Graph Deep Learning for Time Series and Spatiotemporal Data

Daniele **Zambon**

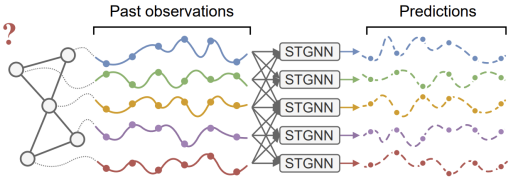
Graph Machine Learning Group (gmlg.ch)
The Swiss AI Lab IDSIA
Università della Svizzera italiana

Università di Padova, Italy · Spring 2026

Module

Latent Graph Learning

Dealing with lack of relational information



☹ Relational information is not always (or only partially) available,

☹ or might be ineffective in capturing spatial dynamics.

☺ Relational architectural biases can nonetheless be exploited

→ **extract a graph** from the time series or node attributes

- It can be interpreted as regularizing a spatial attention operator.

⚡ When possible, the learned graph should be sparse and relevant to solve the downstream task.

This task is found under different names:

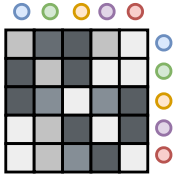
graph structure learning, latent graph learning, graph inference...

Extract graphs from data

Time-series similarities

Probably, the simplest approach to extract a graph from the time series is by computing time series similarity scores.

- Pearson correlation
- Correntropy
- Granger causality
- Kernels for time series
- . . .



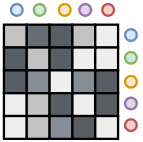
→ Thresholding might be necessary to obtain binary and sparse graphs.

Example: Correlation-based graph construction

Compute the Pearson correlation between any pair of time series $\boldsymbol{x}^i$ and $\boldsymbol{x}^j$:

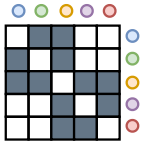
$$\Phi_{ij} = \frac{\mathrm{Cov}(\boldsymbol{x}^i, \boldsymbol{x}^j)}{\sigma(\boldsymbol{x}^i) \cdot \sigma(\boldsymbol{x}^j)}$$

and store them in matrix $\Phi \in \mathbb{R}^{N \times N}$ with $\Phi_{ij} \in [-1, 1]$,

$\boldsymbol{\Phi} =$ 

To obtain a (binary) adjacency matrix $\boldsymbol{A}$, we can

- Apply thresholding: $\boldsymbol{A}_{ij} = \mathbb{I}\{|\Phi_{ij}| > \tau\}$, for some $\tau \in (0, 1)$,
  - or take the top-k edges for each node,
  - ...
- Optionally, make it symmetric: $\boldsymbol{A} \leftarrow \max(\boldsymbol{A}, \boldsymbol{A}^\top)$.

$\widetilde{\boldsymbol{A}} =$ 

⚠ Threshold τ becomes an hyperparameter: too low → dense graph; too high → disconnected.

Inferring latent structures from time series

Model the graph as a latent variable determining the realizations of the time series.

- Typically relying on assumptions, such as of signal smoothness and of a diffusion process.

Dedicated loss functions are formulated and minimized, e.g.,

$$\text{trace}(\boldsymbol{X}^\top \boldsymbol{L}\boldsymbol{X}) = \frac{1}{2}\sum_{ij} \boldsymbol{A}_{i,j}\|\boldsymbol{X}_i - \boldsymbol{X}_j\|_2^2$$

constraining $\boldsymbol{L}$ (or $\boldsymbol{A}$) to be a Laplacian (adjacency matrix) and promoting sparsity.

$\rightarrow$ These approaches are commonly derived from a graph signal processing point of view.

[1] Dong *et al.*, "Learning Laplacian matrix in smooth graph signal representations", IEEE TSP 2016.
[2] Mateos *et al.*, "Connecting the dots: Identifying network structure via graph signal processing", IEEE SP Mag 2019.

Optimize the downstream task

Task-oriented latent graph learning

An integrated approach: learn the **relations** end-to-end with the downstream task

$\rightarrow$ e.g., by minimizing the forecasting error (MAE, MSE...).

$$\begin{aligned}\ell\left(\widehat{\boldsymbol{Y}}, \boldsymbol{Y}\right) &= \ell\left(\mathcal{F}_\theta(\boldsymbol{X}_{t-W:t}, \boldsymbol{A}), \boldsymbol{X}_{t:t+H}\right) \\ &= \ell\left(\mathcal{F}_\theta(\boldsymbol{X}, \boldsymbol{A}), \boldsymbol{Y}\right)\end{aligned}$$

Two different formulations:

1. learning directly an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$;
2. learning a probability distribution over graphs $p_\Phi(\boldsymbol{A})$

⚠ One key challenge is keeping both $\boldsymbol{A}$ and the subsequent computations sparse.
$\rightarrow$ non-trivial with gradient-based optimization.

Direct approach

A direct approach consists in learning an adjacency matrix $\boldsymbol{A}$ as function $\xi(\cdot)$ of edge scores $\Phi \in \mathbb{R}^{N \times N}$ as

$$\boldsymbol{A} = \xi(\Phi)$$

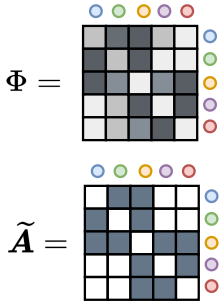
Edge scores Φ can be

- a matrix of learnable model parameters,
- obtained as a function of the inputs and/or other parameters:

$$\Phi = \Phi(\boldsymbol{X}, \widetilde{\Phi}).$$

Function $\xi(\cdot)$ can enforce structures on $\boldsymbol{A}$, like,

$\rightarrow$ making $\boldsymbol{A}$ binary, a k-NN graph, a tree...



Learning a binary variable with gradient descent

We would like to optimize a binary edge variable $\boldsymbol{A}_{ij} \in \{0, 1\}$ with gradient descent.

But a hard binary choice is a step function of its score Φ_{ij}:

$$\boldsymbol{A}_{ij} = \mathbb{I}[\Phi_{ij} > 0].$$

This creates the core optimization issue:

$$\frac{\partial \boldsymbol{A}_{ij}}{\partial \Phi_{ij}} = 0 \quad \text{almost everywhere}$$

(and undefined at $\Phi_{ij} = 0$).

⚠ This is one or the reasons why learning sparse binary structures end-to-end is non-trivial with standard gradient-based training.

Pro & Cons of the direct approach

- ☺ Easy to implement.
- ☺ Many possible parametrizations.
- ☺ Edge scores are usually easy to learn end-to-end.

- ☹ It often results in dense computations with $\mathcal{O}(N^2)$ complexity.
- ☹ Sparsifying $\boldsymbol{A}$ results in sparse gradients.
- ☹ Encoding prior structural information requires smart parametrizations.

Probabilistic approaches

Probabilistic methods

In this context, probabilistic methods aim at learning a probability distribution p_Φ over graphs $\boldsymbol{A}$ end-to-end with the rest of the model, e.g.,

$$\widehat{\boldsymbol{Y}} = \mathcal{F}_\theta(\boldsymbol{X}, \boldsymbol{A}), \qquad \text{with } \boldsymbol{A} \sim p_\Phi(\boldsymbol{A}).$$

- ☺ Graphs are discrete objects $\boldsymbol{A} \in \{0,1\}^{N \times N}$; (sparse and efficient computation)
- ☹ Gradient-based optimization not always trivial.
- ☺ Edge probabilities provide some degree of model interpretability.

Examples of distributions

Different parametrizations of p_Φ allow for embedding graph structural priors on the sampled graphs, e.g., edge density, bound node degrees.

Graphs of independent edges

For every edge (i, j) $\quad \boldsymbol{A}_{i,j} \sim \text{Bernoulli}(\sigma(\Phi_{i,j}))$.

Fixed-degree graphs

For each node i, sample w/o replacement k nodes from $\quad \text{Categorical}\left(\text{SoftMax}(\Phi_{i,1}, \ldots, \Phi_{i,N})\right)$.

[3] Kazi *et al.*, "Differentiable graph module (dgm) for graph convolutional networks", IEEE TPAMI 2022.
[4] Cini, Zambon, and Alippi, "Sparse graph learning from spatiotemporal time series", JMLR 2023.

Learning graph distributions

Probability distribution $p_\Phi(\boldsymbol{A})$ over graphs $\boldsymbol{A}$ can be learned **end-to-end** with the rest of the model

$$\widehat{\boldsymbol{X}}_{t:t+H} = \mathcal{F}_\theta(\boldsymbol{X}_{t-W:t}, \boldsymbol{A}) \qquad \widehat{\boldsymbol{Y}} = \mathcal{F}_\theta(\boldsymbol{X}, \boldsymbol{A}) \quad \textit{(for simplicity)}$$

optimizing a training loss $\mathcal{L}(\theta, \Phi)$ on both the model (θ) and the graph parameters (Φ).

Training losses include

$$\mathcal{L}(\theta, \Phi) = \mathbb{E}_{\boldsymbol{A}\sim p_\Phi}\left[\ell(\widehat{\boldsymbol{Y}}, \boldsymbol{Y})\right], \qquad \mathcal{L}(\theta, \Phi) = \ell\left(\mathbb{E}_{\boldsymbol{A}\sim p_\Phi}\left[\widehat{\boldsymbol{Y}}\right], \boldsymbol{Y}\right),$$

focusing on point predictions, but also

$$\mathcal{L}(\theta, \Phi) = \Delta\left(p_\Phi(\widehat{\boldsymbol{Y}}), p(\boldsymbol{Y})\right).$$

by means of divergence measures Δ on the predictive distribution.

Learning via gradient-based optimization

Consider loss function

$$\mathcal{L}(\theta, \Phi) = \mathbb{E}_{p_\Phi}[\ell(\widehat{\boldsymbol{Y}}, \boldsymbol{Y})].$$

Gradient-based optimization requires computing

$$\nabla_\theta \, \mathbb{E}_{p_\Phi}[\ell(\mathcal{F}_\theta(\boldsymbol{X}, \boldsymbol{A}), \boldsymbol{Y})] \qquad \text{and} \qquad \nabla_\Phi \, \mathbb{E}_{p_\Phi}[\ell(\mathcal{F}_\theta(\boldsymbol{X}, \boldsymbol{A}), \boldsymbol{Y})].$$

$\rightarrow$ It requires differentiating w.r.t. the parameters Φ of the integrated distribution.

- ☹ Analytical computations is often unfeasible;
- ⚠ Monte Carlo approximations require care.

Monte Carlo gradient estimators

Consider loss function $\mathcal{L}(\theta, \Phi) = \mathbb{E}_{p_\Phi}[\ell(\mathcal{F}_\theta(\boldsymbol{X}, \boldsymbol{A}), \boldsymbol{Y})]$.

One approach is to reparametrize $\boldsymbol{A} \sim p_\Phi(\boldsymbol{A})$ as:

$$\boldsymbol{A} = g(\Phi, \boldsymbol{\varepsilon}), \qquad \boldsymbol{\varepsilon} \sim p(\boldsymbol{\varepsilon})$$

decoupling parameters Φ from the random component $\boldsymbol{\varepsilon}$:

$$\begin{aligned}\nabla_\Phi \mathcal{L}(\theta, \Phi) &= \mathbb{E}_{\boldsymbol{\varepsilon}}\left[\nabla_\Phi \ell\big(\mathcal{F}_\theta(\boldsymbol{X}, \ \boldsymbol{A}) \quad , \boldsymbol{Y}\big)\right] \\ &= \mathbb{E}_{\boldsymbol{\varepsilon}}\left[\nabla_\Phi \ell\big(\mathcal{F}_\theta(\boldsymbol{X}, g(\Phi, \boldsymbol{\varepsilon})), \boldsymbol{Y}\big)\right].\end{aligned}$$

Now, we can approximate it via Monte Carlo

$$\ldots \quad \approx \frac{1}{M} \sum_{m=1}^{M} \nabla_\Phi \ell\big(\mathcal{F}_\theta(\boldsymbol{X}, g(\Phi, \boldsymbol{\varepsilon}_m)), \boldsymbol{Y}\big)$$

with $\{\boldsymbol{\varepsilon}_m\}_m^M$ i.i.d. samples from $p(\boldsymbol{\varepsilon})$.

Score-function gradient estimator

Consider loss function $\mathcal{L}(\theta, \Phi) = \mathbb{E}_{p_\Phi}[\ell(\mathcal{F}_\theta(\boldsymbol{X}, \boldsymbol{A}), \boldsymbol{Y})]$.

Conversely, score-function (SF) gradient estimators rely on the relation

$$\nabla_\Phi \mathbb{E}_{p_\Phi}[\ell(\mathcal{F}_\theta(\boldsymbol{X}, \boldsymbol{A}), \boldsymbol{Y})] = \mathbb{E}_{p_\Phi}[\ell(\mathcal{F}_\theta(\boldsymbol{X}, \boldsymbol{A}), \boldsymbol{Y}) \nabla_\Phi \log p_\Phi(\boldsymbol{A})]$$

In this form, we can approximate it via Monte Carlo

$$\ldots \approx \frac{1}{M} \sum_{m=1}^{M} \ell(\mathcal{F}_\theta(\boldsymbol{X}, \boldsymbol{A}_m), \boldsymbol{Y}) \nabla_\Phi \log p_\Phi(\boldsymbol{A}_m)$$

with $\{\boldsymbol{A}_m\}_m^M$ i.i.d. samples from $p_\Phi(\boldsymbol{A})$.

Pros and Cons

Reparameterization trick

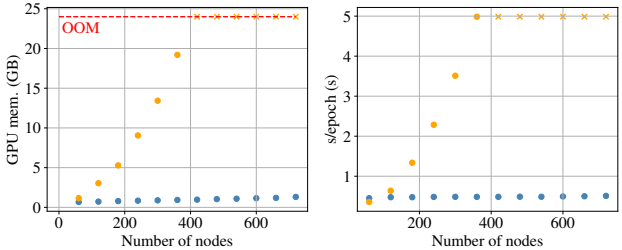
- ☺ Practical and easy to implement,
- ☹ rely on continuous relaxations and make subsequent computations scale with $\mathcal{O}(N^2)$.

Score-function gradient estimators

- ☹ suffer from high variance (use variance reduction techniques),
- ☺ allow to keep computations sparse through the model.

[5] Kipf *et al.*, "Neural relational inference for interacting systems", ICML 2018.
[4] Cini, Zambon, and Alippi, "Sparse graph learning from spatiotemporal time series", JMLR 2023.

Computational efficiency



Score-based gradient estimators:

$$\nabla_\Phi \mathcal{L}(\theta, \Phi) = \mathbb{E}_{\boldsymbol{A} \sim p_\Phi} \left[\ell(\widehat{\boldsymbol{Y}}, \boldsymbol{Y}) \nabla_\Phi \log p_\Phi(\boldsymbol{A}) \right]$$

☺ are computationally efficient
- do not rely on continuous relaxation of discrete random variables to compute ∇_Φ;
- allow for sparse message passing to compute $\widehat{\boldsymbol{Y}}$ (and, in turn, of $\ell(\widehat{\boldsymbol{Y}}, \boldsymbol{Y})$) by rely on sparse matrices $\boldsymbol{A}$.

☹ can be sample inefficient due to the high variance of the gradient estimates.

[4] Cini, Zambon, and Alippi, "Sparse graph learning from spatiotemporal time series", JMLR 2023.

Uncertainty quantification

While probabilistic models have been used to enable learning of discrete variables (graph edges), the associated edge probabilities can carry information about the relevance of the associated connections.

→ It enables some **degree of interpretability** and better informed **decision-making**.

⚠ Assessing the calibration of latent variables is hard on real data.

→ This is due to their latent nature, for which observations are not available.

Studies provide some learning guarantees, e.g.,

→ Minimizing appropriate divergence measures $\Delta\left(p_{\theta,\Phi}(\widehat{\boldsymbol{Y}}), p(\boldsymbol{Y})\right)$ of the data and predictive distributions can enable calibration of the $p_\Phi(\boldsymbol{A})$.

[6] Gray *et al.*, "Bayesian inference of network structure from information cascades", IEEE TSIPN 2020.
[7] Manenti, Zambon, and Alippi, "Learning Latent Graph Structures and their Uncertainty", ICML 2025.

Key takeaways

- Graph structures can be learned directly from data when they are not explicitly given.
- Probabilistic models allow for learning discrete graphs ensuring computational efficiency.
- Calibration of the latent graph is possible and provide insights on model's prediction and trustworthiness.

References i

[1] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst, **"Learning laplacian matrix in smooth graph signal representations,"** *IEEE Transactions on Signal Processing*, vol. 64, no. 23, pp. 6160–6173, 2016.

[2] G. Mateos, S. Segarra, A. G. Marques, and A. Ribeiro, **"Connecting the dots: Identifying network structure via graph signal processing,"** *IEEE Signal Processing Magazine*, vol. 36, no. 3, pp. 16–43, 2019.

[3] A. Kazi, L. Cosmo, S.-A. Ahmadi, N. Navab, and M. M. Bronstein, **"Differentiable graph module (dgm) for graph convolutional networks,"** *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 2, pp. 1606–1617, 2022.

[4] A. Cini, D. Zambon, and C. Alippi, **"Sparse graph learning from spatiotemporal time series,"** *Journal of Machine Learning Research*, vol. 24, no. 242, pp. 1–36, 2023.

[5] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel, **"Neural relational inference for interacting systems,"** in *International conference on machine learning*, PMLR, 2018, pp. 2688–2697.

[6] C. Gray, L. Mitchell, and M. Roughan, **"Bayesian inference of network structure from information cascades,"** *IEEE Transactions on Signal and Information Processing over Networks*, vol. 6, pp. 371–381, 2020.

References ii

[7] A. Manenti, D. Zambon, and C. Alippi, **“Learning latent graph structures and their uncertainty,”** in *Proceedings of the 42nd International Conference on Machine Learning*, vol. 267, 13–19 Jul 2025, pp. 42 882–42 901. [Online]. Available: https://proceedings.mlr.press/v267/manenti25a.html.

[8] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, **“Graph wavenet for deep spatial-temporal graph modeling,”** in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 2019, pp. 1907–1913.

[9] L. Bai, L. Yao, C. Li, X. Wang, and C. Wang, **“Adaptive graph convolutional recurrent network for traffic forecasting,”** *Advances in Neural Information Processing Systems*, vol. 33, pp. 17 804–17 815, 2020.