A Study on Graph-Structured Recurrent Neural Networks and Sparsification with Application to Epidemic Forecasting

Zhijian Li[1], Xiyang Luo[2], Bao Wang[2], Andrea L. Bertozzi[2], and Jack Xin[1]

[1] UC Irvine zhijil2@uci.edu, jxin@math.uci.edu
[2] UCLA xylmath@gmail.com, wangbaonj@gmail.com, bertozzi@math.ucla.edu

Abstract. We study epidemic forecasting on real-world health data by a graph-structured recurrent neural network (GSRNN). We achieve state-of-the-art forecasting accuracy on the benchmark CDC dataset. To improve model efficiency, we sparsify the network weights via a transformed-ℓ_1 penalty without losing prediction accuracy.

1 Introduction

Epidemic forecasting has been studied for decades [8]. Many statistical and machine learning methods have been successfully used to detect epidemic outbreaks [5]. In previous works, epidemic forecasting is mainly considered as a time-series problem. Time-series methods, such as Auto-Regression (AR), Long Short-term Memory (LSTM) neural networks and their variants have been applied to this problem. One of the current directions is to use social media data [9]. In 2008, Google launched Google Flu Trend, a digital service to predict influenza outbreaks using Google search data. The Google algorithm was discontinued due to flaws, however Yang et al. [13] designed another algorithm ARGO in 2015 also using Google search pattern data. Google Correlate, a collection of time-series data of Google search trends, plays a vital role in this refined regression algorithm. Though ARGO succeeded in accuracy as a time series algorithm, it lacks spatial structure and requires the additional input of external features (e.g., social media data). The infectious and spreading nature of the epidemics suggests that forecasting is also a spatial problem. Here we study a model to take advantage of the spatial information so that the data from the adjacent regions can introduce regional spatial features. This way, we minimize external data input and the accompanying computational cost. Structured recurrent neural network (SRNN) is a model for the spatial-temporal problem first adopted by A. Jain et al. [4] for motion forecasting in computer vision. Wang et al. [11,12,10] successfully adapted SRNN to forecast real-time crime activities. Motivated by [4,11,10], we present an SRNN model to forecast epidemic activity levels. We test our model with data provided by the Center for Disease Control (CDC), which collects data from approximately 100 public and 300 private laboratories in the US [1]. The CDC data [1] is a well-established authoritative data set widely used by researchers, which makes it easy for us to compare our model with previous

work. CDC provides the influenza data by the geography of Health and Human Services regions (HHS regions). We take the geographic structure of ten HHS regions as our spatial information. The rest of the paper is organized as follows. In section 2, we overview RNN. In sections 3-5, we present a graph-structured RNN model, graph description of spatial correlations, and sparsity promoting penalties. Experimental results and concluding remarks are in sections 6 and 7.

2 A Short Review of Recurrent Neural Network

Recurrent Neural Network (RNN) is a neural network designed for sequential data. It is widely used in natural language processing (NLP) and time-series analysis because of its capability of processing sequential information. The idea of RNN comes from unfolding a recursive computation for a chain of states. If we have a chain of states, in which each state depends on the last steps: $s^n = f(s^{n-1})$, for some function f. Then we can unfold this equation to: $s^n = f(f(...f(s^0)))$. Suppose we have a sequential data $x^1, x^2, ..., x^n$, the idea of RNN is to unfold the $x^n = f(x^{n-1}, \theta)$ to a computational graph. An unfolded RNN is illustrated in Fig. 1 and given by the recursion:

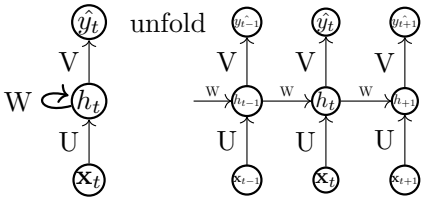


Fig. 1: an unfolded recurrent neural network.

$$h_t = \tanh(b + W\, h^{t-1} + U\, x^t), \;\; y_t = \tanh(V\, h_t + c),$$

where tanh is the activation function; (U, V, W) are the weight matrices; b, c are bias vectors. Given y_i^t the true signal at time t, the popular loss function for classification task is the cross-entropy loss, which reads in the binary case: $\mathcal{L}(\theta) = -\sum_t y_t \cdot \ln(\hat{y_t}) + (1-y)\ln(1-\hat{y})$. The mean-square-error loss is widely used for regression problem: $\mathcal{L}(\theta) = \sum_t (y_t - \hat{y_t})^2$. Then, as in most neural networks, RNN is trained by stochastic gradient descent. A major issue of RNN is the problem of exploding and vanishing gradients. Since

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L_t(y^t, \hat{y^t})}{\partial W}, \;\; \frac{\partial L_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial L_n}{\partial \hat{y^t}} \frac{\partial \hat{y^t}}{\partial s_t} \frac{\partial s_t}{\partial s_k} \frac{\partial s_k}{\partial W}, \;\; s_t = \tanh(Ux^t + Wh^{t-1}),$$

$$\text{then} \quad \frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y^t}} \frac{\partial \hat{y^t}}{\partial s_t} \Big(\prod_{j=k+1}^{t} \frac{\partial s_j}{\partial s_{j-1}} \Big) \frac{\partial s_k}{\partial W}.$$

It is well-known [7] that

$$||\frac{\partial \hat{y^t}}{\partial s_t}(\prod_{i=k}^{t-1}\frac{\partial s_{i+1}}{\partial s_i})|| \leq \eta^{t-k}||\frac{\partial s_t}{\partial \hat{y^t}}||$$

where $\eta < 1$ under the assumption of no bias is used and the spectral norm of W being less than 1. We see that the gradient vanishes exponentially fast in large t. Hence, the RNN is learning less and less as time goes by. LSTM [3], is a special kind of RNN that resolves this problem.

$$\begin{pmatrix} f_t \\ i_t \\ o^t \\ \tilde{C}_t \end{pmatrix} = \begin{pmatrix} \sigma(W[h_{t-1,x_t}]) \\ \sigma(W[h_{t-1,x_t}]) \\ \sigma(W[h_{t-1,x_t}]) \\ \tanh(W[h_{t-1,x_t}]) \end{pmatrix} + \begin{pmatrix} b_f \\ b_i \\ b_o \\ b_C \end{pmatrix}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t, \quad h_t = o_t * \tanh(C_t).$$

Since one does not directly apply the same recurrent function to h_t every time step in the gradient flow, there is no intrinsic factor η in $\frac{\partial L_t}{\partial W}$. This way the gradient has much less chance to vanish as time goes by. In our model, we use LSTM for all RNNs.

3 Graph-Strutured RNN model

Similar to previous work of structured RNN, we partition the nodes into different classes, and for each class we join the nodes in the class. We compare the level of activity of nodes by summing up the data of each node. Then, we partition the nodes based on their activity level, from the class that has highest activity level to the lowest one. After some experiments, we find that SRNN works the best when we have two classes (see Fig. 2). We denote the class with relatively high activity level H, and the other class L. After some experiments, we classify the nodes based on following criteria: Let G be a weighted graph with nodes indexed by $Z = \{1, \ldots, N\}$, and edge weights $w_{ij} \geq 0$. Let $g : Z \mapsto \{1, \ldots, C\}$ be the function that assigns each node to its corresponding group, and assume for simplicity that g is a surjection. Let us define:

$$|v| = \text{sum of the historical activity level of node } v,$$

$$M = \arg\max_v |v|, \quad m = \arg\min_v |v|, \quad g(v) = \lfloor \frac{|v| - m}{M - m + 10^{-6}} \rfloor.$$

In our model, nodes with label 0 are in the relatively inactive class, nodes with label 1 or higher belong to another class, the relatively active class.

We define an RNN $E_{i,j}$ for each connected edge $w_{ij} \neq 0$. We denote $E_{i,j}$ as the *edge* RNN since it models the pairwise interaction between two connected nodes. We enforce weight sharing among two edge RNNs, $\text{RNN}_{E_{i',j'}}$ and $\text{RNN}_{E_{i,j}}$, if $g(i) = g(i')$, $g(j) = g(j')$, i.e., if the class assignments of the two node pairs are

the same. Similarly, we define an RNN N_i, for each node in Z, which we denote as a *node* RNN, and apply weight sharing if $g(i') = g(i)$. Even though the RNNs share weights, there state vector are still different, and thus we denote them with distinct indices.

Let $\{v_i^t, i \in 1 \dots N\}$ be the set of node features at time t. The GSRNN makes a prediction at node i, time t by first feeding neighboring features to its respective edge RNN, and then feeding the averaged output along with the node features to the respective node RNN. Namely,

$$f_i^t = \sum_j w_{ij} \text{RNN}_{E_{i,j}}(v_i^t, \alpha_{ij} v_j^t), \quad \hat{y}_i^t = \text{RNN}_{N_i}(v_i^t, f_i^t). \tag{1}$$

Let y_i^t be the true signal at time t. We use the mean square loss function below:

$$L^t(\Theta) = \frac{1}{N} \sum_i (\hat{y}_i^t - y_i^t)^2. \tag{2}$$

We back-propagate through time (BPTT), a standard method for training RNNs, with the understanding that the weights for edge RNNs and node RNNs are shared according to the description above.

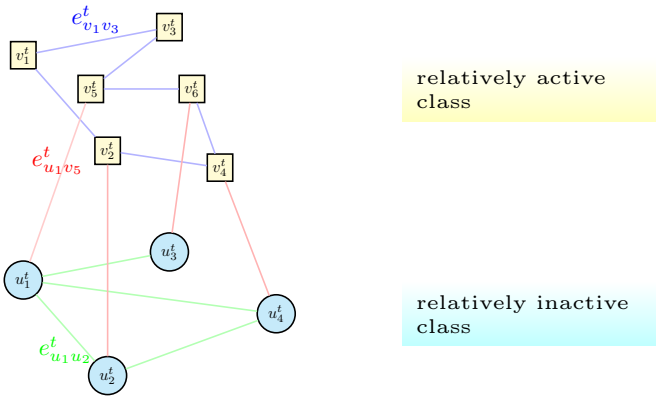


Fig. 2: red edges are of type H-L, green edges are of type L-L, and blue edges are of type H-H.

In our model of $C = 2$, we have three types of edges, H-H, L-L, and H-L. The H-H is the type of edge between two nodes in class H, L-L is the type of edge between two nodes of class L, and H-L is the type of edge edge a node of class H and a node of class L. Each type of edge features will be fed into a different RNN. We normalize our edge weight by maximum degree. Each edge has weight

$\alpha_{ij} = w_{ij} = \frac{1}{M_e}, \forall\, i$ and j, where M_e is the maximum degree over the ten nodes. We use a look-back window of two to generate training data for RNN: the node feature of v^t contains the information of node v at $t-1$ and $t-2$. Then, the edge features of a node $v \in H$ with the edges E_v are:

$$e^t_{v,H} = \left[\frac{v^t_1}{M_e}, \frac{v^t_2}{M_e} \cdots\right]$$

for all $v_i \in H$ such that $(v, v_i) \in E_v$,

$$e^t_{v,L} = \left[\frac{u^t_1}{M_e}, \frac{u^t_2}{M_e} \cdots\right]$$

for all $u_i \in L$ such that $(v, u_i) \in E_v$. We feed $e^t_{v,H}$ and $e^t_{v,L}$ into the corresponding edgeRNNs:

$$f^t = \frac{1}{M_e}\text{edgeRNN}_{H-L}(v^t, e^t_{v,L}), \quad h^t_v = \frac{1}{M_e}\text{edgeRNN}_{H-H}(v^t, e^t_{v,H}).$$

Each edge RNN will jointly train all the nodes that have an edge belong to its type:

$$\arg\min_{\theta} \mathcal{L}_{H-L}(\Theta) = \frac{1}{|N_w|}\sum_{w \in N_w}\sum_{t}(y_w{}^t - \hat{y}^t_w)^2$$

where $N_w = \{w \in H \cup L|\ \ E_w \text{ contains an element of type L-H}\}$.

$$\arg\min_{\theta} \mathcal{L}_{H-H}(\Theta) = \frac{1}{|N_v|}\sum_{v \in N_v}\sum_{t}(y_v{}^t - \hat{y}^t_v)^2$$

where $N_v = \{v \in H|\ \ E_v \text{ contains an element of type H-H}\}$.
Finally, we have a node RNN that jointly trains all the nodes in this class:

$$\arg\min_{\theta} \mathcal{L}_{H}(\Theta) = \frac{1}{|H|}\sum_{v \in H}\sum_{t}(y_v{}^t - \hat{y}^t_v)^2, \quad \forall v \in H.$$

We feed the outputs of two edge RNNs, together the node feature of v itself into nodeRNN_H:

$$v^{t+1} = \text{nodeRNN}_H(v^t, f^t, h^t).$$

4 Graph Description of Spatial Correlation

The graph is a flexible representation for irregular geographical shapes which is especially useful for many spatio-temporal forecasting problems. In this work, we use a weighted directed graph for space description where each node corresponds to a state. There are multiple ways to infer the connectivity and weights of this weighted directed graph. In the previous work [10], Wang et al. utilized a

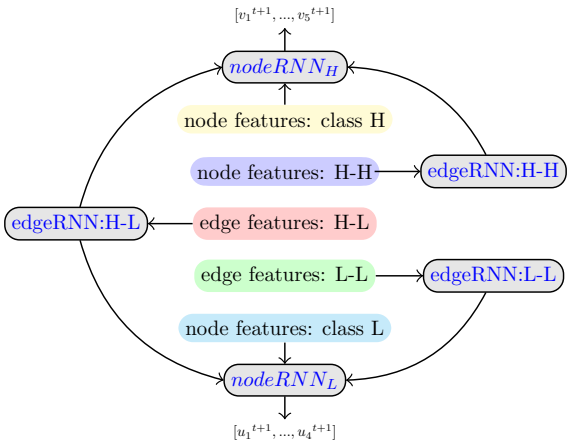


Fig. 3: Edge features of the same type are jointly trained by one edge RNN. Nodes from the same class are jointly trained by one node RNN.

multivariate Hawkes process to infer such a graph for crime and traffic forecasting, where the connectivity and weight indicate the mutual influence between the source and the sink nodes. Alternatively, one can opt for space closeness and connect the closest few nodes on the graph, and the weight is proportional to the historical moving average activity levels of the source node. In this work, we employ the second strategy, where we regard two nodes as connected if the corresponding two states are geographically adjacent to each other. The graph in this work is demonstrated in Fig. 4. We will explore the first strategy in the future.

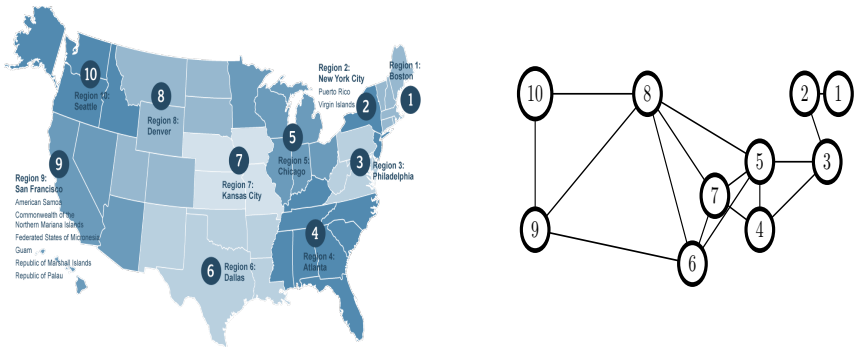
5 Sparsity Promoting Penalties

The convex sparsity promoting penalty is the ℓ_1 norm. In this study, we also employ a Lipschitz continuous non-convex penalty, the so-called transformed-ℓ_1.

Definition 1. *The transformed ℓ_1 ($T\ell_1$) penalty function on $x = (x_1, \cdots, x_d) \in \mathbb{R}^d$ is*

$$P_a(x) := \sum_{i=1}^{d} \rho_a(x_i), \quad \rho_a(x_i) = \frac{(a+1)|x_i|}{a+|x_i|}, \quad \text{parameter } a \in (0, +\infty). \tag{3}$$

Since $\lim_{a\to 0^+} \rho_a(x_i) = 1_{\{x_i \neq 0\}}$, $\lim_{a\to+\infty} \rho_a(x_i) = |x_i|$, $\forall i$, the Tℓ_1 penalty interpolates ℓ_1 and ℓ_0. For its sparsification in compressed sensing and other applications, see [14] and references therein. To sparsify weights in GSRNN training via ℓ_1 and Tℓ_1, we add them to the loss function of GSRNN with a multiplicative penalty parameter $\alpha > 0$, and call stochastic gradient descent optimizer on



(a) HHS regions on map (b) Graph Structure

Fig. 4: HHS Graph

Tensorflow. Though a relaxed splitting method [2] can enforce sparsity much faster, we shall leave this as part of future work on ℓ_0 penalty.

6 Experimental Results

Among the previous works on influenza forecasting, ARGO [13] is the current state-of-the-art prediction model for the entire U.S. influenza activity. To compare with previous works conveniently, we use the CDC data from 2013 to 2015 as our test data. The accuracy is measured in: RMSE=$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}$.

We use a single layer LSTM with 40 hidden units for edge RNNs, and a three-layer multilayer LSTM with hidden units $[10, 40, 10]$ for node RNNs. We use the Adam optimizer to train GSRNN. The RMSE of the forecasting from 2013/1/19 to 2015/8/15, 135 weeks in total, is shown in Table. 1. We outperform LSTM and Autoregressive Model of order 3 (AR(3)) in all nodes, and ARGO in 8 nodes, see Fig. 5 for activity plots in each region. It is easy to see that in regions 1, 2, 7 and 8, there are some under-predictions, while GSRNN's prediction is almost identical to the ground-truth. The general form of an AR(p) model for time-series data is

$$X_t = \mu + \sum_{i=1}^{p} \phi_i \, X_{t-i} + \epsilon,$$

where $\boldsymbol{\phi} = (\phi_1, ..., \phi_p)$ is computed through the backshift operator. ARGO[13], as a refined autoregressive model, models the flu activity level as:

$$\hat{y}_t = \mu_y + \sum_{j=1}^{52} \alpha_j \, y_{t-j} + \sum_{i=1}^{100} \beta_i \, X_{i,t} + \epsilon_t, \quad [\mu_y, \boldsymbol{\alpha}, \boldsymbol{\beta}] := \underset{\mu_y, \boldsymbol{\alpha}, \boldsymbol{\beta}}{\arg\min} \sum_t (y_t - \hat{y}_t)^2,$$

ϵ_t being i.i.d Gaussian noise, $X_{i,t}$ the log-transformed Google search frequency of term i at time t.

We observe that ARGO has inconsistent performance over nodes. We believe this is because the external feature of ARGO, the Google search pattern data, does not offer useful information, since the national search pattern does not necessarily apply to a certain HHS region. Meanwhile, we also have much less computational cost than ARGO, which takes in top 100 search terms related to influenza as well as their historical activity levels, with a look-back window length of 52 weeks. During the time for ARGO to compute one node, our model finishes all the ten nodes.

We sparsify the network through ℓ_1 and Tℓ_1 (eq. (3) using $a = 1$ and penalty parameter $\alpha = 10^{-8}$ during training. Post training, we hard threshold small network weights to 0 at threshold 10^{-3}, and find that high sparsity under Tℓ_1 regularization is achieved while maintaining the accuracy at the same level, see Table 2 and Table 3. Hard-thresholding improves the predictions for some nodes but not all of them, however it reduces the inference latency and is thus beneficial for the overall algorithm.

Table 1: The RMSE between the predicted and ground-truth activity levels by different methods over 10 different states.

Node	1	2	3	4	5	6	7	8	9	10
AR(3)	0.242	0.383	0.481	0.415	0.345	0.797	0.401	0.305	0.356	0.317
ARGO	0.281	0.379	0.397	0.335	**0.285**	0.673	0.449	**0.244**	0.356	0.310
LSTM	0.271	0.364	0.487	0.349	0.328	0.751	0.421	0.333	0.335	0.310
GSRNN	**0.223**	**0.354**	**0.374**	**0.320**	0.289	**0.664**	**0.361**	0.275	**0.284**	**0.303**

Table 2: Percentages of weights $< 10^{-3}$ in absolute value in GSRNN w/ and w/o ℓ_1, Tℓ_1 penalties.

Penalty	1	2	3	4	5
$\alpha = 0$	51.2%	47.8%	50.3%	50.6%	49.9%
$l_1(\alpha = 5 \cdot 10^{-8})$	67.7%	51.8%	57.7%	60.7%	61.2%
$TL1(\alpha = 5 \cdot 10^{-8})$	**82.3**%	**58.9**%	**71.9**%	**64.2**%	**71.1**%

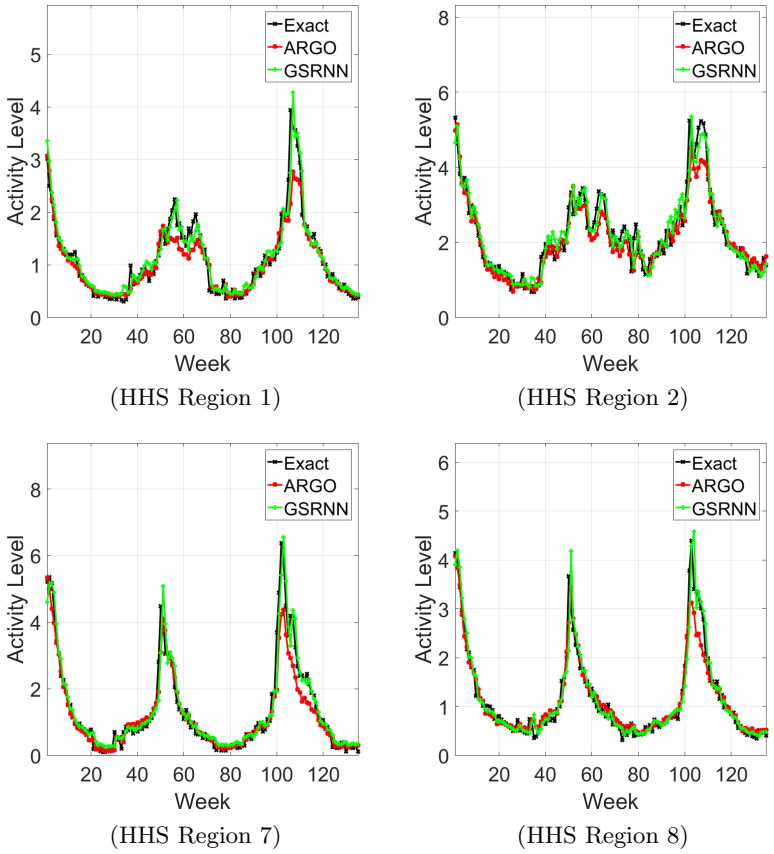


Fig. 5: The exact and predicted flu activity levels by GSRNN and ARGO.

7 Concluding Remarks

We studied epidemic forecasting based on a graph-structured RNN model to take into account geo-spatial information. We also sparsified the model and reduced 70% of the network weights to zero while maintaining the same level of prediction accuracy. In future work, we plan to explore wider neighborhood interactions and more powerful sparsification methods. Furthermore, training RNNs with the recently developed Laplacian smoothing gradient descent [6] is worth exploring.

Acknowledgments

This material is based on research sponsored by the Air Force Research Laboratory and DARPA under agreement number FA8750-18-2-0066; the U.S. Department of Energy, Office of Science, doe-sc0013838; the National Science Founda-

Table 3: Node-wise RMSE of GSRNNs via post-training hard thresholding at threshold 10^{-3}.

Node	1	2	3	4	5	6	7	8	9	10
$\alpha = 0$	0.230	0.351	0.390	0.334	0.314	0.676	0.380	0.297	0.287	0.316
$l_1(\alpha = 5 \cdot 10^{-8})$	0.234	0.351	0.388	0.327	0.306	0.685	0.363	0.290	0.281	0.296
$TL1(\alpha = 5 \cdot 10^{-8})$	0.225	0.363	0.379	0.328	0.296	0.690	0.365	0.272	0.311	0.305

tion DMS-1554564 (STROBE), DMS-1737770, DMS-1522383, IIS-1632935. The authors thank Profs. M. Hyman, and J. Lega for helpful discussions.

References

1. CDC data: gis.cdc.gov/grasp/fluview/fluportaldashboard.html.
2. T. Dinh and J. Xin. Convergence of a relaxed variable splitting method for learning sparse neural networks via ℓ_1, ℓ_0, and transformed-ℓ_1 penalties. *ArXiv: 1812.05719*, 2018.
3. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
4. A. Jain, A. Zamir, S. Savarese, and A. Saxena. Structural-RNN: Deep learning on spatio-temporal graphs. In *Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, 2016.
5. E. Nsoesie, J. Brownstein, N. Ramakrishnan, and M. Marathe. A systematic review of studies on forecasting the dynamics of influenza outbreaks. *Influenza and other Respiratory Viruses*, 8(3):309–316, 2014.
6. S. Osher, B. Wang, P. Yin, X. Luo, M. Pham, and A. Lin. Laplacian smoothing gradient descent. *arXiv:1806.06317*, 2018.
7. R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proc. of the 30th International Conference on Machine Learning*, 2013.
8. N. Perra and B. Goncalves. Modeling and predicting human infectious disease. *Social Phenomena, Springer*, pages 59–83, 2015.
9. S. Volkova, E. Ayton, K. Porterfield, and C. Corley. Forecasting influenza-like illness dynamics for military populations using neural networks and social media. *PLOS one*, 12(12):e0188941, 2017.
10. B. Wang, X. Luo, F. Zhang, B. Yuan, A. Bertozzi, and P. Brantingham. Graph-based deep modeling and real time forecasting of sparse spatio-temporal data. *arXiv:1804.00684*, 2018.
11. B. Wang, P. Yin, A. Bertozzi, P. Brantingham, S. Osher, and J. Xin. Deep learning for real-time crime forecasting and its ternarization. *arXiv:1711.08833*, 2017.
12. B. Wang, D. Zhang, D. Zhang, P. Brantingham, and A. Bertozzi. Deep learning for real-time crime forecasting. *arXiv:1707.03340*, 2017.
13. S. Yang, M. Santillana, and S. Kou. Accurate estimation of influenza epidemics using google search data via ARGO. *Proceedings of the National Academy of Sciences*, 112(47), 2015.
14. S. Zhang and J. Xin. Minimization of transformed ℓ_1 penalty: Theory, difference of convex function algorithm, and robust application in compressed sensing. *Mathematical Programming, Series B*, 169(1):307–336, 2018.