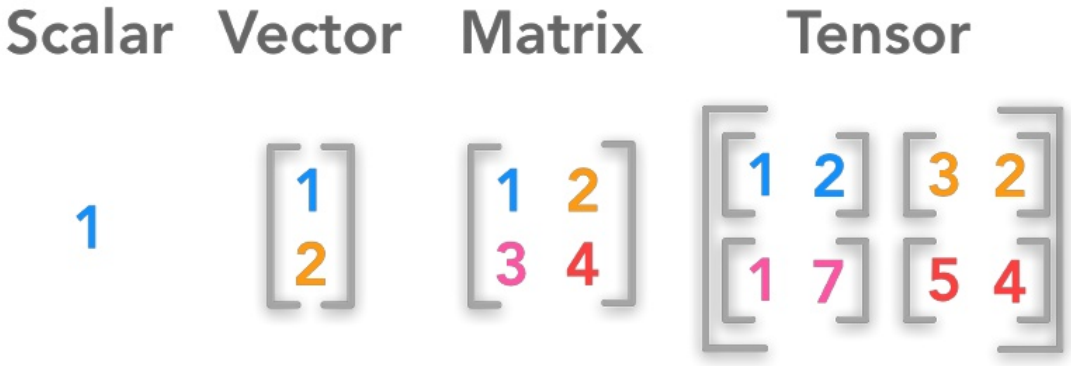
Tensors Introduction

May 25, 2021

1 Introduction to Tensor Notes :

A tensor is a container which can house data in N dimensions, along with its linear operations, though there is nuance in what tensors technically are and what we refer to as tensors in practice.



[1]:
```
#Import Tensorflow
import tensorflow as tf
print(tf.__version__)
import numpy
```

```
2.3.0
```

2 Create tesnors with tf.constant()

[2]:
```
scalar = tf.constant(7)
scalar
```

[2]:
```
<tf.Tensor: shape=(), dtype=int32, numpy=7>
```

[3]:
```
#Check the number of dimensions of tensor using ndim
scalar.ndim
```

[3]:
```
0
```

[4]:
```
#Create a vector
vector = tf.constant([10,10])
vector
```

[4]:
```
<tf.Tensor: shape=(2,), dtype=int32, numpy=array([10, 10])>
```

[5]:
```
vector.ndim
```

[5]:
```
1
```

[6]:
```
#Create a matrix
matrix = tf.constant([[10,7],
                      [7,10]]
                      )
matrix
```

[6]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[10,  7],
       [ 7, 10]])>
```

[7]:
```
matrix.ndim
```

[7]:
```
2
```

At this point, we can relate that ndim represents the number of elements in the shape tuple

[8]:
```
#Create another matric
matrix_2 = tf.constant([[10.,7.],
                        [3.,2.],
                         [1.,2.]],dtype=tf.float16) #Here we use float16 since␣
 ↪our numbers are small
matrix_2
```

[8]:
```
<tf.Tensor: shape=(3, 2), dtype=float16, numpy=
array([[10.,  7.],
       [ 3.,  2.],
       [ 1.,  2.]], dtype=float16)>
```

[9]:
```
matrix_2.ndim
```

[9]:
```
2
```

[10]:
```
#Create a tensor
tensor = tf.constant([[[1,2,3],
                       [4,5,6]],
                        [[7,8,9],
                         [10,11,12]],
                        [[13,14,15],
                         [16,17,18]]])
tensor
```

[10]:
```
<tf.Tensor: shape=(3, 2, 3), dtype=int32, numpy=
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]],

       [[13, 14, 15],
        [16, 17, 18]]])>
```

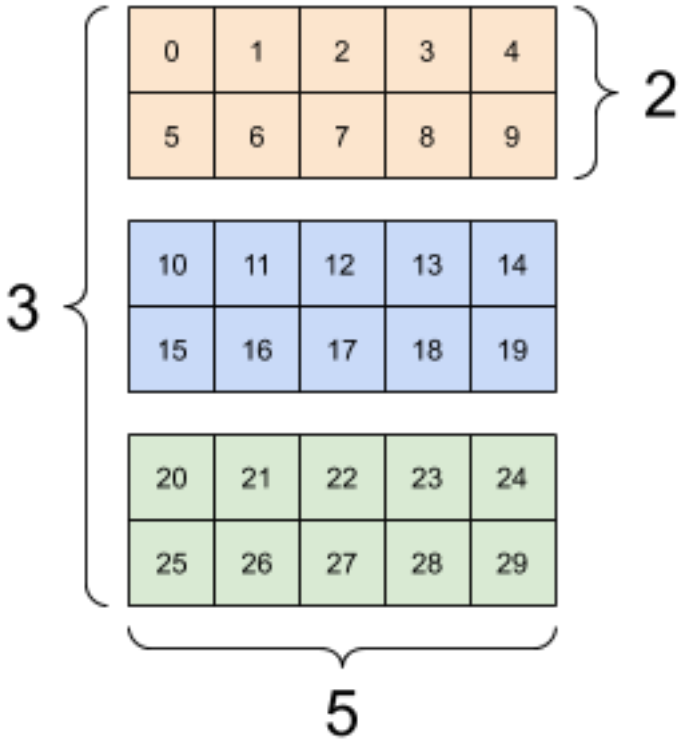
[11]:
```
rank_3_tensor = tf.constant([
  [[0, 1, 2, 3, 4],
   [5, 6, 7, 8, 9]],
  [[10, 11, 12, 13, 14],
   [15, 16, 17, 18, 19]],
  [[20, 21, 22, 23, 24],
   [25, 26, 27, 28, 29]],])

print(rank_3_tensor)
```

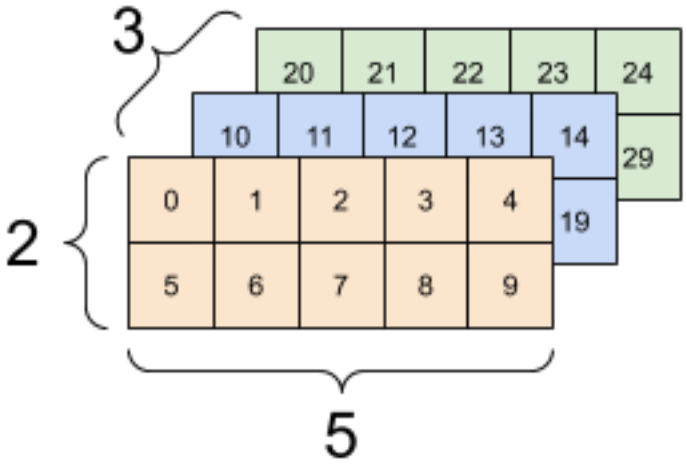
```
tf.Tensor(
[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]]

 [[10 11 12 13 14]
  [15 16 17 18 19]]

 [[20 21 22 23 24]
  [25 26 27 28 29]]], shape=(3, 2, 5), dtype=int32)
```



The above example for a 3 dimensional tensor represents : Number of matrices , Number of rows in a matrices and Number of columns in a matrices i.e. Shape = (3,2,5). You can also visualise this as matrices stacked on top of each other to produce a 3D structure as shown below



2.1 Summary so far :

1. Scalar : Single number
2. Vector : A number with both direction and magnitude
3. Matrix : A 2 dimensional array of numbers
4. Tensor : A n-dimensional array of numbers which can constitude all of the above as well.

3 Create tesnors with tf.Variable()

[12]:
```
#Create a tensor with tf.Variable and see the difference between tf.constant

changeable_tensor = tf.Variable([10,10])
unchageable_tensor = tf.constant([10,10])
```

[13]:
```
changeable_tensor , unchageable_tensor
```

[13]:
```
(<tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([10, 10])>,
 <tf.Tensor: shape=(2,), dtype=int32, numpy=array([10, 10])>)
```

[14]:
```
# Changing element in the changeable tensor
changeable_tensor[0]
```

[14]:
```
<tf.Tensor: shape=(), dtype=int32, numpy=10>
```

This gives a values of numpy 10

[15]:
```
#Now lets try channging using assignment method
changeable_tensor[0] = 7
```

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

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-15-204a6df08bff> in <module>
  1 #Now lets try channging using assignment method
----> 2 changeable_tensor[0] = 7


TypeError: 'ResourceVariable' object does not support item assignment
```

We see that the changeable tensor doesnt allow item assignment. This is where we refer to tensorflow documentation to see how to assign values to change the tensor.

[16]:
```
#using the assign method to change the value of changeable tensor
changeable_tensor[0].assign(7)
changeable_tensor
```

[16]:
```
<tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([ 7, 10])>
```

[17]:
```
#Trying the same as above for unchangeable tensor
unchageable_tensor[0].assign(7)
unchageable_tensor
```

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

AttributeError                            Traceback (most recent call␣
↪last)

<ipython-input-17-289acc285a83> in <module>
  1 #Trying the same as above for unchangeable tensor
----> 2 unchageable_tensor[0].assign(7)
  3 unchageable_tensor


AttributeError: 'tensorflow.python.framework.ops.EagerTensor' object has␣
↪no attribute 'assign'
```

Thus, the above examples conclude the difference between varaible and constant tensors. > **Note 1:** If you declare a tf.Variable, you can change it's value later on if you want to. On the other hand, tf.constant is immutable, meaning that once you define it you can't change its value.

> **Note 2:** Most of the time in practice you will need to decide between using tf.constant or tf.variable depending on the use case. However, most of the time, Tensorflwo will automatically decide or choose for you when loading or modelling the data

4 Create random tensors

Random tensors are tensors of some abitrary size which contain random numbers. Why would you want to create random tensors? This is what neural networks use to intialize their weights (patterns) that they're trying to learn in the data.

[18]:
```
#Create two random tensors
random_tensor_1 = tf.random.Generator.from_seed(42) #seed is used for␣
 ↪reproducability
random_tensor_1=random_tensor_1.normal(shape=(3,2))
random_tensor_1
```

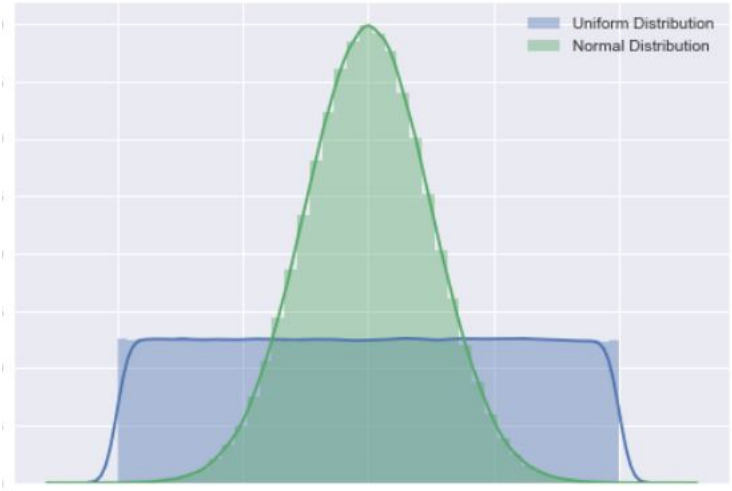
[18]:
```
<tf.Tensor: shape=(3, 2), dtype=float32, numpy=
array([[-0.7565803 , -0.06854702],
       [ 0.07595026, -1.2573844 ],
       [-0.23193763, -1.8107855 ]], dtype=float32)>
```

[19]:
```
random_tensor_2 = tf.random.Generator.from_seed(42)
random_tensor_2=random_tensor_2.uniform(shape=(3,2))
random_tensor_2
```

[19]:
```
<tf.Tensor: shape=(3, 2), dtype=float32, numpy=
array([[0.7493447 , 0.73561966],
```

```
       [0.45230794, 0.49039817],
       [0.1889317 , 0.52027524]], dtype=float32)>
```

Note 3: Normal Distribution Vs Uniform Distribution Normal Distribution is a probability distribution where probability of x is highest at centre and lowest in the ends whereas in Uniform Distribution probability of x is constant.



5 Shuffling order of elements in a Tensor

Why do we want to shuffle the elements in a Tensor? Let's say you working with 15,000 images of cats and dogs and the first 10,000 images of were of cats and the next 5,000 were of dogs. This order could effect how a neural network learns (it may overfit by learning the order of the data), instead, it might be a good idea to move your data around.

[20]:
```
not_shuffled = tf.constant([[1,2],
                            [3,4],
                            [5,6]])
not_shuffled , not_shuffled.ndim
```

[20]:
```
(<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
 array([[1, 2],
        [3, 4],
        [5, 6]])>,
 2)
```

[21]:
```
#Shuffling the above tensor
tf.random.shuffle(not_shuffled)
```

[21]:
```
<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
array([[3, 4],
       [5, 6],
       [1, 2]])>
```

The above tf.random.shuffle is shuffled around based on the first dimension

6 Other methods to creating Tensors

[22]:
```
#1. Tensorflow operation similar to numpy ones
tf.ones([5,5],dtype='int32')
```

[22]:
```
<tf.Tensor: shape=(5, 5), dtype=int32, numpy=
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])>
```

[23]:
```
#2. Tensorflow operation similar to numpy zeroes
tf.zeros(shape=(5,5),dtype='int32')
```

[23]:
```
<tf.Tensor: shape=(5, 5), dtype=int32, numpy=
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])>
```

6.1 Turn numpy arrays into Tensors

Note 4: Why Tensors over Numpy arrays ? This because TensorFlow tesnors can be run on a GPU much faster for numerical computing than numpy

[24]:
```
#Numpy into Tensors

import numpy as np
numpy_A = np.arange(1,25,dtype=np.int32)
numpy_A , numpy_A.shape
```

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

[25]:
```
#converting above numpy_A to tensor
A = tf.constant(numpy_A, shape=(2,3,4))
A
```

[25]:
```
<tf.Tensor: shape=(2, 3, 4), dtype=int32, numpy=
array([[[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]],

       [[13, 14, 15, 16],
        [17, 18, 19, 20],
```

```
       [21, 22, 23, 24]]])>
```

[26]:
```
B = tf.constant(numpy_A,shape=(6,4))
B
```

[26]:
```
<tf.Tensor: shape=(6, 4), dtype=int32, numpy=
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20],
       [21, 22, 23, 24]])>
```

[27]:
```
#Now trying to make a tensor with different shape which doesnt multiplies to 24
C = tf.constant(numpy_A,shape=(10,2))
C
```

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

       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-27-9703eb7c673f> in <module>
         1 #Now trying to make a tensor with different shape which doesnt␣
↪multiplies to 24
   ----> 2 C = tf.constant(numpy_A,shape=(10,2))
         3 C


       D:
↪\Anaconda\envs\fyp\lib\site-packages\tensorflow\python\framework\constant_op.
↪py in constant(value, dtype, shape, name)
       262   """
       263   return _constant_impl(value, dtype, shape, name,␣
↪verify_shape=False,
   --> 264                         allow_broadcast=True)
       265
       266


       D:
↪\Anaconda\envs\fyp\lib\site-packages\tensorflow\python\framework\constant_op.
↪py in _constant_impl(value, dtype, shape, name, verify_shape, allow_broadcast)
       273       with trace.Trace("tf.constant"):
       274         return _constant_eager_impl(ctx, value, dtype, shape,␣
↪verify_shape)
```

```
    --> 275     return _constant_eager_impl(ctx, value, dtype, shape,␣
↪verify_shape)
        276
        277   g = ops.get_default_graph()


        D:
↪\Anaconda\envs\fyp\lib\site-packages\tensorflow\python\framework\constant_op.
↪py in _constant_eager_impl(ctx, value, dtype, shape, verify_shape)
        322   raise TypeError("Eager execution of tf.constant with unsupported␣
↪shape "
        323                   "(value has %d elements, shape is %s with %d␣
↪elements)." %
    --> 324                   (num_t, shape, shape.num_elements()))
        325
        326


        TypeError: Eager execution of tf.constant with unsupported shape (value␣
↪has 24 elements, shape is (10, 2) with 20 elements).
```

Thus, we have to take note of the shape and ensure the dimensions tally with the original dimensions.

7 Getting more Information from Tensors

Attribute	Meaning	Code
Shape	The length (number of elements) of each of the dimensions of a tensor.	`tensor.shape`
Rank	The number of tensor dimensions. A scalar has rank 0, a vector has rank 1, a matrix is rank 2, a tensor has rank n.	`tensor.ndim`
Axis or dimension	A particular dimension of a tensor.	`tensor[0], tensor[:, 1]…`
Size	The total number of items in the tensor.	`tf.size(tensor)`

[28]:
```
#Creating a rank-4 tensor
rank_4_tensor= tf.zeros(shape=(2,3,4,5))
rank_4_tensor
```

[28]:
```
<tf.Tensor: shape=(2, 3, 4, 5), dtype=float32, numpy=
array([[[[0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.]],

        [[0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.]],

        [[0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.]]],


       [[[0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.]],

        [[0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.]],

        [[0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.]]]], dtype=float32)>
```

[29]:
```
#Verifying the rank of the above tensor
rank_4_tensor.ndim
```

[29]:
```
4
```

[30]:
```
rank_4_tensor[0] #oth axis
```

[30]:
```
<tf.Tensor: shape=(3, 4, 5), dtype=float32, numpy=
array([[[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
```

```
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]]], dtype=float32)>
```

[31]:
```
tf.size(rank_4_tensor) #120 elements present i.e. 2x3x4x5
```

[31]:
```
<tf.Tensor: shape=(), dtype=int32, numpy=120>
```

[32]:
```
# Get various attributes of tensor
print("Datatype of every element:", rank_4_tensor.dtype)
print("Number of dimensions (rank):", rank_4_tensor.ndim)
print("Shape of tensor:", rank_4_tensor.shape)
print("Elements along axis 0 of tensor:", rank_4_tensor.shape[0])
print("Elements along last axis of tensor:", rank_4_tensor.shape[-1])
print("Total number of elements (2*3*4*5):", tf.size(rank_4_tensor).numpy()) # .
 ↪numpy() converts to NumPy array
```

```
Datatype of every element: <dtype: 'float32'>
Number of dimensions (rank): 4
Shape of tensor: (2, 3, 4, 5)
Elements along axis 0 of tensor: 2
Elements along last axis of tensor: 5
Total number of elements (2*3*4*5): 120
```

7.1 Summary of attributes from Tensors:

1. Data type
2. Number of dimension or Rank
3. Shape
4. Number of elements

8 Indexing Tensors

Tensors can be indexed like Python lists

[33]:
```
#Get the first two elements of each dimension of the rank 4 tensor above
rank_4_tensor
```

[33]:
```
<tf.Tensor: shape=(2, 3, 4, 5), dtype=float32, numpy=
array([[[[0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.]],
```

```
       [[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]]],


      [[[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]]]], dtype=float32)>
```

[34]:
```
rank_4_tensor[:2,:2,:2,:2]
```

[34]:
```
<tf.Tensor: shape=(2, 2, 2, 2), dtype=float32, numpy=
array([[[[0., 0.],
         [0., 0.]],

        [[0., 0.],
         [0., 0.]]],


       [[[0., 0.],
         [0., 0.]],

        [[0., 0.],
         [0., 0.]]]], dtype=float32)>
```

[35]:
```
#create a Rank2 tensor
rank_2_tensor = tf.constant([[10,1],
                             [7,2]])
rank_2_tensor
```

[35]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[10,  1],
       [ 7,  2]])>
```

[36]:
```
#Get last item of each of our row of rank2 tensor
rank_2_tensor[:,-1].numpy()
```

[36]:
```
array([1, 2])
```

[37]:
```
#Add in extra dimenion to our rank2 tensor
rank_3_tensor = rank_2_tensor[...,tf.newaxis]
rank_3_tensor
```

[37]:
```
<tf.Tensor: shape=(2, 2, 1), dtype=int32, numpy=
array([[[10],
        [ 1]],

       [[ 7],
        [ 2]]])>
```

Note 5: rank_2_tensor[...,tf.newaxis] is same as rank_2_tensor[:,:,tf.newaxis]

[38]:
```
#Alternatice to tf.newaxis
tf.expand_dims(rank_2_tensor,axis=-1) #"-1" means expand final axis
```

[38]:
```
<tf.Tensor: shape=(2, 2, 1), dtype=int32, numpy=
array([[[10],
        [ 1]],

       [[ 7],
        [ 2]]])>
```

[39]:
```
tf.expand_dims(rank_2_tensor,axis=0) #Extra dimension in the front
```

[39]:
```
<tf.Tensor: shape=(1, 2, 2), dtype=int32, numpy=
array([[[10,  1],
        [ 7,  2]]])>
```

9 Tensor operations

[40]:
```
# Addition operator
tensor=tf.constant([[10,7],
                    [3,4]])
tensor+10
```

[40]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[20, 17],
       [13, 14]])>
```

[41]:
```
#Multiplication
tensor*10
```

[41]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[100,  70],
       [ 30,  40]])>
```

[42]:
```
#subtraction
tensor-10
```

[42]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[ 0, -3],
       [-7, -6]])>
```

[43]:
```
#Division
tensor/10
```

[43]:
```
<tf.Tensor: shape=(2, 2), dtype=float64, numpy=
array([[1. , 0.7],
       [0.3, 0.4]])>
```

[44]:
```
#Using the tensorflow builtin functions
tf.multiply(tensor,10)
```

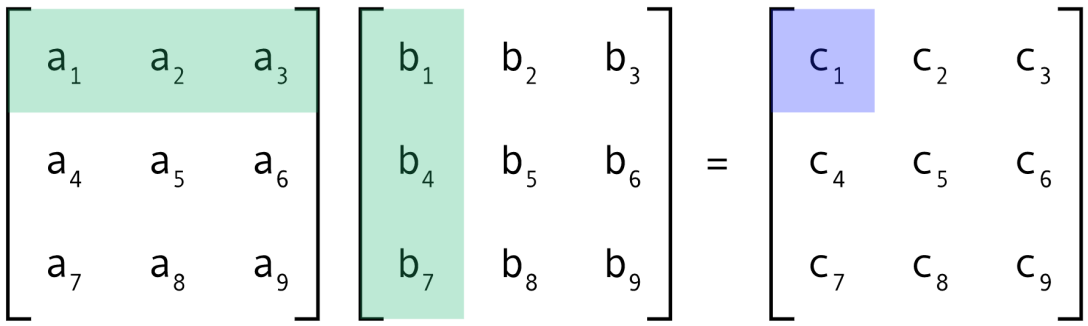
[44]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[100,  70],
       [ 30,  40]])>
```

[45]:
```
tf.add(tensor,10)
```

[45]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[20, 17],
       [13, 14]])>
```

The above will take advatage of the gpu to speed up the computation

10 Matrix Multiplication using tf.linalg.matmul



Note 5: The main two rules for matrix multiplication to remember are: 1.The inner dimensions must match: 2.The resulting matrix has the shape of the outer dimensions

Visualization of matrix : http://matrixmultiplication.xyz/

[46]:
```
#Matrix multiplication in tensorflow
print(tensor)
#In tensorflow we can drop the intermediate areas i.e. instead of using tf.
 ↪linalg.matmul we can use tf.matmul
tf.matmul(tensor,tensor)
```

```
tf.Tensor(
[[10  7]
 [ 3  4]], shape=(2, 2), dtype=int32)
```

[46]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[121,  98],
       [ 42,  37]])>
```

The above shows multiplication between two tensors

[47]:
```
tensor*tensor
```

[47]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[100,  49],
       [  9,  16]])>
```

The above does element wise multiplication between the corresponding elements

[48]:
```
#To do matrix multiplication with python operator use @
tensor@tensor
```

[48]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[121,  98],
       [ 42,  37]])>
```

[49]:
```
X = tf.constant([[1,2],
                 [3,4],
                  [5,6]])
Y = tf.constant([[7,8],
                 [9,10],
                  [11,12]])
```

[50]:
```
tf.matmul(X,Y)
```

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

       InvalidArgumentError                      Traceback (most recent call␣
↪last)

       <ipython-input-50-cc348e6d8216> in <module>
   ----> 1 tf.matmul(X,Y)


       D:\Anaconda\envs\fyp\lib\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


       D:\Anaconda\envs\fyp\lib\site-packages\tensorflow\python\ops\math_ops.py␣
↪in matmul(a, b, transpose_a, transpose_b, adjoint_a, adjoint_b, a_is_sparse,␣
↪b_is_sparse, name)
      3253     else:
      3254       return gen_math_ops.mat_mul(
   -> 3255           a, b, transpose_a=transpose_a, transpose_b=transpose_b,␣
↪name=name)
      3256
      3257


       D:
↪\Anaconda\envs\fyp\lib\site-packages\tensorflow\python\ops\gen_math_ops.py in␣
↪mat_mul(a, b, transpose_a, transpose_b, name)
      5622       return _result
      5623     except _core._NotOkStatusException as e:
```

```
   -> 5624       _ops.raise_from_not_ok_status(e, name)
      5625     except _core._FallbackException:
      5626       pass


      D:\Anaconda\envs\fyp\lib\site-packages\tensorflow\python\framework\ops.
↪py in raise_from_not_ok_status(e, name)
      6841   message = e.message + (" name: " + name if name is not None else␣
↪"")
      6842   # pylint: disable=protected-access
   -> 6843   six.raise_from(core._status_to_exception(e.code, message), None)
      6844   # pylint: enable=protected-access
      6845


      D:\Anaconda\envs\fyp\lib\site-packages\six.py in raise_from(value,␣
↪from_value)


      InvalidArgumentError: Matrix size-incompatible: In[0]: [3,2], In[1]:␣
↪[3,2] [Op:MatMul]
```

Thus, we need to reshape one of the matrix to perform the multiplication

[ ]:
```
#reshaping matrix Y
Y = tf.reshape(Y,shape=(2,3))
Y
```

[ ]:
```
tf.matmul(X,Y)
```

Transpose is when the cols become rows and the rows become cols

[51]:
```
X = tf.constant([[1,2],
                 [3,4],
                  [5,6]])
Y = tf.constant([[7,8],
                 [9,10],
                  [11,12]])
```

[52]:
```
tf.matmul(X,tf.transpose(Y))
```

[52]:
```
<tf.Tensor: shape=(3, 3), dtype=int32, numpy=
array([[ 23,  29,  35],
       [ 53,  67,  81],
       [ 83, 105, 127]])>
```

[53]:
```
#Seeing difference between transpose and reshape
Y,tf.reshape(Y,shape=(2,3)) , tf.transpose(Y)
```

[53]:
```
(<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
 array([[ 7,  8],
        [ 9, 10],
        [11, 12]])>,
 <tf.Tensor: shape=(2, 3), dtype=int32, numpy=
 array([[ 7,  8,  9],
        [10, 11, 12]])>,
 <tf.Tensor: shape=(2, 3), dtype=int32, numpy=
 array([[ 7,  9, 11],
        [ 8, 10, 12]])>)
```

Thus, transposing is shifting the axises while reshaping reshuffles the elements in the matrix

10.1 The dot product

Multiplying matrices by eachother is also referred to as the dot product.

You can perform the `tf.matmul()` operation using `tf.tensordot()`

[54]:
```
X,Y
```

[54]:
```
(<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
 array([[1, 2],
        [3, 4],
        [5, 6]])>,
 <tf.Tensor: shape=(3, 2), dtype=int32, numpy=
 array([[ 7,  8],
        [ 9, 10],
        [11, 12]])>)
```

[55]:
```
#perform the dot product
tf.tensordot(tf.transpose(X),Y,axes=1)
```

[55]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[ 89,  98],
       [116, 128]])>
```

11 Changing datatype of a tensor

[56]:
```
#Create a new tensor with default datatype (float32)
B = tf.constant([1.7,2])
B
```

[56]:
```
<tf.Tensor: shape=(2,), dtype=float32, numpy=array([1.7, 2. ], dtype=float32)>
```

[57]:
```
#Change float 32 to float 16
B = tf.cast(B,dtype=tf.float16)
B
```

[57]:
```
<tf.Tensor: shape=(2,), dtype=float16, numpy=array([1.7, 2. ], dtype=float16)>
```

12 Aggregating Tensors

12.0.1 Finding the min, max, mean, sum (aggregation)

You can quickly aggregate (perform a calculation on a whole tensor) tensors to find things like the minimum value, maximum value, mean and sum of all the elements.

To do so, aggregation methods typically have the syntax `reduce()_[action]`, such as: * `tf.reduce_min()` - find the minimum value in a tensor. * `tf.reduce_max()` - find the maximum value in a tensor (helpful for when you want to find the highest prediction probability). * `tf.reduce_mean()` - find the mean of all elements in a tensor. * `tf.reduce_sum()` - find the sum of all elements in a tensor. * **Note:** typically, each of these is under the `math` module, e.g. `tf.math.reduce_min()` but you can use the alias `tf.reduce_min()`.

[58]:
```
# Get the absolute values of Tensor
D = tf.constant([-7,-10])
tf.abs(D)
```

[58]:
```
<tf.Tensor: shape=(2,), dtype=int32, numpy=array([ 7, 10])>
```

[60]:
```
#Create a random tensor
E = tf.constant(np.random.randint(0,100,size=50))
E
```

[60]:
```
<tf.Tensor: shape=(50,), dtype=int32, numpy=
array([20,  6, 76,  5, 30, 27, 87, 72, 40, 91, 82, 28,  1, 15, 57, 93, 45,
       85, 21, 58,  7, 38, 79, 86, 32, 30,  6, 79,  0, 31, 42, 24, 60, 68,
       58, 27, 45, 30, 64, 95, 20, 31, 62, 78, 66, 91, 36, 81, 43, 50])>
```

[71]:
```
tf.size(E).numpy(), E.shape, E.ndim
```

[71]:
```
(50, TensorShape([50]), 1)
```

[67]:
```
#Find the minimum
tf.reduce_min(E).numpy()
```

[67]:
```
0
```

[72]:
```
#Find the maximum
tf.reduce_max(E).numpy()
```

[72]:
```
95
```

[75]:
```
#Find the mean
tf.reduce_mean(E).numpy()
```

[75]:
```
47
```

[76]:
```
#Find the sum
tf.reduce_sum(E).numpy()
```

[76]:
```
2398
```

You can also find the standard deviation (`tf.math.reduce_std()`) and variance (`tf.math.reduce_variance()`) of elements in a tensor using similar methods.

[83]:
```
tf.math.reduce_std(tf.cast(E,dtype=tf.float32)).numpy()
```

[83]:
```
28.037802
```

[84]:
```
tf.math.reduce_variance(tf.cast(E,dtype=tf.float32)).numpy()
```

[84]:
```
786.11835
```

13 Find the positional maximum and minimum

How about finding the position a tensor where the maximum value occurs?

This is helpful when you want to line up your labels (say `['Green', 'Blue', 'Red']`) with your prediction probabilities tensor (e.g. `[0.98, 0.01, 0.01]`).

In this case, the predicted label (the one with the highest prediction probability) would be `'Green'`.

You can do the same for the minimum (if required) with the following: * `tf.argmax()` - find the position of the maximum element in a given tensor. * `tf.argmin()` - find the position of the minimum element in a given tensor.

[90]:
```
#Find the positional max and min for a random tensor
tf.random.set_seed(42)
F = tf.random.uniform(shape=[50])
F
```

[90]:
```
<tf.Tensor: shape=(50,), dtype=float32, numpy=
array([0.6645621 , 0.44100678, 0.3528825 , 0.46448255, 0.03366041,
       0.68467236, 0.74011743, 0.8724445 , 0.22632635, 0.22319686,
       0.3103881 , 0.7223358 , 0.13318717, 0.5480639 , 0.5746088 ,
       0.8996835 , 0.00946367, 0.5212307 , 0.6345445 , 0.1993283 ,
       0.72942245, 0.54583454, 0.10756552, 0.6767061 , 0.6602763 ,
       0.33695042, 0.60141766, 0.21062577, 0.8527372 , 0.44062173,
       0.9485276 , 0.23752594, 0.81179297, 0.5263394 , 0.494308  ,
       0.21612847, 0.8457197 , 0.8718841 , 0.3083862 , 0.6868038 ,
       0.23764038, 0.7817228 , 0.9671384 , 0.06870162, 0.79873943,
       0.66028714, 0.5871513 , 0.16461694, 0.7381023 , 0.32054043],
```

```
       dtype=float32)>
```

[91]:
```
tf.argmax(F).numpy()
```

[91]:
```
42
```

[93]:
```
F[42]
```

[93]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=0.9671384>
```

[97]:
```
tf.argmin(F).numpy()
```

[97]:
```
16
```

[98]:
```
F[16]
```

[98]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=0.009463668>
```

14 Squeezin a Tensor i.e. removing all single dimensions

[99]:
```
tf.random.set_seed(42)
G = tf.constant(np.random.randint(0, 100, 50), shape=(1, 1, 1, 1, 50))
G.shape, G.ndim
```

[99]:
```
(TensorShape([1, 1, 1, 1, 50]), 5)
```

[101]:
```
G
```

[101]:
```
<tf.Tensor: shape=(1, 1, 1, 1, 50), dtype=int32, numpy=
array([[[[[46, 64, 14, 86, 48, 69, 48, 15, 63,  0, 20, 24, 27, 31, 12,
           41, 84, 57, 69, 89, 18, 63, 63, 27, 79, 75, 80, 35,  7, 85,
           49, 57, 87, 49, 32, 46,  4, 36, 41, 63, 99, 75, 29, 69, 82,
           36, 60, 63, 55, 20]]]]])>
```

[100]:
```
tf.squeeze(G)
```

[100]:
```
<tf.Tensor: shape=(50,), dtype=int32, numpy=
array([46, 64, 14, 86, 48, 69, 48, 15, 63,  0, 20, 24, 27, 31, 12, 41, 84,
       57, 69, 89, 18, 63, 63, 27, 79, 75, 80, 35,  7, 85, 49, 57, 87, 49,
       32, 46,  4, 36, 41, 63, 99, 75, 29, 69, 82, 36, 60, 63, 55, 20])>
```

Squeeze removes all those 1 dimensions

15 One hot Encoding Tensors

[104]:
```
list_colors = [0,1,2,3,0,2,3,4]
tf.one_hot(list_colors,depth=5)
```

[104]:
```
<tf.Tensor: shape=(8, 5), dtype=float32, numpy=
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]], dtype=float32)>
```

16 Other mathematical operations

[105]:
```
H = tf.range(1,10)
```

[106]:
```
H
```

[106]:
```
<tf.Tensor: shape=(9,), dtype=int32, numpy=array([1, 2, 3, 4, 5, 6, 7, 8, 9])>
```

[108]:
```
#Squaring the tensor
tf.square(H)
```

[108]:
```
<tf.Tensor: shape=(9,), dtype=int32, numpy=array([ 1,  4,  9, 16, 25, 36, 49,
64, 81])>
```

[112]:
```
#Find the square root
tf.sqrt(tf.cast(H,dtype=tf.float32))
```

[112]:
```
<tf.Tensor: shape=(9,), dtype=float32, numpy=
array([0.99999994, 1.4142134 , 1.7320508 , 1.9999999 , 2.236068  ,
       2.4494896 , 2.6457512 , 2.8284268 , 3.        ], dtype=float32)>
```

17 Tensors and NumPy

We've seen some examples of tensors interact with NumPy arrays, such as, using NumPy arrays to create tensors.

Tensors can also be converted to NumPy arrays using:

- `np.array()` - pass a tensor to convert to an ndarray (NumPy's main datatype).
- `tensor.numpy()` - call on a tensor to convert to an ndarray.

Doing this is helpful as it makes tensors iterable as well as allows us to use any of NumPy's methods on them.

[113]:
```
#Create a tensor directly from an numpy array
H = tf.constant(np.array([1,2,3,4,5]))
H
```

[113]:
```
<tf.Tensor: shape=(5,), dtype=int32, numpy=array([1, 2, 3, 4, 5])>
```

[115]:
```
#Convery back to numpy array
np.array(H) , type(np.array(H))
```

[115]:
```
(array([1, 2, 3, 4, 5]), numpy.ndarray)
```

[116]:
```
#Another method
H.numpy(), type(H.numpy())
```

[116]:
```
(array([1, 2, 3, 4, 5]), numpy.ndarray)
```

[ ]:
```

```