Machine Learning Engineer Course
Day 21

- Deep Learning Framework 2 – Keras -

DIVE INTO CODE

Thursday September 9, 2021

DIOP Mouhamed

Agenda

1. Check-in
2. How to proceed
3. Quick Review
4. Keras
5. Assignment
6. Keras – Sample Code
7. Check-out

Check-in

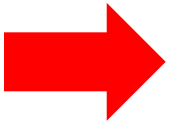
3 minutes Please post the following point to Zoom chat.

Q. What did you learn in the previous week?
(Anything is fine.)

How to proceed - Objective

What is the purpose?

1. Understand and be able to use Keras
2. Solve the previous problems using Keras



Let’s learn the basics of Keras here

Quick Review (TensorFlow)

Tensorflow is an open software library for numerical computation and large-scale machine learning developed by Google Brain.

It is a data flow programming language that describes a series of connections between data (directed graph) and expresses the movement of data (data flow).

The previous version of TensorFlow used Define-and-Run.
Define-by-Run is adopted in ver. 2.0.

What is Keras?

A framework for DeepLearning written in Python.

Keras (κέρας) is the Greek word for horn.
This is a play on words to enjoy the similar sound of κέρας (horn) / κραίνω (execution) and ἐλέφας (ivory) / ἐλεφαίρομαι (deception).
https://keras.io/

Author of Keras

François Chollet, the author of Keras, is an artificial intelligence researcher.

Deep Learning research and Tensorflow-Keras development at Google since August 2015.

For his own research and experimentation, he committed the first version of Keras to GitHub on March 27, 2015.
Keras was preferred by many researchers and developers because of its efficient and easy-to-use API designed compared to other frameworks of the time (Torch-Theano-Caffe).

François Chollet's personal page
https://fchollet.com/

François Chollet's Twitter
https://twitter.com/fchollet?ref_src=twsrc%5Egoogle%7Ctwcamp%5Eserp%7Ctwgr%5Eauthor

The Keras backend

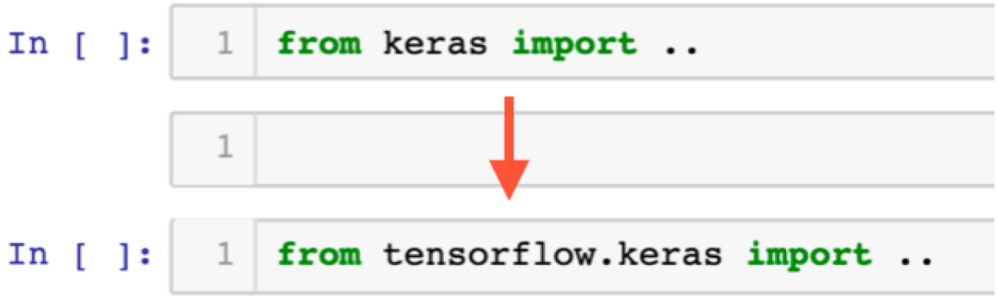
Keras required a backend (i.e., a computation engine; it builds the graph-topology, optimizes it, and performs numerical computations) to train the network.

Since Keras can be thought of as an abstract set of languages that access backends, we could switch between several backends as we wished.
Initially, the default backend for Keras was Theano, but in November 2015, Google released Tensorflow, and Keras also started supporting Tensorflow as a backend.

Introduction of tf.Keras

In TensorFlow v1.10.0, Keras was integrated within the Tensorflow package and tf.keras was introduced as a submodule. Furthermore, in June 2019, when TensorFlow v2.0 was announced, tf.keras became an official high-level API.

The transition from independent Keras to tf.keras is simple and requires only a rewrite of the import statement (or rewriting keras as tf.keras in the code).



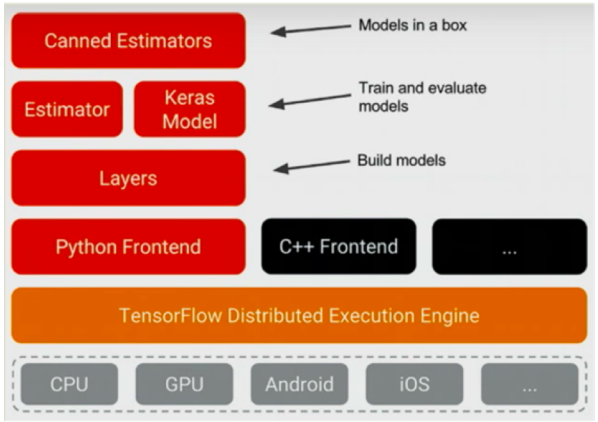
From tf.layers to tf.keras.layers

Prior to Tensorflow v1.13, the tf.layers API was used in **model building** as standard.

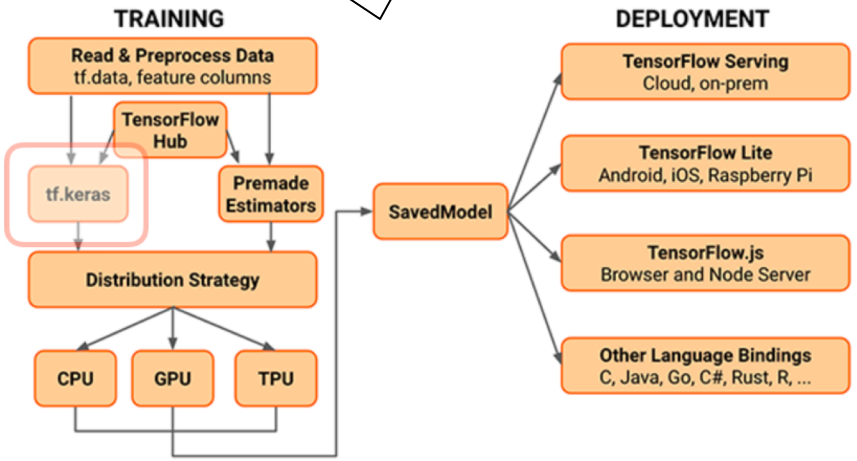
Since v1.13, tf.layers has been deprecated (a warning appears) and replaced by tf.keras.layers in Tensorflow v2.0 and later.

From tf.layers to tf.keras.layers

Research, experimentation, model preparation/quantization, and deployment into production became faster and more efficient.



Frameworks Design Transition



Tensorflow v1.13 and earlier

Tensorflow v2.x

What is Keras?

The following three notations are available for building models with tf.keras.
(You can choose any notation you like, depending on your needs.

Sequential Model API (dead simple!)

The model is built by instantiating Sequential( ) and stacking layer instances linearly by the add( ) method. There are restrictions on the structure of the model. (single-input, single-output)

functional API (like Lego!)

An instance of a layer can be invoked functionally, returning tensor (a multidimensional array)[1] as a return value. The output from a layer becomes the input of the next layer, and by connecting the input and output layers of a Model( ) instance as arguments, a model with a complex structure can be constructed. (Multi-input, multi-output)
[1] What is a tensor: https://www.tensorflow.org/tutorials/eager/basics?hl=ja

Model subclassing (fully-customizable!)

It is suitable for developers who want to build models flexibly.

Keras Sequential Model API

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

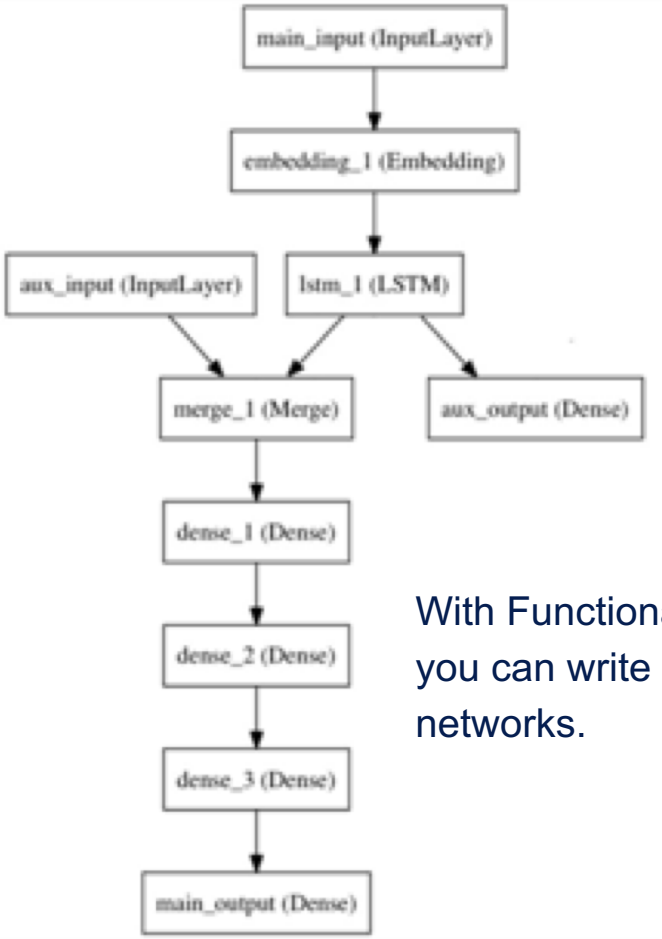
If we write the import statement like this on p7
The previous tf.layers code can be reused.

```
model = keras.Sequential()
model.add(layers.Dense(20, activation='relu',
input_shape=(10,)))        #Number of output units: 20
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(20,activation='softmax'))

optimizer = keras.optimizers.RMSprop()
loss =
keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metrics=['accuracy']

model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
model.fit(x, y, epochs=10, batch_size=32)
```

Keras Guide：https://keras.io/ja/models/sequential/
tf.keras guide：https://www.tensorflow.org/api_docs/python/tf/keras/Sequential

Keras functional API

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(10,))

x = layers.Dense(20, activation='relu')(inputs)
x = layers.Dense(20, activation='relu')(x)

outputs = layers.Dense(10,activation='softmax')(x)

optimizer = keras.optimizers.RMSprop()
loss = keras.losses.SparseCategoricalCrossentropy(from_logits
metrics=['accuracy']

model = keras.Model(inputs, outputs)
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
model.fit(x, y, epochs=10, batch_size=32)
```



With Functional APIs, you can write complex networks.

Keras Guide： https://keras.io/ja/getting-started/functional-api-guide/
tf.keras guide：https://www.tensorflow.org/guide/keras/functional

Model subclassing

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

class MyModel(keras.Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.dense1 = layers.Dense(20,
                  activation='relu')
    self.dense2 = layers.Dense(20,
                  activation='relu')
    self.dense1 = layers.Dense(10,
                  activation='softmax')
  def call(self, inputs):
    x = self.dense1(x)
    x = self.dence2(x)
    return self.dense3(x)

optimizer = keras.optimizers.RMSprop()
loss =
keras.losses.SparseCategoricalCrossentropy(from
_logits=True)
metrics=['accuracy']

model = MyModel()
model.compile(optimizer=optimizer, loss=loss,
metrics=metrics)
model.fit(x, y, epochs=10, batch_size=32)
```

Keras Guide : https://keras.io/models/about-keras-models/#model-subclassing
tf.keras guide : https://www.tensorflow.org/api_docs/python/tf/keras/Model

Model subclassing

Example: Class LeNet

Published by Yann LeCun in 1998

The original CNN architecture

http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

```
class LeNet(tf.keras.Model):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv2d_1 = tf.keras.layers.Conv2D(filters=6,
                                               kernel_size=(3, 3),
                                               activation='relu',
                                               input_shape=(32,32,1))

        self.average_pool = tf.keras.layers.AveragePooling2D()

        self.conv2d_2 = tf.keras.layers.Conv2D(filters=16,
                                               kernel_size=(3, 3),
                                               activation='relu')

        self.flatten = tf.keras.layers.Flatten()
        self.fc_1 = tf.keras.layers.Dense(120, activation='relu')
        self.fc_2 = tf.keras.layers.Dense(84, activation='relu')
        self.out = tf.keras.layers.Dense(10,activation='softmax')

    def call(self, input):
        x = self.conv2d_1(input)
        x = self.average_pool(x)
        x = self.conv2d_2(x)
        x = self.average_pool(x)
        x = self.flatten(x)
        x = self.fc_2(self.fc_1(x))
        return self.out(x)

lenet = LeNet()
```

What happens next

Further check the basic notation of Sequential model API and functional API

Keras Sequential model API

Baseline for model building

①Create an instance of Sequential() and stack the layers
②Compile an instance of Sequential().
Specify the optimizer, loss, and metrics here.

③Fit, evaluate, and predict the instance.
④Save the instance
Overwrite each epoch and save the last weight.

sequential.py de Keras
https://github.com/keras-team/keras/blob/master/keras/engine/sequential.py

```python
import keras
from keras.models import Sequential
from keras.layers import Dense

#Create Sequential model with Dense layers, using the add method
model = Sequential()

#Dense implements the operation:
#        output = activation(dot(input, kernel) + bias)
#Units are the dimensionality of the output space for the layer,
#     which equals the number of hidden units
#Activation and loss functions may be specified by strings or classes
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))

#The compile method configures the model's learning process
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

#The fit method does the training in batches
# x_train and y_train are Numpy arrays --just like in the Scikit-Learn
model.fit(x_train, y_train, epochs=5, batch_size=32)

#The evaluate method calculates the losses and metrics
#     for the trained model
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)

#The predict method applies the trained model to inputs
#     to generate outputs
classes = model.predict(x_test, batch_size=128)
```

Keras Sequential model API

How to write different model builds

①How to create a Sequential instance and pass a list of layer instances

②How to create a Sequential instance and stack layer instances with the add method

Running **model.layers** will return a list of layers as follows

```
[<keras.layers.core.Dense at 0x12dd97588>,
 <keras.layers.core.Activation at 0x12dd972e8>,
 <keras.layers.core.Dense at 0x12dd97278>,
 <keras.layers.core.Activation at 0x12dd97e48>]
```

```
model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

①

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

②

Keras Sequential model API

How to write different input shapes

①Using the input_shape argument

②Using the input_dim argument

③Use the batch_input_shape argument

④Do not explicitly write the input shape.

```
# Optionally, the first layer can receive an `input_shape` argument:
model = Sequential()
model.add(Dense(32, input_shape=(500,)))
# Afterwards, we do automatic shape inference:
model.add(Dense(32))

# This is identical to the following:
model = Sequential()
model.add(Dense(32, input_dim=500))

# And to the following:
model = Sequential()
model.add(Dense(32, batch_input_shape=(None, 500)))

# Note that you can also omit the `input_shape` argument:
# In that case the model gets built the first time you call `fit` (or other
# training and evaluation methods).
model = Sequential()
model.add(Dense(32))
model.add(Dense(32))
model.compile(optimizer=optimizer, loss=loss)
# This builds the model for the first time:
model.fit(x, y, batch_size=32, epochs=10)
```

① ② ③ ④

Running **model.weights** will return a list of weights as follows

```
[<tf.Variable 'dense_1/kernel:0' shape=(500, 32) dtype=float32_ref>,
 <tf.Variable 'dense_1/bias:0' shape=(32,) dtype=float32_ref>,
 <tf.Variable 'dense_2/kernel:0' shape=(32, 32) dtype=float32_ref>,
 <tf.Variable 'dense_2/bias:0' shape=(32,) dtype=float32_ref>]
```

Do not hold weights until fit

④If you run **model.weights** before fit, you will get [] (an empty list) will be returned..

Keras functional API

①Output tf.Tensor from Input() and layer instance
②Pass the output of ① to the instance of Model()
③Compile an instance of Model().
Specify the optimizer, loss, and metrics here.
③Fit, evaluate, and predict the instance.
④Save the instance
Overwrite each epoch and save the last weight.

functional.ipynb
https://github.com/tensorflow/docs/blob/master/site/en/guide/keras/functional.ipynb

input_layer.py
https://github.com/keras-team/keras/blob/master/keras/engine/input_layer.py#L114

core.py
https://github.com/keras-team/keras/blob/master/keras/layers/core.py

```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

※ Evaluation and prediction are omitted.

Keras functional API

What are the parentheses that touch the layer instance?

```
# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)
```

A layer is a class whose instance can be called like a function (the call process is triggered when called like a function).

Keras Design Philosophy

Let's consider two different codes that make a hamburger for the user.

① Checkbox-driven design (describing the procedures on the restaurant side)

```
cooked_burger = cook_burger(burger,
    grill_model='GR12',  # Burger cooker
    time_on_grill=120,  # Cooking time 120 seconds
    grill_temperature=150  # Heat at 150°C
)
```

② User-driven design (focus on the user's experience)

```
cooked_burger = cook_burger(burger,
    level='medium rare'
)                    # The user variable level represents the user's order, and
                     # Mapping to time_on_grill and grill_temperature
```

François Chollet

Chollet designed Keras so that it can be written based on user-centered thinking, as shown in ②.

Keras Design Philosophy

Examples of variables in each design

Checkbox-driven variables.
graph、session、scope、buffer、param_group
→ Solution-driven

User-driven variables:.
layer、model、optimizer、weights、initializer
→Problem-oriented and domain-oriented

Keras Design Philosophy

In addition, Chollet references scikit-learn in the design of the Keras API.

In scikit-learn, no matter how complex the model, the same basic settings (predicates and code correspondences that represent user expectations to "fit" and "predict") apply.

The reason why model.fit and model.predict appear in Keras is because they inherited this elegant notation from scikit-learn.

Example: linear regression model in scikit-learn

```
from sklearn.linear_model import LinearRegression
model = LinearRegression（fit_intercept = False）
model.fit（X、y）
y_pred = model.predict（X_pred）
```

Bonus

Keras official website

https://keras.io/

Sample code

How to solve problems

“Deep Learning Framework 2”

[Problem 1] Share and run the official tutorial model
[Problem 2] (Advance Problem) Implement various methods
[Problem 3] Learning Iris (binary classification) with Keras
[Problem 4] Learning Iris (multi-class classification) with Keras
[Problem 5] Learning House-Prices with Keras
[Problem 6] Training MNIST with Keras
[Problem 7] (Advance Problem) Rewritting to PyTorch
[Problem 8] (Advance Problem) Comparison of Frameworks

Sprint 14 – Keras

Explanation about this Sprint is given but please try it on your own first.

Sprint 14 – Keras

Please work on your own after class and submit your assignments on DIVER.

Sprint 14 – Keras

A Sample Code of this Sprint is given but please try it on your own.

Sprint 14 – Keras

Please work on your own after class and submit your assignments on DIVER.

ToDo by next class

✅ Next class will be Zoom : Thursday September 16, 2021 19:30 ~ 20:30

🚨ToDo: Paper Reading
https://diver.diveintocode.jp/curriculums/1904

Check-out

3 minutes Please post the following point to Zoom chat.

Q. Current feelings and reflections
(joy, anger, sorrow, anticipation, nervousness, etc.)

Thank You For Your Attention