**Ex1 : Tensor Basics and Implementation of Perceptron using Pytorch**

1-Calculate the Determinant of matrix A using Tensors:

def determinant(A):

n = len(A)

matrix = torch.tensor(A)

for i in range(n):

for j in range(i+1,n):

item = matrix[j][i]/matrix[i][i]

for k in range(n):

matrix[j][k]= matrix[j][k] - item \* matrix[i][k]

product = 1.0

for m in range(n):

product \*= matrix[m][m]

return product

import torch

from torch import Tensor

A = torch.tensor([[1,5,1],[12,-1,31],[4,33,2]],dtype = torch.float32)

ans = determinant(A)

ans

2-Demonstrate the Autograd functionality with examples

import torch

from torch.autograd import Variable

a = Variable(torch.Tensor([[1,2],[3,4]]),requires\_grad = True)

a

y = torch.sum(a\*\*2)

y

y.backward()

print(a.grad)

print(a.T)

b = Variable(torch.Tensor([[2,4],[8,10]]),requires\_grad = True)

b

3-Implement OR function in Perceptron

import torch

xin = torch.tensor([[0,0],[0,1],[1,0],[1,1]],dtype=torch.int32)

xin

yout = torch.tensor([[0],[1],[1],[1]],dtype=torch.int32)

yout

w1 = 1

w2 = -1

b = 0

a=1

for j in range(10):

for i in range(4):

yin = b+(w1\*xin[i][0])+(w2\*xin[i][1])

if(yin>0):

y=1

elif(yin==0):

y=0

else:

yin=-1

if(y!=yout[i]):

w1 = w1+(a\*yout[i]\*xin[i][0])

w2 = w2+(a\*yout[i]\*xin[i][1])

b = b+(a\*yout[i])

else:

w1 = w1

w2 = w2

b = b

print("Epoch = ",j,"yin Value = ",yin,"y value = ",y,"Weights And Bias = ",w1," ",w2," ",b," ")

4-Implement Linear regression using Pytorch and include the following:

1. EDA of the dataset should be displayed
2. Missing values should be handled
3. Plot the testing accuracy vs training accuracy

import pandas as pd

import torch

data = pd.read\_csv("seeds.csv")

data["Type"].unique()

data.isnull().sum()

# from sklearn.impute import SimpleImputer

# imputer = SimpleImputer(strategy='most\_frequent')

# data.iloc[:,:] = imputer.fit\_transform(data)

# data.isnull().sum()

# from sklearn.preprocessing import LabelEncoder

# lb = LabelEncoder()

# data["sex"] = lb.fit\_transform(data["sex"])

# data["island"] = lb.fit\_transform(data["island"])

data["Type"].value\_counts()

X = torch.tensor(data.drop("Type", axis=1).values, dtype=torch.float)

y = torch.tensor(

data["Type"],

dtype=torch.long

)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, y, train\_size=0.8, random\_state=42)

X\_train, X\_test, Y\_train, Y\_test = torch.tensor(X\_train, dtype=torch.float32),torch.tensor(X\_test, dtype=torch.float32),torch.tensor(Y\_train, dtype=torch.float32),torch.tensor(Y\_test, dtype=torch.float32)

samples, features = X\_train.shape

from torch import nn

class Regressor(nn.Module):

def \_\_init\_\_(self):

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

self.first\_layer = nn.Linear(features, 3)

self.second\_layer = nn.Linear(3, 10)

self.third\_layer = nn.Linear(10, 15)

self.fourth\_layer = nn.Linear(15, 30)

self.fifth\_layer = nn.Linear(30, 35)

self.final\_layer = nn.Linear(35,1)

self.relu = nn.ReLU()

def forward(self, X\_batch):

layer\_out = self.relu(self.first\_layer(X\_batch))

layer\_out = self.relu(self.second\_layer(layer\_out))

layer\_out = self.relu(self.third\_layer(layer\_out))

layer\_out = self.relu(self.fourth\_layer(layer\_out))

layer\_out = self.relu(self.fifth\_layer(layer\_out))

return self.final\_layer(layer\_out)

regressor = Regressor()

preds = regressor(X\_train[:5])

preds

def TrainModel(model, loss\_func, optimizer, X, Y, epochs=500):

for i in range(epochs):

preds = model(X)

loss = loss\_func(preds.ravel(), Y)

optimizer.zero\_grad()

# loss.backward()

optimizer.step()

if i % 100 == 0:

print("MSE : {:.2f}".format(loss))

from torch.optim import SGD, RMSprop, Adam

torch.manual\_seed(42)

epochs = 1000

learning\_rate = torch.tensor(1/1e3) # 0.001

regressor = Regressor()

mse\_loss = nn.MSELoss()

optimizer = SGD(params=regressor.parameters(), lr=learning\_rate)

TrainModel(regressor, mse\_loss, optimizer, X\_train, Y\_train, epochs=epochs)

test\_preds = regressor(X\_test)

test\_preds[:5]

from sklearn.metrics import r2\_score

print("Train R^2 Score : {:.2f}".format(r2\_score(train\_preds.detach().numpy().squeeze(), Y\_train.detach().numpy())))

print("Test R^2 Score : {:.2f}".format(r2\_score(test\_preds.detach().numpy().squeeze(), Y\_test.detach().numpy())))