In [4]:
```
#@title Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

TensorFlow 2 quickstart for beginners

 View on TensorFlow.org  Run in Google Colab  View source on GitHub  Download notebook

This short introduction uses Keras to:

1. Build a neural network that classifies images.
2. Train this neural network.
3. And, finally, evaluate the accuracy of the model.

This is a Google Colaboratory notebook file. Python programs are run directly in the browser—a great way to learn and use TensorFlow. To follow this tutorial, run the notebook in Google Colab by clicking the button at the top of this page.

1. In Colab, connect to a Python runtime: At the top-right of the menu bar, select *CONNECT*.
2. Run all the notebook code cells: Select *Runtime* > *Run all*.

Download and install TensorFlow 2. Import TensorFlow into your program:

Note: Upgrade `pip` to install the TensorFlow 2 package. See the install guide for details.

In [5]:
```
#!pip install tensorflow
import tensorflow as tf
```

Load and prepare the MNIST dataset. Convert the samples from integers to floating-point numbers:

In [6]:
```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
```

Build the `tf.keras.Sequential` model by stacking layers. Choose an optimizer and loss function for training:

In [7]:
```
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)
])
```

For each example the model returns a vector of "logits" or "log-odds" scores, one for each class.

In [8]:
```
predictions = model(x_train[:1]).numpy()
predictions
```

Out[8]:
```
array([[ 0.04991845, -0.54526854, -0.06618283,  0.34738594,  0.5419145 ,
         0.700216  , -0.31279525, -0.18271805,  0.36077833, -0.6906227 ]],
      dtype=float32)
```

The `tf.nn.softmax` function converts these logits to "probabilities" for each class:

In [9]:
```
tf.nn.softmax(predictions).numpy()
```

Out[9]:
```
array([[0.09372372, 0.05168483, 0.0834502 , 0.1261938 , 0.15329242,
        0.17958504, 0.06521162, 0.07427058, 0.12789519, 0.0446927 ]],
      dtype=float32)
```

Note: It is possible to bake this `tf.nn.softmax` in as the activation function for the last layer of the network. While this can make the model output more directly interpretable, this approach is discouraged as it's impossible to provide an exact and numerically stable loss calculation for all models when using a softmax output.

The `losses.SparseCategoricalCrossentropy` loss takes a vector of logits and a `True` index and returns a scalar loss for each example.

In [10]:
```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

This loss is equal to the negative log probability of the true class: It is zero if the model is sure of the correct class.

This untrained model gives probabilities close to random (1/10 for each class), so the initial loss should be close to `-tf.log(1/10) ~= 2.3`.

In [11]:
```
loss_fn(y_train[:1], predictions).numpy()
```

Out[11]:
```
1.7171065
```

In [12]:
```
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
```

The `Model.fit` method adjusts the model parameters to minimize the loss:

In [13]:
```
model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2965 - accura
cy: 0.9137
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1438 - accura
```

```
cy: 0.9577: 0s - loss: 0.1441 - accuracy: 0. - ETA: 0s - loss: 0.1440 - ac
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1073 - accura
cy: 0.9679
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0886 - accura
cy: 0.9724
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0754 - accura
cy: 0.9759
```

Out[13]:
```
<tensorflow.python.keras.callbacks.History at 0x7fab586b4970>
```

The `Model.evaluate` method checks the models performance, usually on a "Validation-set" or "Test-set".

In [14]:
```
model.evaluate(x_test,  y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.0783 - accuracy: 0.9761
```

Out[14]:
```
[0.07827632874250412, 0.9761000275611877]
```

The image classifier is now trained to ~98% accuracy on this dataset. To learn more, read the TensorFlow tutorials.

If you want your model to return a probability, you can wrap the trained model, and attach the softmax to it:

In [15]:
```
probability_model = tf.keras.Sequential([
  model,
  tf.keras.layers.Softmax()
])
```

In [16]:
```
probability_model(x_test[:5])
```

Out[16]:
```
<tf.Tensor: shape=(5, 10), dtype=float32, numpy=
array([[5.6654585e-08, 4.2959244e-10, 1.1458536e-06, 7.0364469e-05,
        1.5496997e-10, 1.4045238e-08, 3.3726233e-14, 9.9992728e-01,
        3.3536256e-07, 8.9967909e-07],
       [8.2618634e-08, 4.2479205e-05, 9.9993896e-01, 1.6392616e-05,
        2.3710192e-14, 1.7800151e-06, 7.4820089e-10, 1.8842066e-15,
        1.8462870e-07, 8.7347751e-12],
       [6.9690714e-08, 9.9944288e-01, 1.2171241e-04, 3.7109655e-06,
        7.7681842e-05, 6.0693164e-06, 3.2691210e-06, 1.7590183e-04,
        1.6824721e-04, 3.1467070e-07],
       [9.9994397e-01, 9.7927595e-09, 4.9758004e-05, 4.4126317e-08,
        3.5000713e-07, 1.1055045e-06, 3.6276213e-06, 2.4023174e-07,
        5.4763141e-09, 8.2756935e-07],
       [8.9672203e-06, 5.3519296e-09, 1.3106385e-06, 3.0591227e-07,
        9.9742281e-01, 5.5576851e-07, 8.5628960e-07, 9.3900817e-05,
        2.8237610e-06, 2.4686062e-03]], dtype=float32)>
```