In this notebook we are going to cover some of the most fundamental concepts of tensor using TensorFlow

More specifically we are going to cover:

- Introduction to tensors
- Getting information from tensors
- Manpulating tensors
- Tensors and Numpy
- Using @tf.function (a way to speed up regular Python functions)
- Using GPUs and TPUs

Introduction to Tensors

```
# Import TensorFlow
import tensorflow as tf
print(tf.__version__)
```

```
2.5.0
```

```
# Create tensors with tf.constant()
scalar = tf.constant(7)
scalar
```

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

```
# Check the number of dimensions of a tensor
scalar.ndim
```

```
0
```

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

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

```
# Check the dimension of our vector
vector.ndim
```

```
1
```

```
# Create a matrix
matrix = tf.constant([[10,7],
                      [7, 20]])
matrix
```

```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[10,  7],
       [ 7, 20]], dtype=int32)>
```

```
matrix.ndim
```

```
2
```

```
# Create another matrix
another_matrix = tf.constant([[10., 7.],
                              [1.,2.],
                              [3.,8.]], dtype=tf.float16)
another_matrix
```

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

```
# Checking dimension of another_matrix
another_matrix.ndim
```

```
2
```

```
# Let's 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
```

```
<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]]], dtype=int32)>
```

```
tensor.ndim
```

```
3
```

What we have created so far:

- Scalar: A single number
- Vector: A number with direction
- Matrix: A 2-dimensional array of numbers
- Tensor: A n-dimensional array of numbers

Creating Tensors with `tf.Variable`

```
# Creating tensor with tf.Variable
changeable_tensor = tf.Variable([10,7])
unchangeable_tensor = tf.constant([10,7])
changeable_tensor, unchangeable_tensor
```

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

```
changeable_tensor[0] = 7
changeable_tensor
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-134-05d0bdd98eb4> in <module>()
----> 1 changeable_tensor[0] = 7
      2 changeable_tensor

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

SEARCH STACK OVERFLOW

```
# That's not working! Let's try with assign()
changeable_tensor[0].assign(7)
changeable_tensor
```

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

```
# Let's try changing unchangeable tensor
unchangeable_tensor[0].assign(7)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-136-beaf8da8bd1e> in <module>()
      1 # Let's try changing unchangeable tensor
----> 2 unchangeable_tensor[0].assign(7)

/usr/local/lib/python3.7/dist-packages/tensorflow/python/framework/ops.py in __getattr__(self,
name)
    399         import tensorflow.python.ops.numpy_ops.np_config
    400         np_config.enable_numpy_behavior()""".format(type(self).__name__, name))
--> 401     self.__getattribute__(name)
    402 
    403   @staticmethod

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

SEARCH STACK OVERFLOW

Creating random tensors

Random tensors are some arbitrary tensors of random numbers

```
# Create random tensors
```

```
random_1 = tf.random.Generator.from_seed(42)
random_1 = random_1.normal(shape=(3,2))
random_2 = tf.random.Generator.from_seed(42)
random_2 = random_2.normal(shape=(3,2))

#Checking equality
random_1, random_2, random_1 == random_2
```

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

Shuffle order of elements in a tensor

```
# Shuffle a tensor so that the inherent order changes
not_shuffled = tf.constant([[10, 7],
                            [3, 4],
                            [2, 5]])

# Let's shuffle our data
tf.random.shuffle(not_shuffled)
```

```
<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
array([[ 2,  5],
       [ 3,  4],
       [10,  7]], dtype=int32)>
```

It looks like if we want our shuffled tensors to be in the same order, we need to use both the global level random seed as well as the operation level random seed.

```
tf.random.set_seed(42)
tf.random.shuffle(not_shuffled, seed=42)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-139-8b5d5def4d01> in <module>()
----> 1 tf.random.set_seed(42)
      2 tf.random.shuffle(not_shuffled, seed=42)

TypeError: 'int' object is not callable
```

SEARCH STACK OVERFLOW

Other ways to create tensors

```
# Create a tensor of all ones
```

```
tf.ones(shape=(10,7))
```

```
<tf.Tensor: shape=(10, 7), dtype=float32, 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., 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., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.]], dtype=float32)>
```

```
# Create a tensor of all zeros
tf.zeros(shape=(3,4))
```

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

Turn NumPy arrays into TensorFlow tensors

The main difference between numpy arrays and tensorflow tensors are that the tensors can be run on GPU (much faster)

```
# Importing and creating a numpy array
import numpy as np
num_A = np.arange(1, 25, dtype=np.int32)
num_A
```

```
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], dtype=int32)
```

```
A = tf.constant(num_A, shape=(2,3,4))
B = tf.constant(num_A)
A, B
```

```
(<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]]], dtype=int32)>,
 <tf.Tensor: shape=(24,), 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], dtype=int32)>)
```

```
A.ndim
```

```
3
```

Getting information from tensors

When dealing with tensors, we might want to get the following attributes:

- Shape
- Rank
- Axis or Dimension
- Size

```
# Create a rank 4 tensor
rank_4_tensor = tf.zeros(shape=(2,3,4,5))
rank_4_tensor
```

```
<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)>
```

```
rank_4_tensor[0].ndim
```

```
3
```

```
rank_4_tensor.shape, rank_4_tensor.ndim, tf.size(rank_4_tensor)
```

```
(TensorShape([2, 3, 4, 5]), 4, <tf.Tensor: shape=(), dtype=int32, numpy=120>)
```

```
# Get various attributes of tensors
print("Datatype of every element: ", rank_4_tensor.dtype )
print("Dimension of tensor(Rank): ", rank_4_tensor.ndim)
print("Shape of tensor: ", rank_4_tensor.shape)
print("Elements along zero axis: ", rank_4_tensor.shape[0])
print("Elements along last axis: ", rank_4_tensor.shape[-1])
print("Total number of elements in our tensor:", tf.size(rank_4_tensor))
```

```
Datatype of every element:  <dtype: 'float32'>
Dimension of tensor(Rank):  4
Shape of tensor:  (2, 3, 4, 5)
Elements along zero axis:  2
Elements along last axis:  5
Total number of elements in our tensor: tf.Tensor(120, shape=(), dtype=int32)
```

Indexing tensors

Indexing tensors is similar to indexing python list

```
# Get the first two elements of each of the dimension

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

```
<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)>
```

```
# Get the first element from each of the dimension except the final one
rank_4_tensor[:1, :1, :1, :]
```

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

```
# Create a rank 2 tensor (2 dimension)
rank_2_tensor = tf.constant([[10, 7],
                             [3, 4]])
rank_2_tensor.shape, rank_2_tensor.ndim
```

```
(TensorShape([2, 2]), 2)
```

```
# Get the last item of each dimension
rank_2_tensor[:,-1]
```

```
<tf.Tensor: shape=(2,), dtype=int32, numpy=array([7, 4], dtype=int32)>
```

```
rank_3_tensor = rank_2_tensor[..., tf.newaxis]
rank_3_tensor
```

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

       [[ 3],
        [ 4]]], dtype=int32)>
```

```
# Alternative to tf.newaxis
tf.expand_dims(rank_2_tensor, axis=1)
```

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

       [[ 3,  4]]], dtype=int32)>
```

```
tf.expand_dims(rank_4_tensor, axis=0)
```

```
<tf.Tensor: shape=(1, 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)>
```

Manipulating tensors (tensor operations)

Basic Operations

```
# You can add values to a tensor with the addition operator
tensor = tf.constant([[10, 7], [3, 4]])
tensor + 10
```

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

```
# Multiplication
tensor * 10
```

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

```
# We can also use the built in tensorflow function
tf.multiply(tensor, 10)
```

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

```
tf.add(tensor, 100)
```

```
     <tf.Tensor: shape=(2, 2), dtype=int32, numpy=
     array([[110, 107],
            [103, 104]], dtype=int32)>
```

Matrix Multiplication

In machine learning, matrix multiplication is one of the most important operations

```
# Matrix multiplication in tensorflow
tf.matmul(tensor, tensor)
```

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

```
tensor_1 = tf.constant([[1, 2, 5], [7, 2, 1], [3, 3, 3]])
tensor_2 = tf.constant([[3, 5], [6, 7], [1, 8]])

tf.matmul(tensor_1, tensor_2)
```

```
     <tf.Tensor: shape=(3, 2), dtype=int32, numpy=
     array([[20, 59],
            [34, 57],
            [30, 60]], dtype=int32)>
```

```
# Matrix multiplication with python @ operator
tensor_1 @ tensor_2
```

```
     <tf.Tensor: shape=(3, 2), dtype=int32, numpy=
     array([[20, 59],
            [34, 57],
            [30, 60]], dtype=int32)>
```

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

```
tf.matmul(X, Y)
```

```
---------------------------------------------------------------------------
InvalidArgumentError                      Traceback (most recent call last)
<ipython-input-164-b58f5d491930> in <module>()
----> 1 tf.matmul(X, Y)

4 frames
/usr/local/lib/python3.7/dist-packages/six.py in raise_from(value, from_value)

InvalidArgumentError: In[0] mismatch In[1] shape: 2 vs. 3: [3,2] [3,2] 0 0 [Op:MatMul]
```

```
# Let's reshape Y
tf.reshape(Y, shape=(2, 3))
```

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

```
tf.matmul(X, tf.reshape(Y, shape=(2, 3)))
```

```
<tf.Tensor: shape=(3, 3), dtype=int32, numpy=
array([[ 27,  30,  33],
       [ 61,  68,  75],
       [ 95, 106, 117]], dtype=int32)>
```

```
# Now let's try changing the shape of X
tf.matmul(tf.reshape(X, shape=(2, 3)), Y)
```

```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[ 58,  64],
       [139, 154]], dtype=int32)>
```

The Dot Product

Matrix multiplication is also referred to as dot product. We can perform matrix multiplication using:

- `tf.matmul()`
- `tf.tensordot()`

```
X, Y
```

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

```
# For dot product X or Y needs to be transposed
tf.tensordot(tf.transpose(X), Y, axes=1)
```

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

Changing the data type of a tensor

```
B = tf.constant([1.4, 2.6])
B.dtype
```

```
tf.float32
```

```
C = tf.constant([1, 2])
C.dtype
```

```
tf.int32
```

```
# Change from float 32 to float 16 (reduced precision)
B = tf.cast(B, dtype=tf.float16)
B
```

```
<tf.Tensor: shape=(2,), dtype=float16, numpy=array([1.4, 2.6], dtype=float16)>
```

```
E = tf.cast(C, dtype=tf.float32)
E
```

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

Aggregating tensors

```
# Getting absolute values from tensors
D = tf.constant([-10, -7])
tf.abs(D)
```

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

```
# The minimum of a tensor
A = tf.constant([[1,2,3],
                 [4,-5,6],
                 [7,8,-9]])

tf.reduce_min(A, axis=1)
```

```
<tf.Tensor: shape=(3,), dtype=int32, numpy=array([ 1, -5, -9], dtype=int32)>
```

```
E = tf.constant(np.random.randint(0, 100, 50))
E
```

```
<tf.Tensor: shape=(50,), dtype=int64, numpy=
array([97, 59, 36, 16,  8, 69, 30, 26, 24, 21, 32, 89, 95,  4, 60, 56, 63,
       41, 68, 22, 15,  6,  5, 46, 55, 36, 75, 59, 73, 59, 82, 17,  3, 43,
       90, 40, 74, 89, 47, 10, 33,  5, 13, 30, 19, 70, 27, 49, 98, 11])>
```

```
tf.size(E), E.shape, E.ndim
```

```
(<tf.Tensor: shape=(), dtype=int32, numpy=50>, TensorShape([50]), 1)
```

```
# Find the minimum
tf.reduce_min(E)
```

```
<tf.Tensor: shape=(), dtype=int64, numpy=3>
```

```
# Find the maximum
tf.reduce_max(E)
```

```
<tf.Tensor: shape=(), dtype=int64, numpy=98>
```

```
# Find the mean
tf.reduce_mean(E)
```

```
<tf.Tensor: shape=(), dtype=int64, numpy=43>
```

```
# Find the sum
tf.reduce_sum(E)
```

```
<tf.Tensor: shape=(), dtype=int64, numpy=2195>
```

```
# Find the standard deviation
tf.math.reduce_std(tf.cast(E, dtype=tf.float32))
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=28.352955>
```

```
# Find the variance
tf.math.reduce_variance(tf.cast(E, dtype=tf.float32))
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=803.89>
```

```
# Creating a new tensor to find positional minimum and maximum
tf.random.set_seed=42
F = tf.random.uniform(shape=[50])
F
```

```
<tf.Tensor: shape=(50,), dtype=float32, numpy=
array([0.7402308 , 0.33938193, 0.5692506 , 0.44811392, 0.29285502,
       0.4260056 , 0.62890387, 0.691061  , 0.30925727, 0.89236605,
       0.66396606, 0.30541587, 0.8724164 , 0.1025728 , 0.56819403,
       0.25427842, 0.7253866 , 0.4770788 , 0.46289814, 0.88944995,
       0.6792555 , 0.09752727, 0.01609659, 0.4876021 , 0.5832968 ,
       0.41212583, 0.731905  , 0.93418944, 0.5298122 , 0.9664817 ,
       0.88391197, 0.10578597, 0.44439578, 0.7851516 , 0.47332513,
       0.89893615, 0.04290593, 0.8717004 , 0.6068529 , 0.12963045,
       0.4527359 , 0.24573493, 0.34777248, 0.582147  , 0.82298195,
       0.82862926, 0.877372  , 0.5319803 , 0.03594303, 0.03986669],
      dtype=float32)>
```

```
# Find the positional maximum
tf.argmax(F)
```

```
<tf.Tensor: shape=(), dtype=int64, numpy=29>
```

```
# Indexing the largest value
F[tf.argmax(F)]
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.9664817>
```

```
tf.math.reduce_max(F)
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.9664817>
```

```
# FInd positional minimum
tf.argmin(F)
```

```
<tf.Tensor: shape=(), dtype=int64, numpy=22>
```

```
#Find the minimum value
F[tf.argmin(F)]
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.016096592>
```

Squeezing the tensor

```
tf.random.set_seed=42
G = tf.constant(tf.random.uniform(shape=[50]), shape=(1,1,1,1,50))
G
```

```
<tf.Tensor: shape=(1, 1, 1, 1, 50), dtype=float32, numpy=
array([[[[[0.803156  , 0.49777734, 0.37054038, 0.9118674 , 0.637642  ,
           0.18209696, 0.63791955, 0.27701473, 0.04227114, 0.84219384,
           0.90637195, 0.222556  , 0.9198462 , 0.68789077, 0.42705178,
           0.878158  , 0.6943959 , 0.46567595, 0.52925766, 0.33019018,
           0.12754858, 0.16153514, 0.5085137 , 0.44301772, 0.35205877,
           0.8969147 , 0.24940813, 0.76328313, 0.85935795, 0.08480155,
           0.20418596, 0.28848922, 0.65142167, 0.7106751 , 0.8695041 ,
           0.23745108, 0.6688912 , 0.7115667 , 0.21899498, 0.7702793 ,
           0.45055628, 0.95493364, 0.71695936, 0.98945487, 0.1511141 ,
           0.06240606, 0.15209746, 0.99522185, 0.7830266 , 0.10455871]]]]],
      dtype=float32)>
```

```
# Squeezing the tensor (removing one dimensional shapes)
G_squeezed = tf.squeeze(G)
G_squeezed, G_squeezed.shape
```

```
(<tf.Tensor: shape=(50,), dtype=float32, numpy=
 array([0.803156  , 0.49777734, 0.37054038, 0.9118674 , 0.637642  ,
        0.18209696, 0.63791955, 0.27701473, 0.04227114, 0.84219384,
        0.90637195, 0.222556  , 0.9198462 , 0.68789077, 0.42705178,
        0.878158  , 0.6943959 , 0.46567595, 0.52925766, 0.33019018,
        0.12754858, 0.16153514, 0.5085137 , 0.44301772, 0.35205877,
        0.8969147 , 0.24940813, 0.76328313, 0.85935795, 0.08480155,
        0.20418596, 0.28848922, 0.65142167, 0.7106751 , 0.8695041 ,
        0.23745108, 0.6688912 , 0.7115667 , 0.21899498, 0.7702793 ,
        0.45055628, 0.95493364, 0.71695936, 0.98945487, 0.1511141 ,
        0.06240606, 0.15209746, 0.99522185, 0.7830266 , 0.10455871],
       dtype=float32)>, TensorShape([50]))
```

One-hot encoding tensors

```
# Creating a random list
some_list = [0, 1, 2, 3]
```

```
# One-hot encode our list
tf.one_hot(some_list, 4)
```

```
<tf.Tensor: shape=(4, 4), dtype=float32, numpy=
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]], dtype=float32)>
```

```
# One-hot encoding with custom values
tf.one_hot(some_list, depth=4, on_value="Wreet", off_value="**")
```

```
<tf.Tensor: shape=(4, 4), dtype=string, numpy=
array([[b'Wreet', b'**', b'**', b'**'],
       [b'**', b'Wreet', b'**', b'**'],
       [b'**', b'**', b'Wreet', b'**'],
       [b'**', b'**', b'**', b'Wreet']], dtype=object)>
```

Square, log and square root

```
# Squaring values in a tensor
H = tf.range(1,10)
tf.math.square(H)
```

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

```
# Getting the square root
tf.math.sqrt(tf.cast(H, dtype=tf.float64))
```

```
<tf.Tensor: shape=(9,), dtype=float64, numpy=
array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798,
       2.44948974, 2.64575131, 2.82842712, 3.        ])>
```

```
# Find log
tf.math.log(tf.cast(H, dtype=tf.float64))
```

```
<tf.Tensor: shape=(9,), dtype=float64, numpy=
array([0.        , 0.69314718, 1.09861229, 1.38629436, 1.60943791,
       1.79175947, 1.94591015, 2.07944154, 2.19722458])>
```

Finding access to GPU

```
import tensorflow as tf
tf.config.list_physical_devices()
```

```
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
 PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
!nvidia-smi
```

```
Wed Aug 11 11:05:57 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.42.01    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   44C    P8     9W /  70W |      3MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

✓ 0s completed at 5:05 PM