Basics of TensorFlow

What is TensorFlow?

TensorFlow is an open-source software library designed for training deep learning models. It provides all the necessary tools to create and manage neural networks. With TensorFlow, we can train neural networks ranging from simple to complex architectures using large datasets.
TensorFlow is utilized in a variety of applications, including image and speech recognition, natural language processing (NLP), and robotics.

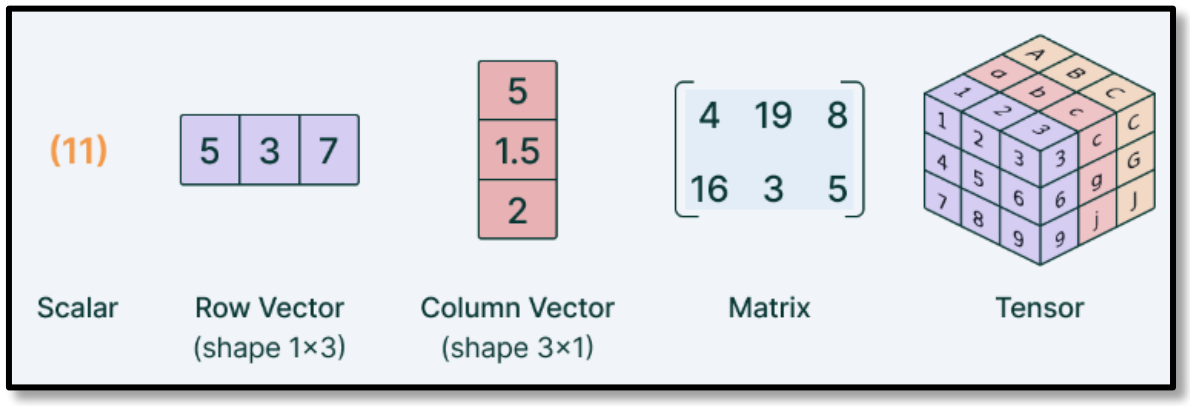
By leveraging TensorFlow, we can build powerful AI models that achieve high accuracy and performance.
In this lecture, we will explore tensors and learn how to work with them using TensorFlow.

What is a Tensor?

In TensorFlow, all computations involve tensors. A tensor is a mathematical object that generalizes scalars, vectors, and matrices. More specifically, a tensor can be thought of as a vector or a matrix of n dimensions, representing various types of data.

To clarify:

- A **scalar** is a single number
- A **vector** is a one-dimensional array of numbers
- A **matrix** is a 2-dimensional array
- A **tensor** is an n-dimensional array



Variables, constants, Placeholders and Sparse Tensors are four main tensors we can create.

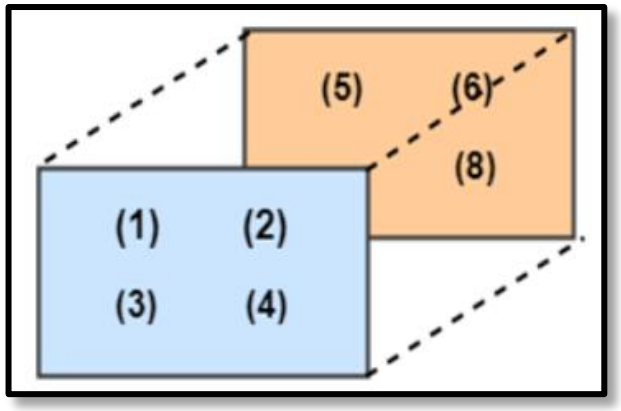
TensorFlow representation of tensors

Now, let us examine the representation of a tensor in TensorFlow. Consider the following matrix:

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

TensorFlow represents this matrix as follows:

[[1, 3, 5],
[2, 4, 6]]



As another example, consider the above 3 matrix.
TensorFlow represents this matrix as the following:

[[[1, 2],
[3, 4]],
[[5, 6],
[7, 8]]]

Code implementation of a tensor

The most important attributes of a tensor are its **name**, **shape** (dimension) and **dtype** (data type).

- $\boldsymbol{Tensor.shape}$ indicates the size of the tensor along each of its axes.
- $\boldsymbol{Tensor.dtype}$ indicates the type of all the elements of the tensor.

Now, let us take a look at the code implementation of a tensor.

```
import tensorflow as tf

x = tf.constant([[1., 2., 3.],
                 [4., 5., 6.]])

print(x)
print(x.shape)
print(x.dtype)
```

Here is the output of this code:

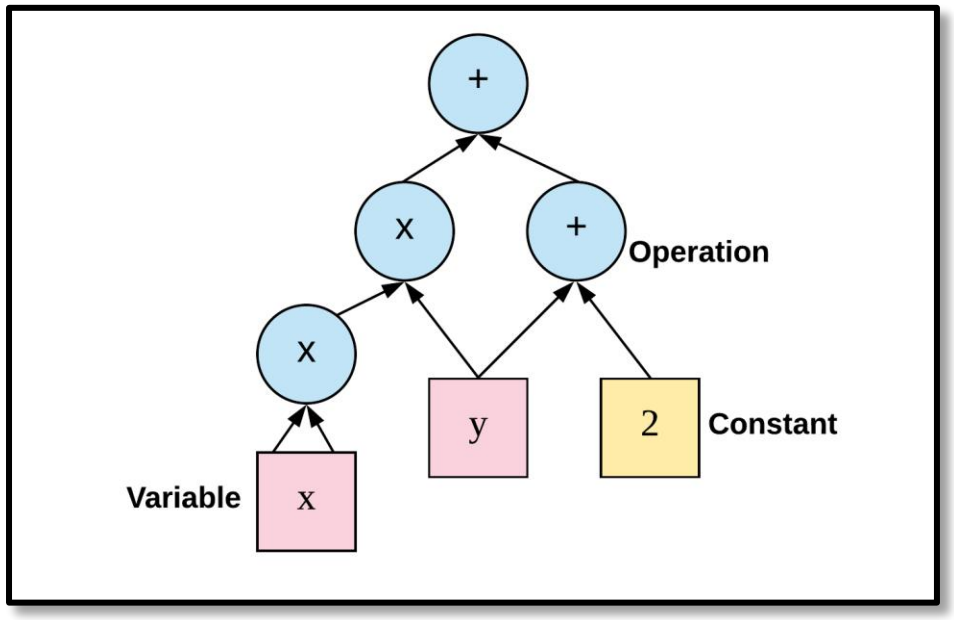
```
[[1. 2. 3.]
 [4. 5. 6.]]
(2, 3)
<dtype: 'float32'>
```

What is a Computational Graph?

A computational graph in TensorFlow is a way to represent and execute mathematical operations. TensorFlow computations are represented as a Directed Acyclic Graph (DAG), where nodes represent operations and edges represent the data flow between operations.

Nodes is a computational graph represent operations like addition, multiplication, etc.

Edges in a computational graph represent the data dependencies-



between operations, like function arguments.

What is a Session?

A TensorFlow session is used to run operations and evaluate tensors in a computational graph.
Sessions allocate resources to execute operations defined in a graph. They hold the actual values of intermediate results and variables during a graph execution.
In order to create a session in TensorFlow, we use the $tf.Session()$ constructor. We use the $with$ statement to create a session as a context manager. This automatically-

```python
# Using tf.Session()
sess = tf.Session()
# Use sess to run operations
sess.close()

# Using a context manager
with tf.Session() as sess:
    # Use sess to run operations
    pass
```

closes the session when exiting the block.

What is a Session?

Here is an example demonstrating the execution of an operation using a session in TensorFlow.
First, we create two constants (which will be explained further in the lecture) with initialized values.
Next, we define an operation *c* that multiplies the two constants.
Finally, we create a session and, within the session, execute the multiplication operation to retrieve the result.

We can observe that the output is 30.

```
import tensorflow as tf

a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b

with tf.Session() as sess:
    result = sess.run(c)
    print(result)
```

```
30.0
```

Types of tensors

1. tf.Variable

A $\boldsymbol{tf.Variable}$ in TensorFlow is a tensor whose value can be modified by running ops on it. It provides a flexible and efficient way to store and modify tensors as its value can be updated as the program runs.

A variable is created using the $\boldsymbol{tf.Variable}()$ constructor, which takes an initial value. The initial value defines the type and shape of the variable, which are fixed after creation.

The value of a variable can be modified using different methods, namely $\boldsymbol{assign}()$, $\boldsymbol{assign\_add}()$ and $\boldsymbol{assign\_sub}()$.

We will now examine a code example that will further clarify these explanations.

Types of tensors

1. tf.Variable

In this example, we first create a variable using the constructor and assign it an initial value. Next, we modify the value of the variable. Finally, we increment the elements of our tensor by one.
The given code creates a variable var with an initial value of $[0.0, 0.0, 0.0]$, then updates it to $[1, 2, 3]$, and finally adds $[1, 1, 1]$ to it, resulting in the final value of $[2, 3, 4]$.

```python
import tensorflow as tf

var = tf.Variable([0.0, 0.0, 0.0])
print(var)
var.assign([1, 2, 3])
print(var)
var.assign_add([1, 1, 1])
print(var)
```

```
 <tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.], dtype=float32)>
 <tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([1., 2., 3.], dtype=float32)>
 <tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([2., 3., 4.], dtype=float32)>
```

Types of tensors

2. tf.constant

A $tf.constant$ in TensorFlow is a tensor whose value cannot be modified after it is created. This means once a constant is set, it cannot be altered, making it suitable for values that should remain constant throughout the program.
A constant is created using the $tf.constant()$ constructor, which takes an initial value. The initial value cannot be modified and defines the type and shape of the constant, which are fixed after creation.
We will now examine a code example that will further clarify these explanations.

Types of tensors

2. tf.constant

In this example, we first create a constant using the constructor and assign it an initial value. Next, we print the value of the constant.
The given code creates a constant $sample\_data$ with an initial value of $[1, 2, 3, 4, 5, 6]$.
As we can observe, the output of the provided code includes the value of the tensor, as well as its shape and data type.

```
import tensorflow as tf

sample_data = tf.constant([1, 2, 3, 4, 5, 6])
print(sample_data)
```

```
tf.Tensor([1 2 3 4 5 6], shape=(6,), dtype=int32)
```

Types of tensors

3. tf.placeholder

A $\boldsymbol{tf.placeholder}$ in TensorFlow is a specialized type of tensor that serves as a placeholder for the input data. It enables the construction of the computation graph without requiring the actual data until runtime. In other words, a placeholder is a variable that can be populated with data at a later time.

A placeholder is created using the $\boldsymbol{tf.placeholder}()$ constructor.

However, it does not require an initial value.

We will now examine a code example that will further clarify these explanations.

Types of tensors

3. tf.placeholder

In this example, we first create a placeholder using the constructor. It can accept a $\boldsymbol{float32}$ type input. The shape is set to $\boldsymbol{None}$, which means it can accept any shape of input data.
Next, we add 10 to the input tensor. Finally, we create a TensorFlow session $sess$ and execute the operation b, feeding the placeholder a with the list $[10, 20, 30, 40]$.

```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

a = tf.placeholder(tf.float32, None)
b = a + 10

with tf.Session() as sess:
    result = sess.run(b, feed_dict={a: [10, 20, 30, 40]})
    print(result)
```

```
[20. 30. 40. 50.]
```

Types of tensors

4. tf.SparseTensor

A $tf.SparseTensor$ in TensorFlow is a special type of tensor, used to represent sparse data efficiently. It is defined by three components:

- **Indices**: A 2-D tensor that contains the indices of the elements in the sparse tensor.
- **Values**: A 1-D tensor that contains the values of the elements in the sparse tensor.
- **dense_shape**: A 1-D tensor that contains the shape of the dense tensor that the sparse tensor represents.

A sparse tensor is created using the $tf.SparseTensor()$ constructor.

We will now examine a code example that will further clarify these explanations.

Types of tensors

4. tf.SparseTensor

In this example, $\boldsymbol{indices}$ specifies that the values 1 and 2 are located at indices $(0, 0)$ and $(1, 2)$ respectively.
$\boldsymbol{values}$ contains the actual values of the sparse tensor.
$\boldsymbol{dense\_shape}$ specifies that the sparse tensor represents a 3×4 dense tensor.

We can observe the dense tensor represented by this sparse tensor.

```
indices = [[0, 0], [1, 2]]
values = [1, 2]
dense_shape = [3, 4]

sparse_tensor = tf.SparseTensor(indices, values, dense_shape)
```

```
[[1 0 0 0]
 [0 0 2 0]
 [0 0 0 0]]
```