Open in Colab

Ungraded Lab: Introduction to Keras callbacks

In Keras, `Callback` is a Python class meant to be subclassed to provide specific functionality, with a set of methods called at various stages of training (including batch/epoch start and ends), testing, and predicting. Callbacks are useful to get a view on internal states and statistics of the model during training. The methods of the callbacks can be called at different stages of training/evaluating/inference. Keras has available callbacks and we'll show how you can use it in the following sections. Please click the **Open in Colab** badge above to complete this exercise in Colab. This will allow you to take advantage of the free GPU runtime (for faster training) and compatibility with all the packages needed in this notebook.

Model methods that take callbacks

Users can supply a list of callbacks to the following `tf.keras.Model` methods:

- `fit()`, `fit_generator()` Trains the model for a fixed number of epochs (iterations over a dataset, or data yielded batch-by-batch by a Python generator).
- `evaluate()`, `evaluate_generator()` Evaluates the model for given data or data generator. Outputs the loss and metric values from the evaluation.
- `predict()`, `predict_generator()` Generates output predictions for the input data or data generator.

Imports

In [ ]:
```
from __future__ import absolute_import, division, print_function, unicode_literals

try:
    # %tensorflow_version only exists in Colab.
    %tensorflow_version 2.x
except Exception:
    pass

import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
import io
from PIL import Image

from tensorflow.keras.callbacks import TensorBoard, EarlyStopping, LearningRateScheduler, ModelCheckpoint, CSVLogger, ReduceL
%load_ext tensorboard

import os
import matplotlib.pylab as plt
import numpy as np
import math
import datetime
import pandas as pd

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

Examples of Keras callback applications

The following section will guide you through creating simple Callback applications.

In [ ]:
```
# Download and prepare the horses or humans dataset

splits, info = tfds.load('horses_or_humans', as_supervised=True, with_info=True, split=['train[:80%]', 'train[80%:]', 'test']

(train_examples, validation_examples, test_examples) = splits

num_examples = info.splits['train'].num_examples
num_classes = info.features['label'].num_classes
```

In [ ]:
```
SIZE = 150 #@param {type:"slider", min:64, max:300, step:1}
IMAGE_SIZE = (SIZE, SIZE)
```

In [ ]:
```
def format_image(image, label):
  image = tf.image.resize(image, IMAGE_SIZE) / 255.0
  return  image, label
```

In [ ]:
```
BATCH_SIZE = 32 #@param {type:"integer"}
```

In [ ]:
```
train_batches = train_examples.shuffle(num_examples // 4).map(format_image).batch(BATCH_SIZE).prefetch(1)
validation_batches = validation_examples.map(format_image).batch(BATCH_SIZE).prefetch(1)
test_batches = test_examples.map(format_image).batch(1)
```

In [ ]:
```
for image_batch, label_batch in train_batches.take(1):
  pass

image_batch.shape
```

In [ ]:
```
def build_model(dense_units, input_shape=IMAGE_SIZE + (3,)):
  model = tf.keras.models.Sequential([
      tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=input_shape),
      tf.keras.layers.MaxPooling2D(2, 2),
      tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
      tf.keras.layers.MaxPooling2D(2, 2),
      tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
      tf.keras.layers.MaxPooling2D(2, 2),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(dense_units, activation='relu'),
      tf.keras.layers.Dense(2, activation='softmax')
  ])
  return model
```

TensorBoard

Enable visualizations for TensorBoard.

In [ ]:
```
!rm -rf logs
```

In [ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir)

model.fit(train_batches,
          epochs=10,
          validation_data=validation_batches,
          callbacks=[tensorboard_callback])
```

In [ ]:
```
%tensorboard --logdir logs
```

Model Checkpoint

Callback to save the Keras model or model weights at some frequency.

In [ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

model.fit(train_batches,
          epochs=5,
          validation_data=validation_batches,
          verbose=2,
          callbacks=[ModelCheckpoint('weights.{epoch:02d}-{val_loss:.2f}.h5', verbose=1),
          ])
```

In [ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

model.fit(train_batches,
          epochs=1,
          validation_data=validation_batches,
          verbose=2,
          callbacks=[ModelCheckpoint('saved_model', verbose=1)
          ])
```

In [ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

model.fit(train_batches,
          epochs=2,
          validation_data=validation_batches,
          verbose=2,
          callbacks=[ModelCheckpoint('model.h5', verbose=1)
          ])
```

Early stopping

Stop training when a monitored metric has stopped improving.

In [ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

model.fit(train_batches,
          epochs=50,
          validation_data=validation_batches,
          verbose=2,
          callbacks=[EarlyStopping(
              patience=3,
              min_delta=0.05,
              baseline=0.8,
              mode='min',
              monitor='val_loss',
              restore_best_weights=True,
              verbose=1)
          ])
```

CSV Logger

Callback that streams epoch results to a CSV file.

In [ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

csv_file = 'training.csv'

model.fit(train_batches,
          epochs=5,
          validation_data=validation_batches,
          callbacks=[CSVLogger(csv_file)
          ])
```

In [ ]:
```
pd.read_csv(csv_file).head()
```

Learning Rate Scheduler

Updates the learning rate during training.

In [ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

def step_decay(epoch):
    initial_lr = 0.01
    drop = 0.5
    epochs_drop = 1
    lr = initial_lr * math.pow(drop, math.floor((1+epoch)/epochs_drop))
    return lr

model.fit(train_batches,
          epochs=5,
          validation_data=validation_batches,
          callbacks=[LearningRateScheduler(step_decay, verbose=1),
                     TensorBoard(log_dir='./log_dir')])
```

In [ ]:
```
%tensorboard --logdir log_dir
```

ReduceLROnPlateau

Reduce learning rate when a metric has stopped improving.

In [ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

model.fit(train_batches,
          epochs=50,
          validation_data=validation_batches,
          callbacks=[ReduceLROnPlateau(monitor='val_loss',
                                       factor=0.2, verbose=1,
                                       patience=1, min_lr=0.001),
                     TensorBoard(log_dir='./log_dir')])
```

In [ ]:
```
%tensorboard --logdir log_dir
```