

410255: LP-V. DL1 22-23 (2 nd Sem)					
Student Name :					
BE Computer Division		Roll No		Batch	
Experiment No: 1	Linear Regression by using Deep Learning				

Aim : 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.

Objective of the Assignment: Students should be able to perform Linear regression by using Deep Neural network on Boston House Dataset.

Prerequisite:

1. Basic of programming language
2. Concept of Linear Regression
3. Concept of Deep Neural Network

Contents for Theory:

1. What is Linear Regression
2. Example of Linear Regression
3. Concept of Deep Neural Network
4. How Deep Neural Network Work
5. Code Explanation with Output

What is Linear Regression?

Linear regression is a statistical approach that is commonly used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the variables and uses mathematical methods to estimate the coefficients that best fit the data.

Deep neural networks are a type of machine learning algorithm that are modeled after the structure and function of the human brain. They consist of multiple layers of interconnected neurons that process data and learn from it to make predictions or classifications.

Linear regression using deep neural networks combines the principles of linear regression with the power of deep learning algorithms. In this approach, the input features are passed through one or more layers of neurons to extract features and then a linear regression model is applied to the output of the last layer to make predictions. The weights and biases of the neural network are adjusted during training to optimize the performance of the model.

This approach can be used for a variety of tasks, including predicting numerical values,

such as stock prices or housing prices, and classifying data into categories, such as detecting whether an image contains a particular object or not. It is often used in fields such as finance, healthcare, and image recognition.

Example Of Linear Regression

A suitable example of linear regression using deep neural network would be predicting the price of a house based on various features such as the size of the house, the number of bedrooms, the location, and the age of the house.

In this example, the input features would be fed into a deep neural network, consisting of multiple layers of interconnected neurons. The first few layers of the network would learn to extract features from the input data, such as identifying patterns and correlations between the input features. The output of the last layer would then be passed through a linear regression model, which would use the learned features to predict the price of the house.

During training, the weights and biases of the neural network would be adjusted to minimize the difference between the predicted price and the actual price of the house. This process is known as gradient descent, and it involves iteratively adjusting the model's parameters until the optimal values are reached. Once the model is trained, it can be used to predict the price of a new house based on its features. This approach can be used in the real estate industry to provide accurate and reliable estimates of house prices, which can help both buyers and sellers make informed decisions.

Concept of Deep Neural Network

A deep neural network is a type of machine learning algorithm that is modeled after the structure and function of the human brain. It consists of multiple layers of interconnected nodes, or artificial neurons, that process data and learn from it to make predictions or classifications.

Each layer of the network performs a specific type of processing on the data, such as identifying patterns or correlations between features, and passes the results to the next layer. The layers closest to the input are known as the "input layer", while the layers closest to the output are known as the "output layer". The intermediate layers between the input and output layers are known as "hidden layers". These layers are responsible for extracting increasingly complex features from the input data, and can be deep (i.e., containing many hidden layers) or shallow (i.e., containing only a few hidden layers).

Deep neural networks are trained using a process known as backpropagation, which involves adjusting the weights and biases of the nodes based on the error between the predicted output and the actual output. This process is repeated for multiple iterations until the model reaches an optimal level of accuracy. Deep neural networks are used in a variety of applications, such as image and speech recognition, natural language processing, and recommendation systems. They are capable of learning from vast

amounts of data and can automatically extract features from raw data, making them a powerful tool for solving complex problems in a wide range of domains.

How Deep Neural Network Work Boston House Price Prediction is a common example used to illustrate how a deep neural network can work for regression tasks. The goal of this task is to predict the price of a house in Boston based on various features such as the number of rooms, crime rate, and accessibility to public transportation. Here's how a deep neural network can work for Boston House Price Prediction:

1. **Data preprocessing:** The first step is to preprocess the data. This involves normalizing the input features to have a mean of 0 and a standard deviation of 1, which helps the network learn more efficiently. The dataset is then split into training and testing sets.
2. **Model architecture:** A deep neural network is then defined with multiple layers. The first layer is the input layer, which takes in the normalized features. This is followed by several hidden layers, which can be deep or shallow. The last layer is the output layer, which predicts the house price.
3. **Model training:** The model is then trained using the training set. During training, the weights and biases of the nodes are adjusted based on the error between the predicted output and the actual output. This is done using an optimization algorithm such as stochastic gradient descent.
4. **Model evaluation:** Once the model is trained, it is evaluated using the testing set. The performance of the model is measured using metrics such as mean squared error or mean absolute error.
5. **Model prediction:** Finally, the trained model can be used to make predictions on new data, such as predicting the price of a new house in Boston based on its features.
6. By using a deep neural network for Boston House Price Prediction, we can obtain accurate predictions based on a large set of input features. This approach is scalable and can be used for other regression tasks as well.

Boston House Price Prediction Dataset-

Boston House Price Prediction is a well-known dataset in machine learning and is often used to demonstrate regression analysis techniques. The dataset contains information about 506 houses in Boston, Massachusetts, USA. The goal is to predict the median value of owner-occupied homes in thousands of dollars.

The dataset includes 13 input features, which are:

CRIM: per capita crime rate by town

ZN: proportion of residential land zoned for lots over 25,000 sq.ft.

INDUS: proportion of non-retail business acres per town

CHAS: Charles River dummy variable (1 if tract bounds river; 0 otherwise)

NOX: nitric oxides concentration (parts per 10 million)

RM: average number of rooms per dwelling

AGE: proportion of owner-occupied units built

prior to 1940

DIS: weighted distances to five
Boston employment centers

RAD: index of
accessibility to radial highways

TAX: full-value property-tax rate per \$10,000

PTRATIO: pupil-teacher ratio by town

B: $1000(B_k - 0.63)^2$ where B_k is the proportion of black people by town

LSTAT: % lower status of the population

The output variable is the median value of owner-occupied homes in thousands of dollars (MEDV).

To predict the median value of owner-occupied homes, a regression model is trained on the dataset. The model can be a simple linear regression model or a more complex model, such as a deep neural network.

After the model is trained, it can be used to predict the median value of owner-occupied homes based on the input features. The model's accuracy can be evaluated using metrics such as mean squared error or mean absolute error.

Boston House Price Prediction is an example of regression analysis and is often used to teach machine learning concepts. The dataset is also used in research to compare the performance of different regression models.

Conclusion- In this way we can Predict the Boston House Price using Deep Neural Network.

Assignment Question

1. What is Linear Regression?
2. What is a Deep Neural Network?
3. What is the concept of standardization?
4. Why split data into train and test?
5. Write Down Application of Deep Neural Network?

Attendance (5 Marks)	Conduction (5 Marks)	Oral (5 Marks)	Total (15 Marks)	Sign of Teacher

Source Code with Explanation-

#Importing the pandas for data processing and numpy for numerical computing

import numpy as np

import pandas as pd

Importing the Boston Housing dataset from the sklearn

from sklearn.datasets import load_boston

boston = load_boston()

#Converting the data into pandas dataframe

data = pd.DataFrame(boston.data)

#First look at the data

data.head()

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

#Adding the feature names to the dataframe

data.columns = boston.feature_names

#Adding the target variable to the dataset

data['PRICE'] = boston.target

#Looking at the data with names and target variable

data.head(n=10)

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21	28.7
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43	22.9
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15	27.1
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93	16.5
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	17.10	18.9

#Shape of the data

```
print(data.shape)
```

#Checking the null values in the dataset

```
data.isnull().sum()
```

CRIM 0

ZN 0

INDUS 0

CHAS 0

NOX 0

RM 0

AGE 0

DIS 0

RAD 0

TAX 0

PTRATIO 0

B 0

LSTAT 0

PRICE 0

dtype: int64

#Checking the statistics of the data

```
data.describe()
```

This is sometimes very useful, for example if you look at the CRIM the max is

88.97 and 75% of the value is below 3.677083 and

mean is 3.613524 so it means the max values is actually an outlier or there are outliers present in the column

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000

```
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
# Column Non-Null Count Dtype
```

```
-----
0 CRIM 506 non-null float64
1 ZN 506 non-null float64
2 INDUS 506 non-null float64
3 CHAS 506 non-null float64
4 NOX 506 non-null float64
5 RM 506 non-null float64
6 AGE 506 non-null float64
7 DIS 506 non-null float64
8 RAD 506 non-null float64
9 TAX 506 non-null float64
10 PTRATIO 506 non-null float64
11 B 506 non-null float64
12 LSTAT 506 non-null float64
13 PRICE 506 non-null float64
```

```
dtypes: float64(14)
memory usage: 55.5 KB
```

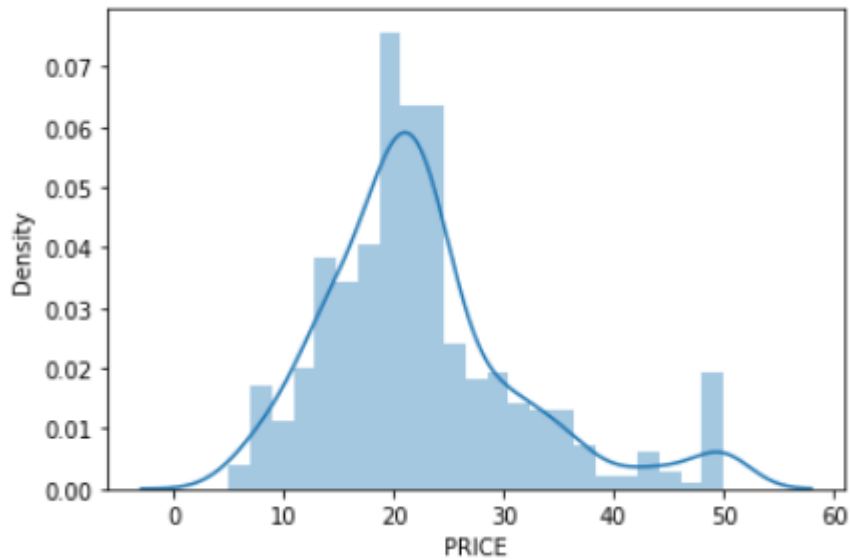
#checking the distribution of the target variable

```
import seaborn as sns
```

```
sns.distplot(data.PRICE)
```

#The distribution seems normal, has not be the data normal we would have perform log transformation or took to square root of the data to make the data normal.
Normal distribution is need for the machine learning for better predictibilty of the model

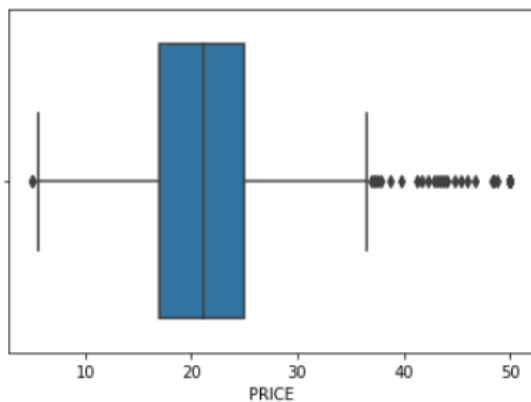
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f44d082c670>
```



#Distribution using box plot

```
sns.boxplot(data.PRICE)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f44d077ed60>
```



#Checking the correlation of the independent feature with the dependent feature

Correlation is a statistical technique that can show whether and how strongly pairs of variables are related. An intelligent correlation analysis can lead to a greater understanding of your data

#checking Correlation of the data

```
correlation = data.corr()
```

```
correlation.loc['PRICE']
```

```
CRIM -0.388305
```

```
ZN 0.360445
```

```
INDUS -0.483725
```

```
CHAS 0.175260
```

```
NOX -0.427321
```

```
RM 0.695360
```


AGE -0.376955

DIS 0.249929

RAD -0.381626

TAX -0.468536

PTRATIO -0.507787

B 0.333461

LSTAT -0.737663

PRICE 1.000000

Name: PRICE, dtype: float64

plotting the heatmap

import matplotlib.pyplot as plt

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

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

By looking at the correlation plot LSTAT is negatively correlated with -0.75 and RM is positively correlated to the price and PTRATIO is correlated negatively with -0.51



Checking the scatter plot with the most correlated features

plt.figure(figsize = (20,5))

features = ['LSTAT','RM','PTRATIO']

for i, col in enumerate(features):

plt.subplot(1, len(features) , i+1)

x = data[col]

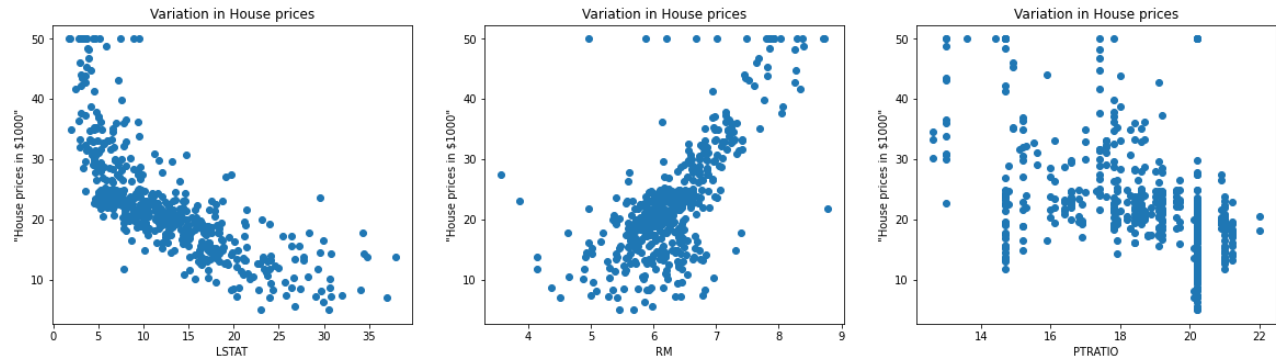
y = data.PRICE

plt.scatter(x, y, marker='o')

plt.title("Variation in House prices")

plt.xlabel(col)

plt.ylabel("House prices in \$1000")



Splitting the dependent feature and independent feature

```
#X = data[['LSTAT','RM','PTRATIO']]
```

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

```
y= data.PRICE
```

In order to provide a standardized input to our neural network, we need the perform the normalization of our dataset.

This can be seen as an step to reduce the differences in scale that may arise from the existent features.

We perform this normalization by subtracting the mean from our data and dividing it by the standard deviation.

One more time, this normalization should only be performed by using the mean and standard deviation from the training set,

in order to avoid any information leak from the test set.

```
mean = X_train.mean(axis=0)
```

```
std = X_train.std(axis=0)
```

```
X_train = (X_train - mean) / std
```

```
X_test = (X_test - mean) / std
```

#Linear Regression

```
from sklearn.linear_model import LinearRegression
```

```
regressor = LinearRegression()
```

#Fitting the model

```
regressor.fit(X_train,y_train)
```

Model Evaluation

#Prediction on the test dataset

```
y_pred = regressor.predict(X_test)
```

Predicting RMSE the Test set results

```
from sklearn.metrics import mean_squared_error
```

```
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
print(rmse)
```

```
from sklearn.metrics import r2_score
```

```
r2 = r2_score(y_test, y_pred)
```

```
print(r2)
```

Neural Networks

#Scaling the dataset

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

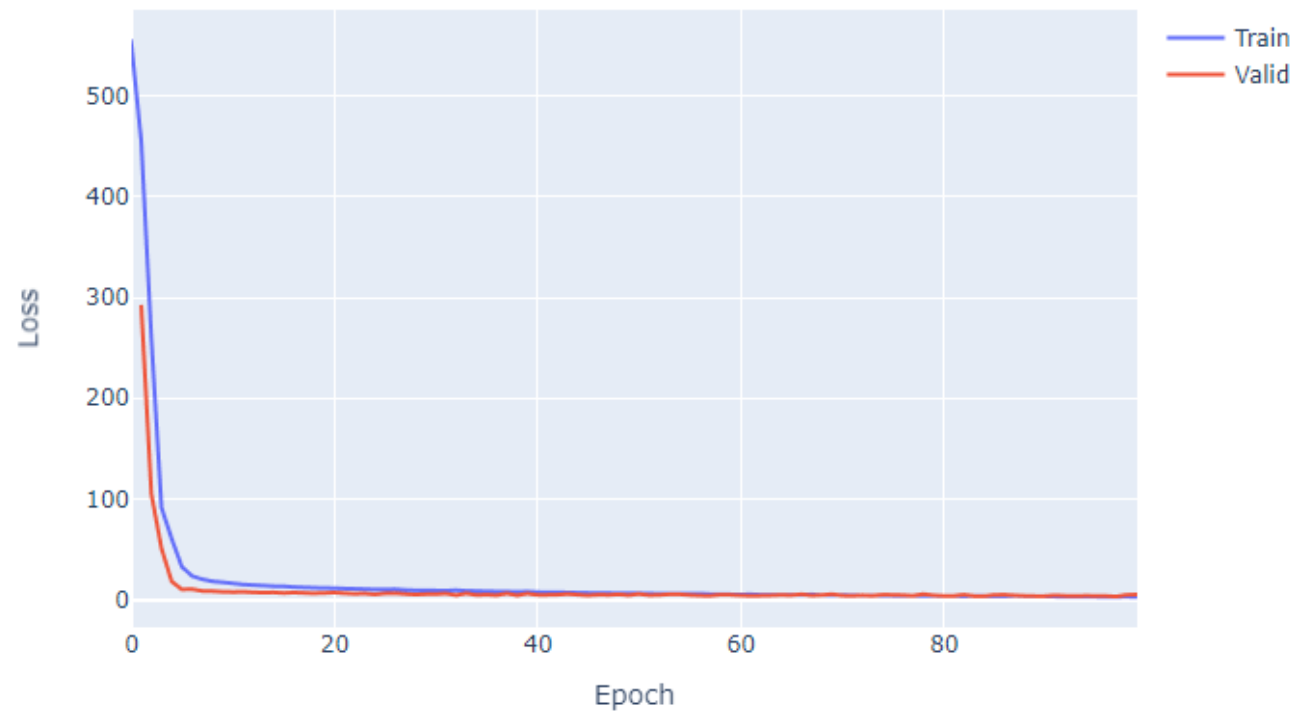
```

# Due to the small amount of presented data in this dataset, we must be careful
to not create an overly complex model,
# which could lead to overfitting our data. For this, we are going to adopt an
architecture based on two Dense layers,
# the first with 128 and the second with 64 neurons, both using a ReLU activation
function.
# A dense layer with a linear activation will be used as output layer.
# In order to allow us to know if our model is properly learning, we will use a
mean squared error loss function and to report the performance of it we will
adopt the mean average error metric.
# By using the summary method from Keras, we can see that we have a total of
10,113 parameters, which is acceptable for us.
#Creating the neural network model
import keras
from keras.layers import Dense, Activation, Dropout
from keras.models import Sequential
model = Sequential()
model.add(Dense(128,activation = 'relu',input_dim =13))
model.add(Dense(64,activation = 'relu'))

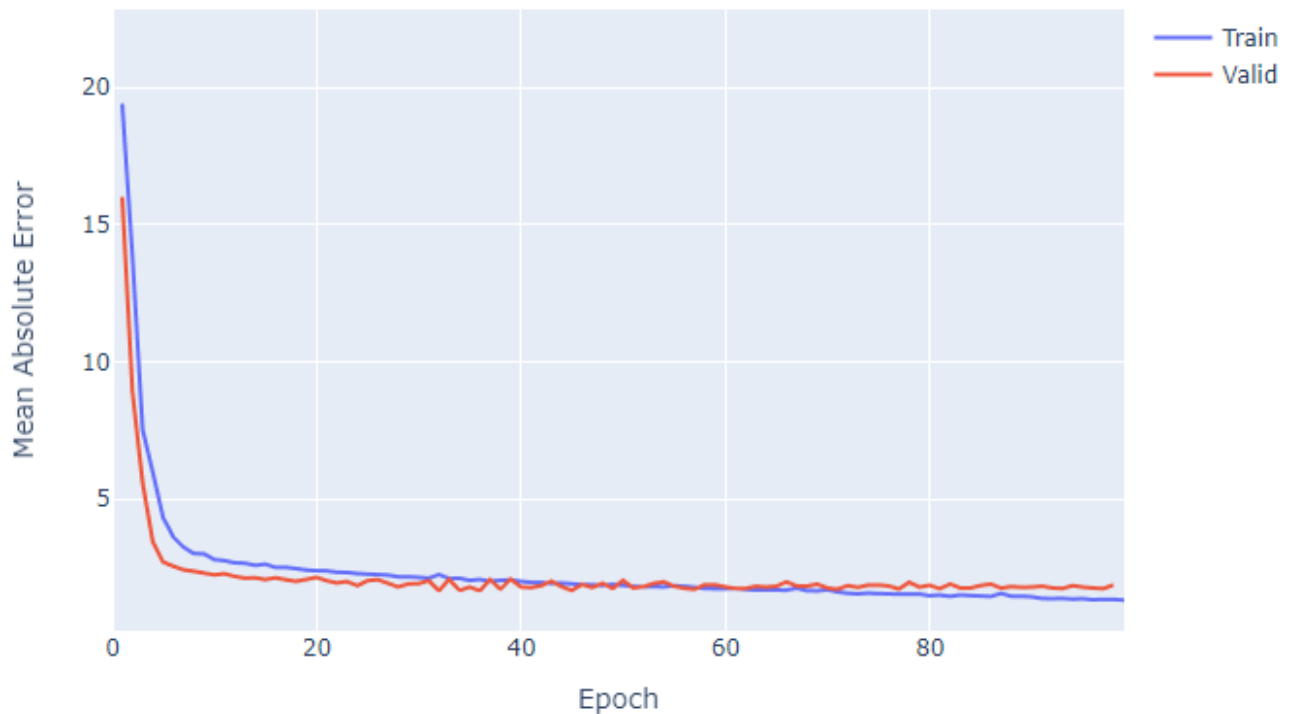
model.add(Dense(32,activation = 'relu'))
model.add(Dense(16,activation = 'relu'))
model.add(Dense(1))
#model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.compile(optimizer = 'adam',loss = 'mean_squared_error',metrics=['mae'])
!pip install ann_visualizer
!pip install graphviz
from ann_visualizer.visualize import ann_viz;
#Build your model here
ann_viz(model, title="DEMO ANN");
history = model.fit(X_train, y_train, epochs=100, validation_split=0.05)
# By plotting both loss and mean average error, we can see that our model was
capable of learning patterns in our data without overfitting taking place (as
shown by the validation set curves)
from plotly.subplots import make_subplots
import plotly.graph_objects as go
fig = go.Figure()
fig.add_trace(go.Scattergl(y=history.history['loss'],
name='Train'))
fig.add_trace(go.Scattergl(y=history.history['val_loss'],
name='Valid'))
fig.update_layout(height=500, width=700,
axis_title='Epoch',
yaxis_title='Loss')
fig.show()

```

```
fig = go.Figure()
```



```
fig.add_trace(go.Scattergl(y=history.history['mae'],  
name='Train'))  
fig.add_trace(go.Scattergl(y=history.history['val_mae'],  
name='Valid'))  
fig.update_layout(height=500, width=700,  
xaxis_title='Epoch',  
yaxis_title='Mean Absolute Error')  
fig.show()
```



#Evaluation of the model

```
y_pred = model.predict(X_test)
mse_nn, mae_nn = model.evaluate(X_test, y_test)
print('Mean squared error on test data: ', mse_nn)
print('Mean absolute error on test data: ', mae_nn)
4/4 [=====] - 0s 4ms/step - loss: 10.5717 - mae: 2.2670
Mean squared error on test data: 10.5717334731445
Mean absolute error on test data: 2.2669904232025146
#Comparison with traditional approaches
#First let's try with a simple algorithm, the Linear Regression:
from sklearn.metrics import mean_absolute_error
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
```

```
y_pred_lr = lr_model.predict(X_test)
mse_lr = mean_squared_error(y_test, y_pred_lr)
mae_lr = mean_absolute_error(y_test, y_pred_lr)
print('Mean squared error on test data: ', mse_lr)
print('Mean absolute error on test data: ', mae_lr)
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
0.8812832788381159
# Predicting RMSE the Test set results
from sklearn.metrics import mean_squared_error
```

```

rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
print(rmse)
3.320768607496587
# Make predictions on new data
import sklearn
new_data = sklearn.preprocessing.StandardScaler().fit_transform([[0.1, 10.0,
5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300, 10]])
prediction = model.predict(new_data)
print("Predicted house price:", prediction)
1/1 [=====] - 0s 70ms/step
Predicted house price: [[11.104753]]
#new_data =

```

sklearn.preprocessing.StandardScaler().fit_transform([[0.1, 10.0, 5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300, 10]]) is a line of code that standardizes the input features of a new data point.

In this specific case, we have a new data point represented as a list of 13 numeric values ([0.1, 10.0, 5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300, 10]) that represents the values for the 13 features of the Boston House Price dataset.

The StandardScaler() function from the sklearn.preprocessing module is used to standardize the data. Standardization scales each feature to have zero mean and unit variance, which is a common preprocessing step in machine learning to ensure that all features contribute equally to the model.

The fit_transform() method is used to fit the scaler to the data and apply the standardization transformation. The result is a new data point with standardized feature values.