```
linear.py
              Wed Nov 29 10:26:09 2023
 1: import tensorflow as tf
 2: import numpy as np
 3: import matplotlib.pyplot as plt
 4:
 5:
 6: ## want to initalize our weights in a very specific way
 7: class Linear(tf.Module):
        def __init__(self, num_inputs, num_outputs, first_layer=True, bias=True):
 8:
            super().__init__()
 9:
10:
            rng = tf.random.get_global_generator()
11:
            self.first_layer = first_layer
12:
13:
             # Look into what this needs to be
14:
15:
            if self.first_layer:
                 self.w = tf.Variable(
16:
17:
                     rng.uniform(
18:
                         shape=[num_inputs, num_outputs],
19:
                         minval=(-1 / num\_inputs),
20:
                         maxval=(1 / num_inputs),
21:
                     ),
22:
                     trainable=True,
23:
                     name="Linear/w",
24:
                 )
25:
26:
27:
                Hence, we propose to draw weights with c=6
28:
                 so that wi U(-\hat{a}\210\2326/n, \hat{a}\210\2326/n). This ensures that
29:
                the input to each sine activation is normal
30:
                distributed with a standard deviation of 1.
                 11 11 11
31:
32:
            else:
                 self.w = tf.Variable(
33:
34:
                     rng.uniform(
35:
                         shape=[num_inputs, num_outputs],
36:
                         minval=-np.sqrt(6 / num_inputs) / 30,
37:
                         maxval=np.sqrt(6 / num_inputs) / 30,
38:
                     ),
39:
                     trainable=True,
40:
                     name="Linear/w",
41:
                 )
42:
43:
            self.bias = bias
44:
45:
            if self.bias:
46:
                 self.b = tf.Variable(
47:
                     tf.zeros(
48:
                         shape=[1, num_outputs],
49:
                     ),
50:
                     trainable=True,
51:
                     name="Linear/b",
52:
                 )
53:
54:
        def __call__(self, x):
55:
            z = x @ self.w
56:
57:
            if self.bias:
58:
                 z += self.b
59:
```

return z

60: