**ASSESMENT-2**

**PROBLEM STATEMENT-** Installing Keras, Tensorflow and Pytorch libraries and making use of them

**SOURCE CODE:**

* Open the folder in the Spyder IDE
* The Command to install

**“pip install keras tensorflow torch torchvision”**

**File Name: program2.py**

# Importing libraries

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import torch

import torch.nn as nn

import torch.optim as optim

# Simple dataset

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

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

# TensorFlow/Keras model

model\_tf = Sequential([

    Dense(8, input\_dim=2, activation='relu'),

    Dense(1, activation='sigmoid')

])

model\_tf.compile(optimizer='adam',

                 loss='binary\_crossentropy',

                 metrics=['accuracy'])

model\_tf.fit(X, y, epochs=1000, verbose=0)

# PyTorch model

class Net(nn.Module):

    def \_\_init\_\_(self):

        super(Net, self).\_\_init\_\_()

        self.fc1 = nn.Linear(2, 8)

        self.fc2 = nn.Linear(8, 1)

    def forward(self, x):

        x = torch.relu(self.fc1(x))

        x = torch.sigmoid(self.fc2(x))

        return x

model\_pt = Net()

criterion = nn.BCELoss()

optimizer = optim.Adam(model\_pt.parameters(), lr=0.01)

X\_torch = torch.tensor(X, dtype=torch.float32)

y\_torch = torch.tensor(y, dtype=torch.float32).view(-1, 1)

for epoch in range(1000):

    optimizer.zero\_grad()

    output = model\_pt(X\_torch)

    loss = criterion(output, y\_torch)

    loss.backward()

    optimizer.step()

# Evaluating models

# TensorFlow/Keras model prediction

print("\nTensorFlow/Keras Model Predictions:")

print(model\_tf.predict(X))

# PyTorch model prediction

print("\nPyTorch Model Predictions:")

with torch.no\_grad():

    model\_pt.eval()

    print(model\_pt(X\_torch).numpy())

**Explanation:**

Here's a breakdown of the code:

1. **Importing Libraries:**

* The code begins by importing necessary libraries including NumPy, TensorFlow, Keras, and PyTorch.

1. **Simple Dataset:**

* A simple dataset for the XOR problem is defined using NumPy arrays. X contains the input features, and y contains the corresponding labels.

1. **TensorFlow/Keras Model:**

* A neural network model is defined using the Sequential API from Keras. It consists of two fully connected (Dense) layers with ReLU activation in the hidden layer and sigmoid activation in the output layer.
* The model is compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy metric.
* The model is trained on the XOR dataset (X, y) using model\_tf.fit() for 1000 epochs.

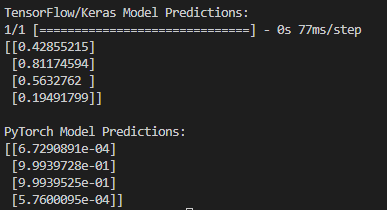
1. **PyTorch Model:**

* A neural network model is defined using PyTorch's nn.Module class. It has two linear (fully connected) layers with ReLU activation in the hidden layer and sigmoid activation in the output layer.
* A custom neural network class Net is defined, where the forward pass is implemented.
* The model is trained using stochastic gradient descent with the Adam optimizer and binary cross-entropy loss function for 1000 epochs.

1. **Evaluating Models:**

* The trained TensorFlow/Keras model and the PyTorch model are evaluated on the XOR dataset, and their predictions are printed out.
* The purpose of this code is to demonstrate how to implement a simple neural network to solve the XOR problem using both TensorFlow/Keras and PyTorch frameworks. It provides a comparison between the two frameworks in terms of ease of use and syntax.

**OUTPUT:**

****