

Title of the Assignment: Linear regression by using Deep Neural network: Implement Boston housing price. Prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
#import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from keras.optimizers import Adam
from keras.layers import LeakyReLU, Dropout
```

```
# Load the dataset
file_path = "BostonHousing.csv"
data = pd.read_csv(file_path)
```

```
# Preprocess the data
X = data.drop(columns=['lstat']) # Features
y = data['lstat'] # Target
```

```
# Drop the "Unnamed: 14" column
# data = data.drop(columns=["Unnamed: 14"])
```

```
# Remove rows with NaN values
data = data.dropna()
```

```
# Preprocess the data again
X = data.drop(columns=['medv']) # Features
y = data['medv'] # Target
```

```
# Split the data into training and test sets
```

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

```
# Standardize the features
```

```
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

```
# Build the model
```

```
model = Sequential([
```

```
    Dense(128, activation=LeakyReLU(alpha=0.1), input_shape=(X_train_scaled.shape[1],)),
```

```
    Dropout(0.2), # Adjust dropout rate
```

```
    Dense(64, activation=LeakyReLU(alpha=0.1)),
```

```
    Dropout(0.2), # Adjust dropout rate
```

```
    Dense(32, activation=LeakyReLU(alpha=0.1)),
```

```
    Dropout(0.2), # Adjust dropout rate
```

```
    Dense(1) # Linear regression output layer
```

```
])
```

```
# Adjusted Training Parameters
```

```
optimizer = Adam(learning_rate=0.0001) # Adjust learning rate
```

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

```
model.fit(X_train_scaled, y_train, epochs=200, batch_size=32, validation_split=0.2)
```

```
# Evaluate the model
```

```
test_loss = model.evaluate(X_test_scaled, y_test)
```

```
print("Test Loss:", test_loss)
```

```
# Make predictions
```

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












```
# Print some predictions and actual values

print("Some Predictions and Actual Values:")

for i in range(10):

    print("Predicted Price:", predictions[i][0], "Actual Price:", y_test.iloc[i])
```

OUTPUT:

```
11/11  0s 7ms/step - loss: 586.2780 - val_loss: 554.0833
Epoch 3/200
11/11  0s 7ms/step - loss: 638.6615 - val_loss: 549.9124
Epoch 4/200
11/11  0s 7ms/step - loss: 567.4115 - val_loss: 545.4855
Epoch 5/200
11/11  0s 8ms/step - loss: 606.4957 - val_loss: 540.7936
Epoch 6/200
11/11  0s 7ms/step - loss: 592.2616 - val_loss: 535.6504
Epoch 7/200
11/11  0s 7ms/step - loss: 619.2240 - val_loss: 530.1999
Epoch 8/200
11/11  0s 7ms/step - loss: 603.3009 - val_loss: 524.2396
Epoch 9/200
11/11  0s 7ms/step - loss: 565.3638 - val_loss: 518.0078
Epoch 10/200
11/11  0s 8ms/step - loss: 573.4431 - val_loss: 511.1730
Epoch 11/200
11/11  0s 7ms/step - loss: 576.8746 - val_loss: 503.8320
Epoch 12/200
11/11  0s 7ms/step - loss: 555.8558 - val_loss: 495.9380
Epoch 13/200
...
Predicted Price: 13.876977 Actual Price: 17.8
Predicted Price: 14.338985 Actual Price: 14.0
Predicted Price: 23.746489 Actual Price: 19.6
Predicted Price: 15.878633 Actual Price: 16.8
```