LSTM Time Series Predictor for COVID-19

Sophie Zheng (sz374)

github: https://github.com/sophiezheng0711/LSTM-Covid-Predictor

Section 1: Introduction

COVID-19 has been arguably the biggest life-changer for our generation. It has made 2020 void of vacation, reunions, parties, gatherings, etc. Visiting the John Hopkins COVID website has become a daily routine, making us wonder: when will this disaster end?

Using existing data on COVID-19 in the United States (From January 2020 to November 2020), I decided to use machine learning to predict the number of cases of COVID-19 in the US. I am using an LSTM (Long Short-Term Memory) RNN (Recurrent Neural Network) architecture. LSTMs are generally known for predicting the stock market prices and completing the given sentence, and are widely used in fields such as NLP(Natural Language Processing). Due to the limited knowledge of the virus the current world possesses, no known model is able to accurately predict the number of cases of COVID-19. Therefore, this project only aims to predict a fairly accurate trend. I am using Python and Pytorch, which is a commonly-used module for Deep Learning.

Section 2: Background

In this section, I will introduce the background of Deep Neural Networks (DNNs) and LSTMs related to this project.

2.1: DNN

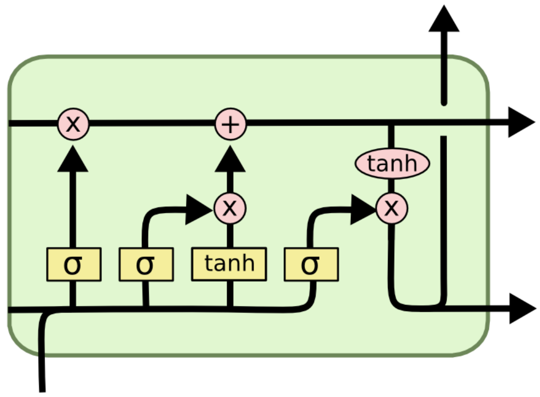
DNNs, or Deep Neural Networks, is a neural network with multiple layers between the input and output. The intuition behind this is when data is not clearly linearly separable (or, either “black” or “white”), we take the data to higher dimensions by analyzing more features to attempt to separate the data. Like this intuition, DNNs are commonly used to train classifiers. A common algorithm used in the training of these networks is backpropagation, which computes the gradient of the loss function (in other words, the penalty for the model) and efficiently updates the weights of the model. There are many choices for the loss function. In our case, we chose the Mean Squared Error (MSE) loss, which is defined as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Where Y is the true value of the sample, and $\hat{Y}$ is the predicted value by the model. This is a common method to evaluate estimated parameters, very similar to the MLE methods we learned in class. This also looks very similar to the sample variance formula. In addition, we need to choose an optimizer for the model. There are many of these, and I will not go into detail in this paper. Common ones are Stochastic Gradient Descent, Adam, and Adagrad. We will use Adam (Adaptive Moment Estimation) for our model. This optimizer is known for its adaptive trait and its minimal needs for hyperparameter optimization.

2.2: LSTM

Long Short-term Memory (LSTM) is a Recurrent Neural Network (RNN, a type of DNN) architecture. It can not only process single data points, but also sequences of data. A common LSTM is composed of an input gate, an output gate, and a forget gate. In Pytorch terminology, that is an input layer, lstm layers, dropout layers, and an output layer.



LSTMs are more focused on the order of data than normal RNNs. Thus, they excel on jobs such as text sentiment analysis, and time series predictions--which is why we chose LSTM to predict COVID-19.

Section 3: Data and Preprocessing

We will be using the time series COVID-19 US dataset provided by spdin (see [data_source] in Appendix). The data roughly looks like the image below:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	UID	iso2	iso3	code3	FIPS	Admin2	Province_Sta	Country_Reg	Lat	Long_	Combined_K	1/22/20	1/23/20	1/24/20	1/25/2
2	84001001	US	USA	840	1001	Autauga	Alabama	US	32.5395275	-86.644082	Autauga, Ala	0	0	0	
3	84001003	US	USA	840	1003	Baldwin	Alabama	US	30.7277499	-87.722071	Baldwin, Ala	0	0	0	
4	84001005	US	USA	840	1005	Barbour	Alabama	US	31.868263	-85.387129	Barbour, Ala	0	0	0	
5	84001007	US	USA	840	1007	Bibb	Alabama	US	32.9964206	-87.125115	Bibb, Alabam	0	0	0	
6	84001009	US	USA	840	1009	Blount	Alabama	US	33.9821092	-86.567906	Blount, Alab	0	0	0	
7	84001011	US	USA	840	1011	Bullock	Alabama	US	32.1003053	-85.712655	Bullock, Alab	0	0	0	
8	84001013	US	USA	840	1013	Butler	Alabama	US	31.753001	-86.680575	Butler, Alaba	0	0	0	
9	84001015	US	USA	840	1015	Calhoun	Alabama	US	33.7748373	-85.826304	Calhoun, Ala	0	0	0	
10	84001017	US	USA	840	1017	Chambers	Alabama	US	32.9136008	-85.390727	Chambers, A	0	0	0	
11	84001019	US	USA	840	1019	Cherokee	Alabama	US	34.1780598	-85.60639	Cherokee, Al	0	0	0	
12	84001021	US	USA	840	1021	Chilton	Alabama	US	32.8504413	-86.717326	Chilton, Alab	0	0	0	
13	84001023	US	USA	840	1023	Choctaw	Alabama	US	32.0222734	-88.265644	Choctaw, Ala	0	0	0	
14	84001025	US	USA	840	1025	Clarke	Alabama	US	31.6809986	-87.835486	Clarke, Alaba	0	0	0	
15	84001027	US	USA	840	1027	Clay	Alabama	US	33.2698419	-85.858361	Clay, Alabam	0	0	0	
16	84001029	US	USA	840	1029	Cleburne	Alabama	US	33.676792	-85.520059	Cleburne, Al	0	0	0	
17	84001031	US	USA	840	1031	Coffee	Alabama	US	31.3993283	-85.98901	Coffee, Alab	0	0	0	
18	84001033	US	USA	840	1033	Colbert	Alabama	US	34.6984745	-87.801685	Colbert, Alab	0	0	0	
19	84001035	US	USA	840	1035	Conecuh	Alabama	US	31.434017	-86.9932	Conecuh, Al	0	0	0	
20	84001037	US	USA	840	1037	Coosa	Alabama	US	32.9369015	-86.248477	Coosa, Alaba	0	0	0	
21	84001039	US	USA	840	1039	Covington	Alabama	US	31.2477854	-86.450509	Covington, A	0	0	0	
22	84001041	US	USA	840	1041	Crenshaw	Alabama	US	31.729418	-86.315931	Crenshaw, Al	0	0	0	
23	84001043	US	USA	840	1043	Cullman	Alabama	US	34.130203	-86.86888	Cullman, Ala	0	0	0	
24	84001045	US	USA	840	1045	Dale	Alabama	US	31.4303712	-85.610957	Dale, Alabam	0	0	0	
25	84001047	US	USA	840	1047	Dallas	Alabama	US	32.326881	-87.108667	Dallas, Alaba	0	0	0	
26	84001049	US	USA	840	1049	DeKalb	Alabama	US	34.4594686	-85.807829	DeKalb, Alab	0	0	0	
27	84001051	US	USA	840	1051	Elmore	Alabama	US	32.5978541	-86.144153	Elmore, Alab	0	0	0	
28	84001053	US	USA	840	1053	Escambia	Alabama	US	31.1256789	-87.159187	Escambia, Al	0	0	0	

This dataset is updated until 11/23/2020. As stated in Section 1, we are only interested in the total confirmed cases in the US daily, we do not need columns 1 to 11. We also do not need the division between states and counties, as we are only interested in the total amount daily. Keeping this in mind, we begin preprocessing the data.

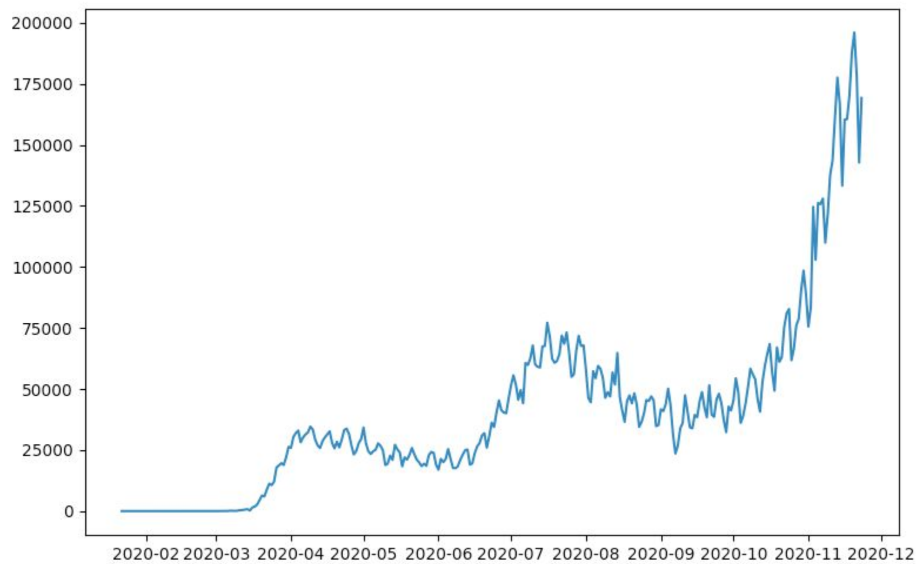
```
df = pd.read_csv("./data/time_series_covid19_confirmed_US.csv")
df = df.iloc[:, 11:]
df.isnull().sum().sum()
daily_cases = df.sum(axis=0)
daily_cases.index = pd.to_datetime(daily_cases.index)
daily_cases = daily_cases.diff().fillna(daily_cases[0]).astype(np.int64)
daily_cases.head()
```

As shown above, we first read in the data, then we delete the unnecessary columns. Then we sum across all rows, and we reformat the data. The data is also given in cumulative sums across the days, and we only want the daily increase. In addition, we need to divide the dataset into a training set and a test set, where the training set is "learning material" for our LSTM model, and the test set is our evaluation criteria. In this model, I set the test data size to 60, and the rest will be training data. In addition, given the "short-term" nature of LSTM models, we would like to further divide the training and testing data into smaller "windows". We do so by the following code:

```
def preprocess(raw, lookback):
   xs = []
   ys = []
   for i in range(len(raw) - lookback - 1):
       x = raw[i:(i+lookback)]
       y = raw[i+lookback]
       xs.append(x)
       ys.append(y)
   return np.array(xs), np.array(ys)
test_data_size = 60
lookback = 4
train_data = daily_cases[:-test_data_size]
test_data = daily_cases[-test_data_size:]
scaler = MinMaxScaler()
scaler = scaler.fit(np.expand_dims(train_data, axis=1))
```

```
train_data = scaler.transform(np.expand_dims(train_data, axis=1))
test_data = scaler.transform(np.expand_dims(test_data, axis=1))
```

After preprocessing, the data looks like this:



Here, we also need to normalize the data so that it ranges from 0 to 1. This is to aid any use of activation functions like ReLU and tanh. Here, we use the MinMaxScaler function provided by scipy, a commonly used statistical analysis tool for Python. Eventually, after training, we will have to revert this operation so that real data values are displayed instead of normalized ones.

Section 4: Model Construction and Training

This LSTM model architecture is inspired by [model_inspiration] in Appendix. It is a simple LSTM model consisting of 2 LSTM layers and 1 dense layer (or linear layer). Each LSTM consists of 32 hidden nodes. The input to the model is 1, as there is 1 number per day; the output of the model is 1, for the same reason. Thus, we construct our model with Pytorch as follows:

```
class CovidPredictor(torch.nn.Module):
   def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
```

```
        super(CovidPredictor, self).__init__()
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers
        self.lstm = torch.nn.LSTM(input_dim, hidden_dim, num_layers, dropout=0.2,
batch_first=True)
        self.dense = torch.nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad_()
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad_()

        out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))
        out = self.dense(out[:, -1, :])
        return out

model = CovidPredictor(input_dim=input_dim, hidden_dim=hidden_dim, num_layers=num_layers,
output_dim=output_dim)
```

In addition, we need to define a loss function and an optimizer for this model. In our case, we chose the MSE function as the loss function, and Adam as the optimizer (with learning rate 0.01). Both are commonly used in Deep Learning.

```
loss_fn = torch.nn.MSELoss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

This concludes the Model Construction of our model, and it's time for the training!

For training, we first allow our model to make the prediction. Then, we have to penalize/reward it for its prediction given the true answer. We then return this result to the model, and allow it to update the weights using backpropagation (described in Section 2). Pytorch makes this process extremely easy:

```
for epoch in range(epochs):
    output = model(x_train)
    training_pred = output.detach().numpy()
    loss = loss_fn(output, y_train)
    if epoch % 10 == 0 and epoch !=0:
        print("Epoch ", epoch, "MSE: ", loss.item())
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

And this is it! Our model has been trained. It is ready to be tested!

Section 5: Test Results

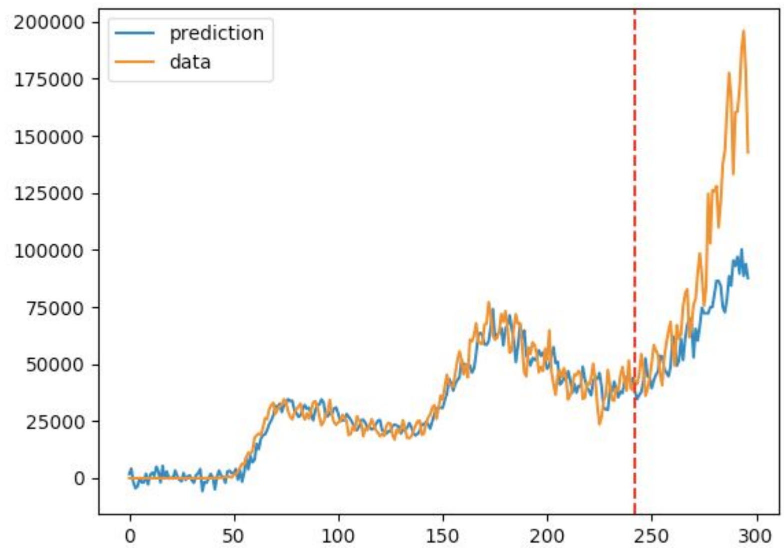
As mentioned in Section 3, we need to reverse the normalization before we plot the predictions and analyze the results:

```
test_pred = scaler.inverse_transform(model(x_test).detach().numpy())
test_plot = scaler.inverse_transform(y_test.detach().numpy())
train_pred = scaler.inverse_transform(training_pred)
train_plot = scaler.inverse_transform(y_train.detach().numpy())
```

I decided to concatenate the training predictions and the test predictions to compare the model performance. After doing so, we use matplotlib, a common Python graph maker, to create the graphs:

```
plt.plot(np.concatenate((train_pred, test_pred)), label="prediction")
plt.plot(np.concatenate((train_plot, test_plot)), label="data")
plt.axvline(x=len(x_train), c='r', linestyle='--')
plt.legend()
plt.show()
```

After executing the code, we see that our model has predicted a fairly accurate trend of the number of cases of COVID-19. In the graph below, the blue line represents the prediction of the model, while the yellow line is the real data. The red dotted line divides the training data (left) and the test data (right). As we can see, our model is slightly overfit to the training data, since it's very accurate during the training phase, but not as much during the testing phase. This could be due to the small amount of data we have (COVID has been around for only a year), and the simple structure of our model.



Section 6: Conclusion

Overall, our LSTM model successfully completed the task to approximate the trend of COVID-19. It is a shame that the number of cases in the US seems to be steadily increasing, but it is delightful to see our model's stunning performance in Section 5. LSTMs are known for good performance in time series predictions, like the stock market prices, airline passengers (see [LSTM_inspiration] in Appendix), and now, COVID-19 predictions. Although not perfect, our model shows great potential for machine learning in the study of COVID-19, and the importance of statistical analysis in real-world problems.

Section 7: Appendix

[LSTM_inspiration]:

https://github.com/spdin/time-series-prediction-lstm-pytorch/blob/master/Time_Series_Predi

ction_with_LSTM_Using_PyTorch.ipynb

[data_source]:

https://github.com/CSSEGISandData/COVID-19/blob/master/csse_covid_19_data/csse_covid_1

9_time_series/time_series_covid19_confirmed_US.csv

[model_inspiration]:

https://www.kaggle.com/taronzakaryan/stock-prediction-lstm-using-pytorch

[lstm_explanation]:https://en.wikipedia.org/wiki/Long_short-term_memory