Deep Learning

Roll No. :- 19121028

Assignment No: 1

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 pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Load the data from the CSV file
data = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data",
                   delim_whitespace=True, header=None, names=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PT

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(data.drop('MEDV', axis=1), data['MEDV'], test_size=0.2, random_state=42)

# Define the linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 24.291119474973478
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import StandardScaler

# Load the Boston Housing dataset
data = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data",
                   delim_whitespace=True, header=None, names=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PT
```

```
data.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

```
data.columns
```

```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'MEDV'],
      dtype='object')
```

```
data['MEDV']
```

```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
```

```
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: MEDV, Length: 506, dtype: float64
```

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

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

```
(506, 14)
```

```
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
MEDV       0
dtype: int64
```

```
data.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.

```
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    int64
 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    int64
 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  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

```
#checking the distribution of the target variable
import seaborn as sns
sns.distplot(data.MEDV)
```

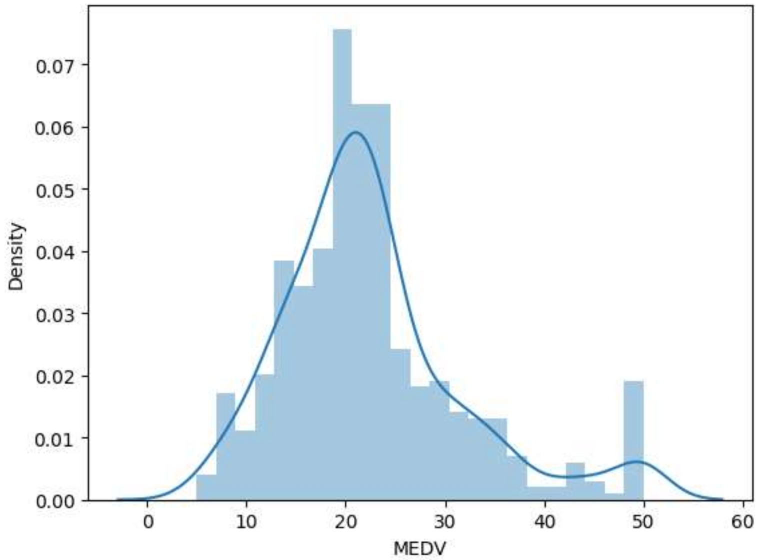
```
<ipython-input-11-366a0ef1c28f>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data.MEDV)
<Axes: xlabel='MEDV', ylabel='Density'>
```




```
#Distribution using box plot
sns.boxplot(data.MEDV)
```

```
<Axes: >
```

```
correlation = data.corr()
correlation.loc['MEDV']
```

```
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
MEDV       1.000000
Name: MEDV, dtype: float64
```

```
# plotting the heatmap
import matplotlib.pyplot as plt
fig,axes = plt.subplots(figsize=(15,12))
sns.heatmap(correlation,square = True,annot = True)
```

```
<Axes: >
```


```
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.MEDV
    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.MEDV
```

```
# 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 meanand 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)
```

```
▾ LinearRegression
LinearRegression()
```

```
# 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)
```

```
4.928602182665338
```

```
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)
```

```
0.6687594935356318
```

```
# Due to the small amount of presented data in this dataset, we must be carefulto not create an overly complex model,
# which could lead to overfitting our data. For this, we are going to adopt anarchitecture based on two Dense layers,
# the first with 128 and the second with 64 neurons, both using a ReLU activationfunction.
# 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 amean squared error loss function and to report the perform
# By using the summary method from Keras, we can see that we have a total of10,113 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;
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting ann_visualizer
  Downloading ann_visualizer-2.5.tar.gz (4.7 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: ann_visualizer
  Building wheel for ann_visualizer (setup.py) ... done
  Created wheel for ann_visualizer: filename=ann_visualizer-2.5-py3-none-any.whl size=4167 sha256=b1b4c05c91208873a49ce351c16a7a555
  Stored in directory: /root/.cache/pip/wheels/6e/0f/ae/f5dba91db71b1b32bf03d0ad18c32e86126093aba5ec6b6488
Successfully built ann_visualizer
Installing collected packages: ann_visualizer
Successfully installed ann_visualizer-2.5
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (0.20.1)
```

```
#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 wascapable of learning patterns in our data without overfitting
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,
xaxis_title='Epoch',
yaxis_title='Loss')
fig.show()
```

```
Epoch 1/100
12/12 [==============================] - 7s 22ms/step - loss: 580.5351 - mae: 22.2160 - val_loss: 46
Epoch 2/100
12/12 [==============================] - 0s 15ms/step - loss: 466.3240 - mae: 19.6319 - val_loss: 31
Epoch 3/100
12/12 [==============================] - 0s 11ms/step - loss: 244.4679 - mae: 13.2928 - val_loss: 10
Epoch 4/100
12/12 [==============================] - 0s 9ms/step - loss: 89.8134 - mae: 7.3993 - val_loss: 74.14
Epoch 5/100
12/12 [==============================] - 0s 12ms/step - loss: 58.6862 - mae: 5.8714 - val_loss: 58.4
Epoch 6/100
12/12 [==============================] - 0s 16ms/step - loss: 38.4400 - mae: 4.5324 - val_loss: 51.6
Epoch 7/100
12/12 [==============================] - 0s 13ms/step - loss: 28.2537 - mae: 3.9522 - val_loss: 50.3
Epoch 8/100
12/12 [==============================] - 0s 14ms/step - loss: 24.3619 - mae: 3.6402 - val_loss: 50.3
Epoch 9/100
12/12 [==============================] - 0s 16ms/step - loss: 21.8563 - mae: 3.4186 - val_loss: 48.4
Epoch 10/100
12/12 [==============================] - 0s 14ms/step - loss: 19.7121 - mae: 3.2665 - val_loss: 46.0
Epoch 11/100
12/12 [==============================] - 0s 12ms/step - loss: 18.2454 - mae: 3.1222 - val_loss: 45.1
Epoch 12/100
12/12 [==============================] - 0s 11ms/step - loss: 16.8629 - mae: 2.9965 - val_loss: 44.4
Epoch 13/100
12/12 [==============================] - 0s 13ms/step - loss: 15.6940 - mae: 2.8980 - val_loss: 42.2
Epoch 14/100
12/12 [==============================] - 0s 19ms/step - loss: 14.6472 - mae: 2.7925 - val_loss: 38.9
Epoch 15/100
12/12 [==============================] - 0s 20ms/step - loss: 13.9116 - mae: 2.7373 - val_loss: 38.6
Epoch 16/100
12/12 [==============================] - 0s 9ms/step - loss: 13.2498 - mae: 2.6521 - val_loss: 38.69
Epoch 17/100
12/12 [==============================] - 0s 14ms/step - loss: 12.6561 - mae: 2.6163 - val_loss: 35.6
Epoch 18/100
12/12 [==============================] - 0s 12ms/step - loss: 12.2746 - mae: 2.5564 - val_loss: 36.8
Epoch 19/100
12/12 [==============================] - 0s 12ms/step - loss: 11.7560 - mae: 2.5261 - val_loss: 34.2
```

```
#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
4/4 [==============================] - 0s 4ms/step - loss: 9.5319 - mae: 1.9659
Mean squared error on test data:  9.531891822814941
Mean absolute error on test data:  1.9659347534179688
```

```
#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)
```

```
Mean squared error on test data:  24.291119474973513
Mean absolute error on test data:  3.1890919658878474
0.8700204594160786
```

```
# 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)
```

```
3.0873761423674453
```

```
# 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 22ms/step
Predicted house price: [[11.541498]]
```