TPGNN Tutorial

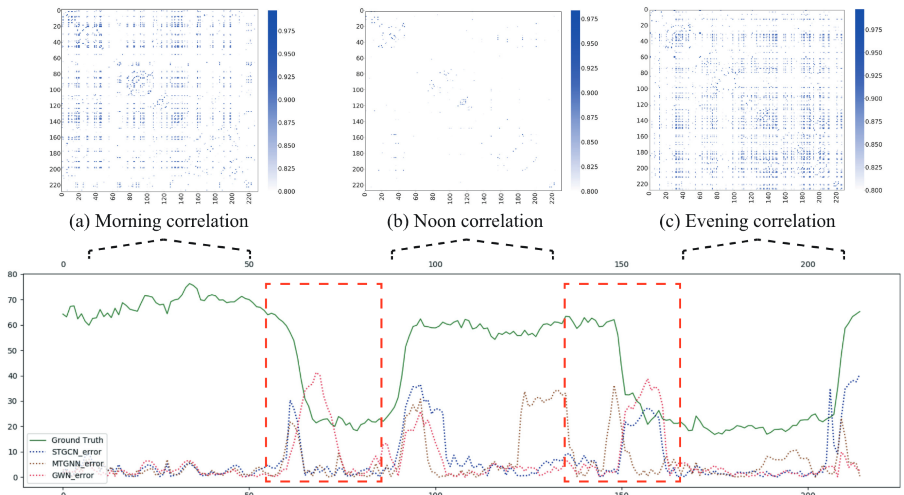
https://proceedings.neurips.cc/paper_files/paper/2022/file/7b102c908e9404dd040599c65db4ce3e-Paper-Conference.pdf

WHY

Accurate forecasting of Multivariate Time Series (MTS) is difficult due to complex and time-varying correlations between variables, which traditional methods struggle to capture (GCNs like STGCN, DCRNN,…), which leads to bias in prediction.

There is also a gap in dependence learning. Previous approaches often don't really analyze the gap between actual correlation and learned one. There is no adequate theoretical or empirical analysis of this gap.

Figure 1: Empirical proof of the necessarity of the work



(d) Ground truth and prediction error along the timeline

HOW

TPGNN (Temporal Polynomial Graph Neural Network) introduces a novel method for modeling time-varying correlations in MTS (Multivariate Time Series) data using a **temporal matrix polynomial** approach.
The authors propose an improved
approximation error analysis in the form **of** theoretical bounds for the approximation error of the dynamic correlations. This offers insights into its accuracy (**Theorem**)

WHAT

TPGNN improves forecasting performance, outperforming existing methods by **23.41%** in approximation error and provides a **theoretical framework** for analyzing the correlation modeling gap.

PEEL: Point, Evidence, Example, Link

Defintions

Paper Theory	Code Implementation
N Variables of MTS data	n_route
$A \in \mathbb{R}^{N \times N}$ Adjacency Matrix → Static → Predefined	opt.dis_mat = weight_matrix_nl(opt.adj_matrix_path, epsilon=opt.eps)
$E \in \mathbb{R}^{N \times c}$ is a c- dimensional embedding of N	-
T number historical observations	n_his
T' number future time steps	n_pred
V: Set of nodes (variables). There are N nodes, each representing a variable in the MTS data.	n_route (n_attr = 1 in our case)
$E(t)$: Set of edges at time t, representing relationships (dependencies) between variables. → **weather** two nodes are connected	-
$X(t) \in \mathbb{R}^{N \times 1}$ Observations or signals of the N variables at time t, one sample	forward method of SrcProcess
$W(t) = \sum a(t)_k A^k \in \mathbb{R}^{N \times N}$: Weighted adjacency matrix indicating the correlation between variables. ∘ If variable i and j are dependent, $W(t)_{ij}$ is non-zero and reflects the strength of the correlation. ∘ If not, $W(t)_{ij} = 0$ (no edge) → **strength** of connections → time-varying/ **dynamic (main contribution of the paper)**	TempoEnc
K order of the matrix polynomial - used to represent temporal dependencies	

At each time step t, the MTS data is represented as a **graph signal set:**

$$G(t) = \{V, E(t), X(t), W(t)\}$$

Graph data regarded as latent[1] function of MTS data [1] **vorhanden, aber noch nicht erkennbar, versteckt, verborgen, nicht offenkundig**

Forecasting Goal

- **Input:** The model is given T historical observations of the graph signal sets:

$$G(t), G(t+1), \ldots, G(t+T-1)$$

- **Output:** The model aims to predict the future signals of the variables for $T\prime$ time steps:

$$X(t+T), X(t+T+1), \ldots, X(t+T+T\prime-1)$$

- **Function F:** The goal is to find a mapping F that uses the historical graph signals to make these predictions.
  → sequence to sequence / time-series forecasting model

Temporal Polynomial Graph

Adjacency Matrix Construction (Eq. 2-4)

Equation 2 - Self-adaptive graph

#Is the same as *MTGNN*, identify the position of this step in the code

$A = SoftMax(ReLU(EE^T))$

E output of encoder module

- it represents the embeddings of the nodes/variables

$\mathbf{EE}^T$ matrix product

- Representing similarity between variables

$ReLU$

- all negative values to zeros, only positive probabilities are important

$SoftMax()$

- normalizes every row of the matrix to probabilities, the sum of each row is 1
- emphasizes differences of dependencies more

Equation 3 - Final adjacency matrix

$A = SoftMax(ReLU(EE^T)) + L$

L Symmetric and Normalized Laplace matrix

(In *Newman's Networks: An Introduction* - Graph Laplacian)

$L = D - W$

D Degree Matrix: Diagonal Matrix with entries d_i the sum of weights of each each adjacent edge to node i

W Weighted Adjacency Matrix (Correlations)

$Lsym = D^{-1/2}(I + W)D^{-1/2}$

- Scaling of adyacency matrix W: Scaling down high degree nodes
- Goal:
  - Adding topological information
  - Stabilizing: Support loops and node relationships based on distances and correlations
  - L might not be needed in our electricity case since no spatial / physical proximity between nodes

Equation 4 - Dynamic Graph Representation using Matrix Polynomials

$W(t) = \sum a(t)_k A^k$

- K-order matrix polynom with A^k being the power of adjacency matrix modeled for specific time step t
- Coefficients $a(t)_k$ determine influence of the order on overall weight
- Not specified how K is found

→ Time-Dependent Weight Matrix / Correlation Matrix

Propagation

Eq. 5-7

Equation 5 - Temporal Coefficient Generation

$(a(t), ..., a(t + T - 1)) = (e(t\%Tp)_{ts}, ..., e(t + T - 1\%Tp)_{ts})W_c$

Model uses **cyclic timestamp embeddings** to generate time-varying polynomial coefficients $a(t)_k$ for each time step.

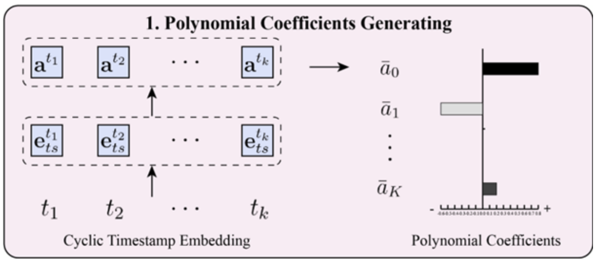
- **[**A cyclic embedding introduces **periodicity,** capturing cyclic nature of the data] $e(t\%Tp)$
- Every time step t is mapped to specific position within cyclic interval
- Periodic cycles of length Tp, mapping done using modulo $t\%Tp$
- $W_c \in \mathbb{R}^{D_e \times (K+1)}$ is the parameter matrix that maps the cyclic embeddings to polynomial coefficients. K+1 corresponds to the number of polynomial terms
- D_e embedding dimension, size of cyclic embedding

Equation 6 - Average Coefficients for Efficient Prediction

$\bar{a} = (\bar{a}_0, \bar{a}_1, ..., \bar{a}_K) = (a(t), ..., a(t + T - 1))W_a$

Instead of calculating different polynomial terms for each of the T time steps, the model calculates an **average polynomial** to improve efficiency and robustness

W_a is a learned parameter matrix



Equation 7 - TPG Module Propagation

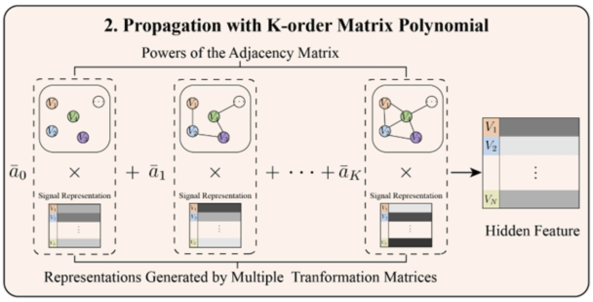
$Z^{(t)} = \sum_k \bar{a}_k A_k X^{(t)} \frac{W_k}{||W_k||_F}$

Describes how the time-varying graph signal X(t) is processed through the temporal graph structure, according to A, $\bar{a}_k$ and W_k

- $Z(t)$ is the final hidden feature at time step t, result of propagation through graph structure
- W_k Model parameters defining how embeddings of variables are transformed at each polynomial term, each trained seperately
- $||W_k||_F$ Frobenius Normalization normalizing the matrix, used to prevent large values

→ Captures Dynamic Relationships

→ Incorporate Higher-Order Dependencies



Encoder-Decoder Pipeline

- Prediction in Auto-Regressive manner

Encoding Process

→ Captures structure and temporal relationships in input data

Input

Input is historical data embeddings derived using linear transformation. Linear transformation is to get representation that the model can work with more effectively.

$X(t : t + T - 1) \in \mathbb{R}^{T \times N \times D_e}$

1. Temporal Attention Layer
    - Inspired by Transformer architecture
    - Identifies intra-series patterns for each variable
2. Temporal Graph Encoding (**TPG module**)
    - Equation 7
    - Propagates information across temporal graph

Output

Result is of Encoding is Encoded Data with rich topological and temporal information.

$Z^{(t:t+T-1)} = \sum_k \bar{a}_k A_k X^{(t:t+T-1)} \frac{W_k}{||W_k||_F} \in \mathbb{R}^{T \times N \times D_e}$ (Sequence from Equation 7)

Decoding Process

→ Generates prediction step by step using previously generated outputs to refine future outputs

Input

- BOS token $E_{\text{BOS}} \in \mathbb{R}^{N \times D_e}$

  Is an initial input to the decoder, to signify start of predcition sequence
- Encoded data $Z(t : t + T - 1)$
- “Embedding” (Framework Figure)

Equations 8 - First Prediction (Time Step $t + T$)

→ Establishes a **starting point**. It predicts the next time step based only on the historical data and the initial token, without any prior forecasted values.

$E(t + T)_X = Decoder(E_{BOS}, Z(t : t + T - 1))$

- **Query Result**, Encoded representation of forecast for time step $t + T$

$X\ (t + T) = E(t + T)_X W_{pred}$

- **Final forecast** is computed by projecting the query result using a prediciton matrix $W_{pred} \in \mathbb{R}^{D_e \times 1}$
- $\tilde{X}(t + T) \in \mathbb{R}^{N \times 1}$ forecast values for time step $t + T$

Equations 9 - Subsequent Predictions

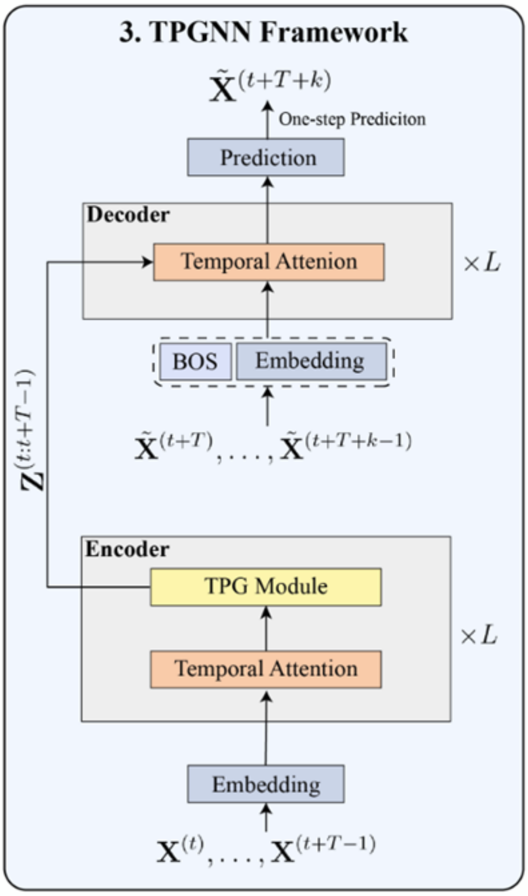
→ Serve to **refine** and **propagate** the forecast over time, benefiting from the AR mechanism where each prediction helps improve the next - capture long-term dependencies

$E(t+T+k)_X = Decoder((E(t+T+k-L_{max})_X, ..., E(t+T+k-1)_X), Z(t:t+T-1)))$

- **Auto-Regressive Querying** to predict time step $t+T+k$ uses L_{max} (number of precious predictions) most recent query results to refine future forecasts

$\tilde{X}(t+T+k) = E(t+T+k)_X W_{pred}$

- How forecasted value for $t+T+k$ is obtained



⇒ Explainability of TPGNN

Theorem 1 (Equation 10)

$(1-\lambda_{max}) \cdot \mathbb{E}_t \mid\mid G(t) \mid\mid_F^2 \leq e(1:T) \leq (1-\lambda_{max}) \cdot \mathbb{E}_t \mid\mid G(t) \mid\mid_F^2$

→ States that the approximation error of TPGNN's learned graph **lies within a certain range depending on the eigenvalues of the Laplacians** of the true graph

Goal: Minimize approximation error

Denotations and Definition

- Adjacency matrix A
- Time-varying Laplacian matrices $G(t)$
  - symmetric due to undirected graph
  - ideal theoretical construct, not explictely imputed (?)
- Approximation error $e(1:T) = \frac{1}{T}\sum_{t=1}^{T} \mid\mid W(t) - G(t) \mid\mid_F^2$
  - How well TPGNN's learned graph structure approximates the true graph structure for each time step

- **Lower bound**
  $(1 - \lambda_{max}) \cdot \mathbb{E}_t \mid\mid G(t) \mid\mid_F^2$
- **Upper bound**
  $(1 - \lambda_{min}) \cdot \mathbb{E}_t \mid\mid G(t) \mid\mid_F^2$
  - $\lambda_{max}, \lambda_{min}$ maximum and minimum eigenvalues of matrix
  - $\mathbb{E}_t \mid\mid G(t) \mid\mid_F^2 = \frac{1}{T}\sum_{t=1}^{T} \mid\mid G(t) \mid\mid_F^2$ is the average squared Frobenius norm of the true graph Laplacians over time
  - $\mid\mid \cdot \mid\mid F$ Frobenius norm, a measure of matrix size or magnitude

- **Observation**: Smaller eigenvalue spread results in smaller error and vice versa

Conditions

- Laplacian matrices G(t) are **symmetric** for each time step. This is a key condition for the analysis in the theorem
  $G_iG_j = G_jG_i$ for all i, j
- Polynomial order K large enough → Model should consider enough terms in polynomial to capture sufficient dynamics of graph structre
- **Condition for Perfect approximation**
  If the learned adjacency matrix A commutes with the true graph Laplacians G(t): $A \cdot G(t) = G(t) \cdot A$
  - In this case the **maximum** and **minimum** eigenvalues of the Laplacian matrix G(t) are equal to 1: $\lambda_{max} = \lambda_{min} = 1$ which results to $e(1:T) = 0$

Conclusion

- High-quality approximation of graph structure even in dynamic environments
- Performance is close to optimal graph structure under right conditions

---

Experiments

Datasets

Dataset	#Samples	#Nodes	Sample Rate	Input Length	Output Length	Task type	Domain
Traffic	17,544	862	1 hour	168	1	Single future step	Transpor

Solar-Energy	52,560	137	10 minutes	168	1	Single future step	Energy
Electricity	26304	321	1 hour	168	1	Single future step	Energy
Exchange-Rate	7,588	8	1 day	168	1	Single future step	Economi
PEMS-D7	12672	228	5 minutes	12	12	Multi future step	Traffic
PEMS-Bay	52116	325	5 minutes	12	12	Multi future step	Traffic

Baseline - Competitor models

Metrics Denotation

Metric	Formula	Desirable Direction	Notes	
RSE = Root Squared Error	$\mathrm{RSE} = \frac{\sqrt{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}}{\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}$	Low	RSE measures the discrepancy between predicted and actual values.	
CORR = Empirical Correlation Coefficient	$CORR = \frac{\sum(x_i-\bar{x})(y_i-\bar{y})}{\sqrt(\sum(x_i-\bar{x})^2\sum(y_i-\bar{y})^2)}$	High	Empirical correlation coefficient indicates how well the model captures the relationship between variables.	
MAE = Mean Absolute Error	$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}\lvert y_i-\hat{y}_i\rvert$	Low	Measures the average absolute difference between the actual and predicted values, providing a straightforward indication of prediction accuracy.	
MAPE = Mean Absolute Percentage Error	$\mathrm{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left\lvert\frac{y_i-\hat{y}_i}{y_i}\right\rvert \times 100$	Low	Calculates the average percentage difference between the actual and predicted values, giving insight into prediction accuracy relative to the scale of the actual values.	
RMSE = Root Mean Squared Error	$RMSE = \sqrt{\frac{1}{n}\sum(y_i-\hat{y}_i)^2)}$	Low	Root mean squared error penalizes larger errors more, providing an indication of prediction accuracy.	

Notes

- Single vs Multi-step Forecasting:
    - In **single-step forecasting**, the model predicts only **one value for the next time step**
    - The model predicts **multiple values** ahead

- **Single-step horizons (3, 6, 9, 12)**:

  These horizons represent the number of time steps (e.g., hours or days) ahead for which a model predicts a single future value.
- **Multi-step horizons (15, 30, 60 minutes)**:

  These horizons refer to the time intervals (e.g., 15, 30, or 60 minutes) for which a model predicts multiple consecutive future values.

- **Models not GNN-based:**
    - **ARIMA**: Traditional time-series model.

- **FC-LSTM**: Sequence-to-sequence model using recurrent networks.
- **Informer**: Transformer-based model for time-series forecasting.

Single-step dataset results

Methods	Metrics	SE 3	SE 6	SE 12	SE 24	T 3	T 6
VARMLP	RSE	0.1922	0.2679	0.4244	0.6841	0.5582	0.6579
	CORR	0.9829	0.9655	0.9058	0.7149	0.8245	0.7695
GP	RSE	0.2259	0.3286	0.5200	0.7973	0.6082	0.6772
	CORR	0.9751	0.9448	0.8518	0.5971	0.7831	0.7406
RNN-GRU	RSE	0.1932	0.2628	0.4163	0.4852	0.5358	0.5522
	CORR	0.9823	0.9675	0.9150	0.8823	0.8511	0.8405
LSTNet	RSE	0.1843	0.2559	0.3254	0.4643	0.4777	0.4893
	CORR	0.9843	0.9690	0.9467	0.8870	0.8721	0.8690
TPA-LSTM (SOTA)	RSE	0.1803	0.2347	0.3234	0.4389	0.4487	0.4658
	CORR	0.9850	0.9742	0.9487	0.9081	0.8812	0.8717
MTGNN (SOTA)	RSE	0.1778	0.2348	0.3109	0.4270	0.4162	0.4754
	CORR	0.9852	0.9726	0.9509	0.9031	0.8963	0.8667
TPGNN (SOTA)	RSE	0.1850	0.2412	0.3059 (1.61%)	0.3498 (18.08%)	0.3989	0.4715
	CORR	0.9840	0.9716	0.9529 (1.47%)	0.9710 (6.93%)	0.9232	0.8945
						CORR T avg. 2.21%	CORR T 2.21%

Multi-step dataset results

Model	**PEMS-BAY** (Horizon 3/6/12)	**PEMS-D7** (Horizon 3/6/12)
ARIMA	**MAE**: 1.62/2.33/3.38 — **MAPE**: 3.50/5.40/8.30 — **RMSE**: 3.30/4.76/6.50	**MAE**: 5.55/5.86/6.27 — **MAPE**: 12.92/13.94/15.20 — **RMSE**: 9.00/9.13/9.38
FC-LSTM	**MAE**: 2.05/2.20/2.37 --- **MAPE**: 4.80/5.20/5.70 --- **RMSE**: 4.19/4.55/4.96	**MAE**: 3.57/3.92/4.16 --- **MAPE**: 8.60/9.55/10.10 --- **RMSE**: 6.20/7.03/7.51
STGCN	**MAE**: 1.39/1.84/2.42 --- **MAPE**: 3.00/4.22/5.58 --- **RMSE**: 2.92/4.12/5.33	**MAE**: 2.25/3.03/4.02 --- **MAPE**: 5.26/7.33/9.85 --- **RMSE**: 4.04/5.70/7.64
DCRNN	**MAE**: 1.38/1.74/2.07 --- **MAPE**: 2.90/3.90/4.90 --- **RMSE**: 2.95/3.97/4.74	**MAE**: 2.25/2.98/3.83 --- **MAPE**: 5.30/7.39/9.85 --- **RMSE**: 4.04/5.58/7.19
StemGNN	**MAE**: 1.52/1.94/2.45 --- **MAPE**: 3.38/4.58/6.03 --- **RMSE**: 3.06/4.07/5.04	**MAE**: 2.94/3.66/4.66 --- **MAPE**: 7.63/9.66/12.58 --- **RMSE**: 5.05/6.35/8.00
Graph WaveNet	**MAE**: 1.30/1.63/1.95 --- **MAPE**: 2.73/3.67/4.63 --- **RMSE**: 2.74/3.70/4.52	**MAE**: 2.18/2.95/3.88 --- **MAPE**: 5.02/7.22/10.03 --- **RMSE**: 4.18/5.82/7.61
Informer (SOTA)	**MAE**: 2.30/2.40/2.55 --- **MAPE**: 5.02/5.32/5.73 --- **RMSE**: 4.21/4.49/4.85	**MAE**: 3.64/3.77/4.09 --- **MAPE**: 8.66/9.07/9.87 --- **RMSE**: 6.02/6.34/6.85
MTGNN (SOTA)	**MAE**: 1.32/1.65/1.94 --- **MAPE**: 2.77/3.69/4.53 --- **RMSE**: 2.79/3.74/4.49	**MAE**: 2.17/2.89/4.02 --- **MAPE**: 5.03/6.93/9.93 --- **RMSE**: 4.01/5.84/8.78
TPGNN	**MAE**: 1.26/1.65/2.05 --- **MAPE: 2.56/3.47/4.40** --- **RMSE: 2.64/3.65**/4.58	**MAE: 2.12/2.72/3.22** --- **MAPE**: 5.00/6.73/8.22 --- **RMSE**: 4.05/5.45/6.56
TPGNN sign. Improvement %	MAPE: avg. 5.47% RMSE: 4.30%/2.41%/-	MAE: avg. 8.04% MAE: avg. 6.61% RMSE: -/2.33%/4.23%

Conclusion

- Included in table (**SOTA, improvements**)
- TPGNN fails to achieve SOTA performance on the exchange-rate data (ER). Authors think main reason is small sample size causing difficulties in capturing the dynamic variable dependence
- **Efficiency:** TPGNN is lightweight compared to other SOTA methods, as demonstrated by parameter scale comparisons.

Complexity Analysis

Not mentioned in the paper, approximate?

1. Graph propagation
2. Temporal Attention
3. Auto-Regressive Decoding

Implementation

Debug, debug, debug...

Tutorial

1. Prepare the data
    a. Create Correlation matrix and time stamp vector for your dataset
2. Configuration class adjustments, adapted to data
    a. F.e. Adjusting to intervalls (day_slot)
3. Hardcode distributed in different areas of the code, removed now
4. No gpu

Pseudo Code

```
main_stamp.py

    1. Import libraries and modules
    - PyTorch, NumPy, and custom modules (models, data, utils, etc.)

    2. Set random seed for reproducibility

    3. test function
```

```
    - Set model to evaluation mode
    - Calculate loss for the test data and return average loss

    4. train function
    - Parse options from configuration
    - Set up paths for logs, checkpoints,...
    - Load adjacency matrix  # spatial dependencies, not required


      - Load training, validation, test datasets
          - Create datasets using `STAGNN_stamp_Dataset`
        ( Here I did subset training data for faster training (10% of original data))
        - Data loaders for batch processing

        - masks for different temporal and spatial operations

        - L1 loss for training

        - Train the model
        - Initialize with configurations and masks
        - Optimizer (Adam) and learning rate scheduler

        - Start training loop for epochs:
        - For each batch, feed data to the model, calculate predictions and loss
        - Apply regularization (penalizing large differences in predictions)
        - Update model weights with backpropagation
        - Evaluate model on validation data
        - Save model checkpoints if validation loss improves

        - After training, evaluation on test data
        - Calculate performance metrics (MAE, MAPE, RMSE)
        - Print and save results
        - Track and log training time

If running as main program, start training using `fire.Fire()`
```

```
STAGNN_stamp.py

SrcProcess (Source Process):

    Prepares input data by applying different encodings:
        ConvExpandAttr for convolution
        SpatioEnc for spatial encoding
        TempoEnc for temporal encoding

TrgProcess (Target Process):

    Target processing, similar so Source Process
    Uses MLP for encoding/decoding

Encoder:

    EncoderLayer_stamp
    DecoderLayer to produce forecasted output
```

```
Timestamp:

    Embeds time information using embedding layer (nn.Embedding)
    Followed by temporal encoding (TempoEnc)

STAGNN_stamp:

    Core model combines all processing modules (SrcProcess, TrgProcess, timestamp, Encoder, D
    T4N (?)
    Computes loss based on prediction errors
    Applies a regularization term for adjacency matrix A
```

```
dataset.py

    transform function

            not used in our case

            Input:
            data: Tensor containing (energy) data
            train: Boolean whether function is for training
            opt: Configurations
            start: Starting index for splitting data.

            n_his, n_pred, n_route, day_slot, and T4N_step from opt
            compute n_days and slots (here was hardcode f.e.)
            slice data into chunks and reshape

            Output: Transformed tensors x and y for training or prediction


        # x is for n_his
        # y is for n_pred

        transform_time function    # same as transform but includes stamps

                    Input:
                    data: Tensor containing (energy) data
                    train: Boolean whether function is for training
                    opt: Configurations
                    start: Starting index for splitting data
                    time_stamp: Timestamps associated with the data

                  same as transform but also stamp tensor is sliced and reshaped

                 Output: Transformed tensors x, stamp, and y


         STAGNN_Dataset class

                     not used in our case, same as STAGNN_stamp_Dataset
                     also without timestamp information

                     uses transfrom function
```

```
STAGNN_stamp_Dataset class


            Input:
                   opt: Options and parameters
                   train: Boolean for training dataset
                   val: Boolean for validation dataset


           __init__:
               Load data and timestamp labels
               Split data into training, validation, or test datasets
               Scaling the data
               Transforms data using transform_time function
```

Annex

Table 1: Dataset statistics.

Datasets	# Samples	# Nodes	Sample Rate	Input Length	Output Length
Traffic	17,544	862	1 hour	168	1
Solar-Energy	52,560	137	10 minutes	168	1
Electricity	26304	321	1 hour	168	1
Exchange-Rate	7,588	8	1 day	168	1
PEMS-D7	12672	228	5 minutes	12	12
PEMS-Bay	52116	325	5 minutes	12	12

Dataset		Solar-Energy				Traffi				Electricity				Exchange-Rate			
		Horizon				Horizon				Horizon				Horizon			
Methods	Metric	3	6	12	24	3	6	12	24	3	6	12	24	3	6	12	24
VARMLP	RSE	0.1922	0.2679	0.4244	0.6841	0.5582	0.6579	0.6023	0.6146	0.1392	0.1620	0.1557	0.1274	0.0265	0.0394	0.0407	0.0578
VARMLP	CORR	0.9829	0.9655	0.9058	0.7149	0.8245	0.7695	0.7929	0.7891	0.8708	0.8389	0.8192	0.8679	0.8609	0.8725	0.8280	0.7675
GP	RSE	0.2259	0.3286	0.5200	0.7973	0.6082	0.6772	0.6406	0.5995	0.1500	0.1907	0.1621	0.1273	0.0239	0.0272	0.0394	0.0580
GP	CORR	0.9751	0.9448	0.8518	0.5971	0.7831	0.7406	0.7671	0.7909	0.8670	0.8334	0.8394	0.8818	0.8713	0.8193	0.8484	0.8278
RNN-GRU	RSE	0.1932	0.2628	0.4163	0.4852	0.5358	0.5522	0.5562	0.5633	0.1102	0.1144	0.1183	0.1295	0.0192	0.0264	0.0408	0.0626
RNN-GRU	CORR	0.9823	0.9675	0.9150	0.8823	0.8511	0.8405	0.8345	0.8300	0.8597	0.8623	0.8472	0.8651	0.9786	**0.9712**	0.9513	0.9223
LSTNet	RSE	0.1843	0.2559	0.3254	0.4643	0.4777	0.4893	0.4950	0.4973	0.0864	0.0931	0.1007	0.1007	0.0226	0.0280	0.0356	0.0449
LSTNet	CORR	0.9843	0.9690	0.9467	0.8870	0.8721	0.8690	0.8614	0.8588	0.9283	0.9135	0.9077	0.9119	0.9735	0.9658	0.9511	0.9354
TPA-LSTM	RSE	0.1803	**0.2347**	0.3234	0.4389	0.4487	0.4658	0.4641	0.4765	0.0823	0.0916	0.0964	0.1006	0.0174	**0.0241**	**0.0341**	**0.0444**
TPA-LSTM	CORR	0.9850	**0.9742**	0.9487	0.9081	0.8812	0.8717	0.8717	0.8629	0.9439	0.9337	0.9250	0.9133	0.9790	0.9709	**0.9564**	**0.9381**
MTGNN	RSE	**0.1778**	0.2348	0.3109	0.4270	0.4162	0.4754	**0.4461**	**0.4535**	0.0745	0.0878	0.0916	0.0953	0.0194	0.0259	0.0349	0.0456
MTGNN	CORR	**0.9852**	0.9726	0.9509	0.9031	0.8963	0.8667	0.8794	0.8810	**0.9474**	0.9316	0.9278	0.9234	0.9786	0.9708	0.9551	0.9372
TPGNN	RSE	0.1850	0.2412	**0.3059**	**0.3498**	**0.3989**	**0.4715**	0.4476	0.4696	**0.0627**	**0.0685**	**0.0699**	**0.0936**	**0.0174**	0.0250	0.0350	0.0458
TPGNN	CORR	0.9840	0.9716	**0.9529**	**0.9710**	**0.9232**	**0.8945**	**0.9028**	**0.8858**	0.9417	**0.9362**	**0.9285**	**0.9293**	**0.9792**	0.9687	0.9509	0.9306

Model	PEMS-BAY (Horizon 3/6/12)			PEMS-D7 (Horizon 3/6/12)		
	MAE	MAPE(%)	RMSE	MAE	MAPE(%)	RMSE
ARIMA	1.62/2.33/3.38	3.50/5.40/8.30	3.30/4.76/6.50	5.55/5.86/6.27	12.92/13.94/15.20	9.00/9.13/9.38
FC-LSTM	2.05/2.20/2.37	4.80/5.20/5.70	4.19/4.55/4.96	3.57/3.92/4.16	8.60/9.55/10.10	6.20/7.03/7.51
STGCN	1.39/1.84/2.42	3.00/4.22/5.58	2.92/4.12/5.33	2.25/3.03/4.02	5.26/7.33/9.85	4.04/5.70/7.64
DCRNN	1.38/1.74/2.07	2.90/3.90/4.90	2.95/3.97/4.74	2.25/2.98/3.83	5.30/7.39/9.85	4.04/5.58/7.19
StemGNN	1.52/1.94/2.45	3.38/4.58/6.03	3.06/4.07/5.04	2.94/3.66/4.66	7.63/9.66/12.58	5.05/6.35/8.00
Graph WaveNet	1.30/**1.63**/1.95	2.73/3.67/4.63	2.74/3.70/4.52	2.18/2.95/3.88	5.02/7.22/10.03	4.18/5.82/7.61
Informer	2.30/2.40/2.55	5.02/5.32/5.73	4.21/4.49/4.85	3.64/3.77/4.09	8.66/9.07/9.87	6.02/6.34/6.85
MTGNN	1.32/1.65/**1.94**	2.77/3.69/4.53	2.79/3.74/**4.49**	2.17/2.89/4.02	5.03/6.93/9.93	**4.01**/5.84/8.78
TPGNN	**1.26**/1.65/2.05	**2.56/3.47/4.40**	**2.64/3.65**/4.58	**2.12/2.72/3.22**	**5.00/6.73/8.22**	4.05/**5.45/6.56**

Tutorial beginning

Paper address the approximation gap between previous static graph (in GCN) and real world time-varying correlation.

propose TPGNN: dynamic variable correlation as a temporal matrix (adj) polynomial in two steps

,we use a set of time-varying coefficients and the matrix basis to construct a matrix polynomial for each time step.

step 1 : $\mathbf{A}$

step 2: correlation $\sigma(w_1\mathbf{A} + w_2\mathbf{A}^2 + w_3\mathbf{A}^3)$

Experiment design: six synthetic MTS datasets generated by a non-repeating random walk model.

Theoretical analysis:

Real World dataset: two traffic datasets & four benchmark datasets

Downtask: short-term and long-term MTS forecastings