business innovation through algorithms

Graph Machine Learning

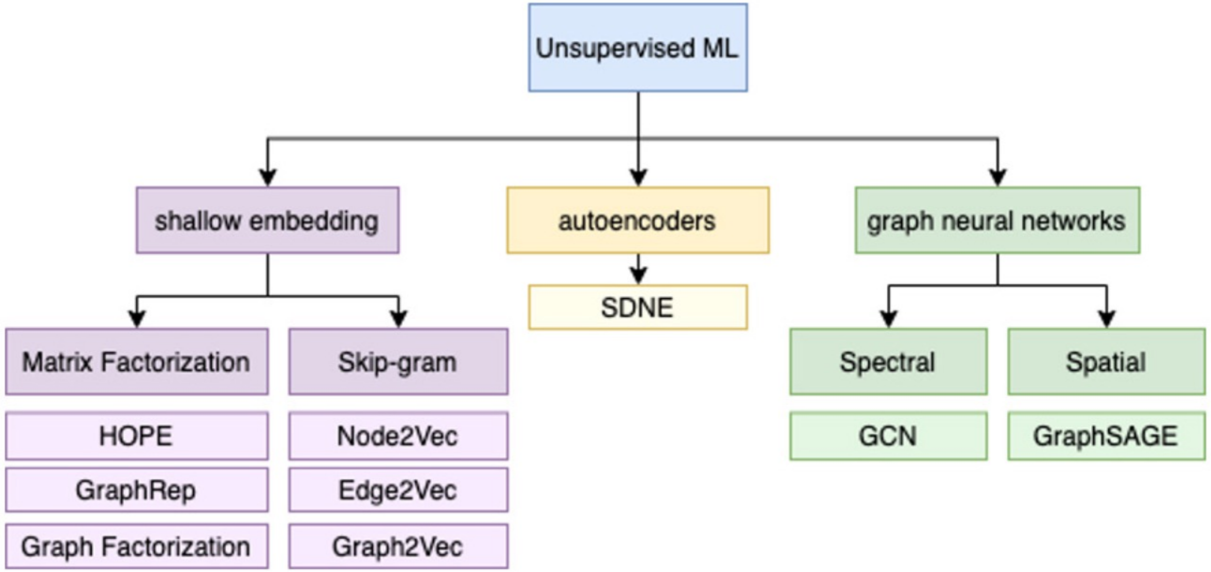
CGnal S.r.l – Corso Venezia 43 - Milano

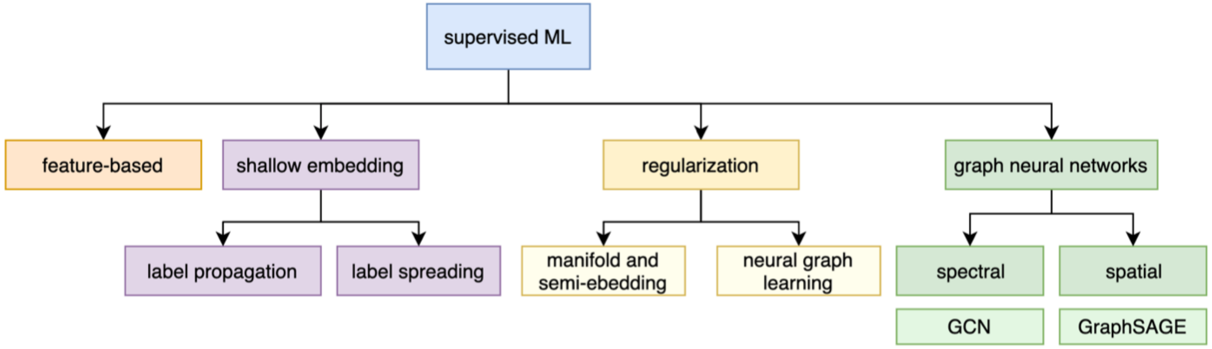
26 Novembre 2021| Milano

Overview on Machine Learning with Graphs

Machine Learning on Graphs: A Model and Comprehensive Taxonomy

Ines Chami[*,†], Sami Abu-El-Haija[‡], Bryan Perozzi[††], Christopher Ré[‡‡], and Kevin Murphy[††]

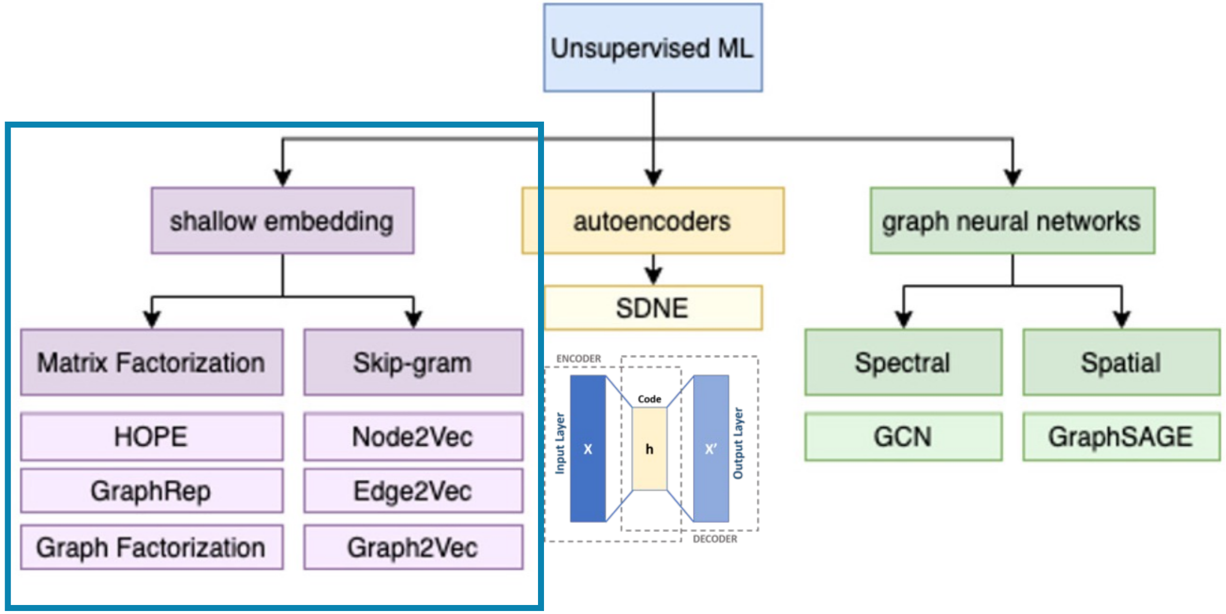


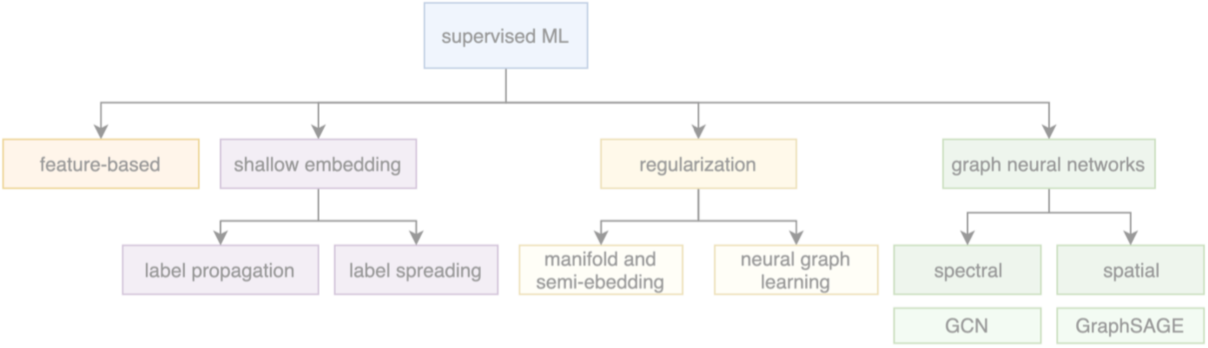


Overview on Machine Learning with Graphs

Machine Learning on Graphs:
A Model and Comprehensive Taxonomy

Ines Chami[*†], Sami Abu-El-Haija[‡], Bryan Perozzi[††], Christopher Ré[‡‡], and Kevin Murphy[††]

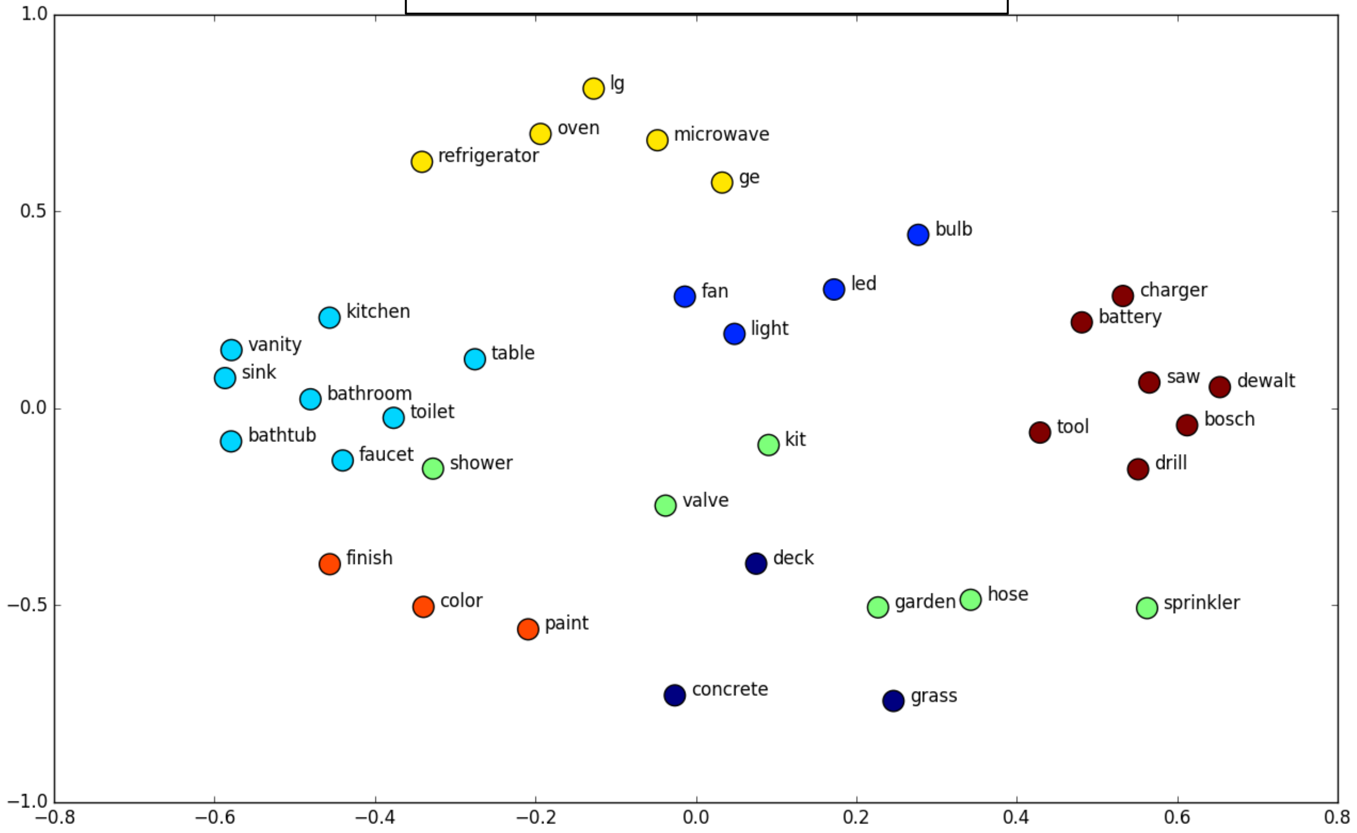


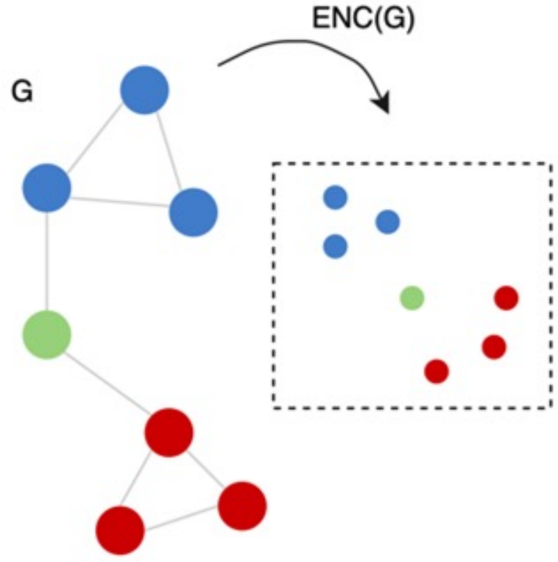


The generalized graph embedding problem – Representation Learning

Representation learning (**network embedding**) is the task that aims to learn a mapping function $f: G \rightarrow \mathbb{R}^d$ from a discrete graph to a continuous domain. Function will be capable of performing a low-dimensional vector representation such that the **properties (local and global) of graph are preserved**.

Word embedding example

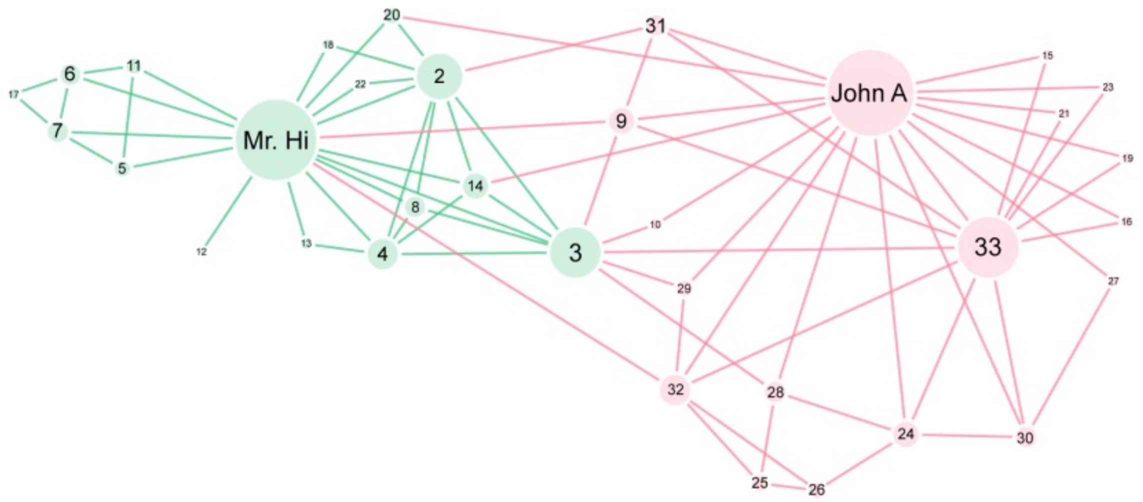




Node-level embedding

Shallow Embedding Methods

- They can only return a vectorial representation of the data they learned during the *fit_transform* procedure.
- **They are not able to generalize the function to unseen graphs (QUESTION: inductive or transductive?)**



node2Vect

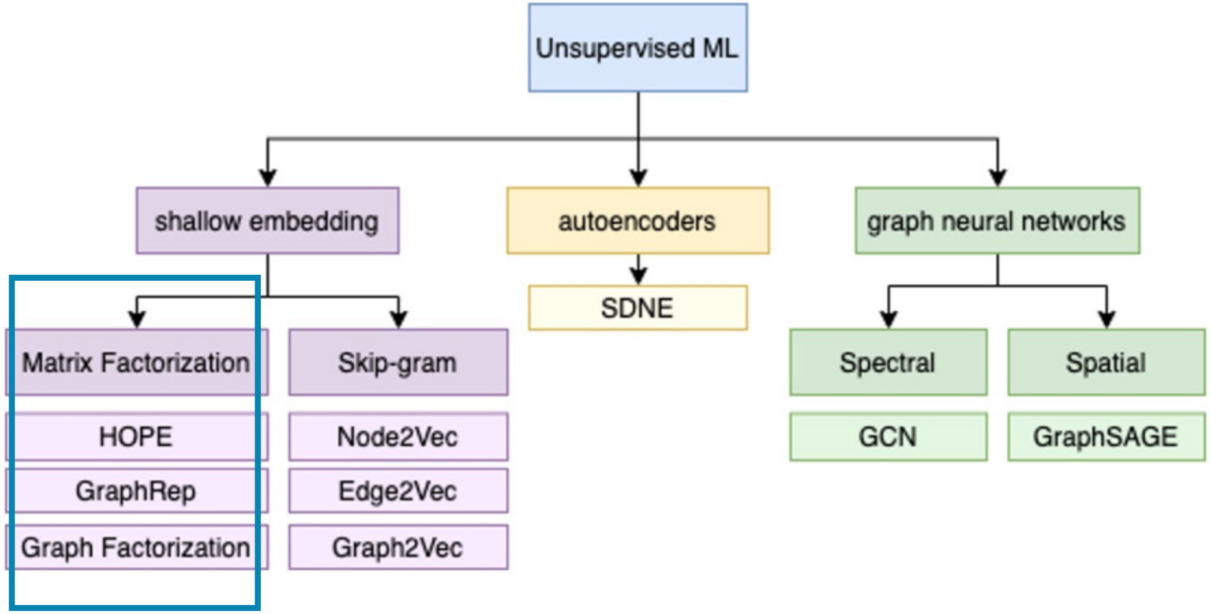
embedding =

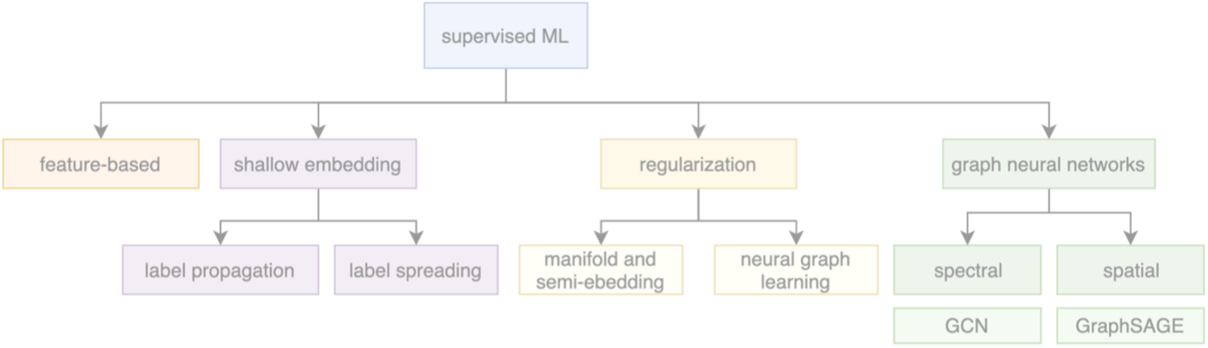
GraphId	Dim 1	Dim 2	…	Dim d
Mr. Hi	0.9	0.12	…	0.32
Mr. John	0.20	0.88	…	0.11
Node 1	0.25	0.76	…	0.09

Overview on Machine Learning with Graphs

Machine Learning on Graphs: A Model and Comprehensive Taxonomy

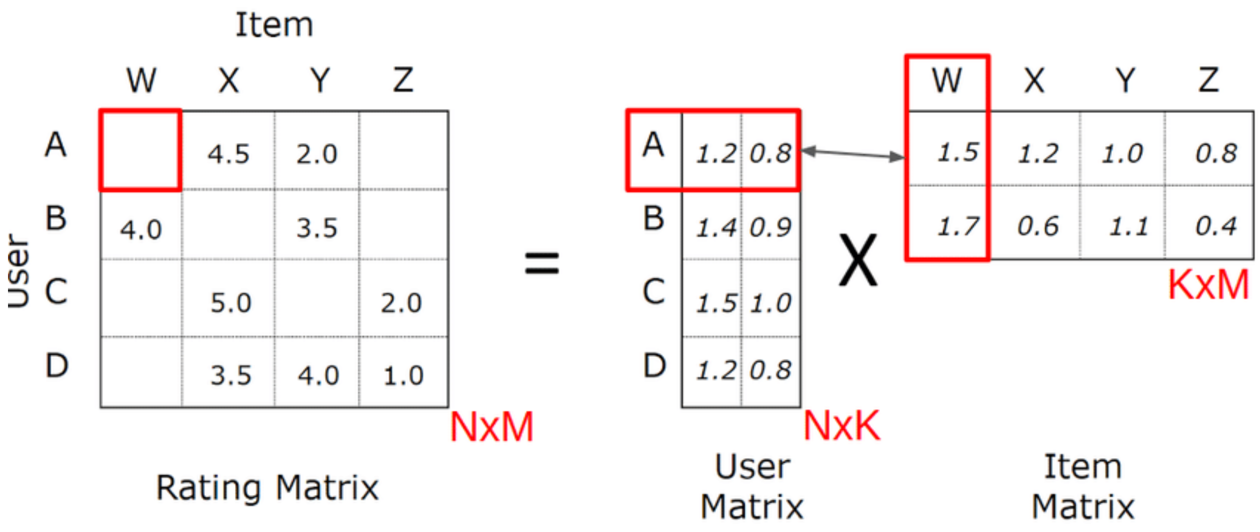
Ines Chami*†, Sami Abu-El-Haija‡, Bryan Perozzi††, Christopher Ré‡‡, and Kevin Murphy††





Matrix Factorization

$$L = \underbrace{\frac{1}{2}\sum_{(i,j)\in E}\left(A_{i,j} - Y_{i,:}Y_{j,:}{}^{T}\right)^2}_{\text{Reconstruction error}} + \underbrace{\frac{\lambda}{2}\sum_{i}\left\|Y_{i,:}\right\|^2}_{\text{Regularization term}}$$



Vanilla Version (Matrix Factorization)

- Reconstruction on the Adjacency Matrix only cares about first-order proximity, direct links and not quite topology
- The matrix decomposition using a single matrix $Y_{i,k}$ is only valid for symmetric matrices

Extend to Higher-Order Proximity (Neighbours of neighbours similarity)

Instead of A_{ij} we use other matrices, such as the similarity matrix $S = M_g^{-1} \cdot M_l$ or multiple step transition matrix $T_k = \prod_k D^{-1} \cdot A$

Extend to Non-Symmetric cases

Instead of symmetric decomposition use more general ones, like NMF or asymmetric decompositions:

$$L = \left\|S - Y_s \times Y_t{}^T\right\|_F^2$$

Matrix Factorization (Overview of algorithms)

Matrix Factorization

$$\left(A_{i,j} - Y_{i,:} Y_{j,:}{}^{T}\right)$$

- Only first-order proximity
- Only valid for symmetric matrices

HOPE

$$L = \left\| S - Y_s \times Y_t{}^{T} \right\|_F^2$$

$$S = M_g \cdot M_l$$

- Several kind of proximities can be integrated (Adamic-Adar, Katz Index, Rooted Page Rank and Common Neighbors
- Valid for directed graphs

GraphRep

$$L_k = \left\| X^k - Y_s^k \times Y_t^{k\,T} \right\|_F^2$$

$$X^k = \prod_k (D^{-1}A)$$

- Decide the order of the proximity, by tuning k
- Valid for directed graphs

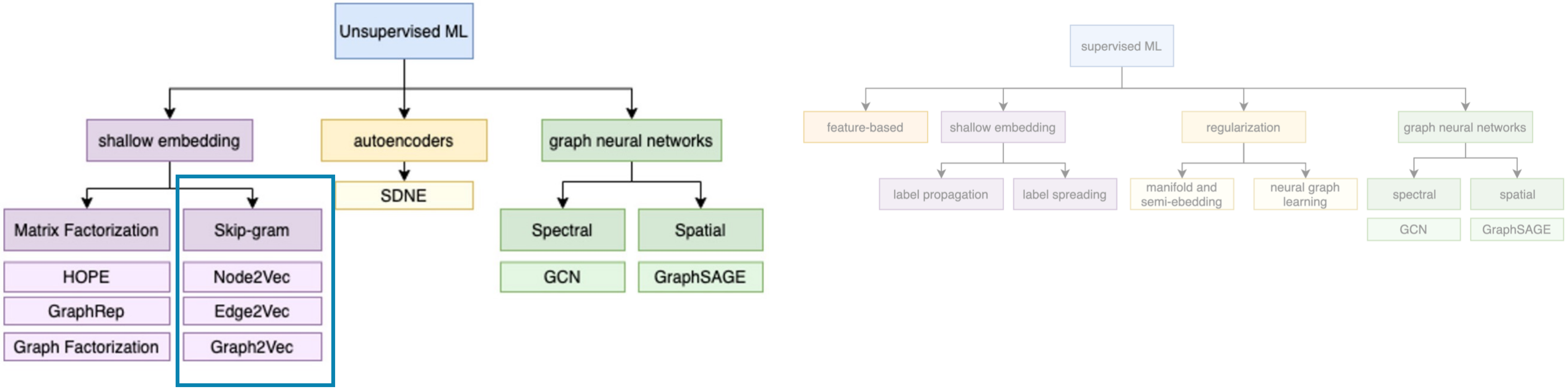
Many Algorihtms, One Library

GEM: https://github.com/palash1992/GEM

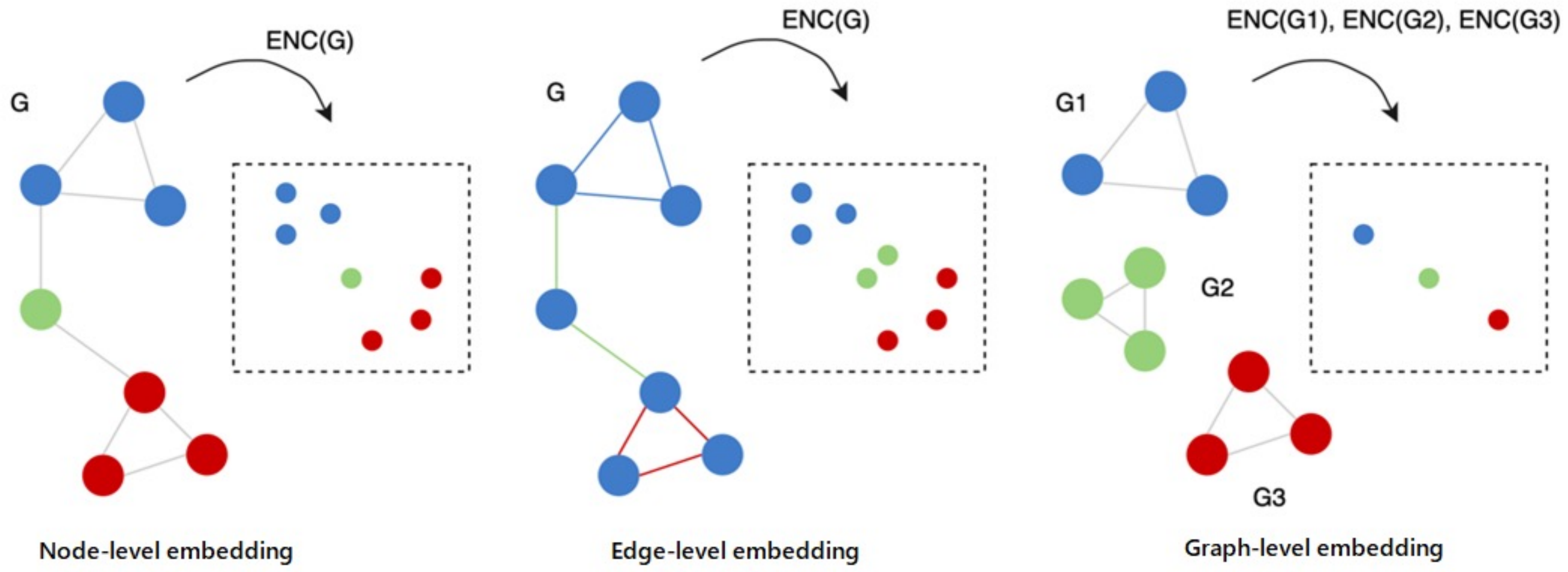
Overview on Machine Learning with Graphs

Machine Learning on Graphs: A Model and Comprehensive Taxonomy

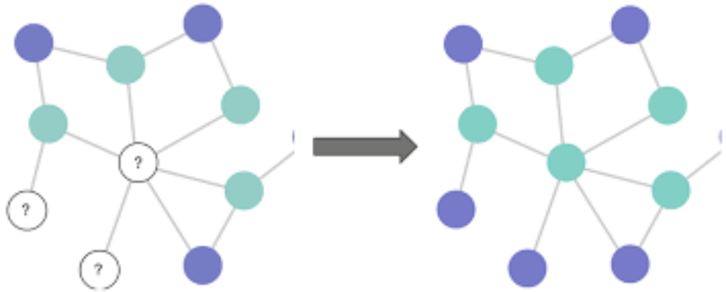
Ines Chami[*,†], Sami Abu-El-Haija[‡], Bryan Perozzi[††], Christopher Ré[‡‡], and Kevin Murphy[††]



node2Vect – edge2Vect – graph2Vect overview

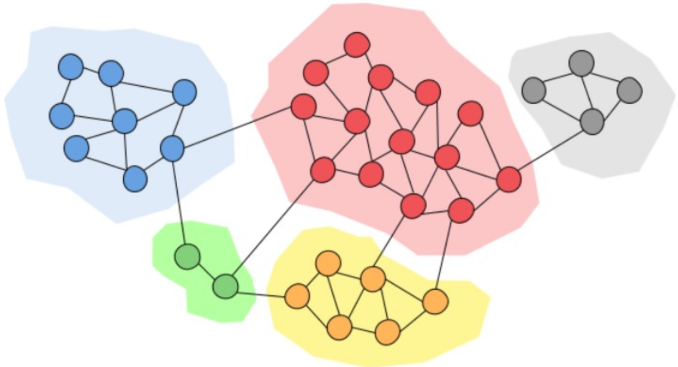


What can I do with embeddings?

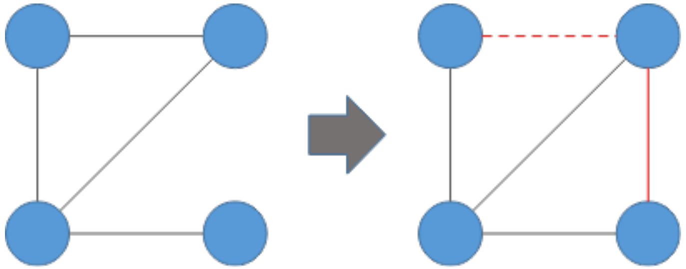


Node classification
(Using Node Embedding)

Fraud detection

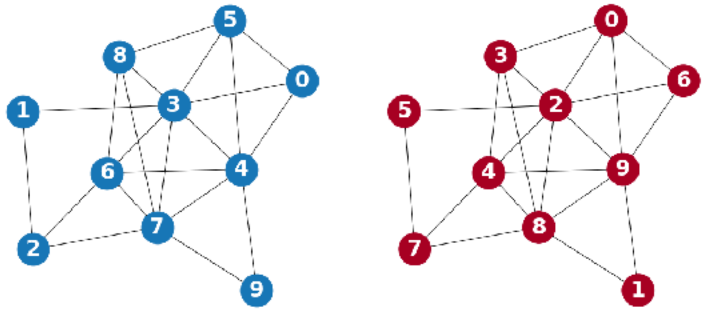


Community detection
(Using Node Embedding)



Missing link prediction
(Using Edge Embedding)

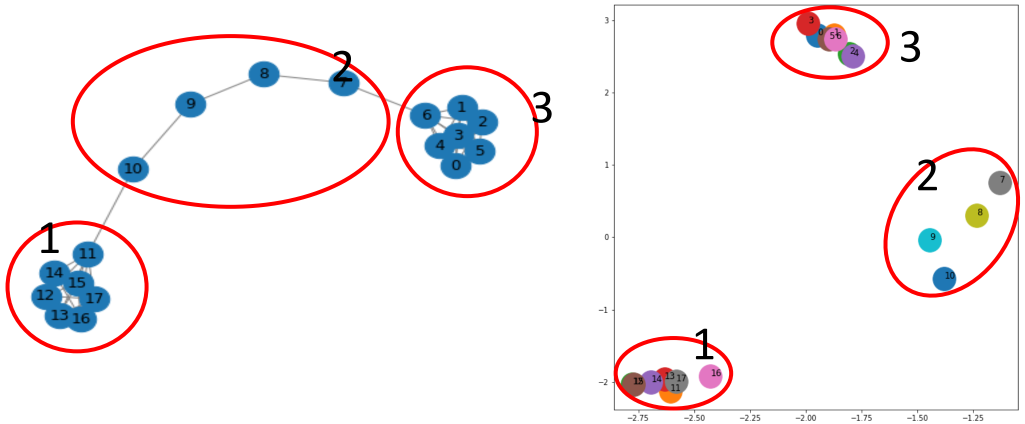
Reccomandation systems



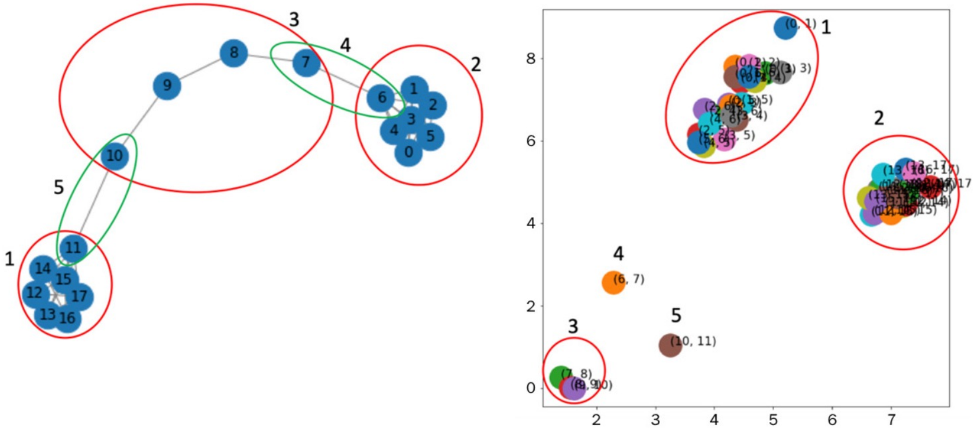
Graph similarity/clustering
(Using Graph Embedding)

Skip-gram based embedding algorithms– Node2Vec, Edge2Vec, Graph2Vec

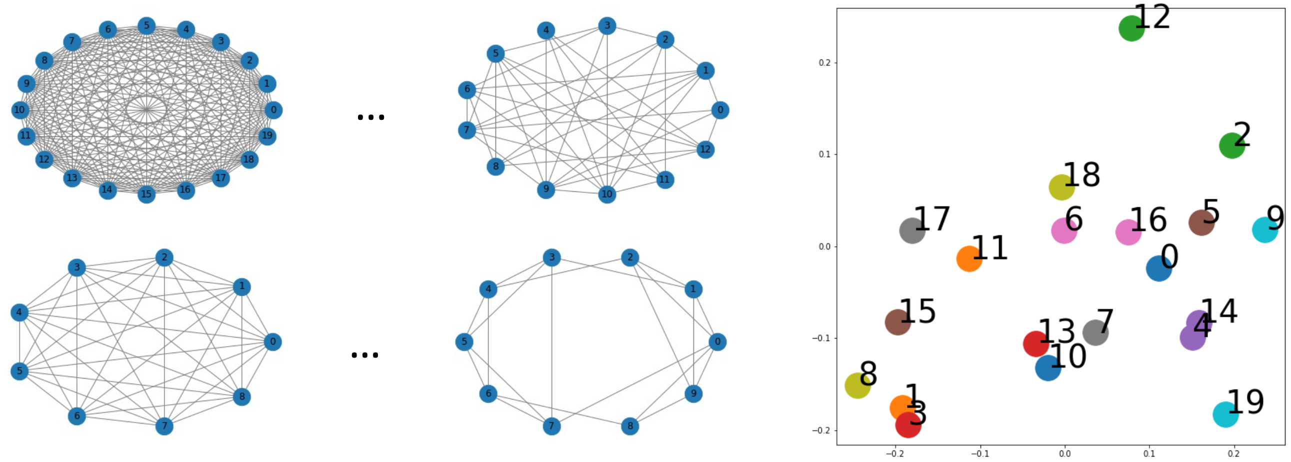
Node2Vec



Edge2Vec

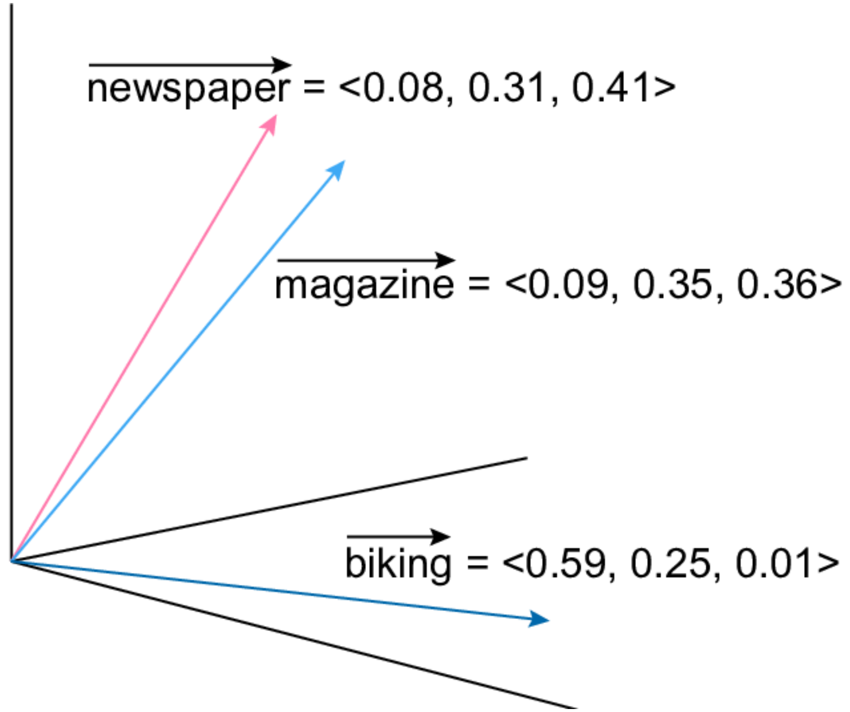


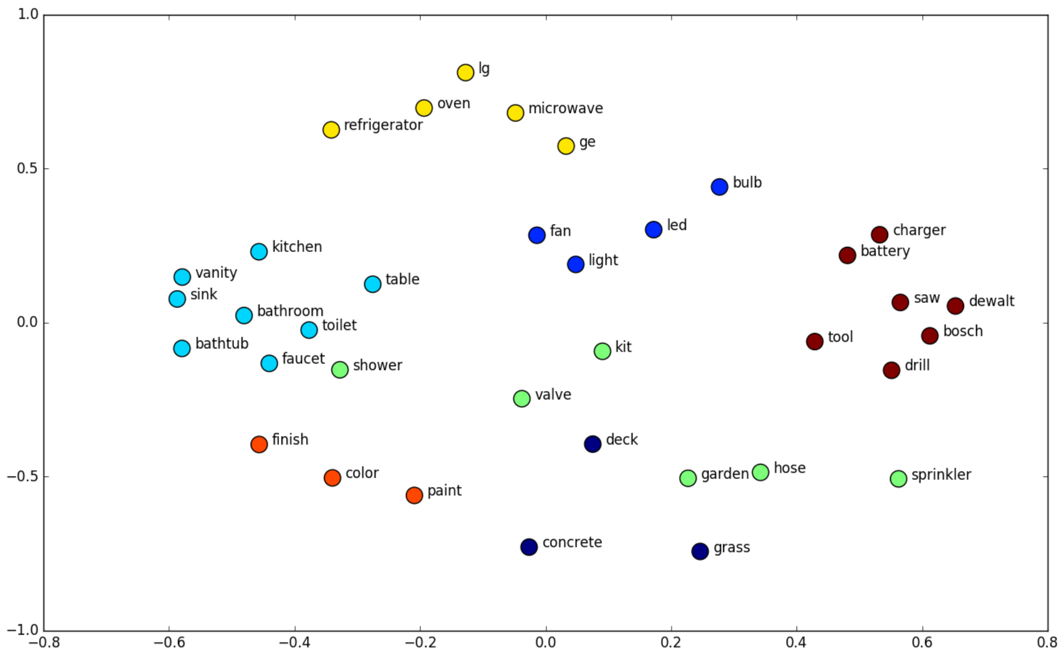
Graph2Vec



Skip-gram based embedding algorithms

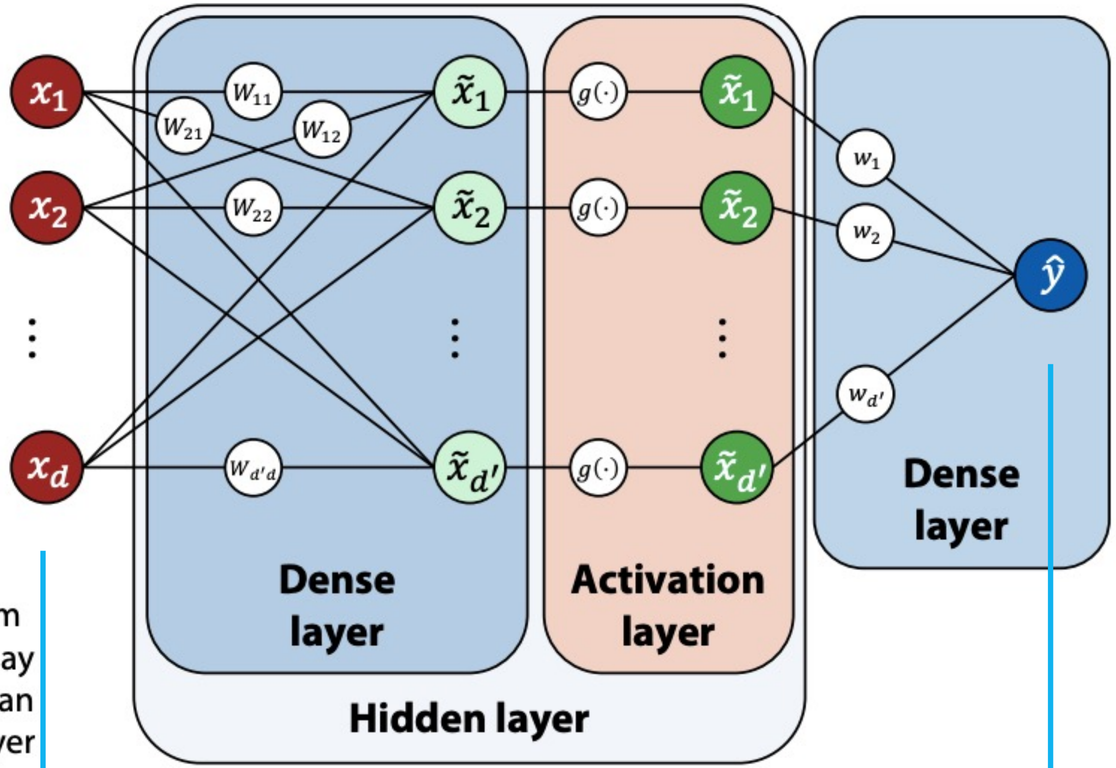
- Skip-gram architecture was created for «word embedding» problem also known as Word2Vec
- For a given word we want to build a vectorial representation in a *d* dimensional space





- Skip-gram Node2Vec use the same logic BUT instead of **words** we have **nodes**

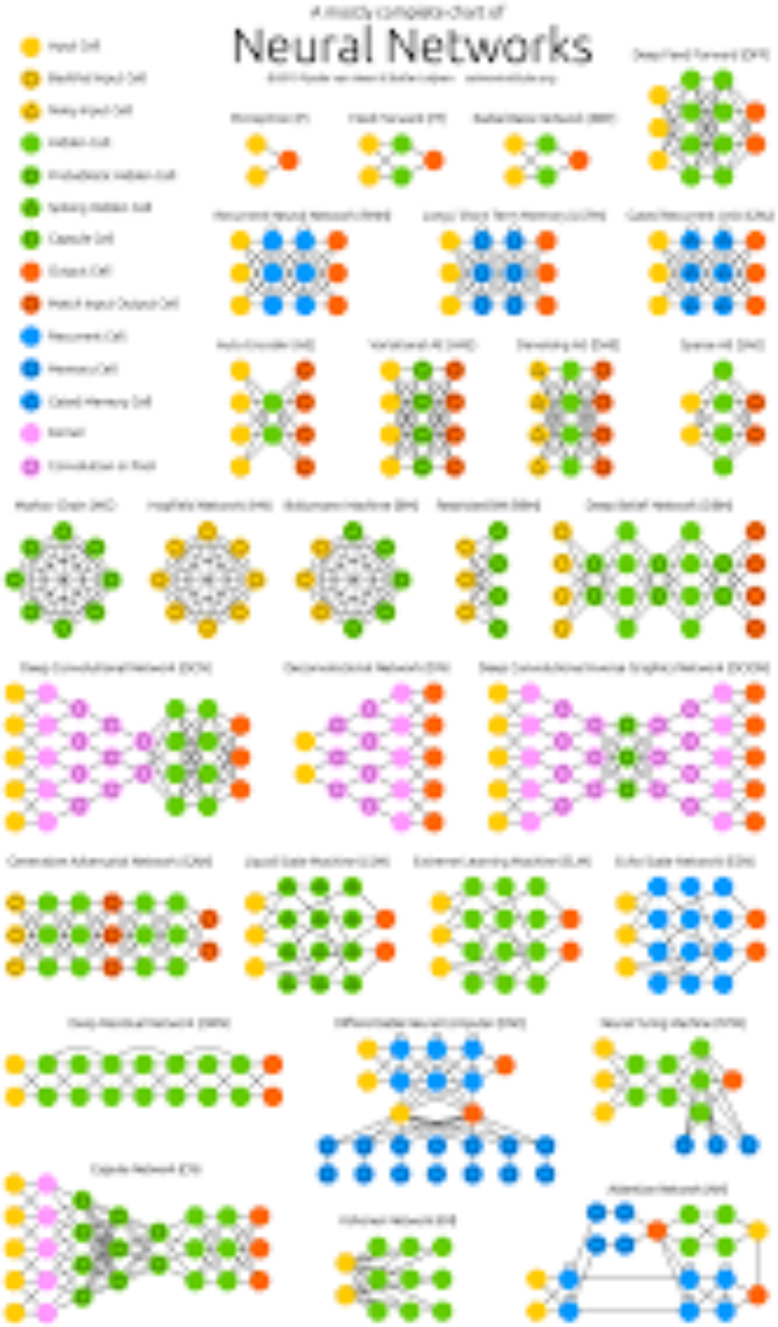
Neural Network basic idea



Note: the term "activation" may also stand for an output of a layer

INPUTS

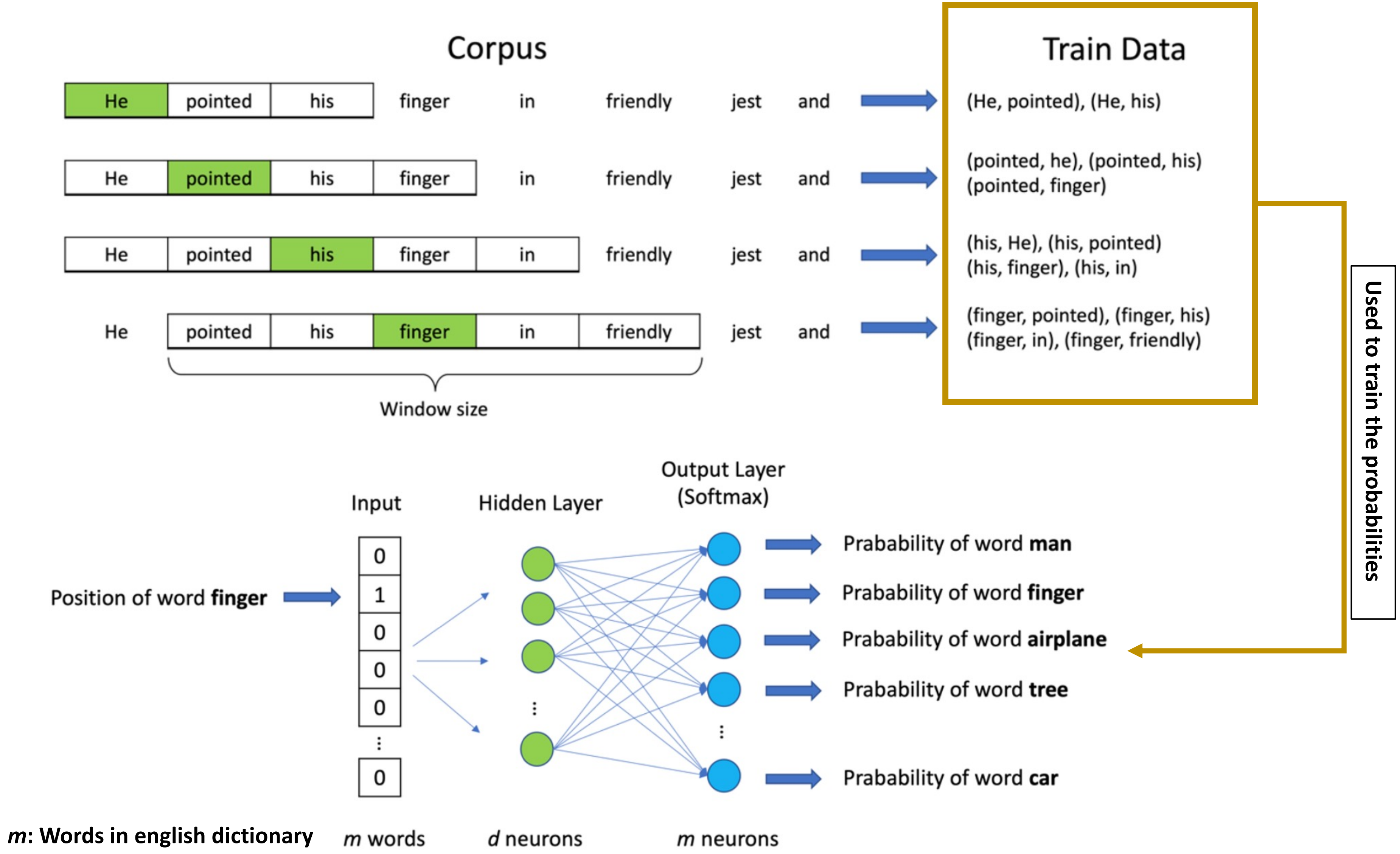
OUTPUTS



Skip-gram based embedding algorithms

Step 1. Dataset Generation

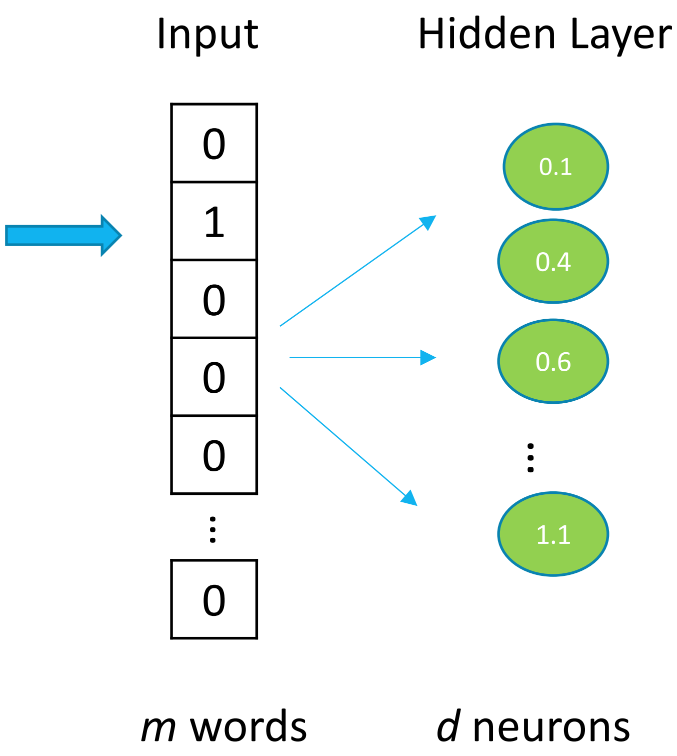
Step 2. Model Training



Skip-gram based embedding algorithms

Step 3. Embedding Extraction

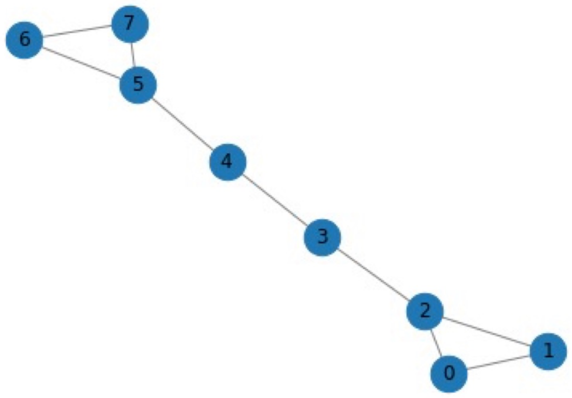
Position of word **finger**



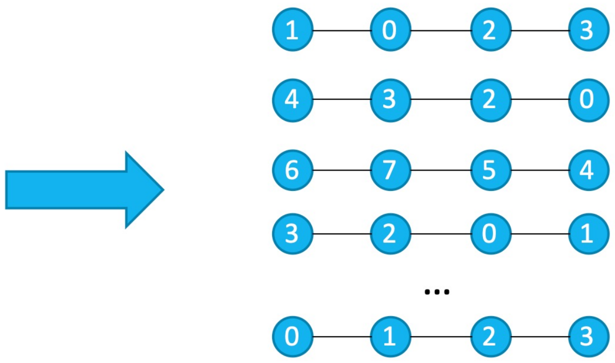
Word	Dim 1	Dim 2	Dim 3	…	Dim d
Finger	0.1	0.4	0.6		1.1

Skip-gram based embedding algorithms – Node2Vec

Step 0. Random Walk Generator



Input Graph

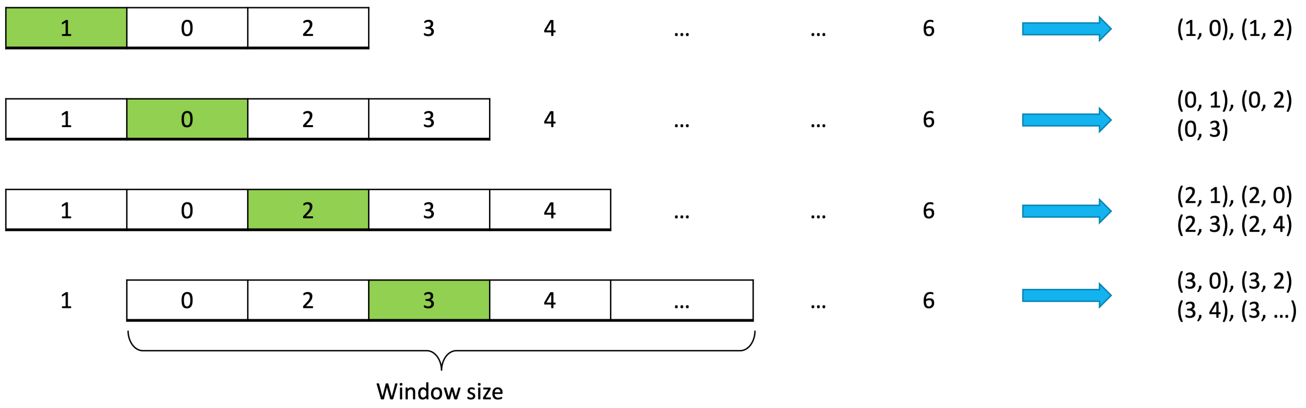


Random Walk Generation

Step 1. Dataset Generation

Corpus

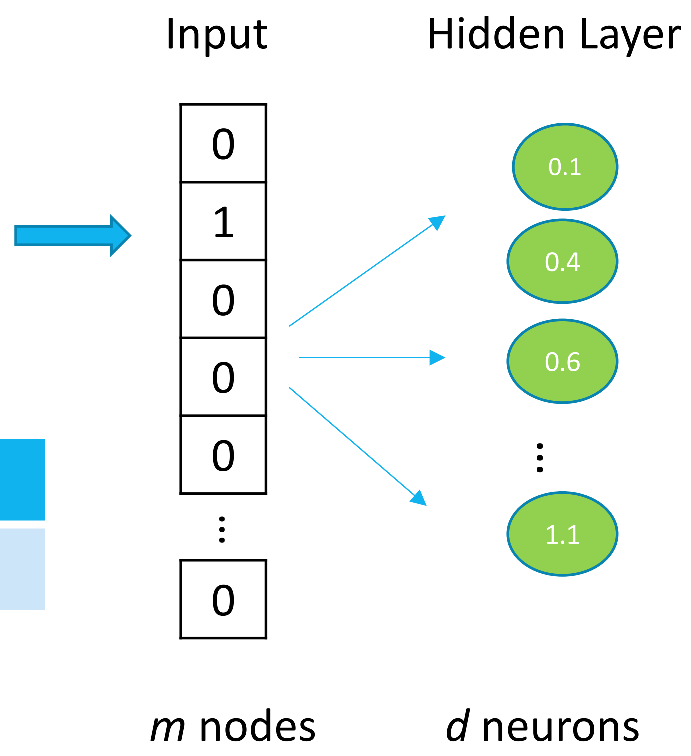
Train Data



Skip-gram based embedding algorithms – Node2Vec

Step 3. Embedding Extraction

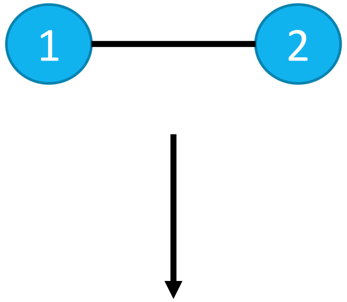
Position of Node **2**



NodeId	Dim 1	Dim 2	Dim 3	…	Dim d
Node2	0.1	0.4	0.6		1.1

```
import networkx as nx
from node2vec import Node2Vec
node2vec = Node2Vec(G, dimensions=d)
model = node2vec.fit(window=10)
embeddings = model.wv
```

Edge2Vec



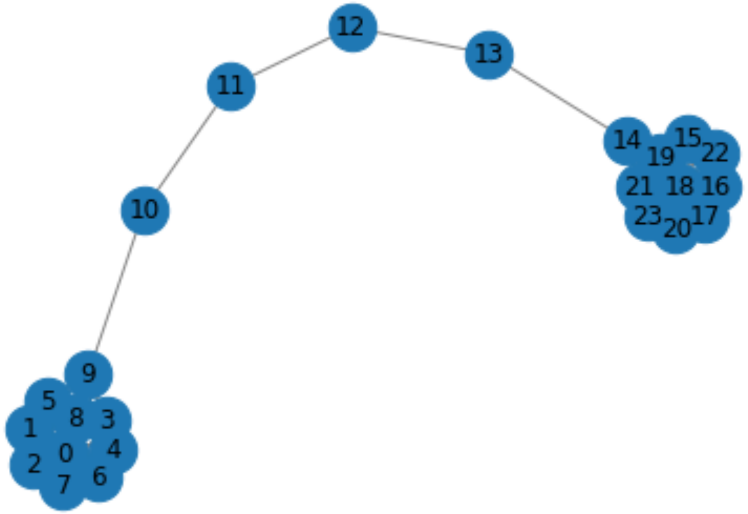
E(Embedding(Node 1), Embedding(Node 2))

Operator	Symbol	Definition
Average	$\boxplus$	$[f(u) \boxplus f(v)]_i = \frac{f_i(u)+f_i(v)}{2}$
Hadamard	$\boxdot$	$[f(u) \boxdot f(v)]_i = f_i(u) * f_i(v)$
Weighted-L1	$\|\cdot\|_{\bar{1}}$	$\|f(u) \cdot f(v)\|_{\bar{1}i} = \lvert f_i(u) - f_i(v) \rvert$
Weighted-L2	$\|\cdot\|_{\bar{2}}$	$\|f(u) \cdot f(v)\|_{\bar{2}i} = \lvert f_i(u) - f_i(v) \rvert^2$

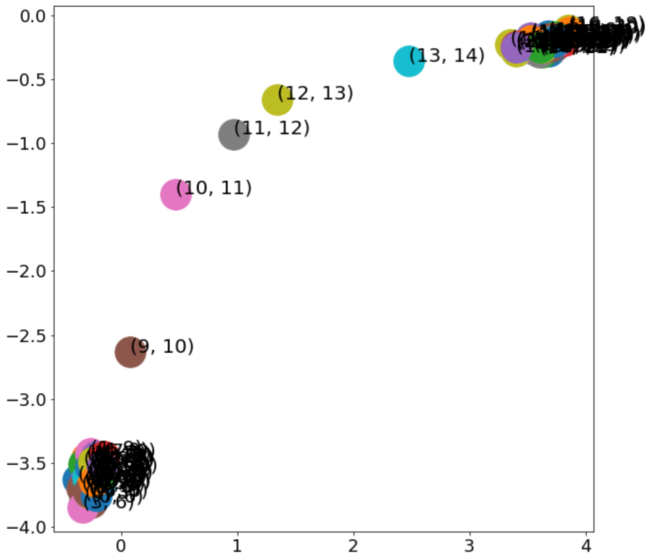
```
import networkx as nx
from node2vec import Node2Vec
node2vec = Node2Vec(G, dimensions=d)
model = node2vec.fit(window=10)
embeddings = model.wv

from node2vec.edges import HadamardEmbedder
embedding =
HadamardEmbedder(keyed_vectors=model.wv)
```

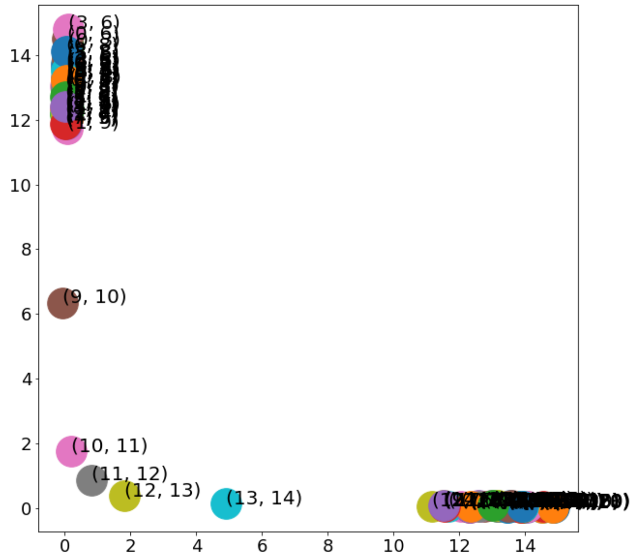
Edge2Vec



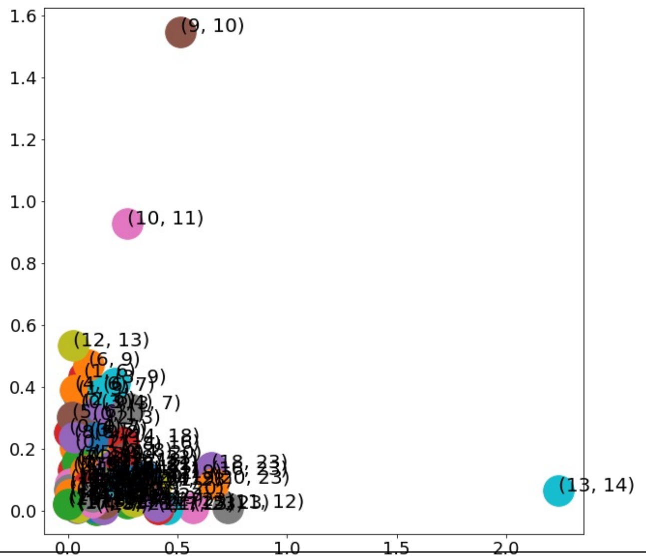
AverageEmbedder



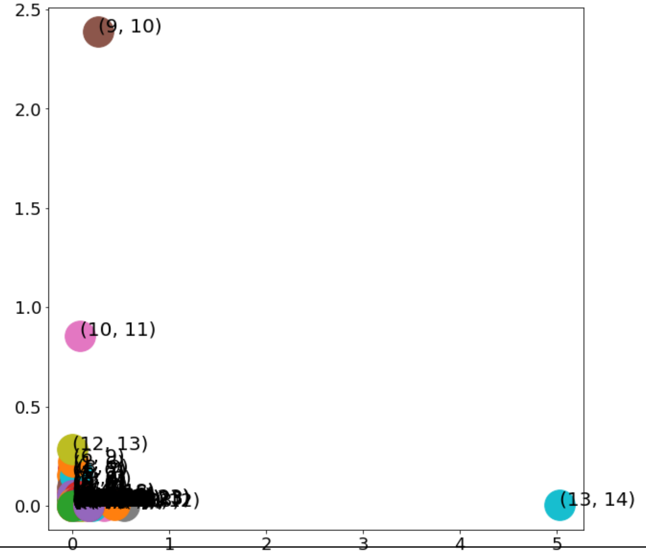
HadamardEmbedder



WeightedL1Embedder

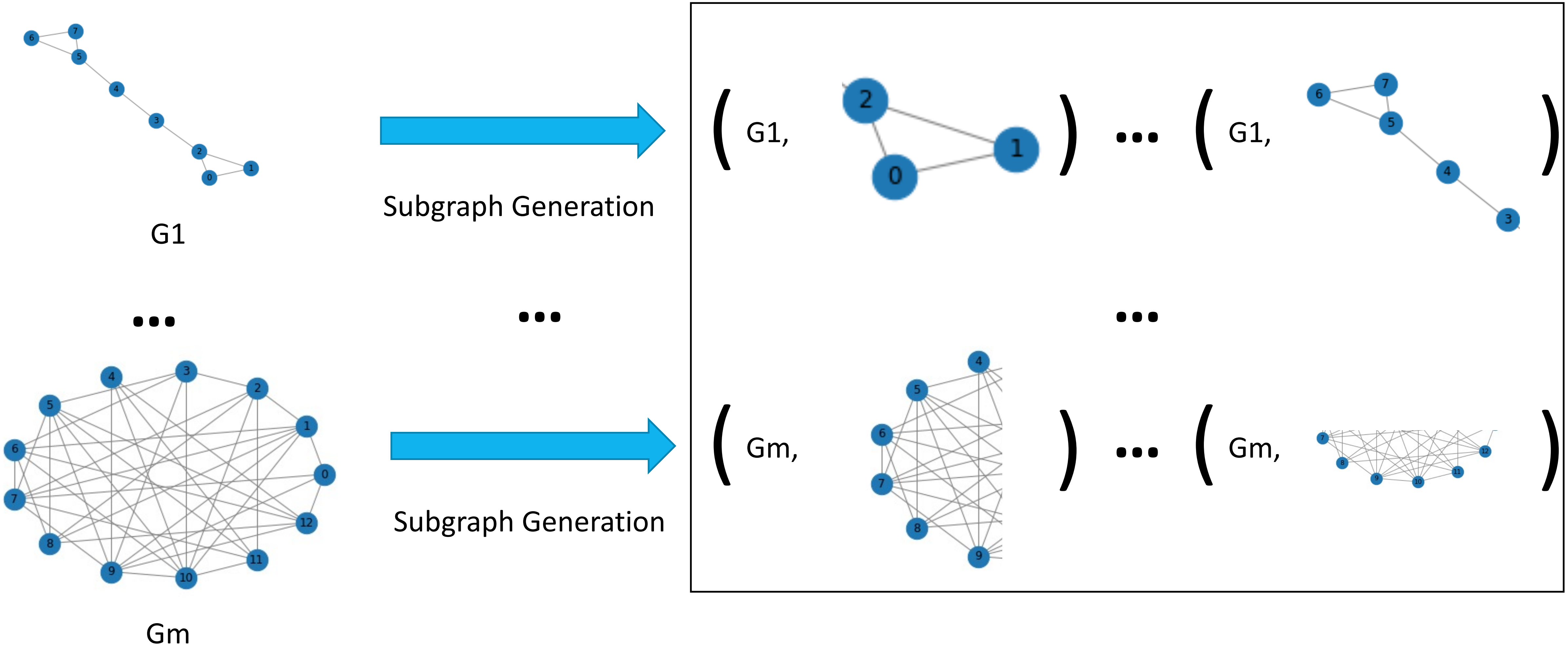


WeightedL2Embedder

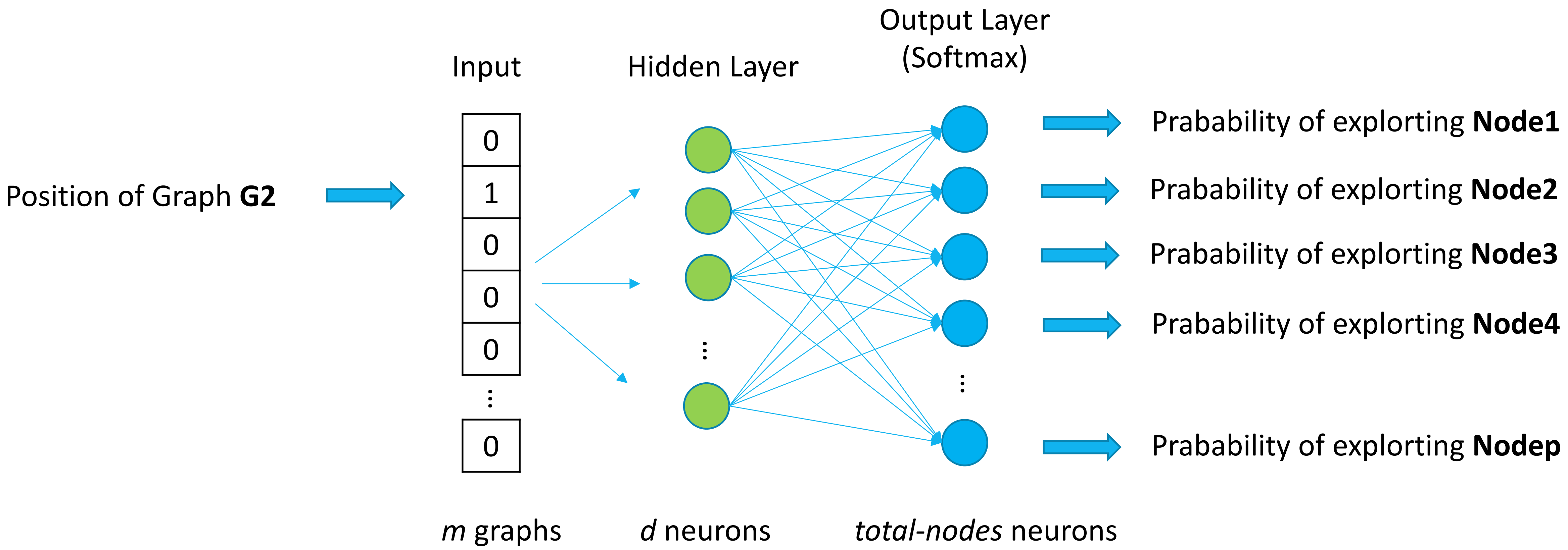


Skip-gram based embedding algorithms – Graph2Vec

Input Graphs **Training Set**



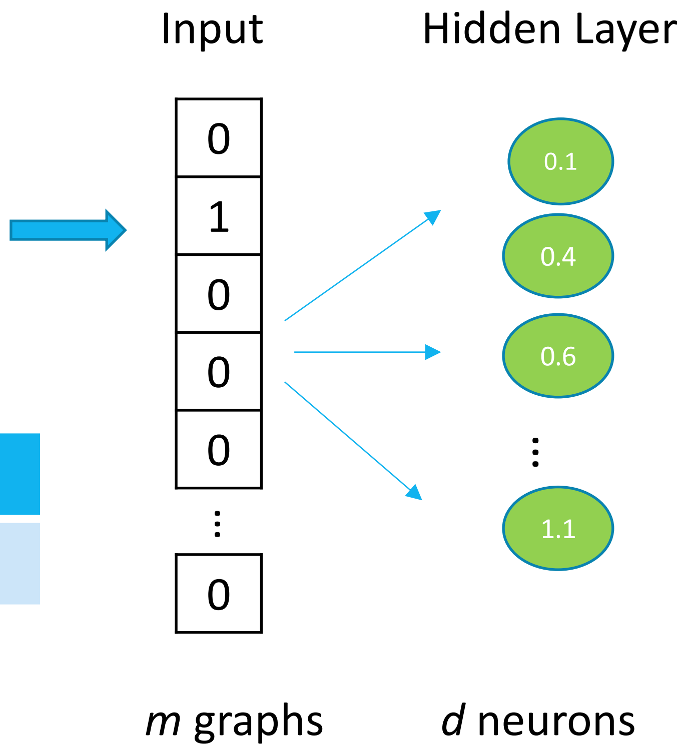
Skip-gram based embedding algorithms – Graph2Vec



Skip-gram based embedding algorithms – Graph2Vec

Step 3. Embedding Extraction

Position of Graph **2**



GraphId	Dim 1	Dim 2	Dim 3	…	Dim d
Graph2	0.1	0.4	0.6		1.1

```
from karateclub import Graph2Vec
Gs = [G1, G2, G….]
model = Graph2Vec(dimensions=2)
model.fit(Gs)
embeddings = model.get_embedding()
```

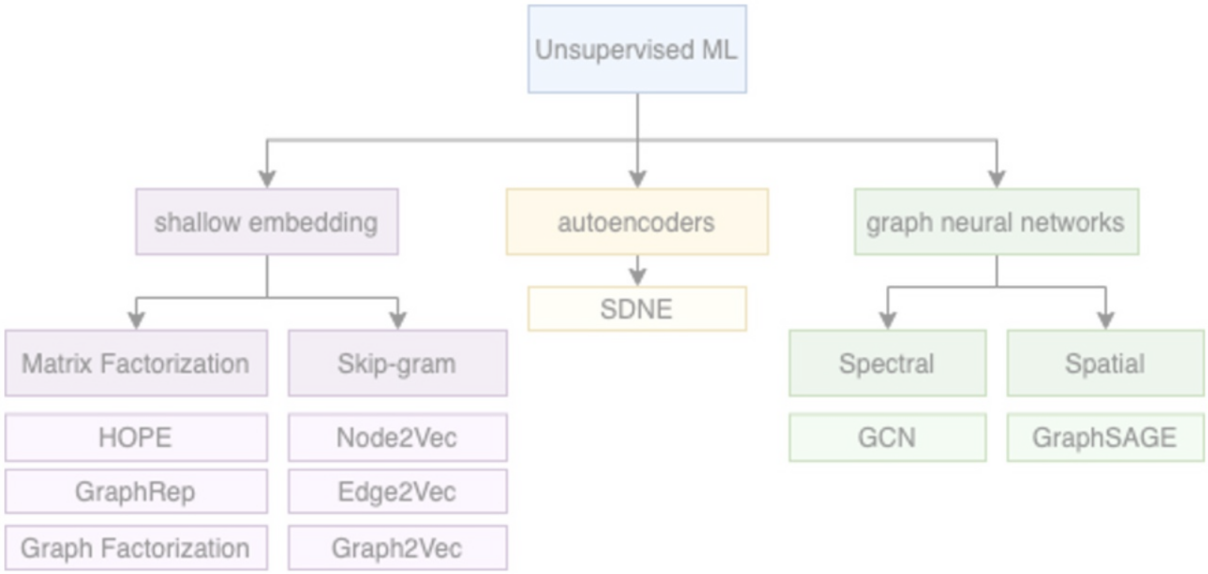
CGnal

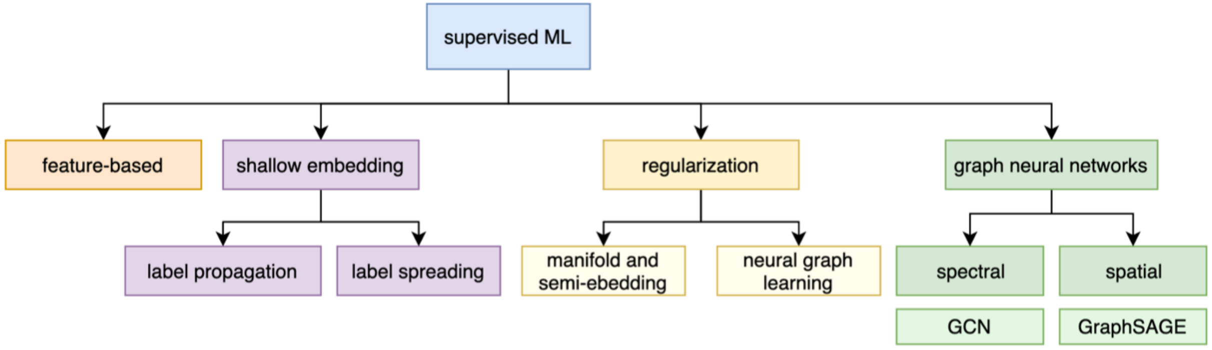
Excercise

Overview on Machine Learning with Graphs

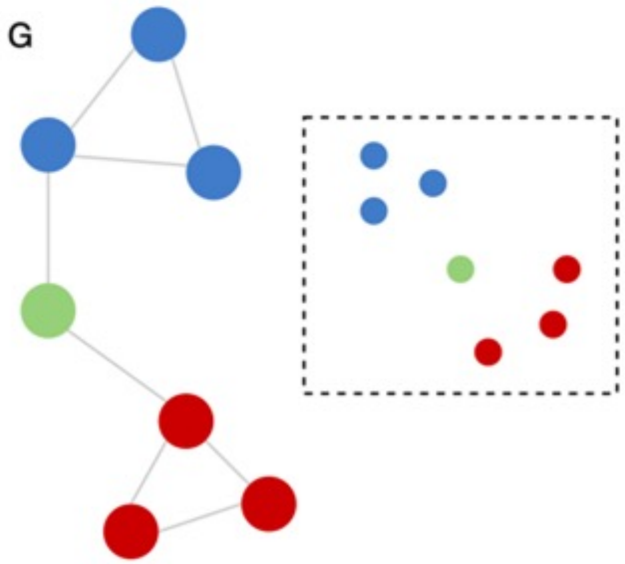
Machine Learning on Graphs: A Model and Comprehensive Taxonomy

Ines Chami[*†], Sami Abu-El-Haija[‡], Bryan Perozzi[††], Christopher Ré[‡‡], and Kevin Murphy[††]

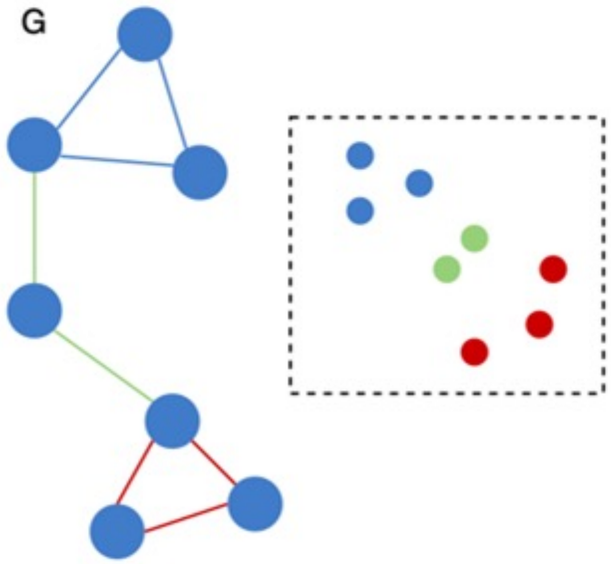




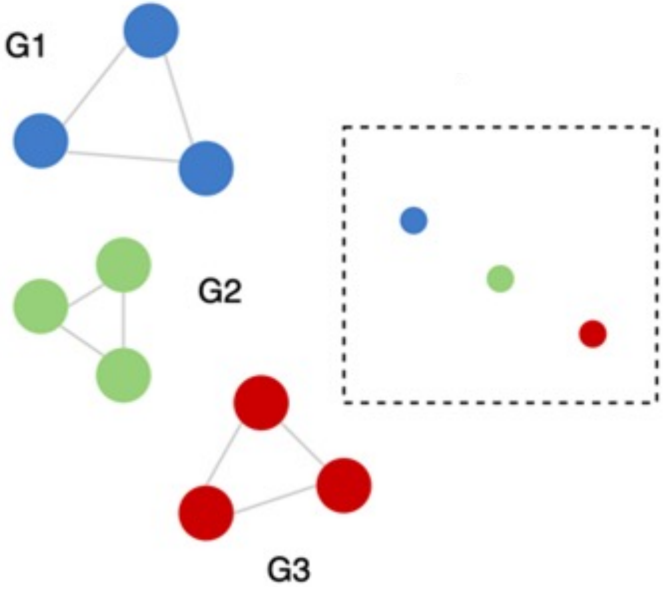
Supervised Graph Machine Learning at different scale



<u>Node Level Classification</u>



<u>Edge Level Classification</u>

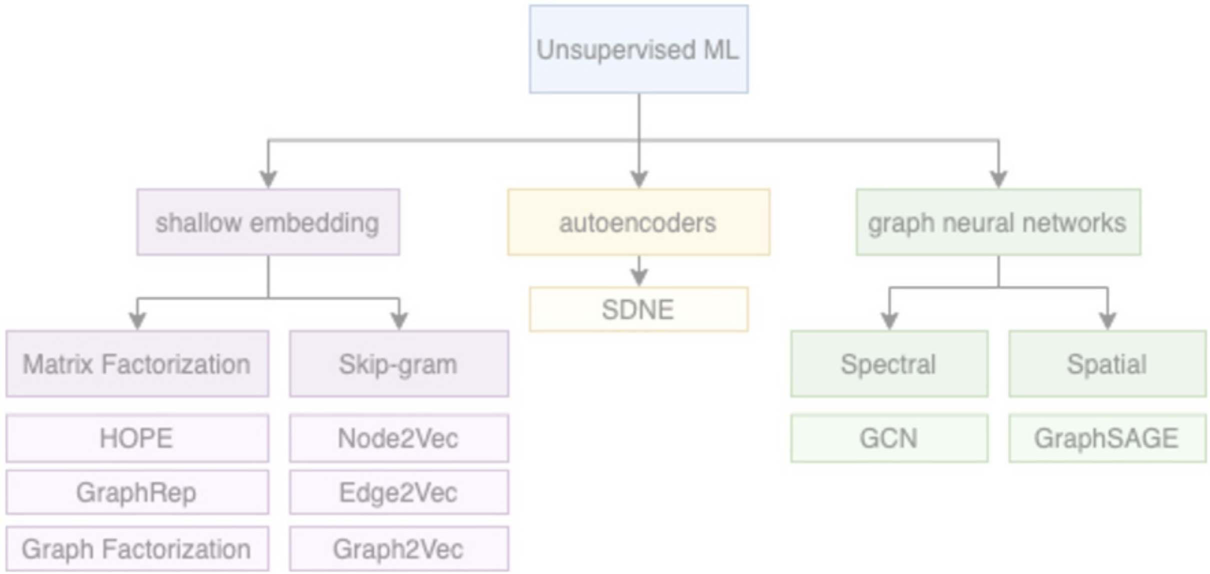


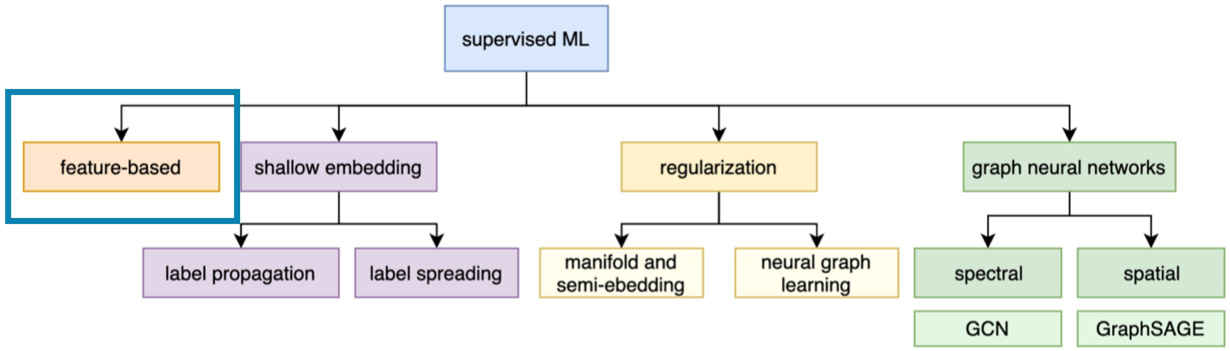
<u>Graph Level Classification</u>

Overview on Machine Learning with Graphs

Machine Learning on Graphs: A Model and Comprehensive Taxonomy

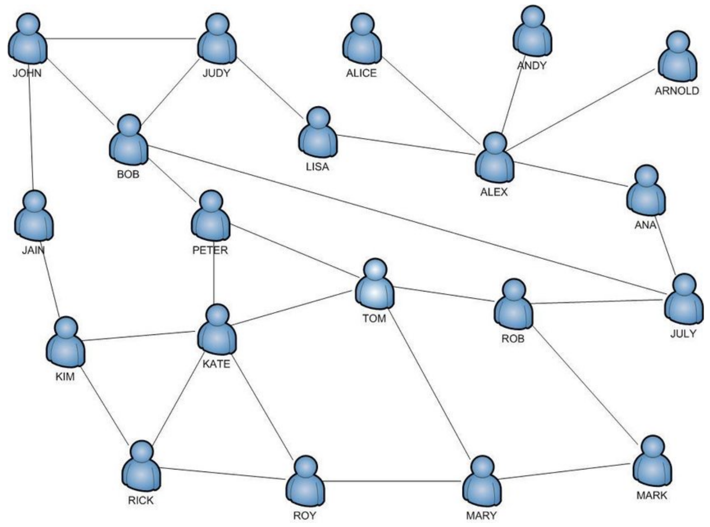
Ines Chami[*†], Sami Abu-El-Haija[‡], Bryan Perozzi[††], Christopher Ré[‡‡], and Kevin Murphy[††]





A Straightforward Approach to apply Machine Learning on Graph

- In «classical» machine learning applications, a common way to process the input data is to build a set of features
- For a **node classification** task, we compute a set of metrics or known features to generate the feature vector



Feature engineering

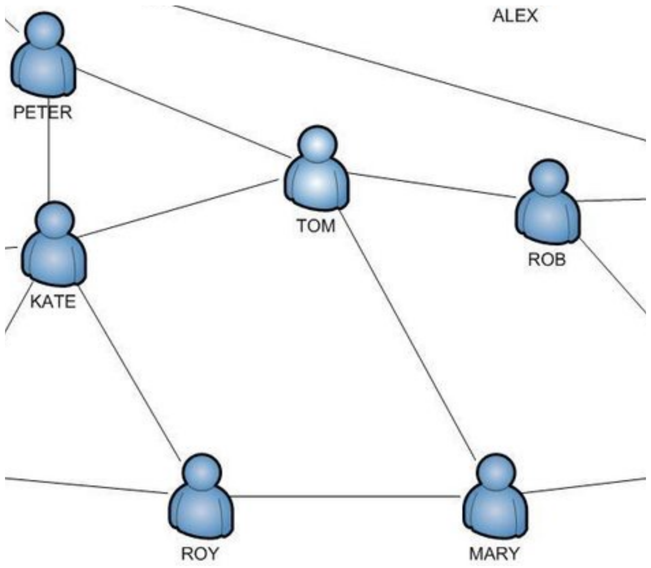
NodeId	Node Degree	…	Age
Alice	0.9	0.12	28
..	0.20	0.88	…
Bob	0.25	0.76	31

1. **Node Attributes**: We know that the nodes in a graph represent entities and these entities have their own characteristic attributes (Age, Sex, Gender, Average Consumption, etc..).
2. **Local Structural Features:** Node features like degree (count of adjacent nodes), mean of degrees of neighbor nodes, number of triangles a node forms with other nodes, etc.

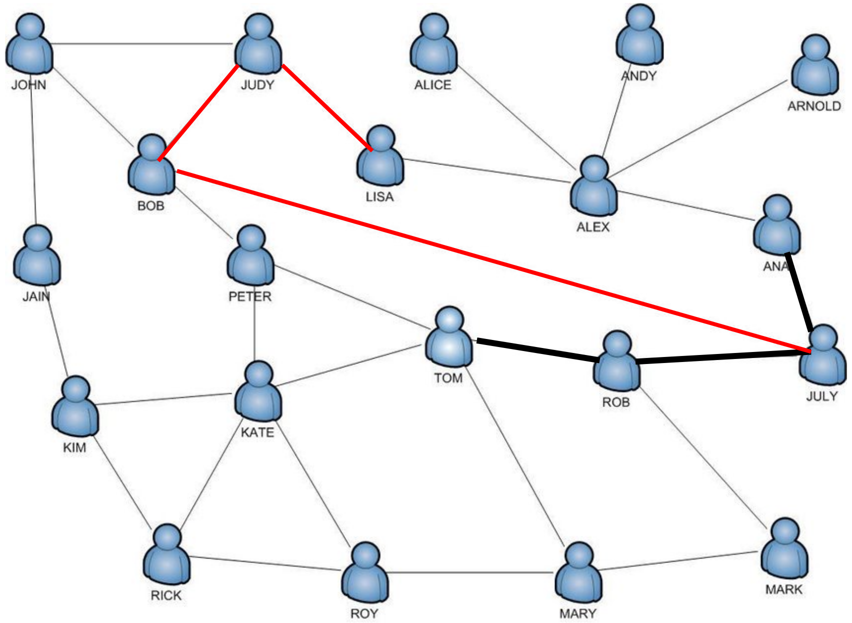
A Straightforward approach to apply Machine Learning on Graph - Limitation

1. **Node Attributes**: We know that the nodes in a graph represent entities and these entities have their own characteristic attributes (Age).
2. **Local Structural Features:** Node features like degree (count of adjacent nodes), mean of degrees of neighbor nodes, number of triangles a node forms with other nodes, etc.
3. **Embeddings:** The above-discussed features carry only node related information. They do not capture the information about the **context** of a node (the surrounding nodes).

Local Structure Features



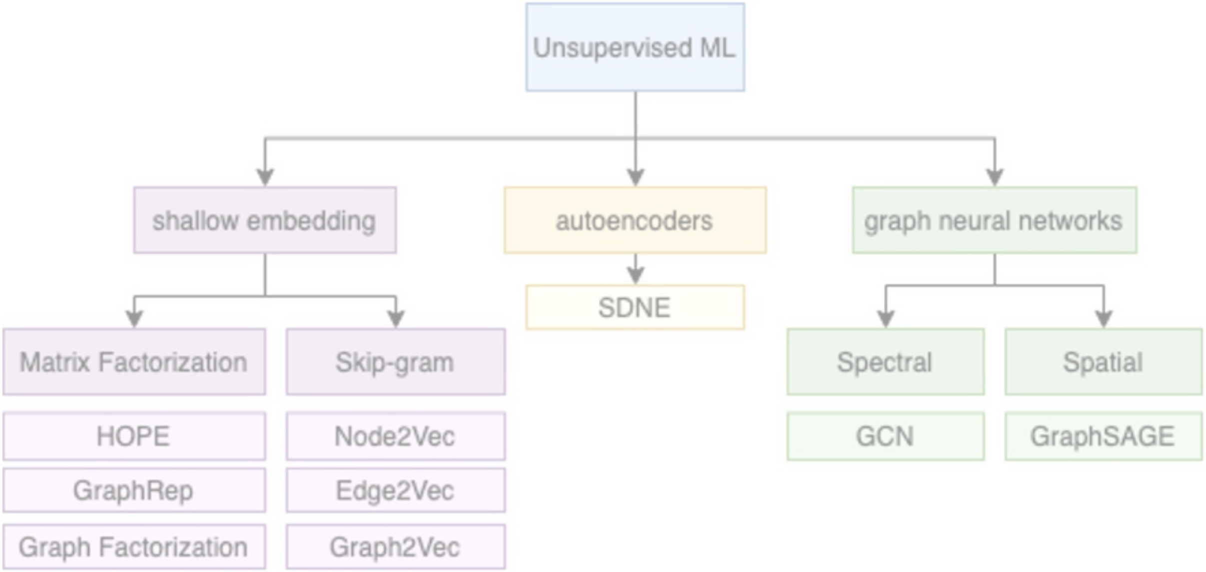
Embeddings

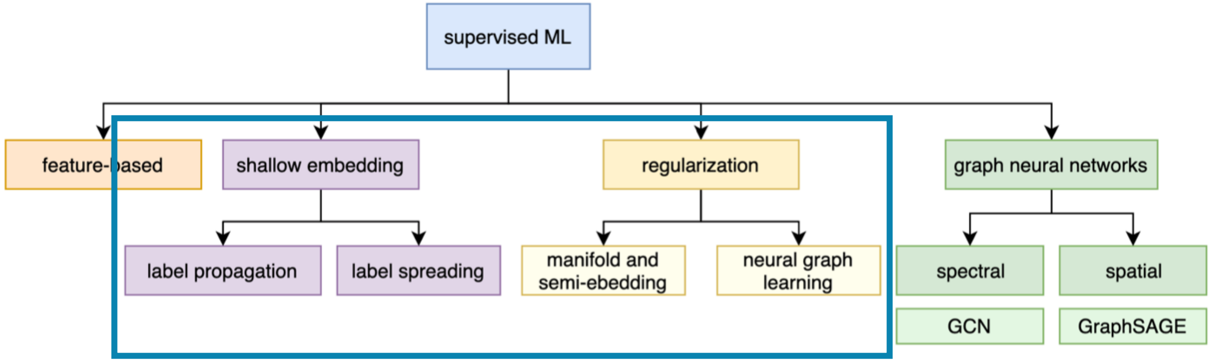


Overview on Machine Learning with Graphs

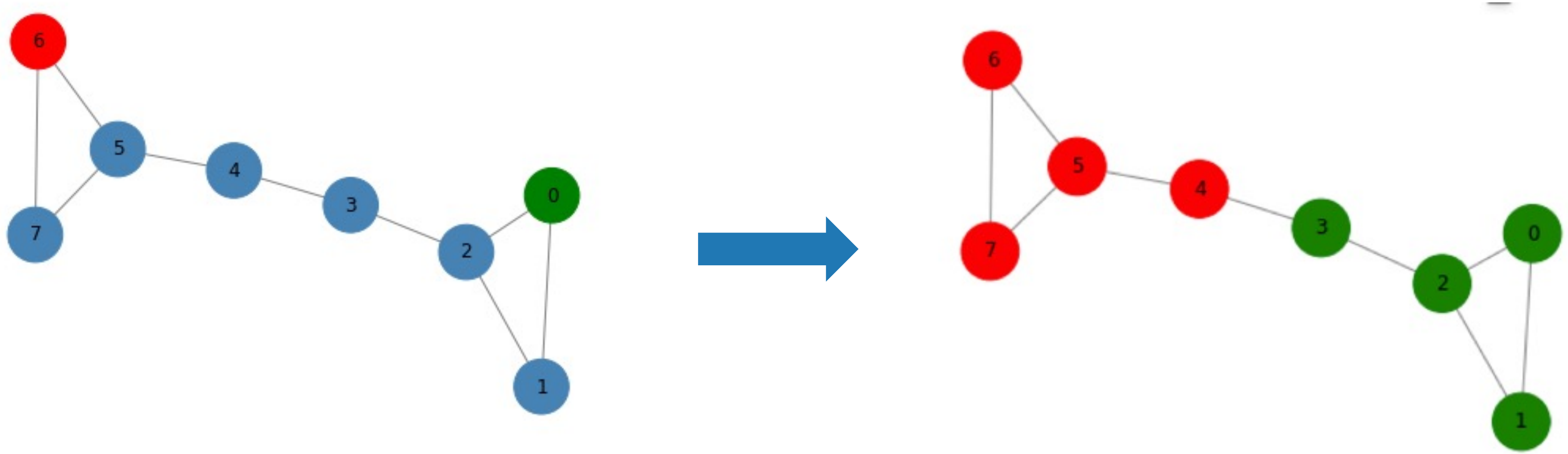
Machine Learning on Graphs: A Model and Comprehensive Taxonomy

Ines Chami[*†], Sami Abu-El-Haija[‡], Bryan Perozzi[††], Christopher Ré[‡‡], and Kevin Murphy[††]





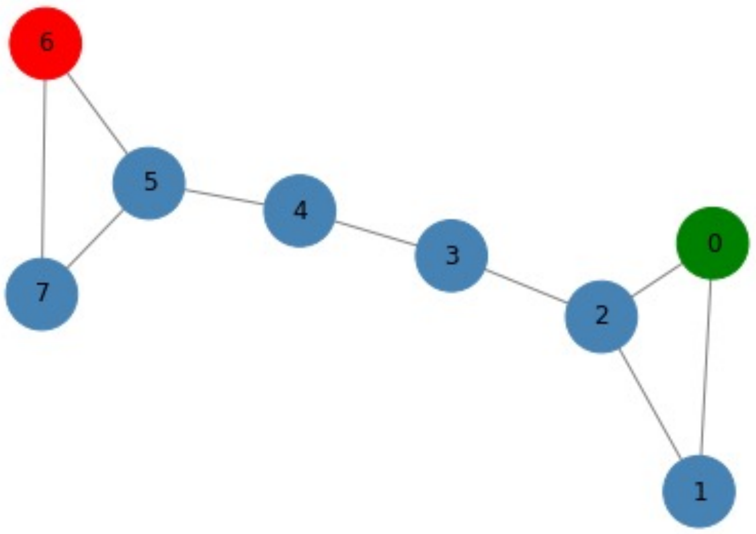
Label Propagation and Label Spreading



- Unlabelled node
- Label 1
- Label 2

Intuition: The closer we are to nodes of a certain label, the higher the probability that the node belongs to that label

Label Propagation and Label Spreading



- Unlabelled node
- Label 1
- Label 2

Transition matrix: Probability of ending up in node j, given we start from node i.

$$L = D^{-1}A$$

$$L = \begin{bmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0.33 & 0.33 & 0 & 0.33 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.33 & 0 & 0.33 & 0.33 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 \\ 0.5 & 0 \\ 0.33 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0.33 \\ 0 & 0 \\ 0 & 0.5 \end{bmatrix} \quad Y^1 = LY^0 \quad \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

Probability of assignment

Initial condition assignment

Label Propagation and Label Spreading

$$\begin{bmatrix} 0 & 0 \\ 0.5 & 0 \\ 0.33 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0.33 \\ 0 & 0 \\ 0 & 0.5 \end{bmatrix} \quad Y^1 = LY^0 \quad \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

Probability of assignment

Initial condition assignment

Two problems:

- Some nodes are not yet inferred (node 3 and 4)
- The known label have been reset to 0

  Solution: resetting the known values to the intial condition.

$$Y^1 = \begin{matrix} 1 \\ 0.55 \\ 0.33 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

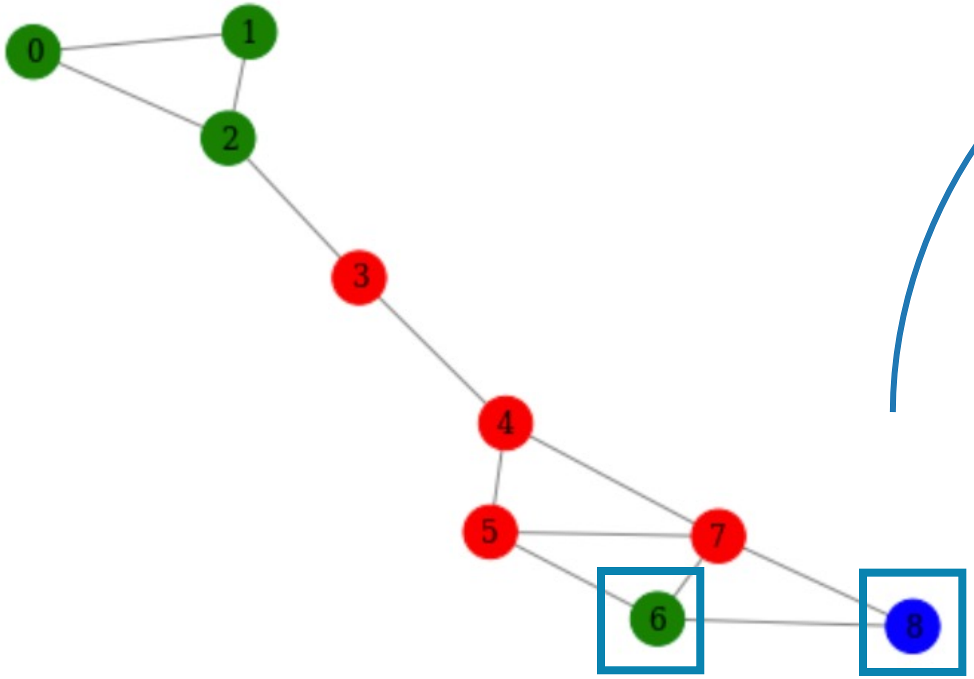
Solution: Label propagation

$Y^{n+1} = LY^{n}$ and force resetting known values

The algorithm performs the computation until some stop conditions are satisfied:

- **Number of iterations**: computation stops when a given number of iterations has been performed.
- **Solution tolerance error**: computation stops when the absolute difference of the solution obtained in two consecutive iteration y^{t-1} and y^t is lower than a given threshold value.

Label Propagation and Label Spreading



Node labelling may be affected by noise or mistakes, or just anomalies that may influence classification

This node has 50-50 chances of being of either labels

Weaker Node Assignment

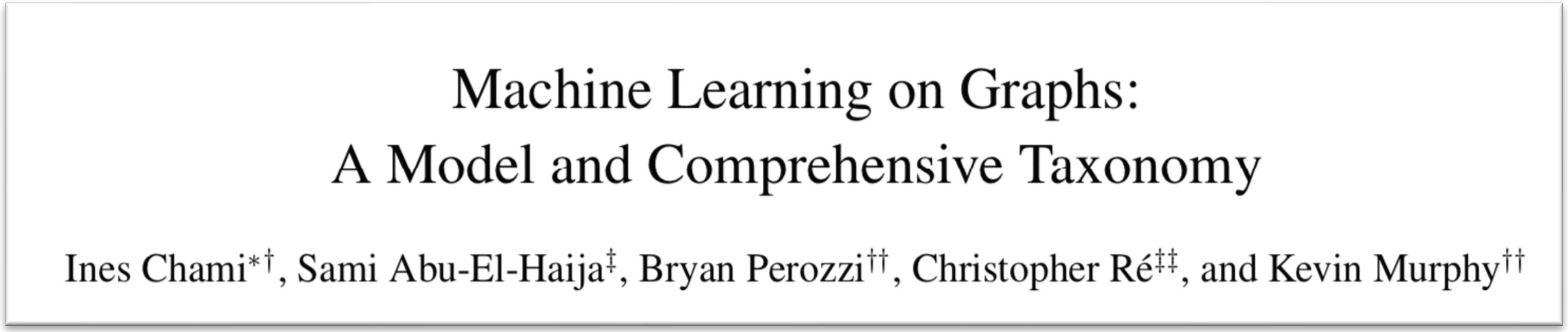
$$Y^t = \alpha \mathcal{L} Y^{t-1} + (1-\alpha) Y^0 \quad \text{with } \mathcal{L} = D^{-1/2} A D^{-1/2}$$

We therefore mix two contribution:

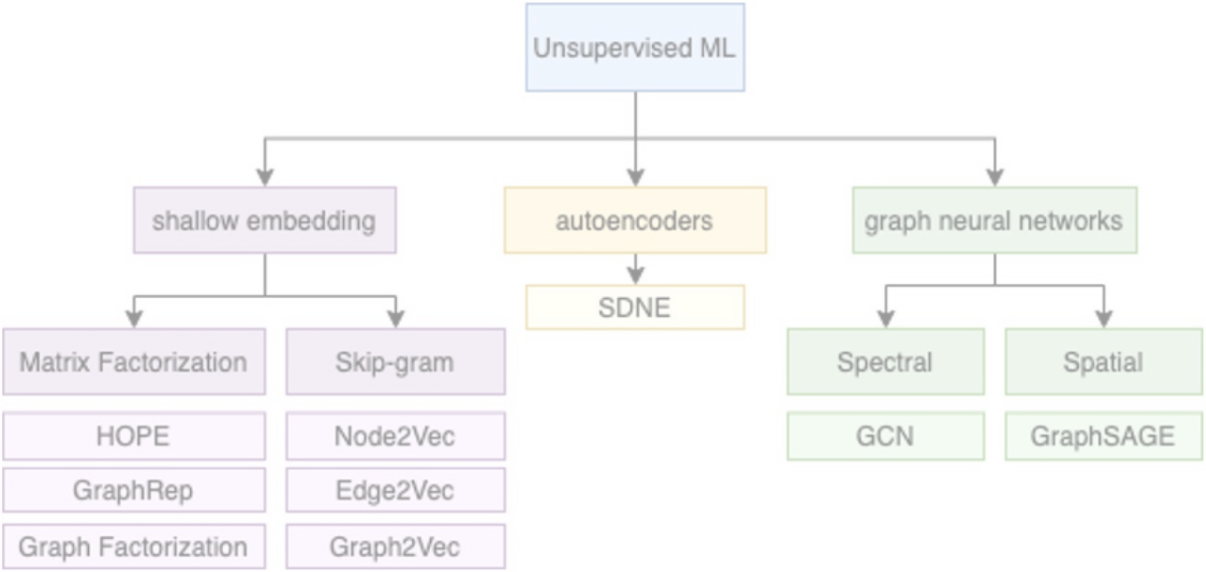
- one that comes from propagating label information, assuming **smoothness** of the labelling
- one that comes from label **assignment** based on "training"

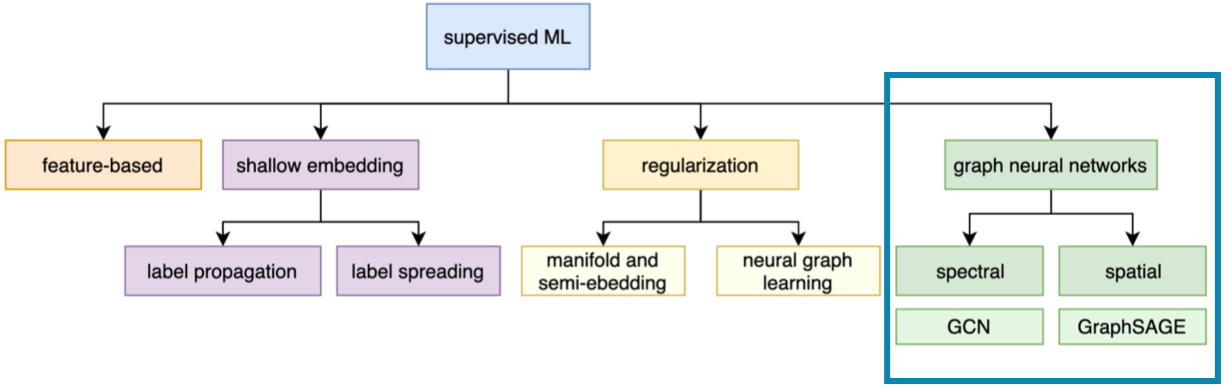
Graph regularization over labelling

Overview on Machine Learning with Graphs

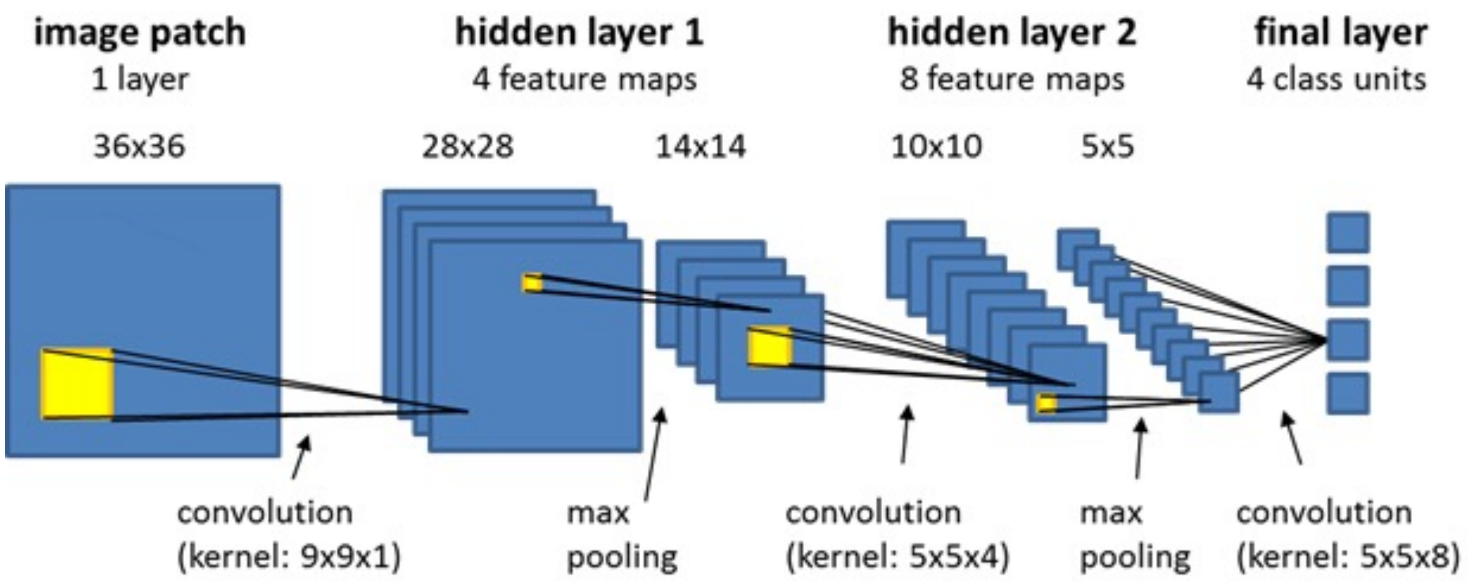

Machine Learning on Graphs:
A Model and Comprehensive Taxonomy

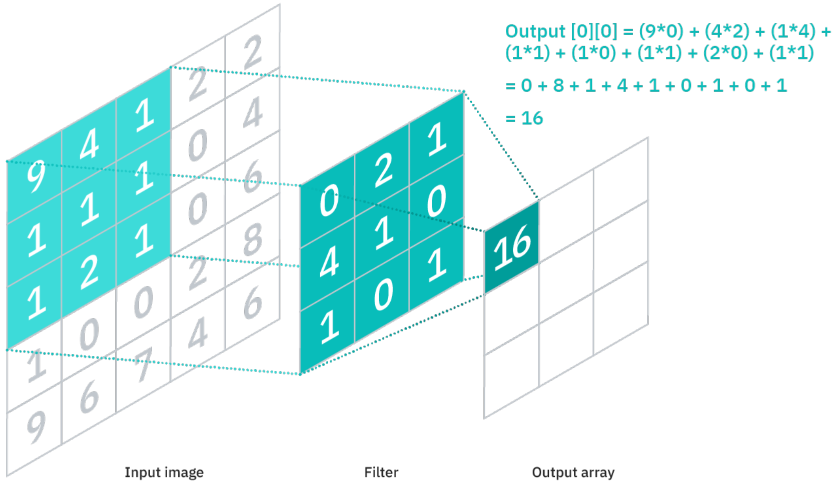
Ines Chami[*†], Sami Abu-El-Haija[‡], Bryan Perozzi[††], Christopher Ré[‡‡], and Kevin Murphy[††]

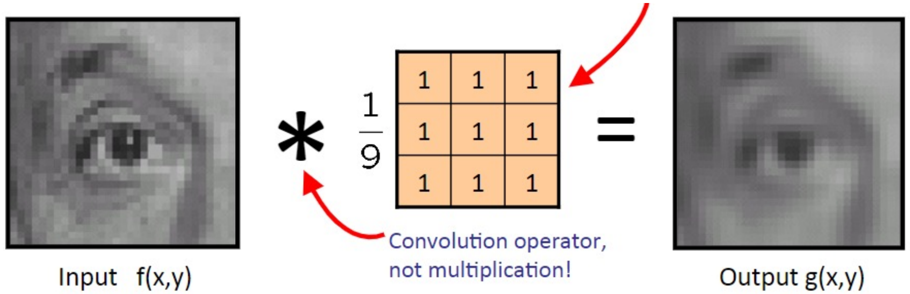


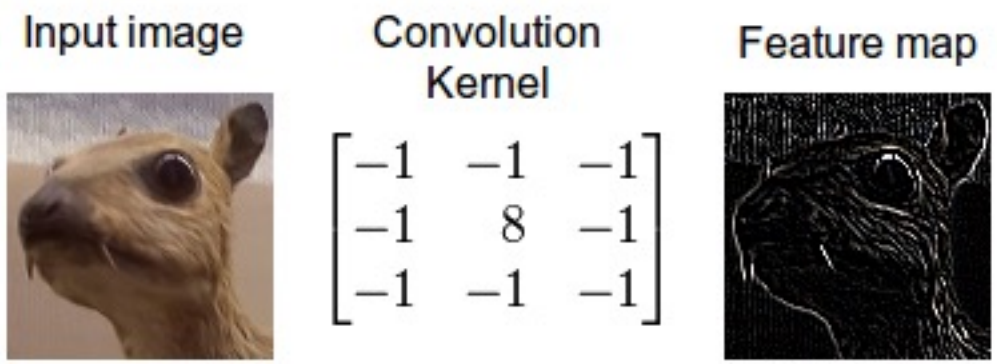


The Convolutional Kernel





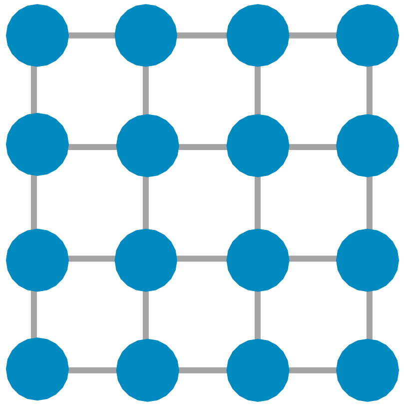




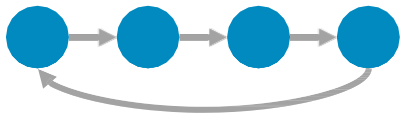
Data Structures as Graphs

Regular Data Structures

Images

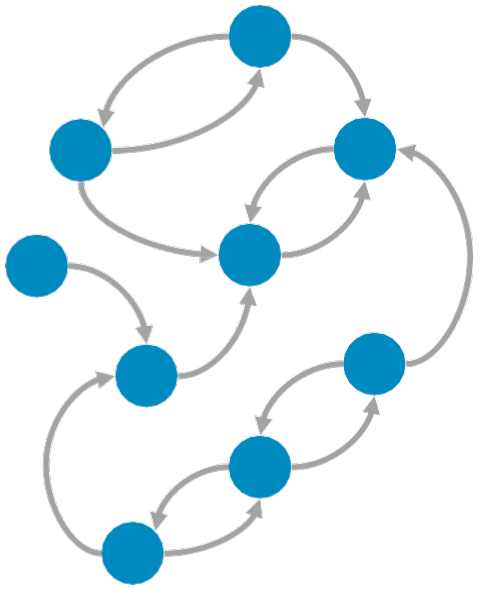


Time Series

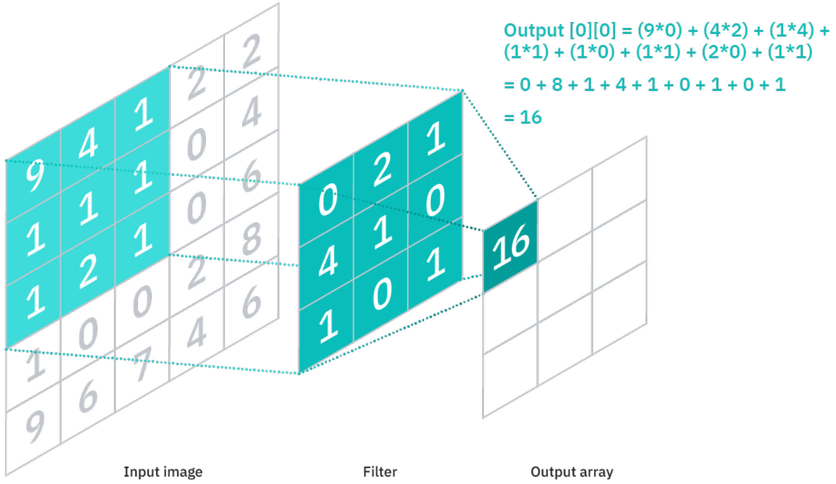


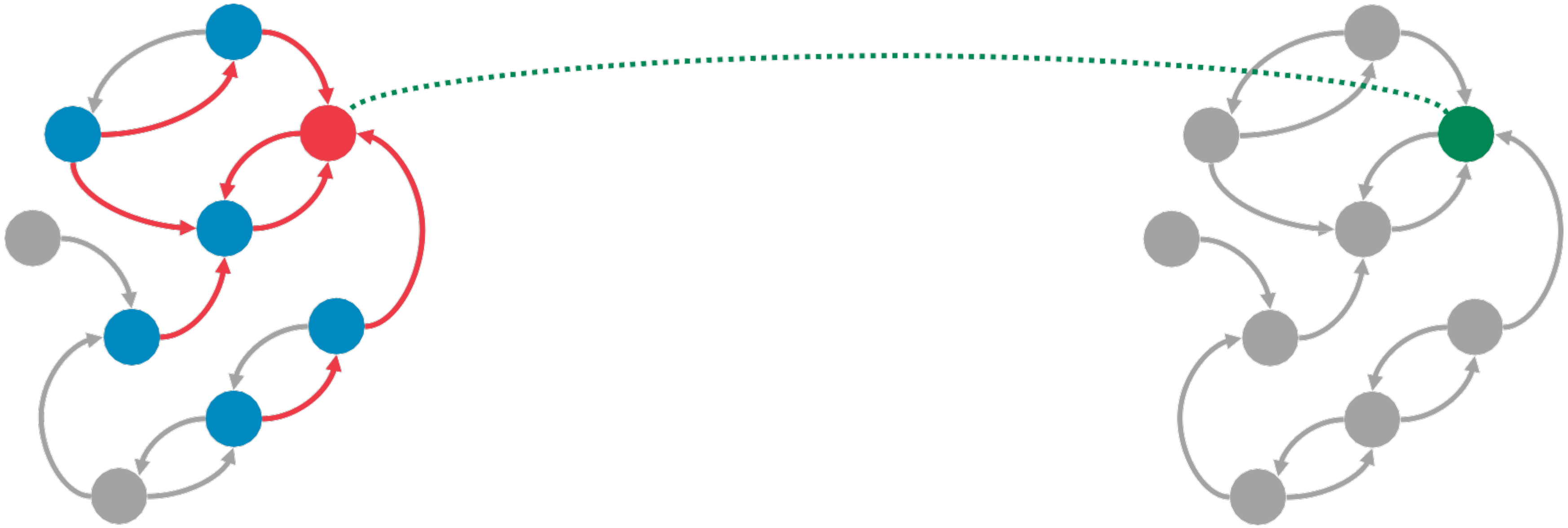
Irregular Data Structures

Social Networks
World Wide Web
Telecom Networks
Supply Chains
Biological Systems
Semantic Lexicons
Chemical Models
State Machines
Call Graphs
...



The Convolutional Kernel





Graph Neural Network in practice

PyTorch geometric

STELLAR GRAPH

NetworkX
Network Analysis in Python

google / gcnn-survey-paper