Lab No.	Topics
1.	Overview Of TensorFlow and Its Operations
2.	

Lab 1

OBJECTIVE:

- Understand the basics of deep learning and its applications.
- Set up the Python environment for deep learning.
- Perform basic operations using TensorFlow and Scikit-learn.

heory

Tools/Libraries to Use:

1. **TensorFlow**: A powerful library for deep learning. It's widely used for building and training neural networks.
2. **Scikit-learn**: A machine learning library for simpler tasks like data preprocessing, basic ML algorithms, and evaluation metrics.
3. **NumPy**: For numerical computations.
4. **Matplotlib**: For data visualization.

Why TensorFlow and Scikit-learn?

a) TensorFlow:

1 Core deep learning engine for building and training neural networks.
2 Handles complex computations and large-scale models.
3 Keras integration simplifies model creation.

b) Scikit-learn:

1. Essential for data preprocessing and feature engineering.
2. Provides tools for understanding basic machine learning concepts.
3. Enables easy model evaluation and data splitting.
4. Creates baseline models for comparison.

c) Synergy:

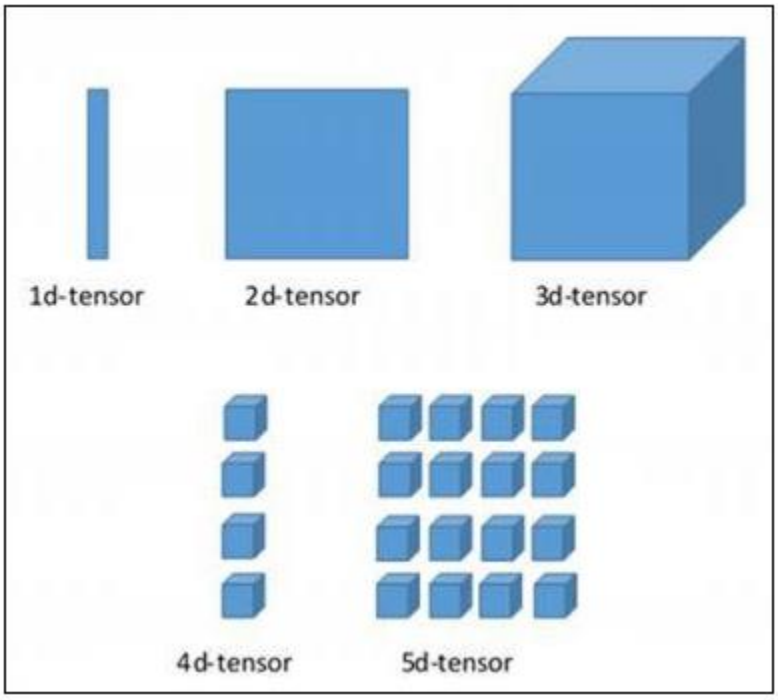
1. Scikit-learn prepares data and provides a foundation, TensorFlow handles deep learning.
2. Combined, they offer a complete workflow from data to deployment.

d) **NumPy:** Efficient numerical operations, array manipulation.

e) **Matplotlib:** Data visualization, model performance analysis.

What are Tensors?

In this TensorFlow tutorial, before talking about TensorFlow, let us first understand *what are tensors*. Tensors are nothing but a de facto for representing the data in deep learning.



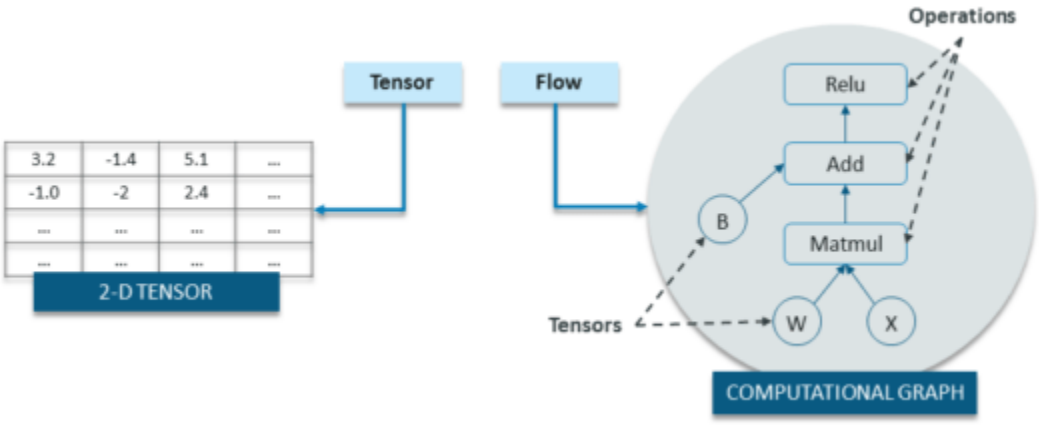
In TensorFlow, a **tensor** is a generalized form of vectors and matrices to potentially higher dimensions. It is the primary data structure used in TensorFlow to represent data. Tensors are similar to NumPy arrays but with additional features that make them suitable for machine learning tasks, such as GPU acceleration and automatic differentiation.

Key Properties of Tensors:

1. **Rank**: The number of dimensions in a tensor. For example:
    - A scalar (a single number) is a rank-0 tensor.
    - A vector is a rank-1 tensor.
    - A matrix is a rank-2 tensor.
    - Higher-dimensional arrays are rank-3 or higher tensors.
2. **Shape**: The size of each dimension of the tensor. For example, a tensor with shape (3, 4) has 3 rows and 4 columns.
3. **Data Type**: The type of data stored in the tensor, such as float32, int32, or bool.

What is TensorFlow?

TensorFlow is a library based on Python that provides different types of functionality for implementing Deep Learning Models. As discussed earlier, the term TensorFlow is made up of two terms – Tensor & Flow:



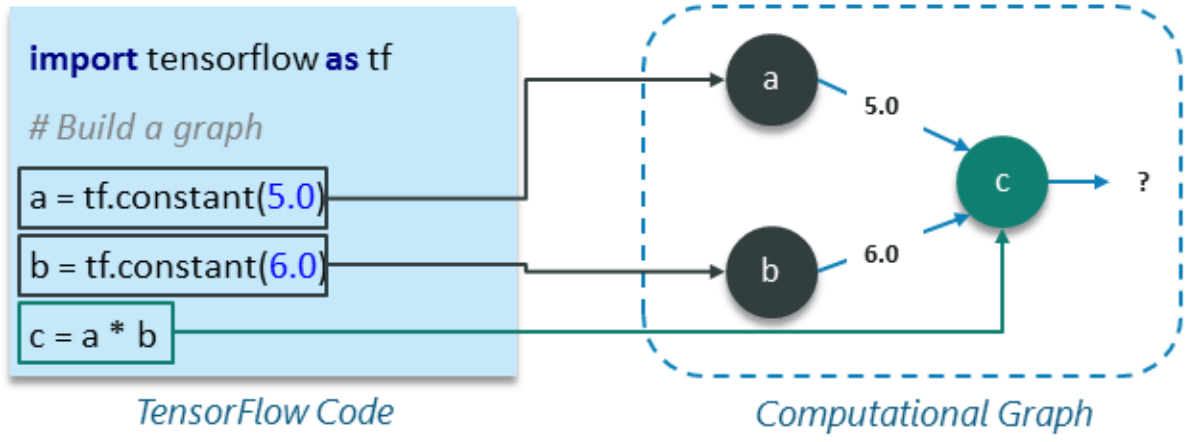
In TensorFlow, the term tensor refers to the representation of data as multi-dimensional array whereas the term flow refers to the series of operations that one performs on tensors as shown in the above image

Code Basics

Basically, the overall process of writing a TensorFlow program involves two steps:

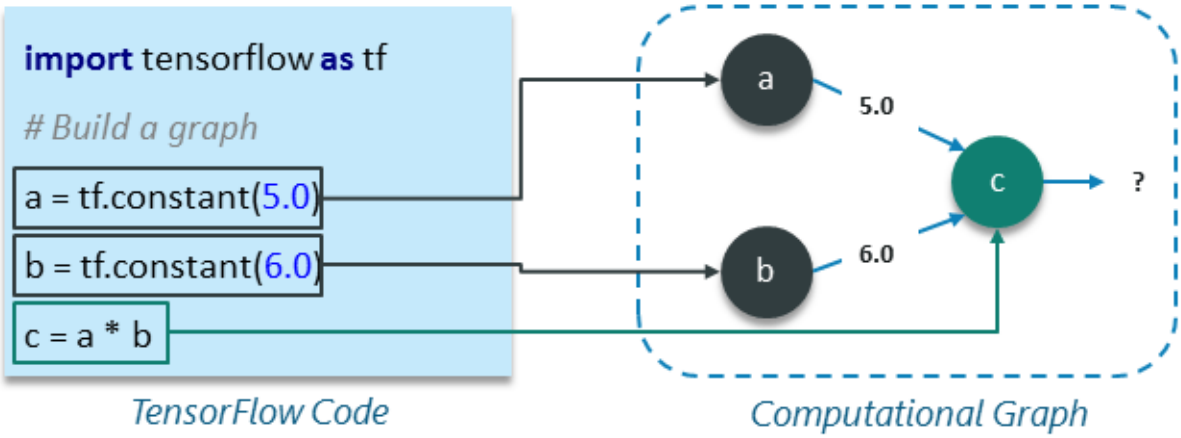
1. Building a Computational Graph
2. Running a Computational Graph

Let me explain you the above two steps one by one:



1. Building a Computational Graph

So, *what is a computational graph?* Well, a computational graph is a series of TensorFlow operations arranged as nodes in the graph. Each nodes take 0 or more tensors as input and produces a tensor as output. Let me give you an example of a simple computational graph which consists of three nodes – ***a***, ***b*** & ***c*** as shown below:



2. Running a Computational Graph

Let us take the previous example of computational graph and understand how to execute it. Following is the code from previous example:

```
1  import tensorflow as tf
2
3  # Build a graph
4  a = tf.constant(5.0)
5  b = tf.constant(6.0)
6  c = a * b
```

Now, in order to get the output of node c, we need to run the computational graph within a **session**. Session places the graph operations onto Devices, such as CPUs or GPUs, and provides methods to execute them.

A session encapsulates the control and state of the TensorFlow runtime i.e. it stores the information about the order in which all the operations will be performed and passes the result of already computed operation to the next operation in the pipeline. Let me show you how to run the above computational graph within a session (Explanation of each line of code has been added as a comment):

```
# Create the session object
sess = tf.Session()

#Run the graph within a session and store the output to a variable
output_c = sess.run(c)

#Print the output of node c
print(output_c)

#Close the session to free up some resources
sess.close()
```

Output:

```
30
```

Task.1

1. Scalar (Rank-0 Tensor)

A scalar is a single value with no dimensions.

Example.

```
[1]: import tensorflow as tf

# Create a scalar tensor
scalar = tf.constant(5)

print("Scalar:", scalar)
print("Rank:", tf.rank(scalar).numpy())  # Output: 0
print("Shape:", scalar.shape)  # Output: ()
```

```
Scalar: tf.Tensor(5, shape=(), dtype=int32)
Rank: 0
Shape: ()
```

Task 2.

2. Vector (Rank-1 Tensor)

A vector is a 1-dimensional array of values.

Example:

```
Shape: ()
```

```
[2]: # Create a vector tensor
vector = tf.constant([1, 2, 3, 4, 5])

print("Vector:", vector)
print("Rank:", tf.rank(vector).numpy())  # Output: 1
print("Shape:", vector.shape)  # Output: (5,)
```

```
Vector: tf.Tensor([1 2 3 4 5], shape=(5,), dtype=int32)
Rank: 1
Shape: (5,)
```

Task 3.

3. Matrix (Rank-2 Tensor)

A matrix is a 2-dimensional array of values.

Example:

```
[3]: # Create a matrix tensor
matrix = tf.constant([[1, 2, 3], [4, 5, 6]])

print("Matrix:", matrix)
print("Rank:", tf.rank(matrix).numpy())  # Output: 2
print("Shape:", matrix.shape)  # Output: (2, 3)
```

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

Task 4.

4. Rank-3 Tensor

A rank-3 tensor is a 3-dimensional array of values.

Example:

```
[4]: # Create a rank-3 tensor
     rank3_tensor = tf.constant([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

     print("Rank-3 Tensor:", rank3_tensor)
     print("Rank:", tf.rank(rank3_tensor).numpy())  # Output: 3
     print("Shape:", rank3_tensor.shape)  # Output: (2, 2, 2)
```

```
Rank-3 Tensor: tf.Tensor(
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]], shape=(2, 2, 2), dtype=int32)
Rank: 3
Shape: (2, 2, 2)
```

Task 5.

5. Operations on Tensors

Tensors support various mathematical operations, such as addition, multiplication, and reshaping.

Example: Tensor Addition

```
[5]: # Create two tensors
     tensor1 = tf.constant([[1, 2], [3, 4]])
     tensor2 = tf.constant([[5, 6], [7, 8]])

     # Add the tensors
     result = tf.add(tensor1, tensor2)

     print("Result of Addition:", result)
```

```
Result of Addition: tf.Tensor(
[[ 6  8]
 [10 12]], shape=(2, 2), dtype=int32)
```

Task 6.

Example: Reshaping a Tensor

```
[6]: # Reshape a tensor
original_tensor = tf.constant([1, 2, 3, 4, 5, 6])
reshaped_tensor = tf.reshape(original_tensor, (2, 3))

print("Reshaped Tensor:", reshaped_tensor)
```

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

2. Creating Variables

Variables are created using **tf.Variable**.

A variable allows you to add such parameters or node to the graph that are trainable i.e. the value can be modified over a period of a time. Variables are defined by providing their initial value and type as shown below:

```
1 var = tf.Variable( [0.4], dtype = tf.float32 )
```

```
[7]: import tensorflow as tf

# Create a variable with an initial value
initial_value = tf.constant([1, 2, 3])
variable = tf.Variable(initial_value)

print("Variable:", variable)
print("Value of variable:", variable.numpy())
```

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

2. Updating Variables

Variables can be updated using the assign method.

[8]:
```
# Create a variable
variable = tf.Variable([1, 2, 3])

# Assign a new value to the variable
variable.assign([4, 5, 6])
print("Updated variable:", variable.numpy())

# Add to the variable
variable.assign_add([1, 1, 1])
print("After adding:", variable.numpy())

# Subtract from the variable
variable.assign_sub([2, 2, 2])
print("After subtracting:", variable.numpy())
```

```
Updated variable: [4 5 6]
After adding: [5 6 7]
After subtracting: [3 4 5]
```

3. Using Variables in Mathematical Operations

Variables can be used in mathematical operations just like tensors.

[9]:
```
# Create variables
a = tf.Variable(5.0)
b = tf.Variable(3.0)

# Perform operations
result = a + b
print("Result of addition:", result.numpy())

result = a * b
print("Result of multiplication:", result.numpy())
```

```
Result of addition: 8.0
Result of multiplication: 15.0
```

Create a Function.

Create a TensorFlow function called add_tensors that takes two tensors as input and returns their sum. Test the function with the following tensors:

[17]:
```
import tensorflow as tf

# Define the function
def add_tensors(tensor1, tensor2):
    # Perform addition
    result = tensor1 + tensor2
    return result

# Test the function
tensor1 = tf.constant([1, 2, 3], dtype=tf.float32)
tensor2 = tf.constant([4, 5, 6], dtype=tf.float32)

# Call the function
sum_result = add_tensors(tensor1, tensor2)

# Print the result
print("Sum of tensors:", sum_result.numpy())
```

```
Sum of tensors: [5. 7. 9.]
```

What is NumPy?

- NumPy (Numerical Python) is a Python library used for numerical computations.
- It provides support for arrays, matrices, and many mathematical functions.
- It is the foundation for many scientific computing libraries in Python, such as Pandas, SciPy, and Matplotlib.

Why Use NumPy?

- Efficient handling of large datasets.
- Faster computations compared to native Python lists.
- Provides a wide range of mathematical operations.

Creating NumPy Arrays

- From a Python List:

[18]:
```
import numpy as np
array = np.array([1, 2, 3, 4, 5])
print(array)
```

```
[1 2 3 4 5]
```

Array with Random Values:

[19]:
```
random_array = np.random.rand(3, 3)  # 3x3 array of random values between 0 and 1
print(random_array)
```

```
[[0.75399175 0.14889861 0.2960161 ]
 [0.63106933 0.67605598 0.05631214]
 [0.45797115 0.75827213 0.08717853]]
```

Reshaping Arrays:

[20]:
```
array = np.arange(6)
reshaped_array = array.reshape(2, 3)
print("Reshaped Array:\n", reshaped_array)
```

```
Reshaped Array:
 [[0 1 2]
 [3 4 5]]
```

. Indexing and Slicing

Indexing:

[21]:
```
array = np.array([10, 20, 30, 40, 50])
print("First element:", array[0])
print("Last element:", array[-1])
```

```
First element: 10
Last element: 50
```

Slicing:

```
[22]: print("First three elements:", array[:3])
      print("Elements from index 2 to 4:", array[2:5])

      First three elements: [10 20 30]
      Elements from index 2 to 4: [30 40 50]
```

Lab TASKS

Task 1.

- Create a TensorFlow tensor with random float values. Calculate the sum, mean, and maximum of the tensor.

  Hint: Use tf.reduce_sum(), tf.reduce_mean(), tf.reduce_max().

- Create a 3D TensorFlow tensor. Access and print a specific element, a 2D slice, and a 1D slice.

  Hint: Use standard Python indexing and slicing syntax.

TASK 2

Create a TensorFlow function called tensor operations that takes two tensors as input and performs the following arithmetic operations:

1. Addition (tensor1 + tensor2)
2. Subtraction (tensor1 - tensor2)
3. Multiplication (tensor1 * tensor2)
4. Division (tensor1 / tensor2)

The function should return the results of these operations as a dictionary with the following keys:

- "addition"
- "subtraction"
- "multiplication"
- "division"

Test your function with the following tensors:

tensor1 = [10, 20, 30]
tensor2 = [5, 10, 15]

Task 3.

1. Create a NumPy array with random float values. Sort the array in ascending and descending order.

Hint: Use np.sort() and array slicing.

2. Create a 2x2 NumPy matrix. Calculate its transpose and inverse.

Hint: Use np.transpose() and np.linalg.inv().

3. Create a 2D NumPy array. Calculate the sum of each row, the mean of each column, and the standard deviation of the entire array.

Note :

Dear Students,

Please note the following instructions regarding weekly Lab task submissions:

1. **Account Creation**: Create accounts on Kaggle and GitHub using your section and roll number details.
2. **Weekly Tasks**: Submit assigned tasks weekly on both Kaggle and GitHub. Include detailed documentation and code files.
3. **Assessment**: Marks will be based on tasks published on Kaggle and GitHub. Submit tasks on time to avoid penalties.

This initiative aims to enhance your practical skills in data science. For queries, reach out to me.