LP-5 DEEP LEARNING Practical 1

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.

Name: **ONASVEE BANARSE**

Rollno: **09**

BE COMP 1

Dataset

The dataset used in this project comes from the UCI Machine Learning Repository. This data was collected in 1978 and each of the 506 entries represents aggregate information about 14 features of homes from various suburbs located in Boston.

The features can be summarized as follows:

- CRIM: This is the per capita crime rate by town
- ZN: This is the proportion of residential land zoned for lots larger than 25,000 sq.ft.
- INDUS: This is the proportion of non-retail business acres per town.
- CHAS: This is the Charles River dummy variable (this is equal to 1 if tract bounds river; 0 otherwise)
- NOX: This is the nitric oxides concentration (parts per 10 million)
- RM: This is the average number of rooms per dwelling
- AGE: This is the proportion of owner-occupied units built prior to 1940
- DIS: This is the weighted distances to five Boston employment centers
- RAD: This is the index of accessibility to radial highways
- TAX: This is the full-value property-tax rate per 1000 bucks
- PTRATIO: This is the pupil-teacher ratio by town
- B: This is calculated as $1000(Bk — 0.63)^2$, where Bk is the proportion of people of African American descent by town
- LSTAT: This is the percentage lower status of the population
- MEDV: This is the median value of owner-occupied homes in 1000s

Importing libraries and the dataset

In [1]:
```
#Importing the pandas for data processing and numpy for numerical computing
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
# Importing the Boston Housing dataset from the sklearn
from sklearn.datasets import load_boston
boston = load_boston()
```

In [3]:
```python
#Converting the data into pandas dataframe
data = pd.DataFrame(boston.data)
```

First look at the dataset

In [4]:
```python
#First look at the data
data.head()
```

Out[4]:

	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

In [5]:
```python
#Adding the feature names to the dataframe
data.columns = boston.feature_names
```

In [6]:
```python
#Adding the target variable to the dataset
data['PRICE'] = boston.target
```

In [7]:
```python
#Looking at the data with names and target variable
data.head()
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
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

In [8]:
```python
#Shape of the data
print(data.shape)
```

```
(506, 14)
```

In [9]:
```python
#Checking the null values in the dataset
data.isnull().sum()
```

```
Out[9]: 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
```

No null values in the dataset, no missing value treatement needed

```
In [10]: #Checking the statistics of the data
data.describe()
```

Out[10]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.79
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.10
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.12
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.10
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.20
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.18
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.12

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

```
In [11]: 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
```

Visualisation

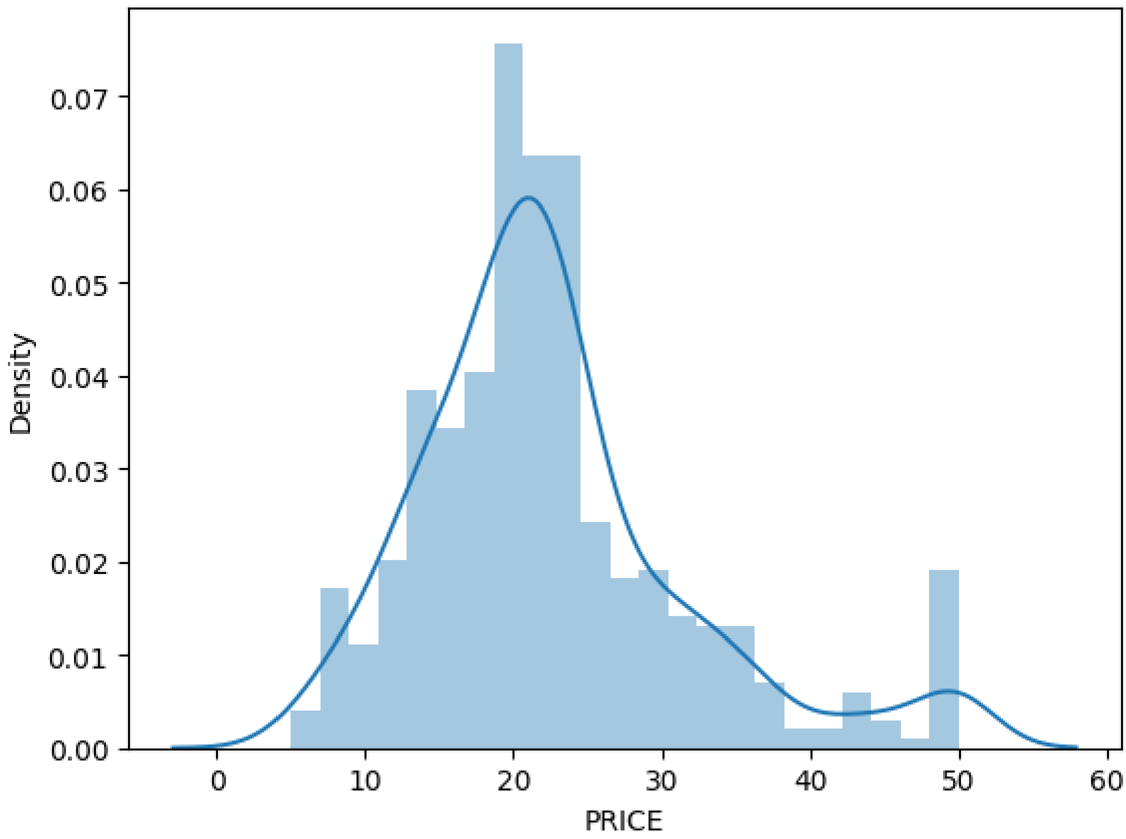
In [12]:

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

```
C:\Users\ORIONORIGINAL\AppData\Local\Temp\ipykernel_8836\4212025153.py:3: UserWarn
ing: 

`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.PRICE)
```

Out[12]:

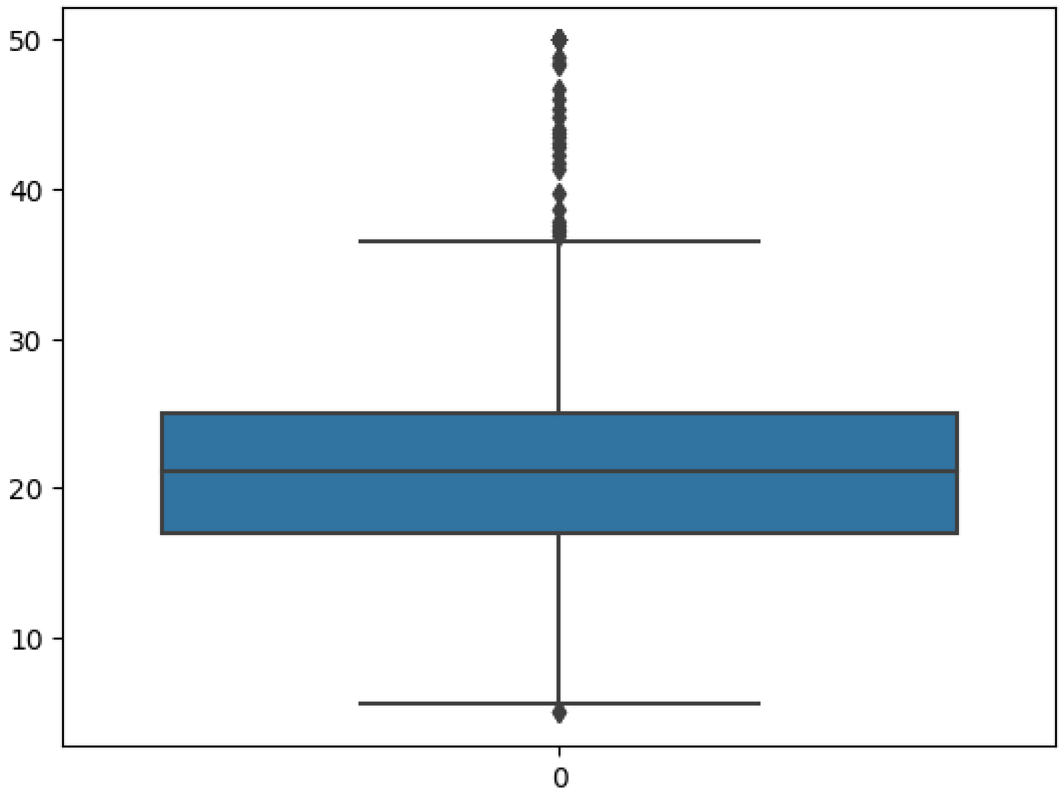
```
<AxesSubplot: xlabel='PRICE', ylabel='Density'>
```



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 predictiblity of the model

In [13]:
```
#Distribution using box plot
sns.boxplot(data.PRICE)
```

Out[13]: <AxesSubplot: >



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

In [14]:
```
#checking Correlation of the data
correlation = data.corr()
correlation.loc['PRICE']
```

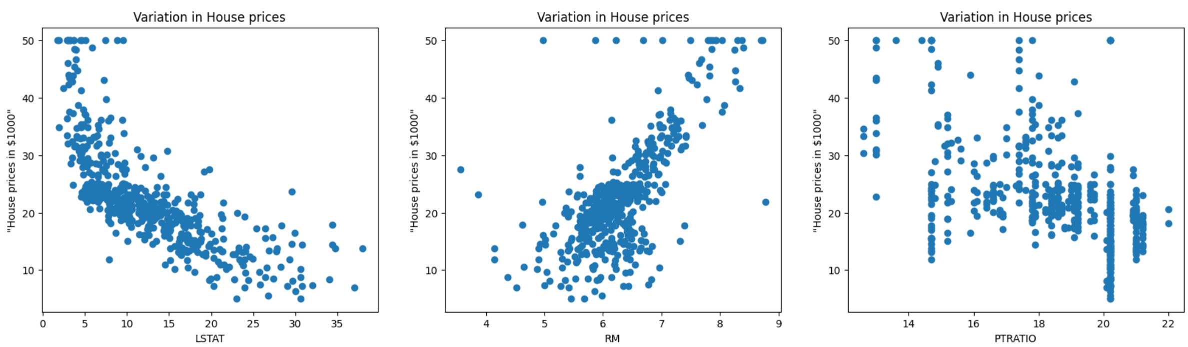
Out[14]:
```
CRIM      -0.388305
ZN         0.360445
INDUS     -0.483725
CHAS       0.175260
NOX       -0.427321
RM         0.695360
AGE       -0.376955
DIS        0.249929
RAD       -0.381626
TAX       -0.468536
PTRATIO   -0.507787
B          0.333461
LSTAT     -0.737663
PRICE      1.000000
Name: PRICE, dtype: float64
```

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

Out[15]:
```
<AxesSubplot: >
```

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

In [16]:
```
# 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

In [17]:
```
#X = data[['LSTAT','RM','PTRATIO']]
X = data.iloc[:,:-1]
y= data.PRICE
```

Splitting the data for Model Validation

In [18]:
```
# Splitting the data into train and test for building the model
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_st
```

Building the Model

In [19]:
```
#Linear Regression
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
```

In [20]:
```
#Fitting the model
regressor.fit(X_train,y_train)
```

Out[20]:
```
▾ LinearRegression
LinearRegression()
```

Model Evaluation

In [21]:
```
#Prediction on the test dataset
y_pred = regressor.predict(X_test)
```

In [22]:
```
# 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)
```

```
5.041784121402059
```

In [23]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
```

```
0.7263451459702501
```

Neural Networks

In [24]:
```
#Scaling the dataset
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [51]:
```
#Creating the neural network model
import keras
```

```python
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 = 'mean_squared_error')
```

In [ ]:
```python
results=model.fit(X_train, y_train, epochs = 100)
```

Evaluation of the model

In [53]:
```python
y_pred = model.predict(X_test)
```

```
4/4 [==============================] - 0s 1ms/step
```

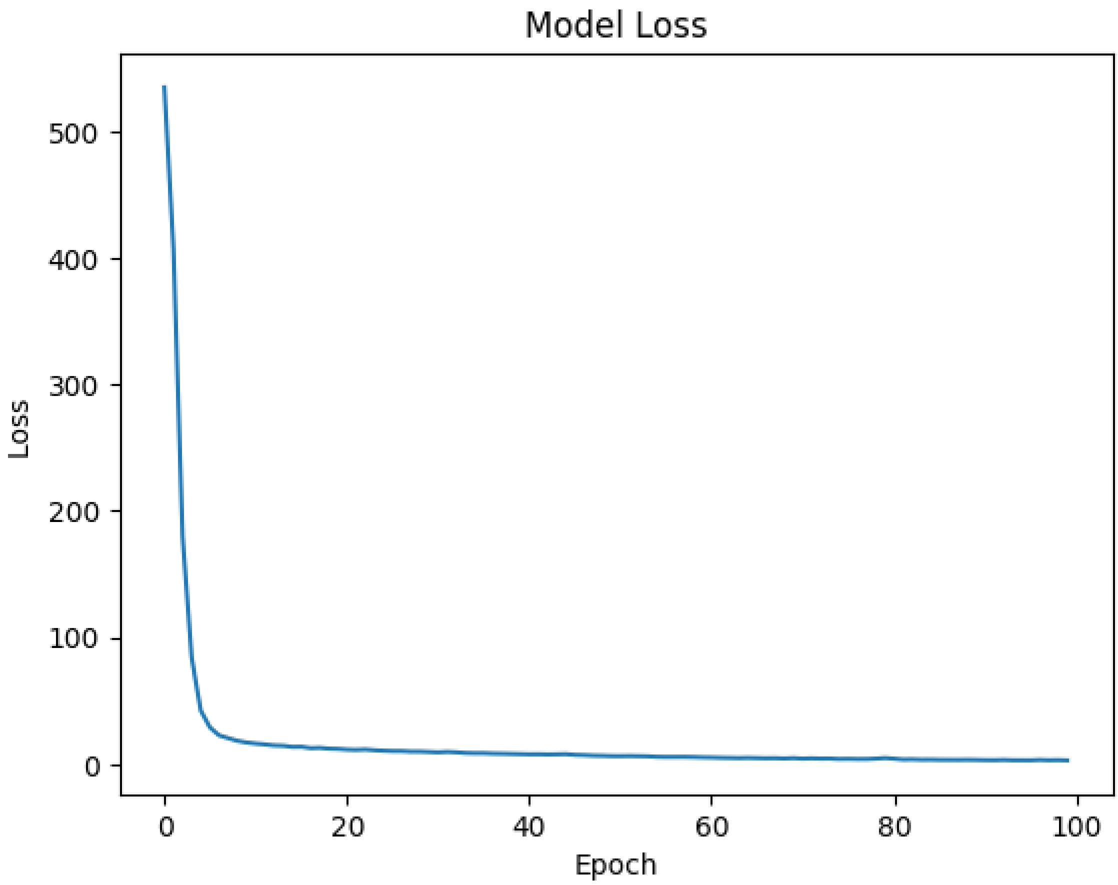
In [54]:
```python
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
```

```
0.897062774233037
```

In [55]:
```python
# 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)
```

```
3.0922094134897433
```

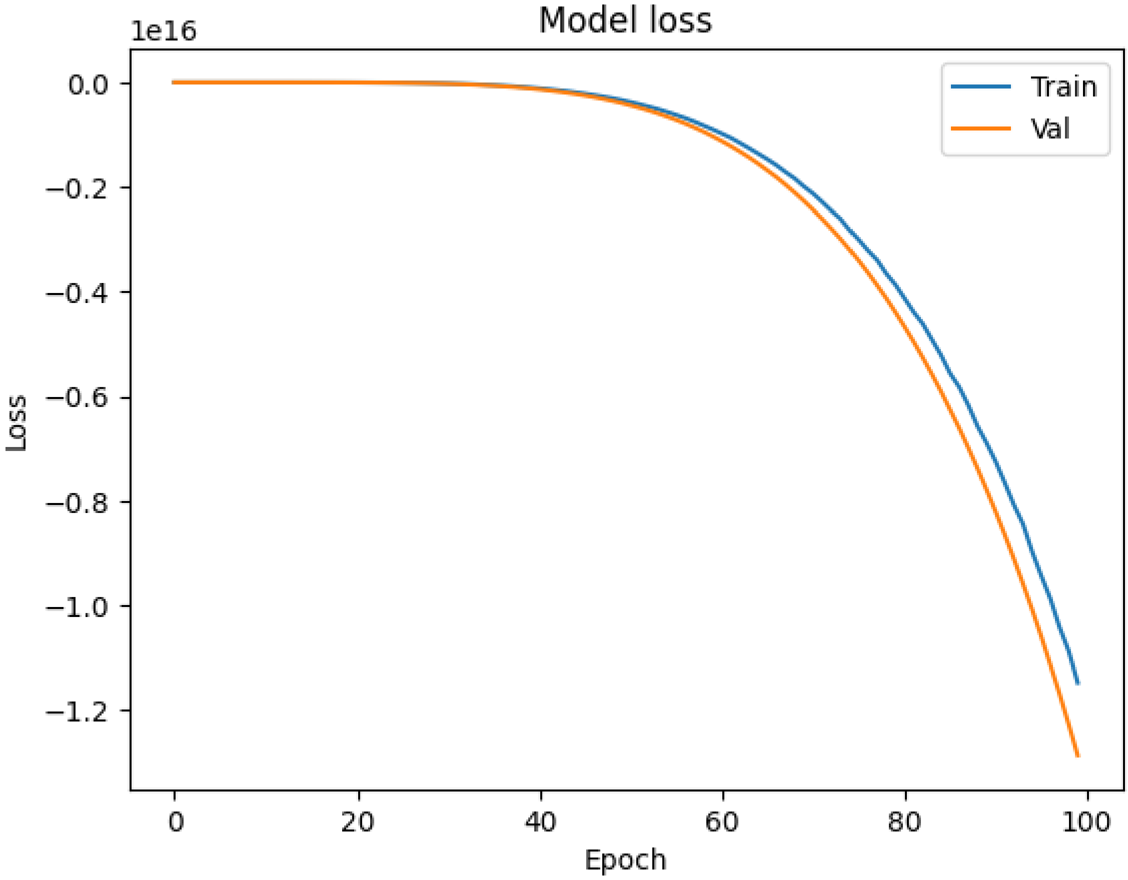
In [58]:
```python
plt.plot(results.history['loss'])
plt.title('Model Loss')
plt.ylabel('Loss ')
plt.xlabel('Epoch')
plt.show()
```



In [64]:
```
from keras.layers import Dropout
from keras import regularizers


model_3 = Sequential([
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01), input_
    Dropout(0.3),
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1, activation='sigmoid', kernel_regularizer=regularizers.l2(0.01)),
])
```

In [ ]:
```
model_3.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
hist_3 = model_3.fit(X_train, y_train,
          batch_size=32, epochs=100,
          validation_data=(X_test, y_test))
```

In [71]:
```
plt.plot(hist_3.history['loss'])
plt.plot(hist_3.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```



Model loss
1e16
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
Loss
0
20
40
60
80
100
Epoch
Train
Val

In [ ]: