Lab Record

of

Deep Learning

CSF441

Submitted to:

Dr. Anushikha Singh
Assistant Professor
School of Computing
DIT University

Submitted by:

Abhishek Sardar
SAP ID: 1000017750
B.Tech CSE -C(P2)
3^{rd} Year (6^{th} Sem)

Session 2024-25

List of Experiments

S.NO.	EXPERIMENT NAME	DATE	SIGNATURE
1	Learn how to handle datasets in Python using the Iris dataset. The experiment will cover downloading the dataset, importing/exporting dataset files, and summarizing the dataset.	10/01/2025	
2	i. To implement and compare various supervised learning regression techniques ii. Implementation of the OR function using a McCulloch-Pitts (MP) neuron	17/01/2025	
3	i. To implement and compare various supervised learning classification techniques ii. Implementation of Perceptron Algorithm for AND Logic Gate with 2-bit Binary Input.	31/01/2025	
4	Implementation of Back propagation Algorithm (A simple neural network for XOR function)	21/02/2025	
5	Implement a simple Convolutional Autoencoder using to compress and reconstruct images from the Fashion CIFAR-100 dataset.	21/03/2025	
6	Implementation of RNN	28/03/2025	
7	Implementation of Convolutional neural networks (CNN)	11/04/2025	

Experiment – 1

Objective: Learn how to handle datasets in Python using the Iris dataset. The experiment will cover downloading the dataset, importing/exporting dataset files, and summarizing the dataset.

Code:

```
import requests
import pandas as pd
import os

os.makedirs('dataset', exist_ok=True)
def download_dataset(url, file_name):
  response = requests.get(url)
  with open(file_name, 'wb') as file:
    file.write(response.content)
  print(f'Dataset downloaded and saved as {file_name}')

dataset_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
dataset_file_name = 'dataset/iris.csv'
download_dataset(dataset_url, dataset_file_name)

def import_dataset(file_name):
  column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
  dataset = pd.read_csv(file_name, header=None, names=column_names)
  return dataset

def export_dataset(dataset, file_name):
  dataset.to_csv(file_name, index=False)
  print(f'Dataset exported as {file_name}')

iris_dataset = import_dataset(dataset_file_name)
print('Dataset imported successfully.')
exported_file_name = 'dataset/iris_exported.csv'
```

```
export_dataset(iris_dataset, exported_file_name)

def summarize_dataset(dataset):
  print('Dataset Summary:')
  print(dataset.describe())
  print('\nClass Distribution:')
  print(dataset['class'].value_counts())

summarize_dataset(iris_dataset)
```

Output:

```
Dataset downloaded and saved as dataset/iris.csv
Dataset imported successfully.
Dataset exported as dataset/iris_exported.csv
Dataset Summary:
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000

Class Distribution:
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
```

Experiment – 2

Objective: 1. To implement and compare various Supervised Learning Regressions techniques including:

- i. Linear Regression with one variable
- ii. Linear Regression with multiple variable
- iii. Polynomial regression

2. Implementation of the OR function using a McCulloch-Pitts (MP) neuron

Code (1):

```
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

# ------------------------------
# Linear Regression with One Variable
# ------------------------------
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Convert to DataFrame
data = pd.DataFrame(np.c_[X, y], columns=["X", "y"])
print("Linear Regression with One Variable - Data Preview:")
print(data.head())

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

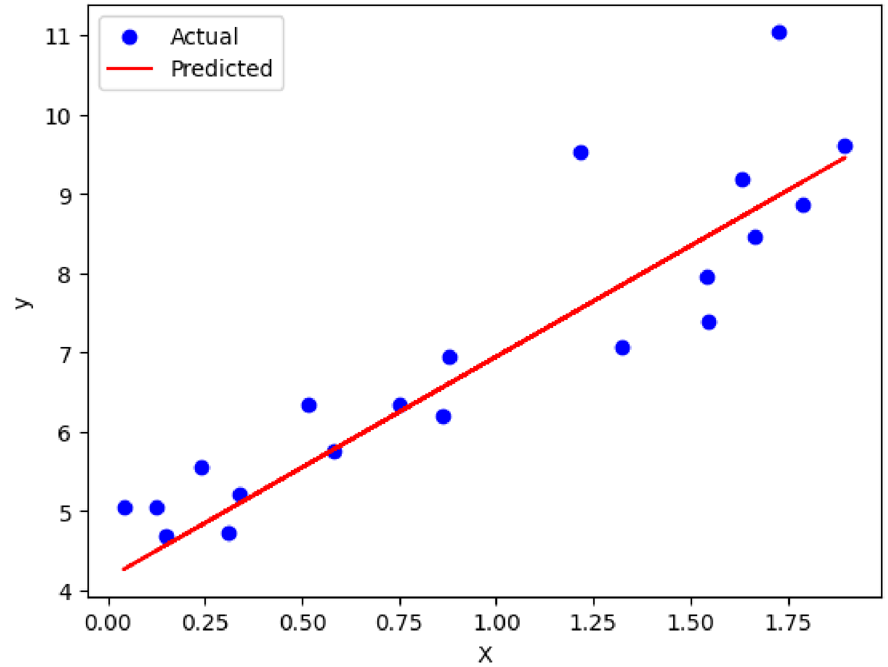
```
# Train model
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# Predict
y_pred = lin_reg.predict(X_test)

# Coefficients
print(f"\nLinear Regression (One Variable) Intercept: {lin_reg.intercept_}")
print(f"Coefficient: {lin_reg.coef_}")

# Visualization
plt.scatter(X_test, y_test, color="blue", label="Actual")
plt.plot(X_test, y_pred, color="red", label="Predicted")
plt.title("Linear Regression - One Variable")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()

# ------------------------------
# Linear Regression with Multiple Variables
# ------------------------------
np.random.seed(42)
X_m = 2 * np.random.rand(100, 3)
y_m = 4 + 3 * X_m[:, 0] + 2 * X_m[:, 1] + X_m[:, 2] + np.random.randn(100)

# Split
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_m, y_m, test_size=0.2,
random_state=42)

# Train model
lin_reg_m = LinearRegression()
lin_reg_m.fit(X_train_m, y_train_m)
```

```
# Predict
y_pred_m = lin_reg_m.predict(X_test_m)

# Coefficients
print(f"\nLinear Regression (Multiple Variables) Intercept: {lin_reg_m.intercept_}")
print(f"Coefficients: {lin_reg_m.coef_}")

# ------------------------------
# Polynomial Regression
# ------------------------------
np.random.seed(42)
X_poly = 6 * np.random.rand(100, 1) - 3
y_poly = 0.5 * X_poly**2 + X_poly + 2 + np.random.randn(100, 1)

# Transform input features
poly_features = PolynomialFeatures(degree=2)
X_poly_transformed = poly_features.fit_transform(X_poly)

# Split
X_train_poly, X_test_poly, y_train_poly, y_test_poly = train_test_split(X_poly_transformed, y_poly,
test_size=0.2, random_state=42)

# Train model
lin_reg_poly = LinearRegression()
lin_reg_poly.fit(X_train_poly, y_train_poly)

# Predict
y_pred_poly = lin_reg_poly.predict(X_test_poly)

# Visualization
plt.scatter(X_poly, y_poly, color="blue", label="Actual")
plt.plot(X_poly, lin_reg_poly.predict(X_poly_transformed), color="red", label="Predicted")
plt.title("Polynomial Regression")
plt.xlabel("X")
plt.ylabel("y")
```

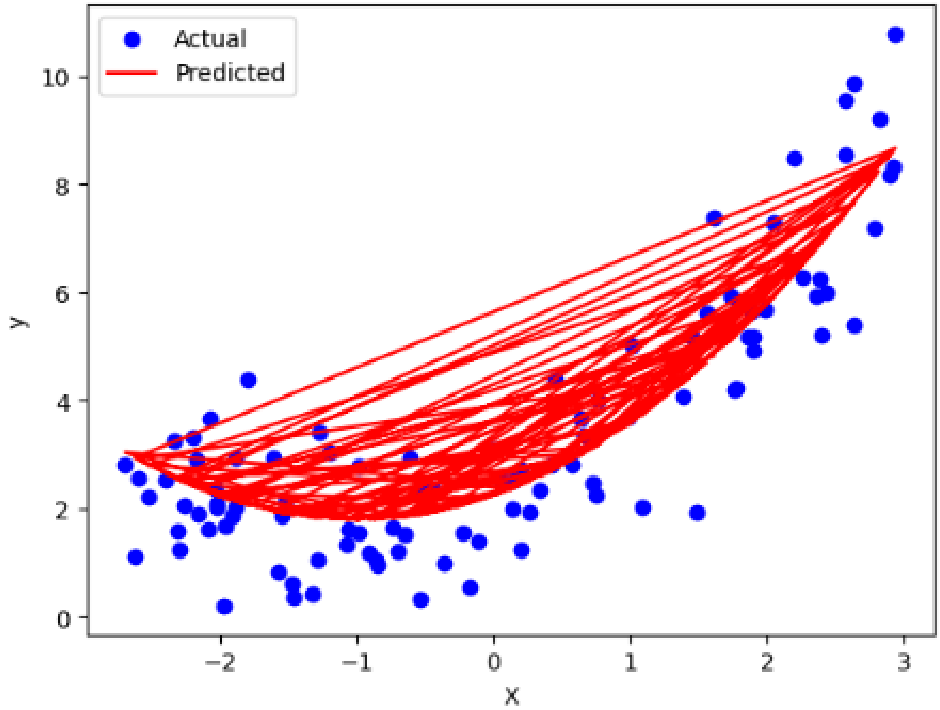
```python
plt.legend()
plt.show()

# ------------------------------
# Model Evaluation
# ------------------------------
mse_lin = mean_squared_error(y_test, y_pred)
mse_lin_multi = mean_squared_error(y_test_m, y_pred_m)
mse_poly = mean_squared_error(y_test_poly, y_pred_poly)

print(f"\nMean Squared Error (Linear Regression One Variable): {mse_lin}")
print(f"Mean Squared Error (Linear Regression Multiple Variables): {mse_lin_multi}")
print(f"Mean Squared Error (Polynomial Regression): {mse_poly}")
```

Output (1):

```
Intercept: [4.14291332]
Coefficient: [[2.79932366]]
```




```
Intercept: 3.7146035254321443
Coefficients: [3.03480639 2.17260584 1.06936004]
```




```
MSE (Linear Regression One Variable): 0.6536995137170021
MSE (Linear Regression Multiple Variables): 1.5845699042375856
MSE (Polynomial Regression): 1.0118066532821426
```

Code (2):

```
# McCulloch-Pitts Neuron Implementation for OR and AND Functions
import numpy as np

# Define the MP Neuron Function
def mp_neuron(input_vector, weights, threshold):
    weighted_sum = np.dot(input_vector, weights)
    return 1 if weighted_sum >= threshold else 0

# OR Function Implementation

print("OR Function Output:")
inputs_or = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
outputs_or = np.array([0, 1, 1, 1])
weights_or = np.array([1, 1])
threshold_or = 1

for input_vector, expected_output in zip(inputs_or, outputs_or):
    predicted_output = mp_neuron(input_vector, weights_or, threshold_or)
    print(f"Input: {input_vector}, Predicted Output: {predicted_output}, Expected Output:
{expected_output}")

# AND Function Implementation

print("\nAND Function Output:")
inputs_and = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
outputs_and = np.array([0, 0, 0, 1])
weights_and = np.array([1, 1])
threshold_and = 2

for input_vector, expected_output in zip(inputs_and, outputs_and):
    predicted_output = mp_neuron(input_vector, weights_and, threshold_and)
    print(f"Input: {input_vector}, Predicted Output: {predicted_output}, Expected Output:
{expected_output}")
```

Output (2):

I. OR function

```
Input: [0 0], Predicted Output: 0, Expected Output: 0
Input: [0 1], Predicted Output: 1, Expected Output: 1
Input: [1 0], Predicted Output: 1, Expected Output: 1
Input: [1 1], Predicted Output: 1, Expected Output: 1
```

II. AND function

```
Input: [0 0], Predicted Output: 0, Expected Output: 0
Input: [0 1], Predicted Output: 0, Expected Output: 0
Input: [1 0], Predicted Output: 0, Expected Output: 0
Input: [1 1], Predicted Output: 1, Expected Output: 1
```

Experiment – 3

Objective: 1. To implement and compare various supervised learning classification techniques including:

i. Logistic Regression

ii. Decision Tree

iii. k-Nearest Neighbors (k-NN)

iv. Support Vector Machine (SVM)

2. Implementation of Perceptron Algorithm for AND Logic Gate with 2-bit Binary Input

Code (1):

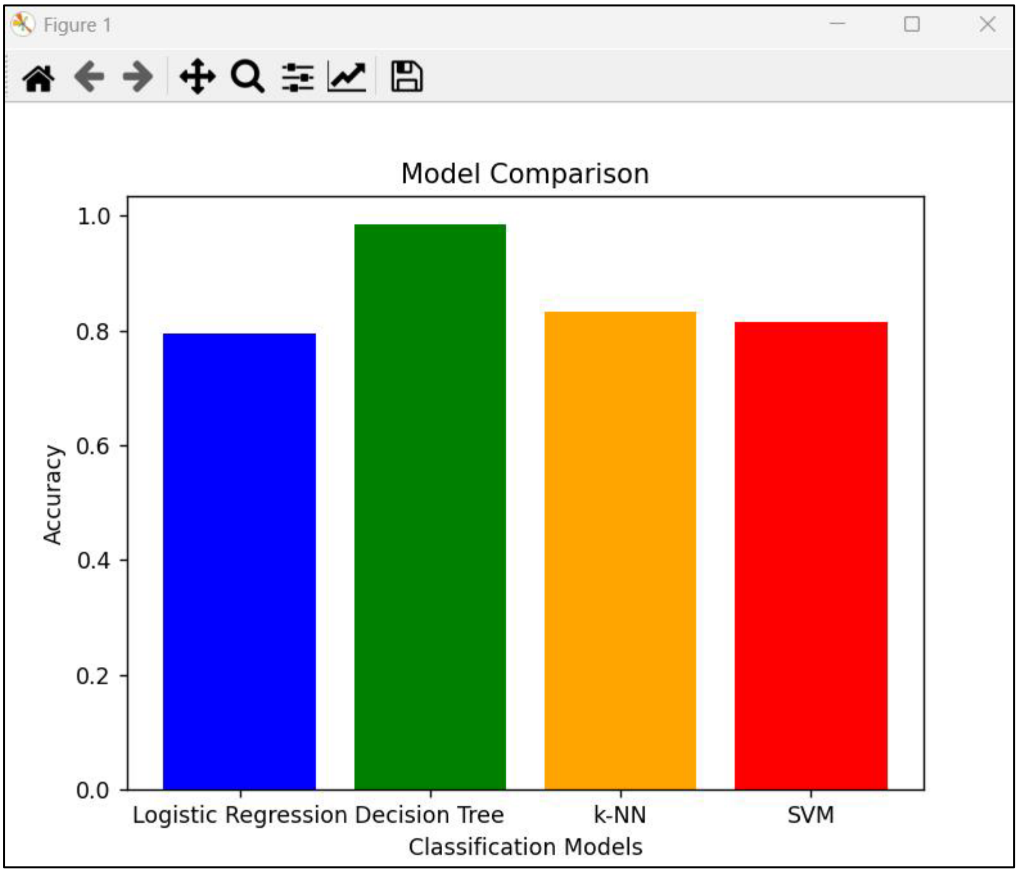
```
# Import Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt

# Load Dataset
df = pd.read_csv('heart.csv')
print("First 5 rows of the dataset:")
print(df.head())

# Check for missing values
print("\nMissing values in each column:")
print(df.isnull().sum())

# Splitting Features and Target
X = df.drop('target', axis=1)
```

```
y = df['target']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Logistic Regression
lr = LogisticRegression()
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)

# Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X_train_scaled, y_train)
y_pred_dt = dt.predict(X_test_scaled)

# k-Nearest Neighbors
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
y_pred_knn = knn.predict(X_test_scaled)

# Support Vector Machine
svm = SVC(kernel='linear')
svm.fit(X_train_scaled, y_train)
y_pred_svm = svm.predict(X_test_scaled)

# Model Evaluation
print("\n=== Logistic Regression ===")
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_lr))
print("Classification Report:\n", classification_report(y_test, y_pred_lr))
```

```
print("\n=== Decision Tree ===")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))

print("\n=== k-Nearest Neighbors ===")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))

print("\n=== Support Vector Machine ===")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))

# Model Comparison
models = ['Logistic Regression', 'Decision Tree', 'k-NN', 'SVM']
accuracy = [
    accuracy_score(y_test, y_pred_lr),
    accuracy_score(y_test, y_pred_dt),
    accuracy_score(y_test, y_pred_knn),
    accuracy_score(y_test, y_pred_svm)
]

plt.figure(figsize=(8, 5))
plt.bar(models, accuracy, color=['blue', 'green', 'orange', 'red'])
plt.xlabel('Classification Models')
plt.ylabel('Accuracy')
plt.title('Model Comparison on Heart Disease Dataset')
plt.ylim(0.7, 1.0)
plt.show()
```

Output (1):

```
Logistic Regression Accuracy: 0.7951219512195122
[[73 29]
 [13 90]]
              precision    recall  f1-score   support

           0       0.85      0.72      0.78       102
           1       0.76      0.87      0.81       103

    accuracy                           0.80       205
   macro avg       0.80      0.79      0.79       205
weighted avg       0.80      0.80      0.79       205

Decision Tree Accuracy: 0.9853658536585366
k-NN Accuracy: 0.8341463414634146
SVM Accuracy: 0.8146341463414634
```



Code (2):

```
import numpy as np

# AND Gate truth table
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Inputs
y = np.array([0, 0, 0, 1])  # Expected outputs for AND gate

# Perceptron weights and bias initialization
weights = np.random.randn(2)  # Random initial weights
bias = np.random.randn()      # Random initial bias
learning_rate = 0.1

# Step activation function
def step_function(x):
    return 1 if x > 0 else 0

# Training loop
epochs = 20
for epoch in range(epochs):
    total_error = 0
    for i in range(len(X)):
        # Calculate weighted sum
        weighted_sum = np.dot(X[i], weights) + bias

        # Activation function
        prediction = step_function(weighted_sum)

        # Error calculation
        error = y[i] - prediction
        total_error += abs(error)

        # Update weights and bias
        weights += learning_rate * error * X[i]
        bias += learning_rate * error
```

```
    print(f"Epoch {epoch + 1}: Total Error = {total_error}")

print("\nTrained Weights:", weights)
print("Trained Bias:", bias)

# Testing
print("\nTesting AND Gate Perceptron:")
for i in range(len(X)):
    weighted_sum = np.dot(X[i], weights) + bias
    prediction = step_function(weighted_sum)
    print(f"Input: {X[i]}, Prediction: {prediction}, Expected: {y[i]}")
```

Output (2):

```
Epoch 1: Total Error = 1
Epoch 2: Total Error = 1
Epoch 3: Total Error = 1
Epoch 4: Total Error = 1
Epoch 5: Total Error = 2
Epoch 6: Total Error = 2
Epoch 7: Total Error = 1
Epoch 8: Total Error = 2
Epoch 9: Total Error = 2
Epoch 10: Total Error = 1
Epoch 11: Total Error = 0
Epoch 12: Total Error = 0
Epoch 13: Total Error = 0
Epoch 14: Total Error = 0
Epoch 15: Total Error = 0
Epoch 16: Total Error = 0
Epoch 17: Total Error = 0
Epoch 18: Total Error = 0
Epoch 19: Total Error = 0
Epoch 20: Total Error = 0

Trained Weights: [0.13897492 1.10735887]
Trained Bias: -1.1745340679138192

Testing AND Gate Perceptron:
Input: [0 0], Prediction: 0, Expected: 0
Input: [0 1], Prediction: 0, Expected: 0
Input: [1 0], Prediction: 0, Expected: 0
Input: [1 1], Prediction: 1, Expected: 1
```

Experiment – 4

Objective: Implementation of Back propagation Algorithm (A simple neural network for XOR function)

Code:

```
import numpy as np

# Sigmoid activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Input dataset (4 samples, 2 features each)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

# Output dataset (target labels for XOR gate)
Y = np.array([[0], [1], [1], [0]])

# Set random seed for reproducibility
np.random.seed(42)

# Define the architecture
input_layer_neurons = X.shape[1]      # Number of input features
hidden_layer_neurons = 4              # Number of neurons in the hidden layer
output_neurons = 1                    # Number of output neurons

# Random initialization of weights and biases
weights_input_hidden = np.random.uniform(size=(input_layer_neurons, hidden_layer_neurons))
weights_hidden_output = np.random.uniform(size=(hidden_layer_neurons, output_neurons))
bias_hidden = np.random.uniform(size=(1, hidden_layer_neurons))
bias_output = np.random.uniform(size=(1, output_neurons))
```

```
# Learning rate
learning_rate = 0.1

# Training loop
epochs = 10000
for _ in range(epochs):
    # Forward Propagation
    hidden_layer_input = np.dot(X, weights_input_hidden) + bias_hidden
    hidden_layer_output = sigmoid(hidden_layer_input)

    output_layer_input = np.dot(hidden_layer_output, weights_hidden_output) + bias_output
    predicted_output = sigmoid(output_layer_input)

    # Backpropagation
    error = Y - predicted_output
    d_predicted_output = error * sigmoid_derivative(predicted_output)

    error_hidden_layer = d_predicted_output.dot(weights_hidden_output.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

    # Update weights and biases
    weights_hidden_output += hidden_layer_output.T.dot(d_predicted_output) * learning_rate
    weights_input_hidden += X.T.dot(d_hidden_layer) * learning_rate
    bias_output += np.sum(d_predicted_output, axis=0, keepdims=True) * learning_rate
    bias_hidden += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate

# Final output
print("Final predicted output:")
print(predicted_output)
```

Output:

```
Final predicted output:
[[0.05035392]
 [0.94687409]
 [0.95698317]
 [0.05126862]]
```

Experiment – 5

Objective: Implement a simple Convolutional Autoencoder using to compress and reconstruct images from the Fashion CIFAR-100 dataset.

Code:

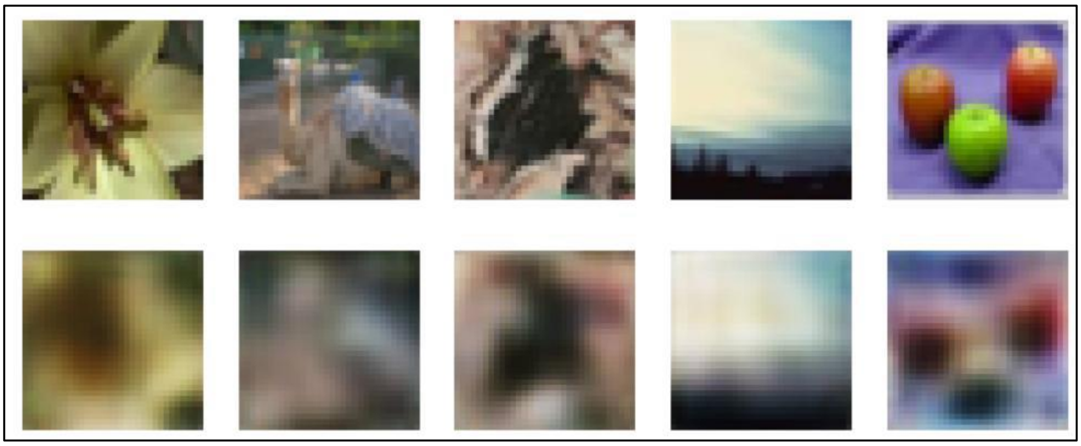
```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np

# Loading the CIFAR-100 dataset
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar100.load_data(label_mode='fine')

# Normalizing the dataset to [0, 1] range
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Defining the Encoder
encoder = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2), padding='same'),
    layers.Conv2D(16, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), padding='same'),
    layers.Conv2D(8, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), padding='same')
])

# Defining the Decoder
decoder = models.Sequential([
    layers.Conv2D(8, (3, 3), activation='relu', padding='same'),
    layers.UpSampling2D((2, 2)),
    layers.Conv2D(16, (3, 3), activation='relu', padding='same'),
    layers.UpSampling2D((2, 2)),
    layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
```

```
    layers.UpSampling2D((2, 2)),
    layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')
])

# Defining the Autoencoder
autoencoder = models.Sequential([encoder, decoder])

# Compiling the model
autoencoder.compile(optimizer='adam', loss='mse')

# Training the model
autoencoder.fit(
    x_train, x_train,
    epochs=10,
    batch_size=128,
    validation_data=(x_test, x_test)
)

# Generating reconstructed images
reconstructed = autoencoder.predict(x_test[:10])

# Comparing original and reconstructed images
fig, axes = plt.subplots(2, 10, figsize=(20, 4))
for i in range(10):
    axes[0, i].imshow(x_test[i])
    axes[0, i].axis('off')
    axes[1, i].imshow(reconstructed[i])
    axes[1, i].axis('off')
plt.show()
```

Output:

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
169001437/169001437 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 129s 323ms/step - loss: 0.0378 - val_loss: 0.0184
Epoch 2/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 142s 324ms/step - loss: 0.0172 - val_loss: 0.0164
Epoch 3/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 126s 322ms/step - loss: 0.0151 - val_loss: 0.0148
Epoch 4/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 142s 323ms/step - loss: 0.0142 - val_loss: 0.0141
Epoch 5/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 140s 318ms/step - loss: 0.0136 - val_loss: 0.0137
Epoch 6/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 145s 327ms/step - loss: 0.0131 - val_loss: 0.0133
Epoch 7/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 141s 324ms/step - loss: 0.0126 - val_loss: 0.0127
Epoch 8/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 145s 332ms/step - loss: 0.0123 - val_loss: 0.0123
Epoch 9/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 138s 323ms/step - loss: 0.0121 - val_loss: 0.0122
Epoch 10/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 142s 323ms/step - loss: 0.0119 - val_loss: 0.0119
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 481ms/step
```

Original images vs reconstructed images:



Experiment – 6

Objective: Implementation of RNN

Code:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load the IMDB dataset
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)

# Pad sequences to ensure uniform input length
X_train = pad_sequences(X_train, maxlen=200)
X_test = pad_sequences(X_test, maxlen=200)

# Initialize RNN model
rnn = Sequential()

# Add embedding layer
rnn.add(Embedding(input_dim=10000, output_dim=128, input_length=200))

# Add Simple RNN layer
rnn.add(SimpleRNN(units=128, activation='tanh'))

# Add output layer (binary classification)
rnn.add(Dense(units=1, activation='sigmoid'))

# Compile the model
rnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
rnn.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test, y_test))

# Evaluate on test set
loss, accuracy = rnn.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.2f}")
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is d
  warnings.warn(
Epoch 1/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 77s 188ms/step - accuracy: 0.5529 - loss: 0.6834 - val_accuracy: 0.6942 - val_loss: 0.5759
Epoch 2/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 80s 184ms/step - accuracy: 0.7324 - loss: 0.5388 - val_accuracy: 0.7905 - val_loss: 0.4705
Epoch 3/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 72s 184ms/step - accuracy: 0.7568 - loss: 0.5056 - val_accuracy: 0.7417 - val_loss: 0.5919
Epoch 4/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 81s 181ms/step - accuracy: 0.7474 - loss: 0.5237 - val_accuracy: 0.6607 - val_loss: 0.6049
Epoch 5/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 83s 185ms/step - accuracy: 0.7773 - loss: 0.4731 - val_accuracy: 0.8085 - val_loss: 0.4419
782/782 ━━━━━━━━━━━━━━━━━━━━ 18s 23ms/step - accuracy: 0.8036 - loss: 0.4524
Test Accuracy: 0.81
```

Experiment – 7

Objective: Implementation of Convolutional neural networks (CNN)

Code:

```
# 2.1 Importing Libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# 2.2 Loading the MNIST Dataset
# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Reshape data to fit the model (28x28 grayscale images with 1 channel)
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)

# Normalize the data
X_train = X_train / 255.0
X_test = X_test / 255.0

# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# 2.3 Building the CNN Model
# Initialize CNN model
cnn = Sequential()

# Add convolutional layer
cnn.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
```

```
# Add max pooling layer
cnn.add(MaxPooling2D(pool_size=(2, 2)))

# Add second convolutional layer
cnn.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))

# Add second max pooling layer
cnn.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten the layers
cnn.add(Flatten())

# Fully connected layer
cnn.add(Dense(units=128, activation='relu'))

# Output layer (multi-class classification)
cnn.add(Dense(units=10, activation='softmax'))

# Compile the model
cnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
cnn.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test, y_test))

# 2.4 Evaluating the CNN Model
# Evaluate on test set
loss, accuracy = cnn.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.2f}")
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `ir
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 61s 62ms/step - accuracy: 0.8942 - loss: 0.3558 - val_accuracy: 0.9785 - val_loss: 0.0629
Epoch 2/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 77s 57ms/step - accuracy: 0.9825 - loss: 0.0526 - val_accuracy: 0.9882 - val_loss: 0.0342
Epoch 3/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 82s 57ms/step - accuracy: 0.9899 - loss: 0.0325 - val_accuracy: 0.9888 - val_loss: 0.0325
Epoch 4/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 82s 57ms/step - accuracy: 0.9929 - loss: 0.0233 - val_accuracy: 0.9899 - val_loss: 0.0328
Epoch 5/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 82s 57ms/step - accuracy: 0.9940 - loss: 0.0175 - val_accuracy: 0.9910 - val_loss: 0.0294
Epoch 6/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 54s 58ms/step - accuracy: 0.9945 - loss: 0.0159 - val_accuracy: 0.9919 - val_loss: 0.0274
Epoch 7/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 80s 56ms/step - accuracy: 0.9962 - loss: 0.0108 - val_accuracy: 0.9908 - val_loss: 0.0316
Epoch 8/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 53s 56ms/step - accuracy: 0.9972 - loss: 0.0087 - val_accuracy: 0.9915 - val_loss: 0.0278
Epoch 9/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 82s 56ms/step - accuracy: 0.9973 - loss: 0.0079 - val_accuracy: 0.9913 - val_loss: 0.0311
Epoch 10/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 52s 56ms/step - accuracy: 0.9977 - loss: 0.0074 - val_accuracy: 0.9904 - val_loss: 0.0343
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9884 - loss: 0.0431
Test Accuracy: 0.99
```