# Introduction_to_TensorFlow

September 26, 2021

[ ]:

## 1 Introduction to TensorFlow

TensorFlow is a software library used for high-performance numerical computation. It can be used from a variety of languages and deployed on a large number of operating systems and hardware platforms.

TensorFlow is particularly good at machine learning and is often associated with it. However, TensorFlow isn't limited to machine learning applications.

### 1.1 TensorFlow Version

In late 2019, TensorFlow moved from version `1.x` to version `2.x`. This move signaled a major change in how programmers use TensorFlow. Many of the APIs used in version 1 are no longer core to the libarary.

In this course we use TensorFlow 2 exclusively.

TensorFlow 1 and TensorFlow 2 can't be installed in the same environment at the same time. As of early 2020, Colab still defaulted to using TensorFlow 1. In order to enable TensorFlow 2, run the code cell below:

[ ]:
```
%tensorflow_version 2.x
```

```
UsageError: Line magic function `%tensorflow_version` not found.
```

This cell runs a magic that tells Colab to use TensorFlow 2 instead of TensorFlow 1 by default. This magic needs to run before you actually load TensorFlow.

In the future this magic might not be necessary. If you are ever in doubt of which TensorFlow you are using, import `tensorflow` and print out the version.

```
[ ]: import tensorflow as tf

     print(tf.__version__)
```

```
2.4.1
```

## 1.2 Tensors

TensorFlow gets its name from **tensors**, which are arrays of arbitrary dimensionality. Using TensorFlow, you can manipulate tensors with a very high number of dimensions. With that being said, most of the time you will work with one or more of the following low-dimensional tensors:

### 1.2.1 Scalars

A **scalar** is a 0-d array (a 0th-order tensor). An example might be `"Howdy"` or `5`.

How would we create a scalar tensor, or any tensor for that matter?

One ways is to use `tf.constant()`. This creates a **constant tensor**, which is a tensor that cannot change.

```
[ ]: scalar_tensor = tf.constant("Hi Mom!")
     print(scalar_tensor)
```

```
tf.Tensor(b'Hi Mom!', shape=(), dtype=string)
```

Notice that the shape of the tensor is `()`, which indicates that the tensor is a scalar.

### 1.2.2 Vectors

A **vector** is a 1-d array (a 1st-order tensor). An example might be `[2, 3, 5, 7, 11]` or even `[5]`.

We can again use `tf.constant()` to create the tensor.

```
[ ]: vector_tensor = tf.constant([1, 2, 3])
     print(vector_tensor)
```

```
tf.Tensor([1 2 3], shape=(3,), dtype=int32)
```

Notice now that our shape has changed to `(3,)`.

### 1.2.3 Matrix

A **matrix** is a 2-d array (a 2nd-order tensor). For example:

```
[
  [3.1, 8.2, 5.9],
  [4.3, -2.7, 6.5],
]
```

```
[ ]: matrix_tensor = tf.constant([[1.2, 3.4, 5.6], [7.8, 9.0, 1.2]])
     print(matrix_tensor)
```

```
tf.Tensor(
[[1.2 3.4 5.6]
 [7.8 9.  1.2]], shape=(2, 3), dtype=float32)
```

### 1.2.4 Cubes

A **cube** is a 3d array (a 3rd-order tensor). For example:

```
  [
    [
      [3.1, 8.2, 5.9],
      [4.3, 2.7, 6.5],
    ],
    [
      [4.5, 5.2, 3.1],
      [3.4, 2.0, 5.9],
    ],
    [
      [4.2, 3.7, 9.1],
      [6.4, 1.2, 6.4],
    ],
    [
      [9.9, 6.1, 8.8],
      [3.1, 8.7, 4.5],
    ],
  ]
```

**Exercise 1: Create a Cube** Using `tf.constant()` create 3rd-order tensor. Print the tensor.

**Student Solution**

```
[ ]: # Your Code Goes Here




     #shape(second arguements equates to the order of tensor)
     #order equates to the number of arrays within the array
     new_tensor = tf.constant(([[1.2, 3.4, 5.6], [7.8, 9.0, 1.2], [1,2,3]]))
     print (new_tensor)
```

```
tf.Tensor(
[[1.2 3.4 5.6]
 [7.8 9.  1.2]
 [1.  2.  3. ]], shape=(3, 3), dtype=float32)
```

---

### 1.2.5 Higher-Order Tensors

Tensors can have an arbitrarily large number of dimensions. Your mathematical needs might require large numbers of dimensions, but for the most part you'll likely see tensors in these smaller dimensions that we've covered.

### 1.2.6 Variable Tensors

A variable is a type of tensor that can change. Creating variable tensors doesn't look much different from creating constant tensors.

```
[ ]: vector_tensor = tf.Variable([1, 2, 3])
     print(vector_tensor)
```

```
<tf.Variable 'Variable:0' shape=(3,) dtype=int32, numpy=array([1, 2, 3],
dtype=int32)>
```

But with variable tensors you can change the values using the `assign()` method.

```
[ ]: vector_tensor = tf.Variable([1, 2, 3])

     vector_tensor.assign([5, 6, 7])

     print(vector_tensor)
```

```
<tf.Variable 'Variable:0' shape=(3,) dtype=int32, numpy=array([5, 6, 7],
dtype=int32)>
```

### 1.2.7 Tensor Operations

In order for variables to change, something must operate on them. TensorFlow operations create, destroy, and manipulate tensors. Most of the lines of code in a typical TensorFlow program are operations.

Let's start by looking at a basic operation that adds two tensors together.

```
[ ]: tensor = tf.constant(3) + tf.constant(4)
     print(tensor)
```

```
tf.Tensor(7, shape=(), dtype=int32)
```

We create two scalar constant tensors and add them together. The result is another tensor.

Notice that in order to perform a simple tensor addition operation, we could simply use the standard Python addition symbol. This is because TensorFlow overrides many of the standard operators for use with `Tensor` objects.

This override works as long as there is a `Tensor` on at least one side of the operator:

```
[ ]: print(tf.constant(3) + 4)
     print(3 + tf.constant(4))
```

```
tf.Tensor(7, shape=(), dtype=int32)
tf.Tensor(7, shape=(), dtype=int32)
```

TensorFlow also has support for many more operations than can be represented by standard Python operators. For instance, there are a number of linear algebra operations such as the matrix transposition:

```
[ ]: matrix = tf.constant([
       [11, 12, 13],
       [21, 22, 23],
       [31, 32, 33],
     ])

     print(tf.transpose(matrix))
```

```
tf.Tensor(
[[11 21 31]
 [12 22 32]
 [13 23 33]], shape=(3, 3), dtype=int32)
```

You can also calculate matrix multiplication using `tf.tensordot()`:

```
[ ]: matrix_one = tf.constant([
       [2, 2, 2],
       [2, 2, 2],
       [2, 2, 2],
     ])

     matrix_two = tf.constant([
       [2, 2, 2],
       [2, 2, 2],
       [2, 2, 2],
     ])

     tf.tensordot(matrix_one, matrix_two, axes=1)
```

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

This is necessary because by default * performs elementwise multiplication:

```
[ ]: matrix_one = tf.constant([
       [2, 2, 2],
       [2, 2, 2],
       [2, 2, 2],
     ])

     matrix_two = tf.constant([
       [2, 2, 2],
       [2, 2, 2],
       [2, 2, 2],
     ])

     matrix_one * matrix_two
```

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

Interestingly, `Tensor` objects can also by passed to NumPy functions in some cases:

[ ]:
```python
import tensorflow as tf
import numpy as np

matrix_one = tf.constant([
  [2, 2, 2],
  [2, 2, 2],
  [2, 2, 2],
])

matrix_two = tf.constant([
  [2, 2, 2],
  [2, 2, 2],
  [2, 2, 2],
])

np.dot(matrix_one, matrix_two)
```

[ ]:
```
array([[12, 12, 12],
       [12, 12, 12],
       [12, 12, 12]], dtype=int32)
```

**Exercise 2: Linear Equation** Create a TensorFlow graph that performs the linear equation, `y = mx + b`.

- `m` should be a 0D constant tensor with a value of 12.
- `b` should be a 0D constant tensor with a value of 32.
- `x` should be a 1D constant tensor of shape (5,) and any values you choose.
- `y` should be 1D tensor that receives the result of `mx + b`.

Run the equation and print `y`.

**Student Solution**

[ ]:
```python
# Your code goes here
```

### 1.2.8 Extracting Values

Notice that every value that we have seen so far has been a Tensor. This is fine as long as you are working inside of TensorFlow (and even NumPy) in many cases, but what if you wanted to extract the values in a Tensor out to more standard Python values?

For that you can use the `.numpy()` method.

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

     tensor.numpy()
```

```
[ ]: array([[1, 2],
            [3, 4]], dtype=int32)
```

**Exercise 3: Tensor to Python List** In the example above, you found out how to take the values stored in a `Tensor` object and convert them to a NumPy array using the `numpy()` method. In this exercise you'll go one step further and convert the `Tensor` value into a core Python list.

**Student Solution**

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

     # Extract the values inside the tensor as a Python list stored
     # in a variable named tensor_values.

     tensor_num = tensor.numpy()

     tensor_values = tensor_num.tolist()



     # Print the type of tensor_values
     print(type(tensor_values))
     # Print tensor_values

     print(tensor_values)
```

```
<class 'list'>
[[1, 2], [3, 4]]
```

---

### 1.2.9 Conclusion

In this lab we briefly covered TensorFlow versions and introduced you to the concept of tensors. This is just the tip of the iceberg in regard to how to use TensorFlow. In future labs, you'll learn about the TensorFlow Estimator interface and the Keras interface. You'll train models, measure model quality, and use models to make predictions. You'll store model state and load saved models.