CORONA CASES PREDICTION USING NEURAL NETWORK

Machine Learning Khalil Ahmad

INTRODUCTION:

Neural networks basically work in the way to find relationships between data, similar to the brain of humans. Neural networks that are basically the sub-group of non-linear models, and are used in the situation where the relationship between data is very complex. In this project I used a 2 hidden layer neural network. The details of the model are given below.

1: Data Set

The features and labels are in two different csv files. The files for world wide prediction are in the folder "world wide" and files for USA are in the 'usa' folder.

1.1 : World wide data set size and detail

In world wide data set total number of samples are 66967, and total number of features taken are '17'. Theses features from 1st to last column are given in below sequence.

"continent, total_cases, total_deaths, new_deaths, reproduction_rate, icu_patients, hosp_patients, weekly_icu_admissions, weekly_hosp_admissions, new_tests, total_tests, positive_rate, population, median_age, aged_65_older, aged_70_older, cardiovasc_death_rate".

The world wide prediction is done country wise, so all the features in the above list should be of the country for which you want to predict cases. In the first column for entering the location of the country you just need to give the continent number (a file with the name "numbers assigned to each contenent" is created in which a number is given to each continent).

Total features matrix shape for world wide data is, 66967 X 17.

1.2 : Code and Methodology

The code is written in python using numpy library. Google colab is used for running the code. Google colab provide easy implementation for data. Data files are saved in csv format.

First when you run the code it will require for input features,so enter in features file, After that enter the label file.

2.1 implemented model

Basically, it has total of three layers. Fig 1: shows the detail of the neural network model.

```
l1  =  np.dot(X,self.w1) + self.b1
a_l1 = self.relu(l1)
l2  =  np.dot(a_l1,self.w2.T)   + self.b2
a_l2 = self.relu(l2)
output  =  np.dot(a_l2,self.w3.T)   + self.b3
```

Figure 1:Neural Network

L1 is layer number one whose inputs are features (x) and w1 and b1 are the weights and bias of layer number one. a_l1 is the output of layer 1 after applying the relu activation function.

L2 is layer number 2, whose input matrix is the output of layer 2 and w2 and b2 are the weights and bias for layer 2. a_l2 is the output of layer 2 after applying the relu activation function.

output is layer number 3, whose input matrix is the output of layer 2 after activation and w2, b2 are the weights and bias for layer 2. a_l2 is the output of layer 2 after applying the relu activation function.

The details of the dimensions of input matrix and weight matrices are given in below sections.

2.2: objective function

We used mean square error function for calculating the cost of the model. L2 regularization is also used in the cost function. Mathematical equation of cost function is given as,

$$J = \sum_{1}^{n}(h(x) - y) + l2\_regularization$$

Here h(x) is the predicted value of the model, y is the label data and l2 regularization is also added in the cost function. mathematical model is described below for l2_regularization.

2.3: l2 Regularization

The models that uses l2 regularization are called ridge regression. In l2 regression a squared term of weights is added in the cost function. mathematical model is,

$$J = \sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Here the part that is highlighted is the l2 regularization part. Here lamda is the regularization term. Much higher values will tend the model towards under fitting while low values will tend it to over fitting. Beta term here is the weights of the system.

2.4: parameter optimization

The weights are updated using the l2 regularization term in gradient also. The mathematical form for gradient is given below.

$$\text{Gradient} = J(W; X, y) + \lambda \cdot ||W||^2$$

Here j(w;x,y) is the plain loss equation while the lamda* weight part is the extra term of the l2 regularizatoin.

When the gradient is calculated the weights are updated as,

$$W = w - alpha * gradient$$

Here w is the weight matrix, alpha is the learning rate and gradient is the derivative term of the cost function with l2 regularization.

2.5 Model output:

Output of the model predicts the new corona cases based on the type of data that is explained in the data set section.

3: Model training detail:

The model is trained for “65” epochs for the world wide data and 200 ephocs for the USA data. We chosen the number of ephocs such that if we increase the ephocs then its loss increases further.

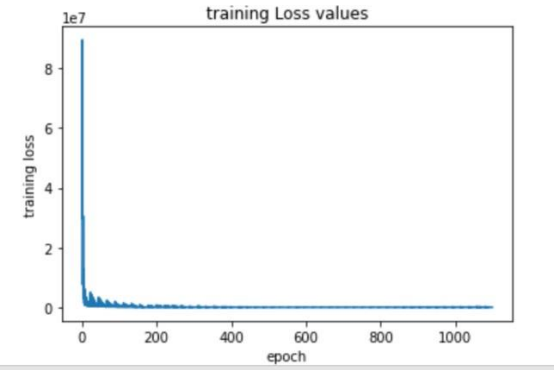
A batch size of 5000 is used for world wide prediction and a batch size of 32 is used for USA.

4: plots

The model is trained on 66967 number of samples . The plots for training, validation and testing losses are given below.

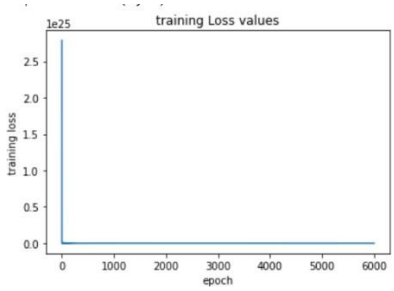
4.1 : plots for world wide prediction:

Training loss plot

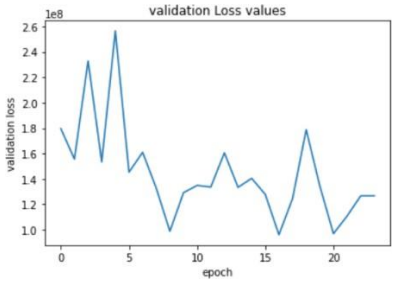


4.2 : plots for usa

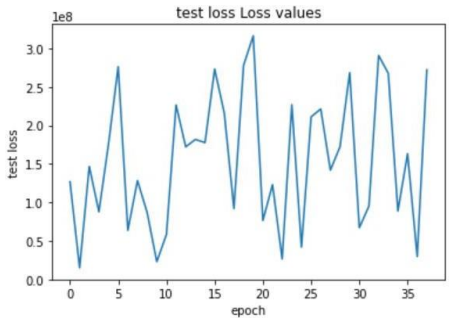
Training loss



Validation loss



Testing loss



Annex A

Instructions on running the code:

World wide

If you want to predict cases for world wide case in this project then open the “world wide” folder, copy the python code from here and paste it in google colab.

After running the code it will ask for inputs,

Then enter the ww_inputs file, after that it will ask for labels then enter the ww_labels file.

If you want to predict new cases for new day for a country then you need to give the below inputs in the given sequence,

“continent, total_cases, total_deaths, new_deaths, reproduction_rate, icu_patients, hosp_patients, weekly_icu_admissions, weekly_hosp_admissions, new_tests, total_tests, positive_rate, population, median_age, aged_65_older, aged_70_older, cardiovasc_death_rate”.

USA

If you want to predict cases for USA case in this project then open the “usa” folder, copy the python code from here and paste it in google colab.

After running the code it will ask for inputs,

Then enter the usa_inputs file, after that it will ask for labels then enter the usa_labels file.

If you want to predict new cases for new day for a country then you need to give the below inputs in the given sequence,

“continent, total_cases, total_deaths, new_deaths, reproduction_rate, icu_patients, hosp_patients, weekly_icu_admissions, weekly_hosp_admissions, new_tests, total_tests, positive_rate, stringency_index, population, median_age, aged_65_older, aged_70_older, cardiovasc_death_rate”, diabetus prevalence”.

Annex B

Optimal parameters:

I take the sizes of weight matrices so big whose pic is attached below and be pasted here, if you want to print it, then the code for it is given in the end of each python file in zip file.

```
shape of w1 is (20, 128)
shape of b1 is (1, 128)
shape of w2 is (64, 128)
shape of b2 is (1, 64)
shape of w3 is (1, 64)
shape of b3 is (1, 1)
```

Training function code:

```
#training_function
  def training(self,X, y, learning_rate = 0.01, batch_size = 32):
    self.error_list = []
    ephocs = 300
    l=len(X)
    for itr in range(ephocs):
      n = np.float(x.shape[0]) # total number of samples
      mini_batches = self.create_mini_batches(X, y, batch_size)
      for mini_batch in mini_batches:
        X_mini, y_mini = mini_batch


        lambd = 5
        m = np.size(X_mini)


        l1  =  np.dot(X_mini,self.w1) + self.b1                  #l1,l2,l3 are layer 1, 2 and 3 respectively
```

```
        a_l1 = self.relu(l1)

        l2 = np.dot(a_l1,self.w2.T) + self.b2

        a_l2 = self.relu(l2)

        l3 = np.dot(a_l2,self.w3.T) + self.b3

        slope1 = np.sum(np.dot(X_mini.T,(a_l1-y_mini))) + (lambd/m)*self.w1 # adding regularizatoin factor in weight 1

        intecept1=np.sum(a_l1-y_mini)

        slope2 = np.sum(np.dot(X_mini.T,(a_l2-y_mini))) + (lambd/m)*self.w2 # adding regularizatoin factor in weight 2

        intecept2=np.sum(a_l2-y_mini)

        slope3 = np.sum(np.dot(X_mini.T,(l3-y_mini))) + (lambd/m)*self.w3 # adding regularizatoin factor in weight 3

        intecept3 = np.sum(l3-y_mini)

        self.w1 = self.w1 - learning_rate * (slope1/l)

        self.b1 = self.b1 - learning_rate *(intecept1/l)

        self.w2 = self.w2 - learning_rate *(slope2/l)

        self.b2 = self.b2 - learning_rate *(intecept2/l)

        self.w3 = self.w3 - learning_rate *(slope3/l)

        self.b3 = self.b3 - learning_rate *(intecept3/l)


        #calculating the l2 regularization cost here

        L2_regularization_cost = (np.sum(np.square(self.w1)) + np.sum(np.square(self.w2)) + np.sum(np.square(self.w3)))*(lambd/(2*m))

        total_cost = self.cost(X_mini, y_mini)  +       L2_regularization_cost

        self.error_list.append(total_cost)
```

Annex C

```
  #prediction_function that consist total of three layers two hidden and one output layer

  def prediction(self,X):
```

```
l1 = np.dot(X,self.w1) + self.b1
a_l1 = self.relu(l1)
l2 = np.dot(a_l1,self.w2.T) + self.b2
a_l2 = self.relu(l2)
output = np.dot(a_l2,self.w3.T) + self.b3
return output
```