UNIVERSITY OF COPENHAGEN

Graph Fundamentals and Graph Neural Networks

NORA Summer School on Geometric Deep Learning

Raghavendra Selvan

Assistant Professor (TT)
Dept. of Computer Science (ML Section)
University of Copenhagen
Pioneer Centre for Artificial Intelligence, Denmark

raghav@di.ku.dk
𝕏 /raghavian
https://raghavian.github.io

June 11, 2024

Overview of Lectures

1. Lecture-1
   - Motivation
     - Why Graph ML?
     - Deep Learning Basics
   - Graph Learning in Spectral domain
     - ChebyNet
   - Graph Learning in Spatial domain
     - Node GNNs
   - Applications: Node/Graph Classification

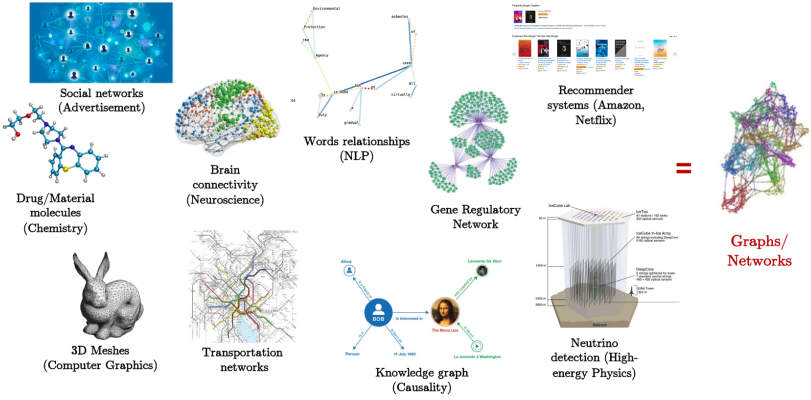
1. Lecture-2
   - Learning on Graphs
   - GraphSAGE
   - Jumping Knowledge Networks
   - Graph Isomorphism Network
   - Graph Attention Network
   - Relational GNNs

1. Lecture-3
   - Graph Representation Learning
   - Self-Supervised GNNs
   - Generative Modelling for Graphs
     - Variational Autoencoders
     - Diffusion Models
   - Resource Efficiency and Graph ML
     - Energy Consumption Aware NAS
     - Extreme Quantization of Activation Maps
   - Summary

Graph Fundamentals and GNNs: Overview

1. Lecture-1
   - Motivation
     - Why Graph ML?
     - Deep Learning Basics
   - Graph Learning in Spectral domain
     - ChebyNet
   - Graph Learning in Spatial domain
     - Node GNNs
   - Applications: Node/Graph Classification

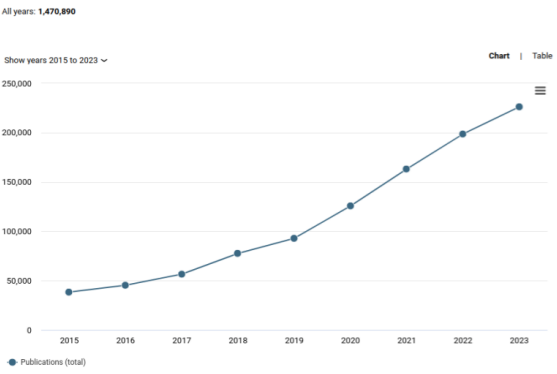
Graphs are everywhere!



Heterogenous graph-structured data are non-Euclidean data

https://graphdeeplearning.github.io/project/spatial-convnets/

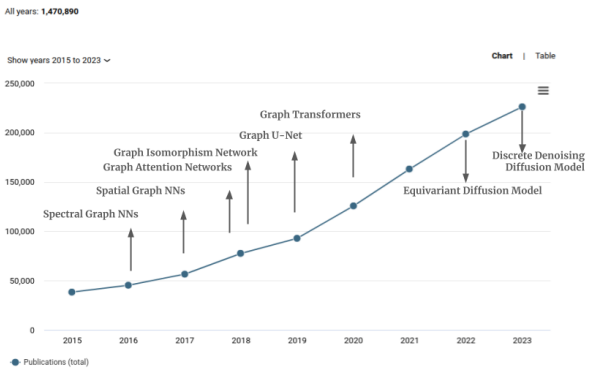
Graph/Geometric Deep Learning is also going everywhere!



Adapted from `https://app.dimensions.ai/`

Defferrard et al. 2016. Convolutional neural networks on graphs with fast localized spectral filtering.
Kipf & Welling. 2017. Semi-supervised classification with graph convolutional networks.
Veličković et al. 2018 "Graph attention networks."; Xu, Keyulu, et al. 2018 "How Powerful are Graph Neural Networks?
Gao & Ji. 2019. Graph U-Nets; Yun et al. 2019. Graph Transformer Networks
Hoogeboom et al. Equivariant Diffusion for Molecule Generation in 3D. 2022; Vignac et al. DiGress: Discrete Denoising diffusion for graph generation. 2023

Graph/Geometric Deep Learning is also going everywhere!



Adapted from `https://app.dimensions.ai/`

Defferrard et al. 2016. Convolutional neural networks on graphs with fast localized spectral filtering.
Kipf & Welling. 2017. Semi-supervised classification with graph convolutional networks.
Veličković et al. 2018 "Graph attention networks."; Xu, Keyulu, et al. 2018 "How Powerful are Graph Neural Networks?
Gao & Ji. 2019. Graph U-Nets; Yun et al. 2019. Graph Transformer Networks
Hoogeboom et al. Equivariant Diffusion for Molecule Generation in 3D. 2022; Vignac et al. DiGress: Discrete Denoising diffusion for graph generation. 2023

Let us start with a quiz

- Multiple choice questions
- Choose the most correct answer as you see it
- Some specialised terms/acronyms are intentionally used
- Log on to socrative.com
- Choose “Student Login”
- Room name: RAGHAV

One slide introduction to Deep Learning

Observed data	$\mathbf{G} = \{\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_N\}$
Labels / Targets	$\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}$
Decision functions/ Models	$g_\theta(\cdot) : \mathbf{G} \rightarrow \mathbf{Y}$

- $g_\theta(\cdot)$ can be any DL model
- θ are the trainable parameters; $|\theta| >> N$
- Over-parameterised, non-linear function approximator
- Supervised training uses large labelled datasets
- Gradient descent based optimisation
- Multi-layered perceptrons, Convolutional/Graph neural networks, Transformers...

Convolutional neural networks (CNNs)

- Learnable convolution kernels
- Technically, not convolution but cross-correlation
- Kernel flipping can be overcome during learning
- Known to learn general image descriptors
- And also, specialized task-specific kernels

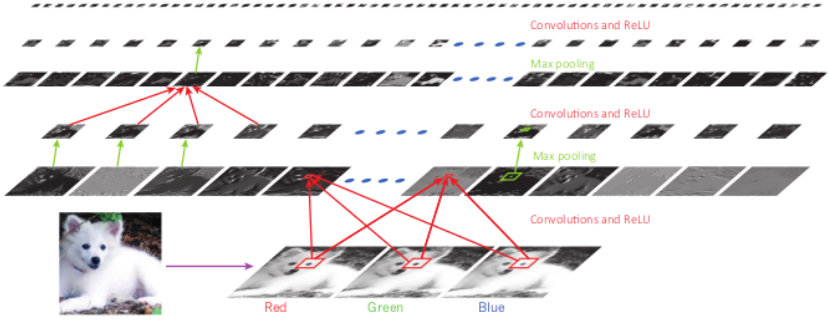


Figure reproduced from Le Cun et al. Deep Learning. 2015

Key properties exploited by CNNs when operating on images, or generally data in Euclidean domain

Success of CNNs in computer vision is due to several properties.

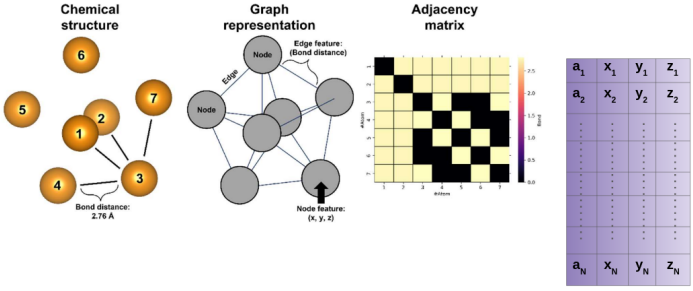
Consider a compact d-dimensional Euclidean domain $\Omega = [0,1]^d \in \mathbb{R}^d$, on which square integrable functions $f \in L^2(\Omega)$ are defined. Further, consider a translation operation $\mathcal{T}$

- **Translation invariance**: $y = f(\mathcal{T}(x)) = f(x)$,
- **Translation equivariance**: $y = f(\mathcal{T}(x)) = \mathcal{T}(f(x))$
- **Scale separation**: Long range dependencies from multi-scale interaction terms
- **Compositionality**: $\mathbf{y} = f_1 \circ f_2 \circ \cdots f_L(\mathbf{x})$
  where $f_i(\cdot)$ are comprised of convolution kernels, non-linearities and sometimes pooling operations.

What about permutation invariance: $f(\Pi(\mathbf{x})) = f(\mathbf{x})$ with permutation operator, $\Pi(\cdot)$

$\int_{-\infty}^{+\infty} |f(x)|^2 dx < \infty$

Graphs as data structures

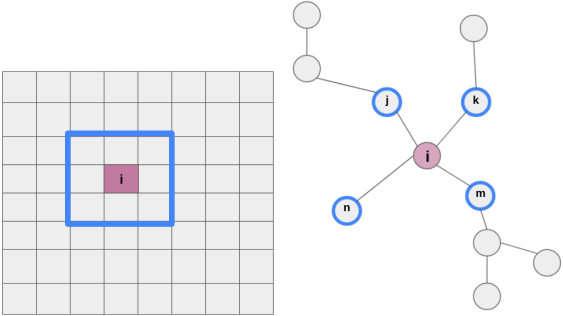


Graph Notations

Consider a graph $\mathcal{G} = (\mathbf{X}, \mathbf{A}, \mathbf{E})$ with N nodes comprising F features per node: $\mathbf{X} \in \mathbb{R}^{N \times F}$ and adjacency matrix $\mathbf{A} \in \{0, 1\}^{N \times N}$ and edge attributes $\mathbf{E} \in \mathbb{R}^{E \times N \times N}$.

Neighbourhoods in Euclidean and non-Euclidean domains

Regular vs Irregular neighbourhoods



Graph Isomorphism

Graph Isomorphism

Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, where V_1 and V_2 are the sets of vertices, E_1 and E_2 sets of edges, and A_1 and A_2 are adjacency matrices.

Graph Isomorphism

Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, where V_1 and V_2 are the sets of vertices, E_1 and E_2 sets of edges, and A_1 and A_2 are adjacency matrices.

- Two graphs G_1 and G_2 are **isomorphic** if there is a bijection between their vertex sets that preserves adjacency.

Graph Isomorphism

Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, where V_1 and V_2 are the sets of vertices, E_1 and E_2 sets of edges, and A_1 and A_2 are adjacency matrices.

- Two graphs G_1 and G_2 are **isomorphic** if there is a bijection between their vertex sets that preserves adjacency.
- Essentially, G_1 and G_2 are structurally identical, even if their vertex labels are different.

Graph Isomorphism

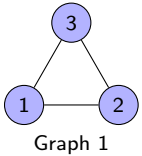
Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, where V_1 and V_2 are the sets of vertices, E_1 and E_2 sets of edges, and A_1 and A_2 are adjacency matrices.

- Two graphs G_1 and G_2 are **isomorphic** if there is a bijection between their vertex sets that preserves adjacency.
- Essentially, G_1 and G_2 are structurally identical, even if their vertex labels are different.
- Problem is to determine whether there exists a bijection $\phi : V_1 \to V_2$ such that $(u, v) \in E_1$ if and only if $(\phi(u), \phi(v)) \in E_2$ for all $u, v \in V_1$.

Graph Isomorphism

Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, where V_1 and V_2 are the sets of vertices, E_1 and E_2 sets of edges, and A_1 and A_2 are adjacency matrices.

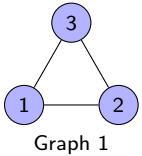
- Two graphs G_1 and G_2 are **isomorphic** if there is a bijection between their vertex sets that preserves adjacency.
- Essentially, G_1 and G_2 are structurally identical, even if their vertex labels are different.
- Problem is to determine whether there exists a bijection $\phi : V_1 \to V_2$ such that $(u, v) \in E_1$ if and only if $(\phi(u), \phi(v)) \in E_2$ for all $u, v \in V_1$.



Graph Isomorphism

Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, where V_1 and V_2 are the sets of vertices, E_1 and E_2 sets of edges, and A_1 and A_2 are adjacency matrices.

- Two graphs G_1 and G_2 are **isomorphic** if there is a bijection between their vertex sets that preserves adjacency.
- Essentially, G_1 and G_2 are structurally identical, even if their vertex labels are different.
- Problem is to determine whether there exists a bijection $\phi : V_1 \to V_2$ such that $(u, v) \in E_1$ if and only if $(\phi(u), \phi(v)) \in E_2$ for all $u, v \in V_1$.



Formal Definition of Graph Isomorphism

Given two graphs G_1 and G_2, determine whether there exists a permutation Π of V_1 such that $A_1^{\Pi} = A_2$, where A_1^{Π} denotes the adjacency matrix of G_1 under permutation Π.

Many Open Problems in Graph Isomorphism

Complexity

Many Open Problems in Graph Isomorphism

Complexity

- Complexity of the graph isomorphism problem is an area of active research.
- It is not known to be in P or NP-complete, it is widely believed to be in the complexity class NP-intermediate ($P \neq NP$)
- The best known algorithms for graph isomorphism have a complexity of $O(2^{O(\sqrt{n}\log n)})$, where $n = \max(n_1, n_2)$.

Applications

Many Open Problems in Graph Isomorphism

Complexity

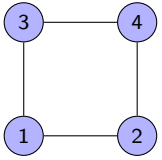
- Complexity of the graph isomorphism problem is an area of active research.
- It is not known to be in P or NP-complete, it is widely believed to be in the complexity class NP-intermediate ($P \neq NP$)
- The best known algorithms for graph isomorphism have a complexity of $O(2^{O(\sqrt{n}\log n)})$, where $n = \max(n_1, n_2)$.

Applications

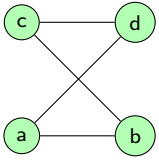
- Chemistry: Identifying molecular structures and chemical compounds.
- Computer Vision: Matching objects and shapes in images.
- Cryptography: Generating cryptographic keys using isomorphic graphs.

Graph Isomorphism: Example with 4 Vertices

Graph Isomorphism: Example with 4 Vertices

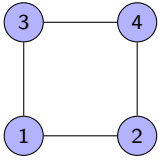


Graph 3

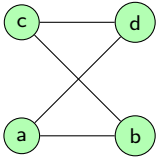


Graph 4

Graph Isomorphism: Example with 4 Vertices

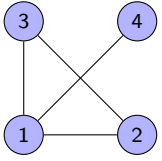


Graph 3

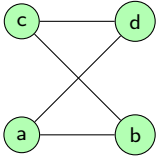


Graph 4

Graph 3 and Graph 4 are isomorphic.



Graph 5



Graph 6

Graph 5 and Graph 6 are not isomorphic.

Complexity of Graph Isomorphism Tests

Complexity of Graph Isomorphism Tests

- **Naive Approach**
  - Tries all possible permutations of vertices.
  - Complexity: $O(n!)$, where n is the number of vertices.
  - Impractical for large graphs due to factorial growth.
- **State-of-the-Art Algorithms**

Complexity of Graph Isomorphism Tests

- **Naive Approach**
  - Tries all possible permutations of vertices.
  - Complexity: $O(n!)$, where n is the number of vertices.
  - Impractical for large graphs due to factorial growth.
- **State-of-the-Art Algorithms**
  - Best known exact algorithms: $O(2^{O(\sqrt{n}\log n)})$.
  - Exponential but significantly better than the naive approach.
- **Heuristic Algorithms**

Complexity of Graph Isomorphism Tests

- **Naive Approach**
  - Tries all possible permutations of vertices.
  - Complexity: $O(n!)$, where n is the number of vertices.
  - Impractical for large graphs due to factorial growth.
- **State-of-the-Art Algorithms**
  - Best known exact algorithms: $O(2^{O(\sqrt{n}\log n)})$.
  - Exponential but significantly better than the naive approach.
- **Heuristic Algorithms**
  - Various heuristic methods can speed up the process.
  - Exploit graph properties and use efficient data structures.
- **Average and Worst-Case Complexity**

Complexity of Graph Isomorphism Tests

- **Naive Approach**
  - Tries all possible permutations of vertices.
  - Complexity: $O(n!)$, where n is the number of vertices.
  - Impractical for large graphs due to factorial growth.
- **State-of-the-Art Algorithms**
  - Best known exact algorithms: $O(2^{O(\sqrt{n}\log n)})$.
  - Exponential but significantly better than the naive approach.
- **Heuristic Algorithms**
  - Various heuristic methods can speed up the process.
  - Exploit graph properties and use efficient data structures.
- **Average and Worst-Case Complexity**
  - Average-case complexity varies with graph distribution.
  - Worst-case complexity is the most critical measure.
- **Implementation Factors**

Complexity of Graph Isomorphism Tests

- **Naive Approach**
  - Tries all possible permutations of vertices.
  - Complexity: $O(n!)$, where n is the number of vertices.
  - Impractical for large graphs due to factorial growth.
- **State-of-the-Art Algorithms**
  - Best known exact algorithms: $O(2^{O(\sqrt{n}\log n)})$.
  - Exponential but significantly better than the naive approach.
- **Heuristic Algorithms**
  - Various heuristic methods can speed up the process.
  - Exploit graph properties and use efficient data structures.
- **Average and Worst-Case Complexity**
  - Average-case complexity varies with graph distribution.
  - Worst-case complexity is the most critical measure.
- **Implementation Factors**
  - Size of input graphs.
  - Implementation details and efficiency of data structures.

Weisfeiler-Leman Test

B Weisfeiler and A Leman, "The reduction of a graph to canonical form and the algebra which appears therein," 1968

Weisfeiler-Leman Test

- Weisfeiler-Leman (WL) test is an algorithm for testing graph isomorphism.

B Weisfeiler and A Leman, "The reduction of a graph to canonical form and the algebra which appears therein," 1968

Weisfeiler-Leman Test

- Weisfeiler-Leman (WL) test is an algorithm for testing graph isomorphism.
- It operates by refining vertex labels iteratively.
- The key idea is to maintain a multiset of vertex labels.

B Weisfeiler and A Leman, "The reduction of a graph to canonical form and the algebra which appears therein," 1968

Weisfeiler-Leman Test

- Weisfeiler-Leman (WL) test is an algorithm for testing graph isomorphism.
- It operates by refining vertex labels iteratively.
- The key idea is to maintain a multiset of vertex labels.
- Efficient for graphs with bounded vertex degrees, i.e., $deg(v_i) \leq D \quad \forall\ v_i \in V$, where $D \in \mathbb{I}^+$ is the upper bound.

B Weisfeiler and A Leman, "The reduction of a graph to canonical form and the algebra which appears therein," 1968

Weisfeiler-Leman Test

- Weisfeiler-Leman (WL) test is an algorithm for testing graph isomorphism.
- It operates by refining vertex labels iteratively.
- The key idea is to maintain a multiset of vertex labels.
- Efficient for graphs with bounded vertex degrees, i.e., $deg(v_i) \leq D \quad \forall\ v_i \in V$, where $D \in \mathbb{I}^+$ is the upper bound.
- Complexity: The time complexity of the WL test is $O(|V| \log |V|)$ in the worst case.
- This complexity arises from the need to sort the labels in each iteration.

B Weisfeiler and A Leman, "The reduction of a graph to canonical form and the algebra which appears therein," 1968

Weisfeiler-Leman Test

- Weisfeiler-Leman (WL) test is an algorithm for testing graph isomorphism.
- It operates by refining vertex labels iteratively.
- The key idea is to maintain a multiset of vertex labels.
- Efficient for graphs with bounded vertex degrees, i.e., $deg(v_i) \leq D \quad \forall\ v_i \in V$, where $D \in \mathbb{I}^+$ is the upper bound.
- Complexity: The time complexity of the WL test is $O(|V| \log |V|)$ in the worst case.
- This complexity arises from the need to sort the labels in each iteration.

Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

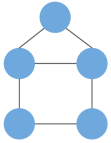
Input: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, ..., h_N^{(0)})$
Output: Final node coloring $(h_1^{(T)}, h_2^{(T)}, ..., h_N^{(T)})$
$t \leftarrow 0$;
repeat
  for $v_i \in \mathcal{V}$ **do**
    $h_i^{(t+1)} \leftarrow \text{hash}\left(\sum_{j \in \mathcal{N}_i} h_j^{(t)}\right)$;
  $t \leftarrow t + 1$;
until *stable node coloring is reached*;

Original version formulated it as a iterative node coloring problem.

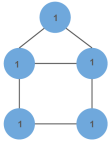
B Weisfeiler and A Leman, "The reduction of a graph to canonical form and the algebra which appears therein," 1968

Graph Isomorphism: Example with 5 Vertices

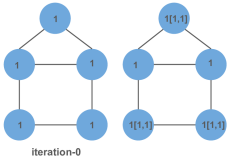
Graph Isomorphism: Example with 5 Vertices



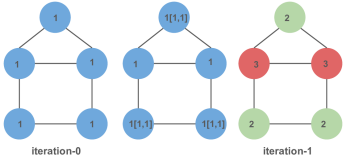
Graph Isomorphism: Example with 5 Vertices



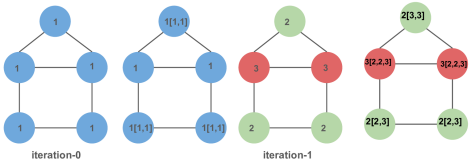
Graph Isomorphism: Example with 5 Vertices



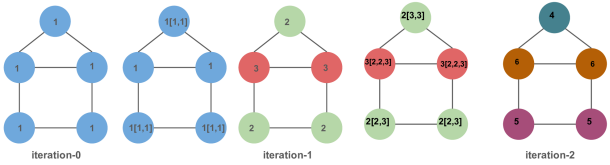
Graph Isomorphism: Example with 5 Vertices



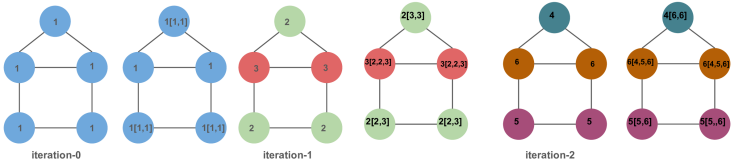
Graph Isomorphism: Example with 5 Vertices



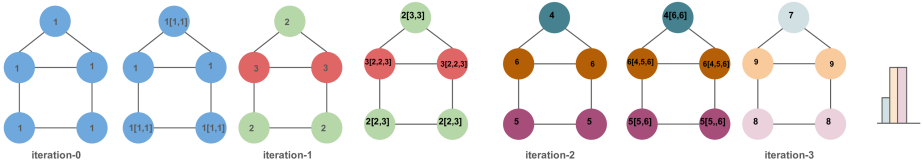
Graph Isomorphism: Example with 5 Vertices



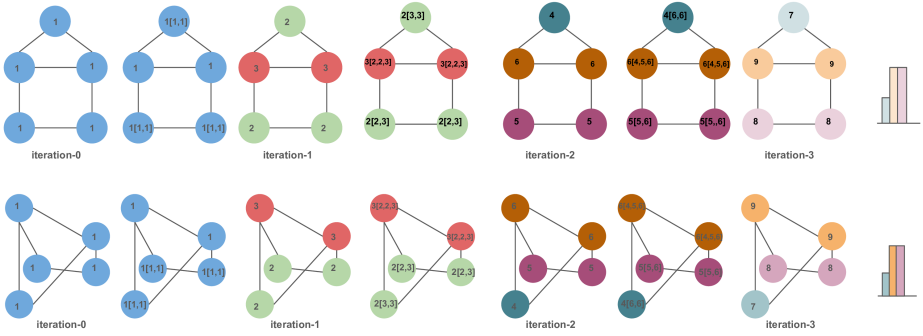
Graph Isomorphism: Example with 5 Vertices



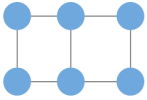
Graph Isomorphism: Example with 5 Vertices



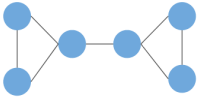
Graph Isomorphism: Example with 5 Vertices



Let's try out the 1-WL test!



Graph A



Graph B

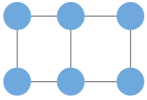
Shervashidze et al. Weisfeiler-Lehman Graph Kernels. 2011

Let's try out the 1-WL test!

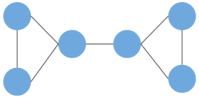
Work out the canonical graphs for the two graphs

Shervashidze et al. Weisfeiler-Lehman Graph Kernels. 2011

Let's try out the 1-WL test!

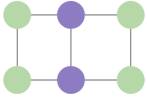


Graph A

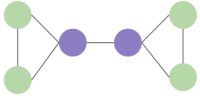


Graph B

Work out the canonical graphs for the two graphs



Graph A



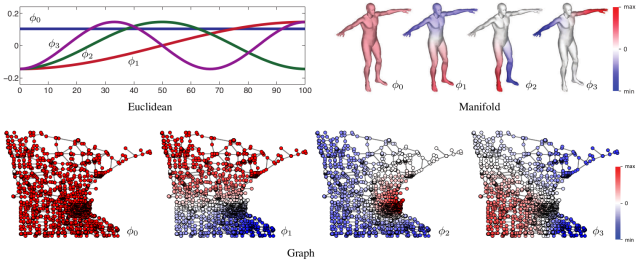
Graph B

Shervashidze et al. Weisfeiler-Lehman Graph Kernels. 2011

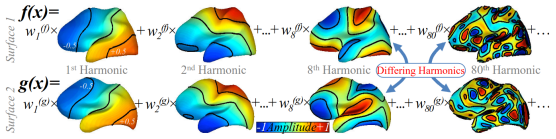
Limitations

- **Non-Guarantee**: The WL test does not guarantee correctness for all instances.
- **Rare Failures**: Instances where non-isomorphic graphs have identical label multisets are rare in practice.
- **Dependency on Labeling**: The performance of the WL test depends on the initial vertex labeling.
- **Limited Application**: While versatile, the WL test may not be suitable for certain specialized cases or highly irregular graphs.

Fourier decomposition of signals



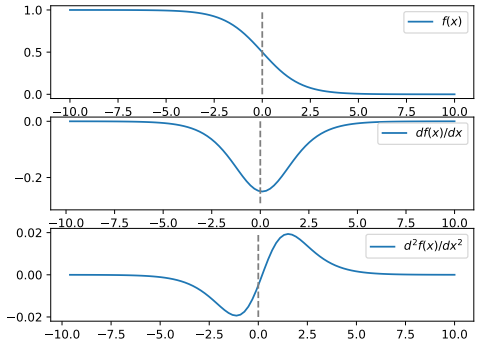
Shape Spectrum – Surface Functions can be written as composition of shape harmonics



Bronstein et al. 2016. Geometric deep learning: Going beyond Euclidean data.
Lombaert 2015. Brain Transfer: Spectral analysis of cortical surfaces and functional maps

Laplacian on different domains means (almost) the same

Laplacian: Second order differential operator
(Tells you how smoothly the function is changing in its domain)



When specified for graphs (discrete grids) → Graph Laplacian

Graph Laplacian

For a graph, $\mathcal{G}$ with adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$: the graph Laplacian is simply given as:

$$\mathbf{L} = \mathbf{D} - \mathbf{A} \tag{1}$$

$\mathbf{D} \in \mathbb{R}^{N \times N}$ is the degree matrix with $D_{ii} = \sum_j A_{ij}$

where, $\mathbf{\Phi} = (\phi_0, \phi_1 \dots \phi_{N-1}) \in \mathbb{R}^{N \times N}$ are the orthogonal eigenfunctions forming the Fourier basis, with corresponding eigenvalues in the diagonal matrix $\mathbf{\Lambda} = \text{diag}(\lambda_0, \lambda_1, \dots, \lambda_{N-1})$

Graph Neural Networks (GNNs): Overview

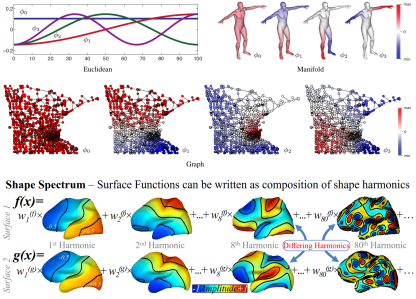
1. Lecture-1
   - Motivation
     - Why Graph ML?
     - Deep Learning Basics
   - Graph Learning in Spectral domain
     - ChebyNet
   - Graph Learning in Spatial domain
     - Node GNNs
   - Applications: Node/Graph Classification

How to Perform Convolution on Graphs?

Consider a signal operating on the nodes, $\mathbf{x} \in \mathbb{R}^N$ and a filter g_Θ parameterized by $\Theta \in \mathbb{R}^N$, the graph convolution is given as:

$$g_\Theta \star \mathbf{x} = \tag{2}$$

where, $\mathbf{\Phi} = (\phi_0, \phi_1 \dots \phi_{N-1}) \in \mathbb{R}^{N \times N}$ are the orthogonal eigenfunctions forming the Fourier basis



Bronstein et al. 2016. Geometric deep learning: Going beyond Euclidean data.
Lombaert 2015. Brain Transfer: Spectral analysis of cortical surfaces and functional maps

Convolution on graphs in Fourier domain

Consider a signal operating on the nodes, $\mathbf{x} \in \mathbb{R}^N$ and a filter g_Θ parameterized by $\Theta \in \mathbb{R}^N$, the graph convolution is given as:

$$g_\Theta \star \mathbf{x} = (\mathbf{\Phi} g_\Theta)(\mathbf{\Phi}^T \mathbf{x}) \tag{3}$$

Recollect that: $\mathbf{L} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^T$, hence $g_\Theta = \mathbf{\Lambda}$ and $\mathbf{\Phi}^T\mathbf{x}$ is the Fourier transform of $\mathbf{x}$

Turns out that $g_\Theta = \mathbf{\Lambda}$ is not very useful

- Non-localized, non-parametric (all free parameters)
- Computationally expensive

So, use a polynomial parameterization for localized filters with:

$$g_\Theta(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k \mathbf{\Lambda}^k \tag{4}$$

Approximating Convolution on Graphs with Chebyshev polynomials

Recursive formulation for fast filtering:

$$g_\Theta(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k \mathbf{\Lambda}^k \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{\Lambda}}) \tag{5}$$

with $\tilde{\mathbf{\Lambda}} = \frac{2}{\lambda_{\max}}\mathbf{\Lambda} - \mathbf{I}_N \in [-1, 1]$ and Chebyshev coefficients $\Theta \in \mathbb{R}^K$.

ChebyNet

Incorporating the recursive approximation yields the popular spectral graph convolution method: **ChebyNet**

$$g_\Theta \star \mathbf{x} = (\mathbf{\Phi} g_\Theta)(\mathbf{\Phi}^T \mathbf{x}) \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}})\mathbf{x} \tag{6}$$

with $\tilde{\mathbf{L}} = \frac{2}{\lambda_{\max}}\mathbf{L} - \mathbf{I}_N$.

Chebyshev polynomial $T_k(y)$ of order k is given by the recurrence: $T_k(y) = 2yT_{k-1}(y) - T_{k-2}(y)$ with $T_0 = 1$, $T_1 = y$

Defferrard et al. (2016). Convolutional neural networks on graphs with fast localized spectral filtering.

Overview of ChebyNet: CNN for Graphs (but in spectral domain)

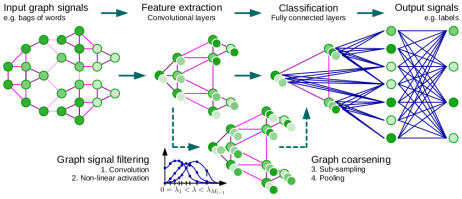
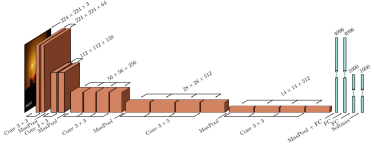


Figure 1: Architecture of a CNN on graphs and the four ingredients of a (graph) convolutional layer.

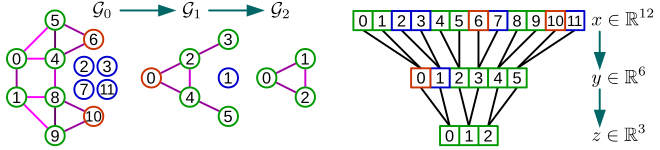
Spectral CNNs mimic standard CNNs



Defferrard et al. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. 2016

How to Perform Pooling on Graphs?

Graph Clustering is NP-hard



Based on a greedy algorithm (Graculus), accelerated by using balanced binary trees using fake nodes.

Defferrard et al. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. 2016
I. Dhillon, Y. Guan, and B. Kulis. Weighted Graph Cuts Without Eigenvectors: A Multilevel Approach. 2007.
Liu et al. Graph Pooling for Graph Neural Networks: Progress, Challenges, and Opportunities. 2023

Graph Neural Networks (GNNs): Overview

Further approximation of ChebyNet yields the class of Node GNNs

Recall that, from ChebyNet, $g_\Theta \star \mathbf{x} = (\mathbf{\Phi} g_\Theta)(\mathbf{\Phi}^T \mathbf{x}) \approx \left(\mathbf{\Phi} \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{\Lambda}})\right)(\mathbf{\Phi}^T \mathbf{x}) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}})\mathbf{x}$,
With $K = 2, \lambda_{max} = 2$, the graph convolution operation becomes:

$$g_\Theta \star \mathbf{x} \approx \theta_0 \mathbf{x} + \theta_1 \tilde{\mathbf{L}} \mathbf{x} \tag{7}$$

$$= \theta_0 \mathbf{x} + \theta_1 (\mathbf{L} - \mathbf{I}_N)\mathbf{x} \tag{8}$$

$$= \theta (\mathbf{I}_N - \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2})\mathbf{x} \tag{9}$$

$$g_\Theta \star \mathbf{x} \approx \theta (\tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2} \mathbf{x}) \tag{10}$$

with $\theta_0 = -\theta_1$, and $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$. More generally, for input $\mathbf{X} \in \mathbb{R}^{N \times F}$ and weight matrix $\mathbf{\Theta} \in \mathbb{R}^{F \times L}$:

$$\mathbf{H} = \sigma(\hat{\mathbf{A}} \mathbf{X} \mathbf{\Theta}) \tag{11}$$

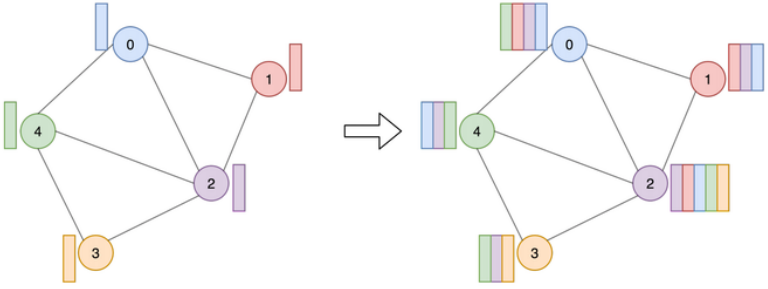
Stacking multiple of these layers with non-linearities yields the class of node GNNs[3]!

[3] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." (2016)

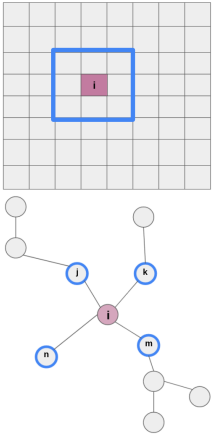
Note: Chebyshev polynomial $T_k(y)$ of order k is given by the recurrence: $T_k(y) = 2yT_{k-1}(y) - T_{k-2}(y)$ with $T_0 = 1$, $T_1 = y$

GNNs update states of nodes based on neighbourhood

Graphically, $\mathbf{H} = \sigma(\hat{\mathbf{A}}\mathbf{X}\mathbf{\Theta})$ means:



GNNs when seen from a node's point of view



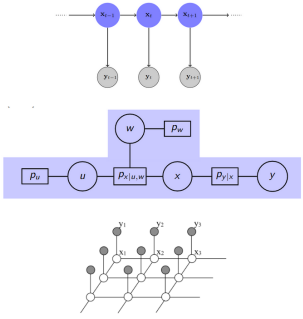
- For a node $i \in \mathcal{V}$ with neighbours $\mathcal{N}_i$ the GNN operation in layer-ℓ is given as:

$$\mathbf{h}_i^{(\ell)} = \sigma\left(\frac{\Theta^{(\ell)}}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i}\mathbf{h}_j^{(\ell-1)}\right) \tag{12}$$

  with $\mathbf{h}_i^{(0)} = \mathbf{x}_i$ and $\Theta^{(\ell)}$ are trainable parameters.
- Aggregation of transformed neighbouring node features
- M-layered GNN provides information from M-hops away!
- A form of learnable message passing!

Rich classes of message passing algorithms exist for inference on PGMs

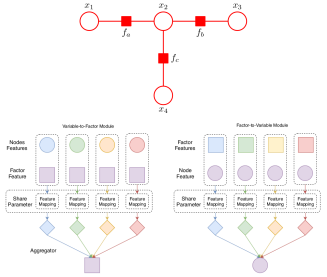
- Kalman filter for Markov chains
- Sum-product algorithm for factor graphs
- Loopy belief propagation for graphs with cycles
- Mean field approximation for Markov random fields



Can GNNs be interpreted as generalizations of these algorithms?

Message Passing Neural Networks

- Spectral and Spatial to notion of generalized message passing
- Close connections to probabilistic graphical models (PGMs)
- Similarities with classic methods in PGMs like max-product algorithm/ belief propagation



Gilmer et al. Neural Message Passing for Quantum Chemistry. 2017.
Bishop. Pattern Recognition and Machine Learning. 2006
Zhang et al. Factor Graph Neural Network. 2020.

Graph Neural Networks (GNNs): Overview

Node Classification using GCNs

Observed data	$\mathbf{G} : \mathbf{X} \in \mathbb{R}^{N \times F}, \mathbf{A} \in \mathbb{R}^{N \times N}$
Node-level Labels	$\mathbf{Y} : \{0, 1\}^N$
Node Classifier	$g_\theta(\cdot) : \mathbf{G} \rightarrow \mathbf{Y}$

L-layered GCN for node classification

$$\mathbf{H}^{(\ell)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(\ell-1)}\mathbf{\Theta}^{(\ell)}\right) \tag{13}$$

where, $\mathbf{H}^{(0)} = \mathbf{X}$ and $\mathbf{H}^{(L)} = \mathbf{Y}$
Network weights Θ are trained in a supervised manner by optimizing an appropriate loss function: $\mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}})$.

Graph Classification using GCNs

Observed data: $\mathbf{G} = \{\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_N\} : \mathbf{G}_i : \mathbf{X}_i, \mathbf{A}_i, \mathbf{E}_i$

Labels $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\} : \mathbf{y}_i \in [0, C-1]^N$

Graph Classifier $g_\theta(\cdot) : \mathbf{G} \to \mathbf{Y}$

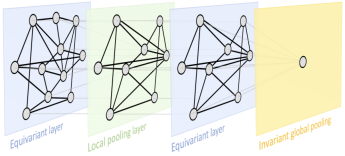


Figure 8: Geometric Deep Learning blueprint, exemplified on a graph. A typical Graph Neural Network architecture may contain permutation equivariant layers (computing node-wise features), local pooling (graph coarsening), and a permutation-invariant global pooling layer (readout layer).

L-layered GCN for node classification

$$\mathbf{H}^{(\ell)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(\ell-1)}\mathbf{\Theta}^{(\ell)}\right) \tag{14}$$

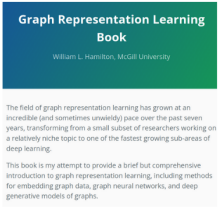
$$\hat{\mathbf{Y}} = \text{Aggregate}(\mathbf{H}^{L}) \tag{15}$$

where, $\mathbf{H}^{(0)} = \mathbf{X}$

Trained in a fully supervised manner by optimizing an appropriate loss function: $\mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}})$.

Bronstein et al. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. 2021

Resources



https://www.cs.mcgill.ca/~wlh/grl_book/
https://geometricdeeplearning.com/
https://ogb.stanford.edu/
https://pytorch-geometric.readthedocs.io/en/latest/

Summary

- Convolutions on graphs can be approximated in spectral domain
- ChebyNet uses a polynomial of spectral filter approximated with Chebyshev polynomials
- First order approximation to ChebyNet yields spatial graph convolutions
- Message passing networks can be used learn *useful* embeddings for downstream tasks
- Simple GCNs for node classification; can be extended to graph classification models
- Community started out with convolution on graphs (GCNs)
- Currently, primarily viewed as learnable message passing (GNNs)

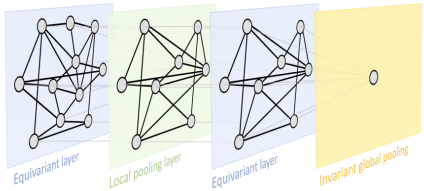


Figure 8: Geometric Deep Learning blueprint, exemplified on a graph. A typical Graph Neural Network architecture may contain permutation equivariant layers (computing node-wise features), local pooling (graph coarsening), and a permutation-invariant global pooling layer (readout layer).