**Practical 1**

**Matrix Multiplication, Eigen Vectors**

**Aim:**  **Perform Matrix multiplication and finding eigen vectors and eigen values using tenserflow**

**Description:**

**Matrix:**

* In mathematics, a matrix is a structured arrangement of numbers or symbols in rows and columns.
* It serves as a fundamental tool for organizing and manipulating data in various mathematical operations, such as addition, subtraction, multiplication, and inversion.
* Matrices find extensive applications in diverse fields ranging from computer graphics and quantum mechanics to economics and engineering.
* A group of numbers and letters

  Description automatically generated

**Vectors:**

* Vectors are mathematical entities characterized by both magnitude and direction. They are represented geometrically as arrows in a multi-dimensional space, with their length signifying the magnitude and their orientation indicating the direction.
* Vectors play a crucial role in many areas of mathematics, physics, and engineering. They are used to represent forces, velocities, and displacements in physics; define points and directions in geometry; and model quantities such as preferences and probabilities in economics and statistics.
* A close up of words

  Description automatically generated

**Eigen vectors:**

* Eigen vectors are special vectors associated with linear transformations that retain their direction but may be scaled during the transformation process.
* They are characterized by the property that when a linear transformation is applied to them, they are only scaled by a scalar factor, without changing their direction.
* They provide valuable insights into the behavior of linear systems, help analyze stability and equilibrium points in dynamic systems
* 

**Eigen values:**

* Eigen values are scalars associated with eigen vectors that represent the factor by which the corresponding eigen vectors are scaled during a linear transformation. They signify the amount of stretching or compression experienced by the eigen vectors when subjected to the transformation.
* Eigen values are crucial in understanding the behavior of linear systems, stability analysis, and solving differential equations.

**Tensor:**

* A tensor is a mathematical object that generalizes the concept of vectors and matrices to higher dimensions. It represents multidimensional arrays of data, characterized by multiple indices that specify components along different axes or directions.

Tensors find widespread applications in physics, engineering, and computer science, where they are used to describe the properties of physical systems, model complex phenomena, and facilitate computations in various algorithms

**Code**

import tensorflow as tf

print("Matrix Multiplication Demo")

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

print(a)

b=tf.constant([7,8,9,10,11,12],shape=[3,2])

print(b)

c=tf.matmul(a,b)

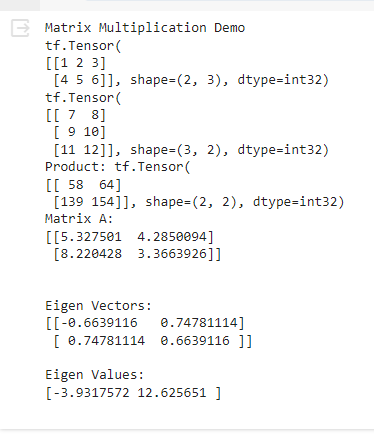
print("Product:",c)

e\_matrix\_A=tf.random.uniform([2,2],minval=3,maxval=10,dtype=tf.float32,name="matrixA")

print("Matrix A:\n{}\n\n".format(e\_matrix\_A))

eigen\_values\_A,eigen\_vectors\_A=tf.linalg.eigh(e\_matrix\_A)

print("Eigen Vectors:\n{}\n\nEigen Values:\n{}\n".format(eigen\_vectors\_A,eigen\_values\_A))



**Code and output:**

import tensorflow as tf

print("Matrix Multiplication Demo")

a=tf.constant([8,3,7,9,1,4],shape=[2,3])

print(a)

b=tf.constant([4,6,3,7,5,1],shape=[3,2])

print(b)

c=tf.matmul(a,b)

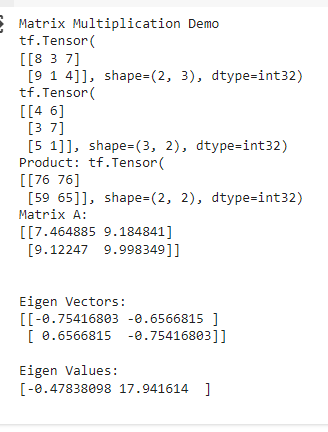
print("Product:",c)

e\_matrix\_A=tf.random.uniform([2,2],minval=3,maxval=10,dtype=tf.float32,name="matrixA")

print("Matrix A:\n{}\n\n".format(e\_matrix\_A))

eigen\_values\_A,eigen\_vectors\_A=tf.linalg.eigh(e\_matrix\_A)

print("Eigen Vectors:\n{}\n\nEigen Values:\n{}\n".format(eigen\_vectors\_A,eigen\_values\_A))



**Pr****actical 2**

**Solving XOR problem using deep feed forward network**

**Aim:** Solving XOR problem using deep feed forward network

**Description:**

A deep feedforward network, also known as a feedforward neural network or multilayer perceptron (MLP), is a foundational architecture in deep learning. It consists of multiple layers of interconnected nodes, with each layer feeding its output forward as input to the next layer. These networks are characterized by their ability to learn complex representations of data through hierarchical feature extraction. They are widely used for supervised learning tasks such as classification and regression, where the input-output mapping is learned from labeled data. Training is typically done using techniques like backpropagation and stochastic gradient descent to minimize a specified loss function.

**Sigmoid Function:**

The sigmoid function is a popular activation function used in neural networks, especially in the output layer for binary classification tasks. It squashes the output of a neuron to a value between 0 and 1, which can be interpreted as a probability.

Mathematically, the sigmoid function is defined as:

\[ f(x) = \frac{1}{1 + e^{-x}} \]

A mathematical equation with numbers and symbols

Description automatically generated

Here's what the sigmoid function does:

- For large positive values of \( x \), \( f(x) \) approaches 1.

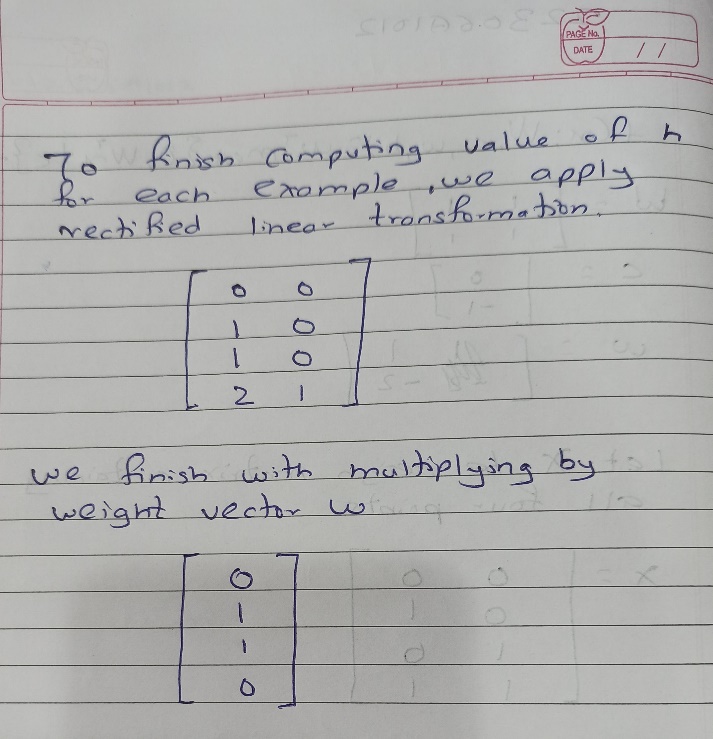
- For large negative values of \( x \), \( f(x) \) approaches 0.

- For \( x = 0 \), \( f(x) \) is exactly 0.5.

The sigmoid function introduces non-linearity to the network, letting it learn from the error and adjust the weights during training. However, it's worth noting that sigmoid can suffer from the vanishing gradient problem, especially in deeper networks, which is why other activation functions like ReLU are often preferred for hidden layers.

**Handwritten:**

A piece of paper with writing on it

Description automatically generated

**Code:**

import numpy as np

from keras.models import Sequential

from keras.layers import Dense

model = Sequential()

model.add(Dense(units = 2, activation = 'relu', input\_dim = 2))

model.add(Dense(units=1, activation = 'sigmoid'))

model.compile(loss = 'binary\_crossentropy', optimizer = 'adam', metrics = 'accuracy')

X = np.array([[0.,0.],[0.,1.],[1.,0.],[1.,1.]])

print("Input data:")

print(X)

y = np.array([0.,1.,1.,0.])

print("\nTarget labels:")

print(y)

model.get\_weights()

model.fit(X,y,epochs = 500)

predictions = model.predict(X)

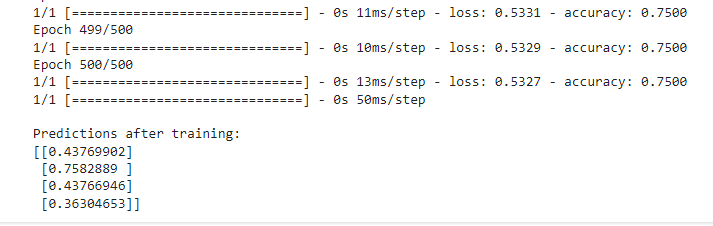
print("\nPredictions after training:")

print(predictions)

**Output:**

A screenshot of a computer code

Description automatically generated



**Learning:**

This code defines a neural network model using Keras to perform binary classification on a dataset with four samples, each containing two features. The model is trained for 500 epochs using binary crossentropy loss and the Adam optimizer. After training, it makes predictions on the input data, yielding the output probabilities of belonging to the positive class.