

✓ **Congratulations! You passed!**

TO PASS 80% or higher

Keep Learning

GRADE
100%

Custom Layers

LATEST SUBMISSION GRADE

100%

1. Lambda layer allows to execute an arbitrary function only within a Sequential API model.

1 / 1 point

- ☐ True
- ☒ False

✓ **Correct**
Correct!

2. Which one of the following is the correct syntax for mapping an increment of 2 to the value of "x" using a Lambda layer? (tf = Tensorflow)

1 / 1 point

- ☒ tf.keras.layers.Lambda(lambda x: tf.math.add(x, 2.0))
- ☐ tf.keras.Lambda(x: tf.math.add(x, 2.0))
- ☐ tf.keras.layers(lambda x: tf.math.add(x, 2.0))
- ☐ tf.keras.layers.Lambda(x: tf.math.add(x, 2.0))

✓ **Correct**
Correct!

3. One drawback of Lambda layers is that you cannot call a custom built function from within them.

1 / 1 point

- ☐ True
- ☒ False

✓ **Correct**
Correct!

4. A Layer is defined by having "States" and "Computation". Consider the following code and check all that are true:

1 / 1 point

```
class SimpleDense(Layer):  
  
    def __init__(self, units=32):  
        super(SimpleDense, self).__init__()  
        self.units = units  
  
    def build(self, input_shape):  
        w_init = tf.random_normal_initializer()  
        self.w = tf.Variable(name="kernel",  
                             initial_value=w_init(shape=(input_shape[-1], self.units),  
                                                    dtype='float32'), trainable=True)  
  
        b_init = tf.zeros_initializer()  
        self.b = tf.Variable(name="bias",  
                             initial_value=b_init(shape=(self.units,), dtype='float32'),  
                             trainable=False)  
  
    def call(self, inputs):  
        return tf.matmul(inputs, self.w) + self.b
```

- ☐ def call(self, inputs): performs the computation and is called when the Class is instantiated.
- ☐ In def __init__(self, units=32): you use the *super* keyword to initialize all of the custom layer attributes
- ☒ You use def build(self, input_shape): to create the state of the layers and specify local input states.

✓ **Correct**
Correct!

- ☐ After training, this class will return a $w \cdot X + b$ computation, where X is the input, w is the weight/kernel tensor with trained values, and b is the bias tensor with trained values.

5. Consider the following code snippet.

1 / 1 point

```
class SimpleDense(Layer):

    def __init__(self, units=32):
        super(SimpleDense, self).__init__()
        self.units = units

    def build(self, input_shape):
        w_init = tf.random_normal_initializer()
        self.w = tf.Variable(name="kernel",
                            initial_value=w_init(shape=(input_shape[-1], self.units),
                                                  dtype='float32'), trainable=True)

        b_init = tf.zeros_initializer()
        self.b = tf.Variable(name="bias",
                            initial_value=b_init(shape=(self.units,), dtype='float32'),
                            trainable=False)

    def call(self, inputs):
        return tf.matmul(inputs, self.w) + self.b
```

What are the function modifications that are needed for passing an activation function to this custom layer implementation?

- ☐ def build(self, units=32, activation=None):
-
-
- self.activation = activation
- def call(self, inputs):
- return self.activation(tf.matmul(inputs, self.w) + self.b)
- ☒ def __init__(self, units=32, activation=None):
-
-
- self.activation = tf.keras.activations.get(activation)
- def call(self, inputs):
- return self.activation(tf.matmul(inputs, self.w) + self.b)
- ☐ def __init__(self, units=32):
-
-
- self.activation = tf.keras.activations.get(activation)
- def call(self, inputs):
- return self.activation(tf.matmul(inputs, self.w) + self.b)
- ☐ def build(self, input_shape):
-
-
- self.activation = tf.keras.activations.get(activation)
- def call(self, inputs):
- return self.activation(tf.matmul(inputs, self.w) + self.b)



Correct
Correct!