C2_W1_Lab_1_basic-tensors

February 4, 2025

1 Basic Tensors

In this ungraded lab, you will try some of the basic operations you can perform on tensors.

1.1 Imports

[1]:
```
try:
    # %tensorflow_version only exists in Colab.
    %tensorflow_version 2.x
except Exception:
    pass

import tensorflow as tf
import numpy as np
```

1.2 Exercise on basic Tensor operations

Lets create a single dimension numpy array on which you can perform some operation. You'll make an array of size 25, holding values from 0 to 24.

[2]:
```
# Create a 1D uint8 NumPy array comprising of first 25 natural numbers
x = np.arange(0, 25)
x
```

[2]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

Now that you have your 1-D array, next you'll change that array into a `tensor`. After running the code block below, take a moment to inspect the information of your tensor.

[3]:
```
# Convert NumPy array to Tensor using `tf.constant`
x = tf.constant(x)
x
```

[3]:
```
<tf.Tensor: shape=(25,), dtype=int64, numpy=
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
```

```
       17, 18, 19, 20, 21, 22, 23, 24])>
```

As the first operation to be performed, you'll square (element-wise) all the values in the tensor `x`

[4]:
```
# Square the input tensor x
x = tf.square(x)
x
```

[4]:
```
<tf.Tensor: shape=(25,), dtype=int64, numpy=
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100, 121, 144,
       169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576])>
```

One feature of tensors is that they can be reshaped. When reshpaing, make sure you consider dimensions that will include all of the values of the tensor.

[5]:
```
# Reshape tensor x into a 5 x 5 matrix.
x = tf.reshape(x, (5, 5))
x
```

[5]:
```
<tf.Tensor: shape=(5, 5), dtype=int64, numpy=
array([[  0,   1,   4,   9,  16],
       [ 25,  36,  49,  64,  81],
       [100, 121, 144, 169, 196],
       [225, 256, 289, 324, 361],
       [400, 441, 484, 529, 576]])>
```

Notice that you'll get an error message if you choose a shape that cannot be exactly filled with the values of the given tensor.
* Run the cell below and look at the error message * Try to change the tuple that is passed to `shape` to avoid an error.

[6]:
```
# Try this and look at the error
# Try to change the input to `shape` to avoid an error
tmp = tf.constant([1,2,3,4])
tf.reshape(tmp, shape=(2,3))
```

```
␣
↪---------------------------------------------------------------------------

       InvalidArgumentError                      Traceback (most recent call␣
↪last)

       <ipython-input-6-4a03bca24981> in <module>
         2 # Try to change the input to `shape` to avoid an error
         3 tmp = tf.constant([1,2,3,4])
   ----> 4 tf.reshape(tmp, shape=(2,3))
```

```
      /opt/conda/lib/python3.7/site-packages/tensorflow/python/util/dispatch.
↪py in wrapper(*args, **kwargs)
      199     """Call target, and fall back on dispatchers if there is a␣
↪TypeError."""
      200     try:
  --> 201       return target(*args, **kwargs)
      202     except (TypeError, ValueError):
      203       # Note: convert_to_eager_tensor currently raises a ValueError,␣
↪not a


      /opt/conda/lib/python3.7/site-packages/tensorflow/python/ops/array_ops.
↪py in reshape(tensor, shape, name)
      193     A `Tensor`. Has the same type as `tensor`.
      194   """
  --> 195   result = gen_array_ops.reshape(tensor, shape, name)
      196   tensor_util.maybe_set_static_shape(result, shape)
      197   return result


      /opt/conda/lib/python3.7/site-packages/tensorflow/python/ops/
↪gen_array_ops.py in reshape(tensor, shape, name)
     8227     try:
     8228       return reshape_eager_fallback(
  -> 8229           tensor, shape, name=name, ctx=_ctx)
     8230     except _core._SymbolicException:
     8231       pass  # Add nodes to the TensorFlow graph.


      /opt/conda/lib/python3.7/site-packages/tensorflow/python/ops/
↪gen_array_ops.py in reshape_eager_fallback(tensor, shape, name, ctx)
     8252   _attrs = ("T", _attr_T, "Tshape", _attr_Tshape)
     8253   _result = _execute.execute(b"Reshape", 1, inputs=_inputs_flat,␣
↪attrs=_attrs,
  -> 8254                              ctx=ctx, name=name)
     8255   if _execute.must_record_gradient():
     8256     _execute.record_gradient(


      /opt/conda/lib/python3.7/site-packages/tensorflow/python/eager/execute.
↪py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
       58     ctx.ensure_initialized()
       59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name,␣
↪op_name,
  ---> 60                                         inputs, attrs, num_outputs)
       61   except core._NotOkStatusException as e:
```

```
     62     if name is not None:


InvalidArgumentError: Input to reshape is a tensor with 4 values, but the requested shape has 6 [Op:Reshape]
```

Like reshaping, you can also change the data type of the values within the tensor. Run the cell below to change the data type from `int` to `float`

[7]:
```
# Cast tensor x into float32. Notice the change in the dtype.
x = tf.cast(x, tf.float32)
x
```

[7]:
```
<tf.Tensor: shape=(5, 5), dtype=float32, numpy=
array([[  0.,   1.,   4.,   9.,  16.],
       [ 25.,  36.,  49.,  64.,  81.],
       [100., 121., 144., 169., 196.],
       [225., 256., 289., 324., 361.],
       [400., 441., 484., 529., 576.]], dtype=float32)>
```

Next, you'll create a single value float tensor by the help of which you'll see `broadcasting` in action

[8]:
```
# Let's define a constant and see how broadcasting works in the following cell.
y = tf.constant(2, dtype=tf.float32)
y
```

[8]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=2.0>
```

Multiply the tensors `x` and `y` together, and notice how multiplication was done and its result.

[9]:
```
# Multiply tensor `x` and `y`. `y` is multiplied to each element of x.
result = tf.multiply(x, y)
result
```

[9]:
```
<tf.Tensor: shape=(5, 5), dtype=float32, numpy=
array([[   0.,    2.,    8.,   18.,   32.],
       [  50.,   72.,   98.,  128.,  162.],
       [ 200.,  242.,  288.,  338.,  392.],
       [ 450.,  512.,  578.,  648.,  722.],
       [ 800.,  882.,  968., 1058., 1152.]], dtype=float32)>
```

Re-Initialize `y` to a tensor having more values.

[10]:
```
# Now let's define an array that matches the number of row elements in the `x` array.
y = tf.constant([1, 2, 3, 4, 5], dtype=tf.float32)
y
```

[10]:
```
<tf.Tensor: shape=(5,), dtype=float32, numpy=array([1., 2., 3., 4., 5.],
dtype=float32)>
```

[11]:
```
# Let's see first the contents of `x` again.
x
```

[11]:
```
<tf.Tensor: shape=(5, 5), dtype=float32, numpy=
array([[  0.,   1.,   4.,   9.,  16.],
       [ 25.,  36.,  49.,  64.,  81.],
       [100., 121., 144., 169., 196.],
       [225., 256., 289., 324., 361.],
       [400., 441., 484., 529., 576.]], dtype=float32)>
```

Add the tensors `x` and `y` together, and notice how addition was done and its result.

[12]:
```
# Add tensor `x` and `y`. `y` is added element wise to each row of `x`.
result = x + y
result
```

[12]:
```
<tf.Tensor: shape=(5, 5), dtype=float32, numpy=
array([[  1.,   3.,   7.,  13.,  21.],
       [ 26.,  38.,  52.,  68.,  86.],
       [101., 123., 147., 173., 201.],
       [226., 258., 292., 328., 366.],
       [401., 443., 487., 533., 581.]], dtype=float32)>
```

1.2.1 The shape parameter for tf.constant

When using `tf.constant()`, you can pass in a 1D array (a vector) and set the `shape` parameter to turn this vector into a multi-dimensional array.

[13]:
```
tf.constant([1,2,3,4], shape=(2,2))
```

[13]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[1, 2],
       [3, 4]], dtype=int32)>
```

1.2.2 The shape parameter for tf.Variable

Note, however, that for `tf.Variable()`, the shape of the tensor is derived from the shape given by the input array. Setting `shape` to something other than `None` will not reshape a 1D array into a multi-dimensional array, and will give a `ValueError`.

[14]:
```
try:
    # This will produce a ValueError
    tf.Variable([1,2,3,4], shape=(2,2))
```

```
except ValueError as v:
    # See what the ValueError says
    print(v)
```

```
The initial value's shape ((4,)) is not compatible with the explicitly supplied
`shape` argument ((2, 2)).
```

[ ]: