CSE 574
Assignment -1

Part-1 (Implement Logistic Regression)

Assignment Overview

The main aim of this assignment(part-1) is to implement logistic regression without using sklearn library. Logistic Regression- Logistic regression is a supervised classification algorithm. In a classification problem, the target variable can take only discrete values for the inputs. In the first part of the assignment, I have implemented logistic regression using Gradient Descent .

Dataset

The dataset- Pima Indians Diabetes has 768 instances with 8 features. I have split data as training, validation and testing data constituting 60%, 20% and 20% of overall data. Training dataset has 460 samples, the validation dataset has 153 samples whereas the testing dataset has 153 samples.

The dataset contains 8 features like - Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin level, BMI, Diabetes, Pedigree Function, Age. The features in the dataset have no null values.

Reading the dataset-

```
#loading the Dataset
diabetes_df=pd.read_csv(r"diabetes.csv")
diabetes_df
```

We are importing the diabetes data set by using pd.read_csv() function, which takes values into a data frame.

Environment

I have used Google Colab for implementation of logistic regression.

Data Pre-Processing

It is very important to preprocess our data before we give it to our model. Preprocessing the data will eliminate Null Values and also normalizes the data.

```
1)diabetes_df.describe()
```

describe() is used for some statistical details like percentile, mean, std etc. of a data frame .

```
2)diabetes_df.isnull()
```

This function is used to check whether there are any null values present in dataset

3)To plot a correlation matrix we use heatmap which represents data in graphical format

```
plt.figure(figsize=(8, 8))
sns.heatmap(data=correlation_matrix, annot=True)
plt.show()
```

4)For converting into a numpy matrix

```
np.asmatrix(diabetes_df[['Pregnancies','Glucose','BloodPressure','Ski
nThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']])
Target_value = np.asmatrix(diabetes_df['Outcome']).T
#Scaling the features
parameters=scale(parameters)
parameters
```

5)Splitting the data into train, validation and testing

```
train_x,test_x,train_y, test_y = train_test_split(parameters,Target_value,
test_size=0.2,random_state=1)
train_x, x_val, train_y, y_val = train_test_split(train_x, train_y,
test_size=0.25)
train_x.shape
```

6)We need to initialize the weights to zeros at the start and we will take transpose of it for the matrix multiplication

```
weights = np.asmatrix([0,0,0,0,0,0,0,0]).T
```

7)Hyper-parameters

In machine learning there are two kinds of parameters namely model and hyper-parameters, hyperparameters cannot be directly learned from the training process.

Learning rate= alpha = 0.01

no of iterations=epochs= 500

Calculation of accuracy

```
def calculate_Accuracy(predict,test_y):
    no_of_cases=0
    for i in range(len(predictions)):
        if(predict[i]<0.5 and test_y[i]==0):
            no_of_cases=no_of_cases+1
        if(predict[i]>=0.5 and test_y[i]==1):
            no_of_cases=no_of_cases+1
    Accuracy=(no_of_cases/len(test_y))*100
    return Accuracy
```

Using gradient descent and cost

Cost function-

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

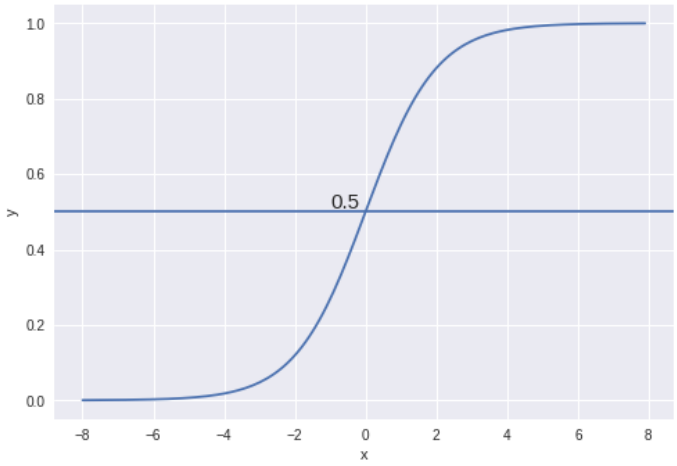
Gradient descent-

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^i \log(h_\theta(x^i)) + (1 - y^i)\log(1 - h_\theta(x^i))$$

Hypothesis function-

$$h\theta(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

Sigmoid function-

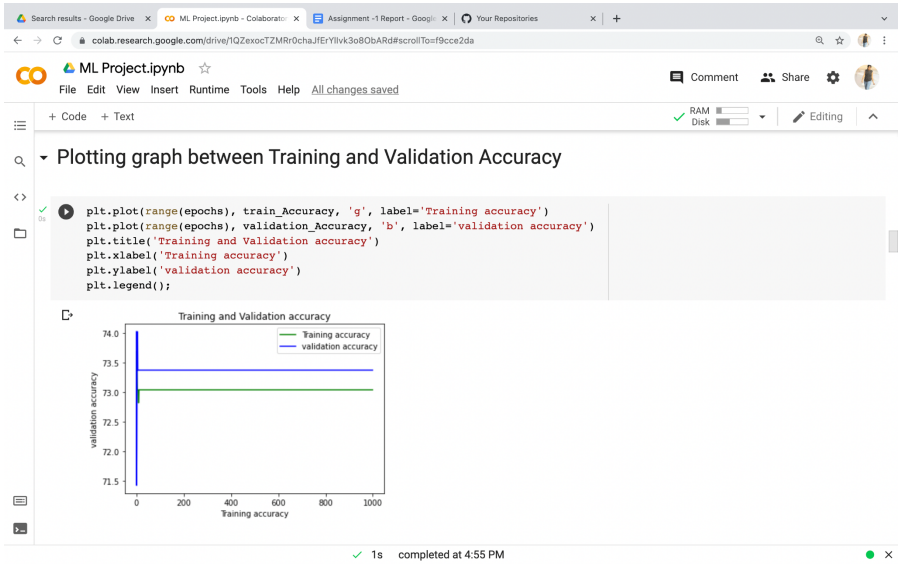



```
#Calculating gradient descent and cost functions
total_cost=[]
total_cost_val=[]
train_Accuracy=[]
validation_Accuracy=[]
for epoch in range(epochs):
    y_predict= 1 / (1 + np.exp(-np.dot(train_x,weights)))
    y_predict_val= 1 / (1 + np.exp(-np.dot(x_val,weights)))

gradient=np.dot(train_x.transpose(), (y_predict-train_y) )
cost_function=-np.sum(np.multiply(np.log(y_predict),train_y)+
np.multiply((1 - train_y), np.log(1 - y_predict)))/m
cost_function_val=-np.sum(np.multiply(np.log(y_predict_val),y_val)+
np.multiply((1 - y_val), np.log(1 - y_predict_val)))/m
weights=weights - alpha * gradient


#Calculate train and validation predictions
train_predictions= 1 / (1 + np.exp(-np.dot(train_x,weights)))
train_Accuracy.append(np.squeeze(calculate_Accuracy(train_predictions,trai
n_y)))
validation_predictions=1 / (1 + np.exp(-np.dot(x_val,weights)))
```

```
validation_Accuracy.append(np.squeeze(calculate_Accuracy(validation_predictions,y_val)))
# Computing the total cost for training and validation data
total_cost.append(np.squeeze(cost_function))
total_cost_val.append(np.squeeze(cost_function_val))

#Train accuracy in every epoch
print(train_Accuracy)
print(train_Accuracy[0])

#Validation accuracy in every epoch
print(validation_Accuracy)
```

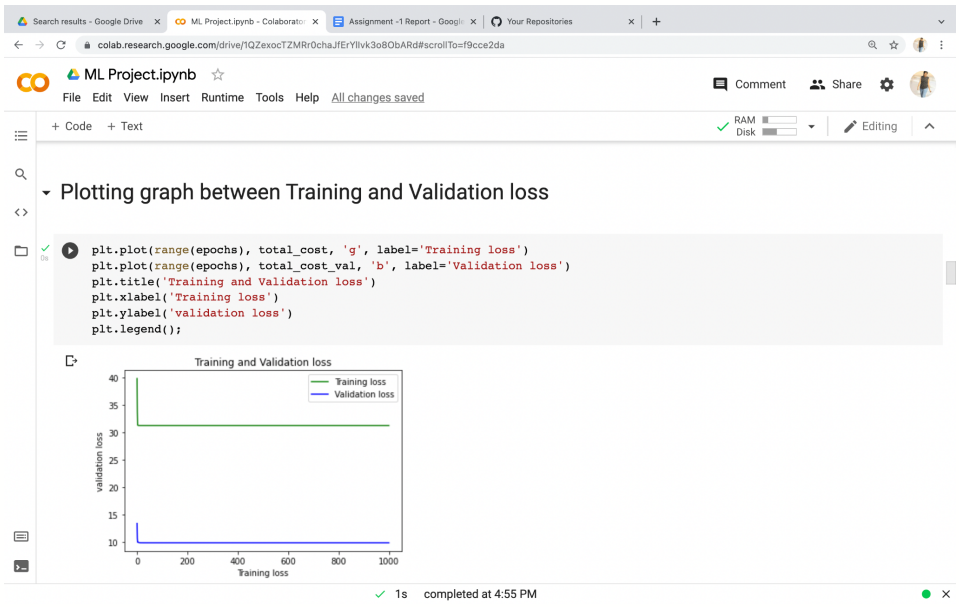
Plotting graph between Training and Validation Accuracy

```
plt.plot(range(epochs),train_Accuracy,'g',label='Training accuracy')

plt.plot(range(epochs), validation_Accuracy, 'b', label='validation accuracy')

plt.title('Training and Validation accuracy')

plt.xlabel('Training accuracy')
plt.ylabel('validation accuracy')
plt.legend();
```



Plotting graph between Training and Validation loss

```
plt.plot(range(epochs), total_cost, 'g', label='Training loss')
plt.plot(range(epochs), total_cost_val, 'b', label='Validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Training loss')
plt.ylabel('validation loss')
plt.legend();
```



OUTPUT RESULTS-

```
#Calculating train accuracy
train_predictions= 1 / (1 + np.exp(-np.dot(train_x,weights)))
train_accuracy=calculate_Accuracy(train_predictions,train_y)
print(train_accuracy)
```

Train accuracy as-73.04347826086956

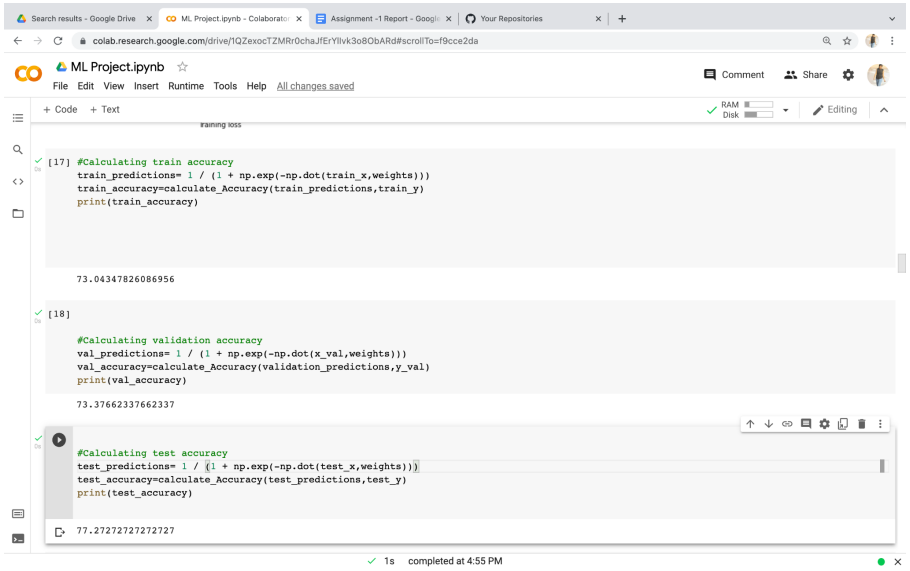
```
#Calculating validation accuracy
val_predictions= 1 / (1 + np.exp(-np.dot(x_val,weights)))
val_accuracy=calculate_Accuracy(validation_predictions,y_val)
print(val_accuracy)
```

Validation accuracy as-73.37662337662337

```
#Calculating test accuracy
test_predictions= 1 / (1 + np.exp(-np.dot(test_x,weights)))
test_accuracy=calculate_Accuracy(test_predictions,test_y)
print(test_accuracy)
```

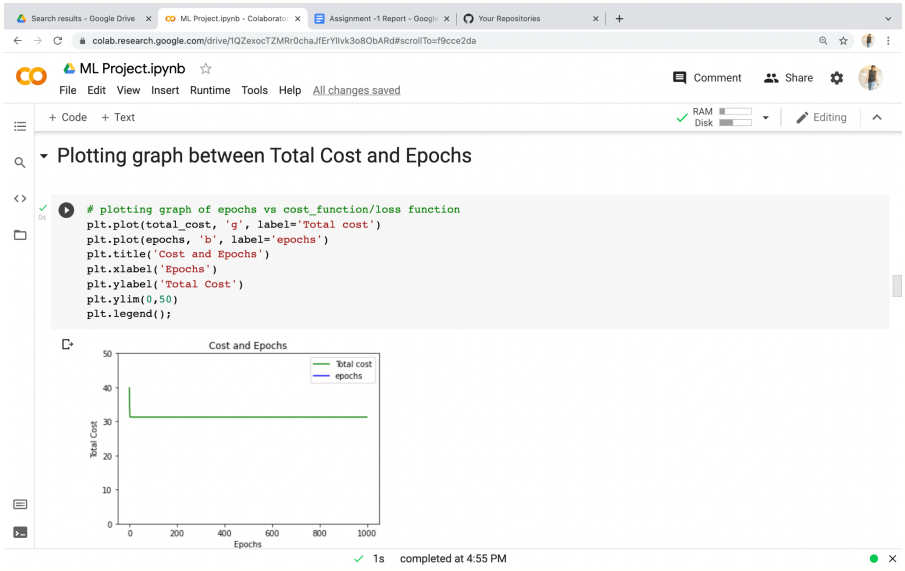
Test accuracy -77.27272727272727

OUTPUT- I got the test accuracy as 77.2



Plotting graph between Total Cost/Loss and Epochs

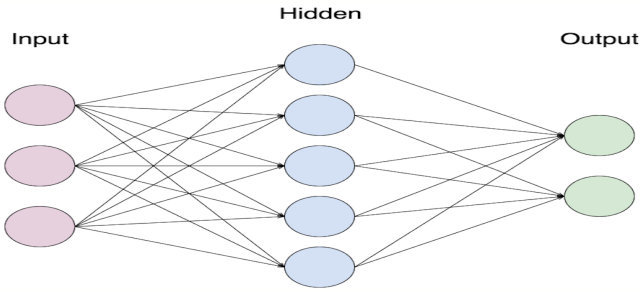
```
# plotting graph of epochs vs cost_function/loss function

plt.plot(total_cost, 'g', label='Total cost')
plt.plot(epochs, 'b', label='epochs')
plt.title('Cost and Epochs')
plt.xlabel('Total Cost')
plt.ylabel('Epochs')
plt.legend();
```


Search results - Google Drive
ML Project.ipynb - Colaborato
Assignment -1 Report - Google
Your Repositories
colab.research.google.com/drive/1QZexocTZMRr0chaJfErYIlvk3o8ObARd#scrollTo=f9cce2da
ML Project.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Editing
Plotting graph between Total Cost and Epochs
# plotting graph of epochs vs cost_function/loss function
plt.plot(total_cost, 'g', label='Total cost')
plt.plot(epochs, 'b', label='epochs')
plt.title('Cost and Epochs')
plt.xlabel('Epochs')
plt.ylabel('Total Cost')
plt.ylim(0,50)
plt.legend();
Cost and Epochs
Total cost
epochs
Total Cost
Epochs
0
10
20
30
40
50
0
200
400
600
800
1000
1s completed at 4:55 PM

PART-2 NEURAL NETWORKS

Neural Network-

Neural network is a class of machine learning algorithms used to model complex patterns in datasets using multiple hidden layers.



Implementing the neural network

1)The activation function used here is **"relu"** and loss used here is **"binary_crossentropy"** and the optimizer used here is **"adam".** Here I am taking epochs = 450 and batch size = 8

2)**One Hot Encoding-** Label encoding where we will assign a numerical value to the labels as 0 and 1. This encoding will add bias to the model so we will use One Hot Encoding technique.Where machine can easily understand the values.

```
#Loading the dataset
df = pd.read_csv("diabetes.csv")
data = df.to_numpy(dtype=np.float64)
X = data[:,:8]
Y = data[:,-1]
1)Now we are splitting the data with test size-20%, train data-60% and
validation data-20%
y_train = to_categorical(y_train, num_classes=None, dtype='float64')
y_validation = to_categorical(y_validation,
num_classes=None,dtype='float64')
y_test = to_categorical(y_test, num_classes=None, dtype='float64')
```

2)**L2-Regularization** is a technique which is used to reduce the errors by fitting the function appropriately on the given training set and avoid overfitting. The main aim of this regularization is to avoid overfitting of the data.

```
prototype = Sequential()

prototype.add(Dense(500, input_dim=8,
activation='relu',activity_regularizer=regularizers.l2(1e-4)))

prototype.add(Dense(100,
activation='relu',activity_regularizer=regularizers.l2(1e-4)))

prototype.add(Dense(2, activation='softmax'))

prototype.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

epochs = 450

batch_size = 8

history = prototype.fit(x_train,y_train, epochs=epochs,
batch_size=batch_size, validation_data=(x_val, y_val))

print("Train acc-",history.history["accuracy"][-1])

print("Val acc-",history.history["val_accuracy"][-1])

print("Training loss-", history.history["loss"][-1])

test_results = prototype.evaluate(x=x_test, y=y_test)

test_results
```

3)**ReLu Activation function-** It is used to tell the output of a neural network like yes or no. It maps the resulting values in between 0 to 1. The ReLu activation function is used because it is simple, fast.

4)**SoftMax Activation function-**The softmax function is an activation function in the output layer. This function is used as an activation function in hidden layers of the neural network.

4)**Binary Cross Entropy-** I used loss function as binary cross entropy. It is commonly used loss function for classification

5)**SGD**-Keras provides the stochastic gradient descent that implements the stochastic gradient descent optimizer.

```
# Accuracy

plt.figure(figsize=(15, 10))

plt.plot(history.history['accuracy'])

plt.plot(history.history['val_accuracy'])

plt.title('2 Hidden Layer: Training and Validation Accuracy')

plt.ylabel('Training accuracy')

plt.xlabel('Validation accuracy')

plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='upper
right')

plt.show()

# Loss

plt.figure(figsize=(15, 10))

plt.plot(history.history['accuracy'])

plt.plot(history.history['val_loss'])

plt.title('2 Hidden Layer: Training and Validation Loss')

plt.ylabel('Training Loss')

plt.xlabel('Validation Loss')

plt.legend(['Training accuracy', 'Validation Loss'], loc='upper right')

plt.show()
```

OUTPUT RESULT

L2 REGULARIZATION-

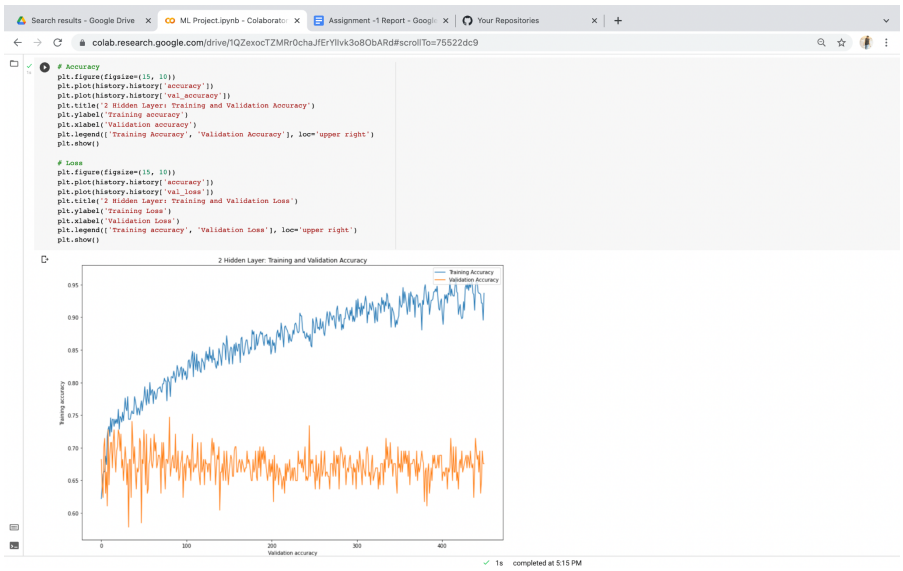
(graph of train accuracy and validation accuracy) and (graph of training loss and validation loss)

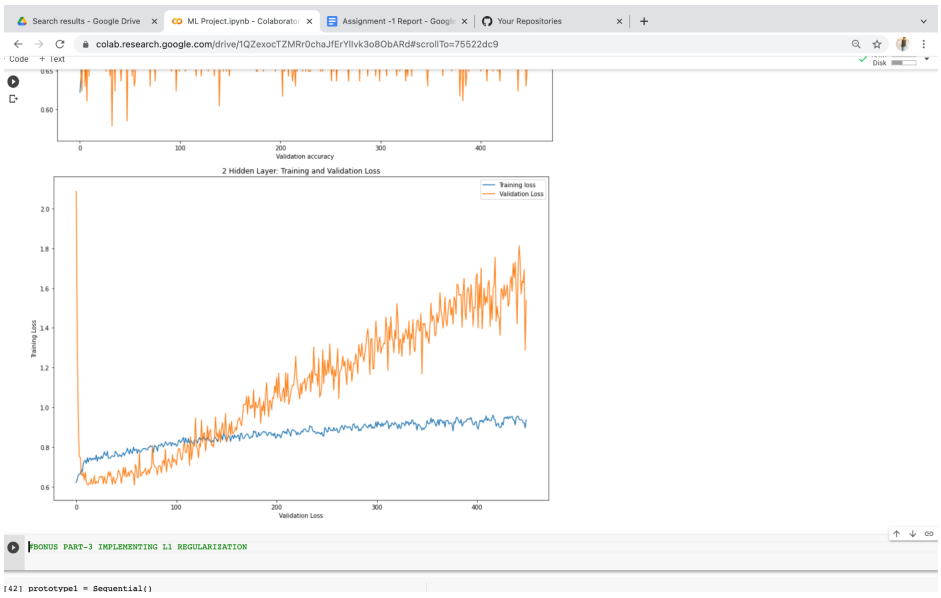
Train acc- 0.936956524848938

Val acc- 0.6753246784210205

Training loss- 0.1639959216117859

Test accuracy -75.97





BONUS PART-3

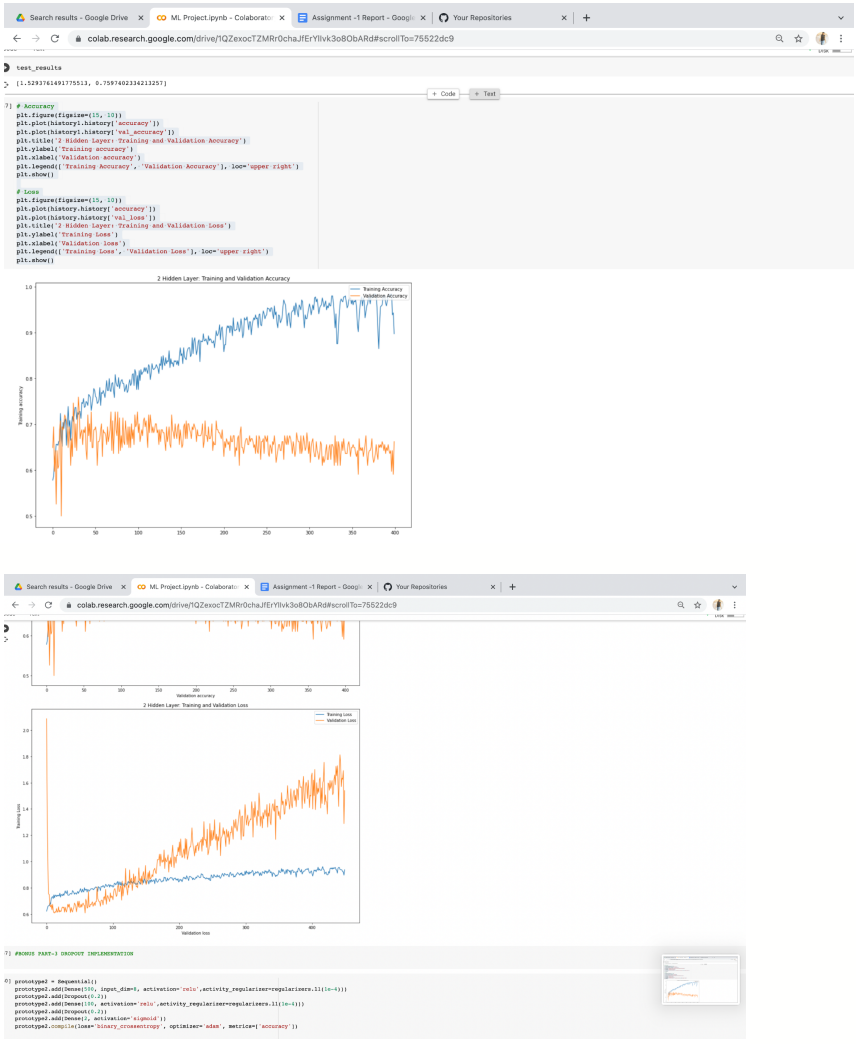
L1 regularization-(Graph of train accuracy and validation accuracy) and (graph of training loss and validation loss)

Train acc- 0.897826075553894

Val acc- 0.6623376607894897

Training loss- 0.24957261979579926

Test accuracy -75.97



BONUS PART-3

DROPOUT REGULARIZATION

train acc- 0.897826075553894

Val acc- 0.6623376607894897

Training loss- 0.24957261979579926

Test_results-75.97