Deep Learning

Tensorflow 2.15.x - Keras

Tensorflow

- **What is Tensorflow?**
  - TensorFlow is an end-to-end **open source platform for machine learning**. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications. (excerpted from https://www.tensorflow.org/)
  - Mainly **for Neural Network model development**

- **Tensorflow 2.15.x**
  - https://www.tensorflow.org/
  - https://www.tensorflow.org/install

Keras

- **What is Keras?**
  - Keras is **a deep learning API** written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on **enabling fast experimentation**. Being able to go from idea to result as fast as possible is key to doing good research.
  - Keras is the **high-level API of TensorFlow 2**: an approachable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning. It provides essential **abstractions and building blocks for developing and shipping machine learning solutions** with high iteration velocity. (excerpted from https://keras.io/about/).

- **Keras**
  - https://keras.io/

Tensorflow, Keras에 관해서 수업에서 다룰 내용들

- **Tensorflow, keras의 기초적인 사용법**에 대해 다룬다.
  - 중/고급 사용법은 다루지 않는다.
    - 고급 기능을 사용하여 구현해야 할 시스템은 매우 적다.
    - 버전이 올라가면 완전히 구조가 바뀔 수 있다.
      - 예) Tensorflor 1.x와 2.x는 interface가 완전히 다르다.
        - 하위 호환되지 않는다.
        - Tensorflow 1.x에 정통한 사람도 2.x 다시 배워야 한다.
  - 관심 있다면 수업시간에 학습한 기초적인 사용법에 대한 지식을 바탕으로 Tensorflow 문서를 찾아가면서 중/고급 사용법을 학습하여 전문가가 되어 보자.
- **Neural network의 개념이 실제 Neural network OSS framework에서 어떤 형태로 구현되는지**를 학습한다.

다음 MLP Class 코드를 작성한다. (MLP.py)

```
import tensorflow as tf

class MLP:
    # "hidden_layer_conf" is the array indicates the number of layers (num_of_elements)
    # and the number of elements in each layer.
    def __init__(self, hidden_layer_conf, num_output_nodes):
        self.hidden_layer_conf = hidden_layer_conf
        self.num_output_nodes = num_output_nodes
        self.logic_op_model = None
```

다음 MLP Class 코드를 작성한다. (MLP.py) (Cont'd)

```
### A member function of Class MLP
 def build_model(self):
       input_layer = tf.keras.Input(shape=[2, ])
       hidden_layers = input_layer

       if self.hidden_layer_conf is not None:
          for num_hidden_nodes in self.hidden_layer_conf:
             hidden_layers = tf.keras.layers.Dense(units=num_hidden_nodes,
                                                   activation=tf.keras.activations.sigmoid,
                                                   use_bias=True)(hidden_layers)

       output = tf.keras.layers.Dense(units=self.num_output_nodes,
                                      activation=tf.keras.activations.sigmoid,
                                      use_bias=True)(hidden_layers)

       self.logic_op_model = tf.keras.Model(inputs=input_layer, outputs=output)

       sgd = tf.keras.optimizers.SGD(learning_rate=0.1)
       self.logic_op_model.compile(optimizer=sgd, loss="mse")
```

다음 MLP Class 코드를 작성한다. (MLP.py) (Cont'd)

```
### A member function of Class MPL
    def fit(self, x, y, batch_size, epochs):
        self.logic_op_model.fit(x=x, y=y, batch_size=batch_size, epochs=epochs)

    def predict(self, x, batch_size):
        prediction = self.logic_op_model.predict(x=x, batch_size=batch_size)
        return prediction
```

MLP Class를 사용하는 XOR classifier를 작성한다. (xor_classifier.py)

(Binary) AND Classifier
- 다음과 같은 bit-wise XOR 연산을 Neural Network으로 구현

- **입력 :** Binary 입력 X1 과 X2
- **출력 : X1 XOR X2**

입력 (X1, X2)	X1 = 0	X1 = 1
X2 = 0	**0**	**1**
X2 = 1	**1**	**0**

MLP Class를 사용하는 XOR classifier를 작성한다. (xor_classifier.py) (Cont'd)

```
import tensorflow as tf
from MLP import MLP

def xor_classifier_example ():
    input_data = tf.constant([[0.0, 0.0], [0.0, 1.0], [1.0, 0.0], [1.0, 1.0]])
    input_data = tf.cast(input_data, tf.float32)

    xor_labels = tf.constant([0.0, 1.0, 1.0, 0.0])
    xor_labels = tf.cast(xor_labels, tf.float32)

    batch_size = 1
    epochs = 1500

    mlp_classifier = MLP(hidden_layer_conf=[4], num_output_nodes=1)
    mlp_classifier.build_model()
    mlp_classifier.fit(x=input_data, y=xor_labels, batch_size=batch_size, epochs=epochs)
```

MLP Class를 사용하는 XOR classifier를 작성한다. (xor_classifier.py) (Cont'd)

```
### definition of  xor_classifier_example () function (cont'd)

    ######## MLP XOR prediciton
    prediction = mlp_classifier.predict(x=input_data, batch_size=batch_size)
    input_and_result = zip(input_data, prediction)
    print("====== MLP XOR classifier result =====")
    for x, y in input_and_result:
        if y > 0.5:
            print("%d XOR %d => %.2f => 1" % (x[0], x[1], y))
        else:
            print("%d XOR %d => %.2f => 0" % (x[0], x[1], y))

# Entry point
if __name__ == '__main__':
    xor_classifier_example()
```

생각해 보기 : 입출력 데이터

```
import tensorflow as tf
from MLP import MLP

def xor_classifier_example ():
    input_data = tf.constant([[0.0, 0.0], [0.0, 1.0], [1.0, 0.0], [1.0, 1.0]])
    input_data = tf.cast(input_data, tf.float32)

    xor_labels = tf.constant([0.0, 1.0, 1.0, 0.0])
    xor_labels = tf.cast(xor_labels, tf.float32)

    batch_size = 1
    epochs = 1500

    mlp_classifier = MLP(hidden_layer_conf=[4], num_output_nodes=1)
    mlp_classifier.build_model()
    mlp_classifier.fit(x=input_data, y=xor_labels, batch_size=batch_size, epochs=epochs)
```

이 부분이 뜻하는 것은?

이 부분이 뜻하는 것은?

생각해 보기 : 네트워크 형태

```
import tensorflow as tf
from MLP import MLP

def xor_classifier_example ():
    input_data = tf.constant([[0.0, 0.0], [0.0, 1.0], [1.0, 0.0], [1.0, 1.0]])
    input_data = tf.cast(input_data, tf.float32)

    and_labels = tf.constant([0.0, 1.0, 1.0, 0.0])
    xor_labels = tf.cast(xor_labels, tf.float32)

    batch_size = 1
    epochs = 1500

    mlp_classifier = MLP(hidden_layer_conf=[4], num_output_nodes=1)
    mlp_classifier.build_model()
    mlp_classifier.fit(x=input_data, y=xor_labels, batch_size=batch_size, epochs=epochs)
```

이 부분으로 어떤 형태의 MLP가 생성되는가?

```
mlp_classifier = MLP(hidden_layer_conf=[4], num_output_nodes=1)
mlp_classifier.build_model()
mlp_classifier.fit(x=input_data, y=xor_labels, batch_size=batch_size, epochs=epochs)
```

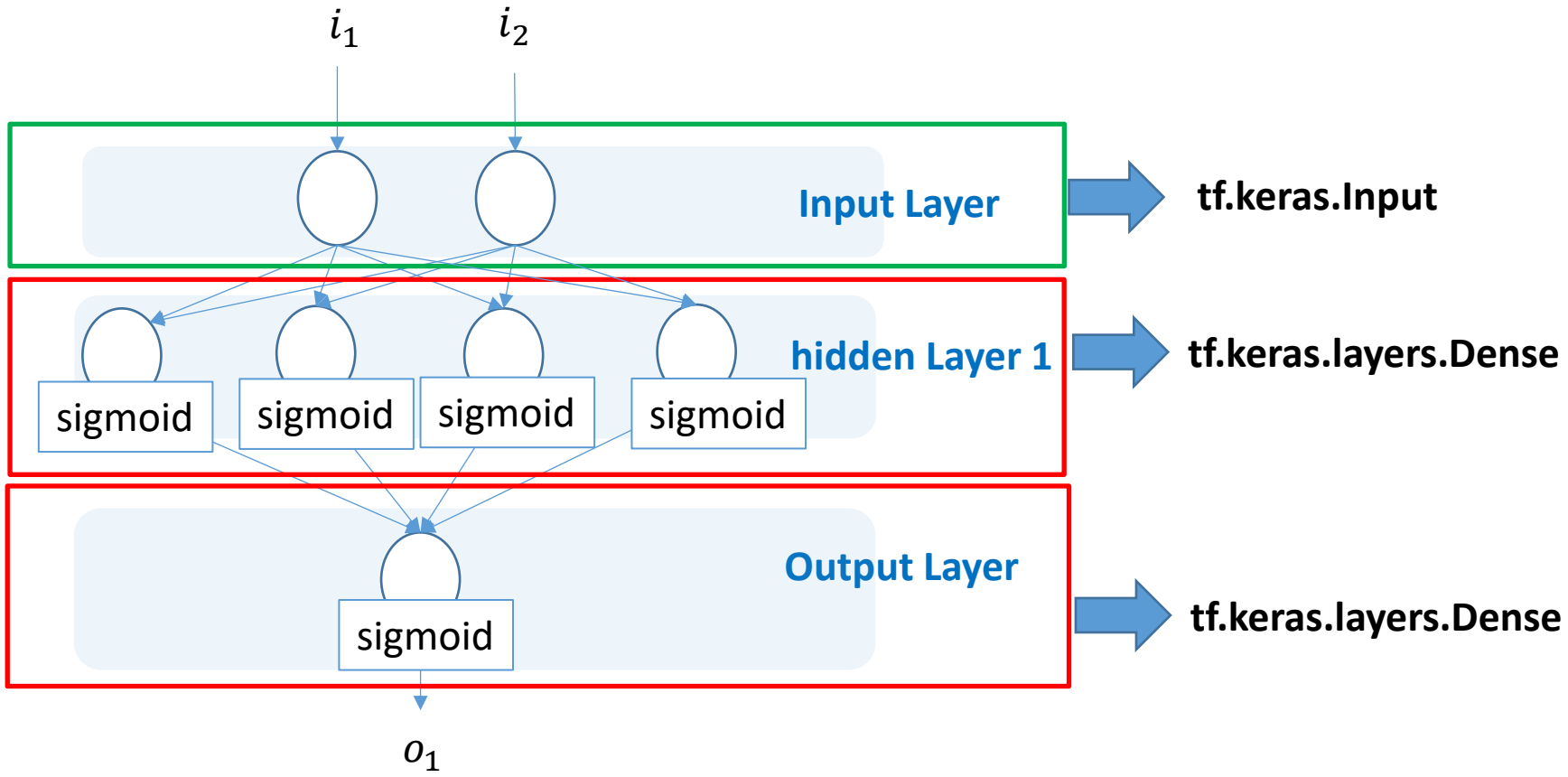
이 부분으로 어떤 형태의 MLP가 생성되는가?

```
### A member function of Class MPL
def build_model(self):
    input_layer = tf.keras.Input(shape=[2, ])
    hidden_layers = input_layer

    if self.hidden_layer_conf is not None:  ## 위의 코드에 의하면 self.hidden_layer_conf의 값은 "[4]"
        for num_hidden_nodes in self.hidden_layer_conf:
            hidden_layers = tf.keras.layers.Dense(units=num_hidden_nodes,
                                                  activation=tf.keras.activations.sigmoid,
                                                  use_bias=True)(hidden_layers)

    output = tf.keras.layers.Dense(units=self.num_output_nodes, ## 위의 코드에 의하면 self.num_output_nodes 의 값은 1
                                   activation=tf.keras.activations.sigmoid,
                                   use_bias=True)(hidden_layers)

    self.logic_op_model = tf.keras.Model(inputs=input_layer, outputs=output)

    sgd = tf.keras.optimizers.SGD(learning_rate=0.1)
    self.logic_op_model.compile(optimizer=sgd, loss="mse")
```

```
mlp_classifier = MLP(hidden_layer_conf=[4], num_output_nodes=1)
mlp_classifier.build_model()
mlp_classifier.fit(x=input_data, y=xor_labels, batch_size=batch_size, epochs=epochs)
```

이 부분으로 어떤 형태의 MLP가 생성되는가?

정답



생각해 보기 : 기타 디자인 이슈

- 왜 Input Layer는 shape = [2,] 일까?
- 왜 output node는 하나일까?
- Hidden node 및 output node의 activation function은 sigmoid 함수 이외의 다른 함수를 사용할 수 있을까?

Model Training

- 모델 학습: 모델이 주어진 입력에 대해, 원하는 결과를 출력할 수 있도록 최적의 Model Parameter값을 결정하는 것.
- 학습을 위해 필요한 도구
  - Objective Function (목적 함수)
  - Optimization Method (최적화 방법)
  - 데이터
    - Training data, Validation data, Test data

Model Training : Objective Function

- Objective Function (목적 함수)
  - 모델이 사용자가 원하는 예측을 하고 있는지를 "수학적인 함수"를 사용하여 평가
  - 일반적으로 사용자가 원하는 값과 실제 예측 차이의 차 (Error, 오차)를 측정함
  - Convex Function (볼록 함수)
  - 주로 사용되는 것으로 MSE, Cross-Entropy Loss등이 있음.

MSE (Mean Square Loss)

- **각각의 Training data에 대해, "모델을 사용하여 예측한 값"과 "실제 값"의 차이의 제곱의 합.**

Training data 가 $< x_1, y_1 >, < x_2, y_2 > \ldots, < x_n, y_n >$ 의 n쌍 있을 때,
Simple linear regression "$y = wx + b$" 의 MSE는 다음과 같다.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w \cdot x_i + b))^2$$

Optimization Method

- Optimization Method
  - 주어진 데이터(Training Data)에 대해 Objective Function의 값을 최소화 하는 Model Parameter값을 계산하는 알고리즘
    - Objective Function의 값을 최소화한다는 것은 어떤 뜻일까?
  - Neural Network 의 기본적인 알고리즘으로 Gradient Descent가 존재함
    - Adam 등 주로 사용되는 optimization metho는 Gradient Descent의 파생형

Gradient Descent

$$w_i = w_i - \alpha \cdot \frac{\partial obj}{\partial w_i}$$

$$b = b - \alpha \cdot \frac{\partial obj}{\partial b}$$

Parameter를 gradient ($\frac{\partial obj}{\partial w_i}$, $\frac{\partial obj}{\partial b}$)의 반대 방향으로 update한다.

MLP class build_model() AGAIN

```
### A member function of Class MPL
 def build_model(self):
      input_layer = tf.keras.Input(shape=[2, ])
      hidden_layers = input_layer

      if self.hidden_layer_conf is not None:
         for num_hidden_nodes in self.hidden_layer_conf:
            hidden_layers = tf.keras.layers.Dense(units=num_hidden_nodes,
                                    activation=tf.keras.activations.sigmoid,
                                    use_bias=True)(hidden_layers)

      output = tf.keras.layers.Dense(units=self.num_output_nodes,
                                    activation=tf.keras.activations.sigmoid,
                                    use_bias=True)(hidden_layers)

      self.logic_op_model = tf.keras.Model(inputs=input_layer, outputs=output)

      sgd = tf.keras.optimizers.SGD(learning_rate=0.1)
      self.logic_op_model.compile(optimizer=sgd, loss="mse")
```

이 부분은 뭐하는 부분?

MLP class build_model() AGAIN

```
### A member function of Class MPL
def fit(self, x, y, batch_size, epochs):
    self.logic_op_model.fit(x=x, y=y, batch_size=batch_size, epochs=epochs)

def predict(self, x, batch_size):
    prediction = self.logic_op_model.predict(x=x, batch_size=batch_size)
    return prediction
```

학습을 위한 코드.
x, y, batch_size, Epochs는 각각 무엇일까?

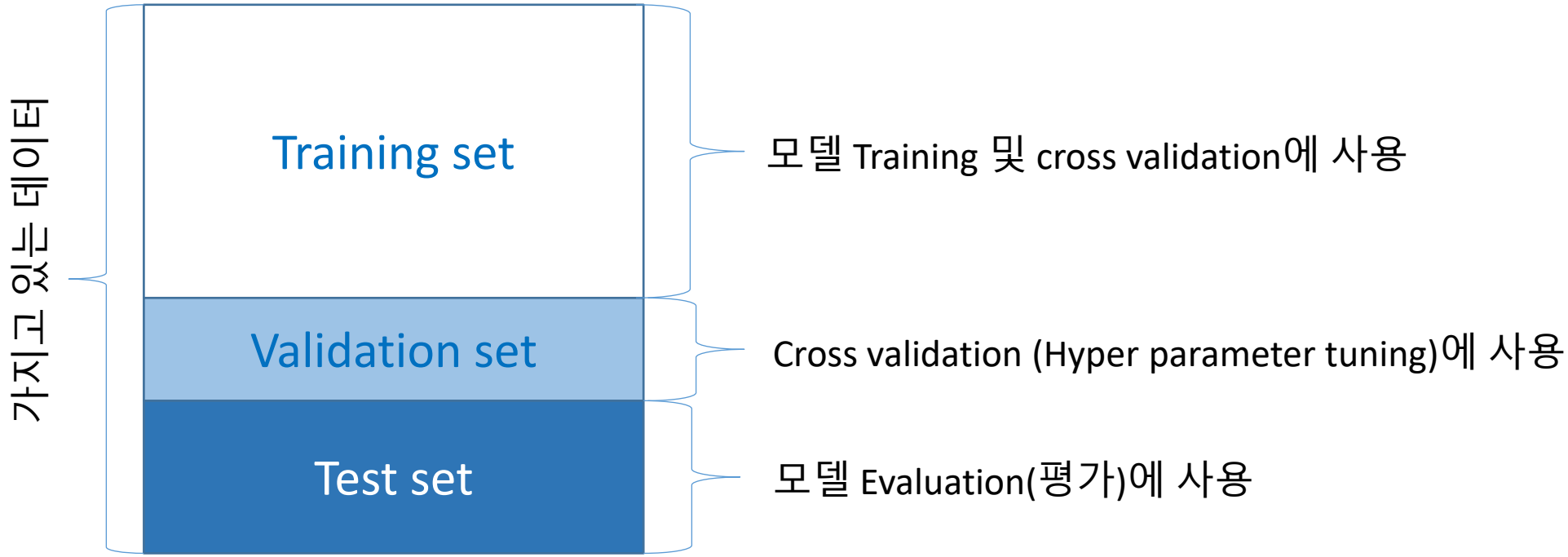
xor_classifier.py 예

```
import tensorflow as tf
from MLP import MLP

def xor_classifier_example ():
    input_data = tf.constant([[0.0, 0.0], [0.0, 1.0], [1.0, 0.0], [1.0, 1.0]])
    input_data = tf.cast(input_data, tf.float32)

    xor_labels = tf.constant([0.0, 1.0, 1.0, 0.0])
    xor_labels = tf.cast(xor_labels, tf.float32)

    batch_size = 1
    epochs = 1500

    mlp_classifier = MLP(hidden_layer_conf=[4], num_output_nodes=1)
    mlp_classifier.build_model()
    mlp_classifier.fit(x=input_data, y=xor_labels, batch_size=batch_size, epochs=epochs)
```

이 예는 Training set 이 곧 Test set. Evaluation set은 사용하지 않음.

Data: Training set, Validation set, Test set

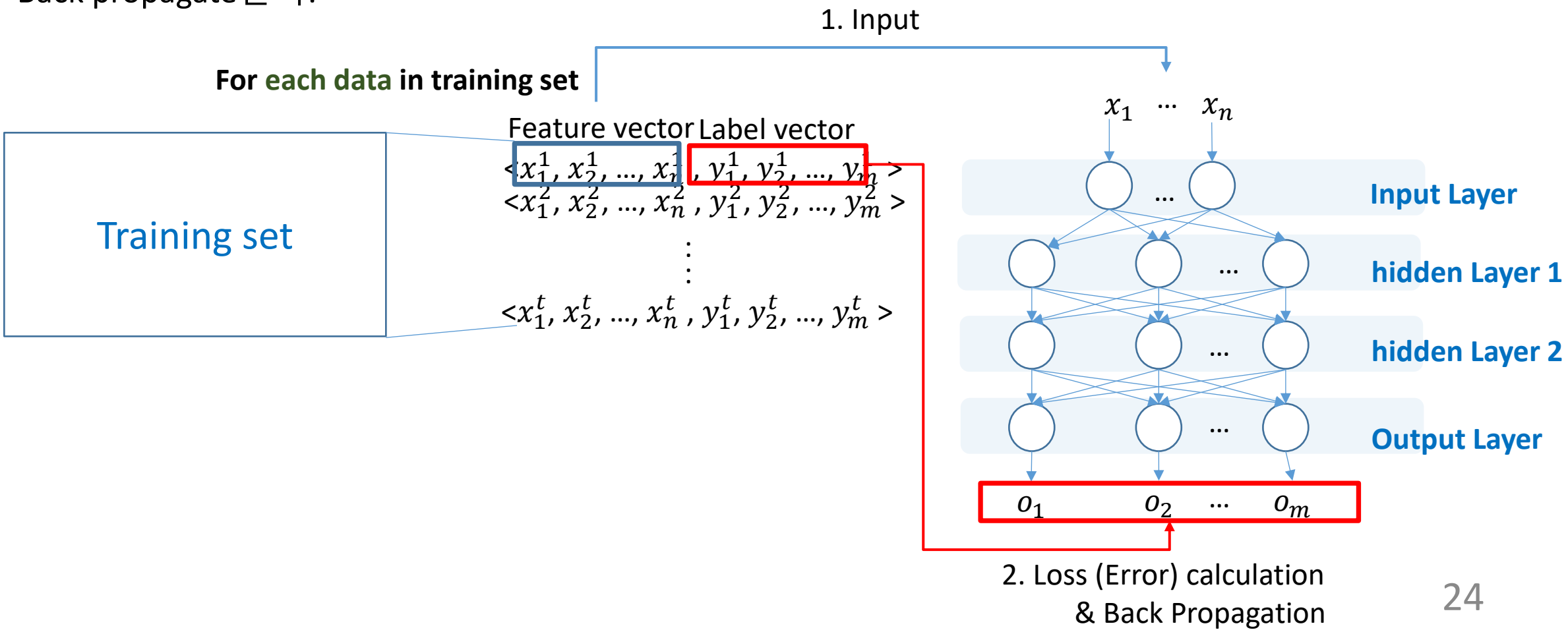
- 모델이 입력값에 대해 원하는 출력값을 생성하도록 학습 시키기 위한 데이터
  - Training (Data) Set, Validataion (Data) Set, Test (Data) Set로 구성됨



- 위 각각의 set 사이에 중복되어 포함되는 데이터는 없다.
  - $\text{Training\_set} \cap \text{Validation\_set} = \Phi$ AND $\text{Validation\_set} \cap \text{Test\_set} = \Phi$ AND $\text{Test\_set} \cap \text{Training\_set} = \Phi$
- 각 set의 크기는 일반적으로 Training set > Test set > Validation Set이다.
  - 예) Training set : 0.7, Validation set: 0.1, Test set: 0.2

Neural Network Training – Epochs

Training 데이터 각각의 feature vector를 입력으로 network를 사용하여 결과 값(예측 값)을 계산하여, 이를 해당 feature vector에 대한 정답 값(관측 값Label vector) 과 비교하여 Loss(Error)를 계산하고 이를 Back propagate한다.



Neural Network Training – Epochs (Cont'd)

모든 training data에 대해 Error 계산 & Back propagation을 완료하였지만

아직 더 model parameter를 학습할 필요가 있으면 (Loss function (Objective function)의 값을 더 최소화 할 여지가 있으면), 다시 training data를 처음부터 사용하여 Error 계산 & Back propagation을 반복한다.

이를 Loss function의 값이 충분히 최소화 될 때 까지 반복한다.

Error 계산 & Back propagation에 있어서 Training data를 처음부터 끝까지 1회씩 사용할 때까지를 1 Epoch라 한다.

```
PERCEPTRONLEARNING[M+, M−]
w = arbitrary vector of real numbers
Repeat
    For all x ∈ M+
        If w x ≤ 0 Then w = w + x
    For all x ∈ M−
        If w x > 0 Then w = w − x
Until all x ∈ M+ ∪ M− are correctly classified
```

옆의 **Perceptron Learning**식에서 Perceptron Learning 완료할 때까지Repeat를 1000회 하였다면, 이는 training을 1000 epochs 수행하였다는 의미와 동일 하다.

Neural Network Training – Stochastic, Batch, Mini-batch Training

- **Batch Gradient Descent Vs. Stochastic Gradient Descent : Model parameter update timing**

$$w_i^j = w_i^j - \alpha \cdot \frac{\partial obj}{\partial w_i^j}$$

$$b^j = b^j - \alpha \cdot \frac{\partial obj}{\partial b^j}$$

옆의 gradient descent를 사용한 node j의 parameter update rule을 살펴보자.

Parameter의 gradient ($\frac{\partial obj}{\partial w_i^j}$, $\frac{\partial obj}{\partial w_i^j}$)의 반대 방향으로 update한다.

그건 좋은데 어떤 Timing에서 model parameter를 Update 할 것인가?

Neural Network Training – Stochastic, Batch, Mini-batch Training (Cont'd)

- **Batch Gradient Descent Vs. Stochastic Gradient Descent : Model parameter update timing**

$$w_i^j = w_i^j - \alpha \cdot \frac{\partial obj}{\partial w_i^j}$$

$$b^j = b^j - \alpha \cdot \frac{\partial obj}{\partial b^j}$$

어떤 Timing에서 model parameter를 Update 할 것인가?

Stochastic Gradient Descent

- **Training Data 하나에 대해 Gradient를 계산**한 다음, 이를 바탕으로 Update. (1 data에 1회 model parameter update.)

$\frac{\partial obj}{\partial w_i^j} = \frac{\partial obj_x}{\partial w_i^j}$

x: data $x \in Tr$

obj_x : data x 로 계산한 결과에 대한 Error.

Batch Gradient Descent

- **모든 Training Data에 대해 Gradient를 계산한 다음, 이를 평균**한 Gradient 값을 사용하여 Update. (1 Epoch에 1회 model parameter update)

$\frac{\partial obj}{\partial w_i^j} = \frac{1}{|Tr|} \sum_{x \in Tr} \frac{\partial obj_x}{\partial w_i^j}$

Tr: the Training set

obj_x : data x 로 계산한 결과에 대한 Error.

Batch Vs. Stochastic: Loss(Cost or Error or Objective) Function decrease graph

Batch Gradient Descent가 Smooth하게 Loss Function 값이 감소하는데 비해 Stochastic Gradient Descent는 감소 pattern이 noisy하다.

- Stochastic Gradient Descent는 Loss Function 값이 더 내려갈 수 있는지, 이미 Local minima인지 판단하기 어려울 때가 있다. (올라갔다 내려갔다를 반복하므로)

Batch Gradient Descent
Cost decrease graph

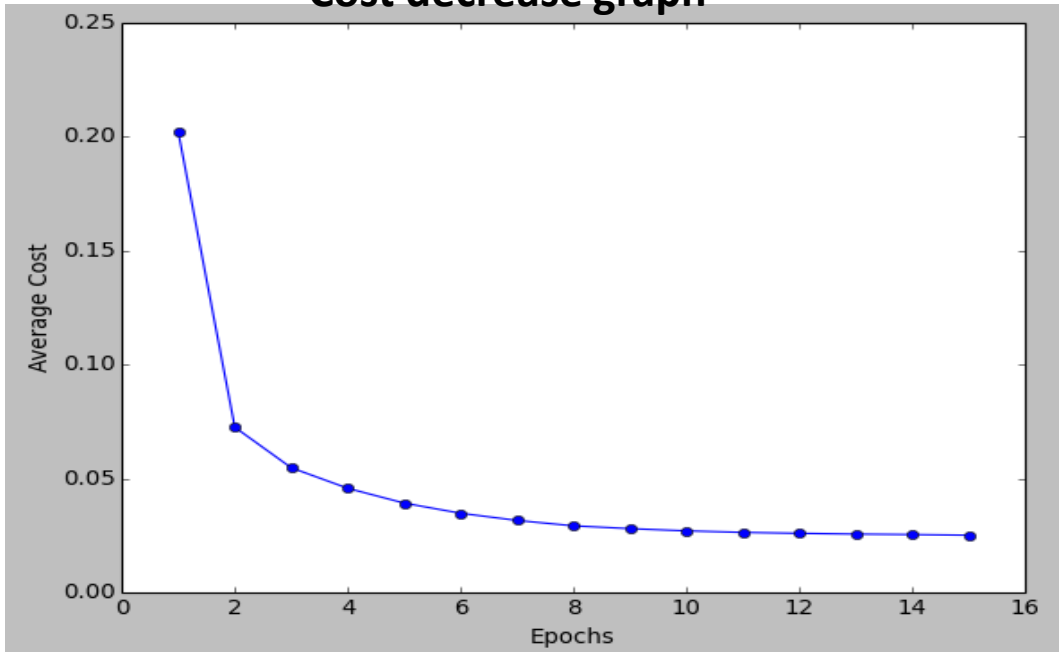


Image Source: https://www.bogotobogo.com/python/scikit-learn/scikit-learn_batch-gradient-descent-versus-stochastic-gradient-descent.php

Stochastic Gradient Descent
Cost decrease graph

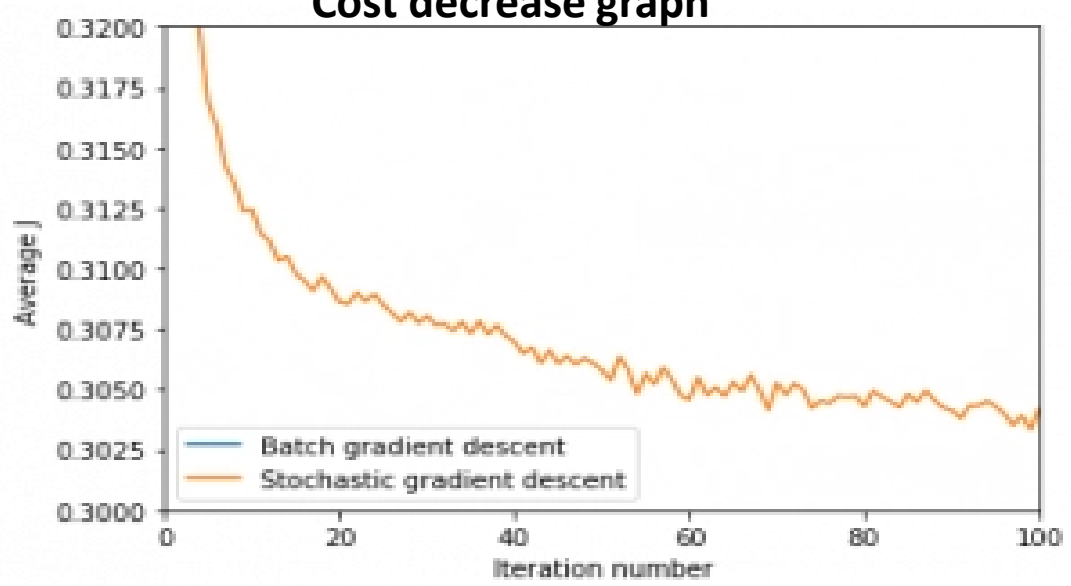


Image source :https://adventuresinmachinelearning.com/stochastic-gradient-descent/

Batch Vs. Stochastic: Update Cost

각각의 Model parameter 관점에서,

- Batch gradient descent는 epoch 1회에 1회의 update만 발생한다.
  - Training data 가 1M 개일 경우, 1M개의 데이터에 대한 gradient를 다 계산하고 나야 parameter가 1회 update된다.
  => 1 Epoch에 1회 update => **1회 update 비용이 너무 크다.**

- Stochastic Gradient Descent는 Training data 하나에 대해 1회 update 되므로,
  - Training data 가 1M 개일 경우, 1M회 update 된다.
  => 1 Epoch에 1M 회 update => **1회 update 비용이 작다.**

Mini batch gradient descent

- Batch Gradient Descent와 Stochastic Gradient Descent의 절충안.
  - b 개의 training data에 대해, Batch gradient descent처럼 gradient 평균을 구해 model parameter를 update한다.
    - 각각의 Model parameter 관점에서 data b개당 1회 update.
    - b 개는 64, 128, 256, 512, 1024 정도로
      - 1M에 비하면 매우 작다.
      - 1 에 비하면 충분히 크다.
- 최근의 Neural network training은 대부분 Mini batch gradient descent를 사용하여 Training 한다.

Prediction with Trained Model

학습된 모델을 사용하여 주어진 입력을 모델에 주면 이에 대한 예측 결과가 출력

```
### definition of  xor_classifier_example () function (cont'd)

    ######## MLP XOR prediciton
    prediction = mlp_classifier.predict(x=input_data, batch_size=batch_size)
    input_and_result = zip(input_data, prediction)
    print("====== MLP XOR classifier result =====")
    for x, y in input_and_result:
        if y > 0.5:
            print("%d XOR %d => %.2f => 1" % (x[0], x[1], y))
        else:
            print("%d XOR %d => %.2f => 0" % (x[0], x[1], y))

# Entry point
if __name__ == '__main__':
    and_classifier_example()
```

XOR예제는 이론상 출력은 1 아니면 0밖에 없을 텐데, 왜 이런 코드로 결과를 출력할까?

Tensorflow data processing Type : Tensor

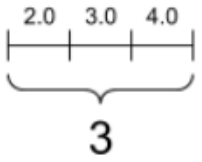
- **Tensor**
  - Tensorflow에서 정의한 Data type.
  - Tensorflow에서 data를 처리하기 위한 I/O 데이터 형식
    - Tensorflow model의 Input 데이터는 Tensor type이어야 한다.
    - Tensorflow model의 Output 데이터는 Tensor type 이다.
  - **개념은 n-dimensional array**이다.
    - 개념은 n-dimensional array이나 python, numpy의 array가 아니기 때문에, tensor data type을 새로 생성해야 한다.
    - Immutable data type
      - 생성된 tensor의 element 값을 생성 이후에 변경할 수 없다.
        - element값을 바꾸고 싶으면 내용이 바뀐 tensor를 다시 생성한다. (새로 메모리 할당 받는다)
    - python, numpy의 array, list등을 tensor로 변환하는 함수를 제공한다.
    - **아래 Tensor 개념에 관한 문서를 읽어서 이해할 것**
    - https://www.tensorflow.org/guide/tensor

Tensor Shape (Dimension)

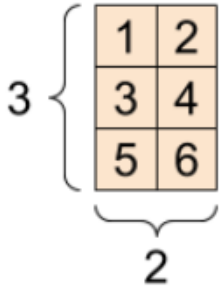
A scalar, shape: [ ]

4

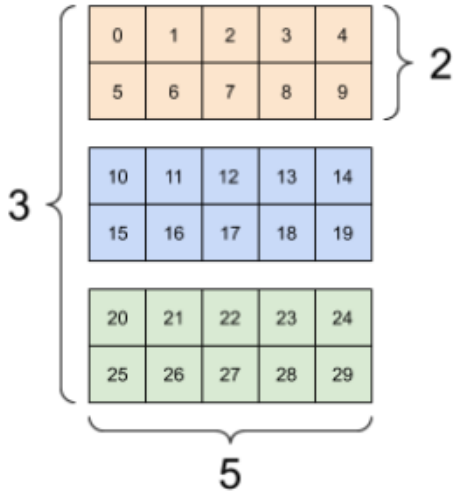
A vector, shape: [3]

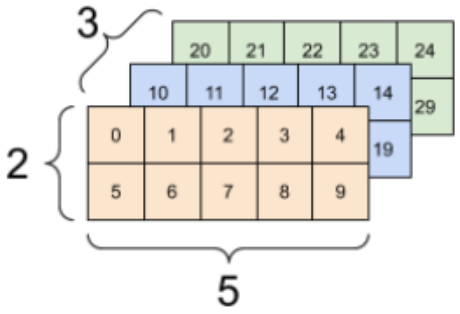


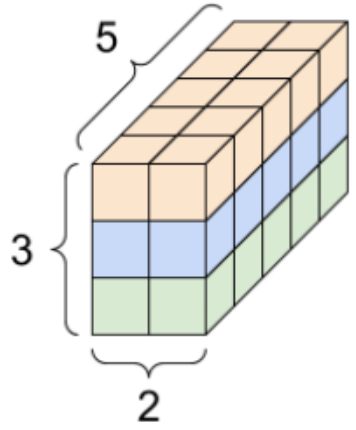
A matrix, shape: [3, 2]



A 3-axis tensor, shape: [3, 2, 5]

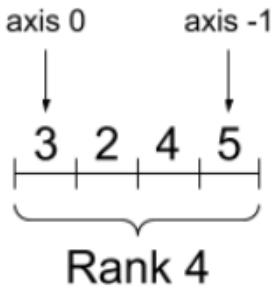


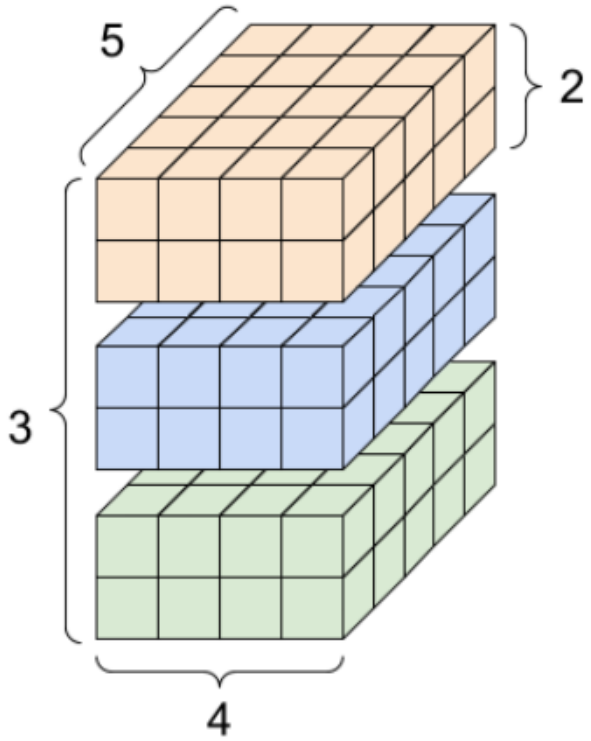




Tensor Shape (Dimension) (Cont'd)

A rank-4 tensor, shape: [3, 2, 4, 5]





Tensor 관련 Functions: tf.constant()

- tf.constant()
  - Value 에 입력된, array, list등을 같은 내용을 가지는 Tensor로 변환.

```
tf.constant(
    value, dtype=None, shape=None, name='Const'
)
```

Doc: https://www.tensorflow.org/api_docs/python/tf/constant

예)

```
input_data = tf.constant([[0.0, 0.0], [0.0, 1.0], [1.0, 0.0], [1.0, 1.0]])
```

위 input_data의 내용을 그림으로 그리면?
위 input_data의 shape는?

Tensor 관련 Functions: tf.convert_to_tensor()

- tf.convert_to_tensor()
  - value 에 입력된, array, list (python 혹은 numpy)등을 같은 내용을 가지는 Tensor로 변환하는데 데이터 타입을 dtype에서 지시하는 type로 type casting.
  - dtype이 입력으로 들어오지 않을 경우 알아서 type 변환.
    - 오변환 방지를 위해 dtype 설정해 주는 것이 낫다.

```
tf.convert_to_tensor(
    value, dtype=None, dtype_hint=None, name=None
)
```

예)

```
my_array = [[0, 0], [0, 1], [1, 0], [1, 1]]
input_data = tf.convert_to_tensor(value=my_array, dtype=tf.float32)
```

Tensor 관련 Functions: tf.cast()

- tf.cast()
  - x 에 입력된 Tensor의 데이터 타입을 dtype에서 지시하는 type로 type casting.
  - Neural network연산은 모두 floating pointer 연산이기 때문에 Integer를 입력으로 주지 않도록 유의하자.

```
tf.cast(
    x, dtype, name=None
)
```

Doc: https://www.tensorflow.org/api_docs/python/tf/cast

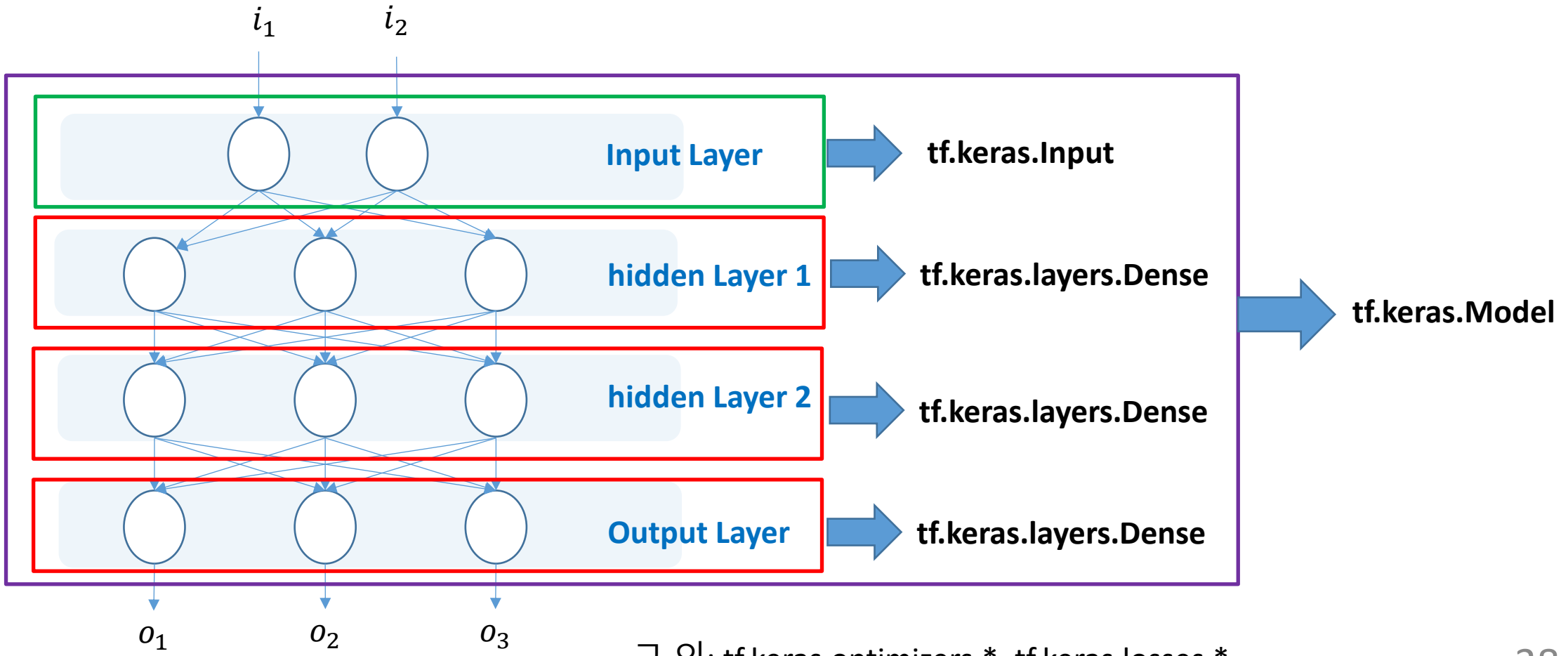
Data Type List는 다음을 참조:
https://www.tensorflow.org/api_docs/python/tf/dtypes/DType

예)

```
input_data = tf.constant([[0, 0], [0, 1], [1, 0], [1, 1]])
input_data = tf.cast(input_data, tf.float32)
```

Keras 주요 Classes

Keras는 Neural Network 구성을 위해 필요한 각각의 부품들을 Class로 제공한다.

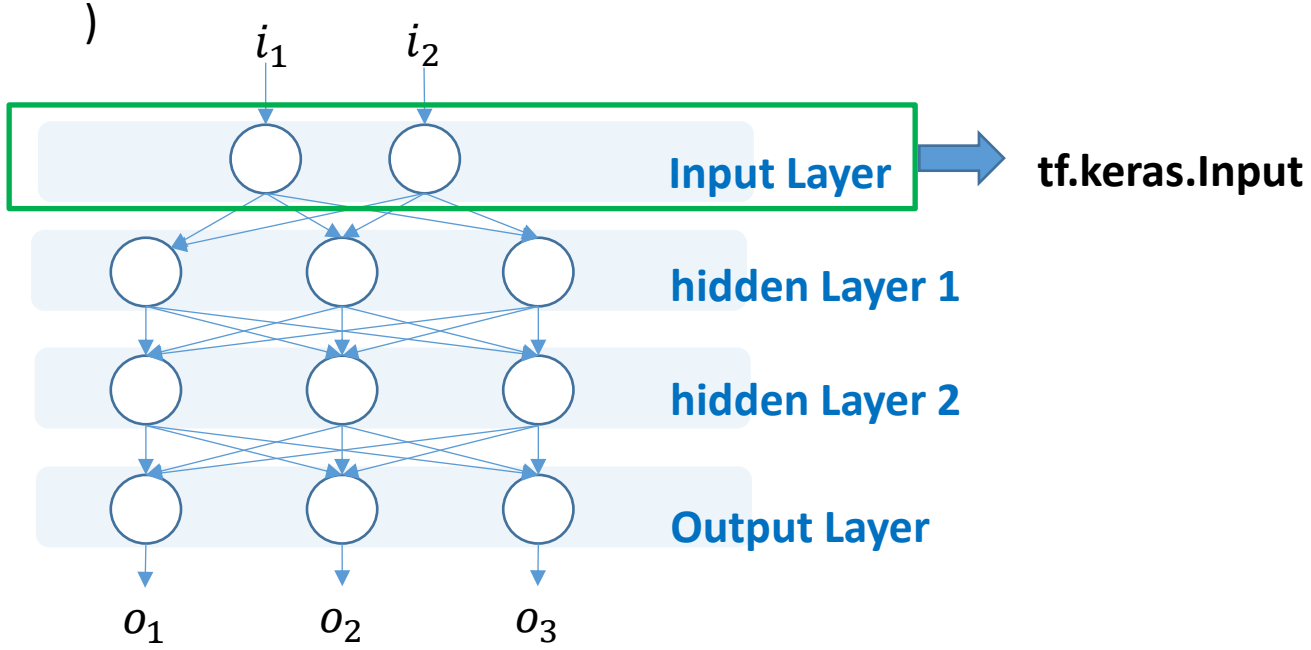


그 외: tf.keras.optimizers.*, tf.keras.losses.*

Keras: tf.keras.Input

입력 Layer를 정의: shape에 입력의 dimension을 정의한다.

```
tf.keras.Input(
    shape=None, batch_size=None, name=None, dtype=None, sparse=False, tensor=None,
    ragged=False, **kwargs
)
```



tf.keras.Input

-예) 입력이 2 dimensions일 경우 다음과 같이 shape를 정의한다.

input_layer = tf.keras.Input(**shape=[2,]**)

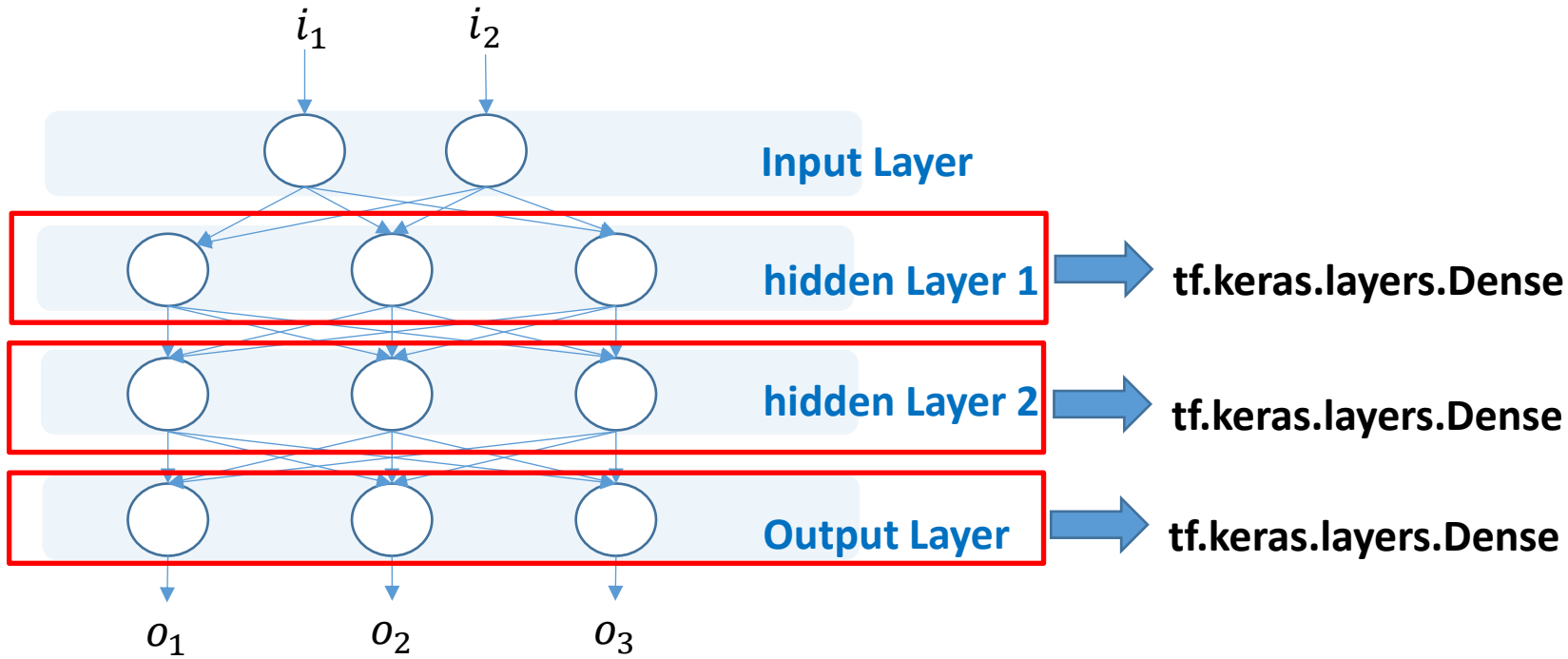
[2,] 와 같이 마지막을 , 로 비워 놓는 이유는 batch size가 아직 정의되지 않았기 때문이다.

- **Documentation**
  - https://www.tensorflow.org/api_docs/python/tf/keras/Input

Keras : tf.keras.layers.Dense

Full Connected Layer 1층을 정의: units에 Layer에 배치할 node 개수, activation에 activation 함수를 assign 한다.

tf.keras.layers.Dense(**units, activation**=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None, **kwargs)

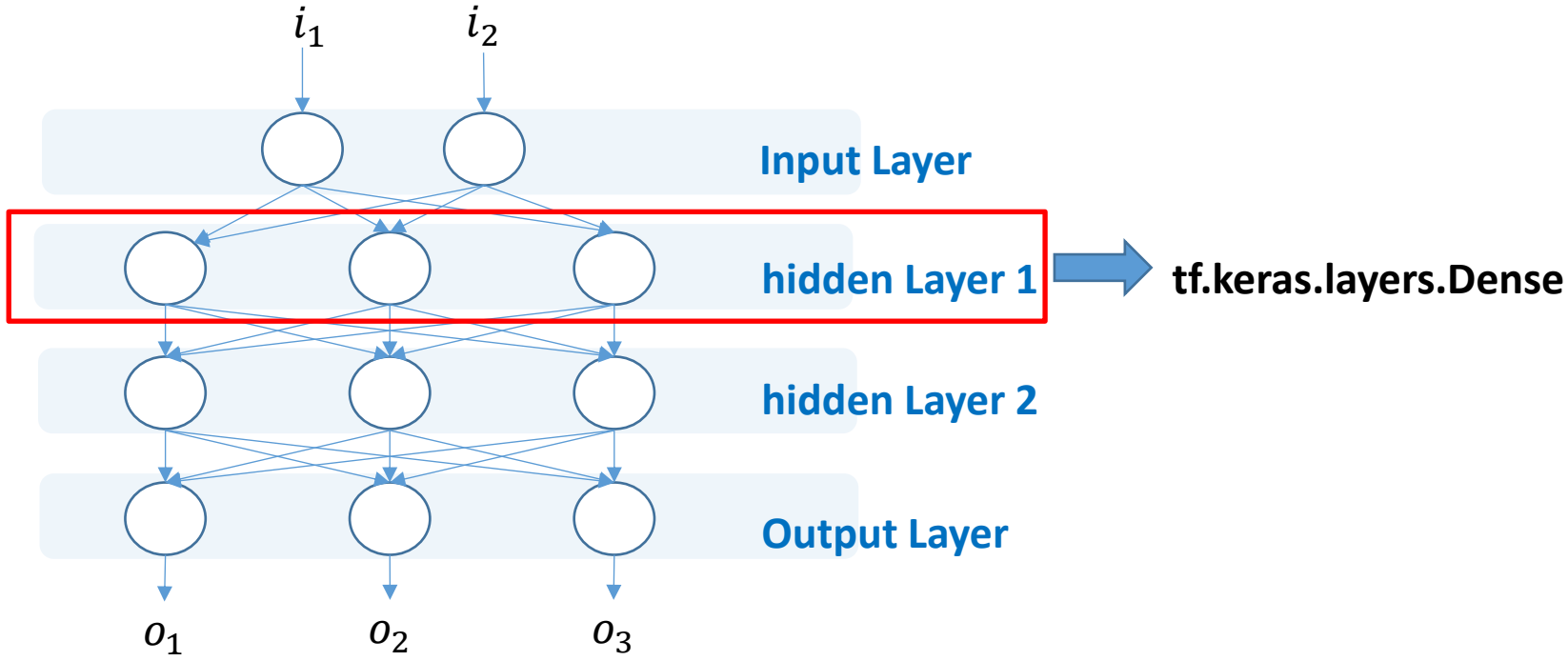


- Documentation
  - https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

Keras : tf.keras.layers.Dense (Cont'd)

-예) Input layer의 출력을 입력으로 하는 Node가 3 개 있고, 각 node의 activation함수가 sigmoid함수인 Layer를 정의한다.

hidden_layer1 = tf.keras.layers.Dense(units=3, activation=**tf.keras.activations.sigmoid**)(input_layer)



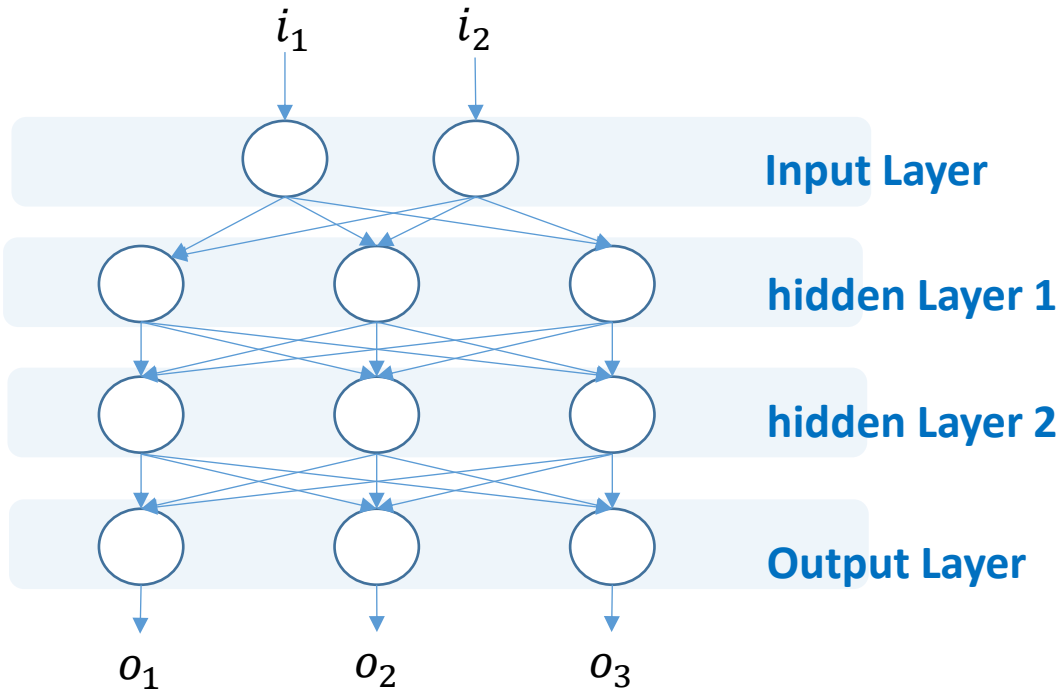
Keras : tf.keras.layers.Dense (Cont'd)

- tf.keras.layers 아래에는 Dense layer 말고도 neural network에 사용되는 여러 layer들이 추상화되어 제공된다.
  - 예) CNN, RNN 등.
  - 자세히는 https://www.tensorflow.org/api_docs/python/tf/keras/layers/ 를 참고.

- tf.keras.activations 아래에는 sigmoid 말고도 여러 neural network에 사용되는 여러 activation function들이 추상화되어 제공된다.
  - 예) tanh, relu 등.
  - 자세히는 https://www.tensorflow.org/api_docs/python/tf/keras/activations 참고.

Keras : Functional API

Functional API:
Layer를 정의할 때 먼저 정의한 앞 Layer들을 연결하기 위해, 정의하는 Layer를 함수처럼 사용하고, 앞 Layer들을 정의하는 Layer의 함수 입력처럼 사용하여 Layer의 연결을 표현하는 API 형태.

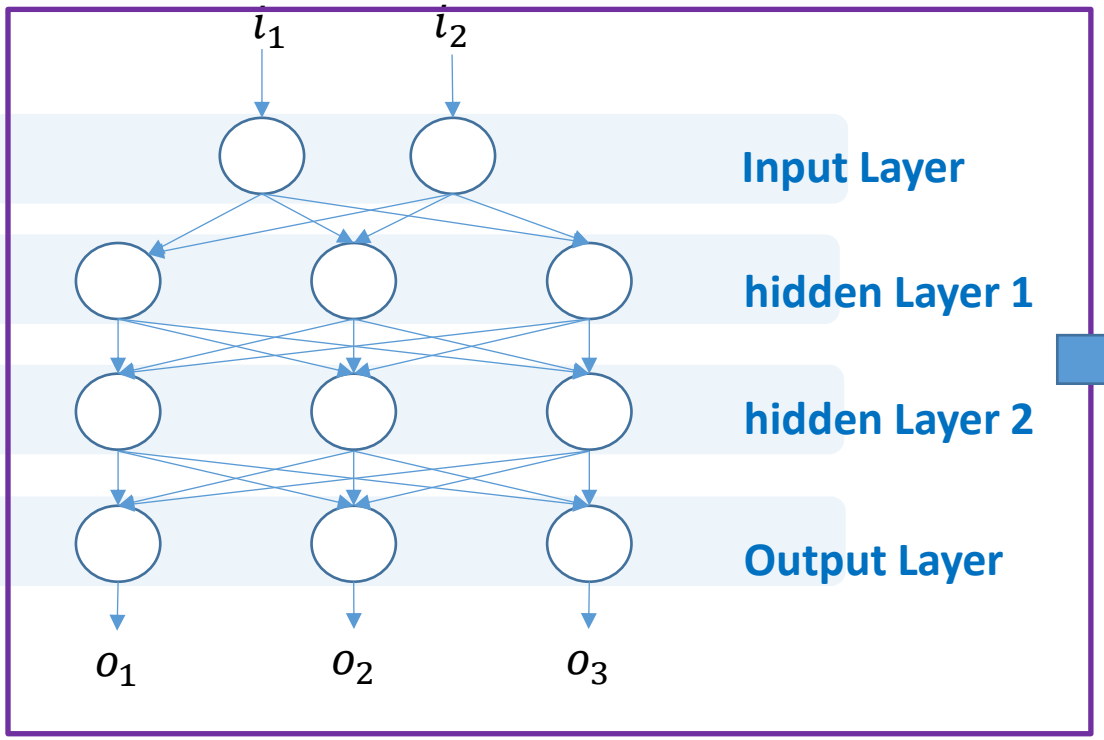



```
input_layer = tf.keras.Input(shape=[2, ])

hidden_layer1 = tf.keras.layers.Dense(units=3,
activation=tf.keras.activations.sigmoid)(input_layer)

hidden_layer2 = tf.keras.layers.Dense(units=3,
activation=tf.keras.activations.sigmoid)(hidden _layer1)

output_layer = tf.keras.layers.Dense(units=3,
activation=tf.keras.activations.sigmoid)(hidden _layer2)
```

이 외에 add() 와 같은 함수를 사용하여 Model에 Layer를 추가하는 Sequential API 형태도 제공하나 Functional API가 더 강력한 Expressive power를 제공하므로 수업에서는 Functional API만 다룸.
관심 있으면 https://www.tensorflow.org/api_docs/python/tf/keras/Sequential 참조.

Keras : tf.keras.Model

앞의 tf.keras.Input, tf.keras.layers 을 사용하면 Network을 정의할 수 있다.
다만, Network 정의만으로는 모델 Training 및 inference를 하기에 정보가 충분하지 않다.



tf.keras.Model

어떤 Loss Function (Objective Function)을 사용할 것인가?

어떤 알고리즘으로 Model parameter를 update 할 것인가?

Training data를 사용하여 실제로 예측값(output)을 계산하고, 이를 해당 data의 관측값 (label)과 비교하여 Objective Function의 값을 계산하고 이를 이용하여 Model parameter를 update하는 일련의 처리는 어떻게 할 것인가?

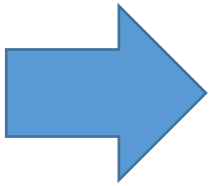
- Documentation
  - https://www.tensorflow.org/api_docs/python/tf/keras/Model

Keras : tf.keras.Model (Cont'd)

어떤 Loss Function (Objective Function)을 사용할 것인가?

어떤 알고리즘으로 Model parameter를 update 할 것인가?

Training data를 사용하여 실제로 예측 값(output)을 계산하고, 이를 해당 data의 관측값 (label)과 비교하여 Objective Function의 값을 계산하고 이를 이용하여 Model parameter를 update하는 일련의 처리는 어떻게 할 것인가?



tf.keras.Model class 가 추상화

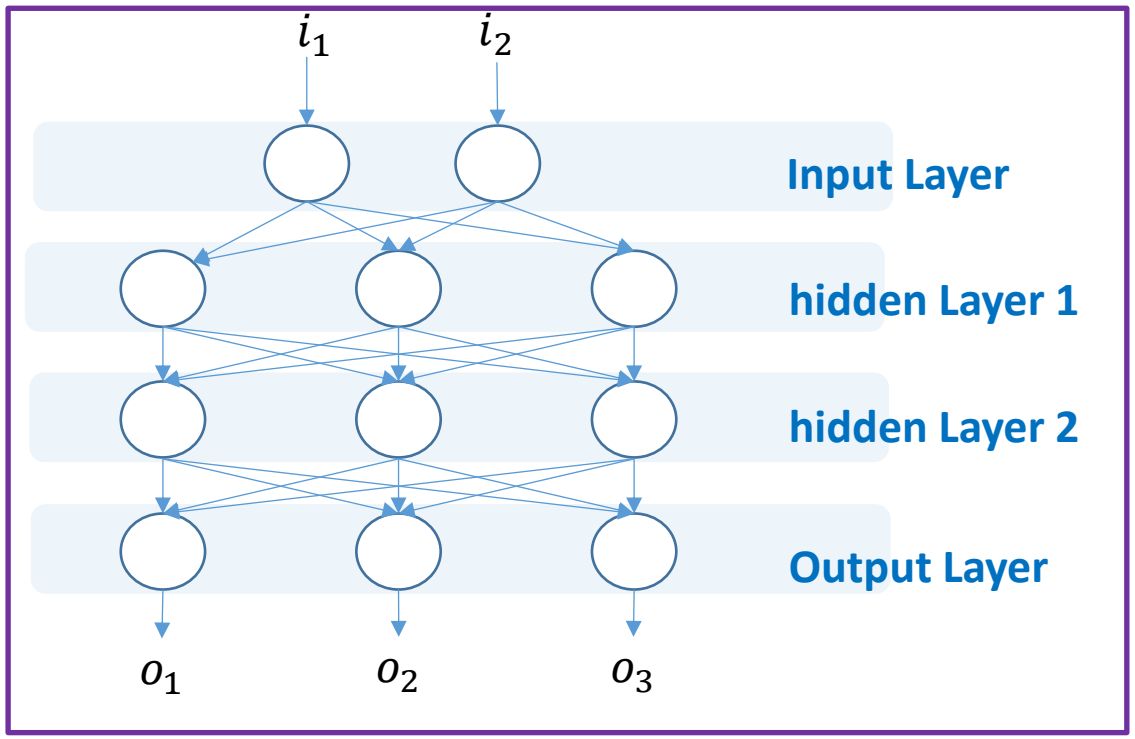
tf.keras.Model Member Function

- 주요한 Member Function 및 기능
  - compile() : loss function, optimizer (loss function optimization algorithm) 등록.
  - fit() : training data를 사용하여 model training.
  - predict() : test data 혹은 실제 데이터를 입력으로 model을 사용하여 예측.

Model Creation

Model에 **(1) Input Layer를 등록**하고,
(2) Output을 출력하기 위한 전체 Layer를 등록하여 이들을 처리하기 위한 Model class를 생성한다.

tf.keras.Model




```
input_layer = tf.keras.Input(shape=[2, ])

hidden_layer1 = tf.keras.layers.Dense(units=3,
activation=tf.keras.activations.sigmoid)(input_layer)

hidden_layer2 = tf.keras.layers.Dense(units=3,
activation=tf.keras.activations.sigmoid)(hidden _layer1)

output_layer = tf.keras.layers.Dense(units=3,
activation=tf.keras.activations.sigmoid)(hidden _layer2)
```

```
logic_op_model = tf.keras.Model(inputs=input_layer,
outputs=output_layer)
```

Model Compile

loss function, optimizer(loss function optimization algorithm) 등록.
Model을 생성했으면 반드시 compile() 함수로 Loss function과 Optimizer를 등록해야 training이 가능하다.

```
compile(
    optimizer='rmsprop', loss=None, metrics=None, loss_weights=None,
    weighted_metrics=None, run_eagerly=None, steps_per_execution=None, **kwargs
)
```

예) Model에 Stochastic Gradient Descent와 Mean Square Error 함수를 등록하는 예.

```
sgd = tf.keras.optimizers.SGD(learning_rate=0.1)

logic_op_model.compile(optimizer=sgd, loss="mse")
```

- tf.keras.optimizers 아래에 여러 다른 optimization 알고리즘이 제공됨.
  - Documentation: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers
- tf.keras.losses 아래에 여러 다른 loss function이 제공됨.
  - Documentation: https://www.tensorflow.org/api_docs/python/tf/keras/losses

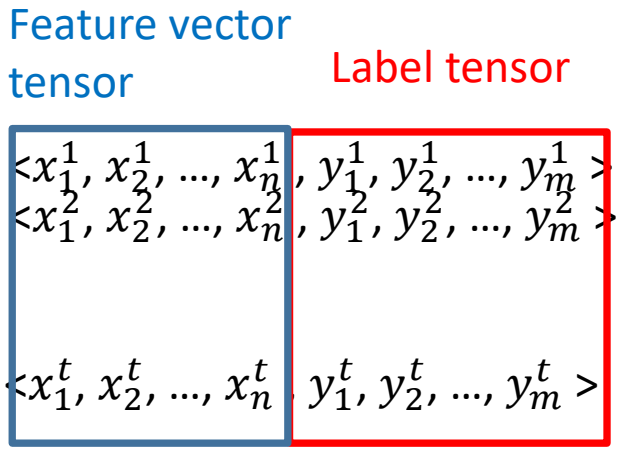
Model Fit

Training data의 feature vector tensor **x** 와 각 data의 label tensor **y**를 사용하여 model을 training한다.
batch_size 는 batch gradient descent의 batch data 개수 b.
epochs 는 몇 epoch까지 training할 것인가?

```
fit(
    x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None,
    validation_split=0.0, validation_data=None, shuffle=True, class_weight=None,
    sample_weight=None, initial_epoch=0, steps_per_epoch=None,
    validation_steps=None, validation_batch_size=None, validation_freq=1,
    max_queue_size=10, workers=1, use_multiprocessing=False
)
```

$< x_1^1, x_2^1, \ldots, x_n^1 , y_1^1, y_2^1, \ldots, y_m^1 >$

Training data가 위와 같이 파란 부분이 feature vector이고,
빨간 부분이 label인 형태라 할 때,
오른쪽 그림은 training dataset에 t개의 데이터가 존재할 때,
Feature vector tensor와 Label tensor를 나타낸다.



Model Predict

Test data 혹은 실제 data의 feature vector tensor **x** 를 input주면 예측한 label의 tensor를 return한다.

```
predict(
    x, batch_size=None, verbose=0, steps=None, callbacks=None, max_queue_size=10,
    workers=1, use_multiprocessing=False
)
```