Ungraded Lab: Lambda Layer

This lab will show how you can define custom layers with the Lambda layer. You can either use lambda functions within the Lambda layer or define a custom function that the Lambda layer will call. Let's get started!

Imports

In [1]:

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

import tensorflow as tf
from tensorflow.keras import backend as K
```

Prepare the Dataset

In [2]:

```
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 [==============================] - 0s 0us/step
```

Build the Model

Here, we'll use a Lambda layer to define a custom layer in our network. We're using a lambda function to get the absolute value of the layer input.

In [3]:

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128),
  tf.keras.layers.Lambda(lambda x: tf.abs(x)),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

In [4]:

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 5s 80us/sample - loss: 0.2236 - accuracy: 0.9370
Epoch 2/5
60000/60000 [==============================] - 5s 75us/sample - loss: 0.0941 - accuracy: 0.9725
Epoch 3/5
60000/60000 [==============================] - 5s 76us/sample - loss: 0.0651 - accuracy: 0.9804
Epoch 4/5
60000/60000 [==============================] - 5s 75us/sample - loss: 0.0493 - accuracy: 0.9850
Epoch 5/5
```

```
60000/60000 [==============================] - 5s 75us/sample - loss: 0.0395 - accuracy: 0.9874
10000/10000 [==============================] - 0s 40us/sample - loss: 0.0799 - accuracy: 0.9757
```

Out[4]:

```
[0.07994030159247341, 0.9757]
```

Another way to use the Lambda layer is to pass in a function defined outside the model. The code below shows how a custom ReLU function is used as a custom layer in the model.

In [5]:

```
def my_relu(x):
    return K.maximum(-0.1, x)

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128),
    tf.keras.layers.Lambda(my_relu),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 5s 79us/sample - loss: 0.2587 - accuracy: 0.9256
Epoch 2/5
60000/60000 [==============================] - 5s 76us/sample - loss: 0.1141 - accuracy: 0.9662
Epoch 3/5
60000/60000 [==============================] - 5s 76us/sample - loss: 0.0791 - accuracy: 0.9767
Epoch 4/5
60000/60000 [==============================] - 5s 76us/sample - loss: 0.0593 - accuracy: 0.9821
Epoch 5/5
60000/60000 [==============================] - 5s 76us/sample - loss: 0.0453 - accuracy: 0.9859
10000/10000 [==============================] - 0s 30us/sample - loss: 0.0830 - accuracy: 0.9738
```

Out[5]:

```
[0.08298594974055887, 0.9738]
```