Tongtong Zhao

CPTG 323

06/08/2020

Final Project Report

A. Background and Purpose of the project

The outbreak of COVID-19 in 2020 will surely be recorded as one of the biggest events in the twenty-first century. When China was preparing to celebrate the Spring Festival, in its province Wuhan, which a new type of coronavirus that is highly contagious broke out and it was unscrupulously eroding human lungs and threatening innocent lives. The first known case of COVID-19 occurred on December 8, 2019, and the Wuhan Health Commission conducted its early notification on December 31, 2019. Twenty-seven cases of "viral pneumonia" have been found. The first death occurred on January 11, 2020. On January 12, 2020 , the World Health Organization officially named Wuhan's new coronavirus as 2019nCoV (2019-novel Corona Virus). Within a month, COVID-19 has spread throughout the country and even around the world along with people's travels; and closure and isolation have followed.

Eventually, the virus has arrived in the United States. On February 27, the first case of COVID-19 in the USA was reported, followed by a daily surge of positive cases and deaths. Due to different national conditions, the United States did not adopt the closure policy at the beginning. Then, two weeks later the WHO organization announced 2019nCoV as a global epidemic. At the same time, the COVID-19 test-positive exceeded 100,000 cases. The state governments ordered the closure of the city, the public places were closed, and the school turned to online teaching. At that time, there have been 30,000 cases. Anxiety and panic began to spread among the crowd, accompanied by a shortage of medical protective equipment. I believe that everyone doubted at that time: When will the epidemic end? What is the trend of the future epidemic situation? Solving these two problems is the attempt of this project.

B. Problem encountered & Background of Approach

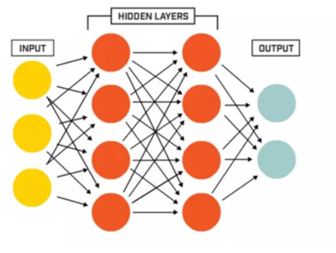
The entire project is divided into three parts: finding information about the main problem, confirming and programming the specific type of prediction model that will be used, and completing the analysis report. When searching for information, I found that predicting the end time of the outbreak proposed in the previous paragraph could not be completed in the model. The reasons are as follows: the end of the epidemic is closely related to the isolation policy, medical resources, economic turmoil, and human psychological activities. Therefore, based on my existing knowledge and available information that I can find online, I am not capable of establishing a prediction model that can calculate the above biases' weight in an accurate range. Thus, I concentrated on solving the second problem: predicting the trend of the future epidemic, or in other words, predicting the numbers of cases that COVID-19 test-positive and deaths.

I decided to use the Long Short Term Memory model to complete the establishment of the prediction model by looking up the techniques and the previous model established for the SARS virus trend analysis. Before introducing LSTM, let us first understand the artificial neural network and its branches. Artificial Neural Network, or ANN for short, is a way to implement machine learning tasks. It is a network structure composed of many simple elements. This network structure is similar to the biological nervous system and is used to simulate the interaction between living things and the natural environment. A neural network is a relatively broad concept. For different learning tasks such as speech, text, and images, a neural network model that is more suitable for specific learning tasks is derived, such as Recurrent Neural Network (RNN), convolutional neural Network (Convolutional Neural Network, CNN), etc. The

LSTM model I use is a particular type of RNN. The RNN is selected because each n+1 training of the model may be related to the previous training, for example, translation: there are n words in a sentence, and each word is a training instance. The semantic meaning of a word depends on its context; in our case, the context will be the previous or next training. The basic RNN model only processes the output of the last unit. In this way, the output of the unit is far away from the original node because of the multiple processing in the middle; the weight effect gradually disappears. This brings a Long term dependency problem. To retain useful key information as much as possible, LSTM appeared. The function and characteristics of LSTM are not hard to understand. Let's think, in this way, due to the limited memory capacity, and it learns to retain only relevant (or necessary) information to make the next step's predictions and forget irrelevant data to save the room.

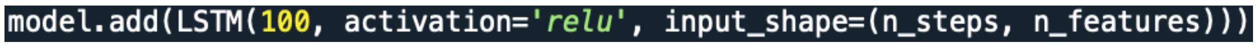
C. Approach in the Project

After understanding the basics of LSTM, I started to build predictive models. First of all, finding the right and complete data set is the top priority. On the Kaggle website, I found the ideal data set, which contains the data of the US states from the first case to today, and has a detailed classification of positive, death, negative, positive increase and other records. After downloading the data set, I decided to use Spyder in anaconda for model programming, because Spyder's library can reduce my burden compared to Pycharm. First, I imported various packages: Keras (deep learning framework), pandas (reading data sets), NumPy (different matrix data types and vector processing), matplotlib (generating forecast trend graphs), datetime (a standard library handles dates and times), sklearn (contains mainstream machine learning algorithms). Then I defined the Callback class based on python tf.callbacks source code to form a new abstract base class of callback function to store each loss to graph and wrote a LossHistory class to save the loss and accuracy of the training set (Source code line 48-129, will use “SC” to abbreviate



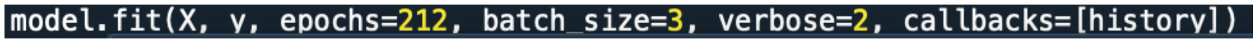
“Source code”). Both of these classes can refer to the reference link in the source code. Because LSTM is a model for time series forecasting, it is essential to establish Multilayer Perceptron (MLP) for fitting complex functions and to solve classification problems. In addition to the input layer and output layer, there can be multiple hidden layers in the middle of the MLP, as shown on the left. For example, if the input is an n-dimensional vector, there are n neurons in the input layer, and they are fully connected to hidden layers. Assuming that a vector X represents the input layer, the output of the hidden layer is f(W1X+b1), W1 is the weight (also called the connection coefficient), b1 is the offset, and the function (f) can be commonly used sigmoid function or tanh function. Therefore, all the parameters of MLP are the connection weight and offset between each layer. The framework of MLP can be found in an eBook called “Introduction to Time Series Forecasting With Python” by Nader Nazemi. After completed MLP, we can start to code the function that predicts the next seven days’ trend of test-positive and death cases. First of all, I defined the function and added the parameters that will be used in the prediction function to the parentheses; then I created an empty list of prediction data. Setting the number of days that need to be predicted, we also need to increase the number of days used to predict the trend (n_step). For instance, there are 100 data in the data set, and setting n_step=10 means to use the 10 days data as a group to predict the next seven days of data. After completing these presets, wrap a while loop used to append the prediction data into the list. In this loop, I used np.array to reshape the data format and model.predict to complete the data prediction and store the prediction data in the variable “yhat.” There is a trick since we want to predict the direction of the two datasets (test-positive and death), so we need to set the dimension of reshaping to 2. Finally, use “return fur_data” to get a list of added data (SC line 149-163). Following, complete the construction of the data visualization framework (SC line 166-207). Before cleaning the data set, it’s better to use Excel to sort the date, and then use pd.read_csv to read the file in Python; After that, use data.loc to set the data filtering conditions; in this case, we set the state to CA since we only predict California’s data. Then, use data.fillna to fill in the missing data with the fill value set to 0. Finally, set the start date of the predicted data and convert it into a list. Finally, use matplotlib for drawing.

The steps listed above are all in the framework of building a predictive model. We will process the data set and then put it into the frame for parameter adjustments and training (SC line 211-272). Like what we did for the last step, first, define the main function, read the data and fill in the missing values, filter the data (only retain the California data), reformat the positive and death data to get two columns of data, -1 represents row unknown. Use sklearn package’s MinMaxScalar normalizes the dataset, so it converges faster, and loss decreases more quickly. Among them, don’t forget to use fit_transform to fit the dataset first, find its overall indicators, such as mean, variance, maximum and minimum, etc. (according to the specific conversion purpose), and then transform the dataset to achieve data standardization, normalization. Now, set n_step to 4. Here is why I choose four days as a cycle; by observing JHU’s data statistics barplot, it can be determined that every four days is a growth period. On the fifth day, the data has declined due to the last working day or other reasons, which does not conform to the growing trend. Set n_feature to 2, which is the same as the dimension mentioned earlier. After reshaping data again, you can use Sequential model (it is a simplified version of the functional model, the simplest linear, from start to end structural sequence, without bifurcation, a linear stack of multiple network layers) and LSTM for prediction. 100 represents the number of Hidden Layers,



```
model.add(LSTM(100, activation='relu', input_shape=(n_steps, n_features)))
```

activation function is used to fit training data, set input to days in a cycle and data feature. Then set the loss value to RMSE (root mean square error, which is: if we have a linear model, the RMSE is the radius from every data point to the fit line). Next, define the number of iterations as 212 (train the model 212 times), number of batches (which contains several runs of



```
model.fit(X, y, epochs=212, batch_size=3, verbose=2, callbacks=[history])
```

computations) as 4, verbose determines whether to display each iteration, and callback is predefined. Use history.loss_plot('epoch') to draw and train the acc-loss curve, then we just need to use model.predict one more time to demonstrate the prediction. Finally, use the math package to complete the calculation of RMSE and then print all the predicted data in 2 columns.

D. Result analysis

Now, let’s run the code to analyze the prediction data and graphs. The first one is loss accuracy graph, the y-axis is the missing value of each training, and the x-axis is the number of training. It can be seen from the figure 1 that the efficiency played by the MinMaxScalar function: after several training, the loss value starts to decrease in a straight line and approaches 0. This trend has been maintained until the end of the training. This picture shows the accuracy of this prediction model in each training; the smaller the loss value the higher accuracy. According to the above analysis and the curve in the figure 1, we can conclude that the

Figure 1

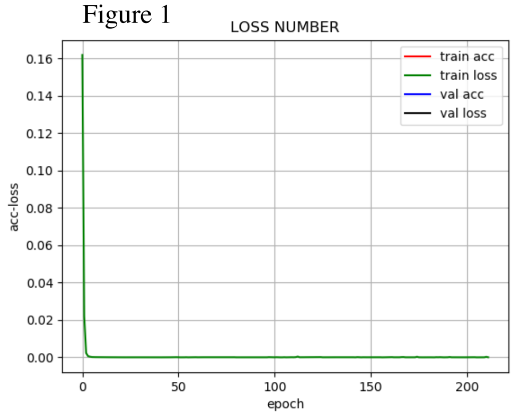
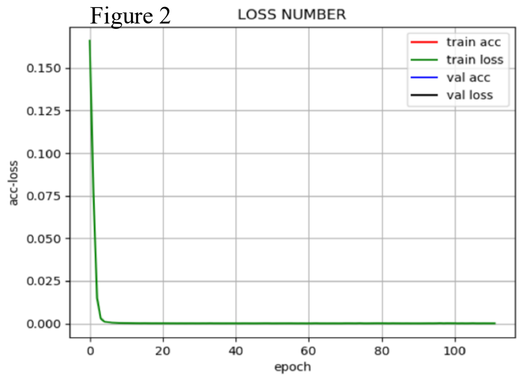
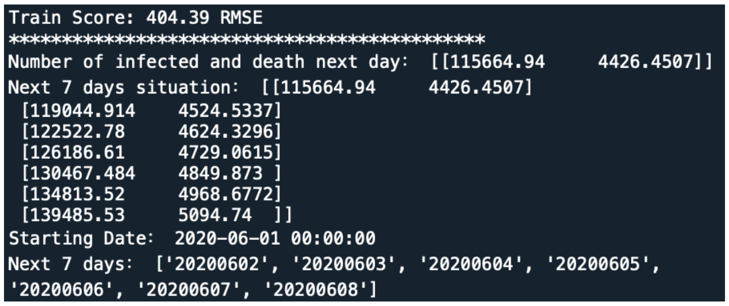


Figure 2

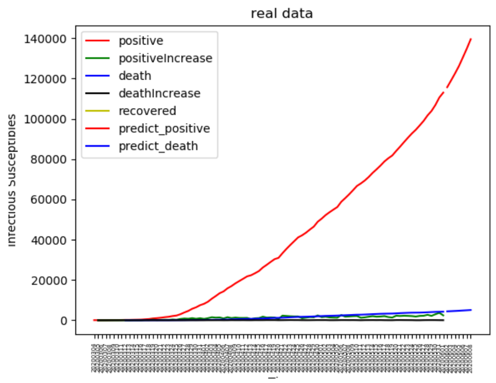


prediction model training is very successful, and the predicted data can be used as a comparison to analyze the trend of real data. Here is the accuracy vs. loss graph (figure 2) I generated a month ago, the dataset's time frame used there is from January to May. By comparing them, I realized that the slope in figure 1 is steeper than figure 2's which indicated figure 1 has a faster convergence to 0 or a better outcome of training. I think this is due to the larger dataset and the pattern I found in step size (4 days as a cycle; I set the step size to 3 in figure 2 training model). Another thing worth noticing is that as the dataset gets larger, the number of training needs to be increased to get a lower RMSE (smaller error). For the previous dataset, I asked the model to be training for 112 times, and I got an RMSE about 323. Then I used the same training times for the updated dataset, and I got 1030 for RMSE, which was way beyond my acceptable range. Thus,



```
Train Score: 404.39 RMSE
*********************************************
Number of infected and death next day:  [[115664.94     4426.4507]]
Next 7 days situation:  [[115664.94     4426.4507]
 [119044.914    4524.5337]
 [122522.78     4624.3296]
 [126186.61     4729.0615]
 [130467.484    4849.873 ]
 [134813.52     4968.6772]
 [139485.53     5094.74  ]]
Starting Date:  2020-06-01 00:00:00
Next 7 days:  ['20200602', '20200603', '20200604', '20200605',
'20200606', '20200607', '20200608']
```

except for changing the step size, I also increased the number of training to 212 and got a 404.39 RMSE. So far is the predicted data for the updated dataset; let's compare this with the real-time data. For Jun 2nd, the test positive cases are 117215, and death cases are 4305, the error percent ($\frac{Theoretical-Actual}{Theoretical} * 100\%$) for positive and death cases are 1.3% and 2.7%, which are higher than the numbers in my project midterm report (positive 0.17%, death 1.8%). For Jun 3rd, the test positive cases are 119348, and death cases are



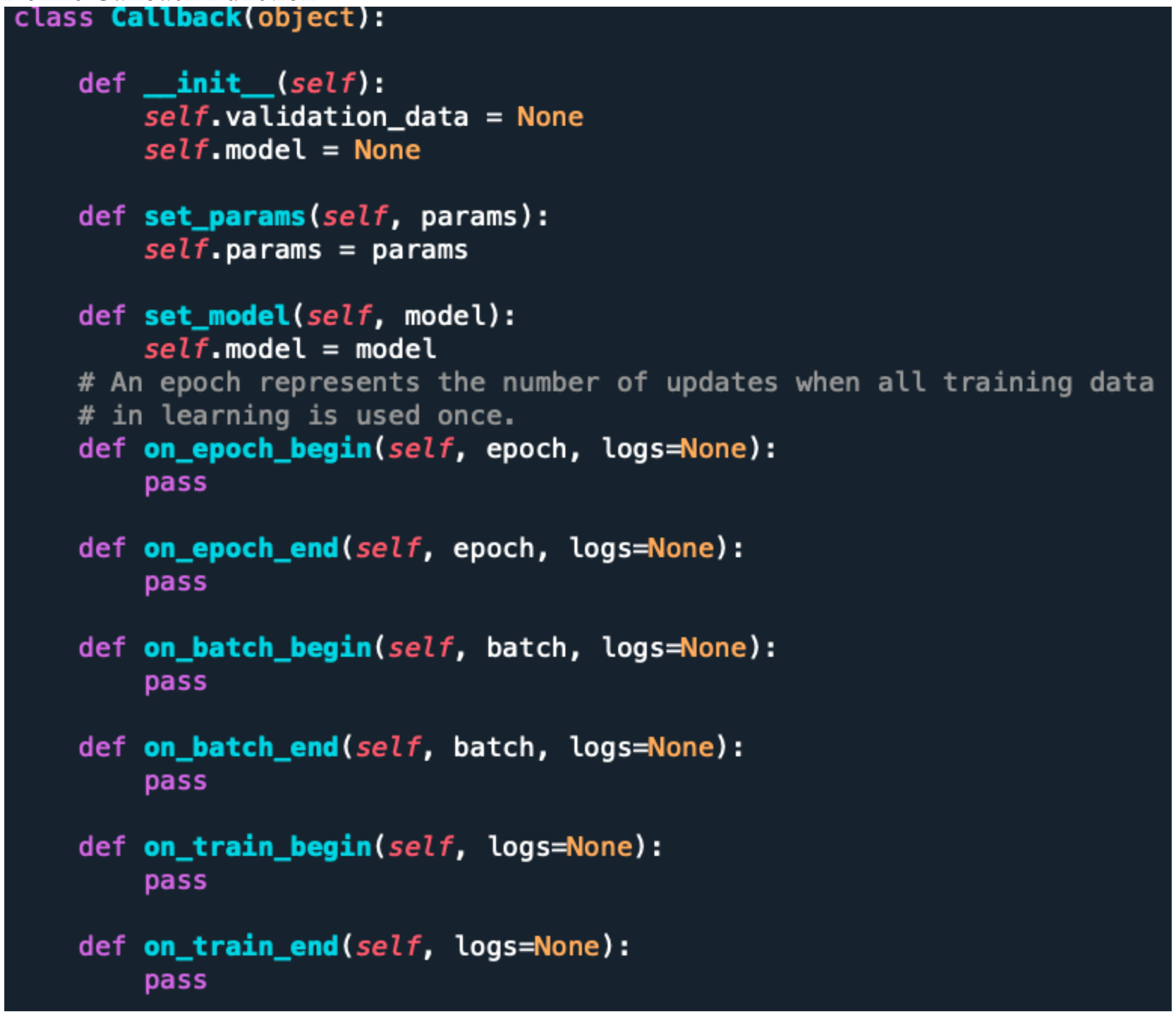
4424, the error percent for positive and death cases are 0.25% and 2.2%. Thus, my hypothesis for the error percent increment on Jun 2nd is: there are quite an amount of people protesting recently, and companies are trying to reopen, the social distance rule is not applicable during this time. Last, let’s look at the predicted trend graph. As the result of the prediction model is showing in the graph, we could reach the conclusion: the positive cases will continue to increase at a steeper rate, the death cases will increase by a slow rate since the slope for the blue line is nearly 0 (after the gap, the red line is positive cases’ prediction, and the blue line is for death cases).

E. Future changes to improve the model

According to the visual and numerical results above, if the dataset is updated to a larger volume, I will choose to predict three categories of data (positive, death, and positive increase) instead of 2. For large data sets, increasing dimension can increase the accuracy rate to a certain extent while reducing the fitting, but we must remove the garbage data first. Otherwise, we will commit a taboo of machine learning: garbage in garbage out. In addition, we can determine the step size by observing the data growth characteristics and adjust it. At the same time, changing the training times in time is also an extremely important factor in helping us get the ideal RMSE. It is worth mentioning that if the model is going to be used for forecasting the trend of the epidemic situation in other countries, the parameters that need to be changed will be the same as I mentioned above.

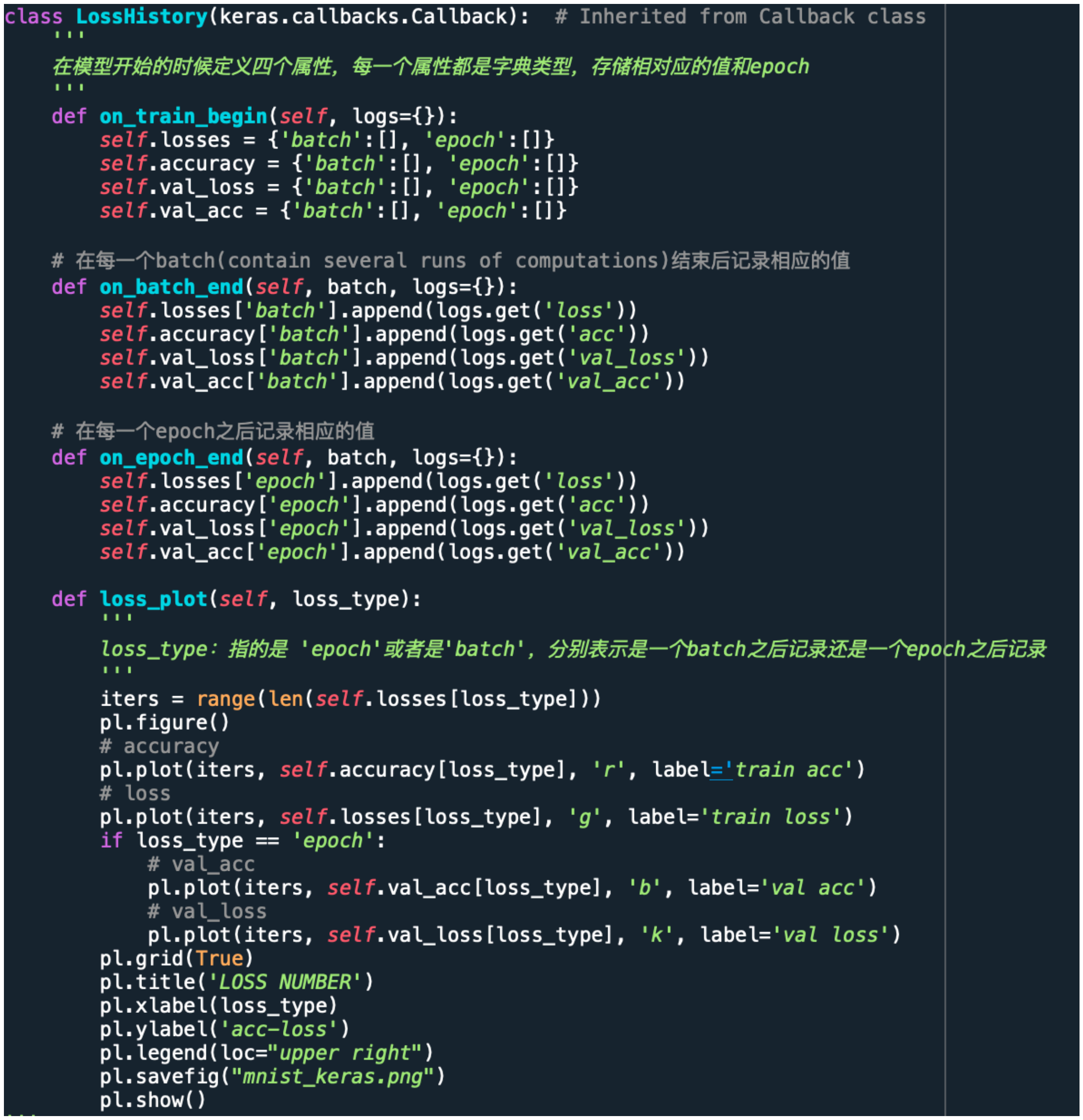
Source Code Process

Define Callback Function



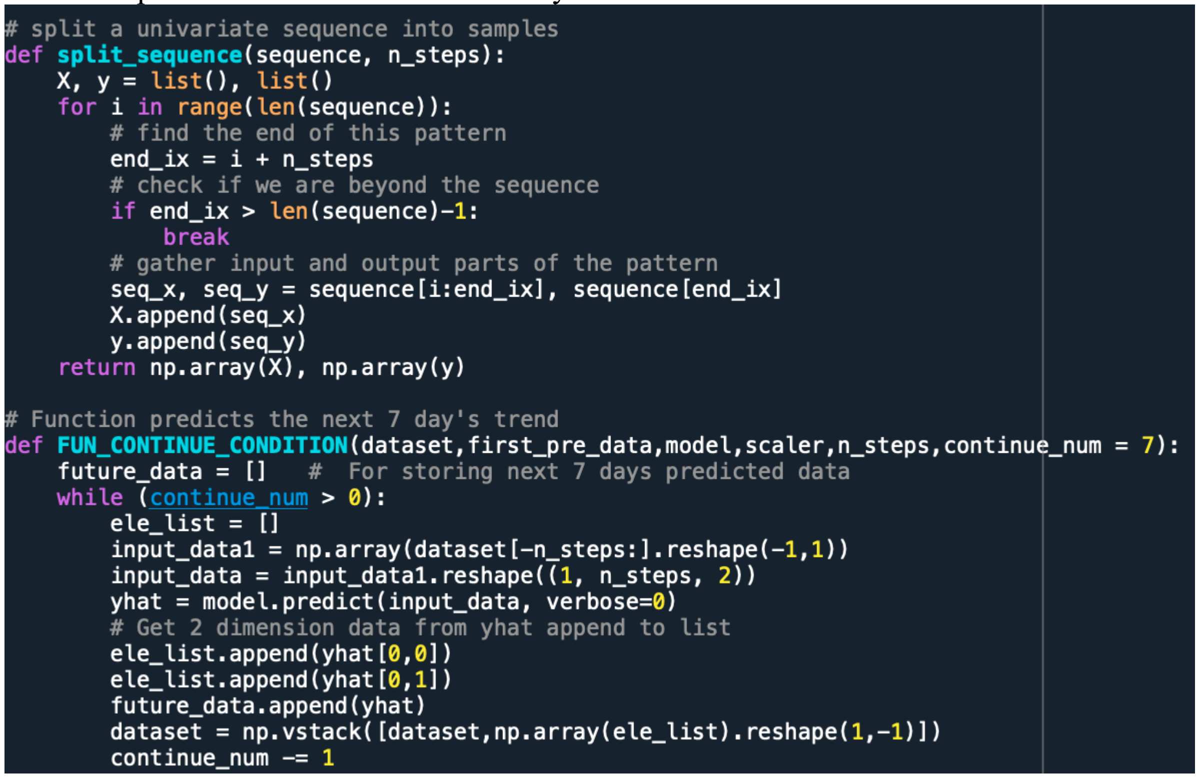
```python
class Callback(object):

    def __init__(self):
        self.validation_data = None
        self.model = None

    def set_params(self, params):
        self.params = params

    def set_model(self, model):
        self.model = model
    # An epoch represents the number of updates when all training data
    # in learning is used once.
    def on_epoch_begin(self, epoch, logs=None):
        pass

    def on_epoch_end(self, epoch, logs=None):
        pass

    def on_batch_begin(self, batch, logs=None):
        pass

    def on_batch_end(self, batch, logs=None):
        pass

    def on_train_begin(self, logs=None):
        pass

    def on_train_end(self, logs=None):
        pass
```

Define LossHistory Function to store the loss and accuracy of the training set



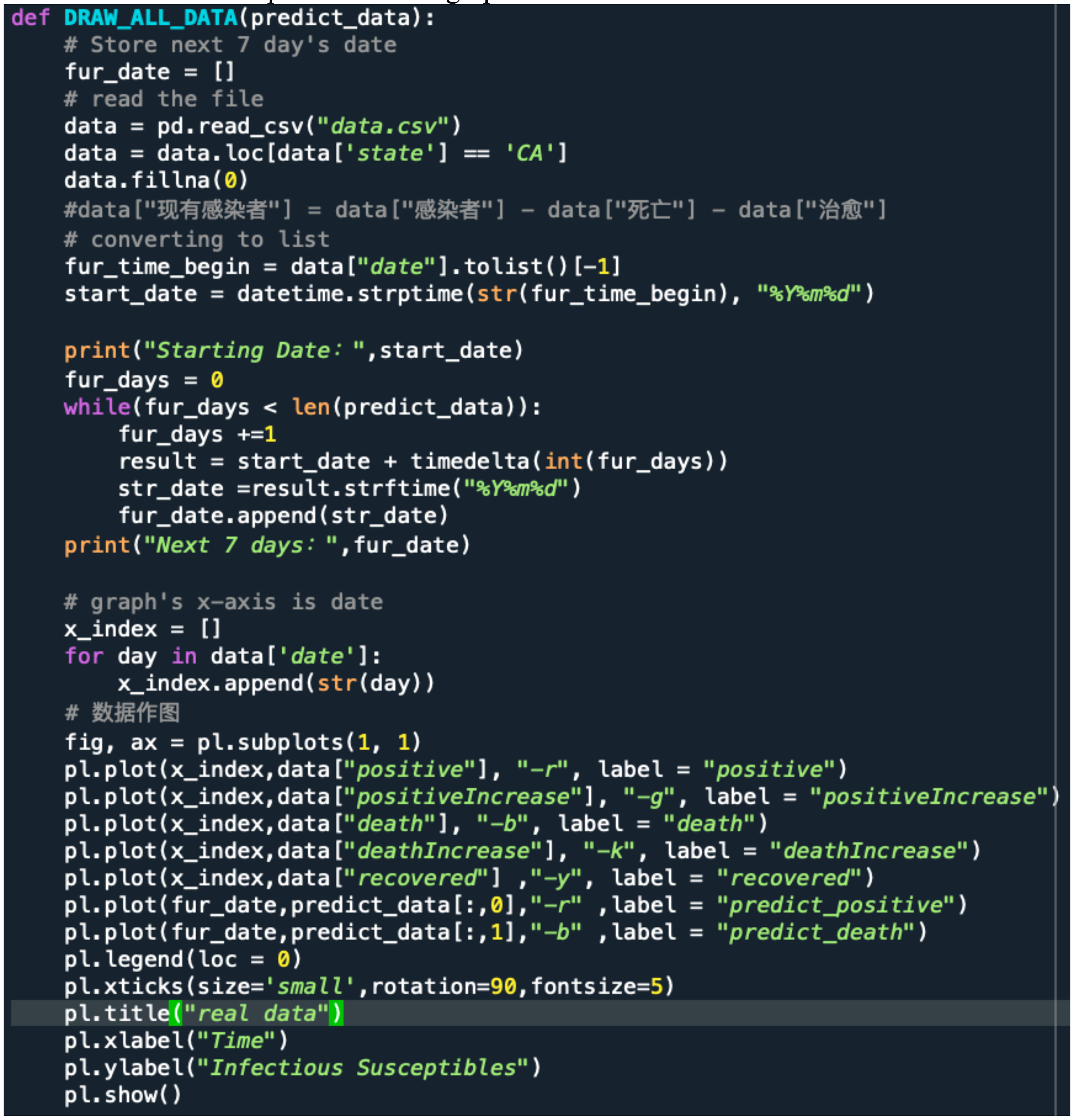
```python
class LossHistory(keras.callbacks.Callback):  # Inherited from Callback class
    '''
    在模型开始的时候定义四个属性，每一个属性都是字典类型，存储相对应的值和epoch
    '''
    def on_train_begin(self, logs={}):
        self.losses = {'batch':[], 'epoch':[]}
        self.accuracy = {'batch':[], 'epoch':[]}
        self.val_loss = {'batch':[], 'epoch':[]}
        self.val_acc = {'batch':[], 'epoch':[]}

    # 在每一个batch(contain several runs of computations)结束后记录相应的值
    def on_batch_end(self, batch, logs={}):
        self.losses['batch'].append(logs.get('loss'))
        self.accuracy['batch'].append(logs.get('acc'))
        self.val_loss['batch'].append(logs.get('val_loss'))
        self.val_acc['batch'].append(logs.get('val_acc'))

    # 在每一个epoch之后记录相应的值
    def on_epoch_end(self, batch, logs={}):
        self.losses['epoch'].append(logs.get('loss'))
        self.accuracy['epoch'].append(logs.get('acc'))
        self.val_loss['epoch'].append(logs.get('val_loss'))
        self.val_acc['epoch'].append(logs.get('val_acc'))

    def loss_plot(self, loss_type):
        '''
        loss_type：指的是 'epoch'或者是'batch'，分别表示是一个batch之后记录还是一个epoch之后记录
        '''
        iters = range(len(self.losses[loss_type]))
        pl.figure()
        # accuracy
        pl.plot(iters, self.accuracy[loss_type], 'r', label='train acc')
        # loss
        pl.plot(iters, self.losses[loss_type], 'g', label='train loss')
        if loss_type == 'epoch':
            # val_acc
            pl.plot(iters, self.val_acc[loss_type], 'b', label='val acc')
            # val_loss
            pl.plot(iters, self.val_loss[loss_type], 'k', label='val loss')
        pl.grid(True)
        pl.title('LOSS NUMBER')
        pl.xlabel(loss_type)
        pl.ylabel('acc-loss')
        pl.legend(loc="upper right")
        pl.savefig("mnist_keras.png")
        pl.show()
```

Define split_sequence function which divides the given univariate sequence into multiple samples, and each sample has a specified time step, and the output is a single time step. Then define the prediction function for next 7 days



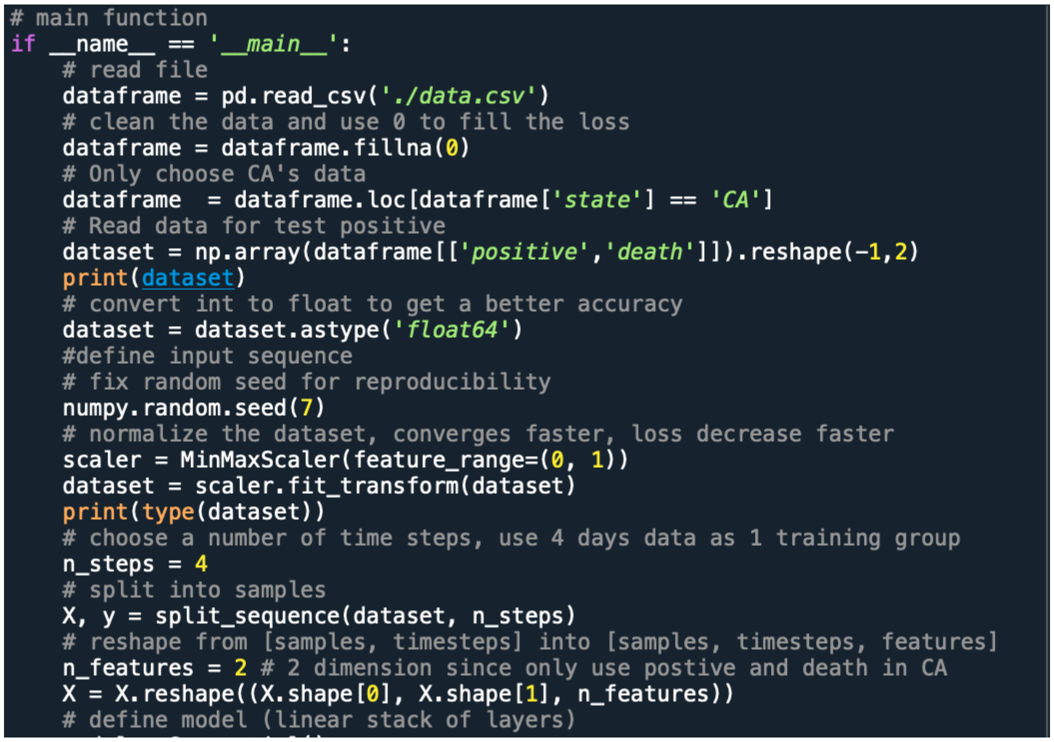
```
# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)

# Function predicts the next 7 day's trend
def FUN_CONTINUE_CONDITION(dataset,first_pre_data,model,scaler,n_steps,continue_num = 7):
    future_data = []   #  For storing next 7 days predicted data
    while (continue_num > 0):
        ele_list = []
        input_data1 = np.array(dataset[-n_steps:].reshape(-1,1))
        input_data = input_data1.reshape((1, n_steps, 2))
        yhat = model.predict(input_data, verbose=0)
        # Get 2 dimension data from yhat append to list
        ele_list.append(yhat[0,0])
        ele_list.append(yhat[0,1])
        future_data.append(yhat)
        dataset = np.vstack([dataset,np.array(ele_list).reshape(1,-1)])
        continue_num -= 1
```

Visualize the data and prediction into graph

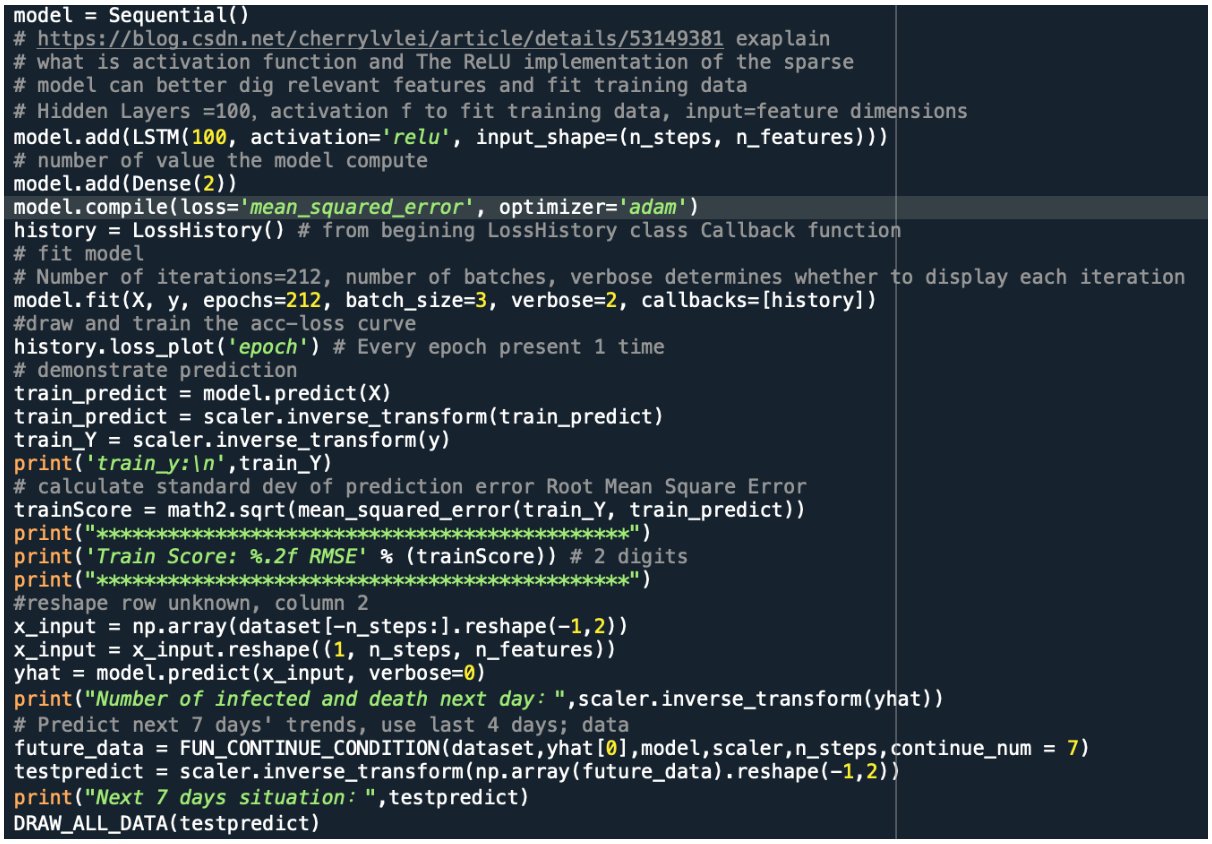


```python
def DRAW_ALL_DATA(predict_data):
    # Store next 7 day's date
    fur_date = []
    # read the file
    data = pd.read_csv("data.csv")
    data = data.loc[data['state'] == 'CA']
    data.fillna(0)
    #data["现有感染者"] = data["感染者"] - data["死亡"] - data["治愈"]
    # converting to list
    fur_time_begin = data["date"].tolist()[-1]
    start_date = datetime.strptime(str(fur_time_begin), "%Y%m%d")

    print("Starting Date: ",start_date)
    fur_days = 0
    while(fur_days < len(predict_data)):
        fur_days +=1
        result = start_date + timedelta(int(fur_days))
        str_date =result.strftime("%Y%m%d")
        fur_date.append(str_date)
    print("Next 7 days: ",fur_date)

    # graph's x-axis is date
    x_index = []
    for day in data['date']:
        x_index.append(str(day))
    # 数据作图
    fig, ax = pl.subplots(1, 1)
    pl.plot(x_index,data["positive"], "-r", label = "positive")
    pl.plot(x_index,data["positiveIncrease"], "-g", label = "positiveIncrease")
    pl.plot(x_index,data["death"], "-b", label = "death")
    pl.plot(x_index,data["deathIncrease"], "-k", label = "deathIncrease")
    pl.plot(x_index,data["recovered"] ,"-y", label = "recovered")
    pl.plot(fur_date,predict_data[:,0],"-r" ,label = "predict_positive")
    pl.plot(fur_date,predict_data[:,1],"-b" ,label = "predict_death")
    pl.legend(loc = 0)
    pl.xticks(size='small',rotation=90,fontsize=5)
    pl.title("real data")
    pl.xlabel("Time")
    pl.ylabel("Infectious Susceptibles")
    pl.show()
```

Define the main function and load the dataset to forecast and train the model



```
# main function
if __name__ == '__main__':
    # read file
    dataframe = pd.read_csv('./data.csv')
    # clean the data and use 0 to fill the loss
    dataframe = dataframe.fillna(0)
    # Only choose CA's data
    dataframe  = dataframe.loc[dataframe['state'] == 'CA']
    # Read data for test positive
    dataset = np.array(dataframe[['positive','death']]).reshape(-1,2)
    print(dataset)
    # convert int to float to get a better accuracy
    dataset = dataset.astype('float64')
    #define input sequence
    # fix random seed for reproducibility
    numpy.random.seed(7)
    # normalize the dataset, converges faster, loss decrease faster
    scaler = MinMaxScaler(feature_range=(0, 1))
    dataset = scaler.fit_transform(dataset)
    print(type(dataset))
    # choose a number of time steps, use 4 days data as 1 training group
    n_steps = 4
    # split into samples
    X, y = split_sequence(dataset, n_steps)
    # reshape from [samples, timesteps] into [samples, timesteps, features]
    n_features = 2 # 2 dimension since only use postive and death in CA
    X = X.reshape((X.shape[0], X.shape[1], n_features))
    # define model (linear stack of layers)
```



```
model = Sequential()
# https://blog.csdn.net/cherrylvlei/article/details/53149381 exaplain
# what is activation function and The ReLU implementation of the sparse
# model can better dig relevant features and fit training data
# Hidden Layers =100, activation f to fit training data, input=feature dimensions
model.add(LSTM(100, activation='relu', input_shape=(n_steps, n_features)))
# number of value the model compute
model.add(Dense(2))
model.compile(loss='mean_squared_error', optimizer='adam')
history = LossHistory() # from begining LossHistory class Callback function
# fit model
# Number of iterations=212, number of batches, verbose determines whether to display each iteration
model.fit(X, y, epochs=212, batch_size=3, verbose=2, callbacks=[history])
#draw and train the acc-loss curve
history.loss_plot('epoch') # Every epoch present 1 time
# demonstrate prediction
train_predict = model.predict(X)
train_predict = scaler.inverse_transform(train_predict)
train_Y = scaler.inverse_transform(y)
print('train_y:\n',train_Y)
# calculate standard dev of prediction error Root Mean Square Error
trainScore = math2.sqrt(mean_squared_error(train_Y, train_predict))
print("**********************************************")
print('Train Score: %.2f RMSE' % (trainScore)) # 2 digits
print("**********************************************")
#reshape row unknown, column 2
x_input = np.array(dataset[-n_steps:].reshape(-1,2))
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print("Number of infected and death next day: ",scaler.inverse_transform(yhat))
# Predict next 7 days' trends, use last 4 days; data
future_data = FUN_CONTINUE_CONDITION(dataset,yhat[0],model,scaler,n_steps,continue_num = 7)
testpredict = scaler.inverse_transform(np.array(future_data).reshape(-1,2))
print("Next 7 days situation: ",testpredict)
DRAW_ALL_DATA(testpredict)
```

Reference

Bioinformatician, Nader Nazemi, et al. “Time Series Forecasting With Python.” *Machine Learning Mastery*, 16 Jan. 2020, machinelearningmastery.com/introduction-to-time series-forecasting-with-python/.

Dennybritz. “Dennybritz/Nn-from-Scratch.” *GitHub*, 19 Oct. 2017, github.com/dennybritz/nn from-scratch.

“计算机的潜意识.” *神经网络浅讲：从神经元到深度学习 - 计算机的潜意识 - 博客园*, www.cnblogs.com/subconscious/p/5058741.html#second.

计算机应用技术博士 陈钢.css-1cd9gw4{margin-left:.3em;}.css-1pc1mic{box-sizing:border box;margin:0;min-width:0;color:#175199;display:inline block;width:18px;height:18px;margin-left:.3em;}.css-18biwo{display:-webkit-inline box;display:-webkit-inline-flex;display:-ms-inline-flexbox;display:inline-flex;-webkit align-items:center;-webkit-box-align:center;-ms-flex-align:center;align-items:center;} .css-1ifz0go{overflow:visible!important;}中南大学. “用 Python 从头实现一个神经网络.” *知乎专栏*, zhuanlan.zhihu.com/p/58964140.

“完整教程---Python-手写-BP-实现神经网络.” *完整教程---Python-手写-BP-实现神经网 _CoderPai 的博客-CSDN 博客_手写 bp 神经网络*, blog.csdn.net/CoderPai/article/details/80317064.