EXPERIMENT NUMBER : 04

AIM : Design a neural network for predicting house prices using Boston Housing Price dataset

THEORY :

This experiment focuses on building a regression-based neural network model to predict housing prices using the Boston Housing Dataset. The dataset contains various attributes of residential areas in Boston, and the goal is to estimate the median value of owner-occupied homes (MEDV) based on these features.

1. Dataset Overview:

The dataset consists of 13 numerical input features and 1 output target (MEDV - Median house value in $1000s). Some important features include:

- CRIM: Crime rate by town
- RM: Average number of rooms per dwelling
- LSTAT: Percentage of lower status population
- TAX: Property tax rate
- PTRATIO: Pupil-teacher ratio, etc.

2. Data Preprocessing:

Before feeding data to the neural network:

- The dataset is loaded using multiple separator checks to ensure flexibility in file format.
- Column names are assigned manually if not present.
- Feature and target variables are separated (X and y respectively).
- The data is split into training and testing sets (80% training, 20% testing).
- StandardScaler is used for feature normalization, ensuring faster convergence during training and better model performance.

3. Neural Network Architecture:

The model is built using TensorFlow and Keras, with the following architecture:

- Input Layer: Takes 13 features.
- First Hidden Layer: 64 neurons, ReLU activation.
- Second Hidden Layer: 32 neurons, ReLU activation.
- Output Layer: Single neuron (since it's a regression task), no activation function

4. Compilation and Training:

- Optimizer: Adam — an efficient optimization algorithm that adjusts learning rate adaptively.
- Loss Function: Mean Squared Error (MSE) — appropriate for regression tasks.
- Metric: Mean Absolute Error (MAE) — measures average prediction error.

- The model is trained for 100 epochs with a batch size of 32 and 20% validation split to monitor overfitting during training.

CODE :

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
import matplotlib.pyplot as plt

# Load the dataset
try:
    df = pd.read_csv('housing.csv', sep=',')
    if df.shape[1] == 1: #Check if the dataframe only has one column
        print("Dataframe loaded with only one column. Trying to load with different
        separator.")
        raise KeyError #Force the except to run
except KeyError:
    try:
        df = pd.read_csv('housing.csv', sep='\s+') #Try loading with whitespace
        if df.shape[1] == 1: #Check if the dataframe only has one column
        print("Dataframe loaded with only one column. Trying to load with different
        separator.")
        raise KeyError
    except KeyError:
        df = pd.read_csv('housing.csv', sep=';') #Try loading with semicolon
        if df.shape[1] == 1: #Check if the dataframe only has one column
        print("Dataframe still has only one column. Verify that the correct seperator is
        being used")

if 'MEDV' not in df.columns:
    try:
        df.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV'] #Assign column names to dataframe if
        they do not exist
    except:
        print("Error: Could not assign column names to the dataframe. Please verify the
        .csv is formatted correctly")
```

```
# Prepare the data
X = df.drop('MEDV', axis=1).values
y = df['MEDV'].values

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

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the neural network model
model = tf.keras.models.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) # Output layer with a single neuron for regression
])

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

# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2,
verbose=0)

# Evaluate the model
loss, mae = model.evaluate(X_test, y_test, verbose=0)
print(f'Mean Absolute Error on test data: {mae}')

# Make predictions on the test set
y_pred = model.predict(X_test, verbose = 0).flatten()

# Create a plot of actual vs. predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6)
plt.plot(y_test, y_test, color='red') # Ideal prediction line
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted House Prices")
```

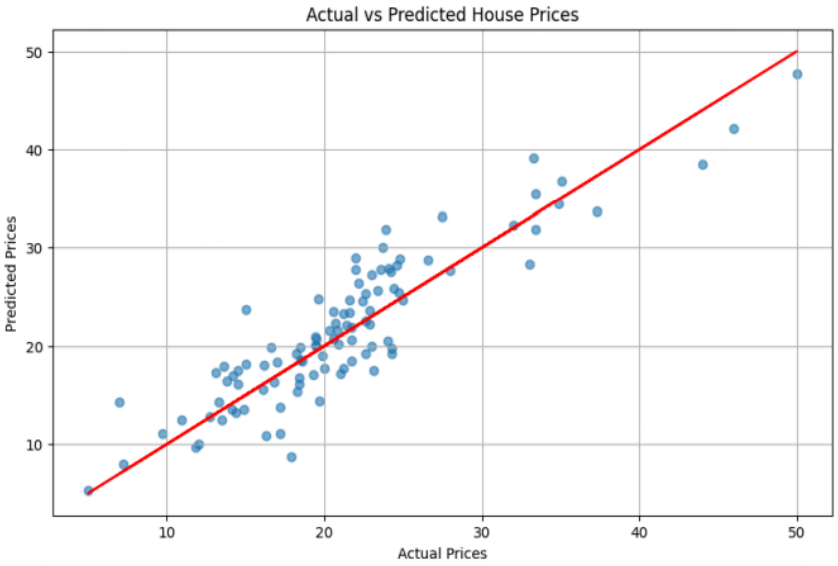
```
plt.grid(True)
plt.show()

# Calculate and display the error
errors = y_pred - y_test

print("\nExample Predicted Prices vs Actual Prices (First 10 samples)")
for i in range(10):
        print(f"Predicted: {y_pred[i]:.2f}, Actual: {y_test[i]:.2f}, Error: {errors[i]:.2f}")

# Example prediction (replace with your desired input features)
example_input = np.array([[0.00632, 18.0, 2.31, 0, 0.538, 6.575, 65.2, 4.0900, 1, 296, 15.3,
396.90, 4.98]]) #Example from the first row
example_input_scaled = scaler.transform(example_input)
prediction = model.predict(example_input_scaled, verbose = 0)
print(f"\nPrediction for the example input: {prediction[0][0]:.2f}")
```

OUTPUT :




```
Example Predicted Prices vs Actual Prices (First 10 samples)
Predicted: 25.30, Actual: 22.60, Error: 2.70
Predicted: 32.31, Actual: 32.00, Error: 0.31
Predicted: 17.93, Actual: 13.60, Error: 4.33
Predicted: 25.65, Actual: 23.40, Error: 2.25
Predicted: 35.46, Actual: 33.40, Error: 2.06
Predicted: 21.56, Actual: 20.80, Error: 0.76
Predicted: 9.96, Actual: 12.00, Error: -2.04
Predicted: 13.26, Actual: 14.40, Error: -1.14
Predicted: 24.84, Actual: 19.60, Error: 5.24
Predicted: 16.35, Actual: 16.80, Error: -0.45

Prediction for the example input: 29.35
```

LEARNING OUTCOMES :