Experiment 1

Title :- Installation of Tensorflow & Keras (Tensorflow (v1.0.0), TFLearn, Keras, and many other pre-installed python libraries (Numpy, pandas).

Aim:- To implement Installation of Tensorflow, Keras, TFLearn, Keras, and many other pre-installed python libraries (Numpy, pandas).

1) **TensorFlow**

TensorFlow is a popular framework of machine learning and deep learning. It is a free and open-source library which is released on 9 November 2015 and developed by Google Brain Team. It is entirely based on Python programming language and use for numerical computation and data flow, which makes machine learning faster and easier.

TensorFlow can train and run the deep neural networks for image recognition, handwritten digit classification, recurrent neural network, word embedding, natural language processing, video detection, and many more. TensorFlow is run on multiple CPUs or GPUs and also mobile operating systems.

The word TensorFlow is made by two words, i.e., Tensor and Flow

- Tensor is a multidimensional array
- Flow is used to define the flow of data in operation.

TensorFlow in our system in 2 ways:

1. Through pip (Python package library)
2. Through Anaconda Navigator (conda)

2) **Keras**

Keras is an open-source high-level Neural Network library, which is written in Python is capable enough to run on Theano, TensorFlow, or CNTK. It was developed by one of the Google engineers, Francois Chollet. It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks. It not only supports Convolutional Networks and Recurrent Networks individually but also their combination.

Features : Keras leverages various optimization techniques to make high level neural network API easier and more performant. It supports the following features –

- Consistent, simple and extensible API.
- Minimal structure - easy to achieve the result without any frills.
- It supports multiple platforms and backends.
- It is user friendly framework which runs on both CPU and GPU.
- Highly scalability of computation.

Keras allows you to switch between different back ends. The frameworks supported by Keras are:

Tensorflow, Theano, PlaidML, MXNet, CNTK

Keras depends on the following python libraries.

- Numpy
- Pandas
- Scikit-learn
- Matplotlib
- Scipy
- Seaborn

Step 1) Installing Keras.

```
C:\Users\acer>pip install keras
Requirement already satisfied: keras in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (2.11.0)
```

Step 2) Installing TensorFlow & related library.

```
C:\Users\acer>pip install TensorFlow
Requirement already satisfied: TensorFlow in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (2.11.0)
Requirement already satisfied: tensorflow-intel==2.11.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from TensorFlow) (2.11.0)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (23.1.21)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (3.8.0)
Requirement already satisfied: libclang>=13.0.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (15.0.6.1)
Requirement already satisfied: numpy>=1.20 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (1.24.3)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (3.3.0)
Requirement already satisfied: packaging in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (23.1)
Requirement already satisfied: protobuf<3.20,>=3.9.2 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (3.19.6)
Requirement already satisfied: setuptools in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (63.2.0)
Requirement already satisfied: six>=1.12.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (2.2.0)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (4.6.3)
Requirement already satisfied: wrapt>=1.11.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (1.53.0)
Requirement already satisfied: tensorboard<2.12,>=2.11 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (2.11.2)
Requirement already satisfied: tensorflow-estimator<2.12,>=2.11.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (2.11.0)
Requirement already satisfied: keras<2.12,>=2.11.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (2.11.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.11.0->TensorFlow) (0.30.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from astunparse>=1.6.0->tensorflow-intel==2.11.0->TensorFlow) (0.38.4)
Requirement already satisfied: google-auth<3,>=1.6.3 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->TensorFlow) (2.16.3)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->TensorFlow)
(0.4.6)
Requirement already satisfied: markdown>=2.6.8 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->TensorFlow) (3.4.1)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->TensorFlow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->TensorF
low) (0.6.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->TensorFlow) (1.
8.1)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->TensorFlow) (2.3.4)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0-
>TensorFlow) (5.3.1)
Requirement already satisfied: pyasn1-modules>=0.2.1 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->
TensorFlow) (0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->TensorFl
ow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.12,>=2.11->tensorflow-i
ntel==2.11.0->TensorFlow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0-
>TensorFlow) (3.1.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->TensorFlow)
 (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->Tenso
rFlow) (2.0.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\acer\appdata\local\programs\python\python310\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->Tenso
```

Conclusion

Successful installation of essential frameworks like Tensorflow, TFLearn, and Keras, along with a host of pre-installed Python libraries such as Numpy and Pandas, lays a robust foundation for our experiments.

Experiment 2

Title :- Data Manipulation (Numpy library) Operations Broadcasting Indexing and slicing.

Aim:- To Data Manipulation (Numpy library) Operations Broadcasting Indexing and slicing.

Data manipulation using the NumPy library is a fundamental skill in scientific computing and data analysis with Python. NumPy provides powerful array operations, broadcasting, indexing, and slicing capabilities that make it efficient and convenient for handling large datasets.

Step 1) Install NumPy library

```
In [1]: pip install numpy

        Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (1.20.1)
```

Step 2) Creating NumPy Arrays:

You can create a NumPy array using the numpy.array() function or other specialized functions like numpy.zeros(), numpy.ones(), and numpy.random.

```
In [14]: import numpy as np

         # Create a NumPy array from a list
         arr = np.array([1, 2, 3, 4, 5])
         print("Array list \n",arr)

         # Create an array of zeros with shape (3, 4)
         zeros_array = np.zeros((3, 4))
         print("Array list zeros \n",zeros_array)

         # Create an array of ones with shape (2, 3)
         ones_array = np.ones((2, 3))
         print("Array list ones \n",ones_array)

         # Create a random array with shape (2, 2)
         random_array = np.random.rand(2, 2)
         print("Random Array list \n",random_array)

Array list
 [1 2 3 4 5]
Array list zeros
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
Array list ones
 [[1. 1. 1.]
 [1. 1. 1.]]
Random Array list
 [[0.74598031 0.78585436]
 [0.38974045 0.17106252]]
```

Step 3) Broadcasting:

NumPy allows you to perform operations on arrays with different shapes by broadcasting the smaller array to match the shape of the larger array.

```
In [21]: # Broadcasting example
         a = np.array([1, 2, 3])
         scalar = 2
         result = a * scalar  # Broadcasting the scalar to the array [2, 4, 6]
         print("Broadcasting", result)

         Broadcasting [2 4 6]
```

Step 4) Array Operations:

You can perform various operations on NumPy arrays, such as element-wise arithmetic, matrix operations, and statistical functions.

```
In [20]: # Element-wise arithmetic operations
         a = np.array([1, 2, 3])
         b = np.array([4, 5, 6])
         print("addition",a + b)
         print("subtraction",a - b)
         print("multiplication",a * b)
         print("division",a / b)

         # Matrix operations
         matrix_a = np.array([[1, 2], [3, 4]])
         matrix_b = np.array([[5, 6], [7, 8]])
         matrix_product = np.dot(matrix_a, matrix_b)  # Matrix multiplication
         print("Matrix Product \n",matrix_product)

         # Statistical functions
         data = np.array([10, 15, 20, 25, 30])
         print(data)
         mean = np.mean(data)
         print("Mean",mean)
         median = np.median(data)
         print("Median",median)
         std_deviation = np.std(data)
         print("Std Deviation",std_deviation)

         addition [5 7 9]
         subtraction [-3 -3 -3]
         multiplication [ 4 10 18]
         division [0.25 0.4  0.5 ]
         Matrix Product
          [[19 22]
          [43 50]]
         [10 15 20 25 30]
         Mean 20.0
         Median 20.0
         Std Deviation 7.0710678118654755
```

Step 5) Indexing and Slicing:

You can access specific elements in a NumPy array using indexing. Slicing allows you to extract a portion of an array.

```
In [23]: arr = np.array([10, 20, 30, 40, 50])

         # Accessing elements using indexing
         element = arr[0]  # Access the first element (10)
         print("[0] ",element)
         element_2 = arr[-1]  # Access the last element (50)
         print("[-1] ",element_2)

         [0]  10
         [-1]  50
```

```
In [22]: arr = np.array([10, 20, 30, 40, 50])

         # Slicing
         slice_1 = arr[1:4]  # Get elements from index 1 to index 3 (20, 30, 40)
         print("arr[1:4] ",slice_1)
         slice_2 = arr[:3]  # Get elements from the beginning up to index 2 (10, 20, 30)
         print("arr[:3] ",slice_2)
         slice_3 = arr[2:]  # Get elements from index 2 to the end (30, 40, 50)
         print("arr[2:] ",slice_3)

         arr[1:4]  [20 30 40]
         arr[:3]  [10 20 30]
         arr[2:]  [30 40 50]
```

Conclusion

Successful installation of essential library, data Manipulation (Numpy library) Operations Broadcasting Indexing and slicing.

Experiment 3

Title :- Data Preprocessing Reading the Dataset Handling Missing Data Conversion to the Tensor Format.

Aim:- To data Preprocessing Reading the Dataset Handling Missing Data Conversion to the Tensor Format.

Data preprocessing is a crucial step in the data analysis and machine learning pipeline. It involves cleaning, transforming, and preparing the data to be used in machine learning models. Let's go through the steps of data preprocessing, including reading the dataset, handling missing data, and converting the data into the tensor format.

Step 1) Reading the Dataset:

To read a dataset into Python, you can use various libraries such as NumPy, Pandas, or TensorFlow (tf.data) depending on the format of your data. Pandas is a popular choice for handling tabular data.

```
[1]: import pandas as pd

     # Read a CSV file into a Pandas DataFrame
     df = pd.read_csv('your_dataset.csv')
     print(df)
```

```
      Name   Age  Gender  Salary
0    Alice  25.0  Female   50000
1      Bob  30.0    Male   60000
2  Charlie   NaN    Male   45000
3    David  28.0    Male   70000
4      Eva  35.0  Female   55000
```

Step 2) Handling Missing Data:

Missing data is a common issue in datasets. You need to handle missing data appropriately to avoid biased or incorrect results in your analysis or model training.

```
[2]: # Check for missing values in the DataFrame
     print(df.isnull().sum())

     # Option 1: Drop rows with missing values
     #df = df.dropna()

     # Option 2: Fill missing values with a specific value (e.g., mean, median, or custom value)
     mean_age = df['Age'].mean()
     df['Age'].fillna(mean_age, inplace=True)
     #df = pd.DataFrame(df)

     df['Gender'] = df['Gender'].map({'Male':1 , 'Female':0})
     print(df)
```

```
Name      0
Age       1
Gender    0
Salary    0
dtype: int64
      Name   Age  Gender  Salary
0    Alice  25.0       0   50000
1      Bob  30.0       1   60000
2  Charlie  29.5       1   45000
3    David  28.0       1   70000
4      Eva  35.0       0   55000
```

Step 3) Conversion to the Tensor Format:

Machine learning models, especially those built with TensorFlow, often require data in tensor format. A tensor is a multi-dimensional array.

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

     # Extract features and labels from the DataFrame
     X = df[['Age', 'Gender']].values
     y = df['Salary'].values

     # Convert to TensorFlow tensors
     X_tensor = tf.constant(X, dtype=tf.float32)
     y_tensor = tf.constant(y, dtype=tf.int32)

     # Alternatively, you can use NumPy arrays directly as TensorFlow tensors
     X_tensor = tf.convert_to_tensor(X, dtype=tf.float32)
     y_tensor = tf.convert_to_tensor(y, dtype=tf.int32)

     print("\n X tensor \n",X_tensor)
     print("\n Y tensor\n",y_tensor)
```

```
 X tensor
 tf.Tensor(
[[25.   0. ]
 [30.   1. ]
 [29.5  1. ]
 [28.   1. ]
 [35.   0. ]], shape=(5, 2), dtype=float32)

 Y tensor
 tf.Tensor([50000 60000 45000 70000 55000], shape=(5,), dtype=int32)
```

Conclusion

Successful installation of essential library, Data Preprocessing Reading the Dataset Handling Missing Data Conversion to the Tensor Format.

Experiment 4

Title :- **Linear Algebra Tensors Tensor arithmetic Implementing matrix multiplication.**

Aim:- To Linear Algebra Tensors Tensor arithmetic Implementing matrix multiplication.

Linear algebra plays a fundamental role in machine learning, and tensors are essential data structures used for representing multi-dimensional arrays. In this section, I'll cover basic tensor operations, tensor arithmetic, and how to implement matrix multiplication using NumPy.

A tensor is a generalization of vectors and matrices and is easily understood as a multidimensional array. A vector is a one-dimensional or first order tensor and a matrix is a two-dimensional or second order tensor.

Step 1) Tensors and Tensor Arithmetic:

Tensors are multi-dimensional arrays. In the context of linear algebra, vectors are 1-dimensional tensors, matrices are 2-dimensional tensors, and so on.

In [7]:
```
import numpy as np

# Creating tensors (arrays) using NumPy
# 1-D tensor (vector)
vector = np.array([1, 2, 3])

# 2-D tensor (matrix)
matrix = np.array([[1, 2], [3, 4]])

# 3-D tensor
tensor_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

# Tensor arithmetic (element-wise operations)
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

addition = a + b
print("addition ", addition)
subtraction = a - b
print("subtraction ", subtraction)
elementwise_multiplication = a * b
print("multiplication ", elementwise_multiplication)
elementwise_division = a / b
print("division ", elementwise_division)
```

```
addition  [5 7 9]
subtraction  [-3 -3 -3]
multiplication  [ 4 10 18]
division  [0.25 0.4  0.5 ]
```

Step 2) Implementing Matrix Multiplication:

Matrix multiplication is a fundamental operation in linear algebra. In NumPy, you can use the `dot()` function or `@` operator to perform matrix multiplication.

```
In [9]: import numpy as np

        # Matrix multiplication using dot() function
        matrix_a = np.array([[1, 2], [3, 4]])
        matrix_b = np.array([[5, 6], [7, 8]])

        # Option 1: Use dot() function
        matrix_product = np.dot(matrix_a, matrix_b)
        print("Dot \n", matrix_product)

        # Option 2: Use @ operator (Python 3.5+)
        matrix_product = matrix_a @ matrix_b
        print("@ operator\n ", matrix_product)
```

```
Dot
 [[19 22]
 [43 50]]
@ operator
  [[19 22]
 [43 50]]
```

Conclusion

Successful Implement McCulloch Pitts neural network using Tensorflow.

Experiment 5

Title :- Implement McCulloch Pitts neural network using Tensorflow.

Aim:- To Implement McCulloch Pitts neural network using Tensorflow.

McCulloch Pitts neural network:

The McCulloch-Pitts neuron, also known as the McCulloch-Pitts model, is a simplified mathematical model of a biological neuron. It was introduced by Warren McCulloch and Walter Pitts in 1943. The McCulloch-Pitts neuron is a foundational concept in neural network theory and serves as the basis for understanding more complex artificial neural networks.

The McCulloch-Pitts neuron operates using binary logic (0 or 1) and consists of the following components:

1. Inputs: Each neuron receives binary inputs (0 or 1) from various sources, which represent the incoming signals from other neurons or external stimuli.

2. Weights: Each input is assigned a weight that represents the strength of the connection between the input and the neuron. The weights can be positive or negative.

3. Threshold: The neuron has a threshold value that determines the level of activation required for it to fire an output signal.

4. Activation Function: The McCulloch-Pitts neuron applies a simple activation function. It sums the products of inputs and their corresponding weights and compares this sum to the threshold. If the sum is greater than or equal to the threshold, the neuron outputs 1; otherwise, it outputs 0.

Mathematically, the output of the McCulloch-Pitts neuron can be represented as:

Output = 1 if $\Sigma(w_i * x_i) \geq \theta$ (threshold)
Output = 0 if $\Sigma(w_i * x_i) < \theta$

This basic neuron model can be used to create logical circuits and perform simple binary computations. It serves as a building block for more complex neural network architectures, such as perceptrons and multi-layer perceptrons, which have enhanced capabilities for pattern recognition and classification.

Single Layer Neural Network:

A single-layer neural network, often referred to as a single-layer perceptron, is a type of artificial neural network architecture that consists of only one layer of interconnected neurons. It is a basic and elementary form of a neural network, and it can be used for simple linear classification tasks.

Here's how a single-layer perceptron works:

1. Inputs:Similar to the McCulloch-Pitts neuron, a single-layer perceptron receives inputs (features) from the external world. Each input is associated with a weight, which represents the strength of the connection between the input and the neuron.

2. Weights: The weights are multiplied by their corresponding inputs, and the resulting weighted inputs are summed up. Mathematically, this can be represented as:

Weighted Sum = Σ(w_i * x_i)

3. Bias:In addition to weights and inputs, a bias term is often included. The bias is a constant value that is added to the weighted sum before applying the activation function. It allows the network to control the threshold at which the neuron fires.

4. Activation Function:The weighted sum (including the bias) is then passed through an activation function. Common activation functions include the step function (similar to McCulloch-Pitts threshold), sigmoid function, or ReLU (Rectified Linear Unit). The activation function determines the output of the neuron based on the computed value.

Output = Activation_Function(Weighted Sum + Bias)

5. Output: The final output of the single-layer perceptron is the result of applying the activation function to the weighted sum of inputs.

Output McCulloch Pitts neural network

```
class McCullochPittsNeuron:
    def __init__(self, weights, threshold):
        self.weights = weights
        self.threshold = threshold

    def activate(self, inputs):
        if len(inputs) != len(self.weights):
            raise ValueError("Number of inputs should match the number of weights
        result = sum([inputs[i] * self.weights[i] for i in range(len(inputs))])
        return 1 if result >= self.threshold else 0

def main():
    # Define the weights and threshold for the AND gate
    and_gate_weights = [1, 1]
    and_gate_threshold = 2

    # Create a McCulloch-Pitts neuron for the AND gate
    and_gate_neuron = McCullochPittsNeuron(and_gate_weights, and_gate_threshold)

    # Test the AND gate
    inputs = [
        [0, 0],
        [0, 1],
        [1, 0],
        [1, 1]
    ]
    for input_pair in inputs:
        output = and_gate_neuron.activate(input_pair)
        print(f"{input_pair} => {output}")

if __name__ == "__main__":
    main()
```

```
[0, 0] -> 0
[0, 1] -> 0
[1, 0] -> 0
[1, 1] -> 1
```

Output of single-layer perceptron

```python
import numpy as np

class SingleLayerPerceptron:
    def __init__(self, input_size):
        self.weights = np.random.rand(input_size)
        self.bias = np.random.rand()
        self.learning_rate = 0.1

    def predict(self, inputs):
        net_input = np.dot(inputs, self.weights) + self.bias
        return 1 if net_input >= 0 else 0

    def train(self, training_data, epochs):
        for epoch in range(epochs):
            errors = 0
            for inputs, label in training_data:
                prediction = self.predict(inputs)
                error = label - prediction
                self.weights += self.learning_rate * error * inputs
                self.bias += self.learning_rate * error
                errors += abs(error)
            if errors == 0:
                print(f"Converged after {epoch + 1} epochs.")
                break

def main():
    # Sample training data for OR gate
    training_data = [
        (np.array([0, 0]), 0),
        (np.array([0, 1]), 1),
        (np.array([1, 0]), 1),
        (np.array([1, 1]), 1)
    ]

    input_size = len(training_data[0][0])
    perceptron = SingleLayerPerceptron(input_size)

    epochs = 100
    perceptron.train(training_data, epochs)

    # Test the trained perceptron
    test_data = [
        np.array([0, 0]),
        np.array([0, 1]),
        np.array([1, 0]),
        np.array([1, 1])
    ]

    print("Testing the perceptron:")
    for inputs in test_data:
        prediction = perceptron.predict(inputs)
        print(f"{inputs} -> {prediction}")

if __name__ == "__main__":
    main()
```

```
Converged after 3 epochs.
Testing the perceptron:
[0 0] -> 0
[0 1] -> 1
[1 0] -> 1
[1 1] -> 1
```

Conclusion

Successful Implement McCulloch Pitts neural network using Tensorflow.

Experiment 6

Title :- Forward pass with matrix multiplication Forward pass with hidden layer (matrix multiplication) Forward pass with matrix multiplication with Keras Forward passes with hidden layer (matrix multiplication) with Keras.

Aim:- To Implement Forward pass with matrix multiplication Forward pass with hidden layer Forward pass with matrix multiplication with Keras Forward passes with hidden layer with Keras.

Overview of implementing forward passes using matrix multiplication and Keras for both a single-layer neural network (perceptron) and a multi-layer neural network (with a hidden layer):

1) Forward Pass with Matrix Multiplication in Keras (Single-Layer Neural Network).

2) Define a Keras model with a single layer: model

3) To perform the forward pass and obtain the output.

4) Forward Pass with Hidden Layer in Keras (Multi-Layer Neural Network):

5) Create input data.

6) Define a Keras model with a hidden layer and an output layer.

7) To perform the forward pass and obtain the output.

In [3]:
```
# 1.Forward Pass with Matrix Multiplication (Without Hidden Layer)

import numpy as np

# Input data (a 3x2 matrix)
input_data = np.array([[1, 2],
                       [3, 4],
                       [5, 6]])

# Weight matrix (a 2x2 matrix)
weights = np.array([[0.1, 0.2],
                    [0.3, 0.4]])

# Forward pass without a hidden layer (matrix multiplication)
output_data = np.dot(input_data, weights)

print("Output without hidden layer:")
print(output_data)
```

```
Output without hidden layer:
[[0.7 1. ]
 [1.5 2.2]
 [2.3 3.4]]
```

In [4]:
```python
# 2. Forward Pass with Hidden Layer (Matrix Multiplication)

import numpy as np

# Input data (a 3x2 matrix)
input_data = np.array([[1, 2],
                       [3, 4],
                       [5, 6]])

# Weight matrices (2 hidden units, 2 input features)
weights_hidden = np.array([[0.1, 0.2],
                           [0.3, 0.4]])

# Weight matrices (2 output units, 2 hidden units)
weights_output = np.array([[0.5, 0.6],
                           [0.7, 0.8]])

# Forward pass with a hidden layer (matrix multiplication)
hidden_layer_output = np.dot(input_data, weights_hidden)
output_data = np.dot(hidden_layer_output, weights_output)

print("Output with hidden layer:")
print(output_data)
```

```
Output with hidden layer:
[[1.05 1.22]
 [2.29 2.66]
 [3.53 4.1 ]]
```

In [5]:
```python
# 3. Forward Pass with Matrix Multiplication using Keras:

import numpy as np
from keras.models import Sequential
from keras.layers import Dense

# Input data (a 3x2 matrix)
input_data = np.array([[1, 2],
                       [3, 4],
                       [5, 6]])

# Define the Keras model
model = Sequential()
model.add(Dense(units=2, input_dim=2, activation='sigmoid'))
model.add(Dense(units=2, activation='sigmoid'))

# Weight matrices will be automatically initialized by Keras

# Perform the forward pass using Keras
output_data = model.predict(input_data)

print("Output with Keras:")
print(output_data)
```

```
1/1 [==============================] - 1s 653ms/step
Output with Keras:
[[0.4551778  0.5146156 ]
 [0.48358917 0.46191478]
 [0.5012354  0.43267027]]
```

In [6]:
```python
# 4. Forward passes with hidden layer (matrix multiplication) with Keras.

import numpy as np
from keras.models import Sequential
from keras.layers import Dense

# Input data (a 3x2 matrix)
input_data = np.array([[1, 2],
                       [3, 4],
                       [5, 6]])

# Define the Keras model
model = Sequential()

# Add a hidden layer with 2 units and input dimension 2, using matrix multipli
model.add(Dense(units=2, input_dim=2, activation='sigmoid'))

# Add the output layer with 2 units, also using matrix multiplication
model.add(Dense(units=2, activation='sigmoid'))

# Weight matrices will be automatically initialized by Keras

# Perform the forward pass using Keras
output_data = model.predict(input_data)

print("Output with hidden layer using Keras:")
print(output_data)
```

```
1/1 [==============================] - 0s 43ms/step
Output with hidden layer using Keras:
[[0.4015603  0.67632186]
 [0.39087415 0.69307435]
 [0.39129204 0.6895925 ]]
```

Conclusion

Successful Implement Forward pass with matrix multiplication Forward pass with hidden layer Forward pass with matrix multiplication with Keras Forward passes with hidden layer.