HFSSG: A Hybrid Framework combining SIR pandemic model and Spatio-temporal Graph neural network forecasting the COVID-19 Impact in the United States

Liqin Zhang[1], Siyin Ma[2], Zhejian Jin[3]

Abstract Researches on infectious diseases has always been an important topic in interdisciplinary fields. The spread of infectious diseases not only depends on the attributes of diseases itself, but also depends on the network structure of the crowd. Since late 2019, COVID-19 has become a worldwide pandemic. In order to better understand and control the spread of COVID-19, many researchers collaborated and modeled this process to forecast its influence on the neighbouring areas. SIR model is a common compartmental model that well-defines disease transmission dynamics. However, it assumes a relatively closed system, and it does not take interactions among regions into consideration. Based on this, we utilize Graph Neural Network using data that describes inter-region interactions, which captures the features of messages among the regions. In this work, we propose a novel framework for COVID-19 case examination and prediction that uses Graph Neural Network. Different with existing time series models, this model learns from a spatio-temporal graph. We then evaluate this model on the California COVID-19 data set.

Key words SIR(SEIR) Pandemic Model, Machine Learning, ST-Graph Neural Network

1. Student ID: 517370910123, Email: graves_zhang@sjtu.edu.cn
2. Student ID: 517370910003, Email: msy841@sjtu.edu.cn
3. Student ID: 517370910167, Email: jinzhejian@outlook.com

Contents

Introduction

Problem Statement

With the rapid growth and spread of Covid-19, the ability to accurately forecast caseload is extremely important to help inform policymakers on how to provision limited healthcare resources, rapidly control outbreaks, and ensure the safety of the general public. In order to prepare, understand, and control the spread of the disease, we have come together in a collaborative effort to model and forecast COVID-19. In this project, we combine two powerful models (SIR and ST-GNN) into one integrate framework to perform better prediction capabilities in COVID-19 cases with lower cost. We propose a novel hybrid multi-network based framework named HFSSG that learns to select relevant edges and graph representations by pre-computing results from SIR model, along with RMSLE and Pearson values. On top of that, this model, with the help of ST-GNN, it learns the parameters for the pandemic model, and better fit the coronavirus model in real life. This main flowchart of our framework is shown in Figure 1.

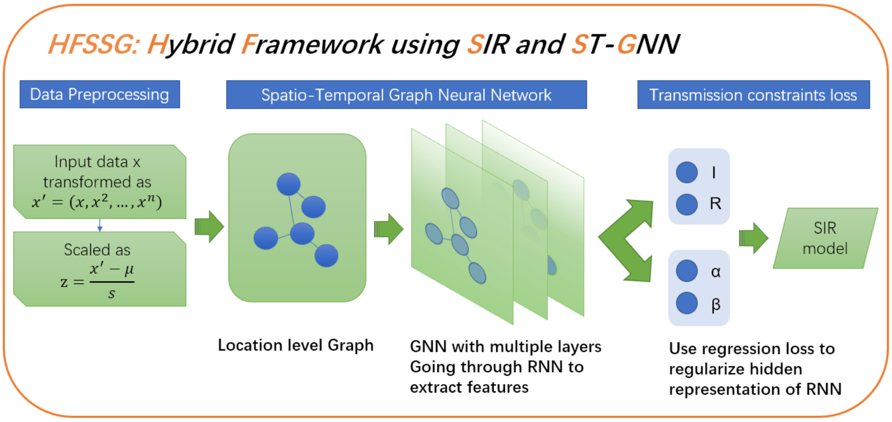

HFSSG: Hybrid Framework using SIR and ST-GNN
Data Preprocessing
Spatio-Temporal Graph Neural Network
Transmission constraints loss
Input data x transformed as $x' = (x, x^2, \dots, x^n)$
Scaled as $z = \frac{x' - \mu}{s}$
Location level Graph
GNN with multiple layers Going through RNN to extract features
I
R
α
β
SIR model
Use regression loss to regularize hidden representation of RNN

FIGURE 1. *Flowchart of our main framework*

Planned Approach

SIR Pandemic Model SIR model is a simple method describing the epidemic spread that can be applied to any network structures. It divides the popularity into three categories, representing three stages of a person. The three functions with time t is used to simulate the three compartments.

1. Susceptible $S(t)$: people before getting infected, but can be infected by neighbors.
2. Infectious $I(t)$: people who have been infected, and can infect others.

3. Removed $R(t)$: people who endured a complete infection cycle and are immune to the disease.

The number of population is $N(t)$, where $N(t) = S(t) + I(t) + R(t)$. The SIR model describes the change between the three parts of people. We introduce two parameters β and γ.

- β Effective contact rate
- γ Recovery rate

To be more specific, β describes number of susceptible persons that a patient can infect per unit time is proportional to the total number of susceptible persons in the environment. γ implies that the number of people removed from the infected person per unit time is proportional to the number of patients. Three differential equations are used to sketch the model:

$$\frac{\mathrm{d}S}{\mathrm{d}t} = -\frac{\beta SI}{N}$$
$$\frac{\mathrm{d}I}{\mathrm{d}t} = \frac{\beta SI}{N} - \gamma I$$
$$\frac{\mathrm{d}R}{\mathrm{d}t} = \gamma I$$

We have to estimate the value of β and γ simultaneously. The method we use to fit the simple and static SIR is optimization minimizing the sum of squared error of estimated value and real value for infected and recovered population. We use the function provided by python in scipy to do the optimization.

Graph Neural Netwok GNN is a suitable model to disseminate the geographical information as well as the human mobility across location to address the preliminary prediction with a certain amount of factors taken into account. In order to imitate infectious disease modeling, we plan to take create a graph with multiple nodes representing the states of a country and edges defining the spatial and temporal dependencies among states. We present our graph as a multi-layer stack, where each layer manifest the county connectivity graph for one day. Similar as the message-passing framework, we define the update at layer l as

$$\boldsymbol{m}_i^{(l+1)} = \sum_{j \in \mathcal{N}(i)} \mathcal{F}^{(l)}\left(\boldsymbol{h}_i^{(l)}, \boldsymbol{h}_j^{(l)}\right), \quad \boldsymbol{h}_i^{(l+1)} = \mathcal{G}^{(l)}\left(\boldsymbol{h}_i^{(l)}, \boldsymbol{m}_i^{(l+1)}\right)$$

where $\mathcal{F}^{(l)}$ and $\mathcal{G}^{(l)}$ are learned message functions and node update functions respectively, $\boldsymbol{m}^{(l)}$ are the messages passed between nodes, and $\boldsymbol{h}^{(l)}$ are the node representations. Notice that the bottom layer representing the date cases began appearing in the US and the top layer representing the newest date in our dataset.

Related Works

Various work on SIR model and its extensions has been applied on COVID-19 analysis. Hao et al.[4] reconstruct the outbreak in Wuhan (China) using an extended SAPHIRE model including seven compartments. A CovsirPhy Development Team[5] provided data analysis methods in python package based on SIR-derived models including SIR-F/SIR-FV/SEWIR-F. Yang et al.[6] used previous pandemic data to pretrain the LSTM, and then apply it to predict COVID-19 progression in China. However, different pandemics have different infect ability, so it may lead to inferior prediction results if the model transfer previous pandemic progression directly at the early stage of the pandemic. We will mainly focus on SIR fitting with least squares. We simplify the compartment process since our approach is to apply SIR result in featurization of graph neural network.

Apart from traditional epidemic infectious models like illness incidence forecasting and dynamic disease transmission model as SIR, SEIR[7]. Many novel approaches are based on deep neural networks. Deng et al.[8] proposed a graph message passing framework to combine learned feature embeddings and an attention matrix to model disease propagation over time. Google research team Kapoor et al.[9] apply a simple graph neural network for COVID-19 forecast prediction.

Moreover, some hybrid model has ingeniously combined disease transmission model and gnn model. Jie Zhou, Ganqu Cui etal.[10] combine SEIR and RNN and develop a spatio-temporal graph neural model, where the node feature is based on SEIR and the edge feature is based on the RNN model.

Datasets and Insights

We make full use of two datasets: the New York Times (NYT) COVID-19 dataset and the Google COVID-19 Aggregated Mobility Research Dataset. The NYT dataset collects all COVID-19 related data and is up to date til Nov.12 2020. It includes but not limits to the number of active cases, confirmed cases and deaths among different locations in the US. The Aggregated Mobility Research Dataset helps us understand the quantity of movement, while the other information combined make distinct node features for NYT. The following figure shows a overview of the infection rate in NY in the United States.

4. Xingjie Hao 2020.
5. Lisphilar 2020.
6. Yang **andothers** 2020.
7. Pei **and** Shaman 2020.
8. Deng **andothers december** 2019.
9. Kapoor **andothers** 2020.
10. Zheng **andothers** 2020.

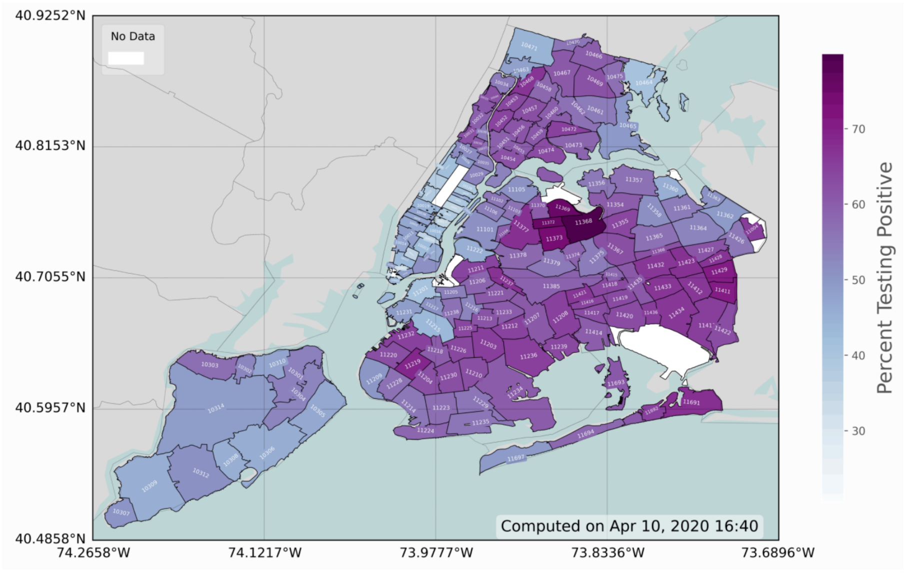
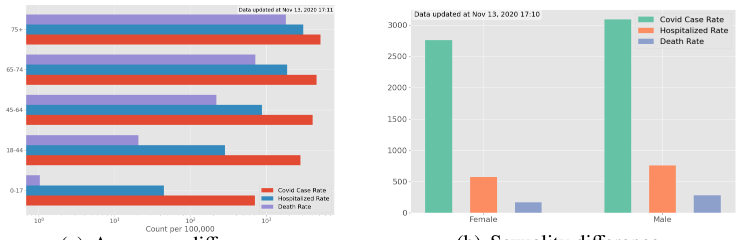


FIGURE 2. *The zip code map infection rate of New York City*

Information Visualization

The aim of this part is to provide a initial understanding of the vast amount of data related to COVID-19, based on the insight we gain from the graph, we can embed the essential information to create graph structure from that data and allow users to easily understand node features and edge weights.



(a) Age group difference (b) Sexuality difference

FIGURE 3. *Covid-19 Cases/Hospitalized/Death rate*

To gain some insights as we are constructing the graph neural network, we notice that different aged groups and different sexuality tend to have different symptoms facing the COVID-19. As a result, we decide to extract the feature and design a factor

representing those information, since the average values across all aged groups or bi-sexuality won't be useful when training our GNN model.
From the above figure, we find that other than the structural node labels, the node information matrix X also provides opportunities to include explicit features, by concatenating each node's embedding vector to its specific row in X, we can make the HFSSG model simultaneously learn from all types of features.

Graph Representation

Node Embedding

To construct a graph from scratch, we have to define a set of nodes and their edges that denotes connectivity. In case of a geographic map, we take 50 nodes as the states of the United States, and the edges be the common borders between two given states. Therefore, we have 50 nodes in total in our epidemic network. Figure 4 visualizes the embedding of nodes with the name labeling the states:

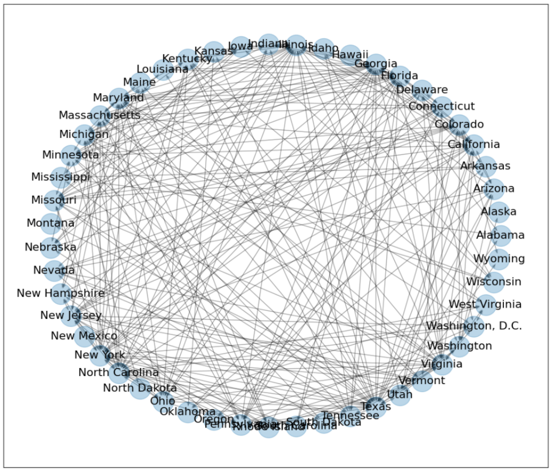


FIGURE 4. *State embedded nodes*

The features of a node is both static and dynamic, we combine both of the two properties. We include 4 variables for the static evaluation, which are population, population density, longitude and latitude. For the dynamic features, we includes the number of cases in the epidemic. The type of cases are confirmed, recovered and death.

Edge Embedding

For the weight of each edge, we decide to include the properties including the geographical characteristics and population of two connected nodes. We define the edge weight in the formula,

$$w_{ij} \propto p_i^{\alpha_1} p_j^{\alpha_2} \exp\left(-\frac{d_{ij}}{\alpha_3}\right)$$

where $\alpha_1, \alpha_2, \alpha_3$ are hyper parameters. Through this formula we want to show that the state with larger population are more likely to influence another state connected with it in the network. And the traffic mobility is also an important factor where convenient and developed transports facilitate the propagation of epidemic. And here the distance about traffic we use is the air traffic distance.

Methods

SIR with Machine Learning

SIR model divide the population into three compartments, susceptible, infectious and recovered. There are two parameters, β and γ. The problem is how to fitting the two parameters to data. We apply the least square method, which allow us to fit β and γ simultaneously. It means that the sum of squared errors is a surface in two dimenions and we are looking for the minimum of this surface. This process is referred to as optimization and, fortunately for us, there are many robust algorithms available for this purpose. One of them, Limited Memory BFGS, is implemented in the function `scipy.optimize.minimize` in python. So we can use it to finish the fitting process. Before that, we look at the data of cases in the united states. We focus on the susceptible, infectious and recovered population from the dataset[11] confirmed, recovered and death population.
The plot compares the estimated and real epidemic condition in the USA. It is accurate for the known value, but for estimation, we know that the cases has already exceed the value predicted in the graph. Then we find that the prediction is not that accurate if we want to predict the long-run data based on just a few train set.

Result Analysis

The reason behind the inaccuracy is that the parameters are always changing with the real events like the release of policies and enhance of people's awareness in wearing masks. So we have to find a way to generate a dynamic or times series method fitting the transmission rate and recovery rate. We have to chop the time serires into multiple time stamps and determine the parameters for each of the period.
Also, the limitation of using only population data is that it omits various information geographically, like the latitude and longitude. It also cannot embed the network structural information between different states like the distance and population density

11. Guidotti **and** Ardia 2020.

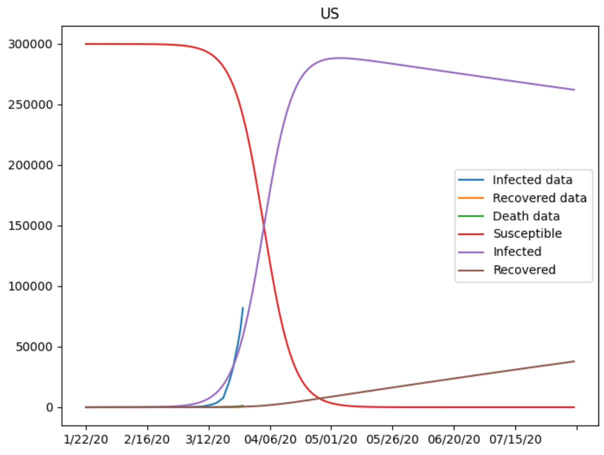


FIGURE 5. *Stable SIR model fitting.*

similarity. Therefore, the original way is not applicable. So we design HFSSG to fit the dynamic parameters.

Spatio-Temporal Graph Neural Network

Graphs are natural representations for a wide variety of real-life data. Spatio-temporal graphs are a kind of graph that model connections between nodes as a function of time and space, and have found uses in a wide variety of fields.

Use GCN to Extract Spatial Features

With considering the disease transmission status of similar locations, more accurate prediction can be made. Here we apply the Graph Convolution Networks (GCN) model to extract spatial features.
We construct a two-layer GCN. First we build a sliding window. Let $\boldsymbol{X}_t = [\boldsymbol{X}_1, \boldsymbol{X}_2]$ with $\boldsymbol{X}_1 \in \mathbb{R}^{N\times T\times L_I D_1}, \boldsymbol{X}_2 \in \mathbb{R}^{N\times T\times L_I D_2}$, where N is the number of nodes,T is the number of time stamps, L_1 is the length of the input sliding window, D_1 is the number of static features and D_2 is the number of dynamic features. The GCN can be expressed as:

$$z_t = \widehat{A}\,\mathrm{Relu}\left(\widehat{A}X_t W_0\right) W_1$$

where A denotes the normalized adjacency matrix of the graph G, W_0, W_1 denote the weight matrix in the first layer and the second layer.

We use the Graph Attention Networks (GAT) model to extract spatio features. In our model, each node representing one location get information from its neighbours. For example, a city with large amount of population will have more effect on it neighbours. The node feature for similarity can be calculated as:

$$e_{ij} = a\left(\boldsymbol{W}_a \mathbf{z}_t^i, \boldsymbol{W}_a \mathbf{z}_t^j\right)$$

where a is the attention coefficient, $\mathbf{z}_t^i$ is the i-th nodes in h_t. W_a is the linear transformation weight matrix which maps the input to a feature space of D-dimension. Then, the final attention coefficient is defined as:

$$a_{ij} = \text{softmax}\left(e_{ij}\right) = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{N} \exp\left(e_{ik}\right)}$$

We use multi-head strategy to calculate **K** independent attention information and sum together to the get representation $\widetilde{\mathbf{z}}_t^i$ for i-th node:

$$\widetilde{\mathbf{z}}_t^i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j=1}^{N} a_{ij}^k \boldsymbol{W}^k z_t^i\right)$$

Here, W^k is the weight matrix for the k-th head.

Use RNN to Extract Temporal Features

The node embedding contains spatial features extracted from the graph. Despite the spatial features, pandemic prediction is also related with temporal features. We also want to use historical temporal patterns to make a better prediction. We input the node embedding into Gate Recurrent Unit (GRU) in order to learn temporal features. We use max-pooling to integrate node embeddings for each location, which is:

$$\widetilde{Z_t} = \text{maxpool}(\widetilde{Z_t^0}, \widetilde{Z_t^1}, ..., \widetilde{Z_t^N})$$

By using the MaxPooling operator, the most important features of all of the nodes are selected, and the dimension of the node information is reduced.
We use Gated Recurrent Unit (GRU) to extract temporal features. GRU is a recurrent neural network that has good performance on temporal sequences modeling. The GRU's hidden representation can be calculated as:

$$h_t = \text{GRU}(\widetilde{Z_1}, \widetilde{Z_2}, ..., \widetilde{Z_t})$$

For each location, we build a model belonging to that location. The location index i is omitted. h_t therefore contains both spatial and temporal features based on our STGNN model.

Hybrid Framework using SIR and ST-GNN

Since we have learned the representation of the nodes in epidemic network. Then we need to combine the STGNN with SIR model, which means using the embedding value to fit the parameters of SIR model.
As we have discussed in the result analysis part of SIR with Machine Learning section, the original model fails to catch the dynamic characters of the epidemic transmission. Therefore, we derive a way to split the entire time into multiple slots. Define prediction window L_p where β and γ keep constant within one window. A time stamp represents a prediction window, and there are T/L_p sections in total. Each time step is symbolised using index t. So our tast is to predict β_t, γ_t for all the time stamp t. We use Multi-Layer Perceptron and sigmoid activation for the neural network construction.

$$\beta_t, \gamma_t = \text{sigmoid}\left(\text{MLP}\left(\boldsymbol{h}_t\right)\right)$$

where MLP indicates Multi-Layer Perceptron, and h_t is the output from the former embedding.
We can also predict the increment of infected and recovered cases, denoted as $\widehat{\Delta I_t}$ and $\widehat{\Delta R_t}$. Similar to the method we use for the parameter fitting, we also use Multi-Layer Perceptron.

$$\widehat{\Delta I_t}, \widehat{\Delta R_t} = \text{MLP}\left(\boldsymbol{h}_t\right)$$

where h_t is the output from the former embedding.

$$\begin{aligned} \widehat{\boldsymbol{I}_t} &= I_{t-1} + \sum\left(\widehat{\Delta I_t}\right) \\ \widehat{\boldsymbol{R}_t} &= R_{t-1} + \sum\left(\widehat{\Delta \boldsymbol{R}_t}\right) \end{aligned}$$

Then we can derive the infected cases and recovery cases through a cumulative sum over the increment value of them.
We define the loss function in the normal way using Mean Squared Error(MSE). We can optimize our estimated value through restricting it close to the real value.

$$L = \left(\widehat{I_t} - I_t\right)^2 + \left(\widehat{\boldsymbol{R}_t} - \boldsymbol{R}_t\right)^2$$

where $\widehat{\boldsymbol{I}_t}$ and $\widehat{\boldsymbol{R}_t}$ denotes the estimated value and $\boldsymbol{I}_t$ and $\boldsymbol{R}_t$.

Results

Utilizing the $\boldsymbol{I}_t$ and $\boldsymbol{R}_t$ given from the Multi-layer Perceptron, we could calculate and represent the predicted case information as normalized vectors that happened during the last d days. We use `networkx` and `kamada_kawai_layout` scheme to represent the structure of network and death count numbers published by the New York Times, which includes daily reports of new infections and deaths at both state and county level in US. As is shown in figure 6, the darker the color indicates more cases of COVID-19 locally, which usually also has more edges connected to other nodes.

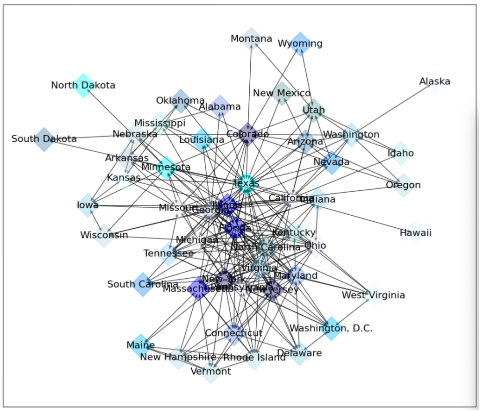


FIGURE 6. *Nodes with death counts*

In order to properly utilize the geo-information we embed in the nodes, we locate all nodes on a us map based on its latitude and longitude. For a better visualization, we consider the basic reproduction number as an important factor to denote the contagiousness of a node among its neighbors. Given d_i as the degree of each node, we estimate q as the proportion of its neighbors subsequently infected and average, thus we define the basic reproduction number as

$$R_0 = q * d * \frac{\operatorname{avg}\left(d_i^2\right)}{\left(\operatorname{avg} d_i\right)^2}$$

Consider for all reproduction values ranging from 0 to 1, we denote the top 10% with higher probability to cause infection cascade as the strong nodes, while for the rest are denoted as weak nodes. For the edges connecting the nodes, we only preserve the top 50% weighted neighbouring edges for simplicity. Figure 7 shows the combined geo-info graph structure map.

Sensitive Analysis

HFSSG model is compared with some of the existing baselines:

1. **SIR**: the Susceptible Infectious Removed model. It's a simple and traditional epidemiological model.
2. **SEUR**: the Susceptible Exposed Infectious Removed model. SEIR has one more exposed state compared with the SIR model.
3. **HFSSG**: Our proposed model.

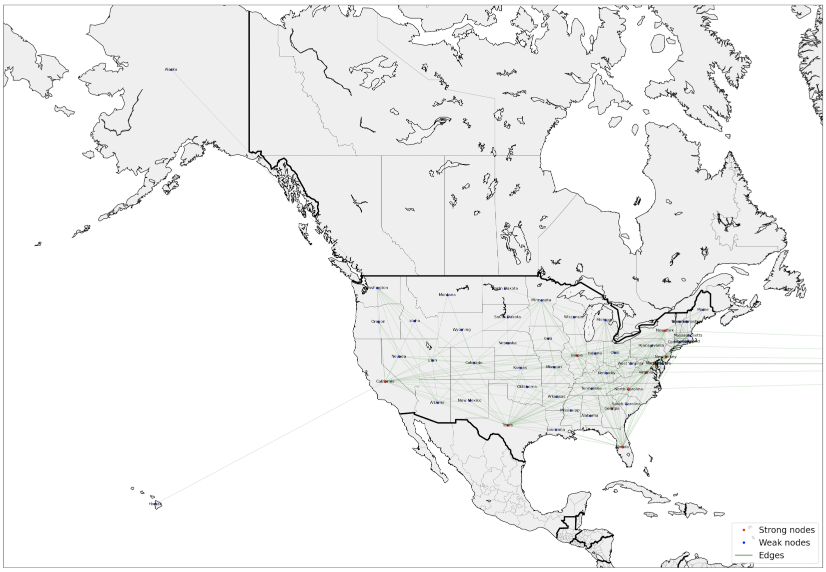


FIGURE 7. *Graph structured Geo-Info Map*

The prediction is in state-level scale. In order to evaluate the ability of HFSSG, we use the mean square error (MSE) and mean absolute error (MAE). We established a table based on this evaluation method. We use different history data window to compare the performance.

Performance comparison with data window size 5:

Model	MSE	MAE
SIR	2965210	910.08
SEIR	1950414	710.34
HFSSG	421372	264.13

We also calculated the Performance comparison with data window size 20:

Model	MSE	MAE
SIR	47592364	3680.85
SEIR	28741861	2836.37
HFSSG	5256103	1204.36

From the results, we can see the HFSSG has a more accurate prediction than SIR and SEIR, both in short time period and long time period.

Strength and Weaknesses

We discuss the strength and weaknesses of our model. First from the result section, we can find our model performance exceeds that of SIR and SEIR. The main reason is that we use state-level data and include location information to predict the epidemic propagation.
We consider this dependency between two neighboring states where the communication is more frequently due to the transports convenience. Through network embedding with deep learning methods, we build a clear pipeline from raw information to low-dimensional representations, which allows us to include geographical characters during prediction work. The ability to explain hidden features is why our model performance exceeds that of traditional SIR and SEIR.
In node embedding process, we also consider some other factors like population density and population. They also reflects the similarity between two nodes in the sight of population similarity.
Although the deep learning method achieves greater performance in the prediction work, it is also limited by some weaknesses. One of them exists in the prediction window design. The partition of time series data provides us more flexibility but also arouses some trouble. The data quality is fluctuating during the epidemic process due to multiple reasons like error cases counting due to unfamiliar about the symptom of the disease. Therefore, it could happen that inside some time section the data has low quality, and thus influencing the quality of the prediction work. Moreover, if some emergency policies are announced or significant events take place, resulting in sudden change in the cases number, the prediction inside such window may output less meaningful result.
We can also improve the design for loss function to add more constraints and include more situations in reality. Like what we have referred, different policies lead to change in transmission rate and recovery rate. We generally use MSE to optimize the learning of number of cases and parameters in SIR model. It is possible to add more loss representations for other interpretations.

Conclusion

In this project report, our proposed HFSSG model is able to make reasonable prediction over traditional model such as SIR and SEIR, which is shown in final values for MSE comparison. Despite the SIR model performs seemingly better with the parameters trained with machine learning, our model is expected to perform better with a longer period of time, since the MLP could aggregate the values from the neighbors and utilize the temporal aspect of our data. Our abstraction of edges assumed that there is a communication between the nodes, which may represent the fact that the flow of people between different states is not being canceled. Clearly due to the quarantine, the edges are shut down and will definitely modify the prediction results like the one presented here. In the future, we plan to tune the hyper-parameters given the prediction results with the ground truth, and we can identify more representative descriptors for the spread of COVID-19.

Appendix

Visualization

Featurization of sexuality

```python
import csv
import matplotlib.pyplot as plt
import datetime
import numpy as np

plt.style.use('ggplot')

age_data = './boroughs-by-sex.csv'
txt = []

with open(age_data, 'r', newline='') as file:
    reader = csv.reader(file)
    for line in reader:
        txt.append(line)

header = txt[0]

fig,ax = plt.subplots(figsize=(12.5,7))
spacer = -0.25
cii = 0
for plot_indx in range(4,7):
    data_to_plot,x_range = [],[]
    for jj in range(1,len(txt)-1):
        x_range.append(txt[jj][0])
        data_to_plot.append(float(txt[jj][plot_indx]))
    x_plot = np.arange(0,len(x_range))+spacer
    hist =
     ↪  ax.bar(x_plot,data_to_plot,label=header[plot_indx].replace('_','
     ↪  ').title(),width=0.15,color=plt.cm.Set2(cii))
    spacer+=0.25
    cii+=1

ax.set_xticks(np.arange(0,len(x_range)))
ax.set_xticklabels(x_range)
ax.legend(fontsize=16)
ax.tick_params('both',labelsize=16)
# textbox showing the date the data was processed
txtbox = ax.text(0.0, 0.975, 'Data updated at '+datetime.datetime.now().strftime('%b %d, %Y %H:%M'), transform=ax.transAxes, fontsize=14,
        verticalalignment='center', bbox=dict(boxstyle='round', facecolor='w',alpha=0.5))
txtbox.set_x(0.36-(txtbox.figure.bbox.bounds[2]-(txtbox.clipbox.bounds[2]-t
fig.savefig(header[0]+'_in_nyc.png',dpi=300,facecolor='#FCFCFC',bbox_inche
↪ = 'tight')
plt.show()
```

Featurization of different age group

```python
import csv
import matplotlib.pyplot as plt
import datetime
import numpy as np

plt.style.use('ggplot')

age_data = './boroughs-by-age.csv'
txt = []

with open(age_data, 'r', newline='') as file:
    reader = csv.reader(file)
    for line in reader:
        txt.append(line)

header = txt[0]

fig,ax = plt.subplots(figsize=(20,10))
spacer = -0.25
for plot_indx in range(4,7): ## Only focus on BK_CASE_RATE,BK_HOSPITALIZED_RATE,BK_DEATH_RATE
    data_to_plot,x_range = [],[]
    for jj in range(1,len(txt)-1):
        x_range.append(txt[jj][0])
        data_to_plot.append(float(txt[jj][plot_indx]))
    # print(data_to_plot)
    x_plot = np.arange(0,len(x_range))+spacer
    hist =
     ↪  ax.barh(x_plot,data_to_plot,label=header[plot_indx].replace('_',
     ↪  ').title(),height=0.25,log=True)
    spacer+=0.25

ax.set_xlabel('Count per 100,000',fontsize=20)
ax.set_yticks(np.arange(0,len(x_range)))
ax.set_yticklabels(x_range)
ax.legend(fontsize=16)
ax.tick_params('both',labelsize=16)
# fig.suptitle('COVID-19 in NYC by '+header[0].replace('_','
 ↪  ').title(),x=0.4,y=0.92,fontsize=18)

txtbox = ax.text(0.0, 0.975, 'Data updated at
 ↪  '+datetime.datetime.now().strftime('%b %d, %Y %H:%M'),
 ↪  transform=ax.transAxes, fontsize=14,
        verticalalignment='center', bbox=dict(boxstyle='round',
         ↪  facecolor='w',alpha=0.5))
txtbox.set_x(1.04-(txtbox.figure.bbox.bounds[2]-(txtbox.clipbox.bounds[2
fig.savefig(header[0]+'_in_nyc.png',dpi=300,facecolor='#FCFCFC',bbox_incl
 ↪  = 'tight')
plt.show()
```

Graph Structured Representation

```
from torch_geometric.utils.convert import to_networkx
import matplotlib.pyplot as plt
import networkx as nx
from src.graph_loader import make_fc_graph
from pathlib import Path
from src.data_loader import load_pickle
import pickle
from mpl_toolkits.basemap import Basemap as Basemap
import pandas as pd
import matplotlib.lines as mlines

SOURCE_PATH = Path(__file__).parent

def visualize_on_map(data, labels):
        df = pd.read_csv(SOURCE_PATH /
         ↪  '../dataset/lat_lon_usa.csv')

        plt.figure(figsize=(300, 400))
        m = Basemap(
                projection='merc',
                llcrnrlon=-180,
                llcrnrlat=10,
                urcrnrlon=-50,
                urcrnrlat=70,
                lat_ts=0,
                resolution='l',
                suppress_ticks=True)

        data_graph = to_networkx(data)
        pos = {}
        temp_x, temp_y = [], []
        edge_weights = data.edge_attr.numpy()
        degrees = dict(data_graph.degree)
        edges = data_graph.edges
        strong_nodes = []
        weak_nodes = []
        node_strengths = {}
        weightage = 0.5
        for index, edge in enumerate(edges):
                n1, n2 = edge

                if n1 in node_strengths:
                        node_strengths[n1] +=
                         ↪  edge_weights[index]
                else:
                        node_strengths[n1] = edge_weights[index]
                         ↪  + weightage*degrees[n1]
                if n2 in node_strengths:
                        node_strengths[n2] +=
                         ↪  edge_weights[index]
                else:
                        node_strengths[n2] = edge_weights[index]
                         ↪  + weightage*degrees[n2]

        print(degrees)
        print(labels)
        for k, v in node_strengths.items():
                print("State: ", labels[k], " Strength: ", v)
        # The top x nodes get assigned the strong nodes
         ↪  category
        x = 10
        counter = 0
```

```python
    for k, v in sorted(node_strengths.items(), key=lambda item: item[1], reverse=True):
        if counter < x:
            strong_nodes.append(k)
        else:
            weak_nodes.append(k)

        counter += 1

    # Assign cor-ordinates to plot the graph
    for index, node in enumerate(data_graph.nodes):
        state = labels[node]

        lon = df.loc[df['State'] == state, 'lon'].item()
        lat = df.loc[df['State'] == state, 'lat'].item()
        temp_x.append(lon)
        temp_y.append(lat)
    mx, my = m(temp_x, temp_y)

    for index, val in enumerate(data_graph.nodes):
        pos[val] = (mx[index], my[index])

    nx.draw_networkx_nodes(data_graph, pos=pos, nodelist=strong_nodes, node_size=20, node_color='r', alpha=0.8)
    nx.draw_networkx_nodes(data_graph, pos=pos, nodelist=weak_nodes, node_size=15, node_color='b', alpha=0.6)
    nx.draw_networkx_edges(data_graph, pos=pos, edgelist=data_graph.edges, width=1.0, alpha=0.1, arrows=False, edge_color='g')

    nx.draw_networkx_labels(data_graph, pos=pos, labels=labels, font_size=8)

    m.drawcountries(linewidth=3)
    m.drawstates(linewidth=0.2)
    m.drawcoastlines(linewidth=1)
    m.fillcontinents(alpha=0.3)
    line1 = mlines.Line2D(range(1), range(1), color="white", marker='o', markerfacecolor="red")
    line2 = mlines.Line2D(range(1), range(1), color="white", marker='o', markerfacecolor="blue")
```

```
        line3 = mlines.Line2D(range(1), range(1), color="green",
         ↪  marker='', markerfacecolor="green")
        plt.legend((line1, line2, line3), ('Strong nodes', 'Weak
         ↪  nodes', 'Edges'),
                           loc=4, fontsize='xx-large')
        plt.savefig(SOURCE_PATH / '../results/usmap.png')
        plt.show()


def visualize_graph(data, labels, positions=None):
        colors = [i for i in range(17)]*3
        colors = [0.00001*i for i in colors]
        data_graph = to_networkx(data)
        pos = nx.kamada_kawai_layout(data_graph)

        plt.figure(1, figsize=(14, 12))

        nx.draw_networkx_nodes(data_graph, pos=pos,
         ↪  nodelist=data_graph.nodes,
         ↪  alpha=0.1,node_size=700,cmap=plt.cm.jet)
        # nx.draw(data_graph, cmap=plt.get_cmap('Set1'),
         ↪  node_size=75, linewidths=6)
        nx.draw_networkx_edges(data_graph, pos=pos,
         ↪  edgelist=data_graph.edges, width=1.0, alpha=0.5)

        nx.draw_networkx_labels(data_graph, pos=pos,
         ↪  labels=labels, font_size=16)
        # plt.savefig(SOURCE_PATH / '../results/usnode2.png')
        plt.show()


if __name__ == '__main__':
        metadata, data = make_fc_graph()

        with open(SOURCE_PATH /
         ↪  '../dataset/generated/usa/node_to_ss', 'rb') as
         ↪  file:
                node_to_ss = pickle.load(file)

        visualize_graph(data, node_to_ss)
        # visualize_on_map(data, node_to_ss)
        # Load to node to ss dict
```

Traditional SIR Method

```r
library (readr)
library("COVID19")
require(deSolve)
usa_data <- covid19('USA')
par(mfrow=c(1,4))
plot(x = usa_data$date, y = 326687501 - usa_data$deaths -
 ↪  usa_data$confirmed - usa_data$recovered, type = 'l', col =
 ↪  4,xlab = "Time/month", ylab = 'Population', main =
 ↪  'Susceptible')
plot(x = usa_data$date, y = usa_data$confirmed -
 ↪  usa_data$recovered - usa_data$deaths, type = 'l',col = 2,
 ↪  xlab = "Time/month", ylab = 'Population', main = 'Infected')
plot(x = usa_data$date, y = usa_data$recovered, type = 'l',col =
 ↪  3, xlab = "Time/month", ylab = 'Population', main =
 ↪  'Recovered')

plot(x = usa_data$date, y = 326687501 - usa_data$deaths -
 ↪  usa_data$confirmed - usa_data$recovered,
     type = 'l',col = 2, xlab = "Time/month", ylab =
      ↪  'Population', main = 'COVID-19 case',
     ylim = c(0, 3.3*10E7)
     ,yaxs = "i"
     )
lines(x = usa_data$date, y = usa_data$confirmed -
 ↪  usa_data$recovered - usa_data$deaths, col = 4)
lines(x = usa_data$date, y = usa_data$recovered, col = 3)
legend("left", legend=c("Susceptible", "Infected","Recovered"),
       col=c("red", "blue", "green"), lty=1)
```

Training ST-GNN

```python
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from utils import Data2Graph
import scipy.stats
from dgl import DGLGraph
import sklearn
import model

```

```python
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

data_filename='FILEPATH'
auxiliary_filename='uszips.csv'

slot_len = 6
pred_horizon = 60
step_size = 5
mid_time = (pred_horizon//2)+1

nhid1 = 400
nhid2 = 200
gru_dim = 100
num_heads = 3
lr = 1e-3
r = 1

num_epochs = 2000


feat_tensor, Adj, active_cases, confirmed_cases, popn, selected_counties,g = Data2Graph.load_data(data_filename, auxiliary_filename,active_thresh=1000,r=r)
recovered_cases = confirmed_cases - active_cases
state_names=np.array(selected_counties['state'])

d_active = []
d_recovered = []
for i in range(active_cases.size(0)):
  i_active = [0]
  i_recovered = [0]
  for j in range(1,active_cases.size(1)):
    i_active.append(active_cases[i,j]-active_cases[i,j-1])
    i_recovered.append(recovered_cases[i,j]-recovered_cases[i,j-1])
  d_active.append(i_active)
  d_recovered.append(i_recovered)

d_active = torch.tensor(d_active).to(device)
d_recovered = torch.tensor(d_recovered).to(device)


```

```
N,T,D = feat_tensor.shape
nfeat = slot_len*D
nclass = pred_horizon*3
attn_dim=int(nfeat*1.5)


num_pred_time = len(range(slot_len,T))

num_val_pred_time = 60
num_test_pred_time = 5
num_train_pred_time = num_pred_time-num_val_pred_time -
 ↪  num_test_pred_time

test_pred_time=np.arange(T-num_test_pred_time,T)
val_pred_time=np.arange(test_pred_time[0]-num_val_pred_time,test_pred_ti
train_pred_time =
 ↪  np.arange(slot_len,val_pred_time[0]+pred_horizon-1)

train_input_time = np.arange(train_pred_time[-1]+1-pred_horizon)
val_input_time =
 ↪  np.arange(val_pred_time[0]-slot_len,val_pred_time[-1]+1-pred_horizon
test_input_time = np.array([78,79,80,81,82,83])


# train, val, test split along the spatial dimension

num_train = int(1*N)
num_val = int(0*N)
num_test=N-num_train-num_val

#shuffle_order = np.random.permutation(N)
shuffle_order=np.arange(N)

train_locs = np.arange(N)[shuffle_order[:num_train]]
val_locs =
 ↪  np.arange(N)[shuffle_order[num_train:num_train+num_val]]
test_locs = np.arange(N)[shuffle_order[num_train+num_val:]]


# data splitting

W_train=Adj[train_locs,:][:,train_locs].to(device)
train_features=feat_tensor[train_locs,:,:][:,train_input_time,:][:].to(d
```

```python
train_labels_active = active_cases[train_locs,:][:,train_pred_time].to(device)
train_d_active = d_active[train_locs,:][:,train_pred_time].to(device)
train_labels_confirmed = confirmed_cases[train_locs,:][:,train_pred_time].to(device)
train_labels_recovered = recovered_cases[train_locs,:][:,train_pred_time].to(device)
train_d_recovered = d_recovered[train_locs,:][:,train_pred_time].to(device)
train_popn=popn[train_locs,:][:].to(device)


# For validation and testing, we use the full training data points as well

val_set = np.concatenate((train_locs, val_locs),axis=0)

W_val=Adj[val_set,:][:,val_set].to(device)
val_features = feat_tensor[val_set,:,:][:, val_input_time,:][:].to(device)
val_labels_active = active_cases[val_set,:][:,val_pred_time].to(device)
val_d_active = d_active[val_set,:][:,val_pred_time].to(device)
val_labels_confirmed = confirmed_cases[val_set,:][:,val_pred_time].to(device)
val_popn=popn[val_set,:][:].to(device)
val_d_recovered = d_recovered[val_set,:][:,val_pred_time].to(device)

# For testing, we use the full training set as well for graph based inference

test_set = np.concatenate((train_locs, test_locs),axis=0)

W_test=Adj[test_set,:][:,test_set].to(device)
test_features = feat_tensor[test_set, :,:][:,test_input_time].to(device)
test_labels_active = d_active[test_set,:][:,test_pred_time].to(device)
test_d_active = active_cases[test_set,:][:,test_pred_time].to(device)
```

```
test_labels_confirmed =
 ↪  confirmed_cases[test_set,:][:,test_pred_time].to(device)
test_d_recovered =
 ↪  d_recovered[test_set,:][:,test_pred_time].to(device)


for target_loc in range(45):
  print('-----------Start training for loc
   ↪  %d-----------'%target_loc)
  num_train_steps = len(train_input_time)-slot_len+1
  batch_feat = []
  batch_active = []
  batch_recover = []
  batch_I = []
  batch_R = []
  batch_S = []
  batch_It = []
  batch_Rt = []

  for t in range(0, num_train_steps, step_size):
    t_feat =
     ↪  train_features[:,t:t+slot_len,:].view(int(num_train),slot_len*D)

     ↪  t_active=train_d_active[:][target_loc,t:t+pred_horizon].float()
    t_recovered =
     ↪  train_d_recovered[:][target_loc,t:t+pred_horizon].float()
    last_I =
     ↪  active_cases[target_loc,train_pred_time[t]-1].unsqueeze(-1).floa
    last_R =
     ↪  recovered_cases[target_loc,train_pred_time[t]-1].unsqueeze(-1).f
    last_S = popn[target_loc,0].to(device).float().unsqueeze(-1)
     ↪  - last_I - last_R

    batch_It.append(d_active[target_loc,
     ↪  train_pred_time[t]-1].float())
    batch_Rt.append(d_recovered[target_loc,
     ↪  train_pred_time[t]-1].float())

    batch_feat.append(t_feat)
    batch_active.append(t_active)
    batch_recover.append(t_recovered)
    batch_I.append(last_I)
    batch_R.append(last_R)
    batch_S.append(last_S)


  batch_It = torch.stack(batch_It).to(device).squeeze()
  batch_Rt = torch.stack(batch_Rt).to(device).squeeze()
  batch_feat = torch.stack(batch_feat).to(device)
  batch_active = torch.stack(batch_active).to(device)
  batch_recover = torch.stack(batch_recover).to(device)
  batch_I = torch.stack(batch_I).to(device).squeeze()
  batch_R = torch.stack(batch_R).to(device).squeeze()
  batch_S = torch.stack(batch_S).to(device).squeeze()

  valid_feat = []
  valid_active = []
  valid_recover = []
  valid_I = []
  valid_R = []
  valid_S = []
  valid_It = []
  valid_Rt = []

  num_val_steps = val_features.shape[1]-slot_len+1
  for t in range(0, num_val_steps, step_size):
    t_feat =
     ↪  val_features[:,t:t+slot_len,:].view(int(num_train),slot_len*D).floa

     ↪  t_active=val_d_active[:][target_loc,t:t+pred_horizon].float()
    t_recovered =
     ↪  val_d_recovered[:][target_loc,t:t+pred_horizon].float()

    last_I =
     ↪  active_cases[target_loc,val_pred_time[t]-1].unsqueeze(-1).float().t
    last_R =
     ↪  recovered_cases[target_loc,val_pred_time[t]-1].unsqueeze(-1).float
    last_S = popn[target_loc,0].to(device).float().unsqueeze(-1)
     ↪ - last_I - last_R

    valid_It.append(d_active[target_loc,
     ↪  val_pred_time[t]-1].float())
    valid_Rt.append(d_recovered[target_loc,
     ↪  val_pred_time[t]-1].float())

    valid_feat.append(t_feat)
    valid_active.append(t_active)
    valid_recover.append(t_recovered)
    valid_I.append(last_I)
    valid_R.append(last_R)
    valid_S.append(last_S)

  valid_It = torch.stack(valid_It).to(device).flatten()
  valid_Rt = torch.stack(valid_Rt).to(device).flatten()
  valid_feat = torch.stack(valid_feat).to(device)
  valid_active = torch.stack(valid_active).to(device)
  valid_I = torch.stack(valid_I).to(device).flatten()
  valid_R = torch.stack(valid_R).to(device).flatten()
  valid_S = torch.stack(valid_S).to(device).flatten()



  # training the model
  model_name = state_names[target_loc]
  model=model.GAT(g, nfeat, nhid1, nhid2, gru_dim, N, num_heads,
   ↪  pred_horizon,attn_dim).to(device)
  model = model.float()
  optimizer = torch.optim.Adam(model.parameters(), lr=lr)

  criterion = nn.MSELoss(reduction='sum')

  for epoch in range(num_epochs):

    model.train()
    optimizer.zero_grad()

    active_pred, recovered_pred, phy_active, phy_recover =
     ↪  model(batch_feat, popn[target_loc,0].to(device).float(),
     ↪  batch_I, batch_R,batch_S,batch_It,batch_Rt) # forward
     ↪  pass

    loss =
     ↪  F.mse_loss(active_pred,batch_active)+F.mse_loss(recovered_pred,ba
     ↪  batch_active.flatten())+F.mse_loss(phy_recover,
     ↪  batch_recover.flatten())

    loss.backward()
    optimizer.step()

    #Evaluate
    model.eval()
    with torch.no_grad():
      active_val, recovered_val, vphy_active, vphy_recover =
       ↪  model(valid_feat,
       ↪  popn[target_loc,0].to(device).float(), valid_I,
       ↪  valid_R,valid_S,valid_It,valid_Rt) # forward pass
      mse =
       ↪  sklearn.metrics.mean_squared_error(valid_active.cpu().detach().fl
      ccc =
       ↪  model.ccc(valid_active.squeeze().cpu().detach().flatten().numpy()
      r2 =
       ↪  sklearn.metrics.r2_score(valid_active.cpu().detach().flatten().nu
       ↪  active_val.cpu().detach().flatten().numpy())

    if epoch==0:
      min_loss = mse
      min_r2 = r2
      torch.save(model.state_dict(),model_name)
    if mse < min_loss:
      min_loss = mse
      min_r2 = r2
      torch.save(model.state_dict(),model_name)
    if epoch % 100 == 0:
      print("Epoch %d, Train loss %f, Val loss %f, Val R2 %f,
       ↪  Val ccc %f, %.2f, %.2f"%(epoch, loss, mse,
       ↪  np.mean(r2), np.mean(ccc), model.alpha_list,
       ↪  model.beta_list))
```

References

Deng, Songgaojun, Shusen Wang, Huzefa Rangwala, Lijing Wang **and** Yue Ning. **december** 2019. Graph Message Passing with Cross-location Attentions for Long-term ILI Prediction ().

Guidotti **and** Ardia. 2020. Covid-19 Data Hub. Available at <https://covid19datahub.io/>.

Kapoor, Amol, Xue Ben, Luyang Liu, Bryan Perozzi, Matt Barnes, Martin Blais **and** Shawn O'Banion. 2020. Examining COVID-19 Forecasting using Spatio-Temporal Graph Neural Networks. arXiv: 2007.03113 [cs.LG].

Lisphilar. 2020. CPython package for COVID-19 analysis with SIR-derived ODE models. Available at <https://github.com/lisphilar/covid19-sir>.

Pei, Sen, **and** Jeffrey Shaman. 2020. Initial Simulation of SARS-CoV2 Spread and Intervention Effects in the Continental US. *medRxiv,* https://doi.org/10.1101/2020.03.21.20040303. eprint: https://www.medrxiv.org/content/early/2020/03/27/2020.03.21.20040303.full.pdf. Available at <https://www.medrxiv.org/content/early/2020/03/27/2020.03.21.20040303>.

Xingjie Hao, et al. 2020. Reconstruction of the full transmission dynamics of COVID-19 in Wuhan. *Nature* 584:420–424.

Yang, Zifeng, Zhiqi Zeng, Ke Wang, Sook-San Wong, Wenhua Liang, Mark Zanin, Peng Liu **andothers**. 2020. Modified SEIR and AI prediction of the epidemics trend of COVID-19 in China under public health interventions. *Journal of Thoracic Disease* 12 (3). ISSN: 2077-6624. Available at <http://jtd.amegroups.com/article/view/36385>.

Zheng, Yunling, Zhijian Li, Jack Xin **and** Guofa Zhou. 2020. A Spatial-Temporal Graph Based Hybrid Infectious Disease Model with Application to COVID-19. *arXiv preprint arXiv:2010.09077.*