QUESTION 1 : Image Classification

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The data is loaded using keras.datasets.cifar10. In this we are considering 20 percent of the entire training dataset to be out current training data. In the 10000 test images we have we are making a 50:50 split to create a validation dataset which will help us to better optimize the model and tune the hyper parameters.

PREPROCESSING: The pixel values in images must be scaled prior to providing the images as input to a deep learning neural network model for evaluation. first convert the data type from unsigned integers to floats, then divide the pixel values by the maximum value, which scales the input to lie between 0-1 We also randomize the input that is fed to the neural network in order to train the model better.

The common attributes considered for training our model is the number of epochs, batch size, input shape.

OUTPUT LAYER AND LOSS FUNCTION: Since we have one hot encoded the target value we are considering the output layer to be a dense layer with ten neurons with the softmax activation function. The loss function that is used for this model is categorical cross entropy and the metric considered is categorical accuracy. The reason for choosing softmax as the activation function and categorical cross entropy as a loss function is because the probelm we have considered is a multi-class classification problem and the softmax activation gives the probability distribution over all the classes whereas categorical cross entropy is designed to quantify the difference between two probability distributions.

EFFECT OF NUMBER OF LAYERS AND NUMBER OF NEURONS IN MLP: We are considering the batch size (32) , the number of epochs (5) to be constant and train the different models with different number of neurons and layers.

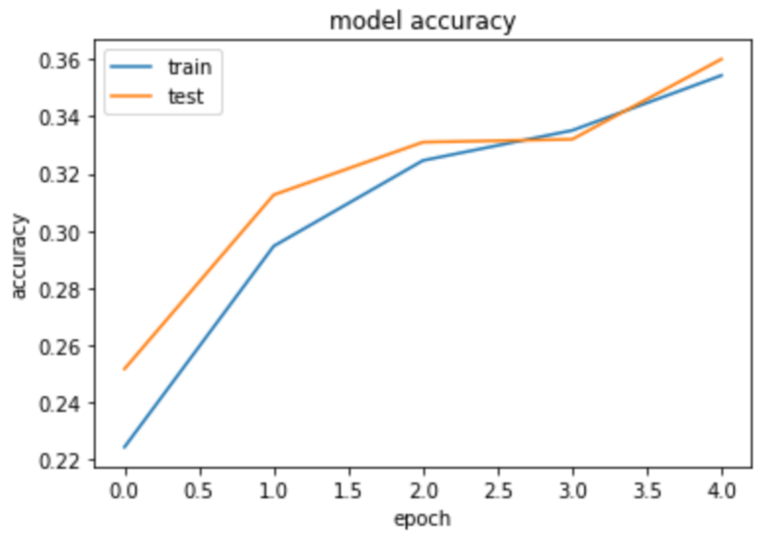
From the tabulated result we are able to observe that the single layer model itself provides us with a good accuracy than deeper models that results in overfitting of the data. For this problem a single dense layer with optimal number of neurons in this scenario 256 or 512 neurons better observes the features and provides a better result. We can see from the table that as the number of neurons exceed the the training accuracy that was gradually increasing for smaller number of neurons starts decreasing when the number of neurons are given to be 1024/2048 etc. Same is the scenario with the increasing number of dense layers.

Loss is also minimized for a single layer model with optimal number of neuron [256/512].Loss begins to increase as the number of layers and neurons are increased because of overfitting of data.

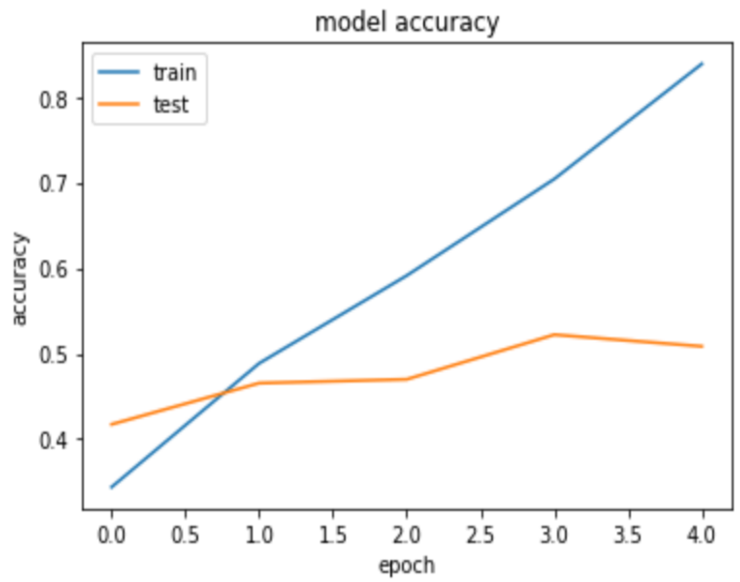
PERFORMANCE EVALUATION OF THE THREE NETWORKS (MLP vs CNN): MLP performs the least by providing a lower train and test accuracy. Whereas the first CNN model that has convolution layer and dense layer performs better than MLP and provides the best train accuracy. The second CNN model with convolution layer, max pooling layer, dropout layer and dense layers performs the best by providing a stable train accuracy and a better validation and test accuracy.

The plots also approves the same.

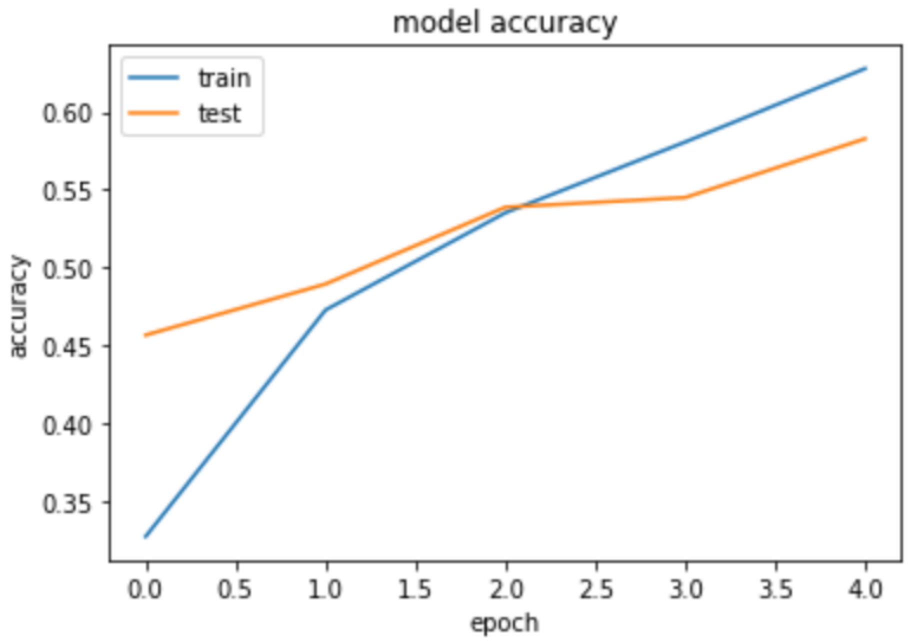
MLP: (TEST ACCURACY : 36.03)



CNN-1: (TEST ACCURACY : 52.06)



CNN-2: (TEST ACCURACY : 58.84)



	MLP	**CNN-1**	**CNN-2**
Train accuracy	35.43	83.86	50.84
Validation accuracy	36	62.79	58.28
Test accuracy	36.03	52.06	58.84

CNN performs better than MLP because the convolution layer and the max pooling layers present in the CNN helps to better extract the feature and CNN understands the spatial relation between nearby pixels of image and between pixels of images.

COMPARISION BETWEEN THE TWO CNN MODELS:

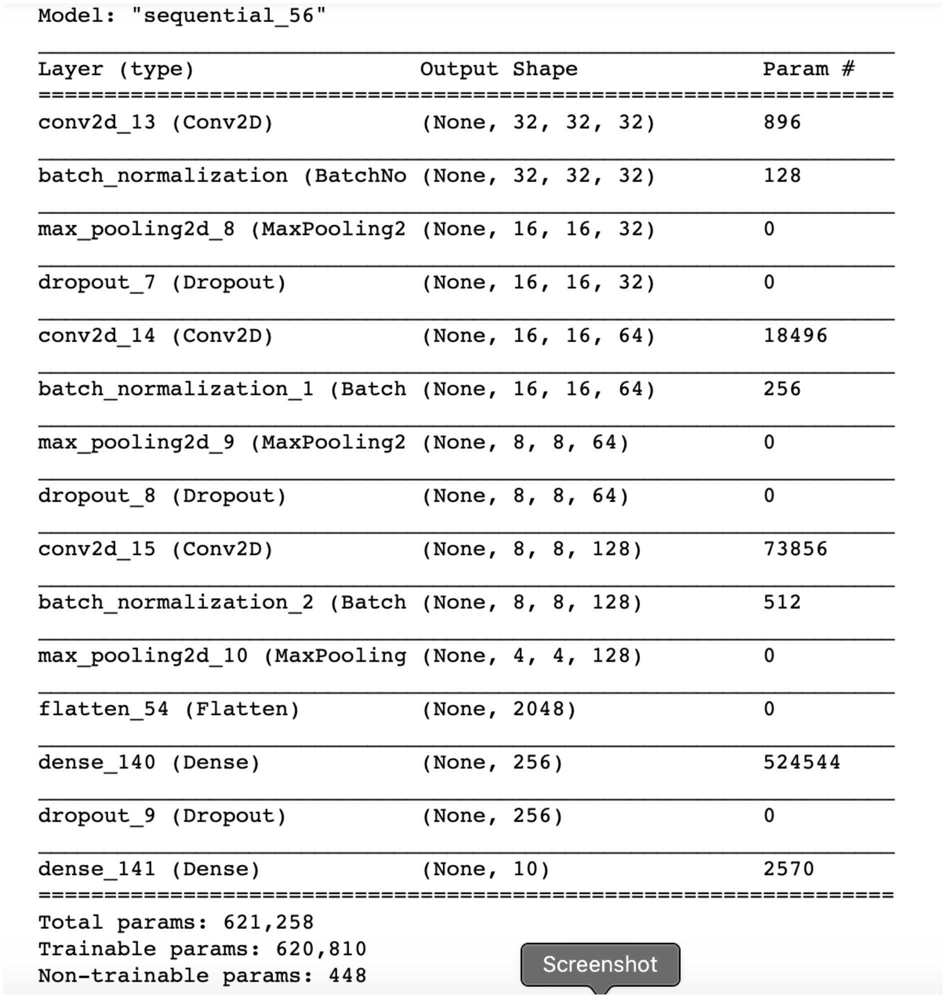
The first cnn model has two convolution layer and two dense layer whereas the second cnn model has two convolution layer followed by the max pool layer and two dense layer followed by the dropout layer. The second model performs better because the number of parameters trained by the first model is reduced in the second model by the usage of max pooling layer and dropout layer which substantially reduces the trainable parameters and helps to fit the model better without any under fitting or over fitting. The CNN-1 approximately takes 56s /epoch to train the inputs and CNN-2 takes approximately 11s /epoch to train the model because of the lightweight approach followed.

When you further increase the epochs, we can observe that the training accuracy increases rapidly and achieves 95-100 percent but the validation accuracy gets saturated after reaching certain point [60-70 percent may be obtained].The reason is that the loss doesn’t reduce that as as the accuracy improves so the trainable parameters doesn’t change much and hence has the least effect in prediction of the test results.

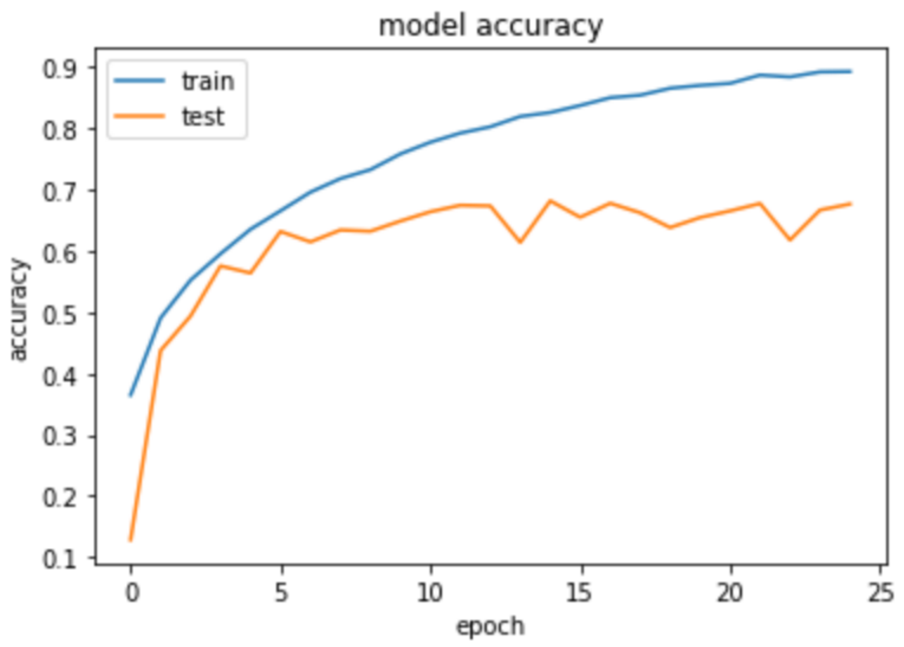
IMPROVEMENT OF THE NETWORK TO IMPROVE THE MODEL PERFORMANCE:

We can improve the model by adding more convolution layer followed by batch normalization layer which will help us to solve the problem of vanishing gradient and helps us to train the features better. The convolution layer along with the batch norm layer is followed by the max pooling layer. Similar type of optimization on the trainable parameter is done by adding a dropout layer after every dense layer. The model I have implemented using the above idea after running for 25 epochs achieved a training accuracy of 89 percent and validation and test accuracy of approximately 67 percent which is a good improvement from the implemented CNN model 1 and 2.

The architecture of the above model:



```
Model: "sequential_56"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_8 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
flatten_54 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_140 (Dense)            (None, 256)               524544
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_141 (Dense)            (None, 10)                2570
=================================================================
Total params: 621,258
Trainable params: 620,810
Non-trainable params: 448
```



QUESTION 2 : RNN for Regression

DATASET CREATION: DataCreation()

We are loading the data from the given csv file “q2_dataset” and creating the dataset with only the required features which are open,high,low prices and their volumes. We are considering the step size to be three which implies that in order to predict the current day’s opening price we will consider the features of the past three days. So, as a result we finally get 12 feature input and one target value. Totally we had 1259 inputs and after considering the step size we will get 1256 records as our new input. Here we are randomizing the data and splitting the data in the ratio of 7:3 framing the train and the test dataset and loading it in their corresponding csv file namely “train_data_RNN.csv” and “test_data_RNN.csv”.

PREPROCESSING THE DATA:

We are normalizing the data to lie between o-1 by applying minmaxscaler fit transform function from the sklearn library. The training data’s features are fit transformed using the minmax scaler and a pickle file is created to store the scaler object so that a similar type of fit transformation can be applied on the test data as well to obtain precise result. The pickle file created in the train_RNN.PY “scalerobj.pickle” is loaded in the test_RNN.py file to perform the above mentioned normalization on the test data. The input is reshaped to three dimensional array as the Recurrent neural networks accepts three dimensional inputs. So, we are considering the input shape of (None, 3, 4).

DESIGN STEPS TO FIND THE BEST NETWORK:

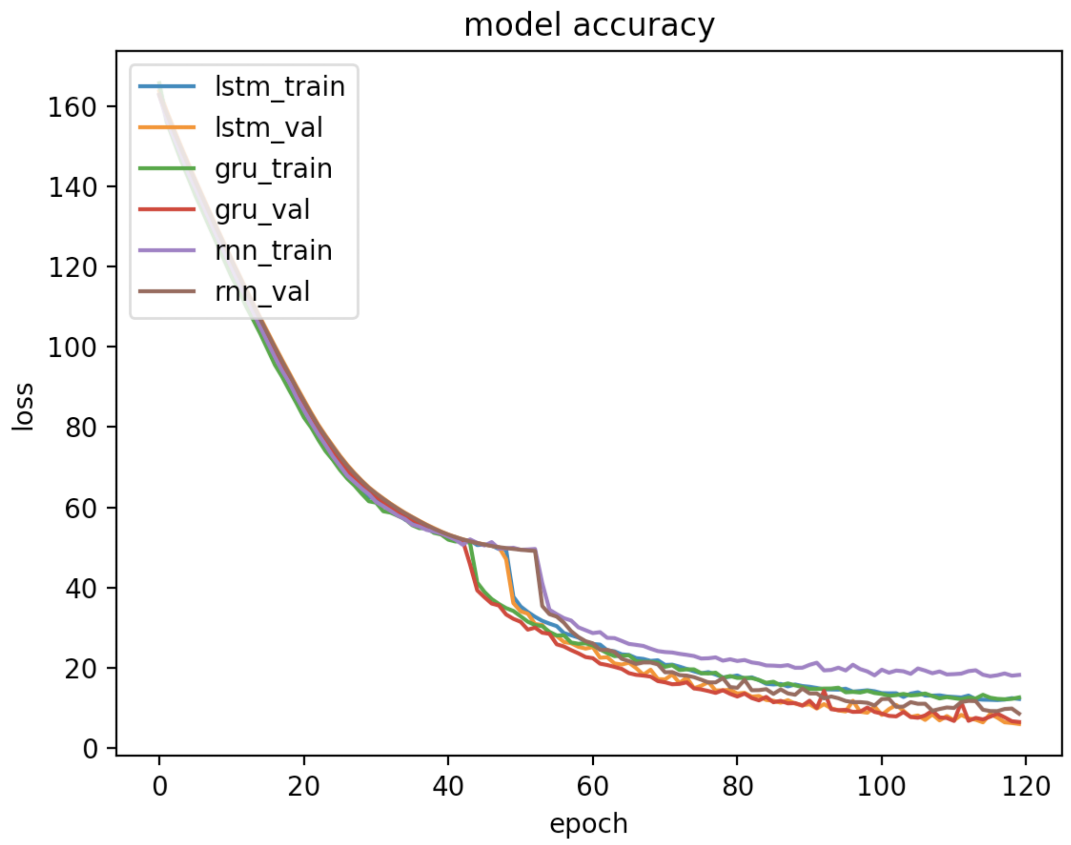
I considered three different implementations:

- SimpleRNN
- LSTM
- GRU

For the purpose of evaluation of the training model we have further divided the training data set into training and validation data by using the train_test_split functionality with the test_size as 0.2

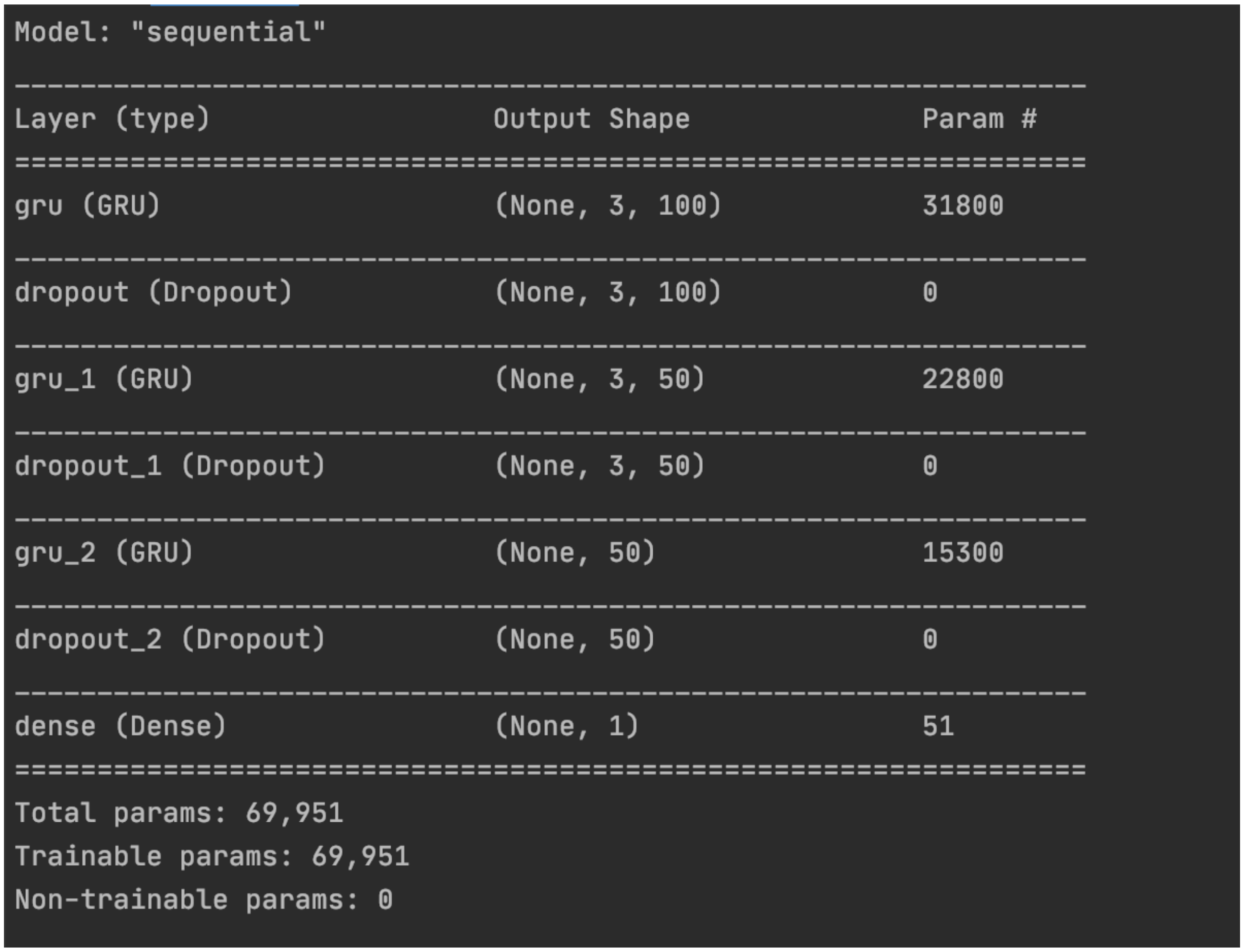
I have considered similar number of layers and similar number of neurons to each layer in all of the three above mentioned implementations. On exploratory trial and error we found that the optimal number of layers is 3 and the optimal number of neurons that can be considered in each layer can be around 50-120 after which the data overfitting occurs and the loss starts to increase drastically.

I have plotted the graph depicting the train and validation losses and mean square error or the root mean square error are some of the best metrics to be considered for these kind of regression problems to choose which model performs better.



All the three implementations are run for 120 epochs and they are run with a batch size of 8 and they fetch similar results with minor differences. From the models I have trained I have considered "GRU" to be the best model as it provides a lower loss at a very low number of epochs and performs well on predicting the test data.

ARCHITECTURE OF THE FINAL NETWORK



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 3, 100)            31800
_________________________________________________________________
dropout (Dropout)            (None, 3, 100)            0
_________________________________________________________________
gru_1 (GRU)                  (None, 3, 50)             22800
_________________________________________________________________
dropout_1 (Dropout)          (None, 3, 50)             0
_________________________________________________________________
gru_2 (GRU)                  (None, 50)                15300
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 69,951
Trainable params: 69,951
Non-trainable params: 0
```

Batch size :8
Number of epochs:200
Loss Function:RootMeanSquare
Numer of layers: 3 GRU layers with 50-100 neurons

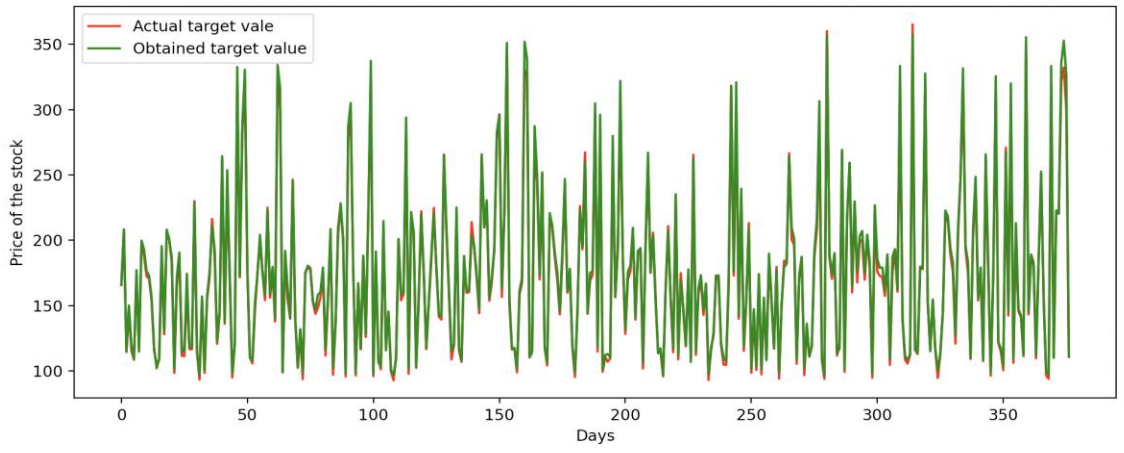
OUTPUT FROM TRAINING:
"refer to the epochs results printed as the output in the program"
On running the loop we are able to observe that on increasing the number of epoch the loss drastically reduces and reaches a stable state after 200 epochs and the optimal number of epochs would be 200 if further trained may result in overfitting of data.

GRU_MODEL:

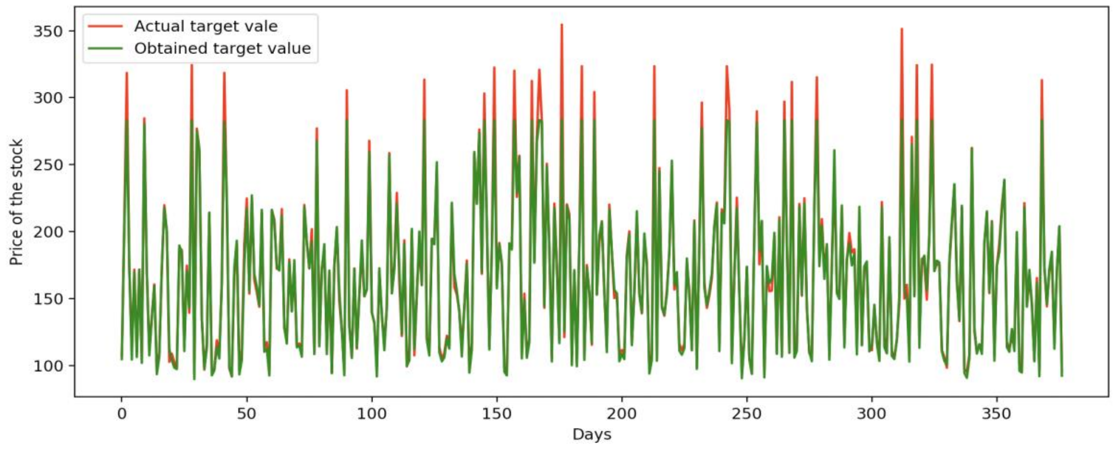
Training loss: 16.0230
Validation loss: 27.9894
Test loss: 21.21527

OUTPUT FROM TEST DATA:

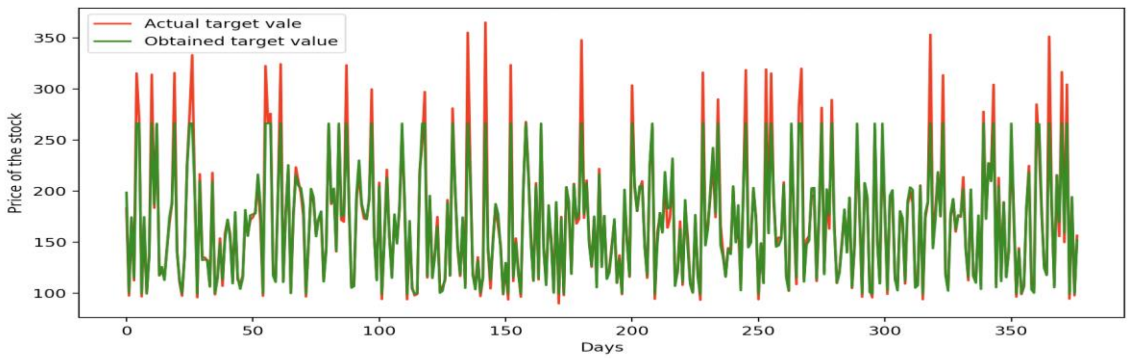
GRU: The prediction results obtained from the gru model seems to be more precise than the LSTM and simpleRNN. The graph plots obtained are as follows:



LSTM



SIMPLE_RNN (More test data error seen in this model)



If more data is used for training in regression problem the features considered becomes more redundant and meaningless resulting in more trainable parameters and causes overfitting resulting in higher value of loss than usual thereby resulting in a poor forecasting. So when we add more features we need to make sure that we omit the highly correlated features and consider only the necessary inputs to train the model.

QUESTION 3: SENTIMENTAL ANALYSIS

In [67]:
```
#importing all necessary libraries
import numpy as np
import tensorflow as tf
import keras as k
import matplotlib.pyplot as plt
import pandas as pd
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten , Dense, Activat
ion,Dropout,InputLayer,BatchNormalization,Embedding
import pickle
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
import os
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from tensorflow.python.keras import models
from tensorflow.python.keras.preprocessing import text
from tensorflow.keras import layers, models
from keras.preprocessing import sequence
import re
```

In [68]:
```
#creating the training dataset with positive and negative reviews
"""
The Large Movie Review Dataset contains 25000 highly polar reviews for
 training and 25000 dataset for testing.
So, we are attempting to do a binary classification on the reviews by c
lassifying them as positive or negative
reviews by using natural language processing.

In the downloaded datafile the ‘pos’ and the ‘neg’ files contain the po
sitive and the negative reviews respectively.
Considering these two files as input we need to create our data that wi
ll have the positive and negative reviews
with labels 1 and 0 for good and bad respectively.
"""
neg=[]
```

```
for root,directory,files in os.walk("/Users/chitraaramachandran/Downloa
ds/aclImdb/train/neg",topdown=True):
    for file in files:
        if file.endswith('.txt'):
            with open(os.path.join(root, file), 'r') as f:
                data = f.read()
                neg.append(data)
pos=[]
for root, directory, files in os.walk("/Users/chitraaramachandran/Downl
oads/aclImdb/train/pos",topdown=True):
    for file in files:
        if file.endswith('.txt'):
            with open(os.path.join(root, file), 'r') as f:
                data = f.read()
                pos.append(data)

#creating the test dataset with positive and negative reviews

neg_test=[]
for root, directory, files in os.walk("/Users/chitraaramachandran/Downl
oads/aclImdb/test/neg",topdown=True):
    for file in files:
        if file.endswith('.txt'):
            with open(os.path.join(root, file), 'r') as f:
                data = f.read()
                neg_test.append(data)
pos_test=[]
for root, directory, files in os.walk("/Users/chitraaramachandran/Downl
oads/aclImdb/test/pos",topdown=True):
    for file in files:
        if file.endswith('.txt'):
            with open(os.path.join(root, file), 'r') as f:
                data = f.read()
                pos_test.append(data)
```

In [69]:

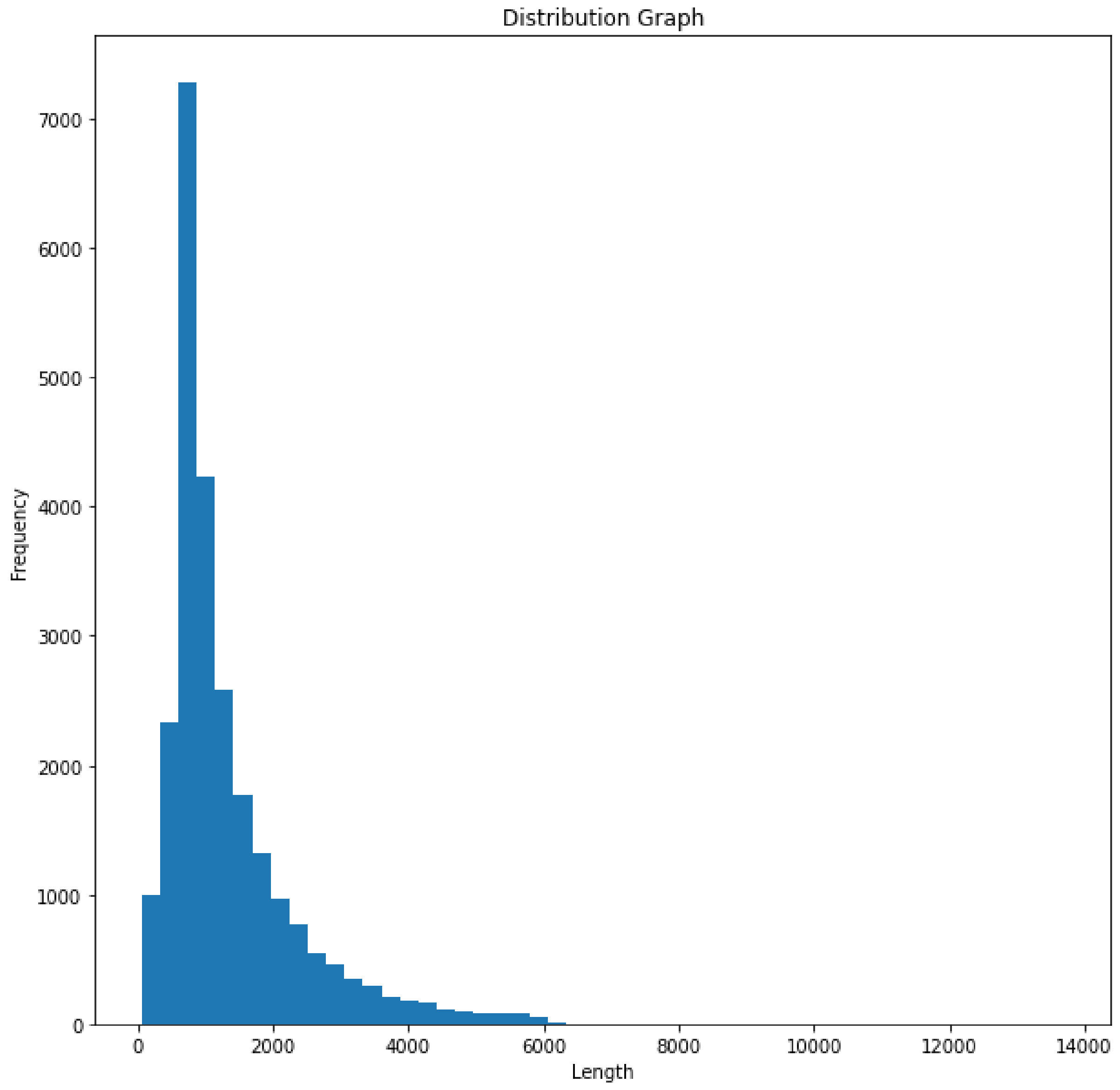
```
#creating the target polarity label with 1 for positive reviews and 0 f
or negative reviews
polarity = [0 if i < 12500 else 1 for i in range(25000)]
#creating the train and test set by appending together both the positiv
```

```
e and negative reviews
for i in range(12500):
    neg.append(pos[i])
    neg_test.append(pos_test[i])
X_train,y_train,X_test,y_test=([] for i in range(4))
for i in list(range(25000)):
    X_train.append(neg[i])
    y_train.append(polarity[i])
    X_test.append(neg_test[i])
    y_test.append(polarity[i])
```

In [70]:
```
tab=pd.DataFrame({"review":X_train,"polarity":y_train})
```

To proceed further we need to do some analysis to get an idea of the average review length, For this purpose i will be plotting a graph to show the review lengths

In [71]:
```
#plotting an histogram for length of the reviews
plt.figure(figsize=(10,10))
plt.hist([len(review) for review in list(X_train)], 50)
plt.xlabel('Length')
plt.ylabel('Frequency')
plt.title('Distribution Graph')
plt.show()
```



From the plot obtained, we can consider the average movie review length to be approximately around 2000 or lessser than that

```
In [72]: """
         The reviews contain a lot of unwanted data that do not hold any informa
         tion to train the model and
         they need to be removed by doing data preprocessing.The unwanted data o
         bserved in the reviews are numbers,
         punctuations,html tags and other special characters.These can be identi
         fied and removed using the regex matches
         and substitutions.
         """
         def remove(row):
             non_alpha = re.compile('[^a-zA-Z]')
             spcl_char = re.compile('[;.!:\',\"()?\[\]]')
             html = re.compile(r'<.*?>')
             row= [html.sub(" ",i) for i in row]
             row = [spcl_char.sub(" ", i) for i in row]
             row = [non_alpha.sub(" ", i.lower()) for i in row]
             return row
         #removing the unwanted data from the train and test
         X_train=remove(X_train)
         X_test=remove(X_train)
         #process of tokenization
         """
         Tokenization is the process of splitting a text into a list of tokens t
         hereby creating a word-to-index dictionary
         where each word is considered as a key and a unique index is created fo
         r it.
         """
         vocabulary = 5000
         tokenizer = Tokenizer(num_words=vocabulary)
         #we create the indexes based on word frequency so,we use fit_on_texts f
         untion from keras tokenizer
         tokenizer.fit_on_texts(X_train)
         #transforms each text in texts to a sequence of integer using texts_to_
         sequence to form a valid input to the neural network
         X_train=tokenizer.texts_to_sequences(X_train)
         X_test=tokenizer.texts_to_sequences(X_test)
         '''
         X_train and X_test are now a list of values and each list corresponds t
         o the sentence in the train and test set.
```

```
For purpose of training (To provide a proper input to the neural networ
k) we set a max length to the list in such a way that if the size of th
e list is greater than
the length mentioned then we truncate it and in the other scenario when
 the size is small we pad the list with 0
'''
max_length = 500
X_train = sequence.pad_sequences(X_train,maxlen=max_length)
X_test = sequence.pad_sequences(X_test,maxlen=max_length)
'''
We have assigned the vocabulary as 5000 and max length as 500 to train
 the model
'''
```

Out[72]:
```
'\nWe have assigned the vocabulary as 5000 and max length as 500 to tra
in the model\n'
```

Model Training

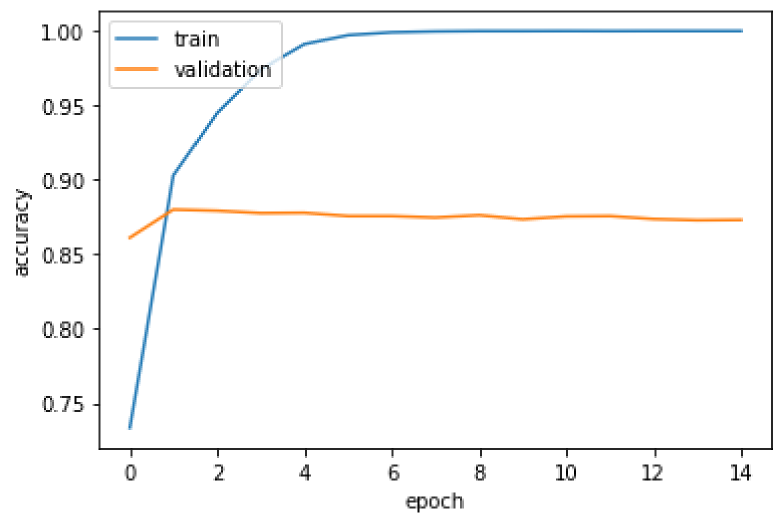
I have considered a sequential model and the vocabulary size to be 5000 and the maximum length of a sentence to be 500. We will be using the embedding layer that takes as input the reviews and forms embedding vectors which helps in dimensionality reduction and reduces the computational complexity and hence speeds up the training process unlike the large dimensionality produced through one hot encoding of target.We use the dropout layer with 0.2 after every embedding layer inorder to reduce the trainable parameters and help the model to train without overfitting.Flatten layer is used to feed the input to the dense layer. For the output we use a dense layer with softmax activation function and one neuron that outputs the polarity of the review. The model is compiled with the adam optimizer and the loss function considered is binary cross entropy as sentimental analysis probelm considered here is a binary classification probelm.

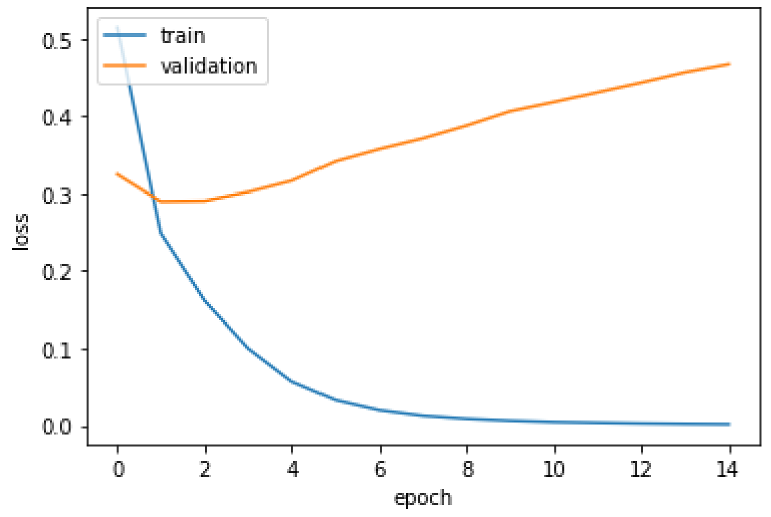
In [73]:
```
if __name__ == "__main__":
    #load the training dataset and create labels
    y_train = [0 if i < 12500 else 1 for i in range(25000)]
    y_train = np.asarray(y_train)
    # Splitting the training dataset into train and validation data to
```

```
 analyse the performance of the model
    x_train, x_val, y_train, y_val = train_test_split(X_train,y_train,
test_size=0.2,random_state=42)
    #Fixing the vocabulary and maximum length for the review
    vocabulary = 5000
    max_length = 500
    tf.keras.backend.clear_session()
    #defining the model
    imdb=models.Sequential()
    # adding the embedding layer which takes the input length and vocab
ulary size
    imdb.add(Embedding(vocabulary,64,input_length=max_length))
    imdb.add(Dropout(0.2))
    imdb.add(Flatten())
    imdb.add(Dense(1, activation='sigmoid'))
    #compiling the model using adam optimizer and binary_crossentropy
    op = tf.keras.optimizers.Adam()
    imdb.compile(optimizer=op,loss='binary_crossentropy',metrics=['accu
racy'])
    imdb.summary()
    #store the trained model
    file_name="/Users/chitraaramachandran/Downloads/20820331_NLP_model.
h5"
    checkpoint = k.callbacks.ModelCheckpoint(file_name ,
                             monitor = 'val_loss',
                             verbose = 0,
                             save_best_only = True,
                             mode = 'min')
    history = imdb.fit(x_train, y_train, validation_data=(x_val, y_val
), epochs=15, batch_size=64, verbose=2, callbacks = [checkpoint])
    #loading the trained weights
    imdb.load_weights(file_name)
    # plotting the training and validation accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train','validation'], loc='upper left')
    plt.show()
    # plotting the training and validation loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train','validation'], loc='upper left')
    plt.show()
    #printing the final obtained results
    loss,accuracy=imdb.evaluate(x_train,y_train)
    print("Accuracy on train: ",accuracy,"Loss on train: ",loss)
    loss,accuracy=imdb.evaluate(x_val,y_val)
    print("Accuracy on val: ",accuracy,"Loss on val: ",loss)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 500, 64)           320000
_________________________________________________________________
dropout (Dropout)            (None, 500, 64)           0
_________________________________________________________________
flatten (Flatten)            (None, 32000)             0
_________________________________________________________________
dense (Dense)                (None, 1)                 32001
=================================================================
Total params: 352,001
Trainable params: 352,001
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
313/313 - 3s - loss: 0.5141 - accuracy: 0.7333 - val_loss: 0.3247 - v
al_accuracy: 0.8612
Epoch 2/15
313/313 - 3s - loss: 0.2487 - accuracy: 0.9029 - val_loss: 0.2894 - v
al_accuracy: 0.8802
Epoch 3/15
313/313 - 3s - loss: 0.1617 - accuracy: 0.9451 - val_loss: 0.2901 - v
al_accuracy: 0.8794
Epoch 4/15
313/313 - 3s - loss: 0.0999 - accuracy: 0.9742 - val_loss: 0.3018 - v
```

```
al_accuracy: 0.8778
Epoch 5/15
313/313 - 3s - loss: 0.0570 - accuracy: 0.9911 - val_loss: 0.3164 - v
al_accuracy: 0.8780
Epoch 6/15
313/313 - 3s - loss: 0.0331 - accuracy: 0.9972 - val_loss: 0.3414 - v
al_accuracy: 0.8758
Epoch 7/15
313/313 - 3s - loss: 0.0199 - accuracy: 0.9991 - val_loss: 0.3572 - v
al_accuracy: 0.8758
Epoch 8/15
313/313 - 3s - loss: 0.0126 - accuracy: 0.9997 - val_loss: 0.3710 - v
al_accuracy: 0.8748
Epoch 9/15
313/313 - 3s - loss: 0.0086 - accuracy: 0.9999 - val_loss: 0.3875 - v
al_accuracy: 0.8764
Epoch 10/15
313/313 - 3s - loss: 0.0061 - accuracy: 0.9999 - val_loss: 0.4061 - v
al_accuracy: 0.8736
Epoch 11/15
313/313 - 3s - loss: 0.0044 - accuracy: 1.0000 - val_loss: 0.4180 - v
al_accuracy: 0.8756
Epoch 12/15
313/313 - 3s - loss: 0.0034 - accuracy: 0.9999 - val_loss: 0.4306 - v
al_accuracy: 0.8758
Epoch 13/15
313/313 - 3s - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.4431 - v
al_accuracy: 0.8738
Epoch 14/15
313/313 - 3s - loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.4564 - v
al_accuracy: 0.8730
Epoch 15/15
313/313 - 3s - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.4671 - v
al_accuracy: 0.8732
```





```
625/625 [==============================] - 1s 1ms/step - loss: 0.1656
- accuracy: 0.9498
Accuracy on train:  0.9497500061988831 Loss on train:  0.165553599596
02356
157/157 [==============================] - 0s 940us/step - loss: 0.28

94 - accuracy: 0.8802
Accuracy on val:  0.8802000284194946 Loss on val:  0.2893925905227661
Accuracy on val:  0.8802000284194946 Loss on val:  0.2893923905227661
```

In [75]:
```
#prediction on test data
vocabulary = 5000
max_length = 500
imdb = models.Sequential()
imdb.add(layers.Embedding(vocabulary, 64, input_length = max_length))
imdb.add(layers.Dropout(0.4))
imdb.add(layers.Flatten())
imdb.add(layers.Dense(1, activation='sigmoid'))
op = tf.keras.optimizers.Adam()
imdb.compile(optimizer=op,loss='binary_crossentropy',metrics=['accurac
y'])
if __name__ == "__main__":
    #creating the test labels
    y_test=[0 if i < 12500 else 1 for i in range(25000)]
    y_test=np.asarray(y_test)
    #loading the trained weights
    imdb.load_weights("/Users/chitraaramachandran/Downloads/20820331_NL
P_model.h5")
    loss,accuracy=imdb.evaluate(X_test,y_test)
    #printing the final accuracy obtained on test data
    print("Accuracy on test data : ",accuracy)
```

```
782/782 [==============================] - 1s 914us/step - loss: 0.1903
- accuracy: 0.9358
Accuracy on test data :  0.9358400106430054
```

REPORT

From the obtained result we can say that the optimal number of epochs will be 10 after which the training accuracy has reached 100 and there are no more trainable paramters and the validation accuracy also saturates.so is the loss function.The accuracy obtained on test accuracy is 93.5.This reported value might change because of the randomization.