Mathematics of Data: From Theory to Computation

Prof. Volkan Cevher
volkan.cevher@epfl.ch

Supplementary Lecture: Learning on Graphs

Laboratory for Information and Inference Systems (LIONS)
École Polytechnique Fédérale de Lausanne (EPFL)

EE-556 (Fall 2025)

License Information for Mathematics of Data Slides

- This work is released under a Creative Commons License with the following terms:
- **Attribution**
  - The licensor permits others to copy, distribute, display, and perform the work. In return, licensees must give the original authors credit.
- **Non-Commercial**
  - The licensor permits others to copy, distribute, display, and perform the work. In return, licensees may not use the work for commercial purposes – unless they get the licensor's permission.
- **Share Alike**
  - The licensor permits others to distribute derivative works only under a license identical to the one that governs the licensor's work.
- Full Text of the License

Motivation: graphs are everywhere

- Real-world data often exhibits complex relational structure, naturally represented as **graphs**.
- Develop ML methods to capture these relationships.

Example

- **Social Networks:** Nodes represent users, edges represent friendships or connections.
- **Biological Networks:** Protein-protein interactions, drug-target interactions.
- **Molecules:** Represented as graphs with atoms as nodes and chemical bonds as edges.
- **Knowledge Graphs:** Network of relationships between entities (e.g., people, objects, places).

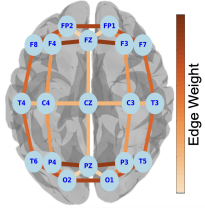

Figure: EEG electrodes as graph. Source: [1].

What are graphs?

Definition: Graph

An attributed **graph** is a tuple $G = (V, E, X)$, where:

- V is a set of n **vertices** (nodes).
- $E = \{(x_i, x_j) \,|\, x_i, x_j \in V\}$ is a set of **edges** (links) between them.
- $X \in \mathbb{R}^{n \times d}$ is a data matrix of **node features**, i.e., additional information associated with each node, where d is the feature dimension.

The **adjacency matrix** $A \in \mathbb{R}^{n \times n}$ represents the connections in the graph:

$$A_{ij} = \begin{cases} w_{ij} & \text{if } (x_i, x_j) \in E, \\ 0 & \text{otherwise,} \end{cases}$$

where w_{ij} is the weight of the edge (x_i, x_j).

Example

- In a social network, node features could be user profiles (age, location, interests).
- In a molecule, node features could be atom types and chemical properties.

Tasks on graphs: overview

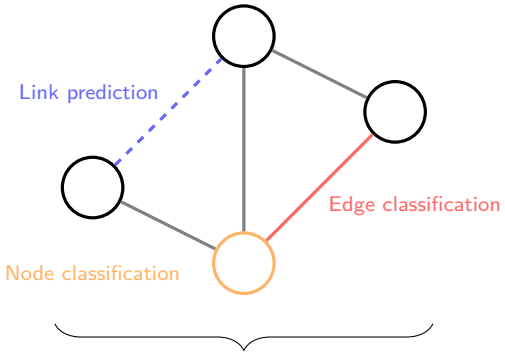

Figure: Overview of ML tasks on graphs.

Tasks on graphs: node-level tasks

- **Node Classification:** Predicting a categorical label for each node.

Example

- **Community Detection:** Identifying groups of densely connected nodes in a social network.
- **Role Discovery:** Determining the function of proteins in a protein-protein interaction network.

- **Node Regression:** Predicting a continuous value for each node.

Example

- **Traffic Forecasting:** Predict the traffic flow (e.g., vehicles per hour) at different points in a city's road network, where each intersection is represented as a node in a graph and road segments as edges.

Tasks on graphs: edge-level tasks

- **Link Prediction:** Predicting the existence of an edge between two nodes.

Example

- **Recommender Systems:** Suggesting items a user might like based on past interactions.
- **Protein-Protein Interaction:** Predict potential interactions between proteins in a biological network.

Tasks on graphs: graph-level tasks

- **Graph Classification:** Assigning a categorical label to an entire graph.

Example

- **Molecule Properties:** Predicting the toxicity or solubility of a molecule based on its structure.
- **Scene Understanding:** Classifying an image based on the objects and their relationships.

- **Graph Generation:** Creating new graphs with desired properties.

Example

- **Drug Design:** Generating novel molecules with specific pharmacological properties.

Challenges of learning on graphs & graph representation learning

- **Arbitrary size and topology:** Graphs have varying numbers of nodes and edges
  - difficult to apply traditional deep learning architectures designed for fixed-sized data like images [2]

Permutation invariance

A function $h : \mathbb{R}^{|V|\times|V|} \times \mathbb{R}^{|V|\times m} \to \mathbb{R}^d$ mapping an attributed graph G to a d-dimensional embedding is permutation invariant if: $h(A, X) = h(PAP^T, PX)$ for any permutation matrix $P \in \mathbb{R}^{|V|\times|V|}$.

Graph representation learning

Core idea: Learn a mapping $f : V \to \mathbb{R}^d$ that embeds nodes $v \in V$ into a low-dimensional vector space $\mathbb{R}^d$, such that the embeddings capture the graph's structure and node features.

Remarks:

- Unlike sequences, there is often no natural order for nodes in a graph.
- Graph models must be permutation invariant (i.e., the same output regardless of node ordering).
- Graph representation learning allows us to apply traditional machine learning algorithms,
  - classification, regression, clustering, ... to graph data.
- These learned embeddings are called **node embeddings**.

Random walk embeddings: DeepWalk

- **Core Idea:** Learn node representations by capturing how nodes co-occur in random walks [9].
- **Steps:**
  1. Generate random walks starting from each node in the graph.
  2. Treat the walks as sentences and nodes as words.
  3. Use the Skip-gram model [8] to learn node embeddings.
- **Intuition:** Nodes that frequently co-occur in random walks should have similar embeddings.

DeepWalk: skip-gram model

Skip-gram objective [8]

Maximize the probability of observing context nodes given a center node in the random walk:

$$\max_{\phi} \frac{1}{n} \sum_{u \in V} \sum_{v \in N_R(u)} \log \mathrm{p}(v|u;\phi),$$

where ϕ is the embedding map and $N_R(u)$ is the multiset of nodes visited in random walks starting from node u. $\mathrm{p}(v|u;\phi)$ is usually efficiently parametrized with softmax as $\mathrm{p}(u \mid v;\phi) = \frac{\exp(\phi(u)^\top \phi(v))}{\sum_{w \in V} \exp(\phi(w)^\top \phi(v))}$.

Example

Consider a random walk: [A, B, C, D, E].

- With a context window size of 2, for the center node C, the context nodes are B, D.
- The skip-gram model aims to maximize the probabilities $\mathrm{p}(B|C;\phi)$ and $\mathrm{p}(D|C;\phi)$.

node2vec: exploring local and global structure

- **node2vec** [4] is an unsupervised learning method to generate vector embeddings of nodes in a graph.
- It combines properties of both breadth-first (BFS) and depth-first (DFS) graph search strategies.
- **Goal**: Learn to map nodes to a continuous vector space where nodes with similar neighborhoods are close.
- Random walk controlled by two parameters:
  - **p (return parameter):** Controls the likelihood of returning to the previous node.
  - **q (in-out parameter):** Controls the likelihood of exploring inward vs. outward nodes.

node2vec: biased random walk

- node2vec generates biased random walks to explore the graph structure.

Transition probability

Let c_i be the i-th node in the random walk, the transition probability from node v to x is

$$\mathsf{p}(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v,x) \in E \\ 0 & \text{otherwise} \end{cases}$$

where π_{vx} is the unnormalized transition probability and Z is the normalization constant.

Biased random walk

Given a random walk that just traversed edge (t, v) and is currently at node v, the unnormalized transition probability to node x is:

$$\pi_{vx} = \begin{cases} \frac{1}{p}, & \text{if } d_{tx} = 0 \text{ (return to previous node)} \\ 1, & \text{if } d_{tx} = 1 \text{ (stay at the same distance)} \\ \frac{1}{q}, & \text{if } d_{tx} = 2 \text{ (move further away)} \end{cases}$$

where d_{tx} is the shortest path distance between nodes t and x.

node2vec: optimization objective

- Skip-gram model is used to optimize the node embeddings.
- Goal: Maximize the log-probability of observing context nodes.

Objective function

Given a source node u, the node2vec objective maximizes the log-probability of observing its network neighborhood $N_S(u)$ with sampling strategy S:

$$\max_f \sum_{u \in V} \log \mathsf{p}(N_S(u)|\phi(u)).$$

Remarks:

- Assume conditional independence: $\mathsf{p}(N_S(u)|\phi(u)) = \prod_{n_i \in N_S(u)} \mathsf{p}(n_i|\phi(u))$.
- Model probability using softmax: $\mathsf{p}(n_i|\phi(u)) = \frac{\exp(\phi(n_i)^\top \phi(u))}{\sum_{v \in V} \exp(\phi(v)^\top \phi(u))}$.

Beyond shallow embeddings: the need for GNNs

- Recap limitations of shallow methods:
  - **Transductivity:** Cannot generalize to unseen nodes/graphs
  - **Structural similarity:** Struggles to capture higher-order patterns
  - **Node features:** Unable to leverage rich attribute information

Example

In a molecule classification task, the chemical properties are often determined by substructures involving multiple atoms. Shallow embeddings may fail to represent these complex substructures effectively.

Remarks:

- Shallow encoders are limited by the expressiveness of the adjacency matrix A as a similarity measure $\rightarrow$ can't easily capture complex, multi-hop relationships.
- Motivatess Graph Neural Networks (GNNs) as a solution.

Overview of a GNN layer

Aggregating information

Core concept: Aggregate information from neighbors to update node representations [2].

- Two-step process:
  1. **Message computation:** Transforms neighbor information into messages.
  2. **Message aggregation:** Combines messages from neighbors.

Non-linearity (activation function) is applied for expressiveness to obtain embedding $h_v^{(l)}$.

Example

In a citation network, a paper's embedding could be updated based on both its own content and the aggregated information from papers that cite it.

Overview of a GNN layer (cont.)

Message passing formulation

$$m_v^{(l)} = \mathsf{AGG}^{(l)}\left(\{m_u^{(l)} | u \in \mathcal{N}(v)\}\right). \tag{1}$$

- $m_u^{(l)}$: Message from neighbor u to node v at layer l.
- $\mathsf{AGG}^{(l)}(\cdot)$: Aggregation function at layer l (e.g., sum, mean, max).
- $\mathcal{N}(v)$: Set of neighbors of node v.

Remarks:

- The node's own information at the previous layer is often also included in the aggregation by $h_v^{(l)} = \mathsf{COMBINE}^{(l)}\left(h_v^{(l-1)}, m_v^{(l)}\right)$.

Classical GNN layers

Graph Convolutional Network (GCN) [7]

- Aggregates neighbor information using MEAN.
- Computationally efficient due to its straightforward formulation.
- Can be expressed in matrix form, enabling efficient implementation.

$$h_v^{(l)} = \sigma\left(W^{(l)} \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{h_u^{(l-1)}}{|\mathcal{N}(v)| + 1}\right),$$

where:

- $h_v^{(l)}$: Node embedding of node v at layer l.
- $\mathcal{N}(v)$: Set of neighbors of node v.
- $W^{(l)}$: Learnable weight matrix at layer l.
- $\sigma(\cdot)$: Activation function (e.g., ReLU).

Classical GNN layers (cont.)

GraphSAGE [5]

- Employs a two-stage aggregation process:
  1. Aggregate information from the node's neighbors.
  2. Concatenate the result from step 1 with the node's previous embedding.

Message passing in GraphSAGE:

$$h_{\mathcal{N}(v)}^{(l)} = \text{AGGREGATE}_k\left(\{h_u^{(l-1)} : u \in \mathcal{N}(v)\}\right)$$

$$h_v^{(l)} = \sigma\left(W^{(l)} \cdot \text{CONCAT}(h_v^{(l-1)}, h_{\mathcal{N}(v)}^{(l)})\right)$$

Remarks:

- Offers flexibility through different aggregation options:
  - Mean aggregation (similar to GCN).
  - Max-pooling aggregation: Selects the maximum value across feature dimensions.
  - LSTM aggregation: Applies LSTMs to a random permutation of the node's neighbors.

Stacking GNN layers

- Building a multi-layer GNN allows to capture increasingly complex relationships within the graph.
- Each layer expands the receptive field, incorporating information from nodes further away.

Multi-Layer GNN structure

- **Input:** Raw node features $X \in \mathbb{R}^{n \times d}$
- **Hidden Layers:** Apply L GNN layers sequentially, where each layer k produces embeddings $H^{(k)} \in \mathbb{R}^{n \times d'}$.
- **Output:** Final node embeddings $Z = H^{(L)} \in \mathbb{R}^{n \times d'}$.

Example

Let $\hat{A} = D^{-1/2} A D^{-1/2}$ be the normalized adjacency matrix. A 2-layer GCN can be expressed as:

$$H^{(1)} = \sigma(\hat{A} X W^{(0)})$$
$$Z = H^{(2)} = \sigma(\hat{A} H^{(1)} W^{(1)})$$

Remarks:
- The receptive field of a node at layer k is the set of nodes influencinc its embedding at that layer.
- For a standard GNN, the receptive field corresponds to the k-hop neighborhood.

The over-smoothing problem

- As we increase the number of GNN layers, the receptive fields of different nodes start to overlap.
- This leads to nodes aggregating similar information, causing their embeddings to converge.
- This phenomenon is known as *over-smoothing*.

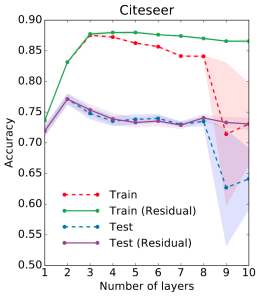

Figure: Illustration of over-smoothing in a GCN. Source: [7].

Remarks:

- Key Takeaway: Carefully choose the number of GNN layers! Too few layers may limit expressiveness, but too many layers can lead to over-smoothing. Usually $L = 2$.

Hierarchical pooling

Motivation

Graph-level tasks (e.g., predicting molecule properties or classifying social network structures) often require capturing hierarchical information present in the graph. Traditional GNNs with max/average pooling struggle to represent this hierarchy effectively.

DiffPool: differentiable graph pooling [13]

DiffPool learns a differentiable clustering of nodes at each layer, enabling the GNN to operate on progressively coarsened graph representations.

Core idea:

- Standard GNN modules extract node embeddings at each layer.
- A separate GNN learns a soft cluster assignment matrix $S^{(k)} \in \mathbb{R}^{n_k \times n_{k+1}}$ for each layer k.

DiffPool algorithm [13]

Learning cluster assignments

Two GNN modules are used.

- Embedding GNN: $Z^{(k)} = \text{GNN}_k^{\text{embed}}(A^{(k)}, X^{(k)})$ - This GNN focuses on extracting informative node embeddings.
- Pooling GNN (followed by row-wise softmax): $S^{(k)} = \text{SOFTMAX}(\text{GNN}_k^{\text{pool}}(A^{(k)}, X^{(k)}))$ - This GNN learns which nodes to group together in clusters.

Graph coarsening

The soft assignments are used to pool nodes into clusters.

- Pooled Embeddings: $X^{(k+1)} = {S^{(k)}}^T Z^{(k)}$. The original embeddings are combined based on the cluster assignments.
- Coarsened Adjacency Matrix: $A^{(k+1)} = {S^{(k)}}^T A^{(k)} S^{(k)}$. This reflects the connections between the newly formed clusters.

Remarks:

- DiffPool produces soft cluster assignments that allow differential pooling operation.
- This learnable coarsening allows for better generalization on graph-level tasks, capturing both local (node-level) and global (graph-level) features.

Representational power of GNNs: GIN [12]

Graph Isomorphism Network (GIN)

The GIN layer update rule is defined as:

$$h_v^{(k)} = \text{MLP}^{(k)} \left((1 + \epsilon^{(k)}) h_v^{(k-1)} + \sum_{u \in N(v)} h_u^{(k-1)} \right), \tag{2}$$

where:

- $h_v^{(k)}$ is the embedding of node v at layer k.
- $\text{MLP}^{(k)}$ is a multi-layer perceptron at layer k.
- $\epsilon^{(k)}$ is a learnable scalar at layer k.

The graph-level readout for GIN is:

$$h_G = \text{CONCAT} \left(\sum_{v \in V} h_v^{(k)} \Big| k = 0, ..., K \right). \tag{3}$$

Representational power of GNNs: the Weisfeiler-Lehman test

Weisfeiler-Lehman (WL) test [6]

The WL test is an iterative algorithm for graph isomorphism testing. It assigns colors to nodes based on their neighborhood structure:

$$c^{(k+1)}(v) = \text{HASH}\left(c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)}\right) \tag{4}$$

where:

- $c^{(k)}(v)$ is the color of node v at iteration k,
- HASH is an injective function that maps different inputs to distinct colors.

Remarks:

- If two graphs have different color histograms after convergence, they are guaranteed to be non-isomorphic.
- The other direction does not hold: there are non-isomorphic graphs where the difference is not detected (e.g., some regular graphs).

Representational power of GNNs: GIN's expressiveness

Theorem (Xu et al., ICLR 2019, [12])

GIN is as powerful as the WL test in distinguishing graph structures. GIN with sufficient layers can map any two graphs that the WL test deems non-isomorphic to different embeddings.

Remarks:

- GIN has the highest possible representational power among message-passing GNNs.
- GIN achieves this by using an injective aggregation function (summation) and MLPs, which are universal approximators.

E(n) equivariant GNNs [10]

Permutation invariance vs equivariance

A function $f(A, X)$ is permutation invariant if reordering the nodes in the graph does not change the output:

$$f(A', X') = f(A, X) \tag{5}$$

where $A' = PAP^T$ and $X' = PX$ for any permutation matrix P. A function $F(A, X)$ is permutation equivariant if reordering the nodes in the graph leads to a corresponding reordering of the output. Formally:

$$F(A', X') = PF(A, X) \tag{6}$$

where $A' = PAP^T$ and $X' = PX$ for any permutation matrix P.

E(n) equivariant GNNs [10]

E(n) GNNs are designed to be equivariant to rotations and translations in Euclidean space. For a graph with coordinates C and features H, an E(n) GNN $\mathsf{EGNN}(H, C, A)$ satisfies:

$$H, CQ + s = \mathsf{EGNN}(H, CQ + s, A) \tag{7}$$

for any rotation matrix Q and shift s.

E(n) equivariant GNNs (cont.)

Transformation in EGNNs

EGNNs updates both node features and spatial coordinates:

$$h_i^{l+1}, c_i^{l+1} = \text{EGNN}(h_i^l, c_i^l, e_{ij})$$

where h_i^l is the node feature of node i at layer l, c_i^l is the coordinate embedding, e_{ij} is the edge feature between nodes i and j.

Message passing in EGNNs

Message between nodes i and j: $m_{ij} = \phi_e(h_i^l, h_j^l, \|c_i^l - c_j^l\|, e_{ij})$

Coordinate update: $c_i^{l+1} = c_i^l + \sum_{j \neq i}(c_i^l - c_j^l)\phi_c(m_{ij})$

Message aggregation: $m_i = \sum_{j \neq i} m_{ij}$

Node feature update: $h_i^{l+1} = \phi_h(h_i^l, m_i)$

Example

E(n) equivariant GNNs are particularly useful for 3D molecular data, where the spatial arrangement of atoms is critical for determining molecular properties.

Generalization of message passing

Let us consider a more general form of a message-passing neural network [3]:

Convolutional

- Traditional graph convolution.
- Aggregate neighbor features with weights determined by edge attributes c_{ij}.

$$h_i = \phi\left(x_i, \oplus_{j \in \mathcal{N}_i} c_{ij} \psi(x_j)\right)$$

Attentional

- Use attention mechanism to compute edge-specific weights α_{ij}.
- Attention score depends on node features.

$$h_i = \phi\left(x_i, \oplus_{j \in \mathcal{N}_i} \alpha(x_i, x_j) \psi(x_j)\right)$$

Message passing

- General form of message passing.
- Use learned message functions m_{ij} to aggregate information.

$$h_i = \phi\left(x_i, \oplus_{j \in \mathcal{N}_i} \psi(x_i, x_j)\right)$$

Graph attention networks (GAT) [11]

Adapting self-attention to graphs

- GATs [11] apply self-attention to graph-structured data.
- Node i's representation is updated by attending to its neighbors:

$$h_i^{(l)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(l)} W^{(l)} h_j^{(l-1)}\right).$$

- $\alpha_{ij}^{(l)}$ is the attention weight from node j to node i in layer l.

Attention mechanism

- Computed similarly to Transformers, but using node features:

$$\alpha_{ij}^{(l)} = \text{softmax}_j\left(\text{LeakyReLU}\left(a^T[W^{(l)} h_i^{(l-1)} || W^{(l)} h_j^{(l-1)}]\right)\right).$$

References I

[1] Arshia Afzal, Grigorios Chrysos, Volkan Cevher, and Mahsa Shoaran.
REST: Efficient and accelerated EEG seizure analysis through residual state updates.
In *International Conference on Machine Learning*, volume 235, pages 271–290, 2024.
(Cited on page 3.)

[2] Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda.
A gentle introduction to deep learning for graphs.
Neural Networks, 129:203–221, 2020.
(Cited on pages 9 and 16.)

[3] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Velickovic.
Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021.
(Cited on page 29.)

[4] Aditya Grover and Jure Leskovec.
node2vec: Scalable feature learning for networks.
In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.
(Cited on page 12.)

References II

[5] Will Hamilton, Zhitao Ying, and Jure Leskovec.
Inductive representation learning on large graphs.
In *Advances in neural information processing systems*, volume 30, 2017.
(Cited on page 19.)

[6] Ningyuan Teresa Huang and Soledad Villar.
A short tutorial on the weisfeiler-lehman test and its variants.
In *ICASSP*, pages 8533–8537. IEEE, 2021.
(Cited on page 25.)

[7] Thomas N Kipf and Max Welling.
Semi-supervised classification with graph convolutional networks.
In *International Conference on Learning Representations*, 2022.
(Cited on pages 18 and 21.)

[8] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean.
Efficient estimation of word representations in vector space.
In *International Conference on Learning Representations*, 2013.
(Cited on pages 10 and 11.)

References III

[9] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena.
Deepwalk: Online learning of social representations.
In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.
(Cited on page 10.)

[10] Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling.
E(n) equivariant graph neural networks.
In *International Conference on Machine Learning*, pages 9323–9332. PMLR, 2021.
(Cited on page 27.)

[11] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio.
Graph attention networks.
In *International Conference on Learning Representations*, 2018.
(Cited on page 30.)

[12] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka.
How powerful are graph neural networks?
In *International Conference on Learning Representations*, 2019.
(Cited on pages 24 and 26.)

References IV

[13] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec.
Hierarchical graph representation learning with differentiable pooling.
In *Advances in neural information processing systems*, volume 31, 2018.
(Cited on pages 22 and 23.)