May 26, 2025

# Graph Structure Learning


Alessandro **Manenti**

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

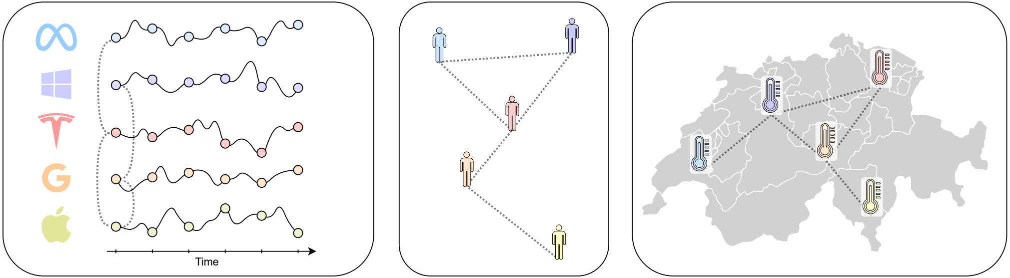
# Introduction

# Introduction

GNNs use an adjacency matrix $A$ as an effective inductive bias.

☹ $A$ might be unknown or of coarsely available

Some examples:




Can we learn relationships from data?

# Introduction

☺ It is possible to learn relations from data

Graph Structure Learning (GSL) investigates methods to infer relational structures from data.

GSL effectiveness depends on:

1. The presence of a "true" underlying relational structure.
2. The number of available data

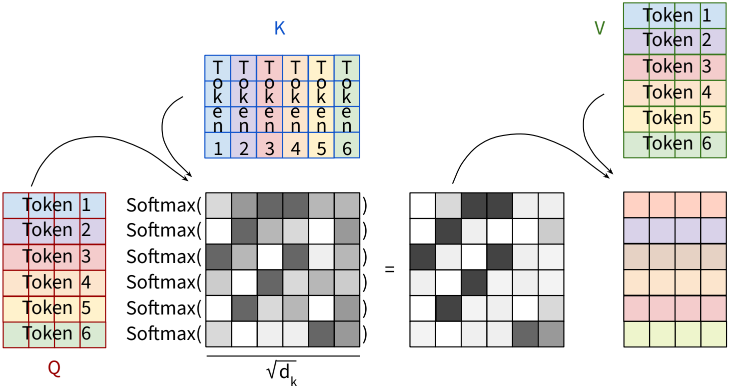
The Transformer learns relational structures from data too:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad \text{with: } Q/K/V = W_Q/W_K/W_V \cdot \boldsymbol{X}$$

Q: Where is the relational structure here?

# Attention mechanism

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad \text{with: } Q/K/V = W_Q/W_K/W_V \cdot \boldsymbol{X}$$



# Overview

| **Using original structure** or **Adjacency matrix initialization** | **Graph structure learning** | **Transformer-based techniques** |
|---|---|---|
| · Pre-processing techniques used to infer an initial, static topology | · Techniques that parametrize and optimize the structure to solve a task | · Techniques based on the attention mechanism |

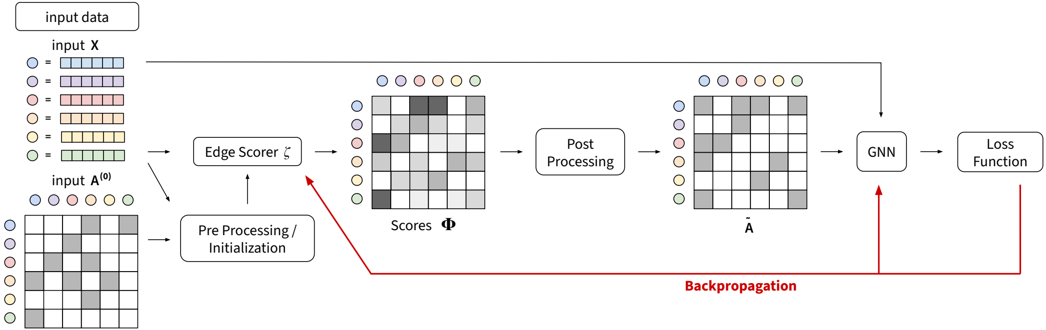
Limited data → Abundant data

Computationally efficient → Computationally expensive

- For further reading, refer to [1], [2]

[1] Zhiyao *et al.*, "Opengsl: A comprehensive benchmark for graph structure learning" 2024.
[2] Fatemi *et al.*, "Ugsl: A unified framework for benchmarking graph structure learning" 2023.

# General GSL Framework




- Input: $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ and, optionally, an initial adjacency matrix $\boldsymbol{A}^{(0)} \in \mathbb{R}^{N \times N}$
- Trainable modules: Edge Scorer and GNN
- Loss function: Usually designed to solve a (self-)supervised task

# Structure initialization techniques

# Structure initialization techniques

- Extract (or modify) an adjacency matrix independently from the downstream task.
- Different techniques rely on different assumptions.
- ☺ Topological structures obtained from this pre-processing can be used as initialization for the GSL edge scorer.

Some examples include:

1. Pearson Correlation.
2. Granger causality.
3. Pairwise input similarity.
4. Dirichlet Energy Minimization.
5. Rewiring techniques (if initial $\boldsymbol{A}^{(0)}$ given).

# Pearson correlation

The Pearson correlation coefficient is a measure of the linear relationship between two variables.

$$\rho \equiv \frac{\mathrm{Cov}(\boldsymbol{X}_i, \boldsymbol{X}_j)}{\sigma_{\boldsymbol{X}_i}\sigma_{\boldsymbol{X}_j}}$$

For real-world data the formula is:

$$\hat{\rho} = \frac{\sum_{d=1}^{D}(\boldsymbol{X}_{i,d} - \overline{\boldsymbol{X}_i})(\boldsymbol{X}_{j,d} - \overline{\boldsymbol{X}_j})}{\sqrt{\sum_{d=1}^{D}(\boldsymbol{X}_{i,d} - \overline{\boldsymbol{X}_i})^2 \sum_{d=1}^{D}(\boldsymbol{X}_{j,d} - \overline{\boldsymbol{X}_j})^2}}$$

An adjacency matrix $\boldsymbol{A}$ can be built from $\hat{\rho}$.

# Pearson correlation

- $\rho$ is a normalized value: $-1 \leq \rho \leq 1$
- The magnitude of $\rho$ indicates the strength of the relationship,
- The sign indicates its direction.
- Be aware that it is not perfect! (see Figure)

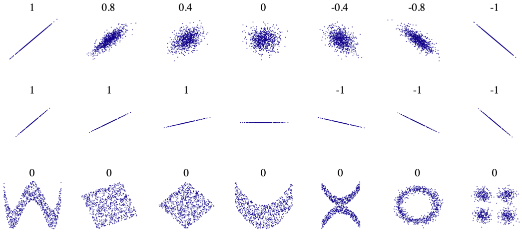



**Figure 1:** Pearson correlation for different sets of $(x, y)$ points. Image from Wikipedia

# Granger causality

For Granger causality, we restrict $\boldsymbol{X}$ to be a set of time series.

- Granger causality test exists if time series $\boldsymbol{X}_i$ "causes" time series $\boldsymbol{X}_j$.
- Test whether past values of $\boldsymbol{X}_i$ contain useful information for predicting $\boldsymbol{X}_j$, beyond the information contained in past values of $\boldsymbol{X}_j$ alone.

Build two linear models:

**Restricted model** (without $\boldsymbol{X}_j$)

$$\boldsymbol{X}_{i,t} = \alpha_0 + \sum_{a=1}^{p} \alpha_i \, \boldsymbol{X}_{i,t-a} + \epsilon_t$$

**Unrestricted model** (with $\boldsymbol{X}_j$)

$$\boldsymbol{X}_{i,t} = \alpha_0 + \sum_{a=1}^{p} \alpha_a \, \boldsymbol{X}_{i,t-a} + \sum_{b=1}^{p} \gamma_b \, \boldsymbol{X}_{j,t-b} + \eta_t$$

The Granger causality test assesses whether $\boldsymbol{X}_j$ helps to predict $\boldsymbol{X}_i$.

# Granger causality

Formulate the null hypothesis $H_0$ and alternative hypothesis $H_1$:

$$\begin{aligned} H_0 &: \quad \gamma_1 = \gamma_2 = \cdots = \gamma_p = 0 \\ H_1 &: \quad \text{At least one } \gamma_b \neq 0 \text{ for some } b \in \{1, 2, \ldots, p\} \end{aligned}$$

$H_0$: none of the past values of $\boldsymbol{X}_j$ contain linear predictive information about the current value of $\boldsymbol{X}_i$.

To test $H_0$, compare the fit of the restricted and unrestricted models. This is typically done using an F-test:

1. Compute the residual sum of squares (RSS) for both the restricted model ($\text{RSS}_R$) and the the unrestricted model ($\text{RSS}_U$)
2. Compute the F-statistic:
   $$\frac{(\text{RSS}_R - \text{RSS}_U)/p}{\text{RSS}_U/(T - 2p - 1)}$$
   Under $H_0$, the F-statistic follows an F-distribution with $p$ and $(T - 2p - 1)$ degrees of freedom.
3. Check if the p-value is below a predetermined significance level.

# Pairwise input similarity

- The most common initialization technique if $\boldsymbol{A}^{(0)}$ is not given.
- Assumption: similar inputs should be connected.
- Input similarity can be defined in different ways. For example:
  1. Cosine similarity $\left(\frac{\boldsymbol{X}_i \cdot \boldsymbol{X}_j}{||\boldsymbol{X}_i||||\boldsymbol{X}_j||}\right)$
  2. Decreasing function of a distance $\mathbf{d}$ $\left(\text{e.g., } \frac{1}{\mathbf{d}(\boldsymbol{X}_i, \boldsymbol{X}_j)}\right)$
  3. Kernels $\left(\text{e.g., the RBF kernel: } e^{-||\boldsymbol{X}_i - \boldsymbol{X}_j||^2}\right)$

☺ Easy to implement.

☺ Computationally and memory efficient.

☹ If $\boldsymbol{A}^{(0)}$ is not perfected afterwards, performance on the considered task may not exceed that of a structure agnostic baseline [3].

[3] Errica, "On class distributions induced by nearest neighbor graphs for node classification of tabular data" 2024.

# Dirichlet Energy Minimization

- Graph signal processing perspective. [4], [5]
- Often considers symmetric and non-negative matrices. [6]
- Smoothness assumption: in amenable graph structures the graph signal varies smoothly across edges.

Define the Dirichlet Energy:

$$\mathcal{E} = \frac{1}{2}\sum_{i,j} \boldsymbol{A}_{ij}||\boldsymbol{X}_i - \boldsymbol{X}_j||^2 \equiv \frac{1}{2}\sum_{i,j} \boldsymbol{A}_{ij}\boldsymbol{Z}_{ij}$$

Minimization problem for smooth signals:

$$\boldsymbol{A}^{(0)} = \underset{\boldsymbol{A}}{\operatorname{argmin}} \left\{ \frac{1}{2}\sum_{i,j} \boldsymbol{A}_{ij}\boldsymbol{Z}_{ij} \right\}$$

Q: What is the trivial solution of this minimization problem?

[4] Dong *et al.*, "Learning Laplacian matrix in smooth graph signal representations" 2016.
[5] Dong *et al.*, "Learning graphs from data: A signal representation perspective" 2019.
[6] Kalofolias, "How to learn a graph from smooth signals" 2016.

# Dirichlet Energy Minimization

- An additional term $f(\boldsymbol{A})$ imposes prior information and avoids converging towards the trivial solution.
- The complete minimization problem becomes:

$$\boldsymbol{A}^{(0)} = \underset{\boldsymbol{A}}{\text{argmin}} \left\{ \frac{1}{2} \sum_{i,j} \boldsymbol{A}_{ij} \boldsymbol{Z}_{ij} + \lambda f(\boldsymbol{A}) \right\}$$

☺ The Dirichlet Energy Minimization problem and provides a theoretical framework to different input similarity techniques. For example, if:

$$f(\boldsymbol{A}) = 2\frac{\sigma^2}{\lambda} \sum_{ij} \boldsymbol{A}_{ij} (\log(\boldsymbol{A}_{ij}) - 1)$$

the solution to the minimization problem is a RBF initialization $\boldsymbol{A}_{ij}^{(0)} = e^{-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}}$

☺ Interpretable assumptions embedded in $f$

☺ Rich literature present

☹ Less straightforward to implement (and optimize)

# Rewiring techniques

- GNNs suffer from oversmoothing and oversquashing [7]
- Rewiring modifies the initial connectivity $\boldsymbol{A}^{(0)}$ to alleviate those problems. [8]

**Oversmoothing**: repeated rounds of message passing make node representations converge to similar embeddings.

Q: Connect the Dirichlet energy to oversmoothing: how does it change adding more GNN layers?

[7] Rusch *et al.*, "A survey on oversmoothing in graph neural networks" 2023.
[8] Attali *et al.*, "Rewiring Techniques to Mitigate Oversquashing and Oversmoothing in GNNs: A Survey" 2024.

# Rewiring techniques

**Oversquashing**: exponential loss of information increases with the number of GNN layers employed.

Notation:

- $h_i^{(\ell)}$: representation of node $i$ at layer $\ell$.
- $\hat{\boldsymbol{A}}$: normalized augmented adjacency matrix.

Given two nodes $i$ and $j$ at distance $r$, it has been shown [9]:

$$\left|\frac{\partial h_i^{(r)}}{\partial x_j}\right| \leq (K)^r (\hat{\boldsymbol{A}}^r)_{ij} \qquad \text{with } K \text{ being a GNN-specific constant}$$

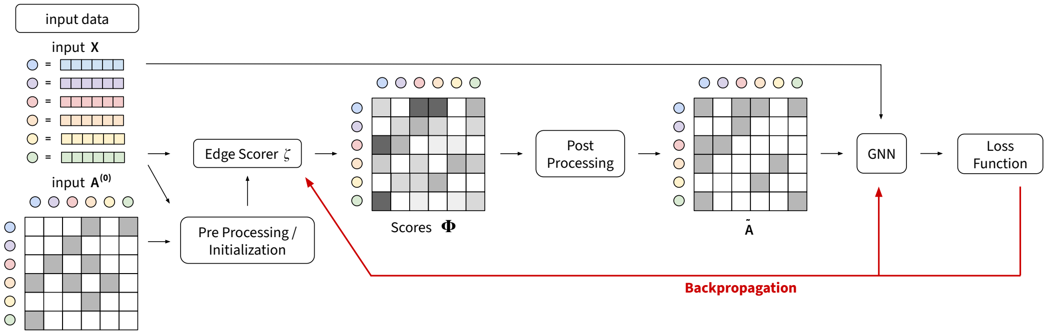
☺ Changing the graph structure can alleviate both.

- [9] proposes to iteratively add and remove edges via the Stochastic Discrete Ricci Flow algorithm.
- Some rewiring techniques completely ignore the original structure [10].

[9] Topping *et al.*, "Understanding over-squashing and bottlenecks on graphs via curvature" 2021.
[10] Attali *et al.*, "Delaunay Graph: Addressing Over-Squashing and Over-Smoothing Using Delaunay Triangulation" 2024.

# Edge Scorer

# General GSL Framework



# Edge Scorer

- An edge scorer is a parametric function $\xi_\theta(\boldsymbol{X}, \boldsymbol{A})$ that returns relational structures $\boldsymbol{\Phi}$, often modeled as pairwise scores between inputs.
- Edge Scorer's parameters $\theta$ can be trained on the considered downstream task.

An edge scorer should:

- align, whereas possible, with physical model: Are scores input-dependent? Should complex relationships be considered?
- be designed having in mind constraints set by the problem. How many nodes are present? How much data is available?

Edge Scorer's parameters can often be initialized using extracted adjacency matrices.

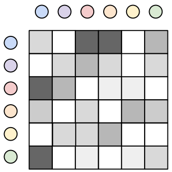
# Lookup table

Assume a fixed and input-independent graph structure $\longrightarrow \xi_\theta(\boldsymbol{X}, \boldsymbol{A}) = \xi_\theta$.

## $\mathbf{N} \times \mathbf{N}$ table

The function $\xi_\theta$ is a table of parameters:

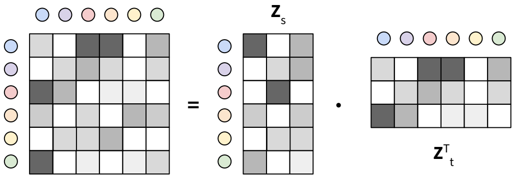
$$\xi_\theta = \boldsymbol{\Phi} \in \mathbb{R}^{N \times N}$$




## Embedding factorization

Parameters contained in node embeddings:

$$\xi_\theta = \boldsymbol{\Phi} = \boldsymbol{Z}_s \boldsymbol{Z}_t^T \text{ with } \boldsymbol{Z}_. \in \mathbb{R}^{N \times d}$$

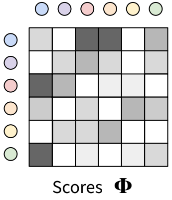



☺ Finer control

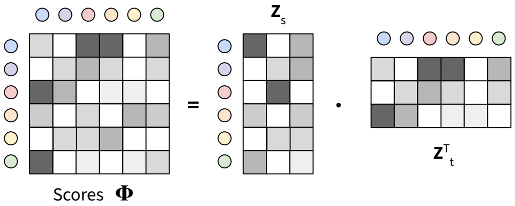
☺ More parameter efficient

# Lookup table

**$\mathrm{N} \times \mathrm{N}$ table**




**Embedding factorization**




- ☺ Common choice in the literature [11]–[14]
- ☺ Easy to implement and learn
- ☹ May oversimplify the problem

[11] Franceschi *et al.*, "Learning discrete structures for graph neural networks" 2019.
[12] Wu *et al.*, "Graph wavenet for deep spatial-temporal graph modeling" 2019.
[13] Cini *et al.*, "Sparse Graph Learning from Spatiotemporal Time Series" 2023.
[14] Manenti *et al.*, "Learning Latent Graph Structures and their Uncertainty" 2024.

# Input dependent

The Edge Scorer $\xi_\theta(\boldsymbol{X}, \boldsymbol{A})$ is a function, enabling different inductive biases [2], [15], [16]:

- Some methods simply use a MLP
- Some others employ a Graph Neural Networks
- Others use simple attention-based architectures

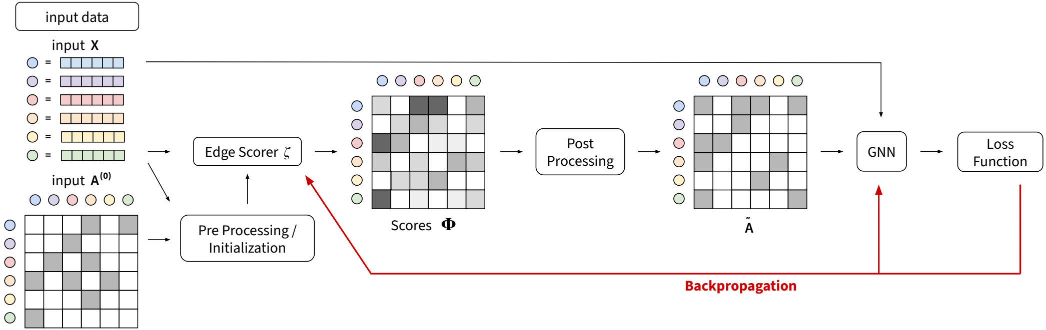
⚠ Iterative score updates and GNN processing blur the distinction between the Edge Scorer and GNN. In those scenarios, a clear decomposition may not be possible.

As a general rule: keep things simple!

[2] Fatemi *et al.*, "Ugsl: A unified framework for benchmarking graph structure learning" 2023.
[15] Wang *et al.*, "Dynamic graph cnn for learning on point clouds" 2019.
[16] Kazi *et al.*, "Differentiable graph module (dgm) for graph convolutional networks" 2022.

# Post-processing techniques & Loss functions

# General GSL Framework



# Post-processing techniques

The score matrix $\mathbf{\Phi}$ is transformed into an adjacency matrix $\tilde{A}$ to enforce desired properties.

Common objectives include:

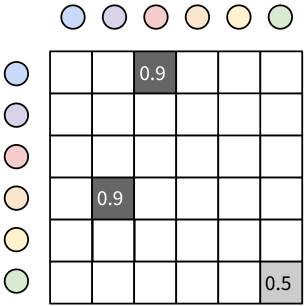
- Training facilitation: row normalization, value clamping, etc.
- Enforcement of structures: symmetrization, minimum spanning tree construction, etc.
- Sparsification: top-k selection, Bernoulli sampling, thresholding, etc.

Specific application requirements often necessitate post-processing techniques.

⚠ Post-processing can introduce unwished consequences.

Let's focus on sparsification techniques, as it is a desirable property.

# Sparse matrices




- A sparse matrix is a matrix in which the majority of elements are zero.
- Sparsity of a matrix = percentage of zero elements.

Q: Why do you think sparse matrices are desirable?

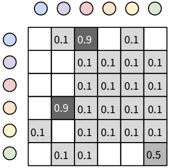
- Most common sparse representation of adjacency matrices in GDL is the COO (coordinate) format: two tensors, one for non-zero indices location and the other for corresponding values:
  e.g., `indices` $= [[0, 3, 5], [2, 1, 5]]$ `values` $= [0.9, 0.9, 0.5]$
- Other possibilities: CSR, CSC, BSR, BSC, … formats

# Sparse Matrices

Q: What is the computational complexity of a dense GCN layer $\boldsymbol{X}' = \boldsymbol{A}\boldsymbol{X}\boldsymbol{W}$ ?

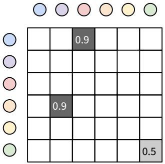
Q: What is the computational complexity of the same GCN layer with sparse matrix multiplications?

Two post-processing techniques that enforce sparsity:



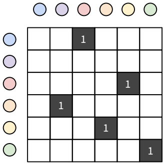

**Thresholding**

Keep edges if score $>$ threshold



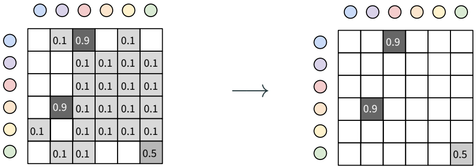

**Bernoulli sampling**

Treat scores as logits to sample from



# Thresholding

- Thresholding involves selecting a threshold hyperparameter $\tau$ and zeroing entries for which $\Phi_{ij} < \tau$.



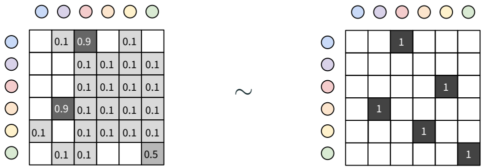

- Can control sparsity level
- Easy to implement
- Biased gradient

**Q:** Why is the gradient biased?

- Other sparsification methods, such as top-k or top-p selection, exhibit similar advantages and disadvantages.

# Bernoulli sampling

- Sample each edge with probability $\mathbf{\Phi}_{ij}$ (or sigmoid($\mathbf{\Phi}_{ij}$)).




- Offers an inherently probabilistic framework.
- Gradient propagation in stochastic operations - e.g. VAEs - is challenging. In VAEs problem was solved with the reparameterization trick.
- Issues arise as gradients are computed with respect to $\mathbf{\Phi}$:

$$\nabla_{\mathbf{\Phi}} \mathbb{E}_{A \sim P_{\mathbf{\Phi}}} [\mathcal{L}(A, \boldsymbol{X})]$$

# Reparameterization trick

- Direct sampling from a distribution (e.g., Gaussian) introduces a non-differentiable operation, blocking gradient flow.
- Reparameterization trick solves this problem separating the stochastic nature from the trainable parameters

1. Express the sampled variable $\hat{\boldsymbol{A}}$ as a deterministic function of trainable parameters $\boldsymbol{\Phi}$ and a random variable $\epsilon$.
2. Example (Gaussian): $\hat{\boldsymbol{A}} = \mu(\boldsymbol{\Phi}) + \sigma(\boldsymbol{\Phi}) \odot \epsilon$, where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$.
   $\mu(\boldsymbol{\Phi})$ represents the mean tensor, parameterized by $\boldsymbol{\Phi}$.
   $\sigma(\boldsymbol{\Phi})$ represents the standard deviation tensor, parameterized by $\boldsymbol{\Phi}$.
   $\odot$ is the element wise multiplication.

☹ Being Bernoulli random variables discrete, the reparameterization is not applicable.

# Bernoulli Sampling

- Issue arises as gradients are calculated with respect to $\mathbf{\Phi}$, the parameter vector defining the distribution:

$$\nabla_{\mathbf{\Phi}} \mathbb{E}_{A \sim P_{\mathbf{\Phi}}} [\mathcal{L}(A, \boldsymbol{X})]$$

- Different possible gradient estimators for Bernoulli Random Variables [17]:
  1. Straight-Through gradient estimator (treat discrete sample as identity in backward pass) [18]
  2. Gumbel-Softmax trick (continuous relaxation of Bernoulli) [19]

☹ Both methods need dense computation or biased gradient estimation.

  3. REINFORCE and/or Score-Function gradient estimator. [20], [21].

[17] Mohamed *et al.*, "Monte carlo gradient estimation in machine learning" 2020.
[18] Bengio *et al.*, "Estimating or propagating gradients through stochastic neurons for conditional computation" 2013.
[19] Jang *et al.*, "Categorical Reparametrization with Gumble-Softmax" 2017.
[20] Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning" 1992.
[21] Sutton *et al.*, "Policy gradient methods for reinforcement learning with function approximation" 1999.

# Bernoulli Sampling - REINFORCE

The score function gradient estimator directly approximates the gradient of an expectation by leveraging the log-likelihood trick to enable gradient computation through discrete random variables.

$$\begin{aligned}
\nabla_{\mathbf{\Phi}} \mathbb{E}_{A \sim P_{\mathbf{\Phi}}}[\mathcal{L}(A, \boldsymbol{X})] &= \nabla_{\mathbf{\Phi}} \int \mathcal{L}(A, \boldsymbol{X}) P_{\mathbf{\Phi}}(A) dA \\
&= \int \mathcal{L}(A, \boldsymbol{X}) \nabla_{\mathbf{\Phi}} P_{\mathbf{\Phi}}(A) dA \\
&= \int \mathcal{L}(A, \boldsymbol{X}) P_{\mathbf{\Phi}}(A) \nabla_{\mathbf{\Phi}} \log P_{\mathbf{\Phi}}(A) dA \\
&= \mathbb{E}_{A \sim P_{\mathbf{\Phi}}}[\mathcal{L}(A, \boldsymbol{X}) \nabla_{\mathbf{\Phi}} \log P_{\mathbf{\Phi}}(A)] \\
&\approx \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(A_i, \boldsymbol{X}) \nabla_{\mathbf{\Phi}} \log P_{\mathbf{\Phi}}(A_i)
\end{aligned}$$

☺ Sparse computations and unbiased gradient estimates.

☹ High variance (slow or no convergence). It can be mitigated using control variates.

# Bernoulli Sampling - REINFORCE

- Control variates are used to reduce the variance of the gradient estimate.
- Idea: subtract a function with known expectation from the noisy estimate.

How it works:

1. Let $\nabla_{\mathbf{\Phi}} \mathbb{E}_{A \sim P_{\mathbf{\Phi}}} [\mathcal{L}(A, \boldsymbol{X})]$ be the gradient to estimate.
2. Find a control variate $c(A, \boldsymbol{X})$ with known expectation $\mathbb{E}_{A \sim P_{\mathbf{\Phi}}} [c(A, \boldsymbol{X})]$.
3. Modify the function:
$\nabla_{\mathbf{\Phi}} \mathbb{E}_{A \sim P_{\mathbf{\Phi}}} [\mathcal{L}(A, \boldsymbol{X})] \approx \nabla_{\mathbf{\Phi}} \mathbb{E}_{A \sim P_{\mathbf{\Phi}}} \left[\mathcal{L}(A, \boldsymbol{X}) - \beta \left(c(A, \boldsymbol{X}) - \mathbb{E}_{A \sim P_{\mathbf{\Phi}}} [c(A, \boldsymbol{X})]\right)\right]$

⚠ The control variate $c(A)$ should be correlated with $\mathcal{L}(A, \boldsymbol{X}) \nabla_{\mathbf{\Phi}} \log P_{\mathbf{\Phi}}(A)$.
⚠ The expectation $\mathbb{E}_{A \sim P_{\mathbf{\Phi}}} [c(A)]$ must be known or easily computable.

# Loss functions

Total loss typically composed of two components:

1. (Un/Self-)Supervised Loss: Drives learning towards meaningful graph structures for solving a specific downstream task.
2. Regularization Loss: Enforces desired properties and constraints on the learned graph.

| **(Self-)Supervised Loss** | **Regularization Loss** |
|---|---|
| Downstream task (MAE, MSE, Cross-Entropy, ...) | Closeness to initial graph structure |
| Denoising loss | Large weights penalization (L1, L2) |
| Contrastive loss | Discourage large / low degree nodes |
| | Enforce symmetry |
| | Enforce or discourage specific graph density |

# Conclusions

# Conclusions

- Learning relational structures offers a powerful alternative to rely on pre-defined or potentially flawed adjacency matrices
- We explored a range of techniques. Each offers different trade-offs in terms of complexity, expressiveness, and gradient estimation properties.

Some bits of advice:

- Don’t underestimate pre-processing! If possible, initialize your scores.
- While challenging, try to visualize small learned graphs. Do the learned connections make sense in your domain?
- GSL papers are noisy! Check if the claims made are sustained in practice with rigorous validations.

# Thank you for your attention!

**Questions?**

# References i


[1] Z. Zhiyao, S. Zhou, B. Mao, *et al.*, **"Opengsl: A comprehensive benchmark for graph structure learning,"** *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[2] B. Fatemi, S. Abu-El-Haija, A. Tsitsulin, *et al.*, **"Ugsl: A unified framework for benchmarking graph structure learning,"** *arXiv preprint arXiv:2308.10737*, 2023.

[3] F. Errica, **"On class distributions induced by nearest neighbor graphs for node classification of tabular data,"** *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[4] 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.

[5] X. Dong, D. Thanou, M. Rabbat, and P. Frossard, **"Learning graphs from data: A signal representation perspective,"** *IEEE Signal Processing Magazine*, vol. 36, no. 3, pp. 44–63, 2019.

[6] V. Kalofolias, **"How to learn a graph from smooth signals,"** in *Artificial intelligence and statistics*, PMLR, 2016, pp. 920–929.

[7] T. K. Rusch, M. M. Bronstein, and S. Mishra, **"A survey on oversmoothing in graph neural networks,"** *arXiv preprint arXiv:2303.10993*, 2023.

# References ii


[8] H. Attali, D. Buscaldi, and N. Pernelle, **"Rewiring techniques to mitigate oversquashing and oversmoothing in gnns: A survey,"** *arXiv preprint arXiv:2411.17429*, 2024.

[9] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, **"Understanding over-squashing and bottlenecks on graphs via curvature,"** *arXiv preprint arXiv:2111.14522*, 2021.

[10] H. Attali, D. Buscaldi, and N. Pernelle, **"Delaunay graph: Addressing over-squashing and over-smoothing using delaunay triangulation,"** in *Forty-first International Conference on Machine Learning*, 2024.

[11] L. Franceschi, M. Niepert, M. Pontil, and X. He, **"Learning discrete structures for graph neural networks,"** in *International conference on machine learning*, PMLR, 2019, pp. 1972–1982.

[12] 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.

[13] A. Cini, D. Zambon, and C. Alippi, **"Sparse graph learning from spatiotemporal time series,"** *Journal of Machine Learning Research*, vol. 24, pp. 1–36, 2023.

[14] A. Manenti, D. Zambon, and C. Alippi, **"Learning latent graph structures and their uncertainty,"** *arXiv preprint arXiv:2405.19933*, 2024.

# References iii


[15] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, **"Dynamic graph cnn for learning on point clouds,"** *ACM Transactions on Graphics (tog)*, vol. 38, no. 5, pp. 1–12, 2019.

[16] 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.

[17] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, **"Monte carlo gradient estimation in machine learning,"** *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5183–5244, 2020.

[18] Y. Bengio, N. Léonard, and A. Courville, **"Estimating or propagating gradients through stochastic neurons for conditional computation,"** *arXiv preprint arXiv:1308.3432*, 2013.

[19] E. Jang, S. Gu, and B. Poole, **"Categorical reparametrization with gumble-softmax,"** in *International Conference on Learning Representations (ICLR 2017)*, OpenReview. net, 2017.

[20] R. J. Williams, **"Simple statistical gradient-following algorithms for connectionist reinforcement learning,"** *Machine learning*, vol. 8, pp. 229–256, 1992.

[21] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, **"Policy gradient methods for reinforcement learning with function approximation,"** *Advances in neural information processing systems*, vol. 12, 1999.