Training on the most fundamental concepts of tensors using TensorFlow.

More specifically, I will work on :

```
Introduction to tensors
Getting informations from tensors
Manipulating tensors
Tensors & Numpy
Using @tf.function (a way to speed up the regular Python functions)
Using GPUs with TensorFlow (or CPUs)
```

Introduction to Tensors

Entrée [1]:

```python
# Import TensorFlow
import tensorflow as tf

print(tf.__version__)
```

```
2.10.0
```

Entrée [2]:

```python
import numpy as np
import tensorflow_probability as tfp
```

Entrée [3]:

```python
# Creating tensors with tf.constant()

scalar = tf.constant(7)
scalar
```

Out[3]:

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

Entrée [4]:

```python
# checking the number of dimensions of a tensor
scalar.ndim
```

Out[4]:

```
0
```

Entrée [5]:

```python
# ------------------------
```

Entrée [6]:

```
# Creating a vector
vector = tf.constant([10,15])
vector
```

Out[6]:

```
<tf.Tensor: shape=(2,), dtype=int32, numpy=array([10, 15])>
```

Entrée [7]:

```
# Checking the dimension of the vector
vector.ndim
```

Out[7]:

```
1
```

Entrée [8]:

```
# --------------------------
```

Entrée [9]:

```
# Creating a matrix (has more than one dimension)
matrix = tf.constant([[5,3],[12,15],[10,5]])
matrix
```

Out[9]:

```
<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
array([[ 5,  3],
       [12, 15],
       [10,  5]])>
```

Entrée [10]:

```
# Checking the dimension of the matrix
matrix.ndim
```

Out[10]:

```
2
```

Entrée [11]:

```
# ----------------------
```

Entrée [12]:

```
# Creating a matrix while specifing its data's type
matrix2 = tf.constant([[1,5.],[15.,18.],[9.,15]], dtype=tf.float16)
matrix2
```

Out[12]:

```
<tf.Tensor: shape=(3, 2), dtype=float16, numpy=
array([[ 1.,  5.],
       [15., 18.],
       [ 9., 15.]], dtype=float16)>
```

Entrée [13]:

```
# Checking the dimensions of the matrix
matrix2.ndim
```

Out[13]:

```
2
```

Entrée [14]:

```
# ------------------------
```

Entrée [15]:

```
# Creating a tensor
tensor = tf.constant([
    [ [1,5,16],[12,20,9] ],
    [ [13,19,12], [24,31,15]],
    [ [9,11,8], [19,8,5] ],
    [ [7,16,3], [98,12,42] ]
],dtype=tf.float32)
tensor
```

Out[15]:

```
<tf.Tensor: shape=(4, 2, 3), dtype=float32, numpy=
array([[[ 1.,  5., 16.],
        [12., 20.,  9.]],

       [[13., 19., 12.],
        [24., 31., 15.]],

       [[ 9., 11.,  8.],
        [19.,  8.,  5.]],

       [[ 7., 16.,  3.],
        [98., 12., 42.]]], dtype=float32)>
```

Entrée [16]:

```
tensor.ndim
```

Out[16]:

```
3
```

Entrée [17]:

```
# --------------------
```

What we have created so far :

```
scalar : a single number
vector : a number with direction (e.g. wind speed and direction)
matrix : a 2-dimensional array of numbers
tensor : a n-dimensional array of numbers
```

Creating Tensor with `tf.Variable`

Entrée [18]:

```
tf.Variable
```

Out[18]:

```
tensorflow.python.ops.variables.Variable
```

Entrée [19]:

```
tf.constant
```

Out[19]:

```
<function tensorflow.python.framework.constant_op.constant(value, dtype=Non
e, shape=None, name='Const')>
```

Entrée [20]:

```
changeable_tensor = tf.Variable([10,7])
unchangeable_tensor = tf.constant([10,7])
changeable_tensor, unchangeable_tensor
```

Out[20]:

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

Entrée [21]:

```
# Getting the elements in the changeable_tensor
changeable_tensor[0], changeable_tensor[1]
```

Out[21]:

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

Entrée [22]:

```
# Getting one of the elements in the changeable_tensor
changeable_tensor[0].assign(23)
changeable_tensor
```

Out[22]:

```
<tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([23,  7])>
```

Entrée [ ]:

```

```

Creating random tensors

Random tensors are tensors of some arbitrary size containing random numbers.

Entrée [23]:

```
#creating two random (but the same) tensors
random_1 = tf.random.Generator.from_seed(42) #set seed for reproducibility
random_1 = random_1.normal(shape=(3,2))
random_1
```

Out[23]:

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

Entrée [ ]:

```

```

Entrée [24]:

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

Out[24]:

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

Entrée [25]:

```
# are random_1 and random_2 equal ?

random_1, random_2, random_1==random_2
```

Out[25]:

```
(<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]])>)
```

Entrée [ ]:

```

```

Shuffling the order of elements in a tensor

> It is valuable for when you want to shuffle the data so the inherent order doesn't affect learning

Entrée [26]:

```
not_shufled = tf.constant([ [10,8], [12,5], [2,5] ])
not_shufled.ndim
```

Out[26]:

```
2
```

Entrée [27]:

```
not_shufled
```

Out[27]:

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

Entrée [28]:

```
# shuffle the non-shuffled tensor
tf.random.shuffle(not_shufled)
```

Out[28]:

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

Entrée [29]:

```
# shuffle the non-shuffled tensor
tf.random.set_seed(42)  #global level random seed
tf.random.shuffle(not_shufled,seed=42) #operation level random seed
```

Out[29]:

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

https://www.tensorflow.org/api_docs/python/tf/random/set_seed (https://www.tensorflow.org/api_docs/python/tf/random/set_seed)

Il looks like if we want our shuffled tensors to be in the same order (everytime they are shuffled), we've got to use the global level random as well as the operation level random see.

> Rule 4 : "If both the global and the operation seed are set : both seeds are used in conjunction to determmine the random sequence".

Others ways to make tensors

Entrée [30]:

```
tf.ones([4,3],dtype=tf.float16)
```

Out[30]:

```
<tf.Tensor: shape=(4, 3), dtype=float16, numpy=
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]], dtype=float16)>
```

Entrée [31]:

```
tf.zeros([5,2])
```

Out[31]:

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

Entrée [32]:

```
tf.zeros(shape=(5,2))
```

Out[32]:

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

Entrée [ ]:

```

```

Turning Numpy arrays into tensors

The main difference btewen Numpy arrays and TensorFlow tensors is that tensors can be run on a GPU (much faster for numerical computing)

Entrée [33]:

```
numpy_A = np.arange(1,25,dtype=np.int16)
numpy_A
# X = tf.constant(some_matrix)  #capital for matrix or tensor
# y = tf.constant(vector) #non-capital for vector
```

Out[33]:

```
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=int16)
```

Entrée [34]:

```
A = tf.constant(numpy_A)
A
```

Out[34]:

```
<tf.Tensor: shape=(24,), dtype=int16, 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=int16)>
```

Entrée [35]:

```
2*3*4
```

Out[35]:

```
24
```

Entrée [36]:

```
A2 = tf.constant(numpy_A, shape=(2,3,4))
A2
```

Out[36]:

```
<tf.Tensor: shape=(2, 3, 4), dtype=int16, 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=int16)>
```

Entrée [ ]:

```

```

Getting informations from tensors

- Shape : The length of each of the dimensions of a tensor
- Rank : The number of tensor dimensions. A scalar has rank 0, a vector has rank 1, a matrix is rand 2, a tensor has rank n
- Axis or dimension : A particular dimension of a tensor.
- Size : The total number of items in the tensor

Entrée [37]:

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

Out[37]:

```
<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)>
```

Entrée [38]:

```
rank_4_tensor[0]
```

Out[38]:

```
<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)>
```

Entrée [39]:

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

Out[39]:

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

Entrée [40]:

```
# Geting various attributes of our 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 the 0 axis : ", rank_4_tensor.shape[0])
print("Elements along the last axis : ", rank_4_tensor.shape[-1])
print("Total number of elements in our tensor : ", tf.size(rank_4_tensor))
print("Total number of elements in our tensor (2) : ", tf.size(rank_4_tensor).numpy())
```

```
Datatype of every element :  <dtype: 'float32'>
Number of dimensions (rank) :  4
Shape of tensor :  (2, 3, 4, 5)
Elements along the 0 axis :  2
Elements along the last axis :  5
Total number of elements in our tensor :  tf.Tensor(120, shape=(), dtype=int
32)
Total number of elements in our tensor (2) :  120
```

Entrée [ ]:

```

```

Indexing tensors

Tensors can be indexed just like Python lists

Entrée [41]:

```
# Creating a rank 4 tensor (4 dimensions)
rank_4_tensor2 = tf.ones([2,3,4,5])
rank_4_tensor2
```

Out[41]:

```
<tf.Tensor: shape=(2, 3, 4, 5), 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.],
         [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)>
```

Entrée [42]:

```
# Getting the first 2 elements of each dimension
rank_4_tensor2[:2,:2,:2,:2]
```

Out[42]:

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

        [[1., 1.],
         [1., 1.]]],


       [[[1., 1.],
         [1., 1.]],

        [[1., 1.],
         [1., 1.]]]], dtype=float32)>
```

Entrée [43]:

```
#Get the first element of each dimension from each index except the final one
rank_4_tensor2[:1,:1,:1,:]
```

Out[43]:

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

Entrée [44]:

```
#Get the first element of each dimension from each index except for the second last one
rank_4_tensor2[:1,:1,:,:1]
```

Out[44]:

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

Entrée [ ]:

```

```

Entrée [45]:

```
# Creating a rank 2 tensor (2 dimensions)
rank_2_tensor = tf.constant([[5,2,3], [19,5,8.]])
rank_2_tensor
```

Out[45]:

```
<tf.Tensor: shape=(2, 3), dtype=float32, numpy=
array([[ 5.,  2.,  3.],
       [19.,  5.,  8.]], dtype=float32)>
```

Entrée [46]:

```
rank_2_tensor.shape, rank_2_tensor.ndim
```

Out[46]:

```
(TensorShape([2, 3]), 2)
```

Entrée [47]:

```
# Getting the last item of each dimension of our rank 2 tensor
rank_2_tensor[:,-1]
```

Out[47]:

```
<tf.Tensor: shape=(2,), dtype=float32, numpy=array([3., 8.], dtype=float32)>
```

Entrée [48]:

```
# Adding in extra dimension to our rank 2 tensor (we will not change the informations,
rank_3_tensor = rank_2_tensor[...,tf.newaxis] # the same as rank_2_tensor[:,:,tf.newax
rank_3_tensor
```

Out[48]:

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

       [[19.],
        [ 5.],
        [ 8.]]], dtype=float32)>
```

Entrée [49]:

```
# Alternative to tf.newaxis
rank_3_tensor2 = tf.expand_dims(rank_2_tensor, axis=-1) # "-1" means expand the final
rank_3_tensor2
```

Out[49]:

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

       [[19.],
        [ 5.],
        [ 8.]]], dtype=float32)>
```

Entrée [50]:

```
tf.expand_dims(rank_2_tensor, axis=0) # expand the 0 axis
```

Out[50]:

```
<tf.Tensor: shape=(1, 2, 3), dtype=float32, numpy=
array([[[ 5.,  2.,  3.],
        [19.,  5.,  8.]]], dtype=float32)>
```

tf.expand_dims(rank_2_tensor, axis=1) # expand the 1 axis

Entrée [ ]:

```

```

Manipulating tensors (tensor operations)

Basic operations : + , - , * , / ,

Entrée [51]:

```
tensor = tf.constant([ [10,7,4],[12,1,9] ])
tensor
```

Out[51]:

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

Entrée [52]:

```
#Addition
tensor+10
```

Out[52]:

```
<tf.Tensor: shape=(2, 3), dtype=int32, numpy=
array([[20, 17, 14],
       [22, 11, 19]])>
```

Entrée [53]:

```
#Notice the original tensor is unchanged
tensor
```

Out[53]:

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

Entrée [54]:

```
tensor-2
```

Out[54]:

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

Entrée [55]:

```
tensor*2
```

Out[55]:

```
<tf.Tensor: shape=(2, 3), dtype=int32, numpy=
array([[20, 14,  8],
       [24,  2, 18]])>
```

Entrée [56]:

```
tensor/2
```

Out[56]:

```
<tf.Tensor: shape=(2, 3), dtype=float64, numpy=
array([[5. , 3.5, 2. ],
       [6. , 0.5, 4.5]])>
```

Entrée [57]:

```
tensor//2
```

Out[57]:

```
<tf.Tensor: shape=(2, 3), dtype=int32, numpy=
array([[5, 3, 2],
       [6, 0, 4]])>
```

Entrée [ ]:

```

```

Entrée [58]:

```
tensor
```

Out[58]:

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

Entrée [59]:

```
# We can use the TensorFlow built-in function
tf.multiply(tensor,2)
```

Out[59]:

```
<tf.Tensor: shape=(2, 3), dtype=int32, numpy=
array([[20, 14,  8],
       [24,  2, 18]])>
```

Entrée [60]:

```
tf.add(tensor,10)
```

Out[60]:

```
<tf.Tensor: shape=(2, 3), dtype=int32, numpy=
array([[20, 17, 14],
       [22, 11, 19]])>
```

Entrée [61]:

```
tf.math.add(tensor,20)
```

Out[61]:

```
<tf.Tensor: shape=(2, 3), dtype=int32, numpy=
array([[30, 27, 24],
       [32, 21, 29]])>
```

Matrix Multiplication

There are two rules our tensors (or matrices) need to fulful if we are going to matrxi multiply them:

- The inner dimensions of the two matrices must match
- The resulting matrix has the shape of the inner dimensions

Entrée [62]:

```
tensor = tf.constant([ [10,7],[3,4] ])
tensor
```

Out[62]:

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

Entrée [63]:

```
tf.matmul(tensor,tensor)
```

Out[63]:

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

Entrée [64]:

```
# Wrong way to do matrix multiplication with Python operator
tensor*tensor
```

Out[64]:

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

Entrée [65]:

```
# Matrix multiplication with Python operator "@"
tensor @ tensor
```

Out[65]:

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

Entrée [ ]:

```

```

Matrix multiplication with matrices having different shapes

Entrée [66]:

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

Out[66]:

```
(<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]])>)
```

Entrée [67]:

```
X.shape, Y.shape
```

Out[67]:

```
(TensorShape([3, 2]), TensorShape([3, 2]))
```

Entrée [68]:

```
# Reshaping Y
tf.reshape(Y,shape=(2,3))
```

Out[68]:

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

Entrée [69]:

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

Out[69]:

```
(TensorShape([3, 2]), TensorShape([2, 3]))
```

Entrée [70]:

```
# Matrix multiplying X by reshaped Y
X @ tf.reshape(Y,shape=(2,3))
```

Out[70]:

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

Entrée [71]:

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

Out[71]:

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

Entrée [72]:

```
# Matrix multiplying reshaped X by Y
tf.matmul(tf.reshape(X,shape=(2,3)),Y)
```

Out[72]:

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

Entrée [73]:

```
X
```

Out[73]:

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

Entrée [74]:

```
tf.transpose(X)
```

Out[74]:

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

Entrée [ ]:

```

```

The dot product

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

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

Entrée [75]:

```
X,Y
```

Out[75]:

```
(<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]])>)
```

Entrée [76]:

```
# Performing dot product on X and Y (requires X or Y to be transposed)
tf.tensordot(tf.transpose(X),Y, axes=1)
```

Out[76]:

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

Entrée [77]:

```
tf.tensordot( tf.reshape(X,shape=(2,3)), Y, axes=1 )
```

Out[77]:

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

Entrée [78]:

```
# Checking the values of Y, reshape Y, and transposed Y
print("Normal Y: ")
print(Y, "\n")

print("Y reshaped to (2,3): ")
print(tf.reshape(Y, (2,3)), "\n")

print("Y transposed: ")
print(tf.transpose(Y))
```

```
Normal Y:
tf.Tensor(
[[ 7  8]
 [ 9 10]
 [11 12]], shape=(3, 2), dtype=int32)

Y reshaped to (2,3):
tf.Tensor(
[[ 7  8  9]
 [10 11 12]], shape=(2, 3), dtype=int32)

Y transposed:
tf.Tensor(
[[ 7  9 11]
 [ 8 10 12]], shape=(2, 3), dtype=int32)
```

Generally, when performing matrix multiplication on two tensors and one of the axes doesn't line up, you will transpose (rather than reshape) one of the tensors to satisfy matrix multiplication rules.

Entrée [ ]:

```

```

Changing the datatype of a tensor

Entrée [79]:

```
tf.__version__
```

Out[79]:

```
'2.10.0'
```

Entrée [80]:

```
# Create a new tensor with default datatype (float32)
tensor = tf.ones(shape=(3,2))
tensor
```

Out[80]:

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

Entrée [81]:

```
B = tf.constant([1.7, 1.2])
B.dtype
```

Out[81]:

```
tf.float32
```

Entrée [82]:

```
C = tf.constant([1, 5])
C.dtype
```

Out[82]:

```
tf.int32
```

Entrée [83]:

```
# Change from float32 to float16 (reduce precision)
D = tf.cast(B, dtype=tf.float16)
D.dtype
```

Out[83]:

```
tf.float16
```

Entrée [84]:

```
# Change from int32 to float32
E = tf.cast(C,dtype=tf.float32)
E.dtype
```

Out[84]:

```
tf.float32
```

Entrée [ ]:

```

```

Aggregating tensors

Aggreating tensors = Condensing them from multiple values down to a smaller amount of values

Entrée [85]:

```
D = tf.constant([-7, -10])
D
```

Out[85]:

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

Entrée [86]:

```
# Get the absolute values
tf.abs(D)
```

Out[86]:

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

Let's go through the following forms of aggregation :

- Get the minimum of a tensor
- Get the maximum of a tensor
- Get the mean of a tensor
- Get the sum of a tensor

Entrée [87]:

```
# Create a random tensor with values between 0 and 100 of size 50
E = tf.constant(np.random.randint(0,100, size=50))
E
```

Out[87]:

```
<tf.Tensor: shape=(50,), dtype=int32, numpy=
array([35, 24,  4, 31, 10,  8, 34, 23, 91, 28, 59, 28, 59, 12, 66, 60, 32,
       53, 36,  8, 87, 54, 59, 60, 23, 55, 49, 54, 55, 44, 81, 53, 44, 41,
       11, 79, 32, 44, 86, 89,  3, 14, 19, 52, 38,  3, 89, 41, 34,  0])>
```

Entrée [88]:

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

Out[88]:

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

Entrée [89]:

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

Out[89]:

```
<tf.Tensor: shape=(), dtype=int32, numpy=0>
```

Entrée [90]:

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

Out[90]:

```
<tf.Tensor: shape=(), dtype=int32, numpy=91>
```

Entrée [91]:

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

Out[91]:

```
<tf.Tensor: shape=(), dtype=int32, numpy=41>
```

Entrée [92]:

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

Out[92]:

```
<tf.Tensor: shape=(), dtype=int32, numpy=2094>
```

Entrée [93]:

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

Out[93]:

```
<tf.Tensor: shape=(), dtype=float32, numpy=630.78564>
```

Entrée [94]:

```
# Find the variance
tfp.stats.variance(E)
```

Out[94]:

```
<tf.Tensor: shape=(), dtype=int32, numpy=631>
```

Entrée [95]:

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

Out[95]:

```
<tf.Tensor: shape=(), dtype=float32, numpy=25.115446>
```

Entrée [96]:

```
# Find the standard deviation
tfp.stats.stddev( tf.cast(E, dtype=tf.float32) )
```

Out[96]:

```
<tf.Tensor: shape=(), dtype=float32, numpy=25.115446>
```

Entrée [ ]:

```

```

Finding the positional maximum and minimum

Entrée [97]:

```
# Creating a new tensor for finding potitional minimum and maximum
tf.random.set_seed(42)
F = tf.random.uniform(shape=[50])
F
```

Out[97]:

```
<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)>
```

Entrée [98]:

```
# Find the positional maximum (Index of the largest value position)
tf.argmax(F)
```

Out[98]:

```
<tf.Tensor: shape=(), dtype=int64, numpy=42>
```

Entrée [99]:

```
np.argmax(F)
```

Out[99]:

```
42
```

Entrée [100]:

```
# Value of the positional maximum
F[ tf.argmax(F) ]
```

Out[100]:

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

Entrée [101]:

```
# Find the max value of F
tf.reduce_max(F)
```

Out[101]:

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

Entrée [102]:

```
# Check for equality
F[ tf.argmax(F) ] == tf.reduce_max(F)
```

Out[102]:

```
<tf.Tensor: shape=(), dtype=bool, numpy=True>
```

Entrée [103]:

```
# Find the positional minimum (Index of the lower value position)
tf.argmin(F)
```

Out[103]:

```
<tf.Tensor: shape=(), dtype=int64, numpy=16>
```

Entrée [104]:

```
# Find the minimum using the positional minimum index
F[ tf.argmin(F) ]
```

Out[104]:

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

Entrée [105]:

```
tf.reduce_min(F)
```

Out[105]:

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

Entrée [ ]:

```

```

Squeezing a tensor (removing all single dimensions)

Entrée [106]:

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

Out[106]:

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

Entrée [107]:

```
G.shape
```

Out[107]:

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

Entrée [108]:

```
G_squeezed = tf.squeeze(G)
G_squeezed
```

Out[108]:

```
<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)>
```

Entrée [ ]:

```

```

One-hot encoding

One-hot encode a list

Entrée [109]:

```
some_list = [0,1,2,3]

tf.one_hot( some_list, depth= 4 )
```

Out[109]:

```
<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)>
```

Entrée [110]:

```
some_list2 = [0,1,2,3]

tf.one_hot( some_list2, depth= 5 )
```

Out[110]:

```
<tf.Tensor: shape=(4, 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.]], dtype=float32)>
```

Entrée [111]:

```
some_list3 = [1,2,0,6]

tf.one_hot( some_list3, depth= 7 )
```

Out[111]:

```
<tf.Tensor: shape=(4, 7), dtype=float32, numpy=
array([[0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1.]], dtype=float32)>
```

Specify custom value for one-hot encoding

Entrée [112]:

```
# Specify custom value for one-hot encoding
tf.one_hot(some_list, depth=4, on_value="yo I love deep learning", off_value="I also li
```

Out[112]:

```
<tf.Tensor: shape=(4, 4), dtype=string, numpy=
array([[b'yo I love deep learning', b'I also like to dance',
        b'I also like to dance', b'I also like to dance'],
       [b'I also like to dance', b'yo I love deep learning',
        b'I also like to dance', b'I also like to dance'],
       [b'I also like to dance', b'I also like to dance',
        b'yo I love deep learning', b'I also like to dance'],
       [b'I also like to dance', b'I also like to dance',
        b'I also like to dance', b'yo I love deep learning']],
      dtype=object)>
```

Entrée [ ]:

```

```

Squaring, log, square root

Entrée [113]:

```
# Create a new tensor
H = tf.range(1,10)
H
```

Out[113]:

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

Entrée [114]:

```
# Square a tensor
tf.square(H)
```

Out[114]:

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

Entrée [115]:

```
# Find the log of a tensor
tf.math.log( tf.cast(H, dtype=tf.float32) )
```

Out[115]:

```
<tf.Tensor: shape=(9,), dtype=float32, numpy=
array([0.       , 0.6931472, 1.0986123, 1.3862944, 1.609438 , 1.7917595,
       1.9459102, 2.0794415, 2.1972246], dtype=float32)>
```

Entrée [116]:

```
# Find the square root of a tensor
tf.sqrt( tf.cast(H, dtype=tf.float32) )
```

Out[116]:

```
<tf.Tensor: shape=(9,), dtype=float32, numpy=
array([1.       , 1.4142135, 1.7320508, 2.       , 2.236068 , 2.4494898,
       2.6457512, 2.828427 , 3.       ], dtype=float32)>
```

Entrée [ ]:

```

```

TensorFlow and NumPy's compatibility

TensorFlow interacts beautiffuly with NumPy arrays.
One of the main differences between a TensorFlow tensor and a NumPy array, though, is that a TensorFlow tensor can be run on a GPU or TPU (for faster numercial processing).

Entrée [117]:

```
# Create a tensor directly from a NumPy array
H = tf.constant( np.arange(0,10) )
H
```

Out[117]:

```
<tf.Tensor: shape=(10,), dtype=int32, numpy=array([0, 1, 2, 3, 4, 5, 6, 7,
8, 9])>
```

Entrée [118]:

```
# Create a tensor directly from a NumPy array
J = tf.constant( np.array( [3.,7,10] ) )
J
```

Out[118]:

```
<tf.Tensor: shape=(3,), dtype=float64, numpy=array([ 3.,  7., 10.])>
```

Entrée [121]:

```
# Convert a tensor to NumPy array
np.array(J)
```

Out[121]:

```
array([ 3.,  7., 10.])
```

Entrée [122]:

```
type(np.array(J))
```

Out[122]:

```
numpy.ndarray
```

Entrée [123]:

```
# Convert tensor to NumPy array
J.numpy()
```

Out[123]:

```
array([ 3.,  7., 10.])
```

Entrée [124]:

```
type(J.numpy())
```

Out[124]:

```
numpy.ndarray
```

Entrée [125]:

```
J
```

Out[125]:

```
<tf.Tensor: shape=(3,), dtype=float64, numpy=array([ 3.,  7., 10.])>
```

Entrée [126]:

```
J.numpy()[0]
```

Out[126]:

```
3.0
```

Entrée [128]:

```
J.numpy()[-1]
```

Out[128]:

```
10.0
```

Entrée [129]:

```
# The default types of each are slightly different
J1 = tf.constant( np.array([3.,7,10.]) )
J2 = tf.constant( [3.,7,10.] )
# Check the datatype of each
J1.dtype, J2.dtype
```

Out[129]:

```
(tf.float64, tf.float32)
```

Entrée [ ]:

```

```

Findind access to GPUs

Entrée [130]:

```
tf.test.is_gpu_available
```

Out[130]:

```
<function tensorflow.python.framework.test_util.is_gpu_available(cuda_only=F
alse, min_cuda_compute_capability=None)>
```

Entrée [133]:

```
tf.config.list_physical_devices()
```

Out[133]:

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

Entrée [134]:

```
tf.config.list_physical_devices("CPU")
```

Out[134]:

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

Entrée [135]:

```
tf.config.list_physical_devices("GPU")
```

Out[135]:

```
[]
```

Entrée [136]:

```
# Check what type of GPU is being used
!nvidia-smi
```

```
'nvidia-smi' n'est pas reconnu en tant que commande interne
ou externe, un programme ex,cutable ou un fichier de commandes.
```

> **Note:** If there is access to a CUDA-enabled GPU, TensorFlow will automatically use it whenever possible.

Entrée [ ]:

```

```

Entrée [ ]:

```

```