Gradient Tape Basics

In this ungraded lab, you'll get familiar with Tensorflow's built in API called Gradient Tape which helps in performing automatic differentiation.

Imports

In [1]:

```
import tensorflow as tf
```

Exercise on basics of Gradient Tape

Let's explore how you can use tf.GradientTape() to do automatic differentiation.

In [2]:

```
# Define a 2x2 array of 1's
x = tf.ones((2,2))

with tf.GradientTape() as t:
    # Record the actions performed on tensor x with `watch`
    t.watch(x)

    # Define y as the sum of the elements in x
    y =  tf.reduce_sum(x)

    # Let z be the square of y
    z = tf.square(y)

# Get the derivative of z wrt the original input tensor x
dz_dx = t.gradient(z, x)

# Print our result
print(dz_dx)
```

```
tf.Tensor(
[[8. 8.]
 [8. 8.]], shape=(2, 2), dtype=float32)
```

Gradient tape expires after one use, by default

If you want to compute multiple gradients, note that by default, GradientTape is not persistent (`persistent=False`). This means that the GradientTape will expire after you use it to calculate a gradient.

To see this, set up gradient tape as usual and calculate a gradient, so that the gradient tape will be 'expired'.

In [3]:

```
x = tf.constant(3.0)

# Notice that persistent is False by default
with tf.GradientTape() as t:
    t.watch(x)

    # y = x^2
    y = x * x

    # z = y^2
    z = y * y

# Compute dz/dx. 4 * x^3 at x = 3 --> 108.0
dz_dx = t.gradient(z, x)
print(dz_dx)
```

```
tf.Tensor(108.0, shape=(), dtype=float32)
```

Gradient tape has expired

See what happens if you try to calculate another gradient after you've already used gradient tape once.

In [4]:

```
# If you try to compute dy/dx after the gradient tape has expired:
try:
    dy_dx = t.gradient(y, x)  # 6.0
    print(dy_dx)
except RuntimeError as e:
    print("The error message you get is:")
    print(e)
```

```
The error message you get is:
GradientTape.gradient can only be called once on non-persistent tapes.
```

Make the gradient tape persistent

To make sure that the gradient tape can be used multiple times, set `persistent=True`

In [5]:

```
x = tf.constant(3.0)

# Set persistent=True so that you can reuse the tape
with tf.GradientTape(persistent=True) as t:
    t.watch(x)

    # y = x^2
    y = x * x

    # z = y^2
    z = y * y

# Compute dz/dx. 4 * x^3 at x = 3 --> 108.0
dz_dx = t.gradient(z, x)
print(dz_dx)
```

```
tf.Tensor(108.0, shape=(), dtype=float32)
```

Now that it's persistent, you can still reuse this tape!

Try calculating a second gradient on this persistent tape.

In [6]:

```
# You can still compute dy/dx because of the persistent flag.
dy_dx = t.gradient(y, x)  # 6.0
print(dy_dx)
```

```
tf.Tensor(6.0, shape=(), dtype=float32)
```

Great! It still works! Delete the tape variable `t` once you no longer need it.

In [7]:

```
# Drop the reference to the tape
del t
```

Nested Gradient tapes

Now let's try computing a higher order derivative by nesting the `GradientTapes:`

Acceptable indentation of the first gradient calculation

Keep in mind that you'll want to make sure that the first gradient calculation of `dy_dx` should occur at least inside the outer `with` block.

In [8]:

```
x = tf.Variable(1.0)

with tf.GradientTape() as tape_2:
    with tf.GradientTape() as tape_1:
        y = x * x * x

    # The first gradient calculation should occur at leaset
    # within the outer with block
    dy_dx = tape_1.gradient(y, x)
d2y_dx2 = tape_2.gradient(dy_dx, x)

print(dy_dx)
print(d2y_dx2)
```

```
tf.Tensor(3.0, shape=(), dtype=float32)
tf.Tensor(6.0, shape=(), dtype=float32)
```

The first gradient calculation can also be inside the inner with block.

In [9]:

```
x = tf.Variable(1.0)

with tf.GradientTape() as tape_2:
    with tf.GradientTape() as tape_1:
        y = x * x * x

        # The first gradient calculation can also be within the inner with block
        dy_dx = tape_1.gradient(y, x)
d2y_dx2 = tape_2.gradient(dy_dx, x)

print(dy_dx)
print(d2y_dx2)
```

```
tf.Tensor(3.0, shape=(), dtype=float32)
tf.Tensor(6.0, shape=(), dtype=float32)
```

Where not to indent the first gradient calculation

If the first gradient calculation is OUTSIDE of the outer `with` block, it won't persist for the second gradient calculation.

In [10]:

```
x = tf.Variable(1.0)

with tf.GradientTape() as tape_2:
    with tf.GradientTape() as tape_1:
        y = x * x * x

# The first gradient call is outside the outer with block
# so the tape will expire after this
dy_dx = tape_1.gradient(y, x)

# The tape is now expired and the gradient output will be `None`
d2y_dx2 = tape_2.gradient(dy_dx, x)

print(dy_dx)
print(d2y_dx2)
```

```
tf.Tensor(3.0, shape=(), dtype=float32)
```

```
None
```

Notice how the `d2y_dx2` calculation is now `None`. The tape has expired. Also note that this still won't work even if you set persistent=True for both gradient tapes.

In [11]:

```
x = tf.Variable(1.0)

# Setting persistent=True still won't work
with tf.GradientTape(persistent=True) as tape_2:
    # Setting persistent=True still won't work
    with tf.GradientTape(persistent=True) as tape_1:
        y = x * x * x

# The first gradient call is outside the outer with block
# so the tape will expire after this
dy_dx = tape_1.gradient(y, x)

# the output will be `None`
d2y_dx2 = tape_2.gradient(dy_dx, x)

print(dy_dx)
print(d2y_dx2)
```

```
tf.Tensor(3.0, shape=(), dtype=float32)
None
```

Proper indentation for the second gradient calculation

The second gradient calculation `d2y_dx2` can be indented as much as the first calculation of `dy_dx` but not more.

In [12]:

```
x = tf.Variable(1.0)

with tf.GradientTape() as tape_2:
    with tf.GradientTape() as tape_1:
        y = x * x * x

        dy_dx = tape_1.gradient(y, x)

        # this is acceptable
        d2y_dx2 = tape_2.gradient(dy_dx, x)

print(dy_dx)
print(d2y_dx2)
```

```
tf.Tensor(3.0, shape=(), dtype=float32)
tf.Tensor(6.0, shape=(), dtype=float32)
```

This is also acceptable

In [13]:

```
x = tf.Variable(1.0)

with tf.GradientTape() as tape_2:
    with tf.GradientTape() as tape_1:
        y = x * x * x

        dy_dx = tape_1.gradient(y, x)

    # this is also acceptable
    d2y_dx2 = tape_2.gradient(dy_dx, x)

print(dy_dx)
print(d2y_dx2)
```

```
tf.Tensor(3.0, shape=(), dtype=float32)
tf.Tensor(6.0, shape=(), dtype=float32)
```

This is also acceptable

In [14]:

```
x = tf.Variable(1.0)

with tf.GradientTape() as tape_2:
    with tf.GradientTape() as tape_1:
        y = x * x * x

        dy_dx = tape_1.gradient(y, x)

# this is also acceptable
d2y_dx2 = tape_2.gradient(dy_dx, x)

print(dy_dx)
print(d2y_dx2)
```

```
tf.Tensor(3.0, shape=(), dtype=float32)
tf.Tensor(6.0, shape=(), dtype=float32)
```