SAUC: Sparsity-Aware Uncertainty Calibration for Spatiotemporal Prediction with Graph Neural Networks (Supplemental Materials)

Anonymous Author(s)
Affiliation
Address
email

1 Data Description

We use two spatiotemporal datasets in our study.

(1) **Chicago Traffic Crash Data (CTC)** [1] sourced from 277 police beats between January 1, 2016 and January 1, 2023. The CTC data records show information about each traffic crash on city streets within the City of Chicago limits and under the jurisdiction of Chicago Police Department.

(2) **Chicago Crime Records (CCR)** [2] derived from 77 census tracts spanning January 1, 2003 to January 1, 2023. This dataset reflects reported incidents of crime (with the exception of murders where data exists for each victim) that occurred in the City of Chicago.

Despite both CTC and CCR data originating from the Chicago area, their disparate reporting sources lead to different spatial units: census tracts for CCR and police beats for CTC. The temporal resolutions of the datasets are varied to demonstrate the ubiquitous sparsity issue and its practical significance in spatiotemporal analysis. For example, four temporal resolutions are created for CTC and CCR datasets, with further details available in Table 1. We designate the 1-hour and 8-hour cases of Crash and Crime datasets as sparse instances, due to a higher prevalence of zeros. Both datasets use the first 60% timesteps for training, 20% for calibration and validation, and 20% for testing.

Dataset	Resolution	Size	Sparsity	Mean	Max
CCR	1-hour	(77, 175321)	67%	0.3	59
	8-hour	(77, 21916)	18%	2.7	206
	1-day	(77, 7306)	4%	8	223
	1-week	(77, 1044)	$< 0.1\%$	55.8	539
CTC	1-hour	(277, 61369)	96%	$< 0.1\%$	5
	8-hour	(277, 7672)	76%	0.4	9
	1-day	(277, 2558)	47%	1.1	15
	1-week	(277, 366)	7%	7.3	45

Table 1: Characteristics of various datasets showing variation in sparsity at different temporal resolutions. Dataset sizes are represented as (spatial, temporal) dimension pairs. Sparse cases are marked grey.

For adjacency matrices, we calculate geographical distances d_{ij} between centroids of regions i and j, be they census tracts or police beats. This distance is then transformed into a similarity measure, $\mathbf{A}_{ij} = e^{-d_{ij}/0.1}$, where 0.1 is a scaling parameter, thus forming an adjacency matrix. Notice that this scaling parameter can be varied, like using the standard deviations of the data as the parameter [**?**]. In our study, the results before and after calibrations come from the same model, therefore the

[1] `https://data.cityofchicago.org/Transportation/Traffic-Crashes-Crashes/85ca-t3if`
[2] `https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2`

Submitted to Temporal Graph Learning Workshop @ NeurIPS 2023, New Orleans. Do not distribute.

selection of the adjacency matrix is pretty flexible in spatiotemporal prediction with Graph Neural Networks (GNNs).

In the spatiotemporal dataset, the granularity of the temporal resolutions will result in different levels of resolutions, which are important. For crime and accident data, we control the data sparsity by varying the temporal resolution. We vary the temporal resolution from weekly, daily, 8 hours, and 1 hour. Such division is based on the fact that crime and accident data less frequently happened on the city scale.

2 Implementation of Modified GNN

2.1 Negative Binomial distribution

A random variable that follows Negative Binomial (NB) distribution has a probability mass function f_{NB} as:

$$f_{NB}(x_k;\mu,\alpha) \equiv Pr(X = x_k) = \begin{pmatrix} x_k + n - 1 \\ n - 1 \end{pmatrix} (1-p)^{x_k} p^n. \tag{1}$$

where n and p are the shape parameters that determine the number of successes and the probability of a single failure respectively. In our case, we use μ (the mean) and α (the dispersion parameter) instead, which has the form as:

$$n = \frac{\mu\alpha}{1-\alpha}; p = \frac{1}{1+\mu\alpha}. \tag{2}$$

We choose μ and α since the value of μ is directly related to the mean values, which is straightforward and keep consistent with location and scale parameters from Gaussian distributions.

2.2 Modification of GNN Models

Inspired by recent probabilistic GNN models that predominantly assume data distributions and leverage spatiotemporal architectures for parameter estimation, we transitioned traditional numerical GNN models to a probabilistic framework by altering the last layer. We modify two existing popular spatiotemporal forecasting models: Spatio-temporal Graph Convolution Network (STGCN) [**?**] and Graph WaveNet (GWN) [**?**]. They are designed for numerical value outputs. We modify their last layer to NB distributions to adapt our context. Specifically, we substituted parameters to ensure outcomes of μ and α. For example, STGCN applies one fully connected layer to the outputs of its spatial and temporal encoding. We modify it into two fully connected layers, with the same spatial and temporal encoding as the inputs for both layers, but output the values of μ and α respectively.

Therefore, the loss function needs to change, from mean square error loss to the loss defined in Equation 3:

$$\begin{aligned} \mathcal{L}(\mu,\alpha,y) = &- \left[y \cdot \log\left(\frac{\mu+\epsilon}{\mu+\alpha+2\epsilon}\right)\right] \\ &- \Gamma(y+\alpha+\epsilon) + \Gamma(y+1) + \Gamma(\alpha+\epsilon), \\ &- \alpha \cdot \log\left(\frac{\alpha+\epsilon}{\mu+\alpha+2\epsilon}\right) + \lambda \cdot ||\alpha||^2 \end{aligned} \tag{3}$$

where y is the target variable, μ and α are model outputs, Γ is the gamma function, λ is the regularization parameter, and ϵ is a small constant added to improve numerical stability. This loss function is derived from the likelihood of NB distribution controlled by μ and α.

3 Implementation of the Modified GNNs

To implement the modified GNNs, we use the Github repositories for STGCN [3] and GWN [4] respectively. We basically keep all the hyper-parameters and important parameters as the default from the original repositories, which can be referred to in our Github repository [5]. Notice that we keep the

[3] `https://github.com/FelixOpolka/STGCN-PyTorch`

[4] `https://github.com/nnzhan/Graph-WaveNet`

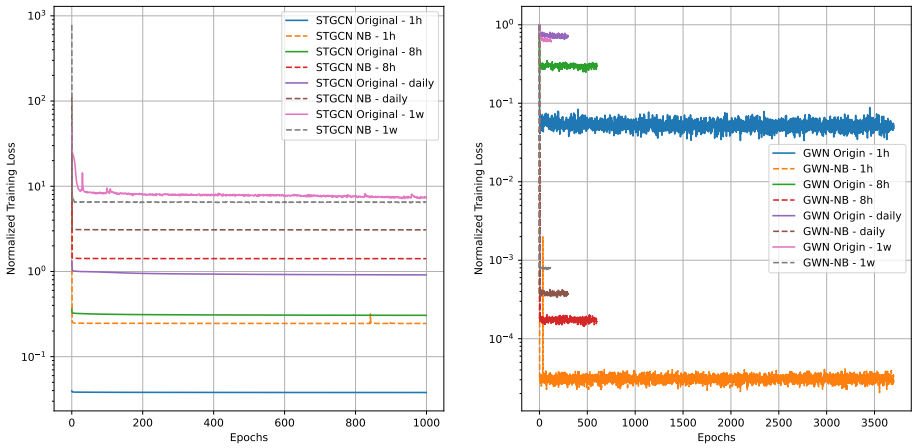
[5] `https://github.com/AnonymousSAUC/SAUC`

same model parameters in both the original models and the modified models, which can be found in Table 2

Parameters/Models	STGCN	GWN
Input window length	12	12
Output window length	12	12
Learning rate	0.001	0.001
Batch size	24	24
Output layer	Linear	Conv2d
Training epochs	1000	100
Early stop	No	Yes

Table 2: Parameters of original and modified GNN models.

Training loss on CCR datasets can be found in Figure 1. All models are converged properly.



(a) Training Loss of STGCN on CCR dataset (b) Training Loss of GWN on CCR dataset

Figure 1: Training loss of STGCN, GWN, and their modifications on the CCR dataset.

The full implementation results can be found in Figure 3. It is clear that the model's performance in terms of numerical accuracy is close to the results using models before modification. In fact, the NB modification shows better performance in the sparse dataset as the NB distributions are more suitable for the nature of discrete data.

3.1 Reproducibility

All our experiments are implemented on a machine with Ubuntu 22.04, with Intel(R) Core(TM) i9-10980XE CPU @ 3.00GHz CPU, 128GB RAM, and NVIDIA GeForce RTX 4080 GPU.

4 Baseline Calibration Methods

We list the introductions and pseudo codes to implement the baseline models.

4.1 Temperature Scaling

Temperature scaling is a parametric method that modifies the outputs of a regression model. This modification is controlled by a single parameter, the temperature T. The algorithms can be formulated as follows:

Dataset	Models	MAE	MSE	RMSE
CCR_1h	STGCN	0.429	0.406	0.637
	STGCN-NB	0.317	0.494	0.703
	GWN	0.848	0.926	0.962
	GWN-NB	0.334	0.587	0.766
CCR_8h	STGCN	1.499	5.691	2.386
	STGCN-NB	1.452	5.413	2.326
	GWN	1.395	4.212	2.052
	GWN-NB	1.478	5.645	2.376
CCR_daily	STGCN	2.951	26.698	5.167
	STGCN-NB	2.719	18.368	4.286
	GWN	2.521	14.316	3.784
	GWN-NB	2.681	19.152	9.703
CCR_weekly	STGCN	14.146	544.415	23.333
	STGCN-NB	12.505	409.741	20.242
	GWN	9.938	259.311	16.103
	GWN-NB	14.516	539.679	23.231
CTC_1h	STGCN	0.084	0.046	0.214
	STGCN-NB	0.044	0.048	0.220
	GWN	0.960	0.960	0.980
	GWN-NB	0.044	0.048	0.220
CTC_8h	STGCN	0.453	0.401	0.633
	STGCN-NB	0.346	0.527	0.726
	GWN	0.810	0.855	0.924
	GWN-NB	0.352	0.557	0.746
CTC_daily	STGCN	0.861	1.318	1.148
	STGCN-NB	1.433	3.193	1.786
	GWN	0.895	1.332	1.154
	GWN-NB	1.019	2.404	1.55
CTC_weekly	STGCN	2.474	11.004	3.317
	STGCN-NB	3.618	15.442	3.929
	GWN	2.423	10.098	3.178
	GWN-NB	3.279	13.608	3.689

Table 3: Comparisons between modified models and the original models.

Algorithm 1 Temperature Scaling for Regression

1: Train a model on a training dataset, yielding point predictions $f(x)$.
2: Optimize the temperature T on the validation set by minimizing the mean squared error between the predicted and true values.
3: **for** each new input x **do**
4: Compute the model's prediction $f(x)$.
5: Apply temperature scaling with the learned T to the prediction $f(x)$, yielding scaled prediction $g(f(x))$.
6: **end for**

Temperature scaling is an efficient and straightforward method for calibration that requires optimization of only a single parameter. By scaling the outputs, it can yield calibrated predictions without modifying the rank ordering of the model's predictions.

4.2 Isotonic Regression

Isotonic regression is a non-parametric method utilized for the calibration of a predictive model's outputs. In the context of regression, this method operates as shown below:

Notice that isotonic regression makes no assumptions about the form of the function connecting the model's predictions to calibrated predictions. This flexibility allows it to fit complex, non-linear mappings, which can provide improved calibration performance when the model's outputs are not well-modeled by a simple function.

Algorithm 2 Isotonic Regression for Regression

1: Train a model on a training dataset, providing point predictions $f(x)$.
2: Fit an isotonic regression model on the validation set predictions $f(x)$, mapping them to calibrated predictions.
3: **for** each new input x **do**
4: Compute the model's prediction $f(x)$.
5: Use the isotonic regression model to convert the prediction $f(x)$ to a calibrated prediction $g(f(x))$.
6: **end for**

4.3 Histogram Binning

Histogram binning can also be applied in a regression setting to calibrate predictions. The idea is to split the range of your model's outputs into several bins, and then adjust the predictions within each bin to match the average actual outcome within that bin. The specific algorithm is:

Algorithm 3 Histogram Binning for Regression

1: Train a model on a training dataset, yielding predictions $f(x)$.
2: Determine the bins for the predictions on the validation set.
3: **for** each bin **do**
4: Compute the average true value for all examples in the bin.
5: Adjust the prediction for each example in the bin to the average true value.
6: **end for**
7: **for** each new input x **do**
8: Compute the model's prediction $f(x)$.
9: Find the bin that $f(x)$ falls into and adjust $f(x)$ to the average true value for that bin.
10: **end for**

This method assumes that predictions within each bin are uniformly mis-calibrated. It is less flexible than isotonic regression, but it is simpler and less prone to overfitting, especially when the number of bins is small.

4.4 Platt Scaling

In a regression setting, Platt Scaling can be interpreted as applying a sigmoid function transformation to the model's outputs. The output is then considered as the mean of a Bernoulli distribution. While this might be useful in some contexts, it is not generally applicable to regression problems but can be useful in sparse datasets. A linear transformation is fitted to the model's predictions to minimize the mean squared error on the validation set. The parameters of this transformation are learned from the data, making this an instance of post-hoc calibration. Note that this adaptation might not be suitable for all regression tasks, especially those where the target variable has a non-linear relationship with the features.

Algorithm 4 Platt Scaling for Regression

1: Train a model on a training dataset, yielding predictions $f(x)$.
2: Minimize the mean squared error on the validation set between $g(f(x))$ and the true outcomes, where $g(f(x)) = af(x) + b$ is a linear transformation of the model's predictions.
3: **for** each new input x **do**
4: Compute the model's prediction $f(x)$.
5: Apply the learned linear transformation to $f(x)$ to get the calibrated prediction.
6: **end for**