

**CCS355 NEURAL NETWORKS AND DEEP LEARNING LAB**

**LIST OF EXPERIMENTS**

1. Implement simple vector addition in Tensor Flow.
2. Implement a regression model in Keras.
3. Implement a perceptron in Tensor Flow/Keras Environment.
4. Implement a Feed-Forward Network in Tensor Flow/Keras
5. Implement an Image Classifier using CNN in Tensor Flow/Keras.
6. Improve the Deep learning model by fine tuning hyper parameters.
7. Implement a Transfer Learning concept in Image Classification.
8. Using a pre trained model on Keras for Transfer Learning.
9. Perform Sentiment Analysis using RNN.
10. Implement an LSTM based Auto encoder in Tensor Flow/Keras.

**ADDITIONAL EXPERIMENTS**

1. Train a Deep learning model to classify a given image using pre trained model.
2. Recommendation system from sales data using Deep Learning.
3. Implement Object Detection using CNN.
4. Implement any simple Reinforcement Algorithm for an NLP problem.

**EXP.NO:0 DATE: 14/2/2024**

**BASIC OF TENSERFLOW**

**Aim:** To study the basic of tensor flow along python programs using Colab laboratory.

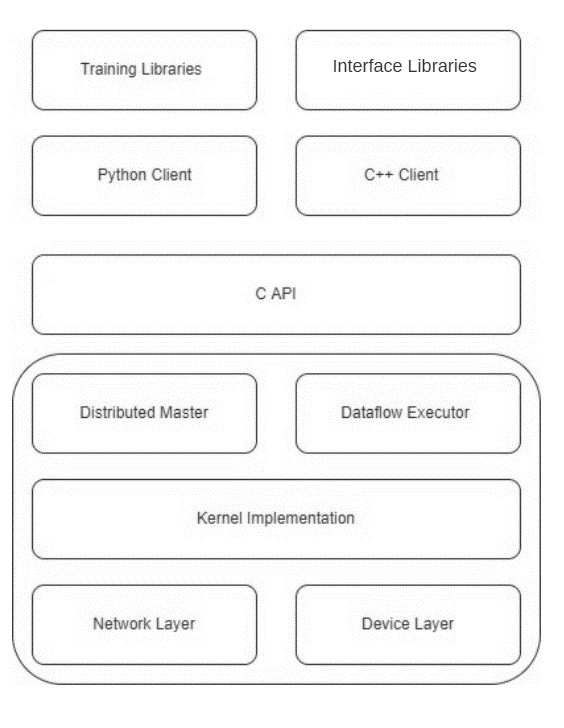
**Questions:**

1. Architecture of tensor flow
2. How to convert Numpy array to tensor flow
3. Matrix using tensor flow

**Description:**

1. **Architecture of Tensor Flow**

* TensorFlow is an end-to-end open-source machine learning platform developed by Google.
* It is scalable and flexible, running on various devices like data centers and mobile phones.
* Computation in TensorFlow is represented by a directed graph with nodes as operations/functions and edges as input/output flows.
* Tensors, multi-dimensional arrays, serve as inputs and outputs in TensorFlow.
* The architecture includes device and network layers, kernel implementations, distributed components, and API layers in C.



* Advanced features include custom operations, custom gradients, TensorFlow Eager Execution, TensorFlow Keras, TensorFlow Probability, and TensorFlow Model Optimization.
* TensorFlow is widely used for image classification, natural language processing, time-series analysis, generative models, reinforcement learning, and anomaly detection.

1. **Convert Numpy array to tensor flow**

The syntax for the `tf.convert\_to\_tensor()` method in TensorFlow is as follows:

**tf.convert\_to\_tensor(value, dtype=None, dtype\_hint=None, name=None)**

* value: The input object to be converted into a tensor. It can be a NumPy array, Python list, scalar value, etc. This parameter is mandatory.
* dtype: (Optional) The data type of the elements in the resulting tensor. If not specified, the data type is inferred from the type of the value. It allows to explicitly set the data type for the tensor.
* dtype\_hint: (Optional) When dtype is None, dtype\_hint is an optional preferred data type for the resulting tensor. It serves as a hint to TensorFlow about the preferred data type but has no effect if the conversion is not possible. This parameter is useful when you want to suggest a preferred data type without enforcing it.
* name: (Optional) An optional name for the resulting tensor. If a new tensor is produced during the conversion, this name can be used to identify it in the TensorFlow graph.

The tf.convert\_to\_tensor() function is commonly used to convert various types of objects, including NumPy arrays, into TensorFlow tensors. The resulting tensor can then be seamlessly integrated into TensorFlow operations and workflows.

1. **Matrix using Tensorflow**

The syntax for creating a matrix using TensorFlow's `tf.constant()` method is as follows:

**tf.constant(value, dtype=None, shape=None, name='Const', verify\_shape=False)**

* value: The values to be used for initializing the constant tensor. In the context of creating a matrix, this would be a nested list representing the matrix elements.
* dtype: (Optional) The data type of the elements in the tensor. If not specified, the data type is inferred from the type of the `value`.
* shape: (Optional) The shape of the tensor. For creating a matrix, you can specify the shape as a tuple representing the number of rows and columns.
* name: (Optional) An optional name for the operation.
* verify\_shape: (Optional) A boolean that controls whether to check if the shape of the provided value matches the specified shape argument. If set to `True` and the shapes do not match, a `ValueError` will be raised.

**Program:**

1. **Example 1:**

# import packages

import tensorflow as tf

import numpy as np

#create numpy\_array

numpy\_array = np.array([[1,2],[3,4]])

# convert it to tensorflow

tensor1 = tf.convert\_to\_tensor(numpy\_array)

print(tensor1)

**Output:**

tf.Tensor(

[[1 2]

[3 4]], shape=(2, 2), dtype=int64)

**Example 2:**

# import packages

import tensorflow as tf

import numpy as np

# create numpy\_array

numpy\_array = np.array([[1, 2], [3, 4]])

# convert it to tensorflow

tensor1 = tf.Variable(numpy\_array, dtype=float, name='tensor1')

tensor1

**Output:**

<tf.Variable 'tensor1:0' shape=(2, 2) dtype=float32, numpy=

array([[1., 2.],

[3., 4.]], dtype=float32)>

1. **Matrix using Tensorflow**

import tensorflow as tf

# Define a 2x3 matrix

matrix\_data = [[1, 2, 3],

               [4, 5, 6]]

# Create a TensorFlow constant tensor from the matrix data

matrix\_tensor = tf.constant(matrix\_data, dtype=tf.int32, shape=(2, 3), name='MyMatrix')

# Print the TensorFlow matrix tensor

print("TensorFlow Matrix:")

print(matrix\_tensor)

**Output:**

TensorFlow Matrix:

tf.Tensor(

[[1 2 3]

[4 5 6]], shape=(2, 3), dtype=int32)

**Result:**

Thus the python program using tensor flow for above experiment is successfully executed and verified.

**EXP.NO:1 DATE: 14/2/2024**

**Aim:** To Implement simple vector addition in Tensor Flow using Colab laboratory.

**Questions:**

1. Scalar
2. **Checking the dimensions of scalar**
3. Vector
4. **Checking the dimensions of vector**
5. **Matrix**
6. Mathematical**Operation**
   * 1. Addition
     2. Subtraction
     3. Multiplication
     4. Division
     5. Transpose
     6. Dot Product

**Algorithm:**

1. Start Colab:

a. Open Google Colab in your web browser.

b. Create a new notebook.

2. Scalar:

a. Import TensorFlow.

b. Define a scalar using `tf.constant()`.

3. Checking the Dimensions of Scalar:

a. Use `tf.rank()` to determine the rank of the scalar.

b. If `tf.rank()` returns 0, it is a scalar.

4. Vector:

a. Define a vector using `tf.constant()` with a 1D array.

5. Checking the Dimensions of Vector:

a. Use `tf.rank()` to determine the rank of the vector.

b. If `tf.rank()` returns 1, it is a vector.

6. Matrix:

a. Define a matrix using `tf.constant()` with a 2D array.

7. Mathematical Operations:

i. Addition:

a. Use `tf.add()` for element-wise addition.

ii. Subtraction:

a. Use `tf.subtract()` for element-wise subtraction.

iii. Multiplication:

a. Use `tf.multiply()` for element-wise multiplication.

iv. Division:

a. Use `tf.divide()` for element-wise division.

v. Transpose:

a. Use `tf.transpose()` for matrix transposition.

vi. Dot Product:

a. Use `tf.matmul()` for matrix multiplication.

8. Run the Code:

a. Execute each cell in the Colab notebook to see the results.

**Program:**

Inport Tensorflow Import tensorflow as tf

scalartf.constant(42.0)

#Checking the Dimensions of Scalar

Is.scalartf.rank(scalar).numpy () 0

Display Result

print("Is Scalar: (is\_scalar)")

Is Scalar: True

#Vector vector tf.constant \ [1, 2, 3]

Checking the Dimensions of Vector Is vector tf.rank(vector).numpy() 1

Display Rasult print("Is Vector: ( 1 s \ ) )")

Is Vector: True

#Matrix matrix tf.constant (\ [1, 2] , (3, 411)

a Mathematical Operations Addition add\_resulttf.add(vector, vector)

Display Result

print(f"Addition Result: (add\_result.numpy())")

Addition Result: [2 \* 4f]

Mathematical Operations Subtraction sub result tf. subtract (vector, vector) sub result.numpy() Display the result array ( [0, 0, 0] , dtype-int32)

Mathematical Operations Multiplication mul\_resulttf.multiply(natrix, 2) mul\_result.numpy() # Display the result

array([[2, 4], [6, 811 , dtype=int32)

Mathematical Operations Division div result tf.divide(matrix, 2) div result.numpy() # Display the result

array t [[0.5,1.], (1.5, 2, 11)

Mathematical Operations Transpose transpose result tf.transpose(matrix) transpose\_result.numpy() Display the result

array (\ \{1, 3\} [2, 4] /, dtype-int32)

#Mathematical Operations Dot Product

dot product\_result t = t (eatrix, tf.transpose(matrix))

dot product\_result.numpy() Display the result

array ( 11, 5, 11 \ , [11, 25)], dtype-Int32)

**Result:**

Thus the python program using tensor flow for about mathematical operations is successfully executed and verified.