TensorFlow 2 quickstart for beginners

Run in Google Colab (https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/quic

This short introduction uses Keras (https://www.tensorflow.org/guide/keras/overview) 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 (https://colab.research.google.com/notebooks/welcome.ipynb) 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:

Jpgrade **`pip`** to install the TensorFlow 2 package. See the install guide (https://www.tensorflow.org/instal

```
t tensorflow as tf
```

Load and prepare the MNIST dataset (http://yann.lecun.com/exdb/mnist/). Convert the samples from integers to floating-point numbers:


```
 = tf.keras.datasets.mnist

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

Build the **`tf.keras.Sequential`** (/api_docs/python/tf/keras/Sequential) model by stacking layers. Choose an optimizer and loss function for training:

```
. = tf.keras.models.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dropout(0.2),
keras.layers.Dense(10)
```

For each example the model returns a vector of "logits (https://developers.google.com/machine-learning/glossary#logits)" or "log-odds (https://developers.google.com/machine-learning/glossary#log-odds)" scores, one for each class.

```
.ctions = model(x_train[:1]).numpy()
.ctions
```

```
'([[-0.40252417, -0.36244553, -0.6254247 ,  0.3470046 ,  0.53377753,
    -0.25291196,  0.42313334, -0.85892683,  0.16624598,  0.01534149]],
  dtype=float32)
```

The **`tf.nn.softmax`** (/api_docs/python/tf/nn/softmax) function converts these logits to "probabilities" for each class:

```
ı.softmax(predictions).numpy()
```

```
'([[0.06724608, 0.06999595, 0.05380994, 0.14229287, 0.17151323,
    0.07809851, 0.15354846, 0.04260433, 0.11876287, 0.10212774]],
  dtype=float32)
```


t is possible to bake this **`tf.nn.softmax`** (/api_docs/python/tf/nn/softmax) in as the activation function ver of the network. While this can make the model output more directly interpretable, this approach is raged as it's impossible to provide an exact and numerically stable loss calculation for all models when u ax output.

The `losses.SparseCategoricalCrossentropy` (/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy) loss takes a vector of logits and a `True` index and returns a scalar loss for each example.

```
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`.

```
fn(y_train[:1], predictions).numpy()
```

```
7844
```

```
compile(optimizer='adam',
          loss=loss_fn,
          metrics=['accuracy'])
```

The `Model.fit` (/api_docs/python/tf/keras/Model#fit) method adjusts the model parameters to minimize the loss:

```
fit(x_train, y_train, epochs=5)
```

```
 1/5
1875 [==============================] - 3s 2ms/step - loss: 0.2944 - accuracy
 2/5
1875 [==============================] - 3s 2ms/step - loss: 0.1415 - accuracy
 3/5
1875 [==============================] - 3s 2ms/step - loss: 0.1066 - accuracy
 4/5
1875 [==============================] - 3s 2ms/step - loss: 0.0864 - accuracy
```

The **`Model.evaluate`** (/api_docs/python/tf/keras/Model#evaluate) method checks the models performance, usually on a "Validation-set (https://developers.google.com/machine-learning/glossary#validation-set)" or "Test-set (https://developers.google.com/machine-learning/glossary#test-set)".

```
..evaluate(x_test,  y_test, verbose=2)
```

```
13 - 0s - loss: 0.0766 - accuracy: 0.9773

664269208908081, 0.9772999882698059]
```

The image classifier is now trained to ~98% accuracy on this dataset. To learn more, read the TensorFlow tutorials (https://www.tensorflow.org/tutorials/).

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

```
bility_model = tf.keras.Sequential([
el,
keras.layers.Softmax()
```

```
bility_model(x_test[:5])
```

```
ensor: shape=(5, 10), dtype=float32, numpy=
([[2.8919533e-07, 2.4904485e-09, 3.0876611e-06, 2.8938421e-06,
   2.0726233e-10, 6.4404236e-07, 1.2202126e-12, 9.9998796e-01,
   2.2825203e-08, 5.0464591e-06],
  [1.9124901e-08, 1.9528694e-05, 9.9996126e-01, 1.4802480e-05,
   3.0759908e-13, 1.0432574e-06, 2.1039098e-06, 9.0888365e-13,
   1.2716699e-06, 1.1599616e-12],
  [3.8589778e-08, 9.9956876e-01, 2.9700059e-05, 3.1009504e-06,
   1.9118587e-05, 2.0751606e-06, 5.4347752e-06, 2.9230418e-04,
   7.8798155e-05, 1.7011742e-07],
  [9.9985933e-01, 3.7119894e-11, 1.1895904e-04, 4.9979627e-07,
   3.8895456e-07, 6.6416396e-07, 1.2701520e-05, 6.1525652e-06,
   9.6602575e-09, 1.2553021e-06],
```

[1.4213600e-06 7.6032194e-09 6.6021362e-06 7.5497255e-08

Last updated 2020-06-12.