기계학습
(Machine Learning)
이론 및 실습

7. Tensorflow Keras

Tensorflow 2.6.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 2.6.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 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 코드를 작성한다. (MLP.py) (Cont'd)

```
### 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")
```

다음 MLP 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
```

다음 and classifier를 작성한다. (and_classifier.py)

```
import tensorflow as tf
from MLP import MLP


def and_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, 0.0, 0.0, 1.0])
    and_labels = tf.cast(and_labels, tf.float32)

    and_labels
    batch_size = 1
    epochs = 1500
```

다음 and classifier를 작성한다. (and_classifier.py) (Cont'd)

definition of and_classifier_example () function (cont'd)

```
slp_classifier = MLP(hidden_layer_conf=None, num_output_nodes=1)
slp_classifier.build_model()
slp_classifier.fit(x=input_data, y=and_labels, batch_size=batch_size, epochs=epochs)

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

다음 and classifier를 작성한다. (and_classifier.py) (Cont'd)

```
### definition of  and_classifier_example () function (cont'd)
######## SLP AND prediciton
  prediction = slp_classifier.predict(x=input_data, batch_size=batch_size)
  input_and_result = zip(input_data, prediction)
  print("====== SLP AND classifier result =====")
  for x, y in input_and_result:
     if y > 0.5:
        print("%d AND %d => %.2f => 1" % (x[0], x[1], y))
     else:
        print("%d AND %d => %.2f => 0" % (x[0], x[1], y))

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

다음 and classifier를 작성한다. (and_classifier.py) (Cont'd)

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

Tensorflow data processing Type : Tensor

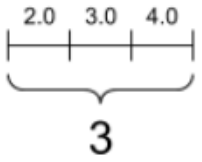
- **Tensor**
  - Tensorflow에서 정의한 Data type.
  - Tensorflow에서 data를 처리하기 위한 I/O 데이터 형식
    - Tensorflow model의 Input 데이터는 Tensor type이어야 한다.
    - Tensorflow model의 Output 데이터는 Tensor type 이다.
  - **개념은 n-dimensional array**이다.
    - 개념은 n-dimensional array이나 python, numpy의 array가 아니기 때문에, tensor data 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)

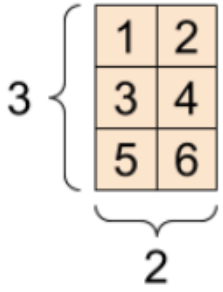
A scalar, shape: [ ]

4

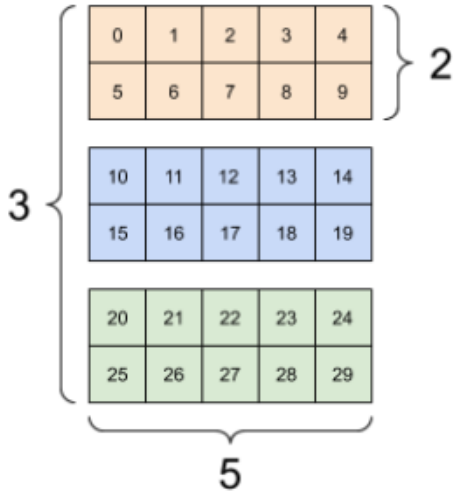
A vector, shape: [3]

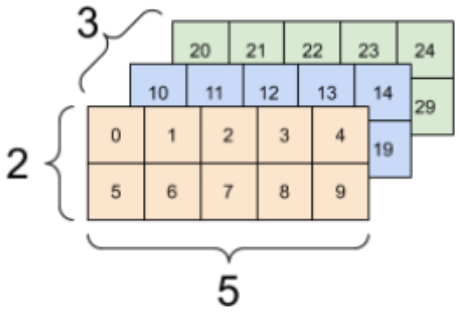


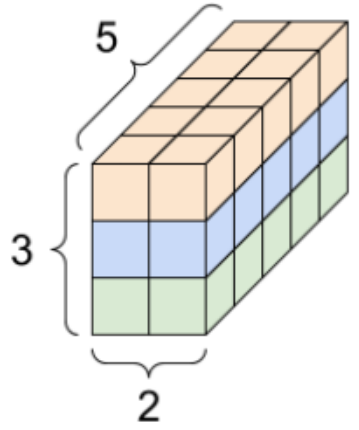
A matrix, shape: [3, 2]



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

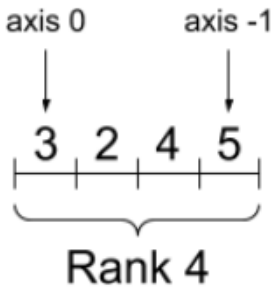


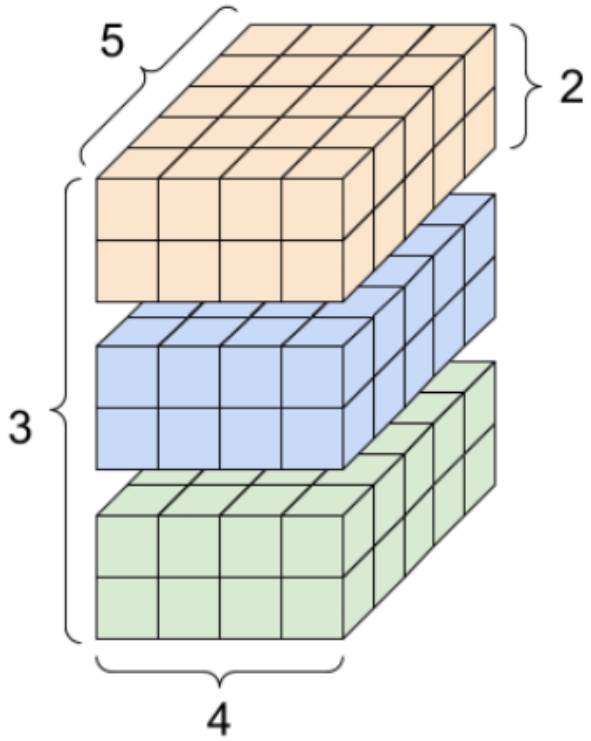




Tensor Shape (Dimension) (Cont'd)

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.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)
```

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.

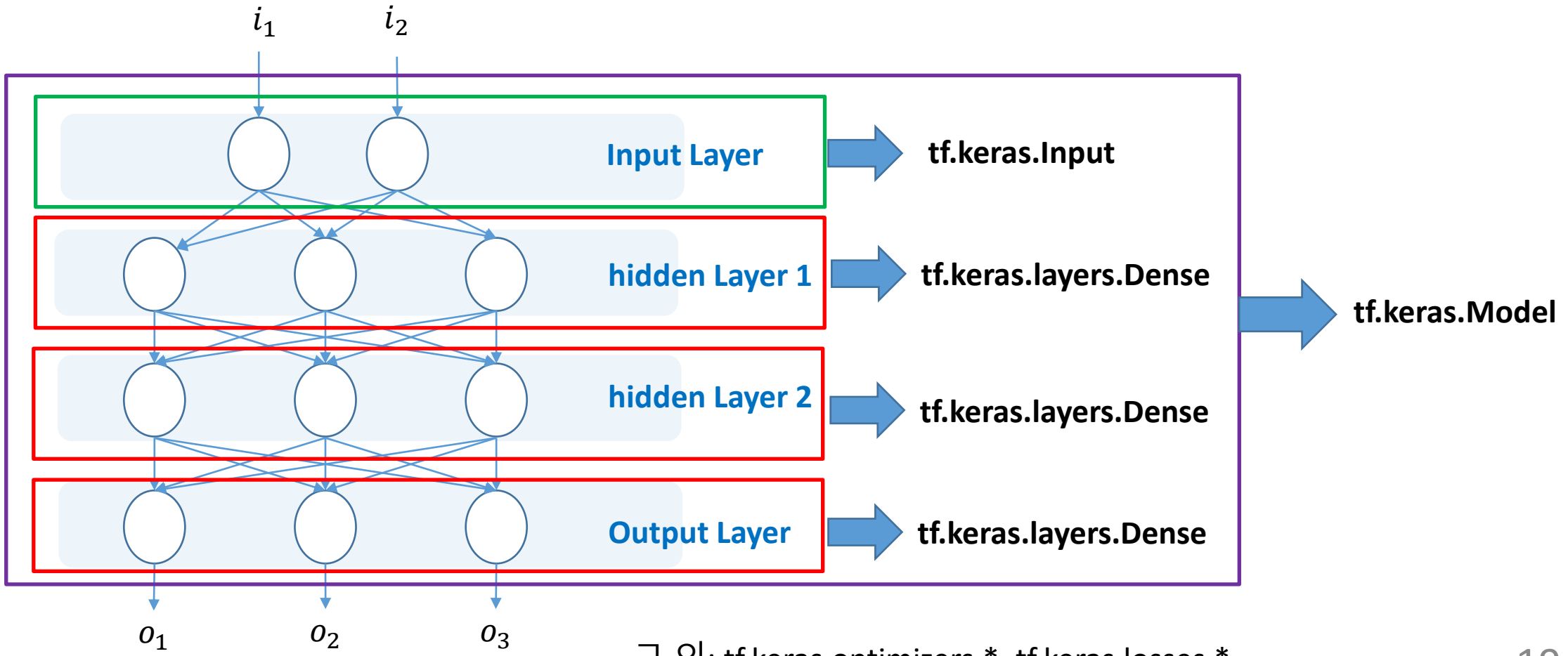
```
tf.convert_to_tensor(
    value, dtype=None, dtype_hint=None, name=None
)
```

예)

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

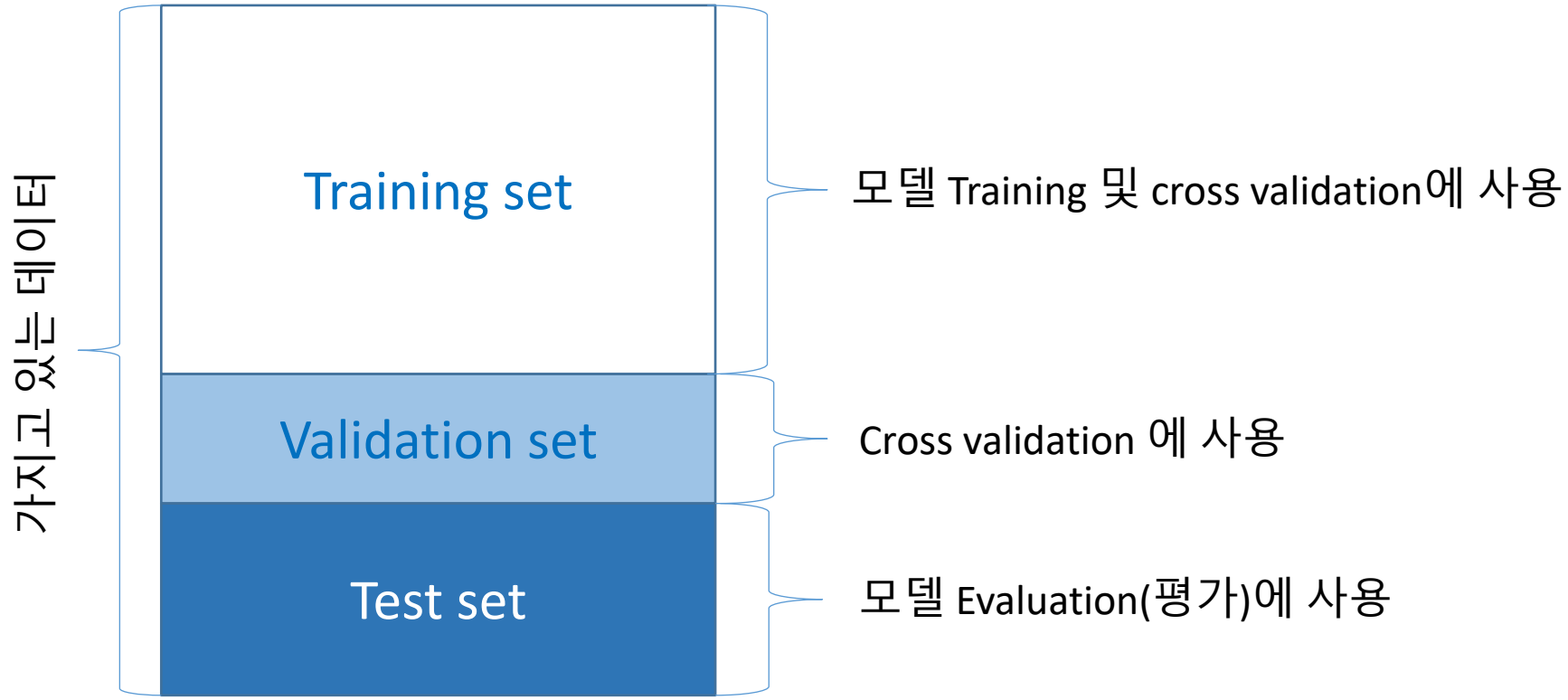
Keras 주요 Classes

Keras는 Neural Network 구성을 위해 필요한 각각의 부품들을 Class로 제공한다.



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

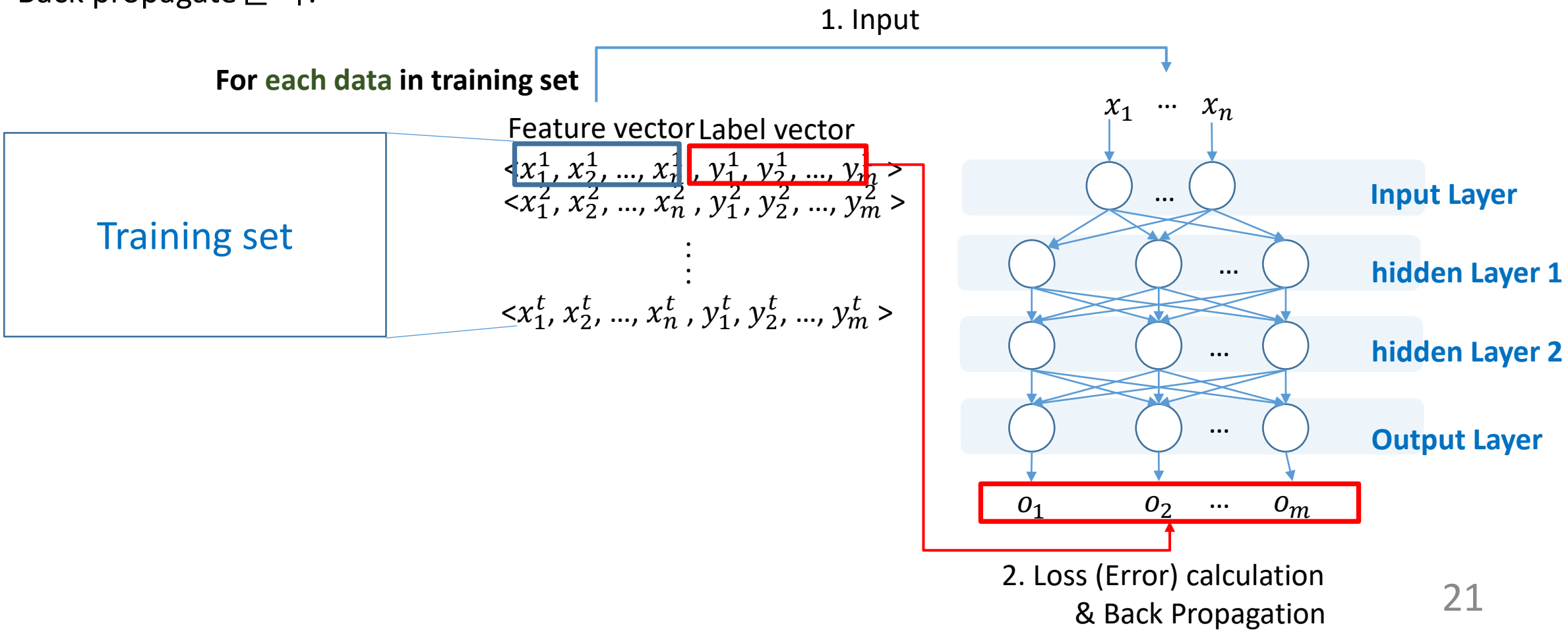
Training set, Evaluation set, Test set



- 위 각각의 set 사이에 중복되어 포함되는 데이터는 없다.
  - Training_set ∩ Validation_set = Φ AND Validation_set ∩ Test_set = Φ AND Test_set ∩ Training_set = Φ
- 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 1회 처음부터 끝까지 사용할 때 까지를 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?

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.

Stochastic Gradient Descent

- **Training Data에 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 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

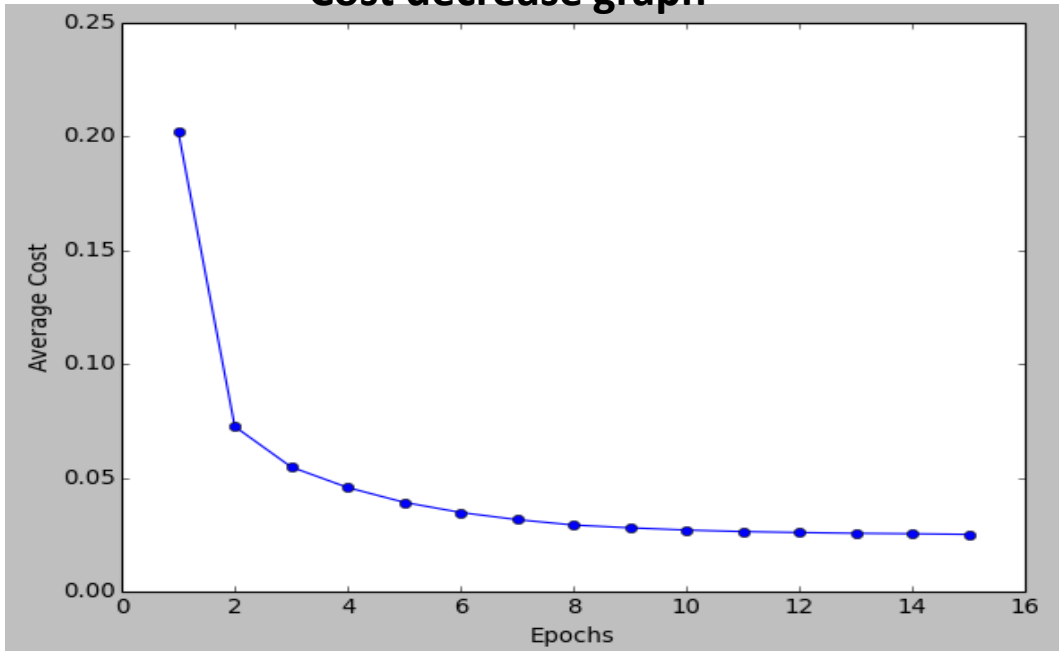


Image Source: https://www.bogotobogo.com/python/scikit-learn/scikit-learn_batch-gradient-descent-versus-stochastic-gradient-descent.php

Stochastic Gradient Descent
Cost decrease graph

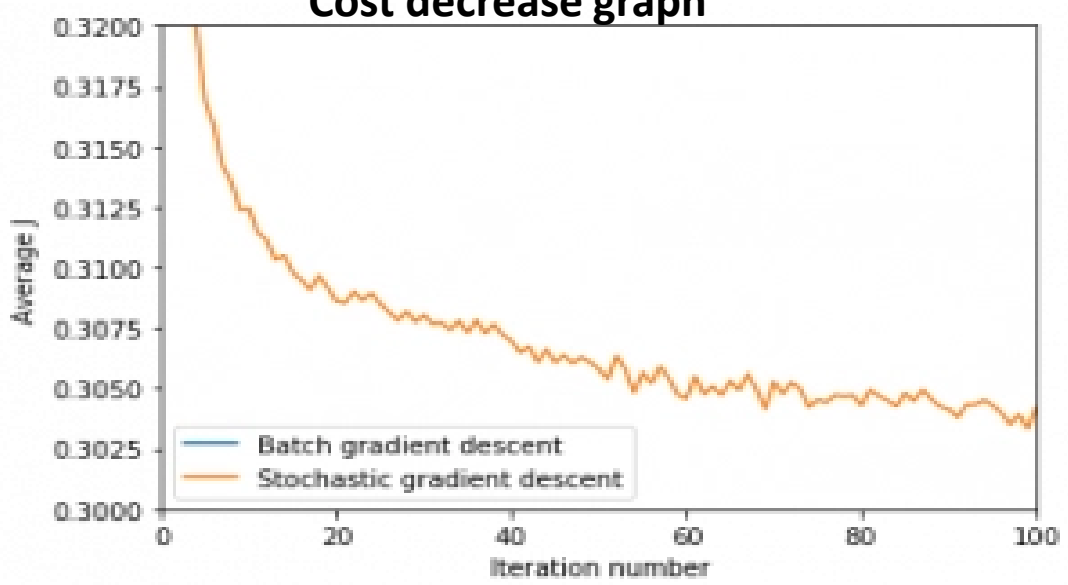


Image source :https://adventuresinmachinelearning.com/stochastic-gradient-descent/

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.**

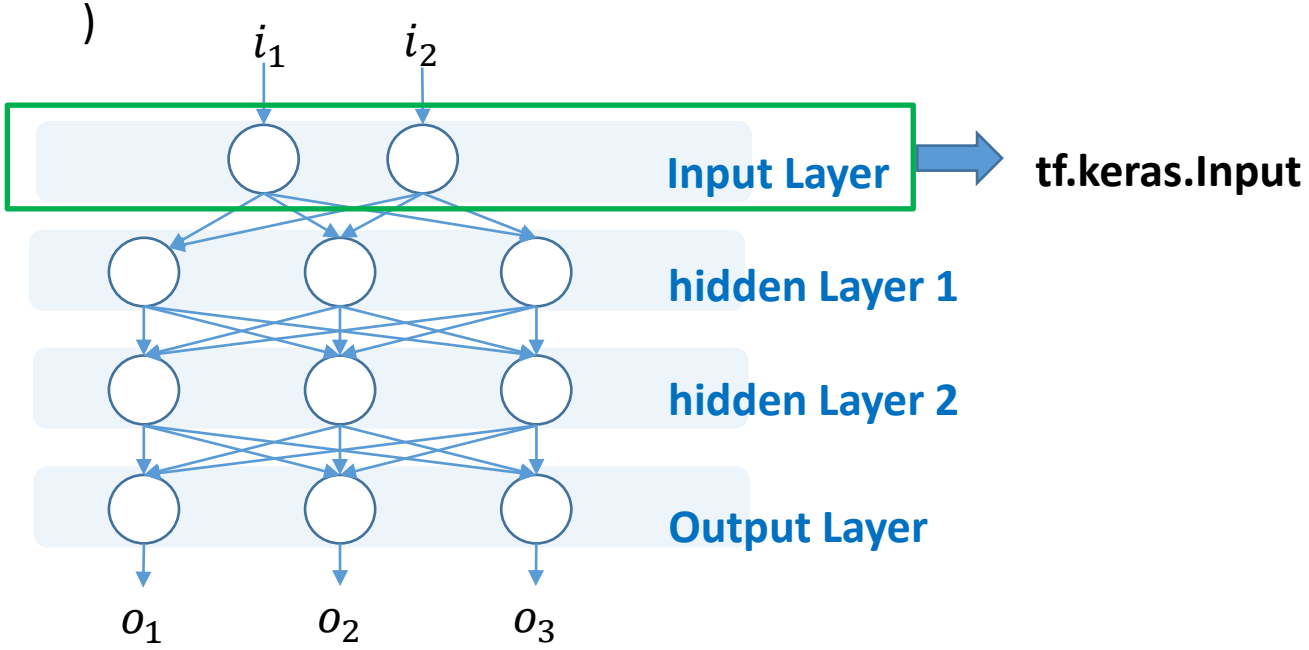
Mini batch gradient descent

- 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.

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,]**)

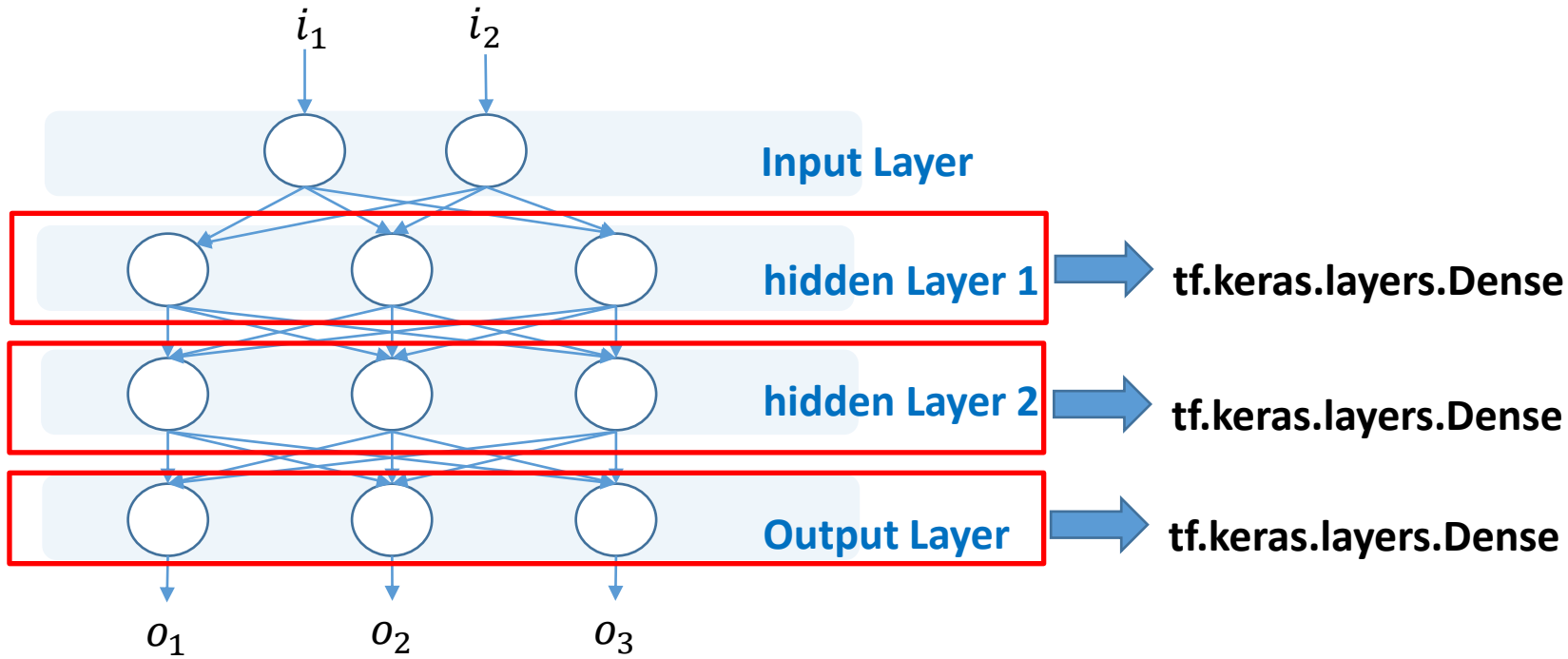
[2,] 와 같이 마지막을 , 로 비워 놓는 이유는 batch size가 아직 정의되지 않았기 때문이다.

- **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)

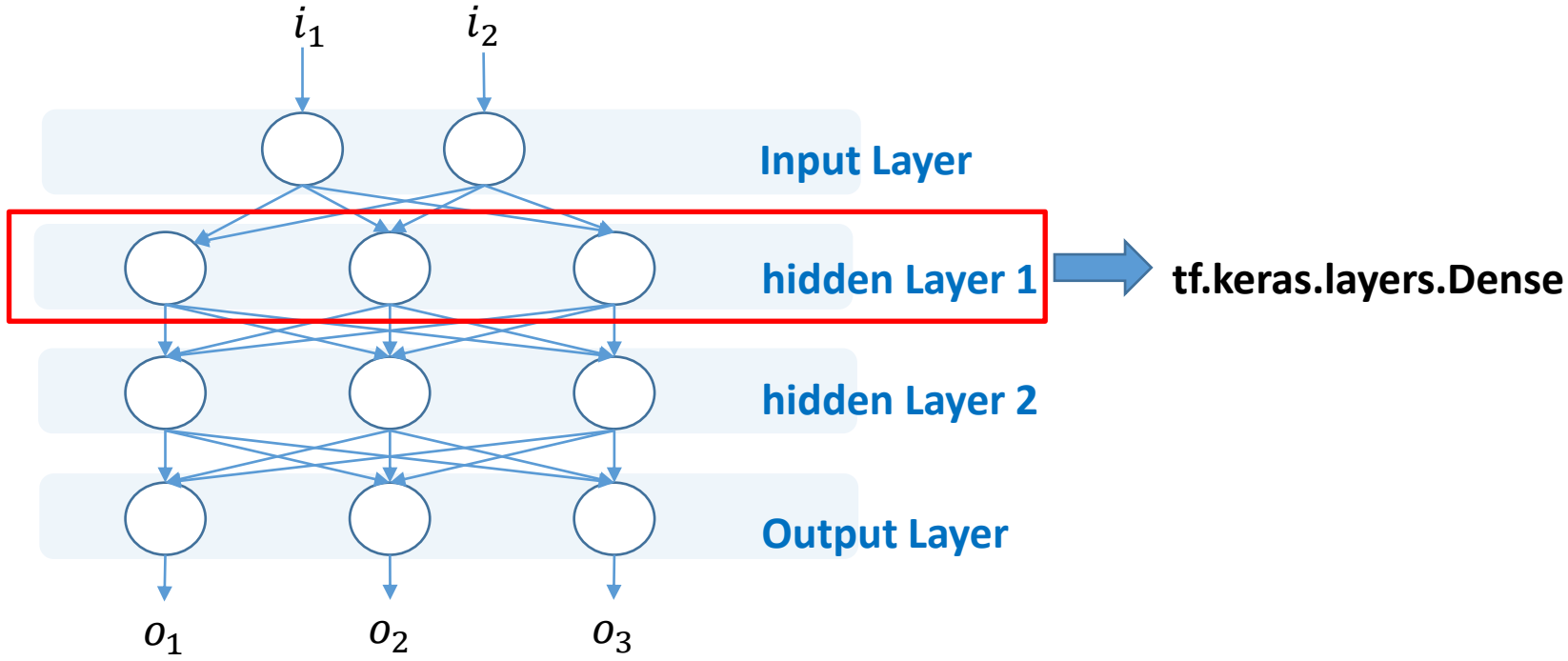


- Documentation
  - https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

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)



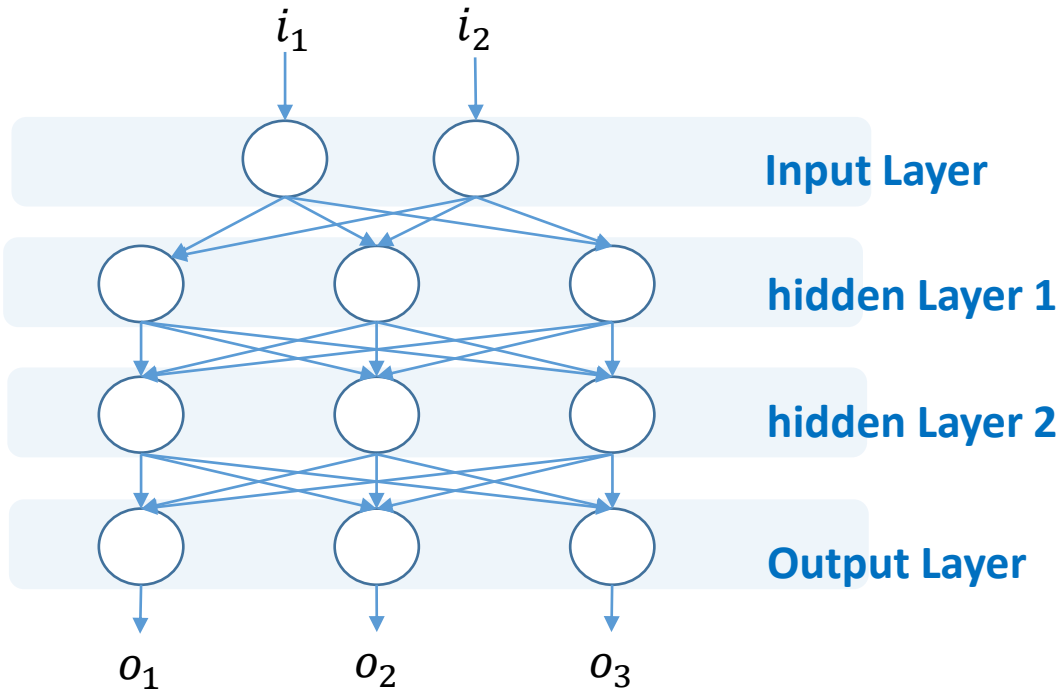
Keras : tf.keras.layers.Dense (Cont'd)

- 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 Layer 정의하는 Layer의 입력처럼 사용하여 Layer의 연결을 표현하는 API 형태.

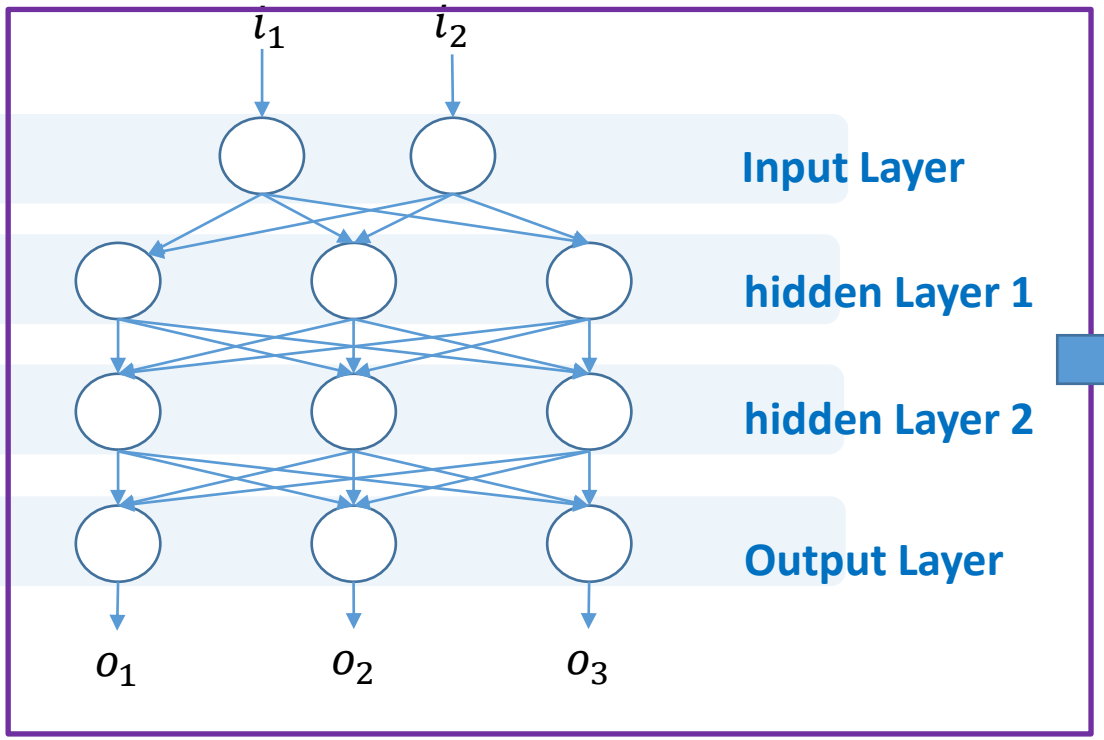



```
input_layer = tf.keras.Input(shape=[2, ])

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?

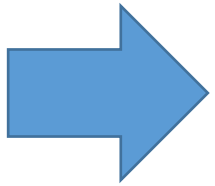
- Documentation
  - https://www.tensorflow.org/api_docs/python/tf/keras/Model

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

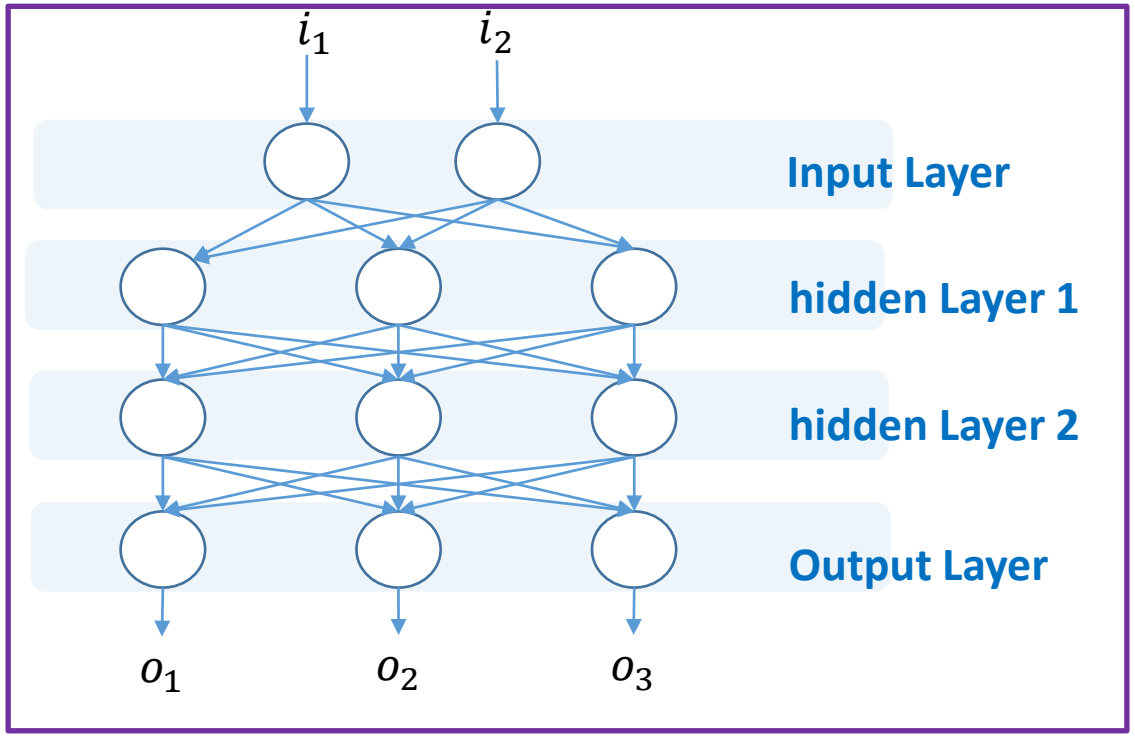
tf.keras.Model Member Function

- 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

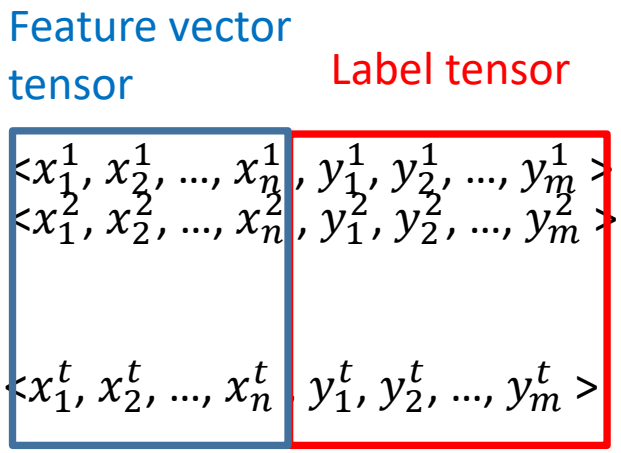
Model Fit

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
)
```

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

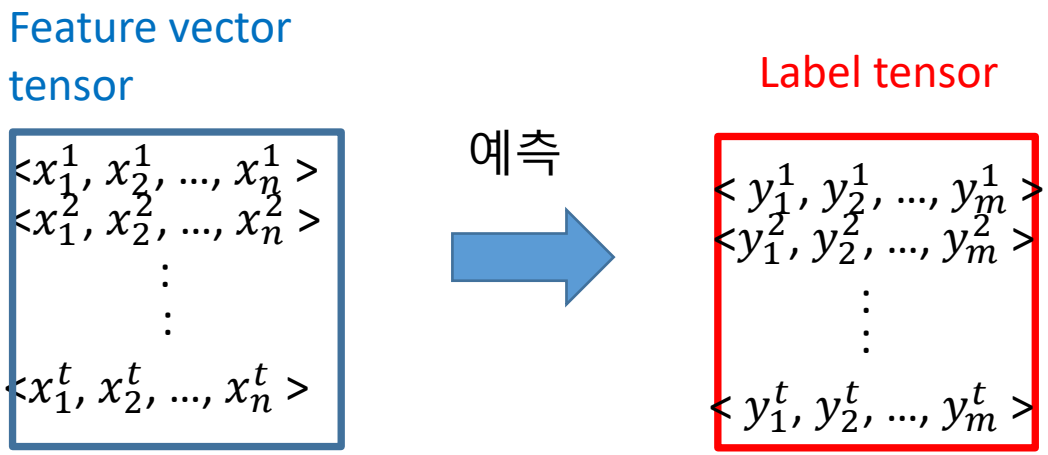
Training data가 위와 같이 파란 부분이 feature vector이고,
빨간 부분이 label인 형태라 할 때,
오른쪽 그림은 training dataset에 t개의 데이터가 존재할 때,
Feature vector tensor와 Label tensor를 나타낸다.



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
)
```



연습문제

- **and_classifier.py 을 변경하여 xor_classifer.py를 만들고 SLP와 MLP의 성능 차이를 관찰하라.**

Image Classification by using MLP

Image classification 문제

- **목표: Fashion-MNIST data를 사용하여 각 의류 이미지의 Class를 분류한다.**
- **Fashion-MNIST data**
  - https://github.com/zalandoresearch/fashion-mnist
  - Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

Data Description

Data : 28x28 grayscale images

Label

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

main.py

```
import tensorflow as tf
import matplotlib.pyplot as plt

def run_classifier():
    fashion_mnist = tf.keras.datasets.fashion_mnist
    (train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
    class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                   'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot’]

    print("Train data shape")
    print(train_images.shape)
    print("Train data labels")
    print(train_labels)
    print("Test data shape")
    print(test_images.shape)
    print("Test data labels")
    print(test_labels)
```

main.py (cont'd)

def run_classifier() 내용 이어서.

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()

train_images = train_images / 255.0
test_images = test_images / 255.0

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

classifier train and predict – begin

classifier train and predict – end

def run_classifier() 끝

```
if __name__ == "__main__":
    # execute only if run as a script
    run_classifier()
```

Network Design Process

- **1. 사용할 Network 구조를 선택한다.**
- **2. Input Layer를 정의한다.**
  - 입력 데이터 차원은 어떻게 정의할 것인가?
- **3. Output Layer를 정의한다.**
  - 출력 데이터 차원 (node 개수), Activation Function은 무엇을 사용할 것인가?
- **4. Objective Function을 정의한다.**
  - Model 학습은 어떤 함수의 값을 최소화 할 것인가?
- **5. Hidden Layer 층을 정의한다.**
  - Hidden Layer는 몇 층으로 할 것인가?
  - 각 Layer의 node 개수, Activation Function은 무엇을 사용할 것인가?
- **6. Optimization Algorithm을 선택한다.**
  - 예) Gradient Descent? ADAM?

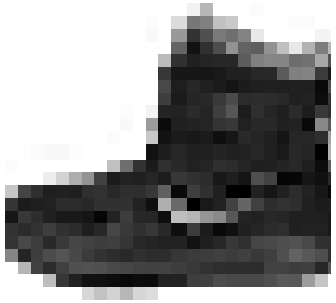
Network Design Process (Cont'd)

- **1. 사용할 Network 구조를 선택한다.**
  - 하려는 Task를 수행하기에 적합한 Network 구조를 선택하는 것이 좋다.
  - Image 를 사용한 분류이므로 Image에서 중요 feature를 추출하기 위한 popular한 구조 중 하나인 CNN (Convolutional Neural Network)를 사용할 수 도 있다.
  - 수업에서 배운 구조는 MLP 이므로 이번에는 우리가 알고 있는 MLP를 사용한다.
    - 분류 Task및 이미지의 형태가 상대적으로 단순하므로 MLP를 사용해도 어느 정도의 성능이 나온다고 기대하자.

Network Design Process (Cont'd)

- **2. Input Layer를 정의한다.**
  - 입력 데이터 차원은 어떻게 정의할 것인가?

분류하려는 이미지가 28x28 grayscale image 이므로 2 차원으로 표현된다.
따라서 입력 데이터의 shape는 28 by 28이 된다.

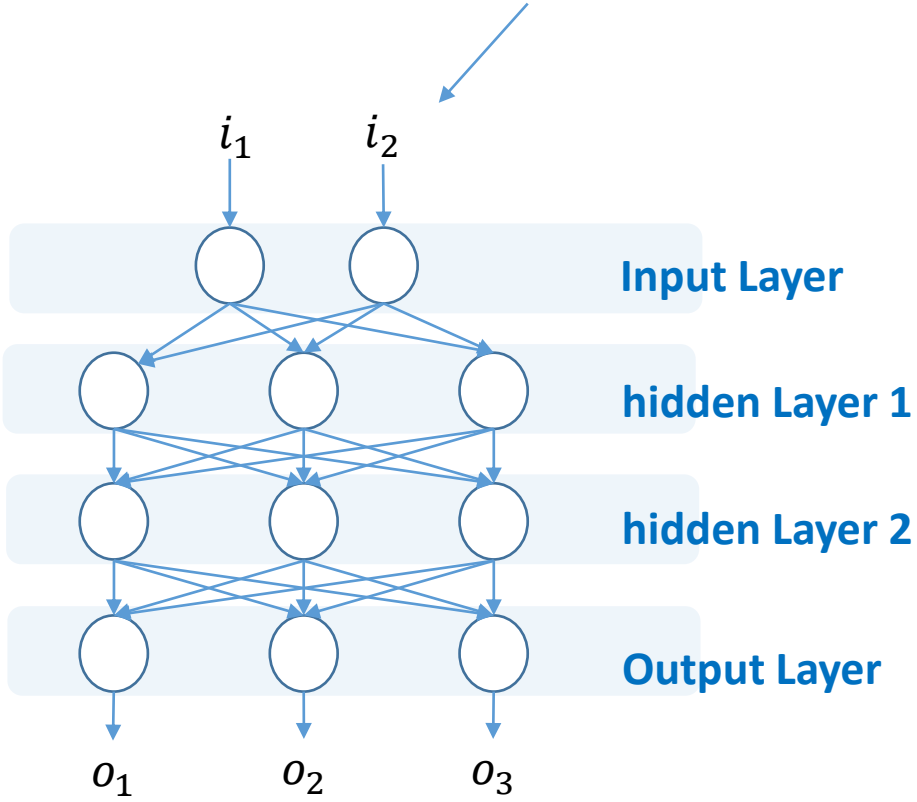


예)

```
input_layer = tf.keras.layers.Input(shape=[28, 28, ])
```

Network Design Process (Cont'd)

- **2. Input Layer를 정의한다. (Cont'd)**
  - 입력 데이터 차원은 어떻게 정의할 것인가?
  - 그런데 다음 layer는 Input layer의 node의 차원이 1차원이라 가정한다.
  - => **2 차원을 1 차원으로 변경**할 필요가 있다.



tf.keras.layers.Flatten(data_format=None, **kwargs)

입력으로 들어온 층을 1차원으로 변경.

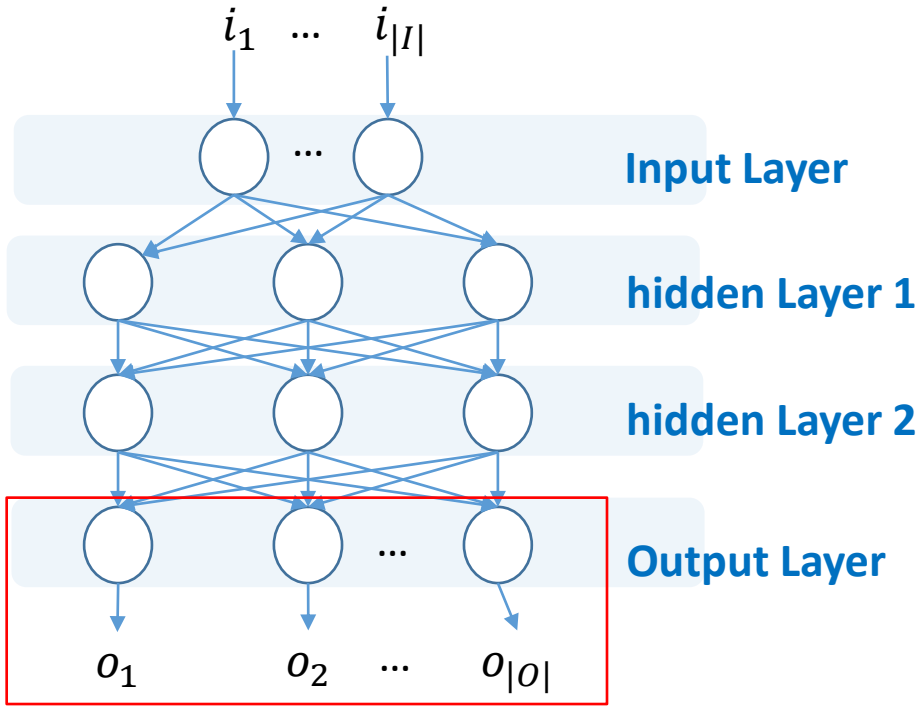
변경 방식: C에서 n-dimensional Array가 메모리에서 1차원으로 저장되는 것과 유사한 방식.

예)

```
input_layer = tf.keras.layers.Input(shape=[28, 28, ])
flatten_layer = tf.keras.layers.Flatten()(input_layer)
```

Network Design Process (Cont'd)

- **3. Output Layer를 정의한다.**
  - 출력 데이터 차원 (node 개수), Activation Function은 무엇을 사용할 것인가?



Label

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Label (Class)가 3개 이상 존재하므로 multi-class classification problem 이다.
그렇다면 출력 데이터의 차원은 얼마가 좋을까?

Network Design Process (Cont'd)

- **3. Output Layer를 정의한다.**
  - 출력 데이터 차원 (node 개수), Activation Function은 무엇을 사용할 것인가?

Output Layer의 Activation 함수는 softmax를 사용한다.

예)

```
ac_func_softmax = tf.keras.activations.softmax
output_layer = tf.keras.layers.Dense(units=self.num_labels, activation=ac_func_softmax)(hidden_layer_2)
```

Softmax Function

$\vec{c}$ 가 다음과 같은 k차원의 real vector라 하자.

$\vec{c}$=<c_1, c_2, …, c_k>

그러면 $\vec{c}$ 의 i 차원 element의 값에 대한 softmax 값은 다음과 같이 정의 된다.

$$softmax(c_i) = \frac{e^{c_i}}{\sum_{j=1}^{k} e^{c_j}}$$

$$\sum_{i=1}^{k} softmax(c_i) = 1$$

Softmax Function은 Sigmoid Function (=Logistic Function)을 muli-dimension으로 generalization한 형태이다.

Softmax Function (Cont'd)

Why Softmax?

- 우리가 알고 싶은 것은 주어진 데이터가 각 category (class)에 속할 확률.
- 확률은 [0.0, 1.0] 의 범위.
- 데이터를 x, class 집합$C = \{c_1, c_2, \ldots, c_{|C|}\}$ 라하고, x가 class c_i에속할 확률을 $P_{x,i}$ 라 하면, $\sum_{c_i \in C} P_{x,i}$=1.0

Softmax Function is **often used as the last activation function of a neural network** to normalize the output of a network to a probability distribution over predicted output classes.

In probability theory, the output of the Softmax function can be **used to represent a categorical distribution** (A probability distribution over K different possible outcomes)

Network Design Process (Cont'd)

- Classifer 가 **Binary Classifier라면 output layer의 activation function은 softmax 대신 sigmoid를 사용**할 수 있다.
  - Binary Classifier라면 Class 개수가 2개이다.
  - 따라서 $\vec{c}$ 가 다음과 같은 2차원의 real vector라 하자.
    - $\vec{c}$=<c_1, c_2>
  - Binary Classifier는 하나의 class의 확률을 알면 다른 하나는 자동으로 알게 되므로 output dimension이 하나이다.
  - 따라서 Output의 값을 $\vec{c}$=<c_1, c_2> 에서 c_1 이라 하면, c_2 는 classifier의 출력이 없으므로 0 이라 할 수 있다.
    - $\vec{c}$=<c_1, 0>
  - $\vec{c}$=<c_1, 0> 일 때 $softmax(c_1)$ 의 값을 계산해 보라

Network Design Process (Cont'd)

- **4. Objective Function을 정의한다.**
  - Model 학습은 어떤 함수의 값을 최소화 할 것인가?
  - **Cross Entropy Loss**를 사용해 본다.
    - 이번 예에서는 Cross Entropy Loss의 variation 중 하나인 tf.keras.losses.SparseCategoricalCrossentropy를 사용한다.

```
tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=False, reduction=losses_utils.ReductionV2.AUTO,
    name='sparse_categorical_crossentropy'
)
```

from_logits=False (default 값이 False)로 되어 있으면, 입력으로 확률 값 [0.0, 1.0] 이 들어오는 것을 가정하고 Crossentropy를 계산함.
from_logits=True 라면, 입력값이 확률값이 아니라 가정하고 sigmoid or softmax함수를 사용하여 확률값으로 변경한 다음에 Crossentropy를 계산함.

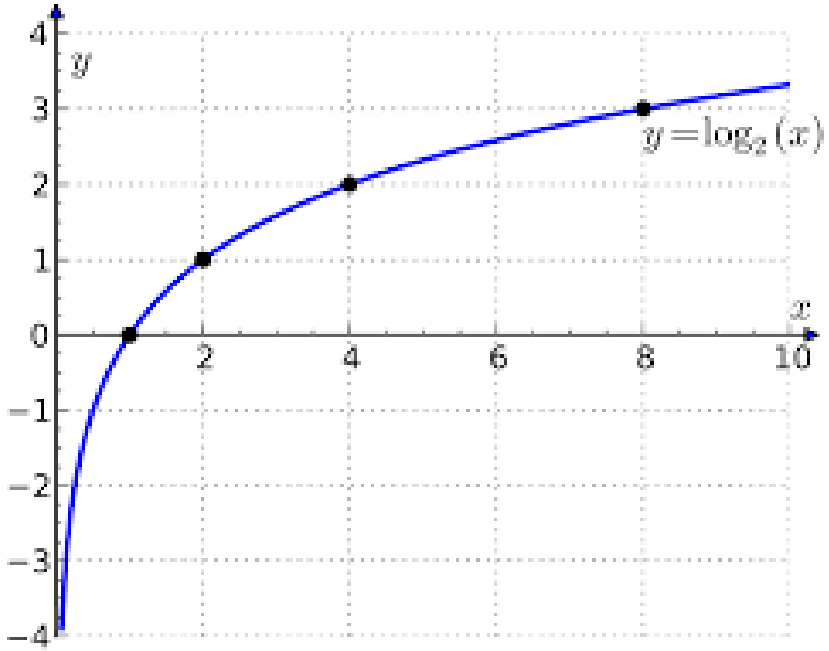
Cross-Entropy Loss (Multinomial (Categorical) Cross Entropy Loss)

$P = < p_1, p_2, \ldots, p_k >$ 가 데이터 x 가 k 개의 class 각각에 속할 **정답 확률(label vector)**이라 하고, $Q = < q_1, q_2, \ldots, q_k >$가 x 가 k 개의 class 각각에 속할 **예측 확률(predicted label vector)하 하면,** P 와 Q 사이의 Cross Entropy는 다음 식과 같다.

$$H_{cr}(P, Q) = -\sum_{i=1}^{k} p_i \cdot \log_2 q_i$$

정답 class를 잘 맞출 수록 $H_{cr}(P, Q)$ 의 값은 작아진다.

P = <1.0, 0.0>, Q = <0.5, 0.5> 일 경우와
P = <1.0, 0.0>, Q = <1.0, 0.0> 일 경우의
$H_{cr}(P, Q)$ 를 계산해 보라.



Binary Cross Entropy Loss

- **Cross-Entropy Loss의 Binary Classifier 특화 버전**

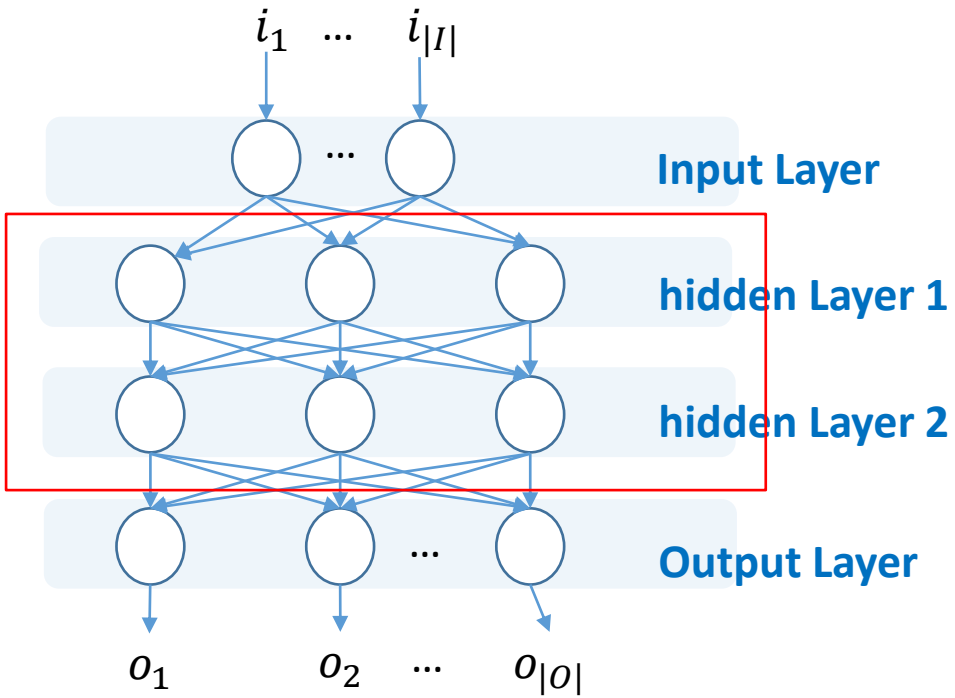
p_1 이 데이터 x 가 class 1에 속할 **정답 확률(label vector)**이라 하고, q_1 이 x 가 class1에 속할 **예측 확률(predicted label vector)이라 하면,** 정답과 예측 사이의 Binary Cross Entropy는 다음 식과 같다.

$$H_{cr}(P,Q) = -p_1 \cdot \log_2 q_1 - (1-p_1) \cdot \log_2(1-q_1)$$

위 식을 일반 Cross-Entropy의 식으로부터 유도해 보아라.

Network Design Process (Cont'd)

- **5. Hidden Layer 층을 정의한다.**
  - Hidden Layer는 몇 층으로 할 것인가?
  - 각 Layer의 node 개수, Activation Function은 무엇을 사용할 것인가?



이번 예에서는 Hidden Layer는 2 층, 각 Layer에 node 수 128개, Activation Function은 ReLU로 설정해 본다.
앞의 **Layer 수, node 수, Activation Function등의 설정은 일반적으로 cross-validation을 여러 번 해 보면서 최적의 값을 찾아야** 한다.

예)

```
ac_func = tf.keras.activations.relu
    hidden_layer_1 = tf.keras.layers.Dense(units=128, activation=
ac_func)(flatten_layer)
    hidden_layer_2 = tf.keras.layers.Dense(units=128, activation=
ac_func)(hidden_layer_1)
```

Network Design Process (Cont'd)

- **6. Optimization Algorithm을 선택한다.**
  - Adam Optimizer을 사용해 본다.

예) 모델 정의 전체 예.

```
input_layer = tf.keras.layers.Input(shape=[self.img_shape_x, self.img_shape_y, ])
flatten_layer = tf.keras.layers.Flatten()(input_layer)

ac_func_relu = tf.keras.activations.relu
hidden_layer_1 = tf.keras.layers.Dense(units=128, activation=ac_func_relu)(flatten_layer)
hidden_layer_2 = tf.keras.layers.Dense(units=128, activation=ac_func_relu)(hidden_layer_1)

ac_func_softmax = tf.keras.activations.softmax
output_layer = tf.keras.layers.Dense(units=self.num_labels, activation=ac_func_softmax)(hidden_layer_2)
classifier_model = tf.keras.Model(inputs=input_layer, outputs=output_layer)

opt_alg = tf.keras.optimizers.Adam(learning_rate=0.001)
loss_cross_e = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False)
classifier_model.compile(optimizer=opt_alg, loss=loss_cross_e, metrics=['accuracy'])
```

모델 학습

- fit() 함수와 data를 사용하여 모델을 Training해 보라.
  - Epoch 는 10 으로 한다.

ImageClassifier.py

```
class ImageClassifier:
    def __init__(self, img_shape_x, img_shape_y, num_labels):
        super(ImageClassifier, self).__init__()
        self.img_shape_x = img_shape_x
        self.img_shape_y = img_shape_y
        self.num_labels = num_labels
        self.classifier = None


    def fit(self, train_imgs, train_labels, num_epochs):
        self.classifier.fit(train_imgs, train_labels, epochs=num_epochs)

    def predict(self, test_imgs):
        predictions = self.classifier.predict(test_imgs)
        return predictions
```

ImageClassifier.py (Cont'd)

```
# class ImageClassifier 의 member function

def configure_model(self):
    input_layer = tf.keras.layers.Input(shape=[self.img_shape_x, self.img_shape_y, ])
    flatten_layer = tf.keras.layers.Flatten()(input_layer)

    ac_func_relu = tf.keras.activations.relu
    hidden_layer_1 = tf.keras.layers.Dense(units=128, activation=ac_func_relu)(flatten_layer)
    hidden_layer_2 = tf.keras.layers.Dense(units=128, activation=ac_func_relu)(hidden_layer_1)

    ac_func_softmax = tf.keras.activations.softmax
    output_layer = tf.keras.layers.Dense(units=self.num_labels, activation=ac_func_softmax)(hidden_layer_2)
    classifier_model = tf.keras.Model(inputs=input_layer, outputs=output_layer)

    opt_alg = tf.keras.optimizers.Adam(learning_rate=0.001)
    loss_cross_e = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False)
    classifier_model.compile(optimizer=opt_alg, loss=loss_cross_e, metrics=['accuracy'])

    self.classifier = classifier_model
```

Model Prediction

- main.py에서 ImageClassifier를 import하고,
  - from ImageClassifier import ImageClassifier
- Class ImageClassifier를 사용해서 model train 및 test(predict)를 구현해 보아라.
- Test(predict)할 때는 다음 코드를 참고한다.

```
predicted_labels = my_classifier.predict(test_imgs=test_images)
predicted_labels = tf.math.argmax(input=predicted_labels, axis=1)

plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(test_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[predicted_labels[i]])
plt.show()
```

- tf.math.argmax가 어떤 동작을 하는지
- 특히 input parameter “axis”의 값이 무엇을 의미하는지,
- https://www.tensorflow.org/api_docs/python/tf/math/argmax
- 를 찾아보고 실제 debugger을 통해 결과를 확인해 보라.

결과

- 잘 되었다면 다음과 같은 분류 결과를 볼 수 있다.