Chapter 19 – Training and Deploying TensorFlow Models at Scale

This notebook contains all the sample code in chapter 19.

Open in Colab

Setup

First, let's import a few common modules, ensure MatplotLib plots figures inline and prepare a function to save the figures. We also check that Python 3.5 or later is installed (although Python 2.x may work, it is deprecated so we strongly recommend you use Python 3 instead), as well as Scikit-Learn ≥0.20 and TensorFlow ≥2.0.

In [1]:

```
# Python ≥3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Is this notebook running on Colab or Kaggle?
IS_COLAB = "google.colab" in sys.modules
IS_KAGGLE = "kaggle_secrets" in sys.modules

if IS_COLAB or IS_KAGGLE:
    !echo "deb http://storage.googleapis.com/tensorflow-serving-apt stable tensorflow-mod
el-server tensorflow-model-server-universal" > /etc/apt/sources.list.d/tensorflow-servin
g.list
    !curl https://storage.googleapis.com/tensorflow-serving-apt/tensorflow-serving.relea
se.pub.gpg | apt-key add -
    !apt update && apt-get install -y tensorflow-model-server
    %pip install -q -U tensorflow-serving-api

# Scikit-Learn ≥0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# TensorFlow ≥2.0 is required
import tensorflow as tf
from tensorflow import keras
assert tf.__version__ >= "2.0"

if not tf.config.list_physical_devices('GPU'):
    print("No GPU was detected. CNNs can be very slow without a GPU.")
    if IS_COLAB:
        print("Go to Runtime > Change runtime and select a GPU hardware accelerator.")
    if IS_KAGGLE:
        print("Go to Settings > Accelerator and select GPU.")

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)
tf.random.set_seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
```

```
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "deploy"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

Deploying TensorFlow models to TensorFlow Serving (TFS)

We will use the REST API or the gRPC API.

Save/Load a `SavedModel`

In [2]:

```
(X_train_full, y_train_full), (X_test, y_test) = keras.datasets.mnist.load_data()
X_train_full = X_train_full[..., np.newaxis].astype(np.float32) / 255.
X_test = X_test[..., np.newaxis].astype(np.float32) / 255.
X_valid, X_train = X_train_full[:5000], X_train_full[5000:]
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X_new = X_test[:3]
```

In [3]:

```
np.random.seed(42)
tf.random.set_seed(42)

model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28, 1]),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=keras.optimizers.SGD(learning_rate=1e-2),
              metrics=["accuracy"])
model.fit(X_train, y_train, epochs=10, validation_data=(X_valid, y_valid))
```

```
Epoch 1/10
1719/1719 [==============================] - 2s 1ms/step - loss: 1.1140 - accuracy: 0.706
6 - val_loss: 0.3715 - val_accuracy: 0.9024
Epoch 2/10
1719/1719 [==============================] - 1s 713us/step - loss: 0.3695 - accuracy: 0.8
981 - val_loss: 0.2990 - val_accuracy: 0.9144
Epoch 3/10
1719/1719 [==============================] - 1s 718us/step - loss: 0.3154 - accuracy: 0.9
100 - val_loss: 0.2651 - val_accuracy: 0.9272
Epoch 4/10
1719/1719 [==============================] - 1s 706us/step - loss: 0.2765 - accuracy: 0.9
223 - val_loss: 0.2436 - val_accuracy: 0.9334
Epoch 5/10
1719/1719 [==============================] - 1s 711us/step - loss: 0.2556 - accuracy: 0.9
276 - val_loss: 0.2257 - val_accuracy: 0.9364
Epoch 6/10
1719/1719 [==============================] - 1s 715us/step - loss: 0.2367 - accuracy: 0.9
321 - val_loss: 0.2121 - val_accuracy: 0.9396
Epoch 7/10
1719/1719 [==============================] - 1s 729us/step - loss: 0.2198 - accuracy: 0.9
390 - val_loss: 0.1970 - val_accuracy: 0.9454
Epoch 8/10
1719/1719 [==============================] - 1s 716us/step - loss: 0.2057 - accuracy: 0.9
425 - val_loss: 0.1880 - val_accuracy: 0.9476
Epoch 9/10
```

```
1719/1719 [==============================] - 1s 704us/step - loss: 0.1940 - accuracy: 0.9
459 - val_loss: 0.1777 - val_accuracy: 0.9524
Epoch 10/10
1719/1719 [==============================] - 1s 711us/step - loss: 0.1798 - accuracy: 0.9
482 - val_loss: 0.1684 - val_accuracy: 0.9546
```

Out[3]:

```
<tensorflow.python.keras.callbacks.History at 0x7fe3b8718590>
```

In [4]:

```
np.round(model.predict(X_new), 2)
```

Out[4]:

```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  ],
       [0.  , 0.  , 0.99, 0.01, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 0.97, 0.01, 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.  ]],
      dtype=float32)
```

In [5]:

```
model_version = "0001"
model_name = "my_mnist_model"
model_path = os.path.join(model_name, model_version)
model_path
```

Out[5]:

```
'my_mnist_model/0001'
```

In [6]:

```
import shutil

shutil.rmtree(model_name)
```

In [7]:

```
tf.saved_model.save(model, model_path)
```

```
INFO:tensorflow:Assets written to: my_mnist_model/0001/assets
```

In [8]:

```
for root, dirs, files in os.walk(model_name):
    indent = '    ' * root.count(os.sep)
    print('{}{}/'.format(indent, os.path.basename(root)))
    for filename in files:
        print('{}{}'.format(indent + '    ', filename))
```

```
my_mnist_model/
    0001/
        saved_model.pb
        variables/
            variables.data-00000-of-00001
            variables.index
        assets/
```

In [9]:

```
!saved_model_cli show --dir {model_path}
```

```
The given SavedModel contains the following tag-sets:
'serve'
```

In [10]:

```
!saved_model_cli show --dir {model_path} --tag_set serve
```

```
The given SavedModel MetaGraphDef contains SignatureDefs with the following keys:
SignatureDef key: "__saved_model_init_op"
```

```
SignatureDef key: "serving_default"
```

In [11]:

```
!saved_model_cli show --dir {model_path} --tag_set serve \
                      --signature_def serving_default
```

```
The given SavedModel SignatureDef contains the following input(s):
  inputs['flatten_input'] tensor_info:
      dtype: DT_FLOAT
      shape: (-1, 28, 28, 1)
      name: serving_default_flatten_input:0
The given SavedModel SignatureDef contains the following output(s):
  outputs['dense_1'] tensor_info:
      dtype: DT_FLOAT
      shape: (-1, 10)
      name: StatefulPartitionedCall:0
Method name is: tensorflow/serving/predict
```

In [12]:

```
!saved_model_cli show --dir {model_path} --all
```

```
MetaGraphDef with tag-set: 'serve' contains the following SignatureDefs:

signature_def['__saved_model_init_op']:
  The given SavedModel SignatureDef contains the following input(s):
  The given SavedModel SignatureDef contains the following output(s):
    outputs['__saved_model_init_op'] tensor_info:
        dtype: DT_INVALID
        shape: unknown_rank
        name: NoOp
  Method name is:

signature_def['serving_default']:
  The given SavedModel SignatureDef contains the following input(s):
    inputs['flatten_input'] tensor_info:
        dtype: DT_FLOAT
        shape: (-1, 28, 28, 1)
        name: serving_default_flatten_input:0
  The given SavedModel SignatureDef contains the following output(s):
    outputs['dense_1'] tensor_info:
        dtype: DT_FLOAT
        shape: (-1, 10)
        name: StatefulPartitionedCall:0
  Method name is: tensorflow/serving/predict

Defined Functions:
  Function Name: '__call__'
    Option #1
      Callable with:
        Argument #1
          flatten_input: TensorSpec(shape=(None, 28, 28, 1), dtype=tf.float32, name='flat
ten_input')
        Argument #2
          DType: bool
          Value: True
        Argument #3
<<45 more lines>>
          DType: bool
          Value: False
        Argument #3
          DType: NoneType
          Value: None
    Option #2
      Callable with:
        Argument #1
          inputs: TensorSpec(shape=(None, 28, 28, 1), dtype=tf.float32, name='inputs')
        Argument #2
          DType: bool
          Value: True
        Argument #3
```

```
          DType: NoneType
          Value: None
    Option #3
      Callable with:
        Argument #1
          flatten_input: TensorSpec(shape=(None, 28, 28, 1), dtype=tf.float32, name='flat
ten_input')
        Argument #2
          DType: bool
          Value: True
        Argument #3
          DType: NoneType
          Value: None
    Option #4
      Callable with:
        Argument #1
          flatten_input: TensorSpec(shape=(None, 28, 28, 1), dtype=tf.float32, name='flat
ten_input')
        Argument #2
          DType: bool
          Value: False
        Argument #3
          DType: NoneType
          Value: None
```

Let's write the new instances to a `npy` file so we can pass them easily to our model:

In [13]:

```
np.save("my_mnist_tests.npy", X_new)
```

In [14]:

```
input_name = model.input_names[0]
input_name
```

Out[14]:

```
'flatten_input'
```

And now let's use `saved_model_cli` to make predictions for the instances we just saved:

In [15]:

```
!saved_model_cli run --dir {model_path} --tag_set serve \
                     --signature_def serving_default    \
                     --inputs {input_name}=my_mnist_tests.npy
```

```
2021-02-18 22:15:30.294109: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating
XLA devices, tf_xla_enable_xla_devices not set
2021-02-18 22:15:30.294306: I tensorflow/core/platform/cpu_feature_guard.cc:142] This Ten
sorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the f
ollowing CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flag
s.
WARNING:tensorflow:From /Users/ageron/miniconda3/envs/tf2/lib/python3.7/site-packages/ten
sorflow/python/tools/saved_model_cli.py:445: load (from tensorflow.python.saved_model.loa
der_impl) is deprecated and will be removed in a future version.
Instructions for updating:
This function will only be available through the v1 compatibility library as tf.compat.v1
.saved_model.loader.load or tf.compat.v1.saved_model.load. There will be a new function f
or importing SavedModels in Tensorflow 2.0.
INFO:tensorflow:Restoring parameters from my_mnist_model/0001/variables/variables
2021-02-18 22:15:30.323498: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:19
6] None of the MLIR optimization passes are enabled (registered 0 passes)
Result for output key dense_1:
[[1.1347984e-04 1.5187356e-07 9.7032893e-04 2.7640699e-03 3.7826971e-06
  7.6876910e-05 3.9140293e-08 9.9559116e-01 5.3502394e-05 4.2665208e-04]
 [8.2443521e-04 3.5493889e-05 9.8826385e-01 7.0466995e-03 1.2957400e-07
  2.3389691e-04 2.5639210e-03 9.5886099e-10 1.0314899e-03 8.7952529e-08]
```

```
[4.4693781e-05 9.7028232e-01 9.0526715e-03 2.2641101e-03 4.8766597e-04
  2.8800720e-03 2.2714981e-03 8.3753867e-03 4.0439744e-03 2.9759688e-04]]
```

In [16]:

```
np.round([[1.1347984e-04, 1.5187356e-07, 9.7032893e-04, 2.7640699e-03, 3.7826971e-06,
           7.6876910e-05, 3.9140293e-08, 9.9559116e-01, 5.3502394e-05, 4.2665208e-04],
          [8.2443521e-04, 3.5493889e-05, 9.8826385e-01, 7.0466995e-03, 1.2957400e-07,
           2.3389691e-04, 2.5639210e-03, 9.5886099e-10, 1.0314899e-03, 8.7952529e-08],
          [4.4693781e-05, 9.7028232e-01, 9.0526715e-03, 2.2641101e-03, 4.8766597e-04,
           2.8800720e-03, 2.2714981e-03, 8.3753867e-03, 4.0439744e-03, 2.9759688e-04]],
2)
```

Out[16]:

```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  ],
       [0.  , 0.  , 0.99, 0.01, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 0.97, 0.01, 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.  ]])
```

TensorFlow Serving

Install Docker if you don't have it already. Then run:

```
docker pull tensorflow/serving

export ML_PATH=$HOME/ml # or wherever this project is
docker run -it --rm -p 8500:8500 -p 8501:8501 \
   -v "$ML_PATH/my_mnist_model:/models/my_mnist_model" \
   -e MODEL_NAME=my_mnist_model \
   tensorflow/serving
```

Once you are finished using it, press Ctrl-C to shut down the server.

Alternatively, if `tensorflow_model_server` is installed (e.g., if you are running this notebook in Colab), then the following 3 cells will start the server:

In [17]:

```
os.environ["MODEL_DIR"] = os.path.split(os.path.abspath(model_path))[0]
```

In [18]:

```
%%bash --bg
nohup tensorflow_model_server \
     --rest_api_port=8501 \
     --model_name=my_mnist_model \
     --model_base_path="${MODEL_DIR}" >server.log 2>&1
```

In [19]:

```
!tail server.log
```

```
2021-02-16 22:33:09.323538: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc
:93] Reading SavedModel debug info (if present) from: /models/my_mnist_model/0001
2021-02-16 22:33:09.323642: I external/org_tensorflow/tensorflow/core/platform/cpu_featur
e_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Libra
ry (oneDNN) to use the following CPU instructions in performance-critical operations:  AV
X2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flag
s.
2021-02-16 22:33:09.360572: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc
:206] Restoring SavedModel bundle.
2021-02-16 22:33:09.361764: I external/org_tensorflow/tensorflow/core/platform/profile_ut
ils/cpu_utils.cc:112] CPU Frequency: 2200000000 Hz
2021-02-16 22:33:09.387713: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc
:190] Running initialization op on SavedModel bundle at path: /models/my_mnist_model/0001
2021-02-16 22:33:09.392739: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc
:277] SavedModel load for tags { serve }; Status: success: OK. Took 71106 microseconds.
```

```
2021-02-16 22:33:09.393390: I tensorflow_serving/servables/tensorflow/saved_model_warmup_
util.cc:59] No warmup data file found at /models/my_mnist_model/0001/assets.extra/tf_serv
ing_warmup_requests
2021-02-16 22:33:09.393847: I tensorflow_serving/core/loader_harness.cc:87] Successfully
loaded servable version {name: my_mnist_model version: 1}
2021-02-16 22:33:09.398470: I tensorflow_serving/model_servers/server.cc:371] Running gRP
C ModelServer at 0.0.0.0:8500 ...
[warn] getaddrinfo: address family for nodename not supported
2021-02-16 22:33:09.405622: I tensorflow_serving/model_servers/server.cc:391] Exporting H
TTP/REST API at:localhost:8501 ...
[evhttp_server.cc : 238] NET_LOG: Entering the event loop ...
```

In [20]:

```python
import json

input_data_json = json.dumps({
    "signature_name": "serving_default",
    "instances": X_new.tolist(),
})
```

In [21]:

```python
repr(input_data_json)[:1500] + "..."
```

Out[21]:


```
'\'{"signature_name": "serving_default", "instances": [[[[0.0], [0.0], [0.0], [0.0], [0.0
], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [
0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]], [[0.0], [0.0
], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [
0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]
, [0.0]], [[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [
0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]
, [0.0], [0.0], [0.0], [0.0]], [[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [
0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]
, [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]], [[0.0], [0.0], [0.0], [0.0], [0.0], [
0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]
, [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]], [[0.0], [0.0], [
0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]
, [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0
.0]], [[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]
, [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0
.0], [0.0], [0.0], [0.0]], [[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.32941177487373..
.'
```


Now let's use TensorFlow Serving's REST API to make predictions:

In [22]:

```python
import requests

SERVER_URL = 'http://localhost:8501/v1/models/my_mnist_model:predict'
response = requests.post(SERVER_URL, data=input_data_json)
response.raise_for_status() # raise an exception in case of error
response = response.json()
```

In [23]:

```python
response.keys()
```

Out[23]:

```
dict_keys(['predictions'])
```

In [24]:

```python
y_proba = np.array(response["predictions"])
y_proba.round(2)
```

Out[24]:

```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  ],
       [0.  , 0.  , 0.99, 0.01, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 0.97, 0.01, 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.  ]])
```

Using the gRPC API

In [25]:

```
from tensorflow_serving.apis.predict_pb2 import PredictRequest

request = PredictRequest()
request.model_spec.name = model_name
request.model_spec.signature_name = "serving_default"
input_name = model.input_names[0]
request.inputs[input_name].CopyFrom(tf.make_tensor_proto(X_new))
```

In [26]:

```
import grpc
from tensorflow_serving.apis import prediction_service_pb2_grpc

channel = grpc.insecure_channel('localhost:8500')
predict_service = prediction_service_pb2_grpc.PredictionServiceStub(channel)
response = predict_service.Predict(request, timeout=10.0)
```

In [27]:

```
response
```

Out[27]:

```
outputs {
  key: "dense_1"
  value {
    dtype: DT_FLOAT
    tensor_shape {
      dim {
        size: 3
      }
      dim {
        size: 10
      }
    }
    float_val: 0.00011425172124290839
    float_val: 1.513665068841874e-07
    float_val: 0.0009818424005061388
    float_val: 0.0027773496694862843
    float_val: 3.758880893656169e-06
    float_val: 7.6266449468676e-05
    float_val: 3.9139514740327286e-08
    float_val: 0.995561957359314
    float_val: 5.344580131350085e-05
    float_val: 0.00043088122038170695
    float_val: 0.0008194865076802671
    float_val: 3.5498320357874036e-05
    float_val: 0.9882420897483826
    float_val: 0.00705744931474328
    float_val: 1.2937064752804872e-07
    float_val: 0.00023402832448482513
    float_val: 0.0025743397418409586
    float_val: 9.668431610876382e-10
    float_val: 0.0010369382798671722
    float_val: 8.833576004008137e-08
    float_val: 4.441547571332194e-05
    float_val: 0.970328688621521
    float_val: 0.009044423699378967
    float_val: 0.0022599005606025457
    float_val: 0.00048672096454538405
    float_val: 0.002873610006645322
    float_val: 0.002268279204145074
```

```
    float_val: 0.008354829624295235
    float_val: 0.004041312728077173
    float_val: 0.0002978229313157499
  }
}
model_spec {
  name: "my_mnist_model"
  version {
    value: 1
  }
  signature_name: "serving_default"
}
```

Convert the response to a tensor:

In [28]:

```
output_name = model.output_names[0]
outputs_proto = response.outputs[output_name]
y_proba = tf.make_ndarray(outputs_proto)
y_proba.round(2)
```

Out[28]:

```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  ],
       [0.  , 0.  , 0.99, 0.01, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 0.97, 0.01, 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.  ]],
      dtype=float32)
```

Or to a NumPy array if your client does not include the TensorFlow library:

In [29]:

```
output_name = model.output_names[0]
outputs_proto = response.outputs[output_name]
shape = [dim.size for dim in outputs_proto.tensor_shape.dim]
y_proba = np.array(outputs_proto.float_val).reshape(shape)
y_proba.round(2)
```

Out[29]:

```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  ],
       [0.  , 0.  , 0.99, 0.01, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 0.97, 0.01, 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.  ]])
```

Deploying a new model version

In [30]:

```
np.random.seed(42)
tf.random.set_seed(42)

model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28, 1]),
    keras.layers.Dense(50, activation="relu"),
    keras.layers.Dense(50, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=keras.optimizers.SGD(learning_rate=1e-2),
              metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_valid, y_valid))
```

```
Epoch 1/10
1719/1719 [==============================] - 1s 748us/step - loss: 1.1567 - accuracy: 0.6
691 - val_loss: 0.3418 - val_accuracy: 0.9042
Epoch 2/10
1719/1719 [==============================] - 1s 697us/step - loss: 0.3376 - accuracy: 0.9
032 - val_loss: 0.2674 - val_accuracy: 0.9242
Epoch 3/10
```

```
1719/1719 [==============================] - 1s 676us/step - loss: 0.2779 - accuracy: 0.9
187 - val_loss: 0.2227 - val_accuracy: 0.9368
Epoch 4/10
1719/1719 [==============================] - 1s 669us/step - loss: 0.2362 - accuracy: 0.9
318 - val_loss: 0.2032 - val_accuracy: 0.9432
Epoch 5/10
1719/1719 [==============================] - 1s 670us/step - loss: 0.2109 - accuracy: 0.9
389 - val_loss: 0.1833 - val_accuracy: 0.9482
Epoch 6/10
1719/1719 [==============================] - 1s 675us/step - loss: 0.1951 - accuracy: 0.9
430 - val_loss: 0.1740 - val_accuracy: 0.9498
Epoch 7/10
1719/1719 [==============================] - 1s 667us/step - loss: 0.1799 - accuracy: 0.9
474 - val_loss: 0.1605 - val_accuracy: 0.9540
Epoch 8/10
1719/1719 [==============================] - 1s 673us/step - loss: 0.1654 - accuracy: 0.9
519 - val_loss: 0.1543 - val_accuracy: 0.9558
Epoch 9/10
1719/1719 [==============================] - 1s 671us/step - loss: 0.1570 - accuracy: 0.9
554 - val_loss: 0.1460 - val_accuracy: 0.9572
Epoch 10/10
1719/1719 [==============================] - 1s 672us/step - loss: 0.1420 - accuracy: 0.9
583 - val_loss: 0.1359 - val_accuracy: 0.9616
```

In [31]:

```
model_version = "0002"
model_name = "my_mnist_model"
model_path = os.path.join(model_name, model_version)
model_path
```

Out[31]:

```
'my_mnist_model/0002'
```

In [32]:

```
tf.saved_model.save(model, model_path)
```

```
INFO:tensorflow:Assets written to: my_mnist_model/0002/assets
```

In [33]:

```
for root, dirs, files in os.walk(model_name):
    indent = '    ' * root.count(os.sep)
    print('{}{}/'.format(indent, os.path.basename(root)))
    for filename in files:
        print('{}{}'.format(indent + '    ', filename))
```

```
my_mnist_model/
    0001/
        saved_model.pb
        variables/
            variables.data-00000-of-00001
            variables.index
        assets/
    0002/
        saved_model.pb
        variables/
            variables.data-00000-of-00001
            variables.index
        assets/
```

Warning: You may need to wait a minute before the new model is loaded by TensorFlow Serving.

In [34]:

```
import requests

SERVER_URL = 'http://localhost:8501/v1/models/my_mnist_model:predict'
```

```
response = requests.post(SERVER_URL, data=input_data_json)
response.raise_for_status()
response = response.json()
```

In [35]:

```
response.keys()
```

Out[35]:

```
dict_keys(['predictions'])
```

In [36]:

```
y_proba = np.array(response["predictions"])
y_proba.round(2)
```

Out[36]:

```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  ],
       [0.  , 0.  , 0.99, 0.01, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 0.99, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]])
```

Deploy the model to Google Cloud AI Platform

Follow the instructions in the book to deploy the model to Google Cloud AI Platform, download the service account's private key and save it to the `my_service_account_private_key.json` in the project directory. Also, update the `project_id`:

In [37]:

```
project_id = "onyx-smoke-242003"
```

In [38]:

```
import googleapiclient.discovery

os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "my_service_account_private_key.json"
model_id = "my_mnist_model"
model_path = "projects/{}/models/{}".format(project_id, model_id)
model_path += "/versions/v0001/" # if you want to run a specific version
ml_resource = googleapiclient.discovery.build("ml", "v1").projects()
```

In [39]:

```
def predict(X):
    input_data_json = {"signature_name": "serving_default",
                       "instances": X.tolist()}
    request = ml_resource.predict(name=model_path, body=input_data_json)
    response = request.execute()
    if "error" in response:
        raise RuntimeError(response["error"])
    return np.array([pred[output_name] for pred in response["predictions"]])
```

In [40]:

```
Y_probas = predict(X_new)
np.round(Y_probas, 2)
```

Out[40]:

```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  ],
       [0.  , 0.  , 0.99, 0.01, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 0.99, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]])
```

Using GPUs

Note: `tf.test.is_gpu_available()` **is deprecated. Instead, please use** `tf.config.list_physical_devices('GPU')`**.**

In [41]:

```
#tf.test.is_gpu_available() # deprecated
tf.config.list_physical_devices('GPU')
```

Out[41]:

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU'),
 PhysicalDevice(name='/physical_device:GPU:1', device_type='GPU')]
```

In [42]:

```
tf.test.gpu_device_name()
```

Out[42]:

```
'/device:GPU:0'
```

In [43]:

```
tf.test.is_built_with_cuda()
```

Out[43]:

```
True
```

In [44]:

```
from tensorflow.python.client.device_lib import list_local_devices

devices = list_local_devices()
devices
```

Out[44]:

```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 7325002731160755624,
 name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 11139884032
 locality {
   bus_id: 1
   links {
     link {
       device_id: 1
       type: "StreamExecutor"
       strength: 1
     }
   }
 }
 incarnation: 7150956550266107441
 physical_device_desc: "device: 0, name: Tesla K80, pci bus id: 0000:00:04.0, compute cap
ability: 3.7",
 name: "/device:GPU:1"
 device_type: "GPU"
 memory_limit: 11139884032
 locality {
   bus_id: 1
   links {
     link {
       type: "StreamExecutor"
       strength: 1
     }
   }
 }
 incarnation: 15909479382059415698
```

```
 physical_device_desc: "device: 1, name: Tesla K80, pci bus id: 0000:00:05.0, compute cap
ability: 3.7"]
```

Distributed Training

In [45]:

```
keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)
```

In [46]:

```
def create_model():
    return keras.models.Sequential([
        keras.layers.Conv2D(filters=64, kernel_size=7, activation="relu",
                            padding="same", input_shape=[28, 28, 1]),
        keras.layers.MaxPooling2D(pool_size=2),
        keras.layers.Conv2D(filters=128, kernel_size=3, activation="relu",
                            padding="same"),
        keras.layers.Conv2D(filters=128, kernel_size=3, activation="relu",
                            padding="same"),
        keras.layers.MaxPooling2D(pool_size=2),
        keras.layers.Flatten(),
        keras.layers.Dense(units=64, activation='relu'),
        keras.layers.Dropout(0.5),
        keras.layers.Dense(units=10, activation='softmax'),
    ])
```

In [47]:

```
batch_size = 100
model = create_model()
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=keras.optimizers.SGD(learning_rate=1e-2),
              metrics=["accuracy"])
model.fit(X_train, y_train, epochs=10,
          validation_data=(X_valid, y_valid), batch_size=batch_size)
```

```
Epoch 1/10
550/550 [==============================] - 11s 18ms/step - loss: 1.8163 - accuracy: 0.397
9 - val_loss: 0.3446 - val_accuracy: 0.9010
Epoch 2/10
550/550 [==============================] - 9s 17ms/step - loss: 0.4949 - accuracy: 0.8482
- val_loss: 0.1962 - val_accuracy: 0.9458
Epoch 3/10
550/550 [==============================] - 10s 17ms/step - loss: 0.3345 - accuracy: 0.901
2 - val_loss: 0.1343 - val_accuracy: 0.9622
Epoch 4/10
550/550 [==============================] - 10s 17ms/step - loss: 0.2537 - accuracy: 0.926
7 - val_loss: 0.1049 - val_accuracy: 0.9718
Epoch 5/10
550/550 [==============================] - 10s 17ms/step - loss: 0.2099 - accuracy: 0.939
4 - val_loss: 0.0875 - val_accuracy: 0.9752
Epoch 6/10
550/550 [==============================] - 10s 17ms/step - loss: 0.1901 - accuracy: 0.943
9 - val_loss: 0.0797 - val_accuracy: 0.9772
Epoch 7/10
550/550 [==============================] - 10s 18ms/step - loss: 0.1672 - accuracy: 0.950
6 - val_loss: 0.0745 - val_accuracy: 0.9780
Epoch 8/10
550/550 [==============================] - 10s 18ms/step - loss: 0.1537 - accuracy: 0.955
4 - val_loss: 0.0700 - val_accuracy: 0.9804
Epoch 9/10
550/550 [==============================] - 10s 18ms/step - loss: 0.1384 - accuracy: 0.959
2 - val_loss: 0.0641 - val_accuracy: 0.9818
Epoch 10/10
550/550 [==============================] - 10s 18ms/step - loss: 0.1358 - accuracy: 0.960
2 - val_loss: 0.0611 - val_accuracy: 0.9818
```

Out[47]:

```
<tensorflow.python.keras.callbacks.History at 0x7f014831c110>
```

In [48]:

```
keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

distribution = tf.distribute.MirroredStrategy()

# Change the default all-reduce algorithm:
#distribution = tf.distribute.MirroredStrategy(
#    cross_device_ops=tf.distribute.HierarchicalCopyAllReduce())

# Specify the list of GPUs to use:
#distribution = tf.distribute.MirroredStrategy(devices=["/gpu:0", "/gpu:1"])

# Use the central storage strategy instead:
#distribution = tf.distribute.experimental.CentralStorageStrategy()

#if IS_COLAB and "COLAB_TPU_ADDR" in os.environ:
#  tpu_address = "grpc://" + os.environ["COLAB_TPU_ADDR"]
#else:
#  tpu_address = ""
#resolver = tf.distribute.cluster_resolver.TPUClusterResolver(tpu_address)
#tf.config.experimental_connect_to_cluster(resolver)
#tf.tpu.experimental.initialize_tpu_system(resolver)
#distribution = tf.distribute.experimental.TPUStrategy(resolver)

with distribution.scope():
    model = create_model()
    model.compile(loss="sparse_categorical_crossentropy",
                  optimizer=keras.optimizers.SGD(learning_rate=1e-2),
                  metrics=["accuracy"])
```

```
INFO:tensorflow:Using MirroredStrategy with devices ('/job:localhost/replica:0/task:0/dev
ice:GPU:0', '/job:localhost/replica:0/task:0/device:GPU:1')
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then broadcast to
('/job:localhost/replica:0/task:0/device:CPU:0',).
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then broadcast to
('/job:localhost/replica:0/task:0/device:CPU:0',).
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then broadcast to
('/job:localhost/replica:0/task:0/device:CPU:0',).
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then broadcast to
('/job:localhost/replica:0/task:0/device:CPU:0',).
```

In [49]:

```
batch_size = 100 # must be divisible by the number of workers
model.fit(X_train, y_train, epochs=10,
          validation_data=(X_valid, y_valid), batch_size=batch_size)
```

```
Epoch 1/10
INFO:tensorflow:batch_all_reduce: 10 all-reduces with algorithm = nccl, num_packs = 1
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then broadcast to
('/job:localhost/replica:0/task:0/device:CPU:0',).
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then broadcast to
('/job:localhost/replica:0/task:0/device:CPU:0',).
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then broadcast to
('/job:localhost/replica:0/task:0/device:CPU:0',).
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then broadcast to
('/job:localhost/replica:0/task:0/device:CPU:0',).
INFO:tensorflow:batch_all_reduce: 10 all-reduces with algorithm = nccl, num_packs = 1
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then broadcast to
('/job:localhost/replica:0/task:0/device:CPU:0',).
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then broadcast to
('/job:localhost/replica:0/task:0/device:CPU:0',).
550/550 [==============================] - 14s 16ms/step - loss: 1.8193 - accuracy: 0.395
7 - val_loss: 0.3366 - val_accuracy: 0.9102
Epoch 2/10
```

```
550/550 [==============================] - 7s 13ms/step - loss: 0.4886 - accuracy: 0.8497
- val_loss: 0.1865 - val_accuracy: 0.9478
Epoch 3/10
550/550 [==============================] - 7s 13ms/step - loss: 0.3305 - accuracy: 0.9008
- val_loss: 0.1344 - val_accuracy: 0.9616
Epoch 4/10
550/550 [==============================] - 7s 13ms/step - loss: 0.2472 - accuracy: 0.9282
- val_loss: 0.1115 - val_accuracy: 0.9696
Epoch 5/10
550/550 [==============================] - 7s 13ms/step - loss: 0.2020 - accuracy: 0.9425
- val_loss: 0.0873 - val_accuracy: 0.9748
Epoch 6/10
550/550 [==============================] - 7s 13ms/step - loss: 0.1865 - accuracy: 0.9458
- val_loss: 0.0783 - val_accuracy: 0.9764
Epoch 7/10
550/550 [==============================] - 8s 14ms/step - loss: 0.1633 - accuracy: 0.9512
- val_loss: 0.0771 - val_accuracy: 0.9776
Epoch 8/10
550/550 [==============================] - 8s 14ms/step - loss: 0.1422 - accuracy: 0.9570
- val_loss: 0.0705 - val_accuracy: 0.9786
Epoch 9/10
550/550 [==============================] - 7s 13ms/step - loss: 0.1408 - accuracy: 0.9603
- val_loss: 0.0627 - val_accuracy: 0.9830
Epoch 10/10
550/550 [==============================] - 7s 13ms/step - loss: 0.1293 - accuracy: 0.9618
- val_loss: 0.0605 - val_accuracy: 0.9836
```

Out[49]:

```
<tensorflow.python.keras.callbacks.History at 0x7f259bf1a6d0>
```

In [50]:

```
model.predict(X_new)
```

Out[50]:

```
array([[2.53707055e-10, 7.94509292e-10, 1.02021443e-06, 3.37102080e-08,
        4.90816797e-11, 4.37713789e-11, 2.43314297e-14, 9.99996424e-01,
        1.50591750e-09, 2.50736753e-06],
       [1.11715025e-07, 8.56921833e-05, 9.99914169e-01, 6.31697228e-09,
        3.99949344e-11, 4.47976906e-10, 8.46022008e-09, 3.03771834e-08,
        2.91782563e-08, 1.95555502e-10],
       [4.68117065e-07, 9.99787748e-01, 1.01387537e-04, 2.87393277e-06,
        5.29725839e-05, 1.55926125e-06, 2.07211669e-05, 1.76809226e-05,
        9.37155255e-06, 5.19965897e-06]], dtype=float32)
```

Custom training loop:

In [51]:

```
keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

K = keras.backend

distribution = tf.distribute.MirroredStrategy()

with distribution.scope():
    model = create_model()
    optimizer = keras.optimizers.SGD()

with distribution.scope():
    dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train)).repeat().batch(batc
h_size)
    input_iterator = distribution.make_dataset_iterator(dataset)

@tf.function
def train_step():
    def step_fn(inputs):
        X, y = inputs
```

```
        with tf.GradientTape() as tape:
            Y_proba = model(X)
            loss = K.sum(keras.losses.sparse_categorical_crossentropy(y, Y_proba)) / bat
ch_size

        grads = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(grads, model.trainable_variables))
        return loss

    per_replica_losses = distribution.experimental_run(step_fn, input_iterator)
    mean_loss = distribution.reduce(tf.distribute.ReduceOp.SUM,
                                    per_replica_losses, axis=None)
    return mean_loss

n_epochs = 10
with distribution.scope():
    input_iterator.initialize()
    for epoch in range(n_epochs):
        print("Epoch {}/{}".format(epoch + 1, n_epochs))
        for iteration in range(len(X_train) // batch_size):
            print("\rLoss: {:.3f}".format(train_step().numpy()), end="")
        print()
```

```
INFO:tensorflow:Using MirroredStrategy with devices ('/job:localhost/replica:0/task:0/dev
ice:GPU:0', '/job:localhost/replica:0/task:0/device:GPU:1')
WARNING:tensorflow:From <ipython-input-9-acb7c62c8738>:36: DistributedIteratorV1.initiali
ze (from tensorflow.python.distribute.input_lib) is deprecated and will be removed in a f
uture version.
Instructions for updating:
Use the iterator's `initializer` property instead.
Epoch 1/10
INFO:tensorflow:batch_all_reduce: 10 all-reduces with algorithm = nccl, num_packs = 1
INFO:tensorflow:batch_all_reduce: 10 all-reduces with algorithm = nccl, num_packs = 1
Loss: 0.380
Epoch 2/10
Loss: 0.302
Epoch 3/10
Loss: 0.285
Epoch 4/10
Loss: 0.294
Epoch 5/10
Loss: 0.304
Epoch 6/10
Loss: 0.310
Epoch 7/10
Loss: 0.310
Epoch 8/10
Loss: 0.306
Epoch 9/10
Loss: 0.303
Epoch 10/10
Loss: 0.298
```

Training across multiple servers

A TensorFlow cluster is a group of TensorFlow processes running in parallel, usually on different machines, and talking to each other to complete some work, for example training or executing a neural network. Each TF process in the cluster is called a "task" (or a "TF server"). It has an IP address, a port, and a type (also called its role or its job). The type can be `"worker"`, `"chief"`, `"ps"` (parameter server) or `"evaluator"`:

- Each **worker** performs computations, usually on a machine with one or more GPUs.
- The **chief** performs computations as well, but it also handles extra work such as writing TensorBoard logs or saving checkpoints. There is a single chief in a cluster, typically the first worker (i.e., worker #0).
- A **parameter server** (ps) only keeps track of variable values, it is usually on a CPU-only machine.
- The **evaluator** obviously takes care of evaluation. There is usually a single evaluator in a cluster.

The set of tasks that share the same type is often called a "job". For example, the "worker" job is the set of all workers.

To start a TensorFlow cluster, you must first define it. This means specifying all the tasks (IP address, TCP port, and type). For example, the following cluster specification defines a cluster with 3 tasks (2 workers and 1 parameter server). It's a dictionary with one key per job, and the values are lists of task addresses:

In [52]:

```
cluster_spec = {
    "worker": [
        "machine-a.example.com:2222",   # /job:worker/task:0
        "machine-b.example.com:2222"    # /job:worker/task:1
    ],
    "ps": ["machine-c.example.com:2222"] # /job:ps/task:0
}
```

Every task in the cluster may communicate with every other task in the server, so make sure to configure your firewall to authorize all communications between these machines on these ports (it's usually simpler if you use the same port on every machine).

When a task is started, it needs to be told which one it is: its type and index (the task index is also called the task id). A common way to specify everything at once (both the cluster spec and the current task's type and id) is to set the `TF_CONFIG` environment variable before starting the program. It must be a JSON-encoded dictionary containing a cluster specification (under the `"cluster"` key), and the type and index of the task to start (under the `"task"` key). For example, the following `TF_CONFIG` environment variable defines the same cluster as above, with 2 workers and 1 parameter server, and specifies that the task to start is worker #1:

In [53]:

```
import os
import json

os.environ["TF_CONFIG"] = json.dumps({
    "cluster": cluster_spec,
    "task": {"type": "worker", "index": 1}
})
os.environ["TF_CONFIG"]
```

Out[53]:

```
'{"cluster": {"worker": ["machine-a.example.com:2222", "machine-b.example.com:2222"], "ps
": ["machine-c.example.com:2222"]}, "task": {"type": "worker", "index": 1}}'
```

Some platforms (e.g., Google Cloud ML Engine) automatically set this environment variable for you.

TensorFlow's `TFConfigClusterResolver` class reads the cluster configuration from this environment variable:

In [54]:

```
import tensorflow as tf

resolver = tf.distribute.cluster_resolver.TFConfigClusterResolver()
resolver.cluster_spec()
```

Out[54]:

```
ClusterSpec({'ps': ['machine-c.example.com:2222'], 'worker': ['machine-a.example.com:2222
', 'machine-b.example.com:2222']})
```

In [55]:

```
resolver.task_type
```

Out[55]:

```
'worker'
```

In [56]:

```
resolver.task_id
```

Out[56]:

```
1
```

Now let's run a simpler cluster with just two worker tasks, both running on the local machine. We will use the `MultiWorkerMirroredStrategy` to train a model across these two tasks.

The first step is to write the training code. As this code will be used to run both workers, each in its own process, we write this code to a separate Python file, `my_mnist_multiworker_task.py`. The code is relatively straightforward, but there are a couple important things to note:

- **We create the `MultiWorkerMirroredStrategy` before doing anything else with TensorFlow.**
- **Only one of the workers will take care of logging to TensorBoard and saving checkpoints. As mentioned earlier, this worker is called the *chief*, and by convention it is usually worker #0.**

In [57]:

```
%%writefile my_mnist_multiworker_task.py

import os
import numpy as np
import tensorflow as tf
from tensorflow import keras
import time

# At the beginning of the program
distribution = tf.distribute.MultiWorkerMirroredStrategy()

resolver = tf.distribute.cluster_resolver.TFConfigClusterResolver()
print("Starting task {}{}".format(resolver.task_type, resolver.task_id))

# Only worker #0 will write checkpoints and log to TensorBoard
if resolver.task_id == 0:
    root_logdir = os.path.join(os.curdir, "my_mnist_multiworker_logs")
    run_id = time.strftime("run_%Y_%m_%d-%H_%M_%S")
    run_dir = os.path.join(root_logdir, run_id)
    callbacks = [
        keras.callbacks.TensorBoard(run_dir),
        keras.callbacks.ModelCheckpoint("my_mnist_multiworker_model.h5",
                                        save_best_only=True),
    ]
else:
    callbacks = []

# Load and prepare the MNIST dataset
(X_train_full, y_train_full), (X_test, y_test) = keras.datasets.mnist.load_data()
X_train_full = X_train_full[..., np.newaxis] / 255.
X_valid, X_train = X_train_full[:5000], X_train_full[5000:]
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]

with distribution.scope():
    model = keras.models.Sequential([
        keras.layers.Conv2D(filters=64, kernel_size=7, activation="relu",
                            padding="same", input_shape=[28, 28, 1]),
        keras.layers.MaxPooling2D(pool_size=2),
        keras.layers.Conv2D(filters=128, kernel_size=3, activation="relu",
                            padding="same"),
        keras.layers.Conv2D(filters=128, kernel_size=3, activation="relu",
                            padding="same"),
        keras.layers.MaxPooling2D(pool_size=2),
        keras.layers.Flatten(),
        keras.layers.Dense(units=64, activation='relu'),
        keras.layers.Dropout(0.5),
        keras.layers.Dense(units=10, activation='softmax'),
    ])
    model.compile(loss="sparse_categorical_crossentropy",
                  optimizer=keras.optimizers.SGD(learning_rate=1e-2),
```

```
                  metrics=["accuracy"])

model.fit(X_train, y_train, validation_data=(X_valid, y_valid),
          epochs=10, callbacks=callbacks)
```

```
Overwriting my_mnist_multiworker_task.py
```

In a real world application, there would typically be a single worker per machine, but in this example we're running both workers on the same machine, so they will both try to use all the available GPU RAM (if this machine has a GPU), and this will likely lead to an Out-Of-Memory (OOM) error. To avoid this, we could use the `CUDA_VISIBLE_DEVICES` environment variable to assign a different GPU to each worker. Alternatively, we can simply disable GPU support, like this:

In [58]:

```
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
```

We are now ready to start both workers, each in its own process, using Python's `subprocess` module. Before we start each process, we need to set the `TF_CONFIG` environment variable appropriately, changing only the task index:

In [59]:

```
import subprocess

cluster_spec = {"worker": ["127.0.0.1:9901", "127.0.0.1:9902"]}

for index, worker_address in enumerate(cluster_spec["worker"]):
    os.environ["TF_CONFIG"] = json.dumps({
        "cluster": cluster_spec,
        "task": {"type": "worker", "index": index}
    })
    subprocess.Popen("python my_mnist_multiworker_task.py", shell=True)
```

That's it! Our TensorFlow cluster is now running, but we can't see it in this notebook because it's running in separate processes (but if you are running this notebook in Jupyter, you can see the worker logs in Jupyter's server logs).

Since the chief (worker #0) is writing to TensorBoard, we use TensorBoard to view the training progress. Run the following cell, then click on the settings button (i.e., the gear icon) in the TensorBoard interface and check the "Reload data" box to make TensorBoard automatically refresh every 30s. Once the first epoch of training is finished (which may take a few minutes), and once TensorBoard refreshes, the SCALARS tab will appear. Click on this tab to view the progress of the model's training and validation accuracy.

In [60]:

```
%load_ext tensorboard
%tensorboard --logdir=./my_mnist_multiworker_logs --port=6006
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

That's it! Once training is over, the best checkpoint of the model will be available in the `my_mnist_multiworker_model.h5` file. You can load it using `keras.models.load_model()` and use it for predictions, as usual:

In [61]:

```
from tensorflow import keras

model = keras.models.load_model("my_mnist_multiworker_model.h5")
Y_pred = model.predict(X_new)
np.argmax(Y_pred, axis=-1)
```

Out[61]:

```
array([7, 2, 1])
```

And that's all for today! Hope you found this useful. 😊

Exercise Solutions

1. to 8.

See Appendix A.

9.

Exercise: Train a model (any model you like) and deploy it to TF Serving or Google Cloud AI Platform. Write the client code to query it using the REST API or the gRPC API. Update the model and deploy the new version. Your client code will now query the new version. Roll back to the first version.

Please follow the steps in the [Deploying TensorFlow models to TensorFlow Serving] section above.

Please follow the steps in the Deploying TensorFlow models to TensorFlow Serving section above.

10.

Exercise: Train any model across multiple GPUs on the same machine using the `MirroredStrategy` (if you do not have access to GPUs, you can use Colaboratory with a GPU Runtime and create two virtual GPUs). Train the model again using the `CentralStorageStrategy` and compare the training time.

Please follow the steps in the Distributed Training section above.

11.

Exercise: Train a small model on Google Cloud AI Platform, using black box hyperparameter tuning.

Please follow the instructions on pages 716-717 of the book. You can also read this documentation page and go through the example in this nice blog post by Lak Lakshmanan.

In [ ]:

```

```