

# Graph Neural Networks

Introduction and application to knowledge bases

---

**Indro Spinelli, Simone Scardapane**



ISPAMM Laboratory  
Sapienza University of Rome

# Introduction

---

The importance of graphs

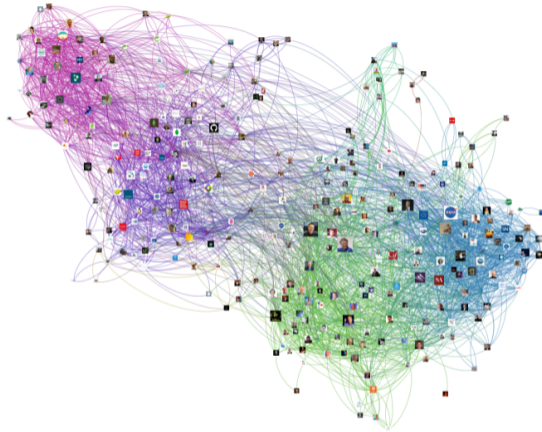
**Graph structured** data is all around us:

- ▶ Social Networks
- ▶ Communication Networks
- ▶ Energy Networks
- ▶ Transportation Networks
- ▶ Internet
- ▶ Recommender Systems
- ▶ Physical Systems
- ▶ Multi-Agent Systems
- ▶ Biological Systems (Molecules)
- ▶ Knowledge Bases

This kind of data has a great impact on the everyday life.



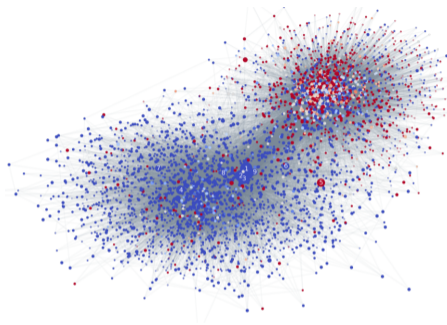
# Twitter followers graph



<http://allthingsgraphed.com/2014/11/02/twitter-friends-network/>



## What Graph Neural Networks can do



User credibility score on Twitter<sup>2</sup> (blu credible, red unreliable) used to detect fake news diffusion.

---

<sup>1</sup>Monti et al. “Fake News Detection on Social Media using Geometric Deep Learning”. 2019.

<sup>2</sup>Monti et al. “Fake News Detection on Social Media using Geometric Deep Learning”. 2019.



# Introduction

---

Learning on Graph Data

## A primer on graphs

Generically, a graph is a pair  $(\mathcal{V}, \mathcal{E})$ , where  $\mathcal{V}$  is a set of  $n$  **vertices** (or **nodes**), while  $\mathcal{E}$  is a set of **edges** connecting the graph.

Another common description is in term of matrices (all  $n \times n$ ):

- ▶ The **adjacency** matrix  $\mathbf{A}$ , where  $A_{ij} = 1$  if nodes  $i$  and  $j$  are connected.
- ▶ The **degree** matrix  $\mathbf{D}$ , with  $D_{ii} = \sum_j A_{ij}$ .
- ▶ The **laplacian** matrix  $\mathbf{L} = \mathbf{D} - \mathbf{A}$  or  $\mathbf{L} = \widehat{\mathbf{D}}^{-\frac{1}{2}} \widehat{\mathbf{A}} \widehat{\mathbf{D}}^{-\frac{1}{2}}$  where  $\widehat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$  and  $\widehat{\mathbf{D}}$  is built from  $\widehat{\mathbf{A}}$  for the normalized version.



## Features on graphs

In a machine learning context, graphs can have features at multiple levels:

- ▶ **Node-level** features: each node in the graph can have a set of features (e.g., *attributes of a user in a social network*);
- ▶ **Vertex-level** features: each vertex can have features/labels (e.g., *sequence of interactions between two users in a social network*);
- ▶ **Graph-level** features: features associated to the full graph (e.g., *time in which the social network snapshot was taken*).



Similarly, depending on the labels we can have many different tasks:

- ▶ Node **classification** or **regression**;
- ▶ Edge **classification** or **regression**;
- ▶ Graph **classification** or **regression**;
- ▶ **Generation** of all the above items...



## Families of algorithms on graphs

There are many different ideas for using ML solutions in the graph case:

- ▶ Graph kernels *Airola et al., 2008*;
- ▶ Graph neural networks *Scarselli et al., 2008*;
- ▶ Graph convolutional networks (spectral) *Bruna et al., 2014*,
- ▶ Graph convolutional networks (spatial) *Kipf and Welling, 2016*;
- ▶ Graph attention networks *Veličković et al., 2017*;
- ▶ Predict then Propagate *Klicpera et al., 2019*



# Graph Convolutional Networks

---

Introduction to the model

The de-facto standard of GNN is the Graph Convolutional Network **GCN** . We will exploit this framework to introduce **node classification**<sup>3</sup>, **link prediction**<sup>4</sup> and an extension of both tasks to **knowledge bases**<sup>5</sup>

---

<sup>3</sup>Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *Proc. 2017 International Conference on Learning Representations (ICLR)*. 2017.

<sup>4</sup>Thomas Kipf and Max Welling. “Variational Graph Auto-Encoders”. In: *ArXiv abs/1611.07308* (2016).

<sup>5</sup>Michael Sejr Schlichtkrull et al. “Modeling Relational Data with Graph Convolutional Networks”. In: *ESWC*. 2017.

Convolution is the process of adding each element (pixel/node) to its local neighbors, weighted by a kernel learned during the training procedure.

Local neighbors on an **image** typically represent the same region of space.

Local neighbors on a **graph** can represent any kind of relationship existing between the samples.



## From FNN to GNN

Let's start with standard fully-connected layer of a neural network that takes  $\mathbf{X}$  (where the rows are the nodes of a graph and the columns the features associated to them):

$$f(\mathbf{X}) = \sigma(\mathbf{X}\Theta) .$$

This is linear model, operates **independently** on every node.

If we pre-multiply by the matrix  $\mathbf{L}$  that encodes the graph structure:

$$f(\mathbf{X}) = \sigma(\mathbf{L}\mathbf{X}\Theta) ,$$

we will **propagate** the update across the neighbours.

## Single Node Update

Implementation wise the single node update can be represented as:

$$\mathbf{x}_i^{(l)} = \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{\sqrt{\deg(i)} \cdot \sqrt{\deg(j)}} \cdot \left( \Theta \cdot \mathbf{x}_j^{(l-1)} \right).$$

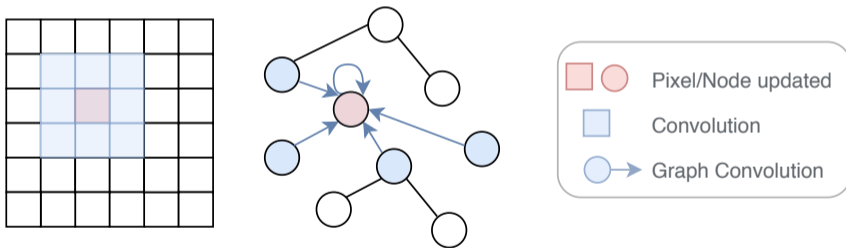
This formula can be summarized with the following steps:

### Update of the $i^{\text{th}}$ node

1. Transform the  $j^{\text{th}}$  neighboring node features by a weight matrix  $\Theta$ .
2. Normalize by the product of square roots of their degree.
3. Sum them up for every node in  $\mathcal{N}(i) \cup \{i\}$ .
4. Eventually apply an activation function.



## 2D Convolution vs Graph Convolution



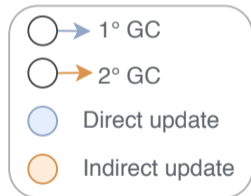
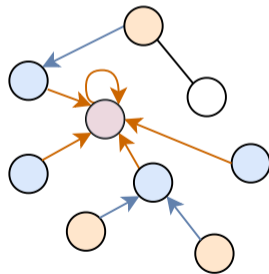
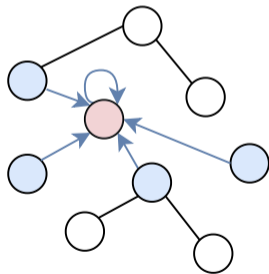


## More Layers!

$$\mathbf{H}_1 = \sigma(\mathbf{L}\mathbf{X}\Theta_1),$$

$$\mathbf{H}_2 = \sigma(\mathbf{L}\mathbf{H}_1\Theta_2).$$

- ▶  $\mathbf{L}\mathbf{X}$  propagates the information across the 1-hop neighbors.
- ▶  $\mathbf{L}\mathbf{H}_1$  propagates the information across the 2-hop neighbors.



# Graph Convolutional Networks

---

Tasks

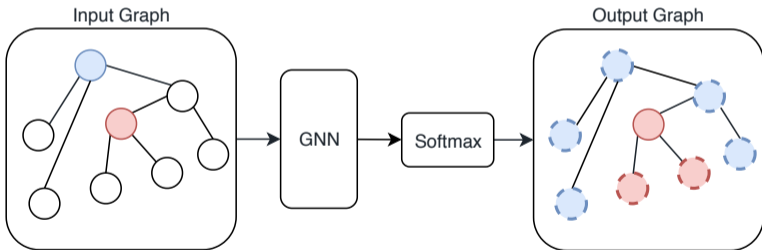
The two tasks we are interested in are:

- ▶ Node Classification: label data samples by taking into account their neighbours.
- ▶ Link Prediction: predict most likely links in the graph.

Both tasks are usually performed in a **transductive** settings, thus we have the advantage that we can train the model on the entire dataset, even if only a subset (the labelled one) can be used for driving the optimization process.

## Node classification

For the node classification, we can stack two graph convolutional layer to form a Graph Neural Network. A  $\text{softmax}()$  applied on each node embedding of the last layer will provide the prediction  $\hat{y}$ .



## Node classification

The training procedure minimizes the cross-entropy loss only on the labeled nodes  $\mathcal{Y}$  (remember the transductive setting):

$$\Theta^* = \arg \min \sum_{i \in \mathcal{Y}} \text{CrossEntropy}(\mathbf{y}_i, \hat{\mathbf{y}}_i) .$$

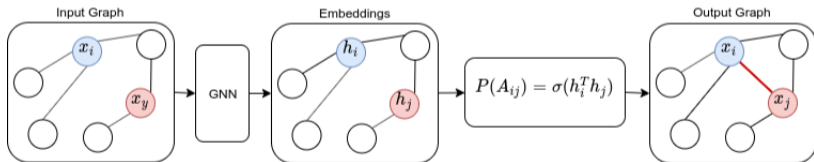
These kinds of models are usually trained with the entire graph as input.

For larger graphs, it is possible to implement a batching strategy exploiting sub-graphs.

## Link Prediction

In link prediction, the objective is to predict whether two nodes in a network are likely to have a link.

We can get the probability for each edge by taking the inner product of the embedding produced by a GNN.



Also in this case the model is optimized by minimizing the cross-entropy loss function.

# Graph Convolutional Networks

---

Modelling Relational Data

## More Relations!

**Knowledge graphs** enable a wide variety of applications, including question answering, information retrieval and many other tasks in the field of **Relational Learning**.

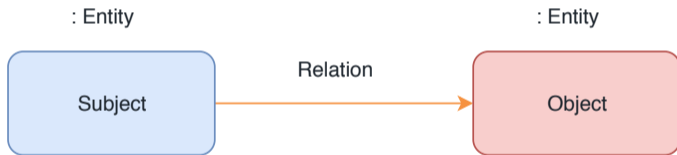
This kind of graphs are able to model many different relations at the same time.

Knowledge graphs are directed and labeled, in  $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$  the nodes  $v_i \in \mathcal{V}$  represent the **entity**, and edges  $(v_i, r, v_j) \in \mathcal{E}$  are labeled with the **relation** type  $r \in \mathcal{R}$ .

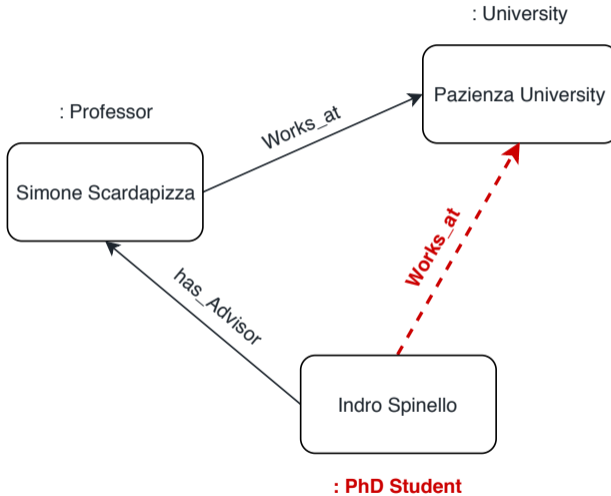


# Subject, Relation, Object

A basic unit of a knowledge graph contains:



# An Example



Known entities and relations.



Missing entities and relations.

They exist in the real world but are not modeled into the graph.

## The curse of missing things

Even the largest and newest knowledge bases, despite enormous effort invested in their maintenance, are **incomplete**.

Missing information can severely **reduce** the **performance** of downstream applications.

For this reason node classification and link prediction with **GNN** are of a great interest for this field for **restoring** incomplete knowledge bases.

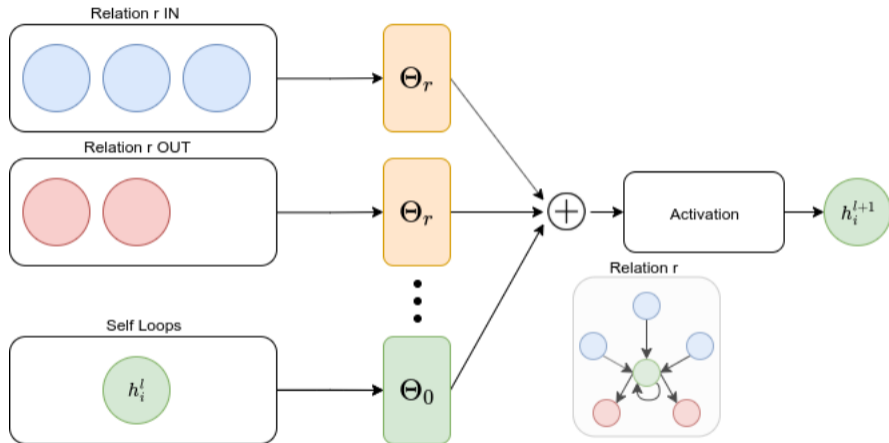
$$\mathbf{x}_i^{(l)} = \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \frac{1}{C_{r,i}} \cdot \Theta_r \cdot \mathbf{x}_j^{(l-1)} + \Theta_0 \cdot \mathbf{x}_i^{(l-1)}.$$

This formula can be summarized with the following steps:

### Update of the $i^{\text{th}}$ node

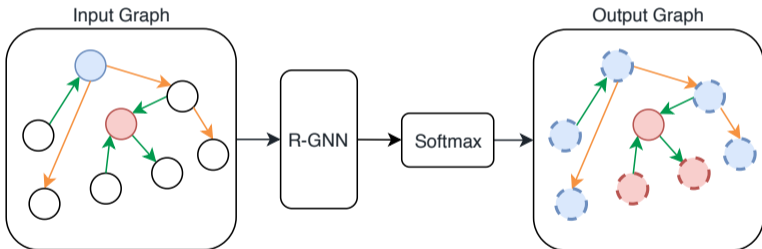
1. Transform the  $i^{\text{th}}$  node by the weight matrix of self-loops  $\Theta_0$ .
2. Transform the  $j^{\text{th}}$  neighboring node of relation  $r$  by the weight matrix  $\Theta_r$ .
3. Normalize by problem-specific constant such as  $C_{i,r} = |\mathcal{N}_i^r|$ .
4. Sum for every node in involved in relation  $r$  with the  $i^{\text{th}}$  node and every  $r \in \mathcal{R}$ .
5. Eventually apply an activation function.

# Visualization of Relational GCN



## Node classification

To classify the nodes in the graph with missing entities associated we can simply repeat what we have done before by stacking two Relational Convolutional Layers forming an R-GNN.



The optimization problem do not change.

## Link Prediction in Knowledge Bases

Knowledge bases are **directed** and **labeled** graphs.

Therefore predicting a new link translates into a prediction of the triplet (**subject**, **relation**, **object**).

The score of a possible connection will be given by the DisMult<sup>6</sup> factorization for each possible combination:

$$f(s, r, o) = h_s^T R_r h_o$$

where  $R_r$  is a relation specific diagonal matrix.

The optimization process has to take into account the greater amount of complexity.

<sup>6</sup>Bishan Yang et al. "Embedding Entities and Relations for Learning and Inference in Knowledge Bases". In: *CoRR* abs/1412.6575 (2014).

## Final remarks

---

A formal derivation for GCNs



Consider the eigendecomposition of the Laplacian matrix:

$$\mathbf{L} = \mathbf{\Phi} \mathbf{\Lambda} \mathbf{\Phi}^T, \quad (1)$$

where  $\mathbf{\Phi}$  is the matrix of eigenvectors, and  $\mathbf{\Lambda}$  a diagonal matrix of (sorted) eigenvalues. For a node signal  $\mathbf{x} \in \mathbb{R}^n$  with a single channel, the Graph Fourier transform (GFT) is defined as:

$$\hat{\mathbf{x}} = \mathbf{\Phi}^T \mathbf{x}, \quad (2)$$

This operation has a lot of analogies with the classical Fourier transform on signals, where the eigenvalues corresponds to the 'frequencies' of the graph<sup>7</sup>.

<sup>7</sup>Aliaksei Sandryhaila and Jose Moura. "Discrete Signal Processing on Graphs". In: *IEEE Transactions on Signal Processing* 61 (Oct. 2012). DOI: 10.1109/TSP.2013.2238935.

## Graph convolutional layer (formal)

Remember: **convolution in time is a product in the frequency domain.**

By analogy, we can define a convolutional layer in a formal way as follows<sup>8</sup>:

$$\mathbf{g} = \phi(\overbrace{\Phi \Gamma \Phi^T}^{\text{Filtering+Anti-GFT}} \mathbf{x}), \quad (3)$$

GFT

where  $\Gamma$  is a diagonal matrix acting as a filter on the graph frequencies.

---

<sup>8</sup>Joan Bruna. "Spectral Networks and Deep Locally Connected Networks on Graphs". In: 2014.

Computing the **GFT is expensive**, and we would need to do it at every layer!

It can be shown that the **GCN is a simplification** of the previous model, where we consider filtering operations that are *linear* w.r.t. to the frequency coefficients.

## Final remarks

---

Current Research Interests

Right now there is a huge interest on how to propagate the information on the graph.

The matrix  $L$  in the vanilla GCN is the renormalized Laplacian, which improves the numerical stability w.r.t the Laplacian itself.

Graph attention networks<sup>9</sup> leverages a masked **self-attention** mechanism to combine the nodes of the neighborhood.

Personalized propagation of neural predictions (PPNP)<sup>10</sup>, exploits **PageRank** to derive an improved propagation scheme.

---

<sup>9</sup>Petar Velickovic et al. “Graph Attention Networks”. In: *ArXiv abs/1710.10903* (2017).

<sup>10</sup>Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. “Predict then Propagate: Graph Neural Networks meet Personalized PageRank”. In: *ICLR*. 2018.



## And Many More

Opening research fields includes:

- ▶ Building deeper models.
- ▶ Adversarial Machine learning on graphs.
- ▶ Extension to spatio-temporal graph data.
- ▶ New applications.

# Software

---

- ▶ **PyTorch Geometric** ([https://github.com/rusty1s/pytorch\\_geometric](https://github.com/rusty1s/pytorch_geometric))  
almost a standard in the research community;
- ▶ **Deep Graph Library** (<https://www.dgl.ai/>)  
PyTorch and MXNet, almost production-ready;
- ▶ **Spektral** (<https://github.com/danielegrattarola/spektral>)  
Keras-based, less maintained;
- ▶ **Graph Net** ([https://github.com/deepmind/graph\\_nets](https://github.com/deepmind/graph_nets))  
DeepMind, less content, less documentation.





# Graph Neural Networks

From ISPAMM Lab

---

**Indro Spinelli, Simone Scardapane**



ISPAMM Laboratory  
Sapienza University of Rome

## Introduction

---

Real data is not perfect

## The Problem of Missing Data

Many real-world datasets are affected by the problem of **missing values**.

ID	A	B	C
1	2.7	cat	0
2	0.5	NaN	1
3	NaN	dog	NaN

Fitting a model or performing training with a dataset that has a lot of missing values can drastically impact the model's quality.

For this reason, the field of Missing Data Imputation (MDI) has attracted significant attention.

## State Of The Art

A great number of methods have been proposed to solve the MDI task.

- ▶ mean imputation Little et al. 1986
- ▶ MICE Van Buuren et al. 2011
- ▶ k-nearest neighbors Acuna et al. 2004
- ▶ random forest Stekhoven et al. 2011
- ▶ linear models  
Lakshminarayan et al. 1996
- ▶ matrix factorization Mnih et al. 2008
- ▶ support vector machines  
Wang et al. 2006
- ▶ neural networks Yoon et al. 2018

We will use them to validate the performance of our contribution.

# Predictive Imputation

Many of these methods derive from **classical** machine learning algorithms (e.g., for regression and classification) with a few modifications.

This is possible because data imputation can be framed under a predictive framework<sup>11</sup>.

Some of these algorithms build a **global** model for data imputation, others instead, use **similar** data points to infer the missing components.

---

<sup>11</sup>Dimitris Bertsimas, Colin Pawlowski, and Ying Daisy Zhuo. “From Predictive Methods to Missing Data Imputation: An Optimization Approach.”. In: *Journal of Machine Learning Research* 18 (2017), pp. 196–1.

## Contribution

---

Graph Imputation Neural Network

## Our contribution

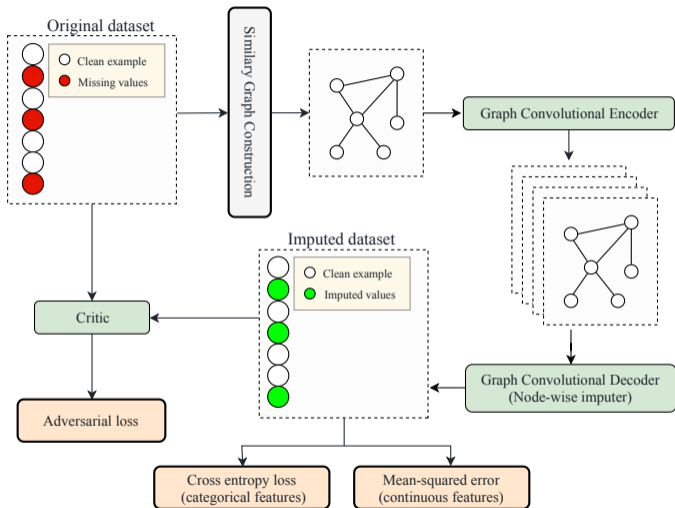
The contribution of this thesis work consists of a new framework for MDI, **GINN**<sup>12</sup>(Graph Imputation Neural Network), that exploits both **similar** data points for each imputation and a **global** model built from the overall data set.

This is possible thanks to a new class of **neural network** that is able to model and exploit structured information (in the form of relationships between samples), by working in the domain of **graphs**.

---

<sup>12</sup>Indro Spinelli, Simone Scardapane, and Aurelio Uncini. “Missing Data Imputation with Adversarially-trained Graph Convolutional Networks”. In: *Submitted to Neural Networks (Elsevier)* (2019).

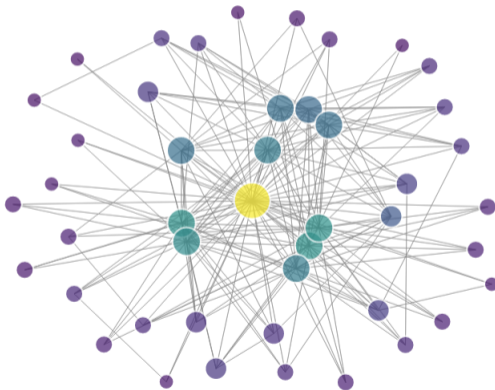
# GINN Schematics





## An Example on Iris

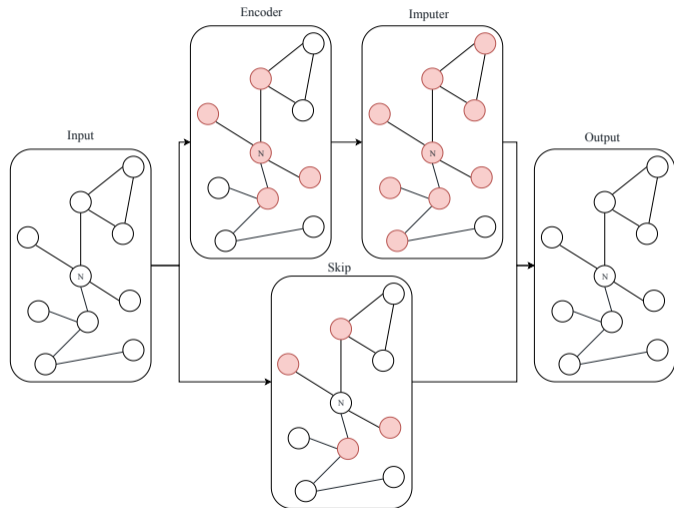
The **similarity graph** describes the structural proximity between samples. For each node the color intensity represents the number of connections and the size the number of missing features.



# Our's Graph Convolutional Autoencoder

$$\mathbf{H} = \text{ReLU}(\mathbf{L}\mathbf{X}\Theta_1),$$

$$\hat{\mathbf{X}} = \text{Sigmoid}(\mathbf{L}\mathbf{H}\Theta_2 + \tilde{\mathbf{L}}\mathbf{X}\Theta_3).$$



# Adversarial Training

To improve the quality of the imputed values, we use an adversarial training strategy where a **critic**, a feedforward network in our case, learns to distinguish between **imputed** and **real** data.

The autoencoder is thus trained to **fool the critic** with an additional component in the loss function.

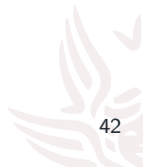
Having an adversarial loss during reconstruction forces the imputed vector to lie close to the natural distribution of the original data.

## Graph Convolutional Autoencoder

$$L_{Reconstruction} = \alpha \text{MSE}(\mathbf{X}, \hat{\mathbf{X}}) + (1 - \alpha) \text{CE}(\mathbf{X}, \hat{\mathbf{X}}),$$

$$L_{Total} = L_{Reconstruction} + \lambda L_{Adversarial}.$$

- ▶ **MSE**( $\mathbf{X}, \hat{\mathbf{X}}$ ) is the mean squared error for numerical variables.
- ▶ **CE**( $\mathbf{X}, \hat{\mathbf{X}}$ ) is the cross entropy loss for categorical variables.



## Extension

---

From Imputation to Augmentation

## Our contribution

We propose a new method to perform data augmentation for general **vectorial** data sets.

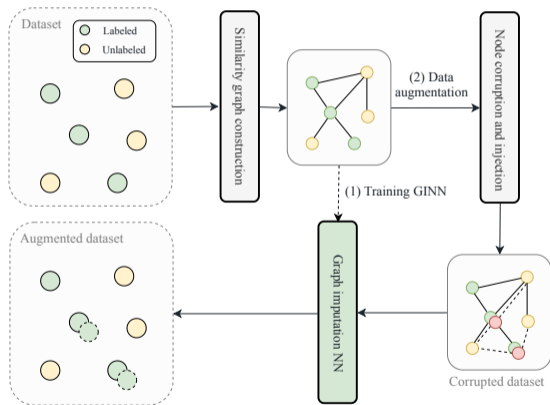
We reformulated the problem of data **augmentation** as a problem of data **imputation** under extreme level of noise<sup>13</sup>.

With this reformulation we can use **GINN** (Graph Imputation Neural Network), our new framework for missing data imputation.

---

<sup>13</sup>Indro Spinelli et al. “Efficient data augmentation using graph imputation neural networks”. In: *Presented at WIRN conference 2019* (2019).

Overall schema of our data augmentation pipeline:



## Corrupt, Inject, Impute

To perform the data augmentation we apply three additional steps:

1. **Corrupt** randomly some of the labelled nodes by removing up to 80% of their features.
2. **Inject** these new nodes in the graph recomputing on-the-fly their connections with unlabelled nodes. Only the non-zero elements of the corrupted nodes are used for the computation.
3. **Impute** the missing feature of these new nodes using the previously trained GINN architecture, generating new labelled samples that can be added to the data set.



Thank you!