**Custom Layers**

**Latest Submission Grade 100%**

**1.**

Question 1

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

**1 / 1 point**



True



False

**Correct**

Correct!

**2.**

Question 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.Lambda(x: tf.math.add(x, 2.0))



tf.keras.layers.Lambda(x: tf.math.add(x, 2.0))



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



tf.keras.layers(lambda x: tf.math.add(x, 2.0))

**Correct**

Correct!

**3.**

Question 3

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

**1 / 1 point**



False



True

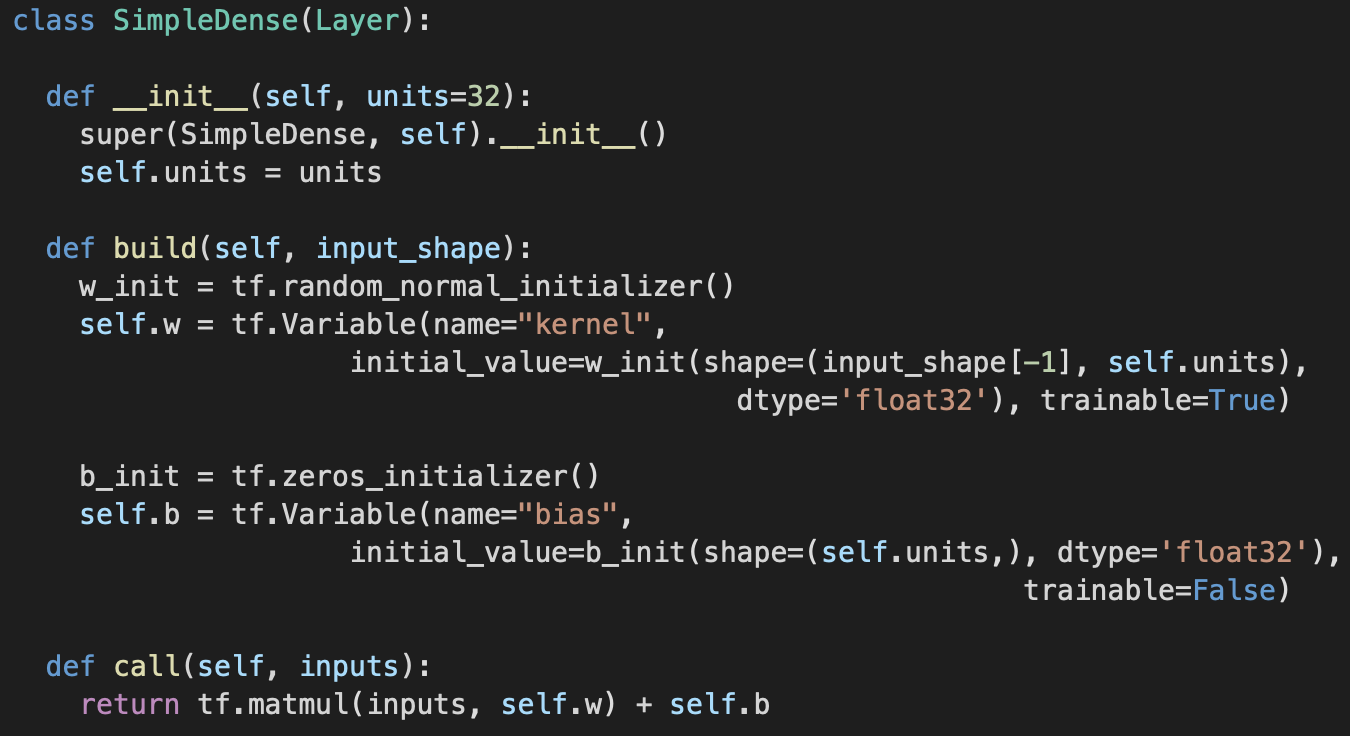
**Correct**

Correct!

**4.**

Question 4

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



**1 / 1 point**



You use def build(self, input\_shape): to create the state of the layers and specify local input states.

**Correct**

Correct!



In def \_\_init\_\_(self, units=32): you use the *super* keyword to initialize all of the custom layer attributes



def call(self, inputs): performs the computation and is called when the Class is instantiated.

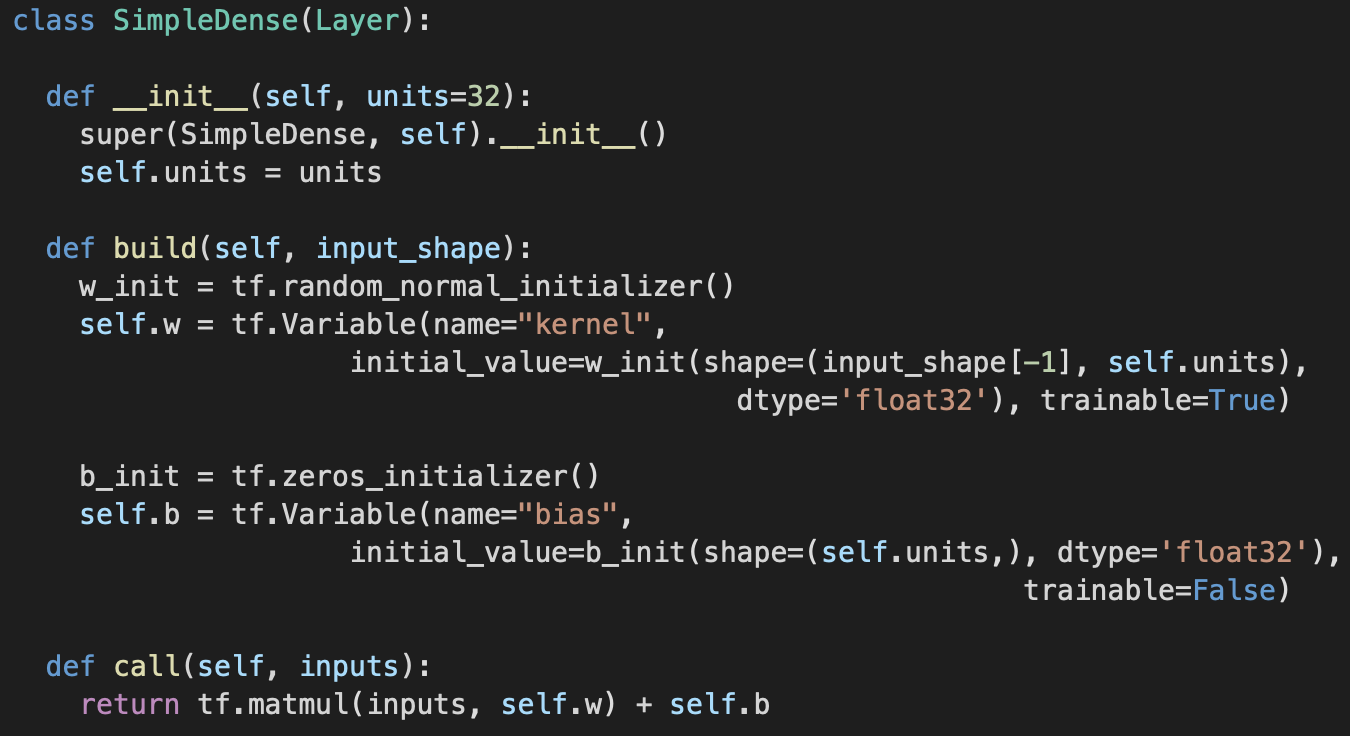


After training, this class will return a w\*X + b computation, where X is the input, w is the weight/kernel tensor with trained values, and b is the bias tensor with trained values.

**5.**

Question 5

Consider the following code snippet.



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

**1 / 1 point**



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)



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 \_\_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 build(self, units=32, activation=None):

self.activation = activation

def call(self, inputs):

return self.activation(tf.matmul(inputs, self.w) + self.b)

**Correct**

Correct!