Ex-4: Design a neural network for predicting house prices using Boston Housing Price dataset.

Objective

To design and implement a neural network model for predicting house prices using the **Boston Housing Price dataset**. The model will use TensorFlow/Keras for implementation.

---

1. Introduction

Predicting house prices is a **regression problem**, where the goal is to estimate a continuous value. The Boston Housing dataset contains information about **housing prices** in Boston, along with **features** such as crime rate, property tax, and number of rooms.

Dataset Details:

- **Features (X)**: 13 numerical attributes (e.g., crime rate, average number of rooms, etc.)
- **Target (Y)**: Median house price in $1000s

Tools & Libraries Required:

- Python 3.x
- TensorFlow/Keras
- NumPy, Pandas, Matplotlib, Seaborn
- Scikit-learn

---

2. Steps to Implement the Neural Network

Step 1: Install & Import Libraries

```
import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Load the Dataset

```
from sklearn.datasets import fetch_california_housing

# Load dataset
data = fetch_california_housing()

# Convert to Pandas DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['PRICE'] = data.target  # Add target variable

# Display first few rows
df.head()
```

Note: fetch_california_housing is used as a modern replacement for load_boston, which has been deprecated.

Step 3: Data Preprocessing

```
# Split features and target variable
X = df.drop('PRICE', axis=1)
y = df['PRICE']

# Split into training & testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize data for better neural network performance
```

```python
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Step 4: Define the Neural Network Model

```python
# Create a Sequential model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(1)  # Output layer (single neuron for regression)
])

# Compile the model
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

Explanation:

- Dense(128, activation='relu'): First hidden layer with 128 neurons and ReLU activation.
- Dropout(0.2): Helps prevent overfitting.
- Dense(64, activation='relu'): Second hidden layer with 64 neurons.
- Dense(32, activation='relu'): Third hidden layer with 32 neurons.
- Dense(1): Output layer with **one neuron** since this is a regression problem.
- Adam optimizer: Efficient optimization algorithm.
- MSE (Mean Squared Error): Loss function for regression.
- MAE (Mean Absolute Error): Performance metric.

Step 5: Train the Model

```
# Train the model
history = model.fit(X_train, y_train, epochs=150, batch_size=16, validation_split=0.2)
```

Key Parameters:

- epochs=150: Increased training iterations.
- batch_size=16: Number of samples per update.
- validation_split=0.2: 20% of training data used for validation.

Step 6: Evaluate the Model

```
# Evaluate on test data
loss, mae = model.evaluate(X_test, y_test)
print(f"Test Loss (MSE): {loss}")
print(f"Test MAE: {mae}")
```

Mean Absolute Error (MAE): Measures how far predicted prices are from actual prices.

Step 7: Visualize Training Performance

```
# Plot training loss and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Step 8: Make Predictions

```
# Predict house prices
y_pred = model.predict(X_test)
```

```
# Compare actual vs predicted values
plt.figure(figsize=(10,6))
sns.scatterplot(x=y_test, y=y_pred.flatten())
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted House Prices")
plt.show()
```

---

3. Observations & Result

1. **Data Preprocessing**: Standardization helped the model converge faster.
2. **Model Performance**: The neural network showed good performance with **low MAE**.
3. **Hyperparameter Tuning**: Experimenting with more layers, neurons, and different optimizers can improve accuracy.
4. **Generalization**: Regularization (Dropout, L2) and more data can further prevent overfitting.

---

4. Experimentation Ideas

✓ Try adding **more layers or neurons** to improve accuracy. ✓ Test different **activation functions** (e.g., tanh, leaky ReLU). ✓ Implement **early stopping** to prevent overfitting. ✓ Try a **different optimizer** (SGD, RMSprop). ✓ Use **feature engineering** to create new useful features.

---

5. Summary Table

Step	Task
1	Import Libraries
2	Load and Explore Dataset
3	Data Preprocessing (Train-Test Split, Scaling)
4	Define Neural Network Model
5	Train the Model
6	Evaluate Performance
7	Visualize Training and Results
8	Make Predictions

---

6. Conclusion

This lab demonstrated how to build a neural network for predicting **house prices** using TensorFlow/Keras. The model successfully learned patterns in the data and made accurate predictions. Further optimizations can improve performance!

---

7. References

- TensorFlow Documentation: https://www.tensorflow.org
- Keras API: https://keras.io
- Scikit-learn: https://scikit-learn.org