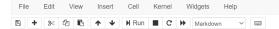


Not Trusted Python 3 O



Ungraded Lab: Keras custom callbacks

A custom callback is a powerful tool to customize the behavior of a Keras model during training, evaluation, or inference. Towards the end of this guide, there will be demos of creating a couple of simple callback applications to get you started on your custom callback.

Imports

```
In []: Wimport tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np

import datetime
import io

from PIL import Image
from IPython.display import Image as IPyImage
import imageio

print("Version: ", tf._version_)
tf.get_logger().setLevel('INFO')

In []: Wimport ion and a callbacks to
def get_model():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Dense(1, activation = 'linear', input_dim = 784))
    model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=0.1), loss='mean_squared_error', metrics=['mae'])
    return model
```

Then, load the MNIST data for training and testing from Keras datasets API:

```
In []: W # Load example MNIST data and pre-process it
  (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
  x_train = x_train.reshape(60000, 784).astype('float32') / 255
  x_test = x_test.reshape(10000, 784).astype('float32') / 255
```

Now, define a simple custom callback to track the start and end of every batch of data. During those calls, it prints the index of the current batch.

Providing a callback to model methods such as tf.keras.Model.fit() ensures the methods are called at those stages:

An overview of callback methods

Common methods for training/testing/predicting

For training, testing, and predicting, following methods are provided to be overridden.

on_(train|test|predict)_begin(self, logs=None)

Called at the beginning of fit / evaluate / predict .

on_(train|test|predict)_end(self, logs=None)

Called at the end of fit / evaluate / predict.

on_(train|test|predict)_batch_begin(self, batch, logs=None)

Called right before processing a batch during training/testing/predicting. Within this method, logs is a dict with batch and size available keys, representing the current batch number and the size of the batch.

on_(train|test|predict)_batch_end(self, batch, logs=None)

 $Called \ at \ the \ end \ of \ training/testing/predicting \ a \ batch. \ Within \ this \ method, \ \ logs \ \ is \ a \ dict \ containing \ the \ stateful \ metrics \ result.$

Training specific methods

In addition, for training, following are provided.

on_epoch_begin(self, epoch, logs=None)

Called at the beginning of an epoch during training.

on_epoch_end(self, epoch, logs=None)

Called at the end of an epoch during training.

Usage of logs dict

The logs dict contains the loss value, and all the metrics at the end of a batch or epoch. Example includes the loss and mean absolute error.

```
print("Epoch: {}, Val/Train loss ratio: {:.2f}".format(epoch, logs["val_loss"] / logs["loss"]))
                model = get_model()
                = model.fit(x_train, y_train,
validation_data=(x_test, y_test),
batch_size=64,
                             epochs=3,
                             verbose=0
                             callbacks=[callback])
self.threshold = threshold
                     def on_epoch_end(self, epoch, logs=None):
    ratio = logs["val_loss"] / logs["loss"]
    print("Epoch: {}, Val/Train loss ratio: {:.2f}".format(epoch, ratio))
                           if ratio > self.threshold:
                                print("Stopping training...")
self.model.stop_training = True
                batch_size=64,
epochs=3,
verbose=0,
                                  callbacks=[DetectOverfittingCallback()])
           Similarly, one can provide callbacks in evaluate() calls.
           Custom callback to Visualize predictions
In [ ]: ▶ # Load example MNIST data and pre-process it
                (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = x_train.reshape(60000, 784).astype('float32') / 255
x_test = x_test.reshape(10000, 784).astype('float32') / 255
In [ ]: # Visualization utilities
                plt.rc('font', size=20)
plt.rc('figure', figsize=(15, 3))
                def display_digits(inputs, outputs, ground_truth, epoch, n=10):
    plt.clf()
                     plt.yticks([])
                     plt.grid(None)
inputs = np.reshape(inputs, [n, 28, 28])
inputs = np.swapaxes(inputs, 0, 1)
inputs = np.reshape(inputs, [28, 28*n])
plt.imshow(inputs)
                     plt.imsnow(inputs)
plt.xticks([28*x+14 for x in range(n)], outputs)
for i,t in enumerate(plt.gca().xaxis.get_ticklabels()):
                          if outputs[i] == ground_truth[i]:
    t.set_color('green')
                                t.set color('red')
                     plt.grid(None)
In [ ]:  M GIF_PATH = './animation.gif'
self.images = []
self.display_freq = display_freq
                           self.n_samples = n_samples
                     def on_epoch_end(self, epoch, logs=None):
                            Randomly sample data
                          indexes = np.random.choice(len(self.inputs), size=self.n_samples)
X_test, y_test = self.inputs[indexes], self.ground_truth[indexes]
predictions = np.argmax(self.model.predict(X_test), axis=1)
                          # Plot the digits
display_digits(X_test, predictions, y_test, epoch, n=self.display_freq)
                          # Save the figure
buf = io.BytesIO()
plt.savefig(buf, format='png')
                           buf.seek(0)
                           image = Image.open(buf)
self.images.append(np.array(image))
                           # Display the digits every 'display_freq' number of epochs
if epoch % self.display_freq == 0:
                               plt.show()
                     def on_train_end(self, logs=None):
    imageio.mimsave(GIF_PATH, self.images, fps=1)
model = tf.keras.Sequential()
                     model.add(tf.keras.layers.Dense(32, activation='linear', input_dim=784))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=le-4), loss='sparse_categorical_crossentropy', metrics=['accuracy'
                     return model
callbacks=[VisCallback(x_test, y_test)])
In [ ]: M SCALE = 60
                # FYI, the format is set to PNG here to bypass checks for acceptable embeddings
IPyImage(GIF_PATH, format='png', width=15 * SCALE, height=3 * SCALE)
```