C1W3_Assignment

December 15, 2022

1 Week 3 Assignment: Implement a Quadratic Layer

In this week's programming exercise, you will build a custom quadratic layer which computes $y = ax^2 + bx + c$. Similar to the ungraded lab, this layer will be plugged into a model that will be trained on the MNIST dataset. Let's get started!

1.0.1 Imports

[1]:
```
import tensorflow as tf
from tensorflow.keras.layers import Layer

import utils
```

1.0.2 Define the quadratic layer (TODO)

Implement a simple quadratic layer. It has 3 state variables: a, b and c. The computation returned is $ax^2 + bx + c$. Make sure it can also accept an activation function.

`__init__`

- call `super(my_fun, self)` to access the base class of `my_fun`, and call the `__init__()` function to initialize that base class. In this case, `my_fun` is `SimpleQuadratic` and its base class is `Layer`.
- self.units: set this using one of the function parameters.
- self.activation: The function parameter `activation` will be passed in as a string. To get the tensorflow object associated with the string, please use `tf.keras.activations.get()`

`build` The following are suggested steps for writing your code. If you prefer to use fewer lines to implement it, feel free to do so. Either way, you'll want to set `self.a`, `self.b` and `self.c`.

- a_init: set this to tensorflow's `random_normal_initializer()`
- a_init_val: Use the `random_normal_initializer()` that you just created and invoke it, setting the `shape` and `dtype`.

- The `shape` of `a` should have its row dimension equal to the last dimension of `input_shape`, and its column dimension equal to the number of units in the layer.

- This is because you'll be matrix multiplying x^2 * a, so the dimensions should be compatible.
- set the dtype to 'float32'

- self.a: create a tensor using tf.Variable, setting the initial_value and set trainable to True.
- b_init, b_init_val, and self.b: these will be set in the same way that you implemented a_init, a_init_val and self.a
- c_init: set this to `tf.zeros_initializer`.
- c_init_val: Set this by calling the tf.zeros_initializer that you just instantiated, and set the `shape` and `dtype`
  - shape: This will be a vector equal to the number of units. This expects a tuple, and remember that a tuple `(9,)` includes a comma.
  - dtype: set to 'float32'.
- self.c: create a tensor using tf.Variable, and set the parameters `initial_value` and `trainable`.

`call` The following section performs the multiplication x^2a + xb + c. The steps are broken down for clarity, but you can also perform this calculation in fewer lines if you prefer. - x_squared: use tf.math.square() - x_squared_times_a: use tf.matmul().
- If you see an error saying `InvalidArgumentError: Matrix size-incompatible`, please check the order of the matrix multiplication to make sure that the matrix dimensions line up. - x_times_b: use tf.matmul(). - x2a_plus_xb_plus_c: add the three terms together. - activated_x2a_plus_xb_plus_c: apply the class's `activation` to the sum of the three terms.

[40]:
```
# Please uncomment all lines in this cell and replace those marked with `# YOUR␣
↪CODE HERE`.
# You can select all lines in this code cell with Ctrl+A (Windows/Linux) or␣
↪Cmd+A (Mac), then press Ctrl+/ (Windows/Linux) or Cmd+/ (Mac) to uncomment.



class SimpleQuadratic(Layer):

    def __init__(self, units=32, activation=None):
        '''Initializes the class and sets up the internal variables'''
        # YOUR CODE HERE
        super().__init__()
#         super(SimpleQuadratic,self).__init__
        self.units = units
        self.activation = tf.keras.activations.get(activation)

    def build(self, input_shape):
```

```
        '''Create the state of the layer (weights)'''
        # a and b should be initialized with random normal, c (or the bias)␣
↪with zeros.
        # remember to set these as trainable.
        # YOUR CODE HERE
        a_init = tf.random_normal_initializer()
        a_init_val = a_init(shape=(input_shape[-1], self.units), dtype␣
↪='float32')
        self.a =  tf.Variable(name = "kernel", initial_value =␣
↪a_init_val,trainable = True)

        b_init = tf.random_normal_initializer()
        b_init_val = b_init(shape=(input_shape[-1], self.units), dtype␣
↪='float32')
        self.b =  tf.Variable(name = "kernel", initial_value = b_init_val,␣
↪trainable = True)

        c_init = tf.zeros_initializer()
        c_init_val = c_init(shape= (self.units,), dtype='float32')

        self.c = tf.Variable(name = "bias", initial_value=c_init_val,␣
↪trainable=True)
        super().build(input_shape)


    def call(self, inputs):
        '''Defines the computation from inputs to outputs'''
        # Remember to use self.activation() to get the final output
        # YOUR CODE HERE
        x_squared= tf.math.square(inputs)
        x_squared_times_a =  tf.matmul(x_squared, self.a)
        x_times_b = tf.matmul(inputs, self.b)
        x2a_plus_xb_plus_c = x_squared_times_a + x_times_b +  self.c

        return self.activation(x2a_plus_xb_plus_c)
```

Test your implementation

[41]:
```
utils.test_simple_quadratic(SimpleQuadratic)
```

```
All public tests passed
```

You can now train the model with the `SimpleQuadratic` layer that you just implemented. Please uncomment the cell below to run the training. When you get the expected results, you will need to comment this block again before submitting the notebook to the grader.

[42]:
```
# # You can select all lines in this code cell with Ctrl+A (Windows/Linux) or
↪Cmd+A (Mac), then press Ctrl+/ (Windows/Linux) or Cmd+/ (Mac) to uncomment.

# # THIS CODE SHOULD RUN WITHOUT MODIFICATION
# # AND SHOULD RETURN TRAINING/TESTING ACCURACY at 97%+

# 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

# model = tf.keras.models.Sequential([
#   tf.keras.layers.Flatten(input_shape=(28, 28)),
#   SimpleQuadratic(128, activation='relu'),
#   tf.keras.layers.Dropout(0.2),
#   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 [==============================] - 14s 230us/sample - loss: 0.2701 -
accuracy: 0.9196
Epoch 2/5
60000/60000 [==============================] - 14s 227us/sample - loss: 0.1340 -
accuracy: 0.95933s - loss: 0.1566 - accuracy: 0 - E
Epoch 3/5
60000/60000 [==============================] - 14s 228us/sample - loss: 0.1020 -
accuracy: 0.9685
Epoch 4/5
60000/60000 [==============================] - 14s 227us/sample - loss: 0.0844 -
accuracy: 0.9731- loss: 0.0845 - accuracy: 0.97 - ETA: 0s - loss: 0.0847 - ac
Epoch 5/5
60000/60000 [==============================] - 14s 227us/sample - loss: 0.0709 -
accuracy: 0.9773- loss: 0.0706 - accuracy:
10000/10000 [==============================] - 1s 81us/sample - loss: 0.0743 -
accuracy: 0.9780
```

[42]:
```
[0.07433946279445663, 0.978]
```

IMPORTANT

Before submitting, please make sure to follow these steps to avoid getting timeout issues with the grader:

1. Make sure to pass all public tests and get an accuracy greater than 97%.
2. Click inside the training code cell above.
3. Select all lines in this code cell with `Ctrl+A` (Windows/Linux) or `Cmd+A` (Mac), then press `Ctrl+/` (Windows/Linux) or `Cmd+/` (Mac) to comment the entire block. All lines should turn green as before.
4. From the menu bar, click `File > Save and Checkpoint`. *This is important.*
5. Once saved, click the `Submit Assignment` button.