CS 245 - Final Project Report

Team - Big Data Group

(Jason Kao, Sripath Mishra, Vishnu Devarakonda, Boya Ouyang)

University of California, Los Angeles

1. Introduction

In November of 2019, a new extremely infectious new disease, COVID-19, began to ravage the world [1]. The key characteristic that makes COVID extremely dangerous is its long incubation period [2]. With this in mind, it is of utmost importance to be able to accurately forecast the spread of the virus and its benefits could have compounding effects. First of all, the healthcare industry, with the knowledge of potential spikes in new cases, would be able to prepare to adequately handle the influx of sick patients and not be overwhelmed. This will allow hospitals to manage to staff accordingly; being able to handle the influx not only helps the COVID patients, other patients hospitalized will be able to receive treatment in a timely manner as well. Second, policymakers will be able to make decisions such as new lockdown measures more effectively with the foresight of case spikes. Last but not least, the general population would be able to have confidence in those in charge knowing that they have all the tools required at their disposal. This could potentially cull civil unrest and restore confidence in the administration's ability to handle a pandemic.

1.1 Project Goals

Our objective is to design a dynamic graph neural network which forecasts the Covid-19 pandemic spread inside the United States. We will approach this problem by building a graph G = (V, E) where the set V contains nodes representing places (e.g. cities, counties, states) while the set E contains edges representing relationships between the nodes. These relationships are static or dynamic. Static edges could be defined as an edge between a city and a county that the city is part of or two counties that share borders. Dynamic edges could be defined by time-dependent data like human mobility between two states. The space and time-dependent nature of a pandemic require us to capture the temporal and spatial features that contribute to the spread. We need to come up with a model that can reconcile these spatial and temporal features to forecast the count of infections at each granularity level (cities, counties, states). Dynamic graph neural networks are explored as a possible approach to make these forecasts.

1.2 Problem Statement

The biggest issue of the current forecasting methods is that they do not sufficiently utilize all of the available data. For instance, regression models, while powerful in their own right, do not make use of the geographical data. In the same vein, graph neural networks and their variants do not take the evolving nature of pandemic data into consideration as they only treat the graphs statically. In this project, we attempt to combine the temporal features of COVID data as well as mobility data of people into account to better forecast the spread of the virus.

2. Data Source and Preparation

During the paper surveys, we had identified multiple sources of data from the beginning. These sources could be categorized into the following sets: Dataset of states that share borders, SafeGraph mobility data which has information about how people are moving, CDC pandemic dataset which tracks the infection counts at different granularity, and County level weather condition data sets. We focused on USA COVID cases only. We wanted to start with the state-level data. In this, we wanted to create a graph that would represent states as the nodes and the edge between the nodes as the interactions between two states (Number of people going from one state to another). We started with the state-wise level of granularity as they do not have a lot of nodes as compared to counties or cities. We have implemented our code independent of the place granularity level. We started with the CDC data source.

For the structure of the data with a time component, a sliding window will be used to create individual timesteps and their corresponding label following the scheme below [3].

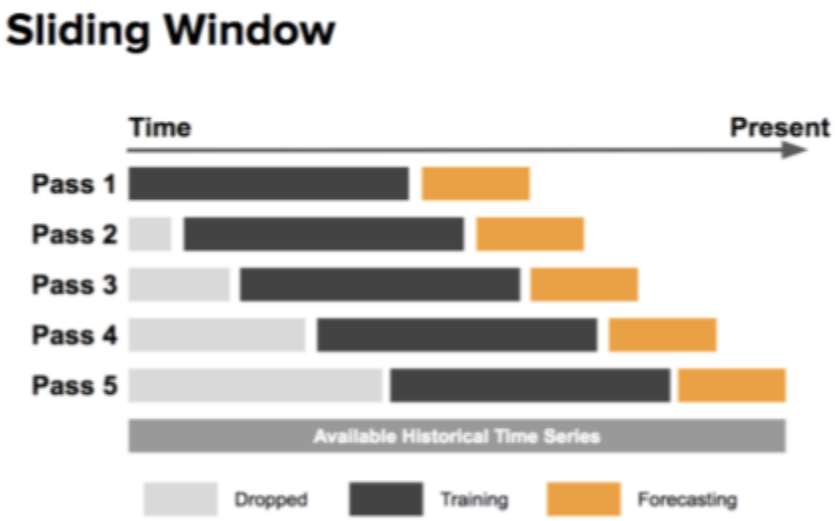

Figure 1: Window sliding through the data

The benefits of partitioning are two-fold. This scheme allows for a truth label that contains the same number of timesteps which is also a requirement imposed by Tensorflow/Keras. In addition to the ease of usage, the idea is that the data on the last day within a window should be more closely correlated to the previous days in that same window. For instance, during Thanksgiving, the number of new cases shot up. Figure 2 is the data from google showing the jump in new coronavirus cases.

These data points would not be indicative of the overall trend of the virus but it would be for that particular short time period.

CDC Data source: Centers for Disease Control and Prevention is a government entity that is a major collector of COVID data. They have two major data sets that could be useful to extract important information that can be used as input features for the nodes. They have the State-wise daily Covid data and the daily deidentified cases data. The state-wise dataset has 19.5 thousand rows of data and can be found at https://data.cdc.gov/Case-Surveillance/United-States-COVID-19-Cases-and-Deaths-by-State-o/9mfq-cb36 [4]. The state-wise data has the following columns: submission_date (Date of counts), state (Jurisdiction), tot_cases (Total number of cases), conf_cases (Total confirmed cases), prob_cases (Total probable cases), new_case (Number of new cases), pnew_case (Number of new probable cases), tot_death

Number of cases on 11/26

Number of cases on 11/27

Figure 2: Number of cases before and on Thanksgiving day

(Total number of deaths), conf_death (Total number of confirmed deaths), prob_death (Total number of probable deaths), new_death (Number of new deaths), pnew_death (Number of new probable deaths), created_at (Date and time record was created), consent_cases (If Agree, then confirmed and probable cases are included. If Not Agree, then only total cases are included), consent_deaths (If Agree, then confirmed and probable deaths are included. If Not Agree, then only total deaths are included.). A snapshot of the data is shown below in table 1:

This dataset is very valuable as it tells us the number of positive COVID cases and deaths. It has a high frequency of daily which gives us a lot of information and does not need us to create a distribution to divide weekly or monthly data. We took this dataset from January 20, 2020 (starting date of this dataset) till November 10, 2020. These dates gave us 17,641 rows of data to handle. We removed all the probate columns of data they are probable and no error

correction range was provided. This rule removed the prob_cases, pnew_case, prob_death, and pnew_death.

submission_date	state	tot_cases	conf_cases	prob_cases	new_case	pnew_case	tot_death	conf_death	prob_death	new_death	pnew_death	created_at	consent_cases	consen
09/24/2020	VA	144433	137283	7150	941	106	3136	2930	206	23	0	09/25/2020 01:5	Agree	Agree
09/25/2020	VA	145408	138173	7235	975	85	3144	2937	207	8	1	09/26/2020 01:4	Agree	Agree
09/26/2020	VA	146144	138734	7410	736	175	3159	2951	208	15	1	09/27/2020 01:3	Agree	Agree
09/27/2020	VA	146593	139144	7449	449	39	3172	2962	210	13	2	09/28/2020 01:5	Agree	Agree
09/28/2020	VA	147516	139961	7555	923	106	3187	2976	211	15	1	09/29/2020 01:5	Agree	Agree
09/29/2020	VA	148272	140615	7657	756	102	3208	2995	213	21	2	09/30/2020 02:1	Agree	Agree
09/30/2020	VA	148721	140990	7731	449	74	3228	3015	213	20	0	10/01/2020 01:4	Agree	Agree
10/01/2020	VA	149687	141850	7837	966	106	3250	3037	213	22	0	10/02/2020 01:4	Agree	Agree
10/02/2020	VA	150803	142923	7880	1116	43	3270	3057	213	20	0	10/03/2020 01:3	Agree	Agree

Table 1: Snapshot of the data

Then we also removed the consent columns (consent_cases and consent_deaths) as they do not provide any information regarding the infection spread in a state or between two states. We also changed the column headings to be more readable and make the column headings to be uniform between different sources. Upon completion of the cleaning set, we get the following column names: Date, State, Total_cases, Confirmed_cases, New_cases, Total_deaths, Confirmed_death, New_deaths. A snapshot of the cleaned data is given below:

Date	State	Total_cases	Confirmed_cas	New_cases	Total_deaths	Confirmed_dea	New_deaths
07/22/2020	AZ	150609	150609	1926	2974	2974	56
07/23/2020	AZ	152944	152944	2335	3063	3063	89
07/24/2020	AZ	156301	156301	3357	3142	3142	79
07/25/2020	AZ	160041	160041	3740	3286	3286	144
07/26/2020	AZ	162014	162014	1973	3305	3305	19
07/27/2020	AZ	163827	163827	1813	3304	3304	-1
07/28/2020	AZ	165934	165934	2107	3408	3408	104
07/29/2020	AZ	168273	168273	2339	3454	3454	46
07/30/2020	AZ	170798	170798	2525	3626	3626	172
07/31/2020	AZ	174010	174010	3212	3694	3694	68
08/01/2020	AZ	177022	177022	3012	3747	3747	53
08/02/2020	AZ	178467	178467	1445	3765	3765	18

Table 2: A snapshot of the cleaned CDC data

Next, we shift our focus to the county-level data provided. CDC provides de-identified data for every reported case in the USA. This data set has 8.4 million rows of data and can be found at https://data.cdc.gov/Case-Surveillance/COVID-19-Case-Surveillance-Public-Use-Data/vbim-akqf [4]. It has the following columns of data: cdc_report_dt(Initial case report date to CDC), pos_spec_dt (Date of first positive specimen collection), onset_dt (Symptom onset date, if symptomatic), current_status(Case Status: laboratory-confirmed case; Probable case), sex(Sex: Male; Female; Unknown; Other), age_group(Age Group: 0 - 9 Years; 10 - 19 Years; 20 - 39 Years; 40 - 49 Years; 50 - 59 Years; 60 - 69 Years; 70 - 79 Years; 80 + Years), Race and ethnicity (combined)(Race and ethnicity (combined): Hispanic/Latino; American Indian / Alaska Native, Non-Hispanic; Asian, Non-Hispanic; Black, Non-Hispanic; Native Hawaiian / Other Pacific Islander, Non-Hispanic; White, Non-Hispanic; Multiple/Other, Non-Hispanic), hosp_yn(Hospitalization status), icu_yn(ICU admission status), death_yn(Death status), medcond_yn(Presence of underlying comorbidity or disease). Upon a cursory glance, we can

see that this dataset does not provide any information about the location of the COVID case. This was the primary reason for our group to not use this dataset at all.

Safegraph mobility data: This data source logs human mobility using mobile devices. This is an important data source as it tells us about human mobility behavior during but not limited to the pandemic. Its data sets can be divided into three categories: (1) Core Places: Base information such as location name, address, category, and brand association for points of interest (POIs) where people spend time or money. Available for ~6.9MM POI including permanently closed POIs. (2) Geometry: POI footprints with spatial hierarchy metadata depicting when child polygons are contained by parents or when two tenants share the same polygon. Available for ~6.4MM POI (Geometry metadata not provided for closed POIs). (3) Patterns: Place traffic and demographic aggregations that answer: how often people visit, how long they stay, where they came from, where else they go, and more. Available for ~4.4MM POI. Our main focus was the patterns data as the Core and Geometry were unable to provide mobility data and were focused on POI. The documentation for each of these sections can be found at https://docs.safegraph.com/docs [5].

The patterns dataset was contained in four unique subsections (five for the enterprise version which we did not have access to nor was required as the fifth section had month-wise data for the number of visitors for a particular brand). The first subsection is the Home Location Distributions by State/Census Block Group. This was a month-wise number of residents of a particular state or census block group. It had the following columns: year(Calendar Year), month (Calendar month starting from 1 as of January), state(Lowercase abbreviation of U.S. state or territory), census_block_group(FIPS code for this Census block group), number_devices _residing(Number of distinct devices observed with a primary nighttime location in the specified census block group.). This gave us data about the number of people in a particular state. We took the information from January 2020 - November 2020. This gave us 5,062,411 rows of data. The high number is due to each row for each census block for each month. These rows of data were in 10 different files which were first combined into one raw file. We aggregated the state population and removed all the census block data as we were focused on the state-wise data. We also rename the column names to Date, State, State_resident_number for uniformity and readability. The final dataset has 1288 rows of data while a snapshot of the data is given below:

Date	State	State_resident_number
01/13/2020	SD	390558
01/13/2020	TN	57373
01/13/2020	TX	522099
01/13/2020	UT	2287805
01/13/2020	VA	179586
01/13/2020	VI	538210
01/13/2020	VT	945
01/13/2020	WA	27708
01/13/2020	WI	419028
01/13/2020	WV	381834

Table 3: Final Cleaned data

Notice that the state abbreviation was changed to uppercase to be uniform with the CDC data. The next subsection of data is found in the Number of Visits/Visitors by State section. This is the weekly number of visitors to a state. This is an important dataset as it tells us the mobility of people between states. It has the following columns: year(Calendar Year), month(Calendar month starting from 1 as January), state(Lowercase abbreviation of U.S. state or territory String), num_visits(Number of point-of-interest visits observed in the specified state), num_unique_visitors(Number of unique visitors observed with at least 1 point-of-interest visit in the specified state). Notice that this data does not tell from which state this visitor is coming from. This is because it is hard to track the mode of transportation for the user. A raw snapshot is given below:

year	month	state	num_visits	num_unique_visitors
2020	4	de	1195363	96985
2020	4	fl	37228870	2147601
2020	4	ga	21417041	1151037
2020	4	gu	4493	813
2020	4	hi	2103010	92866
2020	4	ia	6054528	412871
2020	4	id	2388217	170438
2020	4	il	17855361	1289760
2020	4	in	10654596	799298
2020	4	ks	5347741	365611

Table 4: Number of visitors of each state

We took the information from January 2020 - November 2020. This gave us 602 rows of data. These rows of data were in 10 different files which were first combined into one raw file. We did the following cleaning to the combined data: removed the year column (as it was 2020), removed the num_visits as we did not have any use of the number of POI's visited, changed state abbreviation to uppercase to be uniform with the CDC data and renamed column headings to Month, State, State_visitor_number for uniformity and readability. We get a small snapshot of the data in table 5.
The next subsection is the Normalization Stats which had the following rows of data: year(Calendar Year, month(Calendar month starting from 1 as January), day(Calendar day), region(When iso_country_code == US, then this is the USA state or territory. When iso_country_code == CA, then this is the Canadian Province or territory.), total_visits(All visits we saw on the given day in local time (includes visits to POI and visits to homes)), total_devices_seen(Total devices in our panel which we saw on the given day with any visit in local time (POI or home visit)), total_home_visits(Visits we saw on the given day in local time to the device's home geohash-7), total_home_visitors(Total devices we saw on the given day with at least 1 visit to the device's home geohash-7).

Month	State	State_visitor_number
1	AK	80249
1	AL	1068273
1	AR	650685
1	AS	102
1	AZ	1190181
1	BC	78
1	CA	5052878
1	CO	1016753

Table 5: Data with num_visits removed

We decided not to use this dataset as the data was based on the previous years as well which can make our results more inaccurate due to the sudden change in social interaction laws (like lockdowns) and social behavior.

Our final subsection is the Patterns section which is a dataset of visitor and demographic aggregations available for ~4.4MM POI. It has the following columns: place key(Unique and persistent ID tied to U.S. POIs), safegraph_place_id(Unique and persistent ID tied to this POI.), parent_placekey(If the place is a tenant / sub-store inside a larger place (e.g. mall, airport, stadium), this lists the place key of the parent place, otherwise null.), parent_safegraph_place_id(If the place is encompassed a larger place (e.g. mall, airport, stadium), this lists the safegraph_place_id of the parent place; otherwise null.), location_name(The name of the place of interest.), street_address(Street address of the place of interest.), and many more (A total of 29 columns). We decided not to use this data source as it had granularity higher than state-wise data, had a very large number of rows of data, a lot of preprocessing was required which was not feasible for a team of 4 in the given time, and the format was not consistent between January 2020 - November 2020.

New York times county data: after an initial collection of data, an additional source of data was added in the later stages. This dataset provided us the number of cases and deaths at a county level. These datasets can be found at https://github.com/nytimes/covid-19-data [6]. We took the county-level data from January 2020 till November 2020. It had the following columns of data: date(For each date, we show the cumulative number of confirmed cases and deaths as reported that day in that county or state. All cases and deaths are counted on the date they are first announced. Each date includes all cases and deaths announced that day through midnight Eastern Time. As the West Coast and Hawaii tend to release all of their new data early enough in the day.), county(In some instances, we report data from multiple counties or other non-county geographies as a single county. For instance, we report a single value for New York City, comprising the cases for New York, Kings, Queens, Bronx, and Richmond Counties. In these instances, the FIPS code field will be empty. (We may assign FIPS codes to these geographies in the future.)), state(full name of the state in lower case), fips(FIPS code for this Census block group), cases(The total number of cases of Covid-19, including both confirmed and probable.), deaths(The total number of deaths from Covid-19, including both confirmed and

probable.). The following is the snapshot of the raw data:

date	county	state	fips	cases	deaths
2020-08-30	McKean	Pennsylvania	42083	37	1
2020-08-30	Mercer	Pennsylvania	42085	545	13
2020-08-30	Mifflin	Pennsylvania	42087	160	1
2020-08-30	Monroe	Pennsylvania	42089	1712	127
2020-08-30	Montgomery	Pennsylvania	42091	10996	861
2020-08-30	Montour	Pennsylvania	42093	133	5
2020-08-30	Northampton	Pennsylvania	42095	4114	300

Table 6: NYTimes county data

First, we had to replace the state name with the uppercase abbreviation of the same state to make this dataset uniform with the other data sets. We also had to change the date format which was different. We also changed the column's name to make them uniform: Date, State, County_ Name, County_Fips, Cases, Deaths. This cleaning gave us 803,684 rows of data. The final cleaned data looks like this:

Date	State	County_Name	County_Fips	Cases	Deaths
08/28/2020	WV	Marshall	54051	133	1
08/28/2020	WV	Mason	54053	102	1
08/28/2020	WV	McDowell	54047	70	0
08/28/2020	WV	Mercer	54055	290	22
08/28/2020	WV	Mineral	54057	144	4
08/28/2020	WV	Mingo	54059	229	5
08/28/2020	WV	Monongalia	54061	1083	5
08/28/2020	WV	Monroe	54063	96	3

Table 7: Final cleaned county data

These sources are combined to create a single data set. Because this is all-time series data, the time column in each data source will be used to join the data. But before this step, it may be necessary to look at the individual data sources themselves. Any missing data points will be interpolated from the local average. Additionally, because this is time-series data there could be cyclical components(weekly patterns, seasonal patterns) that need to be removed. Since the state is under lockdown measures, it is expected that seasonal and weekly mobility patterns will be drastically different and can be ignored. Finally, the data will be divided into training and validation sets with 70% of data and 30% of data respectively. The training data will be further partitioned. Each partition will train the model and will subsequently be grouped together on a rolling basis. For example, if we have three groups in the training set T_1, T_2, T_3, the model will be trained withT_1, then $T_1 \cup T_2$, then, $\cup T_2 \cup T_3$. This approach is called forward chaining and trains the model by using all training data as both training and testing sets.

The last thing to note is the uniformity of the data collected. There are counties that have been collecting data for much longer than others and not every county reported their case number every day. As a result, only a subset of this dataset can be taken such that every county has the same number of data points.

Facebook Social Connectedness Index: While the NYTimes county data contains the number of covid cases for a given county, a dataset that captures the relationships between counties is still needed to construct an adjacency matrix. Facebook provides a dataset that describes the social connectedness between all counties, parishes, boroughs, and census areas. The dataset can be found at https://dataforgood.fb.com/docs/covid19/ [7]. The Social Connectedness Index measures the strength of connectedness between two geographic areas as represented by Facebook friendship ties. These connections can reveal important insights about economic opportunities, social mobility, trade, and more. In the context of covid, it could serve as another input to the model as we have done with our project.

The dataset is very large as it contains the connectedness of every county in America. There are a total of 3229 counties, boroughs, parishes, and census areas in the U.S. which means there are 3229^2 rows of data. Every row represents a connection between two of these census areas.

1	user_loc	fr_loc	scaled_sci
2	1001	1001	13971142
3	1001	1003	145133
4	1001	1005	200193
5	1001	1007	371688
6	1001	1009	95859
7	1001	1011	570502
8	1001	1013	970078
9	1001	1015	110689

Table 8: Facebook Social Connectedness Index Data

As shown in the picture above, there are three entries for each row of data. The first two columns represent the two census areas and the third column represents the social connectedness of these two areas.

Since the dataset does not contain extraneous information and the dataset has a uniform format, there is no extra work required for cleaning. However, before the dataset is able to be used to train our model, an adjacency matrix needs to be extracted. This could be easily done by converting the 3229 column vectors, one for each of the areas, into a 3229 by 3229 matrix. Lastly, since the NYtimes data does not cover every region in this dataset, the missing areas in the county covid cases dataset need to be taken out of the adjacency matrix formed as well. Another note of importance is that this dataset is slightly different from the other ones in this project. All the other datasets are time-evolving whereas this set of data is static. In other words, unlike the adjacency matrices formed by the SafeGraph Mobility data, which has one for every timestep, this dataset can only produce a single adjacency matrix static in time. However, this should not violate the premise of the project since the node features are still time evolving.

3. Approach Outline & Rationale

If county data was also included, there would be more than 3000 nodes in the graph. In order to reduce complexity, the first model will only use state and country data. The county nodes will be added later when we are confident about the approach. This means the first GNN will have 52 nodes where 51 nodes are the 50 states and DC and 1 node is the United States.

3.1 Method

This methodology is based on the Transfer Graph Neural Networks for Pandemic Forecasting article [8]. The approach is a combination of a message passing neural network that feeds embeddings into an LSTM network. The embeddings are derived from the graph's spatial components which the LSTM exploits to decipher temporal contributions that enable the LSTM to forecast infection count. We will start out by first defining the graph G = (V; E). The nodes represent each of the states and a single node represents the United States. Edges exist between nodes that share borders and an edge exists from every state node to the United States node. Note that these edges exist regardless of time and hence are static edges. Dynamic edges, which are dependent on the mobility data, are also added to the graph. The data is time series so we replicate the graph for each timestamp taking care to update the weighted edges based on the mobility data. This will lead to a collection of graphs: $G^1, G^2, \ldots, G^T$ for each timestamp. Note that the graphs can have nodes with self-loops as people can move within the states themselves. We define a vector that contains the number of cases for each one of the past d days in region u.

$$x_u^{(t)} = (c_u^{(t-d)}, \ldots, c_u^{(t)})^T \in \Re^d \quad (1)$$

This *d* day data aggregation accounts for any variations caused by possible incorrect infection count misreporting. After doing this for all nodes *u*, they are vertically inserted into a row matrix X^T. Next, we compute the Z^u, which is a vector that aggregates the mobility into *u* by accounting for both mobility within and coming towards *u.* Z^u is simply a matrix multiplication of the adjacency matrix of G^t and X^t. Note that the Adjacency matrix A^t is weighted according to the movement data into the node and is time-dependent.

$$Z^t = A^t X^t \quad (2)$$

Using an MPNN, we update the representations of the vertices of each of the input graphs by applying

$$H^{i+1} = f(\overline{A} H^i W^{i+1}) \quad (3)$$

Where f is the nonlinear activation function, $H^0 = X$, W^i is the matrix of trainable parameters of layer i, and $\overline{A}$ is the normalized adjacency matrix where the sum of the weights of the incoming edges of each node is unity. Notice that when $H^0 = X$, we calculate Z^t. The above model is applied to all input graphs separately. For a model of K neighborhood aggregation layers, the matrices $\overline{A}$ and $H^0, \ldots, H^K$ are specific to a single graph while the weight matrices $W^1, \ldots W^K$ are shared across all graphs. In order to retain useful local information, we can concatenate matrices $H = CONCAT(H^0, H^1, H^2, \ldots, H^K)$ horizontally. We can then feed these into an LSTM network to capture the long-range temporal dependencies. Finally, we apply ReLU to the output layer of the network as the prediction must be positive. Apply the mean squared error loss function to calculate the gradients.

$$L = \frac{1}{nT} \sum_{t=1}^{T} \sum_{v \in V} (y_v^{(t+1)} - \hat{y}_v^{(t+1)})^2 \quad (4)$$

above, $y_v^{(t+1)}$ is the known infection count while $\hat{y}_v^{(t+1)}$ is predicted. Figure 3 shows the architecture of the model used to make forecasts.

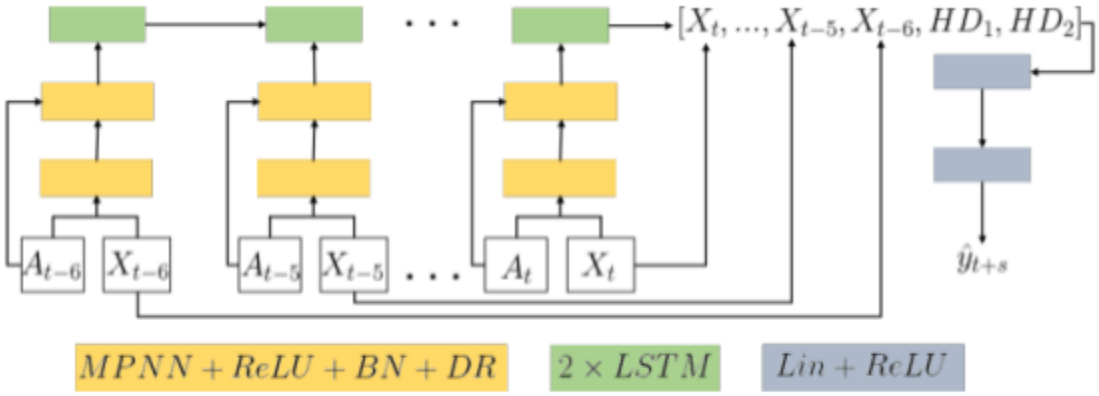

Figure 3: Overview of the MPNN+LSTM Architecture

3.2 Implementation - Model

For the implementation of the network, we have chosen to use Tensorflow/Keras and Spektral due to their compatibility and ease of usage. More specifically, Spektral is used to implement the MPNN and Keras is used for the LSTM module. One advantage of Spektral over other Graph neural network libraries is its seamless integration with Keras, so any layers from Spektral can be inserted into a Keras model.

The MPNN+LSTM model is implemented as a subclass of *Keras.models* and the MPNN is packaged into a standalone function within the subclass. Figure 4 demonstrates the implementation of the MPNN construction.

```
def build_MPNN_unit(self, dropout, net_id=1):
    """
    Function builds an MPNN unit
    args:
      dropout: float. Probabiltiy of dropout.
      net_id: int. Id of the network.
    return:
      None
    """
    L1 = []
    L1.extend((
        spektral.layers.MessagePassing(aggregate='mean',
                                       activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(dropout)))
    L2 = []
```

```
L2.extend((
    spektral.layers.MessagePassing(aggregate='sum',
                                   activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(dropout)))
self._nets[net_id] = (L1, L2)
```

Figure 4: MPNN construction function

As shown in the code above (Figure 4), there are two MPNN layers. Both of these layers are followed by a batch normalization layer and a dropout layer. In addition, they both utilize the same Relu nonlinearity for the update phase of the MPNN. They are different, however, in how they aggregate the neighboring node features; Layer 1 takes the average of the neighboring node features while layer 2 simply aggregates them.

```
def run_MPNN_unit(self, Adj, X, net_id=1):
    """
    Function calls layer given by id
    args:
      Adj: Tensor. [:, x, y]. Adjacency matrix of the graph
      X: Tensor. [:, x, z]. Node feature matrix.
    returns:
      Tensor. [x, z*2]. ouput of passig a message through the MPNN
    """
    L1, L2 = self._nets[net_id]
    Adj = sp_matrix_to_sp_tensor(Adj)
    y = None
    for i in range(0,len(L1)):
        if i == 0: # MessagePassing layer
            y = L1[i].propagate(X, Adj)
            continue
        y = L1[i](y)
    H1 = y
    for i in range(0, len(L2)):
        if i == 0: # MessagePassing Layer
            y = L2[i].propagate(y, Adj)
            continue
        y = L2[i](y)
    H2 = y
    return tf.concat((H1,H2), axis=1)
```

Figure 5: MPNN layers initialization

As shown in figure 5, the two MPNN layers take in a sparse tensor as the adjacency matrix and a NumPy array as the node feature matrix. In the second layer, the network takes in the same adjacency matrix while taking the output of the first layer as the feature matrix. Lastly, the two outputs are concatenated horizontally into one matrix.

```
def train(self, model_input, y):
    """
    Function Trains the network
    args:
      model_input: [Adj, X]
        Adj: Tensor. [:, x, x]. Adjacency matrix of the graph
        X: Tensor. [:, x, z]. Node feature matrix
      y: Tensor. [x, 1]
    returns:
    """
    Adj, X = model_input
    Adj = tf.squeeze(
        tf.convert_to_tensor(Adj, dtype=tf.float32))
    X = tf.convert_to_tensor(X[0], dtype=tf.float32)
    y = tf.convert_to_tensor(y[0], dtype=tf.float32)
    with tf.GradientTape() as tape:
        y_pred = self((Adj, X), training=True)
        loss = tf.keras.losses.mean_squared_error(y, y_pred)

    trainable_vars = self.trainable_variables
    gradients = tape.gradient(loss, trainable_vars)
    self.optimizer.apply_gradients(zip(gradients, trainable_vars))

    # metrics
    # self.loss_tracker.update_state(loss)
    self.mse_metric.update_state(y, y_pred)
    # return {"loss": self.loss_tracker.result().numpy(),
    return {"mse": self.mse_metric.result().numpy()}
```

Figure 6: Custom train function for Spektral library

One of the complications of using Spektral is that fitting the model requires a custom train function to be implemented. Figure 6 shows the code for it. The Adjacency matrix, node feature matrix, and the label matrix need to be extracted from the graph object that is passed into the function before they can be passed for training.

```
def call(self, model_input, training=True):
    """
    Function calls the net
    args:
      Adj: Tensor. [:, x, y]. Adjacency matrix of the graph
      X: Tensor. [:, x, z]. Node feature matrix
    returns:
      x: Tensor. [x, 1]. Output of the neural net
    """
    Adj, X = model_input
```

```
if not training:
    Adj = sp_matrix_to_sp_tensor(Adj)
LSTM_input = []
for i in range(0, self._window):
    LSTM_input.append(
        self.run_MPNN_unit(Adj[i,:,:],
                           X[i,:,:], net_id=str(i+1)))
x = tf.stack(LSTM_input, axis=1)
x = self.LSTM1(x)
output, final_memory_state, final_carry_state = self.LSTM2(x)
#x = X+x
#Lin?
x = final_memory_state
x = self.Lin(x)
return x
```

Figure 7: Call function and LSTM layer implementation

The last function in the model subclass is the call function which finally builds the entire model. This is where our LSTM network processes the output from the MPNN network. As shown in figure 7, the number of MPNN created is equal to the size of the window(hyperparameter), and the approach is reasoned out in the previous section. The LSTM module comprises two layers. The first LSTM layer only returns the entire output sequence. The output is then passed to the second LSTM layer where the final memory state of the LSTM is passed to a linear layer with Relu activation. The output then is the predicted number of new cases for each of the 52 nodes.

3.3 Implementation - Dataset Creation

Creating the dataset to be passed to the MPNN network is challenging. A simple method is to use the prebuilt Spektral library where data i.e. Adjacency matrix and etc can be packaged into a graph object.

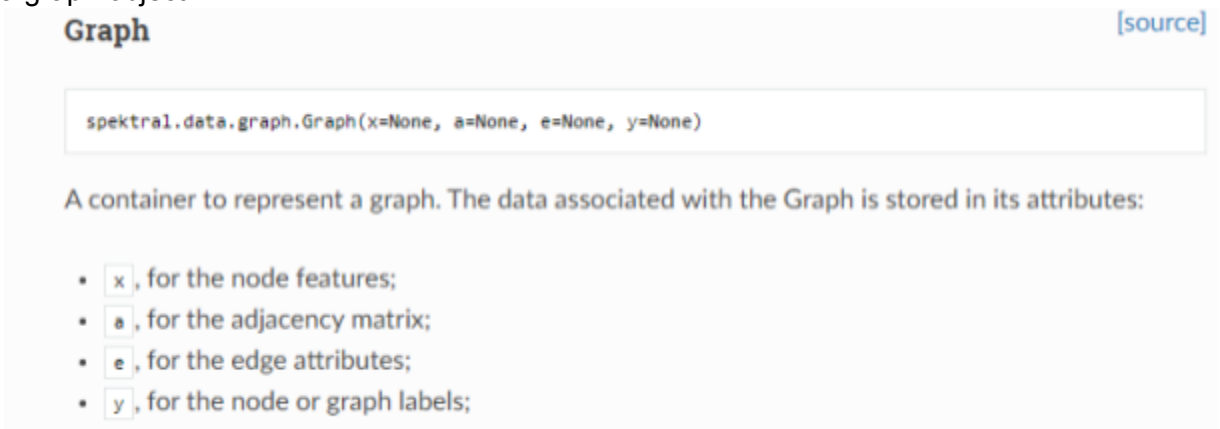

Figure 8: Container function for graph object creation

The Graph function that returns a graph object. In our case, we will not pass in an edge attribute matrix. Another point to note is that the adjacency matrix needs to first be converted into a Scipy compressed sparse row matrix for the function to work correctly.

When passing the training data into the Keras fit function, a loader function called SingleLoader needs to be called. The data Is then passed through the model.fit function which takes in the data from the training loader and the validation loader with the epoch steps and callback function. This can then be seen in the figure shown below:

```
# Train model
loader_tr = SingleLoader(dataset, sample_weights=mask_tr)
loader_va = SingleLoader(dataset, sample_weights=mask_va)
model.fit(loader_tr.load(),
          steps_per_epoch=loader_tr.steps_per_epoch,
          validation_data=loader_va.load(),
          validation_steps=loader_va.steps_per_epoch,
          epochs=epochs,
          callbacks=[EarlyStopping(patience=patience, restore_best_weights=True)])
```

Figure 9: How to call SingleLoader function

Although this approach is straightforward, it proved more useful for us to pass the data directly into the neural network without wrapping it as a graph object. Instead, we simply construct the adjacency matrices, feature matrices, and output vectors. Then, we transform the adjacency matrix into a sparse matrix. These matrices are then passed to the network directly without wrapping them in the graph object.

4. Expected Outcome

Our expectation is that the results will be varied and we will be unable to conclude the effectiveness of the proposed graph neural network. Firstly, there is a limited amount of data from which to train the temporal graph neural network. Typically deep learning models require massive amounts of data and unfortunately, there is not enough data to completely capture the signals in the time sequence. Due to the sparse nature of the datasets, it is difficult to precisely capture human interaction. For example, the state wise visitor data does not tell the starting state of those visitors. We are unable to get information about country wise visitor data as well. With the lack of data sets and with much diversity between them it was very difficult to gather large amounts of data. We believe that ARIMA will be best able to make the forecasts; however, the mean squared error of LSTM and MPNN-LSTM should be much lower.

5. Evaluation Plan/Results

In order to evaluate this regression task, we will test the effectiveness of our model by measuring the mean squared error loss for each of the models. Additionally, we will compare the forecasting ability of each of the three different models to get a more visual result of the strength of the model for forecasting. You can check out the forecasting ability of each of the models in the figures below.
The figure 10 plot shows the forecasting ability of the ARIMA model, the middle plot shows the LSTM model and the bottom-most plot shows the GNN model. We are trying to forecast the number of new cases in the United States. Although forecasting for individual states is also possible, it requires careful consideration to build these for the ARIMA model. Forecast values for individual states exist for the GNN which we will see later.

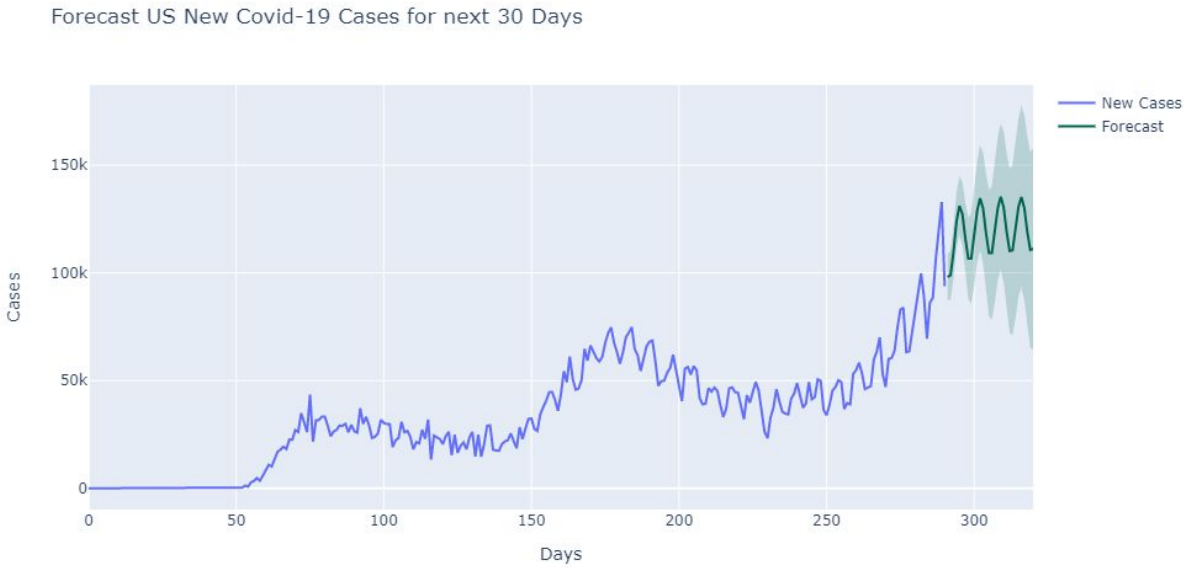

Figure 10: This figure shows 30 day forecast with the ARIMA model

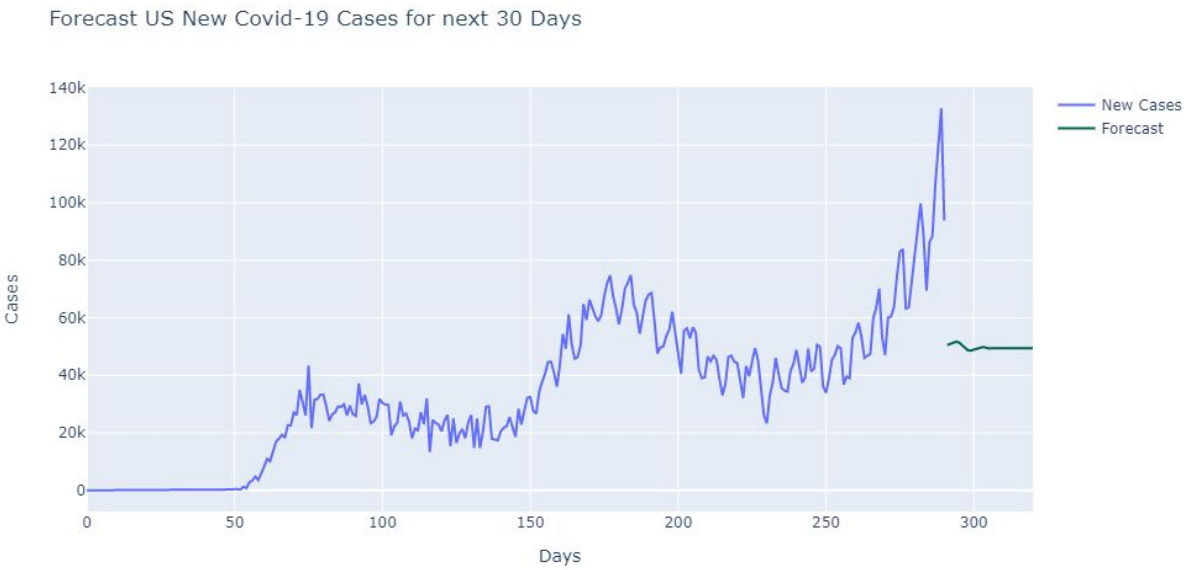

Figure 11: This figure shows the 30 day forecast with the LSTM model.

Looking at the plots, it is very obvious that the ARIMA model was able to perform the best as its forecast better captured the dynamic of the cases. The ARIMA model also offers a confidence interval for its forecast. ARIMA is designed to work with time-series data sets and thus is able to

make a more clear forecast. The other two plots are deep learning models and their forecasts are not that good.

As an educated guess, this could be the result of not properly fine-tuning some of the hyper parameters used in the model. For instance, both the GNN and the LSTM networks use a window of historical data in order to forecast the next value in the time series. The result of the forecasts could be strongly dependent on the size of this window. It is possible that more careful fine-tuning or grid search would enable the deep learning models to capture the signal in the data more effectively.

Additionally, one of the limitations of our models is the lack of data. Deep learning models generally require massive amounts of data in order to be effective. In this case, we only have 291 days of data points to train the model for each state and the United States. This is because each row of data corresponds to a day and we have COVID data from January 2020 till the first half of November 2020. This requires us to process the data multiple times to train effectively which may lead to overfitting.

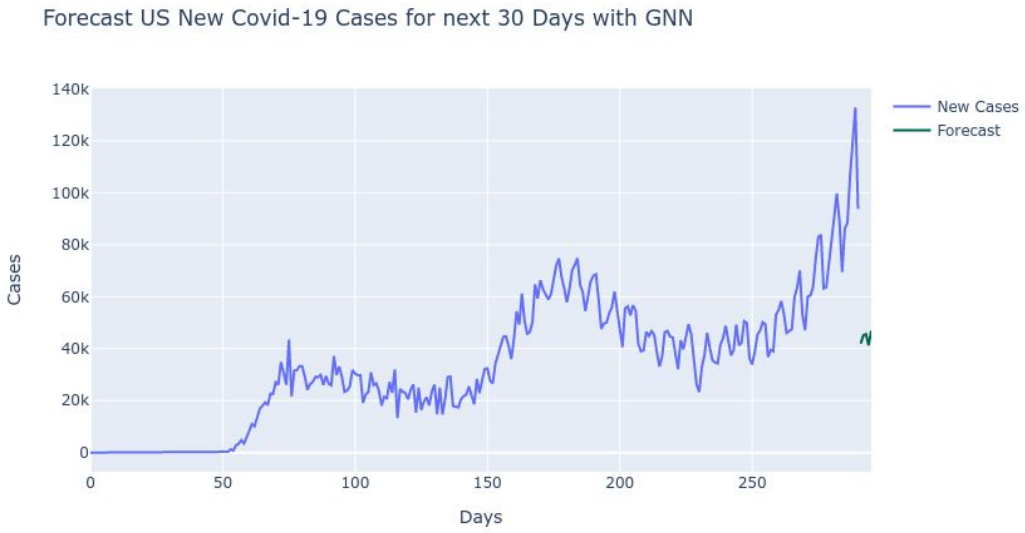

Figure 12: This figure shows the 5 day forecast with the MPNN-LSTM model.

Our goal was to study the effectiveness of the graph neural network so we will take a look at its loss. We used the mean square loss to train the model and you can see the result in figure 11.

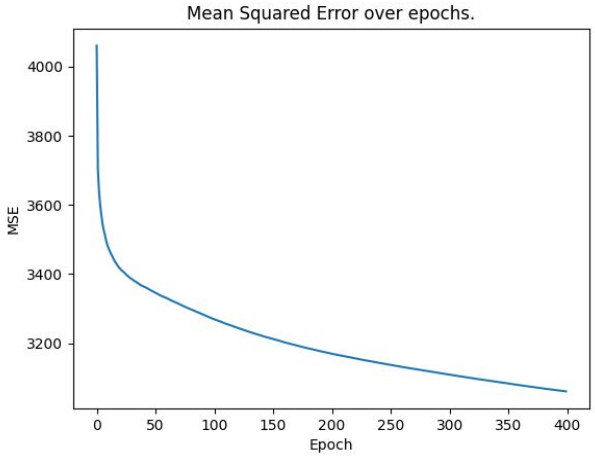

Figure 13: This figure shows the mean squared error of the MPNN-LSTM model over the training epochs.

As expected the loss is converging over the training epochs. The large mean squared error value is a result of trying to estimate the number of new deaths which is in the thousands. It is clear from the figure that the loss has not converged completely as it is not asymptotical to a mean squared error value. We decided to limit the amount of training time to 400 epochs to not overfit the data.. This will allow us to compare the neural network to the LSTM network and the ARIMA model.

Model	MSE - New Cases
ARIMA	377.89
LSTM	2988.08
MPNN-LSTM	3071.39

This table shows the mean squared error for all three models trained to forecast the number of new cases in the United States. Clearly the ARIMA model has the lowest error and outperforms the other two as expected from the forecast plots. This doesn't necessarily mean that the deep learning models are not good as the MPNN-LSTM network was quite effective for the original author. It simply means we cannot conclude which is clearly the better model.

6. Discussion

In the end we were not able to complete our evaluation of the county level data due to an incomplete team and other team related factors. This includes not being able to finish the build county data function within the allotted time and not being able to include a container function that packages the data into a graph object. Despite this, we believe we have made strides given the limited resources such as completing a model that is invariant to the granularity of the input data and the successful evaluation of the state level data.

6.1 Improvements

We can take some actions to further improve the performance of the MPNN-LSTM network. The most obvious step is to perform a grid search over the hyperparameters to the model (window size, learning rate, batch size, etc.). By doing a grid search, we can choose the parameters with the lowest mean squared error. Additionally, we can improve our performance by getting more data. This is difficult for us given as we are bottlenecked by the fact that the data by its nature is limited. Instead, we can pretrain the model under the assumption that other pandemics exhibit the same dynamics. Data from other pandemics could provide valuable signals to train the network on the current pandemic. Infact, an MPNN-TL model which applies transfer learning is discussed in the paper [8], which presents the MPNN-LSTM. By applying some of these methods, we can further improve the effectiveness of the graph neural network approach. Another improvement can be done on gathering more datasets for more information. For this project we had 291 rows of data (as every day is represented as one row of data and we collected from January - November 2020.) We are able to gain a row of data from very day passing by which can prove consequential in improving our model. We can also include more number of datasets with unique information which is lacking in our data set.

6.2 Extension

The application of this type of network is potentially limitless. As long as the dataset exhibits a graph structure and has a time varying component, the network can be used to make some type of forecast. For instance, an example that immediately springs to mind is that of network traffic forecasting. Since these networks typically have a graph structure; street intersections are always connected by streets; an internet or bluetooth network nodes are connected by connections. Each of these nodes have their own set of features. In the street intersection example, it could encapsulate the peak hour time, number of lanes of the adjacent streets, or even the geographical location of the intersection. Likewise, in a wired/wireless network the node features could be the active hours of a particular node or the node's type of connection and therefore its bandwidth. Being able to forecast when the traffic peaks or how other factors could affect the flow of traffic could greatly aid in the decision making process of urban planners or network administrators.
There is, of course, the application of tracking other diseases. Since similar data most likely also exists for other types of diseases, the key difference would simply be what kind of information the adjacency matrix captures.

7. Responsibilities

This section is dedicated to list the roles and responsibilities played by every member in the team since the start. Since the beginning two members out of the six did not contribute nor reply to the team emails in any manner. They are Dongruo Zhou and Chang Xu. Chang did not reply to any emails and was reported to the professor in due time while Dongruo dropped the class after the due date of paper summaries in which he did not contribute. The following members had the following roles.

Sripath Mishra: He is responsible for the submission of assignments and papers for the team. He manages the work to be done and ensures that the work is done within the deadline. He has contributed to the paper summaries and was the main formater after all the paper lists were in. He along with a member of the team ideas was the main person to accrue all the summaries written from the papers collected. He also had written the summaries for the papers he had collected for the papers list. Then he shifted his focus towards the cleaning data part of the project after collecting the data from the CDC and the safe graph he was responsible for preprocessing all those data sets. Codes to that are seen in the Cleaner_script Github folder. Then he worked on slides 1-3 for the paper presentation. Later on, he was responsible for slides 1-4 in the project presentation. Working on slides means preparing the slides and presenting them during the presentation. Then he updated the readme and requirements.txt in the codebase to ensure easy execution of the code. Finally, he worked on parts of the data preparation of this document, formatting this document, and writing this section as well.

Vishnu Devarakonda: He has contributed to the paper summaries. He then diverted his focus towards the project proposal and completed parts of it. He also had written the summaries for the papers he had collected for the papers list. Then he shifted his focus towards the model generation part of the codebase which includes Completing the ARIMA model implementation, LSTM model completion, added the border states data, build_data.py which builds the adjacency matrix and the feature matrix from the data. Then he worked on the networks file along with added trainer and updated older codes to support the new network. He added JSON for hyper-parameters, changed the trainer to model, and implemented a model for training and forecasting. Then he updated the documentation and added a pre-trained model for example. Then he is responsible for slides 4-8 for the paper presentation. Later on, he was responsible for slides 5-14 in the project presentation. Working on slides means preparing the slides and presenting them during the presentation. Finally, he worked on parts section 3 and other parts of this document.

Jason Kao: He has contributed to the paper summaries. He then diverted his focus towards the project proposal and completed parts of it. He also had written the summaries for the papers he had collected for the papers list. Then he focused on preliminary GNN with placeholder data/actual data. He also worked on the LSTM model along with the MPNN layer. He worked on county-level data and created an adjacency matrix and the feature matrix from the data. He also created the labels for the data and worked on cleaning parts of the data and finding missing parts of the data. Then he updated the documentation and added a pre-trained model for example. Then he is responsible for slides 9-19 for the paper presentation. Later on, he was responsible for slides 15-23 in the project presentation. Working on slides means preparing the slides and presenting them during the presentation. Finally, he worked on major parts of this document.

Boya Ouyang: He has contributed to the paper summaries. He then diverted his focus towards the project proposal and completed parts of it. He also had written the summaries for the papers he had collected for the papers list. Then he is responsible for slides 20-26 for the paper presentation. Later on, he was responsible for slides 24-27 in the project presentation. Working on slides means preparing the slides and presenting them during the presentation. Finally, he worked on parts of this document.

8. References

1. Chimmula VKR, Zhang L (2020) Time series forecasting of COVID-19 transmission in Canada using LSTM networks. Chaos Solitons Fractals 109864
2. Lauer SA, Grantz KH, Bi Q, et al (2020) The incubation period of coronavirus disease 2019 (COVID-19) from publicly reported confirmed cases: estimation and application. Ann Intern Med 172:577–582
3. Bell Franziska, Slawek S (2018) Forecasting at Uber: An Introduction. In: Uber Eng. https://eng.uber.com/forecasting-introduction/
4. Calgary COVID-19 Case Surveillance Public Use Data | Data | Centers for Disease Control and Prevention. DataCdcGov
5. (2020) Places Schema. SafeGraph
6. (2020) Nytimes/Covid-19-Data. GitHub
7. Facebook Social Connectedness Index - Humanitarian Data Exchange. DataHumdata.org
8. Panagopoulos G, Nikolentzos G, Vazirgiannis M Transfer Graph Neural Networks for Pandemic Forecasting