Deep Learning and TensorFlow

Machine Learning

Introduction to TensorFlow

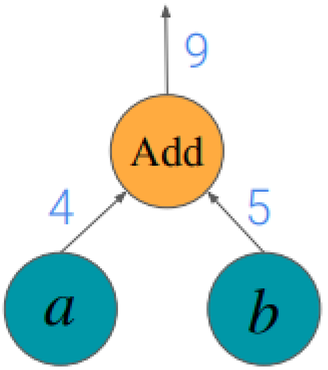
- TensorFlow
  - TensorFlow is an open source platform for machine learning
    - Originally developed by Google Brain team to conduct machine learning and deep neural networks research
  - It has a comprehensive ecosystem of tools, libraries and resources
    - Lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications
  - TensorFlow computations are expressed as dataflow graphs
  - It can run on multiple CPUs and GPUs
- TensorFlow 2.0
  - Introduced a number of simplifications
  - Improvements to the performance on GPU

https://www.tensorflow.org/

Introduction to TensorFlow

- Basic Code Structure
  - View functions as computational graphs
  - TensorFlow = Tensor + Flow = Data + Flow
  - First build a computational graph, and then use session to execute operations in the graph

    → Basic approach, there is also dynamic approach implemented in the recently introduced **eager mode**
  - Nodes are operators(ops), variables, and constants



https://www.cs.tau.ac.il/~joberant/teaching/advanced_nlp_spring_2018/

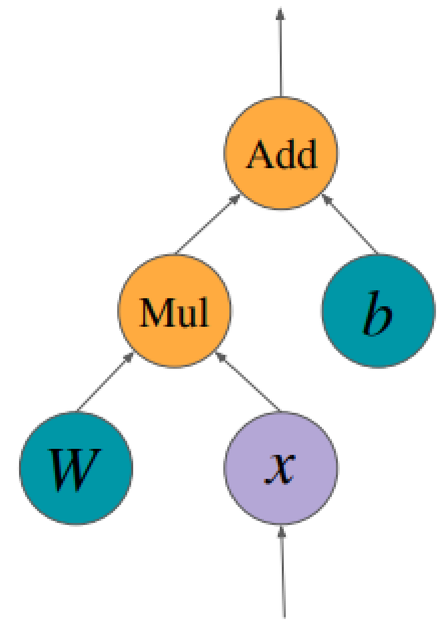
Introduction to TensorFlow

- Basic Code Structure - Graphs
  - Constants are fixed value tensors – not trainable
  - Variables are tensors initialized in a session – trainable
  - Placeholders are tensors of values that are unknown during the graph construction, but passed as input during a session
  - Ops are functions on tensors

Ops

$$\hat{y} = Wx + b$$

Variables Placeholders

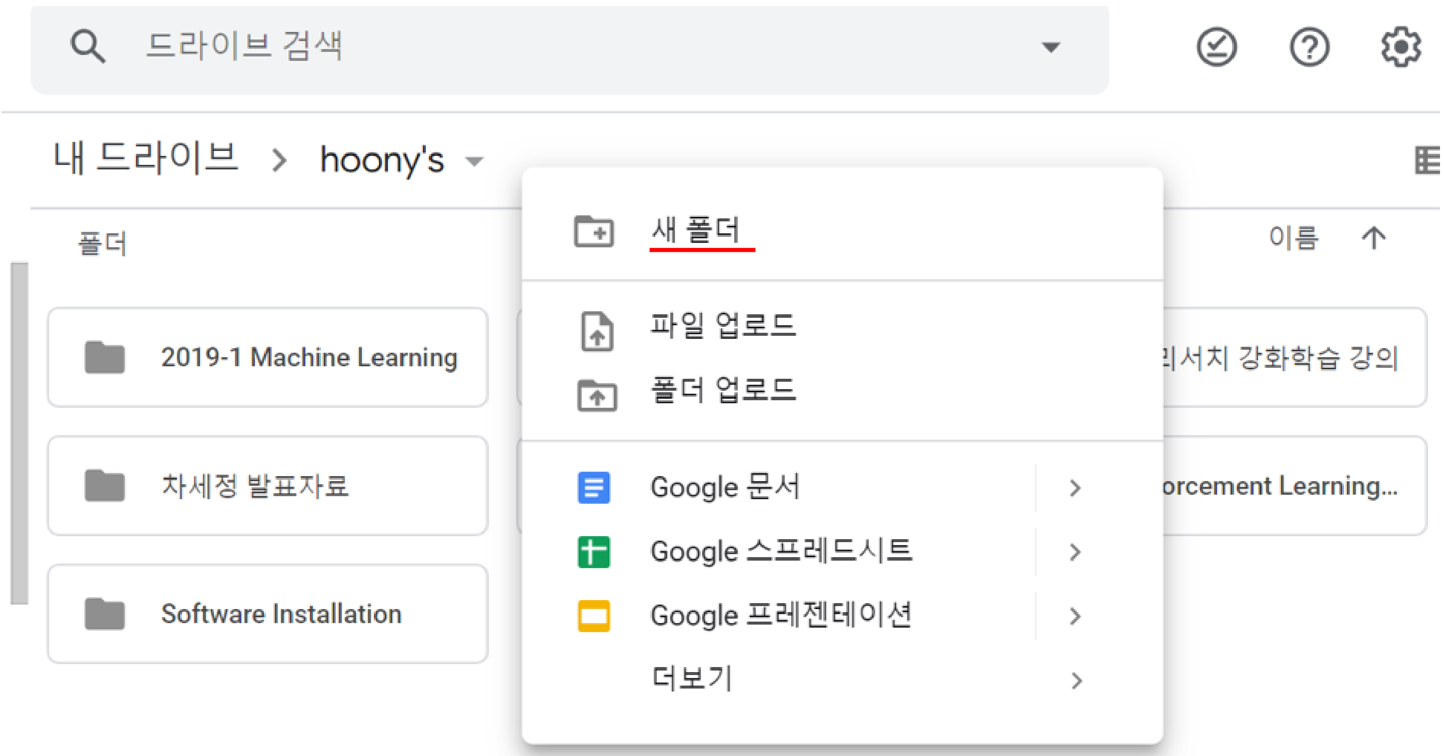


Introduction to TensorFlow

- Introduction to Google Colab.(Colaboratory)
  - You can develop deep learning applications with Google Colab – on the free GPU and TPU – using Keras, TensorFlow and Pytorch
  - Google Colab is a free cloud service and it supports free GPU

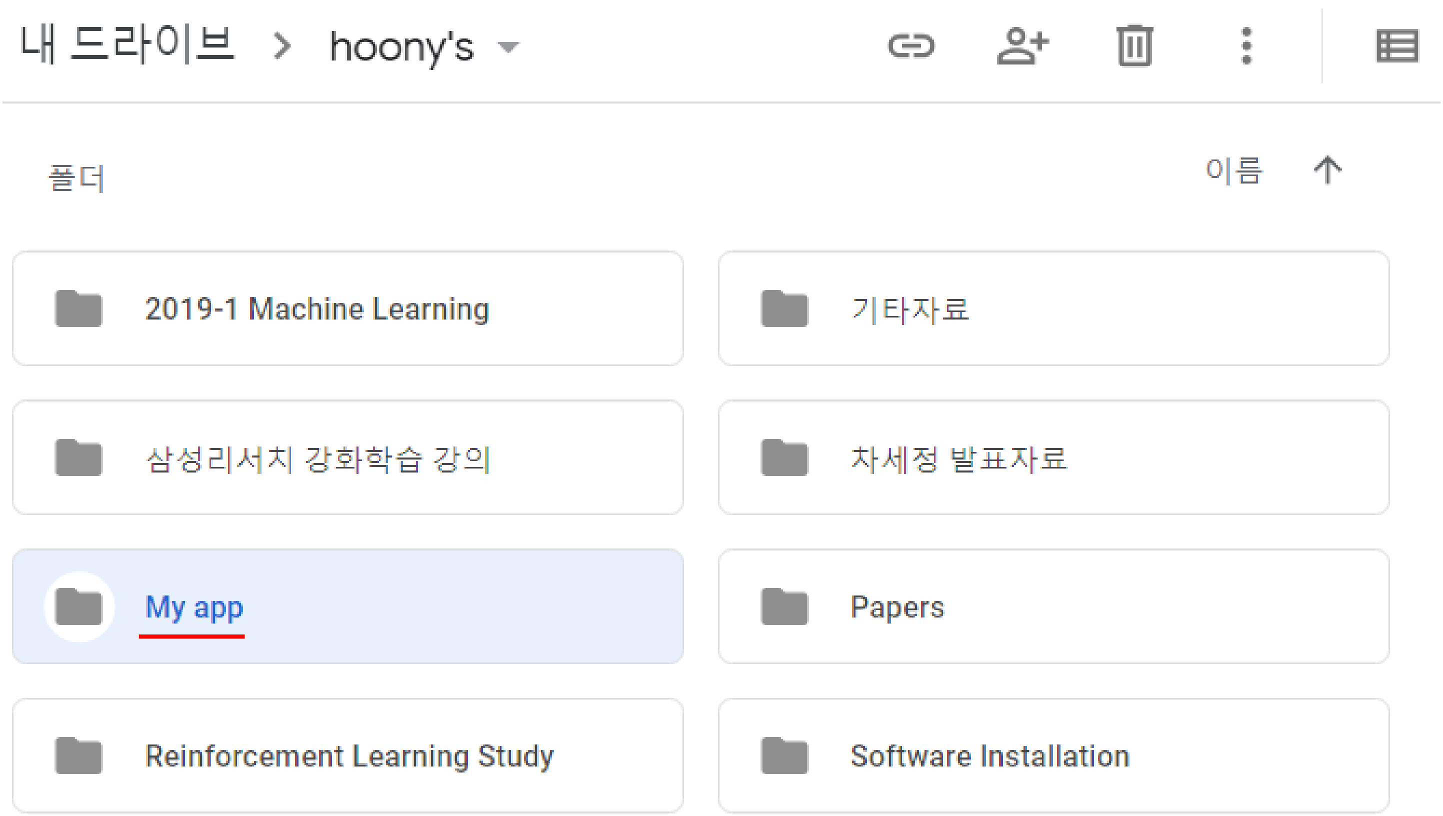
Introduction to TensorFlow

- How to use
  1. Creating your Folder on Google Drive



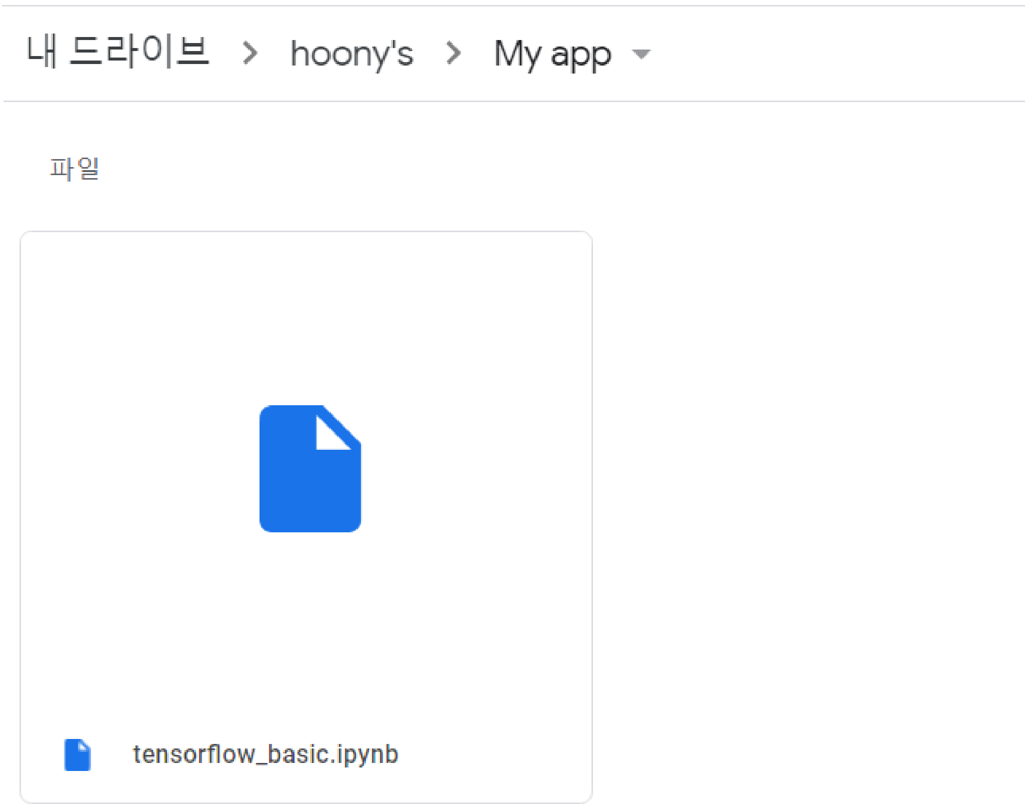
Introduction to TensorFlow

- How to use
  1. Creating your Folder on Google Drive



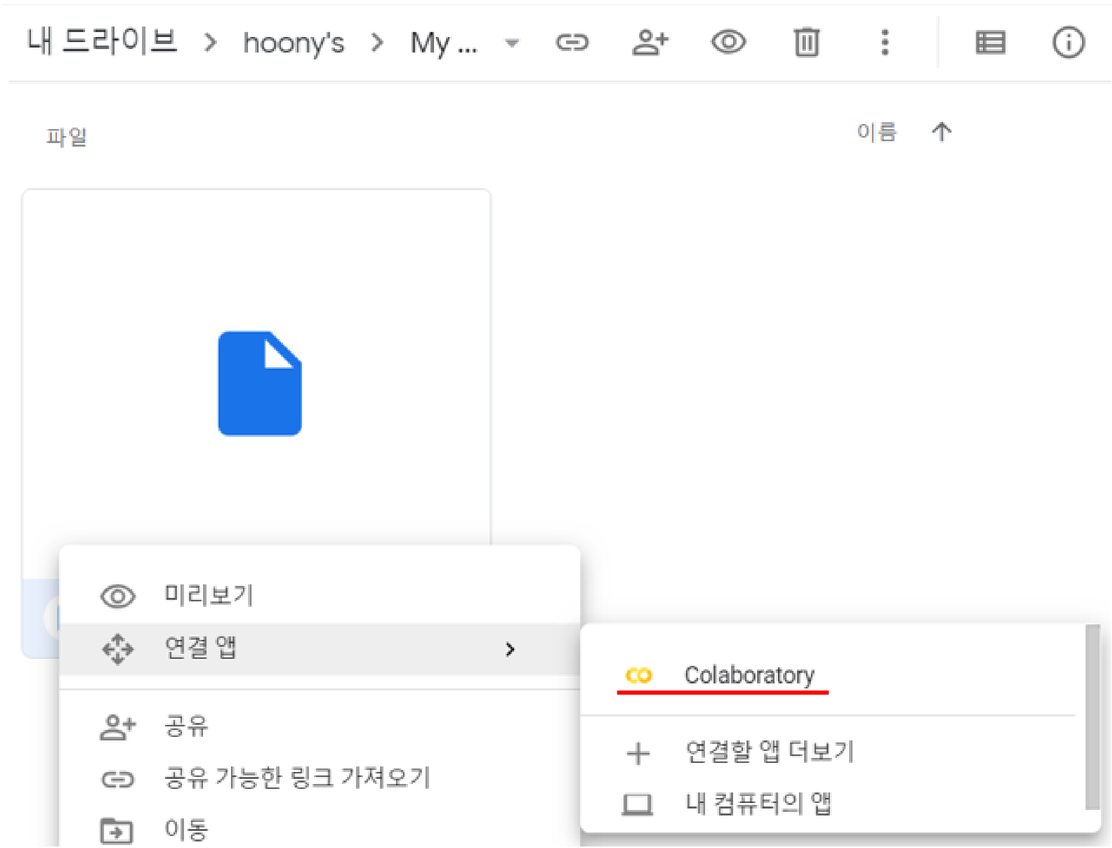
Introduction to TensorFlow

- How to use
  2. Upload your ipynb file



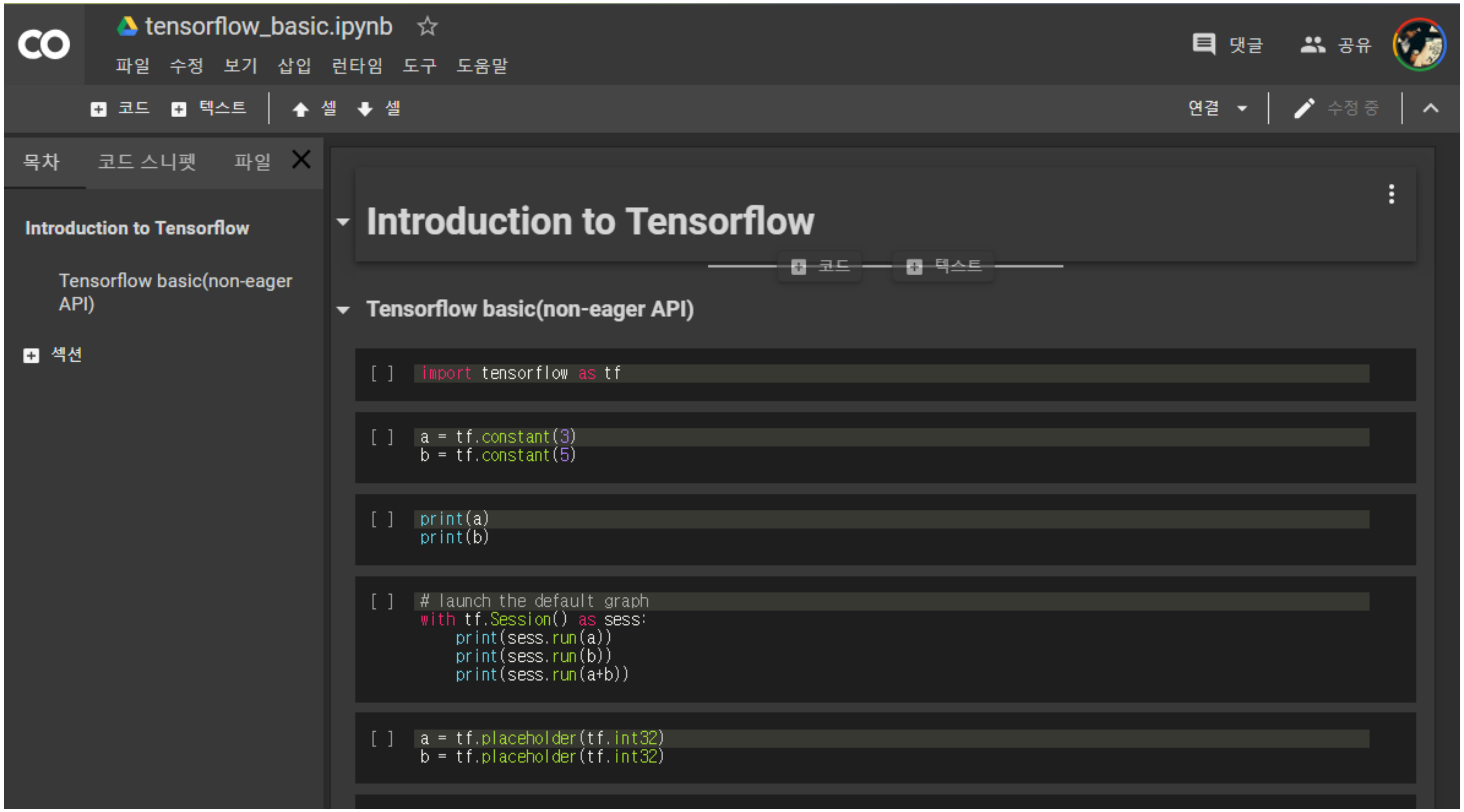
Introduction to TensorFlow

- How to use
  3. Click the Colaboratory(Place cursor in the file and click right mouse button)



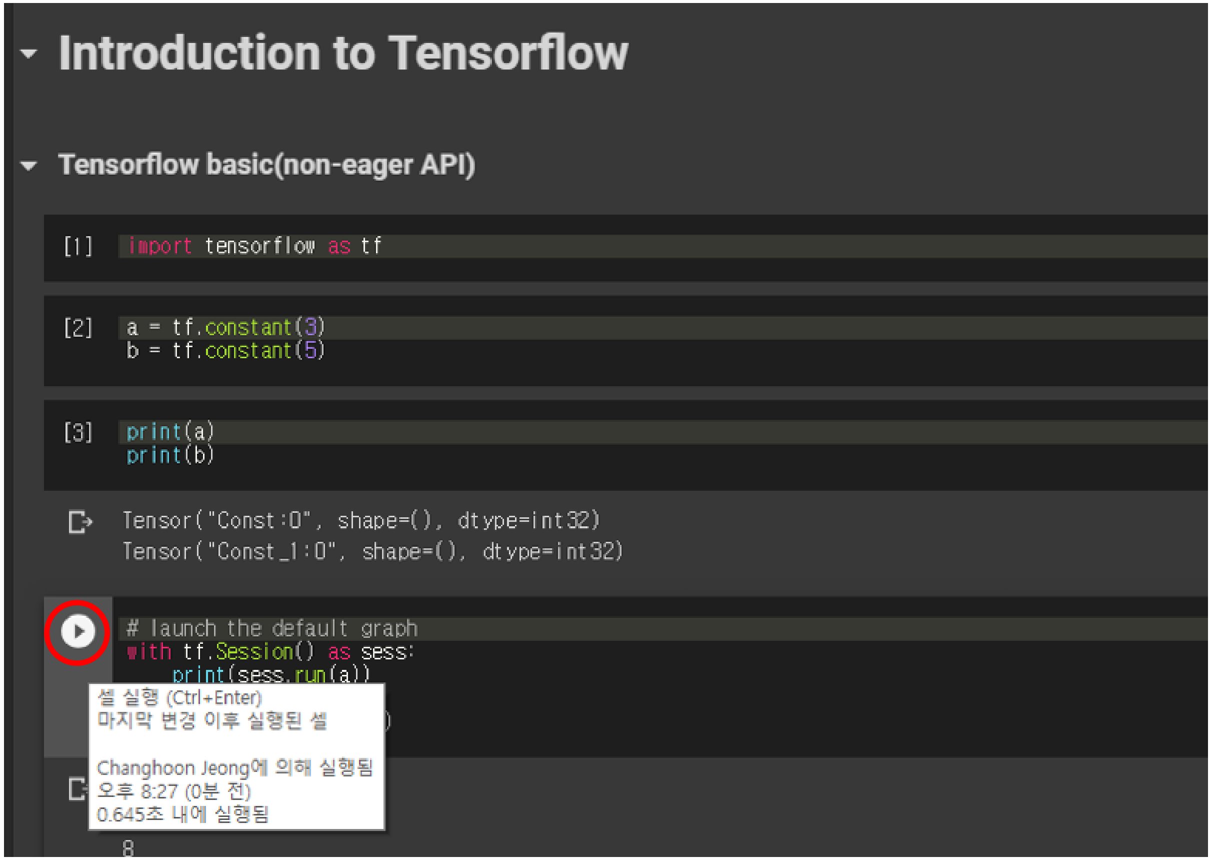
Introduction to TensorFlow

- How to use
  4. Now you can see the Google Colab Notebook!



Introduction to TensorFlow

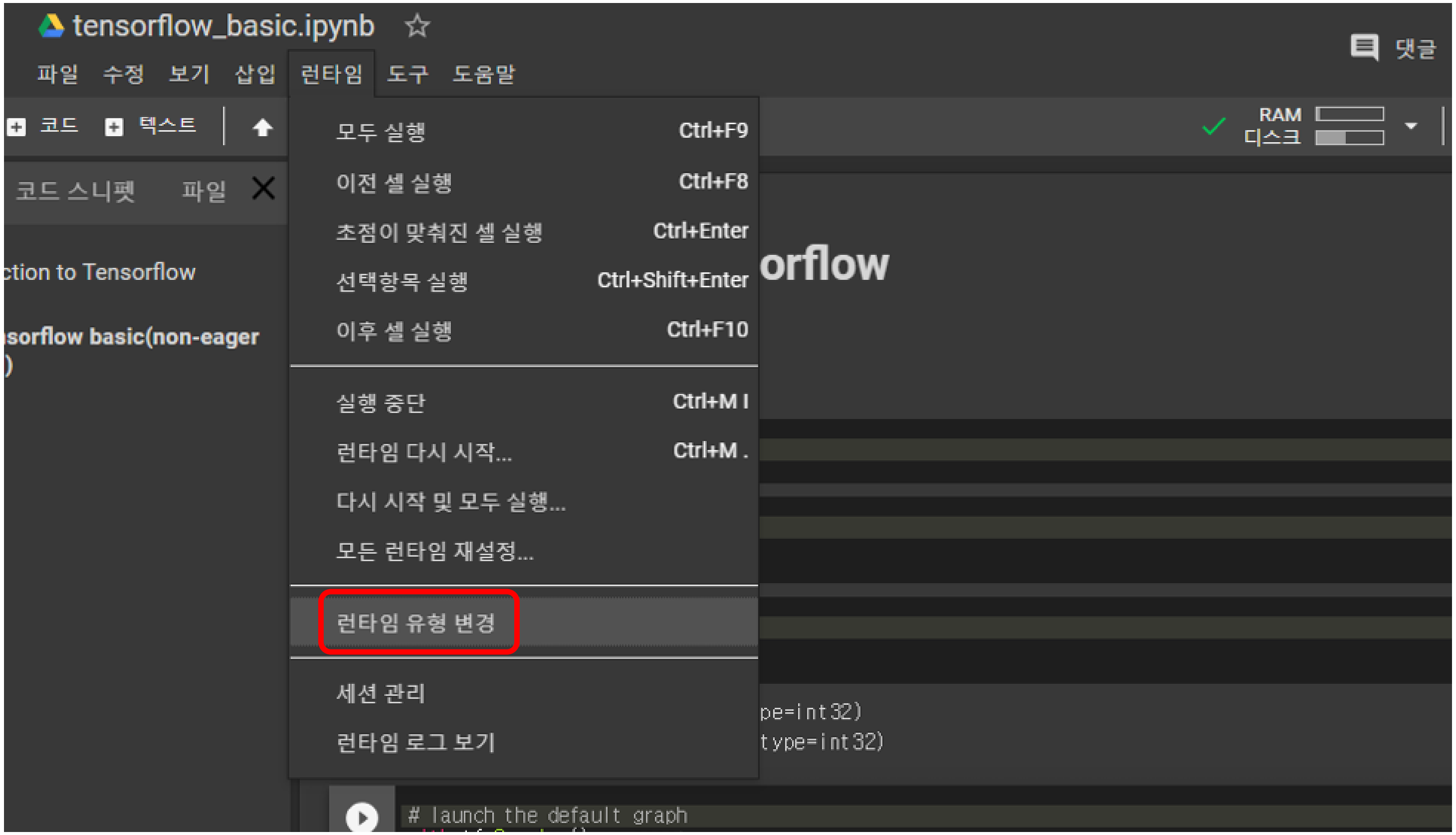
- How to use
  4. You can run the Cell by click the arrow(or ctrl+enter)



Introduction to TensorFlow

- How to use

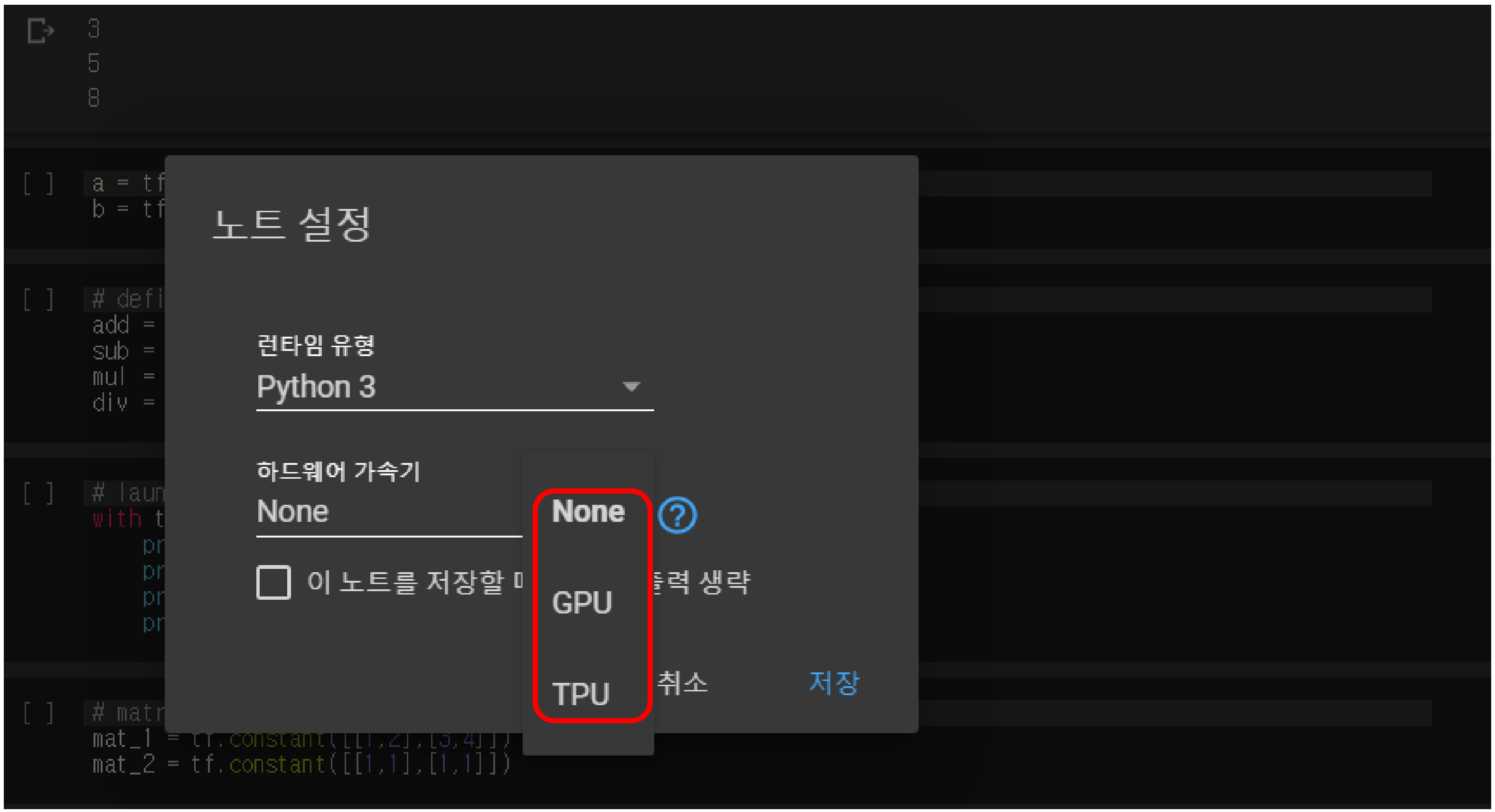
    5. Wanna use GPU or TPU? Click the “runtime”, and select “change runtime type”



Introduction to TensorFlow

- How to use

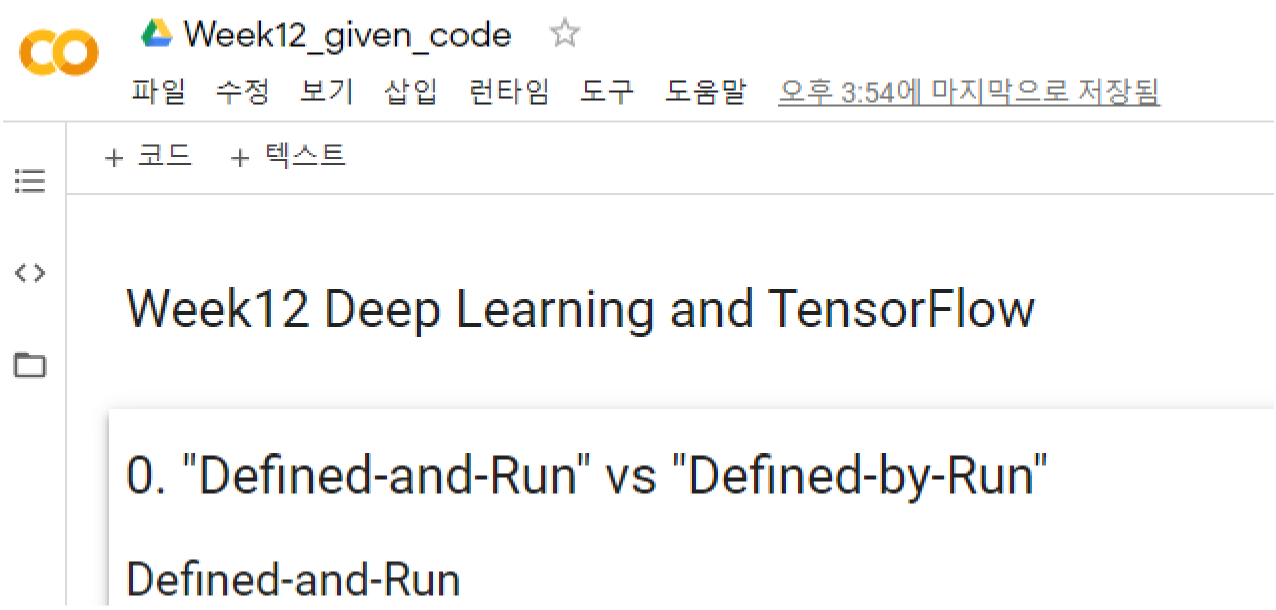
  6. Choose your Hardware Accelerator



Introduction to TensorFlow

- Code template
  - Today, the code template is provided by bellow link
  - Copy this file to your google drive and run this file

    https://colab.research.google.com/drive/1t69rCFqfZYJab5QbJYJT_0HNiAEdFIRQ?usp=sharing



TensorFlow Basic with Session(Non-Eager API)

- Import TensorFlow

```
import numpy as np
import matplotlib.pyplot as plt
import time
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior() # Run the tensorflow like v1.x
```

- Costant

```
a = tf.constant(3)
b = tf.constant(5)
# launch the graph
sess = tf.Session()
print(sess.run(a+b))
```

```
b'Hello, TensorFlow!'
```

```
# constants
mat_1 = tf.constant([[1,2],[3,4]])
mat_2 = tf.constant([[1,1],[1,1]])
# define operations
mat_product = tf.matmul(mat_1, mat_2)
# launch the graph
with tf.Session() as sess:
print(sess.run(mat_product))
```

```
[[3 3]
 [7 7]]
```

TensorFlow Basic with Session(Non-eager API)

- Placeholder & Operations

```
# placeholders
a = tf.placeholder(tf.int32)
b = tf.placeholder(tf.int32)

# operations
add = tf.add(a, b)
mul = tf.multiply(a, b)

# launch the graphs
with tf.Session() as sess:
print(sess.run(add, feed_dict={a:3, b:5})) # Feeding input data into placeholder
print(sess.run(mul, feed_dict={a:3, b:5}))
```

```
8
15
```

TensorFlow Basic with Session(Non-eager API)

- Example : Linear Regression with TF(Non-eager API)
  - Birth-Life Dataset

```
Country Birth rate   Life expectancy
Vietnam 1.822   74.828243902
Vanuatu 3.869   70.819487805
Tonga   3.911   72.150658537
Timor-Leste 5.578   61.999853659
Thailand    1.579   73.927658537
Solomon Islands 4.229   67.465195122
Singapore   1.15    81.641463415
Samoa   3.86    72.306390244
Philippines 3.142   68.484317073
Papua New Guinea    3.951   62.440609756
New Zealand 2.16    80.702439024
New Caledonia   2.141   76.301682927
Myanmar 2.002   64.662097561
Mongolia    2.504   68.194975610
Micronesia  3.451   68.764829268
Malaysia    2.635   74.024560976
```

TensorFlow Basic with Session(Non-eager API)

- Example : Linear Regression with TF(Non-eager API)
  - Load the Birth-Life Dataset

```
import pandas as pd
path_to_file = tf.keras.utils.get_file('birth_life_2010.txt',
'https://github.com/uzay00/KaVe/raw/master/2018/Lecture9/\
tf%20code/birth_life_2010.txt')
data = pd.read_csv(path_to_file, sep="\t")
data.head()
```

```
Downloading data from https://github.com/uzay00/KaVe/raw/master/2018/Lectu
8192/5324 [==============================================] - 0s 0us/step
```

	Country	Birth rate	Life expectancy
0	Vietnam	1.822	74.828244
1	Vanuatu	3.869	70.819488
2	Tonga	3.911	72.150659

TensorFlow Basic with Session(Non-eager API)

- Example : Linear Regression with TF(Non-eager API)
  - Load the Birth-Life Dataset

```
birth_rate = data["Birth rate"].values
life_exp = data["Life expectancy"].values

data = list(zip(birth_rate, life_exp))
data = np.asarray(data, dtype=np.float32)

data.shape
```

```
(190, 2)
```

TensorFlow Basic with Session(Non-eager API)

- Example : Linear Regression with TF(Non-eager API)
  - Build the model
    - Variables : tensors initialized in a session - trainable

```
# placeholders
X = tf.placeholder(tf.float32, name='X')
Y = tf.placeholder(tf.float32, name='Y')

# variables
w = tf.get_variable('weights', initializer=tf.constant(0.0))
b = tf.get_variable('bias', initializer=tf.constant(0.0))

Y_predicted = w * X + b
loss = tf.square(Y - Y_predicted, name='loss')
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(loss)
```

TensorFlow Basic with Session(Non-eager API)

- Example : Linear Regression with TF(Non-eager API)
  - Train the model

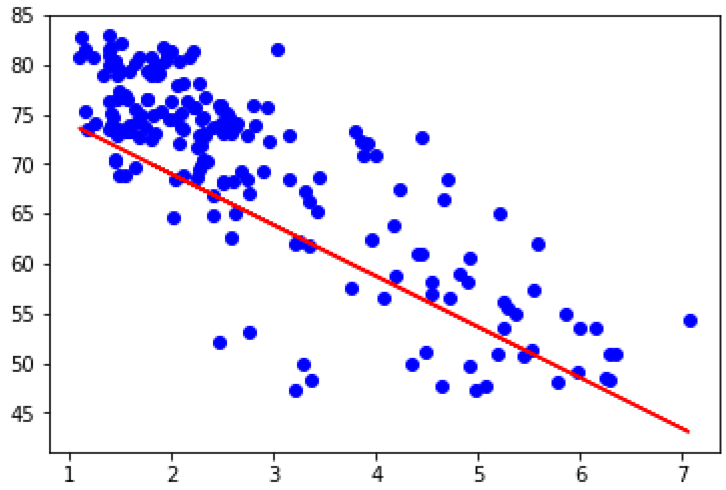
```
n_samples = len(data)

with tf.Session() as sess:
     sess.run(tf.global_variables_initializer())

     # 20 epoch
     for i in range(20):
          total_loss = 0

          # SGD
          for x, y in data:
               _, l = sess.run([optimizer, loss], feed_dict={X : x, Y : y})
               total_loss += l

     print("Epoch {0}: {1}".format(i, total_loss / n_samples))

     w_out, b_out = sess.run([w, b])
```

```
Epoch 0: 490.15910439699593
          ⋮
Epoch 18: 28.49622749929886
Epoch 19: 28.481358767657775
```

TensorFlow Basic with Session(Non-eager API)

- Example : Linear Regression with TF(Non-eager API)
  - Plot the result

```
plt.plot(data[:,0], data[:,1], 'bo')
plt.plot(data[:,0], data[:,0] * w_out + b_out, 'r')
plt.show()
```



TensorFlow Basic without Session(Eager API)

- TensorFlow Using High-level API(Eager API; Keras)

TensorFlow > Learn > TensorFlow Core > Guide

Keras overview

Contents

Run in Google Colab | View source on GitHub | Download notebook

This guide gives you the basics to get started with Keras. It's a 10-minute read.

Import tf.keras

`tf.keras` is TensorFlow's implementation of the Keras API specification. This is a high-level API to build and train models that includes first-class support for TensorFlow-specific functionality, such as eager execution, `tf.data` pipelines, and Estimators. `tf.keras` makes TensorFlow easier to use without sacrificing flexibility and performance.

To get started, import `tf.keras` as part of your TensorFlow program setup:

Standardizing on Keras: Guidance on High-level APIs in TensorFlow 2.0

TensorFlow Follow
Dec 7, 2018 · 8 min read

Posted by Sandeep Gupta, Josh Gordon, and Karmel Allison on behalf of the TensorFlow team

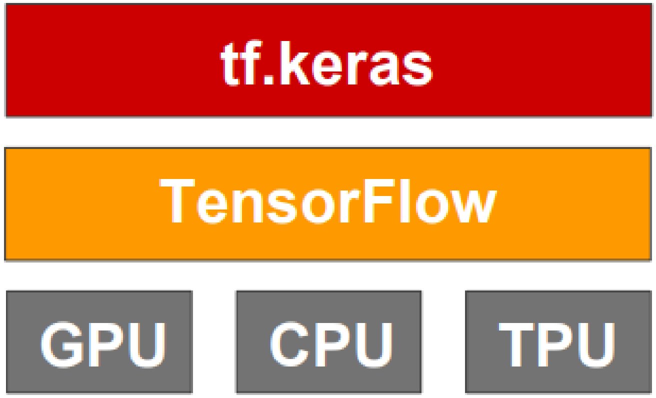
TensorFlow is preparing for the release of version 2.0. In this article, we want to preview the direction TensorFlow's high-level APIs are heading, and answer some frequently asked questions.

Keras is an extremely popular high-level API for building and training deep learning models. It's used for fast prototyping, state-of-the-art research, and production. While TensorFlow supports Keras today, with 2.0, we are integrating Keras more tightly into the rest of the TensorFlow platform.

https://www.tensorflow.org/

TensorFlow Basic without Session(Eager API)

- Keras is the official high-level API of TensorFlow
  - tensorflow.keras module
  - Part of core TensorFlow since v1.4
  - Full Keras API
  - Better optimized for TF
  - A focus on user experience
  - Better integration with TF-specific features
    - Estimator API
    - Eager execution
    - etc.



https://web.stanford.edu/class/cs20si/lectures/

TensorFlow Basic without Session(Eager API)

- Import TensorFlow

```
import pandas as pd
import numpy as np
import time
import matplotlib.pyplot as plt
import tensorflow as tf
```

- Eager Execution

```
a = tf.constant(3)
b = tf.constant(5)
print(a + b)
```

```
Tensor("add:0", shape=(), dtype=int32)
```

```
mat_1 = tf.constant([[1,2],[3,4]])
mat_2 = tf.constant([[1,1],[1,1]])
print(tf.matmul(mat_1, mat_2))
```

```
Tensor("MatMul:0", shape=(2, 2), dtype=int32)
```

TensorFlow Basic without Session(Eager API)

- Example : Linear Regression with Keras
  - Load the Birth-Life Dataset

```
path_to_file = tf.keras.utils.get_file('birth_life_2010.txt',
'https://github.com/uzay00/KaVe/raw/master/2018/Lecture9/\
tf%20code/birth_life_2010.txt')
data = pd.read_csv(path_to_file, sep="\t")
data.head()
```

```
Downloading data from https://github.com/uzay00/KaVe/raw/master/2018/Lectu
8192/5324 [==============================================] - 0s 0us/step
```

	Country	Birth rate	Life expectancy
0	Vietnam	1.822	74.828244
1	Vanuatu	3.869	70.819488
2	Tonga	3.911	72.150659

TensorFlow Basic without Session(Eager API)

- Example : Linear Regression with TF(Eager API)
  - Load the Birth-Life Dataset

```
birth_rate = data["Birth rate"].values
life_exp = data["Life expectancy"].values

data = list(zip(birth_rate, life_exp))
data = np.asarray(data, dtype=np.float32)

Data.shape
```

```
(190, 2)
```

TensorFlow Basic without Session(Eager API)

- Example : Linear Regression with TF(Eager API)
  - Build the model

```
# build a sequential model
layers = tf.keras.layers

model = tf.keras.Sequential([
        layers.Dense(units=1, input_shape=[1])
        ])
```

```
# config the model
optimizer = tf.keras.optimizers.SGD(0.01)
model.compile(loss='mse',
optimizer=optimizer)
```

TensorFlow Basic without Session(Eager API)

- Example : Linear Regression with TF(Eager API)
  - Train the model

```
model.fit(birth_rate, life_exp, epochs=20, batch_size=1)
```

```
Train on 190 samples
Epoch 1/20
190/190 [==============================] - 0s 1ms/sample - loss: 933.1663
Epoch 2/20
190/190 [==============================] - 0s 1ms/sample - loss: 224.8654
Epoch 3/20
190/190 [==============================] - 0s 1ms/sample - loss: 78.1038

                    ⋮

Epoch 19/20
190/190 [==============================] - 0s 1ms/sample - loss: 34.9142
Epoch 20/20
190/190 [==============================] - 0s 1ms/sample - loss: 36.5489
<tensorflow.python.keras.callbacks.History at 0x7f9cc6551400>
```

TensorFlow Basic without Session(Eager API)

- Example : Linear Regression with TF(Eager API)
  - Plot the result

```
w_out = model.layers[0].kernel[0]
b_out = model.layers[0].bias

plt.plot(data[:,0], data[:,1], 'bo')
plt.plot(data[:,0], data[:,0] * w_out + b_out, 'r')
plt.show()
```

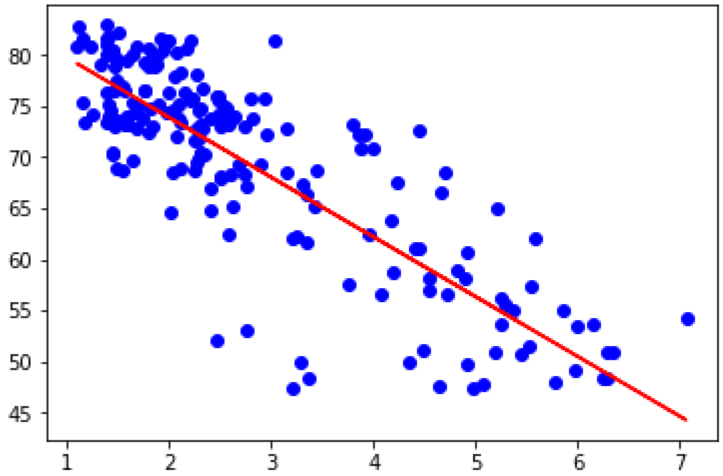


Image Classification using Deep NN with Keras

- There are two ways to build your model with Keras
  - Sequential API(for Beginners)
    - tf.keras.Sequential
    - Stacking layer objects in Sequential object

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

  - Subclassing API(for Experts)
    - tf.keras.Model
    - Defining your model as class by subclassing tf.keras.Model

```
class MyModel(tf.keras.Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.conv1 = Conv2D(32, 3, activation='relu')
    self.flatten = Flatten()
    self.d1 = Dense(128, activation='relu')
    self.d2 = Dense(10, activation='softmax')

  def call(self, x):
    x = self.conv1(x)
    x = self.flatten(x)
    x = self.d1(x)
    return self.d2(x)
model = MyModel()

with tf.GradientTape() as tape:
  logits = model(images)
  loss_value = loss(logits, labels)
grads = tape.gradient(loss_value, model.trainable_variables)
optimizer.apply_gradients(zip(grads, model.trainable_variabl
```

Image Classification using Deep NN with Keras

- Fashion MNIST Dataset

0 T-shirt/top

1 Trouser

2 Pullover

3 Dress

4 Coat

5 Sandal

6 Shirt

7 Sneaker

8 Bag

9 Ankle boot

Like MNIST, Fashion MNIST has 60,000 training sets and 10,000 test sets(28x28 pixels)

Image Classification using Deep NN with Keras

- Sequential API(for beginners)
  - Import TensorFlow

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

  - Load and prepare the Fashion MNIST dataset

```
fashion_mnist = tf.keras.datasets.fashion_mnist

(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
print(X_train.shape)
```

```
(60000, 28, 28)
```

```
# show the image data 0
plt.figure()
plt.imshow(X_train[0])
plt.colorbar()
plt.show()
```

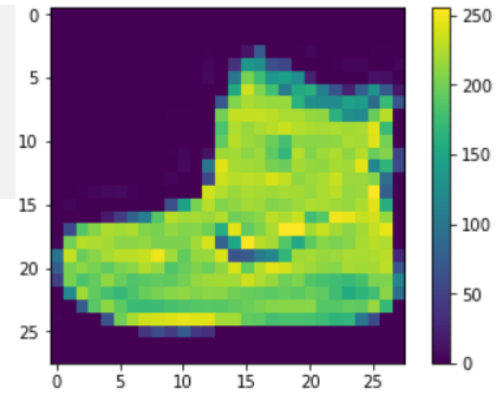


Image Classification using Deep NN with Keras

- Sequential API(for beginners)
  - Load and prepare the Fashion MNIST dataset

```
# Normalizing
X_train, X_test = X_train / 255.0, X_test / 255.0
```

```
# class labels
y_train
```

```
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

```
# names for class labels
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
class_names[y_train[0]]
```

```
'Ankle boot'
```

```
# show first 25 data and label
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(X_train[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[y_train[i]])
plt.show()
```

Image Classification using Deep NN with Keras

- Sequential API(for beginners)
  - Load and prepare the Fashion MNIST dataset

```
# show first 25 data and label
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(X_train[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[y_train[i]])
plt.show()
```

Image Classification using Deep NN with Keras

- Sequential API(for beginners)
  - [*Recap] : Scikit-learn Multi-Layer Perceptron

```
from sklearn.neural_network import MLPClassifier
# build the model with 2 hidden layers
mlp = MLPClassifier(hidden_layer_sizes=(128, 128),
                    activation='relu', verbose=1, max_iter=20)
```

```
# flatten the data
X_train_1d = X_train.reshape(60000, 784)
X_test_1d = X_test.reshape(10000, 784)
```

```
# checking the execution time
import time
start_time = time.time()

# training the model
mlp.fit(X_train_1d, y_train)

print("Time : ", time.time()-start_time)
```

```
Iteration 1, loss = 0.56137671
Iteration 2, loss = 0.39974874
⋮

Iteration 20, loss = 0.18294887
Time :  47.72472381591797
```

Image Classification using Deep NN with Keras

- Sequential API(for beginners)
  - [*Recap] : Scikit-learn Multi-Layer Perceptron

```
# Train accuracy
mlp.score(X_train_1d, y_train)
```

```
0.93725
```

```
# Test accuracy
mlp.score(X_test_1d, y_test)
```

```
0.8885
```

Image Classification using Deep NN with Keras

- Sequential API(for beginners)
  - Build the Neural Network model

```
# build the model with 3 fully connected layer
layers = tf.keras.layers

model = tf.keras.Sequential([
     layers.Flatten(input_shape=(28, 28)),
     layers.Dense(128, activation='relu'),
     layers.Dense(128, activation='relu'),
     layers.Dense(10, activation='softmax'), # output layer])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
```

Image Classification using Deep NN with Keras

- Sequential API(for beginners)
  - Compile and Train the model

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
# Checking the execution time
start_time = time.time()

# Training the model
model.fit(X_train, y_train, epochs=20, batch_size=200)
print("Time : ", time.time()-start_time)
```

```
Epoch 1/20
300/300 [==============================] - 1s 4ms/step - loss: 1.3159 - accuracy: 0.5529

          ⋮

Epoch 20/20
300/300 [==============================] - 1s 3ms/step - loss: 0.3761 - accuracy: 0.8663
Time :  20.95208764076233
```

Image Classification using Deep NN with Keras

- Sequential API(for beginners)
  - Evaluate the model

```
loss, acc = model.evaluate(X_test, y_test)
print('Test Loss : %.4f' % loss)
print('Test Accuracy : %.4f' % acc)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.4152 - accuracy: 0.8510
Test Loss : 0.4152
Test Accuracy : 0.8510
```

  - Make prediction

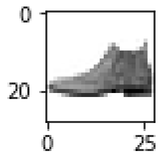
```
predictions = model.predict(X_test)
predictions
```

```
array([[9.83967425e-07, 3.64516985e-08, 5.38087818e-07, ...,
        8.74551237e-02, 2.34868703e-03, 7.64938354e-01],
       [1.72184285e-04, 1.81206706e-04, 9.21737254e-01, ...,
        1.28630801e-22, 8.36566251e-05, 3.77870800e-15],
       [3.00144366e-05, 9.99850869e-01, 6.26571045e-06, ...,
        2.20926294e-12, 9.49819068e-09, 6.89293403e-13],
       ...,
```

Image Classification using Deep NN with Keras

- Sequential API(for beginners)
  - Make prediction

```
plt.figure(figsize=(1, 1))
plt.imshow(X_test[0], cmap=plt.cm.binary)
plt.show()
```



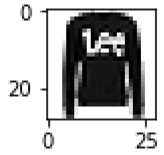

```
y_predicts = np.argmax(predictions[0])
print('True lable = %s' % class_names[y_test[0]])
print('Predicted = %s' % class_names[y_predicts])
```

```
True lable = Ankle boot
Predicted = Ankle boot
```

Image Classification using Deep NN with Keras

- Sequential API(for beginners)
  - Make prediction

```
plt.figure(figsize=(1, 1))
plt.imshow(X_test[1], cmap=plt.cm.binary)
plt.show()
```




```
y_predicts = np.argmax(predictions[1])
print('True lable = %s' % class_names[y_test[1]])
print('Predicted = %s' % class_names[y_predicts])
```

```
True lable = Pullover
Predicted = Pullover
```

Image Classification using Deep NN with Keras

- Subclassing API(for expert)
  - Import TensorFlow

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras import Model
```

  - Load and prepare the Fashion MNIST dataset

```
fashion_mnist = tf.keras.datasets.fashion_mnist

(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0
```

  - DataLoader : Load numpy array

```
train_ds = tf.data.Dataset.from_tensor_slices(
           (X_train, y_train)).shuffle(10000).batch(200)
test_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test)).batch(200)
```

Image Classification using Deep NN with Keras

- Subclassing API(for expert)
  - Build the Neural Networks model

```
class MyModel(Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.flatten = Flatten()
        self.d1 = Dense(128, activation='relu')
        self.d2 = Dense(128, activation='relu')
        self.d3 = Dense(10, activation='softmax')

    def call(self, x):
        x = self.flatten(x)
        x = self.d1(x)
        x = self.d2(x)
        return self.d3(x)

model = MyModel()
```

Image Classification using Deep NN with Keras

- Subclassing API(for expert)
  - Select the loss function and the optimizer

```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy()
optimizer = tf.keras.optimizers.Adam(0.001)
```

  - Select metrics to measure the loss and the accuracy of the model

```
train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='train_accuracy')

test_loss = tf.keras.metrics.Mean(name='test_loss')
test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='test_accuracy')
```

Image Classification using Deep NN with Keras

- Subclassing API(for expert)
  - Define train step

```
@tf.function
def train_step(images, labels):
    with tf.GradientTape() as tape:
        predictions = model(images)
        loss = loss_object(labels, predictions)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    train_loss(loss)
    train_accuracy(labels, predictions)
```

  - Define test step

```
@tf.function
def test_step(images, labels):
    predictions = model(images)
    t_loss = loss_object(labels, predictions)
    test_loss(t_loss)
    test_accuracy(labels, predictions)
```

Image Classification using Deep NN with Keras

- Subclassing API(for expert)
  - Train the model

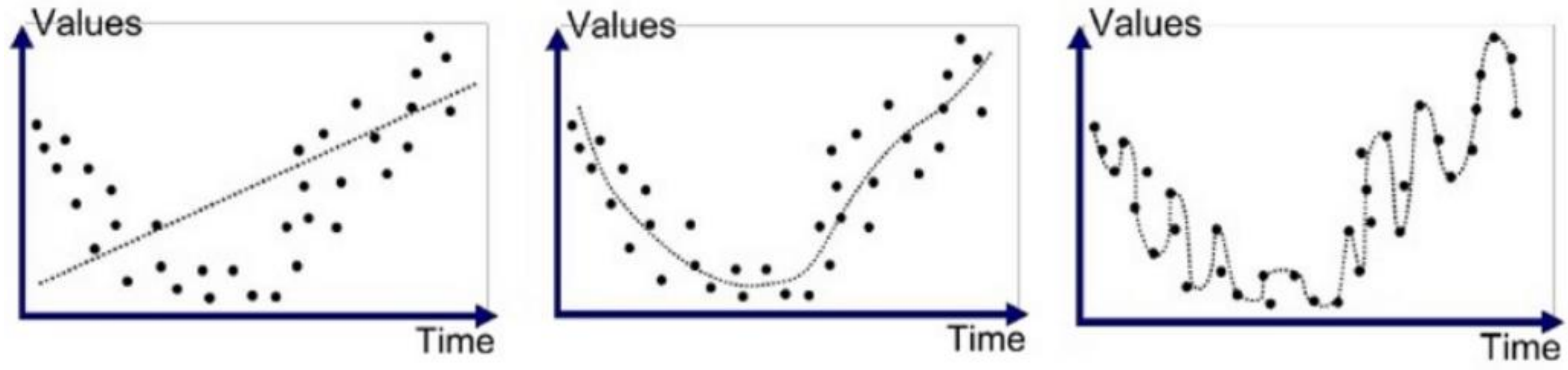
```
EPOCHS = 20

for epoch in range(EPOCHS):
    for images, labels in train_ds:
        train_step(images, labels)

    for test_images, test_labels in test_ds:
        test_step(test_images, test_labels)

    template = 'Epoch: {}, Train Loss: {}, Train Accuracy: {},
                Test Loss: {}, Test Accuracy: {}'
    print (template.format(epoch+1,
                           train_loss.result(),
                           train_accuracy.result()*100,
                           test_loss.result(),
                           test_accuracy.result()*100))
```

```
Epoch: 1, Loss: 0.5604610443115234, Train Accuracy: 80.52666473388672, Test Accuracy: 82.95000457763672
```

⋮

```
Epoch: 20, Loss: 0.2661779224872589, Train Accuracy: 90.2100830078125, Test Accuracy: 87.80500030517578
```

Overfitting and Underfitting



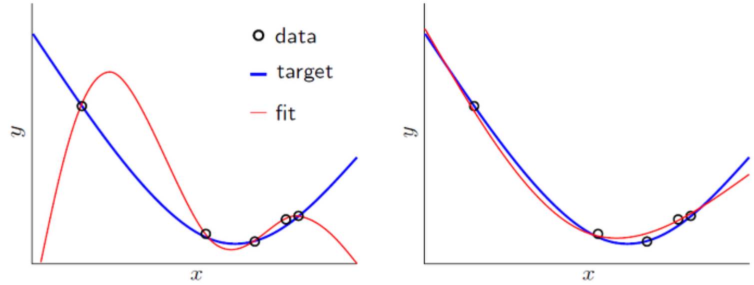
Underfitted | Good fit/Robust | Overfitted

Overfitting and Underfitting

- Avoid the Overfitting, L2 Regularization : A Review
  - The process of adding information in order to prevent overfitting
  - Add regularization term to a cost function

- Example

$$J(w) = \sum_{i=1}^{n}\left[-y^{(i)}\log\left(\phi\left(z^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-\phi\left(z^{(i)}\right)\right)\right] + \frac{\lambda}{2}\|w\|^2$$



(a) without regularization (b) with regularization

Overfitting and Underfitting

- Import TensorFlow

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

- Load the IMDb Dataset

```
NUM_WORDS = 3000
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.imdb.\
load_data(num_words=NUM_WORDS)

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(25000,)
(25000,)
(25000,)
(25000,)
```

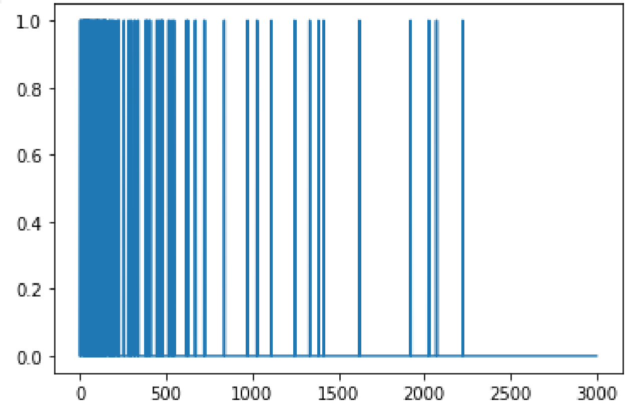
Error : Object arrays cannot be loaded when allow_pickle=False
conda install numpy=1.16.1

Overfitting and Underfitting

- Multi-hot Encoding

```python
def multi_hot_encoding(seq, dims):
    results = np.zeros((len(seq), dims))

    for i, word_indices in enumerate(seq):
        results[i, word_indices] = 1.0

    return results

X_train = multi_hot_encoding(X_train, dims=NUM_WORDS)
X_test = multi_hot_encoding(X_test, dims=NUM_WORDS)

%matplotlib inline
print(X_train[0])
plt.plot(X_train[0])
```

```
[0. 1. 1. ... 0. 0. 0.]
[<matplotlib.lines.Line2D at 0x7fbd1683ba20>]
```



Overfitting and Underfitting

- A baseline Model

```
layers = tf.keras.layers
baseline_model = tf.keras.Sequential([
    layers.Dense(16, activation='relu', input_shape=(NUM_WORDS,)),
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
baseline_model.compile(optimizer='adam',
                      loss='binary_crossentropy',
                      metrics=['accuracy', 'binary_crossentropy'])
baseline_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 16)                48016
_________________________________________________________________
dense_1 (Dense)              (None, 16)                272
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 17
=================================================================
Total params: 48,305
Trainable params: 48,305
Non-trainable params: 0
```

Overfitting and Underfitting

A baseline Model

```
baseline_history = baseline_model.fit(X_train,
                                      y_train,
                                      epochs=20,
                                      batch_size=512,
                                      validation_data=(X_test, y_test),
                                      verbose=2)
```

```
Epoch 1/20
49/49 - 1s - loss: 0.5497 - accuracy: 0.7582 - binary_crossentropy: 0.5497 - val_loss: 0.3968
Epoch 2/20
49/49 - 1s - loss: 0.3148 - accuracy: 0.8804 - binary_crossentropy: 0.3148 - val_loss: 0.2986
Epoch 3/20
49/49 - 1s - loss: 0.2509 - accuracy: 0.9034 - binary_crossentropy: 0.2509 - val_loss: 0.2939
Epoch 4/20
49/49 - 1s - loss: 0.2295 - accuracy: 0.9129 - binary_crossentropy: 0.2295 - val_loss: 0.2960
Epoch 5/20
49/49 - 1s - loss: 0.2172 - accuracy: 0.9167 - binary_crossentropy: 0.2172 - val_loss: 0.3053
```

⋮

```
Epoch 19/20
49/49 - 0s - loss: 0.0626 - accuracy: 0.9850 - val_loss: 0.5281 - val_accuracy: 0.8528
Epoch 20/20
49/49 - 0s - loss: 0.0547 - accuracy: 0.9872 - val_loss: 0.5519 - val_accuracy: 0.8504
```

Overfitting and Underfitting

- A smaller Model

```
smaller_model = tf.keras.Sequential([
     layers.Dense(4, activation='relu', input_shape=(NUM_WORDS,)),
     layers.Dense(4, activation='relu'),
     layers.Dense(1, activation='sigmoid')
])
smaller_model.compile(optimizer='adam',
                      loss='binary_crossentropy',
                      metrics=['accuracy', 'binary_crossentropy'])
smaller_model.summary()
```

```
Model: "sequential_4"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 4)                 12004
_________________________________________________________________
dense_13 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_14 (Dense)             (None, 1)                 5
=================================================================
Total params: 12,029
Trainable params: 12,029
Non-trainable params: 0
```

Overfitting and Underfitting

- A smaller Model

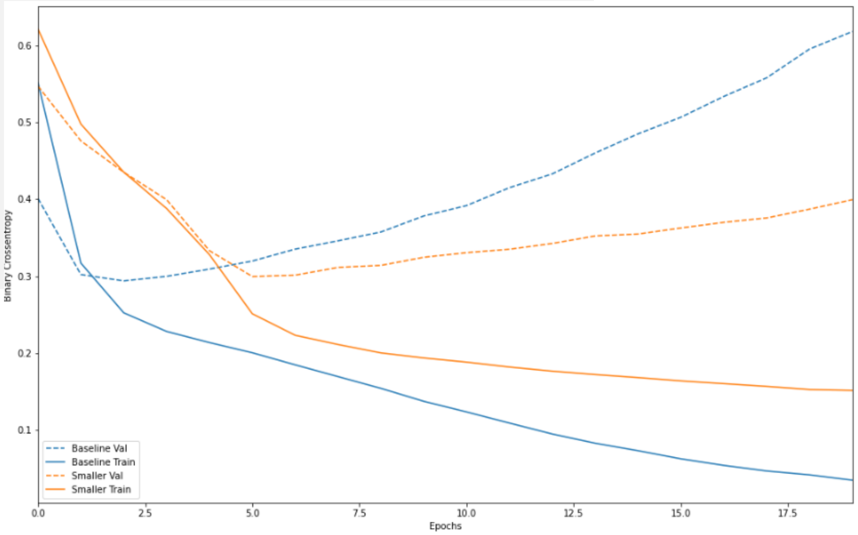
```
smaller_history = smaller_model.fit(X_train,
                                    y_train,
                                    epochs=20,
                                    batch_size=512,
                                    validation_data=(X_test, y_test),
                                    verbose=2)
```

```
Epoch 1/20
49/49 - 1s - loss: 0.6604 - accuracy: 0.5974 - binary_crossentropy: 0.6604 - val_loss: 0.6103
Epoch 2/20
49/49 - 1s - loss: 0.5683 - accuracy: 0.7686 - binary_crossentropy: 0.5683 - val_loss: 0.5415
Epoch 3/20
49/49 - 1s - loss: 0.5097 - accuracy: 0.8290 - binary_crossentropy: 0.5097 - val_loss: 0.4993
Epoch 4/20
49/49 - 1s - loss: 0.4702 - accuracy: 0.8628 - binary_crossentropy: 0.4702 - val_loss: 0.4721
Epoch 5/20
49/49 - 1s - loss: 0.4416 - accuracy: 0.8817 - binary_crossentropy: 0.4416 - val_loss: 0.4535

                                  ⋮

Epoch 19/20
49/49 - 1s - loss: 0.1554 - accuracy: 0.9425 - binary_crossentropy: 0.1554 - val_loss: 0.3578
Epoch 20/20
49/49 - 1s - loss: 0.1499 - accuracy: 0.9446 - binary_crossentropy: 0.1499 - val_loss: 0.3629
```

Overfitting and Underfitting

- Comparison models

```python
def plot_history(histories, key='binary_crossentropy'):
    plt.figure(figsize=(16,10))

    for name, history in histories:
        val = plt.plot(history.epoch, history.history['val_'+key],
                       '--', label=name.title()+' Val')
        plt.plot(history.epoch, history.history[key], color=val[0].get_color(),
                 label=name.title()+' Train')

    plt.xlabel('Epochs')
    plt.ylabel(key.replace('_',' ').title())
    plt.legend()
    plt.xlim([0,max(history.epoch)])

plot_history([('baseline', baseline_history),
              ('smaller', smaller_history)])
```



Overfitting and Underfitting

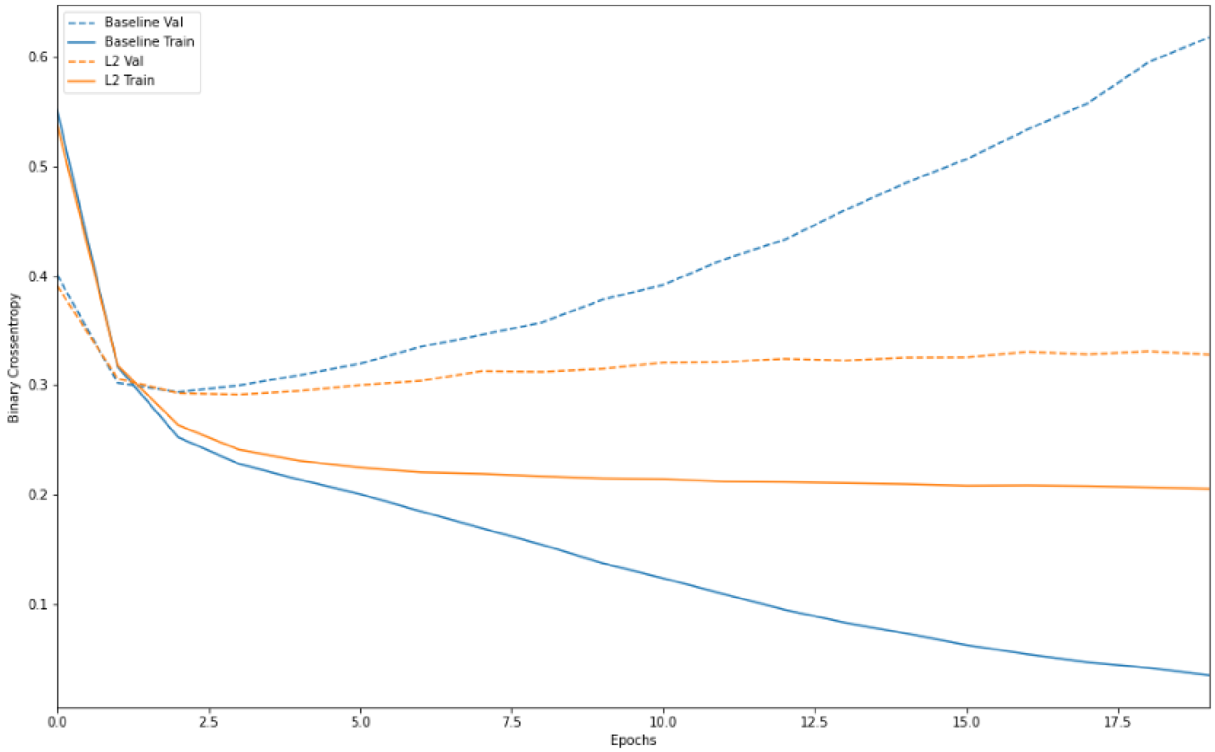
- Avoiding Overfitting : L2 Regularization

```
l2_model = tf.keras.models.Sequential([
    layers.Dense(16, kernel_regularizer=tf.keras.regularizers.l2(0.001),
                 activation='relu', input_shape=(NUM_WORDS,)),
    layers.Dense(16, kernel_regularizer=tf.keras.regularizers.l2(0.001),
                 activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

l2_model.compile(optimizer='adam',
                 loss='binary_crossentropy',
                 metrics=['accuracy', 'binary_crossentropy'])

l2_model_history = l2_model.fit(X_train,
                                y_train,
                                epochs=20,
                                batch_size=512,
                                validation_data=(X_test, y_test),
                                verbose=2)
```

Overfitting and Underfitting

- Avoiding Overfitting : L2 Regularization

```
plot_history([('baseline', baseline_history),
              ('l2', l2_model_history)])
```

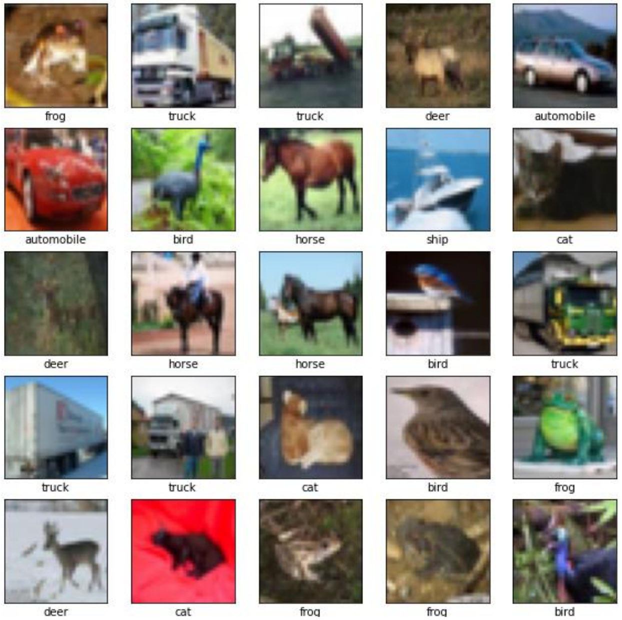


Submit

- To make sure if you have completed this practice,
  Submit your practice file(Week12_givencode.ipynb) to e-class.
- **Deadline : tomorrow 11:59pm**
- Modify your ipynb file name as “Week12_StudentNum_Name.ipynb”
  Ex) **Week11_2020123456_홍길동.ipynb**
- You can upload this file without taking the quiz, but homework will be provided like a quiz every three weeks, so it is recommended to take the quiz as well.

Quiz : Build Neural Networks for image classification

- Dataset : CIFAR-10
  - 10 Classes :
    'airplane', 'automobile', 'bird', 'cat', 'deer','dog', 'frog', 'horse', 'ship', 'truck'
  - Train / Test : 50,000 / 10,000
  - Width, Height, Channel : 32, 32, 3



Quiz : Build Neural Networks for image classification

- To do
  - Build the Neural Networks with Scikit-learn
  - Build the Neural Networks with Sequential API(`tf.keras.models.Sequential`)
  - Build the Neural Networks with Subclasss API(`tf.keras.Model`)