Graph Neural Networks (GNNs)

Valentin Cuzin-Rambaud : valentin.cuzin-rambaud@univ-lyon1.fr

17 September 2026

Organization

- All information on the course's website.
- The content of this course is largely based on the previous version by Rémy Cazabet.
- Evaluation:
  - Project: 30%
    - 5% report, code, questions in last session (1 Oct.)
    - 25% written exam on the project
- Calendar:
  - Tuesday 17 Sep. - **Lecture and exercises**
  - Tuesday 24 Sep. - **TP and Project**
  - Tuesday 1 Oct. - **Last project session**
  - Sunday 11 Oct. at 5pm - **project report submission date**
  - Monday 12 Oct. **written exam on the project**
  - Thursday 2 Nov. - **BIML exam**

Table of content

Graphs

About Networks in Data Science

Networks often refers to real systems

- Tabular data
- Structured data:

Text	Sequence of words: each item is before or after the other ones. One-dimensional organization
Images	Each pixel has a position in a 2D grid; it is on the left, right, top, or bottom compared with the other ones. Two-dimensional organization
Variants	Video (3D), time series (1D continuous), spatial (2D/3D continuous)
Networks	Neighborhoods are not constrained. The graph is the structure; nodes are in a non-Euclidean space.

Networks generalize discrete structures (text, images, videos)

Why Graphs ?

Graph is the mathematical representation of a network

A graph can model any dataset containing relationships between entities.

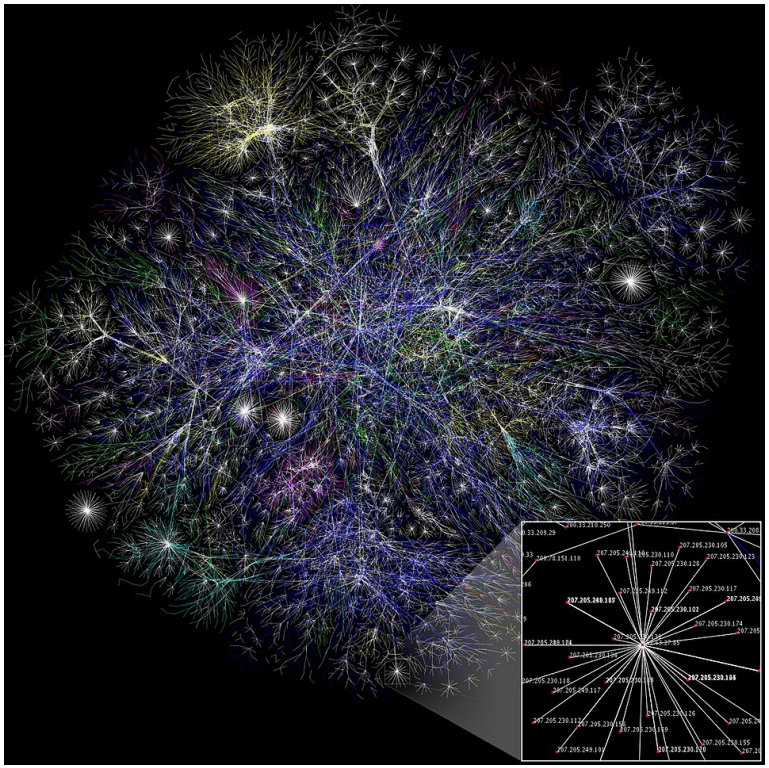


Figure: A Complex Networks

How to define a Graph mathematically?

Plenty Complex Networks, plenty tasks !!

- Infrastructure networks
- Biological networks
- Social networks
- Knowledge Graphs
- Recommender Systems

- Link prediction (follow recommendation, drug/illness relationship)
- Node classification (bot detection in social media)
- Attribute regression (age of individuals in a social media)
- Link classification/regression (relationship is: friend/colleague/lover ?)
- Graph classification (Molecule classification)
- Community detection
- ...

Useful network repositories:
PyG dataset | networkrepository | umich | netzschleuder | OGB.

Plenty Complex Networks, plenty tasks !!

- Infrastructure networks
- Biological networks
- Social networks
- Knowledge Graphs
- Recommender Systems

- Link prediction (follow recommendation, drug/illness relationship)
- Node classification (bot detection in social media)
- Attribute regression (age of individuals in a social media)
- Link classification/regression (relationship is: friend/colleague/lover ?)
- Graph classification (Molecule classification)
- Community detection
- ...

Useful network repositories:
PyG dataset | networkrepository | umich | netzschleuder | OGB.

Plenty Complex Networks, plenty tasks !!

- Infrastructure networks
- Biological networks
- Social networks
- Knowledge Graphs
- Recommender Systems

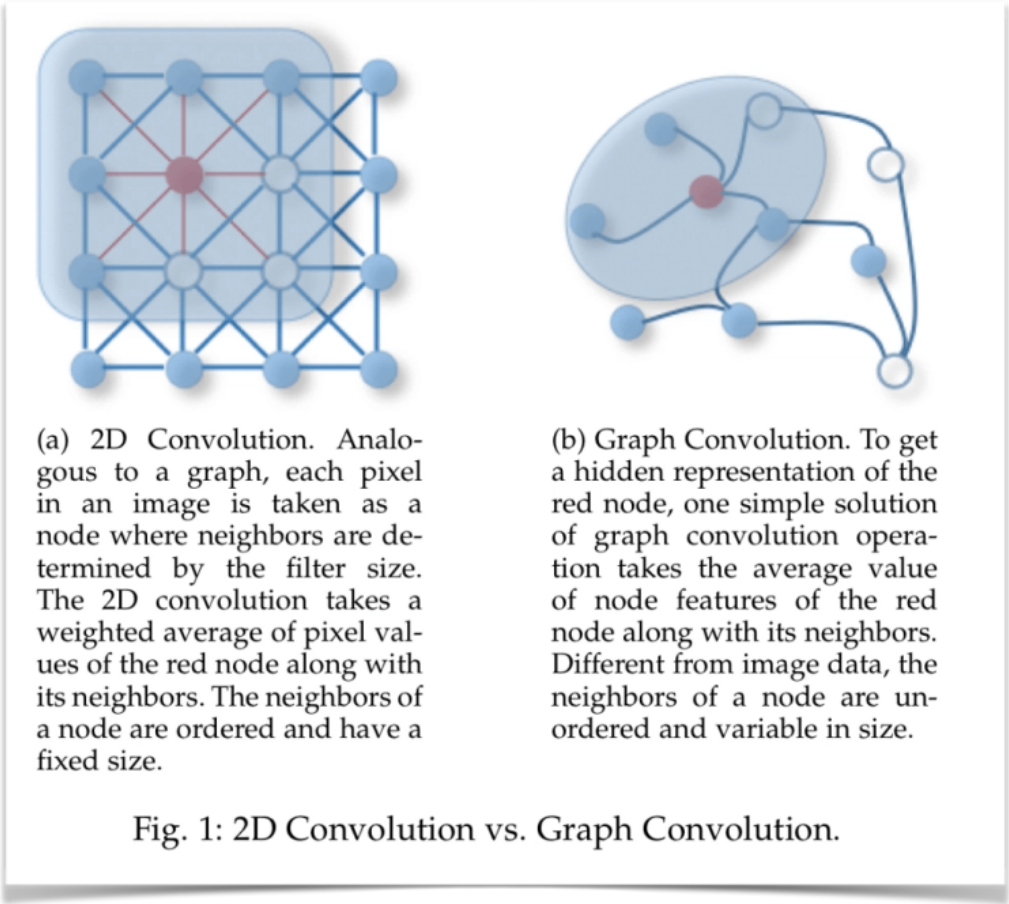
- Link prediction (follow recommendation, drug/illness relationship)
- Node classification (bot detection in social media)
- Attribute regression (age of individuals in a social media)
- Link classification/regression (relationship is: friend/colleague/lover ?)
- Graph classification (Molecule classification)
- Community detection
- ...

Useful network repositories:
PyG dataset | networkrepository | umich | netzschleuder | OGB.

Graph Convolutional Network (GCN)

Why Neural Networks ?

Graph Convolution



(a) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes a weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.

(b) Graph Convolution. To get a hidden representation of the red node, one simple solution of graph convolution operation takes the average value of node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

Fig. 1: 2D Convolution vs. Graph Convolution.

Figure: Wu, Zonghan, et al. "A comprehensive survey on graph neural networks." IEEE transactions on neural networks and learning systems 32.1 (2020): 4-24. arXiv preprint arXiv:1901.00596

Message passing interpretation

Tell me who your friends are, I'll tell you who you are

- Each node sends its information to its neighbors
- Nodes "combine" their neighbors' information (+ their own) to construct new features

Message passing interpretation

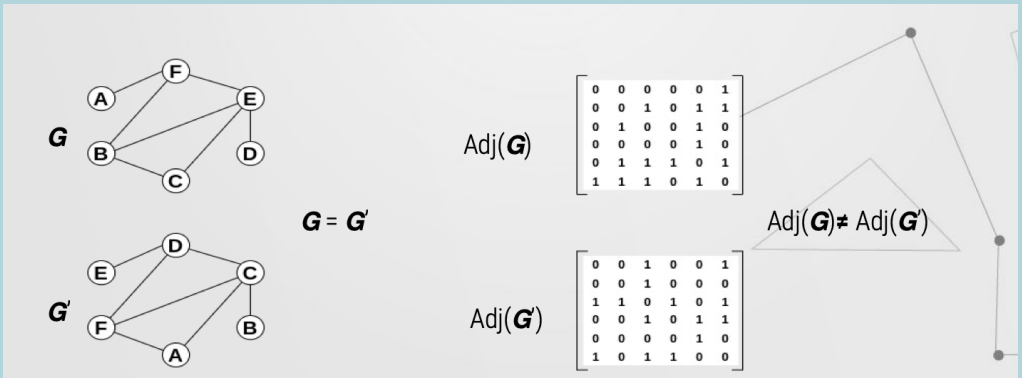
Tell me who your friends are, I'll tell you who you are

- Each node sends its information to its neighbors
- Nodes "combine" their neighbors' information (+ their own) to construct new features

Problem: varying number of neighbors

In networks, the number of neighbors is different for each node. Impossible to have a fixed-size convolution kernel.

Moreover, position in the adjacency matrix (row, column) has no meaning:



Graph Convolution Network (GCN) Layer Intuition

(see more)

Impossible to have a fixed-size kernel due to varying number of neighbors ...

GCN forward unfold in two components

1. Average the neighbors' features (pooling-like)
2. Compute the weighted sum of neighbors' values

GCN forward for layer *l*

- the first representation: $h_i^{(0)} = x_i$, where x_i node features
- the learnable matrix weight: W
- the neighborhoods including itself: $\tilde{\mathcal{N}_i}$
- the degree function: $deg(\cdot)$

$$\mathbf{h}_i^{(l+1)} = \sum_{j \in \tilde{\mathcal{N}_i}} \frac{1}{\sqrt{\deg(i)} \cdot \sqrt{\deg(j)}} \cdot \left(\mathbf{W}^\top \cdot \mathbf{h}_j^{(l)} \right) + \mathbf{b}$$

You can see a simple implementation here

GCN Layer in equation in matrix form

H : node features

A : adjacency matrix ($\widehat{A} = A + I$)

I : layer index

D : Degree matrix (degrees on the diagonal, $\widehat{D} = D + I$)

W : learnable weights

σ : activation function (often ReLU)

Important equation

$$H^{(l+1)} = f(H^{(l)}, A)$$

$$f(H^{(l)}, A) = \sigma(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

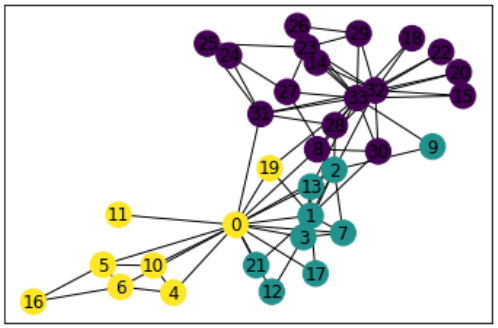
GCN Layer in equation in matrix form

Important equation

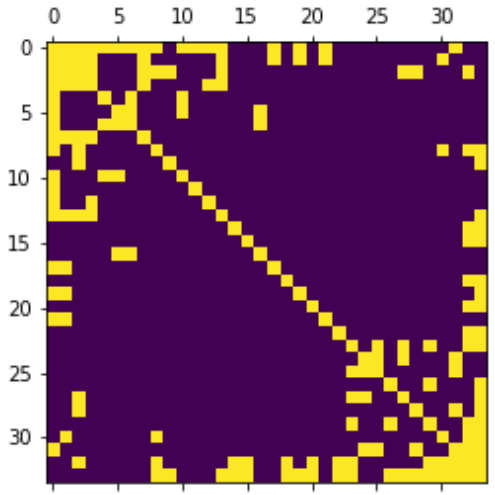
$$H^{(l+1)} = f(H^{(l)}, A) = \sigma(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

Adjacency Matrix A

Zackary Karate Club (with communities for reference)



$\widehat{A}$



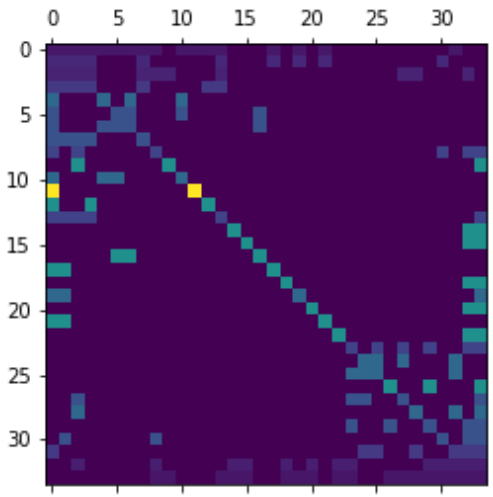
GCN Layer in equation in matrix form

Important equation

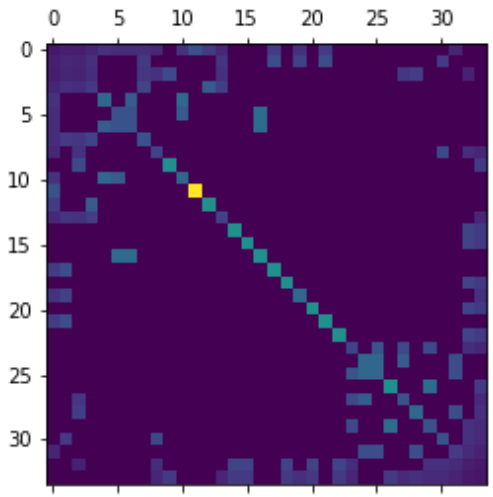
$$H^{(l+1)} = f(H^{(l)}, A) = \sigma(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

Normalization of the adjacency matrix

$\widehat{D}^{-1}\widehat{A}$



$\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}$

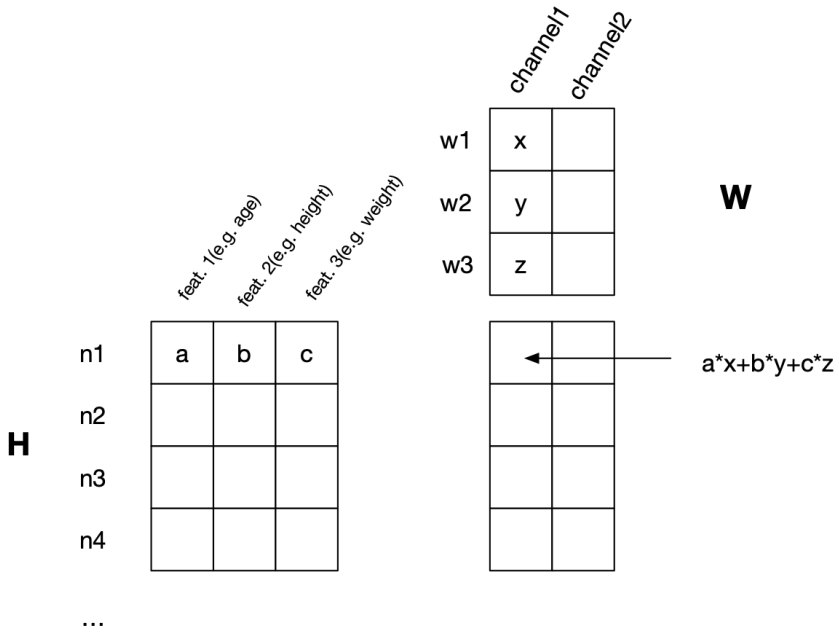


GCN Layer in equation in matrix form

Important equation

$$H^{(l+1)} = f(H^{(l)}, A) = \sigma(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

$$HW = \text{Combine features}$$

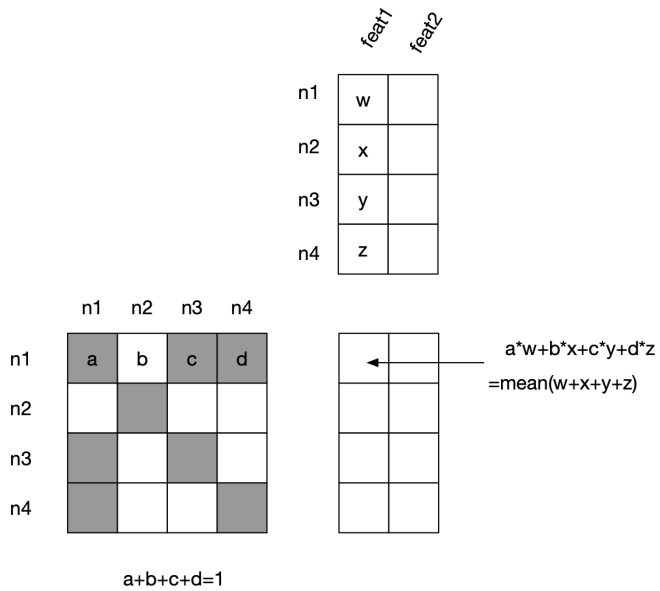


GCN Layer in equation in matrix form

Important equation

$$H^{(l+1)} = f(H^{(l)}, A) = \sigma(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

$A(HW) =$ **Average over neighbors**



Matrix Multiplication is associative

- $(\widehat{D}^{-1}\widehat{A}H)W$ = Embed Average of Neighbors features
- $\widehat{D}^{-1}\widehat{A}(HW)$ = Average Neighbors Embedding

GCN Layer in equation in matrix form

Important equation

$$H^{(l+1)} = f(H^{(l)}, A) = \sigma(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

Activation function

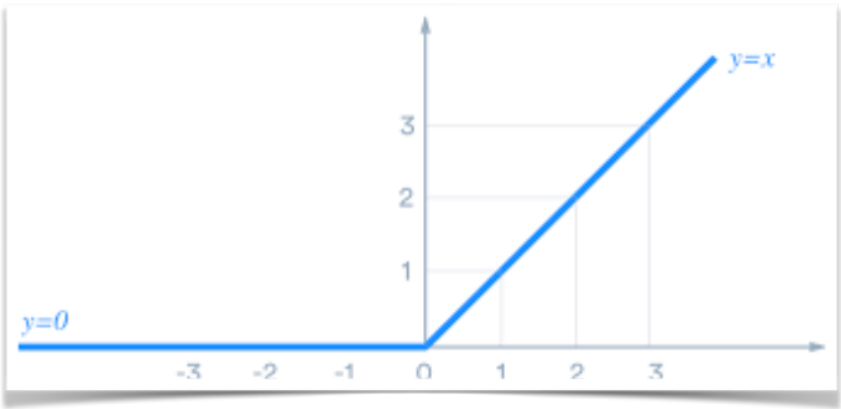


Figure: ReLU function, simple to differentiate and to compute (see more)

It introduces non-linearity

Backward Step

Learning the weights with **back-propagation**

- A **loss** function is defined to compare the predicted values with ground truth labels (at this point, we need some labels)
- The **derivative** of the cost function relative to weights is computed
- Weights are updated using **gradient descent** (i.e., weights are modified in the direction that will minimize the loss)
- https://en.wikipedia.org/wiki/Backpropagation

But graphs are inherently different from image/tabular datasets...

- Images/tabular
  - Each item is independent of the others
  - $\Rightarrow$ We train for each item independently
  - $\Rightarrow$ The test set is composed of new, never-seen items
- Graphs (general case)
  - A single graph, composed of (connected) nodes
  - $\Rightarrow$ Each node is treated as an independent item
  - $\Rightarrow$ But all node features are used in training
  - $\Rightarrow$ For node n, its features can belong to the train test while target in the test set
  - $\Rightarrow$ Semi-supervised learning

Backward Step

Learning the weights with **back-propagation**

- A **loss** function is defined to compare the predicted values with ground truth labels (at this point, we need some labels)
- The **derivative** of the cost function relative to weights is computed
- Weights are updated using **gradient descent** (i.e., weights are modified in the direction that will minimize the loss)
- https://en.wikipedia.org/wiki/Backpropagation

But graphs are inherently different from image/tabular datasets...

Example: Network of Twitter users

- Nodes: users
- Edges: followers
- Attributes: date joined, likes, geographical position, keywords,
- Target: Male/Female, Left/Right, etc.

We know targets for some users, but not all $\Rightarrow$ **Using all users' properties to guess the target for some users, training on the known ones**

GCN Model Architecture

We can represent our GCN Architecture as stacked convolutional layers

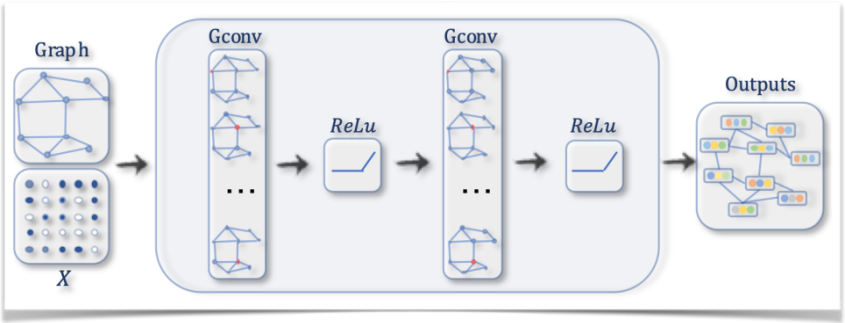
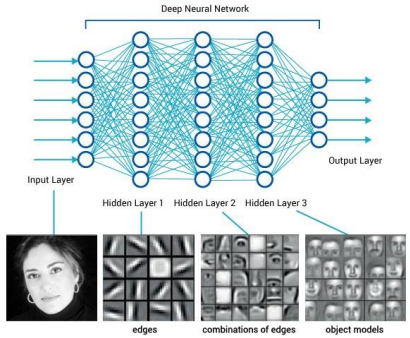


Figure: Wu, Zonghan, et al. "A comprehensive survey on graph neural networks." IEEE transactions on neural networks and learning systems 32.1 (2020): 4-24. arXiv preprint arXiv:1901.00596

Similar to convolutions in images, each convolution layer allows dependence on nodes farther in the network.



Layer 1: results depend only on direct neighbors Layer 2:

- direct neighbors' features are the result of Layer 1
- $\Rightarrow$ results depend on nodes at distance 1 and 2

Etc. . .
L GNN layers permit retrieving information from L-hop of nodes

Good news: average distance in real graphs is short 6 degrees of separation thus, even on a large graph, **a moderate number of convolutional layers** should allow to have impact from most of the graph

First exercise

Graph Attention Network (GAT)

Self-Attention Mechanism

Mechanisms coming mostly from Language models.

Transformers (as in GP**T**) are a particular type of self-attention (see more).

Graph Attention Network (GAT) Intuition, step by step...

(see more)

- In the normal GCN, a limit is the fix rule used to combine the neighbors attributes (weighted average)

$$\mathbf{h}_i^{(l+1)} = \sum_{j \in \tilde{\mathcal{N}_i}} \frac{1}{\sqrt{\deg(i)} \cdot \sqrt{\deg(j)}} \cdot \left(\mathbf{W}^\top \cdot \mathbf{h}_j^{(l)} \right) + \mathbf{b}$$

- Graph attention principle is to allow each node to ”choose“ what ”attention“ to give to each neighbor

$$\mathbf{h}_i^{(l+1)} = \sum_{j \in \tilde{\mathcal{N}_i}} \alpha_{ij} \cdot \left(\mathbf{W}^\top \cdot \mathbf{h}_j^{(l)} \right) + \mathbf{b}$$

  - with α_{ij} attention from i to j

GAT: computing attention head

Important equation

$$\mathbf{h}_i^{(l+1)} = \sum_{j \in \tilde{\mathcal{N}_i}} \alpha_{ij} \cdot \left(\mathbf{W}^\top \cdot \mathbf{h}_j^{(l)} \right) + \mathbf{b}$$

- Step 1: A learnable attention matrix converts the node embeddings into **new embeddings specific for Attention**

$$\mathbf{z_i} = \left(\mathbf{W}^\top \cdot \mathbf{h}_i \right) + \mathbf{b}$$

- Step 2: Concatenate both nodes embeddings $[\mathbf{z}_i || \mathbf{z}_j]$ and then compute an attention coefficient $\mathbf{a}_{ij}$ using learnable weights

$$\mathbf{e}_{ij} = \mathbf{a}^T [\mathbf{z}_i || \mathbf{z}_j]$$

- Step 3: Add an activation function $\mathbf{e}_{ij} = LeakyReLU(\mathbf{a}^T [\mathbf{z}_i || \mathbf{z}_j])$
- Step 4: Use Softmax to normalize attention

$$\alpha_{ij} = softmax(\mathbf{e}_{ij}) = \frac{exp(\mathbf{e}_{ij})}{\sum_{k \in \tilde{\mathcal{N}_i}} exp(\mathbf{e}_{ik})}$$

GAT: computing attention head

Important equation

$$\mathbf{h}_i^{(l+1)} = \sum_{j \in \tilde{\mathcal{N}_i}} \alpha_{ij} \cdot \left(\mathbf{W}^\top \cdot \mathbf{h}_j^{(l)} \right) + \mathbf{b}$$

- Step 1: A learnable attention matrix converts the node embeddings into **new embeddings specific for Attention**

$$\mathbf{z_i} = \left(\mathbf{W}^\top \cdot \mathbf{h}_i \right) + \mathbf{b}$$

- Step 2: Concatenate both nodes embeddings $[\mathbf{z}_i || \mathbf{z}_j]$ and then compute an attention coefficient $\mathbf{a}_{ij}$ using learnable weights

$$\mathbf{e}_{ij} = \mathbf{a}^T [\mathbf{z}_i || \mathbf{z}_j]$$

- Step 3: Add an activation function $\mathbf{e}_{ij} = LeakyReLU(\mathbf{a}^T [\mathbf{z}_i || \mathbf{z}_j])$
- Step 4: Use Softmax to normalize attention

$$\alpha_{ij} = softmax(\mathbf{e}_{ij}) = \frac{exp(\mathbf{e}_{ij})}{\sum_{k \in \tilde{\mathcal{N}_i}} exp(\mathbf{e}_{ik})}$$

GAT: computing attention head

Important equation

$$\mathbf{h}_i^{(l+1)} = \sum_{j \in \tilde{\mathcal{N}_i}} \alpha_{ij} \cdot \left(\mathbf{W}^\top \cdot \mathbf{h}_j^{(l)} \right) + \mathbf{b}$$

- Step 1: A learnable attention matrix converts the node embeddings into **new embeddings specific for Attention**

$$\mathbf{z_i} = \left(\mathbf{W}^\top \cdot \mathbf{h}_i \right) + \mathbf{b}$$

- Step 2: Concatenate both nodes embeddings $[\mathbf{z}_i || \mathbf{z}_j]$ and then compute an attention coefficient $\mathbf{a}_{ij}$ using learnable weights

$$\mathbf{e}_{ij} = \mathbf{a}^T [\mathbf{z}_i || \mathbf{z}_j]$$

- Step 3: Add an activation function $\mathbf{e}_{ij} = LeakyReLU(\mathbf{a}^T [\mathbf{z}_i || \mathbf{z}_j])$
- Step 4: Use Softmax to normalize attention

$$\alpha_{ij} = softmax(\mathbf{e}_{ij}) = \frac{exp(\mathbf{e}_{ij})}{\sum_{k \in \tilde{\mathcal{N}_i}} exp(\mathbf{e}_{ik})}$$

GAT: computing attention head

Important equation

$$\mathbf{h}_i^{(l+1)} = \sum_{j \in \tilde{\mathcal{N}_i}} \alpha_{ij} \cdot \left(\mathbf{W}^\top \cdot \mathbf{h}_j^{(l)} \right) + \mathbf{b}$$

- Step 1: A learnable attention matrix converts the node embeddings into **new embeddings specific for Attention**

$$\mathbf{z_i} = \left(\mathbf{W}^\top \cdot \mathbf{h}_i \right) + \mathbf{b}$$

- Step 2: Concatenate both nodes embeddings $[\mathbf{z}_i || \mathbf{z}_j]$ and then compute an attention coefficient $\mathbf{a}_{ij}$ using learnable weights

$$\mathbf{e}_{ij} = \mathbf{a}^T [\mathbf{z}_i || \mathbf{z}_j]$$

- Step 3: Add an activation function $\mathbf{e}_{ij} = LeakyReLU(\mathbf{a}^T [\mathbf{z}_i || \mathbf{z}_j])$
- Step 4: Use Softmax to normalize attention

$$\alpha_{ij} = softmax(\mathbf{e}_{ij}) = \frac{exp(\mathbf{e}_{ij})}{\sum_{k \in \tilde{\mathcal{N}_i}} exp(\mathbf{e}_{ik})}$$

GAT: Multi-head attention

- A single attention layer might not be powerful enough. What we described is called an attention head, and we typically have multiple heads
- The resulting embeddings are then combined (i.e., Average, Concat)
- This is the similar principle as the multiple channels of a convolution

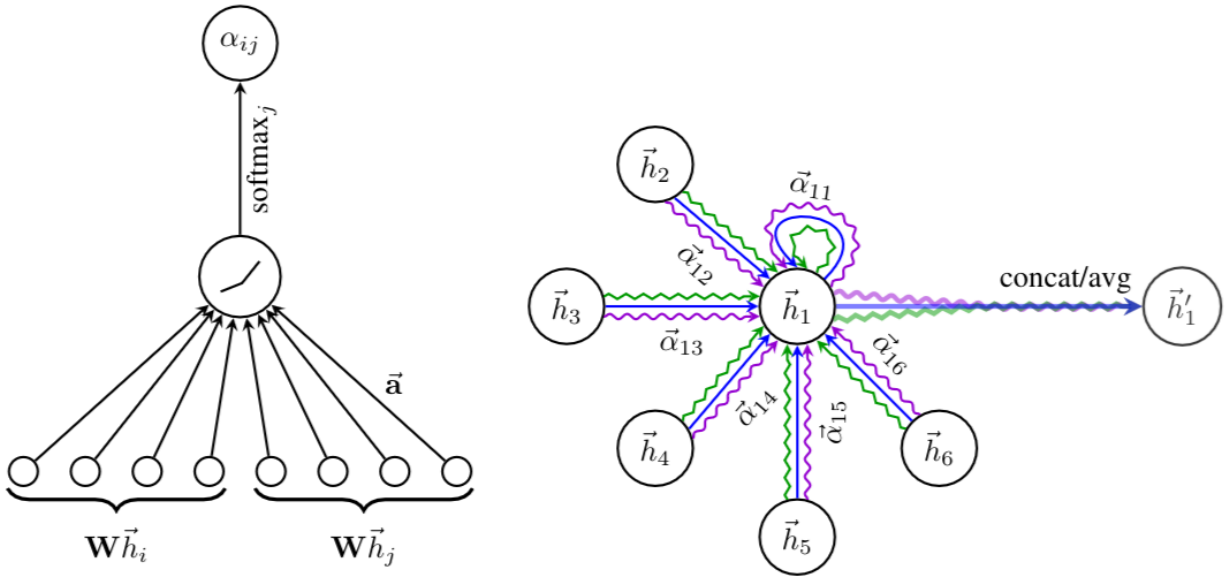


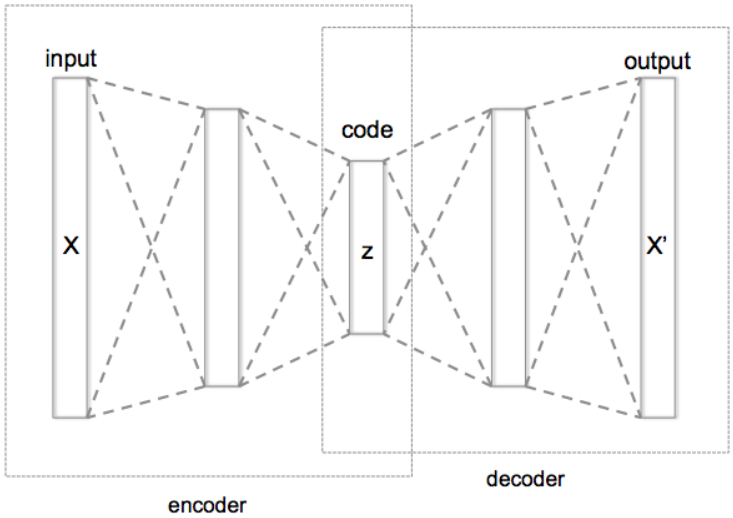
Figure 1: **Left:** The attention mechanism $a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ employed by our model, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **Right:** An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$.

Figure: Velikovi, Petar, et al. "Graph attention networks." arXiv preprint arXiv:1710.10903 (2017).

Variational Graph Auto-Encoder (VGAE)

Auto-Encoders

- Auto-encoders are mostly used for unsupervised learning
- Composed of two parts: An encoder and a decoder
- In the middle is the “embedding”, what we are interested in constrained to be small



The objective is to encode a complex object (e.g., 3 color layers, 256 x 256 image) into a low-dimensional vector (e.g., a vector of size 128), such that these vectors allow reproducing the output with minimal loss of information.

Applications: visualization (like PCA), downstream tasks, generate variations

Graph Auto-Encoders

Same principle, but with graphs :)

Classic architecture

- Encoder: GCN layers (e.g., 2 layers)
- Decoder: Dot product between embeddings
- Loss: Minimize the binary cross-entropy between input and output adjacency matrices

Computed vectors for each node such that their dot product is

- Close to 1 if they are connected (parallel $\Rightarrow$ similar vectors)
- Close to 0 if they are not (orthogonal $\Rightarrow$ different vectors)

Limits of Auto-encoders:

- Embedding space is often poorly structured
- Poor continuity: The “middle” vector between two vectors (v1,v2) does not correspond to a middle image between the two corresponding to v1/v2
- Poor completeness: Space seems “sparse”: many vectors correspond to nothing meaningful

Graph Auto-Encoders

Same principle, but with graphs :)

Classic architecture

- Encoder: GCN layers (e.g., 2 layers)
- Decoder: Dot product between embeddings
- Loss: Minimize the binary cross-entropy between input and output adjacency matrices

Computed vectors for each node such that their dot product is

- Close to 1 if they are connected (parallel $\Rightarrow$ similar vectors)
- Close to 0 if they are not (orthogonal $\Rightarrow$ different vectors)

Limits of Auto-encoders:

- Embedding space is often poorly structured
- Poor continuity: The “middle” vector between two vectors (v1,v2) does not correspond to a middle image between the two corresponding to v1/v2
- Poor completeness: Space seems “sparse”: many vectors correspond to nothing meaningful

Graph Auto-Encoders

Same principle, but with graphs :)

Classic architecture

- Encoder: GCN layers (e.g., 2 layers)
- Decoder: Dot product between embeddings
- Loss: Minimize the binary cross-entropy between input and output adjacency matrices

Computed vectors for each node such that their dot product is

- Close to 1 if they are connected (parallel $\Rightarrow$ similar vectors)
- Close to 0 if they are not (orthogonal $\Rightarrow$ different vectors)

Limits of Auto-encoders:

- Embedding space is often poorly structured
- Poor continuity: The “middle” vector between two vectors (v1,v2) does not correspond to a middle image between the two corresponding to v1/v2
- Poor completeness: Space seems “sparse”: many vectors correspond to nothing meaningful

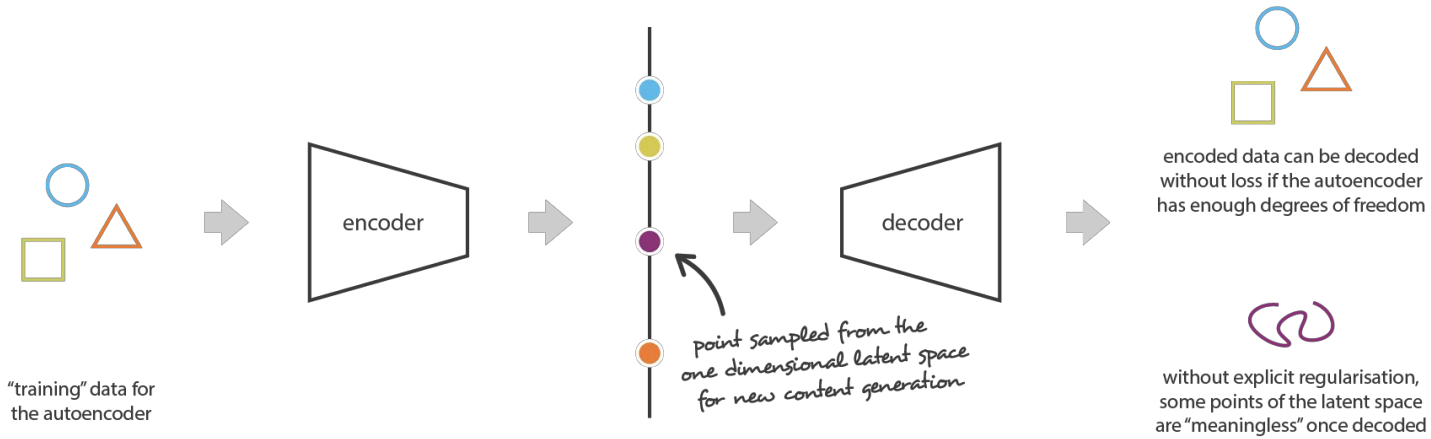
Popular improvement: Variational Graph Auto-Encoders (VGAE)

(see more)

Auto-encoders' limits $\Rightarrow$ VAE solution: Instead of encoding an input as a single point, we encode it as a distribution over the latent space

The model is trained as follows:

1. the input is encoded as a Gaussian distribution over the latent space
2. a point from the latent space is sampled from that distribution
3. the sampled point is decoded, and the reconstruction error can be computed
4. finally, the reconstruction error is backpropagated through the network



see more: understanding-variational-autoencoders-vaes

Popular improvement: Variational Graph Auto-Encoders (VGAE)

(see more)

Auto-encoders' limits $\Rightarrow$ VAE solution: Instead of encoding an input as a single point, we encode it as a distribution over the latent space

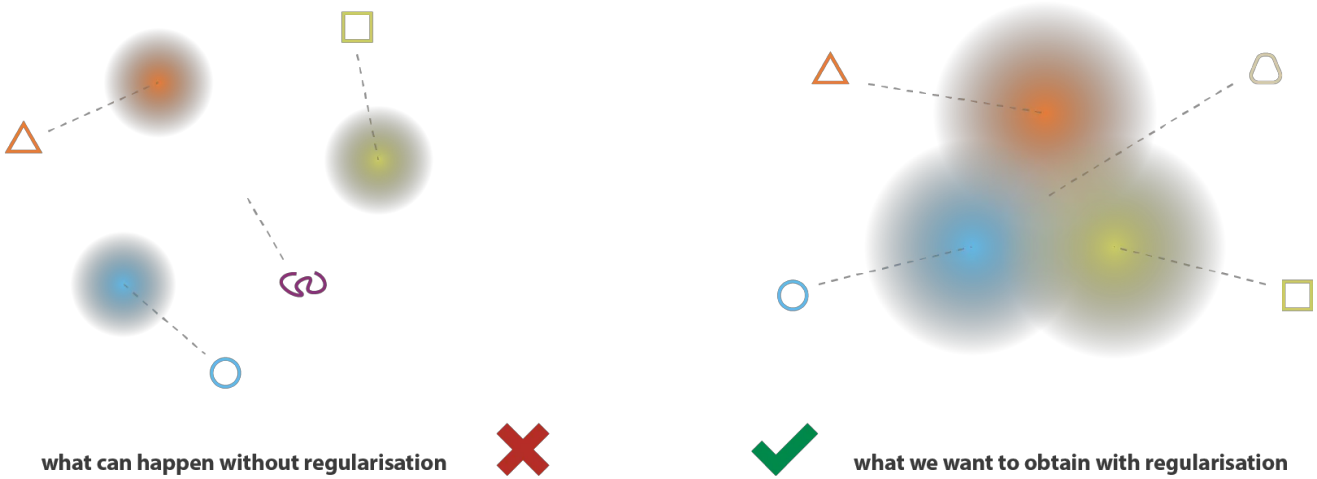
The model is trained as follows:

1. the input is encoded as a Gaussian distribution over the latent space
2. a point from the latent space is sampled from that distribution
3. the sampled point is decoded, and the reconstruction error can be computed
4. finally, the reconstruction error is backpropagated through the network

see more: understanding-variational-autoencoders-vaes

VAE regularization

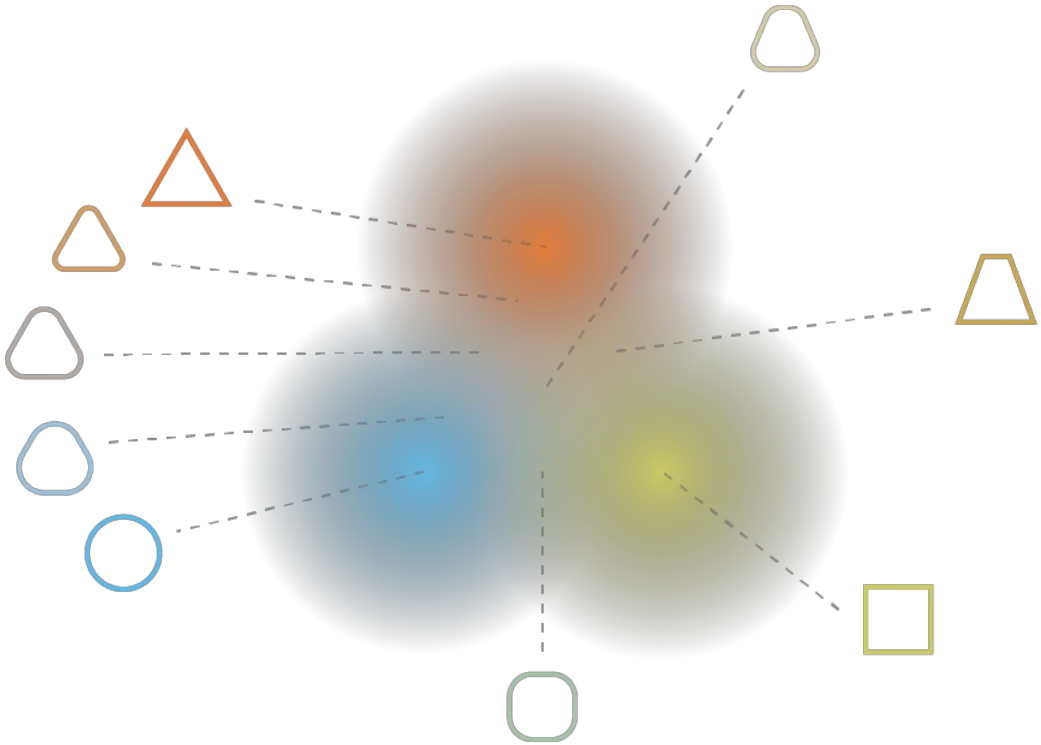
Regularization: trade-off between best fit to data and distance between each Gaussian and a standard Gaussian (centered, unit variance) with the KL divergence (see more)



see more: understanding-variational-autoencoders-vaes

VAE regularization

Regularization: trade-off between best fit to data and distance between each Gaussian and a standard Gaussian (centered, unit variance) with the KL divergence (see more)



see more: understanding-variational-autoencoders-vaes

VGAE in practice

Simple adaptation to graphs, i.e., a classic graph auto-encoder in which the encoding part is replaced by a variational mechanism.

Encoder in practice:

- Layer 1: normal GCN
- Layer 2: two parallel GCN layers
  - One to learn the centroid
  - One to learn the variance (diagonal of the covariance matrix)

$\Rightarrow$ For each node, instead of having 1 vector of size d, we have two vectors of size d.
To decode, we take a random point from the multivariate Gaussian.

Second exercise !!

Basic tasks on graphs

Basic tasks on graphs

As we said, there are plenty of tasks on graphs that we can resolve with GNNs:

- ...

Basic tasks on graphs

As we said, there are plenty of tasks on graphs that we can resolve with GNNs:

- Link prediction (follow recommendation, drug/illness relationship)
- Node classification (bot detection in social media)
- Attribute regression (age of individuals in social media)
- Link classification/regression (relationship is: friend/colleague/lover ?)
- Graph classification (Molecule classification)
- Community detection
- ...

Link prediction

(see more)

On the Observed network (current state), we intend to predict links that might have been missed: **missing link prediction**, or that might appear in the future: **future link prediction**

Predicting links is a classification objective:

- binary classes: edge or no edge
- Usually evaluation is based on class probability (AUC, AP, Hits@k, MRR)
- The evaluation process:
  1. Hide some edges in the graphs (positive links test set)
  2. Sample some disconnected (i.e., without a link) pairs of nodes (negative links test set)
  3. Training on the remaining edges
  4. Predict well on the test set !!

Classic methods ⇒ Work only on nodes at distance two

- Common Neighbors
- Adamic Adar
- PageRank
- ...

Advanced methods ⇒ Do not take node features into account

- Graph embeddings (DeepWalk, Node2Vec)
- Community structure, random walks
- ...

Link prediction

(see more)

On the Observed network (current state), we intend to predict links that might have been missed: **missing link prediction**, or that might appear in the future: **future link prediction**

Predicting links is a classification objective:

- binary classes: edge or no edge
- Usually evaluation is based on class probability (AUC, AP, Hits@k, MRR)
- The evaluation process:
  1. Hide some edges in the graphs (positive links test set)
  2. Sample some disconnected (i.e., without a link) pairs of nodes (negative links test set)
  3. Training on the remaining edges
  4. Predict well on the test set !!

Classic methods $\Rightarrow$ Work only on nodes at distance two

- Common Neighbors
- Adamic Adar
- PageRank
- ...

Advanced methods $\Rightarrow$ Do not take node features into account

- Graph embeddings (DeepWalk, Node2Vec)
- Community structure, random walks
- ...

Link prediction

(see more)

On the Observed network (current state), we intend to predict links that might have been missed: **missing link prediction**, or that might appear in the future: **future link prediction**

Predicting links is a classification objective:

- binary classes: edge or no edge
- Usually evaluation is based on class probability (AUC, AP, Hits@k, MRR)
- The evaluation process:
  1. Hide some edges in the graphs (positive links test set)
  2. Sample some disconnected (i.e., without a link) pairs of nodes (negative links test set)
  3. Training on the remaining edges
  4. Predict well on the test set !!

Classic methods $\Rightarrow$ Work only on nodes at distance two

- Common Neighbors
- Adamic Adar
- PageRank
- ...

Advanced methods $\Rightarrow$ Do not take node features into account

- Graph embeddings (DeepWalk, Node2Vec)
- Community structure, random walks
- ...

Link prediction with GNNs

Using VGAE: by reconstructing the graphs, we predict whether an edge is present or not $\Rightarrow$ directly a link prediction objective.

Using a GNN directly: GNNs produce node embeddings that we can use to predict links…

- …directly with the dot product (like a GAE)
- …through a link-predictor head (e.g., MLP) that takes combined embeddings of node pairs (Hadamard product).
- …More recent methods, that mix heuristic-base and GNN like Neural Common Neighbot with Completion

Nodes classification and graphs classification

- **Nodes classification**
  - Often Semi-supervised learning with only a set of node which have labels. We can also predict an attribute of nodes.
  - With an MLP or a softmax layer as the output layer, any GNNs are able to perform node-level tasks in an end-to-end manner.
  - In self-supervised, contrastive or non-contrastive learning methods as Bootstrapped Graph Latents
- **Graphs classification**
  - To obtain a compact representation on the graph level, GNNs are often combined with pooling and readout operations.

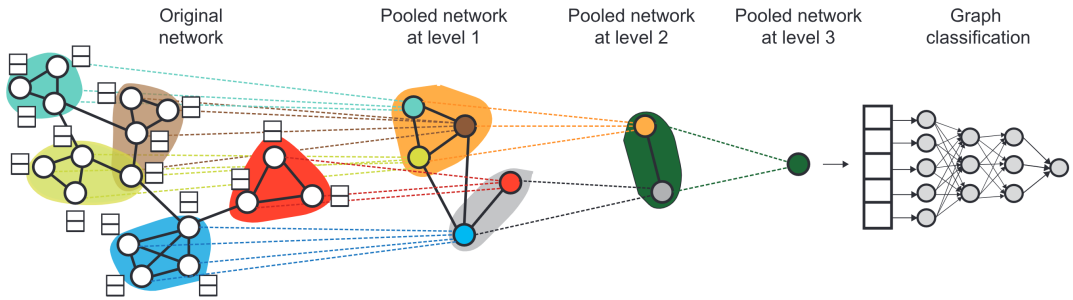


Figure 1: High-level illustration of our proposed method DIFFPOOL. At each hierarchical layer, we run a GNN model to obtain embeddings of nodes. We then use these learned embeddings to cluster nodes together and run another GNN layer on this coarsened graph. This whole process is repeated for L layers and we use the final output representation to classify the graph.

Figure: Ying, Zhitao, et al. "Hierarchical graph representation learning with differentiable pooling." Advances in neural information processing systems 31 (2018).

Graph anomaly detection

- detecting anomalous nodes (bots)
- detecting anomalous graphs among set of graphs
- detecting anomalous structure / link / substructure (e.g., cybersecurity)

A complet survey: Ma, Xiaoxiao, et al. "A comprehensive survey on graph anomaly detection with deep learning." IEEE transactions on knowledge and data engineering 35.12 (2021): 12012-12038.

A simple popular methods:

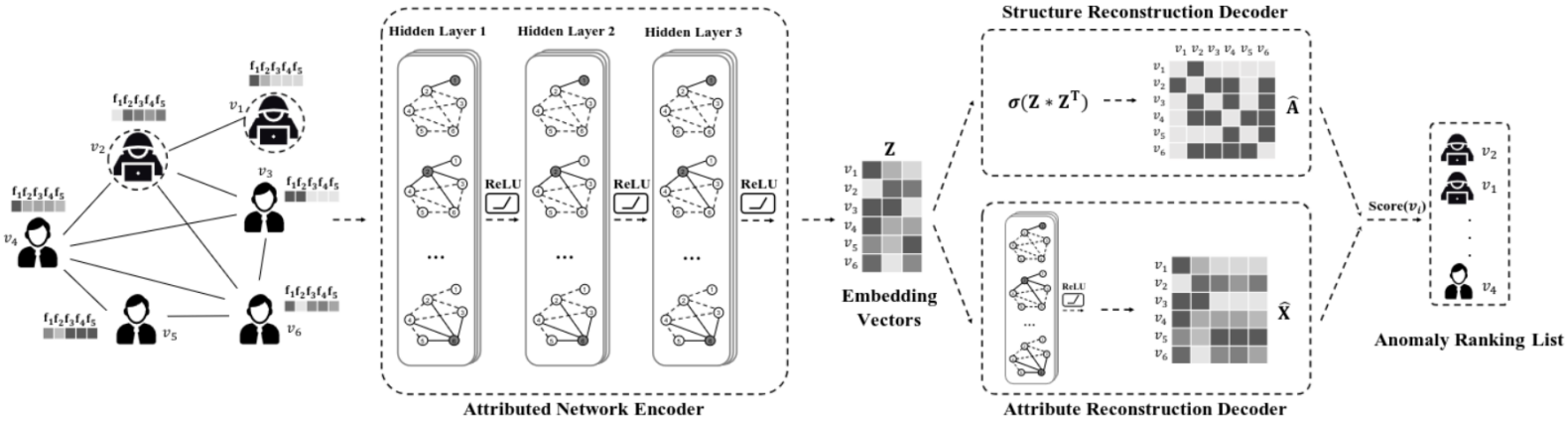


Figure: Ding, Kaize, et al. "Deep anomaly detection on attributed networks." Proceedings of the 2019 SIAM international conference on data mining. Society for Industrial and Applied Mathematics, 2019.

Different graphs, different context

Transductive / Inductive

- **Transductive learning**: the model is trained and tested on a unique graph fixed (what we discussed until now).
- **Inductive learning**: the model learn to generalize to unseen nodes:
  - Train on a set of nodes/graphs
  - A GCN layer can be trained on multiple (sub)networks, and learned weights used on a new scenario (but not very efficient)
  - GraphSAGE $\Rightarrow$ Works for each node on a local graph centered on the node, by sampling a fixed number of neighbors. Transform the graph problem in a more classic problem (see more)

Dynamic Graphs (see more)

- When different graphs are correlated in time, we call them dynamic graphs (or temporal graphs)
- We can model dynamic graphs either…
  - … discrete-time dynamic graphs (DTDG): several snapshots taken at regular time intervals.
  - … continuous-time dynamic graphs (CTDG): a collection of events (node participating, event type, timestamp)
- Several methods of dynamic GNNs use both Recurrent Neural Networks (RNNs) and GNN layers in their architecture.
- Research tends to develop more dynamic GNN methods on CTDG since we don't lose information like DTDG assumes.

Different graphs, different context

Multi-Partite Graphs

- Nodes of multiple types: Item/Users, Drug/illness
- Each type of node has their own attributes $\Rightarrow$ Cannot learn a single GCN layer $\Rightarrow$ Learn 2 independent layers
  - User attributes to Item attributes
  - Item attributes to User attributes

Knowledge Graphs

- Knowledge Graphs are labeled directed multi-graphs.
- Tasks: entity classification model which predict missing attributes, link prediction that scores a triplet (subject, relation, object).
- Relational Graph Convolutional Networks (R-GCN) is an extension of GCN on Knowledge Graphs: it learns different weights for each type of relation (see more).

In practice

Libraries and code documentations

We are going to code in Python with the main library torch geometric based on PyTorch, which you can install here, and with the documentation here.
Moreover, for complex networks, we can use networkx.

I really recommend that you set up a virtual environment for your Python packages, such as venv or Miniconda…

Before the next session, you can take a look at the first tutorial for not being lost :)