ViEWS Competition: Adding Event Data and Spatio-Temporal Graph Convolutional Networks*

Patrick T. Brandt[1], Vito J. D'Orazio[1], Latifur Khan[2], Yi-Fan Li[2], Javier Osorio[3], Marcus Sianan[1]

[1]School of Economic, Political, and Policy Sciences, University of Texas, Dallas
[2]Erik Jonsson School of Engineering and Computer Science, University of Texas, Dallas
[3]School of Government and Public Policy, University of Arizona

September 29, 2020

1 Introduction

Our approach to improving the prediction of PRIO-GRID objects of interest is to focus on three major components: the input data for the model, the models themselves, and the forecast metrics of interest.

In addition to the standard variables in the existing ViEWS fatality model we add monthly ICEWS event data (Boschee et al., 2015) for the last 20 plus years. While some research has shown that ICEWS does not always lead to improvements in predictive performance (Chiba and Gleditsch, 2017), others have shown successful applications of the ICEWS data for forecasting civil conflict when the features constructed from the raw event data are theoretically informed (Blair and Sambanis, 2020). Here, we use simple CAMEO event counts and rely on the flexibility of our modeling technique to capture the important relations between civil conflict and ICEWS events.

Conflict forecasting models exhibit many non-linear relationships as well as spatial and temporal dependencies (D'Orazio, 2020). Researchers have employed different machine learning and regression techniques to account for these attributes (Beck et al., 2000; Hill Jr and Jones, 2014; Cranmer and Desmarais, 2017). Yet, there is no consensus on the best modeling approach, and researchers are constantly innovating to improve predictive performance (Dorff et al., 2020). The models we select are spatio-temporal graph convolutional neural networks (ST-GCN), which is highly flexible and can account for spatial and temporal dependencies.

Finally, our third approach to improve predictive models of conflict is to employ the continuous rank probability score (CRPS) (Hersbach, 2000). The CRPS is a proper scoring metric that accounts for the distribution of the prediction, not only the point, as is the case with mean squared error (Brandt et al., 2014). This metric can improve model selection, as accounting for the full distribution can help researchers to avoid cases where the MSE is low but predictions exhibit high variance.

2 Proposed Models and Components

Fatality prediction for the ViEWS project can be analyzed as a typical time-series forecasting problem, and it aims to predict future length-Q number of fatalities based on previous P observations so we then map our training data

*This research is supported by the National Science Foundation, award number OAC-1931541. The authors are responsible for the paper's contents

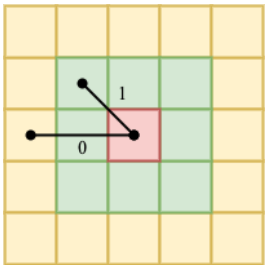

Figure 1: Demonstration adjacency graph connections. In the adjacency matrix, the edge weights between the red cell and green cells are **1**, for a **direct connections**. The edge weights between the red cell and yellow cells are **0**, indicating **no direct connection**.

x_t to forecasted values y_t using some forecast function $f(\cdot)$:

$$[x_{(t-P+1)}, \cdots, x_{(t)}] \xrightarrow{f(\cdot)} [y_{(t+1)}, \cdots, y_{(t+Q)}] \tag{1}$$

We focus on predicting the change in log fatalities at the PRIO-GRID month level (pgm). Given the geographical connections between these PRIO-GRID cells, the grid map can be defined as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, where $\mathcal{V}$ is a set of cells with $|\mathcal{V}| = N$; $\mathcal{E}$ is a set of edges representing the connectivity between cells, and; $\mathbf{A} \in \mathbb{R}^{N \times N}$ denotes the adjacency matrix of graph $\mathcal{G}$. Notably, the overall raw inputs to our model are denoted by $X \in \mathbb{R}^{P \times N \times d}$ where d is the dimension of the data. Here the solution for the fatalities prediction considers both spatial and temporal dependencies within the existing data. We split this into two sets of convolutional neural networks over space and time and combine them together with a layering technique.

2.1 Spatial Dependency Modeling

The Graph Convolutional Network (GCN) is an effective model when mining graph structured data, e.g., social networks, online advertising, etc. (Kipf and Welling, 2017). Define the map of Africa as an undirected graph, such that the nodes of graph are the PRIO-GRID cells. For a given cell, it is connected to its 8 surrounding cells, but is not directly connected to all other cells. This relationship can be referred to as an *adjacency matrix* as illustrated in Figure 1. The GCN module handles the spatial dependencies by passing the information to each cell from its surrounding cells using a weighted average (convolutions) of their features.

Following Kipf and Welling (2017), the spatial dependency model for each time step $t \leq P$ has two inputs in this GCN module. The first one is a variable $x_{sp} \in \mathbb{R}^{N \times d_0}$, where d_0 is the dimensions of features for each PRIO-GRID cell. The second put is the adjacency matrix $\mathbf{A}$ of the graph as in Figure 1.

For each time step $t \leq P$, the layer-wise propagation rule of the GCN relationships $H^{(l+1)}$ for layer l of the neural network is :

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{2}$$

where $H^{(l)} \in \mathbb{R}^{n \times d}$ is the matrix of activations in the l^{th} layer with $H^{(0)} = X$. With I_N an identity matrix with dimension of n, the $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected PRIO-GRID graph $\mathcal{G}$ with self-connections. Here $\tilde{D}$ is the degree matrix which can be represented as $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. $W^{(l)} \in \mathbb{R}^{d \times d'}$ is a layer-specific trainable weight matrix. d and d' are the number of feature dimensions for input and out respectively at the l^{th} layer. Finally, $\sigma(\cdot) = \mathrm{ReLU}(\cdot) = \max(0, \cdot)$ is an activation function that maps across the neural network layers. We choose this because it is an effective activation function in deep learning that allows non-linearity in the neural network.

Equation 2 implies a two-step propagation rule of GCN network: (1) aggregate the information from each cell's neighbors; (2) update the neural network. The $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}$ term in Equation 2 represents the graph convolution, which takes the graph connections into consideration, passes neighbor information to each cell, and

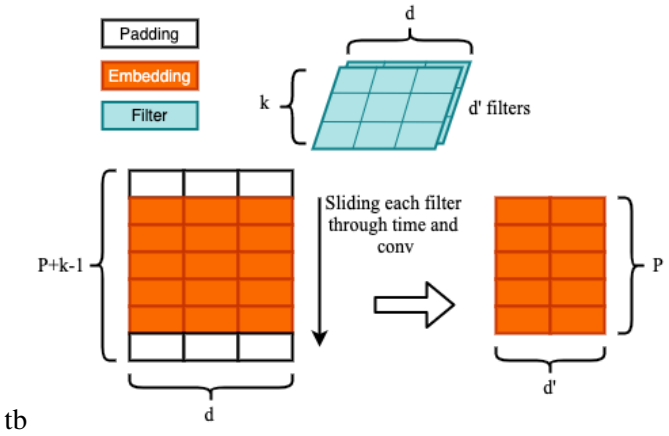

Figure 2: Demonstration of 1D Convolution. By applying tricks such as padding, we keep the time steps of the outputs the same as inputs. The feature dimension of outputs is determined by the number of filters.

aggregate them based on the connection weights. The aggregated matrix is put into a standard neural network, which essentially multiplies it by the trainable weight matrix $W^{(l)}$, and feeds it to the activation function $\sigma(\cdot)$.

2.2 Temporal Dependency Modeling

For the temporal aspects of the model, Temporal Convolution Networks (TCN) and Recurrent Neural Networks (RNN) are two main deep learning approaches (Lim and Zohren, 2020). Here we develop temporal models based on both of the spatial and temporal blocks separately. The temporal inputs for each node are $x_{tp} \in \mathbb{R}^{P \times d}$. The TCN, which is the method selected for temporal modeling, is described in an appendix. We leave demonstrations of long-short term model (LSTM) to the supplemental materials.

The basic idea of TCN-based models is that temporal information is aggregated by learnable convolution filters – like moving averages (Yu et al., 2018). Fig. 2 shows the basic idea of one dimensional temporal convolution with a kernel of size k. Basically this operation allows the model to explore the temporal dependency of each node for k months at a time. The convolutional kernel K_t is designed to map the input x_{tp} to a embedding. In order to keep time steps embeddings consistent along time as P, we pad zero to both beginning and end of the window so it stays the same size.

Thus the dimension of embedding after this convolution step is $P \times d'$. Then, an activation function is applied to this embedding and this whole process can be abstracted as:

$$H^{(l+1)} = \sigma\left(\text{conv}(H^{(l)}, W^{(l)})\right) \tag{3}$$

where conv denotes the convolution operation, $W^{(l)}$ is the trainable weight representing convolutional kernel here with dimension of $k \times d \times d'$.

2.3 Combined Spatio-Temporal Model

We apply two types of networks to model both the spatial and temporal dependencies. Both TCN and LSTM are explored to model the temporal dependencies. The overall model parameters are shown in Table 1. Similar to last section, we start with the full STGCN-TCN model here and leave the descriptions of STGCN-LSTM to the supplemental materials. Also, we include demonstrations of the basics of STGCN-TCN and STGCN-LSTM models in the supplemental materials.

Overall, the STGCN-TCN model is composed of a stacked modules, and a Fully Connected (FC) layer at the end to generate predictions. We stack three identical STGCN modules altogether to jointly process graph-structured time series. Note it is possible to stack more STGCN modules depending on the requirement of the

Parameter	Values	Meaning
P	54	Number of past steps used for generating predictions
Q	6	Number of future steps for forecasting
d	Variable	Input feature dimension for each instance
d'	Variable	Output feature dimension for each instance
S_{tcn}	3	Number of stacks for STGCN modules in STGCN-TCN
S_{lstm}	3	Number of stacks for GCN modules in STGCN-LSTM
k_{tp}	3	Kernel size of temporal convolution

Table 1: Notations and values of parameters.

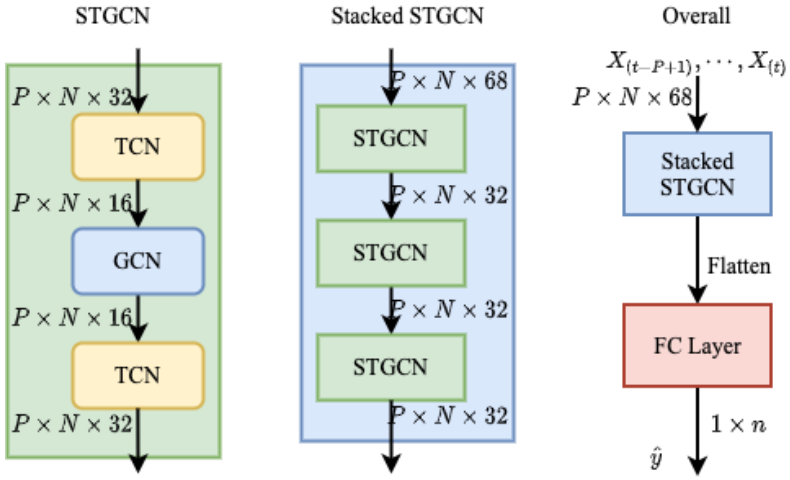

Figure 3: Architecture of STGCN-TCN. This framework consists three STGCN blocks with a fully-connected (FC) layer by the end to generate the forecast outputs.

task and the input dimension for the first STGCN block is different from latter ones as the input features here are the raw data with $d = 68$.

Figure 3 shows, in each STGCN block, how we stack the sub-modules in TCN-GCN-TCN order. The spatial layer is between the two temporal layers and links those two temporal layers together. Considering the dimensions of both temporal and spatial layers, we purposely design it similar to a "sandwich" structure. This trick is also know as "bottleneck strategy", which is very useful when designing a neural network. Having such design encourages the network to compress feature representations to best fit in the available space, in order to get the best loss function during training. Based on existing parameters, each STGCN module is capable of aggregating 1-hop of spatial dependencies, and three months of temporal dependencies. Since the stacked STGCN model has three STGCN modules in it sequentially, overall our proposed framework can aggregate up to 3-hops of spatial dependencies (so 1.5 grid degrees in any direction), and up to 5 months of temporal dependencies. These are clearly tuneable parameters in the model

2.4 Model Metrics

For both STGCN-TCN and STGCN-LSTM models, we use Mean Squared Error (MSE) as the training objective function. This objective function represents the average squared differences between predictions for each grid cell of each month to the ground truth. Mathematically, this loss function is defined as:

$$L = \frac{1}{QN} \sum_{i=1}^{Q} \sum_{j=1}^{N} \left(\hat{y}_j^{(t+i)} - y_j^{(t+i)} \right)^2 \tag{4}$$

	Starts	Ends
Training	January 2000	December 2014
Validation	January 2015	December 2015
Testing	January 2016	December 2017

Table 2: Demonstrations of the training, validation, and testing splits for this paper

We also consider the continuous rank probability score (CRPS) discussed by (Hersbach, 2000, 560–1) and Brandt et al. (2014). For a continuous forecast $\hat{y}_j^{(t+i)}$ and an observed value $y_j^{(t+i)}$ suppose we have a pdf for some $\rho(y)$. Then the CRPS is

$$CRPS(P, y_j^{t+i}) = \int_{-\infty}^{\infty} [P(\hat{y}_j^{(t+i)}) - P_a(\hat{y}_j^{(t+i)})]^2 \partial y_j^{t+i}, \tag{5}$$

The CRPS thus compares the cumulative distributions P and P_a where

$$P(x) = \int_{-\infty}^{y_j^{t+i}} \rho(y) dy \tag{6}$$

$$P_a(x) = H(x - x_a). \tag{7}$$

Here, H is the Heaviside function: $H(x - x_a) = 0$ if $(x - x_a) < 0$ and $H(x - x_a) = 1$ if $(x - x_a) \geq 0$. This is an indicator function that compares the total areas of the predicted and observed cumulative distributions, with lower values indicating better alignment of the forecast and observed densities.

3 Data and results

3.1 Datasets: ViEWS and ICEWS

Our forecasts are based on using 1) the data features of the existing ViEWS baseline random forest for the log fatalities (Hegre et al., 2019), and 2) augmenting these with event data from ICEWS (Boschee et al., 2015) (we discuss how we use these data below). A total of 18 years of monthly data are used here, between January 2000 and December 2017. To align forecasts with the ViEWS contest benchmarks the data are split for training, validation and testing as shown in Table 2.

From the ViEWS' dataset the input data are the 48 variables listed in Table 3. These include both conflicts or fatalities related indicators, and geographic related indicators. For some of the forecasts in addition to the ViEWS dataset features we add ICEWS data (Boschee et al., 2015) retrieved from `http://eventdata.utdallas.edu` using the UTD Event Data R package (Kim et al., 2019). The logic is that applying conflict features from ViEWS could provide promisingly accurate predictions, we might improve the models performance with the additional fine-grained event data. For each PRIO-GRID cell month we aggregate the ICEWS event data into counts each of the 20 CAMEO (2-digit) roots codes (Gerner et al., 2009).

3.1.1 Computing Hardware

Placeholder

3.2 Results

Our initial results are shown in Table 4. Code for our implementation are publicly available for easier replication.[1] The Random Forest model (RF) is the benchmark model that provided by the ViEWS competition organizer.[2] We

[1] `https://github.com/evanli05/ViEWS_Competition`

[2] `https://github.com/UppsalaConflictDataProgram/OpenViEWS2`

acled_count_pr	acled_fat_pr	ged_best_ns	ged_best_os	ged_best_sb
ged_count_os	ged_count_sb	ged_count_ns	pgd_agri_ih	pgd_aquaveg_gc
pgd_barren_ih	pgd_bdist3	pgd_capdist	pgd_cmr_mean	pgd_diamprim
pgd_drug_y	pgd_excluded	pgd_forest_gc	pgd_forest_ih	pgd_gcp_mer
pgd_goldplacer	pgd_goldsurface	pgd_goldvein	pgd_grass_ih	pgd_gwarea
pgd_herb_gc	pgd_imr_mean	pgd_landarea	pgd_maincrop	pgd_mountains_mean
pgd_pasture_ih	pgd_petroleum	pgd_pop_gpw_sum	pgd_savanna_ih	pgd_shrub_gc
pgd_temp	pgd_ttime_mean	pgd_urban_gc	pgd_urban_ih	pgd_water_gc
pgd_agri_gc	pgd_barren_gc	pgd_diamsec	pgd_gem	pgd_harvarea
pgd_nlights_calib_mean	pgd_shrub_ih	pgd_water_ih		

Table 3: Full list of features used from ViEWS dataset

compare our proposed methods (STGCN-LSTM and STGCN-TCN) with this benchmark model. We do this with and without the inclusion of the ICEWS data so that we can separate the performance of the STGCN forecast model variants from the additional data.

Metrics		RF Regressor				STGCN-LSTM (b48-w48)				STGCN-TCN (b32-w60)			
Harware		CPU: i9-7920X				GPU: Quadro RTX 8000							
Features		**v**		**v+u**		**v**		**v+u**		**v**		**v+u**	
Training Time		∼10 mins		∼10 mins		∼9 hrs (50 epochs)		∼9 hrs (50 epochs)		∼45 mins (50 epochs)		∼45 mins (50 epochs)	
Metrics		MSE	CRPS	MSE	CRPS	MSE	CRPS	MSE	CRPS	MSE	CRPS	MSE	CRPS
Time steps	1	3.03×10^{-2}	1.33×10^{-4}	2.69×10^{-2}	1.58×10^{-4}	2.59×10^{-2}	4.47×10^{-2}	2.37×10^{-2}	1.25×10^{-2}	2.32×10^{-2}	1.61×10^{-2}	2.42×10^{-2}	3.54×10^{-2}
	2	3.09×10^{-2}	1.65×10^{-4}	3.04×10^{-2}	9.98×10^{-5}	2.31×10^{-2}	2.25×10^{-2}	2.42×10^{-2}	1.66×10^{-2}	2.32×10^{-2}	3.03×10^{-2}	2.40×10^{-2}	3.40×10^{-2}
	3	3.12×10^{-2}	1.46×10^{-4}	3.06×10^{-2}	1.52×10^{-4}	2.59×10^{-2}	4.47×10^{-2}	2.53×10^{-2}	1.32×10^{-2}	2.21×10^{-2}	7.07×10^{-2}	2.24×10^{-2}	7.33×10^{-2}
	4	3.26×10^{-2}	1.62×10^{-4}	3.21×10^{-2}	2.21×10^{-4}	3.00×10^{-2}	8.36×10^{-2}	2.55×10^{-2}	3.00×10^{-2}	2.36×10^{-2}	2.50×10^{-2}	2.13×10^{-2}	6.54×10^{-2}
	5	3.34×10^{-2}	1.63×10^{-4}	3.31×10^{-2}	1.15×10^{-4}	3.07×10^{-2}	5.60×10^{-2}	2.63×10^{-2}	3.81×10^{-2}	2.14×10^{-2}	1.04×10^{-2}	2.22×10^{-2}	1.60×10^{-2}
	6	3.35×10^{-2}	1.73×10^{-4}	3.32×10^{-2}	1.39×10^{-4}	3.07×10^{-2}	7.66×10^{-2}	2.47×10^{-2}	2.26×10^{-2}	2.21×10^{-2}	2.01×10^{-2}	2.21×10^{-2}	2.63×10^{-2}

Table 4: Comparisons of results between our proposed methods and baseline methods. "**v**" represents "ViEWS" dataset features and "**u**" represents "ICEWS / UTD Event Data" features.

References

Beck, N., G. King, and L. Zeng (2000). Improving quantitative studies of international conflict: A conjecture. *American Political science review*, 21–35.

Blair, R. A. and N. Sambanis (2020). Forecasting civil wars: Theory and structure in an age of "big data" and machine learning. *Journal of Conflict Resolution*, 0022002720918923.

Boschee, E., J. Lautenschlager, S. O'Brien, S. Shellman, J. Starz, and M. Ward (2015). ICEWS coded event data. *Harvard Dataverse 12*.

Brandt, P. T., J. R. Freeman, and P. A. Schrodt (2014). Evaluating forecasts of political conflict dynamics. *International Journal of Forecasting 30*(4), 944–962.

Che, Z., S. Purushotham, K. Cho, D. Sontag, and Y. Liu (2018). Recurrent neural networks for multivariate time series with missing values. *Scientific reports 8*(1), 1–12.

Chiba, D. and K. S. Gleditsch (2017). The shape of things to come? expanding the inequality and grievance model for civil war forecasts with event data. *Journal of Peace Research 54*(2), 275–297.

Cranmer, S. J. and B. A. Desmarais (2017). What can we learn from predictive modeling? *Political Analysis 25*(2), 145–166.

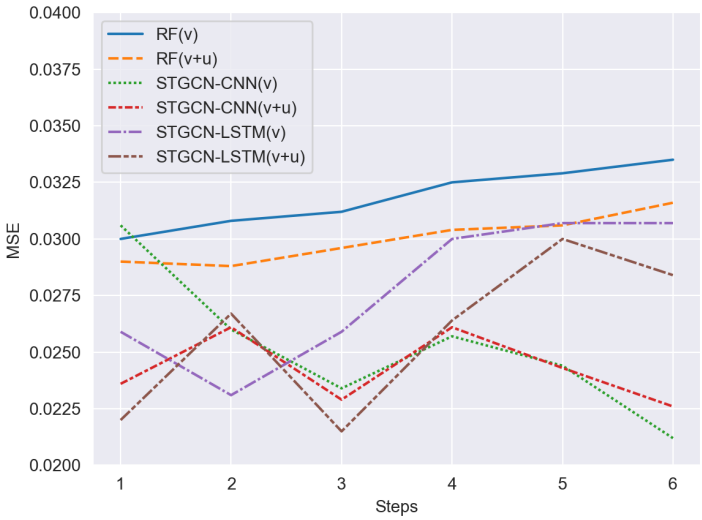

Figure 4: Comparisons of MSE results of proposed methods and baseline methods.

D'Orazio, V. (2020). Conflict forecasting and prediction. In *Oxford Research Encyclopedia of International Studies*.

Dorff, C., M. Gallop, and S. Minhas (2020). Networks of violence: Predicting conflict in nigeria. *The Journal of Politics 82*(2), 476–493.

Gerner, D. J., P. A. Schrodt, and Ö. Yilmaz (2009). Conflict and mediation event observations (CAMEO): An event data framework for a post Cold War world. In J. Bercovitch and S. Gartner (Eds.), *International Conflict Mediation: New Approaches and Findings.*, Chapter 13, pp. 287–304. New York: Routledge.

Hegre, H., M. Allansson, M. Basedau, M. Colaresi, M. Croicu, H. Fjelde, F. Hoyles, L. Hultman, S. Högbladh, R. Jansen, et al. (2019). Views: a political violence early-warning system. *Journal of peace research 56*(2), 155–174.

Hersbach, H. (2000). Decomposition of the continuous ranked probability score for ensemble prediction systems. *Weather and Forecasting 15*, 559–70.

Hill Jr, D. W. and Z. M. Jones (2014). An empirical evaluation of explanations for state repression. *American Political Science Review*, 661–687.

Kim, H., V. D'Orazio, P. T. Brandt, J. Looper, S. Salam, L. Khan, and M. Shoemate (2019). UTDeventdata: An R package to access political event data. *Journal of Open Source Software 4*(36), 1322.

Kipf, T. N. and M. Welling (2017). Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Li, Y., R. Yu, C. Shahabi, and Y. Liu (2018). Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Lim, B. and S. Zohren (2020). Time series forecasting with deep learning: A survey. *arXiv preprint arXiv:2004.13408*.

Tai, K. S., R. Socher, and C. D. Manning (2015). Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pp. 1556–1566. The Association for Computer Linguistics.

Yu, B., H. Yin, and Z. Zhu (2018). Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In J. Lang (Ed.), *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pp. 3634–3640. ijcai.org.

Supplemental Materials

Long Short Term Memory (LSTM)

A Recurrent Neural Network (RNN) model is popular framework for modeling sequences like time-series forecasting (Che et al., 2018). The RNN mechanism allows it to exhibit dynamic behaviors that take advantage of their internal hidden state and process variable length sequences of inputs (Li et al., 2018). Figure 5 illustrates the basic inputs and outputs of the state-space of the LSTM model.

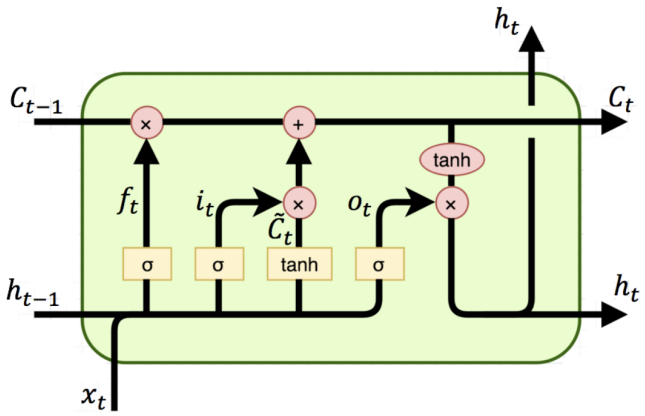

Figure 5: Architecture of one LSTM cell. Here σ and tanh are activation functions.

One version of the RNN model implemented here is called Long Short Term Memory Network (LSTM). Tai et al. (2015) describe the architecture with the following equations:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{8}$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{9}$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{10}$$
$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{11}$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{12}$$
$$h_t = o_t \odot \tanh(C_t) \tag{13}$$

where x_t is the input of current time step, which is of dimension $n \times d$. The sigmoid function σ is a logistic function, and $\odot$ denotes elementwise multiplication. The variable h_t represents the short-term memory (embedding), and C_t represents the long term memory, and h, C have random initialization at time step 0.

STGCN-LSTM

Figure 6 illustrates the architecture of the STGCN-LSTM framework. Unlike the STGCN-TCN framework, which mixes the spatial layers and temporal layers together in each STGCN block, STGCN-LSTM is a two step model which can be separated clearly by (1) spatial dependency modeling, and (2) temporal dependency modeling. More specifically, first we use stacked GCN blocks to extract spatial information out of our data, and embed them into the H_t at each time step. Each stacked GCN module contains three GCN blocks in order to obtain multi-hop information for the target grid cell.

Then, we feed H_t into the LSTM model in a sequential manner and generate a unique hidden embedding Emb after P steps. The LSTM model uses Emb and make predictions sequentially for Q steps.

Based on existing parameters, each stacked GCN module is capable to aggregate 3-hop spatial dependencies. The long term memory mechanism helps the LSTM module to maintain all previous information, which can be tracked back to $t - P + 1$ at time step t, which makes the extracted temporal dependencies upto P steps.

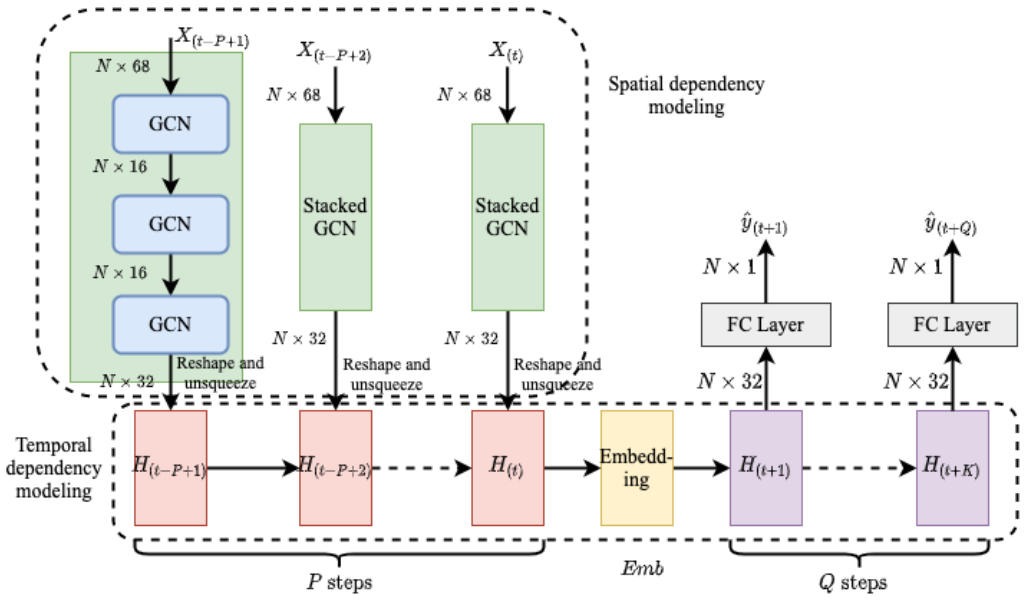

Figure 6: Architecture of STGCN-RNN. This framework uses GCN for spatial modeling, and applies LSTM as backbone for temporal modeling.

Demo of STGCN

Here we further demonstrate our proposed model by a simplified example. Suppose that there is a area with 5×5 PRIO-GRID cells, for each of the grid cell we have 3 dimensional features, and we have a total of 4 months of data available. Also, we want the model to have the ability of predicting two month ahead. Therefore for the training, we have 2 months of features (X_0 and X_1) and 2 months of labels (y_2 and y_3) available.

First, we demonstrate the model of STGCN-LSTM in Fig 7. Since there is at most 2-hop connection w.r.t the center cell x_{0a}, here we apply two layers of GCN module. Since each GCN layer can aggregate 1-hop information, therefore after the first GCN layer, those surrounding nodes information from x_{1u} where $u \in \{a-h\}$ are encoded into x_{0a}. Noticing that nodes such as x_{1a} also aggregate its surrounding information from nodes such as x_{2a} in this first GCN layer. Then, we feed embedding generated by the 1st layer to the 2nd GCN layer. Since x_{1u} already got x_{2v}, where $v \in \{a-p\}$ encoded from last layer, then here x_{0a} can theoretically aggregates information from 2-hops away.

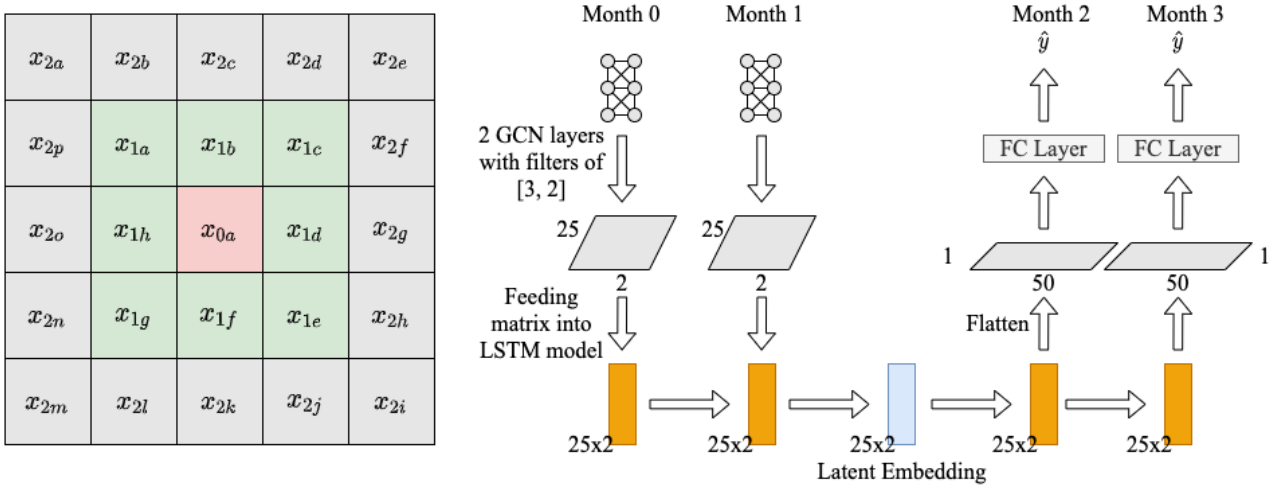

Figure 7: Example of STGCN-LSTM model

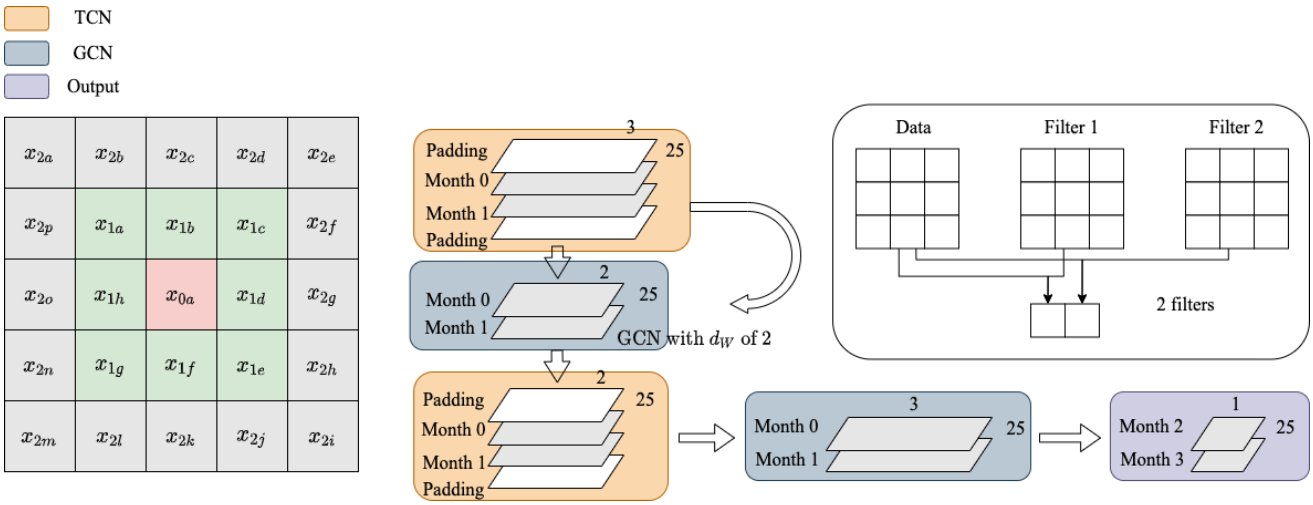

Figure 8: Example of STGCN-TCN model