Assignment-1 (DNN + Linear Regression + Price Prediction)

 # import modules and libraries

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

from keras.models import Sequential

from keras.layers import Dense, Dropout

from keras.callbacks import EarlyStopping

import matplotlib.pyplot as plt

 # Load the dataset

df = pd.read_csv("P:\DL\BostonHousingmedv.csv")

 # Display the first few rows of the dataset

print(df.head())

 # Preprocess the data

X = df.drop('medv', axis=1)

y = df['medv']

 # Scale the input features

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

 # Display the first few rows of the scaled input features

print(X_scaled[:5])

 # Split the dataset

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

 # Print the shapes of the training and testing sets

print('Training set shape:', X_train.shape, y_train.shape)

print('Testing set shape:', X_test.shape, y_test.shape)

 # Define the model architecture

model = Sequential()

model.add(Dense(64, activation='relu', input_dim=13))

```python
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
# Display the model summary
print(model.summary())
# Compile the model
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_absolute_error'])
# Train the model
early_stopping = EarlyStopping(monitor='val_loss', patience=5)
history = model.fit(X_train, y_train, validation_split=0.2, epochs=100, batch_size=32, callbacks=[early_stopping])
# Plot the training and validation loss over epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Training', 'Validation'])
plt.show()
# Evaluate the model
loss, mae = model.evaluate(X_test, y_test)
# Calculate RMSE
rmse = np.sqrt(np.mean((predictions - y_test) ** 2))
# Print both MAE and RMSE
print('Mean Absolute Error:', mae)
print('Mean Absolute Error:', mae)
print('Root Mean Squared Error:', rmse)
```

Assignment-2 (DNN + Binary Classification + IMDB Reviews)

 # Load the IMDB dataset

from keras.datasets import imdb

from keras.preprocessing.sequence import pad_sequences

 # Load the IMDB dataset with the top 10,000 most frequent words

(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)

 # Pad or truncate the sequences to a fixed length of 250 words

max_len = 250

X_train = pad_sequences(X_train, maxlen=max_len)

X_test = pad_sequences(X_test, maxlen=max_len)

 # Define the deep neural network architecture

from keras.models import Sequential

from keras.layers import Embedding, Bidirectional, LSTM, Dense

embedding_dim = 128

model = Sequential()

model.add(Embedding(input_dim=10000, output_dim=embedding_dim,
input_length=max_len))

model.add(Bidirectional(LSTM(64, return_sequences=True)))

model.add(Bidirectional(LSTM(64)))

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

 # Compile the model

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

 # Train the model

history = model.fit(X_train, y_train, batch_size=128, epochs=3,
validation_split=0.2)

 # Evaluate the trained model

loss, accuracy = model.evaluate(X_test, y_test,batch_size=128)

print("Test Loss:", loss)

print("Test Accuracy:", accuracy)

```
# Example
# Get word index mapping
word_index = imdb.get_word_index()
# Reverse the word index mapping
index_to_word = {index: word for word, index in word_index.items()}
# Choose a subset of the test data
import numpy as np
subset_size = 10
subset_indices = np.random.choice(len(X_test), size=subset_size, replace=False)
subset_X = X_test[subset_indices]
subset_y_true = y_test[subset_indices]
# Make predictions and convert probabilities to binary predictions
threshold = 0.5
subset_y_pred = (model.predict(subset_X).flatten() > threshold).astype(int)
# Print sample reviews along with their classification
for i in range(subset_size):
    review = " ".join(index_to_word.get(idx - 3, '?') for idx in subset_X[i] if idx != 0)
    true_label = "Positive" if subset_y_true[i] == 1 else "Negative"
    pred_label = "Positive" if subset_y_pred[i] == 1 else "Negative"
    print("Review:", review)
    print("True Label:", true_label)
    print("Predicted Label:", pred_label)
    print()
```

Assignment-3 (CNN + Multiple Classifier + Clothes Classification)

 # Import Module & Libraries

import tensorflow as tf

from tensorflow import keras

import numpy as np

import matplotlib.pyplot as plt

 # Loading the dataset

fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) =
fashion_mnist.load_data()

 # Normalise the images

train_images = train_images / 255.0

test_images = test_images / 255.0

 # Define the model

model = keras.Sequential([

keras.layers.Flatten(input_shape=(28, 28)),

keras.layers.Dense(128, activation='relu'),

keras.layers.Dense(10, activation='softmax')

])

 # Compile the model

model.compile(optimizer='adam',

loss='sparse_categorical_crossentropy',

metrics=['accuracy'])

 # Train the model

model.fit(train_images, train_labels, epochs=10)

 # Evaluate the model

test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)

 # Make predictions

```
predictions = model.predict(test_images)
predicted_labels = np.argmax(predictions, axis=1)
```

 # Example

```
num_rows = 5
num_cols = 5
num_images = num_rows * num_cols
plt.figure(figsize=(2 * 2 * num_cols, 2 * num_rows))
for i in range(num_images):
plt.subplot(num_rows, 2 * num_cols, 2 * i + 1)
plt.imshow(test_images[i], cmap='gray')
plt.axis('off')

plt.subplot(num_rows, 2 * num_cols, 2 * i + 2)
plt.bar(range(10), predictions[i])
plt.xticks(range(10))
plt.ylim([0, 1])
plt.title(f"Predicted label: {predicted_labels[i]}")
plt.tight_layout()
plt.show()
```

Assignment-4 (RNN + Stock Price Prediction)

 # Import Modules and libraries

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, LSTM, Dropout
```

 # Load the dataset

```
data = pd.read_csv("P:\DL\GOOGL.csv")
```

 # Prepare the data

```
dataset = data['Open'].values.reshape(-1, 1)  # Extract the 'Open' column

scaler = MinMaxScaler(feature_range=(0, 1))

dataset = scaler.fit_transform(dataset)
```

 # Create the training and testing datasets

```
training_data_len = int(len(dataset) * 0.8)

training_data = dataset[:training_data_len]

testing_data = dataset[training_data_len:]


def create_dataset(dataset, time_step=1):

    X, Y = [], []

    for i in range(len(dataset) - time_step - 1):

        X.append(dataset[i:(i + time_step), 0])

        Y.append(dataset[i + time_step, 0])

    return np.array(X), np.array(Y)
```

 # Create the training and testing datasets with a time step of 60 days

```
time_step = 60

X_train, Y_train = create_dataset(training_data, time_step)

X_test, Y_test = create_dataset(testing_data, time_step)
```

 # Reshape the training and testing datasets

```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

 # Create the RNN model

```
model = Sequential()
model.add(LSTM(units=50, return_sequences=True,
input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))
model.add(Dense(units=1))
```

 # Compile the model

```
model.compile(optimizer='adam', loss='mean_squared_error')
```

 # Train the model

```
model.fit(X_train, Y_train, epochs=100, batch_size=32)
```

 # Evaluate the model

```
loss = model.evaluate(X_test, Y_test)
print("Test Loss:", loss)
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

 # Make predictions

```
predictions = model.predict(X_test)
```

 # Calculate metrics

```
mse = mean_squared_error(Y_test, predictions)
mae = mean_absolute_error(Y_test, predictions)
rmse = np.sqrt(mse)
```

```
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("Root Mean Squared Error (RMSE):", rmse)
# Example
# Inverse transform the scaled values to their original scale
Y_test_inverse = scaler.inverse_transform(Y_test.reshape(-1, 1))
predictions_inverse = scaler.inverse_transform(predictions)
# Plotting
plt.figure(figsize=(14, 7))
plt.plot(Y_test_inverse, label='Actual')
plt.plot(predictions_inverse, label='Predicted')
plt.title('Actual vs. Predicted Stock Prices')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```