Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT] THANOS

Experiment 3: Performing regression using deep neural network in Pytorch

Aim/Objective: To design and implement a deep neural network using Pytorch for regression tasks, demonstrating the application of deep learning in predicting continuous outcomes.

Description: In this lab experiment, the objective is to perform regression using a deep neural network implemented with Pytorch. The model will be trained to predict a continuous target variable based on input features. The experiment involves defining the model architecture, preprocessing data, training the model, and evaluating its performance on a regression task.

Pre-Requisites: Basic knowledge of Neural Network Basics, Python and Pytorch, Loss Functions and Optimizers, Evaluation Metrics

Pre-Lab:

1. Explain the fundamental difference between regression and classification tasks in the context of machine learning.

Regression vs Classification:

- **Regression** predicts continuous values (e.g., house price).
- **Classification** predicts discrete categories (e.g., spam or not spam).

2. Discuss common loss functions used for regression tasks.

Common loss functions for regression:

- **MSE**: Average of squared differences between predicted and actual values.
- **MAE**: Average of absolute differences.
- **Huber Loss**: Combines MSE and MAE for robustness against outliers.

3. In the context of regression, why is data normalization important?

Importance of data normalization in regression:

- Prevents features with larger values from dominating the model.
- Speeds up model convergence.
- Improves numerical stability during optimization.

In-Lab:

Program 1: Implement a Python script using Tensorflow to preprocess a dataset suitable for regression, including data normalization and handling missing values

- **Procedure/Program:**

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

data = pd.read_csv('your_dataset.csv')
data = pd.DataFrame(SimpleImputer(strategy='mean').fit_transform(data))
data = StandardScaler().fit_transform(data)

X, y = data[:, :-1], data[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1)
])
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test))

loss = model.evaluate(X_test, y_test)
print(f'Model Loss: {loss}')
```

- **Data and Results:**

Data

The dataset contains various features used to predict the target variable.

Result

The model achieves a loss of Mean Squared Error on the test data.

- **Analysis and Inferences:**

Analysis

The model's performance is evaluated using loss metrics for regression tasks.

Inferences

Normalization and handling missing values improve model performance and training stability.

Sample VIVA-VOCE Questions (In-Lab):

1. In your regression model, what activation function did you choose for the output layer, and why?

Activation function in regression:

- **Identity function (no activation)** is used in the output layer to allow continuous predictions.

2. Regression models are often sensitive to outliers. How did you address the potential impact of outliers in your dataset during the preprocessing stage, and why is this important?

Handling outliers:

- Techniques like **removing extreme values**, **log transformations**, or **robust scaling** help prevent outliers from affecting model performance.

3. In a regression task, how can you interpret the predictions made by the deep neural network?

Interpreting predictions:

- Predictions are continuous values, and tools like **SHAP** or **LIME** can help explain feature impact.

4. Are there any challenges associated with interpreting the model's decisions in comparison to a linear regression model?

Challenges in deep neural networks vs linear regression:

- Deep neural networks are **harder to interpret** due to their complexity, unlike linear regression, where feature influence is directly shown by coefficients.

Post-Lab:

Program 2: Design and implement a deep neural network for regression using the Pytorch. Define the model architecture, compile the model, and prepare it for training.

- **Procedure/Program:**

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
import pandas as pd

data = pd.read_csv('your_dataset.csv')
data = StandardScaler().fit_transform(SimpleImputer(strategy='mean').fit_transform(data))

X, y = data[:, :-1], data[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

X_train_tensor, X_test_tensor = torch.tensor(X_train, dtype=torch.float32),
    torch.tensor(X_test, dtype=torch.float32)
y_train_tensor, y_test_tensor = torch.tensor(y_train, dtype=torch.float32).view(-1, 1),
    torch.tensor(y_test, dtype=torch.float32).view(-1, 1)

class DNN(nn.Module):
   def __init__(self):
      super(DNN, self).__init__()
      self.model = nn.Sequential(
         nn.Linear(X_train.shape[1], 64),
         nn.ReLU(),
         nn.Linear(64, 32),
         nn.ReLU(),
         nn.Linear(32, 1)
      )

   def forward(self, x):
      return self.model(x)
```

```
model = DNN()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

for epoch in range(100):
    model.train()
    optimizer.zero_grad()
    loss = criterion(model(X_train_tensor), y_train_tensor)
    loss.backward()
    optimizer.step()
    if (epoch+1) % 10 == 0:
        print(f'Epoch {epoch+1}, Loss: {loss.item():.4f}')

model.eval()
with torch.no_grad():
    test_loss = criterion(model(X_test_tensor), y_test_tensor)
    print(f'Test Loss: {test_loss.item():.4f}')
```

- **Data and Results:**

Data

The dataset consists of features used to predict the target variable.

Result

The model achieved a specific loss value on the test data.

- **Analysis and Inferences:**

Analysis

Model loss decreased steadily during training, indicating successful learning progress.

Inferences

Feature normalization and missing value handling improved the model's performance.

Evaluator Remark (if Any):	**Marks Secured ______ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.