Deep Learning Fundamentals

Takuma Kimura

Chapter 1
Artificial Neural Network

1. What is Deep Learning?

What is Deep Learning?

- Artificial Intelligence (AI)

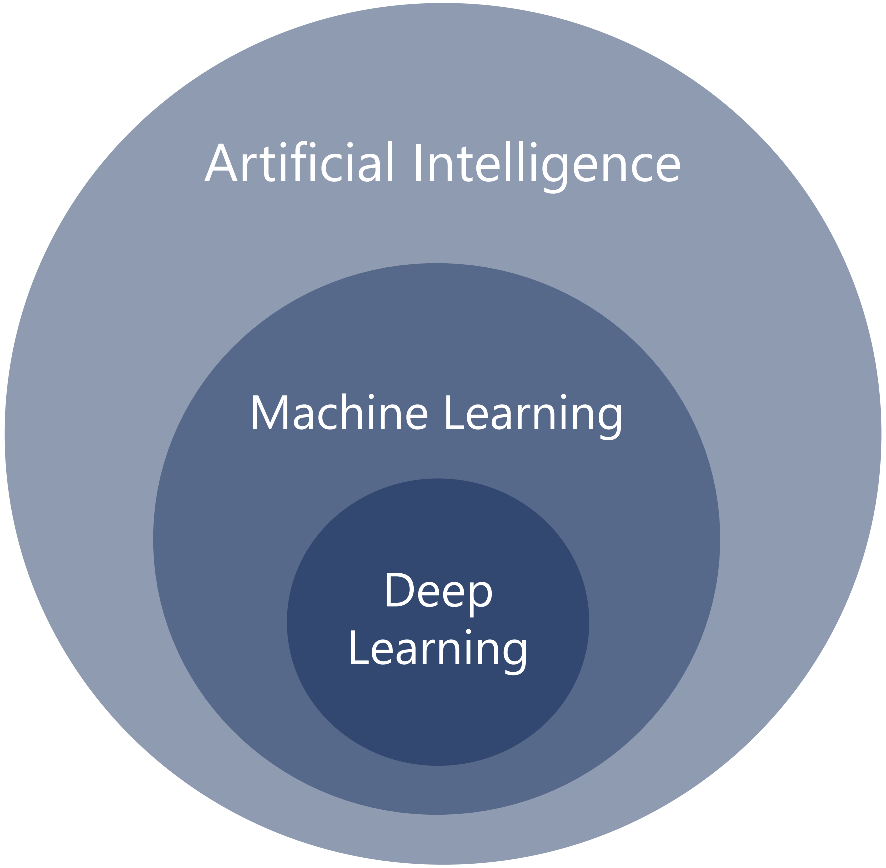
  Imitate intelligent human behavior and cognitive functions.

- Machine Learning (ML): A branch of AI.

  Automatically learn from data and make prediction or judgement.

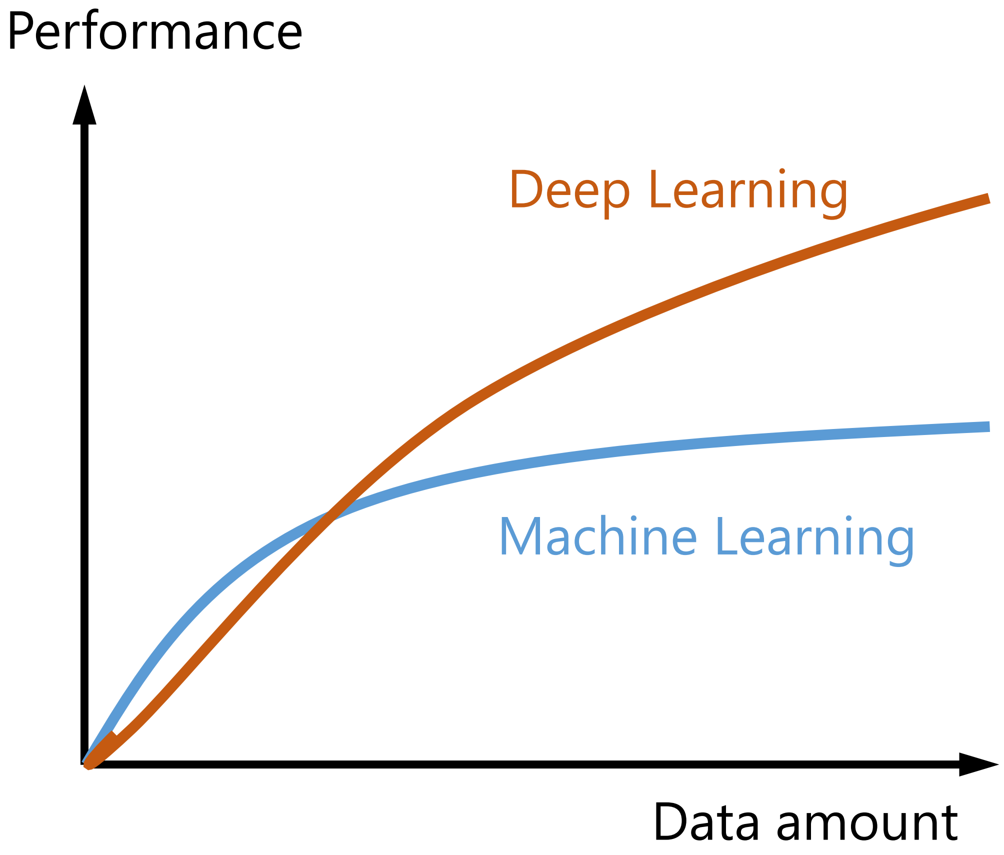
- Deep Learning (DL): A type of ML.

  Use artificial neural network to solve complex problems.



Limitations of Traditional Machine Learning

- Cannot handle high-dimensional data
- Need manual feature extraction
- Not appropriate for image and movie data analysis.



Challenge in Handling High Dimensional Data

e.g., Employee turnover prediction using data of 3,000 employees

Gender	Job	Age	Performance Score	Title
• Male • Female • Others	• Admin • IT • Operation • Others	• 20-29 • 30-39 • 40-49 • 50-59 • 60-	7: Excellent ・・・・ 1: Poor	• Manager • Supervisor • Rank-and-filer
3 types	12 types 3×4	60 types 12×5	420 types 60×7	1,260 types 420×3

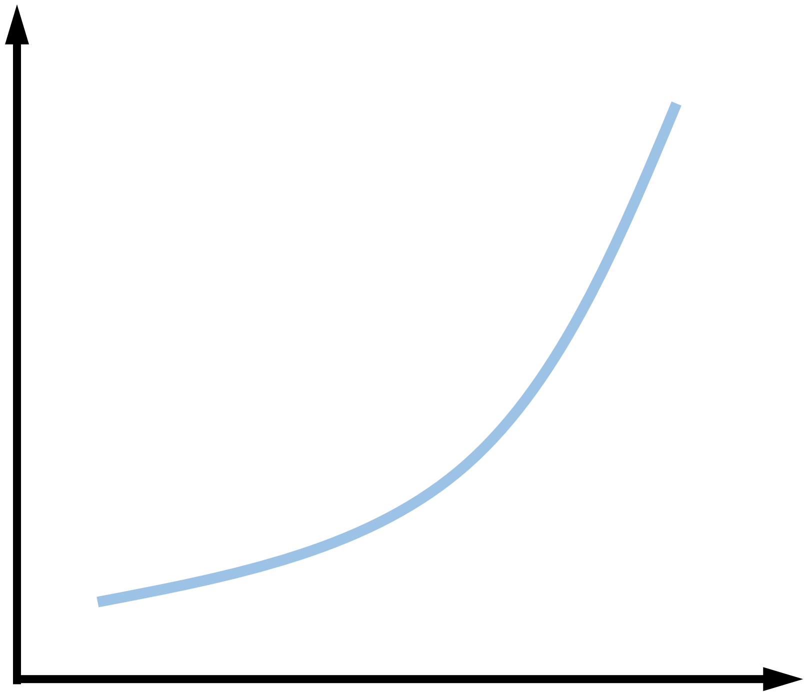
1,260 types in 3,000 employees... → Each type has only 2.5 employees.

Curse of Dimensionality

- The increase in the dimension of data results in the exponential increase in the amount of training data required to develop a generalizable machine learning model.

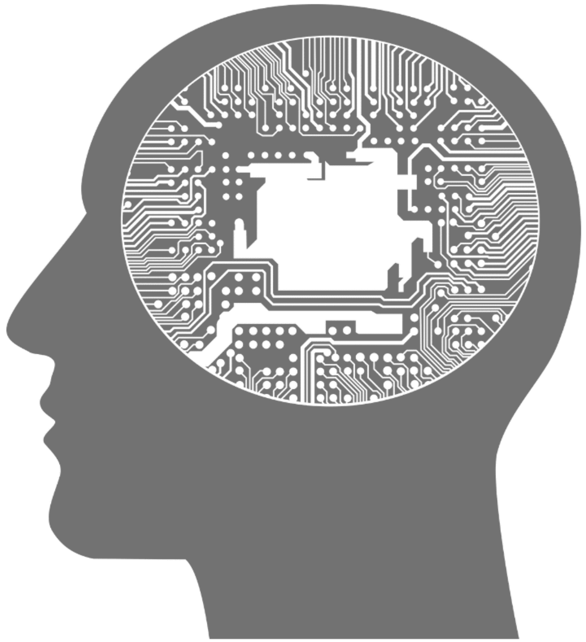
- The increase in the dimension lowers the similarity between samples in the training dataset.

→ The model will be likely to overfit.



Advantage of Deep Learning

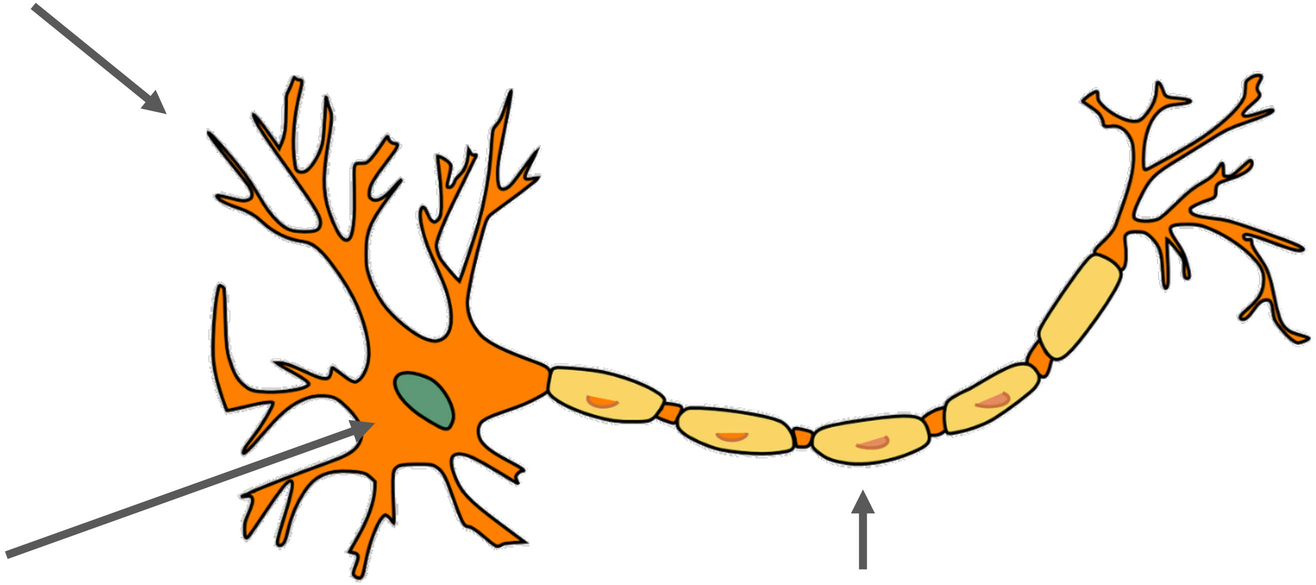
- Deep learning enables us to extract importance features automatically.
- Artificial neural network (ANN) is a better way to extract low dimensional features from high-dimensional data.
- The use of ANN enables us to alleviate the problem of “curse of dimensionality.”



2. Artificial Neural Network

Neurons

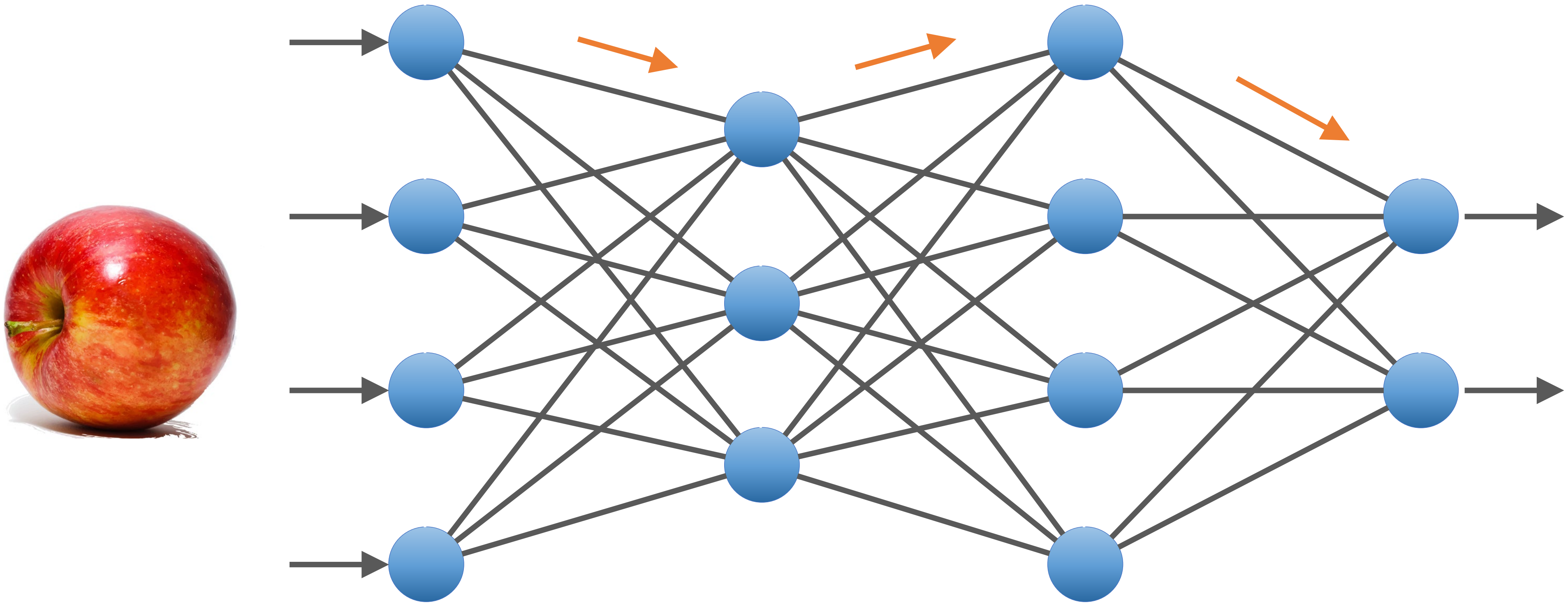
Dendrite:
Receive signals from other neurons



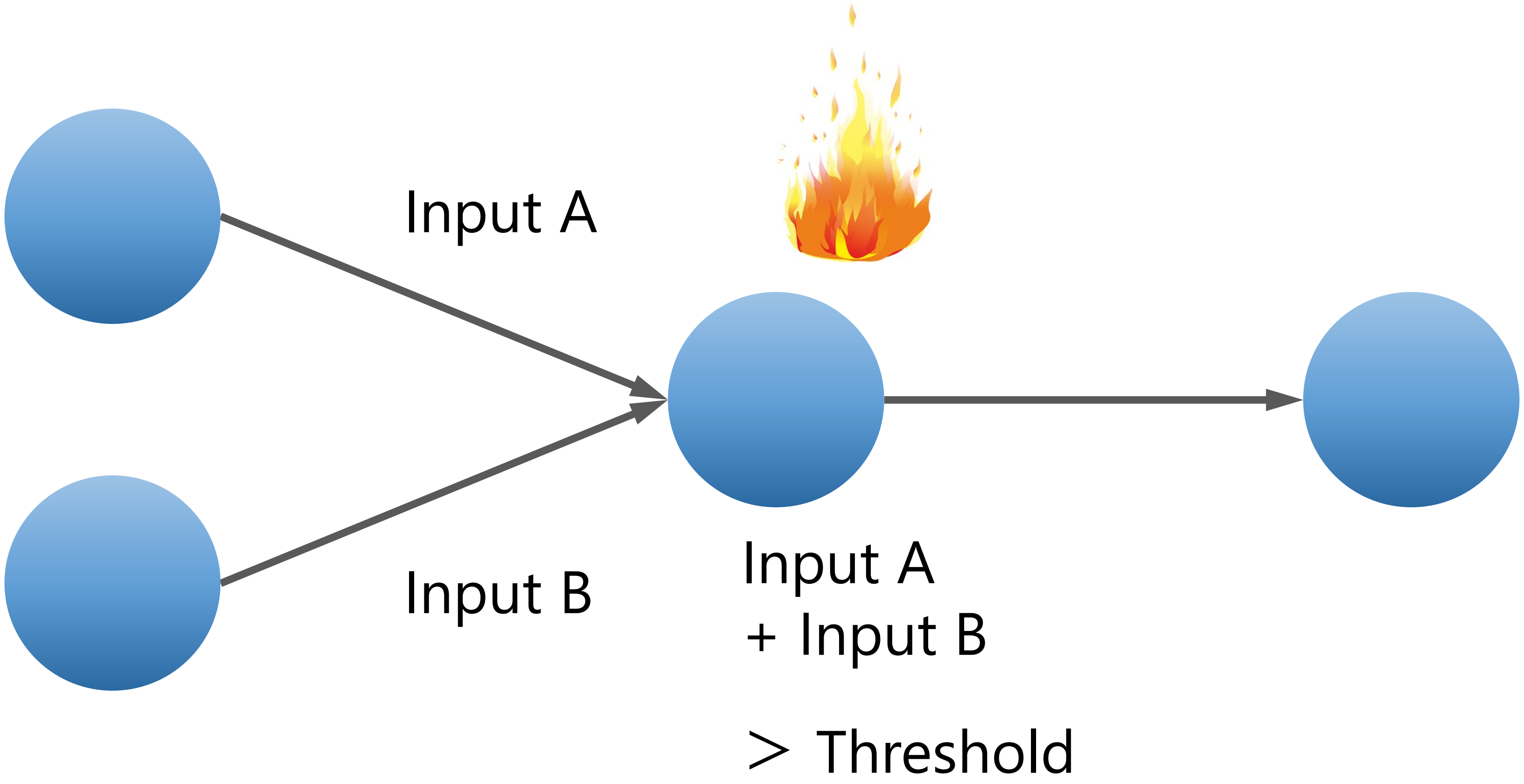
Cell body:
Aggregate the input

Axon:
Send the signal to other neurons

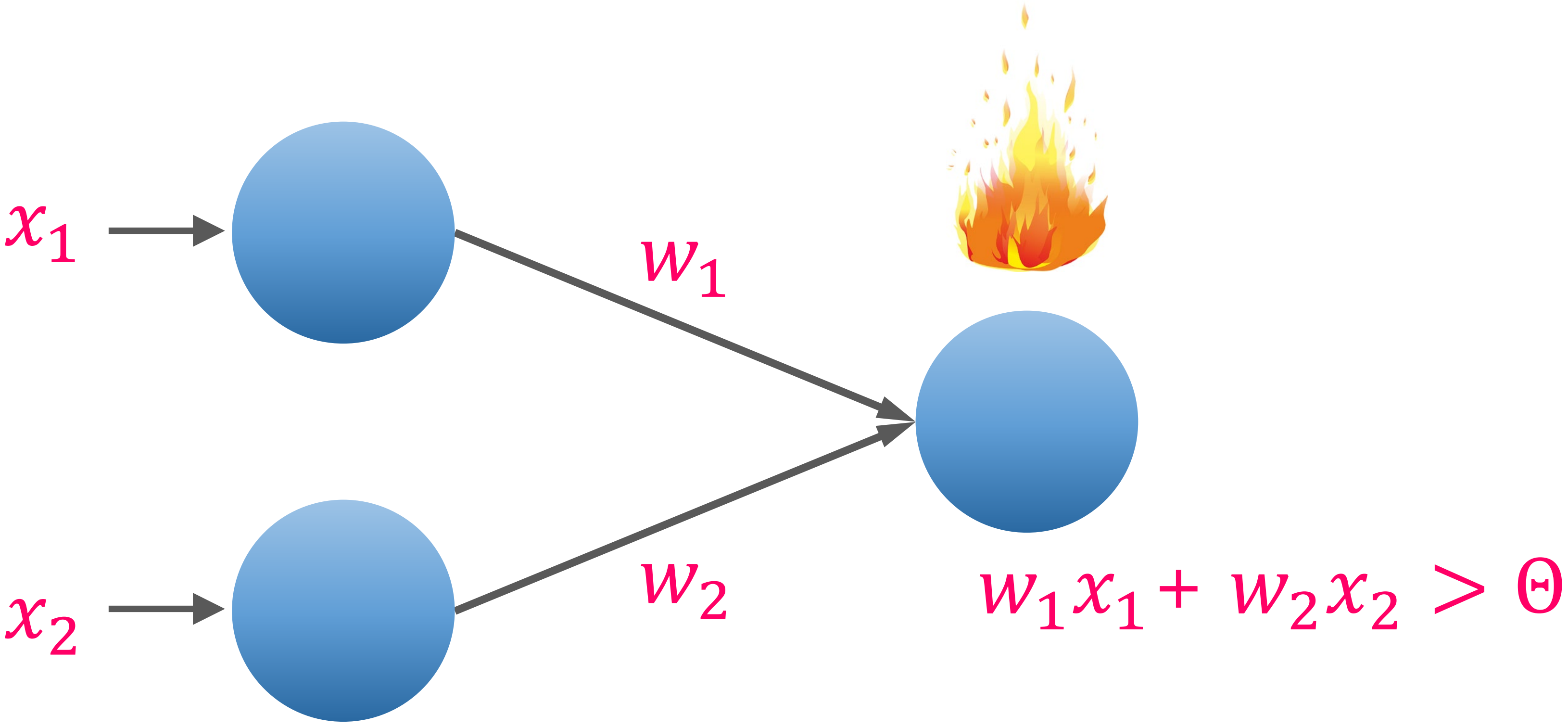
Neural Network



Modeling Neural Network

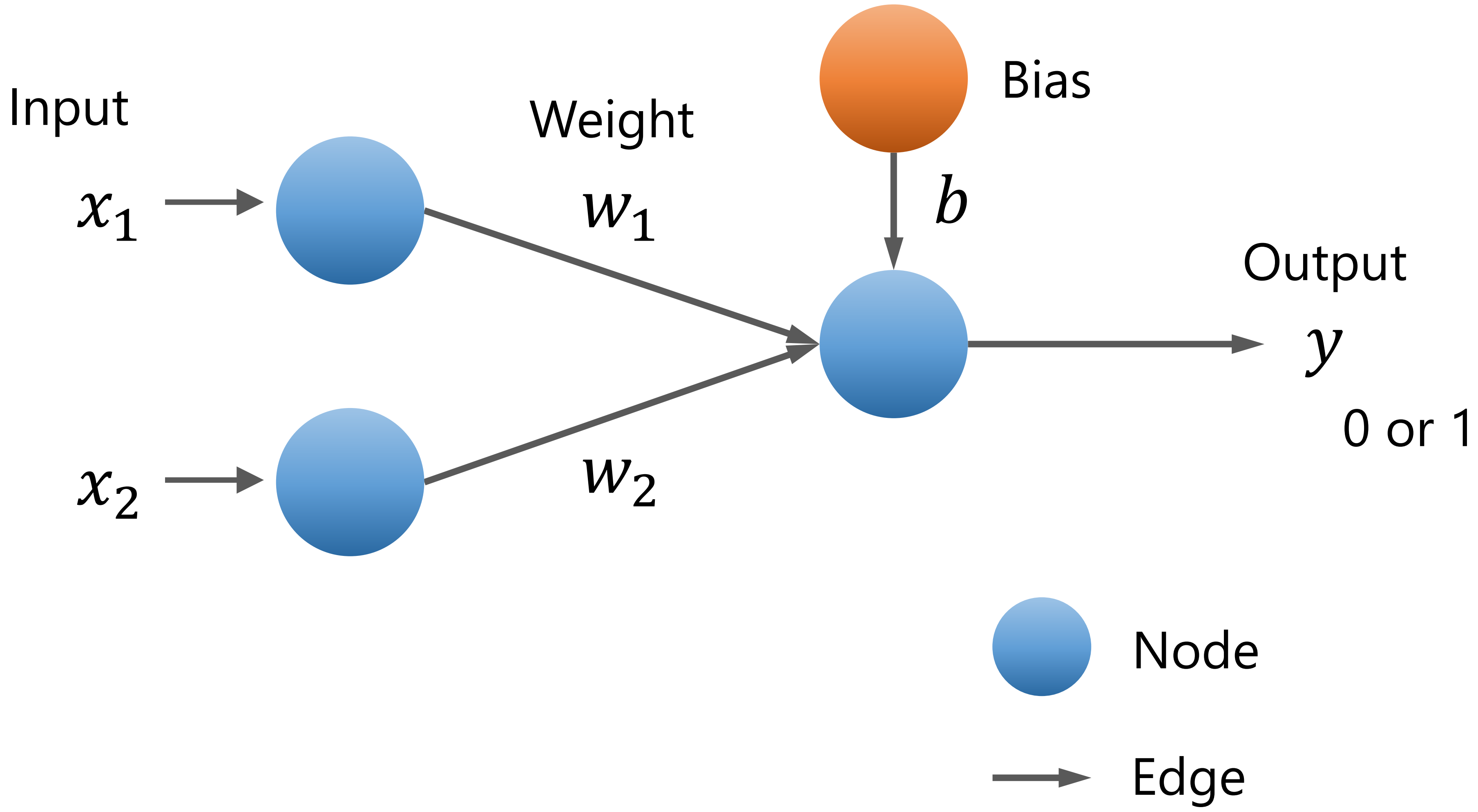


Example

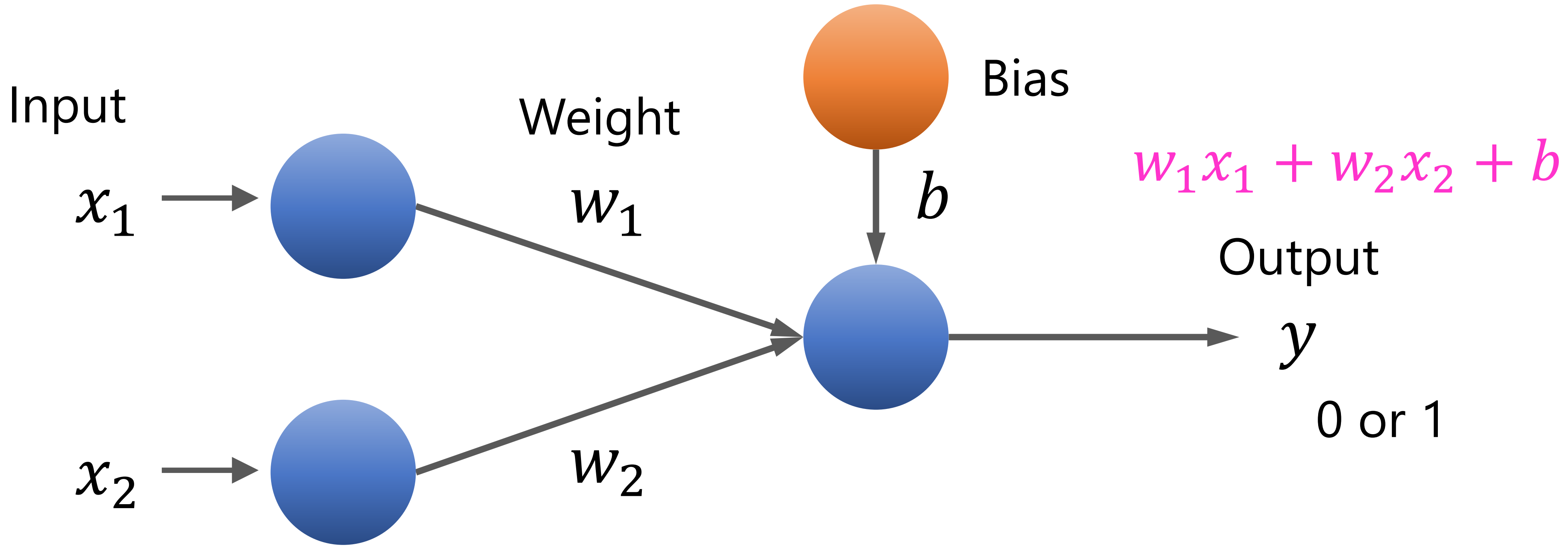


3. Perceptron

What is Perceptron?

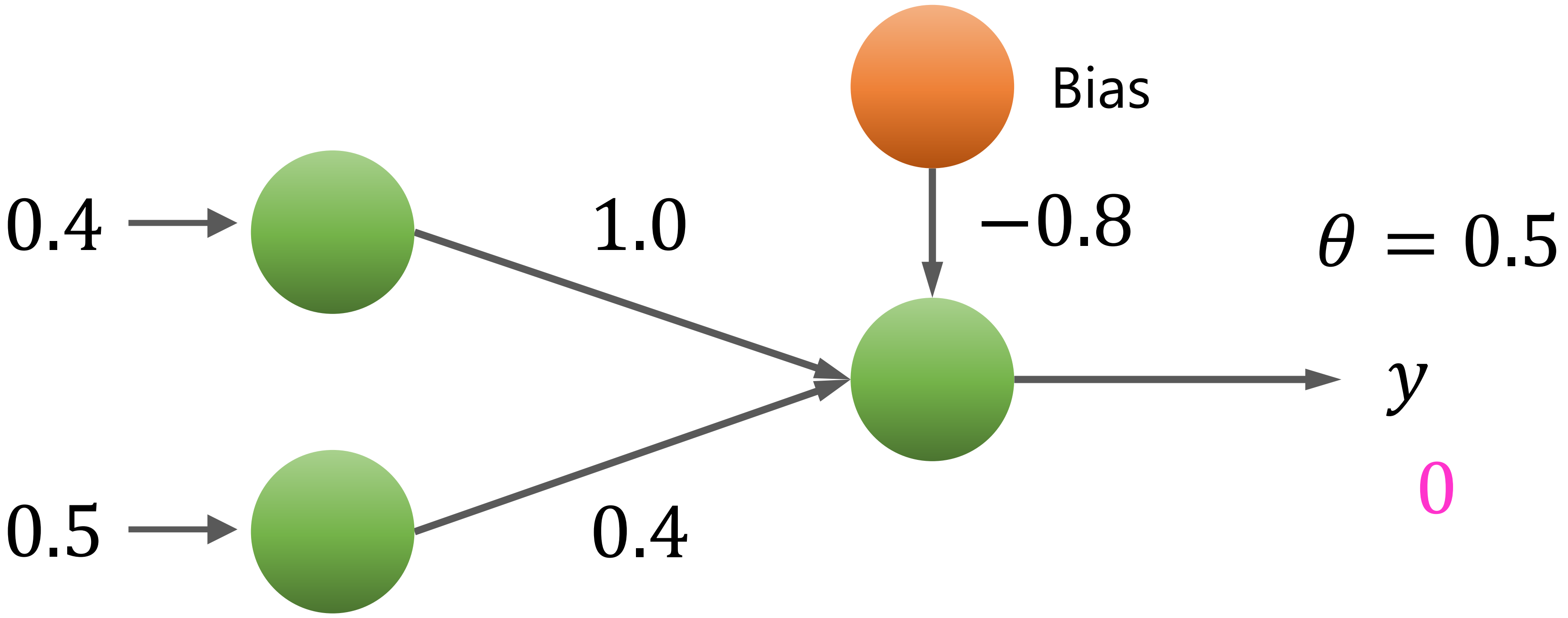


Output of a Perceptron



$$y \begin{cases} 0 & \text{if } w_1x_1 + w_2x_2 + b \leq \theta \\ 1 & \text{if } w_1x_1 + w_2x_2 + b > \theta \end{cases}$$

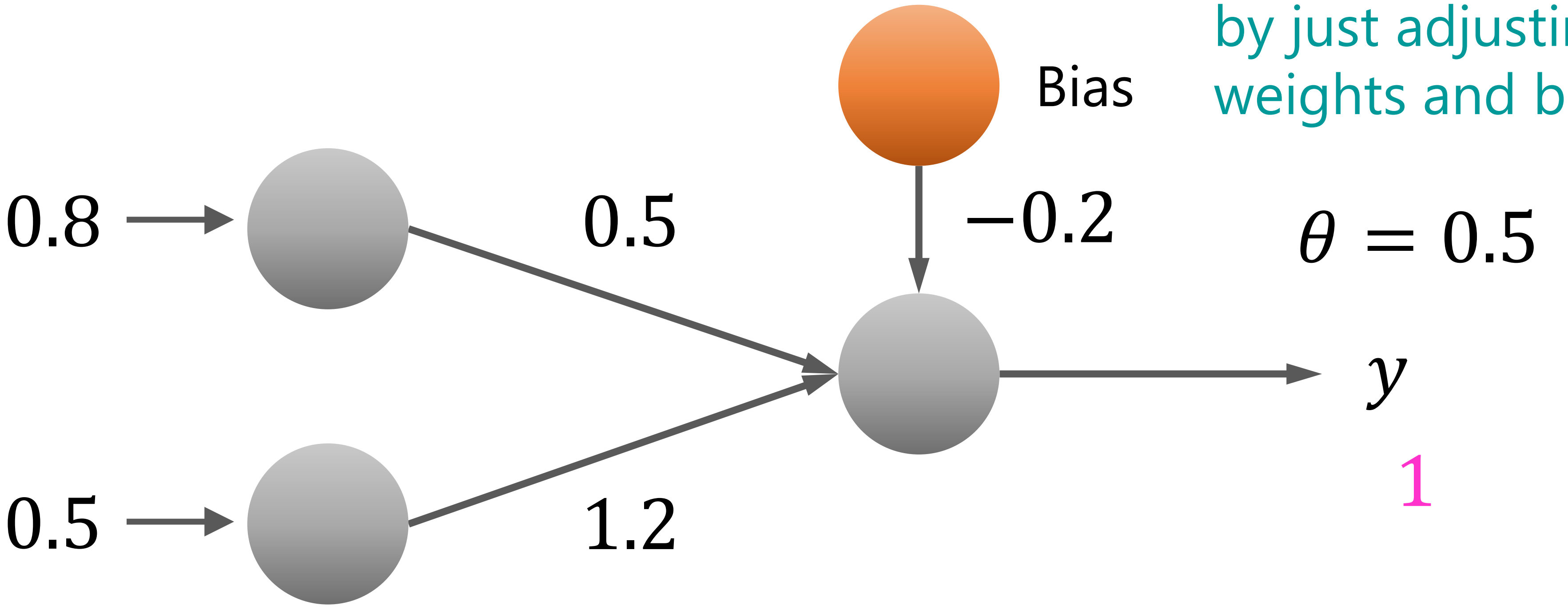
Example



$$0.4 \times 1.0 + 0.5 \times 0.4 - 0.8 = -0.2 < \theta$$

Another Example

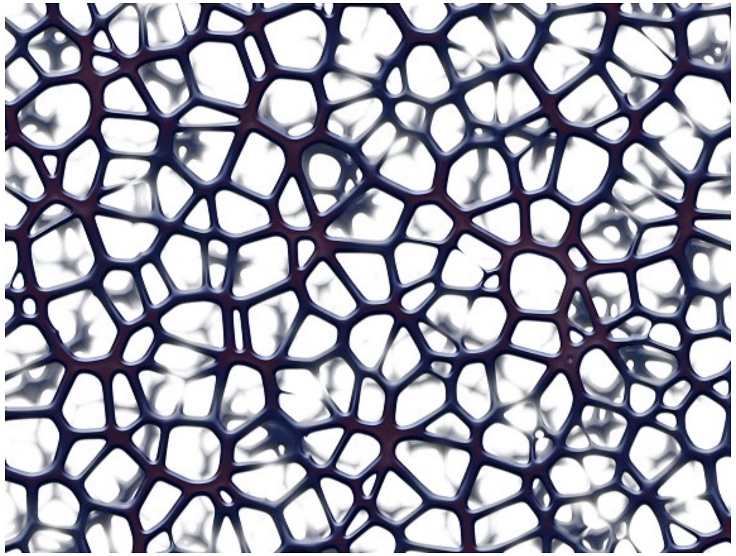
We can change the output of a perceptron by just adjusting the weights and biases.

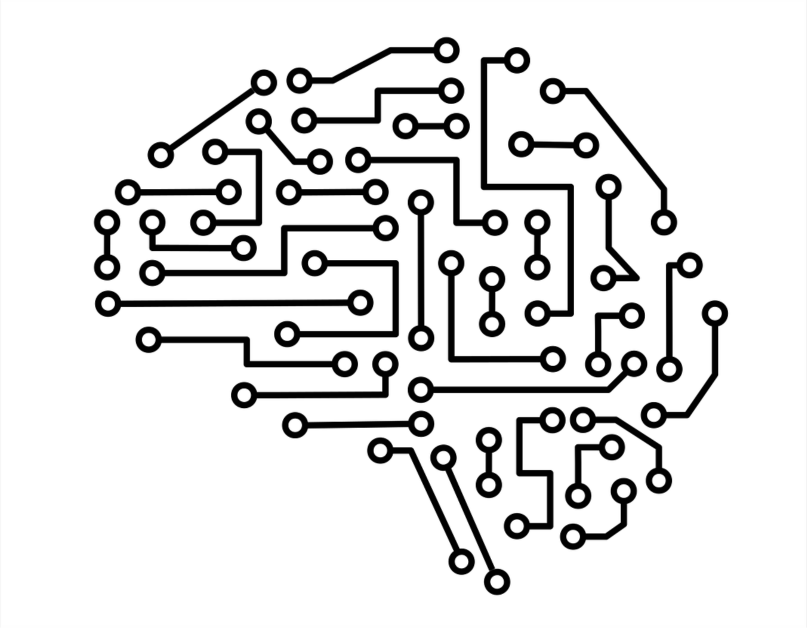


$$0.8 \times 0.5 + 0.5 \times 1.2 - 0.2 = 0.8 > \theta$$

4. Logic Circuit

Logic Gate





1 or 0

- AND
- OR
- NAND

AND Gate

A logic gate that outputs 1 only when both inputs are 1.

Otherwise, it outputs 0.

Truth Table

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

(w_1, w_2, θ)

$(0.6, 0.4, 0.8)$

$(0.2, 0.6, 0.6)$

$(1.2, 0.4, 1.5)$

...

OR Gate

A logic gate that outputs 1 when at least one of the inputs is 1.

Otherwise, it outputs 0.

(w_1, w_2, θ)

$(0.7, 0.7, 0.6)$

$(0.4, 0.3, 0.2)$

$(0.8, 1.2, 0.6)$

...

Truth Table

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

NAND Gate

A logic gate that outputs 1 when at least one of the inputs is 1.

Otherwise, it outputs 0.

(w_1, w_2, θ)

$(-0.6, -0.4, -0.8)$

$(-0.2, -0.6, -0.7)$

$(-1.2, -0.4, -1.5)$

...

Truth Table

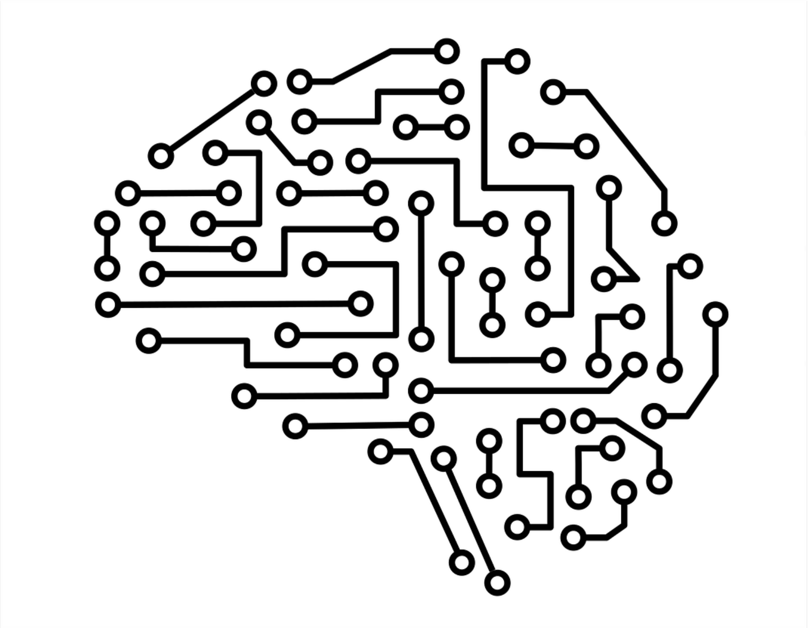
x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0

Perceptron and Logic Gate

Perceptron can express AND gate, OR gate, and NAND gate.

A perceptron can be AND gate, OR gate, and NAND gate depending on its weight and threshold.

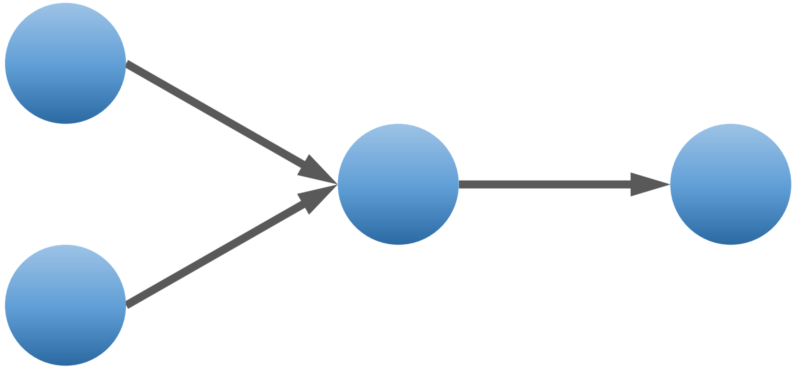
In machine learning, a machine learns appropriate parameter values automatically.



AND

OR

NOT



5. Logic Gate with Python

Logic Gate

AND Gate

A logic gate that outputs 1 only when both inputs are 1. Otherwise, it outputs 0.

OR Gate

A logic gate that outputs 1 when at least one of the inputs is 1. Otherwise, it outputs 0.

NAND Gate

A logic gate that outputs 1 when at least one of the inputs is 1. Otherwise, it outputs 0.

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0

AND Gate

```
# Define AND gate function
def and_gate(x1, x2):
    w1, w2, b, theta = 0.3, 0.3, 0.1, 0.5
    a = x1*w1 + x2*w2 + b
    if a <= theta:
        return 0
    else:
        return 1

# AND gate output
print(and_gate(0, 0))        0
print(and_gate(1, 0))        0
print(and_gate(0, 1))        0
print(and_gate(1, 1))        1
```

AND Gate
A logic gate that outputs 1 only when both inputs are 1. Otherwise, it outputs 0.

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

OR Gate

```
# Define OR gate function
def or_gate(x1, x2):
    w1, w2, b, theta = 0.5, 0.5, 0.1, 0.5
    a = x1*w1 + x2*w2 + b
    if a <= theta:
        return 0
    else:
        return 1

# OR gate output
print(or_gate(0, 0))
print(or_gate(1, 0))
print(or_gate(0, 1))
print(or_gate(1, 1))
```

```
0
1
1
1
```

OR Gate
A logic gate that outputs 1 when at least one of the inputs is 1. Otherwise, it outputs 0.

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

NAND Gate

```
# Define NAND gate function
def nand_gate(x1, x2):
    w1, w2, b, theta = -0.2, -0.2, -0.2, -0.5
    a = x1*w1 + x2*w2 + b
    if a <= theta:
        return 0
    else:
        return 1

# NAND gate output
print(nand_gate(0, 0))          1
print(nand_gate(1, 0))          1
print(nand_gate(0, 1))          1
print(nand_gate(1, 1))          0
```

NAND Gate
A logic gate that outputs 1 when at least one of the inputs is 1. Otherwise, it outputs 0.

x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0

6. Multilayer Perceptron

XOR Gate

Exclusive OR gate

A logic gate that outputs 1 only when either one of x1 or x2 is 1.

Otherwise, it outputs 0.

Truth Table

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

Output of a Perceptron

$$y \begin{cases} 0 & \text{if } w_1x_1 + w_2x_2 + b \leq \theta \\ 1 & \text{if } w_1x_1 + w_2x_2 + b > \theta \end{cases}$$

$$w_1x_1 + w_2x_2 + b = \theta$$

$$\Leftrightarrow$$

$$w_1x_1 + w_2x_2 + b - \theta = 0$$

$$\Leftrightarrow$$

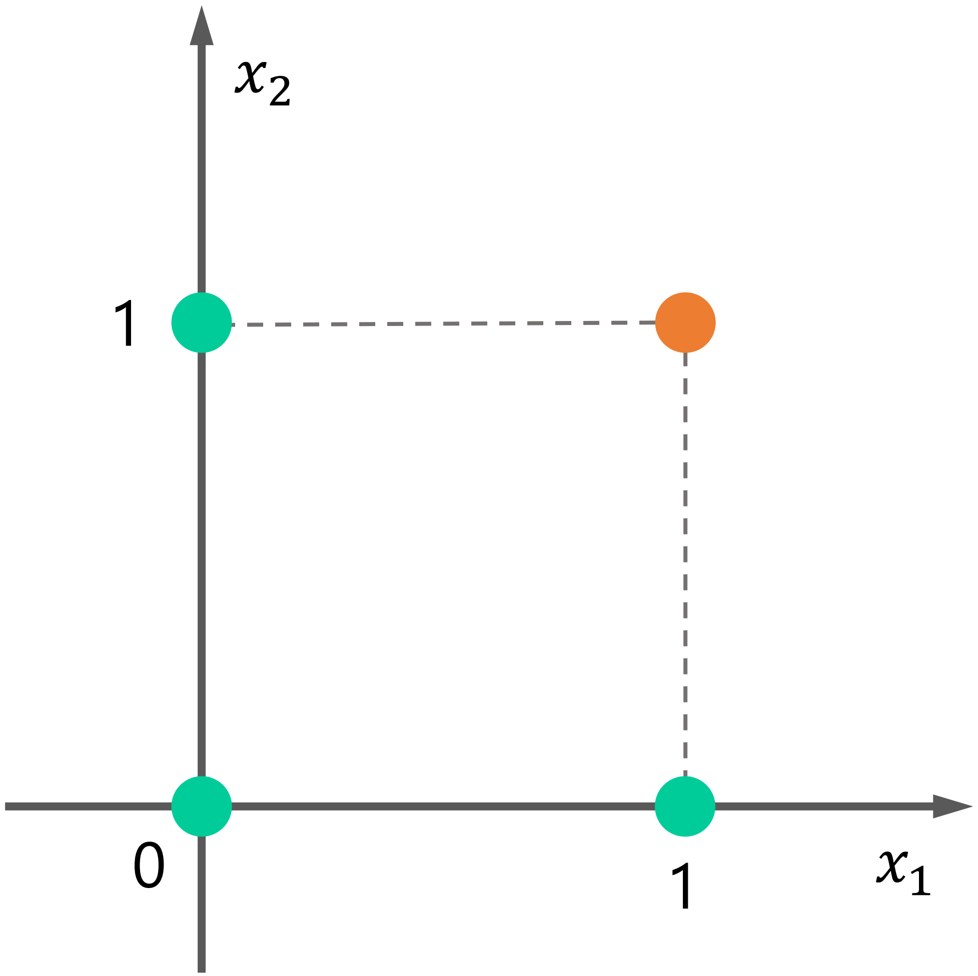
$$w_2x_2 = -w_1x_1 - b + \theta$$

$$\Leftrightarrow$$

$$x_2 = \frac{w_1}{w_2}x_1 + \frac{-b + \theta}{w_2}$$

$$x_2 = \beta_1 x_1 + \beta_2$$

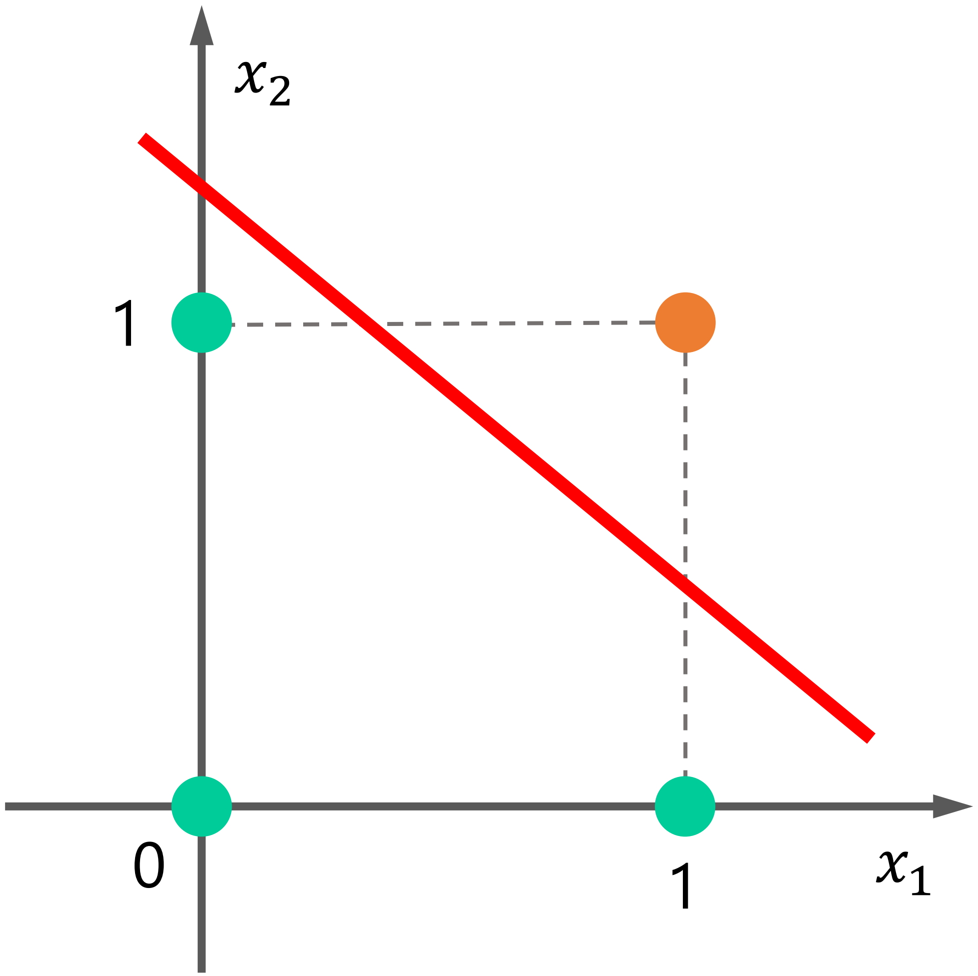
AND Gate and Perceptron



Truth Table

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

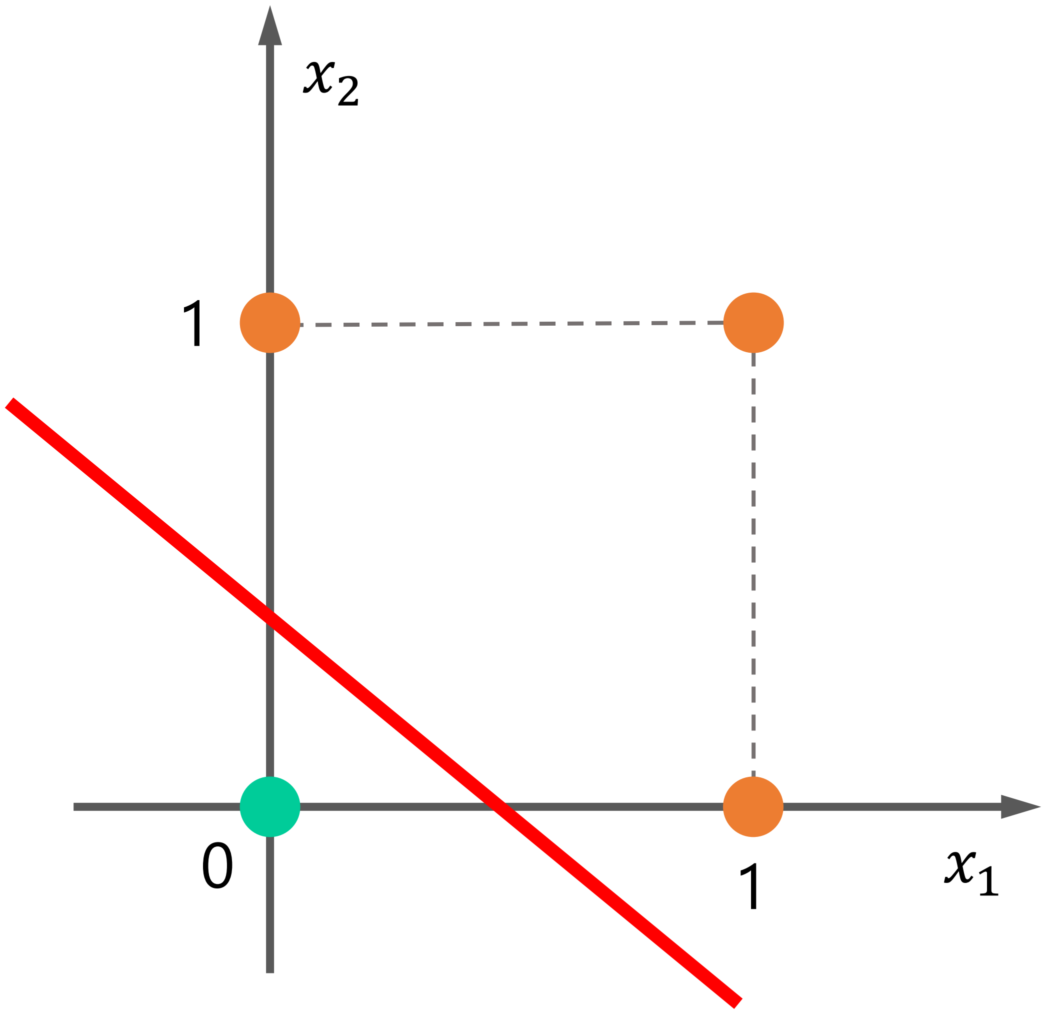
AND Gate and Perceptron



Truth Table

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

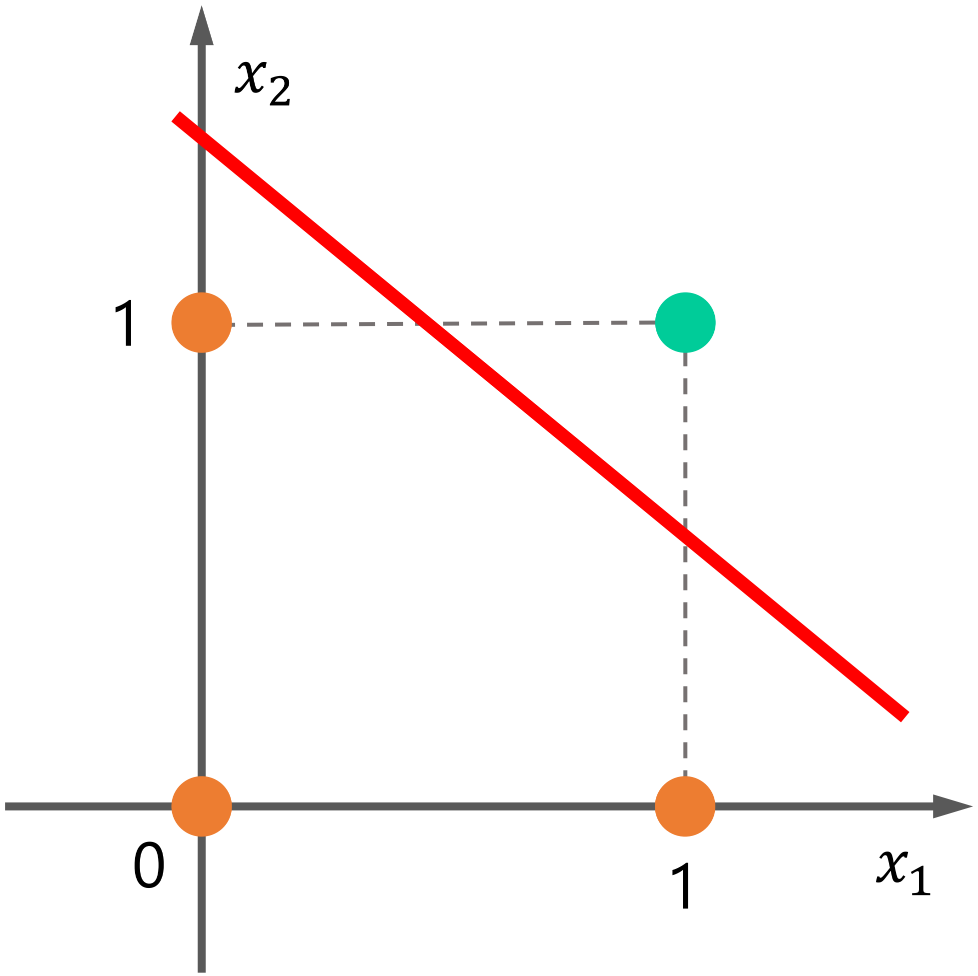
OR Gate and Perceptron



Truth Table

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

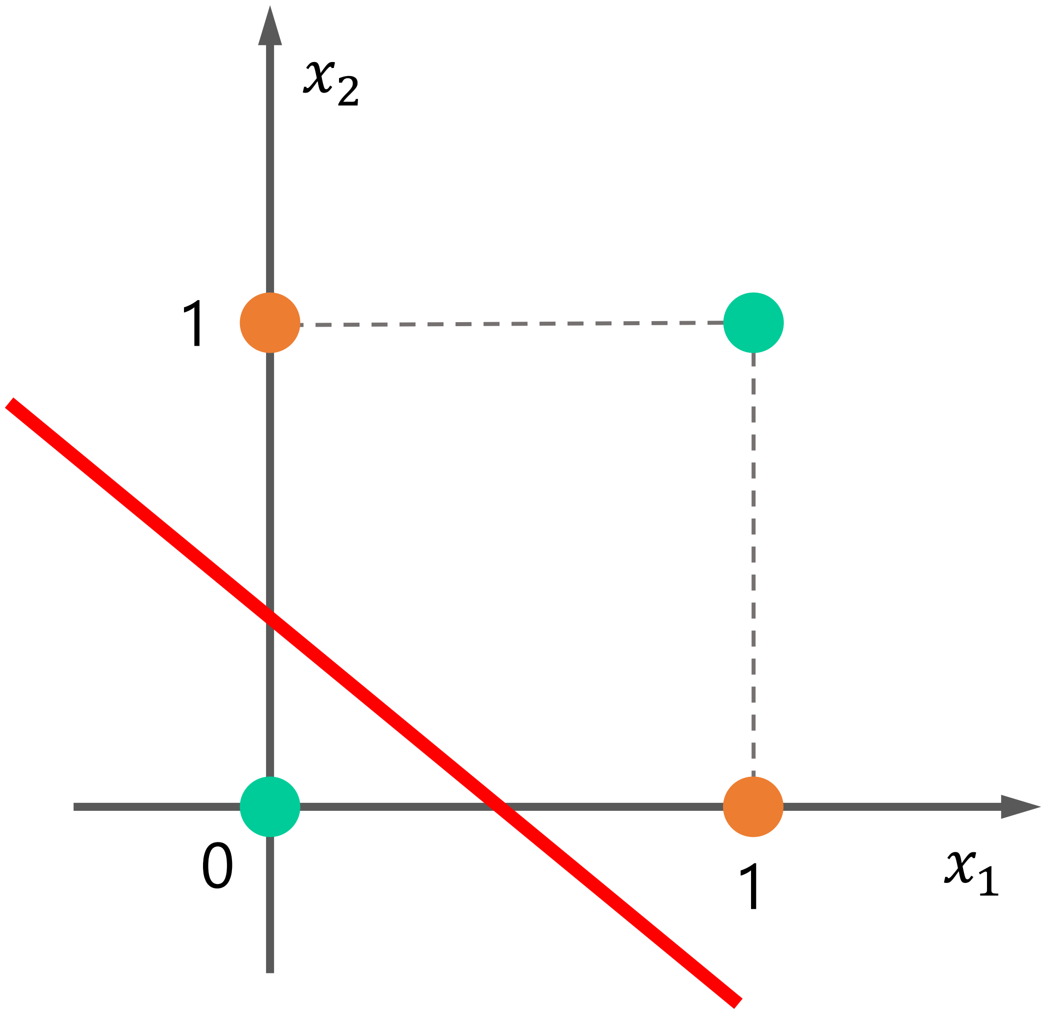
NAND Gate



Truth Table

x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0

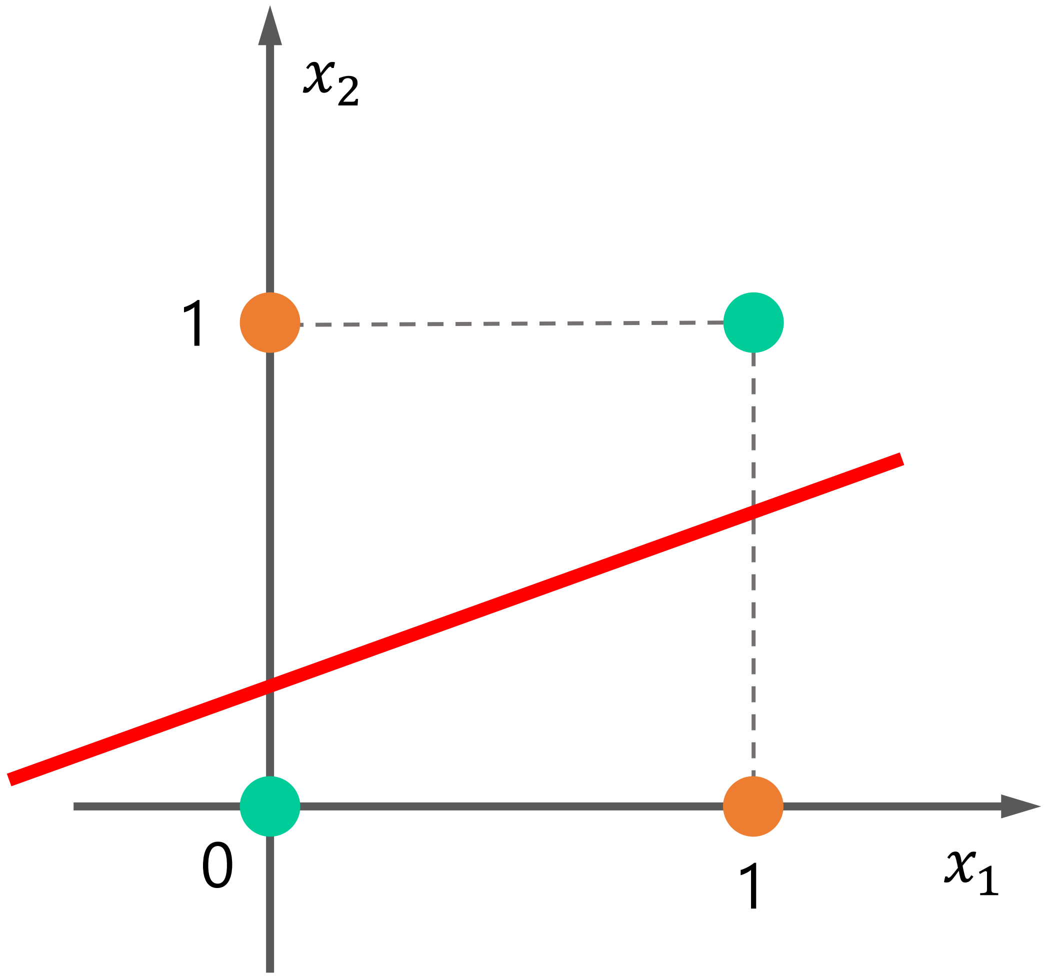
XOR Gate and Perceptron



Truth Table

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

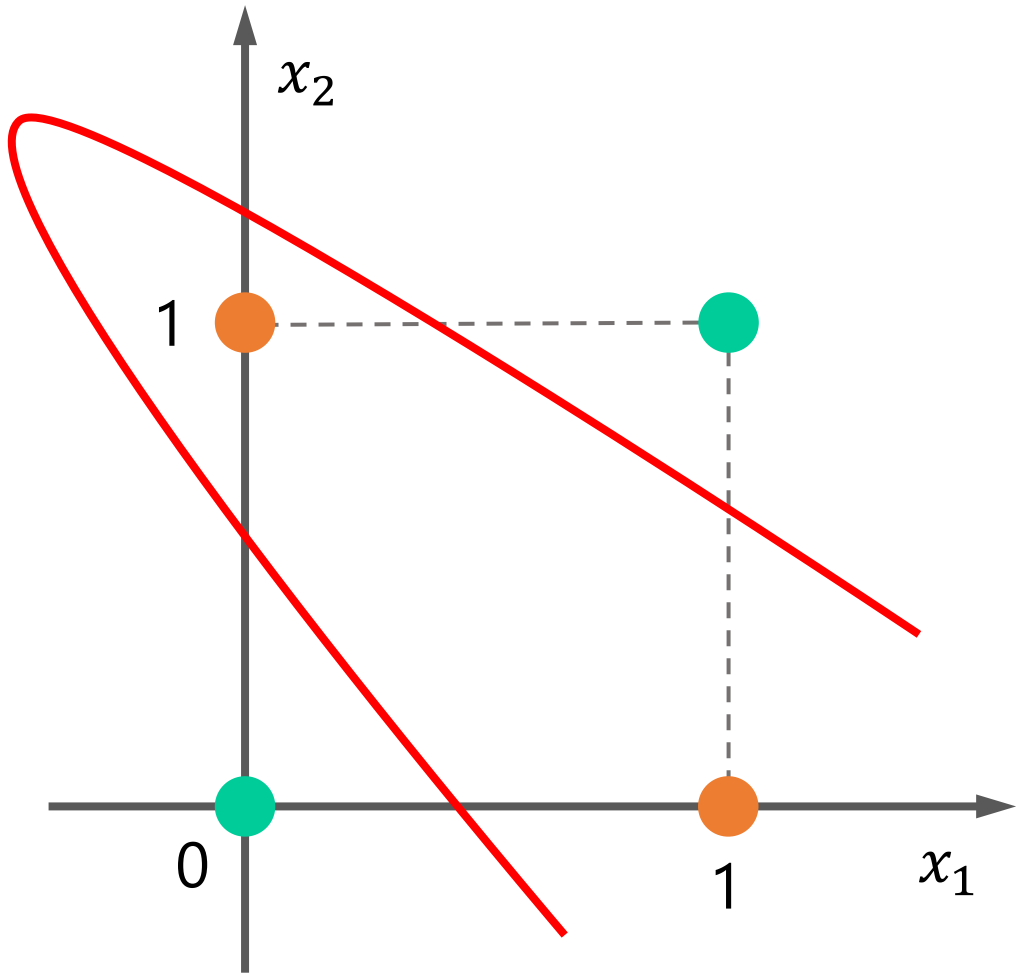
XOR Gate and Perceptron



Truth Table

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

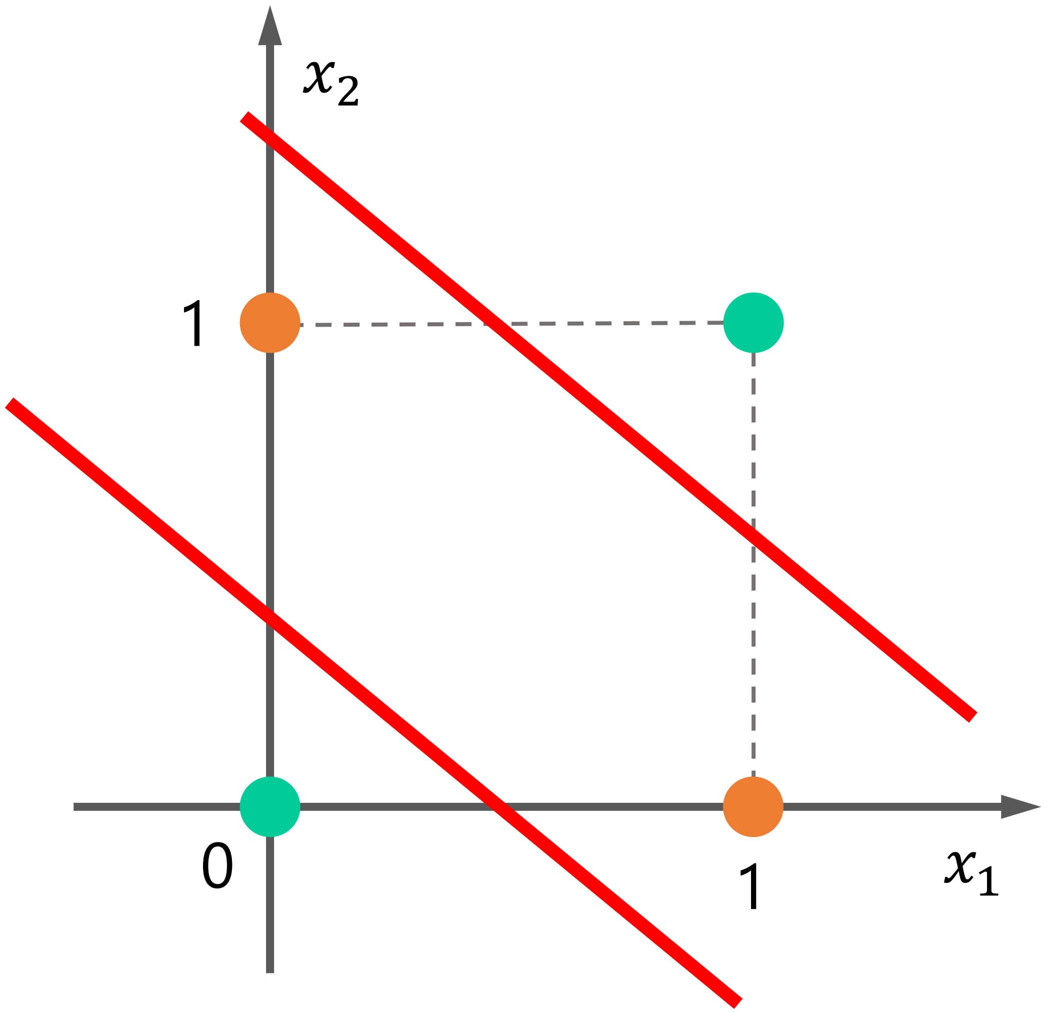
XOR Gate and Perceptron



Truth Table

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

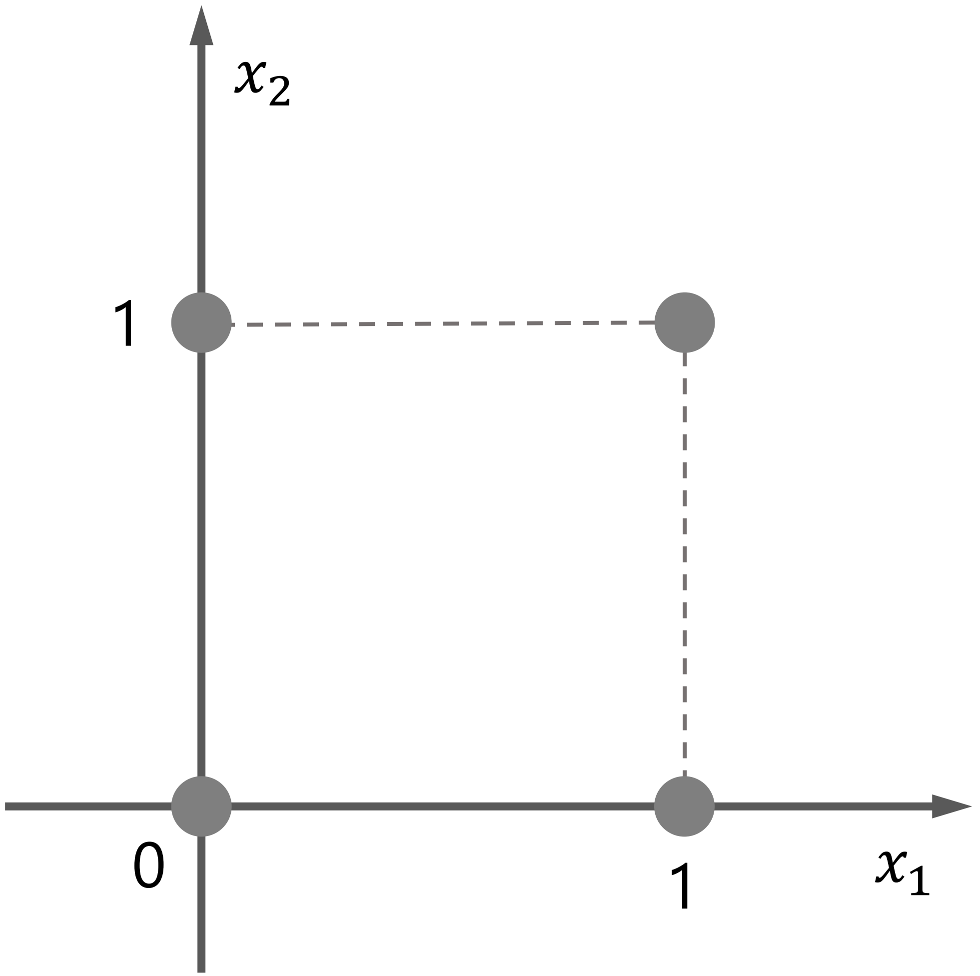
XOR Gate and Perceptron



Truth Table

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

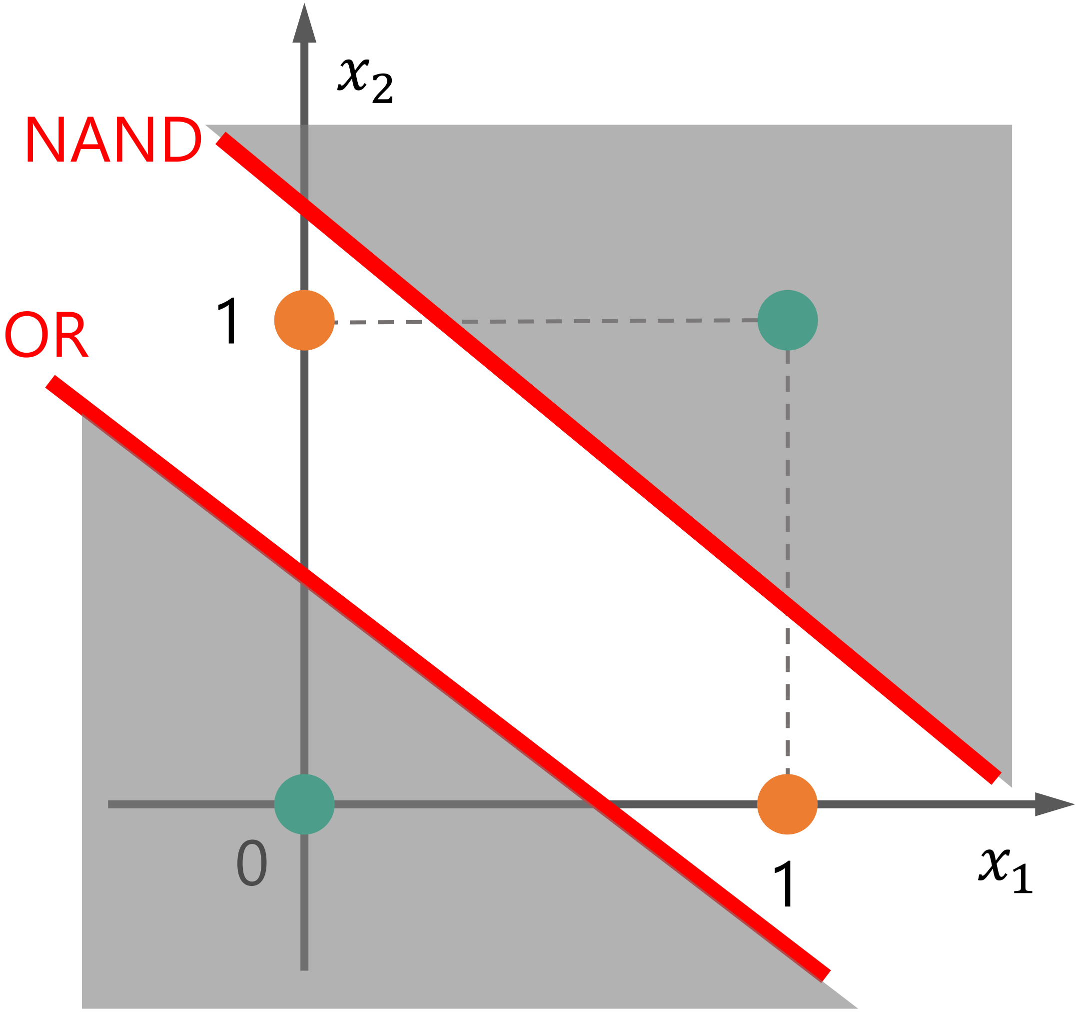
XOR Gate and Multiple Perceptrons



Truth Table

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

XOR Gate and Multiple Perceptrons



Truth Table

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

Truth Tables

NAND

x_1	x_2	s_1
0	0	1
1	0	1
0	1	1
1	1	0

OR

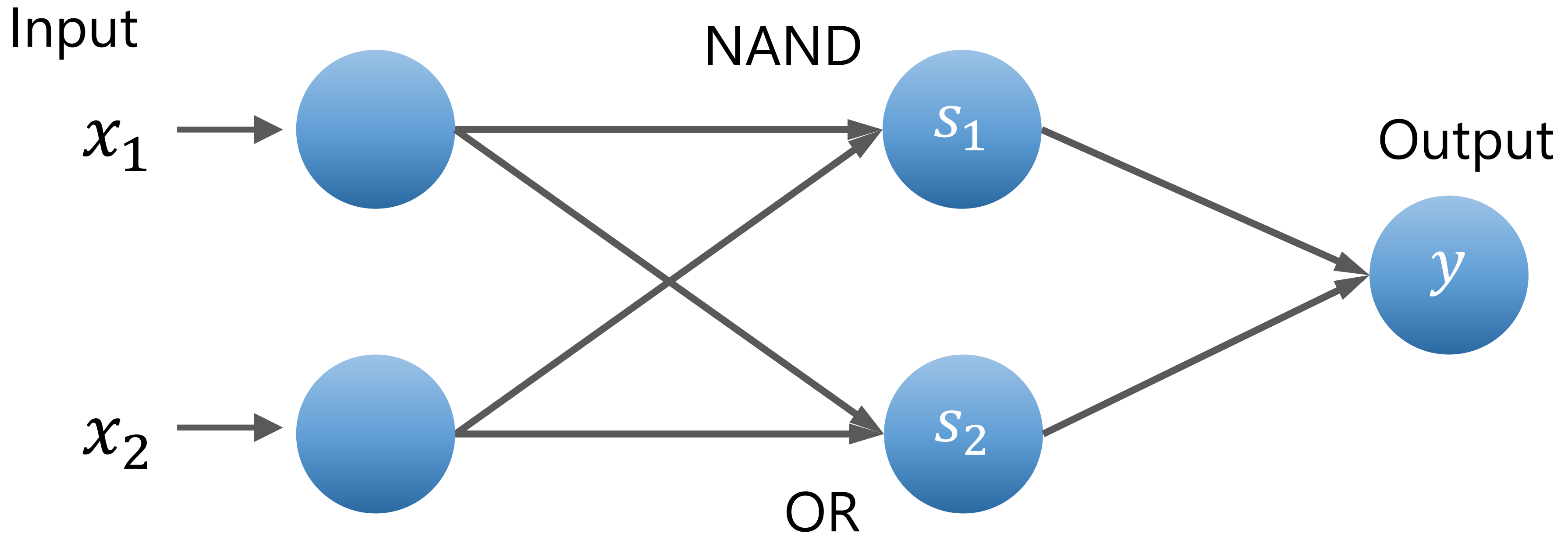
x_1	x_2	s_2
0	0	0
1	0	1
0	1	1
1	1	1

AND Gate

s_1	s_2	y
1	0	0
1	1	1
1	1	1
0	1	0

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

Multilayer Perceptron



7. Multilayer Perceptron with Python

XOR Gate

```python
# Define XOR gate function
def xor_gate(x1, x2):
    s1 = nand_gate(x1, x2)
    s2 = or_gate(x1, x2)
    y = and_gate(s1, s2)
    return y
```

```python
# XOR gate output
print(xor_gate(0, 0))
print(xor_gate(1, 0))
print(xor_gate(0, 1))
print(xor_gate(1, 1))
-------------------------
0
1
1
0
```

Truth Table

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

Multilayer Perceptron for Regression

```
# Import libraries
import pandas as pd
import numpy as np
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Load dataset
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="¥s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]
```

Multilayer Perceptron for Regression (Continued)

```
# Create X and y
X = pd.DataFrame(data)
y = target

# Split data into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Standardize the features
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Multilayer Perceptron for Regression (Continued)

```
# Initiate and fit the MLP regression model
mlp_rg = MLPRegressor()
mlp_rg.fit(X_train, y_train)

# Make Prediction by MLP
y_pred_train = mlp_rg.predict(X_train)
y_pred_test = mlp_rg.predict(X_test)
```

Multilayer Perceptron for Regression (Continued)

```
# Print MLP Performance
print("Multilayer Perceptron Performance")
print("------------------------------------")
print("Train R2 :  ", f'{mlp_rg.score(X_train, y_train):.3f}')
print("Test  R2 :  ", f'{mlp_rg.score(X_test, y_test):.3f}')
print("------------------------------------")
print("Train MSE: ", f'{mean_squared_error(y_train, y_pred_train):.3f}')
print("Test  MSE: ", f'{mean_squared_error(y_test, y_pred_test):.3f}')
```

```
Multilayer Perceptron Performance
------------------------------------
Train R2 :   0.686
Test  R2 :   0.710
------------------------------------
Train MSE:  26.002
Test  MSE:  25.982
```

Multilayer Perceptron for Classification

```
# Import libraries
from sklearn.neural_network import MLPClassifier

# Load the dataset
from sklearn.datasets import load_breast_cancer
breast_cancer = load_breast_cancer()

# Create X and y
X = pd.DataFrame(breast_cancer.data)
y = breast_cancer.target

# Split the data into training and test set
X_train, X_test, y_train, y_test = train_test_split(breast_cancer.data,
                                                    breast_cancer.target)
```

Multilayer Perceptron for Classification (Continued)

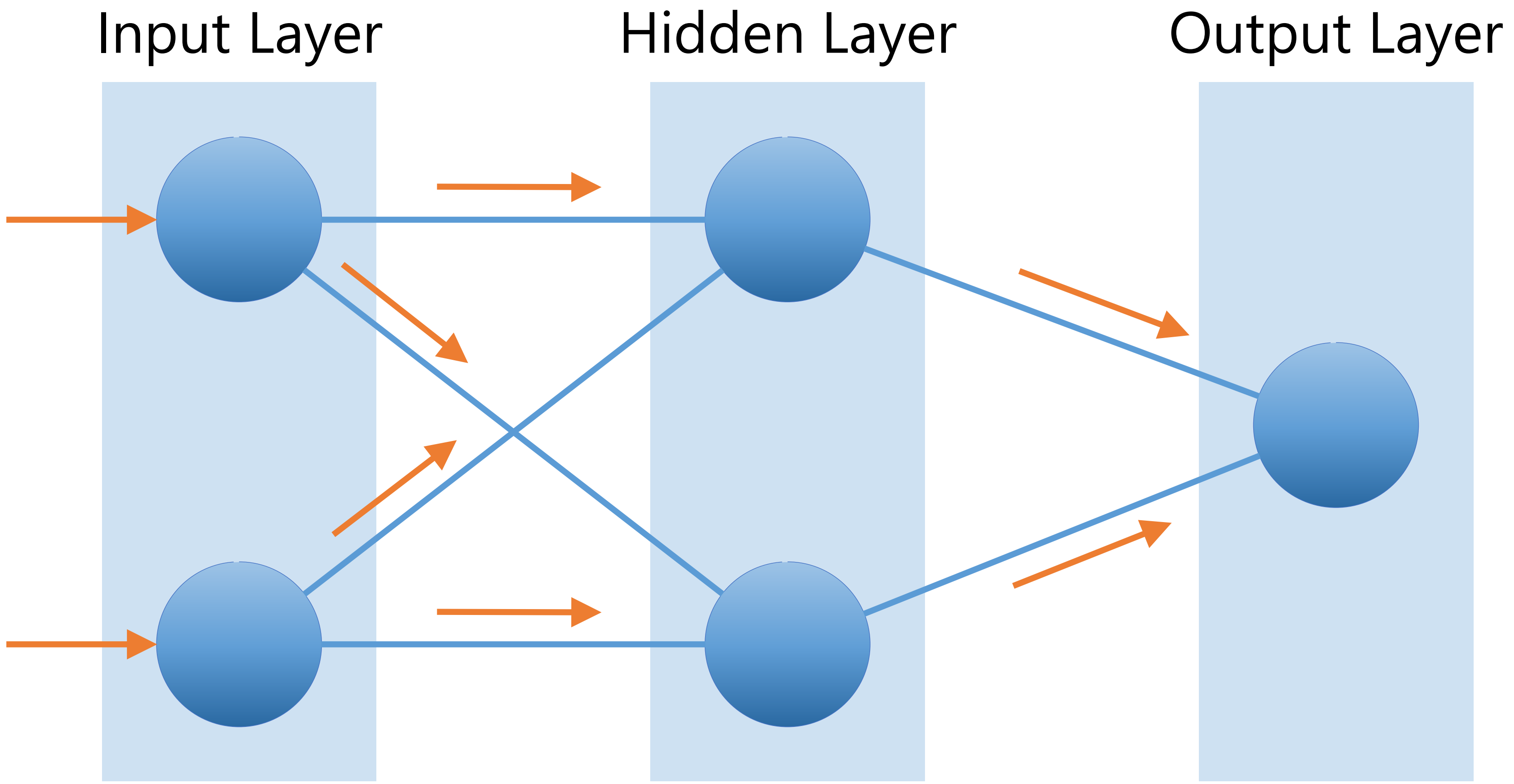
```
# Standardize the features
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Initiate and fit the model
mlp_clf = MLPClassifier()
mlp_clf.fit(X_train, y_train)
```

Multilayer Perceptron for Classification (Continued)

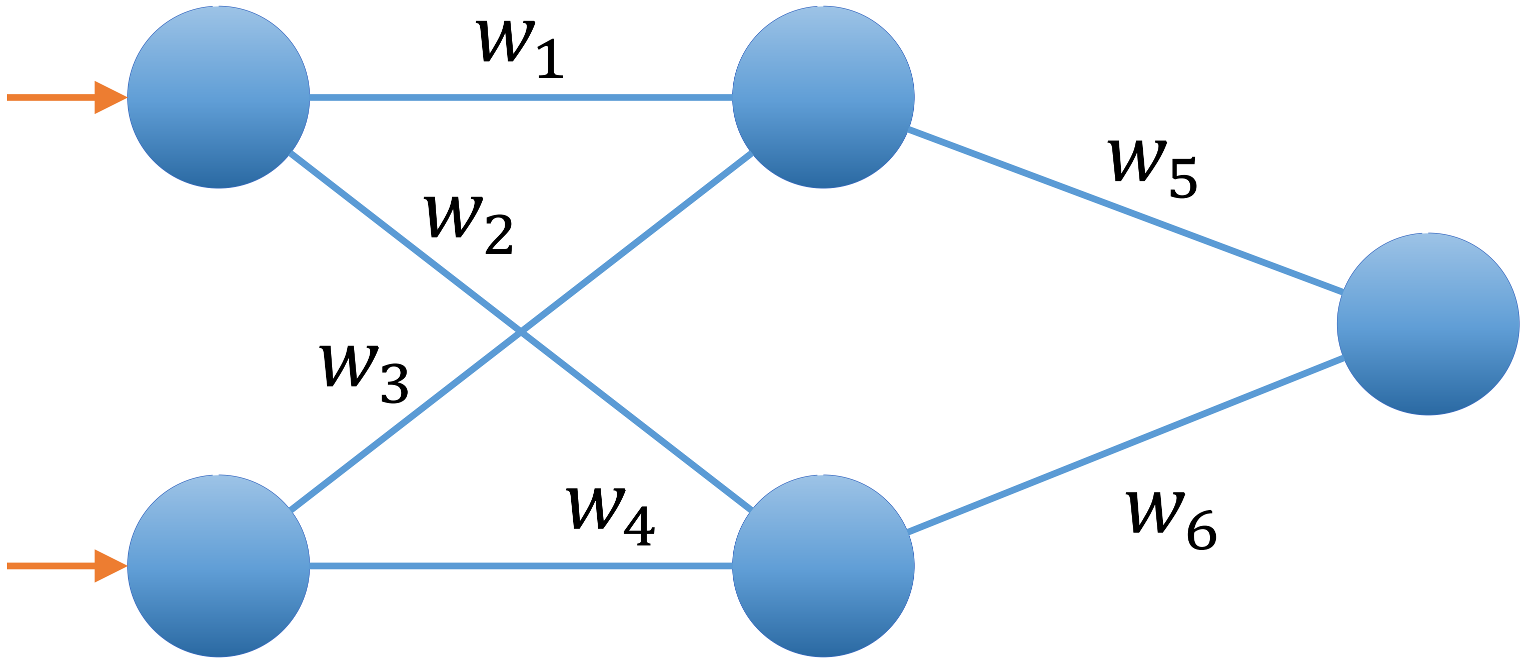
```
# Show accuracy score
print("Training Accuracy: ", mlp_clf.score(X_train, y_train))
print("Test Accuracy     : ", mlp_clf.score(X_test, y_test))
----------------------------------------------------------------
Training Accuracy:  0.9929577464788732
Test Accuracy     :  0.986013986013986
```

8. Neural Network

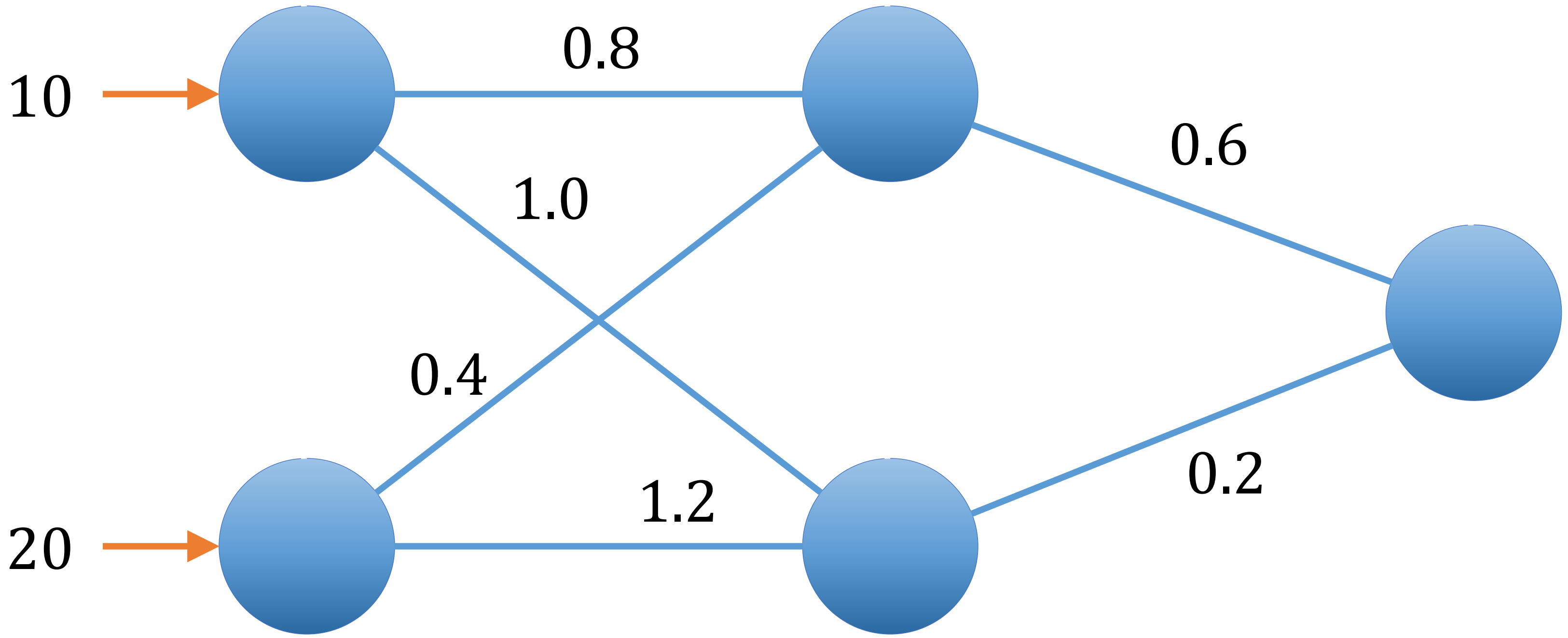
Artificial Neural Network



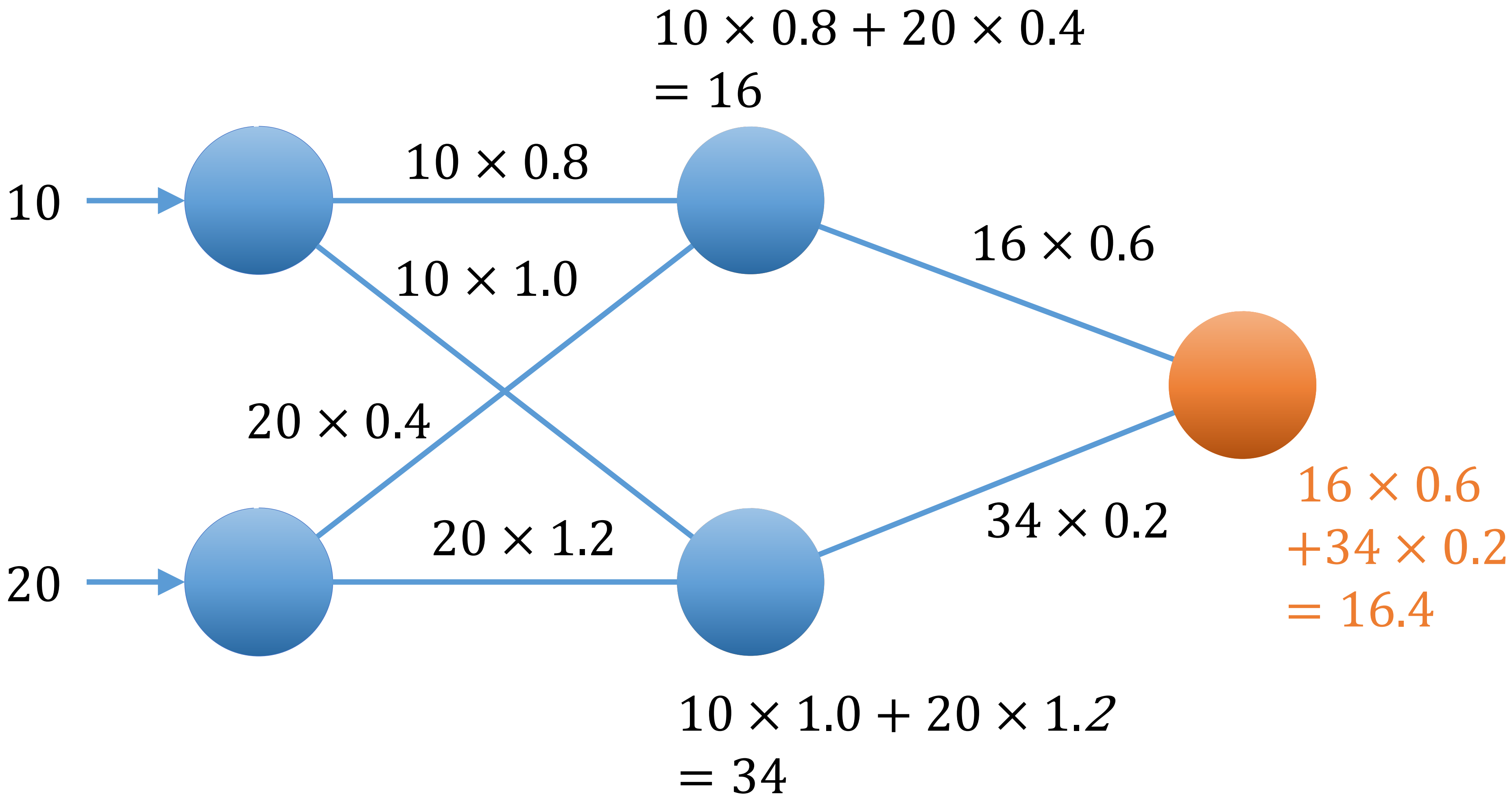
Artificial Neural Network



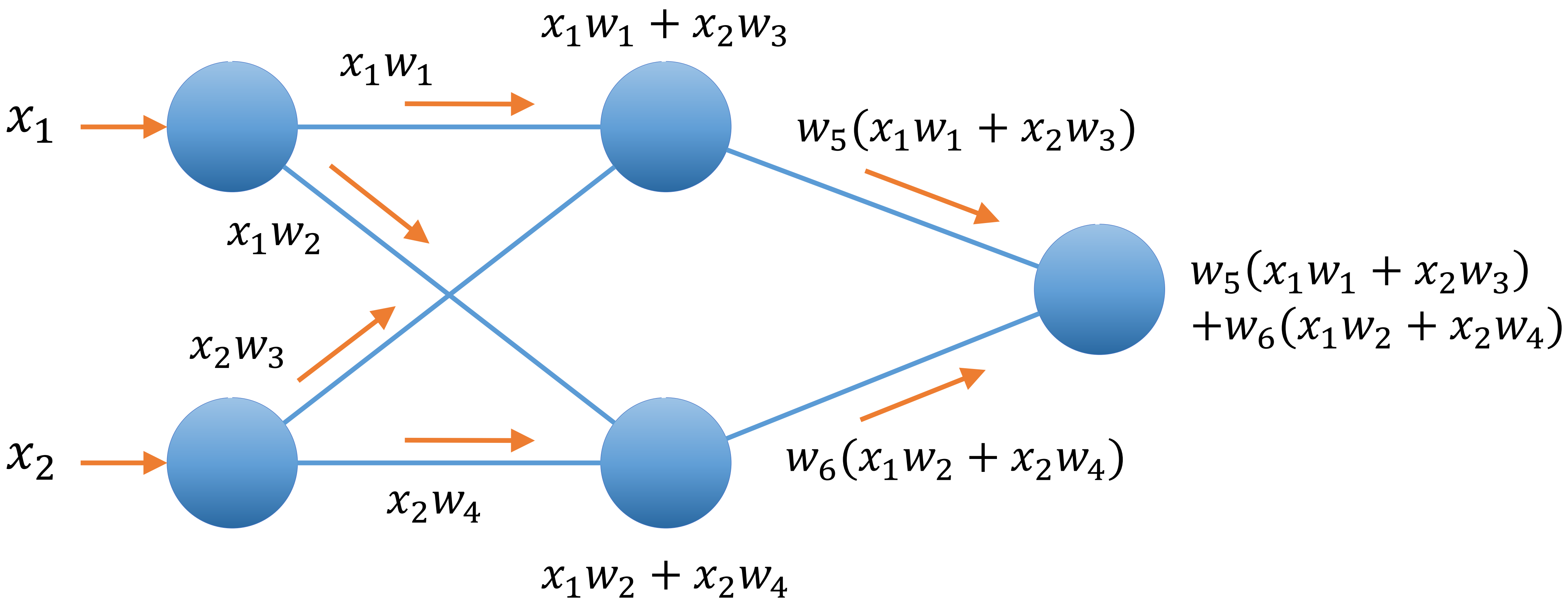
Artificial Neural Network



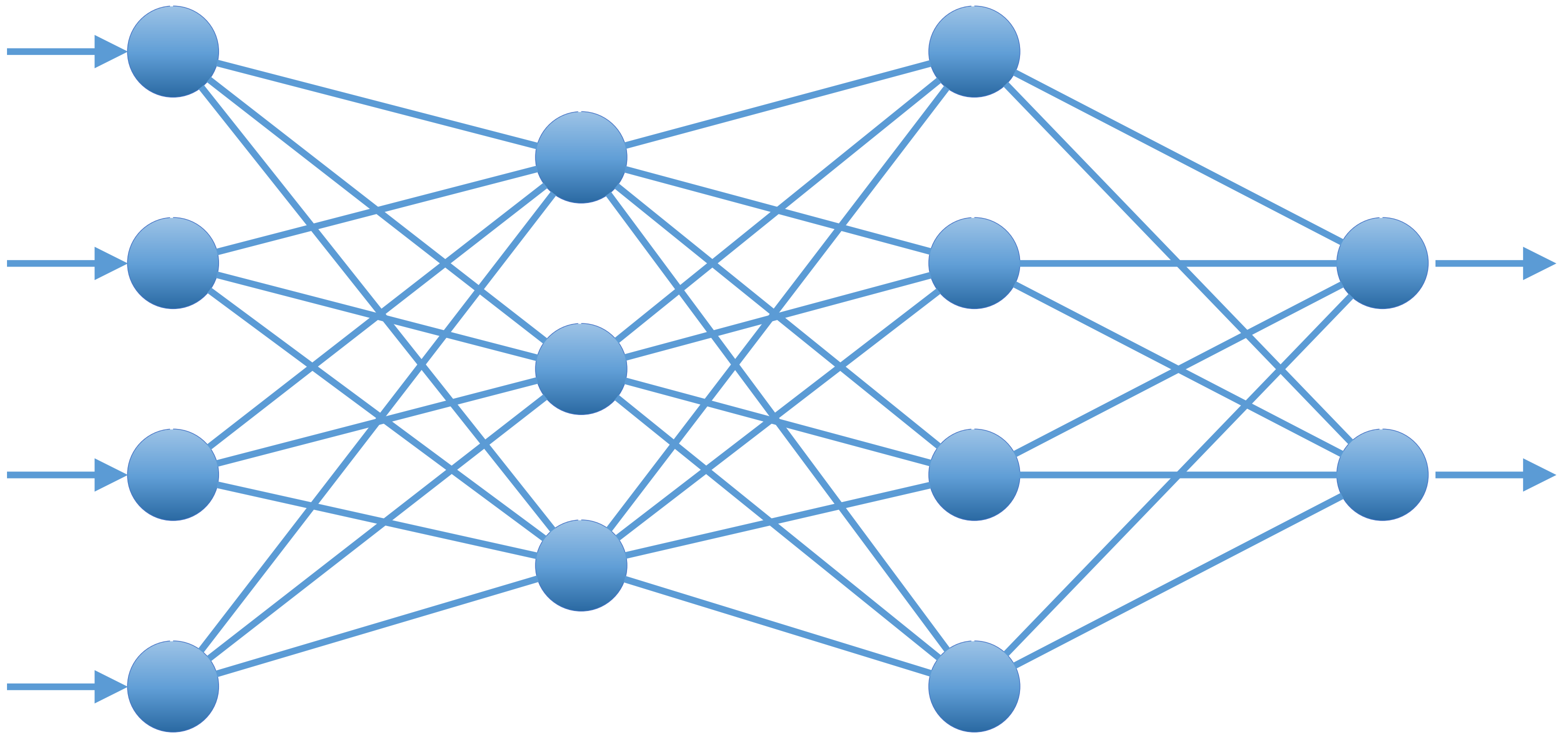
Artificial Neural Network



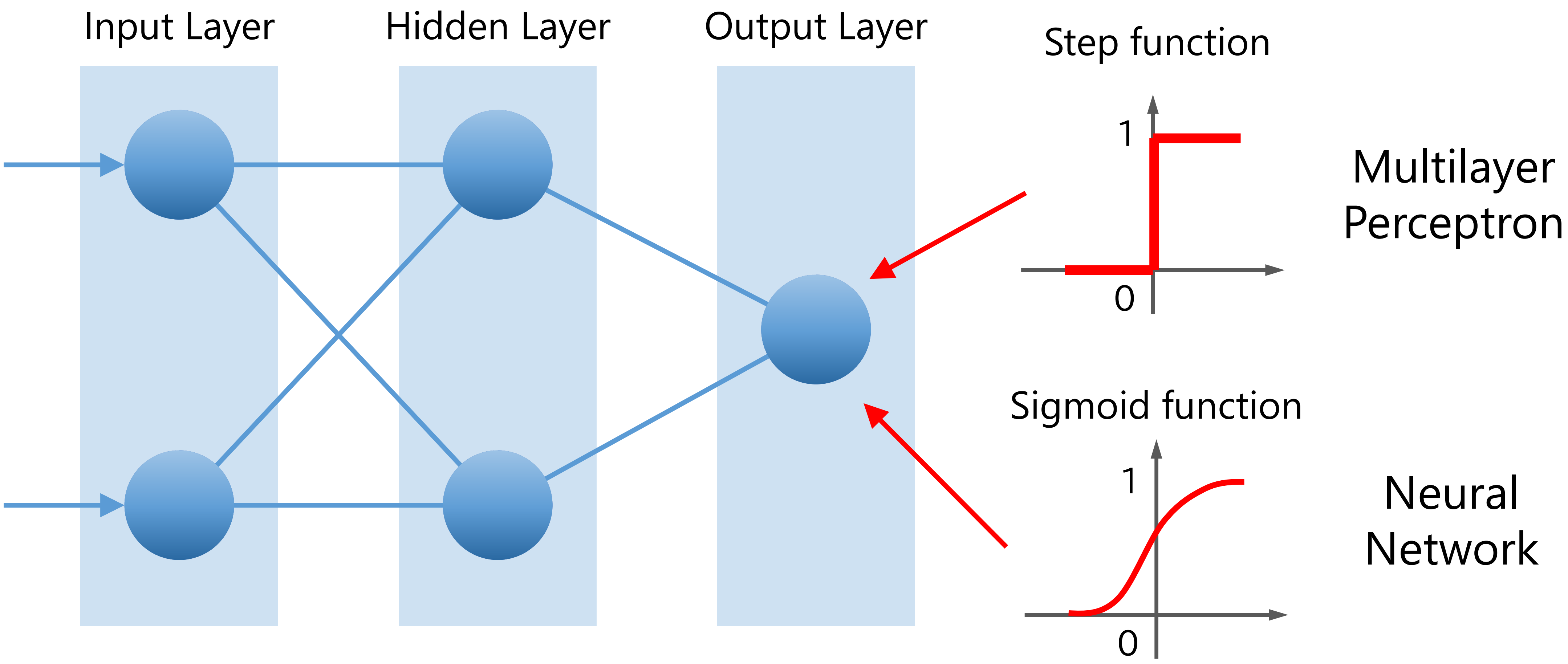
Forward Propagation



Deep Neural Network

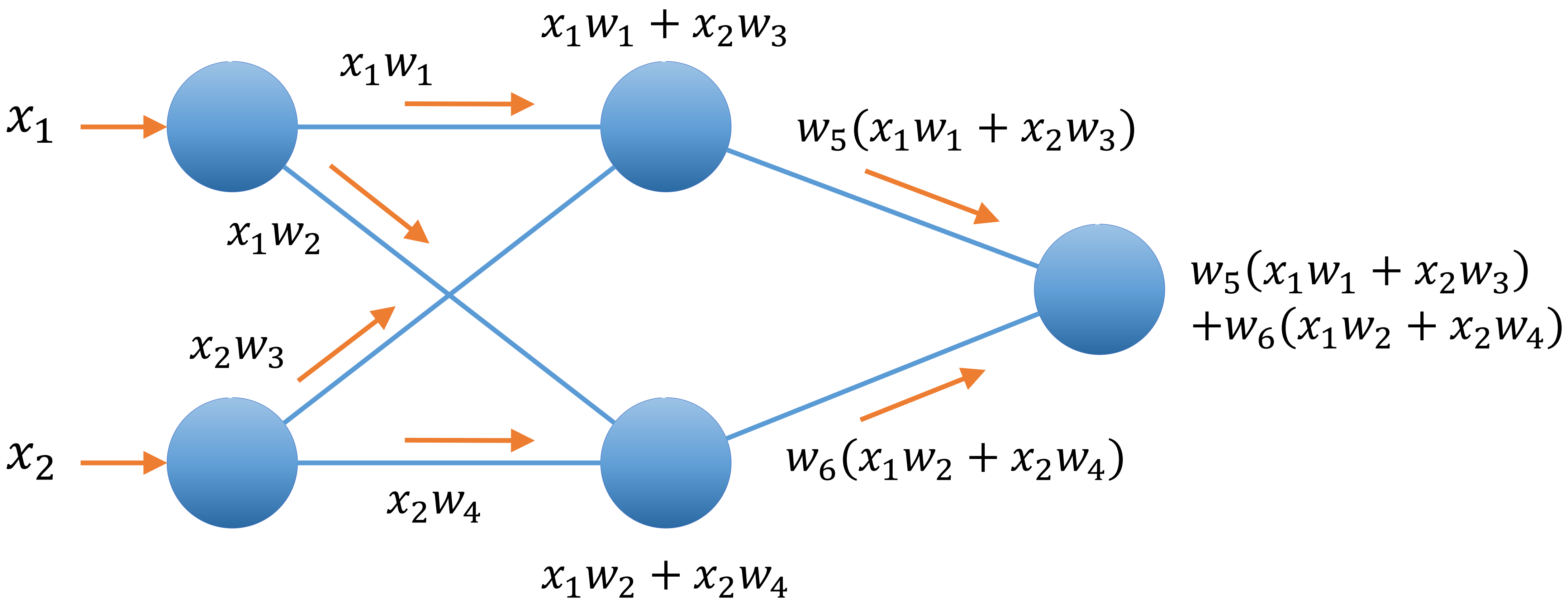


Difference between Multilayer Perceptron and Neural Network



9. Activation Function

Recap: Forward Propagation



Forward Propagation

$$x_1 w_1 + x_2 w_3$$

$$x_1 w_2 + x_2 w_4$$

$$\Leftrightarrow$$

$$x_2 = \frac{w_1}{w_3} x_1$$

$$x_2 = \frac{w_2}{w_4} x_1$$

Forward Propagation

$$w_5(x_1w_1 + x_2w_3) + w_6(x_1w_2 + x_2w_4)$$

$$\Leftrightarrow$$

$$w_5x_1w_1 + w_5x_2w_3 + w_6x_1w_2 + w_6x_2w_4$$

$$\Leftrightarrow$$

$$w_1w_5x_1 + w_3w_5x_2 + w_2w_6x_1 + w_4w_6x_2$$

$$\Leftrightarrow$$

$$w_1w_5x_1 + w_2w_6x_1 + w_3w_5x_2 + w_4w_6x_2$$

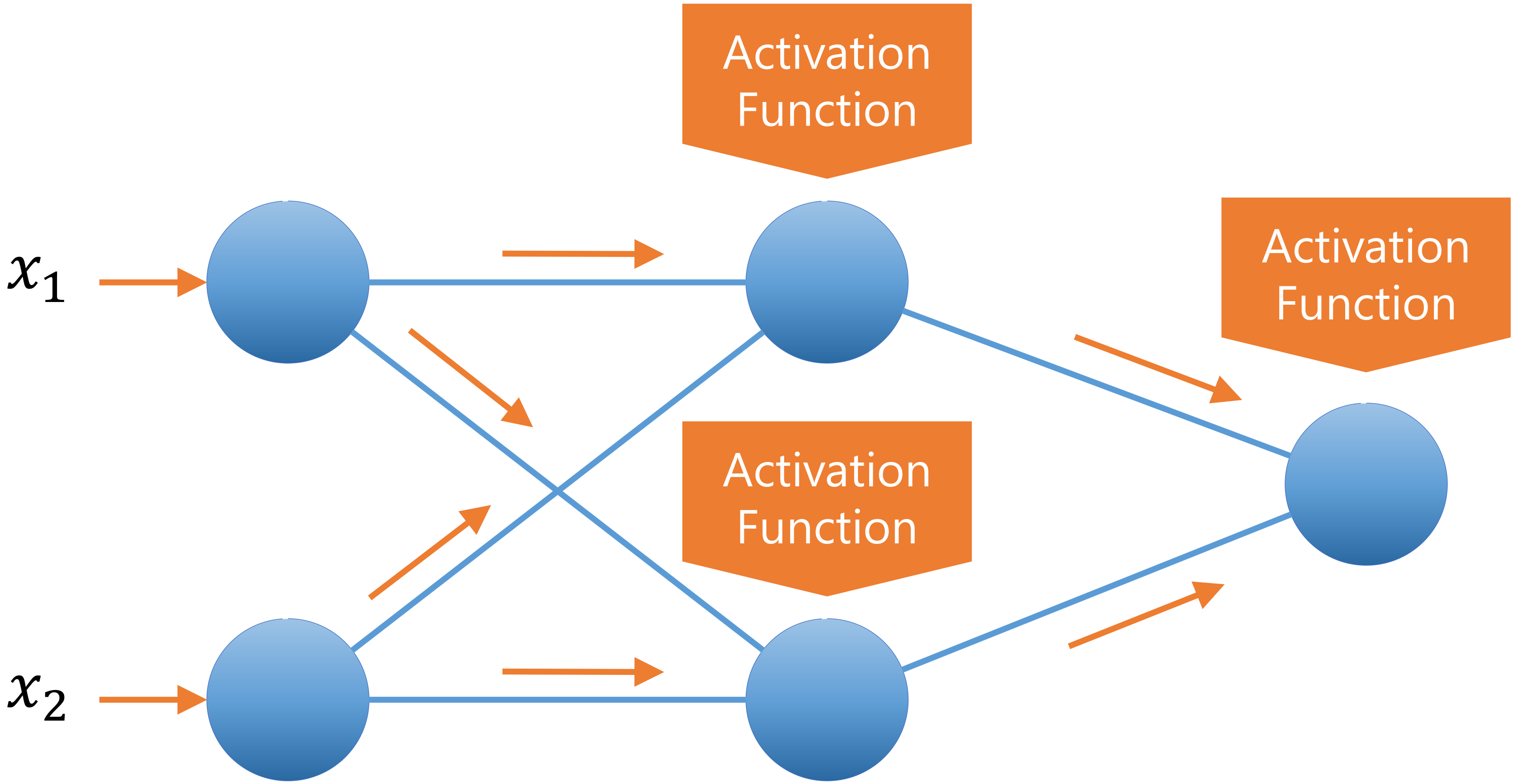
$$\Leftrightarrow$$

$$(w_1w_5 + w_2w_6)x_1 + (w_3w_5 + w_4w_6)x_2$$

$$\Leftrightarrow$$

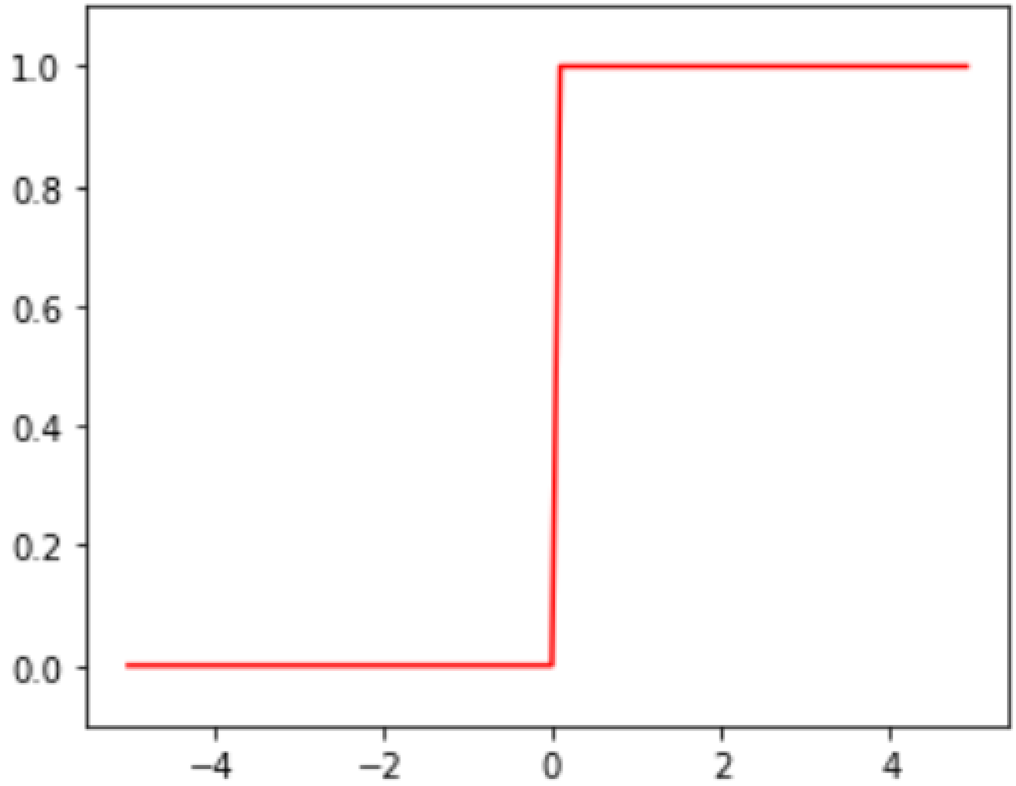
$$x_2 = \frac{(w_1w_5 + w_2w_6)}{(w_3w_5 + w_4w_6)}x_1$$

Activation Function



Step Function

$$y = \begin{cases} 0 \ (x \leq 0) \\ 1 \ (x > 0) \end{cases}$$



Activation Function

Middle Layer

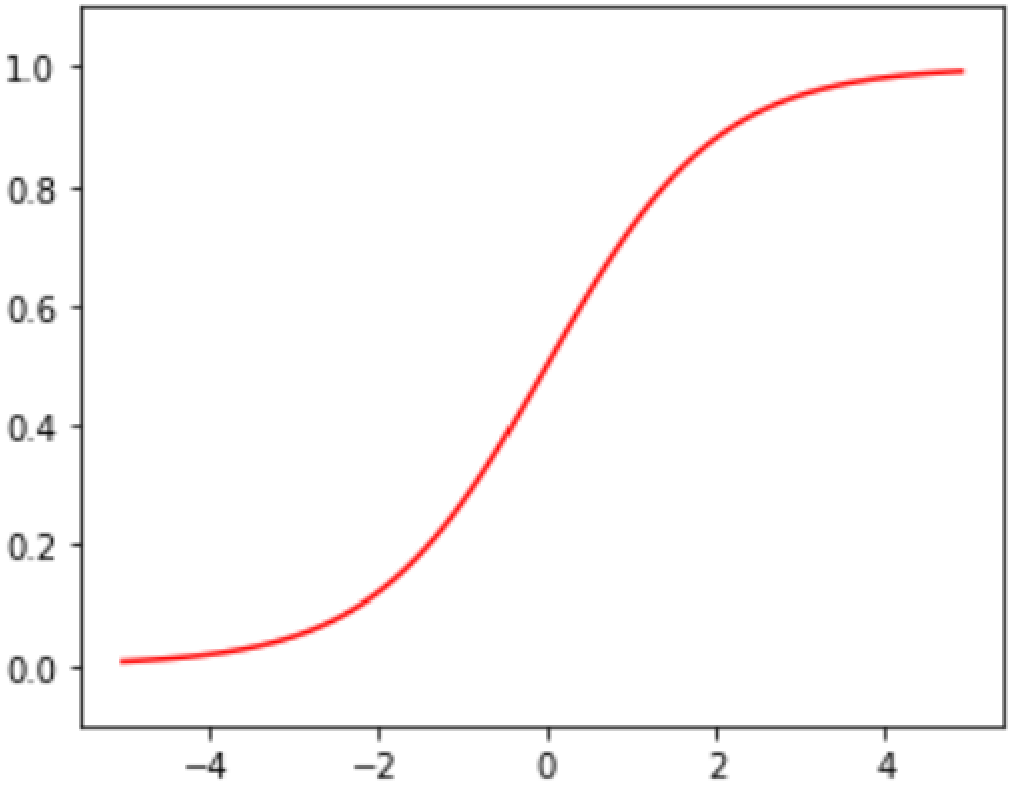
- Sigmoid Function
- Tanh Function
- ReLU Function

Output Layer

- Identity Function
- Sigmoid Function
- Softmax Function

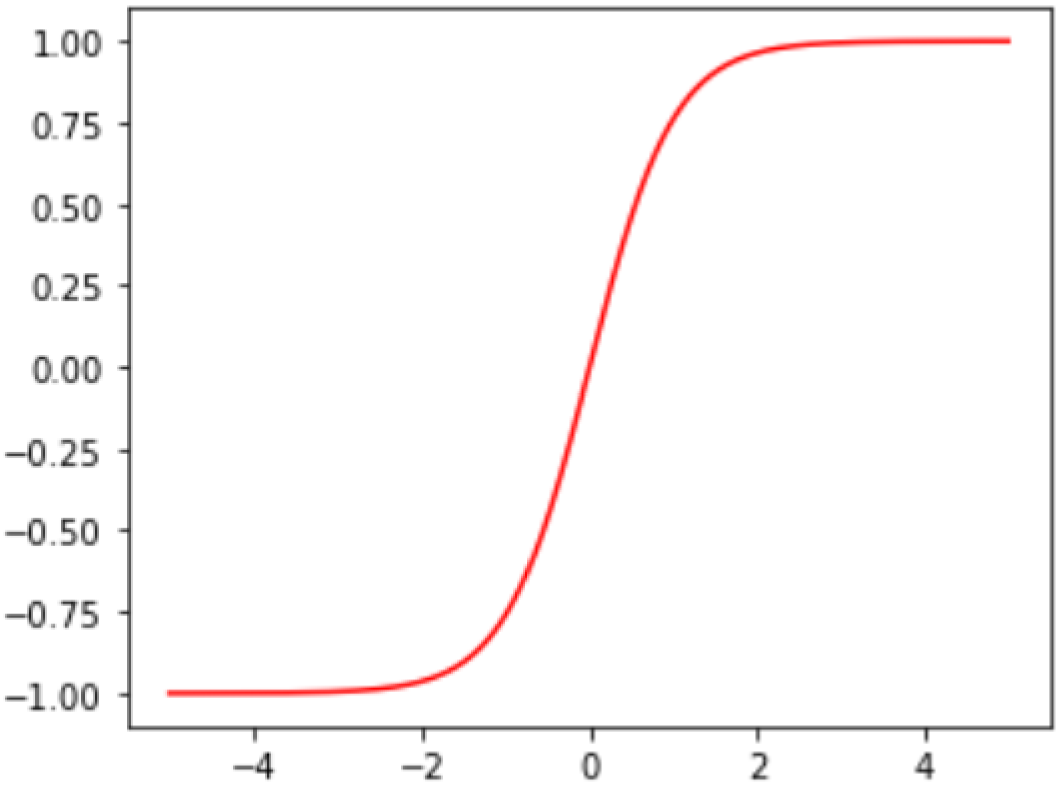
Sigmoid Function

$$y = \frac{1}{1 + \exp(-x)}$$



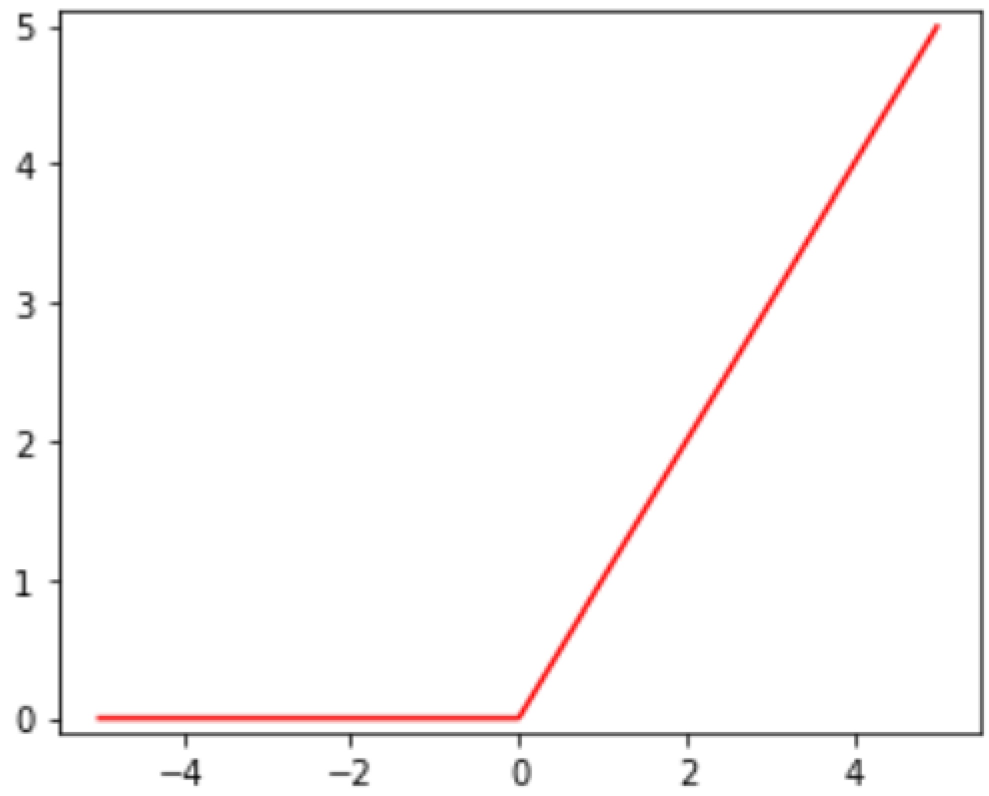
Tahn Function

$$y = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$



ReLU Function

$$y = max(0, x)$$

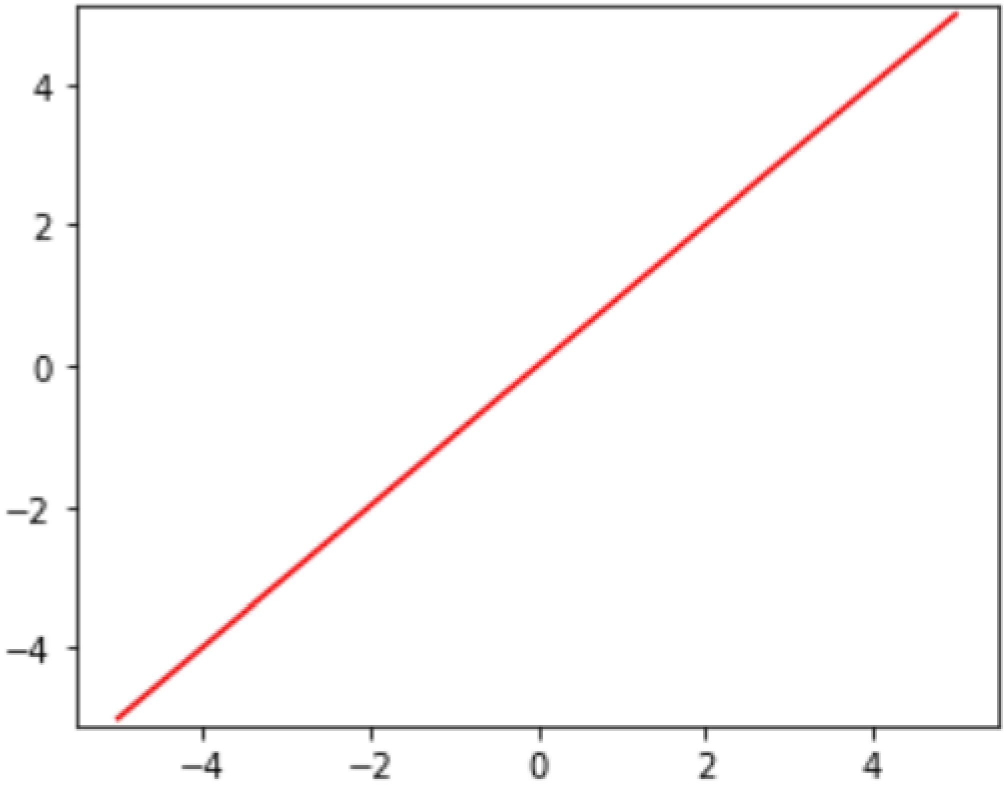


Activation Function for Output Layer

Middle Layer	Output Layer
• Sigmoid Function	• Identity Function
• Tanh Function	• Sigmoid Function
• ReLU Function	• Softmax Function

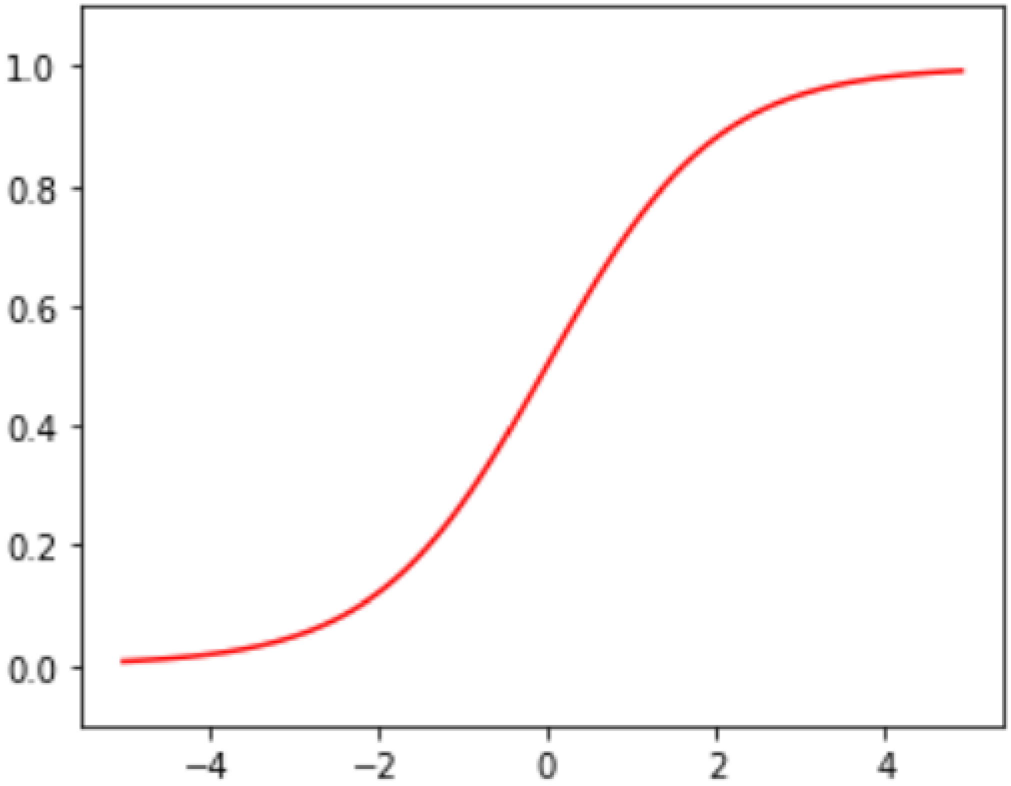
Identity Function

$$y = x$$



Sigmoid Function

$$y = \frac{1}{1 + \exp(-x)}$$



Softmax Function

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

10. Loss Function

Loss Function

Loss function is used to evaluate the model performance.

It quantifies the degree to which the model's predicted values are deviated from the true values.

A neural network model are trained to find the best parameters that minimize the returned value of the loss function.

Mean Squared Error (MSE)

$$MSE(y_i, \hat{y}_i) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

y_i: True values

$\hat{y}_i$: Predicted values

Mean Absolute Error (MAE)

$$MAE(y_i, \hat{y}_i) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

y_i: True values

$\hat{y}_i$: Predicted values

Cross Entropy Loss

$$E(p, y) = -\sum_{i} p_i \log y_i$$

p_i: True probability distribution

y_i: Predicted probability distribution

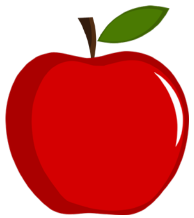
Example: Cross-Entropy Loss

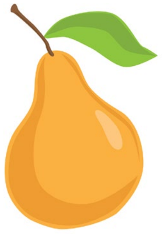
Multiclass classification: Apple, Orange, Pear

The model outputs the probability distribution of the classes.

The class with the highest probability will be the predicted class.

$$[P(Apple), P(Orange), P(Pear)]$$





Example: Cross-Entropy Loss (Continued)

Compute the cross-entropy loss: $E(p,y) = -\sum_i p_i \log y_i$

$$p = \begin{bmatrix} p_1 \\ p_2 \\ p_n \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ y_n \end{bmatrix}$$

$$E(p,y) = -[p_1\ p_2\ p_3]\begin{bmatrix} \log y_1 \\ \log y_2 \\ \log y_n \end{bmatrix}$$

$$= -(p_1 \log y_1 + p_2 \log y_2 + \cdots + p_n \log y_n)$$

Example: Cross-Entropy Loss (Continued)

	True Probability Distribution	Predicted Probability Distribution
	$= [1, 0, 0]$	$= [0.6, 0.2, 0.2]$
	$= [0, 1, 0]$	$= [0.3, 0.4, 0.3]$
	$= [0, 0, 1]$	$= [0.2, 0.3, 0.5]$

$$E(p_i, y_i) = -[1 \;\; 0 \;\; 0] \begin{bmatrix} \log 0.6 \\ \log 0.2 \\ \log 0.2 \end{bmatrix} \qquad E(p, y) = -\sum_i p_i \log y_i$$

$$= -(1 \times \log 0.6 + 0 \times \log 0.2 + 0 \times \log 0.2)$$

$$= -\log 0.6 = 0.22$$

The logarithm base is Napier's e

Cross-Entropy Loss of Binary Classification

$$\text{LogLoss} = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log p_i + (1 - y_i)\log(1 - p_i))$$

p_i: True probability distribution

y_i: Predicted probability distribution

Cross-Entropy Loss of Binary Classification (Continued)

Since the logarithm base is Napier's e, $y_i \log p_i + (1 - y_i) \log(1 - p_i) = \cdots$

If $p_i = 1, y_i = 1$: $$1 \times \log 1 + (1 - 1) \log(1 - 1)$$ $$= 1 \times 0 + 0 = 0$$

If $p_i = 0, y_i = 0$: $$= 0 \times \log 0 + (1 - 0) \log(1 - 0)$$ $$= 0 + \log 1 = 0$$

If $p_i = 1, y_i = 0$: $$= 0 \times \log 1 + (1 - 0) \log(1 - 1)$$ $$= 0 + \log 0 = \infty$$

If $p_i = 0, y_i = 1$: $$= 1 \times \log 0 + (1 - 1) \log(1 - 0)$$ $$= \log 0 + \log 1 = \infty$$

Why Loss Function?

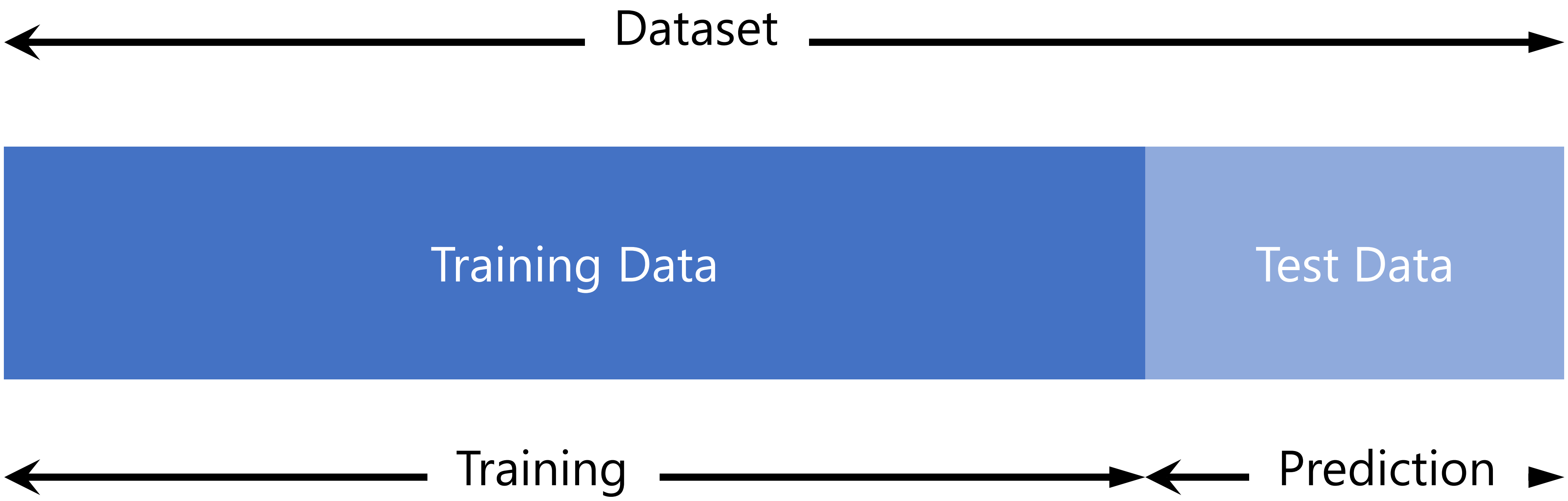
A neural network uses differential to learn the best parameters.

The advantage of using a loss function is that the differential of a loss function will never be 0.

Thus, a neural network can continue to learn until it finds the best parameters.

11. Training Neural Network

Data Splitting and Generalizability



Develop a model that can predict unknown data.

Batch Learning and Mini Batch Learning

Batch Learning:

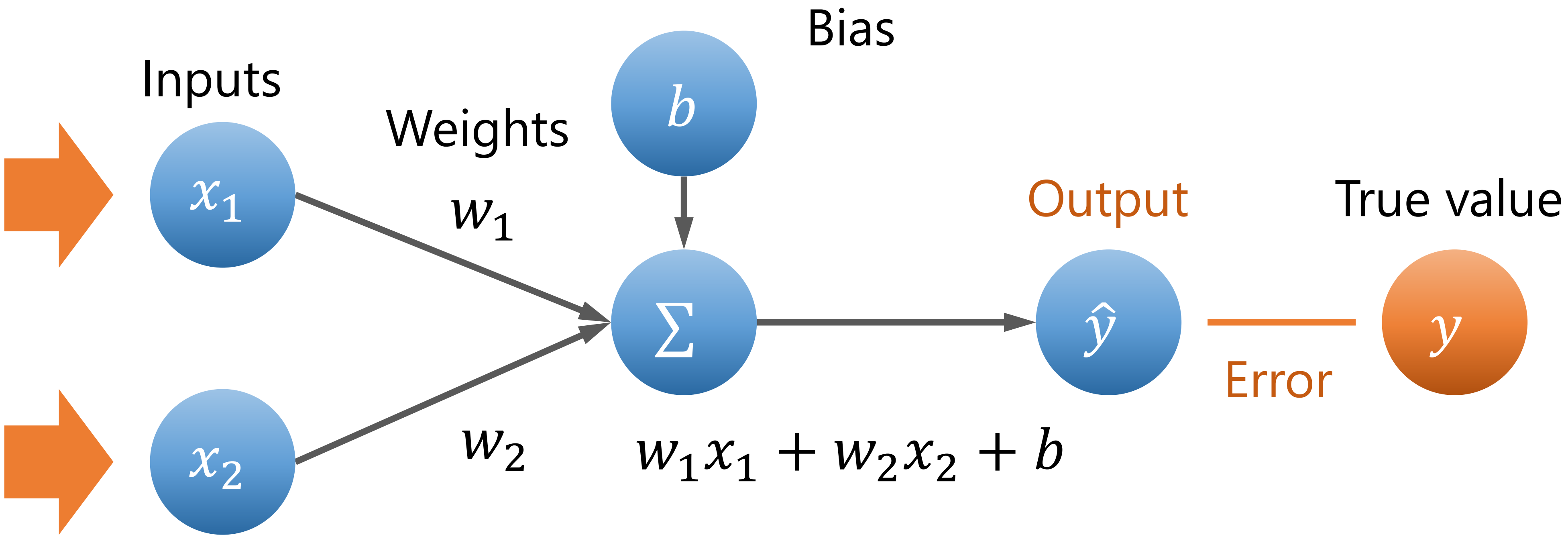
Use all cases in the training dataset for training all together.

Mini-batch learning:

Train the model using a randomly sampled subset of the training set.

Repeat the random sampling and training

ANN's Learning Process



ANN learns the weights and biases that minimizes the difference between the outputs and the true values.

Backpropagation

STEP 1 Compute the output using inputs and randomly set weight and biases.

STEP 2 Compute the error between the output and the true value.

STEP 3 Check if the error was minimized.

STEP 4 If not, update the weights and biases.

STEP 5 Compute the output using the updated weights and biases.

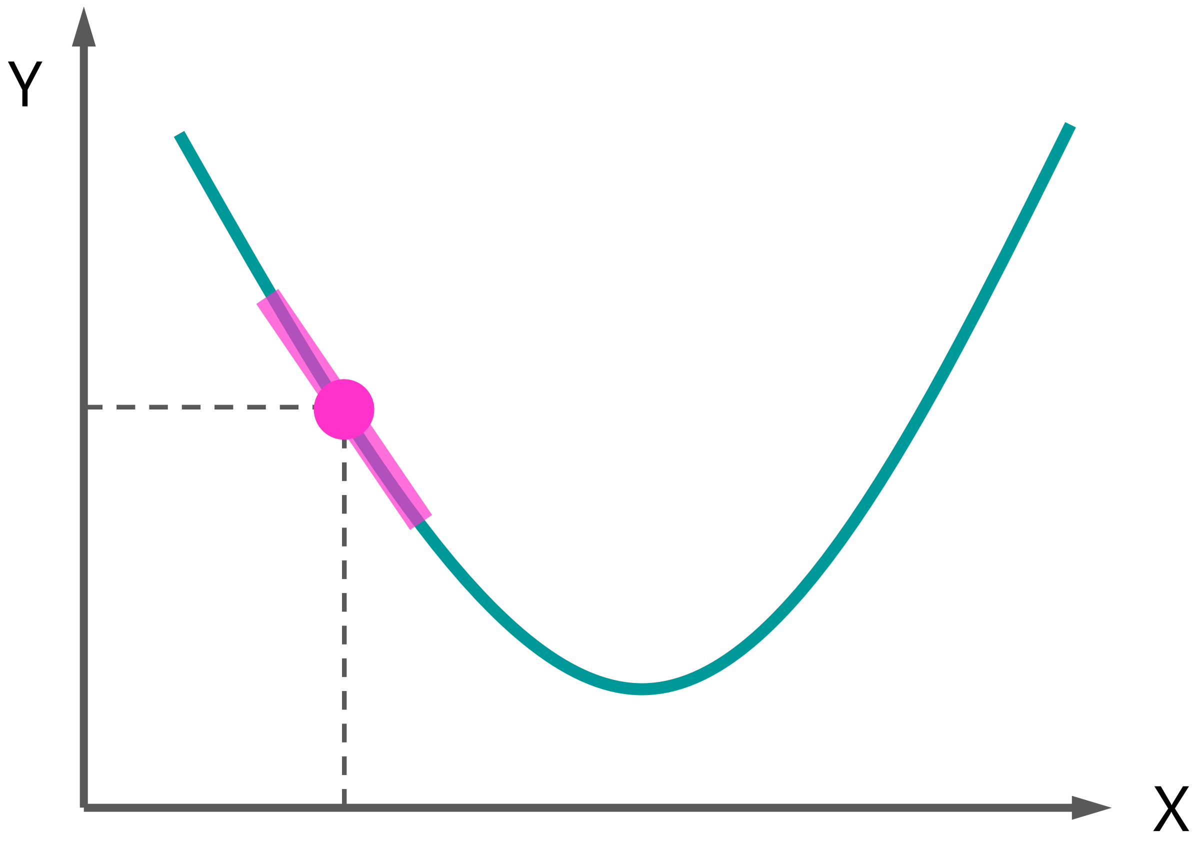
Simplified Example: Backpropagation

$$Wx = \hat{y}$$

Input	True Value	W=2		W=3		W=4		W=5	
		Output	Error	Output	Error	Output	Error	Output	Error
1	4	2	2	3	1	4	0	5	1
2	9	4	5	6	3	8	1	10	1
3	11	6	5	9	2	12	1	15	4
		Σ Error	12	Σ Error	6	Σ Error	2	Σ Error	6

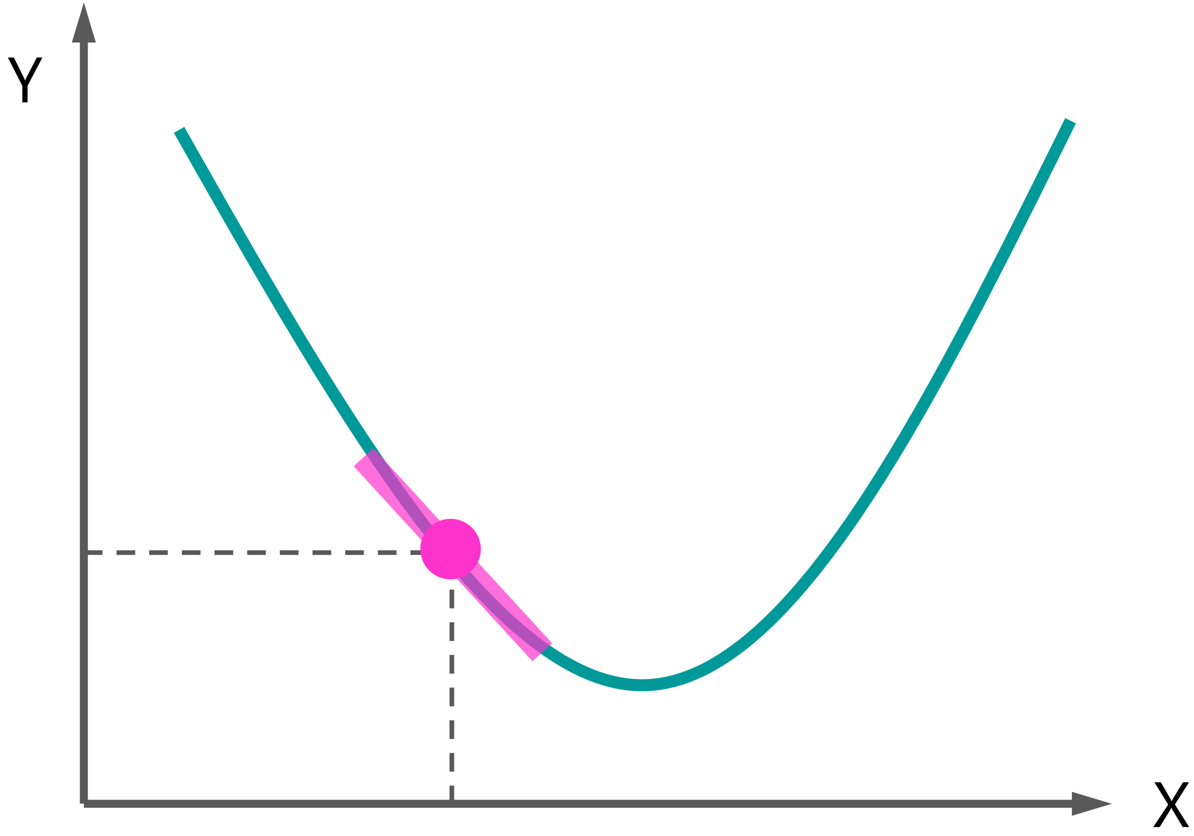
12. Gradient Descent Method (1)

What is Gradient Descent?



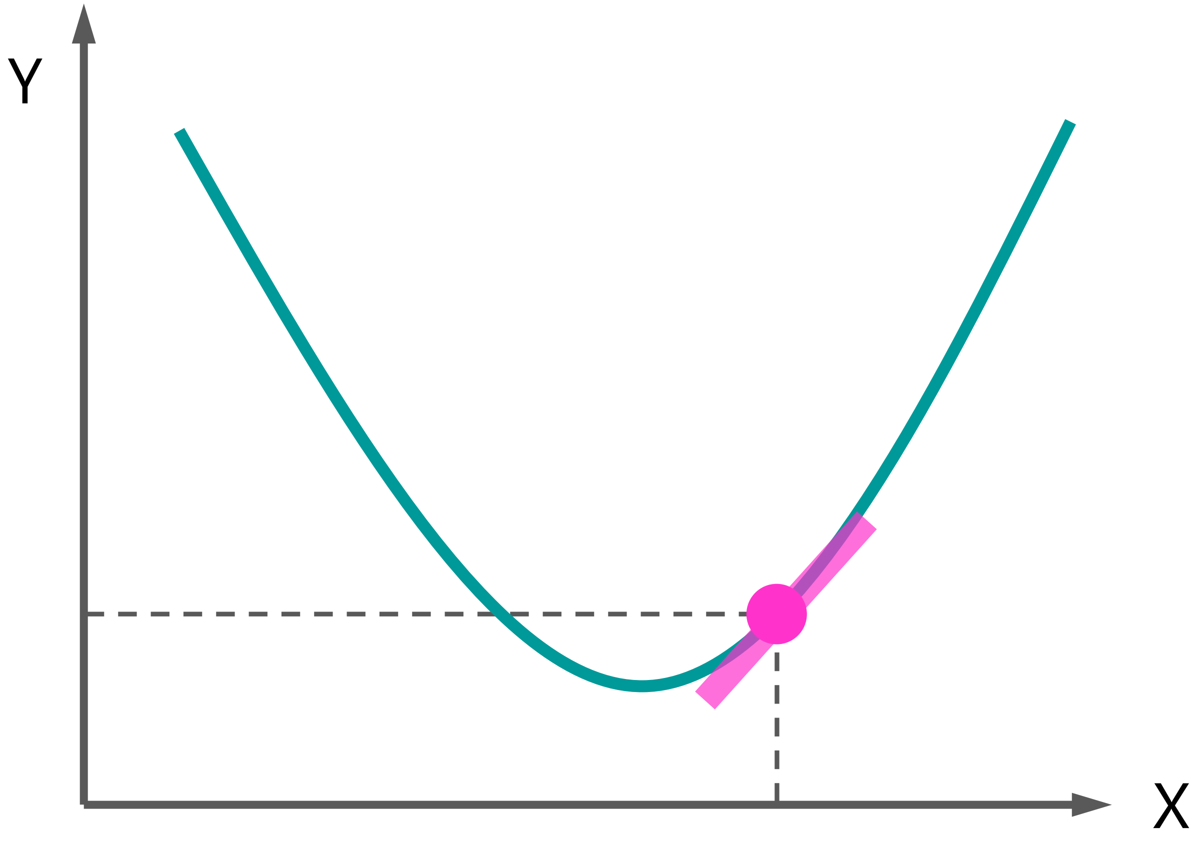
An optimization algorithm that identifies a local minimum of a differentiable function using gradients of a function.

What is Gradient Descent?



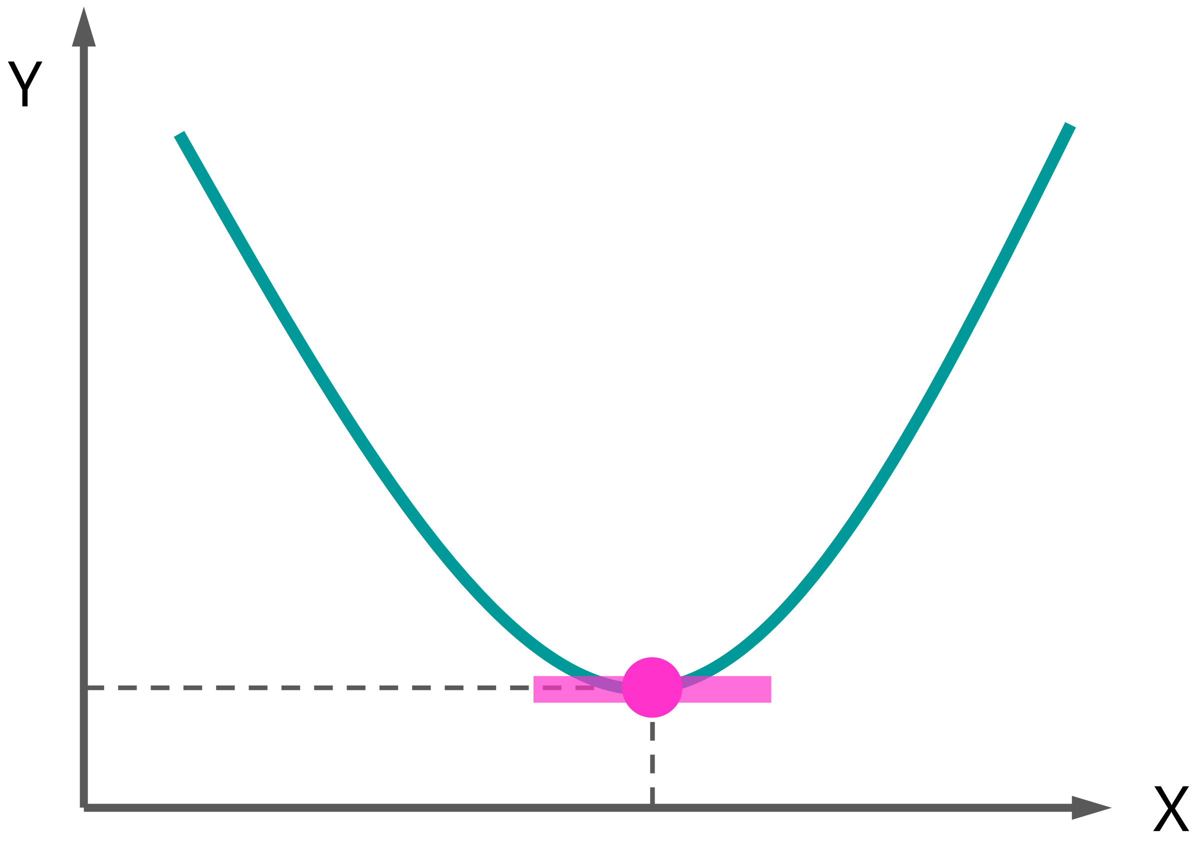
An optimization algorithm that identifies a local minimum of a differentiable function using gradients of a function.

What is Gradient Descent?



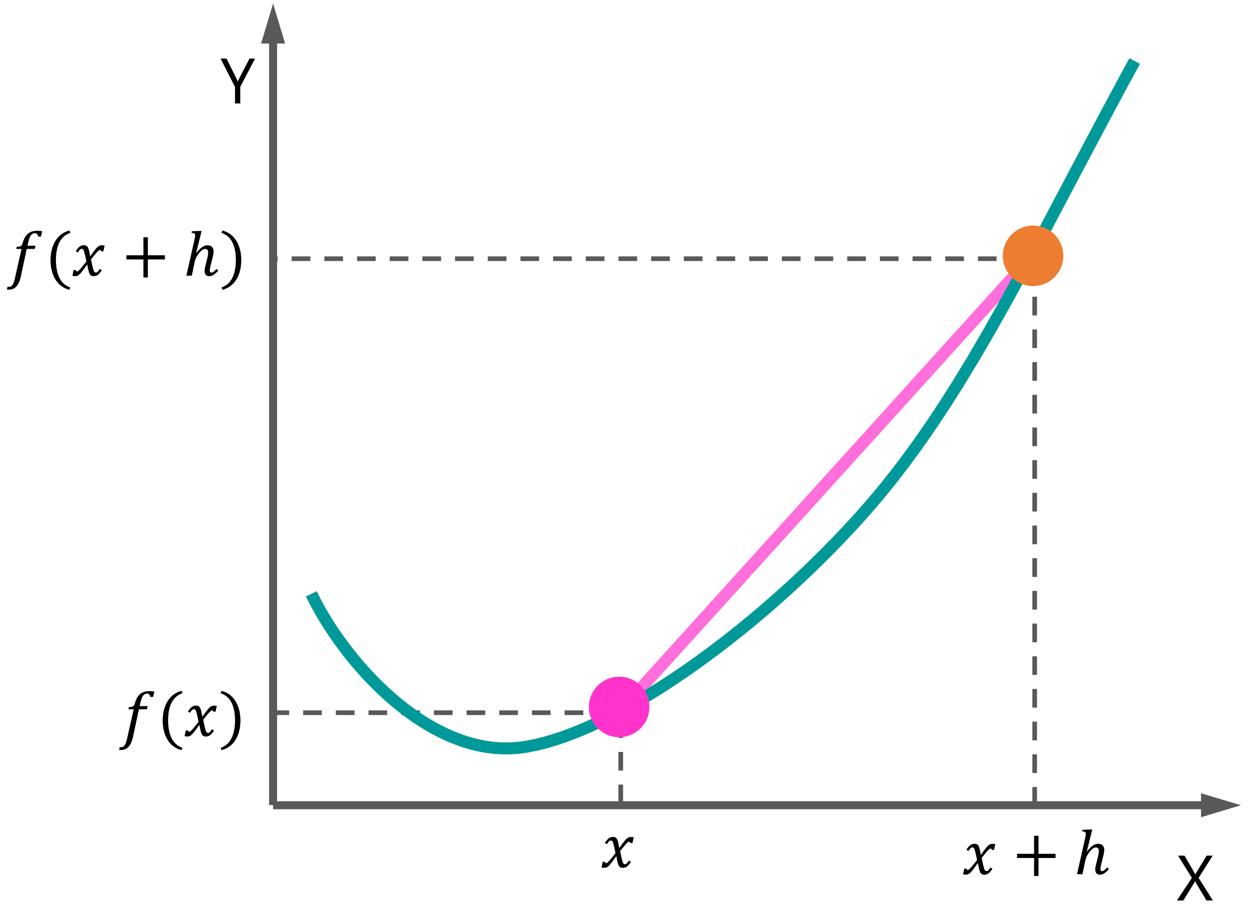
An optimization algorithm that identifies a local minimum of a differentiable function using gradients of a function.

What is Gradient Descent?



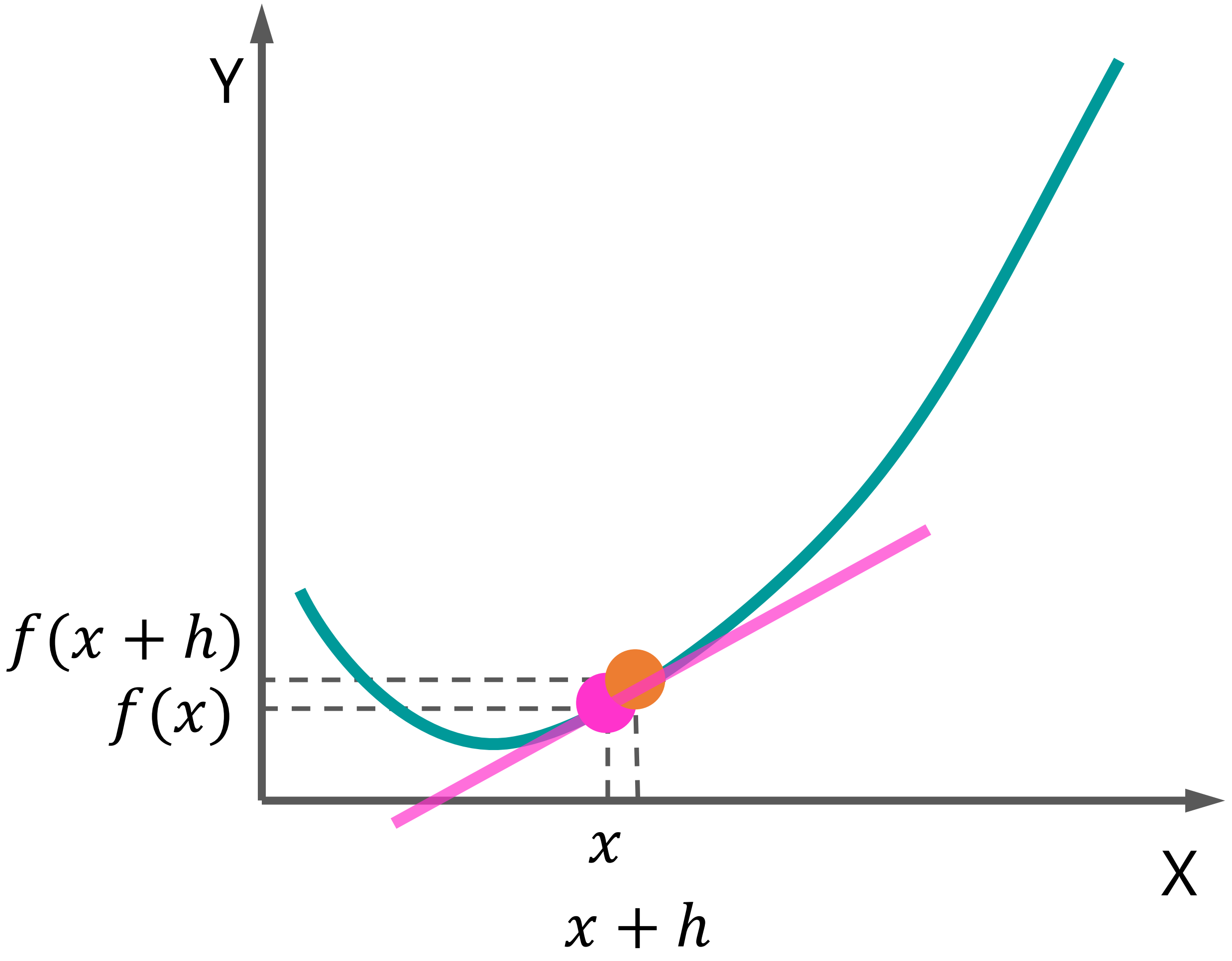
An optimization algorithm that identifies a local minimum of a differentiable function using gradients of a function.

Differential



Differential

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$



Partial Differential

$$\frac{\partial}{\partial x_n} f(x_1, x_2, \dots, x_n)$$

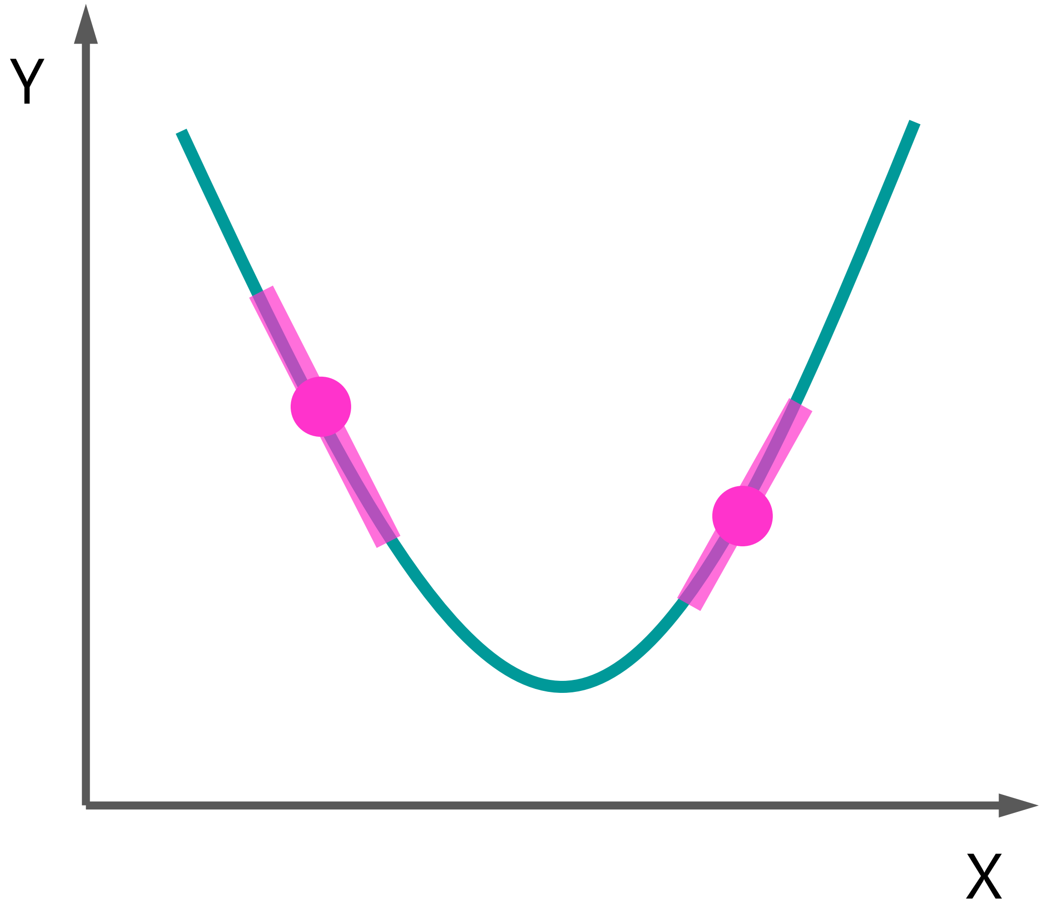
$$\frac{\partial}{\partial x_1}(3x_1^2 + 4x_1 \underbrace{+ 2x_2^2 + 5}_{\text{Constant}})$$

$$= 6x_1 + 4$$

Gradient

The slope of a tangent line at a point in the function.

We can find the optimal parameter value using gradient.



Parameter Optimization

Optimization problem:

A problem where we try to minimize or maximize the output of a function.

Objective function

A function that we try to minimize or maximize.

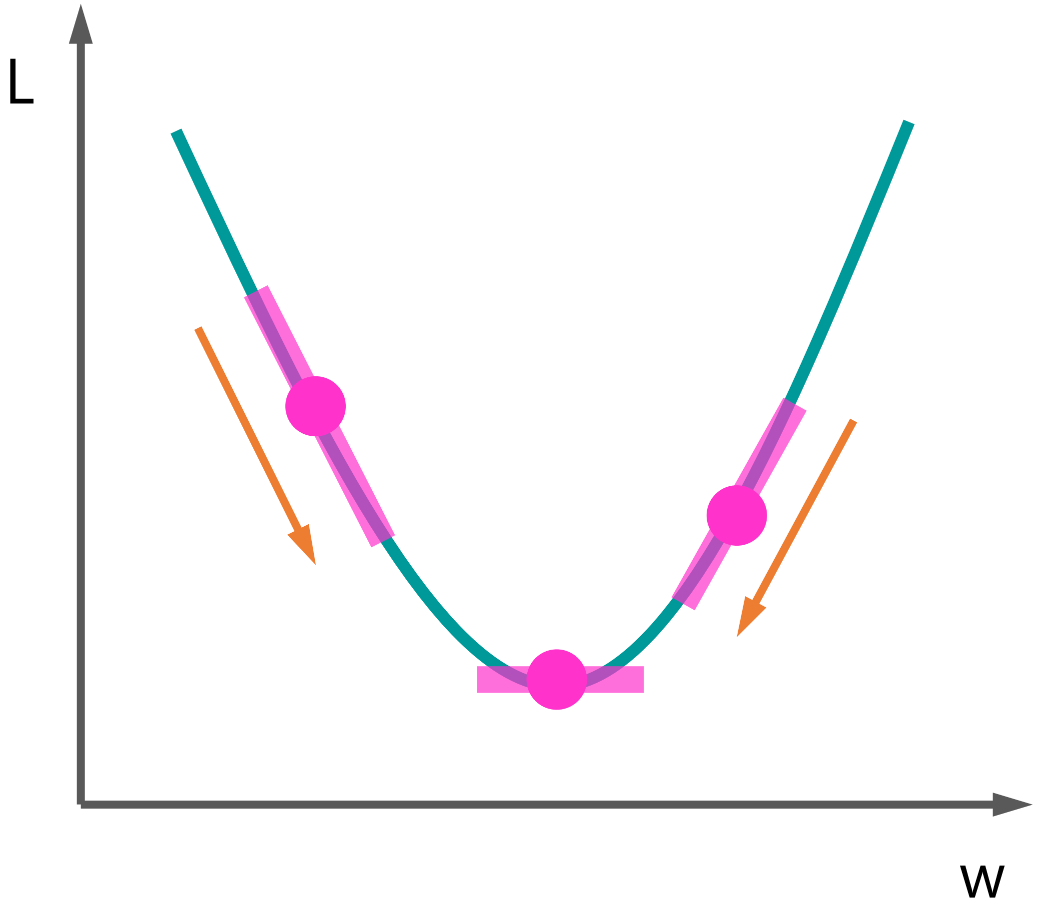
e.g., Loss function

Parameter Optimization (Continued)

$$\frac{\partial L}{\partial w}$$

$$w = w - \eta \frac{\partial L}{\partial w}$$

η: Learning rate



13. Gradient Descent Method (2)

Types of Gradient Descent Method

- Batch gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent

Batch Gradient Descent

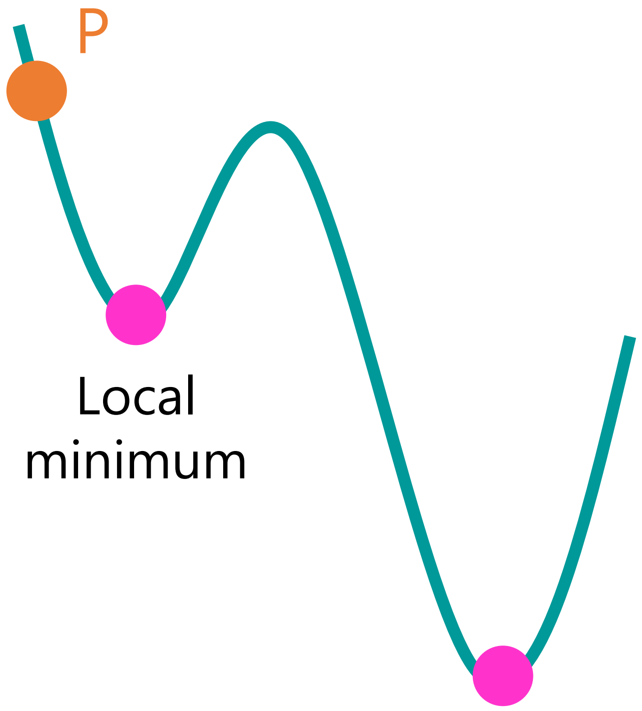
- Uses all the training dataset to explore the minima.
- Strength

  Results are stable.

- Weakness:

  High computation cost and memory usage

  Likely to fall into local minima



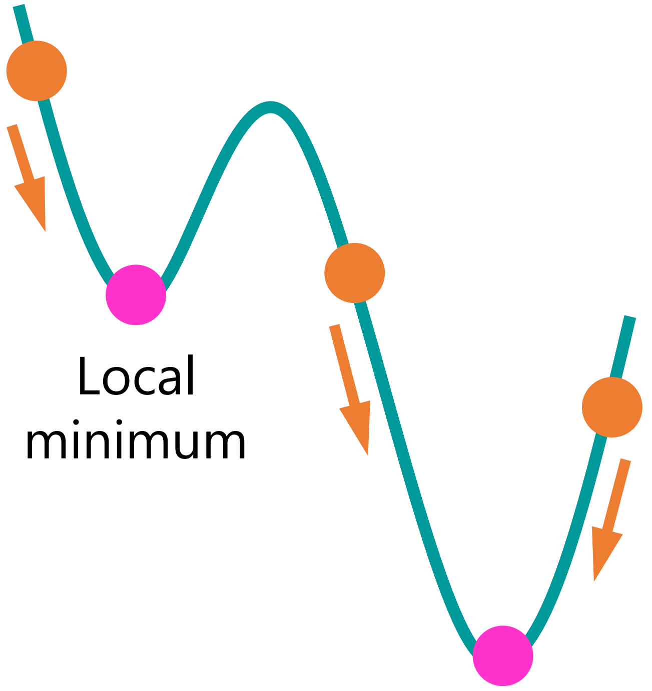
Stochastic Gradient Descent

- Uses randomly selected datapoint.
- Strength

  Low computational cost and fast

  Less likely to fall into local minima
- Weakness:

  Unstable: Susceptible to outliers



Mini-batch Gradient Descent

- Hybrid of batch gradient descent and stochastic gradient descent.
- Splits the training data into smaller mini-batches, and updates parameters for each mini-batch.
- Less computational cost and less memory usage than batch gradient descent.
- More stable (less susceptible to outliers) than stochastic gradient descent.

14. Chain Rule

Chain Rule

$$f(x) \qquad\qquad g(x)$$

Composite function: $F(x) = f(g(x))$

$$x \xrightarrow{g} g(x) \xrightarrow{f} f(g(x))$$

Chain rule: $F'(x) = f'\big(g(x)\big)g'(x)$

Chain Rule: Another Expression

$$F(x) = f(g(x))$$

$$x \xrightarrow{g} g(x) \xrightarrow{f} f(g(x))$$

$u = g(x)$ $\quad$ $y = f(u)$

$$u' = \frac{du}{dx} \qquad y' = \frac{dy}{du}$$

Chain rule: $F'(x) = \dfrac{dy}{dx} = \dfrac{dy}{du} \cdot \dfrac{du}{dx}$

Chain Rule: Mathematical Proof

$$u = g(x) \qquad y = f(u)$$

$$u' = \frac{du}{dx} \qquad y' = \frac{dy}{du}$$

$$du = \frac{du}{dx}dx \qquad dy = \frac{dy}{du}du$$

$$dy = \frac{dy}{du} \cdot \frac{du}{dx}dx$$

$$F'(x) = \frac{dy}{dx}$$

$$= \frac{dy}{du} \cdot \frac{du}{dx}dx\frac{1}{dx}$$

$$= \frac{dy}{du} \cdot \frac{du}{dx}$$

Example: Chain Rule

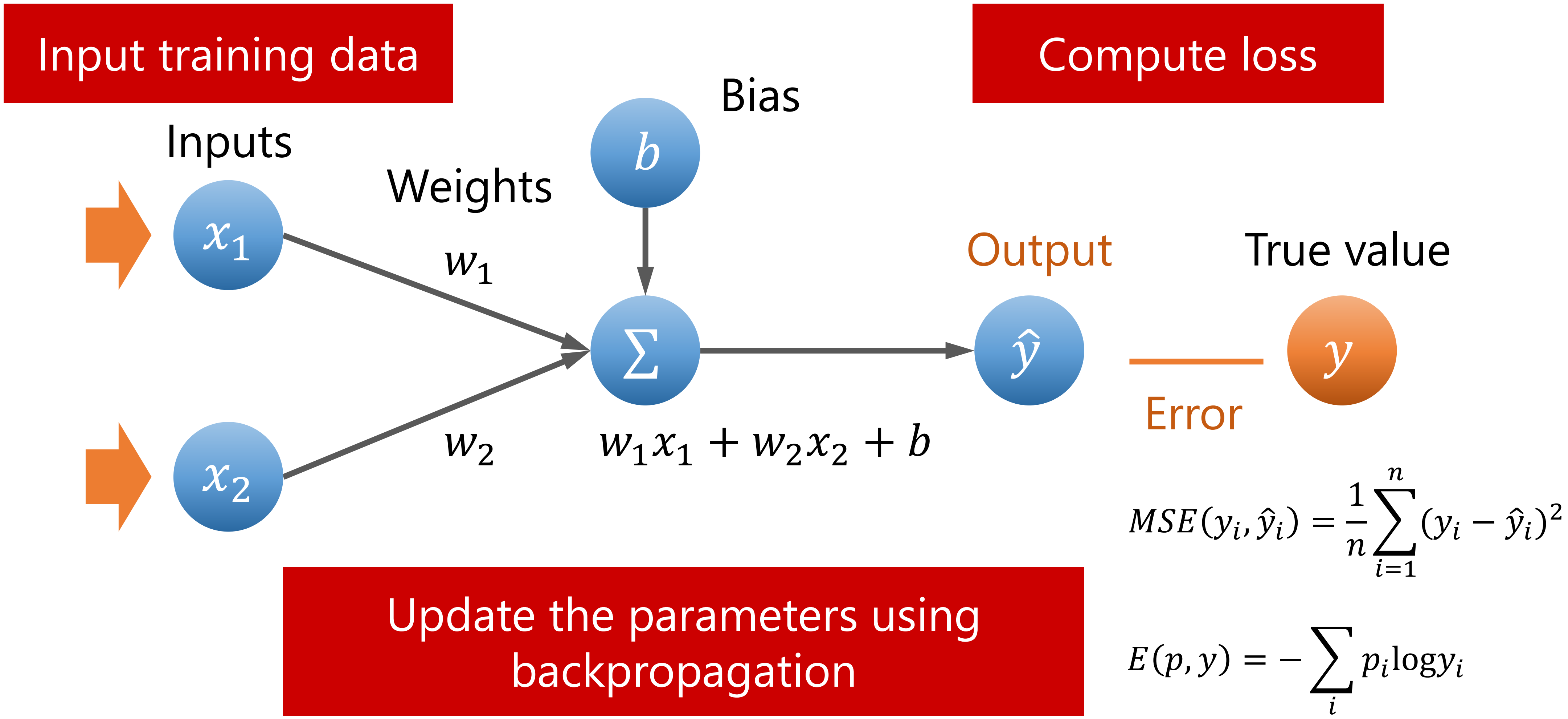
$$f(x) = (2x+1)^3$$

$$u = 2x + 1 \qquad f(u) = u^3$$

Chain Rule: $$\begin{aligned} f'(x) &= f'(u) \cdot u' \\ &= 3u^2 \times 2 \\ &= 6u^2 \\ &= 6(2x+1)^2 \\ &= 24x^2 + 24x + 6 \end{aligned}$$

15. Backpropagation

Recap: ANN's Learning Process



Minimize Loss

Set parameters randomly	Loss function	Optimization

$$w_1x_1 + w_2x_2 + b$$

$$MSE(y_i, \hat{y}_i) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$MAE(y_i, \hat{y}_i) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

$$E(p, y) = -\sum_{i} p_i \mathrm{log} y_i$$

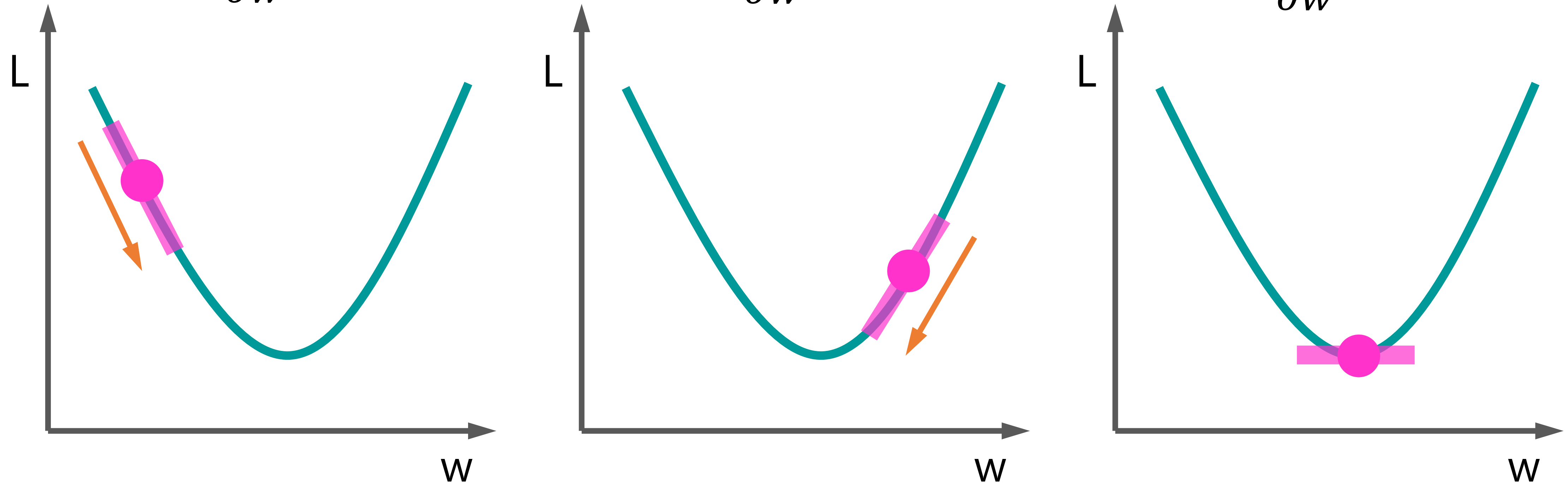
$$\frac{\partial L}{\partial w}$$

$$w = w - \eta \frac{\partial L}{\partial w}$$

η: Learning rate

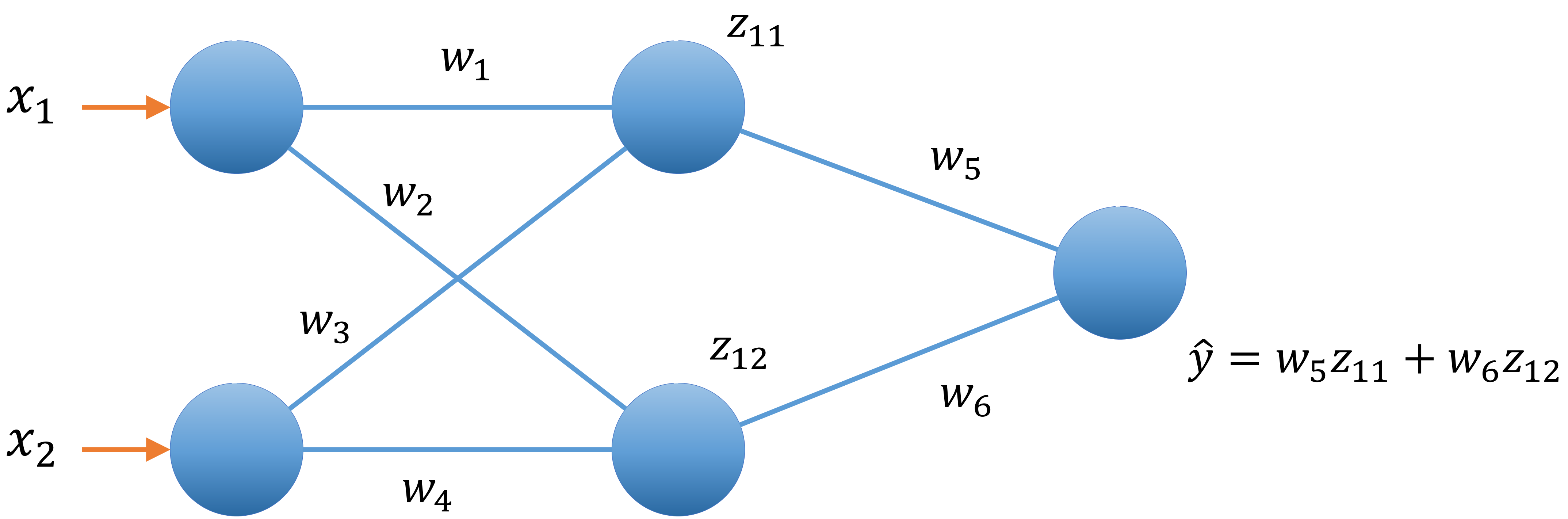
Gradient Descent and Backpropagation

$$\frac{\partial L}{\partial w} < 0 \qquad \frac{\partial L}{\partial w} > 0 \qquad \frac{\partial L}{\partial w} = 0$$



$$w = w - \eta \frac{\partial L}{\partial w}$$

Example: Backpropagation



Example: Backpropagation

$$L = MSE(y_i, \hat{y}_i) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\frac{\partial L}{\partial w_5} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_5}$$

$$L = (y - \hat{y})^2$$

$$= \{y - (w_5 z_{11} + w_6 z_{12})\}^2$$

$$\frac{\partial L}{\partial \hat{y}} = (y^2 - 2y\hat{y} + \hat{y}^2)'$$

$$= -2(y - \hat{y}) = -2(y - w_5 z_{11} - w_6 z_{12})$$

$$= -2(y - w_5 z_{11} - w_6 z_{12})$$

$$\frac{\partial \hat{y}}{\partial w_5} = z_{11}$$

$$\frac{\partial L}{\partial w_5} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_5} = -2z_{11}(y - w_5 z_{11} - w_6 z_{12})$$

Example: Backpropagation

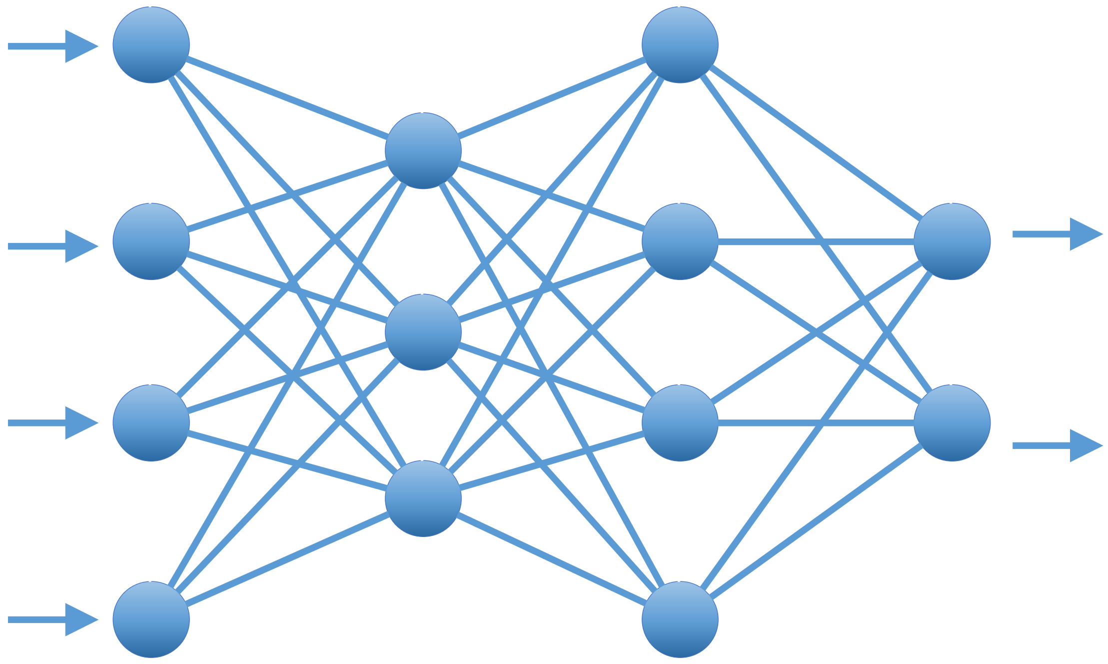
$$w_5 = w_5 - \eta \frac{\partial L}{\partial w_5}$$

16. Vanishing Gradient Problem

Vanishing Gradient Problem

In ANNs, gradients often get smaller and smaller as the increase in the number of hidden layers.

In DNNs with many hidden layers, the gradients can become almost 0, and thus, they can hardly learn from the data.



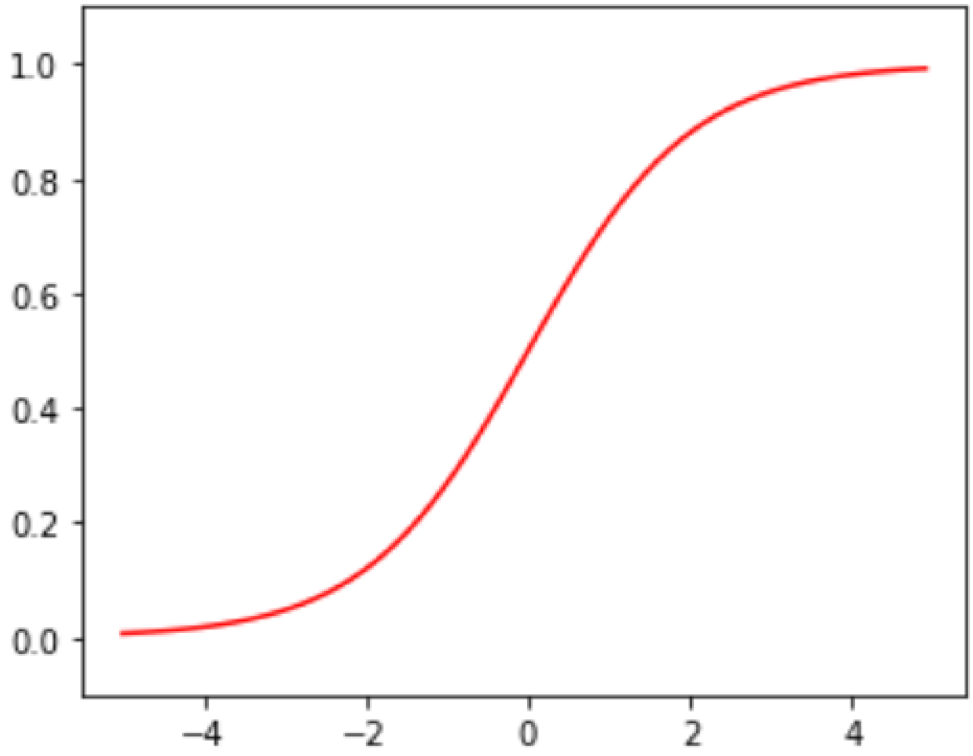
Why do Gradient Vanish?

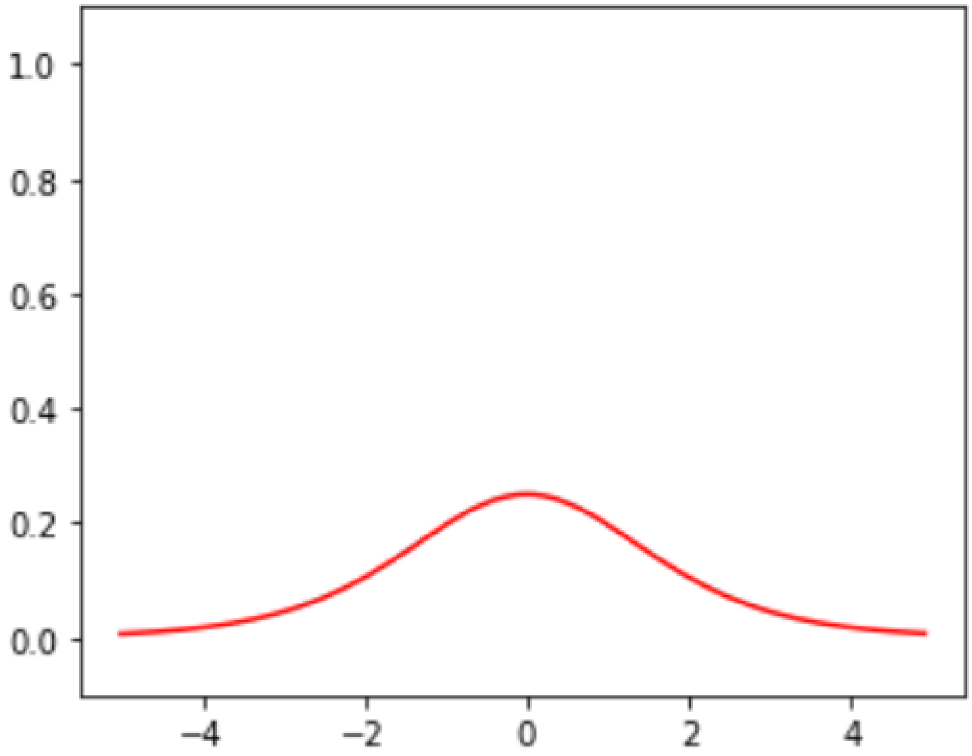
$$f'(x)\, f'(x)\, f'(x)\, f'(x) \cdots f'(x) \approx 0$$

Sigmoid function

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

$$f'(x) = \frac{e^{-x}}{(1+e^{-x})^2}$$

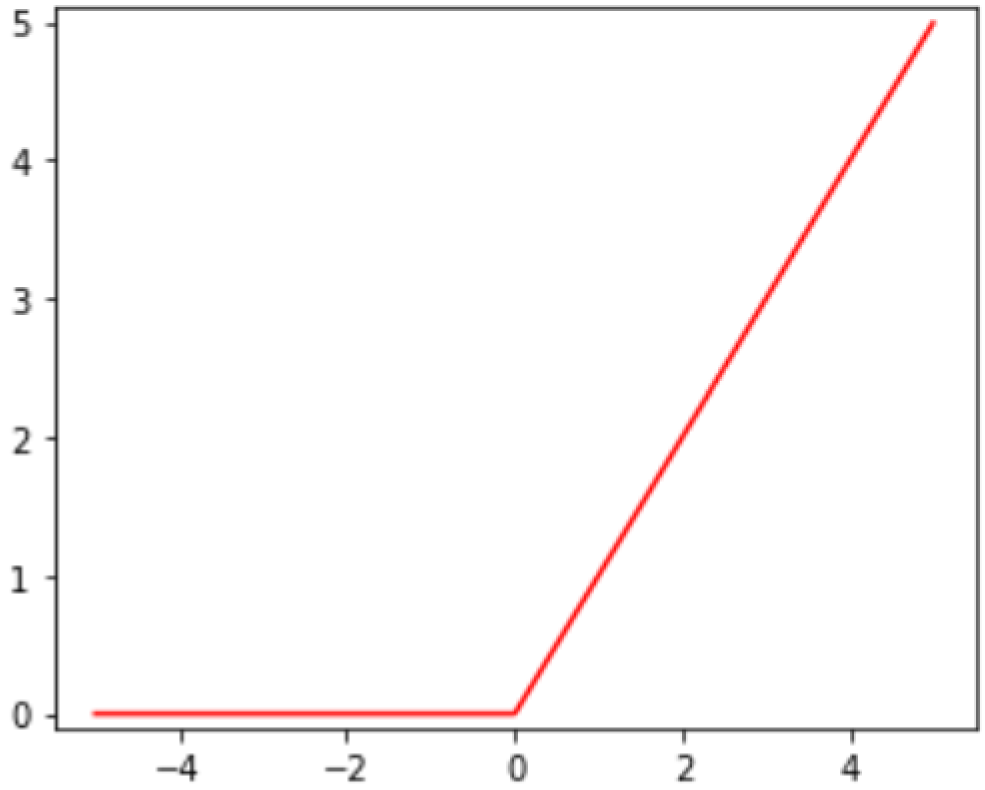




Use ReLU Function

$$f(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

$$f'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$



17. Nonsaturating Activation Functions

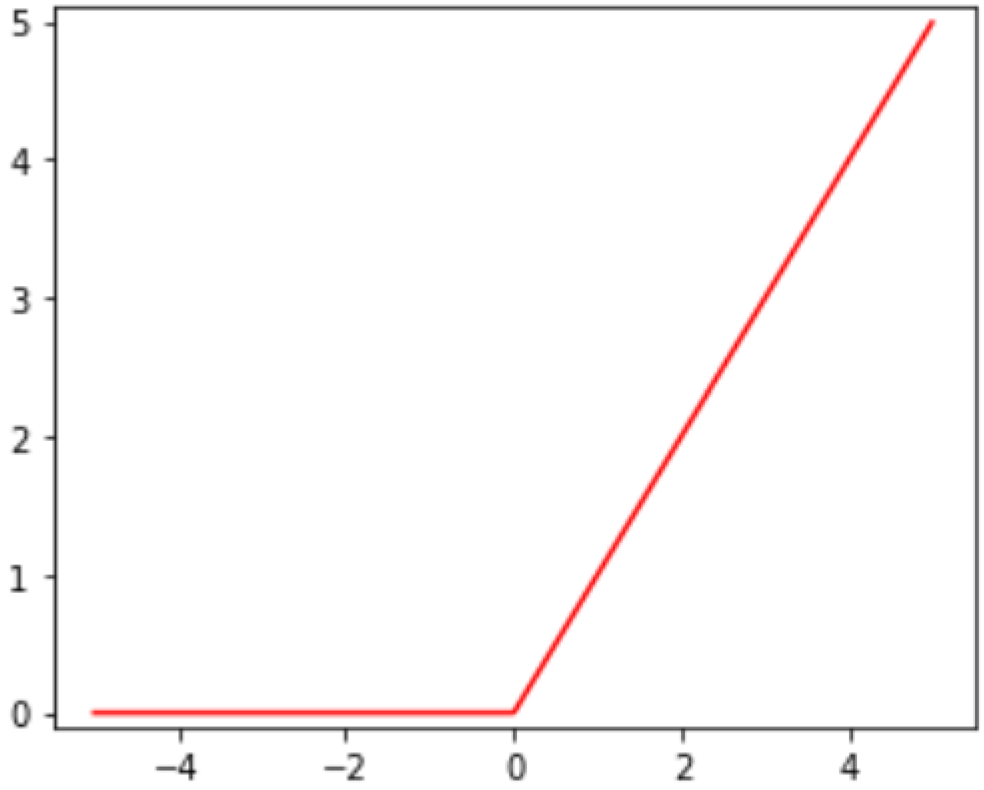
Dying ReLUs

$$w_i = w_i - \eta \frac{\partial L}{\partial w_i}$$

$$f(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

The larger η is, the more likely w_i is negative, and ReLU returns 0.

→ When we use high learning rate, Dying ReLUs tends to occur.



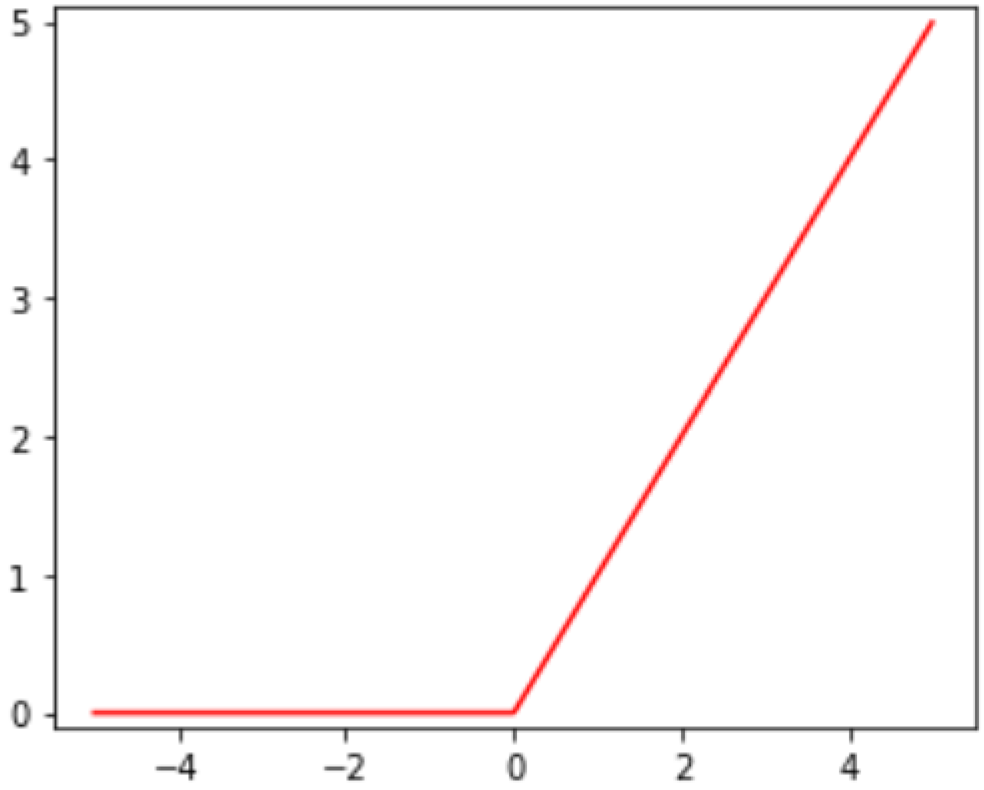
Leaky ReLU

The output is αx when the input is negative. So, the output is not exactly horizontal when x < 0

The output values are slightly larger than 0 when the input value is negative.

The Dying ReLU problem is less likely to occur.

$$f(x) = \begin{cases} x & x > 0 \\ \alpha x & x \leq 0 \end{cases} \qquad (\alpha = \ 0.01)$$



ELU and SELU Functions

ELU function (Exponential Linear Unit):

$$f(x) = \begin{cases} x & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases} \qquad (\alpha > 0)$$

SELU function (Scaled Exponential Linear Unit):

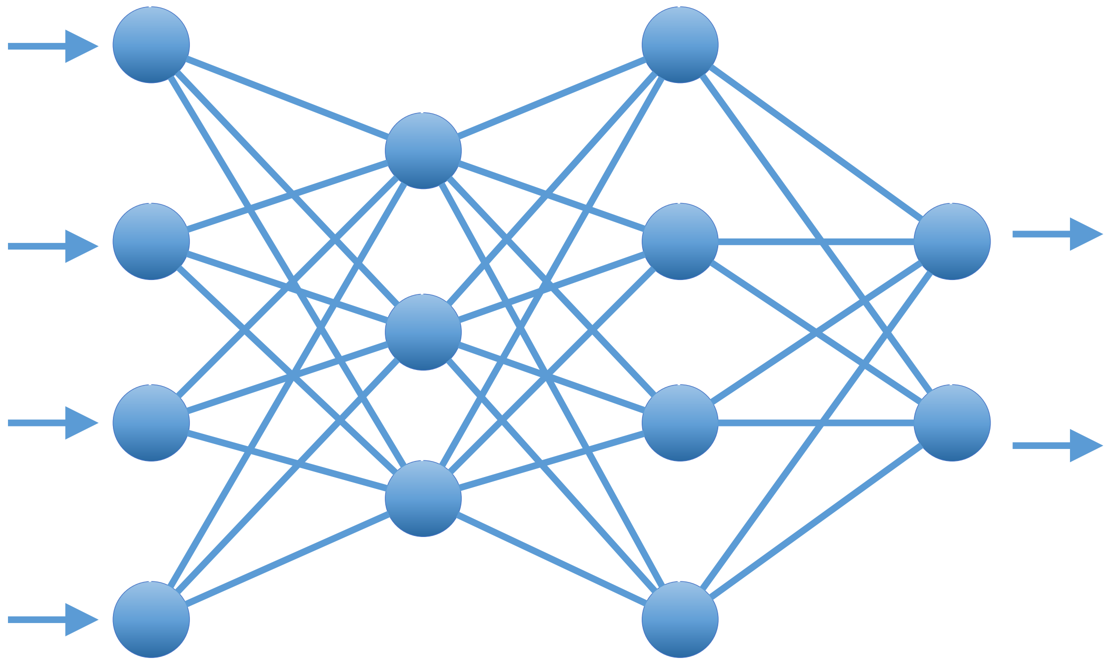
$$f(x) = \lambda \begin{cases} x & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases} \qquad (\lambda > 1, \alpha > 0)$$

18. Parameter Initialization

Recap: Vanishing Gradient Problem

In ANNs, gradients often get smaller and smaller as the increase in the number of hidden layers.

In DNNs with many hidden layers, the gradients can become almost 0, and thus, they can hardly learn from the data.



Parameter Initialization

Measure to prevent vanishing gradient problems.

Initialize parameter values with a larger range when the model size is large, and with a smaller range when the model size is small.

Popular methods:

- Xavier initialization
- He initialization

Xavier Initialization

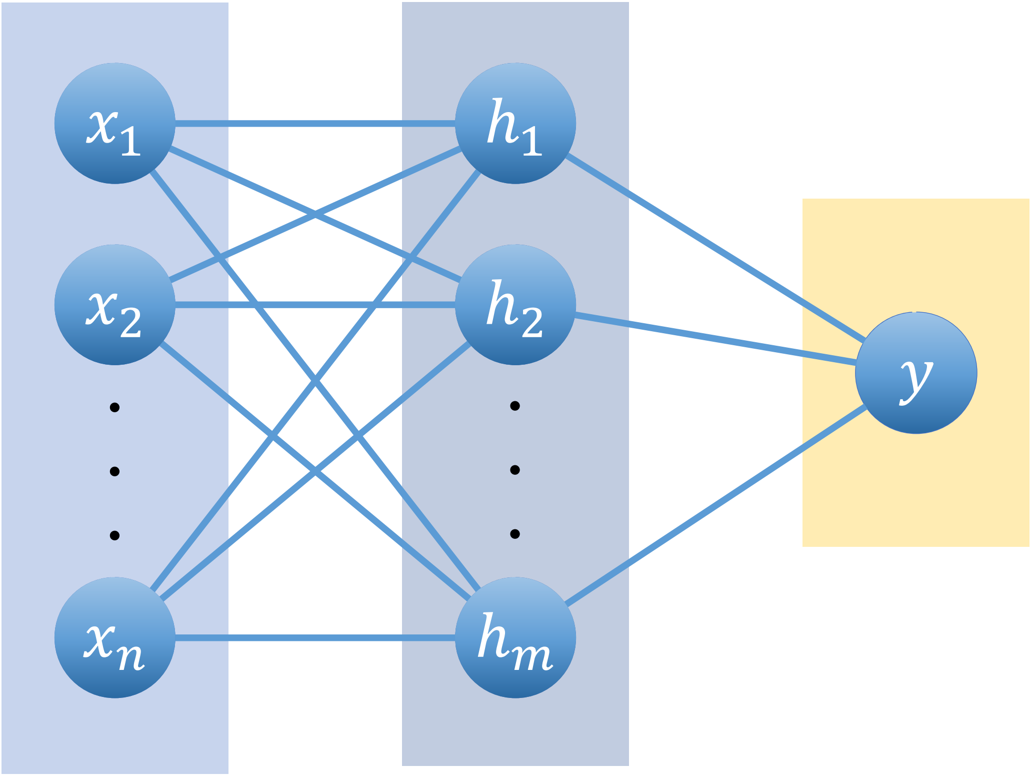
Initial parameter value:

Normal distribution

$$Mean = 0$$

$$std = \frac{1}{\sqrt{n}}$$

n: N of nodes



$Mean = 0$

$std = \frac{1}{\sqrt{n}}$

$Mean = 0$

$std = \frac{1}{\sqrt{m}}$

He Initialization

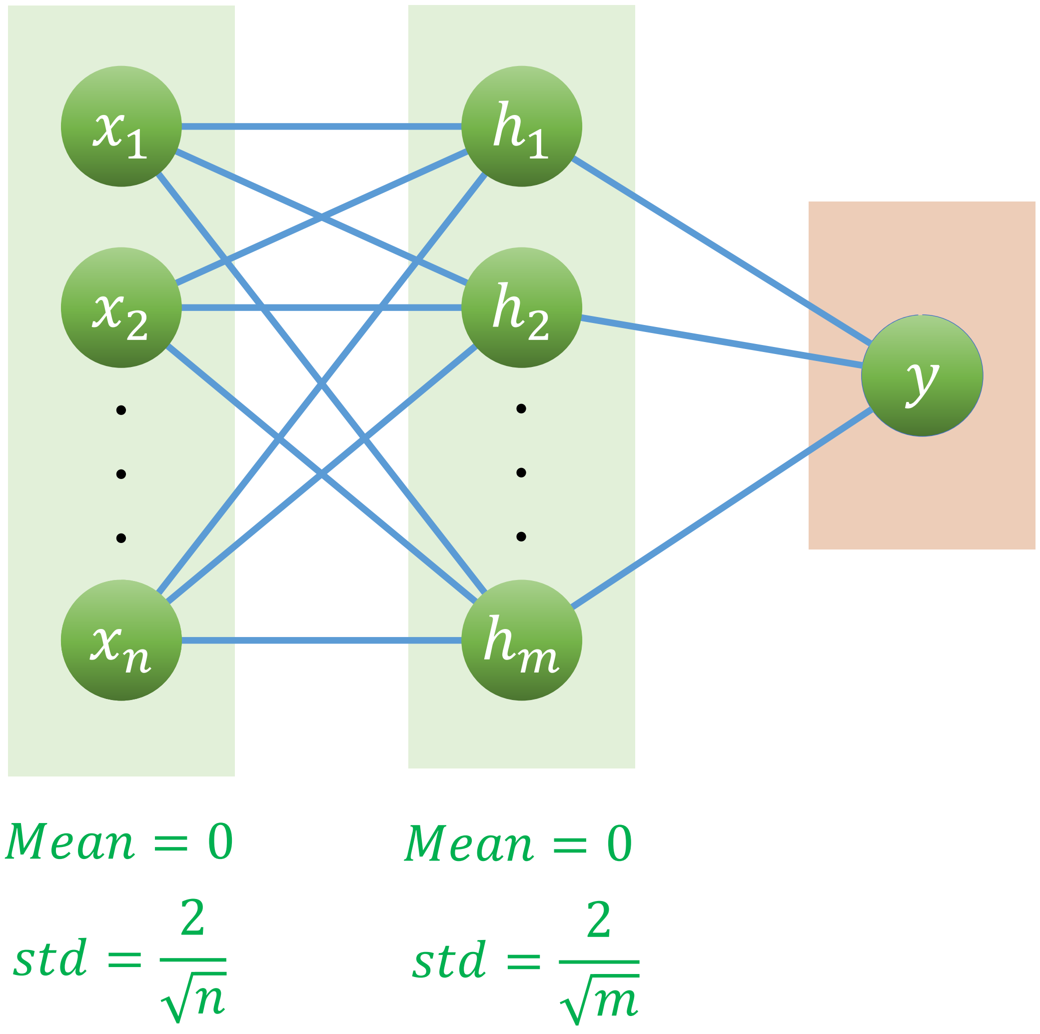
Initial parameter value:

Normal distribution

$$Mean = 0$$

$$std = \frac{2}{\sqrt{n}}$$

n: N of nodes



19. ANN Regression with Keras

Import Libraries

```
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from tensorflow.keras.layers import Activation, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam

import warnings
warnings.filterwarnings('ignore')
```

Load and Prepare the Dataset

```
# Load dataset
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="¥s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]

# Create X and y
X = pd.DataFrame(data)
y = target

# Split data into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Standardize the features
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Define Artificial Neural Network

```
# Define ANN
model = Sequential()
model.add(Dense(128, activation='relu',
                input_shape=(13,)))
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
```

Leaky ReLU function

```
from tensorflow.keras.layers import LeakyReLU
leaky_relu = LeakyReLU(alpha=0.01)
Dense(64, activation=leaky_relu)
```

ELU function

```
activation ='elu'
```

SELU function

```
activation= 'selu'
```

He Initialization

```
# Define ANN
model = Sequential()
model.add(Dense(128, activation='relu’, input_shape=(13,),
                        kernel_initializer="he_normal"))
model.add(Dense(64, activation='relu’,
                        kernel_initializer="he_normal"))
model.add(Dense(64, activation='relu’,
                        kernel_initializer="he_normal"))
model.add(Dense(32, activation='relu’,
                        kernel_initializer="he_normal"))
model.add(Dense(1))
```

Show the Model Summary

```
# Show the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               1792
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33
=================================================================
Total params: 16,321
Trainable params: 16,321
Non-trainable params: 0
_________________________________________________________________
```

1792+8256+4160+208+33=16321

$W_0 \times W_1 + b_1(\text{bias})$

$13 \times 128 + 128 = 1792$

$128 \times 64 + 64 = 8256$

$64 \times 64 + 64 = 4160$

$64 \times 32 + 32 = 2080$

$32 \times 1 + 1 = 33$

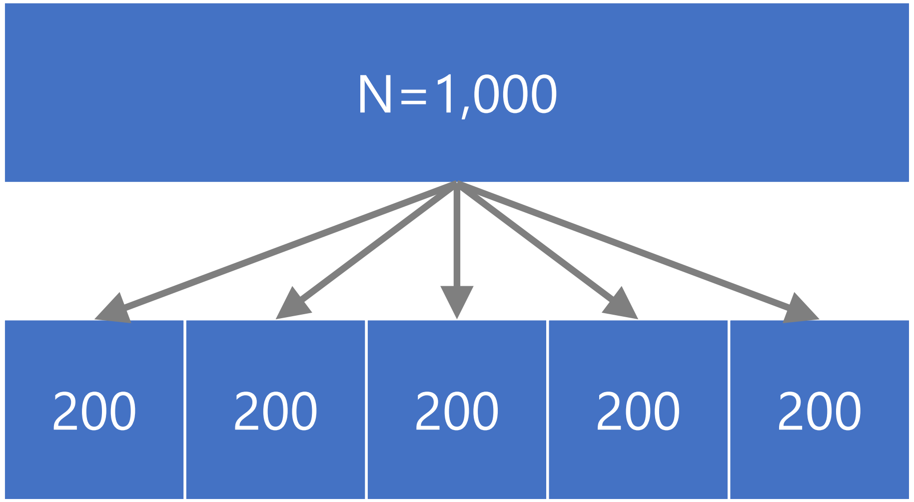
Compile the Model

```
# Compile the model
model.compile(loss='mse’,
              optimizer=Adam(lr=0.01),
              metrics=['mae'])
```

Fit the Model

```
# Fit the model
history = model.fit(X_train, y_train,
                    batch_size = 64,
                    epochs=1000,
                    validation_split=0.2,
                    verbose=1)
```

Mini-batch learning

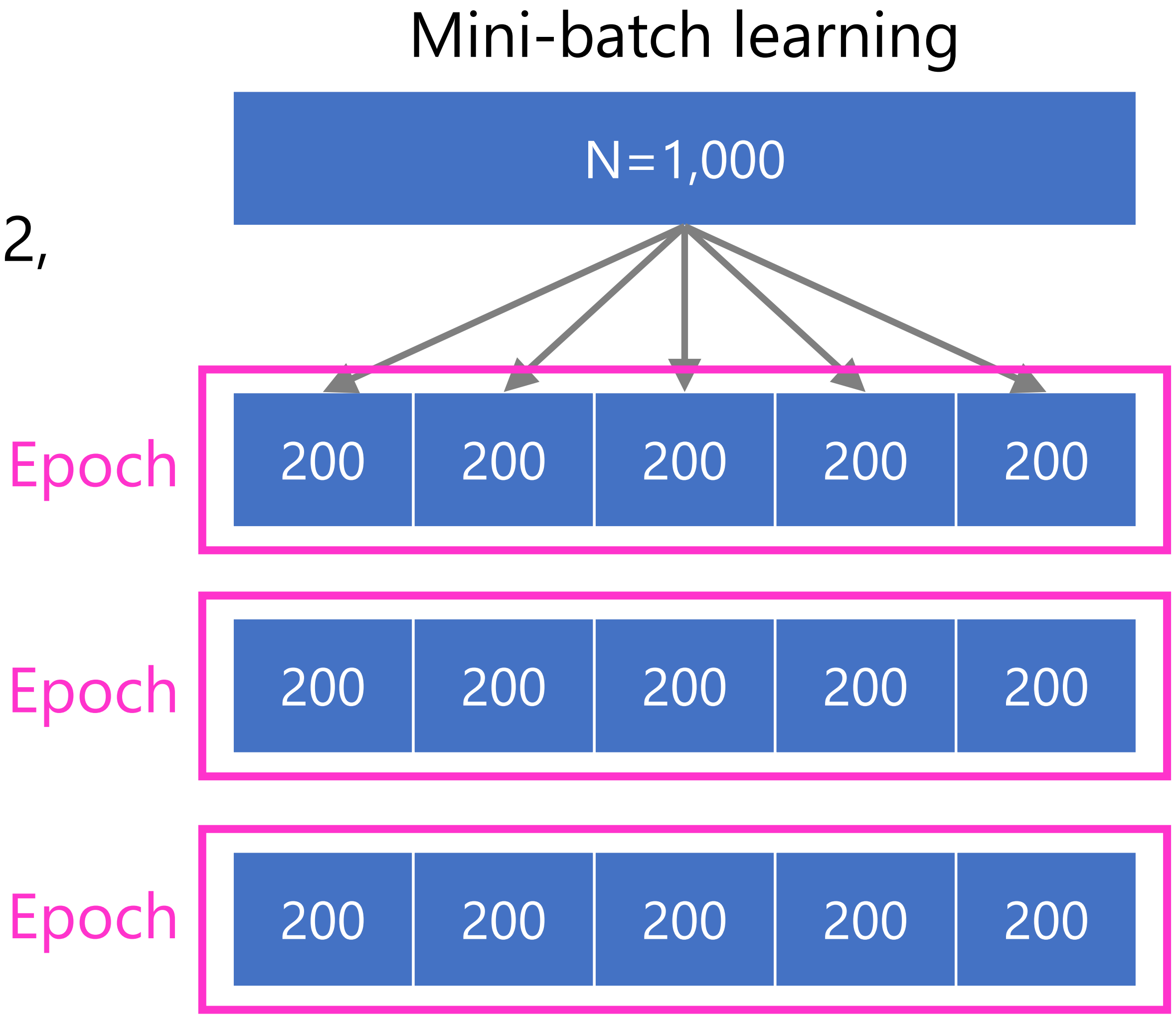


batch_size=200

2^n : 32, 64, 128, 256, 512, 1024, 2048

Fit the Model

```
# Fit the model
history = model.fit(X_train, y_train,
                    batch_size = 64,
                    epochs=1000,
                    validation_split=0.2,
                    verbose=1)
```

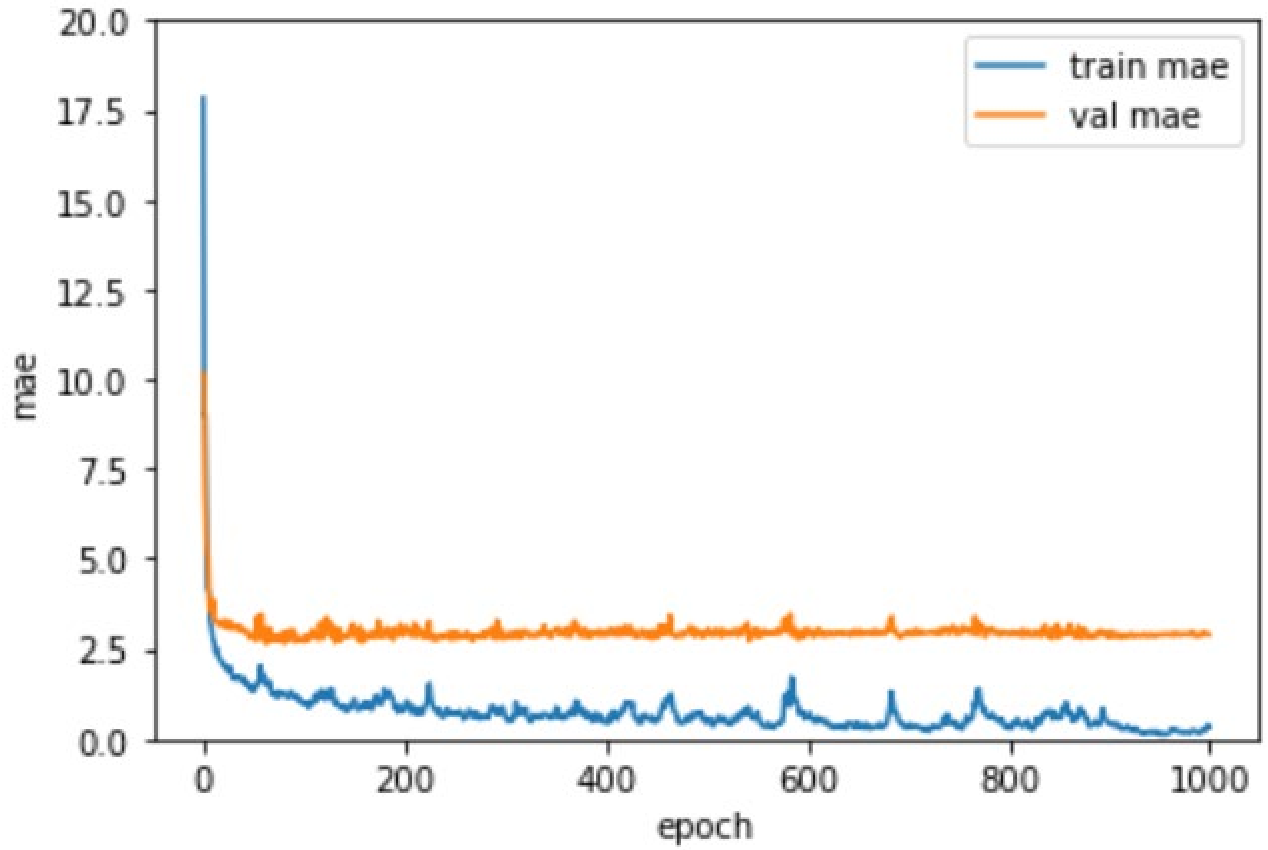


Learning History

```
Train on 303 samples, validate on 76 samples
Epoch 1/1000
303/303 [==============================] - 0s 179us/sample - loss: 0.1522 - mae: 0.2759 -
val_loss: 15.1493 - val_mae: 2.6273
Epoch 2/1000
303/303 [==============================] - 0s 119us/sample - loss: 0.1994 - mae: 0.3372 -
val_loss: 15.3335 - val_mae: 2.6674
Epoch 3/1000
303/303 [==============================] - 0s 116us/sample - loss: 0.3224 - mae: 0.4405 -
val_loss: 15.6130 - val_mae: 2.6099
Epoch 4/1000
303/303 [==============================] - 0s 157us/sample - loss: 0.5778 - mae: 0.5800 -
val_loss: 15.6360 - val_mae: 2.6161
   .
   .
   .
Epoch 1000/1000
303/303 [==============================] - 0s 75us/sample - loss: 0.0391 - mae: 0.1383 -
val_loss: 14.8579 - val_mae: 2.6602
```

Visualize Learning History

```
# Plot the learning history
plt.plot(history.history['mae’],
         label='train mae‘)
plt.plot(history.history['val_mae’],
         label='val mae‘)
plt.xlabel('epoch‘)
plt.ylabel('mae‘)
plt.legend(loc='best‘)
plt.ylim([0,20])
plt.show()
```



Model Evaluation

```
# Model evaluation
train_loss, train_mae = model.evaluate(X_train, y_train)
test_loss, test_mae = model.evaluate(X_test, y_test)
print('train loss:{:.3f}¥ntest loss: {:.3f}'.format(train_loss, test_loss))
print('train mae:{:.3f}¥ntest mae: {:.3f}'.format(train_mae, test_mae))
-----------------------------------------------------------------
train loss: 4.239
test loss: 10.772
train mae: 0.884
test mae: 2.293
```

Make Prediction

```
# Make prediction
y_pred = model.predict(X_test)
print(y_pred)
------------------------------------------------------------
[[28.521143 ]
 [19.221983 ]
 [26.441172 ]
 [19.683641 ]
 [15.493921 ]
     .....
 [17.98763  ]
 [14.546152 ]]
```

20. ANN Classification with Keras

Import Libraries

```python
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from tensorflow.keras.layers import Activation, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam

import warnings
warnings.filterwarnings('ignore')
```

Load and Prepare the Dataset

```
# Load dataset
from sklearn.datasets import load_breast_cancer
breast_cancer = load_breast_cancer()

# Create X and y
X = pd.DataFrame(breast_cancer.data)
y = breast_cancer.target

# Split the data into training and test set
X_train, X_test, y_train, y_test = train_test_split(breast_cancer.data, breast_cancer.target)

# Standardize the features
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Define Artificial Neural Network

```
# Define ANN
model = Sequential()
model.add(Dense(64, activation='relu’,
                        input_shape=(30,)))
model.add(Dense(32, activation='relu‘))
model.add(Dense(1, activation='sigmoid'))
```

Show the Model Summary

```
# Model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                1984
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 4,097
Trainable params: 4,097
Non-trainable params: 0
_________________________________________________________________
```

$W_0 \times W_1 + b_1$(bias)

$30 \times 64 + 64 = 1984$

$64 \times 32 + 32 = 2080$

$32 \times 1 + 1 = 33$

$1984 + 2080 + 33 = 4097$

Compile the Model

```
# Compile model
model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['acc'])
```

Fit the Model

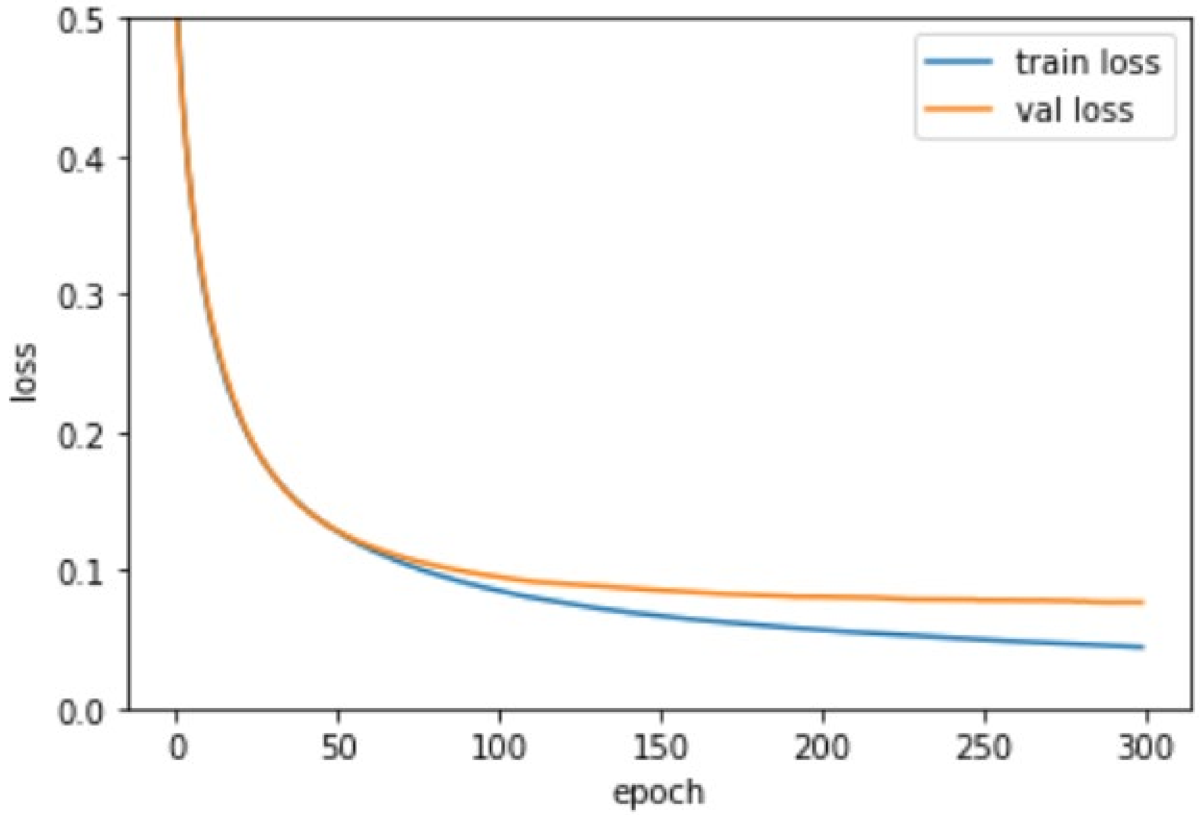
```python
# Fit the model
history = model.fit(X_train, y_train,
                    batch_size=64,
                    epochs=300,
                    validation_split=0.2)
```

Learning History

```
Train on 340 samples, validate on 86 samples
Epoch 1/300
340/340 [==============================] - 1s 2ms/sample - loss: 0.5206 - acc: 0.8441 -
val_loss: 0.5117 - val_acc: 0.8488
Epoch 2/300
340/340 [==============================] - 0s 58us/sample - loss: 0.4763 - acc: 0.8794 -
val_loss: 0.4714 - val_acc: 0.8605
Epoch 3/300
340/340 [==============================] - 0s 72us/sample - loss: 0.4401 - acc: 0.8941 -
val_loss: 0.4377 - val_acc: 0.8605
Epoch 4/300
340/340 [==============================] - 0s 73us/sample - loss: 0.4097 - acc: 0.9118 -
val_loss: 0.4103 - val_acc: 0.8721

 ..........

Epoch 300/300
340/340 [==============================] - 0s 97us/sample - loss: 0.0442 - acc: 0.9882 -
val_loss: 0.0766 - val_acc: 0.9767
```

Visualize Learning History

```
# Plot the learning history
plt.plot(history.history[‘loss‘],
          label='train loss‘)
plt.plot(history.history['val_loss‘],
          label='val loss‘)
plt.xlabel('epoch‘)
plt.ylabel('loss‘)
plt.legend(loc='best‘)
plt.ylim([0,0.5])
plt.show()
```



Model Evaluation

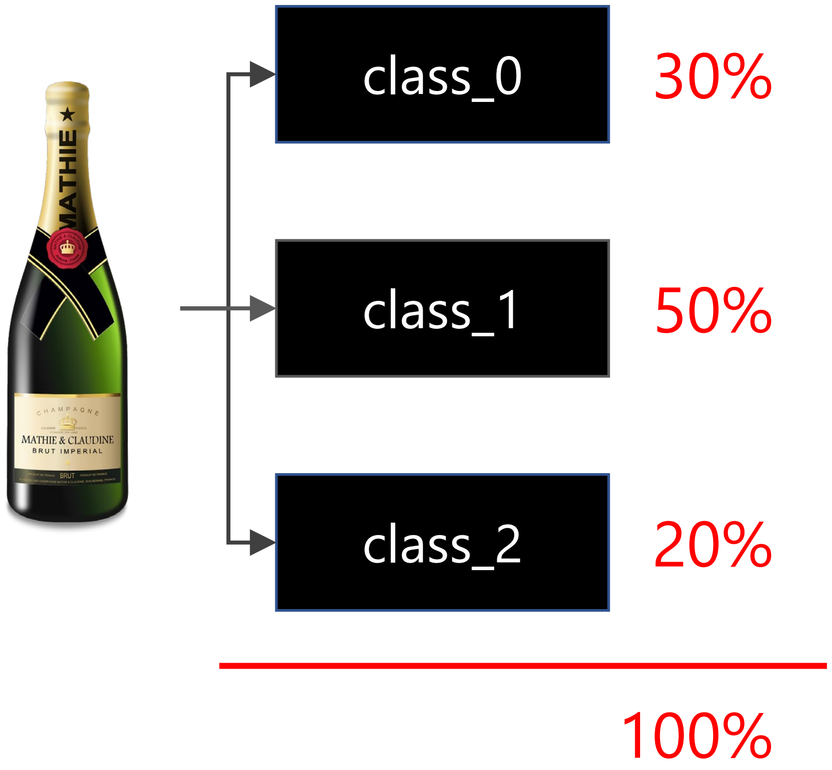
```
# Model evaluation
train_loss, train_acc = model.evaluate(X_train, y_train)
test_loss, test_acc  = model.evaluate(X_test, y_test)
print('train loss:{:.3f}¥ntest loss: {:.3f}'.format(train_loss, test_loss))
print('train acc:{:.3f}¥ntest acc: {:.3f}'.format(train_acc, test_acc))
--------------------------------------------------------------------------
426/426 [==============================] - 0s 30us/sample - loss: 0.0506 - acc: 0.9859
143/143 [==============================] - 0s 63us/sample - loss: 0.0649 - acc: 0.9790
train loss:0.051
test loss: 0.065
train acc:0.986
test acc: 0.979
```

Make Prediction

```
# Make prediction
y_pred = model.predict(X_test)
print(np.round(y_pred, 3))
-------------------------------------------------------------
[[0.993]
 [1.   ]
 [0.   ]
 [0.998]
 [1.   ]
    ....
 [0.996]
 [0.894]]
```

Multiclass Classification

```
# Load dataset
from sklearn.datasets import load_wine
data = load_wine()

# Display target names
list(data.target_names)
----------------------------------------
['class_0', 'class_1', 'class_2’]

# Store features and target
features = data.data
target = data.target
```



Multiclass Classification

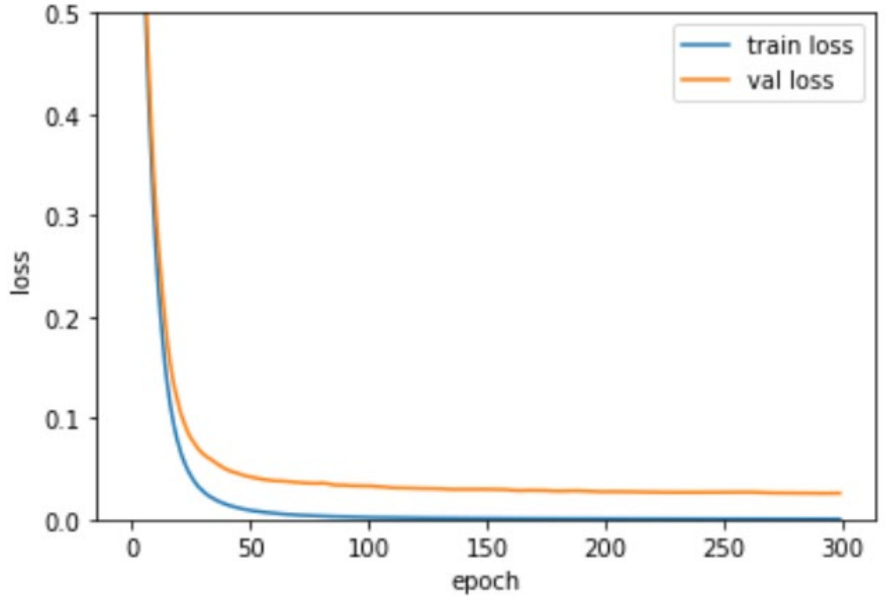
```
# Create X and y
X = pd.DataFrame(features)
y = target

# Split the data into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Standardize the features
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Shape of the data
X_train.shape
------------------------------------------------------------
 (133, 13)
```

Multiclass Classification

```
# Define ANN for multiclass classification
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(13,)))
model.add(Dense(32, activation='relu‘))
model.add(Dense(3, activation='softmax'))

# Compile model
model.compile(optimizer='adam‘,
              loss='sparse_categorical_crossentropy‘,
              metrics=['acc'])
```

Multiclass Classification

```
# Fit the model
history = model.fit(X_train, y_train, batch_size=32, epochs=300, validation_split=0.2)

# Plot the learning history
plt.plot(history.history['loss'], label='train loss‘)
plt.plot(history.history['val_loss'], label='val loss‘)
plt.xlabel('epoch‘)
plt.ylabel('loss‘)
plt.legend(loc='best‘)
plt.ylim([0,0.5])
plt.show()

# Model evaluation
train_loss, train_acc = model.evaluate(X_train, y_train)
test_loss, test_acc  = model.evaluate(X_test, y_test)
print('train loss:{:.3f}¥ntest loss: {:.3f}'.format(train_loss, test_loss))
print('train acc:{:.3f}¥ntest acc: {:.3f}'.format(train_acc, test_acc))
```

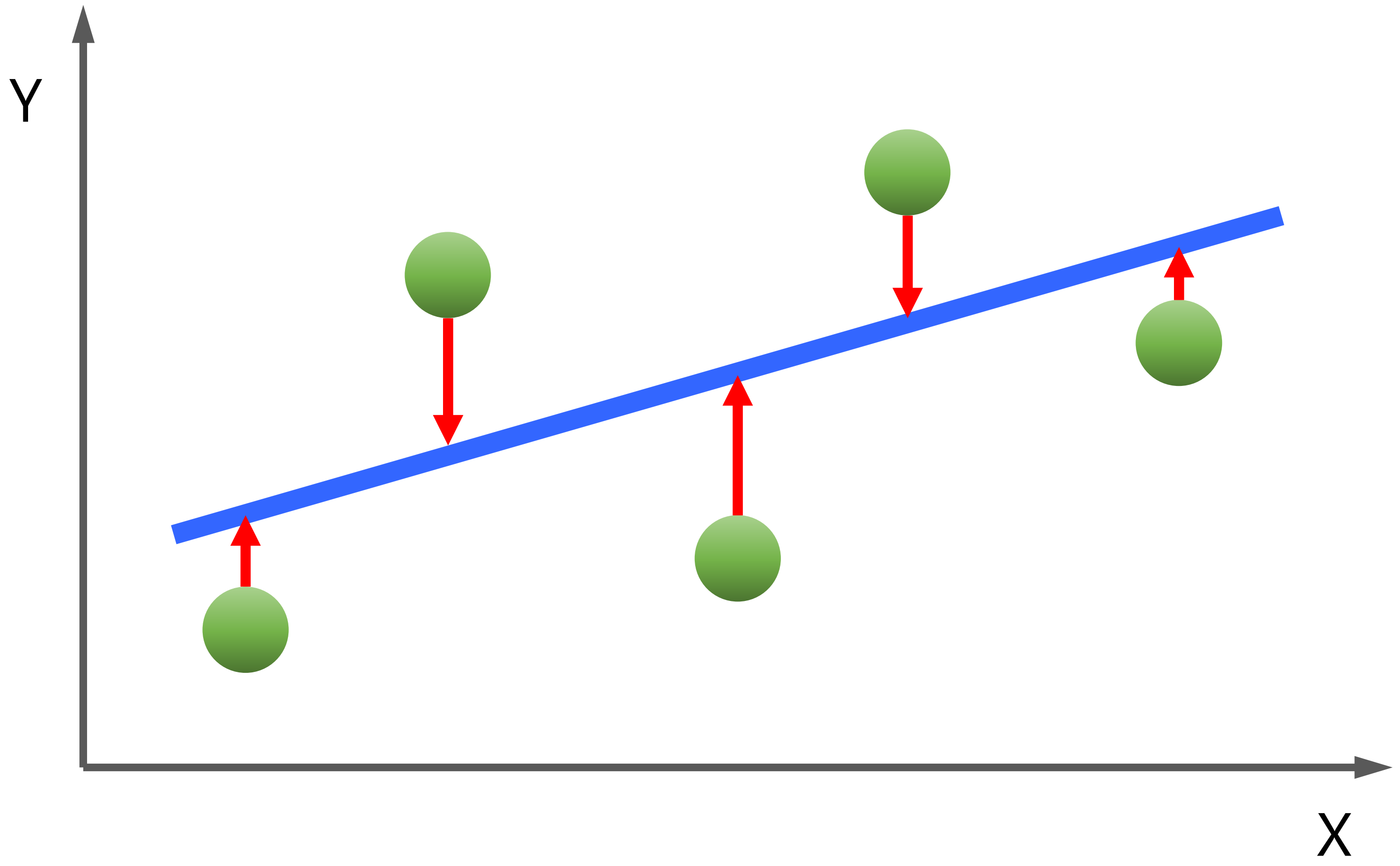


Multiclass Classification

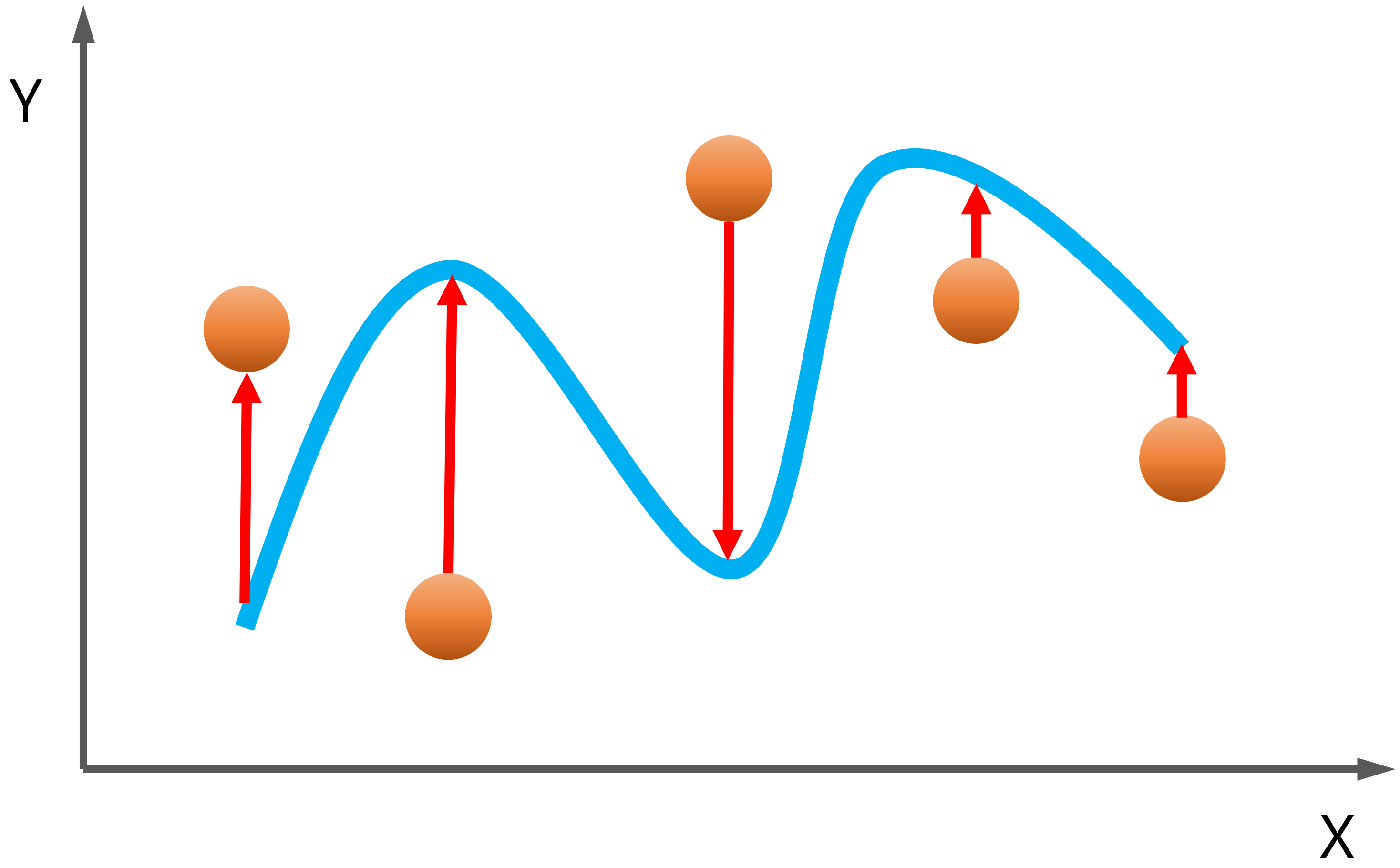
```
# Make prediction
y_pred = model.predict(X_test)
print(np.round(y_pred, 3))
-----------------------------------------
[[0.    1.    0.   ]
 [0.    1.    0.   ]
 [0.    1.    0.   ]
 [0.    1.    0.   ]
 [0.    1.    0.   ]
 [0.01  0.99  0.   ]
 [1.    0.    0.   ]
 [0.    1.    0.   ]
 [0.003 0.997 0.   ]
 [0.    1.    0.   ]
  .............
 [1.    0.    0.   ]
 [0.    0.    1.   ]
 [1.    0.    0.   ]]
```

21. Overfitting

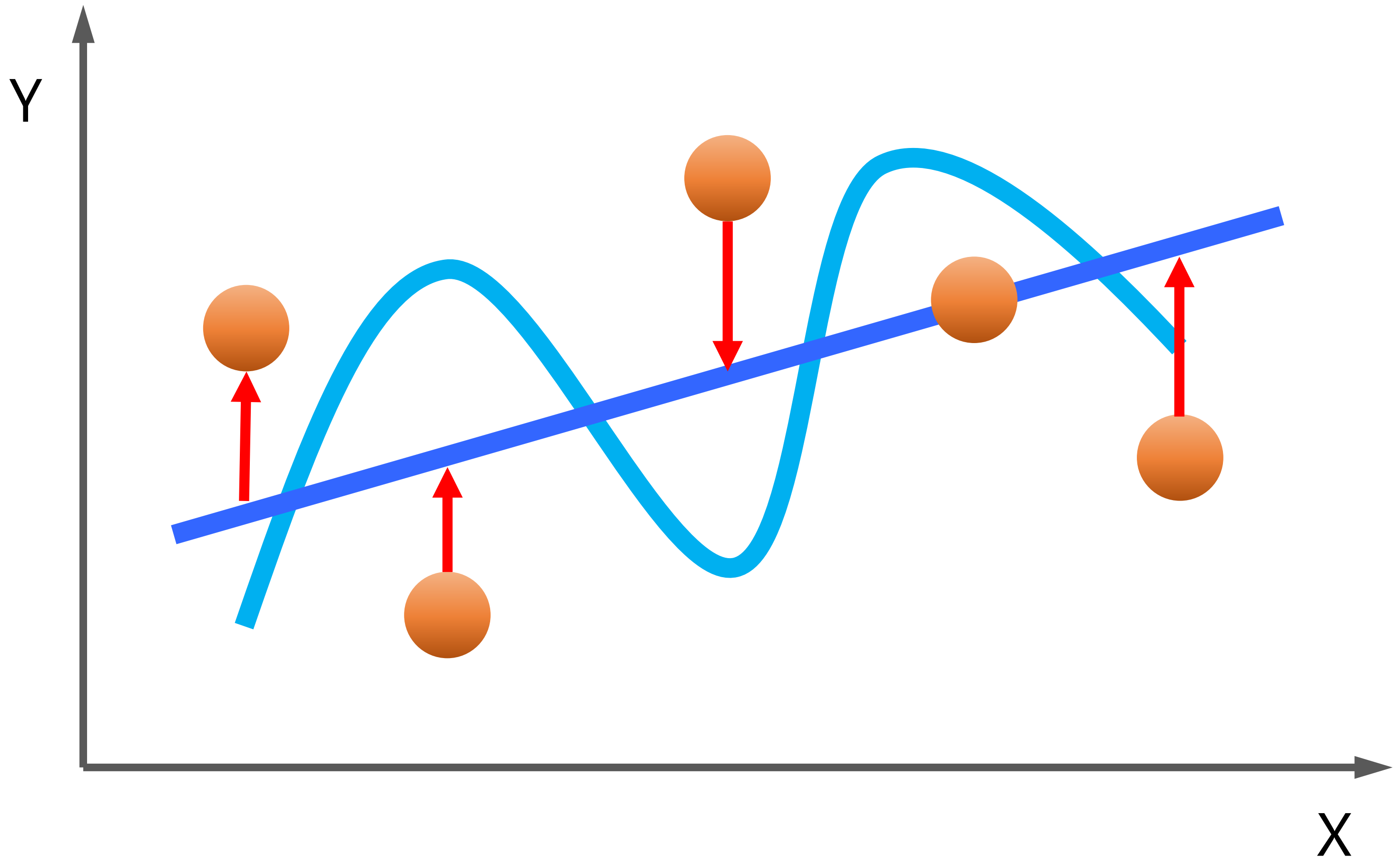
Model Generalizability



Model Generalizability



Model Generalizability

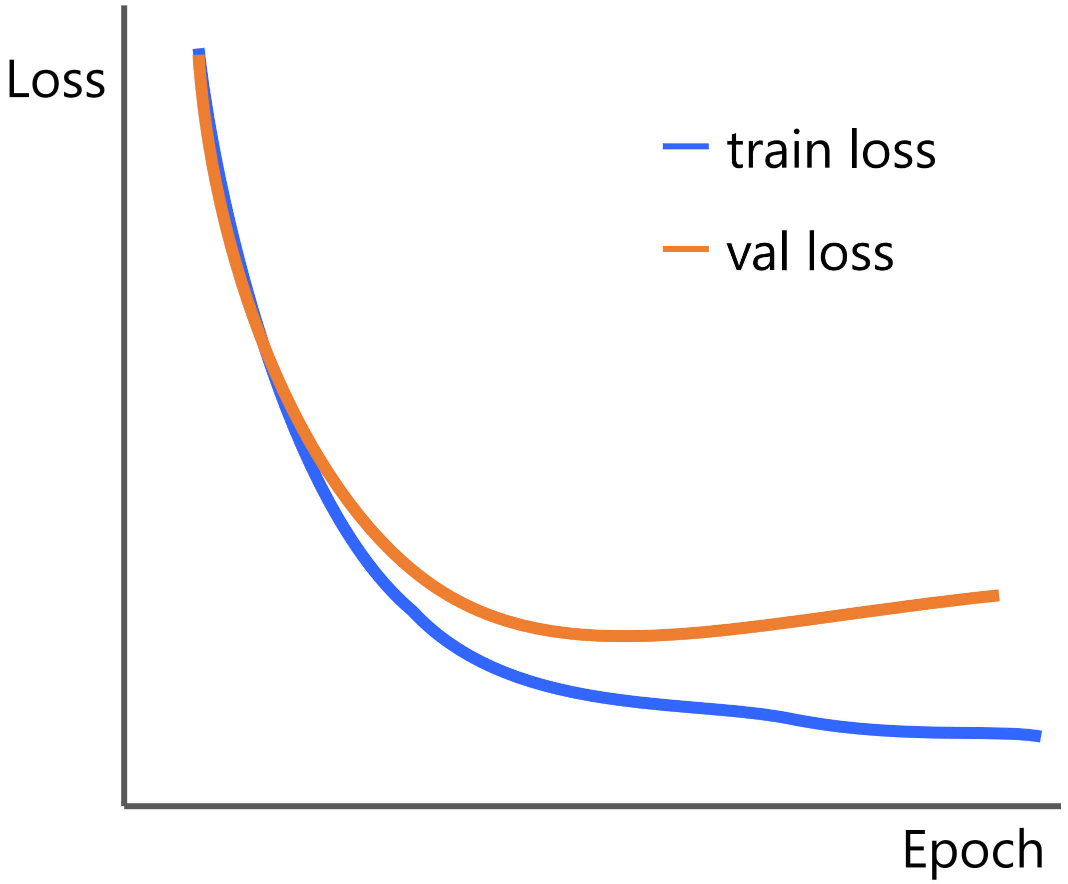


Overfitting

A model is optimized too closely for a particular dataset and fails to perform well for other datasets.

Since we develop a deep learning model to make predictions for unseen data, an overfitted model is useless.

We need to take measures to prevent overfitting.



How Can We Prevent Overfitting?

- Increase training data

- Regularization

- Early Stopping

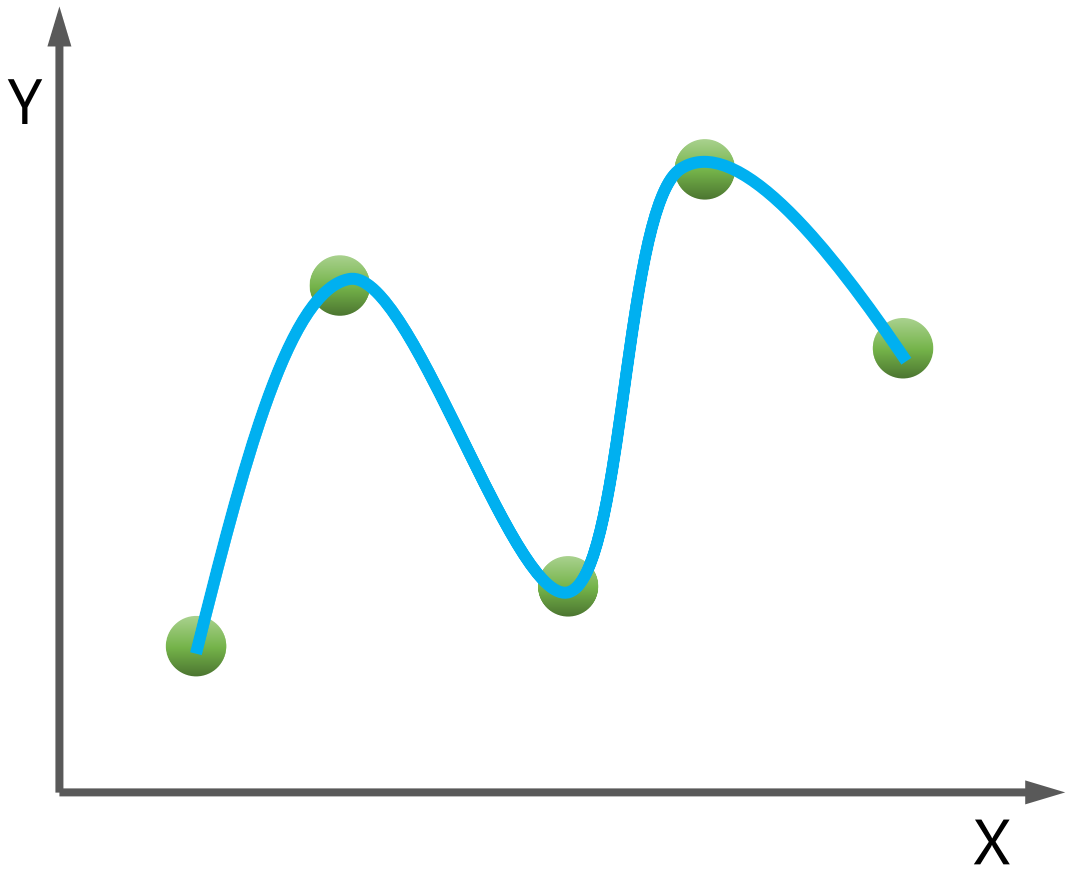
22. L1 & L2 Regularization

Regularization

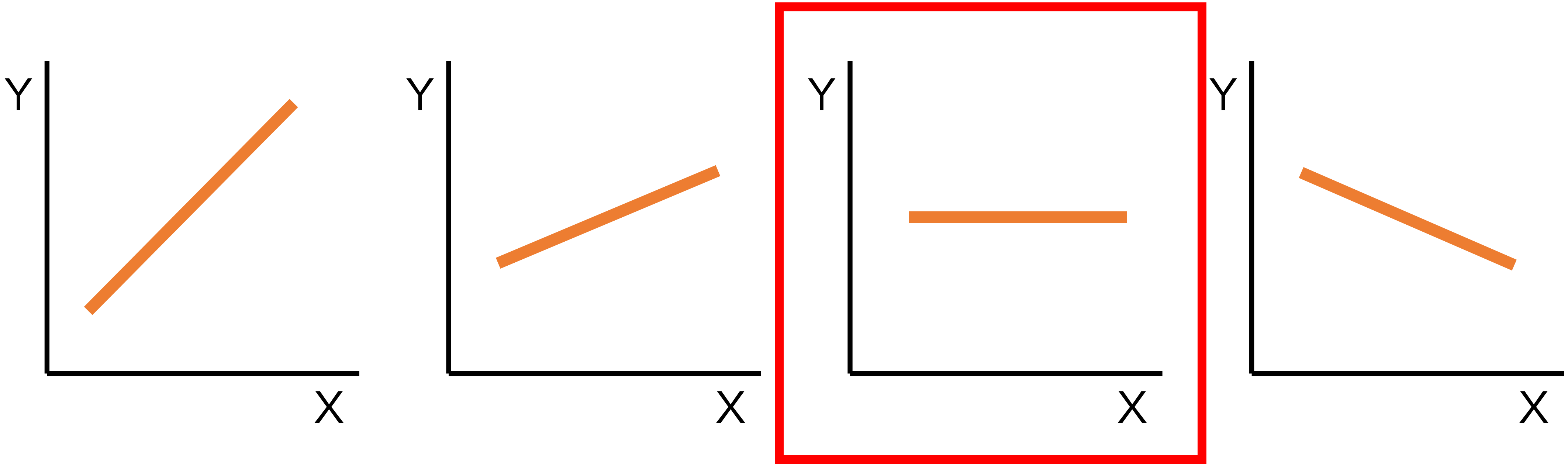
Too complex models are likely to be overfitted to the training data.

It lacks generalizability.

Regularization prevent overfitting by penalizing the complexity of the model.



Q. Which is a case where “X and Y are not related”?



$y = \beta x$

If $\beta = 0$, X is not related to Y.

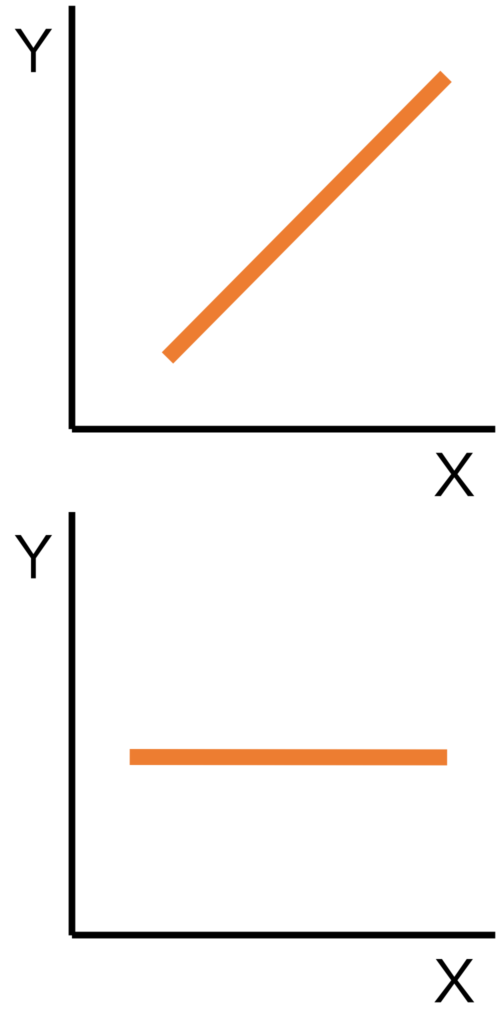
Statistical Hypothesis Testing in Regression Analysis

$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \varepsilon$$

Testing whether $\beta_i = 0$ or not

If $\beta_i \neq 0$

x_i is significantly related to y



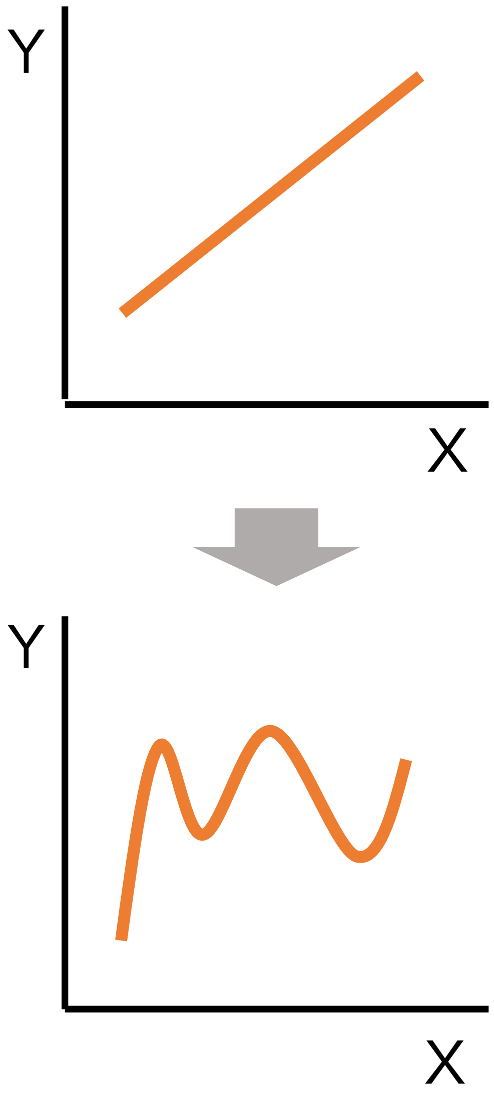
Model Complexity and Overfitting

$$y = \beta_1 x_1 + \varepsilon$$

$$y = \beta_1 x_1 + \beta_2 x_2 + \varepsilon$$

$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \varepsilon$$

$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \varepsilon$$

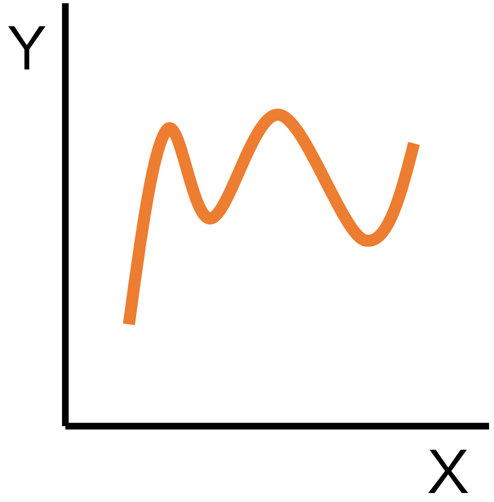


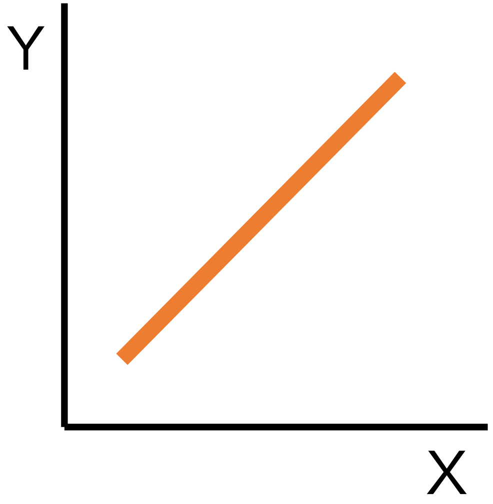
The more complex a model is, the more likely it falls into overfitting.

Regularization

$$y = \beta_1 x_1 + \underbrace{\beta_2}_{0} x_2 + \underbrace{\beta_3}_{0} x_3 + \underbrace{\beta_4}_{0} x_4 + \varepsilon$$

$$= \beta_1 x_1 + \varepsilon$$





Regularization forces the weights of uninformative features to be zero or nearly zero.

By doing so, regularization simplifies the model, and thus, prevent overfitting.

L1 & L2 Regularization

- L1 regularization: Lasso Regression
- L2 regularization: Ridge Regression

The key difference is the regularization term.

Regularization term penalizes the model when some features' weights get larger.

Lasso Regression (L1 Regularization)

Ordinary regression analysis

$$Cost = Residual\ Sum\ of\ Error = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Lasso Regression (Least Absolute Shrinkage and Selection Operator)

$$Cost = RSS + \underbrace{\lambda \sum_{j=0}^{M} |w_j|}_{\text{Regularization term}}$$

Lasso shrinks the less important features' weights (coefficient) to 0.

Ridge Regression (L2 Regularization)

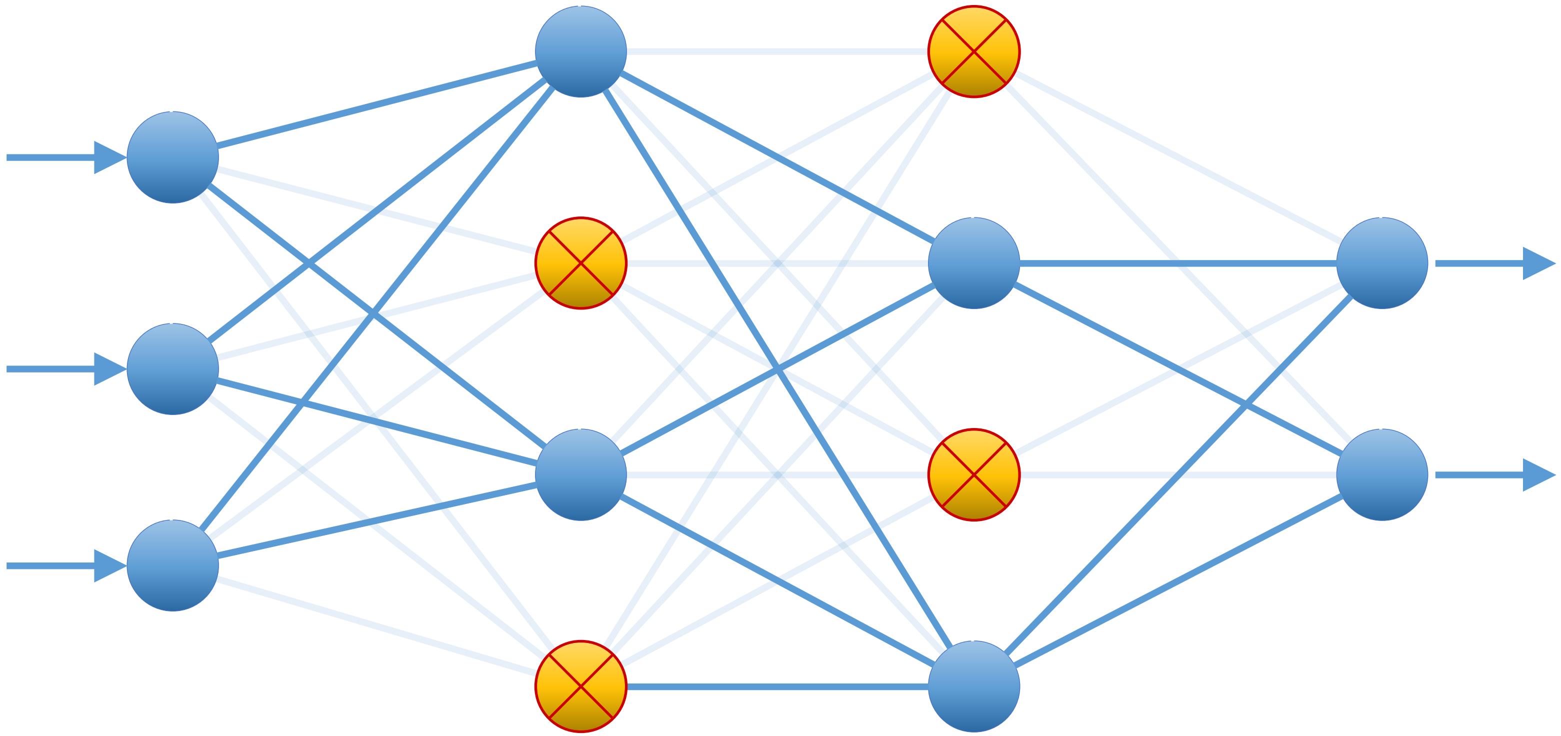
Ridge regression

$$Cost = RSS + \lambda \sum_{j=0}^{M} w_j^2$$

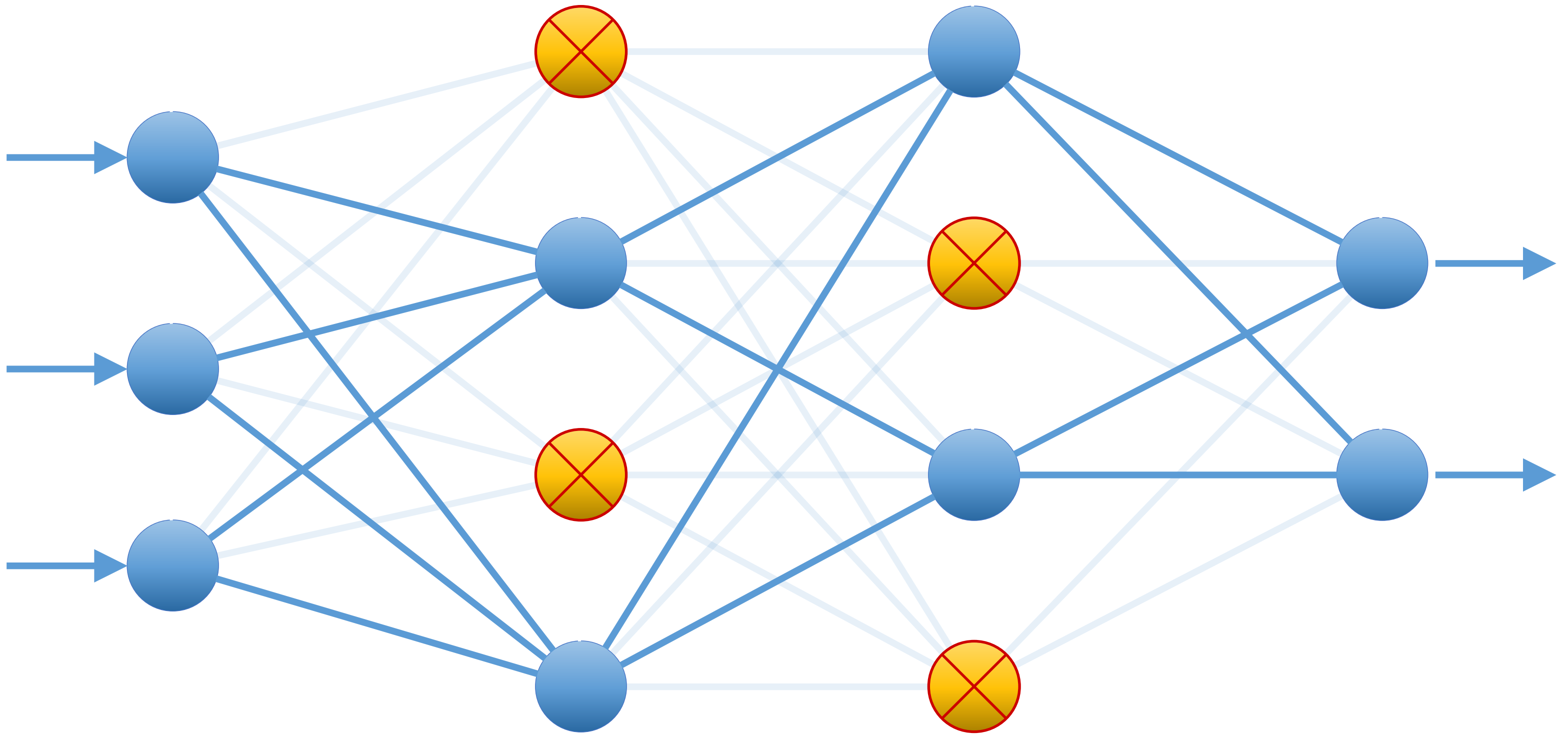
Ridge makes less important features' weights smaller.

23. Dropout

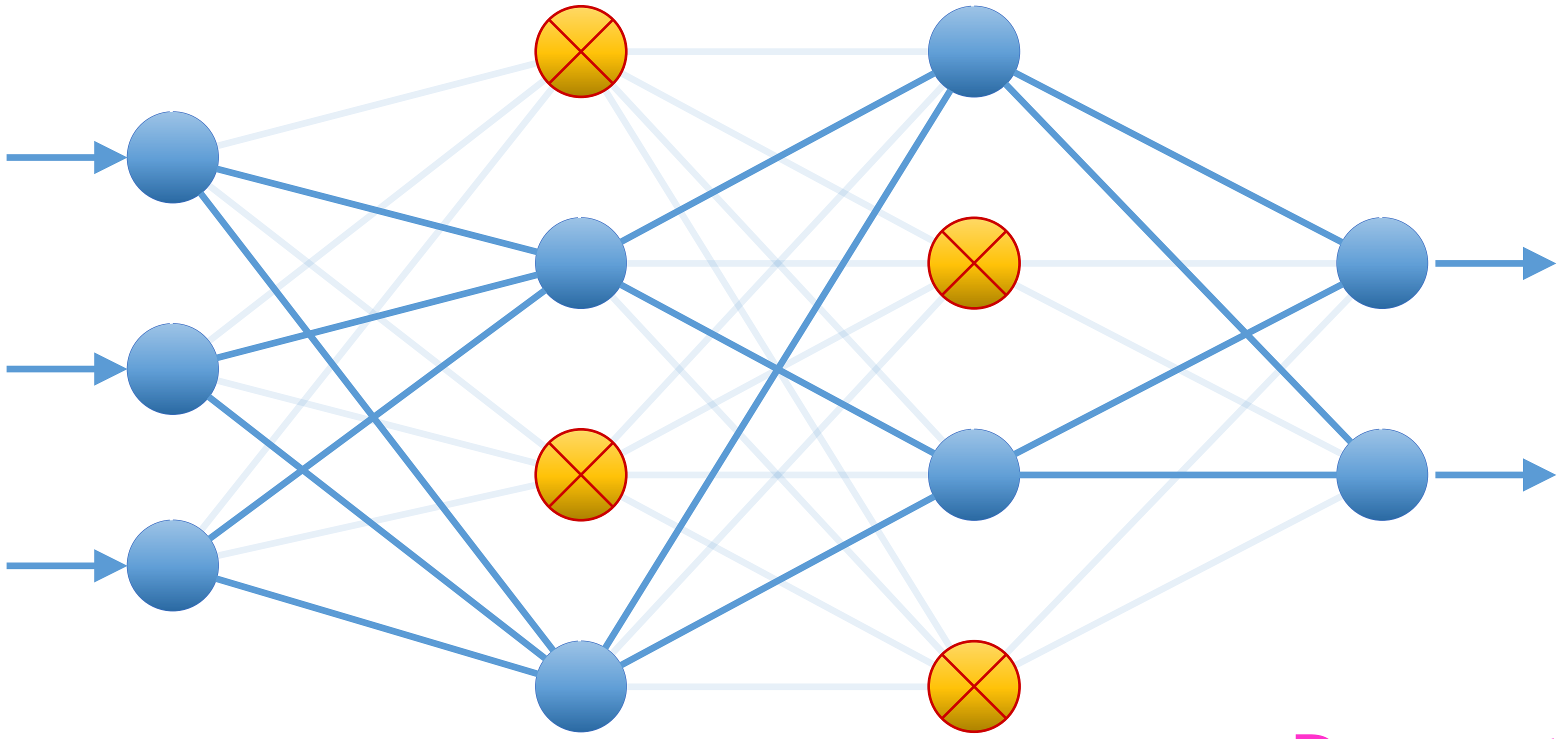
Dropout



Dropout

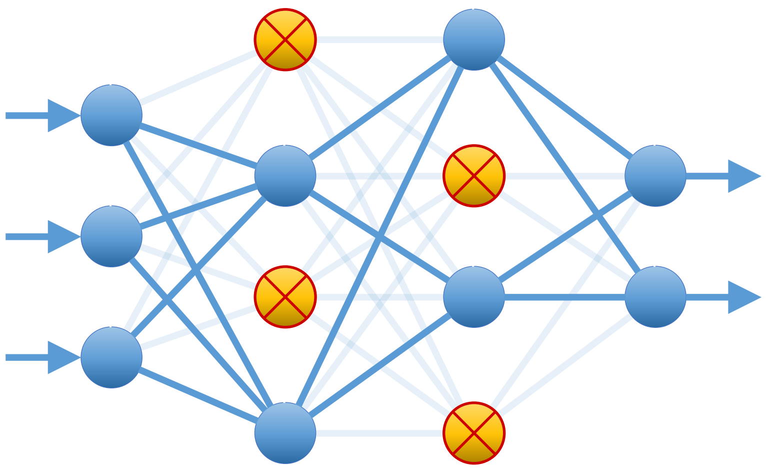


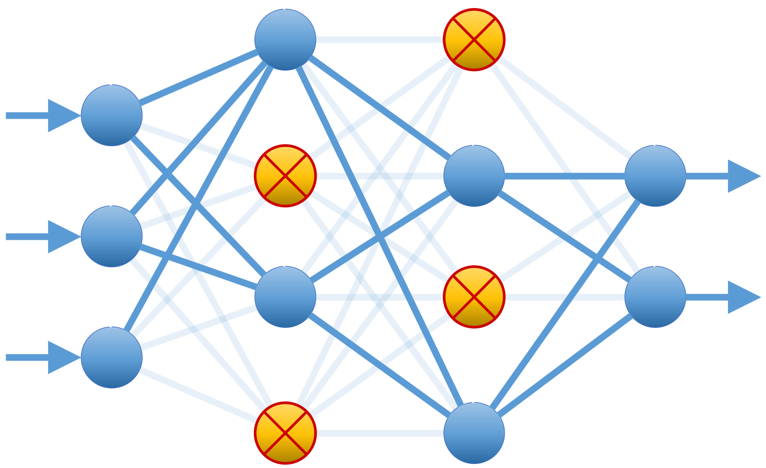
Dropout

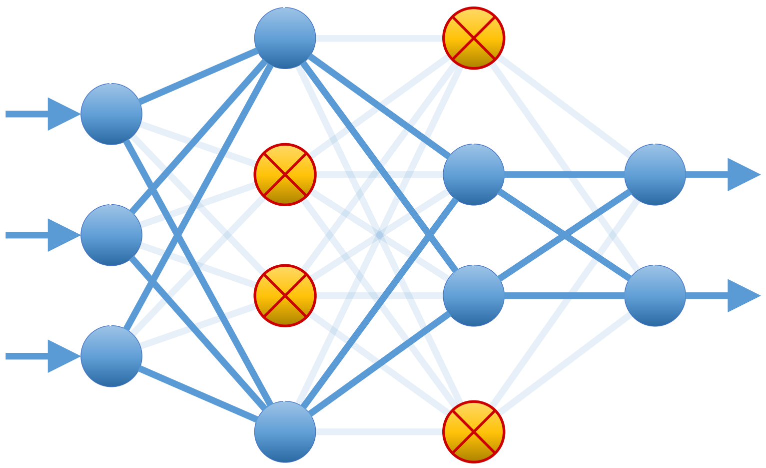


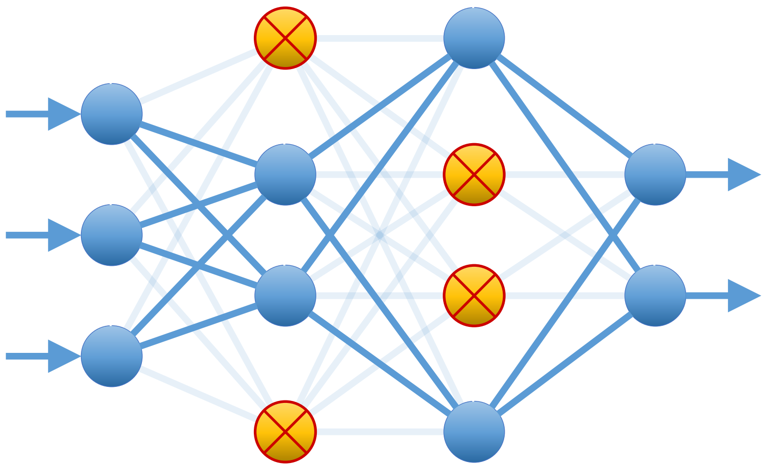
Dropout Rate = 0.5

Dropout and Generalizability









24. Regularization with Keras

Import Libraries

```
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from tensorflow.keras.layers import Activation, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam

import warnings
warnings.filterwarnings('ignore')
```

Load and Prepare the Dataset

```
# Load dataset
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="¥s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]

# Create X and y
X = pd.DataFrame(data)
y = target

# Split data into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Standardize the features
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

L1 Regularization

```
# Import library for L1 regularization
from tensorflow import keras

# Define ANN with L1 regularization
model_l1 = Sequential()
model_l1.add(Dense(128, activation='relu', input_shape=(13,),
                   kernel_regularizer=keras.regularizers.l1(0.01)))
model_l1.add(Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l1(0.01)))
model_l1.add(Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l1(0.01)))
model_l1.add(Dense(32, activation='relu', kernel_regularizer=keras.regularizers.l1(0.01)))
model_l1.add(Dense(1))

# Compile the model
model_l1.compile(loss='mse',
                 optimizer=Adam(lr=0.01),
                 metrics=['mae'])
```

Fit and Predict with L1 Regularization

```
# Fit the model
history_l1 = model_l1.fit(X_train, y_train,
                          batch_size=64,
                          epochs=1000,
                          validation_split=0.2,
                          verbose=0)

# Model evaluation
train_loss_l1, train_mae_l1 = model_l1.evaluate(X_train, y_train)
test_loss_l1, test_mae_l1 = model_l1.evaluate(X_test, y_test)
print('train loss:{:.3f}¥ntest loss: {:.3f}'.format(train_loss_l1, test_loss_l1))
print('train mae:{:.3f}¥ntest mae: {:.3f}'.format(train_mae_l1, test_mae_l1))
------------------------------------------------------------------------
train loss:5.964
test loss: 9.608
train mae:1.132
test mae: 2.114
```

L2 Regularization

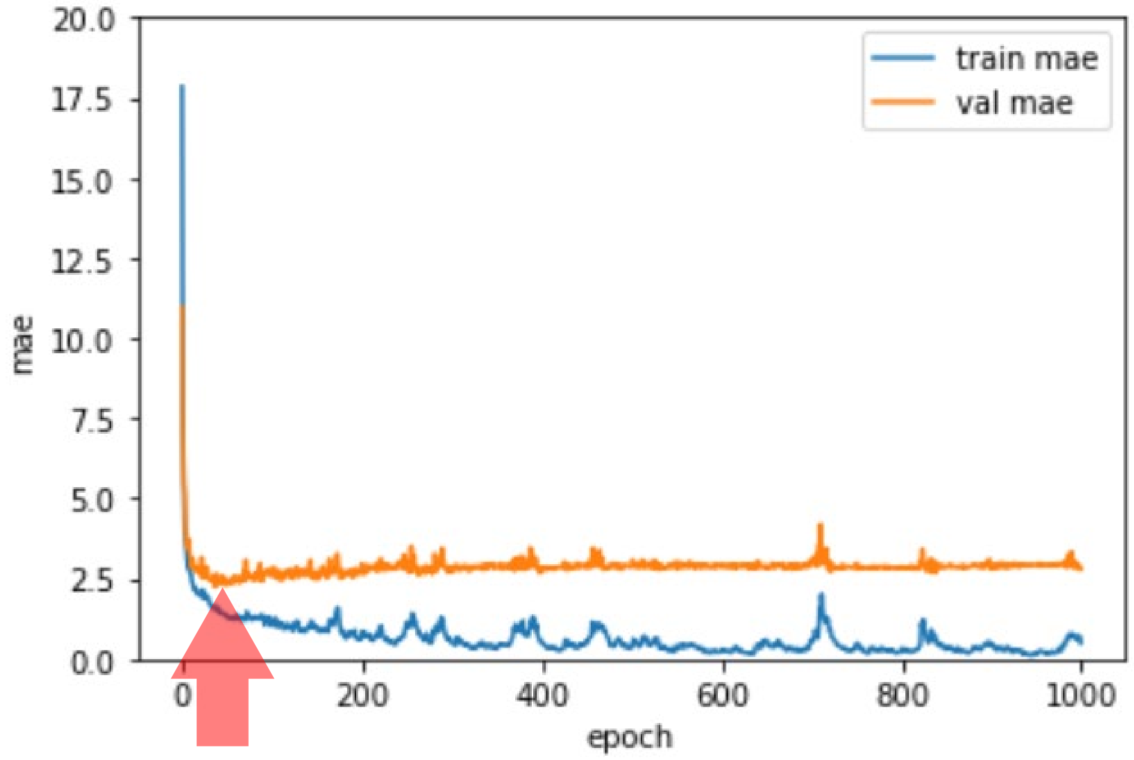
```
# Define ANN with L2 regularization
model_l2 = Sequential()
model_l2.add(Dense(128, activation='relu', input_shape=(13,),
                   kernel_regularizer=keras.regularizers.l2(0.01)))
model_l2.add(Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l2(0.01)))
model_l2.add(Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l2(0.01)))
model_l2.add(Dense(32, activation='relu', kernel_regularizer=keras.regularizers.l2(0.01)))
model_l2.add(Dense(1))

# Compile the model
model_l2.compile(loss='mse',
                 optimizer=Adam(lr=0.01),
                 metrics=['mae'])
```

Dropout

```
# Import library for Dropout
from tensorflow.keras.layers import Dropout

# Define ANN with Dropout (Dropout rate = 0.5)
model_d = Sequential()
model_d.add(Dense(128, activation='relu', input_shape=(13,)))
model_d.add(Dropout(0.5))
model_d.add(Dense(64, activation='relu’))
model_d.add(Dropout(0.5))
model_d.add(Dense(64, activation='relu’))
model_d.add(Dropout(0.5))
model_d.add(Dense(32, activation='relu’))
model_d.add(Dropout(0.5))
model_d.add(Dense(1))

# Compile the model
model_d.compile(loss='mse’, optimizer=Adam(lr=0.01), metrics=['mae'])
```

Early Stopping

A regularization method that stops training when parameter updates no longer improve the model performance.



Early Stopping with Keras

```
# Import library for earlystopping
from tensorflow.keras.callbacks import EarlyStopping

# Define ANN
model_e = Sequential()
model_e.add(Dense(128, activation='relu', input_shape=(13,)))
model_e.add(Dense(64, activation='relu‘))
model_e.add(Dense(64, activation='relu‘))
model_e.add(Dense(32, activation='relu‘))
model_e.add(Dense(1))

# Compile the model
model_e.compile(loss=‘mse‘, optimizer=Adam(lr=0.01), metrics=['mae'])

# Set EarlyStopping
early_stop = EarlyStopping(monitor='val_loss', patience=30)
```

Early Stopping with Keras

```
# Fit the model
history_e = model_e.fit(X_train, y_train,
                        batch_size=64,
                        epochs=1000,
                        validation_split=0.2,
                        callbacks=[early_stop],
                        verbose=1)
```

Model Performance

	Train MAE	Test MAE
No Regularization	2.333	2.840
L1 Regularization	1.132	2.114
L2 Regularization	1.008	2.079
Dropout	2.912	2.802
Early Stopping	2.912	2.054

25. Optimizer

Optimizers

Algorithms used to find;

- The minimum value of the loss function.
- The parameter value that minimizes the loss function.

```python
# Compile the model
model.compile(loss='mse',
              optimizer=Adam(lr=0.01),
              metrics=['mae'])
```

e.g.,

- SGD
- Momentum
- AdaGrad
- RMSProp
- Adam   etc.

SGD (Stochastic Gradient Descent)

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

W: Parameter

η: Learning rate

$\frac{\partial L}{\partial W}$: Gradient

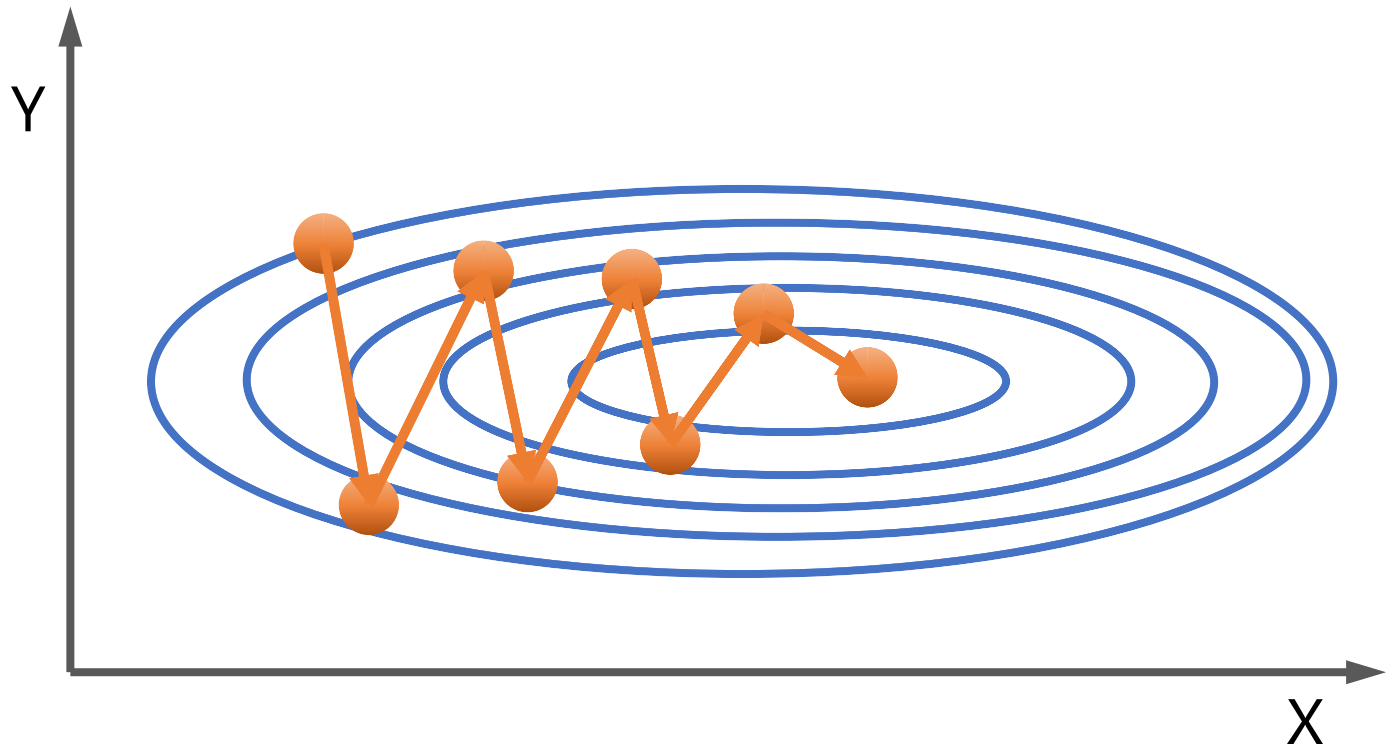
SGD (Stochastic Gradient Descent)

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

W: Parameter

η: Learning rate

$\frac{\partial L}{\partial W}$: Gradient

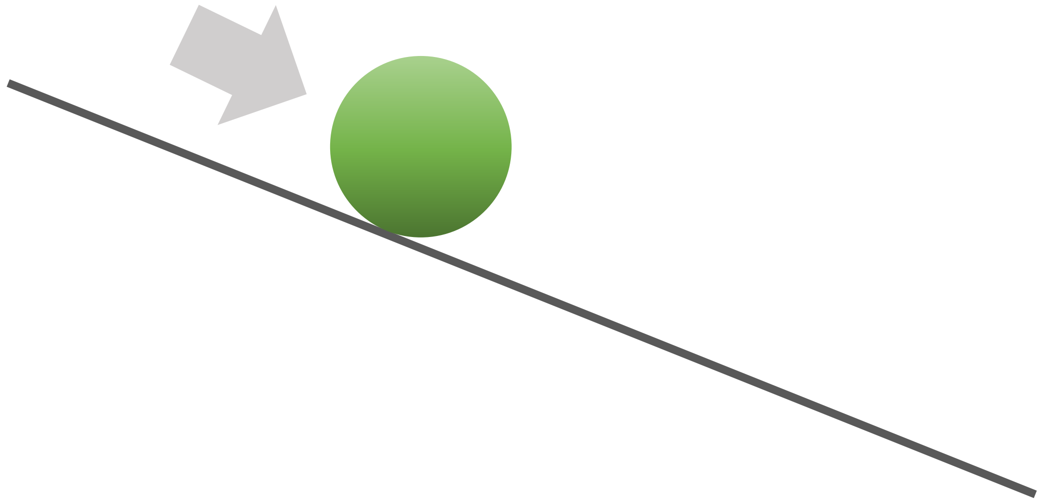


Momentum

$$v \leftarrow \alpha v - \eta \frac{\partial L}{\partial W}$$

$$W \leftarrow W + v$$

v: Velocity

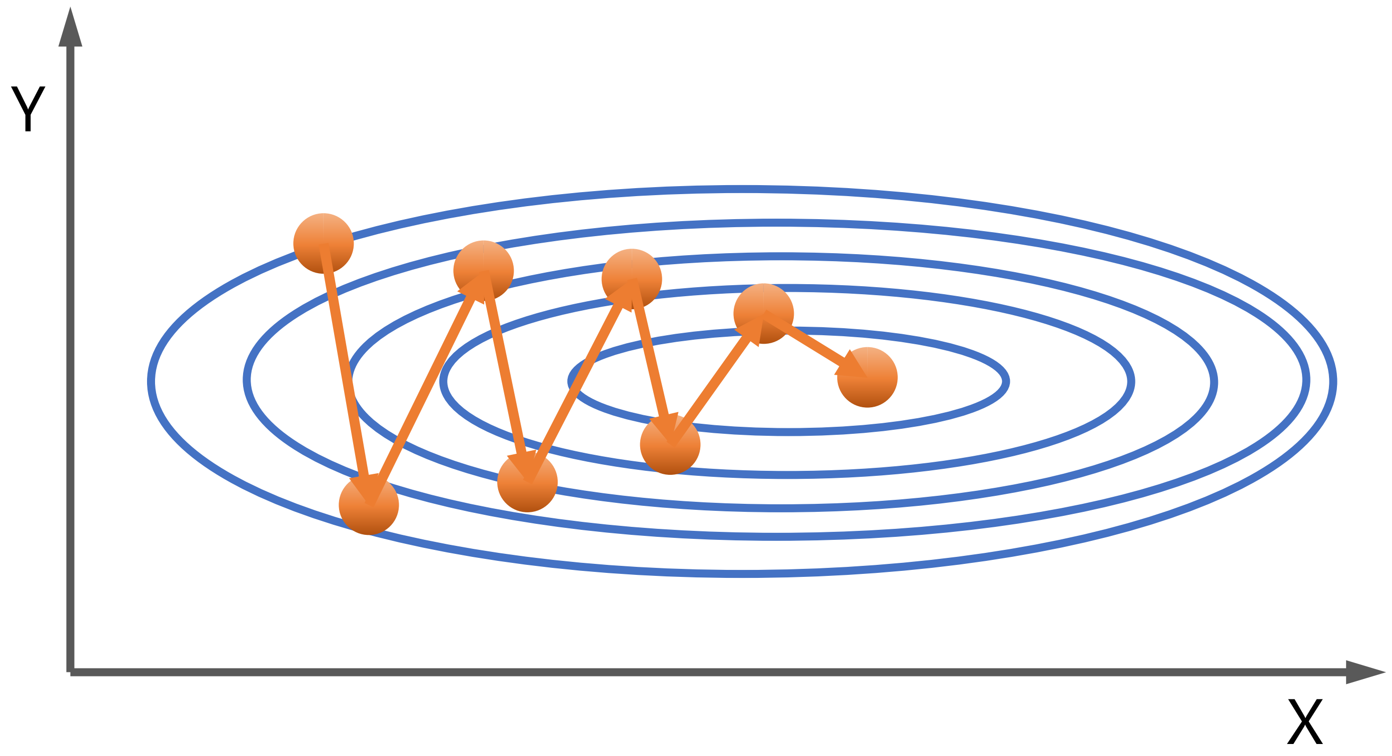


Momentum

$$v \leftarrow \alpha v - \eta \frac{\partial L}{\partial W}$$

$$W \leftarrow W + v$$

v: Velocity

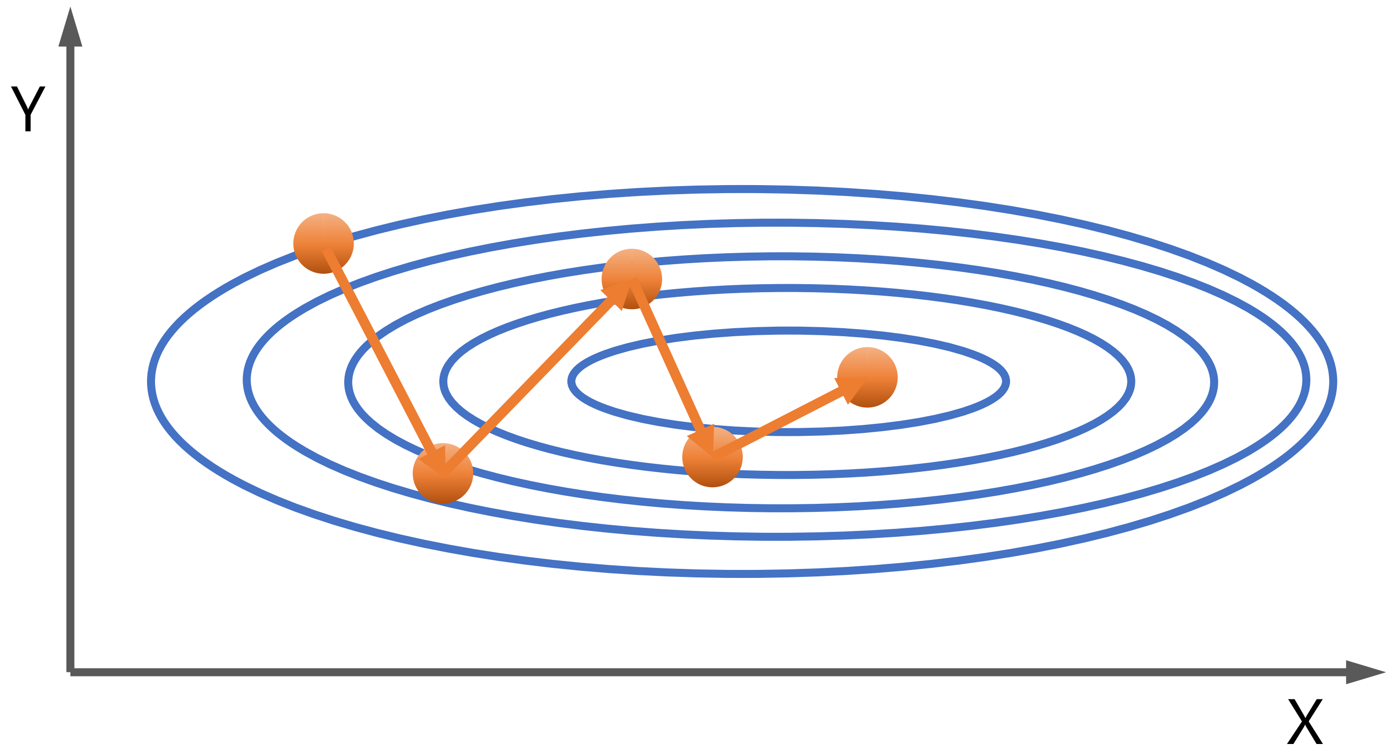


Momentum

$$v \leftarrow \alpha v - \eta \frac{\partial L}{\partial W}$$

$$W \leftarrow W + v$$

v: Velocity



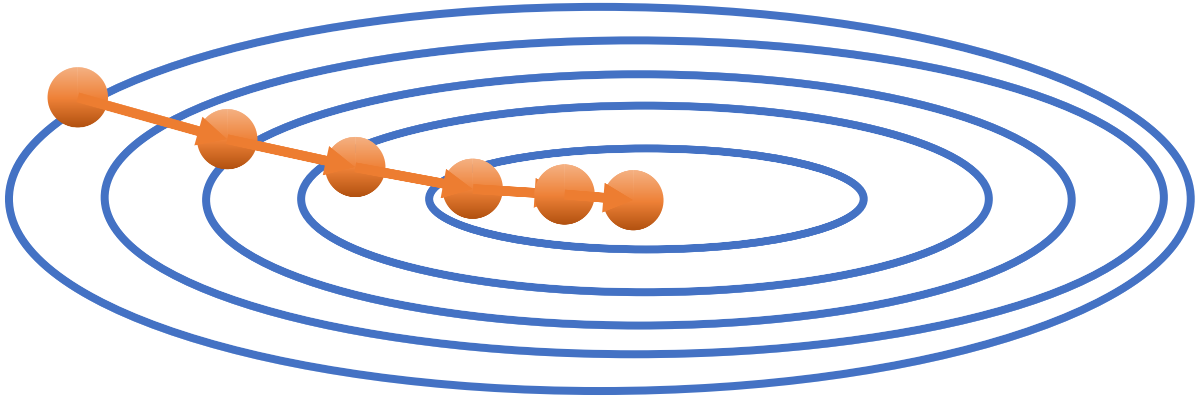
AdaGrad

Learning rate decay: Set a high learning rate at the outset, and gradually lower it.

AdaGrad does it for each parameter adaptively.

$$h \leftarrow h + \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{h + \varepsilon}} \frac{\partial L}{\partial W}$$



RMSProp

AdaGrad tends to lower the learning rate too fast, and thus, the learning rate becomes 0 too early.

RMSPROP calculates the sum of squares of the gradients, but use only the gradients from the recent iterations.

$$h \leftarrow \beta h + (1-\beta)\frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{h+\varepsilon}} \frac{\partial L}{\partial W}$$

β: Decay rate

$= 0.9$

Adam

$$m \leftarrow \beta_1 m + (1-\beta_1)\frac{\partial L}{\partial W}$$

m: Velocity, L: Loss, W: parameter, $\frac{\partial L}{\partial W}$: gradient

β_1: Decay rate

$$v \leftarrow \beta_2 v + (1-\beta_2)\frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$

v: Velocity, β_2: Decay rate

$$\widehat{m} = \frac{m}{1-\beta_1^t} \quad , \quad \hat{v} = \frac{v}{1-\beta_2^t}$$

$$W \leftarrow -\eta \frac{\widehat{m}}{\sqrt{\hat{v}} + \varepsilon}$$

26. Batch Normalization

Recap

- Regularization

  e.g., L1 & L2 regularization, dropout

- Optimizer

  e.g., AdaGrad, RMSprop, Adam

- Parameter Initialization

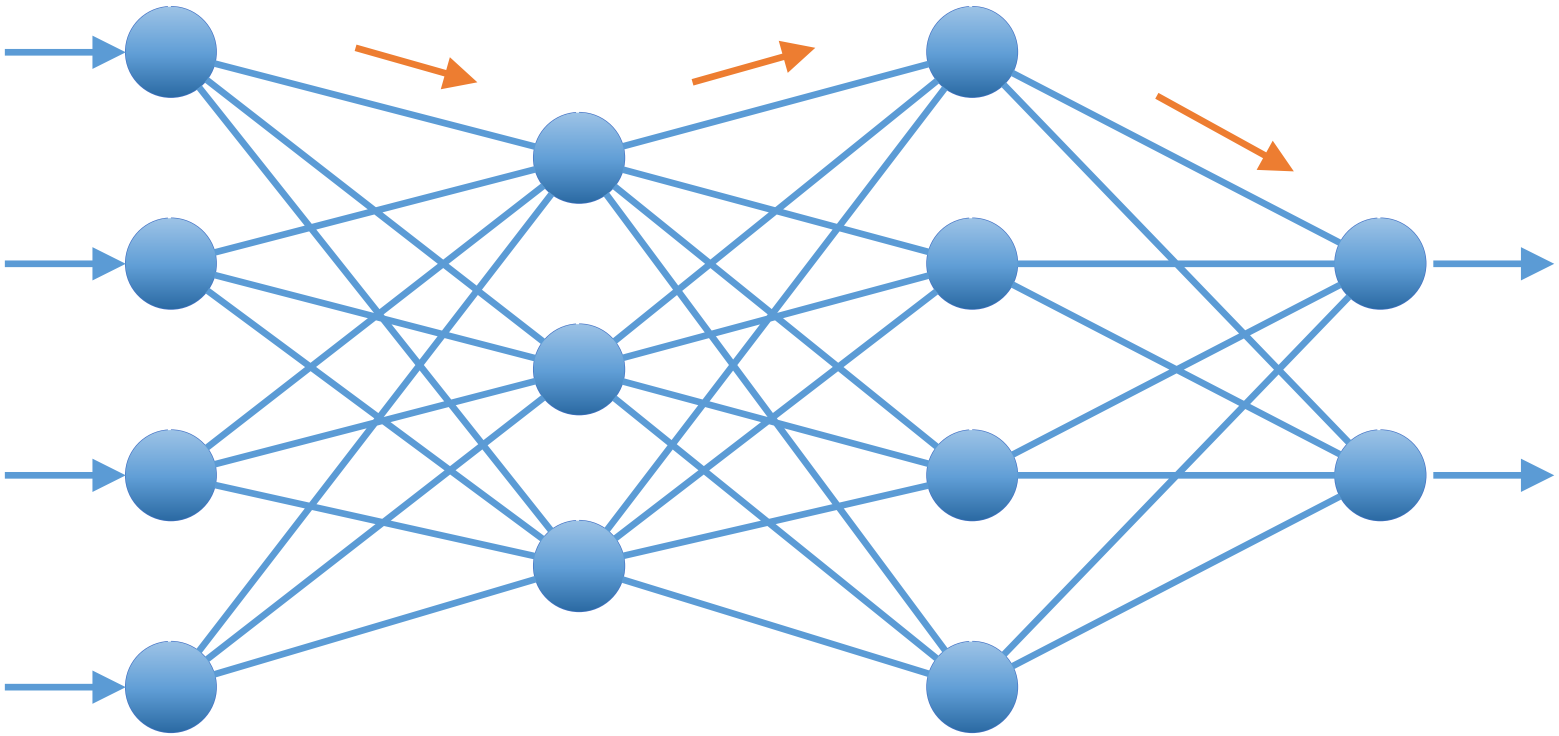
  e.g., Xavier initialization, He initialization

Normalization

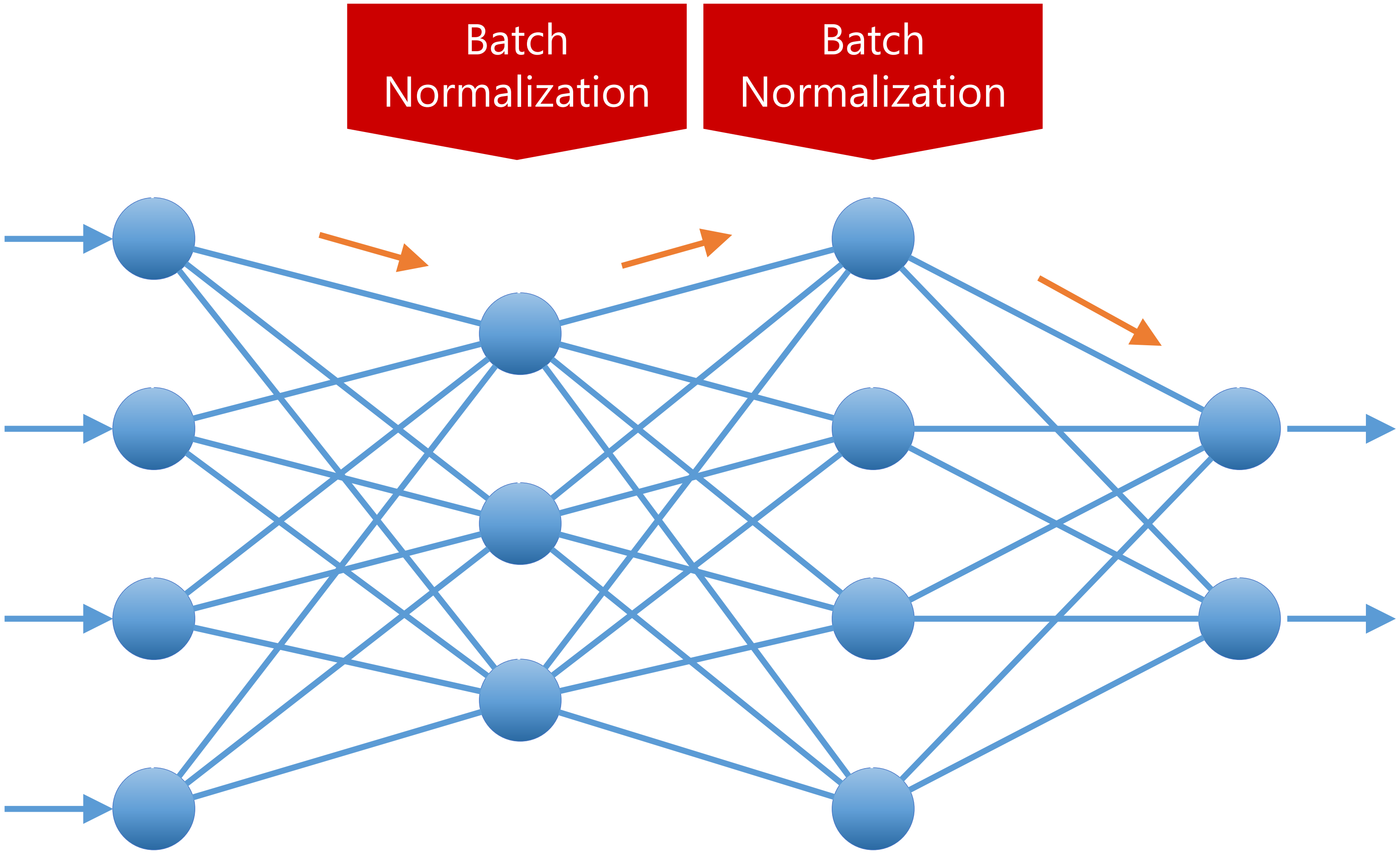
$$x_i^{new} = \frac{x_i - \mu}{\sigma}$$

$$0 \leq x' \leq 1$$

Weakness of Input Data Normalization



Batch Normalization



Batch Normalization: Equations

Mean: $$\mu_b \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

Variance: $$\sigma_b^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_b)^2$$

Standardization: $$\hat{x}_i \leftarrow \frac{x_i - \mu_b}{\sqrt{\sigma_b^2 + \varepsilon}}$$

27. Optimization & Batch Normalization with Keras

Import Libraries

```
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from tensorflow.keras.layers import Activation, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import SGD, Adagrad, RMSprop, Adam
from keras.layers.normalization import BatchNormalization

import warnings
warnings.filterwarnings('ignore')
```

Load and Prepare Dataset

```
# Load dataset
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="¥s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]

# Create X and y
X = pd.DataFrame(data)
y = target

# Split data into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Standardize the features
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Define Artificial Neural Network

```
# Define ANN
def ann():
    model = Sequential()
    model.add(Dense(128, activation='relu', input_shape=(13,)))
    model.add(BatchNormalization())
    model.add(Dense(64, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dense(64, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dense(32, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dense(1))
    model.compile(loss='mse',
                  optimizer=optimizer,
                  metrics=['mae'])
    return model
```

Set Optimizer

```
# Momentum
optimizer=SGD(lr=0.001, momentum=0.9)

# Adagrad
optimizer=Adagrad(lr=0.001, epsilon=10**-10)

# RMSprop
optimizer=RMSprop(lr=0.001, rho=0.9)

# Adam
optimizer=Adam(lr=0.001, beta_1=0.9, beta_2=0.9)
```

Fit and Evaluate the Model

```
# Fit the model
history = model.fit(X_train, y_train, batch_size=64, epochs=1000, validation_split=0.2)

# Plot the learning history
plt.plot(history.history['mae'], label='train mae‘)
plt.plot(history.history['val_mae'], label='val mae‘)
plt.xlabel(‘epoch‘)
plt.ylabel(‘mae‘)
plt.legend(loc='best‘)
plt.ylim([0,20])
plt.show()

# Model evaluation
train_loss, train_mae = model.evaluate(X_train, y_train)
test_loss, test_mae = model.evaluate(X_test, y_test)
print('train loss:{:.3f}¥ntest loss: {:.3f}'.format(train_loss, test_loss))
print('train mae:{:.3f}¥ntest mae: {:.3f}'.format(train_mae, test_mae))
```

Save and Load the Model

```python
# Save model
model.save("my_ann_model.h5")

# Load model
model = keras.models.load_model("my_ann_model.h5")
```

Save the Best Model

```python
# Save the best model only

# Import ModelCheckpoint
from keras.callbacks import ModelCheckpoint

# Set model checkpoint
model_checkpoint =ModelCheckpoint("my_ann_model_2.h5",
                                  save_best_only=True)

# Fit and save model
history = model.fit(X_train, y_train,
                    batch_size=64,
                    epochs=1000,
                    validation_split=0.2,
                    callbacks=[model_checkpoint])
```