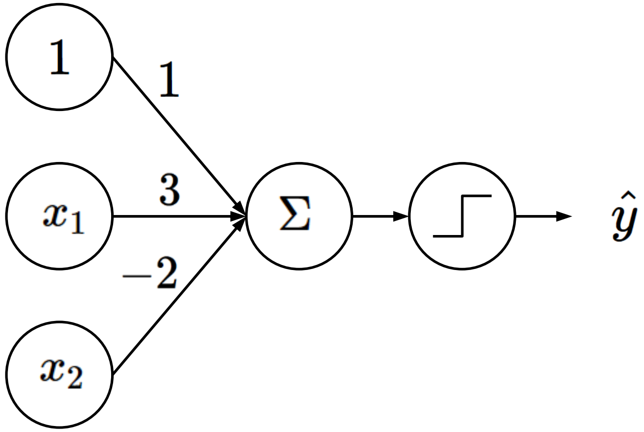
(Artificial) Neural Networks: From Perceptron to MLP

Prof. Seungchul Lee

Industrial AI Lab.

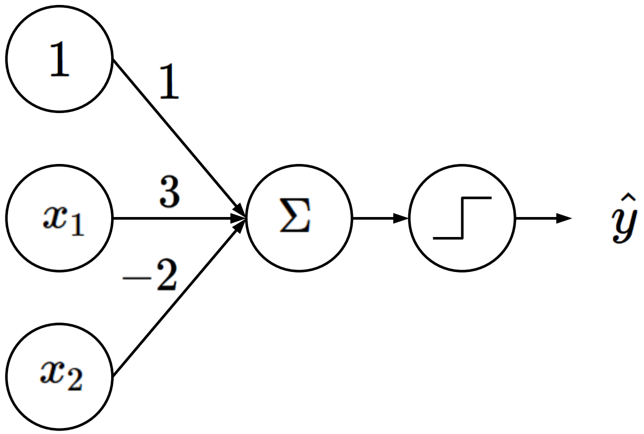
Perceptron: Example

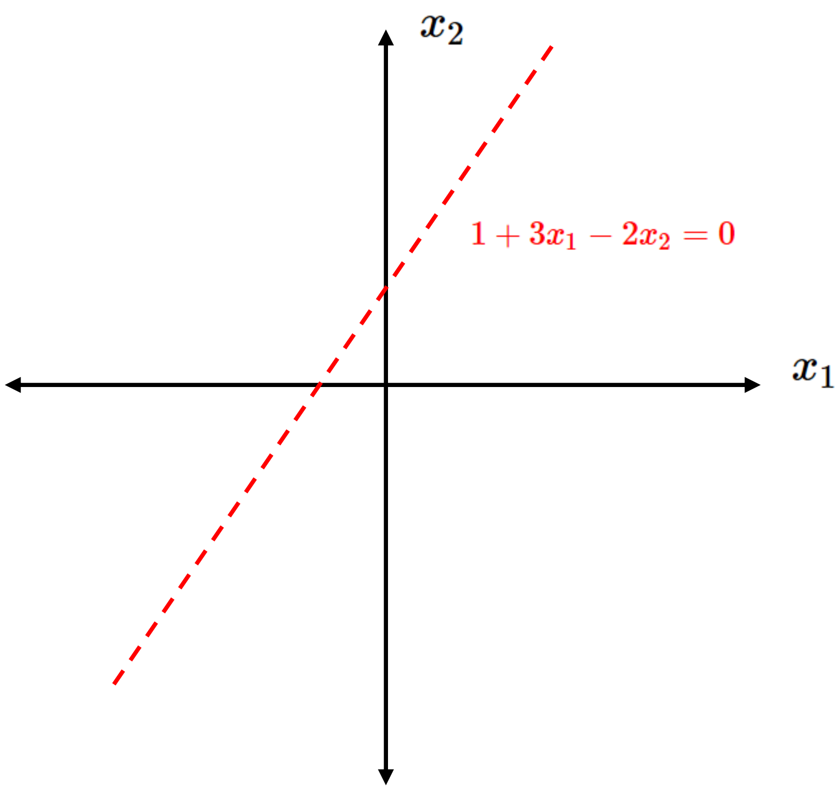


$$
\begin{aligned}
\hat{y} &= g\left(\omega_0 + X^T\omega\right) \\
&= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \\
&= g\left(1 + 3x_1 - 2x_2\right)
\end{aligned}
$$

Perceptron: Example

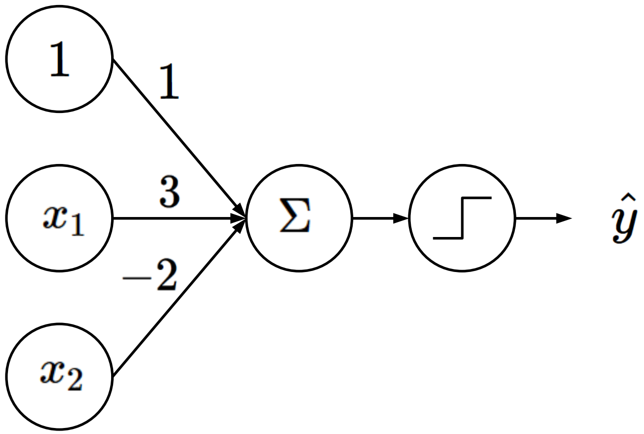
$$\hat{y} = g\left(1 + 3x_1 - 2x_2\right)$$

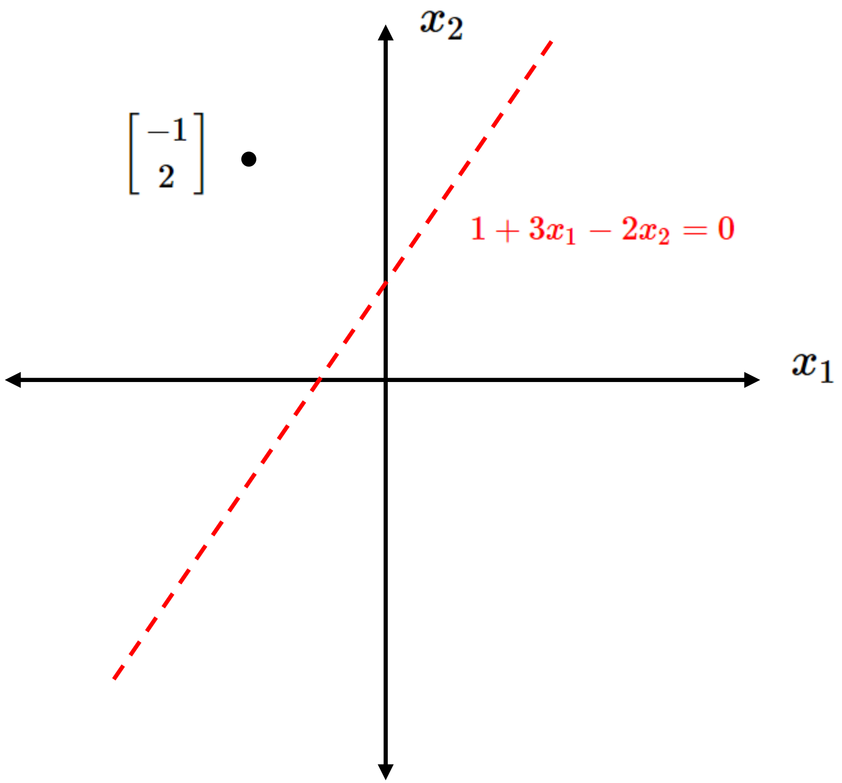




Perceptron: Example

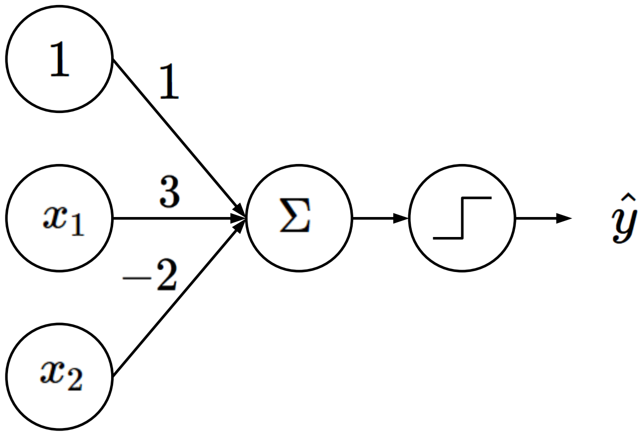
$$\hat{y} = g\left(1 + 3x_1 - 2x_2\right)$$



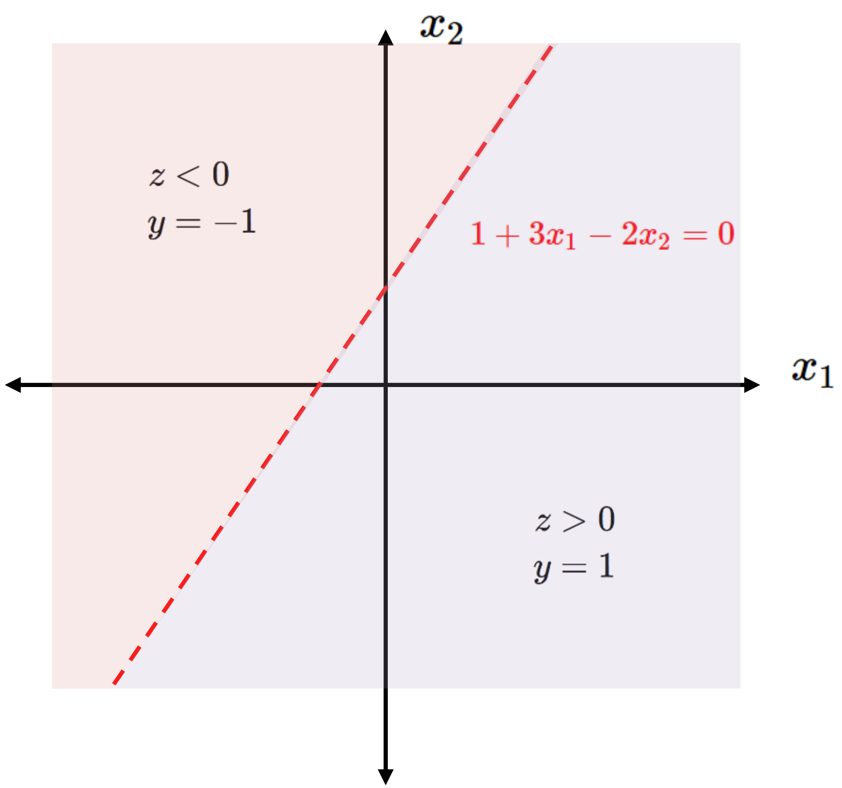


$$\hat{y} = g\left(1 + 3 \times (-1) - 2 \times 2\right) = g(-6) = -1$$

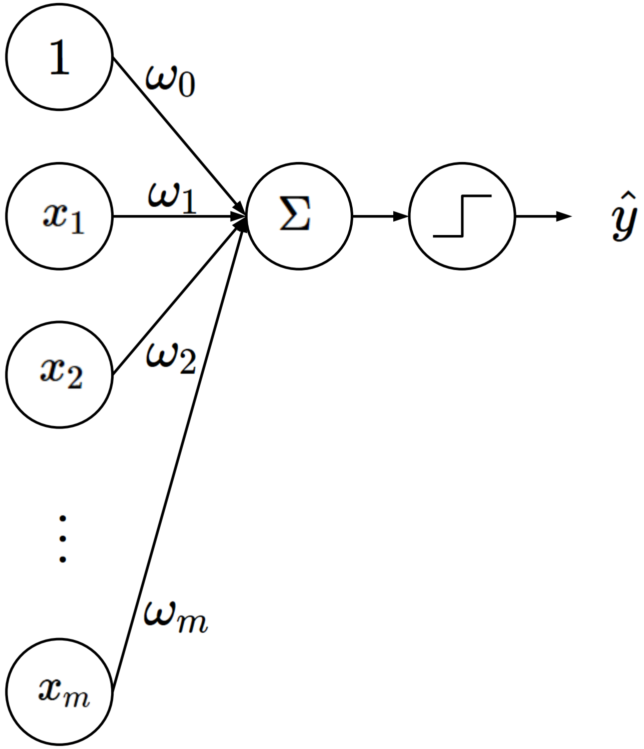
Perceptron: Example



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



Perceptron: Forward Propagation



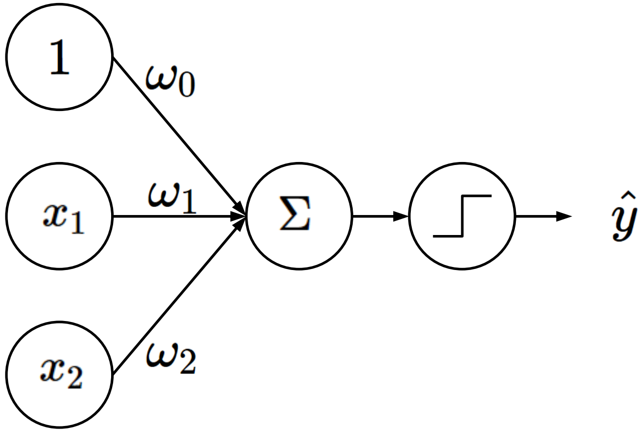
$$\hat{y} = g\left(\omega_0 + X^T\omega\right)$$

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

From Perceptron to MLP

Artificial Neural Networks: Perceptron

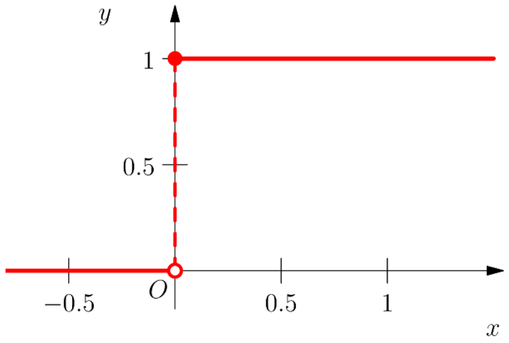
- Perceptron for $h(\theta)$ or $h(\omega)$
  - Neurons compute the weighted sum of their inputs
  - A neuron is activated or fired when the sum a is positive



$$a = \omega_0 + \omega_1 x_1 + \omega_2 x_2$$

$$\hat{y} = g(a) = \begin{cases} 1 & a > 0 \\ 0 & \text{otherwise} \end{cases}$$

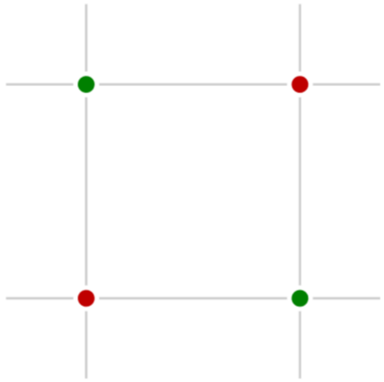
- A step function is not differentiable
- One neuron is often not enough
  - One hyperplane



Here, a step function is illustrated instead of a sign function

XOR Problem

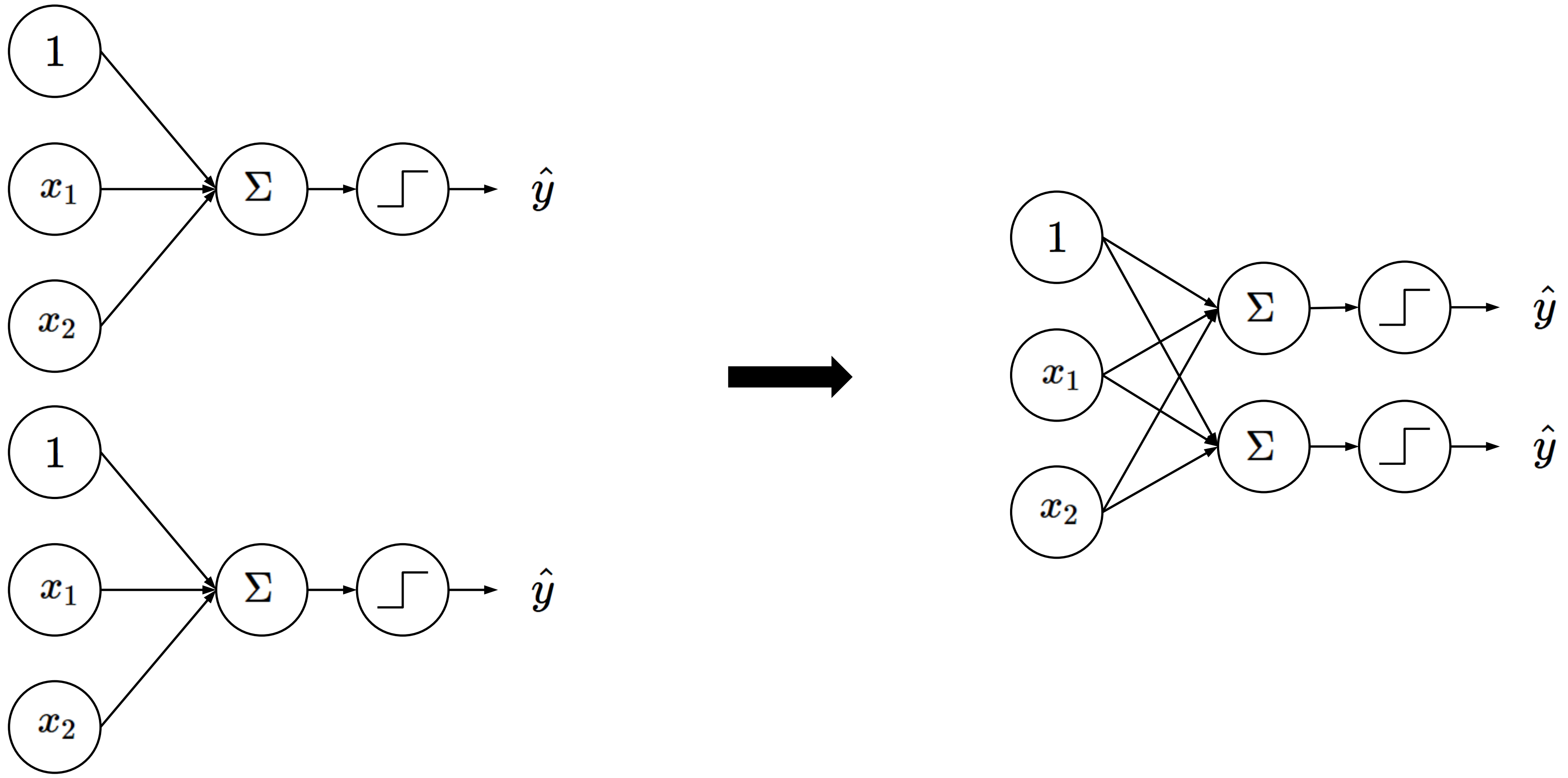
- The main weakness of linear predictors is their lack of capacity.
- For classification, the populations have to be linearly separable.



“xor”

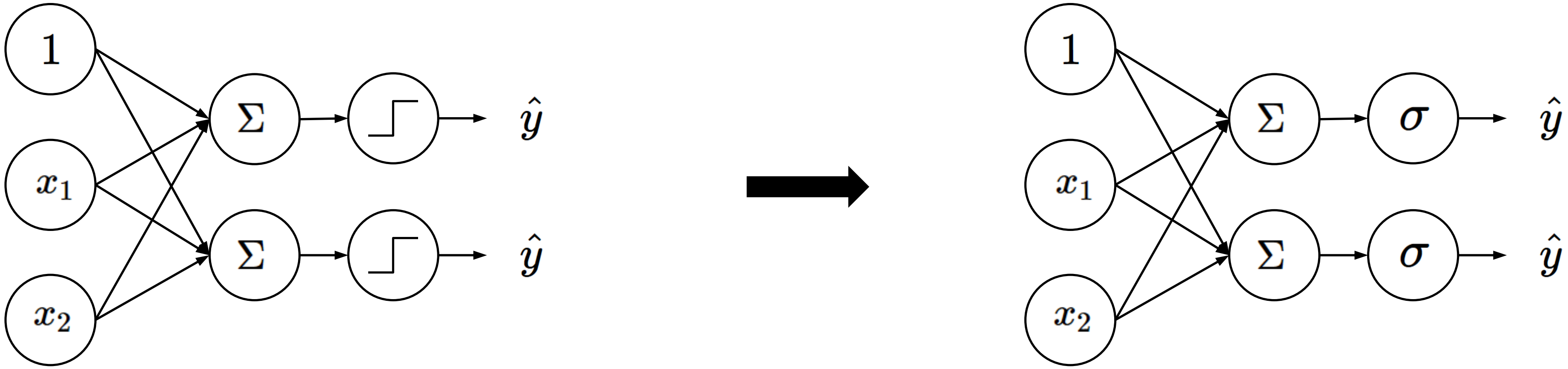
Artificial Neural Networks: MLP

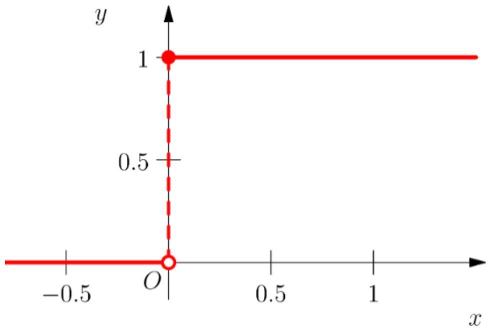
- Multi-layer Perceptron (MLP) = Artificial Neural Networks (ANN)
  - Multi neurons = multiple linear classification boundaries

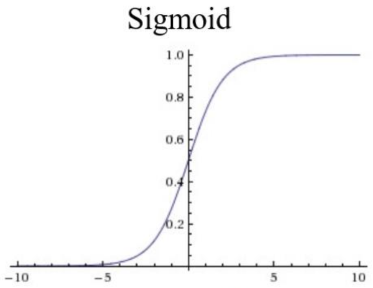


Artificial Neural Networks: Activation Function

- Differentiable nonlinear activation function



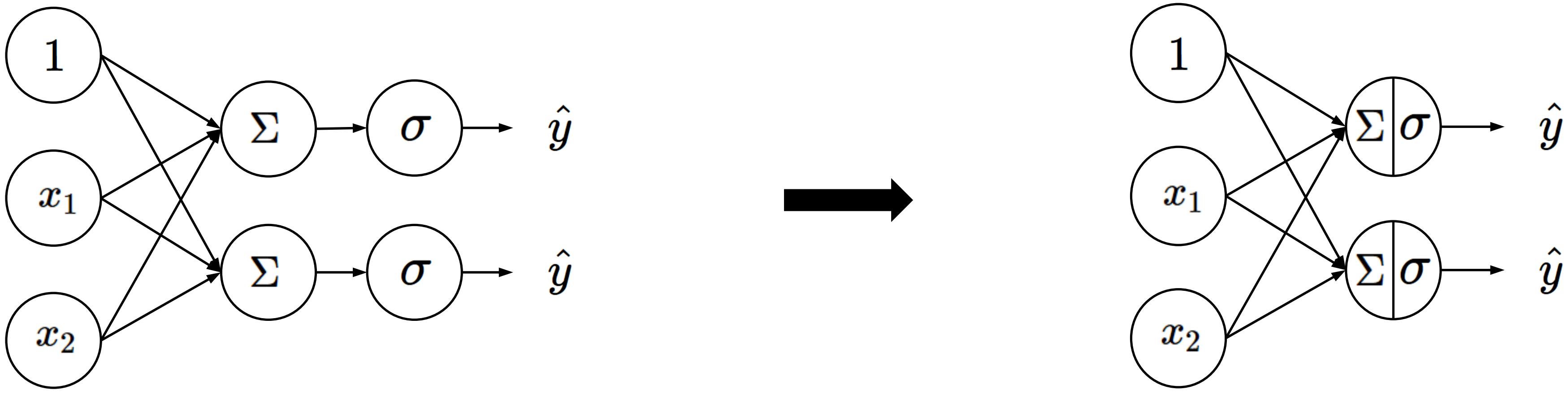




$$g(x) = \frac{1}{1 + e^{-x}}$$

Artificial Neural Networks

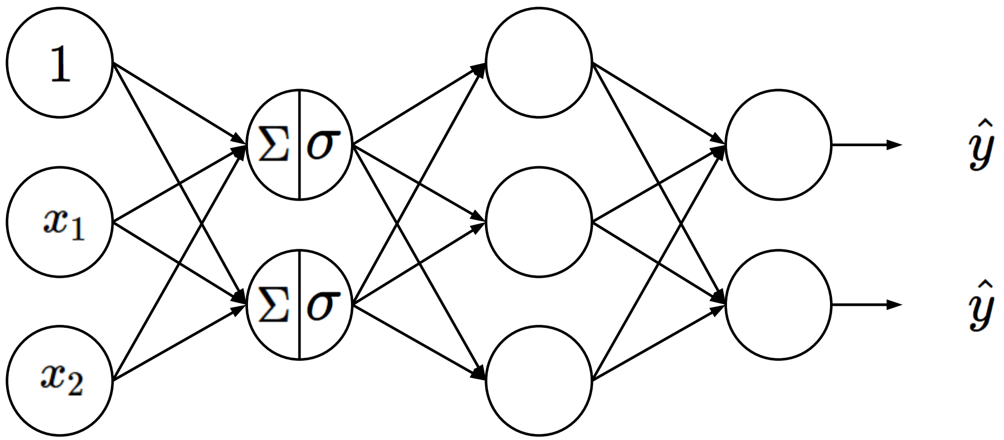
- In a compact representation



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

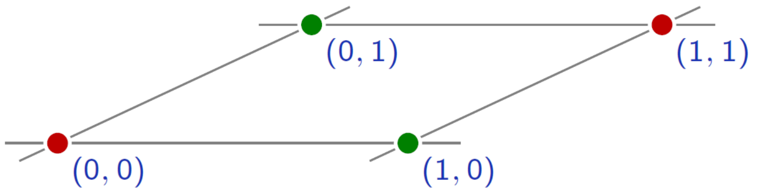


- Multi-layer perceptron
  - Features of features
  - Mapping of mappings

Another Perspective:
ANN as Kernel Learning

Nonlinear Mapping

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

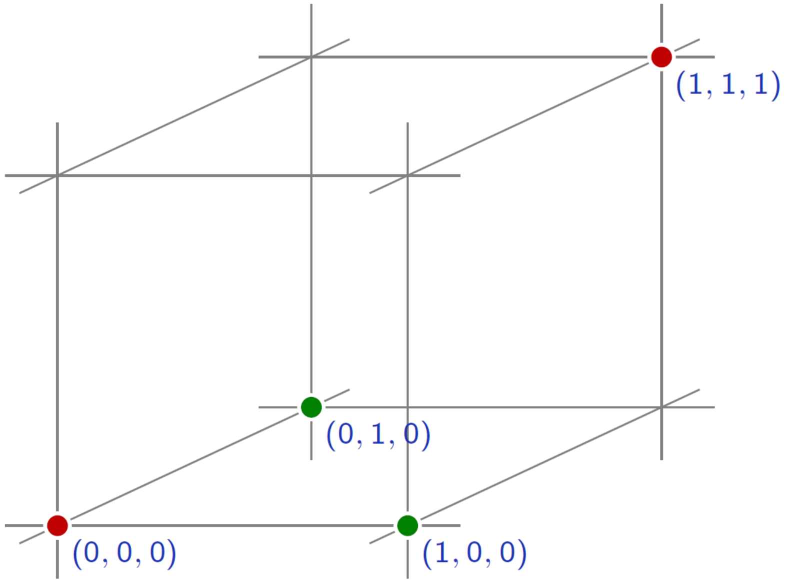


Source: Dr. Francois Fleuret at EPFL

Nonlinear Mapping

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

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

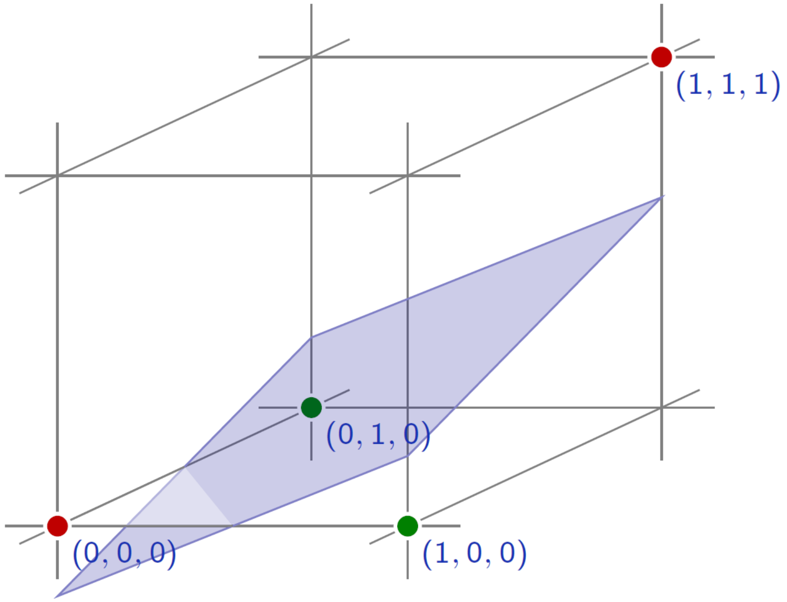


Source: Dr. Francois Fleuret at EPFL

Nonlinear Mapping

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

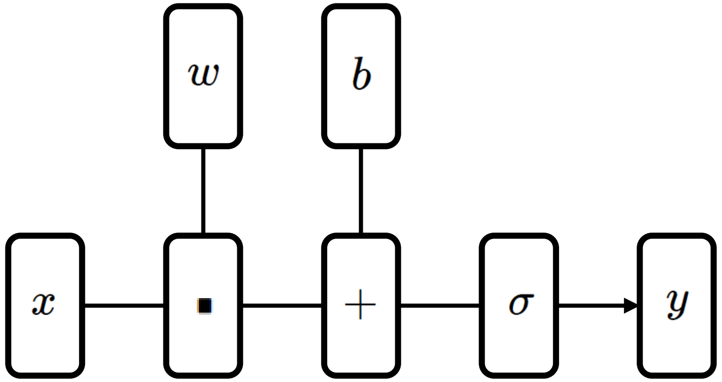
$$\phi : (x_u, x_v) \rightarrow (x_u, x_v, x_u x_v)$$



Neuron

- We can represent this “neuron” as follows:

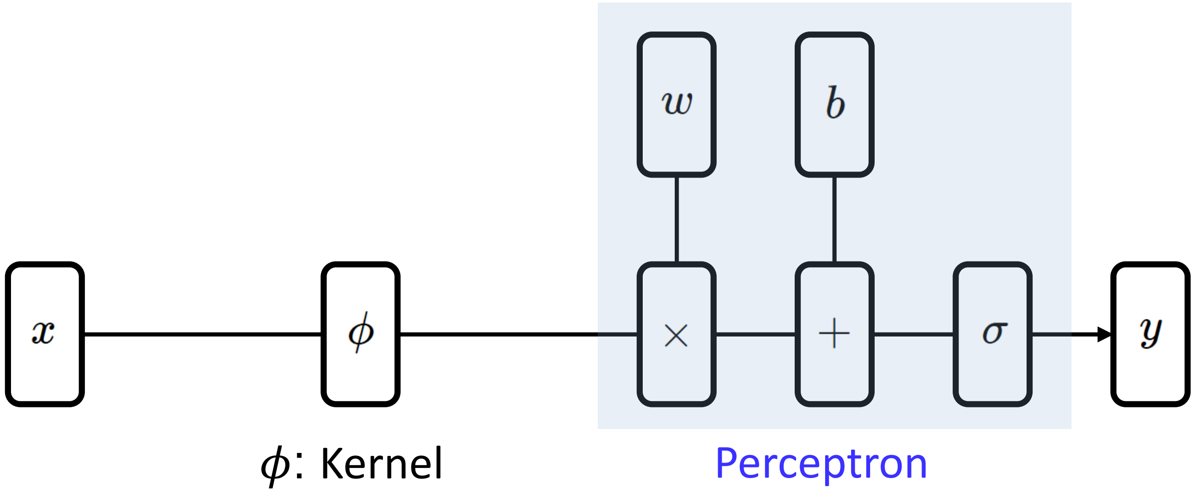
$$f(x) = \sigma(w \cdot x + b)$$



Kernel + Neuron

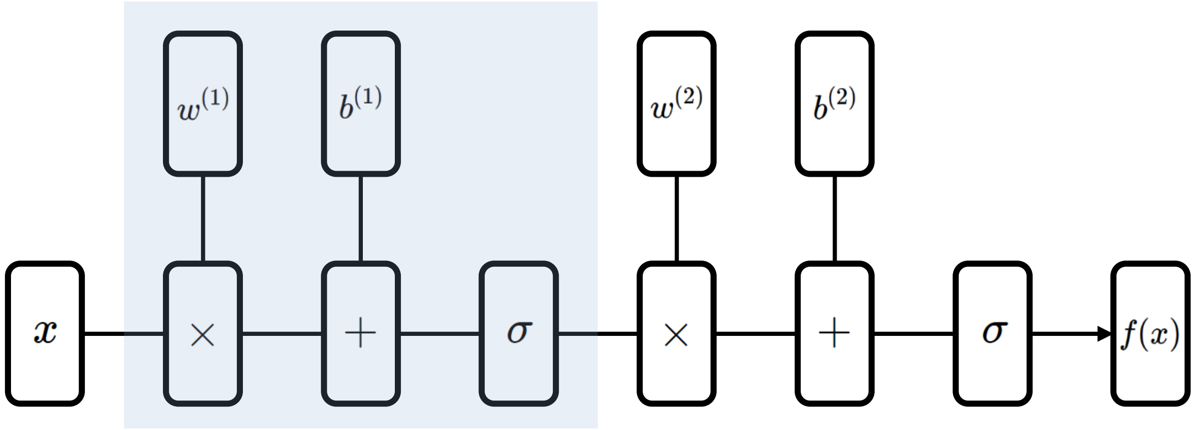
- Nonlinear mapping + neuron

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



Neuron + Neuron

- Nonlinear mapping can be represented by another neurons

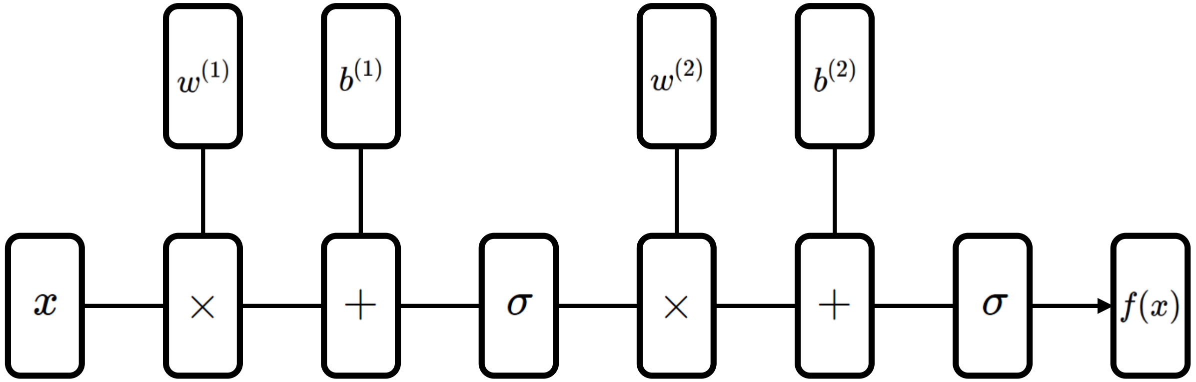


ϕ: Kernel

- Nonlinear Kernel
  - Nonlinear activation functions

Multi Layer Perceptron

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

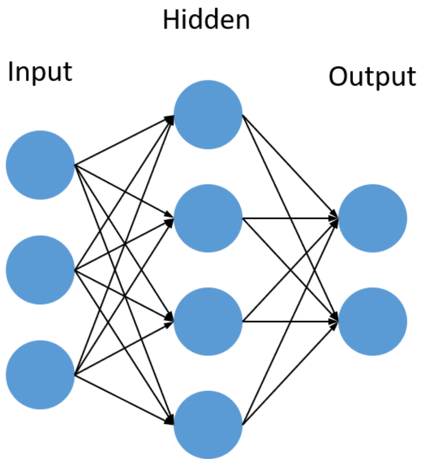


ϕ: Kernel

Summary

- Universal function approximator
- Universal function classifier

- Parameterized

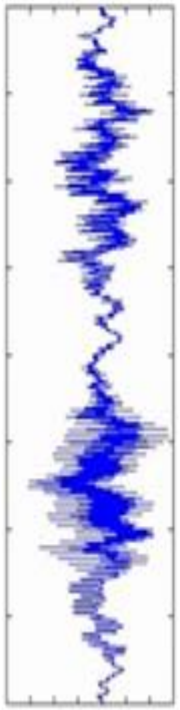


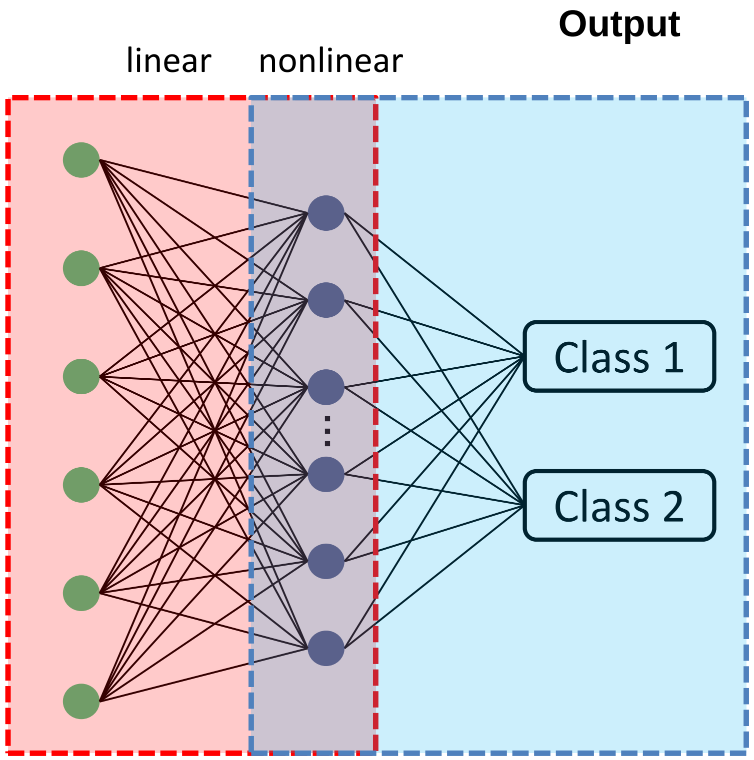
$$x \longrightarrow \boxed{\hat{y} = f_{\omega_1,\cdots,\omega_k}(x)} \longrightarrow y$$

Artificial Neural Networks

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

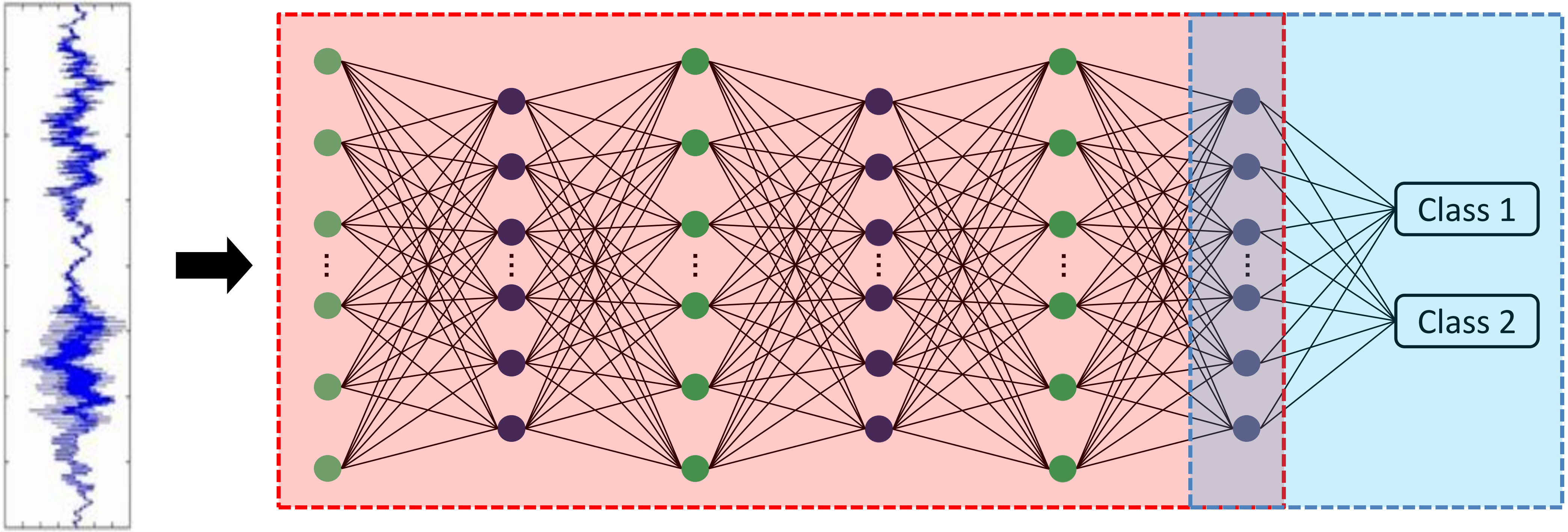
Input





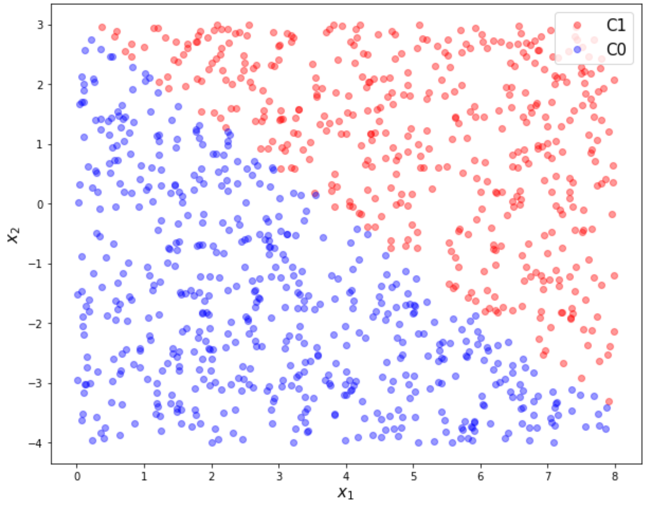
Deep Artificial Neural Networks

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

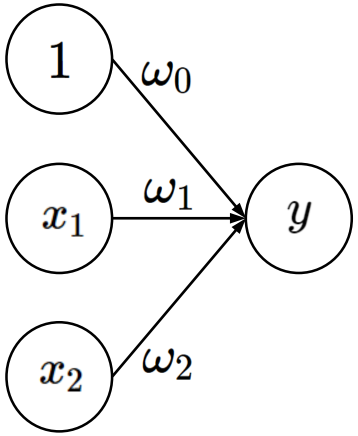


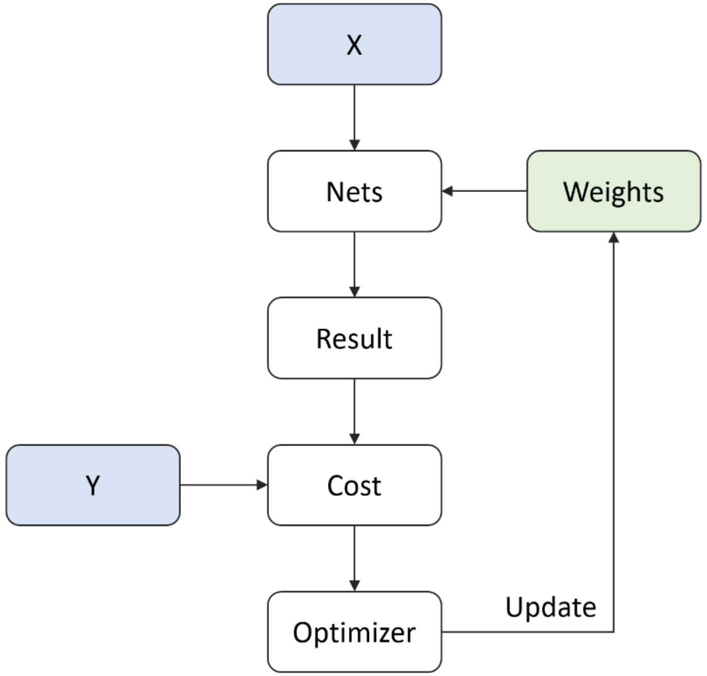
Looking at Parameters

Logistic Regression in a Form of Neural Network



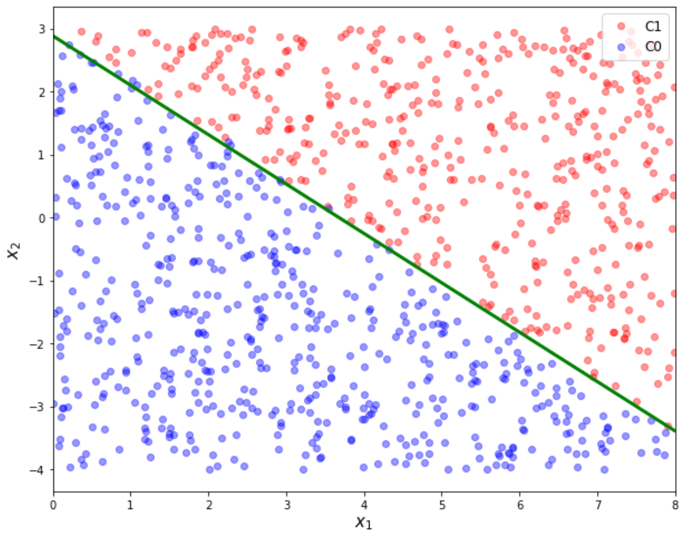
$$y = \sigma\left(\omega_0 + \omega_1 x_1 + \omega_2 x_2\right)$$





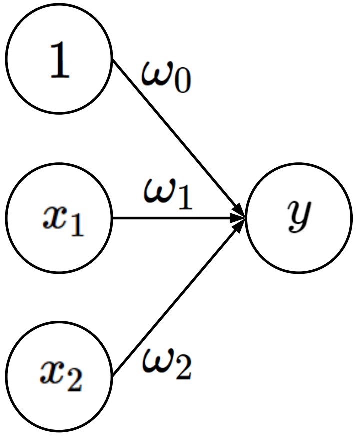
Logistic Regression in a Form of Neural Network

- Neural network convention

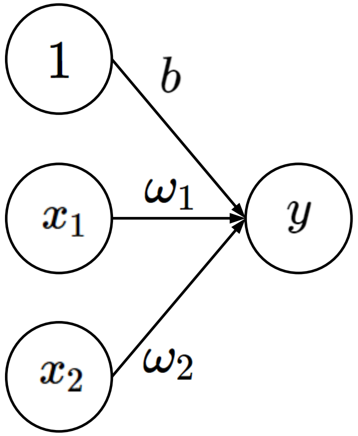


$$y = \sigma\left(\omega_0 + \omega_1 x_1 + \omega_2 x_2\right)$$

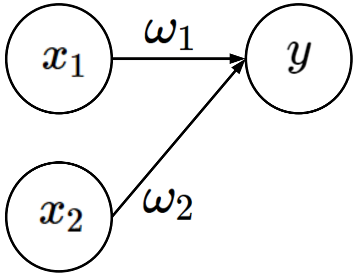
$$y = \sigma\left(b + \omega_1 x_1 + \omega_2 x_2\right)$$



$\omega_0 \longrightarrow b$



Do not indicate bias units

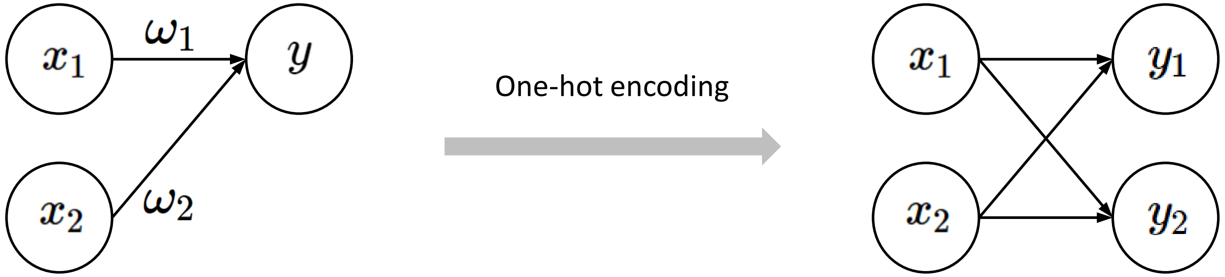



```
n_input = 2
n_output = 1
```

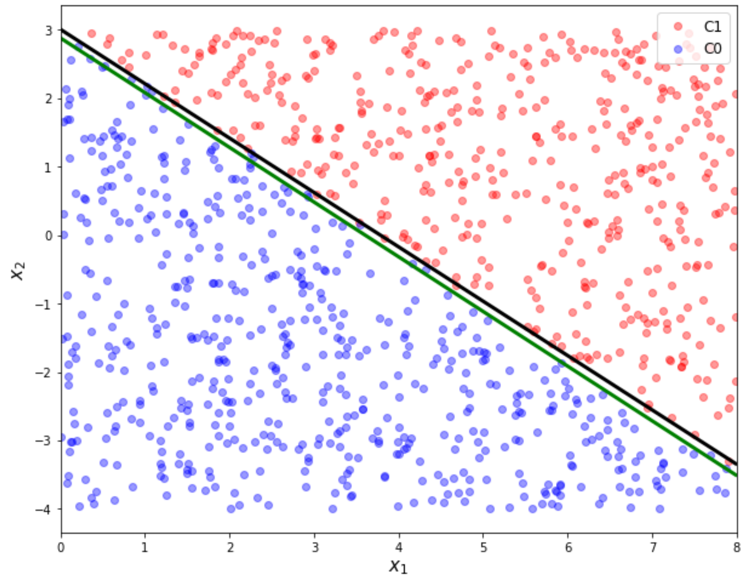
Logistic Regression in a Form of Neural Network

- One-hot encoding
  - One-hot encoding is a conventional practice for a multi-class classification

$$y^{(i)} \in \{1, 0\} \quad \Longrightarrow \quad y^{(i)} \in \{[0,1], [1,0]\}$$

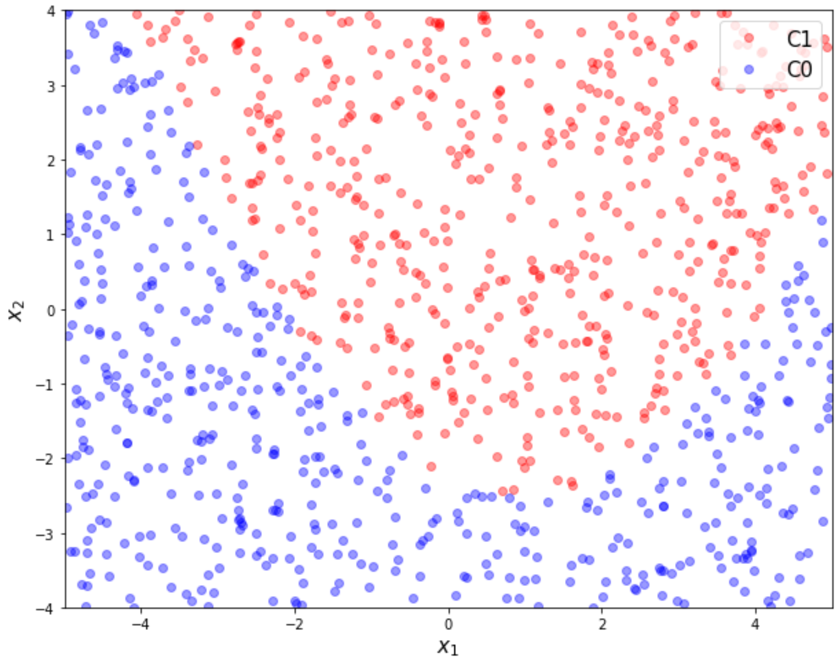


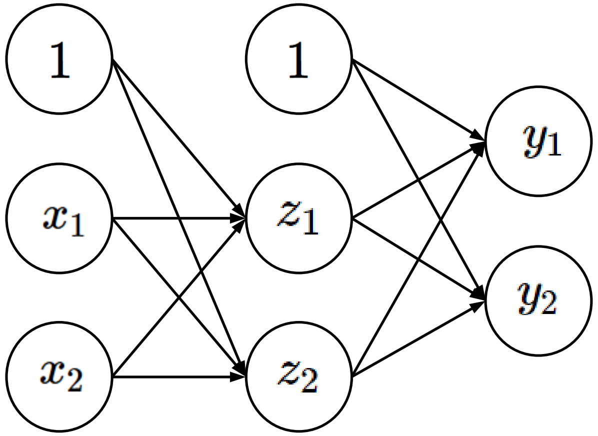

```
n_input = 2
n_output = 2
```



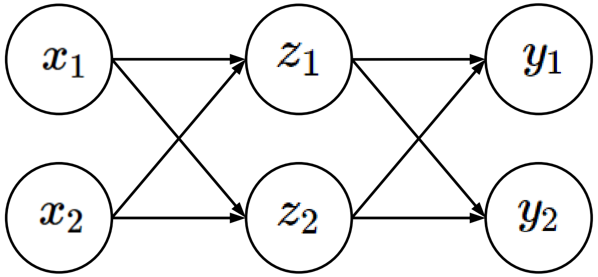
Nonlinearly Distributed Data

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





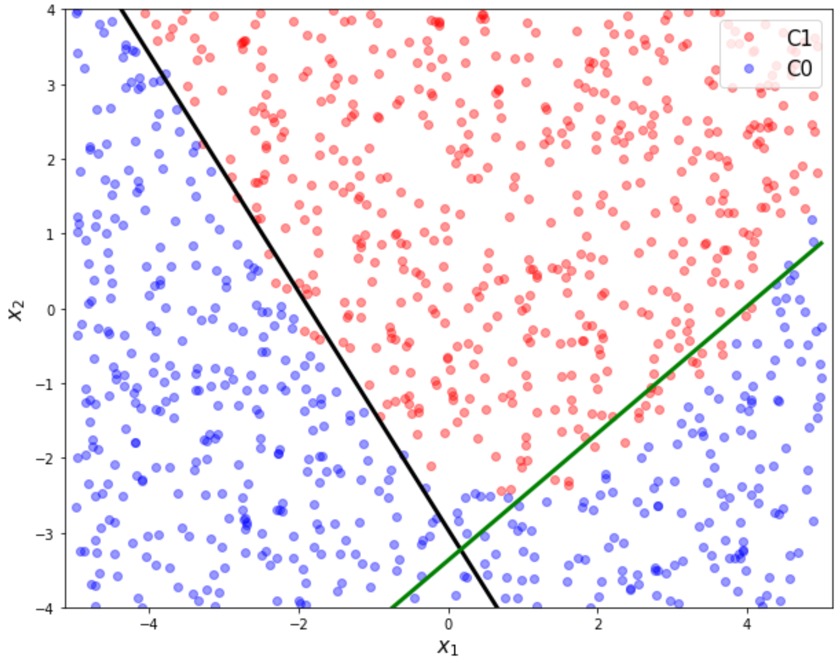
Do not include bias units

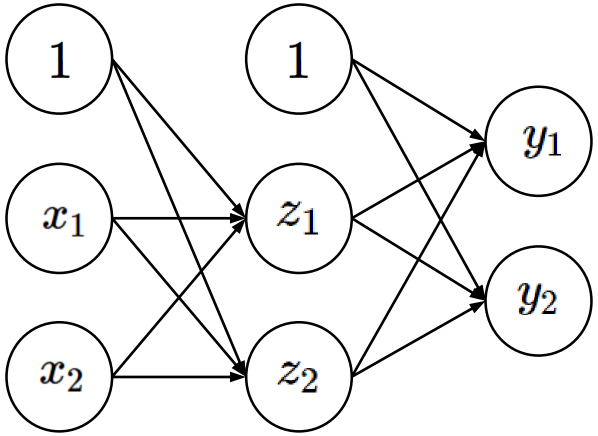



```
n_input = 2
n_hidden = 2
n_output = 2
```

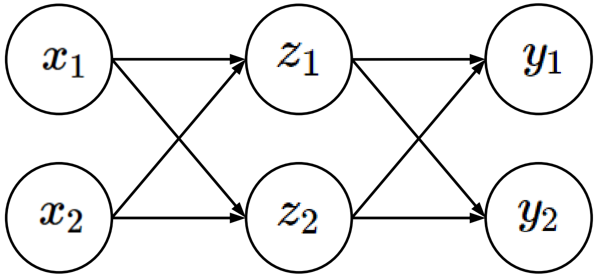
Multi Layers

- x space





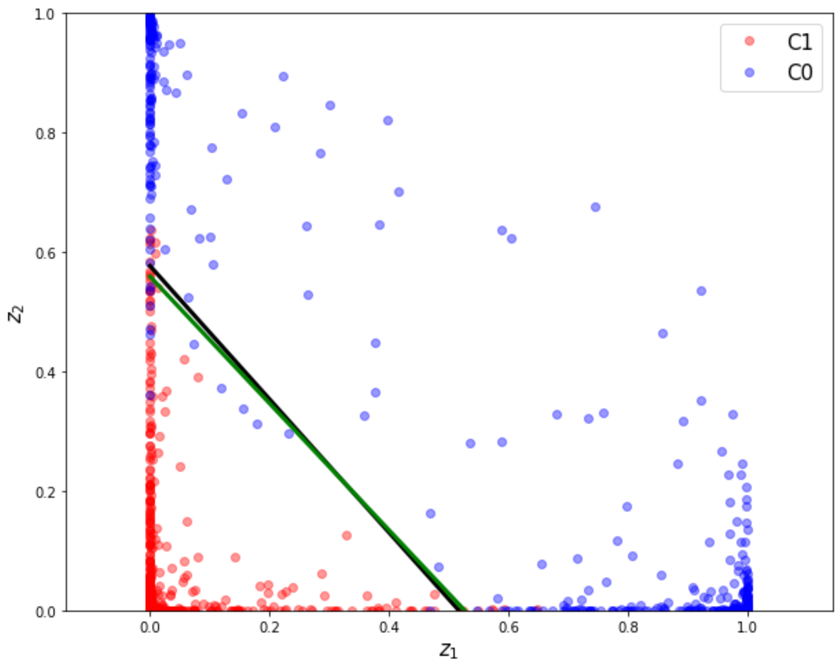
Do not include bias units

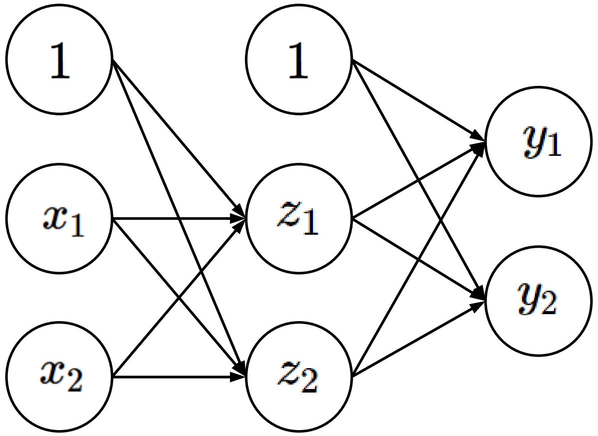



```
n_input  = 2
n_hidden  = 2
n_output  = 2
```

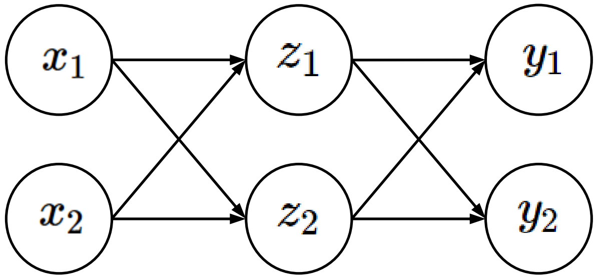
Multi Layers

- z space





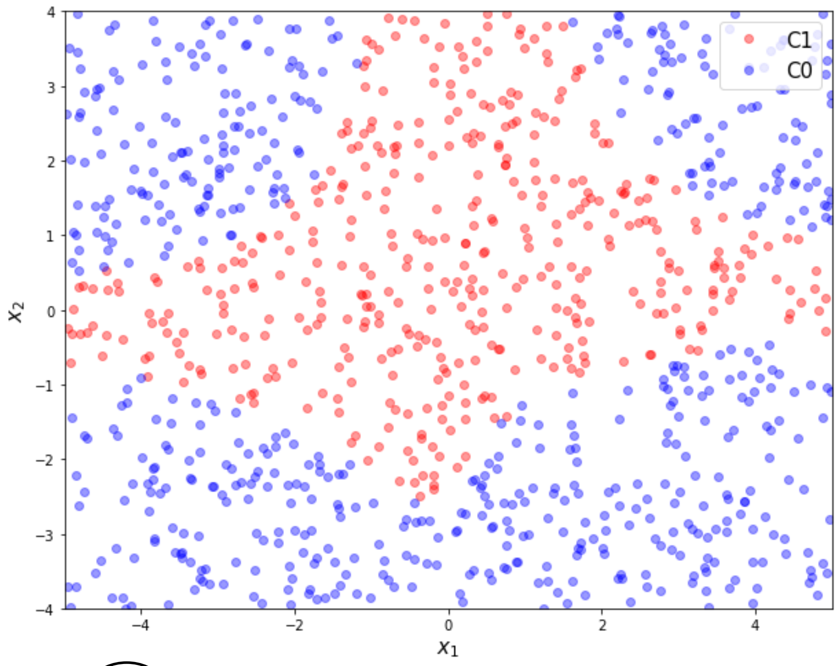
Do not include bias units

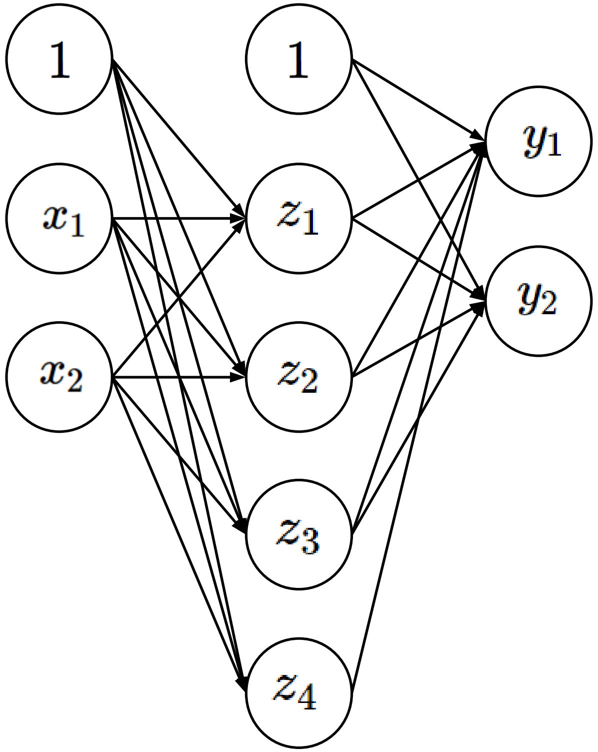



```
n_input = 2
n_hidden = 2
n_output = 2
```

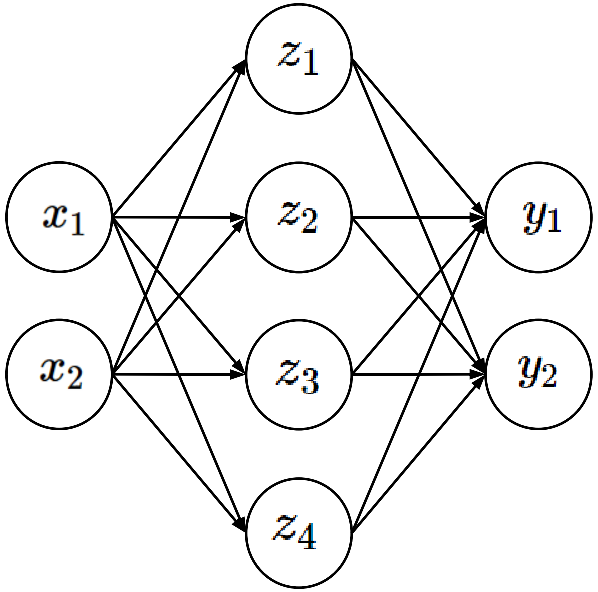
Nonlinearly Distributed Data

- More neurons in hidden layer





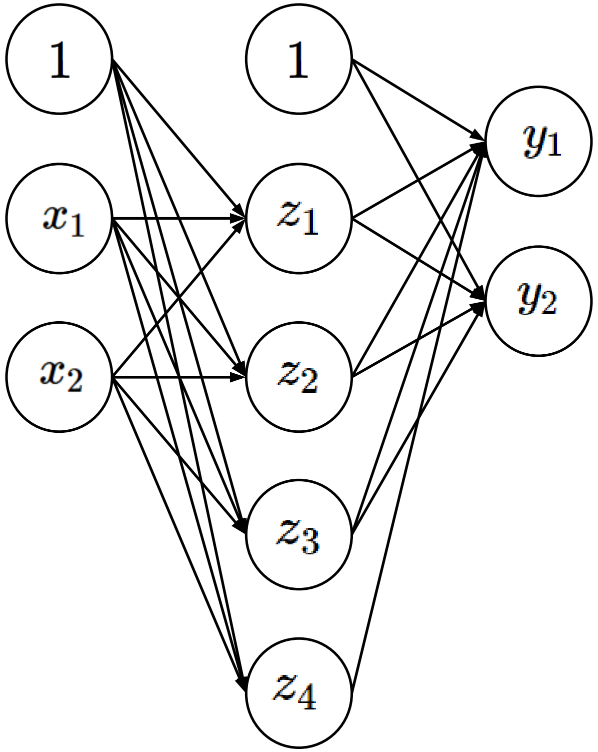
Do not include bias units



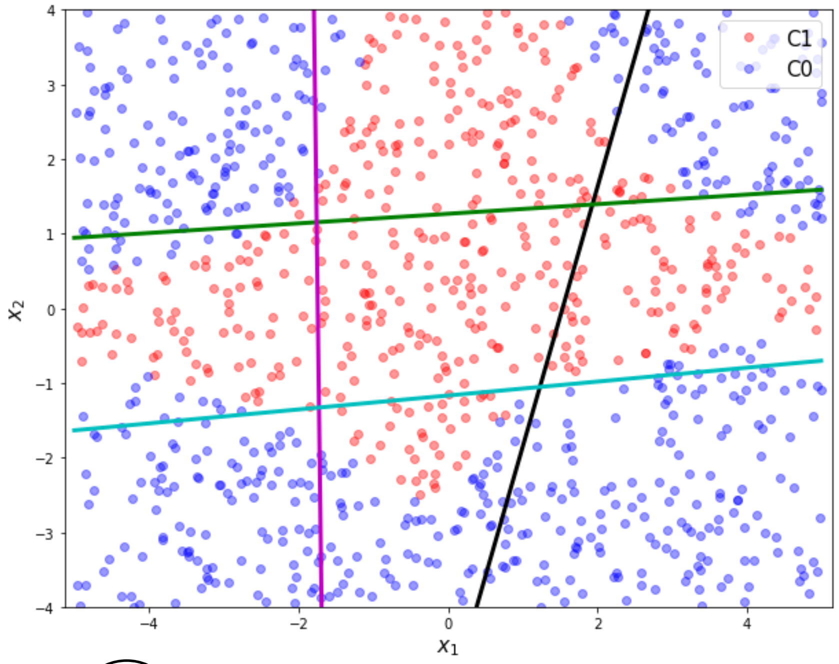

```
n_input = 2
n_hidden = 4
n_output = 2
```

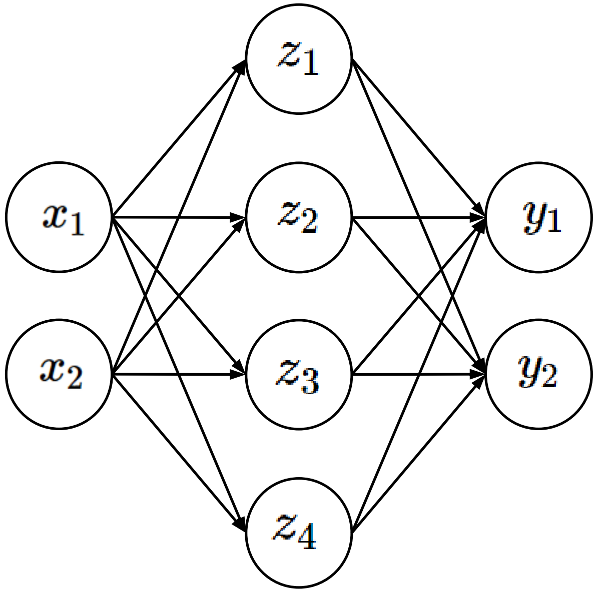
Multi Layers

- Multiple linear classification boundaries



Do not include bias units






```
n_input  = 2
n_hidden = 4
n_output = 2
```

(Artificial) Neural Networks: Training

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):

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

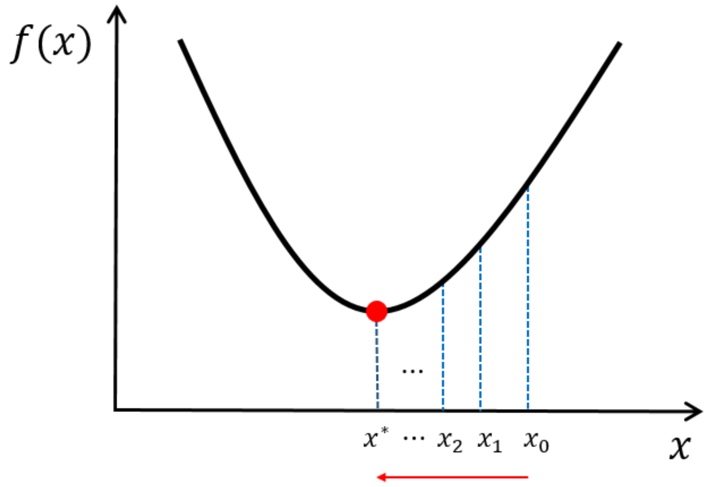
  - Cross entropy (for classification):

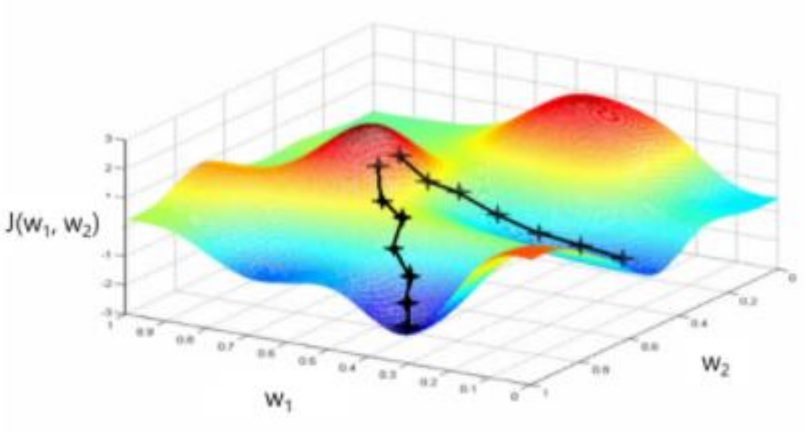
$$-\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log\left(h_{\omega}\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_{\omega}\left(x^{(i)}\right)\right)$$

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)

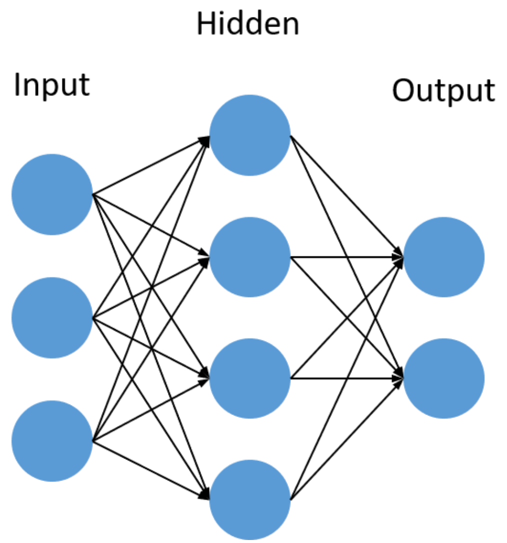
$$\omega \Leftarrow \omega - \alpha \nabla_{\omega} \ell \left(h_{\omega} \left(x^{(i)} \right), y^{(i)} \right)$$





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

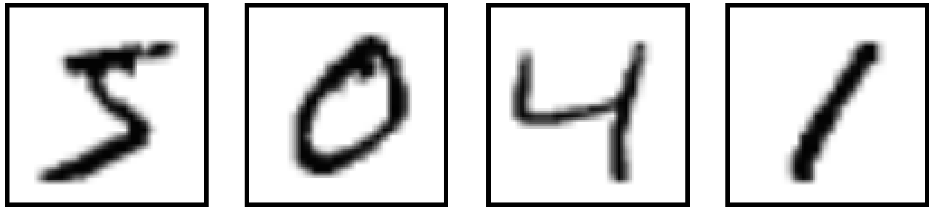


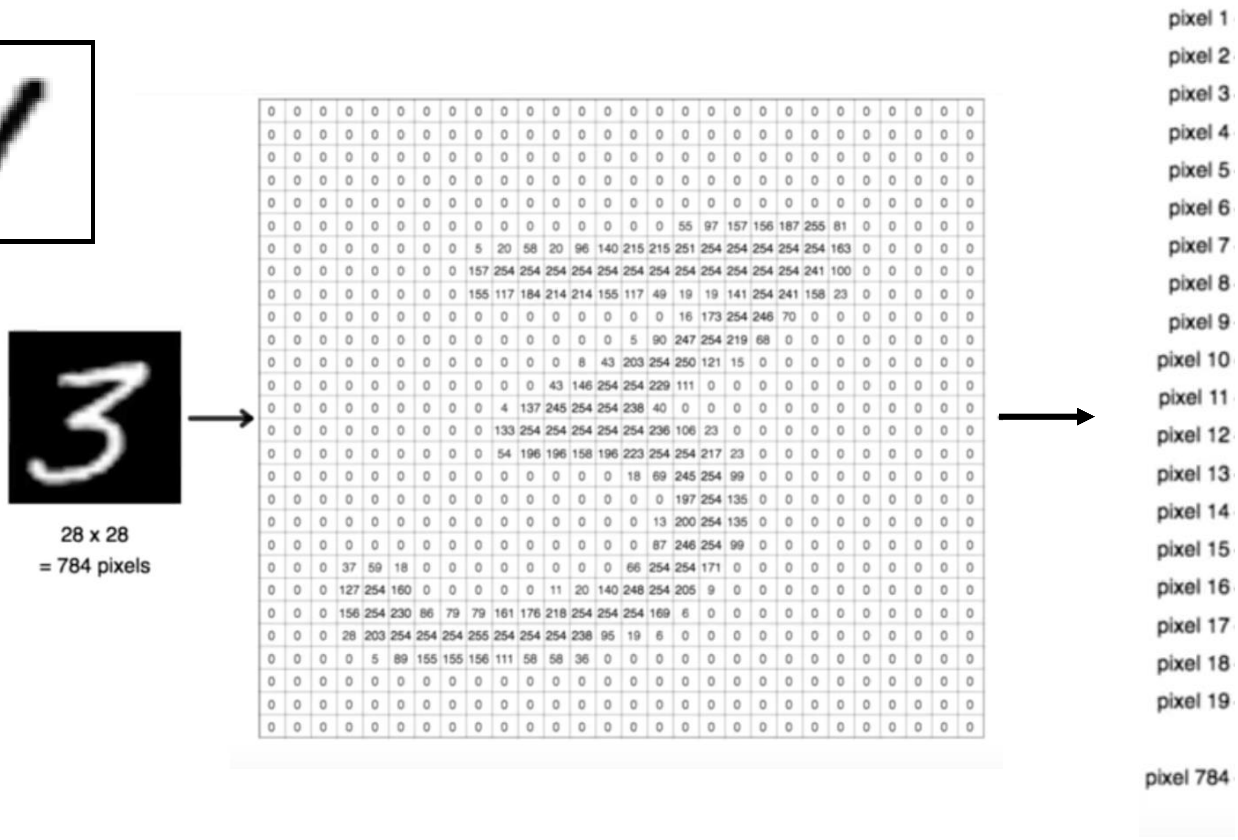
$$x \longrightarrow \boxed{\hat{y} = f_{\omega_1,\cdots,\omega_k}(x)} \longrightarrow y$$

ANN in TensorFlow: MNIST

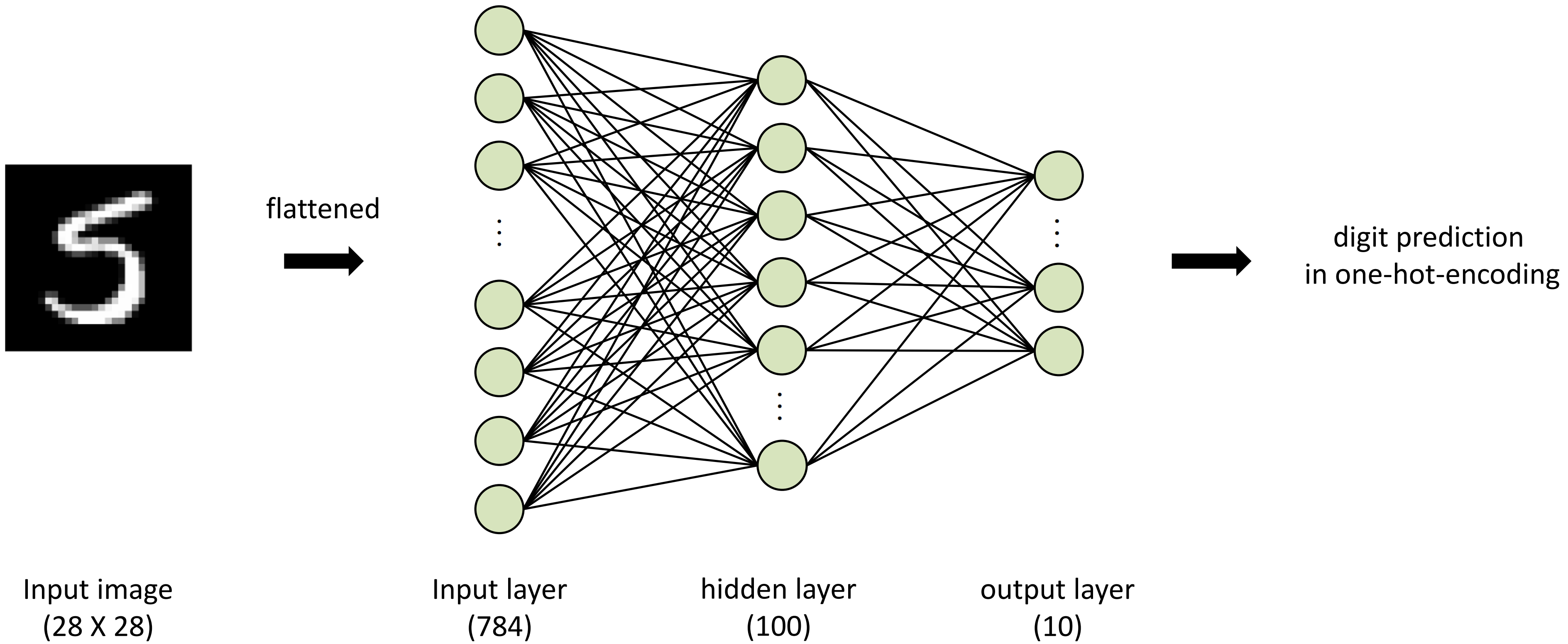
MNIST database

- 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

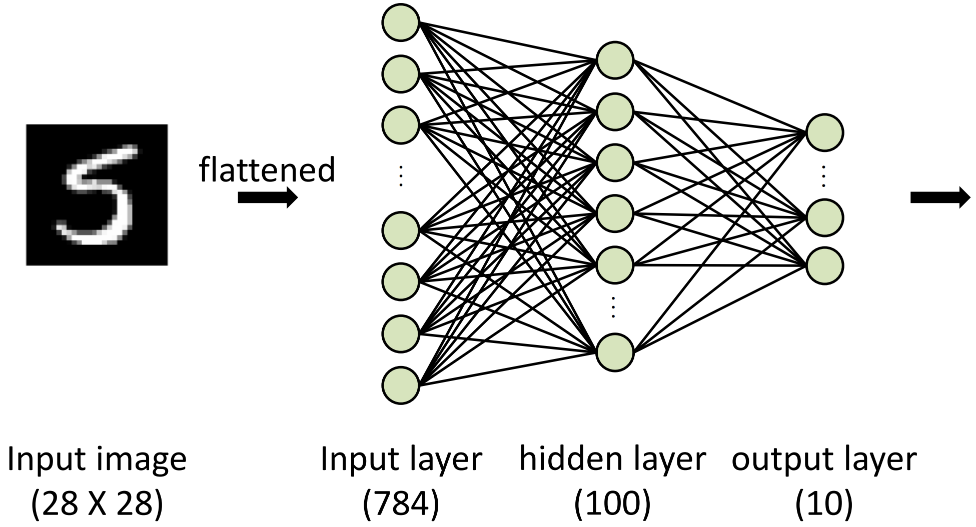


Implementation in Python

```python
mnist = tf.keras.datasets.mnist

(train_x, train_y), (test_x, test_y) = mnist.load_data()

train_x, test_x = train_x/255.0, test_x/255.0
```

```python
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape = (28, 28)),
    tf.keras.layers.Dense(units = 100, activation = 'relu'),
    tf.keras.layers.Dense(units = 10, activation = 'softmax')
])

model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy',
              metrics = ['accuracy'])

loss = model.fit(train_x, train_y, epochs = 5)
```

```python
test_loss, test_acc = model.evaluate(test_x, test_y)
```



Evaluation

```
test_img = test_x[np.random.choice(test_x.shape[0], 1)]

predict = model.predict_on_batch(test_img)
mypred = np.argmax(predict, axis = 1)
```