PRACTICAL NO : 1

Aim : Write a Program to demonstrate following operations

1. Create Vector, Matrix and Tensor
2. Multiplication of two : Vector, Matrix and Tensor
3. Addition of two : Vector, Matrix and Tensor
4. Multiply Matrix with Vector
5. Matrix Dot product and Matrix Inverse

Code :

import numpy as np

import tensorflow as tf

print("Govind Saini")

# Create Vector, Matrix and Tensor

X = np.array([1, 2, 3, 4])

print("Create a Vector:",X)

print("\n")

A = np.array([[1, 2], [3, 4], [5, 6]])

print("Create a Matrix:",A)

print("\n")

Tensor\_A =tf.constant([[1, 2]],dtype=tf.int32)

print("Create a Tensor :", Tensor\_A)

print("\n")

2)

Code:

# Multiplication of two : Vector, Matrix and Tensor

A=np.array([[1, 2],[3, 4],[5, 6]])

print("A = ", A)

print("\n")

B=np.array([[2, 5],[7, 4],[4, 3]])

print("B", B)

print("\n")

C = A\*B

print("Multiplication of two Matrix : ", C)

print("\n")

x= np.array([1, 2, 3, 4])

y=np.array([5, 6, 7, 8])

z=x\*y

print("Multiplication of two Vector:", z)

print("\n")

tensor\_A = tf.constant([[4, 2]], dtype=tf.int32)

print("A :", tensor\_A)

tensor\_B=tf.constant([[7, 4]], dtype=tf.int32)

print("B :", tensor\_B)

tensor\_multiply = tf.multiply(tensor\_A, tensor\_B)

print("Multiplication of two Tensor",tensor\_multiply)

print("\n")

3) code:-

x= np.array([1, 2, 3, 4])

y=np.array([5, 6, 7, 8])

z=x\*y

print("Multiplication of two Vector:", z)

print("\n")

tensor\_A = tf.constant([[4, 2]], dtype=tf.int32)

print("A :", tensor\_A)

tensor\_B=tf.constant([[7, 4]], dtype=tf.int32)

print("B :", tensor\_B)

tensor\_multiply = tf.multiply(tensor\_A, tensor\_B)

print("Multiplication of two Tensor",tensor\_multiply)

print("\n")

4)5)code:

B

# Multiply Matrix with Vector

x = np.array([1, 2, 3, 7, 3, 5, 2])

y=np.array([[1], [3], [5], [7], [8], [8], [2]])

c = x \* y

print("Multiply Matrix with Vector :", c)

print("\n")

# Matrix Dot product and Matrix Inverse

u = [2, 3]

v = [1, 3]

dotproduct = np.dot (u, v)

print("Matrix Dot product :", dotproduct)

print("\n")

A = np.array([[6, 1, 1],

[4, -2, 5],

[2, 8, 7]])

# Calculating the inverse of the matrix

print("Inverse of the matrix :", np.linalg.inv(A))

print("\n")

**Practical 2:**

**Performing matrix multiplication and finding Eigen**

**vectors and Eigen values using TensorFlow**

**code:**

#imports

import tensorflow as tf

print("7\_Govind Saini\n")

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

shape=[2, 3])

print(x)

y=tf.constant([7, 8, 9, 10, 11, 12],

shape=[3, 2])

print(y)

# Multiplying Two matrix

z = tf.matmul(x, y)

print("\n")

print("Multiplying two Matrix :", z)

# Let's see how we can compute the eigen vectors and values from a matrix

e\_matrix\_A = tf.random.uniform(

    [2, 2], minval=3, maxval=10, dtype=tf.float32, name="matrixA")

print("\n")

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

# Calculating the eigen values and vectors using tf.linalg.eigh, if you only want the values you can use eigvalsh

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))

**practical 3**

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

**Code:**

from tflearn import DNN

from tflearn.layers.core import input\_data, fully\_connected

from tflearn.layers.estimator import regression

# Training examples

X = [[0, 0], [0, 1], [1, 0], [1, 1]]

Y = [[0], [1], [1], [0]]

# input layer of size 2

input\_layer = input\_data(shape=[None, 2])

# hidden layer of size 2

hidden\_layer = fully\_connected(input\_layer, 2, activation='relu')

# output layer of size 1

output\_layer = fully\_connected(hidden\_layer, 1, activation='sigmoid')

# use Stohastic Gradient Descent and Binary Crossentropy as loss function

regression = regression(output\_layer, optimizer='Adadelta',

                        loss='categorical\_crossentropy', learning\_rate=5)

model = DNN(regression)

print("\n")

print("7\_Govind Saini")

# fit the model

model.fit(X, Y, n\_epoch=100, show\_metric=True)

# predict all examples

print('Expected:  ', [i[0] > 0 for i in Y])

print('Predicted: ', [i[0] > 0 for i in model.predict(X)])

print("hidden layer", model.get\_weights(

    hidden\_layer.W), model.get\_weights(hidden\_layer.b))

print("output layer", model.get\_weights(

    output\_layer.W), model.get\_weights(output\_layer.b))

|  |  |
| --- | --- |
|  |  |

**practical 4**

# Binary Classification with Sonar Dataset: Baseline

from pandas import read\_csv

from keras.models import Sequential

from keras.layers import Dense

from keras.wrappers.scikit\_learn import KerasClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import StratifiedKFold

# Define the baseline model

def create\_baseline():

    model = Sequential()

    model.add(Dense(60, input\_dim=60, activation='relu'))

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

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

    return model

# Load dataset

dataframe = read\_csv("sonar.csv", header=None)

dataset = dataframe.values

# Split into input (X) and output (Y) variables

X = dataset[:, 0:60].astype(float)

Y = dataset[:, 60]

# Encode class values as integers

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

# Evaluate model with standardized dataset

estimator = KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)

kfold = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=1)

results = cross\_val\_score(estimator, X, encoded\_Y, cv=kfold)

print('\n')

print("7\_Govind Saini \n")

print("Baseline: %.2f%% (%.2f%%)" % (results.mean()\*100, results.std()\*100))