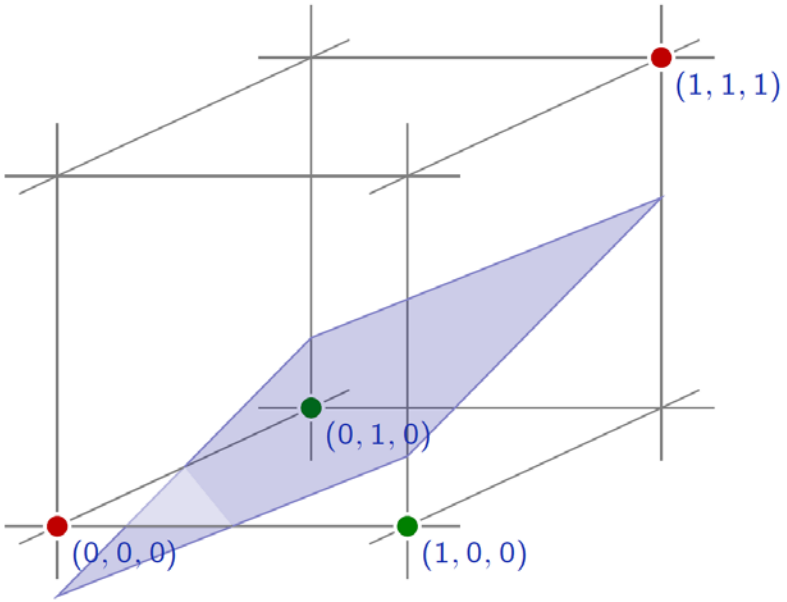
$$\phi : (x_u, x_v) \rightarrow (x_u, x_v, x_u x_v)$$



Nonlinear Mapping

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

$$\phi : (x_u, x_v) \rightarrow (x_u, x_v, x_u x_v)$$



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

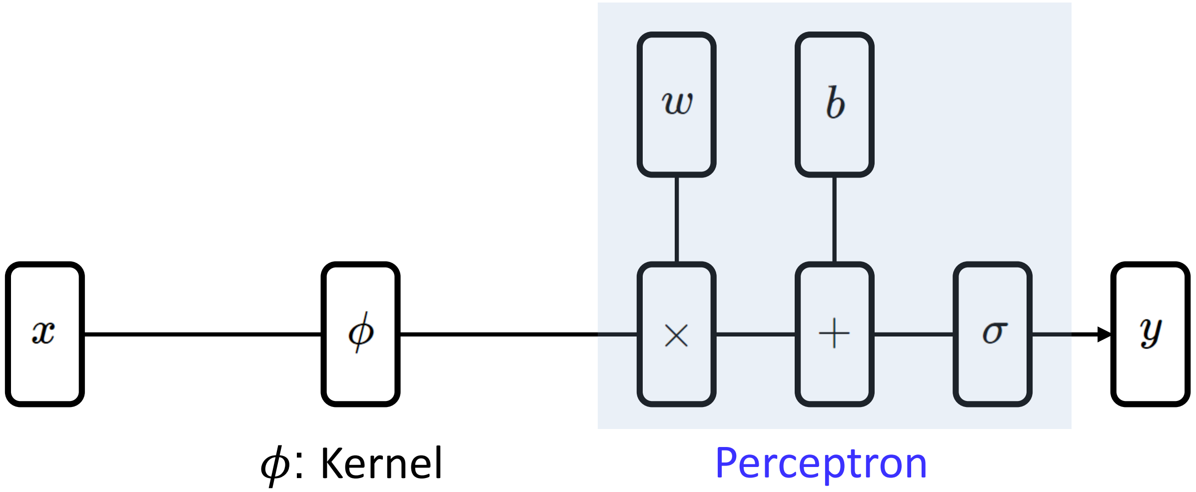
- Linear models (e.g. linear regression, linear SVM) are not just rich enough
  - by mapping data to higher dimensions where it exhibits linear patterns
  - apply the linear model in the new input feature space
  - mapping = changing the feature representation

- Kernels: make linear model work in nonlinear settings

Kernel + Neuron

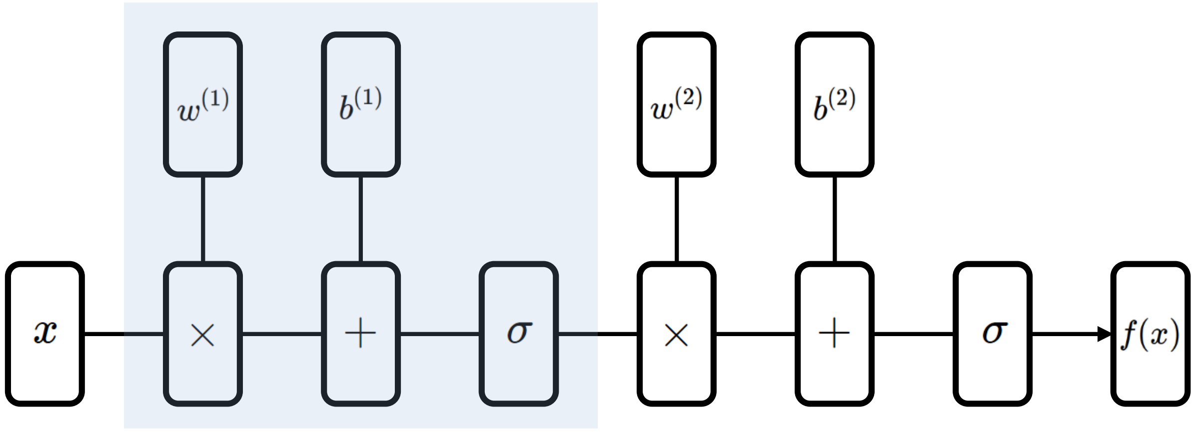
- Nonlinear mapping + neuron

$$\phi : (x_u, x_v) \rightarrow (x_u, x_v, x_u x_v)$$



Neuron + Neuron

- Nonlinear mapping can be represented by another neurons

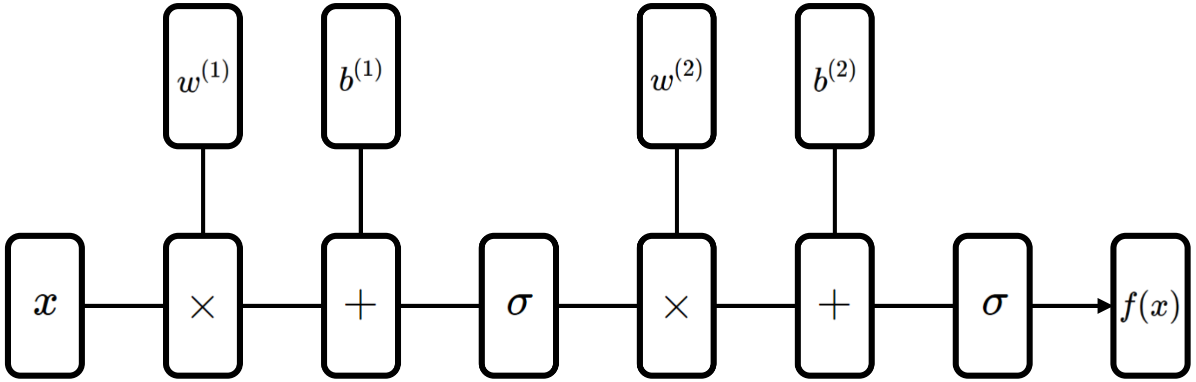


ϕ: Kernel

- Nonlinear Kernel
  - Nonlinear activation functions

Multi Layer Perceptron

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

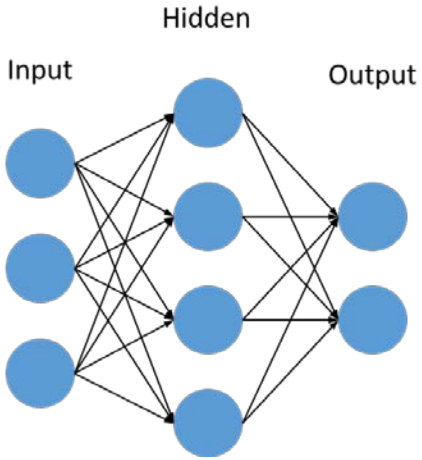


ϕ: Kernel

Summary

- Universal function approximator
- Universal function classifier

- Parameterized

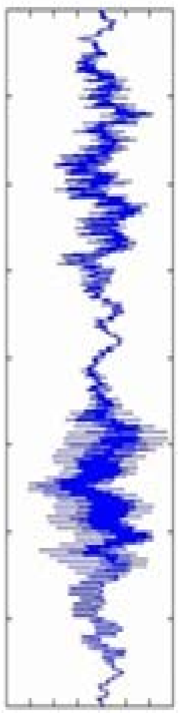


$$x \longrightarrow \boxed{\hat{y} = f_{\omega_1, \cdots, \omega_k}(x)} \longrightarrow y$$

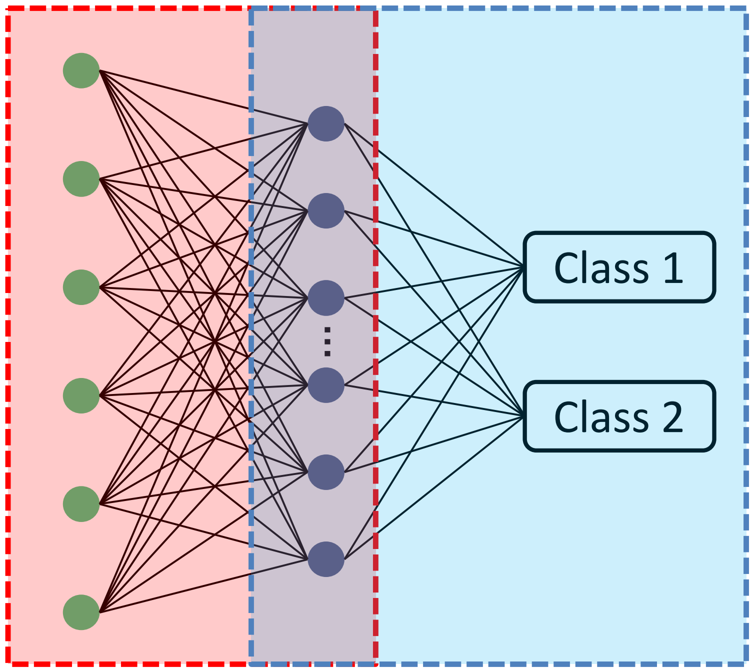
Artificial Neural Networks

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

Input



Output

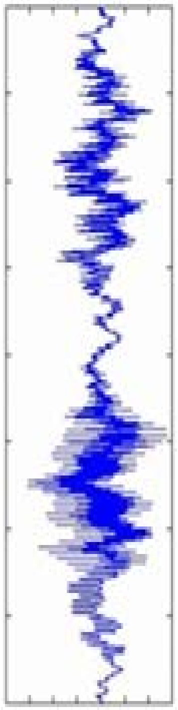


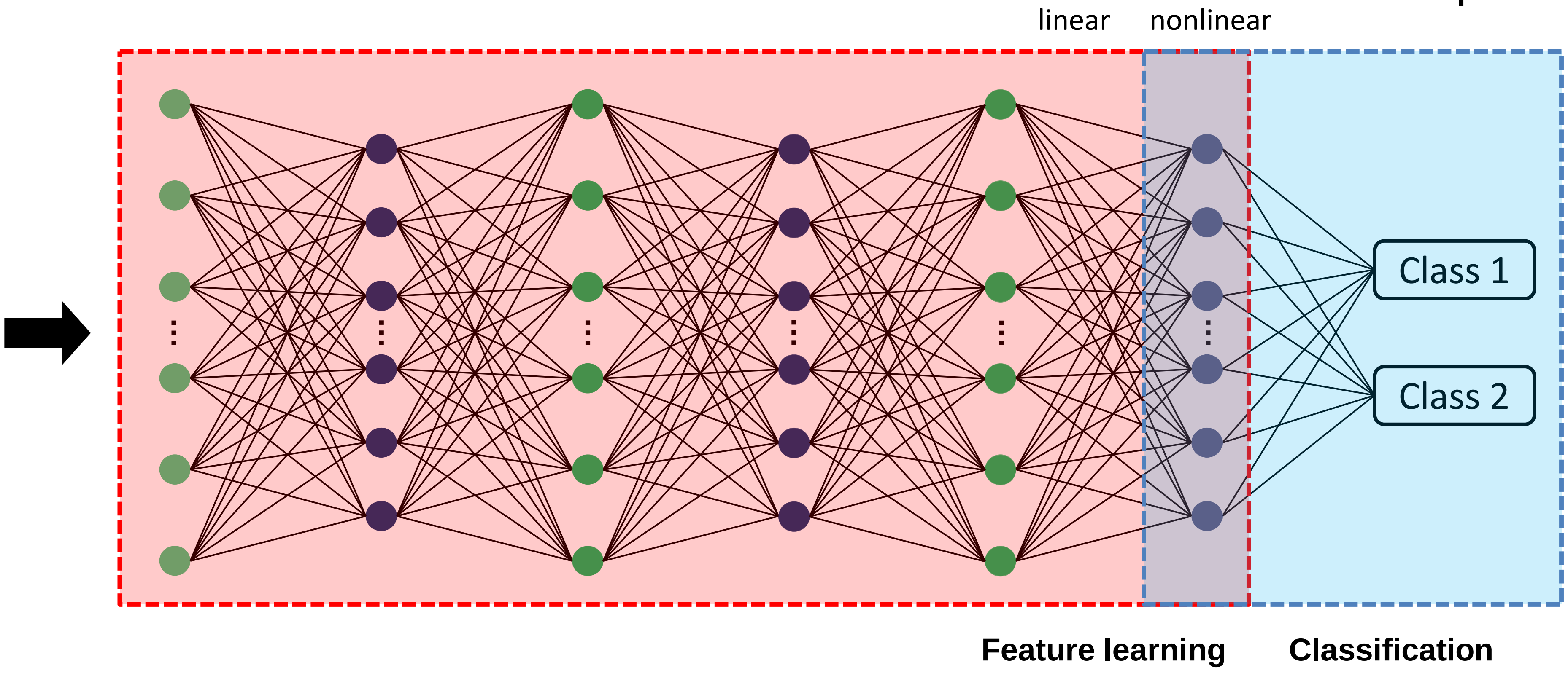
Feature learning **Classification**

Deep Artificial Neural Networks

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

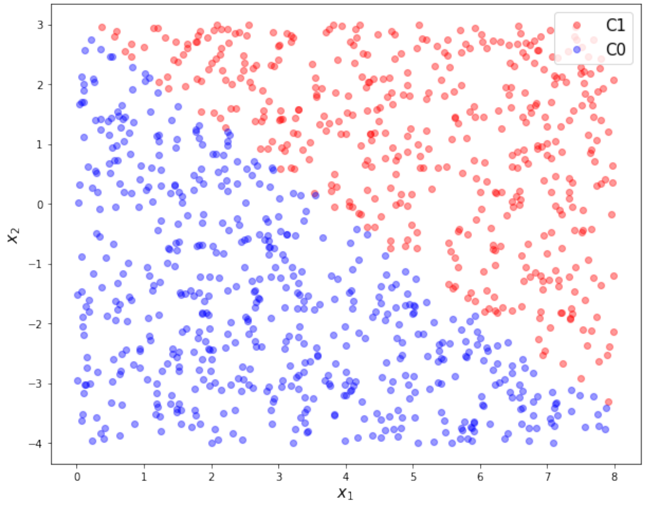
Input



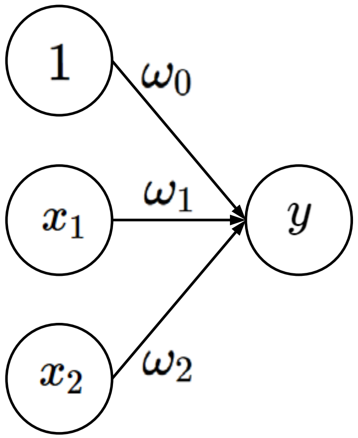


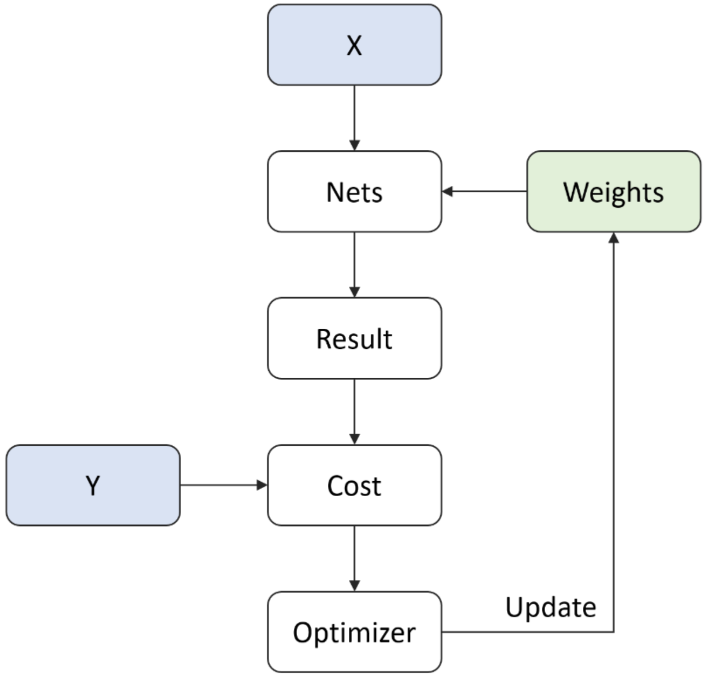
Looking at Parameters

Logistic Regression in a Form of Neural Network



$$y = \sigma\left(\omega_0 + \omega_1 x_1 + \omega_2 x_2\right)$$

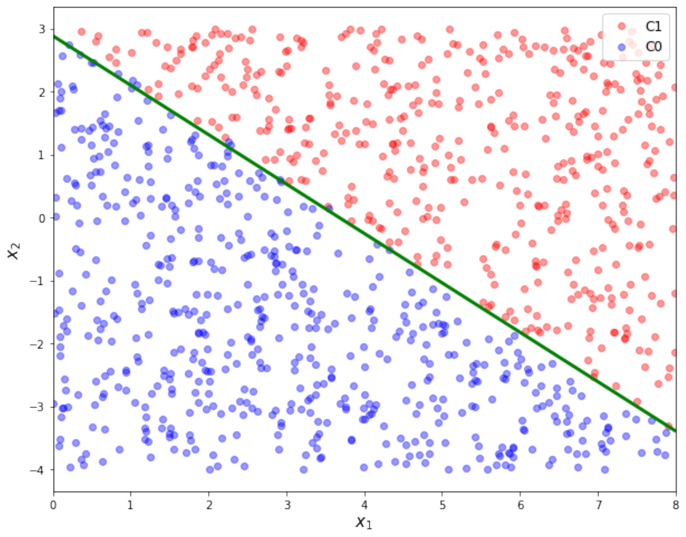




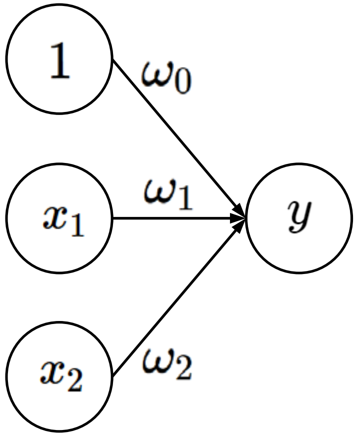
POSTECH

Logistic Regression in a Form of Neural Network

- Neural network convention

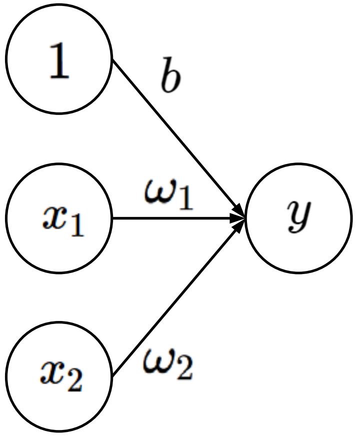


$$y = \sigma\left(\omega_0 + \omega_1 x_1 + \omega_2 x_2\right)$$

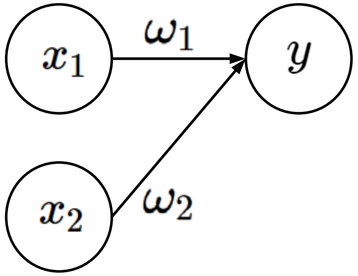


$\omega_0 \longrightarrow b$

$$y = \sigma\left(b + \omega_1 x_1 + \omega_2 x_2\right)$$



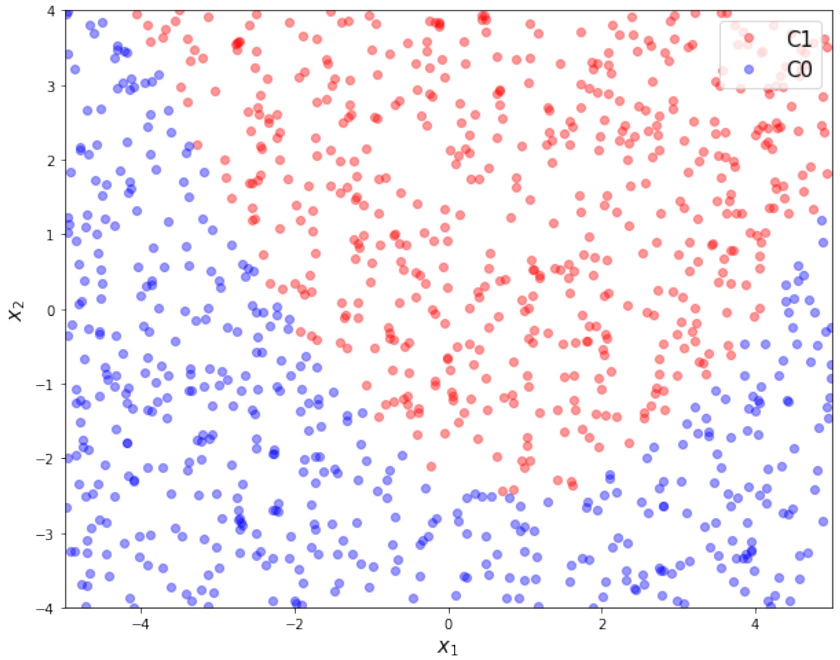
Do not indicate bias units

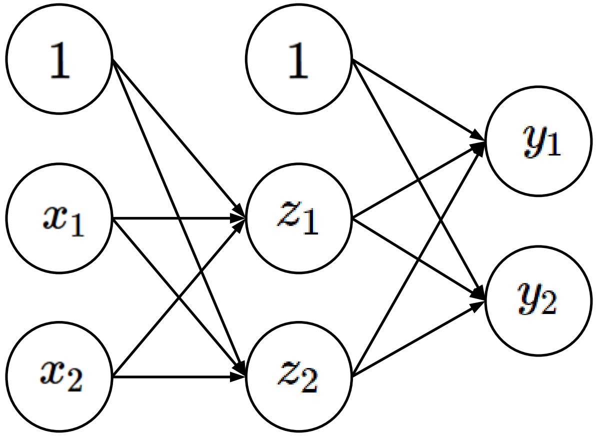



```
n_input = 2
n_output = 1
```

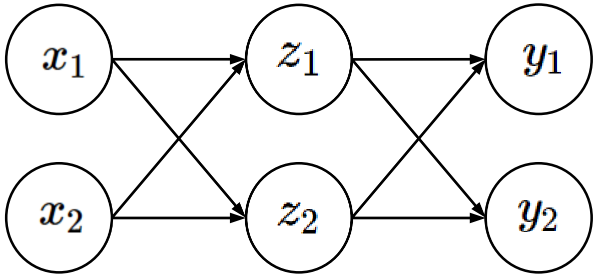
Nonlinearly Distributed Data

- Example to understand network's behavior
  - Include a hidden layer





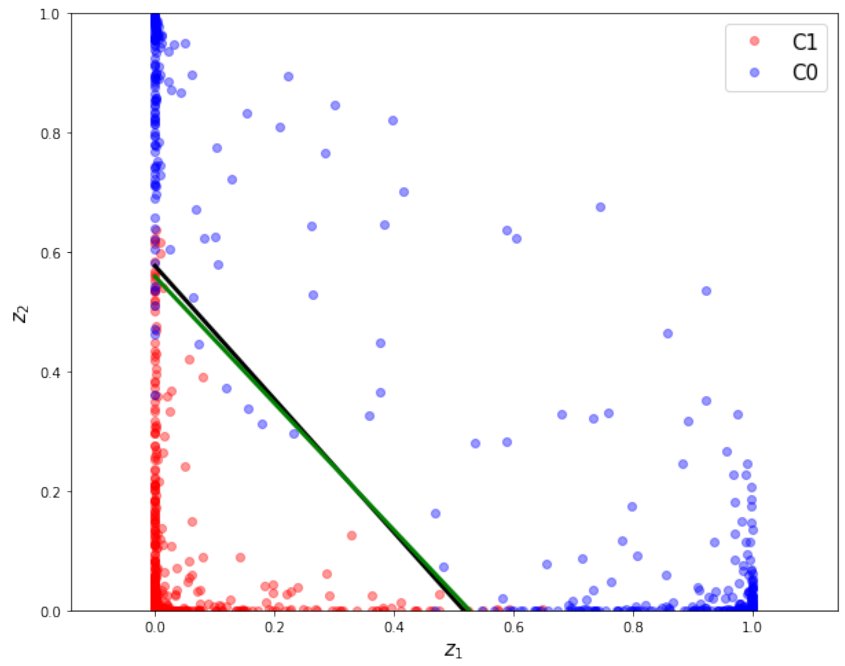
Do not include bias units

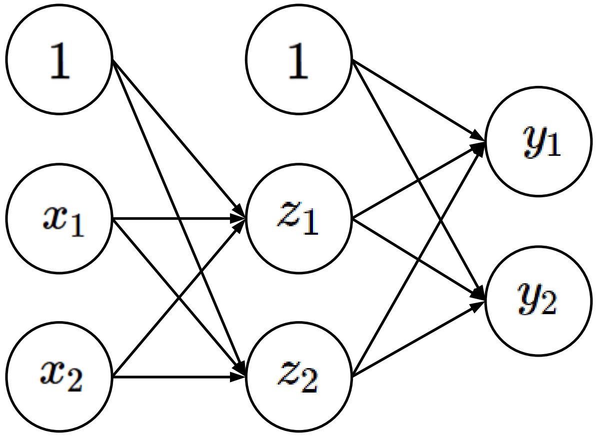



```
n_input  = 2
n_hidden  = 2
n_output  = 2
```

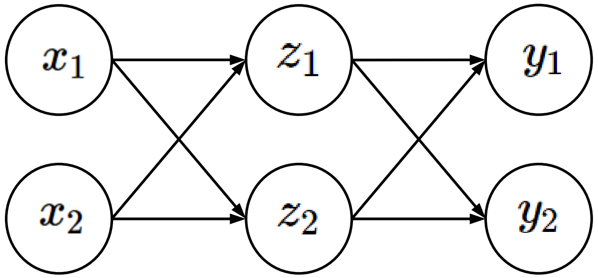
Multi Layers

- z space





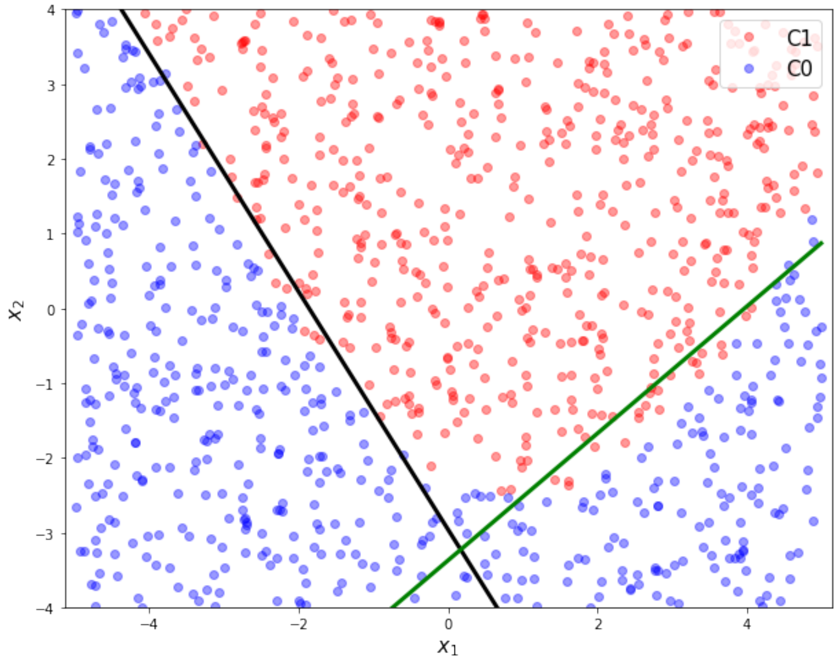
Do not include bias units

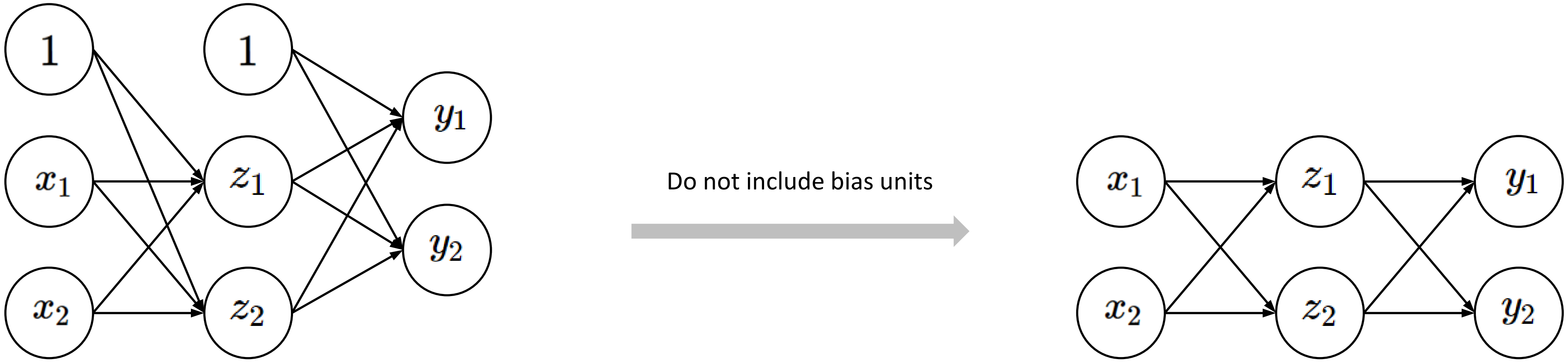


```
n_input = 2
n_hidden = 2
n_output = 2
```

Multi Layers

- x space






```
n_input  = 2
n_hidden  = 2
n_output  = 2
```

(Artificial) Neural Networks: Training

Industrial AI

Prof. Seungchul Lee

Training Neural Networks: Loss Function

- Measures error between target values and predictions

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

- Example
  - Squared loss (for regression):

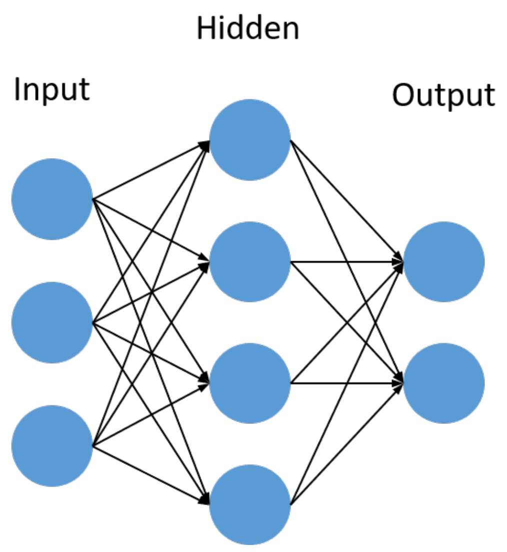
$$\frac{1}{m} \sum_{i=1}^{m} \left(h_\omega \left(x^{(i)} \right) - y^{(i)} \right)^2$$

  - Cross entropy (for classification):

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

Gradients in ANN

- Learning weights and biases from data using gradient descent
- $\frac{\partial \ell}{\partial \omega}$: too many computations are required for all ω



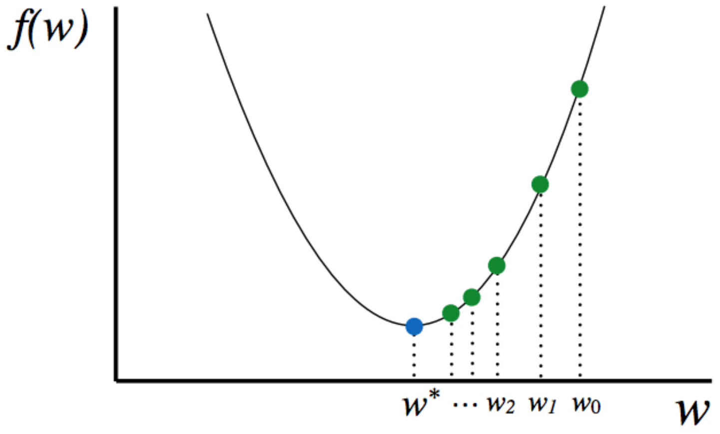
- Structural constraint of NN:
  - Composition of functions
  - Chain rule
  - Dynamic programming

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

Training Neural Networks with TensorFlow

- Optimization procedure

Start at a random point
Repeat
  Determine a descent direction
  Choose a step size
  Update
Until stopping criterion is satisfied

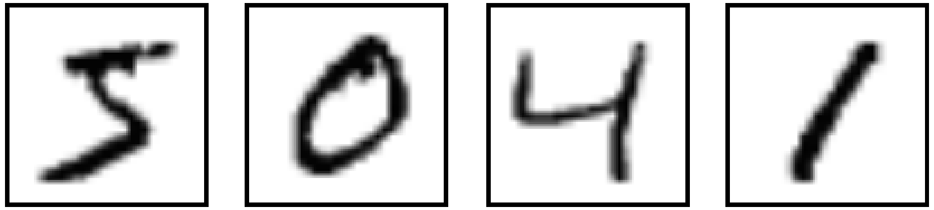


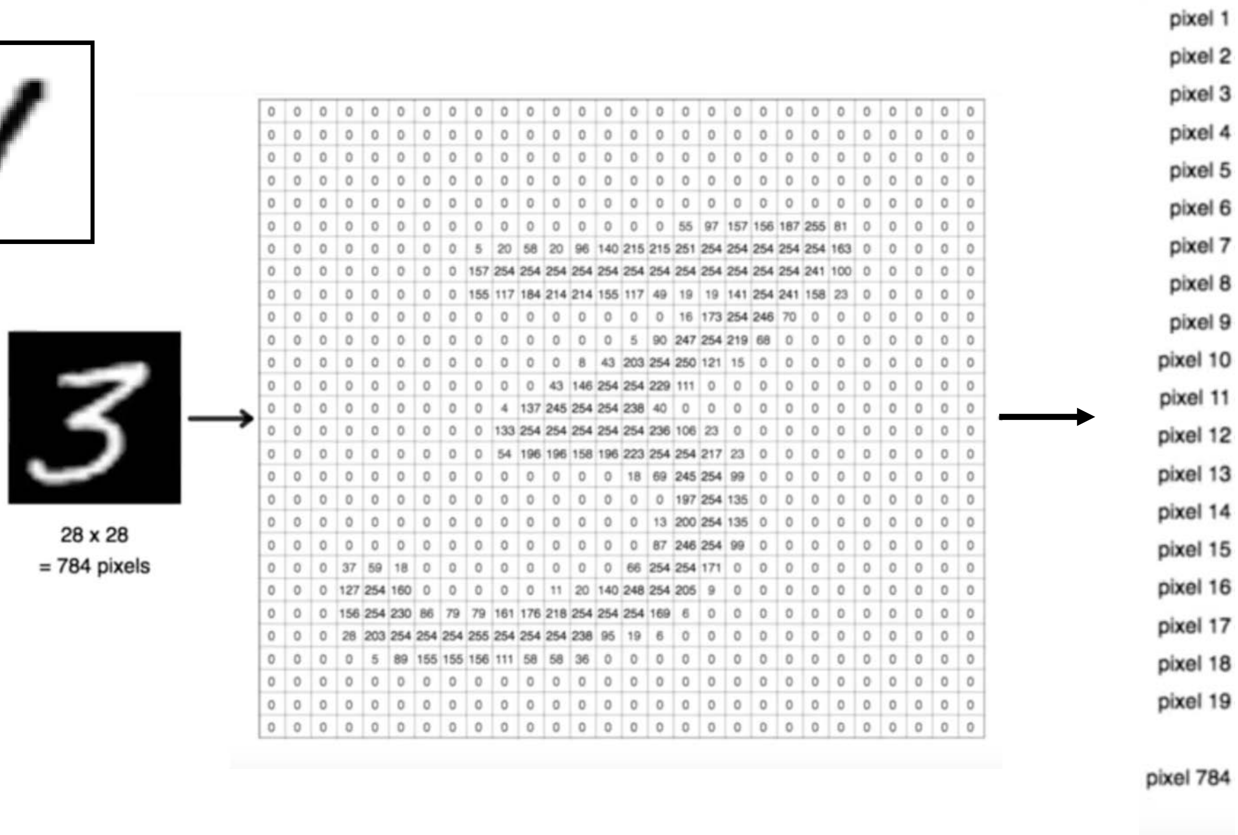
- It is not easy to numerically compute gradients in network in general.
  - The good news: people have already done all the "hard work" of developing numerical solvers (or libraries)
  - There are a wide range of tools → We will use the TensorFlow

ANN in TensorFlow: MNIST

MNIST database

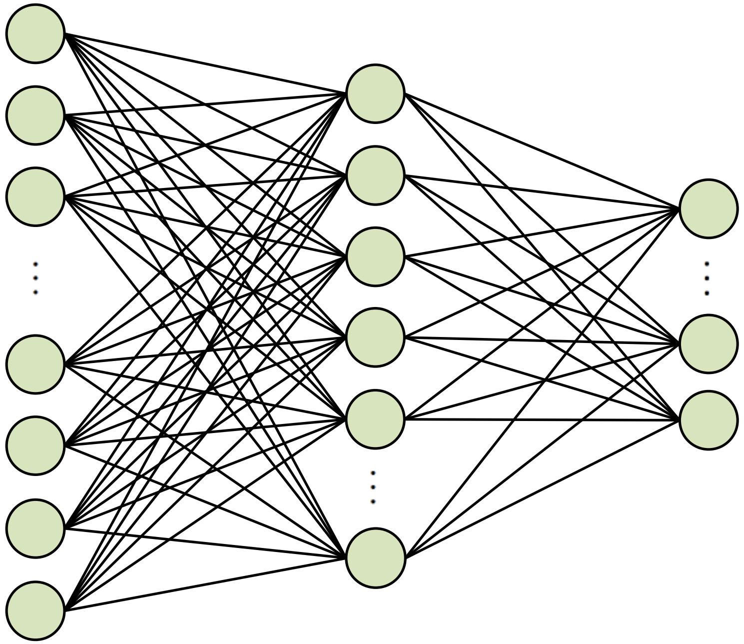
- Mixed National Institute of Standards and Technology database
- Handwritten digit database
- 28×28 gray scaled image
- Flattened matrix into a vector of $28 \times 28 = 784$





Our Network Model

flattened



digit prediction
in one-hot-encoding

Input image
(28 X 28)

Input layer
(784)

hidden layer
(100)

output layer
(10)