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..

Linear regression is a statistical method that tries to show a relationship between variables

These regression estimates are used to explain the

relationship between one dependent variable and one or more independent variables. The simplest

form of the regression equation with one dependent and one independent variable is defined by the

formula y = c + b\*x, where y = estimated dependent variable score, c = constant, b = regression

coefficient, and x = score on the independent variable.

The Boston Housing Dataset is a popular dataset in machine learning and contains information about

various attributes of houses in Boston. The goal of using deep neural networks on this dataset is to

predict the median value of owner-occupied homes.

The Boston Housing Dataset contains 13 input variables or features, such as crime rate, average

number of rooms per dwelling, and distance to employment centers. The target variable is the median

value of owner-occupied homes. The dataset has 506 rows, which is not very large, but still sufficient

to train a deep neural network.

To implement a deep neural network on the Boston Housing Dataset, we can follow these steps:

**Load the dataset:** We can load the dataset using libraries like pandas or numpy.

**Preprocess the data:** We need to preprocess the data by scaling the input features so that they have

zero mean and unit variance. This step is important because it helps the neural network to converge

faster.

**Split the dataset:** We split the dataset into training and testing sets. We can use a 70/30 or 80/20 split

for training and testing, respectively.

**Define the model architecture:** We need to define the architecture of our deep neural network. We

can use libraries like Keras or PyTorch to define our model. The architecture can include multiple

hidden layers with various activation functions and regularization techniques like dropout.

**Compile the model:** We need to compile the model by specifying the loss function, optimizer, and

evaluation metrics. For regression problems like this, we can use mean squared error as the loss

function and adam optimizer.

**Train the model:** We can train the model using the training data. We can use techniques like early

stopping to prevent overfitting.

**Evaluate the model:** We can evaluate the model using the testing data. We can calculate the mean

squared error or the mean absolute error to evaluate the performance of the model

Overall, using a deep neural network on the Boston Housing Dataset can result in accurate predictions

of the median value of owner-occupied homes. By following the above steps, we can implement a

deep neural network and fine-tune its hyperparameters to achieve better performance.

import numpy as np

This line imports the NumPy library and assigns it the alias np. NumPy is a powerful library for numerical computations in Python, providing support for large, multi-dimensional arrays and a collection of mathematical functions.

pythonCopy code

import pandas as pd

This line imports the pandas library and assigns it the alias pd. pandas is a library for data manipulation and analysis. It provides data structures like DataFrame and Series, as well as functions for reading and writing data in various formats.

pythonCopy code

import matplotlib.pyplot as plt

This line imports the pyplot module from the matplotlib library and assigns it the alias plt. matplotlib is a plotting library that provides a MATLAB-like interface for creating static, animated, and interactive visualizations in Python.

pythonCopy code

import seaborn as sns

This line imports the seaborn library and assigns it the alias sns. seaborn is a data visualization library built on top of matplotlib. It provides a high-level interface for creating informative and visually appealing statistical graphics.

pythonCopy code

from sklearn.model\_selection import train\_test\_split

This line imports the train\_test\_split function from the model\_selection module of the scikit-learn library. train\_test\_split is a utility function that splits datasets into random train and test subsets. It is commonly used for evaluating machine learning models.

pythonCopy code

from sklearn.preprocessing import StandardScaler

This line imports the StandardScaler class from the preprocessing module of scikit-learn. StandardScaler is a data preprocessing class that standardizes features by removing the mean and scaling to unit variance. It is often used to normalize data before applying machine learning algorithms.

pythonCopy code

from sklearn.metrics import r2\_scorefrom sklearn.metrics import mean\_squared\_error

These lines import the r2\_score and mean\_squared\_error functions from the metrics module of scikit-learn. These functions are used to evaluate the performance of regression models. r2\_score calculates the coefficient of determination, which indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. mean\_squared\_error calculates the mean squared error between the predicted and true values.

pythonCopy code

import kerasfrom keras.layers import Dense, Activation, Dropoutfrom keras.models import Sequential

These lines import the Keras library for deep learning. keras is a high-level neural networks API that runs on top of other deep learning frameworks such as TensorFlow or Theano. The Dense, Activation, and Dropout classes are used to define layers in a neural network, and the Sequential class is used to build a sequential model by stacking layers.

pythonCopy code

import warnings

warnings.filterwarnings("ignore")

These lines import the warnings module and use the filterwarnings function to ignore warning messages. It is a common practice to suppress warnings that might be displayed during program execution.

These import statements ensure that the required libraries, modules, functions, and classes are available for the subsequent code to run without any issues.

Certainly! Let's break down the code line by line:

```python

data = pd.read\_csv("Boston.csv")

```

- This line reads the data from a CSV file named "Boston.csv" and assigns it to the variable `data`. It uses the `read\_csv()` function from the pandas library (`pd`) to load the data into a DataFrame.

```python

data.head()

```

- This line prints the first few rows of the DataFrame `data`. The `head()` function is used to display the top rows of the DataFrame, which gives a quick overview of the data.

```python

data.drop(data.columns[[0]], axis=1, inplace=True)

```

- This line drops the first column from the DataFrame `data`. It uses the `drop()` function from pandas to remove the specified column. `data.columns[[0]]` selects the first column based on its index, and `axis=1` indicates that we are dropping columns. The `inplace=True` parameter modifies the DataFrame in place, so the change is applied to `data`.

By dropping the first column, it seems that the code is removing an unnecessary column or an identifier that is not relevant for the subsequent analysis.

Certainly! Here's an explanation of each line:

```python

print(data.shape)

```

- This line prints the shape of the DataFrame `data`, which represents the number of rows and columns in the dataset. It outputs two values: the number of rows and the number of columns.

```python

print(data.dtypes)

```

- This line prints the data types of each column in the DataFrame `data`. It provides information about the type of data stored in each column, such as integer, float, or object.

```python

print(data.isnull().sum())

```

- This line calculates the number of missing values (NaN or null) in each column of the DataFrame `data`. It uses the `isnull()` method to create a boolean mask that identifies missing values in the DataFrame, and then the `sum()` method is applied to count the number of True values for each column.

```python

print(data.describe())

```

- This line generates summary statistics for each column in the DataFrame `data`. The `describe()` method computes various statistics, including count, mean, standard deviation, minimum, quartiles, and maximum, for numeric columns in the DataFrame. It provides a quick overview of the distribution and central tendencies of the data.

Each of these lines helps to explore and understand the characteristics of the dataset, such as its shape, data types, missing values, and summary statistics.

Certainly! Let's go through the code snippet step by step:

```python

sns.displot(data.medv)

```

- This line uses the `displot()` function from the Seaborn library (`sns`) to create a distribution plot (histogram) of the 'medv' column from the DataFrame `data`. It visualizes the distribution of the median value of owner-occupied homes in Boston.

```python

correlation = data.corr()

```

- This line calculates the correlation matrix for all columns in the DataFrame `data`. The `corr()` function computes the pairwise correlation of columns, and the result is assigned to the variable `correlation`. The correlation matrix shows how each variable is related to each other variable in the dataset.

```python

correlation.loc['medv']

```

- This line selects the row in the `correlation` DataFrame corresponding to the 'medv' column. It displays the correlation coefficients of the 'medv' column with all other columns in the dataset. This provides insights into the strength and direction of the linear relationship between the 'medv' variable and other variables.

```python

fig, axes = plt.subplots(figsize=(15, 12))

```

- This line creates a figure and a set of subplots using `plt.subplots()` from the Matplotlib library (`plt`). It sets the size of the figure to (15, 12), meaning the resulting plot will have a width of 15 inches and a height of 12 inches. The resulting figure and axes objects are assigned to the variables `fig` and `axes`, respectively.

```python

sns.heatmap(correlation, square=True, annot=True)

```

- This line creates a heatmap using the `heatmap()` function from Seaborn. It visualizes the correlation matrix (`correlation`) as a color-coded matrix where each cell represents the correlation between two variables. The `square=True` parameter ensures that the cells are square-shaped, and the `annot=True` parameter displays the correlation values inside the cells.

In summary, this code snippet creates a histogram to visualize the distribution of the 'medv' variable and a heatmap to visualize the correlations between 'medv' and other variables in the dataset. These visualizations provide insights into the relationships and patterns within the data.

Certainly! Let's go through the code block line by line:

```python

X = data.iloc[:,:-1]

```

- This line selects all rows and all columns except the last one from the `data` DataFrame and assigns it to the variable `X`. It represents the features or independent variables used for training the model.

```python

y = data.medv

```

- This line selects the 'medv' column from the `data` DataFrame and assigns it to the variable `y`. It represents the target variable or dependent variable that we want to predict.

```python

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=4)

```

- This line uses the `train\_test\_split` function from scikit-learn to split the data into training and testing sets. It takes the features `X` and the target variable `y` as inputs and splits them into four separate variables: `X\_train` (training features), `X\_test` (testing features), `y\_train` (training target variable), and `y\_test` (testing target variable). The `test\_size=0.2` parameter specifies that 20% of the data will be used for testing, and `random\_state=4` sets a seed for random shuffling of the data to ensure reproducibility.

```python

sc = StandardScaler()

```

- This line creates an instance of the `StandardScaler` class from scikit-learn and assigns it to the variable `sc`. The `StandardScaler` is used to standardize the features by removing the mean and scaling to unit variance.

```python

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

```

- These lines apply the standardization to the training and testing features. `sc.fit\_transform(X\_train)` fits the scaler on the training data and then transforms it, while `sc.transform(X\_test)` only transforms the testing data using the parameters learned from the training data. This ensures that the scaling is consistent between the training and testing datasets. The standardized features are assigned back to `X\_train` and `X\_test`, respectively.

These steps of selecting features, splitting the data, and standardizing the features are common practices in machine learning workflows to prepare the data for training and testing machine learning models.

Certainly! Let's go through the code step by step:

```python

model = Sequential()

```

- This line creates a new sequential model using the Keras library. The sequential model is a linear stack of layers.

```python

model.add(Dense(128, activation='relu', input\_dim=13))

```

- This line adds a dense (fully connected) layer to the model. The layer has 128 units/neurons, uses the ReLU activation function, and expects an input dimension of 13. The input dimension specifies the shape of the input data.

```python

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

model.add(Dense(32, activation='relu'))

model.add(Dense(16, activation='relu'))

```

- These lines add additional dense layers to the model. Each layer has a specified number of units/neurons and uses the ReLU activation function.

```python

model.add(Dense(1))

```

- This line adds the final dense layer to the model, which has a single unit/neuron. This layer does not have an activation function specified, as it will be used for regression tasks where the output is a continuous value.

```python

model.compile(optimizer='adam', loss='mean\_squared\_error')

```

- This line compiles the model. It specifies the optimizer (in this case, 'adam') and the loss function to be used (mean squared error). The optimizer is responsible for updating the model's weights during training, and the loss function measures the error between the predicted values and the true values.

```python

model.summary()

```

- This line prints a summary of the model, displaying the architecture, the number of parameters, and other useful information.

```python

model.fit(X\_train, y\_train, epochs=100)

```

- This line trains the model using the training data. The `fit` function is used to fit the model to the data. It takes the training features (`X\_train`) and the corresponding target variable (`y\_train`). The `epochs` parameter specifies the number of times the entire dataset will be passed through the model during training.

```python

y\_pred = model.predict(X\_test)

```

- This line uses the trained model to make predictions on the testing data (`X\_test`). The `predict` function returns the predicted values for the target variable.

```python

r2 = r2\_score(y\_test, y\_pred)

rmse = (np.sqrt(mean\_squared\_error(y\_test, y\_pred)))

```

- These lines calculate the evaluation metrics for the model's predictions. `r2\_score` calculates the coefficient of determination (R2 score), which measures the proportion of the variance in the target variable that is predictable from the features. `mean\_squared\_error` calculates the mean squared error (MSE), which measures the average squared difference between the predicted and true values.

```python

print("R2 Score = ", r2)

print("RMSE Score = ", rmse)

```

- These lines print the R2 score and RMSE score to the console, which provide information about the performance of the model on the testing data. R2 score ranges from 0 to 1, with a higher value indicating a better fit. RMSE is a measure of the average prediction error, with a lower value indicating better accuracy.

This code trains a neural network model using the Keras library, performs predictions on the testing data, and evaluates the model's performance using R2 score and RMSE.