# C1_W3_Lab_1_lambda-layer

February 8, 2021

## 0.1 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!

## 0.2 Imports

```
[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
```

## 0.3 Prepare the Dataset

```
[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

## 0.4 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.

```
[3]: model = tf.keras.models.Sequential([
       tf.keras.layers.Flatten(input_shape=(28, 28)),
```

1

```
    tf.keras.layers.Dense(128),
    tf.keras.layers.Lambda(lambda x: tf.abs(x)),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

[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 [==============================] - 4s 72us/sample - loss: 0.2191 -
accuracy: 0.9379
Epoch 2/5
60000/60000 [==============================] - 4s 68us/sample - loss: 0.0905 -
accuracy: 0.9722
Epoch 3/5
60000/60000 [==============================] - 4s 68us/sample - loss: 0.0624 -
accuracy: 0.9807
Epoch 4/5
60000/60000 [==============================] - 4s 69us/sample - loss: 0.0500 -
accuracy: 0.9838
Epoch 5/5
60000/60000 [==============================] - 4s 69us/sample - loss: 0.0383 -
accuracy: 0.9880
10000/10000 [==============================] - 0s 36us/sample - loss: 0.0759 -
accuracy: 0.9779
```

[4]: [0.07594203575255233, 0.9779]

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.

[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 [==============================] - 4s 71us/sample - loss: 0.2578 -
accuracy: 0.9266
Epoch 2/5
60000/60000 [==============================] - 4s 69us/sample - loss: 0.1140 -
accuracy: 0.9667
Epoch 3/5
60000/60000 [==============================] - 4s 69us/sample - loss: 0.0795 -
accuracy: 0.9765
Epoch 4/5
60000/60000 [==============================] - 4s 69us/sample - loss: 0.0598 -
accuracy: 0.9817
Epoch 5/5
60000/60000 [==============================] - 4s 69us/sample - loss: 0.0454 -
accuracy: 0.9859
10000/10000 [==============================] - 0s 26us/sample - loss: 0.0888 -
accuracy: 0.9737
```

[5]: [0.08878334205360151, 0.9737]