Project 2

Neural Network (2-Hidden layers) Model for Prediction of Corona Cases

Submitted To: Dr. Hasan Sajid
Submitted By: Usman Zaheer
Registration Number: 00000327700
Subject: Machine Learning

Contents

1. Problem Statement: 4
2. Dataset Details: 4
i. Gathering and Cleaning: 4
   - USA Dataset Websites/Links 4
   - Worldwide Cases Dataset Websites/Links 4
ii. Size of Data: 4
   - USA Dataset Size: 4
   - Worldwide Dataset Size 5
iii. Features Details and Scaling: 5
   - USA Dataset Features: 5
   - Worldwide Dataset Features: 5
   - Scaling of features: 6
iv. Code and Methodology: 6
3. Mathematical Model Details: 7
   - Architecture of Neural Network Model: 7
   - Initialization of parameters: 7
   - Forward Propagation: 8
   - Cost Function: 9
   - Back Propagation: 9
   - Gradient Descent: 10
   - Complete Neural Network Model with integration of all above mentioned functions: 11
4. Output of the Model: 12
   - USA Dataset Model predictions after training: 12
   - Worldwide Dataset Model predictions after training: 14
5. Model Training Details: 15
   - USA Dataset Model training details: 15
   - Worldwide Dataset Model training details: 16
6. Plots: 16
   - USA Dataset Plots: 16
   - Worldwide Dataset Plots: 18
7. Complete Codes: 20

Annex A 20

- Instructions on running the code:..... 20

Annex B..... 21

- Training Code with Optimal Parameters: 21

Annex C..... 32

- Prediction Code:..... 32

1. Problem Statement:

Write a complete code to train a 2-layer(hidden) Neural Network for predicting corona cases in USA and world with regularization.

In this model, neural network model composed of 2 hidden layers is implemented with regularization for predicting corona cases in USA different states and in the different countries of world.

2. Dataset Details:

i. Gathering and Cleaning:

The following websites/links are used for getting data for USA data set and Worldwide. Data is extracted through manual methods and was cleaned in excel. After that further cleaning was done in Jupyter Notebook.

➢ USA Dataset Websites/Links

- **For corona cases data**
  https://covidtracking.com/data/api
- **For weather states temperature avg data**
  https://www.weatherbase.com/weather/state.php3?c=US&name=United+States+of+America
- **For humidity data of states**
  https://www.currentresults.com/Weather/US/annual-average-humidity-by-state.php-main
  https://www.forbes.com/sites/brianbrettschneider/2018/08/23/oh-the-humidity-why-is-alaska-the-most-humid-state/?sh=72a82a2e330c
- **Population density**
  https://worldpopulationreview.com/state-rankings/state-densities
- **Simple population**
  https://www.worldometers.info/coronavirus/country/us/

➢ Worldwide Cases Dataset Websites/Links

- **For corona cases data:**
  https://github.com/owid/covid-19-data/blob/master/public/data/owid-covid-data.csv
- **For weather temperature data:**
  https://www.timeanddate.com/weather/
- **Population density:**
  https://github.com/owid/covid-19-data/blob/master/public/data/owid-covid-data.csv
- **Human Development Index:**
  https://github.com/owid/covid-19-data/blob/master/public/data/owid-covid-data.csv

ii. Size of Data:

➢ USA Dataset Size:

The USA data set csv is composed of **14840** Rows and **10** Columns. It is further cleaned and splited into train, test and validation sets.

➢ Worldwide Dataset Size

The worldwide data set csv is composed of **54720** Rows and **10** Columns. It is further cleaned for useful features and then splited into train, test and validation sets.

iii. Features Details and Scaling:

➢ USA Dataset Features:

In USA data set there are **7** valid features:

- 1)DATE_CODE = In this model "date" is numerically coded. For example:
  Model date starts from 6th December 2020 and ends at 17th March 2020.
  *Here: 17th March 2020 is 264 and 6th December is 0.
- 2) States_Code: States codes are also numerically coded. For example:
  *Here: 1 = Alaska, 2= Alabama and 56=Wyoming.
- 3) Temperatures_(F): Temperatures in F of state at date.
- 4) Humidity (%): Average humidity in % of state at date.
- 5) Population_per_state: Average population in of state at date.
- 6) LandArea (sq miles): Land Area in square miles of particular state .
- 7) POPULATION_DENSITY: Population Density of state.
  Here, We have 1 label : Cases_per_day in particular state at particular date.

	Date_Code	States_Code	Temperatures_(F)	Humidity_(%)	Population_per_state	LandArea_(sq miles)	Population_Density
0	0	1	11.6	77.1	731545	570641	1.29
1	0	2	46.6	71.6	4903185	50645	96.92
2	0	3	41.3	70.9	3017804	52035	58.40
3	0	4	38.0	80.0	55138	77	716.00
4	0	5	43.6	38.5	7278717	113594	64.95

Figure 1: Features in USA Data Set

In this data, there is 1 label : Cases_per_day in particular state at particular date.

➢ Worldwide Dataset Features:

In worldwide data set there are **5** valid features:

- 1) DATE_CODE = In this model "date" is numerically coded. For example:
  Model date starts from 23 January 2020 and ends at 7th December 2020.
  *Here: 23/01/2020 is 0 and 7th December is 319.
- 2) COUNTRY_CODE: Country codes are also numerically coded. For example:
  *Here: 1 = Afghanistan, 2= Albania and 170=Zimbabwe.
- 3) HUMAN_DEVELOPMENT_INDEX: Average Human development index of state
- 4) POPULATION_DENSITY: Population Density of country.
- 5) TEMPERATURES: Temperatures in C of country at date.

	DATE_CODE	COUNTRY_CODE	POPULATION DENSITY	HUMAN_DEVELOPMENT_INDEX	TEMPERATURES
0	0	0	54.42	0.498	7.0
1	1	0	54.42	0.498	7.0
2	2	0	54.42	0.498	13.0
3	3	0	54.42	0.498	2.0
4	4	0	54.42	0.498	6.0

Figure 2: Worldwide data set features

Here, We have 1 label: CASES in particular country at particular date

➢ Scaling of features:

These features are scaled by using the following formula in both data sets

Here, x are the features.

for j in range(0,len(x.columns)):

x= (x-x.min())/(x.max()-x.min())

	Date_Code	States_Code	Temperatures_(F)	Humidity_(%)	Population_per_state	LandArea_(sq miles)	Population_Density
0	0.0	0.000000	0.140097	0.930456	0.017143	1.000000	0.000000
1	0.0	0.018182	0.562802	0.798561	0.122869	0.088642	0.008291
2	0.0	0.036364	0.498792	0.781775	0.075086	0.091079	0.004952
3	0.0	0.054545	0.458937	1.000000	0.000000	0.000016	0.061967
4	0.0	0.072727	0.526570	0.004796	0.183074	0.198968	0.005519

Figure 3: USA data set features after scaling

	DATE_CODE	COUNTRY_CODE	POPULATION DENSITY	HUMAN_DEVELOPMENT_INDEX	TEMPERATURES
0	0.000000	0.0	0.002711	0.240401	0.338235
1	0.003135	0.0	0.002711	0.240401	0.338235
2	0.006270	0.0	0.002711	0.240401	0.426471
3	0.009404	0.0	0.002711	0.240401	0.264706
4	0.012539	0.0	0.002711	0.240401	0.323529

Figure 4: Worldwide data set features after scaling

iv. Code and Methodology:

The methodology opted for this model is that in the following steps:

1) Data import using pandas.
2) Data Cleaning.
3) Assigning features to x and y.

4) Features scaling.
5) Data splits.
6) Defining the architecture of neural network model.
7) Initiating parameter values.
8) Writing functions for Forward Propagation, Cost Function, Back Propagation and Gradient Descent.
9) Training on training data split and updating parameters.
10) Error metrics and comparison.
11) Results Visualization

3. Mathematical Model Details:

➢ Architecture of Neural Network Model:

The neural network model used in this case is composed of **01** input layer, **02** hidden layers and **01** output layer.

For USA Data set architecture is:

-1) Input Layer (7 Neurons)
- 2) Hidden layer 1 (10 Neurons)
- 3) Hidden Layer 2 (10 Neurons)
- 4) Output Layer (1 Neuron)

For Worldwide Data set architecture is:

- 1) Input Layer (5 Neurons)
- 2) Hidden layer 1 (10 Neurons)
- 3) Hidden Layer 2 (10 Neurons)
- 4) Output Layer (1 Neuron)

Code for layer size definition is:

```
def layer_sizes(x, y):
    n_x = x.shape[0] # size of input layer
    n_h1 = 10
    n_h2 = 10
    n_y = y.shape[0] # size of output layer
    return (n_x, n_h1, n_h2, n_y)
```

➢ Initialization of parameters:

Neural Network is implemented in this model, so for forward propagation the model parameters were initialized.

In this model, there are **06** parameters:

W1 -- weight matrix of shape (n_h1, n_x) b1 -- bias vector of shape (n_h1, 1)
W2 -- weight matrix of shape (n_h2, n_h1) b2 -- bias vector of shape (n_h2, 1)
W3 -- weight matrix of shape (n_y, n_h2) b3 -- bias vector of shape (n_y, 1)

Parameters initial values are initiated using this function:

```
def initialize_parameters(n_x, n_h1,n_h2, n_y)
    np.random.seed(2)
    W1 = np.random.randn(n_h1,n_x)*0.01
    b1 = np.zeros((n_h1,1))
    W2 = np.random.randn(n_h2,n_h1)*0.01
    b2 = np.zeros((n_h2,1))
    W3 = np.random.randn(n_y,n_h2)*0.01
    b3 = np.zeros((n_y,1))
    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2,
                  "W3": W3,
                  "b3": b3}
    return parameters
```

➢ Forward Propagation:

The forward propagation is used to calculates predictions using combination of features, parameters and bias terms:

Forward propagation definition is given by the following function:

Here:
x -- input data of size (n_x, m) **m** is the no of training examples
parameters -- python dictionary containing your parameters (output of initialization function)
A3 -- The output **cache --** a dictionary containing "Z1", "A1", "Z2", "A2" , "Z3" and "A3"

Forward Propagation Code is:

```
def forward_propagation(x, parameters):

    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]
    W3 = parameters["W3"]
    b3 = parameters["b3"]

    Z1 = np.dot(W1,x) + b1
    A1 = sigmoid(Z1)              # Sigmoid Activation
    Z2 = np.dot(W2,A1) + b2
    A2 = sigmoid(Z2)              # Sigmoid Activation
    Z3 = np.dot(W3,A2) + b3       # Simple Linear
    A3 = Z3
```

```
    assert(A3.shape == (1, x.shape[1]))

    cache = {"Z1": Z1,
             "A1": A1,
             "Z2": Z2,
             "A2": A2,
             "Z3": Z3,
             "A3": A3
            }
    return A3, cache
```

➢ Cost Function:

Cost function is basically the difference between prediction by the model and the prediction label.

Cost function is given by:

```
def compute_cost(A3, y, parameters,lambda_):

    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]
    W3 = parameters["W3"]
    b3 = parameters["b3"]
    m = y.shape[1]
cost=(np.sum(np.power((A3-
y_train),2))+lambda_*(np.sum(np.power(W1,2))+np.sum(np.power(W2,2))+np.sum(n
p.power(W3,2))))/(2*m)
    cost = np.squeeze(cost)
    assert(isinstance(cost, float))
    return cost
```

here,

x_train = features in training set. **y_train =** prediction label in training set.

lambda_ = regularization parameter. **m** = the length of training set.

➢ Back Propagation:

Backpropagation is recursive application of the chain rule along a computational graph to compute the gradients of all parameters / intermediates.

Back Propagation is given by following code:

```
def backward_propagation(parameters, cache, x, y,lambda_):

    m = x.shape[1]
```

```
W1 = parameters["W1"]
W2 = parameters["W2"]
W3 = parameters["W3"]

A1 = cache["A1"]
A2 = cache["A2"]
A3 = cache["A3"]

dZ3 = A3-y
dW3 = 1/m*(np.dot(dZ3,A2.T)) + (lambda_/m)*W3
db3 = 1/m*(np.sum(dZ3,axis=1, keepdims=True))
dZ2 = np.multiply(np.dot(W3.T,dZ3),(A2*(1-A2)))
dW2 = 1/m*(np.dot(dZ2,A1.T)) + (lambda_/m)*W2
db2 = 1/m*(np.sum(dZ2,axis=1, keepdims=True))
dZ1 = np.multiply(np.dot(W2.T,dZ2),(A1*(1-A1)))
dW1 = 1/m*(np.dot(dZ1,x.T)) + (lambda_/m)*W1
db1 = 1/m*(np.sum(dZ1,axis=1, keepdims=True))

grads = {"dW1": dW1,
      "db1": db1,
      "dW2": dW2,
      "db2": db2,
      "dW3": dW3,
      "db3": db3}
return grads
```

Here,

parameters -- python dictionary containing our parameters
cache -- a dictionary containing "Z1", "A1", "Z2" "A2", "Z3" and "A3"
x -- input data of shape (7/5, number of examples)
y -- "true" labels vector of shape (1, number of examples)
lambda_ -- Regularization parameter
grads -- python dictionary containing your gradients with respect to parameters of model.

➢ Gradient Descent:

Gradient Descent is used to find the minimum values of parameters, so that our cost will be minimum. Minimum cost indicates that the difference between our prediction and actual label is very low. Regularization is also applied with Gradient Descent to prevent overfitting.

Gradient Descent is given by:

```
def update_parameters(parameters, grads, alpha = 0.001):

   W1 = parameters["W1"]
   b1 = parameters["b1"]
   W2 = parameters["W2"]
```

```
    b2 = parameters["b2"]
    W3 = parameters["W3"]
    b3 = parameters["b3"]

    dW1 = grads["dW1"]
    db1 = grads["db1"]
    dW2 = grads["dW2"]
    db2 = grads["db2"]
    dW3 = grads["dW3"]
    db3 = grads["db3"]

    W1 = W1 - alpha*dW1
    b1 = b1 - alpha*db1
    W2 = W2 - alpha*dW2
    b2 = b2 - alpha*db2
    W3 = W3 - alpha*dW3
    b3 = b3 - alpha*db3

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2,
                  "W3": W3,
                  "b3": b3}
    return parameters
```

Here,

alpha = learning rate **Parameters** = Model Parameter after gradient descent.

- ## Complete Neural Network Model with integration of all above mentioned functions:

All the functions are integrated in the neural network model function. The code for following functions is given below:

```
def nn_model(x, y, n_h1,n_h2,num_iterations=1000 ,print_cost=False, lambda_= 0.7):
    np.random.seed(3)
    n_x = layer_sizes(x, y)[0]
    n_y = layer_sizes(x, y)[3]
    cost_history_train=[]
    cost_history_valid=[]
    cost_history_test= []
    parameters = initialize_parameters(n_x,n_h1,n_h2, n_y)
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
```

```
        b2 = parameters["b2"]
        W3 = parameters["W3"]
        b3 = parameters["b3"]

    for i in range(0, num_iterations):
            A3, cache = forward_propagation(x_train, parameters)
            A3_valid,cache1 = forward_propagation(x_valid, parameters)
            A3_test, cache2 = forward_propagation(x_test, parameters)
            cost_train = compute_cost(A3,y_train,parameters,lambda_)
            cost_history_train.append(cost_train)
            cost_valid = compute_cost(A3_valid,y_valid,parameters,lambda_)
            cost_history_valid.append(cost_valid)
            cost_test = compute_cost(A3_test,y_test,parameters,lambda_)
            cost_history_test.append(cost_test)
            grads = backward_propagation(parameters,cache,x,y,lambda_)
            parameters = update_parameters(parameters,grads)
            if print_cost and i % 1== 0:
                print ("Cost after iteration %i: %f" %(i, cost_train))
        return parameters, cost_history_train,cost_history_valid,cost_history_test
```

Here,

alpha is learning rate **=0.001**
iterations**=1000**
lambda_ is regularization parameter**= 0.7**

4. Output of the Model:

➢ USA Dataset Model predictions after training:

The model parameters of USA data set are given below:

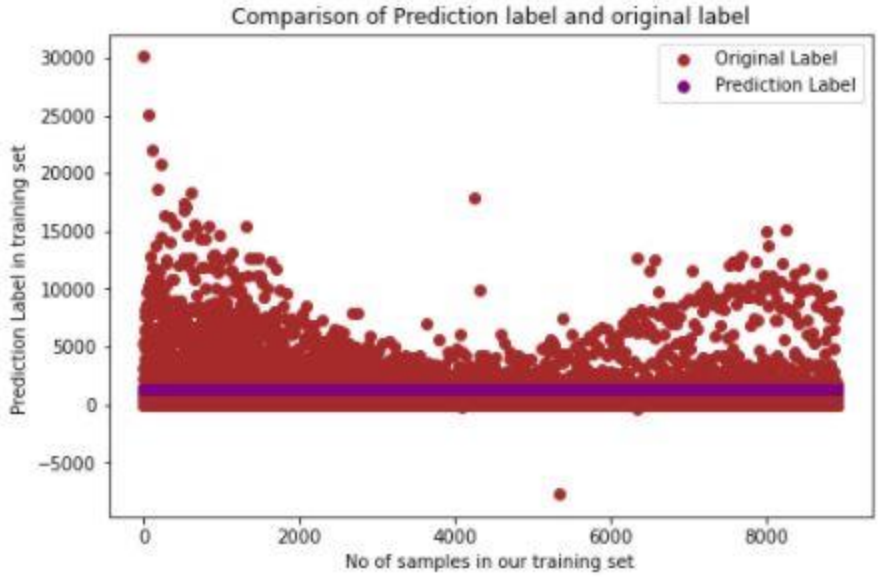
As, in USA data set we have 7 features, therefore we got different parameters after optimization.

The prediction function for predict is given by:

```
def predict(parameters, x):
    A3, cache = forward_propagation(x,parameters)
    predictions = A3
return predictions
```

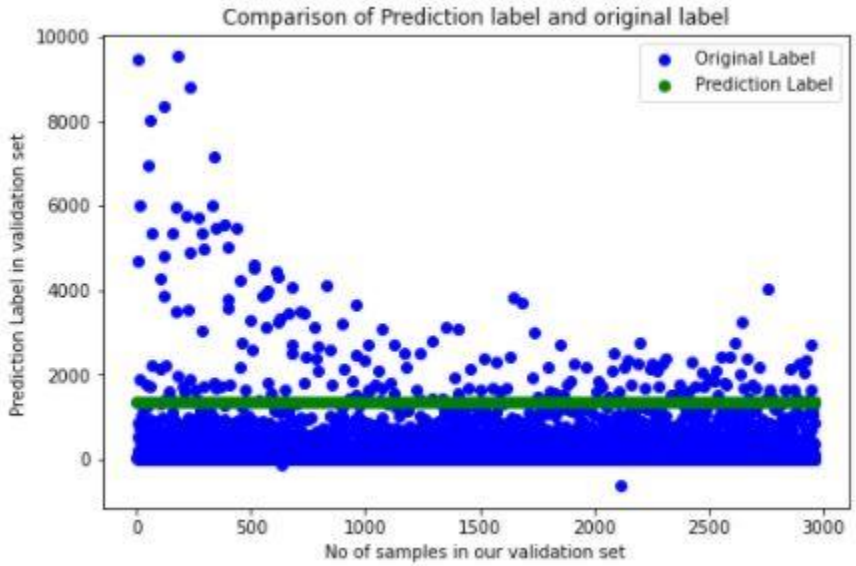
- **`predictions_train`** is our prediction and we are applying our model parameters on training set to get the predictions.

```
predictions_train = predict(parameters, x_train)
```



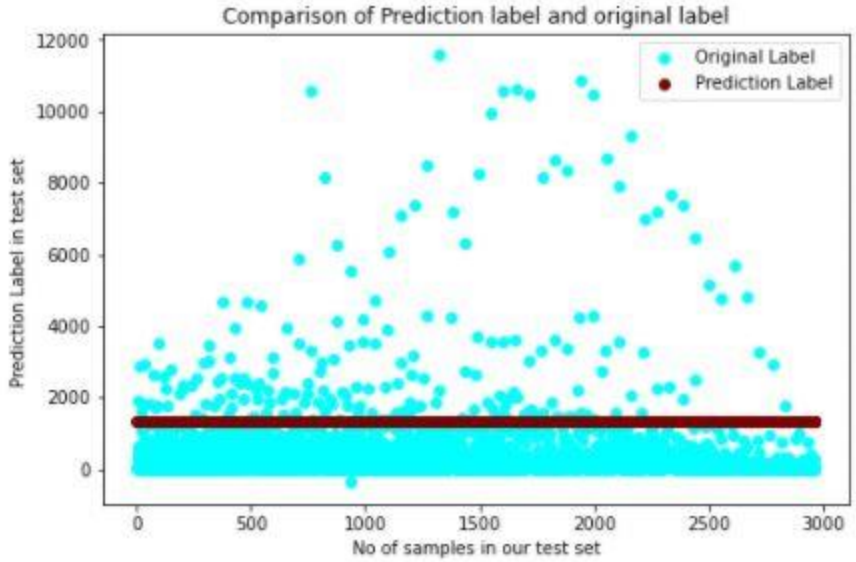
- ` **predictions_valid** ` is our prediction on validation set and we are applying our model parameters on validation set to check the predictions and accuracy of our model.

predictions_valid = predict(parameters, x_valid)



- ` **predictions_test**` is our prediction on test set and we are applying our model parameters on test set to check the predictions and accuracy of our model.

predictions_test = predict(parameters, x_test)



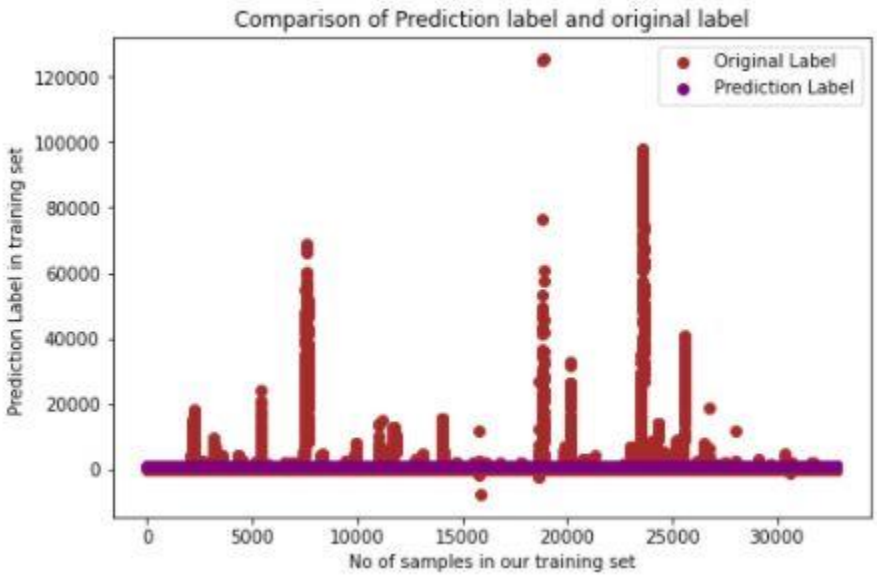
➢ Worldwide Dataset Model predictions after training:

The model parameters and predictions of Worldwide data set are given below:

As, in Worldwide data set we have 5 features, therefore we got multiple parameters after optimization.

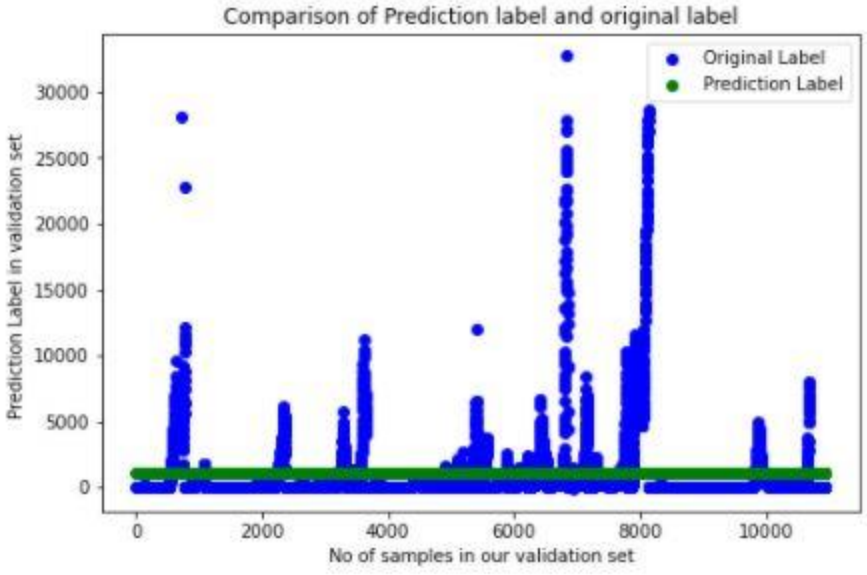
- ` **predictions_train** ` is our prediction and we are applying our model parameters on training set to get the predictions.

predictions_train = predict(parameters, x_train)



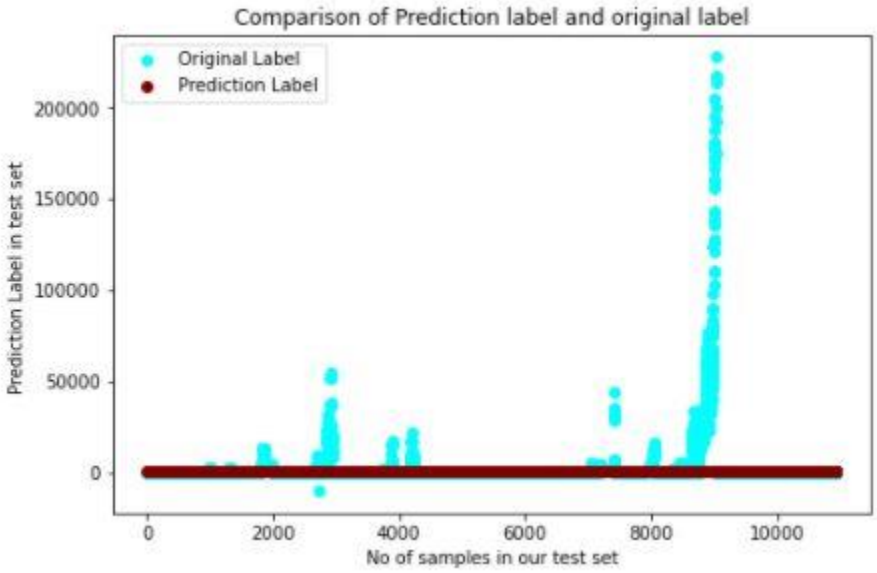
- ` **predictions_valid** ` is our prediction on validation set and we are applying our model parameters on validation set to check the predictions and accuracy of our model.

predictions_valid = predict(parameters, x_train)



- ` **predictions_test** ` is our prediction on test set and we are applying our model parameters on test set to check the predictions and accuracy of our model.

predictions_test = predict(parameters, x_test)



5. Model Training Details:

➢ USA Dataset Model training details:

For USA Data set training data is composed of **8904** columns and **7** rows.

- alpha is learning rate = **0.001**
- Iterations is our loop running factor = **1000**
- lambda_ is regularization parameter **= 0.7**

➢ Worldwide Dataset Model training details:

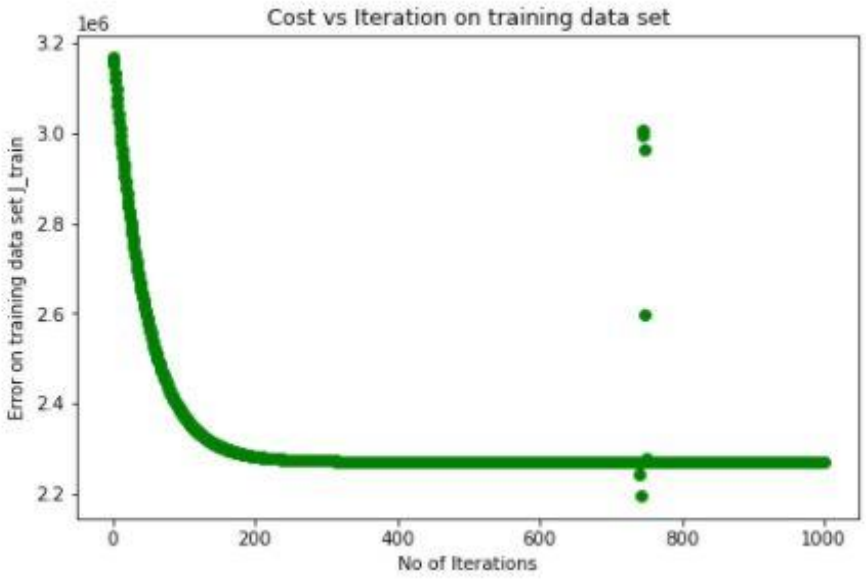
For Worldwide Data set training data is composed of **32832** columns and **5** rows.

- alpha is learning rate = **0.001**
- Iterations is our loop running factor = **1000**
- lambda_ is regularization parameter **= 0.7**

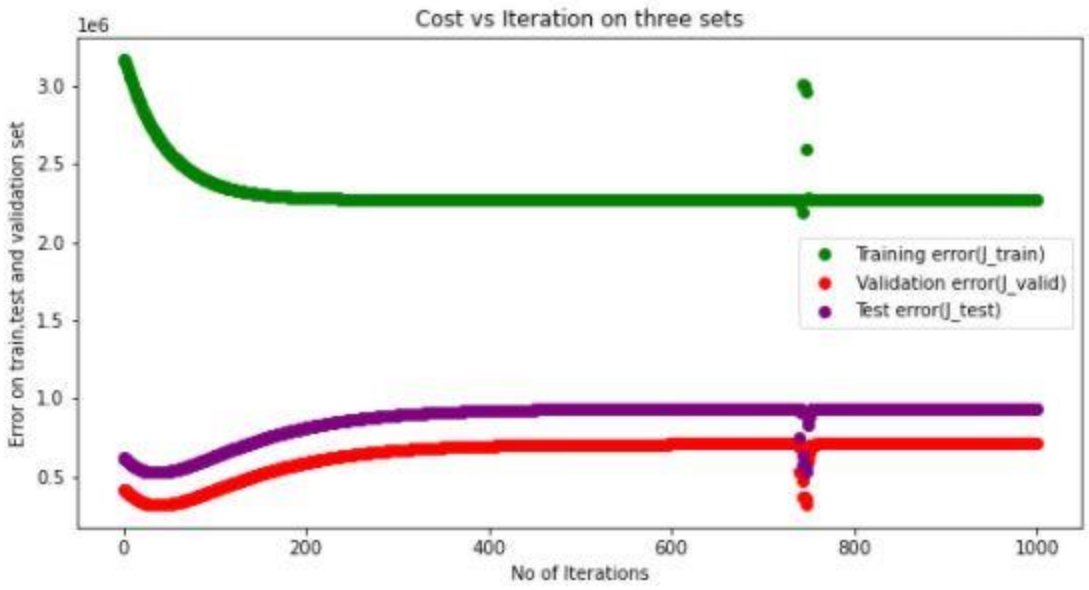
6. Plots:

➢ USA Dataset Plots:

The **training loss** plot with respect to iterations for USA data training set is given below:



Error Plot for all three data splits of USA Data set is given below:



- **Mean Absolute Error** for training set is given by:

MAE_train=np.sum(np.absolute(y_train.T-predictions_train.T))/len(x_train.T)

MAE_train = 1087.1478347699021

Mean absolute Error is `1312` for training set. It means that model is predicting `1312` values different from actual cases

- **Mean Absolute Error** for validation set is given by:

MAE_valid=np.sum(np.absolute(y_valid.T-predictions_valid.T))/len(x_valid.T)

MAE_validation= 1074

Mean absolute Error is `1074` for validation set. It means that model is predicting `1074` values wrong from actual cases

- **Mean Absolute Error** for test set is given by:

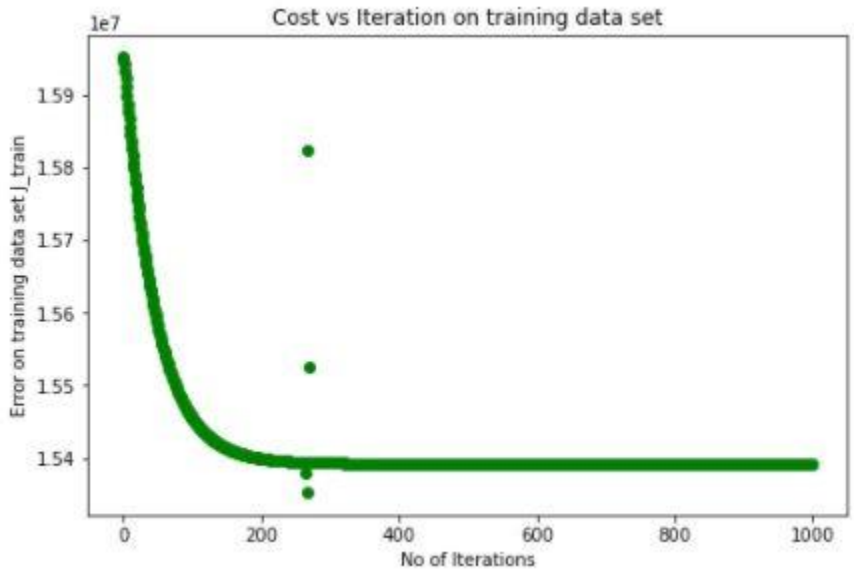
MAE_test=np.sum(np.absolute(y_test.T-predictions_test.T))/len(x_test.T)

MAE_test = 1170.2771819330865

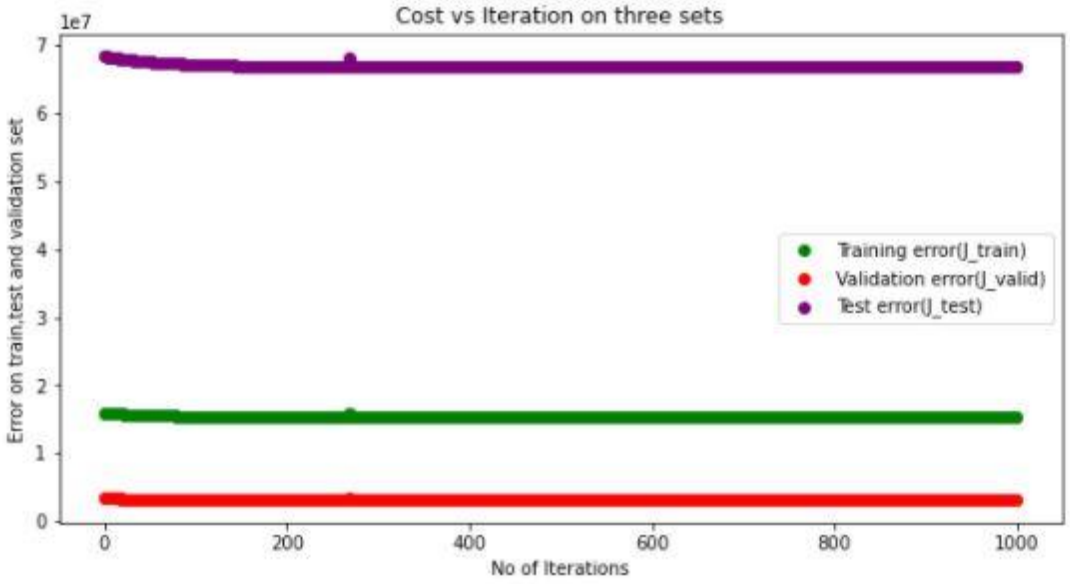
Mean absolute Error is `1170` for test set. It means that model is predicting **`1170`** values wrong from actual cases in test set.

➢ Worldwide Dataset Plots:

The **training loss** plot with respect to iterations for worldwide data training set is given below:



Error Plot for all three data splits of Worldwide Data set is given below:



- **Mean Absolute Error** for training set is given by:

MAE_train=np.sum(np.absolute(y_train.T-predictions_train.T))/len(x_train.T)

MAE_train = 1667.1478347699021

Mean absolute Error is `1667` for training set. It means that model is predicting `1667` values different from actual cases

- **Mean Absolute Error** for validation set is given by:

MAE_valid=np.sum(np.absolute(y_valid.T-predictions_valid.T))/len(x_valid.T)

MAE_validation= 1345

Mean absolute Error is `1345` for validation set. It means that model is predicting `1345` values wrong from actual cases

- **Mean Absolute Error** for test set is given by:

MAE_test=np.sum(np.absolute(y_test.T-predictions_test.T))/len(x_test.T)

MAE_test = 2670.2771819330865

Mean absolute Error is `2670` for test set. It means that model is predicting **`2670`** values wrong from actual cases in test set.

7. Complete Codes:

Annex A

➢ Instructions on running the code:

Four Complete notebooks with all the optimal parameters are provided with the data for both USA Prediction Model and Worldwide Model.

1) **"Usman Zaheer_Notebook_Neural Network Model for Predicting Corona Cases in USA."**
2) **"Usman Zaheer_Notebook_Neural Model for Predicting Corona Cases in World.**
3) **Prediction Code Neural Networks for USA.**
4) **Prediction Code Neural Networks for World.**

For the first two (1 and 2) above mentioned books:

- If user only wants to see the already trained books for both Datasets. Just open both notebooks named:

1) **"Usman Zaheer_Notebook_Neural Network Model for Predicting Corona Cases in USA."**
   This book is composed of USA prediction model, training code and all necessary steps.
   For good user experience, it is available in jupyter notebook format, pdf format as well as in HTML format.
2) **"Usman Zaheer_Notebook_Neural Network Model for Predicting Corona Cases in World.**
   This jupyter notebook is composed of worldwide/country wise prediction model, training code and all necessary steps taken.
   For good user experience, it is available in jupyter notebook format, pdf format as well as in HTML format.

For the last two (3 and 4) above mentioned books:

3) Jupyter notebook named **"Prediction Code Neural Networks for USA**" is composed of Prediction code for USA data set with all the commands and instructions for the user. User have to just run the book and do as it says.

4) Jupyter Notebook named **"Prediction Code Neural Networks for Worldwide**" is composed of Prediction code for Worldwide/Country wise data set with all the commands and instructions for the user. User have to just run the book and do as it says.

Annex B

➢ Training Code with Optimal Parameters:

Two complete Trained notebooks named are in the folder of data:

1) **"Usman Zaheer_Notebook_Neural Network Model for Predicting Corona Cases in USA."** is the name of notebook for USA model.

Training Code:

The neural network model used in this case is composed of **01** input layer, **02** hidden layers and **01** output layer.

For USA Data set architecture is:

-1) Input Layer (7 Neurons)
- 2) Hidden layer 1 (10 Neurons)
- 3) Hidden Layer 2 (10 Neurons)
- 4) Output Layer (1 Neuron)

Code for layer size definition is:

```
def layer_sizes(x, y):
    n_x = x.shape[0] # size of input layer
    n_h1 = 10
    n_h2 = 10
    n_y = y.shape[0] # size of output layer
    return (n_x, n_h1, n_h2, n_y)
```

Neural Network is implemented in this model, so for forward propagation the model parameters were initialized.

In this model, there are **06** parameters:

W1 -- weight matrix of shape (n_h1, n_x) b1 -- bias vector of shape (n_h1, 1)
W2 -- weight matrix of shape (n_h2, n_h1) b2 -- bias vector of shape (n_h2, 1)
W3 -- weight matrix of shape (n_y, n_h2) b3 -- bias vector of shape (n_y, 1)

Parameters initial values are initiated using this function:

```
def initialize_parameters(n_x, n_h1,n_h2, n_y)
    np.random.seed(2)
    W1 = np.random.randn(n_h1,n_x)*0.01
    b1 = np.zeros((n_h1,1))
    W2 = np.random.randn(n_h2,n_h1)*0.01
    b2 = np.zeros((n_h2,1))
```

```
W3 = np.random.randn(n_y,n_h2)*0.01
b3 = np.zeros((n_y,1))
parameters = {"W1": W1,
              "b1": b1,
              "W2": W2,
              "b2": b2,
              "W3": W3,
              "b3": b3}
return parameters
```

The forward propagation is used to calculates predictions using combination of features, parameters and bias terms:

Forward propagation definition is given by the following function:

Here:
x -- input data of size (n_x, m) **m** is the no of training examples
parameters -- python dictionary containing your parameters (output of initialization function)
A3 -- The output **cache --** a dictionary containing "Z1", "A1", "Z2", "A2" , "Z3" and "A3"

Forward Propagation Code is:

```
def forward_propagation(x, parameters):

    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]
    W3 = parameters["W3"]
    b3 = parameters["b3"]

    Z1 = np.dot(W1,x) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(W2,A1) + b2
    A2 = sigmoid(Z2)
    Z3 = np.dot(W3,A2) + b3
    A3 = Z3

    assert(A3.shape == (1, x.shape[1]))

    cache = {"Z1": Z1,
          "A1": A1,
          "Z2": Z2,
          "A2": A2,
          "Z3": Z3,
          "A3": A3
         }
```

```
    return A3, cache
```

Cost function is basically the difference between prediction by the model and the prediction label.

Cost function for linear regression is given by:

```
def compute_cost(A3, y, parameters,lambda_):

    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]
    W3 = parameters["W3"]
    b3 = parameters["b3"]
    m = y.shape[1]
cost=(np.sum(np.power((A3-
y_train),2))+lambda_*(np.sum(np.power(W1,2))+np.sum(np.power(W2,2))+np.sum(n
p.power(W3,2))))/(2*m)
    cost = np.squeeze(cost)
    assert(isinstance(cost, float))
    return cost
```

here,

x_train = features in training set. **y_train =** prediction label in training set.

lambda_ = regularization parameter. **m** = the length of training set.

Backpropagation is recursive application of the chain rule along a computational graph to compute the gradients of all parameters / intermediates.

Back Propagation is given by following code:

```
def backward_propagation(parameters, cache, x, y,lambda_):

    m = x.shape[1]

    W1 = parameters["W1"]
    W2 = parameters["W2"]
    W3 = parameters["W3"]

    A1 = cache["A1"]
    A2 = cache["A2"]
    A3 = cache["A3"]

    dZ3 = A3-y
    dW3 = 1/m*(np.dot(dZ3,A2.T)) + (lambda_/m)*W3
    db3 = 1/m*(np.sum(dZ3,axis=1, keepdims=True))
```

```
dZ2 = np.multiply(np.dot(W3.T,dZ3),(A2*(1-A2)))
dW2 = 1/m*(np.dot(dZ2,A1.T)) + (lambda_/m)*W2
db2 = 1/m*(np.sum(dZ2,axis=1, keepdims=True))
dZ1 = np.multiply(np.dot(W2.T,dZ2),(A1*(1-A1)))
dW1 = 1/m*(np.dot(dZ1,x.T)) + (lambda_/m)*W1
db1 = 1/m*(np.sum(dZ1,axis=1, keepdims=True))

grads = {"dW1": dW1,
      "db1": db1,
      "dW2": dW2,
      "db2": db2,
      "dW3": dW3,
      "db3": db3}
return grads
```

Here,
parameters -- python dictionary containing our parameters
cache -- a dictionary containing "Z1", "A1", "Z2" "A2", "Z3" and "A3"
x -- input data of shape (7/5, number of examples)
y -- "true" labels vector of shape (1, number of examples)
lambda_ -- Regularization parameter
grads -- python dictionary containing your gradients with respect to parameters of model.

Gradient Descent is used to find the minimum values of thetas, so that our cost will be minimum. Minimum cost indicates that the difference between our prediction and actual label is very low. Regularization is also applied with Gradient Descent to prevent overfitting.

Gradient Descent is given by:

```
def update_parameters(parameters, grads, alpha = 0.001):

   W1 = parameters["W1"]
   b1 = parameters["b1"]
   W2 = parameters["W2"]
   b2 = parameters["b2"]
   W3 = parameters["W3"]
   b3 = parameters["b3"]

   dW1 = grads["dW1"]
   db1 = grads["db1"]
   dW2 = grads["dW2"]
   db2 = grads["db2"]
   dW3 = grads["dW3"]
   db3 = grads["db3"]

   W1 = W1 - alpha*dW1
```

```
    b1 = b1 - alpha*db1
    W2 = W2 - alpha*dW2
    b2 = b2 - alpha*db2
    W3 = W3 - alpha*dW3
    b3 = b3 - alpha*db3

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2,
                  "W3": W3,
                  "b3": b3}
    return parameters
```

Here,

alpha = learning rate **Parameters** = Model Parameter after gradient descent.

All the functions are integrated in the neural network model function. The code for following functions is given below:

```
def nn_model(x, y, n_h1,n_h2,num_iterations=1000 ,print_cost=False, lambda_= 0.7):
    np.random.seed(3)
    n_x = layer_sizes(x, y)[0]
    n_y = layer_sizes(x, y)[3]
    cost_history_train=[]
    cost_history_valid=[]
    cost_history_test= []
    parameters = initialize_parameters(n_x,n_h1,n_h2, n_y)
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]
    W3 = parameters["W3"]
    b3 = parameters["b3"]

  for i in range(0, num_iterations):
        A3, cache = forward_propagation(x_train, parameters)
        A3_valid,cache1 = forward_propagation(x_valid, parameters)
        A3_test, cache2 = forward_propagation(x_test, parameters)
        cost_train = compute_cost(A3,y_train,parameters,lambda_)
        cost_history_train.append(cost_train)
        cost_valid = compute_cost(A3_valid,y_valid,parameters,lambda_)
        cost_history_valid.append(cost_valid)
        cost_test = compute_cost(A3_test,y_test,parameters,lambda_)
        cost_history_test.append(cost_test)
        grads = backward_propagation(parameters,cache,x,y,lambda_)
```

```
        parameters = update_parameters(parameters,grads)
        if print_cost and i % 1== 0:
            print ("Cost after iteration %i: %f" %(i, cost_train))
    return parameters, cost_history_train,cost_history_valid,cost_history_test
```

Here,

alpha is learning rate **=0.001**
iterations**=1000**
lambda_ is regularization parameter**= 0.7**

2) **"Usman Zaheer_Notebook_Neural Networks Model for Predicting Corona Cases in World."** Is the name of Worldwide prediction model.
    - In this notebook, complete training code for prediction of cases in USA is included along with the explanations and instructions.
    - PDF and HTML version of this notebook is also available in the folder.

Training Code For Worldwide Data set:

For Worldwide Data set architecture is:

- 1) Input Layer (5 Neurons)
- 2) Hidden layer 1 (10 Neurons)
- 3) Hidden Layer 2 (10 Neurons)
- 4) Output Layer (1 Neuron)

Code for layer size definition is:

```
def layer_sizes(x, y):
    n_x = x.shape[0] # size of input layer
    n_h1 = 10
    n_h2 = 10
    n_y = y.shape[0] # size of output layer
    return (n_x, n_h1, n_h2, n_y)
```

Neural Network is implemented in this model, so for forward propagation the model parameters were initialized.

In this model, there are **06** parameters:

W1 -- weight matrix of shape (n_h1, n_x) b1 -- bias vector of shape (n_h1, 1)
W2 -- weight matrix of shape (n_h2, n_h1) b2 -- bias vector of shape (n_h2, 1)
W3 -- weight matrix of shape (n_y, n_h2) b3 -- bias vector of shape (n_y, 1)

Parameters initial values are initiated using this function:

```
def initialize_parameters(n_x, n_h1,n_h2, n_y)
    np.random.seed(2)
    W1 = np.random.randn(n_h1,n_x)*0.01
```

```
b1 = np.zeros((n_h1,1))
W2 = np.random.randn(n_h2,n_h1)*0.01
b2 = np.zeros((n_h2,1))
W3 = np.random.randn(n_y,n_h2)*0.01
b3 = np.zeros((n_y,1))
parameters = {"W1": W1,
        "b1": b1,
        "W2": W2,
        "b2": b2,
        "W3": W3,
        "b3": b3}
return parameters
```

The forward propagation is used to calculates predictions using combination of features, parameters and bias terms:

Forward propagation definition is given by the following function:

Here:
x -- input data of size (n_x, m) **m** is the no of training examples
parameters -- python dictionary containing your parameters (output of initialization function)
A3 -- The output **cache --** a dictionary containing "Z1", "A1", "Z2", "A2" , "Z3" and "A3"

Forward Propagation Code is:

```
def forward_propagation(x, parameters):

    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]
    W3 = parameters["W3"]
    b3 = parameters["b3"]

    Z1 = np.dot(W1,x) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(W2,A1) + b2
    A2 = sigmoid(Z2)
    Z3 = np.dot(W3,A2) + b3
    A3 = Z3

    assert(A3.shape == (1, x.shape[1]))

    cache = {"Z1": Z1,
         "A1": A1,
         "Z2": Z2,
         "A2": A2,
```

```
                "Z3": Z3,
                "A3": A3
               }
            return A3, cache
```

Cost function is basically the difference between prediction by the model and the prediction label.

Cost function for linear regression is given by:

```
def compute_cost(A3, y, parameters,lambda_):

    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]
    W3 = parameters["W3"]
    b3 = parameters["b3"]
    m = y.shape[1]
cost=(np.sum(np.power((A3-
y_train),2))+lambda_*(np.sum(np.power(W1,2))+np.sum(np.power(W2,2))+np.sum(n
p.power(W3,2))))/(2*m)
    cost = np.squeeze(cost)
    assert(isinstance(cost, float))
    return cost
```

here,

x_train = features in training set. **y_train =** prediction label in training set.

lambda_ = regularization parameter. **m** = the length of training set.

Backpropagation is recursive application of the chain rule along a computational graph to compute the gradients of all parameters / intermediates.

Back Propagation is given by following code:

```
def backward_propagation(parameters, cache, x, y,lambda_):

    m = x.shape[1]

    W1 = parameters["W1"]
    W2 = parameters["W2"]
    W3 = parameters["W3"]

    A1 = cache["A1"]
    A2 = cache["A2"]
    A3 = cache["A3"]

    dZ3 = A3-y
```

```
dW3 = 1/m*(np.dot(dZ3,A2.T)) + (lambda_/m)*W3
db3 = 1/m*(np.sum(dZ3,axis=1, keepdims=True))
dZ2 = np.multiply(np.dot(W3.T,dZ3),(A2*(1-A2)))
dW2 = 1/m*(np.dot(dZ2,A1.T)) + (lambda_/m)*W2
db2 = 1/m*(np.sum(dZ2,axis=1, keepdims=True))
dZ1 = np.multiply(np.dot(W2.T,dZ2),(A1*(1-A1)))
dW1 = 1/m*(np.dot(dZ1,x.T)) + (lambda_/m)*W1
db1 = 1/m*(np.sum(dZ1,axis=1, keepdims=True))

grads = {"dW1": dW1,
      "db1": db1,
      "dW2": dW2,
      "db2": db2,
      "dW3": dW3,
      "db3": db3}
return grads
```

Here,

parameters -- python dictionary containing our parameters
cache -- a dictionary containing "Z1", "A1", "Z2" "A2", "Z3" and "A3"
x -- input data of shape (7/5, number of examples)
y -- "true" labels vector of shape (1, number of examples)
lambda_ -- Regularization parameter
grads -- python dictionary containing your gradients with respect to parameters of model.

Gradient Descent is used to find the minimum values of thetas, so that our cost will be minimum. Minimum cost indicates that the difference between our prediction and actual label is very low. Regularization is also applied with Gradient Descent to prevent overfitting.

Gradient Descent is given by:

```
def update_parameters(parameters, grads, alpha = 0.001):

   W1 = parameters["W1"]
   b1 = parameters["b1"]
   W2 = parameters["W2"]
   b2 = parameters["b2"]
   W3 = parameters["W3"]
   b3 = parameters["b3"]

   dW1 = grads["dW1"]
   db1 = grads["db1"]
   dW2 = grads["dW2"]
   db2 = grads["db2"]
   dW3 = grads["dW3"]
   db3 = grads["db3"]
```

```
W1 = W1 - alpha*dW1
b1 = b1 - alpha*db1
W2 = W2 - alpha*dW2
b2 = b2 - alpha*db2
W3 = W3 - alpha*dW3
b3 = b3 - alpha*db3

parameters = {"W1": W1,
        "b1": b1,
        "W2": W2,
        "b2": b2,
        "W3": W3,
        "b3": b3}
return parameters
```

Here,

alpha = learning rate **Parameters** = Model Parameter after gradient descent.

All the functions are integrated in the neural network model function. The code for following functions is given below:

```
def nn_model(x, y, n_h1,n_h2,num_iterations=1000 ,print_cost=False, lambda_= 0.7):
    np.random.seed(3)
    n_x = layer_sizes(x, y)[0]
    n_y = layer_sizes(x, y)[3]
    cost_history_train=[]
    cost_history_valid=[]
    cost_history_test= []
    parameters = initialize_parameters(n_x,n_h1,n_h2, n_y)
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]
    W3 = parameters["W3"]
    b3 = parameters["b3"]

  for i in range(0, num_iterations):
        A3, cache = forward_propagation(x_train, parameters)
        A3_valid,cache1 = forward_propagation(x_valid, parameters)
        A3_test, cache2 = forward_propagation(x_test, parameters)
        cost_train = compute_cost(A3,y_train,parameters,lambda_)
        cost_history_train.append(cost_train)
        cost_valid = compute_cost(A3_valid,y_valid,parameters,lambda_)
        cost_history_valid.append(cost_valid)
        cost_test = compute_cost(A3_test,y_test,parameters,lambda_)
        cost_history_test.append(cost_test)
```

```
        grads = backward_propagation(parameters,cache,x,y,lambda_)
        parameters = update_parameters(parameters,grads)
        if print_cost and i % 1== 0:
            print ("Cost after iteration %i: %f" %(i, cost_train))
    return parameters, cost_history_train,cost_history_valid,cost_history_test
```

Here,

alpha is learning rate **=0.001**
iterations**=1000**
lambda_ is regularization parameter**= 0.7**

Annex C

➢ Prediction Code:

In the data folder two more notebooks are included. Users have to just run the books and enter the features as instructed by the program.

1) Prediction Code for USA:

```
import pandas as pd
import numpy as np
```

Instructions/Features Information:

Here, We have 7 valid features: (Master sheet for Reference' file can be used for reference.)

- *1) DATE_CODE = In this model "date" is numerically coded. For example:
Model date starts from 6th December 2020 and ends at 17th March 2020.
- *Here: 17th March 2020 is 264 and 7th December is 0.
- *2) States_Code: States codes are also numerically coded. For example:
*Here: 1 = Alaska, 2= Alabama and 56=Wyoming.
- *3) Temperatures_(F): Tempearures in F of particular state at particular date.
- *4) Humidity_(%): Average humidity in % of particular state at particular date.
- *5) Population_per_state: Average populationin of particular state at particular date.
- *6) LandArea_(sq miles): Land Area in square miles of particular state .
- *7) POPULATION_DENSITY: Population Density of particular state.

The below code will ask user to enter features for this model as per his desire one by one when code is run.

Storing max and min values of each feature's column of worldwide data for normalization/scaling of user input features.

```
# Date_Code  max and min
D_max = 264
D_min= 0
# States_Code max and min
S_max= 56
S_min=1
# Temperatures_(F)  max and min
T_max= 82.80
T_min= 0
# Humidity_(%)   max and min
H_max= 80
H_min= 38.30
# Population_per_state max and min
P_max= 39512223.0
P_min= 55138.00
```

```
# LandArea_(sq miles) max and min
LA_max=  570641.0
LA_min = 68.00
# Population_Density max and min
PD_max= 11535.0
PD_min= 1.29

def Date_numerical_coding(year,month,day):
   import datetime
   date = datetime.date(2020, 12, 6)
   date1 = datetime.date(year,month,day)
   date_code = (date-date1)
   dayss = date_code.days
   print ('Date_code is:'  ,dayss)
   return dayss
```

Getting 6 features as input from user:

Getting date from User

```
year = int(input('Enter a year: '))
month = int(input('Enter a month: '))
day = int(input('Enter a day: '))
D= Date_numerical_coding(year,month,day)
D = (D-D_min)/(D_max-D_min)
print("Normalized date code is:", D)
Enter a year: 2020
Enter a month: 3
Enter a day: 17
Date_code is: 264
Normalized date code is: 1.0
```

Getting state code from User.

Note: For State codes, please look into "Master sheet for Reference" data file provided.

```
S = float(input('Enter state Code: '))
S = (S-S_min)/(S_max-S_min)
print("Normalized state code  is:", S)
#here for example, 1= Alaska
Enter state Code: 1
Normalized state code  is: 0.0
```

Getting temperature in Farhenheit from User

```
T = float(input('Enter value of Temperature in F for particular state: '))
T = (T-T_min)/(T_max-T_min)
print("Normalized temperature  is:", T)
```

Enter value of Temperature in F for particular state: 32
Normalized temperature is: 0.3864734299516908

Getting Humidity in % from User

```
H = float(input('Enter value of Humidity in % for particular state: '))
H = (H-H_min)/(H_max-H_min)
print("Normalized humidity  is:", H)
```

Enter value of Humidity in % for particular state: 32
Normalized humidity is: -0.1510791366906474

Getting Population of state from User

```
P = float(input('Enter population of state: '))
P = (P-P_min)/(P_max-P_min)
print("Normalized population  is:", P)
```

Enter population of state: 10000
Normalized population is: -0.0011439770576057507

Getting Landarea in square miles of state from User

```
LA = float(input('Enter land area in sq miles of state: '))
LA= (LA-LA_min)/(LA_max-LA_min)
print("Normalized land area  is:", LA)
```

Enter land area in sq miles of state: 27373
Normalized land area is: 0.04785540149989572

Getting Population Density per square miles of state from User

```
P_max-P_min
PD = float(input('Enter population density of state: '))
PD= (PD-PD_min)/(P_max-P_min)
print("Normalized population density  is:", PD)
```

Enter population density of state: 1000
Normalized population density is: 2.531129707123575e-05

Making array of user input features as per our model:

```
User_input = np.array([D,S,T,H,P,LA,PD])
User_input
array([ 1.00000000e+00,  1.00000000e+00,  0.00000000e+00,  3.86473430e-01,
       -1.51079137e-01, -1.14397706e-03,  4.78554015e-02,  2.53112971e-05])
```

Model paramters of USA dataset from our trained model:

```
W1 =np.array([[-4.99595378, -4.34116041, -4.81039714, -5.6752964 ,  3.44311596,
        0.16035013, -1.23734626],
       [-5.64168892, -5.2055178 , -5.81175153, -6.80090593,  3.47275491,
        0.08248535, -1.3917826 ],
       [-5.0858471 , -4.47559595, -4.95835867, -5.851394  ,  3.45181602,
```

```
        0.1527502 , -1.26943497],
       [-5.20577411, -4.62517815, -5.14321067, -6.06182593,  3.45493945,
        0.13217548, -1.29553957],
       [-5.06674018, -4.40508723, -4.87901833, -5.75561167,  3.46041963,
        0.16561876, -1.25264025],
       [-5.24167924, -4.65985764, -5.18902664, -6.10989157,  3.50499865,
        0.13930489, -1.29324733],
       [-5.50022244, -4.97400822, -5.55209658, -6.52819468,  3.47988875,
        0.10681683, -1.35440319],
       [-5.19822427, -4.60126643, -5.12171186, -6.03977143,  3.4711527 ,
        0.14690022, -1.2767362 ],
       [-5.35962955, -4.7932066 , -5.34000138, -6.29552775,  3.49719894,
        0.12790942, -1.32177476],
       [-5.0107371 , -4.39348398, -4.85443698, -5.7483299 ,  3.42959349,
        0.16365012, -1.2432638 ]])
b1 = np.array([[-7.86717492],
       [-9.41158856],
       [-8.10679542],
       [-8.37972595],
       [-7.9874055 ],
       [-8.45314174],
       [-9.02977076],
       [-8.3525262 ],
       [-8.68784378],
       [-7.94094793]])
W2= np.array([[-0.36137846, -0.72256052, -0.40614801, -0.46957085, -0.38095142,
       -0.4539434 , -0.64261864, -0.45687374, -0.5230878 , -0.39741066],
       [-0.22379668, -0.55234983, -0.25411899, -0.30950735, -0.23878445,
       -0.33580856, -0.43648114, -0.29731796, -0.37978196, -0.23716305],
       [-0.28880363, -0.63828291, -0.33399947, -0.38533549, -0.32374145,
       -0.38878031, -0.53212754, -0.38825664, -0.44396812, -0.33629395],
       [-0.27408307, -0.63032071, -0.34004452, -0.38726602, -0.30240553,
       -0.36565457, -0.50871948, -0.37400997, -0.45479353, -0.31152892],
       [-0.32320981, -0.6871627 , -0.37393792, -0.43596869, -0.3368803 ,
       -0.43477497, -0.57744832, -0.42560114, -0.46215147, -0.35574162],
       [-0.28196233, -0.61242445, -0.31755414, -0.37234274, -0.27722614,
       -0.36376228, -0.50313669, -0.37343435, -0.42083065, -0.30922071],
       [-0.31819086, -0.68196095, -0.37237265, -0.42072651, -0.33302817,
       -0.41612754, -0.58126974, -0.40453055, -0.47915954, -0.33666654],
       [-0.30230882, -0.64045345, -0.3468099 , -0.41526158, -0.26900945,
       -0.38452526, -0.51246744, -0.38547073, -0.44354672, -0.30128231],
       [-0.27382127, -0.62712515, -0.32500381, -0.40607993, -0.28731146,
       -0.38096062, -0.50919516, -0.37496371, -0.4236713 , -0.30428089],
       [-0.30061409, -0.62776303, -0.33583548, -0.40668909, -0.3005112 ,
```

```
       -0.38650187, -0.52744003, -0.37597032, -0.40917541, -0.29987871]])
b2= np.array([[ 2.19228576],
       [14.17746253],
       [14.95635554],
       [21.37781459],
       [ 4.29498662],
       [23.4105892 ],
       [ 4.15239371],
       [18.36297356],
       [23.78061538],
       [19.91399117]])
W3 = np.array([[121.8902599 , 123.03848017, 122.30794508, 122.44789693,122.09279541,
122.63040987, 122.15645207, 122.34785173, 122.47316167, 122.42830373]])
b3=np.array([[129.57107768]])
parameters = {"W1": W1,
              "b1": b1,
              "W2": W2,
              "b2": b2,
              "W3": W3,
              "b3": b3}
```

The Sigmoid Function:

```
def sigmoid(z):
      return 1.0/(1.0+np.exp(-z))
```

The forward Propagation:

```
def forward_propagation(x, parameters):
   W1 = parameters["W1"]
   b1 = parameters["b1"]
   W2 = parameters["W2"]
   b2 = parameters["b2"]
   W3 = parameters["W3"]
   b3 = parameters["b3"]

   Z1 = np.dot(W1,x) + b1
   A1 = sigmoid(Z1)
   Z2 = np.dot(W2,A1) + b2
   A2 = sigmoid(Z2)
   Z3 = np.dot(W3,A2) + b3
   A3 = (Z3)
   cache = {"Z1": Z1,
         "A1": A1,
```

```
        "Z2": Z2,
        "A2": A2,
        "Z3": Z3,
        "A3": A3
       }
  return A3, cache
```

The prediction function:

```
def predict(parameters, x):
  A3, cache = forward_propagation(x,parameters)
  predictions =np.mean(A3)
  return predictions

Output =Prediction(parameters,User_input)

      print("Number of Corona Cases predicted by model are:", Output)
```

Number of Corona Cases predicted by model are: 553.8012166681253

Enter actual Cases on the selected day for selected state

```
      actual_cases = int(input('Enter Number of Actual Cases: '))

      actual_cases
```

Enter Number of Actual Cases: 200

Calculating Mean Absolute Error

```
      MAE_user =np.sum(np.absolute(actual_cases-Output))
      MAE_user= 353.80121666812533
```

MAE_user will give the difference between actual cases and model predicted cases on the selected day for the selectd state of USA.

2) Prediction Code for Worldwide/Country Wise:

Instructions/Features Information

Here, We have 5 features: ('Worldwide Final.csv & Master sheet for Reference' files can be used for reference.)

*1) DATE_CODE = In this model "date" is numerically coded. For example:
Model date starts from 23 January 2020 and ends at 7th December 2020.
*Here: 23/01/2020 is 0 and 7th December is 319.

*2) COUNTRY_CODE: Country codes are also numerically coded. For example:
*Here: 0 = Afghanistan, 2= Albania and 170=Zimbabwe.

*3) HUMAN_DEVELOPMENT_INDEX: Average Human development index of particular state

*4) POPULATION_DENSITY: Population Density of particular country.
*5) TEMPERATURES: Tempearures in C of particular country at particular date.

The below code will ask user to enter features for this model as per his desire one by one when code is run.
Storing max and min values of each feature's column of worldwide data for normalization/scaling of user input features.

```
# DATE_CODE max and min
D_max = 319
D_min= 0
# COUNTRY_CODE max and min
C_max= 170
C_min=0
# POPULATION_DENSITY max and min
P_max= 19347.5
P_min = 1.98
# HUMAN_DEVELOPMENT_INDEX max and min
HDI_max= 0.953
HDI_min= 0.354
 # TEMPERATURES  max and min
T_max= 52.0
T_min= -16
```

Creating function for numerical coding of date:

```
def Date_numerical_coding(year,month,day):
   import datetime
   date = datetime.date(2020, 1, 23) #fixing the date of my model as reference
   date1 = datetime.date(year,month,day)
   date_code = (date1 - date)
   dayss = date_code.days
   print ('Date_code is:'  ,dayss)
   return days

def Date_numerical_coding(year,month,day):
   import datetime
   date = datetime.date(2020, 1, 23) #fixing the date of my model as reference
   date1 = datetime.date(year,month,day)
   date_code = (date1 - date)
   dayss = date_code.days
   print ('Date_code is:'  ,dayss)
   return dayss
Getting date from User
   year = int(input('Enter a year: '))
   month = int(input('Enter a month: '))
```

```
day = int(input('Enter a day: '))
D= Date_numerical_coding(year,month,day)
D = (D-D_min)/(D_max-D_min)
print("Normalized date code is:", D)
year = int(input('Enter a year: '))
month = int(input('Enter a month: '))
day = int(input('Enter a day: '))
D= Date_numerical_coding(year,month,day)
D = (D-D_min)/(D_max-D_min)
print("Normalized date code is:", D)
```

Enter a year: 2020
Enter a month: 12
Enter a day: 7
Date_code is: 319
Normalized date code is: 1.0

Getting country code from User.

Note: For Country codes, please look into "Master sheet for Reference" data file provided.

```
C = float(input('Enter Country Code: '))
C = (C-C_min)/(C_max-C_min)
print("Normalized country  is:", C)
#here for example, 35= China
Enter Country Code: 34
Normalized country  is: 0.2
```

Getting Population density from User

```
P = float(input('Enter population Density of selected country: '))
P = (P-P_min)/(P_max-P_min)
print("Normalized population density  is:", P)
```

Enter population Density of selected country: 45
Normalized population density is: 0.0022237706714526156

Getting Human Development index from User

```
HDI = float(input('Enter value of Human Devlopment Index of selected country: '))
HDI = (HDI-HDI_min)/(HDI_max-HDI_min)
print("Normalized human development index is:", HDI)
```

Enter value of Human Devlopment Index of selected country: 0.4
Normalized human development index is: 0.0767946577629383

Getting temperature in Centigrade from User

```
T = float(input('Enter value of Temperature in C for particular country: '))
T = (T-T_min)/(T_max-T_min)
print("Normalized temperature  is:", T)
```

Enter value of Temperature in C for particular country: -16
Normalized temperature is: 0.0

 # Making array of user input features as per our model

User_input = np.array([D,C,P,HDI,T])
User_input
array([1. , 0.9968652 , 0.20588235, 0.00222377, 0.07679466, 0.])

Model parameters of Worldwide dataset from our trained model:

```
W1=np.array([[ 2.49052515e+00, -1.41587663e+00, -2.24979023e-02,
       -1.61749964e+00, -2.97950958e+00],
      [ 2.48554495e+00, -1.39331416e+00, -1.32458888e-02,
       -1.60989047e+00, -2.93749532e+00],
      [ 2.48729512e+00, -1.38498563e+00, -6.49461500e-04,
       -1.62411839e+00, -2.93952186e+00],
      [ 2.49586997e+00, -1.41492302e+00,  1.07044856e-02,
       -1.63200891e+00, -2.96268609e+00],
      [ 2.47846668e+00, -1.38891335e+00,  1.90326327e-03,
       -1.59376571e+00, -2.91119869e+00],
      [ 2.51821847e+00, -1.45950186e+00, -1.34617903e-02,
       -1.69595146e+00, -3.04265966e+00],
      [ 2.49853906e+00, -1.39899613e+00, -2.54774647e-02,
       -1.63175804e+00, -2.96825227e+00],
      [ 2.51133949e+00, -1.44962765e+00, -1.02664269e-02,
       -1.68969956e+00, -3.03583750e+00],
      [ 2.52436607e+00, -1.45673036e+00, -2.04036677e-02,
       -1.68388757e+00, -3.04693834e+00],
      [ 2.46068321e+00, -1.36047151e+00,  6.26701533e-07,
       -1.56274718e+00, -2.86365578e+00]])
b1= np.array([[-4.8752387 ],
      [-4.81535199],
      [-4.84092062],
      [-4.87191105],
      [-4.78060417],
      [-5.00153555],
      [-4.88761146],
      [-5.00277676],
      [-5.0377857 ],
      [-4.70836228]])
W2= np.array([[0.06874486, 0.05357128, 0.0561491 , 0.05502158, 0.06384042,
       0.06008336, 0.0523321 , 0.05812531, 0.05689442, 0.04695652],
      [0.05882273, 0.06034864, 0.06377285, 0.06333885, 0.04337895,
       0.05212755, 0.05621606, 0.04294517, 0.06436573, 0.07711738],
      [0.05733386, 0.07094791, 0.06119869, 0.06159924, 0.05364253,
       0.06837864, 0.03725173, 0.05728093, 0.0669405 , 0.05026683],
      [0.05509448, 0.0548363 , 0.06634104, 0.06646129, 0.05367198,
       0.0317268 , 0.07051105, 0.06346974, 0.04825192, 0.06722229],
```

```
       [0.06151655, 0.06199892, 0.06090421, 0.06878275, 0.04108204,
        0.05665022, 0.06143124, 0.04982143, 0.06540274, 0.04106953],
       [0.0654061 , 0.05626873, 0.04361638, 0.05510041, 0.0514953 ,
        0.06812823, 0.07201905, 0.0524817 , 0.04245616, 0.05477491],
       [0.06026011, 0.05656756, 0.05566336, 0.05467334, 0.06152525,
        0.04697967, 0.05653212, 0.04854977, 0.0851604 , 0.05555248],
       [0.04795923, 0.0611361 , 0.05608354, 0.05943196, 0.06692028,
        0.05937298, 0.0656099 , 0.0425341 , 0.06523553, 0.04730075],
       [0.05779088, 0.0525982 , 0.04951575, 0.06188352, 0.05787994,
        0.05767182, 0.04407544, 0.061698  , 0.05993324, 0.0670338 ],
       [0.04280782, 0.05311047, 0.04267076, 0.03317788, 0.0905414 ,
        0.05526225, 0.07481997, 0.04700048, 0.05989268, 0.07074627]])
b2 = np.array([[11.72683245],
       [11.77536369],
       [11.78070127],
       [11.61678389],
       [11.71054353],
       [11.62637271],
       [11.80169095],
       [11.69644132],
       [11.75672572],
       [11.66889467]])
W3=np.array([[95.99716235, 95.97117913, 95.96310633, 95.93920582, 95.99995654,
        96.00632301, 95.99780528, 95.97715026, 96.02445262, 95.97641958]]),
b3=np.array([[98.86402433]])

parameters = {"W1": W1,
              "b1": b1,
              "W2": W2,
              "b2": b2,
              "W3": W3,
              "b3": b3}
```

The Sigmoid Function:

```
def sigmoid(z):
    return 1.0/(1.0+np.exp(-z))
```

The forward Propagation:

```
def forward_propagation(x, parameters):
  W1 = parameters["W1"]
  b1 = parameters["b1"]
  W2 = parameters["W2"]
```

```
    b2 = parameters["b2"]
    W3 = parameters["W3"]
    b3 = parameters["b3"]

    Z1 = np.dot(W1,x) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(W2,A1) + b2
    A2 = sigmoid(Z2)
    Z3 = np.dot(W3,A2) + b3
    A3 = (Z3)
    cache = {"Z1": Z1,
             "A1": A1,
             "Z2": Z2,
             "A2": A2,
             "Z3": Z3,
             "A3": A3
            }
    return A3, cache
```

The prediction function:

```
def predict(parameters, x):
    A3, cache = forward_propagation(x,parameters)
    predictions =np.mean(A3)
    return predictions

Output =Prediction(parameters,User_input)

        print("Number of Corona Cases predicted by model are:", Output)
```

Number of Corona Cases predicted by model are: 553.8012166681253

```
         return Cases
        Output =Prediction(Thetas,User_input)
        print("Number of Corona Cases predicted by model are:", Output)
```

Number of Corona Cases predicted by model are: 1584.6019347508002

```
Enter actual Cases on the selected day for selected country
        actual_cases = int(input('Enter Number of Actual Cases: '))
        actual_cases
```

Enter Number of Actual Cases: 2000

```
Calculating Mean Absolute Error

        MAE_user =np.sum(np.absolute(actual_cases-Output))
```

MAE_user = 415.3980652491998

`MAE_user` will give the difference between actual cases and model predicted cases on the selected day for the selected country.