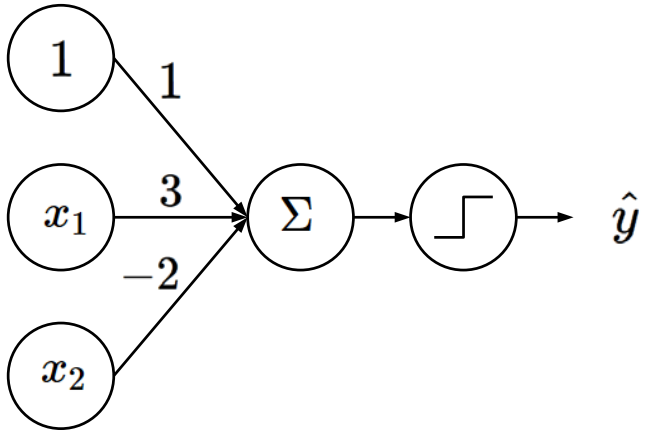
(Artificial) Neural Networks: From Perceptron to MLP

Prof. Seungchul Lee

Industrial AI Lab.

Binary Linear Classifier

- Given weights

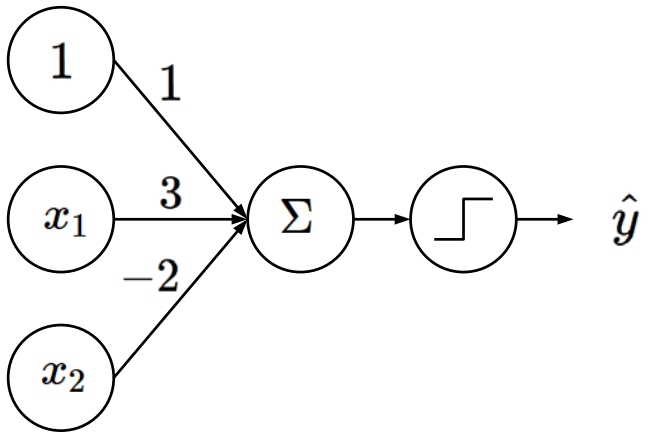


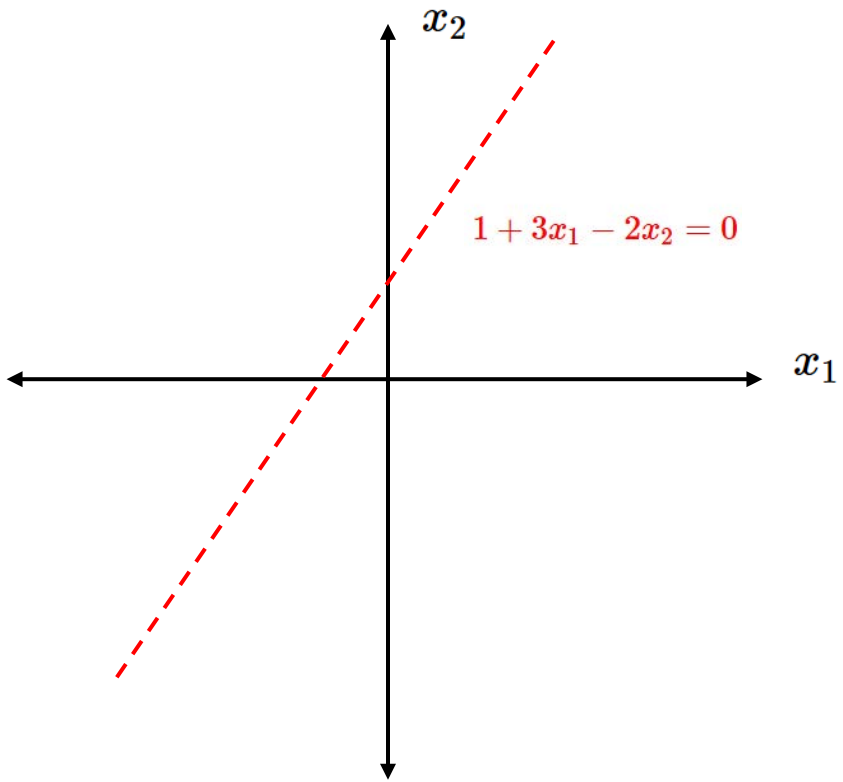
$$\begin{aligned}\hat{y} &= g\left(\omega_0 + X^T\omega\right) \\ &= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \\ &= g\left(1 + 3x_1 - 2x_2\right)\end{aligned}$$

Binary Linear Classifier

- Given weights

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

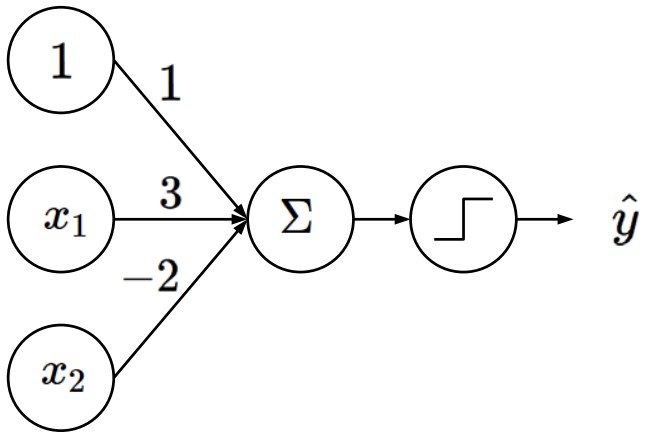


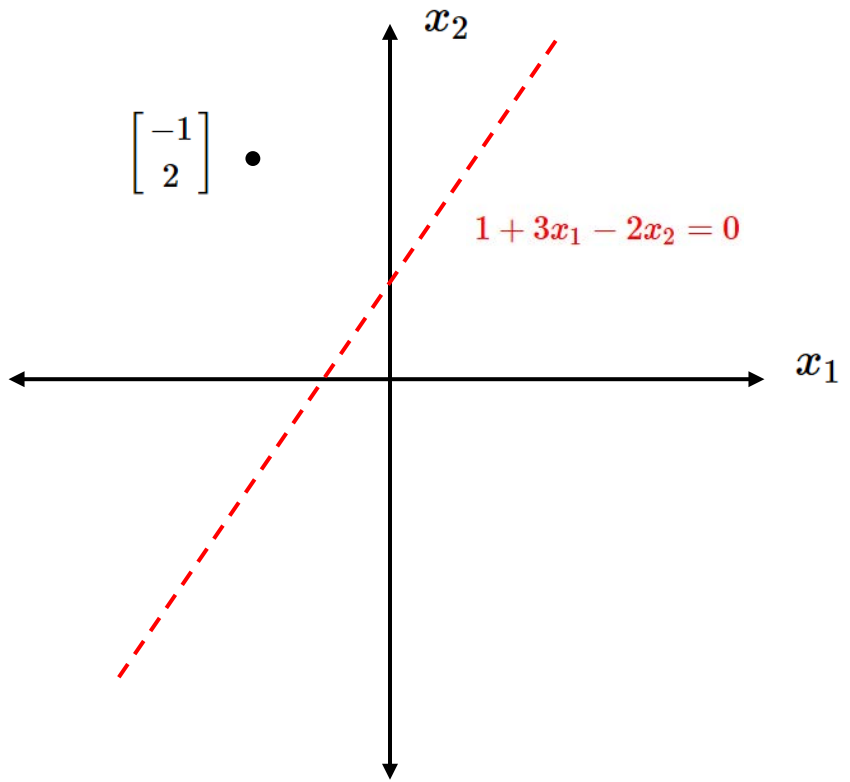


Binary Linear Classifier with New Data

- Forward propagation with new data

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

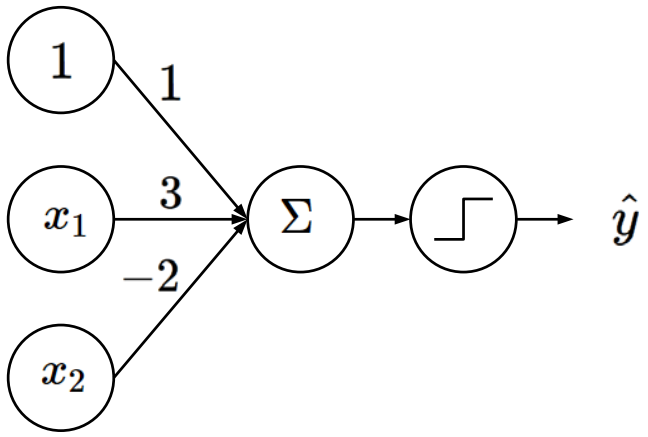


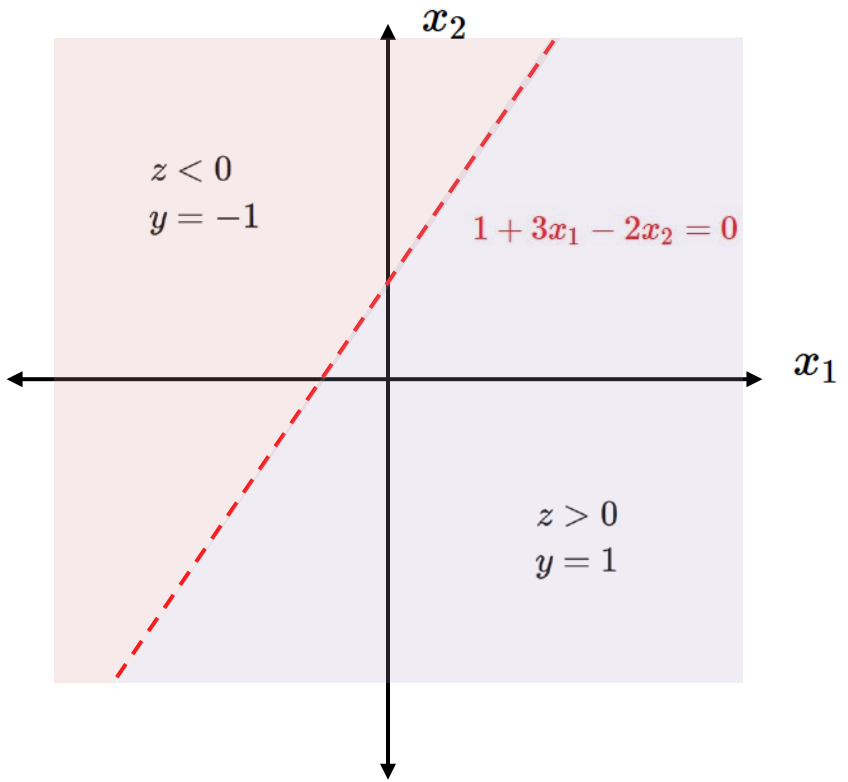


$$\hat{y} = g(1 + 3 \times (-1) - 2 \times 2) = g(-6) = -1$$

Binary Linear Classifier

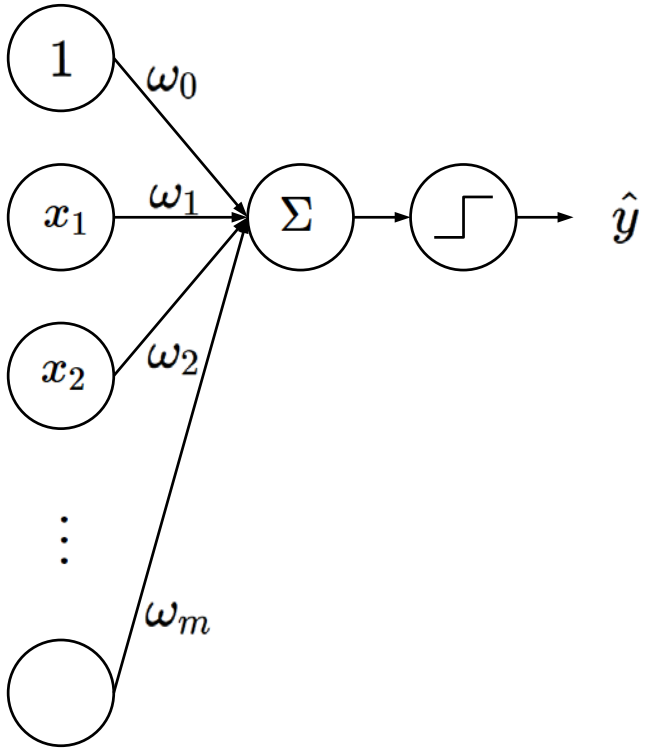
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$





Binary Linear Classifier in High Dimension

- More neurons means
  - hyper-plane in a higher dimension



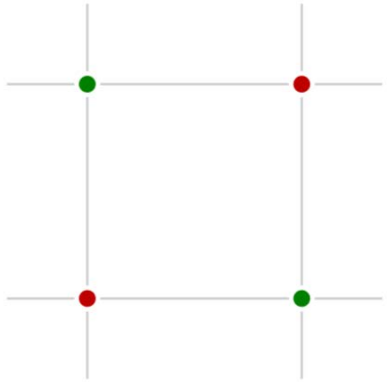
$$\hat{y} = g\left(\omega_0 + X^T \omega\right)$$

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

From Perceptron to MLP

XOR Problem

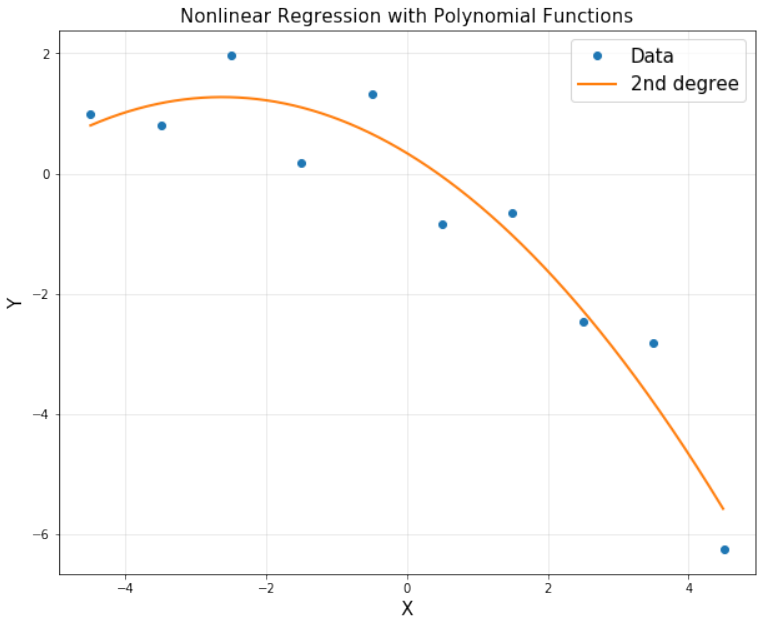
- Not linearly separable
- Limitation of linear classifier



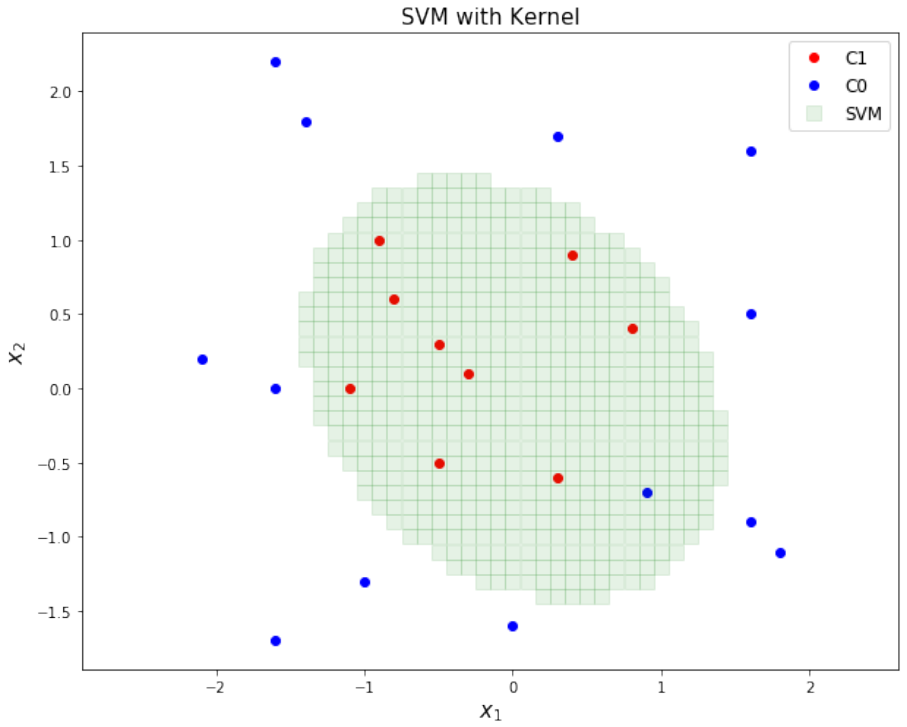
- Single neuron = one linear classification boundary

Nonlinear Curve Approximated by Multiple Lines

- Nonlinear regression

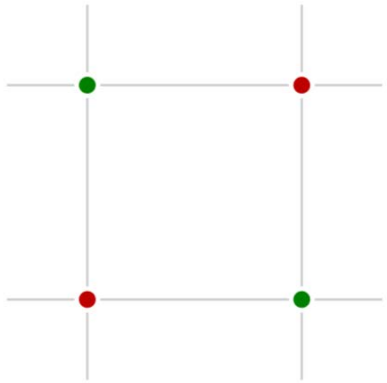


- Nonlinear classification



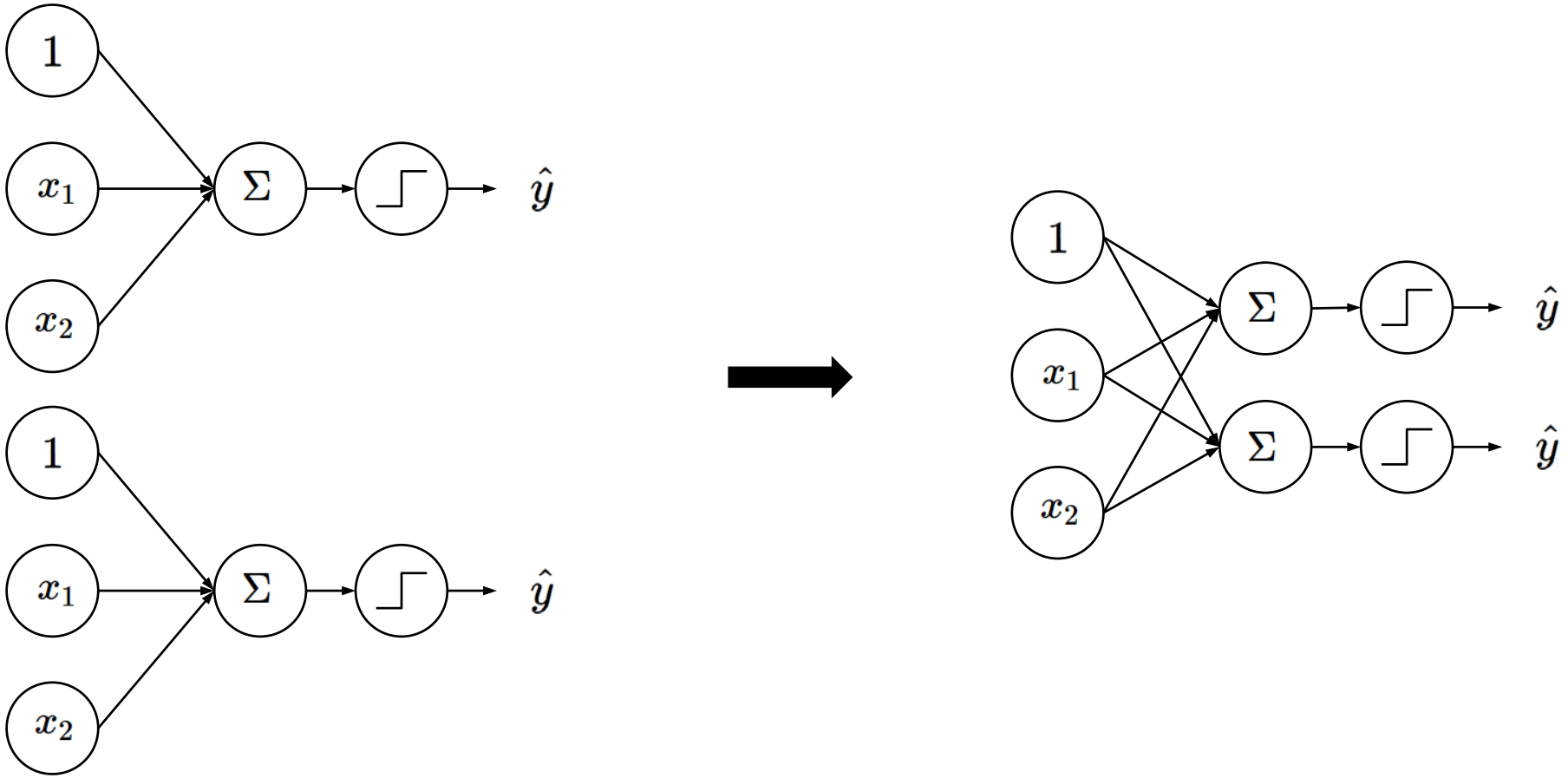
XOR Problem

- At least two lines are required



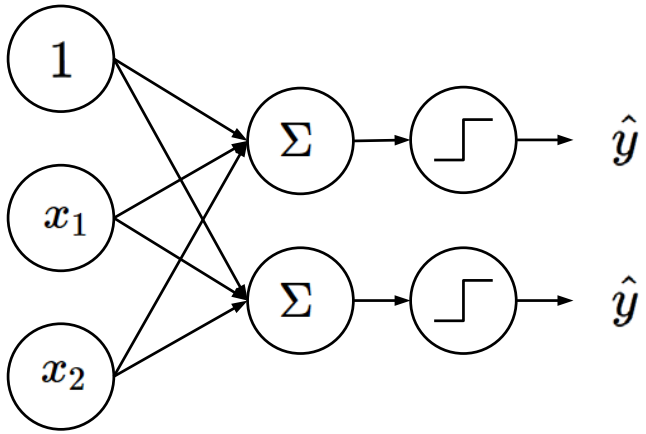
Artificial Neural Networks: MLP

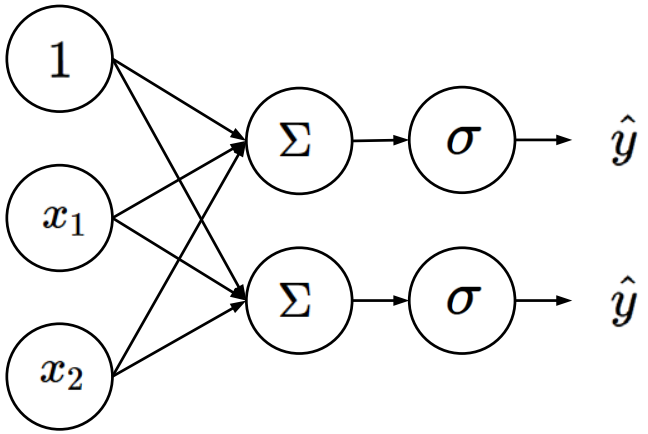
- Multi-layer Perceptron (MLP) = Artificial Neural Networks (ANN)
  - Multi neurons = multiple linear classification boundaries

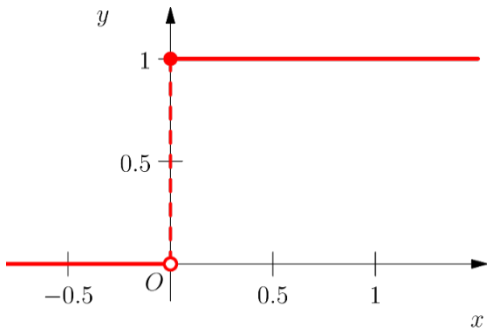


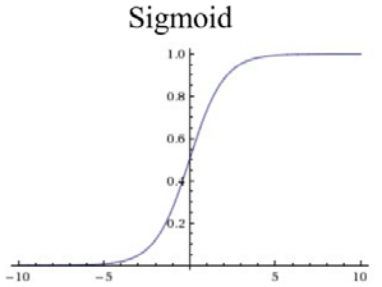
Artificial Neural Networks: Activation Function

- Differentiable nonlinear activation function





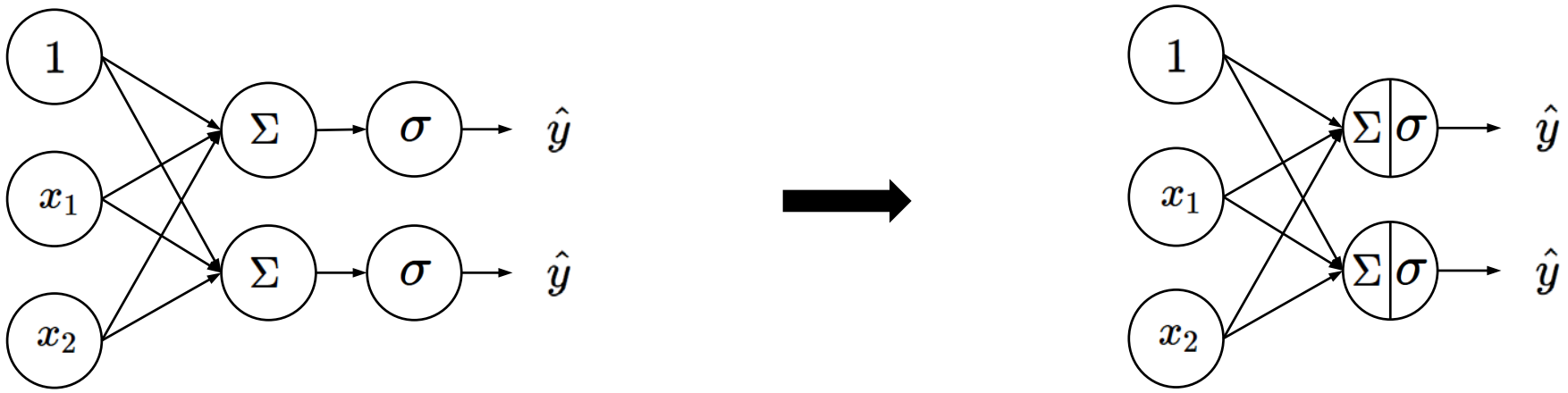




$$g(x) = \frac{1}{1 + e^{-x}}$$

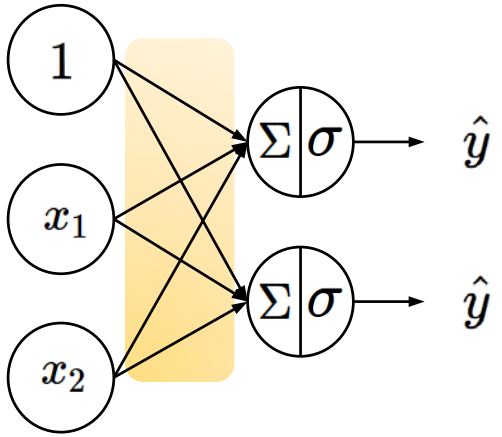
Artificial Neural Networks

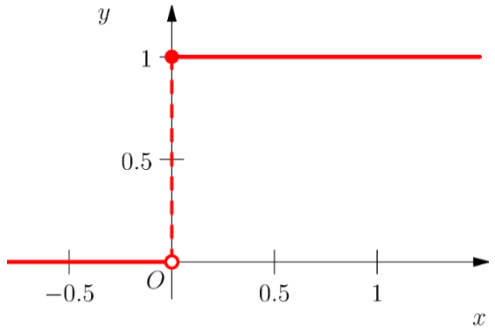
- In a compact representation

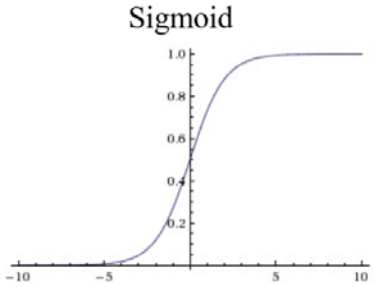


Two Ways of Looking at Artificial Neural Networks

- Still represent lines

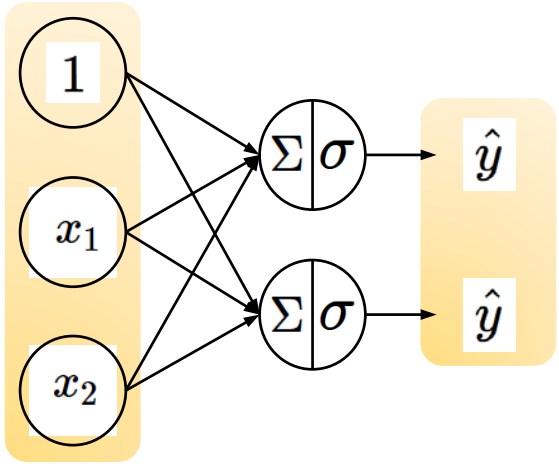


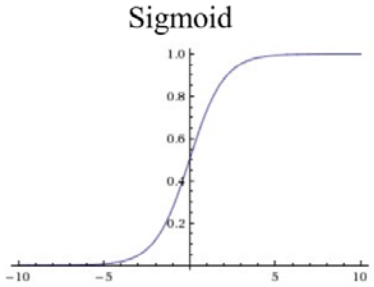




$$g(x) = \frac{1}{1+e^{-x}}$$

- Can represent nonlinear relationship between input and outputs due to nonlinear activation function



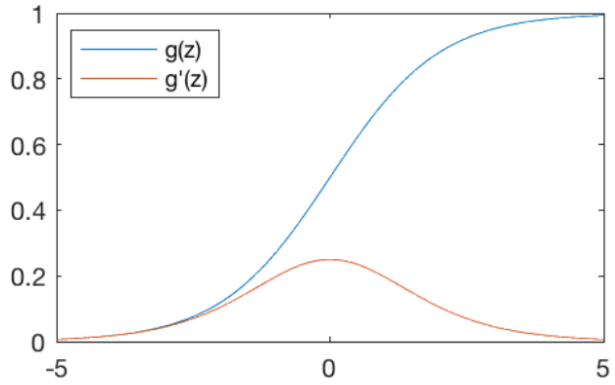


$$g(x) = \frac{1}{1+e^{-x}}$$

Common Activation Functions

Discuss later

Sigmoid Function

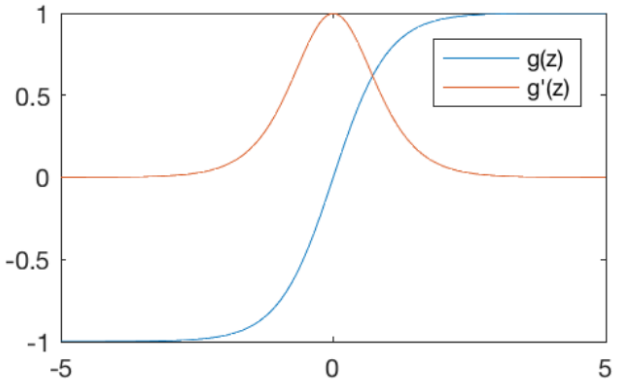


$$g(z) = \frac{1}{1 + e^{-z}}$$

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

`tf.nn.sigmoid(z)`

Hyperbolic Tangent

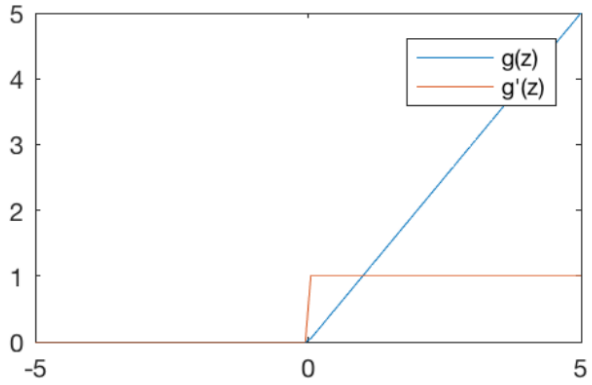


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

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

`tf.nn.tanh(z)`

Rectified Linear Unit (ReLU)



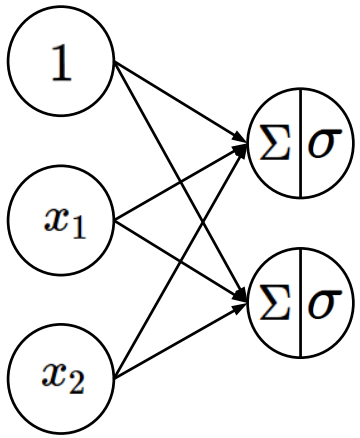
$$g(z) = \max(0, z)$$

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

`tf.nn.relu(z)`

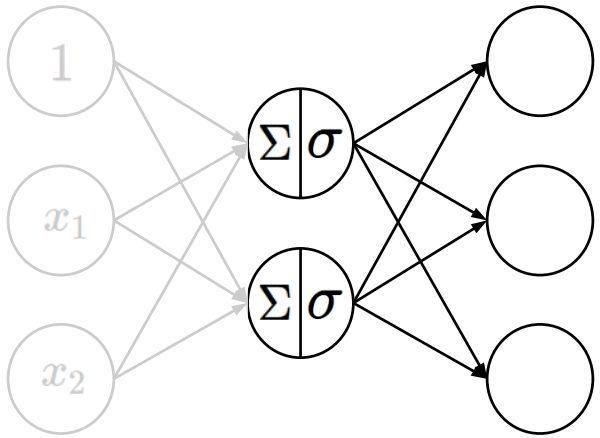
Artificial Neural Networks

- Why do we need multi-layers ?



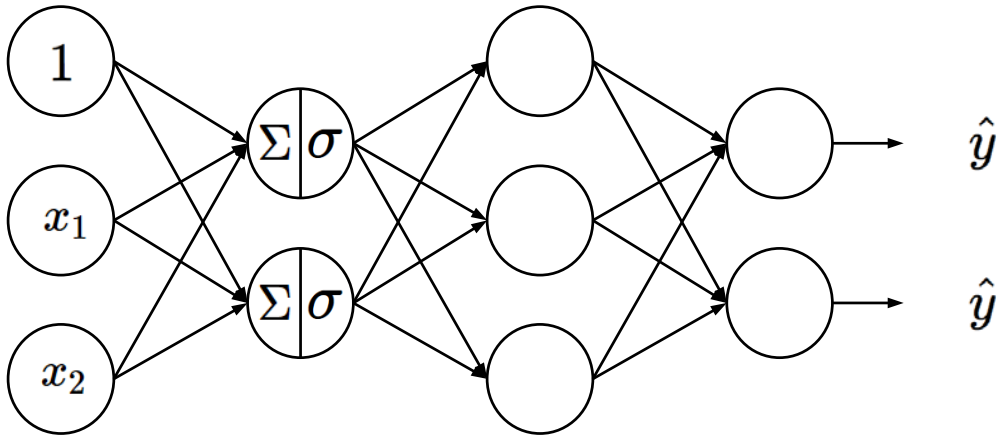
Artificial Neural Networks

- Why do we need multi-layers ?



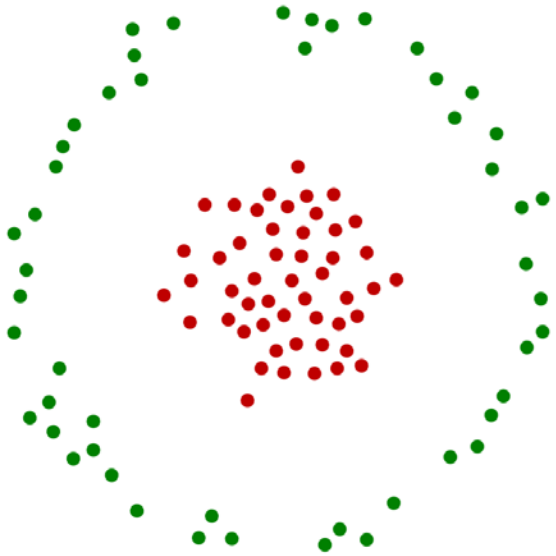
Artificial Neural Networks

- Why do we need multi-layers ?



Another Perspective:
ANN as Kernel Learning

Nonlinear Classification

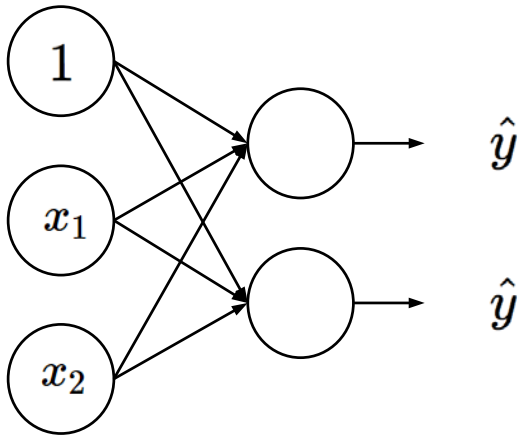


https://www.youtube.com/watch?v=3liCbRZPrZA

Neuron

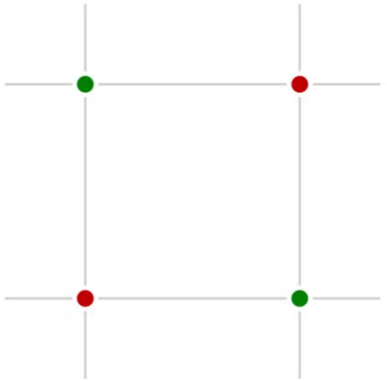
- We can represent this “neuron” as follows:

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



XOR Problem

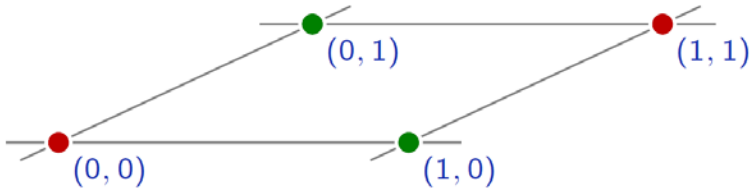
- The main weakness of linear predictors is their lack of capacity.
- For classification, the populations have to be linearly separable.



"xor"

Nonlinear Mapping

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

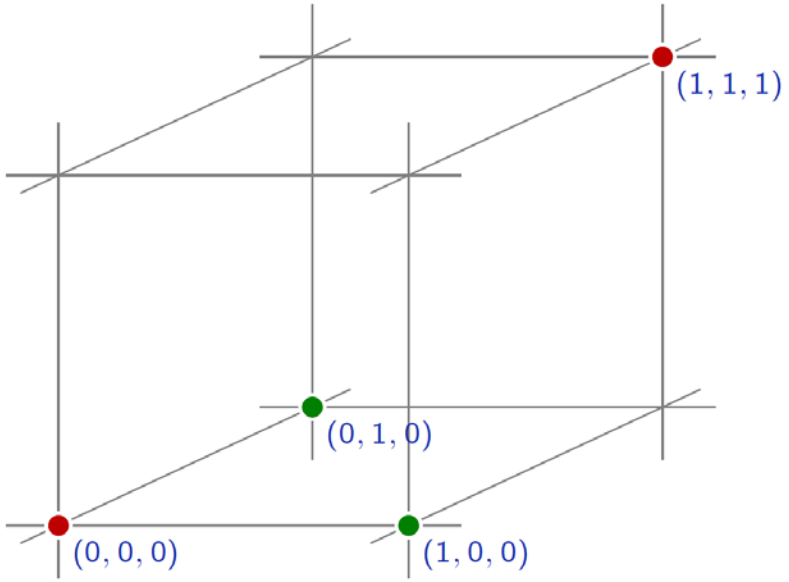


Source: Dr. Francois Fleuret at EPFL

Nonlinear Mapping

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

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

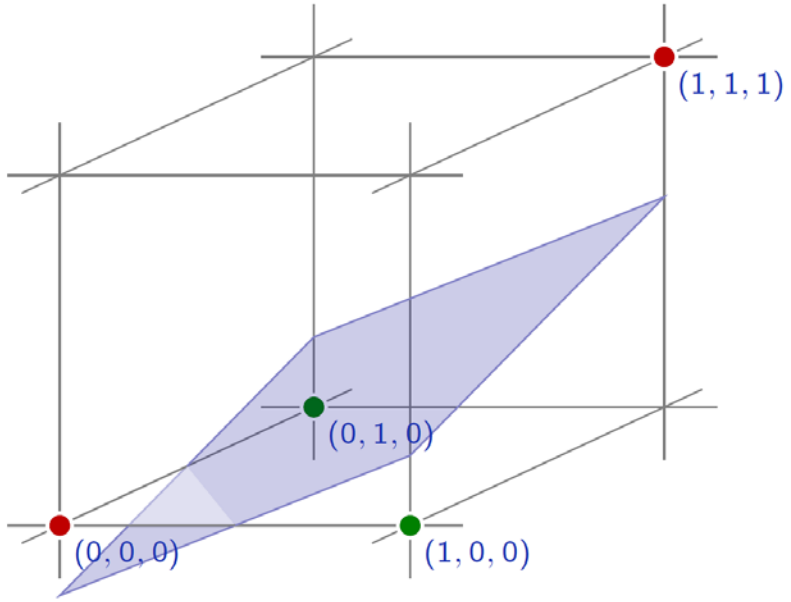


Source: Dr. Francois Fleuret at EPFL

Nonlinear Mapping

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

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

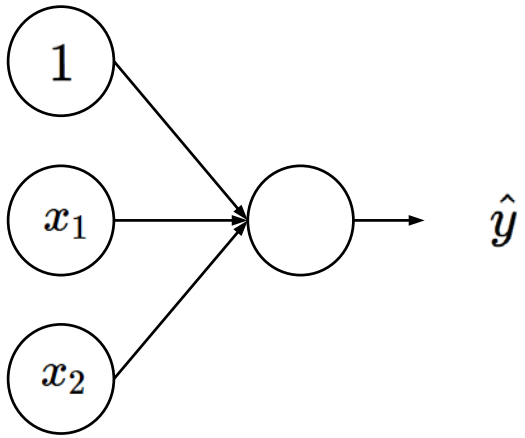


Source: Dr. Francois Fleuret at EPFL

Neuron

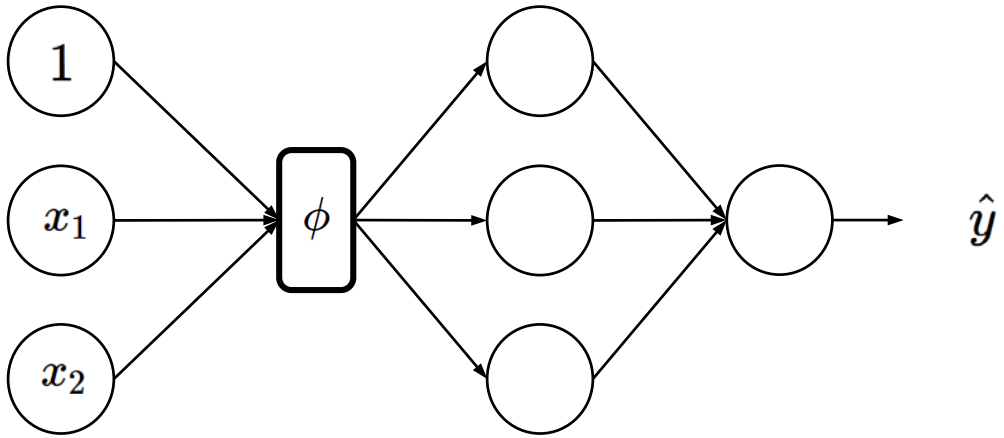
- We can represent this “neuron” as follows:

- Not linearly separable



Kernel + Neuron

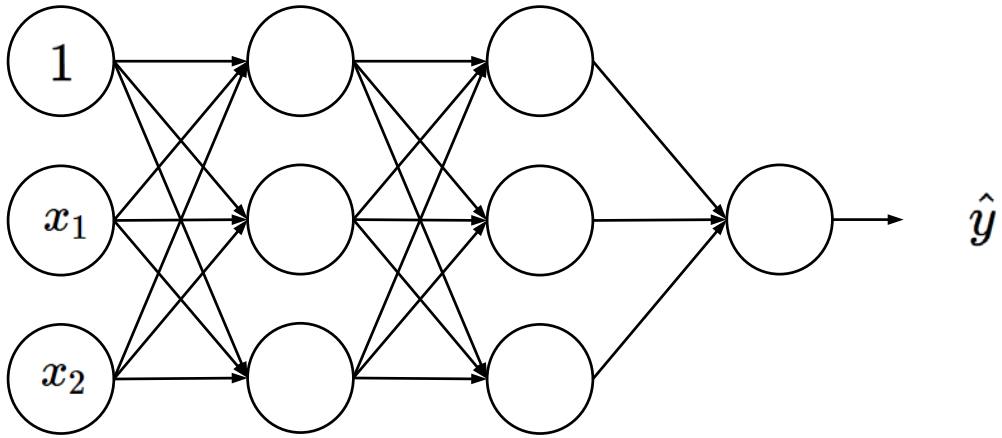
- Nonlinear mapping + neuron



ϕ: Kernel

Neuron + Neuron

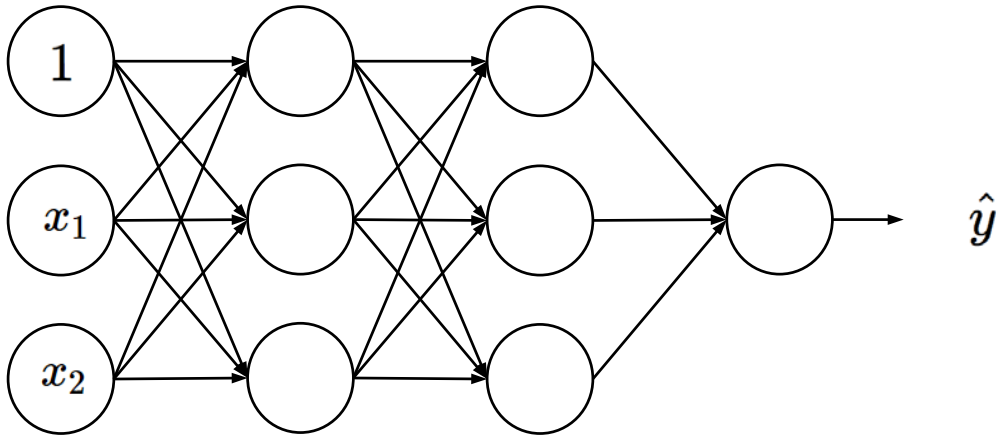
- Nonlinear mapping can be represented by another neurons



- Nonlinear Kernel
  - Nonlinear activation functions

Multi Layer Perceptron

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

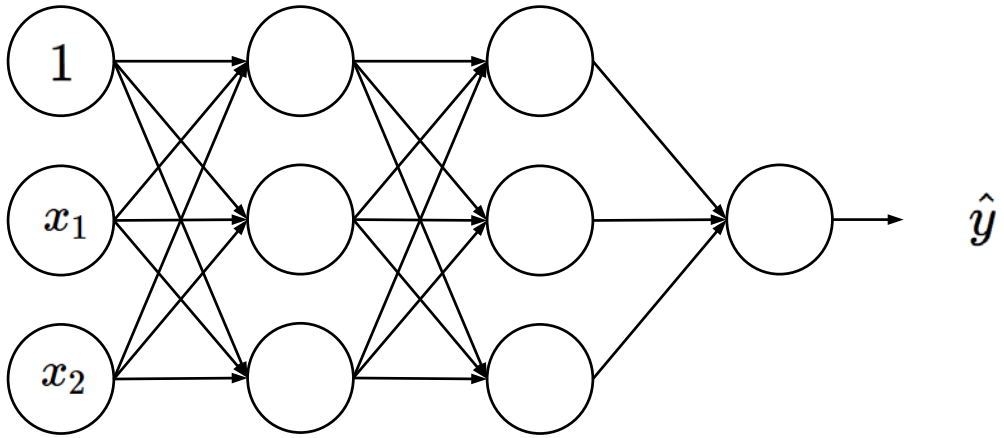


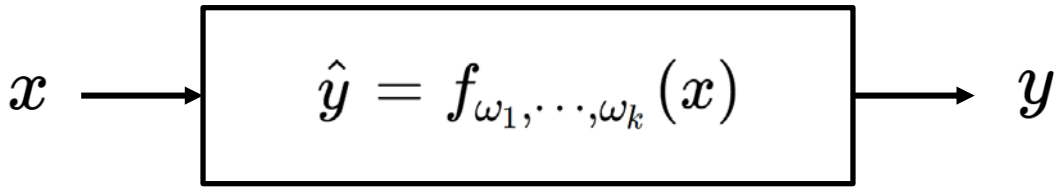
Summary

- Universal function approximator
- Universal function classifier

1) Value propagation point of view
2) Weight point of view

- Parameterized

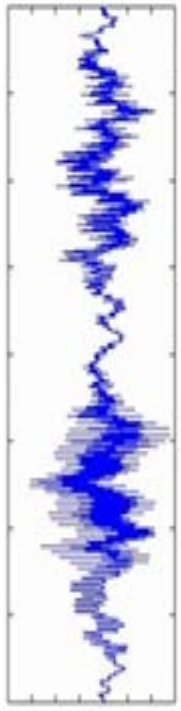


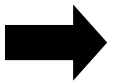


Artificial Neural Networks

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

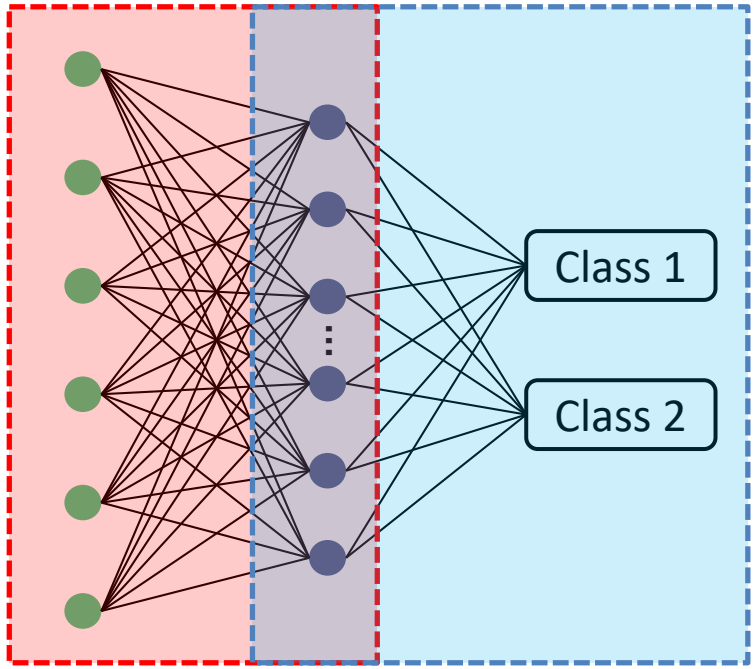
Input





Output

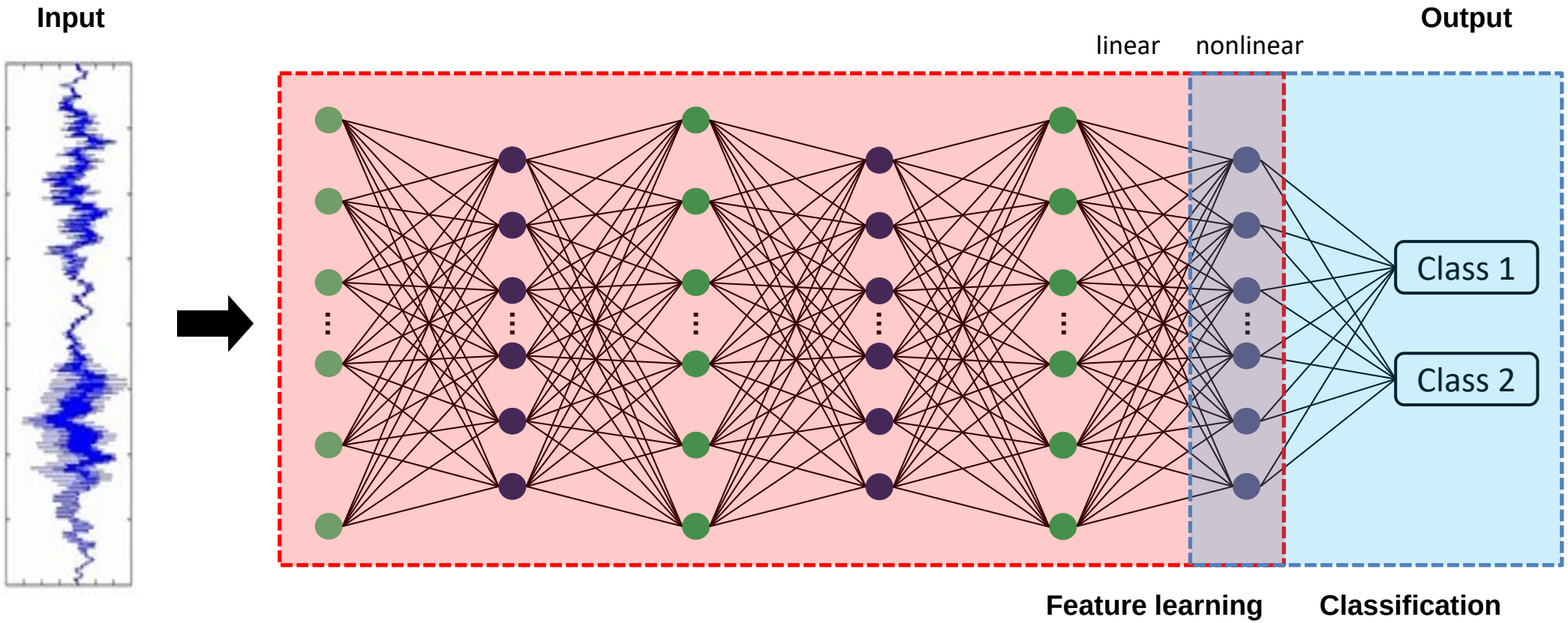
linear nonlinear



Feature learning **Classification**

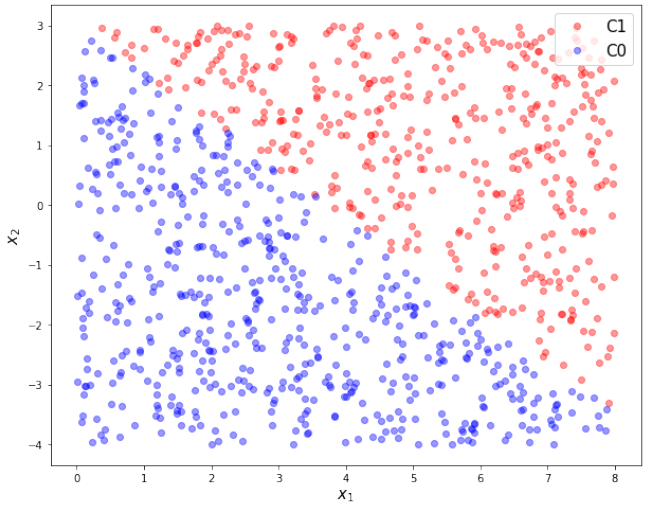
Deep Artificial Neural Networks

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

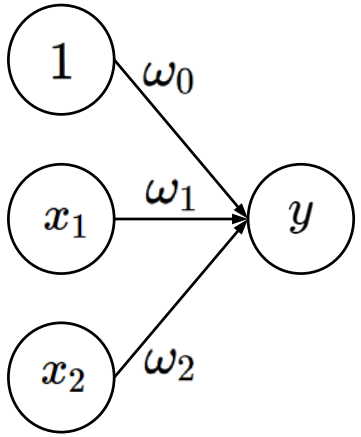


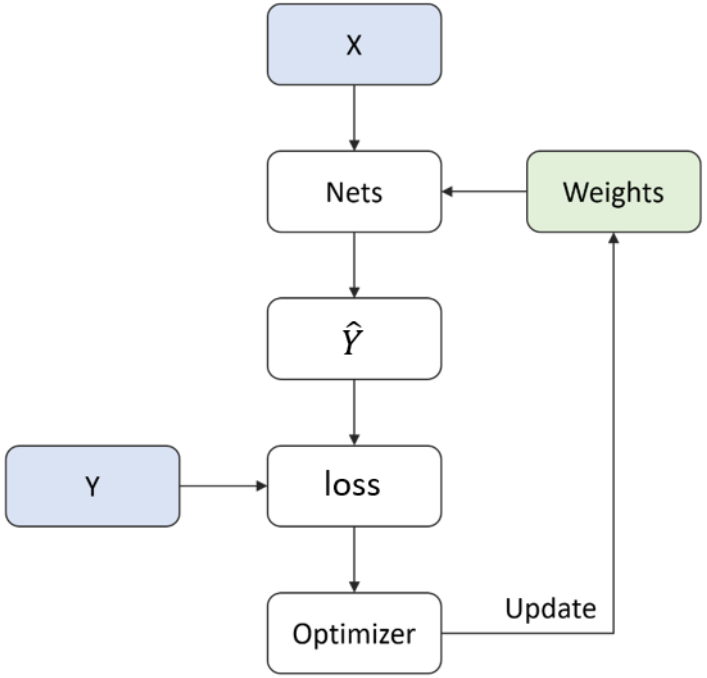
Looking at Parameters

Logistic Regression in a Form of Neural Network



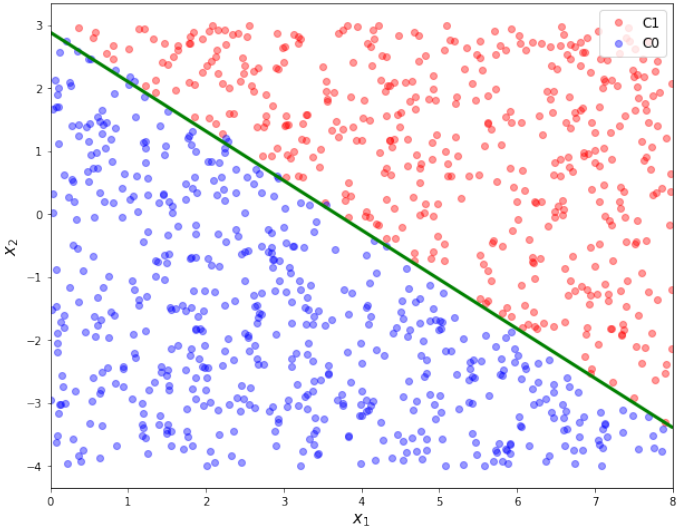
$$y = \sigma\left(\omega_0 + \omega_1 x_1 + \omega_2 x_2\right)$$



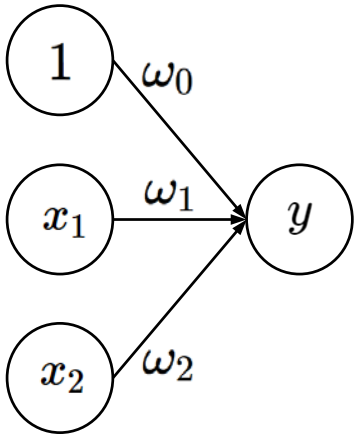


Logistic Regression in a Form of Neural Network

- Neural network convention

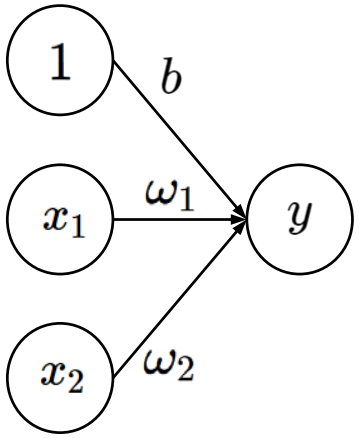


$$y = \sigma\left(\omega_0 + \omega_1 x_1 + \omega_2 x_2\right)$$

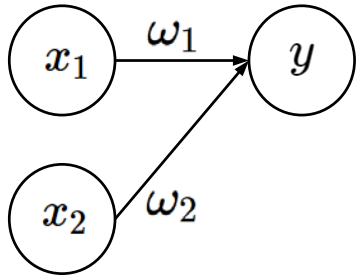


$\omega_0 \longrightarrow b$

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

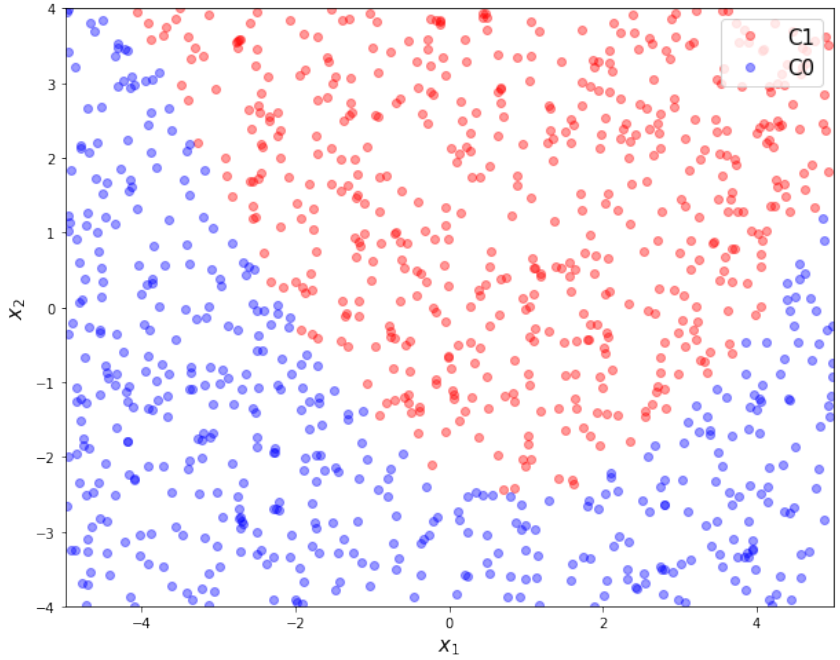


Do not indicate bias units



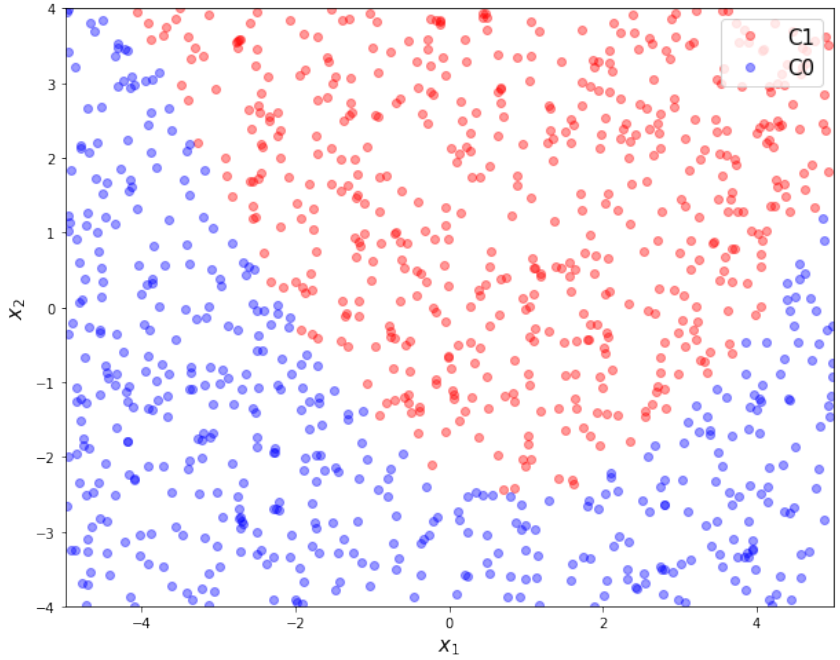
Nonlinearly Distributed Data

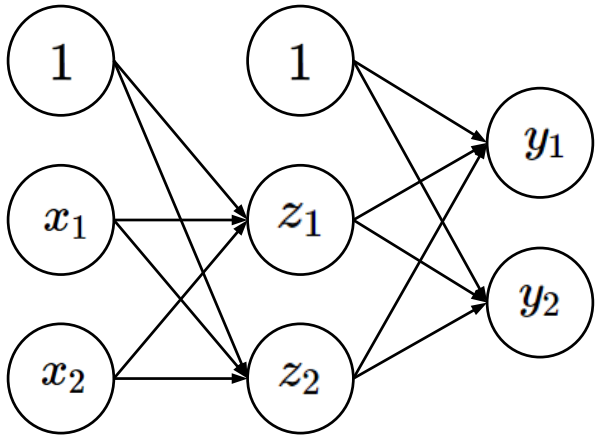
- Example to understand network's behavior
  - Include a hidden layer



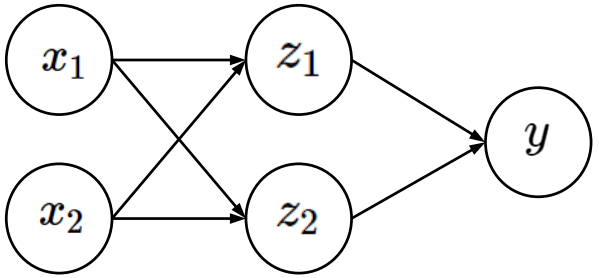
Nonlinearly Distributed Data

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





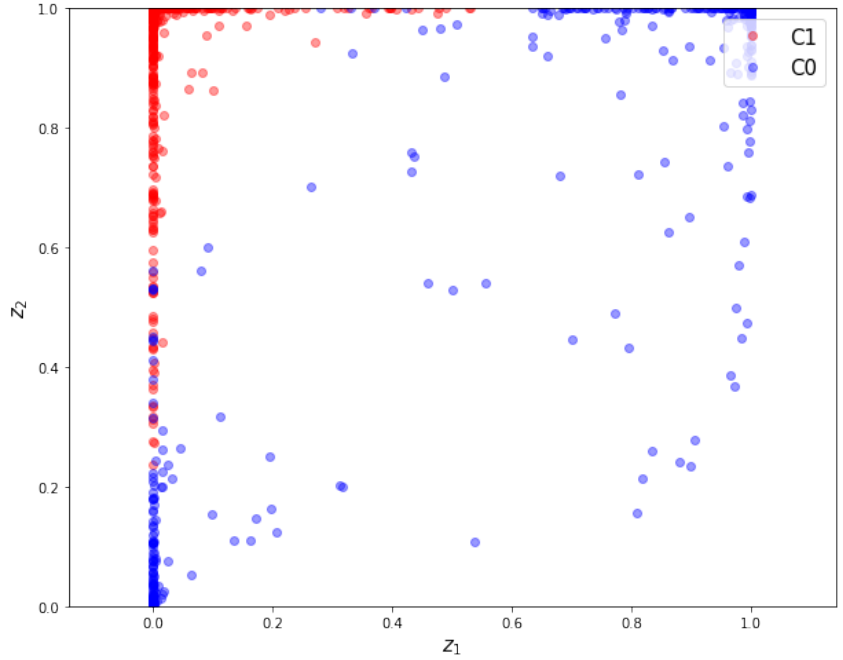
Do not include bias units

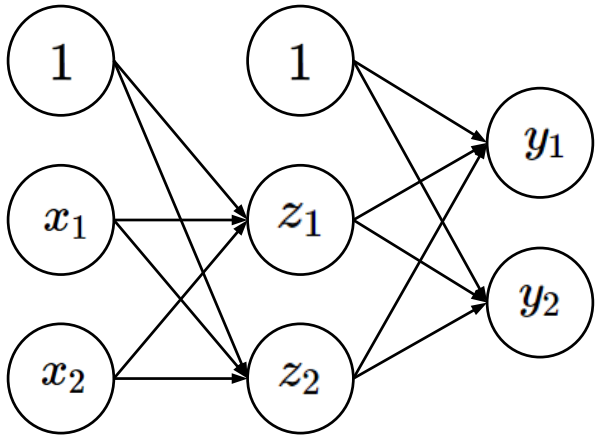


Multi Layers

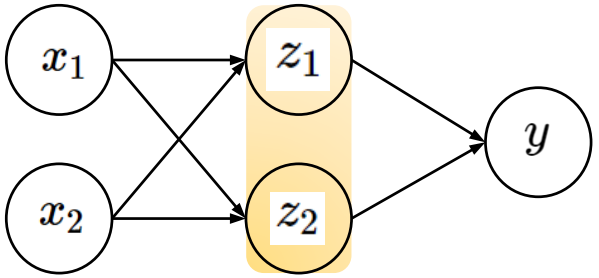
- z space

- Kernel
- Linearly separable



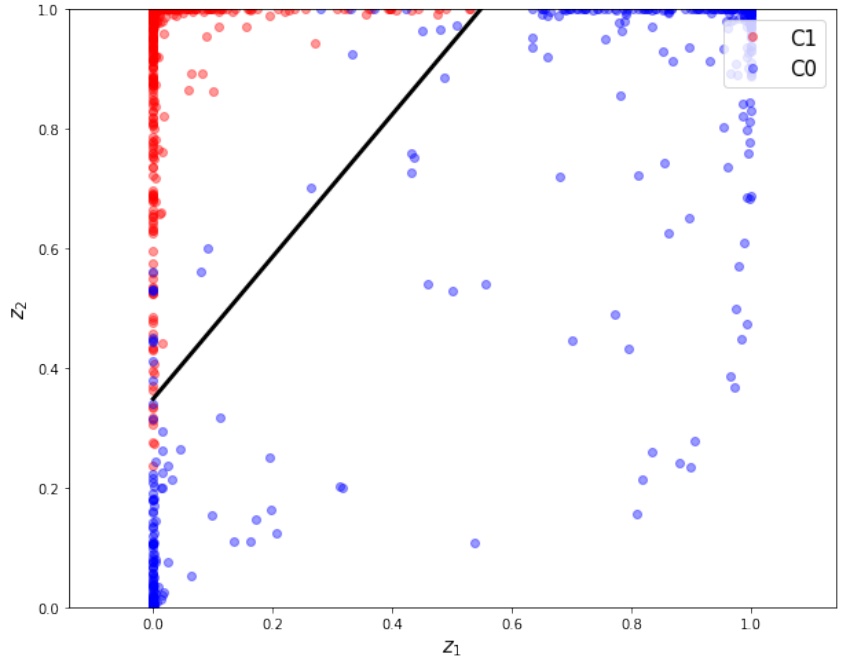


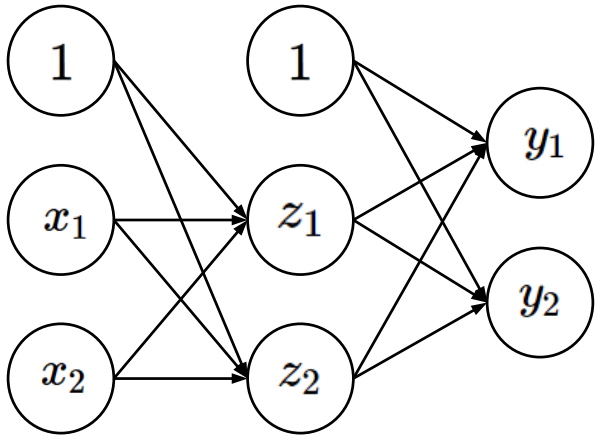
Do not include bias units



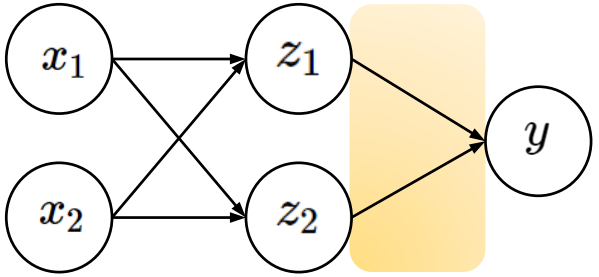
Multi Layers

- z space
- Linear classifier





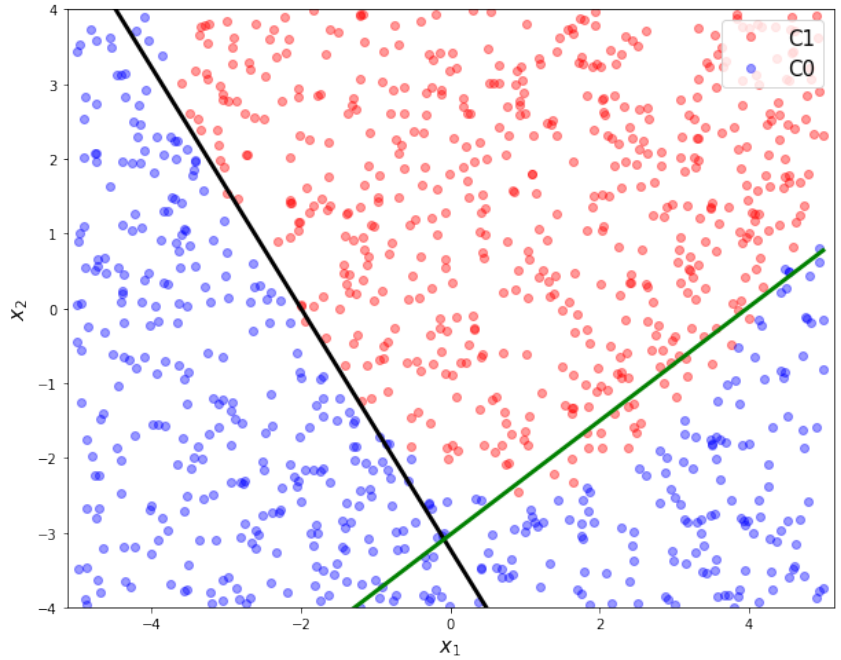
Do not include bias units

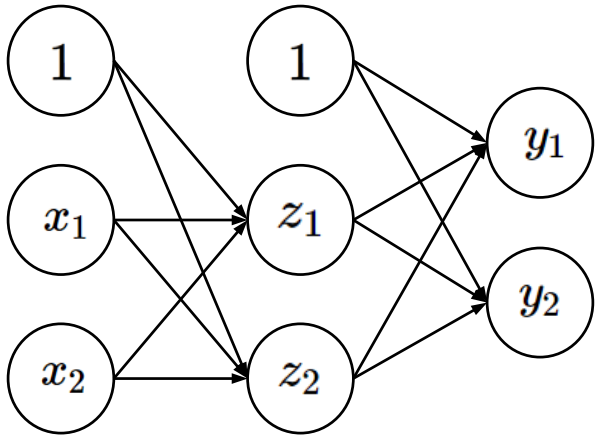


Multi Layers

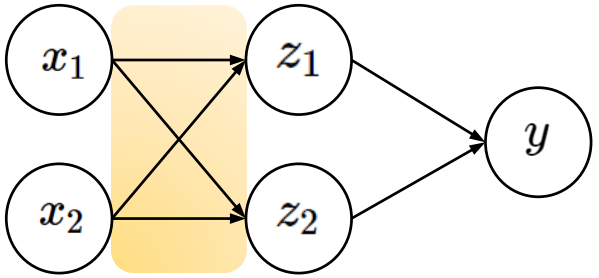
- x space

- Nonlinear classifier in original space
- Piecewise linear classification boundaries



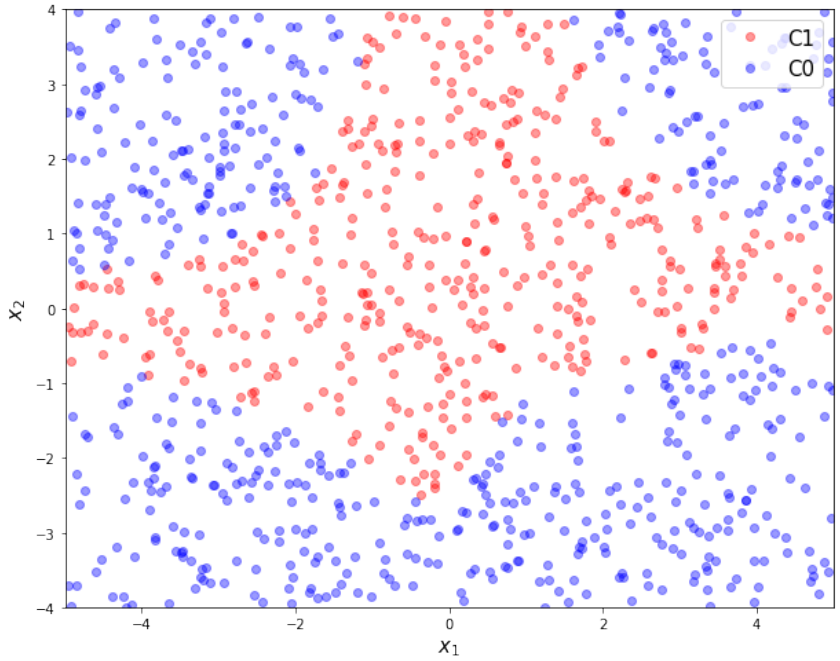


Do not include bias units



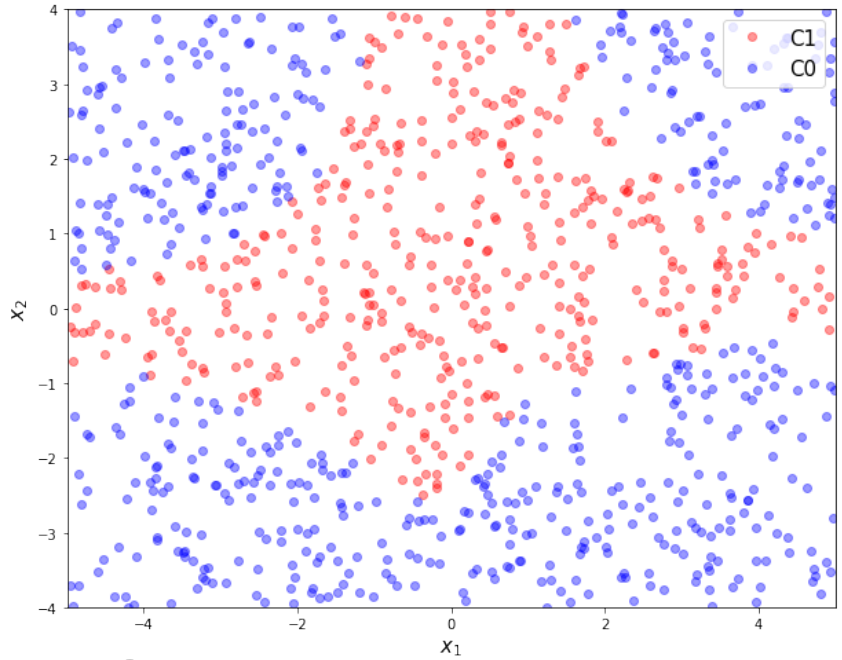
Nonlinearly Distributed Data

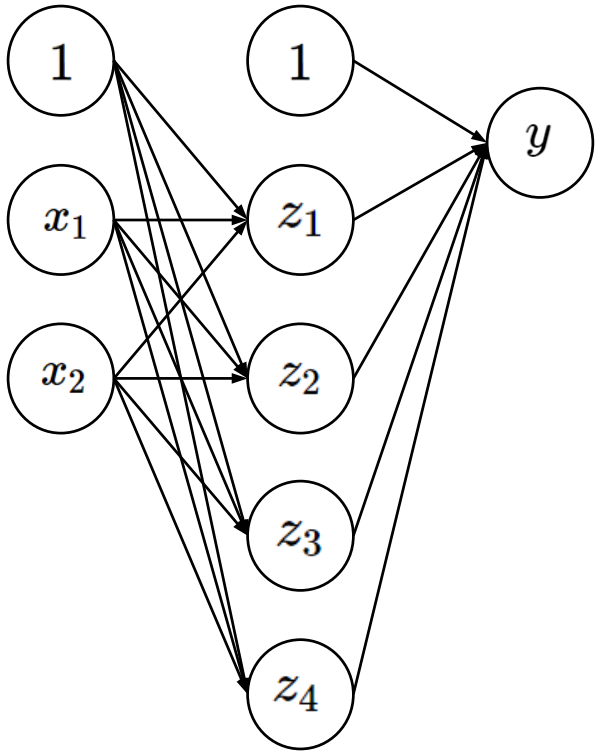
- More neurons in hidden layer



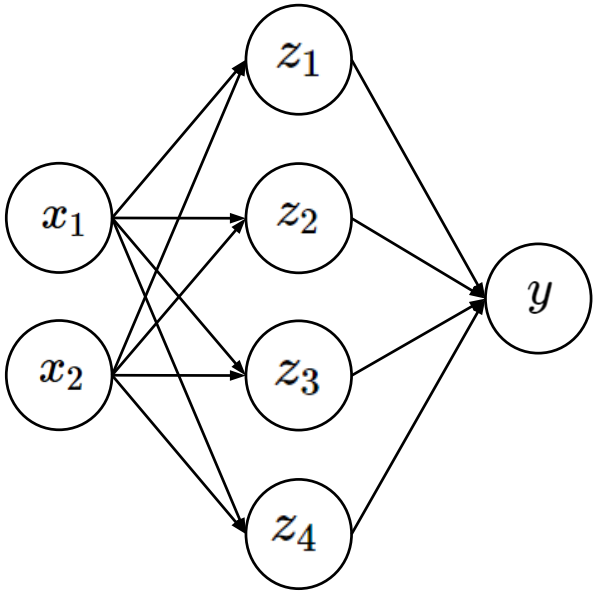
Nonlinearly Distributed Data

- More neurons in hidden layer



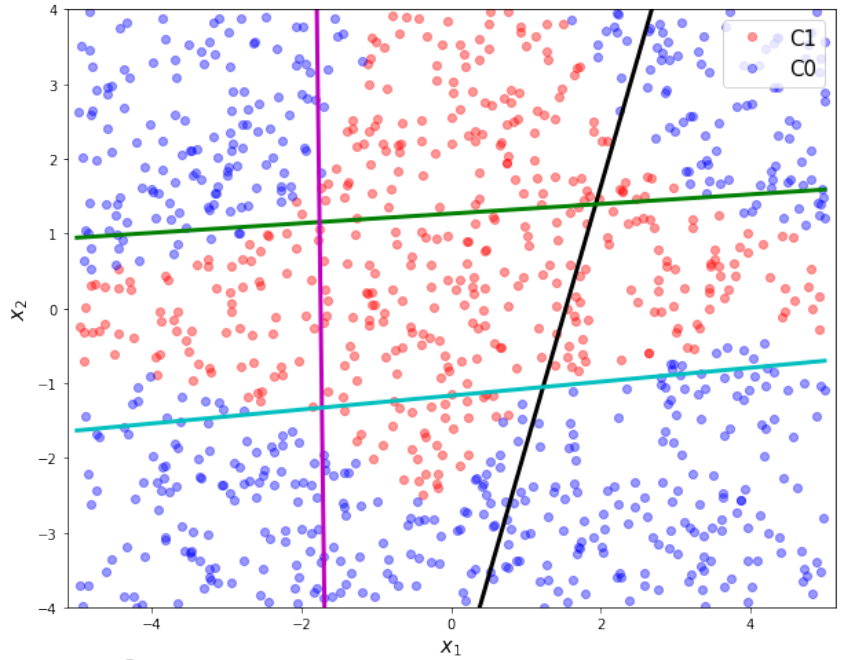


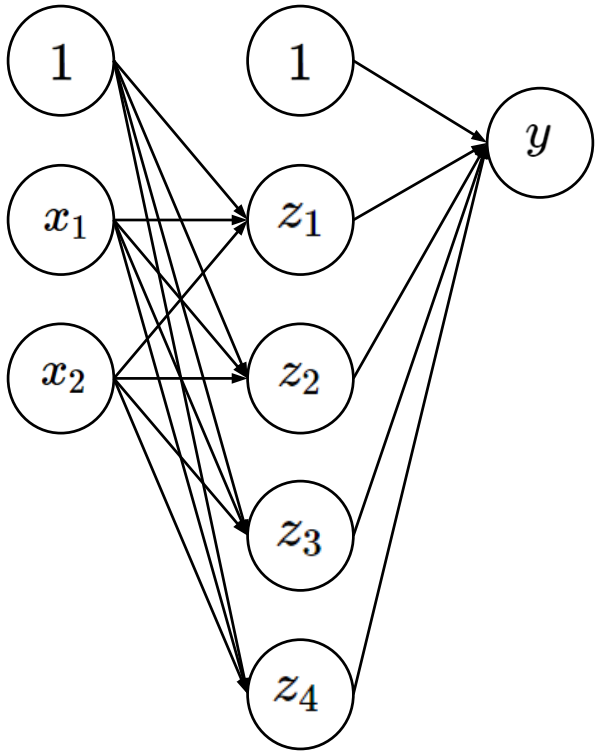
Do not include bias units



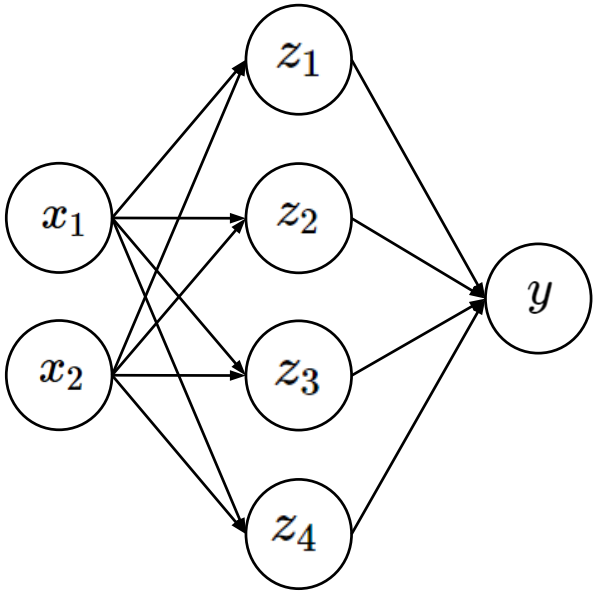
Multi Layers

- Multiple linear classification boundaries





Do not include bias units



(Artificial) Neural Networks: Training

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):

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

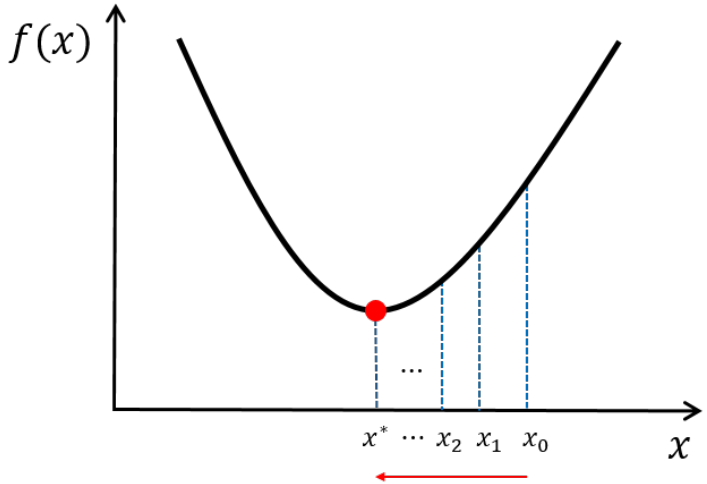
  - Cross entropy (for classification):

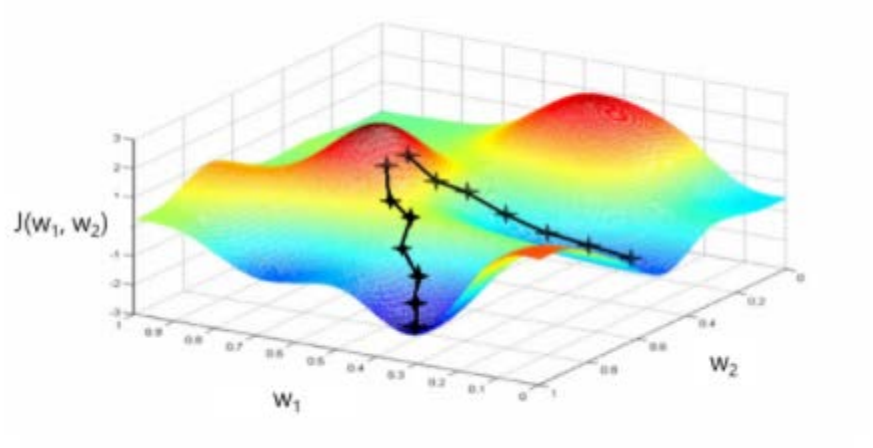
$$-\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log\left(h_{\omega}\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_{\omega}\left(x^{(i)}\right)\right)$$

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)

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

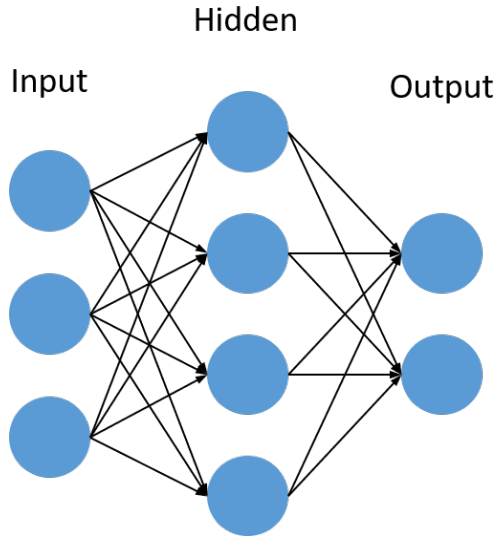


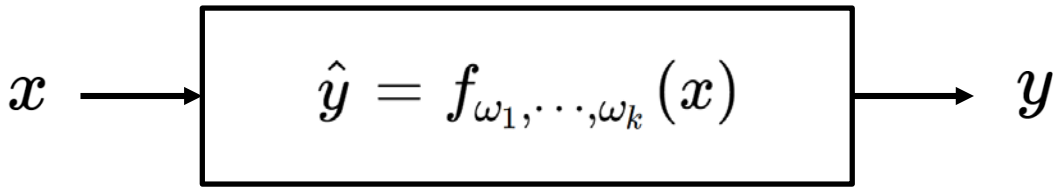


Gradients in ANN

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

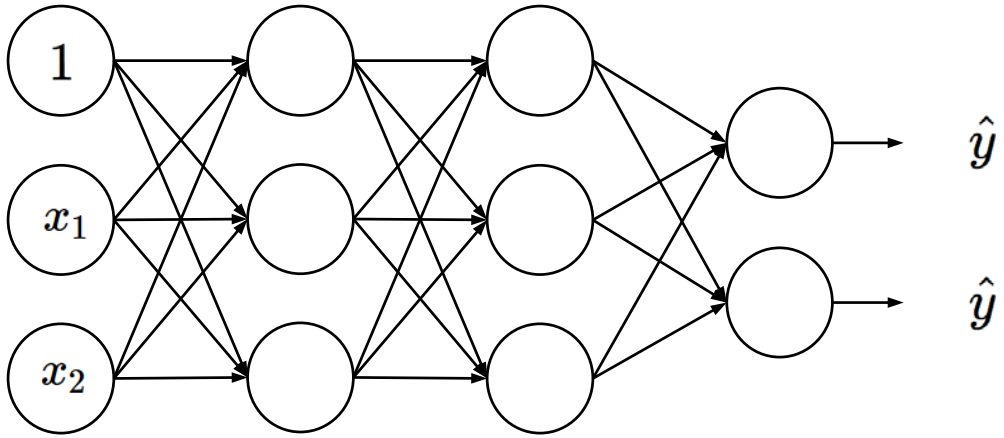
- Structural constraint of NN:
  - Composition of functions
  - Chain rule
  - Dynamic programming





정리하자.

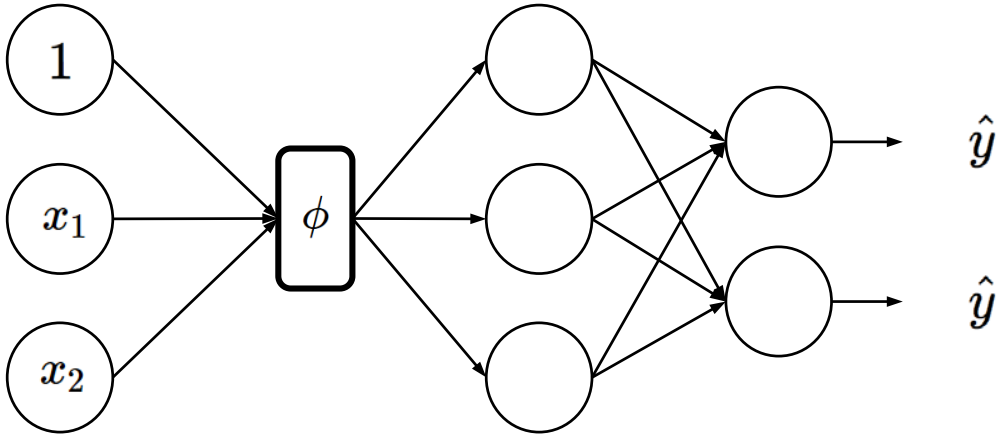
- 데이터에서 인공신경망을 학습한다는 것은
- 입력과 출력 관계를 잘 나타내는 weights and bias 를 gradient descent 방법을 사용해서 찾는다는 것이다.

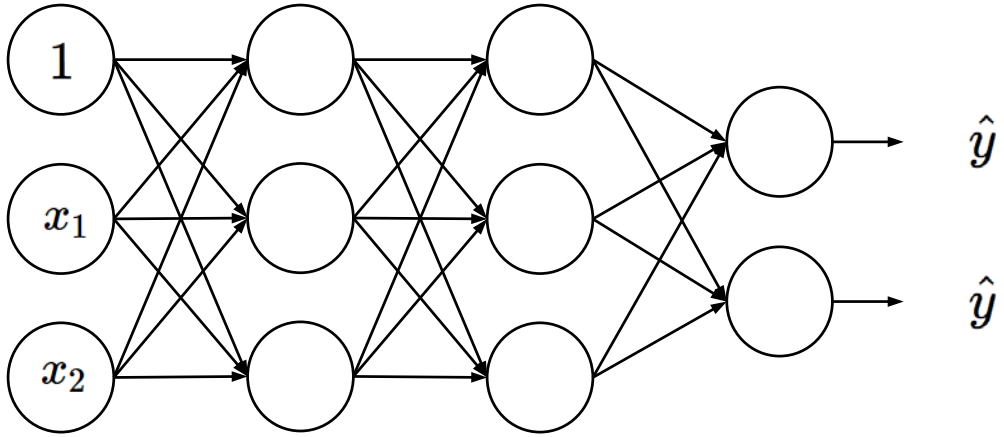


- 인공신경망의 구조는 사용자가 디자인해야한다.
- TensorFlow 가 weights and bias 는 찾아준다.

Machine Learning vs. Deep Learning

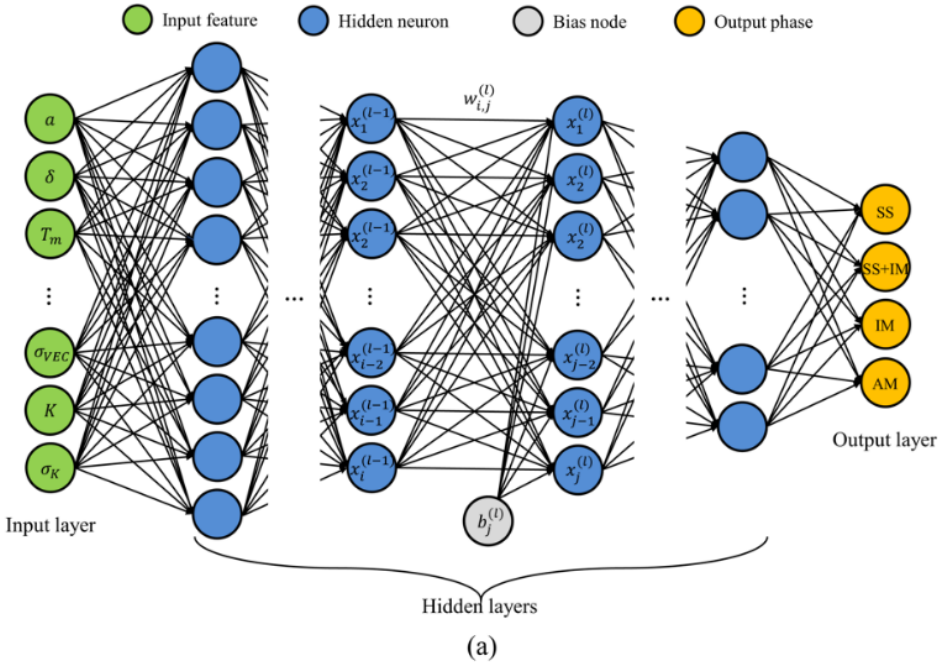
- Artificial intelligence (AI) refers to the ability of machines to mimic human intelligence without explicit programming

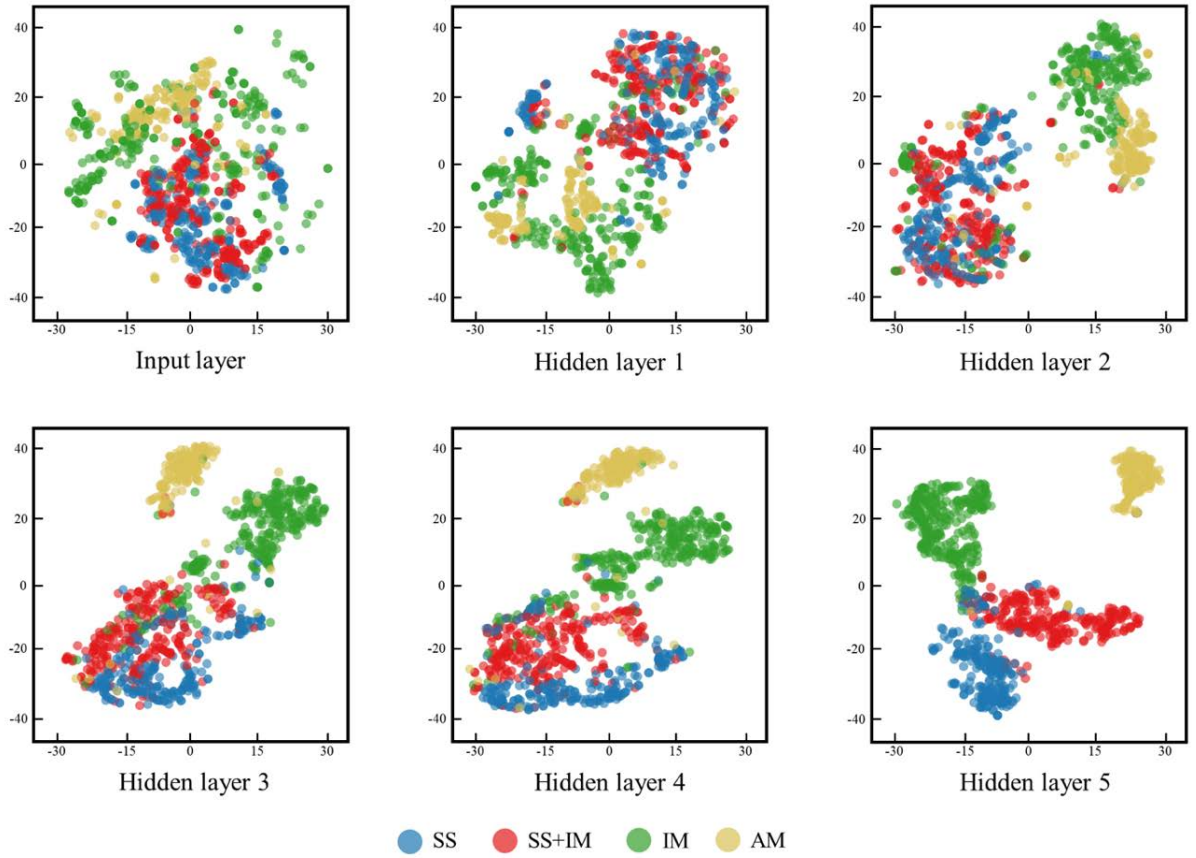




- Feature engineering
- Feature learning

Deep Learning

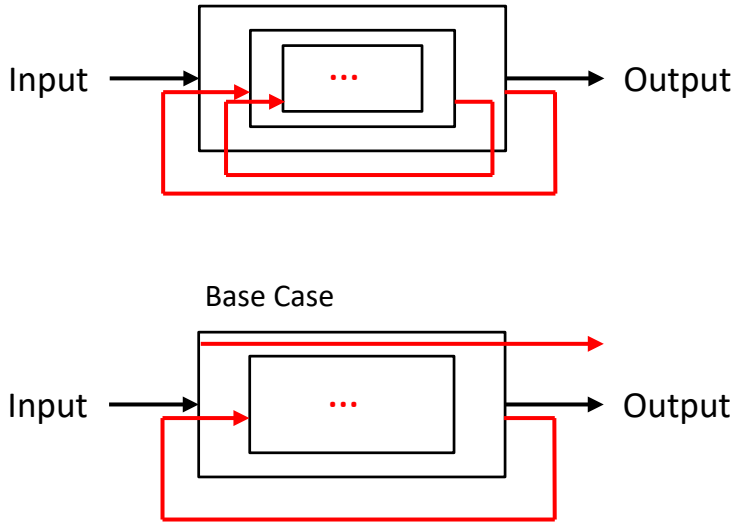




Dynamic Programming

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

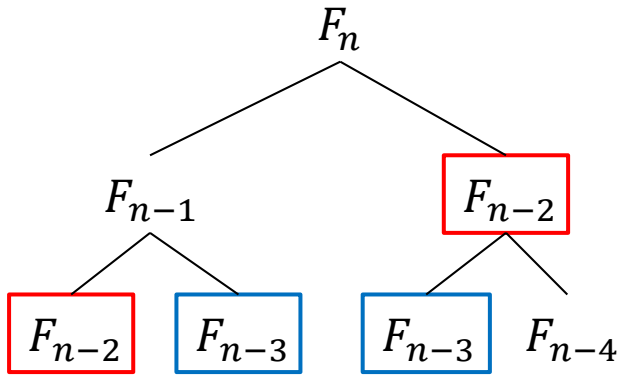
- Fibonacci numbers:

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

Naïve Recursive Algorithm

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

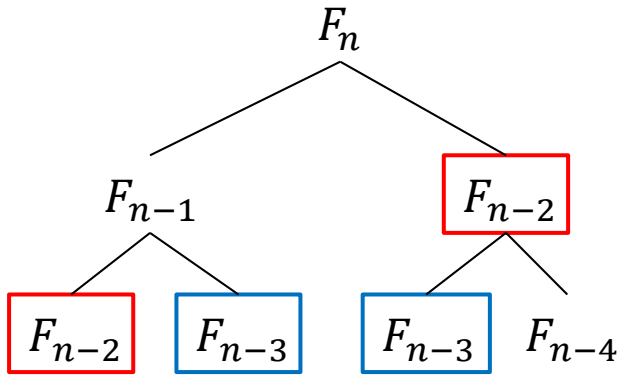
- It works. Is it good?



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

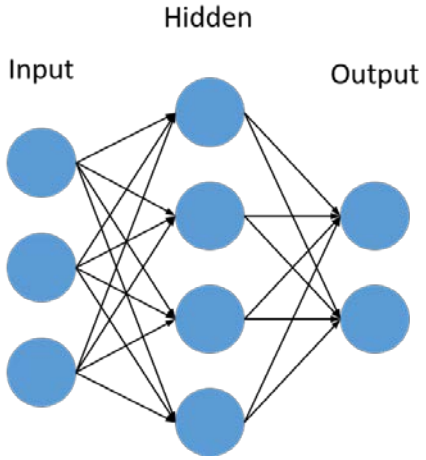
- DP ≈ recursion + memorization

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



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

Training Neural Networks: Backpropagation Learning

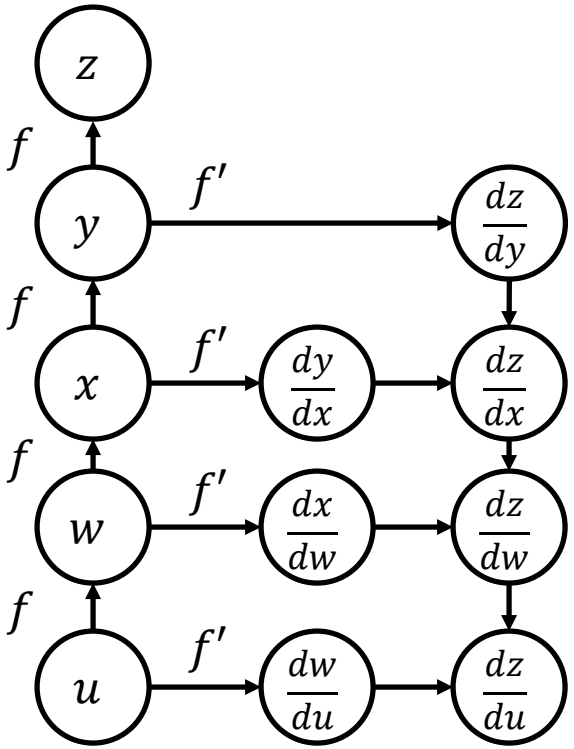
- Forward propagation
  - the initial information propagates up to the hidden units at each layer and finally produces output

- Backpropagation
  - allows the information from the cost to flow backwards through the network in order to compute the gradients

Backpropagation

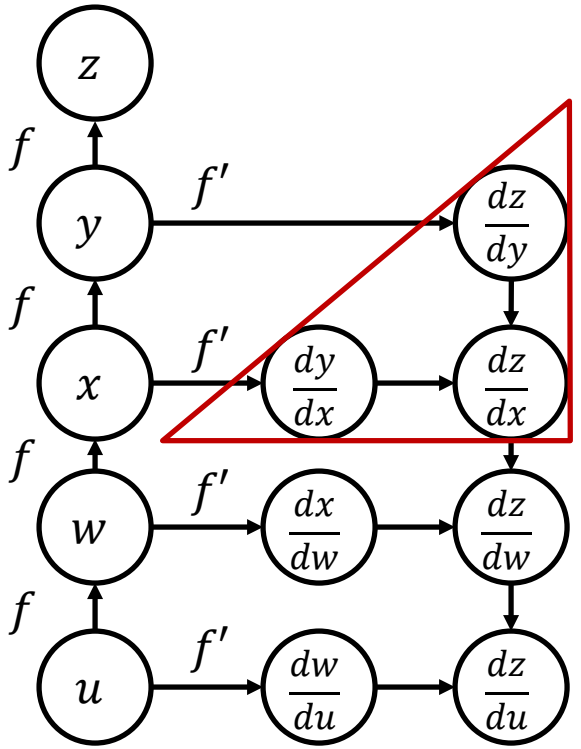
- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy}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

- Backpropagation
  - Update weights recursively



Backpropagation

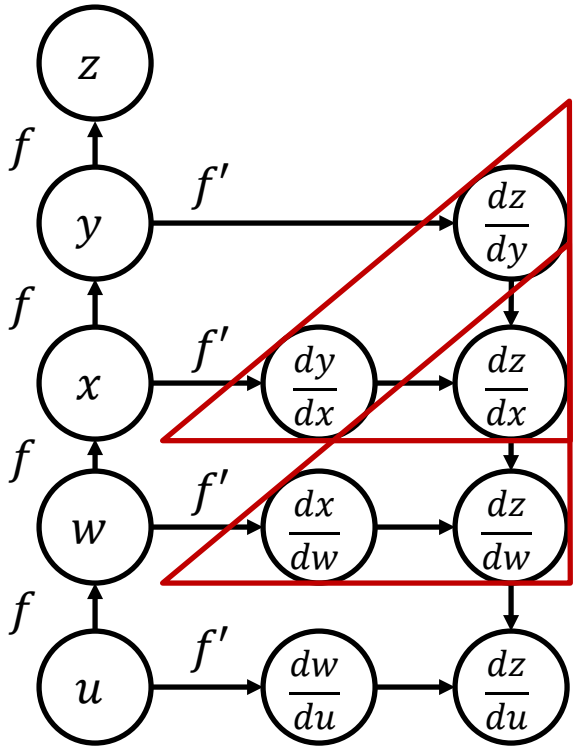
- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy}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



Backpropagation

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

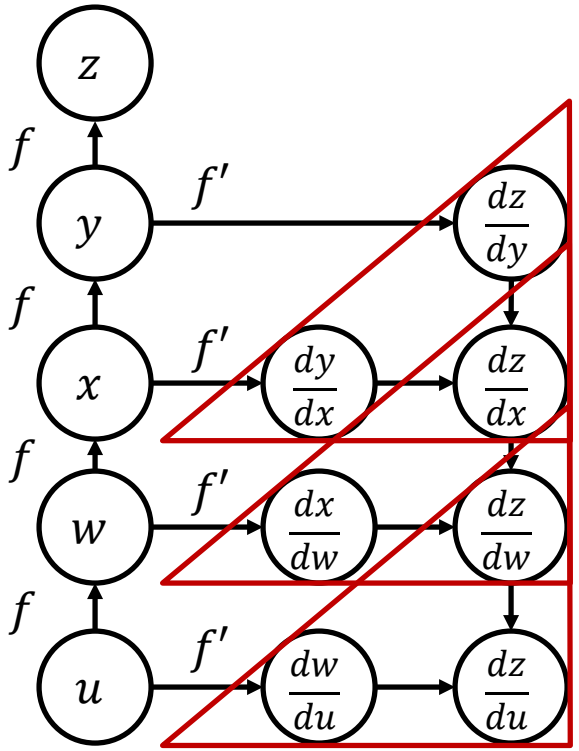
- Backpropagation
  - Update weights recursively



Backpropagation

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

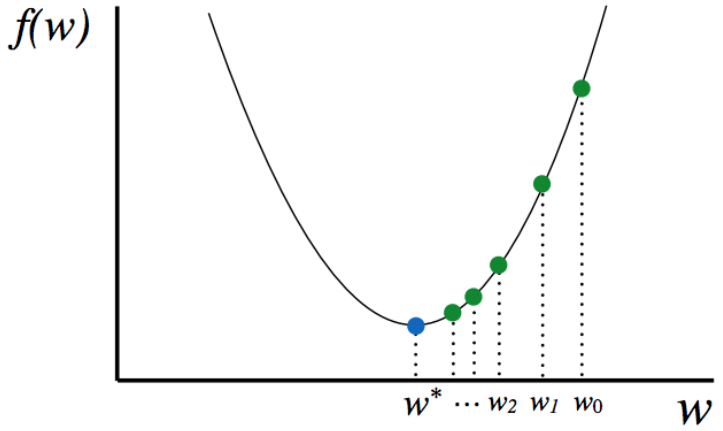
- Backpropagation
  - Update weights recursively with memory



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

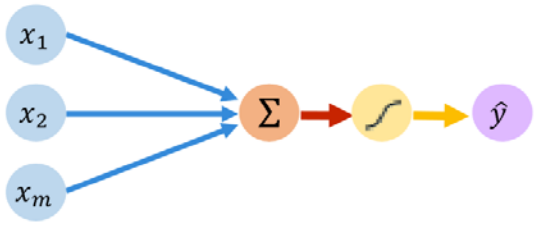


- It is not easy to numerically compute gradients in network in general.
  - The good news: people have already done all the "hard work" of developing numerical solvers (or libraries)
  - There are a wide range of tools → We will use the TensorFlow

Core Foundation Review

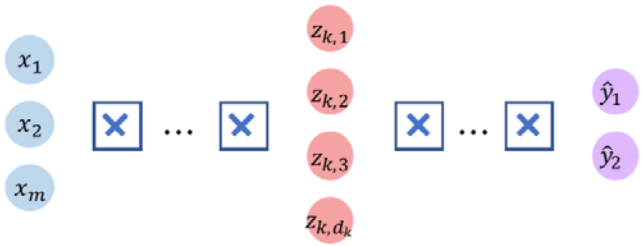
The Perceptron

- Structural building blocks
- Nonlinear activation functions



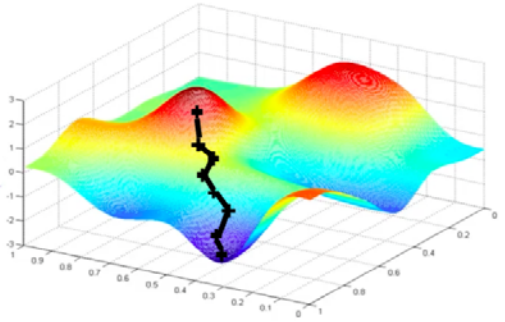
Neural Networks

- Stacking Perceptrons to form neural networks
- Optimization through backpropagation



Training in Practice

- Adaptive learning
- Batching
- Regularization



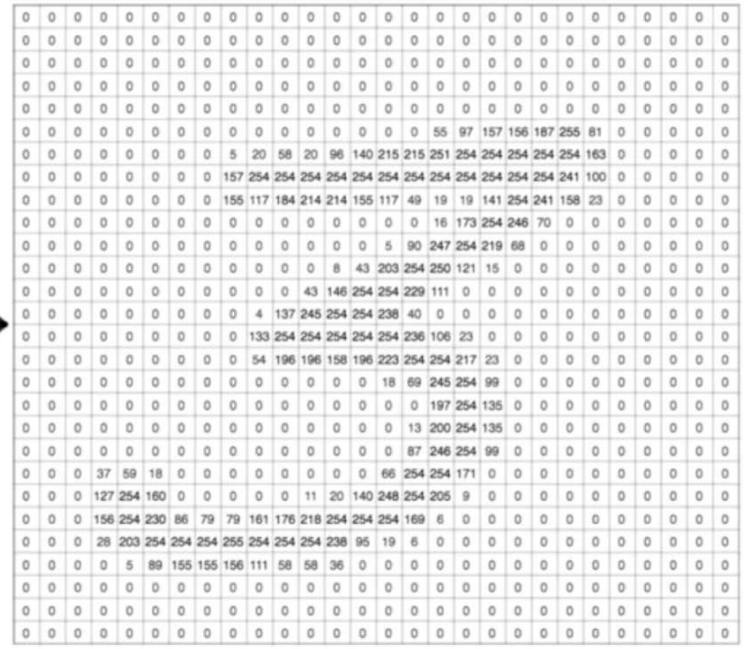
Source: 6.S191 Intro. to Deep Learning at MIT

(Artificial) Neural Networks with TensorFlow

MNIST database

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

28 x 28
= 784 pixels



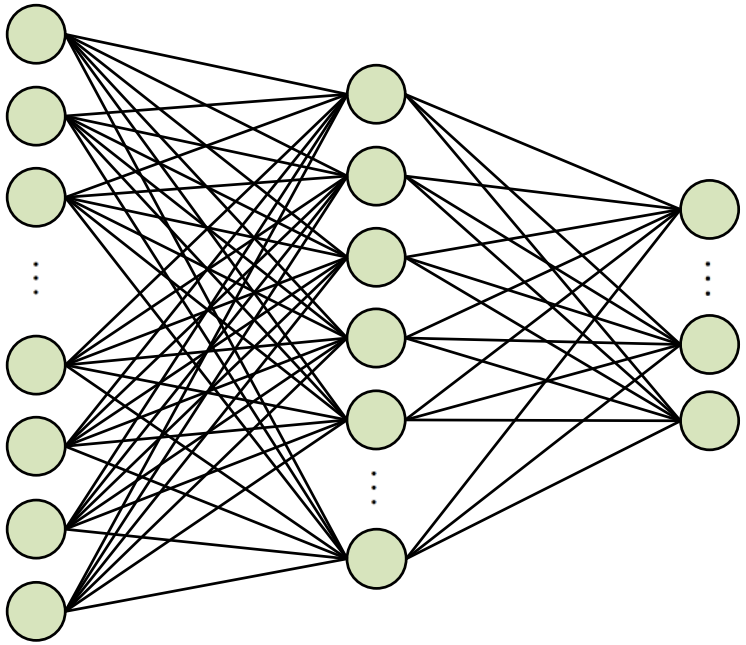
pixel 1
pixel 2
pixel 3
pixel 4
pixel 5
pixel 6
pixel 7
pixel 8
pixel 9
pixel 10
pixel 11
pixel 12
pixel 13
pixel 14
pixel 15
pixel 16
pixel 17
pixel 18
pixel 19

pixel 784

ANN in TensorFlow: MNIST

Our Network Model

flattened



digit prediction
in one-hot-encoding

Input image
(28 X 28)

Input layer
(784)

hidden layer
(100)

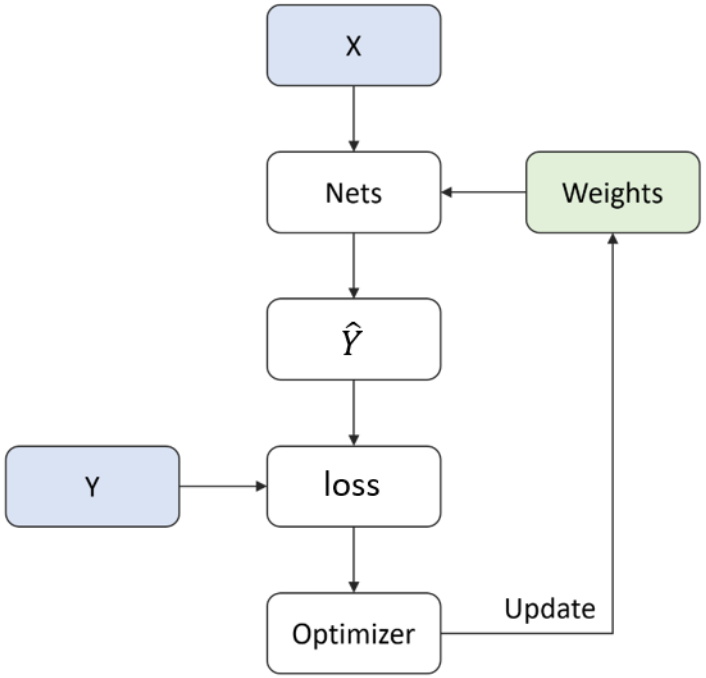
output layer
(10)

Iterative Optimization

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

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

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

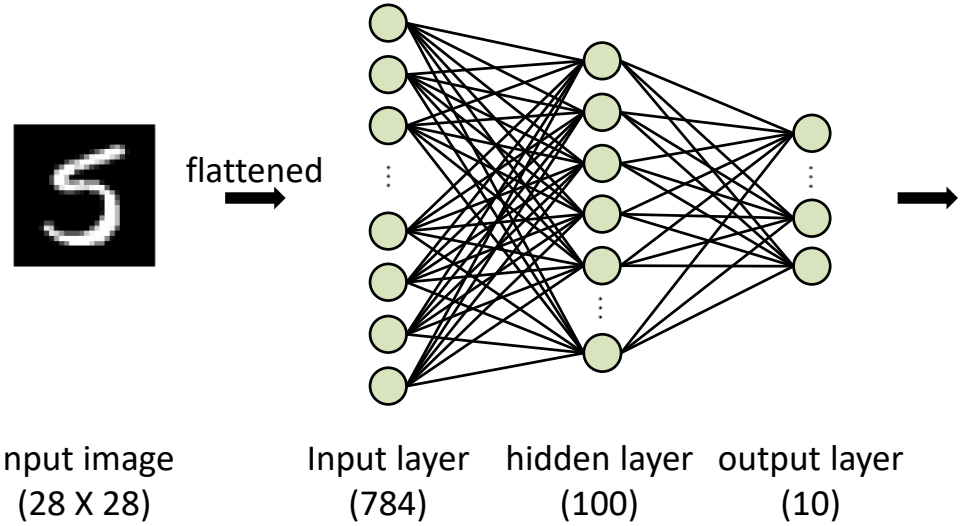


Implementation in Python

```
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
```

```
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)
```

```
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)
```