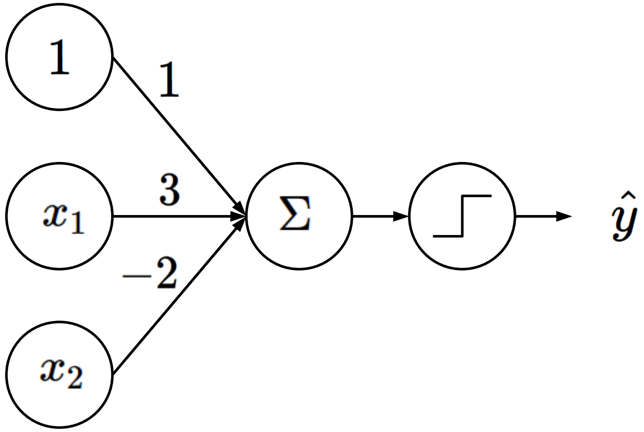
Artificial Neural Networks

Prof. Seungchul Lee
Industrial AI Lab.

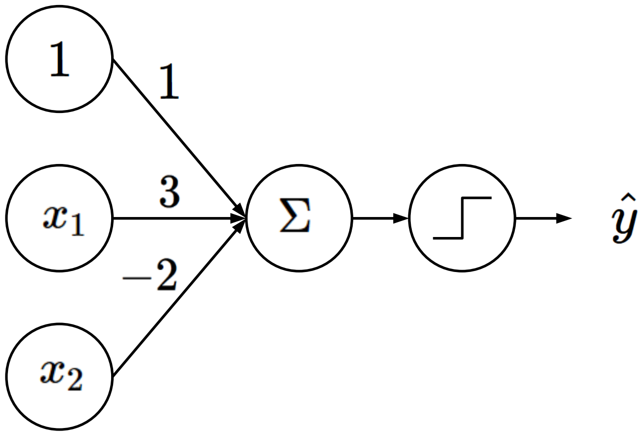
POSTECH

Perceptron: Example

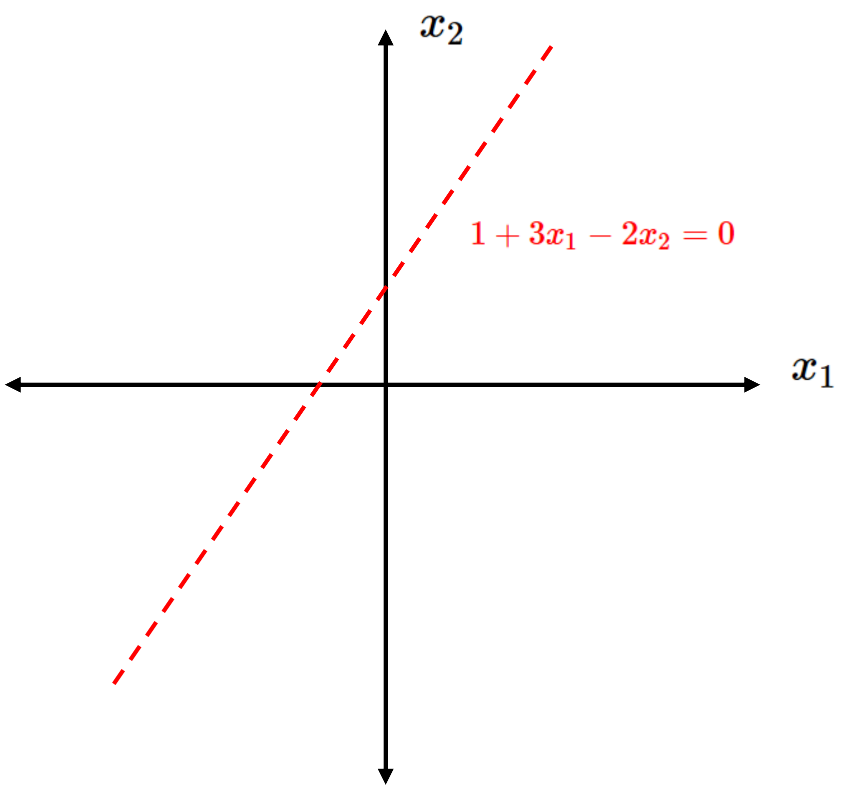


$$
\begin{aligned}
\hat{y} &= g\left(\omega_0 + X^T \omega\right) \\
&= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \\
&= g\left(1 + 3x_1 - 2x_2\right)
\end{aligned}
$$

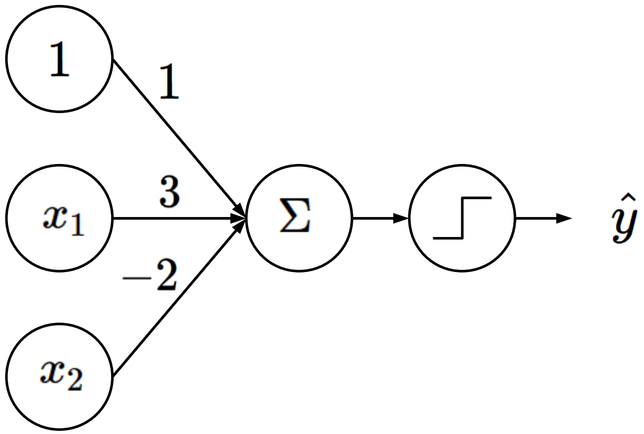
Perceptron: Example



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

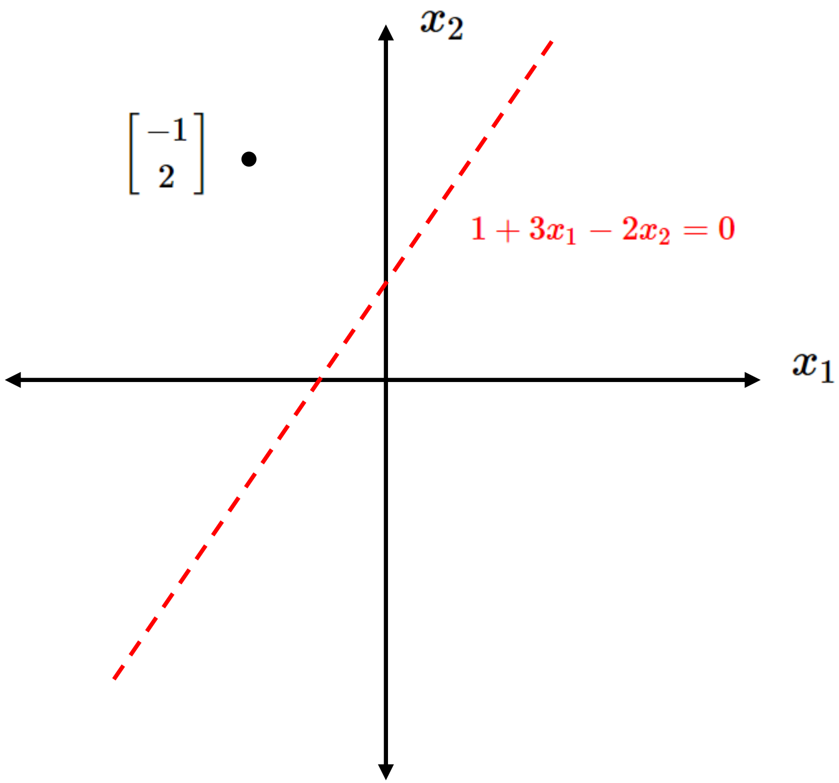


Perceptron: Example

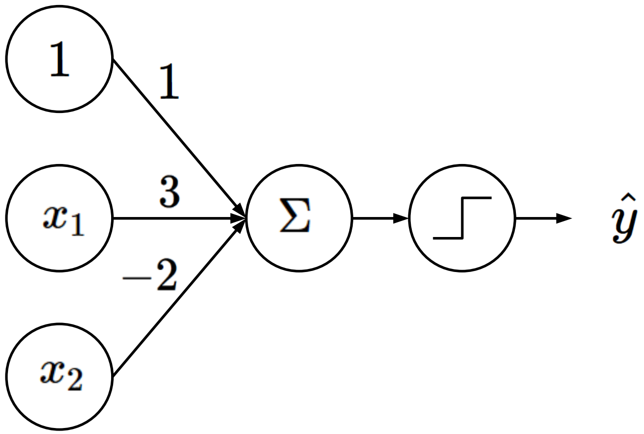


$$\hat{y} = g\left(1 + 3 \times (-1) - 2 \times 2\right) = g(-6) = -1$$

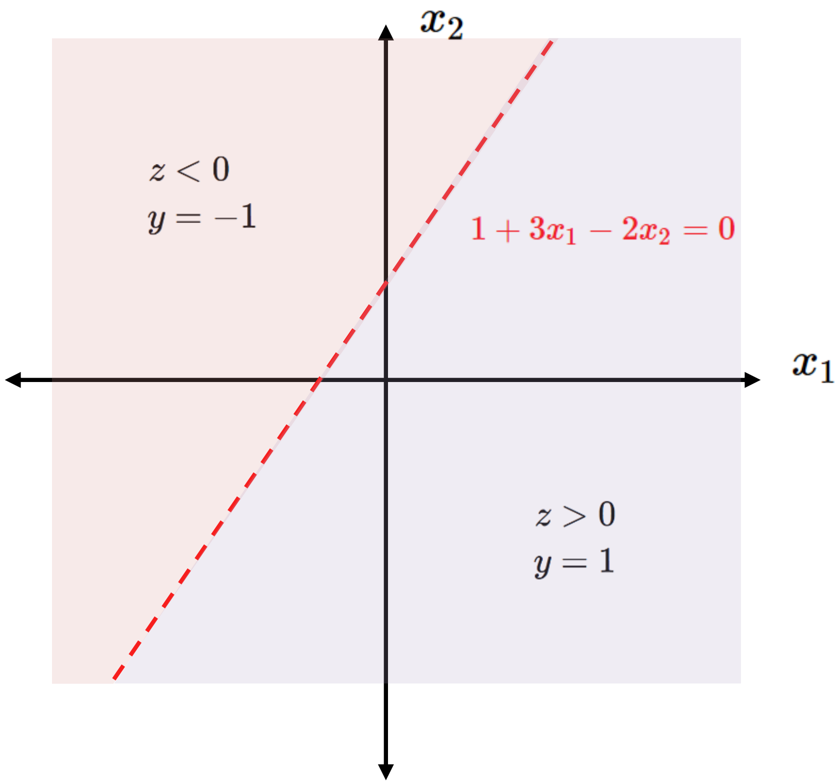
$$\hat{y} = g\left(1 + 3x_1 - 2x_2\right)$$



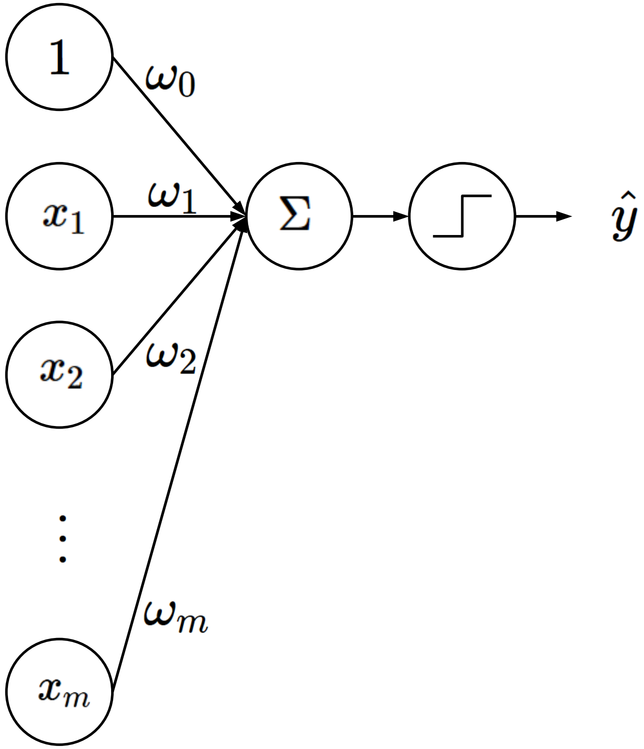
Perceptron: Example



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



Perceptron: Forward Propagation



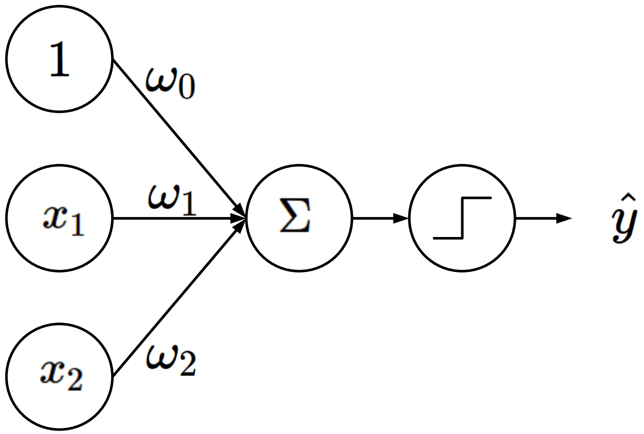
$$\hat{y} = g\left(\omega_0 + X^T\omega\right)$$

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

From Perceptron to MLP

Artificial Neural Networks: Perceptron

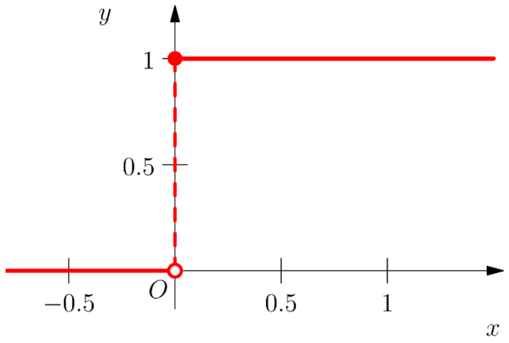
- Perceptron for $h(\theta)$ or $h(\omega)$
  - Neurons compute the weighted sum of their inputs
  - A neuron is activated or fired when the sum a is positive



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

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

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

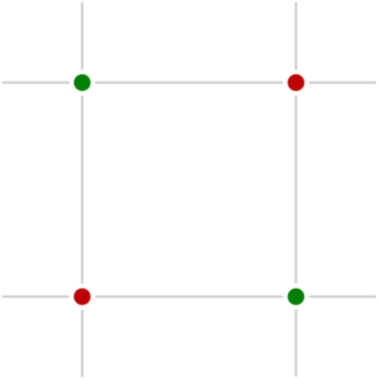


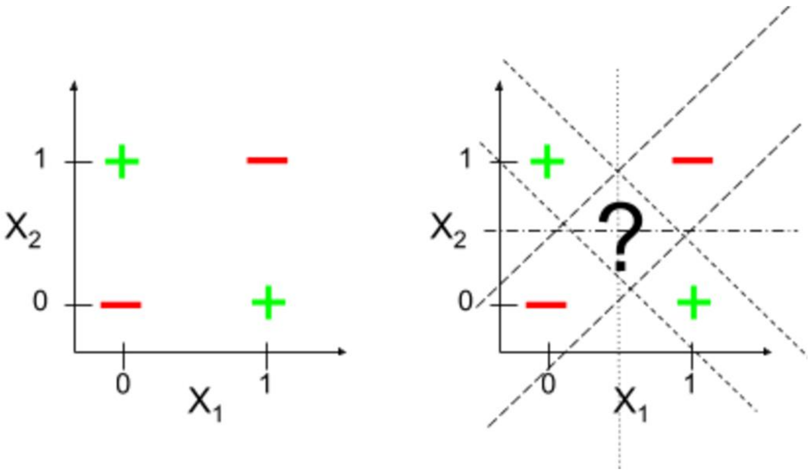
Here, a step function is illustrated instead of a sign function

XOR Problem

- 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

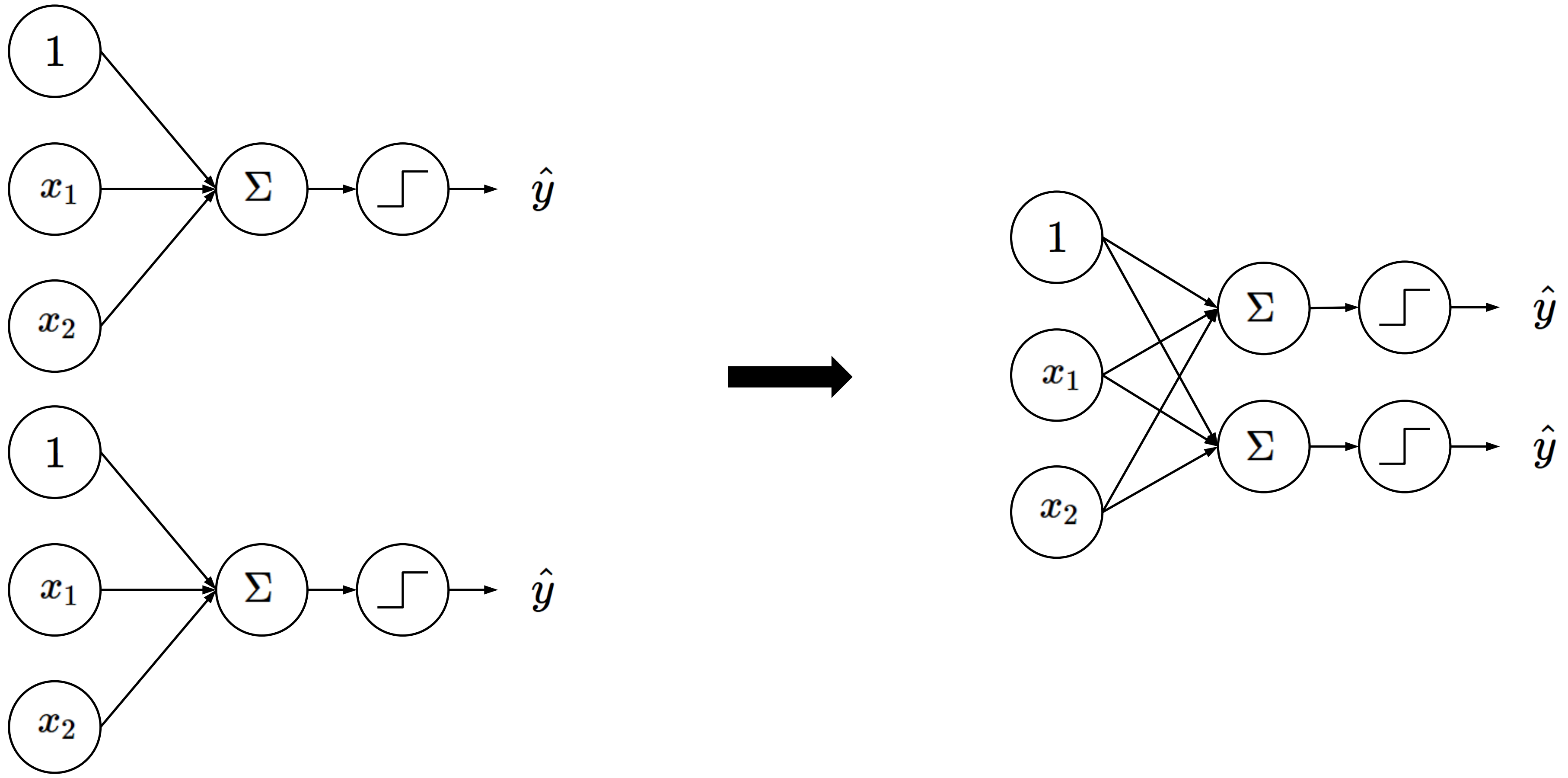




- Single neuron = one linear classification boundary

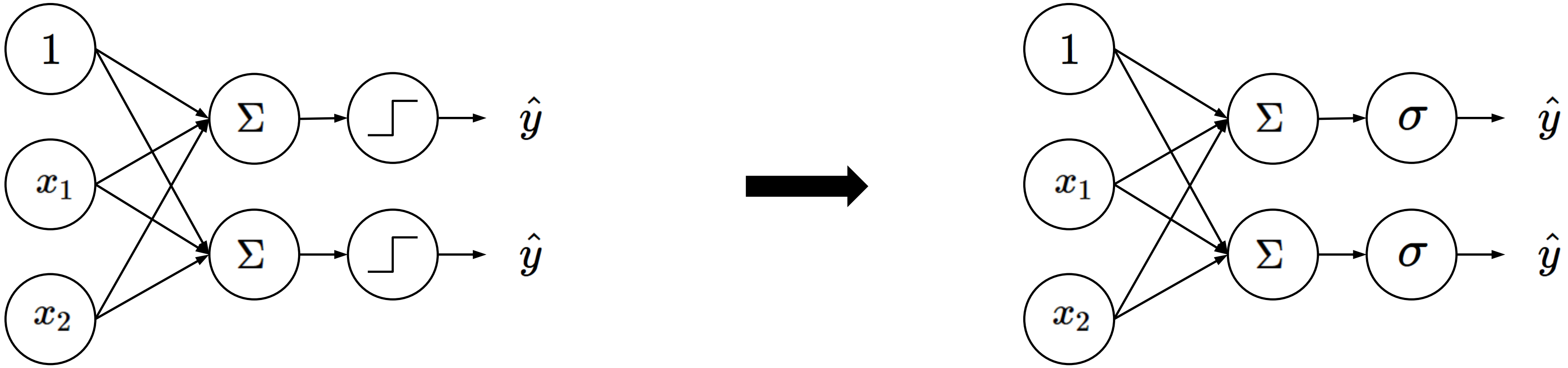
Artificial Neural Networks: MLP

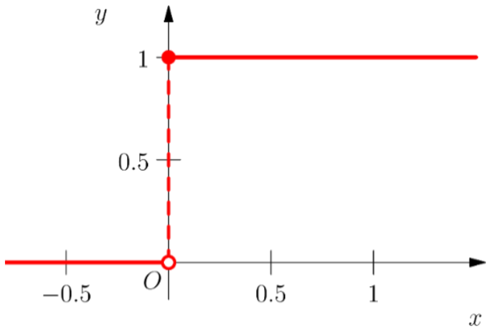
- Multi-layer Perceptron (MLP) = Artificial Neural Networks (ANN)
  - Multi neurons = multiple linear classification boundaries

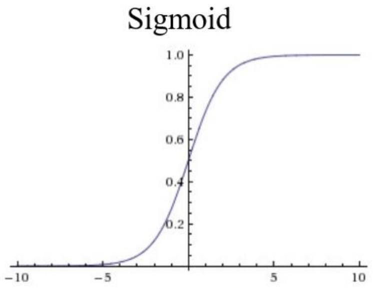


Artificial Neural Networks: Activation Function

- Differentiable nonlinear activation function





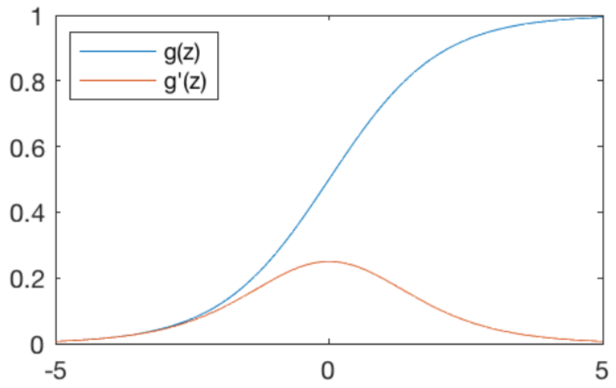


$$g(x) = \frac{1}{1 + e^{-x}}$$

Common Activation Functions

Discuss later

Sigmoid Function

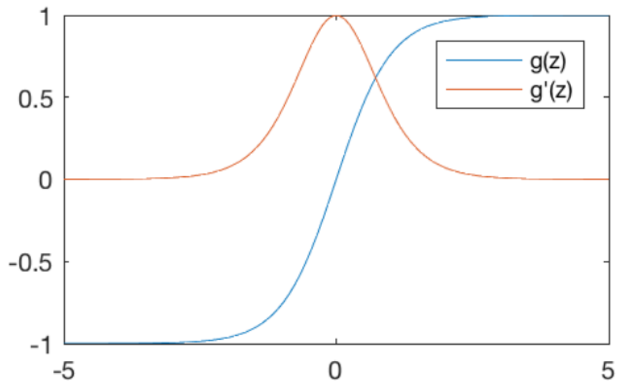


$$g(z) = \frac{1}{1 + e^{-z}}$$

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

`tf.nn.sigmoid(z)`

Hyperbolic Tangent

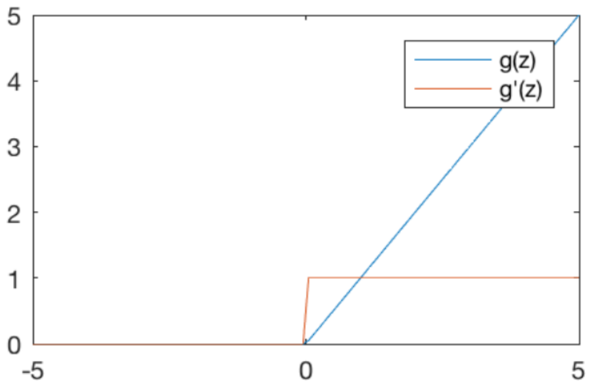


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

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

`tf.nn.tanh(z)`

Rectified Linear Unit (ReLU)



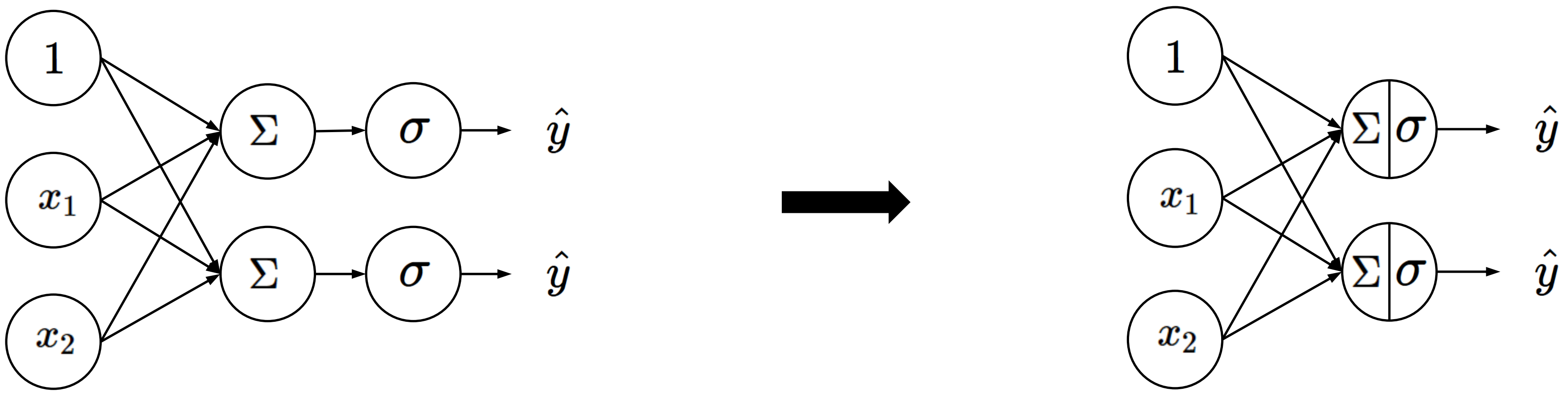
$$g(z) = \max(0, z)$$

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

`tf.nn.relu(z)`

Artificial Neural Networks

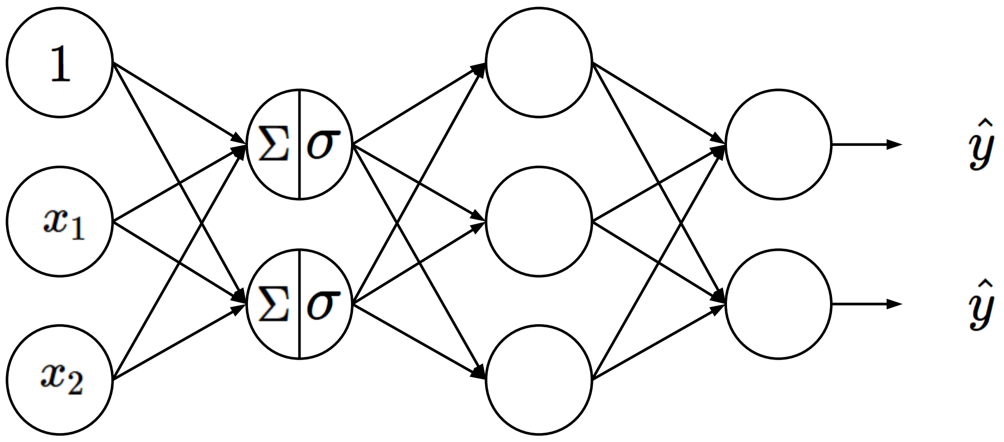
- In a compact representation



Artificial Neural Networks

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

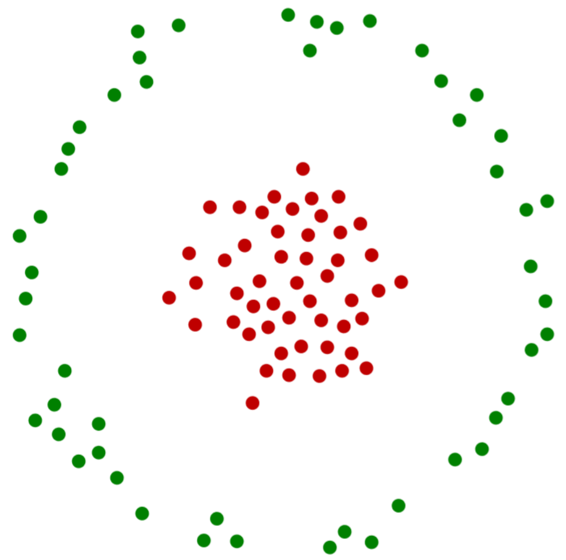
  $\Longrightarrow$ perceptron with many layers and units

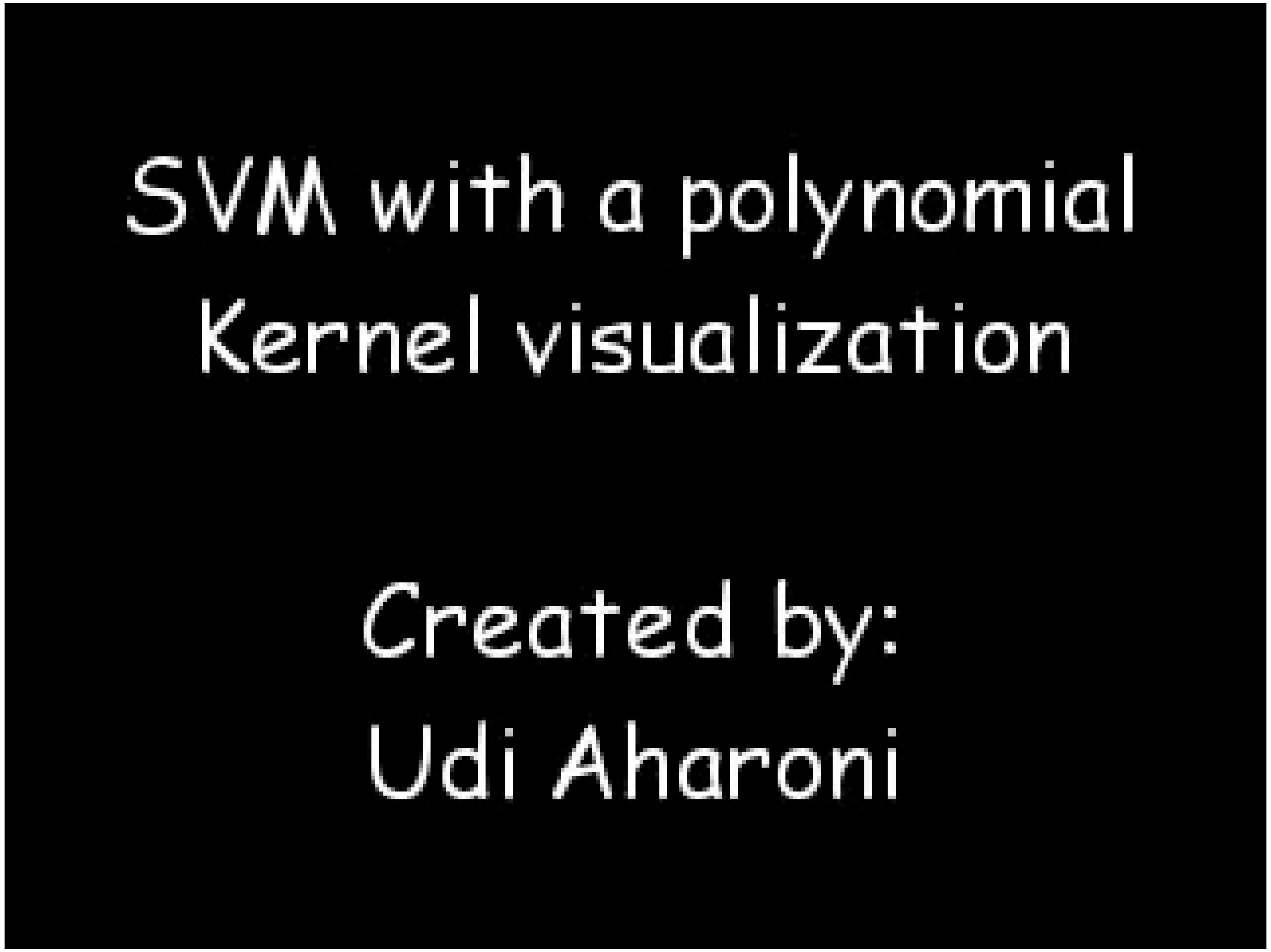


- Multi-layer perceptron
  - Features of features
  - Mapping of mappings

Another Perspective:
ANN as Kernel Learning

Nonlinear Classification



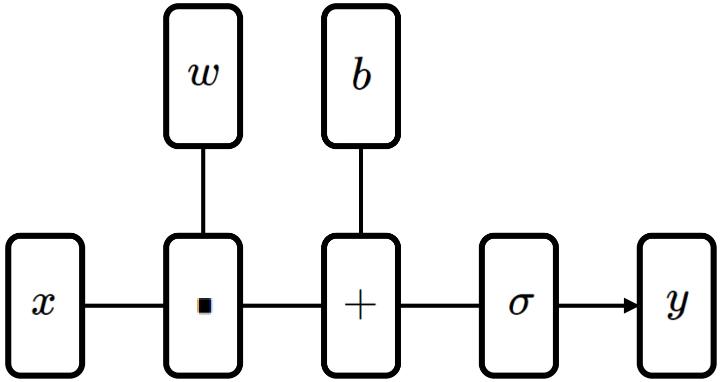


https://www.youtube.com/watch?v=3liCbRZPrZA

Neuron

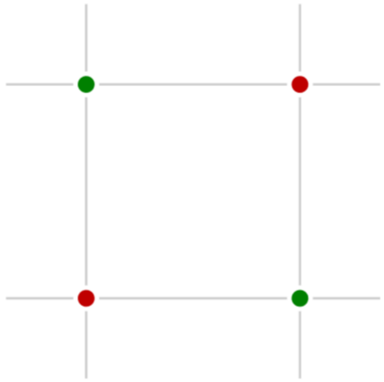
- We can represent this “neuron” as follows:

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



XOR Problem

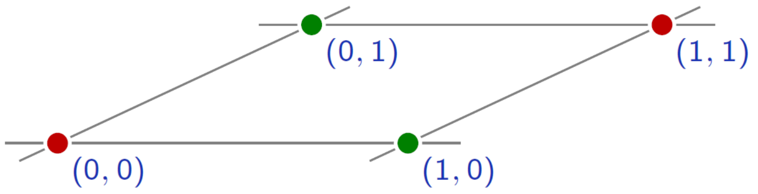
- The main weakness of linear predictors is their lack of capacity.
- For classification, the populations have to be linearly separable.



"xor"

Nonlinear Mapping

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

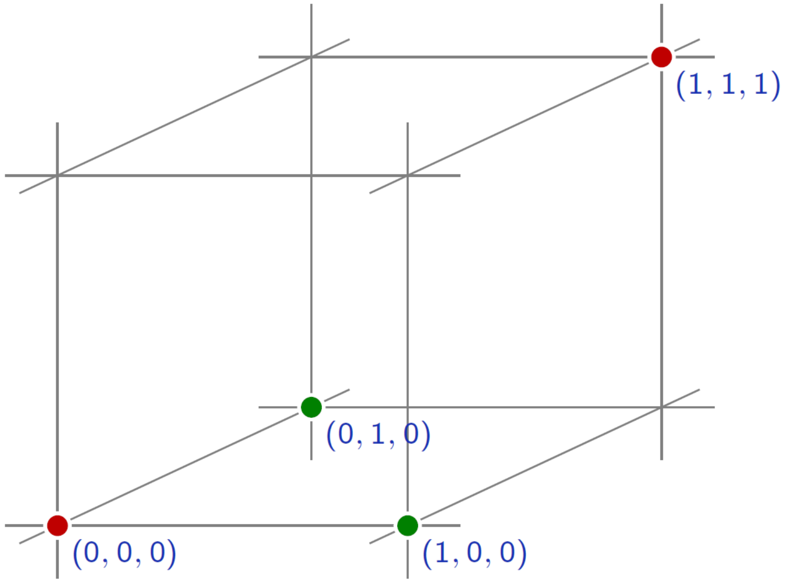


Source: Dr. Francois Fleuret at EPFL

Nonlinear Mapping

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

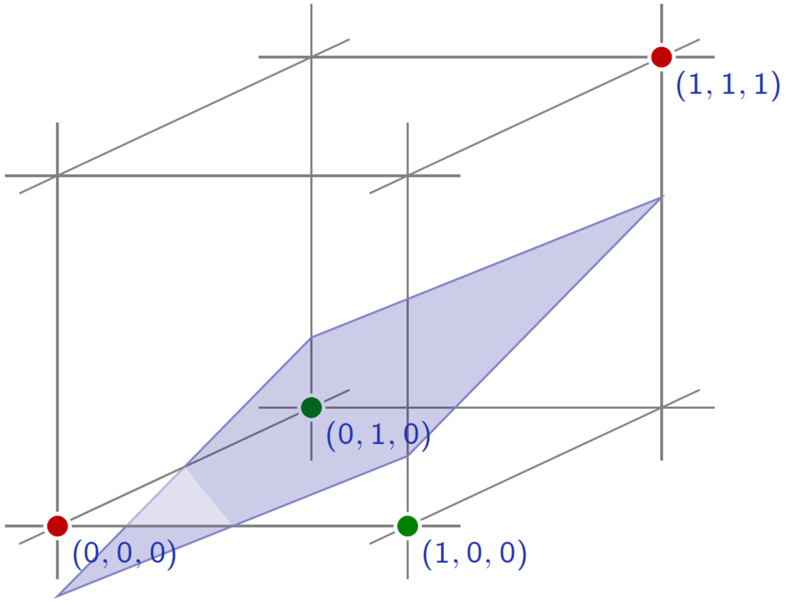
$$\phi : (x_u, x_v) \rightarrow (x_u, x_v, x_u x_v)$$



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 not be separable by a linear boundary

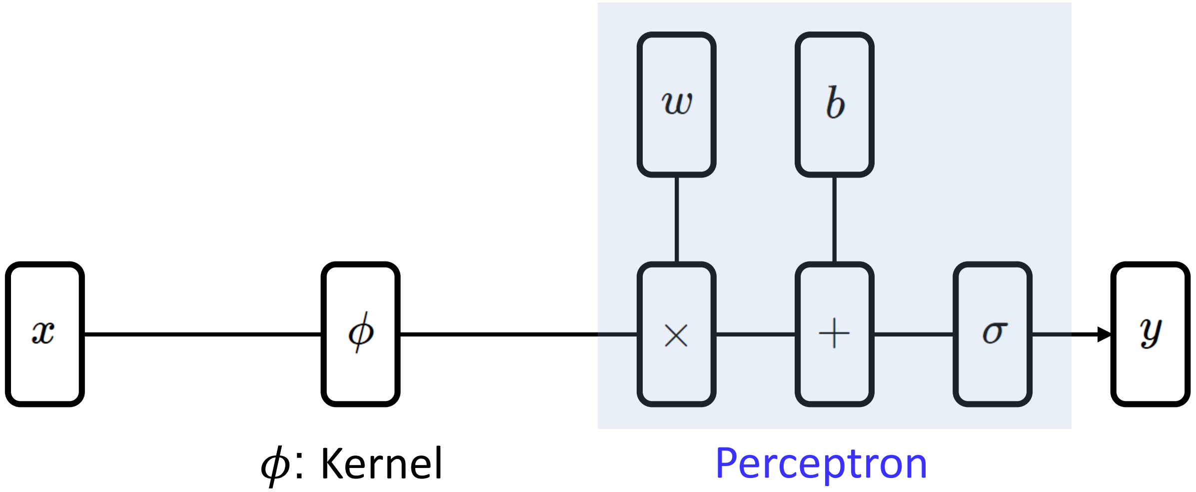
- Linear models (e.g. linear regression, linear SVM) are not just rich enough
  - by mapping data to higher dimensions where it exhibits linear patterns
  - apply the linear model in the new input feature space
  - mapping = changing the feature representation

- Kernels: make linear model work in nonlinear settings

Kernel + Neuron

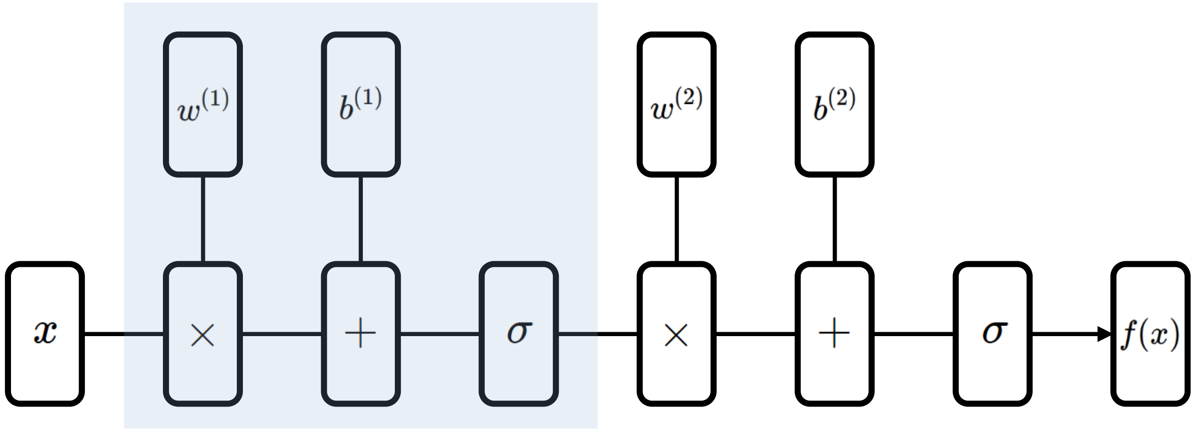
- Nonlinear mapping + neuron

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



Neuron + Neuron

- Nonlinear mapping can be represented by another neurons

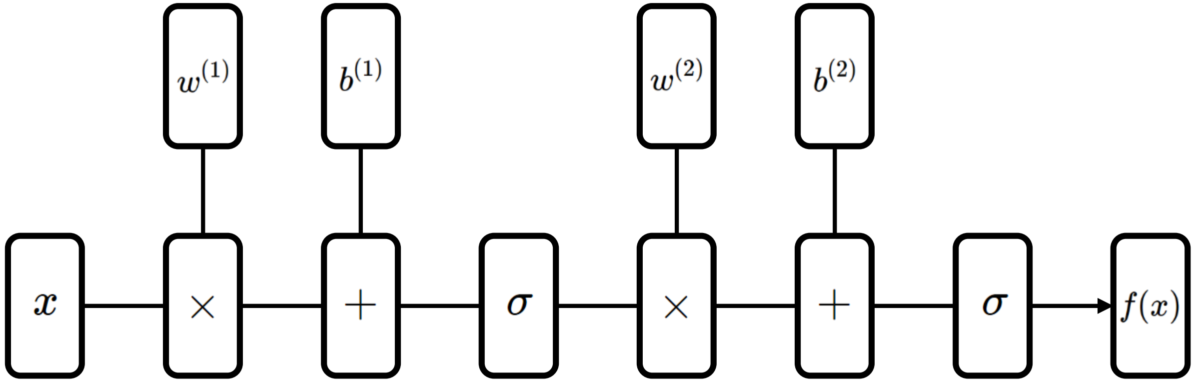


ϕ: Kernel

- Nonlinear Kernel
  - Nonlinear activation functions

Multi Layer Perceptron

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

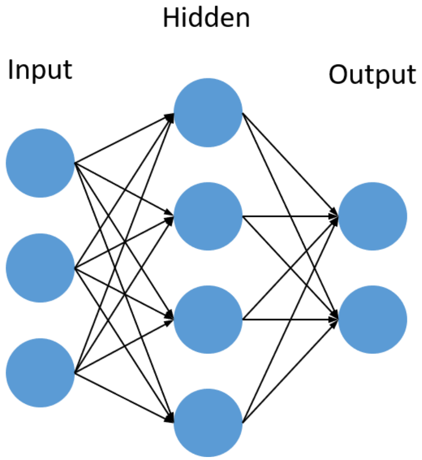


ϕ: Kernel

Summary

- Universal function approximator
- Universal function classifier

- Parameterized

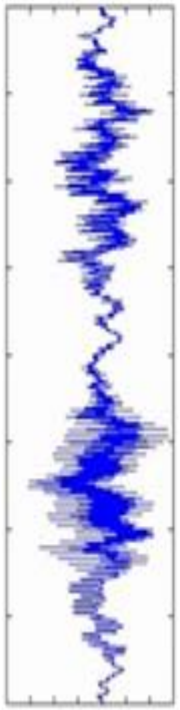


$$x \longrightarrow \boxed{\hat{y} = f_{\omega_1,\cdots,\omega_k}(x)} \longrightarrow y$$

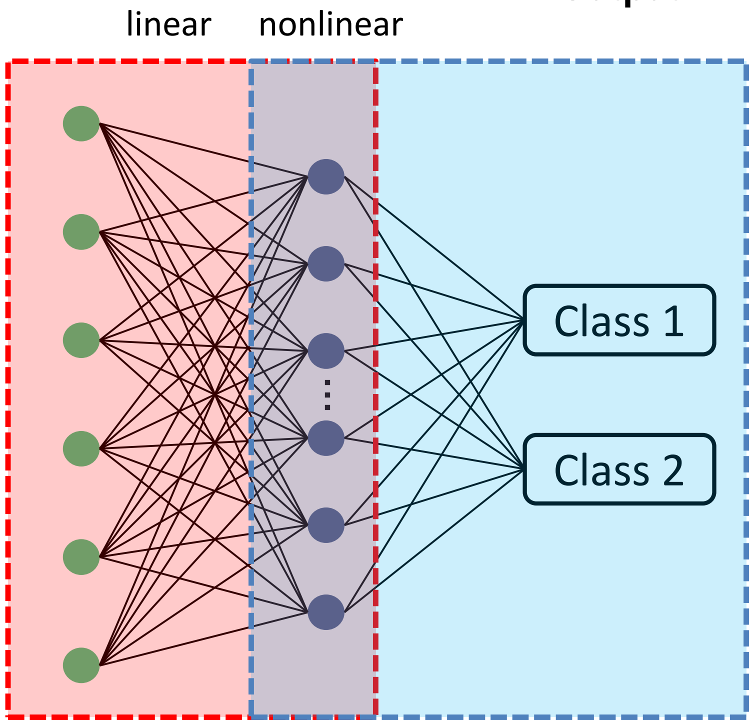
Artificial Neural Networks

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

Input



Output

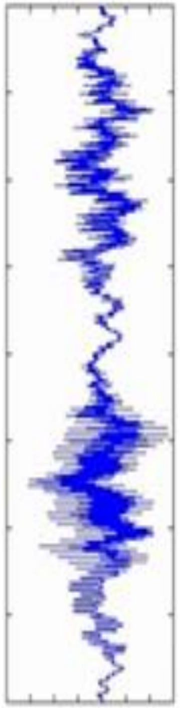


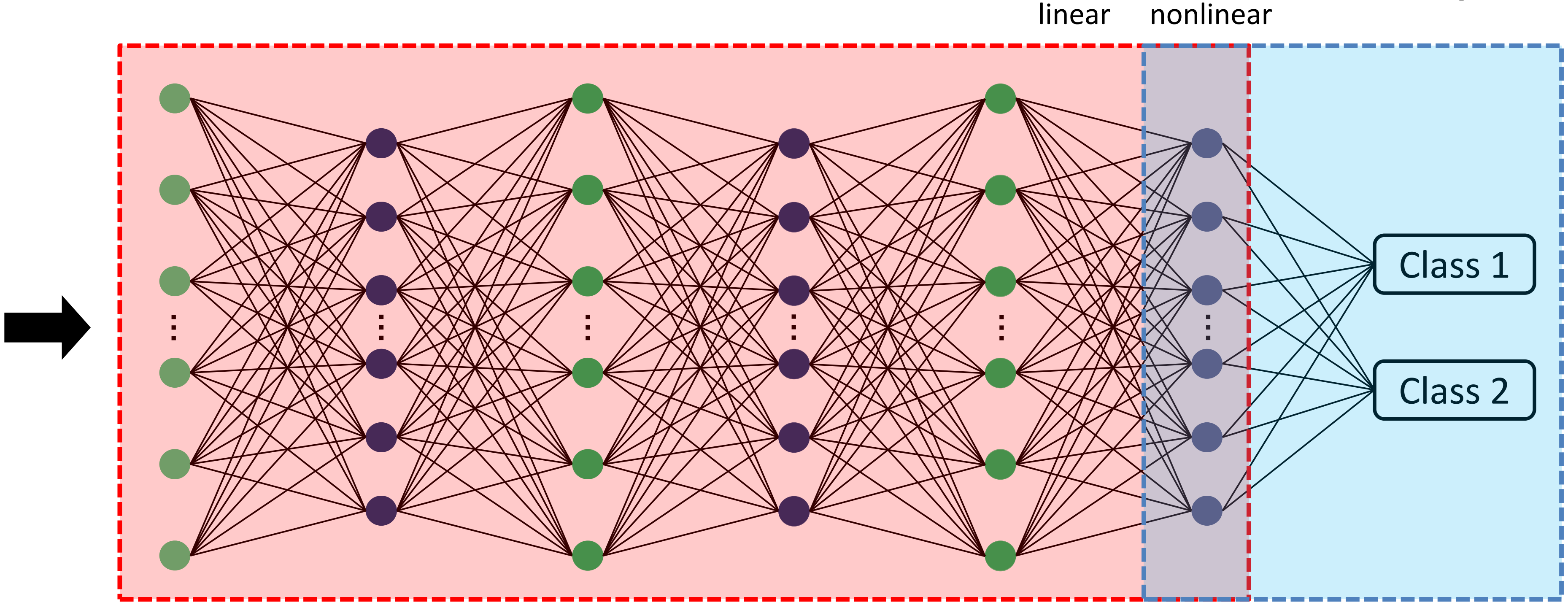
Feature learning **Classification**

Deep Artificial Neural Networks

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

Input

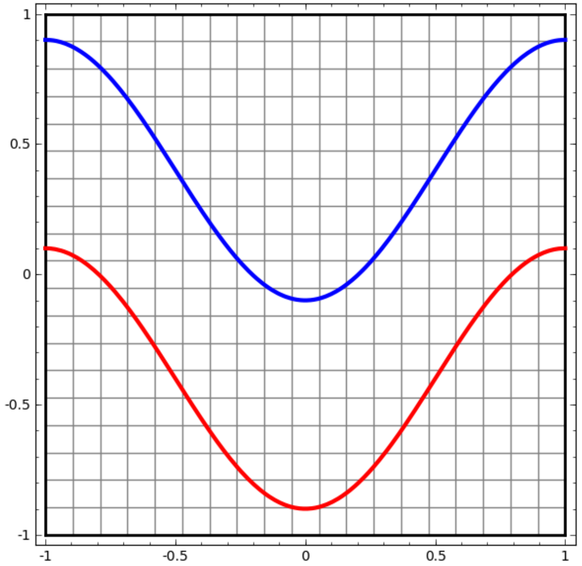


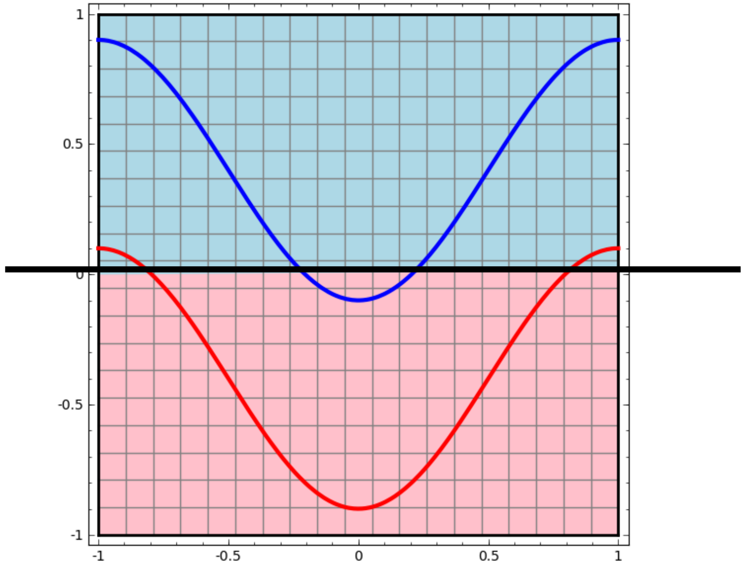


Looking at Parameters

Example: Linear Classifier

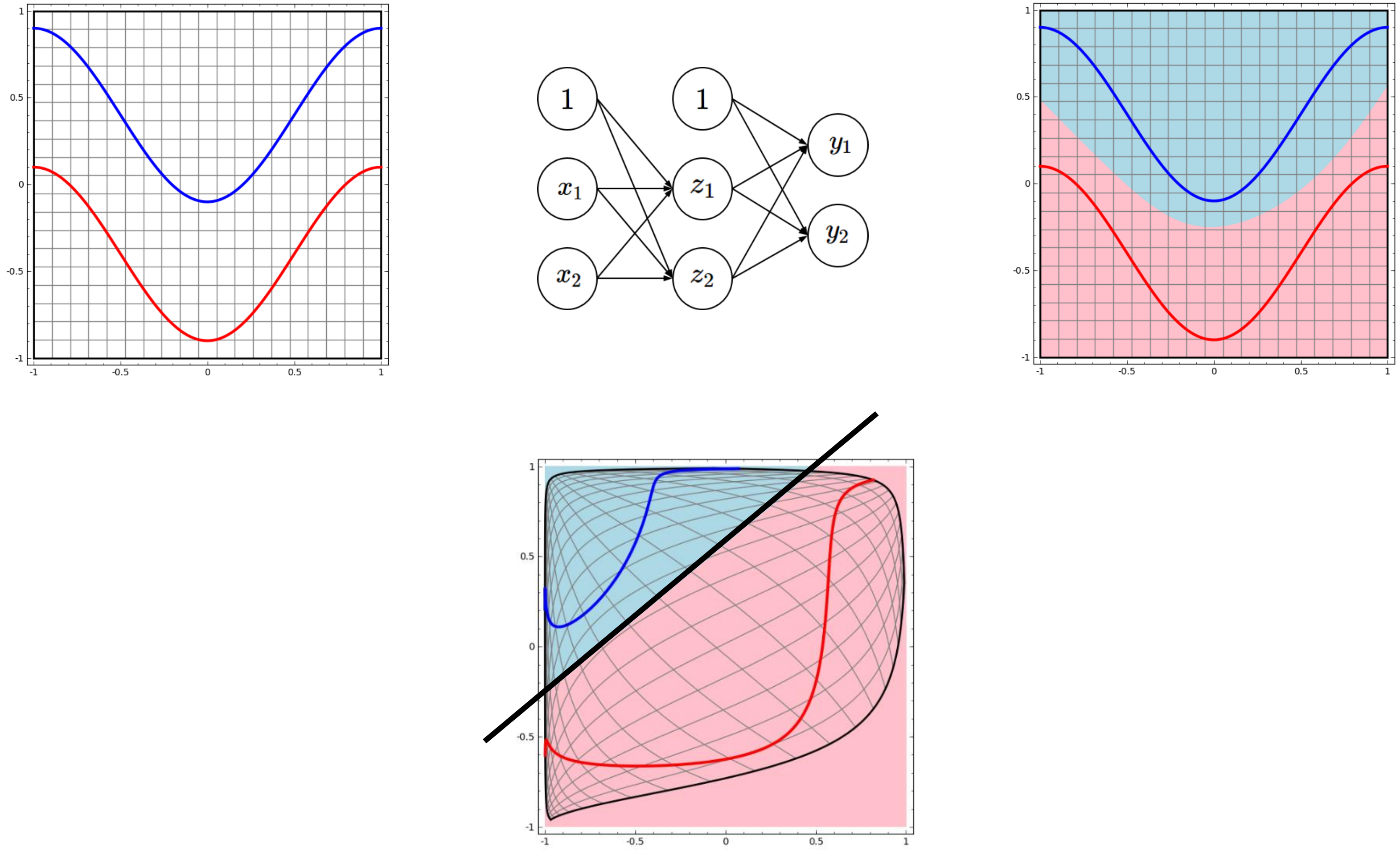
- Perceptron tries to separate the two classes of data by dividing them with a line



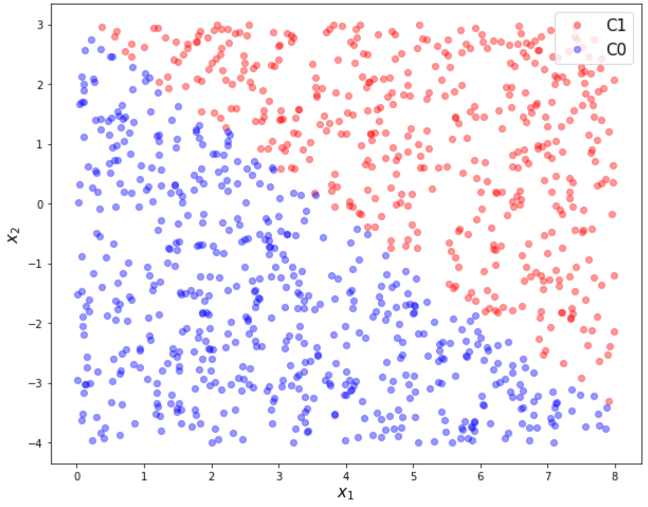


Example: Neural Networks

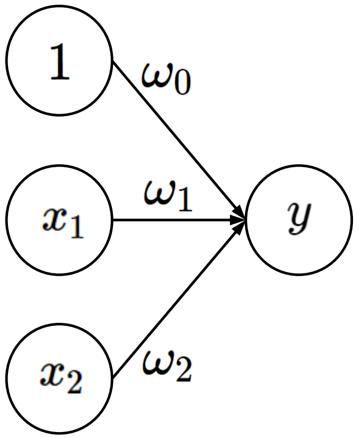
- The hidden layer learns a representation so that the data gets linearly separable

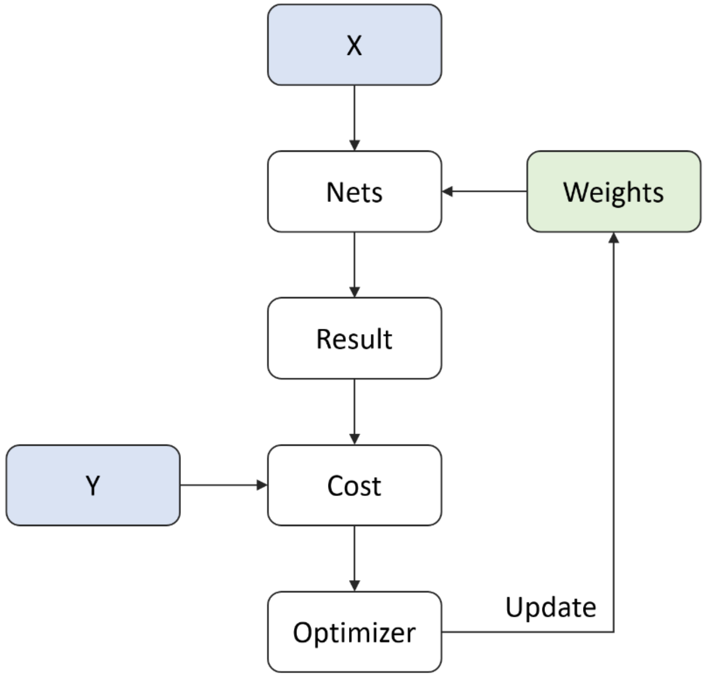


Logistic Regression in a Form of Neural Network



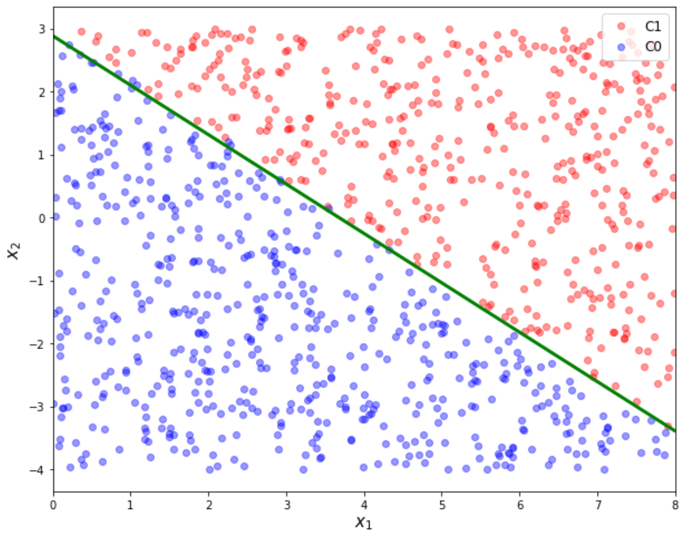
$$y = \sigma\left(\omega_0 + \omega_1 x_1 + \omega_2 x_2\right)$$



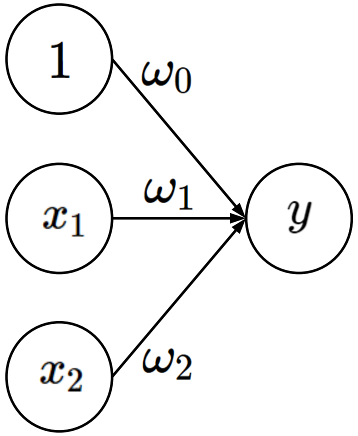


Logistic Regression in a Form of Neural Network

- Neural network convention

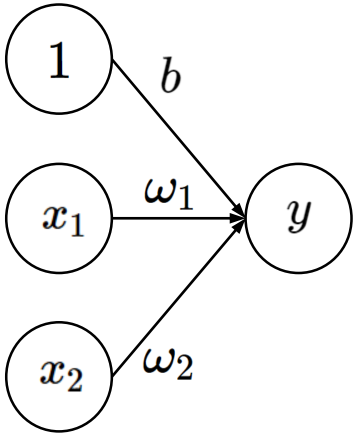


$$y = \sigma\left(\omega_0 + \omega_1 x_1 + \omega_2 x_2\right)$$

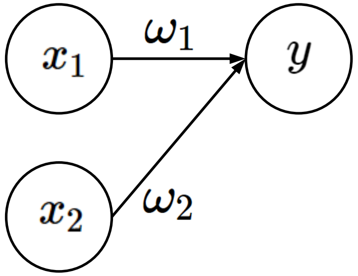


$$\omega_0 \longrightarrow b$$

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



Do not indicate bias units

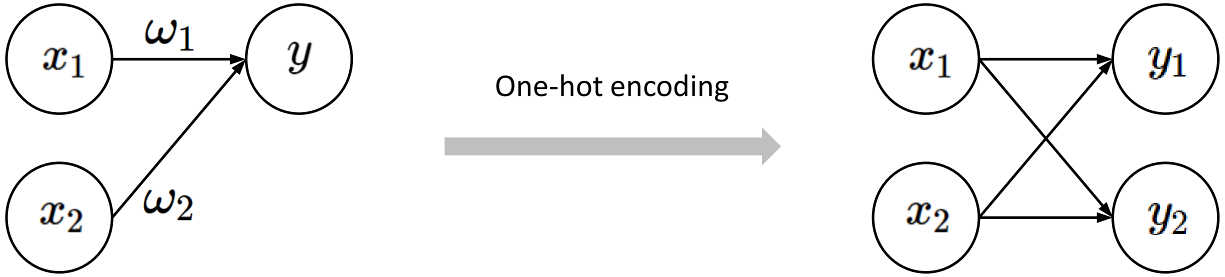



```
n_input = 2
n_output = 1
```

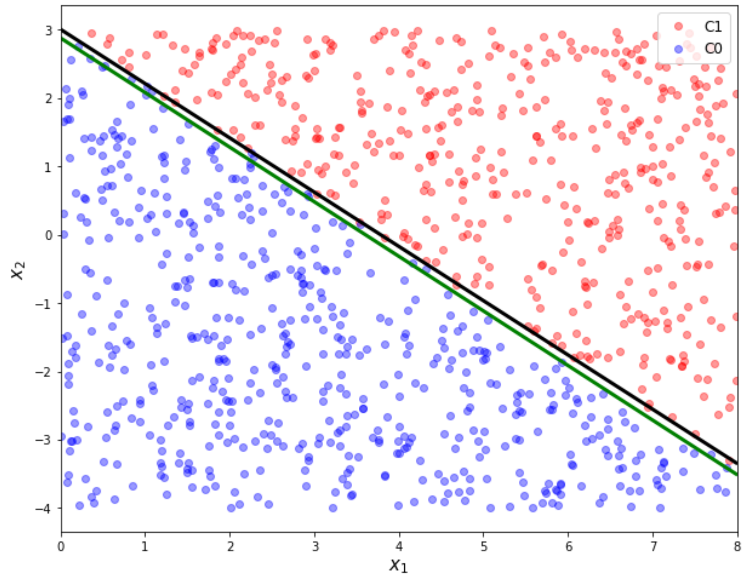
Logistic Regression in a Form of Neural Network

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

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

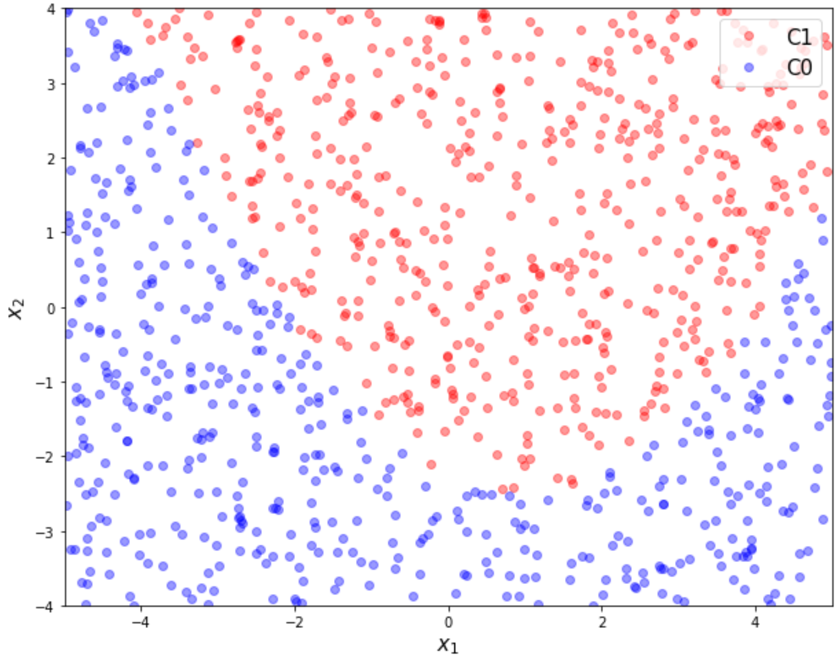


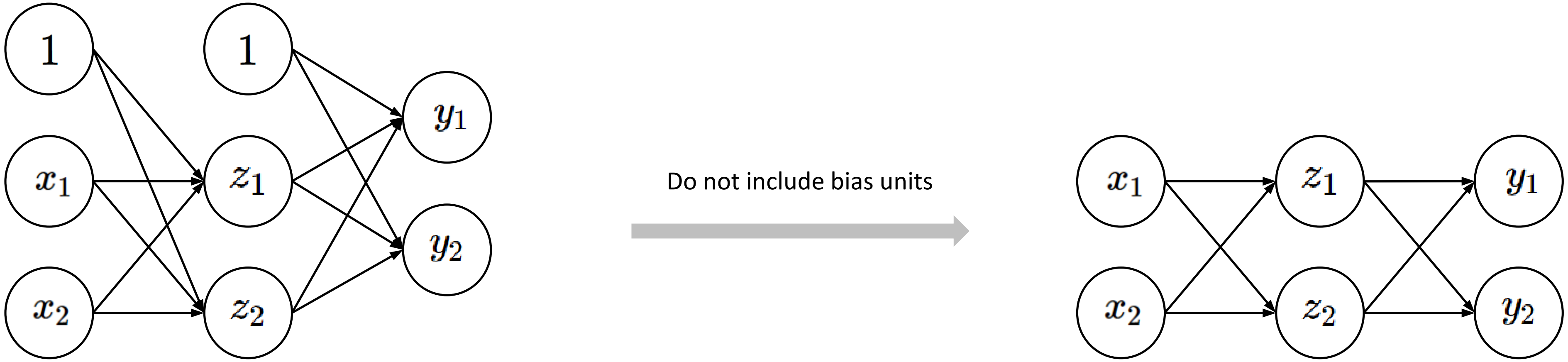

```
n_input = 2
n_output = 2
```



Nonlinearly Distributed Data

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

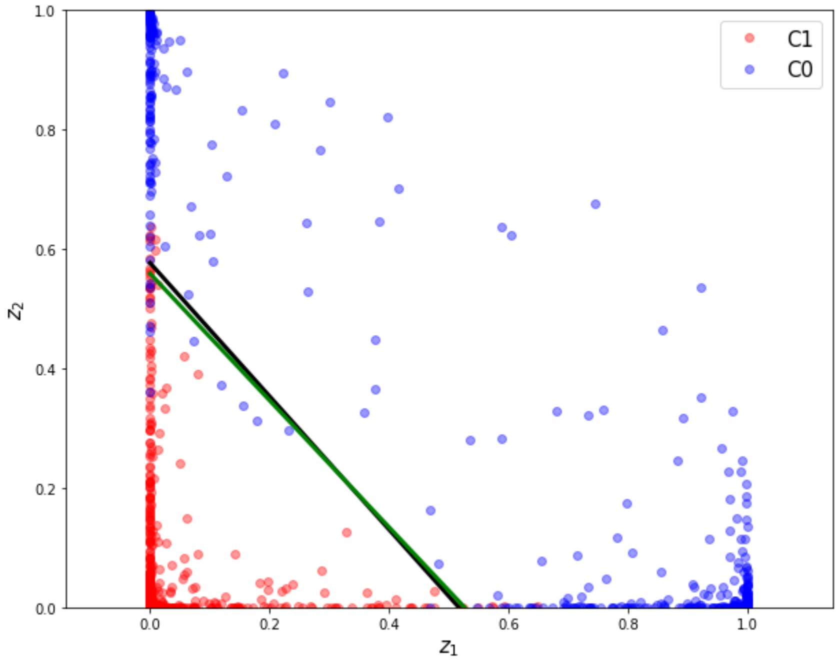


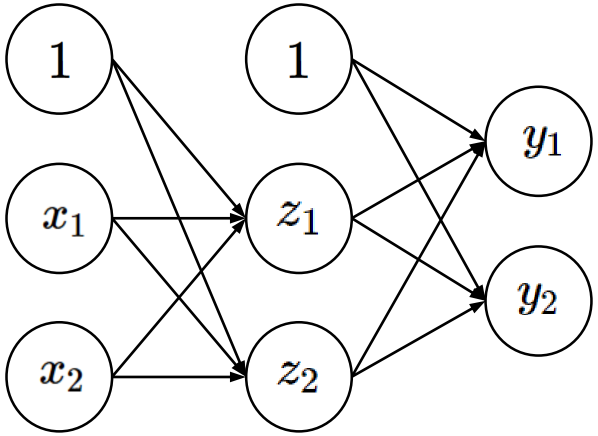



```
n_input  = 2
n_hidden = 2
n_output = 2
```

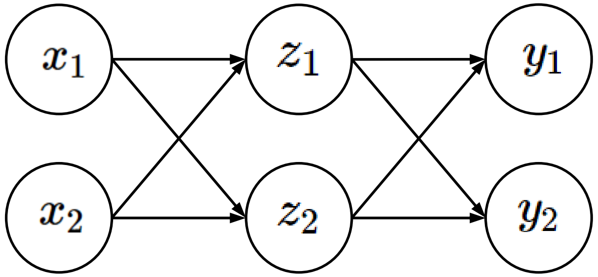
Multi Layers

- z space





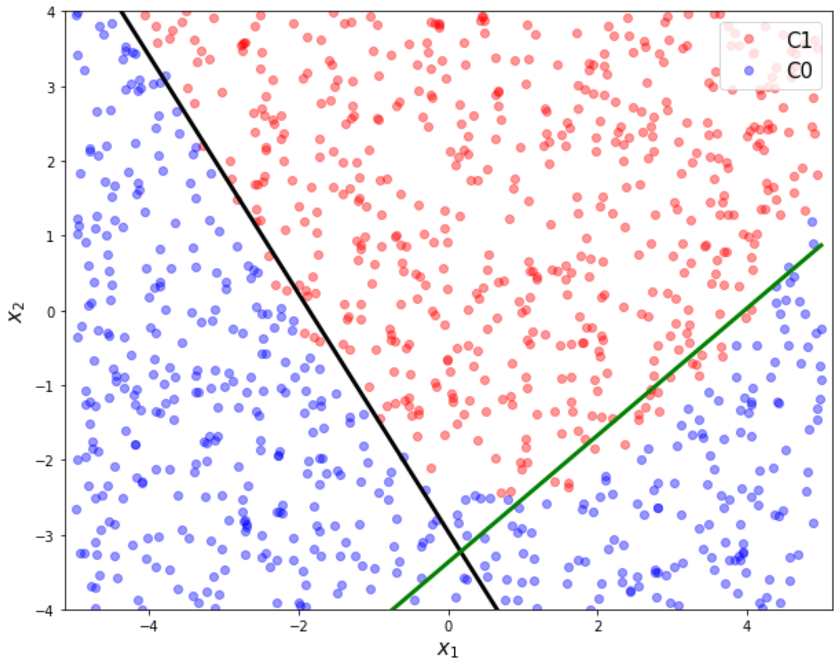
Do not include bias units

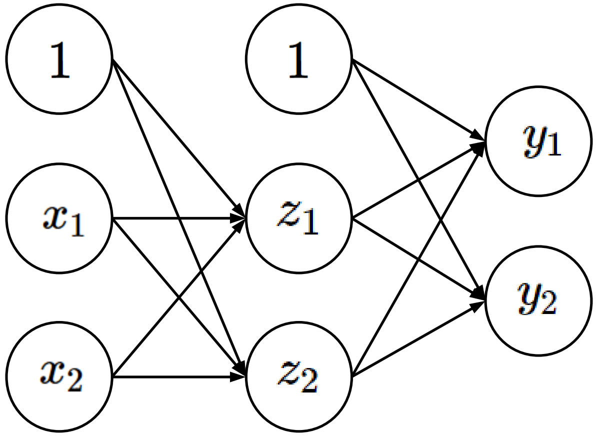



```
n_input  = 2
n_hidden  = 2
n_output  = 2
```

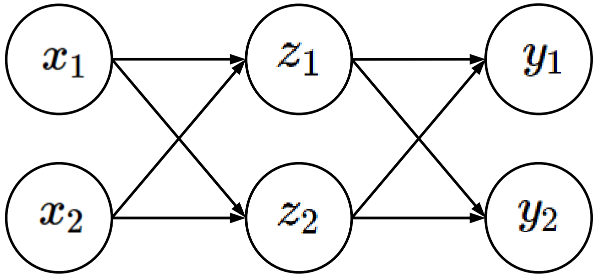
Multi Layers

- x space





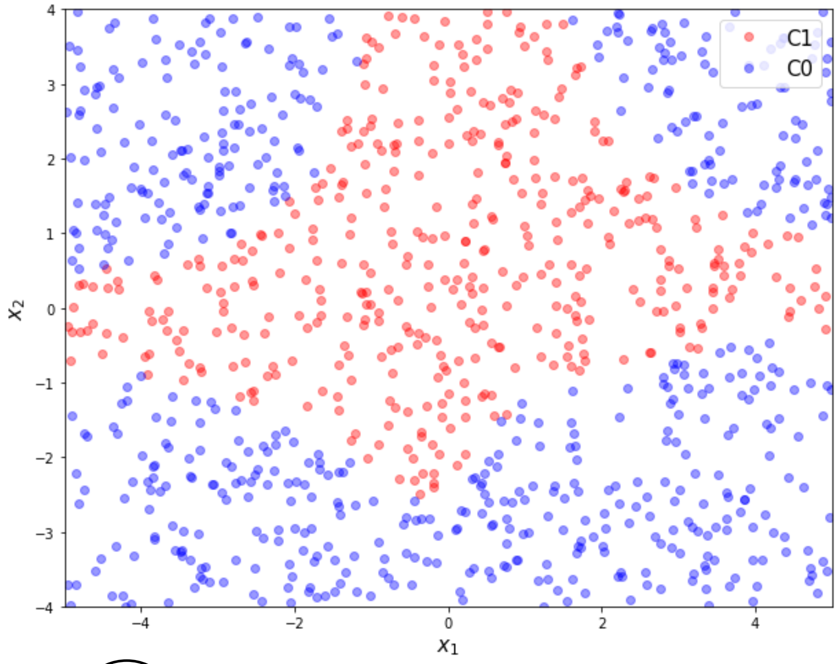
Do not include bias units

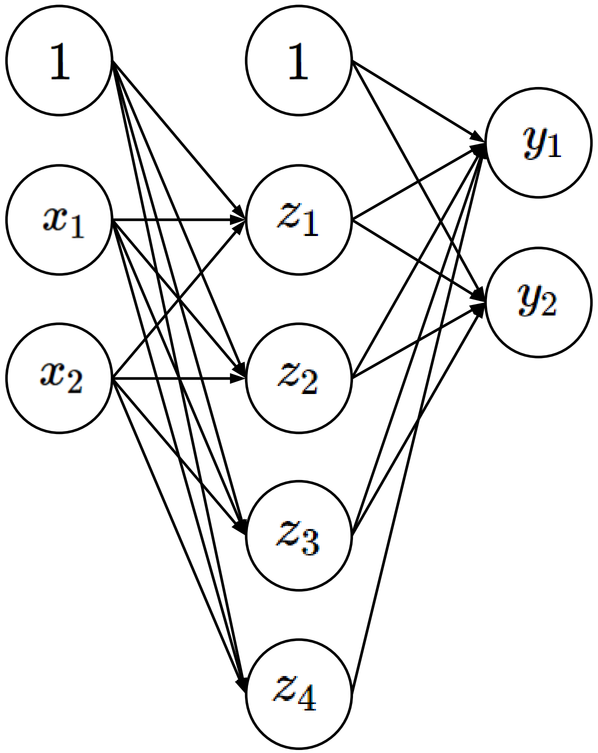


```
n_input = 2
n_hidden = 2
n_output = 2
```

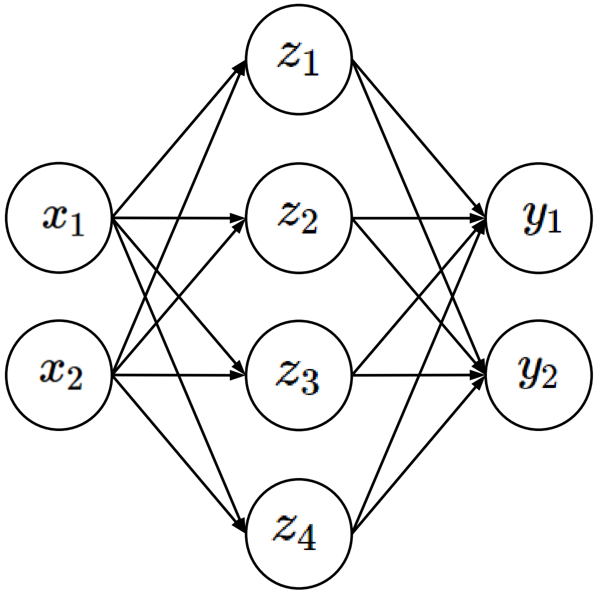
Nonlinearly Distributed Data

- More neurons in hidden layer





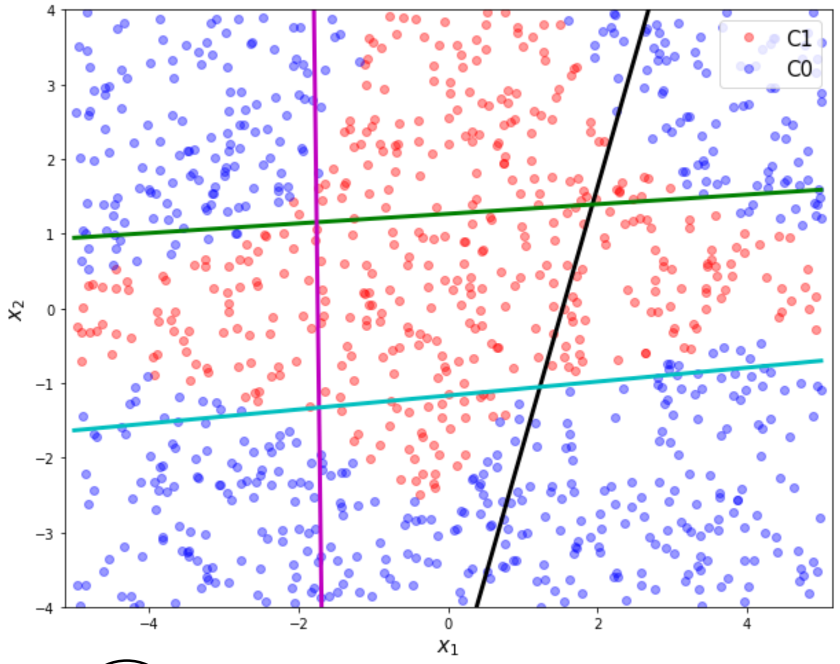
Do not include bias units

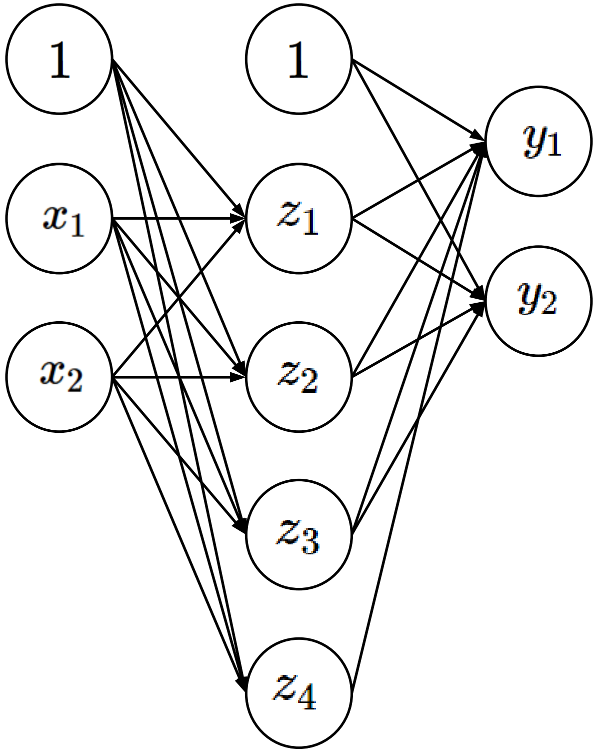



```
n_input = 2
n_hidden = 4
n_output = 2
```

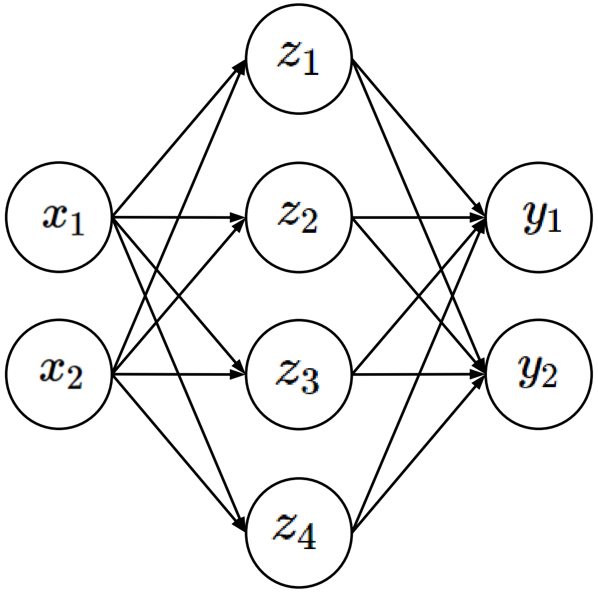
Multi Layers

- Multiple linear classification boundaries





Do not include bias units




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

(Artificial) Neural Networks: Training

Prof. Seungchul Lee

Industrial AI

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 ω
  - constraints $g(\cdot)$

- In mathematical expression

$$\min_{\omega} \quad f(\omega)$$

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):

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

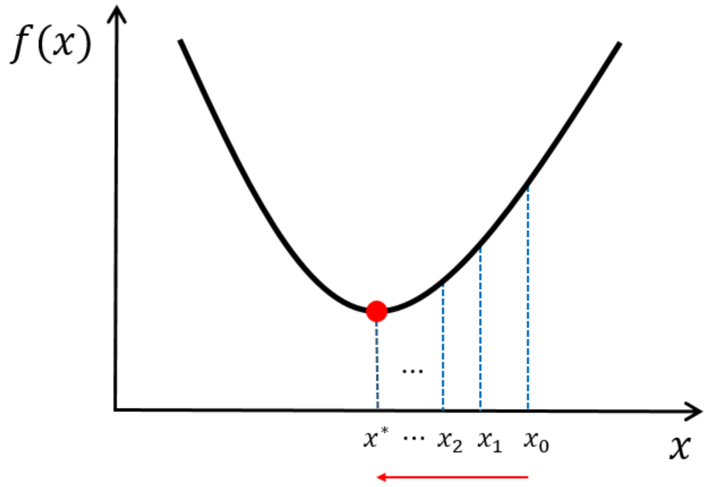
  - Cross entropy (for classification):

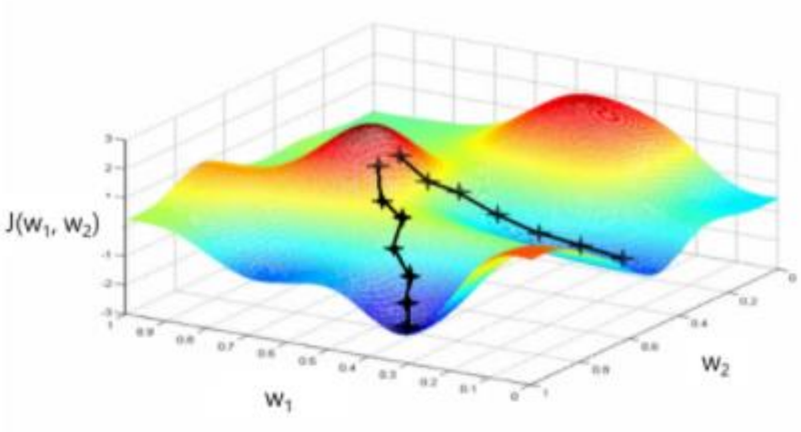
$$-\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log\left(h_{\omega}\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_{\omega}\left(x^{(i)}\right)\right)$$

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)

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

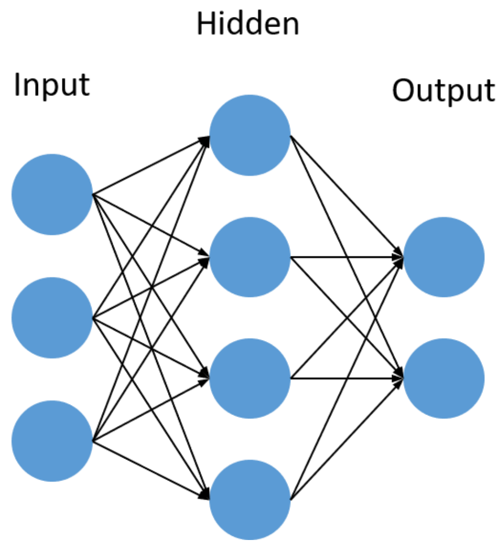




Gradients in ANN

- 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

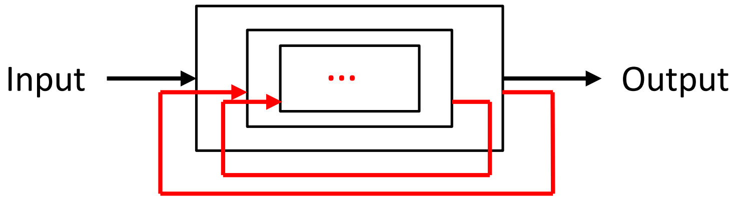


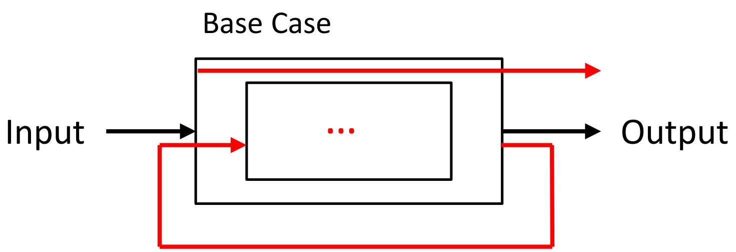
$$x \longrightarrow \boxed{\hat{y} = f_{\omega_1,\cdots,\omega_k}(x)} \longrightarrow y$$

Dynamic Programming

Recursive Algorithm

- One of the central ideas of computer science
- Depends on solutions to smaller instances of the same problem (= sub-problem)
- Function to call itself (it is impossible in the real world)
- Factorial example
  - $n! = n \cdot (n-1) \cdots 2 \cdot 1$





Dynamic Programming

- Dynamic Programming: general, powerful algorithm design technique

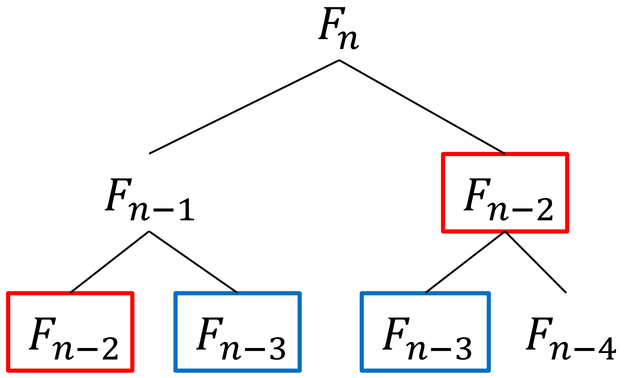
- Fibonacci numbers:

$$F_1 = F_2 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

Naïve Recursive Algorithm

```
fib(n) :
        if n ≤ 2 :  f = 1
        else :  f = fib(n − 1) + fib(n − 2)
        return f
```

- It works. Is it good?



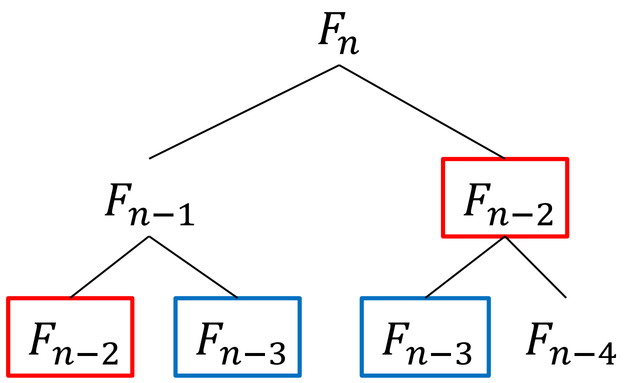
Memorized Recursive Algorithm

```
memo = [ ]
    fib(n) :
        if n in memo : return memo[n]

        if n ≤ 2 :  f = 1
        else :  f = fib(n − 1) + fib(n − 2)

        memo[n] = f
        return f
```

- Benefit?
  - fib(n) only recurses the first time it's called



Dynamic Programming Algorithm

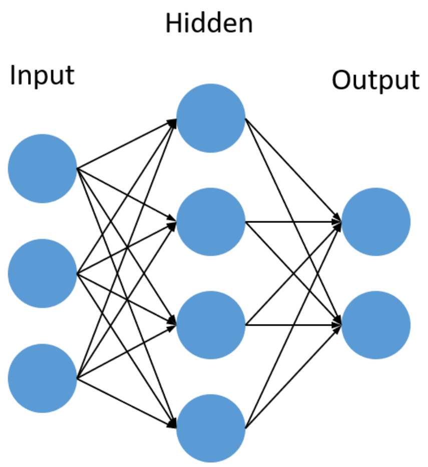
- Memorize (remember) & re-use solutions to subproblems that helps solve the problem

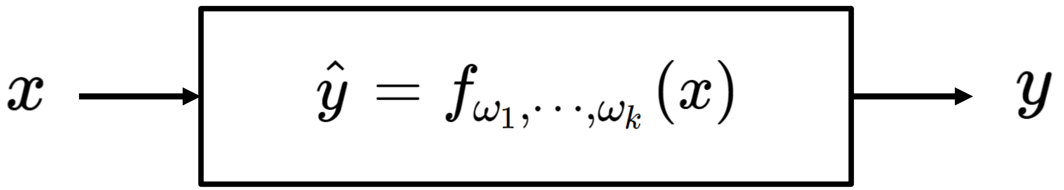
- DP $\approx$ recursion + memorization

Backpropagation

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



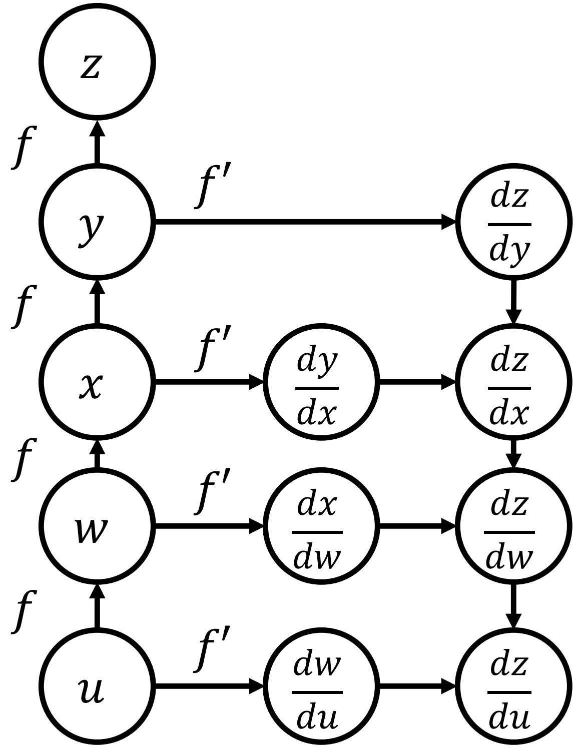


Training Neural Networks: Backpropagation Learning

- 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

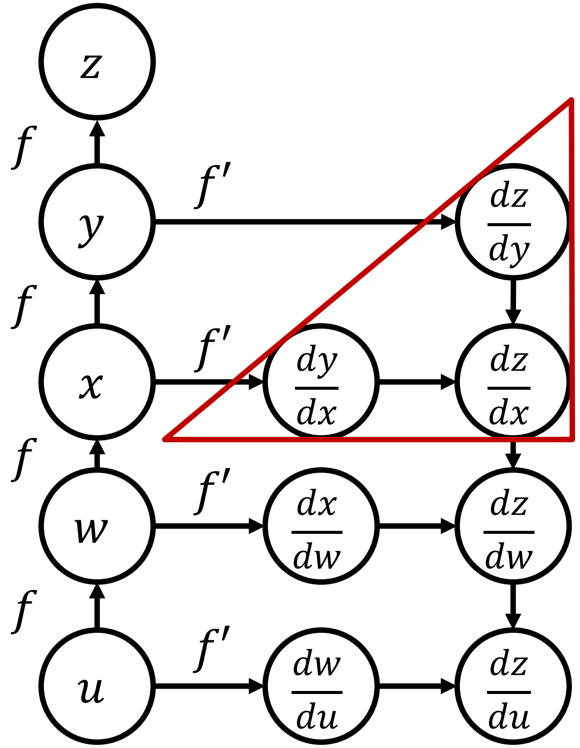
- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\left(g(x)\right)' = f'\left(g(x)\right)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$



- Backpropagation
  - Update weights recursively

Backpropagation

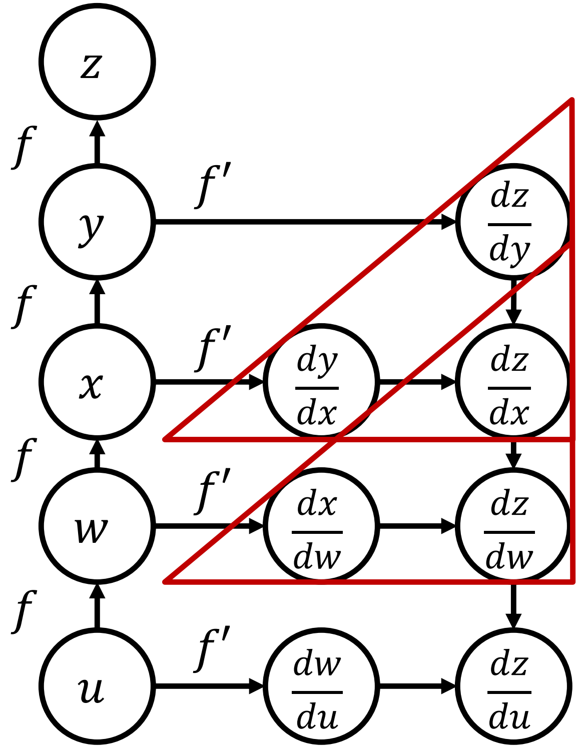
- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\left(g(x)\right)' = f'\left(g(x)\right)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$



- Backpropagation
  - Update weights recursively

Backpropagation

- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$

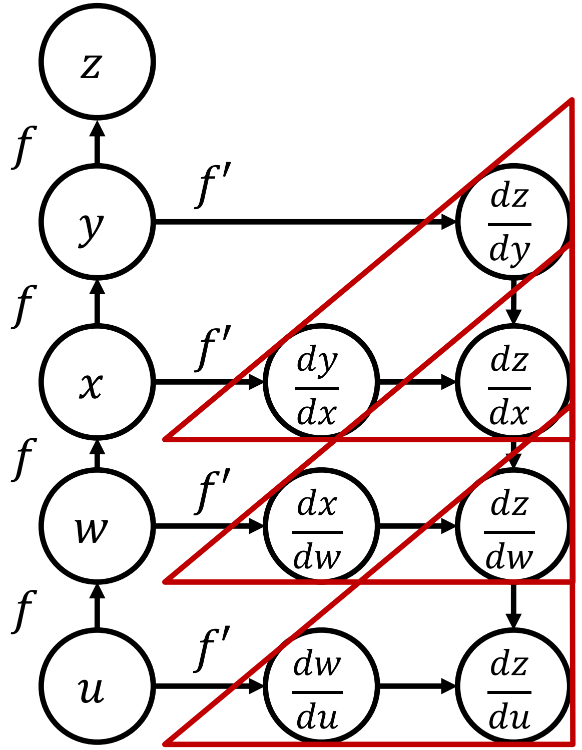


- Backpropagation
  - Update weights recursively

Backpropagation

- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$

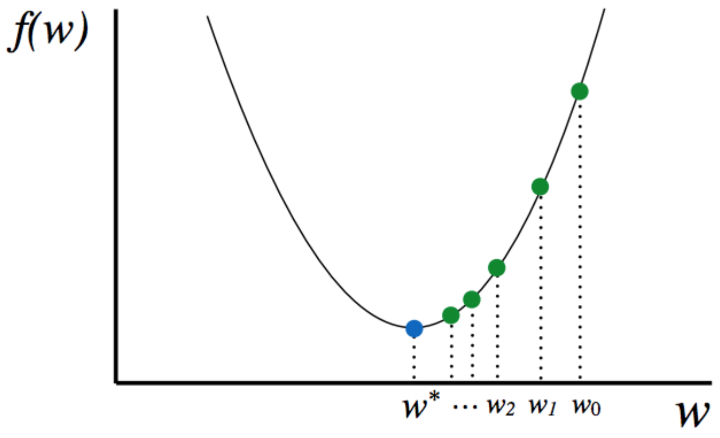
- Backpropagation
  - Update weights recursively with memory



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

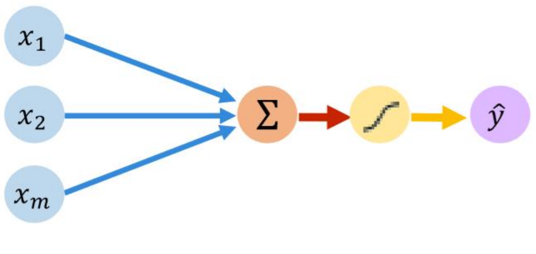


- It is not easy to numerically compute gradients in network in general.
  - The good news: people have already done all the "hard work" of developing numerical solvers (or libraries)
  - There are a wide range of tools → We will use the TensorFlow

Core Foundation Review

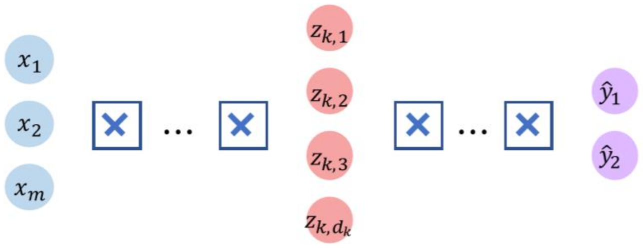
The Perceptron

- Structural building blocks
- Nonlinear activation functions



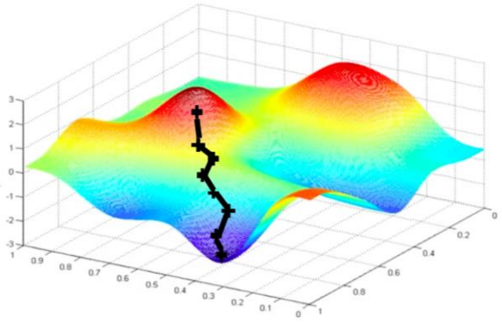
Neural Networks

- Stacking Perceptrons to form neural networks
- Optimization through backpropagation



Training in Practice

- Adaptive learning
- Batching
- Regularization



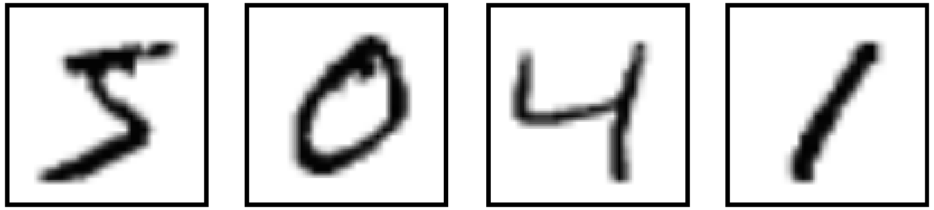
(Artificial) Neural Networks with Scikit-learn

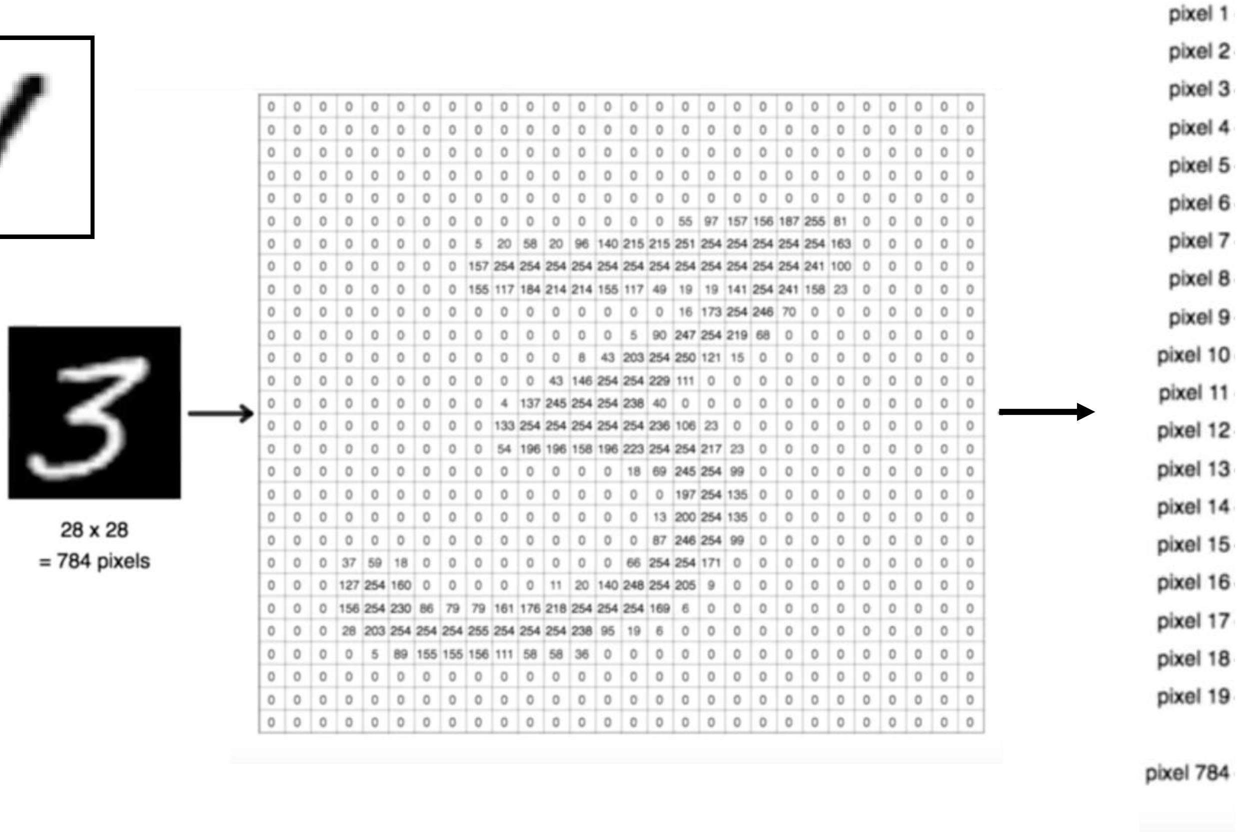
Industrial AI Lab.

Prof. Seungchul Lee

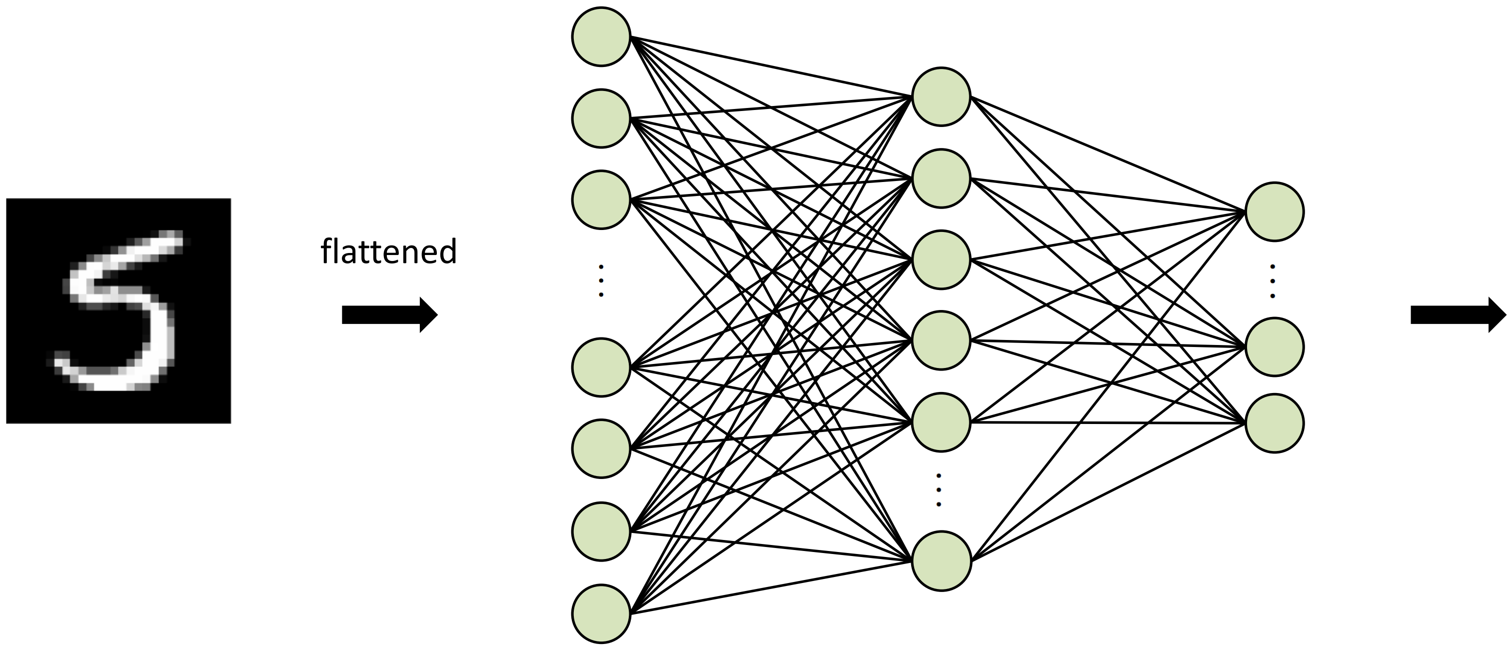
MNIST database

- 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



digit prediction
in one-hot-encoding

Input image
(28 X 28)

Input layer
(784)

hidden layer
(100)

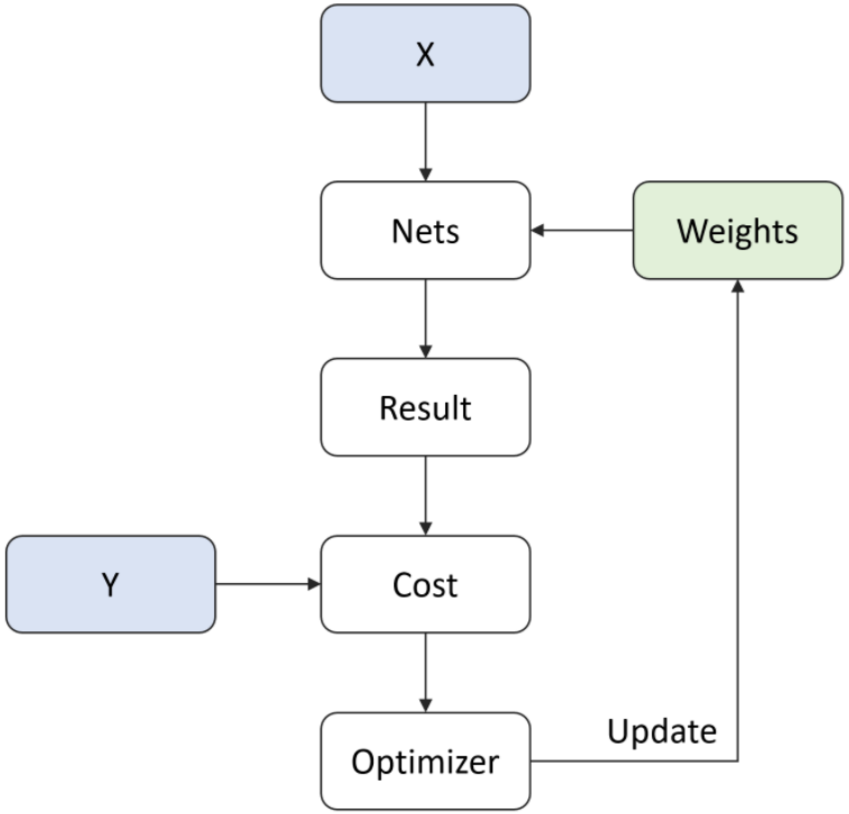
output layer
(10)

Iterative Optimization

- We will use
  - Mini-batch gradient descent
  - Adam optimizer

$$\begin{aligned} \min_{\theta} \quad & f(\theta) \\ \text{subject to} \quad & g_i(\theta) \le 0 \end{aligned}$$

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



ANN with Scikit-learn

- Import Library

```python
# Import Library
import numpy as np
import matplotlib.pyplot as plt

from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
```

- Load MNIST Data
  - Download MNIST data

```python
train_x = np.load('./data_files/mnist_train_images.npy')
train_y = np.load('./data_files/mnist_train_labels.npy')
test_x = np.load('./data_files/mnist_test_images.npy')
test_y = np.load('./data_files/mnist_test_labels.npy')
```

One Hot Encoding

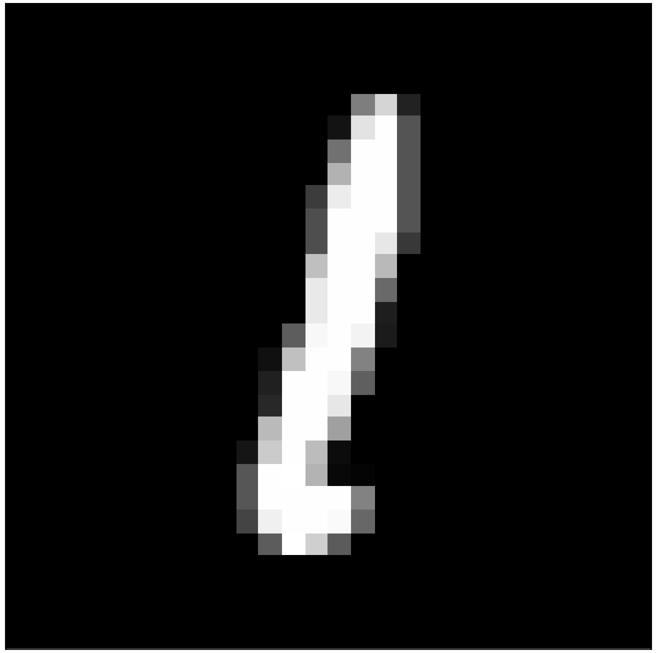
- One hot encoding

```
mnist_train_labels[7]
```

```
array([ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

```
np.argmax(mnist_train_labels[7])
```

```
0
```

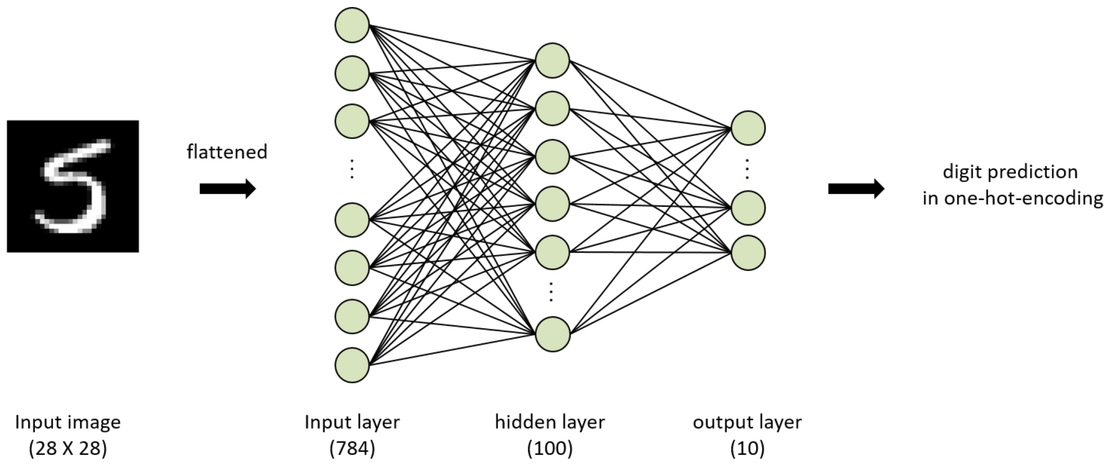


Training

```
clf = MLPClassifier(hidden_layer_sizes = (100,),
                    activation = 'logistic',
                    solver = 'sgd',
                    learning_rate_init = 0.0001,
                    batch_size = 50,
                    max_iter = 100,
                    verbose = True)
```

```
clf.fit(train_x, train_y)
```

```
Iteration 1, loss = 3.57824966
Iteration 2, loss = 3.18791200
Iteration 3, loss = 3.13809828
Iteration 4, loss = 3.08519212
Iteration 5, loss = 3.02767935
Iteration 6, loss = 2.96478203
Iteration 7, loss = 2.89614439
Iteration 8, loss = 2.82202289
Iteration 9, loss = 2.74309025
Iteration 10, loss = 2.66058488
```



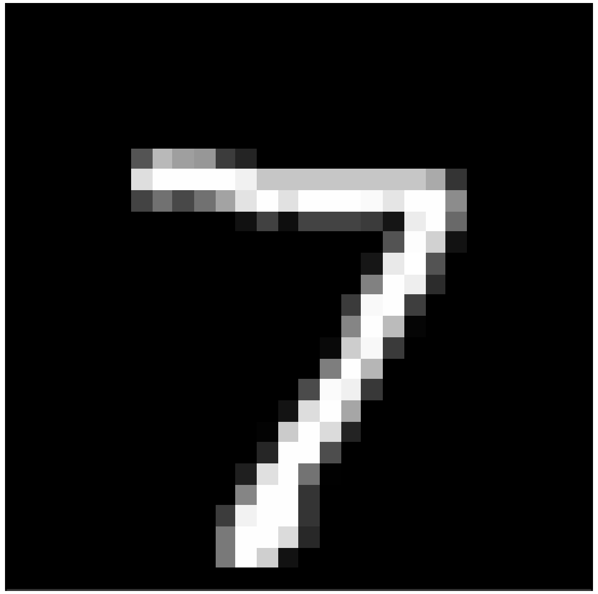
Test or Evaluation

```
pred = clf.predict(test_x)
print("Accuracy : {}%".format(accuracy_score(test_y, pred)*100))
```

```
Accuracy : 96.0%
```

```
logits = clf.predict_proba(test_x[:1])
predict = clf.predict(test_x[:1])

plt.figure(figsize = (6,6))
plt.imshow(test_x[:1].reshape(28,28), 'gray')
plt.xticks([])
plt.yticks([])
plt.show()
```



```
Prediction : 7
Probability : [ 0.02  0.    0.01  0.03  0.01  0.02  0.    0.93  0.01  0.12]
```