NetSciX School Lecture

Deep Graph Learning

A Gentle Introduction for Network Scientists

Prof. Dr. Ingo Scholtes
CAIDAS-Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science
Julius-Maximilians-Universität Würzburg, Germany
ingo.scholtes@uni-wuerzburg.de

Network models of complex systems

complex networks

- **graph** or **network** consists of a collection of **nodes**, where some pairs of nodes are connected by **links**
- universal **mathematical abstraction** for complex systems that consist of **many interacting elements**

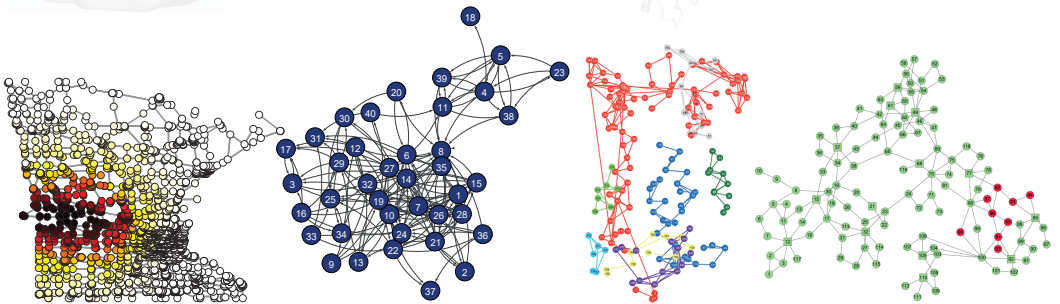
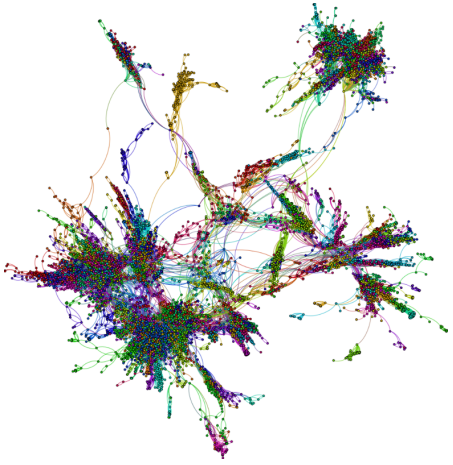


image credit: www.geni.org, pixabay, adapted from → Woods et al., 2017 and → MM Bronstein et al., 2017

From network science to graph learning



- how can we **apply machine learning to complex networks**?
- how can **network scientists help to advance deep learning** for graph-structured data?

Outline of today's lecture

Intro to Machine Learning for Complex Networks → 15 mins

- Supervised vs. Unsupervised Learning
- Machine Learning for Euclidean Data
- Euclidean Machine Learning on Graphs

Deep Learning Fundamentals → 15 mins

- Perceptron Learning
- Feed-Forward Neural Networks
- Gradient Descent and Backpropagation

Deep Graph Learning and Graph Neural Networks → 30 mins

- Neural Message Passing
- Graph Convolutional Networks
- Semi-Supervised Learning

Research Challenges in Deep Graph Learning → 15 mins

- Expressivity, Noisy Data, Heterophilic Networks
- Over-Smoothing/Over-Squashing, Temporal Data and Causality

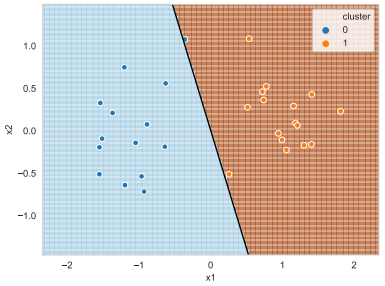
accompanying hands-on tutorial with 12 jupyter notebooks available at

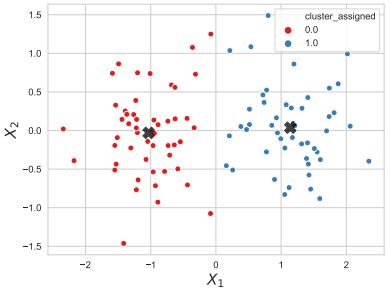
`https://github.com/pathpy/2026-netscix-tutorial/`

Supervised vs. unsupervised machine learning

learn model in **labeled examples**

detect patterns in **unlabeled data**





example: classification with support vector machine (SVM)

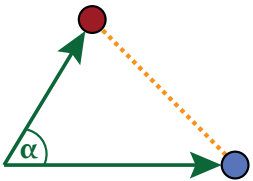
find d — 1-dim. hyperplane that separates classes such that margin of decision boundary is maximized

example: k-means cluster detection

assign data points to k clusters, such that squared distance of points to closest cluster center is minimized

Machine Learning in Euclidean data

- traditional machine learning techniques assume **Euclidean feature spaces**, e.g. $x_i \in \mathbb{R}^d$
- d-dimensional Euclidean space is **metric space** with **Euclidean distance** metric
- Euclidean vector space = d-dimensional **inner product space** over $\mathbb{R}$



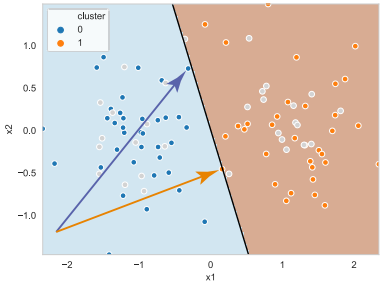
$$\|\vec{y} - \vec{x}\| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

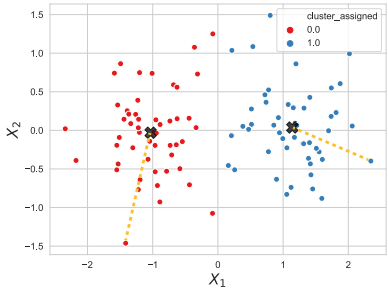
$$\vec{x} \cdot \vec{y} = x_1 y_1 + x_1 x_2 = \|\vec{x}\| \|\vec{y}\| \cos \alpha$$

Euclid of Alexandria as depicted in the fresco "The School of Athens"

born ca. 325 BC

Machine Learning in Euclidean data





example method: suppport vector machine (SVM)

find $d - 1$-dim. hyperplane that separates classes such that margin of decision boundary is maximized → BSc/MSc Lecture: Data Mining

example method: k-means clustering

assign data points to k clusters, such that squared dist. of points to closest cluster center is minimal → Bsc/MSc Lecture: Data Mining

dot product between $x_i \in \mathbb{R}^2$, i.e. we use property of **inner product space**

distance between $x_i \in \mathbb{R}^2$, i.e. we use property of **metric space**

Machine Learning for Complex Networks?

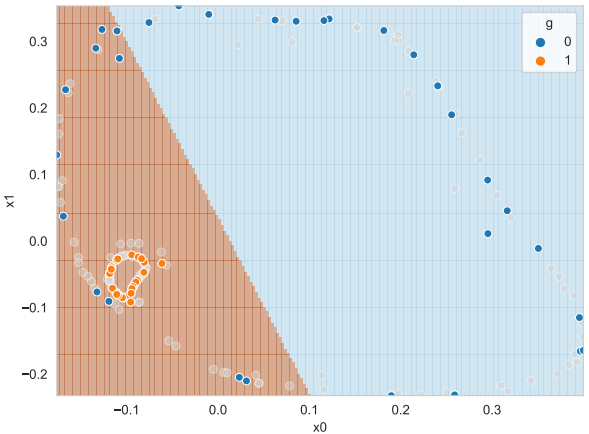
- how can we apply machine learning to **non-Euclidean data** with graph structure?

traditional two-step approach

1. map graph to **Euclidean space**
   → graph embedding, representation learning, graph kernels
2. apply **Euclidean machine learning techniques**, e.g. logistic regression, SVM, neural networks, …

example: binary node classification

1. use **Laplacian Eigenmaps** to generate Euclidean representation of nodes in a graph
2. apply **logistic regression** to classify nodes based on their Euclidean representation



tutorial notebooks

see notebooks 01 – 03 in repository at
→ https://github.com/pathpy/2026-netscix-tutorial

From Logistic Regression to Perceptron

- **perceptron** is a classifier inspired by neuron → F Rosenblatt, 1958
- **linear combination** $f : \mathbb{R}^k \to \mathbb{R}$ of inputs with bias $\beta_0 \in \mathbb{R}$ and weights $\beta_1, \ldots, \beta_k \in \mathbb{R}$, i.e.

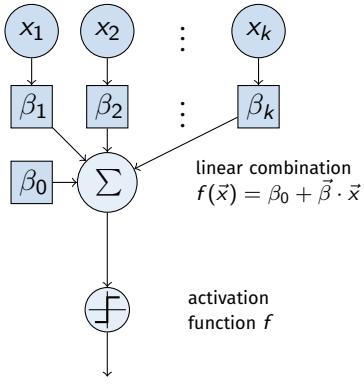
$$f(\vec{x}) := \beta_0 + \sum_{i=1}^{k} \beta_i \cdot x_i = \vec{\beta} \cdot (1, \vec{x})^T$$

with $\vec{\beta} \in \mathbb{R}^{k+1}$ and $(1, \vec{x}) := (1, x_1, \ldots, x_k)$

- **non-linear activation function** yields binary classifier, e.g.

$$\sigma(f(\vec{x})) = \frac{1}{1 + e^{-f(\vec{x})}} \in [0, 1]$$

where σ is **logistic function** (which make perceptron identical to logistic regression!)



Gradient-based Learning of Parameters

- for given parameters $\vec{\beta} \in \mathbb{R}^{k+1}$ and training examples $(\vec{x}_s, \hat{y}_s)$ we define **L2 loss function** as

$$L(\vec{\beta}) := \frac{1}{2} \sum_{s=1}^{n} (\sigma(f(\vec{x}_s)) - \hat{y}_s)^2$$

- idea: move along **gradients of L** to find parameters $\hat{\beta}$ that **minimize loss function**
- calculate how training example $(\vec{x}_s, \hat{y}_s)$ contributes to **partial derivatives of loss function**

$$\frac{\partial L_s}{\partial \beta_j} = (y_s - \hat{y}_s) \cdot y_s \cdot (1 - y_s) \cdot x_{sj} \qquad (\text{for } j = 1, \ldots, k)$$

$$\frac{\partial L_s}{\partial \beta_0} = (y_s - \hat{y}_s) \cdot y_s \cdot (1 - y_s)$$

perceptron learning algorithm (L2 loss, logistic activation function)

1. choose initial parameters $\beta_i = 0$ and learning rate $\eta \in [0, 1]$
2. for each $(\vec{x}_s, y_s)$ in **training batch** do:
   - $\beta_0 = \beta_0 - \eta(y_s - \hat{y}_s) \cdot y_s \cdot (1 - y_s)$
   - $\beta_j = \beta_j - \eta(y_s - \hat{y}_s) \cdot y_s \cdot (1 - y_s) \cdot x_{sj}$ for $j = 1, \ldots, k$
3. repeat 2 until $L(\vec{\beta}) \leq \epsilon$ → M Minsky and S Papert, 1969

graph learning terminology

- **training batch** is a subset of training examples used to calculate the gradient in one iteration of the learning algorithm
- **epoch** is a full pass through the training data, i.e. one iteration of the learning algorithm for each training example

Feed-Forward Neural Networks

- like logistic regression, perceptron classifier has **linear decision boundary**
- idea: couple **multiple layers of perceptrons**
- neuron h_j in **hidden layer** with k inputs with width d

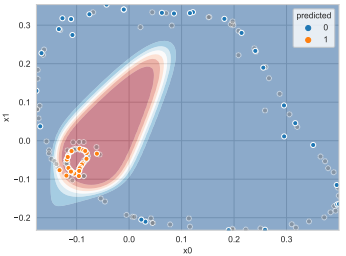
$$h_j := h_j(\vec{x}) = \sigma\left(\beta^1_{0,j} + \sum_{i=1}^{k} \beta^1_{i,j} x_i\right) = \sigma(\vec{\beta}^1_j \cdot (1, \vec{x})^T)$$

- neuron y_i in **output layer** with d inputs

$$y_i := y_i(\vec{x}) = \sigma\left(\beta^2_{0,i} + \sum_{j=1}^{d} \beta^2_{j,i} h_j(\vec{x})\right) = \sigma(\vec{\beta}^2_i \cdot (1, \vec{h})^T)$$

Universal Approximation Theorem

"arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with only a single [...] hidden layer and any continuous sigmoidal nonlinearity" → G Cybenko, 1989



non-linear decision boundary of feed-forward network with one **hidden layer** with width $d = 2$

Gradient Optimization in Neural Networks

- for training samples $(\vec{x}_s, \hat{y}_s)$ and output $y_s := y_1(\vec{x}_s)$ consider **L2 loss function**

$$L(\beta^1, \beta^2) = \frac{1}{2}\sum_{s=1}^{n}(\hat{y}_s - y_s)^2 \quad \text{and} \quad L_s(\beta^1, \beta^2) = \frac{1}{2}(\hat{y}_s - y_s)^2$$

  where β^j is **weight matrix** of neurons in layer j and L_s is contribution of $(\vec{x}_s, \hat{y}_s)$

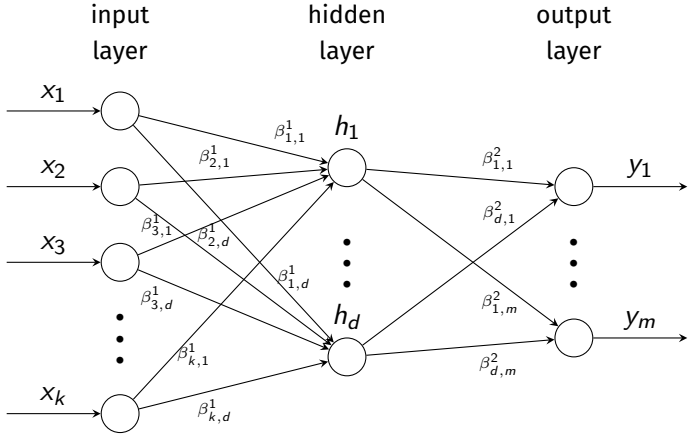
- output y_s of feed-forward network with two layers and activation function σ is given by **composition of functions**, i.e. $y_s = \sigma(\beta^2 \cdot \sigma(\beta^1 \cdot (1, \vec{x}_s)^T))$
- for **output neuron** y_i application of chain rule yields partial derivates w.r.t. $\beta^2_{j,i}$

$$\frac{\partial L_s}{\partial \beta^2_{j,i}} = (\hat{y}_s - y_s)\sigma'(\underbrace{\vec{\beta}^2_i \cdot (1, \vec{h})^T}_{\text{input to } y_i})h_j(\vec{x}_s)$$

- for **hidden neuron** h_j we apply chain rule once more and obtain partial derivatives w.r.t. $\beta^1_{k,j}$

$$\frac{\partial L_s}{\partial \beta^1_{k,j}} = (\hat{y}_s - y_s)\sigma'(\underbrace{\vec{\beta}^2_i \cdot (1, \vec{h})^T}_{\text{input to } y_i})\beta^2_{i,j}\sigma'(\underbrace{\vec{\beta}^1_j \cdot (1, \vec{x}_s)^T}_{\text{input to } h_j}) \cdot x_{sk}$$

Neural Networks as Computation Graphs



- (deep) neural networks = **computation graph** (where neurons between layers are fully connected)
- to calculate gradients of loss function w.r.t weights, we **recursively apply chain rule**, starting at outputs y_i until we reach the inputs x_i

Differentiation via Backpropagation

- to calculate parameter gradients we **propagate model loss backwards** from output to input layer

→ DE Rumelhart et al., 1986

stochastic gradient descent optimization algorithm for feed-forward neural network

1. choose initial parameters β_{ij}^{l} and learning rate $\eta \in [0, 1]$
2. for i in range(iterations) do:
3. batch = random subset of training examples
4. for each $(\vec{x}_s, \hat{y}_s)$ in **training batch** do:
5. update weights of output neurons y_i

$$\beta_{j,i}^{2} = \beta_{j,i}^{2} - \eta(\hat{y}_s - y_s)\sigma'(\vec{\beta}_i^{2} \cdot (1, \vec{h})^T)h_j(\vec{x}_s)$$

6. update weights of hidden neurons h_j

$$\beta_{k,j}^{1} = \beta_{k,j}^{1} - \eta(\hat{y}_s - y_s)\sigma' \vec{\beta}_i^{2} \cdot (1, \vec{h})^T \beta_{i,j}^{2} \sigma'(\vec{\beta}_j^{1} \cdot (1, \vec{x}_s)^T) \cdot x_{sk}$$

tutorial notebooks

see notebooks 04 – 06 in repository at

→ https://github.com/pathpy/2026-netscix-tutorial

see how we can use `pytorch`'s `autograd` feature to automatically calculate gradients of loss functions in feed-forward neural networks

End-to-end deep learning for complex networks?

two-step approach to ML for complex networks

1. map graph to **Euclidean space**
   → embedding, representation learning, graph kernels
2. apply **Euclidean machine learning techniques**
   → e.g. feed-forward neural networks

problem

- Euclidean **representation independent of learning task** and machine learning model
- we want to learn representations tailored to specific learning task (e.g. node classification)

- how can we apply **end-to-end deep learning** to graph-structured data?

	supervised	unsupervised
node level	node classification	community detection node embedding
edge level	link prediction link classification	link prediction graph reconstruction
graph level	graph classification graph regression	graph clustering graph embedding

taxonomy of popular graph learning tasks

Neural Message Passing and GNNs

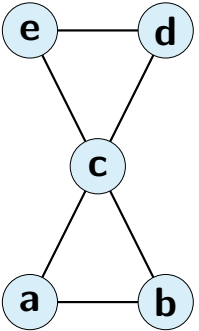
- idea: use graph topology to **update node features** based on **message passing algorithm** → J Gilmer et al. 2017
- network science view: discrete-time dynamical system where $h_i^{(t)} \in \mathbb{R}^d$ denotes **state of node i at time t**
- nodes update their state $h_i^{(t)}$ based on **states of their neighbors**, i.e.

$$h_i^{(t)} = F_{j \in N(i)} h_j^{(t-1)}$$

where F is **aggregation function** and $N(i)$ is set of neighbors of i

- for **add aggregation** we get update rule

$$h_i^{(t)} = \sum_{j \in N(i)} h_j^{(t-1)}$$



add aggregation rule

node	$t=0$	$t=1$	$t=2$
a	1	5	16
b	2	4	17
c	3	12	24
d	4	8	19
e	5	7	20

Neural Message Passing and GNNs

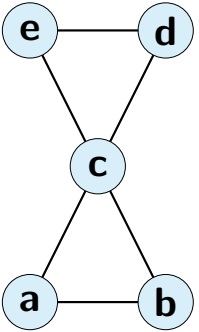
- for networks without self-loops, nodes do not consider their **own prior state**
- to avoid this, we explicitly **add self-loops**

$$h_i^{(t)} = \sum_{j \in N(i) \cup \{i\}} h_j^{(t-1)}$$

- we can additionally transform updated node state with **differentiable function** g (e.g. a perceptron), i.e.

$$h_i^{(t)} = g\left(\sum_{j \in N(i) \cup \{i\}} h_j^{(t-1)} \right)$$

- message passing is **permutation equivariant**, i.e. node permutation $\rightarrow$ consistent permutation of outputs $h_i^{(t)}$



additional transformation with $g(x) = 0.5 + 2 \cdot x$

node	$t = 0$	$t = 1$	$t = 2$
a	1	12.5	111.5
b	2	12.5	111.5
c	3	30.5	209.5
d	4	24.5	159.5
e	5	24.5	159.5

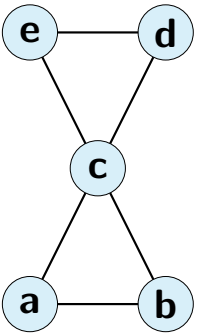
Degree-based Normalization

- heterogeneity of networks requires application of **degree-based normalization**
- we can use **mean rather than add aggregation**, i.e.

$$h_i^{(t)} = g\left(\sum_{j \in N(i)} \frac{h_j^{(t-1)}}{d_i}\right)$$

- we can apply **symmetric degree-based normalization**, i.e.

$$h_i^{(t)} = g\left(\sum_{j \in N(i)} \frac{h_j^{(t-1)}}{\sqrt{d_i d_j}}\right)$$



symmetric normalization (and self-loops)

node	$t=0$	$t=1$	$t=2$
a	1	1.8	2.1
b	2	1.8	2.1
c	3	2.7	3.6
d	4	3.8	3.5
e	5	3.8	3.5

Spectral Graph Convolution

- for graph with adjacency matrix $\mathbf{A}$ and diagonal degree matrix $\mathbf{D}$ consider **Laplacian matrix**

$$\mathcal{L} := \mathbf{D} - \mathbf{A}$$

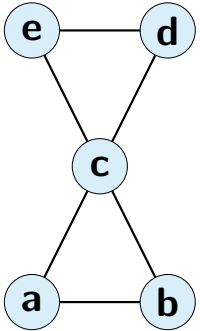
- symmetric degree-based normalization yields **symmetric normalized Laplacian** → F Chung, 1997

$$\mathcal{L}^* = \mathbf{D}^{\frac{1}{2}}\mathcal{L}\mathbf{D}^{\frac{1}{2}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$$

with entries

$$\mathcal{L}^*_{ij} = \begin{cases} -\frac{1}{\sqrt{d_i \cdot d_j}} \text{ if } i \neq j \text{ and } A_{ij} = 1 \\ 1 \text{ if } i = j \\ 0 \text{ else} \end{cases}$$

- message passing can be viewed as **efficient localized version of spectral graph convolution**



Symmetric Normalized Laplacian

$$\mathcal{L}^* = \begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2\sqrt{2}} & 0 & 0 \\ -\frac{1}{2} & 1 & -\frac{1}{2\sqrt{2}} & 0 & 0 \\ -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & 1 & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} \\ 0 & 0 & -\frac{1}{2\sqrt{2}} & 1 & -\frac{1}{2} \\ 0 & 0 & -\frac{1}{2\sqrt{2}} & -\frac{1}{2} & 1 \end{pmatrix}$$

Connection to heat diffusion and CNNs

- consider **Laplacian operator** describing heat diffusion in **continuous Euclidean space**

$$\nabla f := \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- $\nabla f(x, y)$ captures how $f(x, y)$ deviates from average of f in neighborhood of (x, y)
- for discrete lattice network, Laplacian operator corresponds to **Laplacian matrix**
- in image data (where pixels are connected in a lattice) we can use **Laplacian convolution kernel** to **detect edges**
- **spectral graph convolution** allows to detect "boundaries" or "discontinuities" in graphs with arbitrary topology

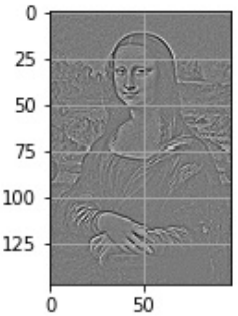


image after applying **Laplace filter**

Laplace filter

$$\omega = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \end{array}$$

Graph Convolutional Networks (GCN)

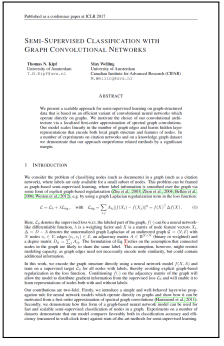
- message passing with self-loops and symmetric degree-normalization defines **Graph Convolutional Networks (GCN)** → T Kipf, M Welling, 2016
- update rule in **message passing layer of GCN** given as

$$h_i^{(k)} := \sigma\left(\mathbf{W}^{(k)} \sum_{j \in N(i) \cup \{i\}} \frac{h_j^{(k-1)}}{\sqrt{d_i d_j}}\right)$$

where $\mathbf{W}^{(k)} \in \mathbb{R}^{d^{(k)}} \times d^{(k-1)}$ are **learnable weights** and σ is **non-linear activation function**

- message passing layer k maps **node representations** $h_i^{(k-1)} \in \mathbb{R}^{d^{(k-1)}}$ to $h_i^{(k)} \in \mathbb{R}^{d^{(k)}}$
- we can add **dense classification layers** after k message passing layers to learn **latent representations** of nodes

Thomas Kipf



Published as a conference paper at ICLR 2017

SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

Thomas N. Kipf
University of Amsterdam
T.N.Kipf@uva.nl

Max Welling
University of Amsterdam
Canadian Institute for Advanced Research (CIFAR)
M.Welling@uva.nl

ABSTRACT

We present a scalable approach for semi-supervised learning on graph-structured data that is based on an efficient variant of convolutional neural networks which operate directly on graphs. We motivate the choice of our convolutional architecture via a localized first-order approximation of spectral graph convolutions. Our model scales linearly in the number of graph edges and learns hidden layer representations that encode both local graph structure and features of nodes. In a number of experiments on citation networks and on a knowledge graph dataset we demonstrate that our approach outperforms related methods by a significant margin.

1 INTRODUCTION

We consider the problem of classifying nodes (such as documents) in a graph (such as a citation network), where labels are only available for a small subset of nodes. This problem can be framed as graph-based semi-supervised learning, where label information is smoothed over the graph via some form of explicit graph-based regularization (Zhu et al., 2003; Zhou et al., 2004; Belkin et al., 2006; Weston et al., 2012), e.g. by using a graph Laplacian regularization term in the loss function:

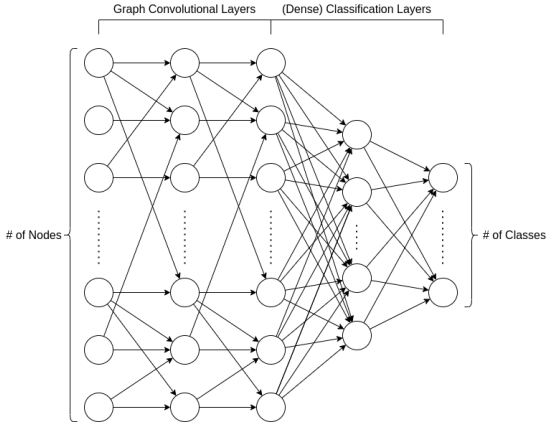
$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}} \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \|f(X_i) - f(X_j)\|^2 = f(X)^\top \Delta f(X), \tag{1}$$

Here, $\mathcal{L}_0$ denotes the supervised loss w.r.t. the labeled part of the graph, $f(\cdot)$ can be a neural network-like differentiable function, λ is a weighting factor and X is a matrix of node feature vectors X_i. $\Delta = D - A$ denotes the unnormalized graph Laplacian of an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes $v_i \in \mathcal{V}$, edges $(v_i, v_j) \in \mathcal{E}$, adjacency matrix A (binary or weighted) and degree matrix $D_{ii} = \sum_j A_{ij}$. The formulation of Eq. 1 relies on the assumption that connected nodes in the graph are likely to share the same label. This assumption, however, might restrict modeling capacity, as graph edges need not necessarily encode node similarity, but could contain additional information.

In this work, we encode the graph structure directly using a neural network model $f(X, A)$ and train on a supervised target $\mathcal{L}_0$ for all nodes with labels, thereby avoiding explicit graph-based regularization in the loss function. Conditioning $f(\cdot)$ on the adjacency matrix of the graph will allow the model to distribute gradient information from $\mathcal{L}_0$ and will enable it to learn representations of nodes both with and without labels.

Our contributions are two-fold. Firstly, we introduce a simple and well-behaved layer-wise propagation rule for neural network models which operate directly on graphs and show how it can be motivated from a first-order approximation of spectral graph convolutions (Hammond et al., 2011). Secondly, we demonstrate how this form of a graph-based neural network model can be used for fast and scalable semi-supervised classification of nodes in graphs. Experiments on a number of public benchmark datasets demonstrate that our model compares favorably both in classification accuracy and efficiency (measured in wall-clock time) against state-of-the-art methods for semi-supervised learning.

Computation Graph of GCN



tutorial notebooks

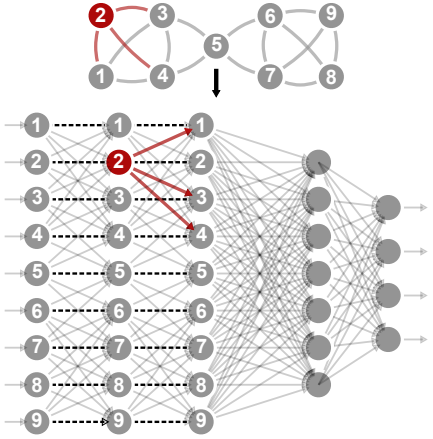
see notebooks 07 – 10 in repository at

→ https://github.com/pathpy/2026-netscix-tutorial/

in the tutorial notebooks we implement convolutional neural networks (CNN) for image classification from scratch and then generalize them to graph convolutional networks (GCN) using `pytorch-geometric`

Deep learning in complex networks

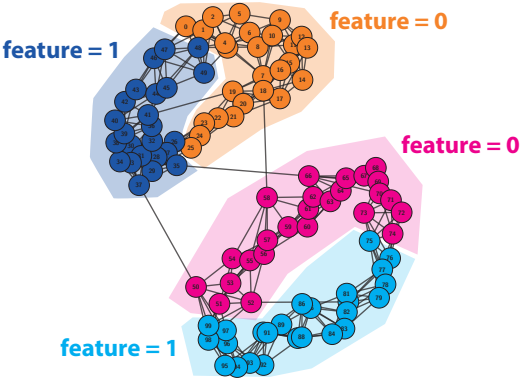
- **graph convolutional network (GCN)** = neural network architecture for **graph-structured data**
  → T Kipf, M Welling, 2017
- **neural message passing**: use complex network to iteratively update node features based on
  1. differentiable function with (learnable) parameters
  2. neighbor aggregation function
  3. non-linear activation function



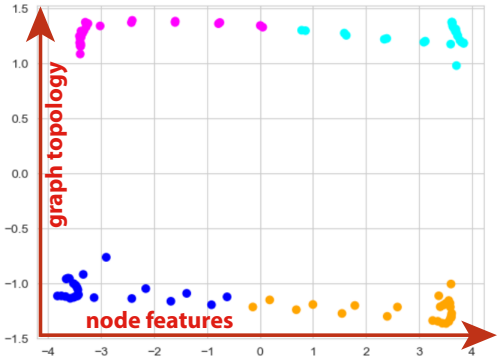
end-to-end representation learning

- use **differentiable loss function** to compare model output to ground truth (supervised setting)
- partial derivatives w.r.t. model parameters yield **gradients** that point towards local minimum of loss function
- GPU-accelerated **backpropagation algorithm** to learn "useful" **vector space representation**
  → DE Rumelhart, GE Hinton, RJ Williams, Nature, 1986

Graph representation learning



synthetic graph with **four classes of nodes**



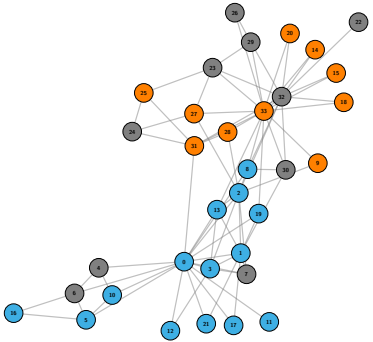
internal representation of nodes learned by Graph Convolutional Network

learned latent representation captures patterns in node features and network topology

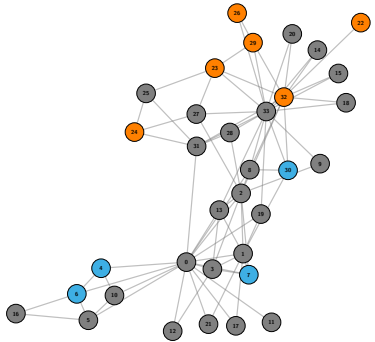
Example: GCN-based Node Classification

example

Karate club network with $n = 34$ nodes and $m = 77$ links, where ground truth node classes $\hat{y}$ are given by groups



training network with 70 % labeled nodes

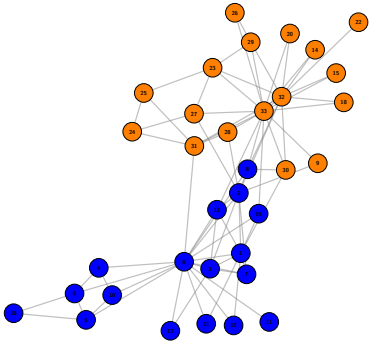


predicted node classes in test set (accuracy 90%)

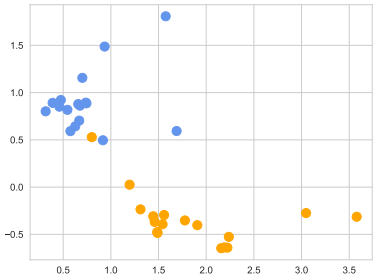
Example: Latent Node Representations

example

Karate club network with $n = 34$ nodes and $m = 77$ links, where ground truth node classes $\hat{y}$ are given by groups



Karate club network with ground truth node labels



latent representation of nodes extracted from activations in first hidden layer ($d = 16$) of GCN (representation in $\mathbb{R}^2$ via Truncated SVD)

Semi-supervised Learning in Graphs

- use of topological features enables application of GCN to **semi-supervised learning** in graphs

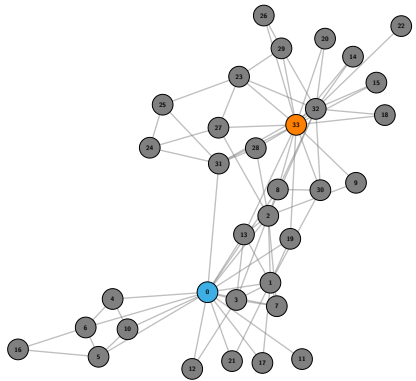
semi-supervised learning

machine learning techniques that can simultaneously use **large amounts of unlabeled data** as well as **small amounts of labeled data**

example

semi-supervised node classification in network with a single labeled node per class

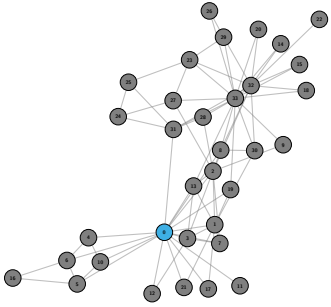
- message passing layers **smoothen** existing labels across unlabeled nodes close to labeled ones



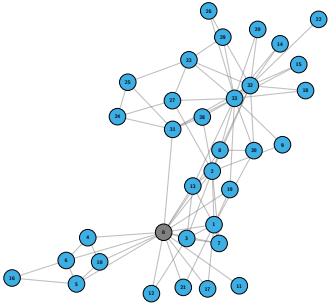
Example: Semi-Supervised Graph Learning

example

- **semi-supervised node classification** in Karate club network with $n = 34$ nodes and $m = 77$ links
- ground truth node class given for **one node in one community**



training network with **single labeled node**

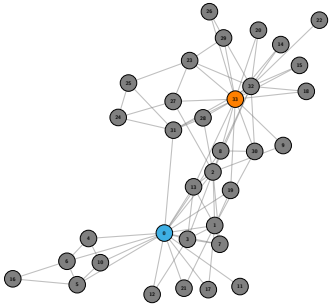


predicted node classes in test set using GCN with single message passing layer

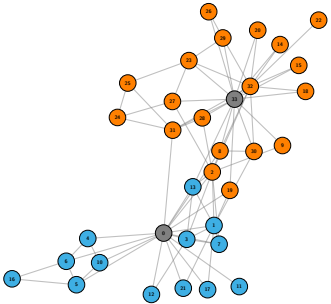
Example: Semi-Supervised Graph Learning

example

- **semi-supervised node classification** in Karate club network with $n = 34$ nodes and $m = 77$ links
- ground truth node class given for **two nodes in two communities**



training network with **two labeled nodes**



predicted node classes in test set using GCN with single message passing layer (accuracy 87.8%)

Research Challenge: Expressive power of GNNs

- which networks are distinguishable by GNNs?
- graph ismormorphism = basis to study **expressive power of neural message passing**

Weisfeiler-Leman (WL) color refinement algorithm

- start with identical node features (e.g. colors)
- update nodes iteratively by aggregating features of neighbors and assigning new features (e.g. R+R = B)
- repeat until no new features are assigned
- final node features = graph signature or “representation”

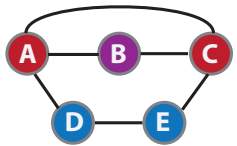
- **WL-algorithm** provides one-sided heuristic to distinguish non-isomorphic graphs

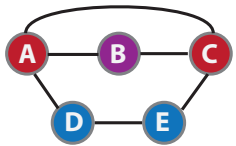
  → B Weisfeiler, A Leman, 1968

- **properly parameterized GNNs** not more powerful than WL-algorithm → C Morris et al., AAAI 2019

graph isomorphism

- consider graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$
- G_1 and G_2 are **isomorphic** iff there exists a bijection $\pi : V_1 \to V_2$ that preserves all edges





Research Challenge: GNNs for Heterophilic Networks

- most GNNs are designed for **homophilic networks** where connected nodes tend to have similar features or labels
- interpretation of GCN in terms of spectral graph convolution: low-pass filter that **smoothens node features across edges**
- in **heterophilic networks** connected nodes tend to have **different features** and/or labels
- simple GNN models like **GCN perform poorly on networks with (malignant) heterophilic patterns**

  → Z Pei et al., ICLR 2021
- **GNN architectures for heterophilic networks** are an active area of research → S Luan et al. 2024

The Heterophilic Graph Learning Handbook:
Benchmarks, Models, Theoretical Analysis, Applications and Challenges

Sitao Luan[1 2] Chenqing Hua[1 2] Qincheng Lu[2] Liheng Ma[1 2] Lirong Wu[3] Xinyu Wang[2] Minkai Xu[4]
Xiao-Wen Chang[5] Doina Precup[1 2 6] Rex Ying[7] Stan Z. Li[3] Jian Tang[1 2 8] Guy Wolf[1 2 9] Stefanie Jegelka[10 11]

Abstract

Homophily principle, *i.e.*, nodes with the same labels or similar attributes are more likely to be connected, has been commonly believed to be the main reason for the superiority of Graph Neural Networks (GNNs) over traditional Neural Networks (NNs) on graph-structured data, especially on node-level tasks. However, recent work has identified a non-trivial set of datasets where GNN's performance compared to the NN's is not satisfactory. Heterophily, *i.e.*, low homophily, has been considered the main cause of this empirical observation. People have begun to revisit and re-evaluate most existing graph models, including graph transformer and its variants, in the heterophily scenario across various kinds of graphs, *e.g.*, heterogeneous graphs, temporal graphs and hypergraphs. Moreover, numerous graph-related applications are found to be closely related to the heterophily problem. In the past few years, considerable effort has been devoted to studying and addressing the heterophily issue.

In this survey, we provide a comprehensive review of the latest progress on heterophilic graph learning, including an extensive summary of benchmark datasets and evaluation of homophily metrics on synthetic graphs, meticulous classification of the most updated supervised and unsupervised learning methods, thorough digestion of the theoretical analysis on homophily/heterophily, and broad exploration of the heterophily-related applications. Notably, through detailed experiments, we are the first to categorize benchmark heterophilic datasets into three sub-categories: malignant, benign and ambiguous heterophily. Malignant and ambiguous datasets are identified as the real challenging datasets to test the effectiveness of new models on the heterophily challenge. Finally, we propose several challenges and future directions for heterophilic graph representation learning. We hope this paper can become a friendly tutorial and an encyclopedia for your research on heterophilic graph representation learning. (Updated until June 30th, 2024.)

[1]Mila - Quebec Artificial Intelligence Institute [2]McGill University [3]Westlake University [4]Stanford University [5]Google DeepMind [6]Yale University [7]HEC Montreal [8]CIFAR AI Chair [9]Université de Montréal [10]Massachusetts Institute of Technology [11]TU Munich.
Correspondence to: Sitao Luan <sitao.luan@mail.mcgill.ca>.

→ https://arxiv.org/pdf/2407.09618

Research Challenge: Over-smoothing and Over-squashing

- number of message passing layers in GNNs determines **how far information can propagate** in a graph
- small number of layers -> GNN cannot capture **long-range dependencies**
- GNNs typically perform best with 2-3 layers

problems with "deep" GNNs

- **over-smoothing**: for large number of layers node representations become indistinguishable
- **over-squashing**: long-range dependencies may not captured due to bottlenecks in the graph

- mitigating over-smoothing and over-squashing is active area of research, addressed e.g. via **targeted rewiring**

→ Attali et al. 2024 or **Graph Transformers** → A Shehzad et al. 2025

Mitigating Over-Smoothing and Over-Squashing using Augmentations of Forman-Ricci Curvature

Lukas Fesser
Harvard University
lukas_fesser@fas.harvard.edu

Melanie Weber
Harvard University
mweber@seas.harvard.edu

Abstract

While Graph Neural Networks (GNNs) have been successfully leveraged for learning on graph-structured data across domains, several potential pitfalls have been described recently. Those include the inability to accurately leverage information encoded in long-range connections (*over-squashing*), as well as difficulties distinguishing the learned representations of nearby nodes with growing network depth (*over-smoothing*). An effective way to characterize both effects is discrete curvature: Long-range connections that underlie over-squashing effects have low curvature, whereas edges that contribute to over-smoothing have high curvature. This observation has given rise to *rewiring* techniques, which add or remove edges to mitigate over-smoothing and over-squashing. Several rewiring approaches utilizing graph characteristics, such as curvature or the spectrum of the graph Laplacian, have been proposed. However, existing methods, especially those based on curvature, often require expensive subroutines and careful hyperparameter tuning, which limits their applicability to large-scale graphs. Here we propose a rewiring technique based on Augmented Forman-Ricci curvature (AFRC), a scalable curvature notation, which can be computed in linear time. We prove that AFRC effectively characterizes over-smoothing and over-squashing effects in message-passing GNNs. We complement our theoretical results with experiments, which demonstrate that the proposed approach achieves state-of-the-art performance while significantly reducing the computational cost in comparison with other methods. Utilizing fundamental properties of discrete curvature, we propose effective heuristics for hyperparameters in curvature-based rewiring, which avoids expensive hyperparameter searches, further improving the scalability of the proposed approach.

1 Introduction

Graph-structured data is ubiquitous in data science and machine learning applications across domains. Message-passing Graph Neural Networks (GNNs) have emerged as a powerful architecture for Deep Learning on graph-structured data, leading to many recent success stories in a wide range of disciplines, including biochemistry [11], drug discovery [40], recommender systems [37] and particle physics [29]. However, recent literature has uncovered limitations in the representation power of GNNs [38], many of which stem from or are amplified by the inability of message-passing graph neural networks to accurately leverage information encoded in long-range connections (*over-squashing* [1]), as well as difficulties distinguishing the learned representations of nearby nodes with growing network depth (*over-smoothing* [18]). As a result, there has been a surge of interest in characterizing over-squashing and over-smoothing mathematically and in developing tools for mitigating both effects.

Among the two, over-squashing has received the widest attention, driven by the importance of leveraging information encoded in long-range connections in both node- and graph-level tasks. Over-smoothing has been observed to impact, in particular, the performance of GNNs on node-level

Lukas Fesser, Mitigating Over-Smoothing and Over-Squashing using Augmentations of Forman-Ricci Curvature. *Proceedings of the Second Learning on Graphs Conference (LoG 2023)*, PMLR 231, Virtual Event, November 27–30, 2023.

https://proceedings.mlr.press/v231/fesser24a/fesser24a.pdf

Research Challenge: GNNs for Noisy Data

- real-world data is often noisy, e.g. due to measurement errors or missing data
- GNNs can be sensitive to noise in the graph structure and/or node feature/labels

exemplary network science approach

- treat observed network as random realization from underlying **statistical ensemble** of graphs
- calculate **uncertainty estimates** for edges
- in message passing, use uncertainty estimates to weight messages

- improving **robustness of GNNs against noise** in input data is active area of research → Z Shafi et al. 2024

Enhancing Graph Neural Networks with Random Graph Ensembles

Jan von Pichowski Vincenzo Perri Ingo Scholtes
Chair of Machine Learning for Complex Networks
Centre for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg, Germany
{jan.pichowski,vincenzo.perri,ingo.scholtes}@uni-wuerzburg.de

Abstract

Graph Neural Networks (GNNs) have shown remarkable performance in various network analysis tasks. However, their results depend on the reliability of the network structure, making them sensitive to inherent variability in real-world data. This study investigates the use of graph ensembles to improve GNN performance, focusing on node classification tasks. We use random graph ensembles to define edge scores, quantifying the deviation of observed edge frequencies from those expected based on node activity. This approach allows us to distinguish between statistically significant connections and those potentially arising from random fluctuations in the network structure. We use this information to refine the message-passing procedure, aiming to enhance node representations and increase performance in downstream tasks. In our experiments, we propose and evaluate two ensemble-based strategies. Our results show that these strategies lead to better GNN performance in four out of five datasets. Our work lays a foundation for future research, opening new avenues for either applying other random graph ensembles to GNNs, or considering other graph-based tasks.

1 Introduction

In graph-based learning, the observed network structure is often assumed to accurately represent the underlying system. However, this assumption overlooks the inherent variability and uncertainty in network formation processes. Graph ensembles offer a way to address this challenge by providing a statistical baseline against which observed networks can be compared, enabling the distinction between meaningful structural patterns and random fluctuations. This methodology has proven valuable in various network analysis tasks, such as node clustering [1], identifying significant temporal patterns in dynamic networks [2], and various other applications [3]. Despite their demonstrated utility in discerning meaningful structural patterns, graph ensembles have not been used for refining the network topology in message-passing graph neural networks.

Identifying statistically meaningful patterns in networks is crucial for Graph Neural Networks (GNNs) due to their message-passing mechanism. GNNs leverage the structure of graph data to propagate information through nodes and edges, capturing complex relational patterns for predictive modeling. However, this reliance on network structure makes GNNs sensitive to the inherent variability in real-world datasets, which graph ensembles are particularly well-suited to address. While data cleaning approaches partially address similar challenges by removing spurious connections due to measurement errors [4–6], ensemble methods offer deeper insights. Beyond the binary classification of *correct* or *incorrect* edges, the existence and frequency of observations carry valuable statistical information in ensemble frameworks. For instance, the number of connections between two high-degree nodes may still be high in absolute terms, but it acquires a different interpretation when compared to its statistical expectation based on a random model that fixes node degrees. If the observed connection frequency is much higher or lower than expected, it suggests that the interaction is not merely a product of random chance but may instead indicate a meaningful pattern.

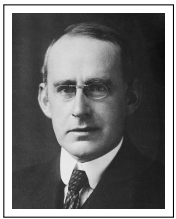
von Pichowski et al., Enhancing Graph Neural Networks with Random Graph Ensembles (Extended Abstract). Presented at the Third Learning on Graphs Conference (LoG 2024), Virtual Event, November 26–29, 2024.

→ https://openreview.net/pdf?id=N8tCUSzUFM

more on Feb 18, 11:30 (stream 1)

Research Challenge: GNNs for Temporal Networks

- graph = model for **possible causal influence** between nodes in a networked system
- **neural message passing** in graph neural networks uses **all possible paths**
- but: **cause must temporally precede effects**



Sir Arthur Stanley Eddington

1882 – 1944

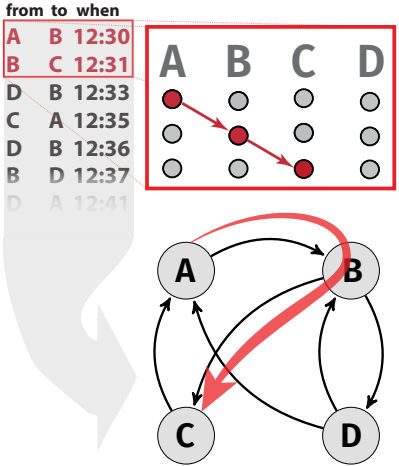
image credit: public domain

> " I shall use the phrase **'time's arrow'** to express this one-way property of time which has **no analogue in space**." → Sir Arthur Eddington

open issue

state-of-the-art temporal graph neural networks **ignore arrow of time** in temporal networks

→ L Qarkaxhija, V Perri, I Scholtes, PMLR 2022



Causality-aware temporal GNNs

- **De Bruijn graph neural network (DBGNN)** = deep learning architecture using **higher-order De Bruijn graphs**
- idea: perform neural message passing to higher-order model that **forces messages to follow arrow of time**

causality-aware graph representation learning

- gradient descent optimization yields **static vector space representation of temporal network** that captures …
  - topology of interactions between nodes
  - "causality" due to temporal order of interactions
- substantially increases model performance compared to state-of-the-art (temporal) GNNs → L Qarkaxhija, V Perri, I Scholtes, PMLR 2022

- integrating **network science insights into temporal networks** with time-aware GNNs is active area of research

Weisfeiler and Leman Follow the Arrow of Time: Expressive Power of Message Passing in Temporal Event Graphs

Franziska Heeg
Chair of Machine Learning for Complex Networks
Center for AI and Data Science (CAIDAS)
University of Wuerzburg
Würzburg, Germany
franziska.heeg@uni-wuerzburg.de

Jonas Sauer
Chair of Computational Analytics
Institute for Computer Science I
University of Bonn
Bonn, Germany
jsauer1@uni-bonn.de

Petra Mutzel
Chair of Computational Analytics
Institute for Computer Science I
University of Bonn
Bonn, Germany
pmutzel@uni-bonn.de

Ingo Scholtes
Chair of Machine Learning for Complex Networks
Center for AI and Data Science (CAIDAS)
University of Wuerzburg
Würzburg, Germany
ingo.scholtes@uni-wuerzburg.de

Abstract

An important characteristic of temporal graphs is how the directed arrow of time influences their *causal topology*, i.e. which nodes can possibly influence each other causally via time-respecting paths. Recent models have shown that this causal topology can be leveraged in graph neural networks (TGNNs). To further explore the potential of TGNNs, we lack a generalization of graph isomorphism for temporal graphs that fully captures their causal topology. Addressing this gap, we introduce the notion of *consistent event graph isomorphism*, which utilizes a time-unfolded representation of time-respecting paths in temporal graphs. We compare this definition with existing notions of temporal graph isomorphisms. We illustrate and highlight the advantages of our approach and develop a temporal generalization of the Weisfeiler-Leman algorithm to heuristically distinguish non-isomorphic temporal graphs. Building on this theoretical foundation, we derive a novel message passing scheme for temporal graph neural networks that operates on the event graph representation of temporal graphs. An experimental evaluation shows that our approach performs well on a temporal graph classification experiment.

1 Motivation

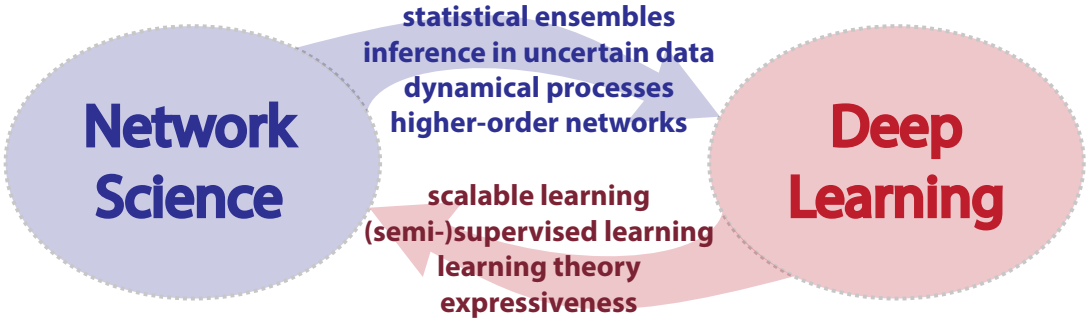
Graph neural networks (GNNs) have become a cornerstone of deep learning in relational data. They have recently been generalized to temporal GNNs (TGNNs) that capture patterns in time series data on *temporal graphs*, where edges carry timestamps. Such temporal graphs are often categorized into *snapshot graphs* and *continuous-time temporal graphs*. The former aggregate edges into discrete timesteps, e.g. yearly snapshots with many edges having identical timestamps. Such temporal graphs can naturally be represented as sequences of static snapshot, where each snapshot includes all edges occurring at a given timestamp. The resulting sequence of snapshots naturally lends itself to a generalization of static network analysis or graph learning techniques, as each individual snapshot can be interpreted as a static graph. In contrast, in *continuous-time temporal graphs* edges carry high-resolution, possibly unique timestamps. This implies that snapshot graphs are very sparse, which

Preprint. Under review.

→ https://arxiv.org/abs/2505.24438

more on Feb 18, 11:45 (stream 1)

Network Science vs. Deep Graph Learning



NetSci / NetSci-X
Phys. Rev. E
Phys. Rev. Lett.
Nature Comm.
Advances in Complex Systems
EPJ Data Science
…

NeurIps
ICLR
ICML
AAAI
SIGKDD
LoG
…

Ngā mihi!

Open Source library `pathpyG`

- based on `torch` and `torch-geometric`
- direct **connection to netzschleuder repository** for data loading
- makes it easy to apply GNNs to **static and temporal networks**
- **causality-aware temporal graph learning** via De Bruijn graph neural networks

PATHPY

www.pathpy.net

Deep Graph Learning will stall without Network Science

Christopher Blöcker [1,2] Martin Rosvall [3] Ingo Scholtes [1] Jevin D. West [4]

Abstract

Deep graph learning focuses on flexible and generalizable models that learn patterns in an automated fashion. Network science focuses on models and measures revealing the organizational principles of complex systems with explicit assumptions. Both fields share the same goal: to better model and understand patterns in graph-structured data. However, deep graph learning prioritizes empirical performance but ignores fundamental insights from network science. *Our position is that deep graph learning will stall without insights from network science*. In this position paper, we formulate six Calls for Action to leverage untapped insights from network science to address current issues in deep graph learning, ensuring the field continues to make progress.

a need for that to change, and argue for integrating network science insights into deep graph learning.

Deep graph learning faces several core challenges. Methods must augment data to cope with limited training data. They must pool node representations for graph-level learning. They must incorporate the time dimension of temporal graphs. And they must develop message-passing schemes that incorporate higher-order interactions beyond pairwise edges. Network science has been thinking about these issues for years, though from a different perspectives and with different motivations. However, deep graph learning does not leverage those insights. Network science offers solutions to address open challenges in deep graph learning: principled data augmentation, rigorous evaluation practices, higher-order models, and temporal pattern recognition. These solutions emerge by connecting network structure with function through principled methodologies, such as probabilistic generative models that provide principled null models for complex networks, statistical inference and network reconstruction methods for noisy relation data, and community detection techniques. These approaches can enhance the theoretical foundation and empirical insight of deep graph learning models.

Our position is that, without leveraging insights from network science, deep graph learning will stall. The past decade has seen rapid advances in deep graph learning architectures across various tasks and applications. However, challenges to apply state-of-the-art graph neural networks to real-world problems also expose limitations that we need to address [2, 3]. We need data augmentation techniques that model noisy or incomplete data to improve generalization. We need theoretically grounded pooling methods that minimize information loss. We need systematic frameworks to characterize which structural properties of temporal graph datasets drive performance. And we need message-passing architectures that capture higher-order interactions beyond pairwise edges. Insights from network science can help us to address these challenges, guiding deep graph learning toward principled architectures, improved interpretability, and more rigorous evaluation methods.

In this position paper, we formulate six *Calls for Action* to integrate insights from network science into deep graph learning and bridging the scientific fields. Ultimately, how-

1. Introduction

In 1982, John Hopfield introduced a neural network model that sparked a flurry of innovations: content-addressable memory, energy dynamics, error correction, and nonlinear architecture [1]. The Nobel Prize committee recently recognized the role these innovations played in the development of modern machine learning[1]. Less recognized, the paper influenced network science just as profoundly. The Hopfield network demonstrated the critical connection between network topology and the collective behaviour of complex systems—one of the enduring themes of network science and now one of the central challenges in deep graph learning. The two fields have diverged since Hopfield's paper. We see

[1]Chair of Machine Learning for Complex Networks, Center for Artificial Intelligence and Data Science (CAIDAS), Julius-Maximilians-Universität Würzburg, Germany [2]Department of Computing Science, Umeå Universitet, Umeå, Sweden [3]Integrated Science Lab (IceLab), Department of Physics, Umeå Universitet, Umeå, Sweden [4]Center for an Informed Public, Information School, University of Washington, Seattle, USA. Correspondence to: Christopher Blöcker <christopher.bloecker@uni-wuerzburg.de>.

Preprint. February 4, 2026.

[1]https://www.nobelprize.org/prizes/physics/2024/hopfield/facts/

→ https://arxiv.org/abs/2502.01177