GRAPH
DRAWING

Graph Neural Networks for Graph Drawing

Gabriele Ciravegna, Matteo Tiezzi, Marco Gori

February, 3^{th} 2022

INRIA, Université Côte d'Azur, Nice, France

SAILab, University of Siena, Siena, Italy

Overview

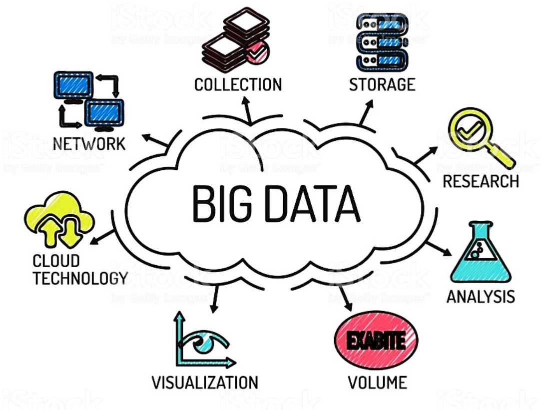
Introduction

What is a graph?

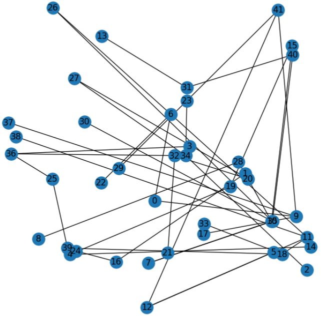
- In **mathematics**, graphs are a set of nodes connected by links
- **Nodes** represent objects possibly with their features
- **Links** represent the relations between the objects and may also be characterized by some properties

Why Graph Drawing?

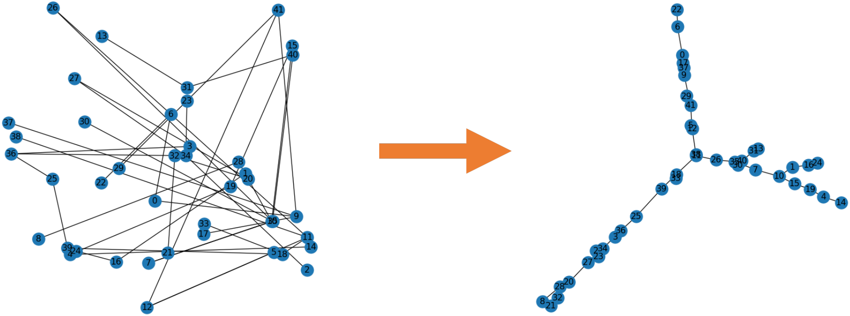
- Big Data are more and more available these days
- Apart from collection, storage, analysis, of crucial importance is their **visualization**



Why Graph Drawing? (II)



Why Graph Drawing? (II)



Graph Drawing Techniques

Standard Graph Drawing Techniques

Optimization-based method

- They find the position of the nodes defining a minimum of a loss function
- Mostly employ **Stochastic Gradient Descent**
- Iteratively, they update the positions of (some) of the nodes following the direction of the gradient

Standard Graph Drawing Techniques

Optimization-based method

- They find the position of the nodes defining a minimum of a loss function
- Mostly employ **Stochastic Gradient Descent**
- Iteratively, they update the positions of (some) of the nodes following the direction of the gradient

The loss function needs to encode a certain **Aesthetic criteria**

Standard Graph Drawing Techniques

Optimization-based method

- They find the position of the nodes defining a minimum of a loss function
- Mostly employ **Stochastic Gradient Descent**
- Iteratively, they update the positions of (some) of the nodes following the direction of the gradient

The loss function needs to encode a certain **Aesthetic criteria**

1. Minimum number of crossing edges

Standard Graph Drawing Techniques

Optimization-based method

- They find the position of the nodes defining a minimum of a loss function
- Mostly employ **Stochastic Gradient Descent**
- Iteratively, they update the positions of (some) of the nodes following the direction of the gradient

The loss function needs to encode a certain **Aesthetic criteria**

1. Minimum number of crossing edges
2. Perpendicular crossing angles

GRAPH DRAWING

Standard Graph Drawing Techniques

Optimization-based method

- They find the position of the nodes defining a minimum of a loss function
- Mostly employ **Stochastic Gradient Descent**
- Iteratively, they update the positions of (some) of the nodes following the direction of the gradient

The loss function needs to encode a certain **Aesthetic criteria**

1. Minimum number of crossing edges
2. Perpendicular crossing angles
3. Minimum variance of edge length

Standard Graph Drawing Techniques

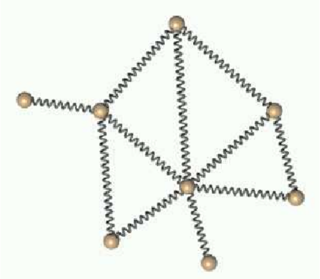
Optimization-based method

- They find the position of the nodes defining a minimum of a loss function
- Mostly employ **Stochastic Gradient Descent**
- Iteratively, they update the positions of (some) of the nodes following the direction of the gradient

The loss function needs to encode a certain **Aesthetic criteria**

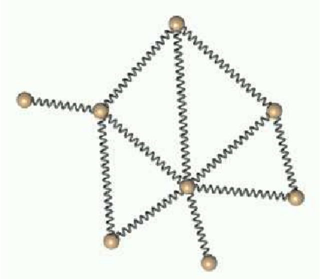
1. Minimum number of crossing edges
2. Perpendicular crossing angles
3. Minimum variance of edge length
4. Minimum edge bends

Force-directed Graph Drawing



Force-directed graph drawing techniques employ forces on the set of nodes to obtain 2 and 3

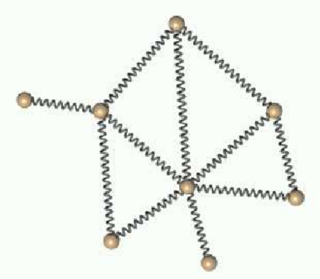
Force-directed Graph Drawing



Force-directed graph drawing techniques employ forces on the set of nodes to obtain 2 and 3

- **Spring-like** forces are used to attract linked nodes that are far from each other

Force-directed Graph Drawing



Force-directed graph drawing techniques employ forces on the set of nodes to obtain 2 and 3

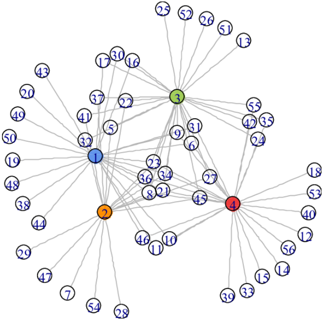
- **Spring-like** forces are used to attract linked nodes that are far from each other
- Repulsive forces are imposed on the nodes that are too near (similarly to the repulsion of **electric particles**)

Kamada-Kawai algorithm

Graph-distance $\delta_{i,j}$ should be equal to the actual distance of the nodes p_i, p_j

$$\text{STRESS} : \sum_{i<j} w_{ij} \left(||p_i - p_j|| - \delta_{ij} \right)^2 \quad (1)$$

- δ_{ij} graph theoretic distance (or shortest-path), the # of hops to reach another node
- p_i, p_j coordinates of vertices i, j
- $w_{ij} = \delta_{ij}^{-1}$, is a weighting factor inversely proportional to d_{ij}



What about avoding intersections?

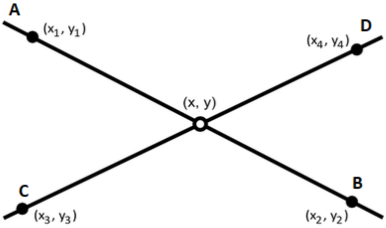
Finding the intersection of 2 lines is simple as resolving this system:

$$\begin{cases} a_1x + b_1y + c_1 = 0, \\ a_2x + b_2y + c_2 = 0 \end{cases} \tag{2}$$

What about avoding intersections?

Finding the intersection of 2 lines is simple as resolving this system:

$$\begin{cases} a_1 x + b_1 y + c_1 = 0, \\ a_2 x + b_2 y + c_2 = 0 \end{cases} \qquad (2)$$



If we are only given the four vertices, we have also to find a_i, b_i, c_i

$$a_1 = B.x - A.x$$
$$b_1 = B.y - A.y$$
$$c_1 = a_1 * A.x + b1 * Ay$$

Finding Intersections

```
def LineIntersection(A, B, C, D)

    # Line AB represented as a1x + b1y = c1
    a1 = B.x - A.x
    b1 = A.y - B.y
    c1 = a1*A.x + b1*A.y

    # Line CD represented as a2x + b2y = c2
    a2 = D.x - C.x
    b2 = C.y - D.y
    c2 = a2*C.x + b2*C.y

    determinant = a1*b2 - a2*b1

    if determinant == 0:
        return False
    else
        return True
```

However, you can clearly see that we cannot optimize this:

- The result is either 0 or 1
- We cannot optimize **noncontinuous** loss functions with SGD

The Neural Aesthete

Neural Aesthete

A Neural Network can **learn** to predict boolean values!

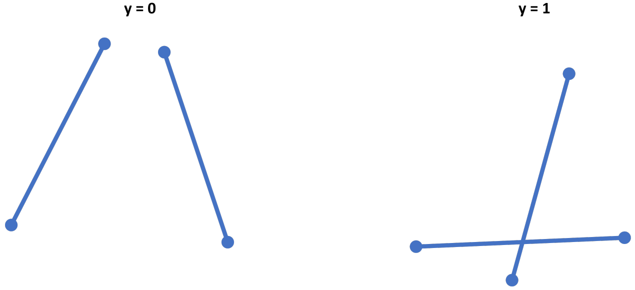
Neural Aesthete

A Neural Network can **learn** to predict boolean values!

Therefore we can:

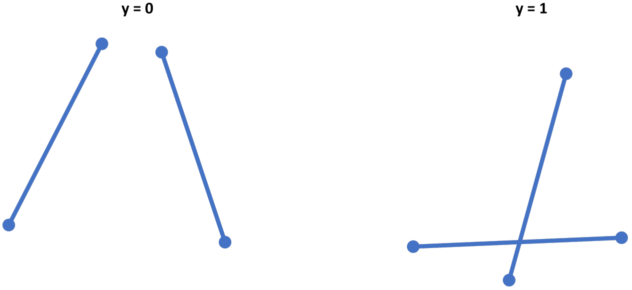
- Create a **dataset** of intersecting/non-intersecting lines
- Train an **MLP** to predict the edge intersection
- Use its loss function to move the points in such a way that edge do **not** intersect

Training a Neural Aesthete



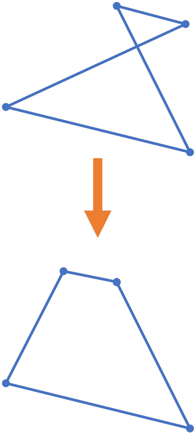
- **10000 lines** intersecting and non-intersecting with vertices $p \in [0, 1]^2$

Training a Neural Aesthete



- **10000 lines** intersecting and non-intersecting with vertices $p \in [0, 1]^2$
- A simple MLP reaches $\sim$ **98 %** accuracy on a separate test set
- It employs [100, 300, 10] hidden neurons and is trained to minimize the standard Cross Entropy loss with an SGD optimizer.

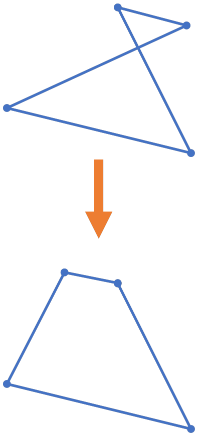
Neural Aesthete to Draw Graphs



The Cross Entropy loss $H(\hat{y}, y)$ can not only be used to train, but also **to draw!**

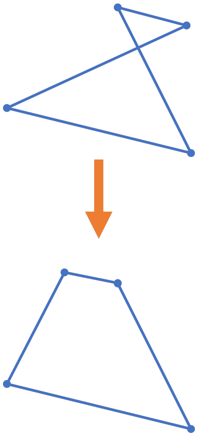
Neural Aesthete to Draw Graphs

When **training**:



$$f^* = \arg\min_f \sum_i^N H(\hat{y}_i, y_i)$$

Neural Aesthete to Draw Graphs



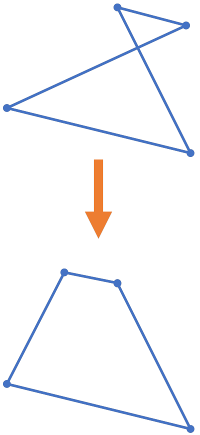
When **training**:

$$f^* = \arg\min_f \sum_i^N H(\hat{y}_i, y_i)$$

To draw non-intersecting lines:

$$e_1^*, e_2^* = \arg\min_{e_1, e_2} H(\hat{y}, \boxed{0}) \qquad (3)$$

Neural Aesthete to Draw Graphs



When **training**:

$$f^* = \arg\min_f \sum_i^N H(\hat{y}_i, y_i)$$

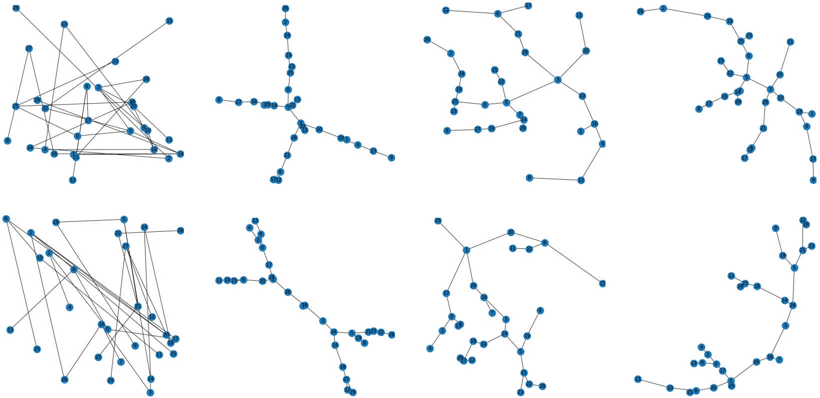
To draw non-intersecting lines:

$$e_1^*, e_2^* = \arg\min_{e_1, e_2} H(\hat{y}, 0) \tag{3}$$

* we go towards non-intersection $H(\hat{y}, 0)$
* $f(e_i, e_j) = \hat{y}$ is the trained MLP
* edges are defined as pair of points $e_i = (p_h, p_k)$

Examples

START	STRESS	AESTHETE	COMBINED



Graph Neural Network

GRAPH DRAWING

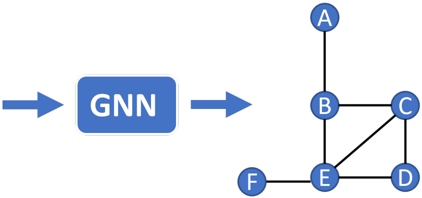
Why using GNN?

No need to optimize every graph

- During training graph layouts are learnt
- During test the GNN:
  - In input receives the adjacency matrix
  - In output produces the graph layout

	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	1	0	1	0
C	0	1	0	1	1	0
D	0	0	1	0	1	0
E	0	1	1	1	0	1
F	0	0	0	0	1	0

Adjacency matrix



How to train GNN for Graph Drawing?

Supervised approach

y: position of the output nodes

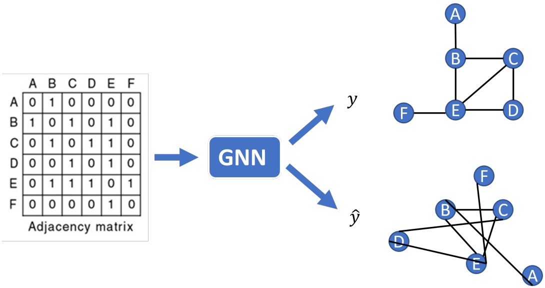
- Labels created by standard graph drawing techniques
- E.g. Kamada Kawai force-directed algorithm

How to train GNN for Graph Drawing?

Supervised approach

y: position of the output nodes

- Labels created by standard graph drawing techniques
- E.g. Kamada Kawai force-directed algorithm



	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	1	0	1	0
C	0	1	0	1	1	0
D	0	0	1	0	1	0
E	0	1	1	1	0	1
F	0	0	0	0	1	0

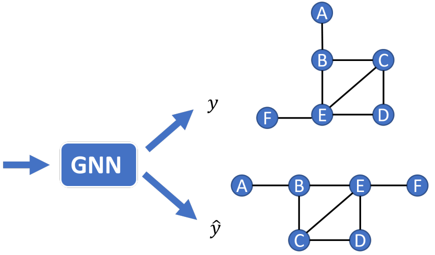
Training GNN: Supervised Approach

Problem: difficult learning setting

- Two graph drawings could be equally "good" but very different
- $\mathcal{L} = \|P - \hat{P}\|_2$??
  - P: position of the nodes as for label y
  - $\hat{P}$ positions predicted by the GNN

	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	1	0	1	0
C	0	1	0	1	1	0
D	0	0	1	0	1	0
E	0	1	1	1	0	1
F	0	0	0	0	1	0

Adjacency matrix



$\mathcal{L} >> 1$

Training GNN: Supervised Approach (II)

Solution: **Procrustes Statistic**

$$R^2 = 1 - \frac{\left(\mathrm{Tr}(P^T \hat{P} \hat{P}^T P)^{\frac{1}{2}}\right)^2}{\mathrm{Tr}(P^T P)\mathrm{Tr}(\hat{P}^T \hat{P})} \tag{4}$$

$Tr(A) = \sum_{i=1}^{n} a_{ii}$

Training GNN: Supervised Approach (II)

Solution: **Procrustes Statistic**

$$R^2 = 1 - \frac{\left(\mathrm{Tr}\left(P^T \hat{P} \hat{P}^T P\right)^{\frac{1}{2}}\right)^2}{\mathrm{Tr}\left(P^T P\right) \mathrm{Tr}\left(\hat{P}^T \hat{P}\right)} \tag{4}$$

$Tr(A) = \sum_{i=1}^{n} a_{ii}$

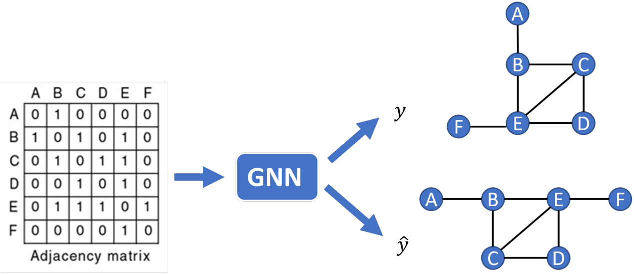
Good **loss function**:

- Measure shape difference among graph layouts
- Independent to affine transformations:
  - Translation
  - Rotation
  - Scaling

Training GNN: Supervised Approach (II)

Solution: **Procrustes Statistic**

$$R^2 = 1 - \frac{\left(\mathrm{Tr}\left(P^T \hat{P} \hat{P}^T P\right)^{\frac{1}{2}}\right)^2}{\mathrm{Tr}\left(P^T P\right) \mathrm{Tr}\left(\hat{P}^T \hat{P}\right)} \tag{5}$$



$R^2 \sim 0$

GNN Models

Message Passing Neural Networks (MPNNs)

$$x_{(i,j)}^{(t-1)} = \mathtt{MSG}^{(t)}\left(x_i^{(t-1)}, x_j^{(t-1)}, l_{(i,j)}\right) \tag{6}$$

$$x_i^{(t)} = \mathtt{AGG}^{(t)}\left(x_i^{(t-1)}, \sum_{j \in \mathcal{N}_i} x_{(i,j)}^{(t-1)}, l_i\right) \tag{7}$$

GNN Models

Message Passing Neural Networks (MPNNs)

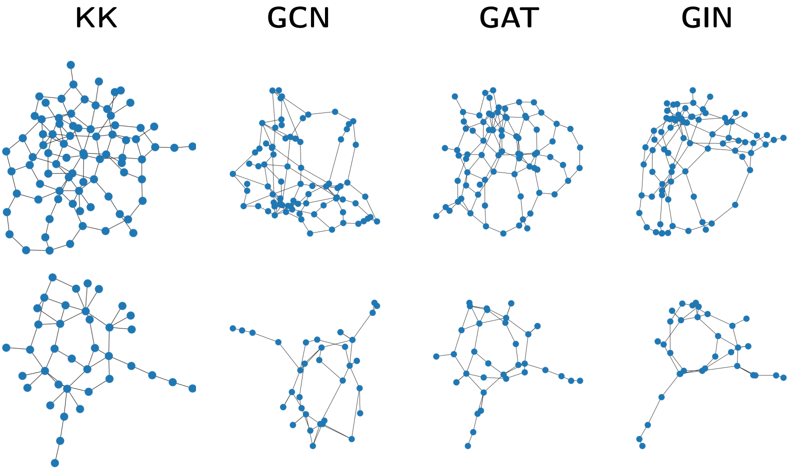
$$x_{(i,j)}^{(t-1)} = \mathtt{MSG}^{(t)}\left(x_i^{(t-1)}, x_j^{(t-1)}, l_{(i,j)}\right) \tag{6}$$

$$x_i^{(t)} = \mathtt{AGG}^{(t)}\left(x_i^{(t-1)}, \sum_{j\in\mathcal{N}_i} x_{(i,j)}^{(t-1)}, l_i\right) \tag{7}$$

Table: Common implementations of AGG mechanisms.

GCN: Mean	$\sigma\big(W_0^{(t)} x_v{}^{(t-1)} + \sum_{u\in\mathcal{N}_v} c_{u,v} W_1^{(t)} x_u{}^{(t-1)}\big)$
GAT: Attention	$\sigma\big(\sum_{u\in\mathcal{N}_v} \alpha_{u,v}^{(t-1)} W^{(t)} x_u^{(t-1)}\big)$
GIN: Sum	$\mathrm{MLP}^{(t)}\big((1+\epsilon)x_v{}^{(t-1)} + \sum_{u\in\mathcal{N}_v} x_u{}^{(t-1)}\ \big)$

Results Supervised Approach



Results Supervised Approach (II)

Model	Train Loss		Test Loss	
	Rome	Sparse	Rome	Sparse
GCN	5.70	7.90	5.74	8.09
GAT	**4.18**	**5.55**	**4.29**	**5.76**
GIN	4.26	5.66	7.91	9.24

Table: Average Stress loss value obtained on the training set and test set by the various models and for each dataset.

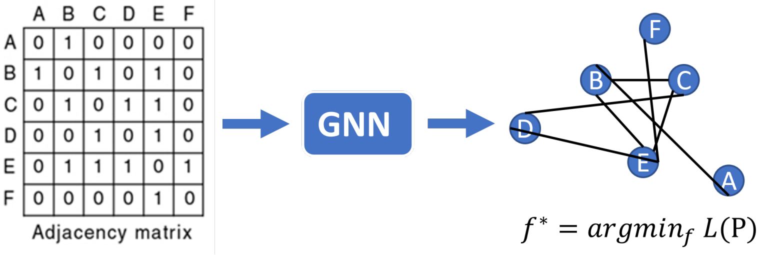
Training GNN: Unsupervised Approach

Iteratively optimizing aesthetic loss at training time:

$$\mathcal{L}(P) = \mathrm{STRESS}(P) + \lambda H(P). \tag{8}$$

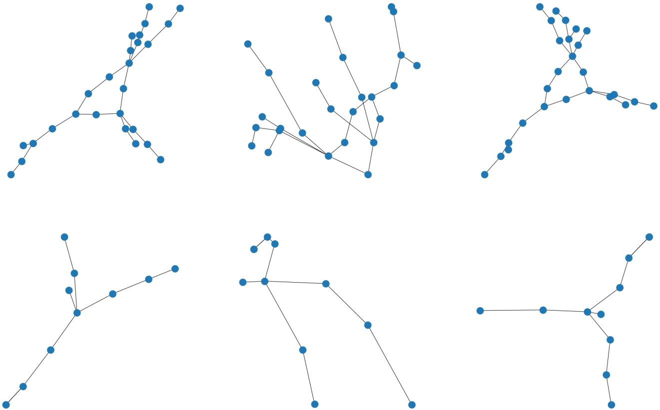
Combination of STRESS(P) and Neural Aesthete loss $H(P)$

At test time, only a forward pass is computed



Result Unsupervised Approach

STRESS NA-Crossing Combined



What is missing?

Node, Edge initialization

Random initialization of both node and edge feature vectors

$\Rightarrow$ GNNs **do not learn**

- Not sufficient information in the adjacency matrix only

GRAPH DRAWING

Node, Edge Initialization (II)

Possible initialization:

- Enriching edge features with **shortest path**
  - PROs: help optimization and generalization
  - CONs: does not scale

GRAPH DRAWING

Node, Edge Initialization (II)

Possible initialization:

- Enriching edge features with **shortest path**
  - PROs: help optimization and generalization
  - CONs: does not scale (require processing a *complete graph*)
- Enriching Node feature with **Laplacian eigenvectors**
  - PROs: describe the position of nodes inside the graph
    describe the neighboring structure

$$x_i^{(0)} = L_i,$$

$$L = \mathrm{I} - D^{-1/2} A D^{-1/2} = U^T \Lambda U,$$